A SIMULATOR FOR THE INTEL 8086 MICROPROCESSOR

by

William A. Chapman

A Thesis Submitted

in

Partial Fulfillment

of the

Requirements for the Degree of

Master of Science

in

Electrical Engineering

Volume 2, pt. 2

Command Procedures

and

Source Code

ROCHESTER INSTITUTE OF TECHNOLOGY

```
-LINE-IDC-PL-SL-

00001   C  0  0 { Title: Debugger (Revision 3)
00002   C  0  0
00003   C  0  0   Purpose: Interpret debugger commands for emulator
00004   C  0  0
00005   C  0  0   Author: William A. Chapman      Date: October 2, 1986
00006   C  0  0
00007   C  0  0   Inputs: KeywordLUTFile
00008   C  0  0           Operator inputs
00009   C  0  0           File name containing load image
00010   C  0  0
00011   C  0  0   Outputs: Commands are executed if valid
00012   C  0  0           Commands are logged to a journal file if requested
00013   C  0  0           Intel instructions are execute
00014   C  0  0
00015   C  0  0   Procedures or Functions Invoked:
00016   C  0  0           SyntaxInterpreter
00017   C  0  0           FetchInputLine
00018   C  0  0           FetchRegPtr
00019   C  0  0           StoreRegPtr
00020   C  0  0           FetchMemory
00021   C  0  0           StoreMemory
00022   C  0  0           MaskShiftRight
00023   C  0  0           IntMaskShiftRight
00024   C  0  0           SignExtend7
00025   C  0  0           SignExtend15
00026   C  0  0           ParseFilename
00027   C  0  0           UnpackHex
00028   C  0  0           GetData
00029   C  0  0           GetAddress
00030   C  0  0           DisplayBreakpoint
00031   C  0  0           InstallBreakpoints
00032   C  0  0           RemoveBreakpoints
00033   C  0  0           Loader
00034   C  0  0           SearchForBreakpoint
00035   C  0  0           ExecuteInstruction
00036   C  0  0           LIB$STOP
00037   C  0  0           $CANCEL
00038   C  0  0           $ASSIGN
00039   C  0  0           $QIOW
00040   C  0  0           AstD
00041   C  0  0           FileExists.FORTRAN
00042   C  0  0           (GetAssignData)
00043   C  0  0           (UnexpectedBreakpoint)
00044   C  0  0           (ProgramHalt)
00045   C  0  0           (GoodBreakpoint)
```

```
00046   C  0  0           (ReportBadStopCode)
00047   C  0  0           (GetPortAddress)
00048   C  0  0           (SearchForPortAddress)
00049   C  0  0           (FindNextPort)
00050   C  0  0           (Get8BitsOfData)
00051   C  0  0           (Put8BitsOfData)
00052   C  0  0
00053   C  0  0   Subprocedures Included:
00054   C  0  0           SearchForPortAddress.subpas
00055   C  0  0           MemoryClass.subpas
```

```
-LINE-IDC-PL-SL-

00056    C  0  0          StackClass.subpas
00057    C  0  0          RegisterClass.subpas
00058    C  0  0          FlagClass.subpas
00059    C  0  0          InputOutputClass.subpas
00060    C  0  0          BreakpointClass.subpas
00061    C  0  0          ExecuteClass.subpas
00062    C  0  0
00063    C  0  0 ]
```

```
-LINE-IDC-PL-SL-

00065       0  0 [inherit ('Sys$Library: Starlet')] module Debugger(input, output, KeywordLookUpTable );
00066       0  0
00067       0  0 const
00068       0  0     %include  [-]Const.defn/list'
00069 I     0  0     FirstOpcodeValue = 0;
00070 I     0  0     LastOpcodeValue = 255;
00071 I     0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00072 I     0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00073 I     0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00074 I     0  0     Bit8Multiplier = %X'100';
00075 I     0  0     Bit8Divisor    = %X'100';
00076 I     0  0     WordMultiplier = %X'100';
00077 I     0  0
00078 I     0  0     EightBits = 0;                       {width for 8 bits}
00079 I     0  0     SixteenBits = 1;                     {width for 16 bits}
00080 I     0  0
00081 I     0  0     SimpleMode = %B'0';
00082 I     0  0     SimpleRM = %B'110';
00083 I     0  0
00084 I     0  0     LowMemoryLimit = 0;                  {low limit on memory array}
00085 I     0  0     HighMemoryLimit = 2048;              {high limit on memory array}
00086 I     0  0
00087 I     0  0     FilenameLength = 40;
00088       0  0
00089       0  0     %include  [-]IOConst.defn/list'
00090 I     0  0     KeyboardInPortNumber  = %X'FF';
00091 I     0  0     CRTOutPortNumber       = %X'FF';
00092 I     0  0
00093 I     0  0     FirstIOPortIndex   = 1;
00094 I     0  0     MaxNumberofIOPorts = 8;
00095 I     0  0
00096 I     0  0     FirstIODataPoint   = 1;
00097 I     0  0     MaxNumberofIOWords = 64;
00098 I     0  0     MaxNumberofIOBytes = 256;     {4 times the number of words}
00099 I     0  0
00100 I     0  0     InputIndicator  = 'I';
00101 I     0  0     OutputIndicator = 'O';
00102 I     0  0     IOIndicator     = 'B';
00103 I     0  0
00104 I     0  0     LowerPortIndicator = 'L';
00105 I     0  0     UpperPortIndicator = 'U';
00106 I     0  0
00107 I     0  0     InvalidAddress = - 1;
00108 I     0  0     InvalidLocation = - 1;
```

```
00109 I     0   0     InvalidIndex = - 1;
00110       0   0
00111       0   0     %include  [-]Flags.defn/list'
00112 I     0   0     SetHigh = true;
00113 I     0   0     Clear = false;
00114 I     0   0
00115 I     0   0     CarryFlag     = %B'000000000001';
00116 I     0   0     ParityFlag    = %B'000000000100';
00117 I     0   0     AuxCarryFlag  = %B'000000010000';
00118 I     0   0     ZeroFlag      = %B'000001000000';
00119 I     0   0     SignFlag      = %B'000010000000';
```

```
-LINE-IDC-PL-SL-

00120 I    0  0     TrapFlag      = %B'000100000000';
00121 I    0  0     InterruptFlag = %B'001000000000';
00122 I    0  0     DirectionFlag = %B'010000000000';
00123 I    0  0     OverflowFlag  = %B'100000000000';
00124      0  0
00125      0  0     %include '[-]DebugConst.defn/list'
00126 I    0  0     FirstKeywordEntry = 0;
00127 I    0  0     LastKeywordEntry  = 100;
00128 I    0  0     UndefinedName = 'Qqqqqqqq';
00129 I    0  0     UndefinedClass = 'z';
00130 I    0  0     UndefinedWidth = -1;
00131 I    0  0     UndefinedID = -2;
00132 I    0  0     PageSize = 55;
00133 I    0  0     FF = 12;
00134 I    0  0     MinID = -1;
00135 I    0  0     MaxID = 25;
00136 I    0  0     MinWidth = 0;
00137 I    0  0     MaxWidth = 3;
00138 I    0  0     SegmentOffset = 16;
00139 I    0  0
00140 I    0  0     BlankClass       =    ;
00141 I    0  0     BreakpointClass  = 'B';
00142 I    0  0     ErrorClass       = 'E';
00143 I    0  0     FlagClass        = 'F';
00144 I    0  0     InputOutputClass = 'I';
00145 I    0  0     MemoryClass      = 'M';
00146 I    0  0     NumericClass     = 'N';
00147 I    0  0     OperatorClass    = 'O';
00148 I    0  0     QualifierClass   = 'Q';
00149 I    0  0     RegisterClass    = 'R';
00150 I    0  0     StackClass       = 'S';
00151 I    0  0     TrueFalseClass   = 'T';
00152 I    0  0     UtilityClass     = 'U';
00153 I    0  0     ExecuteClass     = 'X';
00154 I    0  0
00155 I    0  0     Blank  =    ;
00156 I    0  0     Elipse =  .';
00157 I    0  0
00158 I    0  0     StartOfData = 1;
00159 I    0  0     DataArraySize = 50;
00160 I    0  0
00161 I    0  0     EndofLine = 0;
