A VAX Assembler

Paul R. Potts

December 1986

This is one of the oldest examples of my programming that I still have. I wrote it for a college class on assembly-language programming.

The assignment was to write an assembler, in VAX 11/750 assembly language, for a subset of VAX 11/750 assembly language. I have mercifully forgotten some of the details. I think I preserved this code by downloading it to a floppy disc using a terminal program on a Macintosh, perhaps using Kermit? My memory is a little hazy.

The macros to access the file system were provided for us, by a teaching assistant named Elvis, hence the reference to "Elvis" in the comments.

Not having a VAX 11/750 on hand, I have not tested this code. I'm also not sure I still have the original input data file I used to test it. If you want to experiment with an emulator or something, go for it. I hereby place this code in the public domain.

I remember with a mix of horror and fondness the alternate feelings of exhaustion and elation I felt during the many all-nighters I spent in The College of Wooster's Taylor Hall, in my sophomore year, writing and debugging this code, spreading out yards of tractor-feed paper printouts across the floor and crawling up and down the code like a spider, trying to get the "big picture" before going back to my terminal and staring at it with bleary eyes, marking up the code, refueling myself occasionally with soda and subs from Pizza Express, aka "Pizza Distress," working next to- my classmates and friends John, Ken, and Bill, who were each working through this programming boot camp/hazing experience in their own way.

I'm grateful for the experience, and this was one of the programs I wrote that helped "make my bones" as a programmer. It didn't do anything good for my eyes or my back or my wrists, though. So I'm also grateful I every day that I wake up and *don't* have an assignment due for CS253!

Note that the web version of the code is a bit ugly, because my web page template imposes a strict width restriction on preformatted text, which results in the ends of some comments getting cut off (although you can scroll right to read them). Until I can fix that, I recommend reading the PDF version of this file.

```
; Original VAX filename: assembler.mar.paul;
; Original modification date: 18 Dec 1986
;
; X253 ASSEMBLER
; Version 3.1415926535
; By Paul Potts
;
; Notes added in 2025:
;
; The original version of this file contained tab characters and was written
; assuming 8 character tab stops. In this version of the file, I have converted
; all the tabs to the appropriate numbers of spaces using BBEdit's "Convert
; Spaces to Tabs" feature. This might render this version unreadable by the
; actual VAX assembler program it was written for; I'm not sure.
;
; I have made some additional minor fixes:
; - cleaned up spaces that should have been tabs
; - removed trailing whitespace
; - fixed a line that exceeded 80 characters
; - made the horizontal bars all 78 characters
; - made subroutine names in header comments all capitalized like This,
;   not THIS or this.
;
; Note that comments are sometimes aligned inconsistently. This is because
; the DEC VT220 terminal had an 80-column display, so it was important to
; keep lines of text under this limit.
; ============================================================================

;Pass one of the assembler

passnum:        .blkb   1               ;which pass are we on?
pass1er:        .blkb   1               ;was there an error in pass 1?
pass2er:        .blkb   1               ;was there an error in pass 2?
crlf:           .byte   13, 10          ;character codes for CR and LF
blank:          .byte   ^a/ /           ;blank character
lc:             .blkw   1               ;a word is all that is necessary
oldlc:          .blkw   1               ;lc from the previous line
linenum:        .blkb   1               ;the line number (maximum 255)
instr:          .blkl   1               ;the instruction read
operand:        .blkl   1               ;the current operand to evaluate
opcode:         .blkb   1               ;byte holds the opcode
maxoperands:    .blkb   1               ;maximum operands for this instr.
numoperands:    .blkb   1               ;number of operands for this instr.
symboltype:     .blkb   1               ;holds ascii type of symbol (a/r)
label:          .blkl   1               ;entry for symbol table
lastsymbol:     .blkl   1               ;used to hold last symbol for entry
```

```
decnum:         .blkl   1               ;decimal number stored in ascii
inform:         .blkb   1               ;internal form of a number
outnum:         .blkw   1               ;to hold hex output in ascii
errorreturn:    .blkb   1               ;set to 1 if error occurs
illegal:        .blkb   1               ;used to flag illegal operands
legal:          .blkb   1               ;flag for checkoperands
eolnflag:       .blkb   1               ;set to 1 if eoln reached (char80)
counter1:       .blkl   1               ;counter for searching
counter2:       .blkl   1               ;another counter for searching
counter3:       .blkl   1               ;and another
pointer1:       .blkl   1               ;pointers for the insertion sort
pointer2:       .blkl   1               ;another
pointer3:       .blkl   1               ;and yet another

inline:         .BYTE   ^a/ /[80]       ;the input buffer

outline:        ;the output buffer
outcode:        .blkb   18              ;locations for 7 bytes machine code
outlc:          .blkb   3               ;location counter
                .blkb   2               ;blanks
outlinenum:     .blkb   3               ;space for line number
                .blkb   3               ;blanks
outsymbol:      .blkb   3               ;the symbol (if any)
                .blkb   3               ;blanks
outinstr:       .blkb   3               ;the instruction read
                .blkb   80              ;space for the rest

symboltable:    .blkq   20              ;reserve 20 quadwords for symbols
endsymbol:      .blkq   1               ;null symbol to check for overflow

database:       .ascii  /cmp tst gtr eql lss brb bsb rsb ret /
                .ascii  /mov moa add sub .by .eq .en /

opcodetable:    .byte   ^x91, ^x95, ^x14, ^x13, ^x19, ^x11, ^x10
                .byte   ^x05, ^x04, ^x90, ^x9e, ^x80, ^x82
                .byte   ^xfd, ^xfe, ^xff

maxoperdbase:   .byte   2, 1, 1, 1, 1, 1, 1, 0, 0, 2, 2, 2, 2, 6, 1, 1

machine:        .blkb   500             ;the storage area for machine code
entry:          .blkw   1               ;the entry point for machine code
                                        ;it is an offset

message1:       .ascii  /WARNING: duplicate symbol found in pass one!!     /
message2:       .ascii  /ERROR: An illegal operand was in the line below!! /
message3:       .ascii  /ERROR: An illegal instruction in the line below!! /
```

```
message4:       .ascii  /ERROR: Reference to an undefined symbol below!!   /
message5:       .ascii  /ERROR: Decimal constant out of range below!!      /
message6:       .ascii  /ERROR: Attempted branch to constant symbol below!!/
message7:       .ascii  /- The Symbol Table -                              /
message8:       .ascii  /WARNING: Symbol Table Overflow in pass one!!      /
message9:       .ascii  /ERROR: Need a label for the .eq statement below!! /
message10:      .ascii  /ERROR: Symbol is referencing nothing below!!      /
message11:      .ascii  /ERROR: Error occured in pass 1, will not run.     /
message12:      .ascii  /ERROR: Error occured in pass 2, will not run.     /
message13:      .ascii  /Code in memory now executing. . .                 /
message14:      .ascii  /Code in memory has returned control legally! Done./
message15:      .ascii  /INFO: line below doesn't display all code produced/
message16:      .ascii  /ERROR!  Entry point following .en is illegal!     /
message17:      .ascii  /ERROR!  Too many operands for instruction below!  /

                .blkb   1

;===============================================================================

