COMMON
Meeting Proceedings
Philadelphia, Benjamin Franklin Hotel
September 9-11, 1968

TABLE OF CONTENTS

TABLE OF CONTENTS/cont.

Philadelphia COMMON Meeting

September 9, 1968
September 10, 1968

Benjamin Franklin Hotel
Philadelphia, Pa.

MONDAY, SEPTEMBER 9, 1968, 8:30 A.M.

JAMES C. STANSBURY, President, presiding.

PRESIDENT STANSBURY: Welcome to COMMON and Philadelphia. I suspect that among the older members here at any rate the big question is the result of the elections. I am not going to introduce the officers until later, but I'll tell you the results now.

The new President is Paul Bickford and the Secretary-Treasurer is Chuck Maudlin. Chuck enjoyed the unique position of being unopposed, so we knew that one.

The new Board is: Dave Dunsmore, Frank Maskiell, Norm Goldman, Brian Swain, Wade Norton and Hugh Kerr.

I'll introduce them later; they're not all here yet.

We've tried to give you a good meeting this time. I received compliments on the agenda which I don't deserve; that's Joe Aicher, Mike Schilder, the division managers.

I've also received complaints. I won't say I don't deserve those. We recognize that we can't please everybody. All we can do is try.

We have a rather good turnout this time. I am told the registration is in excess of 600 already, and from the looks of the room I can well believe it.

This is not a keynote address, as you know. It's sort of an introduction to COMMON, to the meeting. We try to tell you any specific problems we might have. There is no planned speech or anything of that sort. As a result of that, I'll take this opportunity, as usual, to ask whether anybody has questions, suggestions, requests, things they want to bring up before the open Board tomorrow night and would like to have people think about before then -- anything of that sort.

(No response)

I don't believe it!

We'll have changes in the agenda posted in the coffee break area, registration area, as they occur. There will be changes. We'll try to keep that up to date. We'll try to announce them during the morning and afternoon coffee breaks, but we'll post them so that people can see them as well.

Mike, any last minute announcements you want to make?

MR. SCHILDER: No, I don't think so.

PRESIDENT STANSBURY: What were all those sessions last night? What were all of the planning sessions doing last night?

MR. SCHILDER: They didn't talk to me.

PRESIDENT STANSBURY: If you do have changes -- section chairmen, anybody -- please see that Mike or Joe Aicher gets them.

MR. SCHILDER: At the coffee breaks we will announce any changes. At the present time there are no changes in sessions, but that's because we haven't met yet. I expect to see changes.

The latest wording on the 1130 group is General and Technical, rather than Commercial and Non-Commercial. At the moment I don't know which way the break is. Whether you'll be in the Crystal Ballroom or the Valley Forge Room depends on which is the bigger group; I'm sure Don Kiel will take care of letting you know about that.

That's all I have now.

PRESIDENT STANSBURY: For once we have a meeting that seems to be well enough organized that there is nothing to say. But right now I am going to introduce four people:

Don Kiel, the 1130 Project Chairman. For

you new members, if you're 1130s Don will be chairing the first session probably that you attend. In any case, he can inform you about 1130s.

Dick Gabriel, of the 1620 Project.

Al Ragsdale, 360 Project Chairman.

Bob Fostrom, 1800 Project Chairman.

Bill Hill, Model 44 Project Chairman.

These are the people you should be looking for if you need information about specific machine oriented activities.

What about our 360-20? Is there any contact here?

(Everett Sylvester)

Gentlemen and ladies, we do need some response from you -- questions, comments, suggestions. I can't believe that no one has any.

PARTICIPANT: Who are you, Mr. Chairman? You introduced everyone else and we don't know who you are.

PRESIDENT STANSBURY: I'm Jim Stansbury. Officially I suppose I am not the Past President of COMMON. Somehow or other I never get around to introducing myself.

PARTICIPANT: Jim, there is one thing you could consider to be brought up at the general Board meeting,

and that is the question of whether or not we are going to change the organization to allow other user groups to join us.

PRESIDENT STANSBURY: I'll repeat the suggestion. COMMON at present has a rule that we cannot have chapters per se. The question was what we would do about local user groups who wish to become members of COMMON. The existing policy is that they may become members as individual installations, but not as a group. We welcome people from such groups. We'll be glad to send them copies of CAST. We'll be glad to publish their news in CAST.

Let me give you a specific example and it will, I think, make it clear. There is a group called the HOUSTON 1130 USERS GROUP. Under the bylaws as they now stand this group cannot become a member of COMMON as a group. They have to become members as individuals. The reason for this is that meeting attendance requirement of ours. We can't have one member from a 35-man 1130 user group, a local man, who represents the user group rather than the installation and qualifies everybody for membership in COMMON on that basis. That's the problem basically.

Again, we welcome them as individuals as we do the members of special user groups, but so far we have

not considered incorporating them as groups.

The suggestion from the floor was simply that we ought to think about this policy, see whether there should be some changes in bylaws to permit it, under what circumstances, and that sort of thing.

This one I think you would need more than two days to think about, but if you have any suggestions bring them up at the open Board meeting which will be tomorrow afternoon at five-thirty.

MRS. LAURA B. AUSTIN (Mid-West Region): I would like to make announcements of coming meetings. This information is in your COMMON reference manual. It's under Section 2, COMMON organization. When you get back to your installation and want to look ahead for future meetings, we try to keep this updated for future meetings; but I will announce the next two here so that you can make notes of them.

The next meeting will be in Houston, Texas, December 9-11 at the Rice Hotel.

The following meeting will be April 14-16, 1969, in Los Angeles at, we think, the Ambassador Hotel (though there may be a change in the hotel).

In the COMMON reference manual is the list

of meetings through 1970, so when you're planning budgets ahead take this into consideration when you're looking at attending COMMON meetings.

The one after that will be September 17-19 in Pittsburgh.

I wanted to announce these future meetings because Wednesday at the general session of the Administrative Division we would like to have some people come forward as volunteers for program chairmen, local arrangements chairmen for these coming meetings. Houston is taken care of, but we do need to start now to plan for the Los Angeles meeting and for the Pittsburgh meeting; so we would like to know if you are available to help work in COMMON to make a successful meeting in those two cities. We would like to have your names and some indication that you could help us and how you could help us at the Wednesday session of the Administrative Division.

PRESIDENT STANSBURY: Since we don't seem to have very much else to do I will introduce the new members of the Executive Board and then our local Arrangements Chairman, Joe Aicher, who has a few announcements he wishes to make.

On my right here is Paul Bickford, the new

President of COMMON.

The gentleman on my left is Chuck Maudlin, Secretary-Treasurer. Since he is a prominent target and is not only our new Secretary-Treasurer but our old one, you can talk to him with a clear conscience. He's not only responsible for what was, but what will be.

The gentleman on his left is Dave Dunsmore, member of the Executive Board.

The next one is Brian Swain, member of the Executive Board.

On my right is Mrs. Laura Austin, who is now the Vice President of COMMON but who decided she could not run again this time. This is her last meeting.

Next, Frank Maskiell, a member of the Executive Board.

Next, Hugh Kerr, a member of the Executive Board.

There are two other people who are not present: Wade Norton and Norm Goldman. Most of you know both of them, I think. They've both been fairly active. We'll introduce them at the open Board meeting tomorrow night.

I should say that this meeting is officially

the responsibility of myself and Chuck, the President and the Secretary-Treasurer. If you have questions or comments or suggestions about any meeting after this, then Paul is your man. But comments about this meeting should come to me.

Joe Aicher, who is our local Arrangements Chairman, has some announcements he wishes to make and I want to introduce you to him and thank him for a job well done.

MR. AICHER: Thank you, Jim.

I guess this is the first time and the only time I'll be able to say this, but as of now I know of no program changes. From here on in I'm sure I'll be talking about more of them.

I would like to call your attention to a few things that are in the program just as a reminder. Perhaps Mike has even mentioned them already -- I wasn't here -- and if he has, the repetition won't be too bad anyway.

Demonstrations of IBM equipment will be available at the IBM Data Center, which is at 18th and Kennedy Boulevard, some twelve blocks from here. We will have both on Monday and Tuesday time blocked out at this Data Center on a variety of equipment: 1130, Mod 20, 30,

40. From 2:00 to 4:00 Monday and Tuesday there will be time for some demonstrations, but there is the possibility of your utilizing these machines.

To transport you from here to there we will have at 1:45 a normal Philadelphia Transportation Company bus (which will probably say "chartered") leaving the Sansom Street exit of the hotel. There will be a sign down there indicating exactly where this is. It's on the ground floor in the rear. The traffic pattern is such that they must load and unload in the rear.

There will be on the board a sheet for anyone wishing to sign up, but even if you do not sign up you can get this bus at 1:45. It will be shuttling back and forth, and there will be chances to get on at the JFK Data Center, also at the IBM Building at 1700 Market Street, and, of course, at the Ben Franklin Hotel. So you can make a round trip. The hours, again, are 2:00 to 4:00.

One other thing I wanted to talk about was this late registration. The hotel is jammed with conventions and we have permission to hold seventy-five late registrations; this means Wednesday at six o'clock. We'll put a sheet on the board and the first seventy-five names on it will be entitled to stay 'til 6:00 on Wednesday. At

the coffee break this sheet will probably be available. If there are more than seventy-five all I can suggest is that we double up or you check out and leave your luggage in care of the Bell Captain.

These birds-of-a-feather sessions will have sheets on the board. We'd like you to fill out the subject. There will be space for other interested parties to sign their names. We'd also like you to put down the day and the time you'd like these to occur. They'll most likely be in the evening. We'll assign the rooms, and you can check back to see what room it is.

That's all I have.

PRESIDENT STANSBURY: There are two announcements I thought of while Joe was speaking. The procedure of getting programs from the 1620 Program Library is to be changed to conform to the procedure used with the other machines. I would suggest that most of the 1620 installations go to the Program Information Department presentation on Wednesday. I think personally that the change is to your advantage. It makes it simpler to get programs, not harder.

Last night there was some sales information about a piece of equipment to be used with an 1130 placed

in the registration area. I had it taken down. The by-laws specifically prohibit advertising in the public area for anything other than IBM equipment. We have absolutely no objection to any of you who represent vendors talking to anybody you want to. That's your privilege. We do ask that you not post notices or not put material on display. I don't know who the installation was. I deliberately have not checked. If they want to discuss it with me or with Paul they're welcome to do so.

Any other questions?

MR. BRIAN SWAIN: I would like to ask a question of Chuck Maudlin. I am assuming that there are a good number of people who are here probably for the first time and, therefore, I wonder if we could spend a minute or two talking about communications. We had this brought up to us last night, that an important facet of the work of COMMON is communications between members.

In my experience it took me some time to find out what was our method of communication. Our publication called CAST was not drawn to my attention very early and it took me a long time, therefore, to find out how valuable it is.

I wonder if it would be of general use if

Chuck would spend a minute or two talking about what is CAST, how often it comes out, how you get something published in CAST if necessary. Would this be of general interest?

MR. MAUDLIN: I have something for you first. The Secretary-Treasurer has a new address and telephone number; that is, on G5 in your book the address is incorrect. Instead of Texas Woman's University it's Texas Christian University, and instead of Denton, Texas it's Fort Worth, Texas; and instead of 387-1322 it's WA6-2461. Zip code is 76129. The user code number if 5130, I think; I'm not real sure.

CAST is a publication that comes out nine times a year. It goes to the printer on the first day of the month every month during which there is not a meeting. The letters supposedly stand for "announcements from the Secretary-Treasurer." It's announcements from anybody to the membership. It consists of correspondence between the Board or members of the organization and IBM, their replies; in some cases announcements by IBM. It contains cross-talk between members where it is of general interest to the group. Presumably I have some decision to make each time I receive something that is for CAST: Does it go in or not?

That's the only decision. So far everything that has come to me has gone into CAST, to the best of my knowledge, if such a request was made.

It gets bigger and bigger and bigger. There have been issues of CAST as small as six pages and there have been larger ones; the last one was 92 pages. I think the last one was a very good one.

I don't know what else to say except that it is a very valuable tool and it's something that you should look at very carefully and something you should contribute to.

MR. DON KIEL: That special information on the 1130 went out, as far as I know, to all CAST recipients. If you have no need for that information we in the 1130 project would like to get that back, so we can have extra copies.

PRESIDENT STANSBURY: Don is referring to that 8-page appendix that came out the last time.

PARTICIPANT: I wonder if we could continue to have the notebook binder holes punched in CAST.

MR. MAUDLIN: As long as I'm having them printed they'll have the three holes.

PARTICIPANT: It might be helpful if the

entries in CAST could be categorized, at least with a key word or subject content on each page, or perhaps something a little fancier, like an index; so then one could go through these ninety pages without having to read a lot about somebody else's system.

MR. MAUDLIN: To the best of my ability that's Page 1, I thought. If you can tell me how to improve Page 1 I'll be glad to listen to anything you have to say.

PARTICIPANT: Would it be possible for the various projects to get a listing of the membership by machine? In our planning session last night of the 1130 group we found that we could usefully use a listing of the 1130 installations. It's very difficult to sort out the general user groups. Is that possible?

MR. MAUDLIN: It's not something that is easily arrived at. In the first place, you can't get the answer by going through the manual. If you look at 5130 I think you find a 1500 for our machine. It happens to be a 1500 sitting on an 1800. We also have a 360 Model 20. We have a 1620. We have a stand-alone 1800 on order and we have a 65 on order. You couldn't find all of those by looking at anything in the listing.

It is my intention to very shortly after this meeting send out to the membership the re-registration form that is required in the bylaws, and to incorporate that in some sense or other in CAST by having all machine types listed. But I don't know any way to do it other than to have a page of 1130s and just list user code numbers.

SAME PARTICIPANT: That would probably be good.

MR. MAUDLIN: And a page of 1800s and user code numbers. With the reference manual itself, that would give you a list of everything and we would be in there three or four times.

PARTICIPANT: Perhaps you could have everyone put a card in with their re-registration -- what computers they have; and then just run it on cards.

MR. MAUDLIN: Just the thought of that scares me -- that many source documents. I would rather punch those cards myself. I'll take it from the forms. It shouldn't take very long. It will probably not be in the October 1 CAST because that's when the registration form will go out. The results will probably not be in the November 1 CAST because I doubt that the replies will be back in time to put that information together. In December

there isn't one, so I would expect it would be in January. Beyond that I don't have any idea how to attack the problem, but just that many cards scares me.

PARTICIPANT: Isn't there another kind of newsletter that somebody by the name of Carroll put out? Is that different?

MRS. AUSTIN: This will be discussed at the general session Wednesday, the newsletter situation.

MR. MAUDLIN: That one I've not heard of.

MRS. AUSTIN: That's Carroll Hall.

MR. MAUDLIN: I'm sorry. I was thinking last names.

For a period of time that was incorporated in CAST. When CAST got off the ground the contributions to the newsletter became very minimal, and it died a very natural death. It may be revived, but it's currently not a live item. There are two newsletters that I get, one of which has been going into CAST and the other one may go into CAST.

PRESIDENT STANSBURY: Thank you, Chuck.

I have a request here for a show of hands on the people who are planning to attend the AMPAC presentation. They will try to have Xerox copies of material available

for all those attending. (Show of hands) AMPAC is Automated Material Procurement and Control.

(President Stansbury repeated the following information given by someone from the floor:)

This system was written for a 1620. It has been implemented on a 360 and is widely useful in any area where this material procurement and control is required. The fact that it has been implemented on a 1620 would indicate that it could be implemented on an 1130.

The time for that is 8:30 Tuesday and there are two additional sessions. There is an abstract of the presentation on T20.

Let's have that show of hands again, please. (Show of hands) It looks as if a hundred copies would be adequate. That allows for the fact that people will be changing their minds, too; a 25% safety margin.

That's all I have, gentlemen. I think we can adjourn until the next session.

(The session recessed at 9:20 a.m.)

OPEN BOARD MEETING

TUESDAY, SEPTEMBER 10, 1968, 5:30 P.M.

JAMES C. STANSBURY, President, presiding.

PRESIDENT STANSBURY: The first thing I would like to cover at this open Board meeting, before I ask for questions from the floor, is a report from Paul Bickford on a proposal that was made to COMMON by Share and Guide for providing certain clerical services, in general something corresponding to the paid executive question that came up in Chicago.

After the Chicago meeting Don Madden of the ACM and I talked about it. I went out to their meeting in Chicago with the President of Share and the President of Guide, and we discussed the proposal that the ACM had submitted.

I've asked Paul Bickford, Chuck Maudlin, Bill Lane and Sam Lynch, who was at that time a candidate for Secretary-Treasurer, to prepare a report on this proposal and tell you what it is and what they feel should be done about it. Since this is a decision for the new Executive Board I will turn the floor over to Paul for that purpose.

MR. BICKFORD: This summary will also cover the report by Porzak, Swanson and Blackney of COMMON and also the ACM Code. The report of Porzak, Swanson and Blackney proposal proposed expenditures (I'm going to briefly summarize what they proposed:) cost of a secretary, $15,000-$25,000 a year; $5,000 for clerical staff; $1,800 for office; $6,000 for travel; $4,000 for telephone; and miscellaneous expenses -- for a total of between $32,000 and $42,000 a year. Also, they recommended a number of bylaw changes in their report.

Just about this time the ACM proposal was given us. Their proposal considered salaries in the range of $19,000 for two people -- a $12,000 man with a $7,000 secretary, with taxes and fringe benefits of $2,000; also, travel, entertainment, phone and other services. But it excluded the printing of the Minutes of the meeting. The ACM proposal did not require any bylaw changes; it required the approval of the Executive Board. But that expenditure amounted to approximately $49,000.

The Executive Board discussed it and decided that the best proposal would be to continue to function under the present system that provides for the Secretary-Treasurer, and a full-time person to assist him in his

duties.

The reasons we turned down the other proposals were: Currently COMMON could not afford $49,000 a year, or even $42,000 a year, or probably not even $40,000 a year to support such endeavors; that's number one. Also, with the Porzak, Swanson, Blackney report, the time it would take to implement the changes in the bylaws would probably be at least a year, and judging from the response we've received on past amendment we probably would not get it done.

So in order to get something going and to provide the help that the Secretary-Treasurer needed to get the CAST out on a regular basis and to perform the other duties of the Secretary-Treasurer, such as keeping track of the membership, balloting, etc., we felt that it would be better to provide the Secretary-Treasurer with funds to hire a secretary to perform these duties.

In the meantime Share and Guide had decided to consider strongly this proposal. I'm not sure yet whether or not they've accepted it, but I feel that if they do accept it and they do find it workable for them then possibly in a year's time we might reconsider their proposal. Possibly COMMON might be on a more sound financial

basis or more definite financial basis, and at that time we can reconsider the issue and if it seemed feasible and reasonable we could then implement it. But there seems to be no question that now it would be premature.

I think that pretty well summarizes. Are there any questions?

PRESIDENT STANSBURY: One of the general appraisals that the three Presidents made at Chicago was that the probable cost would be knocked down from that quoted $49,000 to around $30,000-$35,000 a year, since the three user groups would not require three times the facilities, by any means.

I should like to mention that in addition to not considering the cost of publishing proceedings neither of these reports considered the cost of publishing CAST. That is an additional expense. We'd still need about a $40,000-$50,000 a year budget to cover, and in either case we'd be committed to that expenditure and we would almost certainly have to have, roughly, a year's backlog as a cash reserve in our treasury before we could even consider it.

MR. BICKFORD: Also, we might mention that the proposal that we've accepted will be published in the

next CAST.

PRESIDENT STANSBURY: Is there any discussion from anyone who would care to express his views on it -- endorsement or opposition?

MR. EDWARD J. WOOD (3631): Are you going to tell us a money amount that is going to be allotted to the Secretary-Treasurer to hire this secretary?

PRESIDENT STANSBURY: We have not stipulated a money amount, but we intend a clerical type secretary, someone who would handle the routine duties of the position. We stipulated specifically that Chuck, or the Secretary-Treasurer, was not to hire such a person, but that COMMON would pay the salary. It is expected that the hiring would be handled through the Secretary-Treasurer's installation. If that wasn't feasible, then he would obtain the services of an office temporary on a permanent basis so that COMMON is not placed in a position of being an employer and having all the Social Security, FICA, tax reports and all the other things that have to be submitted in such case.

Does that clarify what we propose to do?

MR. WADE NORTON (1125): I believe that it was clear to most of you what was accepted and what was rejected, but as I sat and listened to the details and the

way it unfolded I am not absolutely sure. As a matter of reiteration, then, that which was rejected was ACM's proposal to provide and that which was accepted was a report which detailed what ACM had proposed and recommended that it be rejected in lieu of the proposal that the Secretary-Treasurer be authorized to hire clerical help.

MR. ROBERT J. SNAILER (1495): Is there a procedure for approving a budget? In other words, would this have to be submitted to someone? Is this to be approved just by the Executive Board or is it open to the general membership? This is a general type question for any type of expenditure.

PRESIDENT STANSBURY: The Secretary-Treasurer is authorized by the bylaws to pay the expenses of being a Secretary-Treasurer. In our opinion the Executive Board, therefore, can direct him to employ someone to enable him to perform these functions. We did not consider that that particular option needed to have the approval of the membership, but it definitely would normally have the approval of the Executive Board.

MR. SNAILER: Then this is just for the information of the people here, right?

PRESIDENT STANSBURY: That's correct. It's

the report that I promised in April at the Chicago meeting.

I'm going to ask Chuck to give us a financial report, not too formal.

MR. MAUDLIN: I made a hurried trip up to the room and picked up many things; I made a list of things I should bring down and I absolutely forgot that.

The only thing I can remember is the first four digits of what our current balance is, and they happen to be correct. We do have in excess of $20,000 -- $20,816+some odd cents. Most of that came from the Chicago meeting; $15,700 came in from the Chicago meeting and there was slightly in excess of $5,000 in the treasury just before that.

I am guessing that there will be $12,000 from this meeting. That's a pure blind guess, with no guidance from the program chairman.

The last issue of CAST cost $1,701, by the way. We have a backlog to produce CAST for about a year and a half if we have no money coming in between now and then and no meeting expenses that aren't taken care of by registration fees.

PRESIDENT STANSBURY: The objective of the Executive Board has been to establish a balance that would

enable us to operate for approximately a year in case things went absolutely awry and we lost on everything. Once we accumulate that balance we're going to consider publishing the proceedings of the meetings themselves, and that's the inducement for belonging to COMMON. The proceedings would be available only to members of COMMON, so that would have some additional sales appeal.

Unless there are other comments on this specific subject I am going to open the floor for questions.

MR. WM. F. BURGGRABE (3418): At the opening session I made a few comments about how local users organizations could possibly become affiliates of COMMON, or something of that sort. Now that the new Board has met I am asking again. Most of these people are also new members. Those of us who have been around for a while realize what the situation is. I would like clarification on what could happen and perhaps a feeling from the Executive Board of what they would like to see.

PRESIDENT STANSBURY: Because this is a future decision actually I'm going to ask Chuck Maudlin to give the answer to that. He was detailing what he could now do and what he was now doing to enable things such as this. Paul and I have discussed it with him, too.

MR. MAUDLIN: I'm afraid I didn't understand everything that came out of that microphone, so I'm going to ask Bill to go through that again and then I'll do the best I can.

MR. BURGGRABE: The question probably ought to be put in two parts. The first is: What are the present possibilities of interphasing local users organizations with COMMON? The second part is: What possibly could we do and perhaps what does the Board feel about this sort of a situation?

I think there are obvious pros and cons, and I think the membership is interested. I think there are advantages and disadvantages.

MR. MAUDLIN: Currently there is nothing clearcut in the bylaws to describe what the requirements are for membership other than association with one of the small IBM machines. After we get out of that category then it's in the realm of interpretation. In the past there were the small regional groups sprouting up and the interpretation was generally that within a single installation there would be at most a membership for each computer installation under separate management inside that organization; that if there were two different computer organizations

they were logically entitled to membership. There were presented to the organization the ground rules of maintaining membership; that is, they had to have enough people at meetings to satisfy meeting attendance requirements for two organizations, and with the current bylaws that would be responding to ballots from both representatives, etc.

There were two things that caused me to interpret a little bit differently in the organization of the Houston 1130 users group. I'll talk about that one first and then defend it on the basis of something else.

In the bylaws there is a provision for the Executive Board doing some things, and when I found out about the Houston 1130 users group the Executive Board wasn't really around to ask everything, so I made some decisions. The Houston 1130 users group is a honest member of the organization in the sense that they have a number and they get a task and they are on IBM's list as an 1130 customer. There is an IBM branch office number and an IBM customer number -- it happens to be a customer number of one of the members -- and they are currently maintaining the meeting attendance requirements by attending the meeting.

In addition to that membership which is in

the name of the Houston 1130 users group and in the name of the president, there are several members of the Houston 1130 users group that are additional members of COMMON. The president is not, and so it means he is only getting one mailing. It is my opinion that at least for the time being they're giving us more than we're giving them by just putting them on the list. I'm not sure that will always be the case because there will be some organizations that don't fit in that category.

One of the things that caused me to make this decision (and I spent well over two weeks in correspondence on it) was an organization that I can't even remember the name of. There are several river forecast centers throughout the country -- I think there's one in Cincinnati, one in Fort Worth, one in Atlanta and I don't know where the rest of them are -- and they're having a hard time making the meetings. They are small installations, just like many others. I think all of the installations got a letter about the same time saying that if they don't make the next meeting they go out.

One of the members proposed a joint membership for the entire group and it would be up to them to distribute information to the rest of the river forecast

centers. He gave me a title that was very long. I think it had "southwest" in it and I think it had "hydrologic" in it, but I can't remember the rest of it. They agreed to distribute to all the river forecast centers the information that was in CAST and in the proceedings that was pertinent to them.

It seemed to me that this group fit in the same category. As a general rule I would vote that we should continue that policy, but I'll stand back now that there's an opportunity to get a consensus other than mine.

That answers one question. I don't even remember what the other one was.

MR. BURGGRABE: I believe that answers most of it except for one thing. We were talking about the fact that if an organization actually became a member they might have a possibility then of also paying a subscription fee to get CAST mailings for all of the members, whether all the members are members of COMMON or not. That was the point.

MR. MAUDLIN: There is a currently approved policy adopted by the Board for distribution of CAST on a subscription basis to IBM installations, that is, to anybody in IBM, anybody that has an address which has "IBM" in

it, provided they pay their $15.00 subscription fee. There have been very few requests for subscriptions other than from IBM, and so there have been no real opportunities to make a decision. I have quoted a price of $15.00, but, to the best of my knowledge, no one has asked to take advantage of it. I don't think I've had to make a decision on that.

I would like to suggest that anyone can subscribe. Fifteen dollars covers the cost of CAST for a year unless it gets a lot bigger than it is, and I think it would be a good idea. It's my understanding that Share has that policy of distributing CAST to her FFDs on a subscription basis, and it would be my opinion that we should do that if anyone wants it. Large numbers, I would like to suggest, is a bad idea, but then, on the other hand, I don't think anybody would want many copies at $15.00.

PRESIDENT STANSBURY: There was one specific clause in the old bylaws which prohibited subscriptions to other than IBM installations or users. When we wrote the new ones, when we drafted the new ones we took out that prohibition. We didn't quote a price for other than IBM installations, but we did take out that prohibition so that it could be done without amending the bylaws.

MR. JAMES MARK (1988): Are we then from our Secretary-Treasurer's remarks to understand that local users that are organization-structured will be permitted in the future to maintain a relationship with COMMON?

MR. MAUDLIN: Unless I am corrected to the contrary at this meeting.

MR. BICKFORD: I think that the philosophy behind COMMON is that we will provide a structure wherein these groups can participate in COMMON. I think the structure and organization is such that we can make it so that they can participate in COMMON, carry out their own goals and still not bother about rules and regulations.

PRESIDENT STANSBURY: I would like to point out my own interpretation of Chuck's rule, too. The specific rule is that the members of a group do not become members because of a group-type membership in COMMON. We intend it to apply so that individuals may become members but permit the group to become a member and permit members of that group who are not members of COMMON and who will not become members of COMMON to subscribe to CAST and obtain the proceedings through the IBM office.

MR. WADE NORTON (1125): I am sure it's not the purpose of any Board member here to defeat the good

purposes of users, and as surely as there is a precedent (although we don't necessarily have to look to precedent) in that stockholders in a particular company may be either other corporations or individuals I think the same should apply to membership in COMMON. I can think particularly of an activity within a project that appears very dear to Dick Gabriel's heart, and that is the 1620 as a tool for teaching in the high schools. There is a wealth of information around on 1620 and that information needs to get to these people that are teaching in the high schools.

I submit to you that when they fight the battle against proration of their salaries that they're not going to have unlimited funds to attend meetings. So I think that the policy that Chuck proposed here serves the interest of users.

MR. EDWARD J. WOOD (3631): For the August 1 CAST I sent an 8-page booklet to Chuck Maudlin to be mailed as a separate mailing. Incidentally, this was called an opinion in the opening meeting and it is not; it's handed out by IBM and I just re-set it so it would be readable.

Apparently this caused a pretty big increase in cost of mailing CAST.

You shake your head, Chuck, but $1,700 sounds

like an awful lot of money. Are there that many new members?

MR. MAUDLIN: Yes.

MR. WOOD: Fine.

PRESIDENT STANSBURY: This was a 90 or 92-page CAST, which was rather substantially larger than the last one. It was not due to the 1130 information. Both Chuck and I felt that this was an appropriate way to distribute it. We recognized it was going to people who didn't need it, but at the same time that the cost of segregating out those who didn't need it from those who did would not be worth the trouble probably.

MR. WOOD: The reason I'm really bringing this up is that the opinion of the few people I've talked to about this (and I haven't talked to everybody, of course) is that brand new COMMON members who are 1130 users could also use this. Now, is there a policy that you could adopt or is there something you could do to see that this would get distributed, or something like it in the future if it came up for 360 or 1620? Is there a policy you could use to be sure that this would be included in the membership things that are received from IBM (for instance, the COMMON Reference Manual) -- that this could be included as a

new members handout?

PRESIDENT STANSBURY: I don't intend to answer that question. Paul?

MR. BICKFORD: If we can find a way in which it could be reasonably implementable, possibly so.

MR. WOOD: I think the thing that has to be taken into consideration is this. I specifically requested from Chuck that this be made an 8-page pull-out or separate piece in this mailing, for the simple reason that it's a tool that you would leave at your desk and pick up whenever you wanted to do some quick coding. I feel this is the way it should be handled, but, of course, your decision would be final.

MR. BICKFORD: It's difficult to handle or to process all the possible variations of requests that we get for distributing contents of CAST. Some people like them bound and punched with holes, and some people don't. Some people like all types of variations. We don't have an organization at the Secretary-Treasurer's installation such that he can implement all these variations easily. I would like to think that if it was something of value to be distributed from now on that we could fairly easily make something that we could do every time, without any significant

additional cost.

MR. WOOD: Every time with CAST?

MR. BICKFORD: The new membership packet, with specific reference to what you're talking about. Any time we try to change CAST in the way it's presented, as I understand it, it's difficult because it is set up and it has worked reasonably well. If Chuck wants to change it and makes this change effective and continuous, then fine, if it's something worthwhile.

Does that sound reasonable, Chuck, that if there is some change you want to make in CAST and the group wants it that way we'll consider it for change.

MR. WOOD: I'm sorry. I don't mean that this should be in CAST. I don't mean that at all.

PRESIDENT STANSBURY: I think I may have a satisfactory alternative; I don't know. As you know, this information is prepared from masters that are prepared from the copy you send to Chuck. How much trouble is it for the submitter on such items to keep a duplicate set of the originals, or request the masters from Chuck -- let the author of such information or, in your case, a person on the 1130 process or possibly a two or three man group be the custodian and Chuck could simply supply them, I am cer-

tain, with the addresses of the new 1130 members and the group could take over the responsibility and possibly charge for reproduction cost or a handling charge for sending it to new members.

MR. WOOD: The author happens to be in New York because it was authored by IBM I don't know how many years ago, and what I received was a Xerox copy of a Xerox copy of a Xerox copy of a Xerox copy and it wasn't very readable; so what I did on my own, at my own expense, was to have it set so it's readable. Maybe the whole answer is to have IBM print them themselves and distribute them to every 1130 installation and just forget about it.

Is that what you would say -- to attempt to get IBM to do it?

PRESIDENT STANSBURY: Obviously. Then it doesn't cost us and it doesn't cost you.

MR. WOOD: Fine. That's what I'll do then.

PRESIDENT STANSBURY: Particularly if it was originally authored by someone in IBM I'd suggest you try to track down the author because it was probably copyrighted.

MR. WOOD: I know it's not copyrighted.

PRESIDENT STANSBURY: As to the actions of

the Secretary-Treasurer in reference to membership for local user groups I would like to have a show of hands from those present who would endorse the Secretary-Treasurer's action, policy, as he stated it.

(Most hands were raised in favor.)

PRESIDENT STANSBURY: Are there any who wish to express an opposing viewpoint?

MR. JOHN CRANDALL (3473): Won't this tend to decrease our membership if there are people that are now small users that will join a local users group and tend to drop out and rely on the services of the users group?

PRESIDENT STANSBURY: You've been at this meeting. You tell me what you get the most information out of. Meetings? The proceedings of meetings? CAST? I think that those people who can attend will become members, that this would only apply to those who cannot. That's my personal opinion.

MR. DONALD S. GARDNER (1150): The problem that Ed is talking about is certainly as old as the group is. Many years ago we had a 1620 users group and we put together newsletters and had lots of information that was quite applicable from their viewpoint, and as late as a year ago I overheard 1620 people talking about the problems of

readers of five years ago. It seems rather foolish to have talent go to the trouble of publishing and writing the newsletters and have that information unavailable to new people, and these new people will now be spending time and resources doing what some of us have done many years back. I think it's a responsibility of COMMON.

PRESIDENT STANSBURY: To confirm Don's statement: About six months or a yearago I got a call from IBM in White Plains wanting to know if I had a particular 1963 issue of the 1620 newsletter. Some IBM installation in Iran needed a copy of that specific newsletter. How they found out that the answer to their problem was in that newsletter I don't know. However, there does exist a file of newsletters and certainly there are complete sets of proceedings in IBM's file. There has been serious consideration several times of having someone take over those files as archivist for the group and permitting him to reproduce at cost specific proceedings, newsletters, papers, what-have-you. It has never been implemented because no one was willing to take the responsibility.

There are now, however, a lot more members than there were then. You might wish to farm it out. But such a thing could be done if the people in the organization

were willing to accept the job of doing it. I don't think it can be imposed upon the Secretary-Treasurer.

MR. WM. A. PEASE (1516): I think it's impossible for any organization to lift the realm that it operates in without lifting itself, and since the purpose of COMMON is to free people's time (among other things) so that they can find out more new things for us I think it would be the proper function of a committee to compile some sort of subject index to the past newsletters of CAST. They could have representatives, say, to the Administrative Division to operate with the archives, so that they could have access to the archives at IBM or could communicate with them; and some sort of lines could be set up to distribute, for example, to 1130 people an index of 1130 articles. You would throw away the old index once you got a new one, so you wouldn't be accumulating paper.

PRESIDENT STANSBURY: Such an index was prepared for the proceedings of the 1620 users group for the first three or four years -- I've forgotten whether it was everything up to '63 or everything up to '64--and was published in one of the proceedings. In other words, there do exist partial indices to past proceedings. I don't believe that any has ever been compiled for the newsletter.

I'm certain about CAST. The point I was trying to make is that I completely agree that it's a subject for a committee. You people organize the committee, please. We'll work with them.

MR. NORTON: I'd like to get a frame of reference here. It appears to me that one of the things we are striving for is to insure that new users get information immediately and, as such, we have been asked to put a lot of things into CAST, to be sure that all of the information that is pertinent to the activities of a project or even of a committee goes via CAST even though it may not be of interest to the full membership.

The other problem, as I see it, is to get that essential information about the organization to everyone. We are a very large organization and we are a very diverse organization.

These things are the things we're talking about. It's really a communications problem: How do we get to those who need to know that which they need to know and in a form in which they don't have to dig out of it or dig through a myriad of stuff just to find what is pertinent.

Now I want to make a comment, having defined

the frame of reference. It is my personal opinion that the reason that the newsletter dies is that it was redundant by the very fact that it was too broad and was trying to cover all of the machines. It was trying to perform the function that had been delegated to CAST in effect.

Now, there is precedent, since we're generically the old 1620 users group, for newsletters that pertain to specific teams or sub-units of the users group. I can cite you the Electric Utilities Newsletter, which was a regular publication, provided information that was needed by the utilities group, and did not clutter up the general newsletter.

This was not a function of the Executive Board, though. It was the function of that particular group.

As I see it, the only problem that exists on newsletters in projects is finding an editor and then being sure that some procedure is set up or some means is made to get this information into the hands of new members. I don't think we have to mail a volume yea thick each month in order to do this.

MR. GARDNER: Wade, I would like to take issue with just part of what you said. The newsletter for

any machine group, whether it's an individual one for a group or a newsletter to cover all groups, in my opinion is very important. I think everybody in this room will benefit by one. I think the reason it died is not that CAST took it over, but that it was allowed to publish things in CAST that belonged to the newsletter. Someone allowed it, perhaps by default because we had no editor. But CAST took it over and started publishing verbatim what was sent in, items relating to a particular machine group, and took over the duties of the newsletter. That's why it died. It had a way to take care of these things.

We could indeed make some effort -- in some committee or by some people -- to revive the idea of a newsletter, with types of items that are common denominators.

PRESIDENT STANSBURY: I would think the implication here is that the newsletter should be confined to the users of a narrow, defined range of interest -- one machine type application, project, or something of the sort. I would say that on a project basis I would probably agree. On a machine type basis I'm afraid I don't.

MR. ROBERT T. SCULLY, JR. (1957): It took us something on the order of nine months to find out that

COMMON existed as an organization for 1130 users. If I may suggest that we can exert any pressure on IBM, I think they should be at least the unit that informs their new customers of this organization for this particular type of equipment. It's because so many requests have been made of our sales office that I found out about the existence of this organization. If in addition to delivering the new equipment they could include as part of the installation procedure a back-up file or some sort of index on what has already been discovered by the users of the equipment prior to the new customer coming on the line, we could get additional benefit out of that.

MR. MAUDLIN: I don't think IBM should be condemned for their role in that particular respect. They do something that we do. Since we do it by bylaws and thought a long time about it, then I can't condemn somebody else for doing the same thing. We won't give out our membership list. The equivalent in IBM is "We won't tell you who has IBM machines."

Now, since they won't give it out the next best thing they can do is to inform all purchasers or renters of machines about the existence of users groups. Corporate IBM frequently (I don't know how often, but

significantly more than once a year) sends to branch offices and sales reps and everybody else associated with customers memoranda describing users groups, pushing them, naming the secretaries, publishing that in the DATA PROCESSOR, which some of you have seen, I am sure.

The customer, however, has one interface into IBM and that's the sales rep, and it's his decision. If he thinks it's going to help his sale or increase his sale or increase his relation, then he's going to make COMMON known; and if he doesn't think so, he may not. There are several subscription services available from IBM and the only way you find out about them is from your sales rep, and corporate IBM tells the sales reps that it's a good deal; then they make it available to the customers. The only people we can condemn are the sales representatives who don't get the word out.

We can condemn us; we don't have the right kind of policy to get the word out, too. Maybe we ought to be advertising in DATA MISSION; I don't know. But corporate IBM is at least making the word known to their salesmen and the information should be gotten to each of the users.

I would like to comment on the real death

of the newsletter also. The newsletter died of lack of items for the newsletter. It was published as part of CAST. There was an editor doing a good job. That editor was forced to resign because of changing installations. At that time the editor had had submitted to her five documents that she had had for almost eight months waiting for enough to make one page. Those things were put together in one blob and went into a CAST as they were submitted. And there has never been any real move to create a newsletter since then.

I would suggest that if there are people who want newsletters for any thing that is a function of the Administrative Division, and we should iron out a way to get that done. But until people want to contribute, that's the situation.

MR. MICHAEL SCHILDER (1557): I was either fortunate or unfortunate in not being part of the 1620 users group (depending on how you want to look at it) in that I can now look at COMMON from a different view than those who were members. We have a number of machines here and we have a number of applications, and the answer you gave before to one of the questions was: How can the Secretary-Treasurer or anybody decide whether or not something pertains

to a particular machine or not. I think the answer really boils down to the fact that there are some areas which are of general interest to COMMON members. There are some areas which are of particular interest to application people and other areas of particular interest to machine people.

I don't have immediately the answer on how to separate these out, but perhaps by finding out from the members themselves what areas they're interested in then we could not reorganize CAST but perhaps organize a newsletter which would go to the people who want to get the newsletter. Then maybe CAST could be separated into an 1130 section and a 360 section, etc.

The important thing is that COMMON has to recognize now that even though we have a common goal (and I'm not trying to make a pun) we at the same time do have individual area interests for which we want to be satisfied. And certainly as Program Chairman of this meeting I found that out. But I think we have to consider the fact that there are two separate and maybe contrary desires on the part of people: to be together and also to be separate. I think we have to recognize this, and I think there are a lot of areas which we may have to reevaluate in order to come up with a solution.

PRESIDENT STANSBURY: I submit that what we are discussing here is a communications problem within CAST. There is a general session of the Administrative Division tomorrow morning at ten o'clock in this room, and ways and means of implementing this I think can be discussed then.

MR. MAUDLIN: There have been two other sessions that I've been in where there was a discussion of thumbing through a large issue of CAST to find those articles you were interested in. I'm probably not the best in the world for creating titles, but there is an honest effort on my part to get"machine type" in the title; and the first page of that document, whether it's good, bad or what-have-you, does contain at least some indication of what is in all of the documents. I suggest that you can go down the first page and if it's machine-oriented then there will be a machine type in that title and if it's not machine-oriented and it's of general interest to the membership the article will be directed to the membership, or the title will be such that it will indicate that it's to the membership.

PRESIDENT STANSBURY: I think that will close the discussion, and I think the appropriate place to con-

tinue it will be in the session tomorrow morning.

I know that the OS project got stubborn and decided they were going to continue meeting. The 1130 projects -- one or two or more -- got stubborn and decided they were going to continue meeting. That sounds as if the Executive Board and Program and Local Arrangements Chairman, and so on, could use some guidance as to why you felt it necessary, what your objections were. May I have some comments to that effect?

MR. WOOD: Tomorrow morning we're starting at 8:15, going to 9:45. The problem was that an original schedule was sent before the agenda was made up. The agenda was made up and scheduled us from 8:30 to 9:30, 10:00 to 11:00 and 11:00 to 12:00. One of these talks must take an hour and a half. Now, we're not going over anybody's head, but it is considered a tutorial and to have to break it for a full hour and a half -- going from 9:30 to 10:00 -- and then through this Administrative Division to 11:00 -- would just hurt us terribly.

PRESIDENT STANSBURY: I'm not criticizing.

MR. WOOD: I know you're not.

PRESIDENT STANSBURY: I'm merely asking the reason.

MR. WOOD: I realize you're not criticizing and I'm merely telling you the reason.

To attain this we had at first decided that we would go from 8:30 and hopefully at 9:30 hold everybody through the coffee break. Then Mike was kind enough to suggest that he would get us an early start and then we'd go fifteen minutes into the coffee break and then still give the people an opportunity to come to the Administrative Division meeting.

MR. JOHN FISH (1878): Our meeting was extended from three sessions to around six, with three more planned. We have encountered many areas which we didn't realize had a common interest, and this meeting has brought that out. We'll plan for a much broader session in Houston, in the area of seven to nine sessions.

PRESIDENT STANSBURY: Then the general consensus seems to be more or less that the project itself didn't anticipate all of its requirements and that they weren't properly communicated to the Program and Local Arrangements Chairmen.

MR. FISH: It's spontaneity.

PRESIDENT STANSBURY: The spontaneity is fine, yes. Don't get me wrong; I'm glad it occurred. I am

very pleased with the way this meeting has been going, and I think Mike and Joe and the division managers deserve some hefty thanks for everything they've done to make it so. Let's give them a round of applause. (Applause)

Are there other items of business anybody would like to discuss?

MR. DONALD KIEL (3625): I realize that our meetings at the present time are planned for at least two years in advance, but I would like to raise this as a topic with the Executive Board for consideration: the question of spacing of meetings; the question of five months, five months and two months or three months apart -- five, four and three. This makes it a bit difficult for us to justify to our employers the short time between taking these junkets.

I think it would be appreciated at least by myself if we could try to space them four months apart, or as an alternative proposal -- Guide and Share both meet twice a year -- the possibility of this group meeting twice a year for perhaps four days rather than three times a year for three days.

I am vitally aware after this meeting of the problems of scheduling and I know there are bound to be overlaps where in small installations frequently only one

person can attend. This has been true for at least the past two meetings as well as this one on my own part, where I would like to attend more than one session at the same time. This might give us a little less chance of overlapping too much.

PRESIDENT STANSBURY: I'll give the Executive Board's position or decision, the reason for the decision, and then open the session to comments from the floor.

The meetings, despite Don's feeling, are not scheduled in quite the intervals he said. They're scheduled, roughly, four months apart during more or less nine months of the year. If you will note, there is a meeting in September (which we had expected to continue), there is one in December about four months away --

MEMBERS: That's three months.

PRESIDENT STANSBURY: I said "about"! You can't make it any later in December and you can't make it early in January. There are obvious reasons for that. There is a meeting in April. We have found from past experience that summer meetings of COMMON hurt and hurt badly. There are too many educational institutions in here, so we deliberately try to avoid the three summer months.

That's the basic reason for the spacing. In

addition to that, we have 1130s, which are relatively new machines, and we have a very large number of new users. There were something like 160 in the new users meetings at this session. If we spread that out and try to get by with two meetings a year, I submit that with the systems in the state of flux that they now are this just wouldn't work. You'd be missing too much information.

I might add that while Share and Guide each meet twice a year those meetings are spaced so that Share's Systems Division meets at the same time that Guide does and Guide's Systems Division meets at the same time that Share does, and as a consequence their Systems Divisions meet four times a year.

That's the reason for our decision. Does anybody care to comment?

MR. ROLAND MAGEE (3079): As to the scheduling of the meetings in the last year or two, it seems to me they follow immediately after or immediately before a holiday and this often creates problem with transportation, and so on. I'm wondering if that couldn't be taken into consideration and spacing changed.

MR. MAUDLIN: I have a real quick answer to that one. There is a great deal of effort that goes into

avoiding holidays and when you avoid all of them you're either immediately before them or immediately after them, and usually both.

MRS. LAURA AUSTIN: Jim, I'd like to add to that, too. In working on the planning of future meetings the Administrative Division has been planning through 1972 for meeting dates, and at this time we have found that in 1971 we could not get the dates we wanted. We were too late trying to get dates in hotels for 1971. When you stop to think that COMMON has not planned this far ahead in the past you can see some of the reason why the meetings have come at the dates they have.

PRESIDENT STANSBURY: Does anybody else wish to sympathize with us?

MR. FISH: Speaking for the OS Committee, during this session we had approximately twenty-four people at the average session. Of these twenty felt they could not attend the Houston meeting because they couldn't justify to their companies attending another session so soon after this one. Of these twenty people many could have contributed a great deal to the Houston meeting and it would have allowed us to plan better for papers and presentations and to set up subcommittees to work harder. I feel this is

an extreme handicap for our group.

PRESIDENT STANSBURY: I agree because the Executive Board has to attend every COMMON meeting and we had a meeting in Philadelphia in July. It does impose a burden and I will have to admit that the hard core of the project effort, whether it be OS or an applications project or 1130 or 1800 or Model 44 -- whatever it be -- will be a relatively few individuals to carry that load; and the success of your project depends upon how much they can contribute and how much they benefit from these meetings.

From what I have gathered (Dick Zerweck, John, is Manager of Operations at our installation, and he's been talking about your meetings) the people who attended here feel that they acquired a great deal. I submit that even if those same people attended Houston there will be additional attendees from the midwest and you will probably find it will be an equally rewarding meeting.

MR. FISH: I felt that the Houston meetings would be much better with the addition of people from the east coast who could plan and present better formal presentations in addition to those of the Houston people. I think this could be done if we had better spacing of meetings, such as every four months.

PRESIDENT STANSBURY: My answer to that is that my boss and Dick's boss is here, and he has requested Dick to attend the Houston meeting.

MR. BERNARD A. SOBEL (1490) I'd like to defend, if I may, the Board. Sometimes it's hard to defend yourself and I'd like to defend the Board in the light of my experience with a few other groups. I detest coming to meetings and I have to detest it. Every day I'm away from my office means that I have to work two days more when I get back. And this is true of all of us, and it's even more true of the members of the Executive Board who must attend every meeting, who must attend interim meetings.

I am grateful to them for doing this. They are doing something that I myself want to have done and that I don't have the time to do myself, and I cannot possibly object to their setting up a meeting in Houston tomorrow or the day after. The fact that I personally will not be able to attend cuts no ice. I will send one of my girls, I hope. I will possibly walk in on it -- and this meeting is only a half hour airplane jaunt from our operation.

The simple fact is that there are many organizations in which all of us are interested. I am inter-

ested in this group; I am interested in Share. I'm also a member of and active in possibly ten technical societies -- in the chemical engineering, chemistry and computing sciences fields. If I were to attend every meeting of every one of these I would spend my entire year attending meetings. I don't think the Ethyl Corporation pays me for that. I don't think that's true of any one of us here. The fact that there are some people who are lucky enough and willing enough to spend their time -- God bless them. (Applause)

MEMBER: Don't put the cross on the Executive Board or on your management. This cross belongs on your own shoulders in this case for the simple reason that the management can't know the value of what you get here unless you communicate it to them in a report when you're still warm and enthusiastic, when you get home. That's when you write your report and write it in a concise manner. Don't try to go into detail on all the papers. Just write in a concise manner to whet their interest so that they can come to you for the details. See that it gets distributed in proper places around your organization. Then you'll be in a good position.

One of the best things that happened to me

was missing the Cincinnati meeting. My boss told me the budget couldn't afford it. What he neglected to do was to check with his boss. His boss happened to be Director of Research who is immediately under the corporate Research Director. Now my boss has to justify my staying home when there's a COMMON meeting!

PRESIDENT STANSBURY: Is there any other discussion?

MR. LARRY ARMBRUSTER (3408): I'm going to kick a horse that I thought had been dead for about a year a couple more times. As some people will remember, I got up at the open Board meeting in Cincinnati and made a request for information as to the bonding of the Secretary-Treasurer who handles, according to his own statement, some $20,000 right now, and the bonding of the Local Arrangements Chairman who, according to the Secretary-Treasurer's report, is handling some $15,000 or so.

I have the utmost trust for every one in this room. If you don't believe me, come up and I'll give you money to buy your dinner tonight if you promise to pay it back. But this is not the point. We are an organization with a rather large membership, as has already been stated, with a lot of money relatively speaking; but they are in no

way protected by a bond.

I would like to get an answer as to whether this has been discussed by the Executive Committee and what decision has been made.

PRESIDENT STANSBURY: There has been no discussion as yet of the bonding for the Local Arrangements or Program Chairman. There was a resolution passed by the Executive Board, I believe originally in San Francisco -- possibly at Cincinnati, directing the Secretary-Treasurer to bond himself at COMMON's expense. Unfortunately, we did not give him a deadline. He has not done so yet. Merely as a member of the new Board and not as the presiding officer, I can say that the Executive Board last night passed a new resolution instructing Mr. Maudlin to bond himself immediately.

MR. ARMBRUSTER: Thank you. And I had my hand slightly tarnished a little bit when I was told that it was already in the bylaws. Now let's beat this other horse -- the Local Arrangements Chairman.

MR. MAUDLIN: That will be a part of the bonding when it takes place. It will be through my bond. He will be an agent of mine.

PRESIDENT STANSBURY: That I didn't know.

Thank you, Chuck.

MR. ROBERT SNAILER (1495): This is about your secretary, Chuck, and I'm completely in favor of the idea. I would like an opinion on this: What is the benefit going to be by Chuck having this secretary? Is this secretary going to serve the existing membership well, or is this secretary also going to entice many more new members than we would have if we didn't have a secretary?

PRESIDENT STANSBURY: I didn't think she was going to be that attractive! I hope not. I don't think Joy would let Chuck get away with it!

Joking aside, though, I don't think it's going to assist us in the slightest to obtain new members except as the prompt and efficient performance of the duties of the Secretary-Treasurer would encourage new members to join the organization. We wouldn't have hopefully too many letters addressed certified mail to the President of COMMON wanting to know why Chuck hadn't replied to a letter which rather obviously he had not received.

It will increase the expenditures of COMMON in order to give you the service that we think COMMON wants and needs now, but we don't expect it to give us new members per se except in such fringe benefits as arise from

the prompt and efficient performance of Chuck's duties.

MRS. AUSTIN: This new membership problem is going to be one of the topics for discussion in the Administrative Division session tomorrow.

PRESIDENT STANSBURY: Laura will be Chairman of that session. Laurs has been the Vice President for two years and I couldn't have done the job without her. (Applause) She now has to give up her position not only as member of the Board and Executive Vice President, but she also has to give up her post as Administrative Division Manager.

I think Paul has a couple of announcements he would like to make.

MR. BICKFORD: Just before coming into the meeting I did contact a new prospect for the position of Administrative Division Manager, Bill Lane, who was formerly on the Executive Board, formerly Western Region President. He was also a candidate for President. He did decide to accept the position of Manager of the Administrative Division. So we now have all Division Manager slots filled and I think we have excellent people in all of them. We're going to strive to continue to build COMMON and strengthen it within. I think that with the good managers and good

project leaders thatwe have we'll have as good or better meetings in the future.

PRESIDENT STANSBURY: And I can also add that for the first time since we've had division managers they aren't members of the Executive Board. We have been trying for two years to get rid of the feeling of wearing two hats because we felt the jobs were impossible for one man to perform successfully -- two different jobs.

MR. SCHILDER: I don't at this point want to start something new, but I do want to bring up a question that I've been thinking about. I don't know how to phrase the question. I know we have a structure which says that the way in which we collect our "dues" is by attendance at meetings. Before this time I have never been aware of a problem that does exist. I raised it with you. Chuck knows about it. The question was also raised with me from people locally. This is it:

When an installation sends a number of people to a meeting because the meeting happens to be located in that city or nearby the end up paying the number of people times the fee. I discussed this with my manager, the one above me, because I had to get his okay to get the money. One of the ideas he suggested which I think I'd like to

offer for consideration is that maybe at this stage of the game, or somewhere along the way, we should consider an installation membership rather than having the individual who attends pay.

PRESIDENT STANSBURY: I'm going to correct your phrasing: an installation registration.

MR. SCHILDER: Okay. Actually the installation does belong, not the individual. One of the things he said was, "I would have no difficulty, for example, okaying $150 a year" -- that's just a figure he picked out of the air -- "if I knew that, as a result, whenever there was a COMMON meeting I could send some reasonable number of people, paying only for their luncheons or their cocktail parties (extra things that normally you put on an expense account anyway)."

I discussed this with Chuck, and I thought I'd bring it up. I don't intend to keep the meeting here all night discussing this, but I just wonder if we shouldn't at this point do something to start to consider this as a possibility, or something like it.

PRESIDENT STANSBURY: Paul has suggested that the Executive Board will take it under advisement. I think I will point out that there are two or three alternatives

here. We agree that the registration fee is not handled equitably for those people who attend every meeting, and particularly for those who have multiple attendees. I think the proper procedure here would be for the Executive Board or the Executive Board plus possibly some ad hoc members from the group (as an ad hoc committee) to look into the matter, look over the various alternatives, see what kind of position they would cost, and have a report at the Houston meeting. Then we can bring it up for discussion, and have some facts and figures when we're talking about it.

But that is Paul's decision and I'm going to have to relinquish that, Mike, to him.

MR. SCHILDER: I think that's a good idea. I wasn't trying to raise it on the floor now, but just wanted to bring up the question for consideration at some time soon. I think it should be considered by the Executive Board.

PRESIDENT STANSBURY: Okay. I don't think we should try to do it here because while we have considered several possibilities I couldn't tell you relative cost or anything of the sort.

MR. BICKFORD: One other announcement. The Executive Board unanimously appointed Wade Norton as Vice

President of COMMON. His primary responsibility will be to have the division managers report directly to him. (Applause)

MR. MAUDLIN: I went up to the headquarters room to get some Xerox copies made a little earlier today and I noticed what I think is an abuse of a privilege. I was there while sixty copies of an in-excess-of-forty-page document was being created and a hundred copies of a nineteen or twenty page document was being created. I don't think that's what the Xerox machine is for. I think when people prepare papers that a part of the presenting of the paper is to think about it before you get to the meeting and reproduce it before you get to the meeting. I'm sure it's cheaper.

MR. SCHILDER: Chuck, I agree with you. However, there were special circumstances there. The gentleman who asked for the sixty copies came with forty copies.

MEMBER: The 19-page document is mine. I cut it down from fifty pages on request. An 1130 installation has to have a lot of money to have a Xerox copier with it. If IBM can't get it collated -- they say they have a problem getting it collated -- we're going to go up and collate it ourselves. We aren't trying to abuse any

privilege that we're given.

PRESIDENT STANSBURY: As a personal opinion, I agree with Mike and tend to disagree with Chuck. I do not think it was an abuse.

Okay, gentlemen, I move we quit.

(Whereupon, the meeting was adjourned at 7:10 p.m.

AMTRAN FOR THE IBM 1130 WITH 8k OF CORE

Presented at

PHILADELPHIA COMMON MEETING

September 8 - 11, 1968

By

T. J. Buntyn

COMPUTER SCIENCES DIVISION
RESEARCH LABORATORIES
BROWN ENGINEERING, A TELEDYNE COMPANY
HUNTSVILLE, ALABAMA

INTRODUCTION

Automatic Mathematical Translator (AMTRAN) is a conversational mode, mathematically oriented language developed to assist in solving a wide variety of mathematical, statistical, and engineering problems with a minimum of programming effort by the user. The AMTRAN system, consisting of the language and special terminals with push-button input and graphical output via storage scopes, was conceived by Dr. Robert Seitz of NASA, Marshall Space Flight Center, Huntsville, Alabama. In 1965-1966, Dr. Seitz developed and implemented the AMTRAN system for the IBM 1620.

Brown Engineering, a Teledyne Company, has developed an abbreviated version of AMTRAN for the IBM 1130 under NASA contract NAS8-20415. Although the system is designed to operate with special terminals, it is felt that 1130 AMTRAN can be of considerable use when operated from the console keyboard with the usual I/O devices.

The 1130 version of AMTRAN is written primarily in FORTRAN and operates under the disk monitor system. AMTRAN provides the following features:

- A declaration free working environment
- A set of fundamental operators with a well defined syntax
- The capability for array arithmetic
- The ability to construct, store, and recall new operators denoted as console programs
- Edit capabilities to allow easy modification of previously defined console programs
- Dynamic core allocation for storage of data and console programs
- Choice of operation in either an immediate execution mode or a suppressed mode.

SOFTWARE DESIGN

The AMTRAN software is designed around the operators presented in Appendix A. A principal objective in designing the AMTRAN software for the IBM 1130 has been to optimally divide core between the system software and the user storage areas, in order to maintain an effective balance between execution speed and core available to the user. To meet this objective the program has been modularized into logical core loads so that only the module or modules necessary for the completion of a specific task need be in core at any particular time during execution of the system. The modules of the system may be grouped into the classes: system contol, incremental compiler, execution, and utility.

SYSTEM CONTROL

The primary functions of the control phase of the program are to initialize the system and to perform the bookkeeping necessary to indicate which portions of the system are needed in core and what work areas are available. The control phase also guides the execution of the program through the various tasks required by each statement.

INCREMENTAL TRANSLATION

Scanning

This process begins with the input of a statement. The first step of the process is to convert each character from the acceptable set {0 1 2 3 4 5 6 7 8 9 space A B C D E F G H I J K L M N O P Q R S T U V W X Y Z * / + - ( ) . , & % # $ } to a corresponding integer 0 to 48. Next the elements of the input statement are recognized and replaced by a three digit integer which is used to classify the various types of elements. This is accomplished in the following manner:

- Console program names are replaced by a number in the range 101 to 110 assigned on the order of first occurrence. The console program name is stored in the called programs section of the program construction area (see Appendix B).

- Delimiters and executable operators are replaced by predetermined numbers in the range 201 through 274.

- Numeric constants are converted to floating point and stored in the constants section of the program construction area. They are replaced by a reference number in the range 301 to 354 assigned on the order of first occurrence.

- User defined variable names are replaced by numbers from 401 to 429 assigned on the order of first occurrence. Variable names are temporarily maintained in core for continuity between statements.

The scanner performs extensive syntax checks as it moves through the source statement. In addition to the conversion and checking, system delimiters are inserted and special formating is performed for several of the operators. The system operators RESET, SUPPRESS, EXECUTE, LIST, NAME, EDIT, DELETE, and SAVE are recognized and the system takes immediate action on these.

Parsing

Upon completion of a successful scan the resultant string is released to the parser. Parsing is the process by which the order of execution is determined. The priority of operations is as follows:

1. operations within parenthesis
2. functions (such as SIN, EXP, etc.)
3. exponentiation
4. concatenation
5. multiplication and division
6. addition and subtraction
7. replacement (=).

Parsing is carried out by working with the input string, a delimiter list, and an output stack. The procedure works as follows. When the input symbol is a:

- Variable - place the variable in the output stack and examine the next input symbol.

- Operator - examine the last symbol in the delimiter list and if the result is

  a. an operator of lower priority

  b. a left parenthesis

  c. the delimiter list is empty

  then place the symbol in the delimiter list and examine the next input symbol.

  When none of the above conditions are satisfied, transfer the last symbol from the delimiter list to the output stack and repeat the process until a, b, or c is satisfied.

- Left parenthesis - place in the delimiter list and examine the next input symbol.

- Right parenthesis - transfer the symbols from the delimiter list (last symbol entered first) to the output stack until a left parenthesis is encountered. Delete the left parenthesis and continue to the next input symbol.

- Comma - transfer the contents of the delimiter list (last symbol entered first) to the output stack until a left parenthesis is reached or the delimiter list is empty, then continue to the next input symbol.

- End of statement - transfer the contents of the delimiter list (last symbol entered first) to the output stack. This will end the parsing procedure with the results in output stack.

Coding

The output stack from the parser is released to the coder. The coder collaspes the stack by cycling through it, replacing operators and operands, and generating a sequence of LOAD, OPERATE, STORE, or OPERATE, STORE commands. This process continues until the entire stack has been transformed into a macro language. In general each macro instruction requires one 16-bit word. The leading 7 bits specify the operator and the remainder of the word specifies the variable.

EXECUTION

Interpretation

This is the process of accessing a macro instruction, classifying it and branching to the appropriate routine that performs the prescribed operation. There are two major routines that handle the majority of actual calculations.

Data Access and Execution

Data access is the process of determining if the variable is defined, its size, its location, and its compatability with the operation requesting it. If the variable is defined and compatible, execution is performed; otherwise, an error message is typed out.

Storage Allocation

Storage allocation for data is a continuous process throughout execution of the macro language instructions. Storage for user variables, temporary results, and the system accumulator is provided in one data area. Storage for any data type is allocated only as needed and in the exact amounts required during execution, is redimensioned as necessary, and, in the case of temporary storage, is made available for further use as soon as possible.

Access to free storage is maintained by a pointer scheme linking together available blocks of contiguous storage locations in the data area. When new space is required for the storage of data, the pointers are scanned until the first block containing at least the space requested is encountered. The exact number of contiguous data words required is then removed from the free storage linkage. If a block of the appropriate size is not encountered, an additional scan is made through the pointers, replacing the sequential linkage to contiguous blocks with a single pointer link. If this process does not provide a large enough block, all data currently stored in the data area is packed, providing one large block of available storage.

When a block of data is returned to free storage, the pointers are adjusted and a pointer is added to include the additional block in the free storage linkage.

UTILITY

The utility portion of the system consists of several subroutines which provide the services of storing, listing, editing, or deleting of console programs, and the deletion or retention of user defined variables. The services in this catagory are initiated by the operators found under SYSTEM OPERATORS in Appendix A.

SAMPLE PROBLEMS

Presented here are three sample problems selected to illustrate some of the capabilities of the system.

EXAMPLE 1.

```
ENTER PROGRAM
1.    X=ARRAY -3,-1,8, Y=ARRAY 0,PI/3,2, N=0.
2.    REPEAT 3, P=Y SUB N,
      Z=3.28+X*X*X*COS P+1/COS P * EXP(-X/(X+7.45)),
      WRITE(P/DEGREES),TAB2(X,Z),N=N+1.

      0.0000
     -3.0000     -21.7576
     -2.7500     -15.7217
     -2.5000     -10.6879
     -2.2500      -6.5692
     -2.0000      -3.2766
     -1.7500      -0.7200
     -1.5000       1.1917
     -1.2500       2.5502
     -1.0000       3.4477


     30.0000
     -3.0000     -17.8367
     -2.7500     -12.6577
     -2.5000      -8.3382
     -2.2500      -4.8047
     -2.0000      -1.9816
     -1.7500       0.2083
     -1.5000       1.8429
     -1.2500       3.0012
     -1.0000       3.7623


     60.0000
     -3.0000      -6.2952
     -2.7500      -3.5281
     -2.5000      -1.2184
     -2.2500       0.6675
     -2.0000       2.1667
     -1.7500       3.3190
     -1.5000       4.1660
     -1.2500       4.7502
     -1.0000       5.1154

3.
```

EXAMPLE 2.

THIS EXAMPLE PRESENTS AN OPERATOR PROGRAMMED IN AMTRAN TO PERFORM THE TRANSFORMATION DUE TO BILHARZ ON THE COEFFICIENTS OF A COMPLEX POLYNOMIAL. THIS PORCESS IS USEFUL IN STUDYING THE STABILITY PROPERTIES OF A SYSTEM OF LINEAR DIFFERENTIAL EQUATIONS*.

```
 1. X=Q,Y=B,M=2 INTERVALS X-2,A=0,L=0.
 2. TYPEOUT BILHARZ MATRIX OF COEFFICIENTS..
 3. WRITE X, WRITE Y,A SUB 0=Y SUB 0,N=1.
 4. F=-X SUB 0/Y SUB 0,Z=LEF(F*Y+X),WRITE Z,
    X=Y,Y=Z,A SUB N=Y SUB 0.
 5. IF N LE M THEN N=N+1,GO TO 4 ELSE F=0,K=2.
 6. N=0,P=1.
 7. REPEAT K,P=P*A SUB N,N=N+1.
 8. L SUB F=P,K=K+2,F=F+1.
 9. IF F+1 LE INTERVALS Y THEN GO TO 6.
10. TYPEOUT ALPHA N FOR N=1,2,....
11. WRITE A.
12. TYPEOUT PRODUCT OF ALPHA N..
13. WRITE L.
14. NAME BILHZ.
```

```
 1. Y=X, Y SUB 0=0, Y=SHIFT(-1,Y).
 2. NAME LEF.
```

*FOR THEORY RELATING TO THIS EXAMPLE, SEE "STABILITY THEOREMS FOR LINEAR MOTIONS" BY S. H. LEHNIGK, PRENTICE HALL, 1966.

GIVEN A COMPLEX POLYNOMIAL OF THE FORM

$$f(s) = \sum_{n=0}^{p} (a_n + b_n i)\, s^{p-n}$$

TO USE THE BILHZ OPERATOR WRITE OUT THE COEFFICIENTS AS FOLLOWS:

$$A1 = -b_0,\ a_1,\ b_2,\ -a_3,\ -b_4,\ \ldots$$
$$A2 = a_0,\ b_1,\ -a_2,\ -b_3,\ a_4,\ \ldots.$$

THEN ENTER BILHZ (A1, A2). HENCE FOR THE POLYNOMIAL

$$f(s) = 2(1 - 3i)\, s^4 + (9 - 13i)\, s^3 + 3(4 - 3i)\, s^2 + 2(3 - i)\, s + 1.$$

```
ENTER PROGRAM
  1. LET X=6,9,-9,-6,0.
  2. LET Y=2,-13,-12,2,1.
  3. BILHZ(X,Y).

     BILHARZ MATRIX OF COEFFICIENTS.
       6.0000       9.0000      -9.0000      -6.0000       0.0000
       2.0000     -13.0000     -12.0000       2.0000       1.0000
      48.0000      27.0000     -12.0000      -3.0000       0.0000
     -14.1250     -11.5000       2.1250       1.0000       0.0000
     -12.0796      -4.7788       0.3982       0.0000       0.0000
      -5.9121       1.6593       1.0000       0.0000       0.0000
      -8.1691      -1.6450       0.0000       0.0000       0.0000
       2.8498       1.0000       0.0000       0.0000       0.0000
       1.2216       0.0000       0.0000       0.0000       0.0000

     ALPHA N FOR N=1,2,...
       2.0000      48.0000     -14.1250     -12.0796      -5.9121      -8.1691       2.8498       1.2216

     PRODUCT OF ALPHA N.
    0.96000E 02   0.16379E 05   0.79109E 06   0.27539E 07
  4.
```

EXAMPLE 3.

THIS EXAMPLE WAS WRITTEN TO STUDY THE SMOOTHING EFFECTS OF THE PARAMETER σ ON THE PROBABILITY DENSITY FUNCTION

$$f(x) = \frac{1}{\sigma(2\pi)^{1/2}} \times \frac{1}{m} \sum_{i=1}^{m} \exp\left[- \frac{(x - s_i)^2}{2\sigma^2} \right]$$

```
1. J=INTERVALS S+1,K=INTERVALS GG+1,H=J*SQRT(2PI),L=0.
2. X=ARRAY -3,3,100,F=0.
3. Y=0*X,G=GG SUB L,N=0.
4. REPEAT J,Y=Y+EXP-((X-S SUB N)**2/(2G*G)),N=N+1.
5. Y=Y/(H*G),B=MAX Y.
6. IF F LT B THEN F=B.
7. A=.5&0&6.5&F&2, PLOTS(X,Y,A),L=L+1.
8. IF L LT K THEN GO TO 3.
9. NAME PTRN.
```

```
ENTER PROGRAM
1.    LET X=-.96,-.88,-.86,-.40,0,.23,.44,.88 .
2.    LET SIGMA=0.1,0.2,0.5,1.
3.    PTRN(X,SIGMA).
```

NOTE: OUTPUT FOR THIS EXAMPLE WAS DISPLAYED ON THE SCOPE. SLIDES OF THE OUTPUT WERE SHOWN AT THE PRESENTATION.

SYSTEM CONSTRAINTS

In the abbreviated version of AMTRAN currently operational on the IBM 1130 with 8k of core, 1,208 words are reserved for user data storage. Hence, a user is allowed to work with up to 604 floating point numbers. Up to 95 console programs may be defined and stored on the disk, and 860 words are reserved for internal storage of console programs. A block of 450 words is reserved for the keyboard execution and program construction area. An AMTRAN statement may consist of up to 209 characters. A total of 89 variables are allowed to be active at any one time and imbedding of console programs through ten levels is allowed.

When constructing a console program or executing at the keyboard, up to 45 statements are allowed per program. However, the total length of the source must not exceed 1,140 characters and the macro language must not exceed 244 words. Each console program may contain up to 29 user defined variables, 54 distinct constants (excluding the integers 0 through 10), and may call up to 10 other programs.

CONCLUSION

AMTRAN (the IBM 1130 console keyboard version) has been available on a limited basis to several employees in Brown Engineering Research Laboratories since June 1968. They have found it to be a reliable and effective tool in evaluating and studying the behavior of functions and in performing user controlled iterations. Feedback from these users has been invaluable in finalizing the 8k AMTRAN system.

APPENDIX A

SUMMARY OF AMTRAN OPERATORS FOR THE IBM 1130

SUMMARY OF AMTRAN OPERATORS

BINARY OPERATIONS

Symbol	Operation	Operands	Results
* or implied	Multiplication	All arithmetic operations listed can be performed between constants, variables, or expressions in the following forms: 1. Two scalars 2. A scalar and an array with n elements 3. An array with n elements and a scalar 4. Two arrays with the same number of entries NOTE: An error condition occurs when an arithmetic operation is attempted between arrays of different lengths.	The results of array arithmetic are the product, quotient, etc. between corresponding elements of the operands. 1. A scalar 2. An array with n elements 3. An array with n elements 4. An array with n elements
/	Division		
+	Addition		
-	Subtraction		
** or POW	Exponentiation		
&	Concatenate	Scalar constant(s), variable(s), or expression(s) and/or array variable(s) or expression(s).	An array with the number of elements equal to the sum of the number of elements in each argument.

MATHEMATICAL FUNCTIONS

Symbol	Function Performed	No. of Arguments	Type[1]	Results[2]
SIN	Trigonometric sine	1	Scalar or array (in radians)	Scalar or array the same size as argument
COS	Trigonometric cosine	1	Scalar or array (in radians)	" " " "
LN	Natural logarithm	1	Scalar or array	" " " "
EXP	Argument power of e	1	Scalar or array	" " " "
SQRT	Square root	1	Scalar or array	" " " "
ATAN	Arctangent	1	Scalar or array	" " " " (in radians)
ABS	Absolute value	1	Scalar or array	" " " "
TANH	Hyperbolic tangent	1	Scalar or array	" " " "
SQ	Quantity squared (used on right of argument)	1	Scalar or array	Scalar or array the same size as argument

[1] The arguments for all these functions may be either constants, variables, or expressions.

[2] The results in the case of array arithmetic are the results of the function performed on each entry of the argument array.

SPECIAL FUNCTIONS FOR ARRAY MANIPULATION

Symbol	Function Performed	No. of Arguments	Type	Results
ARRAY	Generation of an array	3	Scalar constants, variables or expressions	An array with the first element equal to the first argument, the last element equal to the second argument, and the number of equal sized increments specified by the third argument.
LET[1]	Generation of an array	n	Scalar constants, variables, and/or expressions	An array with the n arguments as elements.
SUB	Subscript modification	1	Scalar constant, variable, or expression	Scalar
SUB-THRU	Subscript modification	2	Scalar constants, variables, or expressions	Array
SUB LAST	Subscript modification	0		Scalar, last element of an array
MIN	Selection of the minimum element of an array	1	Array variable or expression	Scalar
MAX	Selection of the maximum element of an array	1	Array variable or expression	Scalar
SUM	Computation of the running summation of the elements in an array	1	Array variable or expression	Array of the same dimension as the argument
SUMF	Computation of the total sum of the elements of an array	1	Array variable or expression	Scalar
SHIFT	Cyclic shift of the elements in an array	2	Scalar constant, variable, or expression and array variable or expression	Array the same dimension as the second argument with elements shifted the number of places and direction specified by the first argument.

[1] The LET statement takes the form: LET variable = argument(s).

LOGICAL OPERATORS AND TRANSFER STATEMENT

Symbol	Description
IF[1]	Begins the IF statement used to control execution based on the relationship between two scalar quantities.
EQ, NE, LT, GT, LE GE	Relations available to the IF statement. These represent =, ≠, <, >, ≤, and ≥ respectively.
THEN	Denotes the beginning of the THEN clause associated with an IF statement.
ELSE	Denotes the beginning of the ELSE clause associated with an IF statement. (The ELSE clause is optional.)
REPEAT	Causes the succeeding substatements of the line to be executed the number of times specified by the scalar argument following the repeat, e.g. Repeat n, ..., ..., ...
GO TO or GOTO	Causes execution to be transferred to the statement number indicated by the numeric argument following the GO TO. In the EXECUTE mode, the GO TO may only refer to a previously defined statement.

[1]NOTE: The general form of the IF statement is:

IF Q1 relation Q2 THEN ..., ..., ... ELSE ...,...

When the relation is satisfied the THEN clause is executed and the ELSE clause skipped. When the relation fails the THEN clause is skipped and the ELSE clause executed.

INPUT AND OUTPUT OPERATORS

Symbol	Description
SET	Allows numeric data to be read in from the console printer (sense switch 15 off) or the card reader (sense switch 15 on). A string of numeric constants in free format is read into memory and associated with the variable name specified by the argument. The variable assumes the dimension of the input string. The numbers may be either integers or decimal numbers in either fixed or floating point format and are separated by commas or blanks. The input string is terminated by two consecutive slashes (//).
WRITE	Causes the value(s) of the argument to be printed on the console printer (sense switch 0 off) or on the 1132 printer (sense switch 0 on) in fixed point format (sense switch 1 off) or floating point format (sense switch 1 on).
PUNCH	Causes the value(s) of the argument to be punched on cards in fixed point format and to be printed on the console printer (sense switch 0 off) or on the 1132 printer (sense switch 0 on).
TYPEOUT	Causes the alphanumeric information following the word TYPEOUT to be printed on the console printer (sense switch 0 off) or on the 1132 printer (sense switch 0 on).
LIST	Causes the source statements of the specified console program or a description of the specified intrinsic operator to be printed on the console printer (sense switch 0 off) or on the 1132 printer (sense switch 0 on). When the argument is a console program name and sense switch 12 is on, the console program will also be punched on cards.
LIST ALL	Causes the names of all console programs defined in the system to be printed on the console printer.
LIST INTRINSICS	Causes the names of the AMTRAN operators to be printed on the console printer.
CARD	Causes the source statements of a console program to be read in from cards.

SPECIAL SYSTEM OPERATORS

Symbol	Description
RESET	Causes the system to be reinitialized in the EXECUTE mode. All variables are deleted and statement numbers begin at 1. (Console programs are unharmed.)
SUPPRESS	Causes the system to be reset in the SUPPRESS mode.
EXECUTE	Same as RESET
DELETE	Reset with deletion of specified variable(s) and/or console program(s). DELETE can be used only in the EXECUTE mode.
SAVE	Reset with specified variable(s) saved. SAVE can be used only in the EXECUTE mode.
NAME	Causes the current sequence of statements to be named and stored as a console program under the name specified by the argument.
EXIT	Used to create multiple exit points from a console program. EXIT can be used only in the SUPPRESS mode.
INPUT	Causes an input parameter to be required in a console program. INPUT can be used only in the SUPPRESS mode.
EDIT	Allows the user to modify specified statements of a previously named console program.
PAUSE	Causes a halt of the current execution. Execution is continued by pressing the PROGRAM START button on the console keyboard.
$	Causes deletion of the current keyboard line.
$$	Causes deletion of the current keyboard statement.
#	Causes deletion of the preceding character.

SYSTEM CONSTANTS

Symbol	Description
PI	3.14159
DEGREES	0.017453 (conversion factor for converting degrees to radians)

APPENDIX B

PROGRAM CONSTRUCTION AREA AND DATA TABLES

I. KEYBOARD EXECUTION AND PROGRAM CONSTRUCTION (450 WORDS)

HEADER (seven words)	VARIABLE REFERENCE (252) CONSTANT REFERENCE (302) PROGRAM REFERENCE (410) NUMBER OF PARAMETERS ACTIVE PROGRAM REFERENCE CALLING PROGRAM RETURN LOCATION
EXECUTABLE MACRO INSTRUCTIONS	UP TO 244 WORDS FOR MACRO INSTRUCTIONS
VARIABLE POINTERS	ONE WORD PER VARIABLE UP TO 50 VARIABLES
CONSTANTS	TWO WORDS PER CONSTANT UP TO 54 CONSTANTS
PROGRAMS CALLED	FOUR WORDS PER PROGRAM UP TO 10 PROGRAMS

II. DATA INDEX (90 entries)

LOCATION	LENGTH

III. DATA STORAGE

1, 280 WORDS

PURCHASING

PURCHASE REQUISITION

QUOTE A	P.O. B	BLANKET P.O. C	← ①		
BLANKET REL. NO.	CHANGE NO.	CANCEL			

P. O. NO.	REQ. NO.	DATE	APPROPRIATION	P. O. SETS
②	③	④	⑤	⑥

CONTRACT NO.	PRIORITY	1st PROD.	ORIG. INIT.	DEL. TO DEPT.	AREA	PUR PER B.P	BLUEPRINT 1. ATT. 2. SENT	TOOL. INST. 1. ATT. 2. SENT	FOLLOW UP CODE	P.O. DATE	QUOTE DATE	BID NUMBER	QUOTE RETURN	BLANKET ORDER LIMIT	BLANKET ITEM LIMIT
⑦	⑧	⑨	⑩	⑪	⑫	⑬	⑭	⑮	⑯	⑰	⑱	⑲	⑳	㉑	㉒

CONFIRMATION CODE 1. PHONE 3. TELEGRAPH 2. LETTER 4. VERBAL	CONFIRMATION WITH	CON. DATE	SHIP VIA CODE A. TRUCK D. RWY. EXP. G. P.P. K. AIRLINE AIR FRT. B. RAIL E. AIR EXP. H. P.P. SPEC. DEL. M. VEN. OPT. C. BUS F. AIR P.F. J. AIR P.P. S.D. N. U.P.S.	F.O.B. 1. OUR PLANT 2. SHIP POINT 3. OTHER	BUYER	L/AREA	PROCURE	TERMS	SMALL BUS.	TYPE CONTRACT	F	F. O. B. CITY
㉓	㉔	㉕	㉖	㉗	㉘	㉙	㉚	㉛	㉜	㉝		㉞

F.O.B. STATE	ROUTING DESCRIPTION (USE CODE IF AVAILABLE WITH PREFIX R-)
	㉟

H	VENDOR	1. ㊱	2.	H 3.	4.
	NUMBER				
	ATTN.				
H	VENDOR	5.	6.	H 7.	8.
	NUMBER				
	ATTN.				

R ±	ITEM	LINE	QUANTITY	U/M	PART NUMBER	REV. LT.R.	MFG. CODE		COM. CL.	UNIT PRICE	DISP. QTY.	TARGET PRICE
㊲	㊳	㊴	㊵	㊶	㊷	㊸	㊹	㊺	㊻	㊼	㊽	㊾

ACTION CODES:
A. QUOTE NOTES
B. PUR. ORD. NOTES
C. SET UP CHG. PER ITEM
D. SET UP CHG. PER SHIP/LOT
F. MIN. CHG. PER ITEM
G. MIN. CHG. PER SHIP/LOT
J. TOOL. CHG. PER ITEM
K. TOOL. CHG. PER SHIP/LOT
M. TEST RPT. CHG. PER ITEM
N. TEST RPT. CHG. PER SHIP/LOT
P. QUAN. INC./DUE TO MIN. BUY

W ±	ITEM	LINE	ACTION CODE	BUY. NOTES	SHOP. ORDER	CHANGE NOTICE	ACCT. DEPT. PROD.	QUAN. REQUIRED	DATE REQUIRED	QUAN. PROM.	DATE PROM.	SET./MIN. CHG	TOOLING CHG. OR TEST REPORT	N BID QUOTE QUANTITIES	
㊿	(51)	(52)	(53)	(54)	(55)	(56)	(57)	(58)	(59)	(60)	(61)	(62)	(63)	1 (64)	2
														3	4
														5	6
														7	

Z		(65)	(66)	VARIABLE DATA (67)

APPROVED	APPROPRIATION	CODING	SUPERVISOR	DEPT. HEAD	STAFF	PURCHASING AGENT	SIGNED BUYER	SIGNED ORIGINATOR
NAME	(68)							
DATE								

700-1B

"RELIC" - REMOTE JOB PROCESSING ON A 1620

by

RICHARD F. GABRIEL

GEORGE J. GERMANN

COMPUTER CENTER
SETON HALL UNIVERSITY
SOUTH ORANGE, NEW JERSEY

PRESENTED AT THE

COMMON MEETING

THE USERS GROUP FOR SMALL IBM COMPUTERS

PHILADELPHIA, PENNSYLVANIA

SEPTEMBER 9, 1968

RELIC-PART I

SYSTEM CONSIDERATIONS

INTRODUCTION

RELIC, though an acronym, does seem to be an appropriate term for an IBM 1620 computer in this day of time-sharing, real-time, on-line computer systems. RELIC (REmote Location Input/Output Capability) is the result of our effort to devise a low cost terminal system which will provide access to our computing facilities from locations remote from the Computer Center. A small third generation computer, interfaced with the IBM 1620, serves as a multiplexing and buffering device. The Interface will provide information transfer both to and from the 1620 at the core speeds of the IBM 1620. The system currently has a one-way interrupt capability; the ability to interrupt the 1620 from a remote location is still under consideration. Three teletypewriters will be available during the Fall Semester and requests for additional stations, one of them a scope, have been received.

In Part II of this paper Mr. Germann will outline for you the extent to which terminal support is being provided by the 1620 and some details of the hardware-software mix that was developed to achieve this. It is my purpose to explain to you why the project was undertaken.

BACKGROUND

Seton Hall University is a privately owned institution some thirty miles west of New York City in South Orange, New Jersey; it has an enrollment of approximately 8,500 students distributed among five schools the largest of which are the College of Arts and Sciences, the School of Business, and the School of Education. Appropriate degrees are offered at both the graduate and undergraduate level.

In March, 1968 an IBM 1620 20K card system was installed and was subsequently augmented by a disk and an on-line printer. Although the Center was developed to provide support for faculty research, student educational programs, and administrative service needs, substantial funding from NIH during the early years of the Center's existence directed much of the Center's activity toward support for biomedical research most of which centered around the College of Medicine and Dentistry which was then a part of the University. Since the separation of the Medical School from the University and the termination of the NIH grant, both of which occurred in 1966, the University has funded the Center without outside support.

Thus, there were three very significant factors before us at the time when everyone was jumping on the 360 bandwagon.

1. We owned an IBM 1620
2. Money was hard to find
3. With the phasing out of the biomedical activity, current use of the facility coupled with reasonable projections for future use did not justify the acquisition of faster equipment.

The first two points relate essentially to funding difficulties which were not insuperable: the third point was far more critical. Justification for more extensive facilities would require more extensive use. However, analysis of our overall operation revealed a built-in self-limiting factor - a state of equilibrium - resulting from space constraints and a continuing commitment to service three major areas education, research, and administration. The possibility of curtailing one of these areas of activity to enhance usage within the others was considered and rejected because the obvious one to curtail, student usage, is precisely the one that has the greatest potential for future growth at the University. Thus, student use was a major factor in the decision to implement RELIC.

III. JUSTIFICATION FOR RELIC

RELIC was devised to break the state of equilibrium imposed by the space constraints and the commitment to continue service in all three areas. To clarify this point a brief description

of some of our activity follows. Four full-time people currently service the administrative data processing needs which are unit record oriented. Computer use in support of this activity rarely exceed 15% of total computer use in any given month. However, the large volume of cards requires storage space and work staging areas. Severe space constraints at the University preclude expansion of the Center in the immediate future. Consequently students coming in to use the facilities are competing with the data processing staff for the limited work space, card storage, and the like.

The installation of the disk file and Monitor System enabled us to attract a new breed of student and faculty whom we call "the casual user". The casual user may be thought of as one who writes no programs of his own but uses the computer to do his homework or research. Our staff has written and stored on the disk a number of programs which enable users to submit data to the computer with appropriate monitor control cards and receive in return functions of input data which previously would have required hours of tedious desk calculator work. Students tend to gang up at the Computer Center just before class deadlines and generally obstruct the work flow at the Center in ways far too numerous to catalog at this time.

We are committed to the premise that faculty and students should have access to our facilities; the best answer is enough space to separate two areas of activity but such space will not be forthcoming before 1970. Terminals, providing remote access was an obvious alternative.

Suffice it to say that the prices quoted to us on hardware-software systems designed to support terminals were prohibitive. Furthermore, even if cost were not significant, the speeds of these machines would find us with an idle computer most of the time - except possibly for system overhead! And the operating systems which were presented to support these terminals had many features for which we had no real need. Under these circumstances it became clear that our objective was simply to extend the life of the 1620 until 1970 when new quarters would be available and to augment use sufficiently to justify new equipment for the new space. This in essence was the motivation for RELIC.

SYSTEM CONSIDERATIONS

Our staff is most competent but limited numerically by the same space considerations mentioned above. Consequently our first decision was to operate with the current 1620 Monitor System and to minimize changes necessary to this system. The paper

tape channel was not being used and of course is available under the Monitor System. Thus communication between terminals and the 1620 could use the paper tape I/O features of the 1620. The "casual user" would still enter the system with Monitor control cards and data in the forms normally presented at the console. It was also considered quite desirable, if not essential, to be able to compile and run rather short FORTRAN programs of the type frequently required by students. This will be discussed in some detail in Part II so I shall simply note this fact at this time.

Teletypewriters are not too expensive and would serve our immediate needs. We contemplated installing three terminals and therefore needed a buffering and multiplexing device with interrupt capability which could be dummied up to look like a paper tape reader and punch to the IBM 1620. Several small third generation computers selling for well under $10,000 met these specifications, and there was a prospect for some grant funding to support this approach.

Of course none of these devices "talk" to the 1620. An interface was necessary and arrangements for its design and construction became a very major problem which delayed the installation of the whole system to this late date. The interface difficulties, in chronological order may be described briefly as follows:

1. IBM policy with regard to the 1620 precluded any effort to provide hardware support for RELIC.

We require continuing IBM maintenance on the 1620 and the prospect of non-IBM personnel rewiring the 1620 gave us as much concern as it probably did our customer engineers. At this point we decided to forego the specification for a hard interrupt in the 1620 and further specified that any interfacing device be designed to plug in and out of the IBM 1620 paper tape adapter.

2. The small computer manufacturers were unwilling to put effort in the development of the 1620 interface. 360 Yes!, 1620 No! Casual estimates, and in one instance a reasonably firm quote, were quite high.

3. Reasonable bids were submitted by several individuals acting as one-man firms but the University became concerned about fixing responsibility for the operation of the entire system. The use of grant funds for the project was a significant factor in this concern.

We were fortunate enough to contact a firm which already had some experience with one of the small computers that happened to be our first choice and furthermore had a contractual affiliation with the manufacturer. This firm agreed to purchase the computer configuration for us, design the interface, and to accept responsibility for providing a complete working system. In retrospect, with the clear vision of hind-sight, it is evident that the design and construction of the interface was in itself no problem. Every delay can be traced ultimately to difficulties in communication and information transfer among people. The whole job could readily have been completed within three months which is the lead-time for delivery on the small computer.

To conclude this part of the presentation I should like to submit several comments or observations either as statements or short paragraphs and I shall gladly amplify these points should any questions arise.

1. Dealing with original equipment manufacturers, although frustrating and discouraging at times, revealed a new world of great promise which merits further exploration. Adequate communication - vocabulary - and the ability to write

proper specifications are attributes to be cultivated. With this market open to us we expect to be able to shape the growth and development of our computing facilities to meet or own needs.

2. The availability of a fast third generation computer with an interrupt capability will enable our staff and computer science students to obtain first-hand experience in programming for time-sharing and real-time systems.

3. The hardware-software complex developed to support the terminal system, with the exception of the interface currently to the 1620, will be independent of the main frame; this means in essence that we can unplug the 1620 and replace it with any computing facility with rather minor modifications to the terminal system. Furthermore our staff efforts will be concentrated on the development of the terminal system at a time when system requirements on the 1620 are light. The terminal system should be reasonably well established at the time we undertake the agonies of transition from 1620 to XYZ.

4. The prospect of terminals on Campus has stimulated a substantial amount of faculty interest in the potential use of computers in a wide spectrum of applications much of which is related to students and class activity.

I should like to be able to say at this time that our system has been working successfully for the past seventeen months, that every one at Seton Hall from the President to our freshmen dropouts make extensive use of our computing facilities, that all terminals are being used simultaneously, and that no one has ever waited longer than thirteen seconds for the answer to his problem, but this would only be partly true. Since obviously I have exhausted my veracity and my time I shall let Mr. Germann present you with the facts.

"RELIC" - REMOTE JOB PROCESSING ON A 1620

PART TWO

IMPLEMENTATION

AUTHOR : GEORGE J. GERMANN

ABSTRACT: Part two of this paper will show how the IBM 1620 Monitor I System can be used to support remote terminals, describe the modifications required, and explain the implementation of the RELIC System being developed at Seton Hall University.

SYSTEM CONFIGURATION

Initially RELIC will consist of the IBM 1620 Data Processing System with a 1622 Card Read-Punch, a 1443 Line Printer, and a 1311 Disk Storage Drive, a small computer used for logic switching and buffering, and three "typewriter-like" remote terminals.

The three remote terminals will tie into the "small computer" which is coupled to the 1620 through the Paper Tape Channel of the 1620. To the small computer the 1620 will appear as another terminal. (see diagram of next page)

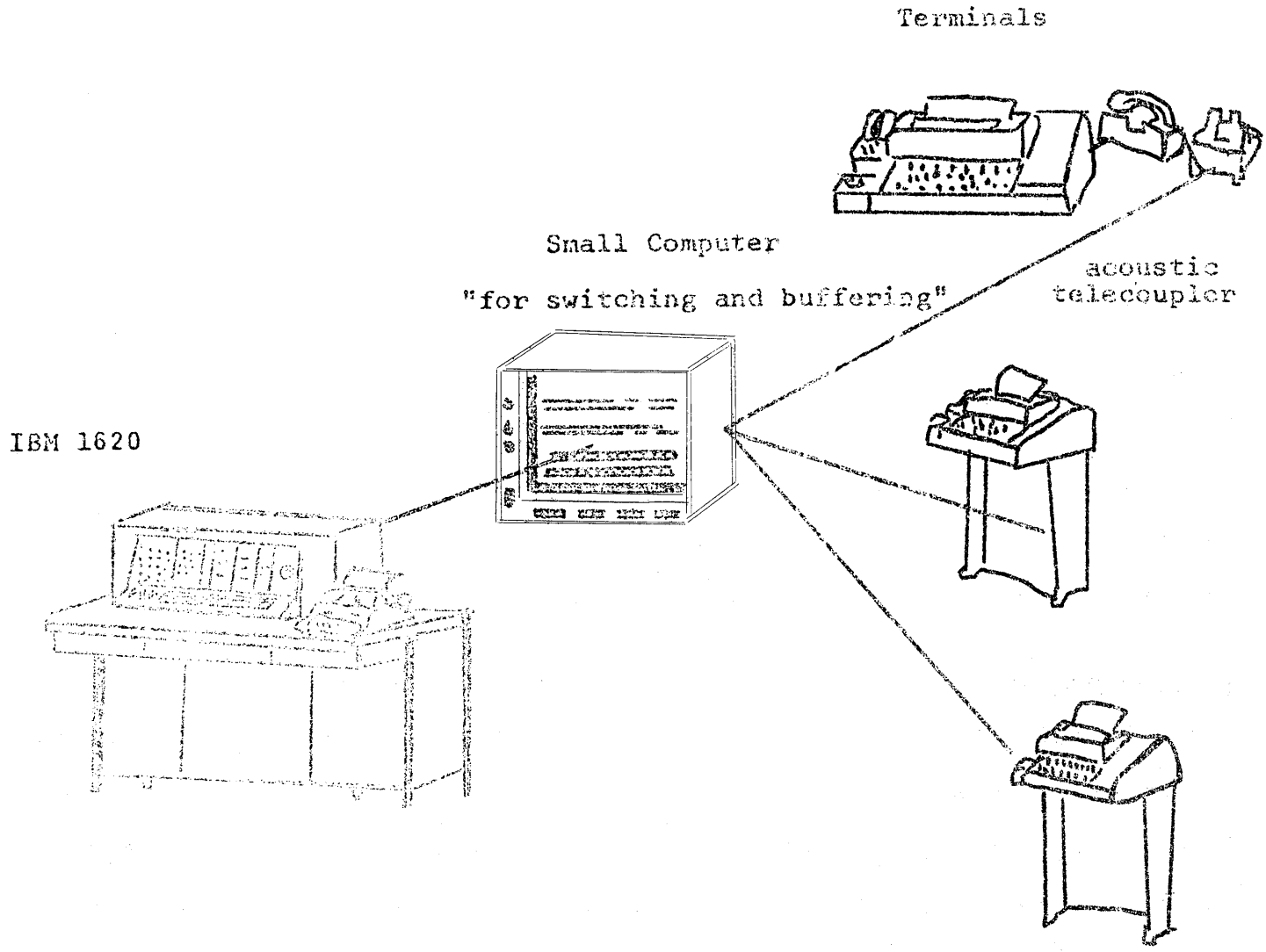
INTERFACING

Two of the terminals will be directly interfaced to the "small computer", and will be located within 1,000 feet of the computer. The third terminal will communicate to the "small computer" over our internal telephone extension lines with a data phone interface at the computer end, and an acoustic coupler at the terminal end. This will permit the third terminal to be mobile and moved where needed.

A specially designed and built interface allows communication between the 1620 and the "small computer". This interface permits data transfer through the 1620 paper tape channel at speeds much faster than the paper tape unit (1621). See Appendix A.

RELIC - REmote Location Input/output Capability

SYSTEM CONFIGURATION



COMMUNICATION AND DATA TRANSFER

There will be basically two types of communication links in the RELIC system:

1. Terminal Communication which is defined as I/O between the "small computer" and the remote terminals.
2. 1620 Communication which is Input/Output between the 1620 and the "small computer" through the 1620 paper tape channel.

TERMINAL COMMUNICATION

Data transfer between the terminals and the "small computer" will be interrupt controlled, permitting the terminals to demand service when the device itself is ready for data transfer. Both the terminals and the "small computer" will be able to demand service from each other in both the read and write modes.

1620 COMMUNICATION

The 1620 will be able to communicate only indirectly with the terminals by directly communicating with the "small computer", and only when the 1620 is ready to send or receive data. Since the 1620 does not have a interrupt capability, the "small computer" cannot raise the 1620 for data transfer until the 1620 has executed a paper tape I/O instruction. (I might add that this problem is being given serious thought and it may be alleviated through some hardware changes in the 1620 display console.) The "small computer" does have the capability to enable or disable the 1620 from interrupting in both the read or write modes.

CODE CONVERSION

The terminals being used in the RELIC system communicates in the U.S.A. Standard Code for Information Interchange (ASCII) which prohibits this code from being utilized by the 1620 until the code is converted. The forty-eight 1620 characters are converted from the ASCII codes into the 8-bit hexadecimal code acceptable to the 1620 through the data lines of the paper tape channel. (See Appendix B for the code conversion tables.)

To expedite data transfer between the 1620 and the "small computer" all code conversions are accomplished during terminal communication. This allows the 1620 to transfer data at its optimal speed. There is no time lost in converting between terminal communication, since conversion takes place while the terminal is reading the next character or typing the previous character. The terminal transfer rate is 10 characters per second.

RELIC'S SOFTWARE SYSTEM

In general, the monitor system within the "small computer" supervises all data transfer and communications to its peripheral devices. RELIC'S monitor system must read and store jobs entered from the terminals, send these jobs to the 1620 when they (both 1620 and job) are ready, receive output from the 1620, and return it to the terminals.

RELIC - MONITOR I

A. CAPABILITIES:

Access

The three terminals are able to demand service "almost" simultaneously by utilizing the interrupt controlled I/O capability of the "small computer". This permits three users at three remote terminals to enter three different jobs concurrently.

Job Storage

Our first approach in the development of the RELIC system was to proceed as simply as possible and to avoid major complications and changes to the 1620 Monitor I system. We, therefore, chose the easiest and most pragmatic method to implement this system. The 8K byte (each byte contains 8 bits) memory of the "small computer" was partitioned. One thousand and five hundred bytes (1.5K) were allocated to each of the three terminals for job storage. This would permit seventy-five FORTRAN statements (20 characters per statement) per terminal. One thousand bytes were

allocated for a common output area for data being returned from the 1620. Approximately 1K bytes were needed for the RELIC monitor system which remains resident in memory, and an additional 1.5K bytes still remain for a fourth terminal if added.

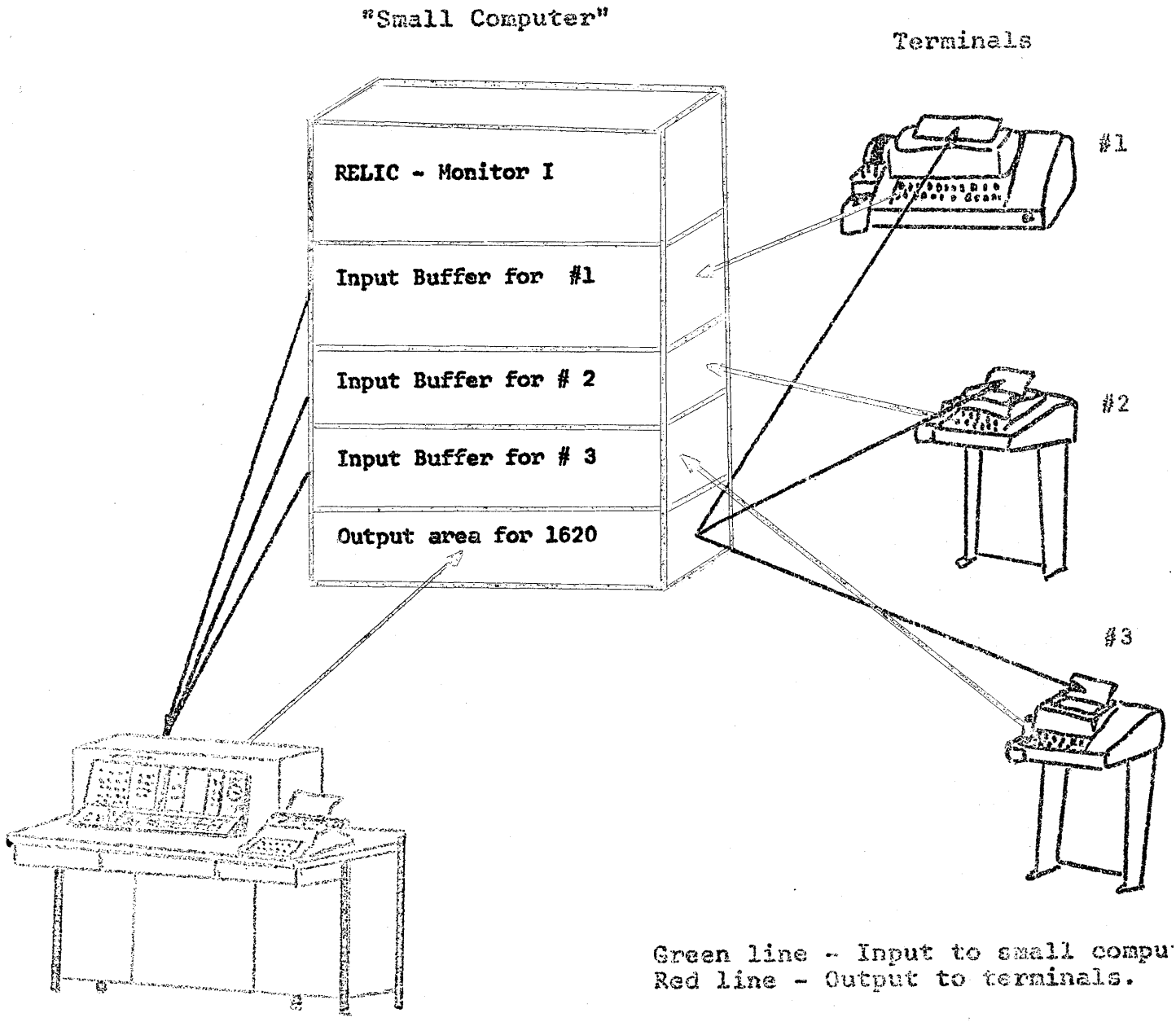
JOB ROUTING

Jobs are entered remotely from the terminals either through the keyboard or from punched paper tape. The first job to be completely read, including program and data, will be the first job sent to the 1620 for execution, when the 1620 is ready to accept it. Each job is stored in its own terminal input area. When the 1620 has read and has begun execution of the job, the two remaining terminals are still able to enter and complete their jobs. Output from the 1620 is returned to a common output area shared by all the terminals (one job at a time) within the "small computer", and then back to the "waiting" terminal. Upon completion of the job return, the 1620 will be ready to accept the next job when ready in the "small computer". (See Job Routing Diagram next page)

TURN-AROUND TIME

We expect turn-around time per job to be about one to two minutes from the beginning of 1620 execution to the outputting of results to the terminal.

REMOTE JOB ROUTING AND STORAGE



Green line - Input to small compu
Red line - Output to terminals.

IBM 1620 Data Processing System

B. TERMINAL JOB

Job Format

Jobs which are sent to the 1620 from the "small computer" follow the same format as any job entered under the IBM 1620 Monitor I system. The first card read by the 1620 from the "small computer" will be a job card, the second will be a Monitor Control Card (e.g. FORX or XEQ), followed by the program and/or data, and an END OF JOB record (≠≠≠≠) at the end of the data.

Job Execution

RELIC - Monitor I protects the 1620 by setting up the 1620 Monitor Control Cards internally in the input area for the terminal or terminals being serviced. If a FORTRAN IID program is to be executed, each FORTRAN statement is entered in free format, one line at a time, and a 1620 paper tape end of line character is placed at the end of each line when the return key is depresseed at the terminal. When the FORTRAN "END" statement is read, the message "ENTER DATA" is typed, and all the data must be entered. If the job is to be a 1620 disk-stored program execution (XEQ), then upon receipt of the program CALL NAME, the message "ENTER DATA" is typed. When all the data has been entered, the job is sent to the 1620 for execution.

The 1620 will read the job one line at a time causing an interrupt each time it is ready to read a new line. While the 1620 is executing the job, the terminal being serviced is blocked from entering any data, unless it is requested by the 1620.

Job Execution continued

Again, while the 1620 is processing each line, the "small computer" is free to accept job entry from the terminals not being serviced by the 1620. When the 1620 has completed compilation of the program, the data, if there is any, will be read. Output is sent from the 1620 on interrupt request to the 1620 output area in the "small computer" one line at a time. This output is returned to the terminal being serviced on an interrupt request between job processing and 1620 communication.

Job Permitted Under RELIC - Monitor I

RELIC permits a wide range of job types. Jobs may be entered remotely for execution under the 1620 Monitor I system or for execution in the "small computer".

1620 Monitor I JOBS:

FORTRAN II D(FORX)

FORTRAN Program compilation and execution will be the major types of jobs for 1620 execution. The proper Job Card and FORTRAN Control cards will be constructed by the RELIC - Monitor System. The FORTRAN program is entered in free format from the terminal and is held in the memory of the "small computer" until the data is read. When the data is read, the job is sent to the 1620 for execution. Any error messages which occur will be sent by the 1620 back to the terminal.

