NASA Technical Memorandum 80067

NASA-TM-80067 19790013628

INTERPRETIVE COMPUTER SIMULATOR FOR THE NASA STANDARD SPACECRAFT COMPUTER-II (NSSC-II)

RUDEEN S. SMITH

AND

MARIE S. NOLAND

MARCH 1979

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

1 Report No	2 Government Accession No	3 Recipient's Catalog No
NASA TM 80067		

4 Title and Subtitle	5 Report Date
Interpretive Computer Simulator for the NASA Standard Spacecraft Computer-II (NSSC-II)	March 1979
	6 Performing Organization Code

7 Author(s)	8 Performing Organization Report No
Rudeen S. Smith Marie S. Noland	
	10 Work Unit No
9 Performing Organization Name and Address	
NASA Langley Research Center Hampton, Virginia 23665	11 Contract or Grant No
	13 Type of Report and Period Covered
12 Sponsoring Agency Name and Address	Technical Memorandum
National Aeronautics and Space Administration Washington, DC 20456	14 Sponsoring Agency Code

15 Supplementary Notes

16 Abstract

An Interpretive Computer Simulator (ICS) for the NASA Standard Spacecraft Computer-II (NSSC-II) has been implemented on the CYBER series computer system at Langley Research Center. The ICS is written in the higher level language PASCAL. The NSSC-II is basically an IBM System/360 (S/360) with additional short (16-bit) and double (64-bit) precision instructions and a 32-bit floating-point instruction set. The design of the ICS is general enough to be used as a S/360 simulator. The system is implemented with sixteen 32-bit general registers, 4 floating-point registers, 64 storage protect registers, a 65K memory, a real-time clock, an interval timer, and a checkpoint/restart capability. The report describes the structural design of the ICS, the interrupt handling capabilities, and discusses the instruction definitions and the implementation of the instructions. Included in the paper are the instruction timings, an example of the control cards required to access the ICS system, and a sample program with the associated output. The output file provides, per instruction, a value race, a time history, and the program status. A program listing and card deck may be obtained from the Computing Software Management and Information Center (COSMIC) of the University of Georgia. The ICS has been used for preliminary verification and testing of NSSC-II flight software for the Annular Suspension and Pointing System (ASPS) project.

17 Key Words (Suggested by Author(s))	18 Distribution Statement
ASPS PASCAL CYBER S/360 Software Verification and Testing NSSC-II ICS	Unclassified - Unlimited Subject Category 61

19 Security Classif (of this report)	20 Security Classif (of this page)	21 No of Pages	22 Price*
Unclassified	Unclassified	56	$5.25

* For sale by the National Technical Information Service, Springfield, Virginia 22161

TABLE OF CONTENTS

SUMMARY

An Interpretive Computer Simulator (ICS) for the NASA Standard Spacecraft Computer-II (NSSC-II) has been developed at Langley Research Center as a code verification and testing tool for the Annular Suspension and Pointing System (ASPS) project. The simulator is written in the higher level language PASCAL and implemented on the CDC CYBER series computer system. It is supported by a meta assembler, a linkage loader for the NSSC-II, and a utility library to meet the application requirements.

The architectural design of the NSSC-II is that of an IBM System/360 (S/360) and supports all but four instructions of the S/360 Standard Instruction Set. This paper discusses the structural design of the ICS, with emphasis on the design differences between it and the NSSC-II hardware. The program flow is diagrammed, with the function of each procedure being defined; the instruction implementation is discussed in broad terms; and the instruction timings used in the ICS are listed.

Included in the paper is an example of the steps required to process an assembly level language program on the ICS. The example illustrates the control cards necessary to assemble, load, and execute assembly language code; the sample program to be executed; the executable load module produced by the loader; and the resulting output produced by the ICS.

The ICS was designed as a verification and testing tool for the NSSC-II, but is general enough to have applications as a basic S/360 simulator.

INTRODUCTION

This report is to serve as the reference manual for the NSSC-II Interpretive Computer Simulator, but should be used in conjunction with the IBM Principles of Operation Manual (ref. 1) for the NSSC-II. The report provides a structural description of the ICS, a general discussion of the NSSC-II instruction set, and any variations between the ICS and the NSSC-II hardware. The IBM document should be referred to for in-depth discussions of the NSSC-II structural design and instruction definitions.

The NSSC-II architecture is basically that of an IBM System/360 and supports 83 of the 87 instructions in the S/360 Standard Instruction Set. The four unsupported S/360 instructions are discussed in the IBM manual (ref. 1).

The NSSC-II hardware provides additional short and double precision instructions and a 32-bit floating-point instruction set.

NSSC-II INTERPRETIVE COMPUTER SIMULATOR STRUCTURAL DESIGN

The NSSC-II ICS is written in PASCAL for the CDC CYBER series computers. The CYBER computer is a ones complement machine with a 60-bit word size. The NSSC-II is a twos complement machine with a 32-bit word size. As discussed below, these two incompatabilities had considerable impact on the design of the ICS.

The ones complement host machine means all arithmetic operations are performed in ones complement arithmetic. The arithmetic operands are converted from twos complement to ones complement, the operation is performed, and the ones complement result is then converted to twos complement before storing the value.

In a ones complement machine there is a positive and negative zero represented by all zeros or all ones, respectively. Negative zero in ones complement has the same representation as a minus one in twos complement. This causes a major consideration in the design of the ICS, and requires special handling in many of the arithmetic operations. These cases are discussed in the appropriate procedure definitions.

In a twos complement system there is no negative zero, thereby creating one more negative number than exists in a ones complement system. The twos complement maximum negative number (the negative number with the greatest absolute value) is not representable in ones complement. A message to this effect is given when the maximum negative number is used in an operation that requires conversion to ones complement. Generally, the operation is performed with the value still in twos complement form.

The fixed-point word size on the NSSC-II is 32 bits for full word, 16 bits for halfword, and 64 bits for double word. The 60-bit CYBER word is used to represent the fixed-point full and half words with the upper 28 and 44 bits, respectively, used for sign extension. The 64-bit double word is represented by two 60-bit words with the first word containing the sign extension in the upper 28 bits and the sign and upper 31 bits of the double word in the lower 32 bits. The second word contains the low 32 bits of the double word in the lower 32 bits; the upper 28 bits are extended with zeros. The upper 28 bits of the full and double word, and the upper 44 bits of the halfword, are ignored when storing the result but are used to record overflow.

The NSSC-II floating-point word size is 32 bits, with a sign bit, a 7-bit exponent, and a 24-bit fraction. When represented as a 60-bit CYBER word, the upper four bits of the 11-bit exponent and the lower 24 bits of the 48-bit fraction are padded with zeros.

Figure 1 shows the 60-bit CYBER representation for the NSSC-II fixed-point and floating-point data forms. Refer to figure 3 for the NSSC-II representation of these data forms.

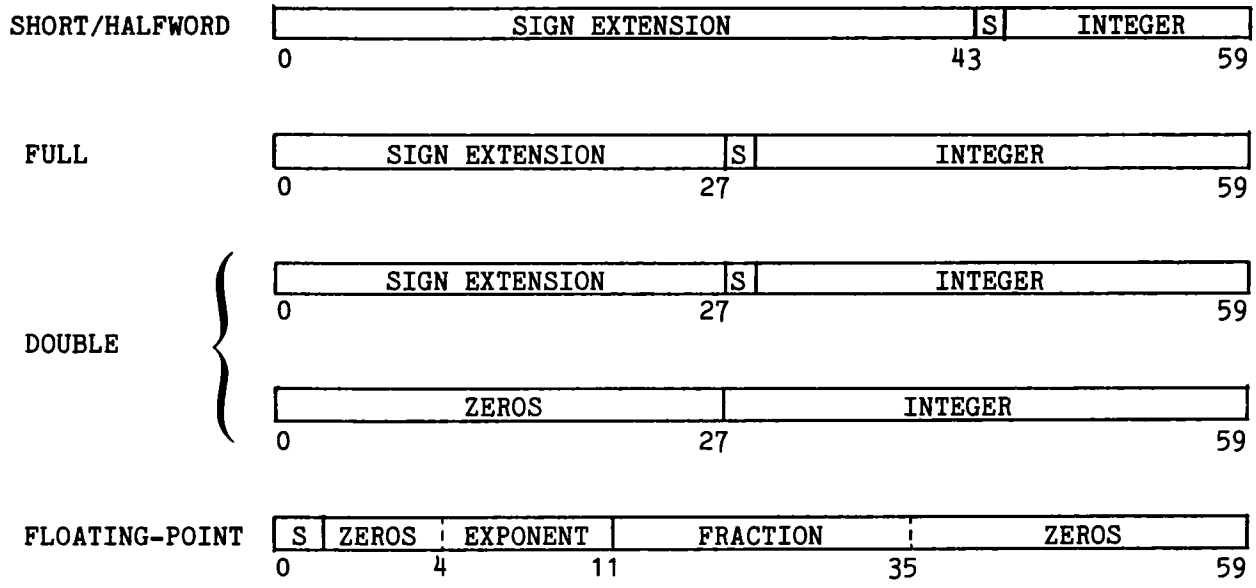


Figure 1.

The ICS and NSSC-II differ somewhat in the handling of program interrupts. During the fetch cycle, the NSSC-II hardware suppresses instructions on some program interrupts while terminating others. The ICS suppresses instructions on all such interrupts except for the addressing exception of the Translate (TR) and Translate and Test (TRT) instructions, which are terminated. The priority of the program interrupts in the ICS are as follows: operation code, addressing, specification, storage protection, privileged, data, execute, floating-point register, fixed-point divide, fixed-point overflow, floating-point divide, exponent overflow, significance, and exponent underflow. An operation exception occurs when the operation code (op code) is not defined. An addressing exception occurs when an address exceeds the available storage. A specification exception occurs when proper alignment is not specified for an operand or an improper register is designated. A storage protection exception occurs when an instruction tries to store into a protected location. A privileged exception occurs when a privileged instruction is encountered in the

problem state. A data exception occurs when a sign or digit code is incorrect. An execute exception occurs when the subject instruction is an EXECUTE instruction. A floating-point register exception occurs when a register designator greater than 6 is specified. A fixed-point divide exception occurs when a quotient exceeds the register size or the divisor is zero. A fixed-point overflow exception occurs when high-order significant bits are lost in add, subtract, and shift operations. A floating-point divide exception occurs when the divisor has a zero fraction. An exponent overflow occurs when the result exponent from an add, subtract, multiply, or divide operation exceeds 127 and the result fraction is not zero; the operation is completed and the exponent is 128 smaller than the correct exponent. A significance exception occurs when the result fraction of an add or subtract operation is zero; if PSW bit 39 is on, an interrupt occurs. An exponent underflow occurs when the result exponent from an add, subtract, multiply, halve, or divide operation is less than zero and the result fraction is not zero. If PSW bit 38 is on, an interrupt occurs; otherwise, the operation is completed and the exponent is 128 larger than the correct exponent. The NSSC-II hardware handles the privileged instructions as special cases; therefore, the program interrupts may not occur in the ICS in the same order as they do in the NSSC-II. Program interrupts are discussed further in the procedure definitions.

The program interrupt codes used in the ICS are as follows:

INTERRUPT CODE		PROGRAM INTERRUPT CAUSE
0	00000000	PSW Key Not Zero
1	00000001	Operation
2	00000010	Privileged Operation
3	00000011	Execute
4	00000100	Protection
5	00000101	Addressing
6	00000110	Specification
7	00000111	Data
8	00001000	Fixed-Point Overflow
9	00001001	Fixed-Point Divide
10	00001010	Invalid SIO Command
11	00001011	Floating-Point Register
12	00001100	Exponent Overflow
13	00001101	Exponent Underflow
14	00001110	Significance
15	00001111	Floating-Point Divide

Supervisor interrupts are handled through the execution of the SUPERVISOR CALL (SVC) instruction, INPUT/OUTPUT interrupts are handled through the execution of the START I/O (SIO) instruction, and external interrupts are handled through the interval timer. NSSC-II machine (hardware) check interrupts can not be detected by the ICS.

