PSEUDO*, A MACRO-BASED HIGH LEVEL LANGUAGE FOR THE PDP-11, (U)

APR 74  B DAVY

UNCLASSIFIED  RSRE-MEMO-2875  DRIC-8R-64523  NL
ROYAL SIGNALS & RADAR ESTABLISHMENT

PSEUDOCIA MACRO-BASED HIGH LEVEL
LANGUAGE FOR THE PDP-11

Author: J. Davy

APR 1974

PROCUREMENT EXECUTIVE,
MINISTRY OF DEFENCE,
RSRE MALVERN,
WORCS.

DTIC ELECTE
NOV 9 1981
SUMMARY

PSEUDO is a pseudo-high-level language, developed for the PDP-11 computer. The language is extremely efficient and particularly suited to real-time programming applications.

CONTENTS

1 Introduction
2 Background
   2.1 The Processor
   2.2 The Assembler
   2.3 MACRO-II Syntax
   2.4 Addressing Modes
3 PSEUDO Statements
   3.1 Symbolic Assignments
   3.2 Data Allocation
   3.3 Data Presetting
   3.4 List Processing
   3.5 Buffer Processing
   3.6 Conditionals
   3.7 Loops
   3.8 Stack Operations
   3.9 Procedure Calls
   3.10 Arithmetic Operations
4 Operation
5 Comments

Accession For

DTIC TAB
Unannounced
Justification

By Per Jr. on File
Distribution/
Availability Codes

Dist

DTIC ELECTE
NOV 9 1981
Appendix 1 - Macro generation: examples

Appendix 2 - Source Text Example

Appendix 3 - Normal Assembly Listing

Appendix 4 - Macro-expanded Assembly Listing

Appendix 5 - PSEUDO Macro Definitions

1 Introduction

The PSEUDO language has been developed for use with the PDP-11 disc operating system, for a specific real-time control application. High level source text statements are interpreted as macrocalls, which are expanded by the standard DOS MACRO-11 assembler.

PSEUDO provides the features normally associated with high level languages, viz, good source text readability, self-documentation, and standardisation of certain data processing techniques. Additionally, it allows facilities which are not normally associated with high level languages, but which were considered essential to the current application. These are:

i Uninhibited use of word or byte operations.

ii Unlimited use of all PDP-11 addressing modes.

iii User control of register assignment and usage, including stack operations.

iv High efficiency, in terms both of run-time and storage, with user choice of one or other type of efficiency in conflicting situations.

No attempt has been made in PSEUDO to replace easily understood assembly language statements simply to emulate existing high level statements. However, the need to write obscure assembly language has been eliminated, particularly in code associated with loops and conditionals.

2 Background

Since PSEUDO is designed specifically around the PDP-11 and its assembler, some features of these must be described briefly before proceeding to a description of PSEUDO itself.

2.1 The Processor

The PDP-11 is a 16 bit machine, with almost equal facility of byte or word operations. Peripheral devices are allocated specific addresses, allowing memory reference instructions to operate directly on data held in peripheral registers. The machine has eight program accessible registers, R0-R7. Two of these are used as program counter (R7) and stack pointer (R6) respectively, leaving six which can be used generally as accumulators, pointers or index registers. The stack pointer points to the last input of a last in - first out stack held in core. Linkage parameters are moved automatically to and from this stack (by hardware), to handle interrupts, sub-routine calls and traps (software interrupts).
2.2 The Assembler

The DOS assembler, MACRO-11R, is a two pass assembler (the second pass handled automatically by DOS) producing relocatable object code modules for input to a linker. In conjunction with the machine architecture, the assembler allows easy writing of position independent and/or re-entrant code. MACRO-11R includes a macro processor. A comprehensive set of assembly directives and macro-expansion directives are provided; these are used extensively by PSEUDO as discussed briefly in Appendix 1.

2.3 MACRO-11 Syntax

PSEUDO incorporates the syntax of the MACRO-11 assembly language; i.e., any legal MACRO-11 statement is a legal PSEUDO statement. The relevant syntax rules refer to expressions and register expressions and are as follows:

(Backus notation has been dropped in favour of typographical layout. Each syntax rule has a class name on its left-hand side. Alternative expansions for the class are on the right-hand side, each on a new line.)

Expression = Term

   Unaryoperator Term

   Expression Binaryoperator Term

Term = Constant

   Symbol

   Asciiconversion

   <Expression>

Constant = Octalnumber

   Decimal number

Octalnumber = Sequence of octal digits

Decimalnumber = Sequence of decimal digits terminated by period

Symbol = Sequence of letters or digits starting with a letter

Asciiconversion = 'Asciicharacter

   "Asciicharacter Asciicharacter

Binaryoperator = ±

   * (multiply)

   / (divide)

   & (logical AND)

   ! (logical OR)
Unary operator = +
-
Expressions are evaluated from left to right, with no operator hierarchy except that terms in paired angle brackets are evaluated first.

Symbols may be defined as labels, to refer to specific locations or data, or may be created and given values by symbolic assignment statements of the form:

Symbol = Expression

e.g.
ON = 1
NOTOFF = ON
TTYREGISTER = 777562

Registers may be named symbolically by the register assignment statement:

Symbol = % Octal digit

A register expression is any expression containing a symbol previously assigned to a register, e.g.-

R1 = %1 ; initial assignment of symbol R1 to register 1
POINTER = R1 ; assignment of name POINTER to register 1.
; R1 is a single term register expression.

2.4 Addressing Modes

Memory reference statements consist of an operand (instruction mnemonic) followed by one or two operand address specifications. These statements assemble to one, two or three words, depending on the number and modes of the address specifications. Address specification formats, A, are expressed below in terms of E, R and ER, where E is any expression, R is any register expression and ER is any register expression or any expression having a value in the range 0-7.

<table>
<thead>
<tr>
<th>MODE</th>
<th>FORMAT OF A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>R</td>
</tr>
<tr>
<td>Deferred register</td>
<td>(ER)</td>
</tr>
<tr>
<td>Auto-increment</td>
<td>(ER)+</td>
</tr>
<tr>
<td>Auto-decrement</td>
<td>-(ER)</td>
</tr>
</tbody>
</table>

The register defined by R contains the operand.
The register defined by ER contains the address of the operand.
The contents of the register defined by ER are incremented after being used as the address of the operand.
The contents of the register defined by ER are decremented before being used as the address of the operand.
<table>
<thead>
<tr>
<th>MODE</th>
<th>FORMAT OF A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deferred auto-</td>
<td>@(ER) The register defined by ER contains the</td>
</tr>
<tr>
<td>increment</td>
<td>pointer to the address of the operand. The pointer is incremented after use.</td>
</tr>
<tr>
<td>Deferred auto-decrement</td>
<td>@-(ER) The contents of the register defined by ER are decremented before</td>
</tr>
<tr>
<td></td>
<td>being used as a pointer to the address of the operand.</td>
</tr>
<tr>
<td>Index</td>
<td>E(ER) The value of E plus the contents of the register defined by ER gives</td>
</tr>
<tr>
<td></td>
<td>the address of the operand.</td>
</tr>
<tr>
<td>Deferred index</td>
<td>@E(ER) The value of E plus the contents of the register defined by ER gives</td>
</tr>
<tr>
<td></td>
<td>the address of the operand.</td>
</tr>
<tr>
<td>Immediate</td>
<td>#E The value of E is the operand.</td>
</tr>
<tr>
<td>Absolute</td>
<td>@#E The value of E is the address of the operand. The address is assembled</td>
</tr>
<tr>
<td></td>
<td>in absolute form.</td>
</tr>
<tr>
<td>Relative</td>
<td>E The value of E is the address of the operand. The address is assembled in</td>
</tr>
<tr>
<td></td>
<td>relative form.</td>
</tr>
<tr>
<td>Deferred relative</td>
<td>@E The value of E is the address of the address of the operand.</td>
</tr>
</tbody>
</table>

*By one for byte instructions, two for word instructions.

The first six modes tabulated do not increase the assembled word length of the instruction. All other modes add one word.

Eg: assuming PARTSUM, OFFSET and TOTAL have been assigned to registers,

ADD (PARTSUM)+, TOTAL ; assembles as one word. The word pointed to by PARTSUM is added to TOTAL. PARTSUM contents are then incremented to point to next word location.

ADD WORKSPACE (OFFSET), TOTAL ; assembles as two words. OFFSET'TH item of WORKSPACE is added to TOTAL.

MOVB #'A, @ # TTYREG ; assembles as three words. Outputs ASCII rep of A to teletype.

3 PSEUDO STATEMENTS

PSEUDO statements consist of one or more macro calls, each consisting of a key word (macro defining symbol) followed by macro parameter words. Legal word separators are space(s) tab(s) or comma.

The following conventions are used:-
i. are any expressions, as defined in 2.3.

ii. are any operand address specification formats, as defined in 2.4.

iii. are any legal symbols, as defined in 2.3.

iv. Square brackets indicate a choice between two or more parameters.

v. Parameters represented by a character string in round brackets may be replaced by any character string not including < > ( ); or any separator.

vi. Parameters represented by numerals in round brackets may be omitted.

Layout characters may be used freely, provided that parameter words remain on the same line as their associated macro defining name. Angle brackets, where shown, are mandatory (these allow character strings including macro parameter delimiters to be passed as a single actual parameter to the macro processor).