EXECUTE JOBS(XEQ)

Disk-stored program execution is possible by entering the XEQ Job Specification Record and the proper program CALL NAME. From this information, the proper Monitor Control cards are established. The data is read and the job sent to the 1620 for execution.

SPS II D

1620 Symbolic Programming System jobs will be allowed, but only by the more competent user.

"Small Computer" Job Execution:

RELIC will provide the capability for the "small computer" system programs to be called for execution within the "small computer" itself. These system programs will be stored on the 1311 Disk Storage Drive of the 1620 by means of a special conversion routine. An XEQ job will first be sent to the 1620 to write the program into the "small computer's" memory and turn control to that program. This will permit the execution of the following programs:

1. An Interactive (or Conversational) FORTRAN
2. Assembly Language (with third generation capability)
3. Production Programs
4. Loaders
5. Test Programs

MODIFICATIONS TO THE 1620 MONITOR I

Modifications to the 1620 Monitor I system are minor and consist of prohibiting the terminal job from using the 1620 peripheral devices other than the paper tape channel. This is achieved by overlaying Supervisor's I/O constants in the Monitor I/O routines. When a job card is read, the digit in column seven (which indicates the input device) is tested. If that digit is a three (##JOB 3), the job being executed is sent from the "small computer". To insure that the job output is returned to the terminal and not to the 1443 Printer, 1622 Card Punch, or 1620 Typewriter, their I/O constants are overlayed to make them appear as the paper tape channel. (e. g. a FORTRAN Print instruction will cause data to be written on to the paper tape channel and not to the 1443.) The new set of constants designed to permit only communication between the 1620 and the paper tape channel are stored in Supervisor on the disk over the normal I/O constants for all terminal jobs. The regular I/O constants are replaced for jobs entered from the Card Reader or Typewriter.

At the completion of a terminal job, control will be returned to Supervisor to read the next job from paper tape (channel). Thus, the call exit status for the 1620 will be an alphameric paper tape read under Monitor I. This will make it possible for the "small computer" to send another terminal job to the 1620 while it is idle and without an operator's intervention.

NEW MONITOR LOADER CARD

A new monitor loader card had to be designed to return job entry to the Card Reader if the 1620 was in an I/O hold for the paper tape channel. This new monitor loader card transmits the I/O constant to enable the Monitor system to read alphamerically from the Card Reader. (A $\overline{3}7$ is changed to a $\overline{5}7$ at location 01761 in in Supervisor.) When the job card is read from the card reader, the proper I/O constants are restored.

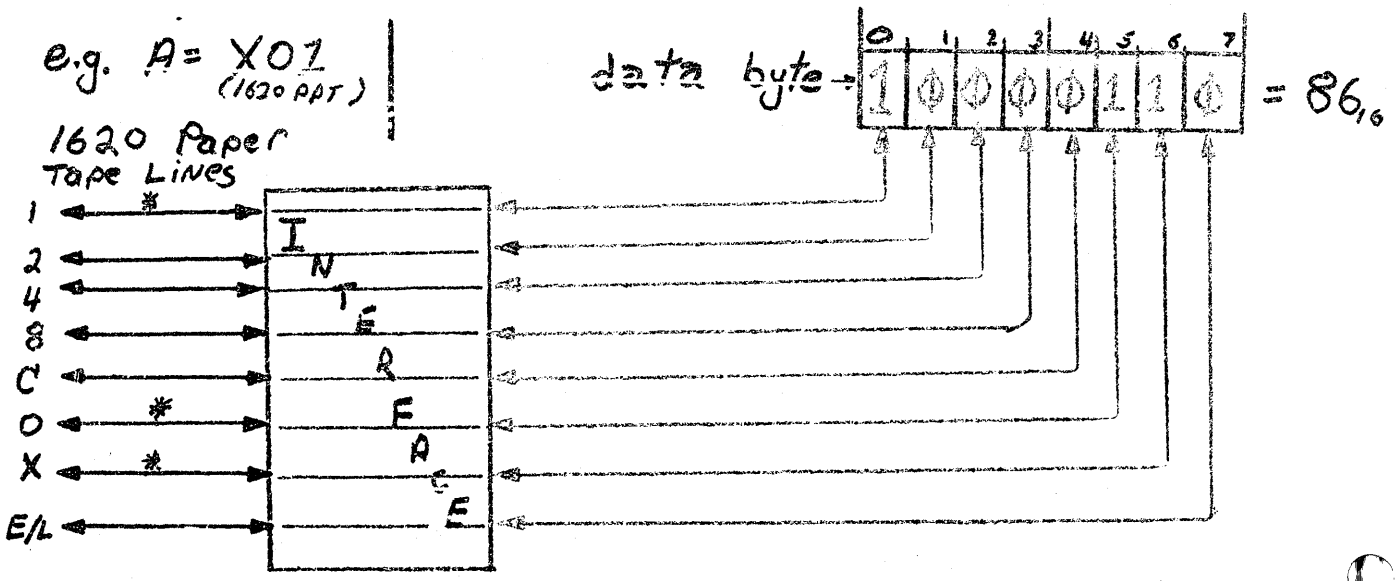
THE NEW MONITOR LOADER CARD FOR RELIC

Location	Instruction	Comment
00000	34 00044 00701	Seek to cyl. for Supervisor
00012	36 00044 00702	Read Supervisor from disk.
00024	15 01761 5000$\overline{5}$	Enable Alpha Card Read
00036	49 02402 0	Branch to Supervisor.
00044	1$\overline{1}$9636$\overline{1}$1$\overline{3}$00102	DDA for Supervisor

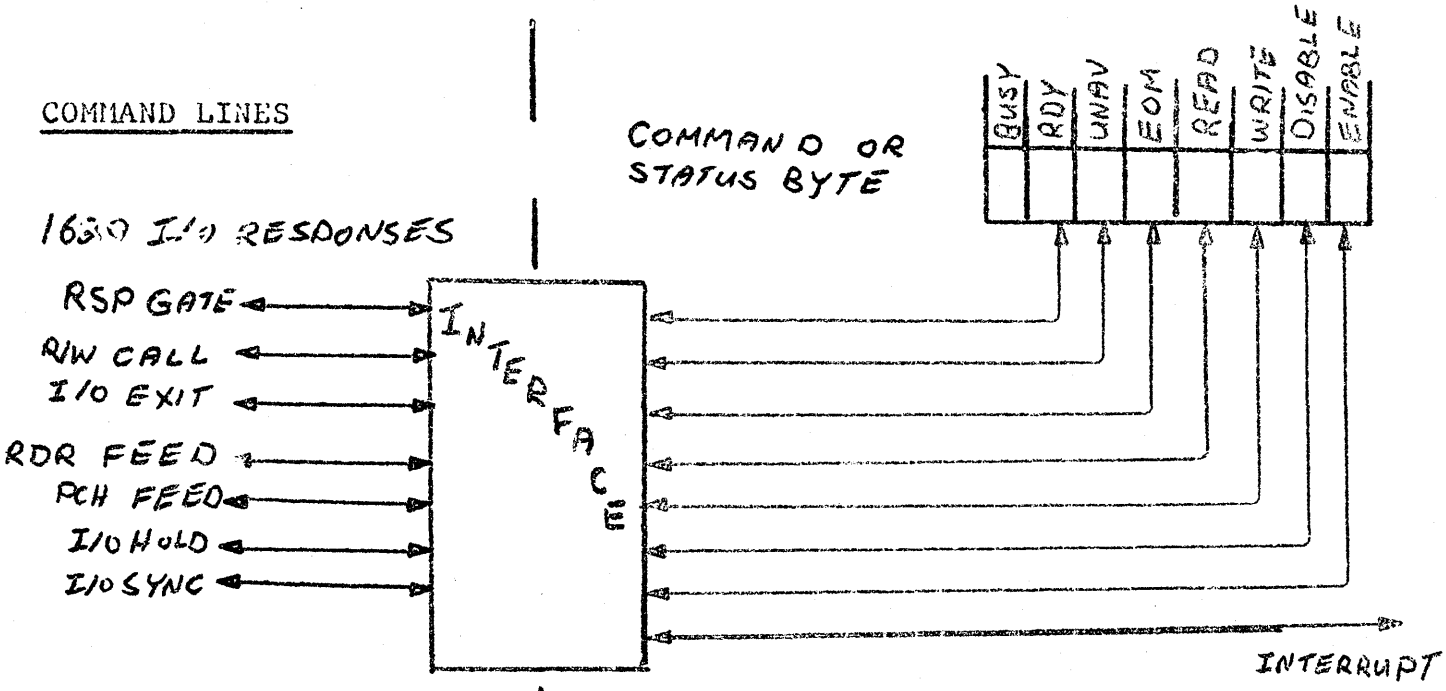
APPENDIX A

INTERFACE SPECIFICATIONS FOR 1620 and the SMALL COMPUTER

DATA LINES



COMMAND LINES



DATA TRANSFER RATE - 2,000CPS

The 1620 parity checking is utilized by staying within the 1620 character set.

APPENDIX B

ASCII - 1620 PAPER TAPE CODE CONVERSION TABLES IN HEXADECIMAL

Character	1620 paper tape code	(7-bit) ASCII	Character	1620 paper tape code	(7-bit) ASCII
A	86	41	0	04	30
B	46	42	1	80	31
C	CE	43	2	40	32
D	26	44	3	C8	33
E	AE	45	4	20	34
F	6E	46	5	A8	35
G	E6	47	6	68	36
H	16	48	7	E0	37
I	9E	49	8	10	38
J	8A	4A	9	98	39
K	4A	4B	blank	08	20
L	C2	4C	.	D6	2E
M	2A	4D	)	3E	29
N	A2	4E	+	0E	2B
O	62	4F	$	DA	24
P	EA	50	*	32	2A
Q	1A	51	-	02	2D
R	92	52	/	8C	2F
S	4C	53	,	DC	2C
T	C4	54	(	8C	28
U	2C	55	=	D0	3D
V	A4	56	@	38	40
W	64	57	≠ (recmk)	54	--
X	EC	58	line feed LF	08	0A
Y	1C	59	carriage CR (EL)	01	0D
Z	94	5A	return end of line		

"CORE-IMAGE" NUMERICAL CONVERSION TABLE - (1620 hexadecimal representation)

"Core-Image"(small computer) Numeric Character(hexadecimal)	1620 Paper Tape Code	1620 "Core-Image" Representation
0	04	0
1	80	1
2	40	2
3	C8	3
4	20	4
5	A8	5
6	68	6
7	E0	7
8	10	8
9	98	9
A	02	$\overline{0}$ or -
B	8A	$\overline{1}$ or J
C	4A	$\overline{2}$ or K
		$\overline{3}$ or L

"CORE-IMAGE NUMERICAL CONVERSION TABLE"

Numeric Character	1620 Paper Tape Code	1620 Core Representation
D	C2	$\overline{3}$ or L
E	2A	$\overline{4}$ or M
F	A2	$\overline{5}$ or N

STORAGE OF "small computer" CORE-IMAGE PROGRAMS ON THE 1311

The "small computer" uses a four-bit character structure which permits the hexadecimal number set (0,1,2,...,9,A,B,C,D, and F). Each byte of the "small computer" contains two hexadecimal characters.

The 1620 utilizes a six-bit data structure (C,F,8,4,2,1). The hex number set may be represented within the 1620 by using the above table. The decimal numbers remain the same with the 6 highest hex numbers (A, B, C, D, F) being represented in the 1620 as flagged digits ($\overline{0}$,$\overline{1}$,$\overline{2}$,$\overline{3}$,$\overline{4}$,$\overline{5}$).

Thus, by means of a simple conversion routine, numerical data can be transferred between the 1620 and the small computer at a very rapid rate. Also, the "small computer" system programs can be stored on the 1311 Disk Pack and they may be called when needed.

EXAMPLE OF FORTRAN 120 UNDER RELIC

```
DEVICE READY
TYPE USER NUMBER 170007
TYPE PROJ NUMBER  701012
TYPE JOB SPECIFICATION RECORD  FORX
ENTER PROGRAM

C  THIS PROGRAM WILL SORT AN ARRAY
READ 1,N
1 FORMAT(I3)
READ 2,(A(I),I=1,N)
2 FORMAT(F10.0)
DO 20 I=1,N
K=N-I
DO 20 J=1,K
IF (ARRAY(J)-ARRAY(J+1)) 20,20,1
1 TEMP=ARRAY(J)
ARRAY(J)=ARRAY(J+1)
ARRAY(J+1)=TEMP
20 CONTINUE
TYPE 2,(A(I),I=1,N)
END

ENTER DATA

005
12.
3.
6.
77.
4.
8

EXECUTION

3.
4.
12.
77.


END OF JOB
```

RED Indicates system commands

GREEN Input

BLACK 1620 Output

JOB CONTROL UNDER THE 1620

MONITOR I

by

GEORGE J. GERMANN

COMPUTER CENTER
SETON HALL UNIVERSITY
SOUTH ORANGE, NEW JERSEY

PRESENTED AT THE

COMMON MEETING

THE USERS GROUP FOR SMALL IBM COMPUTERS

PHILADELPHIA, PENNSYLVANIA

SEPTEMBER 10, 1968

TABLE OF CONTENTS

ABSTRACT

A. IDENTIFICATION

1. TITLE : JOB CONTROL UNDER THE IBM 1620 MONITOR I

2. AUTHOR : GEORGE J. GERMANN

3. ORGANIZATION : SETON HALL UNIVERSITY

4. DIRECT INQUIRES TO : GEORGE J. GERMANN
THE COMPUTER CENTER
SETON HALL UNIVERSITY
SOUTH ORANGE, NEW JERSEY 07079
201 762-9000 X285

5. USER GROUP CODE : 1333

6. DATE : SEPTEMBER, 1968

B. PURPOSE : To record automatically every use of the 1620 and to provide a means of listing weekly, monthly, and year-to-date reports on the 1620's use.

DESCRIPTION : This system requires two additional items of information on the Job Card, a user number, which identifies the user, a project number which identifies the project or Job the user is running. Each time a Job Card is read Supervisor is intercepted and stored on the work cylinders. A program to further analyze the Job Card is called and executed. This program checks the validity of the user number and project number and provides an appropriate message if they are not valid. For each user number and project number a section of the disk-stored "frequency of use" tables is called into memory and a unit is added to a field for the appropriate user and project number, the date is entered under the numbers, and the tables are returned to the disk. Supervisor is called back and control is returned to the Monitor Control Record Analyzer.

Four other programs are used in support of this system. The tables may be interrogated for weekly, monthly, and year-to-date reports to provide information and listings of all 1620 use during the period specified. The tables may be edited, numbers entered or deleted, contents altered or updated. A printed listing of all the numbers under their categorized headings may be obtained at any time. Lastly, the date is written in supervisor every day

C. SPECIFICATIONS : 1. Storage 20K
2. Machine configuration-Card System,Disk,Printer
3. Hardware features-Auto-divide, indirect address

INTRODUCTION

BACKGROUND

This paper is concerned with a system developed at Seton Hall University to record the frequency of use and to control jobs executed under the IBM 1620 Monitor I System. Similar programs have been written to control scheduling and to log under the Monitor System. At the Boston Common Meeting in March of 1967, William Lingo of Pratt Institute in New York presented a paper entitled, "Logging Under Monitor Control for the IBM 1620", which logs users and provides statistics on their use. Patrick P. Emin of the University of New Brunswick in Canada presented an excellent paper on scheduling and "Running FORTRAN II-D in Background Mode with Spooling and Check Points" at the COMMON Meeting in Chicago in April of 1968. And there are probably many more programs of this type. However, you may find that many of the techniques discussed in this paper may be of particular use to you and to your installation.

PROBLEM

When we expanded our IBM 1620-1622 Card System to a Monitor System with a 1311 Disk Storage Drive in the Summer of 1966, we needed to know more accurately the volume of use on the 1620, who was using it and for what purpose. We found that logging time sheets proved somewhat ineffective, since users and operators often failed to fill out time sheets or log time accurately. Since the Computer Center at Seton Hall University is run as an Open Shop it was necessary to know how many faculty members and students were using

the facilities at the Computer Center. Because the Monitor System made computer programming and computer access much easier, we had greater volume of use than before which posed the problem of how to control unauthorized use. We also wanted to know which departments were using the computer and which of those departments had the greatest use. And knowing who was using the computer we could determine who was not using it; and therefore, approach those departments where the computer could be of assistance.

SOLUTION

Our solution was to let the computer do its own "housekeeping". We rationalized that logging amounts of time spent per user by the 1620 could not be easily accomplished without an internal hardware "clock" and an interrupt capability, so we decided to log the frequency of use. Each time a job card is read a unit would be added to a field in tables permanently stored on the disk. In addition, The 1620 Monitor System and the 1311 Disk Storage Drive provided all the necessary means to accomplish this task.

PROGRAMMING REQUIRED

Two additional items of information were required in the Job Card, a user number, which identifies the user, and a project number, which identifies the job or project being executed.

Four programs were written to implement the system:

1. A program to further analyze the Job Card, control job execution, and update tables.

To call this program modifications to Monitor's Supervisor were necessary; however, the Monitor Control Record Analyzer (MCRA) in Supervisor underwent only minor modifications.

2. A program to interrogate these tables to provide information and listings on all 1620 use during a specified period.
3. A program to edit, delete, enter, and/or list numbers in the table.
4. A program to enter the date into an available location in supervisor.

A more extensive description of those programs will follow in this report.

DESCRIPTION OF USER NUMBER AND PROJECT NUMBERS

We are able to monitor control and provide statistics on computer use under users and projects by using a special user and project number classification system.

Every person using the facilities at the Computer Center at Seton Hall University is given a user number and assigned a project number which authorizes the user to use the computer for his particular project.

USER NUMBER

For each user number there are five categories: Faculty (1), Administrative Staff (2), Undergraduate full-time students (3), Undergraduate part-time (4) students, and Graduate Students (5). Each category is represented by a single digit 1 through 5 respectively. Each user number is a six-digit number and has the structure: CSDDNN where the first digit (C) represents one of the five categories, the second digit (S) the school, the third and fourth digits (DD) the department under the school, and the last two digits the (NN) sequence number.

EXAMPLE - 312801 indicates an undergraduate full-time student (3), in the School of Arts and Science (1), in Mathematics Department (28), and is the first student assigned this number (01).

PROJECT NUMBERS

For each project number there are nine categories: Arts and Science (1), Business (2), Divinity (3), Education (4), Medical (5), Dental (6), Computer Center (7), Administrative (8), and Other (9). Each category is represented by a single digit 1 through 9 respectively. Each project number is a six-digit number and has the structure: CDDNNN where the first digit (C) is one of the nine categories, the second and third digits (DD) the department, and the last three digits (NNN) the sequence number or class number.

EXAMPLE - 128181 indicates the project is in the School of Arts and Science (1), under the Mathematics Department (28), for the Mt 181 class (181).

This user and project classification system provided an easy structure from which to design tables.

DESCRIPTION OF TABLES FOR 1620 USE FOR PROJECT AND USER NUMBERS

The files for storing the frequency of 1620 use are separated into three division (1), USER NUMBERS (non-computer center), (2) PROJECT NUMBERS, and (3) COMPUTER CENTER USERS WITH PROJECT NUMBERS.

The files for Divisions 1 and 2 have the capacity for 500 user numbers, 100 for each of the 5 user divisions and 500 project numbers, 50 for each of the nine divisions. The format for the user number and project number file is a six-digit user or project number followed by a three-digit weekly count, a three-digit monthly count, a four-digit year-to-date count, and a four-digit (first four digits of date).

EXAMPLE: $\overline{U}$UUUUU$\overline{W}$WW$\overline{M}$MM$\overline{Y}$YY$\overline{D}$DDD WHERE UUUUUU is the user or project number (each high order position is flagged) WWW is the weekly count
MMM is the monthly count
YYYY is the year-to-date
DDDD is the date of last use.

1. USER NUMBERS are stored on cylinder 18, sector addresses 09000-09099. Each user number and its related fields occupies 20 core locations, and the entire file for a particular user number category occupies 20 sectors. There are 5 categories under which the user numbers are classified.

USER CATEGORIES	FIRST DIGIT	SECTOR ADDRESSES
FACULTY	1	09000-09019
ADMINISTRATIVE	2	09020-09039
FULL-TIME UNDERGRAD.	3	09040-09059
PART-TIME UNDERGRAD.	4	09060-09079
GRADUATE STUDENT	5	09080-09099

2. PROJECT NUMBERS are stored between sector addresses 09100 and 09199.

Each project number and its related fields occupes 20 core locations and the entire file for a project number category occupies 10 sectors. There are nine categories under which a project number can be classified.

PROJECT CATEGORIES	FIRST DIGIT	SECTOR ADDRESSES
ARTS AND SCIENCE	1	09100-09109
BUSINESS	2	09110-09119
DIVINITY	3	09120-09129
EDUCATION	4	09130-09139
MEDICAL	5	09140-09149
DENTAL	6	09150-09159
COMPUTER CENTER	7	09160-09169
ADMINISTRATIVE	8	09170-09179
OTHER	9	09180-09189

Sector addresses 09190-09199 are not used and are available.

3. COMPUTER CENTER USER NUMBERS

All Computer Center user numbers, both academic and administrative staff, are placed into a different file then those described on the previous page. These numbers are stored in cylinder 19. Both the academic and administrative user numbers contain space for 10 users with 20 project numbers under each user. The academic staff for the Computer Center will have a "17" as the first two digits of the number and these numbers are stored between sector address 09200 and 09239. The file for this category occupies 40 sectors. The administrative Computer Center user numbers are stored between sector address 09240 and 09279.

The user number again is a six-digit number and will appear as the first number of the file followed by a project number, a three-digit weekly count, a three-digit monthly, a four-digit yearly and a four-digit date, followed by another project number, etc.

JOBCD-PROGRAM TO FURTHER ANALYZE THE JOB CARD

Modification to the MCRA stores Supervisor on the work cylinders at sector address 01400 and calls the program to further analyze the Job Card.

JOB CARD STRIPPED

The contents of the job card which had been read by Supervisor is found in memory locations 13000 to 13159. Supervisor's Transfer Numeric Strip Routines at 08146 (version 2) is used to strip the numerical digits of the user number and project number on the job card. The calling sequence for this routine is described on page15 of this paper.

Once the user and project numbers have been stripped and re-constructed they are tested for validity. If the numbers are found to be invalid, an error message is typed and 10 seconds is allowed for entry of the correct number. If the correct number is not entered within 10 seconds, the job is abandoned and control is returned to Supervisor to read another job card.

NON-COMPUTER CENTER USER NUMBER PROCEDURE

User numbers of non-Computer Center Users are processed before the project number. The user's category of the user file is read from the disk into memory, and a "straight look-up" method is used to search for the user number in the table called. If the user number punched on the Job Card does not match a number in the table, "USER NUMBER NOT AUTHORIZED", will be typed and the job abandoned. When the user number has been found in the table, a unit is added to the weekly count and the first four digits of the date are entered. The tables are then returned to the disk and the same procedure is followed for the project number.

COMPUTER CENTER STAFF PROCEDURE

Since additional information is required for the Computer Center staff, their user numbers are processed somewhat differently. The project number on their job cards is updated first. This is necessary to test for the validity of the project number, since when a particular Computer Center user's project number is not found under his user number it is entered automatically. Thus, if we test the project number first, we are assured it is authorized if it is entered under the user number. Therefore, a unit of use and the date are entered in the project number field under the Computer Center user number. The fields under the project number in the project number division are also updated to provide a total use for that project regardless of the user.

ADDITIONAL TIME SPENT

The time spent in logging a unit of use and the date under each user number and project number adds approximately one and one-half seconds to two and one-half seconds to the Job Card Routine.

JOB CARD SUMMARY SHEET OPTION

When a digit is punched on the job card between the user number and the project number (col. 38), the job card summary sheet option is executed. This option will print a header sheet identifying the job being run and contains the Seton Hall University Computer Center heading, the date, user and project numbers, and other descriptive information furnished by the job card. (see page17)

SUPERVISOR RETURNED

When the user and project numbers have been updated and/or the Job Card Summary Sheet option executed, Supervisor is read back from the work cylinders and control is returned to the Monitor Control Record Analyzer and the next control card is read.

OTHER APPLICATIONS

This program will be used to control and inhibit certain user's programs from using one of the 1620 peripheral devices by overlaying Monitor I's I/O constants. Thus, any job which has access to our 1620 from our remote terminal system (RELIC) can only return output back to the terminals, and not to the printer, typewriter, or card punch.

This program can be further used to control scheduling of jobs and select the input device which is ready to send.

Lastly, the program provides the possibility of writing messages and instructions to a certain user or users.

DESCRIPTION OF SUPPORTING PROGRAMS

A. WEEKLY - MONTHLY - YEARLY - Interrogates frequency tables for listings.

The purpose of this program is to interrogate the tables (stored on the disk and updated by JOBCD each time a job card is read) and punch a deck of cards from which a weekly, monthly, or year-to-date reports can be listed describing the frequency of use for the 1620 by users and projects for the period specified. The report is divided into three divisions:

1. A listing by non-Computer Center user numbers. This listing will appear as the first page of the report. It will contain use for users under the five categories. If there is no use for the time specified for a particular user number or category, the number or category will not be listed. Only those numbers for which there has been use appears in the listing.
2. A listing of all project numbers. This section contains project numbers under the nine categories, and only those for which there has been use during the period specified.
3. A listing of Computer Center user numbers with project numbers under user. This report will appear as the last section of the listing. The user numbers are classified under Academic and Administrative staff with their projects listed under the user number.

WEEKLY - lists all weekly use and resets the weekly fields (run every week)

MONTHLY- lists all monthly use and resets the monthly fields (run every month)

YEARLY - lists all year-to-date use and reset all fields (run every year)

B. EDTBLE Edit disk-stored frequency tables.

The purpose of this program is to provide the following four capabilities:

1. To enter new user and project numbers into the tables
2. to delete older user and project numbers from the tables
3. to edit or correct frequency counts to the part of the table or tables that need correcting or that get "wiped-out".
4. To provide listings of all user numbers and project numbers which are stored in the tables, and to list them under their proper category.

C. UPDATE program executed each day to store the date.

The purpose of this program is to read and store the current date on the disk in the supervisor program so that it is available for use at any time by other programs. The five digits of the date (month, day, and last digit of the year) are stored in the supervisor at memory locaton 01303 and 01307. The sector address containing the date is 19648. If this sector is read into core location XXXXX, the date would be found at XXXXX+5 with a flag at XXXXX+1. This program was written by Mrs. Judith S. Heyman, a former member of our staff.

EXECUTION AND EXPERIENCE

Two years of experience operating under this system has more than proved its usefulness to our installation. We are now able to control who uses and for what purpose our computer is used.

We are able to document how the computer is being used which is of "some" importance to the administrations who provide the fiscal budget. Also by evaluating past use we have been able to better predict and prepare for increases in future use.

SPECIFICATIONS

MODIFICATIONS TO SUPERVISOR (MONITOR I VERSION 2)

In order to further analyze the Job Card the following modifications have been made to the Supervisor Program.

1. A NOP (41) has been placed at core locations 05238 and 05239 to avoid transmitting a digit into 06094.

   EXAMPLE: 05238 41 06094 02851

2. An area of Supervisor 06086 - 96169 is used when there are more than one disk storage files. Since we have only the single disk file, this area is not used by the Supervisor. Therefore, the instructions listed below are placed in that area. This routine is used to temporarily store the Supervisor on the work cylinders and call the Job Card program to further analyze the Job Card.

```
MJB 20 SK   A                    06086 34 06142 00701
       WDN  A                    06098 38 06142 00702
       TR   IBMMOD,B             06110 31 00440 06156
       BI   MONOCAL, 19          06122 46 00796 01900
       B    OVERLAY              06134 49 00466 00000
A      DDA  ,1,01400,173,02302   06142 1̄01400Ī77̄02302
B      DDA  ,1,SSSSS,CCC,02402   16156 1S̄SSSS̄CCC̄02402
       DC   1,≠                  16170 ≠
```

where SSSSS is the sector address of the program "JOBCD", and CCC is the number of sectors used by JOBCD.

The Sector Addresses where this section of Supervisor is stored is between 19687-19698.

ROUTINE TO EXTRACT NUMERICAL STRIP

The routine to extract numerical strip from the Job Card in Read area (13000-13159) is at core location 08146.

The Calling Sequence

```
                          TFM 09800 , NCHAR
                          BTM 08146 , ADDR
Return from routine TF  WORK  , 09811
```

NCHAR are the number of characters

ADDR is the odd address (high order) of field to be stripped

09811 contains the stripped numerical field.

JOB CARD FORMAT

Each Job Card entered under the Monitor I System for the IBM 1620 at The Computer Center, Seton Hall University must contain a user number and project number according to the specifications below. Additional information may be punched on the job card to identify the job or print a Job Summary Sheet.

COLUMN	SPECIFICATION
1 - 2	≠≠
3 - 5	JOB
6	BLANK
7	1 FOR TYPEWRITER, 3 FOR PAPER TAPE OR 5 for CARD INPUT 1
32 - 37	USER NUMBER
38	BLANK (1 will print a Job Card Summary sheet title page
39 - 44	PROJECT NUMBER
45 - 47	BLANK
48 - 72	IDENTIFICATION
73	BLANK
74 - 79	JOB TYPE
80	BLANK

SETON HALL UNIVERSITY

COMPUTER CENTER

09/03/68

**JOB CARD SUMMARY

USER NUMBER-	170007
PROJECT NUMBER-	700001
IDENTIFICATION.	SUMMARY SHEET PAGE OPTION
JOB TYPE.	SPSII

EXAMPLES OF TYPED MESSAGES FOR INVALID USER OR PROJECT NUMBERS.

```
‡‡JOB 5                                155014 700001

USER NUMBER NOT AUTHORIZED.
END OF JOB


‡‡JOB 5                                170007 300001

PROJECT NUMBER NOT AUTHORIZED.
END OF JOB


‡‡JOB 5

INVALID USER NUMBER, TO TYPE NUMBER SW1 ON.

170007RS
PLEASE TURN OFF SW 1. THANK YOU.
INVALID PROJECT NUMBER, TO TYPE NUMBER SW1 ON.
701011RS


‡‡JOB 5

INVALID USER NUMBER, TO TYPE NUMBER SW1 ON.

END OF JOB
```

REPORT FOR 1620 USE FOR THE WEEK 08 30 68

USER NUMBERS

UNDERGRADUATE FULL TIME STUDENT

NUMBER	WEEKLY	TOTAL	DATE LAST USED
312805	065	0193	08 28 68

WEEKLY TOTAL 00065

REPORT FOR 1620 USE FOR THE WEEK 08 30 68

COMPUTER CENTER

FACULTY

USER NO. 170007

NUMBER	WEEKLY	TOTAL	DATE LAST USED
700001	002	0026	08 30 68
701011	005	0013	08 30 68
700002	002	0026	08 30 68
701012	005	0030	08 30 68

WEEKLY TOTAL 00014

USER NO. 170008

NO WEEKLY USE

USER NO. 170001

NO WEEKLY USE

USER NO. 170002

NO WEEKLY USE

USER NO. 170010

NO WEEKLY USE

USER NO. 170009

NO WEEKLY USE

ADMINISTRATIVE STAFF

USER NO. 270001

NO WEEKLY USE

USER NO. 270006

NUMBER	WEEKLY	TOTAL	DATE LAST USED
216434	006	0010	08 28 68
830001	005	0005	08 23 68

WEEKLY TOTAL 00011

REPORT FOR 1620 USE FOR THE WEEK 08 30 68

PROJECT NUMBERS

BUSINESS

NUMBER	WEEKLY	TOTAL	DATE LAST USED
216434	006	0010	08 28 68

WEEKLY TOTAL 00006

COMPUTER CENTER

NUMBER	WEEKLY	TOTAL	DATE LAST USED
700001	027	0051	08 30 68
700002	069	0215	08 30 68
701006	006	0037	08 26 68
701011	011	0041	08 30 68
702010	001	0001	08 22 68
701012	005	0030	08 30 68

WEEKLY TOTAL 00119

ADMINISTRATION

NUMBER	WEEKLY	TOTAL	DATE LAST USED
830001	005	0005	08 23 68
820005	018	0018	08 20 68

WEEKLY TOTAL 00023

WEEKLY TOTAL 00148

NUMBER	WEEKLY	TOTAL	DATE LAST USED
701006	006	0037	08 26 68
701011	006	0028	08 28 68

WEEKLY TOTAL 00012

USER NO. 270012

NO WEEKLY USE

USER NO. 270013

NUMBER	WEEKLY	TOTAL	DATE LAST USED
702010	001	0001	08 22 68

WEEKLY TOTAL 00001

USER NO. 270016

NUMBER	WEEKLY	TOTAL	DATE LAST USED
700001	019	0019	08 28 68

WEEKLY TOTAL 00019

USER NO. 270007

NUMBER	WEEKLY	TOTAL	DATE LAST USED
820005	018	0018	08 20 68
700002	002	0002	08 28 68
700001	006	0006	08 28 68

WEEKLY TOTAL 00026

USER NO. 270015

NO WEEKLY USE

CARD SYSTEM MODIFICATIONS, SPS AND PDQ

John H. Wise
Washington and Lee University
Lexington, Virginia 24450

This paper describes some modifications in the card system currently used at Washington and Lee. The computer configuration is a 1620, Model 1, with 1622 card read-punch and 1443 printer, and includes the special features package for indirect addressing and TNS, TNF. Memory is 20K, but the modifications would be the same for larger memory machines. The special feature package is required for the PDQ modifications.

The PDQ modifications have gone through a number of stages. Initial changes to permit printer output were made at Washington and Lee as well as by Mr. John W. Holmes, Cooper-Bessemer Corp., Mount Vernon, Ohio. Mr. Andre Lacerte, while at Washington and Lee, made an extensive revision with added subroutine features. About the only Lacerte modification retained here is the E-format style in which the decimal point may be placed wherever desired (however, the total width of an E field is filled with digits even if w is greater than 14).

The sequence of modifications is outlined below.

1) Changes for 1443 printer. Both compiler and subroutines required changes. Features include: Trace on printer in a two line format (address and value); TYPE statements on console, but requiring CONTROL for new lines; an option of 105 point position output and an option for either 72 or 80 column card input.

2) Changes in switch settings so that ON calls the feature into operation (thus SS1 ON calls for printer listing of source).

3) Changes in loader from the digit-by-digit scheme to a card read scheme. Loading time is reduced from 40.5 sec to 22.5 sec with this modification.

4) Symbol table on printer. To include this, the SS2 option of punched referenced source statements has been deleted (SS2 is used below). Listing of the symbol table is under control of SS1. Page skip on channel 12 is included in source and symbol table listing. The listing is four items per line except for dimensioned variables at one line each.

5) A pre-compiler option has been included under control of SS2. If SS2 is ON, object punching and symbol table listing is suppressed. With SS2 OFF, object is punched until an error is detected, and the symbol table is listed only after a successful compilation of the program.

6) Subroutine changes are required for printer output and the E-format mentioned above (the E-format change forces the two-line trace output).

7) A condensed subroutine deck is used for all programs that do not require relocatable subroutines. (A program can be checked for relocatables on the third from last card of the object. This card has -0402 in cc 1-5, and will contain 0's in cc 55 to 70 if no relocatables are needed - otherwise some 1's will be in cc 55 to 70). This deck has been further compressed by a revised loading program that uses a 5 column address instead of the 19 columns of the standard deck. It can not be used if subroutines are included in the object deck (if SS3 is ON during compilation).

After completing these PDQ modifications, the SPS deck was examined. By using a loading program similar to that in the PDQ decks and eliminating some unnecessary or unused instructions (e.g., the test for Model 1 or Model 2 machine), the SPS III deck was condensed from 514 cards to 241 cards. This deck size is much more convenient to handle.

The only SPS modification was to remove the halt which is included in an SPS object deck at the conclusion of loading the object. At present, a branch to start occurs automatically.

Copies of the modified decks can be obtained from the author.

CONVERSION OF FORTRAN II TO FORTRAN IV

BY

PATRICIA E. RAY

D. W. MATTSON COMPUTER CENTER

TENNESSEE TECHNOLOGICAL UNIVERSITY

COOKEVILLE, TENNESSEE 38501

CONVERSION OF FORTRAN II TO FORTRAN IV

INTRODUCTION

WHEN FORTRAN II OR II-D PROGRAMS WRITTEN FOR THE IBM 1620 COMPUTER NEED TO BE MADE COMPATIBLE FOR RUNNING ON A COMPUTER USING FORTRAN IV, IT IS NECESSARY TO CHANGE CERTAIN STATEMENTS. THE FILTER PROGRAM, RUNNING ON THE 1620 SYSTEM, CHANGES MANY OF THESE INCOMPATIBLE STATEMENTS OR POINTS OUT TO THE PROGRAMMER THE MORE DIFFICULT CHANGES TO BE MADE.

BASIC PROGRAM DESCRIPTION

THE PRIMARY CHANGES TO BE MADE IN A FORTRAN II SOURCE DECK ARE FOUND IN THE INPUT/OUTPUT STATEMENTS, READ, PRINT, AND PUNCH IN PARTICULAR. SINCE FORTRAN IV REQUIRES A DEVICE CODE FOR EACH OF ITS I/O STATEMENTS, THE OPERATOR WILL BE ASKED TO ENTER THE DEVICE CODES FOR THE CARD READER, CARD PUNCH, AND PRINTER AT THE BEGINNING OF THIS PROGRAM. THUS, THE FORTRAN II STATEMENT ..PRINT 70, A.. WILL BE CHANGED BY THE PROGRAM TO ..WRITE(N,70)A.., WHERE N REPRESENTS THE DEVICE CODE FOR THE PRINTER. OTHER FORTRAN II I/O STATEMENTS THAT WILL BE DETECTED ARE THE TYPE, ACCEPT, AND IF SENSE SWITCH STATEMENTS, AND THE RECORD, FETCH, AND DEFINE DISK STAEMENTS OF FORTRAN II-D. SINCE THESE STATEMENTS REQUIRE REPROGRAMMING THEY ARE POINTED OUT ON THE CONSOLE TYPEWRITER. NO CHANGE IS MADE TO THE ORIGINAL STATEMENT.

OTHER CHANGES TO A FORTRAN II SOURCE DECK MAY BE NECESSARY WITH THE USE OF THE ARITMETIC FUNCTIONS -- SINF, COSF, ATANF, SQRTF, LOGF, EXPF, AND ABSF. IN ALL CASES THE F MUST BE DROPPED. THIS CAN USUALLY BE DONE BY THE PROGRAM. THE LOGF TO ALOG AND ABSF TO ABS OR IABS CONVERSION MAY REQUIRE MORE CHANGES THAN THIS PROGRAM CAN HANDLE. IF SO, DIAGNOSTICS WILL BE TYPED.

EACH CARD IS GIVEN A SEQUENCE NUMBER THAT IS PLACED IN THE LAST FOUR COLUMNS OF THE CARD. THE PROGRAM TERMINATES UPON THE DETECTION OF AN ..END.. CARD WITHIN THE FORTRAN SOURCE DECK. THIS LAST CARD IS REPRODUCED AND THE PROGRAM HALTS.

THE OUTPUT

AFTER THE PROGRAM ANALYZES EACH FORTRAN II SOURCE CARD, IT EITHER REPRODUCES THAT CARD IMAGE, MAKES NECESSARY CHANGES AND THEN PUNCHES THE CARD, OR DETECTS THE MORE COMPLICATED CHANGES AND PUNCHES THE CARD WITHOUT THAT CHANGE. THE CHANGES THAT COULD NOT BE MADE WILL BE LISTED ON THE TYPEWRITER, GIVING THE DIAGNOSTIC AND THE CARD NUMBER IN THE LAST FOUR COLUMNS OF THE CARD.

THE LISTING

AN OPTION OF THIS PROGRAM IS A PRINTED LISTING OF EVERY CARD IMAGE, INCLUDING CHANGES, AND AN ASTERISK TO THE RIGHT OF THE UNCHANGABLE CARD IMAGES.

THE DIAGNOSTICS

AS SOON AS THE PROGRAM DETECTS AN UNCONVERTIBLE STATEMENT OR ARITHMETIC FUNCTION, A MESSAGE IS TYPED FOR THE PROGRAMMER ALONG WITH THE CARD NUMBER.

THESE DIAGNOSTICS INCLUDE . . .

CHANGES REQUIRED FOR THE TYPE, ACCEPT, FETCH, AND RECORD I/O STATEMENTS.

CHANGES REQUIRED FOR THE DEFINE DISK AND IF SENSE SWITCH STATMENTS.

PROBABLE REQUIREMENTS FOR CONTINUATION CARDS IF THE STATEMENT IS ALREADY USING ALL CARD COLUMNS AVAILABLE AND NEEDS ADDITIONAL ROOM.

DETECTION OF MONITOR CONTROL RECORDS.

SYSTEM CONFIGURATION

IBM 1620, MODEL 1, 20K

SPECIAL FEATURES . . . TNS, TNF, MF

INDIRECT ADDRESSING

1443 PRINTER (OPTIONAL)

PROGRAM OPERATION

SWITCH SETTINGS . . .

OFLOW --PROG

I/O --STOP

PARITY--STOP

DISK --PROG

SENSE SWITCH 1 -- ON FOR PRINTED LISTING ALONG WITH PUNCHED OUTPUT.

DATE TABLE FOR THE MONITOR SYSTEM

BY

FRANK E. BUSH, JR.

D. W. MATTSON COMPUTER CENTER

TENNESSEE TECHNOLOGICAL UNIVERSITY

COOKEVILLE, TENNESSEE 38501

DATE TABLE FOR THE MONITOR SYSTEM

INTRODUCTION

IN AN INSTALLATION THAT HAS A LARGE TURN OVER IN THE NUMBER OF PROGRAMS THAT ARE PERMANENTLY STORED ON DISK, IT IS OFTEN DESIRABLE TO KNOW THE DATE THAT THE PROGRAMS WERE LOADED TO DISK. THIS IS ESPECIALLY TRUE WHEN IT IS NECESSARY TO RUN ON A BACK-UP PACK, OR WHEN SEVERAL UNKNOWN-IN-AGE PROGRAMS ARE FOUND OCCUPYING DISK SPACE. IT IS FOR REASONS LIKE THESE THAT A MEANS OF KEEPING UP WITH THE DATE THAT PROGRAMS WERE LOADED TO DISK HAS BEEN PURSUED.

ADDING THE DATE TABLE TO MONITOR

THERE ARE THREE PHASES THAT MUST BE EXECUTED BEFORE THE DATE TABLE WILL BE OPERATIONAL. THESE PHASES INVOLVE RESERVING SPACE FOR THE DATE TABLE, STORING THE CURRENT DATE ON THE DISK, AND PATCHING SECTIONS OF MONITOR TO ALLOW THE DATE TABLE TO BE MAINTAINED. A FOURTH STEP IS THE LOADING OF AN OUTPUT PROGRAM THAT WILL DISPLAY THE CONTENTS OF THE DATE TABLE.

I. RESERVING SPACE FOR THE DATE TABLE.

IT IS NECESSARY TO ALLOCATE 140 CONSECUTIVE SECTORS ON ONE CYLINDER FOR THE DATE TABLE. THIS IS EASILY ACCOMPLISHED BY USE OF THE *DLOAD DISK UTILITY PROGRAM, LOADING 140 SECTORS FROM THE WORK CYLINDERS TO A CHOSEN AREA ON DISK. IT IS IMPORTANT THAT THE SECTORS ALL BE ON THE SAME CYLINDER, BECAUSE THE OTHER PROGRAMS MAKE NO ALLOWANCE FOR CYLINDER OVERFLOW.

THE 140 SECTORS THAT ARE ALLOCATED WILL BE UTILIZED AS FOLLOWS
11 SECTORS FOR THE TABLE MAINTAINENCE PROGRAMS
89 SECTORS FOR THE DATE TABLE
40 SECTORS FOR A CORE BUFFER AREA.

II. LOAD THE CURRENT DATE PROGRAM.

IN ORDER FOR THE DATE TABLE TO HAVE THE CURRENT DATE AVAILABLE TO IT, THE NEXT PHASE INVOLVES LOADING A PROGRAM THAT WILL STORE THE CURRENT DATE IN A PLACE THAT THE MAINTAINENCE PROGRAMS WILL BE ABLE TO ACCESS IT. THE PLACE FOR THE DATE IS THE DISK LABEL SECTOR --19800. THE DATE WILL BE STORED AS A SIX-DIGIT NUMBER, I.E., TWO DIGITS FOR THE MONTH, TWO DIGITS FOR THE DATE, AND TWO DIGITS FOR THE YEAR. THE DATE WILL BE READ FROM A CARD BY THIS PROGRAM, CHECKED FOR VALIDITY AS THROUGHLY AS POSSIBLE, AND THEN PLACED IN THE DESIGNATED AREA IN THE DISK LABEL SECTOR.

III. LOAD THE MAINTAINENCE PROGRAM PACKAGE.

THIS PROGRAM WILL PUT THE MAINTAINENCE PROGRAMS INTO THE AREA RESERVED ON DISK AND AT THE SAME TIME MAKE THE NECESSARY CHANGES TO MONITOR TO ALLOW THE MAINTAINENCE PROGRAMS TO BE CALLED AT THE PROPER TIME TO UPDATE THE DATE TABLE. NONE OF THE FUNCTIONS OF MONITOR HAVE BEEN ALTERED.

BEFORE THE MAINTAINENCE PACKAGE LOADS ITSELF TO DISK, A MESSAGE WILL BE TYPED ON THE CONSOLE TYPEWRITER AKSING FOR THE LOCATION OF THE AREA THAT WAS RESERVED. IN ADDITION, THE PROGRAM

WILL ASK IF A PRINTER IS ATTACHED TO THE COMPUTER. IF SO, THE MAINTAINENCE PROGRAM WILL BE MODIFIED, BY THE PROGRAM, TO ALLOW THE *DELET AND DK LOADED STATEMENTS TO BE PRINTED AS WELL AS TYPED.

IV. LOAD THE OUTPUT PROGRAM.

SPACE SHOULD THEN BE RESERVED FOR THE OUTPUT PROGRAM THROUGH THE USE OF *DLOAD, 20 SECTORS ARE NEEDED. THE OUTPUT PROGRAM IS ALSO SELF-LOADING AND WILL ALSO INQUIRE ABOUT THE LOCATION OF THE DATE TABLE AND CHECK IF A PRINTER IS ATTACHED. IF THERE IS NO PRINTER, OUTPUT WILL BE ON CARDS.

MAINTAINENCE OF THE DATE TABLE

THE CURRENT DATE WILL BE LOADED TO DISK EACH MORNING BY EXECUTING THE DATE PROGRAM. EACH TIME A PROGRAM IS LOADED TO DISK, A 10-DIGIT ENTRY WILL BE MADE TO THE DATE TABLE--THE 4-DIGIT DIM NUMBER ASSIGNED BY MONITOR AND THE CURRENT 6-DIGIT DATE. THE TABLE IS KEPT IN SEQUENCE BY DIM NUMBER. A NEW ENTRY IS MADE WHEN A PROGRAM IS LOADED TO DISK. THE DATE PORTION OF THE TABLE ENTRY IS UPDATED WHEN THE *DREPL OR *DCOPY UTILITY ROUTINE IS USED. THE COMPLETE TABLE ENTRY FOR A PROGRAM IS REMOVED WHEN THE PROGRAM IS DELETED.

IF A PRINTER IS INSTALLED, *DELET AND DK LOADED MESSAGES WILL ALSO APPEAR ON THE PRINTER LISTING.

ANY TIME A MAINTAINENCE ROUTINE IS NEED TO UPDATE THE TABLE, CORE LOCATIONS 07000 THROUGH 11000 ARE WRITTEN TO THE CORE BUFFER AREA ON DISK AND THE MAINTAINENCE PROGRAM IS BROUGHT IN TO ACCESS THE DATE TABLE. WHEN THE UPDATE IS COMPLETED, CORE IS RESTORED TO ITS PREVIOUS STATE.

OUTPUT OF THE TABLE

THE OUTPUT PROGRAM PROVIDES A CONCISE LISTING OF THE CONTENTS OF THE DATE TABLE. THE EQUIVALENCE TABLE IS CONSULTED FOR THE NAME OF THE PROGRAM. IF THE PROGRAM HAS NO NAME AND ITS DIM NUMBER IS 0170 OR LESS, IT IS GIVEN THE NAME -- (MONITOR) -- TO INDICATE THAT IT IS PART OF THE SYTEM.

IF THE PROGRAM HAS NO NAME AND ITS DIM NUMBER IS GREATER THAN 0170, IT IS DESIGNATED BY -- **NONE* --.

A SAMPLE LISTING IS INCLUDED AT THE BACK OF THIS PAPER.

SYSTEM CONFIGURATION

IBM 1620
SPECIAL FEATURES . . . TNS, TNF, MF
INDIRECT ADDRESSING
1443 PRINTER (OPTIONAL)
ONE OR MORE 1311 DISK DRIVES USING THE MONITOR SYSTEM

DIM NO.	NAME	DATE LOADED
0146	(MONITOR)	AUGUST 15, 1968
0178	CARDS1	JULY 3, 1968
0179	CARDS2	JULY 3, 1968
0180	CARDS3	JULY 3, 1968
0181	CARDS4	JULY 3, 1968
0277	STRESF	AUGUST 15, 1968
0290	SPEEDT	AUGUST 15, 1968
0291	INVRS	SEPTEMBER 5, 1968
0293	FEP1A	SEPTEMBER 5, 1968
0294	ZERO	JULY 29, 1968
0313	FILTER	AUGUST 16, 1968
0314	SIMULT	AUGUST 16, 1968
0315	**NONE*	AUGUST 16, 1968
0323	STRESS	AUGUST 28, 1968
0330	VTOC	AUGUST 16, 1968
0343	RUNGE	JULY 16, 1968
0344	INVRST	AUGUST 27, 1968
0345	FEP02	SEPTEMBER 5, 1968
0346	FEP01	SEPTEMBER 3, 1968
0347	NASA2	AUGUST 10, 1968
0349	NASA3	AUGUST 10, 1968
0351	SIMQ	JULY 24, 1968
0352	FEP07	SEPTEMBER 5, 1968
0353	FEP08	SEPTEMBER 6, 1968
0355	ACV680	SEPTEMBER 6, 1968
0357	NASA1	AUGUST 10, 1968
0358	STRESV	AUGUST 22, 1968
0363	FEM08	AUGUST 20, 1968
0367	GEMAS	JULY 17, 1968
0374	FEM03	JULY 19, 1968
0382	FEM01	JULY 22, 1968
0383	FEP03	AUGUST 23, 1968
0384	LPANT	AUGUST 21, 1968
0385	FEM02	JULY 19, 1968
0387	FEM05	JULY 19, 1968
0392	FEM06	JULY 19, 1968
0394	FEP05	AUGUST 23, 1968
0400	FEP06	AUGUST 23, 1968
0406	FEM04	JULY 31, 1968
0409	FEP4A	AUGUST 23, 1968
0410	FEPXX	AUGUST 23, 1968
0411	LAP	AUGUST 26, 1968
0412	FEP04	AUGUST 28, 1968
0438	TRAP	JULY 17, 1968
0439	DISTIL	JULY 16, 1968
0440	STRES3	JULY 17, 1968
0444	WEDDLE	JULY 17, 1968
0445	GEMAS5	JULY 16, 1968
0446	FEM07	JULY 23, 1968
0447	STRES2	JULY 18, 1968
0448	PATRAM	AUGUST 10, 1968
0451	ELAPSE	JULY 23, 1968
0452	ZTRANS	JULY 23, 1968
0453	ZIMQ	JULY 25, 1968
0465	PATRV	AUGUST 10, 1968
0469	NOPT	AUGUST 8, 1968

M&E

SYSTEM FOR SEGMENTING PROGRAMS

by

Thomas R. Harbron

Director

Anderson College Computing Center

Anderson, Indiana 46011

COMMON meeting

April 8 - 10, 1968

Chicago, Illinois

System For Segmenting Programs

Thomas R. Harbron

This is a system for segmenting large Fortran programs into load-on-call subprograms for running under the 1620 Monitor I System. If the following steps are carefully followed, a working, segmented program can be developed in a minimum of time.

I. Compile the original program to get the symbol table. For large programs, the symbol table will sometimes fill up before the compilation is completed. This is usually no problem as the most important part of the symbol table is that giving the names of all variables in the program. The statement number portion of the symbol table may be used as a guide in step (III).

II. Develop a "Common" statement that includes all variables in the program. If the cards from the symbol table are available, this may be done automatically as follows:

A. Remove all the cards containing variable names from the symbol table deck.

B. Place a blank card after the last variable card.

C. Execute a Fortran program named "COMMON" which is stored on disk. The variable deck formed in Steps A and B is used for input data. (See appendix A for listing of "COMMON").

D. A Common statement or statements will be produced using the names from the symbol table cards. This eliminates keypunch errors.

III. Break the original program into manageable sections using the statement number portion of the symbol table as a guide if available. A manageable section is defined as one small enough to fit. This size depends upon the number and size of library subroutines needed, the size of the common area, etc. Usually 40-100 Fortran statements will be maximum.

Avoid breaking loops where possible. If a "DO" loop must be broken it will be necessary to rewrite it as an "IF" loop.

IV. Compile each segment as a subroutine. Include the "DIMENSION" statement from the original program, and the "COMMON" statement(s) from step II. Remove all Format statements.

In general the following error messages will be generated:

Error 58 - No "RETURN" statement

Error 59 - Statement number(s) missing

If any other error messages appear, the conditions causing them should be eliminated at this point.

The "cores used" and "next common" messages should be used to determine whether or not the segment is manageable. The table in figure one (1) may be helpful in this respect.

The mainline program will usually require about 500 core positions. A table showing the core requirements for the library subroutines can be found on page 126 of the Monitor I manual. In addition to those used by the program, a subroutine named "FLIPER" is also loaded to handle the load

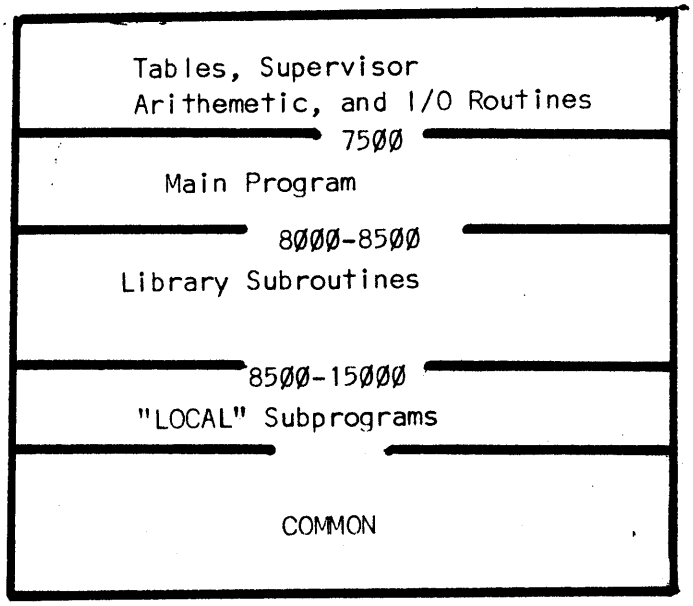


Figure 1a

Core Positions	Use
00000 - 07500	Tables, supervisor, arith. & I/O
07500 - 08212	Mainline Program
08212 - 08270	Library Subroutine #16 (ABS)
08270 - 98796	" " #15 (SQRT)
08796 - 09676	" " #12 (COS)
09676 - 10738	" " #02 (EXP)
10738 - 11540	" " #01 (LOG)
11540 - 11992	FLIPER
11992 - 16312	Segment 1
11992 - 14272	Segment 2
11992 - 17382	Segment 3
11992 - 17332	Segment 4
11992 - 16384	Segment 5
17480 - 19999	COMMON

Figure 1b

on call subprograms. The core requirements of FLIPER vary, but will usually be between 300 and 700. The table in figure 1b shows core allocations for a particular segmented program. A similar one should be made at this time for the program to be segmented.

Notice that the largest segment (#3) comes within 98 positions of the COMMON area. Had it been much larger, it would not have fit.

At this point if any segment appears to be too large, the program should be re-segmented and the trial compilations repeated.

V. The linkages between the segments must now be developed. The principle function of the main program will be to facilitate the linking.

A. Working from the error lists generated in Step IV, locate each missing statement number in the original program. If it is a FORMAT statement, write the word "format" after the statement number on the error list. For all other kinds of statements, write the number of the segment in which that statement is now located.

B. For each segment, make a list of statements in that segment that are referenced by statements in other segments. If the first executable statement of a segment is not numbered, give it a number and add it to the list. Number these statement numbers as shown in Figure 2.

Entry Point	Stmt Nbr
1	3204
2	7916
3	17
4	18
5	633

Figure 2

These will represent the 5 possible entry points to a particular segment.

C. Returning to the error list, place the entry point number after the segment number for each statement number. You now should have a segment number and entry point number for each statement number on the error list except FORMAT statements.

D. Add two fixed point variables to the Common statement (such as LINK1 AND LINK2).

E. To each segment add the following statements for each missing non-format staement:

```
XXXXLINK1 = SN
      LINK2 = EP
      RETURN
```

Where XXXX is the statement number, SN is the corresponding segment number, and EP is the corresponding entry point number.

F. Add a computed GO TO statement to each segment unless there is only one entry point and that is the first executable statement of the segment.

The computed GO TO should immediately follow the COMMON statement. The variable of the computed GO TO will be LINK2. The statement numbers will correspond to the list made up in part B.

G. Insert the required FORMAT statements in each segment. Some FORMAT statements may appear in several segments.

H. Code the mainline program. It will generally follow the pattern shown in Figure 3.

```
      DIMENSION  - - -
      COMMON   - - -
      LINK2 = 1
    1 CALL name 1
   90 GO TO ( 1,2,3,4,5,---------), LINK1
    2 CALL name 2
      GO TO 90
    3 CALL name 3
      GO TO 90
          '
          '
          '
          '
        END
```

Figure 3

Basically the mainline program usues LINK1 to route control to the proper segment. LINK2 must be defined before the first segment is called unless the first segment has no computed GO TO statement.

I. It may be more efficient to place certain statements from the original program into the mainline program. Statements to consider for this are computed GO TOs, STOPs, and CALL EXITs. Each of these statements placed in the mainline program will have a distinct value of LINK1 associated with it.

J. If library subroutines are needed by any of the segments, it will be necessary to place a dummy statement in the mainline program to avoid an "L7" error. As an example, the following

statement would load the LOG, EXP, subscripting, COS, SQRT, and ABS subroutines. A(I+1) = ABS(COS(Ø.))+SQRT(LOG(1.)+EXP(Ø.)) This dummy statement should be placed where it can not be executed. It is also possible to cause proper library subroutine loading by punching the correct indicators into the header record of the mainline program.

VI. Compile all segments as Subroutine Subprograms. Load them on disk or punch object decks. Compile and execute the mainline program. Be sure to include the *LOCAL Control Card immediately after the source deck. Also note that the number of LOCAL records must be given in the FORX or XEQS Control Cards.

It is helpful to turn SS1 on when the "EXECUTION" message is typed. This will cause a core use table to be typed out as the segments are loaded. If an "overlap" message appears at this time it will be necessary to return to Step III.

This system has been used to segment several programs that require over 100,000 core positions. These programs are now running on a 20K system with no difficulties. In one instance a program that was written in three segments for a 60K 1620 was broken down into seventeen segments. Timings on this program show that it ran slightly faster on the 20K machine, in spite of the segmenting. This was probably due to a difference in floating point hardware.

One difficulty not handled by this system is that of reading data in with an "H" specification and later printing out the same data from the same "H" specification. If the input and output statements are in different segments, the data will not be transmitted between the segments. The remedy for this is to change the "H" specification to an "A" specification.

APPENDIX A

"COMMON" LISTING

```
*LDISKCOMMON
      DIMENSION K(323),N(6)
    9 I=1
    1 READ 2,N
    2 FORMAT(6X,6A1)
      DO 3 J=1,6
      IF(N(J)-4000)3,3,4
    3 CONTINUE
      GO TO 7
    4 K(I)=N(J)
      I=I+1
      J=J+1
      IF(J-6)4,4,5
    5 K(I)=2300
      I=I+1
      IF(I-317)1,1,7
    7 I=I-2
      PUNCH 6,(K(J),J=1,I)
    6 FORMAT(6X,7HCOMMON ,59A1/4(5X,1H1,66A1/))
      IF(I-315)8,8,9
    8 CALL EXIT
      END
```

SCAT4: A Binary Synchronous Communication Subroutine for Conversational Use

Presented at COMMON Meeting, September 9, 1968

Philadelphia, Pennsylvania

By

Elsa B. Horowitz - Research Staff Member

T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, N. Y. 10598
Tel: 914/945-1178

ABSTRACT

SCAT4 is a binary synchronous communications (BSC) subroutine designed specifically for computer-to-computer conversational communications. SCAT4 was designed to facilitate conversational communications between an 1130 and another computing system where fast response time and ease of operation were design criteria. SCAT4 provides facilities similar to SCAT2 with the addition of the ability to accept a return message in lieu of an acknowledgment. When not engaged in communications activity, both machines remain in read mode ready to accept a message. If a message is received, the user is notified through his error routine. The machines remain in read mode until a transmit is initiated by the user. SCAT4 is operable in either an automatic or a semi-automatic mode to permit either the fastest response time or the most user supervision of operation.

SCAT4: A Binary Synchronous Communication Subroutine for Conversational Use

INTRODUCTION

This paper describes a subroutine written for the 1130 to provide conversational communications. The specifications for SCAT4 are the result of decisions made by a communications committee set up at the Research Division to study the problem of conversational computer-to-computer communications.

The task of the committee at Research was to decide on standards for communication between computers using either what was being done by others or developing new methods if it felt they would better suit Research's purposes. There are two BSC methods of communication between computers. One is point-to-point, where only two computers are on the line at any one time. There is no normal master-slave relationship since whichever computer is sending is considered the master terminal until it releases the line. When one machine receives an acknowledgment from the other system, it becomes the master until it finishes transmitting its messages. It then releases the line by sending an EOT. Either system may then request the line. The second method of BSC transmission is multi-point, which means that there are three or more computers on the line at the same time. One computer is designated master and all others are slaves. The master may transmit to any slave or may call for any slave to transmit to the master. The slave receives only when selected and transmits only when polled. The master retains control at all times.

The Research Committee decided on the following standards:

1. Point-to-point communication using BSC.
2. 2000 baud facility.
3. EBCDIC code.
4. Full Transparent mode

It was decided that the software on all the systems would use the following conventions:

1. As soon as the message is received, the receiving computer will issue a start-write command to its channel or communication adapter.
2. If the user prepares a return message within the line turn around time, that return message will be acceptable in lieu of an acknowledgment.

3. If no message is available, a normal acknowledgment will be sent. This yields the minimum possible turn around time.

4. If an acknowledgment is transmitted by either system, the line is considered neutral and either system may then attempt to send a message without sending an inquiry request With this system, the only time that an ENQ-ACK procedure must be employed in the middle of communications activity is in the case of contention for the line or in error recovery procedures.

We became interested in implementing conversational mode because we felt that there is a whole class of problems that would benefit from having an interactive hookup between small computer (like an 1130) and a large machine (like the 360/67) When we started investigating the usage of such a system, we became aware of a dead period in the system which is the delay time that a user must wait between the time that the 1130 sends a message and the time that the answer arrives. In addition, we felt that the conventions designed into SCAT2 make the routine too cumbersome and time consuming for use in an interactive system. For example, using SCAT2, the shortest time delay is approximately 5 turn around times compared to 1 turn-around time for SCAT4.

There were other design considerations for SCAT4 besides time. Another problem with SCAT2 is the excessive user supervision of the subroutine. We felt that both machines should be in receive mode except when actually transmitting a message. This would free the user from resetting the receive function if it failed which would be necessary for SCAT2 operation. In SCAT4, after the communications facilities have been verified by the transmit-receive initial procedure, it is possible for both machines to be in receive mode except when actually transmitting a message. The user is, therefore, only responsible for requesting a transmit Otherwise both machines are in a quiescent or neutral state ready to receive a message When a machine receives a message, SCAT4 notifies the user's routines by transferring to the user's error routine.

At present most of the conventions used by SCAT2 are still in SCAT4, to maintain compatibility where possible. One error code has been added to indicate the reception of a message. A change buffer function has been added to allow the user the ability to separate input and output buffers, if so desired.

In the future, the routine will be modified to

1. pack and unpack characters for the user, and to
2. notify the user through his error routine when EOT, DLE EOT or ENQ characters are received and when dial-up requests are detected.

The subroutine is currently being employed in an experimental teleprocessing monitor system for the 1130 which will interact with a System/360 model 67 under TSS. This teleprocessing system is being implemented as part of a research project on simulation.

SUMMARY OF SCAT4 FEATURES

SCAT 4 expands the capabilities of SCAT2, which is intended to support bulk transfer of data from one machine to another. SCAT4 allows either machine to transmit a message without first closing and then re-initializing transmission. SCAT4 will accept either an acknowledgment or a return message as an acknowledgment of a previous transmission. In addition, the routine has both automatic and semi-automatic modes for handling acknowledgments.

In order to facilitate faster conversational use, SCAT4 automatically turns the line around after receiving a message. In either mode the user may transmit a return message instead of an acknowledgment. Under the automatic mode of operation, SCAT4 sends either the appropriate acknowledgment for the incoming message or a return message after 150 msec*, which is the line turn-around time. Under the semi-automatic mode, SCAT4 expects the user to specifically request either the transmission of an ACK, a NAK or a return message. This mode, which is almost identical to SCAT2 operation, gives the user the ability to check a message before proceeding to the next operation.

Another difference between SCAT2 and SCAT4 is that SCAT4 notifies the user that a message has been received while SCAT2 requires the user to test for the completion of a receive request.

SCAT2 and SCAT4 differ in contention handling procedures. SCAT2 provides contention handling only during the initial request for the line. In conversational usage it is possible that both machines could be in receive mode and could then simultaneously attempt to transmit. SCAT4 provides facilities to handle this condition by forcing one of the machines to remain in receive status. The user designates which machine is to be considered the master in cases of contention.

SCAT4 does not generate or inspect the header information. All necessary control characters must be present with the exception of DLE ETB/ETX in transparent mode only.

* Turn around time for 201A4 data set.

Synchronous Communications Adapter Subroutine - SCAT4

The SCAT4 Interrupt Service Subroutine controls the 1130 SCA during point-to-point operation in BSC mode and performs error checking on the data transmitted and received. A four digit control parameter directs the subroutine in the following:

Testing to determine if the previous operation has been completed.

Transmitting.

Receiving.

Turning the audible alarm on and off.

Enabling/disabling the Auto Answer Interrupt feature.

Disconnecting the station from the line.

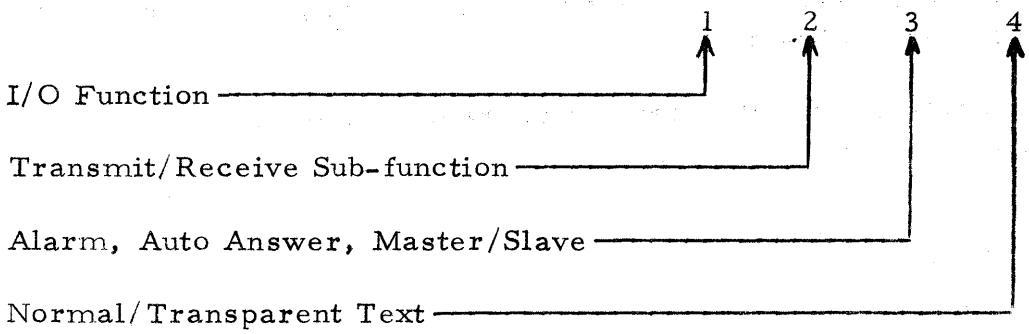
Calling Sequence

```
        LIBF        SCAT4
        DC          /XXXX (Control Parameter)
        DC          IOAR (I/O Area Address)
        DC          ERROR (Error Routine Address)
        .           .
        .           .
        .           .
```

Error	Return Link
	Error Routine
	BSC I ERROR
IOAR	Word Count
	I/O Area

Control Parameter

The control parameter consists of four hexadecimal digits which are used as shown below:



I/O Function

The I/O function digit specifies the operation to be performed by SCAT4 on the SCA. The functions, their associated digital values, and the required parameters are listed and described below:

Function	Digital Value	Required Parameters*
Test	0	Control
Auto Answer	1	Control, I/O Area**
Alarm	2	Control
Close	3	Control
Receive	4	Control, I/O Area, Error
Transmit Block	5	Control, I/O Area, Error
Transmit Text	6	Control, I/O Area, Error
Transmit End	7	Control, I/O Area, Error
Change Buffer	8	Control, I/O Area

* Any parameter not required for a particular function must be omitted.
** I/O Area parameter required only if function is Enable Auto Answer.

I/O function digit	Description
0	Test - Tests the Device Routine Busy indicator and branches to LIBF + 2 if the previous operation has not been completed, or to LIBF + 3 if the previous operation has been completed. It is possible to initiate a Test, Auto Answer, Alarm, or Close operation while any Transmit or Receive operation is in progress.
1	Auto Answer - Enable the automatic answer interrupt if digit 3 of the control parameter is zero; disables the automatic answer interrupt if digit 3 of the control parameter is non-zero. When an Auto Answer Request interrupt occurs, the location specified by the I/O area address is set to a non-zero value and the automatic answer interrupt is disabled.
2	Alarm - Turns on the audible alarm in the local system if digit 3 of the control parameter is zero; turns off the audible alarm if digit 3 of the control parameter is non-zero.
3	Close - Ends all operation on the SCA and disconnects the station from the line. On carrier lines that require a station to disconnect from the line automatically at the end of message transmission, the user must perform a Close operation within two minutes of the transmission of EOT.
4	Receive - Both an automatic and a semi-automatic mode of operation are provided. In the automatic mode the communications after the initial set-up can be handled by SCAT4 without user intervention. The user is still given the ability to over-ride the automatic operation to enable him to send a return message, to close transmission, to request a repeat transmission or to change buffers. In the semi-automatic mode, SCAT4 behaves almost identically to SCAT2. In this mode, the user must request all further operations. Digit 2 of the control parameter is used to specify which sub-function of Receive is requested.

Digit 2	Digit Value	Sub-function
	0	Receive Initial-semi-automatic mode
	1	Receive Continue
	2	Receive Repeat
	3	Receive Initial-automatic mode

- Receive Initial - Monitors the line for ENQ; upon receiving transmits ACKO and receives the first message.

  Automatic Mode - Sets up SCAT4 to automatically transmit after each incoming message the appropriate acknowledgment unless the user initiates another request.

  Semi-automatic Mode - Sets up SCAT4 to reverse the line after each incoming message but to wait for further directions from the user.

- Receive Continue - Transmits the correct positive acknowledgment for the current message and receives the next message when SCAT4 is in semi-automatic mode.

- Receive Repeat - Transmits NAK for the current message and receives the next message.

  When performing a Receive operation, the first word of the I/O area contains the maximum number of unpacked characters (word count) that can be read into that area.

  The entire message that is received is stored in the I/O area, including the SOH character and/or the STX character, (DLE STX, if Transparent text), the ETB character (DLE ETB, if Transparent text), or the ETX character (DLE ETX, if Transparent text). After the message has been received, the number of characters received, including control characters, is stored in the first word of the I/O area.

  All characters in the I/O area are unpacked and left-justified.

  If the record is received in Transparent text mode, SCAT4 deletes the second DLE character (inserted at the transmitting station) in each pair of DLE characters received.

  After receiving the block check characters SCAT4 automatically reverses the line. If no errors were found in the message and if automatic mode was selected, SCAT4 prepares to transmit the appropriate positive acknowledgment. During the line turnaround interval, the user may override the transmission of the acknowledgment. If no errors were found and if the semi-automatic mode was selected, SCAT 4 waits for further

instructions from the user. Upon receipt of a message SCAT4 branches to the user's error routine with $(0080)_{16}$ in the accumulator. The location of the input buffer is in the accumulator extension.

If the block check character (CRC-16) is found to be incorrect or if the message overflows the I/O area, SCAT4 transmits NAK and attempts to receive the message again. After eight unsuccessful attempts, SCAT4 branches to the user's error routine with an error code in the accumulator. (See Post-operation Error Detection.) If the user returns with a positive accumulator, SCAT4 transmits NAK and attempts to receive the message again (up to seven more attempts before branching to the user's error routine). If the user returns with a zero accumulator, SCAT4 performs a Close Operation.

If the user returns with a negative accumulator, SCAT4 clears the Device Routine Busy indicator and stores the number of characters, including control characters, received in the message in the first word of the I/O area, allowing the user to initiate a Receive Repeat or Receive Continue operation.

If a timeout occurs while receiving a message, SCAT4 monitors for ENQ and, when ENQ is received, transmits the last acknowledgment.

If the EOT character is received, SCAT4 clears the first word of the I/O area to zero and clears the Device Routine Busy indicator. The user should then initiate a Receive Initial, Transmit Initial, Transmit EOT, Transmit DLE EOT (Dial line only), or Close operation.

If DLE EOT (Disconnect Signal) is received, SCAT4 sets the first word of the I/O area to $FFFF_{16}$ and performs a Close operation.

5/6 Transmit Block/Text - There are four types of Transmit Block and Transmit Text operations. Digits 2 and 4 of the control parameter are used to specify which sub-function of Transmit Block/Text is requested.

Digit 2	Digital Value	Sub-Function
	0	Transmit Initial-Semi-Automatic
	1	Transmit Continue
	3	Transmit Initial-Automatic
Digit 4		
	0	Normal EBCDIC text
	non-zero	Full-Transparent text

- Transmit Initial Block/Text - Transmits ENQ, receives the acknowledgment (ACK0), transmits the message from the I/O area, transmits the CRC-16, and receives the acknowledgment (ACK1) or return message. (Semi-automatic or automatic mode).

- Transmit Initial Transparent Block/Text - Transmits ENQ, receives the acknowledgment (ACK0), transmits the message from the I/O area, transmits DLE ETB/DLE ETX, transmits the CRC-16 and receives the acknowledgment (ACK1) or return message.

- Transmit Continue Block/Text - Transmits the message from the I/O area, transmits the CRC-16, and receives the acknowledgment or return message.

- Transmit Continue Transparent Block/Text - Transmits the message from the I/O area, transmits DLE ETB/DLE ETX, transmits the CRC-16, and receives the acknowledgment or return message.

Contention exists when the two stations on a line simultaneously bid for control of the line.

SCAT4 provides a means to break contention. If the user wishes to be the master station in the event of contention, digit 3 of the control parameter must be zero. If the user wishes to be the slave station, digit 3 of the control parameter must be non-zero.

In a master station, when contention exists, SCAT4 re-transmits ENQ. After eight attempts, SCAT4 branches to the user's error routine with an error code (4000_{16}) in the accumulator. If the user returns from the error routine with a non-zero accumulator, SCAT4 attempts to break contention seven more times. If the

user returns with a zero accumulator, SCAT4 performs a Close operation.

In a slave station, when contention exits, SCAT4 branches to the user's error routine with an error code (4000_{16}) in the accumulator and, upon return from the error routine, performs a Close operation, allowing the user to initiate a Receive Initial operation.

When performing a Transmit Block/Text operation, the first word of the I/O area contains the number of characters in the message. The character count includes the control characters in the message. All characters in the I/O area are unpacked and left-justified. If the user wishes to start the message with a heading (optional), he must supply the SOH character as the first character of the message.

If there is text in the message, the text portion of the message follows the heading. When digit 4 of the control parameter is zero, the text is Normal EBCDIC text and must begin with STX and end with ETB/ETX. The user must supply these characters. When digit 4 of the control parameter is non-zero, the text is Full-Transparent text and must begin with DLE STX. The user must supply these characters. The ending characters, DLE ETB/ETX, are supplied by SCAT4. SCAT4 transmits a second DLE character after each DLE that is found in the Transparent text.

If a redundancy check of the heading separate from the text is desired, the heading must end with ETB. The ETB is supplied by the user.

The I/O area is not checked for misplaced or incorrect control characters.

SCAT4 transmits the 16-bit block check character (CRC-16) after the ETB/ETX is transmitted. The CRC-16 is generated by SCAT4.

When the proper acknowledgment is received, SCAT4 clears the Device Routine Busy indicator and returns to receive mode.

If SOH, STX or DLE STX is received in response to a message, SCAT4 puts the return message into the last mentioned I/O buffer. If the user does not want the return message to overlay the output buffer, he must initiate a Change Buffer request immediately after requesting a transmit operation.

If ACK0 is not received in response to the initial ENQ, ENQ is re-transmitted, except when contention exists and the station is a slave station.

If NAK is received in response to a message, the message is re-transmitted.

If EOT is received in response to ENQ or to a message, SCAT4 clears the first word of the I/O area to zero and clears the Device Routine Busy indicator, allowing the user to initiate a Receive Initial, Transmit Initial, Transmit End, or Close operation.

If DLE EOT is received in response to ENQ or to a message, SCAT4 sets the first word of the I/O area to $FFFF_{16}$ and performs a Close operation.

If anything other than EOT, DLE EOT, NAK, SOH, STX, DLE STX or a positive acknowledgment is received or if a Receive timeout occurred after the message was transmitted; SCAT4 transmits ENQ. If an incorrect acknowledgment is received before a receive timeout occurred, the message is re-transmitted, otherwise an ENQ is transmitted.

If after eight attempts, the proper positive acknowledgment is not received, SCAT4 branches to the user's error routine with an error code in the accumulator (see Post-operation Error Detection). If the user returns from the error routine with a positive accumulator, the transmission is attempted seven more times. If the user returns with a zero accumulator, SCAT4 performs a Close operation. If the user returns with a negative accumulator, SCAT4 continues as if the proper positive acknowledgment had been received.

7 Transmit End - The Transmit End operation can be one of two types. Digit 2 of the control parameter is used to specify which sub-function of Transmit End is requested.

Digit 2	Digital Value	Sub-function
	0	Transmit EOT
	1	Transmit DLE EOT

Transmit EOT--Transmits EOT, then reacts according to the reply received.

Transmit DLE EOT--Transmits DLE EOT, then performed a Close operation.

Digit 4 of the control parameter is used to specify the action to be taken on a no response condition during a Transmit EOT operation.

Digit 4	Digital Value	Sub-function
	0	Close
	non-zero	Do NOT Close

On a Transmit EOT operation, SCAT4 transmits EOT and receives the response. If there is no response (i.e., a Receive timeout occurs), SCAT4 performs a Close operation if digit 4 of the control parameter is zero. If digit 4 of the control parameter is non-zero, SCAT4 clears the Device Routine Busy indicator and starts the 3-second timer, allowing the user to initiate a Receive Initial, Transmit Initial, or Close operation.

If the response is DLE EOT, SCAT4 sets the first word of the I/O area to $FFFF_{16}$ and performs a Close operation.

If the response is EOT, SCAT4 stores an EOT character in the location specified by the I/O area address and clears the Device Routine Busy indicator, allowing the user to initiate a Transmit Initial, Transmit DLE EOT, or Close operation. If the response is ENQ, SCAT4 stores ENQ in the location specified by the I/O area address and clears the Device Routine Busy indicator, allowing the user to initiate a Receive Continue or Receive Repeat operation.

If a response other than DLE EOT, EOT, or ENQ is received, SCAT4 re-transmits EOT. After eight unsuccessful attempts, SCAT4 branches to the user's error routine. If the user returns with a non-zero accumulator transmission is attempted seven more times. If the user returns with a zero accumulator, SCAT4 performs a Close operation.

8 Change Buffer - Allows the user to separate input and output buffers or to provide a limited multiple buffering capability for input. This operation may be requested at any time without testing for the completion of the ongoing operation. Any new request involving buffer allocation will override the previous Change Buffer request.

The buffer is changed when the next text message is received. If a transmit operation is in effect when the Change Buffer request is initiated then the new buffer will be used if a return text message arrives or if an unsolicated text message arrives. If a receive operation is in effect when a Change Buffer request is initiated the current operation is unaffected but the next incoming message will be inserted into the new buffer.

Error Handling

For a description of error handling procedures, refer to General Error Handling Procedures in the publication IBM 1130 Subroutine Library.

Pre-operation Error Detection

The following conditions result in pre-operation error action (accumulator settings are shown in parentheses):

Invalid function code (8001_{16})

Invalid sub-function code for some Transmit or Receive operation (8001_{16})

Invalid word count (8001_{16})

Post-operation Error Detection

The following conditions result in a branch to the user's error routine (accumulator settings are shown in parentheses):

Data set not ready (8000_{16})

Contention exists (4000_{16})

3-second timeout occurred while receiving a message or monitoring for ENQ, or ENQ not received while monitoring for ENQ (2000_{16})

I/O area overflow (1000_{16})

Block check character (CRC-16) in error (0800_{16})

Receive timeout occurred after transmitting a message or ENQ, or invalid sequence received in response to a message or ENQ (0200_{16})

NAK received, or the incorrect acknowledgment received following a Receive timeout (0400_{16})

Incorrect acknowledgment received with no Receive timeout (0100_{16})

Message received (0800_{16}) EXT has location of input buffer.

HIGH PRECISION INTEGER SUBPROGRAMS FOR IBM 1130 COMMERCIAL APPLICATIONS

by

B. J. SWAIN

The Shawinigan Engineering Company Limited
Box 3010, Station B, Montreal
Canada

July 1968

CONTENTS

Contents cont'd

Contents cont'd

Contents cont'd

1. The Use of Fortran for Commercial Applications

The use of Fortran for commercial applications on the IBM 1130 presents several problems. These problems have to be solved since Fortran is the only compiler supported for the IBM 1130 and it is usually necessary to produce a limited amount of commercial reports despite the fact that the typical IBM 1130 installation is intended for scientific work. The principal problems presented by the use of Fortran for commercial work are the following:

a) Manipulation of character strings

b) Editing problems, for example, floating dollar signs, cheque protection

c) Round off for cross footing purposes

d) Recognition of zone over-punches

e) Selection of the alternate stacker

f) Input/output speed

Various subroutine packages have been evolved to eliminate these problems. The one described in this paper was developed by the Shawinigan Engineering Company Limited in 1967. In common with other subprogram packages which permit the use of Fortran for commercial purposes it is based on FORCOM (IBM 1620 library ol.6.051). However, it incorporates a number of features which have not been published elsewhere.

2. Program Philosophy

Using a commercial subroutine package the programming philosophy adopted is as follows:

a) Read card and store its card image in an alphabetic array

b) Test the card type

c) Extract numeric and alphabetic data, convert numeric data to the form required for processing

2. Programming Philosophy cont'd

d) Process the data, build a printer image for output

e) Print

This philosophy has been followed in the commercial subroutine package now being presented. It is expected that the writer's overlapped output subroutine will be used (see IBM 1130 Library). Therefore the printer image need only be partially built before printing.

3. Data Formats

The following data formats have been used in the package:

a) For alphabetic information a COMET array has been employed. COMET array is defined as an array having a single subscripted integer name. It contains 2 alphabetic characters per word, in EBCDIC coding. The * ONE WORD INTEGERS option must be used to produce maximum packing density. A COMET array can be printed by the use of A2 field specification in a FORMAT statement. (This mode of storage is the one used in the contributed program package COMET. Where possible the subprograms described have been made compatible with the COMET package written by J.R. Hurley and others of IBM.)

b) A standard Fortran integer variable is used for counters and for data whose value does not exceed the range -32,768 to +32,767.

c) For numeric data which cannot be contained in an integer variable, e.g. amounts of money, the high precision integer variable has been defined. This has a real variable name, subscript or non-subscript. The use of standard precision is required. This permits the use of 2 words or 32 bits giving a range of -2,147,483,648 to +2,147,483,647. This range is generally adequate to express money amounts to one cent or one mil.

4. Card Read or Punch

Subroutines have been provided to read a card and store its card image or to punch a card image from storage into a card. These subroutines permit the complete EBCDIC character set to be converted.

QREAD
QPNCH

5. Mode Conversion

Subroutines have been provided to convert alphabetic information contained in a COMET array to integer or high precision integer. The conversion function permits conversion from integer to high precesion integer or fom high precision integer to integer. The EDIT subroutine permits either of these forms to be edited into an alphabetic COMET array into which an EDIT mask has previously been inserted. In this way floating dollar signs, cheque protection and credit indication can be handled. If simple conversion to alphabetic form is required subroutines are available to permit this to be done.

PDATA
IDATA
PCØNV
ICØNV
PEDIT
IEDIT
PPASS
IPASS
PPASX

6. Arithmetic Calculations

In view of the fact that the data contained in Fortran real variables is in fact in the form of integers then normal Fortran arithmetic cannot be used. Instead a series of arithmetic functions have been provided. For example, the statement: R = PMPY(P,Q) causes P and Q to be multiplied and the result stored in R. The use of functions rather than subroutines in this instance permits a series of arithmetic operations to be performed in a single Fortran statement, for example, the calculation: $D = \frac{A + B}{C}$ is handled by the single statement: D = PDIV(PADD(A,B),C)

PADD
PSUB
PDIV
PDIVT
PMØD
PMINO
PMAXO
PLIST
PABS

7. Overlapped Read

Provided it is not necessary to punch information into cards which have been previously read then additional reader speed can be obtained. This is achieved by reading a card into a buffer ahead of the time at which the information is required. At the time the information is called for in the users program it is converted and moved to the required storage in core. Meanwhile reading of the next card takes place simultaneously with processing the data on the previous card.

RØPEN
XREAD
RHØLD

8. Input/Output on Console

Subroutines are provided to support communication with the console keyboard and typewriter. Information to be transmitted to these devices is held in core, in alphabetic form, in COMET arrays.

QTYPE
QTWYR
QWRTY

9. Comparisons

Comparison functions are provided for comparing 2 high precision integers, two alphabetic arrays, to determine if an laphabetic array is blank or if a column of an alphabetic array contains a zone punch. The comparison function for alphabetic arrays has blank at the low end of the collating sequence.

IPCMP
IQCMP
IQBLK
IQZØN

10. Logical Operations

Functions are provided to permit logical operation on 16 bit arguments. The use of the AND function in conjunction with a mask will permit the extraction of part of the data contained in a word. The OR function can be used for packing information. The ROTATE function permits a word to be shifted until any desired

IAND
IØR
IRQL
IALS
IARS

10. Comparisons cont'd

bit is in the high order or low order position. Thus tests on individual bits can be performed or bit patterns treated as arithmetical data. A function is provided to extract a single character of a COMET array or to place a single character into a COMET array.

IPIK
PACK

11. Record Blocking

It has been found very desirable to block records which are to be transferred to or from the disk pack of the IBM 1130. In this way several records may be handled with a single disk WRITE or READ with a corresponding increase in speed. A set of subroutines has been provided to perform the operations required to block several logical records into a longer physical record and to read or write the physical record as required.

DØPN
DØPEN
DPUT
DGET
DCLØS
DPUT1
DGET1
DFIND

12. Comparison with IBM Commercial Subroutine Package

The principal advantage to be gained in the use of this subroutine package in comparion with the IBM Commercial Subroutine Package is the speed of coding and the length of source program required to perform the given task. As an example, a program involving card reading, disk reading and writing, arithmetic calculations and output on the printer has been written using these commercial subroutines and the commercial subroutine package. The number of Fortran statements required is much smaller using these commercial subroutines. The core storage required is somewhat larger but would also be decreased if the program were more complex. The execution time on an IBM 1130, having 8k core, 1132 printer and 1442 Model 6 card reader, is unaffected. The following table shows this comparison:

12. Comparison with IBM Commercial Subroutine Package cont'd

	CSP III	HP
Main Program Statement	64	29
Main Program Core	998	586
Total Core	4706	5288
Execution Time	1 min. 13 sec.	1 min. 17 sec.

Programming Languages Used

Wherever possible, programs have been written in FORTRAN. This may lead to problems in the use of high speed peripheral devices. The assistance of Mr W. Baden of Marshall Communications, Santa Ana, California is acknowledged. He has re-written the disk blocking subroutines in Assembler language, with a resulting increase in speed and decrease in core requirement.

Card Read Subprogram

CALL QREAD(IA,M)

To read a card and to store its card image.

IA = COMEN array in which characters are to be stored.

M = Number of columns to be stored

The first M columns of data are transferred by execution of this statement. Card reading and conversion to EBCDIC are overlapped by use of this subprogram. However control is held in the suborutine until reading the card is completed. XREAD (see below) permits overlapping of card reading and processing.

Note:

Since QREAD uses the IBM supplied CARDO subroutine rather than the normal CARDZ subroutine, then all card reading in a program must be done using this subroutine. Use of this subroutine allows all character codes to be recognized, including $\overset{+}{0}$ and $\overset{-}{0}$ which are not recognized by Fortran READ statements.

The *IØCS record should not contain CARD.

Data Definition and Mode Conversion Subprograms

CALL PDATA(IA,J,M,R)

To convert the characters appearing in a COMET array to a high precision integer variable.

IA	=	COMET array containing data to be converted
J	=	Character number within IA at which conversion is to start (left hand end)
M	=	Number of characters to be converted
R	=	Variable to contain returned value

In interpreting the COMET array IA, blanks will be taken as zeros. A negative number is designated by either a leading minus sign, or by an 11 zone overpunch in the units position, thus converting that character into an alphabetic character. The existence of other characters which are not numeric or blank will cause an error message to be printed, and R to be set to zero.

CALL IDATA(IA,J,M,I)

To convert the characters appearing in a COMET array to an integer variable.

IA	=	COMET array containing characters to be converted
J	=	Column number within IA at which conversion is to start (left hand end)
M	=	Number of characters to be converted

Data Definition and Mode Conversion Subprograms (cont'd)

CALL IDATA (cont'd)

I = Variable to contain returned value

Blanks and non-numeric characters are handled in the same way as described under PDATA.

R = PCONV(I)

To convert an integer variable to the high precision integer mode.

I = Previously defined integer variable or constant to be converted.

R = Returned value

I = ICONV(P)

To convert a high precision integer variable to integer mode.

P = Previously defined high precision integer variable to be converted.

I = Returned value. If the magnitude of P is too large to be stored in I, then the high order digits are lost. No error message is produced.

CALL PEDIT(P,IA,J,M)

To edit a high precision integer variable into a COMET array

P = Previously defined high precision integer variable to be combined with the COMET array.

Data Definition and Mode Conversion Subprograms (cont'd)

CALL PEDIT (cont'd)

IA = Previously defined COMET array, containing an edit mask

J = Column number within IA marking the left-most end of the edit mask

M = Number of columns comprising the edit mask

The final appearance of the array IA depends upon the edit mask which existed before calling this subroutine. The following is a table of characters used to make up the mask, and their respective functions.

Control Character	Function
b(blank)	This character is replaced by a character from the high precision integer variable P.
0(zero)	This character indicates zero suppression, and is replaced by a character from the high precision integer variable P. The position of this character indicates the right-most limit of zero suppression.
.(decimal)	This character remains in the mask field where placed. It is considered an alphabetic character, and may not be zero suppressed.

Data Definition and Mode Conversion Subprograms (cont'd)

CALL PEDIT (cont'd)

Control Character	Function
,(comma)	This character remains in the mask field where placed. However, if zero suppression is requested, this character will be removed if it is to the left of the last character to be zero suppressed.
CR(credit)	These two characters can be placed in the two right-most positions of the mask field. They are undisturbed if the source field is negative. If the source field is positive, the characters C and R are blanked out. Whether CR is blanked out or not, no data will be edited into these positions, where CR is present, but rather into the edit characters to the left. The letters C and R may be used in the remaining of the edit mask where they will be treated as normal alphabetic characters, without being subject to sign control.
-(minus)	This character is handled similarly to CR in the right-most position of the mask field. Otherwise it is treated as an alphabetic character.

Data Definition and Mode Conversion Subprograms (cont'd)

CALL PEDIT (cont'd)

Control Character	Function
-(minus) cont'd	Note: If neither CR or - appears at the right hand end of the mask field, the negative sign on negative numbers will be lost. (See PPASS and PPASX for other forms of negative sign control)
*(asterisk)	This character operates the same as the 0(zero) for zero suppression, except that asterisks are inserted in the field rather than blanks in high order positions, providing asterisk check protection.
$(floating dollar sign)	This character has the same effect as the 0(zero) for zero suppression, except that a $ is inserted to the left of the first significant character found.
'(apostrophe)	This character is removed and replaced by a blank, which then remains as an alphabetic character.

If insufficient space is available to insert all significant digits from the variable P into the edit mask,

Data Definition and Mode Conversion Subprograms (cont'd)

the complete edit mask will be replaced by asterisks.

CALL IEDIT (I,IA,J,M)

To edit an integer variable into a COMET array

I = Previously defined integer variable, to be combined with the COMET array.

IA = Previously defined COMET array, containing an edit mask.

J = Column number within IA marking the left-most character of the edit mask.

M = Number of columns comprising the edit mask.

Except for the mode of the variable to be combined with the COMET array, the function of this subprogram is identical to PEDIT.

CALL PPASS (P,IA,J,M)

To convert the high precision integer variable P to its EBCDIC equivalent, and store the resulting character string in the COMET array IA. Leading zeros are suppressed. If P is negative, a leading minus sign is added to the resulting character string. If insufficient field width has been allowed, high order

Data Definition and Mode Conversion Subprograms (cont'd)

CALL PPASS (cont'd)

characters are lost. No error indications is given.

Note: The identical effect to PPASS can be obtained by the use of PEDIT, with indications of error. However PPASS is shorter and faster, and requires no mask to control the transfer.

P = Previously defined high-precision integer variable, to be converted to EBCDIC.

IA = COMET array to contain EBCDIC equivalent of P.

J = Character number within IA to mark lefthand end of resulting field.

M = Field width in IA of resulting EBCDIC character string.

CALL IPASS (I,IA,J,M)

To convert the integer variable I to its EBCDIC equivalent, and store the resulting character string in the COMET array IA, Leading zeros are suppressed. If P is negative, a leading minus sign is added to the resulting character string. If insufficient field width has been allowed, high order characters are lost. No error indication is given.

I = Previously defined integer variable to be converted to EBCDIC.

IA = COMET array to contain EBCDIC equivalent of P.

Data Definition and Mode Conversion Subprograms (cont'd)

CALL IPASS (cont'd)

J = Character number within IA to mark lefthand end of resulting field.

M = Field width in IA of resulting EBCDIC character string.

CALL PPASX (P,IA,L,M)

To convert the high precision integer variable P to its EBCDIC equivalent, and store the resulting character string in the COMET array IA. The effect of this subprogram is identical to PPASS, except as follows:

1) Leading zeros are not suppressed.

2) If P is negative, an 11 zone code is added to the low order character of the resulting field in IA.

This subroutine will normally be used to convert high precision integers to EBCDIC for punching. It should not be used if the resulting COMET array is to be printed, as $\bar{0}$ is not recognizable by the printer.

Arithmetic Subprograms

R = PMPY (P,Q)

To multiply P x Q and store the result in R as a high precision integer variable.

R = PADD (P,Q)

To add P and Q and store the result in R as a high precision integer variable.

R = PSUB (P,Q)

To subtract Q from P and store the result in R as a high precision integer variable.

R = PDIV (P,Q)

To divide P by Q and store the result in R as a high precision integer variable. The result is rounded to the nearest integer.

R = PDIVT (P,Q)

To divide P by Q and store the result in R as a high precision integer variable. The result is truncated to the nearest integer, and the remainder after division is lost.

R = PMØD (P,Q)

Sets R to the remainder after dividing P by Q.

R = PMINO (P,Q)

Sets R to the lesser of P and Q.

Arithmetic Subprograms cont'd

R = PMAXO (P,Q)

Sets R to the larger of P and Q

R = PABS (P)

Sets R to the absolute value of P

Overlapped Read Subroutines

The purpose of these subprograms is to allow overlap of card reading and processing. They should not be used if card punching is required on the same card as was read.

CALL RØPEN (IRBUF,LR)

Opens Read Buffer IRBUF, having length LR words. The first card in the read hopper is read and is stored in the buffer, one character per word, in card code format. IRBUF should appear in a DIMENSION statement, having a size LR+1, and should not be used for any other purpose. LR should be selected large enough that as many columns as are required in any card read operation in the program can be stored.

CALL XREAD (IA,N)

Transfers the data from IRBUF to IA, converted to EBCDIC, as required for a COMET array. N, which should be even, is the number of characters transferred. If N is odd, N+1 characters are transferred, the remaining character being filled with a blank. After transfer of characters, a new card is then read into IRBUF in overlapped mode, ready for the next data transfer. If XREAD is called before RØPEN or RHØLD, then the computer waits with AA01 displayed in the accumulator.

CALL RHØLD (IRBUF,LR)

Identical to RØPEN, but does not read a new card. Used to

Overlapped Read Subroutines (cont'd)

CALL RHØLD (cont'd)

re-initialize the subroutine on entering a new link of a program involving linked overlays.

These subroutines use the library program CARDO.

Overlapped Typewriter Input/Output Subroutines

CALL QTYPE (IA,N)

Function

To accept data from the 1131 Console keyboard and to present this data in A2 format to the array specified.

Parameter Description

IA — A one word integer array which is to receive the data from the console keyboard in A2 format. The maximum permissible length is 80 positions.

N — This is an integer variable or constant which defines the number of characters to be entered through the console keyboard. This figure is a maximum figure to be anticipated for this keyboard operation.

Detailed Description

This subroutine accepts information from the console keyboard, typing each character as it is entered, accepting the entire sequence of characters until either the character count as defined by length is reached or an EOF character is entered. This routine provides for a carrier return and line feed prior to each entry. A maximum of 80 characters is permitted. The output is presented in A2 format. All EBCDIC characters are accepted. This routine uses IBM LIBF TYPEO subroutine and therefore supports all its features with respect to backspacing and field cancellation. The

Overlapped Typewriter Input/Output Subroutines (Cont'd)

CALL QTYPE (cont'd)

entire internal 80 character buffer is cleared to blanks prior to each entry of keyboard data.

CALL QTYWR (IA,N)

Function

To print on the 1131 Console Printer the contents of the A2 array containing the message. This routine must be used if the program also contains QTYPE and console typewriter output is desired.

Parameter Description

IA The name of a one word integer array which contains the message to be printed on the console typewriter in A2 data format.

N The integer variable or constant which specifies number of characters to be transmitted. N should be even. If it is odd then N + 1 characters will be transmitted.

Detailed Description

This routine uses the IBM LIBF TYPEO subroutine. It accepts data from a one word integer array which data must be in A2 format. This means packed two characters per word. A maximum

Overlapped Typewriter Input/Output Subroutines (cont'd)

CALL QTYWR (cont'd)

of 128 characters or 64 words is a restriction to this routine. Carrier return and restore are provided before each message is typed on the console typewriter.

CALL QWRTY (IA,N)

Function

To accept data from a one word integer array which is in A2 format and display this information on the 1131 Console Typewriter.

Parameter Description

Same as QTYWR.

Detailed Description

This routine is similar to QTYWR except that the IBM LIBF WRTYO routine is also used. It must not be used if QTYPE is also used in the same program for data input on the Console Keyboard.

Errors

The appendix B of the IBM 1130 subroutine library listing applies to the QWRTY error conditions.

Overlapped Typewriter Input/Output Subroutines (cont'd)

Remarks

This subroutine is designed for inclusion in a card input, printer output program to provide error message capability on the 1131 Console Printer. In this configuration, it uses the least amount of core storage of any of the typewriter routines.

CALL IØND no parameters.

Function

To permit all input/output operations to come to an end before a pause or stop statement is entered.

Parameter Description

None

Detailed Description

This routine is required because of the interrupt nature of the input and output subroutines. When it is desired to display information in the accumulator on the console it is necessary that all pending input or output operations be brought to an end before the displayed information will remain in the accumulator. It is therefore necessary to use the call IOND prior to any pause or stop statement in a FORTRAN program.

(IDEAL and CSP II subprogram)

Compare Subprograms

I = IPCMP (P,Q)

To compare the two high precision integer variables.

P = Previously defined high precision integer.

Q = Variable to be compared.

I = Variable to contain returned value.

Sets to 1 if P is greater than Q.

Sets to 0 if P equals Q.

Sets to -1 if P is less than Q.

I = IQCMP (IA,J,IB,L,M)

To compare two alphameric arrays IA and IB.

IA = Previously defined COMET array.

J = Column number within IA at which comparison is to start (left hand end)

IB = Previously defined COMET array.

L = Column number within IB at which comparison is to start (left hand end)

M = Maximum number of characters to be compared.

I = Variable to contain returned value. Comparison begins at column J of IA and L of IB, and continues for a maximum of M columns.

I is set equal to 1 if a column is found in the array A which contains a character which is higher in the collating sequence than in the corresponding column in B. I is set equal to -1 if a column is found in A which has a character which is lower

Compare Subprograms (cont'd)

I = IQCMP (cont'd)

in the collating sequence than the corresponding character in B. I is set equal to 0 if the arrays A and B are found to be identical in the columns examined. The collating sequence in ascending order is as follows:

blank, ., (, +, &, $, *,), -, /, ,, ', =, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

I = IQBLK (IA,J,M)

Tests the COMET array IA for alphabetic blanks.

IA = Previously defined COMET array

J = Column number within IA at which checking is to start

M = Number of columns to be checked

I = Variable to contain returned value

Set to -1 if array is lower in collating sequence than alphabetic blanks (only integer numbers can cause this condition)

Set to 0 if array is blank

Set to 1 if array is higher in collating sequence than alphabetic blanks

Trace Subprogram

Trace subprogram is provided to assist checking, by printing out the results of intermediate calculations.

CALL PLIST (P)

To cause the value of P to be printed in standard format.

Card Punch Subprogram

CALL QPNCH (IA,M)

To punch a COMET array

IA = Previously defined COMET array

M = Number of characters to be punched

Up to 80 characters (one card) can be punched by use of this subroutine. The remainder of the card will be left blank. QPNCH uses the CARDO subroutine. See note in description of QREAD. QPNCH is a secondary entry point to QREAD.

Shifting and Logical Operation Subprograms

K = IALS(I,J)

To shift left an integer word.

I = Number of bits word is shifted

J = Word to be shifted

Low order bits are filled with zeros.

High order bits are lost.

K = IARS(I,J)

To shift right an integer word.

I = Number of bits word is shifted

J = Word to be shifted

Low order bits are lost.

High order bits are filled with sign bit.

Sign bit is not affected.

K = IRQL(I,J)

To rotate left an integer word.

I = Number of bits word is to be rotated

J = Word to be rotated

High order bits are replaced at low order end of word.

K = IAND(I,J)

To form the logical intersection of two integer words.

I, J = Words to be combined

Shifting and Logical Operation Subprograms (cont'd)

K = IØR(I,J)

To form the logical union of two integer words.

I, J = Words to be combined

K = IPIK(IA,J)

To extract one character from a COMET array.

IA = COMET array

J = Character number to be extracted

K = Returned value. K contains the EBCDIC equivalent of the character in the right hand eight bits, and zeros in the left hand eight bits.

This subprogram uses the COMET subprogram QGRAB.

CALL PACK(I,IA,J)

To pack one character into a COMET array.

I = Word containing the EBCDIC equivalent of the character to be packed in the right hand eight bits, and zeros in the left hand eight bits.

IA = COMET array

J = Character number in COMET array to be replaced.

The existing character in column J of the COMET array will be deleted and replaced by the required character.

This subprogram uses the COMET subprogram QSHUV.

Zone Recognition Subprogram

I = IQZØN (IA,J)

To interrogate the zone coding of a character.

IA = Previously defined COMET array

J = Column number within IA whose zone coding is to be interrogated

I = Variable to contain returned value. The table below shows the returned values of this subprogram, depending upon the zone coding encountered.

I is set to:	When the CHARACTER was		
1	A-I Ö	or &	(12 zone)
2	J-R Ō	or -	(11 zone)
3	S-Z	or /	(0 zone)
4	0-9	or blank	(no zone)
5	special		

Disk File Record Blocking Subroutines

Definitions

LOGICAL RECORD - the record handled by the user

PHYSICAL RECORD - the record stored on disk

BLOCKING FACTOR - number of Logical Records in a Physical Record

PHYSICAL RECORD LENGTH as defined in the DEFINE FILE

LOGICAL RECORD LENGTH * BLOCKING FACTOR - this should not exceed 320

File Buffer

All the following subroutines use the file buffer IBUF, which contains control words to regulate the reading and writing of records, and also is used for assembling the block of records for transfer to and from the disk. The following is the appearance of the file buffer.

IBUF(1) = File Number
IBUF(2) = Logical record length
IBUF(3) = Number of logical records in block
IBUF(4) = File type = 1 when block has only been used for GET operations (i.e. is identical to equivalent records on disk)
= 2 when block has been used for PUT operations
IBUF(5) = Number of first logical record in block. Set to zero if no record has been transferred to buffer.
IBUF(6) = Blocked records
etc.

IBUF must appear in the calling program with dimension

LOGICAL RECORD LENGTH * BLOCKING FACTOR + 5

Subroutine Calling Sequences

Call DØPN (IBUF,N,J,LL)

IBUF = Name of file buffer
N = File number
J = Logical Record Length
LL = Length of file buffer

Call DØPEN (IBUF,N,J,L)

To open a file buffer. Sets initial values to the file control words. No transfer of information to or from disk takes place.

Disk File Record Blocking Subroutines cont'd

Call DØPEN (IBUF,N,J,L) cont'd

IBUF = Name of file buffer
N = File number
J = Logical record length
L = Blocking factor

DØPN and DØPEN are alternative forms. Their effect is identical.

Call DGET (IBUF,K,IA)

To transfer logical record K of the file to the array IA. If the required record is already resident in the buffer, it is immediately transferred. If it is not, the block of records in the buffer is stored if necessary, and the correct block of records obtained from disk. Transfer then takes place.

