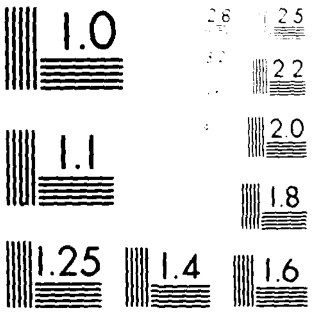
AD-A107 168   ROYAL SIGNALS AND RADAR ESTABLISHMENT MALVERN (ENGLAND)   F/G 9/2
'PSEUDO', A MACRO-BASED HIGH LEVEL LANGUAGE FOR THE PDP-11,(U)
APR 74 B DAVY
UNCLASSIFIED RSRE-MEMO-2875   DRIC-BR-64523   NL

1 OF 1
AD A
107168

END
DATE
FILMED
12-81
DTIC


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

[illegible]

(18) DRIC

(19) BR-64523

RSRE

MEMORANDUM No. 2875

ROYAL SIGNALS & RADAR ESTABLISHMENT

(14) RSRE-MEMO-2875

(12)

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

(10) Author : B Davy

(11) Apr 1974

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

RRE MEMORANDUM No 2875

PSEUDO A MACRO-BASED HIGH LEVEL LANGUAGE FOR THE PDP-11

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

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 langauges, 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 langauges, 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 devies are allocated specific addresses, allowing memory reference instructions to operate directly on data held in peripheral registers. The machine has eight program accessible registers, RO-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 independant 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; ie, 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)

```
Unaryoperator  =  +
                  -
```

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
```

eg:

```
ON = 1
NOTOFF  = ON
TTYREGISTER  = 777562
```

Registers may be named symbollically by the register assignment statement:-

```
Symbol = % Octaldigit
```

A register expression is any expression containing a symbol previously assigned to a register, eg:-

```
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 nem-monic) 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.

MODE	FORMAT OF A	
Register	R	The register defined by R contains the operand.
Deferred register	(ER)	The register defined by ER contains the address of the operand.
Auto-increment	(ER)+	The contents of the register defined by ER are incremented* after being used as the address of the operand.
Auto-decrement	-(ER)	The contents of the register defined by ER are decremented* before being used as the address of the operand.

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

*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 asigned 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 WORSPACE 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 E, E_1 .... are any expressions, as defined in 2.3.

ii A, A_1 .... are any operand address specification formats, as defined in 2.4.

iii S, S_1 .... 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:-

R0 = %0

R1 = %1

R7 = %7

SP = R6 (stack pointer)

PC = R7 (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:-

CREATE $\begin{bmatrix}\text{WORDS}\\ \text{BYTES}\end{bmatrix}$ <S_1, S_2 .... , S_n> (1) (2) (3) (4) (5) (6) (7) (8) (9) (10)

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 E_1 $\begin{bmatrix}\text{WORDS}\\ \text{BYTES}\end{bmatrix}$ E_2 (1) (2) (3) (4) (5) (6) (7)

allocates a block structured list named S, E_1 words or bytes long, E_2 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 5

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 o100000 (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 TELETYPE INPUT.

3.3 Data Presetting

Items of data may preset, using the macro WITH.

WITH (DATA) <E_1, E_2 ... E_n>

associates values E_1, E_2 .... 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  <ONE, TWO, THREE, TEN, SIXTEEN, FIFTY, TWO56>

WITH DATA  <1,2,3,10,20>

WITH DATA  <50.,400>

.EVEN

CREATE WORDS  TWO56ADDRESS,ASCIIXY, TWOADDRESS,LEFTATZERO>

WITH DATA  <TWO56,"XY,ONE+1>

CREATE BUFFER TTYOUTPUT,16. CHARS

WITH DATA  <'O,'U,'T,'P,'U,'T,SPACE,'M,'E,'S,'S,'A,'G,'E,CR,LF>
```

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

$$\text{POINT A (AT)} \begin{bmatrix}\text{FIRST}\\ \text{LAST}\\ \text{IP}\\ \text{OP}\end{bmatrix} \text{BLOCK (OF) E}$$

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 $\begin{bmatrix}\text{STATUS}\\ \text{MODE}\\ \text{COUNT}\end{bmatrix}$ (IN) A

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> $\begin{bmatrix}\text{(NOTIFY) A}\\ \text{VOID}\end{bmatrix}$

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.

(So 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 $\begin{bmatrix}\text{EOM}\\ \text{EOF}\\ \text{TRUNC}\\ \text{MODE}\\ \text{CHKSUM}\end{bmatrix}$ 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 $\begin{bmatrix}\text{WORD}\\ \text{BYTE}\end{bmatrix}$ A_1 (TO) (ASCII) $\begin{bmatrix}\text{OCT}\\ \text{BIN}\end{bmatrix}$ (AT) A_2

converts the word or byte at the location specified by A_1 to an octal or binary ASCII character string in the byte field specified by A_2.

Eg,

CONVERT WORD AZIMUTH TO ASCII OCT AT #OPBUFFER+25

Debug teletype listing is obtained using the macro LIST:-

LIST A_1 $\begin{bmatrix}\text{WORDS}\\ \text{BYTES}\end{bmatrix}$ (FROM) A_2 (IN) $\begin{bmatrix}\text{OCT}\\ \text{BIN}\end{bmatrix}$

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 $\begin{bmatrix}\text{BYTE}\\ \text{WORD}\end{bmatrix}$ A_1 R A_2

where R = (less than

) greater than

= equal to

)(	not equal to
)=	greater than or equal to
(=	less than or equal to
S(	arithmetically less than (signed integer)
S)	arithmetically greater than
S)=	arithmetically greater than or equal to
S(=	arithmetically less than or 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 R1 = @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

$$\text{IF} \quad \begin{bmatrix}\text{BYTE}\\ \text{WORD}\end{bmatrix} \quad A_1 \text{ R } A_2 \quad \begin{bmatrix}\text{BRANCH}\\ \text{JUMP}\end{bmatrix} \quad \text{(TO) E}$$

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:-

$$\text{IF} \quad \begin{bmatrix}\text{BYTE}\\ \text{WORD}\end{bmatrix} \quad A_1 \text{ R } A_2 \text{ THEN <STATEMENT>} \quad \begin{bmatrix}\text{ELSE <STATEMENT>}\\ \text{VOID}\end{bmatrix}$$

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:-

$$\text{IF}\quad \begin{bmatrix}\text{BYTE}\\ \text{WORD}\end{bmatrix}\quad A_1\ R\ A_2$$

```
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 WORD W1 = W2
THEN BEGIN
    IF BYTE FLAG = #0
    THEN BEGIN
                TYPE <W1 = W2, FLAG = 0>
    END
    ELSE BEGIN
                TYPE <W1 = W2, FLAG NON-ZERO>
                CLRB FLAG
                TYPE <FLAG RE-SET TO ZERO>
    END
END
ELSE BEGIN
    IF WORD W1 ) W2 THEN <TYPE <W1 BIGGER>> ELSE <TYPE<W2 BIGGER>>
    IF BYTE FLAG = #0 THEN <IF WORD W1 = #4, JUMP TO LABEL>
    IF BYTE FLAG )( FLAG1
    THEN BEGIN
                TYPE <FLAGS NOT EQUAL>
                MOVB FLAG1,FLAG
    END
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.
```

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 RO-R5 from the stack.

Eg,

UNSAVE REGISTERS

STACK <A_1, A_2, A_3, ...A_n>

pushes the words defined by A_1-A_n onto the stack.

Eg,

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

UNSTACK <A_1,A_2,A_3 ......,A_n>

successively pops word from the stack into the specified locations.

Eg,

UNSTACK <WORD1,WORD2,6(POINTER),@(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 <A_1, A_2, ... A_n>

puts the words specified by A_1, A_2 .... A_n 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 para-
                                             ; meters) 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 (KE11-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 A_1 BY A_2

MUL A_1 BY A_2 (ANSWER) IN A_3

MUL A_1 BY A_2(ANSWER) IN A_3,A_4

MUL BY A_1

DIV A_1 BY A_2

DIV A_1 BY A_2 (ANSWER) IN A_1

DIV A_1 BY A_2 (ANSWER) IN A_3 (REMAINDER) IN A_4

DIV A_1,A_2 BY A_3

DIV A_1, A_2 BY A_3 (ANSWER) IN A_4

DIV A_1, A_2 BY A_3 (ANSWER) IN A_4 (REMAINDER) IN A_5

DIV BY A_1

Eg,

```
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,CNVERT, 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 SFORM1, called by (nested in) macros LOOP, ELSE, END and THEN. This has the definition:-

```
.MACRO SFORM1 S00005

S'S00005 = .

.ENDM.
```

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 SFORM1 has the form:-

```
SFORM1  \S00004
```

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 = o105 the macro call will generate a symbol S105, and equate this to the value of the location counter; ie, it will generate an assemble-time label.