00162 I    0  0     StartofLine = 1;
00163 I    0  0
```

```
00164 I     0   0      ShortStringLength = 8;
00165 I     0   0      InputLineLength = 120;
00166 I     0   0
00167 I     0   0      BooleanID  = 1;
00168 I     0   0      ByteID     = 2;
00169 I     0   0      IntegerID  = 3;
00170 I     0   0      PointerID  = 4;
00171 I     0   0      SIntegerID = 5;
00172 I     0   0      WordID     = 6;
00173 I     0   0      LengthID   = 3;
00174 I     0   0      ToID       = 5;
```

```
-LINE-IDC-PL-SL-

00175 I    0  0
00176 I    0  0     AssignmentID = 1;
00177 I    0  0     ColonID      = 2;
00178 I    0  0     PlusID       = 3;
00179 I    0  0     MinusID      = 4;
00180 I    0  0     CommaID      = 5;
00181 I    0  0
00182 I    0  0     NullEntries = 0;
00183 I    0  0
00184 I    0  0     AddressState   = 'A';
00185 I    0  0     BooleanState   = 'B';
00186 I    0  0     ErrorState     = 'E';
00187 I    0  0     HexState       = 'H';
00188 I    0  0     IntegerState   = 'I';
00189 I    0  0     SegmentState   = 'S';
00190 I    0  0     SeparatorState = ',';
00191 I    0  0     UnaryState     = '-';
00192 I    0  0     ColonState     = ': ';
00193 I    0  0
00194 I    0  0     NegativeOperator = -1;
00195 I    0  0     PositiveOperator = +1;
00196 I    0  0
00197 I    0  0     NullSegmentValue = 0;
00198 I    0  0     NullAddressValue = 0;
00199 I    0  0
00200 I    0  0     FirstBreakpoint  =  0;
00201 I    0  0     LastBreakpoint   = 15;
00202 I    0  0     AllBPIDs         = 16;
00203 I    0  0     Activate         = true;
00204 I    0  0     DeActivate       = false;
00205 I    0  0     BreakpointOpcode = %X'CC';
```

```
-LINE-IDC-PL-SL-

00207       0  0     %include  [-]RegID.defn/list'
00208 I     0  0     ALId = %B'000';
00209 I     0  0     ALWidth = 0;
00210 I     0  0
00211 I     0  0     CLId = %B'001';
00212 I     0  0     CLWidth = 0;
00213 I     0  0
00214 I     0  0     DLId = %B'010';
00215 I     0  0     DLWidth = 0;
00216 I     0  0
00217 I     0  0     BLId = %B'011';
00218 I     0  0     BLWidth = 0;
00219 I     0  0
00220 I     0  0     AHId = %B'100';
00221 I     0  0     AHWidth = 0;
00222 I     0  0
00223 I     0  0     CHId = %B'101';
00224 I     0  0     CHWidth = 0;
00225 I     0  0
00226 I     0  0     DHId = %B'110';
00227 I     0  0     DHWidth = 0;
00228 I     0  0
00229 I     0  0     BHId = %B'111';
00230 I     0  0     BHWidth = 0;
00231 I     0  0
00232 I     0  0     AXId = %B'000';
00233 I     0  0     AXWidth = 1;
00234 I     0  0     ALorAXId = %B'000';
00235 I     0  0
00236 I     0  0     CXId = %B'001 ;
00237 I     0  0     CXWidth = 1;
00238 I     0  0
00239 I     0  0     DXId = %B'010';
00240 I     0  0     DXWidth = 1;
00241 I     0  0
00242 I     0  0     BXId = %B'011';
00243 I     0  0     BXWidth = 1;
00244 I     0  0
00245 I     0  0     SPId = %B'100';
00246 I     0  0     SPWidth = 1;
00247 I     0  0
00248 I     0  0     BPId = %B'101';
00249 I     0  0     BPWidth = 1;
00250 I     0  0
```

```
00251 I     0   0      SIId = %B'110';
00252 I     0   0      SIWidth = 1;
00253 I     0   0
00254 I     0   0      DIId = %B'111';
00255 I     0   0      DIWidth = 1;
00256 I     0   0
00257 I     0   0      SRWidth = 2;
00258 I     0   0
00259 I     0   0      ESId = %B'00';
00260 I     0   0      ESWidth = 2;
00261 I     0   0
```

```
-LINE-IDC-PL-SL-

00317       0  0
00318       0  0     GoContinue = true;
00319       0  0     Stop = false;
00320       0  0
00321       0  0     NoPorts = 0;
00322       0  0     OnePort = 1;
00323       0  0
00324       0  0     InvalidPortID = - 1;
00325       0  0     NullDatum = 0;
00326       0  0
00327       0  0     CTRLD = %B'10000';
00328       0  0
00329       0  0     %include  [-]StopCode.defn/list'
00330 I     0  0     Breakpoint              = 'B';
00331 I     0  0     Call          = 'C';
00332 I     0  0     ControlD              = 'D';
00333 I     0  0     BadOpcode               = 'E';
00334 I     0  0     BadCEAModeValue         = 'F';
00335 I     0  0     BadRMDModeValue         = 'G';
00336 I     0  0     Halt          = 'H';
00337 I     0  0     BadRegisterID = 'I';
00338 I     0  0     BadOpcodeKey  = 'K';
00339 I     0  0     BadMemoryAddress        = 'M';
00340 I     0  0     Normal                  = 'N';
00341 I     0  0     BadPortAddress          = 'P';
00342 I     0  0     BadOpcodeClass          = 'Q';
00343 I     0  0     Return                  = 'R';
00344 I     0  0     BadCheckSum             = 'S';
00345 I     0  0     BadOperandType          = 'T';
00346 I     0  0     BadMemoryWidth          = 'W';
00347 I     0  0     BadOpcodeExtension      = 'X';
00348 I     0  0     NoMemoryAccess        = 'Z';
```

```
-LINE-IDC-PL-SL-

00350       0  0 type
00351       0  0     %include '[-]Type.defn/list'
00352 I     0  0     ZeroOne = -1..1;                          {defines integer with range
00353 I C   0  0                                               of zero to one with an invalid
00354 I C   0  0                                               state of -1}
00355 I     0  0
00356 I     0  0     OpcodeLUTEntry = record                   {defines entries in opcode LUT}
00357 I     0  0   OpcodeClass: char;                         {class D, A, L, S, C, P}
00358 I     0  0   OpcodeKey: integer;                        {key to interpret opcode from
00359 I C   0  0                                               0 to 7}
00360 I     0  0   DirectionBitPresent: boolean;
00361 I     0  0   WidthBitPresent: boolean
00362 I     0  0     end;
00363 I     0  0
00364 I     0  0     OpcodeType = record
00365 I     0  0   Direction: ZeroOne;                        {to or from CPU}
00366 I     0  0   Full: unsigned;                            {full opcode}
00367 I     0  0   Width: ZeroOne                            {8 or 16 bits}
00368 I     0  0     end;
00369 I     0  0
00370 I     0  0     RegisterType = record
00371 I     0  0   Id: integer;                              {identifier}
00372 I     0  0   Width: ZeroOne                            {8 or 16 bits}
00373 I     0  0     end;
00374 I     0  0
00375 I     0  0     EffectiveAddressType = record
00376 I     0  0   Mode: char;                               {register or memory}
00377 I     0  0   Width: ZeroOne;                           {8 or 16 bits}
00378 I     0  0   Address: unsigned;                       {memory address or
00379 I C   0  0                                               register designation}
00380 I     0  0   Segment: integer                          {segment register to use}
00381 I     0  0     end;
00382 I     0  0
00383 I     0  0     NameType = packed array[1..10] of char;
00384 I     0  0
00385 I     0  0     Characters = set of    ..'~';
00386 I     0  0     LettersAndNumbers = set of '0'..'z';
00387 I     0  0     Letters = set of 'A'..'z';
00388 I     0  0     Numbers = set of '0'..'9';
00389 I     0  0
00390 I     0  0     FileName = packed array [1..FilenameLength] of char;
00391       0  0
00392       0  0     %include '[-]IOType.defn/list'
