COPY # 237

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

GUIDANCE AND NAVIGATION

E-1077

PRELIMINARY MOD 3C PROGRAMMERS MANUAL

by

R. Alonso
J.H. Laning, Jr.
H. Blair-Smith

November 1961

INSTRUMENTATION LABORATORY
CAMBRIDGE 39, MASSACHUSETTS

ACKNOWLEDGMENT

This report was prepared under the auspices of DSR Project 55-191 sponsored by the Space Task Group of the National Aeronautics and Space Administration through Contract NAS9-153.

TABLE OF CONTENTS

PRELIMINARY MOD 3C PROGRAMMERS MANUAL

INTRODUCTION

This manual is intended for those people who have some familiarity with the type of computer of which the Mod 3C is an example. Although the 3C computer is like the one described in R-276, there are many differences of details and of nomenclature.

The material presented here neither is complete, nor is it entirely firmly established at the present date. It is meant to provide prospective 3C programmers with enough information to code representative programs, to make estimates of storage and speed requirements, and to aid the 3C designers by forcing commitments as to the nature of the IN-OUT system. Furthermore, it is hoped that this manual will bring about comments and questions by those who program. Despite the foregoing remarks which suggest that the design of 3C is still in a fluid state, it should be emphasized that there is no reason at present to suppose that the material described in this report will be changed in any respect in the final design.

A complete listing of all registers in Groups SC, IO, and C (Table 1) and their tentative address assignments is included in Appendix I. Appendix II contains a listing of control pulses, and Appendix III shows all the control pulse sequences. Appendices IV, V, and VI describe, respectively, the Yul system compiler for compilation of 3C programs and preparation of corresponding rope wiring diagrams, a set of interpreted instructions for extra precision arithmetic and vector operations, and finally a representative program in Yul system language for purposes of illustration.

MACHINE ORGANIZATION

For programming purposes, 3C may be outlined as in

Table I. Other relevant facts are:

a. Bit positions in the 3C word are numbered 0 to 15, reading from right to left. Bit zero is always the parity bit whenever a parity bit exists. Bit 1 is the lowest order digit position and bit 15 the highest. Data words consist of a sign bit (15), 14 bits of data (bits 14-1), and parity bit 0.

b. Number system is ONE's complement. This means that there are two representations of zero.

c. Instruction format consists of 3 bits for instruction code (bits 15-13), and 12 bits for the relevant address (bits 12-1).

d. List of Instructions:

TC	Transfer Control
CCS	Count, Compare and Skip (a kind of branch)
INDEX	Modify Next Instruction
XCH	Exchange
CS	Clear and Subtract
TS	Transfer to Storage
AD	Add and Count on Overflow
MP	Multiply

e. Additional Sequences:

Increment Counters
Interrupt
Resume

Several other operations on a word are possible by the use of specially wired registers. The present content of a register, e. g. register A, is symbolized by

c(A),

and it means "that which would be read out of A". It is sometimes

TABLE I Programmer's View of 3C Organization

Group Name*	
F	Fixed: 3584 Registers Octal Addresses 7777 to 1000, inclusive
E	Normal Erasable: 460 Registers Octal Addresses 0777 to 0064, inclusive
C	Counters: 20 Registers Octal Addresses 0063 to 0040, inclusive
IO	IN-OUT: 16 Registers (6 Input, 4 Output, 4 Unassigned Addresses, and Inhibit Interrupt) Octal Addresses 0037 to 0020, inclusive
SC	Specials and Centrals: 15 Registers Octal Addresses 0017 to 0001, inclusive
A	Accummulator A: 1 Register Octal Address 0000

* Group Name is for convenience in writing the text.

necessary to make a distinction between the present content of a register, and the content of a register before some action or operation. The "before" content is symbolized by

$$b(A).$$

The registers mentioned in the description of instructions are:

A	The Accummulator
Q	The Return Address Register
N	The Uncorrected Sum Register
LP	The Low Order Product

These registers are all addressable as ordinary registers in storage and have addresses and properties as given in Appendix I.

DESCRIPTION OF THE INSTRUCTIONS

When discussing an instruction, it is assumed that L is the location of that instruction.

Code 0. TC x Transfer Control

Action: Take the next instruction from location x, instead of from location L + 1. Set central register Q to

$$c(Q) = L + 1 = TC\ L + 1$$

The last equality holds because the numerical code for TC is 000.

Comments: The instruction TC Q is useful for exiting subroutines. The instruction TC A causes the single instruction in register A to be executed. This action follows because register A has address 0 and register Q has address 1.

Code 1. CCS x. Count, Compare and Skip

Action: If c(x) > 0, set c(A) = c(x) - 1;
Take next instruction from L + 1.

If c(x) = + 0, set c(A) = + 0;
Take next instruction from L + 2.

If c(x) < 0, set c(A) = - c(x) - 1;
Take next instruction from L + 3.

If c(x) = - 0, set c(A) = + 0;
Take next instruction from L + 4.

Comments: The instruction CCS A is permissible. Note, however, that the original contents of A are changed as per the actions described above.

Code 2. INDEX x. Modify Next Instruction by Adding c(x)

Action: Take as the next instruction c(L + 1) + c(x).

Comments: If L + 1 is of Group E, its contents are not correctly restored. Address x may be of Group E without ill effects.

The sum c(L + 1) + c(x) is the overflow-corrected sum. (See Note 3 of Appendix I).

Code 3. XCH x. Exchange c(x)

Action: Set c(A) = b(x)
Set c(x) = b(A)
Exchange c(A) with c(x), unless x is of Group F. If it is, c(x) remains undisturbed.
Take next instruction from L + 1.

Comments: There are implications to the use of XCH C, where C is a counter of Group C. This will be elaborated upon later.

Code 4. CS x. Clear and Subtract x

Action: Set c(A) = - c(x);
Restore x to the original c(x). [c(x) = b(x)].
Take next instruction from L + 1.

Comments: The instruction CS A complements c(A); i.e., c(A) = - b(A)
Beware of CS x, where x is of Groups I0 or SC; restoring the contents of a register implies first clearing that register, and then writing back into it. If the register x is a shift register, for example, its contents will be altered in the process of restoration. This warning applies to all instructions.

Code 5. TS x. Transfer to Storage x

Action: Set c(x) = c(A);
Leave c(A) undisturbed.

Code 6. AD x Add and Count Overflow

Action: Set c(A) = b(A) + c(x), corrected for overflow. Superpose the uncorrected sum onto c(N). N is of Group SC.
If overflow occurs, set a signal to cause incrementing or decrementing of the overflow counter (OVCTR) of Group C, according to whether the addends were positive or negative.

Comments: Superposing onto c(N) means storing in N the bit-by-bit OR of the sum c(A) + c(x), not corrected for overflow, and the previous content of N, i.e., $c(N) = b(N) + \{c(A) + c(x)\}$.
As will be shown later, N can be used for editing purposes. The counter

incrementing feature is useful in double precision subroutines. TS N may be used to set N to some desired initial value. See Appendix I, Note 3 for definition of overflow correction.

Code 7. MP x. Multiply by x

Action: Set c(A) and c(LP) to b(A). c(x), with LP holding the low order part of the product, and A the high order part. Register x is restored. The quantity in LP has the same sign as that in A.

Comments: Multiplication requires 16 Instruction Times, versus 2 for CCS, and 1 for all others.

ADDITIONAL SPECIAL & CENTRAL REGISTERS

There are 15 registers classified as Special or Central. The Special registers are those which perform some sort of transformation or manipulation of the incoming or outgoing bits. Shift and Cycle registers are of this sort, as well as the N register; the N register has its sign bit position connected to a special place in the Adder circuit, rather than to the normal sign bus. Non-special registers are those which are like erasable registers in that they do not modify the information transmitted through them in any way.

The category of Central registers - which is not mutually exclusive with that of Specials - refers to those registers which are addressed directly by the Sequence Generator, as well as by the regular addressing method. For example, register CYL, which cycles a word left by one bit position, is Central because it is used by the MP instruction, and clearly Special because of the tranformation it effects on a word. The programmer may use freely those Specials which are not Centrals; that is, those

special registers which are not committed to any particular sequence or instruction. These registers are:

SR1	Shift Right
SL1	Shift Left
CYR	Cycle Right
EDH	An editing register described in Appendix I. and meant for use in the Interpretive set of subroutines.

The programmer may also use certain other of the special registers such as CYL, or LP, at his own discretion, provided not intervening instructions alter their contents. Other central registers, such as D or Z, should be used only by the programmer who has full knowledge of the specific effects of the control pulse sequences. These registers are discussed in detail in Appendices I and III.

Several Central registers, not Special, are worthy of note because of the use they are put to. These are:

D Holds the data address of an instruction. Programmers stay away.

Z At the point of beginning an instruction at L, c(Z) = L + 1. Stay away.

Q Holds P + 1, where P is the location of the TC instruction last executed. Used in exiting subroutines, as is shown in later examples.

BI Holds a replica of what register B held at the time of an interruption. B is not an addressable register. BI should not be disturbed unless the programmer is trying to resume from an interrupting program to another program different from the original interrupted one. This sort of thing is to be done with extreme caution.

ZI Holds a replica of what Z held at the time of an interruption. Further comments as in BI.

OUTPUT REGISTERS

There are four addresses currently reserved for output registers. The output registers are OUT0, OUT1, OUT2, and OUT3. Each output register consists of 15 bistable latches corresponding to the 15 useful bit positions of the word length and are addressed by executing TS OUTx; those latches will be turned on which correspond to ONEs in the word originally in A. Hence, each latch is addressed by bit position within a word, as well as by word address.

Register OUT0 and OUT1 are such that their latches, once turned on, stay on until a new word is written into them. The useful output signal is a DC level. To turn off all the latches of, say, OUT1, execute TS OUT1, (or XCH OUT1), where c(A) = +0.

Register OUT2 is discussed after Register OUT3.

Register OUT3 is different from the previous two in that all of its latches are connected to the circuitry associated with the Group C registers (the Counters), so that a given latch within OUT3 may be turned on by the overflow of a specific counter, as well as by the execution of TS OUT3. Furthermore, these latches are turned off by every time pulse 6*, so that these latches are never on for longer than a fraction of an Instruction Time; the time these latches remain on is 30 μsec if they were turned on by an overflow; about 10 μsec if turned on by a TS OUT3. The XCH instruction should not be used for addressing either OUT3 or OUT2.

The reason for the complicated arrangement for OUT3 is that the latches are meant to be primarily the overflow (underflow) indicators for counters. The feature of addressing these overflow

*The basic Instruction Time unit has 8 steps, called Time Pulses.

indicators by means of programming was tacked on as an afterthought. It is thought that the ability to simulate overflows will aid in systems and computer tests. The brief duration of these outputs is a logical requirement of the Counter Incrementing and Priority Interrupt systems.

Register OUT2 is like OUT0 and OUT1 in that the output latches can be turned on only by means of a TS instruction. For the most part, these latches will stay on until turned off by an instruction, as in OUT0 and OUT1. Some bit positions, however, may have connections which turn those latches off at every Time Pulse 6, exactly as in the case of OUT3 latches.

There are a total of 60 output latches in 3C. Attempts to read information from an OUTx register, e.g., by means of XCH OUTx or CS OUTx, will result in c(A) = 0; CS OUTx will also result in all the latches of OUTx being turned off.

INPUT REGISTERS

There are six input registers in 3C, IN0 through IN5. These input registers are very like sampling and storing devices, and experience with Mod 1B has shown them to be a source of mild confusion.

Input registers consist of magnetic cores, each of which is tied to some DC source, e.g., a toggle switch, or an output latch. An instruction such as XCH INx first clears those cores to ZERO, sensing their contents into A, and then samples the state of the DC sources; the state of the cores of INx is known to be like the state of the sampled DC sources only after the execution of XCH INx. What this means is that the first XCH INx does not transfer into A the present state of the sampled DC sources, but the state those sources were in the previous time INx was addressed. Prudent practice, then, calls for interrogating input registers by means of two successive orders XCH INx, or equivalent.

Registers IN0, IN1, and IN2 are tied, one to one, with the output latches of registers OUT0 , OUT1, and OUT2. In this way it is possible, by means of programs, to check on the state of output latches. The short duration of the outputs from Register OUT3 makes it impractical to tie an Input Register to it. OUT2 is connected to IN2 because it is anticipated that most of its latches will stay on as do those of OUT0 and OUT1, and hence it is desirable to be able to check the state of those latches.

The remaining registers IN3, IN4, and IN5 are to be connected to the outside world. The input convention is that a ONE is a grounded input line i. e., the DC source supplies a ground to indicate a ONE, and an open or -10v through high impedance when indicating ZERO.

INx registers may not be written into directly by way of programs; it is not possible to transfer c(A) into INx by means of TS INx or XCH INx. Those two instructions will, however, cause the state of the input lines to be sampled into INx.

COUNTER INCREMENTING

Mod 3C has 20 addressable registers which behave as counters. The inputs to those counters are things such as accelerometer pulses, or pulses from a clock, or overflows from other counters. Specification of the inputs to counters is a matter of wiring, not of program, and is one of the desired goals and a necessary condition for the full definition of 3C. Counter incrementing takes place between the end of one instruction and the beginning of the next one. This process requires one Instruction Time per increment executed; this is one reason why statements about time of execution of programs must be qualified.