Typical usage of SFORM1, 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

    SFORM1 \S00004             ;   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 \S00004                     ;  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 \S00004                   ;  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
.SBTTL LIST AND CNVERT S/RS,USED BY PSEUDO.

.MCALL MACROS
PSEUDO






.PAGE
.SBTTL LIST
;DEBUG 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
;                  2=LIST WORDS IN OCTAL
;                  4=LIST BYTES IN BINARY
;                  6=LIST WORDS IN BINARY.
;LISTING IS TO TTY,FORMATTED IN COLUMNS.

CREATE BUFFER DBUFFER,64. CHARS,TO HOLD ONE TTY LINE.
CREATE WORDS <CHARSGENERATED> TO HOLD NUMBER OF CHARS PER TTY WORD.

LISTEND=R0        ;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.
CHARCOUNT=R4      ;COUNT OF NUMBER OF CHARS PUT IN DBUFFER.
BUFFPOINTER=R5    ;DBUFFER POINTER.

TAB=11
WORDBIT=2         ;THIS BIT IS SET IN OPCODE FOR WORD OPERATIONS.

;NUMBER OF CHARACTERS PER COLUMN,AND NUMBER OF COLUMNS PER
;LINE,AS A FUNCTION OF OPCODE:-
WDSIZE: .BYTE 3
TTCOLS: .BYTE 8.
        .BYTE 6
        .BYTE 8.
        .BYTE 8.
        .BYTE 4
        .BYTE 16.
        .BYTE 3

;DESPATCH VECTORS FOR CONVERSION ROUTINES:-
CONVERSIONS:    CONOB
                CONOW
                CONBB
                CONBW

;CONVERSION ROUTINES:-
CONOB:CONVERT BYTE @ITEMPOINTER TO ASCII OCT AT BUFFPOINTER
        EXIT
CONOW:CONVERT WORD @ITEMPOINTER TO ASCII OCT AT BUFFPOINTER
        EXIT
CONBB:CONVERT BYTE @ITEMPOINTER TO ASCII BIN AT BUFFPOINTER
        EXIT
CONBW:CONVERT WORD @ITEMPOINTER TO ASCII BIN AT BUFFPOINTER
        EXIT
```

```
LIST:
ADD R1,R0                    ;FORM LISTEND.
MOVB WDSIZE(OPCODE),CHARSGENERATED
TYPE NL
READY DBUFFER FOR FASCII OUTPUT TO TTY
LOOP
        POINT BUFFPOINTER AT OBUFFER DATA
        MOVB TTCOLS(OPCODE),COLUMNS
        CLR CHARCOUNT
        LOOP
                DO @CONVERSIONS(OPCODE)
                BIT #WORDBIT,OPCODE
                IF SET THEN <ADD #2,ITEMPOINTER> ELSE <INC ITEMPOINTER>
                ADD CHARSGENERATED,CHARCOUNT
                ADD CHARSGENERATED,BUFFPOINTER
                DEC COLUMNS
                IF ZERO,BRANCH TO LINETERMINATION
                MOVB #TAB,(BUFFPOINTER)+
                INC CHARCOUNT
        LOOP IF WORD ITEMPOINTER (= LISTEND

        LINETERMINATION:
        MOVB #CR,(BUFFPOINTER)+
        MOVB #LF,(BUFFPOINTER)+
        ADD #2,CHARCOUNT
        SET DBUFFER COUNT TO CHARCOUNT
        OUTPUT DBUFFER TO TTY
        TEST DBUFFER TRANSFER DONE
LOOP IF WORD ITEMPOINTER (= LISTEND
EXIT

.PAGE
.SBTTL CNVERT
;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:-
;       0=CONVERT BYTE TO OCTAL STRING
;       2=.........WORD.................
;       4=.........BYTE.....BINARY......
;       6=.........WORD.................

;CHARACTER MASKS,AS A FUNCTION OF OPCODE:-
CHMASK: 177770
        177770
        177776
        177776

;NUMBER OF CHARS DEVELOPED,AS A FUNCTION OF OPCODE:-
CHARNO: 3       ;3 CHARS IN AN OCTAL BYTE.
        6       ;ETC.
        8.
        16.

OPCODE=R2
MASK=R5
CHCOUNT=R3
CFIELD=R0
OPITEM=R1
WORKSPACE=R4
```

```
CNVERT:
        MOV CHMASK(OPCODE),MASK
        MOV CHARNO(OPCODE),CHCOUNT
        ADD CHCOUNT,CFIELD          ;FIELD IS FILLED "BACKWARDS".
NEXTCHAR:
        MOV OPITEM,WORKSPACE        ;GET ITEM,
        BIC MASK,WORKSPACE          ;MASK IT,
        ADD #60,WORKSPACE           ;CONVERT TO ASCII,
        MOVB WORKSPACE,-(CFIELD);AND PUT IT IN FIELD.

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

FINISH
```

```
/APPENDIX 3. ASSEMBLY LISTING.
/-----------------------------
EXAMPLE MACRO VR05A 01-JAN-72 02:19
TABLE OF CONTENTS
```

```
EXAMPLE MACRO VR05A 01-JAN-72 02:19 PAGE 1

1
2                      .TITLE EXAMPLE
3                      .SBTTL LIST AND CNVERT S/RS,USED BY PSEUDO.
4
5                      .MCALL MACROS
6 000000               PSEUDO
7
8
9
10
11
12
```

```
EXAMPLE MACRO VN05A 01-JAN-72 02:19 PAGE 2
LIST

 1                     .SBTTL LIST
 2                     ;DEBUG LISTING PROGRAM,ENTERED VIA MACRO "LIST".
 3                     ;ENTERED WITH   R0 CONTAINING NUMBER OF ITEMS FOR LISTING
 4                     ;               R1 CONTAINING ADDRESS OF FIRST ITEM
 5                     ;               R2 CONTAINING LISTING CODE:-
 6                     ;                  0=LIST BYTES IN OCTAL
 7                     ;                  2=LIST WORDS IN OCTAL
 8                     ;                  4=LIST BYTES IN BINARY
 9                     ;                  6=LIST WORDS IN BINARY.
10                     ;LISTING IS TO TTY,FORMATTED IN COLUMNS.
11
12 00000               CREATE BUFFER DBUFFER,64. CHARS,TO HOLD ONE TTY LINE.
13 00106               CREATE WORDS <CHARSGENERATED> TO HOLD NUMBER OF CHARS PER TTY WO
14
15        000000 LISTEND=R0        ;ADDRESS OF LAST ITEM.
16        000001 ITEMPOINTER=R1    ;ADDRESS OF ITEM CURRENTLY BEING LISTED.
17        000003 COLUMNS=R3        ;NUMBER OF COLUMNS LEFT IN CURRENT TTY LINE.
18        000002 OPCODE=R2         ;OPERATION CODE.
19        000004 CHARCOUNT=R4      ;COUNT OF NUMBER OF CHARS PUT IN DBUFFER.
20        000005 BUFFPOINTER=R5    ;DBUFFER POINTER.
21
22        000011 TAB=11
23        000002 WORDBIT=2         ;THIS BIT IS SET IN OPCODE FOR WORD OPERATIONS.
24
25                     ;NUMBER OF CHARACTERS PER COLUMN,AND NUMBER OF COLUMNS PER
26                     ;LINE,AS A FUNCTION OF OPCODE:-
27 00110     003 WDSIZE: .BYTE 3
28 00111     010 TTCOLS: .BYTE 8.
29 00112     006         .BYTE 6
30 00113     010         .BYTE 8.
31 00114     010         .BYTE 8.
32 00115     004         .BYTE 4
33 00116     020         .BYTE 16.
34 00117     003         .BYTE 3
35
36                     ;DESPATCH VECTORS FOR CONVERSION ROUTINES:-
37 00120 000130'CONVERSIONS:     CONOB
38 00122 000162'                 CONOW
39 00124 000212'                 CONBB
40 00126 000246'                 CONBW
41
42                     ;CONVERSION ROUTINES:-
43 00130               CONOB:CONVERT BYTE @ITEMPOINTER TO ASCII OCT AT BUFFPOINTER
44 00160                       EXIT
45 00162               CONOW:CONVERT WORD @ITEMPOINTER TO ASCII OCT AT BUFFPOINTER
46 00210                       EXIT
47 00212               CONBB:CONVERT BYTE @ITEMPOINTER TO ASCII BIN AT BUFFPOINTER
48 00244                       EXIT
49 00246               CONBW:CONVERT WORD @ITEMPOINTER TO ASCII BIN AT BUFFPOINTER
50 00274                       EXIT
51
52 00276               LIST:
53 00276 060100 ADD R1,R0                  ;FORM LISTEND.
54 00300 116267 MOVB WDSIZE(OPCODE),CHARSGENERATED
         000110'
         177600
55 00306               TYPE NL
```