The NSSC-II interval timer and real-time clock are implemented on the ICS and accessed by the TIMER READ AND SET (TMRS) instruction. The 16-bit interval timer has a maximum time of 7.3818624 seconds (hex FFFF) and is decremented

every 112.64 microseconds. An underflow interrupt will occur if the system mask bit 7 is set; a mask of 0 leaves the interrupt pending. The timer is reset to zero to simulate the pending state until the timer mask bit is set, at which time an interrupt will occur. The 32-bit real-time clock is incremented every 112.64 microseconds and has no overflow interrupt.

The ICS is implemented with 16 32-bit general registers (numbered 0 through 15), four floating-point registers (numbered 0,2,4,6), and a memory size of 65536 bytes (64 1024-byte blocks). The memory address range is 0 to 65535 with no memory wraparound. Any memory location greater than 65535 will produce an addressing exception. Memory locations 0 through 131 are reserved for the permanent storage assignments described below:

LOCATION	LENGTH	PURPOSE
0	Double Word	Initial Program Loading PSW
8	Double Word	Unused
16	Double Word	Unused
24	Double Word	External Old PSW
32	Double Word	Supervisor Call Old PSW
40	Double Word	Program Old PSW
48	Double Word	Machine Check Old PSW
56	Double Word	Input/Output Old PSW
64	Double Word	Buffered I/O Status Word
72	Word	Channel Address Word
76	Word	Unused
80	Word	Unused
84	Word	Unused
88	Double Word	External New PSW
96	Double Word	Supervisor Call New PSW
104	Double Word	Program New PSW
112	Double Word	Machine Check New PSW
120	Double Word	Input/Output New PSW
128	Word	Unused

The general registers, floating-point registers, storage protect registers, program status word, real-time clock, total time clock, increment time clock, and memory are initialized to zero unless the Checkpoint/Restart mode is in effect. The Checkpoint/Restart status is determined by the first value of the control file CONTRLF. A zero indicates preinitialization with memory being loaded from the file DATAF and execution starting at the transfer address provided by DATAF. For anything other than zero, the registers, program status word, clocks, and memory are loaded from the file CKPTF and execution resumes at the location stored in the instruction address (bits 48-63) of the PSW. Checkpoint occurs when the clock time stored at the effective memory address is exceeded by the real-time clock value. The memory addresses for both clocks must be aligned on a full word boundary.

The control file CONTRLF is a text file that contains a '0' or '1' in column 1.

The file DATAF is a text file with the following format. The first column of each line is a blank.

```
line 1    : Program name
line 2    : XXXXY----------Y
   .              .
   .              .
   .              .
   .              .
line n-1  : XXXXY----------Y
line n    : FFFFZZZZ
```

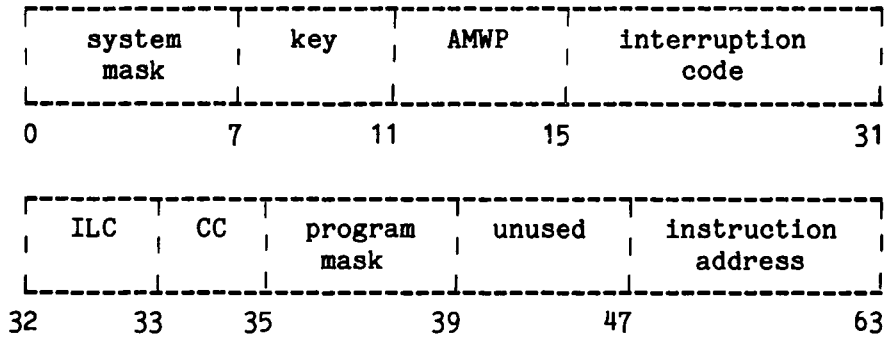
where XXXX is memory location
Y-------Y is code and/or data
FFFF is file terminator
ZZZZ is transfer address

CKPTF is a file of integers with the current register, PSW, clock, and memory values at the time of checkpoint.

Checkpoint/Restart is activated through the Timer Read and Set (TMRS) instruction. The NSSC-II defines the first operand of the instruction for values of 0 (read real-time clock), 1 (read interval timer), 2 (read/set real-time clock), and 3 (read/set interval timer). The ICS uses the value of 4 to signal a checkpoint condition. If the first operand has a value of 4 and the real-time clock is greater than the clock time stored at the effective memory address, then a checkpoint occurs. When the program is restarted, if the value in column 1 of the control file CONTRLF is '1', execution resumes where the checkpoint occurred; otherwise, execution starts from the beginning of the program code.

The files CONTRLF, DATAF, and CKPTF must be precreated as permanent files and replaced at the termination of each program execution.

The Program Status Word (PSW) is a double word (64-bits) that contains the information required for proper program execution. In general, the PSW is used to control instruction sequencing and to hold and indicate the status of the system. The PSW format is as follows:



0	System (I/O) Mask	32-33	Instruction Length Code (ILC)
1-6	Unused	34-35	Condition Code (CC)
7	System (Timer) Mask	36-39	Program Mask
8-11	Must be 0	36	Fixed-Point Overflow Mask
12	ASCII (A)	37	Unused
13	Machine Check Mask (M)	38	Exponent Underflow Mask
14	Wait State (W)	39	Significance Mask
15	Problem State (P)	40-47	Unused
16-31	Interruption Code	48-63	Instruction Address

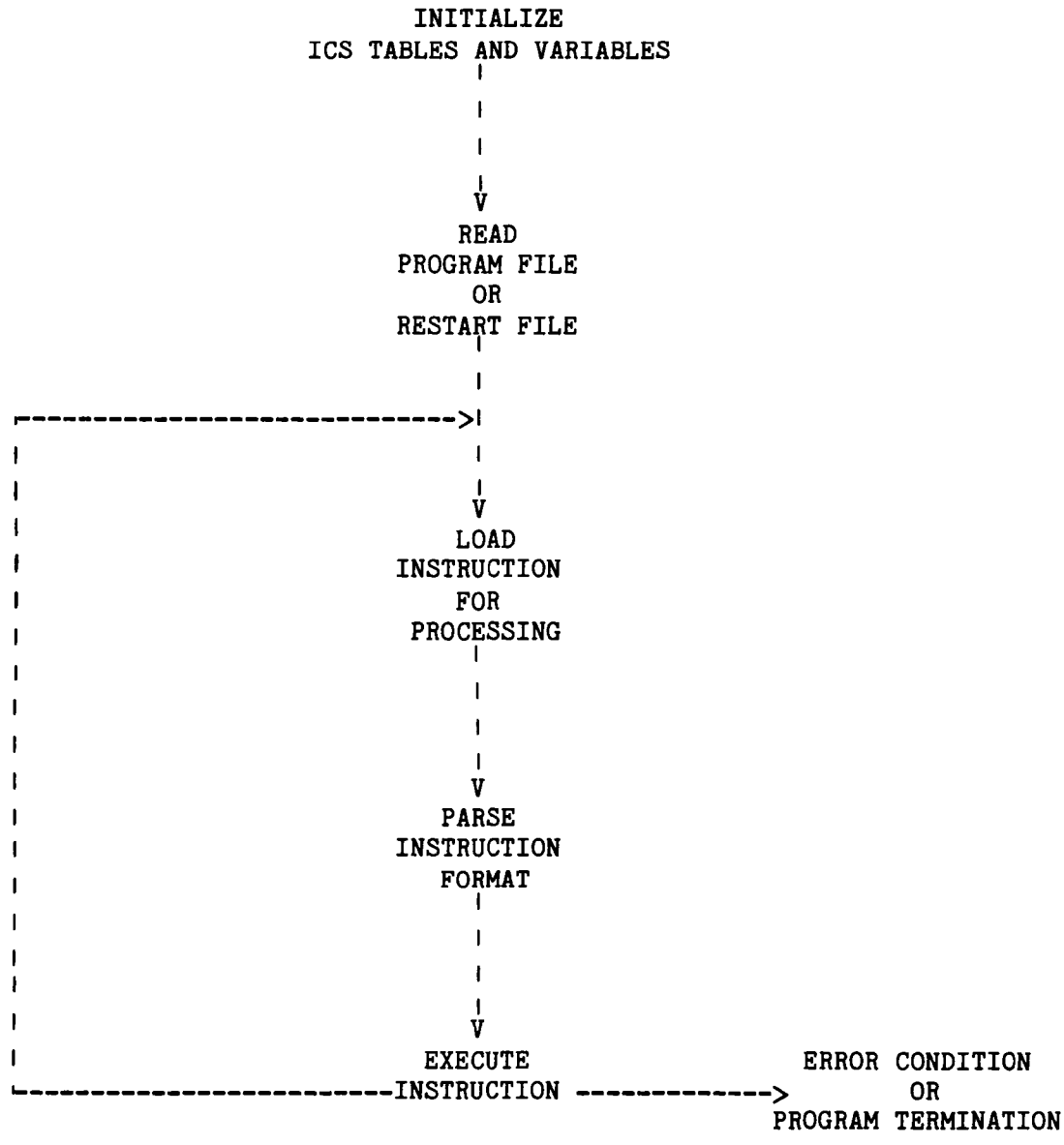
In addition to issuing the supervisor-call interruption, the Supervisor Call (SVC) instruction may be used to terminate a program. If the I field of this instruction is a 3, the program will be terminated at that point. For any other value, the instruction will be processed normally.

The purpose of the DIAGNOSE instruction is for NSSC-II hardware testing. Since this instruction is not intended for problem or supervisor usage, it has not been implemented on the ICS.

The control sequence required to access the ICS is illustrated in appendix A with a control card setup, a sample program, and the resulting output.

Appendix B lists the timings used for the NSSC-II instruction sets.

PROGRAM FLOW DIAGRAM



PROCEDURE DEFINITIONS

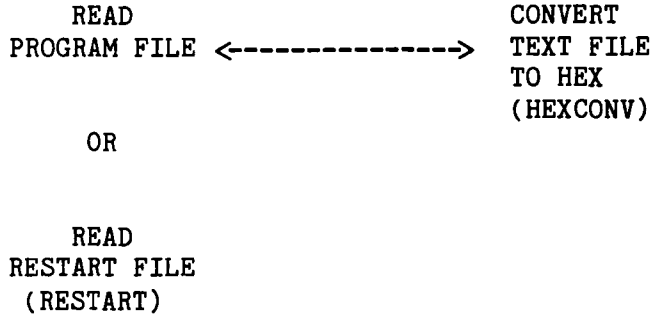
The procedure definitions will be ordered as they are referenced in the ICS. The definition of the primary level procedures will be preceded by a diagram showing the secondary level procedures that may be referenced from the primary level, followed by a discussion of each procedure in the diagram. The secondary level procedure names will appear in parenthesis.

Procedure INITIALIZE

Program tables, arrays, and variables are initalized in this procedure. These include the registers, memory, the block protection array, the clocks, and, per op code, the instruction mnemonic, the instruction format, and the instruction exception conditions.

The fetch cycle exception conditions are initialized in the following order: addressing, specification, protection, privileged, data, execute, and floating-point register check. The execute cycle exceptions are divide fault and overflow/underflow checks, followed by the appropriate condition code setting.

Procedure READINPUT



The first character of the control file CONTRLF is read; if the character is '0' the textfile DATAF is read; if the character is '1' the integer file CKPTF is read.

READINPUT calls the procedure HEXCONV to convert the textfile to its binary equivalent before loading memory. DATAF provides the addresses for loading memory, the memory values, and the transfer address for the start of program execution. If the program is to be restarted, the procedure RESTART is called to read the restart file CKPTF and the transfer address is provided by the program status word.

Procedure HEXCONV

The character value is translated to its hexadecimal equivalent.