00393 I     0  0     PortEntry = record
```

```
00394 I     0   0    Address: unsigned;
00395 I     0   0    AuxIndex: integer;
00396 I     0   0    DataIndex: integer;
00397 I     0   0    InputIndex: integer;
00398 I     0   0    PortWidth: integer;
00399 I     0   0    LowerUpperIndicator: char;
00400 I     0   0    InOutIndicator: char
00401 I     0   0      end;
00402 I     0   0
00403 I     0   0      PortMap = array[ FirstIOPortIndex..MaxNumberOfIOPorts] of PortEntry;
00404       0   0
```

```
-LINE-IDC-PL-SL-

00405       0  0     %include  [-]FlagType.defn/list'
00406 I     0  0     FlagType = record                              {define processor flags}
00407 I     0  0   Carry: boolean;
00408 I     0  0   Parity: boolean;
00409 I     0  0   AuxCarry: boolean;
00410 I     0  0   Zero: boolean;
00411 I     0  0   Sign: boolean;
00412 I     0  0   Trap: boolean;
00413 I     0  0   Interrupt: boolean;
00414 I     0  0   Direction: boolean;
00415 I     0  0   Overflow: boolean
00416 I     0  0     end;
00417       0  0
00418       0  0     %include  [-]DebugType.defn/list'
00419 I     0  0     NameString = packed array[1..ShortStringLength] of char;
00420 I     0  0
00421 I     0  0     KeywordCharacteristics = record
00422 I     0  0   Name: NameString;
00423 I     0  0   Class: char;
00424 I     0  0   Width: ZeroOne;
00425 I     0  0   ID: integer;
00426 I     0  0   Value: unsigned
00427 I     0  0     end;
00428 I     0  0
00429 I     0  0     InputPointerRange = integer;
00430 I     0  0     InputLine = packed array [1..InputLineLength] of char;
00431 I     0  0
00432 I     0  0     ShortString = packed array [1..ShortStringLength] of char;
00433 I     0  0
00434 I     0  0     DataArrayType = array [StartOfData..DataArraySize] of unsigned;
00435 I     0  0
00436 I     0  0     BreakRecord = record
00437 I     0  0   Encountered: boolean;
00438 I     0  0   Activated: boolean;
00439 I     0  0   Segment: unsigned;
00440 I     0  0   Address: unsigned;
00441 I     0  0   PhysicalAddress: unsigned;
00442 I     0  0   Code: unsigned
00443 I     0  0     end;
00444 I     0  0
00445 I     0  0     BreakPointTableType = array [FirstBreakpoint..LastBreakpoint] of BreakRecord;
00446       0  0
00447       0  0     QuadWord = [word] 0..65535;
```

```
-LINE-IDC-PL-SL-

00449       0  0 var
00450       0  0     KeywordLUT: [global] array[FirstKeywordEntry..LastKeywordEntry]
00451       0  0   of KeywordCharacteristics;
00452       0  0
00453       0  0     Continue: [global] LettersAndNumbers;
00454       0  0
00455       0  0     ChannelID: [global, volatile] array [1..2] of QuadWord;
00456       0  0
00457       0  0
00458       0  0
00459       0  0
00460       0  0     %include '[-]RegExternal.defn/list'
00461 I     0  0     AX: [external] unsigned;      {value of the AX register]
00462 I     0  0     CX: [external] unsigned;      {value of the CX register]
00463 I     0  0     DX: [external] unsigned;      {value of the DX register]
00464 I     0  0     BX: [external] unsigned;      {value of the BX register]
00465 I     0  0     SP: [external] unsigned;      {value of the SP register]
00466 I     0  0     BP: [external] unsigned;      {value of the BP register]
00467 I     0  0     SI: [external] unsigned;      {value of the SI register]
00468 I     0  0     DI: [external] unsigned;      {value of the DI register]
00469 I     0  0
00470 I     0  0     ES: [external] unsigned;      {value of the ES register]
00471 I     0  0     CS: [external] unsigned;      {value of the CS register]
00472 I     0  0     SS: [external] unsigned;      {value of the SS register]
00473 I     0  0     DS: [external] unsigned;      {value of the DS register]
00474 I     0  0
00475       0  0
00476       0  0     %include '[-]IOVarExternal.defn/list'
00477 I     0  0     IOPortMap: [external] PortMap;
00478 I     0  0
00479 I     0  0     IOData: [external] array[FirstIOPortIndex..MaxNumberofIOPorts,
00480 I     0  0   FirstIODataPoint..MaxNumberofIOWords] of unsigned;
00481       0  0
00482       0  0     IP: [external] unsigned;
00483       0  0     Flags: [external] FlagType;
00484       0  0
00485       0  0     Punctuation: [external] Characters;
00486       0  0     Numerals: [external] Numbers;
00487       0  0     HexNumerals: [external] LettersandNumbers;
00488       0  0
00489       0  0     JournalingFlag: [external] boolean;
00490       0  0     Journal: [external] text;
```

```
-LINE-IDC-PL-SL-

00492      0  0     KeywordLookUpTable: file of KeywordCharacteristics;
00493      0  0     Keyword: KeywordCharacteristics;
00494      0  0     KeywordFileName: Filename;
00495      0  0
00496      0  0     SourceLine: InputLine;
00497      0  0     SourcePointer:  InputPointerRange;
00498      0  0     SourceLength: InputPointerRange;
00499      0  0
00500      0  0     JournalFilename: Filename;
00501      0  0     LoadFilename: Filename;
00502      0  0     NullName: boolean;
00503      0  0
00504      0  0     Operator: KeywordCharacteristics;
00505      0  0
00506      0  0     Loop: integer;
00507      0  0
00508      0  0     ExitFlag: boolean;
00509      0  0
00510      0  0     EOLNFlag: boolean;
00511      0  0
00512      0  0     Identifier: integer;
00513      0  0     RegisterWidth: ZeroOne;
00514      0  0     RegisterID: integer;
00515      0  0     NewRegister: KeywordCharacteristics;
00516      0  0     DisplayRegister: KeywordCharacteristics;
00517      0  0     DisplayValue: unsigned;
00518      0  0
00519      0  0     AssembledFlags: unsigned;
00520      0  0     FlagID: integer;
00521      0  0     NewFlag: KeywordCharacteristics;
00522      0  0
00523      0  0     SSValue: unsigned;
00524      0  0     SPValue: unsigned;
00525      0  0     StackDisplayLength: integer;
00526      0  0     StackDisplayIndex: integer;
00527      0  0     StackDisplay: KeywordCharacteristics;
00528      0  0     LowStackData: unsigned;
00529      0  0     HighStackData: unsigned;
00530      0  0     StackData: unsigned;
00531      0  0
00532      0  0     DataArray: DataArrayType;
00533      0  0     DataArrayEntries: integer;
00534      0  0     DataArrayPointer: integer;
00535      0  0
```

```
00536      0  0     Format: KeywordCharacteristics;
00537      0  0     Modifier: KeywordCharacteristics;
00538      0  0     LengthInput: KeywordCharacteristics;
00539      0  0
00540      0  0     StartSegment: unsigned;
00541      0  0     StartAddress: unsigned;
00542      0  0     EndSegment:   unsigned;
00543      0  0     EndAddress:   unsigned;
00544      0  0     NullAddress:  boolean;
00545      0  0
00546      0  0     OperatingLength: unsigned;
```

```
-LINE-IDC-PL-SL-

00547       0  0     Command: char;
00548       0  0
00549       0  0     DisplaySegment:  unsigned;
00550       0  0     DisplayAddress:  unsigned;
00551       0  0     HighDisplayData: unsigned;
00552       0  0     DisplayData:     unsigned;
00553       0  0     DisplayLoop:     integer;
00554       0  0     DisplayIntegerData:     integer;
00555       0  0
00556       0  0     AssignSegment:  unsigned;