3.1 Symbolic Assignments

All symbolic assignments and global declarations required by the language itself are made by calling the macro PSEUDO. These include the commonly used symbols:

- \( R_0 = Z_0 \)
- \( R_1 = Z_1 \)
- \( R_7 = Z_7 \)
- \( SP = R_6 \) (stack pointer)
- \( PC = R_7 \) (program counter)
- \( SR = 177776 \) (status register)
- \( SWR = 177570 \) (switch register)
- \( CR = 15 \) (Ascii, carriage return)
- \( LF = 12 \) (Ascii, line feed)
- \( SPACE = 40 \) (Ascii, space)

3.2 Data Allocation

Data storage allocations are made by word, byte, list or buffer declarations using the macro CREATE:

\[
\text{CREATE [WORDS] } <S_1, S_2, ..., S_n> \quad (1) \quad (2) \quad (3) \quad (4) \quad (5) \quad (6) \quad (7) \quad (8) \quad (9) \quad (10)
\]

\[
\text{[BYTES]}
\]

allocates words or bytes named \( S_1, S_2, ..., S_n \), initialised to zero.
Eg:
CREATE WORDS <WORD1,WORD2>
CREATE BYTES <FLAG1,FLAG2,DONE> FOR INPUT/OUTPUT FLAGS
CREATE LIST S E1 [WORDS] E2 (1) (2) (3) (4) (5) (6) (7)
allocates a block structured list named S, E1 words or bytes long, E2 words or bytes per block, headed by

S-10: Length of data area in bytes
S-8: Input pointer location, preset to S
S-6: Output " " " "
S-4: Size of block in bytes
S-2: Address of last block
S : Data area

eg, (identical declarations)
CREATE LIST TYRES 20.WORDS
CREATE LIST TYRES, 24 WORDS, 5 PER BLOCK
CARS = 4 WHEELS = 5
CREATE LIST TYRES, CARS*WHEELS WORDS, WHEELS PER CAR = 5, INCLUDING SPARE.
The examples above illustrate how documentation can be built in to statements, using the optional macro call parameters.
CREATE BUFFER S E (1) (2) (3) (4) (5) (6) (7) (8) (9)
allocates an input/output buffer named S, with a data area of E bytes, with header:-

S : Size of data area in bytes (E)
S+2 : Location for status/mode bytes. Set to 0100000 (Done, no errors)
S+4 : Location for message character count. Set to E
S+6 : Start of data area.

eg, (identical samples)
CREATE BUFFER BUFF1 64. CHARACTERS
TTYCHARS = 100
CREATE BUFFER BUFF1, TTYCHARS LONG, FOR TELETYPewriter INPUT.
3.3 Data Presetting

Items of data may preset, using the macro WITH.

\[
\text{WITH (DATA) } \langle E_1, E_2, \ldots, E_n \rangle
\]

associates values \( E_1, E_2, \ldots, E_n \) with corresponding words or bytes in a preceding CREATE statement. WITH statements may be made consecutively. If the preceding CREATE statement created a list or buffer, data insertion starts at the first word or byte of the data area.

eg:

CREATE BYTES \langle ONE, TWO, THREE, TEN, SIXTEEN, FIFTY, TWO56 \rangle

WITH DATA \langle 1, 2, 3, 10, 20 \rangle

WITH DATA \langle 50., 400 \rangle

.EVEN

CREATE WORDS TWO56ADDRESS, ASCII, XY, TWOADDRESS, LEFTATZERO

WITH DATA \langle TWO56, 'XY,ONE+1 \rangle

CREATE BUFFER TTYOUTPUT, 16. CHAR

WITH DATA \langle 'O,'U,'T,'P,'U,'T,SPACE,'M,'E,'S,'A,'G,'E,CR,LF \rangle

Note that the .EVEN directive, necessary to allow word assembly after creating an odd number of bytes, must come after the byte data WITH statements.

3.4 List Processing

POINT A (AT) \[ \text{FIRST} \] BLOCK (OF) E

LAST

IP

OF

sets the location defined by A to the address of the first block, last block, block pointed to by the list header input pointer, or block pointed to by the list header output pointer, of the list defined by E.

Eg

POINT POINTER AT IP BLOCK OF LIST1 ; POINTER may be a register or

; word location.

POINT @(POINTER) TO LAST BLOCK IN LIST1 ; POINTER must be a register,

; which contains the address of

; the address of the word which
gets pointed to the last block.

POINT A (PAST) END (OF) E (1)

sets the location defined by A to the address of the first byte following the data area of the list defined by E.

Eg,

POINT @WORD1 PAST END OF LIST1 DATA ; WORD1 contained the address of the pointer.

STEP [IP] (POINTER) [ON] (THROUGH) E
            [OP] [BACK]
            [A]

moves the header input pointer, header output pointer, or the pointer defined by A, on or back one block through the list defined by E.

Eg,

STEP IP VALUE ON THRU LIST1
STEP POINTERS(INDEX) PNTR BACK THROUGH LIST1
STEP R1 POINTER ON PAST LIST1

CYCLE [IP] (POINTER) [ON] (THROUGH) E
            [OP] [BACK]
            [A]

is the same as STEP, except that the pointer is reset to the first block if cycled on from the last block, and vice-versa.

SET E [IP] (TO) A
      [OP]

transfers the contents of the location defined by A to the header input pointer or output pointer location of the list defined by E.

Eg,

SET INPUTLIST IP TO NEXTINPUT
SET LIST1 OP TO # LIST1+ <4*BLOCKSIZE> ; Point OP to fourth block.
                                              ; BLOCKSIZE
                                              ; value set by previous assignment.
GET E BLOCKSIZE IN A

sets the location defined by A to the block size (in bytes) of the list defined by E.

Eg, to access block N of LIST 1:-

GET LIST1 BLOCKSIZE IN BLOCKN ; BLOCKN assigned to a register
MUL BLOCKN BY #N, ANSWER IN BLOCKN ; See below (3.10)
CLR LIST1(BLOCKN) ; Clear first word of block N.

3.5 Buffer Processing

Buffer headers interface to an executive program handling input/output to non-file devices on a character per interrupt basis. Briefly, the character count word indicates the number of characters for input or output from the buffer, status byte indicates transmission done or error conditions, and mode byte indicates the type of message (binary or ASCII, formatted or unformatted).

POINT A (TO) E DATA

points the location defined by A to the start of the data area of the buffer defined by E.

Eg,

POINT CHARPOINTER AT TTYBUFFER DATA

POINT A (PAST) E DATA END

points the location defined by A to the first byte following the last message character in the data area of the buffer defined by E.

Eg,

POINT LABEL+2 PAST BUFFER DATA END ; pointer is held in word following LABEL.

GET E [STATUS] (IN) A

MODE

COUNT

allows transfer of buffer header parameters to user locations defined by A.

Eg,

GET BUFFER STATUS IN STATBYTE ; Since status is a byte, STATBYTE

; may be a byte.
GET TTYBUFF COUNT IN TCOUNT ; Count is a word, hence TCOUNT should be a word (or register).

SET E COUNT (TO) A

sets the buffer header count defined by E to the value held at the location defined by A.

Eg,

SET TTYOUTPUT COUNT TO 64.

SET BUFF1 COUNT FROM CHARCOUNT

READY E (FOR) [ASCII ] (1) (2) (3) (4)
           [FASCII]
           BIN
           [FBIN]

sets the mode byte of the buffer defined by E, for input or output in the specified mode.

Eg,

READY TAPEBUFFER FOR FBIN INPUT FROM H.S. READER

OUTPUT E (TO) [LSP] [(NOTIFY) A]
         [HSP] [VOID]
         [TTY]

initiates interrupt driven output from the buffer specified by E to the specified device (teletype punch, high-speed punch or teletype). If the NOTIFY A clause is included the input/output executive will make a call (at interrupt priority level) to the procedure identified by A when buffer transmission is done, or when an error is detected.

Eg,

OUTPUT TTYMESSAGE TO TTY

OUTPUT BUFFER TO LSP, TELL NEXTBUFFERPROCESS

INPUT [LSR] (TO) E [(NOTIFY) A]
       [HSR] [VOID]
       [KBD]

similarly initiates input from low-speed reader, high-speed reader or teletype keyboard.
Eg,

INPUT KBD TO KBOARD

INPUT HSR TO TAPEBUFFER, NOTIFY @PROCADDRESSES(DEVICE)

TYPE <MESSAGE> [ (NOTIFY) A ]

[ VOID ]

outputs MESSAGE (any character string not including ") to the teletype printer, followed by CR, LF.

Eg,

TYPE <THIS IS A MESSAGE>

TYPE <NOW WE ENTER P1>, ENTER P1

TYPE NL

outputs CR, LF to the teletype printer.

(Do does TYPE <> , but at the expense of generating an empty buffer).

TEST E (1) (2) (3) (4)

tests the status byte of the buffer defined by E and suspends processing until any previously initiated input or output is done, or an error detected.

Eg,

TEST OPBUFFER READY FOR NEXT OUTPUT

TEST E ERRORS

sets up a mechanism for use of the following JUMP statements:-

JUMP TO E IF [ EOM ] [ EOF ] [ TRUNC ] [ MODE ] [ CHKSUM ]

ERROR

causes a jump to the address specified by E if the specified error is detected by the input/output executive. The errors are:

EOM: end of medium, eg, no tape in punch.

EOF: end of file.

TRUNC: truncation of an input message (buffer too small).
MODE: message not formatted according to mode.

CHKSUM: Checksum error on formatted binary inputs.

Any number of different error types may be specified, in any order.

Eg,

TEST BUFFER ERRORS
JUMP TO L1 IF TRUNC ERROR
JUMP TO L2 IF MODE ERROR
TEST IPBUFFER ERRORS
JUMP TO BAD1 IF CHKSUM ERROR
JUMP TO BAD2 IF EOM ERROR
JUMP TO BAD3 IF MODE ERROR
JUMP TO BAD4 IF EOF ERROR

TEST and TEST/JUMP statements need not immediately follow the associated INPUT or OUTPUT statement. They could, for example, be located at addresses specified in "notify" clauses.

Eg,
OUTPUT BUFFER TO LSP, NOTIFY DONE ; Processing continues while
, ; buffer is emptied by interrupt.

(End of procedure)

DONE:  (Start of test procedure)

TEST BUFFER ERRORS

JUMP TO BAD1 IF MODE ERROR

etc.

Blank parameter fields in output buffers may be filled using the CONVERT macro:

CONVERT [WORD] A1 (TO) (ASCII) [OCT] (AT) A2

converts the word or byte at the location specified by A1 to an octal or binary ASCII character string in the byte field specified by A2.

Eg,

CONVERT WORD AZIMUTH TO ASCII OCT AT #OPBUFFER+25

Debug teletype listing is obtained using the macro LIST:

LIST A1 [WORDS] (FROM) A2 (IN) [OCT]

[BYTES] [BIN]

Processing is suspended while the listing is in progress.

Eg,

LIST #4 WORDS FROM #OPDATA IN OCT

3.6 Conditionals

Conditional statements are constructed from the macros IF, THEN, ELSE and END.

The general form of conditional clause is

IF [BYTE] A1 R A2
[WORD]

where R = ( less than
) greater than
= equal to
The items compared are the operands defined by address specifications $A_1$ and $A_2$. Thus:

**IF** WORD W1 = W2 means "if the word named (whose address is) W1 is equal to the word named W2".

**IF** BYTE RI = @BYTEADDRESS means "if the low order byte in register 1 equals the byte whose address is in location BYTEADDRESS.

**IF** WORD W1-2 = #4 means "if the word preceding W1 is equal to 4", and is not the same as "IF WORD W1 = #6".

Conditional "GOTO" statements take the form

```
IF [BYTE] A1 R A2 [BRANCH] (TO) E
```

where E defines a label.

Eg.

**IF** BYTE @BYTEADDRESSES(INDEX) = CHARACTER(INDEX), JUMP TO LABEL1+6

BRANCH is shorter and quicker than JUMP, but is restricted to a label offset of ±125 words. (Violation generates an assembler error report).

Simple conditional consequences and alternatives can be contained in the single line statement:-

```
IF [BYTE] A1 R A2 THEN <STATEMENT> [ELSE <STATEMENT>] [VOID]
```

where STATEMENT is any MACRO-11 statement, or any single line PSEUDO statement. (Note that although THEN and ELSE are themselves macro names, in this context they act simply as parameters for the macro IF.)

Eg.

