THE MICROCODE FOR THE CONTROL PROCESSOR OF THE ARO ARRAY PROCESSOR

D.J. HEILBRONN

Approved for Public Release

COPY No. 12

Commonwealth of Australia
AUGUST 1983
THE MICROCODE FOR THE CONTROL PROCESSOR OF THE ARO ARRAY PROCESSOR

D.J. Heilbronn

SUMMARY

A high speed array processor (ARO) has been designed for the processing of radar data in real time. The operation of the ARO is supervised by a bit-slice microprocessor (the Control Processor). This document contains a description of the microcode which was written for the Control Processor to enable it to interpret PDP-11 assembler code.

POSTAL ADDRESS: Director, Electronics Research Laboratory, Box 2151, GPO, Adelaide, South Australia, 5001.
# TABLE OF CONTENTS

1. INTRODUCTION .......................... 1
2. SAMPLE OF MICRO-ASSEMBLER OUTPUT ............. 1
3. THE MICRO-INSTRUCTION FORMAT ................. 2
   3.1 Field description ..................... 3
4. MICROP DEFINITIONS ....................... 3
5. ASSEMBLER SYNTAX ......................... 3
   5.1 ALU function .......................... 4
   5.2 Source register definition ............... 4
   5.3 Destination register definition ......... 4
   5.4 MAR, Q register definition .............. 5
   5.5 The C2904/shift control definition .... 5
   5.6 USR and MSR control .................... 5
   5.7 Transfer type .......................... 5
   5.8 Memory control ........................ 6
   5.9 Loading of immediate field .............. 6
   5.10 Comments ................................ 6
   5.11 Terminator ............................ 6
   5.12 Default field values ................... 6
6. THE PROGRAM SECTION ....................... 7
   6.1 The symbol definition section .......... 7
6.2 The subsidiary routines ...................... 7
   6.2.1 Address evaluation .................. 7
6.3 PDP-11 instruction section ................... 8
   6.3.1 Class Ml (miscellaneous) .......... 8
   6.3.2 Class JUMP, RTS and CC ........... 8
   6.3.3 Class SWAB, BR and JSR ............ 8
   6.3.4 Class SO1, SO2, SO3, SO4, SO5 .... 8
   6.3.5 Class DO1, DO2, DO3, DO4 ......... 9
   6.3.6 Class EIS ................................ 9
   6.3.7 Class BRS and TR .................... 10
   6.3.8 Class SO6, SO7, SO8, SO9, SO10 .. 11
   6.3.9 Class DO5, DO6, DO7, DO8 .......... 11
1. INTRODUCTION

The Control Processor is one element of the Array Oriented Processor (ARO) which has been designed within ERL. An overview of the operation of the ARO is provided elsewhere (ref.9). The ARO works in conjunction with a host PDP 11/34 to provide high speed processing. The host delegates processing to the ARO by transferring an operating system to it, scheduling tasks and initiating it to run autonomously. The ARO has its own inbuilt processor called the Control Processor (CP) which basically serves to run the operating system and control the operation of the remainder of the ARO.

The design philosophy of the ARO is described elsewhere (ref.3,9). This document basically serves to provide sufficient detail of the microcode written for the CP to allow the microcode to be maintained.

The PDP-11 assembler language was chosen as the target language for the CP to be compatible with the host processor. The design philosophy of the CP by necessity took into account the nature of the PDP-11 instruction set. The ground rules followed in implementing the microcode were established by Mr. J. Zuikelis in reference 3. A full description of the hardware of the CP is provided in reference 1. The CP is based on the Am2900 family of bit-slice microprocessor products. The principles involved in designing and microprogramming with the Am2900 family are explained in reference 7.

The CP emulates a PDP-11/34 by reading object code which is stored in the CP main memory into an instruction port (see figure 1). This PDP-11 instruction is fed to a mapping PROM which yields a microcode starting address within the microcode store. The CP then executes that microcode routine which manipulates the CP main memory in a manner identical to a PDP-11/34.

The microcode was generated using the Signetics Micro-assembler (ref.2) which is available on the IBM 370 computer which is installed at DRCS. The micro-assembler enables the user to define the micro-instruction word format and the mnemonics to suit the application. The source code (input to the micro-assembler) consists of three distinct sections - the micro instruction format section, the micro definition section (similar to macro definitions) and the program section. In order to fully appreciate the contents of this document, the reader will have to be familiar with the hardware description of the Control Processor (ref.1,3), the micro-assembler description (ref.2) and the PDP-11 instruction set (ref.8). Within these constraints, Section 5 is provided as a guide to enable the user to modify the microcode. It is anticipated that modification will only be required in the program section with the syntax and micro-instruction format sections remaining unchanged.

The complete listing of the micro-assembler output is shown in Appendices I, II and III. Instructions for using the IBM 370/3033 to generate the microcode are given in Appendix IV.

2. SAMPLE OF MICRO-ASSEMBLER OUTPUT

A sample of the micro-assembler output with the 64 bit microcode interspersed with the source input is shown in figure 2. This section depicts the fetch cycle which reads the PDP-11 code into the instruction port and the single operand PDP-11 instructions CLR, COM, INC, DEC, NEG, ADC and SBC. Note that the start address in the microcode for the fetch cycle is 31 (octal) and that the start address in the microcode of CLRI is 374 (octal). The mnemonic NEXT causes the CP to jump to the start of the fetch cycle.
Figure 2 may be used to illustrate the operation of the CP as follows. On initialisation, the micro-program counter is forced by the host to the start of the fetch cycle of the microcode. Assume that the macro-program counter (equivalent to the PDP-11 register R7) is pointing to a location in CP main memory containing the machine code for:

\[
\text{CLR RO } \quad ; \text{PDP-11 code for clear reg 0}
\]

The execution of the fetch cycle would cause the micro-instructions 31, 32, 33 and 374 to be invoked, followed by another fetch cycle. The macro-program counter would be updated to point to the next PDP-11 instruction in CP main memory.

3. THE MICRO-INSTRUCTION FORMAT

The micro-instruction format is identical to that described in reference 2. The fields (see Table 1) are defined as SRC REG, DST REG, MAR, ALU CONTROL, 2904 STATUS & SHIFT, MICROSEQUENCE CONTROL, SPECIAL OPERATIONS FIELD, IMMEDIATE FIELD and an ERROR FIELD to record assembly time errors. The fields are further divided into sub-fields to link to the nomenclature of references 1, 3, & 4.

TABLE 1. MICRO-INSTRUCTION FORMAT

<table>
<thead>
<tr>
<th>FIELD</th>
<th>SUB-FIELD</th>
<th>WIDTH</th>
<th>COMMENT</th>
<th>DEFAULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC</td>
<td>AMODE</td>
<td>2</td>
<td>A-MODE SELECT ) SOURCE REGISTER</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>AFIELD</td>
<td>4</td>
<td>A-SOURCE SELECT )</td>
<td>0</td>
</tr>
<tr>
<td>DST</td>
<td>BMODE</td>
<td>2</td>
<td>B-MODE SELECT ) DESTINATION REG.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>BFIELD</td>
<td>4</td>
<td>B-SOURCE SELECT )</td>
<td>7</td>
</tr>
<tr>
<td>MARS</td>
<td></td>
<td>1</td>
<td>MEMORY ADDR. REG. SELECT</td>
<td>0</td>
</tr>
<tr>
<td>C2903</td>
<td>I1_4</td>
<td>4</td>
<td>ALU OPERATION SELECT</td>
<td>4(MOV_D)</td>
</tr>
<tr>
<td></td>
<td>I0</td>
<td>1</td>
<td>ALU OPERATION SELECT</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>I5_8</td>
<td>4</td>
<td>ALU DEST CONTROL, SPECIAL FUNC</td>
<td>0CH(TST)</td>
</tr>
<tr>
<td>C2904</td>
<td>I11_12</td>
<td>2</td>
<td>CARRY IN SELECT</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>16_9</td>
<td>4</td>
<td>SHIFT LINKAGE SELECT</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>10_5</td>
<td>6</td>
<td>STATUS REG UPDATE/ COND. TEST</td>
<td>44Q</td>
</tr>
<tr>
<td></td>
<td>EC</td>
<td>1</td>
<td>CARRY UPDATE ENABLE</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CEU</td>
<td>1</td>
<td>USR UPDATE ENABLE</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CEM</td>
<td>1</td>
<td>MSR UPDATE ENABLE</td>
<td>0</td>
</tr>
<tr>
<td>CCMUX</td>
<td></td>
<td>3</td>
<td>COND. CODE TEST SELECT</td>
<td>0(UNCOND)</td>
</tr>
<tr>
<td>C2910</td>
<td></td>
<td>4</td>
<td>MICROSEQUENCER CONTROL</td>
<td>0EH(CONT)</td>
</tr>
<tr>
<td>SOF</td>
<td></td>
<td>5</td>
<td>SPECIAL OPERATION FIELD</td>
<td>OFH</td>
</tr>
<tr>
<td>IMMOP</td>
<td>SLWDTH</td>
<td>4</td>
<td>MIR FIELD SELECTOR WIDTH</td>
<td>0</td>
</tr>
<tr>
<td>TADDR</td>
<td></td>
<td>12</td>
<td>TARGET ADDR. FOR DIRECT JUMP</td>
<td>0</td>
</tr>
</tbody>
</table>
3.1 Field description

The SRC field controls which register provides the source data to the ALU in double operand instructions. The DST field selects which register is read from in single and written to in single and double operand instructions. The MARS field controls the transfer of the address at the output of the ALU to the memory address register. The C2903 field selects the ALU operation to be performed and partially controls the ALU shifting operation, byte manipulations and writing of data to the destination. The C2904 field controls the carry-in to the ALU (I1_12), the linkages provided for shift and rotate instructions (I6_9), the status update and condition code testing (I0_5), and status register update inhibits (EC, CEU and CEM). The CCMUX field controls the feeding of conditional tests to the microsequencer. The C2910 field controls the program flow within the microcode. The SOF (special operations field) controls the interrupt processing, the CP main memory operations and the machine halt. The IMMOP field is used to provide the ALU with data and addresses which may be resolved at assemble-time.

Default values are provided for each field and sub-field so that a complete specification of all fields is not necessary when writing microcode (see Appendix I). The default is not necessarily a null operation. For example, the default condition for the microsequencer (C2910) is CONT which increments the micro-program counter by one to point to the next micro-instruction. Details of the default conditions are described later (see Section 5.12).

4. MICROP DEFINITIONS

The microp definition section serves to define 'macros' which may be used in the program section. The ability to nest microps means that the coding of microps may be simplified. For example, the addressing microp is invoked by the symbol @ within the ALU microps (see Appendix II). Thus the micro-assembler expects a list of addresses to follow the use of an ALU microp.

The microp definition section has been organised in the same order as the micro-instruction format. Thus the addressing register select is followed by ALU control, 2904 control, microsequencer control, special operations field control and lastly, immediate field control.

Where possible the mnemonics suggested in the ARO PROCESSOR DESIGN NOTES(ref.3) have been used. The ALU functions have been redefined to reflect the source (S) and destination (D) structure of the micro-instruction format. The special function mnemonics of the 2903 have been redefined to more closely reflect their operation as far as PDP-11 programmers are concerned.

5. ASSEMBLER SYNTAX

In essence, the microp definition section defines the syntax of the micro-assembler. The syntax of the program section is outlined in figure 3 with details being provided in the following sub-sections. To generate the desired microcode, one of the alternatives listed in figure 3 for each field may be selected. Figure 4 provides a list of the mnemonics recognised in the program section. The seldom used fields in the micro-instruction format are directly specified in the form FIELD=xxxxx. Labels are used extensively to make the microcode more readable and are of the form LABEL: which must be the first
entry on the line. Wherever possible the labels are derived from the PDP-11 assembler mnemonics. All labels which are left justified on the page (see Appendix III) must appear in the mapping PROM.

5.1 ALU function

The Am2903 ALU operation is divided into two classes of operation - standard functions (ALU) and special functions (SPC_FN). Figure 4 shows the correspondence between the mnemonics employed in the program section and the manufacturer's description (ref.4).

5.2 Source register definition

The register definitions must immediately follow the ALU or SPC_FN operation. Address operands may be omitted but their order must be preserved by inserting commas (e.g., RS,,MAR indicates the destination register is not required). Trailing operands may be omitted.

Details of the source register mnemonics are shown in figure 4. The mnemonics are on the left hand of the source register columns with the corresponding micro-instruction field values on the right. Registers R0 to R7 correspond to the PDP-11 registers. Registers R8 to R10 are reserved for later use in coding the routines to control the ARO. Registers ZERO, ONE and TWO hold the constants indicated by their names. Registers TEMP@ and TEMP are used as temporary registers within the microcode routines used to interpret each PDP-11 instruction.

The use of the symbol RS as a source register causes the addressing routine to use bits 6 to 8 of the instruction port to be used as the source register address. (This corresponds to the SS field of the PDP-11 instruction). Similarly, the use of RD as a source register causes bits 0 to 2 of the instruction port to be used as the source register address (the DD field).

The registers DRO, DR1 and DR2 are the input/output registers used to buffer data to the CP main memory. The register IMM corresponds to the micro-instruction field IMMOP and is used to enter immediate data (e.g., constants or target addresses) into the ALU. The use of PSW as a source register causes the processor status to be extracted from the C2904.

The symbol FLDSEL (OFFSET,WIDTH) causes portion of the instruction port to be used as the source register. The parameter OFFSET specifies the number of low order bits to be skipped. The parameter width defines the width in bits of the required field. This portion is then placed at the source input of the ALU with the unfilled high order bits set to zero.

5.3 Destination register definition

The destination register must immediately follow the source register definition delimited by a comma. The mnemonics used for the destination registers are shown in figure 4. Note that some of the micro-instruction field values for the source and destination registers are different even though the mnemonics are the same. This difference is due to the hardware configuration of the Control Processor.
The destination and source register mnemonics are the same with the exclusion of ZERO, ONE, TWO, MIR and QDST. ZERO, ONE and TWO are constants and thus are not available as a destination register. An attempt to use any of these three registers as a destination is trapped by the addressing microcode. The use of MIR causes the instruction port to be loaded with data (which may be a PDP-11 instruction) from the CP main memory. The address of this data is provided by the MAR register. The specification of QDST as a destination causes the Q register of the Am2903 to be loaded with the data at the output of the ALU.

5.4 MAR, Q register definition

The third register field controls the loading of the CP main memory address register (MAR) and the use of the Q register of the Am2903 as an alternative source. The symbol MAR causes the MAR register to be loaded with the address at the output of the ALU. The symbol QIN causes the Q register to be used as an alternative source to the ALU. Note that when QIN is invoked one source is specified by the RSN field, the destination is specified by DSN and the alternative source is the Q register. The symbol MARQ causes both MAR and QIN to be executed.

5.5 The C2904/shift control definition

This field controls the inhibiting of the register write cycle for the Am2903, and the Am2903 shifter operation and the corresponding linkages provided by the Am2904. Specifying TST inhibits the writing of data into the destination register. Note that this is the default condition if the destination register is not specified. STO and STOB cause a word and a byte, respectively, to be written into the destination register. The specification of an ALU or SPC FN function causes the invocation of the STO operation. BSX propagates the sign of the low order byte through the high order byte and stores the result. ROR, ROL and ASL perform a similar operation to that performed by PDP-11 instructions of the same name. However, the overflow bit is not necessarily identical. (The overflow bit is corrected by the microcode routine FIXCC - see Appendix III). Note that the Am2904 is not capable of directly simulating the ASR instruction (see microcode routine ASR3).

5.6 USR and MSR control

This group of instructions controls the manipulation of the status registers within the Am2904. The Am2904 contains two status registers - the micro status register (USR) and the machine status register (MSR). The USR is used to preserve the status of the condition codes at the microcode level. The MSR is used to preserve the status of the condition codes at the machine level. Thus the MSR contains the equivalent of the PSW of the PDP-11. The condition code bits are the sign (N) bit, the zero (Z) bit, the overflow (V) bit and the carry (C) bit.

The mnemonics MSR, MSRCl and MSRNC cause the ALU status to be stored in the MSR 'as is', with carry inverted, and with carry inhibited (NOCarry). The mnemonic MSRZ causes the MSR to be cleared. The mnemonic USR causes the ALU status to be stored in the USR. STOPSW causes the data at the output of the ALU to be stored in the MSR.

5.7 Transfer type

This group of instructions controls the operation of the microsequencer (C2910). Figure 4 shows the operations performed by the C2910 (see reference 4 for further details). The mnemonics JMP LABEL and JSR LABEL cause an unconditional jump (CJIP UNCOND) and unconditional jump to
subroutine (CJS UNCOND), respectively. If the keyword INDEXED is used as a label for the JMP or JSR instruction, then the ALU output is used as the target address. The mnemonic RETURN is used to perform an unconditional return from a subroutine (CRTN UNCOND). NEXT implements the fetch cycle which in turn executes the JMAP instruction of the C2910. TRAP causes the microcode to enter a routine which signifies that an illegal instruction code has been detected. All other C2910 instructions take the form TRANSFR COND with the appropriate mnemonics detailed in figure 4. The conditional test mnemonics are similar to those of the PDP-11 instruction set with 'U' signifying the USR and 'M' signifying the MSR.

5.8 Memory control

This group controls the reading from (READ, READB) or writing to (WRITE, WRITEB) the CP main memory. The character B appended to the mnemonic signifies that a byte memory control operation is to be performed. In addition, READC and WRITEC read and write from the cache memory.

5.9 Loading of immediate field

The mnemonic PRESET causes the micro-assembler to expect a constant which it places in the IMMOP field. This permits constants to be used as source values. The mnemonic TADDR causes the low 12 bits of the IMMOP field to be used as a target address for the microcode.

5.10 Comments

Comments are entered into the source lines by enclosing the comment in double quote marks.

5.11 Terminator

A semicolon signifies the end of each micro-instruction.

5.12 Default field values

Default values are provided for all the microword fields to simplify the writing of the microcode (see Table 1). The destination register defaults to R0 and writing to the destination register is inhibited by the addressing microp. For example, using the addressing sequence R1,,MAR inhibits the writing of data to the destination register. The 11_4 field of the C2903 defaults to the MOV,D instruction as this simply reads the data from the destination and writes it back in again. The 15-8 sub-field of the C2903 defaults to TST which inhibits the writing of data into the destination register.

