RA

stichting
mathematisch
centrum

MC

REKENAFDELING RA MR 140/72 OCTOBER

ANDREW S. TANENBAUM
PDP-11 SIMULATION AND PERFORMANCE MONITORING

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

TABLE OF CONTENTS

INTRODUCTION

The program described in this report simulates a basic PDP-11/45 computer, its assembler, and a batch operating system which allows a series of assembly language programs to be read in as a group and then assembled and executed one after another, without operator intervention.

The primary goal of the simulation package is to provide a facility for running small student programs in conjunction with a course involving PDP-11 programming. A second goal is to provide a facility for measuring the performance of PDP-11 programs and analyzing their behaviour, possibly in conjunction with the design of some PDP-11 systems programs. A third goal is to allow the instructor in a course on PDP-11 programming to test sequences of code before showing them to the class, to avoid unleashing large numbers of faulty programs on the unwitting students. A fourth goal is to ease the problem of debugging PDP-11 programs, since the simulator provides many more diagnostics and debugging aids than the bare machine itself does. Since the simulation package is written in Algol 60, it can be run on any computer with an Algol 60 compiler (in principle). Thus actual PDP-11 programming experience can be acquired at an installation, that does not in fact have a PDP-11 available.

The simulation package is a single Algol program consisting of three logical sections. First, the executive, which prepares the system for use by reading in certain tables and initializing a number of global parameters. Second there is the assembler, which reads an assembly language program and from it produces an object program in absolute binary. Third there is the simulator which interpretively executes an absolute binary program. When the interpretation is finished, an analysis of the executed program is printed, and control returns to the assembler to begin the next assembly.

The machine simulated is the basic PDP-11/45 as described in the Digital Equipment Corporation publication "Processor Handbook". The simulator includes all machine instructions in the standard CPU, (thus not those concerned with the floating point processor or segmentation unit). The paper tape reader, paper tape punch, and console teleprinter are also simulated, including the 7 level hardware and software interrupt system. Automatic code conversion is provided, so that the PDP program can read, punch and type in ASCII code.

It is assumed that the reader of this report has read and intellectually digested the contents of the PDP-11/45 Processor Handbook and the PAL-11R programming manual.

A BRIEF DESCRIPTION OF THE ASSEMBLY LANGUAGE

The input language is close to being a subset of the PAL-11 programming language, warts and all, as supplied by DEC. This description covers only the major features.

A program consists of a series of statements. Each statement is terminated by a carriage return followed by a line feed. Statement elements are identifiers, integers, and the special characters: % = # @ ( ) , ; + - . & ! . Spaces are not permitted within identifiers or integers and both must be followed by a character other than a letter or digit (e.g. space). Spaces may be inserted between statement elements. Identifiers are strings of 1-6 lower case letters and digits, the first of which is a letter. Integers are either octal (not followed by a point) or decimal (followed by a point).

A statement consists of 4 fields, all of which are optional, but when present must be in the order: label, operation, operands, comment. The label field consists of 0 or more labels, a label being an identifier and a colon.

The operation field is a single instruction or pseudo instruction chosen from the PDP machine instructions and pseudo instructions.

The operand field consists of 0 or more operands. If more than one operand is present, the operands must be separated by commas. Each machine instruction requires a specific number of operands. Some pseudo instructions accept a variable number of operands. All pseudo instructions begin with a point.

Each operand has a mode corresponding to one of the 8 hardware modes. The assembler notation for each mode is shown below.

0	register	r		
1	register indirect	@r	or	(r)
2	auto increment	(r)+	or	#e
3	auto increment indirect	@(r)+	or	@#e
4	auto decrement	-(r)		
5	auto decrement indirect	@-(r)		
6	index	e(r)	or	e
7	index indirect	@e(r)	or	@e

where e is an address expression and r a register expression.

An expression consists of a sequence of values separated by operators. A value is an identifier,an integer, or the current location symbol (point). The operators are plus, minus, logical and and logical or. A register expression is an expression containing an identifier or integer prefixed by a percent sign, or a variable equated to a register expression.

A statement of the form identifier, equals symbol, expression assigns the value of the expression to the identifier.

Programs must end with the .end pseudo instruction, followed by the starting address.

The following addresses are needed for input-output and are predefined.

pir	-	programmed interrupt request register
prs	-	paper tape reader status register
prb	-	paper tape reader buffer register
pps	-	paper tape punch status register
ppb	-	paper tape punch buffer register
tps	-	console typewriter status register
tpb	-	console typewriter buffer register

The assembly language differs from the PAL-11 language in the following ways

1. Upper case letters, single quotes, double quotes, tabs, and form feeds should only be used in text.

2. Labels may not contain periods or dollar signs.

3. Missing operands and operators are not allowed.

4. All assemblies are absolute.

5. The following pseudo instructions are the only ones allowed:
   .title,.word,.byte,.ascii,.print,.core,.trace,.stoptr,.regd,.end

6. If the instruction location counter is odd after assembling a statement, it will be increased by 1 before the succeeding statement is assembled.

A program of N statements requires about 3 + N/5 seconds to assemble.

A Context Free PAL Grammar

```
<program> :: = <statement> | <statement> <program>
<statement> :: = <unlabeled statement> | <identifier> : <statement>
<unlabeled statement> :: = <tail> | <body> <tail>
<body> :: = <opcode> | <opcode> <operand list> | <assignment>
<tail> :: = ; <comment> <newline> | <newline>
<operand list> :: = <operand> | <operand> , <operand list>
<operand> :: = <direct> | @ <direct>
<assignment> :: = <identifier> = <expression>
<direct> :: = <expression> | <expression> <paren> | <paren> |
              <paren> +  | # <expression> |  - <paren>
<paren> :: =  (<expression>)
<expression> :: = <unsigned expression> | <arithop> <unsigned expression>
<unsigned expression> :: = <term> | <term> <op> <unsigned expression>
<term> :: = <val> | % <val> | .
<val> :: = <identifier> | <integer>
<arithop> :: = + | -
<op> :: = + | - | & | !
```

```
<newline>    is a carriage return + line feed or a new card
<comment>    is any character sequence not including <newline>
<identifier> is 1-6 letters and digits starting with a letter
<integer>    is 1-6 digits optionally followed by a point
<opcode>     is a machine instruction or pseudo instruction
```

DEBUGGING FACILITIES

Several pseudo instructions have been included to aid program debugging. They are assembled as EMT instructions so that they could be easily implemented on a real PDP-11. These pseudo instructions are:

1. .trace

   This causes the simulator to print out certain status information after each instruction is executed. The information printed consists of

   1. simulated time in milliseconds (decimal)
   2. contents of registers r0 thru r7 (octal)
   3. the 4 condition codes NZVC (0 = off, 1 = on)
   4. the last instruction executed (octal)

2. .stoptr

   This stops tracing as described above. Both .trace and .stoptr are dynamic, that is, tracing is begun when a .trace pseudo instruction is executed during the simulation and stopped when a .stoptr pseudo instruction is executed, independent of their position in the source text. Thus .trace and .stoptr are more analagous to procedure calls than <u>begin</u> <u>end</u> brackets in Algol. Both .trace and .stoptr occupy 1 machine word in the object program. The initial state is .stoptr.

3. .regd (register dump)

   This prints the same information as .trace, but one time only, unlike .trace which may cause many lines to be printed. This allows the user to get the machine status only when desired, and not after every instruction.

4. .print expr1, expr2, ... exprn

   This prints out the machine words (in octal) whose addresses are expr1, expr2, ... exprn. The maximum number of arguments is 16. For example, .print x, y, a, a+2, a+4 will print out 5 machine words, whose addresses are x, y, a, a+2, and a+4. The expr's should always evaluate to even numbers. If an expr is odd, the contents of expr-1 will be printed. The assembled code occupies n+2 words.

5. .core expr1, expr2

This will produce an octal core dump from expr1 to expr2 inclusive. If expr1 or expr2 are not integral multiples of 32, a few extra words will be printed to enhance readability. The pseudo instruction assembles in 3 machine words.

JOB SETUP

Paper tape -

Only ISO code tapes are allowed. The program is followed by any number of data tapes, each containing a PAL program and its associated data. The data begins with the character following the line feed which ends the .end statement. Extra carriage returns or blank tape will be read by the PDP.

The format for each tape is:

1. 30 cm. of blank tape
2. 2 escape characters (ESC on the Olivetti's)
3. new line (i.e. carriage return, line feed)
4. the program. The last statement must be .end followed by the starting address. The statement is terminated by a new line. First line must be .title
5. data, if any.

Cards -

The characters @ & ! do not appear on the IBM-EL card punches, so $ ∧ ∨ should be used in place of them in assembly programs. As data they have their usual meaning.

The format for each program is

1. A card containing 12 - 11 - 1 - 2 punches in columns 1 and 2 (made by superimposing the letters A and K) and .title in columns 3-8. Column 9 must be blank.
2. Rest of program, ending with .end Statement
3. Data, if any. The data must begin in column 1 of the card following .end

EXPLANATION OF OUTPUT

Assembly Listing -

For each program processed, an assembly listing is produced. The listing consists of 6 columns, labeled ERROR MESSAGES, ADDRESS, INST, IMMED 1, IMMED 2, SOURCE STATEMENT. The first column contains error messages. Only the first error message produced by a statement is printed. A halt instruction is assembled in place of the errant statement.

The ADDRESS column contains the octal machine address at which the object code begins. This column is blank if no object code is provided. The INST column contains the machine instruction(or in any case the first word of code) generated by the statement, in octal. The IMMED 1 and IMMED 2 columns contain the first and second immediate operands (or in any case the second and third words of generated code) in octal. The column labelled SOURCE STATEMENT contains the original source text.

At the end of the assembly listing is a line containing the number of assembly errors detected, in decimal, and the X8 time used for the assembly in decimal seconds.

Simulator -

Output from the simulator consists of .trace output, .print output and core dumps, all of which have been described earlier.

Post Mortem Statistics -

After the job has finished executing, 4 groups of information are printed for debugging and performance monitoring. All numbers are decimal except the machine addresses in group 3.

Group 1 is an octal dump of all non-zero memory, 16 words per line, with the address of the first word printed at the left.

Group 2 contains the following items

PDP TIME USED - How much real time the program would take on the PDP-11 in milliseconds.

X8 TIME USED - How many milliseconds of X8 time the simulation phase required.

X8 TIME PER PDP SECOND - The number of seconds of X8 time required to simulate 1 second of PDP-11 time at this rate.

PDP WAIT TIME - How many milliseconds the PDP-11 spent waiting for interrupts as a consequence of wait instructions.

NUMBER OF INTERRUPTS - The number of io interrupts processed.

INSTRUCTIONS EXECUTED - How many PDP-11 instructions were executed during the simulation.

NUMBER OF CHARACTERS READ - How many characters were read from the paper tape.

NUMBER OF LINES TYPED - How many carriage returns were typed by the program. This does not include debug output (trace, dump etc.)

NUMBER OF CHARACTERS PUNCHED - How many characters were punched onto paper tape.

AVERAGE TIME PER INSTRUCTION - How many nanoseconds the mean instructions would require on the PDP-11 itself.

AVERAGE NUMBER OF PDP INSTRUCTIONS PER SECOND OF X8 TIME - The simulation rate.

DIST OF ADDR MODES 0 - 7 - The number of operands that were mode 0, 1, 2, 3, 4, 5, 6, and 7 respectively.

IMMEDIATE - number of operands of mode = 2, register = 7

DIRECT - number of operands of mode = 3, register = 7

PIC - number of operands of mode = 6, register = 7

Group 3 contains a tabular histogram of the program counter. The address space is divided into 128 equal sized regions. The printed table contains 128 values, giving for each region the number of times an instruction was fetched from that region. The lower limits of each region are printed in octal, since they are machine addresses. The number of instructions fetched from that region follows, in decimal. From this distribution the user can find out where the program "spent most of its time". The number of instructions fetched from below the lowest region and above the highest is printed above the table.

Group 4 contains the number of times each machine instruction was executed e.g. the total number of mov instructions, the total number of cmp instructions etc. The first column gives the number of ADC, ASHC, BGE etc. instructions executed. After the mnemonic is the total number of times the instruction was executed, followed by that number as a percent of the total number of instructions executed.

Following Group 4 is the total time, in decimal seconds, for the job, including assembly time, execution time, and time to print the statistics.

ERROR MESSAGES

The following list gives the meaning of messages put out by the assembler. An asterisk after a message indicates detected on pass 1.

ADDRESS ERROR - No code may be generated at addresses above 32767.

ADDRESS EXPECTED - An expression which evaluates to a machine address is required, but something else is present.

ADDRESS FIELD EMPTY - An operand is required, but none is present.

ASSEMBLY TERMINATED BY TABLE OVERFLOW - The assembler's internal tables are full. Assembly cannot continue.

BAD CHAR IN EXPR - An expression contains a character other than a legal statement element.

BR TO ODD ADDRESS - The expression in the address field evaluates to an odd number. Since all instructions begin at an even numbered byte, this is incorrect.

BR TO REG ILLEGAL - A branch instructions may not branch to a register. The given expression evaluates to a register expression.

CLOSE PAREN MISSING - An opening parenthesis has been read but the corresponding closing parenthesis is missing.

.END WAS MISSING - A program must have .end as its last instruction.

ERROR IN RHS OF EXPR* - The right hand side of the definition contains either an undefined symbol, an 8 or a 9 in an octal number, or a syntax error.

ERR IN START ADDR - The expression following the .end instruction must evaluate to a word address <32768.

FIELD 1 MUST BE REG - Assembler syntax requires that the first operand be a register expression.

FIELD 2 MUST BE REG - Assembler syntax requires the second operand be a register expression.

FIRST CHAR OF LABEL* - The assembler expected either a label or an opcode, which means that the first char must be either a letter or a point.

FIRST FIELD WRONG - The first field must be an expression evaluating to a machine address <65536.

JSR TO REG ILLEGAL - A jsr instruction must have a machine address, not a register, as its second operand.

LHS ALREADY DEFINED* - The left hand side of this assignment is a variable previously defined.

MISSING OPERAND - An operand is required and is absent.

MULTIPLY DEFINED SYM* - A label used here has been previously defined.

NN MUST BE 1-63 - The address field of a mark instruction must evaluate to a positive integer <64.

NO CLOSING DELIMITER* - The first non-blank character after the .ascii pseudo instruction is the opening delimiter. Everything following it until the next appearance of the delimiter is text. The delimiter did not occur again before the carriage return.

UPCODE UNKNOWN* - The operation field contains something other than a valid machine instruction or pseudo instruction.

OVERFLOW* - A number may not be larger than 65535.

PARITY ERROR - A character with incorrect parity has been read from the source tape. It has been changed to a question mark.

REG NOT ALLOWED - A register has occurred in a position where only a machine address is permitted.

REGISTER > 7 - A register expression must be in the range 0-7.

REGISTER? - An immediate operand may not be a register.

SECOND FIELD WRONG - The second operand field is not an expression evaluating to a machine address <65536.

SYNTAX ERROR IN EXPR - The assembler cannot recognize the structure of an operand.

TAG FOLLOWED BY ? - After a tag in the label or opcode field, a colon, equal sign or space is expected.

TOO FAR - The address of a branch instruction must not be farther away than -128 or +127 words.

TOO FEW FIELDS* - One or more operands is missing.

TOO MANY DIGITS* - Numbers may not have more than 6 digits.

TOO MANY FIELDS* - An instruction has been supplied with more operands than is allowed.

TOO MANY PERCENTS - A percent sign may not directly follow a percent sign.

TRUNCATION ERROR - A word expression is greater than 65535 or a byte expression is greater than 255.

UNDEFINED SYMBOL - A symbol is used without it ever being defined.

8 OR 9 IN AN OCTAL NUMBER* - Octal numbers consist of the digits 0-7 only. Decimal numbers may contain 8 and 9 but must be followed by a point.

The following list gives the meaning of messages issued by the simulator. All errors stop the simulator.

ADDRESS OUT OF BOUNDS - In a mode 3, 5, or 7 operand, the specified register contains an address larger than the amount of memory available.

AUTODECREMENT REGISTER ODD - In a mode 5 operand, the specified register contains an odd address, where an even address is required.

AUTOINCREMENT REGISTER ODD- In a mode 3 operand, the specified register contains an odd address, where an even address is required.

ATTEMPT TO PRINT NONEXISTENT MEMORY LOCATION - One of the arguments of .print is greater than 32767.

BRANCH OFFSET ERROR - The branch address is either below 256 or above 32767.

CONSOLE TYPEWRITER BUSY - An attempt has been made to type out a character before the typewriter has finished the previous character.

CPU PRIORITY SET TO 0 - The CPU priority must be in the range 1-7.

DESTINATION ADDRESS ODD - The destination address must evaluate to an even number.

DESTINATION ADDRESS OUT OF BOUNDS - The source address is larger than the amount of memory available.

ESCAPE CHARACTER READ. NOT ALLOWED - The next character in the input stream was an escape character (27 decimal). This is forbidden since it is used to separate jobs. A possible cause is an attempt to read more data than is provided.

FLOATING POINT NOT SIMULATED - The optional floating point unit has not been simulated.

ILLEGAL INSTRUCTION - The operation code is not a legal PDP-11 operation code.

INDIRECT INDEXING REGISTER ODD - In a mode 7 operand, the specified register contains an odd address, where an even one is required.

INSTRUCTION LIMIT EXCEEDED - More instructions have been executed than are allowed by the simulator (analogous to time limit on the X8).

JSR TO REGISTER - The destination mode of a jsr instruction must not be made 0, ie. jumps to registers are not permitted.

JUMP TO IO AREA - A jump has been made to an address larger than 32767. This is forbidden in the simulator since it is inconceivable that this is not an error.

JUMP TO REGISTER - The destination mode of a jump instruction must not be mode 0, ie. jumps to registers are not permitted.

NO INTERRUPT PENDING - A wait instruction has been executed, but no interrupt is pending. Consequently it would have waited indefinitely.

NON EXISTENT IO DEVICE - A memory location above 32767 which does not refer to any input-output device has been referenced.

PC IN IO AREA - The program counter is larger than 32767, i.e. the program "ran off" the end of memory.

PC ODD - The program counter must always be even, since instructions are located on word boundaries.

PRINT MAY NOT HAVE MORE THAN 16 PARAMETERS - The .print pseudo-instruction has occurred with more than 16 parameters. Since this error is detected by the assembler, possibly memory has been overwritten.

PUNCH BUSY - An attempt has been made to punch a character while the punch is still busy with the previous one.

READER STILL BUSY - An attempt has been made to start the paper tape reader while it is still occupied with the previous command.

SOURCE ADDRESS ODD - The source address must evaluate to an even number.

SOURCE ADDRESS OUT OF BOUNDS - The source address is larger than the amount of memory available.

STACK OVERFLOW - An interrupt occurred, but there was no room left on the stack, i.e. register 6 was smaller than 256.

STACK POINTER ODD - An interrupt occurred, but register 6 contained an odd number. Register 6 is expected to contain a word address.

STACK POINTER TOO BIG - The stack pointer points to an address beyond available memory.

STACK UNDERFLOW - Return from trap, interrupt, or subroutine cannot proceed since stack is empty, ie. register 6 greater than 32767.

TELETYPE LINE OVERFLOW - More than 80 characters have been typed out since the last carriage return.

TOO MANY INTERRUPTS - More than 16 interrrupts pending. This is a simulator imposed, not a PDP limit.

TOO MANY LINES OF OUTPUT - More printed output has been generated than the simulator allows.

IMPLEMENTATION

The program consists of two major sections,the assembler and the simulator. These will be discussed in turn.

Assembler -

The assembler makes two complete passes over the input. On pass 1, each statement is read character by character, left to right without backtrack. Three tables are constructed by pass 1 and given to pass 2.

1. The symbol table	(SYMBOL TABLE, AUX)
2. The source text table	(TEXT)
3. The program	(LEX)

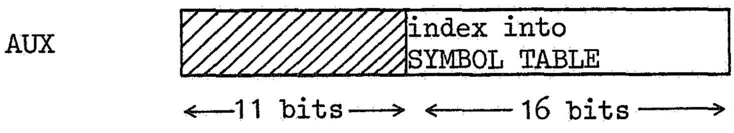
The symbol table records for each identifier

1. Whether it has been defined or not
2. It's value, if defined
3. Whether it is an address or a register.

The symbol table is implemented in two arrays. For each identifier the index is computed according to

$$\text{index} = ((2^{14}*(\text{char } 1 + \text{char } 4) + 2^{7}*(\text{char } 2 + \text{char } 5) + (\text{char } 3 + \text{char } 6)) \text{ modulo } 137) + 1$$

All identifiers with the same index are chained together in order of occurrence on a chain beginning at AUX[index]. The entries themselves are in SYMBOL TABLE, each entry using 3 words. Space is claimed from SYMBOL TABLE sequentially, so it contains no holes. The format of an AUX entry is



The format of a SYMBOL TABLE entry is

SYMBOL TABLE

←6 bits→	←7 bits→	←7 bits→	←7 bits→
▨	Char 1	Char 2	Char 3
▨	Char 4	Char 5	Char 6

link	D	R	value
←9 bits→	1	1	←16 bits→

D: 0 = undefined
1 = defined
R: 0 = address
1 = register
link:pointer to next entry on chain or zero for last one

The source text is stored in TEXT character by character for printing the listing. The format is

TEXT

▨	Char 1	Char 2	Char 3
←6 bits→	←7→	←7→	←7 bits→

The LEX table provides the information from which pass 2 generates object code. Each line of source text. including blank lines and comments produces 1 block in LEX. The first word of the block is the value of the instruction location counter i.e. the address where the next word of generated code will be placed. The second word has 2 fields. The right field is the value of the operation code. The left field is the instruction class. Classes are distinguished by the fact that the required operand format is usually different for each class.

LEX

0	instruction location counter
operand class	opcode value
type	val
type	val
←11 bits→	←16 bits→

type: 1 = identifier or integer
2 = special character
val: if identifier or integer index into SYMBOL TABLE
if special character, the code

The operand classes are

0-4	not used	
5	.word	list of word expressions
6	.byte	list of byte expressions
7	.ascii	text
8	not used	
9	.end	one address expression
10	binary	two operands
11	unary	one operand
12	noop	zero operands
13	trap	expression between 0 and 255
14	br	offset between -128 and +127
15	rts	one register
16	sob	register, followed by expression <64
17	mark	expression between 0 and 63
18	mul	operand followed by register
19	jsr	register followed by operand
20	empty	statement was a comment
21	pass 1 error	pass 1 error detected
22	empty	statement is treated as comment
23	.print	list of up to 16 address expressions
24	.title	text
25	.core	two address expressions
26	xor	register followed by operand

Note that the leftmost 11 bits can only be 0 for the first word of a block. Pass 2 uses this to determine the end of a (variable length) block.

The main pass 1 loop starts at NEW CARD, where variables are reinitialized for each statement. At PASS 1 the program is looking for labels. At GOPER all labels have been processed and the operands are being examined. When a carriage return is seen, control passes to CARET.

Pass 1 looks up opcodes and makes a partial examination of the operands in order to determine how much space to reserve in the object program. Consider the statement MOV X,Y. This will require 1 word if X and Y are registers, 2 words if only 1 is, and 3 words if neither is a register. A line of source text may yield an inherently variable number of machine words, as in .word.

Pass 2 reads the blocks in LEX and processes each one independently of its predecessor and successor. For each block, the format of the operands is known from the second word. The value of the opcode has already been found. Hence the main task of pass 2 is the analysis of the operands and generation of code. The loop that processes each statement begins at SETUP. The switch TYPE dispatches to the proper section of code for the statement. Operands are either address expressions, which yield a mode and register, e.g. #x, (R3)+ , or simply an expression yielding a number. The procedure OPERAND evaluates the former, and the procedure EXPR evaluates the latter. Code is generated by calling EMIT which puts it away. If the right part of the object program is in the buffer, CODE BUFFER, the generated word is put in the buffer. Otherwise the right piece of program is brought in.

The procedure LISTING prints the listing and returns to SETUP to begin the next statement.

Simulator-

The object program is loaded, either from the buffer or the drum as needed. It is stored in the array M. Bytes 0 and 1 are in M[0], bytes 2 and 3 are in M[1] etc. The cpu registers r0, r1, ... r2 are stored in M[-1], M[-2] ... M[-8] respectively. The condition codes N, Z, V, C are stored in the variables N, Z, V, C respectively, with 1 meaning on and 0 off.

The loop beginning at CYCLE picks up the instruction to be executed and decodes it, scattering its bits among 5 variables; bits 15 - 12 in OPCODE, 11 - 9 in SRC MODE, 8 - 6 in SRC REG, 5 - 3 in DST MODE, and 2 - 0 in DST REG. The switch INSTRUCTION TYPE branches to the appropriate piece of code to evaluate the given instruction.

The procedure EVAL BOTH OPERANDS is used to compute the source and destination addresses (SRC ADDR, DST ADDR) and values (SRC OPERAND, DST OPERAND) for which it needs the mode and register already decoded. All the two operand instructions begin with this procedure. The one operand instructions use EVAL ONE OPERAND which computes the destination address and value only.

Thus by the time the real work of the instruction interpretation begins, the source and destination addresses and operands are known. In many cases there is not much to do, mov for example simply puts the known source operand into the known destination address.

Pending interrupts are chained together in 3 parallel chains, INTERRUPT TIME, INTERRUPT VECTOR, and NEXT ON CHAIN. The first contains the simulated time when the interrupt should occur,in nanoseconds. The second the machine address of the interrupt vector, i.e. where the new program counter and program status word are to be fetched from. This is used to distinguish between device types. The third links the entries together. NEXT INTERRUPT TIME always contains the time of the earliest pending interrupt. In the main loop, the clock is compared to this variable before every instruction. If the clock is greater, the interrupt occurs. From the vector address the device type is deduced and the appropriate i.o. operation (e.g. read, punch, print) is performed. The simulator converts MC code to ASCII via the array Y, and ASCII to MC code via the array U.