**IF** WORD W1)(W2 THEN -ADD W3,W4> ELSE <OUTPUT BUFFER TO TTY>

**IF** BYTE FLAG = #ON THEN <IF WORD W1 = W2 THEN <TYPE <MESSAGE>>>

Where more than one line is required, the construction is:-
IF \[ \text{BYTE} \begin{bmatrix} A_1 & R & A_2 \end{bmatrix} \text{WORD} \]
THEN BEGIN
Consequence statement sequence
END
ELSE BEGIN
Alternative statement sequence
END
Nesting is allowed to any practical level. ELSE BEGIN clauses are optional.

Eg,

IF \text{WORD} W_1 = W_2
THEN BEGIN
  IF \text{BYTE} \text{FLAG} = \#0
  THEN BEGIN
    \text{TYPE} \langle W_1 = W_2, \text{FLAG} = 0 \rangle
  END
ELSE BEGIN
  \text{TYPE} \langle W_1 = W_2, \text{FLAG NON-ZERO} \rangle
  \text{CLR} \text{FLAG}
  \text{TYPE} \langle \text{FLAG RE-SET TO ZERO} \rangle
END
END
ELSE BEGIN
  IF \text{WORD} W_1 > W_2 \text{ THEN } \langle \text{TYPE} \langle W_1 B\text{IGGER} \rangle \rangle \text{ ELSE } \langle \text{TYPE} \langle W_2 B\text{IGGER} \rangle \rangle
  IF \text{BYTE} \text{FLAG} = \#0 \text{ THEN } \langle \text{IF} \text{WORD} W_1 = \#4, \text{JUMP TO LABEL} \rangle
  IF \text{BYTE} \text{FLAG} \langle(\text{FLAG1}) \rangle\text{ THEN BEGIN}
    \text{TYPE} \langle \text{FLAGS NOT EQUAL} \rangle
  END
  \text{MOV} \text{FLAG1}, \text{FLAG}
END
Incorrect nesting in the form of too many "ENDS" makes the END macro generate an error report and return the nesting to base level. Too few "ENDS" will normally only be detected by the FINISH macro used to terminate a source text. A check at any END in the text may be forced by giving ? as a parameter. This causes END's to be inserted as required to return the nesting to base level, with an error report if applicable.

The IF clause in all constructions of conditional statements may take a form which makes use of the state of specific bits in the processor status word. These bits, called N, V, C and Z, are set following instruction execution as follows:-

Z: = if the result was zero.
N: = if the result was negative.
C: = if a carry from the most significant bit occurred.
V: = if arithmetic overflow occurred.

This type of IF clause takes the form

IF CONDITION

where CONDITION is one of the symbols CSET, CCLEAR, NSET, NCLEAR, VSET, VCLEAR, ZSET, ZCLEAR, POSITIVE, NEGATIVE, ZERO, NONZERO, SET, CLEAR, OVERFLOW or CARRY, or any symbol equated to one of these symbols by an assignment statement.

Eg,
ADD A,B
IF ZERO, BRANCH TO LABEL ; if A was equal to -B.
TST WORD1 ; Test WORD 1
IF POSITIVE THEN <P> ELSE <Q> ; If WORD 1 is positive do
; statement P else do statement
; Q
BIT #1100, WORD1 ; Test bits 6 and 9 of WORD1
IF SET .... ; If either set ....
BIC #1100, WORD1 ; Clear bits 6 and 9 of WORD1
IF NONZERO .... ; If any other bits set ....

BLACK = POSITIVE
WHITE = NEGATIVE
GREY = ZERO
TST GREYSCALE
IF BLACK JUMP TO L1
IF WHITE JUMP TO L2
IF GREY JUMP TO L3

The last example shows three successive tests being applied to the same result. The tests themselves do not change the result, nor do branch, jump, jump to subroutine, and return from subroutine instructions. Thus the status bits can be used as Boolean communicators.

Eg,

ERROR = VSET

SEN ; Set N bit as a parameter for P1.

DO P1 ; Procedure call.

IF ERROR .... ; If P1 set V bit ....

IF WORD W1 = W2 THEN BEGIN
    IF BYTE B1 = B2 THEN <DO P1> ELSE <DO P2>
END ELSE BEGIN
    IF BYTE B1 = B2 THEN <DO P3> ELSE <DO P4>
END

IF ERROR .... ; If error flagged by whichever procedure ran ....

Care must be taken to avoid ambiguity, however, when status word conditionals follow each other.

Eg,

TST WORD 1
IF POSITIVE THEN <ADD WORD2, WORD3>

IF ZERO .... ; "if WORD1 is zero ...." if WORD1 is non-positive, but
              ; "if WORD3 is now zero ...." if WORD1 is positive.

17
3.7 Loops

The general form of construction for loop control is:

LOOP

LOOP IFCLAUSE

Where IFCLAUSE can be any of the IF clause constructions. If the condition in the IF clause is satisfied, processor control returns to the preceding matching LOOP. LOOPS may be nested to any (practical) level.

An alternative construction is:

LOOP

LOOP A TIMES

where A specifies a register or word location where the loop count is held. This count is decremented on each iteration of the loop, and the loop is left when the count is zero. If A specifies a register, this form gives the fastest and most economical method of control, but is limited to a loop length of 250 words.

Eg,

LOOP

MOV COUNT, LOOPCOUNT

LOOP

LOOP

LOOP IF WORD W1 = W2

LOOP LOOPCOUNT TIMES

BIT #MASK, LOOPCNTRL

LOOP IF SET ; Loop if any masked bits are set.

Nesting errors are detected and reported either by the LOOP macro or by FINISH.

3.8 Stack Operations

SAVE (1)

puts the contents of registers R0-R5 on stack.
Eg,

**SAVE REGISTERS**

**UNSAVE (1)**

restores the contents of registers R0-R5 from the stack.

Eg,

**UNSAVE REGISTERS**

**STACK <A₁, A₂, A₃, .......Aₙ>**

pushes the words defined by A₁-Aₙ onto the stack.

Eg,

**STACK <ITEM1, ITEM2,(POINTER), @ADDRESSES(INDEX),#4,#"XY>**

**UNSTACK <A₁,A₂,A₃ .......Aₙ>**

successively pops word from the stack into the specified locations.

Eg,

**UNSTACK <WORD₁,WORD₂,6(POINTER),v(INDEX)+>**

**RESERVE N (1) (2) (3) (4) (5) (6) (7) (8)**

makes space on the stack for N words.

Eg,

**RESERVE 4 WORDS ON STACK FOR SUB-Routine Answers.**

**DISCARD N (1) (2) (3) (4) (5) (7) (8)**

pops N word off the stack and discards them.

Eg,

**DISCARD 4 STACK WORDS JUST USED FOR SUB-Routine Answers.**

3.9 Procedure Calls

Procedure input or output parameters may be passed on stack, or in registers.

**DO A  A₁, A₂, ....... Aₙ>**

puts the words specified by A₁, A₂ ....... Aₙ on stack, enters the program specified by A, and on return restores the stack to its original state.

Eg:

**DO P₁ <PARAM₁, #5, LIST(INDEX), @(R1)> ; Direct entry.**
DO P2(SWITCHVALUE) ; Switched entry (no parameters) via a jump table

DO @PROCADDRESS <PARAM1, PARAM2, #"XY"> ; Indirect entry.

DO @PROC(PROCNUMBER) <PARAM1, PARAM2> ; Switched indirect entry, via a procedure address table.

These calls use the program counter as a linkage register. The correct procedure exit is set up by the macro call "EXIT".

A calling program can make space on the stack for procedure answers by using RESERVE and DISCARD as shown above. Since the stack is used to hold linkage information for interrupt and sub-routine calls, each procedure must leave the stack pointer, on exit, in the same position as it found it on entry.

3.10 Arithmetic Operations

Macros MUL and DIV assume use of the extended arithmetic unit (KEII-A). All address specifications must define words, and single or double length (32 bit) operations are possible. Where double length operands are specified the first word is least significant.

Permissible MUL and DIV statements are:

MUL A1 BY A2
MUL A1 BY A2 (ANSWER) IN A3
MUL A1 BY A2 (ANSWER) IN A3, A4
MUL BY A1
DIV A1 BY A2
DIV A1 BY A2 (ANSWER) IN A1
DIV A1 BY A2 (ANSWER) IN A3 (REMAINDER) IN A4
DIV A1, A2 BY A3
DIV A1, A2 BY A3 (ANSWER) IN A4
DIV A1, A2 BY A3 (ANSWER) IN A4 (REMAINDER) IN A5
DIV BY A1

E.g.,

MUL WORD1 BY WORD2 ; The product WORD1 X WORD2
MUL BY WORD3 ; X WORD3
MUL BY #'4, ANS IN WORD4, WORD5 ; X4 is put in double length location ; WORD4,WORD5.
DIV (POINTER) BY #6
MUL BY @LIST(INDEX)
DIV BY DIVISOR+4, ANSWER IN WORD1, REM IN @ADDRESS

4 OPERATION

The only programming restriction is that symbols of the form Sdigitstring should not be used.

PSEUDO macros are held on disc in the DOS macro file SYSMAC.SML. A source text is headed by

.MCALL MACROS

PSEUDO

On reading the .MCALL directive, the assembler brings all PSEUDO macros into core. The macro call PSEUDO is then expanded to make all assignments and global declarations required by the language. The text is terminated by the macro call FINISH which checks for nesting errors, and supplies the normal ".END" directive recognised by the assembler. Some PSEUDO statements generate procedure calls. These procedures (BUFFST, SAVE, UNSAVE, CONVERT, LIST, NL and BIOX) are held in a system object file (PSUSRS.OBJ/CC) which must be linked with the object modules generated by PSEUDO.

PSEUDO syntax errors are reported via error reports embedded in the macro definitions. Errors in the generated code are reported normally by the assembler, with printout of the offending code (in assembly language). Listings appended show:

Appendix 2: A typical source text.
Appendix 3: Listing of the assembly, with load map and symbol table.
Appendix 4: Listing of the assembly, with conditionally satisfied macro expansion.
Appendix 5: PSEUDO macro definitions.

Preferably, PSEUDO requires a system with 24K of core store. It has been run on a minimum system, with 16K of core, and 64K disc, the only restriction being that some macros had to be left on disc, (by removing their names from the MACROS macro) and called individually as required by user texts. The macros selected were INPUT, OUTPUT, MUL, DIV, TEST, JUMP, READY, STEP, CYCLE.

5 COMMENTS

PSEUDO has so far been in use for about 9 man-months, producing 10K of fairly complex real-time control software. The time and effort required to write and debug programs written in PSEUDO has proved insignificant in relation to overall system software development. On no occasion has debugging required macro expansion listings. Run-time and storage overheads are virtually nil, compared with normal assembly language.

The power of the language obviously is restricted in relation to modern high level languages; for example, with regard to allowable data structures and data
types. But the power is sufficient to the present application, and to most real-time control applications.