The C2904 field controls the carry-in selection, the shift linkage selection, the status updating and the condition code testing. The carry-in select defaults to inhibiting any carry-in. The 10_5 sub-field defaults to inhibiting the update of the USR or the hSR. The 10_5 sub-field also defaults to the condition code test MEQ(44Q).

The CCMUX field defaults to the unconditional mode. The C2910 (microsequencer) field defaults to the CONT instruction which causes the micro program counter to be incremented by one. The SOF field defaults to the enable interrupt condition.
6. THE PROGRAM SECTION

The program section of the microcode (see Appendix III) consists of three sections - symbol definition, subsidiary routines and the PDP-11 instruction interpretation.

6.1 The symbol definition section

The first portion of the program section serves to define registers and miscellaneous symbols used in the remainder of the program section. All of these symbols were described in figure 4 with the exception of MC and DUMMY ADDR. MC is used to specify that the carry-in is to come from the MSR. DUMMY ADDR specifies a non-existent location in the CP main memory which is used in transferring data from the DRO register along the memory data bus to the instruction port (see REGDST: in the subsidiary routine section).

6.2 The subsidiary routines

This segment contains run-time routines which support the operation of the PDP-11 program section (see Appendix III). SAVE is called by the trap instructions (BPT, IOT, etc) to save the PSW and PC on the PDP-11 stack. ILGCL is used to trap illegal instruction codes and traps through location 10 in CP main memory. Note that FETCH conditionally tests the interrupt status. This is the only time that the interrupt status is interrogated so interrupts are only processed at the macro level. The interrupt processing is nested so that multiple interrupts can be handled satisfactorily.

6.2.1 Address evaluation

The next group of support routines fetches the operand for PDP-11 addressing modes other than mode=0. Separate addressing routines are needed for byte addressing and for evaluating source and destination addresses. Each of the addressing groups interprets mode=1 through to mode=7. DBASE evaluates word destination addresses and DBASEB byte destination addresses. SBASE evaluates word source addresses and SBASEB byte source addresses.

In the next group of support routines, RLGDSP transfers the source register value to the low order position of the instruction port. This is utilised in processing some of the PDP-11 extended instruction set codes. FIXCC is used to set the overflow bit to an identical state to the PDP-11 after arithmetic shift and rotate instructions.

6.3 PDP-11 instruction section

This portion interprets the PDP-11 instructions. It is arranged in order of increasing instruction value (eg HALT=000000 comes first). Labels which are left justified appear in the mapping PROMS. Any labels which start with @ are used only as a jump point within each PDP-11 instruction. Occasionally @ is used within a label to distinguish it from a previously defined symbol. Note that instructions are grouped into classes. The classes are selected on the basis of likeness in starting address decoding and likeness in interpretation. For example, the miscellaneous instructions are grouped together because the low order six bits of the instruction must be evaluated to resolve the starting address.
6.3.1 Class M1 (miscellaneous)

This group includes the HALT, WAIT, RTI, BPT, IOT, RESET and RTT instructions (see Appendix III). The high order ten bits of the PDP-11 instruction are used to determine the starting address of the microcode routine which interprets each instruction. Thus if the six low order bits are necessary to interpret the instruction then the field selector must be used to resolve the starting address within each class of instructions. Note that in this case, the microcode to interpret each PDP-11 instruction must occur at a specific microcode location relative to the associated JMP INDEXED instruction. Hence if the interpretation takes more than one micro-instruction, a second jump is necessary (eg HALT: and @HALT:). Also note that attempts to use the PDP-11 RESET instruction are trapped as this instruction has no significance as far as the CP hardware is concerned. TRAP causes control to be transferred to CP main memory location 10 (octal) which is the standard PDP-11 trap location for an illegal instruction.

6.3.2 Class JUMP, RTS and CC

These three groups of instructions interpret the JMP, RTS and the Condition Code instructions (CLN, CLZ, CLV, CLC, CCC, SEN, SEZ, SEV, SEC and SCC).

The JUMP instruction utilises the field selector to generate an offset address to resolve the various modes (see Appendix III.5). The jump address is then retrieved and replaces the old value in R7. All six jump modes are interpreted.

The RTS instruction uses the field selector to obtain the register number containing the return address. The stack in PDP-11 stack is then popped to retrieve the old register value.

The CC instruction group uses the field selector to prepare a mask based on the op-code. This mask is then used to clear or set designated bits. Note that some illegal instruction codes are possible within the CC group. These are detected and control transferred to ILGL.

6.3.3 Class SWAB, BR and JSR

The function of the Control Processor is to create a monitor to control the operation of the ARO. Hence there are instructions in the PDP-11 instruction set that are used infrequently. Most byte operations fit into this category and thus there is little hardware support for byte manipulation. This makes PDP-11 instructions such as SWAB rather cumbersome as the byte swap is performed by rotating left eight times (see Appendix III.6). Note that the word to be rotated is written into a temporary register so that the same rotate routine (ROT8) can be used with both register and memory resident data.

The branch instructions (BR, BNE, BEQ, BGE, BLT, BGT and BLE) and the JSR instructions are handled in the next two groups. The JSR instructions all require the manipulation of the PDP-11 stack which is held in the CP main memory. This manipulation is performed by the microcoded routine STACKO.

6.3.4 Class S01, S02, S03, S04, S05

This group of instructions (see Appendix III.7) comprise the Single Operand instructions. Classes S01 and S02 interpret CLR, COM, INC, DEC, NEG, ADC, SBC and TST instructions. Class S01 performs register
operations (viz mode=0) and with the exception of SBC they are
accomplished in one microcycle (plus the fetch cycles). Class S02
require memory operations (viz modes other than zero) and invokes the
microcode routine DBASE to obtain the destination address.

Classes S03 and S04 interpret ROR, ROL, ASR, ASL and SXT instructions.
PDP-11 arithmetic shift and rotate operations are a little awkward
to implement on the 2900 family. This arises due to the need to
determine the exclusive-or (XOR) of the N and the C bits after the
rotate operation. The Am2903 determines the condition codes before the
rotate/shift operation is performed so the codes must be determined on a
subsequent cycle. Also the Am2904 cannot directly perform a conditional
jump on testing (N XOR C).

The microcode routine FIXCC is used to evaluate the condition codes
after performing a shift or rotate function. The manipulation of the
overflow bit is performed in the USR in preference to the MSR as the USR
has individual bit control. In addition, the Am2903 is not capable of
'recycling' the sign bit as is required for ASR. These limitations slow
the speed of operation of the Control Processor when handling arithmetic
shift or rotate operations.

Class S05 includes the MARK instruction which tidies the PDP-11 stack on
returning from a subroutine.

6.3.5 Class D01, D02, D03, D04

This group of instructions comprises the Double Operand instructions
(see Appendix III.8). Classes D01, D02, D03 and D04 interpret MOV, CMP,
BIT, BIC, BIS and ADD. With the double operand instructions four
distinct classes arise as entry points are required to separately
process the occurrences of mode=0 for both the source and the
destination registers. Class D01 processes the instruction when modes
of both the source and the operand are zero. Class D02 processes the
instruction when the source mode is zero and the destination mode is not
zero. Similar arrangements apply for Classes D03 and D04.

6.3.6 Class EIS

Class EIS interprets the Extended Instruction Set (see Appendix III.9).
This set includes MUL, DIV, ASH, ASHC, XOR, FADD, FSUB, FMUL, FDIV and
SOB. However FADD, FSUB, FMUL and FDIV are not interpreted but are
trapped.

The MUL instruction is implemented in three stages. The pre-amble (up
to the SMUL operation) serves to place the multiplier in the Q register,
the multiplicand in TEMPQ, and 14 in the Am2910 loop counter. The
multiplication involves fifteen cycles of SMUL (Signed Multiplication)
and one cycle of FMUL (Final Multiplication). The FMUL operation serves
to adjust the partial product depending on the sign of the multiplier.
The post-amble places the 32 bit product in the designated register pair
and fixes the condition codes. The MUL instruction has two entry points
- one for mode=0 and the other for non zero modes.

The PDP-11 divide is integer in nature and to avoid the necessity of
normalizing the dividend and divisor and post-scaling the quotient and
remainder, a scheme similar to that of Rhyme(ref.5) was implemented.
Rhyme's work described the case for fractional arithmetic. In the
PDP-11 case (integer), the dividend (both most and least significant
haves) has to be shifted left arithmetically one position before the
division may proceed. After the division algorithm has been
implemented, Rhyne suggests a correction for the quotient and remainder based on the sign bits of the quotient, the remainder and the dividend. In the integer case, it is simpler to correct the quotient and remainder on the basis of the signs of the quotient (Q), the remainder (and the divisor (X)) as shown in Table 2.

**TABLE 2. QUOTIENT AND REMAINDER CORRECTION**

<table>
<thead>
<tr>
<th>Original signs</th>
<th>Required correction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q  R  X</td>
<td>Q’  X’</td>
</tr>
<tr>
<td>-  -  -</td>
<td>Q+1    R-X</td>
</tr>
<tr>
<td>-  -  +</td>
<td>Q       R</td>
</tr>
<tr>
<td>-  +  -</td>
<td>Q+1    R-X</td>
</tr>
<tr>
<td>+  -  -</td>
<td>Q       R</td>
</tr>
<tr>
<td>+  -  +</td>
<td>Q-1    R+X</td>
</tr>
<tr>
<td>+  +  -</td>
<td>Q+1    R-X</td>
</tr>
<tr>
<td>+  +  +</td>
<td>Q       R</td>
</tr>
</tbody>
</table>

The DIV instruction may be conveniently broken into three sections. The pre-amble places the divisor in DRO, invokes REGDST which places the address of the dividend (least significant half) in the DD field of the instruction (which is held in the instruction port). The pre-amble then places the divisor in TEMP and the dividend (most significant half) in TEMP@. These two values are then converted to sign magnitude representation to ensure that the divisor is larger than the dividend. Note that the dividend is shifted left prior to making this test to take the 31 bit precision of the dividend into account. The divide operation is then performed. NYZD is the first divide operation and it ascertains the sign bit of the quotient. The SBIV (Signed DIVide) operation is then executed fourteen times. The FDIV (Final DIVide) operation then adjusts the quotient by forcing the least significant bit to a one. The post-amble places the quotient and remainder in the appropriate registers and implements the correction outlined in Table 2. The condition code bits are forced to be identical to that of the PDP-11.

Due to the similarity of ASH and ASHC, both of these instructions are treated by the same routine. However ASHC uses the Q register to perform the double word shift. Firstly, the source operand is placed in TEMP@ and the shift direction extracted. Then the number of shifts is placed in the C2910 loop counter. The MIR is then examined to determine if the operation pertains to ASH or ASHC and the appropriate branch made. For ASHC, the routine REGDST is invoked to determine which PDP-11 register pair holds the data to be shifted. The appropriate rotation is then performed. In the case of ASHC, a post-amble (@ASHCC) is used to determine the Z bit.

**6.3.7 Class BRS and TR**

Class BRS includes the BPL, BMI, BHI, BLDS, BVC, BVS, BCC, BHIS, BCS and BLO instructions. These instructions are interpreted in a similar fashion to the class BR group. Class TR includes the EMT and TRAP instructions. These are implemented similar to BPT in the class M1 group.
6.3.8 Class S06, S07, S08, S09, S010

Classes S06, S07, S08 and S09 include all the Single Operand (byte) instructions. CLRB, COMB, INCB, NEGB, ADCB and TSTB are implemented by the S06 (mode=0) and S07 (other modes) groups. Note that the lower byte only is affected for register operations. DBASEB is used to extract the byte operand for memory resident operands. In this case, either a high or low byte may be specified, but the hardware places that byte in the low position in DRO. Classes S08 (mode=0) and S09 (other modes) implement the RORB, ROLB, ASRB and ASLB instructions. The operation of these groups is similar to groups S03 and S04. Group S010 implements the HTPS and MFPS instructions.

6.3.9 Class D05, D06, D07, D08

These classes include the Double Operand (byte) instructions and interpret MOVB, CMPB, BITB, BICB and BISB. The instructions are interpreted in a similar fashion to their word counterparts in classes D01, D02, D03 and D04.

6.3.10 Class D09, D010, D011, D012

These classes interpret the SUB instruction and are similar to the ADD instruction found in D01, D02, D03 and D04.

6.4 Indirect mapping

To avoid the necessity of reburning the mapping PROMs during the debug phase, the entry point for each macro-instruction is indirectly mapped. Direct mapping is applied to each class of instructions. This results in the need to re-burn the mapping PROMs only if the number of instruction classes is altered. The post-processing program CPROC (see Appendix IV) examines the listing file for left justified labels and creates a mapping entry for each such label. The effect of the indirect mapping is to add one micro-cycle to the instruction timing. When the microcode is considered stable the indirect mapping may be removed.

6.5 Special macro-instructions

The 40 bit facilities of the AR0 need to be accessible at the macro level. Special macro-instructions are included in the microcode to enable the reading and writing of the cache memory and performing 40 bit data dependent processing of cache. At the time of writing, the special macro-instructions had only been partially defined. This is the area in which future expansion of the microcode is anticipated.

The instruction codes 17xxxx are used for these special macro-instructions. To facilitate using this group as they are defined, the specials are organised into eight sub-groups decoded by the field selector corresponding to codes 170xxx to 177xxx. The special macro-instructions of Appendix III.14 are tentative in nature and are included only for the sake of completeness. The special macro-instructions will be the subject of a separate document in the future.
7. PROCESSOR PERFORMANCE

7.1 Limitations and exclusions

The entire basic instruction set of the PDP-11/34 has been microcoded with the exclusion of the RESET instruction. (The RESET instruction only pertains to the operation of the UNIBUS and hence is not relevant to the hardware configuration of the Control Processor).

7.2 Processor instruction timing

The execution time for an instruction depends on the instruction itself, and on the addressing modes. In the most general case, the Instruction Execution Time is the sum of a Fetch Time, a Source Address Time, a Destination Address Time, and an Execute Time.

\[
\text{Instr Time} = \text{Fetch Time} + \text{SRC Time} + \text{DST Time} + \text{Exec Time}
\]

Some of the instructions require only some of these times, and are noted. All instructions require the Fetch Time which takes 3 clock cycles, and an Exec Time which takes at least 1 cycle. The number of cycles taken by each of the instructions is shown in Tables 3 to 8. The time taken for each micro-cycle is 300 ns.

7.3 Validation

The correct execution of the microcode was established by running the microcode on an emulator. The emulator has the ability to accept PDP-11 source code and execute it. To prove the correctness of the microcode, the DEC diagnostic packages CPAF and CPBG(ref.6) were run on the emulator. These two packages test the in-line code and the extended instruction set respectively.

8. ACKNOWLEDGEMENTS

The author is indebted to Mr J. Ziukelis, Dr T. Hobbs and Mr P. Drewer for information and countless discussions on the operation of the hardware.
TABLE 3. TIMING FOR MISCELLANEOUS INSTRUCTIONS

<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>EXEC TIME (CYCLES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HALT</td>
<td>3</td>
</tr>
<tr>
<td>WAIT</td>
<td>4(incl 2 cyc loop)</td>
</tr>
<tr>
<td>RT1</td>
<td>8</td>
</tr>
<tr>
<td>BPT</td>
<td>11</td>
</tr>
<tr>
<td>IOT</td>
<td>11</td>
</tr>
<tr>
<td>RTT</td>
<td>9</td>
</tr>
<tr>
<td>RTS</td>
<td>4</td>
</tr>
<tr>
<td>CC</td>
<td>6</td>
</tr>
<tr>
<td>MARK</td>
<td>5</td>
</tr>
<tr>
<td>TRAP</td>
<td>5</td>
</tr>
<tr>
<td>EMT</td>
<td>5</td>
</tr>
<tr>
<td>BR(uncond)</td>
<td>3</td>
</tr>
<tr>
<td>BR(cond)</td>
<td>4</td>
</tr>
</tbody>
</table>

Instr Time = Fetch Time + Exec Time

TABLE 4. TIMING FOR JMP AND JSR INSTRUCTIONS

<table>
<thead>
<tr>
<th>MODE</th>
<th>EXEC TIME (CYCLES)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JMP</td>
<td>JSR</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

Instr Time = Fetch Time + Exec Time
### TABLE 5. TIMING FOR SINGLE OPERAND INSTRUCTIONS

<table>
<thead>
<tr>
<th>INSTR</th>
<th>EXECUTE TIME (CYCLES)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WORD OPERAND</td>
</tr>
<tr>
<td></td>
<td>MODE=0</td>
</tr>
<tr>
<td>CLR</td>
<td>1</td>
</tr>
<tr>
<td>COM</td>
<td>1</td>
</tr>
<tr>
<td>INC</td>
<td>1</td>
</tr>
<tr>
<td>DEC</td>
<td>1</td>
</tr>
<tr>
<td>NEG</td>
<td>1</td>
</tr>
<tr>
<td>ADC</td>
<td>1</td>
</tr>
<tr>
<td>SBC</td>
<td>2</td>
</tr>
<tr>
<td>TST</td>
<td>1</td>
</tr>
<tr>
<td>ROR</td>
<td>6</td>
</tr>
<tr>
<td>ROL</td>
<td>6</td>
</tr>
<tr>
<td>ASR</td>
<td>7</td>
</tr>
<tr>
<td>ASL</td>
<td>6</td>
</tr>
<tr>
<td>SXT</td>
<td>2</td>
</tr>
<tr>
<td>MTPS</td>
<td>1</td>
</tr>
<tr>
<td>MFPS</td>
<td>2</td>
</tr>
<tr>
<td>SWAB</td>
<td>10</td>
</tr>
</tbody>
</table>

Instr Time = Fetch Time + DST Time + Exec Time  
(Note that DST Time is only added if DST mode ≠ 0)

### TABLE 6. TIMING FOR DESTINATION AND SOURCE OPERANDS

<table>
<thead>
<tr>
<th>MODE</th>
<th>DST TIME (CYCLES)</th>
<th>SRC TIME (CYCLES)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WORD</td>
<td>BYTE</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
### TABLE 7. TIMING FOR EXTENDED INSTRUCTION SET