As presently planned, positive or negative input into a counter position causes the action c(K) = b(K) ± 1(o), where 1(o) stands for low order ONE. Every input into a counter position specifies a counter. The overflow or underflow of counters may be used as outputs to be connected to the outside world; or

to serve as inputs to other counters; or to the Priority Interrupt circuits. It is part of the programmer's job to specify such connections when he needs them.

To interrogate a counter without risk of missing input pulses, use CS CTR and not XCH CTR. To preset a counter to a given value, and not miss pulses, use XCH CTR, not TS CTR. The subtleties which cause these recommendations will be explored later in examples.

Table II Preliminary Counter List

Counter No.	Octal Address	Name	Comments
0	0040	OVCTR	Used in connection with AD x
1	0041	Time 1	Low order part of time
2	0042	Time 2	High order part of time
3	0043	Time 3	Presettable counter. See Example II.

A list with a tentative assignment of the first four counters is shown in Table II. The inputs to TIME 1 are pulses from a clock. For the sake of definiteness, assume that these pulses appear at 10 m sec intervals. Counter TIME 1 can store 2^{14} such pulses, or about 160 sec. The overflows from TIME 1 are the inputs to TIME 2. TIME 3 also has input pulses which appears once every 10 m sec. Overflows from TIME 3 cause automatic interruption of whatever program is then being executed, and transfer of control to another program. TIME 3 can be preset to any desired value, so that it may be used to mark off time intervals in multiples of 10 m sec; this method of marking time intervals, which may differ one from the next, is useful in that it avoids clumsy programs for finding out whether it is time to perform some action or not. See Example II.

PROGRAM INTERRUPTION

In general, a program may be interrupted by the occurence of certain external signals. This means that the normal sequence of instructions of a program may be broken into at any point, and that control is transferred to some other program. There is a short subroutine which has the net effect of returning control to the original (interrupted) program, with no loss of information if certain precautions are taken. It would be expected, for example, that the signalling of the computer from the control console by manual intervention would take place through an interrupt operation. In other words, the depressing of the button or turning of the switch on the control console could signal the computer that information was to be read, and the information itself then taken from one of the input registers.

An interrupting program is prevented by interlock logic from itself being interrupted. It is the responsibility of the interrupting program to preserve the contents of registers Q and A, and to restore them to their original content. The content of register Z is preserved automatically. The interrupting program must also restore any other Central or Special register it uses back to its original state. Register A is mentioned specifically because it is necessarily used by an interrupting subroutine. Register Q must usually be preserved, since interrupting subroutines will usually have a TC instruction.

The specific point at which the interrupted program is resumed is when the instruction TS RIP is executed; the content of A may be anything at all; RIP is a specific address (octal 0036), and the mnemonic code stands for "Resume Interrupted Program".

A specific format has been decided upon for the Interrupt System. These are eight Interrupt options available; i. e., all signals which cause an Interrupt belong to one of eight categories,

OPTION 1 through OPTION 8. The first and immediate result of some such signal is to preserve B and Q, and to transfer control to one of eight subroutines named OPTION 1 through OPTION 8. These routines are located in octal addresses 1000 through 1037; for example:

Mnemonic Address	Absolute Address		
OPTION 1	1000	TS	AI
	1	XCH	Q
	2	TS	QI
	1003	TC	INTPNG 1
OPTION 2	1004	TS	AI
	3	XCH	Q
	4	TS	QI
	1005	TC	INTPNG 2
	etc.		

These programs preserve c(A) and c(Q), and transfer control to the appropriate interrupting program INTPNG 1, 2, etc. Registers AI and QI are in E; since there can be no interruption of an interrupt, AI and QI are the same for all options.

The particular actions described above may seem round-about, since it can be arranged for an interruption to transfer control directly to INTPNG X. The reason for first transferring control to an OPTION X is to make definite the wiring of the sequence generator, since each OPTION X now has a definite numerical address associated with it. These addresses are in F; the rope wiring is still left indefinite, of course.

The last act of the INTPNG X program is to transfer control to the RESUME subroutine.

```
RESUME    XCH    QI
          TS     Q
          XCH    AI
          TS     RIP
```

The last instruction causes transfer of control back to the interrupted program.

It is sometimes necessary to guarantee that an interruption will not occur during a certain part of a program, as is shown in Example I. An address has been reserved for this purpose and the instruction TS INHINT, where c(A) < 0, will inhibit interrupt. Removal of the inhibition takes place by executing another TS INHINT, where c(A)> 0. Address INHINT happens to be octal address 0037.

EXAMPLES OF PROGRAMS

Example I. Modification of a Single Bit of an Output Register

Let OUT1 (whose latches are turned ON and OFF by program) be in some uspecified state. Let it be desired to turn to ON bit 5 of OUT1. All other latches must be left in their original state. We may not assume that bit 5 was OFF to being with. Recall that IN1 is tied to OUT1, and the present state of OUT1 may be known by interrogating IN1. The logical problem is then simple: Transfer into OUT1 a word which has a ONE in bit position 5, and ONEs and ZEROs elsewhere as before. If it is known that bit 5 was OFF, then such a word could be obtained by the following program:

```
XCH     IN1,
XCH     IN1,
AD      "Bit 5",   c ("Bit 5") = 000 000 000 010 000
TS      OUT1.
```

If latch 5 of OUT1 were ON, however, such a program would result in a ZERO sum in position 5, and a carry into the higher order bits. What is desired, then,is the OR operation, rather than AD. A particular way of doing this is to blank the content of A as per the content of some register Bk, where blanking means "make ZERO those bit positions of A which correspond to ONEs of Bk. " Once a word is blanked in the desired hit positions, the OR operation

is achieved by the AD instruction.

A blanking subroutine can be as follows, making use of the special register N (see description of AD x).

```
"Blank c(A) by c(Bk)"    CS      A,      Invert c(A)
                  +1     XCH     N,
                  +2     XCH     ZERO    c(ZERO) = 0
                  +3     AD      Bk
                  +4     CS      N,
                  +5     TC      Q
```

The first two instructions result in placing -b(A) in N. XCH ZERO clears A. AD Bk places c(Bk) on the word already in N. This word is the inverse of what is desired. Return of control to the calling subroutine is accomplished by TC Q. This subroutine is useful in turning output latches OFF as well as ON.

Turning on bit 5 of OUT1 is then done as follows:

```
Latch ON    XCH     IN1
      +1    XCH     IN1
      +2    TC      Blank A by "Bit 5",   c(Bk) = all
                                          ZEROs except
                                          for a ONE in
                                          bit 5.
      +3    AD      "Bit 5",
      +4    TS      OUT1.
```

If it is desired to check that all the latches are now in their proper state, the following program might be used.

```
Latch ON +5  TC  CHECK OUT1,
```

and

```
CHECK OUT1      XCH     Temp1       Preserve c(A)
          +1    XCH     IN1
          +2    XCH     IN1
          +3    CS      A
          +4    AD      Temp 1
          +5    TS      Temp 2      Preserve difference
          +6    CCS     Temp 2
          +7    TC      WRONG       Too few latches ON.
          +8    TC      Q           Ok exit
          +9    TC      WRONG       Too many latches ON.
         +10    TC      Q           Ok exit
```

Further routines, remedial or diagnostic, could follow a failure of the above check. If the check succeeds, TCQ causes control to be transferred to "Latch ON +6".

One cause for concern in the above programs might be if the program LATCH ON is interrupted, say after AD "Bit 5", and the interrupting program changes the state of some other latch of OUT1. Upon resumption of LATCH ON, the selection of latches to be turned ON (or OFF) with TS OUT1 is based on the sampling of OUT1 which occurred before the interruption. Thus, as far as OUT1 is concerned, the latches will all be restored to the state they were at before the interruption, excepting for bit 5. This situation can be remedied by inhibiting interruptions for a brief period. Interruptions may be inhibited by the instruction TS INHINT, where c(A) < 0 or c(A) = - 0; i. e., the sign bit position of A contains a ONE. To again allow interruptions, it is necessary to do TS INHINT with the sign bit of A at ZERO. Hence, program LATCH ON should be preceded and succeeded as follows:

```
LATCH ON    -2      CS PLUS     Puts sign bit of A at ONE,
                                if c (PLUS) <0
            -1      TS INHINT
LATCH ON            ETC.
LATCH ON    +6      CS MINUS    c(MINUS) > 0
            +7      TS INHINT
```

There is no need for such precautions if the LATCH ON program is part of an interrupting program.

Example II. Use of Counters

Let the overflow of TIME 3 cause control to be transferred to INTPNG 1. For present purposes, let it be desired that this occur once per second. Since TIME 3 is incremented once every 10^{-2} sec, it should overflow every 100 pulses (octal 144). The largest positive number a register can hold is, in octal, +37777; hence, if upon overflowing TIME 3 is set to +37634, it will overflow 144 (octal) pulses later. Let c (SET) = +37634; then part of the interrupting program is

```
INTPNG 1      XCH    SET
        +1    TS     SET
        +2    XCH    TIME 3
              etc.
EXIT          TC     Resume.
```

In this case the time elapsed between overflow and the execution of INTPNG 1 + 2 is 8 instruction times, counted as follows:

Interrupt Sequence	1 IT
Option 1 Program	4 IT
INTPNG to INTPNG + 2	3 IT.

This time is 320 μsec; hence, there is little fear* of having missed a pulse between overflow and resetting. This, in turn, means that the next overflow will occur exactly 100 (decimal) pulses later, and not 101 pulses later. Upon overflow TIME 3 is left at ZERO.

There may be circumstances in which the time elapse between overflow and resetting of TIME 3 is so great as to make it possible to miss a count. This hazard may be avoided by the program shown below.

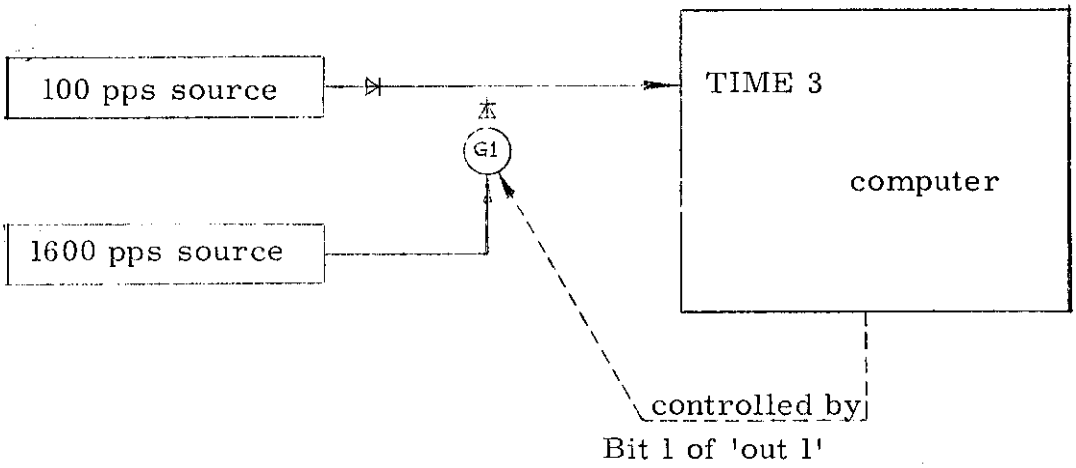
*Assume there are few or none intervening counter increments being performed.

```
INTPNG 1    ---                 Some unspecified action
            ---
            ---
      +M    XCH     SET
            TS      SET
            XCH     TIME 3
            TS      TEMP        TEMP is in E
            CCS     A
            TC      AD          TIME 3 had counted something
                                before being set.
            TC      CONTINUE    it had not; continue.
            TC      WRONG       These two possibilities can only
                                occur if there has been an error.
      AD    XCH     ZERO        Clear A
      +1    XCH     TIME 3
      +2    AD      TEMP
      +3    XCH     TIME 3      Set TIME 3 again, and again bring
                                its previous contents out for
                                examination.
      +4    TC      M + 3       Check again that no pulses have
                                been missed.
CONTINUE    ---                 Other actions of interrupting progran
            ---
      EXIT  TC      RESUME
```

The program checks if TIME 3 was incremented before being set. The XCH order is such that the actual exchange of c (A) and c (register) takes place between Time Pulses 4 and 5, while counter incrementing does not take place until Time Pulse 8. Thus the program guarantees that no pulses will be lost. Unfortunately, this guarantee may be at the expense of an endless loop, since the AD subroutine transfers control back to the start of the checking program to see if any pulses were added between AD + 1 and AD + 3; hence, a sufficiently fast input pulse rate could cause such an endless loop. This checking program

is probably most useful in circumstances where input pulse rates are slow, but where there is reason to think an initial pulse might have been missed.