With a little ingenuity on the part of the programmer (a fraction of that which he normally exercises in generating incomprehensibility) and providing his natural laziness at the typewriter can be overcome, PSEUDO can be used to produce highly readable source texts, requiring little additional documentation. In comparing PSEUDO with a conventional compiler, the reader should note that development of PSEUDO took only 5 man-weeks.
APPENDIX 1 Macro generation: examples.

In its usual form a macro consists of a defined, named, body of code, embodying declared formal parameters. The macro is called by name, with a list of actual parameters which replace corresponding formal parameters in the expansion. For example, using MACRO-11 terminology, after the macro definition.

```
.MACRO DO P ; macro name is DO. Its formal parameter is P
JSR PC, P
.ENDM
```

the statement

DO INPUTPROCEDURE

will generate the code

```
JSR PC, INPUTPROCEDURE
```

In MACRO-11R the use of assembly directives (in the body of the macro definition) and macro-processor directives allows modification of the expanded code, other than the simple replacement of formal parameters by actual parameters. For example, a section of the macro body may be omitted (at expansion time) if particular actual parameters are blank, undefined, have a particular value, consist of a particular character string, etc.

Thus, the PSEUDO macro definition for DO is:­

```
.MACRO DO P X

.IF B <X> ; If X is blank (no actual supplied).
JSR PC, P ; generate the procedure call code.
.MEXIT ; and exit from the macro.
.ENDC ; (end of conditional).

STACK <X> ; Else call macro STACK, to generate the code required to
; put the procedure parameters defined by X on stack, and
; to set symbol S10000 equal to the number of bytes of stack
; space used.
JSR PC, P ; generate the call to "P".
ADD #S10000, SP ; then generate the code required to reset the stack
; pointer, to its original position.

.ENDM
```

Some macros used in PSEUDO do not generate code directly, but are used to create or modify symbols or directives used by the assembler. An example is the macro SFORMI, called by (nested in) macros LOOP, ELSE, END and THEN. This has the definition:­
This generates a symbol SACTUAL, where ACTUAL is the symbol supplied as the actual parameter, and gives it as value the current (compile-time) value of the assembly location counter (represented by the symbol.).

However, the call of SFORML has the form:-
The back-slash is a macro-processor directive, indicating that the actual parameter we wish to pass is not the symbol S00004 but the ASCII octal character string representing the value of S00004. Thus, if S00004 = 0105 the macro call will generate a symbol S105, and equate this to the value of the location counter; i.e., it will generate an assemble-time label.

Typical usage of SFORMI, and of various types of conditionals is exemplified by the macro LOOP:-

```
.MACRO LOOP A I X R Y ; five formal parameters.
.IF B A ; If "A" is blank (no actual parameters) must
         ; be start of a new loop:--
    S00004-S00004+3 ; Increase resting-level count
    SFORMI \S0004 ; and form a label for loop return.
    .MEXIT ; and exit from macro.
.ENDC

.IF NB A ; End of loop.

.IF LT S00004-10 ; If nesting-level count is less than 10.
    ; ("ground" level is 7) there has been a nesting
    ; error.
    SY <LOOP> ; So call macro SY to generate an error report
    ; in the assembly listing.
.ENDC

.IF IDN I, TIMES ; If "I" is the character string TIMES.
    DEC A ; generate the code DEC "A".
    J2 \S00004 ; then call macro J2 to generate the code
    ; required to branch back to the label set up
    ; at the start of this loop if "A" is non-zero;
    S00004-S00004-3 ; drop the nesting level count.
    .MEXIT ; and exit from the macro.
.ENDC

.IF DIF A, IF ; If "A" is not the character string IF (and
    ; we are still in the macro!) there is a syntax
    SY < A ; error so call SY to report
    .MEXIT ; and exit.
.ENDC
```
.IF B X

; If "X" is blank, call is "LOOP IF CONDITION"
; type:-

J3 I \$00004

; Call macro J3 to generate the code required
; to branch or jump (depending on the length
; of the loop) back to the label created at
; the start of this loop, if the condition is
; satisfied.

S00004=S00004-3

; Drop the nesting level count

.MEXIT

; and exit.

.ENDC

; "X" is non-blank. We must have a call of
; the "LOOP IF ITEM X R Y" type:-

P Y

; Call macro P to check that there are no
; unspecified actuals* (otherwise report error).

K I

; Call macro K to check that "I" is the
; character string WORD or BYTE* (otherwise
; report error).

J1 I X R Y \$00004

; Call macro J1 to generate the code required
; to jump or branch (depending on the length
; of the loop) back to the label created at
; the start of this loop if the condition is
; satisfied.

S00004=S00004-3

; Drop the nesting level count.

.ENDC

.ENDM

*The nesting-level count is stepped by 3 to avoid a clash of generated symbols.
LOOP invokes generation of symbols S7, S12, S15 ...., THEN and ELSE invoke
generation of S10, S13, S16 ......, and END invokes generation of S11, S14, S17
......

*P and K are further examples of macros which do not produce code. They direct
the assembler to output an error report to the assembly listing when source
program syntax errors are detected.
APPENDIX 2. TYPICAL SOURCE TEXT.

*TITLE EXAMPLE
*FILENAME LIST AND CONVERT S/R5, USED BY PSEUDO.
*CALL MACROS PSEUDO

*PAGE
*FILENAME LIST
JDEUBUG LISTING PROGRAM, ENTERED VIA MACRO "LIST",
ENTERED WITH A0 CONTAINING NUMBER OF ITEMS FOR LISTING
A1 CONTAINING ADDRESS OF FIRST ITEM
A2 CONTAINING LISTING CODES
B1 LIST BYTES IN OCTAL
B2 LIST WORDS IN OCTAL
B3 LIST BYTES IN BINARY
B4 LIST WORDS IN BINARY.
LISTING IS TO TTY, FORMATTED IN COLUMNS.

CREATE BUFFER DBUFFER, 64, CHAR, TO HOLD ONE TTY LINE.
CREATE WORDS 0CHARGENED TO HOLD NUMBER OF CHARs PER TTY WORD.

LISTEND= A0 ADDRESS OF LAST ITEM.
ITEMPOINTER=R1 ADDRESS OF ITEM CURRENTLY BEING LISTED.
COLUMNS=R3 NUMBER OF COLUMNS LEFT IN CURRENT TTY LINE.
_OPCODE=R2 OPERATION CODE.
_CHARSET=R4 COUNT OF NUMBER OF CHARs PUT IN DBUFFER.
BUFFPOINTER=R5 10DBUFFER POINTER.

TAB=11
WORDBIT=2

NUMBER OF CHARACTERS PER COLUMN, AND NUMBER OF COLUMNS PER
LINE, AS A FUNCTION OF OP CODE=
WIDTH: BYTE 3
TTCOL: BYTE 6
BYTE 4
BYTE 8
BYTE 6
BYTE 4
BYTE 16
BYTE 3

#DESPATCH VECTORS FOR CONVERSION ROUTINES=
CONVERSION1 CONV0B CONV0C CONV0D CONV0E
CONV1B CONV1C CONV1D CONV1E

#CONVERSION ROUTINES=
CONVCONVCONV BYTE ITEMPOINTER TO ASCII OCT AT DBUFFER.
CONVCONVCONV WORD ITEMPOINTER TO ASCII OCT AT DBUFFER.
CONVCONVCONV BYTE ITEMPOINTER TO ASCII BIN AT DBUFFER.
CONVCONVCONV WORD ITEMPOINTER TO ASCII BIN AT DBUFFER.
LISTEN
ADD R1, R0, LISTENO,
MOVB WOSIZE(Opcode), CHARSGENERATED
TYPE ED, NL
READY BUFFER FOR ASCII OUTPUT TO TTY
LOOP
POINT BUFFER AT BUFFER DATA
MOVB TTYCOLS(Opcode), COLUMNS
CLM CHARCOUNT
LOOP
DU #CONVERSIONS(Opcode)
BIT #HONBIT, Opcode
IF SET THEN <ADD #2, ITEMPINTER> ELSE <INC ITEMPINTER>
ADD CHARSGENERATED, CHARCOUNT
ADD CHARSGENERATED, BUFFER POINTER
ULC COLUMNS
IF ZER0, BANCE TO LINTERMINATION
MOVB STAR, BUFFER POINTER
INC CHARCOUNT
LOOP IF WOND ITEMPINTER (= LISTEND
LINTERMINATION1
MOVB CR, BUFFER POINTER
MOVB LP, BUFFER POINTER
ADD #2, CHARCOUNT
SET BUFFER COUNT TO CHARCOUNT
OUTPUT BUFFER TO TTY
TEST BUFFER TRANSFER DONE
LOOP IF WOND ITEMPINTER (= LISTEND
EXIT

PAGE
SBITL CONVERT
#BINARY TO ASCII STRING CONVERSION, ENTERED VIA MACRO "CONVERT"
ENTERED WITH R0 CONTAINING ADDRESS OF FIELD AT WHICH ASCII
CHARS ARE TO BE PLACED, R1 CONTAINING BYTE OR WORD FOR CONVERSION
AND R2 CONTAINING Opcode=1
/ 8=CONVERT BYTE TO OCTAL STRING
/ 2=WORD
/ 4=BINARY
/ 6=BINARY WORD
/ CHARACTER MASKS, AS A FUNCTION OF Opcode=1=
CMASKI: 177778
17776
17776
17776

NUMBER OF CHARs DEVELOPED, AS A FUNCTION OF Opcode=1=
CHARNO: 3 13 CHARs IN AN OCTAL BYTE,
6 13 CHARs IN AN OCTAL BYTE,
8 13 CHARs IN AN OCTAL BYTE,
16, 13 CHARs IN AN OCTAL BYTE

Opcode=R2
MASK=R5
CHARCOUNT=R3
CFILE=xR6
OPITEM=R1
WORKSPACE=R4

2-2
CONVERT:

MOV C H, MASK (OPCODE), #0
MOV CHAR, (OPCODE), #0
ADD CMOUNT, CPFE RD

FIELD IS F I L L E D "B A C K W A R D S ."

NEXTCHAR:

MOV OPITEM, WORKSPACE
GET ITEM,
ADD #06, WORKSPACE
ICONV E R T TO ASCII,
MOV WORKS PAC E, -(CPIELD) AND PUT IT IN FIELD.

DEC CMOUNT
IF NONZERO
THEN BEGIN
IF WORD OPCODE = #4
THEN BEGIN
LSHIFT OPITEM 1 PLACE R
END
ELSE BEGIN
LSHIFT OPITEM 3 PLACES R
END
END
ELSE BEGIN
EXIT
END

FINISH
APPENDIX 3. ASSEMBLY LISTING.

EXAMPLE MACHO YRBSA R1-JAN-72 62119

TABLE OF CONTENTS

1 = 3 LIST AND CONVERT S/HS. USED BY PSEUDO.
2 = LIST
3 = CONVERT
TITLE EXAMPLE

.BOTTLE LIST AND CHVENT SYSRS, USED BY PSEUDO.

