```
0001:   {$U-}
0002:   {$R-}
0003:   {============================================================}
0004:   {                                                            }
0005:   {               UCSD   ADAPTABLE   ASSEMBLER                 }
0006:   {               ----   ---------   ---------                 }
0007:   {       Patterned after The Waterloo Last Assembler (TLA)     }
0008:   {       Core Authors:  William P. Franks and Dennis Volper    }
0009:   {                                                            }
0010:   {                                                            }
0011:   {               Version :       pdp 11 & LSI 11              }
0012:   {               Date    :       Sept.  27, 1978              }
0013:   {               Author  :       Dennis Volper                }
0014:   {               Release :       I.5.b.1                      }
0015:   {                                                            }
0016:   {                                                            }
0017:   {              Institute for Information Systems             }
0018:   {                UC  San Diego, La Jolla,  CA                }
0019:   {                                                            }
0020:   {                  Kenneth L. Bowles, Director               }
0021:   {                                                            }
0022:   {                      Copyright (C) 1978,                   }
0023:   {       Regents of the University of California, San Diego    }
0024:   {                                                            }
0025:   {============================================================}
0026:   PROGRAM SYSTEMLEVEL;
0027:   TYPE    PHYLE=FILE;
0028:   VAR     FILLER:ARRAY[0..6] OF INTEGER;
0029:           USERINFO:RECORD
0030:                   WORKSRC,WORKCODE:^PHYLE;
0031:                   ERRSYM,ERRBLK,ERRNUM:INTEGER;
0032:                   SLOWTERM,STUPID:BOOLEAN;
0033:                   ALTMODE:CHAR;
0034:                   FILLER2:ARRAY[0..21] OF INTEGER; {change with care...allows}
0035:                   WORKTITLE,SYMTITLE:STRING[15]     {more compile time space}
0036:               END;
0037:
0038:   SEGMENT PROCEDURE TLA(III,JJJ:INTEGER);
0039:   CONST  RELEASEVERSION =TRUE;  {Is this for the outside world?}
0040:          NUMKWORDS      =27;    {The number of key words in this assembler}
0041:          HASHRANGE      =128;   {The hash table size}
0042:          HASHTOP        =127;   {One less than HASHRANGE}
0043:          MACROSIZE      =19;    {The buffer size for a MACRO stored on heap}
0044:          BUFBLKS        =2;     {# of blocks for output buffer}
0045:          BUFLIMIT       =1023;  {(BUFBLKS*512) - 1}
0046:          MAXPROC        =10;    {Maximum number of Procedures per Assembly}
0047:          PAGESIZE       =55;    {Lines printed per page}
0048:          VIEWSTACK      =TRUE;  {Display stack & heap while Assembling}
0049:          DEBUG          =FALSE; {for debugging Assembler}
0050:          CODESIZE       =20;    {Testing values}
0051:          RELEASENAME    ='I.5 [b.1]';
0052:
0053:   {Below constants are Assembler dependent}
0054:
0055:          NOP            =160;   {A one byte NOP}
0056:          ASMNAME        ='11';
0057:          BYTEFIT        =5;     {maximum bytes per output line}
0058:          WORDFIT        =3;     {maximum words per output line}
0059:          HIBYTEFIRST    =FALSE; {First byte is the high-order byte?}
0060:          LISTHIFIRST    =TRUE;
0061:          LCCHAR         ='*';   {Location counter character}
0062:          WORDADDRESSED  =FALSE; {Word as opposed to byte addressed}
0063:          AFTERPLUS      =0;     {An impossible character}
0064:          AFTERMINUS     ='(';   { "-(" is always auto decrement}
0065:          DEFRADIX       =8;     {Default radix}
0066:          LISTRADIX      =8;     {Printed listing radix}
```

```
0067:          HEXSWITCH      ='H';   {Char following number which resets radix}
0068:          DECSWITCH      ='.';
0069:          OCTSWITCH      =0;
0070:          BINSWITCH      ='B';
0071:          RELHI          =FALSE; {Relative byte most significant passed PUTWORD}
0072:
0073:    TYPE BITE=0..255;
0074:         PACKNAME=PACKED ARRAY[0..7] OF CHAR;
0075:         WORDSWAP=PACKED RECORD CASE INTEGER OF
0076:                  0:(HWORD:INTEGER);
0077:                  1:(HIBYTE,LOWBYTE:BITE);
0078:                  2:(HEX1,HEX2,HEX3,HEX4:0..15);
0079:                  3:(OCT2,OCT3,OCT4,OCT5,OCT6:0..7;
0080:                                      OCT1:0..1);
0081:                  4:(BIN:PACKED ARRAY[0..15] OF 0..1);
0082:              END;
0083:         HASHTYPE=RECORD CASE BOOLEAN OF
0084:                  TRUE:(INT:INTEGER);
0085:                 FALSE:(BOL:BOOLEAN)
0086:              END;
0087:         BYTESWAP=PACKED RECORD CASE INTEGER OF
0088:                  0:(BWORD:INTEGER);
0089:                  1:(BADBYTE,GOODBYTE:BITE);
0090:                  2:(REGLOW:0..7;
0091:                     MODELOW:0..7;
0092:                     REGHI:0..7;
0093:                     MODEHI:0..7;
0094:                     DUM2:0..15);
0095:                  3:(XOFFSET:0..255;
0096:                     DUM3:0..255);
0097:                  4:(SOBSET:0..63;
0098:                     DUM4:0..1023)
0099:              END;
0100:
0101:    (*$I ASM1.TEXT*)
0102:                           {start of ASM1}
0103:         {Copyright (c) 1978 Regents of the University of California}
0104:
0105:    TOKENS=(EQUAL,NOTEQUAL,BITWISEOR,EXCLUSIVEOR,DIVIDE,MODULO,ONESCOMPLEMENT,TNOT,
0106:           OPENPAREN,CLOSEPAREN,OPENBRACKET,CLOSEBRACKET,OPNBRACE,CLSBRACE,
0107:           COMMA,OPNBROKEN,CLSBROKEN,QUERY,PLUS,MINUS,
0108:           ASTERISK,AMPERSAND,ATSIGN,COLON,NUMBERSIGN,AUTOINCR,AUTODECR,LOCCTR,
0109:           FIRSTOPCODE,
0110:           REF,DEF,OP1,OP2,OP3,OP4,OP5,OP6,OP7,OP8,OP9,OP10,OP11,OP12,OP13,
0111:           OP14,OP15,OP16,OP17,OP18,OP19,OP20,ALIGN,
0112:           TEOF,BLOCK,WORD,BIGHT,ENDLINE,TMOD,PROC,FUNC,CONDEND,TELSE,ORG,
0113:           ASCII,MACRODEF,CONDITION,EQU,PUBLIC,PRIVATE,TCONST,
0114:           LIST,NOLIST,ASECT,PSECT,TEND,TPAGE,TITLE,
0115:           LASTOPCODE,
0116:           INCLUDE,TLABEL,LOCLABEL,TSTRING,CONSTANT,TIDENTIFIER,STARTFILE,
0117:           MACROEND,EXPAND,TNULL);
0118:         CODETYPE=(A,P);
0119:         SOURCETYPE=(MACROSOURCE,PARMSOURCE,FILESOURCE);
0120:         ATRIBUTETYPE=(DEFABS,PROCS,
0121:                OPS1,OPS2,OPS3,OPS4,OPS5,OPS6,OPS7,OPS8,OPS9,OPS10,OPS11,
0122:                OPS12,OPS13,OPS14,OPS15,OPS16,OPS17,OPS18,OPS19,OPS20,
0123:                DEFRP,DEFREG,DEFCC,DEFIR,
0124:                PUBLICS,CONSTS,PRIVATES,REFS,DEFS,FUNCS,ABS,LABELS,UNKNOWN,MACROS);
0125:         MACROPTR=^MACROTYPE;
0126:         MACROTYPE=PACKED ARRAY[0..MACROSIZE] OF CHAR;
0127:         JTABPTR=^JTAB;
0128:         JTAB=RECORD               {Used for linkinfo references}
0129:            PCOFFSET:INTEGER;
0130:            LAST:JTABPTR
0131:          END;
0132:         BKLABELPTR=^BACKLABEL;
```

```
0133:       SYMTABLEPTR=^SYMBOLTABLE;
0134:       SYMBOLTABLE=RECORD        {Symboltable entry}
0135:          NAME:PACKNAME;
0136:          LINK:SYMTABLEPTR;
0137:          CASE ATTRIBUTE:ATRIBUTETYPE OF
0138:            {OPS1,OPS2,OPS3,OPS4,OPS5,OPS6,OPS7,OPS8,OPS9,OPS10,
0139:            OPS11,OPS12,OPS13,OPS14,OPS15,OPS16,OPS17,OPS18,OPS19,OPS20,
0140:            ABS,DEFABS,DEFRP,DEFREG,DEFCC,DEFIR,LABELS,}
0141:                              UNKNOWN:(OFFSETORVALUE:INTEGER;
0142:                                       FWDREF:BKLABELPTR);
0143:                               MACROS:(MACRO:MACROPTR;
0144:                                      EXPANDMCRO:BOOLEAN);
0145:                                 DEFS:(PROCNUM,CODEOFFSET:INTEGER;
0146:                                       DEFFWDREF:BKLABELPTR);
0147:          PUBLICS,PRIVATES,REFS,CONSTS:(NREFS,NWORDS:INTEGER;
0148:                                       LINKOFFSET:JTABPTR);
0149:                            PROCS,FUNCS:(FUNCNUM,NPARAMS:INTEGER)
0150:          END;
0151:       TEMPTABLE=RECORD            {Temporary table entry}
0152:            TEMPNAME:PACKNAME;
0153:            DEFOFFSET:INTEGER;
0154:            FWDREF:BKLABELPTR;
0155:            TEMPATRIB:ATRIBUTETYPE
0156:          END;
0157:       RELTYPE=(LLREL,LABELREL,LCREL,NOTSET);
0158:       RESULTREC=RECORD            {expression evaluator result record}
0159:            ATTRIBUTE:ATRIBUTETYPE;
0160:            OFFSETORVALUE:INTEGER;
0161:          END;
0162:       RELREC=RECORD               {current expression's relocation info}
0163:            TIPE:RELTYPE;
0164:            OFFSETORVALUE,TEMPLABEL:INTEGER;
0165:            ATTRIBUTE:ATRIBUTETYPE;
0166:            SYM:SYMTABLEPTR
0167:          END;
0168:       BACKLABEL=PACKED RECORD     {forward reference record}
0169:            WORDLC,BYTESIZE:BOOLEAN;
0170:            OFFSET,LC,VALUE:INTEGER;
0171:            NEXT:BKLABELPTR
0172:          END;
0173:       JTABREC=ARRAY[0..6] OF INTEGER;     {for storage of relocation info}
0174:       BUFFERTYPE=PACKED ARRAY[0..511] OF BITE;
0175:       SCRATCHREC=RECORD            {scratch file for temporary storage}
0176:            CLASS:INTEGER;
0177:            CASE BOOLEAN OF
0178:                 TRUE:(JUMPS:JTABREC);
0179:                FALSE:(FWDREF:BACKLABEL)
0180:       END;
0181:
0182:   {-------------------------------------------------------------------}
0183:
0184:   VAR SYM:SYMTABLEPTR;      {pointer to current symboltable entry}
0185:       LEXTOKEN:TOKENS;      {current token returned by LEX}
0186:       OUTBLKNO,             {next output block #}
0187:       TEXTINDEX,            {index into TEXTLINE, containing line of source text}
0188:       MACROINDEX,           {index into macro source sitting on heap}
0189:       SPCIALSTKINDEX,       {index into stack of outstanding special symbols}
0190:       CODECOUNT:INTEGER;    {index into array containing current line's code}
0191:       OPBYTE:BYTESWAP;      {used exclusively by ZOP1 - ZOP20}
0192:       CH:CHAR;
0193:       DISPLAY:BOOLEAN;      {currently displaying output?}
0194:       FULLLABEL:BKLABELPTR; {forward referenced labels still unresolved}
0195:       RESULT:RESULTREC;     {result of last call to expression evaluator}
0196:
0197:       BUFBOTTOM,            {start of BUFFER relative to start of output file}
0198:       BUFFERPOS,            {next output byte relative to start of BUFFER}
```

```
0199:       BUFFERTOP,            {next output byte relative to start of file}
0200:       MAXBUFTOP,            {maximum BUFFERTOP}
0201:       OUTBLKTOP,            {next block after current end of output file}
0202:       PROCSTART,            {start of procedure relative to start of file}
0203:       JCOUNT1,JCOUNT2,JCOUNT3, {indexes for relocation records JTABREC's}
0204:       TEMPTOP,TEMPLABEL,
0205:       BLOCKPTR,BNUM,BLOCKNO,ALTBLOCNO,ALTBLOCPTR,
0206:       PROCNUM,SEGSIZE,PAGENO,
0207:       LINENUM,LISTNUM,
0208:       NUMERRORS,
0209:       OPVAL,CONSTVAL,
0210:       PARMPTR,MCSTKINDEX,LINKEND,SCRATCHEND,CONDINDEX,
0211:       LC,ALC,LASTLC,LOWLC                                  :INTEGER;
0212:
0213:       SYMLAST,FOUND,CONSOLE,STARTLINE,FROMPUTWORD,NOTSTRING,LISTING,JUMPINFO,
0214:       ADVANCE,EXPANDMACRO,PARMCHECK,ALTINPUT,EXPRSSADVANCE,DEFMCHOOK  :BOOLEAN;
0215:       MCPTR:MACROPTR;
0216:       BUFFER:^BUFFERTYPE;   {buffer for output code in core}
0217:       TAB:CHAR;
0218:       LISTFILE:INTERACTIVE;
0219:       TITLELINE,STRVAL,CURFNAME,FIRSTFNAME:STRING;
0220:       TEXTLINE,BLANKLINE:PACKED ARRAY[0..79] OF CHAR;
0221:
0222:       RELOCATE,OPERAND1,OPERAND2,OPERAND3,NULLREL:RELREC;
0223:       NEXTJP:JTABPTR;
0224:       JUMP1,JUMP2,JUMP3:JTABREC;
0225:       FREELABEL:BKLABELPTR;
0226:
0227:       CURRENTATRIB:ATRIBUTETYPE;
0228:       SOURCE:SOURCETYPE;
0229:       CODESECTION:CODETYPE;
0230:       MACROSTACK:ARRAY[0..5] OF MACROPTR;
0231:       PARMSTACK,MCINDEX:ARRAY[0..5] OF INTEGER;
0232:       SPECIALSTK:ARRAY[0..5] OF TOKENS;
0233:       TEMP:ARRAY[0..20] OF TEMPTABLE;
0234:       HASH,HASHRES:ARRAY[0..HASHTOP] OF SYMTABLEPTR;
0235:       LASTSYM:SYMTABLEPTR;
0236:
0237:       ALTFILE:FILE;
0238:       SCRATCH:FILE OF SCRATCHREC;
0239:
0240:       KWORDS:ARRAY[0..NUMKWORDS] OF PACKNAME;
0241:       KTOKEN:ARRAY [0..NUMKWORDS] OF TOKENS;
0242:       XBLOCK:PACKED ARRAY[0..1023] OF CHAR;
0243:       CONSTID,HEXCHAR:PACKED ARRAY[0..15] OF CHAR;
0244:       CODE,BLANKCODE:PACKED ARRAY[0..CODESIZE] OF CHAR;
0245:       HEAP:^INTEGER;
0246:       SEGNAME,PROCNAME:PACKNAME;
0247:       PROCTABLE:ARRAY[0..MAXPROC] OF INTEGER;
0248:
0249:
0250:   PROCEDURE ERROR(ERRORNUM:INTEGER); FORWARD;
0251:   PROCEDURE PATCHCODE(FWDREF:BACKLABEL; BUFINDEX:INTEGER); FORWARD;
0252:   PROCEDURE IOCHECK(QUIT:BOOLEAN); FORWARD;
0253:   PROCEDURE LLCHECK; FORWARD;
0254:   PROCEDURE PRINTPAGE; FORWARD;
0255:   PROCEDURE PRINTLINE; FORWARD;
0256:   PROCEDURE PRINTNUM(WORD:INTEGER; BYTESIZE:BOOLEAN);  FORWARD;
0257:   PROCEDURE PUTBYTE(BYTE:BITE); FORWARD;
0258:   PROCEDURE PUTRELWORD(WORD:INTEGER; BYTESIZE,WORDOFFSET:BOOLEAN); FORWARD;
0259:   PROCEDURE PUTWORD(WORD:INTEGER);  FORWARD;
0260:   PROCEDURE GETCHAR; FORWARD;
0261:   PROCEDURE LEX; FORWARD;
0262:   FUNCTION  EXPRESS(OPERANDREQUIRED:BOOLEAN):BOOLEAN; FORWARD;
0263:   FUNCTION  CHECKOPERAND(CKSPCSTK,CKABS,CKRANGE:BOOLEAN;LO,HI:INTEGER):BOOLEAN;
0264:                                                              FORWARD;
```

```
0265:
0266:   {dummy segments necessary since compiled U-}
0267:   SEGMENT PROCEDURE DUMMY2;  BEGIN END;
0268:   SEGMENT PROCEDURE DUMMY3;  BEGIN END;
0269:   SEGMENT PROCEDURE DUMMY4;  BEGIN END;
0270:   SEGMENT PROCEDURE DUMMY5;  BEGIN END;
0271:   SEGMENT PROCEDURE DUMMY6;  BEGIN END;
0272:   SEGMENT PROCEDURE DUMMY7;  BEGIN END;
0273:   SEGMENT PROCEDURE DUMMY8;  BEGIN END;
0274:   SEGMENT PROCEDURE DUMMY9;  BEGIN END;
0275:
0276:
0277:   SEGMENT PROCEDURE INITIALIZE;
0278:   TYPE  OPREC=RECORD
0279:            OPNAME:PACKNAME;
0280:            OPVALUE:INTEGER;
0281:            OPATRIB:ATRIBUTETYPE
0282:         END;
0283:
0284:   VAR  OK:BOOLEAN;
0285:        COUNT:INTEGER;
0286:        OPFILENAME,LISTNAME:STRING;
0287:        OPFILE:FILE OF OPREC;
0288:
0289:   PROCEDURE KEYTOKENSET;
0290:   BEGIN
0291:     KWORDS[0] :='ALIGN   '; KTOKEN[0] :=ALIGN;
0292:     KWORDS[1] :='ASCII   '; KTOKEN[1] :=ASCII;
0293:     KWORDS[2] :='BLOCK   '; KTOKEN[2] :=BLOCK;
0294:     KWORDS[3] :='BYTE    '; KTOKEN[3] :=BIGHT;
0295:     KWORDS[4] :='CONST   '; KTOKEN[4] :=TCONST;
0296:     KWORDS[5] :='EQU     '; KTOKEN[5] :=EQU;
0297:     KWORDS[6] :='FUNC    '; KTOKEN[6] :=FUNC;
0298:     KWORDS[7] :='PUBLIC  '; KTOKEN[7] :=PUBLIC;
0299:     KWORDS[8] :='PRIVATE '; KTOKEN[8] :=PRIVATE;
0300:     KWORDS[9] :='PROC    '; KTOKEN[9] :=PROC;
0301:     KWORDS[10]:='WORD    '; KTOKEN[10]:=WORD;
0302:     KWORDS[11]:='EXPAND  '; KTOKEN[11]:=EXPAND;
0303:     KWORDS[12]:='MACRO   '; KTOKEN[12]:=MACRODEF;
0304:     KWORDS[13]:='ENDM    '; KTOKEN[13]:=MACROEND;
0305:     KWORDS[14]:='IF      '; KTOKEN[14]:=CONDITION;
0306:     KWORDS[15]:='ENDC    '; KTOKEN[15]:=CONDEND;
0307:     KWORDS[16]:='ELSE    '; KTOKEN[16]:=TELSE;
0308:     KWORDS[17]:='REF     '; KTOKEN[17]:=REF;
0309:     KWORDS[18]:='DEF     '; KTOKEN[18]:=DEF;
0310:     KWORDS[19]:='ORG     '; KTOKEN[19]:=ORG;
0311:     KWORDS[20]:='INCLUDE '; KTOKEN[20]:=INCLUDE;
0312:     KWORDS[21]:='LIST    '; KTOKEN[21]:=LIST;
0313:     KWORDS[22]:='NOLIST  '; KTOKEN[22]:=NOLIST;
0314:     KWORDS[23]:='ASECT   '; KTOKEN[23]:=ASECT;
0315:     KWORDS[24]:='PSECT   '; KTOKEN[24]:=PSECT;
0316:     KWORDS[25]:='TITLE   '; KTOKEN[25]:=TITLE;
0317:     KWORDS[26]:='END     '; KTOKEN[26]:=TEND;
0318:     KWORDS[27]:='PAGE    '; KTOKEN[27]:=TPAGE;
0319:   END;
0320:
0321:   PROCEDURE LEXINIT;
0322:   VAR  HASHA,HASHB:INTEGER;
0323:        ID:PACKNAME;
0324:   BEGIN
0325:     FOR COUNT:=0 TO HASHTOP DO HASH[COUNT]:=NIL;
0326:     KEYTOKENSET;
0327:     REPEAT
0328:       ID:=OPFILE^.OPNAME;
0329:       HASHA:=0; FOUND:=FALSE;
0330:       FOR COUNT:=0 TO 7 DO
```

```
0331:         BEGIN
0332:           HASHA:=HASHA + HASHA; {left shift}
0333:           HASHB:=ORD(ID[COUNT]);
0334:           HASHA:=ORD((NOT ODD(HASHA) AND ODD(HASHB)) OR
0335:                      (ODD(HASHA) AND NOT ODD(HASHB)));
0336:         END;
0337:
0338:       HASHB:=HASHA MOD HASHRANGE; {lo-order part}
0339:       HASHA:=HASHA DIV HASHRANGE; {hi-order part}
0340:       HASHA:=ORD((NOT ODD(HASHA) AND ODD(HASHB)) OR
0341:                  (ODD(HASHA) AND NOT ODD(HASHB))); {xor}
0342:       HASHA:=HASHA MOD HASHRANGE;
0343:       SYM:=HASH[HASHA];
0344:       WHILE (NOT FOUND) AND (SYM<>NIL) DO
0345:         IF SYM^.NAME=ID THEN FOUND:=TRUE
0346:           ELSE SYM:=SYM^.LINK;
0347:       IF FOUND THEN WRITELN('Opcode declared twice=',ID)
0348:         ELSE
0349:           BEGIN
0350:             NEW(SYM,UNKNOWN); {using UNKNOWN here is to save compile time space}
0351:             SYM^.NAME:=ID; SYM^.ATTRIBUTE:=OPFILE^.OPATRIB;
0352:             SYM^.OFFSETORVALUE:=OPFILE^.OPVALUE;
0353:             SYM^.LINK:=HASH[HASHA];
0354:             HASH[HASHA]:=SYM;
0355:             IF DEBUG THEN WRITELN(ID,HASHA:10);
0356:           END;
0357:       GET(OPFILE);
0358:     UNTIL EOF(OPFILE);
0359:     EXPANDMACRO:=TRUE;
0360:     PARMCHECK:=FALSE;
0361:     CURRENTATRIB:=UNKNOWN;
0362:     BLOCKNO:=2;
0363:     ADVANCE:=TRUE;
0364:     MCSTKINDEX:=0;
0365:     SOURCE:=FILESOURCE;
0366:     BLOCKPTR:=1024;
0367:     LEXTOKEN:=ENDLINE;
0368:     TEMPTOP:=0;
0369:   END;
0370:
0371:   BEGIN {Segment INITIALIZE}
0372:     (*$I-*)
0373:     OPFILENAME:=CONCAT(ASMNAME,'.OPCODES');
