X-715-72-11
PREPRINT

NASA TM X- 65843

# SUPPORT SOFTWARE FOR THE ADVANCED ON-BOARD PROCESSOR

FLIGHT DATA STORAGE BRANCH
ELECTRONICS DIVISION

(NASA-TM-X-65843) SUPPORT SOFTWARE FOR THE ADVANCED ON-BOARD PROCESSOR (NASA) Jan. 1972 104 p CSCL 09B
N72-19243
Unclas
G3/08 21094

(PAGES)
TMX-65843
(NASA CR OR TMX OR AD NUMBER)
(CODE)
08
(CATEGORY)

JANUARY 1972



GSFC

GODDARD SPACE FLIGHT CENTER
GREENBELT, MARYLAND

X-715-72-11

# SUPPORT SOFTWARE

# FOR THE

# ADVANCED ON-BOARD PROCESSOR*

Flight Data Storage Branch
Electronics Division

January 1972

*Prepared by CSC under contract #NAS5-11790.

GODDARD SPACE FLIGHT CENTER
Greenbelt, Maryland

PRECEDING PAGE BLANK NOT FILMED

# SUPPORT SOFTWARE FOR THE ADVANCED ON-BOARD PROCESSOR


Flight Data, Storage Branch
Electronics Division


## ABSTRACT


This document describes the support software package which exists for the Advanced On-Board Processor (AOP) being developed by the Flight Data Storage Branch, GSFC.

PRECEDING PAGE BLANK NOT FILMED

## CONTENTS

## ILLUSTRATIONS



## TABLES

# SUPPORT SOFTWARE FOR THE ADVANCED ON-BOARD PROCESSOR

## INTRODUCTION

The support software package described herein is a second generation outgrowth of a previous package for the On-Board Processor described in document X-562-68-388, 'Support Software for the Space Electronics Branch On-Board Processor', November 1968. The guiding force in the evolution of the package has been the need to adapt to different operating requirements imposed by the Advanced On-Board Processor. Many improvements were also incorporated, especially in the simulator area.

The Advanced On-Board Processor, AOP, is a general purpose stored program digital computer with an 18-bit word. It has integral input-output and interrupt systems especially designed for spacecraft application. The action of the interrupt system is described under programming notes. A description of the AOP is contained in document X-715-71-451, 'The Advanced On-Board Processor — AOP', October, 1971. This support software is intended to provide the AOP programmer with a convenient, thoroughly proven, tool for writing and debugging AOP programs. It is to be expected that the package will continue to evolve into a still better system as more experience is gained in its use with the AOP.

SUPPORT SOFTWARE SYSTEM

The support software system consists of an assembler, a relocatable loader, a simulator, and a support software executive which interprets the system control cards. The system was written in Fortran as a step towards machine independence so that AOP programs could be developed at different locations.

At present, the system has been run and checked out on an XDS 920 computer at GSFC. A special interface unit between the XDS 920 computer and the AOP is being designed. When connected, it will allow object programs to be loaded into AOP memory and executed. It also permits the transfer of data between AOP memory and the XDS 920 I/O devices.

The support software system is contained on one magnetic tape. The minimum system requirement is a card reader, a CPU with 16K memory, a line printer, and four magnetic tape drives (one for the system and three scratch). The system contains 21 links due to core memory limitations. For this reason the system is now undergoing modifications which will allow utilization of a random access memory device.

The AOP Assembler accepts punched cards as input, assembles program segments into relocatable binary code and data, and writes the program segments on a magnetic tape. The binary segments can then be selectively loaded onto a complete image of AOP memory and written on tape. The memory image tape can be used to either load the AOP memory or to serve as input to the AOP Simulator, a unit of the support software system.

A call to a subprogram not found in any assembled program segment will result in an automatic library search. The library currently does not contain any programs. However, programs can be added as required.

## Assembler

A program to be assembled must be on punched cards and must be preceded by an assemble control card. The format of the assemble control card with an explanation of the various options is covered in the section on control cards.

To simplify the problem of training programmers in the use of the AOP Assembler, the structure of program statements closely follows that of other widely used assemblers, such as SLEUTH II for the Univac 1108 or Meta-Symbol for the XDS 920, with which they may already be acquainted.

## Input Format

The assembler accepts cards in free form, that is, blanks delimit fields. Each card contains up to four fields:

1. Label field. Must start in column 1 and begin with an alphabetic character. This field need not be present. This is indicated by a blank in Column 1.

2. Operation field. Starts with first non-blank after label field.

3. Operand field. Starts with first non-blank after operation field. This field is present only if the operation requires an operand field.

4. Comments field. Starts with first non-blank after last required field. The assembler ignores the contents of this field and it is the only field which may contain imbedded blanks.

The example below shows the input cards which would be generated to check whether a point lies on the locus of the unit circle. As in any fixed point arithmetic involving fractions the programmer must do the scaling.

```
LOCUS   LDA   X        Scaled 2 Minus 3
        MUL   X        Scaled 2 Minus 3
        DSH   XM3      Shift to Scale 2 Minus 3 (XM3 = -3)
        STA   X2
        LDA   Y        Scaled 2 Minus 4
        MUL   Y        Scaled 2 Minus 4
        DSH   XM5      Shift to Scale 2 Minus 3 (XM5 = -5)
        ADD   X2
        BRM   SQRT     Get Square Root
        STA   NORM     Scaled 2 Minus 3
        TAE   (010)    Test Equal to 1 (Scaled 2-3)
        BRC   (COMP)
```

Each line represents an input card, and the mnemonics are those recognized by the standard assembler. A feature of the assembler is its procedure capability which allows the user to define his own mnemonics. This is discussed further in connection with the PROC directive. Literals can be entered in the operand field of an instruction by enclosing the desired expression in parenthesis.

Registers — Registers which can be affected by program execution are listed in Table 1.

TABLE 1

Central Processor Unit Functional Registers

| Register | Symbol | Length (bits) | Function |
|---|---|---|---|
| Accumulator | ACC | 18 | Used for operand storage. |
| Extended Accumulator | EA | 18 | Holds least significant half of a double length operand in multiply/divide operations. |
| Storage limit | SLR | 18 | Controls where writing into memory is permitted. |
| Index | X | 18 | Added to address to form effective address if index bit in instruction word is set. |
| Page | P | 4 | Appended to 12 bit address field to form 16 bit address. |
| Carry | C | 1 | Stores a carry out of bit 17 of the parallel adder. |
| Decision | D | 1 | Conditionally set when executing test instructions. Reset by the conditional transfer instruction. |
| Overflow | OV | 1 | Stores the overflow condition. |
| Activation status | ASR | 16 | Inhibits or allows cycle steal operations on specific channels. |
| Lockout status | LSR | 16 | Inhibits or allows specific interrupts. |
| Interrupt override | IOR | 1 | Inhibits or allows all interrupts except interrupt 0. |

Instruction Set — The AOP has 55 instructions, 31 of which require a memory access. The other 24 instructions have a minor operation code in the operand field of the instruction word. The formats are as follows:

MEMORY ACCESS

| $B_{18}$ $B_{17}$ $B_{16}$ $B_{15}$ $B_{14}$ | $B_{13}$ | $B_{12}$ $B_{11}$ $B_{10}$ $B_9$ $B_8$ $B_7$ $B_6$ $B_5$ $B_4$ $B_3$ $B_2$ $B_1$ |
|---|---|---|
| Operation | Index | Operand address |

NON-MEMORY ACCESS

| $B_{18}$ $B_{17}$ $B_{16}$ $B_{15}$ $B_{14}$ | $B_{13}$ ---------------- $B_6$ | $B_5$ $B_4$ $B_3$ $B_2$ $B_1$ |
|---|---|---|
| 0 0 0 0 0 | | |
| | Not used | Operation |

With 12 bits for the address, 4096 memory words are directly addressable. A memory size as large as 65,536 words requires a 4-bit page register which can be loaded and stored under program control and which is appended as four high-order bits to the 12-bit address field to form a full 16-bit effective address. If the index bit is set, the low order 16-bits of the index register are added to the address to form an effective address and the execution time is increased by 2.0 cycles. The thirty-one instructions which require a memory access may be indexed. Indexing is specified by appending a comma to the operation mnemonic. For example, 'LDA,' will result in a load accumulator instruction with bit 13 set so that the index register will be added to the address field at execution time to form the effective address.

Figure 1 gives a summary of the AOP instruction set by function. Appendix A gives the detailed operations of each instruction. Execution times are given in CPU cycles. At present, one CPU cycle is nominally one microsecond.

Note that the test instructions which set the decision register do not reset it to zero if the test fails. Thus a series of tests can be 'or'ed together. A conditional branch at the end of this series will take place if any of the tests succeeded.

## Assembler Directives

Assembler directives are used to pass information to the assembler concerning a particular program to be assembled. The assembler directives are loaded in with the source program as mnemonics in the operation field of the card. These directives have effect only for the program with which they are assembled and their effect begins when they are encountered during the assembly process. For convenience, the assembler directives can be grouped into four categories according to their usage or function in the program. These categories are:

a. Control of storage allocation

RES — Reserve storage
RORG — Set relocatable origin
AORG — Set absolute origin
LIT — Assign control counter to literals

ORDERED BY FUNCTION

LOAD/STORE INSTRUCTIONS

| Op Code | Mnemonic | Cycle Time* | Description |
|---|---|---|---|
| 20 | LDA | 4 | Load accumulator |
| 40 | LDL | 4 | Load accumulator with effective address |
| 12 | LDI | 6 | Load accumulator indirect |
| 52 | LDE | 4 | Load extension |
| 54 | LDX | 4 | Load index |
| 60 | STA | 6 | Store accumulator |
| 32 | STI | 8 | Store accumulator indirect |
| 10 | STE | 6 | Store extension |
| 74 | STX | 6 | Store index |
| 000013 | LDD | 3 | DSH left 1 bit then load decision register into LSB of EA |
| 000012 | LDP | 3 | Load page register from accumulator |

ARITHMETIC INSTRUCTIONS

| Op Code | Mnemonic | Cycle Time* | Description |
|---|---|---|---|
| 000004 | NEG | 6 | Two's complement accumulator |
| 000006 | ADC | 4 | Add carry to accumulator |
| 000010 | CMP | 6 | One's complement accumulator |
| 000014 | NORM | 4** | Normalize accumulator and extension |
| 02 | ADX | 4 | Add memory to index |
| 04 | ADD | 4 | Add memory to accumulator |
| 24 | SUB | 4 | Subtract memory from accumulator |
| 44 | MUL | 32*** | Multiply accumulator by memory |
| 64 | DIV | 58 | Divide accumulator and extension by memory |

* Add 2 cycles for indexing.
At present, one CPU cycle is nominally one microsecond.
** Add one cycle for each place shifted.
*** Average

Figure 1. AOP Instruction Set

BOOLEAN LOGIC INSTRUCTIONS

| Op Code | Mnemonic | Cycle Time* | Description |
|---|---|---|---|
| 30 | ETR | 4 | Logical AND accumulator with memory |
| 50 | MRG | 4 | Logical OR accumulator with memory |
| 70 | EOR | 4 | Exclusive OR accumulator with memory |

I/O INSTRUCTIONS

| Op Code | Mnemonic | Cycle Time* | Description |
|---|---|---|---|
| 16 | OPT | 6 | Output |
| 76 | IPF | 6 | Input |

REGISTER MANIPULATION INSTRUCTIONS

| Op Code | Mnemonic | Cycle Time* | Description |
|---|---|---|---|
| 14 | SHF | 5** | Arithmetic shift accumulator |
| 36 | DSH | 5** | Arithmetic shift accumulator and extension |
| 34 | CYC | 5** | Cyclic shift accumulator |
| 56 | DCY | 5** | Cyclic shift accumulator and extension |
| 000025 | ACX | 8 | Exchange accumulator and index |
| 000026 | AEA | 8 | Exchange accumulator and extension |
| 000027 | EAX | 8 | Exchange extension and index |
| 000022 | FLP | 3 | Reverse accumulator |

* Add 2 cycles for indexing.
At present, one CPU cycle is nominally one microsecond.
** Add one cycle for each place shifted.

**Figure 1. AOP Instruction Set (Continued)**

CONTROL / BRANCH INSTRUCTIONS

| Op Code | Mnemonic | Cycle Time* | Description |
|---|---|---|---|
| 000000 | HLT | 3 | Halt |
| 000002 | NOP | 3 | No operation |
| 06 | BRM | 8 | Branch and mark place |
| 62 | BRU | 4 | Branch unconditionally |
| 42 | BRC | 4 | Branch conditionally |
| 72 | TIN | 22 | Restore status registers from memory |
| 000016 | EXIT | 36 | Cause exit interrupt |

TEST / SET INSTRUCTIONS

| Op Code | Mnemonics | Cycle Time* | Description |
|---|---|---|---|
| 000001 | TOV | 3 | Test overflow register |
| 000003 | TAP | 3 | Test accumulator positive |
| 000005 | TOP | 22 | Test accumulator for odd parity |
| 000007 | ROV | 3 | Reset overflow register |
| 000017 | CPD | 3 | Complement decision register |
| 000020 | SIO | 3 | Set interrupt override |
| 000021 | TAZ | 4 | Test accumulator for zero |
| 000023 | RED | 3 | Reset decision register |
| 000024 | RIO | 3 | Reset interrupt override |
| 000011 | TIX | 6 | Test index for zero and increment |
| 000015 | TIE | 6 | Test extension for zero and increment |
| 22 | TXLE | 4 | Test index less than or equal to memory |
| 26 | TAL | 4 | Test accumulator less than memory |
| 46 | TAE | 4 | Test accumulator equal to memory |
| 66 | TAG | 4 | Test accumulator greater than memory |

* Add 2 cycles for indexing.
At present, one CPU cycle is nominally one microsecond.

Figure 1. AOP Instruction Set (Continued)

b. Data word generation

e — Enter value of expression in storage

c. Control of code generation

EQU — Equate label to operand field

PROC — Procedure definition

END — End of procedure or assembly

d. Control of assembly listing

UNLS — Don't list following cards

LIST — List following cards

PAGE — Start listing at top of new page

Control of Storage Allocation

| label RES expression |
|---|

Expression is evaluated and that many storage locations reserved. Subsequent references to label refer to the first storage location reserved. No specific value can be expected in the reserved locations at execution time. If expression is zero, no locations are reserved and label will refer to the next location allocated in the assembly.

| RORG expression 1, expression 2 |
|---|

The value of expression 2 is assigned to the location counter specified by the value of expression 1. Storage locations allocated under control of this location counter will be considered relocatable.

AORG expression 1, expression 2

The value of expression 2 is assigned to the location counter specified by the value of expression 1. Storage locations allocated under control of this location counter will be considered absolute.

LIT expression

Literals generated during assembly will be assigned storage under control of the location counter specified by expression

The AOP Assembler uses two location counters to allocate contiguous blocks of relocatable storage. Odd numbers refer to the location counter used for program areas and even numbers refer to the location counter used for data areas. In a given assembly, both location counters start at zero. The appropriate counter is incremented by one when a memory location is allocated to it. At the end of the assembly the largest value assigned to a location counter is the size of the block allocated to it. The loader will relocate each block relative to other blocks allocated under the same location counter so that it winds up with two large relocatable blocks: one for the odd, the other for the even location counter. With this in mind, the four directives for control of storage allocation could be re-defined as simple manipulation of the values of location counters during assembly.

Data Word Generation

```
label expression
```

The value of expression is inserted in storage under control of the current location counter. Whenever the assembler detects a non-alphabetic character as the first character of the operation field, that card is treated as a data word generation card. The value assembled into the location can be referenced by label. The form 0 + name can be used to generate a relocatable address as a data word.

Control of Code Generation

```
label EQU expression
```

Label is assigned the value of expression.

It is often desirable to write an assembly program in such a way that the resultant program is parameterized. That is, the program will do its job for any of a wide range of parameter values. The size of a buffer or a particular I/O device number, as in Figure 2, might be a parameter. The EQU directive provides a means whereby the parameter value can be entered at assembly time. A new program could be generated for a different parameter value by simply changing one EQU card and re-assembling.

```
label PROC
```

Begins a procedure (macro) definition. References to label in the operation field of an instruction will cause insertion of the procedure at that point in the coding.

Programming time can be saved or operation mnemonics redefined through use of the PROC directive. When the programmer writes

```
    label        A,B,C
```

for example, and has previously defined a procedure using the PROC control card

```
label PROC
```

followed by:

```
LDA    LABEL(0)
ADD    LABEL(1)
STA    LABEL(2)
END
```

the code generated by the assembler is as though

```
LDA    A
ADD    B
STA    C
```

were written. Coding time is saved by putting commonly performed sequences of coding in PROC's as it is easier to change one PROC than several similar coding sequences scattered throughout a program. If the programmer prefers the old assembler mnemonic IGZ (for is positive) rather than the new standard instruction mnemonic, TAP (for test accumulator positive), the proc

```
IGZ     PROC
        TAP
        END
```

will allow him to use IGZ in the operation field rather than TAP.

| |
|---|
| END |

Terminates a procedure definition or a program.

Control of Assembly listing

| |
|---|
| UNLS |

Terminates the normal assembly listing until a LIST card is encountered.

| |
|---|
| LIST |

Continue the normal assembly listing.

| |
|---|
| PAGE |

Continue the normal assembly listing at the top of the next page.

Relocatable Loader

The assembler produces relocatable code and data except when it encounters a directive, such as AORG, which uniquely specifies where this code or data will be located. This means that code or data addresses are relative to the beginning code or data address assigned by the loader such that programs and data sets will be automatically stacked in core without overlap and without unused storage locations. The beginning bias for data and code is presently 210 and 4000 octal respectively, where locations 0-207 are used for interrupt storage, and 4000 octal is the mid-point of a 4K memory module. As will be seen in the control card description, these starting biases may be altered.

Figure 2 shows the assembly of a program for inputting data and a subroutine for adding up the values input. Figure 3 shows how the loader will relocate these programs in core while Figure 4 shows the loader printout from which Figure 3 was derived. Making all programs relocatable is a great advantage since changes in one program do not affect the others. The loader resolves all undefined references at load time automatically.

Another feature of the loader is that a binary tape can be built containing many assembled program segments. Any or all of the assemblies (maximum of 25) can be selectively loaded in the order they appear on the tape.