The procedure STATISTICS prints all of the post mortem statistics.

```
'BEGIN'
'COMMENT' THIS PROGRAM SIMULATES A COMPLETE BATCH OPERATING SYSTEM FOR THE PDP-11/45, INCLUDING ASSEMBLER.  THE PROGRAM HAS TWO
MAIN SECTIONS: THE ASSEMBLER AND THE SIMULATOR.

THE ASSEMBLER -
     THE ASSEMBLER IS A TWO PASS ASSEMBLER.  PASS1 IS DEVOTED TO BUILDING THE SYMBOL TABLE AND TRANSFORMING THE INPUT TEXT
INTO A FORM MORE EFFICIENTLY PROCESSED IN PASS 2.  PASS 1 BUILDS 3 DATA STRUCTURES FOR PASS 2: THE SYMBOL TABLE (AUX  AND
SYMBOL TABLE), THE SOURCE TEXT TABLE  TEXT,  AND THE TRANSFORMED INPUT   LEX.

     THE SYMBOL TABLE IS A CHAINED HASH TABLE. NAMES TO BE ENTERED ARE HASHED BY TAKING A WEIGHTED AVERAGE OF THE LETTERS MODULO
137.  ALL NAMES HASHING TO K ARE LINKED TOGETHER IN A CHAIN WHOSE HEAD IS IN AUX[K].  THE ENTRIES THEMSELVES ARE IN SYMBOL TABLE.
EACH ENTRY CONTAINS THE NAME, WHETHER THE NAME HAS BEEN  DEFINED OR NOT, AND IF SO ITS VALUE AND WHETHER IT IS A REGISTER.

     THE TEXT TABLE CONTAINS THE ORIGINAL SOURCE TEXT 3 CHARACTERS PER WORD UNMODIFIED.  IT IS USED TO PRINT THE LISTING.

     THE TRANSFORMED INPUT TEXT IS IN THE ARRAY LEX, ONE BLOCK PER INPUT LINE, INCLUDING BLANK LINES AND COMMENTS.  EACH BLOCK HAS
A MINIMUM LENGTH OF 2 WORDS.  THE FIRST WORD IS THE VALUE OF THE INSTRUCTION LOCATION COUNTER, I.E. WHERE THE GENERATED OBJECT CODE
IS TO GO.  THE SECOND WORD HAS TWO PARTS, THE INSTRUCTION CLASS, IN  ESSENCE THE FORMAT THE OPERAND FIELD MUST BE, AND THE
VALUE OF THE OPCODE.  IT IS NECESSARY TO LOOK THE OPCODE UP ON PASS 1 IN ORDER TO DETERMINE HOW MUCH SPACE TO ALLOCATE FOR THE
GENERATED CODE (SOME INSTRUCTIONS TAKE 1, SOME 2, AND SOME 3 MACHINE WORDS).  SUCESSIVE WORDS IN THE BLOCK CONTAIN 1 ENTRY PER
STATEMENT ELEMENT.  AN IDENTIFIER, AN INTEGER, AND A SPECIAL CHARACTER ARE ALL EXAMPLES OF ELEMENTS.

     PASS 2 EVALUATES THE OPERAND FIELD TO FIND THE MODE OF EACH OPERAND.  IT THEN GENERATES THE COMPLETE INSTRUCTION AND PRINTS
THE ASSEMBLY LISTING.  THE GENERATED OBJECT CODE IS PUT INTO THE BUFFER, APPROPRIATELY NAMED  CODE BUFFER.  IT THE GENERATED OBJECT
CODE EXCEEDS THE BUFFER CAPACITY, THE BUFFER IS AUTOMATICALLY PAGED ONTO THE DRUM.  THE GENERATED PROGRAM IS THUS EITHER ON THE DRUM
OR IN THE BUFFER, DEPENDING ON ITS LENGTH.

THE SIMULATOR-
     THE SIMULATOR LOADS THE OBJECT PROGRAM INTO AN ARRAY M.  M[0] HOLDS BYTES 0 AND 1, M[1] HOLDS BYTES 2 AND 3, ETC.  M[K] FOR
-9 < K < 0 HOLDS CPU REGISTER -K+1.

     THE MAIN LOOP AT CYCLE PICKS UP 1 INSTRUCTION AT A TIME, EXAMINES THE OPCODE AND BRANCHES TO THE APPROPRIATE ROUTINE VIA THE
SWITCH  INSTRUCTION TYPE.  AFTER THE ROUTINE HAS DONE ITS THING, CONTROL IS RETURNED TO THE LABEL EXDONE.

     WHEN A JOB IS COMPLETED, EITHER BY HALTING OR AN ERROR EXIT, THE PROCEDURE STATISTICS IS CALLED TO PRINT A POST MORTEM SUMMARY
OF  NTERESTING GOODIES.  IT FINISHES UP BY PRINTING THE END OF JOB PAGE AND THEN CALLING THE ASSEMBLER TO START ON THE NEXT JOB;



'INTEGER' Q,TEXT BUFFER SIZE,MAXLEX,MAXTEXT,SYMBOL TABLE SIZE,BLOCK SIZE, CURRENTBLOCK,OFFSET,MAXCORE, IO ADDR,INST LIMIT,LINELIMIT,
STARTING ADDRESS, LOWEST ADDRESS,HIGHEST ADDRESS, NJOBS;
'REAL' BEGTIM;
'INTEGER' 'ARRAY' CODE BUFFER[0:1999];
'INTEGER''ARRAY' SRC TIMING, DST TIMING1, DST TIMING2, DST TIMING3, DST TIMING2A, DST TIMING3A[0:7],U[0:127],Y[0:135],
                 OPTABLE[1:614];
'BOOLEAN' 'ARRAY' ALREADY USED[0:20];

'COMMENT' LOAD TABLES WITH PREINITIALIZED VALUES;

'FOR' Q := 0 'STEP' 1 'UNTIL' 7 'DO' SRC TIMING  [Q] := READ;
'FOR' Q := 0 'STEP' 1 'UNTIL' 7 'DO' DST TIMING1 [Q] := READ;
'FOR' Q := 0 'STEP' 1 'UNTIL' 7 'DO' DST TIMING2 [Q] := READ;
'FOR' Q := 0 'STEP' 1 'UNTIL' 7 'DO' DST TIMING3 [Q] := READ;
'FOR' Q := 0 'STEP' 1 'UNTIL' 7 'DO' DST TIMING2A[Q] := READ;
'FOR' Q := 0 'STEP' 1 'UNTIL' 7 'DO' DST TIMING3A[Q] := READ;
'FOR' Q := 0 'STEP' 1 'UNTIL'135'DO' Y[Q] := READ;
```

```
'FOR' Q := 0 'STEP' 1 'UNTIL'127'DO' U[Q] := READ;
'FOR' Q := 1 'STEP' 1 'UNTIL'614'DO' OPTABLE[Q] := READ;
'FOR' Q:=RESYM 'WHILE' Q< 100 'DO' ;

MAXTEXT := TEXT BUFFER SIZE := 4000;
MAXLEX := 12000;
SYMBOL TABLE SIZE := 3000;
BLOCK SIZE := 2000;
OFFSET := 20000;
IO ADDR := 32768:
MAXCORE := IO ADDR + 20;
NJOBS := 0;
LINE LIMIT := 1000;  INST LIMIT := 20000;




BEGIN ASSEMBLY:
'BEGIN' 'COMMENT' THIS PART IS AN ASSEMBLER FOR THE PDP-11;
'REAL'    CHAR,FIRST,ILC,I,NEXLEX,IDN,HASH,TMP,TMP1,TMP2,TMP3,TMP4,
                         NEXT3,POS,PRIME,PRIMEOP,PREVINDEX,OPCODE,
          EXTRA WORDS,EMPTY,PAD,MAX EXTRAS,TOOBIG,TIM,
          COMMA CTR,NEXTEXT,LAST,REG,MODE,VALUE,INST,DATA1,DATA2,DOT,
          PC,BIT14,BIT7,BIT16,BIT17,BIT19,BIT22,SPACER,MARKER,BIT18,J,K,
          OLDILC,TEM,CPTR,ESCAPE,TEXT PAGES ON DRUM, NEXT PAGE,
          COMMA SYMBOL,SEMI SYMBOL,POINT SYMBOL.ER SYMBOL,SPACE SYMBOL,
          COLON SYMBOL,HATCH SYMBOL,AT SYMBOL,OPEN SYMBOL,CLOSE SYMBOL,
          PLUS SYMBOL,MINUS SYMBOL,AND SYMBOL,OR SYMBOL,PERCENT SYMBOL,
          LF SYMBOL,NC SYMBOL,CRSYMBOL,EQUALSYMBOL,COMMA,POINT,AT,OPEN,
          CLOSE,PLUS,MINUS,AAND,OOR,PERCENT,HATCH,ECTR;
'INTEGER' 'ARRAY' AUX[1:137],TEXT[1:TEXT BUFFER SIZE],SYMBOL TABLE[1:SYMBOL TABLE SIZE],
                  ID[1:7],LEX[1:MAXLEX+2];
'BOOLEAN' ASSIGN,LIVE ONE,ODD,PASS TWO,MIDDLE,END WAS MISSING, INDIRECT,ATPAR,PERCENT SEEN,ASSIGN ERR;
'SWITCH' TYPE:=     CANT,  CANT,  CANT,  CANT,  WORD,
                    BYTE,  ASCII, IVEN,  END,   BINARY,
                    UNARY, NOOP,  TRAP,  BR,    RTS,
                    SOB,   MARK,  MUL,   JSR,   EMTY,
                    P1ERR, EMTY,  PRINT, TITLE, CORE,
                    XOR;
'SWITCH' E:=E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,E12,E13,E14;


        'PROCEDURE' GETID;
       'COMMENT' GETID BUILDS AN IDENTIFIER IN ID[], IT STOPS READING
                 WHEN IT SEES SOMETHING OTHER THAN A LETTER,DIGIT,OR POINT;
       'BEGIN' 'FOR' I:= 2 'STEP' 1 'UNTIL' 6 'DO' ID[I]:= PAD;
               ID[1]:= CHAR;
               IDN:= 2;
               'FOR' CHAR:= NEXTSYM 'WHILE'CHAR = POINTSYMBOL ∨ CHAR < 36 'DO'
               'BEGIN' ID[IDN]:= CHAR; IDN:= IDN+1;
                       'IF' IDN > 6 'THEN' IDN:= 7
               'END';
               IDN:= IDN - 1;
       'END' GETID;



       'PROCEDURE' GETINT;
       'COMMENT' GETINT BUILDS AN INTEGER IN ID[];
       'BEGIN' 'FOR' I:= 2 'STEP' 1 'UNTIL' 6 'DO' ID[I]:= PAD;
```

```
        ID[1]:= CHAR;
        IDN:= 2;
        'FOR' CHAR:= NEXTSYM 'WHILE' CHAR < 10 'DO'
        'BEGIN' ID[IDN]:= CHAR; IDN:= IDN+1;
                'IF' IDN > 7 'THEN' ERR(1)
        'END';
        IDN:= IDN - 1;
        'IF' CHAR=POINT SYMBOL 'THEN' 'BEGIN' CHAR:=NEXTSYM; IDN:=IDN+1; ID[IDN]:=POINTSYMBOL 'END';
'END' GETINT;



'INTEGER' 'PROCEDURE' LOOKUP(VAL);
'COMMENT' THE VALUE OF THE FUNCTION IS THE INDEX OF THE SYMBOL IN
          SYMBOL TABLE. IF THE SYMBOL IS NOT IN THE TABLE, IT IS
          PUT THERE.
          VAL CONTAINS THE 3RD WORD OF THE ENTRY;
'BEGIN' TMP1:= BIT14 * ID[1] + BIT7 * ID[2] + ID[3];
        TMP2:= BIT14 * ID[4] + BIT7 * ID[5] + ID[6];
        TEM:= 1 + REMAINDER(TMP1 + TMP2,PRIME);
        HASH:= AUX[TEM];
        PREVINDEX:= 0;
LINK:   'IF' HASH 'NE' 0 'THEN'
        'BEGIN' 'IF' SYMBOLTABLE[3 * HASH - 2] = TMP1 ^
                SYMBOLTABLE[3 * HASH - 1] = TMP2 'THEN'
                'BEGIN' LOOKUP:= HASH;
                        VAL:= SYMBOLTABLE[3 * HASH]
                'END' 'ELSE'
                'BEGIN' PREVINDEX:= HASH;
                        HASH := SYMBOL TABLE[3*HASH]'DIV'BIT18;
                        'GOTO' LINK
                'END'
        'END' 'ELSE'
        'BEGIN' 'COMMENT' NOT IN TABLE, SO PUT IT THERE;
                SYMBOLTABLE[3 * EMPTY - 2]:= TMP1;
                SYMBOLTABLE[3 * EMPTY - 1]:= TMP2;
                'IF' ID[1] > 9
                'THEN' VAL:= BIT17 'ELSE'
                'BEGIN' TMP4:= 'IF' ID[IDN]=POINT SYMBOL 'THEN' 10 'ELSE' 8;
                        'IF'ID[IDN]=POINTSYMBOL'THEN'IDN:=IDN-1;
                        VAL:= 0;
                        'FOR' I:= 1 'STEP' 1 'UNTIL' IDN 'DO'
                        'BEGIN' 'IF' ID[I] > 7 ^ TMP4 = 8 'THEN' ERR(10);
                                VAL:= TMP4 * VAL + ID[I]
                        'END';
                        'IF' VAL > 65535 'THEN' ERR(13);
                'END';
                SYMBOLTABLE[3 * EMPTY]:= VAL;
                LOOKUP:= EMPTY;
                EMPTY:= EMPTY + 1;
                'IF' 3*EMPTY+1 > SYMBOLTABLESIZE 'THEN' FATAL;
                'IF' PREVINDEX = 0 'THEN' AUX[TEM]:= EMPTY - 1 'ELSE'
                SYMBOLTABLE[3 * PREVINDEX]:= SYMBOLTABLE[3 *
                PREVINDEX] + BIT18 * (EMPTY - 1)  ;
        'END'
'END' LOOKUP;



'PROCEDURE' EAT LABEL;
```

```
       'COMMENT' CHECK TO SEE IF SYMBOL IN ID[1] ... ID[6] IS DEFINED.
                  IF NOT DEFINE IT, IF SO, ERROR MESSAGE MULTIPLY DEFINED
                  SYMBOL;
       'BEGIN' TMP3:= LOOKUP(J);
               'COMMENT' SYMBOL IS UNDEFINED. DEFIRE IT;
               'IF' AND(J,BIT17) ≠ 0 'THEN'
               SYMBOLTABLE[3 * TMP3]:= ILC 'ELSE' ERR(4);
       'END' EAT LABEL;



'PROCEDURE' END OF RUN;
'BEGIN' 'COMMENT' THIS IS CALLED ONLY WHEN ALL JOBS HAVE BEEN FINISHED;
        NEW PAGE;
        PRINTTEXT("END OF RUN.");
        ABSFIXT(15,0,NJOBS);
        PRINTTEXT("JOBS PROCESSED.");
        EXIT
'END' END OF RUN;



       'PROCEDURE' FATAL;
       'BEGIN' 'COMMENT' THIS IS CALLED ONLY WHEN SOME CRITICAL TABLE HAS OVERFLOWED.  THE JOB CANNOT CONTINUE;
               NLCR; PRINTTEXT("ASSEMBLY TERMINATED BY TABLE OVERFLOW.
               THE FOLLOWING SYMBOLS WERE NOT PROCESSED."); NLCR;
              'FOR' I:=RESYMBOL 'WHILE' I≠-4096 'DO' PRSYM(I);
               NEWPAGE;  'FOR' I:=1'STEP'1'UNTIL' 576'DO'PRSYM(66);
               NLCR; NLCR; PRINTTEXT("END OF JOB");
               'GOTO' BEGIN ASSEMBLY;
       'END' FATAL;



       'INTEGER' 'PROCEDURE' NEXTSYM;
       'BEGIN' 'COMMENT' READ THE NEXT CHAR.  CHECK FOR PARITY ERRORS AND ATTEMPTS TO READ ESCAPE CHAR;
       L:      TMP := RESYMBOL;
               'IF'          TMP = -255 'THEN' 'GOTO' L;
               'IF' TMP= ESCAPE 'THEN'
                    'BEGIN' 'IF' MIDDLE 'THEN' 'GOTO' END MISSING
                                        'ELSE' 'GOTO' L
                    'END';
               'IF' TMP<0 ^ TMP≠-4096 'THEN' TMP:=63;
               'IF' TMP ≠ -4096 'THEN'
       'BEGIN'
               'IF' TMP = CR SYMBOL 'THEN' 'GOTO' L;
               'IF' TMP = LF SYMBOL 'THEN' TMP := CR SYMBOL;
               'IF' TMP = NC SYMBOL 'THEN' TMP:= CR SYMBOL;
               NEXTSYM:= TMP;
               NEXT3:= 256 * NEXT3 + TMP;
               POS:= POS + 1;
               'IF' POS = 3 'THEN'
               'BEGIN' TEXT[NEXTEXT]:= NEXT3;
                       POS:= 0;
                       NEXT3:= 0;
                       NEXTEXT:= NEXTEXT + 1;
                       'IF' NEXTEXT > TEXT BUFFER SIZE 'THEN'
                           'BEGIN' TO DRUM(TEXT,TEXT PAGES ON DRUM * TEXT BUFFER SIZE);
                                   TEXT PAGES ON DRUM := TEXT PAGES ON DRUM + 1;
                                   'IF' TEXT PAGES ON DRUM>5'THEN'FATAL;
                                   NEXTEXT:=1;  TEXT[1]:=CR SYMBOL;
                           'END';
               'END'
       'END'
          'ELSE' 'IF' CHAR = CR SYMBOL 'THEN' 'GOTO' PASS2 'ELSE' NEXTSYM := CR SYMBOL;
       'END';



       'PROCEDURE' ERR(K);
       'VALUE' K; 'INTEGER' K;
       'BEGIN' 'COMMENT' IF AN ERROR OCCURS DURING PASS 1, CONTROL PASSES TO HERE.  THE OPCODE IS OVER WRITTEN WITH THE ERROR CODE
               AND THE PARAMETERS SET UP TO SKIP THE REST OF THE LINE;
               'IF' LEX[FIRST+1] ≠ 9*BIT16 'THEN'
            'BEGIN'
               LEX[FIRST+1] := 21*BIT16+K;
               'IF' ASSIGN 'THEN'  'BEGIN' LEX[FIRST+1]:=21*BIT16+12; ASSIGN ERR.:= 'TRUE' 'END';
               NEXLEX := FIRST + 2;
               LEX[NEXLEX]:=0;
               ASSIGN:= 'FALSE'; OPCODE:= 0; MAX EXTRAS:= 3; COMMA CTR:= 0;
               'GOTO' EAT COMMENT;
            'END'
       'END' ERR;



       'PROCEDURE' OPLOOKUP;
       'COMMENT' OPLOOKUP LOOKS UP THE OPCODE IN OPTABLE.  EACH ENTRY IN OPTABLE IS 2 WORDS.  THE FIRST
                CONTAINS 5 CHARACTERS OF THE OPCODE.  THE SECOND HAS 3 FIELDS: BITS 0-15 = VALUE,
                BITS 16-21 = TYPE, BITS 22-23 = MAX EXTRAS;
       'BEGIN' TMP:= 0;
               'FOR' I:= 1 'STEP' 1 'UNTIL' 5 'DO' TMP:= 32 * TMP + ('IF' ID[I]=PAD 'THEN' 31 'ELSE'
                     'IF' ID[I]=POINTSYMBOL 'THEN' 30 'ELSE'ID[I]-9);
               HASH:= TMP - PRIMEOP * (TMP 'DIV' PRIMEOP) + 1;
               TMP4 := HASH;
       LOOP:   'IF' OPTABLE[2 * HASH - 1] = TMP 'THEN'
               'BEGIN' TMP1:= OPTABLE[2 * HASH];
                       OPCODE := AND(TMP1,BIT22-1);
                       LEX[FIRST+1] := OPCODE;
                       OPCODE:= OPCODE 'DIV' BIT16;
                       'IF' OPCODE = 0 'THEN' OPCODE := 0;
                       MAXEXTRAS:= TMP1 'DIV' BIT22;
               'END' 'ELSE'
               'IF' OPTABLE[2 * HASH - 1] = 0 'THEN' ERR(6) 'ELSE'
               'BEGIN' HASH:= HASH + TMP4;
                       'IF' HASH > PRIMEOP 'THEN' HASH:= HASH-PRIMEOP;
                       'GOTO' LOOP
               'END';
               'IF' EVEN(ILC) 'NE' 1 'THEN' ERR(9)
       'END' OPLOOKUP;



'PROCEDURE' OCTAL(N);
'BEGIN' 'COMMENT' PRINT OUT N IN OCTAL AS A 6 OCTIT NUMBER;
        'IF' N<65536 'THEN' PRSYM(AND(N,32768)÷32768) 'ELSE' PRSYM(AND(N,32768)÷32768 + 10);
        PRSYM(AND(N 'DIV' 4096,7));
        PRSYM(AND(N 'DIV'  512,7));
        PRSYM(AND(N 'DIV'   64,7));
        PRSYM(AND(N 'DIV'    8,7));
        PRSYM(AND(N,7));
'END' OCTAL;



'PROCEDURE' LISTING(N);
'VALUE' N; 'INTEGER' N ;
'BEGIN' 'COMMENT' LISTING FORMAT IS COLOMNS 1-20 ERROR FIELD,
    23-28 PROGRAM COUNTER, 31-36 INSTRUCTION, 39-44 FIRST DATA,
    47-52 SECOND DATA WORD. N IS THE NUMBER OF MACHINE
    WORDS ASSEMBLED;
    'IF' N 'NE' 0
    'THEN'
        'BEGIN' SPACE(22);
            OCTAL(ILC); SPACE(2);
            OCTAL(INST); SPACE(2);
            'IF' N > 1 'THEN' OCTAL(DATA 1) 'ELSE' SPACE(6); SPACE(2);
            'IF' N > 2 'THEN' OCTAL(DATA 2) 'ELSE' SPACE(6); SPACE(2);
        'END';
    'COMMENT' NOW PRINT THE ASSOCIATED TEXT;
LOOP: 'IF' POS= 0 'THEN'
           'BEGIN' NEXTEXT := NEXTEXT + 1;
                   'IF' NEXTEXT > TEXT BUFFER SIZE 'THEN'
                        'BEGIN' 'IF' NEXTPAGE>TEXT PAGES ON DRUM 'THEN''BEGIN' PRINTTEXT("NONEXISTENT DRUM PAGE"); FATAL; 'END';
                                FROM DRUM(TEXT,NEXT PAGE*TEXTBUFFER SIZE);
                                NEXT PAGE := NEXT PAGE + 1; NEXTEXT:=1
                        'END';
                   NEXT3 := TEXT[NEXTEXT];  POS := 3;
           'END';
    CHAR:= 'IF' POS= 3 'THEN' NEXT3 'DIV' BIT16  'ELSE' 'IF' POS= 2  'THEN'
                                                                   AND(NEXT3 'DIV' 256,255) 'ELSE' AND(NEXT3, 255);
    POS:= POS-1;
    'IF' CHAR 'NE' CRSYMBOL
       'THEN' 'BEGIN' PRSYM(CHAR); 'GOTO' LOOP 'END'
       'ELSE' 'BEGIN' NLCR; 'IF' OPCODE = 9 'THEN' 'GOTO' EXECUTE'ELSE''GOTO' SETUP 'END';
'END' LISTING;



'PROCEDURE' ERR2(STR);
'BEGIN' 'COMMENT' PROCESS PASS2 ERRORS. NOTE THAT IT IS POSSIBLE
        TO GET HERE ON PASS1, SINCE EXPR IS CALLED TO EVALUATE
ASSIGNMENTS;
        'IF' ¬ PASSTWO 'THEN' ERR(12);
        PRINTTEXT(STR); SPACE(2);
        OCTAL(ILC); SPACE(2);
        OCTAL(0); SPACE(18);
        ECTR := ECTR + 1;
        LISTING(0)
'END' ERR2;



'PROCEDURE' EMIT(LOC,VAL)  ;
'VALUE' LOC,VAL; 'INTEGER' LOC,VAL;
'BEGIN' 'COMMENT' THIS PROCEDURE STORES THE EMITTED BINARY OUTPUT,
        THE ADDRESS OF THE OUTPUT WORD IS LOC, ITS VALUE IS VAL;
        'IF' LOC < 0 ∨ LOC > TOO BIG 'THEN'ERR2("ADDRESS ERROR       ");
        Q := LOC ÷(2*BLOCK SIZE);
        'IF' Q ≠ CURRENT BLOCK 'THEN'
```

```
             'BEGIN''COMMENT' TOSS THIS BLOCK OUT AND READ IN A NEW ONE;
                     'IF' CURRENT BLOCK ≠ -1 'THEN' TO DRUM(CODE BUFFER, OFFSET+BLOCK SIZE * CURRENT BLOCK);
                     'IF' ALREADY USED[Q]
                          'THEN' FROM DRUM(CODE BUFFER,OFFSET + BLOCK SIZE * Q)
                          'ELSE' 'BEGIN' 'FOR' I:=0 'STEP' 1 'UNTIL' BLOCK SIZE - 1 'DO'  CODE BUFFER[I] := 0;
                                         ALREADY USED[Q] := 'TRUE';
                                 'END';
                     CURRENT BLOCK := Q
             'END';
        CODE BUFFER[(LOC÷2)-CURRENT BLOCK * BLOCK SIZE] := VAL;
'END' EMIT;



'PROCEDURE' OPERAND(MODE,REG,VALUE);
'COMMENT' SCAN AND EVALUATE AN OPERAND FIELD;
'INTEGER' MODE,REG,VALUE;
'BEGIN' INDIRECT:='FALSE';
        'IF' LEX[MARKER]= AT 'THEN' 'BEGIN' INDIRECT:= 'TRUE';  MARKER := MARKER + 1; 'END';
        'IF' LEX[MARKER]= HATCH
         'THEN'
            'BEGIN' 'COMMENT' IMMEDIATE OR ABSOLUTE MODE,  #E,  AT # E ;
            MARKER:= MARKER + 1;
            VALUE:= EXPR(MARKER);
            'IF' VALUE 'GE' BIT16 'THEN' ERR2("REGISTER ?          ");
            REG:= 7;
            MODE:= 'IF' INDIRECT 'THEN' 3 'ELSE' 2
            'END'
        'ELSE' 'IF' LEX[MARKER]= MINUS ^ LEX[MARKER +1]= OPEN
               'THEN'
                 'BEGIN'  'COMMENT' MODE -(R)  ,  AT(R) ;
                    MARKER:= MARKER + 2;
                    VALUE:= EXPR(MARKER);
                    'IF' LEX[MARKER-1] ≠ CLOSE 'THEN' ERR2("CLOSE PAREN MISSING ");
                    'IF'VALUE>BIT16+7v(VALUE<BIT16^VALUE>7) 'THEN' ERR2("REGISTER > 7        ");
                    REG:= AND(VALUE, BIT16-1);
                    MODE:= 'IF' INDIRECT 'THEN' 5 'ELSE' 4;
                    MARKER := MARKER + 1;
                 'END'
        'ELSE' 'IF' LEX[MARKER]= OPEN
            'THEN'
               'BEGIN' 'COMMENT' (R) , (R)+ , AT (R) , AT (R)+ ;
                    MARKER:= MARKER + 1;
                    VALUE:= EXPR (MARKER);
                    'IF' LEX[MARKER-1]≠CLOSE 'THEN' ERR2("CLOSE PAREN MISSING ");
                    'IF' LEX[MARKER] = PLUS
                    'THEN'
                      'BEGIN' 'COMMENT' (R)+ , AT (R)+  ;
                        MARKER:= MARKER + 2 ;
                        'IF'VALUE>BIT16+7 v (VALUE<BIT16 ^ VALUE>7)  'THEN' ERR2("REGISTER > 7        ");
                        REG:= AND (VALUE,BIT16-1);
                        MODE:= 'IF' INDIRECT 'THEN' 3 'ELSE' 2;
                      'END'
                    'ELSE'
                    'BEGIN' 'COMMENT' (R) , AT (R) ;
                    MODE:= 'IF' INDIRECT 'THEN' 7 'ELSE' 1;
                    'IF'VALUE>BIT16+7v(VALUE<BIT16^VALUE>7) 'THEN' ERR2("REGISTER > 7        ");
                    REG:= AND(VALUE, BIT16-1);
                    VALUE:= 0;
                    MARKER := MARKER + 1;
                    'END'
               'END'
'ELSE' 'BEGIN' 'COMMENT' E(R), CE(R), E, R;
                VALUE:= EXPR(MARKER);
                'IF'LEX[MARKER-1] = OPEN
                'THEN'
                  'BEGIN'
                    TMP:= EXPR(MARKER);
                    'IF' LEX[MARKER-1] ‡ CLOSE 'THEN' ERR2("CLOSE PAREN MISSING ");
                    MARKER := MARKER + 1;
                    MODE:= 'IF' INDIRECT 'THEN' 7 'ELSE' 6;
                    'IF'TMP>BIT16+7∨(TMP<BIT16∧TMP>7)'THEN'ERR2 ("REGISTER > 7         ");
                    REG:= AND(TMP,BIT16-1);
                    'IF' VALUE 'GE' BIT16'THEN' ERR2("TRUNCATION ERROR    ");
                  'END'
    'ELSE' 'IF' VALUE 'GE' BIT16
            'THEN'
