AAEC/E243

AAEC/E243

AUSTRALIAN ATOMIC ENERGY COMMISSION

RESEARCH ESTABLISHMENT
LUCAS HEIGHTS

PDPIIASM AND PDPIISIM - A PDPII ASSEMBLER AND A PDPII SIMULATOR TO RUN ON AN IBM360 COMPUTER

by

G.W. COX

November 1972

ISBN 0 642 99533 8

AUSTRALIAN ATOMIC ENERGY COMMISSION

RESEARCH ESTABLISHMENT

LUCAS HEIGHTS

PDP11ASM AND PDP11SIM

A PDP11 ASSEMBLER AND A PDP11 SIMULATOR

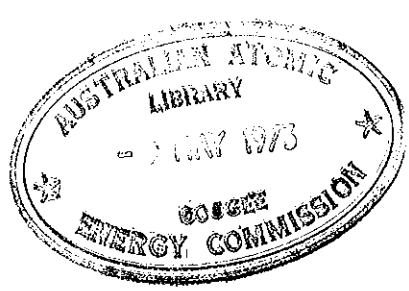
TO RUN ON AN IBM360 COMPUTER

by

G. W. COX

ABSTRACT

An assembler and a simulator for the PDP11 computer, written to run on an IBM360 computer are described. They provide a convenient and efficient way to prepare and test programs for the PDP11. Programs may be tested which use all of the PDP11 facilities, including input-output instructions and internal and external traps.



National Library of Australia card number and ISBN 0 642 99533 8

The following descriptors have been selected from the INIS Thesaurus to describe the subject content of this report for information retrieval purposes. For further details please refer to IAEA–INIS–12 (INIS: Manual for Indexing) and IAEA–INIS–13 (INIS: Thesaurus) published in Vienna by the International Atomic Energy Agency.

COMPUTER CODES; IBM COMPUTERS; PDP COMPUTERS; SIMULATION; TESTING

CONTENTS

	Page
1. INTRODUCTION	1
2. THE ASSEMBLER	1
2.1 General Description	1
2.2 Source Statement Syntax	1
2.2.1 The label field	2
2.2.2 The operator field	2
2.2.3 The operand field	2
2.2.4 The comment field	2
2.3 Terms	2
2.4 Expressions	3
2.5 Mode-Register Operands	3
2.6 Machine Instruction Mnemonics	4
2.7 Assembler Directive Statements	5
2.7.1 The .WORD statement	5
2.7.2 The .BYTE statement	6
2.7.3 The .ASCII statement	6
2.7.4 The .EVEN statement	6
2.7.5 The DS statement	6
2.7.6 The EQU or '=' statement	7
2.7.7 The ORG statement	7
2.7.8 The PRINT statement	7
2.7.9 The SPACE statement	8
2.7.10 The EJECT statement	8
2.7.11 The TITLE statement	8
2.7.12 The END or .END statement	8
2.8 The PARM Field	9
2.9 Core Allocation During Assembly	10
2.10 Assembler Output	10
2.10.1 Assembly listing	10
2.10.2 Object deck output	11
2.11 Job Control Cards Required	12
3. THE SIMULATOR	13
3.1 General Description	13
3.2 The PDP11-Time Clock	13
3.3 Simulation of the Unibus	14
3.4 Simulation of the Central Processing Unit	14
3.5 Simulation of CPU Instructions	15
3.5.1 Single and double operand instructions	15
3.5.2 JSR and JMP instructions	15
3.5.3 Instruction traps	15
3.5.4 The RESET instruction	16
3.5.5 The WAIT instruction	16
3.5.6 The HALT instruction	16

(continued)

CONTENTS (continued)

	Page
3.6 The Interface to Simulated I/O Devices	16
3.7 The Concept of Pseudo-Devices	17
3.8 The Pseudo-Core	17
3.9 How to Use the Simulator	18
3.9.1 Parameters required and the calling program	18
3.9.2 Specifying the device configuration	19
3.9.3 Output from the simulator	20
3.9.4 Simulated devices and pseudo-devices currently available	22
3.10 Testing of the Simulator	31
3.11 Simulation Speed	31
4. CONCLUSIONS	32
5. ACKNOWLEDGEMENTS	32
6. REFERENCES	32

APPENDIX I Assembler Source Statement Syntax Options

APPENDIX II Assembler-Recognised Machine Instruction Mnemonics and their Object Code

APPENDIX III Error Messages Produced by the Assembler

APPENDIX IV Logical Structure of the Assembler Program

APPENDIX V PDP11 Instruction Times Used by the Simulator

APPENDIX VI Sample Simulated Device Program

APPENDIX VII Sample Assembly and Simulation

APPENDIX VIII MAINDEC CPU Diagnostic Programs

1. INTRODUCTION

When using a mini-computer with a minimum or near-minimum input/output configuration, it is usually tedious to assemble and debug programs on the computer itself because of the necessity to use a multipass assembler with low-speed teleprinter input/output. There is thus a need for an assembler for the mini-computer which preferably can accept source program on cards for ease of modification, and which can use the speed and facilities of a large computer. For an on-line application, mini-computer time usually cannot be spared for assemblies; it is also desirable to carry out testing and debugging of a new program remotely.

This report describes a PDP11 assembler and a PDP11 simulator which fill the needs for both remote assembly and remote testing of PDP11 programs, and which are implemented on an IBM360 computer running under the OS/360 operating system.

2. THE ASSEMBLER

2.1 General Description

The PDP11 assembler program, PDP11ASM, accepts PDP11 source program statements and assembles PDP11 object code from them. The source and object code are optionally printed; the object code is optionally stored in pseudo-core for later simulation and optionally punched as object decks.

The assembler input is in 80-column card-image form. Most of the statement syntax accepted by the DEC PAL-11A assembler (D. Barlow, DEC private communication 1969) is accepted by PDP11ASM; the only difference is that PDP11ASM is a card input assembler, whereas PAL-11A is for papertape input.

A number of extensions to the PAL-11A syntax are allowed. They are designed to make the language easier to use and read in its card-oriented form. These differences are tabulated in Appendix I. The details of the syntax and format of the assembler source statements are described in Section 2. The logical structure of the assembler program is described in Appendix IV.

2.2 Source Statement Syntax

Assembler source statements are written one per 80-column card-image. A statement may be one of three types;

(i) comment statement,

(ii) machine instruction mnemonic,

(iii) assembler directive.

Comment statements are defined by the fact that the first character is ';' or '/'. A comment statement produces no object code; its sole function is to allow comments to be written so as to appear on the program listing.

Machine instruction mnemonics and assembler directives have the same basic format, as follows:

label, operator, operand, comment.

Meaningful information is taken only from columns 1 to 72 on the card. Columns 73–80 are reproduced without inspection.

2.2.1 The label field

The label field consists of any number of symbols (including zero) separated from each other and from the operator field by delimiters. The label field serves to define the symbols it contains for the duration of the current assembly. A valid symbol consists of one to eight characters, each of which may be alphabetic, numeric, '$' or '.', except that the first character may not be numeric.

Valid delimiter characters are blank, colon (':'), and equals ('='). If a blank delimiter is used, the symbol must start in column 1 of the card. If ' ' is used, the symbol must start in column 1 and the '-' is taken as operator as well as delimiter. If ':' is used (as in PAL-11A), the label field need not start in column 1 but must be the first non-blank item on the card. Furthermore, additional symbols after the first may be placed in the same label field and must all be delimited by colons.

The label field is optional in all machine instruction mnemonic statements and some assembler directive statements. It is illegal for some assembler directives and mandatory for others.

2.2.2 The operator field

The operator field is optional – i.e. a label field on its own is a valid assembler statement. The operator field consists of a recognised machine instruction or assembler directive mnemonic with a blank or comment-character (; or /) delimiter. If a second field exists and cannot be recognised as an operator, it is taken to be the operand field and the operator is assumed to be the assembler directive '.WORD'.

2.2.3 The operand field

The existence of the operand field depends on the nature of the operator. The operand field usually terminates with the first blank or comment-character after its start, excluding characters which are first after single quote (') or first two after double quote (") (see Section 2.3). Some assembler directives are exceptions to this rule. The operand field must contain the number of operands required by the operator, operands being separated from each other by a comma.

2.2.4 The comment field

The comment field starts after the operand if the operator requires one, or after the operator if it does not. A comment may also be forced to start after any previous field by using a comment-character (/ or ;) as its first character. For instance, a label field followed by a comment starting with ';' is a valid statement, serving to define any symbols present in the label field.

2.3 Terms

Terms are used to construct expressions, and may be of three types – optional, register and address.

An optional term is one of the following;

(i) An octal number, written as a succession of octal digits.

(ii) A decimal number, written as a succession of decimal digits, followed by a decimal point.

(iii) A single-character 8-bit ASCII-code value, written as a single quote (') followed by the character whose ASCII-code is required.

(iv) A two-character 16-bit ASCII-code value, written as a double quote (") followed by two ASCII characters. The first character ASCII value is put in the left or high-order 8 of the 16 bits, and the second character is put in the right or low-order 8 bits.

A register term is one of the following;

(i) A symbol which has been defined in a previous assembler statement as a register symbol, using the EQU or = assembler directive.

(ii) The character '%' followed immediately by an optional term.

An address term is one of the following;

(i) The special symbol '.' which represents the location counter at the start of the current assembler statement.

(ii) A symbol which has not been previously defined as a register symbol. This definition includes symbols defined as address anywhere in the assembly, those defined as register after the current statement, and those which are undefined.

(iii) A machine instruction mnemonic which is undefined as a symbol. The value of this term is then the object code represented by the instruction mnemonic, with any variable bits set to zero. If the mnemonic is defined as a symbol anywhere in the assembly, the defined value is used.

2.4 Expressions

In the syntax of the operands of machine instruction and assembler directive mnemonics expressions may be combined in various ways. An expression consists of one or more valid terms combined by valid operators. Valid operators are '+' (add), '–' (subtract), '*' (multiply), '&' (and), and '|' or '!' (or). The terms are combined working strictly left to right and no parentheses are allowed. For instance, 1+2*2 evaluates as 6.

An expression may be one of two types, register (REXPRN) or address (AEXPRN). A register expression is composed of register and/or optional terms, and its value must be between 0 and 7. An address expression is composed of address and/or optional terms, and its value must be representable in 16 bits.

In some contexts an expression may be of either type. In this case, the first non-optional term appearing in the expression fixes the type of expression. If all terms in the expression are optional, the expression is assumed to be an address type.

2.5 Mode-Register Operands

The single operand and double operand classes of machine instruction mnemonic require one and two mode-register operands respectively. A mode-register operand (denoted OPRND) is a combination of expressions and characters which the assembler converts to a 3-bit mode specification, a 3-bit register specification, and possibly a 16-bit word as well. The modes of addressing in the PDP11 are described in detail in the PDP11 Handbook (Digital Equipment Corporation 1970). The syntax by which these modes are assembled is described here.

Indirect addressing (represented by a 1 in the low order bit of the mode) is assembled when

(i) the first character of the operand is '@' or

(ii) the first and last characters of the operand are '(' and ')' respectively.

After stripping the @ or ( ) from the operand, it must then fall into one of the following four types for explicit statement of addressing mode:

REXPRN	Mode 0
(REXPRN)+	Mode 2
-(REXPRN)	Mode 4
AEXPRN(REXPRN)	Mode 6

The address expression in the mode 6 representation need not be present; if not present, a value of zero is assumed. Immediate and relative addressing are implicit addressing modes which make use of the program counter (register 7). These modes are assembled when the operand takes the forms

# AEXPRN - AEXPRN	immediate mode	(7)+
AEXPRN	relative mode	X(7)

It will be noted that before evaluating an expression to determine its type mode 0 and relative mode addressing cannot be distinguished. This is where the concept of an expression of either type is used, the first non-optional term encountered giving its type to the whole expression. The practical significance of this is that if register mode addressing is desired, this can only be detected by the assembler if register symbols used are predefined, otherwise relative addressing is assumed by the assembler.

2.6 Machine Instruction Mnemonics

When assembling a machine instruction mnemonic, the assembler first checks that the location counter specifies an even address. If not, a zero byte is assembled first, as if a .EVEN assembler directive had preceded the current statement.

All the machine instruction mnemonics which the assembler recognises, together with the object code generated for them are listed in Appendix II by operator type. The format of the operands required for the various classes of instruction mnemonic are given below. The quantities REXPRN, AEXPRN, OPRND are used as defined in the preceding sections.

Instruction Class	Format
Double Operand	OPR OPRND,OPRND
Single Operand	OPR OPRND
JSR	JSR REXPRN,OPRND
RTS	RTS REXPRN
Conditional Branch	OPR AEXPRN
Condition Code Operator	OPR
HALT,WAIT,REJECT,IOT	OPR
EMT,TRAP	OPR AEXPRN

For conditional branch instructions, the address expression must evaluate to a value which is representable as a signed 9-bit displacement from the current location counter + 2. An error message is issued for a larger displacement, and a displacement of zero from the current location is assembled, i.e. 'BR .' .

SCC (set all codes) and CCC (clear all codes) are the only combined condition code operators recognised by the assembler. If others are desired, they can be defined as symbols and used as the operand of an implied .WORD instruction – e.g.

```
CLCV = CLC!CLV
```

The address expression in the operand of a TRAP or EMT instruction is truncated on the left to a value no larger than 8 bits.

Word instructions having mode-register operands require their operands to represent even addresses, otherwise a bus error will occur when the program is run. In the case of relative and deferred immediate addressing modes, the assembler checks word instruction operands and an error message is issued if they are odd. Note that this check is not made for indexed mode as the assembler cannot know whether the index register will be even or odd at the time the program is run.

2.7 Assembler Directive Statements

All of the PAL-11A assembler directives are recognised except .EOT (end-of-tape), as this is inappropriate to a card input assembler. A few additional assembler directives are handled. The complete list of those recognised is

.BYTE	.END	ORG	TITLE
.WORD	END	DS	EJECT
.ASCII	=		SPACE
.EVEN	EQU		PRINT

2.7.1 The .WORD statement

The format of a .WORD statement is

```
[label]  .WORD  AEXPRN,AEXPRN,...etc.
```

This statement causes full words to be assembled on word boundaries. The number of words assembled is equal to the number of commas between address expressions plus one. Address expressions may be missing (i.e. commas may be adjacent) in which case a zero word is assembled to represent the missing expression.

Examples

```
.WORD   0,0,0,0,        (4 zeros)
.WORD   ,,,             (4 zeros)
.WORD   "BA,"DC
.WORD   .,.+2,.+4
.WORD   XYZ
```

2.7.2 The .BYTE statement

The format of this statement is

[label] .BYTE AEXPRN,AEXPRN,...