One way to use the INTPNG programs is to have them execute a TC to register CHOICE, where CHOICE is in E. The content of CHOICE can be set by the INTPNG program itself or by some other program. In this way, INTPNG 1 does different things at different times. As an example, let it be required that TIME 3 overflows twice every 10 m sec: once at the 10 and once at the 10.625 m sec mark. Assume that a ONE in bit 1 of OUT1 connects to TIME 3 a pulse source which provides a pulse every .625 m sec, or 16 times faster than TIME 3's normal pulse source of one pulse per 10 m sec. This latter pulse source is not disconnected from TIME 3 since G1 (Fig.1) is ON but a small portion of the time, and never when the 100 pps source is active.



G1 is a gate which allows fast pulses into TIME 3 if ON.

Fig. 1

The subroutines CHE 1 and CHE 2, shown below, are straight forward examples. The method of turning G1 ON and OFF is good only if the state of G2 is known, in the sense that being in CHE 1 implies that G 1 is OFF, and being in CHE 2 implies G 1 is ON.

```
INTPNG 1        TC      CHOICE
                TC      ALARM
        c(choice) is guaranteed to be either TC CHE 1 or TC CHE 2
CHE 1           XCH     IN 1
                XCH     IN 1
                AD      BIT 1       BIT 1 is a word in F
                TS      OUT 1       TURN ON G 1
                XCH     SET 1       SET 1 is a word in F
                XCH     TIME 3
                XCH     C 2         C 2 is a word in F
                XCH     CHOICE
                TC      RESUME
CHE 2           XCH     IN 1
                CS      IN 1
                AD      BIT 1
                CS      A
                TS      OUT 1       TURN OFF G 1
                XCH     SET 2       SET 2 is a word in F
                XCH     TIME 3
                XCH     C 1         C 1 is a word in F
                XCH     CHOICE
                TC      RESUME
C (BIT 1 )  = 00001
C (SET 1) = + 37777
C (SET 2) = + 37634
C (C 1 )    =   T C CHE 1
C (C 2)     =   T C CHE 2
```

APPENDIX I

COMPLETE LIST OF CENTRALS & SPECIALS WITH NUMERICAL ADDRESS AND BIT TRANSFORMATIONS

APPENDIX I COMPLETE LIST OF CENTRALS & SPECIALS WITH NUMERICAL ADDRESS AND BIT TRANSFORMATIONS

NAME	Octal Address	PURPOSE OR COMMENTS	TRANSFORMATION (See Note 1) Register Bit Position vs Write Buss															Central	Special
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1		
Q	001	Return Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	x	
D	002	Data Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	x	
Z	003	Next Instruction Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	x	
	004	Normal Register	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1		
N	005	Overflow Detection and OR operation Used in ADx	US	14	13	12	11	10	9	8	7	6	5	4	3	2	1	x	x
SR	006	Shift Right	15	15	14	13	12	11	10	9	8	7	6	5	4	3	2		x
SL	007	Shift Left	15	13	12	11	10	9	8	7	6	5	4	3	2	1	15		x
BI	010	Keep c(B) during Interrupts	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	x	
CYL	011	Cycle Left	14	13	12	11	10	9	8	7	6	5	4	3	2	1	15	x	x
EDH	013	Edit - Used in Interpretive Sub-routines	9	-	-	-	-	-	-	-	-	-	14	13	12	11	10		x
CYR	012	Cycle Right	1	15	14	13	12	11	10	9	8	7	6	5	4	3	2		x
CLHP	014	See Note 2 for Comments	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x
HP	015	High Order Register - Used in MPx	15	US	14	13	12	11	10	9	8	7	6	5	4	3	2	x	x
LP	016	Low Order Product - Used in MPx	1	-	14	13	12	11	10	9	8	7	6	5	4	3	2	x	x
ZI	017	Keep c(Z) During Interrupts	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	x	

NAME	Octal Address	PURPOSE OR COMMENTS	TRANSFORMATION (See Note 1) Register Bit Position vs Write Buss
IN 0	0020	Tied to OUT 0	(Addresses 20 and greater have the standard one-to-one transformation)
IN 1	0021	Tied to OUT 1	
IN 2	22	Tied to OUT 2	
IN 3	23	From outside the computer	
IN 4	24	From outside the computer	
IN 5	25	From outside the computer	
OUT 0	26		
OUT 1	27		
OUT 2	0030		
OUT 3	31		
	32	Spare addresses	
	33		
	34		
	35		
RIP	0036	Resume Interrupted Program	
INHINT 7	0037	Inhibit Interrupt if write a ONE into this word, in sign position. Uninhibit Interrupt if write ZERO.	

Notes for Appendix I:

1. Register bit positions are named, as are the 3C word bit positions, 0 through 15. Bit position 0 is the parity position, bit position 14 is the most significant digit position, and 15 is the sign position.

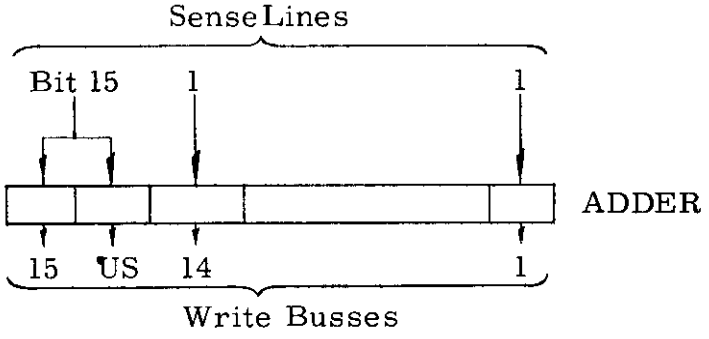
   The Write Busses are 19 in number, named 0 through 15, US, OFW and UFW. Write Busses 0 through 15 correspond to the normal word bit arrangement. Write Buss US is for the Uncorrected Sign (i.e., the sign of an addition, not corrected for overflow; see AD x instruction). Write Buss OFW is for overflow and UFW for underflow. Both of these are part of the Counter Incrementing System, and are not connected to either Central or Special Registers. Buss US is connected to two such registers, as is shown in the accompanying table.

2. Addressing CLHP has the net effect of clearing register HP without restoring anything to it. The register which would normally be associated with CLHP is not connected to that address. It is called B14, and it is not independently addressable. B14 behaves as follows:

   if LP is cleared, B14 is cleared;
   if HP is written into, B14 is written into.

   Registers B14, HP and LP are used in MP x.

3. Definition of overflow correction.
   When two numbers are entered into the Adder circuits, the sign bit of each is repeated as shown below.



The addition performed is that of two 16 bit binary numbers, corresponding each to the original 15 bit binary number, with the sign bit repeated. The resulting sum is a 16 bit number. If the two higher order bits are alike, no overflow or underflow had occurred. If they are different, 01 represents an overflow and 10 an underflow. Positive sign is represented by ZERO in the ONE s complement, number system. The overflow corrected sign is the highest order bit; ZERO in the case of overflow, ONE in the case of underflow. This means that the sign of a sum which overflowed is the same as that of the two numbers summed. The Uncorrected Sign is the next to highest order bit, and the Write Buss associated with this sign is called US. (See Note 1).

APPENDIX II
LIST OF CONTROL PULSES

APPENDIX II

LIST OF CONTROL PULSES

Conventions

CL	means clear register to all ZEROS.
W	means write into register whatever is now present on Write Busses.
ALPHA	samples sense lines of alpha side.
BETA	samples sense lines of beta side.

Control Pulses Having To Do With S & C Registers

CL A, WA		CL B, WB
CL BI, WBI	CL Q, WQ	CLD, WD
CL Z, WZ	(No CL N), WN	CLCYL, WCYL
CL HP, WHP	CLLP, WLP	CLZI, WZI
CL C, WC	CLP1, WP1	CLP2, WP2
CLSQ, WSQ	CLS, WS	CLSUM, WX, WY

Other Control Pulses

TP	Test Parity.
CLE	Clear Memory, F, E, IO, C, or S & C.
WE	Write into memory register last cleared.
CI	Carry into bit 1 of adder.
CI 2	Carry into bit 2 of adder.
BR 1, BR2, BR3	Branching options for SQ.
II	Inhibit interrupt for one instruction time.
00001	Read this number into Write Buss latches.
All 1's	Same.
77776	Same.
CL SN	Clear S, but allow no rope currents to flow.
INH	Inhibit interrupt until further notice.
ENI	Enable interrupt.
CLI	Advance priority.
CL CTR 1	Clear counter, first time.
CL CTR 2	Clear counter, second time.
W CTR	Write counter.

APPENDIX III
3C INSTRUCTIONS

DESCRIPTION OF MODEL 3C INSTRUCTIONS

MODEL 3C HAS EIGHT INSTRUCTIONS... TRANSFER CONTROL, COUNT COMPARE AND SKIP, INDEX, EXCHANGE, CLEAR AND SUBTRACT, TRANSFER TO STORAGE, ADD, AND MULTIPLY. THE SEQUENCES ADD ONE TO COUNTER, SUBTRACT ONE FROM COUNTER, INTERRUPT, AND RESUME ARE BUILT IN.

THE STANDARD QUANTUM OF TIME IS THE INSTRUCTION TIME, OR I-TIME. ONE I-TIME, OR 1 I, IS 8 PULSE TIMES, SINCE THE SHIFT REGISTER IS 8 STEPS LONG. ALL INSTRUCTIONS AND SEQUENCES TAKE 1 I, EXCEPT COUNT COMPARE AND SKIP, WHICH TAKES 2 I, AND MULTIPLY, WHICH TAKES 16 I.

SEQUENCES ARE SELECTED BY A DIODE DECODING MATRIX OPERATED BY THE SQ REGISTER. THE LATTER CONTAINS THE TOP THREE BITS OF AN INSTRUCTION WORD AND THREE MORE BITS FOR BRANCHING WHICH ARE SET BY SOME COMBINATION OF THE PULSES BR1, BR2, AND BR3. THE PULSE CLSQ SELECTS AN INSTRUCTION OR ONE OF A PAIR OF BRANCH SEQUENCES DEPENDING ON THE BRANCH BITS IN THE SQ REGISTER.

MOST CONTROL PULSES CLEAR OR WRITE INTO REGISTERS. CLEARING A REGISTER SETS ITS ELEMENTS (CORES OR LATCHES) TO THE ZERO STATE, AND PLACES ITS INITIAL CONTENTS ON ITS ASSOCIATED SENSE LINES. THE ALPHA-SIDE REGISTERS, INCLUDING FIXED AND ERASABLE STORAGE, ARE MADE OF CORES AND FEED THE ALPHA SENSE LINES. THE EXCEPTIONS ARE THE OUTPUT REGISTERS, WHICH ARE MADE OF LATCHES AND HAVE NO SENSE LINE CONNECTIONS. THE BETA-SIDE REGISTERS ARE MADE OF CORES (THE SPECIAL CIRCUITRY IN THE PARITY REGISTERS AND THE ADDER DOES NOT AFFECT THIS DISCUSSION) AND FEED THE BETA SENSE LINES, EXCEPT FOR P1, WHICH FEEDS THE PARITY TEST CIRCUIT. THE GAMMA-SIDE REGISTERS ARE MADE OF CORES. S FEEDS THE MEMORY ROPE INHIBIT LINES, AND SQ FEEDS THE SEQUENCE SELECTION MATRIX INHIBIT LINES.

WRITING INTO A REGISTER CONDITIONS ITS ELEMENTS TO BE DRIVEN BY ITS ASSOCIATED WRITE BUSSES, WHICH ARE USUALLY THE SENSE AMPLIFIER OUTPUTS. FIXED STORAGE HAS OF COURSE NO WRITE BUS CONNECTIONS, AND INPUT REGISTERS ARE DRIVEN BY DC LEVELS EITHER FROM OUTPUT LATCHES OR FROM EXTERNAL CONDITIONS.

OF THE CONTROL PULSES THAT MANIPULATE REGISTERS, SOME (CLE, WE, CLCTR1, CLCTR2, WCTR, AND CLI) MANIPULATE WHICHEVER OF A SET OF REGISTERS WAS CHOSEN BY SOME PREVIOUS PROCESS. THE OTHERS REFER TO REGISTERS BY NAME, AND ANY REGISTER SO REFERENCED IS CALLED A CENTRAL REGISTER. THIS CLASS INCLUDES ALL THE BETA-SIDE AND GAMMA-SIDE REGISTERS, AND IS TABULATED BELOW.

A	THE ACCUMULATOR REGISTER	(ALPHA)
B	THE BUFFER	(BETA)
BI	BUFFER INTERRUPT STORAGE	(ALPHA)
B14	THE PRODUCT TRANSFER REGISTER	(ALPHA)
C	THE COMPLEMENTING REGISTER	(BETA)
CYL	A CYCLE-LEFT-ONE REGISTER	(ALPHA)
D	THE DATA ADDRESS REGISTER	(ALPHA)
HP	THE HIGH-ORDER PRODUCT REGISTER	(ALPHA)
LP	THE LOW-ORDER PRODUCT REGISTER	(ALPHA)
N	THE NO-OVERFLOW SUM REGISTER	(ALPHA)
P1	THE PARITY TESTING REGISTER	(BETA)
P2	THE PARITY GENERATING REGISTER	(BETA)
Q	THE RETURN ADDRESS REGISTER	(ALPHA)
S	THE MEMORY SELECTION REGISTER	(GAMMA)
SQ	THE SEQUENCE SELECTION REGISTER	(GAMMA)
Z	THE INSTRUCTION LOCATION COUNTER	(ALPHA)
ZI	Z-REGISTER INTERRUPT STORAGE	(ALPHA)

AND THE ADDER, (BETA), WITH INPUTS X AND Y AND OUTPUT DENOTED SUM. WITH NO X INPUT, THE CONTROL PULSES WY CI INCREMENT THE NUMBER TRANSFERRED BY INTRODUCING A CARRY INTO BIT POSITION 1. SIMILARLY, AN INCREMENT OF TWO IS OBTAINED BY WY CI2.