.MACRO          FILE$SERVER
                JMP             FR$START
FILE$PUT:       $RAB_STORE      RAB=FR$RAB_OUT$, RBF=(R5), RSZ=R6
                $PUT            RAB=FR$RAB_OUT$
                RSB
FILE$GET:       $RAB_STORE      RAB=FR$RAB_IN$, UBF=(R5), USZ=R6
                $GET            RAB=FR$RAB_IN$
                RSB
FR$ERR_OPEN:    $RAB_STORE      RAB=FR$OUTRAB, RBF=FR$WARNING1, RSZ=FR$LEN_P
                $PUT            RAB=FR$OUTRAB
                JMP             FC$NMASK
FR$WARNING1:    .ASCII          /?? File not found <retry>:/
                .BYTE           10,13
FR$PRMPT_OUT:   .ASCII          /Enter the output file name: /
FR$PRMPT_IN:    .ASCII          /Enter the input  file name: /
FR$NAM_OUT$:    .BYTE           ^A/ /[16]
FR$NAM_IN$:     .BYTE           ^A/ /[16]
FR$LEN_P:       .LONG           28
FR$LEN_I:       .LONG           16
FR$OUTFAB:      $FAB            FAC = PUT, FNM = <SYS$OUTPUT>
FR$OUTRAB:      $RAB            FAB = FR$OUTFAB
FR$INFAB:       $FAB            FAC = GET, FNM = <SYS$INPUT>
FR$INRAB:       $RAB            FAB = FR$INFAB
FR$FAB_OUT$:    $FAB    FAC=PUT, FNA=FR$NAM_OUT$, FNS=16, ORG=SEQ, rat=cr
FR$RAB_OUT$:    $RAB            FAB=FR$FAB_OUT$
FR$FAB_IN$:     $FAB            FAC=GET, FNA=FR$NAM_IN$, FNS=16
FR$RAB_IN$:     $RAB            FAB=FR$FAB_IN$
```

```
FR$START:       $OPEN           FAB=FR$INFAB        ;SYS INPUT
                $CONNECT        RAB=FR$INRAB
                $OPEN           FAB=FR$OUTFAB       ;SYS OUTPUT
                $CONNECT        RAB=FR$OUTRAB
.ENDM           FILE$SERVER

;*****************************************************************************

.MACRO          FILE$CREATE_FILE
                $RAB_STORE      RAB=FR$OUTRAB, RBF=FR$PRMPT_OUT, RSZ=FR$LEN_P
                $PUT            RAB=FR$OUTRAB
                $RAB_STORE      RAB=FR$INRAB, UBF=FR$NAM_OUT$, USZ=FR$LEN_I
                $GET            RAB=FR$INRAB
                $CREATE         FAB=FR$FAB_OUT$
                $CONNECT        RAB=FR$RAB_OUT$
.ENDM           FILE$CREATE_FILE

;*****************************************************************************

.MACRO          FILE$OPEN_FILE
FC$NMASK:       $RAB_STORE      RAB=FR$OUTRAB, RBF=FR$PRMPT_IN RSZ=FR$LEN_P
                $PUT            RAB=FR$OUTRAB
                $RAB_STORE      RAB=FR$INRAB, UBF=FR$NAM_IN$, USZ=FR$LEN_I
                $GET            RAB=FR$INRAB
                $OPEN           FAB=FR$FAB_IN$
                BLBS            R0,FR$ESCAPE
                JMP             FR$ERR_OPEN
FR$ESCAPE:      $CONNECT        RAB=FR$RAB_IN$
.ENDM           FILE$OPEN_FILE

;*****************************************************************************

READ:           $RAB_STORE      RAB = FR$INRAB, UBF=(R5), USZ=R6
                $GET            RAB = FR$INRAB
                RSB

WRITE:          $RAB_STORE      RAB = FR$OUTRAB, RBF=(R5), RSZ=R6
                $PUT            RAB = FR$OUTRAB
                RSB

;*****************************************************************************
;THE MAIN PROGRAM LOOP

begin:          .word
                jsb     initialize      ;set up everything
```

```
                movb    #1, passnum     ;pass number one
pass1loop:      jsb     getline         ;fetch a line
                moval   inline,counter1 ;reset counter1
                cmpb    inline, #^a/;/  ;is the whole line a comment?
                bneq    checksymbol     ;if not, go do stuff with it
                brb     pass1loop       ;cycle to next line
checksymbol:    movl    inline, label   ;take 4 characters
                jsb     scanforblank    ;move past the label
                jsb     scanforchar     ;skipblanks
                cmpb    eolnflag, #1    ;has end of line been reached?
                beql    pass1loop       ;the line was blank
                jsb     instruction     ;call the instruction subroutine
                jsb     insertsymbol    ;put symbol in symboltable
                jsb     checkoperands   ;calculate their legality
                cmpb    opcode, #^xff   ;is it ".en"
                bneq    pass1loop       ;if not, get another line

                $REWIND RAB=FR$RAB_IN$  ;reset to begining of file
                movb    #2, passnum     ;pass number two

                jsb     initialize2     ;set up stuff for pass 2
pass2loop:      jsb     getline         ;read line of code
                moval   inline,counter1 ;set up pointer
                cmpb    inline, #^a/;/  ;is the whole line a comment?
                bneq    checksymbol2    ;if not, go do stuff with it
                jsb     bigcomment      ;print the whole line
                brb     pass2loop       ;cycle to next line
checksymbol2:   movl    inline, label   ;take 4 characters
                jsb     scanforblank    ;move past the label
                jsb     scanforchar     ;skipblanks
                cmpb    eolnflag, #1    ;has end of line been reached?
                beql    pass2loop       ;the line was blank
                jsb     instruction     ;call the instruction subroutine
                jsb     dumpoperands    ;print the operands
                jsb     checkoperands   ;calculate their legality
                jsb     checknumoper    ;was there a legal num of operands
                jsb     dumpcode        ;put the machine code in outline
                jsb     dumpline        ;fill fields in line & print it
                cmpb    opcode, #^xff   ;is it ".en"
                beql    endloop         ;if so, quit program
                brw     pass2loop       ;repeat for another line
endloop:        jsb     printtable      ;print symbol table
                cmpb    pass1er, #1     ;error in pass 1?
                beql    noexecute1      ;if so, don't execute
                cmpb    pass2er, #1     ;was there an error in pass 2?
                beql    noexecute2      ;if so, don't run code
```

```
                jsb     setentry        ;set entrypoint
                cmpb    pass2er, #1     ;error in the entrypoint?
                beql    noexecute2      ;if so, don't execute
                jsb     setreturn       ;put code for return in memory
                moval   message13, r5   ;print "code now executing. . ."
                jsb     printmessage    ;go print it
                clrl    r5              ;clear out offset
                movw    entry, r5       ;entry offset
                moval   machine, r6     ;start of machine code
                addl2   r5, r6          ;calculate the address
                jsb     (r6)            ;go execute the code
                jmp     finishup        ;executed on successful return

noexecute1:     moval   message11, r5   ;message for pass1
                jsb     printmessage    ;go print it
                jmp     badfinishup     ;quit without message
noexecute2:     moval   message12, r5   ;message for bad pass2
                jsb     printmessage    ;go print it
                jmp     badfinishup     ;quit without message

;=============================================================================

;This subroutine simply initializes everything that needs it

initialize:     file$server                     ;Call the macros Elvis wrote
                file$create_file                ;Set up an input file
                file$open_file                  ;Get the filename and open it
                movc5   #0, blank, #0, #168, symboltable        ;erase it
                movb    #0, pass1er             ;default
initialize2:    movw    #0, lc                  ;location counter start at 0
                movw    #0, oldlc               ;oldlc start at zero
                movw    #0, linenum             ;first line
                movb    #0, pass2er             ;default
                rsb

;=============================================================================

;This is a generic routine for scanning through the inline.  It walks
;through until it finds a character, with counter1.  The second routine
;walks through until it finds a blank.