IBUF = Name of file buffer
K = Required logical record
IA = Array to contain record obtained

Note that IA is integer. If real values are required, they can be obtained by use of a suitable EQUIVALENCE statement, in which real variables are assigned to EVEN locations in the array IA. The real variables then occupy the designated location, and the next lower location in the array. e.g.

```
DIMENSION IA(40)
EQUIVALENCE(B,IA,(16))
```

B occupies IA(16) and IA(15)

Note:
The use of an EQUIVALENCE statement in this way is not strictly speaking permitted. However, it works satisfactorily, providing nothing is done to force the addresses of real variables onto uneven word numbers. To prevent this, the following rules should be observed:

1) Equivalence only to even locations in the integer array IA.

2) The integer array IA should be dimensioned to an even number of words.

3) If IA is in COMMON, then any previous variables in COMMON together occupy an even number of words.

e.g.
```
COMMON IX,IY(2),IZ,IA(20)
EQUIVALENCE (IA(2),B)
```
is valid

```
COMMON IX, IA(20)
EQUIVALENCE (IA(2),B)
```
is not valid

Disk File Record Blocking Subroutines cont'd

Call DPUT (IBUF,K,IA)

To transfer the array IA to logical record K of the file. Operation is similar to DGET. The contents of IA are transferred only as far as the file buffer, and not written on disk.

Call DPTS (IBUF,K,IA)

To transfer the array IA to logical record K of the file. The operation is similar to DPUT, except that it is expected that the file will be written sequentially. Hence a block of records is not read from the disk before transferring data from IA to IBUF, in anticipation of later modifying all logical records in the block.

Call DFIND (IBUF,K)

To position the disk in the correct position to process logical record K of the file. No transfer of information takes place.

Call DCIØS (IBUF)

To close the file. If a block or records requires transfer to disk, the transfer is made.

Call DGET1 (IBUF,K,M,IX)

To transfer one word of a logical record K to the location IX.

IBUF	=	Name of file buffer
K	=	Required logical record
M	=	Word number within logical record K from which data is to be obtained.
IX	=	Variable to contain returned value.

Apart from the fact that only one word is transferred, the operation of this subroutine is identical to DGET.

Call DPUT1 (IBUF,K,M,IX)

To transfer one word from the location IX to the logical record K.

IBUF	=	Name of file buffer
K	=	Logical record to be modified
M	=	Word number within logival record K into which data is to be stored.
IX	=	Word to be stored

Disk File Record Blocking Subroutines cont'd

Call DPUT1 cont'd

Apart from the fact that only one word is transferred, the operation of the subroutine is identical to DPUT.

Call DUSE (IBUF,K,M,KEY,KL)

To generate a pointer to word M of logical record K. If the block containing the required logical record is already in the buffer, the pointer is generated immediately. If it is not, the block in the buffer is stored if necessary, and the required block obtained from the disk, after which the generation of the pointer takes place.

IBUF = Name of file buffer
K = Required logical record
M = Word required
KEY = Switch to indicate purpose for which logical record is required.
= -1 no record is being read or written. Write file buffer to disk if any record has been modified.
= 0 records are to be written sequentially.
= 1 record is to be read
= 2 record is to be written random access
KL = Pointer to position in IBUF of required word

Normally DUSE is not called by the user. It is called by DPUT, DGET, DPTS, DCLØS, DPUT1 and DGET1.

DPUT, DGET, DPTS and DCLØS are secondary entry points into the subprogram DUSE, which is written in assembler language. DØPEN and DØPN are separate assembler language subprograms DFIND, DPUT1 and DGET1 are written in Fortran.

Other Subprograms not Normally Called by Users

These are required in core to satisfy calls from the subprograms listed above.

PSFAC	Moves double word from ACC and EXT to FAC
QEDIT	(COMET subroutine) Edits alphabetic fields
QERØR	(COMET subroutine) Error routine for COMET. Required to satisfy call in QEDIT, but never used, as the conditions which cause an error in QEDIT are tested in PEDIT before QEDIT is called.
QGRAB	(COMET subroutine) Extracts single character from a COMET array. Cannot be called directly from FORTRAN.
QSHUV	(COMET subroutine) Stores a single character in a COMET array. Cannot be called directly from FORTRAN.
PNMPY	PNMPY is a secondary entry point to this subprogram. Multiple entry points have been provided to permit the inclusion of a trace feature, similar to *ARITHMETIC TRACE. It has, however, been found unnecessary.
PNADD	PADD, PSUB and PNSUB are secondary entry points to this subprogram. Multiple entry points have been provided for the same

Other Subprograms not Normally Called by Users (cont'd)

PMADD (cont'd)

reason as in PMMPY above

PMDVT To perform division of high-precision integers. The quotient and remainder are returned as a separate argument, for manipulation as required by other subprograms.

XRØTE To reverse an alphabetic array.

TYWR (IDEAL Subprogram) To perform typewriter input/output.

WRTY (IDEAL Subprogram) To perform typewriter output.

IBM library programs required:

CARDO

TYWRO

WRTYO

SPEED

COMMON

PHILADELPHIA, PENNSYLVANIA
SEPTEMBER 9-11, 1968

PROJECT:	Management Installation Division, Personnel Project
SUBJECT:	Computer Selection and Justification of Computer Applications
SPEAKER:	Sander de Haan, Supervisor, Equipment Evaluation & Control Section Data Processing Department Automotive Assembly Division Ford Motor Company
FOR PRESENTATION:	Tuesday, September 10, 1968, 8:30 AM, Session T1A

FORD MOTOR COMPANY
AUTOMOTIVE ASSEMBLY DIVISION

COMPUTER SELECTION AND JUSTIFICATION OF COMPUTER APPLICATIONS

This morning I would like to tell you a little about how the Automotive Assembly Division of Ford Motor Company goes about the business of selecting computers and related equipment to perform the numerous business applications in its assembly plants, and mention some of the problems related to multi-installations.

To give you an idea of the environment, let me start off with saying that the Automotive Assembly Division's assignment is to assemble cars and trucks for the Ford Motor Company. This means that all car lines from Lincoln and Thunderbird to the smallest Falcon as well as all commercial vehicles, from the largest diesel to the smallest Bronco are assembled in some 20 assembly plants in the United States and Canada. Each plant assembles vehicles for one or more car lines; some plants have one assembly line which mixes light trucks and passenger vehicles while other plants have separate assembly lines for passenger vehicles and commercial vehicles. Most plants operate two production shifts.

To support these plants, we have a standard computer equipment configuration which consists of a Honeywell 200 computer, 49K core memory, four tape drives with card punch, card reader and printer and a 9.6 million character disk. This equipment is used as a batch processor. We also have an IBM 1130 with 8K core memory; one 512K word disk, and a card reader. The applications on the batch processor are of a commercial nature, and include Productive Materials Inventory Reports, Accounts Receivable, Accounts Payable, Vehicle Invoicing, Labor Planning & Control and the like. The applications on the IBM 1130 include an Early Warning system used to alert shop supervision to peak workloads on the assembly line in sufficient time to take appropriate action. In advance of each work shift, orders are blended and a report is printed showing the sequence of orders which will provide the best possible mix on the line. At present we are testing an on-line time and attendance system in one of our plants with the idea that such a system could operate on the IBM 1130 simultaneously with the Early Warning application. Another application being tested involves the operation by computer of a storage system for commercial vehicles between the paint and trim lines in one of our West Coast plants. A system to control a high bay warehouse for storage of truck parts is also under development. The latter two systems may require replacement of the IBM 1130 by an IBM 1800 computer.

Selection of hardware and related equipment as you will readily recognize requires careful consideration. Cost-wise we are continually faced with the 20 plant multiplier. The impact of $1,000 rental per month per plant immediately propels itself into an annual cost figure of almost a quarter of a million dollars. It is small wonder therefore, that we take great

pains to establish in advance the dollar impact of any equipment change or acquisition.

The rapid technological advances in the computer field dictate that from time to time we review the total computer hardware situation. We approach this problem by maintaining contact with the major vendors and evaluating new equipment offered. When we find that new equipment might be of use to us, discussions with the vendor(s) are initiated. The major considerations are established. Generally these are:

a) Hardware availability

- Delivery schedules
- Factory shipping dates
- In-transit times
- Installation time

b) Hardware capabilities

- Processing time
- Reliability
- Throughput
- Peripherals

c) Facilities

- Suggested layout and space requirements
- Air conditioning and humidity requirements
- Cabling requirements
- Power panel and related requirements
- Safety considerations

d) Manpower

- Console operator requirements
- Programmers/systems analysts needed for initial development
- Programmers/systems analysts needed for continuing maintenance
- Systems engineers available to us for consultation on a continuing basis

e) Software

- Compilers
- Utility programs
- Operating Systems
- Languages
- Training of programmers

f) Conversion problems

- Software tools to convert programs
- Demonstration of conversion techniques
- Supplier provided assistance
- Testing facilities
- Parallel runs required
- Program documentation

g) Installation

- Freight costs
- Timetable for conversion of each installation
- Field engineering support during installation period

h) Field Engineering

- Number of qualified engineers available at each location
- Response time
- Preventive maintenance requirements
- Spare parts availability
- Regional and/or district support

i) Back Up

- Back up arrangements for each location

Our personnel will discuss each of these items with the vendors' representatives and indicate that their proposal should include specific answers to these questions. At this point the vendors' representatives will spend considerable time with our personnel to familiarize themselves with the applications involved. The number of computer programs involved is substantial; approximately 250 divisional H-200 programs and about 45 IBM 1130 programs (exclusive of sort programs). A basic knowledge of the systems and program operation is necessary if the vendor is to make an adequate proposal.

Next the vendor is asked to demonstrate his capability to convert our programs. For this purpose we select a few programs reflecting the complexity of our system, including the various languages employed in the programs. The demonstration programs therefore will not be average, but will generally include one of the more complex programs, a sort and other programs which would be either compute or peripheral bound. We have found it difficult to compute the expected total workload on the basis of the results of such a demonstration.

After successful completion of the demonstration the vendor may submit his proposal, which subsequently will be carefully analyzed by our own personnel. This generally results in many questions and discussions with the vendor's personnel to clarify any doubtful points.

Once we have reached this point we project the estimated running times for

each location and make a careful analysis of the changes in operating costs and estimates of the development, implementation and launching costs. The impact on scheduling of computer operations in the plants is another important item to be studied. Paired with this is an estimate of facilities costs which includes such items as office re-arrangements, air conditioning, power supply, freight costs and initial supplies to be purchased.

The same data are prepared for each vendor's proposal and comparisons against current operations are made to determine the incremental costs and/or savings for each proposal. Upon evaluation of the various proposals, a recommendation is made and approval of the Administrative Planning Office Manager, the Divisional Controller and the Divisional Manager is obtained, as required.

The proposal is summarized on a standard form named, "Computer Equipment Proposal", and detailed schedules are attached with the Division's reasons for the selection or rejection of each of the alternatives considered. Final approval of the Systems Office, Finance Staff, is then requested.

When a proposal affects all locations an installation schedule is attached. Because of the annual model change-over, we generally do not convert or install new equipment between May and September. In addition there are many practical reasons for not wishing to extend a major project beyond one model year. Consequently the installation is usually scheduled for a six month period between October and May, the ideal period being from November through April, which leaves May available for cleaning up any loose ends. It means that after the first plant is installed and operating to our satisfaction, the conversion teams proceed at the rate of one plant per week. To accomplish this it is evident that all details for all locations have to be worked out completely ahead of time. I assume the logistics problem involved would seem minor to the Defense Department, though I can assure you that a great deal of effort and planning is necessary to accomplish it.

Let us turn now to the matter of deciding whether or not a certain application should be computerized. The approach is less involved because hardware selection is not necessary and the vendors, therefore, are not involved. First a quick look at our internal organization. The Division General Office has control over equipment and programming. Programs are developed, compiled and tested in the General Office and the plants are furnished compiled programs and program listings. Two departments in the General Office share the responsibility for data processing systems. The Methods & Systems Department negotiates with "customer" departments and develops the specifications for the application to be computerized. The Data Processing Department has responsibility for the equipment and the programming. The latter consists of writing, compiling and testing the programs; providing documentation for the plants, and in case of significant changes, the programmers will visit the plants and assist in installing the new programs.

So, when a new application is considered, Methods & Systems will develop preliminary specifications and based on these obtain programming estimates. From the "customer" they will obtain an estimate of savings/cost expected from the new application. A time table is developed showing starting and completion dates for the following areas:

- Fact finding
- Systems development
- Programming
- Equipment procurement
- Test installation
- System evaluation
- Division-wide installation

Here again schedules are developed for recurring operating costs and non-recurring development and launch costs. The objective or purpose of the application is described and the specific benefits expected are indicated. All of this information is summarized on a form entitled "Systems Projects". Actually the originating "customer" department will prepare the top half of the "Systems Project" form which describes the application and the benefits expected. The Methods & Systems Department will complete the form and obtain the approval of its manager, or, depending on costs involved, higher level management, before proceeding with the project.

Subsequent to implementation of a new or modified application the locations may be required to prepare another form named "Cost Evaluation of System Applications" for the purpose of comparing and evaluating results of the new or modified application to the cost objectives originally set down.

A few other items that may interest you:

- Under the centralized programming concept every effort is made to avoid dual programming on a continuing basis. While a conversion is in progress we have little choice but to contend with dual program maintenance. This is one of the major reasons for keeping conversions within a six month period.

- We have been concerned about not being able to compare data processing performance between plants. This applies mostly to the batch processing portion. Extensive computer usage reports are being prepared by each location. However, the need for standards by program has become very evident. Considerable effort is currently expended by our staff to develop such standards. These will then enable us to rank the plants on their performance in each of 16 major applications. It is hoped that such reports will have a good effect on individual plant performance and permit the Division General Office to pinpoint areas of concern.

Copies of this presentation with sample forms attached are available for you on your way out. I shall be glad to answer any questions you care to ask.

COMPUTER EQUIPMENT PROPOSAL
(In U.S. Dollars)

DIVISION, STAFF, OR SUBSIDIARY	USING ACTIVITY	DATE

EQUIPMENT REQUIRED AND RELEASED

DESCRIPTION MANUFACTURER	MODEL NAME & NO.	DATE	QTY.	BASE RENTAL PER MONTH	PURCHASE PRICE	USAGE HRS/MO.
TOTAL						

PURPOSE AND BENEFITS (SUMMARY)

FINANCIAL EVALUATION ($000'S)

NONRECURRING COST		NET RECURRING COST (INCREASE)/DECREASE	ACCOUNTED	INCREMENTAL
CAPITALIZED ITEMS:		PERSONNEL:		
EQUIPMENT (ABOVE - NET)		DIRECT NO.____		
OTHER		AVOIDANCE NO.____		
TOTAL CAPITALIZED ITEMS				
		EQUIPMENT RENTAL:		
		DIRECT		
EXPENSE ITEMS:		AVOIDANCE		
INSTALLATION				
LAUNCHING		OTHER:		
OTHER		DIRECT		
TOTAL EXPENSE ITEMS		AVOIDANCE		
TOTAL NONRECURRING COST		TOTAL NET RECURR. COSTS		

INDEX TO SUPPORTING MATERIAL

REFERENCE	MATERIAL	REFERENCE	MATERIAL
_____	ALTERNATIVES CONSIDERED	_____	DETAILED EQUIPMENT SPECIFICATIONS AND COSTS
_____	BENEFITS	_____	EXPENSE ITEMS (NONRECURRING COSTS)
_____	CAPITALIZED ITEM (NONRECURRING COSTS)	_____	PERSONNEL REQUIREMENTS (NET RECURRING COSTS)
_____	CURRENT AND FORECAST EQUIPMENT UTILIZATION	_____	PROJECT APPROPRIATION REQUEST (IF APPLICABLE)
		_____	OTHER

PROPOSED BY	CONCUR	CONCUR
Manager of Proposing Activity Date	Controller of Proposing Activity Date	Finance Staff Date

SYSTEMS PROJECTS

M & S CONTROL NO _____

REQUESTING DEPARTMENT	OTHER DEPTS. OR AFFECTED	ACTION RPTD	DATE
		ISSUED	
REQUESTING ASSY PLANT	PLANT ACTIVITY AFFECTED	REVISED	
PROJECT TITLE	PROJECT NO	DISCONTINUED	
APPLICATIONS AFFECTED	REVISION	COMPLETE	

REQUESTING ACTIVITY

PURPOSE OR OBJECTIVE _____

SPECIFIC BENEFITS EXPECTED _____

ANNUAL RECURRING (COST)/SAVINGS

*GROSS SAVINGS EXCLUDING NEW SYSTEM COSTS

TANGIBLE		INTANGIBLE		COST AVOIDANCE	
PERSONNEL	$	PERSONNEL	$	PERSONNEL	$
EQUIPMENT	$	EQUIPMENT	$	EQUIPMENT	$
SUPPLIES	$	SUPPLIES	$	SUPPLIES	$
OTHER	$	OTHER	$	OTHER	$
TOTAL	$	TOTAL	$	TOTAL	$

METHODS & SYSTEMS

STUDY PLAN:	EST'D. START	EST'D. COMPLETION	TOTAL HOURS	SYSTEMS DEVELOPMENT HOURS/MONTH		PROGRAMMING HOURS/MONTH	
FACT FINDING				1	2	1	2
SYSTEMS DEVELOPMENT				3	4	3	4
PROGRAMMING				5	6	5	6
EQUIPMENT PROCUREMENT				7	8	7	8
TEST INSTALLATION				9	10	9	10
SYSTEM EVALUATION				11	12	11	12
DIVISION WIDE INSTALLATION				13	14	13	14
OTHER				BALANCE		BALANCE	

NEW SYSTEM COSTS:

NON RECURRING LAUNCH COSTS*	
PERSONNEL	$
SERVICES	$
EQUIPMENT	$
TRAVEL	$
OTHER	$
TOTAL	$

*ANNUAL RECURRING COSTS OF NEW SYSTEM	
PERSONNEL	$
SERVICES	$
EQUIPMENT	$
TRAVEL	$
OTHER	$
TOTAL	$

*PROVIDE DETAILED SUPPORT ON SEPARATE SCHEDULE INCLUDING BUDGET ACCOUNTS AFFECTED

M & S DEC 67 AAD 7209 PRINTED IN U.S.A.

_____
REQUESTING ACTIVITY APPROVAL

COST EVALUATION OF SYSTEM APPLICATIONS

DATE PREPARED _____

REPORTING ACTIVITY	PERIOD REPORTED
TITLE OF APPLICATION	PROJECT OR CODE NO.
APPLICATION MODIFIED OR DISPLACED	CODE NO

	CURRENT OR PRIOR SYSTEM		NEW OR MODIFIED SYSTEM		NEW SYSTEM (OVER) UNDER PRIOR SYSTEM	
	EQUIVALENT MANPOWER	COST	EQUIVALENT MANPOWER	COST	EQUIVALENT MANPOWER	COST
PERSONNEL						
METHODS AND SYSTEMS		$		$		$
PROGRAMMING						
OPERATIONS: CONSOLE OPERATORS						
EAM OPERATORS						
KEYPUNCH						
OTHER SALARY						
DIRECT HOURLY						
INDIRECT HOURLY						
SUB TOTAL PERSONNEL		$		$		$
EQUIPMENT COMPUTER: _____ SERIES						
_____ SERIES						
EAM SORTER						
COLLATOR						
INTERPERTERS						
REPRODUCERS						
TABULATORS						
TRANSCEIVERS						
OTHER						
GENERAL INVENTORIES						
OTHER						
SUPPLIES CARD STOCK						
FORMS/PAPER						
OTHER						
OTHER TRAVEL EXPENSE						
FREIGHT						
ASSESSMENTS (SPECIFY)						
OTHER						
SUB TOTAL EQUIPMENT, SUPPLIES, OTHER		$		$		$
TOTAL		$		$		$

REMARKS _____

F N
OCT 66 AAD 7210 PRINTED IN U.S.A.

_____
CONTROLLER OF

BUDGETING AND DISTRIBUTING COMPUTER OPERATION COSTS

For Presentation to COMMON
at
Philadelphia, Pennsylvania
September 10, 1968

ANNOTATION

A method of accounting for cost of feasibility studies, conversion, and data processing center operations serving multiple departments.

C. M. Atkinson
Duquesne Light Company
Pittsburgh, Pennsylvania

BUDGETING AND DISTRIBUTING COMPUTER OPERATION COSTS

C. M. Atkinson

Duquesne Light Company
Pittsburgh, Pennsylvania

The purchase or lease of a computer and other EDP equipment constitutes a major expenditure in most companies. Feasibility studies, computer programming, and related activities also represent a substantial incremental cost. Before purchasing EDP equipment, competent executives have learned to study all costs related to computer acquisition and operation, and relate the total cost to expected results to be achieved. In addition, the utility executive considers a data processing installation a capital acquisition from which a measurable return must be obtained.

The fact that many companies have not, in the past, attempted to relate subsequent actual costs with the assumed gains has been indeed fortunate. Most such attempted comparisons would not have been meaningful. Costs, such as system design, personnel training, programming of existing operations, and conversions were probably charged directly to functional expense accounts. Such costs could not be retrieved and accurately accumulated. Cost comparisons would have been further distorted by the concurrent programming and testing of any new data processing projects.

Today, most companies are not satisfied with merely merging data processing costs in diverse functional accounts. Many companies are searching for a simple and equitable method of assigning data processing costs to the departments served. In instances where a large centrally-located computer serves a number of related utilities in a utility system, costs must be allocated to the specific utilities served.

I would like to review briefly a system of accounting for data processing costs which Duquesne Light Company uses to classify EDP operating and development costs. This data is used for analysis and budget comparison and provides essential data for management control. The method is simple in design yet provides all the essential features needed to collect and catalog costs of data processing operations under reasonable management requirements.

In developing the data processing cost system for use in our company, the following basic criteria were specified:

1. To allow the manager to segregate the costs of programming, testing, and conversion of <u>existing</u> accounting functions or engineering studies.

2. To permit the manager to segregate the costs of designing, programming, and testing <u>new programs</u> for accounting, engineering, or other purposes.

3. To help the manager compare monthly and annual EDP operating costs of performing existing accounting functions with costs of previously performing these operations.

4. To aid the manager in comparing monthly and annual EDP operating costs of performing continuing engineering studies with the costs of previously performing the studies, either within the company or by an outside data processing service center.

5. To assist the manager to assign data processing operation costs to functional departments served, when responsibility budgeting and accounting is used.

Within the Duquesne Light Company, the cost of performing existing general and customer accounting functions and engineering as well as on-going management studies during the conversion to the large scale computer were charged to the appropriate customer accounting, operating or general and administration expense account. The data processing department retained the budget cost responsibility. Alternatively, in companies without direct responsibility accounting, these costs could be charged back to the functional department as a budget responsibility of the department served.

Costs relating to the design, programming, and testing of new or revised work programs and any necessary conversion costs are accumulated as a deferred cost. The costs are subsequently amortized over a reasonable period from the completion of individual phases of the EDP project. This amortization period should approximate the pay-out period of economic justification of the project.

To meet the requirements of management as to the segregation of current operating costs and identification of deferred EDP costs for subsequent amortization, we have simply superimposed a memorandum work order procedure on the direct accounting charge distribution.

A Data Processing Order is used by the Data Processing Department to authorize the performance of specific work and to accumulate costs of each individual data processing function or application. (Exhibit A)

We use two types of Data Processing Orders to identify the operational nature of EDP work:

1. Continuing Data Processing Orders are used to identify and accumulate the costs of general and customer accounting or other data processing operations or functions of a continuing nature.

2. Specific Data Processing Orders are used when it is desirable to find the cost of system development work, a particular data processing job, or engineering study project which is not of a continuing nature.

These Data Processing Orders are closed when the project is completed.

The identifying number for each Continuing and Specific type of Data Processing Order is issued and controlled by the Data Processing Department. Four copies of the Order are prepared from information already available or from information supplied by the department requesting a work assignment. The four copies of the Order are sent to the department requesting the work for signature. Copies are then distributed as follows:

3 Copies - Returned to the Data Processing Department. One copy sent to the department requesting the work when the assignment is completed.

1 Copy - Retained by the department requesting the work.

A four-digit DP Order code is used in assigning order numbers in order to facilitate the identification and accumulation of data processing costs by function, department, and data processing application. The attached Exhibit B contains a tabulation of some assigned data processing application codes currently used by our company.

The allocation of computer operation labor and other expenses to functional accounts and to individual data processing orders must be made in a reasonably precise manner if meaningful cost data is to be accumulated. Individual time sheets are required of all employees in the department. Computer and peripheral hardware logged operating time and tabulating card counts form the basis for allocation of such costs to functional accounts and individual DP Orders. Computer and other hardware rentals are allocated on the basis of machine operation logs. Supplies such as tabulating cards and paper are charged on an estimate of usage basis from memorandum records maintained by the department.

Each month the Data Processing Department prepares a detailed report of expenditures to each Data Processing Order showing the responsibility budget code, each charge by type to the Order, as well as the expense account or other functional account which was charged. A sample page of this report is attached as Exhibit C.

The above description is, of course, only one of the accounting methods that can be used to segregate EDP costs for budget and accounting purposes. The exploitation of the tremendous capacity of the new generation computers, however, can only be realized if management is provided with the tools of objective data processing budgeting and accounting. These tools are necessary to adequately measure the progress made in attaining EDP objectives and determining if the tangible and intangible benefits of the computer are worth the actual incurred costs.

DATA PROCESSING ORDER

EXHIBIT A

FORM DSS-2910

DUQUESNE LIGHT COMPANY

Order No. 1402

Nature of Work Requested: Materials and Supplies

Program File No.

Department Requesting Work: Data Processing Department

Continuing D.P. Order [X]

Account To Be Charged: Account 163.01

Specific D.P. Order [ ]

Date Work Completion Required: Continuing

Date Completed: N/A

Description of Work:

Charge to this DP Order the time, material, rent and other expenses incurred by the Operations Section in processing the complete materials and supplies distribution application. Include such items as preparing distribution check lists, material distribution reports, stock ledgers and pricing out-files for Duquesne Light Company and subsidiary companies.

Submitted By	Approved By	Work Authorized By
Manager, General Accounting Section		Manager
Title	Title	Title
1-5-67		December 16, 1966
Date	Date	Date

EXHIBIT B

DATA PROCESSING ORDER NUMBER CODES

DATA PROCESSING NUMBER			
Function	Dept. or Division	Application	
1			Operations Function
	1		Operations Division
		01	G-15 operations
		02	Coal billing summaries
		03	Radiation exposure reports
2			Development Function
	1		Operations Division
		01	Service interruption program
		02	EEI Load Diversity Report
1			Operations Function
	2		Sales Division
		01	Billing data - cards and tape
		13	Appliance saturation study
2			Development Function
	2		Sales Division
		01	Program conversion - billing data
1			Operations Function
	3		Engineering & Construction Division
		01	Tower Design - Phillips - North
2			Development Function
	3		Engineering & Construction Division
		01	Transmission tower design program
1			Operations Function
	4		Fiscal Division
		01	Payroll and labor distribution
		02	Materials and supplies
		03	Continuing property records
		04	Accounts payable
2			Development Function
	4		Fiscal Division
		01	Payroll and labor distribution
		02	Materials and supplies
		03	Continuing property records
		04	Accounts payable

EXHIBIT B
(Continued)

DATA PROCESSING NUMBER			
Function	Dept. or Division	Application	
1			Operations Function
	5		System Planning Department
		01	G-15 operations
		02	Load flow studies
2			Development Function
	5		System Planning Department
		01	Financial analysis program
1			Operations Function
	6		Data Processing Department
		01	Supervision - Manager and Directors
2			Development Function
	6		Data Processing Department
		01	Supervision - Manager and Directors
		02	General system development costs
		03	Development Section Office Rent
1			Operations Function
	7		Personnel Department
		01	Personnel data files
2			Development Function
	7		Personnel Department
		01	
1			Operations Function
	9		Miscellaneous
		01	Label printing - Secretary Department
		02	Forms control work
2			Development Function
	9		Miscellaneous
		01	

EXHIBIT C

OPERATING EXPENSE AND CLEARING ACCOUNTS DETAIL LEDGERS

FORM 310-5...

COMPANY DUQUESNE LIGHT COMPANY DATE JUNE 1968

CO	MO	REF	CLASS OF WORK	BUDGET CODE	ORDER NUMBER	ACCOUNT NUMBER	STATION NUMBER	TYPE OF CHARGE	DESCRIPTION OF ACCOUNT	CURRENT AMOUNT	CALENDAR YEAR TO DATE
1		5098	1	42	2407	186 8300		50	CONSTRUCTION COST LEDGER	239	
									BUDGET AND ACCOUNT TOTAL	239	239
									TYPE CHARGE TOTAL	239	239
1		5098	1	42	2407	186 8300		60	CONSTRUCTION COST LEDGER	592	
									BUDGET AND ACCOUNT TOTAL	592	592
									TYPE CHARGE TOTAL	592	592
1		5098	1	42	2407	186 8300		70	CONSTRUCTION COST LEDGER	306	
									BUDGET AND ACCOUNT TOTAL	306	306
									TYPE CHARGE TOTAL	306	306
1		5098	1	43	2407	186 8300		10	CONSTRUCTION COST LEDGER	3520	
									BUDGET AND ACCOUNT TOTAL	3520	685807
									TYPE CHARGE TOTAL	3520	685807
1			1	43	2407	186 8300		22	CONSTRUCTION COST LEDGER		
									BUDGET AND ACCOUNT TOTAL		1008
									TYPE CHARGE TOTAL		1008
1		2110	1	43	2407	186 8300		43	CONSTRUCTION COST LEDGER	57982	
1		2110	1	43	2407	186 8300		43	CONSTRUCTION COST LEDGER	3335	
									BUDGET AND ACCOUNT TOTAL	61317	398359
									TYPE CHARGE TOTAL	61317	398359
1			1	43	2407	186 8300		45	CONSTRUCTION COST LEDGER		
									BUDGET AND ACCOUNT TOTAL		160524
									TYPE CHARGE TOTAL		160524
1		5098	1	43	2407	186 8300		50	CONSTRUCTION COST LEDGER	95	
									BUDGET AND ACCOUNT TOTAL	95	2855

COMPUTER COSTS CHARGED AS OVERHEAD COST

R. H. Magee

THE MAGNAVOX COMPANY

To better understand the situation as it exists in the Magnavox Company, I think a few background and historical comments are in order. The Magnavox Company went into the computer game a little over twelve years ago. At that time, an IBM 650 computer was shared by the Data Processing (Tab Department) and Engineering. After about a year the computer was moved to the Tennessee operations for accounting work. About a year later, Engineering received its own computer and since that time there have been separate computer installations for Data Processing and Engineering. It is the Engineering installation that will be discussed in this presentation.

When the Engineering computer was installed, the questions of cost charging arose. After considerable discussions on the topic, it was decided that it should be an overhead expense. Although this decision was reached over ten years ago, I believe the same arguments are still valid for this type of operation. For this reason, I think we should review some of these arguments in some detail.

First, since the installation (including computer, personnel, supplies, etc.) is doing 100% engineering and scientific work, it is very difficult to apply costs savings. For example: A job requiring one hour of programming and two minutes of computer time may be as important to the company as one requiring ten hours of programming and ten minutes of computer time. Should both of these jobs be charged the same? Or should costs be based on programming and computer time? If this can be resolved,

all is well and good; but we could not resolve it so simply.

As you may have guessed, many of our jobs are one shot, or at least of short duration; and many of these are what might be considered small jobs requiring an hour or so of programming time and a few minutes of computer time. Now the question of economy arises.....does it cost more to spread the charges between the charge numbers than it does to absorb them in overhead? We chose to absorb them as overhead.

Another problem or question arose.....due to our accounting structure, there may be 100 different charge numbers open at any one time and these may vary from week to week. So if costs were charged, it would almost have to be on a weekly basis and this involves considerable more effort than a monthly charging.

We also encountered the problem of budgeting. Our manpower budgeting is done for a six-month period and is done in man-months to the closest tenth of a man-month. With our type of jobs it becomes almost impossible to do any budgeting within these boundaries. The people in charge of projects could not come close to estimating the programming and computer times required. This, by the way, was tried for one six-month period and it was disastrous. This signaled the demise of trying to do any budgeting of programming time.

These factors all contributed to cause of an overhead shop. A number of years ago we were accepted as an overhead function by the cogizant government accounting agency and this pretty well stifiled all more recent attempts to do any direct charging.

I should point out that there is some direct charging within the

computer facility. On any requested overtime, the labor charges are charged to the requesting project; no machine time is charged, however. I might add that this is not a great amount and only amounts to a few dollars per month. We use it mainly to make sure the work has be to done during the overtime period.

These, then, are some of the reasons that we do charge most of our charges to overhead. They have worked well for us for the past twelve years and certainly have made my job a lot easier. I hope it continues this way, but there are questions being raised on the feasibility of it at the present time. I do not foresee going to a complete turnabout, but that maybe 25 or 30% of the charges will be made directly. The arguments given within this presentation are still valid and will continue to cover most of our work.

R. H. Magee
6 September 1968

SESSION T1

SEPTEMBER 10, 1968

A SELECTIVE DISSEMINATION OF INFORMATION SYSTEM FOR MEDICAL LITERATURE

JAMES L. GRISELL, Ph. D.

ROGER GUDOBBA

APPLICATIONS/ BIOMEDICAL PROJECT

14 PAGES

COMMON MEETING

PHILADELPHIA,PA.

A SELECTIVE DISSEMINATION OF INFORMATION SYSTEM FOR MEDICAL LITERATURE

by

James L. Grisell, Ph.D. and Roger Gudobba

The Lafayette Clinic Medical SDI system has been designed to keep scientific investigators routinely informed of the world's literature in any selected area of medicine. The system can provide either a current awareness of new articles or a bibliographic search of an entire file of articles for references relating to any topic. At the Lafayette Clinic we use the system for the literature on the mental illness of schizophrenia.

The current awareness portion of the system is designed to work on an automatic basis. Each investigator serviced by the system has on file in the Computing Laboratory a list of key terms, designated as a profile, which define that segment of the schizophrenia literature of interest to him. The key terms used in profiles are obtained from a dictionary of all terms occurring in the entire file. Once a month, all profiles are compared against all new articles entered into the system during the preceding four weeks. The investigator receives a complete list of all articles which matched his profile.

If an investigator wants a bibliography, he prepares a profile which defines his area of research interest. When the key terms in the profile match the key terms in an article, this article will be included in the bibliography.

This system has been designed to provide a maximum of flexibility in writing profiles so that each investigator can define his area of interest with optimum

precision. Continuing feedback is also provided regarding profile key terms which are matched with article key terms. The ultimate goal of the system is to bring to the attention of each investigator only those articles which are of interest to him.

Description of Entries

The ultimate success or failure of any current awareness or bibliographic search system is the adequacy of the bibliographic reference data which is used. This data must meet two important criteria: (1) it must cover the world's literature of the area of interest as thoroughly as possible and (2) it must contain an adequate, but concise, description of the contents of each entry.

A source of bibliographic information which adequately meets both of these criteria is the MEDLARS system of the National Library of Medicine. The National Library is currently indexing approximately 25,000 journals containing 250,000 medical articles. As each journal is received at the National Library it is checked for articles of medical interest. For medical journals, all articles are routinely indexed. For a journal such as Science, only those articles pertaining to medical topics are included.

Each article is read by an indexer. The indexer then assigns a series of tags, or index words, which define the subject content of the article. These tags are words which appear in Medical Subject Headings (MESH). The tags or index words are also the subject headings under which articles are listed in the Index Medicus. To keep the size of Index Medicus within reason, indexing is done on two levels. The first level consists of tags which are most representative of the article's contents. These are used for cross indexing in

Index Medicus and are preceded by an * in a MEDLARS listing. The other tags are not used for Index Medicus but are available for searches of the MEDLARS' master file. The entries in the MEDLARS system are available from the National Library on the basis of either a demand search, or a recurring search. A demand search requires a list of tags and the dates within which the search is to be performed. A recurring search is routinely made on a monthly basis.

The Format of an Article

Each article in the MEDLARS system contains certain basic information descriptive of the articles. The following items are included:

(1) The author's name. If the article has multiple authorship all are included in the listing, with the senior author listed first. If there is no author (1.5%) it is listed as anonymous.

(2) The title. If the paper is in English the title will be listed exactly as it appears in the article. If the paper is in a foreign language, the title will be in English translation. The language in which the article was written is indicated by a standard abbreviation.

(3) The source of publication. The journals are given in standard abbreviated form. The volume number, month and year of publication and pagination are also given.

(4) Index terms or tags. The index terms from MESH which describe the subject contents of the article are also included.

Preparation of MEDLARS entries

All articles received to date, and they now number 5025, have been prepared for entry into our SDI system. One of the guiding principles of our system is that it be as automatic as possible. Consequently, we do a minimum of pre-editing of article listings. Each entry is assigned a document number. The order in which the entry parts are listed is consistent. Each line of an entry is given a sequence number and is labeled (A) author, (T) title, (S) source, or (K) index terms (designated key terms). They are now ready for keypunching and verifying. (See Figure 1)

Master Tape File

The articles are loaded on a magnetic tape in the following manner. First the article is stored in card image format. (There is provision for a maximum of 20 cards/article). Secondly, the individual terms from the title, author, source and key term cards are pulled out and stored in an array. (There is provision for a maximum of 50 terms/article.) All duplications and trivial words (such as and, or, the, etc.) are eliminated. In addition, the number of characters for each term is computed. As you will see later this technique of breaking down the article and counting the characters for each term will save a great deal of computer time later on. This tape can now be used for the current awareness run and then added to the master file. By using MAGOP we can store 5000 articles on one reel of tape. (See Figure 2)

Preparation of the Dictionaries

One of the most important aspects of our SDI system is the dictionaries. Three separate dictionaries are maintained: (1) authors (2) sources (3) key terms (from the title and key term cards).

Dictionary of Authors - Each author is included. 4266 authors in first 3800 articles.

Dictionary of Sources - Each source in its abbreviated form is included. 520 sources in first 3800 articles.

Dictionary of Key Terms - Since the goal of the SDI system was to make it as automatic as possible, only single terms are extracted from article titles. To get multiple word terms would require the pre-editing of the titles and designating which consecutive word groups should be treated as a single term. This is done in some SDI systems. For example, one may wish to designate

"social factors" as a two-word term. However, this will appear in the dictionary as two terms; "social" and "factors". While this may be regarded as a shortcoming of the system, provision can be made in the construction of profiles to treat these two words as a unit.

When dealing with the key term cards, multiple word terms are treated as one. The purpose of this was to retain all of the terms as they appear in MESH. 6067 key terms in first 3800 articles.

The cumulative total is kept for each term in the dictionary. Also, any terms appearing for the first time are flagged. It is now a simple task to give the investigators a listing of the new dictionary entries for the month. If any terms are of interest to the investigator he can add them to his profile before the current awareness run for that month.

Since the articles on tape already have the individual terms extrapolated the job of generating the dictionaries is simplified. (See Figure 3)

Profiles

The ultimate success or failure of an SDI system is a function of the ease and accuracy with which an investigator can define his area of interest on the basis of the terms in the articles being searched. This is done by constructing a profile of terms to be compared with the terms in an article. If the profile is a good one it will maximize the number of articles of interest it finds and minimize the number of articles designated as interesting, but which are not. If the profile is not a good one, the reverse will then be true.

The profiles used in this system are similar to the profiles used in the IBM SDI system, but with modifications. In the Lafayette Clinic Medical SDI system a profile has a hit level, which may range from -9 to +99. Each term

in the profile is compared with each term in the article and every time a match is found the weight of the term in the profile is summed. After the last comparison is made, the sum is compared to the hit level of the profile. If the sum is equal to or greater than the hit level, then the article is designated as being of interest. If the sum is less the article is ignored.

In addition, two types of terms may be used: complete terms and root terms. A complete term must appear in the article exactly as it is in the profile to result in an equal compare. A root term, on the other hand, will result in comparing only as many letters in the article word as are in the root term in the profile. For example if "child" is designated as a root term in the profile then it will result in an equal compare with children, childhood and, of course, child. The use of root terms is a convenient way of encompassing all variants of a term which may have one of several different endings.

One additional refinement in a profile is the use of modifiers: must and not. A must modifier simply indicates that any time a must term is found the investigator will get that article regardless of the hit level. A not term will do the opposite. When a must term and a not term are both found in the same article, the must term overrides the not term. Either a root or a complete term can have either of the modifiers. (See Figure 4)

Searches

Since this is the most frequently used program in the system, a number of techniques have been employed to decrease the time required to search the file. First, the program handles one profile at a time and makes a pass through the entire Master File of Articles. This enables us to keep the output for each profile separate without having to do a sort. Secondly, keeping the profile in core has the following advantage: when the profile is originally read in

the program calculates the number of characters in each term. Since the terms in the article also include character counts this is used as follows: For complete terms the character counts are compared and if they are not equal the alpha compare is not necessary. For root terms the character counts are also compared and the alpha compare is done only if the term in the article is equal to or longer than the term in the profile. Finally, we have written our own alpha compare routine. Since the articles are stored with A2 format to conserve tape and core requirements the use of an alpha compare routine such as NCOMP (1130 Commercial Subroutine Package) would necessitate unpacking the article terms first. Our alpha compare routine will handle A1 or A2 format. (See Figure 5)

If the sum of the weights of the compare terms equals or exceeds the hit level, or if a MUST term was found, the entire article is listed followed by the profile term matches for that article. For root term matches the complete term from the article is listed. Inclusion of the matching terms helps the investigator make value judgments regarding the interest level of the material a term brings to him and to modify his profile accordingly. (See Figure 6)

Reprint library

If the system is to be effective in disseminating the world's literature on schizophrenia to a group of investigators it is imperative that a complete reprint library be maintained. With approximately 40% of the articles in a foreign language, appearing in some 25,000 journals, it would be of little use if the investigator has to obtain the reprint by himself.

Consequently, we are also in the process of establishing a complete reprint file. This has been accomplished to date by (1) sending reprint request cards directly to the author, (2) obtaining Xerox copies from Wayne State University Medical

Library, and (3) obtaining Xerox copies from the National Library of Medicine for any articles that cannot be obtained by Steps 1 or 2. This has turned out to be both fruitful and worthwhile. Of the first 3700 articles entered into the system we have copies of all but two.

Conclusion

Although this system was designed for a specific medical research area, namely schizophrenia, it could easily be adapted to any area of any discipline. Except for the alpha compare routine, which is coded in Assembler, all of the programs have been coded in FORTRAN.

FIGURE 1

Article Entries from MEDLARS after pre-editing

```
1  A  1  AALL L
1  T  2  (SYMPOSIUM ON SCHIZOPHRENIA-LIKE PSYCHOSES AND ETIOLOGY OF
1  T  3  SCHIZOPHRENIA.  EXPERIENCES REGARDING THE TOPIC IN TANGANYIKA)  (GER)
1  S  4  SCHWEIZ ARCH NEUROL PSYCHIAT 93,377-9, 1964
1  K  5  *SCHIZOPHRENIA, TANGANYIKA (1)

2  A  1  AARONSON BS
2  T  2  AGING, PERSONALITY CHANGE, AND PSYCHIATRIC DIAGNOSIS.
2  S  3  J GERONT 19,144-8, APR 64
2  K  4  ADOLESCENCE, ADOLESCENT PSYCHOLOGY, *AGING, DIAGNOSIS, *MENTAL
2  K  5  DISORDERS, *MMPI, *PERSONALITY, SCHIZOPHRENIC PSYCHOLOGY,
2  K  6  SOCIOPATHIC PERSONALITY

3  A  1  ABELY P, LAUZIER B
3  T  2  (THE FATE OF THE CONCEPTS OF PERIODICITY, ATYPISM AND INCURABILITY
3  T  3  IN PRACTICAL PSYCHIATRY)  (FR)
3  S  4  ANN MEDICOPSYCHOL (PARIS) 122,729-46, MAY 64
3  K  5  CLASSIFICATION, *MENTAL DISORDERS, *NOMENCLATURE, PERIODICITY,
3  K  6  PROGNOSIS, *PSYCHIATRY, PSYCHOTHERAPY, SCHIZOPHRENIC PSYCHOLOGY

4  A  1  ABRAHAM G
4  T  2  (THE PROBLEM OF MIXED PSYCHOSES)  (FR)
4  S  3  ANN MEDICOPSYCHOL (PARIS) 122,481-90, NOV 64
4  K  4  CLASSIFICATION, *DEPRESSION, *EPILEPSY, *NEUROSES, *PSYCHOSES,
4  K  5  *PSYCHOSES, MANIC-DEPRESSIVE, *SCHIZOPHRENIA

5  A  1  ABRAMS S
5  T  2  A VALIDATION OF PIOTROWSKI'S ALPHA FORMULA WITH SCHIZOPHRENICS
5  T  3  VARYING IN DURATION OF ILLNESS.
5  S  4  AMER J PSYCHIAT 121,45-7, JUL 64
5  K  5  DIAGNOSIS, DIFFERENTIAL, *RORSCHACH TEST, *SCHIZOPHRENIA

6  A  1  ABRAMSON HA
6  T  2  ANTISEROTONIN ACTION OF LSD-25 AND OTHER LYSERGIC ACID DERIVATIVES,
6  T  3  FACT AND FICTION.
6  S  4  J ASTHMA RES 1,207-11, MAR 64
6  K  5  *ALLERGY, ASTHMA, AUTISM, CHILD, *HALLUCINOGENS, *LYSERGIC
6  K  6  ACID DIETHYLAMIDE, METHYSERGIDE (3), MIGRAINE, PHARMACOLOGY,
6  K  7  SCHIZOPHRENIA, CHILDHOOD, *SEROTONIN INHIBITORS, TOXICOLOGIC
6  K  8  REPORT (4)

7  A  1  ACHILLES M
7  T  2  (ATTEMPT AT A STATISTICAL DIAGNOSIS OF THE DRIVE STRUCTURE IN
7  T  3  PROBLEM STUDENTS)  (GER)
7  S  4  PRAX KINDERPSYCHOL 13,177-81, JUL 64
7  K  5  ADOLESCENCE, AGGRESSION, AUTISM, CHILD, *CHILD BEHAVIOR
7  K  6  DISORDERS, EDUCATION OF MENTALLY DEFECTIVE, MOTIVATION,
7  K  7  PERSONALITY, PUBERTY, SEX, STATISTICS, THEMATIC APPERCEPTION
7  K  8  TEST
```

FIGURE 2

Article entries as they are used in the SDI System

```
1  A  AALL L
   T  (SYMPOSIUM ON SCHIZOPHRENIA-LIKE PSYCHOSES AND ETIOLOGY OF
   T  SCHIZOPHRENIA.  EXPERIENCES REGARDING THE TOPIC IN TANGANYIKA)  (GER)
   S  SCHWEIZ ARCH NEUROL PSYCHIAT 93,377-9, 1964
   K  *SCHIZOPHRENIA, TANGANYIKA (1)
         28    SCHWEIZ ARCH NEUROL PSYCHIAT
          6    AALL L
          9    SYMPOSIUM
          9    PSYCHOSES
          8    ETIOLOGY
         11    EXPERIENCES
          9    REGARDING
          5    TOPIC
         10    TANGANYIKA
          3    GER

2  A  AARONSON BS
   T  AGING, PERSONALITY CHANGE, AND PSYCHIATRIC DIAGNOSIS.
   S  J GERONT 19,144-8, APR 64
   K  ADOLESCENCE, ADOLESCENT PSYCHOLOGY, *AGING, DIAGNOSIS, *MENTAL
   K  DISORDERS, *MMPI, *PERSONALITY, SCHIZOPHRENIC PSYCHOLOGY,
   K  SOCIOPATHIC PERSONALITY
          8    J GERONT
         11    AARONSON BS
          5    AGING
         11    PERSONALITY
          6    CHANGE
         11    PSYCHIATRIC
          9    DIAGNOSIS
         11    ADOLESCENCE
         21    ADOLESCENT PSYCHOLOGY
         16    MENTAL DISORDERS
          4    MMPI
         23    SOCIOPATHIC PERSONALITY

3  A  ABELY P, LAUZIER B
   T  (THE FATE OF THE CONCEPTS OF PERIODICITY, ATYPISM AND INCURABILITY
   T  IN PRACTICAL PSYCHIATRY)  (FR)
   S  ANN MEDICOPSYCHOL (PARIS) 122,729-46, MAY 64
   K  CLASSIFICATION, *MENTAL DISORDERS, *NOMENCLATURE, PERIODICITY,
   K  PROGNOSIS, *PSYCHIATRY, PSYCHOTHERAPY, SCHIZOPHRENIC PSYCHOLOGY
         17    ANN MEDICOPSYCHOL
          7    ABELY P
          9    LAUZIER B
          4    FATE
          8    CONCEPTS
         11    PERIODICITY
          7    ATYPISM
         12    INCURABILITY
          9    PRACTICAL
         10    PSYCHIATRY
          2    FR
         14    CLASSIFICATION
         16    MENTAL DISORDERS
         12    NOMENCLATURE
          9    PROGNOSIS
         13    PSYCHOTHERAPY
```

FIGURE 3

Samples from the dictionaries

AUTHORS

1		AARONSON BS
1		ABD EL NABY S
3		ABELY P
1	*	ABENSON MH
3		ABRAHAM G
6		ABRAMS S
1		ACHILLES M
10		ACHTE KA
2		ACKER CW
2		ACKER M

SOURCES

14		BEHAV RES THER
2		BEHAV SCI
4		BIOCHEM PHARMACOL
1	*	BOLL MAL ORECCH
2		BOLL SOC ITAL BIOL SPER
6		BRAIN NERVE
1	*	BRAIN
1		BRATISL LEK LISTY
117		BRIT J PSYCHIAT
7		BRIT J SOC CLIN PSYCHOL

KEY TERMS

2		CHLORALOSE
2	*	CHLORAMBUCIL
36		CHLORDIAZEPOXIDE
2		CHLORIDES
2		CHLORMEZANONE
1	*	CHLOROQUINE
1		CHLOROTRIANISENE
17		CHLORPROMAZINE TOXICOLOGY
287		CHLORPROMAZINE
34		CHLORPROTHIXENE

FIGURE 4

Sample Profile

LAFAYETTE CLINIC MEDICAL SDI SYSTEM
BIBLIOGRAPHIC SEARCH OF SDI SCHIZ MADE ON 9/ 4/68
TAPE FILE CREATED ON 6/22/68 FOR ARTICLES 1 TO 5025

PROFILE DESCRIPTION

PROFILE 10000 NAME SAMPLE PROFILE - COMMON MEETING LOCATION LC HIT LEVEL 2

MODIFIER	WORD TYPE	WEIGHT	KEY TERM
MUST	COMP	1	ABRAMS S
NOT	COMP	1	CHILD BEHAVIOR DISORDERS
	COMP	1	CLASSIFICATION
	COMP	2	DIAGNOSIS
	COMP	1	NOMENCLATURE
NOT	ROOT	1	RORSCHACH

FIGURE 5

Sample run times on a 2 μsec machine

Number of terms in profile	Number of articles found	Time in minutes
4	10	10
4	14	10
5	41	14
7	2	11
7	63	15
7	421	40
16	178	41
50	446	72

Search made on 4600 articles

FIGURE 6

Articles from Figure 1 which match the Profile in Figure 4

2 A AARONSON BS
T AGING, PERSONALITY CHANGE, AND PSYCHIATRIC DIAGNOSIS.
S J GERONT 19,144-8, APR 64
K ADOLESCENCE, ADOLESCENT PSYCHOLOGY, *AGING, DIAGNOSIS, *MENTAL
K DISORDERS, *MMPI, *PERSONALITY, SCHIZOPHRENIC PSYCHOLOGY,
K SOCIOPATHIC PERSONALITY

MODIFIER	WORD TYPE	WEIGHT	KEY TERM
	COMP	2	DIAGNOSIS

3 A ABELY P, LAUZIER B
T (THE FATE OF THE CONCEPTS OF PERIODICITY, ATYPISM AND INCURABILITY
T IN PRACTICAL PSYCHIATRY) (FR)
S ANN MEDICOPSYCHOL (PARIS) 122,729-46, MAY 64
K CLASSIFICATION, *MENTAL DISORDERS, *NOMENCLATURE, PERIODICITY,
K PROGNOSIS, *PSYCHIATRY, PSYCHOTHERAPY, SCHIZOPHRENIC PSYCHOLOGY

MODIFIER	WORD TYPE	WEIGHT	KEY TERM
	COMP	1	CLASSIFICATION
	COMP	1	NOMENCLATURE

5 A ABRAMS S
T A VALIDATION OF PIOTROWSKI'S ALPHA FORMULA WITH SCHIZOPHRENICS
T VARYING IN DURATION OF ILLNESS.
S AMER J PSYCHIAT 121,45-7, JUL 64
K DIAGNOSIS, DIFFERENTIAL, *RORSCHACH TEST, *SCHIZOPHRENIA

MODIFIER	WORD TYPE	WEIGHT	KEY TERM
MUST	COMP	1	ABRAMS S
	COMP	2	DIAGNOSIS
NOT	ROOT	1	RORSCHACH TEST

TrB

PROJECT CONTROL SYSTEMS

GUS HILDELRON

IBM

PERT - CPM Bibliography

① Moder and Phillips (Reinhold, 1964)
"PROJECT MANAGEMENT with CPM and PERT"

② B. J. Hansen (America House, 1964)
"PRACTICAL PERT" (Paperback)

③ James J. O'Brien (McGraw-Hill, 1965)
"CPM in Construction Management"

④ Antill and Woodhead (Wiley, 1965)
"CRITICAL PATH METHODS in CONSTRUCTION PRACTICE"

⑤ Archibald and Villoria (Wiley, 1967)
"NETWORK-BASED MANAGEMENT SYSTEMS"

IBM SRL References

A. PCS/360

① PCS/360 Application Description Manual
Version 2 H20-0222-2

② PCS/360 Program Description and Operations
Manual Version 2 H20-0376-1

B. PCS/1130

① 1130 Project Control System Application
Description Manual H20-0211-1

② 1130 Project Control System Operators
Manual H20-0343-1

③ 1130 Project Control System Program
Description Manual H20-0342-1

NAME OF PRIME COMMITTEE: Biomedical

SUBJECT: Computerized Organization and Maintenance of Files in the Automated Clinical Laboratory

SPEAKER'S NAME: M. Ball, J. Lukins, and W. B. Stewart

COMPANY SPEAKER REPRESENTS: University of Kentucky, Medical School

MAILING ADDRESS AND PHONE NUMBER: University of Kentucky
Medical Center
Lexington, Kentucky 40505
606-233-5000

DAY, TIME AND SESSION NUMBER: Tuesday, 10-11 a.m., T2C

NUMBER OF PAGES OF TEXT: 8 pages

Computerized Organization and Maintenance of Files in the Automated Clinical Laboratory

by

M. Ball, J. Lukins, and

W. B. Stewart

The purpose of this paper is to describe the organization and maintenance of clinical laboratory data files at the University of Kentucky Medical Center, utilizing an IBM 1800 Data Acquisition and Control System.

The data acquisition process will be described briefly, and the structure of the necessary data files outlined in detail.

Computerized Organization and Maintenance of Files

in the Automated Clinical Laboratory

The Clinical Laboratory at the University of Kentucky Medical Center is presently employing an IBM 1800 computer to acquire analog data from the laboratory's many autoanalyzers, choose peak values from this data, and calculate and report test results by interpolation against standard peaks. It has been found that the results obtained are more accurate, more precise, and more legible than was possible using manual calculations.

In order to accomplish this task, it is necessary that the computer maintain a number of internal files. One is composed of the data that is acquired as it is received from the autoanalyzers. Another, kept simultaneously, contains pertinent information about the patients whose specimens are being analyzed. These two culminate in a final file which associates each patient with his result calculated from the autoanalyzer data. The computer also maintains complete daily and historical tape files containing all laboratory findings, thereby eliminating manual filing procedures in the laboratory.[1]

Now let us analyze the construction of each of these files (refer to Figure 1 as an outline).

---

1. Lukins, J., M. Ball, W. B. Stewart, N. Hill and R. O'Desky, "Computerization in the Clinical Laboratory", presented to and to be published in the proceeding of April, 1968, Meeting of Common, Chicago.

The first task is to read the pertinent patient information from cards and store it in a disk file which we will call PTST. It is defined as a file of 1000 20-word records:

2(1000, 20, U, LA2)

To save precious disk space, each sequence of patients is preceeded by a header record containing information common to all of the patients which follow that header. The format of the header record is as follows:

Word	Contents of word	Format
1	Patient count (no. of patients following this header)	I3
2-3	Test code (unique for each laboratory test)	2A2
4-8	Test name*	5A2
9-12	Units of test*	4A2
13	Overflow address - indicates where any additional patients for this test may be found	I4
14-16	Date test performed	3A2
17-20	Zeros (Not used at present)	4I1

Once a header record has been established for a particular laboratory test, all patients requiring that test are entered into the

---

* Only the test code is read from the patient card: Other information about the test, such as name and test units, are retrieved from the standard file (see description) and placed in this record. This eliminates the need for punching that information into each test card.

file behind the appropriate header. The format for the patient records is:

Word	Contents of word	Format
1-2	Patient's hospital number	F7.0
3-14	Patient's name	12A2
15	Hospital location of patient	1A2
16-17	Zeros (not used at present)	2I1
18	Dilution factor	I13
19	Last digits of test code	1A2
20	Total urine volume, where applicable	I14

While the patient information is being recorded on disk, the computer is busily collecting analog data from the autoanalyzers, which must be analyzed and stored in a disk file which we shall call COLOT. Its file definition is:

I1(32, 320, U, LA11)

and its format is:

Word	Contents	Format
1	Number of peaks contained in this record	I13
2-3	Test code	2A2
4	Indicates where any additional peaks for this test may be found	I12
5-320	Peak values	158F10.4

Before test results can be calculated and reported, it is necessary that the computer know how many of the recorded peaks are standards, what the actual values of the standard peaks are, what the name of the test is, and in what units the test should be reported. This information is relatively permanent, and resides in a disk file which we shall refer to as STDF:

1(316, 33, U, LA1)

There is one disk record in file STDF for each laboratory test, containing the following information:

Word	Contents of word	Format
1-2	Test code	2A2
3-7	Test name	5A2
8-11	Units of test	4A2
12	Number of standards	I2
13-32	Standard values	10F10.4
33	Indicates whether to calculate optical density or % transmittance	I1

In order to facilitate fast access to any particular test within this standard file, the file is indexed in another disk file (we have called it INDX) which can be quickly scanned to obtain the disk record number of the desired test within the standard file STDF. This index is defined:

4(1, 316, U, LA4)

It is simply a list of the test codes in the same sequence as they appear in the standard file STDF, and can be used in a table look-up fashion.

Once the two files PTST and COLOT have been completed, the two may be correlated, record for record, to form a complete disk file which we shall call FFLE, defined as:

31(1000, 40, U, LA31)

It contains both sets of information in the following format:

Word	Content of word	Format
1-2	* Hospital number	1F7.0
3-14	* Patient name	12A2
15	* Location of patient	1A2
16-20	* Test name	5A2
21-22	** Test result	1F10.4
23-26	* Units of test	4A2
27-28	*** Technologist's initials	2A2
29-30	* Test code	2A2
31-33	* Date test performed	3A2
34-35	Zero (not used by this program)	1I1
36	* Urine volume (if applicable)	1I4
37-40	Zero (not used at present)	4I1

This completes the description of disk files used by the process. We have collected, analyzed, and correlated the information for one "laboratory run." But disk storage space is limited, so

---

* From File PTST

** From File COLOT

*** Retrieved from an incidental file beyond the scope of this paper

this data is now stored on magnetic tape to enable us to reuse the disk files for the next laboratory run. Each day's results are stored in a tape file which we shall call the Daily Master Tape. It contains a series of 40-word records which are identical to the records in disk file FFLE, and sorted by hospital number. As data is collected throughout the day, it is merged into this Daily Master Tape file in proper sequence by hospital number.

For ease of retrieval of information on any given patient, it it practical to maintain a tape file containing all laboratory information on all patients currently in the hospital. This file we shall call the Grand Master File, and it is kept current by daily additions and deletions. To maximize utilization of tape storage space, the patient information is written only once as a header, and is followed by all of his laboratory analyses. The format of the records is as follows:

(1) Patient Header Record (total record length 16 words).

Word	Contents of Word	Format
1	Record length in words	I12
2-3	Hospital number	1F7.0
4-15	Name of patient	12A2
16	Location of patient	1A2

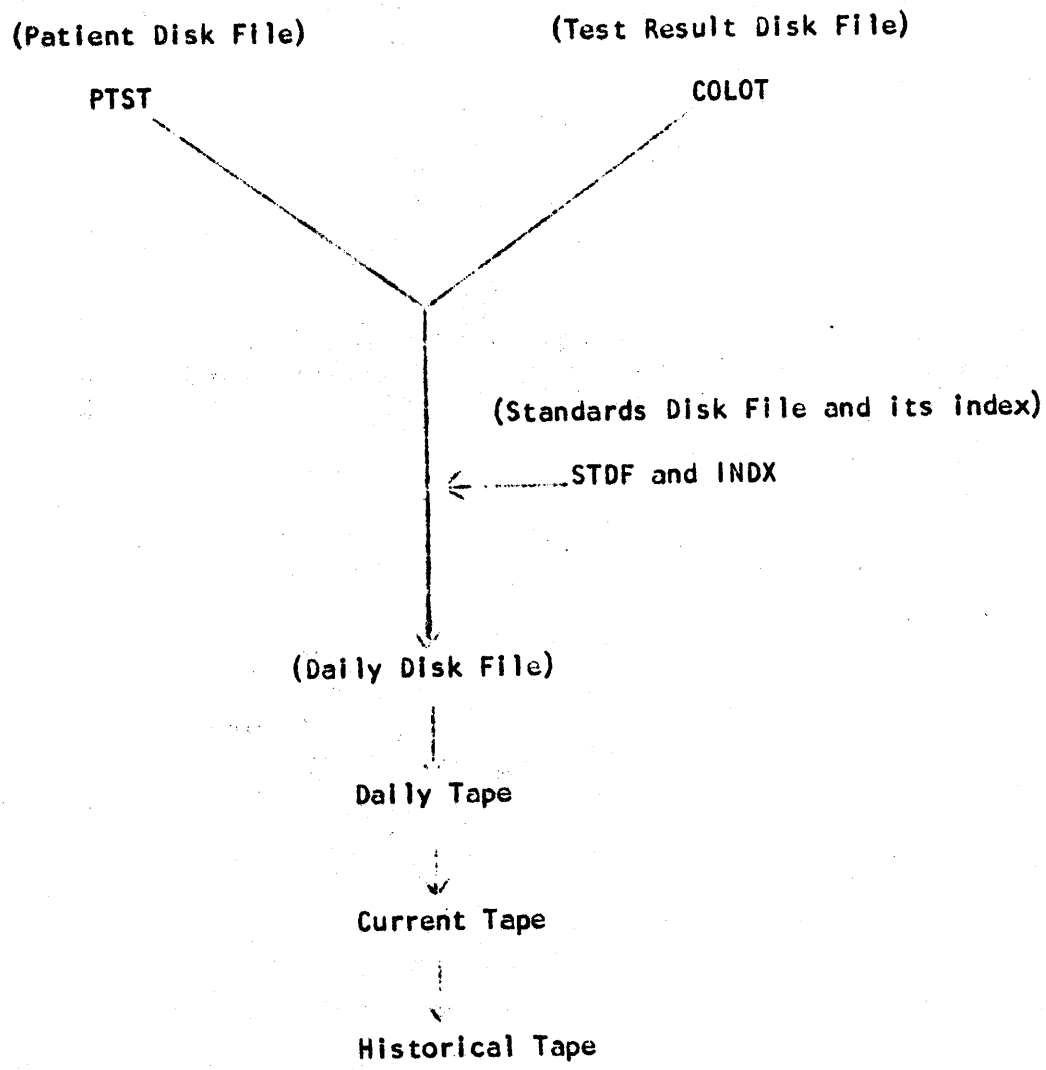
(2) Test Record (total record length 8 words).

Word	Contents of word	Format
1	Record length in words	I12
2-4	Date of test	3A2
5-6	Test code	2A2
7-8	Test Result	1F10.4

This brings us to the last major file, namely the Historical Tape file. This tape file is updated by the daily deletions from the Grand Master Tape file, since these patients have been discharged from the hospital. Their laboratory history must be recorded on this tape for future reference and for statistical purposes. The format of these records is identical to that of the Grand Master Tape explained above.

Within these major disk and tape files, then, all of the data collected in the automated clinical laboratory is contained in a systematic and easily retrievable form, and can be accessed without resort to manual filing techniques. It is a relatively new concept in laboratory management, and will no doubt require alterations and improvements, but when operating at full efficiency, it should significantly improve patient care.

Figure 1


(Patient Disk File)
PTST
(Test Result Disk File)
COLOT
(Standards Disk File and its index)
STDF and INDX
(Daily Disk File)
Daily Tape
Current Tape
Historical Tape

NAME OF PRIME COMMITTE: Biomedical

SUBJECT: Data Reduction Techniques in a Clinical Laboratory

SPEAKER'S NAME: R. I. O'Desky, Marion Ball, and W. B. Stewart

COMPANY SPEAKER REPRESENTS: University of Kentucky, Medical School

MAILING ADDRESS AND PHONE NUMBER: University of Kentucky
Medical Center
Lexington, Kentucky 40505
606-233-5000

DAY, TIME AND SESSION NUMBER: Tuesday, 10-11 a.m., T2C

NUMBER OF PAGES OF TEXT: 10 pages

Data Reduction Techniques in a Clinical Laboratory

by R.I. O'Desky, Marion Ball and W.B. Stewart

Within any clinical laboratory there are many problems caused by automation. At the University of Kentucky Clinical Pathology Laboratory, Dr. W.B. Stewart, Chairman of the Department of Pathology, is using a computer to assist him in more efficiently running his laboratory. The equipment Dr. Stewart choose to use in his lab is the IBM 1800 Data Acquisition and Control System Computer. The purpose of this paper is to discuss the most efficient use of the facilities of this system.

The process of data acquisition is the most important function of the 1800. Since this acquisition occurs on a cycle steal basis of a data channel, the 1800 Central Processing Unit does not use significant amounts of time servicing this analog point recognition. Therefore, within the system, the data acquisition is relegated highest priority.

Within any data acquisition computer there are three sources of storage available. These media are magnetic tape, disk, and core. Since magnetic tape is bulk storage media primarily used for extensive master files it will not be considered as critical to the laboratory data acquisition system. Its relatively slow speed and sequentially organized files are not appropoe for the volatile acquisition system.

The fastest storage medium available is core. The problem arises when it is realized that core is limited to 32,768 16 bit words of storage.[1] Since it is desirable to take advantage of the IBM supplied

TSX[2] programming system, a prime consideration is the maximization of variable core. This means that the user would like to make the most efficient use of the core storage which the Programming System requires. This then allows the user to have the maximum amount of core storage available for his programs which run as a foreground job in the time-shared environment.

The other storage medium is the replaceable disk cartridge. This means that limitless storage facilities are available on these interchangeable cartridges. In order to maximize efficiency in the system, it is desirable to keep the disk files as economical as possible. This is because the disk access time is much slower than core storage cycle time. An economical file reduces the number of disk seeks and reads, resulting in shorter access times for the pertinent information stored there. By keeping the files on a single disk cartridge no operator intervention is required. This means the system is never waiting for human response in order to carry out its function. Also, economical files allow more working storage to be available to the foreground programs running under time sharing.

At this point the speed, accuracy, and efficiency of storage media for the system are to be considered.

The ideal situation would be a point by point representation of the autoanalyzer output.[3] See Figure 1. This sequence of points would then be retained by one of the storage media. Since core is limited, this implies using disk storage to save all our point values. Operating on a continuous scan basis, we would become disk bound and would exhaust our disk storage in a matter of minutes. Therefore, physical limitations prohibit the

use of this method.

Two alternatives are now available to facilitate the user's handling of the large quantities of data generated by the autoanalyzer scan. These alternatives are, first, a hardware data reduction technique and, second, a software[4] technique. Two distinct hardware approaches were considered.

The first hardware method takes advantage of the comparator feature of the 1800.[5] See Figure 2. Since the range of the analog input signal is known, it is convenient to have the computer calculate 2% of this range.[6] Let this 2% of the range value be called DELY. The lower limit of the comparator word is set to the base line value which is determined by an analog read preceeding the initialization of actual testing. Let this value be called B. Then the upper limit of the comparator word is set to B + DELY. As we get interrupts we save the B + DELY value and update the comparator word by replacing B with B + DELY.[7] At the position on the curve where the comparator interrupts by going below the lower limit, we save the B value and replace B by B - DELY and B + DELY by B. In this manner we have defined the curve by a much smaller number of points.

This is a practical solution to defining the curve, but a large section of the disk is wasted by saving non-significant points. Also, an inherent error of 2% of the range appears in the answer. Thus, it is desirable to consider another hardware solution.

The second hardware solution uses the interval timers[8] which are available in the 1800. See Figure 3. The timer will trigger an interrupt at predetermined intervals. Servicing the interrupt consists of reading the analog signal, saving this value in either a core or disk table, and

returning the computer to the status it had prior to the timer interrupt. If the autoanalyzer tests are being run at a rate of 60/hour, this means that one peak a minute will result. A scan rate of once per second will result in a table of 60 points per test. Again with this method, a large number of superfluous points are being saved. Thus, the most efficient use of storage media is not being affected.

Thus far only methods of defining autoanalyzer output as a sequence of points have been considered. Now the problem of picking the peak[9] value of the curve must be considered.

The most straight-forward method of picking peaks is to scan the ARRAY of points which have been chosen to simulate the curve. The largest value is then saved as input to a program which interpolates this value into a meaningful final result. A combination of point-by-point curve definition and scanning to determine the peak is the most common programming approach used in a clinical lab today.

In order to optimize the use of storage media, a method of combining curve definition and peak picking would be highly desirable. The least sophisticated method of doing this would be a simple greater-than compare between successive analog reads. This method could be used with either a comparator or timer hardware approach to data reduction. The special cases which arise during the course of a test run make this method undesirable. A spike or a shoulder[10] is not detectable with this approach.

In viewing the entire system, the most desirable condition would be the investigation of every point on the curve, as in the first scan method mentioned, and the saving of only the significant point for each test. Before formulating an approach to combining these two methods, let us recall the definition of a derivative from elementary calculus.

Consider a function f, defined for values of the variable x in the interval (a,b). Let x be any fixed point of the interval and consider the ratio $\frac{f(x) - f(x_o)}{x-x_o}$ where $x \neq x_o$ and x is a variable point of the interval. The ratio is called a difference quotient.

Definition:[11] If the difference quotient approaches a limit as x approaches x_o, the limit is called the derivative of f at $x=x_o$ and is denoted by $f'(x_o)$. Thus by definition

$$f'(x_o) = \lim_{x \to x_o} \frac{f(x) - f(x_o)}{x-x_o}$$

provided the limit exists.

If $x=x_o+\Delta x$ the definition becomes

$$f'(x_o) = \lim_{\Delta x \to 0} \frac{f(x_o+\Delta x) - f(x_o)}{\Delta x}$$

This then becomes the equation which is to be considered. In the chain scanned system mentioned as the ideal situation the analog values corresponding to $f(x_o+\Delta x)$ and $f(x_o)$ would be saved in core. These two values are the only data points which have to be saved. Thus our core requirement is drastically trimmed by comparison to the ideal situation. The Δx is very small[12] and as a result the difference quotient can be considered as an approximation to the derivative.

Knowing that a valid approximation to the derivative is easily calculated within the 1800, it is now advantageous to consider the classical applications of the derivative concept. See Figure 4. In this situation it would

be convenient to be able to predict the value of $f(x_o+2\Delta x)$ from the given two data points $f(x_o)$ and $f(x_o+\Delta x)$. This is easily done by saying that $|f(x_o+2\Delta x)| \leq f'(x_o+2\Delta x) + \varepsilon$, and this is the criterion for eliminating spikes. Since $f'(x)$ is calculated for the previous purpose, further use of this value would increase the efficiency of time spent for doing the calculation. The second use would be to determine when the curve approaches a peak. The physical peak occurs at x_o when $f'(x_o) = 0$. Thus only points which have to be saved are in an area of the curve where $f'(x_o) \leq \varepsilon$ for εvery small.[13] This, then, minimizes the number of values which are to be saved on the disk. The criterion for the completion of saving points is that $f'(x_o) < 0$. Physically this means that the apex of the curve has been passed. The table of saved points is then scanned to find the maximum value.[14] This max value serves as input to the interpolation routine which, on the basis of the standards, yields the final reuslts of the laboratory test under consideration. This final result is then collated with the patient record to result in the final report.

Summary: This paper is the result of an attempt to maximize the efficiency of usage of both core and disk storage for an IBM 1800 DACS used in a clinical laboratory. The development of techniques is treated chronologically and culminates with the derivative method, which, experimentally and by calculations, appears to yield the most efficient and economical result.

1. The IBM 1800 DACS computer comes in core sizes of 8,192, 16,384, or 32,976 16 bit words.

2. Time-Sharing Executive System monitor

3. The autoanalyzer is the instrument in the Clinical laboratory which is the source of the analog signal which the 1800 recognizes.

4. Software is the user's program to direct the computer as to what to do.

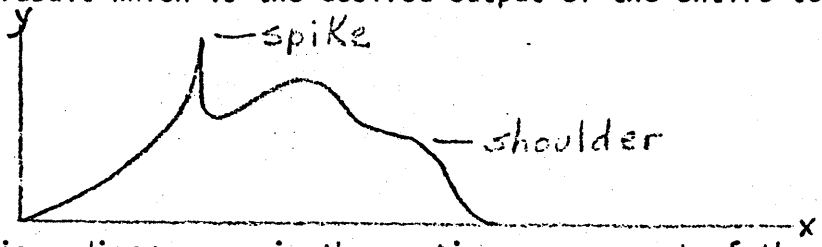
5. The comparator performs selective checking on the digital values converted by the ADC. A range type check is made to confirm that the converted values are within specified limits. The limits are obtained from the Multiplexer Address Table (one P-C cycle delay allows both limits to be acquired) whenever a check is required. The P-C is informed of an out-of-limits condition by interrupt. Def. taken from IBM 1800 Functional Characteristics manual A26-5918-5 page 77.

6. 2% is an arbitrary value which can be as large or small as the programmer desires. It is also the maximum error value for any peak reading.

7. This only considers the case for a increasing curve, but an analagous situation exists for the case of a decreasing curve.

8. An interval timer is an 1800 hardware feature which acts as a clock to keep the computer informed of time status and conditions.

9. The peak value is the maximum valid value on the curve and represents the test result which is the desired output of the entire testing system.

10. 

A <u>spike</u> is a discrepency in the continuous movement of the test curve. A <u>shoulder</u> is a peak which is of magnitude so small that there is no apparent rise in the curve to make the peak discernable.

11. ADVANCED CALCULUS by Angus E. Taylor pp. 14, 15

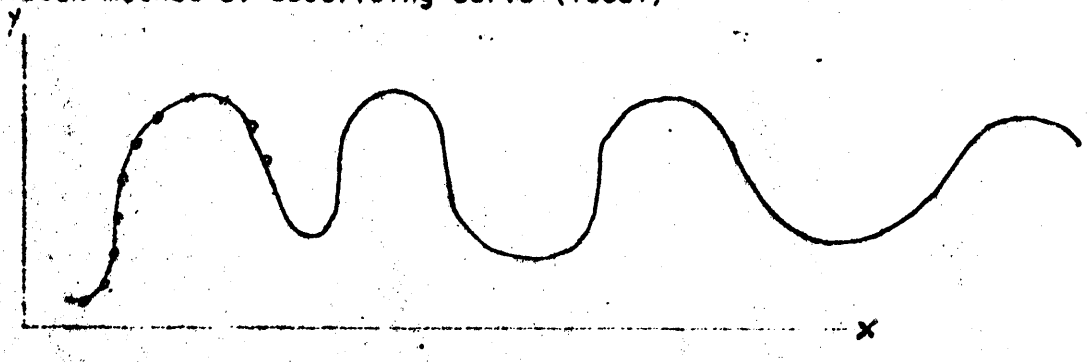
12. x is on the order of 58 * (the number of input signals) Micro sec.

13. For 11 bit resolution should be of the order of magnitude of the tenth bit.

14. This table is usually no more than four or five values.

Figure 1

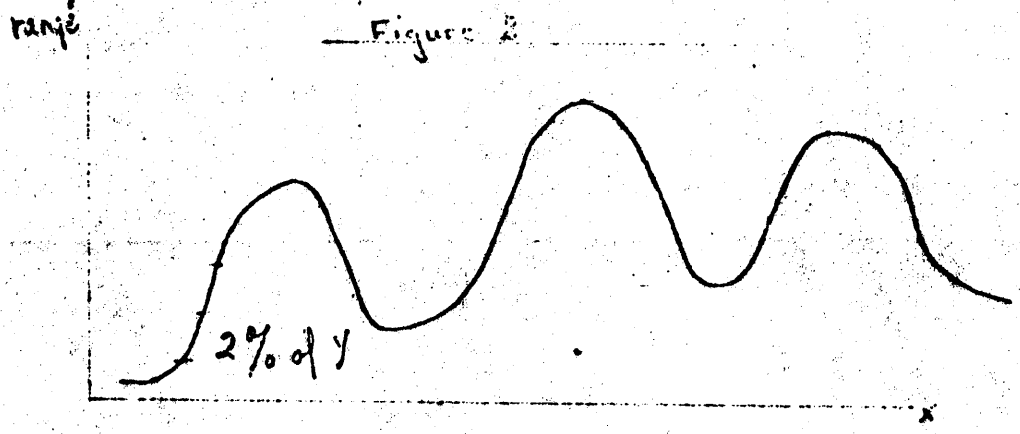
chain scan method of describing curve (ideal)



points are saved as rapidly as the multiplexor presents them to the computer

comparator interrupt method

Figure 2

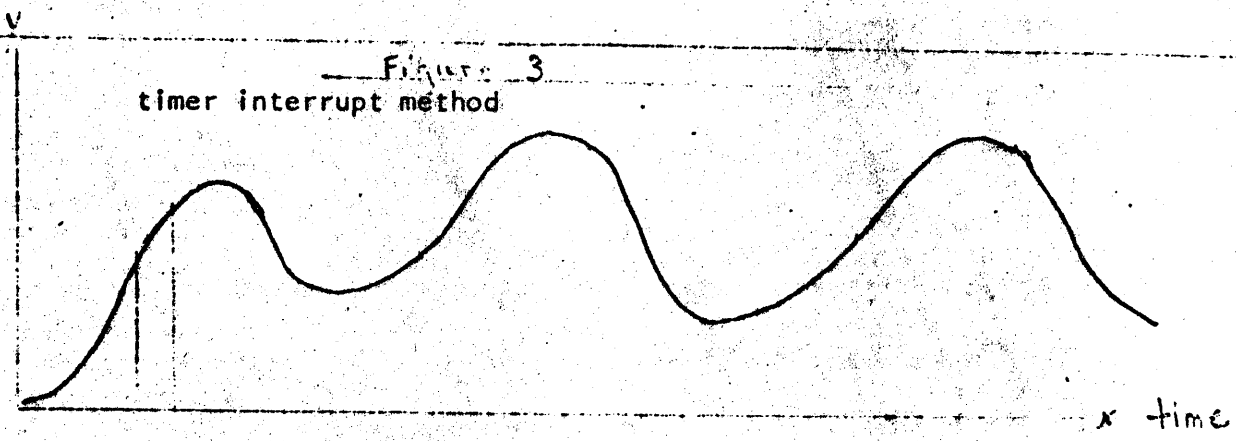


points are saved in increments of 2% of range

(a function of y)

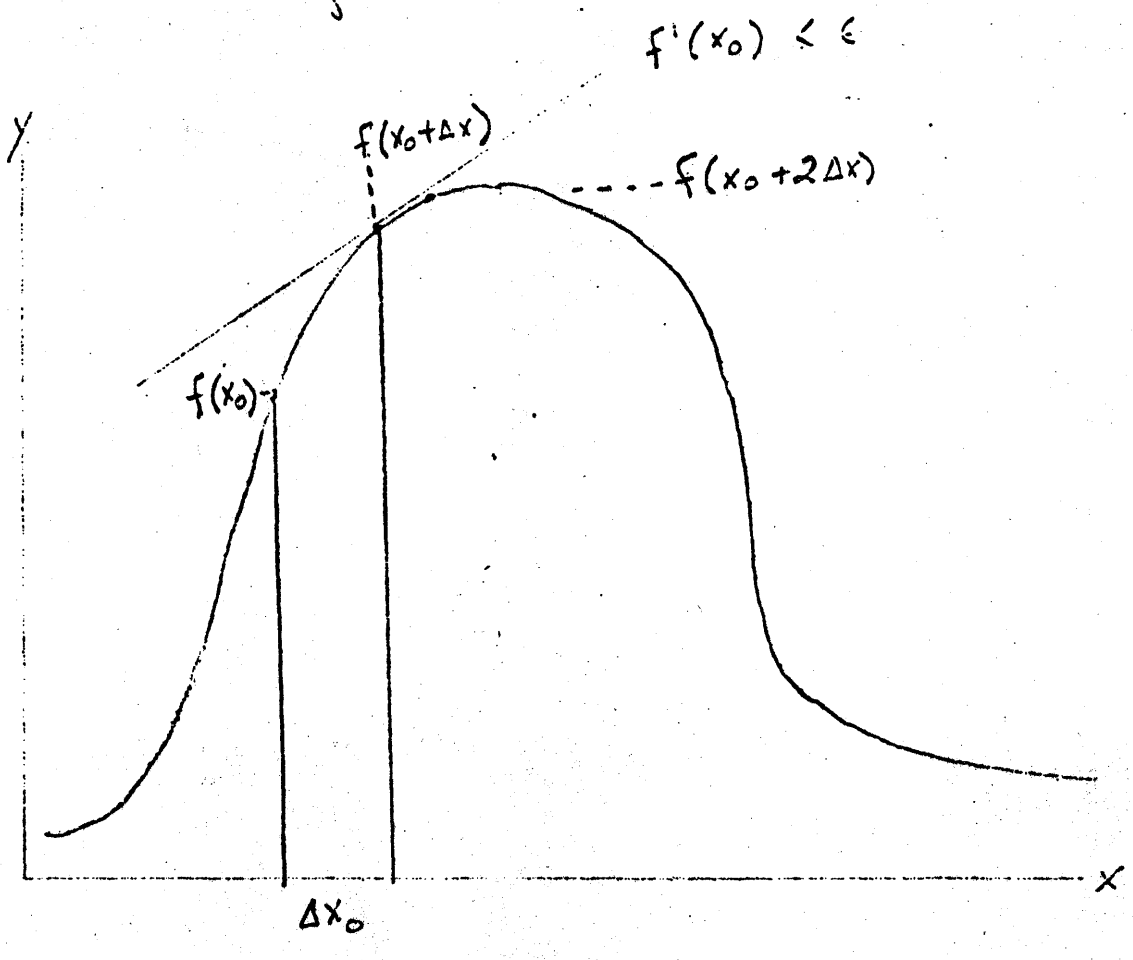
Figure 3

timer interrupt method



points are saved for fixed time increments

Figure 4



derivative considerations (superimposed point by point on Figure 1)

Installation Management Committee
Personnel Project

PROFESSIONALISM IN PROGRAMMING

Dr. Paul S. Herwitz
IBM Corporation
Old Orchard Road
Armonk, New York 10504

(914)-765-4543

Philadelphia COMMON
Tuesday, September 10, 1968
11:00 A.M.

10 pages, text only

PROFESSIONALISM IN PROGRAMMING

I am going to start by defining a profession. Next we will look at important characteristics of professions and see why programming ought to be one. Then we will look more closely at programming to see some of the places I believe it falls short of being a profession.

Probably as good a definition of profession as I have found comes from Webster's New International Dictionary, Second Edition, and goes like this: "A profession is a calling in which one professes to have acquired some special knowledge used by way either of instructing, guiding, or advising others or of serving them in some art". I think for our purposes the key words in this definition are special knowledge and serving others. If we think about other activities that are generally accepted as professions such as the three so-called "learned professions" - Theology, Law, Medicine - it's easy to make a list of major attributes that seem to characterize professions. They are as follows:

Special Knowledge
Service to Others
Ethics
Standards
Language
Control
Work Structure

Everyone can think of professions in which each of these characteristics stand out. For example, special knowledge and service to others are certainly characteristic of Law and Medicine. Both Law and Medicine also have ethics based on longstanding tradition. Engineering is a perfect example of a profession that is governed by many standards. (In this respect, Programming is coming along). Law, Medicine, Engineering, and Programming all have their own language. Probably due to the fact that the professions generally provide some service to others, there is usually some sort of outside control imposed on them; both Engineering and Architecture require state licensing, Medicine requires passage of state board examinations, Law requires admittance to the Bar Association, etc. Again many of the professions, if not most of them, are characterized by some kind of work structure; in Medicine we have the doctor, the nurse, and the lab technician each performing his own part of the total function; similarly in Law, we have the attorneys, their law clerks, etc.; architects use draftsmen, designers, etc., in profusion; in Engineering a good part of the work is performed by "non professional" technicians. In particular, we

we will look at three of these characteristics in light of their applicability to Programming. These are work structure, special knowledge, and service to others.

In order to understand the question of work structure, let's first see what Programming is all about. A rather simple definition of Programming is the following: Programming is the technique of designing and preparing procedures and instructions that direct computing systems in automatic information processing and problem solving. It's perfectly clear that by this definition special knowledge is required and Programming does provide a service to others. I guess it differs from a craft in that Programming generally makes use of intellectual skills, while most crafts usually require manual skills. At any rate, if we take this definition at face value, it looks like Programming ought to be a profession. Let's look a little closer at what the various parts of the job actually are and show by analogy with Engineering, for example, that it indeed ought to be a profession.

Looking at the programming process itself, I have identified eight parts as follows:

1. Problem conception, or identification of requirements.
2. Problem analysis.
3. Statement of objectives.
4. Program specification.
5. Program design.
6. Program development.
7. Program implementation.
8. Program maintenance.

Interestingly enough, we don't actually start working on the program itself until the first four parts are dealt with. Incidentally, some people would say that the problem is solved and the intellectual challenge is gone by the time program specifications are prepared. I think this is rather an over simplification and yet it's at the heart of the question of structuring the work. We'll come back to this. I think most of these parts named are pretty clear, but very briefly every process must start somewhere; hence the identification of requirements or problem conception. Problem analysis, of course, is in attempt to find out precisely what is wanted. The statement of objectives is really a question of documenting the agreed upon job to be done. Program specifications are functional and performance parameters needed to reach the objectives. Program design is the determination of the actual procedures that should be used to fulfill these specifications. Program development is of course writing and testing. Program implementation is putting the program into operational use. And finally, program maintenance of course is correcting errors, modifying, and updating the program. I think that intellectual skills are used in essentially every stage of this process although some of the parts become somewhat routine in time.

I want to digress briefly to make an excuse for programming! I think that a good part of the problem attendant upon attaining professional status really comes from growing pains. We should take a few moments and recognize how programming has grown in a short period of 21 or 22 years. (It's just because of this rapid growth that we have not as yet imposed a proper work structure on it, we have not provided the proper organized foundation of knowledge, and we have not learned how to best serve others). Let's look at the record.

The following table shows estimated growth of computing in the United States.

1930	Differential Analyzer	
1944	ASCC (Mark I)	
1946	ENIAC	
1950	15 computers	1,000 ops/sec.
1966	35,000 computers	1,000 K ops/sec.
1970	50,000 computers	?
1975	85,000 computers	?
(C&A 1967	51,000)	

We really date our consideration of modern computers back to 1930 when Vannevar Bush put the Differential Analyzer into operation at MIT. This was the first important automatic problem solving machine. However, it was not the kind of machine we are most concerned with here, as it was an analog computer. The Automatic Sequence Controlled Calculator, better known as the Mark I, was built by IBM for Harvard University. Really this was like a set of interconnected desk calculators and relays. The true beginning of the "modern age of computing" was in 1946 when the Moore School of Engineering at the University of Pennsylvania put the first digital computer into operation. This machine was the ENIAC and was really the first all electronic digital computer. By 1950 a rough count of digital computers in this country showed something of the order of 15 (and they performed at a maximum speed of about one thousand operations per second). Business Automation and Computers and Automation both indicate in 1966 that there were about 35,000 computers in use (the maximum speed of operation had already become about one million operations per second). The 1970 and 1975 estimates are from a projection made by the American Federation of Information Processing Societies, however, Computers and Automation said in 1967 that there were 51,000 computers already in operation. The importance of this list of course is that from 1946 on, all of these computers required programmers to operate them.

No one actually knows how many programmers and analysts there are in the country today. The best estimates have been made on the basis of surveys of computer users that asked how many programmers were required for each computer. By comparing all published estimates and by using surveys conducted by IBM of its own customers, I have come to the conclusion that from a handful of programmers in 1946 we probably went to about a quarter of a million programmers, analysts, and managers in 1965; and we can expect

this to be about half a million by 1970, and something like three quarters of a million in 1975. Incidentally, an AFIPS estimate is roughly in the same ball-park. I think that today there are about four hundred thousand programmers, analysts, and managers. Over the same period of time IBM's programming staff has grown at least as rapidly; and although IBM's programmers represent a very small percentage of the total in the country, by looking at ourselves we probably have the best available opportunity for examining and understanding what has happened to programmers from a professional point of view. Therefore I am going to use what's happened in IBM as a base from which my conclusions about the profession in general will be drawn.

I'd like to look now at the development process and see who gets into the act. They come in all shapes and sizes, but it is surprising how many different kinds of people do very similar work. One of the problems I face in IBM is understanding just who wants to be called what. We have programmers, systems analysts, planners, engineers, systems engineers, and customer engineers--all of whom get into programming in one way or another. So let's take a close look at two examples of the development processes--first, software development and second, hardware development. I want to show that there is a very strong analogy between the two processes, and that the activities of those involved in each are quite similar. We have already listed eight parts of the programming development process and we could write a similar list of parts of the hardware development process. The only difference that we would find would be that the sixth part, program development would become hardware development and hardware manufacture. If you will just substitute the word systems wherever it appears in the original list for the word program then you have the process that applies both to software and to hardware development.

Now who are the people who take part? In IBM programming requirements are identified by any number of people; it may be by the programmers themselves or by people in the Data Processing Division - that is, our systems engineers, our marketing people, and so on. As far as hardware is concerned, again it may be that the requirements are identified by our marketing people, our systems engineering people, or the development engineers themselves. People who identify requirements tend to overlap with the people who do the requirements analysis; in the case of software this means systems planners and/or systems analysts and/or systems programmers, and for hardware its generally systems planners and/or development engineers. These same people also usually write the systems objectives. When it comes to specifications and design, then either the systems programmers are involved for software or the development engineers for hardware. The completion of these five activities finishes the design phase of the systems development process.

The second phase is the build phase or the development part for software systems, and the development and manufacture parts for hardware systems. In Programming this is pretty strictly the job of the systems programmer, although there is some overlap with customer engineering at times. For the hardware system it's the job of the development and manufacturing engineers.

The final phase is the use phase which includes installation and maintenance . For software systems, this is combined province of the systems programmer and the customer engineers. For hardware systems this is pretty much the job of the customer engineers alone.

So although there are minor differences between the processes and major differences in the output, generally the work is almost identical in function. Thus again by analogy Programming ought to be a profession since Engineering is a profession.

Now let's try to get a different view of the kind of work done in Programming. We'll confine ourselves to those categories of people that we call planners, systems analysts, and programmers. In IBM planners come from two places, from programming and from engineering. Systems analysts, however, primarily come out of programming. Occasionally we find some people who come to IBM with a particular speciality (such as mathematics, for example) and who show the immediate ability to apply their knowledge to the problems at hand and to almost instinctively lay out solutions to these problems in the manner most useful to programmers. However, most of our people go through a rather prolonged apprenticeship in Programming. As they become more experienced, most of them begin to perform more and more as data gathers and interpreters. Some of these people become known as the systems analysts, some planners, while the others continue to be called programmers. After the apprenticeship of two or three or four years, whatever it may be, all three groups become journeymen. As they become more and more expert they spend more and more of their time in problem analysis, and design activity, although they may be analyzing different kinds of problems. The planner usually does his job before the hardware or program system is built; the systems analyst does his job after the hardware and software is available; the systems programmer tends to do his work somewhere in between; and the applications programmer, of course does his analysis whenever the applications to be programmed are conceived. So, to my way of thinking, the kind of skill and intellectual activity applied is really very much the same for the planner, the systems analysts, and the programmer, if one takes into consideration the humble programming origin of most of them and the final destination of all of them in problem analysis activity. One can analyze engineering activity in very much the same way, and find again by analogy Programming must be professional in nature.

Let's look now at the IBM career opportunities and jobs that are open to programmers, systems analysts, and planners. We have five levels of exempt positions. They are known as associate, senior associate, project or staff, development or advisory, and senior. Our senior associate level is what corresponds generally to the senior programmer level in business and industry. Project and development programmers are managers, staff and advisory programmers are non-managers. These titles apply not only to programmers but to systems analysts and planners as well. Among non exempt personnel we have two lines of progression, one for the college graduate and the other for the non college graduate. The college graduate comes in to IBM as a student briefly and then becomes a junior programmer. On the average he is eligible for promotion to exempt status in twelve to eighteen months. The non college graduate comes in as a programming technician trainee and can advance through three levels known as programming technician, senior programming technician, and finally programming specialist. A programming specialist who shows the ability to do exempt work can be promoted to associate programmer. So I think we have everything that is necessary for work structure. We have some eight or nine levels of jobs starting from technician trainee and moving up to the most senior level, we have exempt and non exempt positions, and we have personnel performing at all those levels. The problem then is why not formally structure the work in line with the work process?

I mentioned before that some people feel the most intellectual activities are finished when the design phase is done. Why should not this be the work of the exempt personnel and the remaining work be that of the non exempt personnel? Well, many managers think this is the right idea, and many think it is the wrong way to go. Why does the latter group believe its wrong? - well for the following reasons: First of all, the pressures due to business growth say that the risk is too great. What happens is we have guessed wrong and indeed technicians can't do the development process in satisfactory fashion? Secondly, its difficult to structure the work and quite time consuming in practice. And finally, its a tough managerial problem to have a large number of non exempt people pushing up to the exempt rank, because in practice its hard to distinguish work actually being done by low level exempt people from high level technicians. This results in unhappy technicians who feel they are getting short changed: - they feel they are doing the same work as exempt personnel who are recognized as professionals and paid according to exempt scales. And the technicians may indeed be right, because programming is an extreme case of a mixture of legally exempt and non exempt work. The parts of the law that apply to professional exemption for our purposes are as follows:

1. The employee must have as his primary duty work requiring knowledge of an advance type in a field of science or learning.

2. Work must require the consistent exercise of discretion and judgment.

3. Work must be predominantly intellectual and varied in character as opposed to routine mental, manual, mechanical, or physical.

Clearly, the work a programmer does requires the consistent exercise of discretion and judgment; and clearly, much of the work that's done -- particularly in the analysis of the problem -- requires knowledge of an advance type; finally it's also clear that much of the work is predominantly intellectual and varied in character. However, it is also quite true that much of the work is a routine, plodding, rote kind of mental exercise.

Now historically programmers first came from academic ranks in the schools that first built computers, and from scientists in the large scientific installations that first used computers. The tendancy was for these people to do the entire job from problem definition through maintenance. Thus tradition says that programmers will be college graduates and will continue to do the entire job, and we must fight tradition if we are to structure the work. For it's clear that a structure could be imposed, and would probably reduce the cost and might improve job satisfaction. I think that a lot of the work that is done by professional exempt programmers is so routine that it leads to early disillusionment, job hopping, and general dissatisfaction.

I would now like to turn my attention to the special knowledge that programmers and analysts must have. In IBM the growth of the programming population has been so great that although our average experience level is about three and a half years it will take approximately three years for this average level to increase by a year. In this first three or four years a programmer probably has had at most a couple of assignments, and probably both in the same area. We don't want to take the risk of putting him in a different area that might expand his horizons. But we know that the higher level jobs require both breadth and depth of experience. One might consider an educational career path something like the following. The formative years, say the first five years in the business, conform somewhat the first four or five years in college. We could postulate that at the end of the formative years ones experiences should be such that he qualifies for a bachelor's degree in information processing. I call the years beyond the first five the years of impact. Maybe the next two years would be equivalent to getting a master's degree in information processing. The people at this level generally are the technical work leaders and the new first line managers. The next two or three years could be considered to be the preparation for a doctorate degree; these people are the problem solvers, the advisors and the second line managers. Finally, those people who remain for ten or more years ought to be experienced enough to be considered to have a doctorate in information processing. In IBM these are the senior programmers, senior analysts, senior planners; they are the experts and the strategists; they are the recognized authorities in their field.

In order to try to get some additional depth and breadth of experience for the programmers in IBM we have introduced two programs in advanced education. The first we call Intermediate Programmer Training, and the second we call the Advanced Programming Option at the Systems Research Institute. The objectives of Intermediate Programmer Training are to introduce breadth in such areas as languages and processors, supervisors and monitors, communications and real time systems, machine organization and systems architecture, selected applications, and special technology. Hopefully depth can be introduced in at least one area other than the present assignment of the programmer. The program is designed for all programmers who have completed at least two years in IBM. The only way to implement such a program is to introduce formal course work, to require selected readings from the literature, to attempt to select job assignments, and to encourage participation in special seminars. So far, we have implemented a number of new courses; and about five hundred students have gone to some twenty classes as of June this year. Titles of some of the courses are Math and Logic, Probability and Statistics, Introduction to Telecommunications, Time Sharing Concepts, Micro Programming Techniques, Performance Measurement, Compiler Design, Modeling and Simulation, Languages and Translation, and File Organization and Data Management. We would like to see everyone with two years experience spend at least one week a year (and preferably two) taking such courses.

The Systems Research Institute is a Corporate sponsored activity dedicated to graduate level education and research for professional systems people from all divisions of the Corporation. Courses are offered in three departments in SRI, Systems Design and Analysis (these courses discuss the systems design and development process), Systems Architecture (discussing the organization of and techniques used in information processing systems and sub-systems), and Systems Disciplines (courses discussing the formal subjects required to understand the tools, systems, and applications). In September we are offering a number of new courses open only to IBM programmers who have had a minimum of five years experience. The kind of subjects taught in these courses are of course very similar to what is taught in Intermediate Programming Training, however they proceed at a much more sophisticated level. Some of the titles are Design of Software Systems, Programming Systems Development, Fundamental Algorithms and Their Use, Storage Management Concepts, Parallel Operations and Computers, Systems Performance Measurement, Operating Systems Theory and Practice, Comparative Analysis of Programming Languages, and Combinatorial Information Structures and Algorithms for their manipulation.

A major question, of course, is where lies the burden of training all the programmers that will be needed by business and industry? IBM finds it a struggle to provide all the advanced training thought to be necessary for its own people. When we think in terms of another quarter of a million people coming into the field within the next few years, it's perfectly clear that the burden must be on at least part of the academic community. It seems to me that the most reasonable place to prepare one for a programming career is in universities, colleges and especially the two year community colleges and vocational schools.

Now let's look at the question of service to others. I am afraid I won't have too much to say about this because this is probably the most difficult aspect of programming to evaluate today. The question of service is intimately connected with a measurement of the value of programming, and this is directly related to quality. To illustrate what I mean I will try to tell you something about what programming means to IBM. First of all, we support our products by both systems programs and our Type II applications programs. Secondly, programming provides internal administrative support through the management information systems which we are developing, through the personnel data system, and so on. Third, we provide a talent pool for our customers to use via contract in both our Service Bureau Corporation and in our Federal Systems Division. Fourth, programming helps us improve the efficiency of our marketing, installation, and maintenance activity. This comes about because of special tools developed for the customer engineers to use, by the simulation of complex installations, etc. And finally, we provide an actual software product service which is a source of revenue for us (exemplified by QUIKTRAN). Now the thing that we find difficult to do is to actually determine the dollar value of these various activities to IBM. And since we are going to be increasing our programming development expense over the next six years or so, we need some way of comparing the value and the cost. A key measure to us is how value changes with increasing costs. (If you think about it for a moment, you see that if value is a function of cost, then the derivative of this function represents profitability). Now it is easy enough to measure the cost of programming. But it is very difficult to quantity value. Value is intimately connected with quality and quality is something that none of us have really paid enough attention to in this youthful profession.

I think the improvement of quality is something that you do "in the small". By this I mean that outside of such general considerations as does the data processing installation need fulfill its function in the company, one must have an intimate knowledge of the actual programs produced in order to judge their quality. Such knowledge becomes a responsibility of the very first level of management. Here one can exercise some control on the quality if one's evaluation program includes a close examination of the work that is actually done. Let's not ask is this programmer really a good guy, let's ask how good in fact is the work that he has done.

Quality has both external and internal aspects. Among the external aspects are considerations such as was the program on schedule? If it was late, why was it late? If it was early was it all there? Was the size on target? Was it too large or too small? What was the program performance? (This is a difficult thing to measure but at least one can ask is it better or worse than comparable programs). And how about the functional capability asked for? Was it all there, if not, why not? (Typically the answer would be that parts of the problem were found not to be feasible). Was there too much function there? If so, what was the cost of this extra function? All of these items I

consider to be the job parameters. The other external aspect of quality is the question of correctness. How well was the program debugged? What were the nature of the errors? Did these errors show a lack of understanding? If so, what are the hidden problems that we haven't discovered as yet?

Internal aspects of quality include such things as program documentation, the use of proven techniques versus innovation and the question of the general elegance of the work. By elegance I mean simplicity and lucidity, novelty, and logical step-by-step development of the program.

These are aspects of programming that one has to begin to look at very carefully. I have a feeling that most programming that has been done has indeed served its purpose, but the cost of much of it was very much beyond what was expected. In part this is due to the newness of the game, and partly it is due to a lack of understanding of what quality programming means. All the other professions judge quality in some fashion: the quality of the surgeons work is judged by the recovery of his patients, the quality of a lawyers work is judged by either his record of convictions, or his record of successful defenses as the case may be; the quality of the engineers work is generally measured by some sort of strict quality control procedure. Programming doesn't have a quality control procedure as yet and indeed it's probably one of the most difficult aspects of making programming truly a professional activity. But it is something that we must learn to deal with.

I have talked for some time now and have tried to convey to you some of the questions I become concerned about when people talk about professionalism in programming, and about the profession of programming. I hope this has given you some insight into some of the problems. To a great extent the degree by which programming becomes truly professional will depend upon its ability to mature. And maturity only comes with a certain amount of painful introspection.

NUMERICAL CONTROL LANGUAGES

FOR THE SMALL COMPUTER

Neil J. Mizen

September 10, 1968

TABLE OF CONTENTS

INTRODUCTION

We are at a point in time when computer languages available to assist programming numerically-controlled machine tools are not readily useable or satisfactory for a large number of N/C programmers. This situation has come about because most computer programs available are useable only on relatively large computers, which, in turn, are not conveniently or economically available to many people. Thus, often organizations that wish to program parts utilizing a computer are unable to do so because of high incurred cost and undesirable operational factors.

One alternative is to utilize an inexpensive computer that can be justified largely on the basis of numerical control programming. Many advantages exist for this approach. The difficulty, however, is that computer manufacturers do not supply numerical-control processors for the small computer that are satisfactory to users.

This paper describes computer languages that have been prepared at Anderson Bros. Mfg. Co., so that an IBM 1130 computer which has a core memory of 8000 words and a single disk drive can be utilized to program N/C machines. Two computer languages are described. One is particularly useful for machine operations that require generating contours, while the other is particularly convenient for point-to-point operations.

2. ADVANTAGES OF THE SMALL COMPUTER

The inherent low cost of a small computer enables an N/C user to justify installation of a small computer for a few special purposes. Thus, the system allows one to avoid the complexities of scheduling and sharing computer time. The computer is therefore more readily available, and, equally important, processing time becomes less significant. Consequently, the use of a plotter becomes practical since the processor speed is not of major importance. Thus the user can well afford to match the output of the computer with the needs of the part programmer.

The low cost has a secondary benefit that must not be neglected. By reducing the computer cost, the cost of each tape prepared is reduced. The cost of preparing the tape is a significant factor in selecting parts which may be econonically machined on the N/C machine tool. Thus, lowering the cost of punched tapes allows the machine tool to be utilized for more parts. The same resultant occurs when the computer input and output are made to match the needs of the part programmer. The punched taped prepared are more accurate, and therefore the likelihood of a machine tool being idle due to bad tapes is lessened. Thus, again, more parts can be profitably programmed for the N/C machine.

3. SOFTWARE

Certain requirements exist for useful numerical-control processing languages. Clearly, the final purpose of the computer program is to allow the part programmer to obtain a punched tape in the most convenient economical way. To accomplish this efficiency of operation it is necessary that the language be intuitively logical and reasonable to persons who normally think in terms of machine tools; for it is these people that will determine the final success of the program. Fortunately, the languages need not allow for all possible piece-part configurations, however, because frequently such complexities result in more confusion than in real benefits. Finally, the processors must accomodate both point-to-point and contour work pieces, for when an organization begins using computer-assisted programming it is desirable to be able to program all machines in a similar manner.

3.1 POINT - TO - POINT PROGRAMMING

The computer programs that best assist point-to-point programming have certain major differences compared with contouring languages. These differences occur because point-to-point programming must treat sequences of machine tool motions rather than only cutter motions. A second consideration concerns the type of machining operations typically performed. Sequences of motions are rarely repeated in contouring and frequently repeated with point-to-point programming.

Consider machining the work piece shown in Fig. 1 utilizing a tape-controlled drill. Although the part can be easily programmed manually, an example of how it could be p rogrammed utilizing a computer program, named ANDRP, follows. The purpose of selecting this simple part is to demonstrate the approach used, so that it can be shown later how the procedure has been adapted into more complicated situations, and to show certain basic benefits of computerized programming. The part program shown in Fig. 2 would machine this part on a Cintimatic Tape Drill.

Admitedly, the program discussed above could be programmed manually with ease. However, in so doing significant benefits are lost. The plotter output is not available to verify accuracy of the input data. The input data is not available or documented in a manner convenient to allow future changes. An estimate of required machining time is not readily available. The part program must be completely

rewritted if the part is to be machined on a different machine. The manuscript must be manually typed to acquire the punched tape. Thus, there is justification for utilizing computer-assist programming even for extremely simple parts that are to be manufactured on machines that possess motion cycles.