               'BEGIN'
                    MODE := 'IF' INDIRECT 'THEN' 1 'ELSE' 0;
                    REG := -(BIT16-VALUE);
                    'IF' REG > 7 'THEN' ERR2("REGISTER > 7        ");
               'END'
           'ELSE' 'BEGIN' MODE := 'IF' INDIRECT 'THEN' 7 'ELSE' 6;
                        REG:= 7
                    'END'
       'END'
'END' OPERAND;



'INTEGER' 'PROCEDURE' EXPR(MARKER);
'INTEGER' MARKER;
'BEGIN' 'COMMENT' EVALUATE AN EXPR STOP WHEN COMMA, RIGHT
    PAREN, OR NEXT SOURCE TEXT LINE SCANNED;
    'INTEGER' VALUE,CHAR,TMP;
    TMP:= VALUE:= 0;
    'IF' MARKER > LAST 'THEN' ERR2("MISSING OPERAND     ");
    CHAR:= LEX[MARKER];
    'IF' CHAR= PLUS ∨ CHAR= MINUS 'THEN' 'GOTO' TERM;
    'IF' CHAR=PERCENT ∨ CHAR=POINT 'THEN' 'GOTO' SYM;
    'IF' CHAR=COMMA ∨ CHAR=OPEN ∨ CHAR=CLOSE ∨ CHAR<BIT16 'THEN'     ERR2("MISSING OPERAND     ");
    'IF' CHAR=131135 'THEN' ERR2("PARITY ERROR        ");
    'IF' CHAR < BIT17 'THEN' 'GOTO' SYM 'ELSE' ERR2("BAD CHAR IN EXPR    ");
      'COMMENT' IDENTIFIER, PERCENT OR DOT EXPECTED HERE;
TERM:  MARKER := MARKER + 1;
SYM: CHAR:= LEX[MARKER];
    'IF' CHAR=131135 'THEN' ERR2("PARITY ERROR        ");
    'IF' MARKER > LAST 'THEN' ERR2("MISSING OPERAND     ");
    'IF' CHAR= PERCENT 'THEN' 'BEGIN' TMP:= BIT16;     MARKER:= MARKER + 1; CHAR:= LEX[MARKER] 'END';
    'IF' CHAR<BIT16 ∨ (CHAR>BIT17 ∧ CHAR‡POINT ) 'THEN' ERR2("SYNTAX ERROR IN EXPR");
    TMP2:='IF' CHAR=POINT 'THEN' ILC 'ELSE' AND(SYMBOL TABLE[3 * (CHAR-BIT16)], BIT18-1);
    'IF' AND(TMP2, BIT17) 'NE' 0 'THEN' ERR2("UNDEFINED SYMBOL    ");
    'IF' TMP2≥ BIT16 'THEN' 'BEGIN' TMP:= BIT16; TMP2:= TMP2-BIT16 'END';
    TEM:= LEX[MARKER-1];
OP: 'IF' TEM= PLUS 'THEN' VALUE:= VALUE + TMP2
    'ELSE' 'IF' TEM= MINUS 'THEN' VALUE:= VALUE +('IF' TMP2=0 'THEN' 0     'ELSE' 65536 - TMP2)
    'ELSE' 'IF' TEM= AAND 'THEN' VALUE:= AND(VALUE,TMP2)
    'ELSE' 'IF' TEM= OOR 'THEN' VALUE:= OR(VALUE, TMP2)
'ELSE' 'IF' TEM=PERCENT 'THEN' 'BEGIN' TEM:=LEX[MARKER-2];
                                       'IF'TEM=PERCENT'THEN'ERR2("TOO MANY PERCENTS   ")'ELSE' 'GOTO' OP;
```

```
                               'END'
                        'ELSE' 'BEGIN' TEM:= PLUS; 'GOTO' OP 'END';
    'IF' VALUE=0 'THEN' VALUE:=0;
    'IF' VALUE ≥ BIT16 'THEN' VALUE := AND(VALUE,32767);
    MARKER:= MARKER + 1  ;
    CHAR := LEX[MARKER];
    'IF' CHAR≠COMMA ∧ CHAR≠OPEN ∧ CHAR≠CLOSE ∧ CHAR≥ BIT16 'THEN' 'GOTO' TERM;
    EXPR:= VALUE + TMP;
    MARKER:= MARKER + 1;
'END' EXPR;











'COMMENT' INITIALIZE THE ASSEMBLER;
BEGTIM := TIME;
LOWEST ADDRESS := 1000000;  HIGHEST ADDRESS := 0;
TOOBIG:=IOADDR; ECTR := 0;   MIDDLE := 'FALSE';
CURRENT BLOCK := -1;
TEXT PAGES ON DRUM := NEXT PAGE := 0;
END WAS MISSING := 'FALSE';
STARTING ADDRESS := 256;
PRIMEOP := 307;
CHAR:=FIRST:=ILC:=I:=NEXLEX:=IDN:=HASH:=TMP:=TMP1:=TMP2:=TMP3:=TMP4:=NEXT3:=PREVINDEX:=OPCODE:=EXTRAWORDS:=-0;
EMPTY:=MAXEXTRAS:=COMMA CTR:=NEXTEXT:=LAST:=-0;
BIT7:= 128; BIT14:= 16384; BIT16:= 65536;BIT17:=131072; BIT18:=262144; BIT19:=524288; BIT22:=4194304;
ASSIGN:=LIVE ONE := ODD := 'FALSE';
ESCAPE := -27;
ILC := 256; NEXLEX := 1; POS := 0; NEXT3 :=0; PRIME := 137;
EMPTY := 1;
'FOR' I:= 1 'STEP' 1 'UNTIL' PRIME 'DO' AUX[I]:= 0;
'FOR' I:= 1 'STEP' 1 'UNTIL'   SYMBOL TABLE SIZE 'DO'SYMBOL TABLE[I]:=0;
'FOR' I:= 1 'STEP' 1 'UNTIL' MAXLEX 'DO' LEX[I]:= 0;
NEXTEXT:=1;  PAD := 127;   CPTR:=1;
PASSTWO:= 'FALSE';
CR SYMBOL:= 134; LF SYMBOL:= 135; NC SYMBOL:= 119; COMMA SYMBOL:= 87;
SEMI SYMBOL:= 91; POINT SYMBOL:= 88; SPACE SYMBOL:= 93; COLON SYMBOL:= 90;
HATCH SYMBOL:= 125; AT SYMBOL:= 128; OPEN SYMBOL:= 98; CLOSE SYMBOL:=99;
PLUS SYMBOL:= 64; MINUS SYMBOL:= 65; AND SYMBOL:= 123; OR SYMBOL:= 129;
PERCENT SYMBOL:= 132; EQUAL SYMBOL := 70;
COMMA:=COMMASYMBOL+BIT17; POINT:=POINTSYMBOL+BIT17; HATCH:=HATCHSYMBOL+BIT17; AT:=ATSYMBOL+BIT17;
OPEN:=OPENSYMBOL+BIT17; CLOSE:=CLOSESYMBOL+BIT17; PLUS:=PLUSSYMBOL+BIT17; MINUS:=MINUSSYMBOL+BIT17;
AAND:=ANDSYMBOL+BIT17; OOR:=ORSYMBOL+BIT17; PERCENT:=PERCENTSYMBOL+BIT17;
'FOR' Q:=0 'STEP' 1 'UNTIL' 20 'DO' ALREADY USED[Q] := 'FALSE';

'COMMENT' PRE LOAD THE SYMBOLS PRS, PRB, PPS, PPB, TPS, TPB, PIR;
IDN := 3;  ID[4] := ID[5] := ID[6] := PAD;
ID[1] := 25;  ID[2] := 27;  ID[3] := 28;
SYMBOL TABLE[3*LOOKUP(J)] := IOADDR;
ID[3] := 11;  SYMBOL TABLE[3*LOOKUP(J)] := IO ADDR + 2;
ID[2] := 25;  ID[3] := 28;  SYMBOL TABLE[3*LOOKUP(J)] := IO ADDR + 4;
ID[3] := 11;  SYMBOL TABLE[3*LOOKUP(J)] := IO ADDR + 6;
ID[1] := 29;  ID[3] := 28;  SYMBOL TABLE[3*LOOKUP(J)] := IO ADDR + 8;
```

```
ID[3] := 11;  SYMBOL TABLE[3*LOOKUP(J)] := IO ADDR + 10;
ID[1] := 25; ID[2]:=18; ID[3]:=27; SYMBOL TABLE[3*LOOKUP(J)]:=IOADDR+12;

'COMMENT' SKIP FORWARD UNTIL A .TITLE LINE IS SEEN;
'FOR' I := 1 'STEP' 1 'UNTIL' 6 'DO'
      'BEGIN' ID[I] := RESYMBOL;
              'IF' ID[I] = - 4096 'THEN' END OF RUN
      'END';
IDN := 6;
SEEK:  'IF' ID[1]=POINT SYMBOL ^ ID[2]=29 ^ ID[3]=18 ^ ID[4]=29 ^ ID[5]=21 ^ ID[6]=14
              'THEN' 'BEGIN' CHAR := RESYMBOL;
                             'FOR' CHAR := NEXTSYM 'WHILE' CHAR ≠ CR SYMBOL 'DO' ;
                             LEX[1] := ILC;
                             LEX[2] := 24*BIT16;
                             NEXLEX := 3;
                     'END'
              'ELSE' 'BEGIN' 'FOR' I:=1 'STEP' 1 'UNTIL' 5 'DO' ID[I] :=  ID[I+1];
                             ID[6] := RESYMBOL;
                             'IF' ID[6] = -4096 'THEN' END OF RUN 'ELSE' 'GOTO' SEEK
                     'END';






'COMMENT' MAIN LOOP OF PASS1 BEGINS HERE;
'COMMENT' EVERY LINE OF SOURCE TEXT, INLCUDING BLANK LINES, CAUSES ANENTRY OF AT LEAST 2 WORDS TO BE MADE
IN LEX.  THIE FIRST CONTAINS THE ILC, THE SECOND THE OPCODE;
NEWCARD: FIRST:= NEXLEX;
       LEX[FIRST]:=ILC;  LEX[FIRST+1]:=20*BIT16;
       OLDILC := ILC;
       OPCODE:=20;  MAXEXTRAS:=3;
       LIVE ONE:= ASSIGN:= ATPAR := ASSIGN ERR := PERCENT SEEN:='FALSE';
       NEXLEX := FIRST + 2; 'IF' NEXLEX > MAXLEX 'THEN' FATAL;
       COMMA CTR:= 0;
       EXTRA WORDS:= 0;
PASS1: CHAR:= NEXTSYM;
       'IF' CHAR = SPACE SYMBOL 'THEN' 'GOTO' PASS1;
       'IF' CHAR = CR SYMBOL 'THEN' 'GOTO' CARET;
       'IF' CHAR = SEMI SYMBOL 'THEN' 'GOTO' EAT COMMENT;
       'IF' CHAR 'NE' POINT SYMBOL ^ (CHAR < 10 v CHAR > 35) 'THEN' ERR(1);
       'COMMENT' MUST BE LABEL OR OPCODE;
       GET ID;
       'COMMENT' CHAR NOW CONTAINS FIRST CHARACTER AFTER ID;

SKIP:  'IF' CHAR = SPACE SYMBOL 'THEN' 'BEGIN' CHAR:= NEXTSYM; 'GOTO'  SKIP 'END';
       'COMMENT' COLON MEANS THE IDENTIFIER WAS A LABEL;. .
       'IF' CHAR = COLON SYMBOL
           'THEN' 'BEGIN' EAT LABEL; 'GOTO' PASS1 'END'
           'ELSE' 'IF' CHAR = EQUAL SYMBOL 'THEN'
           'BEGIN' ASSIGN:= 'TRUE';
                                CHAR := NEXTSYM;
               'IF' ID[1] = POINT SYMBOL ^ IDN = 1
               'THEN' LEX[FIRST+1]:=BIT16*22
               'ELSE' 'BEGIN' LEX[FIRST+1] := BIT16*22 + LOOKUP(TMP);
                                'IF' AND(TMP,BIT17)  =   0 'THEN' ERR(5);
                      'END';
           'END'
           'ELSE' OPLOOKUP;
           'IF' OPCODE = 24 'THEN' 'GOTO' EAT COMMENT;

           'COMMENT' PROCESS THE OPERANDS;

GOPER:     'IF' CHAR = SPACE SYMBOL 'THEN' 'BEGIN' CHAR:=NEXTSYM; 'GOTO' GOPER 'END';
           'IF' CHAR = CR SYMBOL 'THEN' 'GOTO' CARET;
       'COMMENT' CHECK TO SEE IF THIS IS THE .ASCII PSEUDO OP.  IF SO THE DATA CHARACTERS MUST BE PACKED ;
           'IF' OPCODE = 7 'THEN'
       'BEGIN' TEM := CHAR;
           'FOR' CHAR:=NEXTSYM 'WHILE' CHAR ≠ CR SYMBOL 'DO'
              'BEGIN' TMP4 := Y[CHAR];
               'IF' CHAR = TEM
                   'THEN' 'BEGIN' EXTRAWORDS :=(EXTRAWORDS+1)'DIV' 2;
                                  NEXLEX := FIRST + 2 + EXTRA WORDS;
                                  'GOTO' EAT COMMENT;
                          'END'
                   'ELSE' 'BEGIN' TMP1:= FIRST + 2 + EXTRA WORDS 'DIV'2;
                                  LEX[TMP1]:='IF'EVEN(EXTRAWORDS)=1
                                       'THEN'23*BIT16+TMP4'ELSE'LEX[TMP1]+256*TMP4;
                                  EXTRA WORDS := EXTRA WORDS + 1;
                          'END';
              'END';
           ERR(8)
       'END';

               'IF' CHAR = SEMI SYMBOL 'THEN' 'GOTO' EAT COMMENT;
               'IF' CHAR < 10
                           'THEN' 'BEGIN' GET INT;
                                      LEX[NEXLEX]:= BIT16 + LOOKUP(TMP);
                                      NEXLEX:= NEXLEX + 1;
                                      'IF' NEXLEX > MAXLEX 'THEN' FATAL;
                                      LIVE ONE := 'TRUE';
                                  'END'
                           'ELSE'
                                  'IF' CHAR < 36 'THEN'
                                  'BEGIN' GETID;
                                      LEX[NEXLEX]:= BIT16 + LOOKUP(TMP);
                                      'IF' AND(TMP,BIT16)= 0 'THEN'
                                      LIVE ONE:= 'TRUE';
                                      NEXLEX:= NEXLEX + 1;
                                      'IF' NEXLEX > MAXLEX 'THEN' FATAL;
                                  'END'
                           'ELSE'
                                  'BEGIN' 'COMMENT' IT IS ONE OF THE  SPECIAL CHARACTERS;
                                      'IF'CHAR=133'THEN'CHAR:=AT SYMBOL'ELSE''IF'CHAR=80'THEN'CHAR:=ANDSYMBOL
                                            'ELSE' 'IF' CHAR=79  'THEN' CHAR:=OR SYMBOL;
                                      'IF' CHAR=PERCENT SYMBOL  'THEN' PERCENT SEEN := 'TRUE';
                                      LEX[NEXLEX]:= BIT17 + CHAR;
                                      'IF' CHAR = COMMA SYMBOL 'THEN'
                                      'BEGIN' 'IF' PERCENT SEEN 'THEN' LIVE ONE := 'FALSE';
                                              'IF' LIVE ONE 'THEN'EXTRA WORDS := EXTRA WORDS + 1;
                                              LIVE ONE:= 'FALSE';
                                              COMMA CTR:= COMMA CTR + 1    ;
                                              PERCENT SEEN := 'FALSE';
                                              'IF'LEX[NEXLEX-1]=CLOSE ^ ATPAR 'THEN' EXTRA WORDS := EXTRA WORDS + 1;
                                              ATPAR := 'FALSE';
                                      'END';
                                      'IF' CHAR=CLOSE SYMBOL 'THEN' LIVE ONE := 'FALSE';
                                      NEXLEX:= NEXLEX + 1;
                                      'IF' CHAR= POINT SYMBOL 'THEN' LIVE ONE := 'TRUE';
                                      'IF' CHAR=OPEN SYMBOL ^ LIVE ONE 'THEN' EXTRA WORDS := EXTRA WORDS + 1;
```

```
                                            'IF' CHAR = OPEN SYMBOL ∧LEX[NEXLEX-2]=AT 'THEN' ATPAR := 'TRUE';
                                            'IF' NEXLEX > MAXLEX 'THEN' FATAL;
                                            CHAR:= NEXTSYM;
                                    'END';
                                    'GOTO' GOPER;
EAT COMMENT: 'IF' CHAR ≠ CR SYMBOL'THEN' 'BEGIN' CHAR:=NEXTSYM; 'GOTO'EAT COMMENT 'END';



CARET:  'IF' ASSIGN
        'THEN'
           'BEGIN' MARKER := FIRST + 2;
                   LEX[NEXLEX] := 0;
                   LAST := NEXLEX - 1;
                   TMP := EXPR(MARKER);
                   TMP1 := AND(LEX[FIRST+1],BIT16-1);
                   'IF' TMP1 = 0 'THEN' ILC := TMP 'ELSE' SYMBOL TABLE[3*TMP1] := TMP;
                   'IF' TMP1=0 ∧ TMP ≥BIT16 'THEN' ERR(12);
                   NEXLEX := FIRST + 2 ;  LEX[NEXLEX]:=0;
           'END'
        'ELSE'
           'IF' MAXEXTRAS ≠ 3 'THEN'
                                          'BEGIN' 'IF' PERCENT SEEN 'THEN' LIVE ONE := 'FALSE';
                                                  'IF' LIVEONE 'THEN' EXTRAWORDS:=EXTRAWORDS+1;
                                                  'IF'LEX[NEXLEX-1]=CLOSE ∧ ATPAR 'THEN' EXTRA WORDS := EXTRA WORDS +1;
                                                  TMP:='IF' COMMA CTR>0'THEN'COMMACTR+1 'ELSE''IF'NEXLEX=FIRST+2'THEN'0'ELSE'1;
                                                  'IF' TMP > MAXEXTRAS'THEN' ERR(7);
                                                  'IF' TMP < MAXEXTRAS 'THEN' ERR(14);
                                                  'IF'OPCODE≠24'THEN'ILC := ILC + 2 + 2*EXTRAWORDS;
                                                  'IF'OPCODE=9'THEN' 'GOTO' PASS2;
                                          'END'
        'ELSE'
           'BEGIN' 'IF' OPCODE = 5 'THEN' ILC := ILC + 2 * (COMMA CTR+1)
                   'ELSE' 'IF' OPCODE = 6 'THEN' ILC := ILC + COMMACTR+1
                   'ELSE' 'IF' OPCODE = 7 'THEN' ILC := ILC + EXTRAWORDS
                   'ELSE' 'IF' OPCODE = 0 'THEN' ILC := OLDILC+2
                   'ELSE' 'IF' OPCODE =23 'THEN' ILC := ILC + 2*(COMMA CTR + 3)
                   'ELSE' 'IF' OPCODE = 8 ∧ EVEN(ILC) 'NE' 1 'THEN' ILC:= ILC + 1
           'END';
        'IF' 2*(ILC'DIV'2) ≠ ILC 'THEN' ILC := ILC + 1;
        'IF' ASSIGN ERR 'THEN' ILC := ILC - 2;
        MIDDLE := 'TRUE';
        'IF' OPCODE>11 ∧ OPCODE<18 'THEN' ILC := OLDILC + 2;
        'IF' ILC < LOWEST ADDRESS 'THEN' LOWEST ADDRESS := ILC;
        'IF' ILC > HIGHEST ADDRESS 'THEN' HIGHEST ADDRESS := ILC;
        'GOTO' NEWCARD;


END MISSING:  END WAS MISSING := 'TRUE'; LEX[FIRST+1] := 9*BIT16; 'GOTO' PASS2;
```

```
'COMMENT' INITIALIZE SECOND PASS OF THE ASSEMBLER;
PASS2:
        'IF' POS=1 'THEN' TEXT[NEXTEXT]:=BIT16*NEXT3+257*CRSYMBOL
        'ELSE' 'IF' POS=2 'THEN' TEXT[NEXTEXT] := 256*NEXT3 + CRSYMBOL
        'ELSE' TEXT[NEXTEXT]  := (BIT16+256+1) * CR SYMBOL;
       NEXTEXT:= POS:= LAST:= 0; PASSTWO:= 'TRUE';
    PRINTTEXT("   ERROR MESSAGES    ADDRESS    INST   IMMED1  IMMED2    SOURCE STATEMENT"); NLCR; NLCR;
    'COMMENT' SEE IF THE DRUM WAS USED FOR STORING TEXT.  IF SO, WRITETHE LAST BUFFER LOAD OUT, AND BRING IN THE FIRST;
    'IF' TEXT PAGES ON DRUM ≠ 0 'THEN'
           'BEGIN' TO DRUM(TEXT,TEXT BUFFER SIZE*TEXT PAGES ON DRUM);
                   FROM DRUM(TEXT,0);
                   TEXT PAGES ON DRUM := TEXT PAGES ON DRUM + 1
           'END';
    'FOR' I:= 0'STEP' 1 'UNTIL' BLOCK SIZE - 1 'DO' CODE BUFFER[I]:=0;
    'COMMENT' SEE IF DRUM WILL BE NEEDED FOR OBJECT CODE.  IF SO, ZERO THE NEEDED PAGES;
    'IF' HIGHEST ADDRESS ≥BLOCK SIZE 'THEN'
         'BEGIN' 'FOR' I:= 0 'STEP' 1 'UNTIL' 8 'DO' TO DRUM(CODE BUFFER,I*BLOCK SIZE+OFFSET);
         'END';

'COMMENT' MAIN LOOP OF PASS TWO;

SETUP: FIRST:= LAST + 1;
    LAST:= FIRST + 2;
    'FOR' I:=-1 'WHILE' LEX[LAST] 'DIV' BIT16 'NE' 0 'DO' LAST:= LAST + 1;
    LAST:= LAST -1;
    ILC:= LEX[FIRST];
    MARKER:= FIRST + 2;
    OPCODE:= LEX[FIRST + 1] 'DIV' BIT16;
    'IF' OPCODE = 0 'THEN' OPCODE := 0;
    INST := -(BIT16*OPCODE - LEX[FIRST+1]);
    'GOTO' TYPE[OPCODE];




CANT: PRINTTEXT("CANT GET HERE.  THIS MESSAGE WILL NEVER BE PRINTED");
LISTING(0);

WORD: TIM:= ILC;
    INST:= EXPR(MARKER);
    EMIT(TIM,INST);
    'IF' LEX[MARKER-1] < BIT16 'THEN' LISTING(1);
    TIM:= TIM + 2;
    DATA1:= EXPR(MARKER);
    EMIT(TIM,DATA1);
    'IF' LEX[MARKER-1]< BIT16 'THEN' LISTING(2);
    TIM:= TIM + 2;
    DATA2:= EXPR(MARKER);
    EMIT(TIM,DATA2);
WOOP: 'IF' LEX[MARKER-1] < BIT16 'THEN' LISTING(3);
    TIM:= TIM + 2;
    TMP1:= EXPR(MARKER);
    EMIT(TIM, TMP1);
    'GOTO' WOOP;


BYTE: 'FOR' I:= 1 'STEP' 1 'UNTIL' 6 'DO' ID[I]:= 0;
    IDN := 0;
    'FOR' IDN := IDN + 1 'WHILE' MARKER ≤ LAST ∧ IDN ≤ 6 'DO'
        'BEGIN' ID[IDN] := EXPR(MARKER);
```

```
                'IF' ID[IDN]>BIT16'THEN' ERR2("REGISTER NOT ALLOWED");
                    'IF' ID[IDN]>255 'THEN'ERR2("TRUNCATION ERROR    ");
        'END';
    IDN := IDN - 1;
    'IF' IDN = 0 'THEN' ERR2("ADDRESS FIELD EMPTY ");
    INST:= 256 * ID[2] + ID[1];
    DATA1:= 256 * ID[4] + ID[3];
    DATA2:=256 * ID[6] + ID[5];
    'IF' IDN 'GE' 1 'THEN' EMIT(ILC,INST);
    'IF' IDN 'GE' 3 'THEN' EMIT(ILC + 2, DATA1);
    'IF' IDN 'GE' 5 'THEN' EMIT (ILC + 4, DATA2);
    I := ILC + 4;
    'FOR' I:=I+2 'WHILE' MARKER ≤ LAST 'DO'
    'BEGIN'    TMP1:=TMP2:= 0  ;
               TMP1 := EXPR(MARKER);
               'IF' MARKER ≤ LAST 'THEN' TMP2:=EXPR(MARKER);
               EMIT(I,256*TMP2+TMP1);
    'END';
    LISTING('IF' IDN>4'THEN'3 'ELSE' IDN+1 ;2);


ASCII: IDN:= LAST - FIRST-1;
    INST:= LEX[FIRST + 2] - 23 * BIT16;
    DATA1:= LEX[FIRST + 3] - 23 * BIT16;
    DATA2:= LEX[FIRST + 4] - 23 * BIT16;
    'IF' IDN 'GE' 1 'THEN' EMIT(ILC,INST);
    'IF' IDN 'GE' 2 'THEN' EMIT(ILC + 2, DATA1);
    'IF' IDN 'GE' 3 'THEN' EMIT(ILC + 4, DATA2);
    'IF' IDN 'LE' 3 'THEN' LISTING (IDN);
    'FOR' I:= FIRST + 5 'STEP' 1 'UNTIL' LAST 'DO'
    EMIT(ILC + I - FIRST, LEX[I] - 23 * BIT16);
    LISTING(3);


IVEN: LISTING(0);


END: 'IF' END WAS MISSING 'THEN'
         'BEGIN' PRINTTEXT(".END WAS MISSING      "); OCTAL(256);
                 SPACE(36);  PRINTTEXT(".END 400"); NLCR;
                 ECTR := ECTR + 1;
                 'GOTO' EXECUTE;
         'END';
     VALUE:=EXPR(MARKER);  'IF' VALUE ≥TOOBIG'THEN' 'BEGIN' VALUE:=256; ERR2("ERR IN START ADDR   ") 'END';
    SPACE(22); OCTAL(VALUE); SPACE(26);
    STARTING ADDRESS := VALUE;
    LISTING(0);

BINARY: OPERAND(MODE,REG,VALUE);
    IDN:= 1;
    INST:= INST + 512 * MODE + 64 * REG;
    'IF' (MODE > 5) ∨ (MODE= 2 ∧ REG= 7) ∨ (MODE=3 ∧ REG=7) 'THEN'
    'BEGIN' IDN:= IDN + 1;
        DATA1:= 'IF' MODE > 5 ∧ REG= 7 'THEN' VALUE-ILC-IDN-IDN 'ELSE'     VALUE;
        'IF' DATA1 < 0 'THEN' DATA1:=65536+DATA1 'ELSE' 'IF' DATA1=0 'THEN' DATA1 := 0;
    'END';
   OPERAND(MODE,REG,VALUE);
    INST:= INST + 8 * MODE + REG;
    'IF' (MODE > 5) ∨ (MODE= 2 ∧ REG= 7) ∨ (MODE= 3 ∧ REG= 7) 'THEN'
    'BEGIN' IDN:= IDN + 1;
```

```
        DATA2:= 'IF' MODE > 5 ∧ REG= 7 'THEN' VALUE-ILC-IDN-IDN 'ELSE'     VALUE;
        'IF' DATA2 < 0 'THEN' DATA2:=65536+DATA2 'ELSE' 'IF' DATA2=0 'THEN' DATA2 := 0;
        'IF' IDN= 2 'THEN' DATA1:= DATA2
    'END';
    EMIT(ILC,INST);
    'IF' IDN > 1 'THEN' EMIT(ILC + 2, DATA 1);
    'IF' IDN > 2 'THEN' EMIT(ILC + 4, DATA2);
    LISTING(IDN);


UNARY:  OPERAND(MODE,REG,VALUE);
    IDN:= 1;
    INST:= INST + 8 * MODE + REG;
    'IF' (MODE > 5) ∨ (MODE= 2 ∧ REG= 7) ∨ (MODE= 3 ∧ REG= 7) 'THEN'
    'BEGIN' IDN:= IDN + 1;
        DATA1:= 'IF' MODE > 5 ∧ REG= 7 'THEN' VALUE - ILC-IDN-IDN      'ELSE' VALUE;
        'IF' DATA1 < 0 'THEN' DATA1:=65536+DATA1 'ELSE' 'IF' DATA1=0 'THEN' DATA1 := 0;
    'END';
    EMIT(ILC,INST);
    'IF' IDN > 1 'THEN' EMIT(ILC + 2, DATA1);
    LISTING(IDN);


NOOP: EMIT(ILC,INST);
    LISTING(1);


TRAP: TMP:= EXPR(MARKER);
    'IF' TMP 'GE' BIT16 'THEN' ERR2("REG NOT ALLOWED     ");
    'IF' TMP ≥ 256 'THEN' ERR2("TRUNCATION ERROR    ");
    INST:= INST + TMP;
    EMIT(ILC,INST)  ;
    LISTING(1);


BR: TMP:= EXPR(MARKER);
    'IF' TMP≥ BIT16 'THEN' ERR2("BR TO REG ILLEGAL   ");
    'IF' EVEN(TMP) 'NE' 1 'THEN' ERR2("BR TO ODD ADDRESS   ");
    TMP1:= (TMP-ILC-2) 'DIV' 2;
    'IF' TMP1 > 127 ∨ TMP1 < -128 'THEN' ERR2("TOO FAR              ");
    INST:= INST +('IF' TMP1 'GE' 0 'THEN' TMP1 'ELSE' 256 + TMP1);
    EMIT(ILC,INST);
    LISTING(1);


RTS: TMP:= EXPR(MARKER);
    'IF' TMP > BIT16 'THEN' TMP:= TMP-BIT16;
    'IF' TMP > 7 'THEN' ERR2("REGISTER > 7        ");
    INST:= INST + TMP;
    EMIT(ILC,INST);
    LISTING(1);


MUL:  OPERAND(MODE,REG,VALUE);
    'IF' LEX[MARKER] = OPEN 'THEN' ERR2("FIELD 2 MUST BE REG ");
    TMP1:= EXPR(MARKER);
    'IF' TMP1 < BIT16 'THEN' ERR2("FIELD 2 MUST BE REG ");
MULL: TMP1 := TMP1 - BIT16;