```
EXAMPLE MACRO V005A 01-JAN-72 02:19 PAGE 2+
LIST

56 00312        READY DBUFFER FOR FASCII OUTPUT TO TTY
57 00316        LOOP
58 00316                  POINT BUFFPOINTER AT DBUFFER DATA
59 00322 116203           MOVB TTCOLS(OPCODE),COLUMNS
         000111'
60 00326 005004           CLR CHARCOUNT
61 00330                  LOOP
62 00330                            DO @CONVERSIONS(OPCODE)
63 00334 032702                     BIT #WORDBIT,OPCODE
         000002
64 00340                            IF SET THEN <ADD #2,ITEMPOINTER> ELSE <INC ITEMP
65 00352 066704                     ADD CHARSGENERATED,CHARCOUNT
         177530
66 00356 066705                     ADD CHARSGENERATED,BUFFPOINTER
         177524
67 00362 005303                     DEC COLUMNS
68 00364                            IF ZERO,BRANCH TO LINETERMINATION
69 00366 112725                     MOVB #TAB,(BUFFPOINTER)+
         000011
70 00372 005204                     INC CHARCOUNT
71 00374                  LOOP IF WORD ITEMPOINTER (= LISTEND
72
73 00400                  LINETERMINATION:
74 00400 112725           MOVB #CR,(BUFFPOINTER)+
         000015
75 00404 112725           MOVB #LF,(BUFFPOINTER)+
         000012
76 00410 062704           ADD #2,CHARCOUNT
         000002
77 00414                  SET DBUFFER COUNT TO CHARCOUNT
78 00420                  OUTPUT DBUFFER TO TTY
79 00430                  TEST DBUFFER TRANSFER DONE
80 00436        LOOP IF WORD ITEMPOINTER (= LISTEND
81 00442        EXIT
82
EXAMPLE MACRO VR05A 01-JAN-72 02:19 PAGE 3
CNVERT

1                       .SBTTL CNVERT
2                       ;BINARY TO ASCII STRING CONVERSION,ENTERED VIA MACRO "CONVERT".
3                       ;ENTERED WITH R0 CONTAINING ADDRESS OF FIELD AT WHICH ASCII
4                       ;CHARS ARE TO BE PLACED,R1 CONTAINING BYTE OR WORD FOR CONVERSIO
5                       ;AND R2 CONTAINING OPCODE:-
6                       ;       0=CONVERT BYTE TO OCTAL STRING
7                       ;       2=.........WORD................
8                       ;       4=.........BYTE.....BINARY......
9                       ;       6=.........WORD................
10
11                      ;CHARACTER MASKS,AS A FUNCTION OF OPCODE:-
12 00444 177770 CHMASK: 177770
13 00446 177770         177770
14 00450 177776         177776
15 00452 177776         177776
16
17                      ;NUMBER OF CHARS DEVELOPED,AS A FUNCTION OF OPCODE:-
18 00454 000003 CHARNO: 3       ;3 CHARS IN AN OCTAL BYTE.
19 00456 000006         6       ;ETC.
20 00460 000010         8.
21 00462 000020         16.
22
23       000002 OPCODE=R2
24       000005 MASK=R5
25       000003 CHCOUNT=R3
26       000000 CFIELD=R0
27       000001 OPITEM=R1
28       000004 WORKSPACE=R4
29
30 00464        CNVERT:
31 00464 016205         MOV CHMASK(OPCODE),MASK
         000444'
32 00470 016203         MOV CHARNO(OPCODE),CHCOUNT
         000454'
33 00474 060300         ADD CHCOUNT,CFIELD      ;FIELD IS FILLED "BACKWARDS".
34 00476        NEXTCHAR:
35 00476 010104         MOV OPITEM,WORKSPACE    ;GET ITEM,
36 00500 040504         BIC MASK,WORKSPACE      ;MASK IT,
37 00502 062704         ADD #60,WORKSPACE       ;CONVERT TO ASCII,
         000060
38 00506 110440         MOVB WORKSPACE,-(CFIELD);AND PUT IT IN FIELD.
39
40 00510 005303         DEC CHCOUNT
41 00512                IF NONZERO
42 00514                THEN BEGIN
43 00520                        IF WORD OPCODE )= #4
44 00526                        THEN BEGIN
45 00532                                LSHIFT OPITEM 1 PLACE R
46 00536                        END
47 00536                        ELSE BEGIN
48 00542                                LSHIFT OPITEM 3 PLACES R
49 00552                        END
50 00552 000751                 BR NEXTCHAR
51 00554                END
52 00554                ELSE BEGIN
53 00560                        EXIT
54 00562                END




EXAMPLE MACRO VR05A 01-JAN-72 02:19 PAGE 3+
CNVERT

55
56 00562        FINISH
```

```
EXAMPLE MACRO V005A 01-JAN-72 02:19 PAGE 3+
SYMBOL TABLE

AC      = 177302           BIOX    = ****** G          BUFFPO =%000005
BUFTST= ****** G           BYTE    = 000010            CARRY  = 000003
CCLEAR= 000006             CFIELD=%000000              CHARCO =%000004
CHANNO  000454R            CHANSG   000106R            CHCOUN =%000003
CHMASK  000444R            CLEAN   = 000000            CNVERT   000464RG
COLUMN=%000003             CONBB    000212R            CONBW    000246R
CONOB   000130R            CONOW    000162R            CONVER   000120R
CR      = 000015           CSET    = 000003            DBUFFE   000000R
DIV     = 177300           HSP     = 000006            HSR    = 000005
ITEMPO=%000001             KBD     = 000000            LF     = 000012
LINETE  000400R            LIST     000276RG           LISTEN =%000000
LSP     = 000004           LSR     = 000005            MASK   =%000005
MQ      = 177304           MUL     = 177306            NCLEAR = 000001
NEGATI= 177777             NEXTCH   000476R            NL     = ****** G
NONZER= 000002             NSET    = 177777            OPCODE =%000002
OPITEM=%000001             OVERFL= 000004              PC     =%000007
POSITI= 000001             R0      =%000000            R1     =%000001
R2      =%000002           R3      =%000003            R4     =%000004
R5      =%000005           R6      =%000006            R7     =%000007
SAVE    = ****** G         SET     = 000002            SETUPP = ****** G
SP      =%000006           SPACE   = 000040            SR     = 177776
SWR     = 177570           S00000= 000002              S00001 = 000000
S00004= 000007             S00005= 000010              S00006 = 000011
S10000= 000002             S12     = 000316R           S13    = 000554R
S14     = 000562R          S15     = 000330R           S16    = 000536R
S17     = 000552R          TAB     = 000011            TTCOLS   000111R
TTY     = 000001           UNSAVE= ****** G            VCLEAR = 000005
VSET    = 000004           WDSIZE   000110R            WORD   = 000007
WORDBI= 000002             WORKSP=%000004              ZERO   = 000000
ZSET    = 000000
. ABS.  000000      000
        000562      001

ERRORS DETECTED:  0
FREE CORE:  7585. WORDS
,DT1<DT1X
```

```
;APPENDIX 4. ASSEMBLY LISTING,WITH MACRO EXPANSION.
;--------------------------------------------------------
;(NON-SATISFIED CONDITIONALS NOT LISTED.)
EXAMPLE MACRO VH05A 01-JAN-72 02:20
TABLE OF CONTENTS

    1-  3 LIST AND CNVERT S/RS,USED BY PSEUDO.
    2-  1 LIST
    3-  1 CNVERT
```

```
EXAMPLE MACRO VR05A 01-JAN-72 02:20 PAGE 1


1
2                      .TITLE EXAMPLE
3                      .SBTTL LIST AND CNVERT S/RS,USED BY PSEUDO.
4
5                      .MCALL MACROS
6 000000               PSEUDO
               000000 R0=%0
               000001 R1=%1
               000002 R2=%2
               000003 R3=%3
               000004 R4=%4
               000005 R5=%5
               000006 R6=%6
               000007 R7=%7
               000007 PC=%7
               000006 SP=%6
               177776 SW=177776
               177570 SWR=177570
               177304 MQ=177304
               177302 AC=177302
               177300 DIV=177300
               177306 MUL=177306
               000001 POSITI=1
               177777 NEGATI=-1
               177777 NSET=-1
               000000 ZERO=0
               000000 ZSET=0
               000000 CLEAR=0
               000002 NONZER=2
               000002 SET=2
               000003 CARRY=3
               000003 CSET=3
               000004 OVERFL=4
               000004 VSET=4
               000001 NCLEAR=1
               000005 VCLEAR=5
               000006 CCLEAR=6
               000007 WORD=7
               000010 BYTE=8.
                      .GLOBL BUFTST,SAVE,UNSAVE,SETUPP,CNVERT,BIOX,LIST,NL
               000012 LF=12
               000040 SPACE=40
               000015 CR=15
               000007 $00004=7
               000010 $00005=8.
               000011 $00006=9.
7
8
9
10
11
12
```