00557       0  0     AssignAddress:  unsigned;
00558       0  0     AssignData:     unsigned;
00559       0  0     AssignDataLow:  unsigned;
00560       0  0     AssignDataHigh: unsigned;
00561       0  0     AssignLoop:     integer;
00562       0  0
00563       0  0     BreakPointTable: BreakPointTableType;
00564       0  0     BreakSegment: unsigned;
00565       0  0     BreakAddress: unsigned;
00566       0  0     BreakPointLoop: integer;
00567       0  0     BreakPointEntry: integer;
00568       0  0     BreakPointFound: boolean;
00569       0  0
00570       0  0     Execute: boolean;
00571       0  0     StopCode: char;
00572       0  0     NewCSValue: unsigned;
00573       0  0     NewIPValue: unsigned;
00574       0  0
00575       0  0     PStepCount: integer;
00576       0  0     Response: char;
00577       0  0     Yes: LettersAndNumbers;
00578       0  0
00579       0  0     IODone: boolean;
00580       0  0     PortCount: integer;
00581       0  0     PortAddress: unsigned;
00582       0  0     FinalPortAddress: unsigned;
00583       0  0     PortIndex: integer;
00584       0  0     PortAddressPresent: boolean;
00585       0  0
00586       0  0     TempDataIndex: integer;
00587       0  0
00588       0  0     SavedCS: unsigned;
00589       0  0     SavedIP: unsigned;
00590       0  0     ExecuteStopCode: char;
```

```
00591      0  0
00592      0  0     SysStat: integer;
00593      0  0     IOFunction: integer;
00594      0  0     ControlMask: array [1..2] of integer;
00595      0  0     CtrlDStopCode: [volatile] char;
00596      0  0
00597      0  0     Exists: boolean;
```

```
-LINE-IDC-PL-SL-

00599      1  0 [global] procedure Debugger( DFormat: char);
00600      1  0
00601      2  0 procedure SyntaxInterpreter( var SIKeyword: KeywordCharacteristics;
00602      2  0           var SIOperatorInput: InputLine;
00603      2  0           var SIInputPointer: InputPointerRange;
00604      2  0           var SIInputLength: InputPointerRange;
00605      1  0           var SIEndofLine: boolean); external;
00606      1  0
00607      2  0 procedure FetchInputLine( var FILInput: InputLine;
00608      2  0           var FILInputPointer: InputPointerRange;
00609      1  0           var FILInputLength: InputPointerRange); external;
00610      1  0
00611      2  0 procedure FetchRegPtr(FRPWidth: integer;
00612      2  0           FRPDesignator: integer;
00613      2  0           var FRPValue: unsigned;
00614      1  0           var FRPStopCode: char); external;
00615      1  0
00616      2  0 procedure StoreRegPtr(SRPWidth: integer;
00617      2  0           SRPDesignator: integer;
00618      2  0           SRPValue: unsigned;
00619      1  0           var SRPStopCode: char); external;
00620      1  0
00621      2  0 procedure FetchMemory(FMAddress: unsigned;
00622      2  0           FMSegment: unsigned;
00623      2  0           var FMValue: unsigned;
00624      1  0           var FMStopCode: char); external;
00625      1  0
00626      2  0 procedure StoreMemory(SMAddress: unsigned;
00627      2  0           SMSegment: unsigned;
00628      2  0           SMValue: unsigned;
00629      1  0           var SMStopCode: char); external;
00630      1  0
00631      2  0 function MaskShiftRight( MSRValue: unsigned;
00632      2  0           MSRMask: unsigned;
00633      1  0           MSRDivisor: integer): unsigned; external;
00634      1  0
00635      2  0 function IntMaskShiftRight( IMSRValue: unsigned;
00636      2  0           IMSRMask: unsigned;
00637      1  0           IMSRDivisor: integer): integer; external;
00638      1  0
00639      1  0 function SignExtend7( SE7Value: unsigned): integer; external;
00640      1  0
00641      1  0 function SignExtend15( SE15Value: unsigned): integer; external;
```

```
-LINE-IDC-PL-SL-

00643      2  0 procedure ParseFilename( var PFNullFlag: boolean;
00644      2  0           var PFNameOfFile: Filename;
00645      2  0           PFOperatorInput: InputLine;
00646      2  0           var PFPointer: InputPointerRange;
00647      2  0           PFInputLength: InputPointerRange;
00648      1  0           var PFEndOfLineFlag: boolean); external;
00649      1  0
00650      2  0 function UnpackHex( UHPackedHex: unsigned;
00651      1  0           UHSignValue: integer): integer; external;
00652      1  0
00653      2  0 procedure GetData( var GDArray: DataArrayType;
00654      2  0           var GDArrayEntries: integer;
00655      2  0           GDSourceLine: InputLine;
00656      2  0           var GDSourcePointer: InputPointerRange;
00657      2  0           GDSourceLength: InputPointerRange;
00658      2  0           var GDEOLNFlag: boolean;
00659      1  0           GDFormatIdent: integer); external;
00660      1  0
00661      2  0 procedure GetAddress( var GASegmentValue: unsigned;
00662      2  0           var GAAddressValue: unsigned;
00663      2  0           GASourceLine: InputLine;
00664      2  0           var GASourcePointer: InputPointerRange;
00665      2  0           GASourceLength: InputPointerRange;
00666      2  0           var GAEOLNFlag: boolean;
00667      1  0           var GANullAddress: boolean); external;
00668      1  0
00669      2  0 procedure DisplayBreakpoint( DBreakpointID: integer;
00670      1  0           DBreakpointEntry: BreakRecord); external;
00671      1  0
00672      2  0 procedure InstallBreakpoints( var IBBreakPointTable: BreakPointTableType;
00673      1  0           var IBStopCode: char); external;
00674      1  0
00675      2  0 procedure RemoveBreakpoints( var RBBreakPointTable: BreakPointTableType;
00676      1  0           var RBStopCode: char); external;
00677      1  0
00678      2  0 procedure SearchForBreakpoint( var SFBBreakPointTable: BreakPointTableType;
00679      2  0           var SFBBreakPointEntry: integer;
00680      2  0           var SFBBreakPointFound: boolean;
00681      2  0           SFBCS: unsigned;
00682      1  0           SFBIP: unsigned); external;
00683      1  0
00684      2  0 procedure Loader( LNameOffile: Filename;
00685      2  0           LFormat: char;
00686      2  0           var LStopCode: char;
```

```
00687      2  0           var LCS: unsigned;
00688      1  0           var LIP: unsigned); external;
00689      1  0
00690      1  0 procedure ExecuteInstruction( var EIStopCode: char); external;
00691      1  0
00692      1  0 function FileExists( FEFileName: Filename): boolean; Fortran;
```

```
-LINE-IDC-PL-SL-

00694       1  0 [asynchronous] procedure Lib$Stop( %immed L$SStopValue: integer); external;
00695       1  0
00696       2  0 [asynchronous, unbound] procedure ASTD;
00697       2  0
00698       2  0 var
00699       2  0     AsyncStatus: integer;
00700       2  0
00701       2  1 begin
00702       2  1     AsyncStatus := $Cancel( ChannelID[ 2]);
00703       2  1     if (not odd( AsyncStatus)) then Lib$Stop( AsyncStatus);
00704       2  1
00705       2  1     CtrlDStopCode := ControlD;
00706       1  0 end;
```

```
-LINE-IDC-PL-SL-

00708    C  1  0 { Title: GetAssignData
00709    C  1  0