If the part is to be manufactured on a machine tool that does not have pre-programmed cycles, additional justification for utilizing a computer is apparent. With manual programming the proper speed and feed-rate must be specified for each incremental motion. Thus, far more opportunities for errors are present because far more tape sequences are required. The part program required to machine the part shown in Fig. 1 utilizing a more complex machine (the machine selected is a Milwaukee-matic Series Eb) is shown in Fig. 3. The part program shown in Fig. 3 is similar to the program shown in Fig. 2, even though the machine tools selected are basically different. The only differences are those made necessary to account for the differences in the axes and geometry of the machine tools.

If the work piece is only slightly more complex, the task of manually programming the part may become unmanageable, while the computer part program is still similar to the simple part shown previously. Fig. 4 shows another part that could be machined on the Milwaukee-matic Eb, and Fig. 5 presents the computer output.

3.2 CONTOUR PROGRAMMING

The justification of utilizing a computer for contour programming is evident from the fact that a large number of parts, frequently encountered, simply cannot be programmed manually in a practical nammer. A program named ANDR has been prepared so the advantages of a small computer (the 1130) can be acquired for contoured parts.

The philosophy used in preparing ANDR was that it should serve the needs of controlling contouring machines in a simple, direct manner. We were constantly aware that the users of the program would be persons skilled as machinists, not persons skilled as computer operators or programmers. Thus, the basic approach of AD-APT was retained, although modified slightly. The language of ANDR includes means of describing the geometry of a work-piece using points, lines and circles, and means of directing the cutting tool to generate the contour defined. ANDR is a two-axis language. Output of the program includes a plotted path of the work piece and the tool path, and punched tape, as well as listings of the input cards and cutter-location files. The program is operable on an IBM 1130 computer with 8000 word core memory.

The following presents most statements allowed, and a brief description of how each is used. The program ignors blanks, and data may be placed anywhere in the card. Integer numbers may be shown with or without a decimal point. The vocabulary statements are divided into three categories:

1. General statements used to control the program and certain machine-tool functions.

2. Geometric statements.

3. Machine-tool motion statements.

GENERAL STATEMENTS

MACHIN/name, n

The MACHIN statement is used to specify the machine tool to be used, and to control certain parts of the ANDR program.

The name of the machine tool is placed after the slash. Use of the number "n" is intended to avoid confusion if more than one kind of machine tool is given the same name. Its use is optional if the name is unique. For example, the statement

MACHIN/LEBLD

causes the LeBlond post-processor to be called.

The statement

MACHIN/c

causes an extensive cutter location table to be printed, which can be used to isolate certain errors in the ANDR program, and to aid in preparing additional post-processors.

The statement

MACHIN/CPLOT, n

causes the plotter sections of the program to utilized. The number "n" is the scale factor used; if "n" is a positive number the work-piece contour is plotted; if "n" is a negative number the path of the tool center is plotted.

REMARK/message

The REMARK statement causes the message placed after the slash mark to be printed with the listing of the input cards. No other action is taken. This statement can be used to indicate the functions of various parts of the program.

COLUMN/n

The numeric specified in the COLUMN statement is the column beyond which the processor will not scan. The entire card is printed, however, and thus the remaining columns may be used for remarks. If this statement is not used all eighty columns are scaned. If a column number is specified it remains in effect until it is changed or until a new part is processed.

PPRINT/message

The message specified in the PPRINT statement is printed in the post-processor listing. No other action is taken. This statement is often used to instruct the machine-tool operator in regard to settings and procedures.

INSERT/characters

Characters placed after the slash mark of the INSERT statement will be converted to E.I.A. standard coding, and will be punched verbatim onto the tape. Every character desired must be given, including the sequence number. The end-of-field character is automatically inserted by the computer.

PARTNO/part number

The PARTNO statement is used to punch the part number onto the tape (inE.I.A. standard coding). The part number specified is also written on the plotter output and in the post-processor listing. It must be the first command delivered to the post-processor, otherwise it will be ignored. A good practice is to use the PARTNO/statement for the first card in the program.

LEADER/n, O

The LEADER statement is used to generate leader. The number is used to specify the length of the leader, (measured in inches). If the letter "O" is used odd parity is punched onto the tape; if the letter "E" is used even parity is punched. If parity is not specified odd parity is assumed.

STOP
STOP/OPT
END
FINI

The STOP statement will cause the machine tool to stop. The STOP/OPT statement will cause the machine tool to stop if the optional stop switch is set. The END statement causes the machine tool and controller to be shut down. The FINI statement is the last statement in a ANDR program; it must always be used.

SPINDL/n, UNITS, DIRECTION, CLUTCH RANGE NO.
SPINDL/O

The SPINDL statement is used to control direction and speed of the spindle. The spindle speed, n, may be specified either as revolutions per minute (units would be RPM) or surface feet per minute (units would be SFM). Units must always be specified. Dwells required are automatically inserted by the computer.

Spindle direction can be either clockwise (CW) or counter clockwise (CCW). If the direction is not specified counter clockwise is assumed.

The clutch must be specified as either high or low range, and the speed range number must be given.

The SPINDL/O statement (numeric zero) turns the spindle off.

FEDRAT/n, IPM, HIGH
FEDRAT/n, IPM, LOW
FEDRAT/n, IPR, HIGH
FEDRAT/n, IPR, LOW
RAPID

For the FEDRAT statement the number, n, is the federate desired. If neither high nor low range is specified low range is assumed. If neither inches per minute nor inches per revolution are specified, inches per minute is assumed. Thus, the statement

FEDRAT/5

will cause f feedrate of 5 inches per minute, and the low clutch range will be used.

The RAPID statement cause the cutting tool to move in rapid traverse. This statement remains in effect until a FEDRAT statement is encountered.

TURRET/n, m, deltaX, deltaY

The TURRET statement is used to define the position of the cutting tool relative to the turret center. The number "n" is the number given to the turret position being specified. The number "m" is the offset number which corresponds to a particular offset dial on the machine-tool console. DeltaX is the distance measured paralleled to the X-axis from the turret center to the tool center; a positive number indicates that the tool is to the left of the turret center. DeltaY is the distance measured parallel to the Y-axis from the center of the turret to the center of the tool; a positive number indicates the tool is ahead of the tool center.

CUTTER/rad

The CUTTER statement is used to specify the cutter radius. It remains in effect until another cutter statement is encountered.

MIRROR/X
MIRROR/Y
MIRROR/X, Y

The MIRROR statement causes the computer to reverse the axis or axes specified. For example, the statement MIRROR/Y will cause the computer to change the algebraic sign of all Y's specified.

INTOL/number
OUTTOL/number

For large radii and for machine tools that do not have circular interpolation, it is necessary to approximate the curve by a series of straight lines. The tolerance for the approximation is specified by the above statements. The number used with the INTOL statement becomes the maximum distance allwoed between the theoretical curve and the inside of the cutting edge of the tool. (Inside is defined as towards the tool center, and away from the defined curve). The OUTTOL statement is used to specify the maximum allowed deviation outside (away from the cutter center) the curve. The INTOL and OUTTOL statements should not be used unless required for they increase computer time required for processing. It is a good practice to reset the INTOL and OUTTOL to zero after they are used unless they are to remain in effect throughout the program.

ROUGH/n

The ROUGH statement is used to allow finishing stock allowance to be conveniently specified. The number after the slash mark is the stock allowed. The stock specified remains in effect until another ROUGH statement is encountered. Thus, the statement ROUGH/0 will normally occur before the final finishing cuts begin.

The resultant of a ROUGH statement is that the amount of roughing stock specified is added to the cutter radius to produce an altered cutter radius for all tool moves. The cutter will always advance to or retract from the surface

being cut at the beginning of each cut statement, giving the specified stock allowance. Care should be taken to insure that the sum of the roughing stock plus the cutter radius is less than the radius of any inside radius being cut.

GEOMETRIC STATEMENTS

The ANDRA language allows points, lines and circles to be defined. Each entity defined must be given a unique name of six or fewer characters (if more than six are used only the first six are interpreted). The first character of each name must be alphabetic. The vocabulary word for point is POINT/orP/; for line it is LINE/ or L/; for circle it is CIRCLE/ or C/. Nesting definitions may occur to any depth in any statement. Each name must be defined before it is used in other definitions.

POINT/

Points may be defined by either specifying the coordinates of the Point (the "X" coordinate and then the "Y" coordinate) or by specifying two intersecting lines. Examples of points follow.

```
PL=POINT/3, 4
PNT47=P/.2.53125, -4.875
K47SA=POINT/L7, L4
L44=POINT/2.5, 4.5
C17=P/ LIN5, LIN6
P14=P/ (LINE/P4, P6), L4
```

LINE/

Lines may be defined either by: 1), specifying a point through which the line passes and the angle of the line (Measured in degrees) relative to the "X" axis or, 2), or by specifying two points which lie on the line. Lines which are specified always have a direction associated with them. If the point-slope definition is used the direction is from the point outward along the line at the angle specified. If the two-point definition is used the direction is from the first point toward the second point. Examples of lines follow.

```
L5=LINE/3, 4, 90
LK427=L/P5, P7
A42L6=LINE/4.275,6.750,-90
C4=L/ -LK427
YAXIS=LINE/0, 0, 90
XAXIS=L/0, 0, 0
```

CIRCLE/

Circles may be defined 1), by specifying two lines tangent to the circle and the radius of the circle, or 2), by specifying the center location and radius of the circle. Each circle has a direction associated with it; clockwise or counter-clockwise. If the two-line definition is used the direction of the circle is from the tail of the first line twoards the head of the second line. If the center location and radius definition is used the direction should be specified clockwise (CW) or counter-clockwise (CCW). If the direction is not specified counter-clockwise is assumed if the radius specified is positive and assumed clockwise if the radius specified is negative.

If the two-line definition is used the lines must be specified so that the arc between the lines is less than 180

degrees. In other words, the circle must be specified along the arc nearest the vertex of the intersecting lines. Examples of circles follow in Fig. 8.

TOOL-PATH STATEMENTS

The following statements are used to direct the cutting tool to follos a particular path. The drive surface is the surface being machined. The check surface is the surface at which said cut ends.

When a CUT statement is encountered, and if the cutter is not located on the surface to be cut, the tool will be positioned on the drive surface before the cutting motion specified begins. The cutter will move along the shortest straight line path to the drive surface specified, unless the CUT statement is immediately prededed by an INDIRP statement.

When cutting a configuration of lines and circles where more than one intersection is possible, it is assumed that the first intersection encountered is intended unless the CUT statement has a comma and the number 2 following the check surface.

FROM/point

The FROM statement must be the first motion statement in a program. It identifies the starting location of the cutter center. The point may be specified using a symbol name for a previously defined point or by a nested definition. Examples of FROM statements follow:

```
FROM/ PT1
FROM/ (POINT/1, 15)
FROM/ (P/.7, -5)
```

INDIRP/ point

The INDIRP statement is used to specify the direction for the following motion statement. The check surface of the motion statement following must be in the path specified or a diagnostic error will be printed. Examples of INDIRP statements follow.

```
INDIRP/ P7
INDIRP/ (P/.7, 11)
```

GO/TO, name
GO/ON, name
GO/PAST, name

The GO statement is used to position the cutter prior to the actual cutting operation. The name may correspond to a point, line or circle. Unless an INDIRP statement is used prior to the GO statement the path selected will be the shortest, straight line move to the surface. If neither TO, ON, or PAST is specified the instruction TO is assumed. The rate of movement is controlled by the FEDRAT statement. Examples of GO statement follow.

```
GO/TO, L5
GO/PAST, CIR7
```

CUT/name1, TO, name2
CUT/name1, ON, name2
CUT/name1, PAST, name2

The symbol name1 is the line or circle that is to be machined (termed the drive surface); name2 is the surface at which the cut is to end (termed the check surface). If neither TO, ON or PAST is specified TO, is assumed. If the cutting tool cannot reach name2 by traveling along name1 an error message

is printed. The rate of cutting is determined by the FEDRAT and SPINDL statements. When cutting a circle, the cutter will follow the circle in the direction in which the circle has been defined. If the other direction is desired the drive circle should be preceded by a negative sign. Examples of CUT statements follow.

```
CUT/L1, TO, L5
CUT/ (LINE/ (POINT/5,2), 90) PAST, C4
CUT/CIRC7, ON, LN16, 2
CUT/-CIR7, PAST, CIR9,2
CUT/LN7, ON, LN5
```

Examples of blending lines and radii follow in Fig. 9.

GODLTA/x motion, y motion

The cutting tool will move in a straight line path to the new location specified. The rate of movement will be at the value specified in the last defined FEDRAT or RAPID statement. The algebraic sign of the incremental movements corresponds with conventional mathematical notation. Examples of the GODLTA statement follow.

```
GODLTA/0.5, 0.75
GODLTA/-2.5,-4.75
```

```
          ON
THREAD/A, TO  B,  C,  D,  E,  F,  G,  H,  J
          PAST
```

A is the name of the surface to be threaded; B is the name of the surface where the threads end; C is the lead; D is the infeed angle, which must be greater than 180 degrees for internal threads and less than 180 degrees for external threads; E is the depth of cut per pass; F is the normal depth of the thread; G is the cutter clearance during return movements, H is the number of pitches per lead; J is the number of finish

passes at zero infeed. The THREAD statement for a 0.125 pitch thread follows;

```
                                                                  Number Of Passes
                                                                  At Zero Infeed

THREAD/LIN1, TO, LIN2, 0.125, 118, 0.010, 0.7668, 0.005, 1, 2
Major Diameter          Lead        Depth of        Normal
                                    Cut Per         Clearance
                                    Pass

      End Of Threads          Infeed        Normal Depth      Number
                              Angle         of Thread         Of Threads
```

The following table lists the error messages that will be printed if the procedures outlined is not followed.

ERROR NUMBER	CAUSE
1	Illegal Major Word
2	Illegal Character in Floating Point Number Field
3	Illogical Motion Statement
4	Parenthesis Nest Error
5	Illegal Definition of Point, Line or Circle
6	Undefined Name of Point, Line or Circle
7	Illegal Statement
8	Too Many Points, Lines and Circles
9	Illegal Thread Statement
10	Illegal Postprocessor Call
11	Circular Interpolation Error
12	Duplicate Name For Point, Line or Circle

An example of how ANDR would be used to program a typical lathe part is shown in Figures 6 and 7. Fig. 6 shows the work piece, and Fig. 7 presents the computer input and output.

4. CONCLUDING COMMENTS

This paper has shown what one organization has done utilizing a small computer for numerical control programming. The author beleives that the benefits derived are sufficient to warrant additional activity in this area. Perhaps the most significant single benefit is the plotter output available. The opportunity ot verify accuracy of input statements greatly improves the accuracy of punched taped that reach the machine-shop floor. The subsequent improvement in efficiency of the N/C machinery justifies full involvement in preparing punched tapes in the manner described.

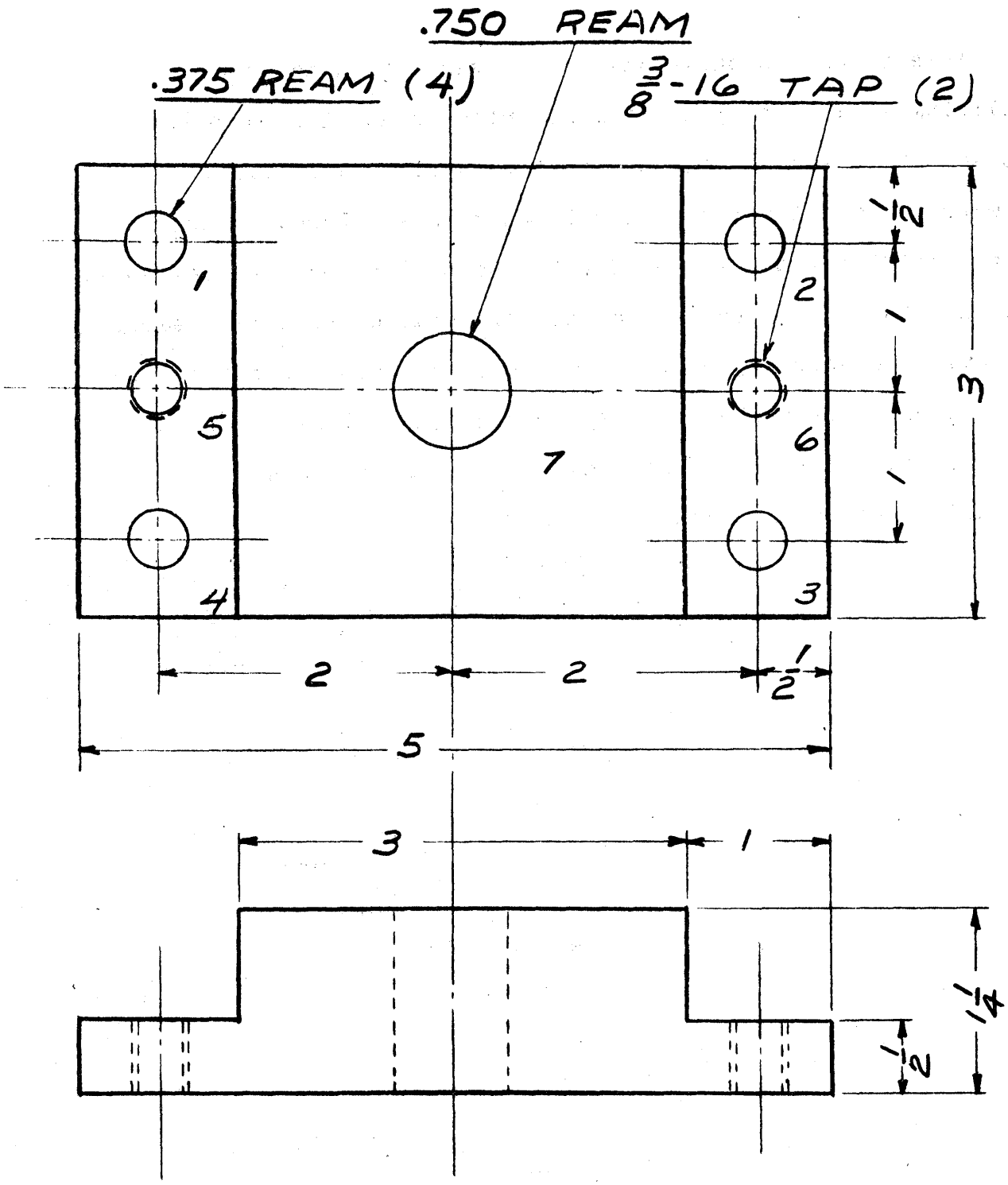


FIGURE 1

342

ANDERSON BROTHERS MFG. CO. ROCKFORD, ILL.
POINT-TO-POINT PROCESSOR LISTING

```
 1 [illegible]T NO/ SAMPLE 1A
 2 [illegible]ADER/20
 3 MACHIN/CINT
 4 MACHIN/PLOT
 5 COOLNT/ON
 6 CLPRINT
 7 SPINDL/CLW
 8 START  X15.  Y8.
 9 END  X15.  Y8.
10 P1  X0.5  Y0.5  R0.75
11 P2  X4.5
12 P3  Y2.5
13 P4  X0.5
14 P5  Y1.5
15 P6  X4.5
16 P7  X2.5  R0.0
17 CDRILL  P1-P7  DP0.2  DIA0.125  CS40  TL1
18 DRILL  P1-P4  DPT0.5  DIA0.360  TL2
19 REAM  P1-P4  DP0.6  DIA0.375  CS30  TL3
20 DRILL  P5-P6  DPT0.5  DIA0.3125  CS40  TL4
21 TAP  P6, P5  TAP 0.375  16  DP0.7  TL5
22 DRILL  P7  DPT1.25  DIA0.735  TL6
23 REAM  P7  DP1.4  DIA0.750  CS30  TL7
24 FINI
```

FIGURE 2

PART NO/ SAMPLE 1A

CARD	POINT	X-AXIS	Y-AXIS	R-AXIS	OPER.	DEPTH	CUT-SPD	TL-DIAM	TL-NO	MISC	TAP-DIAM	THD/IN
1	1	15.500	8.500	0.750	DRILL	0.200	40	0.125	1	13		
1	2	19.500	8.500	0.750	DRILL	0.200	40	0.125	1	13		
1	3	19.500	10.500	0.750	DRILL	0.200	40	0.125	1	13		
1	4	15.500	10.500	0.750	DRILL	0.200	40	0.125	1	13		
1	5	15.500	9.500	0.750	DRILL	0.200	40	0.125	1	13		
1	6	19.500	9.500	0.750	DRILL	0.200	40	0.125	1	13		
1	7	17.500	9.500	0.000	DRILL	0.200	40	0.125	1	13		
2	1	15.500	8.500	0.750	DRILL	0.500	40	0.360	2	13		
2	2	19.500	8.500	0.750	DRILL	0.500	40	0.360	2	13		
2	3	19.500	10.500	0.750	DRILL	0.500	40	0.360	2	13		
2	4	15.500	10.500	0.750	DRILL	0.500	40	0.360	2	13		
3	1	15.500	8.500	0.750	REAM	0.600	30	0.375	3	13		
3	2	19.500	8.500	0.750	REAM	0.600	30	0.375	3	13		
3	3	19.500	10.500	0.750	REAM	0.600	30	0.375	3	13		
3	4	15.500	10.500	0.750	REAM	0.600	30	0.375	3	13		
4	5	15.500	9.500	0.750	DRILL	0.500	40	0.312	4	13		
4	6	19.500	9.500	0.750	DRILL	0.500	40	0.312	4	13		
5	6	19.500	9.500	0.750	TAP	0.700			5	13	0.375	16
5	5	15.500	9.500	0.750	TAP	0.700			5	13	0.375	16
6	7	17.500	9.500	0.000	DRILL	1.250	40	0.735	6	13		
7	7	17.500	9.500	0.000	REAM	1.400	30	0.750	7	13		

FIGURE 2 · CONT

THE CINTIMATIC IS TO BE USED

THE FOLLOWING IS A LISTING OF THE PAPER TAPE

STARTING POSITIONS X = 15.000 Y = 8.000

ENDING POSITIONS X = 15.000 Y = 8.000

H 1	G81	X15.500	Y 8.500	Z0.238	F410	R0.750	S711	T 1	M13
H 2	G80	X15.500	Y 8.500	Z0.238	F410	R0.000	S711	T 1	M13
H 3	G81	X19.500	Y 8.500	Z0.238	F410	R0.750	S711	T 1	M13
H 4	G80	X19.500	Y 8.500	Z0.238	F410	R0.000	S711	T 1	M13
H 5	G81	X19.500	Y10.500	Z0.238	F410	R0.750	S711	T 1	M13
H 6	G80	X19.500	Y10.500	Z0.238	F410	R0.000	S711	T 1	M13
H 7	G81	X15.500	Y10.500	Z0.238	F410	R0.750	S711	T 1	M13
H 8	G80	X15.500	Y10.500	Z0.238	F410	R0.000	S711	T 1	M13
H 9	G81	X15.500	Y 9.500	Z0.238	F410	R0.750	S711	T 1	M13
H 10	G80	X15.500	Y 9.500	Z0.238	F410	R0.000	S711	T 1	M13
H 11	G81	X19.500	Y 9.500	Z0.238	F410	R0.750	S711	T 1	M13
H 12	G80	X19.500	Y 9.500	Z0.238	F410	R0.000	S711	T 1	M13
H 13	G81	X17.500	Y 9.500	Z0.238	F410	R0.000	S711	T 1	M 6
H 14	G81	X15.500	Y 8.500	Z0.708	F410	R0.750	S649	T 2	M13
H 15	G80	X15.500	Y 8.500	Z0.708	F410	R0.000	S649	T 2	M13
H 16	G81	X19.500	Y 8.500	Z0.708	F410	R0.750	S649	T 2	M13
H 17	G80	X19.500	Y 8.500	Z0.708	F410	R0.000	S649	T 2	M13
H 18	G81	X19.500	Y10.500	Z0.708	F410	R0.750	S649	T 2	M13
H 19	G80	X19.500	Y10.500	Z0.708	F410	R0.000	S649	T 2	M13
H 20	G81	X15.500	Y10.500	Z0.708	F410	R0.750	S649	T 2	M13
H 21	G80	X15.500	Y10.500	Z0.708	F410	R0.000	S649	T 2	M 6
H 22	G81	X15.500	Y 8.500	Z0.600	F420	R0.750	S615	T 3	M13
H 23	G80	X15.500	Y 8.500	Z0.600	F420	R0.000	S615	T 3	M13
H 24	G81	X19.500	Y 8.500	Z0.600	F420	R0.750	S615	T 3	M13
H 25	G80	X19.500	Y 8.500	Z0.600	F420	R0.000	S615	T 3	M13
H 26	G81	X19.500	Y10.500	Z0.600	F420	R0.750	S615	T 3	M13
H 27	G80	X19.500	Y10.500	Z0.600	F420	R0.000	S615	T 3	M13
H 28	G81	X15.500	Y10.500	Z0.600	F420	R0.750	S615	T 3	M13
H 29	G80	X15.500	Y10.500	Z0.600	F420	R0.000	S615	T 3	M 6
H 30	G81	X15.500	Y 9.500	Z0.694	F410	R0.750	S649	T 4	M13
H 31	G80	X15.500	Y 9.500	Z0.694	F410	R0.000	S649	T 4	M13
H 32	G81	X19.500	Y 9.500	Z0.694	F410	R0.750	S649	T 4	M13
H 33	G80	X19.500	Y 9.500	Z0.694	F410	R0.000	S649	T 4	M 6
H 34	G84	X19.500	Y 9.500	Z0.700	F471	R0.750	S610	T 5	M13
H 35	G80	X19.500	Y 9.500	Z0.700	F471	R0.000	S610	T 5	M13
H 36	G84	X15.500	Y 9.500	Z0.700	F471	R0.750	S610	T 5	M13
H 37	G80	X15.500	Y 9.500	Z0.700	F471	R0.000	S610	T 5	M 6
H 38	G81	X17.500	Y 9.500	Z1.570	F410	R0.000	S622	T 6	M13
H 39	G80	X17.500	Y 9.500	Z1.570	F410	R0.000	S622	T 6	M 6
H 40	G81	X17.500	Y 9.500	Z1.400	F410	R0.000	S570	T 7	M13

FIGURE 2. CONT

345

```
H 41    G80    X[illegible].500    Y 9.500    Z1.400    F410    R0.000    S570    T 7    M 6
H 42    G80    X[illegible].000    Y 8.000    Z1.400    F410    R0.000    S570    T 7    M 2
```

FIGURE 2. CONT

346

TOOL CHANGE TIME 3.249

TRAVERSE TIME 0.262

CUTTING TIME 10.249

TOTAL RUN TIME 13.762

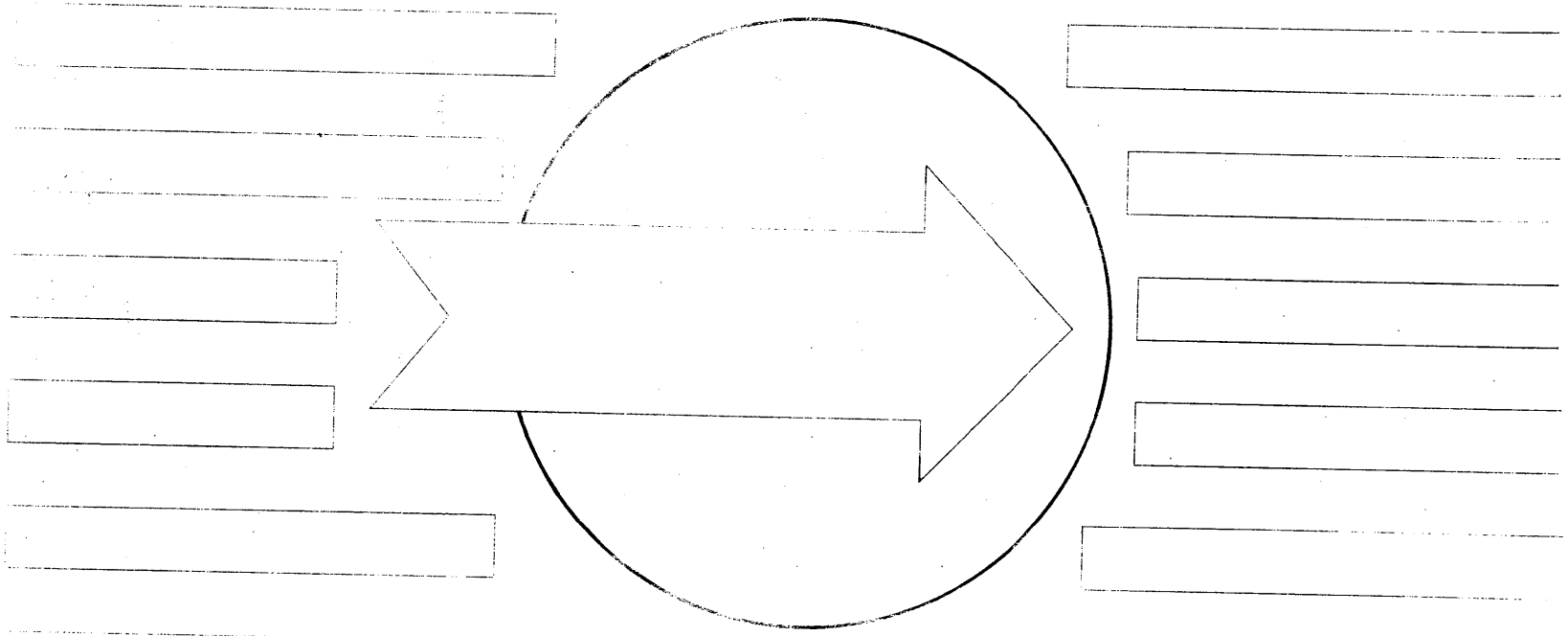
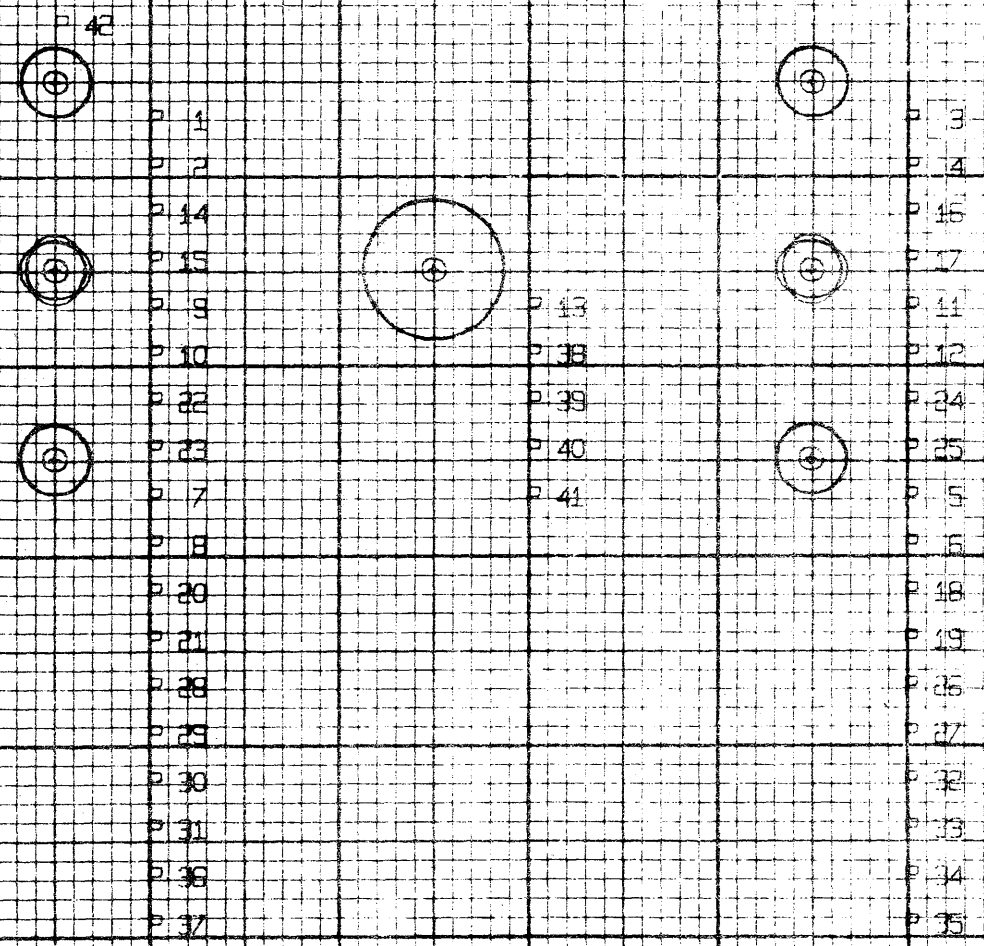


FIGURE 2. CONT.

PART NO. 1 SAMPLE 1A

LOCATE ORIGIN AT X = 15.000 Y = 8.000 SCALE 1.000



CINTIMATIC

FIGURE 2 - CONT.

ANDERSON BROTHERS MFG. CO.
POINT-TO-POINT PROCESSOR LISTING

ROCKFORD, ILL.

```
 1 PART NO/ SAMPLE 1B
 2 LEADER/20
 3 MACHIN/MILW
 4 MACHIN/ PLOT
 5 COOLNT/ON
 6 CLPRINT
 7 SPINDL/CLW
 8 START  X9.5  Y2.5
 9 TOOL 1  5.6
10 TOOL 2  3.9
11 TOOL 3  4.75
12 TOOL 4  4.625
13 TOOL 5  3.25
14 TOOL 6  5.25
15 TOOL 7  6.125
16 SIDE 1  10.000
17 P1  X0.5  Y2.5  R0.75  S1
18 P2  X4.5
19 P3  Y0.5
20 P4  X0.5
21 P5  Y1.5
22 P6  X4.5
23 P7  X2.5  R0.0
24 CDRILL  P1-P7  DP0.2  DIA0.2  S26  F4  TL1
25 DRILL  P1-P4  DPT0.5  DIA0.360  S22  F6  TL2
26 REAM  P1-P4  DP0.6  DIA0.375  S16  F12  TL3
27 DRILL  P5-P6  DPT0.5  DIA0.3125  S24  F8  TL4
28 TAP  P6, P5  TAP0.375  16  DP0.7  S1  F62  TL5
29 DRILL  P7  DPT1.25  DIA0.735  S14  F10  TL6
30 REAM  P7  DP1.4  DIA0.750  S5  F15  TL7
31 FINI
```

FIGURE 3

349

PART NO/ SAMPLE 1B

TOOL NO. 1 TOOL LENGTH 5.600
TOOL NO. 2 TOOL LENGTH 3.900
TOOL NO. 3 TOOL LENGTH 4.750
TOOL NO. 4 TOOL LENGTH 4.625
TOOL NO. 5 TOOL LENGTH 3.250
TOOL NO. 6 TOOL LENGTH 5.250
TOOL NO. 7 TOOL LENGTH 6.125
SIDE NO. 1 DISTANCE 10.000

CARD	POINT	X-AXIS	Y-AXIS	R-AXIS	OPER.	DEPTH	CUT-SPD	TL-DIAM	TL-NO	MISC	TAP-DIAM	THD/IN
1	1	10.000	5.000	0.750	DRILL	0.200	****	0.200	1	13		
1	2	14.000	5.000	0.750	DRILL	0.200	****	0.200	1	13		
1	3	14.000	3.000	0.750	DRILL	0.200	****	0.200	1	13		
1	4	10.000	3.000	0.750	DRILL	0.200	****	0.200	1	13		
1	5	10.000	4.000	0.750	DRILL	0.200	****	0.200	1	13		
1	6	14.000	4.000	0.750	DRILL	0.200	****	0.200	1	13		
1	7	12.000	4.000	0.000	DRILL	0.200	****	0.200	1	13		
2	1	10.000	5.000	0.750	DRILL	0.500	****	0.360	2	13		
2	2	14.000	5.000	0.750	DRILL	0.500	****	0.360	2	13		
2	3	14.000	3.000	0.750	DRILL	0.500	****	0.360	2	13		
2	4	10.000	3.000	0.750	DRILL	0.500	****	0.360	2	13		
3	1	10.000	5.000	0.750	REAM	0.600	****	0.375	3	13		
3	2	14.000	5.000	0.750	REAM	0.600	****	0.375	3	13		
3	3	14.000	3.000	0.750	REAM	0.600	****	0.375	3	13		
3	4	10.000	3.000	0.750	REAM	0.600	****	0.375	3	13		
4	5	10.000	4.000	0.750	DRILL	0.500	****	0.312	4	13		
4	6	14.000	4.000	0.750	DRILL	0.500	****	0.312	4	13		
5	6	14.000	4.000	0.750	TAP	0.700			5	13	0.375	16
5	5	10.000	4.000	0.750	TAP	0.700			5	13	0.375	16
6	7	12.000	4.000	0.000	DRILL	1.250	****	0.735	6	13		
7	7	12.000	4.000	0.000	REAM	1.400	****	0.750	7	13		

FIGURE 3. CONT.

THE MILWAUKEE-MATIC IS TO BE USED

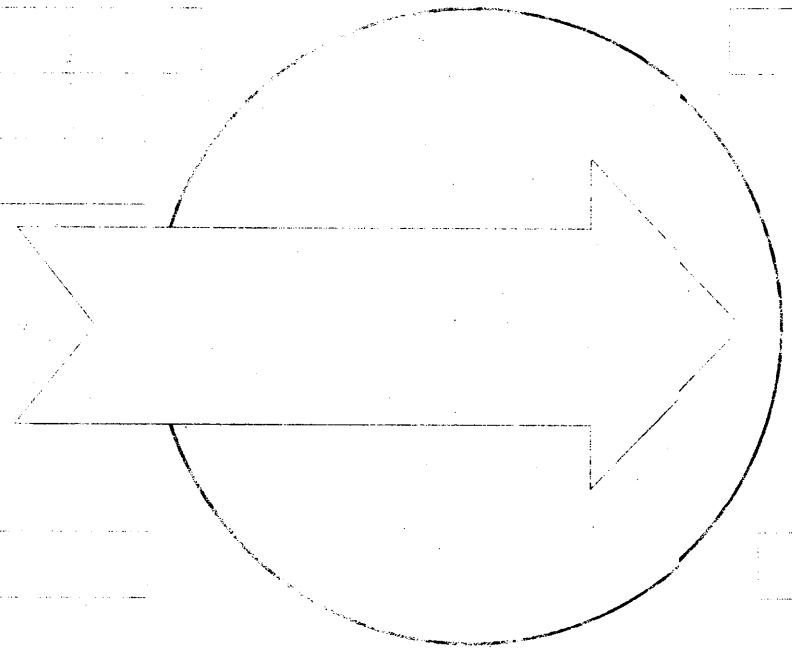


FIGURE 3. CONT.

OPERATION	SEQUENCE NUMBER	PREP. COMMAND	LONGITUD (X)	VERTICAL (Y)	DEPTH (Z)	ARC CENTER OFFSET	FEED RATE	TABLE INDEX	SPINDLE SPEED	FIRST TOOL	MISC. FUNCTION
REWIND STOP CODE											
SET X-Y-Z TO HOME	N 1	G 0	12.000	16.000	16.000		60				
LOCATE FIRST TOOL	N 2	G 0								T	
SPINDLE KEYLOCK	N 3	G 0							8		35
CHANGE TO TOOL 1	N 4	G 0									6
10 DELETE CODES											
INDEX TABLE	N 1	G 0						B			
POSITION X AND Y	N 2	G 0	10.000	5.000	16.000		98		26		8
POSITION Z AXIS	N 3	G 0	10.000	5.000	8.950		99		26		3
CUT TO DEPTH	N 4	G 0	10.000	5.000	8.590		4		26		
RETRACT TOOL	N 5	G 0	10.000	5.000	9.700		60		26		
POSITION X AND Y	N 6	G 0	14.000	5.000	9.700		99		26		
POSITION Z AXIS	N 7	G 0	14.000	5.000	8.950		60		26		
CUT TO DEPTH	N 8	G 0	14.000	5.000	8.590		4		26		
RETRACT TOOL	N 9	G 0	14.000	5.000	9.700		60		26		
POSITION X AND Y	N10	G 0	14.000	3.000	9.700		98		26		
POSITION Z AXIS	N11	G 0	14.000	3.000	8.950		60		26		
CUT TO DEPTH	N12	G 0	14.000	3.000	8.590		4		26		
RETRACT TOOL	N13	G 0	14.000	3.000	9.700		60		26		
POSITION X AND Y	N14	G 0	10.000	3.000	9.700		99		26		
POSITION Z AXIS	N15	G 0	10.000	3.000	8.950		60		26		
CUT TO DEPTH	N16	G 0	10.000	3.000	8.590		4		26		
RETRACT TOOL	N17	G 0	10.000	3.000	9.700		60		26		
POSITION X AND Y	N18	G 0	10.000	4.000	9.700		98		26		
POSITION Z AXIS	N19	G 0	10.000	4.000	8.950		60		26		
CUT TO DEPTH	N20	G 0	10.000	4.000	8.590		4		26		
RETRACT TOOL	N21	G 0	10.000	4.000	9.700		60		26		
POSITION X AND Y	N22	G 0	14.000	4.000	9.700		99		26		
POSITION Z AXIS	N23	G 0	14.000	4.000	8.950		60		26		
CUT TO DEPTH	N24	G 0	14.000	4.000	8.590		4		26		
RETRACT TOOL	N25	G 0	14.000	4.000	9.700		60		26		
POSITION X AND Y	N26	G 0	12.000	4.000	9.700		60		26		
CUT TO DEPTH	N27	G 0	12.000	4.000	9.340		4		26		
RETRACT TOOL	N28	G 0	12.000	4.000	9.700		60		26		
RETRACT Z TO HOME	N29	G 0	12.000	4.000	16.000		99		26		
MOVE Y TO HOME	N30	G 0	12.000	16.000	16.000		99		26		5
CHANGE TO TOOL 2	N31	G 0									6
10 DELETE CODES											
POSITION X AND Y	N 1	G 0	10.000	5.000	16.000		98		22		8
POSITION Z AXIS	N 2	G 0	10.000	5.000	7.250		99		22		3
CUT TO DEPTH	N 3	G 0	10.000	5.000	6.441		6		22		
RETRACT TOOL	N 4	G 0	10.000	5.000	8.000		60		22		
POSITION X AND Y	N 5	G 0	14.000	5.000	8.000		99		22		
POSITION Z AXIS	N 6	G 0	14.000	5.000	7.250		60		22		
CUT TO DEPTH	N 7	G 0	14.000	5.000	6.441		6		22		
RETRACT TOOL	N 8	G 0	14.000	5.000	8.000		60		22		

FIGURE 3. CONT.

352

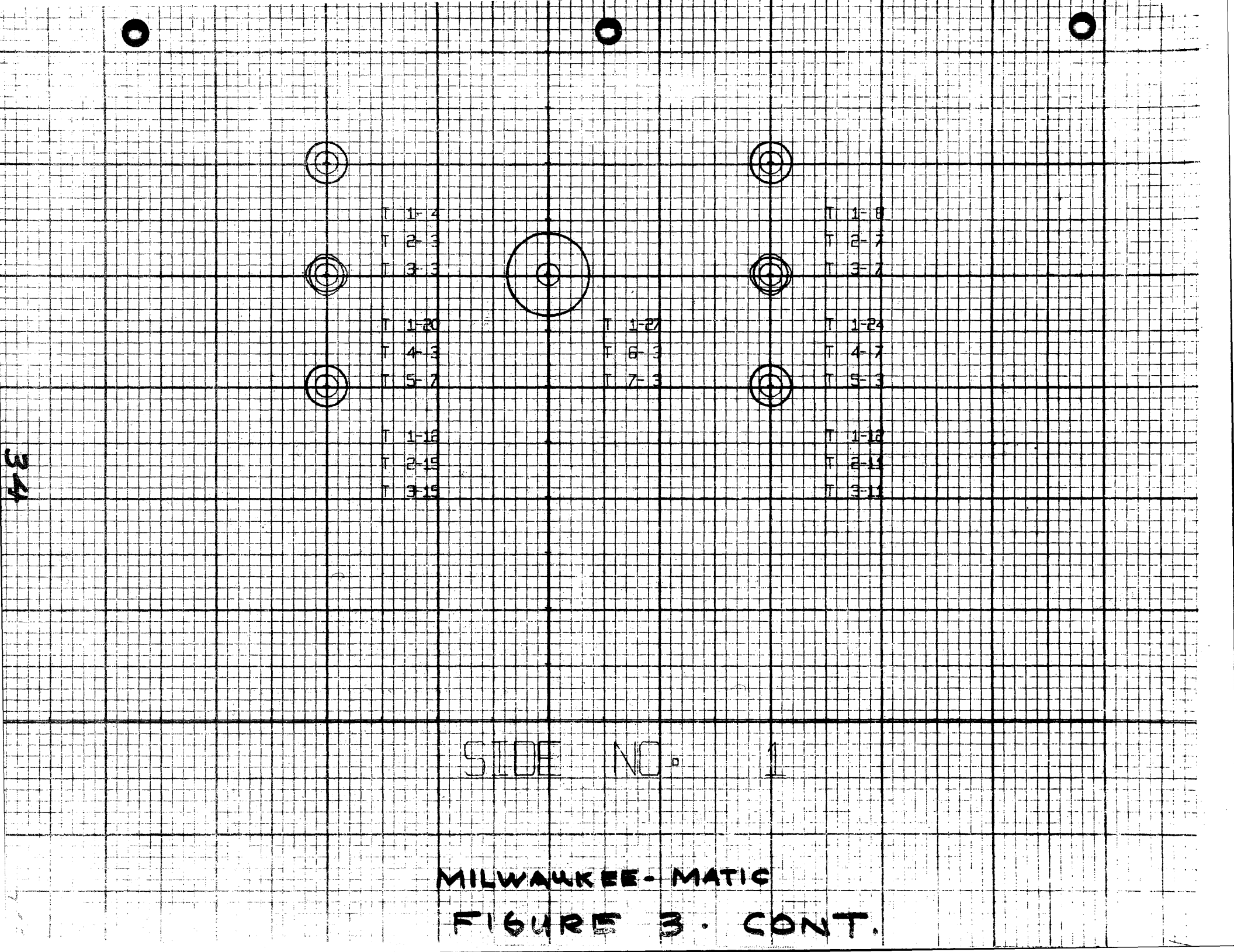
POSITION X AND Y	N 9	G 0	14.000	3.000	8.000		98	22	
POSITION Z AXIS	N10	G 0	14.000	3.000	7.250		60	22	
CUT TO DEPTH	N11	G 0	14.000	3.000	6.441		6	22	
RETRACT TOOL	N12	G 0	14.000	3.000	8.000		60	22	
POSITION X AND Y	N13	G 0	10.000	3.000	8.000		99	22	
POSITION Z AXIS	N14	G 0	10.000	3.000	7.250		60	22	
CUT TO DEPTH	N15	G 0	10.000	3.000	6.441		6	22	
RETRACT TOOL	N16	G 0	10.000	3.000	8.000		60	22	
RETRACT Z TO HOME	N17	G 0	10.000	3.000	16.000		99	22	
MOVE Y TO HOME	N18	G 0	10.000	16.000	16.000		99	22	5
CHANGE TO TOOL 3	N19	G 0							6
10 DELETE CODES									
POSITION X AND Y	N 1	G 0	10.000	5.000	16.000		99	16	8
POSITION Z AXIS	N 2	G 0	10.000	5.000	8.100		99	16	3
CUT TO DEPTH	N 3	G 0	10.000	5.000	7.399		12	16	
RETRACT TOOL	N 4	G 0	10.000	5.000	8.850		60	16	
POSITION X AND Y	N 5	G 0	14.000	5.000	8.850		99	16	
POSITION Z AXIS	N 6	G 0	14.000	5.000	8.100		60	16	
CUT TO DEPTH	N 7	G 0	14.000	5.000	7.399		12	16	
RETRACT TOOL	N 8	G 0	14.000	5.000	8.850		60	16	
POSITION X AND Y	N 9	G 0	14.000	3.000	8.850		98	16	
POSITION Z AXIS	N10	G 0	14.000	3.000	8.100		60	16	
CUT TO DEPTH	N11	G 0	14.000	3.000	7.399		12	16	
RETRACT TOOL	N12	G 0	14.000	3.000	8.850		60	16	
POSITION X AND Y	N13	G 0	10.000	3.000	8.850		99	16	
POSITION Z AXIS	N14	G 0	10.000	3.000	8.100		60	16	
CUT TO DEPTH	N15	G 0	10.000	3.000	7.399		12	16	
RETRACT TOOL	N16	G 0	10.000	3.000	8.850		60	16	
RETRACT Z TO HOME	N17	G 0	10.000	3.000	16.000		99	16	
MOVE Y TO HOME	N18	G 0	10.000	16.000	16.000		99	16	5
CHANGE TO TOOL 4	N19	G 0							6
10 DELETE CODES									
POSITION X AND Y	N 1	G 0	10.000	4.000	16.000		99	24	8
POSITION Z AXIS	N 2	G 0	10.000	4.000	7.975		99	24	3
CUT TO DEPTH	N 3	G 0	10.000	4.000	7.181		8	24	
RETRACT TOOL	N 4	G 0	10.000	4.000	8.725		60	24	
POSITION X AND Y	N 5	G 0	14.000	4.000	8.725		99	24	
POSITION Z AXIS	N 6	G 0	14.000	4.000	7.975		60	24	
CUT TO DEPTH	N 7	G 0	14.000	4.000	7.181		8	24	
RETRACT TOOL	N 8	G 0	14.000	4.000	8.725		60	24	
RETRACT Z TO HOME	N 9	G 0	14.000	4.000	16.000		99	24	
MOVE Y TO HOME	N10	G 0	14.000	16.000	16.000		99	24	5
CHANGE TO TOOL 5	N11	G 0							6
10 DELETE CODES									
POSITION X AND Y	N 1	G 0	14.000	4.000	16.000		99	1	8
POSITION Z AXIS	N 2	G 0	14.000	4.000	6.600		99	1	3
TAP TO DEPTH	N 3	G 0	14.000	4.000	5.800		62	1	5 52 51
RETRACT TOOL	N 4	G 0	14.000	4.000	7.350		62	1	4 5
POSITION X AND Y	N 5	G 0	10.000	4.000	7.350		99	1	8

FIGURE 3. CONT.

353

POSITION Z AXIS	N 6	G 0	10.000	4.000	6.600	60	1	3	
TAP TO DEPTH	N 7	G 0	10.000	4.000	5.800	62	1	3	52 51
RETRACT TOOL	N 8	G 0	10.000	4.000	7.350	62	1	4	3
RETRACT Z TO HOME	N 9	G 0	10.000	4.000	16.000	99	1		
MOVE Y TO HOME	N10	G 0	10.000	16.000	16.000	99	1	5	
CHANGE TO TOOL 6	N11	G 0						6	
10 DELETE CODES									
POSITION X AND Y	N 1	G 0	12.000	4.000	16.000	98	14	8	
POSITION Z AXIS	N 2	G 0	12.000	4.000	9.350	99	14	3	
CUT TO DEPTH	N 3	G 0	12.000	4.000	7.679	10	14		
RETRACT TOOL	N 4	G 0	12.000	4.000	9.350	60	14		
RETRACT Z TO HOME	N 6	G 0	12.000	4.000	16.000	99	14		
MOVE Y TO HOME	N 7	G 0	12.000	16.000	16.000	99	14	5	
CHANGE TO TOOL 7	N 8	G 0						6	
10 DELETE CODES									
POSITION X AND Y	N 1	G 0	12.000	4.000	16.000	99	5	8	
POSITION Z AXIS	N 2	G 0	12.000	4.000	10.225	99	5	3	
CUT TO DEPTH	N 3	G 0	12.000	4.000	8.725	15	5		
RETRACT TOOL	N 4	G 0	12.000	4.000	10.225	60	5		
RETRACT Z TO HOME	N 6	G 0	12.000	4.000	16.000	99	5		
MOVE Y TO HOME	N 7	G 0	12.000	16.000	16.000	99	5	5	
CHANGE TO TOOL 8	N 8	G 0						6	
10 DELETE CODES									
SET X-Y-Z TO HOME	N 1	G 0	12.000	16.000	16.000	60			
INDEX TABLE	N 2	G 0					8		
INDEX TABLE	N 3	G 0					8		
INDEX TABLE	N 4	G 0					8		
END OF TAPE	N 5							30	

FIGURE 3. CONT.



MILWAUKEE-MATIC

FIGURE 3. CONT.

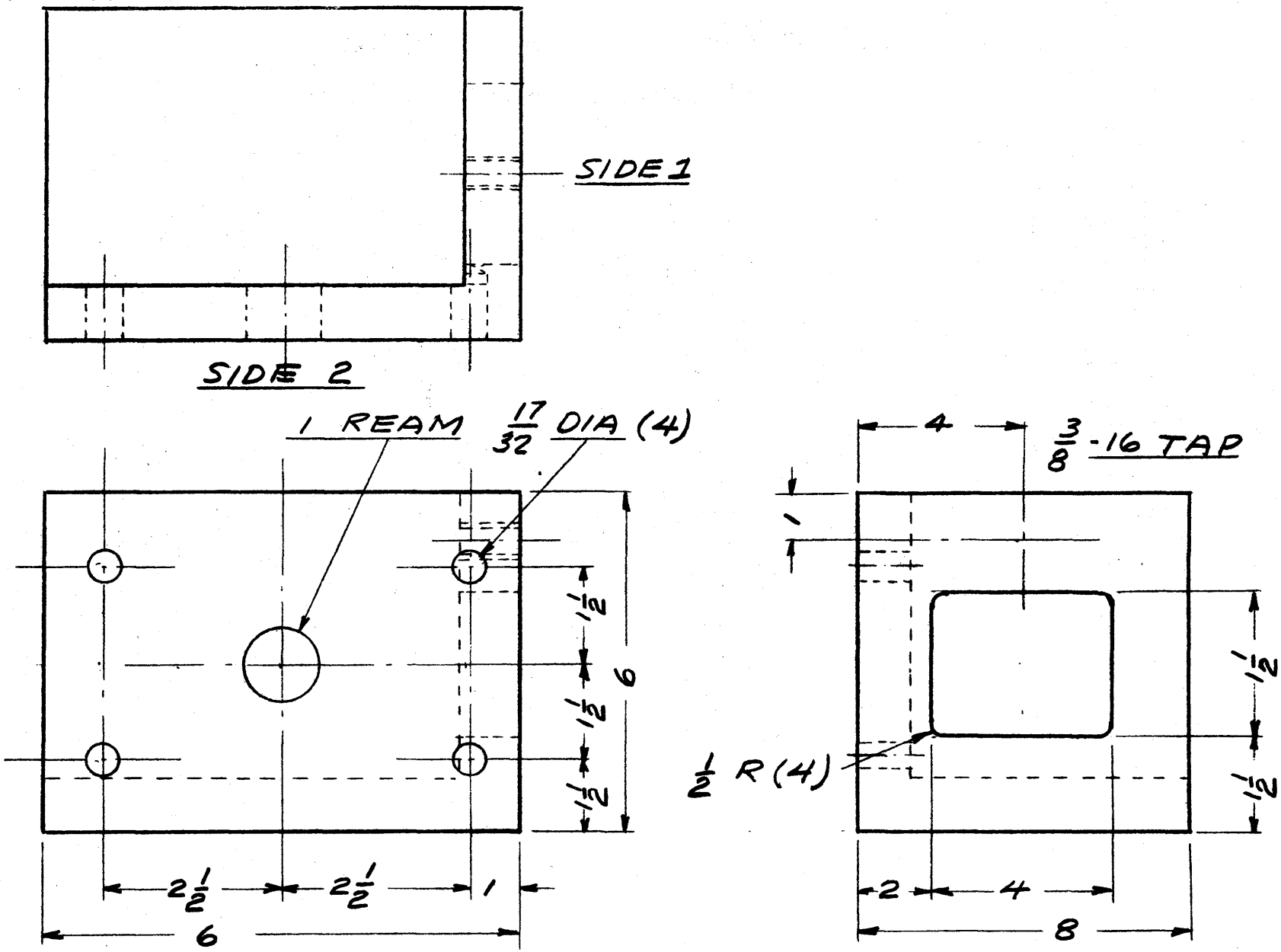

SIDE 1
SIDE 2
1 REAM
17/32 DIA (4)
1 1/2
1 1/2
1 1/2
6
2 1/2
2 1/2
1
6
4
3/8 -16 TAP
1
1 1/2
1 1/2
1/2 R (4)
2
4
8

FIGURE 4

ANDERSON BROTHERS MFG. CO.
POINT-TO-POINT PROCESSOR LISTING

ROCKFORD, ILL.

```
 1 PART NO/ SAMPLE NO. 2
 2 MACHIN/MILW
 3 CLPRINT
 4 COOLNT/ON
 5 SPINDL/CLW
 6 LEADER/20
 7 TOOL 1  4.75
 8 TOOL 2  5.625
 9 TOOL  3  5.25
10 TOOL 4  7.5
11 TOOL 5  6.00
12 TOOL  6  4.25
13 TOOL 7  6.25
14 SIDE 1  3.000
15 SIDE 2   4.000
16 START  X12.  Y5.
17 P1  X-2.5  Y1.5  S2
18 P2  X2.5
19 P3  Y-1.5
20 P4  X-2.5
21 P5  X0.  Y0.
22 P6  X0.  Y2.   S 1
23 P10  X1.5  Y0.375  S1
24 P11  X1.625  Y0.25
25 P12  X1.5  Y0.25
26 P13  X1.625  Y-0.25
27 P14  X1.5  Y-0.375
28 P15  X1.5  Y-0.25
29 P16  X-1.5  Y-0.375
30 P17  X-1.625  Y-0.25
31 P18  X-1.5  Y-0.25
32 P19  X-1.625  Y0.25
33 P20  X-1.5  Y0.375
34 P21  X-1.5  Y0.25
35 CDRILL  P1-P6  F3  S22  DP0.2  DIA0.125  TL1
36 DRILL  P1-P5  F8  S16  DPT1.0  DIA0.531  TL2
37 DRILL  P5  F6  S5  DPT1.0  DIA.968  TL3
38 REAM  P5  F12  S5  DP1.3  DIA1.000  TL4
39 DRILL  P6  F8  S23  DPT1.0  DIA0.3125  TL5
40 TAP  P6  TAP.375  16  S1  F62  TL6
41 MILL  P12-P10  F4  S10  DP1.25  DIA0.75  TL7
42 CONTR P10 & P11, P12  F4  S10
43 MILL  P11-P13  F4  S10
44 CONTR  P13 & P14, P15  F4  S10
45 MILL  P14-P16  F4  S10
46 CONTR  P16 & P17, P18  F4  S10
47 MILL  P17-P19  F4  S10
48 CONTR  P19 &P20, P21  F4 S10
49 MILL P20-P10  F 4  S 10
50 MILL P10-P12  F 4  S 10
```

FIGURE 5

```
51 FINI
```

FIGURE 5. CONT.

PART NO/ SAMPLE NO. 2

TOOL NO.	1	TOOL LENGTH	4.750
TOOL NO.	2	TOOL LENGTH	5.625
TOOL NO.	3	TOOL LENGTH	5.250
TOOL NO.	4	TOOL LENGTH	7.500
TOOL NO.	5	TOOL LENGTH	6.000
TOOL NO.	6	TOOL LENGTH	4.250
TOOL NO.	7	TOOL LENGTH	6.250
SIDE NO.	1	DISTANCE	3.000
SIDE NO.	2	DISTANCE	4.000

CARD	POINT	X-AXIS	Y-AXIS	R-AXIS	OPER.	DEPTH	CUT-SPD	TL-DIAM	TL-NO	MISC	TAP-DIAM	THD/IN
1	1	9.500	6.500	0.000	DRILL	0.200	****	0.125	1	13		
1	2	14.500	6.500	0.000	DRILL	0.200	****	0.125	1	13		
1	3	14.500	3.500	0.000	DRILL	0.200	****	0.125	1	13		
1	4	9.500	3.500	0.000	DRILL	0.200	****	0.125	1	13		
1	5	12.000	5.000	0.000	DRILL	0.200	****	0.125	1	13		
1	6	12.000	7.000	0.000	DRILL	0.200	****	0.125	1	13		
2	1	9.500	6.500	0.000	DRILL	1.000	****	0.531	2	13		
2	2	14.500	6.500	0.000	DRILL	1.000	****	0.531	2	13		
2	3	14.500	3.500	0.000	DRILL	1.000	****	0.531	2	13		
2	4	9.500	3.500	0.000	DRILL	1.000	****	0.531	2	13		
2	5	12.000	5.000	0.000	DRILL	1.000	****	0.531	2	13		
3	5	12.000	5.000	0.000	DRILL	1.000	****	0.968	3	13		
4	5	12.000	5.000	0.000	REAM	1.300	****	1.000	4	13		
5	6	12.000	7.000	0.000	DRILL	1.000	****	0.312	5	13		
6	6	12.000	7.000	0.000	TAP	1.000			6	13	0.375	16
7	12	13.500	5.250	0.000	MILL	1.250	****	0.750	7	13		
8	10	13.500	5.375	0.000	MILL	1.250	****	0.750	7	13		
9	11	13.625	5.250	0.000	MILL	1.250	****	0.750	7	13		
10	13	13.625	4.750	0.000	MILL	1.250	****	0.750	7	13		
11	14	13.500	4.625	0.000	MILL	1.250	****	0.750	7	13		
12	16	10.500	4.625	0.000	MILL	1.250	****	0.750	7	13		
13	17	10.375	4.750	0.000	MILL	1.250	****	0.750	7	13		
14	19	10.375	5.250	0.000	MILL	1.250	****	0.750	7	13		
15	20	10.500	5.375	0.000	MILL	1.250	****	0.750	7	13		
16	10	13.500	5.375	0.000	MILL	1.250	****	0.750	7	13		

FIGURE 5·CONT.

OPERATION	SEQUENCE NUMBER	PREP. COMMAND	LONGITUD (X)	VERTICAL (Y)	DEPTH (Z)	ARC CENTER OFFSET	FEED RATE	TABLE INDEX	SPINDLE SPEED	FIRST TOOL	MISC. FUNCTION
REWIND STOP CODE											
SET X-Y-Z TO HOME	N 1	G 0	12.000	16.000	16.000		60				
LOCATE FIRST TOOL	N 2	G 0								T	
SPINDLE KEYLOCK	N 3	G 0							8		35
CHANGE TO TOOL 1	N 4	G 0									6
10 DELETE CODES											
INDEX TABLE	N 1	G 0						B			
INDEX TABLE	N 2	G 0						B			
POSITION X AND Y	N 3	G 0	9.500	6.500	16.000		98		22		8
POSITION Z AXIS	N 4	G 0	9.500	6.500	2.850		99		22		3
CUT TO DEPTH	N 5	G 0	9.500	6.500	2.512		3		22		
RETRACT TOOL	N 6	G 0	9.500	6.500	2.850		60		22		
POSITION X AND Y	N 7	G 0	14.500	6.500	2.850		99		22		
CUT TO DEPTH	N 8	G 0	14.500	6.500	2.512		3		22		
RETRACT TOOL	N 9	G 0	14.500	6.500	2.850		60		22		
POSITION X AND Y	N10	G 0	14.500	3.500	2.850		99		22		
CUT TO DEPTH	N11	G 0	14.500	3.500	2.512		3		22		
RETRACT TOOL	N12	G 0	14.500	3.500	2.850		60		22		
POSITION X AND Y	N13	G 0	9.500	3.500	2.850		99		22		
CUT TO DEPTH	N14	G 0	9.500	3.500	2.512		3		22		
RETRACT TOOL	N15	G 0	9.500	3.500	2.850		60		22		
POSITION X AND Y	N16	G 0	12.000	5.000	2.850		98		22		
CUT TO DEPTH	N17	G 0	12.000	5.000	2.512		3		22		
RETRACT TOOL	N18	G 0	12.000	5.000	2.850		60		22		
RETRACT Z TO HOME	N19	G 0	12.000	5.000	16.000		99		22		
INDEX TABLE	N20	G 0						B			
INDEX TABLE	N21	G 0						B			
INDEX TABLE	N22	G 0						B			
POSITION X AND Y	N23	G 0	12.000	7.000	16.000		98		22		
POSITION Z AXIS	N24	G 0	12.000	7.000	1.850		99		22		
CUT TO DEPTH	N25	G 0	12.000	7.000	1.512		3		22		
RETRACT TOOL	N26	G 0	12.000	7.000	1.850		60		22		
RETRACT Z TO HOME	N27	G 0	12.000	7.000	16.000		99		22		
MOVE Y TO HOME	N28	G 0	12.000	16.000	16.000		99		22		5
CHANGE TO TOOL 2	N29	G 0									6
10 DELETE CODES											
INDEX TABLE	N 1	G 0						B			
POSITION X AND Y	N 2	G 0	9.500	6.500	16.000		98		16		8
POSITION Z AXIS	N 3	G 0	9.500	6.500	3.725		99		16		3
CUT TO DEPTH	N 4	G 0	9.500	6.500	2.365		8		16		
RETRACT TOOL	N 5	G 0	9.500	6.500	3.725		60		16		
POSITION X AND Y	N 6	G 0	14.500	6.500	3.725		99		16		
CUT TO DEPTH	N 7	G 0	14.500	6.500	2.365		8		16		
RETRACT TOOL	N 8	G 0	14.500	6.500	3.725		60		16		
POSITION X AND Y	N 9	G 0	14.500	3.500	3.725		99		16		
CUT TO DEPTH	N10	G 0	14.500	3.500	2.365		8		16		

FIGURE 5- CONT.

RETRACT TOOL	N11	G 0	14.500	3.500	3.725		60		16	
POSITION X AND Y	N12	G 0	9.500	3.500	3.725		99		16	
CUT TO DEPTH	N13	G 0	9.500	3.500	2.365		8		16	
RETRACT TOOL	N14	G 0	9.500	3.500	3.725		60		16	
POSITION X AND Y	N15	G 0	12.000	5.000	3.725		98		16	
CUT TO DEPTH	N16	G 0	12.000	5.000	2.365		8		16	
RETRACT TOOL	N17	G 0	12.000	5.000	3.725		60		16	
RETRACT Z TO HOME	N18	G 0	12.000	5.000	16.000		99		16	
MOVE Y TO HOME	N19	G 0	12.000	16.000	16.000		99		16	5
CHANGE TO TOOL 3	N20	G 0								6
10 DELETE CODES										
POSITION X AND Y	N 1	G 0	12.000	5.000	16.000		99		5	8
POSITION Z AXIS	N 2	G 0	12.000	5.000	3.350		99		5	3
CUT TO DEPTH	N 3	G 0	12.000	5.000	1.859		6		5	
RETRACT TOOL	N 4	G 0	12.000	5.000	3.350		60		5	
RETRACT Z TO HOME	N 6	G 0	12.000	5.000	16.000		99		5	
MOVE Y TO HOME	N 7	G 0	12.000	16.000	16.000		99		5	5
CHANGE TO TOOL 4	N 8	G 0								6
10 DELETE CODES										
POSITION X AND Y	N 1	G 0	12.000	5.000	16.000		99		5	8
POSITION Z AXIS	N 2	G 0	12.000	5.000	5.600		99		5	3
CUT TO DEPTH	N 3	G 0	12.000	5.000	4.199		12		5	
RETRACT TOOL	N 4	G 0	12.000	5.000	5.600		60		5	
RETRACT Z TO HOME	N 6	G 0	12.000	5.000	16.000		99		5	
MOVE Y TO HOME	N 7	G 0	12.000	16.000	16.000		99		5	5
CHANGE TO TOOL 5	N 8	G 0								6
10 DELETE CODES										
INDEX TABLE	N 1	G 0				B				
INDEX TABLE	N 2	G 0				B				
INDEX TABLE	N 3	G 0				B				
POSITION X AND Y	N 4	G 0	12.000	7.000	16.000		99		23	
POSITION Z AXIS	N 5	G 0	12.000	7.000	3.100		99		23	
CUT TO DEPTH	N 6	G 0	12.000	7.000	2.375		8		23	
RETRACT TOOL	N 7	G 0	12.000	7.000	3.100		60		23	
RAPID TO DEPTH	N 8	G 0	12.000	7.000	2.475		60		23	
CUT TO DEPTH	N 9	G 0	12.000	7.000	1.806		8		23	
RETRACT TOOL	N10	G 0	12.000	7.000	3.100		60		23	
RETRACT Z TO HOME	N11	G 0	12.000	7.000	16.000		99		23	
MOVE Y TO HOME	N12	G 0	12.000	16.000	16.000		99		23	5
CHANGE TO TOOL 6	N13	G 0								6
10 DELETE CODES										
POSITION X AND Y	N 1	G 0	12.000	7.000	16.000		99		1	8
POSITION Z AXIS	N 2	G 0	12.000	7.000	1.350		99		1	3
TAP TO DEPTH	N 3	G 0	12.000	7.000	0.056		62		1	5 52 51
RETRACT TOOL	N 4	G 0	12.000	7.000	1.350		62		1	4 5
RETRACT Z TO HOME	N 6	G 0	12.000	7.000	16.000		99		1	
MOVE Y TO HOME	N 7	G 0	12.000	16.000	16.000		99		1	5

FIGURE 5 · CONT.

CHANGE TO TOOL 7	N 8	G 0									6
10 DELETE CODES											
POSITION X AND Y	N 1	G 0	13.500	5.250	16.000			98		10	8
POSITION Z AXIS	N 2	G 0	13.500	5.250	3.350			99		10	3
MILL TO DEPTH	N 3	G 0	13.500	5.250	2.000			4		10	
MILL ON X-Y PLAIN	N 4	G 0	13.500	5.375	2.000			4		10	
MILL CONTOUR	N 5	G 2	0.125	-0.125	2.000	0.000	0.125	320		10	
MILL ON X-Y PLAIN	N 6	G 0	13.625	4.750	2.000			4		10	
MILL CONTOUR	N 7	G 2	-0.125	-0.125	2.000	0.125	0.000	320		10	
MILL ON X-Y PLAIN	N 8	G 0	10.500	4.625	2.000			4		10	
MILL CONTOUR	N 9	G 2	-0.125	0.125	2.000	0.000	0.125	320		10	
MILL ON X-Y PLAIN	N10	G 0	10.375	5.250	2.000			4		10	
MILL CONTOUR	N11	G 2	0.125	0.125	2.000	0.125	0.000	320		10	
MILL ON X-Y PLAIN	N12	G 0	13.500	5.375	2.000			4		10	
MILL ON X-Y PLAIN	N13	G 0	13.500	5.250	2.000			4		10	
RETRACT TOOL	N14	G 0	13.500	5.250	3.350			60		10	
RETRACT Z TO HOME	N15	G 0	13.500	5.250	16.000			99		10	
MOVE Y TO HOME	N16	G 0	13.500	16.000	16.000			99		10	5
CHANGE TO TOOL 8	N17	G 0									6
10 DELETE CODES											
SET X-Y-Z TO HOME	N 1	G 0	12.000	16.000	16.000			60			
INDEX TABLE	N 2	G 0							B		
INDEX TABLE	N 3	G 0							B		
INDEX TABLE	N 4	G 0							B		
END OF TAPE	N 5										30

FIGURE 5. CONT.

T4A

JOB COST ACCOUNTING

and

STUDENT MONITOR SYSTEMS

by

Thomas R. Harbron

Director

Anderson College Computing Center

Anderson, Indiana 46011

COMMON Meeting

April 8 - 10, 1968

Chicago, Illinois

Job Cost Accounting

There are at least four reasons for job cost accounting in a computing center. First, it provides a record of the work that is actually done. Second, it greatly facilitates the invoicing of users both internal and external. Third, it simplifies the problem of supervising a job from the time it enters the computing center until it is done. Fourth, it is an aid in resource allocation. This includes inventory control of supplies, machine time, and programming and systems analysis time.

The basis of the job cost accounting system is the job order form (see figure 1). This is a preprinted three part form using NCR paper. When a job enters the computing center the top portion of the job order form is filled out including the date, the user, the due date, and a description of the job to be done. The third copy is then seperated and placed in a job pending file. The first two copies are given to the person responsible for the job.

As the job progressess through its various steps, the quanity of resources used is filled in. These resources include the time to the nearest hundredth of an hour for systems analyst, programmers, operators, machines, etc., and the quanities of supplies such as cards and paper forms.

When the job is completed the information at the bottom of the job order form is filled out. This includes the date, the person to whom the job was delivered, and the operator responsible for completion of the job.

The two copies of the job order form now completely filled out are returned for filing. The corresponding third copy is pulled from the job

pending file and discarded. The second copy is placed in a monthly file of jobs completed. The first copy is placed in the user's file.

At the month's end the monthly file, which contains the second copy of all job orders completed that month, is used to generate input data for the computer. Cards are manually punched and verified from this second copy. Each card contains the data from one line on a job order. This includes a resource identification number, the quanity of that resource, and rate and cost figures if these differ from the system standard. Each card also contains a user identification code. The cards are grouped by user and the users are grouped by category. We are currently using three categories which are education, administration, and commercial.

A program named "JOBTOT" is next executed. Input to this program includes a rate table which gives the standard costs and billing rate for each resource, and the line cards which were punched from the job orders. The output data consists of a job report for each user as shown in figure two. This report shows the total quanity of each resource, the direct cost of this to the computing center, the gross profit and the charges made to the user. Totals are also given for each user of the charges, gross profit and direct cost. A tally is also kept on the number of jobs processed for each user.

A similar table is also prepared showing the totals by category and finally grand totals for all jobs processed during the month.

Invoices are automatically prepared by a routine called "BILL". The input to BILL consists of two decks. The first deck is a table containing user identification code for each user and the corresponding name and address of the user. The second deck is the output data from

"JOBTOT". The program prepares an invoice for each user that appears in both the user address table and the "JOBTOT" report.

A typical invoice is shown in figure 3. Mutiple copies of the invoice are printed and these are sent to the user, to the business office, and one copy is placed in the user's file and attached to the job orders that it represents.

In addition to the job orientated system just described data is also kept in the form of a computer log. A sample page is shown in figure 4. The log is a complete record of all computer time used. The data recorded for each date includes the starting and finishing time of the job (by the computer clock), the operator, the job type, and a space is also provided for comments.

At the end of each month the data in the log is punched into cards the data carried into the cards includes the starting and finishing, time for each job, and the classification category of that job. These cards are then processed by a program called "LOGTOT". LOGTOT produces totals by each category and grand totals for all computer use during the month.

This information is useful for cross checking with the job report, as well as monitoring student use which is not reported on the job report.

Student Monitor System

For approximately the first two years after installing our 1620, the Anderson College Computing Center was operated on an open shop basis with respect to the students. For this purpose the 1620 Monitor I System with its "load and go" provisions seemed well suited.

It soon become obvious, however, that many students had fallen into a "trial and error" method of programming. This is bad for two reasons. First, the student does not learn effective programming methods. Second, it is wasteful of computer time.

The only alternatives appeared to be either a closed shop arrangement with extensive record keeping, or some automatic means of monitoring student activity. The latter course was chosen and the result is the Student Monitor System.

The objectives of the Student Monitor System were (1) keep a record of student work, (2) control student usage, and (3) simplify student operation.

To gain some perspective on the Student Monitor System it is useful to first look at the 1620 Monitor System (figure 5). In figure 5 the flow of control is shown by solid lines and the flow of data by broken lines.

Typically the sequence begins with the student loading a cold start card. The cold start card brings in the 1620 job analyzer routine. It is the function of this routine to read the monitor control records and provide the necessary linkage to the appropriate compiler and program loaders. Typically the Fortran compiler is called in and in turn reads the compiler control records and the student's source deck. If an error is detected during compilation control is returned to the 1620 monitor job record analyzer. If the compilation is successful the object program is loaded into core and

executed. The object program may at this point read input data. If the execution is not successful control is normally returns to the 1620 Monitor job record analyzer. If the execution is successful a normal exit is made from the program back to the job record analyzer.

A similar perspective for the Student Monitor System is shown in figure 6. Again the sequence is normally started with a cold start card. This time the student monitor is brought into the computer rather than the 1620 job record analyzer. The student monitor reads a single student control card. This card contains the following mandatory information: the course number and section, the student's social security number, the language in which the student's program is written, and a program identification number. Optional information that may be included in the control card includes: the subroutine set and LOCAL count, comments, and other identification.

The student monitor routine first checks the mandatory data against the student monitor file which is kept on disk. It first determines that the student (identified by his social security number) is enrolled in the course given. It then checks the program number to ascertain that it has been authorized by the instructor. This having been done the student monitor routine updates the student's record on the student monitor file to show that he has attempted the program listed. Two identification fields are also placed in the student monitor file pointing to the individual student and the program he is using.

The appropriate indicators within 1620 monitor system are set and the linkage to the compiler is implimented.

If an error is detected during compilation the compiler will still return to the 1620 monitor system. There seems no simple way to avoid this difficulty. If the compilation is successful the appropriate loader is called, the object program is loaded into core, and execution begins.

If the execution is successful, the program is terminated by the statement "CALL FINIS" in fortran, or the statement "CALL LINK SMEXIT" if the program is written in SPS. FINIS is a linkage routine to SMEXIT.

SMEXIT accesses the student monitor file and adds a tally of completes to the appropriate student's record as indicated by the pointers left by SM. SMEXIT then links back to the student monitor routine which is ready for the next student.

The student monitor file consists of one hundred sectors of disks storage and is divided into two parts. The first portion is a communication area which contains control data for the rest of the file. In the present system there is provision made for three courses. The communications sector includes the upper and lower limits in the file area for each course, and the programs that can be run for each course. In addition there is storage space for the pointers mentioned earlier.

The remaining 99 sectors of the file are used to store the records of up to 198 students. Each student's record consists of his social security number, the total number of attempts and the total number of completions since the last report. This record occupies approximately 50 digits of storage.

There are seven programs involved in the student monitor system. These will be discussed one at a time and it may be helpful to refer to figure 6 at points.

Student monitor (SM) is the primary routine in the system. Its functions have been previously described and briefly are, to read and check the student control card, update the student record, and link through the 1620 monitor system to the appropriate compiler.

SMEXIT has also been described earlier in function. Briefly it updates the student record after execution, types the message "Execution completed" and links back to student monitor.

FINIS is a short linkage routine which may be called by a Fortran program. This is made necessary by a restriction of the 1620 Monitor System that an SPS program may not be called from a Fortran program.

SMCS is a core image routine which includes the arithemetic tables, 1620 monitor supervisor, and a link to SM. It is used to initialize the system and bring in the student monitor. It is loaded by machine language instructions contained on the student cold start card.

SMRPT is used to generate a student monitor report (see figure 8). This report includes the social security number, name, total attempts and total completions for each of 9 programs, and total attempts and total completions for each student since the last report. The student's name is obtained from the registrar's files located on another disk pack. This routine resets the cummalative totals to zero after each report.

SMDFIN is used by the instructor to tell the system which program or programs the students are permitted to work on at any given time.

SMLOAD is used to initially load the student monitor file. One control card is read in along with the class cards for each class. Each instructor has class cards which contains among other information the student's social security number.

The student monitor system has been reasonable successful in the semester and a half that it has been used. This is indicated by the fact that the per student computer time is running about half of the corresponding figure a year ago. The students are also producing better programs in less time.

As yet no control facilities have been incorparated into the system.

It would, however, be a simple matter to modify SM so that the number of attempted executions would be limited to some arbitrary figure. Fifteen would seem to be a reasonable upper limit on the number of attempts a student should make on a given program.

ANDERSON COLLEGE COMPUTING CENTER

JOB ORDER

DATE 4-8-68 FOR Registrar DATE DUE 4-10-68

JOB DESCRIPTION

Generate master cards for summer registration. (R131A)

RESOURCES REQUIRED		ESTIMATED	ACTUAL	OPERATOR	DATE	COST
DESIGN						
SYSTEMS	1					
LIAISON	2					
PROGRAMMING	3					
BOARD WIRING	4					
PROCESSING						
OPERATOR CLASS 1	5		1.00	ML		
OPERATOR CLASS 2	6					
1620 SYSTEM	7		.17	ML		
026	8					
056	9					
TAPE-TO-CARD	10					
082	11					
085	12					
407	13		.25	ML		
519	14					
548	15		.50	ML		
SUPPLIES						
CARDS	16		1600			
FORMS	17		30			
	18					

DATE FINISHED 4-9-68 DELIVERED TO Lucy Strawn BY ML.

Figure 1

USER IDENT REG SUMMARY OF USAGE FOR MONTH ENDING FE 29 68

ITEM	QUANT	DIR COST	GROSS PROFIT	CHARGES
1	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00
3	19.35	96.75	120.94	217.69
4	0.00	0.00	0.00	0.00
5	41.70	125.10	156.38	281.48
6	0.00	0.00	0.00	0.00
7	17.54	0.00	698.44	698.44
8	7.15	0.00	7.51	7.51
9	4.90	0.00	4.26	4.26
10	0.00	0.00	0.00	0.00
11	2.40	0.00	2.28	2.28
12	0.00	0.00	0.00	0.00
13	13.40	0.00	119.26	119.26
14	1.40	0.00	4.41	4.41
15	6.95	0.00	10.08	10.08
16	17275.00	17.28	8.64	25.91
17	1324.00	20.62	10.31	30.93
18	0.00	0.00	0.00	0.00
TOTALS		259.75	1142.50	1402.25

21 JOBS PROCESSED

FIGURE 2.

ANDERSON COLLEGE COMPUTING CENTER

I N V O I C E

ARMSTRONG CORK CO.
DUNKIRK, IND 47336
ATTN.
MR. DICK SCHRECK

SERVICES FOR MONTH ENDING NOV 30 1967

I T E M	Q U A N T I T Y	A M O U N T
COMPUTER OPERATOR	9.25	62.44
TAB. OPERATOR	2.00	6.80
COMPUTER SYSTEM	1.26	50.65
KEYPUNCH	.85	.94
ACCOUNTING MACHINE	.85	12.07
TAB. CARDS	1700.00	2.55
FORMS	51.00	.38
MISCELLANEOUS	17.00	.85
TOTAL DUE		$ 136.68

PLEASE REMIT TO BUSINESS OFFICE
ANDERSON COLLEGE
ANDERSON, INDIANA 46011

FIGURE 3

ANDERSON COLLEGE COMPUTING CENTER

COMPUTER LOG

TIME / DATE		OPERATOR	TYPE	COMMENTS
START	FINISH			
2841.30	2841.41	ML	4	R 131 A
				FIGURE 4

1. Instruction, 2. Student Research, 3. Faculty, 4. Administrative, 5. Outside, 6. Computing Center, 7. Miscellaneous (Explain).

1620 MONITOR SYSTEM

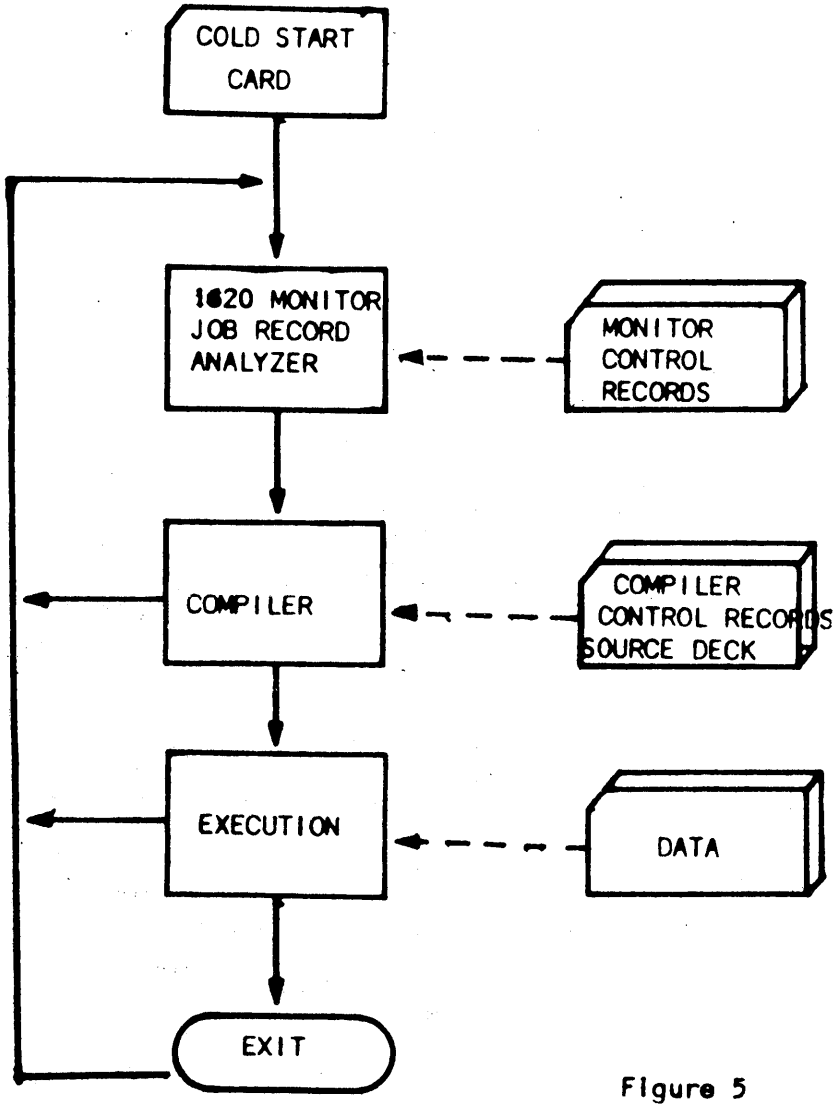


Figure 5

STUDENT MONITOR SYSTEM

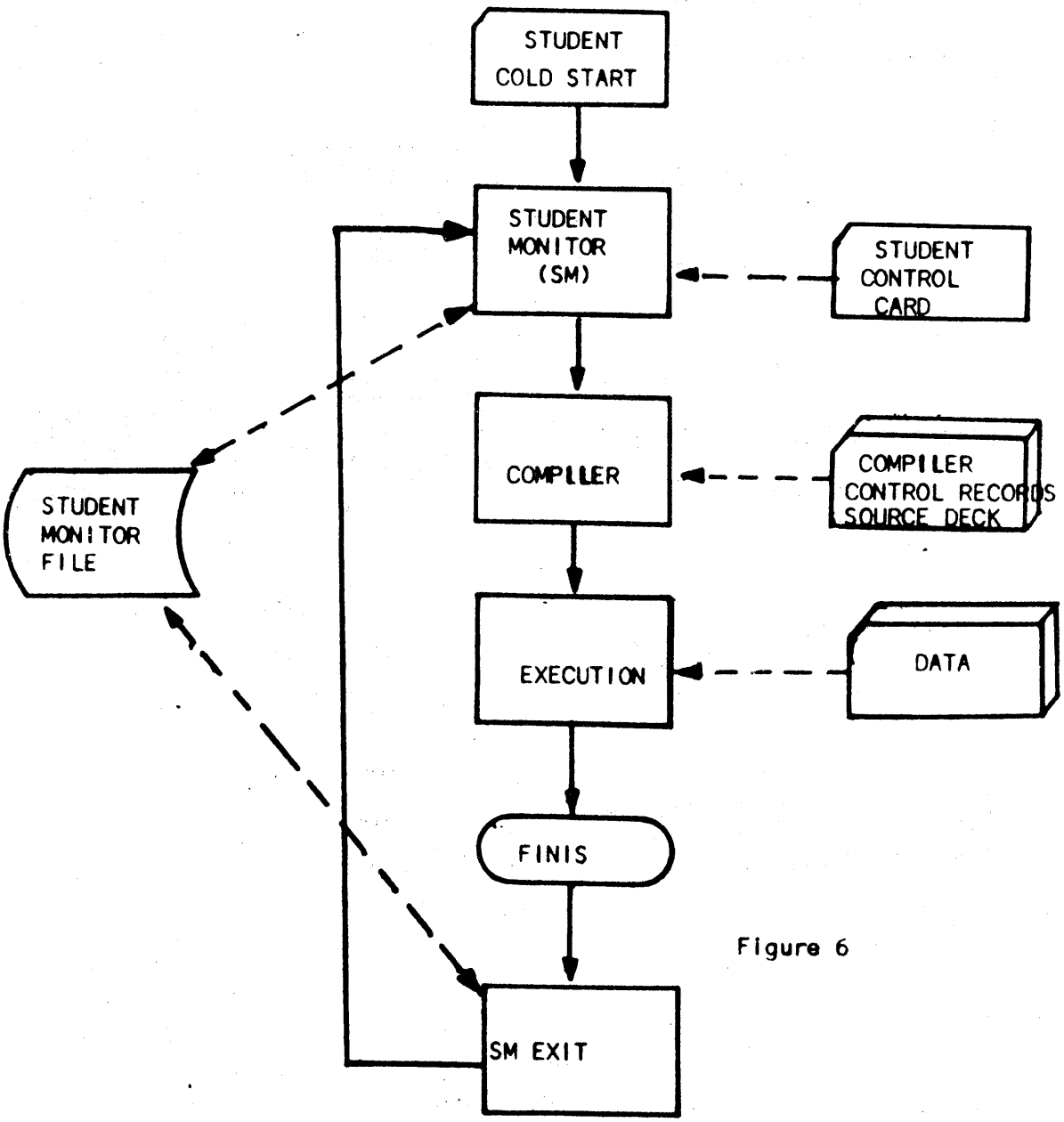


Figure 6

STUDENT CONTROL CARD

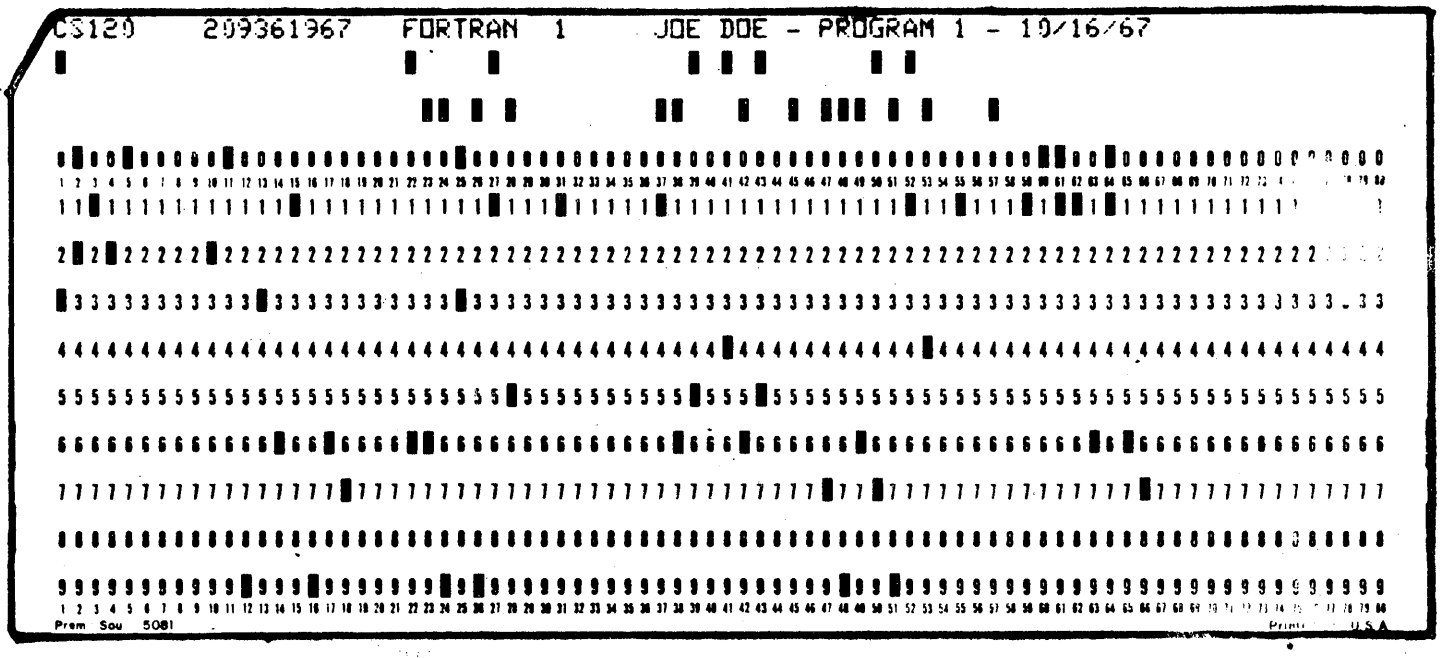


FIGURE 7

SSN	NAME	PGM 1	PGM 2	PGM 9	TOTALS
311506327	BALDWIN AUDREA LEE	03 03	00 00	00 00	10 06
527842918	BECKER RONALD DALE	00 00	00 00	00 00	00 00
307449077	BURCH LARRY LYNN	03 01	02 02	00 00	06 03
385488685	BURG GERALD BEAUDETTE	02 02	01 01	00 00	08 07
317362330	BURKETT HOMER EUGENE	02 02	02 02	00 00	04 01
239786965	BURNS GLENN ALLEN	02 01	02 02	00 00	05 03
293448540	CURRENT STEVE GERALD	01 01	02 01	00 00	00 00
384543943	EADS LINDA SUE	00 00	00 00	00 00	00 00
308428012	FLYNT TIMOTHY CRAIG	03 01	04 03	00 00	05 05
344426840	FORONDA FRANCINE ELEANOR	01 01	01 01	00 00	05 03
280424358	FRIEND CONSTANCE JO	02 02	01 01	00 00	06 05
396427293	HATCH KENNETH EUGENE	01 01	03 01	00 00	17 08
270449516	HILL THOMAS EUGENE	01 01	04 03	00 00	15 04
268484602	JOLLIFF MELVIN FRANCIS	01 01	03 01	00 00	18 05
373521849	KENDRICK JOHN DAVID	01 01	01 01	00 00	14 08
331441055	LEATHERMAN DUANE LYNN	01 01	03 01	00 00	05 03
162076970	MILLER WALTER WOODROW	01 01	00 00	00 00	00 00
214466246	MILSTEAD REBECCA LYNN	01 01	10 04	00 00	04 02
373462245	NICKEL CURTISS HARTLEY	01 01	01 01	00 00	09 06
307509917	RICHARDSON PATRICIA ANN	01 01	01 01	00 00	10 06
112385020	SHAW RICARDO ALEJANDRO	06 05	00 00	00 00	00 00
311466623	WHITESEL MICHAEL ARLO	02 02	02 02	00 00	09 03
561603693	BENSON FLOYD GERALD	01 01	02 01	00 00	04 00
306509294	FOX STEPHEN LANGDOWNE	01 00	02 01	00 00	07 04
505601346	MCKINLEY ANITA KAYLEEN	02 01	02 01	00 00	03 03
505620820	MOREY JEANNETTE RENEE	01 01	02 01	00 00	05 03
294444362	RICH CHARLES ELMER	03 02	01 01	00 00	13 09
206430759		16 11	08 01	00 00	00 00

FIGURE 8

AN ITERATIVE INFORMATION RETRIEVAL SYSTEM*

Daniel U. Wilde
New England Research Application Center
University of Connecticut
Storrs, Connecticut

The purpose of an iterative information retrieval system is to help the strategy designer make more efficient use of his data base. In the past when the results of an information retrieval run were unsatisfactory, the strategy designer (SD) had no choice but to redo his previous strategy and to request a complete computer rerun. This paper presents a strategy design procedure which permits SD to focus in on his final strategy by taking advantage of the results of his previous iterations. The strategy design procedure is general and can be applied to any computer system. The iterative procedure is especially suited for a small machine.

*This research was sponsored in part by the National Aeronautics and Space Administration Contract NSR-07-002-029.

AN ITERATIVE INFORMATION RETRIEVAL SYSTEM

Daniel U. Wilde
New England Research Application Center
University of Connecticut
Storrs, Connecticut

Introduction

The purpose of an iterative information retrieval system is to help the strategy designer make more efficient use of his data base. In the past when the results of an information retrieval run were unsatisfactory, the strategy designer (SD) had no choice but to redo his previous strategy and to request a complete computer rerun. This paper presents a strategy design procedure which permits SD to focus in on his final strategy by taking advantage of the results of his previous iterations. The strategy design procedure is general and can be applied to any computer system. The iterative procedure is especially suited for a small machine.

Background of the Problem

During the past year, the New England Research Application Center has operated as a NASA Research Dissemination Center at the University of Connecticut. The purpose of the center is to aid and promote technology transfer in the New England area by aiding regional industry locate appropriate technical information. During this period, we have performed 400 complete retrospective computer searches on our data base of 315,000 documents. In

general the results of the first search attempt for a new search strategy have fallen into two categories. Usually the number of retrieved document citations is either overwhelmingly large or disappointingly small. If the retrieval output is large, SD must tighten his next strategy attempt. If the retrieval output is small, he must loosen his strategy and broaden his search coverage. In either case, we have found that SD follows no orderly process in improving his next retrieval run. The iterative procedure was designed to permit SD to focus in on his final strategy by taking advantage of the results of his previous attempts.

The Iterative Retrieval System

For simplicity let us assume that SD is designing Boolean logic search strategies. (However, the following discussion is also applicable to threshold logic, sometimes referred to as weighted term logic.)

$$\text{STRATEGY} = A^* (B+C) + D^*E \qquad (1)$$

From the above Boolean expression it can be seen that one of the following abbreviated expressions must be true if the complete expression is also to be true.

$$S_1 = A + D \qquad S_2 = A + E$$
$$S_3 = B + C + D \qquad S_4 = B + C + E \qquad (2)$$

Why not break the computer search into a series of passes and let SD revise or refine his strategy after he has evaluated the results of the previous pass?

First, SD designs a very general abbreviated strategy which will hopefully retrieve a great percentage of the relevant documents from the data bank. It should be expected that this first iteration will also retrieve a great amount of trash. The abbreviated expression is used to search the complete data bank. If during the search a document record satisfies the abbreviated expression, the record is copied onto a scratch tape. The scratch tape becomes a subset of the input file and approximates the desired final result. At the end of the pass, a limited portion of the scratch tape is printed and given to SD for his evaluation.

Second, SD evaluates the results of the previous pass. If his evaluation shows that the scratch tape contains relevant documents plus too much trash, SD refines his strategy by making it more specific. Now the above first step is repeated using the scratch tape as the input file. If however the scratch tape contains few relevant documents, SD specifies a broader abbreviated expression; and the first step must be repeated using the complete data bank as the input file.

An Iterative Retrieval Example

In order to illustrate the above operational concept, an actual search, performed at the University of Connecticut using its complete file of 315,000 documents, will be described. "Orbits of Comets" was chosen as the example search.

After a brief discussion with the search requester, SD specified a six term abbreviated search strategy, S_1, which he felt covered the major search concepts.

$$S_1 = \text{ORBIT} + \text{ORBITS} + \text{COMET} + \text{COMETS} + \text{CELESTIAL MECHANICS} + \text{ASTRONOMY} \quad (3)$$

SD arbitrarily requested a printout of the first 100 documents from the first-pass scratch tape. The results of the iterative retrieval passes are summarized in Table 1. The run-times are in minutes for an 8k IBM 1401 with only two 7330 tape drives without process overlap. Similar results have also been established for an IBM 7040 programmed in FORTRAN and an IBM 360/65 programmed in PL/1.

	Number of Search Terms	Run Time	Retrieved Documents	Documents Searched
Pass 1	6	169	8,284	315,000
Pass 2	2	6	674	8,284
Pass 3	15	2	164	674

Table 1 - Summary of the Iterative Retrieval Passes

The first-pass printout listed the first 100 document citations and indicated that the abbreviated strategy copied 8,284 documents onto the scratch tape. SD's evaluation indicated that the two "comet" terms were retrieving relevant documents. He therefore revised his abbreviated strategy down to a two term expression, S_2.

$$S_2 = \text{COMET} + \text{COMETS} \quad (4)$$

SD again requested a printout of the first 100 documents from the second-pass scratch tape.

The second-pass printout indicated that the revised strategy retrieved 674 documents. SD's evaluation of the 100 citations suggested that he should now make his next strategy iteration, S_3, more specific. This he accomplished by "anding" an "ored" series of 13 "orbit" related terms to his second-pass strategy, S_2. He also stated that he was ready to receive a printout of the complete third-pass scratch tape.

$$S_3 = (\text{COMET} + \text{COMETS}) * (\text{ORBIT CALCULATION} + \ldots + \text{SOLAR ORBITS}) \quad (5)$$

The third-pass printout listed 164 document citations. The SD had succeeded in retrieving a reasonable number of documents by focusing in on his subject. He was now ready to begin his final manual selection process.

The above example is typical of the retrospective searches we have performed. The search requester was not familiar with our data base, and the SD had no previous retrieval experience in the field of astronomy. Our statistics show that if the second and third passes had used the whole data bank, they would have used 72 minutes and 210 minutes, respectively, as opposed to 6 and 2. A discussion of relevancy improvement has purposefully been omitted since it has been discussed elsewhere (1).

Advantages of the Iterative System

The iterative system with its intermediate scratch tape permits SD to focus in on his final retrieval strategy. First, SD develops a very general abbreviated search strategy and discovers the approximate relevant contents of his data base. Second,

armed with this information, SD refines his strategy by iteratively searching his scratch tape. No longer must SD fear that he will be overwhelmed by document citations as a result of his first attempt at a new search request. He simply designs a broad search and asks for a partial scratch tape printout.

Strategy design involves two operations. First, the selection of proper index terms; and second, the proper logical interconnection of those terms. The usual retrieval system requires that SD master and succeed at both operations simultaneously. If either operation is not correct, he must repeat the complete operation and ask for another total data base search. However, the iterative system permits SD to do the two operations in sequence. First, he selects the proper index terms and insures their validity by evaluating his intermediate scratch tape printout. Second, he obtains the proper logical interconnection of his index terms by iterating on his scratch tape. Now the two strategy design operations have been separated and are independent.

Finally, if SD uses a broad abbreviated search strategy for the first pass, the iterative system requires only one search of the entire data base. His iterations on the scratch tape can be processed faster since the scratch tape is shorter than the data base file. Thus, SD can satisfy a search request in less time.

Bibliography

1. Wilde, D.U., "Computer-Aided Strategy Design Using Adaptive and Associative Techniques," to be published in the Proceedings of the 31st Annual Meeting of the American Society for Information Science, October 21-24, 1968.

A SURVEY OF PROGRAMS OF POTENTIAL USE TO HIGH SCHOOL USERS

by

Frederick R. Henderson
Director, Computer Center
Rochester Institute of Technology
Rochester, New York 14623
(1620 User #1393)

ABSTRACT

The author has selected about thirty programs from the "IBM Catalog of Programs for IBM 1620 Systems" which are believed to be particularly well suited for use in high school installations. These include compilers, utility programs, mathematical and statistical programs, demonstration programs, and teaching aids.

Presented at
COMMON Meeting
Philadelphia, Pa.
September, 1968

Session T3G

A SURVEY OF PROGRAMS OF POTENTIAL USE TO HIGH SCHOOL USERS

The current issue of the "IBM Catalog of Programs for IBM 1620 Systems" (IBM Form C20-1603) contains abstracts of more than 700 computer programs and is well over 100 pages in length. The attached "Extract" of programs which were selected as being especially useful for high school installations contains less than 30 abstracts on four pages, or less than 4% of the total catalog listings. Of necessity this "Extract" constitutes a highly arbitrary selection from the total catalog, but we hope it will prove to be a useful one.

The compilers and translators listed on pages 1 and 2 of the "Extract" will be discussed this afternoon, but there are two other programs here which deserve brief comments.

UG-01X (COGO I) is an example of a growing body of programs designed for specialized users who have a minimum knowledge of computer programming. We have, for example, conducted brief seminars for practicing land surveyors with no knowledge of computers, who in two or three hours were making very effective use of COGO to process their surveying data.

01.3.009 (SEQUENCE PUNCHER) is one of the most useful utility programs in the Library for sequencing cards either before or after other punching.

On page 3 of the "Extract" are a limited number of mathematical and statistical packages. These were selected because we have used them ourselves and like them, or because other people have and COMMON has certified them. Except for the first one (05.0.035), all of them were written for a basic 20K machine; some do require autodivide hardware.

05.0.035 (SUBROUTINE CROUT) is included because we happen to be partial to this method for solving simultaneous linear systems. We have modified this program slightly to make it an independent one rather than a subroutine and also to calculate the magnitude of the determinant which is a simple by-product in the Crout method. A copy of this program which we supply to our students is attached, as well as a companion program for the Gauss-Seidel iteration. (The same data cards can be used for both programs.) The input/output and format statements would have to be modified to run these on systems which cannot handle the implied DO-loops.

Programs for finding the roots of equations have been intentionally omitted, for it is my feeling that students can and should write such programs themselves. One of our favorite exercises is Richard Andree's quadratic* equation: $X^2 + 40000X + 20 = 0$ (note that zero is not a root). Because of the limitations of finite arithmetic, this is not easily solved using the quadratic formula; iteration, however, works beautifully. This serves to emphasize the importance of mathematical analysis -- brains over brawn -- in connection with any computer problem.

On page 4 of the "Extract" are four of our favorite demonstration programs. These are useful at Open House and P.T.A. meetings, or whenever you have some visitors. The first one (11.0.023) is short and simple and quite impressive if one has never seen a computer in action before. But be sure to try them out in advance to be sure they are working properly.

---

* Andree, R.V., "Computer Programming and Related Mathematics", page 195, Wiley, 1967

Also on page 4 are several programs which I have classified as "teaching aids" because they are of interest to teachers rather than students.

02.0.052 (COMPUTEST) will be of interest to any teachers who wants to experiment a little with computer-assisted instruction, preferably outside of regular school hours.

13.0.013 (FORTRAN TEACH) although programmed for a 40K memory, can be easily modified to run on a 20K machine. This program is described in more detail in an article by Olsen, Pope, and Watkins entitled "Teaching Digital Computer Programming" in the Journal of Engineering Education, Volume 54, Number 10, June, 1964, pages 344-5.