scanforblank:   movb    #0, eolnflag            ;reset flag
                moval   outline, r5
                subl3   counter1, r5, r6        ;see above
                cmpl    #1, r6
                beql    eoln
```

```
                movl    counter1, r7
                incl    counter1                ;next character position
                cmpb    (r7), blank
                bneq    scanforblank            ;repeat until found a blank
                subl2   #1, counter1            ;cancel last increment
                rsb                             ;return

scanforparenth: movb    #0, eolnflag            ;reset flag
                moval   outline, r5
                subl3   r10, r5, r6             ;use r10 as pointer now
                cmpl    #1, r6                  ;see above
                beql    eoln
                incl    r10
                cmpb    (r10), #^a/(/           ;find left parenth
                bneq    scanforparenth          ;loop
                rsb

eoln:           movb    #1, eolnflag            ;set flag
                rsb

scanforchar:    movb    #0, eolnflag            ;reset flag
                moval   outline, r5             ;want the address
                subl3   counter1, r5, r6        ;how many chars over?
                cmpl    #1, r6                  ;is it the 80th char?
                beql    eoln                    ;end of inline
                movl    counter1, r7            ;the address of inline+n
                incl    counter1                ;next character position
                cmpb    (r7), blank             ;found a blank?
                beql    scanforchar             ;if not, we are done
                subl2   #1, counter1            ;cancel last increment
                rsb                             ;go back

scanforcomma:   movb    #0, eolnflag            ;reset flag
                moval   outline, r5
                subl3   counter1, r5, r6        ;see above
                cmpl    #1, r6
                beql    eoln
                movl    counter1, r7
                incl    counter1                ;next character position
                cmpb    (r7), #^a/,/            ;compare to a comma
                bneq    scanforcomma
                rsb

;=============================================================================

;Subroutine: Getline
```

```
;This will read in a line from a file into inline, and echo comments to the
;screen.  That's all it does.

getline:        movb    #0, eolnflag                    ;eoln not found
                movc5   #1, blank, blank, #80, inline   ;blank inline
                movl    #^x2020, label                  ;blank label
                moval   inline,r5                       ;buffer
                movl    #80,r6                          ;set length
                jsb     file$get                        ;read the line
                movc5   #1, blank, #32, #80, outline    ;clear outline
                rsb                                     ;return

;=============================================================================

;This routine prints out an entire-line comment from infile

bigcomment:     moval   inline, r5      ;print the whole line
                movl    #80, r6         ;all 80 chars
                jsb     file$put        ;print comment line
                rsb                     ;return

;=============================================================================

;Subroutine: Setentry
;This code will take the value of the last symbol evaluated and set it
;as the entrypoint (the offset stored in entry.)
;search through the symboltable until you find the value of the last
;string, then move it into entry

setentry:       moval   lastsymbol, r10         ;search for symbol in r10
                moval   symboltable, r5         ;start of symboltable
                moval   endsymbol, r6           ;end of symboltable
eloop:          cmpl    (r5), (r10)             ;did we find symbol?
                beql    foundit                 ;branch if we did
                addl2   #8, r5                  ;increment pointer
                cmpl    r5, r6                  ;did we hit end?
                bneq    eloop                   ;if not, loop
                movb    #1, pass2er             ;set error status
                moval   message16, r5           ;illegal end entrypoint
                jsb     printmessage            ;go print it
                rsb                             ;return
foundit:        cmpb    7(r5), #^a/a/           ;is it absolute symbol
                bneq    oksymbol                ;if not, okay
                movb    #1, pass2er             ;set error status
                moval   message16, r5           ;or trigger error
                jsb     printmessage            ;go print it
```

```
                rsb                                 ;and return
oksymbol:       movw    4(r5), entry                ;the entry offset
                rsb

;=============================================================================

;Subroutine: Setreturn

;this code will move an absolute jump to finishup to the point where the
;.en occurs in the code in memory.  That way, if the code is permitted to
;run through until the .en, it will be able to complete execution without
;crashing.

setreturn:      subw2   #1, lc              ;don't ask me why it works
                moval   machine, r5         ;address of start of machine code
                clrl    r6                  ;clear out temporary storage
                movw    lc, r6              ;this is the offset
                addl2   r6, r5              ;this is the place to put the code
                movb    code, (r5)          ;move first four bytes
                rsb                         ;return to execute code
code:           rsb                         ;this code will be copied directly

;=============================================================================

;Subroutine: Instruction
;This subroutine searches through the database until it finds an instruction
;that matches the one read from the file or runs out of intructions.  If
;the instruction is found to be ".eq" then the symboltype is set to
;absolute.  If a symbol has not been read in an instruction error will be
;generated.
;
;Input: Nothing is passed through registers.  The routines uses the global
;variables counter1 and counter2 to do its work.
;
;Output: the opcode stored in "opcode"

instruction:    movl    counter1, r9            ;use it as an address
                movl    (r9), instr             ;move the instruction to instr
                clrl    counter2                ;counter to find maxoperands
                clrl    counter3                ;counter to find insruction
                moval   database, r8            ;address of database start
                moval   maxoperdbase, r7        ;maxoperands database
scandatabase:   mull3   #4, counter3, r6        ;4 bytes for each entry
                addl3   r6, r8, r5              ;calculate offset
                addl3   counter3, r7, counter2  ;the position of maxoperands
                incl    counter3                ;next position
```

```
                cmpl    counter3, #17           ;are we out of instructions?
                beql    notfound                ;yes, go there
                cmpl    instr, (r5)             ;is this the instruction?
                bneq    scandatabase            ;haven't found it yet
                moval   opcodetable, r5         ;start of opcodes
                subl2   #1, counter3            ;cancel the last increment
                addl2   counter3, r5            ;add in the offset
                movb    (r5), opcode            ;this is the opcode
                movl    counter2, r7            ;the address
                movb    (r7), maxoperands       ;max operands for this instr.
                cmpb    opcode, #^xfe           ;is it the ".eq" instruction
                beql    absolute                ;if so, symbol is absolute
                movb    #^a/r/, symboltype      ;or mark it as relative
                jsb     scanforblank            ;do skipchars
                jsb     scanforchar             ;do skipblank
                rsb
notfound:       movb    #1, pass2er             ;an error for pass2 also
                moval   message3, r5            ;illegal instruction message
                jsb     printmessage            ;go print it
                movb    #0, maxoperands         ;no operands allowed
                movb    #0, opcode              ;error code
                jsb     scanforblank            ;do skipchars
                jsb     scanforchar             ;do skipblank
                rsb
absolute:       cmpw    #^x2020, label          ;is it blank
                bneq    legal2                  ;illegal use of ".eq"
                moval   message9, r5            ;call message 9
                jsb     printmessage            ;go print it
legal2:         movb    #^a/a/, symboltype      ;symbol is absolute
                jsb     scanforblank            ;do skipchars
                jsb     scanforchar             ;do skipblank
                rsb                             ;return to main

;=============================================================================

;Subroutine: Checkoperands
;
;This subroutine checks the operand for type and branches to one of the
;following routines to evaluate it: literal, register, deferred,
;displacement, or relative.  (or byte, if it is a constant with no #)
;
;Input: the inline buffer, and the pointer to it counter1.  This points
;to the next character examined after the instruction has been read
;
;Output: none
```

```
checkoperands:  movb    #0, numoperands         ;no operands found yet
                movw    lc, oldlc               ;hold the previous lc
                clrl    r7                      ;clear it
                movb    opcode, r7              ;temporary for overflow
                cmpl    r7, #^xfc               ;is opcode assembly directive?
                bgtr    no_opcode               ;if so, don't increment
                cmpb    opcode, #0              ;is opcode an error?
                beql    no_opcode               ;if so, don't increment
                cmpb    passnum, #2             ;are we on pass 2
                beql    jpoke_opcode            ;put it in memory
                incw    lc                      ;or else increment lc
no_opcode:      cmpb    eolnflag, #1            ;has end of line been reached?