00710    C  1  0   Purpose: Get data from the data array buffer, recirculating the
00711    C  1  0            buffer pointer as needed
00712    C  1  0
00713    C  1  0   Author: William A. Chapman      Date: September 15, 1986
00714    C  1  0
00715    C  1  0   Inputs: DataArrayPointer
00716    C  1  0           DataArray
00717    C  1  0           DataArrayEntries
00718    C  1  0
00719    C  1  0   Outputs: DataArray value is returned
00720    C  1  0            DataArrayPointer is updated
00721    C  1  0
00722    C  1  0   Procedures Invoked: none
00723    C  1  0 ]
00724       2  0 procedure GetAssignData( var GADValue: unsigned;
00725       2  0           var GADPointer: integer;
00726       2  0           GADArray: DataArrayType;
00727       2  0           GADEntries: integer);
00728       2  0
00729       2  1 begin
00730       2  1     GADValue := GADArray[GADPointer];
00731       2  1     GADPointer := GADPointer + 1;
00732       2  1     if (GADPointer > GADEntries) then GADPointer := StartOfData;
00733       1  0 end;
00734       1  0
00735       1  0
00736    C  1  0 { Title: UnexpectedBreakpoint
00737    C  1  0
00738    C  1  0   Purpose: Process an unexpected breakpoint
00739    C  1  0
00740    C  1  0   Author: William A. Chapman      Date: October 16,1986
00741    C  1  0
00742    C  1  0   Inputs: Execute status
00743    C  1  0
00744    C  1  0   Outputs: Point of encounter is reported
00745    C  1  0            IP is backed up one
00746    C  1  0
00747    C  1  0   Procedures Invoked: none
00748    C  1  0 ]
00749       2  0 procedure UnexpectedBreakpoint( var UBExecuteStatus: boolean;
00750       2  0           UBCS: unsigned;
00751       2  0           UBIP: unsigned);
```

```
00752      2  0
00753      2  0
00754      2  1 begin
00755      2  1     UBExecuteStatus := Stop;
00756      2  1     write( 'Unexpected breakpoint encountered at:  );
00757      2  1     write( hex( UBCS, 4, 4), ':');
00758      2  1     write( hex( UBIP, 4, 4));
00759      2  1     writeln;
00760      1  0 end;
```

```
-LINE-IDC-PL-SL-

00762   C  1  0 { Title: ProgramHalt
00763   C  1  0
00764   C  1  0   Purpose: Report a "halt" code encountered
00765   C  1  0
00766   C  1  0   Author: William A. Chapman      Date: October 16,1986
00767   C  1  0
00768   C  1  0   Inputs: Execute status
00769   C  1  0
00770   C  1  0   Outputs: Halt is reported to operator
00771   C  1  0
00772   C  1  0   Procedures Invoked: none
00773   C  1  0 ]
00774      2  0 procedure ProgramHalt( var PHExecuteStatus: boolean;
00775      2  0           PHCS: unsigned;
00776      2  0           PHIP: unsigned);
00777      2  0
00778      2  1 begin
00779      2  1     PHExecuteStatus := Stop;
00780      2  1     write( 'Halt encountered at: ');
00781      2  1     write( hex( PHCS, 4, 4), ':');
00782      2  1     write( hex( PHIP, 4, 4));
00783      2  1     writeln;
00784      1  0 end;
00785      1  0
00786      1  0
00787   C  1  0 [ Title: GoodBreakpoint
00788   C  1  0
00789   C  1  0   Purpose: Report to the operator a valid breakpoint was encountered
00790   C  1  0
00791   C  1  0   Author: William A. Chapman      Date: October 17, 1986
00792   C  1  0
00793   C  1  0   Inputs: BreakpointTable
00794   C  1  0           BreakpointEntry
00795   C  1  0           ExecuteStatus
00796   C  1  0
00797   C  1  0   Outputs: Breakpoint is flagged as encountered
00798   C  1  0           Breakpoint is reported to operator
00799   C  1  0           ExecuteStatus is set to Stop
00800   C  1  0
00801   C  1  0   Procedures Invoked: none
00802   C  1  0 ]
00803      2  0 procedure GoodBreakpoint( var GBBreakpointTable: BreakPointTableType;
00804      2  0           GBBreakPointEntry: integer;
00805      2  0           var GBExecuteStatus: boolean;
```

```
00806    2  0           GBCS: unsigned;
00807    2  0           GBIP: unsigned);
00808    2  0
00809    2  1 begin
00810    2  1     GBExecuteStatus := Stop;
00811    2  1     GBBreakPointTable[ GBBreakPointEntry].Encountered := Activate;
00812    2  1     write( 'Breakpoint BP', hex( BreakPointEntry, 1, 1), ' encountered at: ');
00813    2  1     write( hex( GBCS, 4, 4), ':', hex( GBIP, 4, 4));
00814    2  1     writeln;
00815    1  0 end;
```

```
-LINE-IDC-PL-SL-

00817   C  1  0 { Title: ReportBadStopCode
00818   C  1  0
00819   C  1  0   Purpose: Report to user the bad stopcode encountered
00820   C  1  0
00821   C  1  0   Author: William A. Chapman      Date: October 17, 1986
00822   C  1  0
00823   C  1  0   Inputs: StopCode
00824   C  1  0           ExecuteStatus
00825   C  1  0           SavedSegmentValue
00826   C  1  0           SavedAddressValue
00827   C  1  0
00828   C  1  0   Outputs: StopCode is reported to the operator
00829   C  1  0              with the Segment and Address of the fault
00830   C  1  0           Execute Status is set to stop
00831   C  1  0
00832   C  1  0   Procedures Invoked: none
00833   C  1  0 }
00834      2  0 procedure ReportBadStopCode( RBSCStopCode: char;
00835      2  0           var RBSCExecuteStatus: boolean;
00836      2  0           RBSCCS: unsigned;
00837      2  0           RBSCIP: unsigned);
00838      2  0
00839      2  0
00840      2  1 begin
00841      2  1     RBSCExecuteStatus := Stop;
00842      2  1     writeln;  writeln;
00843      2  1
00844      2  2     case RBSCStopCode of
00845      2  2
00846      2  2     BadRMDModeValue: write( 'Invalid register / memory decoder mode  );
00847      2  2
00848      2  2     BadOpcode: write( 'Invalid opcode ');
00849      2  2
00850      2  2     BadCEAModeValue: write( 'Invalid effective address ');
00851      2  2
00852      2  2     BadRegisterID: write( 'Invalid register identifier  );
00853      2  2
00854      2  2     BadOpcodeKey: write( 'Invalid opcode key ');
00855      2  2
00856      2  2     BadMemoryAddress: write( 'Invalid memory address ');
00857      2  2
00858      2  2     BadPortAddress: write( 'Invalid IO port address  );
```

```
-LINE-IDC-PL-SL-

00860       2  2     BadOpcodeClass: write( 'Invalid opcode class  );
00861       2  2
00862       2  2     BadChecksum: write( 'Invalid checksum while loading memory  );
00863       2  2
00864       2  2     BadOperandType: write( 'Invalid operand type ');
00865       2  2
00866       2  2     BadMemoryWidth: write( 'Invalid memory access (width)  );
00867       2  2
00868       2  2     BadOpcodeExtension: write( 'Invalid opcode extension  );
00869       2  2
00870       2  2     NoMemoryAccess: write( 'Memory not accessed during load operation ');
00871       2  2
00872       2  2     ControlD: write( 'Aborted by a CtrlD ');
00873       2  2
00874       2  2     otherwise write( 'Invalid stop code ');
00875       2  2
00876       2  1     end; {case RBSCIndex}
00877       2  1
00878       2  1     write( 'encountered at:  );
00879       2  1     write( hex( RBSCCS, 4, 4),  :', hex( RBSCIP, 4, 4));
00880       2  1     writeln;
00881       1  0 end;
```

```
-LINE-IDC-PL-SL-

00883   C  1  0 { Title: GetPortAddress
00884   C  1  0
00885   C  1  0   Purpose: Obtain port address values from the input stream