    'IF' TMP1 > 7 'THEN' ERR2("REGISTER > 7        ");
    INST:= INST + 64 * TMP1 + 8 * MODE + REG;
    EMIT(ILC,INST);
    'IF' MODE > 5 ∨ (MODE= 2 ∧ REG= 7) ∨ (MODE= 3 ∧ REG= 7)
    'THEN' 'BEGIN' DATA1:= 'IF' MODE > 5 ∧ REG= 7 'THEN' VALUE-ILC-4     'ELSE' VALUE;
                   EMIT(ILC+2,DATA1);
                   'IF' DATA1 < 0 'THEN' DATA1:=65536+DATA1 'ELSE'  'IF' DATA1=0 'THEN' DATA1 := 0;
                   LISTING(2)
           'END'
    'ELSE' LISTING(1);


MARK: TMP:= EXPR(MARKER);
    'IF' TMP > 63 'THEN' ERR2("NN MUST BE 1-63     ");
    INST:= INST + TMP;
    EMIT(ILC,INST);
    LISTING(1);


SOB:  K := EXPR(MARKER);
    'IF'  K  < BIT16 'THEN' ERR2("FIELD 1 MUST BE REG ");
    'IF'  K  > BIT16 + 7 'THEN' ERR2("REGISTER > 7        ");
    TMP1 := EXPR(MARKER);
    TMP2:= -(TMP1-ILC-2) 'DIV' 2;
    'IF' TMP2 < 0 ∨ TMP2 > 63 'THEN' ERR2("TOO FAR             ");
    INST := INST + 64 * (K-BIT16) + TMP2;
    EMIT(ILC,INST);
    LISTING(1);


'COMMENT' JSR PC, SUBR;
JSR: OPERAND(MODE,REG,K);
    'IF' K ≥ BIT16 'THEN' K := K-BIT16;
    'IF' MODE ≠ 0 ∨ K>7 'THEN' ERR2("FIELD 1 MUST BE REG ");
    OPERAND(MODE,REG,VALUE);
    'IF' MODE = 0 'THEN' ERR2("JSR TO REG ILLEGAL  ");
    IDN := 1;
    'IF' (MODE>5) ∨ (MODE=2∧REG=7) ∨ (MODE=3∧REG=7) 'THEN'
       'BEGIN' IDN := IDN + 1;
               DATA1 := 'IF' MODE > 5 ∧ REG=7 'THEN' VALUE-ILC-4 'ELSE'VALUE;
               'IF' DATA1<0 'THEN' DATA1:=65536+DATA1 'ELSE' 'IF' DATA1 = 0 'THEN' DATA1 :=0;
       'END';
    INST := INST + 64*K + 8*MODE + REG;
    EMIT(ILC,INST);
    'IF' IDN > 1 'THEN' EMIT(ILC+2,DATA1);
    LISTING(IDN);


TITLE:
EMTY: SPACE(54); LISTING(0);


P1ERR: 'GOTO' E[INST];
E1:    ERR2( "FIRST CHAR OF LABEL ");
E2:    ERR2( "TAG FOLLOWED BY ?   ");
E3:    ERR2( "UNDEFINED SYM ON RHS");
E4:    ERR2( "MULTIPLY DEFINED SYM");
E5:    ERR2( "LHS ALREADY DEFINED ");
E6:    ERR2( "OPCODE UNKNOWN      ");
E7:    ERR2( "TOO MANY FIELDS     ");
E8:    ERR2( "NO CLOSING DELIMITER");
E9:    ERR2( "ILC ODD             ");
```

```
E10:   ERR2( "8 OR 9 IN OCTAL NUM ");
E11:   ERR2( "TOO MANY DIGITS     ");
E12:   PRINTTEXT("ERROR IN RHS OF EXPR"); SPACE(34); ECTR := ECTR + 1;   LISTING(0);
E13:   ERR2( "OVERFLOW            ");
E14:   ERR2( "TOO FEW FIELDS      ");


PRINT: DATA1 := 1;
    DATA2 := EXPR(MARKER);
    'IF' DATA2 ≥ BIT16 'THEN' ERR2("REG NOT ALLOWED     ");
    EMIT(ILC,INST);
    EMIT(ILC+4,DATA2);
POOP:'IF' LEX[MARKER-1] < BIT16 'THEN' 'BEGIN' EMIT(ILC+2,DATA1); LISTING(3) 'END';
    DATA1 := DATA1 + 1;
    K := EXPR(MARKER);
    'IF' K≥BIT16 'THEN' ERR2("REG NOT ALLOWED     ");
    EMIT(ILC+2*DATA1+2,K);
    'GOTO' POOP;


CORE: OPERAND(MODE,REG,DATA1);
    'IF' MODE ≠6 ∨ REG≠7 'THEN' ERR2("ADDRESS EXPECTED    ");
    OPERAND(MODE,REG,DATA2);
    'IF' MODE ≠6 ∨ REG≠7 'THEN' ERR2("ADDRESS EXPECTED    ");
    'IF' DATA1 > TOOBIG 'THEN' ERR2("FIRST FIELD WRONG   ");
    'IF' DATA2 > TOOBIG 'THEN' ERR2("SECOND FIELD WRONG  ");
    EMIT(ILC,INST);
    EMIT(ILC+2,DATA1);
    EMIT(ILC+4,DATA2);
    LISTING(3);


XOR: 'IF' LEX[MARKER] = OPEN 'THEN' ERR2("FIELD 1 MUST BE REG ");
    TMP1 := EXPR(MARKER);
    'IF' TMP1 < BIT16 'THEN' ERR2("FIELD 1 MUST BE REG ");
    OPERAND(MODE,REG,VALUE);
    'GOTO' MULL;


EXECUTE:
       NLCR; NLCR;  ABSFIXT(4,0,ECTR); PRINTTEXT("ERRORS IN ABOVE ASSEMBLY       ");
    ABSFIXT(8,0,TIME-BEGTIM); PRINTTEXT("SECONDS ASSEMBLY TIME ");
'END';






'BEGIN' 'COMMENT' THIS PART IS AN INTERPRETER FOR THE PDP-11 HARDWARE;
'REAL'     BINWIDTH, BIT16, BYTE, C, CPU PRIORITY, DST, DST ADDR, DST INDEX, DST MODE, DST OPERAND, DST REG, DONEBIT,ENABLE,FREELISTHEAD,
           II, III, INCR, INST TIME, INSTRUCTION, TYPED,   J, K, L, LINECOUNT, LOWBIN,          MAXPOS, MAXWORD, N, NEWLINE SYMBOL, I,
           NUM INTERRUPT SLOTS, OLD PC, OPCODE, ORIG SRC REG VAL, ORIG DST REG VAL, PC, PIR, PIRWORD, POINTER, PRI, PRI INDEX, PRB, PRS,
           PPS, PSW, QUEUE NUMBER, SRC, SRC ADDR, SRC INDEX, SRC MODE, SRC OPERAND, SRC REG, SP, SPACER, STACKLIMIT, T, TMP, TMP1, TMP2,
           TPB, TRACEGROUP, V, VEC, WHOLE DST, Z, ZERO, PPB, TPS, INST,  TMP3, INST COUNT,              TYLIN, CHAR RD,CHARPUN,
           NMODE0,NMODE1,NMODE2,NMODE3,NMODE4,NMODE5,NMODE6,NMODE7,NIMMED, NDIR, NPIC, NINTER, NADC, NADCB, NADD, NASL, NASLB,NASH,NASHC,
           NASR, NASRB, NBCC, NBCS, NBEQ, NBGE, NBGT, NBHI, NBIC, NBICB,NBIS, NBISB, NBIT, NBITB, NBLT, NBLE, NBLOS, NBMI,NBNE,NBPL,NBPT,
           NBR, NBVC, NBVS, NCLR, NCLRB, NCMP, NCMPB, NCOM, NCOMB, NCC, NDEC, NDECB, NDIV, NEMT, NHALT, NINC, NINCB, NIOT, NJMP, NJSR,
           NMOV, NMOVB, NMUL, NNEG, NNEGB, NRESET, NROL, NROLB, NROR, NRORB, NRTI, NRTS, NRTT, NSBC, NSBCB, NSOB, NSPL, NSUB, NSWAB, NSXT,
           NTRAP, NTST, NTSTB, NWAIT, NXOR, NMARK,Q1,Q2;
'REAL' BEGIN TIME, CLOCK, END TIME, LONG WAIT, NEXT INTERRUPT TIME, IDLE, E, MUL1, MUL2, MUL3, TIMX, X;
```

```
'BOOLEAN' DUPPER, POS DST, POS SRC, SUPPER, TRACE FLAG,LASTCHARWAS119;
'INTEGER' 'ARRAY'M[-8:MAXCORE÷2],
            INTERRUPT VECTOR[1:16], NEXT ON CHAIN[1:16], SPY[-1:128], TRINDEX[1:63], TRLIST[1:256], FIRST SLOT[1:7];
'REAL' 'ARRAY' INTERRUPT TIME[0:16];
'INTEGER' PRS2, PRB2, PPS2, PPB2, TPS2, TPB2, PIR2;
'SWITCH' INSTRUCTION TYPE := OP IS 0, MOV, CMP, BIT, BIC, BIS, ADD, OP IS 7, OP IS 8, MOVB, CMPB, BITB, BICB, BISB, SUB,FPT;
'SWITCH' OP 0 SPLIT := MIXED, BR1, BR2, BR3, JSR, ONE OPIES1, ONE OPIES2, ILLEGAL;
'SWITCH' MIXEDSPLIT := NOAD, JMP, MIXED1, SWAB, BR, BR, BR, BR;
'SWITCH' NOADSPLIT := HALT, WAIT, RTI, BPT, IOT, RESET, RTT;
'SWITCH' MIXED1SPLIT := RTS, ILLEGAL, ILLEGAL, SPL, CLRCC, CLRCC, SETCC, SETCC;
'SWITCH' ONE OPIES1 SPLIT := CLR, COM, INC, DEC, NEG, ADC, SBC, TST;
'SWITCH' ONE OPIES2 SPLIT := ROR, ROL, ASR, ASL, MARK, ILLEGAL, ILLEGAL, SXT;
'SWITCH' OP 7 SPLIT := MUL, DIV, ASH, ASHC, EXOR, ILLEGAL, ILLEGAL, SOB;
'SWITCH' OP 8 SPLIT := BR4, BR5, BR6, BR7, EMTTRAP, ONE OPIES3, ONE OPIES4, ILLEGAL;
'SWITCH' ONE OPIES3 SPLIT := CLRB, COMB, INCB, DECB, NEGB, ADCB, SBCB, TSTB;
'SWITCH' ONE OPIES4 SPLIT := RORB, ROLB, ASRB, ASLB, ILLEGAL, ILLEGAL, ILLEGAL, ILLEGAL;



'PROCEDURE' EVAL BOTH OPERANDS;
'COMMENT' BOTH OPERANDS ARE EVALUATED.  THE SOURCE ADDRESS, SOURCEOPERAND, DESTINATION ADDRESS AND DESTINATION OPERAND ARE
STORED IN SRC ADDR, SRC OPERAND, DST ADDR, AND DST OPERAND RESPECTIVELY.  SUPPER (DUPPER) IS TRUE IFF THE SOURCE (DESTINATION)
OPERAND IS AN ODD NUMBERED BYTE;
'BEGIN' SRC INDEX := -SRC REG - 1;
        ORIG SRC REG VAL := M[SRC INDEX];
        DST INDEX := -DST REG - 1;
        ORIG DST REG VAL := M[DST INDEX];
        SRC ADDR := OPERAND ADDRESS(SRC INDEX,SRC MODE,ORIG SRC REGVAL);
        'IF' DST INDEX = -8 'THEN' ORIG DST REG VAL := M[-8];
        DST ADDR := OPERAND ADDRESS(DST INDEX,DST MODE,ORIG DST REGVAL);
        'IF' SRC ADDR > MAXCORE 'THEN' ERROR("SOURCE ADDRESS OUT OF BOUNDS");
        'IF' DST ADDR > MAXCORE 'THEN' ERROR("DESTINATION ADDRESS OUT OF BOUNDS");
        SUPPER := 'IF' SRC ADDR > 0 ^ EVEN(SRC ADDR) ≠ 1 'THEN' 'TRUE' 'ELSE' 'FALSE';
        DUPPER := 'IF' DST ADDR > 0 ^ EVEN(DST ADDR) ≠ 1 'THEN' 'TRUE' 'ELSE' 'FALSE';
        'IF' OPCODE < 8 ^ SRC ADDR > 0 ^ SUPPER 'THEN' ERROR("SOURCE ADDRESS ODD");
        'IF' OPCODE < 8 ^ DST ADDR > 0 ^ DUPPER 'THEN' ERROR("DESTINATION ADDRESS ODD");
        SRC := 'IF' SRC ADDR < 0 'THEN' SRC ADDR 'ELSE' SRC ADDR ÷ 2;
        DST := 'IF' DST ADDR < 0 'THEN' DST ADDR 'ELSE' DST ADDR ÷ 2;
        WHOLE DST := M[DST];
        SRC OPERAND := 'IF' OPCODE < 8 'THEN' M[SRC] 'ELSE' 'IF' SUPPER'THEN' M[SRC] ÷ 256 'ELSE' AND(M[SRC],255);
        DST OPERAND := 'IF' OPCODE < 8 'THEN' M[DST] 'ELSE' 'IF' DUPPER'THEN' M[DST] ÷ 256 'ELSE' AND(M[DST],255);
'END' EVAL BOTH OPERANDS;



'PROCEDURE' EVAL ONE OPERAND;
'COMMENT' SAME AS EVAL BOTH OPERANDS BUT FOR DESTINATION ONLY;
'BEGIN' DST INDEX := -DST REG - 1;
        ORIG DST REG VAL := M[DST INDEX];
        DST ADDR := OPERAND ADDRESS(DST INDEX, DST MODE, ORIG DST REG VAL);
        'IF' DST ADDR > MAXCORE 'THEN' ERROR("DESTINATION ADDRESS OUT OF BOUNDS");
        DUPPER := 'IF' DST ADDR > 0 ^ EVEN(DST ADDR) ≠ 1 'THEN' 'TRUE' 'ELSE' 'FALSE';
        'IF' OPCODE < 8 ^ DST ADDR > 0 ^ DUPPER 'THEN' ERROR("DESTINATION ADDRESS ODD");
        DST := 'IF' DST ADDR < 0 'THEN' DST ADDR 'ELSE' DST ADDR ÷ 2;
        WHOLE DST := M[DST];
        DST OPERAND := 'IF' OPCODE < 8 'THEN' M[DST] 'ELSE' 'IF' DUPPER'THEN' M[DST] ÷ 256 'ELSE' AND(M[DST],255);
'END' EVAL ONE OPERAND;
```

```
'INTEGER' 'PROCEDURE' OPERAND ADDRESS(INDEX, MODE, ORIG);
'COMMENT' THIS COMPUTES THE OPERAND ADDRESS.  IT KNOWS ALL ABOUT THE EIGHT HARDWARE ADDRESSING MODES  THE MODE NEEDED IS IN THE
PARAMETER MODE. M[INDEX] GIVES THE REGISTER.  ORIG CONTAINS THE CONTENTS OF THAT REGISTER AS OF THE TIME WHEN
INSTRUCTION EXECUTION BEGAN, THUS UNMODIFIED BY SIDE EFFECTS OF THEOTHER OPERAND EVALUATION.;
'VALUE' INDEX, MODE, ORIG;
'INTEGER' INDEX, MODE, ORIG;
'BEGIN' 'SWITCH' ADDR MODE := MODE0, MODE1, MODE2, MODE3, MODE4, MODE5, MODE6, MODE7;
            INCR := 'IF' OPCODE > 7 ∧ INDEX > -7 'THEN' 1 'ELSE' 2;
            'GOTO' ADDR MODE[MODE+1];
            'COMMENT' OPERAND IS IN A REGISTER;
MODE0:      OPERAND ADDRESS := INDEX;
            NMODE0 := NMODE0 + 1;
            'GOTO' E;
            'COMMENT' OPERAND IS POINTED TO BY A REGISTER;
MODE1:      OPERAND ADDRESS := ORIG;
            NMODE1 := NMODE1 + 1;
            'GOTO' E;
            'COMMENT' AUTO INCREMENT MODE;
MODE2:      OPERAND ADDRESS := ORIG;
            TMP:= M[INDEX] + INCR;
            'IF' TMP = BIT16 'THEN' TMP := 0;
            'IF' TMP > MAXWORD 'THEN' TMP := TMP - BIT16;
            M[INDEX] := TMP;
            NMODE2 := NMODE2 + 1;
            'IF' INDEX = -8 'THEN' NIMMED := NIMMED + 1;
            'GOTO' E;
            'COMMENT' INDIRECT AUTO INCREMENT MODE;
MODE3:      'IF' ORIG > MAXCORE 'THEN' ERROR("ADDRESS OUT OF BOUNDS");
            'IF' EVEN(ORIG) ≠ 1 'THEN' ERROR("AUTOINCREMENT REGISTER ODD");
            OPERAND ADDRESS := M[ORIG ÷ 2];
            TMP := M[INDEX] + 2;
            'IF' TMP = BIT16 'THEN' TMP := 0;
            'IF' TMP > MAXWORD 'THEN' TMP := TMP - BIT16;
            M[INDEX] := TMP;
            NMODE3 := NMODE3 + 1;
            'IF' INDEX = -8 'THEN' NDIR := NDIR + 1;
            'GOTO' E;
            'COMMENT' AUTO DECREMENT MODE;
MODE4:      POINTER := 'IF' ORIG = INCR 'THEN' 0 'ELSE' ORIG - INCR;
            'IF' POINTER < 0 'THEN' POINTER := POINTER + BIT16;
            OPERAND ADDRESS := POINTER;
            TMP := M[INDEX];
            TMP := 'IF' TMP = INCR 'THEN' 0 'ELSE' TMP - INCR;
            'IF' TMP < 0 'THEN' TMP := TMP + BIT16;
            M[INDEX] := TMP;
            NMODE4 := NMODE4 + 1;
            'GOTO' E;
            'COMMENT' INDIRECT AUTO DECREMENT MODE;
MODE5:      POINTER := ORIG - 2;
            'IF' POINTER < 0 'THEN' POINTER := POINTER + BIT16;
            'IF' POINTER > MAXCORE 'THEN' ERROR("ADDRESS OUT OF BOUNDS");
            'IF' EVEN(POINTER) ≠ 1 'THEN' ERROR("AUTODECREMENT REGISTER ODD");
            OPERAND ADDRESS := M[POINTER ÷ 2];
            TMP := M[INDEX];
            TMP := 'IF' TMP = INCR 'THEN' 0 'ELSE' TMP - INCR;
            'IF' TMP < 0 'THEN' TMP := TMP + BIT16;
            M[INDEX] := TMP;
            NMODE5 := NMODE5 + 1;
            'GOTO' E;
            'COMMENT' INDEX MODE;
```

```
MODE6:  PC := M[-8];
        TMP := M[PC÷2];
        PC := PC + 2;
        'IF' PC > IO ADDR 'THEN' ERROR("PC IN IO AREA");
        M[-8] := PC;
        POINTER := TMP + M[INDEX];
        'IF' POINTER > MAXWORD 'THEN' POINTER := POINTER - BIT16;
        OPERAND ADDRESS := POINTER;
        NMODE6 := NMODE6 + 1;
        'IF' INDEX = -8 'THEN' NPIC := NPIC + 1;
        'GOTO' E;
        'COMMENT' INDIRECT INDEX MODE;
MODE7:  PC := M[-8];
        TMP := M[PC÷2];
        PC := PC + 2;
        'IF' PC > IO ADDR 'THEN' ERROR("PC IN IO AREA");
        M[-8] := PC;
        POINTER := TMP + M[INDEX];
        'IF' POINTER > MAXWORD 'THEN' POINTER := POINTER - BIT16;
        'IF' POINTER > MAXCORE 'THEN' ERROR("ADDRESS OUT OF BOUNDS");
        'IF' EVEN(POINTER) ≠ 1 'THEN' ERROR("INDIRECT INDEXING REGISTER ODD");
        OPERAND ADDRESS := M[POINTER ÷ 2];
        NMODE7 := NMODE7 + 1;
E:
'END' OPERAND ADDRESS;



'PROCEDURE' STARTIO;
'COMMENT' WHEN A REFERENCE IS MADE TO MEMORY IN THE IO AREA, THIS PROCEDURE IS CALLED TO FIGURE OUT WHICH IO DEVICE IS AFFECTED,
AND TO START THE IO PROCESS.  USUALLY THIS MEANS SETTING UP AN INTERRUPT.;
'BEGIN' 'SWITCH' IO INIT := READER, READERB, PUNCH, PUNCHB, TTY, TTYB, SOFTWR;
        TMP := (DST ADDR - IO ADDR + 2)÷2;
        'IF' TMP > 7 'THEN' ERROR("NON EXISTENT IO DEVICE");
        'GOTO' IOINIT[TMP];
READER: TMP := M[PRS2];
        'IF' EVEN(TMP) ≠ 1 'THEN'
            'BEGIN' 'IF' TMP > 2047 'THEN' ERROR("READER STILL BUSY");
                    ENABLE := 'IF' EVEN(TMP÷64) ≠ 1 'THEN' 64'ELSE'0;
                    M[PRS2] := 2048 + ENABLE;
                    M[PRB÷2] := 0;
                    PREPARE INTERRUPT(5,56,CLOCK+3 333 333.0);
            'END';
        'GOTO' E;
READERB:M[PRS2] := AND(M[PRS2],65407);
        'GOTO' E;
PUNCH:  'GOTO' E;
PUNCHB: TMP := M[PPS2];
        'IF' EVEN(TMP÷128)=1''THEN' ERROR("PUNCH BUSY") 'ELSE' M[PPS2] := 'IF' TMP = 192 'THEN' 64 'ELSE' 0;
        TMP2 := M[PPB2];
        M[TPB2] := TMP2 - 128 * (TMP2 ÷ 128);
        PREPARE INTERRUPT(4,60,20 000 000.0);
        'GOTO' E;
TTY:    'GOTO' E;
TTYB:   TMP := M[TPS2];
        'IF' EVEN(TMP ÷ 128)=1 'THEN' ERROR("CONSOLE TYPEWRITER BUSY") 'ELSE' M[TPS2] := 'IF' TMP=192 'THEN' 64 'ELSE' 0;
        TMP2:= M[TPB2];
        M[TPB2] := TMP2 - 128 * (TMP2 ÷128);
        PREPARE INTERRUPT(4,52,CLOCK+33 333 333.0);
        'GOTO' E;
```

```
SOFTWR: PIRWORD := M[PIR2] ÷ 512;
        PRI := 'IF' PIRWORD > 32 'THEN' 7
               'ELSE' 'IF' PIRWORD > 16 'THEN' 6
               'ELSE' 'IF' PIRWORD >  8 'THEN' 5
               'ELSE' 'IF' PIRWORD >  4 'THEN' 4
               'ELSE' 'IF' PIRWORD >  2 'THEN' 3
               'ELSE' 'IF' PIRWORD >  1 'THEN' 2
               'ELSE' 'IF' PIRWORD >  0 'THEN' 1  'ELSE' 0;
        'IF' PRI ≠ 0 'THEN'
            'BEGIN' M[PIR2] := 512 * PIRWORD + 34*PRI;
                    PREPARE INTERRUPT(PRI,16U,CLOCK);
            'END';
E:'END' START IO;



'PROCEDURE' PREPARE INTERRUPT(PRIORITY, VECTOR, TIME);
'COMMENT' THIS ROUTINE SETS UP THE INTERRUPT REQUEST ON THE CHAIN;
'VALUE' PRIORITY, VECTOR, TIME;
'INTEGER' PRIORITY, VECTOR;
'REAL' TIME;
'BEGIN' 'INTEGER' USELESS;
        'IF' FREE LIST HEAD = 0 'THEN' ERROR("TOO MANY INTERRUPTS");
        TMP := FREE LIST HEAD;
        FREE LIST HEAD := NEXT ON CHAIN[TMP];
        INTERRUPT TIME[TMP] := TIME;
        INTERRUPT VECTOR[TMP] := VECTOR;
        TMP1 := FIRST SLOT[PRIORITY];
        'IF' TMP1=0 ∨ TIME < INTERRUPT TIME[TMP1]
            'THEN' 'BEGIN' FIRST SLOT[PRIORITY] := TMP;
                           NEXT ON CHAIN[TMP]:= TMP1;
                   'END'
            'ELSE' 'BEGIN' L: 'IF' TIME < INTERRUPT TIME[TMP]
                                'THEN' 'BEGIN' NEXT ON CHAIN[TMP] := TMP1;
                                               NEXT ON CHAIN[TMP2]:=TMP
                                       'END'
                                'ELSE'
                                       'BEGIN' TMP2:= TMP1;
                                               TMP1:=NEXT ON CHAIN[TMP1];
                                               'IF' TMP1 ≠0 'THEN' 'GOTO' L;
                                               NEXT ON CHAIN[TMP2]:=TMP;
                                               NEXT ON CHAIN[TMP] := 0;
                                       'END'
                   'END';
           SET NEXT INTERRUPT TIME;
'END' PREPARE INTERRUPT;



'PROCEDURE' SET NEXT INTERRUPT TIME;
'BEGIN' 'COMMENT' THIS IS CALLED WHENEVER CPU PRIORITY IS CHANGED, IT EXAMINES THE LIST OF PENDING INTERRUPTS AND FINDS THE NEXT ONE;
        NEXT INTERRUPT TIME := LONG WAIT;
        'FOR' I := CPU PRIORITY + 1 'STEP' 1 'UNTIL' 7 'DO'
            'BEGIN' J:= FIRST SLOT[I];
                     'IF' J ≠ 0 'THEN'
                          'BEGIN' 'IF' INTERRUPT TIME[J] ≤ CLOCK ∨ INTERRUPT TIME[J] < NEXT INTERRUPT TIME
                                       'THEN' 'BEGIN' NEXT INTERRUPT TIME := INTERRUPT TIME[J];
                                                      QUEUE NUMBER := I;
                                              'END'
                         'END'
```

```
             'END'
'END' SET NEXT INTERRUPT TIME;



'PROCEDURE' CAUSE INTERRUPT;
'COMMENT' THE ACTUAL IO IS DONE HERE.  THIS IS CALLED AT THE TIME THE INTERRUPT OCCURS.  FIRST THE IO IS PERFORMED, THEN THE
SIMULATED INTERRUPT OCCURS AS DESCRIBED IN THE PROCESSOR HANDBOOK.;
'BEGIN' PRI INDEX := FIRST SLOT[QUEUE NUMBER];
        VEC := INTERRUPT VECTOR[PRI INDEX] ÷ 2;
        'IF' VEC = 26
            'THEN' 'BEGIN' 'COMMENT' TELETYPE PRINTER;
                           TMP := AND(M[TPB2],127);
                           LINE COUNT := 'IF' TMP = NEW LINE SYMBOL 'THEN' 0 'ELSE' LINE COUNT + 1;
                           'IF' LINECOUNT > 81 'THEN' ERROR("TELETYPE LINE OVERFLOW");
                           'IF' TMP=NEW LINE SYMBOL 'THEN' 'BEGIN' TYLIN := TYLIN + 1; TYPED := TYPED + 1 'END';
                           'IF' TYLIN > LINELIMIT 'THEN' ERROR("TOO MANY LINES OF OUTPUT");
                           PRSYM(U[TMP]);
                           'IF' TMP=95 ∨ TMP=124 'THEN' PRSYM(93);
                           M[TPS2] := M[TPS2] + DONE BIT;
                           ENABLE := M[TPS2] ÷ 64;
                   'END'
       'ELSE' 'IF' VEC = 28
            'THEN' 'BEGIN' 'COMMENT' PAPER TAPE READER;
                           'IF' LAST CHAR WAS 119
                                'THEN' 'BEGIN' TMP := 135;
                                               LAST CHAR WAS 119 := 'FALSE'
                                       'END'
                                'ELSE' TMP := RESYMBOL;
                           'IF' TMP=-27 'THEN' ERROR("ESCAPE CHARACTER READ.  NOT ALLOWED ");
                           'IF' TMP=119 'THEN' LASTCHARWAS119 := 'TRUE';
                           M[PRB2]:='IF' TMP>135 'THEN'0 'ELSE' 'IF' 1/TMP < 0 'THEN' -TMP 'ELSE' Y[TMP];
                           M[PRS2] := M[PRS2] - 1920;
                           ENABLE := M[PRS2] ÷ 64;
                           CHAR RD := CHAR RD + 1;
                   'END'
        'ELSE' 'IF' VEC = 30
            'THEN' 'BEGIN' 'COMMENT' PAPER TAPE PUNCH;
                           PUHEP(AND(M[PRB2],255));
                           M[PPS2] := M[PPS2] + DONE BIT;
                           ENABLE := M[PPS2] ÷ 64;
                           CHAR PUN := CHAR PUN + 1;
                   'END'
        'ELSE' 'IF' VEC = 80
            'THEN' ENABLE := 1
        'ELSE' ERROR("IMPOSSIBLE INTERRUPT. SIMULATOR ERROR");
        'IF' EVEN(ENABLE) ≠ 1 'THEN'
            'BEGIN' SWITCH PSW;
                    CLOCK := CLOCK + 5190.0;
                    NINTER := NINTER + 1;
            'END' ;
        FIRST SLOT[QUEUE NUMBER]:= NEXT ON CHAIN[PRI INDEX];
        NEXT ON CHAIN[PRI INDEX] := FREE LIST HEAD;
        FREE LIST HEAD := PRI INDEX;
        SET NEXT INTERRUPT TIME;
'END' CAUSE INTERRUPT;



'PROCEDURE' SWITCH PSW;
```

```
'COMMENT' PUSH THE PSW AND PC ON THE STACK, AND FETCH THE NEW ONE  FROM THE INTERRUPT LOCATION IN MEMORY;
'BEGIN' SP := M[-7];
        'IF' EVEN(SP) ‡ 1 'THEN' ERROR("STACK POINTER ODD");
        'IF' SP > IO ADDR 'THEN' ERROR ("STACK POINTER TOO BIG");
        'IF' SP < STACKLIMIT + 4 'THEN' ERROR("STACK OVERFLOW");
        TMP := (SP ÷2) - 1;