                beql    jnomore                 ;no operands
                movl    counter1, r10           ;temporary pointer
                cmpb    (r10), #^a/;/           ;reached a comment?
                beql    jnomore                 ;if so, no more operands
                clrl    decnum                  ;clear decimal representation
                movb    #0, errorreturn         ;no error yet
                brb     j_operandloop           ;don't execute these branches
jpoke_opcode:   brw     poke_opcode             ;go put opcode in memory
jnext:          brw     next                    ;indirect branch
jnoneofthem:    brw     noneofthem              ;another one
jnomore:        brw     nomore                  ;another one
jregister:      brw     register                ;yet another
j_operandloop:  movw    lc, r9                  ;temporary storage
operandloop:    movl    counter1, r10           ;temporary pointer

                movb    #0, legal               ;preset flag
                cmpb    (r10), #^a/#/           ;is it a pound sign?
                beql    jliteral                ;means literal mode
                brb     reenter1                ;reenter code here
jliteral:       jsb     literal                 ;indirect jump
reenter1:       cmpb    illegal, #1             ;was this a good operand?
                beql    jnoneofthem             ;if so, find next one
                cmpw    lc, r9                  ;test lc
                bgtr    jnext                   ;loop if just found good
                cmpb    legal, #1               ;did it fall through
                beql    jnomore                 ;branch if it did

                cmpb    (r10), #^a/a/           ;is it >= a?
                blss    testregdefer            ;jump down
                cmpb    (r10), #^a/z/           ;is it <= z?
                bgtr    testregdefer            ;jump down

                cmpb    (r10), #^a/r/           ;does it begin with 'r'
                jsb     jregister               ;go test register mode
```

```
                cmpb    illegal, #1             ;was this a good operand?
                beql    symbol                  ;if so, find next one
                cmpw    lc, r9                  ;test lc
                bgtr    jnext                   ;loop if just found good

symbol:         jsb     byterelative            ;go test byte relative mode
                cmpb    illegal, #1             ;was this a good operand?
                beql    jnoneofthem             ;if so, find next one
                cmpw    lc, r9                  ;test lc
                bgtr    next                    ;loop if just found good

testregdefer:   cmpb    (r10), #^a/(/           ;is it left parenthesis
                bneq    testbytedisp            ;if not check the last one
                jsb     regdefer                ;go check it out
                cmpb    illegal, #1             ;was this a good operand?
                beql    noneofthem              ;if so, find next one
                cmpw    lc, r9                  ;test lc
                bgtr    next                    ;loop if just found good

testbytedisp:   cmpb    (r10), #^a/0/           ;see if byte>=0
                blss    noneofthem              ;no
                cmpb    (r10), #^a/9/           ;see if byte<=9
                bgtr    noneofthem              ;no
                jsb     bytedisp                ;go check for this mode
                cmpb    illegal, #1             ;was this a good operand?
                beql    noneofthem              ;if so, find next one
                cmpw    lc, r9                  ;test lc
                bgtr    next                    ;loop if just found good

testbyteconst:  movb    #0, legal               ;preset flag
                jsb     byteconst               ;go test for a byte
                cmpb    illegal, #1             ;was this a good operand?
                beql    noneofthem              ;if so, find next one
                cmpw    lc, r9                  ;test lc
                bgtr    next                    ;loop if just found good
                cmpb    #1, legal               ;did it fall through
                beql    next                    ;branch if so
                brb     noneofthem              ;or else crash with error

next:           addb2   #1, numoperands         ;we have found another operand
                movw    lc, r9                  ;reset the location counter
                jsb     scanforcomma            ;move to next operand
                jsb     scanforchar             ;find it
                cmpb    eolnflag, #1            ;was eoln reached
                beql    nomore                  ;if it was, no more on line
                movl    counter1, r10           ;update pointer
```

```
                cmpb    (r10), #^a/;/           ;or a comment found?  Then
                beql    nomore                  ;no more on this line.
                brw     operandloop             ;or else go look for more

noneofthem:     movb    #1, pass2er             ;or else set error flag
                moval   message2, r5            ;illegal operand message
                jsb     printmessage            ;print it
nomore:         rsb                             ;no more operands
joperandloop2:  brw     operandloop             ;way station

;=============================================================================

;Subroutine: Checknumoper
;This routine compares the numoperands against maxoperands for this
;instruction and calls an error if numoperands is greater than maxoperands.

checknumoper:   cmpb    pass2er, #1                     ;already flagged
                beql    ignore                          ;an error?
                cmpb    numoperands, maxoperands        ;check it
                bgtr    opnumer                         ;call error
                rsb                                     ;else return
opnumer:        movb    #1, pass2er                     ;pass 2 error occurred
                moval   message17, r5                   ;opnum error
                jsb     printmessage                    ;go print it
ignore:         rsb                                     ;return

;=============================================================================

;This subroutine moves a variable-length comment from the input line to
;the next available field in the output line

comment:        moval   outline, r8             ;use to calculate eoln
                moval   symboltable, r9         ;use to calculate eoln
commentloop:    movb    (r5), (r6)              ;move comment field
                subl3   r5, r8, r7              ;has eoln been reached
                cmpl    #1, r7                  ;if it has, go down
                beql    outofroom               ;to outofroom
                subl3   r6, r9, r7              ;has eoln been reached
                cmpl    #1, r7                  ;if it has, go down
                beql    outofroom               ;to outofroom
                incl    r5                      ;move over in inline
                incl    r6                      ;move over in outline
                brb     commentloop             ;loop around
outofroom:      rsb                     ;return from the dumpoperands call

;=============================================================================
```

```
;Subroutine: Dumpoperands
;This will dump operands pointed to by counter1 into the next available
;position in the outline pointed to by counter 2.  These counters are
;not updated.  Instead, r5 and r6 are used as local variables.
;The routine loops until EOLN.

dumpoperands:   moval   outinstr, r10           ;hold address temporarily
                addl3   r10, #6, counter2       ;pointer to outline
                movl    counter1, r5            ;index to inline
                movl    counter2, r6            ;index to outline
nextoperand:    cmpb    (r5), #^a/;/            ;rest of line a comment?
                beql    comment                 ;if so, go deal with it
copyloop:       movb    (r5), (r6)              ;copy char from inline to
                incl    r5                      ;outline
                incl    r6                      ;next available space
                moval   outline, r8             ;eoln for inline
                moval   symboltable, r9         ;eoln for outline
                subl3   r5, r8, r7              ;has eoln been reached
                cmpl    #1, r7                  ;if it has, go down
                beql    done                    ;to done
                subl3   r6, r9, r7              ;check eoln for outline
                cmpl    #1, r7                  ;also, branch if it
                beql    done                    ;has been reached.
                cmpb    (r5), #^a/,/            ;or a comma?
                beql    includecomma            ;include comma in operand
                cmpb    (r5), #^a/ /            ;is there a blank reached?
                bneq    copyloop                ;if not, next char
                brb     skipcomma               ;jump down
includecomma:   movb    (r5), (r6)              ;move last character
skipcomma:      brw     nextoperand             ;do more
done:           rsb                             ;return to mainloop

;=============================================================================

;Subroutine: Poke_opcode
;this subroutine will place the opcode in memory
;it is not called by a jump subroutine but a branch,
;and also returns by a branch.