0374:     OPFILENAME:=CONCAT('*',OPFILENAME);
0375:     RESET(OPFILE,OPFILENAME);
0376:     IF IORESULT<>0 THEN
0377:       BEGIN
0378:         WRITELN(OPFILENAME,' not on vol');
0379:         UNITCLEAR(3);
0380:         EXIT(TLA);
0381:       END;
0382:     FOR COUNT:=0 TO 79 DO BLANKLINE[COUNT]:=CHR(0);
0383:     TEXTLINE:=BLANKLINE;
0384:     WRITELN(ASMNAME,'   Assembler  ',RELEASENAME);
0385:     FOR COUNT:=0 TO CODESIZE DO BLANKCODE[COUNT]:=' ';
0386:     CODE:=BLANKCODE; CODECOUNT:=0; HEXCHAR:='0123456789ABCDEF';
0387:     BUFFERPOS:=0; NUMERRORS:=0;
0388:     TAB:=CHR(9);
0389:     LINENUM:=0;  SPCIALSTKINDEX:=-1; PROCNUM:=0; LISTNUM:=0;
0390:     IF LENGTH(USERINFO.WORKTITLE)=0 THEN
0391:       FIRSTFNAME:=USERINFO.SYMTITLE
0392:     ELSE
0393:       FIRSTFNAME:=USERINFO.WORKTITLE;
0394:     CURFNAME:=FIRSTFNAME;
0395:     REPEAT
0396:       WRITE('Output file for assembled listing: (<CR> for none)');
```

```
0397:       READLN(LISTNAME);
0398:       DISPLAY:=(LISTNAME<>'');  LISTING:=DISPLAY;
0399:       CONSOLE:=(LISTNAME='CONSOLE:') OR (LISTNAME='#1:');
0400:       IF DISPLAY THEN
0401:         IF CONSOLE THEN
0402:           OPENNEW(LISTFILE,'CONSOLE:')
0403:         ELSE
0404:           OPENNEW(LISTFILE,CONCAT(LISTNAME,'.TEXT[*]'));
0405:       OK:=(IORESULT=0);
0406:       IOCHECK(FALSE);
0407:     UNTIL OK;
0408:     (*$I+*)
0409:     IF NOT RELEASEVERSION THEN
0410:       BEGIN
0411:         WRITELN('Relocation info at file end?');
0412:         READ(KEYBOARD,CH);
0413:         JUMPINFO:=(CH='Y') OR (CH='y');
0414:       END
0415:     ELSE JUMPINFO:=TRUE;
0416:     FOR COUNT:=1 TO 9 DO WRITELN;
0417:     NULLREL.TIPE:=NOTSET; NULLREL.TEMPLABEL:=0; NULLREL.SYM:=NIL;
0418:     NULLREL.ATTRIBUTE:=UNKNOWN; NULLREL.OFFSETORVALUE:=0;
0419:     RELOCATE:=NULLREL;
0420:     MARK(HEAP); {To initialize MEMAVAIL}
0421:     EXPRSSADVANCE:=TRUE;  NOTSTRING:=TRUE; DEFMCHOOK:=FALSE;
0422:     ALTINPUT:=FALSE; SYMLAST:=FALSE; FROMPUTWORD:=FALSE;
0423:     LC:=0; LASTLC:=0; LOWLC:=0; ALC:=0;
0424:     CONDINDEX:=-1;
0425:     PROCNAME:='        ';
0426:     PAGENO:=0;
0427:     TITLELINE:=' ';
0428:     IF DISPLAY THEN
0429:       BEGIN
0430:         WRITELN(LISTFILE,'PAGE - ',PAGENO:3);
0431:         PAGENO:=PAGENO + 1;
0432:       END;
0433:     (*$I-*)
0434:     REWRITE(SCRATCH,'*LINKER.INFO'); LINKEND:=0;
0435:     IOCHECK(TRUE);
0436:     (*$I+*)
0437:     NEW(SYM,UNKNOWN); {extra record on heap to garbage}
0438:     LEXINIT;
0439:     IF NOT (CONSOLE AND DISPLAY) THEN
0440:       BEGIN
0441:         WRITELN;
0442:         WRITE('<   0>');
0443:       END;
0444:     CODESECTION:=P;
0445:   END;
0446:
0447:   (*$I ASM2.TEXT*)
0448:                           {start of ASM2}
0449:           {Copyright (c) 1978 Regents of the University of California}
0450:
0451:   SEGMENT PROCEDURE SYMTBLDUMP;
0452:   TYPE  SYMDUMPPTR=^SYMDUMPTYPE;
0453:         SYMDUMPTYPE=RECORD
0454:             SYM:SYMTABLEPTR;
0455:             LLINK,RLINK:SYMDUMPPTR
0456:           END;
0457:
0458:   VAR  HEAP:^INTEGER;
0459:        BUCKET,DUMPCOUNT,SCREENWIDTH,PAGEWIDTH:INTEGER;
0460:        TOPOFDUMP,NEWDUMP:SYMDUMPPTR;
0461:        SAVETITLE,FILL,MSSG:STRING;
0462:
```

```
0463:   PROCEDURE ALPHABETIZE(SYMDUMP:SYMDUMPPTR);
0464:   BEGIN
0465:     IF SYM^.NAME>SYMDUMP^.SYM^.NAME THEN
0466:       IF SYMDUMP^.RLINK=NIL THEN
0467:         BEGIN
0468:           NEW(NEWDUMP);
0469:           NEWDUMP^.RLINK:=NIL;
0470:           NEWDUMP^.LLINK:=NIL;
0471:           NEWDUMP^.SYM:=SYM;
0472:           SYMDUMP^.RLINK:=NEWDUMP;
0473:         END
0474:       ELSE ALPHABETIZE(SYMDUMP^.RLINK)
0475:     ELSE
0476:       IF SYMDUMP^.LLINK=NIL THEN
0477:         BEGIN
0478:           NEW(NEWDUMP);
0479:           NEWDUMP^.RLINK:=NIL;
0480:           NEWDUMP^.LLINK:=NIL;
0481:           NEWDUMP^.SYM:=SYM;
0482:           SYMDUMP^.LLINK:=NEWDUMP;
0483:         END
0484:       ELSE ALPHABETIZE(SYMDUMP^.LLINK);
0485:   END;
0486:
0487:   PROCEDURE DUMPTABLE(SYMDUMP:SYMDUMPPTR);
0488:   BEGIN
0489:     IF SYMDUMP^.LLINK<>NIL THEN DUMPTABLE(SYMDUMP^.LLINK);
0490:     SYM:=SYMDUMP^.SYM;
0491:     WRITE(LISTFILE,SYM^.NAME);
0492:     CASE SYM^.ATTRIBUTE OF
0493:            ABS:MSSG:=' AB ';
0494:         LABELS:MSSG:=' LB ';
0495:          PROCS:MSSG:=' PR ';
0496:          FUNCS:MSSG:=' FC ';
0497:        PUBLICS:MSSG:=' PB ';
0498:       PRIVATES:MSSG:=' PV ';
0499:           REFS:MSSG:=' RF ';
0500:           DEFS:MSSG:=' DF ';
0501:        UNKNOWN:MSSG:=' UD ';
0502:         MACROS:MSSG:=' MC '
0503:       END;
0504:     WRITE(LISTFILE,MSSG);
0505:
0506:     IF (SYM^.ATTRIBUTE=ABS) OR (SYM^.ATTRIBUTE=LABELS) THEN
0507:       BEGIN
0508:         PRINTNUM(SYM^.OFFSETORVALUE,FALSE);
0509:         WRITE(LISTFILE,'|  ');
0510:       END
0511:     ELSE
0512:       WRITE(LISTFILE,FILL);
0513:     DUMPCOUNT:=DUMPCOUNT + 1;
0514:     IF ((DUMPCOUNT MOD PAGEWIDTH=0) AND NOT CONSOLE)
0515:     OR ((DUMPCOUNT MOD SCREENWIDTH=0) AND CONSOLE) THEN
0516:       BEGIN
0517:         WRITELN(LISTFILE);
0518:         LISTNUM:=LISTNUM + 1;
0519:         IF (LISTNUM MOD PAGESIZE=0) THEN PRINTPAGE;
0520:       END;
0521:     IF SYMDUMP^.RLINK<>NIL THEN DUMPTABLE(SYMDUMP^.RLINK);
0522:   END;
0523:
0524:   BEGIN{SYMTBLDUMP}
0525:     MARK(HEAP);
0526:     IF LEXTOKEN=TEND THEN
0527:       BEGIN
0528:         PRINTLINE;
```

```
0529:         TEXTLINE:=BLANKLINE;
0530:       END;
0531:     NEW(SYM);
0532:     SYM^.NAME:='        ';
0533:     NEW(TOPOFDUMP);
0534:     TOPOFDUMP^.SYM:=SYM;
0535:     TOPOFDUMP^.LLINK:=NIL;
0536:     TOPOFDUMP^.RLINK:=NIL;
0537:     FOR BUCKET:=0 TO HASHTOP DO
0538:       BEGIN
0539:         SYM:=HASH[BUCKET];
0540:         WHILE SYM<>NIL DO
0541:           BEGIN
0542:             CASE SYM^.ATTRIBUTE OF
0543:               LABELS,ABS,MACROS,PUBLICS,PRIVATES,CONSTS,REFS,DEFS,
0544:               PROCS,FUNCS,UNKNOWN:
0545:                 ALPHABETIZE(TOPOFDUMP);
0546:             END;
0547:             SYM:=SYM^.LINK;
0548:           END;
0549:       END;
0550:     SAVETITLE:=TITLELINE;
0551:     TITLELINE:='SYMBOLTABLE DUMP';
0552:     PRINTPAGE;
0553:     WRITELN(LISTFILE,
0554:         'AB - Absolute     LB - Label     UD - Undefined     MC - Macro');
0555:     WRITELN(LISTFILE,
0556:         'RF - Ref          DF - Def       PR - Proc          FC - Func');
0557:     WRITELN(LISTFILE,
0558:         'PB - Public       PV - Private   CS - Consts');
0559:     WRITELN(LISTFILE);
0560:     WRITELN(LISTFILE);
0561:     LISTNUM:=LISTNUM + 5;
0562:
0563:     DUMPCOUNT:=0;
0564:     IF LISTRADIX=8 THEN
0565:       BEGIN
0566:         FILL:='------|  ';
0567:         SCREENWIDTH:=3;
0568:         PAGEWIDTH:=6;
0569:       END;
0570:     IF LISTRADIX=16 THEN
0571:       BEGIN
0572:         FILL:='----|  ';
0573:         SCREENWIDTH:=4;
0574:         PAGEWIDTH:=7
0575:       END;
0576:     DUMPTABLE(TOPOFDUMP^.RLINK);
0577:     TITLELINE:=SAVETITLE;
0578:     WRITELN(LISTFILE);
0579:     LISTNUM:=LISTNUM + 1;
0580:     PRINTPAGE;
0581:     RELEASE(HEAP);
0582:   END;
0583:
0584:
0585:
0586:   SEGMENT PROCEDURE PROCEND;
0587:   TYPE   LITYPES=(INVALID,LMODULE,LGLOBALREF,LPUBLIC,LPRIVATE,LCONSTANT,
0588:                   LGLOBALDEF,LPUBLICDEF,LCONSTDEF,LEXTPROC,LEXTFUNC,
0589:                   LSEPPROC,LSEPFUNC);
0590:          LINKREC=RECORD CASE INTEGER OF
0591:                  0:(REFS:ARRAY[0..7] OF INTEGER);
0592:                  1:(NAME:PACKNAME;
0593:                     CASE LITYPE:LITYPES OF
0594:                       LMODULE,LPUBLIC,LPRIVATE,LCONSTANT,LGLOBALREF:
```

```
0595:                                  (FORMAT:(LWORD,LBYTE,LBIG);
0596:                                   NREFS:INTEGER;
0597:                                   NWORDS:INTEGER);
0598:                        LGLOBALDEF:(PROCNUM:INTEGER;
0599:                                   CODEOFFSET:INTEGER);
0600:                  LSEPPROC,LSEPFUNC:(FUNCNUM:INTEGER;
0601:                                   NPARAMS:INTEGER));
0602:                    2:(CLASS:INTEGER;
0603:                       CASE BOOLEAN OF
0604:                            TRUE:(JUMPS:JTABREC);
0605:                           FALSE:(FWDREF:BACKLABEL))
0606:              END;
0607:
0608:   VAR  COUNT,PROCOFFSET,OUTBLKS:INTEGER;
0609:        SWAPLC:WORDSWAP;
0610:        SEGDICT:PACKED ARRAY[0..511] OF CHAR;
0611:        LINKINFO:FILE;
0612:        LINK:FILE OF LINKREC;
0613:        VIEWDUMMY:ARRAY[0..0] OF INTEGER;
0614:
0615:   PROCEDURE PROCEDE;
0616:
0617:   PROCEDURE BUFRESET(NEWPOS:INTEGER);
0618:   VAR OUTBLKS:INTEGER;
0619:
0620:   BEGIN
0621:     (*$I-*)
0622:     IF DEBUG THEN WRITELN('Bufreset');
0623:     IF NEWPOS<BUFBOTTOM THEN
0624:       BEGIN
0625:         OUTBLKS:=(BUFFERTOP DIV 512 - OUTBLKNO) + 1;
0626:         IF OUTBLKS>BUFBLKS THEN OUTBLKS:=BUFBLKS;
0627:         IF BLOCKWRITE(USERINFO.WORKCODE^,BUFFER^,OUTBLKS,OUTBLKNO)<OUTBLKS
0628:           THEN ERROR(54);
0629:         IF OUTBLKNO + OUTBLKS>OUTBLKTOP THEN
0630:           OUTBLKTOP:=OUTBLKNO + OUTBLKS;
0631:         OUTBLKNO:=NEWPOS DIV 512;
0632:         IF IORESULT=0 THEN
0633:          IF BLOCKREAD(USERINFO.WORKCODE^,BUFFER^,BUFBLKS,OUTBLKNO)=0 THEN;
0634:         BUFBOTTOM:=OUTBLKNO*512;
0635:         BUFFERPOS:=NEWPOS MOD 512;
0636:       END
0637:     ELSE IF NEWPOS>BUFBOTTOM + BUFLIMIT THEN
0638:       BEGIN
0639:         OUTBLKS:=(BUFFERTOP DIV 512 - OUTBLKNO) + 1;
0640:         IF OUTBLKS>BUFBLKS THEN OUTBLKS:=BUFBLKS;
0641:         IF BLOCKWRITE(USERINFO.WORKCODE^,BUFFER^,OUTBLKS,OUTBLKNO)<OUTBLKS
0642:           THEN ERROR(54);
0643:         IF OUTBLKNO + OUTBLKS>OUTBLKTOP THEN
0644:           OUTBLKTOP:=OUTBLKNO + OUTBLKS;
0645:         OUTBLKNO:=NEWPOS DIV 512;
0646:         IF OUTBLKNO>=OUTBLKTOP THEN
0647:           BEGIN
0648:             IF IORESULT=0 THEN
0649:               IF BLOCKWRITE(USERINFO.WORKCODE^,BUFFER^,OUTBLKNO-OUTBLKTOP,
0650:                 OUTBLKTOP)<OUTBLKNO-OUTBLKTOP THEN ERROR(54);
0651:             OUTBLKTOP:=OUTBLKNO;
0652:           END
0653:         ELSE
0654:           IF IORESULT=0 THEN
0655:             IF BLOCKREAD(USERINFO.WORKCODE^,BUFFER^,BUFBLKS,OUTBLKNO)=0 THEN;
0656:         BUFBOTTOM:=OUTBLKNO*512;
0657:         BUFFERPOS:=NEWPOS MOD 512;
0658:       END
0659:     ELSE BUFFERPOS:=NEWPOS - BUFBOTTOM;
0660:     IOCHECK(TRUE);
```

```
0661:     (*$I+*)
0662:   END;
0663:
0664:   PROCEDURE PUTJUMPS;
0665:
0666:   PROCEDURE PUTJUMP(CLASS:INTEGER; VAR JUMP:JTABREC);
0667:   VAR I,COUNT,LINKCOUNT:INTEGER;
0668:   BEGIN
0669:     COUNT:=0;
0670:     IF JUMPINFO THEN
0671:       BEGIN
0672:         IF LINKEND<>SCRATCHEND THEN
0673:           BEGIN
0674:             SEEK(LINK,LINKEND);
0675:             FOR LINKCOUNT:=LINKEND+1 TO SCRATCHEND DO
0676:               BEGIN
0677:                 GET(LINK);
0678:                 IF LINK^.CLASS=CLASS THEN
0679:                   FOR I:=0 TO 6 DO
0680:                     IF LINK^.JUMPS[I]<>0  THEN
0681:                       BEGIN
0682:                         PUTWORD(BUFFERTOP - LINK^.JUMPS[I]);
0683:                         COUNT:=COUNT + 1;
0684:                       END;
0685:               END;
0686:           END;
0687:         FOR I:=0 TO 6 DO
0688:           IF JUMP[I]<>0  THEN
0689:             BEGIN
0690:               PUTWORD(BUFFERTOP - JUMP[I]);
0691:               COUNT:=COUNT + 1;
0692:             END;
0693:       END;
0694:     PUTWORD(COUNT);
0695:   END;
0696:
0697:   BEGIN {Putjumps}
0698:     PUTJUMP(1,JUMP1);   {Jumptable entries}
0699:     PUTJUMP(2,JUMP2);
0700:     PUTJUMP(3,JUMP3);
0701:   END;
0702:
0703:   PROCEDURE LINKSET;
0704:   VAR BUCKET:INTEGER;
0705:   BEGIN
0706:     IF DEBUG THEN WRITELN('Linkset');
0707:     IF SCRATCHEND<>0 THEN SEEK(LINK,LINKEND); {ie. file not of length 0}
0708:     FOR BUCKET:=0 TO HASHTOP DO
0709:       BEGIN
0710:         SYM:=HASH[BUCKET];
0711:         WHILE SYM<>NIL DO
0712:           BEGIN
0713:             CASE SYM^.ATTRIBUTE OF
0714:           UNKNOWN:
0715:               BEGIN
0716:                 IF DISPLAY THEN
0717:                   BEGIN
0718:                     WRITELN(LISTFILE);
0719:                     WRITE(LISTFILE,'>>>>>',SYM^.NAME);
0720:                     LISTNUM:=LISTNUM + 1;
0721:                   END;
0722:                 IF NOT (CONSOLE AND DISPLAY) THEN
0723:                   BEGIN
0724:                     WRITELN;
0725:                     WRITE('>>>>>',SYM^.NAME);
0726:                   END;
```

```
0727:                 ERROR(1{Undefined label});
0728:               END;
0729:           PUBLICS,PRIVATES,CONSTS,REFS,DEFS,PROCS,FUNCS: {Linkfile info}
0730:               BEGIN
0731:                 FILLCHAR(LINK^,SIZEOF(LINKREC),0);
0732:
0733:                 CASE SYM^.ATTRIBUTE OF
0734:                   PUBLICS:LINK^.LITYPE:=LPUBLIC;
0735:                   PRIVATES:LINK^.LITYPE:=LPRIVATE;
0736:                   CONSTS:LINK^.LITYPE:=LCONSTANT;
0737:                   REFS:LINK^.LITYPE:=LGLOBALREF;
0738:                   DEFS:LINK^.LITYPE:=LGLOBALDEF;
0739:                   PROCS:LINK^.LITYPE:=LSEPPROC;
0740:                   FUNCS:LINK^.LITYPE:=LSEPFUNC
0741:                 END;
0742:                 LINK^.NAME:=SYM^.NAME;
0743:                 CASE SYM^.ATTRIBUTE OF
0744:                   PUBLICS,PRIVATES,CONSTS,REFS:
0745:                     BEGIN
0746:                       LINK^.FORMAT:=LWORD;
0747:                       LINK^.NREFS:=SYM^.NREFS;
0748:                       LINK^.NWORDS:=SYM^.NWORDS;
0749:                       LINKEND:=LINKEND + 1;
0750:                       PUT(LINK); COUNT:=0;
0751:                       WHILE SYM^.LINKOFFSET<>NIL DO
0752:                         BEGIN
0753:                           LINK^.REFS[COUNT]:=SYM^.LINKOFFSET^.PCOFFSET;
0754:                           COUNT:=COUNT + 1;
0755:                           IF COUNT=8 THEN
0756:                             BEGIN
0757:                               PUT(LINK);
0758:                               FILLCHAR(LINK^,SIZEOF(LINKREC),0);
0759:                               LINKEND:=LINKEND + 1;
0760:                               COUNT:=0;
0761:                             END;
0762:                           SYM^.LINKOFFSET:=SYM^.LINKOFFSET^.LAST;
0763:                         END;
0764:                       IF COUNT<>0 THEN
0765:                         BEGIN
0766:                           PUT(LINK);
0767:                           LINKEND:=LINKEND + 1;
0768:                         END;
0769:                     END;
0770:                   DEFS:
0771:                     IF SYM^.CODEOFFSET=-1 THEN
0772:                       BEGIN
0773:                         WRITELN(LISTFILE);
0774:                         IF DISPLAY THEN WRITE(LISTFILE,SYM^.NAME);
0775:                         IF NOT (CONSOLE AND DISPLAY) THEN
0776:                           BEGIN
0777:                             WRITELN;
0778:                             WRITE(SYM^.NAME);
0779:                           END;
0780:                         ERROR(1{Undefined label});
0781:                       END
0782:                     ELSE
0783:                       BEGIN
0784:                         LINK^.LITYPE:=LGLOBALDEF;
0785:                         LINK^.PROCNUM:=SYM^.PROCNUM;
0786:                         LINK^.CODEOFFSET:=SYM^.CODEOFFSET;
0787:                         LINKEND:=LINKEND + 1; PUT(LINK);
0788:                       END;
0789:
0790:                   PROCS,FUNCS:
0791:                     BEGIN
0792:                       IF SYM^.ATTRIBUTE=PROCS THEN LINK^.LITYPE:=LSEPPROC
```

```
0793:                         ELSE LINK^.LITYPE:=LSEPFUNC;
0794:                       LINK^.FUNCNUM:=SYM^.FUNCNUM;
0795:                       LINK^.NPARAMS:=SYM^.NPARAMS;
0796:                       PUT(LINK);
0797:                       LINK^.LITYPE:=LGLOBALDEF;
0798:                       LINK^.PROCNUM:=SYM^.FUNCNUM;
0799:                       LINK^.CODEOFFSET:=0; {proc's start at LC=0}
0800:                       PUT(LINK);
0801:                       LINKEND:=LINKEND + 2;
0802:                     END
0803:                 END;
0804:                 IF DEBUG THEN WRITELN('link entry:',SYM^.NAME);
0805:               END;
0806:             END;
0807:             SYM:=SYM^.LINK;
0808:           END;
0809:       END;
0810:   END;
0811:
0812:   PROCEDURE LABELFIX; {fix label forward references}
0813:   VAR   SWAP:WORDSWAP;
0814:         FWDREF:BACKLABEL;
0815:         LINKCOUNT:INTEGER;
0816:         KLUDGEPTR:^INTEGER;
0817:   BEGIN
0818:     RESET(LINK,'*LINKER.INFO');
0819:     MARK(KLUDGEPTR);
0820:     IF SCRATCHEND<>LINKEND THEN SEEK(LINK,LINKEND);
0821:     FOR LINKCOUNT:=LINKEND+1 TO SCRATCHEND DO
0822:       BEGIN
0823:         GET(LINK);
0824:         IF LINK^.CLASS=0 THEN
0825:           BEGIN
0826:             FWDREF:=LINK^.FWDREF;
0827:             BUFRESET(FWDREF.OFFSET);
0828:             PATCHCODE(FWDREF,FWDREF.OFFSET-BUFBOTTOM);
0829:           END;
0830:       END;
0831:   END;
0832:
0833:   BEGIN {Procede}
0834:     IF DEBUG THEN WRITELN('Procede');
0835:     IF DISPLAY THEN
0836:       WRITELN(LISTFILE,'Current available space is ',MEMAVAIL,' words');
0837:     IF NOT (DISPLAY AND CONSOLE) THEN
0838:       BEGIN
0839:         WRITELN;
0840:         WRITELN('Current available space is ',MEMAVAIL,' words');
0841:         WRITE('<',LINENUM:4,'>');
0842:       END;
0843:     LLCHECK;
0844:     CLOSE(SCRATCH,LOCK);
0845:     LABELFIX;
0846:
0847:     BUFRESET(MAXBUFTOP);
0848:     BUFFERTOP:=BUFBOTTOM + BUFFERPOS; {BUFRESET doesn't affect BUFFERTOP}
0849:     IF ODD(BUFFERPOS) THEN PUTBYTE(0);
0850:     RELOCATE:=NULLREL;
0851:     PUTJUMPS;                          {Jumptable entries}
0852:     PUTWORD(BUFFERTOP - PROCSTART);    {Enter IC}