Control Cards

All control cards must have a ; (11/8/6) punched in column one. The first fields of the control statement may begin in any column after column one. Preceding blanks are disregarded. Thereafter, one or more blanks are used as delimiters. No control statement can be continued onto a second card. Anywhere number appears, a decimal radix will be assumed unless OCTAL is specified; i.e., 11 is $11_{10}$, but OCTAL 11 is $9_{10}$.

```
: ASSEMBLE INPUT

ASSEMBLER FOR THE ONBOARD PROCESSOR
  1.                       000002         CHAN     EQU    2          CHAN IS A PARAMETER
  2.                                      $(1)
  3.          1 000000 54 0 0004                   LDX    (START)    LOAD LITERAL CONTAINING START ADDRESS
  4.          1 000001 76 0 0002          LOOP     IPF    CHX        CHX CONTAINS DEVICE NUMBER
  5.          1 000002 20 0 0003                   LDA    CHX+1      CHX+1 CONTAINS THE DATA JUST INPUT
  6.          1 000003 61 1 0000                   STA.   0          MOVE THE DATA INTO A BUFFER
  7.          1 000004 02 0 0005                   ADX    (1)        INCREMENT INDEX
  8.          1 000005 22 0 0006                   TXLE   (STOP)     TEST INDEX AGAINST STOP ADDRESS
  9.          1 000006 42 0 0007                   BRC    (LOOP)
 10.   U      1 000007 06 0 0000                   BRM    ADDER      ADDER IS UNDEFINED IN THIS ASSEMBLY
 11.          1 000010    000000                   HLT
 12.                                      $(0)
 13.          0 000000                    START*   RES    1
 14.          0 000001                    STOP*    RES    1          START AND STOP ARE EXTERNAL DEFINITIONS
 15.          0 000002    000002          CHX      0+CHAN
 16.          0 000003                             RES    1
 17.                                               END
              0           000004 000000
              0           000005 000001
              0           000006 000001
              0           000007 000001

        END OF LISTING.    1 LINES FLAGGED.
: ASSEMBLE ADDER

ASSEMBLER FOR THE ONBOARD PROCESSOR
  1.                                      $(1)
  2.          1 000000 54 0 0002          TOP      LDX    (START)    SET INDEX TO START ADDRESS OF BUFFER
  3.          1 000001 21 1 0000                   LDA.   0          LOAD FIRST VALUE INPUT
  4.          1 000002 02 0 0003          LOOP     ADX    (1)        INCREMENT INDEX
  5.          1 000003 22 0 0004                   TXLE   (STOP)     START AND STOP ARE UNDEFINED IN THIS ASSEMBLY
  6.          1 000004 42 0 0005                   BRC    (GOON)
  7.          1 000005 62 0 0000                   BRU    ADDER
  8.          1 000006 05 1 0000          GOON     ADD.   0          SUM NEXT BUFFER ITEM
  9.          1 000007 62 0 0006                   BRU    (LOOP)
 10.                                      $(0)
 11.          0 000000                    ADDER*   RES    1
 12.          0 000001    000000                   0+TOP  THESE TWO WORDS ARE USED BY THE BRM INSTRUCTION IN INPUT
 13.                                               END
       U      0           000002 000000
              0           000003 000001
       U      0           000004 000000
              0           000005 000006
              0           000006 000002

        END OF LISTING.    2 LINES FLAGGED.
 LOAD INPUT

 OADER FOR ON-BOARD PROCESSOR SOFTWARE
```

Figure 2. Assembly of a Main Program, INPUT, and a Subroutine, ADDER

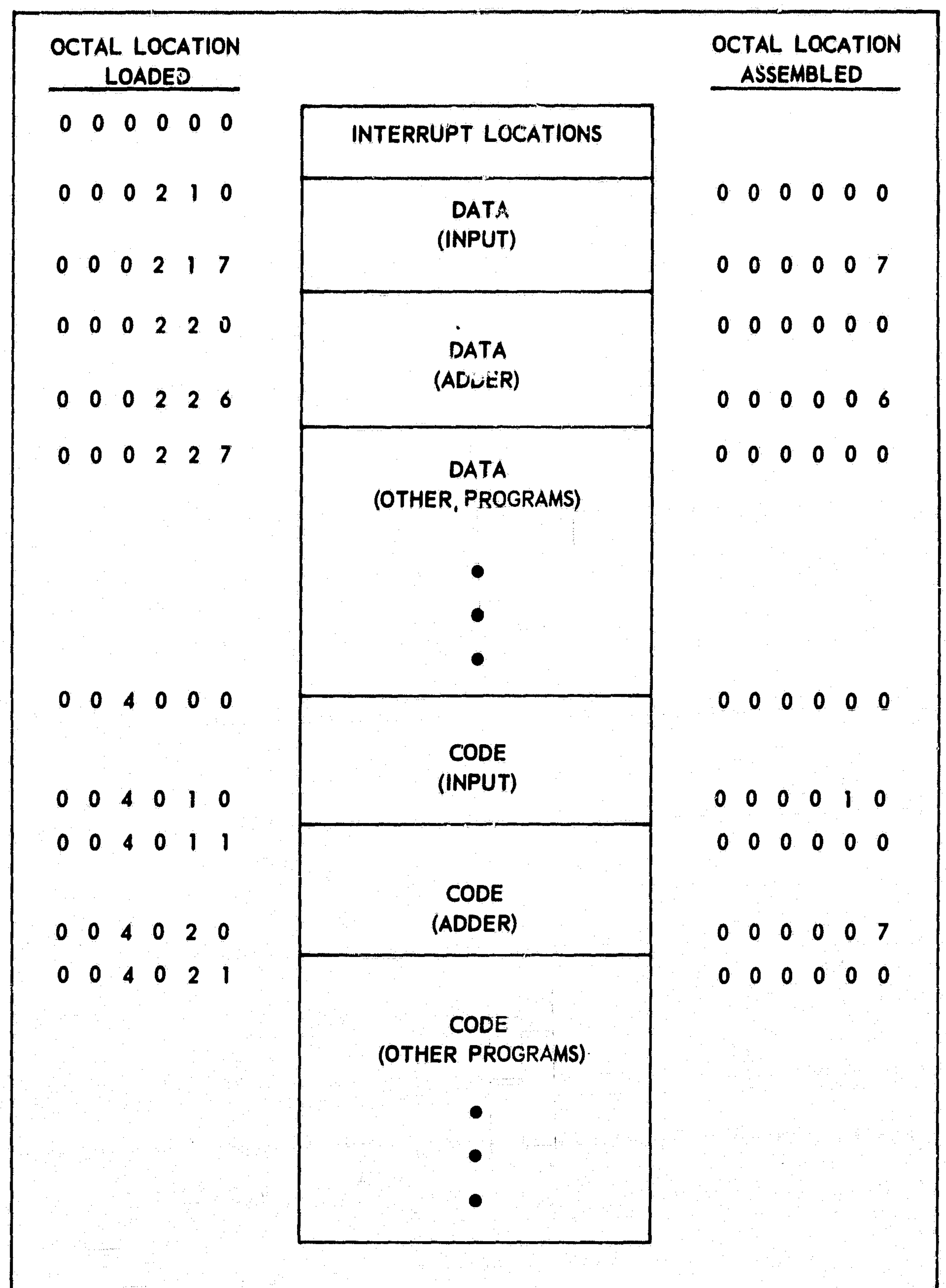



**Figure 3. Core Image Diagram of Automatic Code and Data Relocation**

```
PREAMBLE VALUES FOR INPUT
  DATA LENGTH                    8
  CODE LENGTH                    9
  PRESET LOCATIONS               1
  LITERALS                       3
  INDIRECT ADDRESSES             1
  EXTERNAL DEFINITIONS           2
  UNDEFINED SYMBOLS              1
  NOUNS                          4


EXTERNAL DEFINITIONS
  START
  STOP


PREAMBLE VALUES FOR ADDER
  DATA LENGTH                    7
  CODE LENGTH                    8
  PRESET LOCATIONS               1
  LITERALS                       3
  INDIRECT ADDRESSES             2
  EXTERNAL DEFINITIONS           1
  UNDEFINED SYMBOLS              2
  NOUNS                          4


EXTERNAL DEFINITIONS
  ADDER


   CORE LIMITS
         DATA 000210-000406 / CODE 004000-004020
         STARTING ADDRESS 004000


   CORE ALLOCATION

         INPUT
              DATA 000210-000217
              CODE 004000-004010
         ADDER
              DATA 000400-000406
              CODE 004011-004020

   END OF ALLOCATION
```

**Figure 4. Loader Output for INPUT and ADDER**

; DATE characters

This card causes the first 12 nonblank (blanks are delimiters) characters to be printed as a part of the heading printed by the major functions (assembler, loader, etc.). For example, 7/4/8/,1400 could be used for date and time on the listing for the current run. Note the use of a comma rather than a blank as a separator for date and time.

; ASSEMBLE name (NOLIST)

This card causes the support software executive to call in the assembler to assemble a program designated by name and to inform it of the options desired by the user. The output of the assembler is accumulated on the assembly tape. Only the first twelve characters of name are retained, any remaining characters are ignored. There must not be any blanks interspersed in a program name. The NOLIST field is optional.

Options: NOLIST — This option directs the assembler to suppress its printout. If this option is not present, then a complete listing of the program being assembled will be given.

; MEMORY SIZE IS number BANK(S)

This card designates the memory size in bank units to be loaded, simulated, or listed. The variable number must have a value from 1 to 16 (a bank is 4096 words). If this control card is missing, a memory size of 4096 words, or 1 bank, is assumed.

| |
|---|
| program name 1, program name 2, . . . , program name n |
| ; LOAD $ |
| ; LOAD program name |
| ; LOAD program name DATA AT number AND CODE AT number |

The ; LOAD program name card causes the loader to load the entire assembly tape into core and then writes the Advanced On-Board Processor core image onto the absolute core image tape. The loader will assume a starting location of octal 210 for data.

The assumed starting location for code is one half the memory size. Thus when an 8K memory is being loaded, a starting location of octal 10000 is assumed for code, whereas octal 4000 is assumed when loading a 4K memory. The relative origins assumed by the loader may be altered by using the optional ; LOAD program name DATA AT number AND CODE AT number card. Care must be exercised to prevent data words from overlapping a bank of 4096 words. For example, if a program is loaded with data beginning at location 4000 and there are 100 words of data, then the first 96 words must be accessed with a page register setting of zero, and the remaining four words must be accessed with a page register setting of one. This problem must be taken into consideration by the programmer prior to assembly.

The ; LOAD $ indicates selective loading to the loader. The program names listed on the following cards will be loaded from the assembly tape. There must be a blank in column 1 of the program name cards but as many cards as needed may be used. The number of specified programs is limited to 25. The order in which the specified programs are loaded is the order in which they appear on the assembly tape. The relative origins assumed by the loader during selective loading may be altered by using the optional ; LOAD $ DATA AT number AND CODE AT number card.

```
; WAIT number
```

This card will cause the OBP loader routine to pause. This option is included to allow the XDS 920 user to switch assembly tapes to be loaded. The value of number should be the count of assembly tapes to be loaded. The number of pauses will be one less than the value of number as there will be no pause when the last assembly tape has been loaded.

```
; REWIND ABSOLUTE CORE IMAGE TAPE
; REWIND ASSEMBLY TAPE
```

Either of these cards causes the specified tape to be rewound. The assembly tape should not be rewound between an assemble and a load function.

; PAUSE

This card will cause the support software executive to pause. This option is included to allow the XDS 920 user to switch tapes, save tapes, or mount tapes if necessary.

; END OF FILE ON ASSEMBLY TAPE

This card causes an end-of-file record to be written on the assembly tape. This is to be used if, and only if, the file of relocatable programs on the tape is to be used at a later time.

; SAVE PREVIOUS ASSEMBLIES

This card causes the support software executive to space down the assembly tape until an end-of-file (EOF) record is read. The assembly tape is then backspaced over the EOF record, thus positioning it for further assemblies.

name name name

; DELETE $

; DELETE name FROM PREVIOUS ASSEMBLIES

This card causes the support software executive to search the assembly tape and delete the assembly specified. All other assemblies are preserved. The end-of-file record is removed, and the assembly tape positioned for further assemblies. If a routine is to be reassembled with an assembly tape containing a previous assembly by the same name, the above card must be used to remove the old routine before the new routine is assembled.

The alternate form DELETE $ causes the program names listed on the following cards to be deleted from the assembly tape. These cards must have a blank in column one, up to 25 program names can be specified. The names need not be in any particular order. As many cards as desired may be used.

```
; LIST
; LIST THE NOUN TABLE (ALPHABETICALLY NUMERICALLY)
```

This card causes the support software executive to read in the absolute core image tape prepared by the loader. It then will list the complete symbol table of all the loaded programs. The order of the two options is irrelevant and either one or both may be omitted. If both are omitted, then numeric and alphabetic lists will be given. Both lists may also be obtained with the abbreviated control card ; LIST.

Options: ALPHABETICALLY — This option causes the alphabetically ordered symbol table to be printed.

NUMERICALLY — This option yields a printed list of the symbols used ordered on the relocated addresses assigned to the symbols.

; LIST THE ABSOLUTE CORE IMAGE TAPE

This card causes a complete listing of the absolute core image. The core image, allocation table, and symbol table is read from the absolute core image tape produced by the loader. The allocation table is then used to list data and code for each program segment. Data is listed in an octal format. Code is listed as the octal bit pattern for each instruction, with decoded mnemonics and labels as they were defined in the program. All indirect instructions are flagged with the indirect address.

; CHECK PRINT number

This card causes the support software executive to turn on debugging flags within the AOP software package. This control option is provided as an aid in maintaining the AOP package and is not normally used.

Simulator

The AOP Simulator reads the absolute core image tape, created by the loader, into core and simulates the execution of that program. When a HALT instruction is encountered, the simulator prints out statistics concerning simulated running time and frequency of instruction usage. By means of various control cards, the simulator may be made to give selective tracing and/or dumping in octal. Control cards are also available for specifying periodic interrupts, simulation of input-output from the I/O unit, and such miscellaneous capabilities as halts treated as no-ops and restarting after a halt has been executed. During execution the simulator may, for one of several reasons, enter a routine called the pseudo console which allows the user to examine and alter the contents of simulated registers or memory locations.

Interrupt Simulation — At the completion of each simulated instruction, the interrupt processor determines whether any of the 15 external interrupts appeared during the simulation of the previous instruction. When an interrupt appears, the appropriate bit is set in the ISR (Interrupt Status Register). If the interrupt cannot be honored immediately, it is saved. If an interrupt is currently being saved and another of the same number appears, it is lost. Interrupts are not honored immediately if they are locked out by the LSR (lockout status register) or the IOR (interrupt override register) or by a higher priority pending interrupt. When an interrupt is honored, one instruction of the interrupt routine is executed before any other interrupts can be honored.

Input-Output Simulation — Two kinds of I/O are simulated, program controlled and cycle steal. Program controlled I/O occurs when an OPT (or IPF) instruction is simulated. The content of storage at the effective address is used as an I/O unit specification and data is output from (or input to) the effective address plus one. The actual response of the simulator varies according to the characteristics of the I/O unit.

If the operation is illegal on the requested unit, the message:

* ERROR ILLEGAL IO REQUEST ___unit #___ ___address___ ___type of operation___

is printed. Type of operation is 1 for cycle steal, 2 for IPF, 3 for OPT. If an input operation cannot be completed due to lack of data, the message:

OUT OF DATA FOR DEVICE ___unit #___

is printed. No error messages appear if the operation is successful.

Cycle steal I/O occurs when a request is entered in the AOP's Request Status Register, RSR. The simulator allows the user to control the timing of the first request on a channel. Thereafter the device characteristics coded into the simulator determine the rate at which requests are generated.

When a cycle steal operation is simulated the simulator prints:

CS I/O ON ___unit #___ TO ___address___ AT ___time___.

The ILLEGAL I/O REQUEST and OUT OF DATA error messages will be printed if appropriate to the cycle steal operation.

When a cycle steal I/O request enters the AOP's Request Status Register, RSR, it contends equally with all other requests for memory access. The diagram of Figure 5 shows the activity in detail. As shown in Figure 5, not all cycle steal I/O requests result in the transfer of a data word to or from memory.

<u>Description of Dumps and Traces</u> — A dump is a printout of the contents of memory and generally comprises two parts: data and code. By means of the

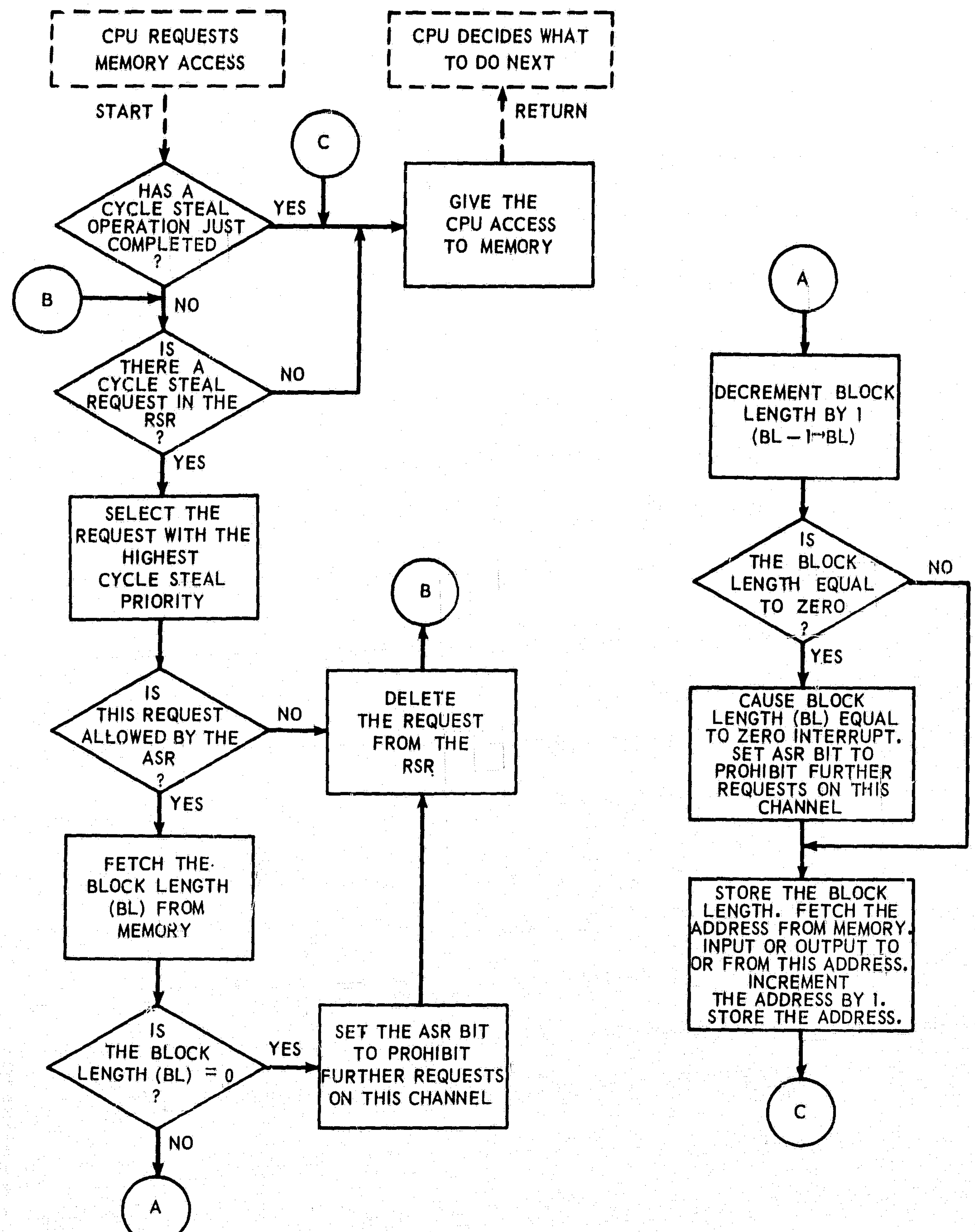



**Figure 5. AOP Memory Contention and Cycle Steal Operations**

several simulator control cards, the user can control the dump to suit his needs. A dump is printed in octal. The entire memory may be dumped or just a specific segment. A dump may be printed upon simulation of a HALT instruction or at any specific point in the program execution. Likewise a trace, the printout of register contents during execution, has almost as many optional forms as the dump and is also user controlled to suit specific needs. Examples of dumps and traces with their various columns explained are contained in the following paragraphs.