        M[TMP] := 32*CPU PRIORITY + 16*T + 8*N + 4*Z + 2*V + C;
        M[TMP-1] := M[-8];
        M[-7] := SP-4;
        M[-8] := M[VEC];
        PSW := M[VEC+1];
        C := PSW - 2*(PSW÷2);   PSW := PSW ÷2;
        V := PSW - 2*(PSW÷2);   PSW := PSW ÷2;
        Z := PSW - 2*(PSW÷2);   PSW := PSW ÷2;
        N := PSW - 2*(PSW÷2);   PSW := PSW ÷2;
        T := PSW - 2*(PSW÷2);
        CPU PRIORITY := PSW ÷2;
'END' SWITCH PSW;



'PROCEDURE' STATISTICS;
'COMMENT' THE POST MORTEM STATISTICS ARE PRINTED HERE;
'BEGIN' ENDTIME := TIME;  SPACER := 2;  NLCR;  NLCR;
        BEGIN TIME := 1000 * BEGIN TIME;  END TIME := ENDTIME * 1000;

'COMMENT' GROUP 1: THE CORE DUMP;
I:= IO ADDR ÷2; Q := I;
'FOR' I := I-1 'WHILE' M[I] = 0 'DO' Q:= I;
CORE DUMP(0,2*Q);

'COMMENT' GROUP 2: TIMES OF EXECUTION;
        X:=NADC+NADCB+NADD+NASL+NASLB+NASH+NASHC+NASR+NASRB+NBCC+NBCS+NBEQ+NBGE+NBGT+NBHI+NBIC+NBICB+NBIS+NBISB+NBIT+NBITB+NBLT+NBLE+
           NBMI+NBNE+NBPL+NBPT+NBR+NBVC+NBVS+NCLR+NCLRB+NCMPB+NCOMB+NCC+NDEC+NDECB+NDIV+NEMT+NHALT+NINC+NINCB+NIOT+NJMP+NJSR+NMARK+NMOV+
           NMUL+NNEG+NNEGB+NREST+NROL+NROLB+NROR+NRORB+NRTI+NRTS+NRTT+NSBC+NSBCB+NSOB+NSPL+NSUB+NSWAB+NSXT+NTRAP+NTST+NTSTB+NWAIT+NXCR+
           NBLOS+NMOVB+NCMP+NCOM;
        CARRIAGE(5);    PRINTTEXT("SIMULATION STATISTICS");  NLCR; NLCR;
        PRINTTEXT("PDP TIME USED = ");   ABSFIXT(8,3,CLOCK/1000000);   PRINTTEXT(" MILLISECONDS "); SPACE(10);
        PRINTTEXT("X8 TIME USED = ");   ABSFIXT(8,3,ENDTIME-BEGINTIME);         PRINTTEXT(" MILLISEC,");  SPACE(10);
        PRINTTEXT("X8 TIME PER PDP SECOND = ");   ABSFIXT(8,0,(ENDTIME-BEGINTIME)*1 000 000/CLOCK);   NLCR;
        PRINTTEXT("PDP WAIT TIME = ");   ABSFIXT(8,3,IDLE/1000000);   PRINTTEXT(" MILLISECONDS");   SPACE(10);
        PRINTTEXT("NUMBER OF INTERRUPTS = ");   ABSFIXT(8,0,NINTER);   SPACE(10);
        ABSFIXT(6,0,X);   PRINTTEXT(" INSTRUCTIONS EXECUTED");  SPACE(10);       NLCR;
        PRINTTEXT("NUMBER OF CHARACTERS READ = "); ABSFIXT(6,0,CHAR RD);
        SPACE(10); PRINTTEXT("NUMBER OF LINES TYPED = "); ABSFIXT(4,0,TYPED); SPACE(10);
        PRINTTEXT("NUMBER OF CHARACTERS PUNCHED = ");ABSFIXT(6,0,CHAR PUN); NLCR;
        PRINTTEXT("AVERAGE PDP TIME PER INSTRUCTION = ");   ABSFIXT(6,0,(CLOCK-IDLE)/X);   PRINTTEXT(" NANOSECONDS");  NLCR;
        PRINTTEXT("AVERAGE NUMBER OF PDP INSTRUCTIONS PER SECOND OF X8 TIME = ");  ABSFIXT(6,0,1000*X/(ENDTIME-BEGINTIME)); NLCR;
        PRINTTEXT("DIST OF ADDR MODES 0-7 ");
        ABSFIXT(6,0,NMODE0); ABSFIXT(6,0,NMODE1); ABSFIXT(6,0,NMODE2); ABSFIXT(6,0,NMODE3); ABSFIXT(6,0,NMODE4);
        ABSFIXT(6,0,NMODE5); ABSFIXT(6,0,NMODE6); ABSFIXT(6,0,NMODE7); SPACE(5);
        PRINTTEXT("IMMEDIATE=");    ABSFIXT(6,0,NIMMED);  PRINTTEXT("    DIRECT=");ABSFIXT(6,0,NDIR);  PRINTTEXT("   PIC=");
        ABSFIXT(6,0,NPIC);  NLCR;  NLCR;

'COMMENT' GROUP 3: PROGRAM COUNTER DISTRIBUTION;
        PRINTTEXT("PROGRAM COUNTER DISTRIBUTION.   ADDRESS (OCTAL)     FREQUENCY OF USE (DECIMAL)");
        NLCR;
        PRINTTEXT("BELOW "); OCTAL(LOWBIN); ABSFIXT(6,0,SPY[-1]); SPACE(10);
        PRINTTEXT("ABOVE ");  OCTAL(LOWBIN+128*BINWIDTH); ABSFIXT(6,0,SPY[128]);  SPACER := 1;
        'FOR' I:= 0 'STEP' 1 'UNTIL' 127 'DO'
```

```
             'BEGIN' 'IF' REMAINDER(I,8)=0 'THEN' NLCR;
                     OCTAL(LOWBIN+I*BINWIDTH);     ABSFIXT(6,0,SPY[I]); SPACE(3);
             'END';
        CARRIAGE(3);
        'IF' X=0 'THEN' X:=1;
        X != X/100.0;

'COMMENT' GROUP 4:  INSTRUCTION FREQUENCIES;
PRINTTEXT("ADC  ");ABSFIXT(5,0,NADC );ABSFIXT(4,1,NADC /X);FIVE;
PRINTTEXT("ADCB ");ABSFIXT(5,0,NADCB);ABSFIXT(4,1,NADCB/X);FIVE;       PRINTTEXT("ADD  ");ABSFIXT(5,0,NADD );ABSFIXT(4,1,NADD /X);FIVE;
PRINTTEXT("ASL  ");ABSFIXT(5,0,NASL );ABSFIXT(4,1,NASL /X);FIVE;       PRINTTEXT("ASLB ");ABSFIXT(5,0,NASLB);ABSFIXT(4,1,NASLB/X);FIVE;
PRINTTEXT("ASH  ");ABSFIXT(5,0,NASH );ABSFIXT(4,1,NASH /X);NLCR;       PRINTTEXT("ASHC ");ABSFIXT(5,0,NASHC);ABSFIXT(4,1,NASHC/X);FIVE;
PRINTTEXT("ASR  ");ABSFIXT(5,0,NASR );ABSFIXT(4,1,NASR /X);FIVE;       PRINTTEXT("ASRB ");ABSFIXT(5,0,NASRB);ABSFIXT(4,1,NASRB/X);FIVE;
PRINTTEXT("BCC  ");ABSFIXT(5,0,NBCC );ABSFIXT(4,1,NBCC /X);FIVE;       PRINTTEXT("BCS  ");ABSFIXT(5,0,NBCS );ABSFIXT(4,1,NBCS /X);FIVE;
PRINTTEXT("BEQ  ");ABSFIXT(5,0,NBEQ );ABSFIXT(4,1,NBEQ /X);NLCR;       PRINTTEXT("BGE  ");ABSFIXT(5,0,NBGE );ABSFIXT(4,1,NBGE /X);FIVE;
PRINTTEXT("BGT  ");ABSFIXT(5,0,NBGT );ABSFIXT(4,1,NBGT /X);FIVE;       PRINTTEXT("BHI  ");ABSFIXT(5,0,NBHI );ABSFIXT(4,1,NBHI /X);FIVE;
PRINTTEXT("BIC  ");ABSFIXT(5,0,NBIC );ABSFIXT(4,1,NBIC /X);FIVE;       PRINTTEXT("BICB ");ABSFIXT(5,0,NBICB);ABSFIXT(4,1,NBICB/X);FIVE;
PRINTTEXT("BIS  ");ABSFIXT(5,0,NBIS );ABSFIXT(4,1,NBIS /X);NLCR;       PRINTTEXT("BISB ");ABSFIXT(5,0,NBISB);ABSFIXT(4,1,NBISB/X);FIVE;
PRINTTEXT("BIT  ");ABSFIXT(5,0,NBIT );ABSFIXT(4,1,NBIT /X);FIVE;       PRINTTEXT("BITB ");ABSFIXT(5,0,NBITB);ABSFIXT(4,1,NBITB/X);FIVE;
PRINTTEXT("BLT  ");ABSFIXT(5,0,NBLT );ABSFIXT(4,1,NBLT /X);FIVE;       PRINTTEXT("BLE  ");ABSFIXT(5,0,NBLE );ABSFIXT(4,1,NBLE /X);FIVE;
PRINTTEXT("BLOS ");ABSFIXT(5,0,NBLOS);ABSFIXT(4,1,NBLOS/X);NLCR;       PRINTTEXT("BMI  ");ABSFIXT(5,0,NBMI );ABSFIXT(4,1,NBMI /X);FIVE;
PRINTTEXT("BNE  ");ABSFIXT(5,0,NBNE );ABSFIXT(4,1,NBNE /X);FIVE;       PRINTTEXT("BPL  ");ABSFIXT(5,0,NBPL );ABSFIXT(4,1,NBPL /X);FIVE;
PRINTTEXT("BPT  ");ABSFIXT(5,0,NBPT );ABSFIXT(4,1,NBPT /X);FIVE;       PRINTTEXT("BR   ");ABSFIXT(5,0,NBR  );ABSFIXT(4,1,NBR  /X);FIVE;
PRINTTEXT("BVC  ");ABSFIXT(5,0,NBVC );ABSFIXT(4,1,NBVC /X);NLCR;       PRINTTEXT("BVS  ");ABSFIXT(5,0,NBVS );ABSFIXT(4,1,NBVS /X);FIVE;
PRINTTEXT("CLR  ");ABSFIXT(5,0,NCLR );ABSFIXT(4,1,NCLR /X);FIVE;       PRINTTEXT("CLRB ");ABSFIXT(5,0,NCLRB);ABSFIXT(4,1,NCLRB/X);FIVE;
PRINTTEXT("CMP  ");ABSFIXT(5,0,NCMP );ABSFIXT(4,1,NCMP /X);FIVE;       PRINTTEXT("CMPB ");ABSFIXT(5,0,NCMPB);ABSFIXT(4,1,NCMPB/X);FIVE;
PRINTTEXT("COM  ");ABSFIXT(5,0,NCOM );ABSFIXT(4,1,NCOM /X);NLCR;       PRINTTEXT("COMB ");ABSFIXT(5,0,NCOMB);ABSFIXT(4,1,NCOMB/X);FIVE;
PRINTTEXT("CC   ");ABSFIXT(5,0,NCC  );ABSFIXT(4,1,NCC  /X);FIVE;       PRINTTEXT("DEC  ");ABSFIXT(5,0,NDEC );ABSFIXT(4,1,NDEC /X);FIVE;
PRINTTEXT("DECB ");ABSFIXT(5,0,NDECB);ABSFIXT(4,1,NDECB/X);FIVE;       PRINTTEXT("DIV  ");ABSFIXT(5,0,NDIV );ABSFIXT(4,1,NDIV /X);FIVE;
PRINTTEXT("EMT  ");ABSFIXT(5,0,NEMT );ABSFIXT(4,1,NEMT /X);NLCR;       PRINTTEXT("HALT ");ABSFIXT(5,0,NHALT);ABSFIXT(4,1,NHALT/X);FIVE;
PRINTTEXT("INC  ");ABSFIXT(5,0,NINC );ABSFIXT(4,1,NINC /X);FIVE;       PRINTTEXT("INCB ");ABSFIXT(5,0,NINCB);ABSFIXT(4,1,NINCB/X);FIVE;
PRINTTEXT("IOT  ");ABSFIXT(5,0,NIOT );ABSFIXT(4,1,NIOT /X);FIVE;       PRINTTEXT("JMP  ");ABSFIXT(5,0,NJMP );ABSFIXT(4,1,NJMP /X);FIVE;
PRINTTEXT("JSR  ");ABSFIXT(5,0,NJSR );ABSFIXT(4,1,NJSR /X);NLCR;       PRINTTEXT("MARK ");ABSFIXT(5,0,NMARK);ABSFIXT(4,1,NMARK/X);FIVE;
PRINTTEXT("MOV  ");ABSFIXT(5,0,NMOV );ABSFIXT(4,1,NMOV /X);FIVE;       PRINTTEXT("MOVB ");ABSFIXT(5,0,NMOVB);ABSFIXT(4,1,NMOVB/X);FIVE;
PRINTTEXT("MUL  ");ABSFIXT(5,0,NMUL );ABSFIXT(4.1,NMUL /X);FIVE;       PRINTTEXT("NEG  ");ABSFIXT(5,0,NNEG );ABSFIXT(4,1,NNEG /X);FIVE;
PRINTTEXT("NEGB ");ABSFIXT(5,0,NNEGB);ABSFIXT(4.1,NNEGB/X);NLCR;       PRINTTEXT("RESET");ABSFIXT(5,0,NREST);ABSFIXT(4,1,NREST/X);FIVE;
PRINTTEXT("ROL  ");ABSFIXT(5,0,NROL );ABSFIXT(4,1,NROL /X);FIVE;       PRINTTEXT("ROLB ");ABSFIXT(5,0,NROLB);ABSFIXT(4.1,NROLB/X);FIVE;
PRINTTEXT("ROR  ");ABSFIXT(5,0,NROR );ABSFIXT(4,1,NROR /X);FIVE;       PRINTTEXT("RORB ");ABSFIXT(5,0,NRORB);ABSFIXT(4,1,NRORB /X);FIVE;
PRINTTEXT("RTI  ");ABSFIXT(5,0,NRTI );ABSFIXT(4,1,NRTI /X);NLCR;       PRINTTEXT("RTS  ");ABSFIXT(5,0,NRTS );ABSFIXT(4,1,NRTS /X);FIVE;
PRINTTEXT("RTT  ");ABSFIXT(5,0,NRTT );ABSFIXT(4,1,NRTT /X);FIVE;       PRINTTEXT("SBC  ");ABSFIXT(5,0,NSBC );ABSFIXT(4,1,NSBC /X);FIVE;
PRINTTEXT("SBCB ");ABSFIXT(5,0,NSBCB);ABSFIXT(4,1,NSBCB/X);FIVE;       PRINTTEXT("SOB  ");ABSFIXT(5,0,NSOB );ABSFIXT(4,1,NSOB /X);FIVE;
PRINTTEXT("SPL  ");ABSFIXT(5,0,NSPL );ABSFIXT(4.1,NSPL /X);NLCR;       PRINTTEXT("SUB  ");ABSFIXT(5,0,NSUB );ABSFIXT(4,1,NSUB /X);FIVE;
PRINTTEXT("SWAB ");ABSFIXT(5,0,NSWAB);ABSFIXT(4,1,NSWAB/X);FIVE;       PRINTTEXT("SXT  ");ABSFIXT(5,0,NSXT );ABSFIXT(4,1,NSXT /X);FIVE;
PRINTTEXT("TRAP ");ABSFIXT(5,0,NTRAP);ABSFIXT(4,1,NTRAP/X);FIVE;       PRINTTEXT("TST  ");ABSFIXT(5,0,NTST );ABSFIXT(4,1,NTST /X);FIVE;
PRINTTEXT("TSTB ");ABSFIXT(5,0,NTSTB);ABSFIXT(4,1,NTSTB/X);NLCR;       PRINTTEXT("WAIT ");ABSFIXT(5,0,NWAIT);ABSFIXT(4,1,NWAIT/X);FIVE;
PRINTTEXT("XOR  ");ABSFIXT(5,0,NXOR );ABSFIXT(4,1,NXOR /X);FIVE;
NLCR;
NLCR; NLCR; PRINTTEXT(" TOTAL TIME FOR THIS JOB = "); ABSFIXT(5,0,TIME-BEGTIM); PRINTTEXT("SECONDS, ");
NEWPAGE; 'FOR' I:=1'STEP'1'UNTIL'144'DO'PRSYM(66);
 'FOR'I:=1'STEP'1'UNTIL'58'DO' 'BEGIN' PRSYM(66);
                                        'IF' I=29 'THEN' 'BEGIN' SPACE(66); PRINTTEXT("END OF JOB"); SPACE(66) 'END''ELSE'SPACE(142);
                                        PRSYM(66);
                                'END';
'FOR' I:=1 'STEP' 1 'UNTIL' 144 'DO' PRSYM(66);  NLCR;
NJOBS := NJOBS + 1;
'GOTO' BEGIN ASSEMBLY;
'END' STATISTICS;
```

```
'PROCEDURE' FIVE; SPACE(4);



'PROCEDURE' ERROR(STR);
'COMMENT' ALL SIMULATION ERRORS ARE FUNNELLED THRU HERE;
'BEGIN' NLCR;  NLCR;  PRINTTEXT("EXECUTION CANNOT CONTINUE DUE TO ");  PRINTTEXT(STR);
          TYLIN := -1 000 000;  USER DUMP;  STATISTICS;
'END' ERROR;



'PROCEDURE' OCTAL(N);
'COMMENT' PRINT N IN OCTAL;
'VALUE' N;
'INTEGER' N;
'BEGIN' 'IF' N<65536 'THEN' PRSYM(AND(N,32768)÷32768) 'ELSE' PRSYM(AND(N,32768)÷32768 + 10);
        PRSYM(AND(N,28672) ÷ 4096);
        PRSYM(AND(N,3584) ÷ 512);
        PRSYM(AND(N,448) ÷ 64);
        PRSYM(AND(N,56) ÷ 8);
        PRSYM(AND(N,7));
        SPACE(SPACER);
'END';



'PROCEDURE' USER DUMP;
'COMMENT' PRINT OUT THE PROCESSOR STATUS: REGISTERS, FLIP FLOPS, INST;
'BEGIN' NLCR;  SPACER := 2;
        PRINTTEXT("TIME (MS) "); ABSFIXT( 9,6,CLOCK/1 000 000);  SPACE(4);  PRINTTEXT("REGISTERS  ");
        'FOR' I := -1 'STEP' -1 'UNTIL' -8 'DO' OCTAL(M[I]);  SPACE(2);
        PRINTTEXT("NZVC  "); ABSFIXT(1,0,N);  ABSFIXT(1,0,Z); ABSFIXT(1,0,V); ABSFIXT(1,0,C); SPACE(2);
        PRINTTEXT("INST " );  OCTAL(INSTRUCTION);
        TYLIN := TYLIN + 1; 'IF' TYLIN>LINE LIMIT 'THEN' ERROR("TOO MUCH PRINTED OUTPUT");
'END' USER DUMP;



'PROCEDURE' CORE DUMP(N1,N2);
'COMMENT' PRINT A CORE DUMP OF LOCATIONS N1 THRU N2;
'VALUE' N1,N2; 'INTEGER' N1,N2;
'BEGIN' NLCR; NLCR; SPACER := 1;
        PRINTTEXT("CORE DUMP ");  NLCR;
        'FOR' K := 32*(N1÷32) 'STEP' 32 'UNTIL' N2    'DO'
        'BEGIN' OCTAL(K); SPACE(2);
          'IF' K≥0 ∧ K+31 ≤ MAXCORE 'THEN'
             'BEGIN'
                'FOR' I:= K   'STEP' 2 'UNTIL' K+7 'DO' OCTAL(M[I÷2]); SPACE(2);
                'FOR' I:= K+8 'STEP' 2 'UNTIL' K+15'DO' OCTAL(M[I÷2]); SPACE(2);
                'FOR' I:= K+16'STEP' 2 'UNTIL' K+23'DO' OCTAL(M[I÷2]); SPACE(2);
                'FOR' I:= K+24'STEP' 2 'UNTIL' K+31'DO' OCTAL(M[I÷2]);NLCR;
        TYLIN := TYLIN + 1; 'IF' TYLIN>LINE LIMIT 'THEN' ERROR("TOO MUCH PRINTED OUTPUT");
             'END';
        'END';
'END' CORE DUMP;
```

```
'COMMENT' INITIALIZATION;
OUTISO;
NEW PAGE;
BIT16 := 65536;    MAXPOS := 32767;    MAXWORD := 65535;    DONEBIT := 128;    NEW LINE SYMBOL := 13;    CPU PRIORITY := 1;
N:=Z:=V:=C:=T:=0; FREE LIST HEAD := 1;    LINE COUNT := 0;              NUM INTERRUPTSLOTS := 16;    PRS := IO ADDR;
PRB := PRS + 2;
PPS := PRB + 2;    PPB := PPS + 2;    TPS := PPB + 2;    TPB :=  TPS + 2;    PIR :=  TPB + 2;    ZERO := 0;
INST COUNT := 0;  TYPED := TYLIN := CHAR RD := CHAR PUN := 0;
STACK LIMIT := 256;    IDLE :=0;
CLOCK := 0.0;    LONG WAIT := NEXT INTERRUPT  TIME := 1.0⏨100;
LAST CHAR WAS 119 := 'FALSE';
SPACER := 1;
NMODE0:=NMODE1:=NMODE2:=NMODE3:=NMODE4:=NMODE5:=NMODE6:=NMODE7:=NIMMED:=NDIR:=NINTER:=NPIC:=0;
NADC:=NADCB:=NADD:=NASL:=NASLB:=NASH:=NASHC:=NASR:=NASRB:=NBCC:=NBCS:=NBEQ:=NBGE:=0;
NBEQ:=NBGE:=NBGT:=NBHI:=NBIC:=NBICB:=NBIS:=NBISB:=NBIT:=NBITB:=NBLT:= 0;
NBLE:=NBLOS:=NBMI:=NBNE:=NBPL:=NBPT:=NBR:=NBVC:=NBVS:=NCLR:=NCLRB:=NCMP:=NCMPB:=NCOM:=NCOMB:=NCC:=NDEC:=NDECB:=NDIV:=NEMT:=NHALT:=0;
NINCB:=NIOT:=NJMP:=NJSR:=NMARK:=NMOV:=NMOVB:=NMUL:=NNEG:=NNEGB:=NREST:=NROL:=NROLB:=NROR:=NRORB:=NRTI:=NRTS:=NRTT:=NSBC:=NSBCB:=0;
NSUB:=NSWAB:=NSXT:=NTRAP:=NTST:=NTSTB:=NWAIT:=NXOR:=NINC:=NSPL:=NSOB:=0;
'FOR' I:= 1 'STEP' 1 'UNTIL' NUM INTERRUPT SLOTS 'DO' NEXT ON CHAIN[I]:= I+1;
NEXT ON CHAIN[NUM INTERRUPT SLOTS] := 0;
'FOR' I:= 1 'STEP' 1 'UNTIL' 7 'DO' FIRST SLOT[I] := 0;
'FOR' I:=-1 'STEP' 1  'UNTIL' 128 'DO' SPY[I] :=0;
PRS2:=PRS÷2;    PPS2:=PPS÷2;    TPS2:=TPS÷2;    PRB2:=PRB÷2;   PPB2:=PPB÷2;    TPB2:=TPB÷2;

'COMMENT' SEE IF THE OBJECT PROGRAM IS IN THE DRUM BUFFER, OR ON THEDRUM ITSELF.  THE DRUM WILL ONLY BE USED IF NEEDED;
'FOR' I:=  0'STEP' 1 'UNTIL' MAXCORE ÷ 2 'DO' M[I] := 0;
'IF' HIGHEST ADDRESS ≥BLOCK SIZE
     'THEN' 'BEGIN' FROM DRUM(M,OFFSET-8);
                    'FOR' I:= 0 'STEP' 1 'UNTIL' BLOCK SIZE - 1 'DO' M[BLOCK SIZE*CURRENT BLOCK +I] := CODE BUFFER[I]
            'END'
     'ELSE' 'BEGIN' 'FOR' I:= 0 'STEP' 1 'UNTIL' HIGHEST ADDRESS 'DO'M[I] := CODE BUFFER[I];    'END';
'FOR' I:= -8'STEP' 1 'UNTIL' -1'DO' M[I] := 0;
TRACE FLAG:='FALSE';    M[TPS÷2]:=M[PPS÷2]:=128;
M[-8]:=STARTING ADDRESS; LOWBIN:=256*(LOWEST ADDRESS÷256);  BINWIDTH:= ((256*(  (HIGHEST ADDRESS+255) ÷256)) - LOWBIN)÷128;

BEGIN TIME := TIME;
'GOTO' CYCLE;




'COMMENT' THIS IS THE MAIN SIMULATION LOOP.  EVERY SIMULATED INSTRUCTION IS PICKED UP HERE AND DECODED  THE OPCODE MODES AND
REGISTERS ARE STORED SEPARATELY.  THE SWITCH INSTRUCTION TYPE DISPATCHES TO THE PIECE OF CODE THAT SIMULATES THE OPCODE;
CYCLE:    'IF' CLOCK ≥ NEXT INTERRUPT TIME 'THEN' CAUSE INTERRUPT;
          OLDPC := M[-8];
          'IF' OLDPC > IO ADDR 'THEN' ERROR("PC IN IO AREA");
          'IF' EVEN(OLDPC) ≠ 1 'THEN' ERROR("PC ODD");
          INSTRUCTION := M[OLDPC ÷ 2];
          M[-8] := OLDPC + 2;
          OPCODE := INSTRUCTION ÷ 4096;
          TMP := INSTRUCTION - 4096 * OPCODE;
          SRC MODE := TMP ÷ 512;    TMP := TMP - 512 * SRC MODE;
```

```
         SRC REG := TMP ÷ 64;      TMP := TMP -  64 * SRC REG;
         DST MODE := TMP ÷ 8;
         DST REG := TMP - 8 * DST MODE;
         INST TIME := 0;
         DST ADDR := 0;
         I := (OLDPC - LOWBIN) ÷ BINWIDTH;
         'IF' I<0 'THEN' I:= -1 'ELSE' 'IF' I> 127 'THEN' I := 128;
         SPY[I] := SPY[I] + 1;
         'IF' OPCODE < 0 ∨ OPCODE > 15 'THEN' ERROR("SIMULATOR ERROR  OPCODE RANGE");
         'GOTO' INSTRUCTION TYPE[OPCODE+1];
EXDONE:  CLOCK := CLOCK + INST TIME;
         'IF' TRACE FLAG 'THEN' USER DUMP;
         'IF' DST ADDR ≥ IO ADDR 'THEN' STARTIO;
        INST COUNT := INST COUNT + 1; 'IF' INST COUNT > INST LIMIT 'THEN' ERROR("INSTRUCTION LIMIT EXCEEDED ");
         'GOTO' CYCLE;
'COMMENT' END OF MAIN SIMULATION LOOP;


OP IS 0: 'GOTO' OP 0 SPLIT[SRC MODE +1];
MIXED:   'GOTO' MIXED SPLIT[SRC REG + 1];
NOAD:    'IF' INSTRUCTION > 6 'THEN' 'GOTO' ILLEGAL 'ELSE' 'GOTO' NOAD SPLIT[DST REG+1];
MIXED1:   'GOTO' MIXED1 SPLIT[DST MODE+1];
BR1:     'IF' SRC REG < 4 'THEN' 'GOTO' BNE 'ELSE' 'GOTO' BEQ;
BR2:     'IF' SRC REG < 4 'THEN' 'GOTO' BGE 'ELSE' 'GOTO' BLT;
BR3:     'IF' SRC REG < 4 'THEN' 'GOTO' BGT 'ELSE' 'GOTO' BLE;
ONEOPIES1:'GOTO' ONE OPIES1 SPLIT[SRC REG+1];
ONEOPIES2:'GOTO' ONE OPIES2 SPLIT[SRC REG+1];
OP IS 7: 'GOTO' OP 7 SPLIT[SRC MODE+1];
OP IS 8: 'GOTO' OP 8 SPLIT[SRC MODE+1];
BR4:     'IF' SRC REG < 4 'THEN' 'GOTO' BPL 'ELSE' 'GOTO' BMI;
BR5:     'IF' SRC REG < 4 'THEN' 'GOTO' BHI 'ELSE' 'GOTO' BLOS;
BR6:     'IF' SRC REG < 4 'THEN' 'GOTO' BVC 'ELSE' 'GOTO' BVS;
BR7:     'IF' SRC REG < 4 'THEN' 'GOTO' BHIS'ELSE' 'GOTO' BLO;
EMTTRAP: 'IF' SRC REG < 4 'THEN' 'GOTO' EMT 'ELSE' 'GOTO' TRAP;
ONEOPIES3:'GOTO' ONE OPIES 3 SPLIT[SRC REG+1];
ONEOPIES4:'GOTO' ONE OPIES 4 SPLIT[SRC REG+1];
FPT:     ERROR("FLOATING POINT NOT SIMULATED");


'COMMENT' HERE ARE THE PIECES OF CODE THAT SIMULATE THE INDIVIDUAL INSTRUCTIONS.  EACH OPCODE IS SIMULATED BY THE PIECE OF
CODE LABELLED BY THE ASSEMBLER MNEUMONIC;
MOV:    EVAL BOTH OPERANDS;
        M[DST]:= SRC OPERAND;
        N:= 'IF' SRC OPERAND > MAX POS 'THEN' 1 'ELSE' 0;
        Z:= 'IF' SRC OPERAND = 0 'THEN' 1 'ELSE' 0;
        V:= 0;
        NMOV:= NMOV + 1;
        INST TIME:= 850 + SRC TIMING[SRC MODE] +
                ('IF' SRC MODE = 0 'THEN' DST TIMING2A[DST MODE]
                                   'ELSE' DST TIMING2[DST MODE]);
        'IF' DST MODE = 0 ∧ DST = -8 'THEN' INST TIME:= INST TIME + 290;
        'GOTO' EXDONE;

MOVB:   EVAL BOTH OPERANDS;
        N := 'IF' SRC OPERAND > 127 'THEN' 1 'ELSE' 0;
        TMP:= 'IF' DST < 0 'THEN' 255*N 'ELSE' 'IF' DUPPER 'THEN'
                AND(M[DST],255) 'ELSE' M[DST] ÷ 256;
        M[DST]:= 'IF' DUPPER 'THEN' 256*SRC OPERAND + TMP
                             'ELSE' 256*TMP + SRC OPERAND;
        Z:= 'IF' SRC OPERAND = 0 'THEN' 1 'ELSE' 0;
```

```
                V:= 0;
                NMOVB:= NMOVB + 1;
                'GOTO' MOVX;

INC:            EVAL ONE OPERAND;
                NINC:= NINC + 1;
                TMP:= 'IF' DST OPERAND = MAXWORD 'THEN' 0 'ELSE' DST OPERAND + 1;
                V:= 'IF' DST OPERAND = MAXPOS 'THEN' 1 'ELSE' 0;
INCX:           M[DST]:= TMP;
TSTX:           N:= 'IF' TMP > MAXPOS 'THEN' 1 'ELSE' 0;
                Z:= 'IF' TMP = 0 'THEN' 1 'ELSE' 0;
SWAX:           INST TIME:= INST TIME + ('IF' DST MODE = 0 'THEN' 850 'ELSE' 1620) +
                        DST TIMING3A[DST MODE];
                'IF' DST MODE = 0 ^ DST = -8 'THEN' INST TIME:= INST TIME + 290;
                'GOTO' EXDONE;

CLR:            EVAL ONE OPERAND;
                NCLR:= NCLR + 1;
                TMP:= V:= C:= 0;
                'GOTO' INCX;

NEG:            EVAL ONE OPERAND; NNEG:= NNEG + 1;
                TMP:= 'IF' DST OPERAND = 0 'THEN' 0 'ELSE' 65536 - DST OPERAND;
                C:= 'IF' TMP = 0 'THEN' 0 'ELSE' 1;
                V:= 'IF' TMP = 32768 'THEN' 1 'ELSE' 0;
                INST TIME:= 'IF' DST MODE = 0 'THEN' 440 'ELSE' 210;
                'GOTO' INCX;

DEC:            EVAL ONE OPERAND;
                NDEC:= NDEC + 1;
                TMP:= 'IF' DST OPERAND = 0 'THEN' MAXWORD
                        'ELSE' 'IF' DST OPERAND = 1 'THEN' ZERO 'ELSE' DST OPERAND - 1;
                V:= 'IF' DST OPERAND = 32768 'THEN' 1 'ELSE' 0;
                'GOTO' INCX;

COM:            EVAL ONE OPERAND;
                NCOM:= NCOM + 1;
                TMP:= 'IF' DST OPERAND = MAXWORD 'THEN' 0 'ELSE' MAXWORD - DST OPERAND;
                V:= 0;
                C:= 1;
                'GOTO' INCX;

TST:            EVAL ONE OPERAND;
                NTST:= NTST + 1;
                V:= C:= 0;
                TMP:= DST OPERAND;
                'IF' DST MODE ≠ 0 'THEN' INST TIME:= -480;
                'GOTO' TSTX;

INCB:           EVAL ONE OPERAND;
                NINCB:= NINCB + 1;
                BYTE:= 'IF' DST OPERAND = 255 'THEN' 0 'ELSE' DST OPERAND + 1;
                V:= 'IF' DST OPERAND = 127 'THEN' 1 'ELSE' 0;
INCBX:          TMP:= 'IF' DUPPER 'THEN' AND(WHOLE DST,255) 'ELSE' WHOLE DST ÷256;
                M[DST]:= 'IF' DUPPER 'THEN' 256 * BYTE + TMP 'ELSE' 256 * TMP + BYTE;
TSTBX:          N:= 'IF' BYTE > 127 'THEN' 1 'ELSE' 0;
                Z:= 'IF' BYTE = 0 'THEN' 1 'ELSE' 0;
                INST TIME:= ('IF' DST MODE = 0 'THEN' 850 'ELSE' 1620)+INST TIME +
                        DST TIMING3A[DST MODE];
                'IF' DST MODE = 0 ^ DST = -8 'THEN' INST TIME:= INST TIME +290;
```

```
         'IF' DUPPER ^ DST MODE ≠ 0 ^ DST ≠ -8 'THEN' INST.TIME:= INST TIME + 150;