This is similar to the .WORD instruction, but bytes instead of words are assembled according to the value of the expressions. If an expression value is more than 8 bits, it is truncated on the left without comment by the assembler. The bytes are assembled into consecutive locations; the first byte is not aligned on a word boundary.

Examples

```
.BYTE   'A,'B,'C,'D,212,215

.BYTE   CR,LF

.BYTE   X–.
```

2.7.3 The .ASCII statement

The format of this statement is

[label] .ASCII delimiter, text, delimiter

The first non-blank character after the word .ASCII is used as a delimiter. The text operand is then defined as all succeeding characters up to but not including the next character which is the same as the first delimiter. Each character of the text operand is translated into its ASCII code and assembled into successive locations. The first character is not aligned on a word boundary. The 8-bit ASCII code used is the standard 7-bit ASCII code in the low-order 7 bits, with a 1 in the high-order bit.

Examples

```
.ASCII  /TEXT/

.ASCII  QTEXTQ

.BYTE   'T,'E,'X,'T
```

These three statements assemble identically.

2.7.4 The .EVEN statement

This statement has no operands. Its format is

[label] .EVEN.

It establishes the current location counter at a word boundary by assembling a zero byte if necessary. It is not necessary to use the .EVEN statement in PDP11ASM as the assembler ensures that all instructions and .WORD statements are assembled at word boundaries – in effect it generates its own .EVEN statement where necessary.

2.7.5 The DS statement

The format of the define storage statement is

[label] DS AEXPRN.

The expression is evaluated and a number of words equal to the value of the expression is reserved. The words start on a word boundary. No object code is assembled into these words; however, the trace bit in pseudo-core is set according to the state of the PRINT TRACE option (q.v.).

All address terms in the address expression must be pre-defined, i.e. their definition must occur physically before the DS instruction.

Examples

```
X      DS   0          (define label)

BUF    DS   BUFSIZE    (BUFSIZE is pre-defined)
```

2.7.6 The EQU or '=' statement

The operators EQU and = have identical functions. The format of this statement is

```
label   EQU   EXPRN
or
label   =     EXPRN
```

If the PAL-11A compatible '=' is used, blanks are optional between label and operator and between operator and operand. A label field containing one symbol only is required. The expression terms must all be pre-defined or optional type, and the expression may be of either address or register type. The function of the EQU statement is to give the type and value of the expression to the symbol in the label field. It is the only way to define a symbol as register type. If the label field is '.', the statement becomes identical with the ORG statement (q.v.).

2.7.7 The ORG statement

The format of this statement is

```
      ORG   AEXPRN
or
   .  =     AEXPRN
```

All address terms in the address expression must be pre-defined, and when the ORG form is used, the label field must not be present. The function of this statement is to reset the location counter to the value specified by the address expression.

For the first assembly of a job step, the location counter is initially zero; for subsequent assemblies it remains where the previous assembly left it.

2.7.8 The PRINT statement

The format of this statement is

```
PRINT   keywords
```

This statement is used to control the listing of the assembler and simulator in a number of ways. The operand consists of one or more keywords separated by commas and containing no blanks. If two keywords opposite in meaning are included, the last to appear is the one which takes effect. The keywords and their meanings are:

ON	Listing of source statements and object code is provided.
OFF	Listing of source statements and object code is not provided unless a diagnostic error is issued for the statement.
DATA	All object code produced by the statement assembly is printed, three words or bytes per line.
NODATA	Only the first line of object code (three words or bytes) is printed. The DATA/NODATA option is relevant only for .ASCII, .WORD and .BYTE statements.
EDIT	The source statement is edited before assembly and listing so that operator, operand and comment fields start in columns 10, 16 and 40 respectively of the card, provided each field is not too long.
NOEDIT	The source statements are analysed and listed without editing.
TRACE	If pseudo-core is being set up by the assembler, all following assembled core locations until the TRACE option is reset will have this trace bit in pseudo-core set to 1 – i.e. simulation of these instructions will result in a trace print-out. Note that this applies to .WORD, .BYTE, .ASCII and DS assembled code, as well as to machine instructions. It does not apply when the location counter is moved by an ORG or a . statement.
NOTRACE	The pseudo-core trace bit is set to 0 for the following assembled core locations.

The defaults for the PRINT options are ON,DATA,NOEDIT,NOTRACE.

2.7.9 The SPACE statement

The operand of the SPACE statement is a single digit between 0 and 7. The operand need not be present, in which case 1 is assumed. The SPACE statement is used to control spacing during assembly listing. The number of blank lines specified by the operand is left blank in the listing. A label field must not be present. The SPACE statement itself is not listed unless in error.

2.7.10 The EJECT statement

This statement needs no operand field and must have no label field. It causes a skip to new page in the assembly listing. The EJECT statement itself is not listed.

2.7.11 The TITLE statement

This statement must not have a label field. The operand field starts at the first non-blank character after the operator, and continues to and includes column 72. There is no comment field. The TITLE statement causes a skip to new page in the assembly listing; the operand is used as a heading on this and all subsequent pages until the appearance of a new TITLE statement. A blank operand field causes a skip to new page and removes the heading defined by a previous TITLE statement.

2.7.12 The END or .END statement

The END and .END operators have the same meaning. The format of this statement is

END [AEXPRN]

The label field is not allowed. The address expression operand field is optional and, if present, causes the object deck end card to contain a transfer to the location indicated by the value of the expression, to automatically start the program after loading. This transfer address is also set up in the parameter field passed to the simulator.

If the operand field of the END statement is omitted, a transfer address of 1 is punched on the object deck end card. An odd address causes the loader to halt after loading the program. No starting address is stored for simulation if the operand field is omitted (a previously specified starting address would remain operative).

The END statement causes the assembler to print out the symbol table under control of the SYMTAB option (see Section 2.8), and also to print out a summary of the statement numbers of statements in error. The symbol table is then cleared and a new assembly may be started. The location counter remains where the previous assembly left it and the PRINT options are set back to their default values.

2.8 The PARM Field

A number of run-time options are available with the assembler. These are specified on the 'parm field' in standard OS/360 format (IBM Job Control Language Reference, IBM-SRL Form No. GC28-6704-0 1970). The parm field contains a number of keywords separated by commas and containing no embedded blanks. Valid keywords and their meanings are

DECK	An object deck of the assembled code will be produced as 80-byte card images on the SYSPUNCH data set. (See Section 2.10.2 for the format of the object deck).
NODECK	No object deck is produced.
XREF	An alphabetical listing of all symbols used or defined is produced at the end of each assembly.
XREF=n	The same as XREF. Additionally, the number of cross-references to be allocated storage is at least n where n is a decimal number.
NOXREF	The cross-reference listing of symbols will not be produced. Nevertheless, a small number of storage locations for the cross-reference of statements in error is reserved.
SYMTAB=n	The number of symbols to be allocated storage for the current assembly is at least n, where n is a decimal number. The SYMTAB=n and XREF=n options are rarely required – see Section 2.9 core allocation.
CORE=nK	This specifies the size of PDP11 core in units of 1K = 1024 16-bit words for which this assembly is destined. This core size is used for two purposes by the assembler – if assembly is specified outside this core size an error diagnostic is issued and if simulation is required a pseudo-core is set up which has this size (see Section 3).
LOAD=SIM	This parameter indicates that after completion of assembly, control is to be transferred to the program PDP11SIM, which will simulate operation of the PDP11 program just assembled. The pseudo-core address and size, and the PDP11 program starting address are transmitted to the simulator. No parameters after the LOAD=SIM parameter are recognised by the assembler. All following information is passed to the simulator as a simulator parm field.

The assembler parm field may be empty, in which case the default parameters used are

NODECK,CORE=4K,XREF=0,SYMTAB=0

2.9 Core Allocation During Assembly

Before the start of assembly, IBM360 core for the duration of the assembly is allocated within the user's region according to the options specified in the parm field.

If LOAD=SIM is specified, pseudo-core for the simulator is first obtained within the user's region. The amount of pseudo-core allocated is 4 bytes per PDP11 byte requested, e.g. 32K bytes of IBM360 core are allocated for CORE=4K (words). If this core cannot be obtained in the user's region an error message is issued and assembly does not proceed.

The largest block of core remaining in the user's region is then obtained. This is split between symbol and cross-reference tables in various ways, depending on the table sizes requested. The symbol table size requested consists of a fixed amount of 256 bytes for a hash table plus an amount per entry for all entries requested on the SYMTAB=n parameter. This amount is 20 bytes for NOXREF and 24 bytes for XREF. To the total is added 4 bytes per entry for all entries requested in the XREF=n parameter. If this total is more than the amount of core obtained in the user's region as above, then an error message is issued and assembly does not proceed. If the storage obtained exceeds this total (as is usual), the excess is divided between symbol and cross-reference entries in the ratio 1 : 3 if cross-reference listing is required, and 15 : 1 if not. The small number of XREF entries then allocated serves only to cross-reference statements in error. It is thus not normally necessary to specify the XREF= or SYMTAB= parameters. If a 1 : 3 ratio of symbols to cross-reference is adequate, the size of the tables can be controlled by the region allocated to the job step.

2.10 Assembler Output

2.10.1 Assembly listing

An assembly listing is produced on the SYSPRINT data set. This commences with a page summarising the core allocation for pseudo-core, and symbol and cross-reference table entries. Source statement listings and symbol table/cross-reference listings are then produced for the number of assemblies in the current job step.

The source statement listings provide information under a number of headings as follows:

LOC — The value in octal of the location counter for the current statement is listed. However, if the operator is EQU, the LOC field contains the value assigned to the symbol in the label field.

OBJECT CODE — The object code, if any, assembled for the current statement is listed. Space for three words is provided – machine instructions produce up to three words of code; assembler directives may produce more than three words or bytes which are printed on successive lines if the PRINT DATA option is in effect. Code is normally printed one word at a time in octal; code assembled by .BYTE and .ASCII assembler directives is printed one byte at a time in octal.

STMT — A sequential statement number is printed. This number is used for cross-referencing.

SOURCE STATEMENT — The 80-byte source statement card image is printed as read by the assembler.

Errors detected by the assembler are noted in the listing immediately after the statement in error. Up to 43 errors per source statement can be separately noted. A line is first printed containing characters which mark the positions of the errors in the source statement. Characters 1–9 and A–Z are used. For each error separately detected, a further line is then printed associating each error position with a message text describing the error. All error messages produced by the assembler are listed in Appendix III. Each error noted has an associated severity code of 4,8,12 or 16. The maximum severity code noted during assemblies is returned as a condition code to the operating system (standard OS/360 convention). Severity codes have meanings as follows:

4	Minor error. Code assembled will probably run as intended by the PDP11 programmer.
8	Code could not be assembled as specified on the source statements. Program will probably run in some fashion.
12	Code assembled will almost certainly not be able to run without a bus error.
16	Catastrophic error – assembly could not be completed.

After the END statement of each source listing a cross-reference listing of all symbols used is printed if the XREF option has been specified. The defined value of the symbol, the statement where it is defined, and a list of the statements where it is referenced is printed. The letter 'R' appears next to the value to denote a register symbol.

If any errors occurred, a cross-reference list of the statement numbers of the statements in error is printed. This appears regardless of the XREF option. If no errors occurred, the message 'NO STATEMENTS FLAGGED IN THIS ASSEMBLY' is printed.

2.10.2 Object deck output

An object deck is produced by the assembler on the SYSPUNCH data set if the DECK option on the parm field is selected. This deck in EBCDIC code is in an AAEC standard format for mini-computers, (G.W. Cox, unpublished). Two types of card may be produced, a text card and an end card. The text card format is

Column	Contents
1	06 (hex) indicates mini-computer object deck.
2–4	TXT (characters).
5	73 (hex) AAEC code for PDP11 computer.
6–11	PDP11 (characters).
12	01 (hex) number of card columns per addressable storage unit.
13	byte count – number of bytes of code on this object card.
14–16	PDP11 address where the first byte of text on this card is to be stored.
17–72	up to 56 consecutive bytes of object code.
73–76	blank.
77–80	a decimal card sequence number.

The end card format is

Column	Contents
1	06 (hex).
2–4	END (characters).
5	73 (hex).
6–11	PDP11 (characters).
12	01 (hex).
13	00 (hex).
14–16	PDP11 address to which control is transferred after loading, or 000001 (hex) if loader is to halt.
17–76	blank.
77–80	a decimal card sequence number.

2.11 Job Control Cards Required

A procedure is available in the system procedure library at AAEC which contains most of the job control language (JCL) statements required to run the assembler. To use this procedure, the following JCL statements should be used (quantities in square brackets are optional):

```
//[name] EXEC PDP11ASM [,PARM.ASM='options'] [,REGION.ASM=nnnK]
//ASM.SYSIN DD *
     Source statement decks
/*
```

If it is required to run the assembler without using the procedure, DD cards with the following ddnames and characteristics must be provided. If any are found to be missing or in error, the assembler terminates with a diagnostic message and a condition code of 16.

SYSPRINT Defines the sequential data set on which the printed output will be put. LRECL is fixed at 121 and may not be set on the DD card. RECFM may be user-specified as FA or FBA. BLKSIZE may be user-specified as any multiple of 121. If RECFM and BLKSIZE are not specified, they default to FBA and 121 respectively.

SYSPUNCH Defines the data set on which the object deck output is written. It is required only if the DECK option is specified. LRECL is fixed at 80. RECFM may be set to F or FB and BLKSIZE may be set to a multiple of 80. Default RECFM and BLKSIZE values are FB and 800 respectively.

SYSUT1 Defines the sequential work data set used for intermediate storage of partially assembled source statements. RECFM and LRECL are fixed at FB and 88 respectively. If BLKSIZE is not specified as a multiple of 88, it is set to 3520. This data set may reside on any non-unit record device.

SYSIN — Defines the input sequential data set containing the user's assembler language source statements. LRECL is fixed at 80, BLKSIZE may be specified as a multiple of 80 and RECFM may be specified as F or FB. These specifications may be either on the DD card or the data set label. If //SYSIN DD * is used, the operating system selects suitable values for these quantities.

The region required to run the assembler depends on the symbol table size. A region of 100 K gives a symbol table size of about 1500 and a cross-reference table size of 4500 if LOAD=SIM is not specified.

3. THE SIMULATOR

3.1 General Description

The simulator program, PDP11SIM, is written in IBM360 assembly language; it simulates and optionally displays the execution of a given PDP11 program. The program is contained in a pseudo-core area, which is a region of IBM360 core containing the core contents of the PDP11 to be simulated. The simulator requires the loaded pseudo-core to be passed to it by a calling program, and also requires a specification of the input-output device configuration required for the simulation run. An important feature is that it attains the closest possible simulation of the internal hardware operation of the PDP11 with software, to the extent that major hardware components such as the core, the CPU, the PDP11 unibus, the unibus-to-device interface, and the devices themselves all have discrete software analogues. This means that it is possible to simulate input-output operations very closely including real-time behaviour and I/O interrupts. The only limitation on I/O simulation is the ability (or incentive) to write a program representing the I/O device which responds to 'signals' on the simulated unibus and may send signals to it (i.e. status, data, or interrupt requests) in the same manner as the real device would behave.