<table>
<thead>
<tr>
<th>SOURCE MODE</th>
<th>ADD FOR EACH SHIFT POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODE=0</td>
<td>MODE#0</td>
</tr>
<tr>
<td>MUL</td>
<td>32</td>
</tr>
<tr>
<td>DIV</td>
<td>40</td>
</tr>
<tr>
<td>ASH(left)</td>
<td>9</td>
</tr>
<tr>
<td>ASH(right)</td>
<td>7</td>
</tr>
<tr>
<td>ASHC(left)</td>
<td>17</td>
</tr>
<tr>
<td>ASHC(right)</td>
<td>17</td>
</tr>
<tr>
<td>XOR</td>
<td>1</td>
</tr>
<tr>
<td>SOB</td>
<td>4</td>
</tr>
</tbody>
</table>

Instr Time = Fetch Time + SRC Time + Exec Time
(Nota that SRC Time is only added if DST mode ≠ 0)

### TABLE 8. TIMING FOR DOUBLE OPERAND INSTRUCTIONS

<table>
<thead>
<tr>
<th>INSTR</th>
<th>EXECUTE TIME (CYCLES)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SRC MODE = 0</td>
</tr>
<tr>
<td></td>
<td>DST MODE=0</td>
</tr>
<tr>
<td>MOV</td>
<td>1</td>
</tr>
<tr>
<td>CMP</td>
<td>1</td>
</tr>
<tr>
<td>BIT</td>
<td>1</td>
</tr>
<tr>
<td>BIC</td>
<td>1</td>
</tr>
<tr>
<td>BIS</td>
<td>1</td>
</tr>
<tr>
<td>ADD</td>
<td>1</td>
</tr>
<tr>
<td>SUB</td>
<td>1</td>
</tr>
<tr>
<td>MOVB</td>
<td>4</td>
</tr>
<tr>
<td>CMPB</td>
<td>4</td>
</tr>
<tr>
<td>BITB</td>
<td>4</td>
</tr>
<tr>
<td>BICB</td>
<td>4</td>
</tr>
<tr>
<td>BISB</td>
<td>4</td>
</tr>
</tbody>
</table>

Instr Time = Fetch Time + SRC Time + DST Time + Exec Time
(Nota that SRC Time is only added if SRC mode ≠ 0
and DST Time is only added if DST mode ≠ 0)
## REFERENCES

<table>
<thead>
<tr>
<th>No.</th>
<th>Author</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ziukelis, J.</td>
<td>&quot;ARO Processor - Schematic Diagrams&quot;. 18 January 1980 (unpublished work)</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>&quot;Signetics Micro-assembler Reference Manuals&quot;. 8X02 AS 1000SS</td>
</tr>
<tr>
<td>3</td>
<td>Ziukelis, J.</td>
<td>&quot;ARO Processor - Design Notes&quot;. 22 November 1979 (unpublished work)</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>&quot;The Am2900 Family Data Book&quot;. Advanced Micro Devices, Inc., 1978</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>&quot;DEC/X11 General Products Module Library 1&quot;. Digital Equipment Corp. December 1976</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>&quot;Build a Microcomputer&quot;. Advanced Micro Devices, Inc., 1978</td>
</tr>
<tr>
<td>9</td>
<td>Ziukelis, J.</td>
<td>&quot;An Overview of the ARO Processor&quot;. 14 September 1979 (unpublished work)</td>
</tr>
</tbody>
</table>
THE SOURCE FOR THE MICRO INSTRUCTION FORMAT

INSTRUCTION WIDTH 65 "MICRO STORE WIDTH"

FIELD SRC WIDTH 6 DEFAULT 0 "THE A REG IS THE SRC"
FORMAT;
FIELD AMODE WIDTH 2 DEFAULT 0 "A-MODE SELECT"
FIELD AFIELD WIDTH 4 DEFAULT 0 "A-SOURCE SELECT"
END FORMAT;
FIELD DST WIDTH 5 DEFAULT 27Q "THE B REG IS THE DST"
FORMAT;
FIELD BMODE WIDTH 1 DEFAULT 1 "B-MODE SELECT"
FIELD BFIELD WIDTH 4 DEFAULT 7 "B-SOURCE SELECT"
END FORMAT;
FIELD MARS WIDTH 1 DEFAULT 0 "MEM ADDR REG SELECT"
FIELD C2903 WIDTH 9 DEFAULT 8CH "2903 CONTROL"
FORMAT
FIELD 11_4 WIDTH 4 DEFAULT 4 "ALU OPN AND SPECIAL FNS"
FIELD 15_1 WIDTH 1 DEFAULT 0 "ALU OPN"
FIELD 15_8 WIDTH 4 DEFAULT 0CH "ALU DEST CNTRL INC SHIFT, SPEC"
END FORMAT;
FIELD C2904 WIDTH 15 DEFAULT 6440Q "2904 CONTROL"
FORMAT
FIELD 11_12 WIDTH 2 DEFAULT 0 "ALU CARRY IN SEL, DEF=INHIBIT"
FIELD 16_9 WIDTH 4 DEFAULT 6 "SHIFT LINKAGE SEL"
FIELD 10_5 WIDTH 6 DEFAULT 44Q "STATUS REG UPDATE AND TEST"
FIELD EC WIDTH 1 DEFAULT 0 "CARRY UPDATE, DEF=INHIBIT"
FIELD CEU WIDTH 1 DEFAULT 0 "USR UPDATE, DEF=INHIBIT"
FIELD CEM WIDTH 1 DEFAULT 0 "MSR UPDATE, DEF=INHIBIT"
END FORMAT;
FIELD COMUX WIDTH 3 DEFAULT 0 "TEST CC SELECT, DEF=UNCOND"
FIELD C2910 WIDTH 4 DEFAULT 0EH "2910 CONTROL, DEF=CONTINUE"
FIELD SOF WIDTH 5 DEFAULT 0FH "SPECIAL OPERATION FIELD"
FIELD IMMOP WIDTH 16 DEFAULT 0 "IMMEDIATE FIELD"
FIELD SLWDTH WIDTH 4 DEFAULT 0 "MIR FIELD SELECTOR WIDTH"
FIELD TADDR WIDTH 12 DEFAULT 0 "TARGET ADDR FOR DIR JUMP"
END FORMAT;
FIELD ERROR WIDTH 1 DEFAULT 0 "SET IF ERROR"
END INSTRUCTION;
APPENDIX II

THE SOURCE FOR THE MICROP DEFINITION SECTION

"THE MICROP DEFINITION SECTION"

II.1 The address interpreter

"ADDRESS INTERPRETER - INVOKED BY 2903 ALU AND SPECIAL OPERATIONS"

MICROP @(RSN(OFFSET,WDTH),RDN,TMAR
ASSGN "SET UP SRC,DST FOR 2903"
   "DEFINITION OF LAST FIELD(TMAR)"
   IF (TMAR EQ 'MAR')
       THEN "WRITE TO MAR"
       MARS=1
   ELSE
       IF (TMAR EQ 'QIN')
           THEN "READ FROM Q"
           IO=1
       ELSE
           IF (TMAR EQ 'MARQ')
               THEN "WRITE TO MAR, READ FROM Q"
               MARS=1 IO=1
           ELSE
               FI
           FI
   FI

"DEFINITION OF SECOND FIELD(DESTINATION)"

IF (RDN EQ 'ZERO') OR (RDN EQ 'ONE') OR (RDN EQ 'TWO')
   THEN "FLAG CATASTROPHIC ERROR"
   ERROR=3 "ATTEMPT TO OVERWRITE CONSTANTS"
   ELSE
   FI

IF (RDN GE 0) AND (RDN LE 15) OR (RDN EQ 15H) OR (RDN EQ 27Q)
   THEN "DST IS GP CPU REG"
   DST=RDN
ELSE
   IF (RDN GE 20H) AND (RDN LE 23H)
       THEN "REDEFINE DEDICATED CPU REG"
       DST=RDN-10H
   ELSE
       IF RDN EQ 'RD'
           THEN "DST IS DEFINED BY DD FIELD"
           DST=30Q
       ELSE
           IF RDN EQ 'RS'
               THEN "DST IS DEFINED BY SS FIELD"
               DST=34Q
           ELSE
               IF RDN EQ 'QDST'
                   THEN "DST IS Q REG"
                   I5_8=6 "STORE F IN Q"
               ELSE
                   "INHIBIT RAM WRITE CYCLE"
                   I5_8=0CH
               FI
           FI
       FI
   FI
"DEFINITION OF FIRST FIELD(SOURCE)"

\[ \text{IF} \ W\text{DTH GT 0 "TEST FOR FIELD SEL OPTION"} \]
\[ \text{THEN} \]
\[ \text{AMODE=3 AFIELD=OFFSET SLWĐT=16-WDTH} \]
\[ \text{ELSE} \]
\[ \text{IF RSN EQ 'RD' "SRC SEL FROM DD FIELD"} \]
\[ \text{THEN} \]
\[ \text{SRC=18H} \]
\[ \text{ELSE} \]
\[ \text{IF RSN EQ 'PSW' "SRC IS PSW (ALU INHIBITED)"} \]
\[ \text{THEN} \]
\[ \text{SOF=6 \_5=40Q} \]
\[ \text{ELSE} \]
\[ \text{IF RSN EQ 'RS' "SRC IS DEFINED BY SS FIELD"} \]
\[ \text{THEN} \]
\[ \text{SRC=10H} \]
\[ \text{ELSE} \]
\[ \text{SRC=RSN "SRC IS GP OR DEDICATED CPU REG"} \]
\[ \text{FI F} \]
\[ \text{"SEE ASSEMBLER DIRECTIVES FOR DEFINITION"} \]
\[ \text{FI} \]

FI

"MNEMONICS SUGGESTED BY J. ZIUKELIS"

"ALU MICROS - CONTROLS 2903 - DEFAULTS TO STORE WORD MODE"

MICRÖP SET _F RSN=0 OFFSET=0,WDTH=0),RDN=99,MAD=O
ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD

I1.4=0 DEFAULT 15_8=4 "SET ALU OUTPUT";

MICRÖP D SUB S RSN=0 OFFSET=0,WDTH=0),RDN=99,MAD=O
ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD

I1.4=1 I11.12=1 DEFAULT 15_8=4 "DST-SRC";

MICRÖP S SUB D RSN=0 OFFSET=0,WDTH=0),RDN=99,MAD=O
ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD

I1.4=2 I11.12=1 DEFAULT 15_8=4 "SRC-DST";

MICRÖP S ADD D RSN=0 OFFSET=0,WDTH=0),RDN=99,MAD=O
ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD

I1.4=3 DEFAULT 15_8=4 "SRC+DST";

MICRÖP MOV D RSN=0 OFFSET=0,WDTH=0),RDN=99,MAD=O
ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD

I1.4=4 DEFAULT 15_8=4 "DST TO F";

MICRÖP NOT D RSN=0 OFFSET=0,WDTH=0),RDN=99,MAD=O
ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD

I1.4=5 DEFAULT 15_8=4 "NOT DST";

MICRÖP MOV S RSN=0 OFFSET=0,WDTH=0),RDN=99,MAD=O
ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD

I1.4=6 DEFAULT 15_8=4 "SRC TO F";

MICRÖP NOT S RSN=0 OFFSET=0,WDTH=0),RDN=99,MAD=O
ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD

I1.4=7 DEFAULT 15_8=4 "NOT SRC";

MICRÖP CLR RSN=0 OFFSET=0,WDTH=0),RDN=99,MAD=O
ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD

I1.4=8 DEFAULT 15_8=4 "CLR ALU OUTPUT";

MICRÖP NS AND D RSN=0 OFFSET=0,WDTH=0),RDN=99,MAD=O
ASSIGN @ RSN(OFFSET,WDTH),RDN,MAD

I1.4=9 DEFAULT 15_8=4 "NOT SRC AND DST";
MICROP S\_NXOR\_D RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_4=0AH DEFAULT I\_5=8=4 "SRC NXOR DST";
MICROP S\_XOR\_D RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_4=0BH DEFAULT I\_5=8=4 "SRC XOR DST";
MICROP S\_AND\_D RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_4=0CH DEFAULT I\_5=8=4 "SRC AND DST";
MICROP S\_NOR\_D RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_4=0DH DEFAULT I\_5=8=4 "SRC NOR DST";
MICROP S\_NAND\_D RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_4=0EH DEFAULT I\_5=8=4 "SRC NAND DST";
MICROP S\_OR\_D RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_4=0FH DEFAULT I\_5=8=4 "SRC OR DST";

"SPC\_FN MICROS - CONTROLS 2903"
MICROP MUL RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_0=0 I\_4=0 I\_5=8=0 "MUL OP";
MICROP SMUL RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_0=0 I\_4=0 I\_5=8=2 "SIGN MUL";
MICROP INC1 RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_0=0 I\_4=0 I\_5=8=4 "INC BY 1";
MICROP INC2 RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_0=0 I\_4=0 I\_5=8=4 I\_11=12=1 "INC BY 2";
MICROP CONV RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_0=0 I\_4=0 I\_5=8=5 "SIGN MAG TO TWO COMP";
MICROP MULF RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_0=0 I\_4=0 I\_5=8=6 "MUL, LAST CYCLE";
MICROP NMZS RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_0=0 I\_4=0 I\_5=8=8 "NORM SINGLE PREC";
MICROP NMZD RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_0=0 I\_4=0 I\_5=8=OAH "NORM DOUBLE PREC";
MICROP SDIV RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_0=0 I\_4=0 I\_5=8=OCH "SIGN DIVIDE";
MICROP FDIV RSN=0(OFFSET=0,WIDTH=0), RDN=99, MAD=0
ASSIGN @ RSN(OFFSET,WIDTH), RDN, MAD
I\_0=0 I\_4=0 I\_5=8=OEH "SIGN DIV, LAST CYC";

II.3 The Am2904 microps

"SHFT MICROS - CONTROLS 2903, 2904"
MICROP TST ASSIGN I\_5=8=OCH I\_6=9=6 "TST OP, INHIBITS WRITE";
MICROP STO ASSIGN I\_5=8=4 I\_6=9=6 "STO OP, STORES WORD";
MICROP STOB ASSIGN I\_5=8=0FH I\_6=9=6 "STOB OP, STORES BYTE";
MICROP BSX ASSIGN 15_9=6 15_8=0EH
   "BSX OP, BYTE SIGN EXTEND";
MICROP ROR ASSIGN 15_8=1 16_9=9
   "ROR OP, AS PER PDP11";
MICROP ROL ASSIGN 15_8=9 16_9=9
   "ROL OP, AS PER PDP11";
MICROP ASL ASSIGN 15_8=9 16_9=0
   "ASL, NOTE SIGN BIT NOT RELOADED";

"STATUS TEST OPTIONS"

MICROP ULE ASSIGN CCMUX=1 10_5=20Q "USR LE";
MICROP UGT ASSIGN CCMUX=1 10_5=21Q "USR GT";
MICROP ULT ASSIGN CCMUX=1 10_5=22Q "USR LT";
MICROP UGE ASSIGN CCMUX=1 10_5=23Q "USR GE";
MICROP UEQ ASSIGN CCMUX=1 10_5=24Q "USR EQ";
MICROP UNE ASSIGN CCMUX=1 10_5=25Q "USR NE";
MICROP UVS ASSIGN CCMUX=1 10_5=26Q "USR V SET";
MICROP UVC ASSIGN CCMUX=1 10_5=27Q "USR V CLEAR";
MICROP UCS ASSIGN CCMUX=1 10_5=28Q "USR C SET";
MICROP UCC ASSIGN CCMUX=1 10_5=29Q "USR C CLEAR";
MICROP ULO ASSIGN CCMUX=1 10_5=30Q "USR LOWER OR SAME";
MICROP UHI ASSIGN CCMUX=1 10_5=31Q "USR HIGHER";
MICROP UM1 ASSIGN CCMUX=1 10_5=32Q "USR MINUS";
MICROP UPL ASSIGN CCMUX=1 10_5=33Q "USR PLUS";
MICROP MLE ASSIGN CCMUX=1 10_5=34Q "MSR LE";
MICROP MGT ASSIGN CCMUX=1 10_5=35Q "MSR GT";
MICROP MLT ASSIGN CCMUX=1 10_5=36Q "MSR LT";
MICROP MGE ASSIGN CCMUX=1 10_5=37Q "MSR GE";
MICROP MEQ ASSIGN CCMUX=1 10_5=38Q "MSR EQ";
MICROP MNE ASSIGN CCMUX=1 10_5=39Q "MSR NE";
MICROP MVS ASSIGN CCMUX=1 10_5=40Q "MSR V SET";
MICROP MVC ASSIGN CCMUX=1 10_5=41Q "MSR V CLEAR";
MICROP MCS ASSIGN CCMUX=1 10_5=42Q "MSR C SET";
MICROP MCC ASSIGN CCMUX=1 10_5=43Q "MSR C CLEAR";
MICROP MLO ASSIGN CCMUX=1 10_5=44Q "MSR LOWER OR SAME";
MICROP MHI ASSIGN CCMUX=1 10_5=45Q "MSR HIGHER";
MICROP MI ASSIGN CCMUX=1 10_5=46Q "MSR MINUS";
MICROP MPL ASSIGN CCMUX=1 10_5=47Q "MSR PLUS";

"STATUS UPDATE MICRONS - CONTROL 2904"

MICROP MSR ASSIGN 10_5=44Q CEM=1 EC=1 "LOAD MSR";
MICROP MSRCI ASSIGN 10_5=30Q CEM=1 EC=1 "LOAD MSR WITH CARRY INV";
MICROP MRSNOC ASSIGN 10_5=44Q CEM=1 EC=1 "LOAD MSR WITH NO CARRY";
MICROP MSRZ ASSIGN 10_5=3 CEM=1 EC=1 "CLEAR MSR";
MICROP USR ASSGIN 10_5=44Q CEM=1 "LOAD USR";
MICROP TOST_PSW ASSIGN 10_5=0 CEM=1 EC=1 SOF=11Q 15_8=4
   "TRANSFER Y TO MSR";

"CCMUX CONTROL MICRONS"

MICROP UNCOND ASSIGN CCMUX=0 "UNCOND JUMP-DEFAULT";
MICROP INTREQ ASSIGN CCMUX=2 "INTERUPT REQUEST RECEIVED";