Procedure RESTART

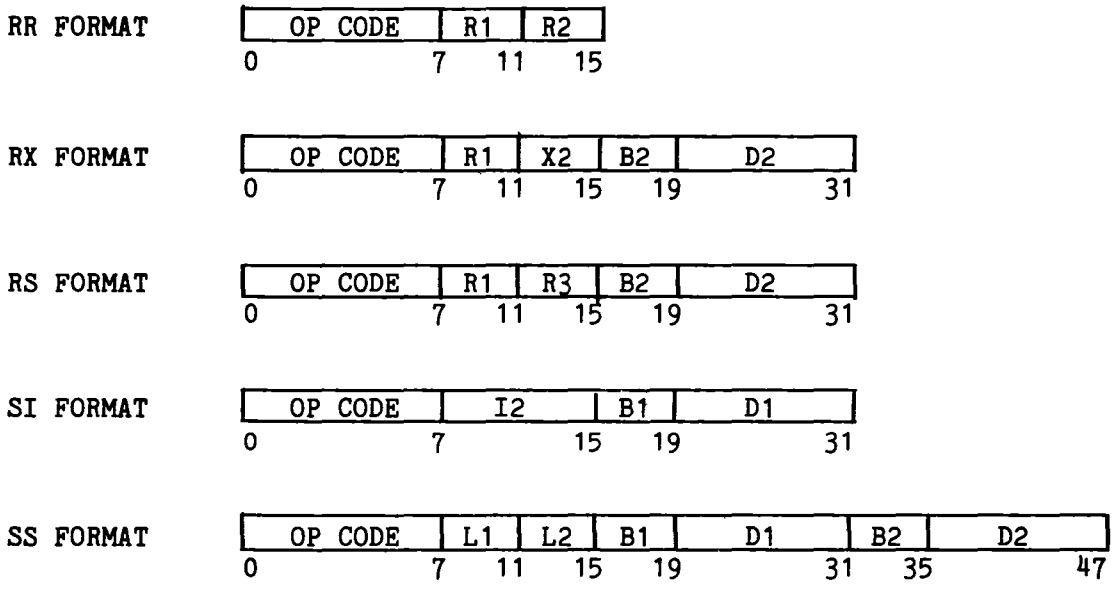
The character '0' is written to CONTRLF to reset the control file for subsequent executions. The registers, memory, PSW, and clocks are loaded from the file CKPTF.

Procedure INSTLOAD

The instruction is loaded from memory into an instruction record for processing. Addressing exception is checked and the instruction length code and instruction address of the program status word are updated.

The first two bits of the instruction op code specifies the length and format of an instruction (figure 2). Some of the short instructions do not adhere to this rule and are handled as special cases in this procedure.

The five basic instruction formats are RR, denoting a register-to-register operation; RX, a register-and-indexed storage operation; RS, a register-and-storage operation; SI, a storage and immediate-operand operation; and SS, a storage-to-storage operation.



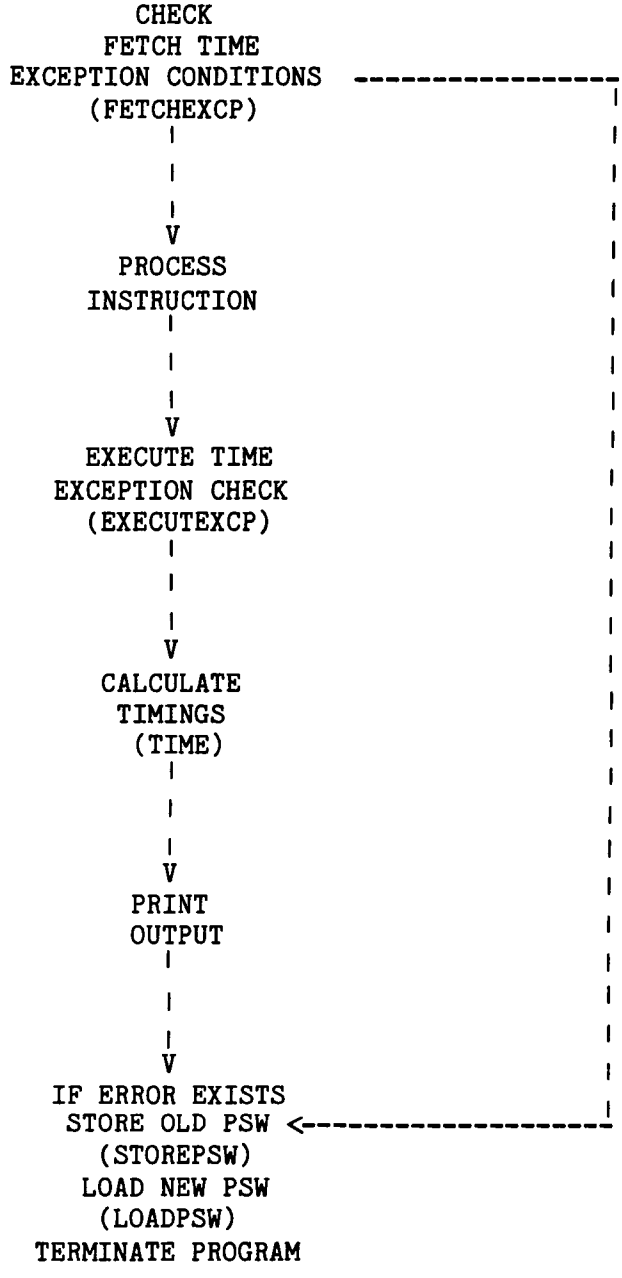
BIT POSITIONS (0-1)	INSTRUCTION LENGTH	INSTRUCTION FORMAT
00	ONE HALFWORD	RR
01	TWO HALFWORDS	RX
10	TWO HALFWORDS	RS or SI
11	THREE HALFWORDS	SS

Figure 2. BASIC INSTRUCTION FORMATS

Procedure FORMAT

An instruction is parsed into the appropriate operand fields. The effective address of the RX, RS, SI, and SS instructions are calculated and the RX flag set for the RX instructions. This flag is used in calculating the RX instruction times (refer to appendix A).

Procedure EXECUTE



The fetch time exception conditions for the instruction are checked (FETCHEXCP); if no error exists, the instruction is processed. After instruction processing is completed the execute exceptions are checked and the condition code is set (EXECUTEXCP), the timing is updated (TIME), and trace and timing information are written to the output file. If no error exists, the next instruction is loaded. If an error does exists, the current PSW is stored as the program old PSW (STOREPSW) and the program new PSW becomes the current PSW (LOADPSW); program execution is terminated.

Due to the number of procedures referenced by EXECUTE, the descriptions will be in alphabetical order rather than in the order referenced. Any procedures that are referenced within another procedure will be denoted by putting the referenced procedure name in parenthesis.

Procedure ADD

The twos complement operands are converted to ones complement (ONESCOMP) and the addition is performed in ones complement arithmetic. If the result is a negative zero, it is converted to a twos complement positive zero before storing the results. A carry out must be tested for in ones complement arithmetic and an end around carry performed if one exists. The result is then converted to twos complement form (TWOSCOMP) and stored in the first operand register.

Procedure ADDDBLE

The twos complement double precision operands are converted to ones complement double precision operands (ONESCOMPD). The lower half of the operands are added, then the upper halves and any carry out of the lower halves are added. An end around carry is performed if there is a carry out of the upper halves. A negative zero result is converted to a positive zero and the carry out test for the lower and upper halves continue until no carry out exists. The double precision result is converted to twos complement form (TWOSCOMPD) and stored in the register pair designated by the first operand.

Procedure ADDLOG

A 32-bit logical add is performed and the result is stored in the first operand register.

Procedure ADDREAL

The two floating-point operands are converted to the CDC CYBER system floating-point format (CDCREAL). The reformatted operands are then added and the result is converted back to the NSSC-II floating-point format (NSSCIIREAL) before storing the result in the first operand register.

Procedure ADDRCK

The address range of an operand is checked before execution of the instruction. If any part of the operand exceeds the maximum address range, an addressing exception error occurs whether or not the data is used in the instruction.

Procedure ADDSHORT

The maximum negative short precision number is not representable in ones complement notation. If either operand is the short maximum negative number, a warning statement is issued, the number is not changed, and the add operation proceeds (ADD).

Procedure CDCREAL

A floating-point operand is converted to the corresponding CDC Cyber floating-point format before a real arithmetic operation is performed.

Procedure CKPOINT

The control file CONTRLF is set to indicate restart mode for subsequent runs. The general registers, floating-point registers, storage block registers, memory, current PSW, real-time clock, interval timer, increment clock, and total time clock are dumped to the checkpoint file CKPTF. This file is used to restart the program at the point where the checkpoint occurred.

Procedure CKSHIFT

The CYBER machine has a left shift wraparound; whereas, the NSSC-II fills the vacated bits with zeros. If a shift wraparound occurs, this procedure voids the wraparound effect.

Procedure DIVIDERR

On a fixed-point divide error an error message is written out, the error condition is set, and the interruption code is set in the PSW. A fixed-point divide error occurs for a quotient value exceeding the register size, for a division by zero, or when the result in Convert to Binary (CVB) exceeds 31 bits.

Procedure EXECUTEXCP

The execution time exception conditions are checked in this procedure. The exceptions consist of a fixed-point overflow check, a floating-point divide check, exponent overflow/underflow checks, a floating-point significance check,

and the setting of the condition codes. Condition codes are set for the following resultant conditions: 1) =0, <0, or >0; 2) overflow; 3) carry; 4) compare equal; 5) compare logical equal; 6) 0 or not 0 compare; 7) all zeros, all ones, or mixed values; and 8) characteristic =0, <0, or >0. A fixed-point overflow will produce an interrupt only if the program mask bit 36 is set. A floating-point significance exception will cause an interrupt to occur only if the program mask bit 39 is set, and an exponent underflow will produce an interrupt only if program mask bit 38 is set.

Procedure EXPONENTCK

The exponent of the floating-point operand is checked for underflow (<0) and overflow (>127) conditions.

Procedure FETCHEXCP

The fetch time exceptions check for addressing (ADDRCK), specification (SPECCK and ODDREGCK), protection (PROTECTERR), privileged (STOREPSW and LOADPSW), data, execute, and floating-point register (REGCK) error conditions are made in this procedure. If an error exists, the instruction is suppressed and execution is terminated.

Procedure LOAD

Memory is loaded into a temporary register (LOADREG) and the memory location pointer is updated by the number of bytes loaded.

Procedure LOADBYTE

A byte from memory is loaded into the lower 8 bits of a temporary register and the memory location pointer is increased by 1.

Procedure LOADDBLE

The eight bytes of memory that make up the second double precision operand are loaded into two temporary registers (LOAD), and the first double precision operand is loaded into two temporary registers.

Procedure LOADDREG

Both double precision operands are loaded into temporary registers.

Procedure LOADPSW

The Program Status Word is loaded from memory.

Procedure LOADREG

From one to four bytes from memory are loaded into a temporary register.

Procedure LOGICAL

The logical operations AND, OR, and XOR are performed on the operands according to the instruction being processed. The set operations operate on 59 of the 60 bits of a word. The reserved bit is used for operational purposes within PASCAL. The bit reserved on the PASCAL version in use at LRC is bit 0, but this is implementation dependent.

Procedure MAXNUMD

In the subtraction of double precision values, if both operands are the maximum negative value, the result zero is stored in the first operand registers. For any other combination of values the second operand is negated (NEGATEDBLE) and the two operands are then added (ADDDBLE).

Procedure MAXNUMF

If two maximum negative numbers are subtracted, the result is zero; otherwise, the second operand is negated (NEGATE) and added to the first operand (ADD).

Procedure MAXNUMS

If two short maximum negative operands are subtracted, the result is zero; otherwise, the second operand is negated (NEGATE) and added to the first operand (ADDSHORT).