```
EXAMPLE MACRO VR05A 01-JAN-72 02:20 PAGE 2
LIST

1                       .SBTTL LIST
2                       ;DEBUG LISTING PROGRAM,ENTERED VIA MACRO "LIST".
3                       ;ENTERED WITH   R0 CONTAINING NUMBER OF ITEMS FOR LISTING
4                       ;               R1 CONTAINING ADDRESS OF FIRST ITEM
5                       ;               R2 CONTAINING LISTING CODE:-
6                       ;                  0=LIST BYTES IN OCTAL
7                       ;                  2=LIST WORDS IN OCTAL
8                       ;                  4=LIST BYTES IN BINARY
9                       ;                  6=LIST WORDS IN BINARY.
10                      ;LISTING IS TO TTY,FORMATTED IN COLUMNS.
11
12 00000                CREATE BUFFER OBUFFER,64. CHARS,TO HOLD ONE TTY LINE.
                000000 S00000=0
         00000  000100 OBUFFER:64.
         00002  100000 100000
         00004  000100 64.
                       .BLKB 64.
                000100 S00000=S00000+64.
                000001 S00001=1
                       .MEXIT
13 00106                CREATE WORDS <CHARSGENERATED> TO HOLD NUMBER OF CHARS PER TTY =C
                000000 S00000=0
                000000 S00001=0
                       .IRP Q,<CHARSGENERATED>
                       Q:0
                       S00000=S00000+2
                       .ENDM
         00106  000000 CHARSGENERATED:0
                000002 S00000=S00000+2
                       .MEXIT
14
15              000000 LISTEND=R0        ;ADDRESS OF LAST ITEM.
16              000001 ITEMPOINTER=R1    ;ADDRESS OF ITEM CURRENTLY BEING LISTED.
17              000003 COLUMNS=R3        ;NUMBER OF COLUMNS LEFT IN CURRENT TTY LINE.
18              000002 OPCODE=R2         ;OPERATION CODE.
19              000004 CHARCOUNT=R4      ;COUNT OF NUMBER OF CHARS PUT IN OBUFFER.
20              000005 BUFFPOINTER=R5    ;OBUFFER POINTER.
21
22              000011 TAB=11
23              000002 WORDBIT=2         ;THIS BIT IS SET IN OPCODE FOR WORD OPERATIONS.
24
25                      ;NUMBER OF CHARACTERS PER COLUMN,AND NUMBER OF COLUMNS PER
26                      ;LINE,AS A FUNCTION OF OPCODE:-
27 00110           003 WDSIZE: .BYTE 3
28 00111           010 TTCOLS: .BYTE 8.
29 00112           006         .BYTE 6
30 00113           010         .BYTE 8.
31 00114           010         .BYTE 8.
32 00115           004         .BYTE 4
33 00116           020         .BYTE 16.
34 00117           003         .BYTE 3
35
36                      ;DESPATCH VECTORS FOR CONVERSION ROUTINES:-
37 00120 000130'CONVERSIONS:     CONOB
38 00122 000162'                 CONOW
39 00124 000212'                 CONBB
40 00126 000246'                 CONBW
```

```
EXAMPLE MACRO VR05A 01-JAN-72 02:20 PAGE 2+
LIST

41
42                    ;CONVERSION ROUTINES:-
43 00130              CONVB:CONVERT BYTE @ITEMPOINTER TO ASCII OCT AT BUFFPOINTER
   00130              P BUFFPOINTER
   00130              K BYTE
   00130              M OCT
   00130              SAVE
   00130              DO SAVE
   00130 004767 JSR PC,SAVE
         000000G
                      .MEXIT
   00134              STACK BUFFPOINTER
         000000 S10000=0
                      .IRP Q,<BUFFPOINTER>
                      MOV Q,-(SP)
                      S10000=S10000+2
                      .ENDM
   00134 010546 MOV BUFFPOINTER,-(SP)
         000002 S10000=S10000+2
   00136 111101 MOVB @ITEMPOINTER,R1
   00140 042701 BIC #177400,R1
         177400
   00144              G OCT
   00144 005002 CLR R2
   00146              UNSTACK R0
                      .IRP Q,<R0>
                      MOV (SP)+,Q
                      .ENDM
   00146 012600 MOV (SP)+,R0
   00150              DO CNVERT
   00150 004767 JSR PC,CNVERT
         000310
                      .MEXIT
   00154              UNSAVE
   00154              DO UNSAVE
   00154 004767 JSR PC,UNSAVE
         000000G
                      .MEXIT
44 00160                      EXIT
   00160 000207 RTS PC
45 00162              CONVW:CONVERT WORD @ITEMPOINTER TO ASCII OCT AT BUFFPOINTER
   00162              P BUFFPOINTER
   00162              K WORD
   00162              M OCT
   00162              SAVE
   00162              DO SAVE
   00162 004767 JSR PC,SAVE
         000000G
                      .MEXIT
   00166              STACK BUFFPOINTER
         000000 S10000=0
                      .IRP Q,<BUFFPOINTER>
                      MOV Q,-(SP)
                      S10000=S10000+2
                      .ENDM
   00166 010546 MOV BUFFPOINTER,-(SP)
         000002 S10000=S10000+2
```

```
EXAMPLE MACRO V005A 01-JAN-72 02:26 PAGE 24
LIST

   00170 011101 MOV @ITEMPOINTER,R1
   00172        F OCT
   00172 012702 MOV #2,R2
         000002
   00176        UNSTACK R0
                .IRP Q,<R0>
                MOV (SP)+,Q
                .ENDM
   00176 012600 MOV (SP)+,R0
   00200        DO CNVERT
   00200 004767 JSR PC,CNVERT
         000260
                .MEXIT
   00204        UNSAVE
   00204        DO UNSAVE
   00204 004767 JSR PC,UNSAVE
         000000G
                .MEXIT
46 00210                  EXIT
   00210 000207 RTS PC
47 00212        CONBRICONVERT BYTE @ITEMPOINTER TO ASCII BIN AT BUFFPOINTER
   00212        P BUFFPOINTER
   00212        R BYTE
   00212        M BIN
   00212        SAVE
   00212        DO SAVE
   00212 004767 JSR PC,SAVE
         000000G
                .MEXIT
   00216        STACK BUFFPOINTER
         000000 S10000=0
                .IRP Q,<BUFFPOINTER>
                MOV Q,-(SP)
                S10000=S10000+2
                .ENDM
   00216 010546 MOV BUFFPOINTER,-(SP)
         000002 S10000=S10000+2
   00220 111101 MOVB @ITEMPOINTER,R1
   00222 042701 BIC #177400,R1
         177400
   00226        G BIN
   00226 012702 MOV #4,R2
         000004
   00232        UNSTACK R0
                .IRP Q,<R0>
                MOV (SP)+,Q
                .ENDM
   00232 012600 MOV (SP)+,R0
   00234        DO CNVERT
   00234 004767 JSR PC,CNVERT
         000224
                .MEXIT
   00240        UNSAVE
   00240        DO UNSAVE
   00240 004767 JSR PC,UNSAVE
         000000G
                .MEXIT
```