00886   C  1  0
00887   C  1  0   Author: William A. Chapman      Date: October 28, 1986
00888   C  1  0
00889   C  1  0   Inputs: OperatorInput
00890   C  1  0           Current input pointer
00891   C  1  0           Length of input
00892   C  1  0
00893   C  1  0   Outputs: Port address value
00894   C  1  0
00895   C  1  0   Procedures Invoked: SyntaxInterpreter
00896   C  1  0           FetchRegPtr
00897   C  1  0 ]
00898      2  0 procedure GetPortAddress( var GPAAddressValue: unsigned;
00899      2  0           GPASourceLine: InputLine;
00900      2  0           var GPASourcePointer: InputPointerRange;
00901      2  0           GPASourceLength: InputPointerRange;
00902      2  0           var GPAEOLNFlag: boolean;
00903      2  0           var GPANullAddress: boolean);
00904      2  0
00905      2  0 var
00906      2  0     GPAAddressRecord: KeywordCharacteristics;
00907      2  0     GPALocalStopCode: char;
00908      2  0
00909      2  1 begin
00910      2  1     GPAAddressValue := InvalidPortID;
00911      2  1     GPANullAddress := false;
00912      2  1
00913      2  1     SyntaxInterpreter( GPAAddressRecord, GPASourceLine, GPASourcePointer,
00914      2  1   GPASourceLength, GPAEOLNFlag);
00915      2  1
00916      2  2     case GPAAddressRecord.Class of
00917      2  2
00918      2  2     BlankClass: GPANullAddress := true;
00919      2  2
00920      2  2     NumericClass: GPAAddressValue := GPAAddressRecord.Value;
00921      2  2
00922      2  2     RegisterClass: with GPAAddressRecord do
00923      2  3   if ( ID <> DXId) then begin
00924      2  3        writeln( 'Invalid register identifier for IO operation');
00925      2  3        GPANullAddress := true;
00926      2  3   end
```

```
00927      2  2   else FetchRegPtr( DXWidth, DXId, GPAAddressValue, GPALocalStopCode);
00928      2  2
00929      2  2     otherwise writeln( 'Invalid Address Component: "', GPAAddressRecord.Name, '"');
00930      2  2
00931      2  1     end; {case GPAAddressRecord.Class}
00932      1  0 end;
```

```
-LINE-IDC-PL-SL-

00934       1  0     %include 'SearchForPortAddress.subpas/list'
00935 I C   1  0 { Title: SearchForPortAddress
00936 I C   1  0
00937 I C   1  0   Purpose: Find a specified port address
00938 I C   1  0
00939 I C   1  0   Author: William A. Chapman      Date: May 11, 1986
00940 I C   1  0
00941 I C   1  0   Inputs: PortMap
00942 I C   1  0           Target Port Address
00943 I C   1  0
00944 I C   1  0   Outputs: PortFound - boolean flag
00945 I C   1  0           PortLocation - index of port
00946 I C   1  0
00947 I C   1  0   Procedures Called: none
00948 I C   1  0 ]
00949 I     2  0 procedure SearchForPortAddress( SFPATargetPortAddress: unsigned;
00950 I     2  0           var SFPASearchPortMap: PortMap;
00951 I     2  0           var SFPAPortFound: boolean;
00952 I     2  0           var SFPAPortLocation: integer);
00953 I     2  0
00954 I     2  0 var
00955 I     2  0     SFPAIndex: integer;
00956 I     2  0     SFPADone: boolean;
00957 I     2  0
00958 I     2  1 begin
00959 I     2  1     SFPAIndex := FirstIOPortIndex;
00960 I     2  1     SFPAPortFound := false;
00961 I     2  1     SFPADone := false;
00962 I     2  1
00963 I     2  2     while (not SFPADone) do begin
00964 I     2  3   if (SFPASearchPortMap[ SFPAIndex].Address = SFPATargetPortAddress) then begin
00965 I     2  3       SFPAPortFound := true;
00966 I     2  3       SFPAPortLocation := SFPAIndex;
00967 I     2  3       SFPADone := true;
00968 I     2  3   end
00969 I     2  2   else SFPAIndex := SFPAIndex + 1;
00970 I     2  3   if (SFPAIndex > MaxNumberofIOPorts) then begin
00971 I     2  3       SFPAPortFound := false;
00972 I     2  3       SFPAPortLocation := InvalidLocation;
00973 I     2  3       SFPADone := true;
00974 I     2  2   end;
00975 I     2  1     end;  {while (not SFPADone)}
00976 I     1  0 end; {SearchForPortAddress}
```

```
-LINE-IDC-PL-SL-

00978   C  1  0 { Title: FindNextPort
00979   C  1  0
00980   C  1  0   Purpose: Find the next IO Port
00981   C  1  0
00982   C  1  0   Author: William A. Chapman      Date: October 28, 1986
00983   C  1  0
00984   C  1  0   Inputs: PortIndex
00985   C  1  0           PortCount
00986   C  1  0           FinalPortAddress
00987   C  1  0
00988   C  1  0   Outputs: PortIndex
00989   C  1  0           Done
00990   C  1  0
00991   C  1  0   Procedures Invoked: SearchForPortAddress
00992   C  1  0 }
00993      2  0 procedure FindNextPort( var FNPPortIndex: integer;
00994      2  0           var FNPPortCount: integer;
00995      2  0           FNPFinalPortAddress: unsigned;
00996      2  0           var FNPNoMorePorts: boolean);
00997      2  0
00998      2  0 var
00999      2  0     FNPAddressIncrement: integer;
01000      2  0     FNPDone: boolean;
01001      2  0     FNPCurrentAddress: unsigned;
01002      2  0     FNPPortFound: boolean;
01003      2  0     FNPMaxCount: integer;
01004      2  0
01005      2  1 begin
01006      2  1     FNPNoMorePorts := false;
01007      2  1     if ( IOPortMap[ FNPPortIndex].LowerUpperIndicator = LowerPortIndicator) then
01008      2  1   FNPCurrentAddress := IOPortMap[ FNPPortIndex].Address
01009      2  1     else FNPCurrentAddress := IOPortMap[ IOPortMap[ FNPPortIndex].AuxIndex].Address;
01010      2  1     FNPAddressIncrement := IOPortMap[ FNPPortIndex].PortWidth + 1;
01011      2  1
01012      2  1     FNPDone := false;
01013      2  1
01014      2  2     if (FNPPortCount = NoPorts) then begin
01015      2  3   while (not FNPDone) do begin
01016      2  3       FNPCurrentAddress := FNPCurrentAddress + FNPAddressIncrement;
01017      2  3       FNPAddressIncrement := 1;
01018      2  4       if (FNPCurrentAddress > FNPFinalPortAddress) then begin
01019      2  4           FNPDone := true;
01020      2  4           FNPNoMorePorts := true;
01021      2  4       end
```

```
01022      2  3       else SearchForPortAddress( FNPCurrentAddress, IOPortMap, FNPDone, FNPPortIndex);
01023      2  2   end; {while not FNPDone}
01024      2  2     end
```

```
-LINE-IDC-PL-SL-

01026       2  2     else begin
01027       2  2   FNPPortCount := FNPPortCount - 1;
01028       2  2   if (FNPPortCount <= NoPorts) then FNPNoMorePorts := true
01029       2  3   else begin
01030       2  3       FNPMaxCount := MaxNumberOfIOPorts;
01031       2  4       while (not FNPDone)  do begin
01032       2  4           FNPCurrentAddress := FNPCurrentAddress + FNPAddressIncrement;
01033       2  4           FNPAddressIncrement := 1;
01034       2  4           SearchForPortAddress( FNPCurrentAddress, IOPortMap, FNPPortFound, FNPPortIndex);
01035       2  4           if (FNPPortFound = true) then FNPDone := true
01036       2  5           else begin
01037       2  5               FNPMaxCount := FNPMaxCount - 1;
01038       2  6               if (FNPMaxCount < FirstIOPortIndex) then begin
01039       2  6                   FNPDone := true;
01040       2  6                   FNPNoMorePorts := true;
01041       2  5               end;
01042       2  4           end;
01043       2  3       end; {while not FNPDone}