.MCALL MACRO

.PSEUDO

.PSEUDO
EXAMPLE MACRO VM054 01-JAN-72 02119 PAGE 2

LIST

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55

`BRTL LIST
IDBGL LISTING PROGRAM ENTERED VIA MACRO "LIST".
ENTERED WITH R0 CONTAINING NUMBER OF ITEMS FOR LISTING
R1 CONTAINING ADDRESS OF FIRST ITEM
R2 CONTAINING LISTING CODE
0=LIST BYTES IN OCTAL
1=LIST WORDS IN OCTAL
2=LIST BYTES IN BINARY
3=LIST WORDS IN BINARY.
LISTING IS TO TTY, FORMATTED IN COLUMNS.
CREATE BUFFER DBUFFER, 64 CHARS, TO HOLD ONE TTY LINE.
CREATE VARIABLES DBUFFER, 64 CHARS, TO HOLD NUMBER OF CHAR PER TTY LINE.

ADDRESS OF LAST ITEM,
ADDRESS OF ITEM CURRENTLY BEING LISTED,
NUMBER OF ITEMS LEFT IN CURRENT TTY LINE.
OPERATION CODE,
COUNT OF NUMBER OF CHAR PUT IN DBUFFER,
DBUFFER POINTER.

NUMBER OF CHARACTERS PER COLUMN AND NUMBER OF COLUMNS PER LINE AS A FUNCTION OF CODE.

IOESPATCH VECTORS FOR CONVERSION ROUTINES:

CONV
CONV
CONV
CONV
CONV
CONV
CONV
CONV

CONV
CONV
CONV
CONV
CONV

LIST
ADD R1, R0
MOV W0, W0 (OPCODE), CHAR GENERATED

TYPE NL

3-3`
Example Mach V: 01-Jan-72 02119 Page 2+

List

56 00312  HEADY BUFFER FOR FASCII OUTPUT TO TTY
57 00316  LOOP
58 00316  POINT BUFFPOINTER AT BUFFER DATA
59 00322  MOVX TCOLS(OPCODE),COLUMNS
60 00326  005000
61 00330  CLR CHANCOUNT
62 00330  LOOP
63 00330  DO #CONVERSIONS(OPCODE)
64 00340  BIT #WORDIT,OPCODE
65 00352  177530
66 00356  177534
67 00362  005303
68 00364  11d725
69 00366  000011
70 00372  005204
71 00374  INC CHANCOUNT
72 00374  LOOP IF #OMIT ITEMPOINTER (# LISTEND
73 00400  LINETERMINATION
74 00400  MOVX #CR,(BUFFPOINTER) *
75 00404  112725
76 00408  000019
77 00414  MOVX #LF,(BUFFPOINTER) *
78 00418  1127e5
79 00424  000012
80 00430  ADD #2,CHARCOUNT
81 00436  SET BUFFER COUNT TO CHARCOUNT
82 00440  OUTPUT BUFFER TO TTY
83 00446  TEST BUFFER TRANSFER DONE
84 00452  LOOP IF WORD ITEMPOINTER (# LISTEND
85 00456  EXIT

3-4
EXAMPLE MACRO VR05A 01-JAN-72 02110 PAGE 3

CONVERT

1  
2  BBTL CONVERT
3  BINANAY TO ASCII STRING CONVERSION, ENTERED VIA MACRO "CONVET"
4  ENTERED WITH R0 CONTAINING ADDRESS OF FIELD AT WHICH ASCII
5  CHARAS ARE TO BE PLACED, R1 CONTAINING BYTE OR WORD FOR CONVERSION
6  JAMU #2 CONTAINING OPCODE
7  I #CONVET BYTE TO OCTAL STRING
8  I 2#........WORD
9  I 4#........BYTE, BINARY ...
10  I 6#........WORD
11  ICHARACTER MASKS, AS A FUNCTION OF OPCODE
12  00444 17777777 CHMASK 17777777
13  00444 17777777 CHMASK 17777777
14  00444 17777777 CHMASK 17777777
15  00444 17777777 CHMASK 17777777
16  17  NUMBER OF CHARS DEVELOPED, AS A FUNCTION OF OPCODE
17  00454 00000 1 CHAN#1 3 13 CHARS IN AN OCTAL BYTE
18  00454 6 5 ETC
19  00440 00000 6 A
20  00442 00000 16,
21  22  U00002 OPCODE##3
23  00444 00000 5 MASKS##5
24  00444 00000 3 CHCOUNTER##3
25  00444 00000 0 FIELD##0
26  00444 00001 OPITEM##1
27  00444 00004 WORKSPACE##4
28  29  CNVERTI
30  00464 01625 MOV CHMASK(OPCODE), MASK
31  00474 01625 MOV CHMASK(OPCODE), Mask
32  00474 01623 MOV CHMASK(OPCODE), CHCOUNT
33  00474 01623 MOV CHMASK(OPCODE), CHCOUNT
34  00474 01620 ADD CHCOUNT, CFIELD IF FIELD IS FILLED "BACKWARDS"
35  00474 01620 ADD CHCOUNT, CFIELD IF FIELD IS FILLED "BACKWARDS"
36  00474 01618 NEXTCHAR
37  00474 01618 NEXTCHAR
38  00474 01606 MOV OPITEM, WORKSPACE IGET ITEM
39  00474 01606 MOV OPITEM, WORKSPACE IGET ITEM
40  00474 01604 BIC MASK, WORKSPACE IMASK IT
41  00474 01604 BIC MASK, WORKSPACE IMASK IT
42  00474 01604 ADD #60, WORKSPACE ICONVERT TO ASCII
43  00474 01604 ADD #60, WORKSPACE ICONVERT TO ASCII
44  00474 01600 MOV WORKSPACE, -(CFIELD) AND PUT IT IN FIELD
45  00474 01600 MOV WORKSPACE, -(CFIELD) AND PUT IT IN FIELD
46  00510 00530 DEC CHCOUNT
47  00510 00530 DEC CHCOUNT
48  00510 00512 IF NONZERO
49  00512 00512 IF NONZERO
50  00514 00514 THEN BEGIN
51  00514 00514 THEN BEGIN
52  00520 00520 IF WORD OPCODE = #A
53  00526 00526 IF WORD OPCODE = #A
54  00526 00526 THEN BEGIN
55  00526 00526 THEN BEGIN
56  00532 00532 LSHIFT OPITEM 1 PLACE R
57  00532 00532 LSHIFT OPITEM 1 PLACE R
58  00536 00536 END
59  00536 00536 END
60  00536 00536 ELSE BEGIN
61  00536 00536 ELSE BEGIN
62  00536 00536 LSHIFT OPITEM 3 PLACES R
63  00536 00536 LSHIFT OPITEM 3 PLACES R
64  00552 00552 BR NEXTCHAR
65  00552 00552 BR NEXTCHAR
66  00534 00534 END
67  00534 00534 END
68  00534 00534 ELSE BEGIN
69  00534 00534 ELSE BEGIN
70  00542 00542 EXIT
71  00542 00542 EXIT

EXAMPLE MACRO VN05A 01-JAN-72 02110 PAGE 3

FINISH

FINISH

3-5
<table>
<thead>
<tr>
<th>AC</th>
<th>177392</th>
<th>BID</th>
<th>******</th>
<th>G</th>
<th>.RUFFP=10000F5</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUPTST</td>
<td>******</td>
<td>BYTE</td>
<td>00010</td>
<td></td>
<td>CARRY=0000F3</td>
</tr>
<tr>
<td>CCLEAR</td>
<td>000000</td>
<td>CFIELD</td>
<td>1000000</td>
<td></td>
<td>CHARCO=1000BF</td>
</tr>
<tr>
<td>CMACU</td>
<td>000000</td>
<td>CMASK</td>
<td>000000</td>
<td></td>
<td>CMCONM=000023</td>
</tr>
<tr>
<td>CMCONU</td>
<td>000000</td>
<td>CMCON</td>
<td>000120</td>
<td></td>
<td>CMVLT=00046R</td>
</tr>
<tr>
<td>CONDB</td>
<td>000133</td>
<td>CONDB</td>
<td>000142</td>
<td></td>
<td>CONN=00248R</td>
</tr>
<tr>
<td>CH</td>
<td>000000</td>
<td>CONDB</td>
<td>000103</td>
<td></td>
<td>CONVIR=000120R</td>
</tr>
<tr>
<td>DIV</td>
<td>177348</td>
<td>MSP</td>
<td>0000006</td>
<td></td>
<td>DBUFF=000000</td>
</tr>
<tr>
<td>ITEMD</td>
<td>20000001</td>
<td>RND</td>
<td>00000000</td>
<td></td>
<td>DEC=000003</td>
</tr>
<tr>
<td>LINTA</td>
<td>000000</td>
<td>LIST</td>
<td>000276G</td>
<td></td>
<td>LF=0000012</td>
</tr>
<tr>
<td>LBP</td>
<td>000000</td>
<td>LBP</td>
<td>000003</td>
<td></td>
<td>LISTH=1000000</td>
</tr>
<tr>
<td>MUP</td>
<td>177304</td>
<td>MUL</td>
<td>17758B</td>
<td></td>
<td>NCLEAN=000001</td>
</tr>
<tr>
<td>NEGATI</td>
<td>177777</td>
<td>NEXTCH</td>
<td>000476R</td>
<td></td>
<td>NL=******G</td>
</tr>
<tr>
<td>NONZEN</td>
<td>00002</td>
<td>NSET</td>
<td>177777</td>
<td></td>
<td>OPCODE=1000002</td>
</tr>
<tr>
<td>OPEMTE</td>
<td>000001</td>
<td>OVENFL</td>
<td>000004</td>
<td></td>
<td>PC=1000007</td>
</tr>
<tr>
<td>POSITA</td>
<td>000001</td>
<td>R8</td>
<td>177777</td>
<td></td>
<td>R1=1000001</td>
</tr>
<tr>
<td>R4</td>
<td>1000002</td>
<td>R4</td>
<td>1000008</td>
<td></td>
<td>R4=1000004</td>
</tr>
<tr>
<td>R5</td>
<td>1000005</td>
<td>R5</td>
<td>1000006</td>
<td></td>
<td>RT=1000007</td>
</tr>
<tr>
<td>SAVE</td>
<td>******</td>
<td>SET</td>
<td>000002</td>
<td></td>
<td>SETUP=******G</td>
</tr>
<tr>
<td>SP</td>
<td>0000006</td>
<td>SPACE</td>
<td>0000000</td>
<td></td>
<td>SR=177776</td>
</tr>
<tr>
<td>SWR</td>
<td>177570</td>
<td>SPACE</td>
<td>000002</td>
<td></td>
<td>SRR=100001</td>
</tr>
<tr>
<td>$10000</td>
<td>0000007</td>
<td>SPACE</td>
<td>000002</td>
<td></td>
<td>$10000=0000001</td>
</tr>
<tr>
<td>$14</td>
<td>000306R</td>
<td>S12</td>
<td>000316R</td>
<td></td>
<td>$13=000564</td>
</tr>
<tr>
<td>$15</td>
<td>000330R</td>
<td>S15</td>
<td>000530R</td>
<td></td>
<td>$14=000256R</td>
</tr>
<tr>
<td>$17</td>
<td>000529H</td>
<td>TAB</td>
<td>000011</td>
<td></td>
<td>TTCOLS=00011R</td>
</tr>
<tr>
<td>TTY</td>
<td>000401</td>
<td>UNSAVE</td>
<td>******G</td>
<td></td>
<td>VCLEAN=000005</td>
</tr>
<tr>
<td>VSET</td>
<td>000004</td>
<td>WOSIZE</td>
<td>00110R</td>
<td></td>
<td>WORD=000005</td>
</tr>
<tr>
<td>WORDB1</td>
<td>000002</td>
<td>WOSIZE</td>
<td>00110R</td>
<td></td>
<td>WORD=000005</td>
</tr>
<tr>
<td>CMD</td>
<td>000000</td>
<td>WORKSP</td>
<td>0000008</td>
<td></td>
<td>WORD=000000</td>
</tr>
</tbody>
</table>

ERRORS DETECTED: 0
FREE CORE: 7585, WORDS: 01, 0T1=0101
IAPPENDIX 4. ASSEMBLY LISTING, WITH MACRO EXPANSION.