```
EXAMPLE MACRO V005A 01-JAN-72 02:20 PAGE 2+
LIST

48 00244                        EXIT
   00244 000207 RTS PC
49 00246        CONV:;CONVERT WORD @ITEMPOINTER TO ASCII BIN AT BUFFPOINTER
   00246        P BUFFPOINTER
   00246        R WORD
   00246        R BIN
   00246        SAVE
   00246        DO SAVE
   00246 004767 JSR PC,SAVE
         0000006
                .MEXIT
   00252        STACK BUFFPOINTER
         000000 S10000=0
                .IRP 0,<BUFFPOINTER>
                MOV 0,-(SP)
                S10000=S10000+2
                .ENDM
   00252 010546 MOV BUFFPOINTER,-(SP)
         000002 S10000=S10000+2
   00254 011101 MOV @ITEMPOINTER,R1
   00256        F BIN
   00256 012702 MOV #6,R2
         000006
   00262        UNSTACK R0
                .IRP 0,<R0>
                MOV (SP)+,0
                .ENDM
   00262 012600 MOV (SP)+,R0
   00264        DO CNVERT
   00264 004767 JSR PC,CNVERT
         000174
                .MEXIT
   00270        UNSAVE
   00270        DO UNSAVE
   00270 004767 JSR PC,UNSAVE
         0000006
                .MEXIT
50 00274                        EXIT
   00274 000207 RTS PC
51
52 00276        LIST1
53 00276 060100 ADD R1,R0                   ;FORM LISTEND,
54 00300 116767 MOVB RDSIZE(OPCODE),CHARSGENERATED
         000110'
         177600
55 00306        TYPE NL
   00306        DO NL
   00306 004767 JSR PC,NL
         0000006
                .MEXIT
56 00312        READY DBUFFER FOR FASCII OUTPUT TO TTY
   00312 005067 CLR DBUFFER+2
         177464
                .MEXIT
57 00316        LOOP
         000012 S00004=S00004+3
   00316        SFORM1 \S00004
```

```
EXAMPLE MACRO VR05A 01-JAN-72 02:20 PAGE 2+
LIST

        000316'S12=.
                .MEXIT
58 00316                        POINT BUFFPOINTER AT DBUFFER DATA
   00316 012705 MOV #DBUFFER+6,BUFFPOINTER
        000006'
                .MEXIT
59 00322 116203                 MOVB TTCOLS(OPCODE),COLUMNS
        000111'
60 00326 005004                 CLR CHARCOUNT
61 00330                        LOOP
        000015 S00004=S00004+3
   00330        SFORM1 \S00004
        000330'S15=.
                .MEXIT
62 00330                                    DO @CONVERSIONS(OPCODE)
   00330 004772 JSR PC,@CONVERSIONS(OPCODE)
        000120'
                .MEXIT
63 00334 032702                             BIT #WORDBIT,OPCODE
        000002
64 00340                                    IF SET THEN <ADD #2,ITEMPOINTER> ELSE <INC ITEMP
   00340        B BNE,BEQ THEN <ADD #2,ITEMPOINTER> ELSE <INC ITEMPOINTER>
   00340        U56
        000013 S00005=S00005+3
        000014 S00006=S00006+3
   00340        SFORM1 \S00005
        000340'S13=.
        000342'.=.+2
   00342 005201 INC ITEMPOINTER
        000346'.=.+2
   00346        SFORM1 \S00006
        000346'S14=.
   00346        SFORM2 \S00005
        000340'.=S13
   00340        SFORM3 BNE \S00006
   00340 001002 BNE S14
        000346'.=S14
   00346 062701 ADD #2,ITEMPOINTER
        000002
   00352        SFORM1 \S00005
        000352'S13=.
   00352        SFORM2 \S00006
        000346'.=S14
        000344'.=.-2
   00344        SFORM3 BR \S00005
   00344 000402 BR S13
        000352'.=S13
   00352        O56
        000011 S00006=S00006-3
        000010 S00005=S00005-3
                .MEXIT
                .MEXIT
65 00352 066704                             ADD CHARSGENERATED,CHARCOUNT
        177530
66 00356 066705                             ADD CHARSGENERATED,BUFFPOINTER
        177524
67 00362 005303                             DEC COLUMNS
```

```
EXAMPLE MACRO V005A 01-JAN-72 02:20 PAGE 2+
LIST

68 00364                           IF ZERO,BRANCH TO LINETERMINATION
   00364        B BEQ,BNE BRANCH <TO> LINETERMINATION <>
   00364 001405 BEQ LINETERMINATION
                .MEXIT
                .MEXIT
69 00366 112725                        MOVB #TAB,(BUFFPOINTER)+
         000011
70 00372 005204                        INC CHARCOUNT
71 00374                       LOOP IF WORD ITEMPOINTER (= LISTEND
   00374        P LISTEND
   00374        K WORD
   00374        J1 WORD ITEMPOINTER (= LISTEND \S00004
   00374        IF WORD ITEMPOINTER (= LISTEND BRANCH TO S15
   00374        K WORD
   00374 020100 CMP ITEMPOINTER,LISTEND
   00376        B BLOS,BHI BRANCH <TO> S15 <>
   00376 101754 BLOS S15
                .MEXIT
                .MEXIT
                .MEXIT
         000012 S00004=S00004-3
72
73 00400                       LINETERMINATION:
74 00400 112725                MOVB #CR,(BUFFPOINTER)+
         000015
75 00404 112725                MOVB #LF,(BUFFPOINTER)+
         000012
76 00410 062704                ADD #2,CHARCOUNT
         000002
77 00414                       SET OBUFFER COUNT TO CHARCOUNT
   00414        P CHARCOUNT
   00414 010467 MOV CHARCOUNT,OBUFFER+4
         177364
                .MEXIT
78 00420                       OUTPUT OBUFFER TO TTY
                .MCALL IO
   00420        IO
         000000 KBD=0
         000001 TTY=1
         000003 LSR=3
         000005 HSR=5
         000004 LSP=4
         000006 HSP=6
   00420        DO @IOX
   00420 004767 JSR PC,@IOX
         000006
                .MEXIT
   00424 000000'OBUFFER
   00426    012 .BYTE 12,TTY
   00427    001
                .MEXIT
79 00430                       TEST OBUFFER TRANSFER DONE
   00430 105767 TSTB OBUFFER+3
         177347
   00434 100375 BPL .-4
80 00436        LOOP IF WORD ITEMPOINTER (= LISTEND
   00436        P LISTEND
```

```
EXAMPLE MACRO VR05A 01-JAN-72 02120 PAGE 2+
LIST

   00436          K WORD
   00436          J1 WORD ITEMPOINTER (= LISTEND \S00004
   00436          IF WORD ITEMPOINTER (= LISTEND BRANCH TO S12
   00436          K WORD
   00436 020100 CMP ITEMPOINTER,LISTEND
   00440          B BLOS,BMI BRANCH <TO> S12 <>
   00440 101726 BLOS S12
                  .MEXIT
                  .MEXIT
                  .MEXIT
         000007 S00004=S00004-3
01 00442          EXIT
   00442 000207 RTS PC
02
EXAMPLE MACRO VR05A 01-JAN-72 02:22 PAGE 3
CNVERT

1                          .SBTTL CNVERT
2                          ;BINARY TO ASCII STRING CONVERSION,ENTERED VIA MACRO "CONVERT",
3                          ;ENTERED WITH R0 CONTAINING ADDRESS OF FIELD AT WHICH ASCII
4                          ;CHARS ARE TO BE PLACED,R1 CONTAINING BYTE OR WORD FOR CONVERSIO
5                          ;AND R2 CONTAINING OPCODE:-
6                          ;       0=CONVERT BYTE TO OCTAL STRING
7                          ;       2=........WORD................
8                          ;       4=........BYTE.....BINARY......
9                          ;       6=........WORD................
10
11                         ;CHARACTER MASKS,AS A FUNCTION OF OPCODE:-
12 00444 177770 CHMASK: 177770
13 00446 177770         177770
14 00450 177776         177776
15 00452 177776         177776
16
17                         ;NUMBER OF CHARS DEVELOPED,AS A FUNCTION OF OPCODE:-
18 00454 000003 CHARNO: 3         ;3 CHARS IN AN OCTAL BYTE.
19 00456 000006         6         ;ETC.
20 00460 000010         8.
21 00462 000020         16.
22
23       000002 OPCODE=R2
24       000005 MASK=R5
25       000003 CHCOUNT=R3
26       000000 CFIELD=R0
27       000001 OPITEM=R1
28       000004 WORKSPACE=R4
29
30 00464        CNVERT:
31 00464 016205         MOV CHMASK(OPCODE),MASK
         000444'
32 00470 016203         MOV CHARNO(OPCODE),CHCOUNT
         000454'
33 00474 060300         ADD CHCOUNT,CFIELD        ;FIELD IS FILLED "BACKWARDS".
34 00476        NEXTCHAR:
35 00476 010104         MOV OPITEM,WORKSPACE      ;GET ITEM.
36 00500 040504         BIC MASK,WORKSPACE        ;MASK IT.
37 00502 062704         ADD #60,WORKSPACE         ;CONVERT TO ASCII,
         000060
38 00506 110440         MOVB WORKSPACE,-(CFIELD);AND PUT IT IN FIELD.
39
40 00510 005303         DEC CHCOUNT
41 00512                IF NONZERO
   00512        B BNE,BEQ  <>  <>
   00512 001002 BNE .+6
                .MEXIT
                .MEXIT
42 00514                THEN BEGIN
   00514        U56
         000013 S00005=S00005+3
         000014 S00006=S00006+3
   00514        SFORM1 \S00005
         000514'S13=.
         000520'.=.+4
43 00520                          IF WORD OPCODE )= #4
   00520        K WORD
```