01044       2  2   end; {else PortCount <= NoPorts}
01045       2  1     end;
01046       1  0 end;
```

```
-LINE-IDC-PL-SL-

01048       1  0     %include 'Get8BitsOfData.subpas/list'
01049  I C  1  0 { Title: Get8BitsofData
01050  I C  1  0
01051  I C  1  0   Purpose: Retreive 8 bits of data from the IO data buffer
01052  I C  1  0            and return to the calling routine.
01053  I C  1  0
01054  I C  1  0   Author: William A. Chapman      Date: May 16, 1986
01055  I C  1  0
01056  I C  1  0   Inputs: IO data is read from IO data buffer.
01057  I C  1  0
01058  I C  1  0   Outputs: Data is returned to calling routine.
01059  I C  1  0
01060  I C  1  0   Procedures Invoked: MaskShiftRight
01061  I C  1  0 }
01062  I    2  0 procedure Get8BitsofData( PortPointer: integer;
01063  I    2  0           Location: integer;
01064  I    2  0           var Value: unsigned);
01065  I    2  0
01066  I    2  0
01067  I    2  0 var
01068  I    2  0     WordPointer: integer;
01069  I    2  0     BytePointer: integer;
01070  I    2  0
01071  I    2  0
01072  I    2  0
01073  I    2  1 begin
01074  I    2  1     WordPointer := (((Location - 1) div BytesPerWord) + 1);
01075  I    2  1     BytePointer := ((Location - 1)  rem BytesPerWord);
01076  I    2  1     Value := NullDatum;
01077  I    2  1
01078  I    2  2     case BytePointer of
01079  I    2  2
01080  I    2  2     0: Value:= MaskShiftRight( IOData[ PortPointer, WordPointer],
01081  I    2  2           Byte0Mask, Byte0Divisor);
01082  I    2  2
01083  I    2  2     1: Value:= MaskShiftRight( IOData[ PortPointer, WordPointer],
01084  I    2  2           Byte1Mask, Byte1Divisor);
01085  I    2  2
01086  I    2  2     2: Value:= MaskShiftRight( IOData[ PortPointer, WordPointer],
01087  I    2  2           Byte2Mask, Byte2Divisor);
01088  I    2  2
01089  I    2  2     3: Value:= MaskShiftRight( IOData[ PortPointer, WordPointer],
01090  I    2  2           Byte3Mask, Byte3Divisor);
01091  I    2  2
```

```
01092 I    2  2     otherwise writeln('Get8BitsofData: Invalid byte pointer:  , BytePointer:1);
01093 I    2  2
01094 I    2  1     end; {case BytePointer}
01095 I    2  1
01096 I    1  0 end; {procedure Get8BitsofData}
```

```
-LINE-IDC-PL-SL-

01098       1  0     %include 'Put8BitsOfData.subpas/list'
01099 I C   1  0 { Title: Put8BitsofData
01100 I C   1  0
01101 I C   1  0   Purpose: Take 8 bits of data passed from the calling routine
01102 I C   1  0            and store it in the IO data buffer.
01103 I C   1  0
01104 I C   1  0   Author: William A. Chapman      Date: May 17, 1986
01105 I C   1  0
01106 I C   1  0   Inputs: Input Data is obtained from the calling routine.
01107 I C   1  0
01108 I C   1  0   Outputs: Data is stored in the IO data buffer.
01109 I C   1  0
01110 I C   1  0   Procedures Invoked: none
01111 I C   1  0 }
01112 I     2  0 procedure Put8BitsofData( PortPointer: integer;
01113 I     2  0           Location: integer;
01114 I     2  0           Value: unsigned);
01115 I     2  0
01116 I     2  0 var
01117 I     2  0     Unchanged: unsigned;
01118 I     2  0     NewDatum: unsigned;
01119 I     2  0
01120 I     2  0     WordPointer: integer;
01121 I     2  0     BytePointer: integer;
```

```
-LINE-IDC-PL-SL-

01123 I    2  1 begin
01124 I    2  1     WordPointer := (((Location - 1) div BytesPerWord) + 1);
01125 I    2  1     BytePointer := ((Location - 1)  rem BytesPerWord);
01126 I    2  1
01127 I    2  2     case BytePointer of
01128 I    2  2
01129 I    2  3     0: begin
01130 I    2  3   Unchanged := uand( IOData[PortPointer,WordPointer], unot(Byte0Mask));
01131 I    2  3   NewDatum := uand( (Value * Byte0Multiplier), Byte0Mask);
01132 I    2  3   IOData[PortPointer,WordPointer] := uor( Unchanged, NewDatum);
01133 I    2  2        end;
01134 I    2  2
01135 I    2  3     1: begin
01136 I    2  3   Unchanged := uand( IOData[PortPointer,WordPointer], unot(Byte1Mask));
01137 I    2  3   NewDatum := uand( (Value * Byte1Multiplier), Byte1Mask);
01138 I    2  3   IOData[PortPointer,WordPointer] := uor( Unchanged, NewDatum);
01139 I    2  2        end;
01140 I    2  2
01141 I    2  3     2: begin
01142 I    2  3   Unchanged := uand( IOData[PortPointer,WordPointer], unot(Byte2Mask));
01143 I    2  3   NewDatum := uand( (Value * Byte2Multiplier), Byte2Mask);
01144 I    2  3   IOData[PortPointer,WordPointer] := uor( Unchanged, NewDatum);
01145 I    2  2        end;
01146 I    2  2
01147 I    2  3     3: begin
01148 I    2  3   Unchanged := uand( IOData[PortPointer,WordPointer], unot(Byte3Mask));
01149 I    2  3   NewDatum := uand( (Value * Byte3Multiplier), Byte3Mask);
01150 I    2  3   IOData[PortPointer,WordPointer] := uor( Unchanged, NewDatum);
01151 I    2  2        end;
01152 I    2  2
01153 I    2  2     otherwise writeln( 'Put8BitsOfData: Invalid Byte Pointer:  , BytePointer:1);
01154 I    2  2
01155 I    2  1     end; {case BytePointer]
01156 I    1  0 end; {Put8BitsofData]
```

```
-LINE-IDC-PL-SL-

01158       1  1 begin
01159       1  1     Punctuation := [' ', '+', ',', '-', ':', '=' ];
01160       1  1     Numerals := ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9' ];
01161       1  1     HexNumerals := ['0', '1 , '2', '3', '4', '5', '6', '7', '8', '9',
01162       1  1   'A', 'B', 'C', 'D', 'E', 'F', 'a', 'b', 'c', 'd', 'e', 'f'];
01163       1  1     Yes := [ 'Y', 'y', '1'];
01164       1  1     Continue := [ Breakpoint, Call, Halt, Normal, Return ];
01165       1  1
01166       1  1     KeywordFileName := 'KeywordLUTFile';
01167       1  1     open( KeywordLookUpTable, KeywordFileName, readonly);
01168       1  1     reset( KeywordLookUpTable);
01169       1  1     for Loop := FirstKeywordEntry to LastKeywordEntry do
01170       1  1   read( KeywordLookUpTable, KeywordLUT[ Loop]);
01171       1  1
01172       1  2     for BreakpointLoop := FirstBreakpoint to LastBreakpoint do begin
01173       1  2   BreakPointTable[ BreakpointLoop].Activated := DeActivate;
01174       1  2   BreakPointTable[ BreakpointLoop].Encountered := DeActivate;
01175       1  1     end;
01176       1  1
01177   C   1  1     {Activate Control D "listening"}
01178       1  1
01179       1  1     ControlMask[1] := 0;
01180       1  1     ControlMask[2] := CTRLD;
01181       1  1
01182       1  1     SysStat := $assign( devnam := 'TT', chan := ChannelID[1]);
01183       1  1     if (not odd( SysStat)) then Lib$Stop( SysStat);
01184       1  1
01185       1  1     SysStat := $assign( devnam := 'TT', chan := ChannelID[2]);
01186       1  1     if (not odd( SysStat)) then Lib$Stop( SysStat);
01187       1  1
01188       1  1     IOFunction := IO$_SETMODE + IO$M_OUTBAND;
01189       1  1     SysStat := $QIOW( efn := 1, chan := ChannelID[1], func := IOFunction,
01190       1  1   p1 := %immed ASTD, p2 := %ref ControlMask);
01191       1  1     if (not odd( SysStat)) then Lib$Stop( SysStat);
01192       1  1
01193       1  1     EOLNFlag := true;
01194       1  1
01195       1  1     ExitFlag := false;
01196       1  1
01197       1  2     while (not ExitFlag) do begin
01198       1  2