Procedure NEGATE

The twos complement of the operand is performed. Plus one, minus one, and the maximum negative number must be handled as special cases. The maximum negative number cannot be complemented and a message to this effect is given.

Procedure NEGATEDBLE

If the operand is the maximum negative double precision number, the negating does not take place and a message is given. For any other value, the twos complement of the operand is performed. Minus one and plus one must be handled as special cases.

Procedure NORMALIZE

The fractional part of the floating-point operand is digit normalized; that is, the uppermost digit may not be 0, but the uppermost bit may be. The exponent is corrected accordingly; for each left digit shift of the fraction, the exponent is reduced by 1.

Procedure NSSCIIREAL

The floating-point operand is converted back to the NSSC-II floating-point format. Since the NSSC-II is a digit biased machine and the CDC machine is bit biased, the NSSC-II exponent and fraction must be corrected accordingly.

Procedure ODDREGCK

If an odd register is specified when an even/odd register combination is required, a specification error occurs. The error condition and interruption code are set.

Procedure ONESCOMP

The negative twos complement operand is converted to ones complement if the operand is not the maximum negative number. If it is, a message is given and the operand remains unchanged. Minus one must be handled separately.

Procedure ONESCOMPD

The negative twos complement double precision operand is converted to ones complement if the operand is not the maximum negative double precision number. If it is, a message is given and the value remains unchanged. Minus one must be handled separately.

Procedure ONESCOMPS

If the twos complement operand is the maximum negative short number, then a message is given. The operand is then converted to ones complement (ONESCOMP).

Procedure PRENORMALIZED

The operands of the floating-point operations multiply and divide are prenormalized before the operation is performed. The exponent of the operands is corrected according to the number of digit shifts required for the prenormalization.

Procedure PROTECTERR

Before a memory byte is stored into, the corresponding storage block is checked for storage protection. If the block is protected, a storage protect exception occurs and the error condition and interruption code are set.

Procedure PROTECTCK

If the instruction terminated prematurely, a protection exception occurred. An error statement is output, the error condition is set, the interruption code is set, and the timing parameter is set to the number of bytes processed before termination.

Procedure REGCK

The NSSC-II has four floating-point registers, numbered 0, 2, 4, 6. An error exception occurs if the register designator is greater than 6. An error message is written and the error condition is set.

Procedure SHIFTD

The timing parameters are evaluated according to the number of bits the double precision operand was shifted.

Procedure SHIFTF

The timing parameters are evaluated according to the number of bits the full precision operand was shifted.

Procedure SHIFTL

The operand is shifted left by multiplying by the appropriate power of 2. The vacated bits are filled with zeros. A shift greater than 30 shifts all zeros into the 31-bit arithmetic operand with the sign being retained. For the logical operands, all 32 bits participate in the shift.

Procedure SHIFTLD

For shifts less than 32, the upper half of the double precision operand is shifted left (SHIFTL) and the result is checked for any left shift wraparound effect (CKSHIFT). The lower half is then moved into the upper half according to the number of bits specified. For shifts greater than or equal to 32, the second half is shifted into and possibly out of the first half and zeros fill the vacated bits (SHIFTL and CKSHIFT). The timing parameters are set to reflect the shift that takes place.

Procedure SHIFTR

The operand is shifted right by dividing by the appropriate power of 2. An arithmetic shift propagates the sign through the vacated bits; a shift greater than 30 propagates the sign throughout the arithmetic operand. For the logical operand, zeros are supplied to the upper bits. Minus one and negative values must be handled as special cases.

Procedure SHIFTRD

For shifts less than 32, the lower half of the double precision operand is shifted out and the upper half shifted into the lower half the number of bits specified (SHIFTR). For shifts greater than or equal to 32, the lower half is shifted out entirely and the upper half is shifted into and possibly out of the lower half (SHIFTR). The timing parameters are evaluated according to the number of shifts that takes place.

Procedure SPECCK

The boundary requirements specify that a double-word operand must be located on a 64-bit boundary, a full word must be located on a 32-bit boundary, and a halfword/short operand must be located on a 16-bit boundary. If the instruction is not on a halfword boundary or if the operand is not properly aligned, a boundary specification exception occurs, an error message is written, the error condition is set, and the interruption code is set.

Procedure STORE

The contents of a register are stored into memory one byte at a time. Up to four bytes may be stored.

Procedure STOREMULT

Multiple register values are stored into memory (STORE). The memory location is checked for storage protection before the store takes place (PROTECTERR).

Procedure STOREPSW

The current PSW is stored into memory at the specified locations.

Function TIME

The timing calculation is made for the current instruction. The timing parameters TFLAG, TFLAG2, and TFLAGR are evaluated during the instruction process.

Procedure TWOSCOMP

The negative ones complement value is converted to a twos complement value before the result is stored in the result register. Minus one must be handled as a special case.

Procedure TWOSCOMPD

The negative ones complement double precision value is converted to twos complement notation before the result is stored in the result register pair. Minus one must be handled as a special case.

Procedure UNNORMALIZE

The floating-point arithmetic operations performed on the CDC machine require normalized operands. The result of the floating-point unnormalized add and subtract operations (AU, AUR, SU, SUR) must be unnormalized before storing into the result register.

NSSC-II INSTRUCTION IMPLEMENTATION

The instructions will be discussed in groups rather than by each individual instruction. The groupings will be ordered alphabetically and not according to any operational hierarchy.

The data forms generally used in the instructions are full word, double word, halfword, short precision, and floating-point (figure 3). A full word occupies a fixed-length format consisting of a one-bit sign followed by a 31-bit integer field (32-bit signed integer). Some multiply, divide, and shift operations use an operand consisting of 64 bits with a sign bit and a 63-bit integer field (64-bit signed integer). When these operands are in registers, they are located in a pair of adjacent registers and are addressed by an even address referring to the left-most register of the pair. The sign-bit position of the right-most register contains part of the integer. Likewise, the double precision data value is contained in a 64-bit signed integer field and, when in registers, occupies an even/odd register pair.

A halfword data value is a 16-bit signed integer, but most generally a halfword operand is extended to a full word operand before the instruction is processed. The extension is performed by propagating the sign-bit through the 16 high-order bit positions of the register.

Short precision numbers occupy a fixed-length format of a one-bit sign followed by a 15-bit integer field (16-bit signed integer). The short precision quantity occupies the right-most 16 bits of a register; generally, the leftmost 16 bits are neither tested nor altered.

Floating-point operands contain a sign bit field, a 7-bit exponent field, and a 24-bit fraction field. The exponent is expressed in excess 64 binary notation (0-127 range); the fraction is expressed as a hexadecimal number having a radix point to the left of the high order digit. When the floating-point operand is contained in a register, the register must be numbered 0, 2, 4, or 6; any other designation is invalid.

For an in-depth discussion of the instructions, the IBM Principles of Operation (ref. 1) should be referenced.

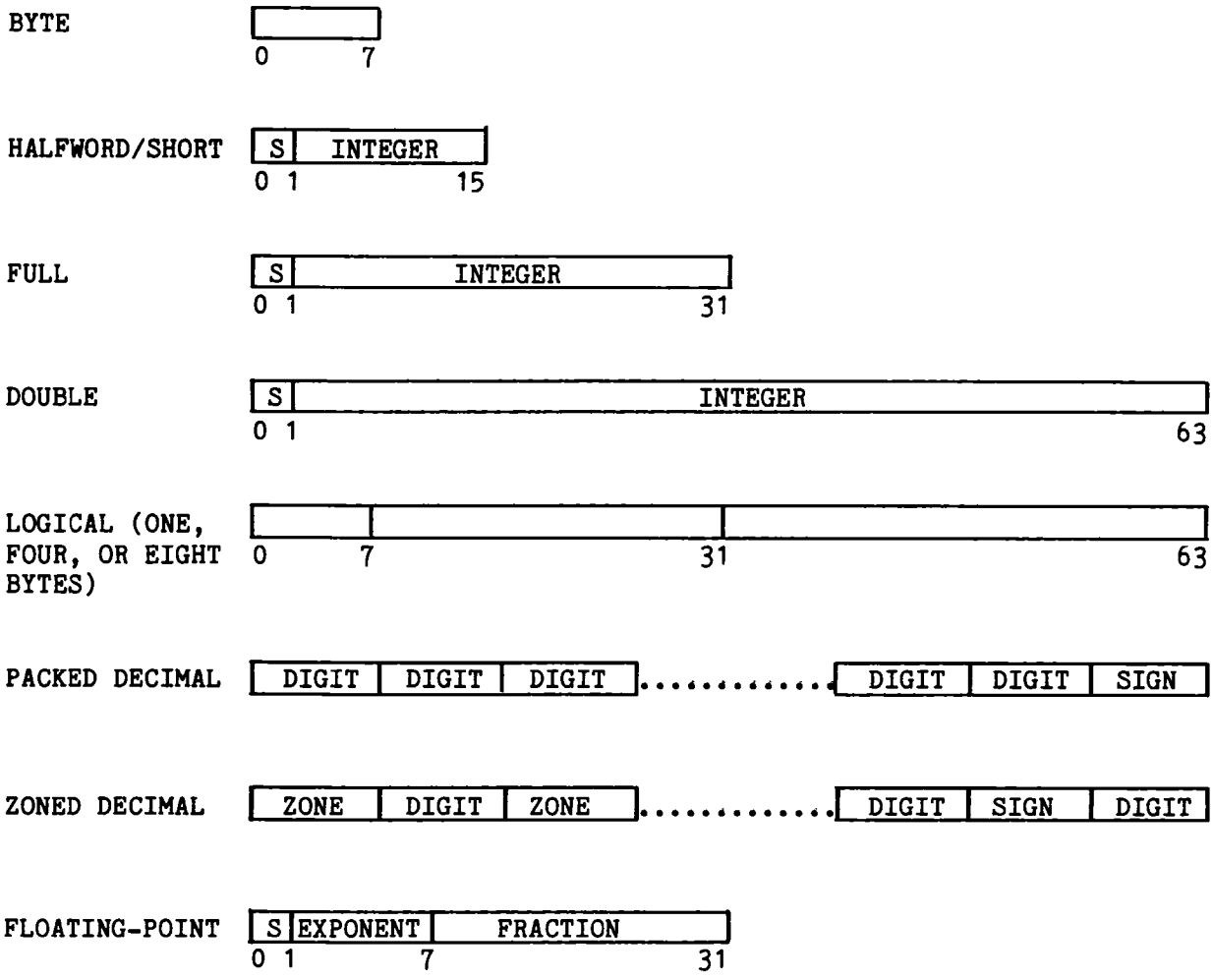


Figure 3. DATA FORMS

ADD INSTRUCTIONS.-

Add	-	A,AR
Add Double	-	AD,ADR
Add Halfword	-	AH,AHI
Add Logical	-	AL,ALR
Add Short	-	AS,ASI,ASR
Add Normalized	-	AE,AER
Add Unnormalized	-	AU,AUR

The second operand is added to the first operand, and the sum is placed in the first operand location.

The halfword second operand is expanded to a full word before the addition by propagating the sign-bit.

Logical addition is performed on 32-bit operands. The occurrence of a carry out of the sign position is recorded in the condition code for the logical adds.

The normalized sum is placed in the first operand after a normalized add; the unnormalized sum is stored after an unnormalized add.