II.4 Microsequencer Microps

"TRNSFR MICRONS - CONTROLS 2910"
MICROP JZ ASSIGN C2910=0 "JUMP ZERO";
MICROP CJS ASSIGN C2910=1 "COND JUMP SUBR";
MICROP JMP ASSIGN C2910=2 "JUMP TO MAP";
MICROP CJP ASSIGN C2910=3 "COND JUMP UIR";
MICROP PUSH ASSIGN C2910=4 "PUSH, COND LOAD CNTR";
MICROP JSRP ASSIGN C2910=5 "COND JUMP SUBR, CNTR OR UIR";
MICROP CJV ASSIGN C2910=6 "COND JUMP VECTOR";
MICROP JRP ASSIGN C2910=7 "COND JUMP CNTR OR UIR";
MICROP RFCT ASSIGN C2910=8 "REPEAT LOOP";
MICROP RPCT ASSIGN C2910=9 "REPEAT UIR";
MICROP CJP ASSIGN C2910=0AH "COND JUMP SUBR, CNTR OR UIR";
MICROP CJPP ASSIGN C2910=0BH "COND JUMP UIR";
MICROP LDCT ASSIGN C2910=0CH "LOAD COUNTER";
MICROP LOOP ASSIGN C2910=0DH "TEST END OF LOOP";
MICROP CONT ASSIGN C2910=0EH "USE NEXT UIR";
MICROP TWB ASSIGN C2910=0FH "THREE WAY BRANCH";

II.5 Subsidiary microps

"SUBSIDIARY MICROPS"

"JMP DEFINITION - JUMPS UNCONDITIONALLY TO ANY 'LABEL'
IF 'INDEXED' IS LABEL THEN IMMEDIATE FIELD USED AS JUMP ADDR"
MICROP JMP LABEL ASSIGN CJP UNCOND
IF (LABEL NE 'INDEXED')
THEN
  TADDR=LABEL
ELSE
  "ADDRESS ALREADY CALC"
FI;

"JSR DEFINITION - PUSHES STACK AND UNCONDITIONALLY JUMPS TO 'LABEL'
IF 'INDEXED' IS LABEL THEN IMMEDIATE FIELD USED AS JUMP ADDR"
MICROP JSR LABEL ASSIGN CJS UNCOND
IF (LABEL NE 'INDEXED')
THEN
  TADDR=LABEL
ELSE
  "ADDRESS ALREADY CALC"
FI;

MICROP RETURN ASSIGN CRTN UNCOND "UNCOND RETURN";
MICROP NEXT ASSIGN JMP FETCH "JUMP TO MACRO FETCH ROUTINE";
MICROP TRAP ASSIGN ERROR=1 JMP ILGL "ILLEGAL INSTRUCTION";

II.6 Memory and immediate field microps

"MEMORY CONTROL MICROPS"

MICROP READ ASSIGN SOF=10H "SET MEM CNTRL BITS FOR READ";
MICROP WRITE ASSIGN SOF=14H "SET MEM CNTRL BITS FOR WRITE";
MICROP READB ASSIGN SOF=11H "SET MEM CNTRL FOR BYTE READ";
MICROP WRITEB ASSIGN SOF=15H "SET MEM CNTRL FOR BYTE WRITE";
MICROP READC ASSIGN SOF=13H "SET MEM CNTRL FOR CACHE READ";
MICROP WRITEC ASSIGN SOF=17H "SET MEM CNTRL FOR CACHE WRITE";

"IMMEDIATE FIELD CONTROL MICROPS"
MICROP PRESET=VALUE ASSGN IMMOP=VALUE "PRESET IMMEDIATE FIELD";
APPENDIX III
THE SOURCE FOR THE PROGRAM DEFINITION SECTION

III.1 The symbol definition section

PROGRAM TEST WIDTH 65 LENGTH 400H;

"THE PROGRAM DEFINITION SECTION"
"SYMBOL DEFINITIONS FOR THE ASSEMBLER"

"REG DEFN FOR RSN (VIZ- SRC REG)"
RO:   EQU 0;
R1:   EQU 1;
R2:   EQU 2;
R3:   EQU 3;
R4:   EQU 4;
R5:   EQU 5;
R6:   EQU 6;
SP:   EQU 6;
R7:   EQU 7;
PC:   EQU 7;
R8:   EQU 8;
R9:   EQU 9;
R10:  EQU 10;
ZERO: EQU 11 "LOAD WITH 0 AT BOOT-UP";
ONE:  EQU 12 "LOAD WITH 1 AT BOOT-UP";
TWO:  EQU 13 "LOAD WITH 2 AT BOOT-UP";
TEMP@: EQU 14;
TEMP: EQU 15;
RS:   EQU 'RS';
PSW:  EQU 'PSW';
RD:   EQU 'RD';
DRO:  EQU 20H;
DR1:  EQU 21H;
DR2:  EQU 22H;
IMM:  EQU 23H;
DUMP: EQU 27Q;
FLDSEL: EQU 'FLDSEL';
MAR:  EQU 'MAR';
QDST: EQU 'QDST';
QIN:  EQU 'QIN';
MARQ: EQU 'MARQ';
INDEXED: EQU 'INDEXED';
NC:   EQU 44Q;
MIR:  EQU 14H;
OPREG: EQU 15H;
DUMMY_ADDR: EQU 177176Q;

III.2 The subsidiary routines

"SUBSIDIARY ROUTINES - NOT DIRECTLY CALLED BY MAPPING PROMS"

"INTERRUPT VECTOR LOCATIONS"

(10): MOVST TEMP,DUMP STO PSW JMP ODDWORD "ODD WORD ADDRESS";
(11): MOVST TEMP,DUMP STO PSW JMP HLTRQ "HALT REQUEST VECTOR";
(12): MOVST TEMP,DUMP STO PSW JMP DCERR "DATA CHANNEL ERROR";
(13): MOVS TEMP,DUMP STO,PSW JMP DCRDY "DATA CHANNEL READY";
(14): MOVS TEMP,DUMP STO,PSW JMP APINTCOND "ARITH PROC TRAP";
(15): MOVS TEMP,DUMP STO,PSW JMP APINTRDY "ARITH PROC READY";

ORG 10H "START SUBSIDIARY ROUTINES AT LOC 16";

"SAVE PSW, PC ON PDP-11 STACK"
SAVE:
  MOVS PSW,DRO "DRO=PSW, MORE";
  D_SUB S TWO,R6,MAR WRITE "R6=R6-2, PUSH TO STACK, MORE";
  D_SUB S TWO,R6,MAR "R6=R6-2, MORE";
  MOVS R7,DRO RETURN "DRO=PC, PUSH TO STACK, END";

"ILLEGAL INSTRUCTION TRAP"
ILGL:
  MOVS IMM,TEMP PRESET=10Q "VECTOR FOR ILLEGAL INSTR";
  JMP GTVCTR "GET NEW PC,PS";
  CONT "DUMMY TO PRESERVE UAR";
  CONT "DUMMY TO PRESERVE UAR";

"MACRO FETCH ROUTINE, (PRESERVE UNA=31 FOR FETCH)"
FCHSPC: MOVS R7, ,MAR READ JMP FETCH "FETCH NO INTRPT";
FETCH: MOVS R7, ,MAR READ "MAR=PC"
  CJP INTREQ TADDR=INTROUT "JUMP TO INTERRUPT HANDLER IF INTERRUPT PRESENT";
@FETCH: DST=MIR
  INC2 ,R7 JMP "INC PC, JUMP TO START ADDR";

"SUBSIDIARY ROUTINES(CONTINUED)"

"GET NEW PC,PS POINTED TO BY TEMP"
GTVCTR: "OLD PC,PS SAVED ON STACK"
  "PC=(TEMP),PS=(TEMP+1)"
  "PRESERVE UNA=34 FOR GTVCTR"
JSR SAVE "SAVE OLD PC,PS ON STACK";
  MOVS TEMP, ,MAR READ "RETRIEVE NEW PC";
  INC2 ,TEMP "POINT TO NEW PS";
  MOVS DRO,R7 "PC=(TEMP)";
  MOVS TEMP, ,MAR READ "RETRIEVE NEW PS";
  CONT "WAIT";
  MOVS DRO,DUMP STO,PSW NEXT "PS=(TEMP+2), END";

"INTERRUPT HANDLER ROUTINE"
INTROUT: DST=MIR "DUMMY PRESERVES DR";
  MOVS PSW,TEMP "SAVE PSW, MORE";
  SOR=5 CJP UNCOND "ENABLE VECTOR, JUMP TO DEFINE INTR TYPE, JUMP BACK TO INTR1";

"MACRO INTERRUPT HANDLER"
"HLTRQ IN INVOKED FROM HOST BY SETTING M-BIT IN INTERFACE"
HLTRQ: MOVS IMM,TEMP PRESET=240Q "VECTOR VIA 240Q";
  JMP GTVCTR "JUMP TO VECTOR";
ODDWORD: MOV IMM,TEMP PRESET=4Q "VECTOR VIA 4Q";
  JMP GTVCCTR "JUMP TO VECTOR";
DECERR: MOV IMM,TEMP PRESET=110Q "VECTOR VIA 110Q";
  JMP GTVCCTR "JUMP TO VECTOR";
DCRDY: MOV IMM,TEMP PRESET=114Q "VECTOR VIA 114Q";
  JMP GTVCCTR "JUMP TO VECTOR";
APINTCOND: MOV IMM,TEMP PRESET=120Q "VECTOR VIA 120Q";
  JMP GTVCCTR "JUMP TO VECTOR";
APINTRDY: MOV IMM,TEMP PRESET=124Q "VECTOR VIA 124Q";
  JMP GTVCCTR "JUMP TO VECTOR";
"REGDST - ROUTINE TO PLACE (MIR6_8 OR 1) IN MIR0_2
VIZ SS OR 1 IN DD FIELD
THIS IS USED AS REG DST IN MUL, DIV, ASHC INSTR
TECHNIQUE: FIELD SELECTOR GENERATES SSS000SS1, LOAD INTO MIR
VIA DUMMY WRITE CYCLE"

REGDST: MOV IMM,DRO PRESET=700Q "DRO=RS MASK, MORE";
  S_AND D FLDSEL(0,9),DRO "DRO=SSS000000, MORE";
  S OR D FLDSEL(6,3),DRO "DRO=SSS0000SS, MORE";
  S OR D ONE,DRO "DRO=SSS0000SS1, MORE";
  MOV IMM, ,MAR PRESET=DUMMY ADDR
  WRITE "MAR=DUMMY ADDR,END";
  DST=MIR RETURN "MIR=DR0,END";

"FIXES OVERFLOW BIT AFTER ROTATES/SHIFTS"

FIXCC: IO_5=16Q CEU=1 "RESET UV";
  UM' CJP TADDR=NSET "JMP IF UN SET";
NCLR: MCS CJP TADDR=SETV "JMP IF MC SET(AND UN CLR)"
XIT: IO_5=2 CEU=1 CEU=1 NEXT "SWAP REG(MC RETAIN), END"
NSET: MCS CJP TADDR=XIT "EXIT IF MC SET(AND UN SET)"
SETV: IO_5=17Q CEU=1 JMP XIT "SET UV AND EXIT"

III.3 Address evaluation

"ADDRESS EVALUATION ROUTINE - WORD INSTRUCTIONS"
"DD FIELD"

DBASE: TRAP "MODE=0 IS ILLEGAL"
  MOV IMM, MAR READ RETURN "MODE=1, FETCH OPND, END"
  MOV IMM, MAR READ JMP DD2 "MODE=2, FETCH OPND, MORE"
  MOV IMM, MAR READ JMP DD3 "MODE=3, FETCH ADDR, MORE"
  D_SUB S TWO,RD,MAR READ RETURN "MODE=4, DEC RD, FETCH OPND"
  D_SUB S TWO,RD,MAR READ JMP DD5 "MODE=5, DEC RD, FETCH ADDR"
  MOV IMM, MAR READ JMP DD6 "MODE=6, FETCH INDEX, MORE"
  MOV IMM, MAR READ JMP DD7 "MODE=7, FETCH INDEX, MORE"
DD2: INC2 ,RD RETURN "AUTO INC, END"
DD3: INC2 ,RD "AUTO INC, MORE"
  MOV IMM, MAR READ RETURN "FETCH OPND, END"
DD5: CONT "WAIT FOR MEM, MORE"
DD6: INC2 ,RD "INC PC(VIZ ORIG PC+4), MORE"
  S_ADD D DRO,RD,MAR TST READ RETURN "MAR=PC+INDEX+4, END"
DD7: INC2 ,RD "INC PC(VIZ ORIG PC+4), MORE"
  S_ADD D DRO,RD,MAR TST READ "MAR=PC+INDEX+4, MORE"
  CONT "WAIT FOR MEM"
"ADDRESS EVALUATION ROUTINE - BYTE INSTRUCTIONS"
"ONLY CHANGE IS TO MODE=2 AND 4"

DBASEB: TRAP
"MODE=0 IS ILLEGAL"

MOVS Rd,,MAR READB RETURN "MODE=1, FETCH OPND, END"

MOVS Rd,,MAR READB JMP DB2 "MODE=2, FETCH OPND, MORE"

MOVS Rd,,MAR READ JMP DB3 "MODE=3, FETCH ADDR, MORE"

MOVS FLDSEL(0,3),TEMP: JMP DB4 "MODE=4, TEMP>=REG NO, MORE"

D SUB S TWO, Rd,,MAR READ JMP DB5 "MODE=5, DEC Rd, FETCH ADDR"

MOVS R7,,MAR READ JMP DB6 "MODE=6, FETCH INDEX, MORE"

MOVS R7,,MAR READ JMP DB7 "MODE=7, FETCH INDEX, MORE"

DB2:
MOVS FLDSEL(0,3),TEMP: "TEMP:=DST REG NO, MORE"

D SUB S IMM, TEMP: USR PRESET=6 "TEMP: GE 0 FOR Rd6,R7"

INC1 ,RD CRNT ULT "RET IF LT 0, MORE"

INC1 ,RD RETURN "INC1 IF Rd6,R7 REL, END"

DB3: INC2 ,RD "AUTO INC, MORE"

MOVS DRO,,MAR READB RETURN "FETCH OPND, END"

DB4: "DEC BY 2 IF Rd6, R7 REL"

D SUB S IMM, TEMP: USR PRESET=6 "TEMP: GE 0 FOR Rd6,R7"

D SUB S ONE, Rd,,MAR READB CRNT ULT "RETURN IF NOT Rd6,R7"

DB5: CONT "WAIT FOR MEM, MORE"

MOVS DRO,,MAR READB RETURN "FETCH OPND, END"

DB6: INC2 ,R7 "INC PC(VIZ ORIG PC+4), MORE"

S_ADD_D DRO, RD,,MAR TST READB RETURN "MAR=PC+INDEX+4, END"

DB7: INC2 ,R7 "INC PC(VIZ ORIG PC+4), MORE"

S_ADD_D DRO, RD,,MAR TST READ "MAR=PC+INDEX+4, MORE"

CONT "WAIT FOR MEM"

MOVS DRO,,MAR READB RETURN "RD=(PC+INDEX+4), FETCH OPND, END"

"SS FIELD ADDRESS EVALUATION"

SBASE: TRAP
"MODE=0 IS ILLEGAL"

MOVS Rs,,MAR READ RETURN "MODE=1, FETCH OPND, END"

MOVS Rs,,MAR READ JMP SS2 "MODE=2, FETCH OPND, MORE"

MOVIS Rs,,MAR READ JMP SS3 "MODE=3, FETCH ADDR, MORE"

D SUB S TWO, Rs,,MAR READ RETURN "MODE=4, DEC Rs, FETCH OPND"

D SUB S ONE, Rs,,MAR READ JMP SS5 "MODE=5, DEC Rs, FETCH ADDR"

MOVIS R7,,MAR READ JMP SS6 "MODE=6, FETCH INDEX, MORE"

MOVIS R7,,MAR READ JMP SS7 "MODE=7, FETCH INDEX, MORE"

SS2: INC2 ,RS RETURN "AUTO INC, END"

SS3: INC2 ,RS "AUTO INC, MORE"

MOVIS DRO,,MAR READ RETURN "FETCH OPND, END"

SS5: CONT "WAIT FOR MEM, MORE"

MOVIS DRO,,MAR READ RETURN "FETCH OPND, END"

SS6: INC2 ,R7 "INC PC(VIZ ORIG PC+4), MORE"

S_ADD_D DRO, RS,,MAR TST READ RETURN "MAR=PC+INDEX+4, END"

SS7: INC2 ,R7 "INC PC(VIZ ORIG PC+4), MORE"

S_ADD_D DRO, RS,,MAR TST READ "MAR=PC+INDEX+4, MORE"

CONT "WAIT FOR MEM"

MOVIS DRO,,MAR READ RETURN "RD=(PC+INDEX+4), FETCH OPND, END"

"ADDRESS EVALUATION ROUTINE - BYTE INSTRUCTIONS"
"ONLY CHANGE IS TO MODE=2 AND 4"
SBASEB: TRAP "MODE=0 IS ILLEGAL";
MOVS RS,,MAR READB RETURN "MODE=1, FETCH OPND, END";
MOVS RS,,MAR READB JMP SB2 "MODE=2, FETCH OPND, MORE";
MOVS RS,,MAR READ JMP SB3 "MODE=3, FETCH ADDR, MORE";
MOVS FLDSEL(6,3),TEMP JMP SB4 "MODE=4, TEMP=REG NO, MORE";
D_SUB_S TWO,RS,,MAR READ JMP SB5 "MODE=5,DEC RS, FETCH ADDR";
MOVS R7,,MAR READ JMP SB6 "MODE=6, FETCH INDEX, MORE";
MOVS R7,,MAR READ JMP SB7 "MODE=7, FETCH INDEX, MORE"