13.0.003 (NUTS) is one of the most useful programs in the Library if mark-sense punching facilities are available. Teachers who give objective-type tests welcome it with open arms, and it has built a lot of good-will for our Computer Center. We have found that this kind of service for the faculty pays big dividends if it is done well.

Last but by no means least, we give you "Linus -- Our Hero". He is not in the Program Library, but if you want to reproduce this for your friends and visitors, just drop us a line, and we shall be glad to send you a copy of the card deck. The address is: Computer Center, Rochester Institute of Technology, Rochester, New York 14623.

EXTRACT FROM IBM CATALOG OF PROGRAMS FOR IBM 1620 SYSTEMS
(IBM Form C20-1603)

The programs given below have been selected as being particularly well adapted for use in high school installations with only 20K memory. For further information, please refer to the complete catalog.

(Prepared for COMMON by F. R. Henderson, User No. 1393, Sept., 1968)

IBM Programs

1620-FO-004 FORTRAN WITH FORMAT /CARD/
ORDER THROUGH LOCAL IBM BRANCH OFFICE
SPECIFY FILE NUMBER 1620-FO-004

THE NEW VERSION INCORPORATES THE SET OF SUBROUTINES PREVIOUSLY AVAILABLE AS 1620 F/F-APP SUBROUTINES, NO. 1620-LM-022 /CARD/ IT ALSO PROVIDES A NEW SET OF SUBROUTINES DESIGNED SPECIFICALLY FOR 1620 SYSTEMS EQUIPPED WITH THE AUTOMATIC FLOATING POINT OPERATIONS, ADDITIONAL INSTRUCTIONS, AND INDIRECT ADDRESSING FEATURES. THE 1620 FORTRAN WITH FORMAT SYSTEM BRINGS THE ADVANTAGES OF THE FORTRAN LANGUAGE TO 1620 USERS. THE LANGUAGE OF FORTRAN CLOSELY RESEMBLES THE LANGUAGE OF MATHEMATICS. THE SIMILARITY ALLOWS TECHNICAL PERSONNEL WHO ARE NOT TRAINED AS PROGRAMMERS TO PREPARE THEIR PROBLEMS FOR THE 1620. FOR SCIENTIFIC AND ENGINEERING COMPUTATIONS WHERE PROBLEMS ARE USUALLY EXPRESSED IN MATHEMATICAL FORMULAE. FORTRAN IS EXTREMELY USEFUL. A SOURCE PROGRAM WRITTEN IN THE FORTRAN LANGUAGE IS PROCESSED BY THE 1620 FORTRAN WITH FORMAT COMPILER /PROCESSOR/ TO PRODUCE A 1620 MACHINE-LANGUAGE OBJECT PROGRAM, WHICH IS THEN LOADED AND EXECUTED. INPUT/OUTPUT FACILITIES CONTAIN PROVISION FOR INCLUDING FIXED OR VARIABLE ALPHABETIC INFORMATION IN THE TYPED OR PUNCHED RESULTS. THE APPROPRIATE SET OF FLOATING POINT ARITHMETIC AND FUNCTIONAL SUBROUTINES MAY BE COMPILED WITH THE OBJECT PROGRAM OR LOADED WITH THE COMPILED PROGRAM PRIOR TO EXECUTION. FUNCTIONAL SUBROUTINES ARE SELECTIVELY INCLUDED AS NECCESSARY. THE PRECISIONS OF FIXED POINT AND FLOATING POINT ARTHMETIC ARE 4 AND 8 DECIMAL DIGITS, RESPECTIVELY. FOUR SETS OF FIXED LENGTH FLOATING POINT SUBROUTINES ARE AVAILABLE, DESIGNED FOR FOUR DIFFERENT 1620 SYSTEM FEATURE CONFIGURATIONS. MINIMUM MACHINE REQUIREMENTS--FOR BOTH COMPILATION AND EXECUTION- A 20K 1620 MODEL 1 OR 2 WITH...1622 CARD READ PUNCH OR 1621 PAPER TAPE READER AND 1624 TAPE PUNCH.

THE MAGNETIC TAPE REQUIRED TO OBTAIN THE OPTIONAL MATERIAL MAY BE SUPPLIED OR ORDERED FROM YOUR IBM REPRESENTATIVE. THE OPTIONAL MATERIAL REQUESTED MUST BE ITEMIZED ON THE ORDER CARD.

BASIC PROGRAM MATERIAL -
DOCUMENTATION - PROGRAM WRITE-UP...LISTINGS...FLOWCHARTS.
ONE OF THE FOLLOWING ALTERNATES MUST BE ORDERED DEPENDING UPON THE FEATURES AVAILABLE ON THE SYSTEM TO BE USED.
OPTION A - SUBROUTINES /WITH LISTINGS/ FOR MACHINES WITH THE AUTOMATIC DIVIDE FEATURE.
1620 FORTRAN WITH FORMAT PROCESSOR.
OPTION B - SUBROUTINES /WITH LISTINGS/ FOR MACHINES WITH THE AUTOMATIC FLOATING POINT OPERATIONS FEATURE.
1620 FORTRAN WITH FORMAT PROCESSOR.
OPTION C - SUBROUTINES /WITH LISTINGS/ FOR MACHINES WITH THE AUTOMATIC FLOATING POINT OPERATIONS, ADDITIONAL INSTRUCTIONS, AND INDIRECT ADDRESSING.
1620 FORTRAN WITH FORMAT PROCESSOR.
OPTION D - SUBROUTINES /WITH LISTINGS/ FOR MACHINES WITHOUT THE AUTOMATIC DIVIDE FEATURE.
1620 FORTRAN WITH FORMAT PROCESSOR.

OPTIONAL PROGRAM MATERIAL -
ONE MAGNETIC TAPE - SOURCE DECKS AVAILABLE ON MAGNETIC TAPE THRU YOUR LOCAL REGIONAL OFFICE.

1620-FO-006 FORTRAN PRE-COMPILER /CARD/
ORDER THROUGH LOCAL IBM BRANCH OFFICE
SPECIFY FILE NUMBER 1620-FO-006

PURPOSE THIS PROGRAM DETECTS AND PERMITS CORRECTION OF ERRORS IN A FORTRAN SOURCE PROGRAM BEFORE THE OBJECT PROGRAM IS COMPILED. THE PRE-COMPILER DETECTS MANY OF THE MORE COMMON PROGRAMMING ERRORS IN INDIVIDUAL SOURCE STATEMENTS, AND INDICATES POSSIBLE LOGICAL ERRORS IN THE SOURCE PROGRAM AS A WHOLE. STORAGE REQUIREMENTS 20,000 POSITIONS. EQUIPMENT SPECIFICATIONS 1620 CPU, 1622 CARD READER PUNCH.

BASIC PROGRAM MATERIAL -
DOCUMENTATION - PROGRAM WRITE-UP... LISTINGS.
CARD DECKS - OBJECT DECKS.

1620-FR-011 GOTRAN /CARD/
ORDER THROUGH LOCAL IBM BRANCH OFFICE
SPECIFY FILE NUMBER 1620-PR-011

PURPOSE A RELATIVELY FAST COMPILER FOR PROGRAMS WHICH WILL GENERALLY BE EXECUTED ONLY ONCE. METHOD GOTRAN STORES THE COMPILED PROGRAM IN MEMORY DURING COMPUTATION. THE OBJECT PROGRAM IS THEN EXECUTED IN AN INTERPRETIVE MODE. NO OBJECT TAPE OR DECK IS PRODUCED. AFTER EXECUTION OF AN OBJECT PROGRAM, COMPUTATION OF A NEW OBJECT PROGRAM IS POSSIBLE WITHOUT LOADING THE PROCESSOR. RESTRICTIONS, RANGE THE LANGUAGE USED IN GOTRAN IS A MODIFIED SUBSET OF FORTRAN, INCLUDING THE FUNCTIONAL SUBROUTINES. ARITHMETIC STATEMENTS ARE RESTRICTED TO ONE ARITHMETIC OPERATION PER STATEMENT. DATA IS HANDLED IN THE FORM OF 10 DIGIT FLOATING POINT NUMBERS OR 3 DIGIT FIXED POINT NUMBERS. INPUT-OUTPUT IS THE SAME FORM AS FORTRAN WITH THE EXCEPTION THAT CARDS ARE PUNCHED WITH ONE ITEM PER CARD. THE MAXIMUM NUMBER OF SYMBOLS THAT MAY BE USED IS 400. THE NUMBER STATEMENTS ALLOWED IS INVERSELY PROPORTIONAL TO THE NUMBER OF SYMBOLS USED. APPROXIMATELY 211 STATEMENTS CAN BE COMPILED USING 200 SYMBOLS. STORAGE REQUIREMENTS NOT GIVEN. EQUIPMENT SPECIFICATIONS BASIC 1620, CARD.

BASIC PROGRAM MATERIAL -
DOCUMENTATION PROGRAM WRITE-UP... LISTINGS.
CARD DECKS - OBJECT CARD DECKS.

1620-UG-01X CIVIL ENGINEERING COORDINATE GEOMETRY COGO 1 /CARD/
ORDER THROUGH LOCAL IBM BRANCH OFFICE
SPECIFY FILE NUMBER 1620-UG-01X

COGO 1 IS A PROBLEM ORIENTED PROGRAMMING SYSTEM DESIGNED FOR CIVIL ENGINEERING GEOMETRY PROBLEMS. THE SYSTEM IS APPLICABLE TO ALL PHASES OF HORIZONTAL GEOMETRICAL DESIGN SUCH AS RIGHT-OF-WAY COMPUTATIONS, SUB-DIVISION LAYOUT, HIGHWAY AND RAMP DESIGN, AND STRUCTURAL BRIDGE GEOMETRY. USING COGO 1 THE ENGINEER MERELY STATES HIS PROBLEM, IN HIS OWN PROFESSIONAL LANGUAGE, TO THE 1620. THE COMPUTER THEN SOLVES THE PROBLEM WITHOUT THE NEED FOR ANY MACHINE PROGRAMMING BY THE ENGINEER. THE CONCEPTS INVOLVED ARE APPLICABLE TO A WIDE RANGE OF ALL ENGINEERING DESIGN PROBLEMS.
FEATURES-
A PROGRAMMING SYSTEM THAT ALLOWS THE CIVIL ENGINEER TO STATE HIS PROBLEM TO THE 1620 IN HIS OWN PROFESSIONAL LANGUAGE.
-AN EFFECTIVE SALES TOOLS, ALLOWING THE ENGINEER TO USE THE COMPUTER ON HIS OWN PROBLEMS WITH NO MACHINE PROGRAMMING TRAINING.
-THE 1620 CAN RECOGNIZE THE CIVIL ENGINEER'S VOCABULARY.
-NO FORMS ARE INVOLVED FOR ORIGINAL DATA PREPARATION. THE ENGINEER CAN USE ANY PIECE OF PAPER TO STATE HIS PROBLEM -- EVEN A SURVEYING FIELD BOOK.
-THERE IS NO LIMIT TO THE SIZE OF PROBLEM THAT CAN BE SOLVED. THE SYSTEM IS OPEN-ENDED.
-NO PROGRAMMING-- IN THE USUSL SENSE OF THE WORD-- IS EVER NECESSARY.
-THE SYSTEM IS EASILY MODIFIED AND EXPANDED TO FIT THE CUSTOMERS NEEDS.
-THE ENGINEER USES HIS EXPERIENCE, JUDGEMENT, AND CREATIVE ABILITY IN CLOSE COMMUNICATION WITH THE 1620.

THE COGO SYSTEM IS DIVIDED INTO A MAIN ROUTINE AND ELEVEN SUBROUTINE DECKS ENCOMPASSING 43 CIVIL ENGINEERING TERMS. THE USER WRITES THE DESCRIPTION OF HIS PROBLEM AND HOW TO SOLVE IT WITH THESE CIVIL ENGINEERING TERMS. EACH LINE OF DATA AND DESCRIPTIVE INFORMATION HE HAS WRITTEN IS PUNCHED ON CARDS OR ENTERED ON THE CONSOLE TYPEWRITER. THE USER THEN SELECTS THE CORRECT SUBROUTINE DECKS AND INCLUDES THEM WITH HIS DESCRIPTION DECK. THEREFORE, THE RUN IS TAILORED TO THE SPECIFIC PROBLEM AT HAND AND SINCE THE TERMS CAN BE REUSED, THERE IS NO LIMIT TO THE EXTENT OF A RUN. MINIMUM MACHINE REQUIREMENT-ANY 20K 1620 CARD.

OPTIONAL MATERIAL REQUESTED MUST BE ITEMIZED ON THE ORDER CARD.

BASIC PROGRAM MATERIAL -
DOCUMENTATION - PROGRAM WRITE-UP... REFERENCE MANUAL... FLOWCHARTS ... LISTINGS.
CARD DECKS - SOURCE DECK... OBJECT DECK.

1620-01.1.010 AFIT IMPROVED FORTRAN /CARD/
AVAILABLE 1ST QUARTER 1965.
SPECIFY FILE NUMBER 1620-01.1.010

AUTHOR...R.L. PRATT
SENIOR COMPUTER PROGRAMMER
DATA CORPORATION
7500 OLD XENIA PIKE
DAYTON, OHIO 45432

DIRECT INQUIRIES TO AUTHOR

IMPROVEMENT OF IBMS 1620 FORTRAN SYSTEM. MAJOR CHANGES ARE- INPUT MAY BE UNFORMATTED- COMPILER AND PRE-COMPILER ARE BUILT INTO THE SAME PROGRAM- SEVERAL PROGRAMS MAY BE COMPILED WITHOUT RELOADING THE COMPILER EACH TIME- LISTER IS INCLUDED TO PROVIDE SPS LISTING OF COMPILED PROGRAM. STORAGE REQUIREMENTS- 20K EQUIPMENT SPECIFICATIONS- CARD 1620, MEMORY 20K. NO OTHER SPECIAL FEATURES REQUIRED.

*** THIS PROGRAM HAS BEEN CERTIFIED BY COMMON.

1620-01.3.009 SEQUENCE PUNCHER /CARD/
AVAILABLE 1ST QUARTER 1965.
SPECIFY FILE NUMBER 1620-01.3.009

AUTHOR...R.L. PRATT
SENIOR COMPUTER PROGRAMMER - DATA CORPORATION
7500 OLD XENIA PIKE
DAYTON,OHIO

DIRECT INQUIRIES TO AUTHOR

PUNCHES ANY DESIRED SEQUENCE NUMBERS IN ANY ONE TO TWENTY COLUMNS OF A CARD DECK. CARD I/O AND A MEMORY OF 20K. WRITTEN IN SPS.

*** THIS PROGRAM HAS BEEN CERTIFIED BY COMMON.

1620-02.0.024 UTO FORTRAN SYSTEM II/62
/CARD/
AVAILABLE 2ND QUARTER 1964.
SPECIFY FILE NUMBER 1620-02.0.024

AUTHORS..E. STEWART LEE JAMES A. FIELD

DIRECT INQUIRIES TO..
E. STEWART LEE
DEPT. OF ELECTRICAL ENGINEERING
UNIVERSITY OF TORONTO
TORONTO 5, ONTARIO, CANADA

PURPOSE- A FORTRAN COMPILER WITH RAPID, BATCH COMPILING AND ADDITIONAL LANGUAGE FACILITIES. IMPROVEMENTS HAVE ALSO BEEN MADE IN THE SUBROUTINES. THE LANGUAGE IS AN EXTENSION OF IBM FORMAT FORTRAN. OBJECT PROGRAMS BEGIN AT LOCATION 07070. CARD I/O NO SPECIAL FEATURES REQUIRED. ANY SIZE MEMORY MAY BE USED.

*** THIS PROGRAM HAS BEEN CERTIFIED BY COMMON.

1620-02.0.029 NCE LOAD AND GO FORTRAN FOR
20K /CARD/
AVAILABLE 1ST QUARTER 1965.
SPECIFY FILE NUMBER 1620-02.0.029

AUTHORS..GEORGE RUMRILL BRUCE FOWLER

DIRECT INQUIRIES TO..
COMPUTING CENTER
NEWARK COLLEGE OF ENGINEERING
323 HIGH ST.
NEWARK 2, N.J.

THIS PROGRAM IS A COMPILER-INTERPRETER CAPABLE OF BATCH PROCESSING SOURCE DECKS WRITTEN IN A MODIFIED FORM OF THE FORTRAN LANGUAGE. ANSWERS ARE IMMEDIATELY AVAILABLE WITHOUT ANY HANDLING OF OBJECT DECKS. THE PROGRAM IS INTENDED TO BRING TO THE USER OF THE 20K 1620 THE ADVANTAGES OF LOAD AND GO OPERATION, ESPECIALLY WHEN PROCESSING SHORT STUDENT PROGRAMS. CANNOT BE DAMAGED BY ERRORS IN PROGRAMS. MANY ERROR CHECKS ARE MADE, BOTH DURING COMPILATION AND EXECUTION. NO FORMAT SPECIFICATIONS ARE USED. OUTPUT FORMAT IS DETERMINED BY THE TYPE AND RANGE OF THE VARIABLE. INPUT FORMAT IS FREE FORM. FEATURES INCLUDE- BUILT-IN TRACE, ARRAY INPUT AND OUTPUT, PROVISION FOR SUB-PROGRAM LINKAGES, LIMITED HOLLERITH LISTING, AND UNDEFINED VARIABLE DETECTION. SPECIFICATIONS-A. STORAGE USED BY PROGRAM-20,000 POSITIONS. B. EQUIPMENT REQUIRED BY PROGRAM-AUTO DIVIDE-INDIRECT ADDRESSING- CARD OR PAPER TAPE, NOT BOTH, MODEL I MACHINES ONLY. C. PROGRAMMING TYPE- OTHER PROGRAMMING LANGUAGE OSAP. OHIO STATE UNIVERSITY ASSEMBLY PROGRAM /A MODIFIED FORM OF SPS LIBRARY NO. 01.1.012.

*** THIS PROGRAM HAS BEEN CERTIFIED BY COMMON.

If you have a disk, be sure to try C4D. →

1620-02.0.031 PDQ FORTRAN /AN INTERPRETIVE
PROGRAM/
AVAILABLE 3RD QUARTER 1964.
SPECIFY FILE NUMBER 1620-02.0.031

AUTHOR...FRANK M. MASKIELL
PENNSYLVANIA TRANSFORMER DIVISION
MCGRAW-EDISON COMPANY
CANONSBURG, PENNSYLVANIA

DIRECT INQUIRIES TO AUTHOR

PDQ FORTRAN IS A MODIFICATION OF THE UTO FORTRAN AND FORTRAN WITH FORMAT, WHICH UTILIZES FLOATING POINT VARIABLES IN THE EXCESS 50 NOTATION. IN THE HUNDRED PLUS PROGRAMS COMPILED TO DATE BY THE SYSTEM, THE OBJECT TIME RUNNING IS LESS, THE SIZE OF THE OBJECT DECK IS SMALLER, AND CORE STORAGE REQUIREMENTS FOR THE OBJECT PROGRAM, SUBROUTINES, AND DATA IS LESS THAN ANY OTHER FORTRAN SYSTEM WITHOUT FLOATING POINT HARDWARE. SPECIFICATIONS- A. STORAGE USED BY PROGRAM- THE PROCESSOR REQUIRES 16006 DIGITS PERMITTING 199 SYMBOL TABLE ENTRIES ON 20K, 2179 SYMBOL TABLE ENTRIES ON 40K. CLASS A SUBROUTINES FOR THE OBJECT PROGRAM REQUIRE 6600 DIGITS. INSTRUCTIONS OF THE OBJECT PROGRAM BEGIN LOADING IN LOCATION 6600.
B. EQUIPMENT REQUIRED BY PROGRAM- CARD SYSTEM- AUTO DIVIDE. PROGRAM WILL OPERATE ON 20K AND WILL INTERNALLY ADJUST FOR ANY ADDITIONAL STORAGE AVAILABLE. PROGRAMS MAY BE COMPILED ON A MACHINE 40K OR GREATER, FOR A MACHINE OF LESSER CAPACITY BY MEANS OF A CONTROL DIGIT.
C. THE PROCESSOR AND SUBROUTINES ARE WRITTEN IN SPS AND THEN COMPRESSED.

ANY PROGRAM IN FO-004 LANGUAGE MAY BE COMPILED IN THE SYSTEM, HOWEVER, ADDITIONAL LANGUAGE FACILITIES ARE INCLUDED. THE FO-004 LANGUAGE HAS BEEN EXPANDED IN THE PDQ FORTRAN SYSTEM TO INCLUDE- /A/ COMMON STATEMENT FOR RESERVING LOCATIONS IN THE SYMBOL TABLE FOR NONSUBSCRIPTED AND SUBSCRIPTED VARIABLES, /B/ BATCH COMPILATION OF PROGRAMS WITHOUT SUBROUTINES, /C/ CONTINUATION CARDS FOR FORMAT AND INPUT/OUTPUT STATEMENTS, /D/ REPETITION OF FIELD FORMAT /NFW.D/ ETC.--FORMAT ALSO INCLUDES AN A AND O SPECIFICATION, /E/ LISTING OR PUNCHING OF REFERENCED SOURCE STATEMENTS AND SYMBOL TABLE, /F/ PROCEDURE STATEMENTS PERMITTING A GROUP OF FORTRAN STATEMENTS TO BE UTILIZED AS A SUBROUTINE SIMILAR TO THE FORTRAN II SUBROUTINE SUBPROGRAM, /G/ TRACE FACILITY WITHOUT GENERATING ADDITIONAL INSTRUCTIONS IN LINE AND INCLUDING THE ADDRESS AS WELL AS THE MAGNITUDE OF THE VARIABLE AT RUNNING TIME,--THE FORMAT OF TRACE AT OBJECT TIME MAY BE ALTERED BY A SINGLE INSTRUCTION, /H/ TWO SUBROUTINE DECKS, ONE PERMITTING A RELAXED INPUT FORMAT REQUIRING ONLY THAT A SPACE OR BLANK COLUMN SEPARATE INPUT VARIABLES, AND THE SECOND REQUIRING INPUT DATA TO BE IN THE PRECISE FORMAT OF THE INPUT STATEMENTS---EITHER SUBROUTINE DECK MAY BE USED WITH THE COMPILED OBJECT PROGRAM DEPENDING ON THE FORMAT OF DATA TO BE USED.

*** THIS PROGRAM HAS BEEN CERTIFIED BY COMMON.

1620-02.0.057 NCE LOAD AND GO FORTRAN
FOR 20K /CARD/
AVAILABLE 4TH QUARTER 1966.
SPECIFY FILE NUMBER 1620-02.0.057

AUTHORS..G. RUMRILL B. FOWLER H. SEWARD

DIRECT INQUIRIES TO..
G. RUMRILL,NEWARK COLLEGE OF ENGINEERING,COMPUTING CENTER,
323 HIGH ST.,NEWARK 2, N.J.

THIS PROGRAM IS A COMPILER-INTERPRETER CAPABLE OF BATCH PROCESSING SOURCE DECKS WRITTEN IN A SUBSET OF THE FORTRAN LANGUAGE. ANSWERS ARE IMMEDIATELY AVAILABLE WITHOUT ANY HANDLING OF OBJECT DECKS. THE PROGRAM IS INTENDED TO BRING TO THE USER OF THE 20K 1620 THE ADVANTAGES OF LOAD AND GO OPERATION, ESPECIALLY WHEN PROCESSING 100 TO 200 CARD SOURCE DECKS. THE SYSTEM CANNOT BE DAMAGED BY ERRORS IN PROGRAMS. MANY ERROR CHECKS ARE MADE, BOTH DURING COMPILATION AND EXECUTION. NO FORMAT SPECIFICATIONS ARE USED. OUTPUT FORMAT IS DETERMINED BY THE TYPE AND RANGE OF THE VARIABLE. INPUT FORMAT IS FREE FORM. FEATURES INCLUDE- BUILT-IN TRACE, SINGLE AND DOUBLE SUBSCRIPTING, COMPUTED GO TO STATEMENTS, LIMITED HOLLERITH LISTING, AND UNDEFINED VARIABLE DETECTION. EQUIPMENT REQUIRED BY PROGRAM- 1620 MODEL I OR II, 20K, AUTO DIVIDE., INDIRECT ADDRESSING CARD OR PAPER TAPE, NOT BOTH. OTHER PROGRAMMING LANGUAGE USED N.C.E. HIGH SPEED ASSEMBLER. LIBRARY NUMBER- 1.1.029.

1620-02.0.059 C4D - AN OPERATING SYSTEM
FOR PDQ FORTRAN /CARD/
AVAILABLE 2ND QUARTER 1967.
SPECIFY FILE NUMBER 1620-02.0.059

AUTHORS..J. GRANT G. LILLY F. MASKIELL
M.L. MCATEER L. POWELL

DIRECT INQUIRIES TO..
FRANK M. MASKIELL,PENNSYLVANIA TRANSFORMER DIV.,
BOX 330,CANONSBURG, PA.

C4D IS A PDQ FORTRAN OPERATING SYSTEM PROVIDING BATCH PROCESSING, A MIXED GROUP OF FORTRAN COMPILATION AND EXECUTION RUNS. IT PERMITS STORAGE OF PROGRAMS AND DATA ON ONE OR MORE 1311 DISK DRIVES. SEGMENTATION OF PROGRAMS IS POSSIBLE. THE SYSTEM CONTAINS VERY COMPREHENSIVE ERROR CHECKING. THE OPERATING SYSTEM RESIDES ON THE DISK AND OCCUPIES LESS THAN 9 PER CENT OF A SINGLE DISK. A 1443 PRINTER MAY BE USED FOR OUTPUT IF AVAILABLE BUT IS NOT REQUIRED FOR THE USE OF THE SYSTEM. PROGRAM REQUIRES A 1620 CARD-DISK FILE SYSTEM WITH TAG, TNF, MF, AUTO DIVIDE AND INDIRECT ADDRESSING. THE ENTIRE SYSTEM HAS BEEN WRITTEN IN AFIT SPS 01.1.029

1620-05.0.035 SUBROUTINE CROUT FOR
SOLUTION OF SIMULTANEOUS LINEAR EQUATIONS /CARD/
AVAILABLE 1ST QUARTER 1965.
SPECIFY FILE NUMBER 1620-05.0.035

AUTHOR...MRS. JOYCE FUDOR

DIRECT INQUIRIES TO..
PROF. C.H. DAVIDSON
DIRECTOR
ENGINEERING COMPUTING LABORATORY
UNIVERSITY OF WISCONSIN
MADISON, WISCONSIN 53706

THIS SUBROUTINE WILL SOLVE A SYSTEM OF SIMULTANEOUS LINEAR EQUATIONS USING THE CROUT METHOD. THE NUMBER OF EQUATIONS IS LIMITED ONLY BY THE SIZE OF THE MAIN PROGRAM AND THE CORE AVAILABLE. SPECIFICATIONS- IBM 1620 ABLE TO USE FORTRAN II. PROGRAMMING TYPE- FORTRAN II SUBROUTINE.

1620-06.0.043 MULTIPLE REGRESSION PACKAGE
/CARD/
AVAILABLE 1ST QUARTER 1964.
SPECIFY FILE NUMBER 1620-06.0.043

AUTHOR...OTTO DYKSTRA, JR.
GENERAL FOODS RESEARCH CENTER
55 SOUTH BROADWAY
TARRYTOWN, NEW YORK

DIRECT INQUIRIES TO AUTHOR

A COMPLETE MULTIPLE REGRESSION PACKAGE WITH CONVENIENT INPUT-OUTPUT FORMATS. ONE PROGRAM WILL MULTI-PROCESS AS MANY AS 18 VARIABLES WITHOUT INTERMEDIATE OUTPUT. A THREE-PART PROGRAM WILL HANDLE UP TO 44 VARIABLES. AN AUXILIARY PROGRAM WILL GIVE PREDICTIONS AND OBTAIN RESIDUALS FOR THE ACTUAL DATA AND PREDICTIONS FOR SUPPLEMENTARY COMBINATIONS. A FOLLOW-UP PROGRAM WILL PREPARE VARIOUS PLOTS OF THE RESIDUALS. MEMORY 20K.

*** THIS PROGRAM HAS BEEN CERTIFIED BY COMMON.

1620-06.0.158 BASIC STATISTICS
/FUNDAMENTAL ANALYSIS OF FINITE SERIES OR SAMPLE DATA /CARD/
AVAILABLE 3RD QUARTER 1964.
SPECIFY FILE NUMBER 1620-06.0.158

AUTHOR...H.J. HIGHLAND
DIRECTOR OF COMPUTER LABORATORY
LONG ISLAND UNIVERSITY
BROOKLYN 1, N.Y.

DIRECT INQUIRIES TO AUTHOR

BASIC STATISTICAL ANALYSIS OF DATA IN SCIENTIFIC, EDUCATIONAL, PSYCHOLOGICAL, INDUSTRIAL ENGINEERING, TESTING AND BUSINESS RESEARCH. PROGRAM PROVIDES 25 DISPARATE MEASURES OF CENTRAL TENDENCY, DISPERSION, NORMALITY AS WELL AS FUNDAMENTAL SUMMATIONS FOR A SERIES OF DATA. BIFURCATED ANALYSIS PERMITS USE OF DIFFERENT FORMULAS FOR TREATMENT OF FINITE SERIES AS COMPARED WITH SAMPLE DATA. STANDARD STATISTICAL FORMULAS ARE USED THROUGHOUT FOR ANALYSIS OF RAW DATA. UP TO 999,999 INDIVIDUAL 10-DIGIT /2 DECIMAL PLACE/ VALUES MAY BE PROCESSED BY THE PROGRAM. STORAGE USED BY PROGRAM- PROGRAM STORAGE FROM 18439 TO 20000. EQUIPMENT REQUIRED BY PROGRAM- REQUIRES A 407 OR OTHER CARD LISTING UNIT. CAN BE USED WITH MINIMUM 1620 WITH 1622 CARD SYSTEM. PROGRAMMING TYPE- FORTRAN WITH FORMAT, MAINLINE, COMPLETE. LANGUAGE USED IN THE WRITEUP- FORTRAN. ALTHOUGH THE PROGRAM IS WRITTEN TO ACCEPT RAW DATA IN AN F10.2 FORMAT, MOST SIGNIFICANT OUTPUT DATA ARE IN F 12.4 FORMAT AND SUMMATIONS /X, X SQUARED, X CUBED, AND X TO THE 4TH POWER/ ARE IN E-FORMAT. IF GREATER NUMBER OF DECIMAL VALUES ARE REQUIRED FOR RAW DATA INPUT OR IN OUTPUT, PROGRAM WILL OPERATE WITH CHANGE OF FORMAT STATEMENTS ONLY.

1620-06.0.183 TIPS- TENNESSEE INTERPRETIVE
PROGRAM FOR STATISTICS /CARD/
AVAILABLE 1ST QUARTER 1966.
SPECIFY FILE NUMBER 1620-06.0.183

AUTHORS..MRS. A. MCEACHRAN DR. C.W. SHEPPARD

DIRECT INQUIRIES TO..
MRS. A. MCEACHRAN
UNIVERSITY OF TENNESSEE
MEDICAL UNITS COMPUTER CENTER
26 SO. DUNLAP
MEMPHIS, TENN.

TIPS WAS WRITTEN FOR STATISTICAL ANALYSES WHICH WERE TOO SMALL FOR THE EXISTING LARGER PROGRAMS AND TOO LARGE TO BE DONE QUICKLY AT THE DESK CALCULATOR. WITH THIS SYSTEM, THE 1620 IS CONVERTED INTO A SPECIAL-PURPOSE STATISTICAL COMPUTER WHICH USES AN INTERPRETIVE PROGRAMMING LANGUAGE AND SYSTEM OF ADDRESS CODES. BY LEARNING THE SIMPLE INTERPRETIVE LANGUAGE AND BY EXTERNAL PROGRAMMING, TIPS CAN OBTAIN A VARIETY OF STATISTICAL INFORMATION /E.G., MEANS, SUMS OF SQUARES OF DEVIATIONS FROM THE MEAN, COEFFICIENTS OF CORRELATION, AND THIRD AND FOURTH MOMENTS/ AS WELL AS PERFORM LINEAR AND NON-LINEAR REGRESSIONS, A T TEST ON TWO MEANS, AND ONE-WAY AND TWO-WAY ANALYSES OF VARIANCE. THE PROGRAM HANDLES GROUPS OF DATA AS VECTORS AND HAS FREE-FORM FLOATING POINT INPUT AND OUTPUT.
A 20K 1620 CARD SYSTEM WITH AUTODIVIDE IS REQUIRED BY PROGRAM. THE REQUIREMENT FOR AUTODIVIDE CAN BE EASILY REMOVED. WRITTEN IN SPS.
THE SOURCE DECKS ARE OPTIONAL MATERIAL AND MUST BE SPECIFICALLY REQUESTED ON THE ORDER CARD.

1620-06.0.216 STATISTICAL PROJECT
ORGANIZER FOR TEACHING /CARD/
AVAILABLE 4TH QUARTER 1966.
SPECIFY FILE NUMBER 1620-06.0.216

AUTHOR...DR. H.J. HIGHLAND

DIRECT INQUIRIES TO..
DR. H.J. HIGHLAND, DIRECTOR,COMPUTER LABORATORY,
THE BROOKLYN CENTER, LONG ISLAND UNIV., BROOKLYN, N.Y.

SPOT - A STATISTICAL PROJECT ORGANIZER FOR TEACHING - CONSISTS OF TWO INDEPENDENT BUT INTERRELATED PROGRAMS- /A/ FREQUENCY ARRAY GENERATOR, AND /B/ BASIC STATISTICS FOR FREQUENCY ARRAY, AND IS PART OF THE CO/STATS /COMPUTER ORIENTED/STATISTICAL TEACHING AND TESTING SERIES/ PROGRAMS. THE TWO PROGRAMS ARE AN AID TO TEACHERS WHO WISH TO PROVIDE STUDENTS WITH INDIVIDUALIZED RAW DATA FOR ANALYSIS FOR STUDENT STATISTICS PROJECTS. THE FIRST PROGRAM CONSISTS OF TWO PHASES UNDER SWITCH CONTROL. PHASE I, USING STURGES RULE, WILL DETERMINE THE NUMBER OF CLASS INTERVALS AND SIZE OF THE CLASS INTERVAL FOR THE SERIES. PHASE II WILL TRANSFORM RAW DATA INTO A STATISTICAL ARRAY USING CLASS INTERVAL SIZE AND NUMBER OF CLASS INTERVALS AT THE USERS OPTION. THE SECOND PROGRAM WILL PRODUCE THE CONVENTIONAL FREQUENCY ARRAY LISTING CLASS INTERVALS, FREQUENCY, D, D SQUARED, FD AND FD SQUARED USED IN COMPUTATIONS. IT WILL ALSO PROVIDE THE USER WITH VARIOUS MEASURES OF CENTRAL TENDENCY AND DISPERSION AMONG WHICH ARE- MEAN, ADJUSTED MODE, MEDIAN, Q SUB 1, Q SUB 3 QUARTILE DEVIATION, STANDARD DEVIATION, COEFFICIENT OF VARIATION, COEFFICIENT OF SKEWNESS /BOTH PEARSON AND QUARTILE METHODS/ AND K, THE CENTER VALUE OF A NORMAL DISTRIBUTION. ALSO INCLUDED IS A SAMPLE STUDENT STATISTICAL PROJECT FORM FOR USE WITH THESE PROGRAMS. PROGRAMS HAVE BEEN WRITTEN FOR A BASIC 20K IBM 1620 MODEL I WITHOUT ANY SPECIAL FEATURES. CARD INPUT/OUTPUT REQUIRED, AS WELL AS A CARD PRINTER, SUCH AS THE IBM 407.

1620-07.0.019 THREE-IN-ONE CURVE FITS
HYPERBOLIC, EXPONENTIAL, POWER FUNCTIONS /CARD/
AVAILABLE 3RD QUARTER 1962.
SPECIFY FILE NUMBER 1620-07.0.019

AUTHOR...WADE A. NORTON

DIRECT INQUIRIES TO..
MR. E. J. ORTH, JR.
SOUTHERN SERVICES, INC.
P.O. BOX 2641
BIRMINGHAM, ALABAMA

THIS PROGRAM, WHICH IS REALLY THREE PROGRAMS IN ONE, COMPUTES THE PARAMETERS FOR THEORETICAL CURVE FITS TO EMPIRICAL DATA. PLOT BACK, INTERPOLATION, STANDARD ERROR AND FUNCTION EVALUATION ARE EACH OPTIONAL UNDER CS CONTROL. STANDARD ERROR FEATURE PERMITS COMPARISON WITH POLYNOMIAL FITS DONE BY PROGRAM 7.0.002 AND FITS BY OTHER SUBPROGRAMS OF THIS ONE. FOR FUNCTION EVALUATION, PARAMETERS ARE KEYED-IN AT CONSOLE TYPEWRITER. CARD I/O, 20K 1620 WITHOUT SPECIAL FEATURES.

*** THIS PROGRAM HAS BEEN CERTIFIED BY COMMON.

1620-09.7.009 1620 MULTICURVE PLOTTING
PROGRAM /CARD/
AVAILABLE 1ST QUARTER 1963.
SPECIFY FILE NUMBER 1620-09.7.009

AUTHORS..JACK BURGESON JAMES SNEDIKER

DIRECT INQUIRIES TO..
J.W. BURGESON,IBM CORP.,ROOM 708,THE ILLUMINATING BLDG.,
CLEVELAND, OHIO

PROGRAM ACCEPTS DATA FROM CARDS AND WILL PLOT UP TO TEN DEPENDENT VARIABLES VERSUS A COMMON INDEPENDENT VARIABLE. EACH DEPENDENT VARIABLE MAY HAVE A SEPARATE AXIS SCALE. OUTPUT IS ON TYPEWRITER, CARDS, PAPER TAPE, OR ANY COMBINATION OF THESE. CARD 1620, 20K MEMORY, NO ADDITIONAL FEATURES. PROGRAM IS WRITTEN IN FORTRAN AND SPS.

1620-10.1.005 TRANSPORTATION PROGRAM FOR
THE IBM 1620 /CARD/
AVAILABLE 1ST QUARTER 1964.
SPECIFY FILE NUMBER 1620-10.1.005

AUTHOR...J. N. ROLES
UNIVERSITY OF CALIFORNIA
207 GIANNINI HALL
BERKELEY 4, CALIFORNIA

DIRECT INQUIRIES TO AUTHOR

THIS PROGRAM IS A SIMPLE ADAPTATION OF THE TRANSPORTATION PROGRAM FOR THE IBM 1620 /TAPE/ BY MADDEN AND SMITH, FILE NO. 10.1.003. IT PROVIDES AN OPTIMAL SOLUTION TO THE LINEAR PROGRAMMING TRANSPORTATION PROBLEM. MEMORY 20K, 1622 CARD-READ-PUNCH

*** THIS PROGRAM HAS BEEN CERTIFIED BY COMMON.

1620-11.0.023 CARD SYSTEM DEMONSTRATION
AVAILABLE 2ND QUARTER 1963.
SPECIFY FILE NUMBER 1620-11.0.023

AUTHOR...CARL F. FINK
IBM CORP.
340 W. WASHINGTON AVE.
MADISON, WISCONSIN

DIRECT INQUIRIES TO AUTHOR

TO DEMONSTRATE THE 1620 CARD SYSTEMS ARITHMETIC, READING, PUNCHING AND TYPING ABILITIES. 1620 WITH CARD INPUT/OUTPUT, NO SPECIAL FEATURES, ANY CORE SIZE. THE PROGRAM IS WRITTEN IN MACHINE LANGUAGE.

1620-11.0.029 DEMOPAK-A FUNCTIONAL DEMONSTRATION OF THE IBM 1620 /CARD/
AVAILABLE 3RD QUARTER 1963.
SPECIFY FILE NUMBER 1620-11.0.029

AUTHORS..RAY PECK W. J. OLMO DAVE MONTGOMERY

DIRECT INQUIRIES TO..
W. J. OLMO
IBM CORP.
340 MARKET STREET
SAN FRANCISCO, CALIF.

DEMOPAK IS DESIGNED TO BE USED AS AN INTRODUCTORY DEMONSTRATION OF VARIOUS 1620 FUNCTIONS AND TO PROVIDE A BACKGROUND FOR THE DEMONSTRATION OF PROGRAMMING SYSTEMS AND PRODUCTION PROGRAMS. THE PROGRAM IS DIVIDED INTO SECTIONS, EACH OF WHICH IS A COMPLETE PROGRAM WHICH WILL TYPE A HEADING AND, IF PROGRAM SWITCH 3 IS ON, A SET OF OPERATING INSTRUCTIONS. THE SECTION THEN COMES TO A HALT AND IS EXECUTED BY DEPRESSING START. UPON COMPLETION THE PROGRAM RETURNS TO THE INITIAL HALT AND IS READY TO BE EXECUTED AGAIN. PROGRAM SWITCHES 1 AND 2 AREUSED IN SOME SECTIONS TO STOP THE OPERATIONS, ALTER LOGIC, OR PROVIDE OPTIONAL INPUT OR OUTPUT. WHEN THE DEMONSTRATION OF ANY SECTION IS COMPLETE THE OPERATOR DEPRESSES RESET, INSERT, RELEASE, AND START TO PROCEED TO THE

1620-11.0.031 MUSIC PROGRAM /CARD/
AVAILABLE 3RD QUARTER 1964.
SPECIFY FILE NUMBER 1620-11.0.031

AUTHOR...LAURA B. STEELE
GENERAL MOTORS INSTITUTE
FLINT, MICHIGAN

DIRECT INQUIRIES TO AUTHOR

DEMONSTRATION PROGRAM TO PLAY MUSIC ON THE 1620 USING A RADIO SPEAKER FOR OUTPUT. USES A TONE SUBROUTINE PLUS CONSTANTS SPECIFYING ADDRESS OF FIELD FOR EACH NOTE AND DURATION OF A NOTE. A. STORAGE USED BY PROGRAM-ACTUAL SUBROUTINE AND CONSTANTS LOCATED FROM 19500-19999. EACH TUNE TAKES VARYING STORAGE ACCORDING TO NUMBER OF NOTES TO BE PLAYED. B. EQUIPMENT REQUIRED BY PROGRAM - CARD SYSTEM, /MODEL I/, INDIRECT ADDRESSING, CAN BE USED ON ANY MACHINE BY CHANGING THE ONE INSTRUCTION WHICH USES INDIRECT ADDRESSING. COULD BE COMPILED ON TAPE AS WELL AS CARDS. LANGUAGE USED IN THE WRITEUP- SPS 1620/1710.

*** THIS PROGRAM HAS BEEN CERTIFIED BY COMMON.

1620-11.0.032 BBC BASEBALL SIMULATION AND DEMONSTRATOR /CARD/
AVAILABLE 1ST QUARTER 1964.
SPECIFY FILE NUMBER 1620-11.0.032

AUTHOR...P.R. BURGESON
4257 OAK KNOLL AVENUE
YOUNGSTOWN 12, OHIO

DIRECT INQUIRIES TO AUTHOR

THIS PROGRAM DEMONSTRATES THE SIMULATION ABILITY OF THE 1620 BY-QUOTE - PLAYING - UNQUOTE - THE GAME OF BASEBALL BETWEEN TWO ALL STAR TEAMS. THE MACHINE OPERATOR PICKS THE VISITING TEAM FROM A MASTER ROSTER OF 90 POSSIBLE PLAYERS, THE 1620 SELECTS /RANDOMLY/ THE HOME TEAM FROM THE 81 PLAYERS LEFT. A COMPLETE BALL GAME IS THEN SIMULATED. TWO SEPARATE RANDOM NUMBER GENERATORS WHICH ARE INITIALIZED AT THE PROGRAM START ASSURE A DIFFERENT GAME EACH TIME. SIMULATION OF THE RULES OF BASEBALL. RESTRICTIONS- THE PROGRAM RECOGNIZES ONLY 90 PARTICULAR PLAYER NAMES AND RECORDS, CONTAINED IN A MASTER TABLE WITH THE PROGRAM. OTHER PLAYER NAMES ARE NOT ACCEPTABLE. EQUIPMENT- BASIC 20K 1620 WITH 1622 CARD READER. NO OTHER FEATURES ARE USED, ALTHOUGH THEIR PRESENCE DOES NOT INHIBIT PROGRAM OPERATION. STORAGE- ALL OF 20K MEMORY IS USED. LANGUAGE- THIS PROGRAM WAS CODED IN MACHINE LANGUAGE. NO PROGRAMMING LANGUAGE WAS EMPLOYED.

1620-02.0.052 COMPUTEST /CARD/
AVAILABLE 4TH QUARTER 1965.
SPECIFY FILE NUMBER 1620-02.0.052

AUTHOR...J.A. STARKWEATHER
J.A. STARKWEATHER,COMPUTER CENTER,UNIV. OF CALIF.,
SAN FRANCISCO, CALIF. 94122

COMPUTEST IS A PROBLEM-ORIENTED PROGRAMMING LANGUAGE FOR COMPUTER-ASSISTED INSTRUCTION, TESTING AND INTERVIEWING. SEQUENCES OF INSTRUCTIONAL MATERIAL AND TEST QUESTIONS MAY BE WRITTEN IN NATURAL LANGUAGE AND A VARIETY OF CUES MAY BE USED FOR THE RECOGNITION OF A RIGHT ANSWER FROM TYPEWRITER INPUT. VARIABLE COMMENTS AND CHOICE OF THE NEXT QUESTION TO BE ASKED MAY BE DETERMINED BY THE EVALUATION OF AN ANSWER. SCORING AND DATA COLLECTION IS OPTIONAL FOR EACH QUESTION. COMPUTEST ACTS AS AN INTERPRETIVE TRANSLATOR OF PROGRAM MATERIAL WHICH IS PRESENT IN THE CARD READER AND READ IN THE COURSE OF TYPEWRITER INTERACTION WITH A SUBJECT. STORAGE USED BY PROGRAM-00402-19999. /AREA 17998-19999 IS USED AS INPUT BUFFER, SO UPPER CORE MAY BE AVAILABLE IF INPUT IS RESTRICTED/. EQUIPMENT REQUIRED- CARD, AUTO DIVIDE, INDIRECT ADDRESSING, 1620 MODEL I. PROGRAM CANNOT BE USED ON LESSER MACHINE. PROGRAMMED IN-1620/1710 SPS.
TYPEWRITER INPUT IS LIMITED TO A MAXIMUM OF 477 CHARACTERS INCLUDING BLANKS.

1620-06.0.135 ITEM ANALYSIS AND SCORING /CARD/
AVAILABLE 1ST QUARTER 1964.
SPECIFY FILE NUMBER 1620-06.0.135

AUTHOR...HOWARD GIVNER
OFFICE OF TESTING AND RESEARCH
BROOKLYN COLLEGE
BEDFORD AV. + AV. H,
BROOKLYN 10, N.Y.

DIRECT INQUIRIES TO AUTHOR

THIS PROGRAM IS DESIGNED TO DO AN ITEM ANALYSIS AND/OR SCORING OF CARDS PREPARED BY MARK SENSE PUNCHING. A FORMAT SPECIFICATION PROVIDES FLEXIBILITY OF INPUT FOR UP TO 200 ITEMS AND UP TO 999 STUDENTS. THE PROGRAM FURNISHES A SERIAL NUMBER FOR EACH STUDENT IN THE LISTING OF SCORES. THIS PROGRAM FINDS THE DIFFICULTY D OF EACH ITEM BY DETERMINING THE PROPORTION OF STUDENTS ANSWERING IT CORRECTLY, AND THE VALIDITY V OF EACH ITEM BY CALCULATING THE POINT-BISERIAL COEFFICIENT. THE MEAN SCORE AND STANDARD DEVIATION OF THE TOTAL GROUP IS ALSO FOUND. EQUIPMENT REQUIRED BY PROGRAM- CARD SYSTEM. PROGRAM CAN BE USED ON LESSER MACHINE. SPECIFY WHICH REQUIREMENTS CAN BE EASILY REMOVED. PROGRAMMING TYPE FORTRAN WITH FORMAT, SPS - 1620/1710, MAINLINE, COMPLETE. MOST OF THE PROGRAM WAS WRITTEN IN FORTRAN. AFTER COMPILATION WITH A FORTRAN/FORMAT PROCESSOR, THE I/O SUBROUTINES IN THE OBJECT DECK WERE MODIFIED. OTHER CHANGES WERE MADE TO ALLOW LISTS TO BE READ. A FORMAT DECODING PROCESSOR ROUTINE WRITTEN IN SPS WAS ADDED TO COMPLETE THE PROGRAM. THIS PROGRAM CAN DISTINGUISH BLANKS, ZEROS AND TWELVE PUNCHES FROM EACH OTHER.

1620-13.0.003 NORTHEASTERN UNIVERSITY TEST SCORING PROGRAM /CARD/
AVAILABLE 1ST QUARTER 1964.
SPECIFY FILE NUMBER 1620-13.0.003

AUTHOR...ROBERT M. OBRIEN
COMPUTATION CENTER
NORTHEASTERN UNIVERSITY
360 HUNTINGTON AVE.
BOSTON 15, MASS.

DIRECT INQUIRIES TO AUTHOR

TO GRADE MULTIPLE CHOICE OBJECTIVE EXAMS TAKEN ON MARK SENSE CARDS AND PUBLISH, IN ADDITION TO THE GRADE FOR EACH STUDENT, A GRADE DISTRIBUTION /WITH MEAN AND STANDARD DEVIATION/ AND AN ANALYSIS OF THE EXAM INDICATING HOW MANY CHOSE EACH CHOICE FOR EACH QUESTION AND THE PERCENT OF CORRECT ANSWERS FOR EACH QUESTION. MAXIMUM OF 150 5-CHOICE QUESTIONS PER EXAM.

STORAGE REQUIREMENTS- 19563 LOCATIONS
MEMORY 20K, 1622, NO SPECIAL FEATURES REQUIRED.
SEE ADDITIONAL REMARKS. SPS LANGUAGE. FIXED POINT.
NOT RELOCATABLE.

*** THIS PROGRAM HAS BEEN CERTIFIED BY COMMON.

1620-13.0.011 SCRAMBLE - COMPUTER PREPARATION OF MULTIPLE FORMS OF AN EXAMINATION /CARD/
AVAILABLE 3RD QUARTER 1964.
SPECIFY FILE NUMBER 1620-13.0.011

AUTHOR...DR. H.J. HIGHLAND
DIRECTOR OF COMPUTER LABORATORY
LONG ISLAND UNIVERSITY
BROOKLYN 1, N.Y.

DIRECT INQUIRIES TO AUTHOR

PROGRAM DESIGNED TO MEET NEEDS OF SCHOOLS WHEREIN LARGE CLASSES DICTATE NEED FOR MULTIPLE COPIES OF THE SAME EXAMINATION. PROGRAM WILL PRODUCE THESE MULTIPLE COPIES BY REARRANGING THE SEQUENCE OF THE EXAMINATION QUESTIONS. PUNCHED CARD OUTPUT IS USED TO PRODUCE /A/ TEACHER/S COPY WITH ANSWER KEY PRINTED NEXT TO EACH QUESTION AND /B/ OFFSET MASTER STENCIL USED FOR PRINTING, BOTH OF THESE CAN BE PRODUCED ON A 407 OR SIMILAR MODEL PRINTER. PROGRAM WILL HANDLE UP TO 50 QUESTIONS, EACH OF WHICH CAN BE UP TO 99 CARDS LONG. STORAGE USED BY PROGRAM-19059 TO 20000. EQUIPMENT REQUIRED BY PROGRAM- CARD SYSTEM. PROGRAMMING TYPE- FORTRAN WITH FORMAT.

1620-13.0.012 EXAMINATION ASSEMBLY PROGRAM /CARD/
AVAILABLE 4TH QUARTER 1964.
SPECIFY FILE NUMBER 1620-13.0.012

AUTHOR...H.B. KERR
DIRECTOR
COMPUTER CENTER
TENNESSEE POLYTECHNIC INSTITUTE
BOX 21A TENN. TECH.
COOKEVILLE, TENN.

DIRECT INQUIRIES TO AUTHOR

THIS PROGRAM PACKAGE CAUSES AN EXAMINATION TO BE MADE UP BY PULLING EXAMINATION QUESTIONS FROM A POOL OF QUESTIONS PUNCHED INTO CARDS. A NEW SEQUENCE NUMBER IS GIVEN TO THE QUESTIONS ,,PULLED,, AND THE EXAMINATION MAY THEN BE LISTED ON DITTO MASTER ON THE 407 ACCOUNTING MACHINE. A MAXIMUM OF 999 QUESTIONS MAY BE CONTAINED IN THE ,,POOL,, OF QUESTIONS. THE QUESTIONS MAY BE OF VARIABLE LENGTH WITH NO LIMITATION PLACED UPON THE MAXIMUM NUMBER OF CARDS ALLOWED. EQUIPMENT SPECIFICATIONS- 20K, INDIRECT ADDRESSING, TNS, TNF. SOURCE LANGUAGE- SPS.

1620-13.0.013 FORTRAN TEACH /CARD/
AVAILABLE 1ST QUARTER 1965.
SPECIFY FILE NUMBER 1620-13.0.013

AUTHOR...PROF. W.L. POPE
COMPUTER CENTER
UTAH STATE UNIV.
LOGAN, UTAH 84321

DIRECT INQUIRIES TO AUTHOR

FORTRAN TEACH IS A SET OF PROGRAMS AND STUDENT PROBLEMS DESIGNED FOR USE DURING THE EARLY PART OF A COURSE TEACHING FORTRAN PROGRAMMING. THEY ENABLE THE STUDENT TO GET SOMETHING ON THE MACHINE BEFORE HE CAN WRITE A COMPLETE PROGRAM, AND THEY ASSIST THE INSTRUCTOR IN CHECKING THE PROBLEMS. EQUIPMENT REQUIRED BY PROGRAM- CARD SYSTEM, 40K CORE MINIMUM. THESE PROGRAMS ARE WRITTEN IN FORTRAN FOR THE FORGO PROCESSOR. PROGRAMMING TYPE- MAINLINE, COMPLETE. PROGRAMMING LANGUAGE-FORGO.

ROCHESTER INSTITUTE OF TECHNOLOGY
COMPUTER CENTER

```
C     CROUT REDUCTION METHOD FOR SIMULTANEOUS LINEAR SYSTEMS
C
C
      DIMENSION A(15,15),B(15),Z(15,15),X(15)
   18 READ 100  N
  107 DO 102 I=1,N
      READ 101,(A(I,J),J=1,N),B(I)
  102 PRINT101,(A(I,J),J=1,N),B(I)
      PRINT 900
      DO 108 I=1,N
      DO 108 J=1,N
  108 Z(I,J)=A(I,J)
      NM1=N-1
      DO 2 J=1,N
      JP1=J+1
      JM1=J-1
      DO 6 I=J,N
      ASUM=0.
      IF (JM1)6,6,7
    7 DO 9 K=1,JM1
    9 ASUM=ASUM+A(I,K)*A(K,J)
    6 A(I,J)=A(I,J)-ASUM
      AMAX=A(J,J)
      IMAX=J
      IF (JP1-N)20,20,21
   20 DO 1 I=JP1,N
      IF(ABSF(AMAX)-ABSF(A(I,J)))3,1,1
    3 AMAX=A(I,J)
      IMAX=I
    1 CONTINUE
   21 IF(ABSF(AMAX)-1.E-30)504,504,4
    4 DO 5 K=1,N
      ASAVE=A(IMAX,K)
      A(IMAX,K)=A(J,K)
    5 A(J,K)=ASAVE
      ASAVE=B(IMAX)
      B(IMAX)=B(J)
      B(J)=ASAVE
       I1=J
      J1=JP1
      IF (JP1-N)22,22,23
   22 DO 8 J2=JP1,N
      ASUM=0.
      IF (JM1)8,8,11
   11 DO 12 K=1,JM1
   12 ASUM=ASUM+A(I1,K)*A(K,J2)
    8 A(I1,J2)=(A(I1,J2)-ASUM)/A(I1,I1)
   23 ASUM=0.
      IF (JM1)2,2,13
   13 DO 14 K=1,JM1
   14 ASUM=ASUM+A(I1,K)*B(K)
    2 B(I1)=(B(I1)-ASUM)/A(I1,I1)
```

Do Column Entries

Find AMAX

Exchange Rows (if needed)

Do Row Entries

Crout Reduction

```
      DO 15 J=1,N
      I1=N-J+1
      I=I1+1
      ASUM=0.
      IF (I1-N)17,15,15
   17 DO 16 K=I,N
   16 ASUM=ASUM+A(I1,K)*X(K)
   15 X(I1)=B(I1)-ASUM
  501 DET=1.0
      DO 502 I=1,N
      DO 499 J=1,N
  499 PRINT 498,I,J,Z(I,J),A(I,J)
  502 DET=DET*A(I,I)
      PRINT 503,DET
      IF (ABS(DET)-.0001)504,504,120
  120 DO 103 I=1,N
  103 PRINT 104,I,X(I)
      PUNCH 101,(X(I),I=1,N)
      PRINT 900
      DO 109 I=1,N
      SUM=0.
      DO 110 J=1,N
  110 SUM=SUM+Z(I,J)*X(J)
  109 PRINT 111,I,SUM
      PRINT 900
      GO TO 106
  504 PRINT 105
      GOTO 120
  106 PRINT 200
  100 FORMAT(I2)
  101 FORMAT(5E15.8)
  104 FORMAT(12H            X(,I2,5H)=    ,E14.8)
  105 FORMAT(/2X,22HTHE MATRIX IS SINGULAR/)
  111 FORMAT(3X,9HCONSTANT(,I2,5H)=    E14.8)
  200 FORMAT (/2X,27HTHIS IS THE END OF THE DATA)
  498 FORMAT(2I5,2E20.8)
  503 FORMAT(/5X,14H DETERMINANT =,E15.8//)
  900 FORMAT (//)
      END
```

Back Substitution

Determinant

Check

DATA

N = NUMBER OF EQUATIONS (MAX. 15)
A(I,J) = ELEMENTS OF COEFFICIENT MATRIX
B(I) = ELEMENTS OF CONSTANT VECTOR

NOTE: THIS IS A MODIFICATION OF PROGRAM NO. 5.0.035 FROM THE IBM 1620 PROGRAM LIBRARY.

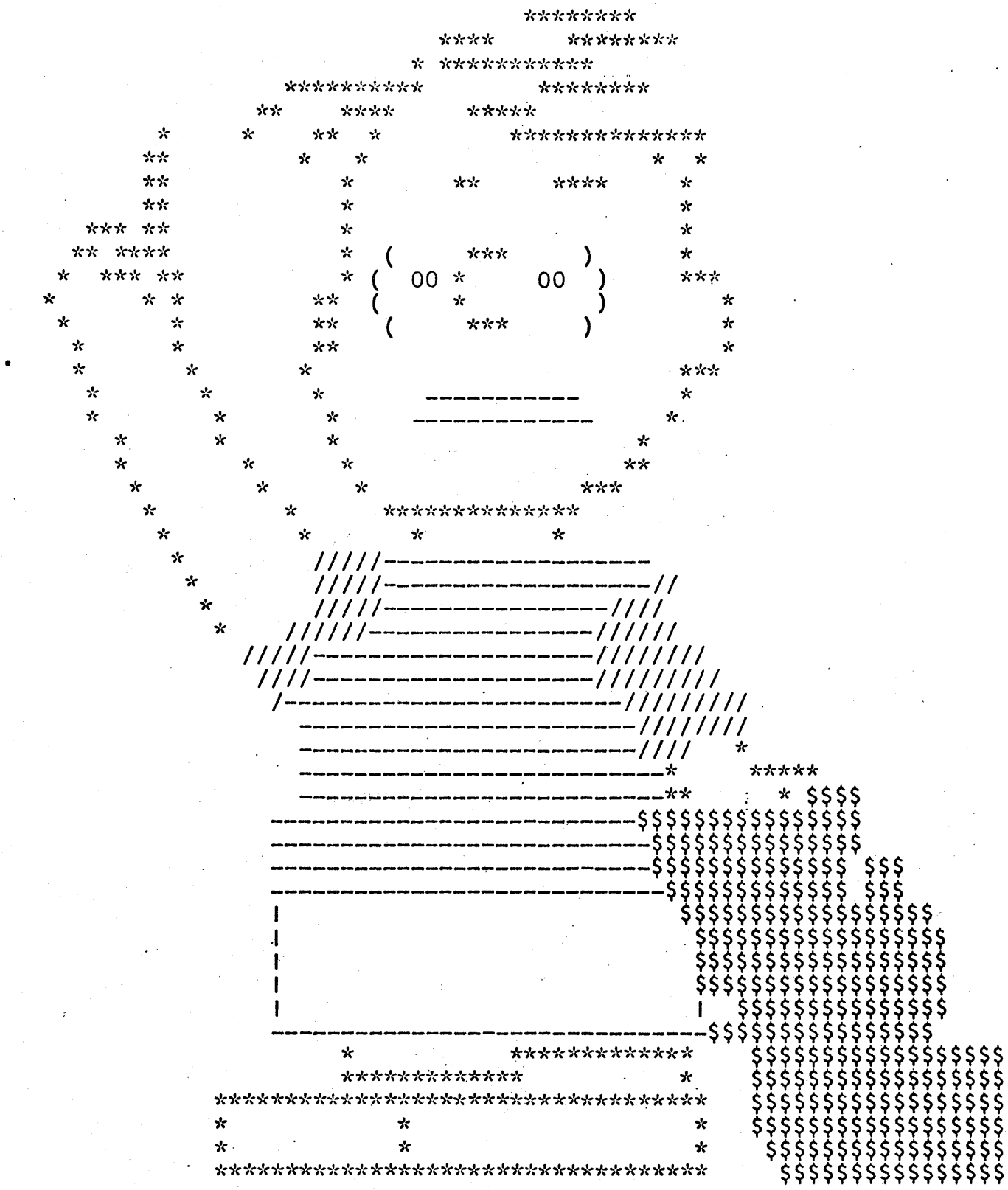
ROCHESTER INSTITUTE OF TECHNOLOGY
COMPUTER CENTER

```
C     GAUSS-SEIDEL ITERATION FOR SIMULTANEOUS LINEAR SYSTEMS
C
C
C     EQUATIONS MUST BE IN PROPER ORDER
      DIMENSION A(15,15), B(15), D(15), S(15), X(15), Y(15)
   11 READ 100,N,(X(I),I=1,N)
      PRINT100,N,(X(I),I=1,N)
      NI = 1
      LIM = 100
    1 DO 2 I = 1,N
      READ 101,(A(I,J),J=1,N), B(I)
      PRINT101,(A(I,J),J=1,N), B(I)
      D(I) = A(I,I)
    2 A(I,I) = 0.
      PRINT 900
    4 ER = 0.
      PRINT 900
      DO 5 I = 1,N
      IF(ISSW(1)-1)13,13,5
   13 PRINT 110,NI,I,X(I)
    5 Y(I) = X(I)
   14 DO 7 I = 1,N
      S(I) = 0.
      DO 6 J = 1,N
    6 S(I) = S(I) + A(I,J)*X(J)
      X(I) = (B(I) - S(I))/D(I)
    7 ER = ER + ABS(X(I)-Y(I))
      IF (ER-0.0000001) 9,9,8
    8 IF(NI-LIM) 10,9,9
   10 NI = NI + 1
      GO TO 4
    9 PRINT113, NI, (X(I),I=1,N)
      LIM = LIM + 100
      PAUSE
      IF(ISSW(1)-1)4,4,12
   12 IF(ISSW(9)-1)15,15,11
  100 FORMAT (I2, 15F5.0)
  101 FORMAT (5E15.8)
  110 FORMAT (2I5,5X,E15.8)
  113 FORMAT (I5,7E15.8/8E15.8)
  900 FORMAT (//)
   15 END
```

DATA

N = NUMBER OF EQUATIONS (MAX. 15)
X(I) = INITIAL GUESSES AS TO SOLUTION
A(I,J) = ELEMENTS OF COEFFICIENT MATRIX
B(I) = ELEMENTS OF CONSTANT VECTOR

LINUS - OUR HERO



NO PROBLEM IS TOO BIG OR TOO
COMPLICATED TO BE RUN AWAY FROM

COMPLIMENTS OF R. I. T. COMPUTER CENTER

1620 H.S. PROJECT

THE PRÉ-COMPILER

by

Frederick R. Henderson
Director, Computer Center
Rochester Institute of Technology
Rochester, New York 14623
(1620 User #1393)

ABSTRACT

Before a Fortran program can be compiled, any grammar and syntax errors must be corrected. The 20K memory of the basic IBM 1620 computer is not large enough to contain both the compiler itself and the related diagnostics needed to catch these programming errors. Consequently separate pre-compilers have been written to do this preliminary error checking. The R.I.T. Pre-Compiler was presented at the 1620 Users Group Meeting in New York in 1965 and is still available from the author. However, other programs such as the NCE Load and Go Fortran and the C4D Operating System have largely superseded it.

Presented at
COMMON Meeting
Philadelphia, Pa.
September, 1968
Session T 5 4 G

THE PRE-COMPILER

This discussion is about pre-compilers in general with only incidental reference to the R.I.T. Pre-Compiler. I have, therefore, taken the liberty of eliminating "R.I.T." from the announced title of this presentation.

One of the questions I asked when we first acquired our IBM 1620 Computer in 1963 was, "What is a pre-compiler, and why do we need it?" Our friendly IBM representative explained that it was a program to check the grammar and syntax of our Fortran statements and to pin-point these errors so that they could be corrected before trying actually to compile our programs. We needed it because well over half of all errors made by beginning students fall into this category. (The other errors are matters of logic which are detected when we get wrong answers or no answers at all after compilation.)

Our initial installation included only the 20K 1620 central processing unit and the 1622 card read punch -- no disk or printer. This meant that we could use only the Fortran with Format compiler which had to be physically loaded before compiling each Fortran source program. By using the IBM Pre-Compiler, we could batch process a number of student programs at one time, correcting their grammar and syntax errors before compiling them. Not only were the diagnostics much better in the pre-compiler than in the compiler, but the total processing time was much less than it would have been with the compiler alone.

Of course, if additional memory (either in core or on disk) had been available, other programs such as FORGO and DIAGNOSTICIAN could have been utilized, but in 1963, the only load-and-go processor available for a basic 20K system was GOTRAN which most people found rather unsatisfactory because it has almost no diagnostics and is easily destroyed in core by errors in source programs.

By 1965, however, many other processors such as AFIT, UTO, PDQ, and NCE had appeared. We had also added a 1311 disk drive and a 1443 printer to our system and were operating largely under Monitor I using Fortran II-D. We continued to use the pre-compiler, however, because it was appreciably faster than the Fortran II-D compiler, and we modified the Monitor so that the pre-compiler operated under it. The only trouble was that there were many valid statements in Fortran II which the pre-compiler did not recognize.

While attending an NSF Computer Institute at Seton Hall University in June, 1965, I discussed with Richard Gabriel the feasibility of trying to modify the IBM Pre-Compiler so that it would recognize additional valid Fortran II instructions. Dr. Gabriel suggested that a version for PDQ Fortran might also be useful. As a result, we eventually produced three separate but related versions of the R.I.T. Pre-Compiler; one was for PDQ Fortran, one was for Fortran II-D without a printer, and one was for Fortran II-D with a printer.

The R.I.T. Pre-Compiler was presented at the 1620 Users Group Meeting in New York City on October 8, 1965,* and enough interest was evident to suggest that it might be worthwhile to consider submitting the results to the 1620 Program Library. However, at this same meeting, I learned that Frank Maskiell and his associates had presented a new operating system called "C4D" at the previous Users Group meeting in Miami which was reported to be very good indeed.

---

* An abstract of this paper is attached as well as a brief summary sheet which we have supplied to our students in the past.

After learning more about "C4D" and noting that its diagnostic routine is a modified version of the DIAGNOSTICIAN program, I concluded that if we could employ "C4D" for processing student programs, we would have little if any use for the pre-compiler.

We have been operating under "C4D" for over a year, and we cannot recommend it too highly. It is twice as fast as Monitor I, and the diagnostics are very good. Furthermore, in most of its arithmetic operations, it rounds instead of truncating, and in most instances, the round-off errors in the answers are appreciably less than with Monitor I.

However, if your installation has only a 20K memory and no disk file, then I believe that PDQ Fortran is probably your best choice among the Fortran compilers with which I am familiar. If you are using PDQ Fortran in this situation, the R.I.T. Pre-Compiler may still be useful. And if anyone is interested in it at this point, we shall be happy to supply the necessary program deck and documentation for that particular configuration.

At the same time, I do want to call your attention to the NCE Load and Go which I understand Bruce Fowler is presenting today also. I think this may be very useful in many high school installations where most of the programs to be run are short student problems.

But if you have a disk file, be sure to try "C4D".

* * * * * * * *

A word or two about programming languages other than Fortran may be helpful.

Many teachers of business subjects ask why not teach Cobol. My answer is that Cobol is useful for manipulating data files, but it is not a general problem-solving language. Texts dealing with business management <u>problems</u> almost all employ Fortran.

Many schools are installing teletypewriter remote terminals, time-sharing a large computer elsewhere instead of obtaining their own on-campus computer. For these installations, the Dartmouth Basic language is certainly first choice.

The newest language, of course, is PL/1. Whether or not it is going to supersede both Fortran and Cobol remains to be seen. I freely admit that I am not a sophisticated programmer, but I still like Fortran and Basic.

There are in addition to the above-mentioned languages, a great many specialized ones such as COGO, SNOBOL, SIMSCRIPT, and COURSEWRITER. It remains to be seen whether or not these have any place in a high school program; at present it seems doubtful.

THE RIT PRE-COMPILER

by Frederick R. Henderson (# 1393)

ABSTRACT

Most Fortran Compilers for the IBM 1620 lack adequate diagnostics for beginning students, and the use of a Pre-Compiler is recommended. For installations with only 20K storage, the IBM Pre-Compiler is the best that is available, and it works well with the Fortran with Format Compiler.

Many schools with 20K, however, are now using PDQ Fortran, or if they have a 1311 Disk, are using Fortran II-D. Both of these compilers have added language facilities not available in Fortran with Format, and the IBM Pre-Compiler prints out too many spurious error messages when used with PDQ or Fortran II-D. To help our new students in debugging their programs, we have modified the IBM Pre-Compiler to make it more useful with PDQ Fortran and Fortran II-D.

There are presently three card versions of the RIT Pre-Compiler. The first is for use with PDQ Fortran; the second is for Fortran II-D without a printer; the third is for Fortran II-D with a 1443 Printer. These are not completely compatible with their respective compilers, but they are more useful than the IBM Pre-Compiler for beginning students.

SUMMARY OF CHANGES INCORPORATED IN RIT PRE-COMPILER

Changes applicable to PDQ and II-D Fortran

1. One continuation card allowed on I/O and FORMAT statements.
2. Undefined variables in statements like N = N + 1 detected.
3. "A" type FORMAT accepted (also "D" for PDQ).
4. "IF (SENSE SWITCH 9)" accepted.
5. All "C" Comment cards printed regardless of switch settings.

Additional Changes under Monitor I (PR-025)

6. I/O statements containing implied DO-loops accepted.
7. "CALL EXIT" recognized as valid statement.
8. Program called by "‡PCOM" Control Card and on completion of job, branches back to "Moncal".
9. Program switch 1 only used (ON to print all statements).
10. "*" cards ahead of source deck and all cards after "END" ignored.

Further changes under Monitor I with Printer (PR-033A)

11. All output on printer except "‡‡" cards.
12. "*" Fortran Control Cards printed but not checked.

(Presented at 1620 Users Group Meeting in New York, October 8, 1965)

ROCHESTER INSTITUTE OF TECHNOLOGY
COMPUTER CENTER

THE R.I.T. PRE-COMPILER

The IBM 1620 Fortran with Format Pre-Compiler has been extensively modified by us for use with Fortran II-D and the 1443 printer, and it will now process most of the instructions ordinarily used. Since it detects most of the common programming errors, you are urged to use the Pre-Compiler on all programs before submitting them for Monitor Runs. Experience has demonstrated that this results in more efficient and faster processing of all source programs.

Error messages are grouped into seven categories as follows:

ARITH - Errors in arithmetic statements such as missing operation symbols, missing parentheses, and mixed mode.

VAR - Errors in variables and/or subscripts such as undefined variables, improper subscripts, and missing DIMENSION statements.

DO - Errors in DO loops such as wrong indices, missing statement numbers, improper nesting, and missing CONTINUE statements.

CONST - Errors in fixed and/or floating point constants such as too many digits, wrong decimal points, and missing exponents.

STNO - Errors in statement numbers such as duplicate or missing numbers and unnumbered statement after a transfer.

TRANS - Errors in transfer statements such as missing commas, missing parentheses, and incorrect indices.

GEN - Miscellaneous errors such as mispelling, unacceptable characters, and incorrect DIMENSION and FORMAT statements.

When an incorrect statement is detected, it is not processed by the Pre-Compiler, and this may cause spurious error messages in other subsequent statements which are correct. A little practise, however, will enable you to detect these readily.

The following statements are valid in Fortran II-D, but they will cause spurious error messages from the Pre-Compiler since they are not permitted in 1620 Fortran with Format:

6-letter variables	FIND	DEFINE DISK
3-dimension subscripts	FETCH	SUBROUTINE
* in subscripts	RECORD	FUNCTION
I/O in matrix form	COMMON	CALL
arithmetic functions	EQUIVALENCE	RETURN.

There are, of course, some errors which the Pre-Compiler does not detect. But additional error checks are built into the Fortran II-D Compiler which catch some of these both during compilation and during execution. It is obvious, however, that the proper solution to the given problem is the responsibility of the programmer, and no computer can detect errors in logic. If the program is incorrect, the computer will print out wrong answers just as quickly as right ones -- hence the common expression: GIGO -- garbage in, garbage out.

To use the Pre-Compiler, merely replace your ‡‡FORX control card with a ‡‡PCOM control card.

September, 1966 Frederick R. Henderson, Director

1620 H.S. JFC

HIGH SCHOOL USE OF A 1620 COMPUTER

by

Frederick R. Henderson
Director, Computer Center
Rochester Institute of Technology
Rochester, New York 14623
(1620 User #1393)

ABSTRACT

Our discussion is confined to the use of the computer as a tool in problem-solving. The IBM 1620 seems well adapted to this kind of use. "Hands-on" experience is very desirable at the outset but not so essential later. An introductory course in Fortran programming and related mathematics is suggested. The emphasis, as Hamming points out, is on insight, not numbers.

Presented at
COMMON Meeting
Philadelphia, Pa.
September, 1968

Session T6G

HIGH SCHOOL USE OF A 1620 COMPUTER

An article entitled "Computers for School Mathematics" appeared in the MATHEMATICS TEACHER for May, 1965, and it formed the basis for a booklet entitled "Computer Facilities for Mathematics Instruction" published in 1967.* Both of these are highly recommended to anyone involved in high school computing.

As is pointed out in this booklet, there are four distinct areas of computer education; these are:

(1) vocational education for computer technicians and programmers,
(2) computer appreciation or the impact of computers on society,
(3) computer-assisted problem solving or using the computer as a tool,
(4) computer-assisted instruction or computers as teaching machines.

We may eliminate the first and last of these categories immediately; vocational education is the responsibility of area vocational schools and community colleges, and computer-assisted instruction is still very much in the experimental stage. Computer appreciation is, however, appropriate at all levels; the difficulty is in obtaining source materials. One useful booklet here is "Computers - Theory and Uses" by Darnowski.** (It has a teacher's guide.)

The remaining topic, computer-assisted problem solving, is the one in which we are primarily interested. It can be used as an instructional aid in teaching mathematics, science, business, and other subjects. It can be used to demonstrate concepts as well as providing for laboratory experimentation. It is, therefore, to this aspect of a computer as an instructional tool, that we shall direct our attention.

In our judgment, the IBM 1620 computer is well adapted to this kind of problem-solving use. It is a stored-program computer with a memory large enough to process Fortran programs efficiently, and students need not be concerned with the intricacies of machine language programming. It is a decimal rather than a binary machine so that one does not have to spend time on the details of converting from one number base to the other. (This is interesting mathematics, but it is quite incidental to computer use.) Finally, in view of IBM's new educational discount, the cost is within reach -- at least for the larger high schools or school systems.

When it comes to actually using the machine, I am a firm believer in some initial "hands-on" computer experience for all new students. There is no better way of dispelling that psychological fear of the machine that so many people have at the outset. After students have gotten acquainted with the computer, "closed-shop" operations may be feasibile, but in the beginning, "open-shop" I believe is very important. In our Computer Center we employ part-time upper-class student assistants to show the new students how to operate the various pieces of equipment from key punch to central processing unit. In a short time most of them are operating the equipment themselves, but the assistants are there to help out if a card jams or a check stop occurs.

For a formal course in computer programming at the eleventh or twelfth grade level, we suggest the School Mathematics Study Group materials on "Algorithms, Computation and Mathematics"# These include a Fortran supplement as well as teacher's guides. Richard Andree's book on "Computer Programming and Related

---

* National Council of Teachers of Mathematics, Washington, D.C. 20036 (price 90¢)
** National Science Teachers Association, Washington, D.C. 20036 (price $1. each)
\# See attached bibliography.

Mathematics" (especially Chapter 5) is also highly recommended.* Another book which we are planning to use with some of our freshmen next year is "Mathematics and Computing" by Dorn and Greenberg.*

On the other hand, if a non-credit introduction to Fortran is wanted, this can be done in about 8 or 10 hours. Attached is an outline for such a course which has been given in a number of high schools in the Rochester, New York area. Judging by the number of repeat performances, it has been reasonably successful. We have prepared a set of notes to supplement the IBM Manual, and we usually start off with the film "Information Explosion" from the National Science Teachers Association.

Assuming now that the minimum computer configuration consisting of a 20K 1620 central processing unit and a 1622 card read punch is to be installed, what additional equipment is needed?

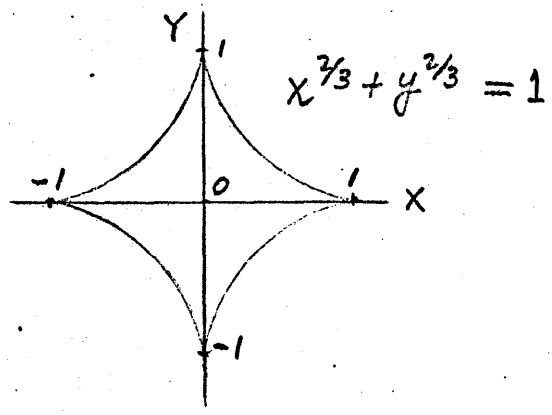
First of all, at least two 026 printing punches are required to punch the source decks and data cards. So long as your operations are confined to problem-solving not requiring large amounts of input data or extensive output, this is adequate since the console typewriter will type out answers directly.

However, if any administrative data processing is contemplated, a printer (either on- or off-line) will be needed to handle the output, and a card sorter will be useful in preparing cards for input. If you can possibly afford it, a 1311 disk file will greatly facilitate all of your operations by providing both temporary and permanent storage for both programs and data.

In conclusion, let me present a simple application in curve-sketching as illustrative of the kind of use which can be made of the computer. Many interesting equations are rarely graphed by hand because the calculation of the necessary set of points is too time-consuming, yet a few minutes on the computer, coupled with the necessary mathematical analysis will easily produce results. The figure below gives the computer program and output data for the equation $x^{2/3} + y^{2/3} = 1$ Data for the first quadrant are all that are required since the graph is symmetrical about both axes. As Richard Hamming has said, "The purpose of computing is insight, not numbers."

X	Y
.0000	1.0000
.1000	.6949
.2000	.5337
.3000	.4099
.4000	.3090
.5000	.2250
.6000	.1550
.7000	.0973
.8000	.0513
.9000	.0176
1.0000	.0000

```
      X=0.
      DX=.1
    1 Y=(1.-X**.66666667)**1.5
      PRINT13,X,Y
      PUNCH13,X,Y
      X=X+DX
      IF(X-1.)1,1,2
    2 STOP
   13 FORMAT(2F15.4)
      END
```



* See attached bibliography.

A BRIEF BIBLIOGRAPHY OF BOOKS ON COMPUTING FOR HIGH SCHOOLS

(Prepared for COMMON by F. R. Henderson, #1393, Sept. 1968)

Andree, Richard V.	Computer Programming and Related Mathematics Wiley, 1967
Bernstein, Jeremy	The Analytical Engine Random House, 1964
Burck, Gilbert	The Computer Age Harper and Row, 1965
Darnowski, Vincent S.	Computers - Theory and Uses (and Teacher's Guide) Nat. Science Teachers Assn., 1964
Davidson & Koenig	Computers Wiley, 1967
Dodes, Irving A.	IBM 1620 Programming for Science and Mathematics Hayden, 1963
Dodes & Greitzer	Numerical Analysis with Scientific Applications Hayden, 1964
Dorn & Greenberg	Mathematics and Computing Wiley, 1967
Freiberger & Prager	Applications of Digital Computers Ginn, 1963
Greenberger, Martin	Computers and the World of the Future M.I.T. Press, 1962
Gruenberger & Jaffray	Problems for Computer Solution Wiley, 1965
Hamming, Richard W.	Numerical Methods for Scientists and Engineers McGraw-Hill, 1962
Hamming, Richard W.	Calculus and the Computer Revolution Houghton-Mifflin, 1968
Hastings, Cecil Jr.	Approximations for Digital Computers Princeton Univ. Press, 1955
Leeson & Dimitry	Basic Programming Concepts and the IBM 1620 Computer Holt, Rinehart & Winston, 1962
McCracken & Dorn	Numerical Methods and Fortran Programming Wiley, 1964
Murray-Shelley, Richard	Teach Yourself Computer Programming Dover (English Univ. Press), 1967
N. C. T. M.	Computer Oriented Mathematics Nat. Council of Teachers of Math., 1963
N. C. T. M.	Computer Facilities for Mathematics Instruction Nat. Council of Teachers of Math., 1967
Newsweek	Words of the Computer Age (a glossary) Newsweek, 1966
Noble, Ben	Numerical Methods (2 volumes) Wiley (Oliver & Boyd), 1964
Polya, George	How to Solve It Doubleday Anchor (A93), 1957
Plumb, S. C.	Introduction to Fortran McGraw-Hill, 1964
Scientific American	Information Freeman, 1966
S. M. S. G.	Algorithms, Computation and Mathematics A.C.Vroman,Inc., Pasadena, Calif., 1966
Westwater, F. L.	Teach Yourself Electronic Computers Dover (English Univ. Press), 1967
Wilkes, M. V.	Short Introduction to Numerical Analysis Cambridge Univ. Press, 1966
I. B. M. Corp.	Various Pamphlets I.B.M. Sales Representative

CSA-ACM COURSE OUTLINE

INTRODUCTION TO COMPUTER PROGRAMMING

Session 1

What a digital computer is and how it operates
How to communicate with a computer
Programming (flow charts, Fortran)

Session 2

Fortran arithmetic statements
Integer and floating-point computations
Constants and variables
Arrays and subscripts
Library functions and subroutines

Session 3

Fortran control and specification statements
Branching (unconditional, conditional)
Looping
Specifications (dimension, common)

Session 4

Fortran input and output statements
Read, punch, and print
Format (numeric, alphameric, other)

Session 5

Review and summary of basic Fortran
Problem session

Session 6

Visit to a computer installation
Computer demonstration
Processing of student programs

Notes:

Sessions 1-5 are usually scheduled for 1½ hours each in the evenings at a convenient location. Session 6 is usually scheduled for 3 hours on a Saturday morning.

The text is the IBM "Fortran II General Information Manual" (form F28-8074). Also NOTES supplied by CSA.

If available, an overhead transparency projector and a 16-mm. sound film projector are needed for Session 1.

(Prepared for COMMON by F. R. Henderson, User No. 1393, Sept., 1968)

TS/-

A MACHINE UTILIZATION PROCEDURE FOR A SMALL UNIVERSITY COMPUTER CENTER

By
Paul A. Bickford
Director of the Computer Center
DePauw University
Greencastle, Indiana

September 9th 1968

A MACHINE UTILIZATION PROCEDURE FOR

A SMALL UNIVERSITY COMPUTER CENTER

BACKGROUND

DePauw is a small liberal arts college (2400) students) and has used Data Processing equipment in various phases of University Administration since 1957. An IBM 1401 system was installed a year ago to automate more effectively new and present Administration applications.

A machine utilization procedure was needed after the 1401 was installed whereby we could indicate to the administration that we were "paying our own way" in services rendered to the various administrative and academic departments of the University. We were essentially to become a service bureau operation to the University. Therefore it was necessary for us to implement a reporting procedure whereby we could provide the University Departments with a statement each month describing machines used and services rendered.

OBJECTIVES

In order to provide a useful billing document as well as an effective accounting tool we felt the procedure should possess the following attributes:

1. Provide a job description title.

2. Indicate for each machine used; time used, hourly charge rate and total amount charged.

3. Summarize all charges to all departments.

4. Be relatively easy to use i.e. provide for a simple method of recording time, preparing Job Cards and running the Department Charge Program.

5. Be relatively easy to program.

Beginning with last things first, because the need was considerably great and time was in short supply, the RPG language was chosen over AUTOCODER (assembly language). Also, our system was to primarily consist of summarizing several hundred cards once a month; therefore, RPG seemed to be the most appropriate language to use. Another influencing factor was the fact that we happened to have handy a very good RPG programmer. I can say this, because the day after the decision was made to write the program it was functioning as it is today.

GENERAL PROCEDURE

The time card (Exhibit A) had been developed in earlier times when the University used only Unit Record equipment. It was decided to continue to use the same format.

1. These cards reside at each machine in the installation and have the individual machine number prepunched.

2. Each Computer Center employee is assigned an arbitrary two digit number. When a project is completed the operator mark senses his number in the first two mark sense positions.

3. The next four mark sense positions are for the month and day of the date. The year is prepunched.

4. The last four mark sense positions are for time recorded in hours and tenths. The Key Punch operators use a time clock to record time in and out on each of their jobs. "In time" is manually substracted from "out time" and the result is mark sensed.

5. Job Description is written in by hand along with the (presigned) project or account number. Comments are encouraged and usually are added. If any charges are questioned by a department the comment section of a time card becomes quite important.

END OF THE MONTH

All cards are processed through the 514 reporducer where job description and mark sensing information is punched into the time cards.

1. Appropriate "project description header" cards (Exhibit B) which contain job description, account and job number (usually prepunched) are merged by hand with each corresponding group of time cards.

2. The cards are passed through the 514 reproducing punch where the mark sensed columns are converted into punches. Also, job description and number, along with account number is gang-punched into the time cards from the project description header cards.

3. The cards are then sorted on machine, job and account number (in that order) in ascending sequence.

4. Finally, the cards are ready for a quick run through the 1401 to produce the detailed description of charges (Exhibit C). Only a few minutes of 1401 time is required to print the report.

EVALUATION

Although we accomplished our original objectives, the following disadvantages soon became apparent:

1. It's difficult to keep an account of one's time when operating several different pieces of equipment as well as working on several jobs simultaneously. This problem results primarily because of the size of the installation i.e. one person being responsible for several jobs concurrently.

2. Too much clerical work is required to prepare job cards for computer processing. At present, at least on e working day is required by a key punch operator to prepare time cards.

EXHIBIT C

Sample Print Out of an Invoice

COMPUTER CENTER CHARGE DETAIL FOR AUGUST, 1968

ACCT.NO.-1247

JOB DESCRIPTION
NEW STUDENT FOLDERS

MACHINE USED	HOURS	HR.RATE	AMOUNT
0024	1.70	$ 5.00	8.50
0056	.32	$ 5.00	1.60
0548	.05	$ 5.00	.25
1401	2.00	$35.00	70.00
CLER	.40	$ 5.00	2.00

TOTAL JOB CHARGE- 82.35

JOB DESCRIPTION
SCHOLARSHIP VOUCHERS

MACHINE USED	HOURS	HR.RATE	AMOUNT
0024	8.50	$ 5.00	42.50
0026	5.08	$ 5.00	25.40
0056	13.50	$ 5.00	67.50
CLER	.22	$ 5.00	1.10

TOTAL JOB CHARGE- 136.50

JOB DESCRIPTION
PRESIDENTS REPORT

MACHINE USED	HOURS	HR.RATE	AMOUNT
1401	7.50	$35.00	262.50
0026	35.54	$ 5.00	177.70
CLER	33.37	$ 5.00	166.85

TOTAL JOB CHARGE- 607.05

JOB DESCRIPTION
LIST OF CLASS STANDING

MACHINE USED	HOURS	HR.RATE	AMOUNT
0083	.25	$ 5.00	1.25
1401	.18	$35.00	6.30

TOTAL JOB CHARGE- 7.55

JOB DESCRIPTION
NEW STUDENTS ADD CHGS

MACHINE USED	HOURS	HR.RATE	AMOUNT
0026	1.10	$ 5.00	5.50
0056	1.26	$ 5.00	6.30
0548	.05	$ 5.00	.25
CLER	.48	$ 5.00	2.40

TOTAL JOB CHARGE- 14.45

TOTAL ACCOUNT CHARGE- 847.90

EXHIBIT D

1401 RPG Department Charge Program Listing

```
CAA  080 C-                                    01
CBB  015 C1016 C4                              02018040140302607
CCC  015 C1016 C6                              03018040140302607
CDD  015 CP                                    04018040140302607
CEE                                            05018040140302607
DHEAD  050            CAA 050
DACCTN0007  B06       CBB 026          CCC 026          CDD 026
DACCTN0007  B06       CEE 026
DDESCRP030            CBB 057          CCC 057          CDD 057
DDESCRP030            CEE 057
DMACH  004            CBB 018          CCC 018          CDD 018
DMACH  004            CEE 018
DFOUR  00502P07B11    CBB 004004A 02
4SIX   00502P08B12    CCC 004004A 03
DPROG  00502P09B13    CDD 004004A 04
DOTHER 00502P10B14    CEE 004004A 05
AFOURT 00702           FOUR  X  @20@A F1      T
ASIXT  00702           SIX   X  @20@A F1      T
APROGT 00702           PROG  X  @10@A F1      T
AOTHERT00702           OTHER X   @3@A F1      T
AJOBT  00702                FOURT A F1        T
AJOBT  00702                SIXT  A F1        T
AJOBT  00702                PROGT A F1        T
AJOBT  00702                OTHERTA F1        T
AACCTT 00702                JOBT  A F2        T
LHAAX        0301     F3
F                            HEAD  069
LDBBX        02       F3
K                                  028        @ACCT.NO.-@
F                            ACCTNO035
LDCCX        01       F2
K                                  048        @JOB DESCRIPTION@
LDDDX        02       F2
F                            DESCRP059
LDEEX        01       F2
K                                  044        @MACHINE USED    HOURS@
K                                  067        @HR.RATE      AMOUNT@
LTFFX                 F1N06
B                            FOURT 067 F1 07N11 @    0.  @
B                            SIXT  067 F1 08N12 @    0.  @
B                            PROGT 067 F1 09N13 @    0.  @
B                            OTHERT067 F1 10N14 @    0.  @
F                            MACH  031 F1
B                            FOUR  044 F1 07N11 @  0.  @
K                                  056 F1 07N11 @$20.00@
B                            SIX   044 F1 08N12 @  0.  @
K                                  056 F1 08N12 @$20.00@
B                            PROG  044 F1 09N13 @  0.  @
K                                  056 F1 09N13 @$10.00@
B                            OTHER 044 F1 10N14 @  0.  @
K                                  056 F1 10N14 @$ 3.00@
LTGGX        02       F2N06
B                            JOBT  053          @    0.  @
K                                  046        @TOTAL JOB CHARGE-@
LTHHX                 F3N06
K                                  039        @TOTAL ACCOUNT CHARGE-@
B                            ACCTT 047        @    0.  @
```

EXHIBIT A

Sample Time Card

1401

DESCRIPTION OF JOB reorganize alumni Master Information file

FOR DEPT OR OFFICE alumni office

ACCT NO. _____ PROJECT NO. _____

COMMENTS:

Two trips were necessary. Program blew up.

446838
446529
3.09

OPERATOR

HRS.	OPR	BUDGET	USE	PROJECT NO.	MACHINE NO.		ACCOUNT NO.

MACHINE USED — CHECK ONE

026 – PRINT KEY PUNCH
024 – KEY PUNCH
056 – VERIFIER
083 – SORTER
077 – COLLATOR
407 – ACCOUNTING MACHINE
514 – REPRODUCER
548 – INTERPRETER
1620 – COMPUTER
831 – ADDRESSOGRAPH

OPR	DATE (MO / DAY)	HOURS USED (START AT LOWER LINE)
0	0 0 0 0	0 0
1	1 1 1 1	1 1 1 1
2	2 2 2 2 2	OUT 2 2 2 2
3	3 3 3 3 3 3	IN 3 3 3
4	4 4 4 4 4 4	OUT 4 4 4 4 IN
5	5 5 5 5 5 5	5 5 5 5
6	6 6 6 6 6 6	OUT 6 6 6 6 IN
7	7 7 7 7 7 7	7 7 7 7
8	8 8 8 8 8	OUT 8 8 8 8 IN
9	9 9 9 9 9 9	9 9 9 9
18 19	20 21 22 23	24 25 26 27

52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

IBM 814763

EXHIBIT B

Sample Project Description Header Card

01020416 1104 TUITION DEPOSIT VOUCHERS

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

IBM 5081

T5A

DOCUMENTATION

by

Thomas R. Harbron

Director

Anderson College Computing Center

Anderson, Indiana 46011

COMMON Meeting

April 8 - 10, 1968

Chicago, Illinois

DOCUMENTATION

Documentation is one of the most expensive activities of a computing center. Probably the only thing that is more expensive than good documentation is poor documentation or no documentation at all. For these reasons it is important that documentation be done well and efficiently.

The word "communication" probably best summarizes the functions of documentation. This communication is most crucial between the systems analyst and the programmer, the user, and the operator. Communication is also vital between the programmer and the operator, and between the programmer and other programmers who may be required to modify the program at a later time.

Three levels of documentation have evolved at the Anderson College Computing Center. Other centers may find a different pattern of documentation serves their needs better. These levels are systems, programming, and the Run Book.

The systems documentation provides an "overview" of the entire system. This includes not only programs but also unit record operations and manual operations. Typically a program is just one step in the system. The system analyst is responsible for the system documentation.

Because systems vary widely in scope there is no fixed pattern for system documentation. System documentation has varied from a couple of pages for a small system to fifty pages for a large system. The following paragraphs describe items that are normally included in the system documentation.

One or more system flowcharts are used to show the relationship between the various parts of the system. This is always true when the system can be broken down into a number of well defined jobs.

The data inputs and outputs from the system are always described in detail. This may include the forms that are used for the data.

The format of the data between the internal steps of the system may be described or details may be left to the discretion of the programmer involved.

A functional description of each step is also included this is normally in the form of a narrative.

The system documentation not only provides a good reference after the system is implemented but may also be used by the system analyst to communicate many of the system aspects to the programmer responsible.

The program documentation provides detailed information about each program. This documentation follows a rigid format which is outlined in figure 9. It is the programmer's responsibility to see that this documentation is provided as each program is completed. The sample program documentation is shown in figure 10a and 10b. The detailed flowchart and program listing are not included in this figure.

The program listing shows not only the source program as written but also (in the case of SPS) the actual machine language code generated. This is a great help in trouble-shooting and reprogramming.

A detailed flowchart is also an essential part of the program documentation. It not only is much easier to read than the program listing but shows the program logic which may have been obscured in the coding. The latter part of this paper is devoted to flowcharting.

The Run Book provides the operator with the information necessary to run a given job. Two assumptions are made in writing the Run Book:

(1) the operator is competent to run the machines but is totally ignorant

of programming and board wiring; (2) the operator knows nothing at all about the job.

The overall responsibility for the Run Book lies with the system analyst. The programmer, however, is responsible for the computer run portions.

The Run Book documentation consists of three parts. The first is a cover page giving the basic information about the job. An example of this is shown in figure 11a. The namdatory information required includes the name of the job, a brief description, the job trigger, the input data to the job, and the output data from the job. Intermediate data is not noted on the cover page.

The second portion of the Run Book documentation is the system flowchart for the job. An example of this is shown is figure 11b. Notice that all steps of the job are shown whether these involve a computer run, a unit record operation, or a manual operation. Also notice that each step of the job is identified by a letter in the lower right hand corner of the box.

The third portion of the Run Book documentation consists of one or more pages describing each step of the operation as shown on the flowchart. Examples of these instructions may be seen in figures 11c-f. A prescribed format is followed for each kind of step.

We found rather early that one of the biggest obstacles to good program documentation was the detailed flowchart required of the programmer. Programmers disliked this chore and tended to put it off as long as possible. The operation was also inefficient sometimes requiring as much of the programmer's time as did writing the original program.

In an effort to make the task of providing a detailed flowchart easier and more efficient, a program was written to automatically produce detail flowcharts directly from assembly language source decks. The program permits the programmer to determine exactly what will and will not appear on the flowchart, but relieves him of the repetitious and mundane details of flowcharting. Each source statement in the program can cause a box on the flowchart to be generated.

The program first examines the SPS statement and determines whether on not a comma is present in column 33 of the statement, or if the statement consists of only a comment. If the statement does not meet one of these requirements, or if it is a declarative, it is passed on and the next statement read. If the statement does meet one of these requirements it will cause an output on the flowchart. The program next looks at the statement to determine what shape the box should be. The box is made hexagonal for a branch instruction, trapazodial for an input/output statement, oval for a terminal, rectangular for processing, and no box at all for a straight comment.

After the shape of the box is determined, the statement number is placed above the box on the right side. If the statement has a label this is placed above the box on the left side. If the statement does not have a label, but a staement containing a label has been read since the last box was generated, that statement's label will be used.

This operation may best be illustrated by referring to figures 12 and 13. Figure 13a is one page of a flowchart made from an SPS source deck. The assembly listing of this source deck is shown is figure 12.

The first box in figure 13a is made from statement number 1220. This statement is an add instruction and therefore a rectangular box is generated. The comment "increment line counter" is placed within the box. A chain is generated down to the next box. The following statement (number 1230) is an

unconditional branch instruction. Therefore a hexagonal box is generated, and the label to which the branch will take place is placed to the right of the box. Again the comment is inserted in the box. Since the branch is unconditional, no chain is generated to the next box.

Other features of the flowcharting program can be seen in the last column of figure 13a. Notice that on the second box (number 1380) there is the label Q2. This is not the label of statement 1380 but is the label of statement 1375 which immediately precedes it. Also notice that while this is a branch instruction it also has a chain extending down to the next box. This is because the branch is a conditional branch and control may either pass to the label "clear" or follow on down to statement number 1390. Statement number 1390 causes a terminal box to be generated. Terminals are used for terminations of programs and for halts. The programmer indicated that a terminal should be charted by placing an asterisk in column 34. If there is to be no chain following the terminal box an additional asterisk is placed in column 35. Labels and comments are handled in the normal fashion.

The results of this program have been very good. Programmers now, in effect, draw their flowchart as the program is coded. Corrections or additions to flowcharts are readily made. The flowcharts are easy to read, and most important of all they get done.

Execution time of the program is I/O limited (reading 500 cards per minute, punching 250 cards per minute). As a result the revision of the flowchart is a minor task.

OUTLINE FOR PROGRAM DOCUMENTATION

I. Name and number

II. Purpose (Short Statement, Paragraph Form)

III. System Flowchart

IV. Control Devices

1. Control Cards
2. Typewriter Controls

V. Input

1. Order
2. Format
3. Example(s)

VI. Output

1. Order
2. Format
3. Example(s)

VII. Non-error Messages

1. Type of Notification (What's Typed, etc.)
2. Cause of error
3. Result and/or response

VIII. Error Messages and Halt Log

1. Halt Number or Effof Message
2. Cause
3. Response

IX. Switch Settings

X. Files Used

XI. Detailed Flowchart

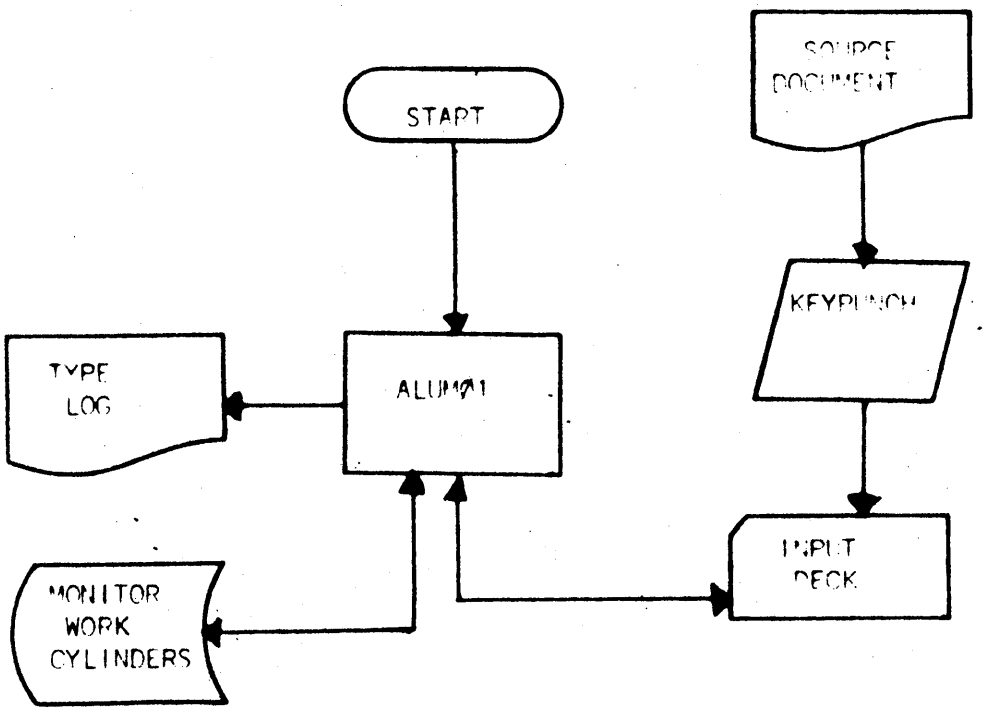
XII. Program Listing

ALUMØ1

Purpose

ALUMØ1 reads Alumni-Development cards and punches an expanded field into the same cards based on information in column 1-5 of each card.

System Flowchart



Control Devices

```
COLD START
**JOB ØØ
**XEQSALUMØ1
DATA
****
```

Input

Cards from Alumni-Development office.

Output

Cards used for input have additional information punched into them.

FIGURE 10a

Non-Error Messages

Message	Action
1. ROUTINE TO REPUNCH DEVELOPMENT-ALUMNI CARDS	1. None
2. XXXXX CARDS IN, XXXXX CARDS OUT	2. None

Error Messages

	Action
1. Standard Disk Errors	1. Standard Action

Switch Settings

	Disk	Parity	I/O	Arith	1	2	3	4
ON								
OFF	X	X	X	X	X	X	X	*

Files Used

MNWKC

Macros Used

RD
WD

FIGURE 10b

I. Name:

Armstrong

II. Description:

This job is used to reduce spectroscope data to meaningful terms that are used in the quality control of green glass manufacturing.

III. Job Trigger:

Telephone call from Armstrong at previously scheduled times.

IV. Input Data:

1. Raw spectroscope data transmitted verbally by telephone.

V. Output Data:

1. Reduced data given verbally over telephone.
2. Typewriter log-mailed to customer
3. Listing of input and output data - mailed to customer

FIGURE 11a

System Flowchart For Armstrong Data Reduction

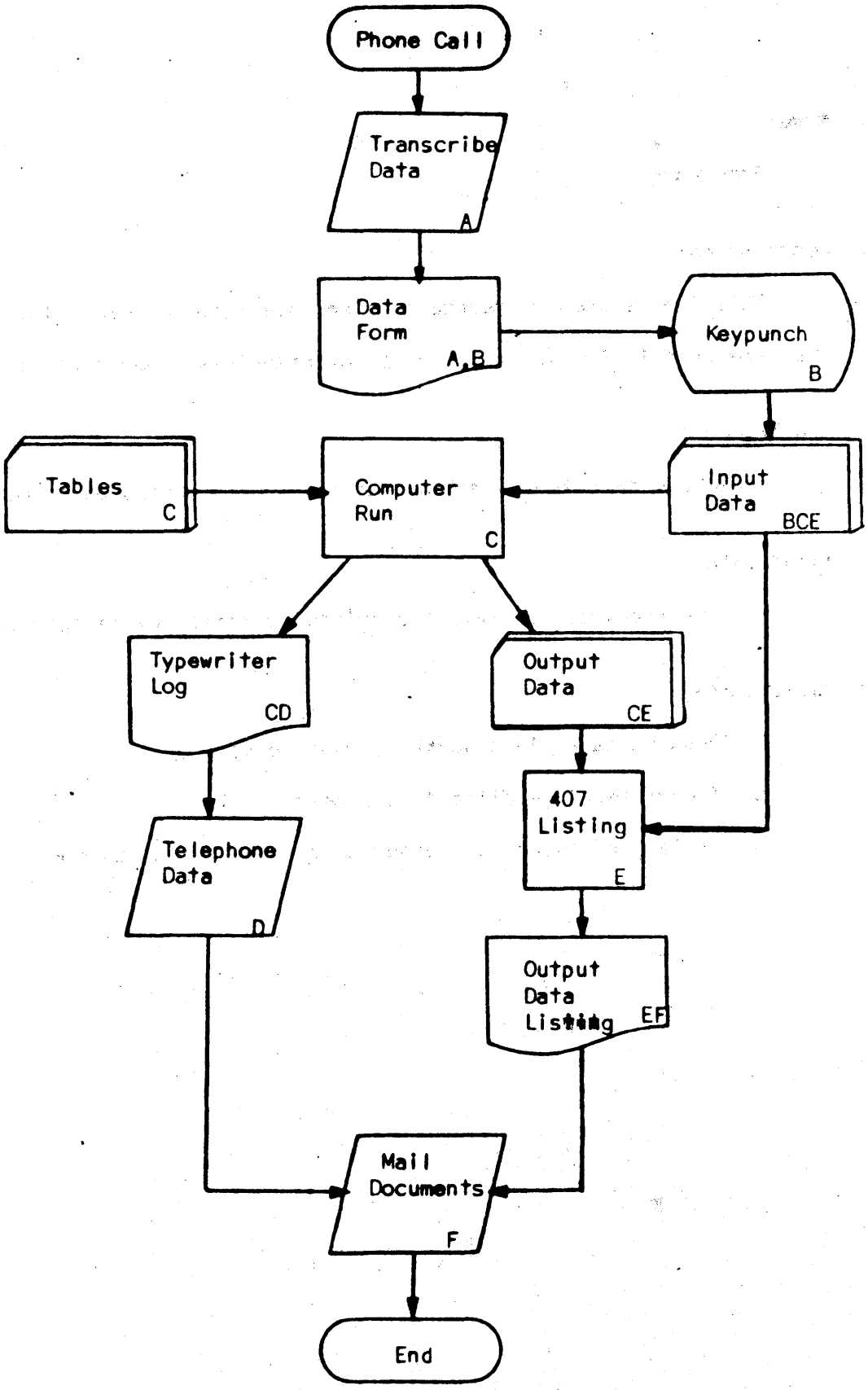


FIGURE 11b

Clerical Instructions

For Armstrong

Step A

I. Accept telephone call and determine which method (1, 2, or 3) is to be used. If methods 1 or 2 are used, select a data form labeled "Weighted Ordinated Method," if method 3 is used select a data form labeled "Selected Ordinate Method."