MEMORY TRANSFERS FROM ALPHA TO BETA AND VICE VERSA DO NOT NEED TO ALTERNATE, SINCE THE SENSE AMPLIFIER GATES ARE OPERATED BY CONTROL PULSES. S AND SQ MAY BE WRITTEN INTO FROM EITHER SIDE.

ALL THE ALPHA-SIDE REGISTERS HAVE ADDRESSES IN ERASABLE STORAGE. THESE AND SOME MAJOR PROPERTIES ARE GIVEN BELOW.

	A	0000	NORMAL WIRING.
	Q	0001	NORMAL WIRING.
	D	0002	NORMAL WIRING.
	Z	0003	NORMAL WIRING.
*	N	0005	SEE BELOW.
	CYL	0011	CYCLES LEFT 1 BIT.
	BI	0010	NORMAL WIRING.
	B14	0014	BIT 14 ONLY. WHP IMPLIES WB14, CLLP IMPLIES CLB14.
*	HP	0015	SHIFTS RIGHT ONE, PLACING BIT 1 INTO B14.
	LP	0016	CYCLES RIGHT ONE, HAS NO BIT 14.
	ZI	0017	NORMAL WIRING.

THE ADDER CONTAINS AN EXTRA BIT (BIT 16). THE SIGN BIT (BIT 15) OF AN INPUT TO THE ADDER IS WRITTEN INTO ADDER BITS 16 AND 15. WHEN THE ADDER IS CLEARED, BIT 16 ACTIVATES WRITE BUS 15 SO THAT THE REGULAR WRITE BUSSES CONTAIN AN OVERFLOW-CORRECTED SUM. ADDER BIT 15 ACTIVATES A SPECIAL LINE WHICH GOES TO THE REGISTERS MARKED * ABOVE. IT FEEDS BIT 15 OF N (WHICH IS NORMALLY WIRED OTHERWISE) AND BIT 14 OF HP .

THE OCTAL CODE, NAME, DURATION, MNEMONIC CODE, ACTION, AND PULSE SEQUENCE OF EACH INSTRUCTION IS GIVEN BELOW. THE FORMAT

L ** X

INDICATES THAT BITS 15-13 OF C(L) ARE INTERPRETED AS THE OPERATION **, AND BITS 12-1 OF C(L) AS THE ADDRESS X. NOTE THAT IF THE PRECEDING INSTRUCTION WAS AN INDEX ORDER, THIS DESCRIPTION APPLIES NOT TO C(L), BUT TO THE SUM OF C(L) AND THE QUANTITY REFERENCED BY THE INDEX. THE APPEARANCE OF L IN THE DESCRIPTION OF THE ACTION OF EACH INSTRUCTION HAS ITS REGULAR MEANING IN THIS CASE, HOWEVER. THE INTERRUPT, RESUME, AND COUNTER INCREMENT/DECREMENT SEQUENCES ARE ALSO DESCRIBED IN THIS SECTION.

CODE 0. TRANSFER CONTROL 1 I

L TC X SET C(Q) = TC L+1 AND TAKE NEXT INSTRUCTION FROM X.

1.	ALPHA	CLZ	WY					
2.	ALPHA	CLE	WB	WP1	WS	WSQ		
3.		CLQ						
4.	BETA	CLSUM	WQ					
5.	ALPHA	CLD	WY	CI				
6.	BETA	CLSUM	WZ					
7.								
8.	BETA	CLB	CLP1	TP	WE	WD	CLS	CLSQ

NOTE THAT TC Q WORKS BUT LEAVES C(Q) SET TO THE LOGICAL SUM OF TC L+1 AND THE ORIGINAL C(Q).

CODE 1. COUNT COMPARE AND SKIP 2 I

L CCS X IF C(X) IS POSITIVE NON-ZERO, SET C(A) TO C(X)-1 AND TAKE NEXT INSTRUCTION FROM L+1.

IF C(X) = +0, SET C(A) = +0 AND TAKE NEXT INSTRUCTION FROM L+2.

IF C(X) IS NEGATIVE NON-ZERO, SET C(A) TO -C(X)-1 AND TAKE NEXT INSTRUCTION FROM L+3.

IF C(X) = -0, SET C(A) = +0 AND TAKE NEXT INSTRUCTION FROM L+4.

C(A) IS SET TO +0 FOR C(X) = +1 OR C(X) = -1.

	1.		CLD						
	2.	ALPHA	CLE	WB	WP1	WSQ	BR1		
	3.	BETA	CLB	CLP1	TP	WE	WD	CLSQ	
4+.	ALPHA	CLD	WC			4-.	ALPHA	CLZ	WY	CI2
5+.	BETA	CLC	WD	CLA		5-.	BETA	CLSUM	WZ	CLA
	6.	ALPHA	CLD	WY	CI					
	7.	BETA	CLSUM	WA	WSQ	BR2	II			
	8.		CLSQ							
1+.	ALPHA	CLZ	WY	CI		1-.	ALPHA	CLA	WC	
2+.	BETA	CLSUM	WZ	CLA		2-.	BETA	CLC	WA	
	3.									
	4.	ALPHA	CLZ	WY	CI	WS				
	5.	BETA	CLSUM	WZ	CLS					
	6.									
	7.	ALPHA	CLE	WB	WP1	WS	WSQ			
	8.	BETA	CLB	CLP1	TP	WE	WD	CLS	CLSQ	

CODE 2. MODIFY NEXT INSTRUCTION 1 I

L INDEX X

TAKE AS THE NEXT INSTRUCTION THE OVERFLOW-CORRECTED SUM OF C(L+1) AND C(X). IF L+1 IS IN ERASABLE, ITS CONTENTS ARE NOT CORRECTLY RESTORED.

1.	ALPHA	CLZ	WY	CI	WS			
2.	ALPHA	CLE	WB	WP1				
3.	BETA	CLSUM	WZ	CLD	CLS			
4.	BETA	CLB	CLP1	TP	WE	WD		
5.	ALPHA	CLE	WX	WP1				
6.	ALPHA	CLD	WY					
7.	BETA	CLSUM	WD	WS	WSQ			
8.	BETA	CLB	CLP1	TP	WE	WD	CLS	CLSQ

CODE 3. EXCHANGE 1 I

L XCH X

INTERCHANGE C(A) WITH C(X). THE OP CODE CAF MAY BE USED WHEN X IS IN FIXED STORAGE (CLEAR AND ADD FIXED).

1.	ALPHA	CLA	WB	WP2				
2.	ALPHA	CLZ	WY	CI	WS	CLP1		
3.	BETA	CLSUM	WZ	CLD				
4.	ALPHA	CLE	WY	WP1				
5.	BETA	CLB	CLP2	WE	CLS			
6.	BETA	CLSUM	CLP1	TP	WA			
7.	ALPHA	CLE	WB	WP1	WS	WSQ		
8.	BETA	CLB	CLP1	TP	WE	WD	CLS	CLSQ

CODE 4. CLEAR AND SUBTRACT 1 I

L .CS X SET C(A) TO THE ONES COMPLEMENT OF C(X).

1.	ALPHA	CLZ	WY	CI	WS			
2.	ALPHA	CLE	WB	WP1	WC			
3.	BETA	CLB	CLP1	TP	WE			
4.		CLA						
5.	BETA	CLSUM	WZ	CLS	CLD			
6.	BETA	CLC	WA					
7.	ALPHA	CLE	WB	WP1	WS	WSQ		
8.	BETA	CLB	CLP1	TP	WE	WD	CLS	CLSQ

CODE 5. TRANSFER TO STORAGE 1 I

L TS X SET C(X) TO C(A). TRANSMISSION IS THROUGH THE ADDER SO THAT TS WILL WORK FOR N AND HP .

1.	ALPHA	CLA	WY	WP2	WB			
2.		CLE						
3.	BETA	CLSUM	CLP2	WE				
4.	ALPHA	CLZ	WY	CI	WS			
5.	BETA	CLSUM	WZ	CLS	CLD			
6.	BETA	CLB	WA					
7.	ALPHA	CLE	WB	WP1	WS	WSQ		
8.	BETA	CLB	CLP1	TP	WE	WD	CLS	CLSQ

CODE 6. ADD AND COUNT ON OVERFLOW 1 I

L AD X SET C(A) TO THE OVERFLOW-CORRECTED SUM OF C(A) AND C(X). SUPERPOSE ON C(N) THE UNCORRECTED SUM. IF OVERFLOW OCCURS, SET AN INTERRUPT SIGNAL TO CAUSE INCREMENTING OR DECREMENTING OF THE OVERFLOW COUNTER, ACCORDING AS THE ADDENDS WERE POSITIVE OR NEGATIVE.

1.	ALPHA	CLZ	WY	CI	WS				
2.	BETA	CLSUM	WZ	CLD					
3.	ALPHA	CLE	WX	WB	WP1				
4.	BETA	CLB	CLP1	TP	WE				
5.	ALPHA	CLA	WY	CLS					
6.	BETA	CLSUM	WA	WN					
7.	ALPHA	CLE	WB	WP1	WS	WSQ			
8.	BETA	CLB	CLP1	TP	WE	WD	CLS	CLSQ	

CODE 7. MULTIPLY 16 I

L MP X DELIVER THE PRODUCT OF C(A) AND C(X) TO A (HIGH-ORDER PART) AND LP (LOW-ORDER PART). THE SIGNS OF THE TWO PARTS AGREE AND ARE STRICTLY DETERMINED BY THE SIGNS OF THE OPERANDS. PULSE SEQUENCE IS ON THE NEXT PAGE.

PULSE SEQUENCE OF MULTIPLY INSTRUCTION.

```
        1.  ALPHA CLA   WB    WC    WSQ   BR1   BR2
        2.        CLLP  CLSQ

3+.  BETA  CLB  WLP  CLD              3-.  BETA  CLC  WLP  CLD
4+.  ALPHA CLE  WY   WP1  WB  CLC     4-.  ALPHA CLE  WY   WP1  WC  CLB

        5.  BETA  CLB   CLC   WD    WSQ   BR2   BR3
        6.  BETA  CLSUM CLP1  TP    WE    CLCYL CLSQ

7+.         CLHP  II          7-.         ALL1S WHP   II
8+.         00001 WCYL        8-.         00001 WCYL

        1.  ALPHA CLLP  WB    WSQ   BR1   BR3
        2.  BETA  CLB   WLP   CLSQ

3+.               3-.  ALPHA CLD   WB    WX
4+.               4-.  BETA  CLB   WD

        5.  ALPHA CLHP  WY
        6.  BETA  CLSUM WHP
        7.  ALPHA CLCYL WB    WSQ   BR3   II
        8.  BETA  CLB   WCYL  CLSQ

        1.
        2.
        3.  ALPHA CLHP  WB
        4.  ALPHA CLZ   WY    CI    WS
        5.  BETA  CLSUM WZ    CLS   CLD
        6.  BETA  CLB   WA
        7.  ALPHA CLE   WB    WP1   WS    WSQ
        8.  BETA  CLB   CLP1  TP    WE    WD    CLS   CLSQ
```

INCREMENT OR DECREMENT 1 I

THIS SEQUENCE IS TRIGGERED BY A SUITABLE INTERRUPT SIGNAL AND FOLLOWS LINE 8 OF ANY SEQUENCE OR 8-LINE PORTION THEREOF, EXCEPT THE DUMMY LINE 8 THAT IMMEDIATELY PRECEDES INTERRUPT AND RESUME. ITS ACTION IS TO SET THE CONTENTS OF THE COUNTER SELECTED AS CTR (C(CTR)) TO C(CTR)+00001 (INCREMENT) OR TO C(CTR)+77776 (DECREMENT), CHECKING PARITY OF THE INITIAL C(CTR) AND GENERATING PARITY FOR THE SUM.

```
                1.  ALPHA CLCTR1 WX      WP1

2+.         00001   WY                2-.         77776   WY

                3.          CLP1   TP
                4.  BETA    CLSUM  WCTR
                5.  ALPHA   CLCTR2 WY      WP2
                6.  BETA    CLSUM  CLP2    WCTR    II
                7.
                8.
```

INTERRUPT 1 I

THIS SEQUENCE IS TRIGGERED BY A SUITABLE INTERRUPT SIGNAL AND MAY FOLLOW ANY LINE 7 NOT INCLUDING THE CONTROL PULSE II. THE PULSE CLI SUPPLIES THE ADDRESS OF THE DESIRED INTERRUPT PROGRAM. THIS SEQUENCE IS PRECEDED BY A DUMMY LINE 8, AND IS FOLLOWED BY THE STANDARD LINE 8 WHICH WAS LEFT SET UP BY THE INTERRUPTIBLE SEQUENCE. THE PULSE INH INHIBITS ANY INTERRUPT UNTIL THE OCCURENCE OF THE PULSE ENI IN THE RESUME SEQUENCE.