0853:     PUTWORD(0);                        {Proc #, Lex level}
0854:     LINKSET;
0855:     PROCTABLE[PROCNUM]:=BUFFERTOP - PROCSTART;
0856:     SEGSIZE:=SEGSIZE + BUFFERTOP - PROCSTART;
0857:     HASH:=HASHRES;
0858:     RELEASE(HEAP);
```

```
0859:   END;
0860:
0861:   PROCEDURE FIRSTPROC; {Set up the buffer for output assembled code}
0862:   VAR   BUFSETUP:^BUFFERTYPE;
0863:   BEGIN
0864:     IF DEBUG THEN WRITELN('Procstart');
0865:     NEW(BUFSETUP); BUFFER:=BUFSETUP;
0866:     HASHRES:=HASH; {For symboltable cutback}
0867:     FOR COUNT:=2 TO BUFBLKS DO
0868:       NEW(BUFSETUP);
0869:     FILLCHAR(BUFFER^,BUFLIMIT,0);{Clear buffer to aid DEBUGGING}
0870:     IF DISPLAY THEN WRITELN(LISTFILE,
0871:               BUFBLKS,' blocks for procedure code  ',MEMAVAIL,' words left');
0872:     IF NOT (DISPLAY AND CONSOLE) THEN
0873:       BEGIN
0874:         WRITELN;
0875:         WRITELN(BUFBLKS,' blocks for procedure code  ',MEMAVAIL,' words left');
0876:         WRITE('<',LINENUM:4,'>');
0877:       END;
0878:     BUFBOTTOM:=512; BUFFERTOP:=512; MAXBUFTOP:=512;
0879:     OUTBLKNO:=1; OUTBLKTOP:=1;
0880:     BUFFERPOS:=0; SEGSIZE:=0;
0881:     FILLCHAR(PROCTABLE,SIZEOF(PROCTABLE),0);
0882:     (*$I-*)
0883:     IF BLOCKWRITE(USERINFO.WORKCODE^,BUFFER^,1)=0 THEN ERROR(54);
0884:     IOCHECK(TRUE);           {Segment dictionary}
0885:     (*$I+*)
0886:   END;
0887:
0888:   BEGIN {Segment Procend}
0889:     IF VIEWSTACK THEN UNITWRITE(3,VIEWDUMMY[-1600],35); {reset display of heap}
0890:     IF DEBUG THEN WRITELN('Procend');
0891:     IF PROCNUM>0 THEN PROCEDE
0892:       ELSE FIRSTPROC;
0893:     IF LEXTOKEN=TEND THEN
0894:       BEGIN
0895:         PROCOFFSET:=2;     {Procedure table}
0896:         FOR COUNT:=PROCNUM DOWNTO 1 DO
0897:           BEGIN
0898:             PUTWORD(PROCOFFSET);
0899:             PROCOFFSET:=PROCOFFSET + PROCTABLE[COUNT] + 2;
0900:           END;
0901:         PUTBYTE(1);               {Segment #}
0902:         PUTBYTE(PROCNUM);         {# of Procedures}
0903:
0904:         SEGSIZE:=PROCOFFSET;
0905:         COUNT:=(BUFFERPOS + 511) DIV 512;
0906:         (*$I-*)
0907:         IF BLOCKWRITE(USERINFO.WORKCODE^,BUFFER^,COUNT,OUTBLKNO)<COUNT
0908:           THEN ERROR(54);
0909:         OUTBLKNO:=OUTBLKNO + COUNT;
0910:         LINK^.LITYPE:=INVALID;  LINKEND:=LINKEND + 1;
0911:         PUT(LINK);  CLOSE(LINK,LOCK);
0912:         RESET(LINKINFO,'*LINKER.INFO');
0913:         COUNT:=((LINKEND*16) + 511) DIV 512;
0914:         IF IORESULT=0 THEN
0915:           IF BLOCKREAD(LINKINFO,BUFFER^,COUNT)=0 THEN;
0916:         FILLCHAR(BUFFER^[LINKEND*16],512,0); {for easier linkinfo debugging}
0917:         IF IORESULT=0 THEN
0918:           IF BLOCKWRITE(USERINFO.WORKCODE^,BUFFER^,COUNT,OUTBLKNO)<COUNT
0919:             THEN ERROR(54);
0920:         FILLCHAR(SEGDICT,SIZEOF(SEGDICT),0);
0921:         SEGDICT[4]:=CHR(1);       {Pointer to starting block}
0922:         SWAPLC.HWORD:=SEGSIZE;    {Segsize}
0923:         IF HIBYTEFIRST THEN
0924:           BEGIN
```

```
0925:             SEGDICT[6]:=CHR(SWAPLC.HIBYTE);
0926:             SEGDICT[7]:=CHR(SWAPLC.LOWBYTE);
0927:           END
0928:         ELSE
0929:           BEGIN
0930:             SEGDICT[6]:=CHR(SWAPLC.LOWBYTE);
0931:             SEGDICT[7]:=CHR(SWAPLC.HIBYTE);
0932:           END;
0933:         FILLCHAR(SEGDICT[64],128,' ');
0934:         FOR COUNT:=72 TO 79 DO
0935:           SEGDICT[COUNT]:=SEGNAME[COUNT-72];
0936:         SEGDICT[194]:=CHR(4);   {Segment type SEPRTSEG}
0937:         IF IORESULT=0 THEN IF BLOCKWRITE(USERINFO.WORKCODE^,SEGDICT,1,0)=0
0938:           THEN ERROR(54);
0939:         IF LISTING AND NOT CONSOLE THEN PAGE(LISTFILE);
0940:         IF LISTING THEN CLOSE(LISTFILE,LOCK);
0941:         CLOSE(LINKINFO,PURGE);
0942:         IOCHECK(TRUE);
0943:         UNITCLEAR(3);
0944:         (*$I+*)
0945:         WRITELN;
0946:         WRITELN('Assembly complete:',LINENUM:10,' lines');
0947:         WRITELN(NUMERRORS:6,'   Errors flagged on this Assembly');
0948:       END
0949:     ELSE
0950:       BEGIN
0951:         MARK(HEAP);
0952:         PROCNUM:=PROCNUM + 1;
0953:         LC:=0; LASTLC:=0; LOWLC:=0;
0954:         FILLCHAR(JUMP1,SIZEOF(JUMP1),0);  JCOUNT1:=0;
0955:         FILLCHAR(JUMP2,SIZEOF(JUMP2),0);  JCOUNT2:=0;
0956:         FILLCHAR(JUMP3,SIZEOF(JUMP3),0);  JCOUNT3:=0;
0957:         SCRATCHEND:=LINKEND;
0958:         IF PROCNUM>1 THEN
0959:           BEGIN
0960:             CLOSE(LINK,LOCK);
0961:             RESET(SCRATCH,'*LINKER.INFO');
0962:             SEEK(SCRATCH,LINKEND);
0963:           END;
0964:         NEW(FULLLABEL); FULLLABEL^.NEXT:=NIL;
0965:         FREELABEL:=NIL;
0966:         PROCSTART:=BUFFERTOP;
0967:         IF LEXTOKEN=PROC THEN CURRENTATRIB:=PROCS
0968:           ELSE CURRENTATRIB:=FUNCS;
0969:         LEX;
0970:         IF LEXTOKEN<>TIDENTIFIER THEN ERROR(3{Must have procedure name})
0971:           ELSE
0972:             BEGIN
0973:               IF PROCNUM=1 THEN SEGNAME:=SYM^.NAME;
0974:               PROCNAME:=SYM^.NAME;
0975:               SYM^.FUNCNUM:=PROCNUM;
0976:               LEX;
0977:               IF LEXTOKEN=COMMA THEN
0978:                 BEGIN
0979:                   LEX;
0980:                   IF LEXTOKEN<>CONSTANT THEN
0981:                      ERROR(4{Number of parameters expected})
0982:                   ELSE SYM^.NPARAMS:=CONSTVAL;
0983:                   LEX;
0984:                 END ELSE SYM^.NPARAMS:=0;
0985:             END;
0986:         CODE:=BLANKCODE; CODECOUNT:=0;
0987:         IF DISPLAY THEN PRINTPAGE;
0988:         IF LEXTOKEN<>ENDLINE THEN
0989:           BEGIN
0990:             ERROR(5{extra garbage on line});
```

```
0991:             WHILE LEXTOKEN<>ENDLINE DO LEX;
0992:           END;
0993:         PRINTLINE;
0994:         TEXTLINE:=BLANKLINE;
0995:         TEXTINDEX:=-1;
0996:         CURRENTATRIB:=UNKNOWN;
0997:       END;
0998:   END;
0999:
1000:
1001:   (*$I ASM3.TEXT*)
1002:                           {start of ASM3}
1003:           {Copyright (c) 1978 Regents of University of California}
1004:
1005:   SEGMENT PROCEDURE ASSEMBLE;
1006:   VAR   VIEWDUMMY:ARRAY[0..0] OF INTEGER;
1007:
1008:   PROCEDURE ZCOND;
1009:   VAR  I,CURRENT:INTEGER;
1010:        ID:PACKNAME;
1011:
1012:   FUNCTION CONDTRUE:BOOLEAN;
1013:   VAR  ISEQUAL,CHECKEQUAL:BOOLEAN;
1014:        INTSAVE:INTEGER;
1015:        STRSAVE:STRING;
1016:
1017:   BEGIN
1018:     LEX;
1019:     IF LEXTOKEN=TSTRING THEN
1020:       BEGIN
1021:         STRSAVE:=STRVAL;
1022:         LEX;
1023:         CHECKEQUAL:=(LEXTOKEN=EQUAL);
1024:         IF NOT CHECKEQUAL THEN
1025:           IF LEXTOKEN<>NOTEQUAL THEN ERROR(62{'=' or '<>' expected});
1026:         LEX;
1027:         IF LEXTOKEN=TSTRING THEN
1028:           BEGIN
1029:             ISEQUAL:=(STRVAL=STRSAVE);
1030:             CONDTRUE:=(CHECKEQUAL=ISEQUAL);
1031:           END
1032:         ELSE
1033:           BEGIN
1034:             ERROR(46{string expected});
1035:             CONDTRUE:=TRUE;
1036:           END;
1037:         LEX;
1038:       END
1039:     ELSE
1040:       BEGIN
1041:         EXPRSSADVANCE:=FALSE;
1042:         IF EXPRESS(TRUE) THEN
1043:           IF SPCIALSTKINDEX=-1 THEN
1044:             CONDTRUE:=(RESULT.OFFSETORVALUE<>0)
1045:           ELSE
1046:             BEGIN
1047:               INTSAVE:=RESULT.OFFSETORVALUE;
1048:               CHECKEQUAL:=(SPECIALSTK[SPCIALSTKINDEX]=EQUAL);
1049:               SPCIALSTKINDEX:=SPCIALSTKINDEX-1;
1050:               IF EXPRESS(TRUE) THEN
1051:                 BEGIN
1052:                   ISEQUAL:=(RESULT.OFFSETORVALUE=INTSAVE);
1053:                   CONDTRUE:=(CHECKEQUAL=ISEQUAL);
1054:                 END
1055:               ELSE CONDTRUE:=TRUE;
1056:             END
```

```
1057:         ELSE CONDTRUE:=TRUE;
1058:       END;
1059:   END;
1060:
1061:   BEGIN
1062:     CONDINDEX:=CONDINDEX + 1;
1063:     CURRENT:=CONDINDEX;
1064:     IF NOT CONDTRUE THEN
1065:       BEGIN
1066:         IF LEXTOKEN<>ENDLINE THEN
1067:           BEGIN
1068:             ERROR(5{Extra garbage on line});
1069:             WHILE LEXTOKEN<>ENDLINE DO LEX;
1070:           END;
1071:         PRINTLINE;  ID:='        ';  I:=0;
1072:         TEXTLINE:=BLANKLINE; TEXTINDEX:=-1;
1073:
1074:         REPEAT
1075:           GETCHAR;
1076:           IF TEXTINDEX>79 THEN ERROR(6{input line over 80 chars});
1077:           IF CH=CHR(13) THEN
1078:             BEGIN
1079:               TEXTLINE:=BLANKLINE; TEXTINDEX:=-1;
1080:             END
1081:           ELSE IF CH='.' THEN
1082:             BEGIN
1083:               I:=0;
1084:               ID:='        ';
1085:             END
1086:           ELSE IF I<5 THEN
1087:             BEGIN
1088:               ID[I]:=CH;
1089:               I:=I + 1;
1090:             END;
1091:           IF ID='IF      ' THEN
1092:             CONDINDEX:=CONDINDEX + 1
1093:           ELSE IF ID='ENDC    ' THEN
1094:             IF CONDINDEX<0 THEN
1095:               BEGIN
1096:                 ERROR(7{Not enough ifs});
1097:                 EXIT(ZCOND);
1098:               END
1099:             ELSE CONDINDEX:=CONDINDEX - 1;
1100:         UNTIL ((CURRENT=CONDINDEX) AND (ID='ELSE    ')) OR
1101:               ((CURRENT=CONDINDEX + 1) AND (ID='ENDC    '));
1102:         LEXTOKEN:=TNULL; {Different from ENDLINE}
1103:         LEX;
1104:       END;
1105:   END;
1106:
1107:   PROCEDURE ZELSE;
1108:   VAR  I,CURRENT:INTEGER;
1109:        ID:PACKNAME;
1110:   BEGIN
1111:     CURRENT:=CONDINDEX;  ID:='        ';  I:=0;
1112:     PRINTLINE;
1113:     REPEAT
1114:       GETCHAR;
1115:       IF TEXTINDEX>79 THEN ERROR(6{input line over 80 chars});
1116:       IF CH=CHR(13) THEN
1117:         BEGIN
1118:           TEXTLINE:=BLANKLINE; TEXTINDEX:=-1;
1119:         END
1120:       ELSE IF CH='.' THEN
1121:         BEGIN
1122:           I:=0;
```

```
1123:           ID:='        ';
1124:         END
1125:       ELSE IF I<5 THEN
1126:         BEGIN
1127:           ID[I]:=CH;
1128:           I:=I + 1;
1129:         END;
1130:
1131:       IF ID='IF      ' THEN
1132:         CONDINDEX:=CONDINDEX + 1
1133:       ELSE IF ID='ENDC    ' THEN
1134:         IF CONDINDEX<0 THEN
1135:           BEGIN
1136:             ERROR(7{Not enough ifs});
1137:             EXIT(ZCOND);
1138:           END
1139:         ELSE CONDINDEX:=CONDINDEX - 1;
1140:     UNTIL (CURRENT=CONDINDEX + 1) AND (ID='ENDC    ');
1141:     LEX;
1142:   END;
1143:
1144:   PROCEDURE COREFIX(ENTRY:BKLABELPTR; ADDVALUE:INTEGER);
1145:   VAR   BUFINDEX:INTEGER;
1146:         NEXTENTRY:BKLABELPTR;
1147:         PRINTLC:WORDSWAP;
1148:   BEGIN
1149:     WHILE ENTRY<>NIL DO
1150:       BEGIN
1151:         NEXTENTRY:=ENTRY^.NEXT;
1152:         BUFINDEX:=ENTRY^.OFFSET-BUFBOTTOM;
1153:         ENTRY^.VALUE:=ENTRY^.VALUE + ADDVALUE;
1154:         IF (NOT WORDADDRESSED) AND (ENTRY^.WORDLC) THEN
1155:           ENTRY^.VALUE:=ENTRY^.VALUE DIV 2;
1156:         IF (BUFINDEX>=0) AND (BUFINDEX<BUFLIMIT) THEN
1157:           PATCHCODE(ENTRY^,BUFINDEX)
1158:         ELSE
1159:           BEGIN
1160:             SCRATCH^.CLASS:=0;  {store it away for PROCEND}
1161:             SCRATCH^.FWDREF:=ENTRY^;
1162:             PUT(SCRATCH);
1163:             SCRATCHEND:=SCRATCHEND + 1;
1164:           END;
1165:         ENTRY^.NEXT:=FREELABEL;
1166:         FREELABEL:=ENTRY;
1167:         ENTRY:=NEXTENTRY;
1168:       END;
1169:   END;
1170:
1171:   PROCEDURE ZLABEL;
1172:   VAR   SWAP:INTEGER;
1173:         NEXTENTRY,ENTRY:BKLABELPTR;
1174:   BEGIN
1175:     ENTRY:=NIL; {Set up nil pointer for error exit}
1176:     IF LEXTOKEN=TLABEL THEN
1177:       BEGIN
1178:         IF SYM^.ATTRIBUTE<>UNKNOWN THEN
1179:           BEGIN
1180:             IF SYM^.ATTRIBUTE=DEFS THEN
1181:               BEGIN
1182:                 SYMLAST:=TRUE;
1183:                 SYM^.CODEOFFSET:=LC;
1184:                 ENTRY:=SYM^.DEFFWDREF;
1185:               END
1186:             ELSE
1187:               BEGIN
1188:                 ERROR(9{identifier previously declared});
```

```
1189:                SYMLAST:=FALSE;
1190:              END;
1191:          END
1192:        ELSE
1193:          BEGIN
1194:            IF CODESECTION=A THEN
1195:              BEGIN
1196:                SYM^.ATTRIBUTE:=ABS;
1197:                SYM^.OFFSETORVALUE:=ALC;
1198:              END
1199:            ELSE
1200:              BEGIN
1201:                SYM^.ATTRIBUTE:=LABELS;
1202:                SYM^.OFFSETORVALUE:=LC;
1203:              END;
1204:            SYMLAST:=TRUE;
1205:            LASTSYM:=SYM;
1206:            IF (CODESECTION=A) AND (ENTRY<>NIL) THEN
1207:              ERROR(8{must be declared in ASECT before used})
1208:             ELSE ENTRY:=SYM^.FWDREF;
1209:          END;
1210:      END
1211:    ELSE
1212:      BEGIN  {Processing a local label}
1213:        SYMLAST:=FALSE;
1214:        IF CODESECTION=A THEN
1215:          ERROR(44{no local labels in ASECT})
1216:        ELSE IF TEMP[TEMPLABEL].TEMPATRIB<>UNKNOWN THEN
1217:          ERROR(9{identifier previously declared})
1218:        ELSE
1219:          BEGIN
1220:            TEMP[TEMPLABEL].TEMPATRIB:=LABELS;
1221:            TEMP[TEMPLABEL].DEFOFFSET:=LC;
1222:            ENTRY:=TEMP[TEMPLABEL].FWDREF;
1223:            TEMP[TEMPLABEL].FWDREF:=NIL;
1224:          END;
1225:      END;
1226:    IF LEXTOKEN=TLABEL THEN LLCHECK;
1227:    LEX;
1228:    IF LEXTOKEN<>EQU THEN COREFIX(ENTRY,LC);
1229:  END;
1230:
1231:  PROCEDURE ZALIGN;
1232:  {Align handles the .Align psuedo-op. The operand represents the
1233:   boundary multiple on which the next desired code is to start.}
1234:  VAR OFFSET,I:INTEGER;
1235:  BEGIN
1236:    IF EXPRESS(TRUE) THEN
1237:      BEGIN
1238:        OFFSET:=LC MOD RESULT.OFFSETORVALUE;
1239:        IF OFFSET>0 THEN
1240:          BEGIN
1241:            OFFSET:=RESULT.OFFSETORVALUE - OFFSET;
1242:            IF WORDADDRESSED THEN
1243:              FOR I:=1 TO OFFSET DO PUTWORD(0)
1244:            ELSE
1245:              FOR I:=1 TO OFFSET DO PUTBYTE(0);
1246:          END;
1247:      END;
1248:  END;
1249:
1250:  PROCEDURE ZASCII;
1251:  VAR STRINGSIZE,COUNT:INTEGER;
1252:  BEGIN
1253:    LEX;
1254:    IF LEXTOKEN=TSTRING THEN
```

```
1255:       BEGIN
1256:         STRINGSIZE:=LENGTH(STRVAL);
1257:         FOR COUNT:=1 TO STRINGSIZE DO
1258:           BEGIN
1259:             IF DISPLAY THEN
1260:               IF (COUNT MOD BYTEFIT=1) AND (COUNT<>1) THEN
1261:                 BEGIN
1262:                   PRINTLINE;
1263:                   TEXTLINE:=BLANKLINE;
1264:                 END;
1265:             PUTBYTE(ORD(STRVAL[COUNT]));
1266:           END;
1267:       END
1268:     ELSE
1269:       ERROR(10{improper format});
1270:     LEX;
1271:   END;
1272:
1273:   PROCEDURE ZEQU;
1274:   BEGIN
1275:     IF NOT SYMLAST THEN
1276:       ERROR(9{identifier previously declared})
1277:     ELSE
1278:       IF EXPRESS(TRUE) THEN
1279:         BEGIN
1280:           IF CODESECTION=A THEN
1281:             BEGIN
1282:               IF LASTSYM^.ATTRIBUTE<>DEFS THEN LASTSYM^.ATTRIBUTE:=ABS;
1283:             END
1284:           ELSE IF RELOCATE<>NULLREL THEN
1285:             IF RELOCATE.TIPE=LLREL THEN
1286:               IF TEMP[RELOCATE.TEMPLABEL].TEMPATRIB=UNKNOWN THEN
1287:                 ERROR(63)
1288:               ELSE
1289:                 BEGIN
1290:                   IF LASTSYM^.ATTRIBUTE<>DEFS THEN LASTSYM^.ATTRIBUTE:=LABELS;
1291:                 END
1292:             ELSE IF RELOCATE.TIPE=LABELREL THEN
1293:               IF (RELOCATE.SYM^.ATTRIBUTE=LABELS) OR
1294:                 ((RELOCATE.SYM^.ATTRIBUTE=DEFS) AND
1295:                  (RELOCATE.SYM^.CODEOFFSET<>-1)) THEN
1296:                 BEGIN
1297:                   IF LASTSYM^.ATTRIBUTE<>DEFS THEN LASTSYM^.ATTRIBUTE:=LABELS;
1298:                 END
1299:               ELSE ERROR(63{may not EQU to undefined labels})
1300:             ELSE
1301:                 BEGIN
1302:                   IF LASTSYM^.ATTRIBUTE<>DEFS THEN
1303:                     LASTSYM^.ATTRIBUTE:=RESULT.ATTRIBUTE;
1304:                 END
1305:           ELSE
1306:             BEGIN
1307:               IF LASTSYM^.ATTRIBUTE<>DEFS THEN
1308:                 LASTSYM^.ATTRIBUTE:=RESULT.ATTRIBUTE;
1309:             END;
1310:           LASTSYM^.OFFSETORVALUE:=RESULT.OFFSETORVALUE;
1311:           IF LASTSYM^.FWDREF<>NIL THEN
1312:             IF LASTSYM^.ATTRIBUTE=LABELS THEN
1313:               COREFIX(LASTSYM^.FWDREF,LASTSYM^.OFFSETORVALUE)
1314:             ELSE
1315:               ERROR(12{must EQU before use if not a label});
1316:         END;
1317:     SYMLAST:=FALSE;
1318:   END;
1319:
1320:   PROCEDURE ZDEFMACRO;
```

```
1321:   VAR  I:INTEGER;
1322:        ID:PACKNAME;
1323:   BEGIN
1324:     CURRENTATRIB:=MACROS;
1325:     IF SOURCE<>FILESOURCE THEN
1326:       ERROR(61{nested Macro definitions are senseless})
1327:     ELSE
1328:       BEGIN
1329:         LEX;
1330:         IF NOT (LEXTOKEN IN [OP1,OP2,OP3,OP4,OP5,OP6,OP7,OP8,OP9,OP10,
1331:         OP11,OP12,OP13,OP14,OP15,OP16,OP17,OP18,OP19,OP20,TIDENTIFIER]) THEN
1332:            ERROR(13{macro identifier expected});
1333:         SYM^.EXPANDMCRO:=EXPANDMACRO;
1334:         SYM^.ATTRIBUTE:=MACROS;
1335:         NEW(MCPTR); SYM^.MACRO:=MCPTR;       {puts macro on heap}
1336:         REPEAT GETCHAR; UNTIL CH=CHR(13);
1337:         ADVANCE:=FALSE;
1338:         MACROINDEX:=0;  I:=0;  ID:='        ';
1339:         DEFMCHOOK:=TRUE;
1340:         REPEAT
1341:           IF MACROINDEX>MACROSIZE THEN
1342:             BEGIN
1343:               NEW(MCPTR);
1344:               MACROINDEX:=0;
1345:             END;
1346:           GETCHAR;
1347:           IF TEXTINDEX>79 THEN ERROR(6{input line over 80 chars});
1348:           MCPTR^[MACROINDEX]:=CH;
1349:           IF CH=CHR(13) THEN