II. Using the selected data form, copy down on it all data supplied. This includes transmission percent values, thickness, computer case number, and method. (1, 2 or 3).

FIGURE 11c

Computer Run Instructions

For Armstrong

Step C

I. Program Source

A. Program Form - Fortran object Program

B. Program Location - Monitor Disk Pack

C. Loading Instructions - Control Cards as follows:

```
  ‡‡JOB
  ‡‡XEQSARST1                    3
*
  ‡‡XEQSARST2
```

*Use the first card for methods 1 and 2.
Use the second card for method 3.

II. Disk Packs

Module Ø - Monitor

Module 1 - None Required

III. Console Switch Settings

	Disk	Parity	I/O	O'flow	1	2	3	4
ON					*	*		
OFF	X	X	X	X	*	*	X	X

*1. Used to control repeating of the program in response to typed message.

2. Same as 1.

FIGURE 11d

12

IV. Input Data Decks:

1. For methods 1 and 2 only, a deck containing table values is read. There are two decks to choose from, one for each method. Both are located in the card filing cabinet. Select the one indicated by the data sheet, return it after use.

2. The data deck punched in Step B follows the table deck.

V. Output Data:

1. Certain data values are produced on the typewriter. These values are identified as X, Y, and B or XBAR, YBAR, and PCT B. These values should be read back over the telephone as soon as they appear (Step D).

2. An output data deck is punched for listing in Step E.

VI. Message and Error Log

A. ARST1

Message	Response
1. Input Data in error	1. Re-execute program by the following: a. Push "Insert" b. Type: 4902226 RS c. Reload tables after checking d. If error presists - call supervisor.
2. SS1 on for new tables, SS2 on to exit, push start.	2. If a single data set is being run, turn SS2 on and push start. If another data set is to run, leave SS2 off and set SS1 on or off depending on whether a different table, or the same is to be used. push start.

3. All fortran error messages are possible, but should not occur. Finish run if possible, call supervisor.

FIGURE 11c

4. Check Stop	4. Restart with cold start, re-execute. If trouble presists, call supervisor.

B. ARST2

1. SS1 on to loop, off to exit, push start.	1. If a single data set is being run, leave SS1 off and push start. Otherwise turn SS1 on and push start.
2. Fortran errors - see #3 above	
3. Check stop - see #4 above	

FIGURE 11f

```
01100          AM    LABAD,1280                                    03742 11 02936 -128(
01110          A     LABAD,COLNO                                   03754 21 02936 1482
01120          AM    LABAD,22                                      03766 11 02936 -002
01130          TFM   COUNT,5,10        ...INITIALIZE          COUNTER
                                                                   03778 16 14653 000-
01140 M        TFM   -LABAD,65,10      ...MOVE CHAIN       CHARACTER
                                                                   03790 16 02930 0000
01150 CHSM     DS    .*                                            03801 00000
01160          SM    COUNT,1,10        ...DECREMENT           COUNTER
                                                                   03802 12 14653 000-
01170          AM    LABAD,160         ...INCREMENT CARD      IMAGE
                                                                   03814 11 02936 -016(
01180          BD    M,COUNT           ...BRANCH IF NOT LAST LINE 03826 43 03790 1465
01190          TFM   CHSM,65,10        ...RESTORE CHAIN           03838 16 03801 0000
01200 CHECK    CM    LINO,ONE+7000                                03850 14 14822 J205
01210          BH    N                 ...BRANCH IF END OF COLUMN 03862 46 03894 0110(
01220          AM    LINO,1920         ...INCREMENT LINE     COUNTER
                                                                   03874 11 14822 -192(
01230          B7    R                 ...                     GO 03886 49 04090 0000(
01240 N        TFM   LINO,ONE-160      ...  RESET  LINE      COUNTER
                                                                   03894 16 14822 -489
01250          CM    COLNO,108                                    03906 14 14827 -010(
01260          BE    O                 ...BRANCH IF END  OF PAGE  03918 46 03950 0120(
01270          AM    COLNO,54          ...INCREMENT          COLUMN          NUMBER
                                                                   03930 11 14827 -005
01280          B7    R                 ...                     GO 03942 49 04090 0000(
01290 O        TFM   COLNO,0           ... RESET COLUMN      COUNTER
                                                                   03950 16 14827 -000(
01300 P        TFM   CARDA,ONE-158     ...INITIALIZE    CARD COUNTER
                                                                   03962 16 03992 -489
01310          WACD  BLANK                                        03974 39 14655 0040(
01320 CARDA    DS    .*+7                                         03992 00000
01330 Q        WACD                    ...PUNCH    CARD           03986 39 00000 0040
01340          AM    CARDA,160         ...INCREMENT     CARD   COUNTER
                                                                   03998 11 03992 -016
01350          CM    CARDA,ONE-158+9600                           04010 14 03992 J449
01360          BNH   Q                 ...BRANCH UNLESS LAST CARD     PUNCHED
                                                                   04022 47 03986 0110
01365 Q1       BNF   Q2,Q1+2                                      04034 44 04058 0403
01370          WACD  BLANK                                        04046 39 14655 0040
01375 Q2       SF    Q1+2                                         04058 32 04036 0000
01380          BNR   CLEAR,CIN         ...BRANCH UNLESS  E.O.J. CARD
                                                                   04070 45 02498 0471
01390          CALL  EXIT              ...**   CALL          EXIT 04082 49 00796 0000
01400 R        CM    OPC-4,0                                      04090 14 04737 -000
01410          BE    READ              ...BRANCH IF NO   LABEL    04102 46 02582 0120
01420          TF    OLAB,OPC-2        ...SAVE LABEL              04114 26 04891 0473
01430          B7    READ              ...  GO TO READ            04126 49 02582 0000
01440          DS    5                                            04137 00005
01450 LDFIG    TF    LABAD,LINO        ...SET UP ADDRESS TO LOAD  BOX
                                                                   04138 26 02936 1482
01460          A     LABAD,COLNO                                  04150 21 02936 1482
01470          AM    LABAD,40                                     04162 11 02936 -004
01480          SF    LDFIG-1           ...SET UP IN-    DIRECT ADDRESSFOR BOX
                                                                   04174 32 04137 0000
01490 LD2      AM    LABAD,160         ...     INCREMENT       LINE
                                                                   04186 11 02936 -016
01500          TF    -LABAD,-LDFIG+1...  MOVE IN LINE     OF   BOX
                                                                   04198 26 02930 0413
01510          SM    LDFIG-1,5         ...  INCREMENT        INDIRECT      ADDRESS
```

FIGURE 12

```
          01220                          01280                          01360
*******************                  **************                  **************
*                 *                  *            *                  *            *
*  INCREMENT LINE *                  *            *                  * BRANCH UNLESS *
*     COUNTER     *                  *     GO     *R                 * LAST CARD     *Q
*                 *                  *            *                  * PUNCHED       *
*                 *                  *            *                  *            *
*******************                  **************                  **************
          V                                                                V
          V                                                                V
          V                                                                V
          V                                                                V
          V 01230                     O      01290                     Q2   V 01380
   **************                  *******************                  **************
  *              *                 *                 *                 *              *
 *                *                *  RESET COLUMN   *                * BRANCH UNLESS  *
*      GO         *R               *     COUNTER     *               *  E.O.J. CARD    *CLEAR
 *                *                *                 *                *              *
  *              *                 *                 *                 *              *
   **************                  *******************                  **************
                                            V                                   V
                                            V                                   V
                                            V                                   V
                                            V                                   V
   N        01240                     P      V 01300                          V 01390
*******************                  *******************                  ***************
*                 *                  *                 *                 *             *
*   RESET  LINE   *                  * INITIALIZE      *                *  * - CALL     *
*     COUNTER     *                  * CARD COUNTER    *                *     EXIT     *
*                 *                  *                 *                *             *
*                 *                  *                 *                 *             *
*******************                  *******************                  ***************
          V                                   V
          V                                   V
          V                                   V
          V                                   V
          V 01260                     Q      V 01330                     R        01410
   **************                  *******************                  **************
  *              *                 *                 *                 *              *
 * BRANCH IF END *                   PUNCH     CARD                   * BRANCH IF NO   *
*   OF PAGE      *O                *                 *               *   LABEL        *READ
 *                *                                                  *              *
  *              *                 *             *                   *              *
   **************                   *************                     **************
          V                                   V
          V                                   V
          V                                   V
          V                                   V
          V 01270                             V 01340                          V 01420
*******************                  *******************                  *******************
*                 *                  *                 *                 *                 *
*  INCREMENT      *                  *  INCREMENT      *                *  SAVE LABEL     *
*     COLUMN      *                  *  CARD   COUNTER *                *                 *
*        NUMBER   *                  *                 *                *                 *
*                 *                  *                 *                *                 *
*******************                  *******************                  *******************
          V                                   V                                   V
          V                                   V                                   V
          V                                   V                                   V
          V                                   V                                   V
          V                                   V                                   V
```

16 FIGURE 13a

FIBa

PROGRAM TO DRAW FLOWCHARTS T R HARBRON 12/19/67

```
                                  CLEAR     00040                          00100
                                **********************              **********************
                                *                    *              *                    *
PROGRAM TO                      *      CLEAR         *              *  RESET LINE AND    *
DRAW FLOWCHART                  *       COUNT        *              *      COLUMN        *
T.R.HARBRON                     *        DIGIT       *              *      COUNTERS      *
                                *                    *              *                    *
                                **********************              **********************
        V                                V                                   V
        V                                V                                   V
        V                                V                                   V
        V                                V                                   V
  START V 00030                          V 00050                       READ  V 00110
*********************           *********************           *********************
*                   *           *                   *           *                   *
        TYPE                    *  SET UP CARD      *           *  SET FLAGS IN     *
*      IDENT        *           *  FIELD  ADDRESS   *           *    CARD AREA      *
                                *                   *           *                   *
*                   *           *                   *           *                   *
  ************                  *********************           *********************
        V                                V                                   V
        V                                V                                   V
        V                                V                                   V
        V                                V                                   V
        V 00034                     S2 V 00060                             V 00120
*********************           *********************           *********************
*                   *           *                   *           *                   *
      REQUEST                   *   CLEAR  CARD     *               READ ALPHA
*     HEADING       *           *       AREA        *           *      CARD       *
                                *                   *
*                   *           *                   *           *                   *
  ************                  *********************             ************
        V                                V                                   V
        V                                V                                   V
        V                                V                                   V
        V                                V                                   V
        V 00036                          V 00070                             V 00130
*********************           *********************              **************
*                   *           *                   *              *                *
      ACCEPT                    *      INCREMENT    *              * BRANCH UNLESS  *
*     HEADING       *           *       CARD        *             *  END OF JOB      *A
                                *      ADDRESS      *              * CARD            *
*                   *           *                   *              *                *
  ************                  *********************              **************
        V                                V                                   V
        V                                V                                   V
        V                                V                                   V
        V                                V                                   V
        V 00038                          V 00080                             V 00140
*********************              **************                 **************
*                   *              *            *                 *              *
      PUNCH                         * BRANCH IF NOT *              * BRANCH FOR END *
*     HEADING       *              *     DONE      *S2            *  OF JOB CARD     *P
                                    *            *                 *              *
*                   *              *            *                 *              *
  ************                     **************                 **************
        V                                V
        V                                V
        V                                V
        V                                V
        V                                V
```

FIGURE 13.1

F 36

T5A

AN ACCOUNTING SYSTEM FOR SMALL COLLEGE/UNIVERSITY COMPUTING CENTERS

By

DR. DONALD L. HENDERSON
Director of Computer Services
Mankato State College
Mankato, Minnesota

September 3, 1968

TABLE OF CONTENTS

I. Purpose

The development of computing centers in colleges and universities is relatively new with most of the centers developing from installations which were part of one of the administrative offices or an academic department. With this background, there was generally little thought or effort given to developing a record keeping system which would show equipment and personnel utilization; whereas, in a commercial business operation one of the first requirements of a data processing section is to show its worth to the company or be disbanded. Why should a college or university installation be different? Thus, we have as the prime purpose of a data processing accounting system, the ability to show that the data processing installation is in step with the objectives of the college/university.

II. Objectives

The record system used in a college computer installation must have definite objectives which will provide information (without interfering with the operations) that is meaningful and which will show an accurate picture of what is happening within the center's activities. This picture is needed by the director of the installation to properly administer the installation as well as by the college/university administration who must support the needs of the center before the supervisory board.

The objectives of the record system should be as follows:

1. Provide accurate data without reducing operations efficiency.

2. The inclusion of all areas, machines, personnel, materials, etc. connected with the computing center.

3. To furnish accurate and timely reports for the administration, director, and users.

4. The reports should contain enough information about the computer center to permit the director to use them in making long range plans for the center and also, to convince the administration that the computer center is in step with the objectives of the college/university.

III. A System

An accounting system with the above purpose and objectives should be flexible enough to fit a variety of installations. The following system is submitted to you for your consideration:

a) Data Collection: The collection of information from all areas of a computer installation is the most difficult phase of the entire system. The areas which must be included are the open and closed shop area for education and research users, the programming and systems section, and the operations section. The areas are examined below:

<u>Education and Research Users</u>: The means of collecting data from users (in a college or university situation this includes mainly students and faculty) must be as convenient as possible. There are two easy ways to make this collection: (1) use a machine usage card or (2) have the user punch a header card for the program and record computer time within the system. The second alternative, however, does not provide data on unit record equipment.

The first approach is recommended with the user recording the information on the Machine Usage Card (See Figure 1) by hand or with a time clock. These cards can then be used for input to the end of the month accounting program. If the users program is processed in closed shop, the closed shop card is used as a Machine Usage Card.

Programming, Systems, and Operation Sections: The user requests in these areas must be approved on the Computer Services Request Form (See Figure 2) and logged in the Log Book (See Figure 3). The machine and personnel usage is then recorded on the Data Processing Work Sheet (See Figure 4) which accompanies a carbon copy of the approved request form (the original copy of the request form is returned to the requester after it has been approved and logged in or disapproved).

The computer services staff must record all time and materials on the work sheets and thus, these sheets contain all the information desired about each request. The accompanying log number allows positive job identification.

b) Processing the Data and Generating Reports: The data on the cards and work sheets is coded where necessary and keypunched once a week. The weekly data is edited and a weekly report processed by a computer program showing the level of work completed that week and also a comparison made between the first shift and the second shift in the operation section. On the last Friday of the month, the Usage Reports are run for each user and summary reports are completed on employee, machine, and material usage.

c) Utilization of the Reports: User reports (See Figure 5) are run by the department/office and show the employee, machine, and material usage by each request (log number sequence) made from that department/office during the month. This report acts as a billing to the department/office and is used to add credit to the computer services office budget. The report is sent to the department/office with the carbon copies of the approved request forms.

The Employee and Machine Summaries (See Figures 6 and 7) are used in reviewing employee and machine utilization for merit pay increases and provide information on when new equipment should be added.

The Material Usage Report (See Figure 8) is used to update the inventory records to determine what supplies need to be ordered.

IV. Conclusion

The above system will provide the data necessary for running reports which can be used by the director to better administer the computer services center within a college/university structure. The director can also use this data to determine if the center is meeting the needs and objectives of the college/university. With this information in hand, the college administration can present a solid case to the governing board on where and how the center fits into the college/university plans. Hence the questions (1) "Is the computer services center a necessary part of the college/university?" and (2) "Is it doing the job it is suppose to do?" can be answered.

COMPUTER CENTER MACHINE USAGE

DATE _____

DEPT _____ CODE NO _____

NAME _____

INSTRUCTOR _____

CHECK ONE	CHECK ONE
___FACULTY	___COURSE WORK
	___INSTRUCTION
___STUDENT	___RESEARCH
	___ADMINISTRATIVE

MACHINE USED-INDICATE AMOUNT OF TIME(MIN)SPENT ON MACHINE IN THE SPACE PROVIDED.*

_____(01) 1620 Computer
_____(11) 026-2-1 KP
_____(13) 026-2 KP
_____(15) 026-3 KP
_____(17) 026-4 KP
_____(19) 029-5 KP
_____( ) _____

CARDS PUNCHED-PLACE NUMBER OF CARDS PUNCHED IN THE SPACE PROVIDED (150 CARDS TO AN INCH.*

_____(01) 5081 Man
_____(03) 5081 Col
_____(16) Fortran
_____(17) SPS
_____(18) Payroll
_____( ) _____
_____( ) _____

*NOTE: If name of machine or card is not listed, use the extra blanks provided. The code number (in parentheses) will be filled in by Computer Services personnel.

Open Shop Machine Usage Card

CLOSED SHOP INSTRUCTIONS CARD

FORTRAN	SPS
NAME _____	NAME _____
CLASS _____	CLASS _____
RUN ALL FOUR STEPS _____	COMPILE (PASSES I & II) _____
LIST _____	SYMBOL TABLE _____
PRE-COMPILE _____	SOURCE DECK LIST _____
COMPILE _____	OBJECT DECK LIST _____
RUN _____	SUBROUTINES:
	FIXED LENGTH _____
COMMENTS:	VARIABLE LENGTH _____
	MANTISSA LENGTH _____
	COMMENTS:

Closed Shop Machine Usage Card

FIGURE I - Machine Usage Card

(7-1-68)

COMPUTER SERVICES REQUEST FORM

Log No.: _____
Date: _____

Please process the request as outlined below by (approximate Due Date): _____
The cost for processing this request should be charged to: _____

REQUEST:

Signed: _____
Tel. No. _____ P.O.# _____

Please indicate the Quarter the information pertains to:

1st SS ____ Winter Qtr. ____
2nd SS ____ Spring Qtr. ____
Fall Qtr. ____

- -

CHECK OR FILL IN:

1. Operations to be performed:
   a. Keypunch ____
   b. Verify ____
   c. Reproduce ____
   d. Sort ____
   e. Collate ____
   f. List ____
2. Size of Paper:
   a. Narrow ____
   b. Wide ____
   e. Memo ____
3. Number of Copies: ____
4. Spacing:
   a. Single ____
   b. Double ____
5. Decollate ____
6. Burst ____
7. If job is a Payroll: (fill in)
   a. Pay period ending ____
   b. Date of Check ____
   c. First Check No. ____
   d. Payroll period No. ____

* *

COMPUTER SERVICES USE ONLY

APPROVED BY: _____ DATE: _____

FIGURE II - Computer Services Request Form

COMPUTER SERVICES LOG

Page _____
Year _____

Log No.	Office	Office Project Director	Date In	Date Due Desired	Date Due Actual	Application

FIGURE III - Log Book

DATA PROCESSING WORK SHEET

Dept. & Code:______________________________ Due Date:______________ Log No.___________

Job Description:____

* *

MACHINE CODE NUMBERS

1\. 1620 Computer
3\. 1401 Computer
5\. 024-2 KP
7\. 029-1 KP
9\. 029-2-1 KP
11\. 026-2-1 KP
13\. 026-2 KP
15\. 026-3 KP
17\. 026-4 KP
19\. 029-5 KP
20\. 046-6 KP
21\. 056 Verifier
23\. 059 Verifier
25\. 548 Interpreter
27\. 519 Reproducer
29\. 083 Sorter
31\. 085 Collator
32\. 5341 KP
33\. 1230 OpScan
34\. Burster
35\. Decollator

* *

TYPE OF WORK CODE NUMBERS

1\. Keypunch Operations
2\. Machine Operations
3\. Programming
4\. Systems & Analysis
5\. Other

* *

CARD & FORM CODE NUMBERS

1\. 5081 Manila
2\. 5081 White
3\. 5081 Colored
9\. 5081 Striped
16\. Fortran
17\. SPS
18\. Cur Earn Cd
21\. UG Off Cl Cd (Salm Strip)
22\. UG Cl Adm Cd (Solid Salm)
23\. Grad Off Cl Cd (Blue Strip)
24\. Grad Cl Adm Cd (Solid Blue)
25\. Lab Adm Cd
26\. Drop Card
27\. Add Card
30\. Housing Cd
31\. Wht Fee Cd
32\. Yel Res Cd
33\. Civ Ser Cd
34\. Stud Time Cd
35\. Valid Cd
36\. Statist Cd
37\. Rel Cens Cd
38\. Din Hall Cd
39\. Info Serv Cd
45\. Nar (1-pt)
46\. Nar (2-pt)
47\. Nar (3-pt)
48\. Nar (4-pt)
49\. Nar (5-pt)
50\. Wide (1-pt)
51\. Wide (2-pt)
52\. Wide (3-pt)
53\. Wide (4-pt)
54\. Wide (5-pt)
55\. Wide (8-pt)
58\. 8½x5½" Memo
60\. Perm Rec Lab
62\. Mail Labels
63\. Pr-On-Single 3½x15/16"
64\. " " Double
65\. " " Triple
66\. " " 7/16x2"
67\. " " 7x5/16"
70\. Fee Stmt Fms
71\. Prel Class Roster
72\. Official Gr Rost.
73\. Stud Gr Rep
74\. Back Sheets
75\. Pay Abs-Cs
76\. Pay Abs-Stud
77\. Con Tab Cd
78\. 10 5/8x4½" White
79\. Registr Fms
80\. 8½x5½" Memo (2-pt)
81\. 8½x5½" Memo (3-pt)

Mech	Min	Mat	Qty	Work Type	Oper	Min

____
(Signature) (Date)

(Form Revised 7-1-68)

FIGURE IV - Data Processing Work Sheet

DEPT 38 FINAIDS JULY 31 1968 BILLING AND REPORT OF COMPUTER SERVICES

JOB	MACH	MINUTES	COST	MATERIAL	AMOUNT	COST	EMPL	MINUTES	COST	COST
00642	1401	40	20.00				25	40	1.45	21.45
00642	1401	30	15.00	1PTW1	71	.47	25	30	[illegible]	16.[illegible]
00642	1401		.00	4PTW1	71	2.55	25		.00	2.55
00642	DBCO	5	.16				05	5	.18	.34
00642	BURS	8	.28				05	8	.29	.57
FA000							18	240	6.72	6.72
FA001							18	240	6.72	6.72
FA001							18	240	6.72	6.72
FA001							18	240	6.72	6.72
FA001							18	240	6.72	6.72
FA001							18	240	6.72	6.72
FA001							18	240	6.72	6.72
FA002	0083	240	10.99				18	240	6.72	17.71
FA002	0083	240	10.99				18	240	6.72	17.71
FA002							18	170	4.76	4.76
FA002							18	50	1.40	1.40
FA002	0291	15	.62	5081	[illegible]	.00	18	20	.56	1.18
TOTALS			30.02			2.97			70.21	131.20**

FIGURE V - User Report

EMPLOYEE TIME REPORT
COMPUTER SERVICES

EMPLOYEE	TIME	COST
1		.
2	8900	632.06
3	1669	104.36
4		.
5	446	16.02
6	490	64.51
7		.
8	4140	222.24
9	4925	369.32
10		.
11		.
12	2813	101.91
13	6790	300.24
14		.
15	2399	83.75
17		.
18	2400	67.20
19		.
20	350	28.92
21	6245	498.09
22	50	1.25
24		.
25	2630	97.34
27		.
31	39	.07
32	2372	79.42
33	320	7.98
34	60	1.50
35	525	60.46
40	1974	71.53
99		.
	49599	2865.97

FIGURE VI - Employee Usage Report.

MACHINE USAGE REPORT
COMPUTER SERVICES

MACHINE	TIME	COST
1620	2633	876.86
1401	5856	2928.00
0242	1245	41.25
0291	3349	138.96
0292	751	31.18
0261	746	27.87
0262	950	35.53
0263	1688	63.02
0264		.
0293	1229	51.07
0466		.
0056	1056	39.50
0059	3075	127.90
0548	1620	73.96
0519	632	28.81
0083	1380	62.79
0085	1130	51.58
5347		.
1230	521	94.04
BURS	142	4.68
DECO	70	2.29
		.
	28073	4679.29

FIGURE VII - Machine Usage Report

MATERIAL USAGE REPORT
COMPUTER SERVICES

MATERIAL	QUANTITY	COST
5081	32320	27.80
5081	4050	3.94
5081	24477	22.70
5081	3	.00
5081	2364	2.19
5081	4	.00
5081	260	.20
5081	292	.22
5081	4251	3.63
5081	260	.19
5081	1512	1.29
5081	674	.59
5081	827	.72
5081		.
5081		.
FOR	3762	2.73
SPS		.
LUREA	25	.02
UGCLA	242	.21
UGADM	25	.02
GRCLA	77	.05
GRADM	50	.04
LAB		.
DROP		.
ADD		.
PURPM		.
WHTMA		.
STUHO	20	.01
WHFEE	600	.64
YEFEE	528	.65
CLEMP	636	.58
SASST	163	.14
VALTO	12064	21.11
STAY	475	.39
RELIG		.
DINMA		.
INFOR		.
MOSEN		.00
MISC		.
1PTNA	422	2.09
2PTNA	212	2.52
3PTNA	479	9.00
4PTNA	209	5.41
5PTNA		.
1PTWI	1842	11.01
2PTWI	270	4.21
3PTWI	3260	58.76
4PTWI	1533	55.16
5PTWI	4	.18
8PTWI		.
BLKBX		.
MEMOS		.
MEMOL	203	.49

FIGURE VIII - Material Usage Report

MATERIAL	QUANTITY	COST
PERMR	1900	.00
PAHLS	1006	1.44
SPOLS	2424	8.23
DPOLS		.
TPOLS	1100	3.74
TPOLL		.
SPOLL	1000	7.00
FEEST	2	.00
PRELR		.
OFFGR		.
STGRD		.
BKGSH	10	.08
CSABS	26	1.56
STABS	23	1.38
CONTB		.
PLACE		.
REGFM		.
MEMO2	600	3.78
MEMO3		.
		.
	105490	266.10

TSB

AREA CODE 313
TELEPHONE 962-9510

ATOMIC POWER DEVELOPMENT ASSOCIATES, INC.
1911 FIRST STREET
DETROIT, MICHIGAN 48226

AN EASY METHOD OF UTILIZING MATHEMATICAL OPERATIONS

Prepared by

J. W. Balnaves

D. M. Green

September 6, 1968

INTRODUCTION

While converting existing programs from a larger computer to a smaller machine, in particular the IBM 1130, a need for an easy method of using mathematical routines for the solution of specific problems became very evident. This was primarily due to the rather large amount of extra coding necessary to utilize the 1130 scientific subroutine package. The procedure itself is not new, but, to our knowledge, the advantages of this system have not been presented or discussed before in general terms.

CURRENT METHODS

Many of the subroutines written for integration, iteration, etc. in the scientific subroutine package and other popular references require a user function subprogram to be prepared as part of the argument list in the call statement. This function subprogram is entered from the subroutine to calculate the mathematical function required for solution of the specific problem. The 1130 does not allow a function to change variables that are transferred to the function either through COMMON or the argument list. Therefore if there is more than one function to evaluate, several quantities may have to be reevaluated within each function. This could be very time consuming especially in an integration involving small step sizes.

In addition, the usual techniques provided for output involve either a separate output subprogram or an output vector transmitted through the argument list from the main program. Again there is the problem of communicating intermediate results and also the size of the output vector can be a problem on smaller machines.

NEW METHODS

In the system we propose, there is a general subroutine which controls the operation (integration, etc.), but rather than using additional subprograms for calculations, output, and termination, the routine returns to the main program. Two indicators are provided in the calling sequence to the general subroutine which are controlled by this routine to flag the current operation. At this point an example would be useful in describing the procedure.

The example we've chosen is a subroutine called SPDEQ to solve a system of first order differential equation. The routine is discussed in more detail in the attachment but briefly a typical main program is given in Figure 1. The arguments to SPDEQ are as follows:

X = independent variable array of dimension 3
Y = dependent variable array of dimension N
F = function array of dimension N
Q = erasable array of dimension N

N = number of equations
ISW = control variable used by main program
INIT = control variable used by SPDEQ
NPRNT = print frequency

The arguments X, Y, F, Q, N, and NPRNT are fairly standard arguments to a differential equation solving subroutine. The variables INIT and ISW are the control indicators. ISW is a flag used by the main program to satisfy the requests of SPDEQ. When we return to the main program after the call statement we check ISW by means of a computed GO TO statement. If ISW is equal to 1 we branch to statement 10 and calculate the derivatives. After calculating we branch to statement 1 and return to SPDEQ. If ISW is equal to 2 we branch to statement 20 and print any desired variables. After printing we branch to statement 1 and again return to SPDEQ. If ISW is equal to 3 the independent variable has reached its final value and we branch to statement 30, terminate, or perform any other post processing.

Many of the disadvantages of the current methods are not characteristic of our procedure. Since all calculations are in the main program, we have complete flexability of communication with any other routines called by the main program. Therefore we can use any intermediate results of the calculations. At any time the user can check the progress of the solution, alter any variables, restart, or control the system anyway he wishes. All output operations can be handled together in the main program. Hence, we do not need extra coding or an output vector. This will help to keep storage requirements at a minimum. Another favorable characteristic of our procedure is that with very little effort the routine may be made reenterant.

Continuing with the example, the SPDEQ subroutine uses Gill's modification of the classical Runge-Kutta method for the solution of initial-value problems. It is a fourth order integration procedure which is stable and self starting. Looking at the basic coding of SPDEQ (Figure 2), we see that the subroutine first checks the value of control argument INIT. This control is used to direct the flow within the subroutine when it is reentered from the main program. If INIT is equal to zero the routine initializes certain variables starting with statement 1, sets ISW (the control variable for the main program) equal to 1, sets control variable INIT equal to 1, and returns to the main program to calculate derivatives for initial conditions. Upon returning to SPDEQ, our check of INIT transfers us to the computed GO TO at statement 9. Since INIT is never set equal to zero in the routine, we will always transfer to statement 9 when SPDEQ is subsequently called. The only exception to this would be if we wish to restart the routine and set INIT equal to zero in the main program. From this point on the key to the internal control of SPDEQ is the computed GO TO at statement 9. After initialization we branch to statement 10. This IF test sends us to statement 11 which sets up our print return using IPRNT and returns us to the main program to print initial conditions. The subroutine then proceeds through the four steps of the calculation flaging the current operations by controlling INIT and ISW. After calculating the final

step, the routine checks for a print option and, if required, satisfies that option. Finally SPDEQ checks the independent variable to see if its final value has been reached. If the final value has been reached the subroutine terminates and returns to the main program with the proper flag. Otherwise, the routine increments the independent variable and proceeds with the solution of another step.

As we can see, the routine is very similar to any standard differential equation solving subroutine. The basic difference is the inclusion of two control variables in the calling sequence and then at the point the normal routine would call another subprogram, these control variables are set and the subroutine returns to the main program.

SUMMARY

The technique described above is not only applicable to differential equations. It has been successfully used in an iteration subroutine at APDA. By including all variables used by the subroutine in the calling sequence, the routine becomes reentrant. That is, the routine may essentially call itself when iterating on one or more dependent functions by merely changing the variable names in the calling sequence for each function.

We have shown how this technique eliminates the need for writing auxiliary subprograms and thereby increases the ability to communicate between various parts of the calculation. The technique may seem strange at first glance (as most new concepts do), but with a surprisingly small amount of effort, we're sure that the user will discover and profit by the many advantages.

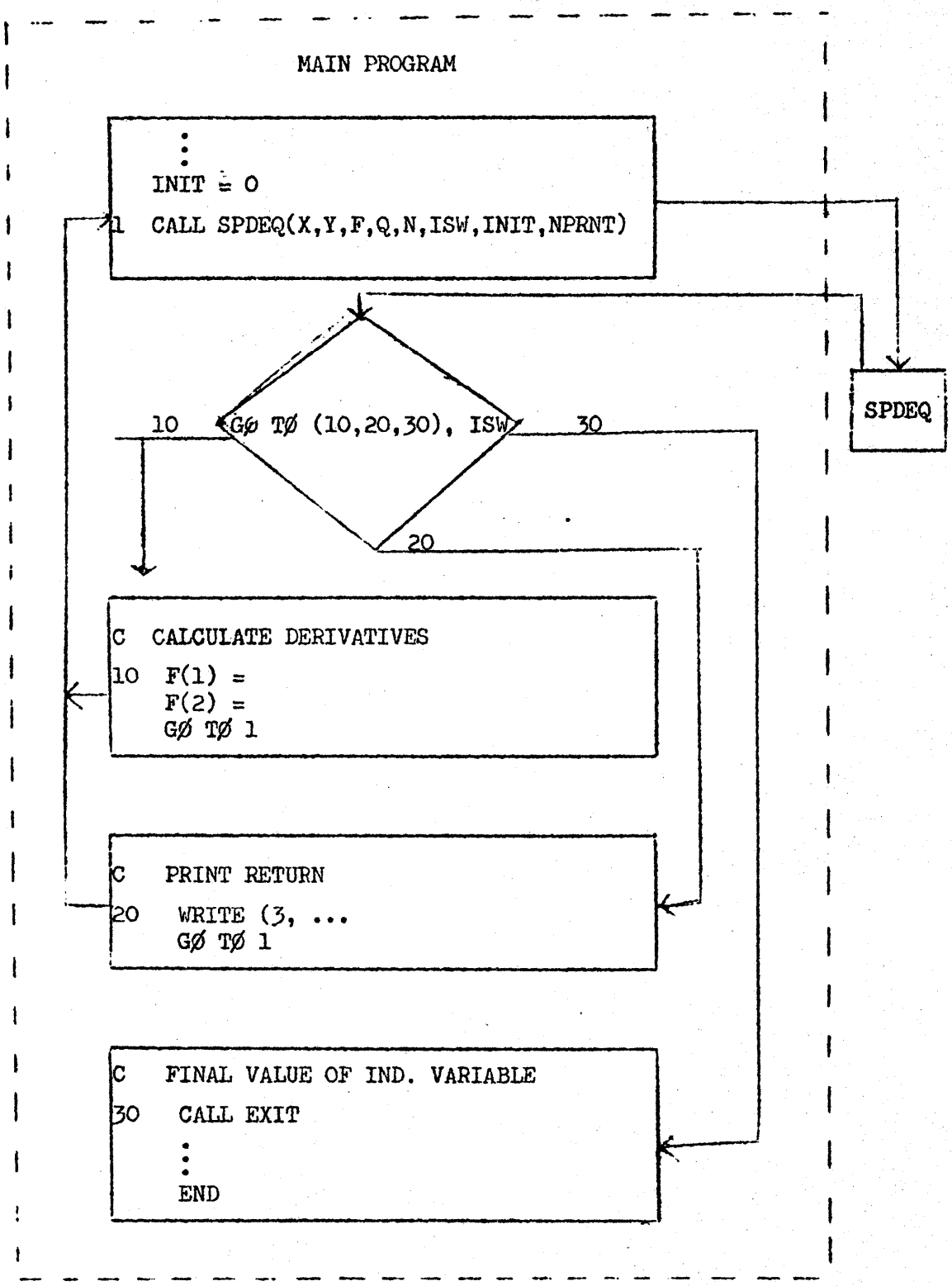


FIGURE 1 TYPICAL MAIN PROGRAM

FIGURE 2 TYPICAL SUBROUTINE

```
      IF(INIT) 9,1,9
C     INITIALIZE AND FIRST STEP
    1 IPRNT = 0
        :
      ISW = 1
      INIT = 1
      RETURN

    9 GO TO (10,20,30,40,50), INIT
   10 IF(IPRNT)  11, 11, 50
C     PRINT RETURN
   11 IPRNT=NPRNT
      ISW = 2
      INIT = 5
      RETURN

   50 IF(INDEPENDENT VARIABLE-FINAL VALUE) 52,52,51
C     TERMINATE
   51 ISW = 3
      RETURN

C     CALCULATE SECOND STEP
        :
   52
      ISW = 1
      INIT = 2
      RETURN

C     CALCULATE THIRD STEP
        :
   20
      INIT = 3
      RETURN

C     CALCULATE FOURTH STEP
        :
   30
      INIT = 4
      RETURN

C     CALCULATE FINAL STEP
        :
   40
      IPRNT=IPRNT-1
      INIT = 1
      RETURN
      END
```

ATOMIC POWER DEVELOPMENT ASSOCIATES, INC.

SPDEQ, AN 1130 SUBROUTINE FOR THE SOLUTION OF FIRST ORDER ORDINARY DIFFERENTIAL EQUATIONS

Prepared by

Joseph W. Balnave

David M. Green

September 4, 1968

REFERENCES A: H20-0252-2, 1130 Scientific Subroutine Package Programmer's Manual.

B: General Motor's Research Laboratories, The Ibsys Book.

In order to utilize the IBM supplied scientific subroutine package for the solution of a system of first order ordinary differential equations two additional subroutines must be supplied by the user. Also, if modifying on existing 7094 program, a significant amount of recoding is required in the user program.

To minimize this programming effort a subroutine called SPDEQ has been written. This routine has a calling sequence very similar to the routine used at General Motors. It does not require any additional subroutines and very little coding is needed to convert an existing 7094 program. This routine may also be used on the IBM 360/75/50.

Although SPDEQ will solve only a first order differential equation, higher order differential equations can be solved as follows:

Suppose we have $$\frac{d^2Z}{dx^2} = f(x, Z)$$

Let $$P = \frac{dZ}{dx}$$

Then $$\frac{dP}{dx} = \frac{d^2Z}{dx}$$

Our system of equations becomes:

$$y_1 = Z \qquad f_1(x, y_1) = y_2$$

$$y_2 = P = \frac{dZ}{dx} \qquad f_2(x, y_2) = \frac{d^2Z}{dx^2} = f(x, Z)$$

I METHOD

The subroutine uses Gill's modification of the classical RUNGE-KUTTA method for the solution of initial-value problems. This method will obtain an approximate solution of a system of first order differential equations with given initial values. It is a fourth order integration procedure which is stable and self starting.

Given a system of first-order ordinary differential equations:

$$y_n^1 = \frac{dy_n}{dx} = f_n(x, y_1, y_2, \ldots, y_n), \; n = 1, 2, 3, \ldots$$

with initial values

$$y_1(x_o) = y_{1,0}, \; y_2(x_o) = y_{2,0} \ldots, \; y_n(x_o) = y_{n,0}$$

where

$$Y(x) = y_1(x), \; y_2(x), \; \ldots \; y_n(x)$$

$$F(x,Y) = f_1(x,Y), \; f_2(x,Y), \; \ldots \; f_n(x,Y)$$

$$Y_o = y_{1,0}, \; y_{2,0}, \; y_{3,0}, \; \ldots \; y_{n,0}$$

the problem becomes:

$$Y^1 = \frac{dY}{dx} = F(x,Y) \text{ with } Y(x_o) = Y_o$$

starting at x_o where $Y(x_o) = Y_o$ and $Q_o = 0$. $Y_4 = Y(x_o + h)$ is calculated accordingly:

$$K_1 = hF(x_o, Y_o) \quad ; \; Y_1 = Y_o + \tfrac{1}{2}(K_1 - 2Q_o)$$

$$Q_1 = Q_o + 3\left[\tfrac{1}{2}(K_1 - 2Q_o)\right] - \tfrac{1}{2}K_1$$

$$K_2 = hF(x_o + \frac{h}{2}, Y_1) \quad ; \; Y_2 = Y_1 + (1 - \sqrt{\tfrac{1}{2}})(K_2 - Q_1)$$

$$Q_2 = Q_1 + 3\left[(1 - \sqrt{\tfrac{1}{2}})(K_2 - Q_1)\right] - (1 - \sqrt{\tfrac{1}{2}})K_2$$

$$K_3 = hF(x_o + \frac{h}{2}, Y_2) \quad ; \; Y_3 = Y_2 + (1 + \sqrt{1/2})(K_3 - Q_2)$$

$$Q_3 = Q_2 + 3\left[(1 + \sqrt{1/2})(K_3 - Q_2)\right] - (1 + \sqrt{1/2})K_3$$

$$K_4 = hF(x_o + h, Y_3) \quad ; \; Y_4 = Y_3 = 1/6\,(K_4 - 2Q_3)$$

$$Q_4 = Q_3 + 3\left[1/6\,(K_4 - 2Q_3)\right] - \tfrac{1}{2}K_4$$

where K_1, K_2, K_3, K_4, Y_1, Y_2, Y_3, Y_4, Q_1, Q_2, Q_3, Q_4 all have n components.

If the above procedure could be accomplished with infinate precision vector Q_4 above would be zero. Actually, Q_4 represents approximately three times the roundoff error accumulated in Y_4 in one step. To adjust for this, Q_4 is used as Q_o in the next step and $(x_o + H)$ and Y_4 serve as x_o and y_o respectively.

II USAGE

Call SPDEQ (X, Y, F, Q, N, ISW, INIT, NPRNT)

where X = array of dimension three
- X(1) = the independent variable
- X(2) = step size used in solution
- X(3) = final value of the independent variable

Y = solution array of dimension N

F = derivative array of dimension N

Q = erasable array of dimension N

N = number of equations

ISW = return argument with following types:
- = 1 calculate the derivative and return to SPDEQ
- = 2 intermediate print and return to SPDEQ
- = 3 independent variable has reached its' final value.

INIT = integer variable used as a control by SPDEQ. It should be set equal to zero before initial call of routine

NPRNT = control returned to calling program every NPRNTth step so answers may be printed. There is also a return at the initial time step.

The value of INIT is controlled by the routine so that the user does not have to be concerned with it after setting equal to zero. If during the calculation it is desired to reinitilize the subroutine (such as change the step size) INIT should be reset equal to 0 and the subroutine called.

III CONVERSION FROM GM SPDEQ

The conversion from the GM subroutine is very easy and straight forward.

GM SPDEQ	1130 SPDEQ
Assign 3 to IFUN	Remove
1. Call SPDEQ 1(X,Y,F,Q,N,IFUN) Call SPDEQ 2(M)	1. INIT = 0 Remove
2. Call SPDEQ (Return) IF (Return) 4, 5, 4	2. Call SPDEQ (X,Y,F,Q,N,ISW,INIT, NPRNT) GO TO (3, 4, 5), ISW
3. Calculate derivatives GØ TO 2	3. Calculate derivatives GØ TO 2
4. Print return GØ TO 2	4. Print return GØ TO 2
5. Independent variable has reached its final value.	5. Independent variable has reached its final value.

IV EXAMPLE

Given the following system of equations:

$$\frac{dY_1}{dx} = x^2 - 2x + Y_{1/2}$$

$$\frac{dY_2}{dx} = 3Y_2 + 4$$

We solve for the value of Y_1 and Y_2 as x varies from 0 to 1. The sample program and output are attached.

```
      DIMENSION  X(3), Y(2), F(2), Q(2)
      READ (2, 1000)  X, Y, NPRT
 1000 FORMAT (3 F10.4/2F10.4,I4)
      WRITE (3, 1001) X, Y, NPRT
 1001 FORMAT (13H1 INPUT  VALUES//1X, 5HX(1) = F7.3, 2X5HX(2) =
           F7.3, 2X5HX(3) = F7. 13//1X, 5HY(1) = F7.2, 2X5HY(2) =
           F7.2, 2X6HNPRNT = I4///1X13HOUTPUT VALUES/ 2/4X4HX(1),
           10X4HY(1), 9X4HY(2))
      INIT = 0
    5 CALL SPDEQ (X, Y, F, Q, 2, ISW, INIT, NPRT)
      IF (ISW-2)  10, 20, 20
   10 F(1) = X(1) ** 2-2. *X(1) + Y(1) /2.
      F(2) = 3.*Y(2) + 4.
      GO TO 5
   20 WRITE (3, 2000) X(1), Y(1), Y(2)
 2000 FORMAT (1HO, F8. 2, 3X2E13.4)
      IF (ISW-3)  5,99,99
   99 CALL EXIT
      END
```

INPUT VALUES

X(1) = 0.000 X(2) = 0.001 X(3) = 1.000

Y(1) = -1.00 Y(2) = -1.00 NPRNT = 100

OUTPUT VALUES

X(1)	Y(1)	Y(2)
0.00	-0.1000E 01	-0.1000E 01
0.09	-0.1061E 01	-0.8833E 00
0.19	-0.1143E 01	-0.7259E 00
0.29	-0.1246E 01	-0.5133E 00
0.39	-0.1369E 01	-0.2264E 00
0.49	-0.1511E 01	0.1607E 00
0.59	-0.1670E 01	0.6834E 00

OUTPUT VALUES (continued)

0.69	-0.1846E	01	0.1388E	01
0.79	-0.2036E	01	0.2341E	01
0.89	-0.2241E	01	0.3626E	01
0.99	-0.2458E	01	0.5361E	01
1.00	-0.2460E	01	0.5381E	01

72

THE PALO ALTO LABORATORY SYSTEM

by

C. J. Jenny, P. J. Friedl,
C. H. Sederholm and T. R. Lusebrink

Abstract

The Palo Alto Laboratory System (PALS) is a multiprogramming monitor system, specifically designed for laboratory automation, providing a sophisticated and versatile interface with a group of analytical instruments and similar devices. The system allocates core and disk dynamically depending on the users' needs. The system also features a problem-oriented laboratory language, an interactive job control language and support for independent asynchronous data transfer between the computer and multiple scientific instruments.

For information contact:
Dr. P. J. Friedl
IBM Scientific Center
2670 Hanover Street
Palo Alto, California 94304
(415) 327-2300

Computers have found a wide range of applications in the analytical chemical laboratory. So far, most systems have been used off-line, but recently many scientists have begun to apply computers on-line to their instruments.

A well-designed on-line laboratory system should meet a series of requirements in order to be a useful tool to the scientist.

1. Isolation of Programs: Each scientist wants to concern himself only with his own problems and does not want to worry about malfunctions of any other users' programs, let alone structuring part of his programs into general purpose programs together with other users (programming by committee).

2. Accessibility: A user wants access to the computer whenever he is ready to perform his experiment. He does not want to schedule his runs far in advance.

3. Acquisition and Processing of Data On-Line: Data must not only be acquired, but also processed, standardized and compared against known parameters, and finally presented in usable report form to the user. At the same time the instrument ought to be controlled and interactive user requests should be honored. All these steps should be performed without taking the computer off-line.

4. Input/Output: A front end is required for interfacing with the instruments. Externally synchronized channels should be available for demand/response input.

   Conversational interaction with the system should either be provided by typewriter-like terminals or pushbutton consoles. Output devices such as line printers should be available for bulky reports.

5. Ease of Operation: The scientist using a laboratory automation system is usually not a programmer. It is therefore mandatory to keep programming and system maintenance as easy as possible. The programming language ought to be problem-oriented with a heavy emphasis on input/output statements. A terminal-oriented job control language will give each scientist the means to communicate with the system, e.g., add and delete files, call for a new program, cancel a running job, etc.

6. Efficiency: The system should be able to take advantage of the fact that most analytical instruments have low duty cycles (i.e., much of the time they are idle). By allowing multiple users on the system it is possible to allocate system facilities dynamically among them and thus reduce the total size of the system below the sum total of each experiment's requirements.

7. Cost: The cost of the system ought to be spread over as many instruments and users as possible without conflicting with any of the above-mentioned requirements.

The Palo Alto Laboratory System (1800 PALS)

As a result of the above considerations we developed a system for asynchronous acquisition of data from multiple instruments and subsequent processing of the acquired data. All users' programs run independently of each other and do not need to take into consideration any other program running in the system at the same time.

Also, each instrument may have its own data path to the core of the computer via a specially designed digital multiplexer channel (Special Purpose Multiplexer Channel, RPQ CO8451/5) which provides up to 32 discrete input/output subchannels. Data transfer is in demand/response mode synchronized by the instrument or its interface. Analog digital conversion takes place in the instrument interface.

The PAL system is designed for an IBM 1800 system with a minimum of 24K words of core. It may be run in degraded form on a 16K machine. Two 2310 disk drives are required as a minimum (Models A2 or C2).

The PALS Monitor

The PALS monitor consists of a group of relocatable modules for input/output control, loading of user programs, controlling multitasking communications and time-slicing operations.

System programs communicate with the various modules through task control blocks and task complete control blocks. When a task is to be performed, a task control block is sent to the appropriate module,

which in turn will queue up the task control block address. As soon as the module has performed all previous tasks which are pending, it will unqueue the control block, execute the task and return a task complete control block to the originating program.

A foreground and a middleground monitor are responsible for translating user task requests into control blocks.

After a user has given out a number of tasks and has completed all necessary processing before one of his I/O operations is completed, he relinquishes control. This removes him from the active status until a task complete control block returns control to him. Similarly, if his processing time exceeds 5 milliseconds in the foreground (and 100 milliseconds in the middleground), his program is temporarily suspended until all other users in the foreground (middleground) have been allocated an equal time slice.

Core Allocation and Multitasking

Variable core (the part of core not being taken up by the system modules) is divided into 512 word pages. The loader module will place the users' programs into a set of empty pages that need not be contiguous and will then link-edit them together. Also all instructions that are not to be modified will be storage protected. Typical load/link-edit time for a seven-page program is about 1.7 seconds.

Subroutines that are shared by multiple system users (e.g., arithmetic and functionals) reside in groups on system subroutine pages. Calls to these routines are linked at load time.

Disk Allocation

Disk files are organized in the form of logical tapes which automatically expand or contract dynamically according to the amount of data stored on them. They are defined by name and allocated by the system one cylinder at a time.

Reading and writing to logical tapes may either be done sequentially or by random access.

47

The PALS Language

The PALS language is a macro language with statements natural to the laboratory environment. It has been designed for easy handling of sensor-based I/O (e.g., multiplexer channel commands, logical statements to set up bit patterns for control of instrument interfaces, etc.), handling of logical tapes, timers and interrupts and also features a set of data analysis-oriented statements which are very similar to FORTRAN and require very little relearning for users familiar with FORTRAN.

The macro language permits the programmer to mix assembler statements with macro statements. He also may define new statements if he so wishes. The macro-assembler is presently being executed as an off-line program.

Conclusion

PALS is an experimental system with a highly dynamic storage facilities allocation scheme. It has been demonstrated at the Pittsburgh Conference on Analytical Chemistry and Applied Spectroscopy in Cleveland in March 1968, interacting with four analytical instruments. It has been submitted for acceptance to the Type III library.

A paper on PALS is to be presented at the 1968 Fall Joint Computer Conference in San Francisco.

COMPUTERIZED CATALOG SYSTEM

By

Peter Woodrow

This system is a group of programs written primarily in 1130 FORTRAN IV. They are designed to read free-format cards containing textual information and store the information on disk. The stored information may be printed out in any format desired by these programs.

They will handle 2000 characters on an 8K 1130 and 16,000 characters on a 16K system.

The programs run in four links and operate at maximum card read speed on printer. Of the four segments, three are for input and the fourth is for output.

The system's primary use is side by side library cards.

The beginning of a field is defined by the dollar sign ($). The field must be at least two characters long (one may be a blank). The equals sign (=) is used as a separator.

The system may be set to check for up to five specific characters, to reject invalid characters, or to group certain items, coded by check character.

Output may be with or without page numbers on listing, or record output. The system does not do hyphenation.

The main lines use two large subroutines to handle library material.

Mr. Wil Braden of Marshall Communications in Santa Ana, California, said that this systems abilities reminded him of an IBM package for the 1130, called "STEEPLE", in its ability to test for individual characters and revise formats.

Members suggested that Systems try to get a presentation from IBM on STEEPLE for the Houston meeting in December.

Information on the Computerized Catalog System may be obtained from Peter Woodrow of Aeronautical Research Associates, 50 Washington Road, New Jersey, 08540.

T6B

SMALL COMPUTERS AND SIMPLEX OPTIMIZATION TECHNIQUE FOR MIXTURES

W. A. Pease, Jr., W. G. Vardell

March 15, 1968

For Presentation at Philadelphia COMMON

September 10, 1968

CHARLESTON RESEARCH LABORATORY

WEST VIRGINIA PULP AND PAPER COMPANY

CHARLESTON, SOUTH CAROLINA

481

ABSTRACT

This technique will optimize properties of any mix on a small computer, where the properties are functions of the ratios of the ingredients. Previous computer methods required plotters and larger core than utilized here. It is based on methods developed by Henry Scheffe and expanded by Gorman and Hinman.

Two things enable one to use a small computer. First, standard multiple regression techniques are used, simplifying programming and reducing core requirements. Second, an efficient program that generates the values for plotting the response surface has been written. One set of runs is needed for each property optimized.

Used as a screening device, the method is excellent for determining profitable areas of study.

TABLE OF CONTENTS

INTRODUCTION

In 1957, Henry Scheffe (2) pointed out that the properties of any mix could be predicted when these properties were functions of the ratios of the ingredients. His procedure is based on the use of a simplex as the plotting surface, within which the coordinates for any point sum to unity. Observed property values at specific points are plotted. These values are, in reality, projections of points existing on a response surface above the simplex. It is the equation of this surface that one attempts to derive.

Scheffe chose specific ratios so that the coefficients of the various terms in the mathematical model of the response surface could be readily calculated by hand. Later users (1) of this method continued to select Scheffe's points and method of coefficient determination even though the computation was now done by computer. Programs have also been published to plot the response contours on simplex coordinates using X-Y plotters (3).

Inflexible use of Scheffe's points and method of determining the coefficients has certain disadvantages, however. It requires more core than many small computers have available. It restricts the realm of study to a triangular area which may not be convenient. It also requires that the property must have a measurable value at each of the specific mix ratios on this triangular subspace. This often is not so.

Later users had overlooked that Scheffe reported his method as a form of polynomial regression. This fact was recognized by Dr. James Walker, Department of Mathematics, Georgia Institute of Technology, who was working with the authors, using a minimum configuration IBM 1620.

Use of multiple regression not only permits use of a smaller computer but gives much more flexibility to the experimenter since he is no longer tied to a given number of specific points. Any number above the minimum of evenly scattered points may be used. One may weight the equation for reliability in desired areas, expand the simplex area by addition of new points, or contract it for intimate exploration of the optimum. We are also permitted to examine subspaces other than triangular areas, though our coordinates remain triangular.

The present paper demonstrates the programming technique used to adapt this method of study to a small computer. The application of the method to a sample problem is also illustrated.

PROGRAMMING TECHNIQUE

Determination of coefficients on a small computer was no problem, since we were already using complex regression techniques at Westvaco. We were faced with the problem of generating the response points for plotting. The typewriter was slow, core was apparently insufficient to hold all the mathematical models desired, and economical segmentation was impractical without disk.

The typewriter problem was overcome by screening out points not on the simplex and outputting only effective points. Switching the order of some of the terms in the model equation was useful to avoid segmentation, since this dramatically reduced the number of decision statements required. This also made the generation of interaction terms faster. Core limitations were also met by using one general equation to cover all models, deleting undesired terms by reading in their coefficients as zero. The enclosed program will give points for the quadratic, special cubic, pure cubic, and special quartic math models (See Mathematical Models). There is room on the IBM 1130 for higher models with 8K.

Of particular interest from a programming point of view are the self-adjusting upper parameters for the nested DO-loops. The method is shown below:

```
      N5= -1              (This makes the first subtraction a zero subtraction)
      DO 50 I= 1,N3
      N5= N5 + 1          (Zero on first pass, due to above)
      N6= N3 - N5
        :
      N7= -1              (Same reason as N5 above)
      DO 50 J= 1,N6       (Note that N6 was defined in the next outer loop)
        :
   49 CONTINUE            (A separate CONTINUE is required for each inner loop)
   50 CONTINUE
```

In the above, N3 equals the number of intervals desired on the simplex border plus one. This is our chosen unit.

Variable N5 is the number of passes already made through the outermost loop.

Since the sum of the loop indices (I,J,K,L,M) must equal the chosen unit, here 11, the next loop index cannot exceed the value of (N3-N5). The same thing applies to the next inner loops, except that there is one additional index value over that of the previous outer loop to be subtracted in computing the current loop's upper index.

This results in continuously decreased upper limits for the inner loops as the outer loops progress. Inefficient passes made through the loops are minimzed.

Generally speaking the upper index for the ith inner loop equals ($N_{4+2i} = N_3 - N_{3+2i}$) and in each case N_{3+2i} is initialized to -1 prior to entry into ith loop.

This technique is only usable where the limits of the nested loops must sum to an integer, in this case 11, since the base for our unit is 10, and we need one more to get the zero values.

(The program is made to permit a constant being used where one forgets to repress it in the regression, and to make it flexible for other uses.)

Note how the zero levels are generated. A_i is made equal to the ith loop index minus one; thus when the index is one, A_i is zero. This is the reason N3 is N2, increased by one.

A more economical way of programming this from a core storage standpoint would be to put FORMAT 5 and the write loops into a subroutine as follows:

```
      SUBROUTINE PTWRT(MS,N1,BP,B,X)
C     MX = LOGICAL OUT PUT UNIT
      DIMENSION B(1),X(1)
      Y = 0.0
      DO 10 I = 1,N1
   10 Y = Y + (B(I)*X(I))
      IF (N1 - 3) 12,14,14
   12 Y = Y + (B(4)*B(4))
   14 Y = Y + B(42) + BP
      WRITE(MX,5)X(1),X(2),X(3),X(8),X(16),Y
    5 FORMAT (F9.2,3X,E16.9)
      RETURN
      END
```

Then a CALL PTWRT(3,N1,BP,B,X) statement could be substituted for statements 21, 26, 32, and 38. Statements after them, through their respective WRITE statements could be removed. These four calls would replace 19 of the present statements in the program. Expanding the WRITE statement in the subroutine would make it useable for more than five components.

In the program, A(I) is used to get the coordinates for the response points. Note how the zero points are generated. The reason for CONV in the program is that some experimenters like to work with unity, some ten, and some with one hundred as a base. This permits the printout of the coordinates to suit their particular desires.

AN in the program is our control to test CT, the constraint variable to insure that only points on the simple surface get printed.

METHOD OF APPLICATION

The steps employed in using this technique are as follows:

1. Define Study Area

   The size of the area studied will depend on our knowledge of the system. For an exploratory run, make the simplex area of investigation large enough to detect unexpected optimums. Prior experience with a system would permit an optimization run within a restricted area of the simplex.

2. Select Model Equation

   From the model equations shown in Appendix A, use the lowest order of equation expected to give a reasonable fit. Generally this order is one more than the number of points of inflection believed present on the response surface. The special cubic is a good place to start for systems of three or more components.

3. Select Experimental Points

   From Table 1, determine the minimum number of points required for the model equation and number of components used. It is important to either replicate or exceed these minimums to insure significance for the derived equation.

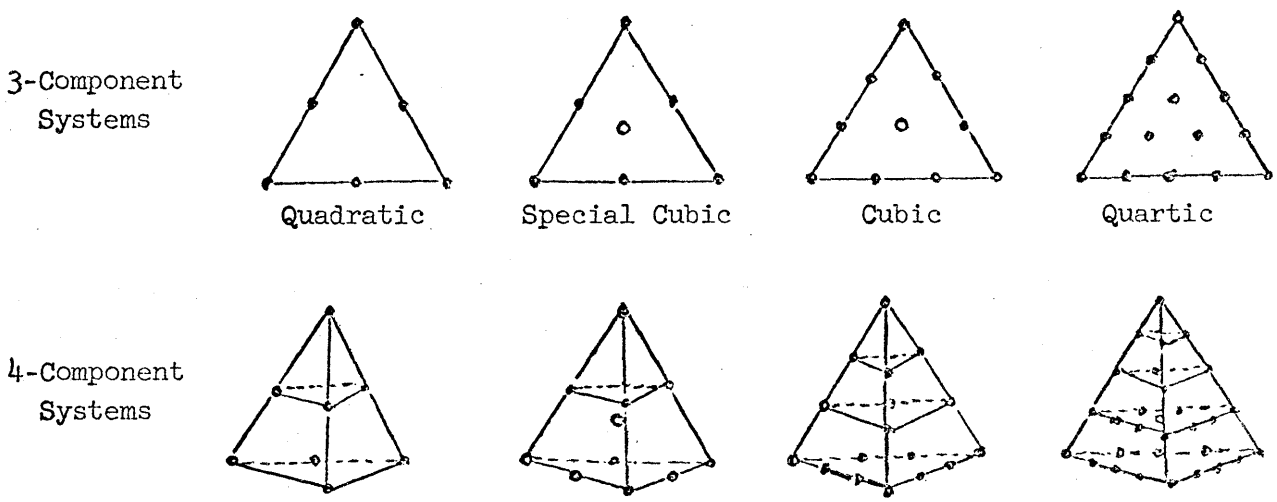
TABLE 1

Math Model Chosen / No. of Components	Quadratic		Special Cubic		Cubic		Special Quartic		Quartic		Special Quintic	
	P	T	P	T	P	T	P	T	P	T	P	T
2	3	2	-	-	4	2	-	-	5	2	-	-
3	6	3	7	3	10	3	-	-	15	4	-	-
4	10	4	11	4	20	4	31	5	38	5	-	-
5	18	5	19	5	40	5	58	6	84	6	97	6

The P-points are the experimental points. The T-points are the additional test-of-fit points, which must be made during initial runs to assure that the chosen mathematical model is sufficiently close to the true response equation. These points should also be run when switching to a different order equation in optimization runs.

It is important to distribute the points over the simplex study area with relatively equal spacing. For initial runs the Scheffe points shown in Figure 1 are mathematically very efficient and should be included where possible.

FIGURE 1



(The open points are centroids.)

Test-of-fit points should be equally spaced around the centroid of the field of study.

4. Making and Testing the Experimental Mixes

Mixes are prepared corresponding to the coordinates for the chosen points in the simplex area of study. For example, for a point whose (a, b, c) coordinates are (10, 60, 30), the mix would be 10% A, 60% B, and 30% C, where A, B and C may be either pure components or other mixtures.

Prepare the mixes in a randomized order and determine their properties.

5. First Computer Phase - Obtaining the Coefficients

The percentages of each component for each mix and the observed properties for that mix are fed into any standard multiple regression program. The following method is used:

a. Repress the constant normally generated by regression.
b. A separate run is made for each property, using the property response as Y.
c. Test-of-fit points are not included in initial runs.
d. Interaction terms are used according to the model equation chosen by the experimenter.

The coefficients derived are checked, using the test-of-fit points. If the variance is equal to or less than that for replication, the equation is acceptable. If not, a higher order equation must be used.

6. Second Computer Phase - Generation of Response Points

Upon achieving a satisfactory fit, program C-9655 (see Appendix B) is used to generate response points at the desired intervals. Input data is defined in the program listing. Again one run is made for each property.

The following comments regarding input should be helpful:

a. The value of "I" in B(I) is position of the accompanying variable in the program's equation, as listed in the heading. (B(I) is zero for unused variables.)
b. For more than three components, the experimenter must give the order desired for coordinate values. For example, if 0% D is to be the base of a four component tetrahedron, its coordinates must be fed in first. (i.e., D must be X1), then those for A, B and C.

The output of this program shows the simplex coordinates of the points of intersection of lines drawn from the interval points of one base to another, with the predicted value for the property at that intersection as the response.

7. Plotting the Response Points

For two component systems the responses are plotted on Cartesian paper using the Y-axis for the response and the X-axis for mix components.

With three components, triangular graph paper is used. The response value is written at the point represented by the mix coordinates.

For more than four components the simplex is examined like a three component mix at various interval levels, representing a constant value for the fourth and/or more component.

8. Interpretation for Optimum Area

Contour lines are drawn through points of equal response value. Examination of the resulting property map will reveal best areas for that particular property. Since it is unlikely that these will be in the same area for all properties, an added step is necessary to optimize the mix as a whole.

Superimposing different sets of contour lines on the same simplex will show where "optimums" for various properties come closest together. A new simplex is then drawn, with this point as the center, and reevaluated for better definition of the optimum area.

For example, if we had the following hypothetical situation:

There is to be a product made from a mixture of A, B and C. C is the most expensive ingredient, so we want to use as little as necessary. Properties 1 and 2 should be as high as possible, and property 3 must lie between 70 and 90. Using a special cubic model, the mixture has been evaluated, and the predicted responses have been plotted and superimposed as shown in Figure 2.

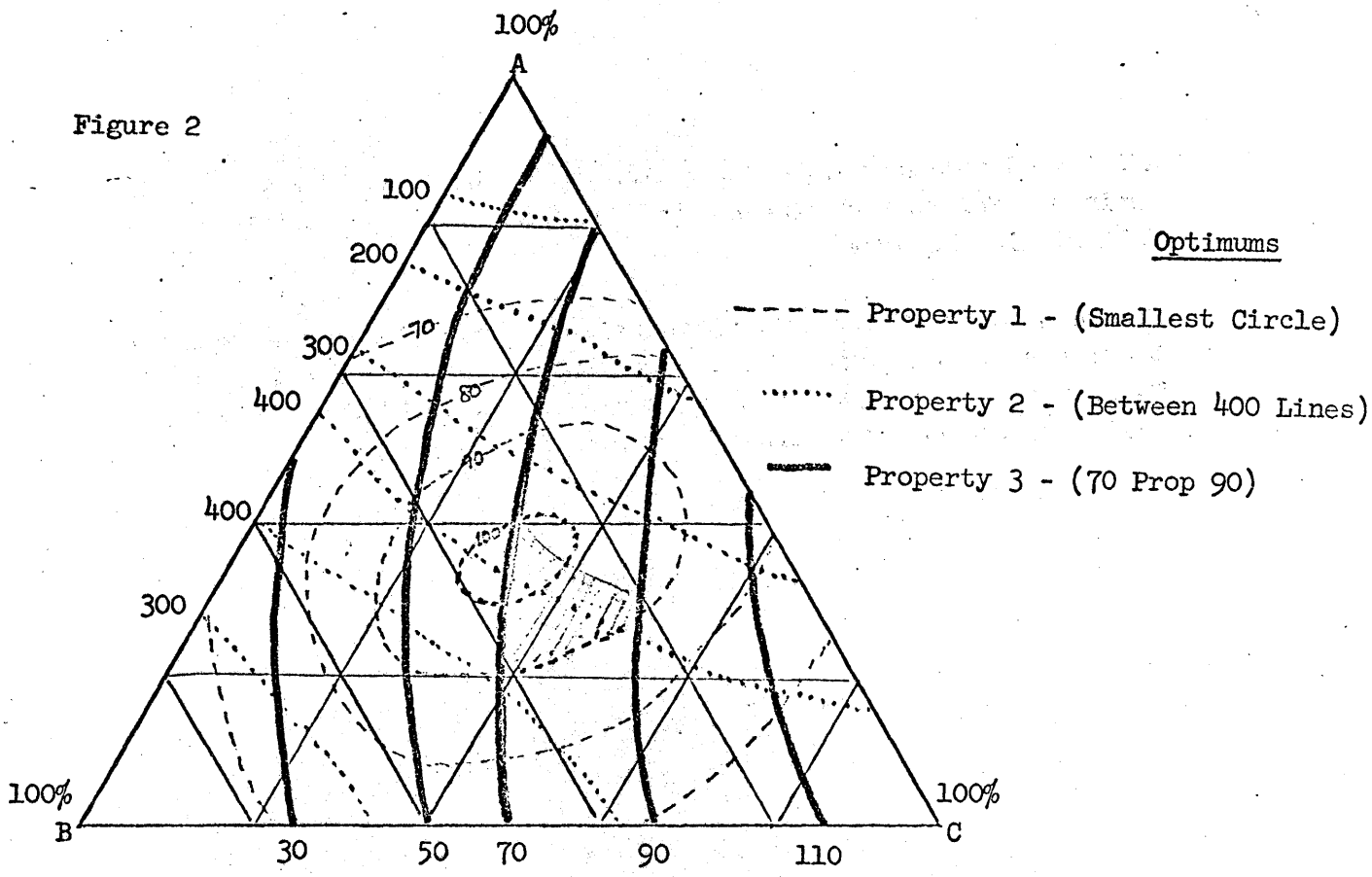
The shaded area in Figure 2 shows the apparent optimum area to be in the vicinity of 30% A, 30% B and 40% C.

The optimum area should then be defined more accurately by reducing the size of the simplex area under study. This has been done in Figure 3 by making a subspace simplex which encloses the apparent optimum. The new simplex is shown with its relationship to the old one represented by dotted lines. New points have been run with

additional mixtures made and evaluated in the immediate area of the optimum. Points from the earlier run may also be included if they fall within the area.

Figure 3 shows that reevaluation produced essentially the same optimum even though some of the porperties have changed slightly. These results, combined with a cost-analysis would give us a sound basis for choice of alternative product compositions.

Figure 2



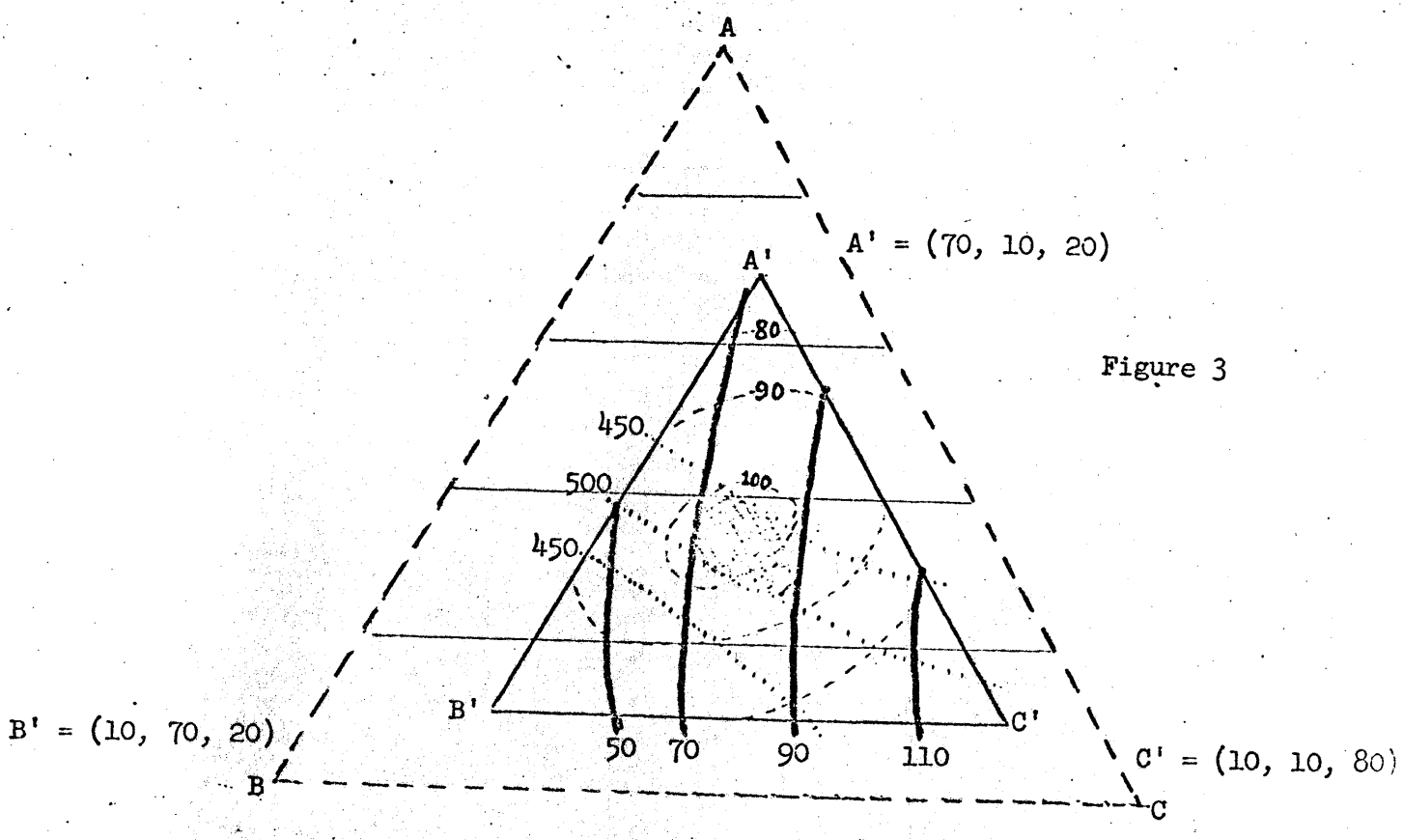


Figure 3

CONCLUSIONS

This technique can effectively contribute to the efficient screening of various alternatives involving mixtures. Afterwards, it may then be used to determine optimum formulations.

1620 time required is 15 minutes for a four-component system regression run. One run is made for each property to be optimized. The same computer needs 45 minutes for a response print out for the same system, using 10 as the interval on the simplex base line. Again one run is needed for each property. For the 1130 the times are 2 minutes and 5 minutes, respectively.

If a plotter is used, this will eliminate the time involved for manual plotting and the intermediate print-out of response points.

It is assumed in the above sample that the special cubic mathematical model is used.

Since many mix problems can be handled by this technique, it is felt that a general awareness of it will be profitable to user installations.

REFERENCES

1. Gorman, J. W. and Hinman, J. E.; "Simplex Lattices Designs for Multicomponent Systems," Technometrics 4:463-487(November 1962).

2. Scheffe, Henry; "Experiments with Mixtures," Journal of the Statistical Society. Series B, 20:344-369(1958).

3* Cruise, D. R.; "Plotting the Composition of Mixtures on Simplex Coordinates," Journal of Chemical Education, 43:30-31(January 1966).

4* Kurotori, I. S.; "Experiments with Mixtures of Components Having Lower Bounds," Industrial Quality Control, XXII(11):592-596 (May 1966).

*These were published after our report to Westvaco, upon which this paper is based, but are included for convenience of other users.

APPENDIX A

MATHEMATICAL MODELS USED

The following equations are the available models from which one may choose. The subscripts in the equations refer to the individual components. A multiple subscript indicates the presence, in the factor, of multiple components.

Models 1(a) and 1(b) will handle most situations.

1.(a) Quadratic: $y = \sum_{1 \le i \le n} \beta_i X_i + \sum_{1 \le i < j \le n} \beta_{ij} X_i X_j$ (n - No. of Components)

(b) Special Cubic: y = Same as 1.(a) $+ \sum_{1 \le i < j < k \le n} \beta_{ijk} X_i X_j X_k$

2.(a) Cubic: y = Same as 1.(b) $+ \sum_{1 \le i < j \le n} \alpha_{ij} X_i X_j (X_i - X_j)$

(b) Special Quartic: y = Same as 2.(a) $+ \sum_{1 \le i < j < k < e \le n} \beta_{ijke} X_i X_j X_k X_e$

(The last term is used only when $n > 3$)

3.(a) Quartic: y = Same as 1.(a) $+ \sum_{1 \le i < j < k \le n} \delta_{ij} X_i X_j (X_i - X_j)^2 + \sum \beta_{iijk} X_i^2 X_j X_k +$

$\sum \beta_{ijke} X_i X_j X_k X_e + \sum \beta_{ijjk} X_i X_j^2 X_k + \sum \beta_{ijkk} X_i X_j X_k^2 + \sum \alpha_{ij} X_i X_j (X_i - X_j)$

(b) Special Quintic: y = Same as 3.(a) $+ \sum_{1 \le i < j < k < e \; m \le n} \beta_{ijke} X_i X_j X_k X_e + \sum \beta_{ijkem} X_i X_j X_k X_m X_e$

Any of the coefficients in the above equations may be zero, and it must be emphasized that the lowest order equation, consistent with a good fit, should be used. The choice of a more complex equation unnecessarily distorts the estimated response surface, and makes the optimum difficult or impossible to find.

There are no constants in the equation for we are examining differences* between the mixes, not what they have in common. It is important to remember this, since our correlation programs for the computer normally generate a constant. In this case, however, the constant must be repressed by the computer operator, otherwise the coefficients will not apply to the simplex. The simplex represents 100% of the mix, but if some portion is predetermined (for economic reasons, for example), then the simplex represents 100% of the flexible portion of the mix.

---

*Full advantage of this characteristic of simplexes is taken in examining systems with four or more components. The systems are examined at each level for three factors while the fourth and fifth factors are held constant at these levels. This permits examining a four dimensional simplex on two dimensional triangular pilots, which simplified interpretation.

```
      PROGRAM 9655-CHR
      GENERATION OF RESPONSE POINTS FROM SIMPLEX OF 2 TO 5 COMPONENTS,
         USING EITHER ABSOLUTE VALUES OR DIFFERENTIALS.
      MATH MODELS THROUGH SPECIAL QUARTIC INCLUSIVE.
      WRITTEN BY WM. A. PEASE,JR.                    MARCH 2, 1967

      READ IN PARAMETER CARD CONTAINING N1,N2,N4,BP,NL,AND KB FIRST.
         THEN FOLLOW WITH NL HEADER CARDS TO LABEL AND SIX DATA CARDS
        - CONTAINING B(I) VALUES FROM --
        B1(X1)+B2(X2)+B3(X3)+B4(X1X2)+B5(X1X3)+B6(X2X3)+B7(X1X2X3)+
        B8(X4)+B9(X1X4)+B10(X2X4)+B11(X3X4)+B12(X1X2X4)+B13(X1X3X4)+
        B14(X2X3X4)+B15(X1X2X3X4)+B16(X5)+B17(X1X5)+B18(X2X5)+B19(X3X5)+
        B20(X1X2X5)+B21(X1X3X5)+B22(X2X3X5)+B23(X1X2X3X5)+B24(X4X5)+
        B25(X1X4X5)+B26(X2X4X5)+B27(X3X4X5)+B28(X1X2X4X5)+B29(X1X3X4X5)+
        B30(X2X3X4X5)+B31(X1X2X3X4X5)+B32((X1-X2)X1X2)+B33((X1-X3)X1X3)+
        B34((X2-X3)X2X3)+B35((X1-X4)X1X4)+B36((X2-X4)X2X4)+
        B37((X3-X4)X3X4)+B38((X1-X5)X1X5)+B39((X2-X5)X2X5)+
        B40((X3-X5)X3X5)+B41((X4-X5)X4X5)+B42

      N1 = NO. OF DIFFERENT COMPONENTS (DIFFERENT X) IN MODEL.
      N2= NO. OF INTERVALS IN SIMPLEX BORDER (AT BASE).
      N4= BASE ON ORIGINAL SCALE. (UNITY,10,OR 100-- FOR PER CENT)
      BP= BASE FOR PROPERTY(SMALLEST VALUE OBSERVED-WHOLE NO.)
      NL= NO. OF LABELING HEADER CARDS
      B(I)= COEFFICIENTS IN MODEL EQUATION.
      KB= 1 FOR PURE CUBIC OR SPECIAL QUARTIC,= 0 IN ALL OTHER CASES.
      X(I)= EXPRESSION IN PARENTHESIS FOR CORRESPONDING B(I) IN MODEL.
      B(42)= CONSTANT IN EQUATION, WHEN USED.
      X(42)= 1 TO KEEP CONSTANT IN EQUATION,WHEN DESIRED.
      A(I)= BASIC COMPONENT RATIO BEFORE CONVERTING FOR BASE.
      CONV= CONVERSION FACTOR TO BRING COMPONENT RATIOS TO ACTUAL AMT.
      AN= CONSTRAINT ON SUM OF COMPONENT VALUES.
      CT= CONSTRAINT TEST TO KEEP POINTS WITHIN SIMPLEX MODEL

      DIMENSION B(48),X(48),A(5)
    1 FORMAT (3I4,F9.0,2I3)
    2 FORMAT (8F9.0)
    4 FORMAT (5X3H X1,6X3H X2,6X3H X3,6X3H X4,6X3H X5,7X9H RESPONSE )
    5 FORMAT (5F9.2,3X,E16.9)
    6 FORMAT (/I4,11H COMPONENTS,4X,I4,10H INTERVALS, I4 ,5H BASE /)
    7 FORMAT (72H
     1                                                                )
    8 FORMAT (1H1)
    9 FORMAT (30H WHAT IS NUMBER OF COMPONENTS.)
   20 FORMAT (40H NO. OF COMPONENTS EXCEEDS 5, TOO LARGE.//)

   10 WRITE (3,8)
      READ (2,1)N1,N2,N4,BP,NL,KB
      IF (N1) 14,15,16
   14 CALL EXIT
   15 WRITE (3,9)
      PAUSE 1
      GO TO 10
   16 IF (N1-5) 18,18,17
   17 WRITE (3,20)
      PAUSE 2
      GO TO 10
   18 DO 11 I=1,NL
      READ (2,7)
```

```
11 WRITE (3,7)
C     INITIALIZATION OF VARIABLES
      DO 12 I=1,48
      X(I)= 0.0
12    B(I)= 0.0
      N3= N2+1
      AN= N2
      AN4=N4
      CONV= AN4/AN
C
      WRITE (3,6)N1,N2,N4
      WRITE (3,4)
      DO 13 I= 1,41,8
13    READ (2,2)B(I),B(I+1),B(I+2),B(I+3),B(I+4),B(I+5),B(I+6),B(I+7)
      X(42)= 1.
      N5= -1
      DO 50 I= 1,N3,1
      N5= N5+1
      N6= N3-N5
      A(1)= I-1
      X(1)= A(1)*CONV
      N7= -1
      DO 50 J= 1,N6,1
      N7= N7+1
      N8= N6-N7
      A(2)= J-1
      X(2)= A(2)*CONV
      X(4)= X(1)*X(2)
      IF (N1-3) 19,24,24
19    CT= A(1)+A(2)
      IF (CT-AN) 50,21,50
21    Y= 0.0
      DO 23 N= 1,2
23    Y= Y+(B(N)*X(N))
      Y= Y+B(4)*X(4)+B(42)+BP
      WRITE (3,5)X(1),X(2),X(3),X(8),X(16),Y
      GO TO 50
24    N9= -1
      DO 49 K= 1,N8,1
      N9= N9+1
      N10= N8-N9
      A(3)= K-1
      X(3)= A(3)*CONV
      X(5)= X(1)*X(3)
      X(6)= X(2)*X(3)
      X(7)= X(1)*X(6)
      IF (N1-4) 46,28,28
25    CT= A(1)+A(2)+A(3)
      IF (CT-AN) 49,26,49
26    Y= 0.0
      DO 27 N=1,7
27    Y= Y+B(N)*X(N)
      Y=Y+B(42)+BP
      WRITE (3,5)X(1),X(2),X(3),X(8),X(16),Y
      GO TO 49
28    N11= -1
      DO 48 L=1,N10
      N11= N11+1
      N12= N10-N11
      A(4)= L-1
      X(8)= A(4)*CONV
```

```
      DO 29 N=1,7
   29 X(N+8)= X(N)*X(8)
      IF (N1-5) 40,34,34
   30 CT= A(1)
      DO 31 N= 2,4
   31 CT= CT + A(N)
      IF (CT-AN) 48,32,48
   32 Y= 0.0
      DO 33 N= 1,15
   33 Y= Y+(B(N)*X(N))
      Y=Y+B(42)+BP
      WRITE (3,5)X(1),X(2),X(3),X(8),X(16),Y
      GO TO 48
   34 DO 47 M= 1,N12,1
      A(5)= M-1
      X(16)= A(5)*CONV
      DO 35 N= 1,15
   35 X(N+16)= X(N)*X(16)
      IF (N1-5) 40,40,17
   36 CT= A(1)
      DO 37 N= 2,5
   37 CT= CT+A(N)
      IF (CT-AN) 47,38,47
   38 Y= 0.0
      DO 39 N= 1,31
   39 Y= Y+(B(N)*X(N))
      Y=Y+B(42)+BP
      WRITE (3,5)X(1),X(2),X(3),X(8),X(16),Y
      GO TO 47
   40 IF (KB) 41,41,42
   41 GO TO (19,19,25,30,36),N1
   42 X(32)= (X(1)-X(2))*X(4)
      X(33)= (X(1)-X(3))*X(5)
      X(34)= (X(2)-X(3))*X(6)
      IF (N1-4) 25,43,43
   43 DO 44 LM= 1,3
   44 X(LM+34)= (X(LM)-X(4))*X(LM+8)
      IF (N1-5) 30,45,45
   45 DO 46 LM= 1,3
   46 X(LM+37)= (X(LM)-X(5))*X(LM+16)
      X(41)= (X(8)-X(5))*X(24)
      GO TO 36
   47 CONTINUE
   48 CONTINUE
   49 CONTINUE
   50 CONTINUE
      GO TO 10
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS
 EXTENDED PRECISION
 IOCS

CORE REQUIREMENTS FOR
 COMMON        0  VARIABLES     352  PROGRAM    1318

END OF COMPILATION
```

RESPONSE FOR PROPERTY 1, SPECIAL CUBIC MODEL

4 COMPONENTS 10 INTERVALS 1 BASE

D X1	A X2	B X3	C X4	X5	RESPONSE
.0	.0	.0	1.0	.0	[illegible]
.0	.0	.1	.9	.0	[illegible]
.0	.0	.2	.8	.0	[illegible]
.0	.0	.3	.7	.0	[illegible]
.0	.0	.4	.6	.0	[illegible]
.0	.0	.5	.5	.0	[illegible]
.0	.0	.6	.4	.0	[illegible]
.0	.0	.7	.3	.0	[illegible]
.0	.0	.8	.2	.0	[illegible]
.0	.0	.9	.1	.0	[illegible]
.0	.0	1.0	.0	.0	[illegible]
.0	.1	.0	.9	.0	[illegible]
.0	.1	.1	.8	.0	[illegible]
.0	.1	.2	.7	.0	[illegible]
		⋮			
.0	.1	.9	.0	.0	[illegible]
.0	.2	.0	.8	.0	[illegible]
.0	.2	.1	.7	.0	[illegible]
.0	.2	.2	.6	.0	[illegible]
		⋮			
.0	.2	.8	.0	.0	[illegible]
.0	.3	.0	.7	.0	[illegible]
.0	.3	.1	.6	.0	[illegible]
.0	.3	.2	.5	.0	[illegible]
		⋮			
.0	.3	.7	.0	.0	[illegible]
.0	.4	.0	.6	.0	[illegible]
.0	.4	.1	.5	.0	[illegible]
.0	.4	.2	.4	.0	[illegible]
		⋮			
.0	.4	.6	.0	.0	[illegible]
.0	.4	.0	.5	.0	[illegible]
.0	.4	.1	.4	.0	[illegible]
.0	.4	.2	.3	.0	[illegible]
.0	.4	.3	.2	.0	[illegible]
.0	.4	.4	.1	.0	[illegible]
.0	.4	.5	.0	.0	[illegible]
.0	.4	.0	.4	.0	[illegible]
.0	.4	.1	.3	.0	[illegible]
.0	.4	.2	.2	.0	[illegible]
.0	.4	.3	.1	.0	[illegible]
		⋮			

for 2 components (at interval chosen) the printout would end here.

.0	.8	.2	.0	.0	.21037680E+03
.0	.9	.0	.1	.0	.91676200E+02
.0	.9	.1	.0	.0	.18389020E+03
.0	1.0	.0	.0	.0	.15399000E+03
.1	.0	.0	.9	.0	.14150846E+04
.1	.0	.1	.8	.0	.13408807E+04
.1	.0	.2	.7	.0	.12543193E+04

← for 3 components, the printout would end here

⋮

.1	.0	.9	.0	.0	.30237860E+03
.1	.1	.0	.8	.0	.10977482E+04
.1	.1	.1	.7	.0	.10086174E+04
.1	.1	.2	.6	.0	.91683288E+03

⋮

.1	.1	.8	.0	.0	.31041940E+03
.1	.2	.0	.7	.0	.82498584E+03
.1	.2	.1	.6	.0	.73063186E+03
.1	.2	.2	.5	.0	.64332908E+03

⋮

.1	.3	.7	.0	.0	.31259836E+03
.1	.3	.0	.6	.0	.59679728E+03
.1	.3	.1	.5	.0	.50692535E+03
.1	.3	.2	.4	.0	.43331105E+03

⋮

.1	.9	.0	.1	.0	.12446268E+03
.1	.9	.1	.0	.0	.20257188E+03
.1	.9	.0	.0	.0	.16371740E+03
.2	.0	.0	.8	.0	.11741224E+04
.2	.0	.1	.7	.0	.10911769E+04
.2	.0	.2	.6	.0	.10017235E+04

⋮

.2	.0	.7	.1	.0	.44213884E+03
.2	.0	.0	.0	.0	.35873840E+03
.2	.1	.0	.7	.0	.30561704E+03
.2	.1	.1	.6	.0	.26460010E+03
.2	.1	.2	.5	.0	.72625620E+03

⋮

.2	.1	.7	.0	.0	.32154950E+03
.2	.2	.0	.6	.0	.67827104E+03
.2	.2	.1	.5	.0	.58906360E+03
.2	.2	.2	.4	.0	.51205776E+03
.2	.2	.3	.3	.0	.44725350E+03
.2	.2	.4	.2	.0	.39465038E+03
.2	.2	.5	.1	.0	.35424980E+03
.2	.2	.6	.0	.0	.32605040E+03

⋮

.2	.3	.0	.5	.0	.49208400E+03
.2	.3	.1	.4	.0	.41430310E+03
.2	.3	.2	.3	.0	.35042812E+03
.2	.3	.3	.2	.0	.32445946E+03
			⋮		
.3	.5	.2	.0	.0	.28653991E+03
.3	.6	.0	.1	.0	.19508248E+03
.3	.6	.1	.0	.0	.24788056E+03
.3	.7	.0	.0	.0	.19846260E+03
.4	.0	.0	.6	.0	.79504760E+03
.4	.0	.1	.5	.0	.71006780E+03
.4	.0	.2	.4	.0	.62817952E+03
			⋮		
.4	.4	.0	.2	.0	.27668152E+03
.4	.4	.1	.1	.0	.27354956E+03
.4	.4	.2	.0	.0	.31232800E+03
.4	.5	.0	.1	.0	.23291500E+03
.4	.5	.1	.0	.0	.27450760E+03
.4	.6	.0	.0	.0	.22360400E+03
.5	.0	.0	.5	.0	.65693500E+03
.5	.0	.1	.4	.0	.57066260E+03
.5	.0	.2	.3	.0	.50063140E+03
.5	.0	.3	.2	.0	.34604140E+03
			⋮		
.5	.4	.0	.1	.0	.27243140E+03
.5	.4	.1	.0	.0	.30378320E+03
.5	.5	.0	.0	.0	.25359500E+03
.6	.0	.0	.4	.0	.25310500E+03
.6	.0	.1	.3	.0	.48669028E+03
.6	.0	.2	.2	.0	.43366594E+03
			⋮		
.7	.3	.0	.0	.0	.32911460E+03
.8	.0	.0	.2	.0	.44829640E+03
.8	.0	.1	.1	.0	.42164420E+03
.8	.0	.2	.0	.0	.41748240E+03
.8	.1	.0	.1	.0	.40107100E+03
.8	.1	.1	.0	.0	.40750100E+03
.8	.2	.0	.0	.0	.27450980E+03
.9	.0	.0	.1	.0	.44731660E+03
.9	.0	.1	.0	.0	.44737060E+03
.9	.1	.0	.0	.0	.42502140E+03
1.0	.0	.0	.0	.0	.48962000E+03

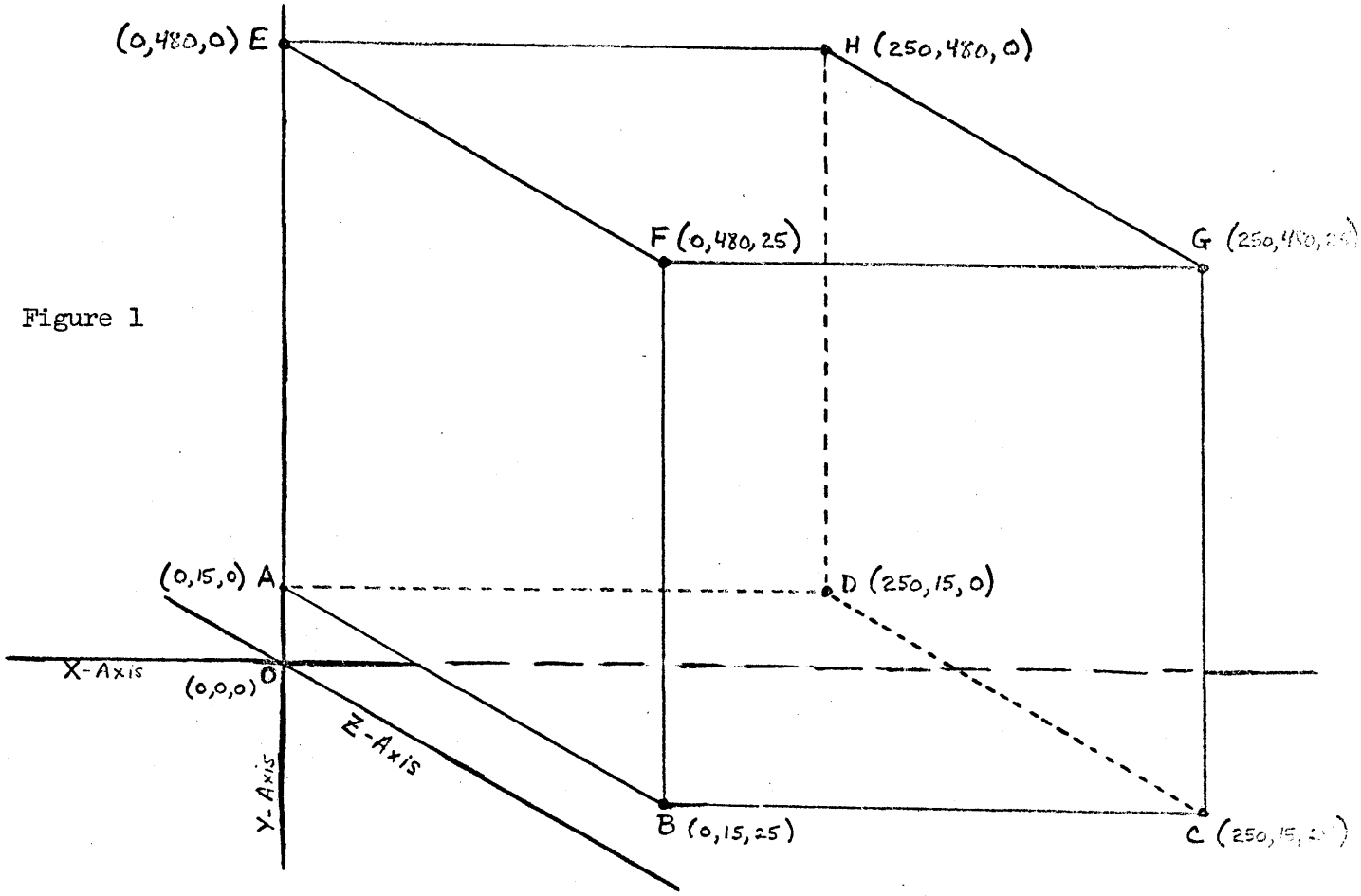
Notice that only points whose coordinates add up to unity are printed by the computer. (The five pages from which this sample was taken were generated in 45 minutes on the 1620 computer.)

Factor Spaces and the Simplex

In any experiment one usually has upper and lower limits upon the various conditions or ingredients. The space bounded by these limits is called a "factor space."

For example, Figure 1 represents such a space. Here the X-axis is temperature (ranging from 0°C - 250°C), the Y-axis is time (ranging from 15 min. to 480 min) and the Z-axis is percent formaldehyde (ranging from 0 to 25%).

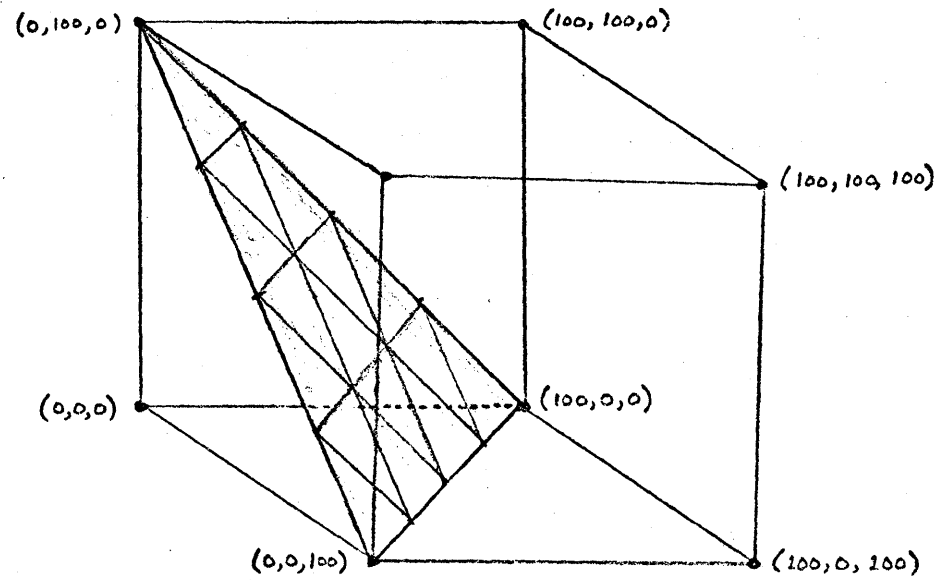
Figure 1



This could represent the factorial space for a resin. Now every formulation within the given limits is to be found within this factor space. Each point may be expressed in terms of factor space. ~~Thus~~ an equation that will permit one to optimize that property. Finding the maximum or minimum value for the equation in this range is done by "mapping" the response surface of this space. Like the contour lines of a map for elevations, this will show the high and low regions.

If we only consider ingredients, however, and omit time and temperature, we know that the ingredients must add up to 100%. A mix that is 40% Phenol, 40% Water and 40% Formalin has no real world meaning. It does have a place in the factor space, however. To keep the points in the real world, then we must put the constraint that the point coordinates sum to 100% upon our space. Figure 2 shows the results. The cube is the factor space.

Figure 2



Only the points on the shaded triangular plane meet the last constraint. The space represented by this surface is a simplex space. A simplex has these characteristics:

1. It is represented by a equation one degree lower than that of the factor space from which it is cut.
2. It thus has one less dimension for diagramatic representation.
3. The sum point within the simplex adds up to the same sum (the sum of the coordinates at any of its vertices).

As you can see, all feasible points are completely within the simplex. By eliminating the rest of the factor space we have greatly reduced the number of examinations necessary, removed meanless combinations from the model equation and reduced the equation representing the property by one degree.

Now examination of a simplex can be done all at once, while but it is harder to do this on a factor space. In IBM's Control Optimization Program (COP), you will find that the program moves over a small area at any one time and homes in on any local optimums. ~~As IBM will~~ explain, one must start from other points on the perimeter of the factor space to make sure that the local optimum found is the optimum for the whole factor space. In

using the simplex, however, one can examine the whole area accurately enough to home in on the property optimum for the whole space.

DYSTAL: A Dynamic Storage Allocation Language in FORTRAN

James M. Sakoda
Brown University

DYSTAL II is a revised version of DYSTAL and is written in ASA Basic FORTRAN IV. It is particularly useful to FORTRAN programmers in need of special programming language capabilities and dynamic storage and virtual memory using a disk file. It can be used in a time-sharing system, such as RAX, which employs FORTRAN as its basic language. It can be used with relatively small machines for which special languages or the more general purpose PL/I generally are not available. DYSTAL was originally written for the IBM 7070 with 5 K words of memory, and rewritten for the IBM 360 Mod 50, and is now in operation on an IBM 1130 with a disk file and 8K half words of core memory. Since it consists of a series of some 70 FORTRAN subroutines, it can be adapted to other machines.

DYSTAL has three features which are of interest to users who desire more than the capabilities normally provided by FORTRAN. First, it has a dynamic storage allocator, which allocates space in core memory and moves arrays from core to a disk file and back again. Second, it is capable of handling complex data structures--in particular tree structures and chains of arrays. Third, it has, among its subroutines, procedures for list processing, string processing, ranking and sorting, matrix operations, statistical operations, and high level input-output routines. It can be used to advantage to write complex programs in such fields as linguistics, data-processing, simulation.

Dynamic Storage Allocation

DYSTAL II, which features relocatable arrays, provides both dynamic storage allocation within core memory, and movement of arrays from core memory to a disk file as room in core becomes unavailable. Dynamic storage allocation allows the user to specify the length of arrays or dimensions of matrices at running time, thus eliminating the need to write a DIMENSION statement for the maximum amount likely to be used. It is also possible to input or create arrays or matrices, the number of which need not be specified at the time a program is written, but can be determined when the program is run. The user keeps track of such arrays by keeping their names on a name array--a feature which FORTRAN does not provide. For example, in XTAB6, the recoding and cross-tabulation program, the number of instructions for recoding and table-making can only be specified at running time. Also, the number and sizes of tables to be tabulated cannot be anticipated in advance. Dynamic storage allocation makes it possible to read in any number of arrays as instructions or to create as many matrices of the desired size as necessary.

DYSTAL II's relocatability permits an expansion of an array should it become necessary. All arrays are accessed through a directory called MAP. When an array is inadequate in size it can be moved to a new location with expanded capacity, and this change noted in the directory. For example, when the instruction

CALL LOAD (WD, LSTA)

is used, the content of WD is placed in the next available location of an array called LSTA, and the counter increased by one. When the capacity of the array is insufficient for LOAD to operate, it calls KMOVE which moves the array to a new location with expanded capacity, and permits LOAD to complete its task. Relocatability also makes it possible to provide virtual memory or two-level store. When room in core memory is exhausted, the array which has been in core the longest is removed and placed on a disk file, and only brought back when needed. Arrays in core are handled on a first-in-core, first-out-of-core basis. Thus room is made for a newly created array. It is therefore possible to deal with a relatively large number of arrays, many of which can reside on a disk file without the user's knowledge. The limitation of the size of the dynamic storage area, of course, prevents the use of very large arrays.

There are two difficulties with the dynamic storage allocation system. First, the system is a little large for an 8K (4K full words) machine, and would work better with a slightly larger machine. The maximum amount of core memory which we have been able to allocate for dynamic storage is about 1400 full words and this only by having separate links for input, calculation and output phases, which permits the shutting off of unneeded devices, as well as shortening the main program. Full use also must be made of the LOCAL which allows subroutines to be called in from disk only when needed. This limitation is due to the fact that the dynamic storage allocation routines by and large must remain in core memory at all times. The second difficulty is that when the disk file is used constantly the operations are slowed down considerably. Much of this is due to access time on the disk, and it is partially the responsibility of the programmer to reduce the frequency of disk access.

I am now working on three approaches to reduce the amount of disk access. The first is to have the system avoid writing an array back on disk when an identical copy resides there already. The second is to allow for permanent designations of arrays as permanent, semi-permanent or temporary. The permanent arrays will stay permanently in core, the

semi-permanent ones will go on disk on a first-in-core, first-out-of core basis, while the temporary ones will be processed on a last-in, first-out scheme. The third approach is to provide for blocking of several arrays into larger units, making possible a single access for a group of arrays. Blocking can be specified for either fixed length or variable length arrays. The use of blocking will be more feasible when we get in our additional 8 K half words of core memory.

Complex Data Structure

Complex programming problems are often problems of structuring data. One of the features of DYSTAL arrays is a seven-word head accessible as a part of the array. The head contains parameters, such as the input-output mode, array counter, matrix dimension, pointer to another array, etc. which makes it possible to turn many programming chores over to DYSTAL routines. In other words, the DYSTAL array is a complex structure consisting of a head and body. The input-output modes, for example, are associated with fixed input-output formats for integers, real numbers, alphabetic words, alphabetic text, etc. I have just added a seventh for alphameric character input-output.

Another important feature is the ability to place names of DYSTAL arrays on other arrays. This permits the creation of complex data structures, such as a tree structure. In our example, the generalized input routine, LSRD, reads in as many arrays of different types and lengths as specified, and puts their names on a name array, from which each array can be accessed. This process of putting names of arrays on other arrays can be carried out to any depth desired. There is a routine which traces through a tree structure and returns the name of each array as it is encountered.

Another type of structure which can be created is the chain of arrays, by use of a pointer from one array to another. Waiting lines, consisting of arrays, for example, can be handled as chains of arrays.

Special Purpose Routines

Finally, DYSTAL can be seen as a general purpose language consisting of a collection of subroutines providing special purpose capabilities. DYSTAL's list processing operations, for example, permit many of the operations performed by list processors such as IPL-V or SLIP. DYSTAL, however, keeps words of arrays in consecutive locations in storage, thus permitting

use of subscripts for faster accessing than is possible with linked word lists. It is possible to insert or delete a word from a list. In our bridge hand example the card which has been dealt is deleted from the list of cards, thus making it possible to draw a random number for the remaining cards, until all cards are dealt. The instruction for this is

NVAL = ITAKE (NTH, LDRAW)

DYSTAL's string processing operations provide for unpacking words into a string of characters, and also packing characters into words. A string of characters on one array can be matched against a longer string, and the location of the match returned when agreement between the arrays are found. A substring of characters can be replaced by another string, and two strings can be concatenated to make a longer string. In our bridge example, JOIN, the concatenation function is used to combine into a long string. In our bridge example, JOIN, the concatenation function is used to combine into a long string a value, the word "of", the name of a suit, and a blank.

DYSTAL provides an internal sort routine, which uses a relatively efficient merge sort method, using an extra array and accessing items from both ends of the two arrays. The ranking routine replaces items with ranks from 1 to N. The record sort routine provides for the specification of one or more key items of the record on which sorts are to be based. The names of records on a name array are then sorted using the specified keys. During the sort many of the records can reside on the disk file.