SB2.
MOVS FLDSEL(6,3),TEMP "TEMP=SRC REG NO,MORE";
D_SUB_S IMM,TEMP USR PRESET=6 "TEMP=TEST VAL,MORE";
INC1 ,RS CRTN ULT "RETURN IF NOT R6,R7,MORE";
INC1 ,RS RETURN "RETURN IF R6,R7 REL, END";
SB3.
INC2 ,RS "AUTO INC, MORE";
MOVS DRO,,MAR READB RETURN "FETCH OPND, END";
SB4.
D_SUB_S IMM,TEMP USR PRESET=6 "TEMP=0 IF R6,R7";
D_SUB_S ONE,RS,,MAR READB CRTN ULT "RETURN IF NOT R6,R7";
D_SUB_S ONE,RS RETURN "RETURN IF R6,R7, END";
SB5.
CONT "WAIT FOR MEM, MORE";
MOVS DRO,,MAR READB RETURN "FETCH OPND, END";
SB6.
S_ADD_D DRO,RS,MAR TST READB RETURN "MAR=PC+INDEX+4, END";
SB7.
S_ADD_D DRO,RS,MAR TST READ "MAR=PC+INDEX+4, MORE";
CONT "WAIT FOR MEM";
MOVS DRO,,MAR READB RETURN "RS=(PC+INDEX+4), FETCH OPND, END";

III.4 Class M1 (miscellaneous)
"PDP 11 INSTRUCTION DESCRIPTION"
"ALL LABELS WHICH ARE LEFT JUSTIFIED APPEAR IN MAPPING PROM"
"MISCELLANEOUS INSTRUCTIONS - CLASS M1"
"HALT, WAIT, RTI, BPT, IOT, RESET, RTT"

M1: "CLASS START ADDR FROM MAPPING PROM"
S_ADD_D FLDSEL(0,3),IMM
TADDR=$+I JIP INDEXED "JUMP TO M1 INSTR";
HALT: SOF=30Q NEXT JMP Indexed "HALT FOR HOST TO CLEAR";
WAIT: JMP @WAIT "LOOP WAITING FOR INTERRUPT";
RTI: "RETURN FROM INTERRUPT OR TRAP INSTR"
MOVS R6,,MAR READ JMP @RTI "POP PC FROM STACK, MORE";
BPT: JMP @BPT "BREAK POINT INSTR";
IOT: JMP @IOT "IOT INSTR";
RESET: TRAP "RESET MEANINGLESS";
RTT: JMP RTI "RTT SAME AS RTI";
@WAIT: CJF INTREQ TADDR=INTROUT "TEST FOR INTERRUPT, MORE";
JMP @WAIT "LOOP TO TEST AGAIN";
@RTI: CONT "WAIT FOR NEW PC, MORE";
MOVS DRO,R7 "RESTORE PC, MORE";
INC2 ,R6,MAR READ "POP PSW FROM STACK, MORE";
CONT "WAIT FOR NEW PSW, MORE";
MOVS DRO,DUMP STO PSW "RESTORE PSW, MORE";
INC2 ,R6 JMP FCHSPC "FIX STACK POINTER, END";
@BPT: MOVS IMM,TEMP PRESET=14Q JMP GTVCTR "VECTOR VIA 14Q";
@IOT: MOVS IMM,TEMP PRESET=16Q JMP GTVCTR "VECTOR VIA 16Q";
III.5 Class JUMP, RTS, CC

"JUMP INSTRUCTION - CLASS JUMP"

JUMP:

S_ADD_D FLDSEL(3,3), IMM
TADDR=$+1 JMP INDEXED
TRAP
MOVCS RD,R7 NEXT
MOVCS RD,R7 JMP @JUMP2
MOVCS RD,,MAR READ JMP @JUMP3 "MODE=3, RD=ADDR, MORE"
MOVCS R7,,MAR READ JMP @JUMP6 "MODE=6, FETCH INDEX, MORE"
MOVCS R7,,MAR READ JMP @JUMP7 "MODE=7, FETCH INDEX, MORE"

@JUMP2: INC2 ,RD NEXT "AUTO INC, END"
@JUMP3: INC2 ,RD "AUTO INC, MORE"
MOVCS R6,,MAR READ "SP TO MAR, MORE"
INC2 ,R6 "INC SP, MORE"
MOVCS DRO,R7 NEXT "LINK REG= TOP OF STACK, END"

"RETURN FROM SUBROUTINE INSTRUCTION - CLASS RTS"

"RTS"

RTS:
MOVCS RD,PC "RTS OP, PC=RD, MORE"
MOVCS R6,,MAR READ "SP TO MAR, MORE"
INC2 ,R6 "INC SP, MORE"
MOVCS DRO,R7 NEXT "LINK REG= TOP OF STACK, END"

"CONDITION CODE INSTRUCTIONS - CLASS CC"

"CLN, CLK, CLV, CLC, CCC, SEN, SEZ, SEV, SEC, SCC"

CC:

"OPCodings 00 02 10 TO 00 02 37 ARE ILLEGAL" "CLASS START ADDRESS FROM MAP"
MOVCS IMM,TEMP PRESET=40Q "TEMP=BAD CODE LIMIT, MORE"
MOVCS IMM,TEMP USR "USR NEG IF CODE ILGL, MORE"
MOVCS IMM,TEMP "JUMP IF CODE ILGL"
MOVCS IMM,TEMP "EXTRACT PSW"
MOVCS IMM,TEMP "JUMP FOR SET CC, MORE"
MOVCS IMM,TEMP "CLEAR CC, MSR=RESULT, MORE"
MOVCS IMM,TEMP "SET CC, MSR=RESULT, END"

III.6 Class SWAB, BR and JSR

"SWAB INSTRUCTION - CLASS SWAB"

SWAB1: "MODE EQ 0"
ERL-0286-TM

**MOVS RD, TEMP JSR ROT8**
"TEMP=RD, JUMP TO ROTATE BY 8";
"RESTORE TO RD, END";

**MOVS TEMP, RD NEXT**

**ROT8:**

**MOVD , TEMP 16 =9= OAH 15 =9= OAH MRZ**
"ROTATE LEFT"

**PUSH UNCOND**
"LOAD COUNTER=6, MORE"

**MOVD , TEMP 16 =9= OAH 15 =9= OAH RFCT**
"ROTATE, LOOP 6 TIMES"

**MOVDS TEMP, TEMP 15 =9= OAH**
"TEMP= DUMMY, SIGN EXT"

**MOVD , TEMP MSR RETURN**
"SIGN EXTEND TO FIX CC, END"

**"MODE=0"**

**SWAB2:**

**S_ADD_D FLDSSEL(3,3), IMM**
"FETCH DST OPND, TADDR=DBASE+MODE"

**TADDR=DBASE JSR INDEXED**
"WAIT FOR MEM"

**CONT**

**MOVD, TEMP JSR ROT8**
"TEMP=RD, JUMP TO ROTATE BY 8"

**MOVS TEMP, DRO WRITE**
"RESTORE TO DRO, END";

**"BRANCH INSTRUCTIONS"**

**"BR, BNE, BEQ, BGE, BLT, BGT, BLE"**

**"BRANCH INSTRUCTIONS - CLASS BR"**

**BR:**

**MOVS FLDSSEL(0,8), TEMP BSX**
"TEMP=OFFSET(SIGN EXT), MORE"

**MOVD , TEMP 15 =9= 16 =9= 2**
"OFFSET2, MC UPDATE INHIBIT"

**S_ADD_D TEMP, R7 NEXT**
"PC=PC+2*OFFSET, END"

**BNE:**

**CJP MNE TADDR=BR**
"IF Z=0 THEN BRANCH"

**NEXT**
"ELSE END";

**BEQ:**

**CJP MEQ TADDR=BR**
"IF Z=1 THEN BRANCH"

**NEXT**
"ELSE END";

**BGE:**

**CJP MGE TADDR=BR**
"IF GE THEN BRANCH"

**NEXT**
"ELSE END";

**BLT:**

**CJP MLT TADDR=BR**
"IF LT THEN BRANCH"

**NEXT**
"ELSE END";

**BGT:**

**CJP MGT TADDR=BR**
"IF GT THEN BRANCH"

**NEXT**
"ELSE END";

**BLE:**

**CJP MLE TADDR=BR**
"IF LE THEN BRANCH"

**NEXT**
"ELSE END";

**JSR INSTRUCTION - CLASS JSR@**

**JSR@:**

**"CLASS START ADDR FROM MAP"**

**S_ADD_D FLDSSEL(3,3), IMM**
"DU MODE GIVES OFFSET, MORE"

**TADDR=8+1 JMP INDEXED**
"MODE=0 IS ILLEGAL"

**JSRO:**

**TRAP**

**JSR1:**

**MOVS RD, TEMP JMP STAKO**
"MODE=1, SAVE RD, MORE"

**JSR2:**

**MOVS RD, TEMP JMP @JSR2**
"MODE=2, SAVE RD, MORE"

**JSR3:**

**MOVS RD, TEMP JMP @JSR3**
"MODE=3, READ (RD), MORE"

**JSR4:**

**D_SUB_S TWO, RD JMP @JSR4**
"MODE=4, DEC, MORE"

**JSR5:**

**D_SUB_S TWO, RD, MAR READ JMP @JSR5**
"MODE=5, DEC, FETCH ADDR"

**JSR6:**

**MOVSD R7, MAR READ JMP @JSR6**
"MODE=6, FETCH INDEX, MORE"

**JSR7:**

**MOVSD R7, MAR READ JMP @JSR7**
"MODE=7, FETCH INDEX, MORE"

**@JSR2:**

**INC2 , RD JMP STAKO**
"AUTO INC, MORE"

**@JSR3:**

**INC2 , RD**
"AUTO INC, MORE"

**MOVS DRO, TEMP JMP STAKO**
"TEMP=DRO, MORE"

**@JSR4:**

**MOVS RD, TEMP JMP STAKO**
"TEMP=DRO, MORE"

**@JSR5:**

**CONT**
"WAIT FOR MEM"

**MOVS DRO, TEMP JMP STAKO**
"TEMP=OPND, MORE"

**@JSR6:**

**INC2 , R7**
"INC R7, MORE"

**MOVS DRO, TEMP**
"TEMP=INDEX, MORE"

**@JSR7:**

**INC2 , R7**
"INC R7, MORE"

**MOVS DRO, TEMP**
"TEMP=INDEX, MORE";
S_ADD D R7, TEMP READ "TEMP=ORIG PC+INDEX+4, MORE";
CONT "WAIT FOR MEM";
MOVS DRO, TEMP JMP STAKO "TEMP=(ORIG PC+INDEX+4), MORE";
STAKO: MOVs RS, DRO "DRO=LINKAGE REG, MORE";
D_SUB S IMM, R6, MAR WRITE PRESET=2 "PUSH STACK";
MOVS R7, RS "RS=PC";
MOVS TEMP, R7 NEXT "PC=SUBR ADDR";

III.7 Class S01, S02, S03, S04, S05

"SINGLE OPERAND INSTRUCTIONS"
"CLR, COM, INC, DEC, NEG, ADC, SBC, TST, ROR, ROL, ASR, SXT, MARK, MFPI, MTPI"

"SINGLE OPERAND INSTRUCTIONS - CLASS S01"

CLR1: CLR , RD MSR NEXT "DST MODE EQ 0"
COM1: NOT D, RD MSRCI NEXT "COM OP, END";
INC1: INC , RD MSRNOC NEXT "INC OP, END";
DEC1: D_SUB S ONE, RD MSRNOC NEXT "DEC OP, END";
NEG1: NOT D, RD 111_12=1 MSRCI NEXT "NEG OP, END";
ADC1: MOV D, RD 111_12=3 MSR NEXT "ADC OP, END";
SBC1: MOVS ZERO, TEMP 111_12=3 IO_5=MC "TEMP=CN, MORE";
D_SUB S TEMP, RD MSRCI NEXT "DST-TEMP, END";
TST1: MOVS RD MSR NEXT "TST OP, END";

"SINGLE OPERAND INSTRUCTIONS - CLASS S02"

S02: "CLASS START ADDR FROM MAP, DST MODE NE 0"
S_ADD D FLDSEL(3,3), IMM
TADDR=DBASE JSK INDEXED "FETCH DST OPND, TADDR=DBASE+MODE";
S_ADD D FLDSEL(6,3), IMM
"TADDR=$+1 JMP INDEXED" "JMP TO S02 INSTR";
CLR2: CLR , DRO MSR WRITE NEXT "CLR OP, END";
COM2: NOT D, DRO MSRCI WRITE NEXT "COM OP, END";
INC2: INC , DRO MSRNOC WRITE NEXT "INC OP, END";
DEC2: D_SUB S ONE, DRO MSRNOC WRITE NEXT "DEC OP, END";
NEG2: NOT D, DRO 111_12=1 MSRC1
WRITE NEXT "NEG OP, END";
ADC2: MOV D, DRO 111_12=3
MSR WRITE NEXT "ADC OP, END";
SBC2: MOVS ZERO, TEMP 111_12=3 IO_5=MC
JMP CSBC2 "TEMP=CN, MORE";
TST2: MOVS DRO MSR NEXT "TST OP, END";
CSBC2: D_SUB S TEMP, DRO MSRCI WRITE NEXT "DST-TEMP, END";

"SINGLE OPERAND INSTRUCTIONS - CLASS S03"

S03: "DST MODE EQ 0"
ROR3: MOV D, RD ROR JMP FXCC3 "ROR OP, END";
ROL3: MOV D, RD ROL JMP FXCC3 "ROL OP, END";
ASR3: MOVS RD 16 9=9 "ASR OP, SHIFT N TO MC, MORE";
MOVD, RD ROR JMP FXCC3 "RIGHT SHIFT, MORE";
ASL3: MOV D, RD ASL JMP FXCC3 "ASL OP, END";
SXT3: CP MIFI TADDR=SSXT3 "SXT OP, MORE";
CLR , RD MSR NEXT "CLR IF NOT MIFI, END";
"SINGLE OPERAND INSTRUCTIONS - CLASS S04"

S04: "CLASS START ADDR FROM MAP, DST MODE NE 0"

S_ADD D FLDSEL(3,3), IMM
  "TADDR=DBASE JSR INDEXED "FETCH DST OPND, TADDR=DBASE+MODE";
S_ADD D FLDSEL(6,3), IMM
  "TADDR=+1 JMP INDEXED "JMP TO S04 INSTR";
ROR4: MOV D, DRO ROR WRITE JMP FXCC4 "ROR OP, END";
ROL4: MOV D, DRO ROL WRITE JMP FXCC4 "ROL OP, END";
ASR4: MOV D, DRO 16 9=9 JMP @ASR4 "ASR OP, SHIFT N TO MC, MORE";
ASH4: MOV D, DRO ASL WRITE JMP FXCC4 "ASL OP, END";
  TRAP; "WRONG ADDR FOR S05 GROUP (SEE BELOW)"
  TRAP;
SXT4: CJP MMI TADDR=SXT4 "SXT OP, MORE";
  CLR D, DRO MSR WRITE NEXT "CLR IF NOT MMI, END";
@SXT4: NOT_S_ONE, DRO 11_12=1 MSRNOC
  NEXT WRITE "-1 IF MMI, WRITE, END";
@ASR4: MOV D, DRO ROR WRITE "RIGHT SHIFT, END";
FXCC4: MOV D, DRO USR JMP FXCC "FIX COND CODES, EXCEPT OVR, MORE";

"SINGLE OPERAND INSTRUCTIONS - CLASS S05"

MARK5: MOV D, FLDSEL(0,5), R6 ASL "MARK OP, SP=2‰NN, MORE";
S_ADD D R7, R6, MAR READ "SP=PC+2‰NN, MORE";
MOV R5, R7 "PC=R5, MORE";
INC2 R6 "AUTO INC SP, MORE";
MOV D, R5 NEXT "RESTORE R5";

"DOUBLE OPERAND INSTRUCTIONS"

"MOV, CMP, BIT, BIC, BIS, ADD"

"DOUBLE OPERAND INSTRUCTIONS - CLASS D01"

"SRC MODE EQ 0, DST MODE EQ 0"

D01:
MOV1: MOV R, R, RD MSRINOC NEXT "MIRSRC IN RS, MIRDST IN RD"
CMP1: S_SUB D RS, RD TST MSRCL NEXT "CMP OP, END";
BIT1: S_AND D RS, RD TST MSRINOC NEXT "BIT OP, END";
BIC1: NS AND D RS, RD MSRINOC NEXT "BIC OP, END";
BIS1: S OR D RS, RD MSRINOC NEXT "BIS OP, END";
ADD1: S_ADD D RS, RD MSR NEXT "ADD OP, END";

"DOUBLE OPERAND INSTRUCTIONS - CLASS D02"

D02:
"CLASS START ADDR FROM MAP, SRC MODE EQ 0, DST MODE NE 0"
S_ADD D FLDSEL(3,3), IMM
  "TADDR=DBASE JSR INDEXED "FETCH DST OPND, TADDR=DBASE+MODE";
S_ADD D FLDSEL(12,3), IMM
  "TADDR=+1 JMP INDEXED "JMP TO D02 INSTR";
MOV2: MOV R, D, DRO MSRINOC NEXT WRITE "MOV OP, END";
CMP2: S_SUB D RS, DR0 TST MSRCL NEXT "CMP OP, END";
BIT2: S AND D RS, DRO TST MSRNOC NEXT "BIT OP, END"
BIC2: NS AND D RS, DRO MSRNOC NEXT WRITE "BIC OP, END"
BIS2: S OR D RS, DRO MSRNOC NEXT WRITE "BIS OP, END"
ADD2: S ADD D RS, DRO MSR NEXT WRITE "ADD OP, END"

"DOUBLE OPERAND INSTRUCTIONS - CLASS D03"

D03: "CLASS START ADDR FROM MAP, SRC MODE NE 0, DST MODE EQ 0"
S ADD D FLDSEL(9,3), IMM
TADDR=SBASE JSR Indexed "FETCH SRC OPND, TADDR=DBASE+MODE"
S ADD D FLDSEL(12,3), IMM "MSK SRC IN DRO, MIRDST IN RD"
TADDR=S JMP Indexed "JMP TO D03 INST"
MOV3: MOV DRO, RD MSRNOC NEXT "MOV OP, END"
CMP3: S SUB D DRO, RD TST MSRC1 NEXT "CMP OP, END"
BIT3: S AND D DRO, RD TST MSRNOC NEXT "BIT OP, END"
BIC3: NS AND D DRO, RD MSRNOC NEXT "BIC OP, END"
BIS3: S OR D DRO, RD MSRNOC NEXT "BIS OP, END"
ADD3: S ADD D DRO, RD MSR NEXT "ADD OP, END"

"DOUBLE OPERAND INSTRUCTIONS - CLASS D04"