```
    00524 022767 CMP OPCODE,#4
          000004
    00524               B NOT(1,GE)  ..   ..
    00524 001002 B?.1 ,..
                 .MEXIT
                 .MEXIT
44 00530                               THEN BEGIN
    00530        .SB
          000016 $00005=$00025+3
          000017 $00006=$00026+3
    00526        $FORM1 \$00005
          000526'$15=,
          000532',=,+4
45 00532                                            LSHIFT OPITEM 1 PLACE R
    00532        P R
    00532 000241 CLC
    00534 006001 ROR OPITEM
          000000 .REPT 1-1
                 ASR OPITEM
                 .ENDR
                 .MEXIT
46 00536                               END
    00536        $FORM1 \$00026
          000536'$17=,
    00536        $FORM2 \$00025
          000526',=$15
    00526        $FORM3 JMP \$00026
    00526 000167 JMP $17
          000004
          000536',=$17
    00536        $SS
          000016 $00026=$00026+3
          000013 $00025=$00025+3
47 00536                               ELSE BEGIN
    00536        USB
          000016 $00025=$00005+3
          000017 $00026=$00006+3
    00536        $FORM2 \$00005
          000526',=$16
    00526        $FORM4 \$00006
    00526 000167 JMP $17+4
          000010
    00532        $FORM2 \$00006
          000536',=$17
    00536        $FORM1 \$00005
          000536'$16=,
          000542',=,+4
48 00542                                            LSHIFT OPITEM 3 PLACES R
    00542        P R
    00542 000241 CLC
    00544 006001 ROR OPITEM
          000002 .REPT 3-1
                 ASR OPITEM
                 .ENDR
    00546 006201 ASR OPITEM
    00550 006201 ASR OPITEM
                 .MEXIT
```

```
                                   END
49 00552
   00552           SFORM1 \S00006
        000552'S17=.
   00552           SFORM2 \S00005
        000536',=S16
   00536           SFORM3 JMP \S00006
   00536 000167 JMP S17
         000010
         000552',=S17
   00552           DS6
         000014 S00006=S00006-3
         000013 S00005=S00005-3
                                   BR NEXTCHAR
50 00552 000751
                             END
51 00554
   00554           SFORM1 \S00006
        000554'S14=.
   00554           SFORM2 \S00005
        000514',=S13
   00514           SFORM3 JMP \S00006
   00514 000167 JMP S14
         000034
         000554',=S14
   00554           DS6
         000011 S00006=S00006-3
         000010 S00005=S00005-3
                             ELSE BEGIN
52 00554
   00554           US6
         000013 S00005=S00005+3
         000014 S00006=S00006+3
   00554           SFORM2 \S00005
        000514',=S13
   00514           SFORM4 \S00006
   00514 000167 JMP S14+4
         000040
   00520           SFORM2 \S00006
         000554',=S14
   00554           SFORM1 \S00005
         000554'S13=.
         000560'.=.+4
                                   EXIT
53 00560
   00560 000207 RTS PC
                             END
54 00562
   00562           SFORM1 \S00006
        000562'S14=.
   00562           SFORM2 \S00005
        000554',=S13
   00554           SFORM3 JMP \S00006
   00554 000167 JMP S14
         000002
         000562',=S14
   00562           DS6
         000011 S00006=S00006-3
         000010 S00005=S00005-3

55
56 00562           FINISH
         000001',END
```

```
EXAMPLE MACRO VR05A 01-JAN-72 02:20 PAGE 3+
SYMBOL TABLE

AC     = 177302          BIOX   = ****** G        BUFFPO=%000005
BUFTST= ****** G         BYTE   = 000010          CARRY  = 000003
CCLEAR= 000006           CFIELD=%000000           CHARCO=%000004
CHARNO  000454R          CHARSG  000106R          CHCOUN=%000003
CHMASK  000444R          CLEAR  = 000000          CNVERT  000464RG
COLUMN=%000003           CONBB   000212R          CONBW   000246R
CONOB   000136R          CONOW   000162R          CONVER  000120R
CR     = 000015          CSET   = 000003          DBUFFE  000600R
DIV    = 177300          HSP    = 000006          HSR    = 000005
ITEMPO=%000001           KBD    = 000000          LF     = 000012
LINETE  000400R          LIST    000276RG         LISTEN=%000000
LSP    = 000004          LSR    = 000003          MASK   =%000005
MQ     = 177304          MUL    = 177306          NCLEAR= 000001
NEGATI= 177777           NEXTCH  000476R          NL     = ****** G
NONZER= 000002           NSET   = 177777          OPCODE=%000002
OPITEM=%000001           OVERFL= 000004           PC     =%000007
POSITI= 000001           R0     =%000000          R1     =%000001
R2     =%000002          R3     =%000003          R4     =%000004
R5     =%000005          R6     =%000006          R7     =%000007
SAVE   = ****** G        SET    = 000002          SETUPP= ****** G
SP     =%000006          SPACE  = 000040          SR     = 177776
SWR    = 177570          S00000= 000002           S00001= 000000
S00004= 000007           S00005= 000010           S00006= 000011
S10000= 000002           S12    = 000316R         S13    = 000554R
S14    = 000562R         S15    = 000330R         S16    = 000536R
S17    = 000552R         TAB    = 000011          TTCOLS  000111R
TTY    = 000001          UNSAVE= ****** G         VCLEAR= 000005
VSET   = 000004          WDSIZE  000110R          WORD   = 000007
WORD01= 000002           WORKSP=%000004           ZERO   = 000000
ZSET   = 000000
. ABS.   000000     000
         000562     001

ERRORS DETECTED:  0
FREE CORE:  7585. WORDS
,DT:Z<DT:X/LI:ME/NL:CND
```

```
;APPENDIX 5, PSEUDO MACRO DEFINITIONS.
;------------------------------------

.MACRO MACROS
.MCALL WITH,TYPE,CREATE,LSHIFT,DL,EL
.MCALL POINT,IF,THEN,END,ELSE,SFORM1,SFORM2,STACK,UNSTACK
.MCALL SFORM3,SFORM4,B,GET,SET,SETUP,INIT,LIST,F,G,H,US6,DS6
.MCALL LOOP,J1,J2,J3,CONVERT,PSEUDO,K,P,UN,SY,SAVE,UNSAVE,DO
.MCALL TEST,JUMP,READY,STEP,CYCLE,INPUT,OUTPUT,MUL,DIV
.MCALL RESERVE,DISCARD,FINISH,EXIT
.ENDM

.MACRO DL
.DSABL LSB
.ENDM
.MACRO EL
.ENABL LSB
.ENDM
.MACRO UN
.ERROR ;UNSPEC PARAM
.ENDM
.MACRO SY X
.ERROR ;SYNTAX!= X
.ENDM
.MACRO P X
.IF B X
UN
.ENDC
.ENDM
.MACRO K I
.IF DIF I,WORD
.IF DIF I,BYTE
.ERROR ;BYTE OR WORD?
.ENDC
.ENDC
.ENDM

.MACRO STACK X
S10000=0
.IRP Q,<X>
MOV Q,-(SP)
S10000=S10000+2
.ENDM
.ENDM
.MACRO UNSTACK X
.IRP Q,<X>
MOV (SP)+,Q
.ENDM
.ENDM
.MACRO DISCARD N A B C D E F G H
ADD #N+N,SP
.ENDM
.MACRO RESERVE N A B C D E F G H
SUB #N+N,SP
.ENDM

.MACRO FINISH X
.IF NE S00005-8.
.ERROR ;END!!
.ENDC
.IF NE S00004-7
.ERROR ;LOOP!!
.ENDC
.END X
.ENDM

.MACRO EXIT
RTS PC
.ENDM
```

```
.MACRO PSEUDO
R0=%0
R1=%1
R2=%2
R3=%3
R4=%4
R5=%5
R6=%6
R7=%7
PC=%7
SP=%6
SW=177776
SWR=177570
MQ=177304
AC=177302
DIV=177300
MUL=177306
POSITI=1
NEGATI=-1
NSET=-1
ZERO=0
ZSET=0
CLEAR=0
NONZER=2
SET=2
CARRY=3
CSET=3
OVERFL=4
VSET=4
NCLEAR=1
VCLEAR=5
CCLEAR=6
WORD=7
BYTE=6.
.GLOBL BUFTST,SAVE,UNSAVE,SETUPP,CNVERT,BIOX,LIST,NL
LF=12
SPACE=40
CR=15
S00004=7
S00005=8.
S00006=9.
.ENDM