Resulting Condition Code:

	0	1	2	3
Add	=0	<0	>0	overflow
Add Double	=0	<0	>0	overflow
Add Halfword	=0	<0	>0	overflow
Add Logical	=0,no carry	≠0,no carry	=0,carry	≠0,carry
Add Short	=0	<0	>0	overflow
Add Normalized	=0	<0	>0	-
Add Unnormalized	=0	<0	>0	-

Program Interruptions:

Addressing (A,AD,AE,AH,AL,AS,AU)
Specification (A,AD,ADR,AE,AER,AH,AL,AS,AU,AUR)
Fixed-Point Overflow (A,AR,AD,ADR,AH,AHI,AS,ASI,ASR)
Significance (AE,AER,AU,AUR)
Exponent Overflow (AE,AER,AU,AUR)
Exponent Underflow (AE,AER)

Procedures Referenced:

LOAD (A,AH,AL,AS,AE,AU)
LOADDBLE (AD)
LOADREG (ADR)
ADD (A,AR,AH,AHI)
ADDDBLE (AD,ADR)
ADDLOG (AL,ALR)
ADDSHORT (AS,ASI,ASR)

ADDREAL (AE,AER,AU,AUR)
NORMALIZE (AE,AER)
UNNORMALIZE (AU,AUR)

AND INSTRUCTIONS.-

And	-	N,NC,NI,NR
And Short	-	NS,NSI,NSR

The logical product (AND) of the bits of the first and second operand is placed in the first operand location. All bits of the fixed-length integers are treated uniformly.

Resulting Condition Code:

	0	1	2	3
And	=0	≠0	-	-
And Short	=0	≠0	-	-

Program Interruptions:
Addressing (N,NC,NI,NS)
Specification (N,NS)
Protection (NC,NI)

Procedures Referenced:
LOAD (N,NS)
LOGICAL (N,NC,NI,NR,NS,NSI,NSR)
PROTECTCK (NC)

BRANCH INSTRUCTIONS.-

Branch and Link	-	BAL,BALR
Branch on Condition	-	BC,BCR
Branch on Count	-	BCT,BCTR
Branch Unconditional	-	BU,BUR
Branch on Index High	-	BXH
Branch on Index Low or Equal	-	BXLE

The Branch and Link instruction stores the rightmost 32 bits of the PSW as link information in the first operand. The instruction address of the PSW is replaced by the branch address in the second operand unless this operand is register 0.

The Branch on Condition instruction replaces the updated instruction address with the branch address if the state of the condition code is as specified by the mask operand; otherwise, normal instruction sequencing proceeds.

The Branch on Count instruction reduces the first operand by one. If the result is zero, normal instruction sequencing proceeds; otherwise, the instruction address is replaced by the branch address.

The Branch Unconditional instruction replaces the instruction address with the branch address.

The Branch on Index High instruction increments the first operand; the sum is then compared algebraically with a second operand. If the sum is high, the instruction address is replaced by the branch address; otherwise, normal sequencing proceeds. The Branch on Index Low or Equal is a similar instruction except that the branch is taken when the sum is low or equal compared to a second operand.

Condition Code:
The code remains unchanged.

Program Interruptions:
None

Procedures Referenced:
ONESCOMP (BCT,BCTR,BXH,BXLE)
TWOSCOMP (BCT,BCTR,BXH,BXLE)

COMPARE INSTRUCTIONS.-

Compare	-	C,CR
Compare Double	-	CD,CDR
Compare Halfword	-	CH,CHI
Compare Logical	-	CL,CLC,CLI,CLR
Compare Logical Short	-	CLS,CLSI,CLSR
Compare Short	-	CS,CSI,CSR
Compare Floating-Point	-	CE,CER

The first operand is compared with the second operand, and the result determines the setting of the condition code.

The halfword second operand is expanded to a full word before the comparison by propagating the sign-bit.

The logical comparisons are binary, proceeding left to right; the other comparisons are algebraic, treating each operand as a signed integer.

The floating-point comparison is algebraic, considering the sign, fraction, and exponent of each operand.

Resulting Condition Code:

	0	1	2	3
Compare	Operands=	1st Operand Low	1st Operand High	-
Compare Double	Operands=	1st Operand Low	1st Operand High	-
Compare Halfword	Operands=	1st Operand Low	1st Operand High	-
Compare Logical	Operands=	1st Operand Low	1st Operand High	-
Compare Logical Short	Operands=	1st Operand Low	1st Operand High	-
Compare Short	Operands=	1st Operand Low	1st Operand High	-
Compare Floating-Point	Operands=	1st Operand Low	1st Operand High	-

Program Interruptions:
   Addressing (C,CD,CE,CH,CL,CLC,CLI,CLS,CS)
   Specification (C,CD,CDR,CE,CER,CH,CL,CLS,CS)

Procedures Referenced:
   LOAD (C,CD,CE,CH,CL,CLS,CS)
   LOADBYTE (CLC)
   CDCREAL (CE,CER)

CONVERT INSTRUCTIONS.-

Convert to Binary	-	CVB
Convert to Decimal	-	CVD

The Convert to Binary instruction changes the radix of the second operand from decimal to binary, and the result is placed in the first operand location.

The Convert to Decimal instruction changes the radix of the first operand from binary to decimal, and the result is stored in the second operand location. The number is treated as a right-aligned signed integer before and after conversion for both instructions.

Condition Code:
   The code remains unchanged.

Program Interruptions:
   Addressing (CVB,CVD)
   Specification (CVB,CVD)
   Protection (CVD)
   Data (CVB)
   Fixed-Point Divide (CVB)

Procedures Referenced:
   NEGATE (CVB,CVD)
   DIVIDERR (CVB)

DIVIDE INSTRUCTIONS.-

Divide	-	D,DR
Divide Short	-	DS,DSI,DSR
Divide Floating-Point	-	DE,DER
Halve	-	HER

The dividend (first operand) is divided by the divisor (second operand) and then replaced by the remainder and quotient for the full divide. The dividend is a 64-bit signed integer occupying an even/odd pair of registers. The 32-bit signed remainder and 32-bit signed quotient replace the even/odd registers, respectively.

The dividend (first operand) is divided by the divisor (second operand) and then replaced by the quotient for the short divide. No remainder is stored. The dividend is a 32-bit signed integer and is replaced by a 16-bit signed quotient with sign extended.

The floating-point dividend (first operand) is divided by the divisor (second operand) and replaced by the quotient; the remainder is not retained. The initial operands are prenormalized to produce a normalized result. The Halve instruction divides the second operand by two, and the normalized quotient is placed in the first operand location. Exponent overflow and underflow are detected but the underflow mask bit may prohibit the underflow interrupt.

Condition Code:
The code remains unchanged.

Program Interruptions:
Addressing (D,DE,DS)
Specification (D,DE,DER,DR,DS,HER)
Fixed-Point Divide (D,DR,DS,DSI,DSR)
Exponent Overflow (DE,DER)
Exponent Underflow (DE,DER,HER)
Floating-Point Divide (DE,DER)

Procedures Referenced:
LOAD (D,DE,DS)
ONESCOMP (DS,DSI,DSR)
ONESCOMPS (DS,DSI,DSR)
NEGATE (D,DR)
NEGATEDBLE (D,DR)
TWOSCOMP (DS,DSI,DSR)
DIVIDERR (D,DR,DS,DSI,DSR)
PRENORMALIZE (DE,DER)
CDCREAL (DE,DER,HER)
EXPONENTCK (DE,DER)
NSSCIIREAL (DE,DER,HER)
NORMALIZE (DE,DER,HER)

EXCLUSIVE OR INSTRUCTIONS.-

Exclusive Or	-	X,XC,XI,XR
Exclusive Or Short	-	XS,XSI,XSR

The modulo-two sum (exclusive OR) of the bits of the first and second operand is placed in the first operand location. All bits of the fixed-length integers are treated uniformly.

Resulting Condition Code:

	0	1	2	3
Exclusive Or	=0	≠0	-	-
Exclusive Or Short	=0	≠0	-	-

Program Interruptions:
   Addressing (X,XC,XI,XS)
   Specification (X,XS)
   Protection (XC,XI)

Procedures Referenced:
   LOAD (X,XS)
   LOGICAL (X,XC,XI,XR,XS,XSI,XSR)
   PROTECTCK (XC)

EXECUTE INSTRUCTION.-

   Execute - EX

The single instruction at the branch address is modified by the contents of the first operand and the resulting subject instruction is executed. The modification is effective only for this execution of the instruction and does not affect the instruction in memory. Normal execution sequencing resumes following the Execute instruction unless a branch occurs while processing the subject instruction.

Condition Code:
   The code may be set by the subject instruction.

Program Interruptions:
   Addressing
   Specification
   Execute

Procedures Referenced:
   INSTLOAD
   FORMAT
   EXECUTE

INSERT CHARACTER INSTRUCTION.-

   Insert Character - IC

The eight-bit character at the second operand is inserted into the lower byte of the first operand.

Condition Code:
   The code remains unchanged.

Program Interruptions:
   Addressing

Procedures Referenced:
   None

LOAD INSTRUCTIONS.-

Load	-	L,LR
Load Double	-	LD,LDR
Load Halfword	-	LH,LHI,LHR
Load Short	-	LS,LSI,LSR
Load Floating-Point	-	LE,LER

The second operand is placed in the first operand location. The halfword second operand is expanded to a full word before storing by propagating the sign-bit.

Condition Code:
The code remains unchanged.

Program Interruptions:
Addressing (L,LD,LE,LH,LS)
Specification (L,LD,LDR,LE,LER,LH,LS)

Procedures Referenced:
LOAD (L,LD,LE,LH,LS)

Load Address	-	LA
Load Address Short	-	LAS

The address of the second operand is inserted in the low-order 24 bits of the first operand for Load Address; the upper 8 bits are made zero. For Load Address Short the address is inserted in the low-order 16 bits; the upper bits are not altered.

Condition Code:
The code remains unchanged.

Program Interruptions:
None

Procedures Referenced:
None

Load Complement	-	LCR
Load Complement Double	-	LCDR
Load Complement Short	-	LCSR
Load Complement Floating-Point	-	LCER

The twos complement of the second operand is placed in the first operand location.

The floating-point value in the second operand is placed in the first operand with a sign change.

Resulting Condition Code:

	0	1	2	3
Load Complement	=0	<0	>0	overflow
Load Complement Double	=0	<0	>0	overflow
Load Complement Short	=0	<0	>0	overflow
Load Complement Floating-Point	=0	<0	>0	-

Program Interruptions:
Specification (LCDR,LCER)
Fixed-Point Overflow (LCR,LCDR,LCSR)

Procedure Referenced:
NEGATE (LCR,LCSR)
NEGATEDBLE (LCDR)

Load Negative	-	LNR
Load Negative Short	-	LNSR
Load Negative Floating-Point	-	LNER
Load Positive	-	LPR
Load Positive Short	-	LPSR
Load Positive Floating-Point	-	LPER

The twos complement of the positive valued second operand is placed in the first operand location for the Load Negative and Load Negative Short instructions. Negative numbers and zero remain unchanged. The floating-point second operand is placed in the first operand location with the sign made minus (one).

The absolute value of the second operand is placed in the first operand location for the Load Positive and Load Positive Short instructions. Positive numbers and zero remain unchanged. The floating-point second operand is placed in the first operand location with the sign made plus (zero).

Resulting Condition Code:

	0	1	2	3
Load Negative	=0	<0	-	-
Load Negative Short	=0	<0	-	-
Load Negative Floating-Point	=0	<0	-	-
Load Positive	=0	-	>0	overflow
Load Positive Short	=0	-	>0	overflow
Load Positive Floating-Point	=0	-	>0	-

Program Interruptions:
Specification (LNER,LPER)
Fixed-Point Overflow (LPR,LPSR)

Procedures Referenced:
NEGATE (LNR,LNSR,LPR,LPSR)

Load Multiple	-	LM
Load Full to Short	-	LFSR
Load PSW	-	LPSW

The Load Multiple instruction loads the register range specified from the memory locations designated by the second operand address. The registers are loaded in ascending order with register 0 following register 15.

The Load Full to Short instruction loads the second operand into the first operand location.

The Load PSW instruction replaces the PSW with the double word at the location designated by the operand address.

Resulting Condition Code:

	0	1	2	3
Load Multiple	The code remains unchanged.			
Load Full To Short	=0	<0	>0	overflow
Load PSW	The code is set according to bits 34 and 35 of the new PSW.			