D04: "CLASS START ADDR FROM MAP, SRC MODE NE 0, DST MODE NE 0"
S ADD D FLDSEL(9,3), IMM
TADDR=SBASE JSR Indexed "FETCH SRC OPND, TADDR=DBASE+MODE"
CONT "WAIT FOR OPND"
MOV3 DRO, TEMP "STORE SRC OPND IN TEMP"
S ADD D FLDSEL(3,3), IMM "FETCH DST OPND, TADDR=DBASE+MODE"
TADDR=DBASE JSR Indexed "MIR SRC IN TEMP, MIRDST IN RD"
S ADD D FLDSEL(12,3), IMM
TADDR=S JMP Indexed "JMP TO D04 INST"
MOV4: MOV TEMP, DRO MSRNOC NEXT WRITE "MOV OP, END"
CMP4: S SUB D TEMP, DRO TST MSRC1 NEXT "CMP OP, END"
BIT4: S AND D TEMP, DRO TST MSRNOC NEXT "BIT OP, END"
BIC4: NS AND D TEMP, DRO MSRNOC NEXT WRITE "BIC OP, END"
BIS4: S OR D TEMP, DRO MSRNOC NEXT WRITE "BIS OP, END"
ADD4: S ADD D TEMP, DRO MSR NEXT WRITE "ADD OP, END"

III.8 Class EIS

"EXTENDED INSTRUCTION SET INSTRUCTIONS - CLASS EIS"
"MUL, DIV, ASH, ASHC, XOR, FADD, FSUB, FMUL, FDIV, SOB"

"MUL INSTRUCTION - DST(REG) AND SRC MULTIPLIED AND STORED IN REG, REG+1"

MUL1: "SRC MODE EQ 0"
"TEMP = SRC OPND"
@MUL: "TEMP=MULTICIPAND, RS=MULTIPiER"
MOV RS, QDST LDCT PRESET=15 "Q=MULTIPLIER, COUNTER=15"
CLR .TEMP "CLEAR TEMP, MORE"
SMUL TEMP . TEMP In 9=06H
RPT CT TADDR=S "2 COMP MUL, LINK $100 TO $003, DECR LOOP"
MULF TEMP . TEMP In 9=06H
111 12=2 "2 COMP MUL LAST CYCLE, LINK $100 TO $003, LINK CNC TO CX, MORE"
"AT THIS STAGE PRODUCT(MS) IN TEMP"
PRODUCT(LS) IN Q
MULTICIPAND IN TEMP"
MULTIPLIER IN RS

MOVS TEMP, RS
JSR REGDST
"RS=PRODUCT(MS), REGDST PLACES (SS OR 1) IN DD FIELD"
USR
"STORE CC FOR MS, N CORRECT, UV=0, UC=0, MORE";
MOVD, RD, QIN, MSR
"STORE LS IN RD,
STORE CC FOR LS IN MSR, MORE";
"REST NEEDED TO FIX CONDITION CODES"
CJP MPL, TADDR=QULA
"BR IF MS N=0, MORE";
10.5=17 Q CEU=1
"SET UV, MORE";
@MULA:
CJP UGE, TADDR=QULB
"BK IF (N(MS) XOR N(LS))=0";
10.5=16 Q CEU=1
"SET UC, MORE";
@MULB:
CJP MEQ, TADDR=QULC
"JUMP IF LS Z=1, MORE";
10.5=10 Q CEU=1
"RESET UZ, MORE";
@MULC:
10.5=2 CEN=1 CEU=1 NEXT
"Z CORRECT, SWAP STATUS REG, END OF DIVIDE";
"SRC MODE NE 0"
MUL2:
S ADD D, FLDESEL(3,3), IMM
TADDR=DBASE, JSR INDEXED
"FETCH SRC OPND, MORE";
CONT
"WAIT FOR OPND";
MOVS DRO, TEMP, JMP @MUL
"TEMP=SRC OPND, MORE";
DIV INSTRUCTION - DIVIDEND(MS) IN REG, DIVIDEND(LS) IN REG OR 1,
SRC IS DIVISOR - PERFORMS TWOS COMPLEMENT DIVIDE"

DIV1:
MOVS RD, DRO, MSR
"SRC MODE EQ 0"
@DIV1:
MOVS DRO, QDST, JSR REGDST
"DRO=SRC OPND, SAVE DIVID SIGN";
MOVD, DRO, QIN
"SAVE DRO"
@DIV2:
MOVS DRO, TEMP, 10.5=3 CEU=1
"RD=DIVIDEND(LS), MORE"
MOVS RS, TEMP
"RESTORE DRO"
CJP MEQ, TADDR=DIVZ
"DIV =DIVISOR, RESET USR"
CONV, TEMP, 1111.12=2 USR
"DIV=DIVIDEND(MS)"
"IF DIVISOR=0 THEN EXIT"
TEMP=S/M DIVIDEND(MS),
LINK CIN TO CX, UPDATE USR";
"TEMP:=2*TEMP:-(MAG DVMDS)"
CJP UVS, TADDR=QUOT
"ABORT IF OVERFLOW DURING CONV";
CONV, TEMP, 1111.12=2 USR
"TEMP=S/M DIVISOR,
LINK CIN TO CX, UPDATE USR";
"ABORT IF OVERFLOW DURING CONV";
CJP UVS, TADDR=QUOT
"ABORT IF OVERFLOW DURING CONV"
S ADD D, IMM, TEMP, PRESET=77777Q
"TEMP=MAG DVM";
S SUB D, TEMP, TEMP, TST USR
"TEST DVMDS-DVM";
CJP UHI, TADDR=QQUOT
"ABORT IF QUOTIENT TOO LARGE"
MOVS RD, QDST
"Q=RD(DIVIDEND(LS))";
MOVD, RS, 15.8=0AH 16 9=4
"SHIFT DVMDS,DVMDS LEFT BEFORE
STARTING DIVIDE";
NYZD, DRO, RS 16 9=0FH
"DOUBLE LENGTH NORM, S100 TO Q103"
PUSH UNCOND, PRESET=0DH
"PUSH LOC ON STACK, LOAD COUNTER"
DIV DRO, RS 1111.12=2 16 9=0FH
"TWO COMP DIV, LINK CIN TO CX,
LINK S100 TO Q103, S103 TO Q100"
RFCT
"LOOP FOR 14 CYCLES"
FDIV DRO, RS 1111.12=2 16 9=3
"DIVIDE CORR, LINK CIN TO CX,
Q101=1"
"QUOTIENT AND REMAINDER CORRECTION IS PERFORMED
SIMILAR TO RHYNE(1971)"
MOVS, TEMP, QIN, USR
"TEMP=Q, UPDATE USR"
MOVS DRO, MSR
"DRO=DVM, UPDATE MSR"
MOVS RS, RD CJP UMI "RD=k"
TADDR=QNEG "BR IF Q NEG"
@QPOS: MOVS RD CJP 10 5=17 Q CCMUX=1 "BR IF R AND DVS SAME SIGN"
TADDR=EN DIV "BR IF R AND DVS DIFF SIGN"
@QNEG: MOVS RD CJP 10 5=16 Q CCMUX=1 TADDR=EN DIV
@CORR1: D_SUB S ONE TEMP TADDR=JQNEG IF BR "TEMP:=Q-1"
S ADD D BRO, RD JMP @EN DIV "RD=R-DVS"
@NEG: MOVS RD CJP 10 5=17 Q CCMUX=1 TADDR=EN DIV
@CORR2: S ADD D ONE TEMP IF R AND DVS SAME SIGN "ROE"
TADDR=:JQNEG ITBR IF "ROE"
@QPOS: MOVS RD CJP 10 5=17 Q CCMUX=1 "TEMP:=Q+1"
TADDR=EN DIV "TEMP:=Q+1"
@QNEG: MOVS RD CJP 10 5=16 Q CCMUX=1 TADDR=EN DIV
@ENDIV: MOVS TEMP, RD MSR NEXT "RS=QUOTIENT, UPDATE MSR"
"************ END OF DIV OPERATION"
@QUOT: 10 5=3 CEC=1 "RESET USR"
10 5=17 Q CEC=1 "SET UV AS QUOTIENT TOO LARGE"
@DIVZ: 10 5=13 Q CEC=1 "SET UV AS DIVIDE BY ZERO"
10 5=2 Q CEC=1 CEM=1 EC=1 NEXT "SWAP USR, MSR, END"
DIV2: S_ADD D FLDS_EL(3,3), IMM "TADDR=DBASE JSR INDEXED "FETCH SRC OPND, MORE"
JSR INDEXED "WAIT FOR SRC"
S_MOV D BRO MSR JMP @DIV1 "SRC OPND, JUMP"

"ASH INSTRUCTION - REG SHIFTED ARITH NN PLACES LEFT OR RIGHT"
"ASH AND ASHC START AT SAME MAPPING ADDRESS"
ASH1:
MOVS RD, TEMP Q "SRC MODE EQ 0"
MOVS TEMP, MSR "TEMP:=SOURCE"
CJP UMI TADDR=S-ASHR "BR IF UN=1, VIZ RIGHT SHIFT"
JMP @ASHR "LEFT SHIFT OP"
ASHR:
NOT S TEMP, IMM 10 5=3 GEA=1 "IMT=(MAG NN)-1, RESET USR"
LCT "STORE MAG OF NO OF LOOPS IN COUNTER"
S_ADD D FLDS_EL(9, 1), IMM "RD=ASH"
TADDR=S+1 JMP INDEXED "ASR INDEXED"
JMP @ASHRA
ASHCR:
JSR REG DST "ASH RIGHT SHIFT OPERATION"
MOVS RD, Q DST "RD=(RD OR 1)"
"RD=(RD OR 1)"
"RD=(RD OR 1)"
MOV D, KS 16 9=9 "SHIFT SIGN TO MC"
MOV D, KS 15 8=2 16 9=0 CH "DOUBLE WORD ASR"
RPCT TADDR=S-1 "LOOP FOR NN CYCLES"
MOV D, USR JMP @ASHCC "FIX COND CODES"
ASHRA:
MOVS KS 16 9=9 "ASH RIGHT SHIFT OPERATION"
MOV D, KS OR "SHIFT SIGN TO MC"
MOV D, KS OR "SHIFT RIGHT"
RPCT TADDR=S+1 "LOOP FOR NN CYCLES"
MOVS KS MSR NOC "UPDATE MSR WITH MC RETAIN"
NEXT "************ END OF ASH (RIGHT SHIFT)"
ASHL:
S_SUB D TEMP, IMM PRESET=1 "ASH, ASHC LEFT SHIFT OPERATION"
10 5=3 CEC=1 LCT "STORE (NS-1) IN LCT"
S_ADD D FLDS_EL(9, 1), IMM "RD=ASH"
TADDR=S+1 JMP @ASHLA "RESET USR"
ASHCL:
JSR REG_DST 10 5=3 CEC=1 "RD=ASH"
MOV D, KS OR "RESET USR"
MOV D, KS OR "DOUBLE WORD ASL"
MOV D, KS 15 8=0 OR 16 9=4 "IN=SIGN BEFORE SHIFT"
"IN=SIGN AFTER SHIFT"
CCMUX=1 TADDR=S+2
10.5=17Q CEU=1
RPCT TADDR=S-3
MOVS RS 10.5=6 CEU=1

"BR IF (IN XOR MN)=1";
"SET UV";
"LOOP FOR NN CYCLES";
"UPDATE USR(U RETAIN)";
"FIX UV FOR ASHC"

@ASHCC:
"UN CORRECT, UV CORRECT, MC"
MOVD .RD,QIN MSRNOC
CJP MEQ TADDR=ASHCD
10.5=10Q CEU=1

"RESET UV";
"SWAP REG WITH MC RETAIN"

@ASHCD: 10.5=2 CEM=1 CEU=1
"RESET Z1";

"END OF ASHC OPERATION";

ASH2:
S ADD D FLDSSEL(3,3),IMM
TADDR=DBASE JSR INDEXED
CONT
MOVS DRO,TEMP2 JMP @ASH

"XOR INSTRUCTION - DST=REG XOR DST"

XOR1:
S_XOR_D RS,RD MSRNOC NEXT "XOR OP, END";
"DST MODE NE 0"

XOR2:
S ADD D FLDSSEL(3,3),IMM
TADDR=DBASE JSR INDEXED
CONT
S_XOR_D RS,DRO MSRNOC WRITE NEXT "XOR OP, END";

"SUB INSTRUCTION - SUBTRACT ONE AND BRANCH IF NE 0"

SOB:
D_SUB S,NE,RS USR
"RS=RS-1, UPDATE USR";
CJP UEQ TADDR=FETCH "IF RS ZERO THEN FETCH NEXT INSTR";
MOVS FLDSSEL(0,n),TEMP ASL "TEMP=2\(^n\)OFFSET, MORE";
D_SUB S TEMP,R7 NEXT "R7=R7-2\(^n\)OFFSET, END";

III.9 Class BRS,TR

"BRANCH INSTRUCTIONS"
"BPL, BMI, BHI, BLO, BVC, BVS, BCC, BHS, BCS, BLO"
"BRANCH INSTRUCTIONS - CLASS BRS"

BRS:
MOVS FLDSSEL(10,81),TEMP BRSX "TEMP=OFFSET(SIGN EXT),MORE";
MOVD ,TEMP 15 S=9 16 S=2
S ADD D TEMP,R7 NEXT "OFFSET+2, NO MC UPDATE, MORE";

BPL:
CJP MPL TADDR=BRS "IF PLUS THEN BRANCH";
"ELSE END";

BMI:
CJP MNI TADDR=BRS "IF MINUS THEN BRANCH";
"ELSE END";

BHI:
CJP MHI TADDR=BRS "IF HIGHER THEN BRANCH";
BLO: CJP NLOs TADDR=BRS "IF LOWER OR SAME THEN BRANCH";
NEXT
BVC: CJP MVC TADDR=BRS "IF V CLEAR THEN BRANCH";
NEXT
BVS: CJP MVS TADDR=BRS "IF V SET THEN BRANCH";
NEXT
BCC: BHIS: CJP MCC TADDR=BRS "IF C CLEAR THEN BRANCH";
NEXT
BCE: BLO: CJP LCS TADDR=BRS "IF C SET THEN BRANCH";
NEXT

"EMT AND TRAP INSTRUCTIONS"
ENT: MOVS IMM,TEMP PRESET=30Q "VECTOR VIA 30Q";
JMP GTVCTR "JUMP TO VECTOR";
TRAP#: MOVS IMM,TEMP PRESET=34Q "VECTOR VIA 34Q";
JMP GTVCTR "JUMP TO VECTOR";

III.10 Class S06,S07,S08,S09,S10
"SINGLE OPERAND INSTRUCTIONS - BYTE"
"CLRB, COMB, INC8, NEGB, ADCB, TSTB"
"SINGLE OPERAND INSTRUCTIONS (BYTE) - CLASS S06"
S06:
CLR6: CLR RD MSR NEXT "CLR8 OP, END";
COM6: MOVS RD,TEMP BSX "SIGN EXT, MORE";
INC6: MOVS RD,TEMP BSX "SIGN EXT, MORE";
DEC6: MOVS RD,TEMP BSX "SIGN EXT, MORE";
NEG6: MOVS RD,TEMP BSX "SIGN EXT, MORE";
ADC6: MOVS RD,TEMP BSX "SIGN EXT, MORE";
SBC6: MOVS RD,TEMP BSX "SIGN EXT, MORE";
TST6: MOVS RD,TEMP BSX "TST OP, TEMP IS DUMMY";
@S06: MOVD ,TEMP MSR NEXT "UPDATE MSR, END";

"SINGLE OPERAND INSTRUCTIONS (BYTE) - CLASS S07"
S07:
S_ADD D FLDSEL(3,3), IMM TADDR=DBASE+MODE "FETCH DST OPND, TADDR=DBASE+MODE";
CONT "WAIT FOR MEM, MORE";
MOVD ,DRO BSX "SIGN EXT, MORE";
S_ADD D FLDSEL(6,3), IMM TADDR=S+1 JMP INDEXED "JUMP TO S07 INSTR, MORE";
CLR7: CLR ,DRO MSR WRITEB NEXT "CLR8 OP, END";
COM7: NOT D, DRO MSR CI WRITEB NEXT "COMB OP, END";
INC37: S ADD D ONE, DRO MSR NOC WRITEB NEXT "INCB OP, END";
DEC7: D SUB S, DRO MSR NOC WRITEB NEXT "DECB OP, END";
NEG7: NOT D, DRO 111_12=1 MSR CI WRITEB NEXT "NEGB OP, END";
ADC7: MOV D, DRO 111_12=3 MSR WRITEB NEXT "ADCB OP, END";
SBC7: MOV ZERO, TEMP 111_12=3 10 5=MC JMP @SBC7 "TEMP=CIN, MORE";
TST7: MOV D, TEMP MSR NEXT "TSTB OP, TEMP IS DUMMY, END";
@SBC7: D_SUB_S TEMP, DRO MSR CI WRITEB NEXT "DST-TEMP, END";

"SINGLE OPERAND INSTRUCTIONS (BYTE) - CLASS S08"

S08: "MODE EQ 0"
"CLEARS HIGH BYTE, ROTATES RIGHT, CORRECTS BIT7 IF NECESSARY"
ROR8: MOV IMM, TEMP 10 5=2 CEU=1 EC=1
PRESET=OFFH "TEMP=LOW BYTE MASK, USR=MSR";
S AND D TEMP, RD ROR CEU=0
*CJP UCC TADDR=@ROR8 "MASK LOW BYTE, ROTATE RIGHT, INHIBIT USR, JUMP IF UC CLEAR";
S OR_D IMM, RD PRESET=80H "IF UC SET CLEAR BIT7, MORE";
@ROR8: MOV D, RD BSX "SIGN EXT, MORE";
@S08: MOV D, RD USR JMP FIXCC "SAVE NZVC, EXCEPT OVR";
ROL8: MOV D, RD BSX "SIGN EXTEND, MORE";
MOV D, RD ROL JMP @S08 "ROTATE LEFT, MORE";