Octal Dump — An octal dump is a printout of memory within specified limits at a particular time. There are two columns of initial octal addresses and 16 columns of octal code or data words (refer to Figure 6). Starting from the left, column 1 and column 10 list the initial addresses of the following eight memory locations. Each set of eight columns (columns 2 through 9 and columns 11 through 18) presents a printout of the contents of memory at that particular location. The entire dump is broken into several parts, the first of which is always a printout of the contents of the interrupt locations. The remainder of the dump comprises a printout of the contents of memory at the locations occupied by the various programs and subroutines located within the limits of the dump. The name of the program or subroutine is given as a heading and is followed by the initial and final addresses of the segment of memory which it occupies. Each program or subroutine is dumped in two parts: data (an octal

```
INTERRUPT LOCATIONS

000200  000000 000000 000000 000000 000000 000000 000000 000000   000210  000000 000000 000002 600000 000210 000001 000211 004001
    INPUT         (DATA) 000210-000217

000200  000000 000000 000000 000000 000000 000000 000000 000000   000210  000000 000000 000002 600000 000210 000001 000211 004001
    INPUT         (CODE) 004000-004010

004000  540214 760212 200213 610000 020215 220216 420217 060400   004010  000000 540402 210000 020403 220404 420405 620400 050000
    ADDER         (DATA) 000400-000406

000400  004010 004011 000210 000001 000211 004017 004013 000000   000410  000000 000000 000000 000000 000000 000000 000000 000000
    ADDER         (CODE) 004011-004020

004000  540214 760212 200213 610000 020215 220216 420217 060400   004010  000000 540402 210000 020403 220404 420405 620400 050000
004020  620406 000000 000000 000000 000000 000000 000000 000000   004030  000000 000000 000000 000000 000000 000000 000000 000000
REMAINING CORE IN DATA REGION

000400  004010 004011 000210 000001 000211 004017 004013 000000   000410  000000 000000 000000 000000 000000 000000 000000 000000
REMAINING CORE IN CODE REGION

004020  620406 000000 000000 000000 000000 000000 000000 000000   004030  000000 000000 000000 000000 000000 000000 000000 000000
```

**Figure 6. Octal Dump**

listing of the data words and their addresses) and code (an octal listing of the code words and their addresses).

Octal Trace — An octal trace is the printout of register contents during execution. The following explains the various trace columns (refer to Figure 7).

| Column | Explanation |
|---|---|
| IC | This column lists the value contained in the Instruction Counter for each simulated instruction. The words that appear in this column but stand alone on a line are the labels used in BRU, BRC, BRM, TIN and EXIT instructions. |
| INST | This column gives a 4-letter abbreviation of the instruction just simulated. |
| OPERAND | This lists the first twelve letters of the operand name used with the instruction if the name is available. |
| [EAD] | This is a printout of the value contained in memory at the effective address after each simulated instruction. |
| AC | This column shows the value contained in the accumulator after each simulated instruction. |
| EA | This column shows the value contained in the extended accumulator after each simulated instruction. |
| ICDO | This lists the value contained in each of the following one-bit registers: interrupt override, carry, decision, and overflow after each simulated instruction. |
| [IC] | This is a printout of the contents of memory at the address indicated by the instruction counter prior to simulation of each instruction. This column provides an octal listing of the instructions executed. |
| P | This lists the value of the page register after each simulated instruction. |

```
 IC    INST    OPERAND     (EAD)   AC     EA     ICDO   (IC)   P INDEX  X  EAD    SLR      TIME  I  LSR    ISR    ASR

 004000 IDX               000210 000000 000000  1000  540214 00 000210 0 000214 000000     4     000000 000000 000000
 004001 IPF   CHX         000002 000000 000000  1000  760212 00 000210 0 000212 000000    10     000000 000000 000000
 004002 IDA   CHX         600000 600000 000000  1000  200213 00 000210 0 000213 000000    14     000000 000000 000000
STORAGE PROTECT VIOLATION AT 004003  000210
 004003 STA               000000 600000 000000  1000  610000 00 000210 1 000210 000000    22     000000 100000 000000
 004004 ADX               000001 600000 000000  1000  020215 00 000211 0 000215 000000    26     000000 100000 000000
 004005 TXLE              000211 600000 000000  1010  220216 00 000211 0 000216 000000    30     000000 100000 000000
 004006 HPC   LOOP        004001 600000 000000  1000  420217 00 000211 0 000217 000000    34     000000 100000 000000
LOOP
 004001 IPF   CHX         000002 600000 000000  1000  760212 00 000211 0 000212 000000    40     000000 100000 000000
 004002 IDA   CHX         600000 600000 000000  1000  200213 00 000211 0 000213 000000    44     000000 100000 000000
STORAGE PROTECT VIOLATION AT 004003  000211
 004003 STA               000000 600000 000000  1000  610000 00 000211 1 000211 000000    52     000000 100000 000000
 004004 ADX               000001 600000 000000  1000  020215 00 000212 0 000215 000000    56     000000 100000 000000
 004005 TXLE              000211 600000 000000  1000  220216 00 000212 0 000216 000000    60     000000 100000 000000
 004006 HRC   LOOP        004001 600000 000000  1000  420217 00 000212 0 000217 000000    64     000000 100000 000000
 004007 FRM   ADDER       004010 600000 000000  1000  060400 00 000212 0 000400 000000    72     000000 100000 000000
ADDER
 004011 IDX               000210 600000 000000  1000  540402 00 000210 0 000402 000000    76     000000 100000 000000
 004012 IDA               000000 000000 000000  1000  210000 00 000210 1 000210 000000    82     000000 100000 000000
 004013 ADX               000001 000000 000000  1000  020403 00 000211 0 000403 000000    86     000000 100000 000000
 004014 TXLE              000211 000000 000000  1010  220404 00 000211 0 000404 000000    90     000000 100000 000000
 004015 HRC   GOON        004017 000000 000000  1000  420405 00 000211 0 000405 000000    94     000000 100000 000000
GOON
 004017 ADD               000000 000000 000000  1000  050000 00 000211 1 000211 000000   100     000000 100000 000000
 004020 HRU   LOOP        004013 000000 000000  1000  620406 00 000211 0 000406 000000   104     000000 100000 000000
LOOP
 004013 ADX               000001 000000 000000  1000  020403 00 000212 0 000403 000000   106     000000 100000 000000
 004014 TXLE              000211 000000 000000  1000  220404 00 000212 0 000404 000000   112     000000 100000 000000
 004015 HRC   GOON        004017 000000 000000  1000  420405 00 000212 0 000405 000000   116     000000 100000 000000
 004016 ERU   ADDER       004010 000000 000000  1000  620400 00 000212 0 000400 000000   120     000000 100000 000000
ADDER
 004010 HLT               000000 000000 000000  1000  000000 00 000212 0 000000 000000   123     000000 100000 000000
```

Figure 7. Octal Trace

| Column | Explanation |
|---|---|
| INDEX | This column gives the value contained in the index register after each simulated instruction. For each instruction, the value listed here is a one if bit 13 of the instruction contains a one, thus designating use of the index register to determine the effective address. |
| EAD | This gives an instruction-by-instruction listing of the contents of the effective address register. The content of this register specifies the memory address of the operand. |
| SLR | This column lists the contents of the storage limit register after each simulated instruction. |
| TIME | This column presents the cumulative total time in CPU cycles for the execution of instructions. |
| I | This is a listing of the octal interrupt number being processed if any interrupts have occurred during the simulation. |
| LSR | This column shows the octal value of the lockout status register at the end of execution of the current instruction. |
| ISR | This column shows a pending interrupt request by setting the appropriate bit to one. Interrupt 0 is the LSB of the octal value. Interrupt 4 would be indicated if the ISR were 000020. An interrupt is pending, for the purposes of this column, if the IOR is 1 and the interrupt is not number 0, if the corresponding LSR bit is 1, or if execution of the one instruction allowed after an interrupt occurs is taking place. |
| ASR | This column shows the contents of the activation status register. |

## Simulator Control Cards

The format for the simulator control cards is the same as the format for all control cards. Column one must contain a semicolon (an 11/8/6 punch) and columns 2 through 80 contain the control information. One or more blanks are used as delimiters. No control function may continue onto a second card.

The control cards for the simulator are grouped into the following six categories: starting, tracing, dumping, interrupting, inputting and stopping (and/or restarting).

To simulate a previously loaded program, the user must first supply a simulate control card followed by any number of simulator control cards to define the kind of simulation desired, followed by a start control card. No non-simulator control cards may intervene.

| |
|---|
| ; SIMULATE |
| ; SIMULATE name |

This control card causes the simulator to be loaded into memory. The simulator sets the instruction counter to the load location of name. If name is omitted, then the instruction counter will be set to the normal, initial load location for instructions. The simulator is then ready to interpret any remaining simulator control cards. This card must be placed between the LOAD control card and the START control card.

| |
|---|
| ; START |
| ; START AT number or label |

This control card, or one of its optional forms, should appear as the last control card. It causes the simulator to commence simulating at the location specified by the SIMULATE card. The optional form, where number or label is specified, causes the simulation to commence at the location specified.

```
; TRACE OCTALLY
```

This control card causes the simulator to print tracing information in the octal mode for each relocatable instruction simulated.

```
; TRACE OCTALLY FROM label or number TO label or number (number TIMES)
```

This card causes the simulator to print octal tracing information for each instruction simulated between the limits specified by label and/or number.

If number TIMES appears, trace information will no longer be printed after the segment between the limits specified has been encountered number times.

```
; TRACE OCTALLY PROGRAM name (number TIMES)
```

This card causes the simulator to print tracing information in the octal mode where the limits of name are taken from the allocation table.

If number TIMES appears, trace information will no longer be printed after the segment between the limits specified has been encountered number times.

; DUMP OCTALLY AT label or number

This card causes an octal dump of the entire memory when the specified location is accessed (either code or data).

; DUMP OCTALLY AT label or number FROM label or number TO label or number

This card causes an octal dump of the segment of memory located within the limits specified by the second and third label or number when the location specified by the first label or number is accessed.

; DUMP OCTALLY AT label or number PROGRAM name

This control card causes an octal dump of the program specified by name, where the limits of name are taken from the allocation table, when the location specified by label or number is accessed.

; DUMP OCTALLY AT HALT

This control card causes an octal dump of the entire memory at the time of simulation of a HALT statement.

; INTERRUPT number EVERY number MICROSECONDS or MILLISECONDS or SECONDS STARTING AT number MICROSECONDS or MILLISECONDS or SECONDS

This card causes the specified interrupt (legal interrupts are 0 through 15) to occur at the specified interval beginning at the specified start time. The user must specify the time units of the interval length and start time in microseconds, milliseconds or seconds.

| |
|---|
| ; INTERRUPT <u>number</u> EVERY <u>number</u> MICROSECONDS or MILLISECONDS or SECONDS |

This control card causes the specified interrupt (legal interrupts are 0 through 15) to occur at the specified interval beginning at time zero. The user must specify the time-units of the interval length as microseconds, milliseconds, or seconds.

| |
|---|
| ; MAXIMUM TIME IS <u>number</u> MICROSECONDS or MILLISECONDS or SECONDS |

This card will cause the simulation to cease at the specified simulation time. If this card is omitted, then a value of 5 milliseconds is assumed.

| |
|---|
| ; MAXIMUM INSTRUCTIONS IS <u>number</u> |

This control card causes the simulation to cease after the <u>number</u> of instructions has been executed. If this card is omitted, a value of 500 instructions is assumed.

<u>Inputting Data</u>

| |
|---|
| ; INITIATE CHANNEL <u>number</u> AT <u>number</u> MICROSECONDS or MILLISECONDS or SECONDS |

This control card causes a cycle steal I/O request at the indicated time over the indicated channel. Channels are numbered from 0 to 15. Further cycle steal I/O will occur under the control of the simulator's I/O device routines.

| |
|---|
| data data data data data data $ |
| ; DATA FOR INPUT DEVICE number |

This is the control card used for inputting data. Data may be input over sixteen different input units numbered zero through fifteen. A total of 218 data words may be specified. Immediately following this card are the cards of input data for the specified input unit. As many cards as needed may be used and the format of these cards is free form with one or more blanks used as delimiters. Column one must be blank. The last data word must be followed by a $.

During the simulation, when the specified unit is referenced in an input operation, the data words are input one word per request until the data buffer is exhausted.

Stopping and/or Restarting — The normal means of ending a simulation is by simulating a HALT. The pseudo console then gains control and the simulation can be terminated by typing STOP. Any remaining control cards will then be honored.

; NO HALT

This control card causes all HALT instructions to be simulated as NOP instructions.

; RESTART AT HALT

This control card causes one HALT instruction to be simulated as a NOP instruction. There must be a RESTART AT HALT card for each HALT that is to be simulated as a NOP instruction.

; STOP AT HALT

This control card allows the HALT statement to be simulated normally. Its main purpose is to allow proper page skipping between multiple jobs.

Pseudo Console

Once simulation has begun, that is, once a START Control card has been read, the user may gain control by setting breakpoint 2. The user can then perform the following functions through the typewriter on the XDS 920. Only the underlined portions in each paragraph title need be typed. Carriage return is indicated by CR. Whenever the user types something the pseudo console routine does not recognize, the message WHAT is printed.

DISPLAY CR

By typing DISPLAY CR the user may then enter any of the following register mnemonics followed by a CR. The pseudo console routine will type the values of the requested register along with several associated registers. Mnemonics are:

| | |
|---|---|
| ACC | Accumulator |
| EA | Extended Accumulator |
| X | Index register |
| SLR | Storage Limits Register |
| IC | Instruction Counter |
| P | Page register |
| C | Carry register |
| OV | Overflow register |
| ASR | Activation Status Register |
| LSR | Lockout Status Register |
| ISR | Interrupt Status Register |
| IOR | Interrupt Override register |
| EAD | Effective Address Register (not a hardware register) |

ENTER CR

By typing ENTER CR the user may then type any of the register mnemonics used for DISPLAY followed by a CR. The user then types in up to 6 octal digits followed by a CR. The pseudo console routine will enter this value, right justified, into the designated register. For all registers, except the ASR, the user enters the value he wants in the register. To enter a value in the ASR, however, the user supplies the address, not the value, of the word to be used to set the ASR. This word can be set up using the STORE command and must be the same bit configuration as would be used were the change to be made using the AOP's OPT instruction.

DUMP CR

By typing DUMP CR the user will cause the pseudo console routine to type out the address in the EAD register followed by the contents of memory at this address. The EAD is incremented so that the next DUMP command will refer to the next sequential memory location. To examine successive sequential locations in memory the user must type DUMP CR for the first, but need only type CR to obtain following locations.

STORE CR

By typing STORE CR the user may then type in up to 6 octal digits followed by a CR. The pseudo console will enter this value, right justified, into the memory location whose address is in the EAD. The EAD is incremented and the user may then enter another value which will be stored in the location following the last and so on. Typing a non-octal digit causes the message WHAT to be typed. The user may then enter another command.

START CR

By typing START CR the user causes simulation to commence at the address in the instruction counter.

STOP CR

By typing STOP CR the user causes the simulation to terminate immediately. The support software system executive gains control immediately and the next control card is read from the card reader.

ADDRESS STOP CR

By typing ADDRESS STOP CR the user insures that the pseudo console will gain control whenever the instruction counter contains a value equal to the octal number typed in following this command. Typing 777777 insures the simulator will not stop since 177777 is the largest AOP address. This must be done if the user has entered an address stop and now wishes to de-activate the feature.

Summary of Pseudo Console Commands

ADDRESS STOP CR

octal value CR

DISPLAY CR

one of: ACC, EA, X, SLR, IC, P, C, OV, D, ASR, LSR, IST, IOR, EAD

then CR

ENTER CR

mnemonic CR

| | |
|---|---|
| octal value CR | Note (if the mnemonic ASR is entered, enter the address of the word to be sent to the ASR rather than the value). |

DUMP CR

STORE CR

octal value CR

START CR

STOP CR

Sample Pseudo Console Session

Lines in capitals are printed by the pseudo console routine.

| | |
|---|---|
| PSEUDO CONSOLE | The user has set BP2, an address stop has been satisfied, or an error has been detected by the simulator. |
| enter | sets up the EAD for subsequent store |
| ead | |
| 7276 | |
| store | |
| 417303 | stores indexed LDL instruction at location 7276. |
| enter | |
| WHAT | pseudo console expected a number |
| display | |

ic

SLR 777000 IC 141110 P 00 C 0 OV 0 D 0

ead

LSR 000000 ISR 000000 IOR 1 EAD 007277

enter

ic

7226

ead

7276

dump

| | | |
|---|---|---|
| 007276 | 02417303 | The contents of simulated memory location 7276 are preceded by two octal digits containing dump and trace codes. These may be ignored. |
| 007277 | 02627216 | The user typed CR to get the next location. |
| start | | Simulation will now continue beginning at location 7226. |

PROGRAMMING NOTES

The support software which has been developed for the AOP was aimed at allowing independent programming by AOP users at different locations. In keeping with this philosophy, it is expected that an executive routine will be provided by a central housekeeping group. This routine will provide an environment within which worker programs can function without being aware of details of AOP hardware I/O, interrupts, timing, or other workers. The items in this section may not, therefore, be of direct interest except to those involved in the programming of the executive.