/(NON-SATISFIED CONDITIONALS NOT LISTED.)
EXAMPLE MACRO VMBSA 81=JAN=72 02120

TABLE OF CONTENTS

1= 3 LIST AND CNVERT 8/RS, USED BY PSEUDO.
2= 1 LIST
3= 1 CNVERT
TITLE EXAMPLE

MACRO LIST AND CONVERT B/R, USED BY PSEUDO.

MCALL MACRO

MOVE WORDs
MOVE M1
MOVE M2
MOVE M3
MOVE 5
MOVE 6
MOVE 7
MOVE PC
MOVE SP
177776 S=177776
177570 S=177570
177594 F=177594
177592 C=177592
177594 U=177594
177706 MUL=177706
177711 POSITIVE=1
177777 NEGATIVE=1
177777 NEGATIVE=1
MOVE ZERO
MOVE ZERO
MOVE CLEAN=0
MOVE NONZ=2
MOVE SET=2
MOVE CARRY=3
MOVE SET=3
MOVE OVERFLOW=4
MOVE UNZ=6
MOVE VSET=6
MOVE CLEA=1
MOVE CLEAN=5
MOVE CLEA=1
MOVE UNZ=7
MOVE CLEAR=6
MOVE SPACE=8

GLOBAL HUFTST, SAVE, UNSAVE, SETUP, CONVERT, BIOX, LIST, NL

S0012 LF=12
S00040 SPACE=40
S00115 CH=15
S007 SQR=9
S0010 4ADD=40
P00116 SPACE=9.
LIST

CREATE BUFFER DBUFFER,64, CHANS, TO HOLD ONE TTY LINE.

CREATE WORDS CHARSGENERATED TO HOLD NUMBER OF CHARS PER TTY.

CREATE WORDS CHARSGENERATED TO HOLD NUMBER OF CHARACTERS PER COLUMN.

INCREASE SHORT LENGTH OF LISTING.

WITH LISTING CODE.

LISTING IS TO TTY, Formatted in Columns.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

LISTING IS TO TTY, Formatted in Columns.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.

CREATE BUFFER DBUFFER,64, CHAR, TO HOLD ONE TTY LINE.
CONVERSION ROUTINES

E0178 E8138 P HUFFPOINTER
E0180 K HTT
E0182 H OCT
E0184 SAVE
E0186 J8H78 J8N PC, SAVE
E0188 .EXIT
E018A 8088A 8188A
E018C 8188A 8288A
E018E 8388A 8488A 8588A 8688A 8788A 8888A 8988A 8A88A 8B88A 8C88A 8D88A 8E88A 8F88A
E0190 8088A 8188A
E0192 8288A 8388A
E0194 8488A 8588A 8688A 8788A
E0198 8088A 8188A
E019C 8288A 8388A
E019E 8488A 8588A 8688A 8788A
E01A0 8088A 8188A
E01A2 8288A 8388A
E01A4 8488A 8588A 8688A 8788A
E01A8 8088A 8188A
E01AC 8288A 8388A
E01AE 8488A 8588A 8688A 8788A
E01B0 8088A 8188A
E01B2 8288A 8388A
E01B4 8488A 8588A 8688A 8788A
E01B8 8088A 8188A
E01BC 8288A 8388A
E01BE 8488A 8588A 8688A 8788A
E01C0 8088A 8188A
E01C2 8288A 8388A
E01C4 8488A 8588A 8688A 8788A
E01C8 8088A 8188A
E01CC 8288A 8388A
E01CE 8488A 8588A 8688A 8788A
E01D0 8088A 8188A
E01D2 8288A 8388A
E01D4 8488A 8588A 8688A 8788A
E01D8 8088A 8188A
E01DC 8288A 8388A
E01E0 8488A 8588A 8688A 8788A
E01E4 8088A 8188A
E01E8 8288A 8388A
E01EC 8488A 8588A 8688A 8788A
E01F0 8088A 8188A
E01F2 8288A 8388A
E01F4 8488A 8588A 8688A 8788A

4-4
EXAMPLE MACHINE V1.0-5-07 CSE21 PAGE 13

```
00176  UNSTACKIW
00179  UN.Popen
00180  DC CONVERT
00181  JSR PC,CONVERT
00182  ,EXIT
00183  UNSAVE
00184  UU UNSAVE
00185  LS 3/7 JSR PC,UNSAVE
00186  ,EXIT

00210  EXIT
00211  AP@77 MTS PC
00212  CUNARICCONVET BYTE @ITEMPOINTER TO ASCII BIN AT BUFFERINTER
00214  P BUFFERINTER
00215  + BYTE
00216  + SAVE
00219  JSP PC,SAVE
0021A  ,EXIT
0021B  STACK BUFFERINTER
00230  S1B200
00231  + RUFFINTER
00232  UN.Popen
00234  ,EXIT
00235  JSR PC,UNSTACK
00237  JSR PC,UNSTACK
00238  ,EXIT
00239  UNSAVE
00240  UU UNSAVE
00241  LS 3/7 JSR PC,UNSAVE
00242  JSP PC,SAVE
00243  ,EXIT
```

4-5
\texttt{LIST}

\texttt{48 \texttt{0044} \texttt{0077} \texttt{M TR} \texttt{PC}}

\texttt{49 \texttt{0044} \texttt{C} \texttt{CURRENT} \texttt{M TR} \texttt{ITEM POINTER TO ASCII BIN AT MTR POINTER}}

\texttt{0044} \texttt{M MTR POINTER}

\texttt{0044} \texttt{M TR}

\texttt{0044} \texttt{SAVE}

\texttt{0044} \texttt{BC4768} \texttt{IN PC, SAVE}

\texttt{MEXIT}

\texttt{W052} \texttt{STACK MTR POINTER}

\texttt{W052} \texttt{MTR POINTER}

\texttt{W052} \texttt{LMT RPTR}

\texttt{W052} \texttt{MTR RPTR}

\texttt{W052} \texttt{MTR RPTR}

\texttt{W052} \texttt{MTR RPTR}

\texttt{W052} \texttt{MTR RPTR}

\texttt{MEXIT}

\texttt{W070} \texttt{UNSAVE}

\texttt{W070} \texttt{UNSAVE}

\texttt{W070} \texttt{BC4768} \texttt{IN PC, UNSAVE}

\texttt{MEXIT}

\texttt{50 \texttt{0074} \texttt{W074} \texttt{MTRPC}}

\texttt{51}

\texttt{52 \texttt{0076} \texttt{LIST}}

\texttt{53 \texttt{0076} \texttt{BC4768} \texttt{M TR MTR END}}

\texttt{54 \texttt{0076} \texttt{BC4768} \texttt{M TR MTR END}}

\texttt{55 \texttt{0076} \texttt{TYPE NL}}

\texttt{55 \texttt{0076} \texttt{TYPE NL}}

\texttt{55 \texttt{0076} \texttt{TYPE NL}}

\texttt{MEXIT}

\texttt{56 \texttt{BC12} \texttt{HEADER BUFFER FOR ASCII OUTPUT TO TTY}}

\texttt{BC12 \texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12}}

\texttt{MEXIT}

\texttt{57 \texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12}}

\texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12}

\texttt{MEXIT}

\texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12}

\texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12}

\texttt{MEXIT}

\texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12}

\texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12} \texttt{BC12}

\texttt{MEXIT}
EXAMPLE MACRO V4M054 RI=JAN-72 P2120 PAGE 20

LIST

and $316$ 

,'MEXIT

50 00316  POINT BUFFPOINTER AT DBUFFEN DATA

00316 012105 MOV #BUFFEN+6,BUFFPOINTER

00316 

,'MEXIT

59 00322 11b203

,'MEXIT

MVB TCOLS(OPCODE),COLUMNS

00311

60 00326 005104 CLR CHMCOUNT

61 00334 005102 LUDF

69 00336 005102 SUBBUF#BUFFPOINTER+3

00336 005102 

,'MEXIT

00336

62 00336 004772 J5W PC,#CONVERSIONS(OPCODE)

004772 

,'MEXIT

63 00335 032767

000002

64 00345  DO #CONVERSIONS(OPCODE)

 IF SET THEN ADD #2,ITEMPOINTER ELSE INC ITEMPOINTER

00345

65 00345  B ONE,BEG THEN INC #2,ITEMPOINTER ELSE INC ITEMPOINTER

66 00345

67 00352  ADD #2,ITEMPOINTER

00352

68 00352  ADD #2,ITEMPOINTER

00352 

,'MEXIT

00352

69 00352  ADD CHARGENCODED,CHMCOUNT

00352

70 00352  ADD CHANGENCODED,BUFFPOINTER

00352

71 00352  DEC COLUMNS

00352
EXAMPLE MACRO YV05A 0I-JAN-72 02128 PAGE 2

LIST

68 U0364 IF ZENO, BRANCH TO LINTERMINATION
   69 U0364 A NEG, ONE BRANCH <TO> LINTERMINATION 
   68 U0364 U0105 NEW LINTERMINATION
   68 U0364 .EXIT