In the Sections that follow, all references to PDP11 components or operations are to be read as the simulation of that component or operation by the relevant part of the simulator program, unless otherwise stated.

3.2 The PDP11-Time Clock

Control of the simulation is largely determined by the value of a PDP11-time clock which is maintained in units of a tenth of a microsecond. This clock is updated normally only by operations taking place in the CPU, e.g. memory access, instruction execution, and trap sequences, all of which take a known time to execute.

The CPU and the devices compete for control of the simulating computer (in this case the IBM360). This is not strictly analogous to the real-world situation where a real device or device interface has control circuits of its own and may act on its own initiative over a period of time. However, a technique such as this is necessary as the simulating computer has only a single path of control. It is the sole function of a small program, hereafter called 'the simulator', to decide on the basis of the PDP11-time clock value whether the CPU or a device should have control of the simulating computer. The simulator so defined may then be thought of as a PDP11 system simulator and is not to be confused with the PDP11 CPU program which, like each device program, runs under 'the simulator'. Requests for control are made by the CPU and the devices and are granted by the simulator in the following ways.

The CPU is usually in the state where it is requesting control of the simulator. The only exception to this is during the PDP11-time between a PDP11 WAIT instruction and an I/O interrupt. When the CPU gets control, it does one unit of work before returning to the simulator. A unit of work may be one of two things – the fetching and execution of a PDP11 instruction, or the granting of an external interrupt, the decision between these being made by the CPU program. Both of these activities will update the PDP11-time clock and either may cause one or more processor traps which the CPU handles before relinquishing control. The simulator then tests again to see whether the CPU or a device should have control next.

Each device has an associated PDP11-time at which it desires control of the simulator. A zero value implies that no request for control is current. When the PDP11-time clock reaches or exceeds the requested time the simulator passes control to the device program. It is stressed that this passing of control is based solely on the PDP11-time clock and has nothing whatever to do with processor or device priority. When a device program gets control of the simulator in this way, it has available to it most aspects of the simulation environment and may affect these in any way it pleases before returning. For example, it may request an interrupt, it may clear an interrupt request, it may make a further request for control of the simulator again, it may alter its own control register values or it may perform some I/O on the simulating computer to simulate PDP11 I/O. In the case of a teleprinter, for example, the request for control might be made for a given time (representing the teleprinter speed) after a character has been moved to the teleprinter buffer. At the first instruction boundary, at or after this time, the teleprinter program will get control and it will set its done flag and may request an interrupt if its interrupt enable bit is set. It is important to note that it is not the function of the teleprinter in this example to simulate an interrupt when it gets control – it may merely request one. It is the function of the CPU when it gets control to grant interrupt requests if appropriate.

It will be apparent that a device frequently will not get control at precisely the PDP11-time requested, but will have to wait for an instruction boundary. This again is unlike the real-world situation, but the difference is unimportant because any interrupt requests or other changes in the simulation environment which the device program may cause cannot have any effect until an instruction boundary.

3.3 Simulation of the Unibus

The commands issued by the PDP11 CPU to the unibus are closely simulated. These are

DATI	Data in to the CPU
DATIP	Data in and pause
DATO	Data out to memory
DATOB	Data out, byte.

The DATI and DATIP commands are simulated identically – both are effectively a DATIP, i.e. the address on the unibus is retained for possible use by a following DATO or DATOB. DATO and DATOB are thus always preceded by a DATIP which has saved the unibus address. On DATI or DATIP, the PDP11-time clock is updated by the memory access time (1.2 microsecond); DATO and DATOB do not update the clock.

When a unibus address reference is made, a check is first made of the core size to see if the address is in core. If so, the read or write operation to pseudo-core is performed. If the address is not in core, it is checked to see if it is the processor status word address (177776 octal). If so it is read or written accordingly. If an address match has still not been obtained, all devices on the unibus are checked to see if they contain registers with the given address; if a match is then found, the unibus operation to that register is simulated. Additionally, after such a unibus operation to a device, the device program gets control of the simulating computer, and may affect itself or the simulation environment in a similar way to when it gets control directly from the simulator. If the unibus address is still not matched after checking core, processor status and devices, a bus time-out is simulated; 10 microseconds are added to the PDP11-time clock, and the bus error trap is set so that a processor trap to location 4 will occur at the next instruction boundary. If the operation detecting the bus error is a DATI for the purpose of instruction 'fetch', then the bus error trap is simulated immediately.

3.4 Simulation of the Central Processing Unit

When the simulator has determined that the CPU shall have control, the CPU program checks

the current processor priority against that of the highest priority interrupt request current. If the processor priority is greater or equal, it proceeds to fetch an instruction from memory and simulate its execution. If the interrupt request priority is greater, a trap sequence is simulated by pushing the current processor status (PS) and program counter (PC) on the stack, and loading a new PC and PS from the interrupt vector appropriate to the requesting device. At the same time, the request from that device is cleared and the PDP11-time clock is updated by the time for an interrupt sequence (9.3 microseconds).

If at any time a processor trap condition arises, this condition is processed before the CPU relinquishes control to the simulator to check device requests. Processor trap conditions in order of processing are

(i) Bus error trap (time-out or odd address for word instruction).

(ii) Illegal instruction trap.

(iii) Trace trap.

(iv) Stack overflow trap.

(v) Power fail trap.

Each of these trap conditions except the power fail trap may be set by the simulation of the operation of the CPU or the unibus. The power fail trap may be set by the operation of a special pseudo-device (see Section 3.9.4.5).

3.5 Simulation of CPU Instructions

Simulation of most of the PDP11 instructions is straight-forward. As each is executed, the PDP11-time clock is updated by the instruction time as given in the PDP11 Handbook (see Appendix V). A description is given here of those instructions whose simulation merits special comment.

3.5.1 Single and double operand instructions

The source (if any) and destination operands are looked up by as many references to the unibus (i.e. core or device registers) as is necessary for the operand mode being used. If no bus error condition then exists, the specified instruction is simulated, the condition code bits in the PS are set according to the result, and a DATO or DATOB to the unibus is issued if necessary. If a bus error condition occurs as a result of looking up either operand, the instruction execution and the setting of the condition code are suppressed.

3.5.2 JSR and JMP instructions

The destination operand is decoded in the same way as the single and double operand instructions. If the mode is zero, the illegal instruction trap is set and the JMP or JSR is suppressed. The times used to update the PDP11-time clock are one memory cycle less than the real times, to allow for the fact that the CPU simulator's operand decoder program fetches the data (i.e. instruction) at the new address, while the actual PDP11 does not in the case of JMP and JSR.

3.5.3 Instruction traps

When any of the reserved instructions or the instructions IOT, Breakpoint Trap, TRAP or EMT is encountered, a processor trap sequence is entered using the appropriate trap vector and the PDP11-time clock is updated. The granting of I/O interrupt requests is inhibited until one instruction after the processor trap. A trace trap is not taken after an instruction trap.

3.5.4 The RESET instruction

The function of the RESET instruction is to clear all I/O devices. This is simulated by passing control to each device in turn. The device program must perform any reset action appropriate to the device. The PDP11-time clock is updated by 20,000 microseconds.

3.5.5 The WAIT instruction

The highest priority interrupt currently pending is checked against the processor priority and the interrupt simulated if it is greater, as for other instructions. If no interrupt can be taken, the PDP11-time clock is advanced to the time of the next device's request for control, the device is serviced, and the pending interrupt priority again checked (the device may have requested an interrupt). If no interrupt can again be taken, this process is repeated until no further devices require service, or until 20 device services have been granted in the wait condition. An arbitrary limit such as 20 is necessary to prevent an endless loop for a device which keeps requesting control (such as a clock wanting to set its done bit at regular time intervals) but does not make an interrupt request. If the processor still remains in a wait condition, the simulation is terminated.

3.5.6 The HALT instruction

This instruction usually causes simulation to terminate immediately. However, if a power-fail trap service routine is being simulated when the HALT instruction is encountered, all registers are set to zero, the PDP11-time is reset to zero, a reset instruction is simulated, and a trap to location 24 takes place to simulate a power auto-restart. When using the power-fail pseudo-device, this enables both power-down and power-up PDP11 routines to be simulated.

3.6 The Interface to Simulated I/O Devices

The Section describes in some detail the way in which a device program is organised. For more complete details, see the sample device program listing in Appendix VI. The mnemonic symbols used here are identical with those used in the assembler language source coding of the device program.

The first part of the device program has a fixed format essential to the proper functioning of the simulator although the device may not be used. This fixed format block is in two parts, the device data block (DEVICE) and the status register data block (STATREG). The device data block contains the following quantities:

(i) DEVDEV — a pointer to the next device program on the bus. The simulator uses this field to find all the devices on the unibus for a given run.

(ii) DEVTIME — the PDP11-time for which a current request for control exists for this device.

(iii) DEVNOSR — the number of status or control registers belonging to this device.

(iv) DEVNAME — a device type name by which this device program is known. More than one device of the same type may be attached to the unibus for a given run.

(v) DEVDDNAM — a device ddname referring to the device. This name is user-given and must be unique for every device on the unibus; it is the ddname used for any OS/360 I/O associated with this device.

(vi) DEVBTIME — the entry point in the device program used when a time-request for control is satisfied.

A status register data block exists for each addressable register associated with the device. Each STATREG block contains:

(i) SRADR — the unibus address (16-bits) of the register concerned.

(ii) SRCONT — the current contents of the register.

(iii) SRMR — a mask indicating those bits of the register which are readable from the bus.

(iv) SRMW — a mask for those bits which may be written from the bus.

(v) SRPRTY — the priority of a currently pending interrupt associated with this register (zero if none pending).

(vi) SRINTV — The interrupt trap vector address in low core.

(vii) SRBDATI — an entry point where the device receives control after a unibus DATI.

(viii) SRBDATO — an entry point where the device receives control after a unibus DATO or DATOB.

(ix) SRBRESET — an entry point where the device receives control when a unibus reset pulse is simulated.

(x) SRBEND — an entry point where the device receives control when the simulator is about to terminate. This gives a device program a chance to flush I/O buffers, close DCB's and free core areas.

At any of the five types of control entries, a device program has directly available to it the current PDP11 time, the time of its previous request for control, the previous contents of its registers and the priority of its current interrupt request. It also has available, more indirectly, the whole of the simulation environment. It may change any of these quantities before returning to the simulator. Note that a device does not get (and does not need) control at the time an interrupt request from it is granted. However, the CPU clears such requests when granted.

At any of the five entries to a device program, two returns are available. Normal return continues the simulation, error return gives a message text and causes the simulation to terminate with the printing of the message.

The fixed format front-end is followed by the device program itself. It is entered via the five types of entry in the device and status register data blocks, and may use any OS/360 facilities. It should use the ddname, which is user-specified in the DEVDDNAM field, for any OS/360 I/O performed.

3.7 The Concept of Pseudo-Devices

The general software interface to a simulated device allows for the possibility of writing pseudo-device programs. Such a program simulates a device which has no real-world analogue, but which affects the simulation in some useful way. Any specific extra feature required of the simulator, can usually be written as a pseudo-device, without incorporating it in the simulator main program. Examples of pseudo-devices are described in Section 3.9.4.

3.8 The Pseudo-Core

The pseudo-core is a region of core in the simulating computer which contains the core-load of the computer to be simulated. This pseudo-core has the same format as that used for other mini-

computer simulators at AAEC (Backstrom 1970, Sanger 1970). This format uses one IBM360 word (32-bits) per addressable unit of the mini-computer. This seems wasteful in the case of the PDP11 where the addressable unit is 8 bits, but this is outweighed by the advantages of compatibility. Two bits of the IBM360 word are used for attributes. The format of one pseudo-core word is as follows:

bit 0 (left bit)	Assigned bit – set to 1 when this byte is unassigned, set to 0 when it is assigned a value by any program using the pseudo-core. This bit is never read by the simulator.
bit 1	Trace bit – set to 1 if the instruction at this address is to be traced or if the interrupt occurring through an interrupt vector at this address is to be traced. The tracing of an instruction or interrupt results in one line being printed describing the simulation of that operation (see Section 3.9.3.2). The trace bit is set by the TRACE function of PDPGENER (Backstrom 1970) or by the PRINT TRACE function of the assembler.
bits 2–23	Unused.
bits 24–31	Bits 7–0 of the PDP11 byte. Note that IBM360 bits are numbered left to right and PDP11 bits are numbered right to left.

The pseudo-core words are arranged in IBM360 core in increasing order of their PDP11 byte addresses.

3.9 How to Use the Simulator

3.9.1 Parameters required and the calling program

The simulator implemented on the IBM360 is always called by a higher level program. This program must provide the simulator with two parameter lists. The first is a three-word list whose address is contained in register 1. These three words must contain respectively the IBM360 address of the pre-loaded pseudo-core, the core size of the PDP11 to be simulated in PDP11 bytes, and the IBM360 address of the PDP11 program's entry point in pseudo-core. The second parameter list is a character field whose address is in register 2. This field consists of a set of keywords separated by commas, containing no embedded blanks, and terminated by a blank or an invalid keyword. Valid keywords and their effect are

LIMIT=nn	This provides a printed output limit. The simulator terminates when the number nn of printed lines is exceeded. If not specified, a limit of 2750 lines (50 pages) is assumed.
DUMP	This causes an octal core dump of the PDP11 pseudo-core to be printed after the end of simulation.
HISTORY=nn	This causes a core buffer of the nn last PDP11 instructions and interrupts simulated to be saved, regardless of the trace option. Upon termination, this buffer is printed out. 40 bytes of IBM360 core are used for each entry in the buffer.

At AAEC two programs are currently available which call the simulator in this way. One of these is the PDP11 assembler described in Section 2. It sets up and passes the first parameter list, and calls the simulator when LOAD=SIM is specified in its parm field. The second parameter list for the simulator is placed after the LOAD=SIM parameter in the assembler parm field. The

assembler loads pseudo-core during assembly and sets the trace bits according to the state of the TRACE/NOTRACE indicator of the PRINT statement.

The other program which may be used to call the simulator is PDPGENER (Backstrom 1970). This program accepts assembler object decks and loads them into pseudo-core. Under user commands, it can modify pseudo-core contents, punch object decks, set or clear trace bits, and then pass control to a simulator or other program. It passes the first simulator parm field to this called program. The second parm field is obtained from the characters after the program name on the PDPGENER EXECUTE command, e.g.