ASR8: "CLEARS BITS 7-15, ROTATES RIGHT, CORRECTS BITS 6&7 IF NECESSARY"
MOV IMM, TEMP PRESET=7FH "TEMP=MASK FOR BTO-6, MORE";
MOV D, RD BSX "SIGN EXTEND, MORE";
MOV D, RD CEU=1 "SAVE SIGN IN USR";
S AND_D TEMP, RD ROR CEU=0
*CJP UPL TADDR=@ASR8 "MASK BITS 0-6, ROTATE RIGHT, INHIBIT USR, JUMP IF USR POS";
S OR_D IMM, RD PRESET=COH "IF NEG, SET BITS 6&7, MORE";
@ASR8: MOV D, RD BSX JMP @S08 "SIGN EXT, SAVE EXCEPT OVR";
ASL8: MOV D, RD BSX "SIGN EXTEND OPND, MORE";
MOV D, RD ASL "SHIFT LEFT, SAVE, MORE";
MOV IMM, RD USR JMP FIXCC "FIX CC, EXCEPT OVR";

"SINGLE OPERAND INSTRUCTIONS (BYTE) - CLASS S09"

S09: "DST MODE NE 0"
"CLEARS HIGH BYTE, ROTATES RIGHT, CORRECTS BIT7 IF NECESSARY"
ROR9: S ADD D FLDSEL(3,3), IMM
TADDR=DBASEB JSR INDEXED "FETCH DST OPND, MORE";
MOV IMM, TEMP 10 5=2 CEU=1 EC=1 PRESET=OFFH "TEMP=LOW BYTE MASK, USR=MSR";
S AND_D TEMP, DRO ROR CEU=0
*CJP UCC TADDR=@ROR9 "MASK LOW BYTE, ROTATE RIGHT, INHIBIT USR, JUMP IF UC CLEAR";
S OR D IMM, DRO PRESET=80H "IF UC SET CLEAR BIT7, MORE";
@ROR9: MOV D, DRO BSX WRITEB "SIGN EXT, MORE";
@S09: MOV D, DRO USR JMP FIXCC "SAVE NZVC EXCEPT OVR";
ROL9: S ADD_D FLDSEL(3,3), IMM
TADDR=DBASEB JSR INDEXED "FETCH DST OPND, MORE";
CONT "WAIT FOR MEM, MORE";
MOV D, DRO BSX "SIGN EXTEND, MORE";
MOV D, DRO ROL WRITEB "ROTATE LEFT, MORE";
MOVS DRO USR   JMP FIXCC "FIX CC EXCEPT OVR";

ASR9:
"CLEARS BITS 7-15, ROTATES RIGHT, CORRECTS BITS 6&7 IF NECESSARY"
S_ADD_D FLDSEL(3,3),IMM
  TADDR=DBASEB JSR INDEXED "FETCH DST OPND,MORE";
MOVS IMM,TEMP PRESET=7FH "TEMP=MASK FOR BITO-6, MORE";
MOVD ,DRO BSX "SIGN EXTEND, MORE";
MOVD ,DRO CEU=1 "SAVE SIGN IN USR";
S_AND_D TEMP,DRO ROR CEU=0
CJP UPL TADDR=<ASR9 "MASK BITSO-6, ROTATE RIGHT, INHIBIT USR, JUMP IF USR POS";
S_OR_D IMM,DRO PRESET=0COH "IF NEG, SET BITS 6&7, MORE";
@ASR9: MOVD ,DRO BSX WRITEB "SIGN EXT, MORE"
  JMP @S09

ASL9: S_ADD_D FLDSEL(3,3),IMM
  TADDR=DBASEB JSR INDEXED "FETCH DST OPND, MORE";
CONT "WAIT FOR MEM, MORE";
MOVD ,DRO BSX "SIGN EXTEND OPND, MORE";
MOVD ,DRO ASL WRITEB "SHIFT LEFT, SAVE, MORE";
MOVS DRO USR   JMP FIXCC "FIX CC, EXCEPT OVR";

"SINGLE OPERAND INSTRUCTIONS - CLASS S010"

S010:

"MTPS MOVES SRC TO PSW"

MTPS1:
  MOVD ,RD BSX "SIGN EXT, MORE";
  MOVS RD,DUMP STO PSW NEXT "PSW=SRC,END";

MTPS2:
S_ADD_D FLDSEL(3,3),IMM
  TADDR=DBASEB JSR INDEXED "FETCH SRC OPND, MORE";
CONT "WAIT FOR SRC, MORE";
MOVD ,DRO BSX "SIGN EXTEND OPND, MORE";
MOVS DRO,DUMP STO PSW NEXT "PSW=SRC,END";

"MFPS MOVES CONTENTS OF PSW TO DST"

MFPS1:
  MOVS PSW,TEMP "EXTRACT PSW, STORE IN TEMP";
  MOVS TEMP,RD BSX "SIGN EXT DST, STORE, MORE";
@MFPS1: MOVD ,RD MRSnoc NEXT "SAVE NZV, END";

MFPS2:
S_ADD_D FLDSEL(3,3),IMM
  TADDR=DBASEB JSR INDEXED "FETCH DST OPND, MORE";
  MOVS PSW,TEMP "EXTRACT PSW, STORE IN TEMP";
  MOVS TEMP,DRO BSX WRITEB "SIGN EXT DST, STORE, END";
@MFPS2: MOVD ,DRO MRSnoc NEXT "SAVE NZV, END";

III.11 Class D05,D06,D07,D08,D09

"DOUBLE OPERAND INSTRUCTIONS(BYTE)"
"MOVB, CMPB, BITB, BICB, BISB"

"DOUBLE OPERAND INSTRUCTIONS - CLASS D05"
"SRC MODE EQ 0, DST MODE EQ 0"

DO5:

"DO NOT MODIFY HIGH BYTE OF RD EXCEPT FOR MOV"

```
MOV RS, TEMP
MOV RD, TEMP
S_ADD D FLDSEL(12,3), IMM
TADDR= S JMP Indexed
```

```
DOV5: MOV TEMP, RD
       JMP @MFPS1 "MOV OP, MORE"
```

```
CMP5: S_SUB D TEMP, TEMP;
       TST MSRC1 NEXT "CMP OP, END"
```

```
BIT5: S_AND D TEMP, TEMP;
       TST MSRNOC NEXT "BIT OP, END"
```

```
BIC5: NS_AND D RS, RD
       STOB MSRNOC NEXT "BIC OP, END"
```

```
BIS5: S_OR_D RS, RD
       STOB MSRNOC NEXT "BIS OP, END"
```

"DOUBLE OPERAND INSTRUCTIONS - CLASS DO6"

```
CLASS START ADDR FROM MAP, SRC MODE EQ 0, DST MODE NE 0"
```

```
S_ADD D FLDSEL(3,3), IMM
TADDR=DBASEB JSR Indexed "MIRSRC IN RS, MIRDST IN DRO"
MOV RS, TEMP
MOV DRO BSX
S_ADD D FLDSEL(12,3), IMM
TADDR= S JMP Indexed "JMP TO DO5 INSTR"
```

```
MOV6: MOV TEMP, DRO
       MSRNOC NEXT WRITEB "MOV OP, END"
```

```
CMP6: S_SUB D TEMP, DRO
       TST MSRC1 NEXT "CMP OP, END"
```

```
BIT6: S_AND D TEMP, DRO
       TST MSRNOC NEXT "BIT OP, END"
```

```
BIC6: NS_AND D TEMP, DRO
       MSRNOC NEXT WRITEB "BIC OP, END"
```

```
BIS6: S_OR_D TEMP, DRO
       MSRNOC NEXT WRITEB "BIS OP, END"
```

"DOUBLE OPERAND INSTRUCTIONS - CLASS DO7"

"DO NOT MODIFY HIGH BYTE OF RD EXCEPT FOR MOVB"

```
CLASS START ADDR FROM MAP, SRC MODE NE 0, DST MODE EQ 0"
```

```
S_ADD D FLDSEL(9,3), IMM
TADDR=DBASEB JSR Indexed "FETCH SRC OPND, TADDR=DBASE+MODE"
MOV DRO, TEMP
MOVD, DRO BSX
S_ADD D FLDSEL(12,3), IMM
TADDR= S JMP Indexed "JMP TO DO6 INSTR"
```

```
MOV7: MOV DRO, RD
       JMP @MFPS1 "MOV OP, SIGN EXT, MORE"
```

```
CMP7: S_SUB D DRO, TEMP
       TST MSRC1 NEXT "CMP OP, END"
```

```
BIT7: S_AND D DRO, TEMP
       TST MSRNOC NEXT "BIT OP, END"
```

```
BIC7: NS_AND D DRO, RD
       STOB MSRNOC NEXT "BIC OP, END"
```

```
BIS7: S_OR_D DRO, RD
       STOB MSRNOC NEXT "BIS OP, END"
```

"DOUBLE OPERAND INSTRUCTIONS - CLASS DO8"

```
CLASS START ADDR FROM MAP, SRC MODE NE 0, DST MODE NEO"
```

```
S_ADD D FLDSEL(9,3), IMM
TADDR=DBASEB JSR Indexed "FETCH SRC OPND, TADDR=DBASE+MODE"
CONT
"WAIT FOR OPND"
```

```
MOV8: MOV TEMP, DRO
       MSRNOC NEXT WRITEB "MOV OP, MORE"
```

```
CMP8: S_SUB D TEMP, DRO
       TST MSRC1 NEXT "CMP OP, MORE"
```
BIT8: S AND D TEMP, DRO TST MSRNC NEXT "BIT OP, MORE";
BIC8: NS AND D TEMP, DRO MSRNC NEXT WRITEB "BIC OP, END";
BIS8: S OR D TEMP, DRO MSRNC NEXT WRITEB "BIS OP, MORE";

"DOUBLE OPERAND INSTRUCTIONS - CLASS D09"

SUB1: "SRC MODE EQ 0, DST MODE EQ 0"
      "MIRSRC IN RS, MIRDST IN RD"
      D SUB S RS, RD MSRCI NEXT "SUB OP, END"

"DOUBLE OPERAND INSTRUCTIONS - CLASS D010"

SUB2: "CLASS START ADDR FROM MAP, SRC MODE EQ 0, DST MODE NE 0"
      S ADD D FLDS(3, 3), IMM "FETCH DST OPND, TADDR=DBASE+MODE"
      TADDR=DBASE JSR INDEXED "MIRSRC IN RS, MIRDST IN DRO"
      CONT "WAIT FOR DST, MORE"
      D_SUB_S RS, RD MSRCI NEXT WRITE "SUB OP, MORE"

"DOUBLE OPERAND INSTRUCTIONS - CLASS D011"

SUB3: "CLASS START ADDR FROM MAP, SRC MODE NE 0, DST MODE EQ 0"
      S ADD D FLDS(9, 3), IMM "FETCH SRC OPND, TADDR=DBASE+MODE"
      TADDR=DBASE JSR INDEXED "FETCH SRC OPND, TADDR=DBASE+MODE"
      CONT "WAIT FOR SRC, MORE"
      D_SUB_S RD, RD MSRCI NEXT "SUB OP, END"

"DOUBLE OPERAND INSTRUCTIONS - CLASS D012"

SUB4: "CLASS START ADDR FROM MAP, SRC MODE NE 0, DST MODE NE 0"
      S ADD D FLDS(9, 3), IMM "FETCH SRC OPND, TADDR=DBASE+MODE"
      TADDR=DBASE JSR INDEXED "FETCH SRC OPND, TADDR=DBASE+MODE"
      CONT "WAIT FOR SRC, MORE"
      MOV S DRO, TEMP "TEMP=MIRSRC, MORE"
      S ADD D FLDS(3, 3), IMM "FETCH DST OPND, TADDR=DBASE+MODE"
      TADDR=DBASE JSR INDEXED "MIRSRC IN TEMP, MIRDST IN DRO"
      CONT "WAIT FOR MEM, MORE"
      D_SUB_S TEMP, DRO MSRCI NEXT WRITE "SUB OP, END"

III.12 Indirect mapping table

"THE FOLLOWING IS A TEMPORARY JUMP TABLE TO AVOID THE NECESSITY OF REBURNING PROMS FOR EACH CORRECTION"

ORG 1140Q;
JILGL: JMP JILGL;
JM1: JMP M1;
JJUMP: JMP JUMP;
JKTS: JMP KTS;
JCC: JMP CC;
JSWB1: JMP SWAB1;
JSWB2: JMP SWAB2;
JBR: JMP BR;
JBEQ: JMP BNE;
JBEQ: JMP BGE;
JBLT: JMP BLT;
JBGT: JMP BGT;
JBLE: JMP BLE;
JSR#: JMP JSR#;
JCLR1: JMP CLR1;
JCOM1: JMP COM1;
JINC1: JMP INC1;
JDEC1: JMP DEC1;
JNEG1: JMP NEG1;
JADC1: JMP ADC1;
JSBC1: JMP SBC1;
JTST1: JMP TST1;
JS2: JMP SO2;
JROR3: JMP ROR3;
JROL3: JMP ROL3;
JASR3: JMP ASR3;
JASL3: JMP ASL3;
JSXT3: JMP SXT3;
JS04: JMP SO4;
JMARK: JMP MARK5;
JMOV1: JMP MOV1;
JCMPI: JMP CMP1;
JBIT1: JMP BIT1;
JBIC1: JMP BIC1;
JBIS1: JMP BIS1;
JADD1: JMP ADD1;
JD02: JMP DO2;
JD03: JMP DO3;
JD04: JMP DO4;
Jmul1: JMP MUL1;
Jmul2: JMP MUL2;
JDIV1: JMP DIV1;
JDIV2: JMP DIV2;
JASHI: JMP ASH1;
JASH2: JMP ASH2;
JXOR1: JMP XOR1;
JXOR2: JMP XOR2;
JSOB: JMP SOB;
JBPL: JMP BPL;
JBMI: JMP BMI;
JBHI: JMP BHI;
JBL0S: JMP BLOS;
JBC: JMP BVC;
JBVS: JMP BVS;
JBCG: JMP BCG;
JBCS: JMP BCS;
JEMT: JMP EMT;
JTRAP: JMP TRAP#
JCLR6: JMP CLR6;
JCOM6: JMP COM6;
JINC6: JMP INCO6;
JDEC6: JMP DEC6;
JNEG6: JMP NEG6;
JADC6: JMP ADC6;
JSBC6: JMP SBC6;
JTST6: JMP TST6;
JS07: JMP SO7;
JROR8: JMP ROR8;
JROL8: JMP ROL8;
JASR8: JMP ASR8;
JASL8: JMP ASL8;
III.13 Special macro-instructions

SPC:
"START OF SPECIAL OPERATION INSTRUCTIONS"
"DECODES 1700XX TO 1777XX ONLY"
"SPECIALS ARE IN 8 GROUPS - SPC0..SPC7"
"(CORRESPONDS TO 1700XX..1777XXX)"
S_ADD_D FLDS(9,3),IMM
TADDR=$+1 JMP INDEXED "JUMP TO SPC GROUP";
S_ADD_D FLDS(6,3),IMM
TADDR=SPC0 JMP INDEXED "JUMP WITHIN SPC0(1 OF 8)";
S_ADD_D FLDS(8,1),IMM
TADDR=SPC1 JMP INDEXED "JUMP WITHIN SPC1(1 OF 2)";
S_ADD_D FLDS(8,1),IMM
TADDR=SPC2 JMP INDEXED "JUMP WITHIN SPC2(1 OF 2)";
JMP ILGL "SPC3 NOT USED";
JMP ILGL "SPC4 NOT USED";
JMP ILGL "SPC5 NOT USED";
JMP ILGL "SPC6 NOT USED";
JMP ILGL "SPC7 NOT USED";
SPC0: "MFCPU,MTCPU,MFCAC,MTCAC,MFIOP,MTHST,MTREF"
"TEMP USED TO MANIPULATE ADDRESSING MODE"
S_ADD_D FLDS(3,3),IMM
TADDR=MFCPU JMP INDEXED "MFCPU ADDRESSING";
S_ADD_D FLDS(3,3),IMM
TADDR=MTCPU JMP INDEXED "MTCPU ADDRESSING";
S_ADD_D FLDS(3,3),IMM
TADDR=MFCAC JMP INDEXED "MFCAC ADDRESSING";
S_ADD_D FLDS(3,3),IMM
TADDR=MTCAC JMP INDEXED "MTCAC ADDRESSING";
S_ADD_D FLDS(3,3),IMM
TADDR=MFIOP JMP INDEXED "MFIOP ADDRESSING";
S_ADD_D FLDS(3,3),IMM
TADDR=MTHST JMP "MTHST INTERPRETING";
DST=260 NEXT "MTREF INTERPRETING";
MFCPU: "MOVES FROM CPU TO DR2,DR1,DR0 (NOTE ORDER)"
"ADDRESSING RESOLVED FIRST"
MOV R2,DR2 JMP @MFCPU "R2=DR2(MODE=0)";
JMP @MFCP1 CLR TEMP "MODE=1";
JMP @MFCP2 CLR TEMP "MODE=2";
JMP ILGL "MODE=3 IS ILLEGAL";
JMP @MFCP4 CLR ,TEMP "MODE=4";
JMP ILGL "MODE=5 IS ILLEGAL";
JMP ILGL "MODE=6 IS ILLEGAL";
JMP ILGL "MODE=7 IS ILLEGAL";

@MFCP0: MOVS R1,DR1 "DR1=R1";
MOVS RO,DRO JMP FCHSPC "DRO=RO, END";

@MFCP4: "ENTRY POINT FOR MODE=4, SUBTRACT 6 FROM POINTER"
D_SUB S IMM, RD PRESET=6 "ADJUST RD (REG ADDR)";
JMP @MFCP1 "EXECUTE INSTR";

@MFCP2: "ENTRY POINT FOR MODE=2, TEMP=2 FOR THIS MODE"
MOVS TWO,TEMP JMP @MFCP "TEMP FOR POINTER CORRN";

@MFCP1: "ENTRY POINT FOR MODE=1"
MOVS IMM,TEMP PRESET=-4 "ADJ ADDR";

@MFCP: MOVS RD,,MAR READ "GET EXP";
INC2 ,RD, MAR "ADJ MAR";
MOVS DRO,TEMP@ READ "TEMP@=EXP, GET MSN";
INC2 ,RD, MAR "ADJ MAR";
MOVS DRO,R10 READ "R10=MSM, GET LSM";
S_ADD D TEMP, RD "ADJ ADDR, DR0=LSM";
MOVS R10,DR1 "DR1=MSM";
MOVS TEMP,DR2 JMP FCHSPC "DR2=EXP, END";