1.	ALPHA	CLD	WB	CLSN	INH	
2.	BETA	CLB	CLP1	TP	WE	WBI
3.	ALPHA	CLZ	WB			
4.	ALPHA	CLI	WY	CI	WS	
5.	BETA	CLSUM	WZ	CLS		
6.	BETA	CLB	WZI			
7.	ALPHA	CLE	WB	WP1	WS	WSQ

RESUME 1 I

THIS SEQUENCE IS TRIGGERED BY REFERENCING THE RESUME ADDRESS, CUSTOMARILY WITH A TS INSTRUCTION. ITS BEHAVIOR WITH RESPECT TO LINES 7 AND 8 IS ANALOGOUS TO THAT OF INTERRUPT. THE PULSE ENI ENABLES INTERRUPT.

1.						
2.		CLD				
3.		CLZ	CLSN	ENI		
4.	BETA	CLB	CLP1	TP	WE	
5.	ALPHA	CLZI	WB			
6.	BETA	CLB	WZ			
7.	ALPHA	CLBI	WB	WP1	WS	WSQ

SINCE INSTRUCTIONS MAKE INTERMIXED REFERENCES TO REGISTERS BY ADDRESSING AND BY NAME, THE BEHAVIOR OF THE COMPUTER WHEN AN ALPHA-SIDE CENTRAL REGISTER IS ADDRESSED IS NOT ALWAYS OBVIOUS. THIS SECTION LISTS THE ALPHA-SIDE CENTRAL REGISTERS REFERENCED EXPLICITLY BY EACH INSTRUCTION, AND BRIEFLY DESCRIBES THE BEHAVIOR OF EACH CASE. IN SOME OF THE USEFUL CASES, A SPECIAL SYMBOLIC OPERATION CODE IS GIVEN. IN YUL LANGUAGE, IT IS PERMISSIBLE TO USE THIS CODE AND LEAVE THE ADDRESS FIELD BLANK. THE ASSEMBLER WILL SUPPLY THE CORRECT ADDRESS.

L TC Q OPTIONAL CODE.....RETURN
TAKE NEXT INSTRUCTION FROM Q AND SUPERPOSE TC L+1 ON Q. COMPLICATED BEHAVIOR RESULTS IF THE INSTRUCTION TAKEN FROM Q IS NOT A TC.

L TC D NO OPTIONAL CODE. CAUSES INFINITE LOOP.

L TC Z NO OPTIONAL CODE. COMPLEX, NOT USEFUL.

L CCS A NO OPTIONAL CODE.
STRAIGHTFORWARD AND USEFUL.

L CCS D NO OPTIONAL CODE.
SET C(A) = +0 AND TAKE NEXT INSTRUCTION FROM L+2. PROBABLY NOT USEFUL.

L CCS Z NO OPTIONAL CODE.
SET C(A) = TC L AND TAKE NEXT INSTRUCTION FROM L+1. PROBABLY NOT USEFUL.

```
L       INDEX  D                NO OPTIONAL CODE.
TAKE AS THE NEXT INSTRUCTION THE OVERFLOW-CORRECTED SUM OF C(L+1) AND
20002.  PROBABLY NOT USEFUL.

L       INDEX  Z                NO OPTIONAL CODE.  EQUIVALENT TO  NOOP .


L       XCH    A                NO OPTIONAL CODE.  EQUIVALENT TO  NOOP .

L       XCH    D                NO OPTIONAL CODE.  COMPLEX, NOT USEFUL.

L       XCH    Z                NO OPTIONAL CODE.  COMPLEX, NOT USEFUL.


L       CS     A                OPTIONAL CODE.....................COM
REPLACE C(A) BY ITS ONES COMPLEMENT.

L       CS     D                NO OPTIONAL CODE.
REPLACE C(A) BY 37775.  PROBABLY NOT USEFUL.

L       CS     Z                NO OPTIONAL CODE.
SET C(A) = -0.  PROBABLY NOT USEFUL.
```

L TS A NO OPTIONAL CODE. EQUIVALENT TO NOOP .

L TS D OPTIONAL CODE.....NOOP
PROCEED TO NEXT INSTRUCTION.

L TS Z OPTIONAL CODE.....TCAA
TAKE NEXT INSTRUCTION FROM THE LOCATION WHOSE ADDRESS IS IN BITS 12-1 OF C(A). LEAVE C(Q) UNDISTURBED. TCAA STANDS FOR TRANSFER CONTROL TO THE ADDRESS IN A . THE VALUES OF BIT 15-13 OF C(A) ARE IRRELEVANT.

L AD A OPTIONAL CODE.....DOUBLE
STRAIGHTFORWARD AND USEFUL.

L AD D OPTIONAL CODE.....ORN
SUPERPOSE (INCLUSIVE OR) C(A) ON N. THIS IS THE ONLY PURE BOOLEAN OPERATION IN THE MACHINE (INDEPENDENT OF THE DATA, THAT IS). AN ADDITION WITH EITHER C(A) OR C(X) = +0 WILL DO THE SAME THING.

L AD Z NO OPTIONAL CODE.
REPLACE C(A) BY C(A) + (TC L+2). SUPERPOSITION ON N , OVERFLOW PROVISIONS AS USUAL. PROBABLY NOT USEFUL.

L AD N NO OPTIONAL CODE. COMPLEX, NOT USEFUL.

L MP A NO OPTIONAL CODE.
SET C(A) AND C(LP) TO ZERO WITH THE SIGN OF THE INITIAL C(A). PROBABLY NOT USEFUL.

L MP D NO OPTIONAL CODE. SAME AS MP A .

L MP Z NO OPTIONAL CODE.
DELIVERS THE PRODUCT OF C(A) AND (TC L+1) TO A AND LP .
STRAIGHTFORWARD, PROBABLY NOT USEFUL.

L MP CYL NO OPTIONAL CODE.
NON-DETERMINISTIC BEHAVIOR.

L MP HP NO OPTIONAL CODE. COMPLEX, NOT USEFUL.

L MP LP NO OPTIONAL CODE.
IF BITS 15 AND 1 OF C(A) HAVE THE SAME VALUE, DELIVER HALF THE SQUARE OF THE INITIAL C(A) TO A AND LP . PROBABLY NOT USEFUL. IF BITS 15 AND 1 OF C(A) DIFFER, COMPLEX AND NOT USEFUL.

THERE ARE THREE OTHER OPTIONAL CODES THAT ARE USEFUL.

L XAQ EXECUTE C(A) USING Q.
THIS IS ASSEMBLED INTO TC A . IT IS STRAIGHTFORWARD, AND SINCE Q FOLLOWS A IN MEMORY, BEHAVES LIKE AN EXECUTE INSTRUCTION WHEN C(A) IS NOT A TC .

L EXIT LEAVE INTERPRETIVE MODE.
THIS IS ASSEMBLED INTO +0 AND IS RECOGNIZED BY THE INTERPRETER AS A SIGNAL TO GIVE CONTROL TO THE BASIC 3C INSTRUCTION IN L+1.

L RESUME RESUME AFTER PROGRAM INTERRUPT.
THIS IS ASSEMBLED INTO TS RIP , WHERE RIP IS THE ADDRESS OF THE RESUME-INTERRUPTED-PROGRAM REGISTER IN MEMORY. C(A) IS IRRELEVANT.

APPENDIX IV
YUL SYSTEM FOR 3C AND RELATED COMPUTERS

APPENDIX IV
THE YUL SYSTEM FOR 3C AND RELATED COMPUTERS

INTRODUCTION

The Yul System is a set of programs for the IBM 650 and the Honeywell 800 which aid the preparation of programs for the Apollo Guidance Computer (3C) and related computers in the Mars computer series. The series consists of machines named 1B, 3S, and 3C. The name 3C is used in what follows, but the discussion applies equally to the other computers, except where noted. The system has two major components, the Assembler and the Simulator. Primary inputs to the Assembler are IBM cards containing 3C programs in symbolic Yul Language, and primary outputs are wiring diagrams for fixed memory ropes. The Assembler maintains files of active programs and subroutines in symbolic form, and of active programs in binary form. Control operations are provided for the manipulation of these files. Inputs to the Simulator are binary records prepared by the Assembler and control cards specifying 3C environment and editing parameters, and outputs are edited and annotated accounts of the action of the 3C programs. The Simulator will be available as a MACH subroutine so that MACH simulation of environment can proceed in parallel with simulation of a 3C program as it interacts with environment.

Appendix VI shows a representative 3C program, containing many but not all of the types of operation which the Yul system accepts. Each line of printing represents one IBM card in this program. Those cards beginning with the letter R represent remarks cards and are ignored by the compiler. It may be seen that the remaining cards are divided into five fields of information. These represent, respectively, the card number,

the location field, the operation field, the address field, and the remarks field occupying the right-hand half of the card. In the operation field are written the instruction codes for the basic computer operations, or instruction codes for interpreted instructions, or, from time to time, certain other symbols having special significance to the Yul system. The location field is frequently blank, but is used when necessary to contain a mnemonic tag or symbol identifying the name of the register corresponding to that instruction. The address field contains the address part of the instruction itself, that is, the twelve low-order bits. Instructions are by and large written and stored in sequence, except when special clerical codes tell the compiler to do otherwise. Further details on the Yul Language are given at a later point in this appendix. The right-hand, or remarks, half of the card is ignored by the Yul system.

ASSEMBLER

The Assembler may be directed to assemble a new program or subroutine; revise a program or experimental subroutine; delete a program or subroutine; turn an experimental subroutine into a library subroutine; punch a symbolic deck of a program or subroutine; print a symbolic listing of a program or subroutine; or wire an assembled program. Wiring may be optionally requested along with a program assembly or revision, but is done only if the Assembler finds no errors in the program.

Assembly or revision takes place in three passes (the wiring diagrammer is called Pass 4), using a system tape and two work tapes. The system tape consists of the Assembler program, various tables and catalogues, and three main files, known as SYPT, SYLT, and BYPT. SYPT, or Symbolic Yul Program Tape, consists of card images of all active programs in the original symbolic Yul Language. SYLT, or Symbolic Yul Library Tape, consists of card images of experimental and library subroutines in Yul Language, and a catalogue showing which programs and subroutines use each subroutine. BYPT, or

Binary Yul Program Tape, consists of binary records generated by assembly. A binary record* (actually several physical tape records) is the binary representation of a program and the subroutines used by it.

Input to Pass 1 consists of symbolic program cards (new program) or symbolic program correction cards and SYPT records (revision), and any necessary SYLT records. In the revision of an experimental subroutine, the correction cards are of course merged with SYLT records. Pass 1 forms tables in core memory and writes combined symbolic and binary information on tape.

Pass 2 finishes the assembly proper, using and updating the internal tables; updates SYPT (program assembly) or SYLT (subroutine assembly); makes up an unsorted binary record; and prints a listing of the program in which each card of Yul Language is reproduced in full, with any pertinent binary equivalent on the same line (mostly in octal), followed by as many lines of error or warning notices as needed. The vertical format originally punched in the cards is used to determine the vertical format of the listing.

Pass 3 prints a summary of the assembly, using the tables generated by Pass 1 and amended by Pass 2. The symbols used are listed in alphabetical order (see Honeywell character chart in H-800 Reference Manual) with the equivalent of each in octal and decimal notation, and the conditions under which it was defined, or any error associated with it. This is followed by a summary showing the number of symbols in each "health" category. Then a storage map is printed, showing the type (fixed, erasable, or special/nonexistent) and the reserved-unreserved status of each location. The binary record, which was sorted during the preceding Pass 3 printing operations, is printed in octal at eight words per line. If the assembly is of a program and is valid, the sorted binary record is inserted in BYPT; and if furthermore, wiring was requested, Pass 4 follows.

* Also colloquially called a "load deck", not because any cards are involved, but out of habit from 650 usage.

Otherwise, the next job begins.

Pass 4, which is also available as a separate operation for previously assembled BYPT records, prints a production wiring diagram and a checkout listing for each rope, using rope description cards supplied by the programmer.

YUL LANGUAGE

This description is not intended to be a complete programming manual, but presents enough of the language so that patches of coding can be written, to be tied into coherent programs when a complete manual (and the Assembler!) is completed. It suffers from the fairly common disease of being about something that does not exist.

(1) Alphabet: The alphabet used in Yul Language is the 57-character set (including blank) that can be printed by the Honeywell high-speed printer. This is the set available (without multipunching) on the keypunches, plus the following characters:

Character	Name	Card Code
'	Apostrophe	8 2
&	Ampersand	8 7
;	Semicolon	+8 2
%	Percent	+8 5
■	Blot	+8 6
#	Sharp	-8 2
"	Quotes	-8 5
@	At	0 8 2
C_R	Credit	0 8 5

It should be noted that the colon (:) in the Honeywell set corresponds to the card code 8 4, which is shown as a dash (-) on the keypunches (upper left-hand key, alphabetic shift, and which prints as a dash on the 407. The Yul System does NOT recognize this character as a minus sign. The SKIP key above the numerals (its skipping function has been disconnected) gives the legitimate

minus sign, in either shift. Any character not in the printer set but in the 64-character set acceptable to the card reader is replaced with a blot (■) upon input to the Yul System. Since the blot has this and other special purposes, its use is not recommended, although not illegal. The appearance of a character not acceptable to the card reader will cause the job to be terminated.