EXECUTE PDP11SIM,DUMP

3.9.2 Specifying the device configuration

At the start of simulation, a data set with ddname DEVIN is read. This specifies the names and characteristics of devices to be used for the simulation run. All devices named must be present as load modules in a partitioned data set referred to by the ddname DEVLIB. The DEVIN data set consists of card images containing the following fields:

DEVDDNAM — This field starts in column 1 and is delimited by one or more blanks. It specifies the ddname to be associated with this device. The field is mandatory, even though the device may not use any IBM360 I/O.

DEVNAME — This field is separated from surrounding fields by blanks. It is the member name of the device program load module in the DEVLIB partitioned data set. More than one device of the same name may be loaded provided different ddnames are used. The simulator uses the OS/360 LOAD macro to load the device program and connects it to the unibus via the DEVDEV field (Section 3.6).

MODIFICATION — The modification field is optional, and when present consists of a number of 'keyword = value' operands, separated from each other by commas, and containing no embedded blanks. The field may not extend past column 71 of the card. If it is necessary to continue a card, break the first card after a comma, and continue on the next card after column 1. A device program has default values for its register address, interrupt vector address and relevant device constants etc. These values may be changed by entries in this modification field. Device constants may be changed by DATA=(list) where the list contains up to 4 decimal integers, separated by commas. The meaning of these data values depends on the particular device concerned. Status register parameters may be changed by the following keywords:

ADR=(list)	the register address
CONT=(list)	the register contents
MR=(list)	the unibus read mask
MW=(list)	the unibus write mask
INTV=(list)	the interrupt vector address

In each case, the list consists of octal numbers separated by commas. Successive items in the list refer to successive status register data blocks. These will usually be in the order of their device addresses, but this depends on the assembly of the device program.

Comment cards may be included with the device specification cards. A comment card is one which has a blank in column 1 and is not a continuation card.

Each device normally has its own priority built in. It is recommended that high priority devices are specified before lower priority ones. Pseudo-devices should be loaded in decreasing order of their frequency of use. With two devices of the same priority, the one specified first is simulated as closer-on-the-bus.

3.9.3 Output from the simulator

All simulator output (excluding that from devices) is written for printing on the SYSPRINT data set. Output is in three parts as described in the following sections.

3.9.3.1 Device configurations

The device specification cards are printed as read. Syntax errors are briefly noted. A summary is then printed of all devices as seen by the simulator for the current run. If default characteristics are used, or if syntax errors are made, this print-out is useful in ascertaining what device characteristics are in effect.

3.9.3.2 Traced instructions

A line describing the current instruction is printed during simulation if the trace bit in the pseudo-core word containing the first byte of the instruction is set. A line describing a trap sequence of any sort is printed if the first byte of the interrupt vector has the trace bit set. An instruction causing a trap (e.g. IOT) is thus traced if either the instruction trace bit or the trap vector trace bit is set. For printing after simulation under control of the HISTORY option, all instructions and trap sequences are traced regardless of the trace bit settings. It is usually good practice to set the trace bits for the bus error and illegal instruction trap vectors, to ensure that these types of errors are traced if they occur.

The transfer of control to a device between instructions does not give rise to any printed output.

Printed lines contain information under the following headings. All values except the PDP11-time are in octal.

TIME	the PDP11-time in microseconds at the start of the current instruction or trap.
PS	the 16-bit processor status at the start of the current instruction or trap.
PC	the program counter (register 7) at the start of the current instruction or trap.
INSTRN	the machine code of the current instruction.
INSTRUCTION MNEMONIC	the mnemonic for the current instruction. The operator field mnemonics are standard except:

Breakpoint trap is '3'

Instructions which set or clear 1 to 3 condition code bits have the prefix 'SE' or 'CL', followed by as many of the letters N,Z,V,C as necessary to represent the bits operated on. If no bits are used, the mnemonic is 'NOP' and if all bits are set or cleared, 'SCC' or 'CCC' is the mnemonic.

Reserved instructions have the mnemonic 'RESV'. In the operand field, all digits appearing represent registers. Index word usages appear for example as 'X(1)' and the bracket notation (see Section 2.5) is exclusively used to represent indirect addressing. Branch instruction operands are written '. ±nn' where '.' is the start of the current instruction and nn is the octal byte displacement.

SOURCE FIELD — The following headings are used:

INDEX WORD — the index word, if any, in the operand.

REG BEFORE — the contents of the register involved in the operand, before the operand is decoded.

CORE ADDRESS — the address in core of the operand data. For mode zero, the register number is printed.

DATA — the operand data to be used in the instruction (for JMP and JSR, this is printed, even though it is not used). For byte instructions, only one byte (3 octal digits) is printed. The ASCII character (if printable) represented by the right 7 bits of the byte is also printed.

DESTINATION FIELD — the same subheadings as under source field are used.

RESULT — the result of the instruction, whether it is stored or not, is printed. The result is preceded by an asterisk if it is the same as the destination data. This feature may be useful in detecting redundant instructions in a program.

COND CODE — the N,Z,V and C bits of the condition code after the instruction execution are printed.

For those instructions which have operands not of mode-register type, the printed output is as follows:

Conditional Branch Instructions

The message BRANCH TAKEN or BRANCH NOT TAKEN is printed, depending on the result of the branch. The destination address of the branch instruction is printed in the 'destination core address' field.

JSR

The nominated register contents before the JSR is executed as printed under the 'source register before' field.

JSR and JMP

If mode zero is used, the message '*ILLEGAL MODE*' is printed.

RTI

The contents of the stack pointer (SP), the processor status (PS) and program counter (PC) after the RTI instruction execution are printed.

RTS

The contents of the register mentioned, as well as the SP, PS and PC after the RTS instruction execution are printed.

Trap sequences all have the same format of printed line, whether they are processor traps, instruction traps, or I/O traps. For processor and I/O traps, the interrupt type and the word 'INTERRUPT' are printed under the heading 'INSTRUCTION MNEMONIC'. For processor traps, the interrupt type may be one of BUS (odd address or time-out), ILLEGAL (JMP or JSR mode 0), TRACE (trap caused by T-bit of PS being set before previous instruction, STACK (stack overflow), or POWER (power-fail or auto-restart trap). For I/O traps, the interrupt type printed is the unique user-given ddname of the device interrupting. The trap vector location (VECTR) and the SP, PS and PC are printed for all interrupts.

3.9.3.3 End of simulation

When the simulator terminates, a reason for termination is always given. Such a reason may be generated by the CPU program or by one of the device programs. Messages generated by the CPU are self-explanatory and are

HALT INSTRUCTION ENCOUNTERED

WAIT INSTRUCTION ENCOUNTERED WITH NO HIGHER PRIORITY PENDING INTERRUPTS

BUS ERROR OCCURRED PROCESSING LAST TRAP

PRINT LIMIT EXCEEDED

In the last case, this is the print line limit set by the LIMIT parameter or the default limit of 2750 if LIMIT was not specified.

The contents of all PDP11 registers, the PS and the PDP11-time are always printed at the end of simulation.

An octal core dump and the history buffer print-out follow the termination message if the DUMP or HISTORY options have been selected.

3.9.4 Simulated devices and pseudo-devices currently available

Device programs representing the switch register, the line frequency clock and the teleprinter keyboard/reader and printer/punch have been written. These devices, together with a few useful pseudo-devices are described in the following sections. The mnemonic names used in the descriptions have been defined in Section 3.6. In all the devices except the switch register real time is involved, and the device speed is specified as one of the variable data words. It is recommended that unless extensive use is made of the WAIT instruction, the time constants of the devices used should be made very much smaller than their real-world values to avoid extensive simulation of the CPU waiting for a device. In the case of the keyboard/reader, for example, it is often suitable to give a time constant of 10 or even 0 microseconds instead of 100 milliseconds.

Many variations on these device programs could be written; the particular methods of simulating the devices described are presented as being realistic and generally useful.

Device programs for A/D converter, card reader and line printer are expected to be available in the near future.

3.9.4.1 SWR – Switch register as a device

The console switch register is simulated as a device. Its characteristics are as follows:

Quantity	Value	Comment
DEVNAME	SWR	
DEVNOSR	1	Only one addressable register
DATA	0	No meaning
SRADR	177570	Address of SWR
SRCONT	000000	Default switch setting – does not change during simulation
SRMR	177777	All bits may be read
SRMW	000000	Not applicable
SRINTV	000000	Not applicable

Entry	Special Action
DEVBTIME	Not used
SRBDATI	None
SRBDATO	Simulation terminates with message ATTEMPT TO WRITE TO SWR
SRBRESET	None
SRBEND	None

An example of a switch register specification is

```
SWITCH    SWR    CONT=010400
```

If the instruction MOV @#177570,R0 were then simulated, the result would be 010400.

3.9.4.2 CLOCK – line-frequency clock

This program simulates the operation of the line-frequency clock, type KW11-L. It has the following characteristics:

Quantity	Value	Comment
DEVNAME	CLOCK	
DEVNOSR	1	
DATA 1	20000	TC-time constant in microseconds Default 20 milliseconds
DATA 2	6	Priority, must be 4,5,6 or 7
SRADR	177546	
SRCONT	000000	Set to 000200 every TC microsecond
SRMR	000300	
SRMW	000100	
SRINTV	000100	

Entry	Special Action
DEVBTIME	Sets done, requests interrupt if done previously off and intr enb on. Requests another entry TC after this one.
SRBDATI	Clears done.
SRBDATO	Requests interrupt if intr enb changes from 0 to 1 and done is set.
SRBRESET	Clears intr enb and done, requests control TC from now.
SRBEND	None

An example of a clock specification is

CLOCK CLOCK DATA=2000

Every 2 milliseconds of PDP11-time this device will set its done flag and request an interrupt at priority 6 if its interrupt enable bit is set.

3.9.4.3 TTYK – teletype keyboard/reader

This program simulates the operation of the teletype keyboard/reader connected to the teletype control, model KL11. For the purposes of simulation, the keyboard/reader and the teleprinter/punch are separate devices. The maintenance function which connects these two devices is not simulated. This program has the following characteristics:

Quantity	Value	Comment
DEVNAME	TTYK	
DEVNOSR	2	2 registers, TKS and TKB
DATA 1	0	TC1 – reader speed
DATA 2	4	Priority of device
DATA 3	0	TC2 – delay before use of keyboard
DATA 4	0	TC3 – speed of operation of keyboard after initial delay
SRADR 1	177560	
SRCONT 1	000000	TKS – teleprinter keyboard status
SRMR 1	004300	
SRMW 1	000101	
SRINTV 1	000060	
SRADR 2	177562	
SRCONT 2	000000	TKB – teleprinter keyboard buffer
SRMR 2	000377	
SRMW 2	000000	Not applicable
SRINTV 2	000000	Not applicable

Entry	Special Action
DEVBTIME	Entered to simulate character transmission complete. One byte of data is transferred to TKB from DEVDDNAM data set, translated EBCDIC to ASCII unless from IBM360 papertape reader. Done flag in TKS is set. Interrupt requested if previous TKS state was done = 0, intr enb = 1.
SRBDATI 1	None
SRBDATO 1	If rdr enb is set, it is cleared again, busy is set and done cleared, and request for control is made TC1 from now. If intr enb set with previous state intr enb = 0, done = 1, interrupt request is made.
SRBRESET 1	If TC2 is non-zero, request for control at TC2 from now is made. TKS is set to 000200.
SRBEND 1	DEVDDNAM data set is closed and buffers are freed.
SRBDATI 2	If done not set, simulator terminates with message 'ATTEMPT TO READ devddnam TKB WHEN DONE NOT SET'. Otherwise clears done, clears TKB, clears interrupt request.
SRBDATO 2	Simulator terminates with message 'ATTEMPT TO WRITE TO devddnam TKB'.
SRBRESET 2	None
SRBEND 2	None

This program is written so that either keyboard operation (where the PDP11 program does not ask for input) or papertape reader operation (where input is requested by setting rdr enb) may be simulated. The reader is simulated if TC2 is zero. The papertape device transmits a character TC1 after the reader enable bit is set. The keyboard is simulated if TC2 is non-zero and rdr enb is never set. In this case, an operator transmitting the first character is simulated TC2 after simulation start or RESET and subsequent characters arrive TC3 after reading TKB thereafter. This process is stopped after a line-feed character (215 or 015 octal) and starts again with the next character TC2 after a further RESET. In the EBCDIC to ASCII translation for non-papertape IBM360 data sets, the character '|' (hex 4F) translates to linefeed (octal 215) and the character '—' (hex 6D) to carriage return (octal 212).