1350:             BEGIN
1351:               PRINTLINE;
1352:               TEXTLINE:=BLANKLINE; TEXTINDEX:=-1;
1353:             END
1354:           ELSE IF CH='.' THEN
1355:             BEGIN
1356:               I:=0;
1357:               ID:='        ';
1358:             END
1359:           ELSE IF I<5 THEN
1360:             BEGIN
1361:               ID[I]:=CH;
1362:               I:=I + 1;
1363:             END;
1364:           MACROINDEX:=MACROINDEX + 1;
1365:         UNTIL ID='ENDM    ';
1366:         IF MACROINDEX<=MACROSIZE THEN MCPTR^[MACROINDEX]:=CHR(13)
1367:           ELSE
1368:             BEGIN
1369:               NEW(MCPTR);
1370:               MCPTR^[0]:=CHR(13);
1371:             END;
1372:         CURRENTATRIB:=UNKNOWN;
1373:         DEFMCHOOK:=FALSE;
1374:       END;
1375:     LEX;
1376:   END;
1377:
1378:   PROCEDURE ZBLOCK;
1379:   VAR  COUNT,SIZE:INTEGER;
1380:        INITVALUE:WORDSWAP;
1381:   {handles the .BLOCK psuedo-op, the operand is the number
1382:    of bytes/words of storage requested.}
1383:   BEGIN
1384:     IF EXPRESS(TRUE) THEN
1385:       IF CHECKOPERAND(TRUE,TRUE,TRUE,0,BUFLIMIT) THEN
1386:         IF CODESECTION=A THEN
```

```
1387:           BEGIN
1388:             ALC:=ALC + RESULT.OFFSETORVALUE;
1389:             LEX;
1390:           END
1391:         ELSE
1392:           BEGIN
1393:             SIZE:=RESULT.OFFSETORVALUE;
1394:             INITVALUE.HWORD:=0;
1395:             IF LEXTOKEN=COMMA THEN
1396:               IF EXPRESS(FALSE) THEN
1397:                 IF CHECKOPERAND(TRUE,TRUE,TRUE,-128,255) THEN
1398:                   INITVALUE.HWORD:=RESULT.OFFSETORVALUE;
1399:             IF WORDADDRESSED THEN
1400:               FOR COUNT:=1 TO SIZE DO PUTWORD(INITVALUE.LOWBYTE)
1401:             ELSE
1402:               FOR COUNT:=1 TO SIZE DO PUTBYTE(INITVALUE.LOWBYTE);
1403:           END;
1404:   END;
1405:
1406:   PROCEDURE ZWORD;
1407:   VAR  COUNT,INITVALUE:INTEGER;
1408:   BEGIN
1409:     INITVALUE:=0;
1410:     COUNT:=0;
1411:
1412:     IF CODESECTION=A THEN
1413:       BEGIN
1414:         IF WORDADDRESSED THEN ALC:=ALC+1 ELSE ALC:=ALC+2;
1415:         LEX;
1416:       END
1417:     ELSE
1418:       REPEAT
1419:         IF EXPRESS(FALSE) THEN
1420:           IF CHECKOPERAND(TRUE,FALSE,FALSE,0,0) THEN
1421:             INITVALUE:=RESULT.OFFSETORVALUE;
1422:         PUTWORD(INITVALUE);
1423:         IF DISPLAY THEN
1424:           BEGIN
1425:             COUNT:=COUNT + 1;
1426:             IF (COUNT MOD WORDFIT=0) AND (LEXTOKEN=COMMA) THEN
1427:               BEGIN
1428:                 PRINTLINE;
1429:                 FILLCHAR(TEXTLINE[2],70,' ');
1430:               END;
1431:           END;
1432:       UNTIL LEXTOKEN<>COMMA;
1433:   END;
1434:
1435:   PROCEDURE ZBYTE;
1436:   VAR  INITVALUE:WORDSWAP;
1437:        COUNT:INTEGER;
1438:   BEGIN
1439:     IF WORDADDRESSED THEN
1440:       ERROR(14{word addressed only})
1441:     ELSE IF CODESECTION=A THEN
1442:       BEGIN
1443:         ALC:=ALC+1;
1444:         LEX;
1445:       END
1446:     ELSE
1447:       BEGIN
1448:         COUNT:=0;
1449:         REPEAT
1450:           INITVALUE.HWORD:=0;
1451:           IF EXPRESS(FALSE) THEN
1452:             IF CHECKOPERAND(TRUE,TRUE,TRUE,-128,255) THEN
```

```
1453:              INITVALUE.HWORD:=RESULT.OFFSETORVALUE;
1454:           PUTBYTE(INITVALUE.LOWBYTE);
1455:           IF DISPLAY THEN
1456:             BEGIN
1457:               COUNT:=COUNT + 1;
1458:               IF (COUNT MOD BYTEFIT=0) AND (LEXTOKEN=COMMA) THEN
1459:                 BEGIN
1460:                   PRINTLINE;
1461:                   FILLCHAR(TEXTLINE[2],70,' ');
1462:                 END;
1463:             END;
1464:         UNTIL LEXTOKEN<>COMMA;
1465:       END;
1466:   END;
1467:
1468:   PROCEDURE ZORG;
1469:   VAR  I,DIFFERENCE:INTEGER;
1470:   BEGIN
1471:     IF EXPRESS(TRUE) THEN
1472:       IF CHECKOPERAND(TRUE,TRUE,FALSE,0,32767) THEN
1473:         IF CODESECTION=A THEN
1474:           ALC:=RESULT.OFFSETORVALUE
1475:         ELSE
1476:           BEGIN
1477:             IF LC=0 THEN
1478:               BEGIN
1479:                 LC:=RESULT.OFFSETORVALUE;
1480:                 LOWLC:=LC;
1481:               END
1482:             ELSE IF RESULT.OFFSETORVALUE<LC THEN
1483:               ERROR(15{backward ORG not allowed})
1484:             ELSE
1485:               BEGIN
1486:                 DIFFERENCE:=RESULT.OFFSETORVALUE - LC;
1487:                 IF WORDADDRESSED THEN DIFFERENCE:=DIFFERENCE + DIFFERENCE;
1488:                 FOR I:=1 TO DIFFERENCE DO PUTBYTE(0);
1489:               END;
1490:           END;
1491:   END;
1492:
1493:   PROCEDURE ZGLOBAL;
1494:   {Privates are not put into the linker information.}
1495:   VAR SAVESYM:SYMTABLEPTR;
1496:   BEGIN
1497:     CASE LEXTOKEN OF
1498:       TCONST:CURRENTATRIB:=CONSTS;
1499:       PUBLIC:CURRENTATRIB:=PUBLICS;
1500:       PRIVATE:CURRENTATRIB:=PRIVATES;
1501:       REF:CURRENTATRIB:=REFS;
1502:       DEF:CURRENTATRIB:=DEFS
1503:     END;
1504:     REPEAT
1505:       LEX;
1506:       IF LEXTOKEN<>TIDENTIFIER THEN
1507:         ERROR(16{Expected identifier})
1508:       ELSE
1509:         BEGIN
1510:           IF SYM^.ATTRIBUTE<>CURRENTATRIB THEN
1511:             ERROR(9{Identifier previously declared})
1512:           ELSE IF CURRENTATRIB=PRIVATES THEN
1513:             BEGIN
1514:               SAVESYM:=SYM;
1515:               LEX;
1516:               IF LEXTOKEN=COLON THEN
1517:                 BEGIN
1518:                   LEX;
```

```
1519:                   IF LEXTOKEN=CONSTANT THEN
1520:                     SAVESYM^.NWORDS:=CONSTVAL
1521:                   ELSE ERROR(17{Constant expected});
1522:                   LEX;
1523:                 END
1524:               ELSE SAVESYM^.NWORDS:=1;
1525:             END
1526:           ELSE LEX;
1527:         END;
1528:     UNTIL LEXTOKEN<>COMMA;
1529:     CURRENTATRIB:=UNKNOWN;
1530:   END;
1531:
1532:   PROCEDURE ZTITLE;
1533:   BEGIN
1534:     LEX;
1535:     IF LEXTOKEN=TSTRING THEN TITLELINE:=STRVAL
1536:       ELSE ERROR(46{string expected});
1537:     LEX;
1538:   END;
1539:
1540:
1541:
1542:   PROCEDURE GETOPER(VAR XMODE,XREG,INDEX:INTEGER;
1543:                     VAR ISINDEXED,RELATIVE:BOOLEAN);
1544:     VAR MODEADJUST:INTEGER;
1545:     {1: evaluate any exterior address
1546:      2: evaluate register number and set register number
1547:      3: check special stack and set mode
1548:      XMODE,XREG,INDEX and ISINDEXED are variables returned by this routine,
1549:      the routine input is the assembly file.
1550:      XMODE is the address mode of the operand.
1551:      XREG  is the register specified (or implied) by the operand.
1552:      INDEX is the value of the index which is specified by the operand, except
1553:            that where the PC register is implied it is the value of the operand.
1554:      ISINDEXED is true if there is an index specified or if the register is the
1555:            PC. It is true in exactly those cases requiring a second word be
1556:            emitted following the emission of the opcode.}
1557:    BEGIN
1558:     MODEADJUST:=0;
1559:     RELATIVE:=FALSE;
1560:     ISINDEXED:=FALSE;
1561:     XMODE:=0;
1562:     XREG:=0;
1563:     IF EXPRESS(FALSE) THEN
1564:      BEGIN
1565:       ISINDEXED:=TRUE;
1566:       INDEX:=RESULT.OFFSETORVALUE;
1567:       IF RESULT.ATTRIBUTE=DEFABS THEN
1568:        BEGIN{A register stands alone. Check special stack, if it is empty the
1569:              mode is 0 otherwise the mode is 1 and we check for an "@". Then
1570:              load the value of the register}
1571:         IF SPCIALSTKINDEX=-1 THEN XMODE:=0 ELSE
1572:          BEGIN
1573:           XMODE:=1;
1574:           IF (SPECIALSTK[0]<>ATSIGN) OR (SPCIALSTKINDEX<>0) THEN
1575:             ERROR(25{illegal use of special symbols});
1576:           SPCIALSTKINDEX:=-1;
1577:          END;
1578:         XREG:=SYM^.OFFSETORVALUE;
1579:         ISINDEXED:=FALSE;
1580:        END ELSE{Indexed addressing. Operand followed by register enclosed
1581:                 in parentheses. If no register is explicit then the PC
1582:                 register is implied}
1583:         IF LEXTOKEN=OPENPAREN THEN
1584:          BEGIN{check special stack and determine mode then get the register}
```

```
1585:           SPCIALSTKINDEX:=SPCIALSTKINDEX-1;{Peel "(" off stack}
1586:           IF (SPCIALSTKINDEX=0) AND (SPECIALSTK[0]=ATSIGN) THEN
1587:            BEGIN
1588:             MODEADJUST:=1;
1589:             SPCIALSTKINDEX:=-1;
1590:            END;
1591:           XMODE:=6+MODEADJUST;
1592:           LEX;
1593:           IF (LEXTOKEN=TIDENTIFIER) AND (SYM^.ATTRIBUTE=DEFABS) THEN
1594:            BEGIN
1595:             XREG:=SYM^.OFFSETORVALUE;
1596:             LEX;
1597:             IF LEXTOKEN<>CLOSEPAREN THEN ERROR(76{")" expected}) ELSE LEX;
1598:            END ELSE ERROR(77{Register expected});
1599:          END ELSE
1600:          BEGIN{The PC is the implied register, check special stack}
1601:           XREG:=7;
1602:           IF SPCIALSTKINDEX=-1 THEN
1603:            BEGIN{Mode=Relative}
1604:             RELATIVE:=TRUE;
1605:             INDEX:=RESULT.OFFSETORVALUE-4;
1606:             XMODE:=6;
1607:            END ELSE
1608:            BEGIN
1609:             IF SPCIALSTKINDEX=0 THEN
1610:               IF SPECIALSTK[0]=ATSIGN THEN
1611:                BEGIN{Mode=Relative defered}
1612:                 RELATIVE:=TRUE;
1613:                 INDEX:=RESULT.OFFSETORVALUE-4;
1614:                 XMODE:=7;
1615:                END ELSE
1616:                 IF SPECIALSTK[0]=NUMBERSIGN THEN XMODE:=2 ELSE{=Immediate}
1617:                   ERROR(25{Special symbol misused})
1618:               ELSE
1619:               IF SPCIALSTKINDEX=1 THEN
1620:                 IF (SPECIALSTK[0]=ATSIGN) AND
1621:                    (SPECIALSTK[1]=NUMBERSIGN) THEN XMODE:=3 ELSE{=Absolute}
1622:                   ERROR(25{Special symbol misused})
1623:                 ELSE ERROR(78{Too many special symbols});
1624:             SPCIALSTKINDEX:=-1;
1625:            END;
1626:          END
1627:      END ELSE
1628:        IF LEXTOKEN=OPENPAREN THEN{Unindexed use of register. Modes 1..5}
1629:         BEGIN
1630:          SPCIALSTKINDEX:=SPCIALSTKINDEX-1;{Peel off the "("}
1631:          IF (SPCIALSTKINDEX<>-1) AND
1632:             (SPECIALSTK[0]=ATSIGN) THEN MODEADJUST:=1;{Auto Inc/Dec Defered}
1633:          LEX;{get register number}
1634:          IF (LEXTOKEN=TIDENTIFIER) AND (SYM^.ATTRIBUTE=DEFABS) THEN
1635:           BEGIN
1636:            XREG:=SYM^.OFFSETORVALUE;
1637:            LEX;
1638:            IF LEXTOKEN=CLOSEPAREN THEN
1639:             BEGIN
1640:              LEX;
1641:              IF LEXTOKEN=PLUS THEN{Check for auto-increment}
1642:               BEGIN
1643:                LEX;
1644:                XMODE:=2+MODEADJUST
1645:               END ELSE
1646:                IF SPCIALSTKINDEX<>-1 THEN{Check for Auto decrement}
1647:                 BEGIN
1648:                  IF SPECIALSTK[SPCIALSTKINDEX]=AUTODECR THEN
1649:                   BEGIN
1650:                    XMODE:=4+MODEADJUST;
```

```
1651:                    SPCIALSTKINDEX:=SPCIALSTKINDEX-1;
1652:                  END ELSE ERROR(79{Unrecognizable operand});
1653:                END ELSE XMODE:=1;
1654:              END ELSE ERROR(76{")" expected});
1655:            END ELSE ERROR(77{Register expected});
1656:          IF MODEADJUST=1 THEN SPCIALSTKINDEX:=SPCIALSTKINDEX-1;
1657:                           {Peel off the "@"}
1658:         END ELSE ERROR(79{Unrecognizable operand});
1659:    END;
1660:
1661:   PROCEDURE ZOP1;
1662:   {instructions with no operands}
1663:   BEGIN
1664:     IF DEBUG THEN WRITELN('Op1');
1665:     IF ODD(LC) THEN PUTBYTE(NOP);
1666:     OPBYTE.BWORD:=SYM^.OFFSETORVALUE;
1667:     PUTWORD(OPBYTE.BWORD);
1668:     LEX;
1669:   END;
1670:
1671:   PROCEDURE ZOP2;
1672:   {branch - short: opcode..offset in words.}
1673:   BEGIN
1674:     IF DEBUG THEN WRITELN('Op2');
1675:     IF ODD(LC) THEN PUTBYTE(NOP);
1676:     OPBYTE.BWORD:=SYM^.OFFSETORVALUE;
1677:     IF EXPRESS(TRUE) THEN
1678:      BEGIN
1679:       RELOCATE.OFFSETORVALUE:=RELOCATE.OFFSETORVALUE-2;{for putrelword's sake}
1680:       PUTRELWORD(OPBYTE.BWORD,TRUE,TRUE);
1681:      END;
1682:   END;
1683:
1684:   PROCEDURE ZOP3;
1685:     VAR MODE1,REG1,OPINDX1:INTEGER;
1686:         HASINDX1,REL1:BOOLEAN;
1687:   {one operand: opcode..mode..register. CLR,COM,INC,DEC,NEG, Shift & rotates,
1688:    and Multiple precision}
1689:   BEGIN
1690:     IF DEBUG THEN WRITELN('Op3');
1691:     IF ODD(LC) THEN PUTBYTE(NOP);
1692:     OPBYTE.BWORD:=SYM^.OFFSETORVALUE;
1693:     GETOPER(MODE1,REG1,OPINDX1,HASINDX1,REL1);
1694:     OPERAND1:=RELOCATE;
1695:     RELOCATE:=NULLREL;
1696:     OPBYTE.MODELOW:=MODE1;
1697:     OPBYTE.REGLOW:=REG1;
1698:     PUTWORD(OPBYTE.BWORD);
1699:     IF HASINDX1 THEN
1700:      BEGIN
1701:       RELOCATE:=OPERAND1;
1702:       IF REL1 THEN PUTRELWORD(OPINDX1,FALSE,FALSE) ELSE PUTWORD(OPINDX1);
1703:      END;
1704:   END;
1705:
1706:   PROCEDURE ZOP4;
1707:   {one operand: opcode..register. RTS, and Floating-point}
1708:   BEGIN
1709:     IF DEBUG THEN WRITELN('Op4');
1710:     IF ODD(LC) THEN PUTBYTE(NOP);
1711:     OPBYTE.BWORD:=SYM^.OFFSETORVALUE;
1712:     LEX;
1713:     IF SYM^.ATTRIBUTE=DEFABS THEN
1714:      BEGIN
1715:       OPBYTE.REGLOW:=SYM^.OFFSETORVALUE;
1716:       PUTWORD(OPBYTE.BWORD);
```

```
1717:       LEX;
1718:      END ELSE ERROR(80{Register reference only});
1719:   END;
1720:
1721:   PROCEDURE ZOP5;
1722:     VAR MODE1,REG1,OPINDX1:INTEGER;
1723:         HASINDX1,REL1:BOOLEAN;
1724:   {opcode..register..mode..register. Used by XOR,JSR}
1725:   BEGIN
1726:     IF ODD(LC) THEN PUTBYTE(NOP);
1727:     IF DEBUG THEN WRITELN('Op5');
1728:     OPBYTE.BWORD:=SYM^.OFFSETORVALUE;
1729:     LEX;
1730:     IF SYM^.ATTRIBUTE=DEFABS THEN OPBYTE.REGHI:=SYM^.OFFSETORVALUE
1731:        ELSE ERROR(81{First operand must be register});
1732:     LEX;
1733:     IF LEXTOKEN<>COMMA THEN ERROR(82{Comma expected});
1734:     GETOPER(MODE1,REG1,OPINDX1,HASINDX1,REL1);
1735:     OPERAND1:=RELOCATE;
1736:     RELOCATE:=NULLREL;
1737:     OPBYTE.MODELOW:=MODE1;
1738:     OPBYTE.REGLOW:=REG1;
1739:     PUTWORD(OPBYTE.BWORD);
1740:     IF HASINDX1 THEN
1741:      BEGIN
1742:       RELOCATE:=OPERAND1;
1743:       IF REL1 THEN PUTRELWORD(OPINDX1,FALSE,FALSE) ELSE PUTWORD(OPINDX1);
1744:      END;
1745:   END;
1746:
1747:   PROCEDURE ZOP6;
1748:
1749:   {handles MARK}
1750:   BEGIN
1751:     IF DEBUG THEN WRITELN('Op6');
1752:     ERROR(83{Unimplemented instruction});
1753:   END;
1754:
1755:   PROCEDURE ZOP7;
1756:   {handles SOB}
1757:   BEGIN
1758:     IF ODD(LC) THEN PUTBYTE(NOP);
1759:     IF DEBUG THEN WRITELN('Op7');
1760:     OPBYTE.BWORD:=SYM^.OFFSETORVALUE;
1761:     LEX;
1762:     IF SYM^.ATTRIBUTE=DEFABS THEN
1763:      BEGIN
1764:       OPBYTE.REGHI:=SYM^.OFFSETORVALUE;
1765:       LEX;
1766:       IF LEXTOKEN=COMMA THEN
1767:        BEGIN
1768:         IF EXPRESS(TRUE) THEN
1769:          BEGIN
1770:           IF RESULT.ATTRIBUTE=LABELS THEN
1771:            BEGIN
1772:             RESULT.OFFSETORVALUE:=(LC+2-RESULT.OFFSETORVALUE) DIV 2;
1773:             IF CHECKOPERAND(TRUE,FALSE,TRUE,0,64) THEN
1774:              BEGIN
1775:               RELOCATE:=NULLREL;
1776:               OPBYTE.SOBSET:=RESULT.OFFSETORVALUE;
1777:               PUTWORD(OPBYTE.BWORD);
1778:              END;
1779:            END ELSE ERROR(84{Must branch backwards to label});
1780:          END;
1781:        END ELSE ERROR(82{Comma expected});
1782:      END ELSE ERROR(81{First operand must be register});
```

```
1783:   END;
1784:
1785:   PROCEDURE ZOP8;
1786:   {The double operand instructions. MOV,CMP,ADD,SUB and logicals}
1787:     VAR MODE1,REG1,OPINDX1,MODE2,REG2,OPINDX2:INTEGER;
1788:         HASINDX1,REL1,HASINDX2,REL2:BOOLEAN;
1789:    BEGIN
1790:     IF ODD(LC) THEN PUTBYTE(NOP);
1791:     IF DEBUG THEN WRITELN('Op8');
1792:     OPBYTE.BWORD:=SYM^.OFFSETORVALUE;
1793:     GETOPER(MODE1,REG1,OPINDX1,HASINDX1,REL1);
1794:     OPBYTE.MODEHI:=MODE1;
1795:     OPBYTE.REGHI:=REG1;
1796:     OPERAND1:=RELOCATE;
1797:     RELOCATE:=NULLREL;
1798:     IF LEXTOKEN<>COMMA THEN ERROR(82{Comma expected});
1799:     GETOPER(MODE2,REG2,OPINDX2,HASINDX2,REL2);
1800:     OPBYTE.MODELOW:=MODE2;
1801:     OPBYTE.REGLOW:=REG2;
1802:     OPERAND2:=RELOCATE;
1803:     RELOCATE:=NULLREL;
1804:     PUTWORD(OPBYTE.BWORD);
1805:     IF HASINDX1 THEN
1806:      BEGIN
1807:       RELOCATE:=OPERAND1;
1808:       IF REL1 THEN PUTRELWORD(OPINDX1,FALSE,FALSE) ELSE PUTWORD(OPINDX1);
1809:      END;
1810:     IF HASINDX2 THEN
1811:      BEGIN
1812:       RELOCATE:=OPERAND2;
1813:       IF REL2 THEN
1814:        BEGIN
1815:         IF HASINDX1 THEN OPINDX2:=OPINDX2-2;
1816:         PUTRELWORD(OPINDX2,FALSE,FALSE)
1817:        END ELSE PUTWORD(OPINDX2);
1818:      END;
1819:    END;
1820:
1821:   PROCEDURE ZOP9;
1822:     VAR MODE1,REG1,OPINDX1:INTEGER;
1823:         HASINDX1,REL1:BOOLEAN;
1824:   {opcode..register..mode..register. Used by MUL,DIV,ASH,ASHC}
1825:   BEGIN
1826:     IF ODD(LC) THEN PUTBYTE(NOP);
1827:     IF DEBUG THEN WRITELN('Op5');
1828:     OPBYTE.BWORD:=SYM^.OFFSETORVALUE;
1829:     GETOPER(MODE1,REG1,OPINDX1,HASINDX1,REL1);
1830:     IF LEXTOKEN<>COMMA THEN ERROR(82{Comma expected});
1831:     LEX;
1832:     IF SYM^.ATTRIBUTE=DEFABS THEN OPBYTE.REGHI:=SYM^.OFFSETORVALUE
1833:        ELSE ERROR(81{First operand must be register});
1834:     LEX;
1835:     OPERAND1:=RELOCATE;
1836:     RELOCATE:=NULLREL;
1837:     OPBYTE.MODELOW:=MODE1;
1838:     OPBYTE.REGLOW:=REG1;
1839:     PUTWORD(OPBYTE.BWORD);
1840:     IF HASINDX1 THEN
1841:      BEGIN
1842:       RELOCATE:=OPERAND1;