.MACRO DO P X
.IF B <X>
JSR PC,P
.MEXIT
.ENDC
STACK <X>
JSR PC,P
ADD #S10000,SP
.ENDM

.MACRO LSHIFT X Y M N
P N
.IF IDN N,R
CLC
ROR X
.REPT Y-1
ASR X
.ENDR
.MEXIT
.ENDC
.IF IDN N,L
.REPT Y
ASL X
.ENDR
.MEXIT
.ENDC
SY N
.ENDM
```

```
.MACRO CONVERT I X TO A T A Z
P Z
K I
H T
SAVE
STACK Z
.IF IDN I,WORD
MOV X,R1
F T
.ENDC
.IF IDN I,BYTE
MOVB X,R1
BIC #177400,R1
G T
.ENDC
UNSTACK R0
DO CNVERT
UNSAVE
.ENDM

.MACRO LIST N I FR A W T
P T
H T
.IF DIF I,BYTES
.IF DIF I,WORDS
SY I
.MEXIT
.ENDC
.ENDC
SAVE
STACK N
DEC (SP)
MOV A,R1
.IF IDN I,WORDS
F T
ASL (SP)
.ENDC
.IF IDN I,BYTES
G T
.ENDC
UNSTACK R0
DO LIST
UNSAVE
.ENDM

.MACRO F T
.IIF IDN T,OCT,MOV #2,R2
.IIF IDN T,BIN,MOV #6,R2
.ENDM
.MACRO G T
.IIF IDN T,OCT,CLR R2
.IIF IDN T,BIN,MOV #4,R2
.ENDM
.MACRO H T
.IF DIF T,BIN
.IF DIF T,OCT
SY T
.ENDC
.ENDC
.ENDM
```

```
.MACRO MUL A Q B AN IN C D
.IF IDN A,BY
MOV Q,@#MUL
.IF NB B
MOV @#MQ,IN
.IF NB C
MOV @#AC,C
.ENDC
.ENDC
.MEXIT
.ENDC
.IF DIF A,BY
MOV A,@#MQ
MOV B,@#MUL
.IF NB AN
MOV @#MQ,C
.IF NB D
MOV @#AC,D
.ENDC
.ENDC
.ENDC
.ENDM

.MACRO DIV A Q X C AN INA D R IN E
.IF IDN A,BY
MOV Q,@#DIV
.IF NB X
.IF NB INA
MOV @#AC,R
.ENDC
MOV @#MQ,AN
.ENDC
.MEXIT
.ENDC
.IF DIF A,BY
MOV A,@#MQ
.IF IDN Q,BY
MOV X,@#DIV
.IF NB C
.IF NB IN
MOV @#AC,IN
.ENDC
MOV @#MQ,INA
.ENDC
.MEXIT
.ENDC
MOV Q,@#AC
MOV C,@#DIV
.IF NB AN
.IF NB E
MOV @#AC,E
.ENDC
MOV @#MQ,D
.ENDC
.ENDC
.ENDM
```

```
.MACRO GET X Y IN Z
P Z
.IF IDN Y,BLOCKSIZE
MOV X-4,Z
.MEXIT
.ENDC
.IF IDN Y,STATUS
MOVB X+3,Z
.MEXIT
.ENDC
.IF IDN Y,MODE
MOVB X+2,Z
.MEXIT
.ENDC
.IF IDN Y,COUNT
MOV X+4,Z
.MEXIT
.ENDC
$Y Y
.ENDM

.MACRO SET X Y TO Z
P Z
.IF IDN Y,IP
MOV Z,X-10
.MEXIT
.ENDC
.IF IDN Y,OP
MOV Z,X-6
.MEXIT
.ENDC
.IF IDN Y,COUNT
MOV Z,X+4
.MEXIT
.ENDC
$Y Y
.ENDM

.MACRO SETUP
DO SETUPP
.ENDM

.MACRO INIT Z T X A Y
P Y
DO BIOX
Y
.BYTE 1,Z
.ENDM

.MACRO SAVE X
DO SAVE
.ENDM

.MACRO UNSAVE X
DO UNSAVE
.ENDM
```

```
.MACRO IF I X R Y T P E Q
.IF EQ I
B BEQ,BNE X <R> Y <T>
.MEXIT
.ENDC
.IF EQ I-1
B BPL,BMI X <R> Y <T>
.MEXIT
.ENDC
.IF EQ I+1
B BMI,BPL X <R> Y <T>
.MEXIT
.ENDC
.IF EQ I-2
B BNE,BEQ X <R> Y <T>
.MEXIT
.ENDC
.IF EQ I-3
B BCS,BCC X <R> Y <T>
.MEXIT
.ENDC
.IF EQ I-4
B BVS,BVC X <R> Y <T>
.MEXIT
.ENDC
.IF EQ I-5
B BVC,BVS X <R> Y <T>
.MEXIT
.ENDC
.IF EQ I-6
B BCC,BCS X <R> Y <T>
.MEXIT
.ENDC
.IF NB <R>
K I
.IIF IDN I,BYTE,CMPB X,Y
.IIF IDN I,WORD,CMP X,Y
.IF IDN <R>,=
B BEQ,BNE T <P> E <Q>
.MEXIT
.ENDC
.IF IDN <R>,)(
B BNE,BEQ T <P> E <Q>
.MEXIT
.ENDC
.IF IDN <R>,)
B BHI,BLOS T <P> E <Q>
.MEXIT
.ENDC
.IF IDN <R>,(
B BLO,BHIS T <P> E <Q>
.MEXIT
.ENDC
.IF IDN <R>,)=
B BHIS,BLO T <P> E <Q>
.MEXIT
.ENDC
```

```
.IF IDN <R>,(=
B BLOS,BHI T <P> E <Q>
.MEXIT
.ENDC
.IF IDN <R>,S)
B BGT,BLE T <P> E <Q>
.MEXIT
.ENDC
.IF IDN <R>,S(
B BLT,BGE T <P> E <Q>
.MEXIT
.ENDC
.IF IDN <R>,S)=
B BGE,BLT T <P> E <Q>
.MEXIT
.ENDC
.IF IDN <R>,S(=
B BLE,BGT T <P> E <Q>
.MEXIT
.ENDC
SY <R>
.ENDC
.ENDM

.MACRO U56
S00005=S00005+3
S00006=S00006+3
.ENDM

.MACRO D56
S00006=S00006-3
S00005=S00005-3
.ENDM

.MACRO B ABR BBR X1 X2 X3 X4
.IF B X1
ABR .+6
.MEXIT
.ENDC
.IF NB X1
.IF IDN X1,BRANCH
.IF B X3
SY ?
.MEXIT
.ENDC
ABR X3
.MEXIT
.ENDC
.IF IDN X1,JUMP
.IF B X3
SY ?
.MEXIT
.ENDC
BBR .+6
JMP X3
.MEXIT
.ENDC
.IF IDN X1,THEN
.IF B <X2>
SY ?
```

```
  .MEXIT
  .ENDC
  .IF NB X3
  .IF IDN X3,ELSE
  .IF B <X4>
  SY ?
  .MEXIT
  .ENDC
  U56
  $FORM1 \$00005
  .=.+2
  X4
  .=.+2
  $FORM1 \$00006
  $FORM2 \$00005
  $FORM3 ABR \$00006
  X2
  $FORM1 \$00005
  $FORM2 \$00006
  .=.-2
  $FORM3 BR \$00005
  U56
  .MEXIT
  .ENDC
  .ENDC
 U56
 $FORM1 \$00005
 .=.+2
 X2
 $FORM1 \$00006
 $FORM2 \$00005
 $FORM3 BBR \$00006
 D56
 .MEXIT
 .ENDC
 .ENDC
 .ENDM

 .MACRO THEN BGN X
 .IF NB BGN
 .IF DIF BGN,BEGIN
 SY BGN
 .MEXIT
 .ENDC
 .ENDC
 .IF B BGN
 .PRINT ;** BEGIN;
 .ENDC
 .IF NB X
 .IF DIF X,;
SY X
.MEXIT
.ENDC
.ENDC
U56
$FORM1 \$00005
.=.+4
.ENDM
```

```
.MACRO SFORM1 S00005
S'S00005=.
.ENDM

.MACRO END X Y
.IF NB X
.IF DIF X,?
.IF DIF X,;
SY X
.MEXIT
.ENDC
.ENDC
.IF IDN X,?
.IF NB Y
.IF DIF Y,;
SY Y
.MEXIT
.ENDC
.ENDC
.ENDC
.ENDC
.IF LE S00005-8.
.ERROR ;TOO MANY ENDS!
.MEXIT
.ENDC
SFORM1 \S00006
SFORM2 \S00005
SFORM3 JMP \S00006
OS6
.IF NB X
.IF IDN X,?
.IF NE S00005-8.
.ERROR ;END MISSING
END ?
.ENDC
.ENDC
.ENDC
.ENDM