3.9.4.4 TTYP – teletype printer/punch

This program simulates the operation of a teleprinter/punch connected to the control module KL11. The simulated device has the following characteristics:

Quantity	Value	Comments
DEVNAME	TTYK	
DEVNOSR	2	2 registers, TPS and TPB
DATA 1	0	TC – time constant in microseconds
DATA 2	4	Priority of device
SRADR 1	177564	
SRCONT 1	000200	TPS – teleprinter status
SRMR 1	000304	
SRMW 1	000104	
SRINTV 1	000064	
SRADR 2	177566	
SRCONT 2	000000	TPB – teleprinter buffer
SRMR 2	000000	
SRMW 2	000377	
SRINTV 2	000000	Not applicable

Entry	Special Action
DEVBTIME	The character in TKB is translated to EBCDIC from ASCII and put in the DEVDDNAM output buffer for IBM360 output. Ready is set. If previous state of TPS was ready = 0, intr enb = 1, an interrupt is requested.
SRBDATI 1	None.
SRBDATO 1	Interrupt requested if register changed from intr enb = 0, ready = 1 to intr enb = 1.
SRBRESET 1	TPS set to 000200, interrupt cleared, request for control cleared.
SRBEND 1	Partially full IBM360 output buffer printed and DEVDDNAM data set closed.
SRBDATI 2	None
SRBDATO 2	If not ready, simulation terminates with message 'DATA MOVED TO devddname TPB WHEN NOT READY'. Otherwise, ready cleared, interrupt cleared, control requested TC from now.
SRBRESET 2	None
SRBEND 2	None

3.9.4.5 PFAIL – power failure pseudo-device

This program causes the simulator to enter a power-fail trap sequence after a given time. It does this by setting the power-fail trap bit inside the simulator itself. Its characteristics are:

Quantity	Value	Comments
DEVNAME	PFAIL	
DEVNOSR	1	
DATA 1	200000	TC1 – microseconds before power-fail
DATA 2	2000	TC2 – microseconds allowed for power-fail processing (default 2 milliseconds).
SRADR	0	Not applicable
SRCONT	0	Not applicable
SRMR	0	Not applicable
SRMW	0	Not applicable
SRINTV	0	Not applicable

Entry	Special Action
DEVBTIME	1st entry sets power fail condition, requests 2nd entry TC2; from now 2nd entry checks that HALT has been encountered and power-auto restart processing has started. If not, termination occurs with message 'POWER FAIL HANDLING NOT COMPLETED IN TIME'.
SRBDATI	Not applicable
SRBDATO	Not applicable
SRBRESET	1st entry makes request for control TC1 from now, subsequent entries do nothing.
SRBEND	None

3.9.4.6 TLIMIT – simulation time limit

This program causes the simulation to terminate after a given PDP11-time. This can be useful if a run extending to long PDP11-times is required with no trace printout, and the PDP11 program is such that there is no other easy way of stopping the simulator. It may be convenient to use the history buffer printout feature in conjunction with the time limit. The characteristics of this device are:

Quantity	Value	Comments
DEVNAME	TLIMIT	
DEVNOSR	1	
DATA 1	20000	TC – microseconds of PDP11-time, default 20 milliseconds
SRADR	0	Not applicable
SRCONT	0	Not applicable
SRMR	0	Not applicable
SRMW	0	Not applicable
SRINTV	0	Not applicable

Entry	Special Action
DEVBTIME	Terminates simulation with message 'TERMINATION REQUESTED BY TLIMIT PSEUDO-DEVICE'
SRBDATI	Not used
SRBDATO	Not used
SRBRESET	1st entry requests control after TC
SRBEND	None

3.9.4.7 DUMMY – A dummy control register

This program is a means of attaching any addressable device register on to the unibus. This may be required, for example, if a PDP11 program contains a reference to a device register, but a program to simulate the device has not yet been written. Although the device cannot be simulated, its address can be made to respond to a unibus command, thereby avoiding a bus error trap. Its characteristics are:

Quantity	Value	Comments
DEVNAME	DUMMY	
DEVNOSR	1	
SRADR	000000	Must be set by user
SRCONT	000000	
SRMR	177777	
SRMW	177777	
SRINTV	000000	

No action is taken on any control entries.

3.9.4.8 TRDST – Trace destination reference

This program traces the last instruction independently of the trace option, if it is a single or double operand instruction, and if its destination field references a specified PDP11 address. This facility is useful if looking for a program bug which incorrectly modifies a word of known address. Use of this pseudo-device will make the simulator run slower as it requests control after every instruction. Its characteristics are:

Quantity	Value	Comments
DEVNAME	TRDST	
DEVNOSR	1	
DATA	0	Not used
SRADR	0	The in-core address to be traced
SRCONT	0	Not used
SRMR	0	Not used
SRMW	0	Not used
SRINTV	0	Not used

Entry	Special Action
DEVBTIME	Entered after every instruction. Checks instruction type and destination address. If a match occurs, calls simulator print routine. Requests control again after next instruction.
SRBDATI	Not used
SRBDATO	Not used
SRBRESET	Requests control after next instruction
SRBEND	None

3.10 Testing of the Simulator

The simulator, when written, was tested and 'debugged' by using the diagnostic tests supplied by DEC for the PDP11 computer itself (see Appendix VII). As well as testing the functions and condition code bit settings of all the CPU instructions, these tests check that processor traps and I/O traps work correctly and in the correct sequence if they occur simultaneously. The tests normally run in an endless loop; each one was modified so as to execute its loop only once. The tests were loaded into pseudo-core direct from the DEC-supplied object tapes by assembling and simulating the bootstrap loader to load the binary loader into pseudo-core from paper tape, and then simulating the binary loader to load each test program in turn.

A number of the finer points of operation of the PDP11 computer were learnt from running these tests. Some of these are:

(i) In a double operand instruction with source mode 0, destination mode 2,3,4 or 5 and using the same register as the source (e.g. MOV RO,(RO)+), the value used for the source operand data is the register contents after incrementing or decrementing for the destination operand.

(ii) Trace traps do not occur after instructions causing an instruction trap.

(iii) I/O device interrupt requests are cleared at the time the request is granted. Thus it is possible to run a device service routine at a priority lower than that of the device if one can be certain the device will not interrupt again.

(iv) If a device requests an interrupt as a direct result of an instruction which modifies one of its status registers, the interrupt is not granted until one more instruction is executed.

(v) I/O interrupts cannot occur until one instruction after any trap sequence, be it a processor trap, instruction trap or I/O trap.

(vi) In instructions which set the condition code bits, they are set before any result is stored. This is manifest in the instruction MOV #0,PS which leaves the Z-bit of the PS clear, despite the fact that the result is zero.

All of the above aspects of behaviour are simulated correctly, and it can now be claimed that there are no known aspects of the behaviour of the PDP11 computer which are not simulated correctly.

As a further test, and demonstration of the simulator's capability, a simulated PDP11 was configured with three teletype reader/printers, two on priority 4 and one on priority 5. One of these had input faster than output, the second had input slower, and the third had them the same speed; all the teletype speeds were different. The three teletypes were driven from a single re-entrant interrupt program which copied data from the readers to the printers via a core buffer. The simulated input was in two cases from the IBM360 card reader input stream, and in the other from the papertape reader. The simulated output from each was directed to the IBM360 printer. The PDP11 program and part of its simulation is listed in Appendix VII.

3.11 Simulation Speed

The time ratio between the speed of execution on the IBM360/50 of a program on the simulated PDP11 and the speed of execution of the same program on the PDP11 computer, as calculated by the simulator, was determined using a variant of the TLIMIT pseudo-device which prints this ratio in its termination message. The ratio was dependent on the type of instruction in the program being simulated, but varied from around 160–200 with no instruction tracing to about 800 with full tracing.

4. CONCLUSIONS

The assembler, PDP11ASM, provides a convenient method of assembling and preparing programs for the PDP11 computer. It is flexible with respect to internal table sizes, hence its core requirements are easily tailored to fit any given assembly. Error diagnostic messages are comprehensive and make the assembler easy to use.

The simulator, PDP11SIM, provides a faithful simulation of the PDP11 computer in all respects except speed. The selective instruction tracing facility gives detailed information about the PDP11 behaviour and this is not available when using the computer itself. The input-output simulation capabilities are very powerful, and the I/O device configuration can be specified as required for each run. The definition of a software interface for simulated devices enables further programs to be written to simulate the behaviour of any PDP11 device. The generality of the PDP11 hardware device interface, while suggesting the form of the software interface, is not essential to it. Hence the I/O simulation techniques used could be implemented for other mini-computer simulators.

5. ACKNOWLEDGEMENTS

I wish to thank members of the Applied Mathematics and Computing Section at Lucas Heights for comments and criticisms during the course of this work. Particular thanks are due to Mr. R. Backstrom who wrote the section of the simulator which sets up and prints the device configuration.

6. REFERENCES

Backstrom, R. P. (1970). – An IBM360 program to reconstruct symbolic source listings for PDP 9L Object Code. AAEC/E210.

Digital Equipment Corporation (1970). – PDP11 Handbook, 2nd ed.

Sanger, P. L. (1970). – NOVASM and NOVASIM – an assembler and simulator for the NOVA and SUPERNOVA computer written to run on an IBM360 computer. AAEC/TM566.

APPENDIX I

ASSEMBLER SOURCE STATEMENT SYNTAX OPTIONS

The following table lists the syntax extensions to the PAL-11 language which are recognised by PDP11ASM:

PAL-11A Syntax	PDP11ASM Option	Meaning
;	/	comment character
:	blank	label delimiter
=	EQU	'Equals' assembler directive
.END	END	'End' assembler directive
#	.	immediate operand symbol
@A	(A)	indirect addressing
!	|	'OR' operator in expression
.EOT	none	End-of-tape indication, not recognised

The following table lists the syntax extensions to the various operand addressing modes:

PAL-11A Syntax	PDP11ASM Option	Addressing Mode
R		0
@R	(R)	1
(R)+		2
@(R)+	((R)+)	3
–(R)		4
@–(R)	(–(R))	5
X(R)		6
@X(R)	(X(R))	7
@(R)	((R))	7
#A	=A	2, immediate
@#A	(=A)	3, deferred immediate
A		6, relative
@A	(A)	7, deferred relative

APPENDIX II

ASSEMBLER-RECOGNISED MACHINE INSTRUCTION MNEMONICS AND THEIR OBJECT CODE

1. DOUBLE OPERAND INSTRUCTIONS

Mnemonic	Op. Code	Meaning
MOV	010000	Move
MOVB	110000	Move byte
CMP	020000	Compare
CMPB	120000	Compare byte
BIT	030000	Bit test
BITB	130000	Bit test byte
BIC	040000	Bit clear
BIC	140000	Bit clear byte
BIS	050000	Bit set
BIS	150000	Bit set byte
ADD	060000	Add
SUB	160000	Subtract

2. SINGLE OPERAND INSTRUCTIONS

Mnemonic	Op. Code	Meaning
CLR	005000	Clear
CLRB	105000	Clear byte
COM	005100	Complement
COMB	105100	Complement byte
INC	005200	Increment
INCB	105200	Increment byte
DEC	005300	Decrement
DECB	105300	Decrement byte
NEG	005400	Negate
NEGB	105400	Negate byte
ADC	005500	Add carry
ADCB	105500	Add carry byte
SBC	005600	Subtract carry
SBCB	105600	Subtract carry byte
TST	005700	Test

APPENDIX II (continued)

Mnemonic	Op. Code	Meaning
TSTB	105700	Test byte
ROR	006000	Rotate right
RORB	106000	Rotate right byte
ROL	006100	Rotate left
ROLB	106100	Rotate left byte
ASR	006200	Arithmetic shift right
ASRB	106200	Arithmetic shift right byte
ASL	006300	Arithmetic shift left
ASLB	106300	Arithmetic shift left byte
JMP	000100	Jump
SWAB	000300	Swap bytes

3. CONDITIONAL BRANCH INSTRUCTIONS

Mnemonic	Op. Code	Meaning
BR	000400	Branch always
BNE	001000	Branch if not equal
BEQ	001400	Branch if equal
BGE	002000	Branch if greater or equal
BLT	002400	Branch if less than
BGT	003000	Branch if greater than
BLE	003400	Branch if less or equal
BPL	100000	Branch if plus
BMI	100400	Branch if minus
BHI	101000	Branch if higher
BLOS	101400	Branch if lower or same
BVC	102000	Branch if overflow clear
BVS	102400	Branch if overflow set
BCC	103000	Branch if carry clear
BHIS	103000	Branch if higher or same
BCS	103400	Branch if carry set
BLO	103400	Branch if lower

APPENDIX II (continued)

4. CONDITION CODE OPERATORS

Mnemonic	Op. Code	Meaning
SEN	000270	Set negative
SEZ	000264	Set zero
SEV	000262	Set overflow
SEC	000261	Set carry
CLN	000250	Clear negative
CLZ	000244	Clear zero
CLV	000242	Clear overflow
CLC	000241	Clear carry
SCC	000277	Set all condition codes
CCC	000257	Clear all condition codes
NOP	000240	No operation

5. MISCELLANEOUS INSTRUCTIONS

Mnemonic	Op. Code	Meaning
JSR	004000	Jump to subroutine
RTS	000200	Return from subroutine
HALT	000000	Halt
WAIT	000001	Wait for interrupt
RTI	000002	Return from interrupt
IOT	000004	Input/output trap
RESET	000005	Reset
EMT	104000	Emulator trap
TRAP	104400	Trap

APPENDIX III

ERROR MESSAGES PRODUCED BY THE ASSEMBLER

1. Severe errors noted before assembly begins (all terminate with condition code 16).

SYSIN DATA SET CANNOT BE OPENED

SYSUT1 DATA SET CANNOT BE OPENED

SYSPUNCH DATA SET CANNOT BE OPENED

INSUFFICIENT CORE AVAILABLE – ASSEMBLY TERMINATED

2. Severe error noted during assembly, causing termination with condition code 16.

SYMBOL TABLE OVERFLOW – ASSEMBLY TERMINATED AT STMT nnn.

nnn is the statement number of the card containing a symbol definition which cannot be put in the symbol table. This message usually occurs during Pass 1, in which case no source statement listing appears.

3. Source statement errors.

Cond. Code	Message
4	END-OF-FILE ON SYSIN, END CARD INSERTED
4	INVALID OPERAND
4	MISSING LABEL FIELD OR TOO MANY LABELS
4	ILLEGAL USE OF % BEFORE NON-NUMERIC FIELD
4	CARD FULLY PUNCHED, TREATED AS COMMENT FROM HERE
4	TOO MANY ERRORS ON THIS CARD, SOME NOT FLAGGED
4	XREF TABLE FULL, NO MORE REFERENCES SAVED
8	LABEL TOO LONG, TRUNCATED TO 8 CHARACTERS
8	ILLEGAL CHARACTER IN LABEL, LABEL TRUNCATED
8	SYMBOL NOT PREDEFINED, A-TYPE AND ZERO ASSUMED
8	SYMBOL UNDEFINED, A-TYPE AND ZERO ASSUMED
8	MULTIPLY DEFINED SYMBOL – PREVIOUS VALUE USED
8	ADDRESS-TYPE SYMBOL IN REGISTER EXPRESSION
8	REG SYMBOL IN ADDRESS EXPRESSION OR NOT PREDEFINED
8	ILLEGAL EXPRESSION OPERATOR USAGE
8	INVALID CHARACTER IN NUMERIC FIELD
8	8 or 9 IN OCTAL NUMBER
8	NUMBER TOO LARGE, TRUNCATED TO 16 BITS
8	EXPRN. VALUE TOO LARGE, TRUNCATED TO 16 BITS
8	REF EXPRN VALUE OUTSIDE RANGE 0–7, 0 ASSUMED

APPENDIX III (continued)

Cond. Code	Message
8	EXPRESSION MISSING, ZERO VALUE ASSUMED
8	CLOSING BRACKET MISSING
8	MISSING OPERAND
8	INVALID CHARACTER IN OPERAND
8	BRANCH DISPLACEMENT TOO LARGE, ZERO ASSUMED
8	ODD BRANCH ADDRESS, ROUNDED DOWN
8	EMT OR TRAP OPERAND TOO LARGE, TRUNCATED
12	ODD OPERAND ADDRESS IN WORD INSTRUCTION
12	REGISTER MODE ADDRESSING ILLEGAL FOR JSR OR JMP
12	DS EXPRN. LARGER THAN CORE SIZE, CORE SIZE USED
12	ASSEMBLY SPECIFIED OUTSIDE CORE, .=0 ASSUMED

APPENDIX IV

LOGICAL STRUCTURE OF THE ASSEMBLER PROGRAM

PDP11ASM is a two-pass Assembler. In Pass 1 all instruction lengths are evaluated and symbol definitions are calculated and stored in a symbol table. Additionally, operand modes and registers are determined, all address expressions in ORG, EQU and DS and all register expressions are evaluated and register symbols cross-referenced. Each source statement read into Pass 1 is stored in an 88 byte record on a work data set (ddname SYSUT1), the last 8 bytes of this record containing information which is used by Pass 2. If any assembly errors are detected during Pass 1, information on these is stored in an additional 88 byte record of the work data set. Thus the output from Pass 1 is the work data set and an in-core symbol table. The format of the work data set records is

(i) Source Statement Record

bytes 1–80 Source statement

bytes 81–82 An S-constant for the Pass 2 operator handler routine

bytes 83–84 The object code of the first word of the instruction so far, including modes and register numbers

byte 85 The length of this instruction in bytes

byte 86 The card column at which the operand starts

byte 87 The length in bytes of the operand

byte 88 A flag byte as follows

- bit 0 Errors detected, following record describes them
- bit 1 Current card is a DS, storage requested is in bytes 83–84, not byte 85
- bit 2 Source syntax errors – no Pass 2 processing
- bit 3 Destination syntax errors – no Pass 2 processing
- bit 4 Source is relative or immediate
- bit 5 Destination is relative or immediate
- bit 6 Unused
- bit 7 This is an END statement

(ii) Error Description Record

bytes 1–2 2* number of errors so far

byte 3 column number of 1st error

byte 4 error number of 1st error

bytes 5–6 same as bytes 3–4 for 2nd error

etc. repeated for up to 43 errors total

APPENDIX IV (continued)

These records may be listed under control of a PASS1/NOPASS1 parameter on the PRINT statement. This is not mentioned in Section 2.7.8, as it is used only for assembler debugging.