poke_opcode:    moval   machine, r6             ;start of machine code
                clrl    r7                      ;clear out temporary
                movw    lc, r7                  ;store offset
                addl    r7, r6                  ;add in the offset
                movb    opcode, (r6)            ;move it into machine code
                incw    lc                      ;increment for pass 2
```

```
                brw     no_opcode               ;return to checkoperands

;=============================================================================

;subroutine to handle literal mode
;this routine has been debugged

literal:        ;this will handle literal mode
                movb    #0, illegal             ;operand legal
                incl    r10                     ;move past number sign
                movl    (r10), decnum           ;move entire literal
                jsb     convertd                ;convert ascii to inform
                cmpb    #1, errorreturn         ;was there an error?
                bneq    constantok              ;if not, continue
                movb    #1, illegal             ;flag for illegal operand
                subl2   #1, r10                 ;restore pointer
                rsb                             ;and die
constantok:     cmpb    passnum, #2             ;are we on pass 2?
                beql    pass2lit                ;if so, go handle value
                cmpb    opcode, #^xfb           ;is it a ".by" or ".eq"
                bgtr    notliteral              ;if so, takes no space
                incw    lc                      ;takes at least one byte
                clrl    r5                      ;must compare as longwords
                movb    inform, r5              ;put inform in r5
                cmpl    r5, #63                 ;check the literal
                bgtr    toobig                  ;too big to fit in one byte
                subl2   #1, r10                 ;restore pointer
                rsb                             ;return
notliteral:     cmpb    opcode, #^xfe           ;is it the ".eq"
                beql    not_eq                  ;if so, don't increment
                incw    lc                      ;takes at least one byte
not_eq:         subl2   #1, r10                 ;restore pointer
                rsb                             ;return
toobig:         subl2   #1, r10                 ;restore pointer
                incw    lc                      ;takes another byte
                rsb                             ;go back to checkoperand
pass2lit:       cmpb    opcode, #^xfe           ;is it the '.eq' instr
                beql    liteq                   ;if so, branch
                clrl    r5                      ;must compare longword
                movb    inform, r5              ;hold for a second
                cmpl    r5, #63                 ;too big to fit in one byte?
                bgtr    pass2toobig             ;if so, branch down
                moval   machine, r6             ;start of machine code
                clrl    r7                      ;clear out temporary
                movw    lc, r7                  ;store offset
                addl    r7, r6                  ;add in the offset
```

```
                movb    inform, (r6)            ;move it into machine code
                incw    lc                      ;increment for pass 2
                rsb                             ;return
pass2toobig:    moval   machine, r6             ;start of machine code
                clrl    r7                      ;clear out temporary
                movw    lc, r7                  ;store offset
                addl2   r7, r6                  ;add in offset
                movb    #^x8f, (r6)             ;this is immediate mode
                incl    r6                      ;next byte
                movb    inform, (r6)            ;this is the literal value
                addw2   #2, lc                  ;update lc
                rsb                             ;return for more
liteq:          moval   symboltable, r5         ;start of symboltable
                moval   endsymbol, r6           ;end of symboltable
lloop:          cmpl    (r5), label             ;did we find symbol?
                beql    lfoundsym               ;if so, found it
                addl2   #8, r5                  ;increment pointer
                cmpl    r5, r6                  ;did we hit end?
                bneq    lloop                   ;loop around
                movb    #1, pass2er             ;pass 2 error occured
                movb    #1, illegal             ;tell checkoperands
                moval   message4, r5            ;undefined symbol reference
                jsb     printmessage            ;go tell user
                rsb                             ;or else return
lfoundsym:      addl2   #4, r5                  ;value of label
                movb    inform, (r5)            ;zero for first byte
                incl    r5                      ;increment to middle position
                movb    #0, (r5)                ;middle byte to zero also
                incl    r5                      ;most significant byte
                movb    #0, r5                  ;msb set to zero
                movb    #1, legal               ;flag routine to fall through
                rsb                             ;return without changing lc

;=============================================================================

;Subroutine to handle byte relative mode

byterelative:   ;In pass 2 this routine will look up the symbol to see
                ;if it exists.  In pass 1 this routine will allow two
                ;bytes for this addressing mode for a word offset, since
                ;the program can be a maximum of 500 bytes long.
                ;make sure that r10 points to the start of the symbol!

                movl    (r10), lastsymbol       ;the last symbol of
                                                ;the program
                movb    #0, illegal             ;reset flag
```

```
                moval   lastsymbol, r10         ;work with this
                movb    #^a/ /, 3(r10)          ;set rightmost char to blank
                cmpb    passnum, #2             ;are we on pass 2?
                beql    pass2byterel            ;if so, branch down
                cmpb    maxoperands, #1         ;is it a branch?
                bgtr    notbranch               ;if no, add 2 to lc
                addw2   #1, lc                  ;branch takes one byte
                rsb                             ;return
notbranch:      addw2   #2, lc                  ;requires 2 more bytes
                rsb                             ;short, wasn't it!
pass2byterel:   moval   symboltable, r5         ;start of symboltable
                moval   endsymbol, r6           ;end of symboltable
sloop:          cmpl    (r5), (r10)             ;did we find symbol?
                beql    foundsym                ;if so, found it
                addl2   #8, r5                  ;increment pointer
                cmpl    r5, r6                  ;did we hit end?
                bneq    sloop                   ;loop around
                movb    #1, pass2er             ;pass 2 error occured
                movb    #1, illegal             ;tell checkoperands
                moval   message4, r5            ;undefined symbol reference
                jsb     printmessage            ;go tell user
                rsb                             ;or else return
foundsym:       cmpb    maxoperands, #1         ;is it a branch instruction
                beql    noabsolute              ;skip check for absolute
                cmpb    7(r5), #^a/a/           ;is it an absolute symbol?
                beql    jabsymbol               ;if so, treat it differently
noabsolute:     cmpb    7(r5), #^a/r/           ;is it an relative symbol?
                beql    goodsymbol              ;if so, not legal branch
                moval   message6, r5            ;call error message
                jsb     printmessage            ;go print it
                movb    #1, illegal             ;alert checkoperands
                rsb                             ;go back to checkoperands
jabsymbol:      brw     absymbol                ;way station
goodsymbol:     ;this will always be a branch displacement to calculate
                cmpb    maxoperands, #1         ;is it a branch
                bgtr    goodsymbol2             ;if not, use different mode
                addw    #1, lc                  ;update lc first
                addl2   #4, r5                  ;point to value of symbol
                clrl    r8                      ;clear it out
                movw    (r5), r8                ;the symbol location
                clrl    r6                      ;storage for lc
                movw    lc, r6                  ;move it in
                subl3   r6, r8, r7              ;r7 holds the offset
                ;now put it in memory at the proper location
                moval   machine, r6             ;start of machine code
                subw2   #1, lc                  ;restore value of lc
```

```
                clrl    r5                      ;clear it out
                movw    lc, r5                  ;r6 holds offset
                addl2   r5, r6                  ;r6 points to spot in code
                movb    r7, (r6)                ;put the offset in memory
                addw2   #1, lc
                rsb                             ;return for more
goodsymbol2:    ;this will always be a branch displacement to calculate
                addw    #1, lc                  ;update lc first
                addl2   #4, r5                  ;point to value of symbol
                clrl    r8                      ;clear it out
                movw    (r5), r8                ;the symbol location
                clrl    r6                      ;storage for lc
                movw    lc, r6                  ;move it in
                subl3   r6, r8, r7              ;r7 holds the offset
                ;now put it in memory at the proper location
                moval   machine, r6             ;start of machine code
                subw2   #1, lc                  ;restore value of lc
                clrl    r5                      ;clear it out
                movw    lc, r5                  ;r6 holds offset
                addl2   r5, r6                  ;r6 points to spot in code
                movb    #^xaf, (r6)             ;put in addressing mode
                addl2   #1, r6                  ;next byte
                movb    r7, (r6)                ;put the offset in memory
                addw2   #2, lc                  ;2 bytes for location
                rsb                             ;return for more
absymbol:       clrl    r6                      ;temporary
                moval   machine, r7             ;the start of machine code
                movw    lc, r6                  ;the offset
                addl2   r6, r7                  ;r7 is the actual address
                addl2   #4, r5                  ;update pointer
                movb    #^x8f, (r7)             ;move addressing mode
                addl2   #1, r7                  ;next memory location
                movb    (r5), (r7)              ;throw thing in as longword
                addw2   #2, lc                  ;now update
                rsb                             ;back for more

;=============================================================================

;Subroutine to handle register mode
;If the convert routine returns an error, then the string must have
;been alphabetical (byte relative mode) and it will return.