70 U0366 112725 MOV $0, (BUFFPOINTER) +

71 U0374 INC CHANCOUNT
    71 U0374 LOOP IF WORD ITEMPOINTER (* LISTEND
    71 U0374 P LISTEND
    71 U0374 X WIRD
    71 U0374 J1 WIRD ITEMPOINTER (* LISTEND \808684
    71 U0374 IF WIRD ITEMPOINTER (* LISTEND BRANCH TO 315
    71 U0374 X WIRD
    71 U0374 R00100 CMP ITEMPOINTER, LISTEND
    71 U0374 B BLOS, HIM BRANCH <TO> 315 <>
    71 U0374 101754 MO3 S15
    71 U0374 .EXIT
    71 U0374 .EXIT
    71 U0374 000112 $0006 = $0006 + 3

72 U0360 LINTERMINATION!
73 U0360 112725 MO3 $CR, (BUFFPOINTER) *

75 U0368 112725 MO3 $LF, (BUFFPOINTER) *

76 U0418 002704 ADD #1, CHARCOUNT

77 U0444 SET BUFFER COUNT TO CHARCOUNT
    77 U0444 014687 MO3 CHARCOUNT, BUFFER + 8
    77 U0444 17736A .EXIT

78 U0420 OUTPUT BUFFER TO TTY
    78 U0420 .MCALL 10
    78 U0420 IO
    78 U0420 KB0 = 0
    78 U0420 TTY = 1
    78 U0420 LS = 3
    78 U0420 MS = 4
    78 U0420 UO B10X
    78 U0420 UO B10X
    78 U0420 JSR PC, B10X
    78 U0420 .EXIT

79 U0424 MOV BUFFER
    79 U0424 MO3 BUFFER*
    79 U0424 0120 812, 13, TTY
    79 U0424 VI
    79 U0424 .EXIT

79 U0430 TEST BUFFER TRANSFR DONE
    79 U0430 101576 TST BUFFER + 3
    79 U0430 177367 .EXIT

80 U0404 LOOP IF WORD ITEMPOINTER (* LISTEND
    80 U0404 P LISTEND

4-8
EXAMPLE MACRO VRB5A 01-JAN-72 02120 PAGE 2

LIST

00436  K WORD
00436  J1 WORD ITEMPOINTED (= LISTEND \SUBB8A
00436  IF WORD ITEMPOINTED (= LISTEND \BRANCH TO S12
00436  K \HORN
00436  U20100 CMP ITEMPOINTED, LISTEND
0P400  B \BLO8,BMI \BRANCH <TO> S12 <<
00400 101726 \BLO8 S12
,\EXIT
,\EXIT
,\EXIT
UPDENT ? \BLO8\=\SUBB8A=3
01 00442 \EXIT
00442 004207 \RTS PC
02
EXAMPLE MACRO VMUSA 01-JAN-72 PAGE 3

CONVERT

1       .UNTIL CONVERT
2       . PRIMARY TO ASCII STRING CONVERSION, ENTERED VIA MACRO "CONVERT",
3       . ENTERED WITH KM CONTAINING ADDRESS OF FIELD AT WHICH ASCII
4       . CHARACTERS ARE TO BE PLACED, H1 CONTAINING BYTE OR WORD FOR CONVERSION
5       . HAN K 2 CONTAINING OP CODE =
6       / 1 #CONVERT BYTE TO OCTAL STRING
7       / 2#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#..#.
EXAMPLE MACNO VH05A 01-JAN-72 02120 PAGE 3

49 US552  END
   US552  SFOM1 \300000
   US552  US552 \9179
   US552  SFOM2 \500000
   US552  US552 \3100
   US552  SFOM3 JMP \500000
   US552  US552 \517
   US552  0010
   US552  US552 \171
   US552  056
   US552  US552 \14 \300000+\300000-3
   US552  US552 \13 \300000+\300000-3
   US751  UN NEXTCHAr
50 US552  END
   US552  SFOM1 \300000
   US552  US552 \14 \300000
   US552  SFOM2 \500000
   US552  US514 \13
   US514  SFOM3 JMP \300000
   US514  US167 JMP 814
   US514  0034
   US554  014
   US554  056
   US554  US554 \500000+\500000-3
   US554  US554 \500000+\500000-3
51 US554  ELSE BEGIN
   US554  056
   US554  US554 \500000+\500000-3
   US554  US554 \500000+\500000-3
   US554  SFOM2 \300000
   US554  US167 JMP 814
   US554  US514
   US514  JMP \300000
   US514  US520
   US520  US554 \13
   US554  SFOM1 \300000
   US554  US554 \313
   US554  US554 \313
   US554  US554 \14
52 US554  EXIT
   US554  US554 \500000+\500000-3
53 US554  END
   US554  SFOM1 \300000
   US554  US554 \313
   US554  SFOM2 \300000
   US554  US554 \313
   US554  SFOM3 JMP \300000
   US554  US167 JMP 814
   US554  0034
   US554  056
   US554  US554 \500000+\500000-3
54 US554  FINISH
   US554  US554 \14
55 US554  END
<table>
<thead>
<tr>
<th>SYMBOL TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AC</strong> = 17732</td>
</tr>
<tr>
<td><strong>BUFSIZE</strong> = ****** G</td>
</tr>
<tr>
<td><strong>CCLUDE</strong> = 00000</td>
</tr>
<tr>
<td><strong>CHANO</strong> = 00050</td>
</tr>
<tr>
<td><strong>CHMARK</strong> = 00040</td>
</tr>
<tr>
<td><strong>COLUMN</strong> = 00000</td>
</tr>
<tr>
<td><strong>CONUM</strong> = 00013</td>
</tr>
<tr>
<td><strong>CR</strong> = 000015</td>
</tr>
<tr>
<td><strong>DIV</strong> = 177350</td>
</tr>
<tr>
<td><strong>ITEMPO</strong> = 004001</td>
</tr>
<tr>
<td><strong>LINETE</strong> = 00040</td>
</tr>
<tr>
<td><strong>LSP</strong> = 00040</td>
</tr>
<tr>
<td><strong>MU</strong> = 177304</td>
</tr>
<tr>
<td><strong>NEGATI</strong> = 177777</td>
</tr>
<tr>
<td><strong>NONZEN</strong> = 000002</td>
</tr>
<tr>
<td><strong>OPITEM</strong> = 000101</td>
</tr>
<tr>
<td><strong>PUSITI</strong> = 0000B1</td>
</tr>
<tr>
<td><strong>R2</strong> = 00400A2</td>
</tr>
<tr>
<td><strong>R5</strong> = 000000</td>
</tr>
<tr>
<td><strong>SAVE</strong> = ****** G</td>
</tr>
<tr>
<td><strong>SP</strong> = 000000</td>
</tr>
<tr>
<td><strong>SNR</strong> = 177570</td>
</tr>
<tr>
<td><strong>SNUI</strong> = 000007</td>
</tr>
<tr>
<td><strong>SIO</strong> = 000004</td>
</tr>
<tr>
<td><strong>S14</strong> = 000562</td>
</tr>
<tr>
<td><strong>S17</strong> = 000552</td>
</tr>
<tr>
<td><strong>TTY</strong> = 000001</td>
</tr>
<tr>
<td><strong>VSET</strong> = 00000A</td>
</tr>
<tr>
<td><strong>WDATE</strong> = 000002</td>
</tr>
<tr>
<td><strong>ZSET</strong> = 000003</td>
</tr>
</tbody>
</table>

**ERRORS DETECTED**: 0
**FREE CORE**: 7565, **Words**
**OTIM=OTIA/LISTEN/MICRU**
APPENDIX 5. PSEUDO MACRO DEFINITIONS.

*MACHO MACRO US
*MCALL WITH, TYPE, CREATE, LSHIFT, OL, EL
*MCALL POINT, IF, THEN, ELSE, SPWON, SPOHE, STACK, UNSTACK
*MCALL SPWON, SPOHE, B, SET, SET, SETUP, INIT, LIST, P, G, M, P2, O
*MCALL LOUP, J1, J2, J3, CONVERT, PSEUDO, R, M, UN, BY, SAVE, UNSAVE, GD
*MCALL TEST, JUMP, HEADD, STEP, CYCLE, INPUT, OUTPUT, MUL, DIV
*MCALL RESERVE, DISCARD, FINISH, EXIT
*ENOM

*MACHO UL
*USAIL LSH
*ENUM
*MACHO EL
*ENAM LSh
*ENUM
*MACHO UN
*ENGIN UNSPEC PARAM
*ENUM
*MACHO Y X
*ERHON SYNTAX: X
*ENUM
*MACHO P A
*IF R E
*ENUC
*ENOM
*MACHO K 1
*IF (IF I = UO)
*IF (IF I = BYTE)
*ERHON BYTE OR WORD?
*ENUC
*ENOM
*ENOM

*MACHO STACK X
$1B3478
*INP U, <X>
*MOV U, = (SP)
$1B4156
*ENOM
*ENOM
*MACHO UNSTACK X
*INP U, $16
*MOV (SP)$0, U
*ENOM
*ENOM
*MACHO USCANU N A B C D E F G H
*ADD $NON, SP
*ENOM
*MACHO RESERVE N A B C D E F G H
*SUB $NON, SP
*ENOM

*MACHO FINISH X
*IF NE SPUMG = 8
*ENOM
*ENUC
*IF NE SPUMG = 7
*ENOM
*ENOM
*MACHO EXIT
*RTS PLC
*ENOM
,MACRO CONVERT I X TO A T A I
P L
K I
M T
SAVE
STACK 1
.IF ION I,WORD
MOV X,R1
P T
,ENUC
.IF ION I,BYTE
MOV R,R1
BIC #174444,R1
G T
,ENUC
UNSTACK MD
DO CONVERT
UNSAVE
,ENDM

,MACRO LIST N I FN A W T
P T
M T
.IF DIF I,BYTES
.IF DIF I,WORDS
BY I
,MEXIT
,ENUC
,ENUC
SAVE
STACK N
DEC (SP)
MOV A,R1
.IF ION I,WORDS
F T
ABL (BP)
,ENUC
,IF ION I,BYTES
G T
,ENDC
UNSTACK N
DO LIST
UNSAVE
,ENDM

,MACRO F T
.IIF ION T,OCT,MOV #2,R2
.IIF ION T,HIN,MOV #4,R2
,ENDM
,MACRO G T
.IIF ION T,OCT,CLR R2
.IIF ION T,BIN,MOV #0,R2
,ENDM
,MACRO H T
.IIF DIF T,BIN
.IIF DIF T,OCT
BY T
,ENDC
,ENDC
,ENDM
.MACRO GET X Y IN Z
P X
.IF ION Y, BLOCKSIZE
MOV X=4, Z
.ENDC
.IF ION Y, STATUS
MOV X=3, Z
.ENDC
.IF ION Y, MODE
MOV X=2, Z
.ENDC
.IF ION Y, COUNT
MOV X=4, Z
.ENDC
.IF ION Y, COUNT
MOV X=4, Z
.ENDC
.ENDM