In Pass 2, all address expressions are evaluated, and address terms cross-referenced. Extra instruction words corresponding to relative, immediate and indexed addressing are determined. The output of Pass 2 is a listing, an object deck and a pseudo-core, all of these being controlled by user options on the PRINT statement or PARM field.

APPENDIX V

PDP11 INSTRUCTION TIMES USED BY THE SIMULATOR

The times quoted here were derived from those given in the PDP11 Handbook (Digital Equipment Corporation 1970).

Operation or Instruction	Time Microsecond
DATI from memory or a device register including	1.2
Instruction fetch and stack access	
DATO	0.0
Extra time to address odd byte	0.6
MOV, CMP, BIS, ADD, SUB	1.1
BIT, BIC, BIS	1.7
Source not mode 0	0.3
Destination not mode 0	0.2
CMP, TST with destination not mode 0	–0.5
Single operand instructions except JMP	1.1
JMP	0.0
RORB, ROLB, ASLB, ASRB extra for odd byte	0.6
Branch when taken	1.4
Branch when not taken	0.5
Condition code operators	0.3
JSR	3.0
RTS	1.1
RTI	1.2
HALT, WAIT	0.6
Trap sequence (I/O, Processor or Instruction)	5.7
RESET	19,998.8

Examples of how this table is used follow:

(i) MOVB 1,%1 where the object code is 116701, displacement

Instruction fetch	1.2
MOVB	1.1
Source mode 6	0.3
Source DATI operations (2)	2.4
Source odd byte reference	0.6
Destination mode 0	0.0
	5.6

APPENDIX V (continued)

(ii) JMP @#0 where the object code is 000137,000000

Instruction fetch	1.2
JMP	0.0
Destination mode 3	0.2
Destination DATI operations (2)	2.4
	3.8

(iii) ROLB (A) where location A contains an odd address. Object code is 106177, displacement.

Instruction fetch	1.2
ROLB	1.1
Destination mode 7	0.2
Destination DATI operations (3)	3.6
Destination extra for odd byte	0.6
ROLB extra for odd byte	0.6
	7.3

APPENDIX VI

SAMPLE SIMULATED DEVICE PROGRAM

```
                                EXTERNAL SYMBOL DICTIONARY                    PAGE    1
                                                                          08.24  2/02/73

SYMBOL    TYPE ID   ADDR   LENGTH LD ID

CLOCK       SD  01 000000 0000AE


                                                                              PAGE    1

 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                  F01MAY72   2/02/73
                                    1          COPY  ELSDEV                        00001000
```

```
LOC    OBJECT CODE   ADDR1 ADDR2  STMT   SOURCE STATEMENT                                              F01MAY72   2/02/73

                                    3 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * 00002000
                                    4 *                                                                       * 00003000
                                    5 *        EACH DEVICE HAS A DATA BLOCK ASSOCIATED WITH IT.               * 00004000
                                    6 *        THIS STARTS AS IN THE DEVICE DSECT, AND IS IMMEDIATELY         * 00005000
                                    7 *        FOLLOWED BY AS MANY BLOCKS OF THE STATREG DSECT AS THERE ARE   * 00006000
                                    8 *        ADDRESSABLE 16 BIT REGISTERS IN THE TOP 4K OF PDP11 CORE       * 00007000
                                    9 *        ASSOCIATED WITH THE DEVICE.  FOR THIS PURPOSE THERE IS NO      * 00008000
                                   10 *        DISTINCTION BETWEEN STATUS REGISTERS AND DATA BUFFER REGISTERS 00009000
                                   11 *        THEY ARE HANDLED IDENTICALLY BY THE SIMULATOR MAIN PROGRAM     * 00010000
                                   12 *        AND ANY DIFFERENCES ARE IN THE DEVICE PROGRAM.                 * 00011000
                                   13 *        THE DATA BLOCKS CONTAIN A NUMBER OF BRANCH INSTRUCTIONS  --    * 00012000
                                   14 *        THE SIMULATOR LINKS TO THESE AT VARIOUS TIMES AND AND THEY     * 00013000
                                   15 *        MUST BRANCH TO THE "DEVICE PROGRAM" WHICH FOLLOWS THE LAST     * 00014000
                                   16 *        STATREG DATA BLOCK AND LOOKS AFTER THE DEVICE SPECIFIC         * 00015000
                                   17 *        FUNCTIONS.                                                     * 00016000
                                   18 *        THERE ARE FIVE DIFFERENT TYPES OF ENTRY TO THE DEVICE          * 00017000
                                   19 *        PROGRAM, AND AT ANY OF THESE, REQUESTS FOR FUTURE DEVICE       * 00018000
                                   20 *        SERVICE OR REQUESTS FOR INTERRUPT MAY BOTH BE EITHER MADE      * 00019000
                                   21 *        OR CANCELLED.  THE FIVE ENTRY POINTS ARE AS FOLLOWS.           * 00020000
                                   22 *                                                                       * 00021000
                                   23 *        1)  SRBRESET  AT START OF SIMULATION AND ON USE OF THE         * 00022000
                                   24 *                      RESET INSTRUCTION, ENTRY IS MADE HERE TO         * 00023000
                                   25 *                      EVERY DEVICE ON THE BUS                          * 00024000
                                   26 *        2)  SRBEND    JUST BEFORE THE END OF A SIMULATION RUN, ENTRY   * 00025000
                                   27 *                      IS MADE HERE FOR EVERY DEVICE.                     00026000
                                   28 *        3)  DEVBTIME  REQUESTS MAY BE MADE BY A DEVICE FOR CONTROL     * 00027000
                                   29 *                      AT A SPECIFIED PDP11 TIME.  ENTRY IS THROUGH     * 00028000
                                   30 *                      DEVBTIME AT THIS TIME.                             00029000
                                   31 *        4)  SRBDATI   ENTRY IS MADE HERE AFTER ANY READING BY THE      * 00030000
                                   32 *                      CPU OF THE DATA IN THIS REGISTER.                * 00031000
                                   33 *        5)  SRBDATC   ENTRY IS MADE HERE AFTER WRITING BY THE CPU      * 00032000
                                   34 *                      TO THIS DATA REGISTER                              00033000
                                   35 *                                                                       * 00034000
                                   36 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * 00035000
                                   37 *                                                                       * 00036000
000000                             38 DEVICE   DSECT                                                            00037000
000000                             39 DEVDEV   DS    A                  NEXT DEVICE DATA BLOCK.  ALL            00038000
                                   40 *                                 DEVICES ARE ATTACHED TO THE             00039000
                                   41 *                                 SIMULATOR VIA THIS QUEUE.               00040000
000004                             42 DEVTIME  DS    A                  TIME ON PDP11 SIMULATED REAL            00041000
                                   43 *                                 TIME CLOCK AT WHICH ENTRY TO            00042000
                                   44 *                                 DEVBTIME IS NEXT REQUIRED               00043000
000008                             45 DEVBTIME DS    A                  CONTAINS A BRANCH INSTRUCTION           00044000
                                   46 *                                 UNDER A USING DEVICE,R10 WHICH          00045000
                                   47 *                                 IS LINKED TO WHEN TIME IS UP.           00046000
00000C                             48 DEVNOSR  DS    A                  THE NUMBER OF 16 BIT REGISTERS          00047000
                                   49 *                                 ADDRESSABLE IN THE TOP 4K OF            00048000
                                   50 *                                 CORE AND ASSOCIATED WITH THIS           00049000
                                   51 *                                 DEVICE                                  00050000
000010                             52 DEVDATA  DS    4A                 SPACE FOR VARIABLE DEVICE DATA          00051000
000020                             53 DEVDDNAM DS    CL8                SPACE FOR DDNAME IF ANY                 00052000
000028                             54 DEVNAME  DS    CL8                NAME OF THE DEVICE - MUST MATCH         00053000
                                   55 *                                 THE MEMBER NAME IN DEVLIB               00054000
000030                             56 DEVSR    DS    A                  ADDR OF 1ST STATUS REGISTER             00055000
                                   57 *                                 DATA BLOCK                              00056000
```

F01MAY72 2/02/73

```
 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT
000000                            59 STATREG  DSECT                                00058000
                                  60 *                                             00059000
                                  61 *                                             00060000
                                  62 *                                             00061000
                                  63 *                                             00062000
                                  64 *                                             00063000
000000                            65 SRADR    DS    H                              00064000
                                  66 *                                             00065000
000002                            67 SRCONT   DS    H                              00066000
000004                            68 SRMR     DS    H                              00067000
                                  69 *                                             00068000
000006                            70 SRMW     DS    H                              00069000
                                  71 *                                             00070000
000008                            72 SRINTV   DS    H                              00071000
                                  73 *                                             00072000
                                  74 *                                             00073000
00000A                            75 SRPRTY   DS    H                              00074000
                                  76 *                                             00075000
                                  77 *                                             00076000
00000C                            78 SRBDATI  DS    A                              00077000
000010                            79 SRBDATO  DS    A                              00078000
000014                            80 SRBRESET DS    A                              00079000
                                  81 *                                             00080000
000018                            82 SRBEND   DS    A                              00081000
                                  83 *                                             00082000
00001C                            84 SRSRB    DS    0A                             00083000

THIS DSECT DESCRIBES THE
BLOCK ASSOCIATED WITH EACH
16-BIT REGISTER BELONGING TO
A DEVICE.  THERE ARE DEVNOSR
OF THESE BLOCKS
THE 16 BIT ADDRESS BY WHICH
THIS REGISTER IS ADDRESSED
THE CONTENTS OF THE REGISTER
MASK FOR THE BITS WHICH MAY BE
READ BY THE CPU
MASK FOR THE BITS WHICH MAY
BE WRITTEN BY THE CPU
ADDRESS OF THE INTERRUPT VECTOR
IN LOW CORE ASSOCIATED WITH THIS
REGISTER
THE PRIORITY OF THE INTERRUPT
CURRENTLY PENDING (ZERO IF NONE
PENDING).
ENTRY HERE AFTER DATI BY CPU
ENTRY HERE AFTER DATO BY CPU
ENTRY HERE AT SIMULATION START
AND IN RESET INSTRUCTION
ENTRY HERE AT SIMULATION END
CAN BE USED TO CLOSE DCB'S ETC
NEXT STATREG BLOCK
```

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                F01MAY72   2/02/73

                                   86 *       CONTENTS OF REGISTERS ON ENTRY AND EXIT TO THE DEVICE ROUTINES 00085000

                                   88 *       13    SAVE AREA - MAY BE USED BY THE DEVICE PROGRAM TO         00087000
                                   89 *             DO 360 I/O                                               00088000
                                   90 *             ALSO ACCESSES DATA UNDER ELSDATA DSECT                   00089000
                                   91 *       14,15,0,1  UNUSED - MAY BE USED FOR 360 ACCESS METHODS ETC.    00090000
                                   92 *       15    ENTERS ZERO, MUST BE ZERO FOR NORMAL RETURN.             00091000
                                   93 *             SET TO ADDRESS OF A DC AL1(LENGTH),C'MESSAGE' IF         00092000
                                   94 *             SIMULATION IS TO BE TERMINATED WITH A MESSAGE            00093000
                                   95 *       12    MUST NOT BE USED BY DEVICE ROUTINES                      00094000
                                   96 *       11    RETURN ADDRESS TO SIMULATOR                              00095000
                                   97 *       10    BASE ADDRESS OF DEVICE.  DEVICE PROGRAM MAY BE           00096000
                                   98 *             ASSEMBLED WITH A USING DEVICE,R10                        00097000
                                   99 *       9     BASE ADDRESS OF STATUS REGISTER BLOCK -                  00098000
                                  100 *       8     CURRENT PDP11 SIMULATED REAL TIME - MAY BE UPDATED IN    00099000
                                  101 *             THE RARE EVENT THAT THIS IS REQUIRED                     00100000
                                  102 *       7     TIME OF CURRENT REQUEST FOR TIME SERVICE.  WILL USUALLY  00101000
                                  103 *             BE CLEARED OR RESET ON ENTRY AT DEVBTIME                 00102000
                                  104 *       6     PRIORITY OF CURRENT PENDING INTERRUPT (ZERO IF NONE)     00103000
                                  105 *             MAY BE SET TO MAKE A REQUEST OR CLEARED TO REMOVE ONE    00104000
                                  106 *       5     OLD CONTENTS OF REGISTER (UNMASKED) FOR SRBDATI AND      00105000
                                  107 *             SRBDATO ENTRIES                                          00106000
                                  108 *       4     NEW CONTENTS OF REGISTER BEFORE MASKING FOR SRBDATO      00107000
                                  109 *             ZERO FOR SRBDATI                                         00108000
                                  110 *       3     SPARE - MAY BE USED                                      00109000
                                  111 *       2     SPARE - MAY BE USED                                      00110000
```

```
 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                          F01MAY72   2/02/73
000000                            113 CLOCK    CSECT                                          00003000