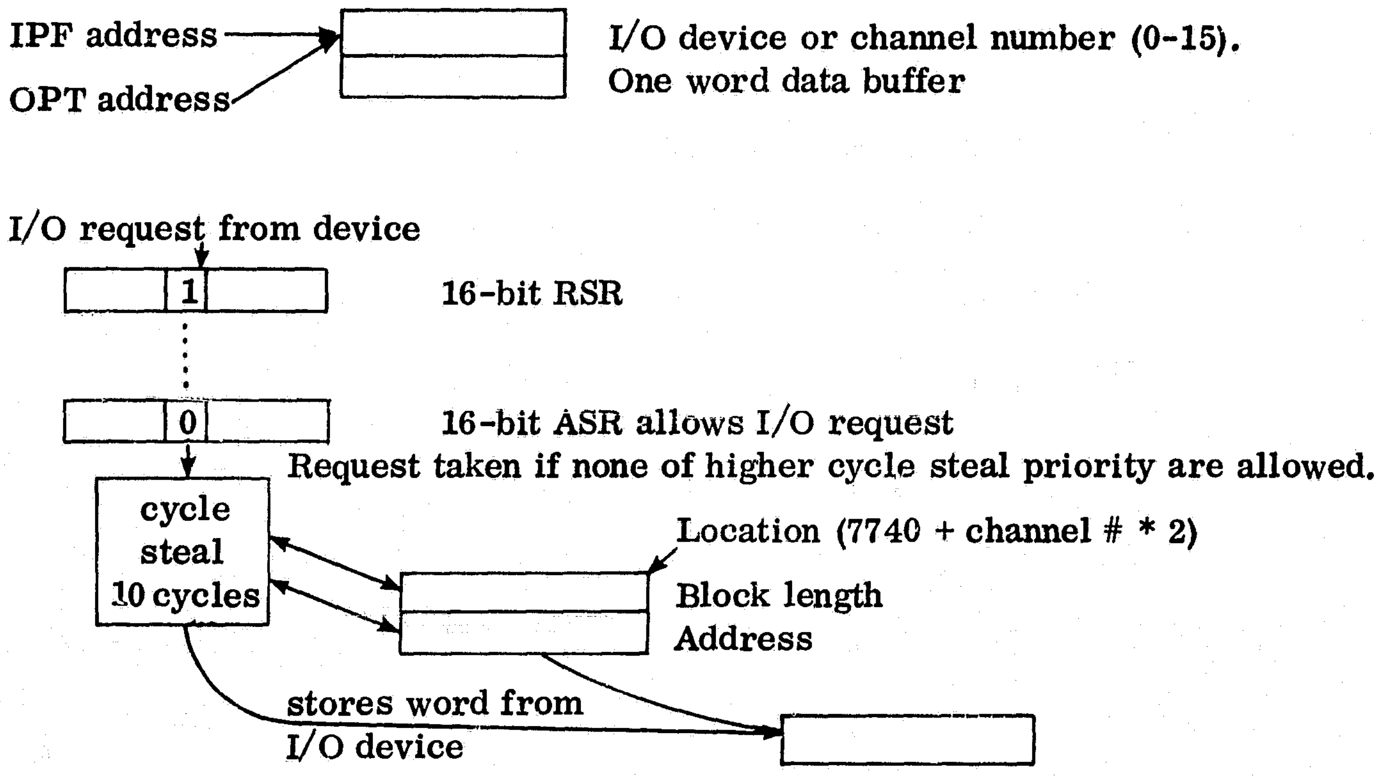
Input/Output

The AOP has two modes of input/output: program-controlled and cycle steal. Both modes can be used on any channel. For program-controlled I/O, either the contents of the effective address+1 is output to a device or the data from a device is input there. For cycle steal operation, there are 16 cycle steal channels in the input/output unit which control block transfers of data between input/output devices and memory. These data transfers are independent of program execution and the external device supplies the I/O request pulses.

One device is connected to each cycle steal channel. Each cycle steal channel requires two memory locations. These 32 locations are situated at the top of the fixed core bank of memory. The first word of each pair contains a block length, the second an address. When a cycle steal request arrives from an external device, the block length is fetched, decremented and stored. Next, the address is fetched and the data word is either stored into or fetched from memory at that address. Then the address is incremented and stored. Cycle steal operations can be inhibited if the corresponding bit of the 16-bit Activation Status Register, ASR, is a one. The ASR is set using program controlled output over channel 10. See Table 2 for the I/O system philosophy.

TABLE 2

I/O System




Should two or more cycle steal requests occur simultaneously, a hard-wired cycle steal priority, which can be altered only in groups of 4 channels, is used to select the first channel to be serviced. The cycle steal operation requires a total of five memory accesses. A CPU memory access is allowed between each cycle steal operation to avoid locking out the CPU by a long queue of cycle steal requests.

Cycle steal I/O rates as high as $10^5$ words/second are possible. Generally, program control of I/O is restricted to 1) devices for which the data transfers are program dependent or 2) very low data rate devices if interrupt of the program is necessary for synchronization.

## Interrupts

There is a 16-bit register called the Interrupt Status Register, ISR, in the CPU which stores interrupt requests. As each request is serviced, the corresponding bit in the register will be reset. There is another 16-bit register in the CPU, called the Lockout Status Register, LSR, where each bit indicates whether or not an interrupt request at that level is to be locked out. A one-bit CPU register called the Interrupt Override Register, IOR, serves to lockout all interrupts, except interrupt 0, when it is set to 1. An interrupt signal sent to the CPU causes an interrupt when the ISR bit is set, the corresponding bit of the LSR is zero and the IOR is zero.

Should two allowable interrupts occur simultaneously, there is a hardwired 16 level priority circuit in the I/O unit (interrupt 0 is of highest priority) to determine which interrupt request is to be serviced first. The LSR is loaded from fixed memory locations during the execution of an interrupt or when the CPU executes an EXIT or a TIN instruction. The TIN instruction is normally executed at the termination of an interrupt routine.

The interrupt service priority is controlled by the contents of the LSR so that the determination of which interrupts are to be allowed can be dynamically changed. The one exception is that interrupt 0 has top priority and cannot be locked out. This interrupt will be used to initiate program execution.

When an interrupt is received by the CPU, the instruction being executed will be completed and then an automatic sequence is entered in which the address of the next instruction to be executed, the contents of the storage limit register, the miscellaneous registers (P, D, OV, and C), and the current status of the LSR are stored in a bank of four memory locations (N0 to N3). Then the same registers will be loaded from the four memory locations N4 to N7. Figure 8 shows the location of the register values in the 8 word interrupt area. It is the system programmer's responsibility to define and load locations N4 to N7 (where N is the interrupt number) with the desired LSR value, miscellaneous register settings, storage limit setting and starting address of interrupt routine N.

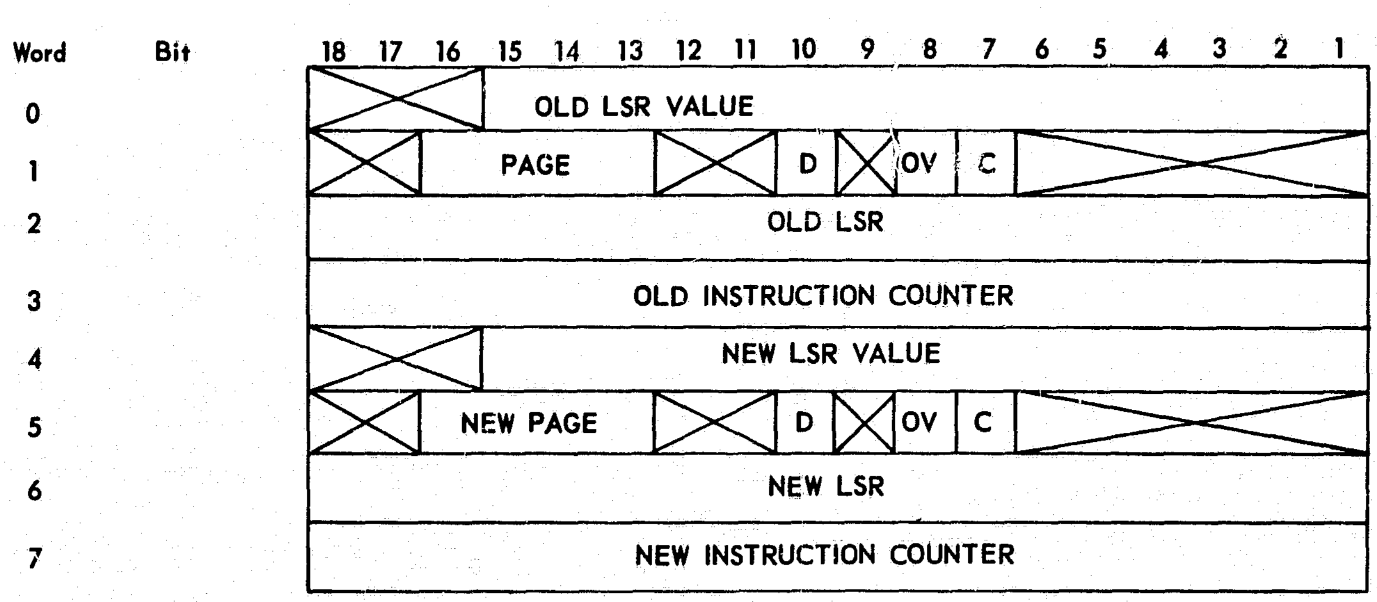



**Figure 8. Interrupt Locations**

At the conclusion of interrupt routine N there must be a TIN instruction which will result in returning control to the interrupted program and restoring the LSR, storage limit and miscellaneous registers. Of course, any addressable register which may be altered during the interrupt routine should be saved upon entering and restored upon leaving the interrupt routine.

Program Linkage

Within a single assembly, control is normally transferred by a BRU (LABEL) or BRC (LABEL) statement. When the name LABEL is defined to the assembler, the literal location (LABEL) will be filled with the correct relocatable address. At load time this relocatable address will be made absolute. Upon execution, the BRU or BRC instruction will cause the absolute address in this literal location to replace the contents of the program counter.

When transferring control between separately assembled programs the BRM LABEL statement is used in the calling program and LABEL must be externally defined in the called program's data area. Two contiguous locations must be allocated in the called program's data area in order to accomplish the transfer. The first must be given the external label and the second must contain the address of the called program's entry point. (See the INPUT and ADDER program assemblies given in Figure 2 for an example.)

When the set of programs is loaded, the undefined reference to the label in the BRM instruction is resolved to refer to the external definition in the called program's data area. Upon execution, the current value of the program counter will be saved in the location called LABEL and the next location after this will be used to supply a new value for the program counter.

Since the programmer may be unaware of the exact location the relocatable code will occupy at execution time, the simple linkage discussed above must be modified to take account of the possibility that the called program's data area may not occupy the same bank as the calling program's data. Thus the page register must be manipulated to allow formation of the correct address. An example of how this can be achieved appears below:

Calling Program (which is itself a called program)

```
$ (1)

START    STE     SAVEP     Save calling programs page register
          •                passed in EA
          •
          •
          •
          •
         LDE     CALLER    Load this program's page register

         LDA     CALLED    Load called program's page register

         LDP               Load page register with proper
                           setting for use by called program

         BRM     SUB2      Branch to called program

         LDP               Restore page register with proper
          •                setting for use by this program.
          •
          •
```

```
$ (0)

SUB1*     RES        1
          0+ START

SAVEP     RES        1
CALLER    0+ SUB 1              The loader will insert the absolute
                                address of the beginning of this
                                program's data area here.

CALLED    0+ SUB 2              The loader will insert the absolute
                                address of the beginning of the
                                called program's data area here.

          END
```

Called Program (which calls no other program)

```
$ (1)

START     STE        SAVEP      Save calling program's page
           •                    register setting
           •
           •
           •
          LDA        SAVEP      Restore called program's page
                                register setting

          BRU        SUB2       Return to caller.

$ (0)

SUB2*     RES        1          Execution of a BRM SUB2 in
                                calling program places the return
                                address here.

          0+ START              The next address placed in the
                                program counter will be the starting
                                address of this program's code
                                region.

SAVEP     RES        1

          END
```

Note how the calling program uses the form 0+LABEL to request the loader to supply the absolute address and therefore, the page register setting of both its own and the called program's data area. These settings are passed in the extended accumulator and accumulator respectively. This leaves the index register free for passing the address of an argument list. Also note that it is the calling program's responsibility to actually set the page register immediately before and after the branch to the called program.

It is the called program's responsibility to make sure the accumulator contains the correct address to allow the calling program's LDP to work properly. Of course, none of this is necessary if absolute code is written. However, the ease of program construction inherent in relocatable code is so great that these conventions for use of the accumulator, extended accumulator, and index register should be practically regarded as standard, where necessary.

Storage Limit Register

The AOP has an 18-bit storage limit register which is used to specify blocks of memory into which writing is permitted. Those instructions which require writing into memory, and therefore use the storage limit register, are STA, BRM, STE, STX, and STI. The register is broken into two 9-bit fields A and B where A = (B1-B9) and B = (B10-B18). Bi = the $i^{th}$ bit of the storage limit register numbered from the right. A and B represent upper and lower limits on the 9 high-order bits of a 16-bit effective address between which writing will be permitted. Stated symbolically, if C = (B8-B16) of the effective operand address for one of the five instructions listed above and A and B are as specified above, then if $B \leq C \leq A$, the write will be permitted, otherwise, write will not be permitted. Note that if A = B then one 128 word block is enabled whereas if $A = 777_8$ and B = 0 then all of memory is enabled.

## APPENDIX A

## AOP INSTRUCTION SET

In the following description of the AOP instructions the assembly language mnemonics recognized by the AOP Assembler are shown. Execution times are given in CPU cycles. One CPU cycle is nominally 1 microsecond.

**Load/Store instructions**

LDA | 2 0 | address |

The contents of storage at the effective address are placed in the accumulator.

Registers altered: Accumulator

Timing: 4 cycles

LDL | 4 0 | address |

The effective address is placed in the accumulator.

Registers altered: Accumulator

Timing: 4 cycles

LDI | 1 2 | address |

The 16 LSB's of memory at the effective address are treated as a new effective address. If bit 18 of the memory word is a one, the contents of the index register are added to the new effective address. Otherwise it remains unchanged. The contents of memory at the new effective address are placed in the accumulator.

Registers altered: Accumulator.

Timing: 6 cycles

| LDE | 5 | 2 | address |
|---|---|---|---|

The contents of storage at the effective address are placed in the extended accumulator.

Registers altered: Extended accumulator

Timing: 4 cycles

| LDX | 5 | 4 | address |
|---|---|---|---|

The contents of storage at the effective address are placed in the index register.

Registers altered: Index register

Timing: 4 cycles

| STA | 6 | 0 | address |
|---|---|---|---|

The contents of the accumulator are stored at the effective address unless that address is protected by the storage limit registers. If storage is protected, no write into memory occurs and interrupt 15 is generated.

Registers altered: None.

Timing: 6 cycles

**STI**

| 3 | 2 | address |
|---|---|---|

The 16 LSB's of memory at the effective address are treated as a new effective address. If bit 18 of the memory word is a one, the contents of the index register are added to the new effective address. Otherwise it remains unchanged. The contents of the accumulator are stored in memory at the new effective address unless that location is protected by the storage limit register. If storage is protected, no write into memory occurs and interrupt 15 is generated.

Registers altered: None.

Timing: 8 cycles

**STE**

| 1 | 0 | address |
|---|---|---|

The contents of the extended accumulator are stored at the effective address unless that address is protected by the storage limit registers. If storage is protected, no write into memory occurs and interrupt 15 is generated.

Registers altered: None.

Timing: 6 cycles

**STX**

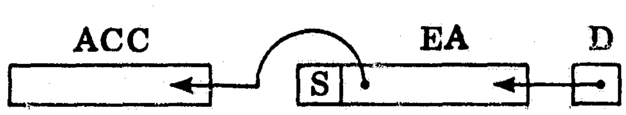
| | | |
|---|---|---|
| 7 | 4 | address |

The contents of the index register are stored at the effective address unless that address is protected by the storage limit registers. If storage is protected, no write into memory occurs and interrupt 15 is generated.

Registers altered: None.

Timing: 6 cycles

**LDD**




| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 3 |

The contents of the accumulator and extended accumulator are shifted left one position. The sign of the extended accumulator is not shifted and the vacated, low-order position of the extended accumulator is filled with the contents of the decision register. The overflow register is not altered.

Registers altered: Accumulator
Extended accumulator

Timing: 3 cycles

**LDP**

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 2 |

The contents of bits 13 through 16 of the accumulator are placed in the page register.

Registers altered: Page register

Timing: 3 cycles

## Arithmetic instructions

**NEG**

| 0 0 | 0 0 | 0 4 |
|---|---|---|

The contents of the accumulator are replaced by its two's complement. Negating all zeros yields a result of zero and sets the carry register to one. Negating the number that has zeros in all bit positions except the sign yields the same number as a result and sets both the carry register and the overflow register to one. Other than these two special cases, the carry register is reset to zero.

Registers altered: Accumulator
Carry register
Overflow register (conditionally)

Timing: 6 cycles

**ADC**

| 0 0 | 0 0 | 0 6 |
|---|---|---|

The content of the carry register is added to the contents of the accumulator and the sum is retained in the accumulator. If a carry occurs at the input of the 18th bit of the two's complement adder, then the carry register is set to one. Otherwise, the carry register is reset to zero. Overflow can occur and will cause the 18th bit of the sum to remain in the sign position and the overflow register to be set to one.

Registers altered: Accumulator
Carry register
Overflow register (conditionally)

Timing: 4 cycles

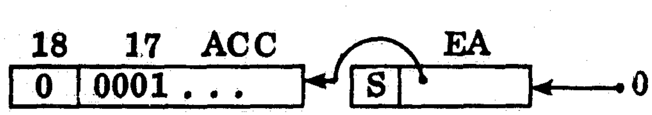
CMP | 00 | 00 | 10 |

The contents of the accumulator are replaced by the one's complement. The 18 bits of the results are computed independently with a one occurring in a bit position of the result only if the accumulator contained a zero in that position.

Registers altered: Accumulator

Timing: 6 cycles

NORM | 00 | 00 | 14 |




The contents of the accumulator and extended accumulator are shifted left until the 17th and 18th bits of the accumulator are different. The sign bit of the extended accumulator is not shifted. Bits leaving the 17th bit of the extended accumulator enter the low-order position of the accumulator. Zeros fill the positions vacated on the right. A count of the number of positions shifted is retained as a 6-bit positive number in the index register. If the contents of the accumulator and bit positions 1 through 17 of the extended accumulator are zero, then the index register is set to zero.

Registers altered: Accumulator
Extended accumulator
Index register

Timing: 4 cycles + 1 cycle per position shifted.

ADX | 0 2 | address |

The contents of storage at the effective address are added to the contents of the index register. The 18-bit result of the two's complement addition is retained in the index register.

Registers altered: Index register

Timing: 4 cycles

ADD | 0 4 | address |

The contents of storage at the effective address are added to the contents of the accumulator and the sum is retained in the accumulator. If a carry occurs at the input of the 18th stage of the two's complement adder, then the carry register is set to one. Otherwise, the carry register is reset to zero. Overflow can occur when two numbers of the same sign are added. Overflow causes the 18th bit of the sum to remain in the sign position and the overflow register to be set to one.