DYSTAL's statistical operations provide the basic capabilities of calculating the sum, sum of squares, sum of crossproducts and variance of items on an array. The matrix operations include matrix transpose, matrix inverse, matrix multiplication, and row by row multiplication. DYSTAL's matrix operations are relatively simple to write because the dimensions of the matrices involved need not be specified as arguments, since they are contained in the head of each matrix. For example, to take the inverse of MA multiplied by MB and stored in MC one can write:

CALL MINV (MATMP (MA, MB, MC))

In our bridge hand example the values and suits are stored as matrices, and the expression LINE (ISUIT, LSUIT) is used to access the Ith line of the matrix LSUIT.

DYSTAL provides high level input-output routines which relieves the programmer of the chore of writing format statements. For example, LSRD (NRD, NAME) will read in any number of arrays or matrices, tree structures or chains of arrays and put the names of arrays on NAME. IDUMP (LSTA) can be used to print out the head and content of the array named LSTA in the proper mode. If the head is not desired the content of an array can be printed but by writing

CALL LWR N(NPR, 1, ITEM (0, LOUT), LOUT)

Here, ITEM (0, LOUT) retrieves the counter value of LOUT, which can be used when the number is not known exactly to the programmer. CALL KDUMP results in the systematic printing out of all arrays in memory, including those which have been put on the disk file. It is a very convenient method of getting a readable dump.

Possibly the most attractive feature of DYSTAL is its implementation. Since it is written in Basic FORTRAN it can be implemented as a series of FORTRAN subroutines and functions. Existing routines can be modified and more can be added to suit the user's needs. It is not only a general purpose language as it stands, but also an expandable one. For ten dollars the Sociology Computer Laboratory of Brown University provides a minitape which contains the FORTRAN subroutines for both the 360 and the 1130, and documentation for DYSTAL II. The older DYSTAL Manual for the older version of DYSTAL is still available at #3 and can be used in conjunction with the DYSTAL II manual. The two systems, however, are not exactly compatible.

The bridge hand example, which is included here, gives some of the flavor of DYSTAL programming. As can be seen, a DYSTAL program is a combination of FORTRAN and DYSTAL functions and subroutines. Only a single large dynamic storage area is dimension, placed in common and LOT and FLOT equivalenced. INLOT is called to set up the dynamic storage for use, and LSTAL is used to create an array. LSRD will both create and input arrays. Arrays whose names have been placed on NAME can be retrieved by writing, for example,

LVAL = ITEM (1, NAME).

LOAD is used to put numbers from 1-52 into LDRAW. A random number generator is then used to determine the card to be dealt. This card is deleted from LDRAW, and the correct line in LVAL and LSUIT determined, to form the output line through concatenation of words from several arrays. To output a line LWR is called which does the printing using a one character per word format. This should provide some idea of the dynamic storage allocation, data structuring, and special language ability offered by DYSTAL.

```
// EJECT
C       ** PROGRAM TO DEAL OUT A BRIDGE HAND.  WRITTEN IN DYSTAL.
      DIMENSION LOT(550),FLOT(550)
      COMMON LOT
      EQUIVALENCE (LOT(1),FLOT(1)),(NRD,LOT(27)),(NPR,LOT(29))
      DEFINE FILE 4 (1000,50,U,JFI)
      CALL INLOT(3,10,550)
C     ** CREATE OUTPUT LIST.
      CALL LSTAL (7,92,LOUT)
C     ** READ IN AN ODD NUMBER TO START RANDOM NUMBER GENERATOR.
      READ(NRD,11) ISEED
   11 FORMAT (9X15)
      CALL LSRD(NRD,NAME)
C     ** FIRST LIST IS LVAL, ACE, 2, 3, JACK, QUEEN, KING IN MODE 7
C     ** WHICH HAS AN (1X72A1) FORMAT.
      LVAL=ITEM(1,NAME)
C     ** SECOND LIST IS LSUIT--CLUBS,DIAMONDS,HEART,SPADES,BLANK
      LSUIT=ITEM(2,NAME)
C       ** THIRD LIST IS LDRAW--CONTAINS NUMBER OF CARDS TO BE DRAWN.
      LDRAW=ITEM(3,NAMF)
C     ** FOURTH LIST CONTAINS OF.
      LOF=ITEM(4,NAME)
    5 WRITE(NPR,310)
  310   FORMAT(1H1,10X'WEST',16X,'NORTH',16X,'EAST'16X,'SOUTH'/)
C     ** PUT CONSECUTIVE NUMBERS IN LDRAW.
      N=ITEM(-1,LDRAW)
      DO 15 I=1,N
        CALL LOAD(I,LDRAW)
   15 CONTINUE
C     ** ZERO OUT COUNTER OF LOUT
   20 CALL IPUT(0,0,LOUT)
      DO 95 J=1,4
      NDRAW = ITEM(0,LDRAW)
      IF(NDRAW)200,200,50
   50 CALL RAN(ISEED,RAND)
       NTH=RAND*FLOAT(NDRAW)+1.
C     ** DELETE NTH VALUE
      NVAL=ITAKE(NTH,LDRAW)
C     ** DETERMINE SUIT AND VALUE TO PRINT OUT
      ISUIT=(NVAL-1)/13+1
      IVAL=NVAL-(ISUIT-1)*13
        CALL JOIN(5,LINE(IVAL,LVAL),LOUT)
        CALL JOIN(4,LOF,LOUT)
      CALL JOIN (8,LINE(ISUIT,LSUIT),LOUT)
      CALL JOIN(6,LINE(5,LSUIT),LOUT)
   95 CONTINUE
      CALL LWR(NPR,1,ITEM(0,LOUT),LOUT)
      GO TO 20
  200 WRITE(NPR,210)
  210 FORMAT(//' OUT OF CARDS')
      PAUSE 111
      CALL KDUMP
      GO TO 5
      END
```

INPUT

```
// XEQ JMSRT     1
*LOCALJMSRT,INLOT,LSRD,KDUMP,JOIN
          1113
          NAME                                1              10
        1 LVAL              -5                7              65              65
 ACE
 2
 3
 4
 5
 6
 7
 8
 9
 10
 JACK
 QUEEN
 KING
        2 LSUI              -8                7              40              40
 CLUBS
 DIAMONDS
 HEARTS
 SPADES

        3 LDRA                                2              52
        4 LOF                                 7               4               4
 OF
          STOP
```

OUTPUT

WEST		NORTH		EAST		SOUTH	
4	OF HEARTS	4	OF DIAMONDS	5	OF DIAMONDS	ACE	OF HEARTS
6	OF HEARTS	8	OF DIAMONDS	ACE	OF SPADES	KING	OF CLUBS
8	OF HEARTS	3	OF HEARTS	3	OF DIAMONDS	KING	OF HEARTS
2	OF HEARTS	6	OF DIAMONDS	3	OF CLUBS	7	OF DIAMONDS
10	OF CLUBS	7	OF HEARTS	4	OF SPADES	JACK	OF SPADES
9	OF HEARTS	QUEEN	OF HEARTS	8	OF CLUBS	KING	OF SPADES
10	OF DIAMONDS	QUEEN	OF CLUBS	JACK	OF DIAMONDS	6	OF CLUBS
5	OF SPADES	7	OF CLUBS	9	OF CLUBS	4	OF CLUBS
2	OF DIAMONDS	9	OF SPADES	5	OF HEARTS	KING	OF DIAMONDS
QUEEN	OF SPADES	ACE	OF CLUBS	5	OF CLUBS	7	OF SPADES
2	OF SPADES	ACE	OF DIAMONDS	8	OF SPADES	6	OF SPADES
9	OF DIAMONDS	2	OF CLUBS	3	OF SPADES	JACK	OF CLUBS
JACK	OF HEARTS	10	OF HEARTS	10	OF SPADES	QUEEN	OF DIAMONDS

0 OF CARDS

```
// EJECT
C       ** PROGRAM TO DEAL OUT A BRIDGE HAND.  WRITTEN IN DYSTAL.
      DIMENSION LOT(550),FLOT(550)
      COMMON LOT
      EQUIVALENCE (LOT(1),FLOT(1)),(NRD,LOT(27)),(NPR,LOT(29))
      DEFINE FILE 4 (1000,50,U,JFI)
      CALL INLOT(3,10,550)
C     ** CREATE OUTPUT LIST.
      CALL LSTAL (7,92,LOUT)
C     ** READ IN AN ODD NUMBER TO START RANDOM NUMBER GENERATOR.
      READ(NRD,11) ISEED
   11 FORMAT (9XI5)
      CALL LSRD(NRD,NAME)
C     ** FIRST LIST IS LVAL, ACE, 2, 3, JACK, QUEEN, KING IN MODE 7
C     ** WHICH HAS AN (1X72A1) FORMAT.
      LVAL=ITEM(1,NAME)
C     ** SECOND LIST IS LSUIT--CLUBS,DIAMONDS,HEART,SPADES,BLANK
      LSUIT=ITEM(2,NAME)
C       ** THIRD LIST IS LDRAW--CONTAINS NUMBER OF CARDS TO BE DRAWN.
      LDRAW=ITEM(3,NAME)
C     ** FOURTH LIST CONTAINS OF.
      LOF=ITEM(4,NAME)
    5 WRITE(NPR,310)
  310   FORMAT(1H1,10X'WEST',16X,'NORTH',16X,'EAST'16X,'SOUTH'/)
C     ** PUT CONSECUTIVE NUMBERS IN LDRAW.
      N=ITEM(-1,LDRAW)
      DO 15 I=1,N
        CALL LOAD(I,LDRAW)
   15 CONTINUE
C     ** ZERO OUT COUNTER OF LOUT
   20 CALL IPUT(0,0,LOUT)
      DO 95 J=1,4
      NDRAW = ITEM(0,LDRAW)
      IF(NDRAW)200,200,50
   50 CALL RAN(ISEED,RAND)
       NTH=RAND*FLOAT(NDRAW)+1.
C     ** DELETE NTH VALUE
      NVAL=ITAKE(NTH,LDRAW)
C     ** DETERMINE SUIT AND VALUE TO PRINT OUT
      ISUIT=(NVAL-1)/13+1
      IVAL=NVAL-(ISUIT-1)*13
        CALL JOIN(5,LINE(IVAL,LVAL),LOUT)
        CALL JOIN(4,LOF,LOUT)
      CALL JOIN (8,LINE(ISUIT,LSUIT),LOUT)
      CALL JOIN(6,LINE(5,LSUIT),LOUT)
   95 CONTINUE
      CALL LWR(NPR,1,ITEM(0,LOUT),LOUT)
      GO TO 20
  200 WRITE(NPR,210)
  210 FORMAT(//' OUT OF CARDS')
      PAUSE 111
      CALL KDUMP
      GO TO 5
      END
```

INPUT

```
// XEQ JMSRT     1
*LOCALJMSRT,INLOT,LSRD,KDUMP,JOIN
          1113
          NAME                          1          10
      1   LVAL          -5              7          65          65
ACE
2
3
4
5
6
7
8
9
10
JACK
QUEEN
KING
      2   LSUI          -8              7          40          40
CLUBS
DIAMONDS
HEARTS
SPADES

      3   LDRA                          2          52
      4   LOF                           7           4           4
 OF
          STOP
```

OUTPUT

```
          WEST                NORTH               EAST                SOUTH

4      OF HEARTS        4      OF DIAMONDS      5      OF DIAMONDS      ACE    OF HEARTS
6      OF HEARTS        8      OF DIAMONDS      ACE    OF SPADES        KING   OF CLUBS
8      OF HEARTS        3      OF HEARTS        3      OF DIAMONDS      KING   OF HEARTS
2      OF HEARTS        6      OF DIAMONDS      3      OF CLUBS         7      OF DIAMONDS
10     OF CLUBS         7      OF HEARTS        4      OF SPADES        JACK   OF SPADES
9      OF HEARTS        QUEEN  OF HEARTS        8      OF CLUBS         KING   OF SPADES
10     OF DIAMONDS      QUEEN  OF CLUBS         JACK   OF DIAMONDS      6      OF CLUBS
5      OF SPADES        7      OF CLUBS         9      OF CLUBS         4      OF CLUBS
2      OF DIAMONDS      9      OF SPADES        5      OF HEARTS        KING   OF DIAMONDS
QUEEN  OF SPADES        ACE    OF CLUBS         5      OF CLUBS         7      OF SPADES
2      OF SPADES        ACE    OF DIAMONDS      8      OF SPADES        6      OF SPADES
9      OF DIAMONDS      2      OF CLUBS         3      OF SPADES        JACK   OF CLUBS
JACK   OF HEARTS        10     OF HEARTS        10     OF SPADES        QUEEN  OF DIAMONDS

OUT OF CARDS
```

SESSION: MGA - Philadelphia Common - 9/9/68

DIVISION: Systems

PROJECT: DOS 360

SUBJECT: PL/I - Funny Things Happen

AUTHORS: R. Fry and W. DeWald
American Electronic Laboratories, Inc.
P. O. Box 552
Lansdale, Pennsylvania

PL/I

"FUNNY THINGS HAPPEN"

INTRODUCTION

The introduction of a new computing language naturally generates considerable interest among potential users. The announcement of PL/I generated considerable interest and words. Some writers attempt to define PL/I, while others are busy defending it. At this stage in its development, PL/I needs both as its allies! In this report, we will neither define nor defend PL/I. Instead, we shall merely attempt to describe what happened to us as users.

Unlike the theorists, we have actually used PL/I to reach specific, and practical solutions. Our observations are based on our personal experiences in solving real problems. The solution of these very basic problems has led us through somewhat unique experiences. It is no understatement to say that we have seen "Funny Things Happen"!

1. PL/I Aims:

The goals of the new language have been expressed this way:*

1. To serve the needs of an unusually large group of programmers. In particular, the committee constantly attempted to encompass among its users the scientific, commercial, real-time, and systems programmers and to allow both the novice and the expert to find facilities at his own level.

2. To take a simple approach which would permit a natural description of programs so that few errors would be introduced during the transcription from the program formulation into NPL.

---

*Communications of the ACM, Jan. 1965 - NPL
"Highlights of a New Programming Language" - George Radin and Paul Rogoway

3. To provide a programming language for contemporary (and, perhaps future) computers, monitors, and applications. As a frequent benchmark, the committee chose not the familiar "can we write NPL in NPL?"; but can we write, in NPL, a real-time operating system to support NPL programs (i.e., an NPL language machine)?"

Almost from the beginning, this new language was conceived as having "features not generally present in higher level languages". That early design followed certain broad criteria:

1. The new compiler must have no "permissive" diagnostics. (This is wrong, but I am so smart that I know what you really mean.)

2. Assembly language for a system which supports the new language, can be forgotten; by definition, the programmer should never have to use it. (By the way, if he does, then PL/I has failed as a criterion for this sytem.)

3. There must be a free, unpenalizing selection of program modules. "It's there if you need it, but if you don't need it, you don't have to specify it or even learn about its existence, and you don't get penalized in compile time or object time efficiency". (Every expression, variable, attribute, and specification must be given a default option!)

4. A certain computer and programmer independence must be built in. This is to say that regardless of computer word-size, I/O configuration, or programmer's experience, the system must be designed so that it can be used in a way that is most natural to both the machine and the man. At last, the listing would be more readable and the chance for coding and keypunch errors would be greatly minimized.

2. <u>The Chronicle of a User</u>:

1. <u>Why PL/I</u>?

Why would a small computer installation contemplating the solution of a large problem choose a new programming language? The answer to this question

is the beginning of our strange tale, for once we choose PL/I - funny things happened! I suppose we really chose PL/I out of a kind of inventive desperation. We could have chosen either BAL or Fortran. We knew how tedious it was to code in any assembly language. And we also knew that BAL was not the easiest of assembly languages to use. To complicate matters, ours was an R & D problem. It was of a magnitude and complexity that was impossible to predict or plan. We understood that frequent rewriting would be required, especially during the development phase. It was obvious that we could not use BAL. We, like many DOS Fortran users, we were well acquainted with its short-comings. For example, we were familiar with its I/O limitations. We had already known how difficult it was to write multi-phase programs. We could not afford the enormous memory consumption by variables placed in Common. Neither could we afford the time to make multiple accesses, required by Fortran to read one disk track.

PL/I seemed to be the answer. If the advertisements were correct, we would save both time and memory. With PL/I we could transfer whole tracks with only one access! With PL/I we could specify length attributes for all variables, internal, or external! Since "Overlay" was already a feature, perhaps multi-phase programming might be easier in PL/I. Our conclusion was that PL/I would give us the advantages offered by both Fortran and BAL, but none of their disadvantages. PL/I combined the ease of coding in Fortran with the efficiency of BAL generated object code.

2. <u>The Learning Phase</u>

We began the study of PL/I borne on the crest of the wave of enthusiasm generated by our boss. And then, the gradual erosion of our enthusiasm began. We had tried to learn the language by reading the manuals. In spite of the confusion created by these computer generated documents, we did manage to learn PL/I. After a quick three-day seminar, conducted by the boss, we began coding.

3. Shakedown Phase

Within a week after our decision to use PL/I had been made, we were actually coding small sample programs. During this period, we were not only testing our knowledge of PL/I but also the compiler. The free form coding of the source statements served to simplify our task. It was reassuring to learn that entire disk records could be written and read with one I/O statement. The versitile compiler would accept multiple statement labels without any confusion. We began to take advantage of many features offered by PL/I. We even tried statement labels and variable names which were a full 31 characters long! It looked as though the period of anguish had come to an end for us - PL/I was going to be nothing less then tremendous!

4. Problem Specification and Coding Phase

The optimistic atmosphere contined to prevail even as work was begun on the main body of the problem. PL/I seemed to be a very simple but powerful language. It was equally facile in developing algorithms or translating formulae. We found that we could generate code almost as fast as it could be written. The ease with which we passed through this initial coding phase only served to confirm our belief that we were, after all, "sitting pretty" - PL/I was on our side!

5. Compilation and Debugging Phase

Then it happened - Everything would have gone so well if we hadn't had to compile all of those powerful statements! They looked so good sitting on the coding sheet, but what the compiler did with them!

What could the compiler mean by telling me that I had a:

"5E071T UNSPECIFIED SYNTHACTICAL ERROR"?

Didn't it know what I had done wrong? Or had I made so many errors it got lost?

And, what could it mean when it told me I had a:

"5E1531T POINTER AS ELEMENT OF DATA LIST"?

What data list? What Pointer?

And, what kind of a sophisticated stochaistic process had it employed to tell me that I had a:

"5E133IT PROBABLY BAD IF NEST"?

Why only probably? Didn't it know for sure? After all, it did look O.K. to me! We reasoned that we had run head long into a random error message generator. After many such experiences, we had to employ the following error message analysis algorithm:

1. Look at the error message and smile!

2. Look at the statement referenced.

3. If the error message seems relevant, fix the statement and try again.

4. If the error message is completely unrelated, look over the entire program and fix anything and everything that looks like it might cause trouble - and try again - and again - and again!

Once we learned to live with the error message, still more funny things happened:

We found out what "Dynamic Storage Allocation" really means -

This phase is really too big to fit in this size memory but why make the programmer worry about such details, as a matter of fact, let's not worry the linkage editor either. Let's make believe this computer has an infinite memory. Later we try to run the program . . . . . .

"5L00I 21", right in the neck!

(which, being interpreted meant "STORAGE OVERFLOW")

Now, this wouldn't be too bad except that there are eleven or twelve phases in the program and you can't be sure which phase blew memory!

Further, we discovered what "BEGIN . . . END" blocks really meant!

"BEGIN" - The consumption of about 180 bytes of memory!

"END" - The programmer's dream of conserving core!

What really seemed like the last straw was the demise of the generalized I/O!

The compiler insisted on giving us a long buffer plus a generalized logical I/O module - a grand total of about 8000 bytes out of our 32K memory! We just couldn't afford that much I/O power on our limited memory budget!

To solve this problem, we decided not to fool around - our disk I/O sections were written in assembler using Physical I/O!

Many surprises came as an inescapable by-product of the implementation limitations on our mod 30. Perhaps, the funniest to watch was a random number generator which behaved like a disk diagnostic. The random number generator could be counted on to cause a "fixed overflow" condition. To stop the endless printout of this error message, a null statement was inserted for the "on fixed overflow" condition. But evidently, the overflow transient routine must still be brought in from disk, even though no printout is asked for. Hence, on every overflow, a disk access is made. The frequency of overflow caused by the random number generator gave the appearance of a disk diagnostic continually accessing the disk to the point of destruction!

Quite aside from surprises due to the limitations of the implementation, we ran into a few compiler errors. (Most of which we have since been informed have been corrected.)

It was with some consternation that we discovered that certain three letter labels would actually hang up the compiler itself! Certain long arithmetic statements followed by a simple assignment statement would make the compiler forget that it should have reserved one of the registers as a base register and not use it for arithmetic. As a result of that compiler error, we got a:

"5L00I 15" - AN ADDRESSING ERROR

Our local C. E. proved to be a great help as he found that a well placed "Begin . . . . End" would remind the compiler to reserve a base register.

But what really caused panic was that moment when the program was finally debugged. The program really runs for the first time. And it runs, and runs, and runs! As a matter of fact, it won't stop! How are we to know that the stop statement was trying to close files for us. But without a file declaration in the root phase the poor compiler got hopelessly lost!

3. Cursory Comparisons

In the midst of our work, we didn't have time for the luxury of comparative language studies. But, we had used some of the more popular languages; and in the midst of our anguish, we could remember what it had been like to use other compilers. Herewith, in outline form is a catalogue of the major differences which we noted between PL/I, Fortran and Extended Algol 60:

FUNCTION	ALGOL 60	FORTRAN IV	PL/I
1) Variables	All must be declared.	May declare - default declarations are full words	May declare - default declarations.
2) Statements	Free Form - ended by , .	Must start after col. 7 and end before col. 73. One statement per card (unless use continuation cards).	Free Form - ended by , .

FUNCTION	ALGOL 60	FORTRAN	PL/I
3) Labels	Begin with alpha - must be declared. Switch label anyplace on card.	Numeric only. Must be in col. 1 - 5.	Begin with alpha-default declarations. Switch label. Anyplace on card.
4) Continuation of statement.	Statement continues until , .	Need continuation	Statement continues until , .
5) Program Form	Procedure, declarations Begin , . End , . All procedures. Type procedures.	Job stream - End , . Main program - call subroutines or function sub - built in functions.	Procedure , . Declarations End , . All procedures. Type procedures.
6) Declarations	Following procedure or begin.	Following subroutine function, or at beginning of job stream.	Following procedure or begin.
7) Common or Transferrable data.	All data declarations are valid in block declared. Desending Hierarchy. Own variables.	Common statement must be present in each phase. Must agree. Two words of memory regardless of size of the variables	Static external, must be present in main or else will be overlayed

FUNCTION	ALGOL 60	FORTRAN	PL/I
8) Comment Statement	Comment . . . , .	C in col. 1 reserves card.	/*. . .*/
9) Debug Aids	Monitor statement. Dump statement.	? ?	Dynadump
10) Subscripts	Entire function used for float. If statements and arithmetic statements allowed in declarations for arrays in inner blocks. Arrays allowed.	Generally, must start with I, J, K, L, M, N. Generally, no array of I (K) allowed a subscript.	If statements allowed and Arithmetic Dummy arguments allowed in declarations for array in inner block. Arrays allowed. Structures allowed.
11) Stream	Very complete stream procedures and instructions.	No way at all.	Have attribute and can handle some data.

4. Suggestion For Improving PL/I

Some improvements are inevitable. For example, the bugs in the compiler itself will eventually disappear. But, it should be obvious that no one group can discover all of the bugs in a compiler. It will require many, many users with a wide range of applications to uncover the lurking latent bugs in PL/I. (No one has written a program to accurately model the unpredictable ways of a programmer!)

There are still bugs being discovered in Fortran and Cobol as well as some implementations of Algol.

Some improvements will be made only after a sort of collective bargaining process is complete. The improvements suggested below probably fall into this category:

1. Let's try to conserve memory (even if our manufacturer is in the business of selling memories!)

This feat could be accomplished in several areas, any of which will entail hugh amounts of work. The most obvious way would be to cut down on the use of the descriptors which are particularly expensive in the case of Static External Variables. This could be done by perhaps another level of abstraction by the compiler, which would result in a grouping of variables, of the same type and length. Another saving of memory could be realized if the length attribute were made to represent the actual number of bits and bytes of memory that were being consumed by this variable. For example, if a variable is given an attribute of Binary (15) that variable will still require a full word of memory. As a matter of fact, the (15) is really superflous, unless one wishes to make sure that the size of the variable will never exceed 15 bits in numerical value. The really extreme in memory consumption occurs when a boolean variable is made Static External. With the present implementation, two words of memory are required to do the work of just one bit! That's an inefficiency factor of 6400%! This packing of items within a word has precedence in many other compilers such as Algol and Jovial. Other memory savings could be realized from a concerted effort to produce one generalized prologue for "begin . . . end" blocks, rather than duplicated code for each. In recent Dos Fortran releases the Logical I/O modules have been growing. This cannot happen in PL/I. They are already too large!

2. Let's save time (even if our rent is based on the reading of a CPU meter!) The most obvious saving in time can be realized in the running of the compiler itself. It takes three to four times the amount of time to compile with PL/I as compared to Fortran. If that time cannot be saved, then the next best thing would be to give meaningful error messages so that the number of times that the compiler would have to be used could be cut to a minimum. It would be a definite aid to know how much memory is being consumed by each DSA. It would also be a definite advantage to know, before execution time that certain formatted I/O statements were in violation of conversion rules.

3. Let's give the programmer some options.

Suppose we have a hypothetical case, - a programmer has declared every variable he intends to use. Let us also suppose that an impossible keypunch error has occured. A "O" has been punched instead of an "o". Now, if the variable is declared or defaulted "Internal", the programmer has no way of knowing that a keypunch error has been made. It would be a comfort to such a programmer to specify by using the UPSI switches, that he wants no default declarations. In which case, the compiler would be in a position to detect an error. If some sort of conversational mode were implemented, then the programmer could also specify whether or not he wished to have the compilation stop on the first error that was detected. Perhaps it would even be a help to have a listing of the "Build In" functions that have been used in a program. Late one night in an unthinking moment, I called an array by the name "EXP". You can guess what happened. I got the exponentiation function of every single subscript that was used in the program. Surely, I made a mistake, but this is advertised as the compiler that won't kill you for what you don't know. Or, will it?

5. <u>Summary Conclusions (or maybe it was all worthwhile!)</u>

Well, if not PL/I, then what language should we have used? We have already given the disadvantages of the other languages at our disposal. It is more

than a rationalization to say that we made a wise choice in PL/I. Those features of the language which worked well, worked very well! A definite saving was realized in the use of structures, address arithmetic, and logical "IF" statements. Quite aside from these considerations, there is certain challenge which accompanies the coding in a new language, not to mention the exilaration to be enjoyed when that language is mastered. Could time be turned back to the beginning of our task, with the knowledge we now have, we would make the same decision. We would use PL/I - Potent or Perplexing, or both! But it is Promising. That is our conclusion. PL/I really is a mnemonic for PROMISING LANGUAGE ONE! If you use it, I'm sure that you will agree.

POWER — SESSION W1A

PRIORITY OUTPUT WRITERS
EXECUTION PROCESSORS,
AND INPUT READERS

(DOS SPOOL LIKE OS HASP)

INTRODUCTION

OBJECTIVES OF A SPOOLING SYSTEM

DESCRIPTION OF POWER

- THEORY
- HARDWARE AND SOFTWARE REQUIREMENTS
- FEATURES

SUGGESTED INSTALLATION PLAN

RUNNING JOBS UNDER POWER

SUMMARY

COMMON
SEPTEMBER 11, 1968

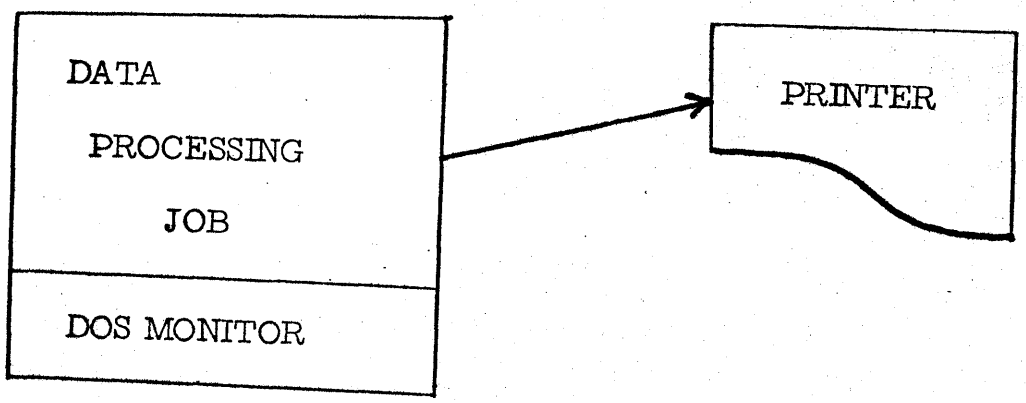
VICTOR L. CUSHMAN
E. I. DU PONT DE NEMOURS & CO., INC.
WILMINGTON, DELAWARE

GENERAL OBJECTIVES OF SPOOL

BETTER USE OF EQUIPMENT

- ECONOMICS
- FLEXIBILITY FOR ORDERLY EXPANSION
- IMPROVED SERVICE (TURNAROUND)

PRINTING WITHOUT SPOOL



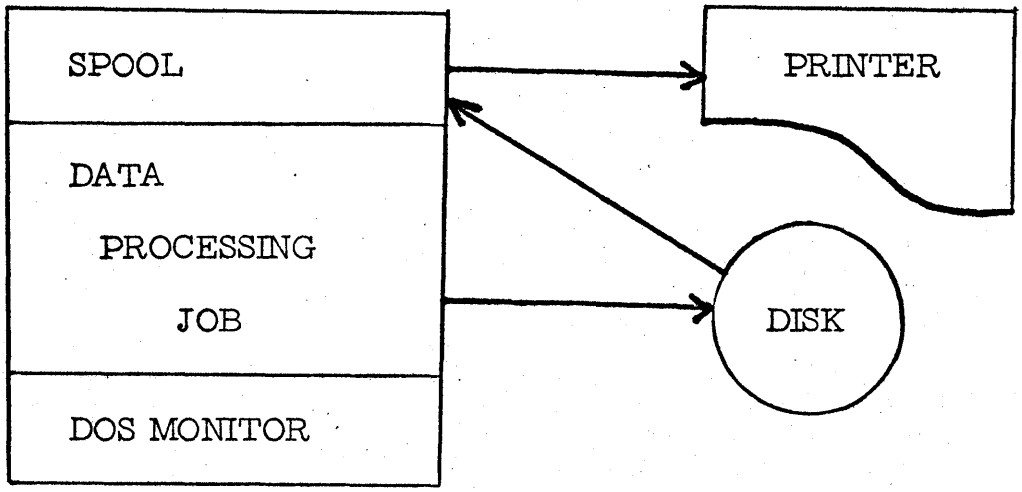
IBM/360

"SPOOL"

(Simultaneous Peripheral Operation On Line)

A multiprogramming technique for increasing thruput by separating printing, punching, and/or card reading from the data processing job.

PRINTING WITH SPOOL

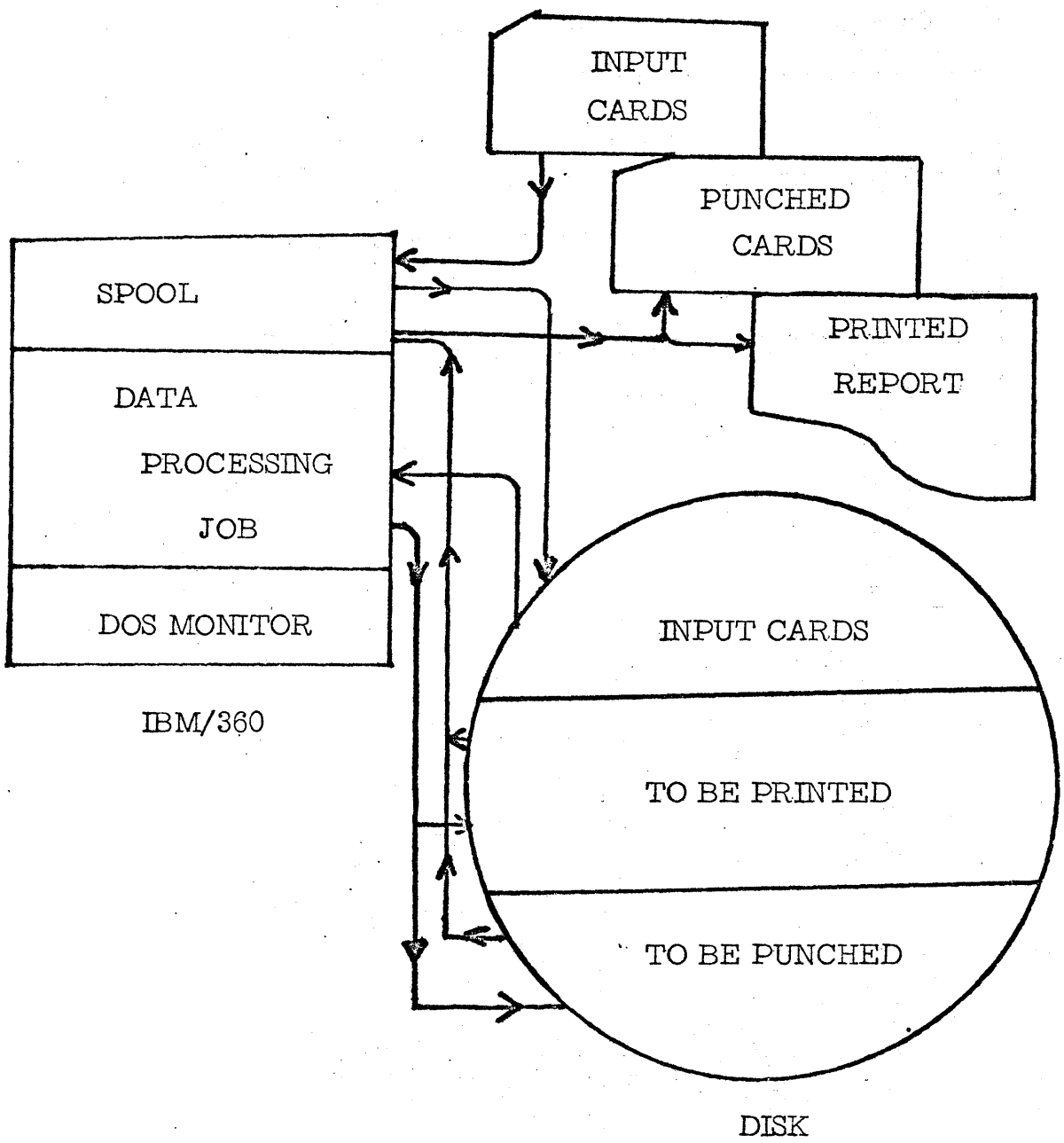


IBM/360

"SPOOL"

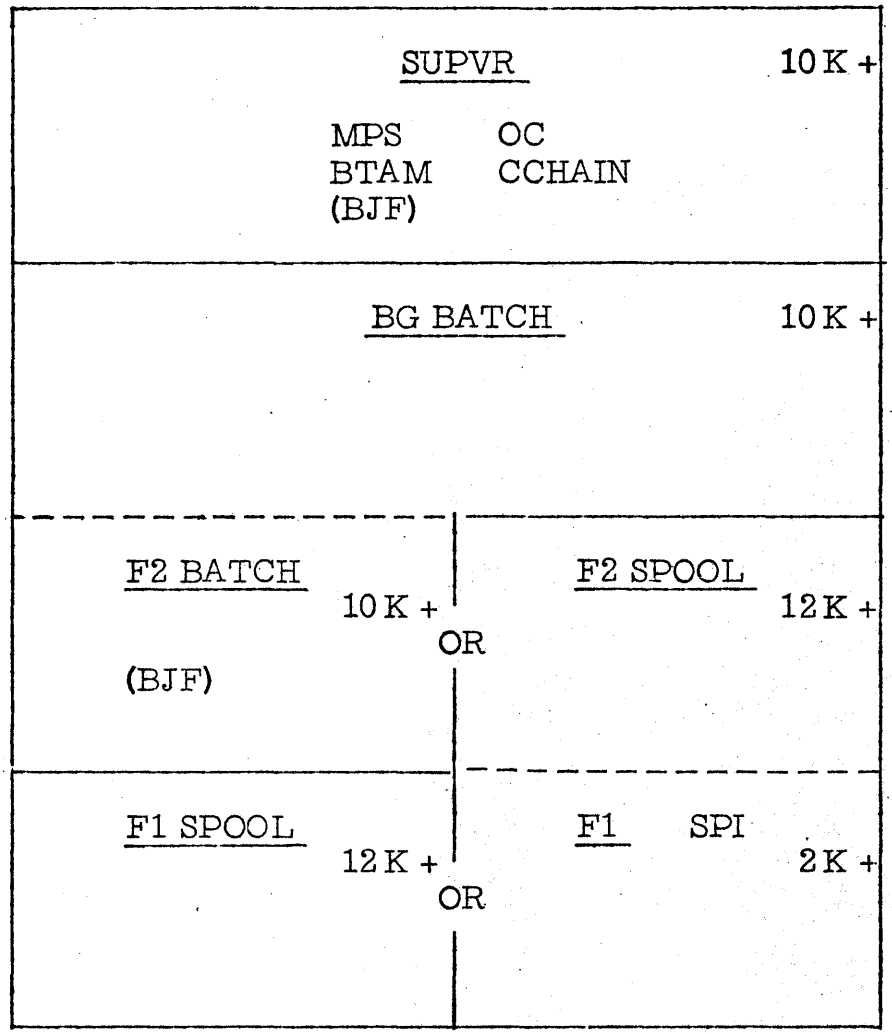
(Simultaneous Peripheral Operation On Line)

A multiprogramming technique for increasing thruput by separating printing, punching, and/or card reading from the data processing job.



DOS SPOOL

CORE MAP



FEATURES OF POWER

- NO IBM OR USER PROGRAM MODIFICATIONS
- FULLY AUTOMATIC -- NO OPERATOR DECISIONS REQUIRED
- FLEXIBLE OPERATIONS
  - SAME JOB CAN RUN UNDER SPOOL OR BATCH WITH NO CONTROL CARD CHANGES
- "WARM START" RECOVERY
  - STORED JOBS ARE SAVED IN EVENT OF FAILURE
- DISK SPACE AUTOMATICALLY REUSED
  - LESS OPERATOR DECISION
- FULL OPERATOR CONTROL
  - COMPLETE OPERATOR COMMUNICATIONS
  - COMMUNICATIONS ALL OPTIONAL
- CARD READ SPOOL
  - AIDS JOB STACKING
- REMOTE JOB ENTRY SUPPORT PROBABLE
- SUPPORTS TWO PARTITIONS FROM ONE OR MORE SETS UNIT RECORD GEAR
- SUPPORTS MULTIPLE READERS/PRINTERS/PUNCHES
- POTENTIAL SUPPORT FOR OTHER DEVICES
- EXTRA COPY CAPABILITY

HARDWARE AND SOFTWARE REQUIREMENTS

HARDWARE

- MINIMUM OF 12 K FOR POWER PROGRAM — 14K for 2 batch partitions.
- ADEQUATE 2311 OR 2314 DISK STORAGE — 50 cylinders required.
- STORAGE PROTECT FEATURE

SOFTWARE

DOS SUPERVISOR — 10 K.

- MPS = YES OR BJF
- TP = BTAM OR QTAM
- OC = YES
- CCHAIN = YES

COS (IF APPLICABLE)

- MPGBLK = 6 (OR GREATER AS NEEDED)
- SYSIO = 001 (OR GREATER)

POWER INSTALLATION PROCEDURES

- REVISE SYSGEN PARAMETERS
- RESERVE CORE AND DISK STORAGE - for spool work files.
- HOLD ORIENTATION SESSIONS FOR
  - MANAGEMENT
  - COMPUTER OPERATIONS
  - USERS
- ISOLATE SPECIAL JOBS WHICH
  - CAN'T BE SPOOLED BECAUSE OF CORE OR DISK
  - SPECIAL LINE COUNTS
- ON-THE-JOB OPERATOR TRAINING
- PREPARE CONTROL CARDS
- GO

POWER CONTROL CARD

```
*   $$JOBNNNNNNNN, H
        JOB NAME     AUTOMATICALLY HOLD JOB

*   $$PRTD, XXXX, YY, MMMMMM
          T
          N

                FORMS   NO        MAXIMUM NO. OF
                 ID    COPIES     LINES OR CARDS
*   $$PUND, XXXX, YY, MMMMMM
          T
          N

        NORMAL DOS

                JOB
                MULTIPLE JOBS
                JOB STEP

*   $$EOJ
```

SAMPLE POWER OPERATOR COMMANDS

S	BGPRT, OOE	START PRINT OUTPUT PROGRAM
S	BGPUN, OOD	START PUNCH OUTPUT PROGRAM
H	BGPRT, ALL	HOLD ALL PRINT JOBS STORED ON DISK
R	BGPUN, ALL	RELEASE ALL PUNCH JOBS STORED ON DISK
P	BGPRT, OOE	STOP PRINTER BUT LEAVE JOB ON DISK
C	BGPRT, OOE	STOP PRINTER AND TAKE JOB OFF DISK
D	BGPUN, ALL	TYPE EVERY PUNCH JOB STORED ON DISK
G	BGPRT, OOE	RESTART PRINTER AFTER FORMS HAVE BEEN CHANGED
E		TERMINATE SPOOL

W3 A,B

Alexander F. Saunders, Senior Analyst
Travelers Research Center, Inc.
250 Constitution Plaza
Hartford, Connecticut 06103

COMMON, Chicago, Illinois
360 Project, Session B7

April 22, 1968

"DOS Physical IOCS and FORTRAN"

Introduction

The reason for presenting the following material is to illustrate that user-written assembler language subroutines in support of DOS FORTRAN are not particularly formidable, and provide a means for performing unique functions faster and with greater flexibility and control. The use of physical IOCS routines for reading and writing binary tape records is the primary area of concern, although tape control functions, etc., will also be discussed. Whenever applicable, examples and performance comparisons with FORTRAN capabilities is provided. The routines to be described have been operational for over a year at the Travelers Research Center Computer Laboratory. The IBM 360/40 and 2402 mod II tape drives at that facility were used to obtain data for this report. The mode of operation was DOS, release 14.

1. The initial impetus for attempting a user-written tape I/O capability was provided by an historical requirement. Prior to obtaining its own computer, TRC had used the IBM 7090 series facilities at The United Aircraft Research Laboratories in East Hartford, Connecticut, and the I/O packages developed by their personnel. To maintain a minimum amount of re-programming effort, similar subroutines were desirable under our own system capability. It was also necessary to generate routines which (1) executed faster than their FORTRAN counterparts, (2) provided greater user control, (3) were easy to use, and (4) had a low core requirement.

2. Logical IOCS did not lend itself adequately to generalization, and, therefore, physical IOCS became the chosen method. As a brief description, PIOCS allowed performance of non-data operations and control of the transfer of data to and from external devices via four supervisor resident routines: start I/O, I/O interrupt, channel scheduler, and device error.

3. Eight subroutines were written to accomplish the specified control and read-write capability. Figure one contains a description of each. At this time the routines are general only to TRC users, and would probably need some revision to perform satisfactorily at another installation.

4. Figure two illustrates comparative execution times with FORTRAN, example one a subroutine used to measure the execution times, and example two demonstrates a typical rewind subroutine. Figure four, though not based on an I/O oriented procedure, was included to encourage assembler language programming in computational areas. It is especially attractive for production type programs when heavy FORTRAN indexing is involved.

5. After each routine having a response indicator as one of its arguments, a computed go to is usually employed to perform branching to the necessary statement number. Branch time was included in collecting data for figures two and three.

6. There are three factors involved in the performance differences indicated in figure two. FORTRAN generates 63 word data records preceded, in release 14, by two control words. Release 13 on down required one leading control word. Computation and testing of the control words and the movement of data to and from buffer areas also add considerably to the execution time. By using the PIOCS subroutines these time consuming functions were avoided. The advantage gained by writing one long record rather than a series of shorter ones may be demonstrated by referring to figure three. The time required to write a single 5000 word record is 0.35 seconds. If, instead,

one-hundred 50 word records were written, the time required jumps to 2.00 seconds. To write 5000 words with FORTRAN would take approximately 4.83 seconds.

7. At this time the core required by all eight routines is 850_{16}. They are presently being re-written to provide facility for handling records up to 16383 words in length, and to sense for not-ready and file protect status. Record size limitations now are within the range of 4 to 8191 words. Control routines RWD, WEF, and UNL are being changed to accept a variable number of arguments, thereby allowing the programmer to request multiple operations in one call statement. Incidently, the number of re-tries in event of I/O error has been programmed at five times reading, and fifteen writing. This has proven quite satisfactory.

8. Discussion to this point has been almost exclusively about magnetic tape I/O. Other external I/O devices (typewriter, printer, and card reader-punch) have also been programmed to perform unique functions. For instance, re-read type subroutines, typewriter communication, and job accounting procedures, to mention a few, have been implemented using physical IOCS.

9. In addition to FORTRAN, the routines described have linked and executed properly with PL/I and assembler language programs. The biggest problem in commencing PIOCS programming is in gathering the necessary material. Hopefully, example two, the references provided, your SE, and considerable reading and imagination will bridge the gap for interested programmers. The investment is well worthwhile.

References:

Source	Content
IBM System/360 Disk Operating System Supervisor and Input/Output Macros C24-5037-2	CCB (command control block) EXCP (execute channel program) WAIT
IBM System/360 Disk and Tape Operating Systems Assembler Language C24-3414-4	CCW (channel command word)
IBM System/360 Principles of Operation A22-6821-3	General
IBM System/360 Disk and Tape Operating System FORTRAN IV Specifications C24-5014-0	Subroutine Linkage
IBM System/360 Reference Data (Green Card) X20-1703-3	Channel Command Codes Channel Address Word Channel Command Word Channel Status Word
IBM System/360 Component Design 2400-Series Magnetic Tape Units and 2816 Switching Unit A22-6866-3	Sense Information
Programming the IBM System/360 Appendix F Staff of Computer Usage Co. John Wiley and Sons, Inc.	I/O Device Responses General Information

PIOCS Subroutine	FORTRAN Counterpart	Use
CALL RBT (A, NWDS, NT, NE)	READ (NT) (A(I), I=1, NWDS)	Read a physical record a known length or the first N words of a longer record.
CALL RBTX (A, NWDS, NT, NE)	-none-	Read a physical record of unknown length.
CALL WBT (A, NWDS, NT, NE)	WRITE (NT) (A(I), I=1, NWDS)	Write a physical record.
CALL RWD (NT)	REWIND NT	Tape rewind.
CALL UNL (NT)	-none-	Tape rewind and unload.
CALL WEF (NT)	END FILE NT	Write a tape mark.
CALL RSKP (NCT, NT, NE)	READ (NT) BACKSPACE NT	Skip forward or backward a given number of records. If an end of file is sensed, skipping is terminated regardless of the requested count.
CALL FSKPX (NCT, NT, NE)	-none-	Skip forward or backward a given number of files.

Symbol Definition

A Name of array or variable where data is to be read into or from in consecutive fashion
NWDS Number of words to be read or written. This is a returned value for RBTX.
NCT Number or records or files to skip. A negative value results in backwards motion.
NT FORTRAN unit assignment. Values of 10 through 14 are acceptable.
NE Response Indicator.

Table of Responses, NE

Routine \ Response	1	2	3	4	5
RBT	OK	end-of-file detected	I/O error	*	Wrong length record (longer or shorter
RBTX	"	"	"	*	
WBT	"	end-of-tape detected	"	*	
RSKP	"	end-of-file detected	*		
FSKPX	"	*			

* Word count (NWDS) or unit selected (NT) in error

FIGURE I

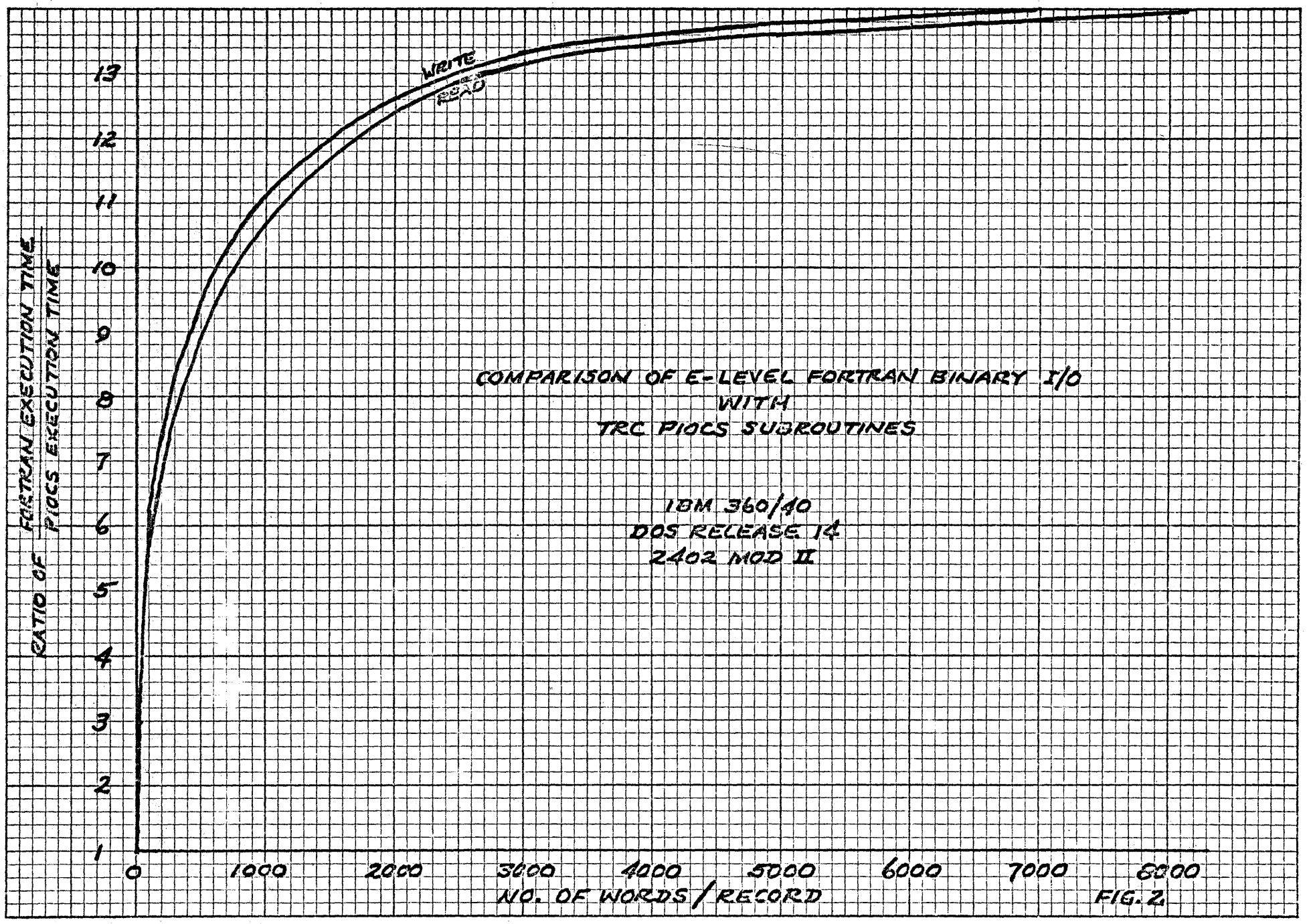


FIG. 2

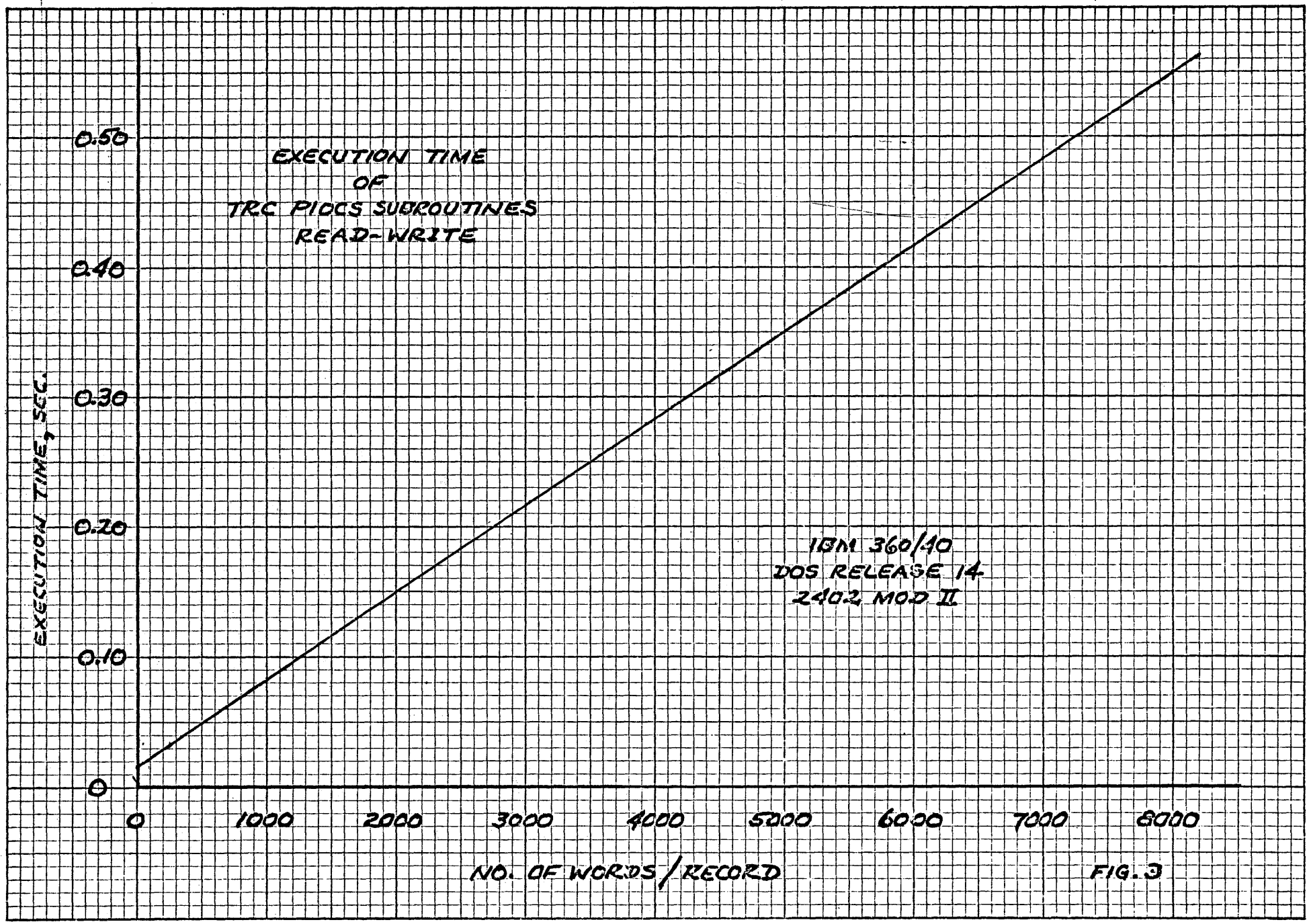

EXECUTION TIME
OF
TRC PIOCS SUBROUTINES
READ-WRITE
IBM 360/40
DOS RELEASE 14
2402 MOD II
EXECUTION TIME, SEC.
0.50
0.40
0.30
0.20
0.10
0
0
1000
2000
3000
4000
5000
6000
7000
8000
NO. OF WORDS/RECORD

FIG. 3

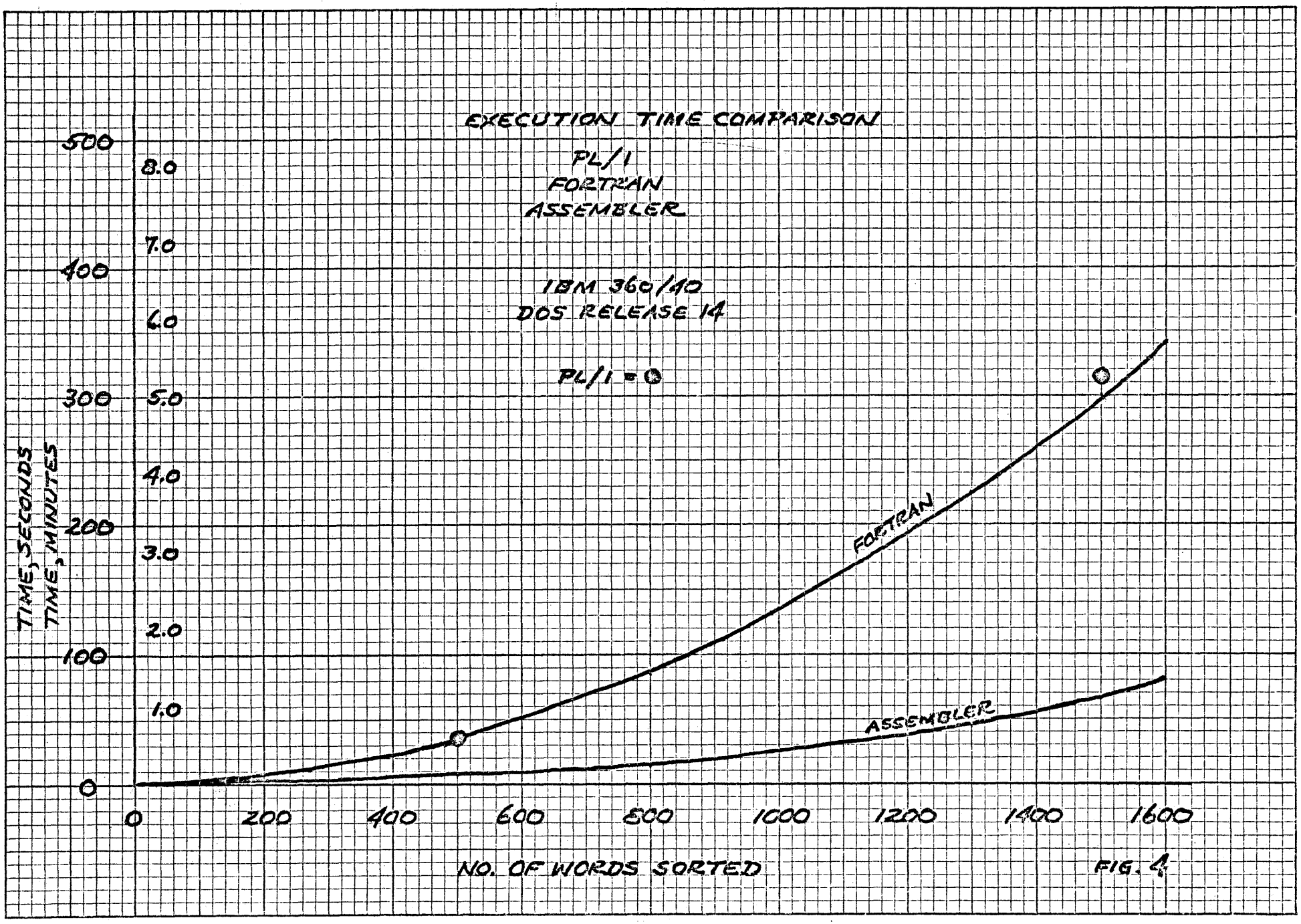


FIG. 4

```
 LOC   OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

000000                                 1 TRCTSEC  START 0                CALL TIME(IT300)
                                       2          ENTRY TIME
000000                                 3          USING *,15
                                       4 TIME     SAVE  (14,12)
                                       5+* 360N-CL-453  SAVE      CHANGE LEVEL  2-0
000000 90EC D00C             0000C     6+TIME     STM   14,12,12+4*(14+2-(14+2)/16*16)(13)
000004 5851 0000             00000     7          L     5,0(1)
                                       8          GETIME TU              GET CLOCK TIME IN 1/300 SEC
                                       9+* 360N-CL-453  GETIME    CHANGE LEVEL  2-0
000008 9801 0050             00050    10+         LM    0,1,80 GET TIMER VALUE IN SEC/76800
00000C 8800 0008             00008    11+         SRL   0,8 TIMER IN SEC/300
000010 1F10                           12+         SLR   1,0 TIME OF DAY IN SEC/300
000012 5015 0000             00000    13          ST    1,0(5)
                                      14          RETURN (14,12)
                                      15+* 360N-CL-453  RETURN    CHANGE LEVEL  2-0
000016 98EC D00C             0000C    16+         LM    14,12,12+4*(14+2-(14+2)/16*16)(13)
00001A 07FE                           17+         BR    14
                                      18          END
```

EXAMPLE 1

```
 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

000000                                1 TRCRWDE  START 0                CALL RWD(NT)
                                      2          ENTRY RWD
000000                                3          USING *,15
                                      4 RWD      SAVE  (14,12)
                                      5+* 360N-CL-453  SAVE       CHANGE LEVEL  2-0
000000 90EC D00C            0000C     6+RWD      STM   14,12,12+4*(14+2-(14+2)/16*16)(13)
000004 58C1 0000            00000     7          L     12,0(1)          FORTRAN UNIT ADDRESS
000008 48BC 0002            00002     8          LH    11,2(12)         FORTRAN UNIT ASSIGNMENT
00000C 4BB0 F03C            0003C     9          SH    11,=H'3'         CONVERT TO SYS NUMBER
000010 42B0 F027            00027    10          STC   11,CONTROL+7     STORE IN CCB
                                     11          EXCP  CONTROL          EXECUTE REWIND
                                     12+* 360N-CL-453  EXCP       CHANGE LEVEL  2-0
000014 5810 F038            00038    13+         L     1,=A(CONTROL)
000018 0A00                          14+         SVC   0
                                     15          RETURN (14,12)
                                     16+* 360N-CL-453  RETURN     CHANGE LEVEL  2-0
00001A 98EC D00C            0000C    17+         LM    14,12,12+4*(14+2-(14+2)/16*16)(13)
00001E 07FE                          18+         BR    14
                                     19 CONTROL CCB   SYS000,TAPE
                                     20+* 360N-CL-453  CCB        CHANGE LEVEL  2-4
000020 0000                          21+CONTROL DC    XL2'0' RESIDUAL COUNT
000022 0000                          22+         DC   XL2'0' COMMUNICATIONS BYTES
000024 0000                          23+         DC   XL2'0' CSW STATUS BYTES
000026 01                            24+         DC   AL1(1) LOGICAL UNIT CLASS
000027 00                            25+         DC   AL1(0) LOGICAL UNIT
000028 00                            26+         DC   XL1'0'
000029 000030                        27+         DC   AL3(TAPE) CCW ADDRESS
00002C 00                            28+         DC   B'00000000' STATUS BYTE                    2-4
00002D 000000                        29+         DC   AL3(0) CSW CCW ADDRESS
000030 0700003000000001              30 TAPE     CCW  7,*,X'00',1
                                     31          END
000038 00000020                      32               =A(CONTROL)
00003C 0003                          33               =H'3'
```

EXAMPLE 2

COMMON S/360 Model 44 Committee

"Design, Features, and Performance of the Engineering Spooling Program"

T. R. Price

IBM Corporation
3130 N. Meridian Street
Indianapolis, Indiana 46208
Area 317-924-3371

September 11, 1968
Wednesday 11:00 A. M.

Text 16 Pages
Graphics 13 Pages

ABSTRACT

The Model 44 Engineering Spooling Program modifies the S/360 Model 44 Programming System to permit input/output requests for the card reader, printer, and punch to be intercepted and satisfied by high speed core-to-core transfers.

Concurrent with execution of a 44PS job stream, ESP attempts to read cards in advance of their actual use, storing them in internal core buffers and on a disk. All queuing is first in/first out, by device. Thus, at a given moment, Job C may be computing, output from Job A may be printing, and output from Job B may be punching, while Jobs D, E, F, etc., are being read and stored on disk, for later execution.

The paper being presented will discuss Design, Features, and Performance of the Engineering Spooling Program.

DESIGN, FEATURES, AND PERFORMANCE OF THE ENGINEERING SPOOLING PROGRAM

Good morning ladies and gentlemen! During the next few minutes we will be discussing some of the important elements of the S/360 Model 44 Engineering Spooling Program, a new addition to the IBM Type 3 program library.

The Model 44 Engineering Spooling Program (ESP) modifies the S/360 Model 44 Programming System to permit input/output requests for the card reader, printer, and punch to be intercepted and satisfied by high speed core-to-core transfers.

Concurrent with execution of a 44PS job stream, ESP attempts to read cards in advance of their actual use, storing them in internal core buffers and on a disk. All queuing is first in / first out, by device. Thus, at a given moment, Job C may be computing, output from Job A may be printing, and output from Job B may be punching, while Jobs D, E, F, etc., are being read ahead and stored on disk, for later execution.

The current status of the system, consisting of the number of jobs read ahead, the number of unit record images queued on disk, and the total of user estimated run times, is printed on the typewriter before the system displays each job card.

This program should be of interest to Model 44 users because of its dramatic effect on job thruput. Some actual job stream timing comparisons will be presented later in this paper.

However, let me first pose the following problem. How can the thruput derived from the Model 44 be

increased without multiprogramming user programs?

One method that has been employed is to speed up the slowest component of the system. If analysis shows that the CPU is limited by the speed of its I/O devices, such as the card reader or printer, replacement of those particular components by higher speed devices, such as tapes, should generally increase thruput. A second application of the same technique might result in replacing the tapes by higher speed tapes, and so on, through successively higher performance tape equipment.

The factor being sought above is some ideal balance between CPU and I/O device speeds in which the CPU does not wait for an I/O device and the I/O device is kept running at its maximum rate. When a change in the speed of some component of the system no longer produces an advantageous or corresponding improvement in thruput, the system may be termed balanced.

Core buffering is a second technique that can be applied to increase thruput. Buffering permits rapid response to the high instantaneous I/O rates that can be asynchronously demanded by a program in the CPU. This same buffer can then be refilled at the rated speed of the I/O device which it is servicing. A core buffer allows the apparent disparity between CPU speed and I/O device rates to be more closely matched. This matching can approach the ideal balance described above.

ESP attempts to increase the thruput of the Model 44 Programming System by applying extended disk and core buffering techniques to unit record devices. This technique improves 44PS thruput, particularly on jobs consisting of significant volumes of unit record input/output operations.

Let us now devote some time to a study of the design objectives of the Engineering Spooling Program. Plans to implement ESP were developed about a year ago when a 256K S/360 Model 44 was installed at a location in the Midwest. A block of memory in this system was reserved for future ESP use by including a 64K dummy routine from the relocatable library in all user Fortran programs. The design level for ESP was thereby fixed at a 64K maximum.

This first slide (G1) illustrates the memory allocation of the 256K bytes from the initial installation in August of 1967 until ESP coding and testing began in November. Actually, the idea that 64K was unused during this period is not entirely correct, for many programmers were devising and optimizing overlay techniques which utilized the excess core during early testing stages.

The ultimate requirements of ESP turned out to be in the range of 25K to 42K, depending on user-specified assembly parameters such as buffer lengths, disk queue space, and other tables. This is shown in the next slide (G2).

A second design objective for ESP specified that the job control language and other functional capabilities of the S/360 Model 44 Programming System should not be altered. Meeting this objective has allowed the operator retraining problem to be minimized.

Associated with this second objective was the idea that ESP should not permanently modify the 44 Programming System. This was to permit the user's system to be operated in either a normal PS mode or a spooling mode. Implementation of this feature has proved useful in diagnosing recalcitrant user or IBM programming problems.

Finally, the dependency of ESP on a given release of 44PS was to be minimized, thus providing a high degree of release independence.

The above design objectives of core, PS compatability, and release independence have all be realized. In addition, a 25 per cent performance increase objective has been surpassed. In fact, a sustained 1.8 to 1.0 performance improvement has been maintained in the original installation since initial use of ESP.

The ESP package is invoked by executing a phase, called ESP1, stored on the 44PS phase library. This phase inserts several modifications into the normal 44PS supervisor and loads a second phase, called ESP2, into high core, where it remains resident. The supervisor modifications operate with the ESP2 resident phase to provide the spooling capability of the ESP system.

After the above steps are completed, all further card reader, printer, and punch operations are intercepted and are satisfied by core-to-core transfers performed by the ESP system. The general flow of control is illustrated in the slide. (G3).

The general flow of unit record data under the Model 44 Programming System is modified by the addition of the Engineering Spooling Program package. The flow of card data in ESP is illustrated in the next slide. (G4). Input card images are read ahead of their actual use and are placed in the prime input buffer (1). When the prime input buffer becomes full, its function is switched with an associated empty alternate buffer (2). The full buffer is written to disk (3) and the empty buffer is used to hold new card images. Card reading is sustained through use of I/O interrupts (4).

Card images are supplied to the requesting program upon receipt of an SVC READ interrupt (5). Card images are supplied to the requesting program from the prime output buffer (6). If the last card image was previously transferred from the prime output device buffer to the problem program, then a switch of the prime and alternate buffers takes place (7). The empty buffer is then filled by reading a buffer from disk (8), or exchanging the empty output buffer for either a filled alternate input buffer (9) or a partially filled prime input device buffer (10). The exchange of input and output buffers, (9) or (10) above, permits output requests to dynamically catch up to the very latest input record.

The status of input and output device buffers is maintained by entries in a Current Position Table. Disk cylinder chain lengths are maintained by a Cylinder Chain Table. The state of I/O operations is maintained in an Interrupt Control Table.

The card data flow described above is functionally equivalent to the normal flow using 44PS. The same is true of the data flow for the printer and card punch. Consequently, programs which operate under 44PS at the READ/WRITE/CHECK level should operate in conjunction with the ESP package without change.

The spooling function of ESP is kept active by intercepting SVC requests for READs and WRITEs, and by executing ESP instructions just before 44PS returns from an I/O interrupt.

This next slide (G5) summarizes the types of interrupts which are intercepted and cause ESP to become temporarily active. READ, WRITE, and CHECK are processed by ESP when they reference the reader, printer, or punch. Other SVCs are inspected, e.g., CANCEL or EOJS, before routing them to the 44PS supervisor for processing. This permits all spooled I/O operations to be terminated normally before the pending I/O of the problem program is purged from the channel queue of the 44PS supervisor.

Issuing SVCs before returning from an I/O interrupt causes a general reentrant problem to occur. The 44PS supervisor will handle six nested SVCs--the worst possible case during normal operation of 44PS. However, superimposing spooling operations on the normal 44PS supervisor will cause the limit of six nested SVCs to be exceeded. This general problem has been solved in ESP by saving the entire SVC pushdown list contained in the 44PS supervisor before ESP I/O requests, and then restoring the list afterwards. This technique has equipped the 44PS supervisor with the additional reentrant capability required by ESP.

A general ESP logic diagram is shown in the next slide (G6). Entry can occur by an SVC interrupt, an I/O interrupt, or by a CALL from the SVC to the I/O interrupt routine. Addresses of all routines and some strategic switches are located in a commonly accessible vector called the Program Communications Region.

The actual coding of the ESP routines and necessary tables accounts for about 12K bytes.

Buffer lengths are variable, and can bring the total memory requirement up to about 42K.

Thus far this presentation has considered design objectives of ESP and some of the program logic. Let us now inspect some of the advantages and features of the Engineering Spooling Program.

The greatest advantage which results from the use of ESP is improved job thruput. Improved thruput can mean improved turnaround time for individual system users.

Increased error recovery capability results when ESP is used, since reader checks, validity checks, card jams, etc. can be corrected while a previously read job is in execution.

READ/WRITE level programs, such as Fortran compiles, link edits, and Fortran execution steps, can be run without change.

Minimum operator retraining is necessary to change from 44PS to 44PS with ESP.

Finally, reduced operations overhead time can result from the use of ESP, since printer paper changes or punched card refills can usually be performed without stopping the CPU.

These advantages are listed on slide G7.

Several unique program features distinguish ESP from other spooling programs available for the

S/360 Model 44. Some of these program features will now be discussed. (Refer to slide G8.)

Input Spooling. Input images are read as far ahead of actual use as possible, up to a maximum queue size specified by the user. These input images remain queued on disk and in core buffers awaiting program requests for card input.

Output Spooling. Printer and punch images generated by programs are transferred from the user output area to the appropriate ESP device buffers. When the capacity of a device buffer is reached, it is written to disk.

Printer and punch images are retrieved from disk and core queues for display on the appropriate I/O device.

Each output device continues to operate independently as long as its output buffer is not empty.

All printer ASA control codes are accepted. Other codes default to single space.

Printer page ejection is generated by ESP if a user specified maximum lines per page value is reached.

All punch ASA stacker select codes are accepted. Other codes default to pocket 1 selection.

Effective Buffering. Twelve I/O buffers can

permit independent double buffering of data transfers to and from disk for reader, printer, and punch.

An output buffer can dynamically approach, catch up to, and later fall behind the most recent record in the corresponding device input buffer. This reduces the amount of required disk queuing space.

Buffer sizes for each device are independent and are specified by the user.

Efficient Disk Queuing. Up to 200 consecutive cylinders of 2311 or 2315 disk storage space can be devoted to disk queue space.

Disk queue space requirements are reduced, due to the ability of each output queue to approach and catch up to the most recent image of the input queue.

An independent maximum cylinder chain length can be specified by the user for reader, printer, and punch disk queues.

Each new cylinder added to a device cylinder chain is picked so as to reduce arm movement.

A dynamic list of available and chained cylinders is maintained in core to permit rapid retrieval of buffer information from disk.

Operator Information. Before each job starts, the following information is displayed on the operator typewriter console.

Number of jobs queued on disk awaiting processing.

Total of user estimated time for all jobs awaiting processing.

Number of reader input images queued on disk.

Number of printer images queued on disk.

Number of punch images queued on disk.

Two major topics remain in this presentation: System Configuration Requirements and Performance Data. The required hardware configuration and programming systems can be easily presented by the next slide. (G9)

The hardware configuration necessary to run ESP is the same as the configuration required to run the S/360 Model 44 Programming System, with the following additions.

1. A 128K or 256K CPU, to provide space for 44PS and for ESP. The ESP resident phase requires from 25K to 42K, depending on user-specified buffer sizes.

2. From 10 to 200 cylinders of 2315 or 2311 DASD space, to be devoted to disk queuing of unit record images.

3. A reader, printer, and punch with unique device addresses. The unique device addresses

permit concurrent reading and punching operations to be performed.

4. Sufficient space in the absolute phase library of 44PS to contain the two ESP phases. A maximum of 75 blocks is required.

We will now examine several measurements of the performance of the Engineering Spooling Program. Three distinct types of performance data will be presented: job stream timing comparisons, CPU interference, and maximum simulated unit record I/O rates.

In order to obtain job time comparisons, two separate job streams were run under 44PS and also under 44PS with ESP. The first job stream consisted entirely of various Fortran compiles, link edits, executes, and disk executes. The second job stream consisted of a series of assemblies.

The timings for the Fortran and Assembler job streams are presented in slides G10 and G11, respectively. To summarize, the Fortran job stream showed an average improvement in thruput of 1.9 to 1.0, under ESP, while the assemblies showed an average gain of 2.4 to 1.0, relative to 44PS without ESP. Assembler decks that could be completely read ahead before execution showed an average gain of 3.2 to 1.

The above figures illustrate the performance improvements obtained at one particular installation.

565

Because of widely differing job profiles and job types, the performance improvement obtained at other installations can be expected to vary from the ratios presented above.

In order to determine the extent of CPU interference due to concurrent ESP I/O operations, a Fortran test job was written. This compute bound job was executed twice, once using 44PS, and a second time while ESP was concurrently processing the reader, printer, and punch at a maximum rate.

The Fortran program executes a simple instruction sequence a large number of times. The gross execution time was determined by executing a Clock Read subroutine before and after a long program loop. Next, the net execution time was computed by subtracting the time to execute the Clock Read subroutine from the gross time.

The results of the test were as follows. CPU performance may be degraded by up to a maximum of 14.8 per cent when reader, printer, and punch are all operated at maximum rate.

However, it is at first paradoxical that the long execution step of a given job may be degraded by the concurrent I/O activity of other jobs, yet the total of compile, link edit, and execute times of the current job may show a decrease. This is possible because the increased I/O speeds for the compile, link edit, and execute steps may more than compensate for the degraded execution speeds.

A second Fortran test job was constructed in

order to determine the I/O rates which ESP could maintain under ideal conditions for reader, printer, and punch. The factor being measured is the instantaneous I/O rate simulated by ESP.

An assembler subroutine, Clock Read, was again used to obtain the execution timings. This routine returns the interval timer value to the calling program.

The next slide (G12) presents the data collected from a Fortran program that performed a series of read, print, and punch operations. The Clock Read routine was used to time one thousand consecutive simulated I/O operations on each device. These values, then, represent the maximum I/O rate, for each device, that can be sustained by ESP in a Fortran job.

The time and rate for intermixed I/O operations on the reader, printer, and punch shows the degree to which the maximum I/O rate is reduced by disk queue arm contention. Disk arm contention can be reduced by increasing the size of one or more of the device buffers.

It is interesting to compare the ESP maximum I/O rates for reader, printer, and punch to the corresponding maximum rates for 90 KB tape drives. For example, the maximum rate for unblocked card images read from a 2401 Model 3 tape drive is about 6.19 ms., or 161 records per second. For comparison, the actual sustained effective transfer rate of blocked card images using ESP is approximately 420 images per second, including all CPU Start I/O and interrupt processing time.

Several extensions to the existing ESP package have been made or are being considered by current users of the ESP system. Some of these modifications will be briefly mentioned. They might serve as topics for a user presentation at some future COMMON meeting. (Slide G13.)

Magnetic Tape to Paper Tape Spooling. An installation has successfully added such a spooling facility to the structure of ESP. The spooling operation was accomplished in an orderly way by extending existing control tables in ESP, and by adding a 1012 paper tape device routine to ESP.

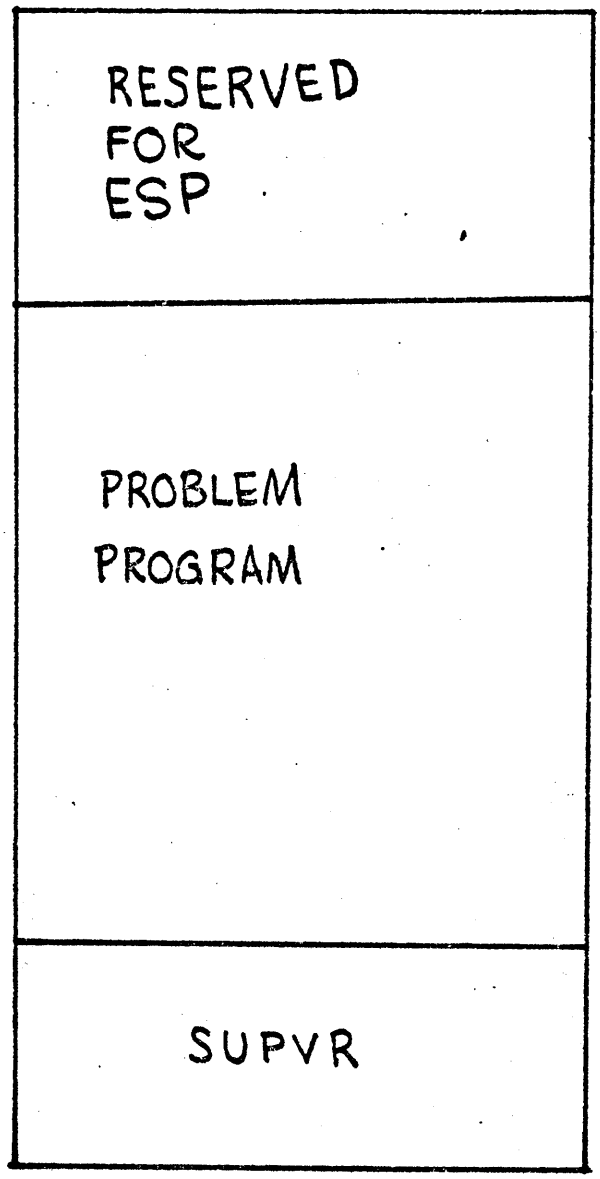
Plotter Spooling. A 1627 plotter spooling capability is being added by another installation by substituting the plotter for the punch device. By specifying a control character rather than ASA mode for the space control character, and appending the plotter write code to each plot record, plotter spooling can be inplemented without major changes to ESP.

Remote Job Entry. Adding appropriate 2701 device routines to the system to permit remote job entry is being considered by at least one installation. The cylinder chaining technique can be modified to chain to cylinders containing previously entered terminal jobs. These jobs can then be executed as part of the normal 44PS job stream. Information in the user's job card can be used to return printed output to the appropriate terminal.

The Engineering Spooling Program can significantly extend the range of performance and operational flexibility of the S/360 Model 44. I urge each of you and your respective IBM Systems Engineers to consider the relative advantages and hardware requirements of ESP in the light of your own installation needs and objectives.

This concludes the formal presentation portion of this session. Are there any questions?

MEMORY MAP AT INSTALLATION TIME



DESIGN OBJECTIVES

MEMORY 25K TO 42K

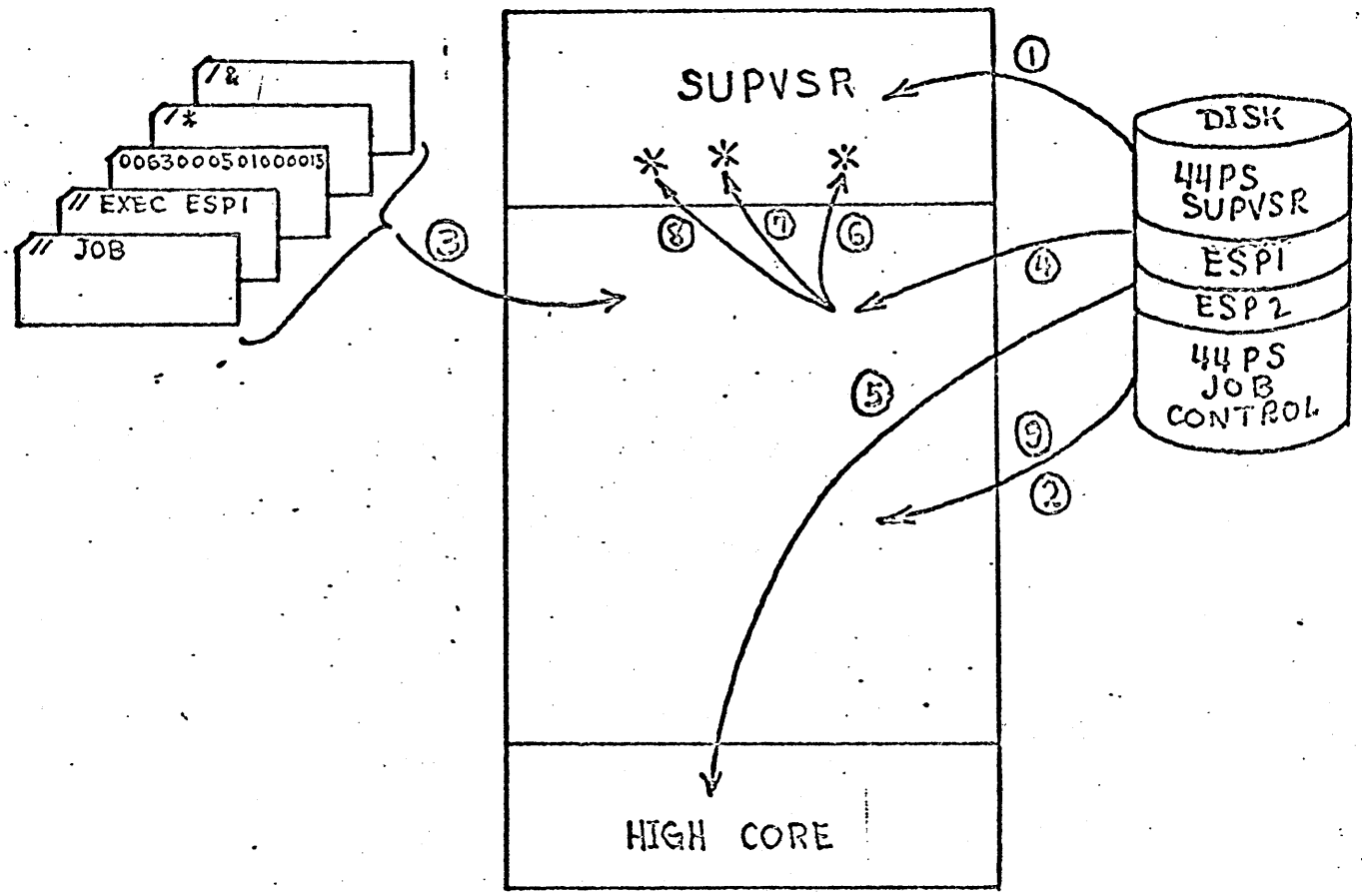
44 PS COMPATABLE

TEMPORARY 44 PS MODIFICATIONS

RELEASE INDEPENDENCE

IMPROVED PERFORMANCE

Figure 1. GENERAL FLOW OF CONTROL IN EXECUTING ESP



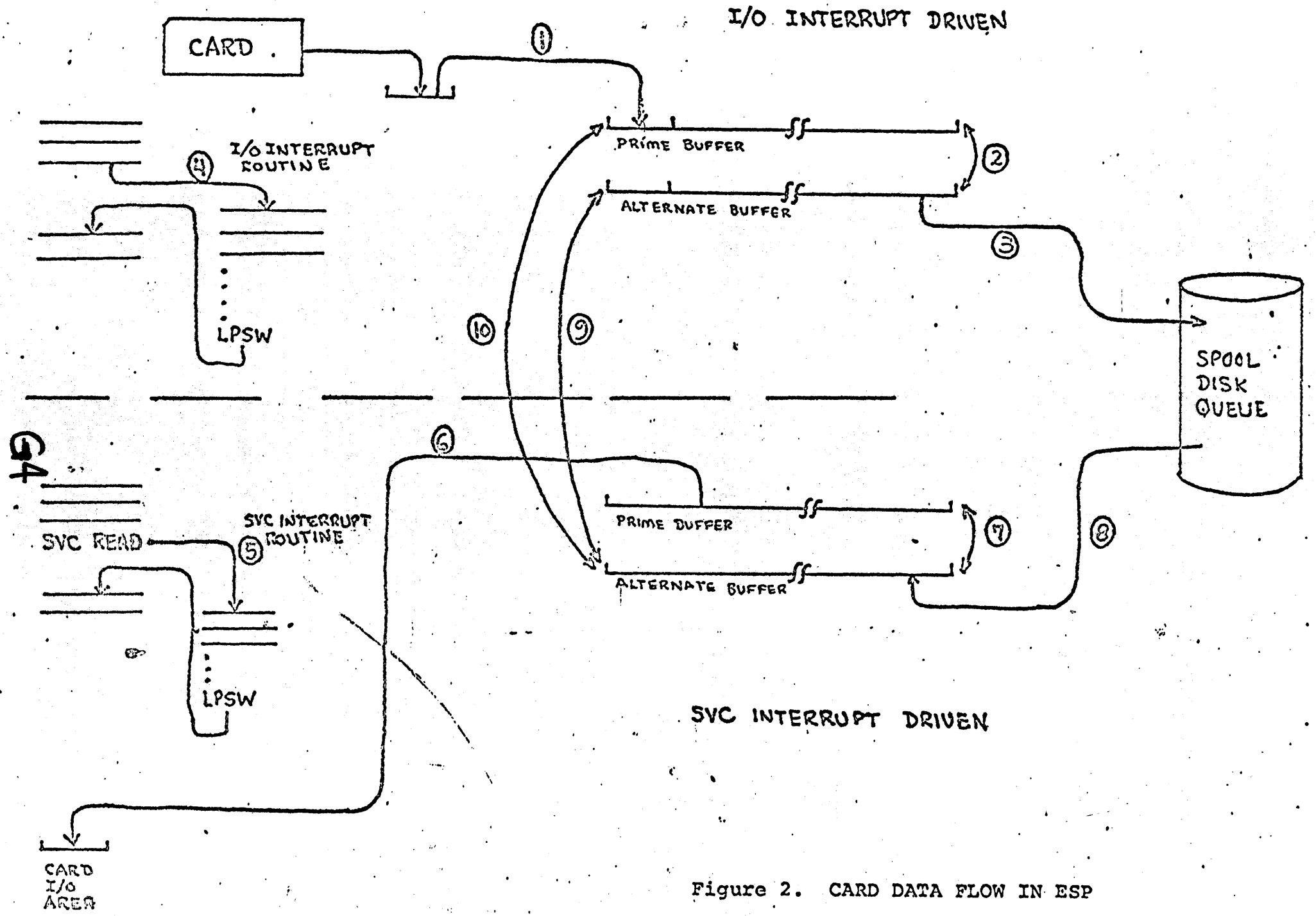


Figure 2. CARD DATA FLOW IN ESP

INTERRUPTS SUSTAINING ESP OPERATION

SVC DRIVEN	I/O INTERRUPT DRIVEN
READER OUT	READER IN
PRINTER IN	PRINTER OUT
PUNCH IN	PUNCH OUT

Flowcharting Worksheet

Programmer: T. R. PRICE Program No.: Date: JUNE 1968 Page: 1 OF 1

Chart ID: Chart Name: ESP GENERAL BLOCK DIAGRAM Program Name:

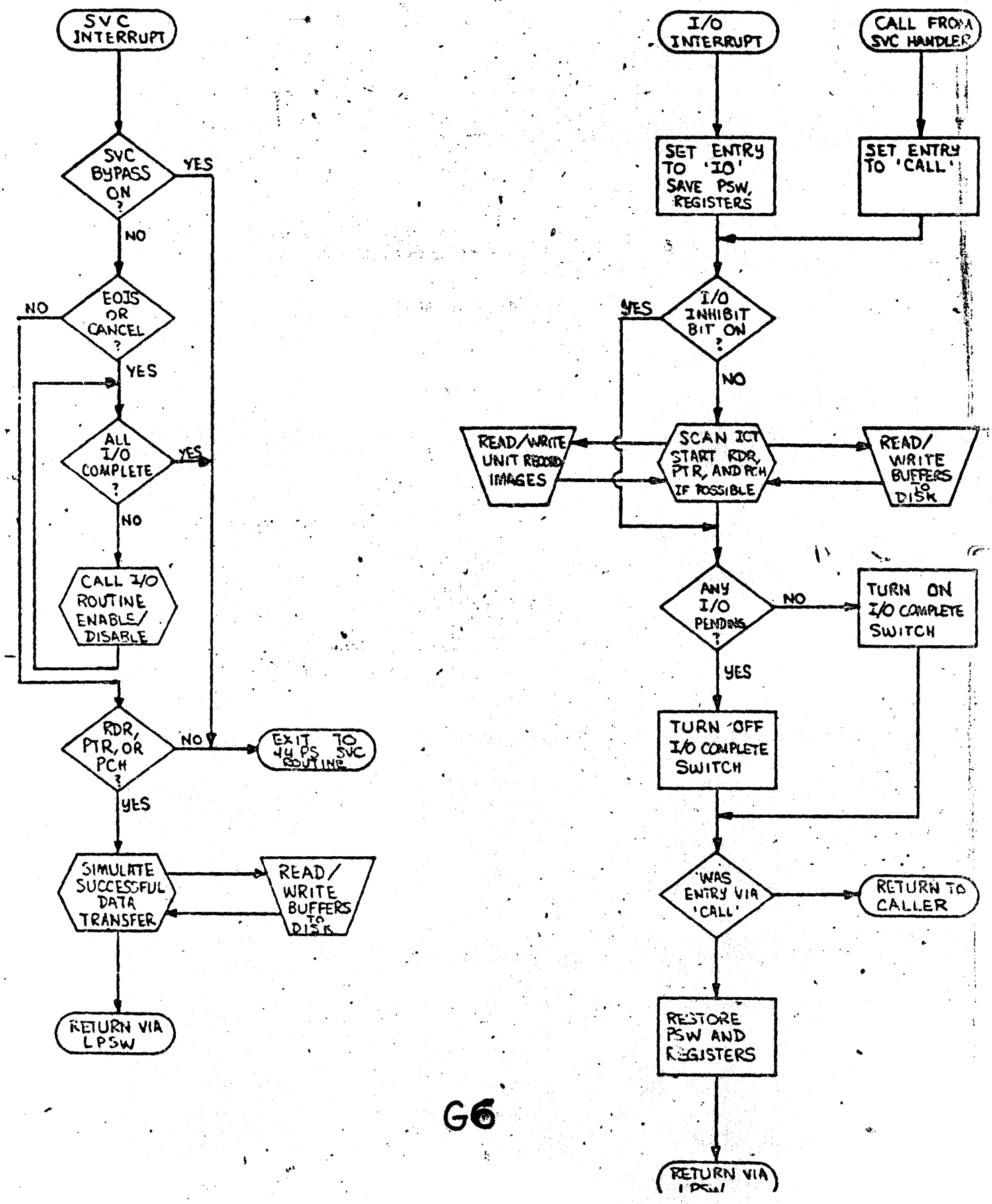


Figure 12. ESP GENERAL BLOCK DIAGRAM

ESP ADVANTAGES

IMPROVED THRUPUT

44PS COMPATABLE

IMPROVED ERROR RECOVERY

MINIMUM OPERATOR RETRAINING

REDUCED OPERATIONS OVERHEAD

ESP FEATURES

INPUT SPOOLING
OUTPUT SPOOLING
EFFECTIVE BUFFERING
EFFECIENT DISK QUEUING
OPERATOR INFORMATION

HARDWARE CONFIGURATION

128K OR 256K SYSTEM/360

10 TO 200 CYLINDERS DASD SPACE

READER, PRINTER, AND PUNCH
(UNIQUE ADDRESSES)

PHASE LIBRARY SPACE

PROGRAMMING SYSTEM REQUIREMENTS

44 PROGRAMMING SYSTEM
ESP (360D 03.4.023)

APPENDIX A.

44PS and ESP FORTRAN JOB TIME COMPARISON

Job Name	44PS	44PS with ESP	Ratio	Type
AP84	5.79	2.56	2.25	CLG
BA84	9.96	5.50	1.81	CLG
BA03	0.96	.55	1.74	C
AQ	1.78	.75	2.37	Disk X
AS18	27.57	12.66	2.18	LG
AS63	1.00	.75	1.35	Disk X
BA73	7.70	5.72	1.34	LG w/Plot
BA24	3.84	1.71	2.24	CLG
AN12	5.59	2.89	1.93	Disk X
BA52	25.82	14.88	1.79	LG
AN92	4.48	2.84	1.57	Disk X
11 jobs	94.49	50.80		

NOTE: All times are in minutes.

Legend for Type

C = Compile
L = Link Edit
G = Go, or Execute

Disk X = Load and Execute from disk.

G10

APPENDIX B.

44PS and ESP ASSEMBLER JOB TIME COMPARISON

	Non-ESP	ESP A	ESP B	ESP A Ratio	ESPB Ratio
JOB1					
D,L,J,M	4.94	1.99	1.52	2.47	3.24
JOB2					
C	2.04	.76	.75	2.68	2.72
JOB3					
B	3.22	1.40	.91	2.30	3.53
Total - 6 Assemblies	10.20	4.55	3.18		

NOTES:

1) The ESP A job stream consisted only of assembler jobs. The reader was not able to get ahead. Printing and punching were overlapped.

2) The ESP B job stream was preceded by a 5-minute compute bound job. This permitted all I/O to be overlapped.

3) All assemblies had DECK, LIST, and XREF options.

Figure 5. UNIT RECORD I/O RATES USING FORTRAN

Device	Record Length	Buffer Length	Sustained Record Rate
Reader	80 bytes	2880	426 records/second
	80	1440	185 records/second
Printer	133	2880	224 records/second
Punch	88	2880	400 records/second
	88	1440	177 records/second
Reader, Printer, and Punch	Mixed	Mixed	70 records/second

G12

27

EXTENSIONS

PAPER TAPE SPOOLING

PLOTTER SPOOLING

REMOTE JOB ENTRY

DATA ACQUISITION SESSION

Subject: Data Acquisition System Under TSX

Speakers: Allen Sandberg and David Rappaport

Address: Dept. of Bio-Medical Engineering
Presbyterian-St. Luke's Hospital
1753 W. Congress Pkwy.
Chicago, Illinois 60612

Phone: 312-663-2768

Session: Wed. Sept. 11, 1968 1:30 P.M.

Pages: Text - 51

Graphics - None

ABSTRACT

The Time-Sharing Executive System (TSX) as delivered by IBM lacks the flexibility necessary to make effective use of the hardware facilities of the 1800 particularly in an on-line real-time data acquisition environment.

To overcome many of the system oriented deficiences in TSX, a great number of relatively simple modifications were implemented such as: a keyboard entry facility for the priority queuing of either process or non-process programs; rapid repetitive execution of subroutines by a modified version of the IBM TIMER subroutine; processing of timer C interrupts on a low priority interrupt level, and many other items of particular use in data acquisition.

A sophisticated set of analog I/O subroutines was developed in order to enable the Fortran user to easily utilize the full capabilities of the 1800. Some of features available are: overlapped analog I/O; simultaneous A/D and D/A conversion useful in stimulus-response applications; independent sampling of different A/D or D/A channels; automatic double buffering and special synchronization options to control the relative timing between different A/D or D/A channels.

A Magnetic Tape System was developed for the organization and collection of data at rates commensurate with real-time problems. The system includes random overlapped accessing of data files on tape, overlapped writting/reading of variable length tape records, and a set of supporting utility programs to make magnetic tape as flexible a storage medium as disk within the obvious speed limitations of magnetic tape.

The hardware and software additions of teletypes to the computer enable the remote user in his laboratory to have control of the computer comparable to a user at the console. In addition, Fortran users are able to give formatted READ/WRITE statements when desiring teletype communications.

The above facilities coupled with a large set of assembly language array processing and other subroutines enable even the Fortran programmer to utilize the sophisticated data acquisition techniques needed in an on-line real-time environment.

TABLE OF CONTENTS

Table of Contents (Cont.)

I. Introduction

In a dynamic, on-line, real-time data acquisition environment, the Time-Sharing Executive System (TSX) distributed by IBM suffers from a lack of flexibility and effective use of the hardware facilities of the 1800. The main emphasis in a research environment, where programs are in a constant state of flux, is ease of programming without significant lose of sophistication. While TSX can even hinder the researcher who is willing to write his entire data acquisition program in assembly language, our ultimate goal was to provide a super-set of TSX and its related subroutines which would allow sophisticated data acquisition from a remote location using Fortran. To achieve this goal a number of modifications were made to TSX and extensive programming of auxiliary Fortran-callable subroutines was done. In total this effort can be broken down into five major categories:

(1) System Modifications

(2) Analog Input/Output Package

(3) Magnetic Tape System

(4) Teletype Communication Package

(5) Utility Subroutines.

The system modifications involved changes to the System Director, TASK, Non-process Supervisor and various related IBM supplied subroutines such as TIMER, SHARE, VIAQ and DPART. Such features as keyboard entry facility for the priority queuing of either process or nonprocess programs and the servicing of timer C interrupts on a low priority interrupt level are characteristic of the facilities needed for a remotely run data acquisition system.

Since our goal was in part flexible real-time data acquisition from Fortran, a sophisticated set of analog input/output subroutines was developed. This package was designed to allow the user to push to its fullest extent the analog conversion capabilities of the 1800. It allows for such things as analog I/O overlapped with processing or other I/O and simultaneous A/D and D/A conversion at independent rates. Double buffering and highly useful synchronization options were also included to allow for the controlling of the relative timings between different channels.

Once the Fortran programmer was able to collect analog data, software was necessary to allow its easy storage and later retrieval for processing. To meet this need, the Magnetic Tape System was developed for the organization and collection of data at rates commensurate with real-time problems. The system provides for the overlapped read-

ing and writing of variable-length data records which may be organized into randomly accessible files. Overlapping is provided during all magnetic tape operations to greatly enhance the use of this large capacity, but moderately slow, storage medium.

The development of hardware and software allowed the addition of teletypes to the 1800. These provided a low cost I/O device which could be conveniently placed remote to the computer. Through their addition the full power of our data acquisition system was made available to the researcher in his laboratory.

While it would have been nice to let the user do all his data manipulation in Fortran, it was apparent that for many real-time data processing problems Fortran, was just too slow. To overcome this deficiency a large set of Fortran - callable array processing subroutines were written in assembly language. Through their use the programmer is able to fashion his program in Fortran to meet the requirements of fast on-line real-time data analysis.

In general, the hardware facilities provided by the 1800 has proved capable of coping with all of our real-time data acquisition problems. We can also see how the TSX system might solve the majority of the problems encountered in an environment more stable than Biomedical research. Hence, we offer the system described in this paper as a means of extending the capabilities of the 1800 to realize its effective use in an on-line data acquisition environment.

II. System Modifications and Additions

This section will briefly describe the main changes or additions made to the TSX system in order to make it a more dynamic operating system. In all fairness to IBM we must give them a great deal of credit in preparing the extensive documentation of the software which made practical many of the changes.

A. Nonprocess Priority Queue.

Since our system was to be used from a remote point the concept of using the card reader as a nonprocess queue was quite restrictive. It would be much more useful if there existed a priority queue for nonprocess programs similar to the one already provided by IBM for process programs. We therefore implemented the concept of nonprocess queeue.

In order to establish a nonprocess queue it was necessary to be able to get the word count and sector addres of any coreload on the disk. For this purpose a FLET searching subroutine was written. It is called with the name of the coreload desired stored in a five-element array and returns, if possible, with that coreload's word count and sector address.

With the establishment of a queue for nonprocess programs, the organizational arrangement became very complicated. Some of the features of this queue and its organization relative to the process queue and jobs waiting to be executed from the card reader are as follows:

(1) Programs entered in the nonprocess queue are assigned priorities 1-32767 with 1 the highest.

(2) All programs in the process queue have higher priority than those in the nonprocess queue. This is accomplished by only searching the nonprocess queue when either the process queue is empty or time-sharing is in effect and the present nonprocess job has been completed.

(3) Jobs waiting in the card reader have effectively a priority of 4095. That is, nonprocess jobs queued with a priority greater than 4095 will be executed no matter what is in the card reader. However, queued jobs with a priority lower than 4095 will not be executed until the card reader has become not ready.

(4) Once a nonprocess job has been started it will continue to completion no matter what its priority is relative to the nonporcess queue or the card reader ready status.

This briefly is the organization of the nonprocess queue. Its implementation involved modifications to the System Director, the Non-process Supervisor and such Fortran oriented routines as SHARE, EXIT and VIAQ.

B. Queuing of Nonprocess and Process Programs.

In order to fully utilize the power provided by the presence of process and nonprocess priority queues, the ability to provide the name of the coreload to be queued at execution time was required. In contrast, the QUEUE subroutine requires the name to be provided at compilation time.

The Fortran - callable subroutine which provide this feature are:

CALL PRQUE (NAME, IP, IE)

CALL NPQUE (NAME, IP, IE)

where the arguments are:

NAME.....A five element fixed point array which contains the coreload name in 5A1 format. This array can be filled by a DATA statement or an appropriate input operation.

IP.....The priority at which the coreload will be queued.

IE.....A variable which designates the error procedure to be taken if the queue is full.

IE=0	Return with IE set to 0 if the queue is not filled and the call was executed. Set IE to 1 if queue filled.
IE=1-32766	Ignore the call and continue execution if queue filled.
IE=32767	Execute the restart coreload.

The complimentary routines are also provided to allow specification, at execution time, of the name and priority of a coreload to be unqueued.

C. XEQ Interrupt Coreload.

To further extend the capabilities of the 1800 as a usable teleprocessing system, the procedure of cold starting process jobs was completely abandoned. All jobs to be run, whether process or nonprocess, must first be placed in the appropriate queue. This is done by means of an interrupt coreload.

This interrupt coreload, called XEQ, is brought into core by placing sense switch 7 down and pressing console interrupt. Once brought into core, it will request the operation to be performed, the coreload name and the priority of the coreload.

Currently the required information must be entered in fixed format starting in column one. The required information is:

OPERATION CORELOAD NAME PRIORITY

where:

Operation Code:	Q.....for queuing U.....for unqueuing DQ.....for dumping the current status of the queues EXIT....ignore call
Coreload Name:	The 1-5 character name of the coreload to be queued or unqueued.
Priority:	H.....for hospital projects O.....for on-line experiments C.....for computational programs B.....for background programs

In this priority scheme, all the priority codes are legal for nonprocess programs and are ranked in the order listed above. Hospital projects are the highest, computational programs the lowest while still above the card reader and background programs unconditionally the lowest. In the case of process programs all programs are assigned an on-line priority when entered through this facility.

When the coreload name entered is legal, i.e. the coreload is present on the disk, the computer returns with the operation being performed, the type of coreload (process or nonprocess), the coreload name and the priority assigned. The operation is then attempted and if completed the machine is returned to the state it was in when XEQ was requested. If the operation if not successful, as in the case of the queue being full, a message to that effect is printed out and the machine returned to its original state.

If the coreload name entered is not legal the message,

D 25 NAME NOT IN L/F

is printed out and the machine returned to its original status.

If the information entered is either illegal, i.e. improper operation or priority, or the format was incorrect, the message ERROR will be typed out and the machine is returned to its original state.

When the operation code entered is DQ, no additional information is required. The program will print out the list of program names and the priorities at which they were queued for both the process and nonprocess queues.

If XEQ had been brought into core by mistake, the user can enter EXIT and have the machine immediately resume the interrupted job.

The I/O device used by XEQ is determined at the time it is brought into core by means of sense switch 6. When SS6 is in its normal position, i.e. down, the teletype will be used and when it is set, i.e. up, the console typewriter is used as its I/O device. In addition, there exists the ability to simulate the process of putting sense switch 7 down and pressing console interrupt by pressing certain special keys on the teletype. These features will be discussed in detail in Section V.