         'GOTO' EX DONE;

DECB:    EVAL ONE OPERAND;
         NDECB:= NDECB + 1;
         BYTE:= 'IF' DST OPERAND = 0 'THEN' 255 'ELSE'
                 'IF' DST OPERAND = 1 'THEN' 0 'ELSE' DST OPERAND - 1;
         V:= 'IF' BYTE = 127 'THEN' 1 'ELSE' 0;
         'GOTO' INCBX;

CLRB:    EVAL ONE OPERAND; NCLRB:= NCLRB + 1;
         BYTE:= C:= V:= 0;
         'GOTO' INCBX;

NEGB:    EVAL ONE OPERAND;
         NNEGB:= NNEGB + 1;
         BYTE:= 'IF' DST OPERAND = 0 'THEN' 0 'ELSE' 256 - DST OPERAND;
         V:= 'IF' BYTE = 128 'THEN' 1 'ELSE' 0;
         C:= 'IF' BYTE = 0 'THEN' 0 'ELSE' 1; INST TIME:= 'IF' DST MODE = 0 'THEN' 440 'ELSE' 210;
         'GOTO' INCBX;

COMB:    EVAL ONE OPERAND;
         NCOMB:= NCOMB + 1;
         BYTE:= 'IF' DST OPERAND = 255 'THEN' 0 'ELSE' 255 - DST OPERAND;
         V:= 0;
         C:= 1;
         'GOTO' INCBX;

TSTB:    EVAL ONE OPERAND;
         NTSTB:= NTSTB + 1;
         V:= C:= 0;
         BYTE:= DST OPERAND;
         'IF' DST MODE ≠ 0 'THEN' INST TIME:= -480;
         'GOTO' INCBX;

BIS:     EVAL BOTH OPERANDS;
         NBIS:= NBIS + 1;
         TMP:= OR(SRC OPERAND,DST OPERAND);
         V:= 0;
BISX:    M[DST]:= TMP;
CMPX:    N:= 'IF' TMP > MAXPOS 'THEN' 1 'ELSE' 0;
         Z:= 'IF' TMP = 0 'THEN' 1 'ELSE' 0;
         INST TIME:= INST TIME + ('IF' DST MODE = 0 'THEN' 850 'ELSE' 1620) +
                 SRC TIMING[SRC MODE] +
                 ('IF' SRC MODE = 0 'THEN' DST TIMING3A[DST MODE]
                                     'ELSE' DST TIMING3[DST MODE]);
         'IF' DST MODE = 0 ^ DST = -8 'THEN' INST TIME:= INST TIME + 290;
         'GOTO' EX DONE;

BIC:     EVAL BOTH OPERANDS;
         NBIC:= NBIC + 1;
         TMP:= 'IF' SRC OPERAND = MAXWORD 'THEN' 0 'ELSE' MAXWORD - SRC OPERAND;
         TMP:= AND(TMP,DST OPERAND);
         V:= 0;
         'GOTO' BISX;

BIT:     EVAL BOTH OPERANDS;
         NBIT:= NBIT + 1;
         TMP:= AND(SRC OPERAND,DST OPERAND);
         V:= 0;
```

```
          'IF' DST MODE ≠ 0 'THEN' INST TIME:= -480;
          'GOTO' CMPX;

CMP:      EVAL BOTH OPERANDS;
          POS SRC:= SRC OPERAND ≤ MAXPOS;
          POS DST:= DST OPERAND ≤ MAXPOS;
          NCMP:= NCMP + 1;
          DST OPERAND:= 'IF' DST OPERAND = 0 'THEN' 0 'ELSE' 65536 - DST OPERAND;
          TMP:= SRC OPERAND + DST OPERAND;
          C:= 0;
          'IF' TMP > MAXWORD 'THEN'
                  'BEGIN' C:= 1; TMP:= TMP - BIT16 'END';
          'IF' DST MODE ≠ 0 'THEN' INST TIME:= -480;
          V:= 'IF' (POS SRC ≡¬POS DST) ∧ (POS DST ≡ (TMP ≤MAXPOS)) 'THEN' 1 'ELSE' 0;
          'GOTO' CMPX;

SUB:      EVAL BOTH OPERANDS;
         SRC OPERAND := M[SRC];
         DST OPERAND := M[DST];
          POS SRC := SRC OPERAND ≤ MAXPOS;
          POS DST := DST OPERAND ≤ MAXPOS;
          NSUB:= NSUB + 1;
          SRC OPERAND := 'IF' SRC OPERAND =0 'THEN'0'ELSE'65536  - SRC OPERAND;
          C:=0;
          TMP := SRC OPERAND + DST OPERAND;
          'IF' TMP > MAXWORD 'THEN'
                   'BEGIN' C:=1;
                           TMP := 'IF' TMP=BIT16 'THEN' 0 'ELSE' TMP-BIT16
                   'END';
          V := 'IF' (POS SRC ≡ ¬POS DST) ∧ (POS SRC ≡(TMP≤MAXPOS)) 'THEN' 1 'ELSE' 0;
          'GOTO' BISX;

ADD:      EVAL BOTH OPERANDS;
          POS SRC:= SRC OPERAND ≤ MAXPOS;
          POS DST:= DST OPERAND ≤ MAXPOS;
          NADD:= NADD + 1;
         C:=0;
          TMP:= SRC OPERAND + DST OPERAND;
          'IF' TMP > MAXWORD 'THEN'
                  'BEGIN' C:= 1;
                          TMP:= 'IF' TMP = BIT16 'THEN' 0 'ELSE' TMP - BIT16
                  'END';
          V:= 'IF' (POS SRC ≡ POS DST) ∧ (POS DST ≡¬(TMP ≤MAXPOS)) 'THEN' 1 'ELSE' 0;
          'GOTO' BISX;


EXOR:     EVAL ONE OPERAND;
          NXOR:= NXOR + 1;
          TMP:= AND(XOR(DST OPERAND,M[-SRC REG -1]),MAXWORD);
          SRC MODE:= 0;
          'GOTO' BISX;
BISB:     EVAL BOTH OPERANDS;
          NBISB:= NBISB + 1;
          BYTE:= OR(SRC OPERAND,DST OPERAND);
          V:= 0;
BISBX:    'IF' DUPPER 'THEN'
                  'BEGIN' TMP1:= (WHOLE DST ÷ 256) * 256;
                          TMP:= WHOLE DST - TMP1;
                          'IF' TMP ≠ 0 'THEN' TMP:= 0;
                  'END'
```

```
                    'ELSE' TMP:= WHOLE DST ÷ 256;
            M[DST]:= 'IF' DUPPER 'THEN' 256 * BYTE + TMP 'ELSE' 256 * TMP + BYTE;
CMPBX:      N := 'IF' BYTE > 127 'THEN' 1 'ELSE' 0;
            Z:= 'IF' BYTE = 0 'THEN' 1 'ELSE' 0;
MOVX:       INST TIME:= ('IF' DST MODE = 0 'THEN' 850 'ELSE' 1620) +
                    SRC TIMING[SRC MODE] +
                    ('IF' SRC MODE = 0 'THEN' DST TIMING3A[DST MODE]
                                        'ELSE' DST TIMING3[DST MODE]);
            'IF' DST MODE = 0 ∧ DST = -8 'THEN' INST TIME:= INST TIME + 290;
            'IF' ¬ DUPPER 'THEN' 'GOTO' EXDONE;
DONE:       'IF' DST = -8 ∨ (SRC MODE = 0 ∧ DST MODE = 0) 'THEN' 'GOTO' EXDONE;
            'GOTO' EXDONE;

BICB:       EVAL BOTH OPERANDS;
            NBICB:= NBICB + 1;
            TMP:= 'IF' SRC OPERAND = 255 'THEN' 0 'ELSE' 255 - SRC OPERAND;
            BYTE:= AND(TMP,DST OPERAND);
            V:= 0;
            'GOTO' BISBX;

BITB:       EVAL BOTH OPERANDS;
            NBITB:= NBITB + 1;
            BYTE:= AND(SRC OPERAND,DST OPERAND);
            V:= 0;
            'IF' DST MODE ≠ 0 'THEN' INST TIME:= -480;
            'GOTO' CMPBX;

CMPB:       EVAL BOTH OPERANDS;
            NCMPB:= NCMPB + 1;
            POS SRC:= SRC OPERAND ≤ 127;
            POS DST:= DST OPERAND ≤ 127;
            DST OPERAND:= 'IF' DST OPERAND = 0 'THEN' 0 'ELSE' 256 - DST OPERAND;
            BYTE:= DST OPERAND + SRC OPERAND;
            C:= 0;
            'IF' BYTE > 255 'THEN'
                    'BEGIN' C:= 1; BYTE:= BYTE - 256 'END';
            'IF' DST MODE ≠ 0 'THEN' INST TIME:= -480;
            V:= 'IF' (POS SRC ≡¬POS DST) ∧ (POS DST ≡ (BYTE ≤127)) 'THEN' 1 'ELSE' 0;
            'GOTO' CMPBX;

BR:         NBR:= NBR + 1;
BRX:        TMP:= INSTRUCTION - 256 * (INSTRUCTION ÷ 256);
            PC:= M[-8] + ('IF' TMP > 127 'THEN' 2 * (TMP - 256) 'ELSE' TMP + TMP);
            M[-8]:= PC;
            'IF' PC  < 256 ∨ PC ≥ IO ADDR 'THEN' ERROR("BRANCH OFFSET ERROR");
            INST TIME:= 1140;
            'GOTO' EXDONE;

BEQ:        NBEQ:= NBEQ + 1;
            'IF' Z = 1 'THEN' 'GOTO' BRX;
NOBR:       INST TIME:= 850;
            'GOTO' EXDONE;

BNE:        NBNE:= NBNE + 1;
            'IF' Z = 0 'THEN' 'GOTO' BRX 'ELSE' 'GOTO' NOBR;

BMI:        NBMI:= NBMI + 1;
            'IF' N = 1 'THEN' 'GOTO' BRX 'ELSE' 'GOTO' NOBR;

BPL:        NBPL:= NBPL + 1;
```

```
            'IF' N = 0 'THEN' 'GOTO' BRX 'ELSE' 'GOTO' NOBR;

BCS:BLO:NBCS:= NBCS + 1;
            'IF' C = 1 'THEN' 'GOTO' BRX 'ELSE' 'GOTO' NOBR;

BCC:BHIS:NBCC:= NBCC + 1;
            'IF' C = 0 'THEN' 'GOTO' BRX 'ELSE' 'GOTO' NOBR;

BVS:        NBVS:= NBVS + 1;
            'IF' V = 1 'THEN' 'GOTO' BRX 'ELSE' 'GOTO' NOBR;

BVC:        NBVC:= NBVC + 1;
            'IF' V = 0 'THEN' 'GOTO' BRX 'ELSE' 'GOTO' NOBR;

BLT:        NBLT:= NBLT + 1;
            'IF' N+V = 1 'THEN' 'GOTO' BRX 'ELSE' 'GOTO' NOBR;

BGE:        NBGE:= NBGE + 1;
            'IF' N+V ‡ 1 'THEN' 'GOTO' BRX 'ELSE' 'GOTO' NOBR;

BLE:        NBLE:= NBLE + 1;
            'IF' (N+V = 1) ∨ (Z = 1) 'THEN' 'GOTO' BRX 'ELSE' 'GOTO' NOBR;

BGT:        NBGT:= NBGT + 1;
            'IF' (N+V ‡1) ∧ (Z = 0)'THEN' 'GOTO' BRX 'ELSE' 'GOTO' NOBR;

BHI:        NBHI:= NBHI + 1;
            'IF' (C = 0) ∧ (Z = 0) 'THEN' 'GOTO' BRX 'ELSE' 'GOTO' NOBR;

BLOS:       NBLOS:= NBLOS + 1;
            'IF' C = 1 ∨ Z = 1 'THEN' 'GOTO' BRX 'ELSE' 'GOTO' NOBR;

CLRCC:      'IF' EVEN(INSTRUCTION) ‡ 1 'THEN' C:= 0;
            'IF' EVEN(INSTRUCTION ÷ 2) ‡ 1 'THEN' V:= 0;
            'IF' EVEN(INSTRUCTION ÷ 4) ‡ 1 'THEN' Z:= 0;
            'IF' EVEN(INSTRUCTION ÷ 8) ‡ 1 'THEN' N:= 0;
            INST TIME:= 1140;
            NCC:= NCC + 1;
            'GOTO' EXDONE;

SETCC:      'IF' EVEN(INSTRUCTION) ‡ 1 'THEN' C:= 1;
            'IF' EVEN(INSTRUCTION ÷ 2) ‡ 1 'THEN' V:= 1;
            'IF' EVEN(INSTRUCTION ÷ 4) ‡ 1 'THEN' Z:= 1;
            'IF' EVEN(INSTRUCTION ÷ 8) ‡ 1 'THEN' N:= 1;
            INST TIME:= 1140;
            NCC:= NCC + 1;
            'GOTO' EXDONE;

SPL:        NSPL:= NSPL + 1;
            CPU PRIORITY:= INSTRUCTION - 152;
            'IF'CPU PRIORITY = 0 'THEN' ERROR("CPU PRIORITY SET TO 0");
            SET NEXT INTERRUPT TIME;
            INST TIME:= 1140;
            'GOTO' EXDONE;

HALT:       NLCR;
            NHALT := NHALT + 1;
            TYLIN := -1 000 000;
            PRINTTEXT("HALT INSTRUCTION EXECUTED AT LOCATION ");
            OCTAL(M[-8]-2);
            STATISTICS;

WAIT:       'IF' NEXT INTERRUPT TIME = LONG WAIT 'THEN' ERROR("NO INTERRUPT PENDING");
            IDLE:= IDLE + NEXT INTERRUPT  TIME - CLOCK;
            CLOCK:= NEXT INTERRUPT TIME;
            NWAIT:= NWAIT + 1;
            SET NEXT INTERRUPT TIME;
            'GOTO' EXDONE;

RESET:      FREE LIST HEAD:= 1;
            NREST := NREST + 1;
            'FOR' I:= 1 'STEP' 1 'UNTIL' NUM INTERRUPT SLOTS 'DO' NEXT ON CHAIN[I]:= I+1;
            NEXT ON CHAIN[NUM INTERRUPT SLOTS]:= 0;
            NEXT INTERRUPT TIME := LONG WAIT;
            'FOR' I:= 1 'STEP' 1 'UNTIL' 7 'DO' FIRST SLOT[I]:= 0;
            M[TPS ÷ 2]:= M[PRS ÷ 2]:= M[PPS ÷ 2]:= 0;
            M[TPB ÷ 2]:= M[PRB ÷ 2]:= M[PPB ÷ 2]:= 0;
            CLOCK:= CLOCK + 20 000 000.0;
            IDLE := IDLE + 20 000 000;
            'GOTO' EXDONE;

SXT:        EVAL ONE OPERAND;
            M[DST]:= 'IF' N = 0 'THEN' ZERO 'ELSE' MAXWORD;
            Z:= 'IF' N = 0 'THEN' 1 'ELSE' 0;
            NSXT:= NSXT + 1;
            'GOTO' SWAX;

RTS:        NRTS:= NRTS + 1;
            SP:= M[-7];
            'IF' EVEN(SP) ≠ 1 'THEN' ERROR("STACK POINTER ODD");
            'IF' SP + 2 > IO ADDR 'THEN' ERROR("STACK UNDERFLOW");
            M[-8]:= M[-DST REG - 1];
            M[-DST REG - 1]:= M[SP ÷ 2];
            M[-7]:= SP + 2;
            INST TIME:= 2040;
            'GOTO' EXDONE;

RTI:        NRTI:= NRTI + 1;
RTX:        SP:= M[-7];
            'IF' EVEN(SP) ≠ 1 'THEN' ERROR("STACK POINTER ODD");
            'IF' SP + 4 > IO ADDR 'THEN' ERROR("STACK UNDERFLOW");
            TMP:= SP ÷ 2;
            M[-8]:= M[TMP];
            PSW:= M[TMP + 1];
            M[-7]:= SP + 4;
            C:= PSW - 2 * (PSW ÷ 2); PSW:= PSW ÷ 2;
            V:= PSW - 2 * (PSW ÷ 2); PSW:= PSW ÷ 2;
            Z:= PSW - 2 * (PSW ÷ 2); PSW:= PSW ÷ 2;
            N:= PSW - 2 * (PSW ÷ 2); PSW:= PSW ÷ 2;
            T:= PSW - 2 * (PSW ÷ 2);
            CPU PRIORITY:= PSW ÷ 2;
            INST TIME:= 3040;
            'GOTO' EXDONE;

RTT:        NRTT:= NRTT + 1;
            'GOTO' RTX;

ILLEGAL: ERROR("ILLEGAL INSTRUCTION");

TRAP:       NTRAP:= NTRAP + 1;
```

```
        VEC:= 14;
        SWITCH PSW;
        INST TIME:= 4890;
        'GOTO' EXDONE;

EMT:    TMP := INSTRUCTION - 256 * (INSTRUCTION↓256);
        'IF' TMP > 250 'THEN'
          'BEGIN' 'SWITCH' SYSBUG := REGD, CORED, STOPTR, TRAC, PR;
                   'GOTO' SYSBUG[TMP-250];
           REGD:   USER DUMP;   'GOTO' CYCLE;
           CORED:  CORE DUMP(M[(OLDPC+2)↓2],M[(OLDPC+4)↓2]);   M[-8] :=OLDPC + 6; 'GOTO' CYCLE;
           STOPTR:  TRACE FLAG := 'FALSE'; 'GOTO' EXDONE;
           TRAC:   TRACE FLAG := 'TRUE'; 'GOTO' CYCLE;
           PR:     NLCR;  PRINTTEXT("PRINT AT LOC ");  SPACER:=6;
                   TYLIN := TYLIN + 1; 'IF' TYLIN>LINE LIMIT 'THEN' ERROR("TOO MUCH PRINTED OUTPUT");
                   OCTAL(OLDPC);  SPACER:=1;
                   TMP := M[(OLDPC+2)↓2];  'IF' TMP>16'THEN'ERROR("PRINT MAY NOT HAVE > 16 PARAMETERS");
                   'FOR' I:=1 'STEP' 1 'UNTIL' TMP 'DO'
                       'BEGIN' TMP1 := M[(OLDPC+2+2*I) ↓2];
                               'IF' TMP1>MAXCORE 'THEN' ERROR("ATTEMPT TO PRINT NONEXISTENT MEMORY LOCATION");
                               OCTAL(M[TMP1↓2]);
                       'END';
                    M[-8] := OLDPC + 2*TMP+4,
                   'GOTO' CYCLE;
          'END';
        VEC:= 12;
        NEMT:= NEMT + 1;
        SWITCH PSW;
        INST TIME:= 4890;
        'GOTO' EXDONE;

BPT:.   NBPT:= NBPT + 1;
        VEC:= 6;
        SWITCH PSW;
        INST TIME:= 4890;
        'GOTO' EXDONE;

IOT:    NIOT:= NIOT + 1;
        VEC:= 8;
        SWITCH PSW;
        INST TIME:= 4890;
        'GOTO' EXDONE;

SWAB:   EVAL ONE OPERAND;
        NSWAB:= NSWAB + 1;
        BYTE:= DST OPERAND ↓ 256;
        TMP:= DST OPERAND - 256 * BYTE;
        'IF' TMP = 0 'THEN' TMP := 0;
        M[DST]:= 256 * TMP + BYTE;
        N:= 'IF' DST OPERAND > MAXPOS 'THEN' 1 'ELSE' 0;
        Z:= 'IF' BYTE = 0 'THEN' 1 'ELSE' 0;
        V:= C:= 0;
        'GOTO' SWAX;

ADC:    EVAL ONE OPERAND;
        NADC:= NADC + 1;
        TMP:= DST OPERAND + C;
        V:= 'IF' DST OPERAND = MAXPOS ∧ C = 1 'THEN' 1 'ELSE' 0;
        C:= 'IF' DST OPERAND = MAXWORD ∧ C = 1 'THEN' 1 'ELSE' 0;
        'IF' TMP = BIT16 'THEN' TMP:= 0;
```

```
          'GOTO' INCX;

ADCB:     EVAL ONE OPERAND;
          NADC:= NADC + 1;
          BYTE:= DST OPERAND + C;
          V:= 'IF' (DST OPERAND = 127) ^ C = 1 'THEN' 1 'ELSE' 0;
          C:= 'IF' (DST OPERAND = 255) ^ C = 1 'THEN' 1 'ELSE' 0;
          'IF' BYTE = 256 'THEN' BYTE:= 0;
          'GOTO' INCBX;

SBC:      NSBC:= NSBC + 1;
          EVAL ONE OPERAND;
          TMP:= DST OPERAND - C;
          V:= 'IF' TMP = 32768 'THEN' 1 'ELSE' 0;
          C:= 'IF' TMP = 0 ^ C = 1 'THEN' 0 'ELSE' 1;
          'IF' TMP = 0 'THEN' TMP:= 0;
          'IF' TMP = -1 'THEN' TMP:= MAXWORD;
          'GOTO' INCX;

SBCB:     EVAL ONE OPERAND;
          NSBCB:= NSBCB + 1;
          BYTE:= DST OPERAND - C;
          V:= 'IF' BYTE = 128 'THEN' 1 'ELSE' 0;
          C:= 'IF' BYTE = 0 ^ C = 1 'THEN' 0 'ELSE' 1;
          'IF' BYTE = 0 'THEN' BYTE:= 0;
          'IF' BYTE = -1 'THEN' BYTE:= 255;
          'GOTO' INCBX;

ASRB:     EVAL ONE OPERAND;
          NASRB:= NASRB + 1;
          BYTE:= DST OPERAND ÷ 2 + ('IF' DST OPERAND > 127 'THEN' 128 'ELSE' 0);
ASRBX:    C:= 'IF' EVEN(DST OPERAND) = 1 'THEN' 0 'ELSE' 1;
          N:= 'IF' BYTE > 127 'THEN' 1 'ELSE' 0;
          V:= 'IF' N+C = 1 'THEN' 1 'ELSE' 0;
          'IF' DUPPER 'THEN' INST TIME:= 150;
          'GOTO' INCBX;

ASLB:     EVAL ONE OPERAND;
          NASLB:= NASLB + 1;
          BYTE:= DST OPERAND + DST OPERAND;
ASLBX:    'IF' BYTE > 255 'THEN' BYTE:= 'IF' BYTE = 256 'THEN' 0 'ELSE' BYTE - 256;
          C:= 'IF' DST OPERAND > 127 'THEN' 1 'ELSE' 0;
          N:= 'IF' BYTE > 127 'THEN' 1 'ELSE' 0;
          V:= 'IF' N+C = 1 'THEN' 1 'ELSE' 0;
          'GOTO' INCBX;

ASR:      EVAL ONE OPERAND;
          NASR:= NASR + 1;
          TMP:= DST OPERAND ÷ 2 + ('IF' DST OPERAND > MAXPOS 'THEN' 32768 'ELSE' 0);
ASRX:     N:= 'IF' TMP > MAXPOS 'THEN' 1 'ELSE' 0;
          C:= 'IF' EVEN(DST OPERAND) = 1 'THEN' 0 'ELSE' 1;
          V:= 'IF' N+C = 1 'THEN' 1 'ELSE' 0;
          'GOTO' INCX;

ASL:      EVAL ONE OPERAND;
          NASL:= NASL + 1;
          TMP:= DST OPERAND + DST OPERAND;
ASLX:     C:= 0;
          'IF' TMP > MAXWORD 'THEN'
                'BEGIN' C:= 1;
```

```
                        TMP:= 'IF' TMP = BIT16 'THEN' 0 'ELSE' TMP - BIT16
                   'END';
              N:= 'IF' TMP > MAXPOS 'THEN' 1 'ELSE' 0;
              V:= 'IF' N+C = 1 'THEN' 1 'ELSE' 0;
              'GOTO' INCX;

     ROL:     EVAL ONE OPERAND;
              NROL:= NROL + 1;
              TMP:= DST OPERAND + DST OPERAND + C;
              'GOTO' ASLX;
     ROR:     EVAL ONE OPERAND;
              NROR:= NROR + 1;
              TMP:= DST OPERAND ÷ 2;
              'IF' C = 1 'THEN' TMP:= TMP + 32768;
              'GOTO' ASRX;

     ROLB:    EVAL ONE OPERAND;
              NROLB:= NROLB + 1;
              BYTE:= DST OPERAND + DST OPERAND + C;
              'GOTO' ASLBX;

     RORB:    EVAL ONE OPERAND;
              NRORB:= NRORB + 1;
              BYTE:= DST OPERAND ÷ 2;
              'IF' C = 1 'THEN' BYTE:= BYTE + 128;
              'GOTO' ASRBX;

     SOB:     NSOB:= NSOB + 1;
              TMP:= M[-SRC REG - 1];
              TMP:= 'IF' TMP = 1 'THEN' 0 'ELSE' 'IF' TMP = 0 'THEN' MAXWORD 'ELSE' TMP - 1;
              'IF' TMP ≠ 0 'THEN' M[-8]:= M[-8] - 16 * DST MODE - 2* DST REG;
              M[-SRC REG - 1] := TMP;
              INST TIME:= 'IF' TMP ≠ 0 'THEN' 1140 'ELSE' 1290;
              'GOTO' EXDONE;
     MARK:    NMARK:= NMARK + 1;
              SP:= M[-7] + 16 * DST MODE + 2 * DST REG + 2;
              M[-8]:= M[-6];
              'IF' SP ≥ IO ADDR - 2 'THEN' ERROR("STACK UNDERFLOW");
              M[-6]:= M[SP ÷ 2];
              M[-7]:= SP + 2;
              INST TIME:= 1990;
              'GOTO' EXDONE;

     MUL:     EVAL ONE OPERAND;
              NMUL:= NMUL + 1;
              SRC OPERAND:= M[-SRC REG - 1];
              SUPPER:= DUPPER:= 'TRUE';
              'IF' SRC OPERAND > MAXPOS 'THEN'
                   'BEGIN' SUPPER:= 'FALSE'; SRC OPERAND:= 65536 - SRC OPERAND 'END';
              'IF' DST OPERAND >MAXPOS 'THEN'
                   'BEGIN' DUPPER:= 'FALSE'; DST OPERAND:= 65536 - DST OPERAND 'END';
              MUL1:= SRC OPERAND;
              MUL2:= DST OPERAND;
              MUL3:= MUL1 * MUL2;
              N:= 'IF' SUPPER ≡ DUPPER 'THEN' 0 'ELSE' 1;
              Z:= 'IF' MUL3 = 0 'THEN' 1 'ELSE' 0;
              V:= 0;
              C:= 'IF' (MUL3 > 32768) ∨ ((N=0) ∧ MUL3 = 32768) 'THEN' 1 'ELSE' 0;
              TMP1:= MUL3 ÷ 65536;
              TMP2:= MUL3 - 65536 * TMP1;
```

```
            'IF' N = 1 'THEN'
                    'BEGIN' TMP1:= MAXWORD - TMP1;
                            TMP2:= 65536 - TMP2;
                            'IF' TMP2 ≥ BIT16 'THEN'
                            'BEGIN' TMP2:= TMP2 - BIT16;
                                    TMP1:= TMP1 + 1;
                                    'IF' TMP1=BIT16 'THEN' TMP1:=0;
                            'END';
                    'END';
            'IF' TMP1 = 0 'THEN' TMP1:= 0;
            'IF' TMP2 = 0 'THEN' TMP2:= 0;
            'IF' EVEN(SRC REG) = 1 'THEN'
                    'BEGIN' M[-SRC REG - 2]:= TMP2;
                            M[-SRC REG -1]:= TMP1
                    'END'
            'ELSE' M[-SRC REG - 1]:= TMP2;
            INST TIME:= 3840 + ('IF' DST MODE = 0 'THEN' DST TIMING3A[DST MODE]
                                                  'ELSE' DST TIMING3[DST MODE]);
            'GOTO' EXDONE;

DIV:        EVAL ONE OPERAND;
            Q1 := M[-SRC REG-1];  Q2 := M[-SRC REG -2];
            NDIV:= NDIV + 1;
            'IF' DST OPERAND = 0 'THEN'
                    'BEGIN' C:= V:= 1; INST TIME:= 1440; 'GOTO' EXDONE 'END';
            K:= 'IF' EVEN(SRC REG) = 1 'THEN' -SRC REG -2 'ELSE' -SRC REG - 1;
            TMP2 := M[K];
            TMP1:= M[-SRC REG - 1];
            III:= 0;
            'IF' TMP1 > MAXPOS 'THEN'
                    'BEGIN' TMP2:= 65536 - TMP2;
                            TMP1:= MAXWORD - TMP1;
                            'IF' TMP2 = BIT16 'THEN'
                                    'BEGIN' TMP2:= 0;
                                            TMP1:= TMP1 + 1;
                                    'END';
                            III:= 1;
                    'END';
            MUL1:= TMP1;
            MUL2:= TMP2;
            MUL3:= 65536.0 * MUL1 + MUL2;
            J:= 'IF' DST OPERAND > MAXPOS 'THEN' 65536 - DST OPERAND 'ELSE'DST OPERAND;
            V:= 'IF' TMP1 > J 'THEN' 1 'ELSE' 0;
            C:= 0;
            TMP:= MUL3 ÷ J; 'IF' TMP = 0 'THEN' TMP:= 0;