Register altered: Accumulator
Carry register
Overflow register (conditionally)

Timing: 4 cycles

SUB | 2 4 | address |

The contents of storage at the effective address are subtracted from the contents of the accumulator and the result retained in the accumulator. Subtraction is performed by adding the one's complement of the contents of storage to the accumulator with a carry forced into the low order stage of the adder. If a carry occurs out of the 17th position, the carry register is set to one. Otherwise it is reset to zero. Overflow can occur when two numbers of unlike sign are subtracted. Overflow causes the overflow register to be set to one and the 18th bit of the difference is retained in the sign position.

Registers altered: Accumulator
Carry register
Overflow register (conditionally)

Timing: 4 cycles

MUL

| 4 | 4 | address |
|---|---|---|

The contents of storage at the effective address are multiplied by the contents of the accumulator. The high-order 17 bits and sign of the product are retained in the accumulator. The low-order 17 bits and sign of the product are retained in the extended accumulator.

Registers altered: Accumulator
Extended accumulator

Timing: Average 32 cycles

DIV

| 6 | 4 | address |
|---|---|---|

The accumulator and extended accumulator form the dividend that is divided by the contents of storage at the effective address. The signed remainder is retained in the accumulator and the signed quotient is retained in the extended accumulator. The divisor and dividend must be positive and the dividend must be less than or equal to the divisor. Otherwise the results are unpredictable. The quotient and remainder are positive and the remainder has a magnitude less than the divisor.

Registers altered: Accumulator
Extended accumulator

Timing: 58 cycles

## Boolean Logic instructions

**ETR** | 3 0 | address |

The contents of storage at the effective address are logically ANDed with the contents of the accumulator. The result is retained in the accumulator. The 18 bits of the result are computed independently with a one occurring in a bit position of the result only if the accumulator and storage both contained a one in that bit position.

Registers altered: Accumulator

Timing: 4 cycles

**MRG** | 5 0 | address |

The contents of storage at the effective address are logically ORed with the contents of the accumulator. The result is retained in the accumulator. The 18 bits of the result are computed independently with a one occurring in a bit position of the result if either the accumulator or storage contained a one in that bit position.

Registers altered: Accumulator

Timing: 4 cycles

**EOR** | 7 | 0 | address |

The contents of storage at the effective address are exclusive ORed with the contents of the accumulator. The result is retained in the accumulator. The 18 bits of the result are computed independently with a one occurring in a bit position of the result if either the accumulator or storage, but not both, contain a one in that bit position.

Registers altered: Accumulator

Timing: 4 cycles

<u>I/O instructions</u>

**OPT** | 1 | 6 | address |

The contents of storage at the effective address plus one are sent over the channel (0-15) designated by the contents of storage at the effective address.

Registers altered: None

Timing: 6 cycles

**IPF** | 7 | 6 | address |

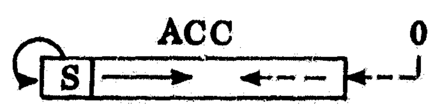
The contents of storage at the effective address plus one are replaced by a data word input over the channel (0-15) designated by the contents of storage at the effective address.

Registers altered: none

Timing: 6 cycles

Register manipulation instructions

SHF



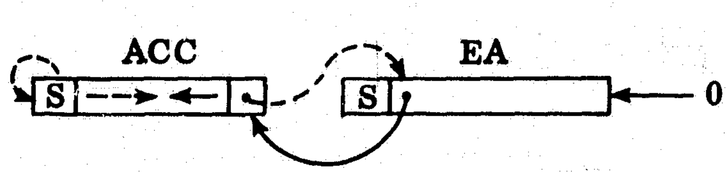

| | | |
|---|---|---|
| 1 | 4 | address |

The low-order 6 bits of the contents of storage at the effective address are used as a two's complement shift count. If the count is negative, then the accumulator is shifted right the number of positions specified by the count, with the contents of the accumulator sign replacing vacated positions on the left. If the count is positive, then the accumulator is shifted left the number of positions specified by the count with zeros filling vacated positions on the right. The overflow register is set to one if the sign bit of the accumulator is changed during the shift.

Registers altered: Accumulator
Overflow register (conditionally)

Timing: 5 cycles + 1 cycle per position shifted

DSH




| | | |
|---|---|---|
| 3 | 6 | address |

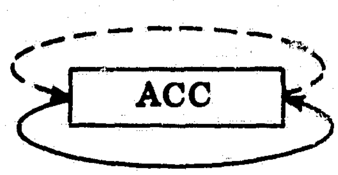
The low-order 6 bits of the contents of storage at the effective address are used as a two's complement shift count. The accumulator and the extended accumulator are shifted together. The extended accumulator is shown to the right of the accumulator and its sign bit is not shifted.

If the count is negative, then the accumulators are shifted right the number of positions specified by the count with the contents of the accumulator sign replacing vacated positions on the left. If the count is positive, then the accumulators are shifted left the number of positions specified by the count with zeros filling vacated positions on the right. The overflow register is set to one if the sign bit of the accumulator is changed during the shift.

Registers altered: Accumulator
Extended accumulator
Overflow register (conditionally)

Timing: 5 cycles + 1 cycle per position shifted

CYC



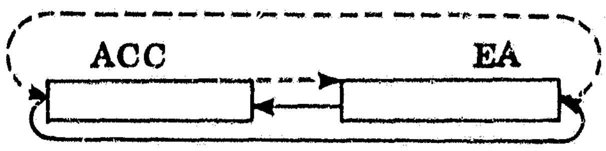

| 3 | 4 | address |
|---|---|---|

The low-order 6 bits of the contents of storage at the effective address are used as a two's complement shift count. If the count is negative, then the contents of the accumulator are shifted cyclically right the number of positions specified by the count, with bits leaving the low-order position entering the sign position. If the count is positive, then the contents of the accumulator are shifted left the number of positions specified by the count with bits leaving the sign position entering the low-order position.

Registers altered: Accumulator

Timing: 5 cycles + 1 cycle per position shifted

DCY




| 5 | 6 | address |
|---|---|---|

The low-order 6 bits of the contents of storage at the effective address are used as a two's complement shift count. If the count is negative, then the contents of the accumulator and extended accumulator are shifted cyclically right the number of positions specified by the count with bits leaving the low-order position of the extended accumulator entering the sign of the accumulator and bits leaving the low-order position of the accumulator entering the sign of the extended accumulator. If the count is positive, then the direction of the above process is reversed.

Registers altered: Accumulator
Extended accumulator

Timing: 5 cycles + 1 cycle per position shifted

ACX

| 0 | 0 | 0 | 0 | 2 | 5 |
|---|---|---|---|---|---|

The contents of the accumulator and index registers are interchanged.

Registers altered: Accumulator
Index register

Timing: 8 cycles

AEA

| 0 | 0 | 0 | 0 | 2 | 6 |
|---|---|---|---|---|---|

The contents of the accumulator and extended accumulator are interchanged.

Registers altered: Accumulator
Extended accumulator

Timing: 8 cycles

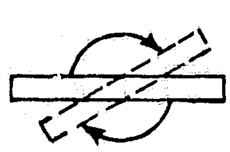
EAX

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 2 | 7 |

The contents of the extended accumulator and index register are interchanged.

Registers altered: Extended accumulator
Index register

Timing: 8 cycles

FLP  ACC

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 2 | 2 |

The contents of the accumulator are reversed. The $(19-n)^{th}$ and $n^{th}$ bits are exchanged for $n = 1, 2, \ldots 9$.

Registers altered: Accumulator

Timing: 3 cycles

## Control/branch instructions

HLT

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |

The processor stops indefinitely. An interrupt signal must be supplied from an external source to start the processor.

Registers altered: None

Timing: 3 cycles before an interrupt will be honored.

**NOP** | 0 0 | 0 0 | 0 2 |

No operation is performed other than the automatic incrementing of the instruction counter.

Registers altered: None

Timing: 3 cycles

**BRM** | 0 6 | address |

The contents of the instruction counter plus one are stored at the effective address unless that address is protected by the storage limit register. If storage is protected, no write into memory occurs and interrupt 15 is generated. The contents of one location greater than the effective address is placed in the instruction counter and execution proceeds from the new address specified by the instruction counter.

Registers altered: None.

Timing: 8 cycles

**BRU** | 6 2 | address |

The contents of storage at the effective address are placed in the instruction counter and execution proceeds from the new address specified by the instruction counter.

Registers altered: None

Timing: 4 cycles

BRC

| 4 | 2 | address |
|---|---|---|

If the contents of the decision register are zero, then the next sequential instruction is executed. If the contents of the decision register are one, then the contents of storage at the effective address are placed in the instruction counter and execution proceeds from the new address specified by the instruction counter. The decision register is reset to zero.

Registers altered: Decision register (always reset to zero)

Timing: 4 cycles

TIN

| 7 | 2 | address |
|---|---|---|

The contents of storage at the effective address is used as the starting address of a 4-word save area. This instruction restores the registers that were saved in these locations (i.e., by the occurrence of an interrupt). Upon completion, execution proceeds normally at the new value in the instruction counter.

Registers altered: Lockout status register
Storage limit register
Page register
Overflow register
Carry register
Decision register

Timing: 22 cycles

EXIT

| 0 | 0 | 0 | 0 | 1 | 6 |
|---|---|---|---|---|---|

This instruction initiates interrupt number 16 which uses locations octal 200 through 207. The status of registers is saved in 200-203 and these registers are loaded from locations 204-207. Upon completion, execution proceeds normally at the new value in the instruction counter.

Registers altered: Lockout status register
Storage limit register
Page register
Overflow register
Carry register
Decision register

Timing: 36 cycles

Test/set instructions

TOV

| 0 | 0 | 0 | 0 | 0 | 1 |
|---|---|---|---|---|---|

If the contents of the overflow register are one, then the contents of the decision register is set to one. Otherwise, it is unchanged. The overflow register is reset to zero.

Registers altered: Decision register (conditionally)
Overflow register

Timing: 3 cycles

TAP

| 0 | 0 | 0 | 0 | 0 | 3 |
|---|---|---|---|---|---|

If the sign position, bit 18, of the accumulator contains a zero, then the contents of the decision register are set to one. Otherwise, it is unchanged.

Registers altered: Decision register (conditionally)

Timing: 3 cycles

TOP

| 0 0 | 0 0 | 0 5 |
|---|---|---|

If the number of ones in the 18-bit accumulator is odd, then the contents of the decision register is set to one. Otherwise, it is unchanged.

Registers altered: Decision register (conditionally)

Timing: 22 cycles

ROV

| 0 0 | 0 0 | 0 7 |
|---|---|---|

The contents of the overflow register are set to zero.

Registers altered: Overflow register

Timing: 3 cycles

CPD

| 0 0 | 0 0 | 1 7 |
|---|---|---|

The contents of the decision register are complemented.

Registers altered: Decision register

Timing: 3 cycles

SIO

| 0 0 | 0 0 | 2 0 |
|---|---|---|

The contents of the interrupt override register are set to one.

Registers altered: Interrupt override register

Timing: 3 cycles

TAZ

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 2 | 1 |

If the contents of the accumulator is equal to zero, then the contents of the decision register are set to one. Otherwise, it is unchanged.

Registers altered: Decision register (conditionally)

Timing: 4 cycles

RED

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 2 | 3 |

The contents of the decision register are reset to zero.

Registers altered: Decision register

Timing: 3 cycles

RIO

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 2 | 4 |

The contents of the interrupt override register are reset to zero.

Registers altered: Interrupt override register

Timing: 3 cycles

TIX

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 |

The contents of the decision register is reset to zero. Then the contents of the index register are tested for zero. If it is non-zero, the contents of the decision register are set to one and the contents of the index register are incremented by one. If the contents of the index register are zero, the decision and index registers remain unchanged.

Registers altered: Decision register
Index register (conditionally)

Timing: 6 cycles

TIE

| 0 0 | 0 0 | 1 5 |
|---|---|---|

The contents of the decision register are reset to zero. Then the contents of the extended accumulator are tested for zero. If it is non-zero, the contents of the decision register are set to one and the contents of the extended accumulator is incremented by one. If the contents of the extended accumulator are zero, the decision register and extended accumulator remain unchanged.

Registers altered: Decision register
Extended accumulator (conditionally)

Timing: 6 cycles

TXLE

| 2 2 | address |
|---|---|

If the contents of the index register are less than or equal to the contents of storage at the effective address, then the contents of the decision register are set to one. Otherwise, it is unchanged.

Registers altered: Decision register (conditionally)

Timing: 4 cycles

TAL

| 2 6 | address |
|---|---|

If the contents of the accumulator are less than the contents of storage at the effective address, then the content of the decision register is set to one. Otherwise, it is unchanged.

Registers altered: Decision register (conditionally)

Timing: 4 cycles

TAE | 4 6 | address |

If the contents of the accumulator are equal to the contents of storage at the effective address then the contents of the decision register are set to one. Otherwise it is unchanged.

Registers altered: Decision register (conditionally)

Timing: 4 cycles

TAG | 6 6 | address |

If the contents of the accumulator are greater than the contents of storage at the effective address, the contents of the decision register are set to one. Otherwise, it is unchanged.

Registers altered: Decision register (conditionally)

Timing: 4 cycles

## AOP INSTRUCTION SET

### Ordered by Operation Code

| Op Code | Mnemonic | Cycle Time* | Description |
|---|---|---|---|
| 000000 | HLT | 3 | Halt |
| 000001 | TOV | 3 | Test overflow register |
| 000002 | NOP | 3 | No operation |
| 000003 | TAP | 3 | Test accumulator positive |
| 000004 | NEG | 6 | Two's complement accumulator |
| 000005 | TOP | 22 | Test accumulator for odd parity |
| 000006 | ADC | 4 | Add carry to accumulator |
| 000007 | ROV | 3 | Reset overflow register |
| 000010 | CMP | 6 | One's complement accumulator |
| 000011 | TIX | 6 | Test index for zero and increment |
| 000012 | LDP | 3 | Load page register from accumulator |
| 000013 | LDD | 3 | DSH left 1 bit then load decision register into LSB of EA |
| 000014 | NORM | 4** | Normalize accumulator and extension |
| 000015 | TIE | 6 | Test extension for zero and increment |
| 000016 | EXIT | 36 | Cause exit interrupt |
| 000017 | CPD | 3 | Complement decision register |
| 000020 | SIO | 3 | Set interrupt override |
| 000021 | TAZ | 4 | Test accumulator for zero |
| 000022 | FLP | 3 | Reverse accumulator |
| 000023 | RED | 3 | Reset decision register |
| 000024 | RIO | 3 | Reset interrupt override |
| 000025 | ACX | 8 | Exchange accumulator and index |
| 000026 | AEA | 8 | Exchange accumulator and extension |
| 000027 | EAX | 8 | Exchange extension and index |
| 02 | ADX | 4 | Add memory to index |
| 04 | ADD | 4 | Add memory to accumulator |
| 06 | BRM | 8 | Branch and mark place |
| 10 | STE | 6 | Store extension |
| 12 | LDI | 6 | Load accumulator indirect |
| 14 | SHF | 5** | Arithmetic shift accumulator |
| 16 | OPT | 6 | Output |
| 20 | LDA | 4 | Load accumulator |
| 22 | TXLE | 4 | Test index less than or equal to memory |
| 24 | SUB | 4 | Subtract memory from accumulator |
| 26 | TAL | 4 | Test accumulator less than memory |
| 30 | ETR | 4 | Logical AND accumulator with memory |
| 32 | STI | 8 | Store accumulator indirect |
| 34 | CYC | 5** | Cyclic shift accumulator |
| 36 | DSH | 5** | Arithmetic shift accumulator and extension |
| 40 | LDL | 4 | Load accumulator with effective address |
| 42 | BRC | 4 | Branch conditionally |
| 44 | MUL | 32** | Multiply accumulator by memory |
| 46 | TAE | 4 | Test accumulator equal to memory |
| 50 | MRG | 4 | Logical OR accumulator with memory |
| 52 | LDE | 4 | Load extension |
| 54 | LDX | 4 | Load index |
| 56 | DCY | 5** | Cyclic shift accumulator and extension |
| 60 | STA | 6 | Store accumulator |
| 62 | BRU | 4 | Branch unconditionally |
| 64 | DIV | 58 | Divide accumulator and extension by memory |
| 66 | TAG | 4 | Test accumulator greater than memory |
| 70 | EOR | 4 | Exclusive OR accumulator with memory |
| 72 | TIN | 22 | Restore status registers from memory |
| 74 | STX | 6 | Store index |
| 76 | IPF | 6 | Input |

*Add 2 cycles for indexing. One CPU cycle is nominally one microsecond.
**Add 1 cycle for each place shifted.
***Average

## Alphabetically Ordered by Mnemonics

| Mnemonic | Op Code | Cycle Time* | Description |
|---|---|---|---|
| ACX | 000025 | 6 | Exchange accumulator and index |
| ADC | 000026 | 4 | Add carry to accumulator |
| ADD | 04 | 4 | Add memory to accumulator |
| ADX | 02 | 4 | Add memory to index |
| AEA | 000026 | 8 | Exchange accumulator and extension |
| BRC | 42 | 4 | Branch conditionally |
| BRM | 06 | 8 | Branch and mark place |
| BRU | 62 | 4 | Branch unconditionally |
| CMP | 000010 | 6 | One's complement accumulator |
| CPD | 000017 | 3 | Complement decision register |
| CYC | 34 | 5** | Cyclic shift accumulator |
| DCY | 56 | 5** | Cyclic shift accumulator and extension |
| DIV | 64 | 58 | Divide accumulator and extension by memory |
| DSH | 36 | 5** | Arithmetic shift accumulator and extension |
| EAX | 000027 | 8 | Exchange extension and index |
| EOR | 70 | 4 | Exclusive OR accumulator with memory |
| ETR | 30 | 4 | Logical AND accumulator with memory |
| EXIT | 000016 | 36 | Cause Exit interrupt |
| FLP | 000022 | 3 | Reverse accumulator |
| HLT | 000000 | 3 | Halt |
| IPF | 76 | 6 | Input |
| LDA | 20 | 4 | Load accumulator |
| LDD | 000013 | 3 | DSH left 1 bit then load decision register into LSB of EA |
| LDE | 52 | 4 | Load extension |
| LDI | 12 | 6 | Load accumulator indirect |
| LDL | 40 | 4 | Load accumulator with effective address |
| LDP | 000012 | 3 | Load page register from accumulator |
| LDX | 54 | 4 | Load index |
| MRG | 50 | 4 | Logical OR accumulator with memory |
| MUL | 44 | 32*** | Multiply accumulator by memory |
| NEG | 000004 | 6 | Two's complement accumulator |
| NOP | 000002 | 3 | No operation |
| NORM | 000014 | 4** | Normalize accumulator and extension |
| OPT | 16 | 6 | Output |
| RED | 000023 | 3 | Reset decision register |
| RIO | 000024 | 3 | Reset interrupt override |
| ROV | 000007 | 3 | Reset overflow register |
| SHF | 14 | 5** | Arithmetic shift accumulator |
| SIO | 000020 | 3 | Set interrupt override |
| STA | 60 | 6 | Store accumulator |
| STE | 10 | 6 | Store extension |
| STI | 32 | 8 | Store accumulator indirect |
| STX | 74 | 6 | Store index |
| SUB | 24 | 4 | Subtract memory from accumulator |
| TAE | 46 | 4 | Test accumulator equal to memory |
| TAG | 66 | 4 | Test accumulator greater than memory |
| TAL | 26 | 4 | Test accumulator less than memory |
| TAP | 000003 | 3 | Test accumulator positive |
| TAZ | 000021 | 4 | Test accumulator for zero |
| TIE | 000015 | 6 | Test extension for zero and increment |
| TIN | 72 | 22 | Load status registers from memory |
| TIX | 000011 | 6 | Test index for zero and increment |
| TOP | 000005 | 22 | Test accumulator for odd parity |
| TOV | 000001 | 3 | Test overflow register |
| TXLE | 22 | 4 | Test index less than or equal to memory |

*Add 2 cycles for indexing. One CPU cycle is nominally one microsecond.
**Add 1 cycle for each place shifted.
***Average

## APPENDIX B

## AOP SUPPORT SOFTWARE DEMONSTRATION PROGRAM

The AOP Simulator is a complete simulation of all AOP features designed to permit rapid debugging in a complex environment. Extensive timing features are provided. To more fully explain the capabilities of the simulator, a special program was written to demonstrate some of these features. This program has the following characteristics:

(1) A 300 cycle clock interrupt begins at 100 cycles. A clock interrupt routine is set up to stop the program after 9 interrupts. Using the pseudo-console, the clock count can be reset and the run continued for an arbitrary number of clock interrupts.