D. Interval Timer Control (ITC) and TIMER Changes.

The IBM supplied TIMER subroutine and the ITC section of the System Director are useful for the 'single shot' execution of a specified program. In data acquisition what is more often needed is the ability to initiate (with a single call) the rapid and repetitive execution of a specified subroutine. Suitable changes were made to ITC and the TIMER subroutine to accomplish the above. The replacement TIMER subroutine has two additional entry points used to stop repetitive mode operations. The new subroutine is described as follows:

CALL TIMR (NAME, ITIM, ICNT)

NAME.....Name of a subroutine the user wishes to execute when the specified timer times out. NAME must appear in an EXTERNAL statement.

ITIM.....Timer and mode specification variable

1.....	Timer A used	Single shot mode
2.....	Timer B used	Single shot mode
-1.....	Timer A used	Repetitive mode
-2.....	Timer B used	Repetitive mode

ICNT.....Positive number which equals the number of timer counts before timer times out and subroutine NAME executed.

Blast Timer (BLSTM)

CALL BLSTM (I)

I= { 1....Timer A Stopped immediately
2....Timer B Stopped immediately

Stop Timer (STPTM)

CALL STPTM (I)

I= { 1....Timer A mode changed to single shot
2....Timer B mode changed to single shot

By single shot mode we mean that when the timer times out the specified subroutine is executed and the timer is turned off. In repetitive mode the timer time is automatically reset to its initial value (ICNT) as well as the subroutine NAME being executed when the timer times out. Unless the timer is stopped it will continue to cause the execution of NAME indefinitely.

E. Multi-Programmed Interrupts per Level.

One of the drawbacks of TSX (which has been eliminated in MPX) is the inability to have more than one programmed interrupt per level. Following some suggestions made by Don Weber of Abbott Laboratories, our TSX system now "effectively" has this multi-programmed interrupt per level capability.

The ordinary programmed interrupt capabilities of TSX are lost but equivalent and expanded facilities are obtained by being able to cause (under program control) what TSX thinks are process interrupts. This is accomplished by adding two words at the beginning of each interrupt level work area. One word contains a phony ILSW bit and the other contains phony PISW bits. Immediately after MIC senses the true ILSW we OR in the phony ILSW word. If the assignment and System Director equate cards have been correctly prepared and the appropriate phony ILSW bit was present, then the system will funnel us to PRIE in the System Director. After MIC senses the true PISW bits we OR

in our phony PISW word. From then on the system can not distinguish between true process interrupts and our phonied ones.

Thus we have utilized the TSX capacity for multi-process interrupt bits per level in order to obtain more than one program initiated interrupt per level. The user establishes linkage to the phonied up process interrupt word via the appropriate *INCLD cards at coreload build time. As in MPX the LEVEL subroutine was recoded so that the Fortran user can easily cause a particular interrupt on any interrupt level and bit position.

F. Timer C Processing on a Low Priority Level.

Many real-time data acquisition programs can be significantly compromised due to a timer C interrupt and the subsequent system oriented processing on the normally high priority level of the interval timers. Since the interrupt level assignments of the three timers can not be separately specified one must find some compromising software solution. This has been done at Presbyterian St. Luke's Hospital by rapidly determining the occurrence of a timer C interrupt and effectively causing a programmed interrupt to a lower level. Hooked to that interrupt level is a subroutine which branches back into the ITC coding pertinent to timer C processing. It takes approximately 70 microseconds to recognize a timer C interrupt, setup for a lower priority interrupt, and return to the interrupted program.

G. Modified CALL SPECL

The CALL SPECL facility in TSX is a very convenient way to execute a given program generally unrelated to the present coreload being run. That is, the special coreload was not intended to communicate large amounts of in-core data back to the calling program. If one desires extensive communication between the mainline and special coreloads the only way of accomplishing this is either through process working storage of the fixed area of the disk. In-skeleton COMMON can be used, but it is generally reserved for the communication of small amounts of data such as flags, etc.

We have provided the facility, if desired, for mainline and special coreloads to communicate via ordinary COMMON. If the user wishes to communicate via ordinary COMMON he calls a subroutine MSPCL (Modify Special) immediately before the CALL SPECL. When the present coreload is saved and brought back via the CALL BACK only the program, and not the COMMON, is written and read to and from the disk respectively. At the time the mainline program is to be saved on the disk we compute the true word count by adding the word count in VCORE to the size of the largest local group. This word count is used in saving

INTERRUPT LEVEL SUBPROGRAMS

& WIGH

INTERRUPT SERVICE SUBPROGRAMS

INTRODUCTION

The following paper was prepared for a tutorial session on the 1130 interrupt handling procedures to be given at the Philadelphia CØMMØN Meeting on September 9-11, 1968. The tutorial is based on a sample program used by the Chicago IBM Educational Center.

The interrupt handling procedures available to an 1130 BAL programmer are discussed through the use of two Input/Output units. The 1442 Card Reader is programmed to accept data from punched cards. The Console Printer is used as the output device. Discussion is carried through great lengths in describing the linkage between the program and the response of the programmed unit.

Bruce K. Anderson
Senior Applications Analyst
Operations Research Group
Uniroyal, Inc., Chemical Div.
Naugatuck, Connecticut 06770

August 7, 1968

ILS & ISS

(Fig. 1)

The purpose of this talk is to give the 1130 BAL programmer some insight into the interrupt philosophy of this computing machine. Because this talk is to be a tutorial session, I will use a program which I received from the IBM Chicago Educational Center to illustrate each point as I explain it.

The definition of an interrupt is "an automatic branch generated by the CPU based on an external impulse". This impulse may be generated by one of the Input/Output units or by the closing of a contact such as the "Program Stop" button. Because an interrupt generated by one of the I/O devices may have more importance than another interrupt generated by a different I/O device, each generated interrupt will have associated with it a priority or interrupt level. For example, an interrupt generated by an 1132 printer is defined as more important than an interrupt generated by the "Program Stop" button. On the 1130 computer there are six priority levels ranging from Zero to Five, with Zero being the highest and Five being the lowest. This idea of priority allows the programmer to service the more important I/O device first. Devices of lower priority will be serviced after the more important devices. Because of the number of different types of I/O units that can generate interrupts, it is not always feasible or practicle for the CPU to branch to a unique core location where the interrupt sercive program is located. Instead the concept of the Interrupt Branch Address Table is employed.

Fig. 1

ISS

AND

ILS

(Fig. 2)

The Interrupt Branch Address Table is a six-word table which contains the addresses of the subroutines which have been created to service a given level interrupt. When an I/O unit generates an impulse, which in turn causes the CPU to generate an interrupt, the effective address of the service subprogram is developed using the constant stored at the interrupt branch table. For example, a level Zero interrupt would cause a branch to the core location whose address is stored in core position Eight. Similarly, a level Five interrupt would cause a branch to the core position whose address was stored in core position Thirteen. It is up to the programmer to determine which unit caused the interrupt and what action should be taken to service that interrupt.

To summarize the idea of interrupts; they are generated by the CPU in response to an external condition. The CPU generated branch is guided to the correct subroutine to service that interrupt by the use of addresses that are stored in the interrupt branch table.

Fig. 2 & 24

INTERRUPT BRANCH ADDRESSES

CORE ADDRESS	CONTENTS
8	ADDRESS OF INTERRUPT LEVEL 0 SUBROUTINE
9	ADDRESS OF INTERRUPT LEVEL 1 SUBROUTINE
10	ADDRESS OF INTERRUPT LEVEL 2 SUBROUTINE
11	ADDRESS OF INTERRUPT LEVEL 3 SUBROUTINE
12	ADDRESS OF INTERRUPT LEVEL 4 SUBROUTINE
13	ADDRESS OF INTERRUPT LEVEL 5 SUBROUTINE

(Fig. 3)

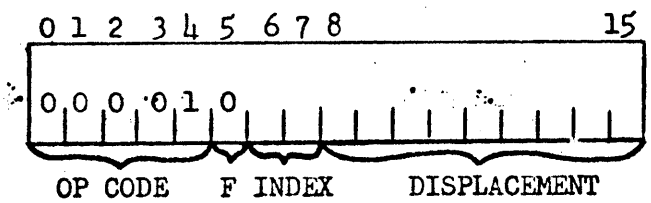
The 1130 computing system has just one way of initiating data transfer to or from the CPU. This is accomplished by the "Execute I/O" instruction whose mnemonic op code is "XIØ". The format of this instruction may be either short or long depending upon the relative location of an additional two control words. These two words are called the Input/Output Control Command or IØCC. Indexing and indirect addressing are permitted in the XIØ format. It should be noted that the contents of the accumulator should be saved before the execution of the XIØ instruction because the accumulator is used in analyzing the Input/Output Control Command.

Fig. 3

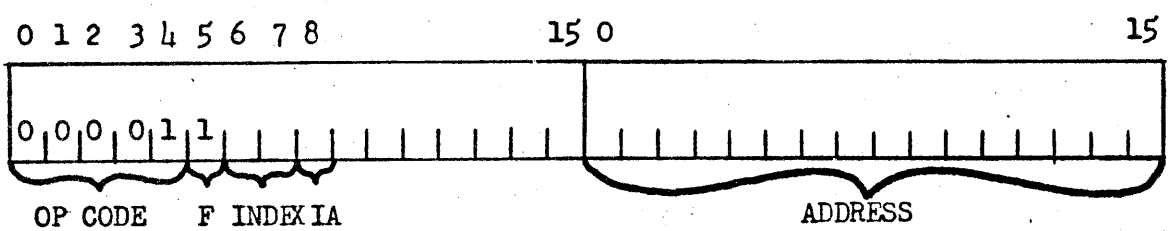
XIO INSTRUCTION FORMAT

MNEMONIC	OP CODE	FORMAT	IX	IA
XIØ	00001	S/L	NO/YES	NO/YES

SHORT FORM



LONG FORM



(Fig. 4)

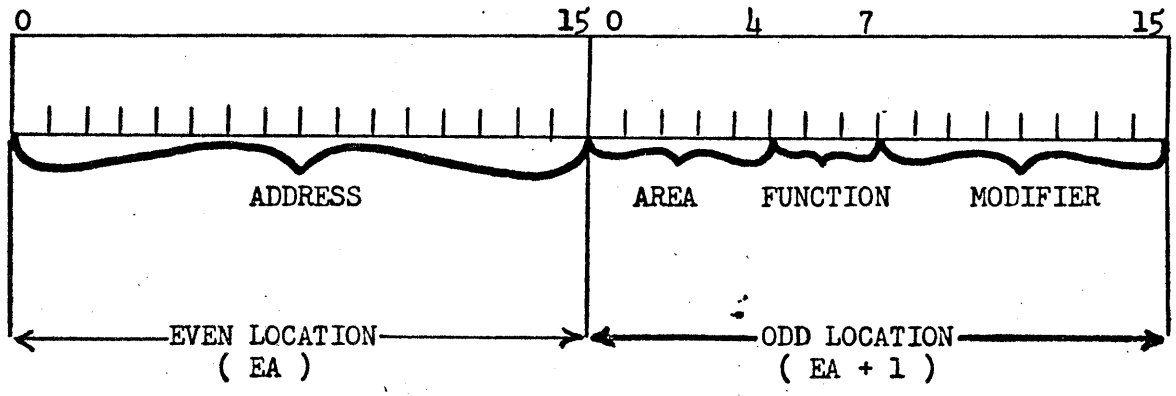
The IØCC is used to provide additional information as to what device is being acted upon and what action is being taken by that device. The placement of the IØCC must start at an even location; therefore, the effective address of the XIØ instruction must always be even. The format of the IØCC is broken down into four logical functions. The first word of the IØCC contains the address of the data that is to be worked upon. The second word of the IØCC contains the other three logical functions. They are: 1) the area code, which specifies the device upon which the data will be transferred; 2) the function code, which will describe what is to be accomplished by the Input/Output device; and 3) the modifier code, which gives additional information for the device and function specified.

In summary, there is one Input/Output instruction available in the 1130 computing system. The effective address of this instruction points to a two-word Input/Output Control Command which provides additional information in explaining the desired input or output function.

Fig. 4

INPUT OUTPUT CONTROL COMMAND

IØCC

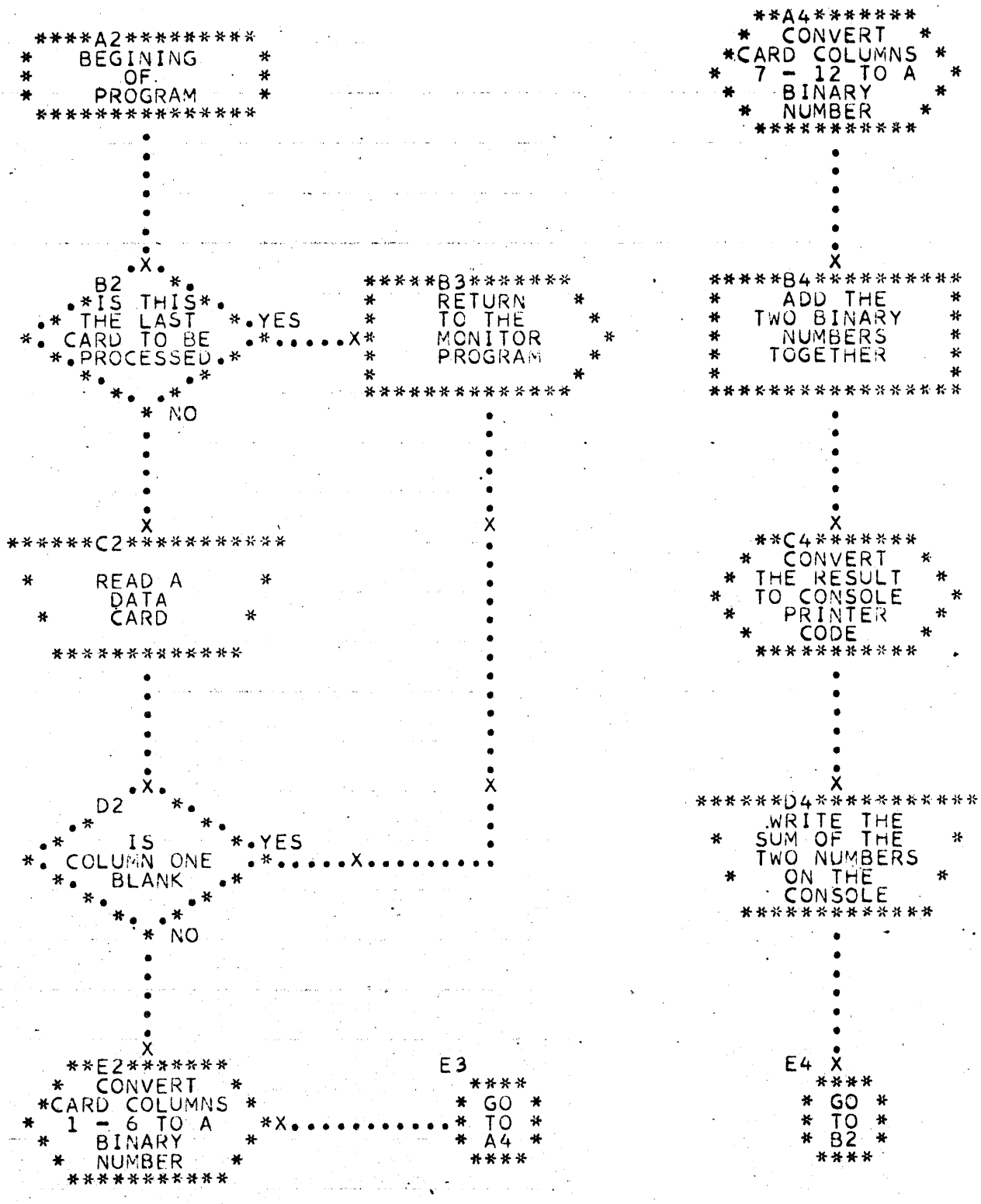


AREA		FUNCTION	
CODE	DEVICE	CODE	DESCRIPTION
00001	CONSOLE	000	NOT USED
00010	1442	001	WRITE
00011	1134	010	READ
00100	DISK STORAGE	011	SENSE INTERRUPT
00101	1627	100	CONTROL
00110	1132	101	INITIATE WRITE
00111	C E S	110	INITIATE READ
01000	1231	111	SENSE DEVICE
01001	2501		
01010	S C A		
10001	DISK 1		
10010	DISK 2		
10011	DISK 3		
10100	DISK 4		
10101	1403		

(Fig. 5)

To clarify some of these points, I am going to use a program which will use the XIØ instruction. It will also use different interrupt levels and different I/O devices. As shown by the basic flowchart of this program, a card will be read by the 1442 card reader. Column One will be checked for a blank. If Column One is blank, control will be returned to the monitor; if it is not blank, the program will extract information from this card. Card Columns One thru Six and Card Columns Seven thru Twelve will be converted to two binary numbers. These two numbers will be added together and the resulting sum will be converted into Console Printer Code. This number will then be written on the console. The program will continue in this cycle until either the last card has been processed or a blank in Column One is encountered. In either case, control is returned to the monitor program.

Fig. 5


A2
BEGINING OF PROGRAM
B2
IS THIS THE LAST CARD TO BE PROCESSED
YES
NO
B3
RETURN TO THE MONITOR PROGRAM
A4
CONVERT CARD COLUMNS 7 - 12 TO A BINARY NUMBER
B4
ADD THE TWO BINARY NUMBERS TOGETHER
C2
READ A DATA CARD
D2
IS COLUMN ONE BLANK
YES
NO
C4
CONVERT THE RESULT TO CONSOLE PRINTER CODE
D4
WRITE THE SUM OF THE TWO NUMBERS ON THE CONSOLE
E2
CONVERT CARD COLUMNS 1 - 6 TO A BINARY NUMBER
E3
GO TO A4
E4
GO TO B2

(Fig. 6)

The first thing that should be considered in studying this program is the operation of the 1442 Card Reader under program control. There are two interrupt levels associated with the card reader. An interrupt on level Zero will occur every time a card column is positioned over the read head. This will enable the program to read that column into core storage. A level Four interrupt will occur after all eighty columns have been read, and it will serve as an "Operation Complete" interrupt.

In programming the 1442 Card Reader, two interrupt service subprograms must be stored in core to handle the different levels of interrupts that will be generated. The first will handle level Zero interrupts; the second will handle level Four interrupts. The one assumption to be made is that only the 1442 Card Reader will generate these interrupts. Interrupts that may at this time occur for devices other than the 1442 Card Reader will not be handled by this program.

In programming this problem, the first thing that must be accomplished is the loading of the Interrupt Branch Address Table with the addresses of the subprograms that will service the two interrupts. Therefore, core location Eight will be loaded with the address of the subprogram that will handle the level Zero interrupts. Likewise, core location Twelve will be loaded with the address of the subprogram that will handle the level Four interrupts. Following this the last card situation should be tested to see if the last card has been processed. If it has, program control should be returned to the monitor. If the last card has not been processed, a check is made on the 1442 Card Reader to see if it is ready to begin the reading operation. Such things as a "Hopper Check" or a full stacker could have made the reader "Not Ready". The testing of the reader to see if it is "Ready" can be

(Fig. 6, Cont'd.)

accomplished by loading a word called the Device Status Word into the accumulator. This will contain information about the 1442 Card Reader.

(Fig. 7)

The Device Status Word is loaded into the accumulator by the execution of an XIØ instruction. The IØCC associated with this instruction should be set up so that the area code is interpreted as the 1442 Card Reader and the function code is interpreted as "sense device". The modifier section is not required when loading the Device Status Word. By the execution of this XIØ instruction the Device Status Word is placed in the accumulator for interpretation by the program.

Fig. 6

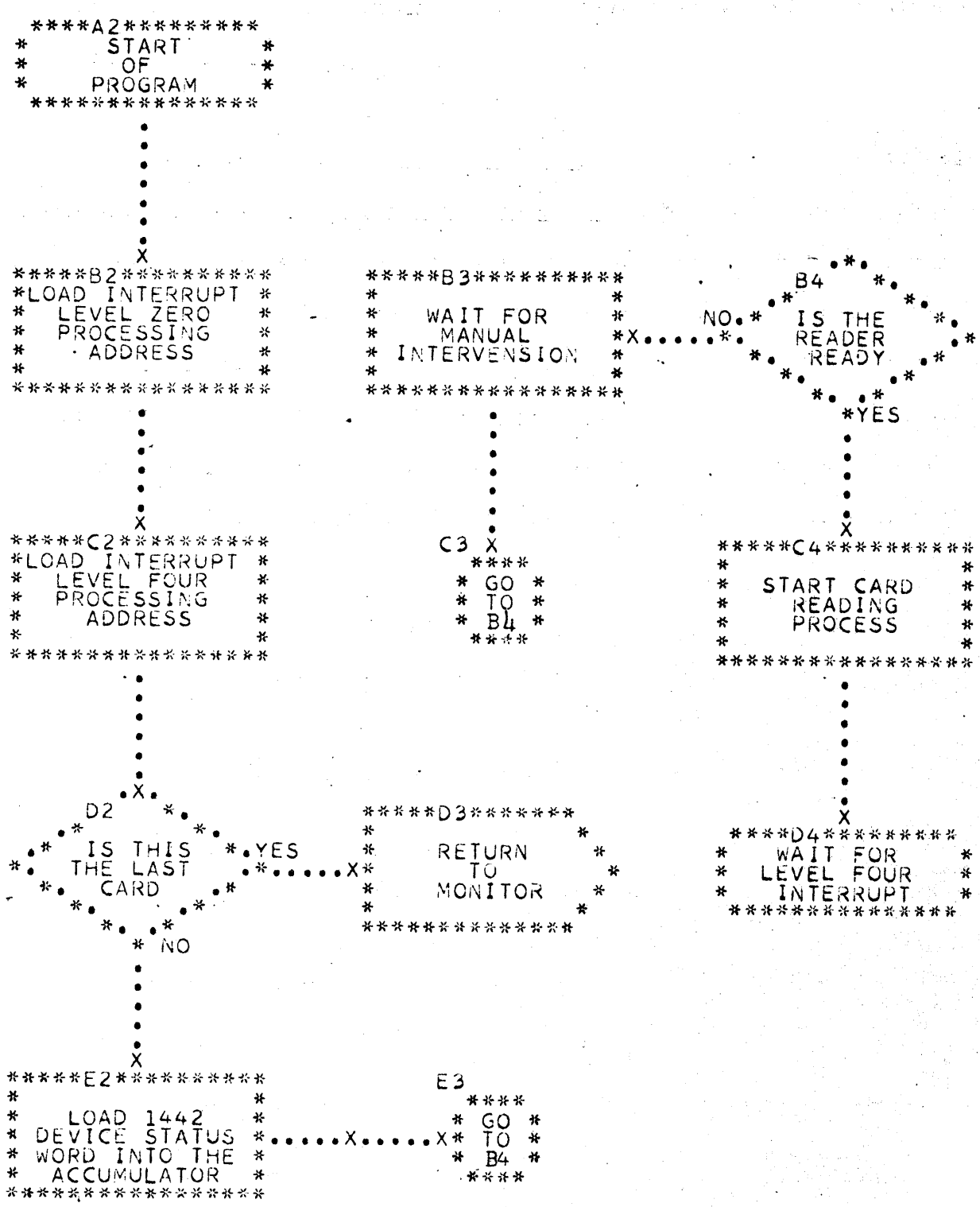

A2
START OF PROGRAM
B2
LOAD INTERRUPT LEVEL ZERO PROCESSING ADDRESS
B3
WAIT FOR MANUAL INTERVENSION
B4
IS THE READER READY
NO
YES
C2
LOAD INTERRUPT LEVEL FOUR PROCESSING ADDRESS
C3
GO TO B4
C4
START CARD READING PROCESS
D2
IS THIS THE LAST CARD
YES
NO
D3
RETURN TO MONITOR
D4
WAIT FOR LEVEL FOUR INTERRUPT
E2
LOAD 1442 DEVICE STATUS WORD INTO THE ACCUMULATOR
E3
GO TO B4

Fig. 7

INSTRUCTION

LABEL	OP CODE	OPERAND	COMMENTS
CHK	XIØ	SENS	LOAD DEVICE STATUS WORD

IØCC

SENS	BSS	E	0	
	DC		/0000	NOT USED
	DC		/1700	(00010/111/00000000)

INTERPRETATION

AREA =	00010	1442 READER
FUNC =	111	SENSEDEVICE
MODIFIER		NOT USED

LOAD THE CPU ACC WITH DEVICE STATUS WORD

(Fig. 8)

The make-up of the Device Status Word is generated automatically by the selected device. By having the program check Bit 15 to see if it is on, it can determine whether or not the 1442 Card Reader is ready. If it is ready, the reading of the data card may begin. If it is not ready, additional checking of the Device Status Word may be programmed so as to determine the cause of the "Not Ready" condition. If the Card Reader is not ready some sort of message should be programmed so as to alert the operator that some manual intervention is required.

When the Card Reader is "Ready" the reading process of this card may commence by the execution of an XIØ instruction with the correct IØCC.

(Fig. 9)

The interpretation of this IØCC is to specify the card reader in the area code, a control in the function code, and modifier Bit 13 on. The modifier bit will be interpreted as a "start reading" instruction. This will cause the Card Reader to begin to physically move the card across the read station. The next instruction after this is a "wait" command. The reason for this instruction is to allow the card to be positioned correctly for the reading operation.

Fig. 8 & 16.

1442

DEVICE STATUS WORD

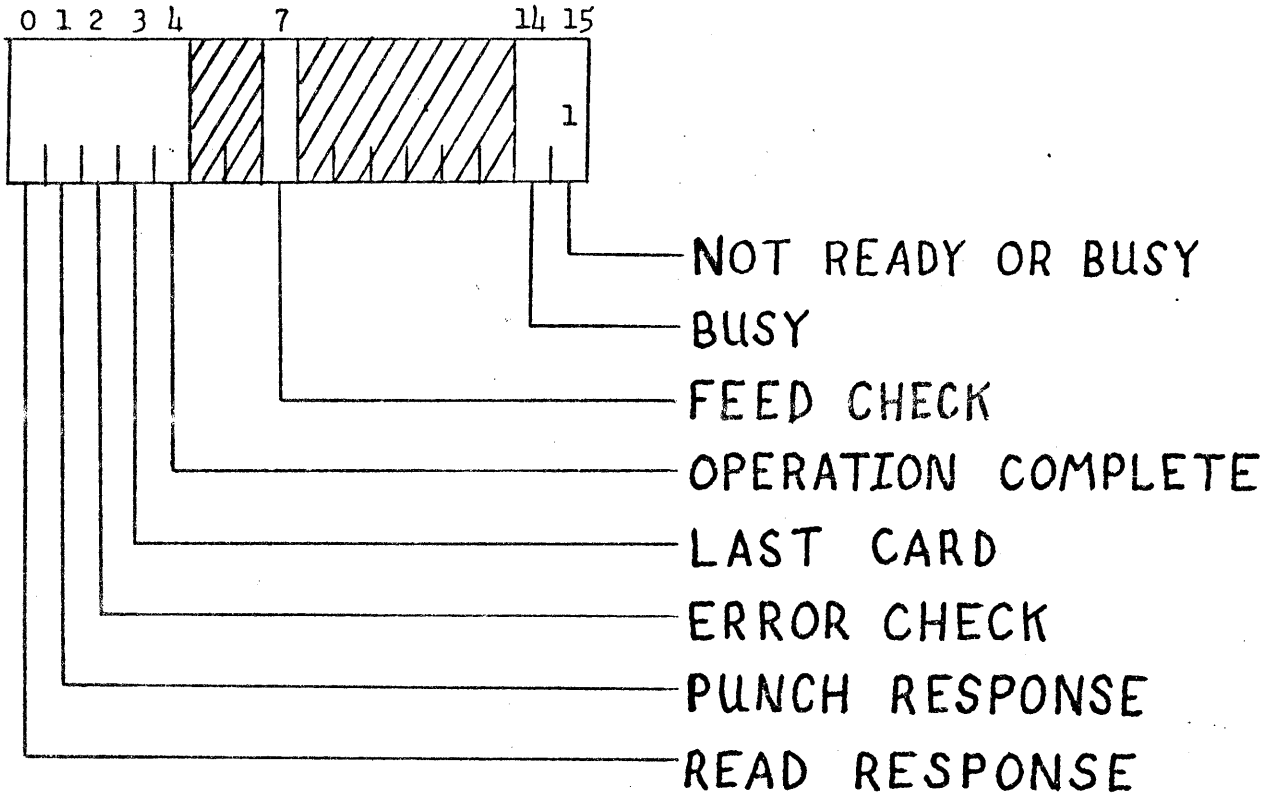


Fig. 9

INSTRUCTION

LABEL	OP CODE	OPERAND	COMMENTS
	XIØ	START	START READING A CARD

IØCC

	BSS	E O	
START	DC	/ 0000	NOT USED
	DC	/1404	(00010/100/00000100)

INTERPRETATION

AREA =	00010	1442 READER
FUNC =	100	CØNTRØL
MODIFIER =	BIT 13	STARTS READING

(Fig. 10)

When a card column is in place and ready to be read into the CPU a level Zero interrupt will be generated. It is in this interrupt level Zero subprogram that the actual reading takes place.

In summary, the 1442 Card Reader makes use of two levels of interrupts; level Zero for card column reading, and level Four for "Card Operation Complete" signalling. The programmer must load the Interrupt Branch Address Table with the addresses of the subprograms that will service these interrupts. The programmer must also check the status of the Card Reader before the card reading operation can begin. Once the Card Reader is found "Ready", a "Start Read" process instruction may be executed which will begin the physical movement of the card across the reading station. The actual data transferral from the card to the CPU is accomplished within the level Zero subprogram.

(Fig. 11)

Entrance into the level Zero subprogram is initiated by a CPU generated level Zero interrupt. The address of this subprogram has been stored in core position Eight. This subprogram will accomplish three main functions. The first is to reset the hardware interrupt which caused the generated branch to this subprogram. The second is to transfer the card column image which is presently at the read station to a given address in core storage. The third is to increase this given address by one so that the next card column read will be placed in sequence in core storage. A return command is given which will transfer control back to the position in core that the program was before the level Zero interrupt occurred. Accompanying this return is an interrupt indicator reset.

Fig. 10

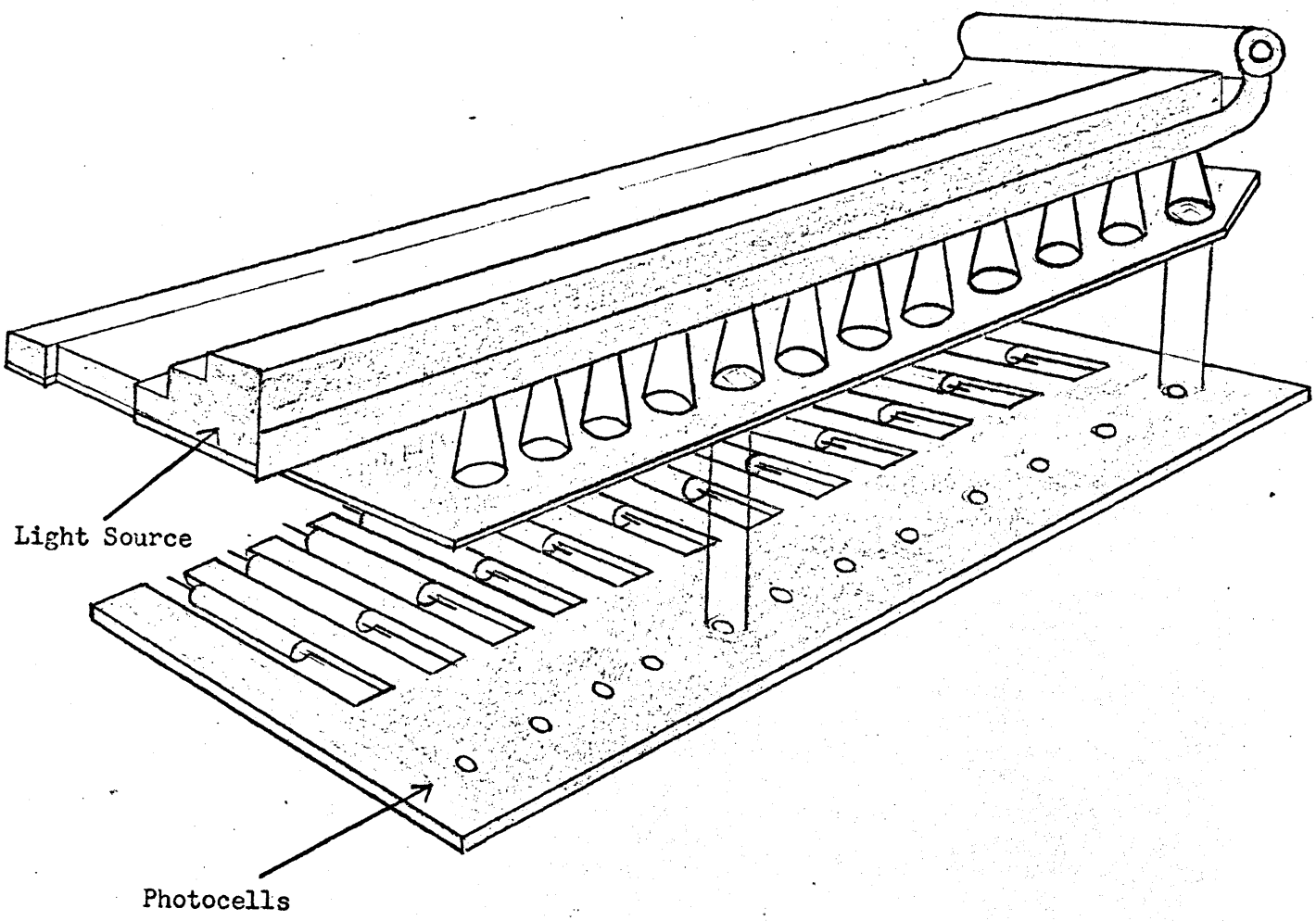


Fig. 11

```
                    *****A3**********
                    *               *
                    *     ENTRY     *
                    *     POINT     *
                    *               *
                    *               *
                    ****************
                          .
                          .
                          .
                          .
                          .
                          .
                          X
                    *****B3**********
                    *               *
                    *     RESET     *
                    *   HARDWARE    *
                    *   INTERRUPT   *
                    *               *
                    ****************
                          .
                          .
                          .
                          .
                          .
                          .
                          X
                    *****C3**********
                    *               *
                    * TRANSFER CARD *
                    *COLUMN TO CORE *
                    *    STORAGE    *
                    *               *
                    ****************
                          .
                          .
                          .
                          .
                          .
                          .
                          X
                    *****D3**********
                    *               *
                    * BUMP TRANSFER *
                    *  ADDRESS BY   *
                    *     ONE       *
                    *               *
                    ****************
                          .
                          .
                          .
                          .
                          .
                          .
                          X
                    *****E3**********
                    * RETURN TO THE *
                    *  PROCESSING   *
                    *  PROGRAM AND  *
                    *RESET INTERRUPT*
                    *   INDICATOR   *
                    ****************

X

THE ADDRESS
OF THIS WORD
IS STORED IN
CORE LOCATION
    8
```

(Fig. 12)

As stated before the first function of the level Zero interrupt subprogram is to reset the hardware interrupt. This is accomplished by executing an XIØ instruction with an IØCC containing a 1442 Card Reader in the area code, a sense in the function code, and Bit 15 of the modifier code on. With this Bit on, the CPU will reset the level Zero hardware interrupt that caused the branch into this subprogram.

(Fig. 13)

Data transferral is accomplished by executing an XIØ instruction with an IØCC specifying the 1442 Card Reader in the area code, and a "Read Column" in the function code. The modifier code is not used. The first sixteen Bits in the IØCC contain the core address to which the card column image will be stored.

In summarizing the level Zero interrupt subprogram, it will reset the hardware interrupt, transfer one column of data, and increment the address at which the data will be stored. In reading a data card, this level Zero subprogram will be entered eighty times, once for each card column read.

Fig. 12

INSTRUCTION

LABEL	OP CODE	OPERAND	COMMENTS
	XIØ	SENSO	RESET HARDWARE INTERRUPT

IØCC

	BSS	E 0	
SENSO	DC	/0000	NOT USED
	DC	/1101	(00010/111/00000001)

INTERPRETATION

AREA	= 00010	1442 READER
FUNC	= 111	SENSE
MODIFIER	= BIT 15	WILL RESET LEVEL 0 INTERRUPT

Fig. 13

INSTRUCTION

LABEL	OP CODE	OPERAND	COMMENTS
	XIØ	READ	TRANSFER CARD COLUMN TO CORE STORAGE

IØCC

	BSS	E	0
READ	DC		CRDIN
	DC		/1200 (00010/010/00000000)

INTERPRETATION

AREA	=	00010	1442 READER
FUNC	=	010	READ COLUMN
MODIFIER			NOT USED

(Fig. 14)

After all eighty columns of the card have been read and stored into core, a level Four interrupt will occur. The subprogram for this interrupt will accomplish three main functions. The first is to check for a last card condition. The second is to reset the input area address that is part of the "Column Read" IØCC. And the third is to return to the processing program and reset the interrupt indicator.

Entrance into the level Four subprogram is initiated by a CPU generated level Four interrupt. The address of this subprogram has been stored in core position Twelve. The first function to be accomplished is a last card check.

(Fig. 15)

By loading the Device Status Word of the Card Reader into the accumulator, a last card condition can be checked. The loading of the Device Status Word is accomplished by executing an XIØ instruction with an IØCC command in the function code, and Bit 14 on. Bit fourteen will reset the level Four hardware interrupt, which caused the branch into this subprogram.

Fig. 14

```
*****A2**********
*               *
*     ENTRY     *
*     POINT     *
*               *
*               *
*****************

THE ADDRESS
OF THIS WORD
IS STORED IN
CORE LOCATION
12

*****B2**********
*               *
*LOAD THE DEVICE*
*STATUS WORD AND*
*RESET INTERRUPT*
*               *
*****************

*****B4**********
*               *
*  RESET CARD   *
*  INPUT AREA   *
*    ADDRESS    *
*               *
*****************

      C2
   IS THE
 LAST CAR
  BIT ON
  NO
  YES

*****C4**********
* RETURN TO THE *
*  PROCESSING   *
*  PROGRAM AND  *
*RESET INTERRUPT*
*   INDICATOR   *
*****************

*****D2**********
*               *
* SET LAST CARD *
* SWITCH TO BE  *
*   NON-ZERO    *
*               *
*****************
```

Fig. 15

INSTRUCTION

LABEL	OP CODE	OPERAND	COMMENTS
	XIØ	SENS4	RESET HARDWARE INTERRUPT

IØCC

	BSS	E 0	
SENS4	DC	/0000	NOT USED
	DC	/1702	(00010/111/00000010)

INTERPRETATION

AREA	=	00010	1442 READER
FUNC	=	111	SENSE COMMAND
MODIFIER	=	BIT 14	WILL RESET LEVEL FOUR INTERRUPT

(Fig. 16)

The Device Status Word can now be checked for a "last card" condition by interrogating Bit Three which is the "last card" indicator. If Bit Three is on, the last card switch in the program should be set to a "non-Zero" condition. The next operation is to reset the card input address area in the first sixteen Bits of the IØCC associated with a read response command. At this point control can be returned to the processing program.

In summarizing the level Four interrupt subprogram, it will reset the hardware indicator, check for a "last card" condition, reset the card input address, and return to the processing program. Once the level Four subprogram has been completed, eighty words of core storage will contain an image of the data card that was just read.

(Fig. 17)

Continuing with the mainline program, Column One is checked for a blank. If a blank is found, control is returned to the monitor program. If no blank is found, Columns One to Six will be converted to a binary numger and Columns Seven to Twelve will be converted to another binary number. These two numbers are added together and the results will be converted into console printer code. At this point the data is ready to be transferred to the Console Printer.

1442

DEVICE STATUS WORD

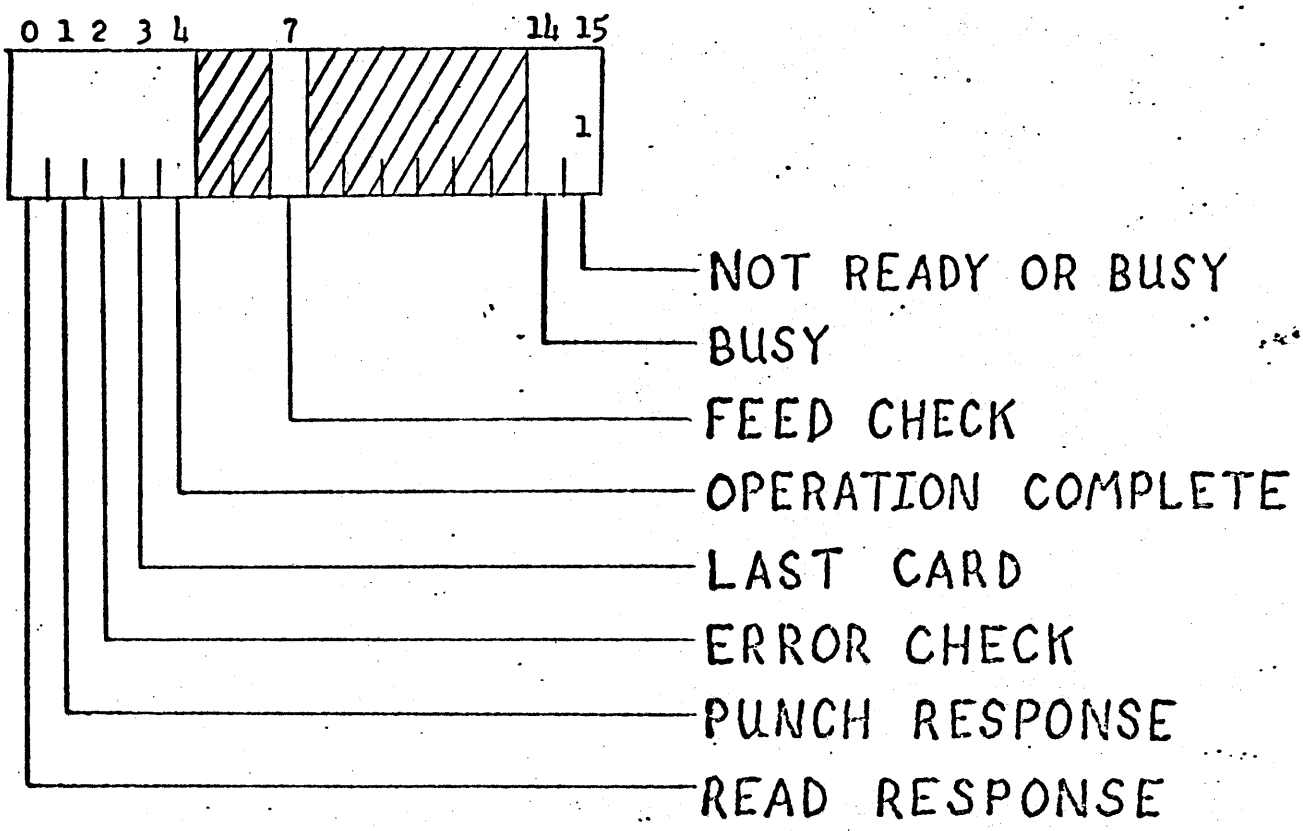
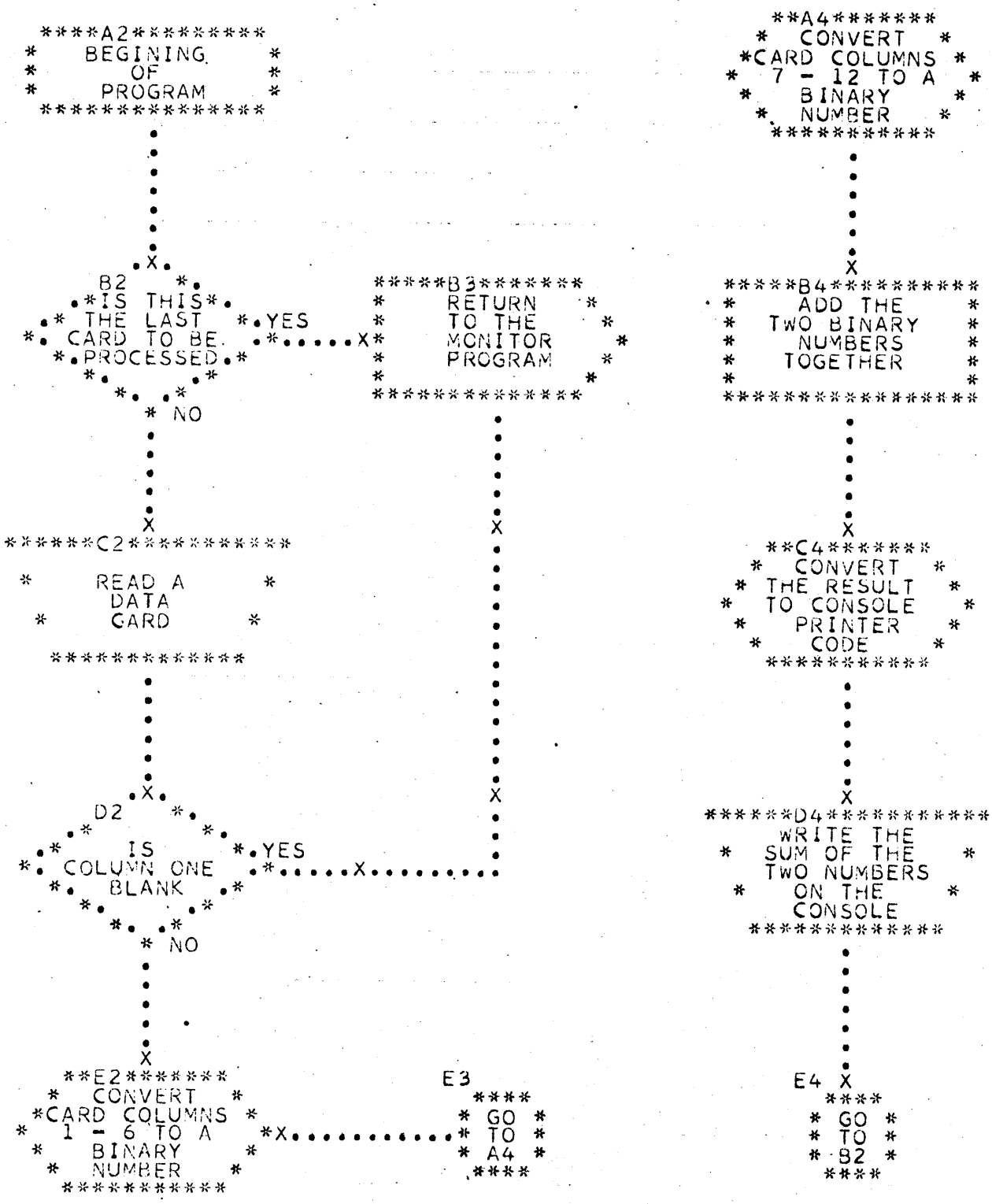


Fig. 17 & 28


A2
BEGINING OF PROGRAM
B2
IS THIS THE LAST CARD TO BE PROCESSED
YES
NO
B3
RETURN TO THE MONITOR PROGRAM
C2
READ A DATA CARD
D2
IS COLUMN ONE BLANK
YES
NO
E2
CONVERT CARD COLUMNS 1 - 6 TO A BINARY NUMBER
E3
GO TO A4
A4
CONVERT CARD COLUMNS 7 - 12 TO A BINARY NUMBER
B4
ADD THE TWO BINARY NUMBERS TOGETHER
C4
CONVERT THE RESULT TO CONSOLE PRINTER CODE
D4
WRITE THE SUM OF THE TWO NUMBERS ON THE CONSOLE
E4
GO TO B2

(Fig. 18)

Operation of the Console Printer under program control requires the data to be left justified, one character per word. This means that only Bits Zero through Seven will be transmitted to the Console Printer for printing. A level Four interrupt will occur each time the Console Printer has just completed printing the data as instructed by a "Write" command. After issuing the first print instruction, the output area must check to see if all the data has been printed on the Console Printer. If all the data has been printed, control is transferred to the beginning of the mainline program. If more data is to be printed, additional print XIØ instructions must be given.

The mainline program must load the address of the subprogram which will handle the level Four interrupts generated by the Console Printer into core location Twelve of the Interrupt Branch Address Table.

(Fig. 19)

A character can be printed by executing an XIØ instruction with an IØCC containing a Console Printer in the area code, and a "Write" in the function code. The modifier of the IØCC is not used. The first Sixteen Bits of the IØCC specify the address of the core location from which the first Eight Bits will be sent to the printer for printing control. The Console Printer will print one character for each XIØ instruction executed. The first Sixteen Bits of the IØCC specified by the XIØ contains the core address of that character to be printed. A level Four interrupt occurs after each character is printed. This level Four interrupt is serviced by a subprogram different than the one used to service the Card Reader level Four interrupt.

Fig. 18

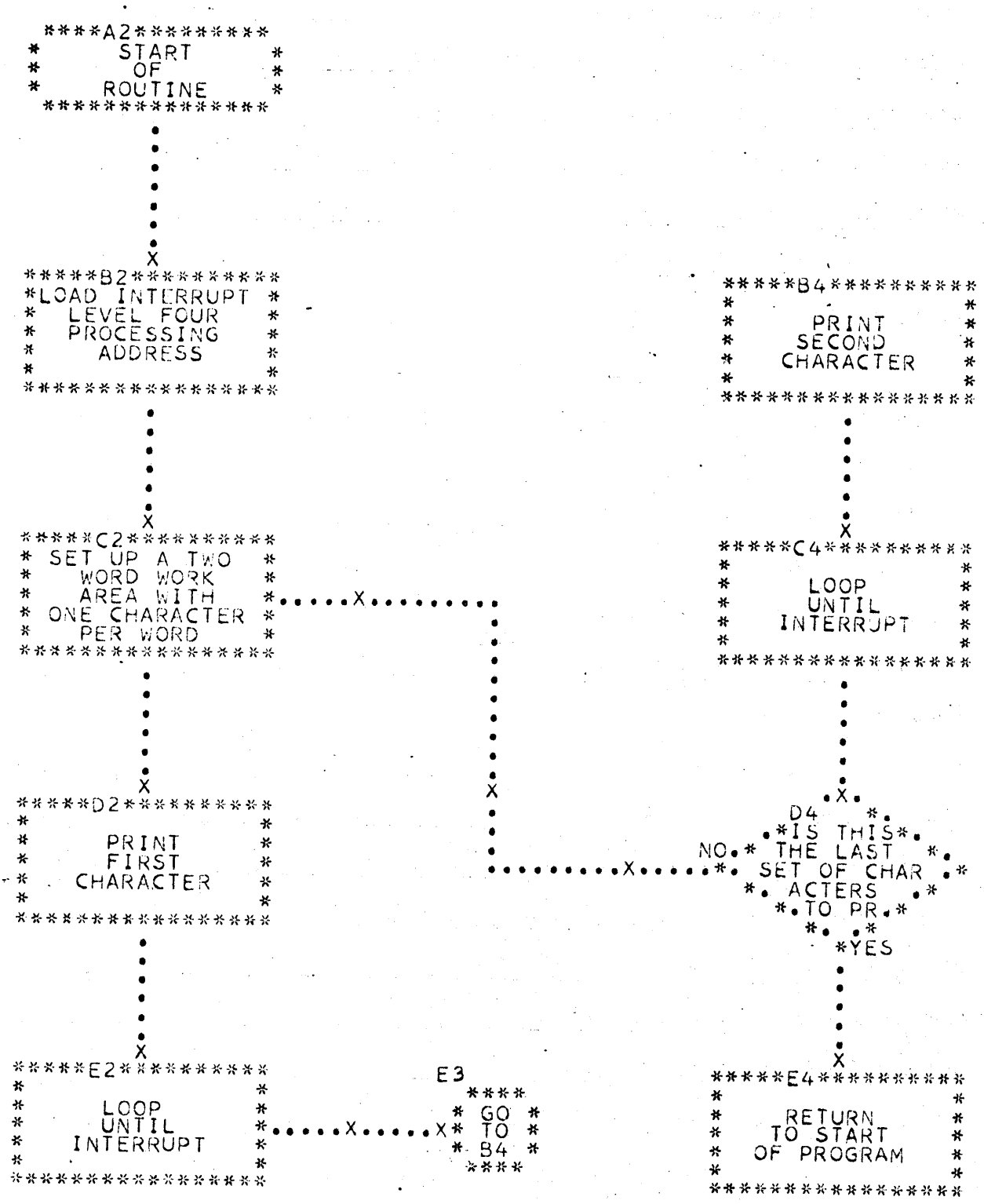

A2
START OF ROUTINE
B2
LOAD INTERRUPT LEVEL FOUR PROCESSING ADDRESS
C2
SET UP A TWO WORD WORK AREA WITH ONE CHARACTER PER WORD
D2
PRINT FIRST CHARACTER
E2
LOOP UNTIL INTERRUPT
E3
GO TO B4
B4
PRINT SECOND CHARACTER
C4
LOOP UNTIL INTERRUPT
D4
IS THIS THE LAST SET OF CHARACTERS TO PR
NO
YES
E4
RETURN TO START OF PROGRAM

Fig. 19

INSTRUCTION

LABEL	OP CODE	OPERAND	COMMENTS
	XIØ	CNSL1	PRINT CHARACTER ON CONSOLE

IØCC

CNSL1	BSS	E	O	
	DC		WORK	
	DC		/0900	(00001/001/00000000)

INTERPRETATION

AREA	=	00001	CONSOLE PRINTER
FUNC	=	001	WRITE
MODIFIER			NOT USED

(Fig. 20)

The subprogram that will service the level Four interrupt generated by the Console Printer is entered by a CPU generated level Four interrupt. The address of the subprogram is obtained from word Twelve in the Interrupt Branch Address Table. The address of this subprogram was stored in core location Twelve by the mainline program at the beginning of the console printing routine. The subprogram that will handle the level Four interrupts generated by the Console Printer will accomplish two functions. The first is to reset the hardware interrupt, and the second is to increment the return address by one. It will then return to the mainline processing program.

(Fig. 21)

The level Four hardware interrupt will be reset by the execution of an XIØ instruction with an IØCC specifying the Console Printer at the area code, a "Sense" in the function code, and Bit Fifteen on. Bit Fifteen will cause the hardware interrupt to be reset. After incrementing the return address by one, this program will return to the processing program.

Fig. 20

```
        X
 THE ADDRESS          *****A3**********
OF THIS WORD          *               *
IS STORED IN          *     ENTRY     *
CORE LOCATION         *     POINT     *
     12               *               *
                      *               *
                      ****************
                              .
                              .
                              .
                              .
                              .
                              .
                              X
                      *****B3**********
                      *               *
                      *     RESET     *
                      *   HARDWARE    *
                      *   INTERRUPT   *
                      *               *
                      ****************
                              .
                              .
                              .
                              .
                              .
                              .
                              X
                      *****C3**********
                      *               *
                      *   INCREMENT   *
                      *RETURN ADDRESS *
                      *    BY ONE     *
                      *               *
                      ****************
                              .
                              .
                              .
                              .
                              .
                              .
                              X
                      *****D3**********
                      * RETURN TO THE *
                      *  PROCESSING   *
                      *  PROGRAM AND  *
                      *RESET INTERRUPT*
                      *   INDICATOR   *
                      ****************
```

Fig. 21

INSTRUCTION

LABEL	OP CODE	OPERAND	COMMENTS
	XIØ	SENCL	WRITE CHARACTER ON CONSOLE

IØCC

	BSS	E 0	
SENCL	DC	/0000	NOT USED
	DC	/0F01	(00001/111/00000001)

INTERPRETATION

AREA	=	00001	CONSOLE PRINTER
FUNC	=	111	SENSE
MODIFIER	=	BIT 15	WILL RESET LEVEL FOUR INTERRUPT

(Fig. 22)

In summarizing the sequence of events that this application has taken, we have seen how two different devices cause an interrupt on level Four. The first was an "Operation Complete" on the Card Reader, and the second was an "Operation Complete" on the Console. In order to service both of these interrupts the program had to be written so that only one type of level Four interrupt could be generated . The reason for this is that the subprograms that handle the interrupts are to perform two distinctly separate functions. In the case of the "Operation Complete" for the Card Reader, the "last card" situation was examined and the core storage address that contained the card image had to be reset. Whereas, the subprogram for the Console Printer interrupt just had to reset the hardware interrupt.

(Fig. 23)

The problem of detecting what device initiated a given interrupt can be alleviated by the use of a single interrupt level subroutine.

Fig. 22

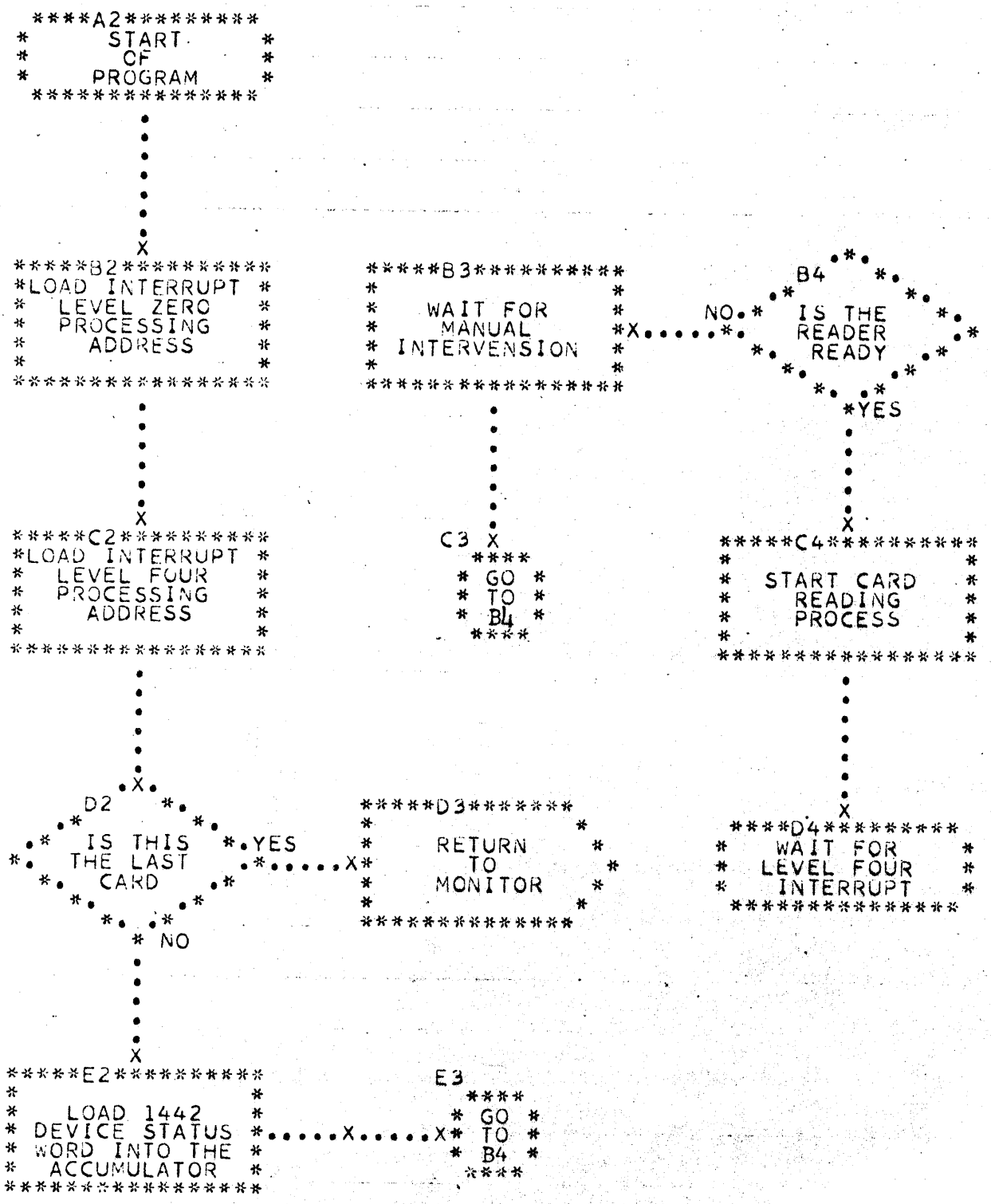
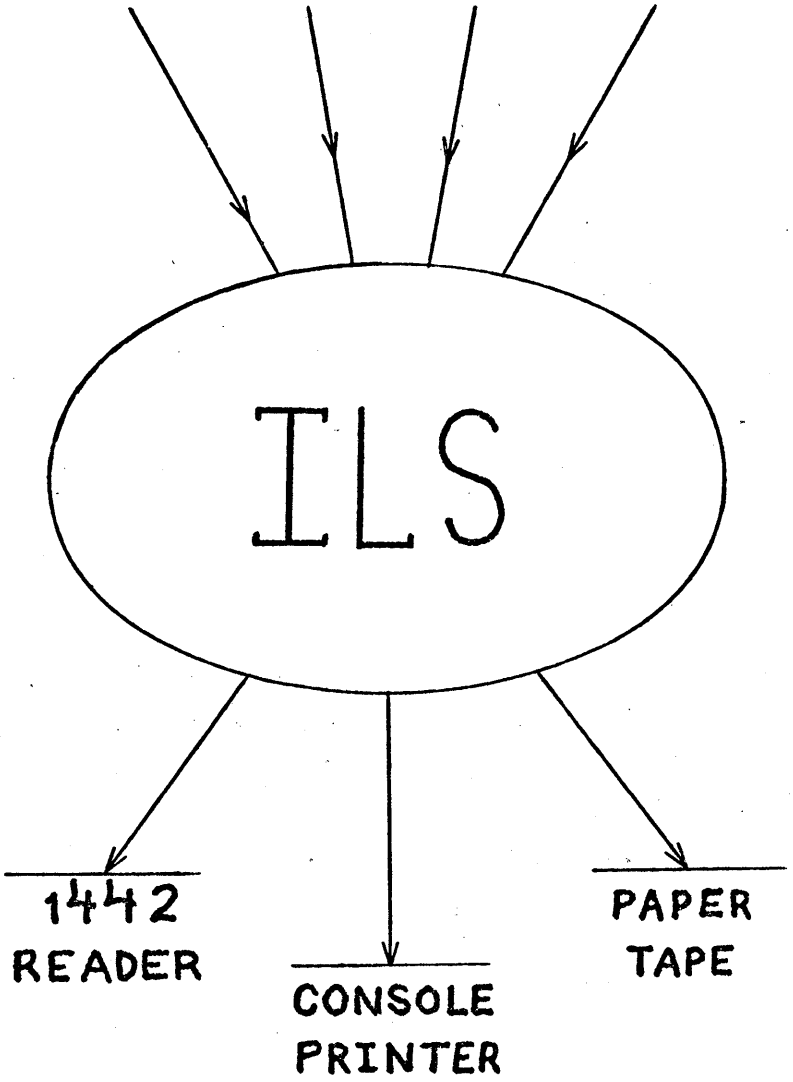

A2
START OF PROGRAM
B2
LOAD INTERRUPT LEVEL ZERO PROCESSING ADDRESS
B3
WAIT FOR MANUAL INTERVENSION
B4
IS THE READER READY
NO
YES
C2
LOAD INTERRUPT LEVEL FOUR PROCESSING ADDRESS
C3
GO TO B4
C4
START CARD READING PROCESS
D2
IS THIS THE LAST CARD
YES
NO
D3
RETURN TO MONITOR
D4
WAIT FOR LEVEL FOUR INTERRUPT
E2
LOAD 1442 DEVICE STATUS WORD INTO THE ACCUMULATOR
E3
GO TO B4

Fig. 23

INTERRUPTS (LEVEL 4)



(Fig. 24)

Instead of loading the individual interrupt subprogram addresses in the Interrupt Branch Address Table, a single subprogram can be created which will determine what device caused a given interrupt. Therefore, the address of this program is placed in the interrupt branch table and all interrupts of a given priority will branch to this subprogram.

(Fig. 25)

By using this technique we can develop one routine for all level Four interrupts. We will call this routine an Interrupt Level Subroutine.

(Fig. 26)

The function of this subroutine will be to interrogate the Interrupt Level Status Word. The Interrupt Level Status Word is similar to the Device Status Word in that it is loaded into the accumulator by the execution of an XIØ instruction. The IØCC will contain a "sense interrupt" in the function code. The area code and modifier sections are not used. It is then possible to interrogate the Interrupt Level Status Word to see which physical unit caused the level Four interrupt.

Fig. 2 & 24

INTERRUPT BRANCH ADDRESSES

CORE ADDRESS	CONTENTS
8	ADDRESS OF INTERRUPT LEVEL 0 SUBROUTINE
9	ADDRESS OF INTERRUPT LEVEL 1 SUBROUTINE
10	ADDRESS OF INTERRUPT LEVEL 2 SUBROUTINE
11	ADDRESS OF INTERRUPT LEVEL 3 SUBROUTINE
12	ADDRESS OF INTERRUPT LEVEL 4 SUBROUTINE
13	ADDRESS OF INTERRUPT LEVEL 5 SUBROUTINE

Fig. 25

DEVELOP

ONE

ROUTINE

FOR

ALL

LEVEL

FOUR

INTERRUPTS

Fig. 26

INTERRUPT STATUS WORD

INSTRUCTION

LABEL	OP CODE	OPERAND	COMMENTS
	XIØ	ISW	

IØCC

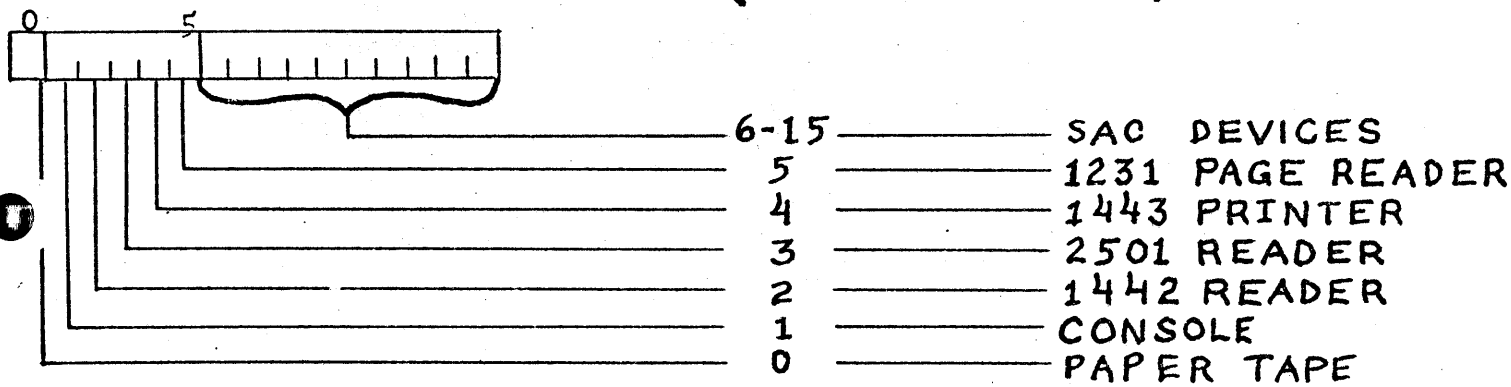
	BSS	E	0
ISW	DC	/0000	NOT USED
	DC	/0300	(00000/011/00000000)

INTERPRETATION

AREA	=	00000	NOT USED
FUNC	=	011	SENSE INTERRUPT
MODIFIER			NOT USED

LOAD INTERRUPT STATUS WORD INTO ACC

ILSW (LEVEL 4)



(Fig. 27)

Using the previous example of the Card Reader and Console Printer, if Bit One was on in the Interrupt Status Word, the Console Printer would have caused the interrupt. If Bit Two was on, the 1442 Card Reader would have caused the interrupt. It is then possible to branch to a given routine based on the Interrupt Status Word.

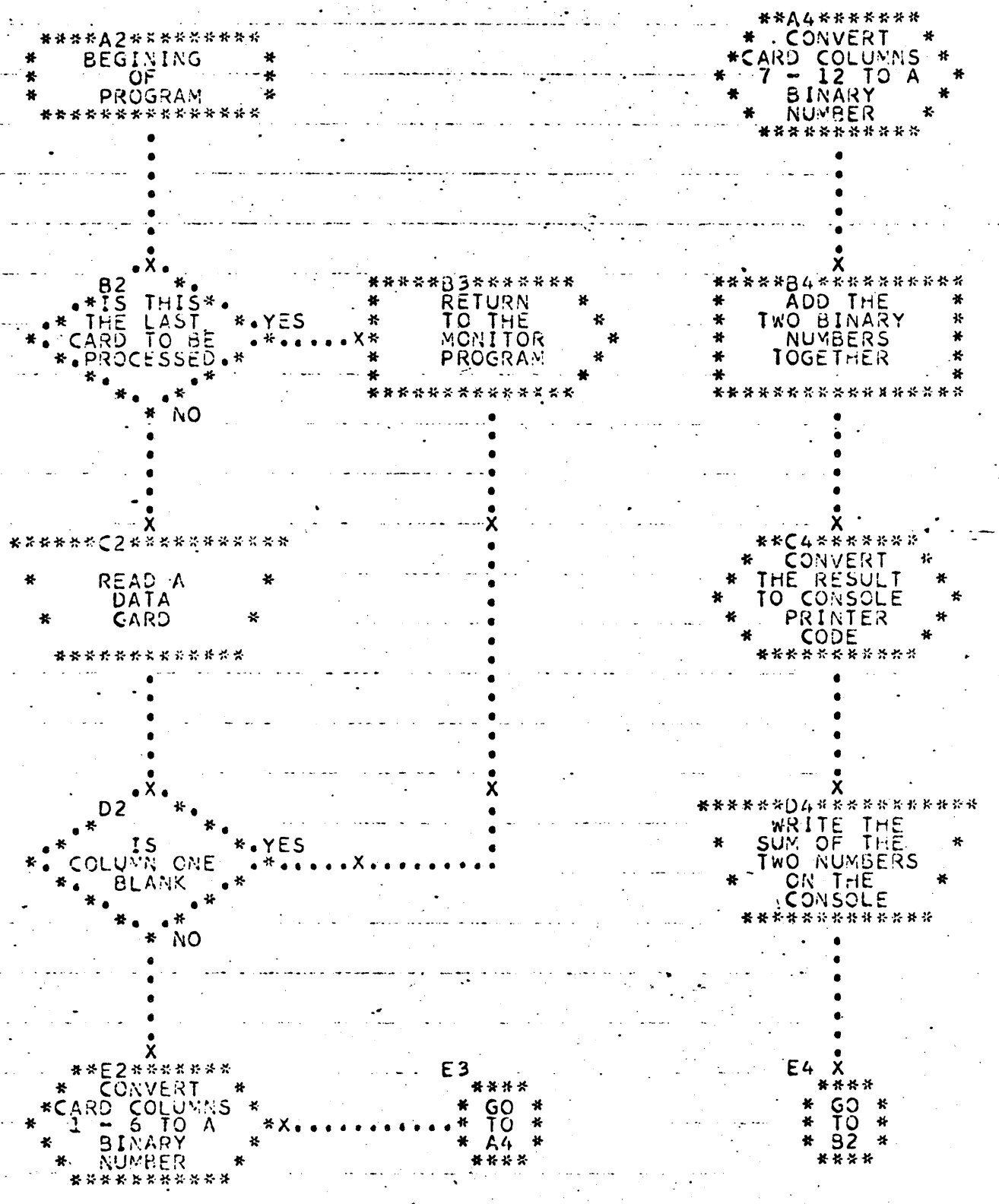
(Fig. 28)

By the creation of the Interrupt Level Four Subroutine it is possible to eliminate several programming steps from the mainline program. There is no need to consider loading the Interrupt Branch Address Table every time a new type of level Four interrupt is expected to occur. The Interrupt Branch Address Table is loaded once with the address of the Interrupt Level Subroutine that will service all level Four interrupts. Therefore, there is the possibility of writing just one subprogram to correctly handle all interrupts on a given priority level. By using the Interrupt Level Status Word it is possible to correctly acknowlege which device caused the interrupt and what programming considerations should be given in handling that device interrupt.

Fig. 27

```
                                  *****A3**********
                                  *               *
 X                                *     ENTRY     *
                                  *     POINT     *
THE ADDRESS                       *               *
OF THIS WORD                      *               *
IS STORED IN                      *               *
CORE LOCATION                     ****************
     12                                   .
                                          .
                                          .
                                          .
                                          .
                                          .
                                          X
                                  *****B3**********
                                  *LOAD THE INTER-*
                                  *  RUPT LEVEL   *
                                  *  STATUS WORD  *
                                  *   INTO THE    *
                                  *  ACCUMULATOR  *
                                  ****************
                                          .
                                          .
                                          .
                                          .
                                          .
                                          .
                                          X
                                     C3   *.
                                   .*     *.
                                 .*   IS    *. NO      *****C4**********
                                *.  BIT ONE  .*.....X*  *              *
                                 *.   ON    .*         *    SERVICE   *
                                   *.     .*           *  CARD READER *
                                     *. .*             *   INTERRUPT  *
                                       *YES            *              *
                                                       ****************
                                          .                   .
                                          .                   .
                                          .                   .
                                          .                   .
                                          .                   .
                                          .                   .
                                          X                   X
                                  *****D3**********   *****D4**********
                                  *               *   * RETURN TO THE *
                                  *    SERVICE    *   *  PROCESSING   *
                                  *    CONSOLE    *.....X*  PROGRAM AND *
                                  *    PRINTER    *   *RESET INTERRUPT*
                                  *  INTERRUPT    *   *   INDICATOR   *
                                  ****************   ****************
```

Fig. 17 & 28


A2
BEGINING OF PROGRAM
B2
IS THIS THE LAST CARD TO BE PROCESSED
YES
NO
B3
RETURN TO THE MONITOR PROGRAM
A4
CONVERT CARD COLUMNS 7 - 12 TO A BINARY NUMBER
B4
ADD THE TWO BINARY NUMBERS TOGETHER
C2
READ A DATA CARD
C4
CONVERT THE RESULT TO CONSOLE PRINTER CODE
D2
IS COLUMN ONE BLANK
YES
NO
D4
WRITE THE SUM OF THE TWO NUMBERS ON THE CONSOLE
E2
CONVERT CARD COLUMNS 1 - 6 TO A BINARY NUMBER
E3
GO TO A4
E4
GO TO B2

In summary, the 1130 BAL programmer has available to him one XIØ instruction which is used to perform all input or output data manipulation functions. Modifying the XIØ instruction is an Input/Output Control Command (IØCC) which provides additional information as to what device is being acted upon and what action is being taken by that device.

There is also available a Device Status Word which the programmer may interrogate to find the status of any Input/Output Device.

In addition, there is available an Interrupt Level Status Word which allows the programmer to interrogate all Input/Output Devices on a given interrupt level and determine what sequence of events should accompany a given interrupt. By using these things the programmer is able to place information into or extract information out of the CPU.

SAMPLE PROGRAM I

Each interrupt service subprogram is independent. The Interrupt Branch Address Table is changed to reflect an expected interrupt.

```
                         *          SET UP REQUIRED INTERRUPT BRANCH
                         *                ADDRESSES. START EXECUTION
                         *
0000 01 65000018         LOAD  LDX  L1 LEV0  LOAD INTERRUPT LEVEL ZERO
0002 00 6D000008               STX  L1 8      PROCESSING ADDRESS INTO WORD 8
0004 01 65000024         BACK  LDX  L1 LEV4  LOAD INTERRUPT LEVEL FOUR
0006 00 6D000000C              STX  L1 12     PROCESSING ADDRESS INTO WD 12
0008 0  C067                   LD     EOJSW CHECK FOR LAST CARD
0009 01 4C200062               BSC  L EOJLC,Z    END OF JOB IF LAST CARD
000B 0  080A             CHK   XIO     SENS  LOAD DEVICE STATUS WORD
000C 01 4C04000F               BSC  L *+1,E BRANCH IF UNIT NOT READY
000E 0  7002                   MDX    *+2   SKIP OVER ERROR ROUTINE IF OK
000F 0  3000                   WAIT         WAIT FOR MANUAL INTERVENSION
0010 0  70FA                   MDX    CHK   RE-CHECK STATUS TO BE SURE
0011 0  0802                   XIO    START START READING A CARD
0012 0  70FF                   MDX    *-1   LOOP UNTIL INTERRUPT ON LEVEL 4
                         *                     THIS WILL SIGNAL END OF CARD
                         *
                         *          INPUT/OUTPUT CONTROL COMMAND FOR THE
                         *                START READING A CARD
                         *                INSTRUCTION
                         *
0014    0000                   BSS  E 0      I/O CTL COMMAND MUST BE EVEN WD
0014 0  0000             START DC      /0000 NOT USED
0015 0  1404                   DC     /1404 AREA=00010,FUNC=100,MOD=BIT 13
                         *
                         *          AREA = 00010 WHICH IS 1442 CARD READ
                         *                       PUNCH UNIT
                         *          FUNC = 100   WHICH CAUSES THE CARD
                         *                       READ-PUNCH TO ACCOMPLISH
                         *                       THE FUNCTION SPECIFIED
                         *                       BY THE MODIFIER.
                         *          MODIF= BIT 13 ON. THIS CAUSES THE
                         *                       CARD TO MOVE THROUGH THE
                         *                       READ STATION. AS EACH
                         *                       COLUMN IS READ AND
                         *                       CHECKED, THE CARD READ
                         *                       PUNCH INITIATES A READ
                         *                       COLUMN RESPONSE
                         *                       INTERRUPT (LEVEL 0)
0016 0  0000             SENS  DC      /0000 NOT USED
0017 0  1700                   DC     /1700 AREA=00010,FUNC=111,MOD=NONE
                         *
                         *          AREA = 00010 WHICH IS THE CARD READ
                         *                       PUNCH UNIT
                         *          FUNC = 111   WHICH DIRECTS THE CARD
                         *                       READ-PUNCH TO PLACE ITS
                         *                       DEVICE STATUS WORD (DSW)
                         *                       INTO CPU ACC
                         *          MODIF = NONE
                         *
```

```
                        *              INTERRUPT LEVEL ZERO PROCESSING
                        *
0018 0  0000            LEVO   DC      *-*    SAVE RETURN ADDRESS
0019 0  0806                   XIO     SENSO  RESET HARDWARE INTERRUPT LEVEL
001A 0  0807                   XIO     READ   TRANSFER CARD COLUMN TO CORE
                        *                     STORAGE
001B 01 74010022               MDX   L READ,+1 BUMP CORE STORAGE ADDRESS
001D 01 4CC00018               BOSC  I LEVO   RETURN TO PROGRAM. TURN
                        *                     INTERRUPT LEVEL INDICATOR OFF.
                        *
0020    0000                   BSS   E 0      I/O CTL COMMAND MUST BE EVEN WD
0020 0  0000            SENSO  DC      /0000  NOT USED
0021 0  1701                   DC      /1701  AREA=00010,FUNC=111,MOD=BIT 15
                        *
                        *              AREA = 00010 WHICH IS THE CARD READ
                        *                           PUNCH UNIT
                        *              FUNC = 111   WHICH DIRECTS THE CARD
                        *                           READ-PUNCH TO PLACE ITS
                        *                           DEVICE STATUS WORD (DSW)
                        *                           INTO CPU ACC.
                        *              MOD BIT 15   MODIFIER BIT 15 RESETS
                        *                           HARDWARE INTERRUPT LEVEL
                        *                           ZERO
                        *
0022 1  0072            READ   DC      CRDIN  ADDRESS TO PLACE CARD COLUMN
0023 0  1200                   DC      /1200  AREA=00010,FUNC=010,MOD= NONE
                        *
                        *              AREA = 00010 WHICH IS THE 1442 CARD
                        *                           READ-PUNCH UNIT
                        *              FUNC = 010   WHICH CAUSES THE CARD
                        *                           IMAGE TO BE ENTERED FROM
                        *                           THE CARD READ-PUNCH UNIT
                        *                           INTO THE CORE STORAGE
                        *                           LOCATION SPECIFIED BY
                        *                           THE ADDRESS
                        *              MODIF IS NOT USED
```

```
                        *              INTERRUPT LEVEL FOUR PROCESSING
                        *
0024 0   0000           LEV4   DC      *-*    SAVE RETURN ADDRESS
0025 0   080A                  XIO     SENS4  RESET HARDWARE INTERRUPT LEVEL
                        *                            AND LOAD DSW
0026 0   E04A                  AND     EOJMK  CHECK BIT 3 OF DSW
0027 0   4820                  BSC     Z      SKIP IF NOT ON (NOT LAST CARD)
0028 0   D047                  STO     EOJSW  SET SWITCH TO NOT ZERO
0029 01  65000072              LDX  L1 CRDIN  RESTORE THE ADDRESS OF THE CARD
002B 01  6D000022              STX  L1 READ    INPUT AREA IN READ IOCC
002D 01  4C400032              BOSC L  LCTST  RETURN TO PROGRAM. TURN
                        *                     INTERRUPT LEVEL INDICATOR OFF.
                        *
0030     0000                  BSS  E  0      I/O CTL COMMAND MUST BE EVEN WD
0030 0   0000           SENS4  DC      /0000  NOT USED
0031 0   1702                  DC      /1702  AREA=00010,FUNC=111,MOD= BIT 14
                        *
                        *              AREA = 00010  WHICH IS THE CARD READ
                        *                            PUNCH UNIT
                        *              FUNC = 111    WHICH DIRECTS THE CARD
                        *                            READ-PUNCH TO PLACE ITS
                        *                            DEVICE STATUS WORD (DSW)
                        *                            INTO CPU ACC
                        *              MOD BIT 14    MODIFIER BIT 14 RESETS
                        *                            HARDWARE INTERRUPT LEVEL
                        *                            FOUR
```

```
0032 0  C03F        LCTST LD      CRDIN LOAD CARD COLUMN ONE
0033 01 4C180066          BSC  L  EOJ,-+ GO TO EXIT IF BLANK
0035 20 040C2255          LIBF    DCBIN CONVERT DECIMAL INTO BINARY
0036 1  0072              DC      CRDIN
0037 0  D037              STO     QTY1 STORE NUMBER
0038 20 040C2255          LIBF    DCBIN CONVERT DECIMAL INTO BINARY
0039 1  0078              DC      CRDIN+6
003A 0  8034              A       QTY1  ADD TWO NUMBERS TOGETHER
003B 20 02255103          LIBF    BINDC CONVERT BINARY TO DECIMAL
003C 1  007E              DC      CRDIN+12
003D 20 08593509          LIBF    HOLPR CONVERT FROM DECIMAL TO CONSOLE
003E 0  0000              DC      /0000
003F 1  007E              DC      CRDIN+12    INPUT AREA
0040 1  006A              DC      CON1+3      OUTPUT AREA
0041 0  0006              DC      6           NUMBER OF CHARACTERS
```

```
                        *              SET UP REQUIRED INTERRUPT BRANCH
                        *                    ADDRESSES
                        *
0042 01 6500005A              LDX  L1 LEV4P LOAD INTERRUPT LEVEL 4 ADDRESS
0044 00 6D000000C             STX  L1 12          FOR CONSOLE PRINTER
0046 0  61FA                  LDX   1 -6
0047 01 C5000060        LOOP  LD   L1 CON1+6      LOAD FIRST TWO CHARACTERS
0049 0  1888                  SRT      8      PLACE SECOND CHARACTER IN EXT
004A 0  1008                  SLA      8      LEFT JUSTIFY FIRST CHARACTER
004B 0  0021                  STO      WORK
004C 0  1090                  SLT      16     LEFT JUSTIFY SECOND CHARACTER
004D 0  0020                  STO      WORK+1
004E 0  0807                  XIO      CNSL1 PRINT FIRST CHARACTER
004F 0  70FF                  MDX  X   -1     WAIT FOR INTERRUPT
0050 0  0807                  XIO      CNSL2 PRINT SECOND CHARACTER
0051 0  70FF                  MDX  X   -1     WAIT FOR INTERRUPT
0052 0  7101                  MDX   1  +1     BUMP POINTER BY ONE
0053 0  70F3                  MDX      LOOP   CONTINUE CYCLE
0054 01 4C000004              BSC  L   BACK   BRANCH TO BEGINING
                        *
                        *
0056    0000                  BSS  E   0      I/O CTL COMMAND MUST BE EVEN WD
0056 1  006D            CNSL1 DC       WORK   ADDRESS OF FIRST CHARACTER
0057 0  0900                  DC       /0900 AREA=00001,FUNC=001,MOD=NONE
                        *
                        *              AREA = 00001 WHICH IS THE CONSOLE
                        *                           PRINTER
                        *
                        *              FUNC = 001   WHICH CAUSES THE WORD AT
                        *                           THE CORE STORAGE
                        *                           LOCATION SPECIFIED BY
                        *                           THE ADDRESS TO BE SENT
                        *                           TO THE CONSOLE-PRINTER
                        *                           FOR PRINTING OR CONTROL
                        *
                        *              MODIF NOT USED
                        *
0058 1  006F            CNSL2 DC       WORK+1 ADDRES OF SECOND CHARACTER
0059 0  0900                  DC       /0900 FUNCTION IS IDENTICAL TO CNSL1
```

```
                        *              INTERRUPT LEVEL FOUR PROCESSING
                        *
005A 0   0000           LEV4P DC       *-*    SAVE RETURN ADDRESS
005B 0   0804                 XIO      SENCL LOAD DSW AND RESET INTERRUPT
005C 01  7401005A             MDX   L  LEV4P,+1    BUMP ADDRESS TO NEXT WORD
005E 01  4CC0005A             BOSC  I  LEV4P RETURN TO PROGRAM TURN
                        *                     INTERRUPT LEVEL INDICATOR OFF
                        *
0060     0000                 BSS   E  0      I/O CTL COMMAND MUST BE EVEN WD
0060 0   0000           SENCL DC       /0000 NOT USED
0061 0   0F01                 DC       /0F01 AREA=00001,FUNC=111,MOD=BIT 15
                        *
                        *              AREA = 00001 WHICH IS THE CONSOLE
                        *                           PRINTER
                        *              FUNC = 111   WHICH CAUSES THE DEVICE
                        *                           STATUS WORD OF THE
                        *                           CONSOLE-PRINTER TO BE
                        *                           PLACED IN THE ACC
                        *              MOD BIT 15   MODIFIER BIT 15
                        *                           SPECIFIES THAT THE
                        *                           RESPONSES ARE TO BE
                        *                           RESET
                        *
```

```
                        *              CLEAN-UP AND HOUSEKEEPING
                        *
0062 0  0801            EOJLC XIO      FEED  PASS THE LAST CARD OUT
0063 0  7002                  MDX      EOJ   GO TO END OF JOB ROUTINE
                        *
                        *              I/O CTL COMMAND FOR FEED CYCLE
                        *
0064    0000                  BSS  E   0     I/O CTL COMMAND MUST BE EVEN WD
0064 0  0000            FEED  DC       /0000 NOT USED
0065 0  1402                  DC       /1402 AREA=00010,FUNC=100,MOD=BIT 14
                        *
                        *              AREA = 00010 WHICH IS THE CARD READ
                        *                           PUNCH UNIT
                        *              FUNC = 100   WHICH CAUSES THE CARD
                        *                           READ-PUNCH UNIT TO
                        *                           ACCOMPLISH THE FUNCTION
                        *                           SPECIFIED BY THE MODIFIE
                        *              MODIF= BIT 14 FEED CYCLE. ADVANCE
                        *                           ALL CARDS BY ONE STATION
                        *
0066 0  6038            EOJ   EXIT           RETURN TO THE MONITOR
                        *
0067 0  8103            CON1  DC       /8103 CARRIER RETURN AND LINE FEED
0068 0  98B2                  DC       /98B2 CHARACTERS 'S' AND 'U'
0069 0  72C2                  DC       /72C2 CHARACTERS 'M' AND '='
006A    0003                  BSS      3     STORAGE FOR ANSWER
006D    0002            WORK  BSS      2     WORK AREA FOR CONSOLE PRINTER
                        *
006F 0  0000            QTY1  DC       0     STORAGE FOR RESULTS IN BINARY
                        *
0070 0  0000            EOJSW DC       0     END OF JOB SWITCH
0071 0  1000            EOJMK DC       /1000 END OF JOB MASK (LAST CARD)
                        *
0072    0050            CRDIN BSS      80
                        *
0002    0000                  END      LOAD
```

SYMBOL TABLE

BACK	0004	CHK	000B	CNSL1	0056	CNSL2	0058	CON1	0067
CRDIN	0072	EOJ	0066	EOJLC	0062	EOJMK	0071	EOJSW	0070
FEED	0064	LCTST	0032	LEV0	0018	LEV4	0024	LEV4P	005A
LOAD	0000	LOOP	0047	QTY1	006F	READ	0022	SENCL	0060
SENS	0016	SENS0	0020	SENS4	0030	START	0014	WORK	006D

NO ERRORS IN ABOVE ASSEMBLY.

SAMPLE PROGRAM II

Interrupt Level Status Word is checked to find the device that caused the interrupt.

```
                        *                SET UP REQUIRED INTERRUPT BRANCH
                        *                      ADDRESSES. START EXECUTION
                        *
0000 01 65000018        LOAD  LDX  L1 LEV0  LOAD INTERRUPT LEVEL ZERO
0002 00 6D000008              STX  L1 8      PROCESSING ADDRESS INTO WORD 8
0004 01 65000024        BACK  LDX  L1 LEV4  LOAD INTERRUPT LEVEL FOUR
0006 00 6D000000C             STX  L1 12     PROCESSING ADDRESS INTO WD 12
0008 0  C067                  LD      EOJSW CHECK FOR LAST CARD
0009 01 4C200062              BSC  L  EOJLC,Z   END OF JOB IF LAST CARD
000B 0  080A            CHK   XIO     SENS  LOAD DEVICE STATUS WORD
000C 01 4C04000F              BSC  L  *+1,E BRANCH IF UNIT NOT READY
000E 0  7002                  MDX     *+2   SKIP OVER ERROR ROUTINE IF OK
000F 0  3000                  WAIT          WAIT FOR MANUAL INTERVENSION
0010 0  70FA                  MDX     CHK   RE-CHECK STATUS TO BE SURE
0011 0  0802                  XIO     START START READING A CARD
0012 0  70FF                  MDX     *-1   LOOP UNTIL INTERRUPT ON LEVEL 4
                        *                      THIS WILL SIGNAL END OF CARD
                        *
                        *                INPUT/OUTPUT CONTROL COMMAND FOR THE
                        *                      START READING A CARD
                        *                      INSTRUCTION
                        *
0014    0000                  BSS  E  0      I/O CTL COMMAND MUST BE EVEN WD
0014 0  0000            START DC      /0000 NOT USED
0015 0  1404                  DC      /1404 AREA=00010,FUNC=100,MOD=BIT 13
                        *
                        *                AREA = 00010 WHICH IS 1442 CARD READ
                        *                             PUNCH UNIT
                        *                FUNC = 100   WHICH CAUSES THE CARD
                        *                             READ-PUNCH TO ACCOMPLISH
                        *                             THE FUNCTION SPECIFIED
                        *                             BY THE MODIFIER.
                        *                MODIF= BIT 13 ON. THIS CAUSES THE
                        *                             CARD TO MOVE THROUGH THE
                        *                             READ STATION. AS EACH
                        *                             COLUMN IS READ AND
                        *                             CHECKED, THE CARD READ
                        *                             PUNCH INITIATES A READ
                        *                             COLUMN RESPONSE
                        *                             INTERRUPT (LEVEL 0)
0016 0  0000            SENS  DC      /0000 NOT USED
0017 0  1700                  DC      /1700 AREA=00010,FUNC=111,MOD=NONE
                        *
                        *                AREA = 00010 WHICH IS THE CARD READ
                        *                             PUNCH UNIT
                        *                FUNC = 111   WHICH DIRECTS THE CARD
                        *                             READ-PUNCH TO PLACE ITS
                        *                             DEVICE STATUS WORD (DSW)
                        *                             INTO CPU ACC
                        *                MODIF = NONE
                        *
```

```
                        *              INTERRUPT LEVEL ZERO PROCESSING
                        *
0018 0  0000            LEVO  DC       *-*   SAVE RETURN ADDRESS
0019 0  0806                  XIO      SENSO RESET HARDWARE INTERRUPT LEVEL
001A 0  0807                  XIO      READ  TRANSFER CARD COLUMN TO CORE
                        *                      STORAGE
001B 01 74010022              MDX  L   READ,+1 BUMP CORE STORAGE ADDRESS
001D 01 4CC00018              BOSC I   LEVO  RETURN TO PROGRAM. TURN
                        *                      INTERRUPT LEVEL INDICATOR OFF.
                        *
0020    0000            .     BSS  E   0     I/O CTL COMMAND MUST BE EVEN WD
0020 0  0000            SENSO DC       /0000 NOT USED
0021 0  1701                  DC       /1701 AREA=00010,FUNC=111,MOD=BIT 15
                        *
                        *              AREA = 00010 WHICH IS THE CARD READ
                        *                           PUNCH UNIT
                        *              FUNC = 111   WHICH DIRECTS THE CARD
                        *                           READ-PUNCH TO PLACE ITS
                        *                           DEVICE STATUS WORD (DSW)
                        *                           INTO CPU ACC.
                        *              MOD BIT 15   MODIFIER BIT 15 RESETS
                        *                           HARDWARE INTERRUPT LEVEL
                        *                           ZERO
                        *
0022 1  0072            READ  DC       CRDIN ADDRESS TO PLACE CARD COLUMN
0023 0  1200                  DC       /1200 AREA=00010,FUNC=010,MOD= NONE
                        *
                        *              AREA = 00010 WHICH IS THE 1442 CARD
                        *                           READ-PUNCH UNIT
                        *              FUNC = 010   WHICH CAUSES THE CARD
                        *                           IMAGE TO BE ENTERED FROM
                        *                           THE CARD READ-PUNCH UNIT
                        *                           INTO THE CORE STORAGE
                        *                           LOCATION SPECIFIED BY
                        *                           THE ADDRESS
                        *              MODIF IS NOT USED
```

```
                        *                INTERRUPT LEVEL FOUR PROCESSING
                        *
0024 0  0000            LEV4  DC         *-*   SAVE RETURN ADDRESS
0025 0  0814                  XIO        ILSW4 LOAD INTERRUPT STATUS IN ACC
0026 0  1001                  SLA        1     SHIFT LEFT ONE BIT TO CHECK
0027 01 4C280033              BSC  L     CONSL,+Z   CONSOLE BIT.
0029 0  0812                  XIO        SENS4 RESET HARDWARE INTERRUPT LEVEL
                        *                          AND LOAD DSW
002A 0  E046                  AND        EOJMK CHECK BIT 3 OF DSW
002B 0  4820                  BSC        Z     SKIP IF NOT ON (NOT LAST CARD)
002C 0  0043                  STO        EOJSW SET SWITCH TO NOT ZERO
002D 01 65000072              LDX  L1    CRDIN RESTORE THE ADDRESS OF THE CARD
002F 01 6D000022              STX  L1    READ   INPUT AREA IN READ IOCC
0031 01 4C40003E              BOSC L     LCTST RETURN TO PROGRAM. TURN
                        *                      INTERRUPT LEVEL INDICATOR OFF.
                        *
                        *                INTERRUPT LEVEL FOUR PROCESSING
                        *                 (CONSOLE PRINTER)
                        *
0033 0  0804            CONSL XIO        SENCL LOAD DSW AND RESET INTERRUPT
0034 01 74010024              MDX  L     LEV4,+1     BUMP ADDRESS TO NEXT WORD
0036 01 4CC00024              BOSC I     LEV4  RETURN TO PROGRAM TURN
                        *                      INTERRUPT LEVEL INDICATOR OFF
                        *
0038    0000                  BSS  E     0     I/O CTL COMMAND MUST BE EVEN WD
0038 0  0000            SENCL DC         /0000 NOT USED
0039 0  0F01                  DC         /0F01 AREA=00001,FUNC=111,MOD=BIT 15
                        *
                        *                AREA = 00001 WHICH IS THE CONSOLE
                        *                             PRINTER
                        *                FUNC = 111   WHICH CAUSES THE DEVICE
                        *                             STATUS WORD OF THE
                        *                             CONSOLE-PRINTER TO BE
                        *                             PLACED IN THE ACC
                        *                MOD BIT 15   MODIFIER BIT 15
                        *                             SPECIFIES THAT THE
                        *                             RESPONSES ARE TO BE
                        *                             RESET
                        *
003A    0000                  BSS  E     0     I/O CTL COMMAND MUST BE EVEN WD
003A 0  0000            ILSW4 DC         /0000 NOT USED
003B 0  0300                  DC         /0300 AREA=0,FUNC=011,MOD2 NONE
                        *
                        *                AREA IS NOT USED
                        *                FUNC = 011 SENSE INTERRUPT STATUS
                        *                MOD IS NOT USED
                        *
003C 0  0000            SENS4 DC         /0000 NOT USED
003D 0  1702                  DC         /1702 AREA=00010,FUNC=111,MOD= BIT 14
                        *
                        *                AREA = 00010 WHICH IS THE CARD READ
                        *                             PUNCH UNIT
                        *                FUNC = 111   WHICH DIRECTS THE CARD
                        *                             READ-PUNCH TO PLACE ITS
                        *                             DEVICE STATUS WORD (DSW)
                        *                             INTO CPU ACC
                        *                MOD BIT 14   MODIFIER BIT 14 RESETS
                        *                             HARDWARE INTERRUPT LEVEL
```

* FOUR

```
003E 0  C033          LCTST LD      CRDIN LOAD CARD COLUMN ONE
 03F 01 4C180066            BSC  L   EOJ,-+ GO TO EXIT IF BLANK
0041 20 040C2255            LIBF     DCBIN CONVERT DECIMAL INTO BINARY
0042 1  0072                DC       CRDIN
0043 0  D028                STO      QTY1 STORE NUMBER
0044 20 040C2255            LIBF     DCBIN CONVERT DECIMAL INTO BINARY
0045 1  0078                DC       CRDIN+6
0046 0  8028                A        QTY1  ADD TWO NUMBERS TOGETHER
0047 20 02255103            LIBF     BINDC CONVERT BINARY TO DECIMAL
0048 1  007E                DC       CRDIN+12
0049 20 085935D9            LIBF     HOLPR CONVERT FROM DECIMAL TO CONSOLE
004A 0  0000                DC       /0000
004B 1  007E                DC       CRDIN+12    INPUT AREA
004C 1  006A                DC       CON1+3      OUTPUT AREA
004D 0  0006                DC       6           NUMBER OF CHARACTERS
```

```
004E 0  61FA                LDX   1 -6
004F 01 C500006D     LOOP   LD    L1 CON1+6      LOAD FIRST TWO CHARACTERS
0051 0  1888                SRT      8      PLACE SECOND CHARACTER IN EXT
0052 0  1008                SLA      8      LEFT JUSTIFY FIRST CHARACTER
0053 0  D019                STO      WORK
0054 0  1090                SLT      16     LEFT JUSTIFY SECOND CHARACTER
0055 0  D018                STO      WORK+1
0056 0  0807                XIO      CNSL1  PRINT FIRST CHARACTER
0057 0  70FF                MDX   X  -1     WAIT FOR INTERRUPT
0058 0  0807                XIO      CNSL2  PRINT SECOND CHARACTER
0059 0  70FF                MDX   X  -1     WAIT FOR INTERRUPT
005A 0  7101                MDX    1 +1     BUMP POINTER BY ONE
005B 0  70F3                MDX      LOOP   CONTINUE CYCLE
005C 01 4C000004            BSC   L  BACK   BRANCH TO BEGINING
                     *
                     *
005E    0000                BSS   E  0      I/O CTL COMMAND MUST BE EVEN WD
005E 1  006D         CNSL1  DC       WORK   ADDRESS OF FIRST CHARACTER
005F 0  0900                DC       /0900  AREA=00001,FUNC=001,MOD=NONE
                     *
                     *                AREA = 00001 WHICH IS THE CONSOLE
                     *                             PRINTER
                     *                FUNC = 001   WHICH CAUSES THE WORD AT
                     *                             THE CORE STORAGE
                     *                             LOCATION SPECIFIED BY
                     *                             THE ADDRESS TO BE SENT
                     *                             TO THE CONSOLE-PRINTER
                     *                             FOR PRINTING OR CONTROL
                     *                MODIF NOT USED
                     *
0060 1  006F         CNSL2  DC       WORK+1 ADDRES OF SECOND CHARACTER
0061 0  0900                DC       /0900  FUNCTION IS IDENTICAL TO CNSL1
```

```
                        *                CLEAN-UP AND HOUSEKEEPING
                        *
0062 0  0801            EOJLC XIO        FEED  PASS THE LAST CARD OUT
0063 0  7002                  MDX        EOJ   GO TO END OF JOB ROUTINE
                        *
                        *                I/O CTL COMMAND FOR FEED CYCLE
                        *
0064    0000                  BSS  E     0     I/O CTL COMMAND MUST BE EVEN WD
0064 0  0000            FEED  DC         /0000 NOT USED
0065 0  1402                  DC         /1402 AREA=00010,FUNC=100,MOD=BIT 14
                        *
                        *                AREA = 00010 WHICH IS THE CARD READ
                        *                             PUNCH UNIT
                        *                FUNC = 100   WHICH CAUSES THE CARD
                        *                             READ-PUNCH UNIT TO
                        *                             ACCOMPLISH THE FUNCTION
                        *                             SPECIFIED BY THE MODIFIE
                        *                MODIF= BIT 14 FEED CYCLE. ADVANCE
                        *                             ALL CARDS BY ONE STATION
                        *
0066 0  6038            EOJ   EXIT             RETURN TO THE MONITOR
                        *
0067 0  8103            CON1  DC         /8103 CARRIER RETURN AND LINE FEED
0068 0  98B2                  DC         /98B2 CHARACTERS 'S' AND 'U'
0069 0  72C2                  DC         /72C2 CHARACTERS 'M' AND '='
006A    0003                  BSS        3     STORAGE FOR ANSWER
006D    0002            WORK  BSS        2     WORK AREA FOR CONSOLE PRINTER
                        *
006F 0  0000            QTY1  DC         0     STORAGE FOR RESULTS IN BINARY
                        *
0070 0  0000            EOJSW DC         0     END OF JOB SWITCH
0071 0  1000            EOJMK DC         /1000 END OF JOB MASK (LAST CARD)
                        *
0072    0050            CRDIN BSS        80
                        *
00C2    0000                  END        LOAD
```

SYMBOL TABLE

BACK	0004	CHK	000B	CNSL1	005E	CNSL2	0060	CONSL	0033
CONI	0067	CRDIN	0072	EOJ	0066	EOJLC	0062	EOJMK	0071
EOJSW	0070	FEED	0064	ILSW4	003A	LCTST	003E	LEV0	0018
LEV4	0024	LOAD	0000	LOOP	004F	QTY1	006F	READ	0022
SENCL	0038	SENS	0016	SENS0	0020	SENS4	003C	START	0014
WORK	006D								

NO ERRORS IN ABOVE ASSEMBLY.

SESSION: T1C

SUBJECT: A 16 Channel On-line Averaging Program

AUTHOR: J. L. Grisell

Lafayette Clinic

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: W3F

SUBJECT: 1. 1800 Remote Keyboards

2. Lafayette Clinic DAO system

AUTHOR: R. Gidobba and J. Porzak

Lafayette Clinic

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: W4G

SUBJECT: A guide to writing interrupt service routines for the 1130 paper tape attachments

AUTHOR: A. Sandberg and D. Rappaport
Presbyterian - St. Lukes's Hospital

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: W3G

SUBJECT: A guide to writing Interrupt Service Routines for the 1130 console typewriter/printer

AUTHOR: W. Martin

U.S. Navy

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: M4D

SUBJECT: Use of packed decimal in punch card accounting

AUTHOR: G. Fishkorm

Westinghouse Corp.

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: T6J

SUBJECT: The application of the 1130 in X-Ray Crystalography

AUTHOR: J. Chambers

IBM

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: T3E

SUBJECT: The use of computer graphic equipment for parts programming of numerically controlled machine tools

AUTHOR: N. F. Michelsen

IBM

Eastern Regional Office

590 Madison Avenue

New York City, New York

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: M4F - M5F

SUBJECT: MPX - TSX: Comparison and system selection

AUTHOR: B. Landeck

IBM

San Jose, California

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: M6F

SUBJECT: 1800 FE TYPE I Program Support

AUTHOR: W. C. Bultman

IBM

San Jose, California

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: W1F

SUBJECT: 1800 MPX/360 OS hybrid system using channel to channel adapters

AUTHOR: J. L. Clarke

IBM

San Jose, California

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: W4A

SUBJECT: System 360 Mod 25

AUTHOR: R. Flynn

IBM

Eastern Regional Office

Field Support Center

590 Madison Avenue

New York City, New York

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: T4J

SUBJECT: The experience of a user in typing multiple chromatographs to the 1800

AUTHOR: H. C. Lawrence & K. Burkhardt
American Cyanamid
Boundbrook, New Jersey

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: M3D

SUBJECT: Mod 20 Systems including description and discussion on various configurations

AUTHOR: A. C. Wilaby

IBM

7 Penn Center Plaza

Philadelphia, Pennsylvania

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: T2D

SUBJECT: Student Information System

AUTHOR: W. G. Verbrugge

St. Josephs College

Rentsselaer, Indiana 47978

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: M3ABC

SUBJECT: 1. Mod 44 Commercial Feature - Hardware addition to provide DOS for Mod 44 users

2. RACS - IBM's Remote Access Computing System for the Mod 44

3. DAMPS - IBM's Data Acquisition Multiprogramming System for support of online scientific applications (Mod 44)

AUTHOR: P. Manikowski

IBM DPHQ

White Plains, New York

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: T4H - T5H

SUBJECT: Tutorial - Problem Language Analyzer (PLAN)

AUTHOR: R. Weber

IBM DPHQ

White Plains, New York

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.

SESSION: T1E

SUBJECT: 1130/360 N. C. Parts Programming

AUTHOR: H. B. Randall, Jr.

IBM DPHQ

White Plains, New York

Note: At time of publication, a copy of this paper was not available. To obtain a copy, please contact the author directly.