Program Interruptions:
Addressing (LM,LPSW)
Specification (LM,LPSW)
Privileged (LPSW)
Fixed-Point Overflow (LFSR)

Procedures Referenced:
LOAD (LM)

LOAD AND TEST INSTRUCTIONS.-

Load and Test	-	LT,LTR
Load and Test Short	-	LTS,LTSR
Load and Test Floating-Point	-	LTER

The second operand is placed in the first operand location.

Resulting Condition Code:

	0	1	2	3
Load and Test	=0	<0	>0	-
Load and Test Short	=0	<0	>0	-
Load and Test Floating-Point	=0	<0	>0	-

Program Interruptions:
   Addressing (LT,LTS)
   Specification (LT,LTER,LTS)

Procedures Referenced:
   LOAD (LT,LTS)

MULTIPLY INSTRUCTIONS.-

Multiply	-	M,MR
Multiply Halfword	-	MH,MHI
Multiply Short	-	MS,MSI,MSR
Multiply Floating-Point	-	ME,MER

The product of the multiplier (the second operand) and the multiplicand (the first operand) replaces the multiplicand. In the full word multiply, both operands are 32-bit signed integers. The product is a 64-bit signed integer and occupies an even/odd register pair.

The halfword multiplier is expanded to a full word by propagating the sign-bit and the multiplication is processed with 32-bit signed integer operands.

The short precision operands are 16-bit signed integers and the product is a 32-bit signed integer.

The normalized product of the floating-point operands replaces the first operand. The multiply operation consists of exponent addition and fraction multiplication; the sign of the product is determined by the rules of algebra. The result is normalized by prenormalizing the initial operands and postnormalizing the intermediate product. Exponent overflow and underflow are detected but the underflow interrupt may be masked off.

Condition Code:
   The code remains unchanged.

Program Interruptions:
   Addressing (M,ME,MH,MS)
   Specification (M,ME,MER,MR,MH,MS)
   Exponent Overflow (ME,MER)
   Exponent Underflow (ME,MER)

Procedures Referenced:
   LOAD (M,ME,MH,MS)
   NEGATE (M,MR)
   NEGATEDBLE (M,MR)
   ONESCOMP (MH,MHI)
   ONESCOMPS (MH,MHI,MS,MSI,MSR)
   TWOSCOMP (MH,MHI,MS,MSI,MSR)
   PRENORMALIZE (ME,MER)
   CDCREAL (ME,MER)

EXPONENTCK (ME,MER)
NSSCIIREAL (ME,MER)
NORMALIZE (ME,MER)

MOVE INSTRUCTIONS.-

Move	-	MVC,MVI
Move With Offset	-	MVO
Move Numerics	-	MVN
Move Zones	-	MVZ

The second operand is placed in the first operand location for the Move instructions. The Move With Offset instruction places the second operand to the left of and adjacent to the low-order four bits of the first operand, and the result is placed in the first operand location. The low-order four bits of each byte in the second operand field, the numerics, are placed in the low-order bit positions of the corresponding bytes in the first operand fields for the Move Numerics instruction. The Move Zones instruction places the zones, the high-order four bits of each byte, of the second operand field in the corresponding zones of the first operand field.

Condition Code:
The code remains unchanged.

Program Interruptions:
Addressing (MVC,MVI,MVO,MVN,MVZ)
Protection (MVC,MVI,MVO,MVN,MVZ)

Procedures Referenced:
PROTECTCK (MVC,MVN,MVZ)

NORMALIZE INSTRUCTION.-

Normalize - NRM

The 32 bits in the first operand are shifted arithmetically left until bit 0 is not equal to bit 1. The number of shifted bit positions is then placed into the second operand. A zero is considered to be normalized.

Resulting Condition Code:

	0	1	2	3
Normalize	=0	<0	>0	-

Program Interruptions:
None

Procedures Referenced:
CKSHIFT

OR INSTRUCTIONS.-

Or	-	O,OC,OI,IOR
Or Short	-	OS,OSI,OSR

The logical sum (OR) of the bits of the first and second operand is placed in the first operand location. All bits of the fixed-length integers are treated uniformly.

Resulting Condition Code:

	0	1	2	3
Or	=0	≠0	-	-
Or Short	=0	≠0	-	-

Program Interruptions:
- Addressing (O,OC,OI,OS)
- Specification (O,OS)
- Protection (OC,OI)

Procedure Referenced:
- LOAD (O,OS)
- LOGICAL (O,OC,OI,IOR,OS,OSI,OSR)
- PROTECTCK (OC)

PACK INSTRUCTION.-

Pack	-	PACK

The format of the second operand is changed from zoned to packed, and the result is placed in the first operand location.

Condition Code:
- The code remains unchanged.

Program Interruptions:
- Addressing
- Protection

Procedures Referenced:
- None

SUBTRACT INSTRUCTIONS.-

Subtract	-	S,SR
Subtract Double	-	SD,SDR
Subtract Halfword	-	SH,SHI
Subtract Logical	-	SL,SLR
Subtract Short	-	SS,SSI,SSR
Subtract Normalized	-	SE,SER
Subtract Unnormalized	-	SU,SUR

The second operand is subtracted from the first operand, and the difference is placed in the first operand location. Subtraction is performed by adding the twos complement of the second operand to the first operand.

The halfword second operand is expanded to a full word before the subtraction by propagating the sign-bit.

The occurrence of a carry out of the sign position is recorded in the condition code for the logical subtracts.

In the Subtract Normalized instruction, the second operand is subtracted from the first operand and the normalized difference is placed in the first operand. In Subtract Unnormalized, the unnormalized difference is placed in the first operand. The subtraction is performed by inverting the sign of the second operand and adding. The exponent underflow interrupt may be inhibited by setting the mask bit.

Resulting Condition Code:

	0	1	2	3
Subtract	=0	<0	>0	overflow
Subtract Double	=0	<0	>0	overflow
Subtract Halfword	=0	<0	>0	overflow
Subtract Logical	-	≠0,no carry	=0,carry	≠0,carry
Subtract Short	=0	<0	>0	overflow
Subtract Normalized	=0	<0	>0	-
Subtract Unnormalized	=0	<0	>0	-

Program Interruptions:

Addressing (S,SD,SE,SH,SL,SS,SU)
Specification (S,SD,SDR,SE,SER,SH,SL,SS,SU,SUR)
Fixed-Point Overflow (S,SR,SD,SDR,SH,SHI,SS,SSI,SSR)
Significance (SE,SER,SU,SUR)
Exponent Overflow (SE,SER,SU,SUR)
Exponent Underflow (SE,SER)

Procedures Referenced:

LOAD (S,SE,SH,SL,SS,SU)
LOADDBLE (SD)
LOADDREG (SDR)
MAXNUMF (S,SR)
MAXNUMD (SD,SDR)
MAXNUMS (SS,SSI,SSR)
NEGATE (SH,SHI,SL,SLR)
ADD (SH,SHI)
ADDLOG (SL,SLR)
ADDREAL (SE,SER,SU,SUR)
NORMALIZE (SE,SER)
UNNORMALIZE (SU,SUR)

START I/O INSTRUCTION.-

Start I/O - SIO

The Start I/O instruction is used to issue one of four 16-bit command words to direct output, direct input, reset interface, and test interface. The SIO instruction is implemented to decode these commands only.

Condition Code:
The code is not set by the ICS.

Program Interruptions:
Addressing
Specification
Privileged

Procedures Referenced:
None

SHIFT INSTRUCTIONS.-

Shift Left Arithmetic - SLA
Shift Left Double Arithmetic - SLDA
Shift Left Arithmetic Short - SLAS

The integer part of the first operand is shifted left the number of bits specified by the low-order six bits of the second operand. The sign of the first operand remains unchanged. Zeros are supplied to the vacated low-order bits.

Resulting Condition Code:

	0	1	2	3
Shift Left Arithmetic	=0	<0	>0	overflow
Shift Left Double Arithmetic	=0	<0	>0	overflow
Shift Left Arithmetic Short	=0	<0	>0	overflow

Program Interruptions:
Specification (SLDA)
Fixed-Point Overflow (SLA,SLDA,SLAS)

Procedures Referenced:
SHIFTL (SLA,SLAS)
SHIFTF (SLA,SLAS,SLDA)
SHIFTLD (SLDA)
CKSHIFT (SLA,SLAS)

Shift Left Logical - SLL
Shift Left Double Logical - SLDL
Shift Left Logical Short - SLLS

The first operand is shifted left the number of bits specified by the low-order six bits of the second operand. High-order bits are shifted out and zeros are supplied to vacated low-order bits.

Condition Code:
The code remains unchanged.

Program Interruptions:
Specification (SLDL)

Procedures Referenced:
SHIFTL (SLL,SLLS)
SHIFTF (SLL,SLLS)
SHIFTLD (SLDL)
CKSHIFT (SLL,SLLS)
SHIFTD (SLDL)

Shift Right Arithmetic - SRA
Shift Right Double Arithmetic - SRDA
Shift Right Arithmetic Short - SRAS

The integer part of the first operand is shifted right the number of bits specified by the low-order six bits of the second operand. The sign of the first operand remains unchanged. Bits equal to the sign are supplied to the vacated high-order bit positions.

Resulting Condition Code:

	0	1	2	3
Shift Right Arithmetic	=0	<0	>0	-
Shift Right Double Arithmetic	=0	<0	>0	-
Shift Right Arithmetic Short	=0	<0	>0	-

Program Interruptions:
Specification (SRDA)

Procedures Referenced:
SHIFTR (SRA,SRAS)
SHIFTF (SRA,SRAS,SRDA)
SHIFTRD (SRDA)

Shift Right Logical - SRL
Shift Right Double Logical - SRDL
Shift Right Logical Short - SRLS

The first operand is shifted right the number of bits specified by the low-order six bits of the second operand. Low-order bits are shifted out and zeros are supplied to the vacated high-order bits.

Condition Code:
The code remains unchanged.

Program Interruptions:
   Specification (SRDL)

Procedures Referenced:
   SHIFTR (SRL,SRLS)
   SHIFTF (SRL,SRDL,SRLS)
   SHIFTRD (SRDL)

SET MASK INSTRUCTIONS.-

Set Program Mask	-	SPM
Set System Mask	-	SSM

The Set Program Mask instruction sets the condition code and program mask bits of the current PSW from bits 2-7 of the first operand.

The Set System Mask instruction replaces the system mask bits of the current PSW with the byte at the location designated by the operand address.

Condition Code:
   Set Program Mask - The code is set according to bits 2 and 3 of the operand.
   Set System Mask - The code remains unchanged.

Program Interruptions:
   Addressing (SSM)
   Privileged (SSM)

Procedures Referenced:
   None

SET STORAGE KEY INSTRUCTION.-

Set Storage Key	-	SSK

The key of the storage block addressed by the second operand is set according to the key in the first operand. Bits 16-21 of the second operand register gives the block address; the two bit key is obtained from bits 30 and 31 of the first operand register.

Condition Code:
   The code remains unchanged.

Program Interruptions:
   Addressing
   Specification
   Privileged

Procedures Referenced:
None

STORE INSTRUCTIONS.-

Store	-	ST
Store Character	-	STC
Store Double	-	STD
Store Halfword	-	STH
Store Multiple	-	STM
Store Floating-Point	-	STE

The first operand is stored at the second operand location. The Store Character instruction places the low-order byte at the second operand address. The Store Multiple instruction stores the contents of a range of registers starting at the location designate by the second operand address. The registers are stored in ascending order with register 0 following register 15.

Condition Code:
The code remains unchanged.

Program Interruptions:
Addressing (ST,STC,STD,STE,STH,STM)
Specification (ST,STD,STE,STH,STM)
Protection (ST,STC,STD,STE,STH,STM)

Procedures Referenced:
STORE (ST,STD,STE,STH)
STOREMULT (STM)

SUPERVISOR CALL INSTRUCTION.-

Supervisor Call	-	SVC

This instruction causes a supervisor call interruption, with the operand field providing the interruption code. The interruption code is used to update the current PSW which is stored as the old program PSW; the new program PSW then becomes the current PSW. If the operand field is 3, the SVC instruction acts as a program terminator.