(2) At 150 cycles the first channel 1 cycle steal I/O operation is performed. I/O requests will continue on this channel at the rate of one each 150 cycles. A block length of 3 words and a buffer of the same length are maintained.

(3) The first block length = 0 interrupt occurs at 450 cycles. The interrupt service routine re-establishes the block length and buffer address words. It then calls a worker whose only function is to sum the words input to the buffer.

The remainder of this appendix refers to the listing in this appendix.

On the first page of the listing, the executive routine containing the clock and cycle steal interrupt handling routines is assembled using the ; ASSEMBLE DEMO control card. First the interrupt locations for interrupts 1, 2, 15 and 16 are set up and then the cycle steal I/O control words for channel 1. Next comes the code for the clock interrupt handler followed by the Block Length (BL) = 0 interrupt handler, the background loop, and the storage protect interrupt handler. The channel 1 ASR control words, clock, and channel 1 buffer are set up in the program's relocatable data area.

The second page of the listing shows the assembly of a worker routine and its subroutine. The subroutine calling sequence conventions have been followed. In addition, an argument list address is passed in the index register.

Page 3 and 4 of the listing show the results of loading these three programs while a complete simulator run follows. Note that the pseudo-console was used to set the IOR to 0 and later, when the storage protect violation occurred, to reset the storage limit register and continue the run. A close examination of this example will reveal several AOP characteristics.

Refer to the sample output (page 9 of the listing) showing the simulation of a portion of the demonstration program. Line 1 shows a background job which consists of one instruction branching unconditionally to itself. The address of the instruction is in the IC column, the branch address in the [EAD] column.

At line 11 a clock interrupt has occurred (the I column contains the octal number of the interrupt currently being serviced). Before the clock routine can complete there is a cycle steal I/O request on channel 1 at 450 cycles simulated time, as indicated at line 13. Note the BRC instruction following line 3. This instruction normally completes in 4 cycles but subtracting completion times in the TIME column shows that 448-462 = 14 cycles were required because of cycle steal I/O interference. Also note in the ISR column of this line that a block length = 0 interrupt was generated by the cycle steal operation (ISR = 000002) and that further activity on this channel is suppressed (ASR = 000002). Since all interrupts are locked out (LSR = 177777) when the clock interrupt occurs, the block length = 0 interrupt is not honored until the clock interrupt routine terminates at line 17. The TIN instruction restores the AOP status prior to the clock interrupt. The block length = 0 interrupt routine gains control immediately at line 18. This routine takes action to enable further cycle steal I/O and transfers at line 23 to a worker routine which will attempt to add up the numbers (1, 2 and 3) input over channel 1. However, before the worker can fetch the first data item from the buffer, a cycle steal operation occurs (line 26) and replaces the contents of the first word of the buffer with a 4. Thus the value fetched by line 27 (see [EAD] column) is a 4 and this is placed in the accumulator instead of the desired 1. Such timing errors are made readily apparent to the programmer. As the program continues, an attempt is made to store the resultant sum at line 38. However, the storage limit register was not set up to allow the worker access to this location (SLR = 002040) so a storage protect interrupt (ISR = 100000) is generated causing the program to halt at line 39.

As the example shows, completeness of simulation and easy debugging are prime features of the simulator. Flexible facilities for dumping and tracing the progress of the simulation are provided. The example used here resulted from a complete octal trace. A pseudo-console facility is available which allows the user to fix up errors in his simulated program as they are detected, without the necessity of reloading or reassembling, just as though he were debugging at an AOP console. The error which caused the termination in the example was handled in this way and the simulation successfully continued. Any size AOP memory can be simulated.

```
$ ASSEMBLE DEMO

ASSEMBLER FOR THE ONBOARD PROCESSOR
                                                      $(0)
                                                       AORG 0.14
              0 000016     037037                      037037
              0 000017     000006                      0+CHPGM
                                                       AORG 0.20
              0 000024     177777                      0177777
                                                       AORG 0.22
              0 000026     001001                      001001
              0 000027     000000                      0+CLOCK
                                                       AORG 0.0177
              0 000177     000016                      0+SPROTECT
                                                       AORG 0.0206
              0 000206     002040                      02040
      U       0 000207     000000                      0+WORKER
                                                       AORG 0.07742
              0 007742     000003                     CH1BL 3
              0 007743     000003                     CH1ADR 0+BUF
                                                       RORG 0.0
                                                      $(1)
              1 000000 54 0 0000                      CLOCK LDX CLK
              1 000001     000011                      TIX
              1 000002 42 0 0014                       BRC (GOON)
              1 000003     000000                      HLT
              1 000004 74 0 0000                      GOON STX CLK
              1 000005 72 0 0006                       TIN (16)
              1 000006 20 0 0007                      CHPGM LDA (3)
              1 000007 60 0 7742                       STA CH1BL
              1 000010 20 0 0010                       LDA (BUF)
              1 000011 60 0 7743                       STA CH1ADR
              1 000012 16 0 0001                       OPT CH1ASR
              1 000013     000016                      EXIT
              1 000014 72 0 0011                       TIN (8)
              1 000015 62 0 0013                      TIGHT BRU TIGHT
              1 000016     000000                     SPROTECT HLT
              1 000017 72 0 0012                       TIN (0170)
                                                      $(0)
              0 000000     777767                     CLK -9
              0 000001     000002                     CH1ASR 2
              0 000002     000002                      2
              0 000003                                BUF* RES 3
                                                       END
              0            000006 000020
              0            000007 000003
              0            000010 000003
              0            000011 000010
              0            000012 000170
              0            000013 000015
              0            000014 000004

       END OF LISTING.     1 LINES FLAGGED.
```

```
$ ASSEMBLE WORKER

ASSEMBLER FOR THE ONBOARD PROCESSOR
                                                      $(0)
      U      0 000000       000000                    B 0+BUF
             0 000001       000002                    ARGLST 0+SUM
             0 000002                                 SUM RES 1
      U      0 000003       000000                    SUB 0+WORKERS
             0 000004       000000                    CALLER 0+B
                                                      $(1)
             1 000000 32 0 0005                       WORKER* LDE (-1)
             1 000001 54 0 0000                        LDX B
             1 000002 21 1 0000                        LDA. 0
             1 000003       000011                    LOOP TIX
             1 000004 05 1 0000                        ADD. 0
             1 000005       000015                     TIE
             1 000006 42 0 0010                        BRC (LOOP)
             1 000007 60 0 0002                        STA SUM
             1 000010 52 0 0004                        LDE CALLER
             1 000011 54 0 0006                        LDX (ARGLST)
             1 000012 20 0 0003                        LDA SUB
             1 000013       000012                     LDP
      U      1 000014 06 0 0000                        BRM  WORKERS
             1 000015       000012                     LDP
             1 000016 72 0 0007                        TIN (0200)
                                                       END
             0              000005 777777
             0              000006 000001
             0              000007 000200
             0              000010 000003

        END OF LISTING.    3 LINES FLAGGED.
$ ASSEMBLE WORKERS

ASSEMBLER FOR THE ONBOARD PROCESSOR
                                                      $(0)
             0 000000                                 WORKERS* RES 1
             0 000001       000000                     0+START
             0 000002                                 SAVEP RES 1
             0 000003                                 ARG RES 1
             0 000004                                 SUMW RES 1
                                                      $(1)
             1 000000 10 0 0002                       START STE SAVEP
             1 000001 21 1 0000                        LDA. 0
             1 000002 60 0 0003                        STA ARG
             1 000003 12 0 0003                        LDI ARG
             1 000004 60 0 0004                        STA SUMW
             1 000005 20 0 0002                        LDA SAVEP
             1 000006 62 0 0000                        BRU  WORKERS
                                                       END

        END OF LISTING.    0 LINES FLAGGED.
```

```
$ LOAD DEMO

LOADER FOR ON-BOARD PROCESSOR SOFTWARE


PREAMBLE VALUES FOR DEMO
   DATA LENGTH                          13
   CODE LENGTH                          16
   PRESET LOCATIONS                     13
   LITERALS                              5
   INDIRECT ADDRESSES                    2
   EXTERNAL DEFINITIONS                  1
   UNDEFINED SYMBOLS                     1
   NOUNS                                10


EXTERNAL DEFINITIONS
   BUF


PREAMBLE VALUES FOR WORKER
   DATA LENGTH                           9
   CODE LENGTH                          15
   PRESET LOCATIONS                      4
   LITERALS                              3
   INDIRECT ADDRESSES                    1
   EXTERNAL DEFINITIONS                  1
   UNDEFINED SYMBOLS                     2
   NOUNS                                 7


EXTERNAL DEFINITIONS
   WORKER


PREAMBLE VALUES FOR WORKERS
   DATA LENGTH                           5
   CODE LENGTH                           7
   PRESET LOCATIONS                      1
   LITERALS                              0
   INDIRECT ADDRESSES                    0
   EXTERNAL DEFINITIONS                  1
   UNDEFINED SYMBOLS                     0
   NOUNS                                 5


EXTERNAL DEFINITIONS
   WORKERS
```

```
CORE LIMITS
       DATA 000210-000601 / CODE 004000-004045
       STARTING ADDRESS 004000
       MINIMUM ABSOLUTE ADDRESS 000016
       MAXIMUM ABSOLUTE ADDRESS 007742


CORE ALLOCATION

       DEMO
            DATA 000210-000224
            CODE 004000-004017
       WORKER
            DATA 000400-000410
            CODE 004020-004036
       WORKERS
            DATA 000600-000601
            CODE 004037-004045

END OF ALLOCATION


  ; SIMULATE DEMO

  ; TRACE OCTALLY

  ; DUMP OCTALLY AT HALT

  ; INTERRUPT 2 EVERY 300 MICROSECONDS STARTING AT 100 MICROSECONDS

  ; MAXIMUM TIME IS 4 MILLISECONDS

  ; INITIATE CHANNEL 1 AT 150 MICROSECONDS

  ; DATA FOR INPUT DEVICE 1

   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
   30 31 32 33 34 35 36 37 38 39 40 $
  ; START AT TIGHT

  ON-BOARD PROCESSOR SIMULATOR
```

| IC | INST | OPERAND | [EAD] | AC | EA | ICDG | [IC] | P | INDEX | X | EAD | SLR | TIME | I | LSR | ISR | ASR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 1000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 4 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 8 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 12 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 16 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 20 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 24 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 28 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 32 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 36 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 40 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 44 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 48 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 52 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 56 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 60 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 64 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 68 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 72 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 76 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 80 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 84 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 88 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 92 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 96 | | 000000 | 000000 | 000000 |
| 004015<br>TIGHT | BRU | TIGHT | 004015 | 000000 | 000000 | 0000 | 620223 | 00 | 000000 | 0 | 000223 | 000000 | 100 | | 000000 | 000000 | 000000 |

```
    IC  INST   OPERAND     [EAD]  AC      EA     ICDO  [IC]   P INDEX  X  EAD   SLR      TIME  I   LSR    ISR    ASR

  004000 LDX  CLK         777767 000000 000000 0000  540210 00 777767 0 000210 001001    140 02 177777 000003 000000
  004001 FIX              000000 000000 000000 0010  000011 00 777770 0 000000 001001    146 02 177777 000003 000000
  004002 BRC  GOON        004004 000000 000000 0000  420224 00 777770 0 000224 001001    150 02 177777 000003 000000
GOON
CS IO ON  1 TO  00000213 AT     150
  004004 STX  CLK         777770 000000 000000 0000  740210 00 777770 0 000210 001001    166 02 177777 000003 000000
  004005 TIN              000020 000000 000000 0000  720216 00 777770 0 000216 000000    186 02 000000 000003 000000
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    192    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    196    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    200    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    204    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    208    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    212    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    216    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    220    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    224    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    228    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    232    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    236    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    240    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    244    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    248    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    252    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    256    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    260    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    264    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    268    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    272    000000 000003 000000
TIGHT
  004015 BRU  TIGHT       004015 000000 000000 0000  620223 00 777770 0 000223 000000    276    000000 000003 000000
```

```
   IC  INST    OPERAND     (EAD)  AC      EA    ICD0   (IC)   P INDEX  X  EAD    SLR      TIME  I  LSR    ISR    ASR

TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    280     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    284     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    288     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    292     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    296     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    300     000000 000000 000000
TIGHT
CS I0 ON  1 TO  00000214 AT     300
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    314     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    318     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    322     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    326     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    330     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    334     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    338     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    342     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    346     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    350     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    354     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    358     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    362     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    366     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    370     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    374     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    378     000000 000003 000000
TIGHT
  004015 BRU  TIGHT        004015 000000 000000  0000  620223 00 777770 0 000223 000000    382     000000 000003 000000
TIGHT
```

```
    IC  INST    OPERAND      [EAD]   XC      EA     ICD0   [IC]   P INDEX  X  EAD    SLR      TIME  I   LSR    ISR    ASR

  004015 BRU  TIGHT         004015 000000 000000  0000  620223 00 777770 0 000223 000000     386      000000 000003 000000
TIGHT
  004015 BRU  TIGHT         004015 000000 000000  0000  620223 00 777770 0 000223 000000     390      000000 000003 000000
TIGHT
  004015 BRU  TIGHT         004015 000000 000000  0000  620223 00 777770 0 000223 000000     394      000000 000003 000000
TIGHT
  004015 BRU  TIGHT         004015 000000 000000  0000  620223 00 777770 0 000223 000000     398      000000 000003 000000
TIGHT
  004015 BRU  TIGHT         004015 000000 000000  0000  620223 00 777770 0 000223 000000     402      000000 000003 000000
TIGHT
  004000 LDX  CLK           777770 000000 000000  0000  540210 00 777770 0 000210 001001     442 02 177777 000003 000000
  004001 TIX                000000 000000 000000  0010  000011 00 777771 0 000000 001001     448 02 177777 000003 000000
CS I0 ON  1 TO  00000215 AT      450
  004002 BRC  GOON          004004 000000 000000  0000  420224 00 777771 0 000224 001001     462 02 177777 000002 000002
GOON
  004004 STX  CLK           777771 000000 000000  0000  740210 00 777771 0 000210 001001     468 02 177777 000002 000002
  004005 TIN                000020 000000 000000  0000  720216 00 777771 0 000216 000000     490 02 000000 000002 000002
  004006 LDA                000003 000003 000000  0000  200217 00 777771 0 000217 037037     530 01 000000 000003 000002
  004007 STA  CHIBL         000003 000003 000000  0000  607742 00 777771 0 007742 037037     536 01 000000 000003 000002
  004010 LDA                000213 000213 000000  0000  200220 00 777771 0 000220 037037     540 01 000000 000003 000002
  004011 STA  CHIADR        000213 000213 000000  0000  607743 00 777771 0 007743 037037     546 01 000000 000003 000002
  004012 OPT  CHIASR        000002 000213 000000  0000  160211 00 777771 0 000211 037037     552 01 000000 000003 000000
  004013 EXIT               000000 000213 000000  0000  000016 00 777771 0 000000 002040     588 01 000000 000003 000000
  004020 LDE  CALLER        777777 000213 777777  0000  520405 00 777771 0 000405 002040     596 01 000000 000003 000000
  004021 LDX  B             000213 000213 777777  0000  540400 00 000213 0 000400 002040     600 01 000000 000003 000000
CS I0 ON  1 TO  00000213 AT      600
  004022 LDA                000004 000004 777777  0000  210000 00 000213 1 000213 002040     616 01 000000 000003 000000
  004023 TIX                000000 000004 777777  0010  000011 00 000214 0 000000 002040     622 01 000000 000003 000000
  004024 ADD                000002 000006 777777  0010  050000 00 000214 1 000214 002040     628 01 000000 000003 000000
  004025 TIE                000000 000006 000000  0010  000015 00 000214 0 000000 002040     634 01 000000 000003 000000
  004026 BRC  LOOP          004023 000006 000000  0000  420410 00 000214 0 000410 002040     638 01 000000 000003 000000
LOOP
  004023 TIX                000000 000006 000000  0010  000011 00 000215 0 000000 002040     644 01 000000 000003 000000
  004024 ADD                000003 000011 000000  0010  050000 00 000215 1 000215 002040     650 01 000000 000003 000000
  004025 TIE                000000 000011 000000  0000  000015 00 000215 0 000000 002040     656 01 000000 000003 000000
  004026 BRC  LOOP          004023 000011 000000  0000  420410 00 000215 0 000410 002040     660 01 000000 000003 000000
STORAGE PROTECT VIOLATION AT 004027  000402
  004027 STA  SUM           000000 000011 000000  0000  600402 00 000215 0 000402 002040     666 01 000000 100003 000000
  004016 HLT                000000 000011 000000  0000  000000 00 000215 0 000000 000000     705 17 000000 000003 000000
```

INTERRUPT LOCATIONS

```
000000  000000 000000 000000 060000 000000 000000 000000 000000   000010  000000 000000 000000 004015 000000 000000 037037 004006
000020  000000 000000 000000 004015 177777 000000 001001 004000   000030  000000 000000 000000 000000 000000 000000 000000 000000
000160  000000 000000 000000 000000 000000 000000 000000 000000   000170  000000 000000 002040 004030 000000 000000 000000 004016
000200  000000 000000 037037 004014 000000 000000 002040 004020   000210  777771 000002 000002 000004 000002 000003 000020 000003
```