@MTCP: "ENTRY POINT FOR MODE=0"
MOVS DRO,R0 JMP @MTCP0 "MODE=0";
JMP @MTCP1 CLR ,TEMP "MODE=1";
JMP @MTCP2 CLR ,TEMP "MODE=2";
JMP ILGL "MODE=3 IS ILLEGAL";
JMP @MTCP4 CLR ,TEMP "MODE=4";
JMP ILGL "MODE=5 IS ILLEGAL";
JMP ILGL "MODE=6 IS ILLEGAL";
JMP ILGL "MODE=7 IS ILLEGAL";

@MTCP0: MOVS DR1,R1 "R1=DR1";
MOVS DR2,R2 NEXT "R2=DR2, END";

@MTCP4: D_SUB S IMM, RD PRESET=6 "ADJUST POINTER FOR MODE=4";
JMP @MTCP1;

@MTCP2: MOVS IMM,TEMP PRESET=6 "CORRN FOR ADDR POINTER";

@MTCP1: "EP FOR MODE=1(TEMP PRESET TO 0)"
S_ADD D IMM, RD, MAR PRESET=4 WRITE "ADJ PTR, WRITE LSM"
MOVS DR1,DRO "DRO=SM";
D_SUB S TWO, RD, MAR WRITE "ADJ PTR, WRITE MSN"
MOVS DR2,DRO "DRO=EXP";
D_SUB S TWO, RD, MAR WRITE "ADJ PTR, WRITE EXP"
S_ADD D TEMP, RD NEXT "CORR PTR, END";

MFCAC: "MOVE FROM CACHE TO DR2,DR1,DR0"
"ADDRESSING RESOLVED FIRST"
JMP ILGL "MODE=0 IS ILLEGAL";
MOVS RD,,MAR READC JMP DELAY "MODE=1";
MOVS RD,,MAR READC JMP @MFCA2 "MODE=2";
JMP ILGL "MODE=3 IS ILLEGAL";
JMP @MFCA4 "MODE=4";
JMP ILGL "MODE=5 IS ILLEGAL";
JMP ILGL "MODE=6 IS ILLEGAL";
JMP ILGL "MODE=7 IS ILLEGAL";

@MFCA2: INC1 ,RD JMP FCHSPC "ADJUST POINTER, END";

@MFCA4: D_SUB S ONE, RD, MAR READC "ADJUST POINTER, READ";

DELAY: JMP FCHSPC "WAIT FOR READ, END";

MTCAC: "MOVE TO CACHE FROM DR0,DR1,DR2"
"ADDRESSING RESOLVED FIRST"
JMP ILGL "MODE=0 IS ILLEGAL";
MOVS RD,,MAR WRITEC NEXT "MODE=1";
MOVS RD,,MAR WRITEC JMP @MTCA2 "MODE=2";
JMP ILGL "MODE=3 IS ILLEGAL";
JMP @MTCA4 "MODE=4";
JMP ILGL "MODE=5 IS ILLEGAL";
JMP ILGL "MODE=6 IS ILLEGAL";
JMP ILGL "MODE=7 IS ILLEGAL";

@MTCA2: INC1 ,RD NEXT "ADJUST POINTER, END";
@MTCA4: D SUB S ONE,RD,MAR WRITEC NEXT "ADJ PTNTER,WRITE";

MFIOP: "MOVES FROM 40BIT 10 PAGE TO DRO,DR1,DR2"
"ADDRESSING RESOLVED FIRST"
JMP ILGL "MODE=0";
JMP @MFIO1 "MODE=1";
JMP ILGL "MODE=2";
JMP ILGL "MODE=3";
JMP ILGL "MODE=4";
JMP ILGL "MODE=5";
JMP ILGL "MODE=6";
JMP ILGL "MODE=7";

@MFIO1: MOVs RD,,MAR READ "FETCH DST OPND INTO DRO,1,2";
JMP FCHSPC "WAIT THEN FETCH NO INTRPT";

MTIOP: "MOVES TO 40BIT 10 PAGE TO DRO,DR1,DR2"
"ADDRESSING RESOLVED FIRST"
JMP ILGL "MODE=0";
JMP @MTIO1 "MODE=1";
JMP ILGL "MODE=2";
JMP ILGL "MODE=3";
JMP ILGL "MODE=4";
JMP ILGL "MODE=5";
JMP ILGL "MODE=6";
JMP ILGL "MODE=7";

@MTIO1: MOVs RD,,MAR WRITE NEXT "WRITE DRO,1,2 TO 10 PAGE";

MTHST: "MOVES DST TO OPREG"
S_AND D FLDSEL(3,3), IMM TADDR=7 USR "UZ=1 IF MODE=0";
CJP UNE TADDR=C@MTHST1 "JMP IF MODE NE 0";
MOVS RD,OPREG MSR NEXT "OPREG=DST(MODE=0)";

@MTHST1: S ADD D FLDSEL(3,3), IMM TADDR=BASE JSR INDEXED "FETCH DST OPND";
CONT MOVs DRO,OPREG MSR NEXT "OPREG=DST(MODE NE 0)";

SPC1: "INTERPRET BDGTR,BDLER"
JMP @SPC1 "ITS BDGTR";
CJP CCMUX=6 TADDR=BR "BR IF COND TRUE(BDLER)";
NEXT "OTHERWISE END";

@SPC1: CJP CCMUX=7 TADDR=BR "BR IF COND TRUE(BDGTR)";
NEXT "OTHERWISE END";

SPC2: "INTERPRET BDGTR,BDLER"
JMP @SPC2 "ITS BDGTR";
CJP CCMUX=6 10_5=45Q TADDR=BR "BR IF COND TRUE(BDLER)";
NEXT "OTHERWISE END";

@SPC2: CJP CCMUX=7 10_5=45Q TADDR=BR "BR IF COND TRUE(BDGTR)";
NEXT "OTHERWISE END";

END;
APPENDIX IV

OPERATING PROCEDURES ON THE IBM370

As described in reference 2, the formatted microcode is generated in a two pass process. The first pass generates the microcode in an intermediate object code form. This is accomplished by the foreground job 'ARO.LIB.CLIST(ASMDENV)'. In practice this task takes two minutes of CPU time so the BATCHTSO command is used to submit this portion of the job. This job processes the source code found in 'ARO.CPROC.DATA' and generates a listing file 'ARO.CPROC.MLST' and an intermediate object code file 'ARO.CPROC.MOBJ'.

The second pass reformats 'ARO.CPROC.MOBJ' into the load module 'ARO.CPROC.MFMT'. This is accomplished by the foreground job 'ARO.LIB.CLIST(FMTRUN)'.

The next stage is to extract the addresses of the left justified labels (see Sections 5.3 and 5.4) from the listing file to generate the contents of the mapping PROMs. Additional information on the PDP-11 instruction code is required at this stage so that the left justified addresses can be associated with particular groups of instruction codes. The foreground job 'ARO.LIB.CLIST(CPROC)' takes the listing file 'ARO.CPROC.MLST' and PDP-11 code information in 'ARO.JMAP.DATA' and generates the mapping PROM contents in 'ARO.PROM.DAT'. The latter file contains a list of the PROM addresses and contents and mnemonics of the left justified labels. Due to the design of the Control Processor, the PROM addresses are mapped from the PDP-11 op-code as follows:- the most significant ten bits of the op-code are directly mapped to the most significant ten bits of the PROM address and the least significant bit of the PROM address is generated from NOT(BIT3 OR BIT4 OR BITS) of the op-code (viz. detects mode 0 in the destination field). Thus destination mode 0 is mapped to odd PROM addresses and destination mode 1 through to mode 7 are mapped to even PROM addresses.

The listing file 'ARO.CPROC.MLST' cannot be generated with octal microcode addresses as is the PDP-11 standard. The foreground job 'ARO.LIB.CLIST(CPOCT)' reads the listing file and produces 'ARO.CPOCT.MLST' which is in the desired format.

The final stage is to generate the magnetic tape volume AROI which is in RSX FILES-11 format containing the files (7,75)MICROCODE.DAT and (7,75)PROM.DAT. This is accomplished by submitting the background job 'ARO.LIB.CNTL(RSXDJH)'. When the data is transferred to the Jindalee Auxiliary Computer, PROM.DAT is converted into the form required by the PROM programmer by the program (7,75)AROPRM.FTN.

IV.1 Usage summary

The source for the microcode is stored in 'ARO.CPROC.DATA'. To recompile the microcode enter:

```
BATCH DSN(LIB.CLIST(ASMDENV)) CLASS(A) MSGCLASS(T) JOBCHAR(Q)
```

To display the results of the compilation, reformat the microcode, generate the mapping PROM contents, convert the addresses to octal, and print the listing file, enter:

```
EX LIB(GENUC)
```

To transfer the formatted microcode and PROM contents to RSX FILES-11 tape enter:
SUBMIT LIB(RSXDJH)

It is essential to check that the following data sets are in the catalog:
'ARO.LIB.LOAD', 'ARO.LIB.CLIST', 'ARO.LIB.CNTL',
'ARO.JMPL.DATA', 'ARO.CPROC.DATA', 'ARO.CPROC.CNND'. 
Figure 1. A schematic of the Control Processor

SIGNETICS BIPOLAR MICRO ASSEMBLER(A)

Figure 2. A sample of micro-assembler output
MICRO ASSEMBLER SYNTAX

<table>
<thead>
<tr>
<th>ALU source dest mar 29H4</th>
<th>usr transfer label memory loading comment</th>
<th>terminator</th>
</tr>
</thead>
<tbody>
<tr>
<td>funct reg defn</td>
<td>shift msc control</td>
<td>type or control of</td>
</tr>
<tr>
<td>LABEL: &lt;ALU &gt; RSN, RD, &lt;MAR &gt; &lt;TST &gt; &lt;MSR &gt; &lt;TS_B &gt; &lt;MAZ &gt; &lt;MSP &gt; &lt;MR &gt; &lt;MSPC &gt;</td>
<td>&lt;JMPLABEL &gt; &lt;READ &gt; &lt;PRESET=0NNH &gt; &quot;COMMENT&quot;</td>
<td>;</td>
</tr>
</tbody>
</table>

Figure 3. The micro-assembler syntax

MNEMONIC DEFINITION

<table>
<thead>
<tr>
<th>ALU 11.4</th>
<th>SPC_FN 15.8</th>
<th>RSN SRC</th>
<th>RDN DST</th>
<th>TRNSFR CSMUL</th>
<th>COND 10.5</th>
<th>CCMUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET F 0</td>
<td>MUL 0</td>
<td>RU 0</td>
<td>RU 0</td>
<td>JZ 0</td>
<td>U0E 10H 1</td>
<td></td>
</tr>
<tr>
<td>D_SUB 1</td>
<td>SMUL 2</td>
<td>R1 1</td>
<td>R1 1</td>
<td>CJ 1</td>
<td>UCT 11H 1</td>
<td></td>
</tr>
<tr>
<td>D_ADD 3</td>
<td>INC 1 4</td>
<td>R2 2</td>
<td>R2 2</td>
<td>JMAP 2</td>
<td>U0L 12H 1</td>
<td></td>
</tr>
<tr>
<td>MOV D 4</td>
<td>CONV 5</td>
<td>R3 3</td>
<td>R3 3</td>
<td>CJP 3</td>
<td>U0C 13H 1</td>
<td></td>
</tr>
<tr>
<td>NOT D 5</td>
<td>MULF 6</td>
<td>R4 4</td>
<td>R4 4</td>
<td>PUSH 4</td>
<td>U0E 14H 1</td>
<td></td>
</tr>
<tr>
<td>MOV S 6</td>
<td>NMZS 8</td>
<td>R5 5</td>
<td>R5 5</td>
<td>JSRP 5</td>
<td>U0E 15H 1</td>
<td></td>
</tr>
<tr>
<td>NOT 1 7</td>
<td>NMZD 0AH</td>
<td>R6 6</td>
<td>R6 6</td>
<td>CBJ 6</td>
<td>U0V 16H 1</td>
<td></td>
</tr>
<tr>
<td>CIP 8</td>
<td>SDIV 0CH</td>
<td>R7 7</td>
<td>R7 7</td>
<td>EFC 8</td>
<td>UCS 1AH 1</td>
<td></td>
</tr>
<tr>
<td>NS AND D 9</td>
<td>FDIV 0IH</td>
<td>PC 7</td>
<td>PC 7</td>
<td>KPC 9</td>
<td>UGC 1BH 1</td>
<td></td>
</tr>
<tr>
<td>NS XOR D 0</td>
<td>OAH</td>
<td>R8 8</td>
<td>R8 8</td>
<td>CRN 0AH</td>
<td>UL0S 1CH 1</td>
<td></td>
</tr>
<tr>
<td>S_AND 0</td>
<td>OAH</td>
<td>R9 9</td>
<td>R9 9</td>
<td>CJP 0B</td>
<td>UHI 10H 1</td>
<td></td>
</tr>
<tr>
<td>NS OR 0</td>
<td>OCH</td>
<td>R10 0AH</td>
<td>R10 0AH</td>
<td>FC 0CH</td>
<td>UHI 11H 1</td>
<td></td>
</tr>
<tr>
<td>S_NAND 0</td>
<td>OEH</td>
<td>RJRO 0</td>
<td>OEH</td>
<td>LL0P 0DH</td>
<td>U0P 11H 1</td>
<td></td>
</tr>
<tr>
<td>S_OR_D 0</td>
<td>OTH</td>
<td>ONI 0</td>
<td>OCH</td>
<td>CONT 0EH</td>
<td>M0L 20H 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>TM0 0</td>
<td>TD0 0</td>
<td>TD0 0</td>
<td>M0G 11H 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>TM0 0</td>
<td>TD0 0</td>
<td>TD0 0</td>
<td>ML0 22H 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RS 10H</td>
<td>RD 18H</td>
<td>R0 1CH</td>
<td>M0C 23H 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RD 18H</td>
<td>RS 18H</td>
<td>DR0 10H</td>
<td>M0C 24H 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR 18H</td>
<td>DR 18H</td>
<td>DR1 10H</td>
<td>M0E 25H 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR 18H</td>
<td>DR 18H</td>
<td>DR2 10H</td>
<td>MV0 26H 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR 18H</td>
<td>DR 18H</td>
<td>DR3 10H</td>
<td>MV0 27H 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR 18H</td>
<td>DR 18H</td>
<td>DR4 10H</td>
<td>M0C 30H</td>
<td>ML0 3AH 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR 18H</td>
<td>DR 18H</td>
<td>DR5 10H</td>
<td>ML0 40H</td>
<td>ML0 2CH 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR 18H</td>
<td>DR 18H</td>
<td>DR6 10H</td>
<td>ML0 3CH 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR 18H</td>
<td>DR 18H</td>
<td>DR7 10H</td>
<td>ML0 4CH 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR 18H</td>
<td>DR 18H</td>
<td>DR8 10H</td>
<td>ML0 5CH 1</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4. Microcode mnemonic definition
DISTRIBUTION

EXTERNAL

In United Kingdom

The Institution of Electrical Engineers,
Hitchin Herts, SG5 1RJ

British Library, Lending Division

Defence Science Representative, London

In United States of America

Cambridge Scientific Abstracts, Riverdale MD 20840

Counsellor, Defence Science, Washington

In Australia

Department of Defence

Chief Defence Scientist

Deputy Chief Defence Scientist

Controller, Projects and Analytical Studies

Superintendent, Science and Technology Programmes

Director, Joint Intelligence Organisation (DSTJ)

Document Exchange Centre

Defence Information Services Branch for:

Microfilming

United Kingdom, Defence Research Information Centre (DRIC)

United States, Defense Technical Information Center

Canadian, Director, Scientific Information Services

New Zealand, Ministry of Defence

National Library of Australia

Director General, Army Development (NSO), Russell Offices

for ABCA Standardisation Officers

UK ABCA representative, Canberra

US ABCA representative, Canberra

Canada ABCA representative, Canberra

NZ ABCA representative, Canberra
### DOCUMENT CONTROL DATA SHEET

**Security classification of this page**

UNCLASSIFIED

<table>
<thead>
<tr>
<th>1 DOCUMENT NUMBERS</th>
<th>2 SECURITY CLASSIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Series Number: ERL-0286-TM</td>
<td>b. Title in Isolation: Unclassified</td>
</tr>
<tr>
<td>Other Numbers:</td>
<td>c. Summary in Isolation: Unclassified</td>
</tr>
</tbody>
</table>

**Title:**

THE MICROCODE FOR THE CONTROL PROCESSOR OF THE ARO ARRAY PROCESSOR

<table>
<thead>
<tr>
<th>4 PERSONAL AUTHOR(S):</th>
<th>5 DOCUMENT DATE:</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.J. Heilbronn</td>
<td>August 1983</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6.1 TOTAL NUMBER OF PAGES</th>
<th>9 COST CODE:</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>DEF 77/036</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6.2 NUMBER OF REFERENCES</th>
<th>8 REFERENCE NUMBERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>a. Task: DEF 77/036</td>
</tr>
<tr>
<td></td>
<td>b. Sponsoring Agency: DEPT. OF DEFENCE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7.1 CORPORATE AUTHOR(S):</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Electronics Research Laboratory</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7.2 DOCUMENT SERIES AND NUMBER</th>
<th>11 COMPUTER PROGRAM(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electronics Research Laboratory 0286-TM</td>
<td>(Title(s) and language(s))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10 IMPRINT (Publishing organisation)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Defence Research Centre Salisbury</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>12 RELEASE LIMITATIONS (of the document):</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Approved for Public Release</td>
<td></td>
</tr>
</tbody>
</table>

Security classification of this page

UNCLASSIFIED
A high speed array processor (ARO) has been designed for the processing of radar data in real time. The operation of the ARO is supervised by a bit-slice microprocessor (the Control Processor). This document contains a description of the microcode which was written for the Control Processor to enable it to interpret PDP-11 assembler code.
The official documents produced by the Laboratories of the Defence Research Centre Salisbury are issued in one of five categories: Reports, Technical Reports, Technical Memoranda, Manuals and Specifications. The purpose of the latter two categories is self-evident, with the other three categories being used for the following purposes:

Reports: documents prepared for managerial purposes.

Technical Reports: records of scientific and technical work of a permanent value intended for other scientists and technologists working in the field.

Technical Memoranda: intended primarily for disseminating information within the DSTO. They are usually tentative in nature and reflect the personal views of the author.