Condition Code:
The code remains unchanged in the old PSW.

Program Interruptions:
None

Procedures Referenced:
STOREPSW
LOADPSW

TEST BITS INSTRUCTIONS.-

Test Bits - TB
Test Bits Immediate - TBI

The state of the first operand bits selected by a mask is used to set the condition code. The second operand is used as a 16-bit mask and corresponds one for one with bits 16-31 of the first operand. A mask bit of one indicates the first operand bit is to be tested; a zero indicates the bit is to be ignored.

Resulting Condition Code:

	0	1	2	3
Test Bits	All Zeros	Mixed	-	All Ones
Test Bits Immediate	All Zeros	Mixed	-	All Ones

Program Interruptions:
Addressing (TB)
Specification (TB)

Procedures Referenced:
LOAD (TB)

Test Under Mask - TM

The state of the first operand bits selected by a mask is used to set the condition code. The byte of immediate data is used as an eight-bit mask and corresponds one for one with the bits of the character in storage specified by the first operand address. A mask bit of one indicates the storage bit is to be tested; a zero indicates the bit is to be ignored.

Resulting Condition Code:

	0	1	2	3
Test Under Mask	All Zeros	Mixed	-	All Ones

Program Interruptions:
Addressing

Procedures Referenced:
None

Test and Set - TS

The leftmost bit of the byte located at the operand address is used to set the condition code and the byte is then set to all ones.

Resulting Condition Code:

	0	1	2	3
Test and Set	Leftmost bit 0	Leftmost bit 1	-	-

Program Interruptions:
   Addressing
   Protection

Procedures Referenced:
   None

TIMER READ AND SET INSTRUCTION.-

   Timer Read/Set - TMRS

The functions performed by this instruction depend on the first operand. The real-time clock and interval timer can be read or read and set according to the value given as the first operand. This value range is 0 through 3; a 0 indicates the real-time clock is to be read; 1, read the interval timer; 2, read and set the real-time clock; and 3, read and set the interval timer. A value of 4 indicates that a checkpoint is to be taken if the real-time clock value exceeds the value stored at the effective memory address. Program execution terminates after the checkpoint is taken and will resume at this point when restarted.

Condition Code:
   The code remains unchanged.

Program Interruptions:
   Addressing
   Specification
   Privileged

Procedures Referenced:
   LOAD
   CKPOINT

TRANSLATE INSTRUCTIONS.-

   Translate - TR
   Translate and Test - TRT

The bytes of the first operand are used as arguments to reference the list designated by the second operand address. The first operand bytes are added to the second operand address to form the function byte address. In the Translate instruction, the function byte replaces the original first operand byte. This process continues until the first operand field is exhausted.

In the Translate and Test instruction, the function byte is used to determine the continuation of the operation. When the byte is zero, the operation proceeds by fetching and translating the next argument byte. When the byte is non-zero, the operation is completed by storing the argument address in register 1 and inserting the function byte in register 2.

Resulting Condition Code:

	0	1	2	3
Translate	The code remains unchanged.			
Translate and Test	All function bytes are zero	Non-zero function byte before first operand field is exhausted	Last function byte is zero	-

Program Interruptions:
   Addressing (TR,TRT)
   Protection (TR)

Procedures Referenced:
   ADDRCK (TR,TRT)
   PROTECTERR (TR)

UNPACK INSTRUCTION.-

   Unpack - UNPK

The format of the second operand is changed from packed to zoned, and the result is placed in the first operand location.

Condition Code:
   The code remains unchanged.

Program Interruptions:
   Addressing
   Protection

Procedures Referenced:
   None

CONCLUDING REMARKS

The interpretive computer simulator described in this paper has successfully been used to develop the utility library that will support the ICS applications, and for preliminary testing of the executive software designed for the ASPS project. The visibility of instruction processing and the control the ICS provides are essential when debugging and testing software code, as the experience with the ASPS executive has confirmed. These features make an ICS a most serviceable software verification and testing tool.

Langley Research Center
National Aeronautics and Space Administration
Hampton, VA 23665
February 1979

APPENDIX A

ICS SYSTEM USAGE AND SUPPORT PROCESSORS

The ICS is supported by a Meta Assembler and a Linkage Loader that reside on the CYBER system at LRC. The input file to the ICS is a load module in the format defined for the DATAF input file (See page 6). If the program to be processed by the ICS is written in NSSC-II assembly level language, the assembler assembles the program and generates an input file for the loader. The loader processes this file and generates a load module file that becomes the DATAF input file for the ICS. The ICS reads the load module file, storing the code and/or data in the appropriate memory locations, and transfers control to the memory address specified as the transfer address of the file. The ICS then executes the code, generating an output file consisting of the instruction mnemonic, the accumulated time, the program status word, the result contained in the first operand expressed as a hexadecimal and decimal value, and any error condition that exists. The assembler-loader-ICS system requires a field length of 140K to assemble, load, and execute.

The LRC Meta Assembler and Linkage Loader may be bypassed by running the assembly code on an IBM S/360 or S/370. A utility exists at LRC that reads the resulting IBM load module and rewrites it in the DATAF format required for the ICS. If the assembler and loader are bypassed, the ICS requires a field length of 77K to execute preassembled code.

A utility library has been built to support the ICS. Currently, the library has sine, cosine, and arctangent functions in both full and short precision, and a full, short, and floating-point square root function.

A program card deck and/or listing of the ICS may be obtained from:

Computer Software Management and Information Center (COSMIC)
112 Barrow Hall
University of Georgia
Athens, Georgia 30602

When contacting COSMIC, inquiries should be for the Interpretive Computer Simulator for the NASA Standard Spacecraft II.

APPENDIX A

The following control cards are required to assemble a program, link and load the object code, and execute the load module of the ICS.

Sample Control Cards

```
*NEW ASSEMBLY OF TEST PROGRAM
*ICSLIB = (SIN,COS,ARCTAN,SQRT,ETC.)
*TAPE15 = NSSCII MACHINE DEFINITION
*NSCLIB = NSSCII LIBRARY
*CONIN = PROGRAM LOAD DIRECTIVE FILE
*MODBIN = META ASSEMBLER
*NWLKBIN = LINKAGE LOADER
*LGO = ICS EXECUTE FILE
*CKPTF = CHECK POINT FILE
*CONTRLF = CHECK POINT CONTROL FILE
*=0 NO CHECK POINT, =1 CHECK POINT
*LIBTP = LOADER LIBRARY
*TAPE8 = ASSEMBLER MODULE
*DATAF = LOAD MODULE
*SYMTBLE = SYMBOL TABLE
GET,TAPE14=ICSLIB.
GET,TAPE15.
GET,TEST,NSCLIB,CONIN.
GET,MODBIN,NWLKBIN/UN=290885N,ST=CPF.
GET,LGO,CKPTF,CONTRLF,LIBTP.
FILE,TAPE8,RT=S,BT=C,FO=SQ.
LDSET,FILES=TAPE8.
LDSET,PRESET=ZERO,MAP=BS.
*ASSEMBLE TEST
MODBIN(TEST)
REWIND,TAPE8.
REPLACE,TAPE8.
REWIND,TAPE15.
REPLACE,TAPE15.
ATTACH,PASCAL,PASLIB/UN=LIBRARY.
RFL,77000.
REDUCE,-.
*LOAD TAPE8 - CREATE DATAF
LDSET,LIB=NSCLIB/PASLIB,MAP=BS.
NWLKBIN(OUTPUT,TAPE8,CONIN,DATAF,LIBTP,SYMTBLE)
*EXECUTE DATAF
LGO.
REWIND(CKPTF,CONTRLF,DATAF)
REPLACE(CKPTF)
REPLACE(CONTRLF)
REPLACE(DATAF)
EXIT.
```

APPENDIX A

The assembly and loading of this program generates the following DATAF file which in turn is executed, producing the output listed.

Sample Program

```
        OPTION LIB = ICSLIB,PREDEF,LIST(48,LIB),BOUND
TEST    START
*
*       TEST PROGRAM
*
MAIN    CSECT
        FFORM   NEG=2COMP,TRUNC
PSTART  BALR    12,0
        USING   *,12
        L       3,DATA1
        L       4,DATA2
        L       5,DATA3
        CR      3,4
        BXH     3,4,B1
        A       4,DATA1
        BU      B2
B1      S       4,DATA2
B2      ST      4,DATA3
        SVC     3
DATA1   DATA    X'189'
DATA2   DATA    2
DATA3   DATA    430
        END     PSTART
```

Sample DATAF

```
PROGRAM NAME140513142D2D2D2D2D2D
010005C05830C0265840C02A5850C02E19348634C01A5A40C0267300C01E5B40C02A
01205040C02E0A03000000000189000000002000001AE
FFFF0100
```

APPENDIX A

Sample Output

```
OPCODE BALR       TOTAL TIME   3.8500000000000E+000      PSW 0000000040000102
   R1 40000102    ( 1073742082 )
OPCODE L          TOTAL TIME   7.1500000000000E+000      PSW 0000000080000106
   R1 00000189    (        393 )
OPCODE L          TOTAL TIME   1.0450000000000E+001      PSW 000000008000010A
   R1 00000002    (          2 )
OPCODE L          TOTAL TIME   1.3750000000000E+001      PSW 000000008000010E
   R1 000001AE    (        430 )
OPCODE CR         TOTAL TIME   1.6170000000000E+001      PSW 0000000060000110
   R1 00000189    (        393 )
OPCODE BXH        TOTAL TIME   2.4420000000000E+001      PSW 00000000A0000114
   R1 0000018B    (        395 )
OPCODE A          TOTAL TIME   2.7720000000000E+001      PSW 00000000A0000118
   R1 0000018B    (        395 )
OPCODE BU         TOTAL TIME   2.9920000000000E+001      PSW 00000000A0000120
   R1 00000000    (          0 )
OPCODE ST         TOTAL TIME   3.3660000000000E+001      PSW 00000000A0000124
   R1 0000018B    (        395 )
OPCODE SVC        TOTAL TIME   4.9720000000000E+001      PSW 0000000000000000
   R1 00000000    (          0 )
```

APPENDIX B

NSSC-II INSTRUCTION SET TIMINGS

Standard Instruction Set

NAME	MNEMONIC	TYPE	CODE	USEC
ADD	AR	RR	1A	2.2
ADD	A	RX	5A	3.3
ADD HALFWORD	AH	RX	4A	3.63
ADD LOGICAL	ALR	RR	1E	2.2
ADD LOGICAL	AL	RX	5E	3.3
AND	NR	RR	14	2.2
AND	N	RX	54	3.3
AND	NI	SI	94	3.3
AND	NC	SS	D4	5.06+2.09L+ 1.65L/64
BRANCH AND LINK	BALR	RR	05	3.85+.44B
BRANCH AND LINK	BAL	RX	45	4.4
BRANCH ON CONDITION	BCR	RR	07	2.42+1.1B
BRANCH ON CONDITION	BC	RX	47	2.86+1.1B
BRANCH ON COUNT	BCTR	RR	06	2.64+1.32B
BRANCH ON COUNT	BCT	RX	46	3.63+1.1B
BRANCH ON INDEX HIGH	BXH	RS	86	8.25
BRANCH ON INDEX LOW OR EQUAL	BXLE	RS	87	8.25
COMPARE	CR	RR	19	2.42
COMPARE	C	RX	59	3.52
COMPARE HALFWORD	CH	RX	49	3.96
COMPARE LOGICAL	CLR	RR	15	2.64
COMPARE LOGICAL	CL	RX	55	3.74
COMPARE LOGICAL	CLC	SS	D5	4.84+2.53L+ 1.65L/64
COMPARE LOGICAL	CLI	SI	95	2.31
CONVERT TO BINARY	CVB	RX	4F	94.4
CONVERT TO DECIMAL	CVD	RX	4E	99.0
DIVIDE	DR	RR	1D	54.12
DIVIDE	D	RX	5D	54.78
EXCLUSIVE OR	XR	RR	17	2.2
EXCLUSIVE OR	X	RX	57	3.3
EXCLUSIVE OR	XI	SI	97	3.3
EXCLUSIVE OR	XC	SS	D7	4.07+2.09L+ 1.65L/64
EXECUTE	EX	RX	44	6.49+TARGET
INSERT CHARACTER	IC	RX	43	3.19
LOAD	LR	RR	18	2.2
LOAD	L	RX	58	3.3