(2) Subfields: the fundamental unit of meaning in Yul Language is the subfield, of which there are four types: blank, numeric, signed numeric, and symbolic. A subfield always occurs in a field of definite size. A blank subfield consists entirely of blanks, and is therefore detectable only when it is alone in a field. A numeric subfield consists of numerals, optionally followed by a D; imbedded, leading, or trailing blanks are ignored, but there may be at most eight non-blank characters. This type of subfield is interpreted as an octal integer, unless the D is included, in which case it is interpreted as a decimal integer. If it contains any 8's or 9's but no D, it is considered to be decimal, but a complaint is printed. A signed numeric subfield has either + or - as its leftmost non-blank character, and has otherwise the same definition as the numeric subfield, except that the sign must be counted toward the maximum number of non-blank characters. A symbolic subfield is any collection of characters which is at most eight characters in length from its leftmost to rightmost non-blank character, which does not satisfy the definitions for other types. Leading blanks are ignored, but imbedded blanks are significant, and since a symbolic subfield is exactly eight characters long for internal processing purposes, some trailing blanks may be significant.

(3) Fields: The fields of primary interest in Yul Language are the location, operation, and address fields, but a word about overall card format is in order here. The 407 board for Honeywell card formats uses columns 1 and 8 for control information, and the Yul System interpretation of these columns includes the 407 conventions. The codes of greatest interest are the following:

(a) Vertical spacing control by column 8:

blank	normal upspace after printing,
2	double upspace after printing,
4	quadruple upspace after printing,
8	skip to next page after printing;

(b) Card content control by column 1:

blank	Yul System four-field format,
R or P	columns 9-80 contain remarks.

When column 1 contains blank or R, the 407 prints columns 1-7 in print positions 1-7, and columns 9-80 in print positions 9-80. In the Pass 2 printout, the Assembler prints columns 1-7 in print positions 1-7, columns 9-80 in print postions 49-120, and complaints and octal translation in print positions 9-47, and upspacing according to column 8. The discussion of fields, then, is concerned with the two types of cards listed in (b). Cards with R in column 1 are inserted in SYPT or SYLT and are printed during Pass 2, but have no other effect on assembly. Cards with column 1 blank are considered to consist of the following four fields:

columns	9-16	Location field,
columns	18-23	Operation field,
columns	25-40	Address field,
columns	41-80	Remarks field.

Columns 2-7 contain the card number. The first card in a deck must have a number greater than zero, and each card must have a number greater than that of the preceding card unless it specifies a sequence break. The characters blank and zero are equivalent in a card number. The occurrence of any character other than blank or a digit in the card number specifies a sequence break: no comparison is made with the preceding card number, and the next card number is required to be greater than zero.

The location and operation fields consist of one subfield each, and the address field consists of one or two subfields (in certain cases the address field contains special formats; these will be described where they apply). The remark field is treated like the contents of an "R" card.

(4) Instruction cards: The card type most frequently occurring specifies the location and makeup of 3C instruction of either the direct or interpreted type. Two control operations must be introduced (in an oversimplified way) to clarify the rules for writing instructions. The LOC operation specifies the location of the next 3C word defined in the program deck. In its simplest form, the LOC operation is written thus:

Location	Operation	Address
blank	LOC	numeric

The Assembler's Location Counter (LC) is set to the value of the address. At the beginning of an assembly, LC contains an illegal address. The HEAD (or TAIL: these words are equivalent) operation is useful when several programmers contribute to the same program (as when subroutines are used), and is written thus:

Location	Operation	Address
blank or one character	HEAD or TAIL	(irrelevant)

The rightmost non-blank character (or blank if there is none) in the location field is set up as the prevailing head until another HEAD operation occurs. Each symbol encountered in a location or address field is left-justified, and if the eighth character is a blank, the prevailing head is inserted. Thus a set of programmers can avoid conflicts in their use of symbols by restricting symbols to 7 characters or less and using different heads. The prevailing head at the beginning of an assembly is a blank. An instruction card is written thus:

Location	Operation	Address
blank or	symbolic or	blank or
signed numeric or	numeric	signed numeric or
numeric or		numeric or
symbolic		symbolic or
		numeric and signed numeric or
		symbolic and signed numeric

If the location field is blank or signed numeric, the word specified in the operation and address fields is assigned to the location given by c (LC). If the location field is symbolic, the same is true (unless the symbol has been defined in certain ways), and the symbol becomes defined as equivalent to c (LC). A numeric location field overrides the setting of LC and resets it. In all cases, c (LC) is incremented by 1 after each 3C word is processed.

The operation field will usually contain a symbolic order code for a direct or interpreted 3C instruction. If it is numeric, it is regarded as occupying the same bits in the word as a direct order code. A blank operation field has the same effect as an R in column 1.

In 3C and other machines the rightmost non-blank character of certain symbolic operation codes may be an asterisk (to indicate indirect addressing). See appendix V for the rules governing indirect addressing in the interpretive 3C language.

If the address field is blank, c (LC) is used as the address. If it is signed numeric, the algebraic sum c (LC) + the value is used. If it is numeric, that value is the address. If it is numeric and signed numeric, the two subfields must be separated by at least one blank, and the algebraic sum is used as the address. A negative address value is meaningless. If it is symbolic, the numeric equivalent of the symbol (assigned either earlier or later in the program deck - -this is why there are two main passes) is the address. If it is symbolic and signed numeric, the two subfields must be separated by at least one blank, and the algebraic sum

of the equivalent of the symbol and the signed numeric value is used as the address. The symbolic subfield (if any) of an address field is treated in a special way to allow an instruction in a section of coding under one prevailing head refer to a symbol defined under a different head. There are several ways of doing this, but a convenient one is provided for as follows. If a symbolic subfield in an address field consists of more than one non-blank character, and if the rightmost of these is separated from the others by at least one blank, that character is shifted with respect to the left-justified symbol so as to be in the eighth, or head, position. An outline of the Assembler's rules for decoding an address field may be helpful:

(a) see if address field is all blank; if so, investigation is finished; if not,

(b) see if the leftmost character is a sign or if there is any sign preceded by at least one blank; if not, go to (e); if so,

(c) see if the rightmost of these, together with the characters to its right, form a signed numeric subfield; if not, go to (e): if so,

(d) record the presence and value of the subfield, and delete it from the address field; then see if the field is blank; if not, go to (e); if so, the field is signed numeric.

(e) see if the field constitutes a numeric subfield; if not, go to (f); if so, the field is wholly or partially numeric.

(f) see if there is more than one non-blank character and if the rightmost of these is preceded by at least one blank; if not, go to (h); if so,

(g) see if the seventh character following the leftmost character is a blank; if not, go to (i); if so, shift the character found in (f) to that postion; then

(h) see if the field constitutes a symbolic subfield; if not, go to (i); if so, the field is wholly or partially symbolic.

(i) the address field is meaningless.

In the light of these rules, the definition of a symbolic subfield should include a provision that it be so structured as to be correctly interpreted when it occurs in an address field.

Certain symbolic operation codes imply a specific fixed address (see the last section of appendix III). In these cases, the address field is not examined.

Two other control operations should be defined to make up a basic usable subset of the Yul Language. The IS or = operation is used to assign an equivalent to a symbol when no other method is convenient.

Location	Operation	Address
symbolic	IS or = or EQUALS	any format usable for instructions

If the address field contains a symbol, that symbol must have been defined earlier in the program deck. If a symbol defined by IS or = occurs subsequently in the location field of an instruction card, the LC is overridden and reset. The OCT or OCTAL code defines a constant, written as an octal integer.

Location	Operation	Address
any format usable for instructions	OCT or OCTAL	numeric or signed numeric

The location field is treated as in instructions. The address field specifies one 3C word; if the field is signed numeric with a minus sign, the 3C word generated is the ONES' complement of the number shown.

APPENDIX V
INTERPRETED INSTRUCTIONS

APPENDIX V

INTERPRETED INSTRUCTIONS

INTRODUCTION

It is recognized that the principal applications for the 3C computer will require a computational accuracy that cannot be met directly with a 16-bit word length. As a result, an early effort has been made to provide extra-precision computer operations as a normal programming tool. Recognizing also that many of the applications will involve complicated geometric considerations, it has seemed natural to provide vector operations as well. Finally, the set of eight basic operations which the 3C computer provides is indeed rather restrictive, and a more diverse set would be highly desirable.

To satisfy these requirements, a computer program has been prepared to interpret a set of pseudo-codes or interpretive instructions. These pseudo-codes are 32 in number and include a variety of double-precision operations, a small number of triple-precision operations, together with a set of double-precision vector operations. The Yul system will recognize these interpretive codes and translate them accordingly, so that for practical purposes a programmer may, for the most part, act as though he has a much more versatile computer at his disposal. However, this facility does not come free. The first price which is paid is approximately 35 words of erasable storage and approximately 700 words of fixed program storage used by the interpreter. In addition, the interpreted instructions generally consume many instruction times to execute. As an example, the double-precision multiplication operation requires approximately 140 instruction times, in contrast to the 16 required for the basic single-precision multiply instruction. On the other hand, the capability to execute some of the more powerful instructions, such as a

vector addition by a single operation code, may lead to very considerable savings in storage requirements.

USE OF INTERPRETED INSTRUCTIONS

To make use of the interpreted instructions, a programmer must initiate a transfer of control to the first word of the interpretive program; that is, to a register labeled INTPRET in the section of the interpretive program shown in Appendix VI. Following the instruction TC INTPRET , he then writes in sequence the instructions which he wishes interpreted. Program INTPRET picks up each instruction in turn, breaks it apart into an operation code and an address, and itself transfers control to the necessary subroutine to carry out the operation. It continues this action on successive instructions until it encounters an instruction word which is identically zero. This it interprets as a request to exit from the interpretive mode back into the basic mode of operation. The control is then transferred to the word immediately following the zero word, and successive instructions thereafter are executed as basic computer instruction. The Yul system is capable of recognizing the operation EXIT, and will replace this instruction by a zero word.

Appendix VI contains the beginning part of the interpreter program, used as an illustration of programming techniques. The program begins in register number octal 1040 or decimal 544, and in its present version extends for 698 consecutive registers. It uses ordinary erasable registers numbered octally from 60 through 121, a total of 33 in all. In addition to the interpreter program proper, a double-precision square root routine is included in this count. In the interpreter as it now stands, a total of 30 out of the possible 32 instructions are defined. The remaining two are temporarily held in reserve pending further programming applications.

The preceding remarks concerning the exact size and storage requirements for the interpreter should not be taken too literally in program planning, nor should the timing estimate for

the interpreted instructions be taken as exact. These estimates of time and storage requirements are obtained from an actual count of program words, and are indicative of the probable time and storage utlimately required. However, the program in its present form has not been rigorously checked out, nor has it been carefully examined from the standpoint of optimizing and integrating its operations. As a result, it is almost certain that changes of one sort or another will be made before the program becomes operative. It is doubtful, however, that any major change in time or storage will result other than those arising from the elimination or addition of major instructions.

Before proceeding with detailed discussion of the individual interpretive instructions, a few of the erasable registers used by the program should be specifically discussed. As noted above, a certain number of triple-precision operations are carried out as well as the common double-precision orders. For this purpose, registers DAC to DAC +2 are reserved as an extra-precision accumulator. The initials DAC are a mnemonic misnomer, standing for the phrase "double accumulator." For most of the instructions, however, use of DAC is restricted to double-precision work. This pair or triplet of registers plays much the same role for the interpreted instructions as does the ordinary accumulator, A, for the basic computer orders. There exist, for example, double-precision operations exactly analogous to the CS, TS, and XCH instructions. In addition, however, there are instructions for executing double-precision division and for shifting the entire contents of the triplet of registers left or right by one or more bit positions. Somewhat analogous to the register Q in the basic machine language, the register IQ is used by the interpreter for much the same purpose.* The interpreted transfer of control instruction, ITC, leaves in register IQ the address immediately following that from which the ITC instruction was executed. An

*And not to be confused with register QI, which is where (Q) is kept during an interrupt. Mnemonics based on the words INTERRUPT and INTERPRET are basically confusing, since they have all but two letters in common.

interpreted transfer of control to the address IQ will serve adequately as a subroutine exit, much as the operation TC Q does. A further instruction, ITA, interpreted transfer address, is provided. This serves directly to store the contents of IQ in a designated address. In addition to the double-precision accumulator DAC, a set of six registers, VAC to VAC +5, are reserved for a double-precision vector accumulator. Details of operations carried out with the double-precision vector accumulator may be found under the discussion of the individual instructions. No attempt is made in the present scheme to execute any triple-precision operations with vectors, with the sole exception that the dot product operation accumulates a triple-precision sum of the three double-precision products involved. Neither, for that matter, are any single-precision vector operations considered here. Finally, it should be mentioned that a set of six registers, VBUF to VBUF +5, are used as temporary erasable registers in various vector operations, but not by the other instructions. As a result, these are available to the programmer for use as temporary storage if his requirements do not involve use of vectors.