DEMO     [DATA] 000210-000224

```
000200  000000 000000 037037 004014 000000 000000 002040 004020   000210  777771 000002 000002 000004 000002 000003 000020 000003
000220  000213 000010 000170 004015 004004 000000 000000 000000   000230  000000 000000 000000 000000 000000 000000 000000 000000
```

DEMO     [CODE] 004000-004017

```
004000  540210 000011 420224 000000 740210 720216 200217 607742   004010  200220 607743 160211 000016 720221 620223 000000 720222
```

WORKER     [DATA] 000400-000410

```
000400  000213 000402 000000 000600 000400 777777 000401 000200   000410  004023 000000 000000 000000 000000 000000 000000 000000
```

WORKER     [CODE] 004020-004036

```
004020  520405 540400 210000 000011 050000 000015 420410 600402   004030  520404 540406 200403 000012 060600 000012 720407 100602
```

WORKERS     [DATA] 000600-000601

```
000600  000000 004037 000000 000000 000000 000000 000000 000000   000610  000000 000000 000000 000000 000000 000000 000000 000000
```

WORKERS     [CODE] 004037-004045

```
004020  520405 540400 210000 000011 050000 000015 420410 600402   004030  520404 540406 200403 000012 060600 000012 720407 100602
004040  210000 600603 120603 600604 200602 620600 000000 000000   004050  000000 000000 000000 000000 000000 000000 000000 000000
```

REMAINING CORE IN DATA REGION

```
000600  000000 004037 000000 000000 000000 000000 000000 000000   000610  000000 000000 000000 000000 000000 000000 000000 000000
```

REMAINING CORE IN CODE REGION

```
004040  210000 600603 120603 600604 200602 620600 000000 000000   004050  000000 000000 000000 000000 000000 000000 000000 000000
007740  000000 000000 000002 000214 000000 000000 000000 000000   007750  000000 000000 000000 000000 000000 000000 000000 000000
```