APPENDIX B

LOAD ADDRESS	LA	RX	41	2.86
LOAD AND TEST	LTR	RR	12	2.2
LOAD COMPLEMENT	LCR	RR	13	2.2
LOAD HALFWORD	LH	RX	48	3.52
LOAD MULTIPLE	LM	RS	98	3.63+1.98N
LOAD NEGATIVE	LNR	RR	11	2.53
LOAD POSITIVE	LPR	RR	10	2.53
LOAD PSW	LPSW	SI	82	10.34
MOVE	MVI	SI	92	2.75
MOVE	MVC	SS	D2	4.4+1.54L+ 1.21L/64
MOVE NUMERICS	MVN	SS	D1	5.06+2.09L+ 1.65L/64
MOVE WITH OFFSET	MVO	SS	F1	6.27+3.63L
MOVE ZONES	MVZ	SS	D3	5.5+2.09L+ 1.65L/64
MULTIPLY	MR	RR	1C	34.045
MULTIPLY	M	RX	5C	34.595
MULTIPLY HALFWORD	MH	RX	4C	15.565
OR	OR(IOR)	RR	16	2.2
OR	O	RX	56	3.3
OR	OI	SI	96	3.3
OR	OC	SS	D6	5.06+2.09L+ 1.65L/64
PACK	PACK	SS	F2	7.7+4.29L
SET STORAGE KEY	SSK	RR	08	3.52
SET PROGRAM MASK	SPM	RR	04	1.76
SET SYSTEM MASK	SSM	SI	80	7.48
SHIFT LEFT DOUBLE	SLDA	RS	8F	4.73+3.52Q+ 1.76R
SHIFT LEFT SINGLE	SLA	RS	8B	3.52+.44Q+.44R
SHIFT LEFT DOUBLE LOGICAL	SLDL	RS	8D	5.06+1.76q+ 1.76r
SHIFT LEFT SINGLE LOGICAL	SLL	RS	89	3.52+.44Q+.44R
SHIFT RIGHT DOUBLE	SRDA	RS	8E	4.73+3.96Q+ 3.96R
SHIFT RIGHT SINGLE	SRA	RS	8A	3.08+.44Q+.44R
SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS	8C	3.85+3.96Q+ 3.96R
SHIFT RIGHT SINGLE LOGICAL	SRL	RS	88	3.08+.44Q+.44R
START I/O	SIO	SI	A5	9.955
STORE	ST	RX	50	3.74
STORE CHARACTER	STC	RX	42	3.19
STORE HALFWORD	STH	RX	40	3.19
STORE MULTIPLE	STM	RS	90	3.63+1.98N
SUBTRACT	SR	RR	1B	2.2

APPENDIX B

SUBTRACT	S	RX	5B	3.3
SUBTRACT HALFWORD	SH	RX	4B	3.63
SUBTRACT LOGICAL	SLR	RR	1F	2.2
SUBTRACT LOGICAL	SL	RX	5F	3.3
SUPERVISOR CALL	SVC	RR	0A	16.06
TEST AND SET	TS	SI	93	3.74
TEST UNDER MASK	TM	SI	91	3.08
TIMER READ/SET	TMRS	RS	A4	7.04+3.41(1) 7.04+6.49(2) 7.04+2.53(3) 7.04+8.91(4)
TRANSLATE	TR	SS	DC	5.72+2.09L+ 1.10L/64
TRANSLATE AND TEST	TRT	SS	DD	6.49+2.53L +1.54L/64
UNPACK	UNPK	SS	F3	8.14+4.07L

NOTE:

The standard mnemonic, OR, is a reserved word in PASCAL forcing a new mnemonic, IOR, to be adopted for use in the ICS.

B - 1 if branch is successful, otherwise 0
L - number of first operand bytes processed
N - number of registers processed
q - integral result of shift count divided by 16
Q - integral result of shift count divided by 4
r - shift count modulo 16
R - shift count modulo 4
(1) - read Real Time Clock
(2) - read/set Real Time Clock
(3) - read Interval Timer
(4) - read/set Interval Timer

Add .5 if instruction type is RX and an index register is specified.

APPENDIX B

Short Instruction Set

NAME	MNEMONIC	TYPE	CODE	USEC
ADD HALFWORD IMMEDIATE	AHI	RI	BA	3.08
ADD SHORT	AS	RX	53	2.75
ADD SHORT IMMEDIATE	ASI	RI	AA	2.2
ADD SHORT REGISTER	ASR	RR	CA	1.76
BRANCH UNCONDITIONAL	BU	RX	73	2.2
BRANCH UNCONDITIONAL REGISTER	BUR	RR	CE	1.76
COMPARE HALFWORD IMMEDIATE	CHI	RI	B9	3.3
COMPARE LOGICAL SHORT	CLS	RX	65	3.19
COMPARE LOGICAL SHORT IMMEDIATE	CLSI	RI	B5	2.2
COMPARE LOGICAL SHORT REGISTER	CLSR	RR	C5	1.76
COMPARE SHORT	CS	RX	61	2.75
COMPARE SHORT IMMEDIATE	CSI	RI	A9	2.42
COMPARE SHORT REGISTER	CSR	RR	C9	1.54
DIVIDE SHORT	DS	RX	4D	18.11
DIVIDE SHORT IMMEDIATE	DSI	RI	B0	17.56
DIVIDE SHORT REGISTER	DSR	RR	CD	17.56
LOAD ADDRESS SHORT	LAS	RX	51	2.64
LOAD COMPLEMENT SHORT REGISTER	LCSR	RR	C3	1.76
LOAD FULL TO SHORT REGISTER	LFSR	RR	0B	2.805
LOAD HALFWORD IMMEDIATE	LHI	RI	B8	2.97
LOAD HALFWORD REGISTER	LHR	RR	D0	1.87
LOAD NEGATIVE SHORT REGISTER	LNSR	RR	C1	2.09
LOAD POSITIVE SHORT REGISTER	LPSR	RR	C0	2.09
LOAD SHORT	LS	RX	74	2.75
LOAD SHORT IMMEDIATE	LSI	RI	A8	2.2
LOAD SHORT REGISTER	LSR	RR	C8	1.76
LOAD AND TEST	LT	RX	62	3.3
LOAD AND TEST SHORT	LTS	RX	52	2.75
LOAD AND TEST SHORT REGISTER	LTSR	RR	C2	1.76

MULTIPLY HALFWORD IMMEDIATE	MHI	RI	BC	15.015
MULTIPLY SHORT	MS	RX	71	8.965
MULTIPLY SHORT IMMEDIATE	MSI	RI	B3	8.525
MULTIPLY SHORT REGISTER	MSR	RR	CC	8.525
NORMALIZE	NRM	RR	CF	5.06+1.32Q+ 1.21R
AND SHORT	NS	RX	64	2.64
AND SHORT IMMEDIATE	NSI	RI	B4	2.2
AND SHORT REGISTER	NSR	RR	C4	1.76
OR SHORT	OS	RX	66	2.75
OR SHORT IMMEDIATE	OSI	RI	A6	2.2
OR SHORT REGISTER	OSR	RR	C6	1.76
SUBTRACT HALFWORD IMMEDIATE	SHI	RI	BB	3.08
SHIFT LEFT ARITHMETIC SHORT	SLAS	RS	A3	3.52+.44Q+ .44R
SHIFT LEFT LOGICAL SHORT	SLLS	RS	A1	3.52+.44Q+ .44R
SHIFT RIGHT ARITHMETIC SHORT	SRAS	RS	A2	3.08+.44Q+ .44R
SHIFT RIGHT LOGICAL SHORT	SRLS	RS	A0	3.08+.44Q+ .44R
SUBTRACT SHORT	SS	RX	72	2.75
SUBTRACT SHORT IMMEDIATE	SSI	RI	AB	2.2
SUBTRACT SHORT REGISTER	SSR	RR	CB	1.76
EXCLUSIVE OR SHORT	XS	RX	63	2.75
EXCLUSIVE OR SHORT IMMEDIATE	XSI	RI	A7	2.2
EXCLUSIVE OR SHORT REGISTER	XSR	RR	C7	1.76
TEST BITS	TB	RX	75	3.52
TEST BITS IMMEDIATE	TBI	RI	AE	2.53

NOTE:

Q - integral result of shift count divided by 4
R - shift count modulo 4

Add .5 if instruction type is RX and an index register is specified.

APPENDIX B

Double Instruction Set

NAME	MNEMONIC	TYPE	CODE	USEC
ADD DOUBLE	AD	RX	6A	6.6
ADD DOUBLE REGISTER	ADR	RR	2A	6.93
COMPARE DOUBLE	CD	RX	69	7.15
COMPARE DOUBLE REGISTER	CDR	RR	29	7.26
LOAD COMPLEMENT DOUBLE REGISTER	LCDR	RR	23	6.05
LOAD DOUBLE	LD	RX	68	4.95
LOAD DOUBLE REGISTER	LDR	RR	28	6.05
SUBTRACT DOUBLE	SD	RX	6B	6.82
SUBTRACT DOUBLE REGISTER	SDR	RR	2B	6.93
STORE DOUBLE	STD	RX	60	6.05

NOTE:

Add .5 if instruction type is RX and an index register is specified.

APPENDIX B

Floating-Point Instruction Set

NAME	MNEMONIC	TYPE	CODE	USEC
ADD NORMALIZE	AER	RR	3A	28.3
ADD NORMALIZE	AE	RX	7A	28.74
ADD UNNORMALIZE	AUR	RR	3E	25.135
ADD UNNORMALIZE	AU	RX	7E	25.685
COMPARE	CER	RR	39	20.57
COMPARE	CE	RX	79	21.01
DIVIDE	DER	RR	3D	54.4+1.65N
DIVIDE	DE	RX	7D	54.84+1.65N
HALVE	HER	RR	34	12.265
LOAD AND TEST	LTER	RR	32	5.06
LOAD COMPLEMENT	LCER	RR	33	5.5
LOAD NEGATIVE	LNER	RR	31	5.5
LOAD POSITIVE	LPER	RR	30	5.06
LOAD	LER	RR	38	3.41
LOAD	LE	RX	78	3.52
MULTIPLY	MER	RR	3C	39.82+1.65N
MULTIPLY	ME	RX	7C	40.37+1.65N
STORE	STE	RX	70	4.51
SUBTRACT NORMALIZED	SER	RR	3B	29.18
SUBTRACT NORMALIZED	SE	RX	7B	29.51
SUBTRACT UNNORMALIZED	SUR	RR	3F	25.945
SUBTRACT UNNORMALIZED	SU	RX	7F	26.055

NOTE:

N - no. of digits of prenormalization required

Add .5 if instruction type is RX and an index register is specified.

REFERENCES

1. IBM NASA Standard Spacecraft Computer II Principles of Operation.
   IBM Number 7935402. Contract Number NAS8-32808.

2. IBM System/360 Principles of Operation.
   Ninth Edition, November 1970. GA22-6821-7.

3. Thornton, J. E.: Design of a Computer - The Control Data 6600.
   Scott, Foresman and Company, 1970.

4. Struble, George: Assembler Language Programming: - The IBM System/360.
   Addison - Wesley Publishing Company, 1971.

End of Document