1843:       IF REL1 THEN PUTRELWORD(OPINDX1,FALSE,FALSE) ELSE PUTWORD(OPINDX1);
1844:      END;
1845:   END;
1846:
1847:   PROCEDURE ZOP10;
1848:   {TRAP and EMT}
```

```
1849:   BEGIN
1850:     IF DEBUG THEN WRITELN('Op2');
1851:     IF ODD(LC) THEN PUTBYTE(NOP);
1852:     OPBYTE.BWORD:=SYM^.OFFSETORVALUE;
1853:     IF EXPRESS(TRUE) THEN
1854:       IF CHECKOPERAND(TRUE,TRUE,TRUE,-128,255) THEN
1855:         OPBYTE.GOODBYTE:=RESULT.OFFSETORVALUE;
1856:     PUTWORD(OPBYTE.BWORD);
1857:   END;
1858:
1859:   PROCEDURE ZOP11;
1860:   BEGIN
1861:   END;
1862:
1863:   PROCEDURE ZOP12;
1864:   BEGIN
1865:   END;
1866:
1867:   PROCEDURE ZOP13;
1868:   BEGIN
1869:   END;
1870:
1871:   PROCEDURE ZOP14;
1872:   BEGIN
1873:   END;
1874:
1875:   PROCEDURE ZOP15;
1876:   BEGIN
1877:   END;
1878:
1879:   PROCEDURE ZOP16;
1880:   BEGIN
1881:   END;
1882:
1883:   PROCEDURE ZOP17;
1884:   BEGIN
1885:   END;
1886:
1887:   PROCEDURE ZOP18;
1888:   BEGIN
1889:   END;
1890:
1891:   PROCEDURE ZOP19;
1892:   BEGIN
1893:   END;
1894:
1895:   PROCEDURE ZOP20;
1896:   BEGIN
1897:   END;
1898:
1899:   (*$I ASM4.TEXT*)
1900:                           {start of ASM4}
1901:           {Copyright (c) 1978 Regents of University of California}
1902:
1903:   PROCEDURE ZNOLIST;
1904:   BEGIN
1905:     IF DISPLAY THEN
1906:       BEGIN
1907:         PRINTLINE;
1908:         IF CONSOLE THEN
1909:           BEGIN
1910:             WRITELN;
1911:             WRITE('<',LINENUM:4,'>');
1912:           END;
1913:         DISPLAY:=FALSE;
1914:       END;
```

```
1915:     LEX;
1916:   END;
1917:
1918:   PROCEDURE ZLIST;
1919:   BEGIN
1920:     IF LISTING THEN
1921:       BEGIN
1922:         IF NOT DISPLAY THEN PRINTPAGE;
1923:         DISPLAY:=TRUE;
1924:       END;
1925:     LEX;
1926:   END;
1927:
1928:   BEGIN {Segment Assemble}
1929:     IF VIEWSTACK THEN
1930:       UNITWRITE(3,VIEWDUMMY[-1600],35); {turn on display of stack & heap}
1931:     IF DISPLAY THEN
1932:       WRITELN(LISTFILE,'Memory after initialization:',MEMAVAIL:8);
1933:     REPEAT
1934:       LEX;
1935:       IF (LEXTOKEN=TLABEL) OR (LEXTOKEN=LOCLABEL) THEN ZLABEL;
1936:       IF ((CODESECTION=A) AND
1937:           NOT (LEXTOKEN IN [WORD,BIGHT,BLOCK,EQU,ORG,LIST,NOLIST,PSECT]))
1938:           OR (LEXTOKEN<=FIRSTOPCODE) OR (LEXTOKEN>=LASTOPCODE)
1939:           OR ((PROCNUM=0) AND (LEXTOKEN<=OP20)) THEN
1940:         BEGIN
1941:           ERROR(18{Invalid structure});
1942:           WHILE LEXTOKEN<>ENDLINE DO LEX;
1943:           PRINTLINE;
1944:         END
1945:       ELSE
1946:         BEGIN
1947:           CASE LEXTOKEN OF
1948:             NOLIST:ZNOLIST;
1949:             LIST:ZLIST;
1950:             ASECT:  BEGIN  CODESECTION:=A;  LEX;  END;
1951:             PSECT:  BEGIN  CODESECTION:=P;  LEX;  END;
1952:             ALIGN:ZALIGN;
1953:             ASCII:ZASCII;
1954:             EQU:ZEQU;
1955:             MACRODEF:ZDEFMACRO;
1956:             BLOCK:ZBLOCK;
1957:             WORD:ZWORD;
1958:             BIGHT:ZBYTE;
1959:             ORG:ZORG;
1960:             TPAGE:BEGIN
1961:                     IF DISPLAY THEN PRINTPAGE;
1962:                     LEX;
1963:                   END;
1964:             TITLE:ZTITLE;
1965:             PROC,FUNC,TEND:EXIT(ASSEMBLE);
1966:             TCONST,PUBLIC,PRIVATE,DEF,REF:ZGLOBAL;
1967:             CONDITION:ZCOND;
1968:             TELSE:ZELSE;
1969:             CONDEND:BEGIN
1970:                       IF CONDINDEX<0 THEN ERROR(7{Not enough ifs})
1971:                         ELSE CONDINDEX:=CONDINDEX - 1;
1972:                       LEX;
1973:                     END;
1974:             OP1:ZOP1;
1975:             OP2:ZOP2;
1976:             OP3:ZOP3;
1977:             OP4:ZOP4;
1978:             OP5:ZOP5;
1979:             OP6:ZOP6;
1980:             OP7:ZOP7;
```

```
1981:             OP8:ZOP8;
1982:             OP9:ZOP9;
1983:             OP10:ZOP10;
1984:             OP11:ZOP11;
1985:             OP12:ZOP12;
1986:             OP13:ZOP13;
1987:             OP14:ZOP14;
1988:             OP15:ZOP15;
1989:             OP16:ZOP16;
1990:             OP17:ZOP17;
1991:             OP18:ZOP18;
1992:             OP19:ZOP19;
1993:             OP20:ZOP20
1994:             {ENDLINE is legal yet ignored!}
1995:           END;
1996:         IF SPCIALSTKINDEX<>-1 THEN
1997:           BEGIN
1998:             SPCIALSTKINDEX:=-1;
1999:             ERROR(19{Extra special symbol});
2000:           END;
2001:         IF LEXTOKEN<>ENDLINE THEN
2002:           BEGIN
2003:             ERROR(5{extra garbage on line});
2004:             WHILE LEXTOKEN<>ENDLINE DO LEX;
2005:           END;
2006:         PRINTLINE; SYMLAST:=FALSE;
2007:       END;
2008:     UNTIL FALSE;
2009:   END;
2010:
2011:   PROCEDURE PRERRNUM(ERRORNUM:INTEGER; EXTRA:BOOLEAN); FORWARD;
2012:
2013:   SEGMENT PROCEDURE PRINTERROR(ERRORNUM:INTEGER);
2014:   TYPE  ERRORSTRING=STRING[40];
2015:   VAR  ERRORFILE:FILE OF ERRORSTRING;
2016:        KLUDGEPTR:^INTEGER;
2017:        NAME:STRING;
2018:
2019:   BEGIN
2020:     (*$I-*)
2021:     NAME:=CONCAT('*',ASMNAME);
2022:     RESET(ERRORFILE,CONCAT(NAME,'.ERRORS'));
2023:     MARK(KLUDGEPTR); {dumps disk directory so next proc call won't STK-OFLW}
2024:     (*$I+*)
2025:     IF IORESULT<>0 THEN
2026:       PRERRNUM(ERRORNUM,TRUE)
2027:     ELSE
2028:       BEGIN
2029:         SEEK(ERRORFILE,ERRORNUM);
2030:         GET(ERRORFILE);
2031:         IF DISPLAY THEN
2032:           BEGIN
2033:             WRITELN(LISTFILE);
2034:             WRITELN(LISTFILE,TEXTLINE);
2035:             WRITELN(LISTFILE,ERRORFILE^);
2036:             LISTNUM:=LISTNUM + 3;
2037:           END;
2038:         IF NOT (CONSOLE AND DISPLAY) THEN
2039:           BEGIN
2040:             WRITELN;
2041:             WRITELN(TEXTLINE);
2042:             WRITELN(ERRORFILE^);
2043:           END;
2044:       END;
2045:   END;
2046:
```

```
2047:   PROCEDURE PRERRNUM;  {ERRORNUM:INTEGER; EXTRA:BOOLEAN}
2048:   VAR   LINES:INTEGER;
2049:   BEGIN
2050:     IF DISPLAY THEN
2051:       BEGIN
2052:         WRITELN(LISTFILE);
2053:         WRITELN(LISTFILE,TEXTLINE);
2054:         WRITELN(LISTFILE,'ERROR #',ERRORNUM:4);
2055:         IF EXTRA THEN
2056:           BEGIN
2057:             WRITELN(LISTFILE,'"*',ASMNAME,'.ERRORS" file not around');
2058:             LINES:=4;
2059:           END
2060:         ELSE LINES:=3;
2061:         LISTNUM:=LISTNUM + LINES;
2062:       END;
2063:     IF NOT (CONSOLE AND DISPLAY) THEN
2064:       BEGIN
2065:         WRITELN;
2066:         WRITELN(TEXTLINE);
2067:         WRITELN('ERROR #',ERRORNUM:4);
2068:         IF EXTRA THEN WRITELN('"*',ASMNAME,'.ERRORS" file not around');
2069:       END;
2070:   END;
2071:
2072:   PROCEDURE ERROR; {ERRORNUM:INTEGER}
2073:   VAR  CH:CHAR;
2074:   BEGIN
2075:     NUMERRORS:=NUMERRORS + 1;
2076:     IF MEMAVAIL>1800 THEN
2077:       PRINTERROR(ERRORNUM)
2078:     ELSE
2079:       PRERRNUM(ERRORNUM,FALSE);
2080:     WITH USERINFO DO
2081:       REPEAT
2082:         WRITELN('E(dit,<space>,<esc>');
2083:         READ(KEYBOARD,CH);
2084:         IF (CH=ALTMODE) OR ((ERRORNUM>=47) AND (ERRORNUM<=60)) THEN EXIT(TLA);
2085:         IF (CH='E') OR (CH='e') THEN
2086:           BEGIN
2087:             IF ALTINPUT THEN
2088:               BEGIN
2089:                 ERRSYM:=ALTBLOCPTR;
2090:                 ERRBLK:=ALTBLOCNO-2;
2091:               END
2092:             ELSE
2093:               BEGIN
2094:                 ERRSYM:=BLOCKPTR;
2095:                 ERRBLK:=BLOCKNO-2;
2096:               END;
2097:             ERRNUM:=ERRORNUM;
2098:             EXIT(TLA);
2099:           END;
2100:       UNTIL CH=' ';
2101:     IF NOT (DISPLAY AND CONSOLE) THEN
2102:       BEGIN
2103:         WRITELN;
2104:         WRITE('<',LINENUM:4,'>');
2105:       END;
2106:     IF DISPLAY AND (LISTNUM MOD PAGESIZE<4) THEN PRINTPAGE;
2107:   END;
2108:
2109:   PROCEDURE PATCHCODE; {FWDREF:BACKLABEL; BUFINDEX:INTEGER}
2110:   VAR   PRINTLC:WORDSWAP;
2111:         SWAP:INTEGER;
2112:
```

```
2113:   PROCEDURE PATCHPRINT(BYTESIZE:BOOLEAN);
2114:   BEGIN
2115:     PRINTNUM(FWDREF.LC,FALSE);
2116:     WRITE(LISTFILE,'* ');
2117:     PRINTNUM(PRINTLC.HWORD,BYTESIZE);
2118:     WRITELN(LISTFILE);
2119:   END;
2120:
2121:   BEGIN {PATCHCODE}
2122:     PRINTLC.HWORD:=FWDREF.VALUE;
2123:     IF FWDREF.BYTESIZE THEN
2124:       IF (PRINTLC.HWORD>127) OR (PRINTLC.HWORD<-128) THEN
2125:         BEGIN
2126:           PRINTLC.HWORD:=FWDREF.LC;
2127:           WRITELN('Location ',HEXCHAR[PRINTLC.HEX1],
2128:             HEXCHAR[PRINTLC.HEX2],HEXCHAR[PRINTLC.HEX3],
2129:             HEXCHAR[PRINTLC.HEX4]);
2130:           ERROR(2{operand out of range});
2131:         END
2132:       ELSE
2133:         BEGIN
2134:           BUFFER^[BUFINDEX]:=PRINTLC.LOWBYTE;
2135:           IF DISPLAY THEN PATCHPRINT(TRUE);
2136:         END
2137:     ELSE
2138:       BEGIN
2139:         IF HIBYTEFIRST THEN
2140:           BEGIN
2141:             BUFFER^[BUFINDEX]:=PRINTLC.HIBYTE;
2142:             BUFFER^[BUFINDEX + 1]:=PRINTLC.LOWBYTE;
2143:           END
2144:         ELSE
2145:           BEGIN
2146:             BUFFER^[BUFINDEX]:=PRINTLC.LOWBYTE;
2147:             BUFFER^[BUFINDEX + 1]:=PRINTLC.HIBYTE;
2148:           END;
2149:         IF NOT LISTHIFIRST THEN
2150:           BEGIN
2151:             SWAP:=PRINTLC.HIBYTE;
2152:             PRINTLC.HIBYTE:=PRINTLC.LOWBYTE;
2153:             PRINTLC.LOWBYTE:=SWAP;
2154:           END;
2155:         IF DISPLAY THEN PATCHPRINT(FALSE);
2156:       END;
2157:     IF DISPLAY THEN
2158:       BEGIN
2159:         LISTNUM:=LISTNUM + 1;
2160:         IF (LISTNUM MOD PAGESIZE=0) THEN PRINTPAGE;
2161:       END;
2162:   END;
2163:
2164:   PROCEDURE IOCHECK; {QUIT:BOOLEAN}
2165:   BEGIN
2166:     IF IORESULT<>0 THEN
2167:       BEGIN
2168:         ERROR(46 + IORESULT);
2169:         IF QUIT THEN
2170:           BEGIN
2171:             UNITCLEAR(3); {remove pretty display of stack & heap on screen}
2172:             EXIT(TLA);
2173:           END;
2174:       END;
2175:   END;
2176:
2177:   PROCEDURE LLCHECK;
2178:   VAR  I:INTEGER;
```

```
2179:   BEGIN
2180:     FOR I:=0 TO TEMPTOP-1 DO
2181:       IF TEMP[I].FWDREF<>NIL THEN
2182:         BEGIN
2183:           IF DISPLAY THEN
2184:             BEGIN
2185:               WRITELN(LISTFILE);
2186:               WRITE(LISTFILE,'>>>>>',TEMP[I].TEMPNAME);
2187:             END;
2188:           IF NOT (CONSOLE AND DISPLAY) THEN
2189:             BEGIN
2190:               WRITELN;
2191:               WRITE('>>>>>',TEMP[I].TEMPNAME);
2192:             END;
2193:           ERROR(1{undefined label});
2194:           TEMP[I].FWDREF:=NIL;
2195:         END;
2196:     TEMPTOP:=0;
2197:   END;
2198:
2199:   PROCEDURE PRINTPAGE;
2200:   BEGIN
2201:     IF CONSOLE THEN
2202:       BEGIN
2203:         WRITELN(LISTFILE);
2204:         WRITELN(LISTFILE);
2205:       END
2206:     ELSE PAGE(LISTFILE);
2207:     WRITE(LISTFILE,'PAGE - ',PAGENO:3,'  ',PROCNAME,'  FILE:',CURFNAME);
2208:     IF DISPLAY AND CONSOLE THEN WRITELN(LISTFILE);
2209:     WRITELN(LISTFILE,'  ',TITLELINE);
2210:     WRITELN(LISTFILE);
2211:     WRITELN(LISTFILE);
2212:     LISTNUM:=0;
2213:     PAGENO:=PAGENO + 1;
2214:   END;
2215:
2216:   PROCEDURE PRINTLINE;
2217:   VAR  COUNT:INTEGER;
2218:        LISTLINE:STRING;
2219:   BEGIN
2220:     LINENUM:=LINENUM + 1;
2221:     IF NOT (DISPLAY AND CONSOLE) THEN
2222:       BEGIN
2223:         WRITE('.');
2224:         IF (LINENUM MOD 50=0) THEN
2225:           BEGIN
2226:             WRITELN;
2227:             WRITE('<',LINENUM:4,'>');
2228:           END;
2229:       END;
2230:     IF DISPLAY THEN
2231:       BEGIN
2232:         LISTNUM:=LISTNUM + 1;
2233:         IF (LISTNUM MOD PAGESIZE=0) THEN PRINTPAGE;
2234:         PRINTNUM(LASTLC,FALSE);
2235:         IF CODECOUNT<CODESIZE-2 THEN    {use blank impression code}
2236:           BEGIN
2237:             COUNT:=CODESIZE - CODECOUNT + 1;
2238:             CODE[CODECOUNT]:=CHR(16);
2239:             CODE[CODECOUNT + 1]:=CHR(COUNT + 32);
2240:             MOVELEFT(CODE,LISTLINE[1],CODECOUNT+2);
2241:             LISTLINE[0]:=CHR(CODECOUNT+2);
2242:             WRITE(LISTFILE,'| ',LISTLINE);
2243:           END
2244:         ELSE
```

```
2245:           WRITE(LISTFILE,'| ',CODE);
2246:         IF TEXTINDEX>79 THEN TEXTINDEX:=79;  {caution abounds in unsure minds}
2247:         MOVELEFT(TEXTLINE,LISTLINE[1],TEXTINDEX+1);
2248:         LISTLINE[0]:=CHR(TEXTINDEX+1);
2249:         IF SOURCE=MACROSOURCE THEN
2250:           WRITELN(LISTFILE,'#',LISTLINE)
2251:         ELSE
2252:           WRITELN(LISTFILE,' ',LISTLINE);
2253:       END;
2254:     IF (CODESECTION=A) THEN  LASTLC:=ALC  ELSE  LASTLC:=LC;
2255:     CODE:=BLANKCODE;
2256:     CODECOUNT:=0;
2257:   END;
2258:
2259:   PROCEDURE PRINTNUM; {WORD:INTEGER; BYTESIZE:BOOLEAN}
2260:   VAR NUM:WORDSWAP;
2261:   BEGIN
2262:     NUM.HWORD:=WORD;
2263:     IF BYTESIZE THEN
2264:       BEGIN
2265:         IF LISTRADIX=16 THEN
2266:           WRITE(LISTFILE,HEXCHAR[NUM.HEX3],HEXCHAR[NUM.HEX4]);
2267:         IF LISTRADIX=8 THEN
2268:           WRITE(LISTFILE,NUM.OCT4:1,NUM.OCT5:1,NUM.OCT6:1)
2269:       END
2270:     ELSE
2271:       BEGIN
2272:         IF LISTRADIX=16 THEN WRITE(LISTFILE,HEXCHAR[NUM.HEX1],HEXCHAR[NUM.HEX2],
2273:                                             HEXCHAR[NUM.HEX3],HEXCHAR[NUM.HEX4]);
2274:         IF LISTRADIX=8 THEN WRITE(LISTFILE,NUM.OCT1:1,NUM.OCT2:1,NUM.OCT3:1,
2275:                                            NUM.OCT4:1,NUM.OCT5:1,NUM.OCT6:1)
2276:       END
2277:   END;
2278:
2279:   PROCEDURE PUTBYTE;  {BYTE:BITE}
2280:   VAR  HEX:WORDSWAP;
2281:   BEGIN
2282:     IF BUFFERPOS>BUFLIMIT THEN
2283:       BEGIN
2284:         (*$I-*)
2285:         IF BLOCKWRITE(USERINFO.WORKCODE^,BUFFER^,1,OUTBLKNO)=0 THEN ERROR(54);
2286:         IOCHECK(TRUE);
2287:         (*$I+*)
2288:         OUTBLKNO:=OUTBLKNO + 1;
2289:         IF OUTBLKNO>OUTBLKTOP THEN OUTBLKTOP:=OUTBLKNO;
2290:         MOVELEFT(BUFFER^[512],BUFFER^[0],(BUFBLKS -1)*512);
2291:         BUFFERPOS:=BUFFERPOS - 512;
2292:         BUFBOTTOM:=BUFBOTTOM + 512;
2293:       END;
2294:     BUFFER^[BUFFERPOS]:=BYTE;
2295:     BUFFERPOS:=BUFFERPOS + 1;
2296:     BUFFERTOP:=BUFBOTTOM + BUFFERPOS;
2297:     IF BUFFERTOP>MAXBUFTOP THEN MAXBUFTOP:=BUFFERTOP;
2298:     IF NOT WORDADDRESSED THEN LC:=LC + 1;
2299:     IF DISPLAY AND NOT FROMPUTWORD THEN
2300:       BEGIN
2301:         HEX.HWORD:=BYTE;
2302:         IF LISTRADIX=16 THEN
2303:           IF CODECOUNT + 2<=CODESIZE THEN
2304:             BEGIN
2305:               CODE[CODECOUNT]:=HEXCHAR[HEX.HEX3];
2306:               CODE[CODECOUNT + 1]:=HEXCHAR[HEX.HEX4];
2307:               CODE[CODECOUNT + 2]:=' ';
2308:               CODECOUNT:=CODECOUNT + 3;
2309:             END;
2310:         IF LISTRADIX=8 THEN
```

```
2311:           IF CODECOUNT + 3<=CODESIZE THEN
2312:             BEGIN
2313:               CODE[CODECOUNT]:=CHR(HEX.OCT4 + ORD('0'));
2314:               CODE[CODECOUNT + 1]:=CHR(HEX.OCT5 + ORD('0'));
2315:               CODE[CODECOUNT + 2]:=CHR(HEX.OCT6 + ORD('0'));
2316:               CODE[CODECOUNT + 3]:=' ';
2317:               CODECOUNT:=CODECOUNT + 4;
2318:             END;
2319:       END;
2320:   END;
2321:
2322:   PROCEDURE SENDWORD(NUM:WORDSWAP; ASTRKCODE:INTEGER);
2323:   VAR  SWAP,LISTNUM:WORDSWAP;
2324:   BEGIN
2325:     SWAP:=NUM;
2326:     IF NOT HIBYTEFIRST THEN
2327:       BEGIN
2328:         NUM.HIBYTE:=SWAP.LOWBYTE;
2329:         NUM.LOWBYTE:=SWAP.HIBYTE;
2330:       END;
2331:     IF DISPLAY THEN
2332:       BEGIN
2333:         IF LISTHIFIRST THEN
2334:           LISTNUM:=SWAP
2335:         ELSE
2336:           BEGIN
2337:             LISTNUM:=NUM;
2338:             ASTRKCODE:=ASTRKCODE DIV 2 + (ASTRKCODE MOD 2)*2;
2339:           END;
2340:         IF LISTRADIX=16 THEN
2341:           IF CODECOUNT + 4<=CODESIZE THEN
2342:             BEGIN
2343:               FILLCHAR(CODE[CODECOUNT],4,'*');
2344:               IF ASTRKCODE<2 THEN
2345:                 BEGIN
2346:                   CODE[CODECOUNT]:=HEXCHAR[LISTNUM.HEX1];
2347:                   CODE[CODECOUNT + 1]:=HEXCHAR[LISTNUM.HEX2];
2348:                 END;
2349:               IF (ASTRKCODE MOD 2<>1) THEN
2350:                 BEGIN
2351:                   CODE[CODECOUNT + 2]:=HEXCHAR[LISTNUM.HEX3];
2352:                   CODE[CODECOUNT + 3]:=HEXCHAR[LISTNUM.HEX4];
2353:                 END;
2354:               CODE[CODECOUNT + 4]:=' ';
2355:               CODECOUNT:=CODECOUNT + 5;
2356:             END;
2357:         IF LISTRADIX=8 THEN
2358:           IF CODECOUNT + 6<=CODESIZE THEN
2359:             BEGIN
2360:               FILLCHAR(CODE[CODECOUNT],6,'*');
2361:               IF ASTRKCODE<2 THEN
2362:                 BEGIN
2363:                   CODE[CODECOUNT]:=CHR(LISTNUM.OCT1 + ORD('0'));
2364:                   CODE[CODECOUNT + 1]:=CHR(LISTNUM.OCT2 + ORD('0'));
2365:                   CODE[CODECOUNT + 2]:=CHR(LISTNUM.OCT3 + ORD('0'));
2366:                 END;
2367:               IF (ASTRKCODE MOD 2<>1) THEN
2368:                 BEGIN
2369:                   CODE[CODECOUNT + 3]:=CHR(LISTNUM.OCT4 + ORD('0'));
2370:                   CODE[CODECOUNT + 4]:=CHR(LISTNUM.OCT5 + ORD('0'));
2371:                   CODE[CODECOUNT + 5]:=CHR(LISTNUM.OCT6 + ORD('0'));
2372:                 END;
2373:               CODE[CODECOUNT + 6]:=' ';
2374:               CODECOUNT:=CODECOUNT + 7;
2375:             END;
2376:       END;
```

```
2377:     IF WORDADDRESSED THEN LC:=LC + 1;
2378:     FROMPUTWORD:=TRUE;
2379:     PUTBYTE(NUM.HIBYTE);
2380:     PUTBYTE(NUM.LOWBYTE);
2381:     FROMPUTWORD:=FALSE;
2382:   END;
2383:
2384:   PROCEDURE PUTWORD;  {WORD:INTEGER}
2385:   VAR  NUM,SWAP:WORDSWAP;
2386:        ASTRKCODE:INTEGER;
2387:
2388:   PROCEDURE FULLSET;
2389:   BEGIN
2390:     FULLLABEL^.OFFSET:=BUFFERTOP;
2391:     FULLLABEL^.LC:=LC;
2392:     FULLLABEL^.BYTESIZE:=FALSE;
2393:     FULLLABEL^.WORDLC:=FALSE;
2394:     FULLLABEL^.VALUE:=WORD;
2395:     ASTRKCODE:=3;
2396:   END;
2397:
2398:   PROCEDURE JUMPSET(VAR JCOUNT:INTEGER; VAR JUMP:JTABREC; CLASS:INTEGER);
2399:   BEGIN
2400:     IF JUMPINFO THEN
2401:       BEGIN
2402:         IF JCOUNT=7 THEN
2403:           BEGIN
2404:             SCRATCH^.CLASS:=CLASS;
2405:             SCRATCH^.JUMPS:=JUMP;
2406:             PUT(SCRATCH); SCRATCHEND:=SCRATCHEND + 1;
2407:             FILLCHAR(JUMP,SIZEOF(JUMP),0);
2408:             JCOUNT:=0;
2409:           END;
2410:         JUMP[JCOUNT]:=BUFFERTOP;
2411:         JCOUNT:=JCOUNT + 1;
2412:       END;
2413:   END;
2414:
2415:   BEGIN {PUTWORD}
2416:     ASTRKCODE:=0;
2417:     NUM.HWORD:=WORD;
2418:     CASE RELOCATE.TIPE OF
2419:       NOTSET:;
2420:        LCREL:BEGIN
2421:                RELOCATE:=NULLREL;
2422:                JUMPSET(JCOUNT1,JUMP1,1);
2423:              END;
2424:        LLREL:BEGIN
2425:                IF TEMP[RELOCATE.TEMPLABEL].TEMPATRIB=UNKNOWN THEN
2426:                  BEGIN
2427:                    FULLSET;
2428:                    FULLLABEL^.NEXT:=TEMP[RELOCATE.TEMPLABEL].FWDREF;
2429:                    TEMP[RELOCATE.TEMPLABEL].FWDREF:=FULLLABEL;
2430:                    IF FREELABEL<>NIL THEN
2431:                      BEGIN
2432:                        FULLLABEL:=FREELABEL;
2433:                        FREELABEL:=FREELABEL^.NEXT;
2434:                      END
2435:                    ELSE NEW(FULLLABEL);
2436:                  END;
2437:                JUMPSET(JCOUNT1,JUMP1,1);
2438:                RELOCATE:=NULLREL;
2439:              END;
2440:     LABELREL:BEGIN
2441:                CASE RELOCATE.SYM^.ATTRIBUTE OF
2442:                  LABELS,UNKNOWN,DEFS:
```

```
2443:                     BEGIN
2444:                       IF (RELOCATE.SYM^.ATTRIBUTE=LABELS) OR
2445:                           ((RELOCATE.SYM^.ATTRIBUTE=DEFS) AND
2446:                           (RELOCATE.SYM^.CODEOFFSET<>-1)) THEN
2447:                       ELSE
2448:                         BEGIN
2449:                           FULLSET;
2450:                           IF RELOCATE.SYM^.ATTRIBUTE=DEFS THEN
2451:                            BEGIN
2452:                             FULLLABEL^.NEXT:=RELOCATE.SYM^.DEFFWDREF;
2453:                             RELOCATE.SYM^.DEFFWDREF:=FULLLABEL
2454:                            END
2455:                           ELSE
2456:                            BEGIN
2457:                             FULLLABEL^.NEXT:=RELOCATE.SYM^.FWDREF;
2458:                             RELOCATE.SYM^.FWDREF:=FULLLABEL;
2459:                            END;
2460:                           IF FREELABEL<>NIL THEN
2461:                             BEGIN
2462:                               FULLLABEL:=FREELABEL;
2463:                               FREELABEL:=FREELABEL^.NEXT;
2464:                             END
2465:                           ELSE NEW(FULLLABEL);
2466:                         END;
2467:                       JUMPSET(JCOUNT1,JUMP1,1);
2468:                     END;
2469:                 PRIVATES,PUBLICS,CONSTS,REFS:
2470:                     BEGIN
2471:                       RELOCATE.SYM^.NREFS:=RELOCATE.SYM^.NREFS + 1;
2472:                       NEW(NEXTJP); NEXTJP^.PCOFFSET:=BUFFERTOP-512;
2473:                       NEXTJP^.LAST:=RELOCATE.SYM^.LINKOFFSET;
2474:                       RELOCATE.SYM^.LINKOFFSET:=NEXTJP;
2475:                       CASE RELOCATE.SYM^.ATTRIBUTE OF
2476:                         PUBLICS,PRIVATES:  JUMPSET(JCOUNT3,JUMP3,3);
2477:                         REFS:   JUMPSET(JCOUNT2,JUMP2,2)
2478:                       END;
2479:                     END;
2480:               END;
2481:               RELOCATE:=NULLREL;
2482:             END
2483:       END;{Main Case}
2484:     SENDWORD(NUM,ASTRKCODE);
2485:   END;
2486:
2487:   PROCEDURE PUTRELWORD; {WORD:INTEGER; BYTESIZE,WORDOFFSET:BOOLEAN}
2488:   VAR NUM,SWAP:WORDSWAP;
2489:       ASTRKCODE:INTEGER;
2490:
2491:   PROCEDURE FULLRELSET;
2492:   BEGIN
2493:     FULLLABEL^.OFFSET:=BUFFERTOP;
2494:     FULLLABEL^.LC:=LC;
2495:     FULLLABEL^.WORDLC:=WORDOFFSET;
2496:   END;
2497:
2498:   PROCEDURE SHORTSPACE;
2499:   BEGIN
2500:     IF TEMP[RELOCATE.TEMPLABEL].TEMPATRIB=UNKNOWN THEN
2501:       BEGIN
2502:         FULLRELSET;
2503:         IF BYTESIZE THEN
2504:           BEGIN
2505:             IF RELHI THEN ASTRKCODE:=2 ELSE ASTRKCODE:=1;
2506:             IF (RELHI AND NOT HIBYTEFIRST) OR
2507:               (NOT RELHI AND HIBYTEFIRST) THEN
2508:                 FULLLABEL^.OFFSET:=BUFFERTOP + 1;
```

```
2509:             FULLLABEL^.BYTESIZE:=TRUE;
2510:             IF RELOCATE.ATTRIBUTE=LABELS THEN
2511:               FULLLABEL^.VALUE:=RELOCATE.OFFSETORVALUE - LASTLC;
2512:           END
2513:         ELSE
2514:           BEGIN
2515:             ASTRKCODE:=3;
2516:             FULLLABEL^.BYTESIZE:=FALSE;
2517:             IF RELOCATE.ATTRIBUTE=LABELS THEN
2518:               FULLLABEL^.VALUE:=WORD - LASTLC;
2519:           END;
2520:         FULLLABEL^.NEXT:=TEMP[RELOCATE.TEMPLABEL].FWDREF;
2521:         TEMP[RELOCATE.TEMPLABEL].FWDREF:=FULLLABEL;
2522:         IF FREELABEL<>NIL THEN
2523:           BEGIN
2524:             FULLLABEL:=FREELABEL;
2525:             FREELABEL:=FREELABEL^.NEXT;
2526:           END
2527:         ELSE NEW(FULLLABEL);
2528:       END
2529:     ELSE
2530:       IF BYTESIZE THEN
2531:         BEGIN
2532:           IF RELOCATE.ATTRIBUTE=LABELS THEN
2533:             SWAP.HWORD:=RELOCATE.OFFSETORVALUE-LASTLC
2534:           ELSE
2535:             SWAP.HWORD:=RELOCATE.OFFSETORVALUE;
2536:           IF NOT WORDADDRESSED AND WORDOFFSET THEN
2537:             SWAP.HWORD:=SWAP.HWORD DIV 2;
2538:           IF (SWAP.HWORD>=-128) AND (SWAP.HWORD<=127) THEN
2539:             IF RELHI THEN
2540:               NUM.HIBYTE:=SWAP.LOWBYTE
2541:             ELSE
2542:               NUM.LOWBYTE:=SWAP.LOWBYTE
2543:           ELSE ERROR(20{branch too far});
2544:         END
2545:       ELSE
2546:         IF RELOCATE.ATTRIBUTE=LABELS THEN
2547:           NUM.HWORD:=WORD - LASTLC;
2548:     RELOCATE:=NULLREL;
2549:   END;
2550:
2551:   BEGIN {PUTRELWORD}
2552:     ASTRKCODE:=0;
2553:     NUM.HWORD:=WORD;
2554:     CASE RELOCATE.TIPE OF
2555:       NOTSET:IF BYTESIZE THEN
2556:                BEGIN
2557:                  SWAP.HWORD:=RELOCATE.OFFSETORVALUE;
2558:                  IF NOT WORDADDRESSED AND WORDOFFSET THEN
2559:                    SWAP.HWORD:=SWAP.HWORD DIV 2;
2560:                  IF (SWAP.HWORD>=-128) AND (SWAP.HWORD<=127) THEN
2561:                    IF RELHI THEN
2562:                      NUM.HIBYTE:=SWAP.LOWBYTE
2563:                    ELSE
2564:                      NUM.LOWBYTE:=SWAP.LOWBYTE
2565:                  ELSE ERROR(20{branch too far});
2566:                END;
2567:        LCREL:BEGIN
2568:                IF BYTESIZE THEN
2569:                  BEGIN
2570:                    IF RELOCATE.ATTRIBUTE=LABELS THEN {not ABS}
2571:                      SWAP.HWORD:=RELOCATE.OFFSETORVALUE-LASTLC
2572:                    ELSE
2573:                      SWAP.HWORD:=RELOCATE.OFFSETORVALUE;
2574:                    IF NOT WORDADDRESSED AND WORDOFFSET THEN
```

```
2575:                     SWAP.HWORD:=SWAP.HWORD DIV 2;
2576:                   IF (SWAP.HWORD>=-128) AND (SWAP.HWORD<=127) THEN
2577:                     IF RELHI THEN
2578:                       NUM.HIBYTE:=SWAP.LOWBYTE
2579:                     ELSE
2580:                       NUM.LOWBYTE:=SWAP.LOWBYTE
2581:                   ELSE ERROR(20{branch too far});
2582:                 END
2583:               ELSE
2584:                 IF RELOCATE.ATTRIBUTE=LABELS THEN
2585:                   NUM.HWORD:=WORD - LASTLC;
2586:               RELOCATE:=NULLREL;
2587:             END;
2588:       LLREL:SHORTSPACE;
2589:    LABELREL:BEGIN
2590:               CASE RELOCATE.SYM^.ATTRIBUTE OF
2591:                 LABELS,UNKNOWN,DEFS:
2592:                   BEGIN
2593:                     IF (RELOCATE.SYM^.ATTRIBUTE=LABELS) OR
2594:                         ((RELOCATE.SYM^.ATTRIBUTE=DEFS) AND
2595:                         (RELOCATE.SYM^.CODEOFFSET<>-1)) THEN
2596:                       BEGIN
2597:                         IF BYTESIZE THEN
2598:                           BEGIN
2599:                             IF RELOCATE.ATTRIBUTE=LABELS THEN
2600:                               SWAP.HWORD:=RELOCATE.OFFSETORVALUE-LASTLC
2601:                             ELSE
2602:                               SWAP.HWORD:=RELOCATE.OFFSETORVALUE;
2603:                             IF NOT WORDADDRESSED AND WORDOFFSET THEN
2604:                               SWAP.HWORD:=SWAP.HWORD DIV 2;
2605:                             IF (SWAP.HWORD>=-128) AND (SWAP.HWORD<=127) THEN
2606:                               IF RELHI THEN
2607:                                 NUM.HIBYTE:=SWAP.LOWBYTE
2608:                               ELSE
2609:                                 NUM.LOWBYTE:=SWAP.LOWBYTE
2610:                             ELSE ERROR(20{branch too far});
2611:                           END
2612:                         ELSE
2613:                           IF RELOCATE.ATTRIBUTE=LABELS THEN
2614:                             NUM.HWORD:=WORD - LASTLC;
2615:                       END
2616:                     ELSE
2617:                       BEGIN
2618:                         FULLRELSET;
2619:                         IF BYTESIZE THEN
2620:                           BEGIN
2621:                             IF RELHI THEN ASTRKCODE:=2 ELSE ASTRKCODE:=1;
2622:                             IF (RELHI AND NOT HIBYTEFIRST) OR
2623:                               (NOT RELHI AND HIBYTEFIRST) THEN
2624:                                 FULLLABEL^.OFFSET:=BUFFERTOP + 1;
2625:                             FULLLABEL^.BYTESIZE:=TRUE;
2626:                             IF RELOCATE.ATTRIBUTE=LABELS THEN
2627:                               FULLLABEL^.VALUE:=RELOCATE.OFFSETORVALUE-LASTLC;
2628:                           END
2629:                         ELSE
2630:                           BEGIN
2631:                             ASTRKCODE:=3;
2632:                             FULLLABEL^.BYTESIZE:=FALSE;
2633:                             FULLLABEL^.VALUE:=WORD-LASTLC;
2634:                           END;
2635:                         IF RELOCATE.SYM^.ATTRIBUTE=DEFS THEN
2636:                          BEGIN
2637:                           FULLLABEL^.NEXT:=RELOCATE.SYM^.DEFFWDREF;
2638:                           RELOCATE.SYM^.DEFFWDREF:=FULLLABEL;
2639:                          END
2640:                         ELSE
```

```
2641:                           BEGIN
2642:                             FULLLABEL^.NEXT:=RELOCATE.SYM^.FWDREF;
2643:                             RELOCATE.SYM^.FWDREF:=FULLLABEL;
2644:                           END;
2645:                         IF FREELABEL<>NIL THEN
2646:                           BEGIN
2647:                             FULLLABEL:=FREELABEL;
2648:                             FREELABEL:=FREELABEL^.NEXT;
2649:                           END
2650:                         ELSE NEW(FULLLABEL);
2651:                       END;
2652:                   END;
2653:                 PRIVATES,PUBLICS,CONSTS,REFS:
2654:                     BEGIN
2655:                       IF DISPLAY THEN
2656:                         BEGIN
2657:                           WRITELN(LISTFILE);
2658:                           WRITE(LISTFILE,RELOCATE.SYM^.NAME);
2659:                         END;
2660:                       IF NOT (CONSOLE AND DISPLAY) THEN
2661:                         BEGIN
2662:                           WRITELN;
2663:                           WRITE(RELOCATE.SYM^.NAME);
2664:                         END;
2665:                       ERROR(21{Variable not PC relative});
2666:                     END
2667:               END;
2668:               RELOCATE:=NULLREL;
2669:             END
2670:       END;{Main Case}
2671:     SENDWORD(NUM,ASTRKCODE);
2672:   END;
2673:
2674:
2675:   (*$I ASM5.TEXT*)
2676:                           {starting ASM5}
2677:           {Copyright (c) 1978 Regents of University of California}
2678:
2679:   PROCEDURE NEWFILE;
2680:   BEGIN
2681:     (*$I-*)
2682:     TEXTLINE:=BLANKLINE; TEXTINDEX:=0;
2683:     IF ALTINPUT THEN
2684:       BNUM:=BLOCKREAD(ALTFILE,XBLOCK,2,BLOCKNO)
2685:     ELSE
2686:       BNUM:=BLOCKREAD(USERINFO.WORKSRC^,XBLOCK,2,BLOCKNO);
2687:     BLOCKPTR:=0;
2688:     BLOCKNO:=BLOCKNO+BNUM;
2689:     IF DEBUG THEN
2690:        WRITELN('BLOCKREAD=',BLOCKNO);
2691:     IF BNUM=0 THEN
2692:       IF ALTINPUT THEN
2693:         BEGIN
2694:           BLOCKNO:=ALTBLOCNO;
2695:           BLOCKPTR:=ALTBLOCPTR;
2696:           BNUM:=BLOCKREAD(USERINFO.WORKSRC^,XBLOCK,2,BLOCKNO - 2);
2697:           ALTINPUT:=FALSE;
2698:           CLOSE(ALTFILE);
2699:           CURFNAME:=FIRSTFNAME;
2700:         END
2701:       ELSE
2702:         BEGIN
2703:           ERROR(36);
2704:           UNITCLEAR(3);
2705:           EXIT(LEX);
2706:         END;
```

```
2707:     IOCHECK(TRUE);
2708:     (*$I+*)
2709:   END;
2710:
2711:   PROCEDURE GETCHAR;
2712:   VAR I:INTEGER;
2713:   BEGIN
2714:     IF DEBUG THEN WRITE(LISTFILE,'Getchar ');
2715:     CASE SOURCE OF
2716:       MACROSOURCE:BEGIN
2717:                     IF ADVANCE THEN
2718:                       BEGIN
2719:                         MACROINDEX:=MACROINDEX + 1;
2720:                         TEXTINDEX:=TEXTINDEX + 1;
2721:                       END
2722:                     ELSE ADVANCE:=TRUE;
2723:                     IF MCPTR^[MACROINDEX]=CHR(16) THEN
2724:                       BEGIN
2725:                         CH:=MCPTR^[MACROINDEX + 1];
2726:                         STARTLINE:=(ORD(CH) - 32=0);
2727:                         IF TEXTINDEX<79 THEN
2728:                           BEGIN
2729:                             TEXTLINE[TEXTINDEX]:=CHR(16);
2730:                             TEXTLINE[TEXTINDEX + 1]:=CH;
2731:                             TEXTINDEX:=TEXTINDEX+2;
2732:                           END;
2733:                         MACROINDEX:=MACROINDEX + 2;
2734:                       END;
2735:                     CH:=MCPTR^[MACROINDEX];
2736:                     IF CH='%' THEN
2737:                       BEGIN
2738:                         CH:=MCPTR^[MACROINDEX + 1];
2739:                         MACROINDEX:=MACROINDEX + 2;
2740:                         IF (CH<'1') OR (CH>'9') THEN
2741:                           ERROR(22{illegal macro parameter index})
2742:                         ELSE
2743:                           BEGIN
2744:                             I:=ORD(CH)-ORD('1');
2745:                             PARMPTR:=MCINDEX[MCSTKINDEX-1];
2746:                             IF MCSTKINDEX>1 THEN
2747:                               BEGIN
2748:                                 MCPTR:=MACROSTACK[MCSTKINDEX - 1];
2749:                                 WHILE (I<>0) AND (MCPTR^[PARMPTR]<>CHR(13)) DO
2750:                                   BEGIN
2751:                                     IF MCPTR^[PARMPTR]=',' THEN I:=I-1;
2752:                                     PARMPTR:=PARMPTR + 1;
2753:                                   END;
2754:                                 I:=SCAN(80,<>' ',MCPTR^[PARMPTR]);
2755:                                 PARMPTR:=PARMPTR + I;
2756:                                 CH:=MCPTR^[PARMPTR];
2757:                                 IF (CH=CHR(13)) OR (CH=';') THEN
2758:                                   MCPTR:=MACROSTACK[MCSTKINDEX];
2759:                               END
2760:                             ELSE
2761:                               BEGIN
2762:                                 WHILE (I<>0) AND (XBLOCK[PARMPTR]<>CHR(13)) DO
2763:                                   BEGIN
2764:                                     IF XBLOCK[PARMPTR]=',' THEN I:=I-1;
2765:                                     PARMPTR:=PARMPTR + 1;
2766:                                   END;
2767:                                 I:=SCAN(80,<>' ',XBLOCK[PARMPTR]);
2768:                                 PARMPTR:=PARMPTR + I;
2769:                                 CH:=XBLOCK[PARMPTR];
2770:                               END;
2771:                            IF (CH<>CHR(13)) AND (CH<>';') THEN SOURCE:=PARMSOURCE;
2772:                             ADVANCE:=FALSE;
```

```
2773:                           GETCHAR;
2774:                         END;
2775:                     END
2776:                   ELSE IF (CH=' ') AND NOTSTRING THEN
2777:                     BEGIN
2778:                       I:=SCAN(80,<>' ',MCPTR^[MACROINDEX]);
2779:                       IF TEXTINDEX + I<80 THEN
2780:                         BEGIN
2781:                           FILLCHAR(TEXTLINE[TEXTINDEX],I,' ');
2782:                           TEXTINDEX:=TEXTINDEX + I - 1;
2783:                         END;
2784:                       MACROINDEX:=MACROINDEX + I - 1;
2785:                     END
2786:                   ELSE IF (EXPANDMACRO) AND (CH<>CHR(13)) THEN
2787:                     BEGIN
2788:                       IF TEXTINDEX<80 THEN TEXTLINE[TEXTINDEX]:=CH;
2789:                       IF CH=TAB THEN CH:=' ';
2790:                     END;
2791:                 END;
2792:       PARMSOURCE:BEGIN
2793:                   IF ADVANCE THEN
2794:                     BEGIN
2795:                       PARMPTR:=PARMPTR + 1;
2796:                       TEXTINDEX:=TEXTINDEX + 1;
2797:                     END
2798:                   ELSE ADVANCE:=TRUE;
2799:                   IF MCSTKINDEX>1 THEN CH:=MCPTR^[PARMPTR]
2800:                     ELSE CH:=XBLOCK[PARMPTR];