.MACRO GET X Y TO Z
P X
.IF ION Y, IP
MOV Z, X=10
.ENDC
.IF ION Y, OP
MOV Z, X=6
.ENDC
.IF ION Y, COUNT
MOV Z, X=4
.ENDC
.IF ION Y, COUNT
MOV Z, X=4
.ENDC
.ENDM

.MACRO SETUP
DO SETUP
.ENDM

.MACRO INIT Z T X A Y
P Y
DO BIOX
Y
.ENDM

.MACRO SAVE X
DO SAVE
.ENDM

.MACRO UNSAVE X
DO UNSAVE
.ENDM
MACRO IF I = N Y T P E O
  IF EW I
  B BNE, BNE X <N> Y <T>
  .MEIXT
  .ENUC
  IF EQ I=1
  B HPL, RMI X <N> Y <T>
  .MEIXT
  .ENOC
  IF EQ I=1
  B BMI, APL X <N> Y <T>
  .MEIXT
  .ENOC
  IF EQ I=2
  B ONE, AEU X <N> Y <T>
  .MEIXT
  .ENOC
  IF EQ I=3
  B OCS, MCC X <N> Y <T>
  .MEIXT
  .ENOC
  IF EQ I=4
  B BVS, MVC X <N> Y <T>
  .MEIXT
  .ENOC
  IF EQ I=5
  B BVC, RVS X <N> Y <T>
  .MEIXT
  .ENOC
  IF EQ I=6
  B MCC, BCS X <N> Y <T>
  .MEIXT
  .ENOC
  .IF NB <Q>
    K I
    .IF ION 1, B, BYTE, CMPB X, Y
    .IF ION 1, ND, CMP X, Y
    .IF ION <R>,
      B BLO, BNE T <P> E <O>
      .MEIXT
      .ENOC
    .IF ION <R>,
      B BNE, BNE T <P> E <O>
      .MEIXT
      .ENOC
    .IF ION <R>,
      B BHI, BLO T <P> E <O>
      .MEIXT
      .ENOC
    .IF ION <R>,
      B BHS, BLO T <P> E <O>
      .MEIXT
      .ENOC
  ENDIF

5-6
,MEXIT
,ENUC
,IF NA X3
,IF IUN X3,EL9E
,IF R < X6>
SY I
,MEXIT
,ENUC
USB
SFOM1 \SBURBS
**+2
X4
**+2
SFOM2 \SBRURB
SFOM3 \SBRURBS
SFOM4 \SBRURBC
X2
SFOM1 \SBUBRS
SFOM2 \SBUBRB
**+2
SFOM3 BR \SBURBS
USB
,MEXIT
,ENDC
,ENUC
USB
SFOM1 \SBURBS
**+2
X2
SFOM1 \SBURB
SFOM2 \SBURB
SFOM3 \SBURBC
0SR
,MEXIT
,ENDC
,ENUM
,MACRO THEN BGN X
,IF NB BGN
,IF DIF BGN,BEGIN
SY BGN
,MEXIT
,ENUC
,ENOC
,IF NB RUN
,PRINT "BEGIN"
,ENDC
,IF NB X
,IF DIF X,1
SY X
,MEXIT
,ENDC
,ENUC
USB
SFOM1 \SBURBS
**+4
,ENUM
.MACRO SPURMS SOURUS
.JMP $F480095
.ENDM

.MACRO CYCLE M Q R B T PL
.P T
.EL
.IF ION M,IP
.IF ION M,OP
.CMP T-10, T-2
.BLO L
.SUB T-12, T-10
.LI ADD T-8, T-10
.DL
.MEXIT
.ENDC
.IF ION M,IP
.CMP T-10, T-2
.BLO L
.SUB T-12, T-6
.LI ADD T-4, T-6
.DL
.MEXIT
.ENDC
.CMP M, T-2
.BLO L
.SUB T-12, M
.LI ADD T-8, M
.DL
.MEXIT
.ENDC
.IF ION M, BACK
.IF ION M, IP
.SUB T-4, T-6
.CMP T-10, #T
.BMIS L
.ADD T-12, T-10
.LI
.DL
.MEXIT
.ENDC
.IF ION M, OP
.SUB T-4, T-6
.CMP T-6, #T
.BMIS L
.ADD T-12, T-6
.LI
.DL
.MEXIT
.ENDC
.SUB T-4, M
.CMP M, #T
.BMIS L
.ADD T-12, M
.LI
.DL
.MEXIT
.ENDC
.BY M
.DL
.ENDM
MACRO POINT P TO Q R S T
IF ION G,END
MOV S-2,P
ADD B+B,P
EXIT
ENC
IF ION H,BLOCK
IF ION G,FIRST
MOV #T,P
EXIT
ENC
IF ION G, LAST
MOV T-2,P
EXIT
ENC
IF ION G, IP
MOV T-1B,P
EXIT
ENC
IF ION G, OP
MOV T-8,P
EXIT
ENC
BY T
ENC
IF ION R,DATA
MOV B+B,P
IF #S 3
IF DIFF 8, END
BY 3
EXIT
ENC
ADD G+B,P
ENC
EXIT
ENC
SY 7
ENUM

MACRO STEP A B C D E F
IF ION C,ON
IF ION A, IP
ADD E+A,E=18
EXIT
ENC
IF ION A, OP
ADD E+=E=6
EXIT
ENC
ADD E+=A
EXIT
ENC
IF ION C, BACK
IF ION A, IP
SUB E+=E=18
EXIT
ENC
EXIT
ENC
SUB E+=A
EXIT
ENC
BY C
ENUM
MACRO

1 1 X N Y

IF N A
   3A0000=3A0000+3
   3F0001=3F0001
   EXIT
   ENUC
   IF NA A
   IF LT 3A0000=10,
      BY <LOOP>
   ENUC
   IF IUN I,TIMES
   DEC A
   JZ 3B0000
   3B004=3B004+3
   EXIT
   ENUC
   IF DIP A,IF
   BY <A>
   EXIT
   ENUC
   IF X
   JS 1 3B0000
   3B0000=3B0000+3
   EXIT
   ENUC
   END

J 1 1 X R Y 3A
   IF LT 3A=246,
   IF X R Y BRANCH TO 3*84
   EXIT
   ENUC
   IF I X N Y JUMP TO 3*84
   ENDM

MACRO J1 1 X R Y 3A
   J 1 1 X R Y 3A
   ENDM

MACRO J2 84
   0NE 8*84
   ENDM

MACRO J3 1 3A
   IF LT 3A=246,
   IF 1 BRANCH TO 3*84
   EXIT
   ENUC
   IF JUMP TO 3*84
   ENDM
MACRO CREATE I X N U M P V W S T Y Z

IF ION $IP, LIST
IF $I $E
UN
MEXIT
ENDOR
IF ION $U, NORUN
NOW
X
X
NOW
IF ION $U, NORUN
X
X
MEXIT
ENDC
BY Y
MEXIT
ENDOR
IF ION $I, BUFFER
IF $I $E
UN
MEXIT
ENDOR
XIN
INTER $A
N
BLAM N
SB8800=SB8888+ON
SB8888+1
MEXIT
ENDOR
IF ION $I, WORDS
SB8888+1,
1RR $O, $I
QIN
SB8888=SB8888+2
ENDM
MEXIT
ENDOR
IF ION $I, BYTES
SB8888+1
1RR $J, $I
01, BYTE 0
SB8888=SB8888+1
ENDM
MEXIT
ENDOR
BY $I
ENDM

5-13
.MACRO WITH \( D \times \)
\( D = 384000 \)
\( D_{\text{max}} = 0 \)
\( \text{IF } D = 0 \# \text{WHI} \1 \)
\( \text{IRP} U, x \)
\( D_{\text{max}} = 038400 \)
\( \text{ENUM} \)
\( \text{ENC} \)
\( \text{IF } U = 38400 \)
\( \text{IRP} U, x \)
\( \text{BYTE} U \)
\( 383000 = 383002 + 1 \)
\( \text{ENOM} \)
\( \text{ENC} \)
\( 383000 = 383000 - 383000 \)
\( \text{END} \)

.MACRO TYPE M, N, D, T, L1, T2, L3, T4
\( \text{IF } D < M, \text{NL} \)
\( \text{EL} \)
\( D = 010X \)
\( L1 \)
\( \text{IF } B \) \( D \)
\( \text{BYTE} 12 : 1 \)
\( \text{ENC} \)
\( \text{IF } M = D \)
\( \text{BYTE} 1 + 1 \)
\( \text{END} \)
\( \text{ENC} \)
\( \text{BN} \) \( L4 \)
\( L1 + 10 \)
\( 0 \)
\( L3 = L3 + 2 \)
\( \text{ASC} \) \( = M \)
\( \text{BYTE} CR, LF \)
\( L21, \text{EVEN} \)
\( L4 \)
\( \text{DL} \)
\( \text{EXIT} \)
\( \text{ENC} \)
\( \text{ENQ} \)
\( \text{DO} \) \( \text{NL} \)
\( \text{ENOM} \)
MACHO READY A F M P U N S
IF B H
BY MOUF
EXIT
ENC
IF ION M,ASCII
MOV #2, A+2
EXIT
ENC
IF ION M, PASCII
CLR A+2
EXIT
ENC
IF ION M, PIN
MOV #3, A+2
EXIT
ENC
IF ION M, PIN
MOV #1, A+2
EXIT
ENC
BY M
ENOM
MACRO OUTPUT A T D N C
IF B U
ERROR 70EVEI
EXIT
ENC
MCALL 10
10
DO BIOS
A
IF H C
BYTE 12, D
EXIT
ENC
BYTE 14, D
C
ENOM
MACRO INPUT A T D N C
IF D D
ERROR 1UFFERT
EXIT
ENC
MCALL 10
10
DO BIOS
D
IF R C
BYTE 11, A
EXIT
ENC
BYTE 13, A
C
ENOM
.MACRO TEST N P Q R S
  .IF N = P
  .IF N = P, ERRORS
  STACK (P+3)
  JSR X,Y, HUP TST
  UNSTACK X
  ADD X, +12, (SP)
  MOV #(SP), (SP)
  JMP #(SP)
  +14
  +12
  +10
  +8
  +6
  +4
  +2
  +1
  .EXIT
  .ENDC
  .ENDC
  XSTB R=3
  XPL = 4
  .ENDM

.MACRO JUMP T L I E X
  .IF ION E, EUM
  .* = 16
  L
  .* = 8
  .EXIT
  .ENDC
  .IF ION E, EOF
  .* = 8
  L
  .* = 6
  .EXIT
  .ENDC
  .IF ION E, TRUNC
  .* = 6
  L
  .* = 6
  .EXIT
  .ENDC
  .IF ION E, MODE
  .* = 6
  L
  .* = 2
  .EXIT
  .ENDC
  .IF ION E, CHKSUM
  .* = 2
  L
  .EXIT
  .ENDC
  BY E
  .ENDM

.MACRO 10
  KB0=8
  TTY=1
  LSR=3
  MBR=5
  LSR=4
  MBR=6
  .ENDM