To provide 32 interpreted instruction codes within the framework of a 15-bit computer word, a revised addressing procedure is used in the interpreter. The 4,096-word computer storage is divided into 16 blocks of 256 words each. The first two of these blocks constitute the erasable storage section, including the special registers and input-output registers. The remaining 14 blocks constitute fixed storage. Two types of addressing are permitted, direct addressing of erasable storage or of another word or words within the same block as the instruction, and indirect addressing which uses another word in the same block as the source of the address actually required. Any interpreted instruction may be translated by the compiler into the form for the latter indirect addressing mode, simply by placing an asterisk after the instruction code. Specific action of the compiler in this and other cases is described for interpreted orders in a paragraph below. The

indirect addressing specified here should not be confused with that normally found in many advanced computers, in which any word in storage may be used as the source of an indirect address. Here, the use is really quite limited, and is intended simply as a crutch to enable any instruction in fixed storage to have reference to any other fixed storage location.

The 15-bit word is divided into a 6-bit order code (bits 15-10) and a 9-bit address (bits 9 -1). If bit 15 is zero, the address is taken directly from bits 9 - 1, that is, the address is an address in erasable storage. If bit 15 is one, bit 9 is used to indicate whether direct (bit 9 = 0) or indirect (bit 9 = 1) addressing is intended for the other eight bits (8 - 1). For direct addressing, bits 8 - 1 are combined with a four-bit bank indicator in digit positions 12 - 9 whose value corresponds to the beginning address of the particular block of 256 words containing the instruction. The resultant address is then used directly as the address of the operand. If indirect addressing is used, the address formed as above is used as the location from which the address of the operand is taken.

It is noted in passing that action of the interpreter program changes the bank indicator (i. e., the work indicating in bits 12 - 9 the block number of the present instruction) only when an ITC or an active IBMN order is encountered. It is therefore not possible for the program to proceed continuously from the last word in one 256 word block to the first word in the next without risking an error; an ITC order should be supplied to pass between blocks. This remark does not apply to the eight basic computer instructions. Further, the bank indicator is set each time upon entering the interpretive mode, so that block boundaries may be freely crossed by basic orders.

INDEXING

The interpretive instruction INI (interpretive index) works, for interpretive orders, much the same as the INDEX order does for basic 3C instructions. Specifically, the INI order causes the contents of the designated address to be added to the order following the INI instruction before execution. In the special case

of indexed instruction with indirect addressing, the indexing part of the operation occurs last, that is, after the indirect addressing is accomplished. Note that indexing works on the address part of the word only. Attempted arithmetic with the order code via the INI order will lead to error. To execute an instruction made up by arithmetic processes, the interpretive DO order is provided.

ADDRESSING EXAMPLES

(All numbers are octal)

Address	Order	Interpretation
2517	35271	Operation 35 on E-address 271.
2517	35671	Operation 35 on E-address 671.
2517	75271	Operation 35 on address B + 271. B = 2400 = bank indicator for the address (2517) of this order.
2517	75671	Operation 35 using the address (5004)
2671	05004	which is found in register 2671
2516	INI 53	This combination causes operation 35 to
2517	75671	be executed with the address 5004 +C(53)
2671	05004	

COMPILER

The instruction **PQR** ADDR is translated as follows by the compiler, whenever **PQR** is an interpreted order.

(1) If the mnemonic symbol ADDR has a value less than octal 1000, it is interpreted as an ordinary E-address and its value is added in directly to the order code.

(2) Otherwise:

(a) The compiler checks that ADDR is in the same 256 word block as the order. Alarm if not. If so, the low-order 8 bits are used and a 1 is put into bit 15.

(b) Bit 9 is 0 or 1 according to whether the operation code PQR is not or is followed by an asterisk to denote indirect addressing.

(3) Use of an asterisk in case (1) above causes an alarm. For noninterpreted orders, the value of the mnemonic address ADDR is directly added to the 3-bit order code with no further ado.

Before listing the individual instructions and their interpretations, a small number of conventions used in this listed should be noted. The code number identifying each instruction is the octal version of the five-digit order code to be found in digit positions 14 to 10. Immediately following the code is the mnemonic symbol for the instruction. The letter X is used uniformly to represent the address part of the word. Immediately following the name of the instruction in words, is listed the normal or average number of instruction times required for the instruction, together with the maximum number of instruction times if it differs from the average number. For simplicity, the times listed correspond to direct addressing of erasable storage. To these must be added approximately 3 instruction times for direct addressing of fixed storage, and approximately 10 instruction times for indirect addressing of fixed storage. The letter L is used uniformly to denote the location or address from which the present instruction has been taken. Unless otherwise noted, the next instruction in sequence is taken from register L + 1. Finally, it should be remarked that unless otherwise noted, the contents of the registers DAC, IQ, VAC, and VBUF, are left unaltered.

The number system used for the double-and triple-precision arithmetic in the following instructions is one in which the binary point is at the left-hand side of the register DAC, that is, between digits 14 and 15. Thus the number which is contained in the registers DAC to DAC +2 is given by

$$c(DAC) + 2^{-14}\ c(DAC + 1) + 2^{-28}\ c(DAC + 2)$$

Sign agreement among the three numbers is not postulated; that is to say registers DAC and DAC +1 may have opposite signs, as may DAC + 1 and DAC +2. A special subroutine is available which can force sign agreement if required for special applications. However,

none of the interpretive instructions require that their operand have this characteristic. The programmer must be on his guard to avoid expecting such sign agreement in his results, if he is to make special use of the individual parts of an extra-precision number.

Code 00, ITC X, Interpreted Transfer Control to X, 35 I.

Action: Next interpreted operation taken from X. c(IQ) set to the 12-digit address L + 1.

Comments: The instruction ITC IQ causes the next interpreted instruction to be taken from the address stored in IQ, leaving IQ set to the address 0.

Code 01, IBMN X, Interpreted Branch Minus to X, 35 I ave., 44 I max.

Action: Test DAC. If zero, test DAC + 1. If zero, test DAC +2. For the first nonzero number found, test sign. If plus, or if all three numbers are zero, take next interpreted instruction from L + 1. If minus, execute ITC to address X.

Comments: +0 and - 0 are equivalent here. For non-branch condition, IQ is undisturbed; it is set as in ITC order when branch is taken. Care should be taken in double-precision operations not to branch inadvertently on a leftover c(DAC + 2) when the two high-order words are zero. When DAC itself is nonzero, 38 I are required for for c(DAC) negative and 31 I for the positive case.

Code 02, INI X, Interpreted Index by X, 33 I.

Action: The next instruction executed is c(L + 1) with the address part of the instruction modified by addition of c(X). The next succeeding instruction is taken from L + 2.

Comments: Only the address part of the instruction is modified by adding c(X). Any attempt to do arithmetic with the order code part of the word will lead to confusion and probable error. Details of how the indexing process is carried out can be obtained by examination of the program in Appendix VI. If the addressing operation in c(L + 1) is indirect, the indexing occurs on the final address.

Code 03, DO X, Do Single Instruction at X, 26 I.

Action: The next operation is taken from X. The operation immediately following is taken from L + 1, unless c(X) is an ITC or an active IBMN instruction.

Comments: The purpose of this instruction is to provide the capability for executing a single order built up by an arithmetic process in erasable storage, or filled in by another program.

Code 04, DCA X, Double Precision Clear and Add X, 40 I.

Action: c(X) into DAC, c(X + 1) into DAC + 1, + 0 into DAC + 2.

Code 05, DCS X, Double Precision Clear and Subtract X, 37 I.

Action: -c(X) into DAC, -c(X+1) into DAC + 1, + 0 into DAC + 2.

Code 06, DAD X, Double Precision Add X, 55 I ave., 58 I max.

Action: c(X, X + 1) + c(DAC, DAC + 1) into (DAC, DAC + 1). c(DAC + 2) is left unaltered.

Code 07, DSU X, Double Precision Subtract X, 63 I ave., 66 I max.

Action: -c(X, X + 1) + c (DAC, DAC + 1) into (DAC, DAC + 1).c (DAC + 2) is left unaltered.

Code 10, DTS X, Double Precision Transfer to Storage X, 37 I.

Action: c(DAC) into X, c(DAC + 1) into X + 1. c(DAC + 2) left unaltered.

Comments: Note also the instruction TTS available for transferring all three registers to storage.

Code 11, DXCH X, Double Precision Exchange with X, 38 I.

Action: c(X) and c(DAC) are exchanged, as are c (X + 1) and c (DAC + 1), with c (DAC + 2) left unaltered.

Code 12, DMP X, Double Precision Multiply by X, 141 I ave., 145 I max.

Action: The product c (X, X + 1) c(DAC, DAC + 1) is written as a triple precision answer in DAC, DAC + 1, and DAC + 2. If the two factors are exact, the result is precise to within 1 in the lowest order significant digit in DAC + 2.

Code 13, DDV X, Double Precision Divide by X, Approx 780 I ave.

Action: c(DAC, DAC + 1) is divided by c(X, X + 1), the result being left in DAC, DAC + 1.

Comments: The numerator must be less than the denominator in order to prevent erroneous results. No hang-up of the computer or loop of indefinite duration will result from violating this constraint, however. An attempt to divide by 0 will produce 0 as a result with no further serious consequences.

Code 14, TSLT X, Triple Shift Left by X, (38 + 16 X) I.

Action: The entire triple accumulator, DAC to DAC + 2, is considered as a single 42-bit register. Contents of this register are shifted left by X binary positions, with digits disappearing from the left hand edge of the register without overflow indication, and with 0 replacing the low-order digit positions.

Comments: This operation is conducted without forcing sign agreement among the three registers. The means by which it is accomplished is by successive doubling of the contents of the 42-bit register by adding it to itself, the lowest order words being added first, and the highest order being added at the end, with overflows propagating from the lowest order towards the highest order words. Overflows resulting through addition in the highest order position are ignored. The overall result is the same as though

the three words were first forced to share a common sign, and the entire 42-bit word then shifted as a single entity, with zeros filling in the lower order digit positions for positive numbers, or ones filling in these digit positions for negative numbers. The result is left in a condition without sign agreement.

Code 15, TSRT X, Triple Shift Right by X, (75 + 5X) I.

Action: The number X must lie between 1 and 14 inclusive. The double-precision number in DAC, DAC + 1 is multiplied by 2^{-X}. The result is then stored as a triple-precision number, in DAC to DAC + 2.

Comments: The result is the same as though the double-precision number in DAC, DAC + 1 were shifted to the right by X binary places, with the lowest order digits moving into the high order digit positions of DAC + 2, the rest of this register being clear.

Code 16, SSP X, Set Single Precision Number in X, 35 I.

Action: c(L + 1) stored in register X. Next operation taken from register L + 2.

Comments: The purpose of this instruction is to permit single words in storage to manipulated by the interpreted instructions, for use as counters, indexing words, and related operations.

Code 17, INCR X, Increment by X, 39 I.

Action: c(L + 1) + c(X) stored in X and in DAC. Registers DAC + 1 and DAC + 2 set to zero. Next operation taken from L + 2.

Comments: This instruction complements the action of SSP above, permitting the incrementing of the single register and at the same time setting in DAC the incremented result to be tested by an instruction such as IBMN. It is noted in passing that c(L + 1) may be any word whatever, for both the SSP and INCR instructions. Thus whole instructions can be modified in erasable storage, for example, for use by the DO instruction.

Code 20, ITA X, Interpreted Transfer Address to X, 34 I.

Action: c(IQ) into register X.

Comments: The quantity stored is a 12-digit address, and can be recovered as the return address of a subroutine by the pair of instructions INI X, and ITC 0. An error would generally result by using the instruction ITC X, since c(X) contains part of the address in bits 12-10.

Code 21, TCS X, Triple Precision Clear and Subtract X, 39 I.

Action: -c(X, X + 1, X + 2) stored in (DAC, DAC + 1, DAC + 2).

Code 22, TAD X, Triple Precision Add X, 68 I ave., 73 I max.

Action: c(X, X + 1, X + 2) + c(DAC, DAC + 1, DAC + 2) into (DAC, DAC + 1, DAC +2).

Code 23, TTS X, Triple Transfer to Storage X, 41 I.

Action: c(DAC) into X, c(DAC + 1) into X + 1, c(DAC + 2) into X + 2.

Code 24, Available.

Code 25, Available.

Code 26, VCA X, Double Precision Vector Clear and Add X, 128 I.

Action: c(X) to c(X + 5) transferred into the vector accumulator, VAC to VAC + 5.

Code 27, VXCH X, Double Precision Vector Exchange with X, 120 I.

Action: c(X) to c (X + 5) exchanged with c (VAC) to c(VAC + 5).

Code 30, VTS X, Double Precision Vector Transfer to Storage X, 120 I.