;This routine has been debugged

register:       incl    r10                     ;move past the "r"
                movb    #0, illegal             ;default is legal
                clrl    decnum                  ;first two bytes
```

```
                movw    (r10), decnum           ;move the register number
                jsb     convertd                ;go convert it
                cmpb    errorreturn, #1         ;did an error occur
                beql    illegalreg              ;if so, must not have been
                                                ;register mode
                cmpb    inform, #9              ;not allowed registers>9
                bgtr    illegalreg              ;illegal register reference
                cmpb    passnum, #2             ;are we at pass 2?
                beql    pass2reg                ;if so, write code
                addw2   #1, lc                  ;register mode takes 1 byte
                rsb                             ;or else return
pass2reg:       clrb    r5                      ;clear out to put code in
                addb2   #^x50, r5               ;set high nibble to 5
                addb2   inform, r5              ;add in register number
                moval   machine, r6             ;store address
                clrl    r7                      ;clear temporary
                movw    lc, r7                  ;hold the offset
                addl2   r7, r6                  ;add in the offset
                movb    r5, (r6)                ;move it into machine code
                incl    lc                      ;next available byte
                rsb                             ;finished register mode
illegalreg:     movb    #1, illegal             ;set for illegal
                subl2   #1, r10                 ;return pointer to previous
                rsb                             ;return to main

;=============================================================================

;Subroutine to test register defered mode
;since they both check the format Rx, and both take up the same space, I
;simply call register mode to verify the correctness of the register

regdefer:       incl    r10                     ;skip over the left parenth
                jsb     register                ;go check register
                cmpb    passnum, #1             ;are we on pass 1
                beql    regdeferer              ;if so, don't write code
                cmpb    illegal, #1             ;was register number legal
                beql    regdeferer              ;if not, return
                subw2   #1, lc                  ;decrement lc for a moment
                clrb    r5                      ;to put code in
                addb2   #^X60, r5               ;set high nibble to 6
                addb2   inform, r5              ;add in the register number
                moval   machine, r6             ;store address
                clrl    r7                      ;clear out temporary register
                movw    lc, r7                  ;hold the offset
                addl2   r7, r6                  ;add in the offset
                movb    r5, (r6)                ;move it into machine code
```

```
                incw    lc                      ;reset lc to updated state
regdeferer:     rsb                             ;go back to main
                                                ;lc was already incremented
                                                ;for pass one

;=============================================================================

;Subroutine to test byte displacement mode
;since they both check the constant, I first call literal mode and then
;call register mode to see if it is okay.

jbad_disp:      brw     bad_disp                ;relative branch
jbad_disp2:     brw     bad_disp2               ;another
bytedisp:       movb    #0, illegal             ;call it legal for now
                movl    (r10), decnum           ;move entire literal
                jsb     convertd                ;convert ascii to inform
                movl    r10, pointer3           ;hold place in line
                cmpb    errorreturn, #1         ;was there an error
                beql    jbad_disp               ;illegal displacement
                jsb     scanforparenth          ;go skip to left parenth
                cmpb    eolnflag, #1            ;did we hit eoln
                beql    jbad_disp               ;if so, bad mode
                clrl    r5                      ;clear for storage
                clrl    r8                      ;clear for storage
                movb    inform, r8              ;value of displacement
                incl    r10                     ;skip over the left parenth
                incl    r10                     ;move past the "r"
                clrl    decnum                  ;first two bytes
                movw    (r10), decnum           ;move the register number
                jsb     convertd                ;go convert it
                cmpb    errorreturn, #1         ;did an error occur
                beql    jbad_disp2              ;if so, must not have been
                                                ;good mode
                cmpb    inform, #9              ;not allowed registers>9
                bgtr    jbad_disp2              ;illegal register reference
                cmpb    illegal, #1             ;was register number legal
                beql    bad_disp                ;if not, return
                cmpb    passnum, #1             ;are we on pass 1
                beql    pass1bytedisp           ;if so, don't write code
                cmpl    r8, #64                 ;is it in range 1-63
                blss    bytedispok              ;if 1-63, can use byte
                brw     worddisp                ;or else use word
bytedispok:     clrl    r7                      ;to put code in
                addb2   #^xa0, r7               ;set high nibble to a
                addb2   inform, r7              ;add in the register number
                moval   machine, r5             ;store address
```

```
                clrl    r6                      ;clear out temporary register
                movw    lc, r6                  ;hold the offset
                addl2   r6, r5                  ;add in the offset
                movb    r7, (r5)                ;move it into machine code
                incl    r5                      ;now move to next byte
                movb    r8, (r5)                ;the byte offset value
                brw     pass1bytedisp           ;update the lc now
worddisp:       clrl    r7                      ;to put code in
                addb2   #^xc0, r7               ;set high nibble to c
                addb2   inform, r7              ;add in the register number
                moval   machine, r5             ;store address
                clrl    r6                      ;clear out temporary register
                movw    lc, r6                  ;hold the offset
                addl2   r6, r5                  ;add in the offset
                movb    r7, (r5)                ;move it into machine code
                incl    r5                      ;now move to next byte
                movb    r8, (r5)                ;the byte offset value
                incl    r5                      ;move to next byte in code
                movb    #0, (r5)                ;high byte of word offset
pass1bytedisp:  addw2   #2, lc                  ;increment lc by two
                cmpl    r8, #64                 ;is it too big for 1 byte
                blss    notbigdisp              ;if 1-63, okay
                addw2   #1, lc                  ;or else reserve more space
notbigdisp:     movl    pointer3, r10           ;restore old line pointer
                rsb                             ;return to checkoperands
bad_disp:       movl    pointer3, r10           ;restore old line pointer
                rsb                             ;go back
bad_disp2:      moval   message2, r5            ;illegal operand message
                jsb     printmessage            ;go print it
                movb    #1, illegal             ;might have been literal
                movl    pointer3, r10           ;restore old line pointer
                movb    #1, legal               ;flag for legal but no lc+
                rsb                             ;return

;=============================================================================

;Subroutine: Byteconst

;Subroutine to test if it is a byte without a # before it, used for
;the opcode .by.  It checks for a valid number.
;this form can also be used for the opcode .eq.

byteconst:      movb    #0, illegal             ;default is legal
                cmpb    opcode, #^xfd           ;is it a ".by" or ".eq"
                blss    notbyte                 ;if not, return
                movl    (r10), decnum           ;move the ascii number
```

```
                jsb     convertd                ;convert to inform
                cmpb    #1, errorreturn         ;error occured?
                beql    notbyte                 ;return if so
                cmpb    passnum, #2             ;are we on pass 2?
                beql    storebyte               ;if so, store the byte
                cmpb    opcode, #^xfe           ;is it ".eq"
                beql    noincrement             ;if so, don't increment
                incw    lc                      ;else reserve 1 byte
noincrement:    rsb
storebyte:      cmpb    opcode, #^xfe           ;is it ".eq"
                beql    bceq                    ;if so, branch
                clrl    r6                      ;use as offset
                movw    lc, r6                  ;hold offset
                moval   machine, r5             ;pointer to machine code
                addl2   r6, r5                  ;put the byte here
                movb    inform, (r5)            ;move it
                incw    lc                      ;update to next location
notbyte:        rsb                             ;go back
bceq:           moval   symboltable, r5         ;start of symboltable
                moval   endsymbol, r6           ;end of symboltable
bcloop:         cmpl    (r5), label             ;did we find symbol?