2801:                   IF (CH=',') OR (CH=CHR(13)) OR (CH=';') THEN
2802:                     BEGIN
2803:                       IF MCSTKINDEX>1 THEN
2804:                         I:=SCAN(-70,<>' ',MCPTR^[PARMPTR - 1])
2805:                       ELSE
2806:                         I:=SCAN(-70,<>' ',XBLOCK[PARMPTR - 1]);
2807:                       TEXTINDEX:=TEXTINDEX + I;
2808:                       SOURCE:=MACROSOURCE;
2809:                       MCPTR:=MACROSTACK[MCSTKINDEX];
2810:                       ADVANCE:=FALSE;
2811:                       GETCHAR;
2812:                     END
2813:                   ELSE IF (CH=' ') AND NOTSTRING THEN
2814:                     BEGIN
2815:                       REPEAT
2816:                         IF TEXTINDEX<80 THEN TEXTLINE[TEXTINDEX]:=' ';
2817:                         TEXTINDEX:=TEXTINDEX + 1;
2818:                         PARMPTR:=PARMPTR + 1;
2819:                         IF MCSTKINDEX>1 THEN CH:=MCPTR^[PARMPTR]
2820:                           ELSE CH:=XBLOCK[PARMPTR];
2821:                       UNTIL CH<>' ';
2822:                       CH:=' ';
2823:                       PARMPTR:=PARMPTR - 1;
2824:                       TEXTINDEX:=TEXTINDEX - 1;
2825:                     END
2826:                   ELSE
2827:                     BEGIN
2828:                       IF TEXTINDEX<80 THEN TEXTLINE[TEXTINDEX]:=CH;
2829:                       IF CH=TAB THEN CH:=' ';
2830:                     END;
2831:                 END;
2832:       FILESOURCE:BEGIN
2833:                   IF ADVANCE THEN
2834:                     BEGIN
2835:                       BLOCKPTR:=BLOCKPTR + 1;
2836:                       TEXTINDEX:=TEXTINDEX + 1;
2837:                     END
2838:                   ELSE ADVANCE:=TRUE;
```

```
2839:                     IF BLOCKPTR>1023 THEN NEWFILE
2840:                       ELSE IF (XBLOCK[BLOCKPTR]=CHR(0)) THEN NEWFILE;
2841:                     IF (XBLOCK[BLOCKPTR]=CHR(16)) AND NOT DEFMCHOOK THEN
2842:                       BEGIN
2843:                         CH:=XBLOCK[BLOCKPTR+1];
2844:                         STARTLINE:=(ORD(CH) - 32=0);
2845:                         IF TEXTINDEX<79 THEN
2846:                           BEGIN
2847:                             TEXTLINE[TEXTINDEX]:=CHR(16);
2848:                             TEXTLINE[TEXTINDEX + 1]:=CH;
2849:                             TEXTINDEX:=TEXTINDEX + 2;
2850:                           END;
2851:                         BLOCKPTR:=BLOCKPTR+2;
2852:                       END;
2853:                     CH:=XBLOCK[BLOCKPTR];
2854:                     IF CH=';' THEN
2855:                       BEGIN
2856:                         I:=SCAN(80,=CHR(13),XBLOCK[BLOCKPTR]);
2857:                         IF TEXTINDEX+I<80 THEN
2858:                           BEGIN
2859:                             MOVELEFT(XBLOCK[BLOCKPTR],TEXTLINE[TEXTINDEX],I);
2860:                             TEXTINDEX:=TEXTINDEX + I - 1;
2861:                           END;
2862:                         BLOCKPTR:=BLOCKPTR + I;
2863:                         CH:=CHR(13);
2864:                       END
2865:                     ELSE IF (CH=' ') AND NOTSTRING AND NOT DEFMCHOOK THEN
2866:                       BEGIN
2867:                         I:=SCAN(80,<>' ',XBLOCK[BLOCKPTR]);
2868:                         IF TEXTINDEX+I<80 THEN
2869:                           BEGIN
2870:                             FILLCHAR(TEXTLINE[TEXTINDEX],I,' ');
2871:                             TEXTINDEX:=TEXTINDEX + I - 1;
2872:                           END;
2873:                         BLOCKPTR:=BLOCKPTR + I - 1;
2874:                       END
2875:                     ELSE IF CH<>CHR(13) THEN
2876:                       BEGIN
2877:                         IF TEXTINDEX<80 THEN TEXTLINE[TEXTINDEX]:=CH;
2878:                         IF CH=TAB THEN CH:=' ';
2879:                       END;
2880:                   END
2881:         END;{CASE}
2882:       IF DEBUG THEN WRITELN(LISTFILE,'CH=',CH,'|ORD:',ORD(CH),
2883:             ' FROM:',ORD(SOURCE));
2884:     END;
2885:
2886:     FUNCTION CHECKOPERAND; {CKSPSTK,CKABS,CKRANGE:BOOLEAN; LO,HI:INTEGER}
2887:     {Tests the result of an operand for correctness}
2888:     BEGIN
2889:       IF CKABS AND NOT (RESULT.ATTRIBUTE IN [ABS,DEFABS,DEFREG,DEFRP,DEFCC,DEFIR])
2890:        THEN
2891:         BEGIN
2892:           ERROR(24{operand not absolute});
2893:           CHECKOPERAND:=FALSE;
2894:         END
2895:       ELSE IF CKRANGE AND
2896:             ((RESULT.OFFSETORVALUE<LO) OR (RESULT.OFFSETORVALUE>HI)) THEN
2897:         BEGIN
2898:           ERROR(2{operand out of range});
2899:           CHECKOPERAND:=FALSE;
2900:         END
2901:       ELSE IF CKSPCSTK AND (SPCIALSTKINDEX<>-1) THEN
2902:         BEGIN
2903:           ERROR(25{illegal use of special symbols});
2904:           SPCIALSTKINDEX:=-1;
```

```
2905:         CHECKOPERAND:=TRUE {operand maybe ok - just warning}
2906:       END
2907:     ELSE CHECKOPERAND:=TRUE;
2908:   END;
2909:
2910:   FUNCTION EXPRESS; {OPERANDREQUIRED:BOOLEAN}
2911:   TYPE  STACKTYPE=PACKED RECORD     {expression evaluator stack}
2912:           TIPE:TOKENS;
2913:           ATRIB:ATRIBUTETYPE;
2914:           VALUE:INTEGER
2915:         END;
2916:   VAR  STKINDEX,COUNT:INTEGER;
2917:        STK:ARRAY[0..10] OF STACKTYPE;
2918:        UNDEFINED:BOOLEAN;
2919:   {The value and type of the calculation should be returned in the
2920:    variable record RESULT}
2921:
2922:   PROCEDURE EXPREXIT;
2923:   BEGIN
2924:     ERROR(26{ill formed expression});
2925:     WHILE (LEXTOKEN<>TEOF) AND (LEXTOKEN<>ENDLINE) DO LEX;
2926:     EXPRESS:=FALSE;
2927:     EXIT(EXPRESS);
2928:   END;
2929:
2930:   PROCEDURE EXPREND;
2931:   BEGIN
2932:     IF (LEXTOKEN IN [OPENPAREN,EQUAL,NOTEQUAL]) THEN
2933:       BEGIN
2934:         SPCIALSTKINDEX:=SPCIALSTKINDEX + 1;
2935:         SPECIALSTK[SPCIALSTKINDEX]:=LEXTOKEN;
2936:       END;
2937:     IF STKINDEX=-1 THEN
2938:       IF LEXTOKEN=OPENPAREN THEN
2939:         BEGIN
2940:           EXPRESS:=FALSE;
2941:           EXIT(EXPRESS);
2942:         END
2943:       ELSE IF OPERANDREQUIRED THEN
2944:         BEGIN
2945:           ERROR(27{not enough operands});
2946:           EXPRESS:=FALSE;
2947:         END
2948:       ELSE EXPRESS:=FALSE
2949:     ELSE IF (STKINDEX=0) AND (STK[STKINDEX].TIPE=TNULL) THEN
2950:       BEGIN
2951:         RESULT.OFFSETORVALUE:=STK[STKINDEX].VALUE;
2952:         RESULT.ATTRIBUTE:=STK[STKINDEX].ATRIB;
2953:         RELOCATE.ATTRIBUTE:=RESULT.ATTRIBUTE;
2954:         RELOCATE.OFFSETORVALUE:=RESULT.OFFSETORVALUE;
2955:         EXPRESS:=TRUE
2956:       END
2957:     ELSE IF (STKINDEX=1) AND (STK[0].TIPE=TNULL) AND
2958:      (STK[STKINDEX].TIPE IN [PLUS,MINUS,ASTERISK]) THEN
2959:       BEGIN
2960:         SPCIALSTKINDEX:=SPCIALSTKINDEX + 1;
2961:         CASE STK[STKINDEX].TIPE OF
2962:           PLUS:SPECIALSTK[SPCIALSTKINDEX]:=AUTOINCR;
2963:           MINUS:SPECIALSTK[SPCIALSTKINDEX]:=AUTODECR;
2964:           ASTERISK:SPECIALSTK[SPCIALSTKINDEX]:=LEXTOKEN
2965:         END;
2966:         RESULT.OFFSETORVALUE:=STK[0].VALUE;
2967:         RESULT.ATTRIBUTE:=STK[0].ATRIB;
2968:         RELOCATE.ATTRIBUTE:=RESULT.ATTRIBUTE;
2969:         RELOCATE.OFFSETORVALUE:=RESULT.OFFSETORVALUE;
2970:         EXPRESS:=TRUE;
```

```
2971:       END
2972:     ELSE EXPRESS:=FALSE;
2973:     EXIT(EXPRESS);
2974:   END;
2975:
2976:   PROCEDURE OPERFOLD;
2977:   VAR  LATTRIBUTE,RATTRIBUTE:ATRIBUTETYPE;
2978:        KLUDGETYPE:TOKENS;
2979:        RVALUE:INTEGER;
2980:        BOTHABSOLUTE:BOOLEAN;
2981:
2982:   BEGIN
2983:     IF (STKINDEX=0) THEN
2984:       EXIT(OPERFOLD)
2985:     ELSE IF (STK[STKINDEX-1].TIPE=OPNBROKEN) THEN
2986:       EXIT(OPERFOLD)
2987:     ELSE IF STKINDEX>=2 THEN
2988:       BEGIN
2989:         IF STK[STKINDEX-2].TIPE=TNULL THEN
2990:         BEGIN
2991:           LATTRIBUTE:=STK[STKINDEX-2].ATRIB;
2992:           RATTRIBUTE:=STK[STKINDEX].ATRIB;
2993:           IF (LATTRIBUTE IN [DEFABS,DEFRP,DEFREG,DEFCC]) THEN LATTRIBUTE:=ABS;
2994:           IF (RATTRIBUTE IN [DEFABS,DEFRP,DEFREG,DEFCC]) THEN RATTRIBUTE:=ABS;
2995:           BOTHABSOLUTE:=((LATTRIBUTE=ABS) AND (RATTRIBUTE=ABS));
2996:           RVALUE:=STK[STKINDEX].VALUE;
2997:           KLUDGETYPE:=STK[STKINDEX-1].TIPE;
2998:           WITH STK[STKINDEX-2] DO
2999:           BEGIN
3000:             IF NOT (KLUDGETYPE IN [PLUS,MINUS,BITWISEOR,AMPERSAND,
3001:               EXCLUSIVEOR,ASTERISK,DIVIDE,MODULO]) THEN
3002:                 EXPREXIT
3003:             ELSE CASE KLUDGETYPE OF
3004:               PLUS:IF (LATTRIBUTE=ABS) OR (RATTRIBUTE=ABS) THEN
3005:                       BEGIN
3006:                         VALUE:=VALUE + RVALUE;
3007:                         IF RATTRIBUTE<>ABS THEN ATRIB:=RATTRIBUTE;
3008:                       END
3009:                    ELSE EXPREXIT;
3010:              MINUS:IF (RATTRIBUTE=ABS) OR
3011:                       ((RATTRIBUTE<>ABS) AND (LATTRIBUTE=RATTRIBUTE)) THEN
3012:                       BEGIN
3013:                         VALUE:=VALUE - RVALUE;
3014:                         IF RATTRIBUTE<>ABS THEN ATRIB:=ABS;
3015:                       END
3016:                    ELSE EXPREXIT;
3017:           BITWISEOR:IF BOTHABSOLUTE THEN
3018:                       VALUE:=ORD(ODD(VALUE) OR ODD(RVALUE))
3019:                    ELSE EXPREXIT;
3020:           AMPERSAND:IF BOTHABSOLUTE THEN
3021:                       VALUE:=ORD(ODD(VALUE) AND ODD(RVALUE))
3022:                    ELSE EXPREXIT;
3023:         EXCLUSIVEOR:IF BOTHABSOLUTE THEN
3024:                       VALUE:=ORD((ODD(VALUE) AND NOT ODD(RVALUE)) OR
3025:                           (NOT ODD(VALUE) AND ODD(RVALUE)))
3026:                    ELSE EXPREXIT;
3027:            ASTERISK:IF BOTHABSOLUTE THEN
3028:                       VALUE:=VALUE*RVALUE
3029:                    ELSE EXPREXIT;
3030:              DIVIDE:IF BOTHABSOLUTE THEN
3031:                       VALUE:=VALUE DIV RVALUE
3032:                    ELSE EXPREXIT;
3033:              MODULO:IF BOTHABSOLUTE THEN
3034:                       VALUE:=VALUE MOD RVALUE
3035:                    ELSE EXPREXIT
3036:             END;{CASE}
```

```
3037:           END;{WITH}
3038:           STKINDEX:=STKINDEX-2;
3039:         END ELSE EXPREXIT;
3040:       END
3041:     ELSE IF STK[STKINDEX].ATRIB=ABS THEN {check for unary operator}
3042:       BEGIN
3043:         CASE STK[STKINDEX-1].TIPE OF
3044:           MINUS:STK[STKINDEX-1].VALUE:=-STK[STKINDEX].VALUE;
3045:           PLUS:STK[STKINDEX-1].VALUE:=STK[STKINDEX].VALUE;
3046:           ONESCOMPLEMENT:STK[STKINDEX-1].VALUE:=-STK[STKINDEX].VALUE - 1
3047:         END;
3048:         STKINDEX:=STKINDEX - 1;
3049:         STK[STKINDEX].TIPE:=TNULL;
3050:         STK[STKINDEX].ATRIB:=ABS;
3051:       END
3052:     ELSE EXPREXIT; {whatever he wanted i couldn't do}
3053:   END;
3054:
3055:   BEGIN  {EXPRESS}
3056:     RELOCATE:=NULLREL;
3057:     STKINDEX:=-1;
3058:     REPEAT
3059:       IF EXPRSSADVANCE THEN LEX
3060:         ELSE EXPRSSADVANCE:=TRUE;
3061:       IF NOT (LEXTOKEN IN [PLUS,MINUS,BITWISEOR,AMPERSAND,EXCLUSIVEOR,
3062:         ASTERISK,DIVIDE,MODULO,AUTOINCR,AUTODECR,EQUAL,NOTEQUAL,
3063:         ENDLINE,COMMA,OPNBROKEN,OPENPAREN,NUMBERSIGN,ATSIGN,LOCCTR,
3064:         TNOT,CLOSEPAREN,CLSBROKEN,ONESCOMPLEMENT,
3065:         CONSTANT,TSTRING,LOCLABEL,TIDENTIFIER]) THEN EXPREXIT
3066:       ELSE
3067:       CASE LEXTOKEN OF
3068:         PLUS,MINUS,BITWISEOR,AMPERSAND,EXCLUSIVEOR,
3069:         DIVIDE,MODULO,OPNBROKEN,ONESCOMPLEMENT:
3070:                   BEGIN
3071:                     STKINDEX:=STKINDEX + 1;
3072:                     STK[STKINDEX].TIPE:=LEXTOKEN;
3073:                   END;
3074:           ASTERISK:IF STKINDEX=-1 THEN
3075:                     IF LCCHAR='*' THEN
3076:                       BEGIN
3077:                         STKINDEX:=STKINDEX + 1;
3078:                         IF CODESECTION=A THEN
3079:                           STK[STKINDEX].VALUE:=ALC
3080:                         ELSE STK[STKINDEX].VALUE:=LASTLC;
3081:                         RELOCATE.TIPE:=LCREL;
3082:                         STK[STKINDEX].ATRIB:=LABELS;
3083:                         STK[STKINDEX].TIPE:=TNULL;
3084:                         OPERFOLD;
3085:                       END
3086:                     ELSE
3087:                       BEGIN
3088:                         SPCIALSTKINDEX:=SPCIALSTKINDEX + 1;
3089:                         SPECIALSTK[SPCIALSTKINDEX]:=LEXTOKEN;
3090:                       END
3091:                   ELSE
3092:                     BEGIN
3093:                       STKINDEX:=STKINDEX + 1;
3094:                       STK[STKINDEX].TIPE:=LEXTOKEN;
3095:                     END;
3096:             LOCCTR:BEGIN
3097:                     STKINDEX:=STKINDEX + 1;
3098:                     IF CODESECTION=A THEN
3099:                       STK[STKINDEX].VALUE:=ALC
3100:                     ELSE STK[STKINDEX].VALUE:=LASTLC;
3101:                     IF RELOCATE=NULLREL THEN
3102:                       RELOCATE.TIPE:=LCREL
```

```
3103:                     ELSE IF RELOCATE.TIPE=LCREL THEN
3104:                       RELOCATE:=NULLREL;
3105:                     STK[STKINDEX].ATRIB:=LABELS;
3106:                     STK[STKINDEX].TIPE:=TNULL;
3107:                     OPERFOLD;
3108:                   END;
3109:   CONSTANT,TSTRING:BEGIN
3110:                     STKINDEX:=STKINDEX + 1;
3111:                     STK[STKINDEX].VALUE:=0;
3112:                     IF LEXTOKEN=CONSTANT THEN
3113:                       STK[STKINDEX].VALUE:=CONSTVAL
3114:                     ELSE IF LENGTH(STRVAL)<=2 THEN
3115:                       FOR COUNT:=1 TO LENGTH(STRVAL) DO
3116:                         STK[STKINDEX].VALUE:=
3117:                           STK[STKINDEX].VALUE*256 + ORD(STRVAL[COUNT])
3118:                     ELSE EXPREXIT;
3119:                     STK[STKINDEX].ATRIB:=ABS; {Constants are absolute}
3120:                     STK[STKINDEX].TIPE:=TNULL;
3121:                     OPERFOLD;
3122:                   END;
3123:         LOCLABEL: BEGIN
3124:                     IF (RELOCATE<>NULLREL) AND (RELOCATE.TIPE<>LCREL) THEN
3125:                       BEGIN
3126:                         IF TEMP[TEMPLABEL].TEMPATRIB=UNKNOWN THEN
3127:                           ERROR(28{cannot handle this relative});
3128:                       END
3129:                     ELSE
3130:                       BEGIN
3131:                         RELOCATE.TIPE:=LLREL;
3132:                         RELOCATE.TEMPLABEL:=TEMPLABEL;
3133:                       END;
3134:                     STKINDEX:=STKINDEX + 1;
3135:                     STK[STKINDEX].VALUE:=TEMP[TEMPLABEL].DEFOFFSET;
3136:                     STK[STKINDEX].ATRIB:=LABELS;
3137:                     STK[STKINDEX].TIPE:=TNULL;
3138:                     OPERFOLD;
3139:                   END;
3140:      TIDENTIFIER: BEGIN
3141:                     UNDEFINED:=FALSE;
3142:                     STKINDEX:=STKINDEX + 1;
3143:                     IF SYM^.ATTRIBUTE IN
3144:                       [ABS,DEFABS,DEFRP,DEFREG,DEFCC,DEFIR,LABELS] THEN
3145:                         STK[STKINDEX].VALUE:=SYM^.OFFSETORVALUE
3146:                     ELSE IF (SYM^.ATTRIBUTE=DEFS) AND (SYM^.CODEOFFSET<>-1) THEN
3147:                       STK[STKINDEX].VALUE:=SYM^.CODEOFFSET
3148:                     ELSE
3149:                       BEGIN
3150:                         STK[STKINDEX].VALUE:=0;
3151:                         UNDEFINED:=TRUE;
3152:                       END;
3153:                     IF (SYM^.ATTRIBUTE<>UNKNOWN) AND (SYM^.ATTRIBUTE<>DEFS) THEN
3154:                       STK[STKINDEX].ATRIB:=SYM^.ATTRIBUTE
3155:                     ELSE
3156:                       STK[STKINDEX].ATRIB:=LABELS;
3157:                     IF NOT (SYM^.ATTRIBUTE IN
3158:                                 [ABS,DEFABS,DEFRP,DEFREG,DEFCC,DEFIR])
3159:                      THEN
3160:                       BEGIN
3161:                         IF (RELOCATE<>NULLREL) AND (RELOCATE.TIPE<>LCREL) THEN
3162:                           BEGIN
3163:                             IF UNDEFINED THEN
3164:                               ERROR(28{cannot handle this relative});
3165:                           END
3166:                         ELSE
3167:                           BEGIN
3168:                             RELOCATE.TIPE:=LABELREL;
```

```
3169:                             RELOCATE.SYM:=SYM;
3170:                           END;
3171:                       END;
3172:                     STK[STKINDEX].TIPE:=TNULL;
3173:                     OPERFOLD;
3174:                   END;
3175:         ENDLINE,COMMA,OPENPAREN,EQUAL,NOTEQUAL:
3176:                   EXPREND;
3177:         NUMBERSIGN,ATSIGN,TNOT,AUTOINCR,AUTODECR,CLOSEPAREN:
3178:                   BEGIN
3179:                     SPCIALSTKINDEX:=SPCIALSTKINDEX + 1;
3180:                     SPECIALSTK[SPCIALSTKINDEX]:=LEXTOKEN;
3181:                   END;
3182:          CLSBROKEN:BEGIN
3183:                     IF STKINDEX=0 THEN EXPREXIT;
3184:                     IF (STK[STKINDEX-1].TIPE<>OPNBROKEN) THEN EXPREXIT;
3185:                     STK[STKINDEX-1].VALUE:=STK[STKINDEX].VALUE;
3186:                     STK[STKINDEX-1].ATRIB:=STK[STKINDEX].ATRIB;
3187:                     STK[STKINDEX-1].TIPE:=STK[STKINDEX].TIPE;
3188:                     STKINDEX:=STKINDEX - 1;
3189:                     IF (STK[STKINDEX].TIPE<>TNULL) THEN EXPREXIT;
3190:                     OPERFOLD;
3191:                   END
3192:       END; {CASE STATEMENT}
3193:     UNTIL FALSE;
3194:   END;
3195:
3196:
3197:   (*$I ASM6.TEXT*)
3198:                           {start of ASM6}