            I:= 'IF' DST OPERAND > MAXPOS 'THEN' 1 'ELSE' 0;
            II:= MUL3 - TMP * J;
            'IF' II = 0 'THEN' II:= 0;
            'IF' TMP > MAXWORD 'THEN' TMP:= AND(TMP,MAXWORD);
            'IF' III + I = 1 'THEN' TMP:= 'IF' TMP = 0 'THEN' 0 'ELSE' 65536 - TMP;
            'IF' III = 1 'THEN' II:= 'IF' II = 0 'THEN' 0 'ELSE' 65536 - II;
            M[-SRC REG - 1]:= TMP;
            'IF' K = -SRC REG - 2 'THEN' M[K]:= II;
            N:= 'IF' TMP > MAXPOS 'THEN' 1 'ELSE' 0;
            Z:= 'IF' TMP = 0 'THEN' 1 'ELSE' 0;
            INST TIME:= 'IF' I + II = 1 'THEN' 8640 'ELSE' 7740;
            'IF'V=1 'THEN' 'BEGIN' M[-SRC REG-1]:=Q1; M[-SRC REG-2]:=Q2'END';
            'GOTO' EXDONE;

ASH:        EVAL ONE OPERAND;
```

```
        TMP:= M[-SRC REG - 1];
        NASH:= NASH + 1;
        V:= 0;
        TMP2:= 'IF' TMP > MAXPOS 'THEN' 32768 'ELSE' 0;
        TMP1:= DST OPERAND - 64 * (DST OPERAND ÷ 64) ;
        'IF' TMP1 > 31 'THEN' TMP1 := TMP1 - 64;
        'IF' TMP1 < 0 'THEN'
                'BEGIN' 'FOR' I:= 1 'STEP' 1 'UNTIL' -TMP1 'DO'
                        'BEGIN' C:= 'IF' EVEN(TMP) ≠ 1 'THEN' 1 'ELSE' 0;
                                TMP:= TMP ÷ 2 + TMP2
                        'END'
                'END'
        'ELSE'
        'FOR' I:= 1 'STEP' 1 'UNTIL' TMP1 'DO'
                'BEGIN' TMP2:= TMP + TMP;
                        C:= 0;
                        'IF' TMP2 > MAXWORD 'THEN'
                                'BEGIN' TMP2:= TMP2 - BIT16;
                                        'IF' TMP2 = 0 'THEN' TMP2:= 0;
                                        C:= 1
                                'END';
                        'IF' (TMP2 > MAXPOS ∧ TMP ≤ MAXPOS)
                        ∨ (TMP2 ≤ MAXPOS ∧ TMP > MAXPOS) 'THEN' V:= 1;
                        TMP:= TMP2
                'END';
        M[-SRC REG - 1]:= TMP;
        N:= 'IF' TMP > MAXPOS 'THEN' 1 'ELSE' 0;
        Z:= 'IF' TMP = 0 'THEN' 1 'ELSE' 0;
        INST TIME:= 1440 + 150 * ABS(TMP1) + DST TIMING3A[DST MODE];
        'IF' DST OPERAND = 0 'THEN' INST TIME:= INST TIME - 150;
        'GOTO' EXDONE;

ASHC:   EVAL ONE OPERAND;
        NASHC:= NASHC + 1;
        II:= 'IF' TMP1 > MAXPOS 'THEN' 32768 'ELSE' 0;
        J:= DST OPERAND - 64 * (DST OPERAND ÷ 64) ;
        'IF' J > 31 'THEN' J := J - 64;
        TMP1:= M[-SRC REG - 1];
        K:= 'IF' EVEN(SRC REG) = 1 'THEN' -SRC REG - 2 'ELSE' -SRC REG - 1;
        TMP2:= M[K];
        V:= 0;
        'IF' J < 0 'THEN'
                'BEGIN' 'FOR' I:= 1 'STEP' 1 'UNTIL' -J 'DO'
                        'BEGIN' C:= 'IF' EVEN(TMP2) ≠ 1 'THEN' 1 'ELSE' 0;
                                TMP2:= TMP2 ÷ 2;
                                'IF' EVEN(TMP1) ≠ 1 'THEN' TMP2:= TMP2 + 32768;
                                TMP1:= TMP1 ÷ 2 + II
                        'END'
                'END'
        'ELSE'
        'FOR' I:= 1 'STEP' 1 'UNTIL' J 'DO'
                'BEGIN' TMP3:= TMP1 + TMP1;
                        C:= 0;
                        'IF' TMP3 > MAXWORD 'THEN'
                                'BEGIN' TMP3:= TMP3 - BIT16;  C:=1;
                                        'IF' TMP3 = 0 'THEN' TMP3:= 0;
                                'END';
                        'IF' (TMP1 > MAXPOS ∧ TMP3 ≤ MAXPOS) ∨ (TMP1  ≤ MAXPOS ∧ TMP3 > MAXPOS) 'THEN' V := 1;
                                TMP1:= 'IF' TMP2 > MAXPOS 'THEN' TMP3 + 1 'ELSE' TMP3;
                                TMP2:= TMP2 + TMP2;
                                'IF' TMP2 > MAXWORD 'THEN'
                                        'BEGIN' TMP2:= TMP2 - BIT16;
                                                'IF' TMP2 = 0 'THEN' TMP2:= 0;
                                        'END'
                'END';
        M[-SRC REG - 1]:= TMP1;
        M[K]:= TMP2;
        Z:= 'IF' TMP1 = 0 ^ TMP2 = 0 'THEN' 1 'ELSE' 0;
        N:= 'IF' TMP1 > MAXPOS 'THEN' 1 'ELSE' 0;
        INST TIME:= 1440 + 150 * ABS(J) + DST TIMING3A[DST MODE];
        'IF' DST OPERAND = 0 'THEN' INST TIME:= INST TIME - 150;
        'GOTO' EXDONE;

JMP:    EVAL ONE OPERAND;
        'IF' DST MODE = 0 'THEN' ERROR("JUMP TO REGISTER");
        'IF' DST MODE = 2 'THEN' DST ADDR:= DST ADDR + 2;
        M[-8]:= DST ADDR;
        'IF' DST ADDR > IO ADDR 'THEN' ERROR("JUMP TO IO AREA");
        INST TIME := 1140 + DST TIMING1[DST MODE];
        NJMP:= NJMP + 1;
        'GOTO' EXDONE;

JSR:    EVAL ONE OPERAND;
        NJSR := NJSR + 1;
        'IF' DST MODE = 0 'THEN' ERROR("JSR TO REGISTER");
        'IF' DST MODE = 2 'THEN' DST ADDR:= DST ADDR + 2;
        SP:= M[-7];
        'IF' EVEN(SP) ‡ 1 'THEN' ERROR("STACK POINTER ODD");
        'IF' SP < STACK LIMIT + 2 'THEN' ERROR("STACK OVERFLOW");
        SP:= SP - 2;
        M[SP; 2] := M[-SRC REG - 1];
        M[-7]:= SP;
        M[-SRC REG-1] := M[-8];
        M[-8]:= DST ADDR;
        INST TIME:= 2290 + DST TIMING1[DST MODE];
        'GOTO' EXDONE;
'END'
'END'
```

```
ERROR MESSAGES    ADDRESS   INST    IMMED1  IMMED2    SOURCE STATEMENT

                                                 WILLIAM WARTHOG, ALGOL IDENTIFIER PROBLEM
                                                 R0=↑0
                                                 R1=↑1
                                                 R2=↑2
                                                 R3=↑3
                                                 R4=↑4
                                                 R5=↑5
                                                 SP=↑6
                                                 PC=↑7

                  000400  012767  000101  077372  RESYM:   MOV #65.,PRS   ;READ ROUTINE
                  000406  000001                           WAIT           ;WAIT UNTIL  THE CHAR HAS BEEN READ
                  000410  000207                           RTS PC         ;RETURN

                  000412  010067  077374          PRSYM:   MOV R0,TPB     ;PRINT ROUTINE
                  000416  000001                           WAIT           ;WAIT UNTIL THE CHARACTER HAS BEEN PRINTED
                  000420  000207                           RTS PC         ;RETURN

                  000422  016700  077354          RIO:      MOV PRB,R0
                  000426  042700  000200                   BIC #128.,R0   ;CLEAR PARITY BIT
                  000432  000002                           RTI            ;RETURN FROM INTERRUPT

                  000434  000002                  PIO:     RTI            ;RETURN FROM PRINTER INTERRUPT

                  000436  012767  000200  177426  START:   MOV #128.,58.  ;SET UP THE READER INTERRUPT VECTOR
                  000444  012767  000422  177416           MOV #RIO,56.   ;"
                  000452  012767  000200  177406           MOV #128.,54.  ;SET UP THE PRINTER INTERRUPT VECTOR
                  000460  012767  000434  177376           MOV #PIO,52.   ;"
                  000466  012706  001750                   MOV #1000.,SP  ;SET UP THE STACK POINTER


                                                  ;THIS PROGRAM READS A SERIES OF IDENTIFIERS AND INDICATES WHICH
                                                  ;ONES ARE NOT VALID ALGOL IDENTIFIERS BY TYPING A QUESTION MARK
                                                  ;AFTER THE DEFECTIVE ONES.

                  000472  004767  177702          MAIN:    JSR PC,RESYM   ;READ A CHAR INTO R0
                  000476  020027  000056                   CMP R0,#46.    ;IS IT A PERIOD?
                  000502  001001                           BNE MORE       ;IF IT IS NOT A PERIOD, CONTINUE
                  000504  000000                           HALT           ;THIS SHOULD BE OBVIOUS
                  000506  004767  000172          MORE:    JSR PC,LETTER  ;THE SUBROUTINE LETTER TELLS IF IT IS A LETTER
                  000512  020127  000001                   CMP R1,#1.     ;IF IT IS A LETTER, IT SETS R1=1
                  000516  001042                           BNE ERROR      ;R1=0 MEANS IT WAS NOT A LETTER,THUS NOT AN IDENTIFIER
                  000520  004767  177666                   JSR PC,PRSYM   ;PRINT IT

                  000524  004767  177650          GET:     JSR PC,RESYM   ;GET THE NEXT CHARACTER
                  000530  020027  000040                   CMP R0,#32.    ; IS IT A SPACE
                  000534  001773                           BEQ GET        ;IF SO, IGNORE IT
                  000536  004767  000142                   JSR PC,LETTER  ;IS IT A LETTER?
                  000542  020127  000001                   CMP R1,#1.     ;CHECK RESULT
                  000546  001412                           BEQ OK         ;IF IT IS A LETTER, GOTO OK
                  000550  004767  000162                   JSR PC,DIGIT   ;IT IS NOT A LETTER,  IS IT A DIGIT?
                  000554  020127  000001                   CMP R1,#1.     ;CHECK RESULT OF THE CALL TO DIGIT
                  000560  001405                           BEQ OK         ;IF IT IS A DIGIT, ALL IS FINE
                  000562  020027  000015                   CMP R0,#13.    ;IS THE CHAR A CARRIAGE RETURN?
                  000566  001406                           BEQ ISID       ;IF SO, THIS IS AN ALGOL IDENTIFIER
                  000570  000167  000030                   JMP ERROR      ;IT IS NOT LETTER,DIGIT OR CARRAIGE RET. THUS ERROR

                  000574  004767  177612          OK:      JSR PC,PRSYM   ;PRINT THE CHAR
```

```
000600  000167  177720                   JMP GET       ;CONTINUE READING

000604  004767  177602          ISID:    JSR PC,PRSYM  ;TYPE OF THE CARRIAGE RETURN
000610  004767  177564                   JSR PC,RESYM  ;READ THE NEXT CHAR WHICH IS ALWAYS LINE FEED
000614  004767  177572                   JSR PC,PRSYM  ;TYPE OUT THE LINE FEED
000620  000167  177646                   JMP MAIN      ;CHECK THE NEXT IDENTIFIER

000624  004767  177562          ERROR:   JSR PC,PRSYM  ;TYPE THE CHARACTER
000630  004767  177544                   JSR PC,RESYM  ;GET THE NEXT ONE
000634  020027  000040                   CMP R0,#32.
000640  001773                           BEQ ERROR+4
000642  020027  000015                   CMP R0,#13.   ;IS IT A CARRIAGE RETURN?
000646  001366                           BNE ERROR     ;IF NOT, GET ANOTHER. I.E. SKIP UNTIL CARRAIGE RETURN
000650  012700  000077                   MOV #63.,R0   ;63 IS QUESTION MARK
000654  004767  177532                   JSR PC,PRSYM  ;TYPE THE QUESTION MARK
000660  012700  000015                   MOV #13.,R0   ;PUT A CARRIAGE RETURN IN R0
000664  004767  177522                   JSR PC,PRSYM  ;TYPE THE CARRIAGE RETURN
000670  004767  177504                   JSR PC,RESYM  ;READ THE LINE FEED
000674  004767  177512                   JSR PC,PRSYM  ;TYPE THE LINE FEED
000700  000167  177566                   JMP MAIN      ;START ON NEXT LINE

000704  012701  000001          LETTER:  MOV #1.,R1    ;TENTATIVELY RESULT IS TRUE
000710  020027  000140                   CMP R0,#96.   ;COMPARE TO A
000714  003002                           BGT L1        ;IT SHOULD BRANCH HERE
000716  012701  000000                   MOV #0.,R1    ;SET RESULT TO FALSE
000722  020027  000172          L1:      CMP R0,#122.  ;COMPARE TO Z
000726  003402                           BLE L2        ;SHOULD BRANCH HERE TOO
000730  012701  000000                   MOV #0.,R1    ;SET RESULT TO FALSE
000734  000207                  L2:      RTS PC        ;RETURN

000736  012701  000001          DIGIT:   MOV #1.,R1    ;TENTATIVE RESULT IS TRUE
000742  020027  000060                   CMP R0,#48.   ;COMPARE TO 0
000746  002002                           BGE D1        ;SHOULD BRANCH
000750  012701  000000                   MOV #0.,R1    ;SET RESULT TO FALSE
000754  020027  000072          D1:      CMP R0,#58.   ;COMPARE TO 9
000760  002402                           BLT D2        ;SHOULD BRANCH
000762  012701  000000                   MOV #0.,R1    ;SET RESULT TO FALSE
000766  000207                  D2:      RTS PC        ;RETURN
000436                                   .END START

0 ERRORS IN ABOVE ASSEMBLY             23 SECONDS ASSEMBLY TIME
```

```
SUZANNE
WILLIAMUFORANGE
TESTNAME.?
NOT-VALID?
OBVIOUSLYVALIDNAME
INPUT@
GAMMA21U
2XX?
X2X
PIGISDANGEROUS
A+B?
$1CU?
THAT'SALLFOLKS?

HALT INSTRUCTION EXECUTED AT LOCATION 000504


CORE DUMP
000000   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000040   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000434 000200   000422 000200 000000 000000
000100   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000140   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000200   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000240   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000300   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000340   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000400   012767 000101 077372 000001   000207 010067 077374 000001   000207 016700 077354 042700   000200 000002 000002 012767
000440   001200 177426 012767 000422   177416 012767 000200 177406   012767 000434 177376 012706   001750 004767 177702 020027
000500   001056 001001 000000 004767   000172 020127 000001 001042   004767 177666 004767 177650   020027 000040 001773 004767
000540   001142 020127 000001 001412   004767 000162 020127 000001   001405 020027 000015 001406   000167 000030 004767 177612
000600   001167 177720 004767 177602   004767 177564 004767 177572   000167 177646 004767 177562   004767 177544 020027 000040
000640   001773 020027 000015 001366   012700 000077 004767 177532   012700 000015 004767 177522   004767 177504 004767 177512
000700   001167 177566 012701 000001   020027 000140 003002 012701   000000 020027 000172 003402   012701 000000 000207 012701
000740   001001 020027 000060 002002   012701 000060 020027 000072   002402 012701 000000 000207   000000 000000 000000 000000
001000   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001040   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001100   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001140   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001200   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001240   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001300   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001340   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001400   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001440   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001500   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001540   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001600   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001640   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001700   001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001740   001000 000410 000040 000476   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000


SIMULATION STATISTICS

PDP TIME USED =      8419.725  MILLISECONDS        X8 TIME USED =      84960.000  MILLISEC.        X8 TIME PER PDP SECOND =        10
PDP WAIT TIME =      8390.000  MILLISECONDS       NUMBER OF INTERRUPTS =      1067            11405  INSTRUCTIONS EXECUTED
```

```
NUMBER OF CHARACTERS READ =     1067            NUMBER OF LINES TYPED =     13          NUMBER OF CHARACTERS PUNCHED =        0
AVERAGE PDP TIME PER INSTRUCTION =     2606  NANOSECONDS
AVERAGE NUMBER OF PDP INSTRUCTIONS PER SECOND OF X8 TIME =      134
DIST OF ADDR MODES 0-7     3871      0     3730       0        0        0     3703        0         IMMEDIATE=    3730      DIRECT=        0     PIC=    3703

PROGRAM COUNTER DISTRIBUTION.   ADDRESS (OCTAL)      FREQUENCY OF USE (DECIMAL)
BELOW 000401          0            ABOVE 001000         0
000400      1067     000402         0     000404        0     000406     1067     000410     1067     000412      145     000414        0     000416      145
000420      145      000422      1067     000424        0     000426     1067     000430        0     000432     1067     000434        0     000436        1
000440        0      000442         0     000444        1     000446        0     000450        0     000452        1     000454        0     000456        0
000460        1      000462         0     000464        0     000466        1     000470        0     000472       14     000474        0     000476       14
000500        0      000502        14     000504        1     000506       13     000510        0     000512       13     000514        0     000516       13
000520       11      000522         0     000524      577     000526        0     000530      577     000532        0     000534      577     000536       87
000540        0      000542        87     000544        0     000546       87     000550       15     000552        0     000554       15     000556        0
000560       15      000562        11     000564        0     000566       11     000570        4     000572        0     000574       76     000576        0
000600       76      000602         0     000604        7     000606        0     000610        7     000612        0     000614        7     000616        0
000620        7      000622         0     000624       26     000626        0     000630      463     000632        0     000634      463     000636        0
000640      463      000642        26     000644        0     000646       26     000650        6     000652        0     000654        6     000656        0
000660        6      000662         0     000664        6     000666        0     000670        6     000672        0     000674        6     000676        0
000700        6      000702         0     000704      100     000706        0     000710      100     000712        0     000714      100     000716       17
000720        0      000722       100     000724        0     000726      100     000730        0     000732        0     000734      100     000736       15
000740        0      000742        15     000744        0     000746       15     000750       11     000752        0     000754       15     000756        0
000760       15      000762         0     000764        0     000766       15     000770        0     000772        0     000774        0     000776        0

ADC        0      .0     ADCB       0      .0     ADD        0      .0     ASL        0      .0     ASLB       0      .0     ASH        0      .0
ASHC       0      .0     ASR        0      .0     ASRB       0      .0     BCC        0      .0     BCS        0      .0     BEQ     1153    10.1
BGE       15      .1     BGT      100      .9     BHI        0      .0     BIC     1067     9.4     BICB       0      .0     BIS        0      .0
BISB       0      .0     BIT        0      .0     BITB       0      .0     BLT       15      .1     BLE      100      .9     BLOS       0      .0
BMI        0      .0     BNE       53      .5     BPL        0      .0     BPT        0      .0     BR         0      .0     BVC        0      .0
BVS        0      .0     CLR        0      .0     CLRB       0      .0     CMP     1436    12.6     CMPB       0      .0     COM        0      .0
COMB       0      .0     CC         0      .0     DEC        0      .0     DECB       0      .0     DIV        0      .0     EMT        0      .0
HALT       1      .0     INC        0      .0     INCB       0      .0     IOT        0      .0     JMP       93      .8     JSR     1327    11.6
MARK       0      .0     MOV     2439    21.4     MOVB       0      .0     MUL        0      .0     NEG        0      .0     NEGB       0      .0