                beql    bcfoundsym              ;if so, found it
                addl2   #8, r5                  ;increment pointer
                cmpl    r5, r6                  ;did we hit end?
                bneq    bcloop                  ;loop around
                movb    #1, pass2er             ;pass 2 error occured
                movb    #1, illegal             ;tell checkoperands
                moval   message4, r5            ;undefined symbol reference
                jsb     printmessage            ;go tell user
                rsb                             ;or else return
bcfoundsym:     addl2   #4, r5                  ;value of label
                movb    inform, (r5)            ;zero for first byte
                incl    r5                      ;increment to middle position
                movb    #0, (r5)                ;middle byte to zero also
                incl    r5                      ;most significant byte
                movb    #0, (r5)                ;msb set to zero
                movb    #1, legal               ;set legality flag
                rsb                             ;return without changing lc

;===============================================================================

; Subroutine: Insertsymbol
; This subroutine will insert a symbol into the symbol table, or call an
; error if one of the following conditions occurs:
;       -a duplicate symbol is found
;       -symbol table overlow occurs
```

```
;
; Input:
; -The symbol stored in label, with the high byte
;  set to blank.
; -The current location counter or value to store in r6
; -the type (absolute or relative) stored in symboltype
;

quitnow:        rsb                             ;return if no symbol exists
insertsymbol:   cmpb    label, #^a/a/           ;is there a symbol?
                blss    quitnow                 ;if not, return
                cmpb    label, blank            ;is it a blank?
                beql    quitnow                 ;this should filter bad symbols
                cmpb    label, #^a/z/           ;is it greater than z
                bgtr    quitnow                 ;ignore it if so
                moval   endsymbol, r5           ;address of endsymbol
                cmpl    #0, symboltable         ;is the first symbol empty
                bneq    case2                   ;no?
                movl    label, symboltable      ;otherwise throw it in
                cmpb    symboltype, #^a/a/      ;is it absolute?
                beql    absolutesym1            ;if so, jump
                movw    lc, symboltable+4               ;into next word
                brb     skipdown1                       ;skip it
absolutesym1:   movb    #0, symboltable+4               ;absolute value
skipdown1:      movb    symboltype, symboltable+7       ;highest byte
                rsb                                     ;done
case2:          moval   symboltable, pointer1   ;pointer is index
searchloop:     movl    pointer1, r6            ;temporary
                cmpc    #4, label, (r6)         ;compare character string
                beql    duplicate               ;a duplicate symbol found
                cmpl    (r6), #0                ;empty spot found?
                beql    addtoend                ;add to end of table
                cmpc    #4, label, (r6)         ;compare character string
                blss    insert                  ;insert behind this one
                addl    #8, pointer1            ;add to the end
                brb     searchloop              ;return to keep looking

insert:         movl    pointer1, r6            ;need the address
movetoend:      addl2   #8, r6                  ;move to next symbol
                cmpb    (r6), #0                ;is it empty?
                bneq    movetoend               ;move to end of table

                cmpl    r6, r5                  ;is it end of symboltable
                beql    overflow                ;if so, print overflow message
                subl3   #8, r6, r7              ;r7 points to last symbol
                movq    (r7), (r6)              ;move whole symbol down
```

```
movedown:       subl2   #8, r7                ;back up and do it again
                subl2   #8, r6                ;move to this point
                moval   symboltable, r10      ;user temporarily
                cmpl    r7, r10               ;have we moved above top?
                blss    addtoend              ;if so, insert here
                movq    (r7), (r6)            ;move the next one
                cmpl    r6, pointer1          ;until there are no more
                bgtr    movedown              ;repeat if not done
addtoend:       cmpl    r6, r5                ;is it end of symboltable
                beql    overflow              ;if so, print overflow message
                movl    label, (r6)           ;otherwise throw it in
                addl2   #4, r6                ;count over 4 bytes
                cmpb    symboltype, #^a/a/    ;is it absolute?
                beql    absolutesym2          ;if so, jump
                movw    lc, (r6)              ;into next word
                brb     skipdown2             ;go down
absolutesym2:   movb    #0, (r6)              ;absolute value
skipdown2:      addl2   #3, r6                ;highest byte of record
                movb    symboltype, (r6)      ;highest byte
                rsb                           ;procedure is done
duplicate:      moval   message1, r5          ;flag for an error
                jsb     printmessage          ;print the error
                rsb                           ;return to main procedure
overflow:       moval   message8, r5          ;message 8 to call
                jsb     printmessage          ;go print the message
                rsb                           ;return to main program

;=============================================================================

;Subroutine: Printmessage
;
;This routine prints an appropriate error/informative message
;
;Input: the address of the message to print stored in r5
;Output: the message to the output file

printmessage:   cmpb            passnum, #1     ;are we on pass 1?
                beql            filterprint     ;if so, go down
                movl            #50, r6         ;messages 50 chars long
                jsb             file$put        ;write to file
                rsb                             ;return from this routine
filterprint:    ;this portion will print certain messages
                ;that may have occured in pass one
                moval           message1, r6    ;redefined symbol
                cmpl            r5, r6          ;was it this one?
                beql            printit         ;if so, okay
```

```
                moval           message8, r6    ;symbol overflow
                cmpl            r5, r6          ;was it this one
                beql            printit         ;if so, okay
                rsb                             ;or else return
printit:        movb            #1, pass1er     ;pass 1 error
                movl            #50, r6         ;50 chars long
                jsb             file$put        ;print line
                rsb                             ;return

;===============================================================================

;Subroutine: Dumpline
;
;this subroutine fills all the fields in the outline for the listing file
;and sends it to be printed.

dumpline:       incb    linenum                 ;increment linenum
                movb    linenum, inform         ;the line number
                jsb     dconvert                ;convert linenum
                movl    outnum, outlinenum      ;print linenum
                movb    oldlc+1, inform         ;hi byte
                jsb     hconvert                ;convert hi byte
                movw    outnum, outlc           ;hi bytes' ascii
                movb    oldlc, inform           ;treat in 2 parts
                jsb     hconvert                ;convert to hex
                movw    outnum, outlc+2         ;high bytes' ascii
                movl    label, outsymbol        ;the symbol
                movl    instr, outinstr         ;the instruction
                moval   outline, r5             ;buffer
                movl    #80, r6                 ;set length
                jsb     file$put                ;write line to file
                rsb                             ;return

;===============================================================================

;Subroutine: Dumpcode
;This routine will print the machine code in hex into the outline.

dumpcode:       clrl    pointer2                ;this will hold number of bytes
                subw3   oldlc, lc, pointer2     ;produced by this line
                cmpb    pointer2, #6            ;were there more than six?
                bgtr    toomany                 ;return if too many
                cmpb    pointer2, #0            ;was any code produced?
                beql    donothing               ;if no, do nothing
                moval   outcode+15, r7          ;rightmost code byte
                movl    #0, pointer3            ;count number of bytes done
```

```
                moval   machine, r8             ;the start of machine code
                clrl    r9                      ;the offset
                movw    oldlc, r9               ;where does the code start
                addl2   r8, r9                  ;the position of the code
cloop:          addl2   #1, pointer3            ;do the next byte
                movb    (r9), inform            ;move binary byte to inform
                jsb     hconvert                ;convert it to hex
                movw    outnum, (r7)            ;move ascii bytes to outline
                subl2   #3, r7                  ;move to the left 3 characters
                movl    pointer2, r5            ;temporary storage
                movl    pointer3, r6            ;temporary storage
                incl    r9                      ;next byte of code
                cmpl    r5, r6                  ;are we done
                bneq    cloop                   ;if not, loop
                rsb
toomany:        moval   message15, r5           ;call too many bytes message
                jsb     printmessage            ;go tell the user
donothing:      rsb                             ;return to pass 2

;================================================================================

;Subroutine: Printtable
;This subroutine prints the symbol table in columns. The table is already
;in sorted order.  It uses input and output to the file macros.