                                  115 *        DEVICE DATA BLOCK                              00005000

000000                            117          USING *,R10                                    00007000
000000 00000000                   118          DC    A(0)          NEXT DEVICE ADDRESS      00008000
000004 00000000                   119          DC    A(0)          TIME REQUEST             00009000
000008 47F0 A05C          0005C   120          B     CLTIME                                 00010000
00000C 00000001                   121          DC    A(1)          1 STATUS REGISTER        00011000
000010 00004E20                   122 CL1C     DC    A(20000)      20 MS DEFAULT TIME CONST 00012000
000014 00000006                   123 CL2C     DC    A(6)          DEFAULT PRIORITY FOR CLOCK 00013000
000018 0000000000000000           124          DC    A(0,0),CL8' ' SPARE                    00014000
000028 C3D3D6C3D2404040           125          DC    CL8'CLOCK'    DEVICE NAME              00015000

                                  127 *        STATUS REGISTER DATA BLOCK                     00017000

000030 FF66                       129          DC    X'FF66'       OCTAL 177546 LKS ADDRESS 00019000
000032 0000                       130 LKS      DC    Y(0)          REGISTER CONTNETS        00020000
000034 00C0                       131          DC    Y(X'00C0')    READABLE BITS            00021000
000036 0040                       132          DC    Y(X'0040')    WRITABLE BITS            00022000
000038 0040                       133          DC    Y(X'0040')    100 OCTAL - TRAP VECTOR  00023000
00003A 0000                       134          DC    Y(0)          PRTY REQUEST             00024000
00003C 47F0 B000          00000   135          B     0(0,R11)      NO ACTION ON DATI        00025000
000040 47F0 A07A          0007A   136          B     CLDATO                                 00026000
000044 47F0 A04A          0004A   137          B     CLRESET                                00027000
000048 07FB                       138          BR    R11                                    00028000

                                  140 *        DEVICE PROGRAM                                 00030000

00004A 5870 A010          00010   142 CLRESET  L     R7,CL1C       TIME CONST IN MICROSEC   00032000
00004E 4C70 A0A8          000A8   143          MH    R7,=Y(0)      CONVERT TO TENTHS        00033000
000052 1E78                       144          ALR   R7,R8         REQUEST TIME SERVICE     00034000
000054 9280 A033       00033      145          MVI   LKS+1,X'80'   SET DONE, CLEAR INT ENB   00035000
000058 1B66                       146          SR    R6,R6         CLEAR PENDING INTERRUPT  00036000
00005A 07FB                       147          BR    R11           RETURN                   00037000
00005C 5800 A010          00010   148 CLTIME   L     R0,CL1C       TIME CONST               00038000
000060 4C00 A0AA          000AA   149          MH    R0,=Y(10)     IN TEMTHS OF MICROSEC    00039000
000064 1E70                       150          ALR   R7,R0         SET FUTURE REQUEST       00040000
000066 9680 A033       00033      151          OI    LKS+1,X'80'   SET DONE                 00041000
00006A 4450 A0A0          000A0   152          EX    R5,CL1X       SEE IF INT WAS ENABLED   00042000
00006E 07EB                       153          BCR   14,R11        RETURN IF NOT            00043000
000070 5860 A014          00014   154          L     R6,CL2C       REQUEST INTERRUPT        00044000
000074 8960 0005          00005   155          SLL   R6,5          SHIFT TO LEFT OF PS BYTE 00045000
000078 07FB                       156          BR    R11           RETURN                   00046000

00007A 9140 A033       00033      158 CLDATO   TM    LKS+1,X'40'   WAS INTEN SET            00048000
00007E 4710 A08C          0008C   159          BC    1,DAT01       BR IF SO                 00049000
000082 4160 0000          00000   160          LA    R6,0          CLEAR INTERRUPT REQUEST  00050000
000086 947F A033       00033      161          NI    LKS+1,X'7F'   CLEAR DONE               00051000
00008A 07FB                       162          BR    R11                                    00052000
00008C 9680 A033       00033      163 DAT01    OI    LKS+1,X'80'   SET INT MONITOR         00053000
000090 4450 A0A4          000A4   164          EX    R5,CL2X       WAS DONE SET AND INT NOT 00054000
000094 077B                       165          BCR   7,R11         RETURN IF NOT            00055000
000096 5860 A014          00014   166          L     R6,CL2C       REQUEST INTERRUPT        00056000
00009A 8960 0005          00005   167          SLL   R6,5          SHIFT TO LEFT OF PS BYTE 00057000
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                F01MAY72   2/02/73

00009E 07FB                       168          BR    R11                            00058000
0000A0 9100 A0AC      000AC       169 CL1X     TM    =X'40',0                       00059000
0000A4 9500 A0AD      000AD       170 CL2X     CLI   =X'80',0                       00060000
0000A8                            171          LTORG                                00061000
0000A8 0000                       172                =Y(0)
0000AA 000A                       173                =Y(10)
0000AC 40                         174                =X'40'
0000AD 80                         175                =X'80'
000000                            176 R0       EQU   0                              00062000
000005                            177 R5       EQU   5                              00063000
000006                            178 R6       EQU   6                              00064000
000007                            179 R7       EQU   7                              00065000
000008                            180 R8       EQU   8                              00066000
000009                            181 R9       EQU   9                              00067000
00000A                            182 R10      EQU   10                             00068000
00000B                            183 R11      EQU   11                             00069000
00000F                            184 R15      EQU   15                             00070000
                                  185          END                                  00071000
```

SYMBOL	LEN	VALUE	DEFN	REFERENCES
CLDATO	00004	00007A	00158	0136
CLOCK	00001	000000	00113	
CLRESET	00004	00004A	00142	0137
CLTIME	00004	00005C	00148	0120
CL1C	00004	000010	00122	0142 0148
CL1X	00004	0000A0	00169	0152
CL2C	00004	000014	00123	0154 0166
CL2X	00004	0000A4	00170	0164
DATO1	00004	00008C	00163	0159
DEVBTIME	00004	000008	00045	
DEVDATA	00004	000010	00052	
DEVDDNAM	00008	000020	00053	
DEVDEV	00004	000000	00039	
DEVICE	00001	000000	00038	
DEVNAME	00008	000028	00054	
DEVNOSR	00004	00000C	00048	
DEVSR	00004	000030	00056	
DEVTIME	00004	000004	00042	
LKS	00002	000032	00130	0145 0151 0158 0161 0163
R0	00001	000000	00176	0148 0149 0150
R10	00001	00000A	00182	0117
R11	00001	00000B	00183	0135 0138 0147 0153 0156 0162 0165 0168
R15	00001	00000F	00184	
R5	00001	000005	00177	0152 0164
R6	00001	000006	00178	0146 0146 0154 0155 0160 0166 0167
R7	00001	000007	00179	0142 0143 0144 0150
R8	00001	000008	00180	0144
R9	00001	000009	00181	
SRADR	00002	000000	00065	
SRBDATI	00004	00000C	00078	
SRBDATO	00004	000010	00079	
SRBEND	00004	000018	00082	
SRBRESET	00004	000014	00080	
SRCONT	00002	000002	00067	
SRINTV	00002	000008	00072	
SRMR	00002	000004	00068	
SRMW	00002	000006	00070	
SRPRTY	00002	00000A	00075	
SRSRB	00004	00001C	00084	
STATREG	00001	000000	00059	

NO STATEMENTS FLAGGED IN THIS ASSEMBLY
STATISTICS SOURCE RECORDS (SYSIN) = 71 SOURCE RECORDS (SYSLIB) = 110
OPTIONS IN EFFECT LIST, DECK, NOLOAD, NORENT, XREF, NOTEST, ALGN, OS, NOTERM, LINECNT = 55
235 PRINTED LINES

APPENDIX VII

SAMPLE ASSEMBLY AND SIMULATION

PDP11ASM - STORAGE ALLOCATION PAGE 1

STORAGE REQUESTED	256 BYTES
STORAGE ALLOCATED	53248 BYTES
SYMBOL TABLE	1472 ENTRIES
XREF TABLE	4416 ENTRIES

```
LOC     OBJECT CODE     STMT   SOURCE STATEMENT

000000                     1 R0=     %0
000001                     2 R1=     %1
000002                     3 R2=     %2
000003                     4 R3=     %3
000004                     5 R4=     %4
000005                     6 R5=     %5
000006                     7 R6=     %6
000007                     8 R7=     %7
```

```
 LOC       OBJECT CODE      STMT   SOURCE STATEMENT

001000                       10 .        =     1000              PROGRAM ORIGIN
001000 012706 001000         11 START    MOV   =.,R6             SET STACK POINTER
001004 012767 000101 176546  12          MOV   =101,177560       ENABLE AND START READER
001012 012767 000101 176530  13          MOV   =101,177550       ENABLE AND START READER
001020 012767 000101 176512  14          MOV   =101,177540       ENABLE AND START READER
001026 000001                15          WAIT                    MAIN PROGRAM
001030 000776                16          BR    .-2

001032 010046                18 INT      MOV   R0,-(R6)          SAVE R0 ON STACK
001034 012500                19          MOV   (R5)+,R0          TKS ADDRESS
001036 103416                20          BCS   TTYP              BRANCH FOR PRINTER
001040 005725                21          TST   (R5)+             OVER OUTPUT POINTER
001042 116035 000002         22          MOVB  2(R0),((R5)+)     PUT BYTE IN BUFFER IN CORE
001046 005245                23          INC   -(R5)             INCREMENT BUFFER POINTER
001050 012760 000100 000004  24          MOV   =100,4(R0)        ENABLE TELEPRINTER
001056 022515                25          CMP   (R5)+,(R5)        SEE IF END YET
001060 001402                26          BEQ   NOENB             NO ENABLE IF SO
001062 012710 000101         27          MOV   =101,(R0)         ENABLE AGAIN
001066 012600                28 NOENB    MOV   (R6)+,R0          UNSAVE R0
001070 012605                29          MOV   (R6)+,R5          RESTORE R5
001072 000002                30          RTI

001074 062700 000004         32 TTYP     ADD   =4,R0             TPS ADDR
001100 022515                33          CMP   (R5)+,(R5)        SEE IF CAUGHT UP TO INPUT
001102 001404                34          BEQ   NOMORE            NO MORE IF SO
001104 012720 000101         35          MOV   =101,(R0)+        ENABLE AGAIN
001110 115510                36          MOVB  (-(R5)),(R0)      CHARACTER FROM CORE TO TPB
001112 005215                37          INC   (R5)            INCREMENT POINTER
001114 012600                38 NOMORE   MOV   (R6)+,R0          RESTORE R0
001116 012605                39          MOV   (R6)+,R5          RESTORE R5
001120 000002                40          RTI
```

```
 LOC      OBJECT CODE      STMT   SOURCE STATEMENT

001122 004567 177704          42 TTY1     JSR   R5,INT              JSR TO INTERRUPT PGM
001126 177560 001166 001166   43          .WORD 177560,B1,B1,B2     TKS, BUFFER POINTERS
       001266
001136 004567 177670          44 TTY2     JSR   R5,INT              JSR TO INTERRUPT PGM
001142 177550 001266 001266   45          .WORD 177550,B2,B2,B3     TKS, BUFFER POINTERS
       001366
001152 004567 177654          46 TTY3     JSR   R5,INT              JSR TO INTERRUPT PGM
001156 177540 001366 001366   47          .WORD 177540,B3,B3,B4     TKS, BUFFER POINTERS
       001466
001166                        48 B1       DS    40                  BUFFER AREA
001266                        49 B2       DS    40                  BUFFER AREA
001366                        50 B3       DS    40                  BUFFER AREA
                              51 B4

000060                        53 .        =     60                  INTERRUPT VECTOR ORIGIN
000060 001122 000240 001122   54          .WORD TTY1,240,TTY1,241   KEYBOARD, PRINTER
       000241
000070 001136 000200 001136   55          .WORD TTY2,200,TTY2,201   KEYBOARD, PRINTER
       000201
000100 001152 000200 001152   56          .WORD TTY3,200,TTY3,201   KEYBOARD, PRINTER
       000201
000110 000114 000300          57          .WORD CLOCK,300           CLOCK VECTOR
000114 000002                 58 CLOCK    RTI

001000                        60          END   START
```

CROSS-REFERENCES

SYMBOL	VALUE		DEFN	REFERENCES														
B1	001166		0048	043	043													
B2	001266		0049	043	045	045												
B3	001366		0050	045	047	047												
B4	001466		0051	047														
CLOCK	000114		0058	057														
INT	001032		0018	042	044	046												
NOENB	001066		0028	026														
NOMORE	001114		0038	034														
R0	000000	R	0001	018	019	022	024	027	028	032	035	036	038					
R1	000001	R	0002															
R2	000002	R	0003															
R3	000003	R	0004															
R4	000004	R	0005															
R5	000005	R	0006	019	021	022	023	025	025	029	033	033	036	037	039	042	044	046
R6	000006	R	0007	011	018	028	029	038	039									
R7	000007	R	0008															
START	001000		0011	060														
TTYP	001074		0032	020														
TTY1	001122		0042	054	054													
TTY2	001136		0044	055	055													
TTY3	001152		0046	056	056													

NO STATEMENTS FLAGGED IN THIS ASSEMBLY

PDP11 DEVICE CONFIGURATION:

```
KEYBD1    TTYK  DATA=(5100,160)
PRINTER1  TTYP  DATA=(5100,160)
KEYBD2    TTYK  DATA=8000,ADR=(177550,177552),INTV=70
PRINTER2  TTYP  DATA=7500,ADR=(177554,177556),INTV=74
KEYBD3    TTYK  DATA=5000,ADR=(177540,177542),INTV=100
PRINTER3  TTYP  DATA=6000,ADR=(177544,177546),INTV=104
TLIMIT    TLIMIT DATA=50000
```

PDP11 DEVICE CONFIGURATION:

DDNAME	DEVNAME	ADR	CONT	INTV	MR	MW	DATA
KEYBD1	TTYK	177560	000100	004060	004300	000101	5100,160,0,0
		177562	000000	000000	000377	000000	
PRINTER1	TTYP	177564	000200	000064	000304	000104	5100,160,0,0
		177566	000000	000000	000000	000377	
KEYBD2	TTYK	177550	000030	000070	004300	000101	8000,128,0,0
		177552	000000	000000	000377	000000	
PRINTER2	TTYP	177554	000000	000074	000304	000104	7500,128,0,0
		177556	000000	000000	000003	000377	
KEYBD3	TTYK	177540	000000	000100	004300	000101	5000,128,0,0
		177542	000000	000000	000377	000000	
PRINTER3	TTYP	177544	000200	000104	000304	000104	6000,128,0,0
		177546	000000	000000	000000	000377	
TLIMIT	TLIMIT	000070	000000	000000	000000	000000	50000,0,0,0

BEFORE TIME	INSTRUCTION PS	PC	INSTRN	INSTRUCTION MNEMONIC	SOURCE FIELD INDEX WORD	SOURCE FIELD REG BEFORE	SOURCE FIELD CORE ADDRESS	SOURCE FIELD CORE DATA	DESTINATION FIELD INDEX WORD	DESTINATION FIELD REG BEFORE	DESTINATION FIELD CORE ADDRESS	DESTINATION FIELD CORE DATA	RESULT	COND CODE NZVC
0.0	000000	001000	012706	MOV (7)+,6		001002	001002	001000		000000	R6	000000	001000	0000
3.7	000000	001004	012767	MOV (7)+,X(7)		001006	001006	000101	176546	001012	177560	000000	000101	0000
9.8	000000	001012	012767	MOV (7)+,X(7)		001014	001014	000101	176530	001020	177550	000000	000101	0000
15.9	000000	001020	012767	MOV (7)+,X(7)		001022	001022	000101	176512	001026	177540	000000	000101	0000
22.0	000000	001026	000001	WAIT										0000
519.8	000000	001030		KEYBD1 INTERRUPT VECTR=		000060	SP=	000774		PS=	000240	PC=	001122	0000
527.9	000240	001122	004567	JSR 5,X(7)		000000			177704	001126	001032	010046		0000
534.5	000240	001032	010046	MOV 0,-(6)		000000	R0	000000		000772	000770	000000	*000000	0100
539.2	000244	001034	012500	MOV (5)+,0		001126	001126	177560		000000	R0	000000	177560	1000
542.9	000250	001036	103416	BCS .+36	BRANCH	NOT TAKEN					001074			1000
544.3	000250	001040	005725	TST (5)+						001130	001130	001166	*001166	0000
547.8	000240	001042	116035	MOVB X(0),((5)+)	000002	177560	177562	1 261		001132	001166	000	1 261	1000
556.1	000250	001046	005245	INC -(5)						001134	001132	001166	001167	0000
559.6	000240	001050	012760	MOV (7)+,X(0)		001052	001052	000100	000004	177560	177564	000200	000100	0000
565.7	000240	001056	022515	CMP (5)+,(5)		001132	001132	001167		001134	001134	001166	000001	0000
570.4	000240	001060	001402	BEQ .+6	BRANCH	NOT TAKEN					001066			0000
571.8	000240	001062	012710	MOV (7)+,(0)		001064	001064	000101		177560	177560	000100	000101	0000
576.7	000240	001066	012600	MOV (6)+,0		000770	000770	000000		177560	R0	177560	000000	0100
580.4	000244	001070	012605	MOV (6)+,5		000772	000772	000000		001134	R5	001134	000000	0100
584.1	000244	001072	000002	RTI			SP=	001000		PS=	000000	PC=	001030	0000
588.9	000000	001030		PRINTER1 INTERRUPT VECTR=		000064	SP=	000774		PS=	000241	PC=	001122	0001
597.0	000241	001122	004567	JSR 5,X(7)		000000			177704	001126	001032	010046		0001
603.6	000241	001032	010046	MOV 0,-(6)		000000	R0	000000		000772	000770	000000	*000000	0101
608.3	000245	001034	012500	MOV (5)+,0		001126	001126	177560		000000	R0	000000	177560	1001
612.0	000251	001036	103416	BCS .+36	BRANCH	TAKEN					001074			1001
614.5	000251	001074	062700	ADD (7)+,0		001076	001076	000004		177560	R0	177560	177564	1000
618.2	000250	001100	022515	CMP (5)+,(5)		001130	001130	001166		001132	001132	001167	177777	1001
622.9	000251	001102	001404	BEQ .+12	BRANCH	NOT TAKEN					001114			1001
624.3	000251	001104	012720	MOV (7)+,(0)+		001106	001106	000101		177564	177564	000300	000101	0001
629.2	000241	001110	115510	MOVB (-(5)),(0)		001132	001166	1 261		177566	177566	000	1 261	1001
635.3	000251	001112	005215	INC (5)						001130	001130	001166	001167	0001
638.8	000241	001114	012600	MOV (6)+,0		000770	000770	000000		177566	R0	177566	000000	0101
642.5	000245	001116	012605	MOV (6)+,5		000772	000772	000000		001130	R5	001130	000000	0101
646.2	000245	001120	000002	RTI			SP=	001000		PS=	000000	PC=	001030	0000
651.0	000000	001030		KEYBD3 INTERRUPT VECTR=		000100	SP=	000774		PS=	000200	PC=	001152	0000
659.1	000200	001152	004567	JSR 5,X(7)		000000			177654	001156	001032	010046		0000
665.7	000200	001032	010046	MOV 0,-(6)		000000	R0	000000		000772	000770	000000	*000000	0100
670.4	000204	001034	012500	MOV (5)+,0		001156	001156	177540		000000	R0	000000	177540	1000
674.1	000210	001036	103416	BCS .+36	BRANCH	NOT TAKEN					001074			1000
675.5	000210	001040	005725	TST (5)+						001160	001160	001166	*001166	0000
679.0	000200	001042	116035	MOVB X(0),((5)+)	000002	177540	177542	* 252		001162	001166	1 261	* 252	1000
687.3	000210	001046	005245	INC -(5)						001164	001162	001166	001167	0000
690.8	000200	001050	012760	MOV (7)+,X(0)		001052	001052	000100	000004	177540	177544	000200	000100	0000
696.9	000200	001056	022515	CMP (5)+,(5)		001162	001162	001167		001164	001164	001166	000001	0000
701.6	000200	001060	001402	BEQ .+6	BRANCH	NOT TAKEN					001066			0000
703.0	000200	001062	012710	MOV (7)+,(0)		001064	001064	000101		177540	177540	000100	000101	0000
707.9	000200	001066	012600	MOV (6)+,0		000770	000770	000000		177540	R0	177540	000000	0100
711.6	000204	001070	012605	MOV (6)+,5		000772	000772	000000		001164	R5	001164	000000	0100
715.3	000204	001072	000002	RTI			SP=	001000		PS=	000000	PC=	001030	0000
720.1	000000	001030		PRINTER3 INTERRUPT VECTR=		000104	SP=	000774		PS=	000201	PC=	001152	0001
728.2	000201	001152	004567	JSR 5,X(7)		000000			177654	001156	001032	010046		0001
734.8	000201	001032	010046	MOV 0,-(6)		000000	R0	000000		000772	000770	000000	*000000	0101
739.5	000205	001034	012500	MOV (5)+,0		001156	001156	177540		000000	R0	000000	177540	1001
743.2	000211	001036	103416	BCS .+36	BRANCH	TAKEN					001074			1001
745.7	000211	001074	062700	ADD (7)+,0		001076	001076	000004		177540	R0	177540	177544	1000

BEFORE TIME	INSTRUCTION PS	INSTRUCTION PC	INSTRN	INSTRUCTION MNEMONIC	SOURCE FIELD INDEX WORD	SOURCE FIELD REG BEFORE	SOURCE FIELD CORE ADDRESS	SOURCE FIELD CORE DATA	DESTINATION FIELD INDEX WORD	DESTINATION FIELD REG BEFORE	DESTINATION FIELD CORE ADDRESS	DESTINATION FIELD CORE DATA	RESULT	COND CODE NZVC
749.4	000210	001100	022515	CMP (5)+,(5)		001160	001160	001166		001162	001162	001167	177777	1001
754.1	000211	001102	001404	BEQ .+12	BRANCH NOT TAKEN						001114			1001
755.5	000211	001104	012720	MOV (7)+,(0)+		001106	001106	000101		177544	177544	000300	000101	0001
760.4	000201	001110	115510	MOVB (-(5)),(0)		001162	001166	* 252		177546	177546	000	* 252	1001
766.5	000211	001112	005215	INC (5)						001160	001160	001166	001167	0001
770.0	000201	001114	012600	MOV (6)+,0		000770	000770	000000		177546	R0	177546	000000	0101
773.7	000205	001116	012605	MOV (6)+,5		000772	000772	000000		001160	R5	001160	000000	0101
777.4	000205	001120	000002	RTI				SP= 001000		PS= 000000	PC=	001030		0000
782.2	000000	001030	000776	BR .-2	BRANCH TAKEN						001026			0000
784.7	000000	001026	000001	WAIT										0000
815.9	000000	001030		KEYBD2 INTERRUPT	VECTR=	000070		SP= 000774		PS= 000200	PC=	001136		0000
824.0	000200	001136	004567	JSR 5,X(7)		000000			177670	001142	001032	010046		0000
830.6	000200	001032	010046	MOV 0,-(6)		000000	R0	000000		000772	000770	000000	*000000	0100
835.3	000204	001034	012500	MOV (5)+,0		001142	001142	177550		000000	R0	000000	177550	1000
839.0	000210	001036	103416	BCS .+36	BRANCH NOT TAKEN						001074			1000
840.4	000210	001040	005725	TST (5)+						001144	001144	001166	*001166	0000
843.9	000200	001042	116035	MOVB X(0),((5)+)	000002	177550	177552	ƶ 332		001146	001166	* 252	ƶ 332	1000
852.2	000210	001046	005245	INC -(5)						001150	001146	001166	001167	0000
855.7	000202	001050	012760	MOV (7)+,X(0)		001052	001052	000100	000004	177550	177554	000200	000100	0000
861.8	000207	001056	022515	CMP (5)+,(5)		001146	001146	001167		001150	001150	001166	000001	0000
866.5	000200	001060	001402	BEQ .+6	BRANCH NOT TAKEN						001066			0000
867.9	000200	001062	012710	MOV (7)+,(0)		001064	001064	000101		177550	177550	000100	000101	0000
872.8	000200	001066	012600	MOV (6)+,0		000770	000770	000000		177550	R0	177550	000000	0100
876.5	000204	001070	012605	MOV (6)+,5		000772	000772	000000		001150	R5	001150	000000	0100
880.2	000204	001072	000002	RTI				SP= 001000		PS= 000000	PC=	001030		0000
885.0	000000	001030		PRINTER2 INTERRUPT	VECTR=	000074		SP= 000774		PS= 000201	PC=	001136		0001
893.1	000201	001136	004567	JSR 5,X(7)		000000			177670	001142	001032	010046		0001
899.7	000231	001032	010046	MOV 0,-(6)		000000	R0	000000		000772	000770	000000	*000000	0101
904.4	000205	001034	012500	MOV (5)+,0		001142	001142	177550		000000	R0	000000	177550	1001
908.1	000211	001036	103416	BCS .+36	BRANCH TAKEN						001074			1001
910.6	000211	001074	062700	ADD (7)+,0		001076	001076	000004		177550	R0	177550	177554	1000
914.3	000217	001100	022515	CMP (5)+,(5)		001144	001144	001166		001146	001146	001167	177777	1001
919.0	000211	001102	001404	BEQ .+12	BRANCH NOT TAKEN						001114			1001
920.4	000211	001104	012720	MOV (7)+,(0)+		001106	001106	000101		177554	177554	000300	000101	0001
925.3	000201	001110	115510	MOVB (-(5)),(0)		001146	001166	ƶ 332		177556	177556	000	ƶ 332	1001
931.4	000211	001112	005215	INC (5)						001144	001144	001166	001167	0001
934.9	000201	001114	012600	MOV (6)+,0		000770	000770	000000		177556	R0	177556	000000	0101
938.6	000205	001116	012605	MOV (6)+,5		000772	000772	000000		001144	R5	001144	000000	0101
942.3	000205	001120	000002	RTI				SP= 001000		PS= 000000	PC=	001030		0000
947.1	000000	001030	000776	BR .-2	BRANCH TAKEN						001026			0000
949.6	000000	001026	000001	WAIT										0000
1086.7	000020	001030		KEYBD1 INTERRUPT	VECTR=	000060		SP= 000774		PS= 000240	PC=	001122		0000
1094.8	000240	001122	004567	JSR 5,X(7)		000000			177704	001126	001032	010046		0000
1101.4	000243	001032	010046	MOV 0,-(6)		000000	R0	000000		000772	000770	000000	*000000	0100
1106.1	000244	001034	012500	MOV (5)+,0		001126	001126	177560		000000	R0	000000	177560	1000
1109.8	000250	001036	103416	BCS .+36	BRANCH NOT TAKEN						001074			1000
1111.2	000250	001040	005725	TST (5)+						001130	001130	001167	*001167	0000
1114.7	000240	001042	116035	MOVB X(0),((5)+)	000002	177560	177562	1 261		001132	001167	000	1 261	1000
1123.6	000250	001046	005245	INC -(5)						001134	001132	001167	001170	0000
1127.1	000240	001050	012760	MOV (7)+,X(0)		001052	001052	000100	000004	177560	177564	000100	*000100	0000
1133.2	000240	001056	022515	CMP (5)+,(5)		001132	001132	001170		001134	001134	001166	000002	0000
1137.9	000240	001060	001402	BEQ .+6	BRANCH NOT TAKEN						001066			0000
1139.3	000240	001062	012710	MOV (7)+,(0)		001064	001064	000101		177560	177560	000100	000101	0000
1144.2	000240	001066	012600	MOV (6)+,0		000770	000770	000000		177560	R0	177560	000000	0100
1147.9	000244	001070	012605	MOV (6)+,5		000772	000772	000000		001134	R5	001134	000000	0100

APPENDIX VIII

MAINDEC CPU DIAGNOSTICS PROGRAMS

MAINDEC-11-D0AA-PB TEST 1 BRANCH 3/17/70

MAINDEC-11-D0BA-PB TEST 2 CON BRANCH 3/17/70

MAINDEC-11-D0CA-PB TEST 3 UNARY 3/17/70

MAINDEC-11-D0DA-PB TEST 4 UNARY+BINARY 3/17/70

MAINDEC-11-D0EA-PB TEST 5 ROTATE SHIFT 3/17/70

MAINDEC-11-D0FA-PB TEST 6 COMPARE 3/17/70

MAINDEC-11-D0GA-PB TEST 7 COMPARE NOT 3/17/70

MAINDEC-11-D0HA-PB TEST 8 MOVE 3/17/70

MAINDEC-11-D0IA-PB TEST 9 BIS,BIC+BIT 3/17/70

MAINDEC-11-D0JA-PB TEST 10 ADD 3/17/70

MAINDEC-11-D0KA-PB TEST 11 SUBTRACT 3/17/70

MAINDEC-11-D0LA-PB TEST 12 JMP 3/17/70

MAINDEC-11-D0MA-PB TEST 13 JSR,RTS,RTI 2/16/70

MAINDEC-11-D0NA-PB TEST 14 TRAP TEST 4/6/70

MAINDEC-11-D0OA-PB TEST 15 COMBINED INSTRUCTION TEST 3/20/70