RESET      0      .0     ROL        0      .0     ROLB       0      .0     ROR        0      .0     RORB       0      .0     RTI     1067     9.4
RTS     1327    11.6     RTT        0      .0     SBC        0      .0     SBCB       0      .0     SOB        0      .0     SPL        0      .0
SUB        0      .0     SWAB       0      .0     SXT        0      .0     TRAP       0      .0     TST        0      .0     TSTB       0      .0
WAIT    1212    10.6     XOR        0      .0

 TOTAL TIME FOR THIS JOB =     121 SECONDS.
```

```
END OF JOB
```

```
ERROR MESSAGES    ADDRESS   INST    IMMED1  IMMED2    SOURCE STATEMENT

                                                    YOUR NAME, ROTATION CIPHER PROBLEM
                                                    R0=↑0
                                                    R1=↑1
                                                    R2=↑2
                                                    R3=↑3
                                                    R4=↑4
                                                    R5=↑5
                                                    SP=↑6
                                                    PC=↑7

                  000400    012767  000101  077372  RESYM:   MOV #65.,PRS  ;READ ROUTINE
                  000406    000001                           WAIT          ;WAIT UNTIL THE CHARACTER HAS BEEN READ
                  000410    000207                           RTS PC        ;RETURN

                  000412    010067  077374          PRSYM:   MOV R0,TPB    ;PRINT ROUTINE
                  000416    000001                           WAIT          ;WAIT UNTIL THE CHARACTER HAS BEEN PRINTED
                  000420    000207                           RTS PC        ;RETURN

                  000422    016700  077354          RIO:     MOV PRB,R0    ;READER INTERRUPT SERVICE ROUTINE
                  000426    042700  000200                   BIC #128.,R0  ;CLEAR PARITY BIT
                  000432    000002                           RTI           ;RETURN FROM INTERRUPT

                  000434    000002                  PIO:     RTI           ;RETURN FROM PRINTER INTERRUPT

                  000436    012767  000200  177426  START:   MOV #128.,58. ;SET UP THE READER INTERRUPT VECTOR
                  000444    012767  000422  177416           MOV #RIO,56.  ;"
                  000452    012767  000200  177406           MOV #128.,54. ;SET UP THE PRINTER INTERRUPT VECTOR
                  000460    012767  000434  177376           MOV #PIO,52.  ;"
                  000466    012706  001750                   MOV #1000.,SP ;SET UP THE STACK POINTER


                                                    ;THIS PROGRAM BREAKS CODES, JUST LIKE JAMES BOND.  IN PARTICULAR IT
                                                    ;SPECIALIZES IN ROTATION CIPHERS.


                  000472    004767  177702          LOOP:    JSR PC,RESYM  ;READ A CHARACTER INTO R0
                  000476    020027  000015                   CMP R0,#13.   ;IS THE CHAR A CARRIAGE RETURN?
                  000502    001420                           BEQ DONE      ;IF IT IS GO TO DONE
                  000504    162700  000001                   SUB #1.,R0    ;SUBTRACT 1 FROM R0. THIS MAKES B INTO A, ETC.
                  000510    020027  000140                   CMP R0,#96.   ;IF R0=96, IT WAS ORIGINALLY THE LETTER A
                  000514    001002                           BNE OK        ;IF IT WASN'T AN A, ALL IS FINE
                  000516    012700  000172                   MOV #122.,R0  ;IT WAS AN A.  PUT Z IN R0
                  000522    020027  000037          OK:      CMP R0,#31.    ;WAS THE CHAR A SPACE?
                  000526    001002                           BNE OK2       ;IF IT IS NOT A SPACE, GO TO OK2
                  000530    062700  000001                   ADD #1.,R0    ;IT WAS A SPACE.  FIX IT
                  000534    004767  177652          OK2:     JSR PC,PRSYM  ;PRINT THE CHARACTER
                  000540    000167  177726                   JMP LOOP      ;GO GET THE NEXT CHAR

                  000544    000000                  DONE:    HALT          ;NO COMMENT
                  000436                                     .END START

 0 ERRORS IN ABOVE ASSEMBLY          14 SECONDS ASSEMBLY TIME
```

```
DE PDP IS HEEL LIEF
HALT INSTRUCTION EXECUTED AT LOCATION 000544


CORE DUMP
000000    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000040    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000434 000200   000422 000200 000000 000000
000100    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000140    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000200    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000240    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000300    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000340    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000400    012767 000101 077372 000001   000207 010067 077374 000001   000207 016700 077354 042700   000200 000002 000002 012767
000440    001200 177426 012767 000422   177416 012767 000200 177406   012767 000434 177376 012706   001750 004767 177702 020027
000500    001015 001420 162700 000001   020027 000140 001002 012700   000172 020027 000037 001002   062700 000001 004767 177652
000540    001167 177726 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000600    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000640    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000700    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
000740    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001000    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001040    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001100    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001140    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001200    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001240    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001300    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001340    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001400    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001440    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001500    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001540    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001600    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001640    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001700    001000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000
001740    001000 000410 000040 000476   000000 000000 000000 000000   000000 000000 000000 000000   000000 000000 000000 000000



SIMULATION STATISTICS

PDP TIME USED =        2940.214  MILLISECONDS          X8 TIME USED =      11690.000  MILLISEC.          X8 TIME PER PDP SECOND =          4
PDP WAIT TIME =        2936.667  MILLISECONDS         NUMBER OF INTERRUPTS =          81            1600  INSTRUCTIONS EXECUTED
NUMBER OF CHARACTERS READ =        81            NUMBER OF LINES TYPED =      0            NUMBER OF CHARACTERS PUNCHED =         0
AVERAGE PDP TIME PER INSTRUCTION =       2217  NANOSECONDS
AVERAGE NUMBER OF PDP INSTRUCTIONS PER SECOND OF X8 TIME =       137
DIST OF ADDR MODES 0-7       629         0       553         0         0         0       487         0       IMMEDIATE=     553     DIRECT=         0     PIC=     487

PROGRAM COUNTER DISTRIBUTION,    ADDRESS (OCTAL)      FREQUENCY OF USE (DECIMAL)
BELOW 000400        0             ABOVE 001000        0
000400        81     000402        0     000404        0     000406       81     000410       81     000412       80     000414        0     000416       80
000420        80     000422       81     000424        0     000426       81     000430        0     000432       81     000434        0     000436        1
000440         0     000442        0     000444        1     000446        0     000450        0     000452        1     000454        0     000456        0
000460         1     000462        0     000464        0     000466        1     000470        0     000472       81     000474        0     000476       81
000500         0     000502       81     000504       80     000506        0     000510       80     000512        0     000514       80     000516        0
000520         0     000522       80     000524        0     000526       80     000530       65     000532        0     000534       80     000536        0
```

```
000540      80   000542       0   000544       1   000546       0   000550       0   000552       0   000554       0   000556       0
000560       0   000562       0   000564       0   000566       0   000570       0   000572       0   000574       0   000576       0
000600       0   000602       0   000604       0   000606       0   000610       0   000612       0   000614       0   000616       0
000620       0   000622       0   000624       0   000626       0   000630       0   000632       0   000634       0   000636       0
000640       0   000642       0   000644       0   000646       0   000650       0   000652       0   000654       0   000656       0
000660       0   000662       0   000664       0   000666       0   000670       0   000672       0   000674       0   000676       0
000700       0   000702       0   000704       0   000706       0   000710       0   000712       0   000714       0   000716       0
000720       0   000722       0   000724       0   000726       0   000730       0   000732       0   000734       0   000736       0
000740       0   000742       0   000744       0   000746       0   000750       0   000752       0   000754       0   000756       0
000760       0   000762       0   000764       0   000766       0   000770       0   000772       0   000774       0   000776       0


ADC        0      .0    ADCB      0     .0    ADD      65    4.1    ASL       0     .0    ASLB      0     .0    ASH       0     .0
ASHC       0      .0    ASR       0     .0    ASRB      0     .0    BCC       0     .0    BCS       0     .0    BEQ      81    5.1
BGE        0      .0    BGT       0     .0    BHI       0     .0    BIC      81    5.1    BICB      0     .0    BIS       0     .0
BISB       0      .0    BIT       0     .0    BITB      0     .0    BLT       0     .0    BLE       0     .0    BLOS      0     .0
BMI        0      .0    BNE     160   10.0    BPL       0     .0    BPT       0     .0    BR        0     .0    BVC       0     .0
BVS        0      .0    CLR       0     .0    CLRB      0     .0    CMP     241   15.1    CMPB      0     .0    COM       0     .0
COMB       0      .0    CC        0     .0    DEC       0     .0    DECB      0     .0    DIV       0     .0    EMT       0     .0
HALT       1      .1    INC       0     .0    INCB      0     .0    IOT       0     .0    JMP      80    5.0    JSR     161   10.1
MARK       0      .0    MOV     247   15.4    MOVB      0     .0    MUL       0     .0    NEG       0     .0    NEGB      0     .0
RESET      0      .0    ROL       0     .0    ROLB      0     .0    ROR       0     .0    RORB      0     .0    RTI      81    5.1
RTS      161    10.1    RTT       0     .0    SBC       0     .0    SBCB      0     .0    SOB       0     .0    SPL       0     .0
SUB       80     5.0    SWAB      0     .0    SXT       0     .0    TRAP      0     .0    TST       0     .0    TSTB      0     .0
WAIT     161    10.1    XOR       0     .0

 TOTAL TIME FOR THIS JOB =     39 SECONDS.
```

```
END OF JOB
END OF RUN.            2 JOBS PROCESSED.
```

```
THE PROGRAM LOADS 4 TABLES AT THE START OF EACH RUN.  THEY ARE:
1.   TIMING TABLE
2.   MC TO ASCII CODE CONVERSION
3.   ASCII TO MC CODE CONVERSION
4.   ASSEMBLER OPCODE TABLE (DEFINING MOV, CMP, .WORD, ETC)



    0   840    840  1830    990  1980  1600  2590
    0   300    300  1050    300  1200  1000  1990
  150   850    850  1840    990  2080  1900  2890
  150   760    760  1660    900  1800  1660  2560
    0   940    940  1830    990  2080  1750  2740
    0   840    840  1650    900  1800  1510  2410


 48   49   50   51   52   53   54   55   56   57   97   98   99  100  101  102
103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118
119  120  121  122  000  065  066  067  068  069  070  071  072  073  074  075
076  077  078  079  080  081  082  083  084  085  086  087  088  089  090  000
043  045  042  047  000  094  061  000  060  000  062  000  126  000  096  125
123  000  000  000  000  000  000  044  046  092  058  059  000  032  000  000
000  000  040  041  091  093  000  000  000  000  000  000  000  000  000  000
000  000  000  000  000  000  009  013  039  034  063  038  000  035  095  124
064  033  000  000  037  036  013  010


122  122  122  122  122  122  122  122  122  118  135  122  122  134  122  122
122  122  122  122  122  122  122  122  122  122  122  122  122  122  122  122
093  129  121  125  133  132  123  120  098  099  066  064  087  065  088  067
000  001  002  003  004  005  006  007  008  009  090  091  072  070  074  122
 28  037  038  039  040  041  042  043  044  045  046  047  048  049  050  051
052  053  054  055  056  057  058  059  060  061  062  100  089  101  069  126
078  010  011  012  013  014  015  016  017  018  019  020  021  022  023  024
025  026  027  028  029  030  031  032  033  034  035  080  127  079  076  122


        0         0         0         0         0         0         0         0
        0         0   2838527   5145856         0         0  32100848    821501
        0         0   2215935   5146368         0         0   2533375   5144832
        0         0         0         0  20614143   9101312         0         0
        0         0   2395231   9093120  14332927   9596928   2506367   5145344
        0         0         0         0         0         0         0         0
  2396159   9060352         0         0         0         0         0         0
        0         0         0         0   2719743   5112064  19539967    786434
  2369121   5146112         0         0         0         0         0         0
        0         0         0         0   3566591    786596         0         0
 20025343    786623         0         0  14851167   4950784         0         0
        0         0  21614687   4950976         0         0         0         0
        0         0  14852095   4918016  31509609  13041664         0         0
 21615615   4918208         0         0   1184767   9068544         0         0
        0         0         0         0         0         0         0         0
        0         0   9950207    786436         0         0         0         0
        0         0         0         0         0         0   3562495    786594
```

```
         0         0   4361311   4950720         0         0         0         0
         0         0         0         0         0         0         0         0
   4362239   4917952   2634751   5144576         0         0         0         0
         0         0  13683071   5311744         0         0         0         0
         0         0   2332671   5112832         0         0         0         0
         0         0         0         0  20730879   4918720         0         0
         0         0         0         0         0         0   2348031   5113344
   2506751   5146368         0         0  20114431    786612  20677727   4915392
         0         0         0         0         0         0         0         0
  31643822    524288         0         0         0         0         0         0
         0         0   2369535   5145088         0         0         0         0
         0         0         0         0  19991647   4950912         0         0
         0         0  32052452    821499   3650655   4950592         0         0
         0         0  19992575   4918144   5690367   5081088         0         0
  19378271   4951104   3651583   4917824         0         0         0         0
  19384415   4951040         0         0   2411615   9097216  19379199   4918336
   2562047   5112320         0         0         0         0  19385343   4918272
         0         0   2412543   9064448         0         0         0         0
  19550207   5177472         0         0         0         0         0         0
         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0
  21562911   5081344         0         0         0         0         0         0
         0         0  32000302  14125311         0         0         0         0
         0         0         0         0         0         0         0         0
   3554303    786600         0         0  15188991    786592         0         0
         0         0         0         0         0         0         0         0
   1680383   9597952  32226884  12910592   2413567   9056256         0         0
         0         0  32122508   1572864         0         0         0         0
  32131107    821502  20102143    786616         0         0         0         0
         0         0  20090879    786609  31635615   4784128         0         0
         0         0         0         0         0         0         0         0
         0         0  20417535   9469440         0         0   9899103   4950656
         0         0   1679487   9598464         0         0   2822143   5145600
         0         0         0         0   9900031   4917888   2199551   5146112
         0         0  19057844    786437  31549061  12976128         0         0
         0         0         0         0   1183743   4918080         0         0
   3543039    786593         0         0         0         0         0         0
         0         0   4512767   9597440         0         0         0         0
         0         0  20460543   5177496         0         0         0         0
         0         0         0         0         0         0         0         0
         0         0         0         0         0         0  24159903    786433
         0         0   2412639   9089024         0         0   5706751   4784128
         0         0         0         0         0         0         0         0
  25676799  10123264         0         0         0         0  19551231    786438
         0         0         0         0         0         0         0         0
         0         0   8434335    786432         0         0   1182815   4950848
         0         0         0         0         0         0   2279423   5112576
         0         0         0         0  10929151   4915264  31571525  10062076
         0         0         0         0   3557471   4950528         0         0
         0         0         0         0         0         0         0         0
         0         0   3558399   4917760   1683551   4951232   3248127    786607
         0         0         0         0   1689695   4951168         0         0
         0         0   1684479   4918464         0         0         0         0
         0         0   1690623   4918400   3588191   9084928         0         0
         0         0  11127807   9635840         0         0   2642943    786435
         0         0   3589119   9052160         0         0         0         0
         0         0         0         0   2496511   5113600         0         0
         0         0         0         0         0         0  14145631   9080832
  20110335    786610         0         0         0         0         0         0
   2511871   5113088         0         0  14146559   9048064
```