3199:           {Copyright (c) 1978 Regents of University of California}
3200:
3201:   PROCEDURE LEX;
3202:
3203:   PROCEDURE PCONST;
3204:   VAR RADIX,I,NUM:INTEGER;
3205:       TEMP,ID:STRING;
3206:       VAL:WORDSWAP;
3207:   BEGIN
3208:     IF DEBUG THEN WRITELN('Pcon');
3209:     TEMP:=' '; ID:=' ';
3210:     WHILE (((CH>='A') AND (CH<='F')) OR ((CH>='0') AND (CH<='9'))) DO
3211:       BEGIN
3212:         IF CH>='A' THEN TEMP[1]:=CHR(ORD(CH)-55)
3213:           ELSE TEMP[1]:=CHR(ORD(CH)-ORD('0'));
3214:         ID:=CONCAT(ID,TEMP);
3215:         GETCHAR;
3216:       END;
3217:     REPEAT
3218:       DELETE(ID,1,1);
3219:     UNTIL (ORD(ID[1])<>0) OR (LENGTH(ID)=1);
3220:     IF ORD(CH)=ORD(HEXSWITCH) THEN
3221:       RADIX:=16
3222:     ELSE IF ORD(CH)=ORD(DECSWITCH) THEN
3223:       RADIX:=10
3224:     ELSE IF ORD(CH)=ORD(OCTSWITCH) THEN
3225:       RADIX:=8
3226:     ELSE IF ORD(CH)=ORD(BINSWITCH) THEN
3227:       RADIX:=2
3228:     ELSE
3229:       BEGIN
3230:         RADIX:=DEFRADIX;
3231:         ADVANCE:=FALSE;
3232:       END;
3233:     LEXTOKEN:=CONSTANT;
3234:     TEMP[1]:=CHR(0);
```

```
3235:     CONSTVAL:=0;
3236:     CASE RADIX OF
3237:       16:IF LENGTH(ID)>4 THEN
3238:            ERROR(29{constant overflow})
3239:          ELSE
3240:            BEGIN
3241:              WHILE LENGTH(ID)<4 DO ID:=CONCAT(TEMP,ID);
3242:              VAL.HEX1:=ORD(ID[1]);
3243:              VAL.HEX2:=ORD(ID[2]);
3244:              VAL.HEX3:=ORD(ID[3]);
3245:              VAL.HEX4:=ORD(ID[4]);
3246:              CONSTVAL:=VAL.HWORD;
3247:            END;
3248:       10:IF LENGTH(ID)>5 THEN
3249:            ERROR(29{constant overflow})
3250:          ELSE
3251:            BEGIN
3252:              WHILE LENGTH(ID)<5 DO ID:=CONCAT(TEMP,ID);
3253:              NUM:=0;
3254:              FOR I:=1 TO 4 DO
3255:                IF ORD(ID[I])>9 THEN
3256:                  BEGIN
3257:                    ERROR(30{illegal decimal constant});
3258:                    EXIT(PCONST);
3259:                  END
3260:                ELSE NUM:=NUM*10 + ORD(ID[I]);
3261:              IF (NUM>3276) OR ((NUM=3276) AND (ORD(ID[5])>7)) THEN
3262:                ERROR(29{constant overflow})
3263:              ELSE CONSTVAL:=NUM*10 + ORD(ID[5]);
3264:            END;
3265:        8:IF (LENGTH(ID)>6) OR ((ORD(ID[1])>1) AND (LENGTH(ID)=6)) THEN
3266:            ERROR(29{constant overflow})
3267:          ELSE
3268:            BEGIN
3269:              WHILE LENGTH(ID)<6 DO ID:=CONCAT(TEMP,ID);
3270:              FOR I:=2 TO 6 DO
3271:                IF ORD(ID[I])>7 THEN
3272:                  BEGIN
3273:                    ERROR(31{illegal octal constant});
3274:                    EXIT(PCONST);
3275:                  END;
3276:              VAL.OCT1:=ORD(ID[1]);
3277:              VAL.OCT2:=ORD(ID[2]);
3278:              VAL.OCT3:=ORD(ID[3]);
3279:              VAL.OCT4:=ORD(ID[4]);
3280:              VAL.OCT5:=ORD(ID[5]);
3281:              VAL.OCT6:=ORD(ID[6]);
3282:              CONSTVAL:=VAL.HWORD;
3283:            END;
3284:        2:IF (LENGTH(ID)>16) THEN
3285:            ERROR(29{constant overflow})
3286:          ELSE
3287:            BEGIN
3288:              WHILE LENGTH(ID)<16 DO ID:=CONCAT(TEMP,ID);
3289:              FOR I:=1 TO 16 DO
3290:                IF ORD(ID[I])>1 THEN
3291:                  BEGIN
3292:                    ERROR(32{illegal binary constant});
3293:                    EXIT(PCONST);
3294:                  END
3295:                ELSE VAL.BIN[16 - I]:=ORD(ID[I]);
3296:              CONSTVAL:=VAL.HWORD;
3297:            END
3298:     END; {Case}
3299:   END;
3300:
```

```
3301:           {Looks up the reserved word in the KWORD array and returns the correct
3302:            token for that key word. Only the LEXTOKEN is returned}
3303:
3304:   PROCEDURE PKWORD;
3305:   VAR I:INTEGER; KLUDGEPTR:^INTEGER;
3306:       ID:PACKNAME;
3307:       TEMP,ALTNAME:STRING;
3308:   BEGIN
3309:     IF DEBUG THEN WRITELN('PKW');
3310:     GETCHAR;{Skip over the period}
3311:     ID:='        ';
3312:     I:=0;
3313:     WHILE (((CH>='A') AND (CH<='Z')) OR ((CH>='0') AND (CH<='9'))) DO
3314:       BEGIN
3315:         IF I<8 THEN ID[I]:=CH;
3316:         I:=I+1;
3317:         GETCHAR;
3318:       END;
3319:     IF I=0 THEN ERROR(45{Keyword expected});
3320:     I:=-1;
3321:     FOUND:=FALSE;
3322:     WHILE NOT FOUND AND (I<NUMKWORDS) DO
3323:       BEGIN
3324:         I:=I+1;
3325:         FOUND:=(KWORDS[I]=ID);
3326:       END;
3327:     IF NOT FOUND THEN
3328:      BEGIN
3329:       WRITELN('>',ID,'<');
3330:       ERROR(33{invalid key word})
3331:      END ELSE
3332:       LEXTOKEN:=KTOKEN[I];
3333:     ADVANCE:=FALSE;
3334:     IF ID='ENDM    ' THEN {macro end}
3335:       BEGIN
3336:         MCSTKINDEX:=MCSTKINDEX - 1;
3337:         IF MCSTKINDEX>0 THEN
3338:           BEGIN
3339:             MCPTR:=MACROSTACK[MCSTKINDEX];
3340:             MACROINDEX:=MCINDEX[MCSTKINDEX];
3341:             WHILE MCPTR^[MACROINDEX]<>CHR(13) DO MACROINDEX:=MACROINDEX + 1;
3342:           END
3343:         ELSE
3344:           BEGIN
3345:             SOURCE:=FILESOURCE;
3346:             WHILE XBLOCK[BLOCKPTR]<>CHR(13) DO BLOCKPTR:=BLOCKPTR + 1;
3347:           END;
3348:         REPEAT
3349:           LEX;
3350:         UNTIL (LEXTOKEN=ENDLINE) OR (LEXTOKEN=TEOF);
3351:         IF LEXTOKEN=TEOF THEN
3352:           ERROR(34{Unexpected end of input - after macro})
3353:         ELSE LEX;
3354:       END
3355:     ELSE IF LEXTOKEN=INCLUDE THEN
3356:       IF ALTINPUT THEN
3357:         ERROR(35{Include files may not be nested})
3358:       ELSE IF SOURCE<>FILESOURCE THEN
3359:         ERROR(37{This is a bad place for an include file})
3360:       ELSE
3361:         BEGIN
3362:           ALTINPUT:=TRUE;
3363:           TEMP:=' '; ALTNAME:=' ';
3364:           REPEAT
3365:             GETCHAR;
3366:             IF (CH<>' ') AND (CH<>CHR(13)) THEN
```

```
3367:               BEGIN
3368:                 TEMP[1]:=CH;
3369:                 ALTNAME:=CONCAT(ALTNAME,TEMP);
3370:               END;
3371:           UNTIL CH=CHR(13);
3372:           ALTBLOCNO:=BLOCKNO;
3373:           ALTBLOCPTR:=BLOCKPTR;
3374:           (*$I-*)
3375:           RESET(ALTFILE,ALTNAME);
3376:           IOCHECK(TRUE);
3377:           (*$I+*)
3378:           MARK(KLUDGEPTR);{dumps disk direc so next proc call won't STK-OFLW}
3379:           CURFNAME:=ALTNAME;
3380:           BLOCKNO:=2; BLOCKPTR:=1024;
3381:           LEXTOKEN:=ENDLINE;
3382:           IF NOT (CONSOLE AND DISPLAY) THEN
3383:             BEGIN
3384:               WRITELN;
3385:               WRITELN(TEXTLINE);
3386:               WRITE('<',LINENUM:4,'>');
3387:             END;
3388:         END;
3389:   END;
3390:
3391:         {Search the symbol tree to locate the identifier and determine
3392:          what it is. The types returned can be: OPCODE1..10,TIDENTIFIER,
3393:          if start-line is true then we return the token type of TLABEL}
3394:
3395:   PROCEDURE PIDENT;
3396:   VAR HASHA,HASHB,I:INTEGER;
3397:       ID:PACKNAME;
3398:
3399:   BEGIN
3400:     IF DEBUG THEN WRITELN('PID');
3401:     ID:='        ';
3402:     I:=0;
3403:     WHILE ((CH>='A') AND (CH<='Z')) OR ((CH>='0') AND (CH<='9')) OR (CH='_') DO
3404:       BEGIN
3405:         IF I<8 THEN ID[I]:=CH;
3406:         I:=I+1;
3407:         GETCHAR;
3408:       END;
3409:     HASHA:=0; FOUND:=FALSE;
3410:     FOR I:=0 TO 7 DO
3411:       BEGIN
3412:         HASHA:=HASHA + HASHA; {left shift}
3413:         HASHB:=ORD(ID[I]);
3414:         HASHA:=ORD((NOT ODD(HASHA) AND ODD(HASHB)) OR
3415:                         (ODD(HASHA) AND NOT ODD(HASHB))); {xor}
3416:       END;
3417:     HASHB:=HASHA MOD HASHRANGE; {lo-order part}
3418:     HASHA:=HASHA DIV HASHRANGE; {hi-order part}
3419:     HASHA:=ORD((NOT ODD(HASHA) AND ODD(HASHB)) OR
3420:                 (ODD(HASHA) AND NOT ODD(HASHB)));
3421:     HASHA:=HASHA MOD HASHRANGE;
3422:     SYM:=HASH[HASHA];
3423:     WHILE (NOT FOUND) AND (SYM<>NIL) DO
3424:       IF SYM^.NAME=ID THEN FOUND:=TRUE ELSE SYM:=SYM^.LINK;
3425:     IF NOT FOUND THEN
3426:       BEGIN
3427:         IF DEBUG THEN WRITELN('not found',ORD(CURRENTATRIB):3);
3428:                {insert at the top of the list}
3429:         CASE CURRENTATRIB OF
3430:           MACROS:
3431:             BEGIN
3432:               NEW(SYM,MACROS);
```

```
3433:              SYM^.EXPANDMCRO:=EXPANDMACRO;
3434:            END;
3435:          DEFS:
3436:            BEGIN
3437:              NEW(SYM,DEFS);
3438:              SYM^.PROCNUM:=PROCNUM;
3439:              SYM^.CODEOFFSET:=-1;
3440:              SYM^.DEFFWDREF:=NIL;
3441:            END;
3442:          PUBLICS,PRIVATES,REFS,CONSTS:
3443:            BEGIN
3444:              CASE CURRENTATRIB OF
3445:                PUBLICS:NEW(SYM,PUBLICS);
3446:                PRIVATES:NEW(SYM,PRIVATES);
3447:                REFS:NEW(SYM,REFS);
3448:                CONSTS:NEW(SYM,CONSTS)
3449:              END;
3450:              SYM^.NREFS:=0;
3451:              SYM^.NWORDS:=1;
3452:              SYM^.LINKOFFSET:=NIL;
3453:            END;
3454:          PROCS:NEW(SYM,PROCS);
3455:          FUNCS:NEW(SYM,FUNCS);
3456:          UNKNOWN:
3457:            BEGIN
3458:              NEW(SYM,UNKNOWN);
3459:              SYM^.OFFSETORVALUE:=0;
3460:              SYM^.FWDREF:=NIL;
3461:            END
3462:          END;
3463:        SYM^.NAME:=ID; SYM^.ATTRIBUTE:=CURRENTATRIB;
3464:        SYM^.LINK:=HASH[HASHA];
3465:        HASH[HASHA]:=SYM;
3466:      END
3467:    ELSE IF SYM^.ATTRIBUTE=MACROS THEN
3468:      BEGIN
3469:        IF MCSTKINDEX>0 THEN
3470:          MCINDEX[MCSTKINDEX]:=MACROINDEX
3471:        ELSE
3472:          BEGIN
3473:            MCINDEX[MCSTKINDEX]:=BLOCKPTR;
3474:            EXPANDMACRO:=SYM^.EXPANDMCRO;
3475:          END;
3476:        WHILE CH<>CHR(13) DO GETCHAR;
3477:        PRINTLINE;
3478:        SOURCE:=MACROSOURCE;
3479:        MCSTKINDEX:=MCSTKINDEX + 1;
3480:        MACROSTACK[MCSTKINDEX]:=SYM^.MACRO;
3481:        MCPTR:=SYM^.MACRO;
3482:        MACROINDEX:=0;
3483:        LEXTOKEN:=ENDLINE;
3484:        LEX; {re-initiate LEX with appropriate SOURCE then exit to return called}
3485:        EXIT(LEX); {LEX's LEXTOKEN.  style - 0, effeciency - 1}
3486:      END;
3487:    IF STARTLINE THEN
3488:     BEGIN
3489:      IF DEBUG THEN WRITELN('STARTLINE true');
3490:      IF CH=':' THEN GETCHAR;
3491:      IF NOT FOUND THEN LEXTOKEN:=TLABEL
3492:        ELSE IF (SYM^.ATTRIBUTE=UNKNOWN) OR (SYM^.ATTRIBUTE=DEFS) THEN
3493:               LEXTOKEN:=TLABEL
3494:                ELSE ERROR(38{only labels & comments may occupy column one});
3495:     END
3496:      ELSE
3497:        IF (SYM^.ATTRIBUTE>=OPS1) AND (SYM^.ATTRIBUTE<=OPS20) THEN
3498:          CASE SYM^.ATTRIBUTE OF
```

```
3499:              OPS1: LEXTOKEN:=OP1;
3500:              OPS2: LEXTOKEN:=OP2;
3501:              OPS3: LEXTOKEN:=OP3;
3502:              OPS4: LEXTOKEN:=OP4;
3503:              OPS5: LEXTOKEN:=OP5;
3504:              OPS6: LEXTOKEN:=OP6;
3505:              OPS7: LEXTOKEN:=OP7;
3506:              OPS8: LEXTOKEN:=OP8;
3507:              OPS9: LEXTOKEN:=OP9;
3508:             OPS10: LEXTOKEN:=OP10;
3509:             OPS11: LEXTOKEN:=OP11;
3510:             OPS12: LEXTOKEN:=OP12;
3511:             OPS13: LEXTOKEN:=OP13;
3512:             OPS14: LEXTOKEN:=OP14;
3513:             OPS15: LEXTOKEN:=OP15;
3514:             OPS16: LEXTOKEN:=OP16;
3515:             OPS17: LEXTOKEN:=OP17;
3516:             OPS18: LEXTOKEN:=OP18;
3517:             OPS19: LEXTOKEN:=OP19;
3518:             OPS20: LEXTOKEN:=OP20
3519:           END
3520:             ELSE LEXTOKEN:=TIDENTIFIER;
3521:     IF DEBUG THEN WRITELN('PASSED=',SYM^.NAME,' VALUE=',
3522:                                    ORD(SYM^.ATTRIBUTE):5,HASHA:10);
3523:     ADVANCE:=FALSE;
3524:   END;
3525:
3526:         {A $ has been encountered and we are now processing a local label}
3527:
3528:   PROCEDURE PLLABEL;
3529:   VAR I:INTEGER;
3530:       ID:PACKNAME;
3531:   BEGIN
3532:     IF DEBUG THEN WRITELN('PLLAB');
3533:     ID:='        ';
3534:     I:=0;
3535:     WHILE (CH>='0') AND (CH<='9') DO
3536:       BEGIN
3537:         IF I<8 THEN ID[I]:=CH;
3538:         I:=I+1;
3539:         GETCHAR;
3540:       END;
3541:     IF I=0 THEN ERROR(39{expected local label});
3542:     FOUND:=FALSE;
3543:     TEMPLABEL:=0;
3544:     WHILE NOT FOUND AND (TEMPLABEL<TEMPTOP) DO
3545:       IF TEMP[TEMPLABEL].TEMPNAME=ID THEN
3546:         FOUND:=TRUE
3547:       ELSE
3548:         TEMPLABEL:=TEMPLABEL+1;
3549:     IF NOT FOUND THEN
3550:       IF TEMPTOP=21 THEN
3551:         BEGIN
3552:           ERROR(40{Local label stack overflow});
3553:           EXIT(TLA);
3554:         END
3555:       ELSE
3556:          BEGIN
3557:            TEMP[TEMPTOP].TEMPNAME:=ID;
3558:            TEMP[TEMPTOP].TEMPATRIB:=UNKNOWN;
3559:            TEMP[TEMPTOP].DEFOFFSET:=0;
3560:            TEMP[TEMPTOP].FWDREF:=NIL;
3561:            TEMPTOP:=TEMPTOP+1;
3562:          END;
3563:     LEXTOKEN:=LOCLABEL;
3564:     IF STARTLINE AND (CH=':') THEN GETCHAR;
```

```
3565:     ADVANCE:=FALSE;
3566:   END;
3567:
3568:         {Returns the value of a string constant in STRVAL and sets
3569:          LEXTOKEN to TSTRING. Checks for the closing double quote}
3570:
3571:   PROCEDURE PSTRING;
3572:   VAR I:INTEGER;
3573:       BACKSCAN:BOOLEAN;
3574:       SCH:STRING;
3575:   BEGIN
3576:     IF DEBUG THEN WRITELN('PSTR');
3577:     LEXTOKEN:=TSTRING;
3578:     NOTSTRING:=FALSE;
3579:     BACKSCAN:=FALSE;
3580:     SCH:=' ';
3581:     STRVAL:='';
3582:     GETCHAR;
3583:     I:=0;
3584:     WHILE (CH<>'"') AND (I<80) AND (CH<>CHR(13)) DO
3585:     BEGIN
3586:       SCH[1]:=CH;
3587:       STRVAL:=CONCAT(STRVAL,SCH);
3588:       IF SOURCE=PARMSOURCE THEN BACKSCAN:=TRUE; {always true if ever!}
3589:       GETCHAR;
3590:       I:=I+1;
3591:     END;
3592:     NOTSTRING:=TRUE;
3593:     IF BACKSCAN THEN
3594:       BEGIN
3595:         I:=SCAN(-I,<>' ',STRVAL[I]);
3596:         STRVAL[0]:=CHR(LENGTH(STRVAL) + I);
3597:       END;
3598:     IF CH=CHR(13) THEN
3599:     BEGIN
3600:       LEXTOKEN:=ENDLINE;
3601:       ERROR(41{string constant must be on one line});
3602:     END;
3603:     IF I>80 THEN
3604:       ERROR(42{string constant exceeds 80 chars});
3605:   END;
3606:
3607:   BEGIN {Lex}
3608:     IF DEBUG THEN WRITELN('Lex');
3609:     STARTLINE:=(LEXTOKEN=ENDLINE);
3610:     IF STARTLINE THEN
3611:       BEGIN
3612:         TEXTLINE:=BLANKLINE;
3613:         TEXTINDEX:=-1;
3614:       END;
3615:     GETCHAR;
3616:     WHILE CH=' ' DO
3617:       BEGIN
3618:         GETCHAR;
3619:         STARTLINE:=FALSE;
3620:       END;
3621:     IF CH=CHR(13) THEN LEXTOKEN:=ENDLINE ELSE
3622:     BEGIN
3623:       CASE CH OF
3624:         '0','1','2','3','4','5','6','7','8','9':PCONST;
3625:         'A','B','C','D','E','F','G','H','I','J','K','L','M',
3626:         'N','O','P','Q','R','S','T','U','V','W','X','Y','Z':PIDENT;
3627:         '.':PKWORD;
3628:         '#':LEXTOKEN:=NUMBERSIGN;
3629:         '(':LEXTOKEN:=OPENPAREN;
3630:         '[':LEXTOKEN:=OPENBRACKET;
```

```
3631:         '{':LEXTOKEN:=OPENBRACE;     (* This is 7 on the numeric pad *)
3632:         ',':LEXTOKEN:=COMMA;
3633:         '~':LEXTOKEN:=ONESCOMPLEMENT;  (* This is 4 on the numeric pad *)
3634:         '?':LEXTOKEN:=QUERY;
3635:         ']':LEXTOKEN:=CLOSEBRACKET;
3636:         ')':LEXTOKEN:=CLOSEPAREN;
3637:         '}':LEXTOKEN:=CLSBRACE;
3638:         ';':LEXTOKEN:=ENDLINE;
3639:         '@':LEXTOKEN:=ATSIGN;
3640:         '$':IF LCCHAR='$' THEN
3641:               BEGIN
3642:                 GETCHAR;
3643:                 IF (CH<'0') OR (CH>'9') THEN
3644:                   BEGIN
3645:                     LEXTOKEN:=LOCCTR;
3646:                     ADVANCE:=FALSE;
3647:                   END
3648:                 ELSE PLLABEL;
3649:               END
3650:             ELSE
3651:               BEGIN
3652:                 GETCHAR;
3653:                 PLLABEL;
3654:               END;
3655:         '"':PSTRING;          {Process a string}
3656:         '/':LEXTOKEN:=DIVIDE;
3657:         '!':LEXTOKEN:=TNOT;
3658:         '+':BEGIN
3659:               GETCHAR;
3660:               IF CH=CHR(ORD(AFTERPLUS)) THEN LEXTOKEN:=AUTOINCR
3661:                 ELSE LEXTOKEN:=PLUS; {Char after plus isn't eaten}
3662:               ADVANCE:=FALSE;
3663:             END;
3664:         '-':BEGIN
3665:               GETCHAR;
3666:               IF CH=CHR(ORD(AFTERMINUS)) THEN LEXTOKEN:=AUTODECR
3667:                 ELSE LEXTOKEN:=MINUS; {Char after minus isn't eaten}
3668:               ADVANCE:=FALSE;
3669:             END;
3670:         ':':LEXTOKEN:=COLON;
3671:         '|':LEXTOKEN:=BITWISEOR;
3672:         '^':LEXTOKEN:=EXCLUSIVEOR;
3673:         '&':LEXTOKEN:=AMPERSAND;
3674:         '*':LEXTOKEN:=ASTERISK;
3675:         '%':LEXTOKEN:=MODULO;
3676:         '<':BEGIN
3677:               GETCHAR;
3678:               IF CH='>' THEN
3679:                 LEXTOKEN:=NOTEQUAL
3680:               ELSE
3681:                 BEGIN
3682:                   LEXTOKEN:=OPNBROKEN;
3683:                   ADVANCE:=FALSE;
3684:                 END;
3685:             END;
3686:         '>':LEXTOKEN:=CLSBROKEN;
3687:         '=':LEXTOKEN:=EQUAL;
3688:       END;(*OF CASE STATMENT*)
3689:     END;
3690:     IF DEBUG THEN WRITELN('LEXTOKEN IS:',ORD(LEXTOKEN));
3691:   END;   (*of procedure LEX*)
3692:
3693:   BEGIN {Main Assembler}
3694:     INITIALIZE;
3695:     REPEAT
3696:       ASSEMBLE;
```

```
3697:       IF (PROCNUM>0) AND LISTING THEN SYMTBLDUMP;
3698:       PROCEND;
3699:     UNTIL LEXTOKEN=TEND;
3700:   END;
3701:
3702:
3703:   BEGIN  {dummy outer block}  END.
3704:

THAT'S ALL FOLKS!      LINES: 3704   CHARACTERS: 113143
```