RUNNING TIME = .001 SECS
INSTRUCTIØNS EXECUTED = 105

| INST | CØUNT | INST | CØUNT | INST | CØUNT |
|---|---|---|---|---|---|
| ADX | 0 | ADD | 2 | BRM | 0 |
| STE | 0 | LDI | 0 | SHF | 0 |
| ØPT | 1 | LDA | 3 | TXLE | 0 |
| SUB | 0 | TAL | 0 | ETR | 0 |
| STI | 0 | CYC | 0 | DSH | 0 |
| LDL | 0 | BRC | 4 | MUL | 0 |
| TAE | 0 | MRG | C | LDE | 1 |
| LDX | 3 | DCY | 0 | STA | 3 |
| BRU | 76 | DIV | 0 | TAG | 0 |
| EØR | 0 | TIN | 2 | STX | 2 |
| IPF | 0 | HLT | 1 | TØV | 0 |
| NØP | C | TAP | 0 | NEG | 0 |
| TOP | 0 | ADC | 0 | RØV | 0 |
| CMP | 0 | TIX | 4 | LDP | 0 |
| LDD | 0 | NØRM | 0 | TIE | 2 |
| EXIT | 1 | CPD | 0 | SIØ | 0 |
| TAZ | 0 | FLP | 0 | RED | 0 |
| RIØ | 0 | ACX | 0 | 4EA | 0 |
| EAX | 0 | | | | |

```
   IC   INST    OPERAND     [EAD]   AC      EA     ICDO   [IC]   P INDEX  X  EAD     SLR      TIME  I   LSR    ISR    ASR

  004000 LDX  CLK        777771 000011 000000  0000  540210 00 777771 0 000210 001001    745 02 177777 000000 000000
  004001 TIX             000000 000011 000000  0010  000011 00 777772 0 000000 001001    751 02 177777 000000 000000
CS IO ON  1 TO  00000214 AT     750
  004002 BRC  GOON       004004 000011 000000  0000  420224 00 777772 0 000224 001001    764 02 177777 000000 000000
GOON
  004004 STX  CLK        777772 000011 000000  0000  740210 00 777772 0 000210 001001    770 02 177777 000000 000000
  004005 TIN             000020 000011 000000  0000  720216 00 777772 0 000216 000000    792 02 000000 000000 000000
  004017 TIN             000170 000011 000000  0000  720222 00 777772 0 000222 040002    814 17 000000 000000 000000
  004030 LDE  CALLER     000400 000011 000400  0000  520404 00 777772 0 000404 040002    818 01 000000 000000 000000
  004031 LDX             000401 000011 000400  0000  540406 00 000401 0 000406 040002    822 01 000000 000000 000000
  004032 LDA  SUB        000600 000600 000400  0000  200403 00 000401 0 000403 040002    826 01 000000 000000 000000
  004033 LDP             000000 000600 000400  0000  000012 00 000401 0 000000 040002    829 01 000000 000000 000000
  004034 BRM  WORKERS    004035 000600 000400  0000  060600 00 000401 0 000600 040002    837 01 000000 000000 000000
WORKERS
  004037 STE  SAVEP      000400 000600 000400  0000  100602 00 000401 0 000602 040002    843 01 000000 000000 000000
  004040 LDA             000402 000402 000400  0000  210000 00 000401 1 000401 040002    849 01 000000 000000 000000
  004041 STA  ARG        000402 000402 000400  0000  600603 00 000401 0 000603 040002    855 01 000000 000000 000000
  004042 LDI  ARG        000402 000000 000400  0000  120603 00 000401 0 000603 040002    861 01 000000 000000 000000
  004043 STA  SUMW       000000 000000 000400  0000  600604 00 000401 0 000604 040002    867 01 000000 000000 000000
  004044 LDA  SAVEP      000400 000400 000400  0000  200602 00 000401 0 000602 040002    871 01 000000 000000 000000
  004045 BRU  WORKERS    004035 000400 000400  0000  620600 00 000401 0 000600 040002    875 01 000000 000000 000000
WORKERS
  004035 LDP             000000 000400 000400  0000  000012 00 000401 0 000000 040002    878 01 000000 000000 000000
  004036 TIN             000200 000400 000400  0000  720407 00 000401 0 000407 037037    900 01 000000 000000 000000
CS IO ON  1 TO  00000215 AT     900
  004014 TIN             000010 000400 000400  0000  720221 00 000401 0 000221 000000    932 01 000000 000002 000002
  004006 LDA             000003 000003 000400  0000  200217 00 000401 0 000217 037037    972 01 000000 000000 000002
  004007 STA  CH1BL      000003 000003 000400  0000  607742 00 000401 0 007742 037037    978 01 000000 000000 000002
  004010 LDA             000213 000213 000400  0000  200220 00 000401 0 000220 037037    982 01 000000 000000 000002
  004011 STA  CH1ADR     000213 000213 000400  0000  607743 00 000401 0 007743 037037    988 01 000000 000000 000002
  004012 OPT  CH1ASR     000002 000213 000400  0000  160211 00 000401 0 000211 037037    994 01 000000 000000 000000
  004013 EXIT            000060 000213 000400  0000  000016 00 000401 0 000000 040002   1030 01 000000 000000 000000
  004020 LDE  CALLER     777777 000213 777777  0000  520405 00 000401 0 000405 040002   1038 01 000000 000000 000000
CS IO ON  1 TO  00000213 AT     1050
  004000 LDX  CLK        777772 000213 777777  0000  540210 00 777772 0 000210 001001   1088 02 177777 000000 000000
  004001 TIX             000000 000213 777777  0010  000011 00 777773 0 000000 001001   1094 02 177777 000000 000000
  004002 BRC  GOON       004004 000213 777777  0000  420224 00 777773 0 000224 001001   1098 02 177777 000000 000000
GOON
  004004 STX  CLK        777773 000213 777777  0000  740210 00 777773 0 000210 001001   1104 02 177777 000000 000000
  004005 TIN             000020 000213 777777  0000  720216 00 777773 0 000216 040002   1126 02 000000 000000 000000
  004021 LDX  B          000213 000213 777777  0000  540400 00 000213 0 000400 040002   1130 01 000000 000000 000000
  004022 LDA             000007 000007 777777  0000  210000 00 000213 1 000213 040002   1136 01 000000 000000 000000
  004023 TIX             000000 000007 777777  0010  000011 00 000214 0 000000 040002   1142 01 000000 000000 000000
  004024 ADD             000005 000014 777777  0010  050000 00 000214 1 000214 040002   1148 01 000000 000000 000000
  004025 TIE             000000 000014 000000  0010  000015 00 000214 0 000000 040002   1154 01 000000 000000 000000
  004026 BRC  LOOP       004023 000014 000000  0000  420410 00 000214 0 000410 040002   1158 01 000000 000000 000000
LOOP
  004023 TIX             000000 000014 000000  0010  000011 00 000215 0 000000 040002   1164 01 000000 000000 000000
  004024 ADD             000006 000022 000000  0010  050000 00 000215 1 000215 040002   1170 01 000000 000000 000000
  004025 TIE             000000 000022 000000  0000  000015 00 000215 0 000000 040002   1176 01 000000 000000 000000
```

```
 IC    INST   OPERAND     (EAD)  AC     EA     ICDO  (IC)   P  INDEX  X  EAD    SLR     TIME I   LSR    ISR    ASR

 004026 BRC   LOOP        004023 000022 000000 0000  420410 00 000215 0 000410 040002  1180 01 000000 000000 000000
 004027 STA   SUM         000022 000022 000000 0000  600402 00 000215 0 000402 040002  1186 01 000000 000000 000000
 004030 LDE   CALLER      000400 000022 000400 0000  520404 00 000215 0 000404 040002  1190 01 000000 000000 000000
 004031 LDX               000401 000022 000400 0000  540406 00 000401 0 000406 040002  1194 01 000000 000000 000000
 004032 LDA   SUB         000600 000600 000400 0000  200403 00 000401 0 000403 040002  1198 01 000000 000000 000000
 004033 LDP               000000 000600 000400 0000  000012 00 000401 0 000000 040002  1201 01 000000 000000 000000
CS IO ON  1 TO  00000214 AT     1200
 004034 BRM   WORKERS     004035 000600 000400 0000  060600 00 000401 0 000600 040002  1218 01 000000 000000 000000
WORKERS
 004037 STE   SAVEP       000400 000600 000400 0000  100602 00 000401 0 000602 040002  1224 01 000000 000000 000000
 004040 LDA               000402 000402 000400 0000  210000 00 000401 1 000401 040002  1230 01 000000 000000 000000
 004041 STA   ARG         000402 000402 000400 0000  600603 00 000401 0 000603 040002  1236 01 000000 000000 000000
 004042 LDI   ARG         000402 000022 000400 0000  120603 00 000401 0 000603 040002  1242 01 000000 000000 000000
 004043 STA   SUMW        000022 000022 000400 0000  600604 00 000401 0 000604 040002  1248 01 000000 000000 000000
 004044 LDA   SAVEP       000400 000400 000400 0000  200602 00 000401 0 000602 040002  1252 01 000000 000000 000000
 004045 BRU   WORKERS     004035 000400 000400 0000  620600 00 000401 0 000600 040002  1256 01 000000 000000 000000
WORKERS
 004035 LDP               000000 000400 000400 0000  000012 00 000401 0 000000 040002  1259 01 000000 000000 000000
 004036 TIN               000200 000400 000400 0000  720407 00 000401 0 000407 037037  1281 01 000000 000000 000000
 004014 TIN               000010 000400 000400 0000  720221 00 000401 0 000221 000000  1303 01 000000 000000 000000
 004000 LDX   CLK         777773 000400 000400 0000  540210 00 777773 0 000210 001001  1343 02 177777 000000 000000
 004001 TIX               000000 000400 000400 0010  000011 00 777774 0 000000 001001  1349 02 177777 000000 000000
CS IO ON  1 TO  00000215 AT     1351
 004002 BRC   GOON        004004 000400 000400 0000  420224 00 777774 0 000224 001001  1363 02 177777 000002 000002
GOON
 004004 STX   CLK         777774 000400 000400 0000  740210 00 777774 0 000210 001001  1369 02 177777 000002 000002
 004005 TIN               000020 000400 000400 0000  720216 00 777774 0 000216 000000  1391 02 000000 000002 000002
 004006 LDA               000003 000003 000400 0000  200217 00 777774 0 000217 037037  1431 01 000000 000000 000002
 004007 STA   CH1BL       000003 000003 000400 0000  607742 00 777774 0 007742 037037  1437 01 000000 000000 000002
 004010 LDA               000213 000213 000400 0000  200220 00 777774 0 000220 037037  1441 01 000000 000000 000002
 004011 STA   CH1ADR      000213 000213 000400 0000  607743 00 777774 0 007743 037037  1447 01 000000 000000 000002
 004012 OPT   CH1ASR      000002 000213 000400 0000  160211 00 777774 0 000211 037037  1453 01 000000 000000 000000
 004013 EXIT              000000 000213 000400 0000  000016 00 777774 0 000000 040002  1489 01 000000 000000 000000
 004020 LDE   CALLER      777777 000213 777777 0000  520405 00 777774 0 000405 040002  1497 01 000000 000000 000000
 004021 LDX   B           000213 000213 777777 0000  540400 00 000213 0 000400 040002  1501 01 000000 000000 000000
CS IO ON  1 TO  00000213 AT     1501
 004022 LDA               000012 000012 777777 0000  210000 00 000213 1 000213 040002  1517 01 000000 000000 000000
 004023 TIX               000000 000012 777777 0010  000011 00 000214 0 000000 040002  1523 01 000000 000000 000000
 004024 ADD               000010 000022 777777 0010  050000 00 000214 1 000214 040002  1529 01 000000 000000 000000
 004025 TIE               000000 000022 000000 0010  000015 00 000214 0 000000 040002  1535 01 000000 000000 000000
 004026 BRC   LOOP        004023 000022 000000 0000  420410 00 000214 0 000410 040002  1539 01 000000 000000 000000
LOOP
 004023 TIX               000000 000022 000000 0010  000011 00 000215 0 000000 040002  1545 01 000000 000000 000000
 004024 ADD               000011 000033 000000 0010  050000 00 000215 1 000215 040002  1551 01 000000 000000 000000
 004025 TIE               000000 000033 000000 0000  000015 00 000215 0 000000 040002  1557 01 000000 000000 000000
 004026 BRC   LOOP        004023 000033 000000 0000  420410 00 000215 0 000410 040002  1561 01 000000 000000 000000
 004027 STA   SUM         000033 000033 000000 0000  600402 00 000215 0 000402 040002  1567 01 000000 000000 000000
 004030 LDE   CALLER      000400 000033 000400 0000  520404 00 000215 0 000404 040002  1571 01 000000 000000 000000
 004031 LDX               000401 000033 000400 0000  540406 00 000401 0 000406 040002  1575 01 000000 000000 000000
 004032 LDA   SUB         000600 000600 000400 0000  200403 00 000401 0 000403 040002  1579 01 000000 000000 000000
```

```
   IC   INST   OPERAND      [EAD]   AC     EA    ICD0  [IC]   P INDEX  X  EAD    SLR      TIME  I  LSR    ISR    ASR

  004033 LDP                000000 000600 000400 0000  000012 00 000401 0 000000 040002   1582 01 000000 000003 000000
  004034 BRM   WORKERS      004035 000600 000400 0000  060600 00 000401 0 000600 040002   1590 01 000000 000003 000000
WORKERS
  004037 STE   SAVEP        000400 000600 000400 0000  100602 00 000401 0 000602 040002   1596 01 000000 000003 000000
  004040 LDA                000402 000402 000400 0000  210000 00 000401 1 000401 040002   1602 01 000000 000003 000000
  004000 LDX   CLK          777774 000402 000400 0000  540210 00 777774 0 000210 001001   1642 02 177777 000003 000000
  004001 TIX                000000 000402 000400 0010  000011 00 777775 0 000000 001001   1648 02 177777 000003 000000
CS I0 ON  1 TO  00000214 AT     1650
  004002 BRC   GOON         004004 000402 000400 0000  420224 00 777775 0 000224 001001   1662 02 177777 000003 000000
GOON
  004004 STX   CLK          777775 000402 000400 0000  740210 00 777775 0 000210 001001   1668 02 177777 000003 000000
  004005 TIN                000020 000402 000400 0000  720216 00 777775 0 000216 040002   1690 02 000000 000003 000000
  004041 STA   ARG          000402 000402 000400 0000  600603 00 777775 0 000603 040002   1696 01 000000 000003 000000
  004042 LDI   ARG          000402 000033 000400 0000  120603 00 777775 0 000603 040002   1702 01 000000 000003 000000
  004043 STA   SUMW         000033 000033 000400 0000  600604 00 777775 0 000604 040002   1708 01 000000 000003 000000
  004044 LDA   SAVEP        000400 000400 000400 0000  200602 00 777775 0 000602 040002   1712 01 000000 000003 000000
  004045 BRU   WORKERS      004035 000400 000400 0000  620600 00 777775 0 000600 040002   1716 01 000000 000003 000000
WORKERS
  004035 LDP                000000 000400 000400 0000  000012 00 777775 0 000000 040002   1719 01 000000 000003 000000
  004036 TIN                000200 000400 000400 0000  720407 00 777775 0 000407 037037   1741 01 000000 000003 000000
  004014 TIN                000010 000400 000400 0000  720221 00 777775 0 000221 000000   1763 01 000000 000003 000000
  004015 BRU   TIGHT        004015 000400 000400 0000  620223 00 777775 0 000223 000000   1767    000000 000003 000000
TIGHT
  004015 BRU   TIGHT        004015 000400 000400 0000  620223 00 777775 0 000223 000000   1771    000000 000003 000000
TIGHT
  004015 BRU   TIGHT        004015 000400 000400 0000  620223 00 777775 0 000223 000000   1775    000000 000003 000000
TIGHT
  004015 BRU   TIGHT        004015 000400 000400 0000  620223 00 777775 0 000223 000000   1779    000000 000003 000000
TIGHT
  004015 BRU   TIGHT        004015 000400 000400 0000  620223 00 777775 0 000223 000000   1783    000000 000003 000000
TIGHT
  004015 BRU   TIGHT        004015 000400 000400 0000  620223 00 777775 0 000223 000000   1787    000000 000003 000000
TIGHT
  004015 BRU   TIGHT        004015 000400 000400 0000  620223 00 777775 0 000223 000000   1791    000000 000003 000000
TIGHT
  004015 BRU   TIGHT        004015 000400 000400 0000  620223 00 777775 0 000223 000000   1795    000000 000003 000000
TIGHT
  004015 BRU   TIGHT        004015 000400 000400 0000  620223 00 777775 0 000223 000000   1799    000000 000003 000000
TIGHT
CS I0 ON  1 TO  00000215 AT     1801
  004015 BRU   TIGHT        004015 000400 000400 0000  620223 00 777775 0 000223 000000   1813    000000 000002 000002
TIGHT
  004006 LDA                000003 000003 000400 0000  200217 00 777775 0 000217 037037   1853 01 000000 000003 000002
  004007 STA   CHIBL        000003 000003 000400 0000  607742 00 777775 0 007742 037037   1859 01 000000 000003 000002
  004010 LDA                000213 000213 000400 0000  200220 00 777775 0 000220 037037   1863 01 000000 000003 000002
  004011 STA   CHIADR       000213 000213 000400 0000  607743 00 777775 0 007743 037037   1869 01 000000 000003 000002
  004012 OPT   CHIASR       000002 000213 000400 0000  160211 00 777775 0 000211 037037   1875 01 000000 000003 000000
  004013 EXIT               000000 000213 000400 0000  000016 00 777775 0 000000 040002   1911 01 000000 000003 000000
  004020 LDE   CALLER       777777 000213 777777 0000  520405 00 777775 0 000405 040002   1919 01 000000 000003 000000
CS I0 ON  1 TO  00000213 AT     1950
```

```
    IC  INST   OPERAND     [EAD]  AC     EA     ICDO  [IC]   P  INDEX  X  EAD    SLR     TIME  I  LSR    ISR    ASR

  004000 LDX  CLK          777775 000213 777777 0000  540210 00 777775 0 000210 001001  1968 02 177777 000000 000000
  004001 TIX               000000 000213 777777 0010  000011 00 777776 0 000000 001001  1974 02 177777 000000 000000
  004002 BRC  GOON         004004 000213 777777 0000  420224 00 777776 0 000224 001001  1978 02 177777 000000 000000
GOON
  004004 STX  CLK          777776 000213 777777 0000  740210 00 777776 0 000210 001001  1984 02 177777 000000 000000
  004005 TIN               000020 000213 777777 0000  720216 00 777776 0 000216 040002  2006 02 000000 000000 000000
  004021 LDX  B            000213 000213 777777 0000  540400 00 000213 0 000400 040002  2010 01 000000 000000 000000
  004022 LDA               000015 000015 777777 0000  210000 00 000213 1 000213 040002  2016 01 000000 000000 000000
  004023 TIX               000000 000015 777777 0010  000011 00 000214 0 000000 040002  2022 01 000000 000000 000000
  004024 ADD               000013 000030 777777 0010  050000 00 000214 1 000214 040002  2028 01 000000 000000 000000
  004025 TIE               000000 000030 000000 0010  000015 00 000214 0 000000 040002  2034 01 000000 000000 000000
  004026 BRC  LOOP         004023 000030 000000 0000  420410 00 000214 0 000410 040002  2038 01 000000 000000 000000
LOOP
  004023 TIX               000000 000030 000000 0010  000011 00 000215 0 000000 040002  2044 01 000000 000000 000000
  004024 ADD               000014 000044 000000 0010  050000 00 000215 1 000215 040002  2050 01 000000 000000 000000
  004025 TIE               000000 000044 000000 0000  000015 00 000215 0 000000 040002  2056 01 000000 000000 000000
  004026 BRC  LOOP         004023 000044 000000 0000  420410 00 000215 0 000410 040002  2060 01 000000 000000 000000
  004027 STA  SUM          000044 000044 000000 0000  600402 00 000215 0 000402 040002  2066 01 000000 000000 000000
  004030 LDE  CALLER       000400 000044 000400 0000  520404 00 000215 0 000404 040002  2070 01 000000 000000 000000
  004031 LDX               000401 000044 000400 0000  540406 00 000401 0 000406 040002  2074 01 000000 000000 000000
  004032 LDA  SUB          000600 000600 000400 0000  200403 00 000401 0 000403 040002  2078 01 000000 000000 000000
  004033 LDP               000000 000600 000400 0000  000012 00 000401 0 000000 040002  2081 01 000000 000000 000000
  004034 BRM  WORKERS      004035 000600 000400 0000  060600 00 000401 0 000600 040002  2089 01 000000 000000 000000
WORKERS
  004037 STE  SAVEP        000400 000600 000400 0000  100602 00 000401 0 000602 040002  2095 01 000000 000000 000000
  004040 LDA               000402 000402 000400 0000  210000 00 000401 1 000401 040002  2101 01 000000 000000 000000
CS IO ON  1 TO  00000214 AT     2101
  004041 STA  ARG          000402 000402 000400 0000  600603 00 000401 0 000603 040002  2117 01 000000 000000 000000
  004042 LDI  ARG          000402 000044 000400 0000  120603 00 000401 0 000603 040002  2123 01 000000 000000 000000
  004043 STA  SUMW         000044 000044 000400 0000  600604 00 000401 0 000604 040002  2129 01 000000 000000 000000
  004044 LDA  SAVEP        000400 000400 000400 0000  200602 00 000401 0 000602 040002  2133 01 000000 000000 000000
  004045 BRU  WORKERS      004035 000400 000400 0000  620600 00 000401 0 000600 040002  2137 01 000000 000000 000000
WORKERS
  004035 LDP               000000 000400 000400 0000  000012 00 000401 0 000000 040002  2140 01 000000 000000 000000
  004036 TIN               000200 000400 000400 0000  720407 00 000401 0 000407 037037  2162 01 000000 000000 000000
  004014 TIN               000010 000400 000400 0000  720221 00 000401 0 000221 000000  2184 01 000000 000000 000000
  004015 BRU  TIGHT        004015 000400 000400 0000  620223 00 000401 0 000223 000000  2188    000000 000000 000000
TIGHT
  004015 BRU  TIGHT        004015 000400 000400 0000  620223 00 000401 0 000223 000000  2192    000000 000000 000000
TIGHT
  004015 BRU  TIGHT        004015 000400 000400 0000  620223 00 000401 0 000223 000000  2196    000000 000000 000000
TIGHT
  004015 BRU  TIGHT        004015 000400 000400 0000  620223 00 000401 0 000223 000000  2200    000000 000000 000000
TIGHT
  004000 LDX  CLK          777776 000400 000400 0000  540210 00 777776 0 000210 001001  2240 02 177777 000000 000000
  004001 TIX               000000 000400 000400 0010  000011 00 777777 0 000000 001001  2246 02 177777 000000 000000
  004002 BRC  GOON         004004 000400 000400 0000  420224 00 777777 0 000224 001001  2250 02 177777 000000 000000
GOON
CS IO ON  1 TO  00000215 AT     2250
  004004 STX  CLK          777777 000400 000400 0000  740210 00 777777 0 000210 001001  2266 02 177777 000002 000002
```

```
 IC  INST   OPERAND    (EAD)  AC     EA     ICDO  (IC)   P  INDEX  X EAD .  SLR     TIME I  LSR    ISR    ASR

 004005 TIN             000020 000400 000400 0000 720216 00 777777 0 000216 000000  2288 02 000000 000002 000002
 004006 LDA             000003 000003 000400 0000 200217 00 777777 0 000217 037037  2328 01 000000 000003 000002
 004007 STA  CH1BL      000003 000003 000400 0000 607742 00 777777 0 007742 037037  2334 01 000000 000003 000002
 004010 LDA             000213 000213 000400 0000 200220 00 777777 0 000220 037037  2338 01 000000 000003 000002
 004011 STA  CH1ADR     000213 000213 000400 0000 607743 00 777777 0 007743 037037  2344 01 000000 000003 000002
 004012 OPT  CH1ASR     000002 000213 000400 0000 160211 00 777777 0 000211 037037  2350 01 000000 000003 000000
 004013 EXIT            000000 000213 000400 0000 000016 00 777777 0 000000 040002  2386 01 000000 000003 000000
 004020 LDE  CALLER     777777 000213 777777 0000 520405 00 777777 0 000405 040002  2394 01 000000 000003 000000
 004021 LDX  B          000213 000213 777777 0000 540400 00 000213 0 000400 040002  2398 01 000000 000003 000000
CS IO ON  1 TO  00000213 AT    2400
 004022 LDA             000020 000020 777777 0000 210000 00 000213 1 000213 040002  2412 01 000000 000003 000000
 004023 TIX             000000 000020 777777 0010 000011 00 000214 0 000000 040002  2418 01 000000 000003 000000
 004024 ADD             000016 000036 777777 0010 050000 00 000214 1 000214 040002  2424 01 000000 000003 000000
 004025 TIE             000000 000036 000000 0010 000015 00 000214 0 000000 040002  2430 01 000000 000003 000000
 004026 BRC  LOOP       004023 000036 000000 0000 420410 00 000214 0 000410 040002  2434 01 000000 000003 000000
LOOP
 004023 TIX             000000 000036 000000 0010 000011 00 000215 0 000000 040002  2440 01 000000 000003 000000
 004024 ADD             000017 000055 000000 0010 050000 00 000215 1 000215 040002  2446 01 000000 000003 000000
 004025 TIE             000000 000055 000000 0000 000015 00 000215 0 000000 040002  2452 01 000000 000003 000000
 004026 BRC  LOOP       004023 000055 000000 0000 420410 00 000215 0 000410 040002  2456 01 000000 000003 000000
 004027 STA  SUM        000055 000055 000000 0000 600402 00 000215 0 000402 040002  2462 01 000000 000003 000000
 004030 LDE  CALLER     000400 000055 000400 0000 520404 00 000215 0 000404 040002  2466 01 000000 000003 000000
 004031 LDX             000401 000055 000400 0000 540406 00 000401 0 000406 040002  2470 01 000000 000003 000000
 004032 LDA  SUB        000600 000600 000400 0000 200403 00 000401 0 000403 040002  2474 01 000000 000003 000000
 004033 LDP             000000 000600 000400 0000 000012 00 000401 0 000000 040002  2477 01 000000 000003 000000
 004034 BRM  WORKERS    004035 000600 000400 0000 060600 00 000401 0 000600 040002  2485 01 000000 000003 000000
WORKERS
 004037 STE  SAVEP      000400 000600 000400 0000 100602 00 000401 0 000602 040002  2491 01 000000 000003 000000
 004040 LDA             000402 000402 000400 0000 210000 00 000401 1 000401 040002  2497 01 000000 000003 000000
 004041 STA  ARG        000402 000402 000400 0000 600603 00 000401 0 000603 040002  2503 01 000000 000003 000000
 004000 LDX  CLK        777777 000402 000400 0000 540210 00 777777 0 000210 001001  2543 02 177777 000003 000000
 004001 TIX             000000 000402 000400 0010 000011 00 000000 0 000000 001001  2549 02 177777 000003 000000
CS IO ON  1 TO  00000214 AT    2551
 004002 BRC  GOON       004004 000402 000400 0000 420224 00 000000 0 000224 001001  2563 02 177777 000003 000000
GOON
 004004 STX  CLK        000000 000402 000400 0000 740210 00 000000 0 000210 001001  2569 02 177777 000003 000000
 004005 TIN             000020 000402 000400 0000 720216 00 000000 0 000216 040002  2591 02 000000 000003 000000
 004042 LDI  ARG        000402 000055 000400 0000 120603 00 000000 0 000603 040002  2597 01 000000 000003 000000
 004043 STA  SUMW       000055 000055 000400 0000 600604 00 000000 0 000604 040002  2603 01 000000 000003 000000
 004044 LDA  SAVEP      000400 000400 000400 0000 200602 00 000000 0 000602 040002  2607 01 000000 000003 000000
 004045 BRU  WORKERS    004035 000400 000400 0000 620600 00 000000 0 000600 040002  2611 01 000000 000003 000000
WORKERS
 004035 LDP             000000 000400 000400 0000 000012 00 000000 0 000000 040002  2614 01 000000 000003 000000
 004036 TIN             000200 000400 000400 0000 720407 00 000000 0 000407 037037  2636 01 000000 000003 000000
 004014 TIN             000010 000400 000400 0000 720221 00 000000 0 000221 000000  2658 01 000000 000003 000000
 004015 BRU  TIGHT      004015 000400 000400 0000 620223 00 000000 0 000223 000000  2662    000000 000003 000000
TIGHT
 004015 BRU  TIGHT      004015 000400 000400 0000 620223 00 000000 0 000223 000000  2666    000000 000003 000000
TIGHT
 004015 BRU  TIGHT      004015 000400 000400 0000 620223 00 000000 0 000223 000000  2670    000000 000003 000000
```

```
   IC  INST    OPERAND     [EAD]   AC     EA    ICDO   [IC]   P INDEX  X   EAD    SLR      TIME  I   LSR    ISR    ASR

TIGHT
  004015 BRU  TIGHT       004015 000400 000400  0000  620223 00 000000 0 000223 000000    2674     000000 000000 000000
TIGHT
  004015 BRU  TIGHT       004015 000400 000400  0000  620223 00 000000 0 000223 000000    2678     000000 000000 000000
TIGHT
  004015 BRU  TIGHT       004015 000400 000400  0000  620223 00 000000 0 000223 000000    2682     000000 000000 000000
TIGHT
  004015 BRU  TIGHT       004015 000400 000400  0000  620223 00 000000 0 000223 000000    2686     000000 000000 000000
TIGHT
  004015 BRU  TIGHT       004015 000400 000400  0000  620223 00 000000 0 000223 000000    2690     000000 000000 000000
TIGHT
  004015 BRU  TIGHT       004015 000400 000400  0000  620223 00 000000 0 000223 000000    2694     000000 000000 000000
TIGHT
  004015 BRU  TIGHT       004015 000400 000400  0000  620223 00 000000 0 000223 000000    2698     000000 000000 000000
TIGHT
CS IO ON   1 TO  00000215 AT     2700
  004015 BRU  TIGHT       004015 000400 000400  0000  620223 00 000000 0 000223 000000    2712     000000 000002 000002
TIGHT
  004006 LDA              000003 000003 000400  0000  200217 00 000000 0 000217 037037    2752 01 000000 000000 000002
  004007 STA  CH1BL       000003 000003 000400  0000  607742 00 000000 0 007742 037037    2758 01 000000 000000 000002
  004010 LDA              000213 000213 000400  0000  200220 00 000000 0 000220 037037    2762 01 000000 000000 000002
  004011 STA  CH1ADR      000213 000213 000400  0000  607743 00 000000 0 007743 037037    2768 01 000000 000000 000002
  004012 OPT  CH1ASR      000002 000213 000400  0000  160211 00 000000 0 000211 037037    2774 01 000000 000000 000000
  004013 EXIT             000000 000213 000400  0000  000016 00 000000 0 000000 040002    2810 01 000000 000000 000000
  004020 LDE  CALLER      777777 000213 777777  0000  520405 00 000000 0 000405 040002    2818 01 000000 000000 000000
CS IO ON   1 TO  00000213 AT     2850
  004000 LDX  CLK         000000 000213 777777  0000  540210 00 000000 0 000210 001001    2868 02 177777 000000 000000
  004001 TIX              000000 000213 777777  0000  000011 00 000000 0 000000 001001    2874 02 177777 000000 000000
  004002 BRC  GOON        004004 000213 777777  0000  420224 00 000000 0 000224 001001    2878 02 177777 000000 000000
  004003 HLT              000000 000213 777777  0000  000000 00 000000 0 000000 001001    2881 02 177777 000000 000000
```

```
INTERRUPT LOCATIONS

000000  000000 000000 000000 000000 000000 000000 000000 000000   000010  000000 000000 000000 004015 000000 000000 037037 004006
000020  000000 000000 040002 004021 177777 000000 001001 004000   000030  000000 000000 000000 000000 000000 000000 000000 000000
000160  000000 000000 000000 000000 000000 000000 000000 000000   000170  000000 000000 040002 004030 000000 000000 000000 004016
000200  000000 000000 037037 004014 000000 000000 040002 004020   000210  000000 000002 000002 000023 000021 000022 000020 000003

    DEMO          [DATA] 000210-000224

000200  000000 000000 037037 004014 000000 000000 040002 004020   000210  000000 000002 000002 000023 000021 000022 000020 000003
000220  000213 000010 000170 004015 004004 000000 000000 000000   000230  000000 000000 000000 000000 000000 000000 000000 000000

    DEMO          [CODE] 004000-004017

004000  540210 000011 420224 000000 740210 720216 200217 607742   004010  200220 607743 160211 000016 720221 620223 000000 720222

    WORKER        [DATA] 000400-000410

000400  000213 000402 000055 000600 000400 777777 000401 000200   000410  004023 000000 000000 000000 000000 000000 000000 000000

    WORKER        [CODE] 004020-004036

004020  520405 540400 210000 000011 050000 000015 420410 600402   004030  520404 540406 200403 000012 060600 000012 720407 100602

    WORKERS       [DATA] 000600-000601

000600  004035 004037 000400 000402 000055 000000 000000 000000   000610  000000 000000 000000 000000 000000 000000 000000 000000

    WORKERS       [CODE] 004037-004045

004020  520405 540400 210000 000011 050000 000015 420410 600402   004030  520404 540406 200403 000012 060600 000012 720407 100602
004040  210000 600603 120603 600604 200602 620600 000000 000000   004050  000000 000000 000000 000000 000000 000000 000000 000000

REMAINING CORE IN DATA REGION

000600  004035 004037 000400 000402 000055 000000 000000 000000   000610  000000 000000 000000 000000 000000 000000 000000 000000

REMAINING CORE IN CODE REGION

004040  210000 600603 120603 600604 200602 620600 000000 000000   004050  000000 000000 000000 000000 000000 000000 000000 000000
007740  000000 000000 000002 000214 000000 000000 000000 000000   007750  000000 000000 000000 000000 000000 000000 000000 000000
```

RUNNING TIME = .003 SECS
INSTRUCTIONS EXECUTED = 324

| INST | COUNT | INST | COUNT | INST | COUNT |
|---|---|---|---|---|---|
| ADX | 0 | ADD | 10 | BRM | 5 |
| STE | 5 | LDI | 5 | SHF | 0 |
| OPT | 6 | LDA | 32 | TXLE | 0 |
| SUB | 0 | TAL | 0 | ETR | 0 |
| STI | 0 | CYC | 0 | DSH | 0 |
| LDL | 0 | BRC | 20 | MUL | 0 |
| TAE | 0 | MRG | 0 | LDE | 11 |
| LDX | 20 | DCY | 0 | STA | 27 |
| BRU | 106 | DIV | 0 | TAG | 0 |
| EOR | 0 | TIN | 20 | STX | 9 |
| IPF | 0 | HLT | 2 | TOV | 0 |
| NOP | 0 | TAP | 0 | NEG | 0 |
| TOP | 0 | ADC | 0 | ROV | 0 |
| CMP | 0 | TIX | 20 | LDP | 10 |
| LDD | 0 | NORM | 0 | TIE | 10 |
| EXIT | 6 | CPD | 0 | SIO | 0 |
| TAZ | 0 | FLP | 0 | RED | 0 |
| RIO | 0 | ACX | 0 | AEA | 0 |
| EAX | 0 | | | | |

: PAUSE