Action: c(VAC) to c(VAC + 5) transferred to registers X to X + 5.

Code 31, VCS X, Double Precision Vector Clear and Subtract X, 129 I.

Action: The negative of the vector stored in registers X to X + 5 is transferred to registers VAC to VAC + 5.

Code 32, VAD X, Double Precision Vector Add X, 120 I ave., 128 I max.

Action: The double-precision vector stored in registers X to X + 5 is added to the double-precision vector stored in the vector accumulator, the result being left in the vector accumulator.

Code 33, VSC X, Double Precision Vector Times Scalar X, 352 I ave., 364 I max.

Action: The vector quantity in the vector accumulator is multiplied by the double-precision quantity in registers X and X + 1. The result is left in the vector accumulator. The multiplication here is the same as in the double-precision multiply operation, as is that used in the cross-product and in the two matrix vector operations.

Code 34, DOT X, Double Precision Vector Dot Product with X, 448 I ave., 466 I max.

Action: The dot product of the vectors contained in registers X to X + 5 and VAC to VAC +5 is formed and left in registers DAC to DAC + 2. The multiplication is the same as in the double-precision multiply operation; however, the sum is accumulated in triple-precision arithmetic.

Comments: The purpose of the triple-precision accumulation is a matter of convenience in scaling. In particular, it is possible by this means to take a reasonably accurate absolute value of a relatively small vector quantity, without first re-scaling the vector.

Code 35, CROSS X, Double Precision Vector Cross Product with X, 881 ave., 914 I max.

Action: The double-precision vector cross product is formed of the vector contents of X times the vector in the vector accumulator. The result is left in registers VBUF to VBUF + 5 and in the vector accumulator.

Comments: Note specifically that the vector contents of X premultiply the vector accumulator.

Code 36, MXV X, Double Precision Matrix Times Vector, 1400 I ave., 1460 I max.

Action: Contents of the vector accumulator are premultiplied by the double-precision matrix stored in registers X to X + 17, the resulting vector being left in the vector accumulator. Contents of registers VBUF to VBUF + 5 are altered in the process.

Code 37, VXM X, Double Precision Vector Times Matrix, 1400 I ave., 1460 I max.

Action: Contents of the vector accumulator are postmultiplied by the double-precision matrix stored in registers X to X + 17, the resulting vector being left in the vector accumulator. Contents of registers VBUF to VBUF + 5 are altered in the process.

APPENDIX VI
ILLUSTRATIVE 3C PROGRAM IN YUL LANGUAGE

```
R0005     ERASABLE REGISTER ASSIGNMENTS
R0010     -------- -------- -----------

R0015     THE FOLLOWING REGISTERS ARE DEFINED BY INFERENCE...A, Q, N,
R0020               EDH, OVCTR, LP

 0025    LOC       =       60
 0030    IQ        =       61
 0035    IND       =       62
 0040    DAC       ERASE   63 +2
 0045    VAC       ERASE   66 +5
 0050    VBUF      ERASE   74 +5
 0055    BUF       ERASE   102 +1
 0060    TEM3      =       104
 0065    BANK      =       105
 0070    TEM2      =       106
 0075    TEM4      =       107
 0080    TAG       ERASE   110 +4
 0085    ADDRWD    =       115
 0090    TEM5      =       116
 0095    TEM6      =       117
 0100    TEM7      =       120
 0105    TEM8      =       121
 0110    INTLOCK   =       122
```

```
R0115    INTERPRETER
R0120    ----------

 0125            LOC    1040

 0130    INTPRET XCH    Q              ENTRY
 0135            TS     LOC            RE-ENTRY FOR ITC

 0140            CS     A              COMPUTE CURRENT BANK INDICATOR
 0145            TS     N
 0150            XCH    MSK1
 0155            ORN                   LOGICAL ADD MSK1 INTO N
 0160            CS     N
 0165            TS     BANK
 0170            TC     INT1 +3

 0175    MSK1    OCT    00377

 0180    IBMN1   CCS    DAC
 0185            TC     INT1
 0190            TC     +2
 0195            TC     ITC1
 0200            CCS    DAC +1         IF HIGH ORDER IS +0 OR -0.
 0205            TC     INT1
 0210            TC     +2
 0215            TC     ITC1
 0220            CCS    DAC +2         IF BOTH HIGH ORDER WORDS ARE ZERO,
 0225            TC     INT1           TEST DAC +2.
 0230            TC     INT1
 0235            TC     ITC1

 0240    INT1    XCH    ONE            NORMAL RE-ENTRY
 0245            AD     LOC
 0250            TS     LOC
```

```
0255            CCS     TAG             TEST FOR INTERPRETIVE INTERRUPT
0260            TC      IRPT            IF (TAG) NON-ZERO, GO TO IRPT

R0265           FOR NO INTERPRETIVE INTERRUPT REQUEST, GO ON WITH NEXT LINE.

0270            INDEX   LOC             C(LOC) = TC  NEXT OP.  = TC   N.
0275            CS      0
0280            TS      EDH
0285            TS      TEM2
0290            CS      EDH
0295            AD      CON1            DELETE ONES FROM DIGITS 14-6.
0300            XCH     TEM2
0305            TS      N

0310            CCS     A               +0 IS IMPOSSIBLE HERE

0315            XCH     MSK3            F-STORAGE
0320            TC      INT2
0325            TC      +3              E-STORAGE

0330            INDEX   LOC             EXIT INSTRUCTION (TC 0) SHOWS UP HERE
0335            TC      1                 EXIT IN BASIC TO LOC+1.

0340            XCH     MSK2
0345            ORN                     DELETE DIGITS 15-10 IN N.

0350            CS      N               POSITIVE 9-BIT E ADDRESS.
0355            TS      ADDRWD
0360    INT3    XCH     ZERO
0365            XCH     IND
0370            AD      ADDRWD
```

```
0375            TS      ADDRWD
0380            INDEX   TEM2
0385            TC      BR              IF SIGN IS MINUS, TC BECOMES CS.
0390            CS      A
0395            TCAA                    C(BR + N) IS ITSELF A TC ORDER.

0400    IRPT    CCS     INTLOCK         TEST IF ALREADY IN INTERPRETIVE
0405            TC      INT1 +5           INTERRUPT MODE.
0410    +2      XCH     ZERO            INITIATE INTERPRETIVE INTERRUPT
0415            XCH     TAG +4
0420            XCH     TAG +3          REGISTERS TAG +N STORE A PRIORITY
0425            XCH     TAG +2          SEQUENCE OF TRANSFER CONTROL ORDERS,
0430            XCH     TAG +1          SET UP BY ZERO-LEVEL INTERRUPT PROG-
0435            XCH     TAG             RAMS FOR LATER ACTION.  THAT IN TAG
0440            TS      INTLOCK         ITSELF IS NOW TO BE EXECUTED.
0445            TCAA

0450    MSK2    OCT     77000
0455    MSK3    OCT     77400

0460    INT2    ORN                     FIXED STORAGE CASE
0465            CS      N               POSITIVE 8-BIT ADDRESS
0470            AD      BANK
0475            TS      ADDRWD
0480            CCS     TEM2
0485            TC      INT3            TO INT3 FOR DIRECT ADDRESSING
0490    CON1    OCT     40037           +0 NOT POSSIBLE HERE.
0495            TC      +2              FOR INDIRECT ADDRESSING
0500            TC      INT3            -0 RESULTS FOR ITC ORDER WITH D9 = 0.

0505            INDEX   ADDRWD          FORM INDIRECT ADDRESS
0510            CS      0
0515            CS      A
0520            TC      INT3 -1
```

```
R0525    INTERPRETIVE BRANCH TABLE

 0530    BR      TC      ITC1            CODE 0  = ITC, INTERPRETIVE TC ORDER
 0535            TC      IBMN1           CODE 1  = IBMN, DP BRANCH MINUS
 0540            TC      INI1            CODE 2  = INI, INTERPRETIVE INDEX
 0545            TC      DO1             CODE 3  = DO, EXECUTE SINGLE ORDER
 0550            TC      DCA1            CODE 4  = DCA, DP CLEAR AND ADD
 0555            TC      DCS1            CODE 5  = DCS, DP CLEAR AND SUBTRACT
 0560            TC      DAD2            CODE 6  = DAD, DP ADD
 0565            TC      DSU1            CODE 7  = DSU, DP SUBTRACT
 0570            TC      DTS1            CODE 10 = DTS, DP TRANSFER TO STORAGE
 0575            TC      DXCH1           CODE 11 = DXCH, DP EXCHANGE
 0580            TC      DMP2            CODE 12 = DMP, DP MULTIPLY
 0585            TC      DPDIV           CODE 13 = DDV, DP DIVIDE
 0590            TC      SHIFTL          CODE 14 = TSLT, TRIPLE LEFT SHIFT
 0595            TC      SHIFTR          CODE 15 = TSRT, TRIPLE RIGHT SHIFT
 0600            TC      SETONE          CODE 16 = SSP, SET SINGLE PRECISION
 0605            TC      INCRMT          CODE 17 = INCR, INCREMENT SINGLE PREC
 0610            TC      ITA1            CODE 20 = ITA, INT. TRANSFER ADDRESS
 0615            TC      TCS1            CODE 21 = TCS, TRIPLE CLEAR AND SUBTR
 0620            TC      TRAD            CODE 22 = TAD, TRIPLE PRECISION ADD
 0625            TC      STORE3          CODE 23 = TTS, TRIPLE TRANS TO STOR.
 0630            TC      0               CODE 24 (AVAILABLE)
 0635            TC      0               CODE 25 (AVAILABLE)
 0640            TC      VCA1            CODE 26 = VCA, DPV CLEAR AND ADD

R0645                    NOTATION DPV = DOUBLE PRECISION VECTOR.

 0650            TC      VXCH1           CODE 27 = VXCH, DPV EXCHANGE
 0655            TC      VTS1            CODE 30 = VTS, DPV TRANSFER TO STOR.
 0660            TC      VCS1            CODE 31 = VCS, DPV CLEAR AND SUBTRACT
```

```
0665            TC      VAD1            CODE 32 = VAD, DPV ADDITION
0670            TC      VSC1            CODE 33 = VSC, DP VECTOR TIMES SCALAR
0675            TC      DOT1            CODE 34 = DOT, DPV DOT PRODUCT
0680            TC      CROS1           CODE 35 = CROSS, DPV CROSS PRODUCT
0685            TC      MXV1            CODE 36 = MXV, DP MATRIX TIMES VECTOR
0690            TC      VXM1            CODE 37 = VXM, DP VECTOR TIMES MATRIX

0695    TCS1    INDEX   ADDRWD          TRIPLE CLEAR AND SUBTRACT ORDER
0700            CS      2
0705            TC      DCS1 +1

0710    DCS1    XCH     ZERO            DOUBLE CLEAR AND SUBTRACT
0715            TS      DAC +2            (DCS CLEARS DAC +2)
0720            INDEX   ADDRWD
0725            CS      1
0730            TS      DAC +1
0735            INDEX   ADDRWD
0740            CS      0
0745    EX3     TS      DAC
0750            TC      INT1

0755    STORE3  CS      DAC +2          TRIPLE PRECISION TRANSFER TO STORAGE
0760            CS      A
0765            INDEX   ADDRWD
0770            TS      2
0775    DTS1    CS      DAC             DOUBLE PRECISION TRANSFER TO STORAGE
0780            CS      A
0785            INDEX   ADDRWD
0790            TS      1
0795            CS      DAC
0800    EX4     CS      A
0805            INDEX   ADDRWD
0810            TS      0
0815            TC      INT1
```

```
0820    ITC1    CS      ADDRWD          TEST WHETHER ADDRESS IS IQ
0825            AD      TCIQ
0830            CCS     A
0835            TC      NOTIQ           TO NOTIQ IF ADDRESS IS NOT IQ.
0840    TCIQ    TC      IQ              (REGISTER NOT USED IN CCS)
0845            TC      NOTIQ
0850            XCH     IQ              IF ADDRESS IS INDEED IQ
0855            TC      INTPRET +1      LEAVES +0 IN IQ

0860    NOTIQ   XCH     ADDRWD          ADDRESS IS NOT IQ
0865            XCH     LOC
0870            AD      ONE
0875            TS      IQ
0880            TC      INT1 +3

0885    INI1    INDEX   ADDRWD          INTERPRETIVE INDEX ORDER.  SET C(IND)
0890            CS      0                 TO THE VALUE IN THE REGISTER WHOSE
0895            COM                       ADDRESS IS IN ADDRWD.
0900            TS      IND
0905            TC      INT1

0910    DO1     INDEX   ADDRWD          INTERPRETIVE DO INSTRUCTION.
0915            CS      0                 EXECUTE OUT OF ORDER THE ISOLATED
0920            TC      INT1 +5           INSTRUCTION WHOSE ADDR IS IN ADDRWD

0925    DXCH1   XCH     DAC +1          DOUBLE PRECISION EXCHANGE ORDER
0930            INDEX   ADDRWD
0935            XCH     1
0940            XCH     DAC +1
0945            XCH     DAC
0950            INDEX   ADDRWD
0955            XCH     0
0960            TC      EX3
```