printtable:     moval   message7, r5            ;print message 7
                jsb     printmessage            ;-symbol-table-
                moval   symboltable, r7         ;r7 indexes table
                cmpb    (r7), #0                ;any symbols?
                bneq    repeat                  ;if so, start looping
                rsb                             ;or else return

repeat:         moval   outline, r8             ;pointer to outline
                cmpl    (r7), #0                ;is the symbol null
                beql    exitloop                ;quit if so
                movc5   #1, blank, #32, #80, outline    ;clear outline
                movl    (r7), outline           ;put symbol in outline
                addl2   #4, r7                  ;next table entry
                addl2   #8, r8                  ;position in outline

                movb    2(r7), inform           ;send byte to convert
                jsb     hconvert                ;convert it
                movw    outnum, (r8)            ;put the hex digits
                addl2   #2, r8                  ;move 2 spaces over

                movb    1(r7), inform           ;repeat for the
```

```
                jsb     hconvert                ;second byte of
                movw    outnum, (r8)            ;location counter/ value
                addl2   #2, r8

                movb    (r7), inform            ;do for the third
                jsb     hconvert                ;byte--don't have to
                movw    outnum, (r8)            ;increment after
                addl2   #6, r8                  ;move over 4 blanks
                addl2   #3, r7                  ;move over 3 in inline

                movb    (r7), (r8)              ;the symboltype
                moval   outline, r5             ;buffer
                movl    #20, r6                 ;set length
                jsb     file$put                ;write line to file
                addl2   #1, r7                  ;next symbol

                moval   endsymbol, r9           ;need the address
                cmpl    r7, r9                  ;are we done?
                bneq    jrepeat                 ;no, go continue looping
exitloop:       rsb                             ;yes, we are done
jrepeat:        brw     repeat                  ;this is indirect jump

;=============================================================================

;Subroutine: Convertd
;This subroutine converts a decimal number stored as up to 3 ascii numbers
;to internal unsigned binary number, in the range 0-255.  This rountine
;uses registers r5 and r6 internally
;
; Input: the number stored in ascii in decnum
;
; Output: the number in binary stored in inform
;
; Sample call:  jsb     convertd

convertd:       clrl    r6
                clrl    r5                      ;clear storage
                cmpb    decnum+1, #^a/0/        ;test right digits
                blss    reset1                  ;if punctuation, reset to zero
test3:          cmpb    decnum+2, #^a/0/        ;test 3rd digit
                blss    reset2                  ;if punctuation, reset it too
                brb     no_reset        ;or don't reset values
reset1:         movb    #0, decnum+1    ;reset the middle digit
                brb     test3           ;test the third digit
reset2:         movb    #0, decnum+2    ;reset the right digit
no_reset:       clrb    errorreturn     ;clear error status
```

```
                cmpb    decnum+1, #0    ;is there a second digit?
                beql    shift2          ;if not, shift digit right 2
                cmpb    decnum+2, #0    ;is there a third digit?
                beql    shift1          ;if not, shift digits right one
                brb     reduce                  ;skip down
shift2:         movb    decnum, decnum+2        ;move left digit to middle
                movb    #^x30, decnum           ;fill with ascii zeroes
                movb    #^x30, decnum+1         ;prevents subtraction underflow
                brb     reduce                  ;reduce the digit
shift1:         movb    decnum+1, decnum+2      ;move middle digit right
                movb    decnum, decnum+1        ;move left digit right
                movb    #^x30, decnum           ;erase empty digit
reduce:         subb2   #^x30, decnum+2         ;convert ascii to digit
                subb2   #^x30, decnum+1
                subb2   #^x30, decnum
                cmpb    decnum+2, #9            ;test the value
                bgtr    er1                     ;go print an error
                mull3   decnum, #100, r6        ;use r6 as accumulator
                bicl2   #^xffffff00, r6         ;get rid of extraneous
                mulb3   decnum+1, #10, r5       ;temporary
                addb2   r5,r6                   ;put tens digit in
                clrl    r5
                movb    decnum+2, r5            ;extract ones digit
                addl2   r5, r6                  ;add it in
                movb    r6, inform
                rsb                             ;go back to main program
er1:            movb    #1, errorreturn         ;set error flag
                rsb                             ;return

;=============================================================================

;Subroutine: hconvert
;This subroutine takes an internal number and converts it to two ascii
;codes of its hexadecimal representation.
;
;Input: The number stored in binary in iform
;
;Output: outnum contains two bytes - hex ascii characters
;
;Sample call:    jsb     hconvert

hconvert:       clrl    r6                      ;hi nibble
                clrl    r5                      ;low nibble
                clrl    outnum                  ;blank this thing
                ashl    #-4, inform, r5         ;put left nibble into r5
                bicl    #^xfffffff0, r5         ;blank out extraneous
```

```
                cmpb    r5, #9                  ;is digit in range 0..9?
                bgtr    hcase2                  ;must be a letter
                addb2   #^x30, r5               ;or else convert it
                brb     lobyte                  ;skip to low byte evaluation

hcase2:         addb2   #87, r5                 ;add 87 to convert

lobyte:         bicb3   #^xf0, inform, r6       ;mask high bits
                cmpb    r6, #9                  ;is digit in range 0..9?
                bgtr    hcase4                  ;must be a letter
                addb2   #^x30, r6               ;ascii value in r6
                brb     notascii                ;skip over next command

hcase4:         addb2   #87, r6                 ;convert to an ascii letter
notascii:       movb    r6, outnum+1            ;output ascii number
                movb    r5, outnum              ;the high order nibble
                rsb                             ;return

;===============================================================================
;
;This subroutine takes an internal representation and converts it into three
;decimal ascii digits with a leading zero.
;
;
;input: the byte in stored in inform

;output: the ascii codes stored in outnum

;Sample call:   jsb     dconvert

dconvert:       clrl    r5              ;clear temporary storage
                clrl    outnum          ;the output field
                clrl    r6              ;temporary storage for inform
                movb    inform, r6
                divl3   #100, r6, r5    ;divide inform by 100
                cmpl    r5, #0          ;is there a hundreds digit?
                beql    tens            ;if not, go to tens
                movb    r5, outnum+1    ;the hundreds digit
                mull2   #100, r5        ;temporary number
                subl2   r5, r6          ;reduce the internal form
tens:           divb3   #10, r6, r5     ;divide inform by 10
                cmpl    r5, #0          ;is there a tens digit?
                beql    ones            ;if not, go to ones
                movb    r5, outnum+2    ;the tens digit
                mulb2   #10, r5         ;temporary number
                subb2   r5, r6          ;reduce internal form
```

```
ones:           cmpb    r6, #0          ;is there a ones digit left?
                beql    dectoascii      ;go change to ascii
                movb    r6, outnum+3    ;the third digit
dectoascii:     addb2   #^x30, outnum   ;change numbers to ascii
                addb2   #^x30, outnum+1
                addb2   #^x30, outnum+2
                addb2   #^x30, outnum+3 ;all four digits
                rsb                     ;return to calling procedure

;===============================================================================

finishup:       moval   message14, r5   ;message for successful completion
                jsb     printmessage    ;go print it

badfinishup:    $CLOSE          FAB=FR$OUTFAB
                $CLOSE          FAB=FR$INFAB
                $CLOSE          FAB=FR$FAB_OUT$
                $CLOSE          FAB=FR$FAB_IN$
                $EXIT_S
                .END            BEGIN
```