.MACRO SFORM2 S00005
.=S'S00005
.ENDM

.MACRO SFORM3 BX SX
BX S'SX
.=S'SX
.ENDM

.MACRO ELSE BGN X
.IF NB BGN
.IF DIF BGN,BEGIN
.ERROR ;ELSE BGN ?
.MEXIT
.ENDC
.ENDC
.IF B BGN
.PRINT ;**BEGIN!
.ENDC
.IF NB X
.IF DIF X,;
SY X
.MEXIT
.ENDC
.ENDC
US6
SFORM2 \S00005
SFORM4 \S00006
SFORM2 \S00006
SFORM1 \S00005
.=.+4
.ENDM
```

```
.MACRO SFORM4 $0RR05
JMP $'$00005+4
.ENDM

.MACRO CYCLE M Q R S T ?L
P T
EL
.IF IDN R,ON
.IF IDN M,IP
CMP T-10,T-2
BLO L
SUB T-12,T-10
L:ADD T-4,T-10
DL
.MEXIT
.ENDC
.IF IDN M,OP
CMP T-6,T-2
BLO L
SUB T-12,T-6
L:ADD T-4,T-6
DL
.MEXIT
.ENDC
CMP M,T-2
BLO L
SUB T-12,M
L:ADD T-4,M
DL
.MEXIT
.ENDC
.IF IDN R,BACK
.IF IDN M,IP
SUB T-4,T-10
CMP T-10,#T
BHIS L
ADD T-12,T-10
L:
DL
.MEXIT
.ENDC
.IF IDN M,OP
SUB T-4,T-6
CMP T-6,#T
BHIS L
ADD T-12,T-6
L:
DL
.MEXIT
.ENDC
SUB T-4,M
CMP M,#T
BHIS L
ADD T-12,M
L:
DL
.MEXIT
.ENDC
SY R
DL
.ENDM
```

```
.MACRO POINT P TO Q R S T
.IF IUN Q,END
MOV S-2,P
ADD S-4,P
.MEXIT
.ENUC
.IF IDN R,BLOCK
.IF IUN Q,FIRST
MOV #T,P
.MEXIT
.ENDC
.IF IDN Q,LAST
MOV T-2,P
.MEXIT
.ENDC
.IF IDN Q,IP
MOV T-10,P
.MEXIT
.ENUC
.IF IDN Q,OP
MOV T-6,P
.MEXIT
.ENDC
SY Q
.ENDC
.IF IUN R,DATA
MOV #W+6,P
.IF NB S
.IF DIF S,END
SY S
.MEXIT
.ENDC
ADD Q+4,P
.ENDC
.MEXIT
.ENUC
SY ?
.ENUM

.MACRO STEP A B C D E
P E
.IF IUN C,ON
.IF IUN A,IP
ADD E-4,E-10
.MEXIT
.ENUC
.IF IDN A,OP
ADD E-4,E-6
.MEXIT
.ENUC
AUD E-4,A
.MEXIT
.ENDC
.IF IUN C,BACK
.IF IDN A,IP
SUB E-4,E-10
.MEXIT
.ENUC
.IF IUN A,OP
SUB E-4,E-6
.MEXIT
.ENDC
SUB E-4,A
.MEXIT
.ENDC
SY C
.ENDM
```

```
.MACRO LOOP A I X R Y
.IF H A
S00004=S00004+3
SFORM1 \S00004
.MEXIT
.ENDC
.IF NH A
.IF LT S00004-10,
BY <LOOP>
.ENDC
.IF IDN I,TIMES
DEC A
J2 \S00004
S00004=S00004-3
.MEXIT
.ENDC
.IF DIF A,IF
BY <A>
.MEXIT
.ENDC
.IF H X
J3 I \S00004
S00004=S00004-3
.MEXIT
.ENDC
P Y
K I
J1 I X R Y \S00004
S00004=S00004-3
.ENDC
.ENDM

.MACRO J1 I X R Y S4
.IF LT .-S'S4-246,
IF I X R Y BRANCH TO S'S4
.MEXIT
.ENDC
IF I X R Y JUMP TO S'S4
.ENDM

.MACRO J2 S4
BNE S'S4
.ENDM

.MACRO J3 I S4
.IF LT .-S'S4-246,
IF I BRANCH TO S'S4
.MEXIT
.ENDC
IF I JUMP TO S'S4
.ENDM
```

```
.MACRO CREATE I X N U M P V W S T Y Z
$00000=0
.IF IDN <I>,LIST
.IF B <M>
UN
.MEXIT
.ENDC
.IF IDN <U>,WORDS
N+N
X
X
M+M
X+N+N-<M>-<M>
X:.BLKW N
$00000=$00000+N+N
$00001=0
.MEXIT
.ENDC
.IF IDN <U>,BYTES
N
X
X
M
X+N-<M>
X:.BLKB N
$00000=$00000+N
$00001=1
.MEXIT
.ENDC
BY Y
.MEXIT
.ENDC
.IF IDN <I>,BUFFER
.IF B N
UN
.MEXIT
.ENDC
X:N
100000
N
.BLKW N
$00000=$00000+N
$00001=1
.MEXIT
.ENDC
.IF IDN <I>,WORDS
$00001=0
.IRP Q,<X>
Q:0
$00000=$00000+2
.ENDM
.MEXIT
.ENDC
.IF IDN <I>,BYTES
$00001=1
.IRP Q,<X>
Q:.BYTE 0
$00000=$00000+1
.ENDM
.MEXIT
.ENDC
BY <I>
.ENDM
```

```
.MACRO WITH D X
.*.-S00000
S00002=0
.IF EQ S00001
.IRP Q,<X>
Q
S00002=S00002+2
.ENUM
.ENUC
.IF NE S00001
.IRP Q,<X>
.BYTE Q
S00002=S00002+1
.ENDM
.ENDC
S00000=S00000-S00002
.*.+S00000
.ENDM

.MACHO TYPE M,N D ?L1 ?L2 ?L3 ?L4
.IF DIF <M>,NL
EL
DO BIOX
L1
.IF B D
.BYTE 12,1
.ENDC
.IF NB D
.BYTE 14,1
D
.ENDC
BR L4
L1:0
0
L3:L2-L3-2
.ASCII "M"
.BYTE CR,LF
L2:.EVEN
L4:
DL
.MEXIT
.ENDC
DO NL
.ENDM
```

```
.MACRO READY A F M P Q R S
.IF B M
BY MODF
.MEXIT
.ENDC
.IF IDN M,ASCII
MOV #2,A+2
.MEXIT
.ENDC
.IF IDN M,FASCII
CLR A+2
.MEXIT
.ENDC
.IF IDN M,BIN
MOV #3,A+2
.MEXIT
.ENDC
.IF IDN M,FBIN
MOV #1,A+2
.MEXIT
.ENDC
SY M
.ENDM

.MACRO OUTPUT A T D N C
.IF B D
.ERROR ;DEVICE?
.MEXIT
.ENDC
.MCALL IO
IO
DO BIOX
A
.IF B C
.BYTE 12,D
.MEXIT
.ENDC
.BYTE 14,D
C
.ENDM

.MACRO INPUT A T D N C
.IF B D
.ERROR ;BUFFER?
.MEXIT
.ENDC
.MCALL IO
IO
DO BIOX
D
.IF B C
.BYTE 11,A
.MEXIT
.ENDC
.BYTE 13,A
C
.ENDM
```

```
.MACRO TEST B P Q R S
.IF NB P
.IF IDN P,ERRORS
STACK #B+3
JSR R5,BUFTST
UNSTACK R0
ADD #.+12,(SP)
MOV @(SP),(SP)
JMP @(SP)+
.+14
.+12
.+10
.+6
.+4
.+2
.MEXIT
.ENDC
.ENDC
TSTB B+3
BPL .-4
.ENDM

.MACRO JUMP T L I E X
.IF IDN E,EOM
.=.-10.
L
.=.+8.
.MEXIT
.ENDC
.IF IDN E,EOF
.=.-8.
L
.=.+6
.MEXIT
.ENDC
.IF IDN E,TRUNC
.=.-6
L
.=.+4
.MEXIT
.ENDC
.IF IDN E,MODE
.=.-4
L
.=.+2
.MEXIT
.ENDC
.IF IDN E,CHKSUM
.=.-2
L
.MEXIT
.ENDC
SY E
.ENDM

.MACRO IO
KBD=0
TTY=1
LSR=3
MSR=5
LSP=4
MSP=6
.ENDM
```