01199       1  2   if (EOLNFlag = true) then
01200       1  2       FetchInputLine( SourceLine, SourcePointer, SourceLength);
01201       1  2
```

```
01202     1  2   SyntaxInterpreter( Keyword, SourceLine, SourcePointer,
01203     1  2       SourceLength, EOLNFlag);
01204     1  2
01205     1  2
01206     1  3   case Keyword.Class of
```

```
-LINE-IDC-PL-SL-

01208       1  3   %include 'MemoryClass.subpas/list'
01209 I     1  4   MemoryClass: begin {set / display memory}
01210 I     1  4     Format := Keyword;
01211 I     1  5     if (EOLNFlag = false) then begin
01212 I     1  5       GetAddress( StartSegment, StartAddress, SourceLine, SourcePointer,
01213 I     1  5           SourceLength, EOLNFlag, NullAddress);
01214 I     1  5       OperatingLength := UnitLength;
01215 I     1  5       Command := DisplayCommand;
01216 I     1  5
01217 I     1  6       while ( EOLNFlag = false ) do begin
01218 I     1  6           SyntaxInterpreter( Modifier, SourceLine, SourcePointer,
01219 I     1  6               SourceLength, EOLNFlag);
01220 I     1  7           case Modifier.Class of
01221 I     1  7
01222 I     1  7           BlankClass: ;
01223 I     1  7
01224 I     1  8           QualifierClass: begin
01225 I     1  9               case Modifier.ID of
01226 I     1  9
01227 I     1 10               LengthID: begin
01228 I     1 10                   SyntaxInterpreter( LengthInput, SourceLine, SourcePointer,
01229 I     1 10                       SourceLength, EOLNFlag);
01230 I     1 10                   OperatingLength := uint( LengthInput.Value);
01231 I     1  9               end; {case LengthID}
01232 I     1  9
01233 I     1 10               ToID: begin
01234 I     1 10                   GetAddress( EndSegment, EndAddress, SourceLine,
01235 I     1 10                       SourcePointer, SourceLength, EOLNFlag, NullAddress);
01236 I     1 11                   if (NullAddress = true) then begin
01237 I     1 11                       writeln( 'Invalid input - address required');
01238 I     1 11                       Command := ErrorCommand;
01239 I     1 10                   end;
01240 I     1 10                   OperatingLength := uint((( EndSegment * 16) + EndAddress)
01241 I     1 10                       - (( StartSegment * 16) + StartAddress));
01242 I     1 11                   case Format.ID of
01243 I     1 11
01244 I     1 11                     ByteID, BooleanID, SIntegerID:
01245 I     1 11                       OperatingLength := OperatingLength + 1;
01246 I     1 11
01247 I     1 11                     WordID, IntegerID:
01248 I     1 11                       OperatingLength := utrunc( OperatingLength / 2) + 1;
01249 I     1 11
01250 I     1 11                     PointerID:
01251 I     1 11                       OperatingLength := utrunc( OperatingLength / 4) + 1;
```

```
01252 I    1 11
01253 I    1 12                    otherwise begin
01254 I    1 12                      writeln( 'Invalid Format ID:   , Format.ID:1);
01255 I    1 12                      Command := ErrorCommand;
01256 I    1 12                      EOLNFlag := true;
01257 I    1 11                    end;
01258 I    1 11
01259 I    1 10                end; {case Format.ID}
01260 I    1  9            end; {case ToID}
```

```
-LINE-IDC-PL-SL-

01262 I    1 10             otherwise begin
01263 I    1 10                 writeln( 'Invalid qualifier: "', Modifier.Name, '"');
01264 I    1 10                 Command := ErrorCommand;
01265 I    1 10                 EOLNFlag := true;
01266 I    1  9             end;
01267 I    1  9
01268 I    1  8             end; {case Modifier.ID}
01269 I    1  7         end; {case Qualifier Class}
01270 I    1  7
01271 I    1  8         OperatorClass: begin
01272 I    1  9             case Modifier.ID of
01273 I    1  9
01274 I    1 10             AssignmentID: begin
01275 I    1 10                 GetData( DataArray, DataArrayEntries, SourceLine,
01276 I    1 10                     SourcePointer, SourceLength, EOLNFlag, Format.ID);
01277 I    1 10                 Command := AssignmentCommand;
01278 I    1  9             end; {case Assign.ID}
01279 I    1  9
01280 I    1 10             otherwise begin
01281 I    1 10                 writeln( 'Invalid assignment operator: "', Modifier.Name,  '"');
01282 I    1 10                 Command := ErrorCommand;
01283 I    1 10                 EOLNFlag := true;
01284 I    1  9             end; {otherwise}
01285 I    1  9
01286 I    1  8             end; {case Modifier.ID}
01287 I    1  8
01288 I    1  7         end; {case Operator Class}
01289 I    1  7
01290 I    1  8         otherwise begin
01291 I    1  8             writeln( 'Invalid modifier: "', Modifier.Name, '"');
01292 I    1  8             Command := ErrorCommand;
01293 I    1  8             EOLNFlag := true;
01294 I    1  7         end; {otherwise}
01295 I    1  7
01296 I    1  6         end; {case Modifier.Class}
01297 I    1  5     end; {while ( EOLNFlag = false)}
```

```
-LINE-IDC-PL-SL-

01299 I     1  6       case Command of
01300 I     1  6
01301 I     1  7       DisplayCommand: begin
01302 I     1  7           DisplaySegment := StartSegment;
01303 I     1  7           DisplayAddress := StartAddress;
01304 I     1  8           for DisplayLoop := 1 to int( OperatingLength) do begin
01305 I     1  8               write( hex( DisplaySegment, 4, 4), ':');
01306 I     1  8               write( hex( DisplayAddress, 4, 4),    ');
01307 I     1  8               FetchMemory( DisplayAddress, DisplaySegment, DisplayData, Stopcode);
01308 I     1  8               DisplayAddress := DisplayAddress + 1;
01309 I     1  9               case Format.ID of
01310 I     1  9
01311 I     1  9                   BooleanID: if (odd( DisplayData) = true) then write( 'True')
01312 I     1  9                       else write( 'False');
01313 I     1  9
01314 I     1  9                   ByteID: write( hex(DisplayData, 2, 2));
01315 I     1  9
01316 I     1 10                   IntegerID: begin
01317 I     1 10                       FetchMemory( DisplayAddress, DisplaySegment, HighDisplayData, Stopcode);
01318 I     1 10                       DisplayAddress := DisplayAddress + 1;
01319 I     1 10                       DisplayData := uor( uand(( HighDisplayData * WordMultiplier),
01320 I     1 10                           High8Bits), DisplayData);
01321 I     1 10                       DisplayIntegerData := SignExtend15( DisplayData);
01322 I     1 10                       write( DisplayIntegerData:1);
01323 I     1  9                   end; {IntegerID}
01324 I     1  9
01325 I     1 10                   PointerID: begin
01326 I     1 10                       FetchMemory( DisplayAddress, DisplaySegment, HighDisplayData, Stopcode);
01327 I     1 10                       DisplayAddress := DisplayAddress + 1;
01328 I     1 10                       DisplayData := uor( uand(( HighDisplayData * WordMultiplier),
01329 I     1 10                           High8Bits), DisplayData);
01330 I     1 10                       write( hex( DisplayData, 4, 4),    );
01331 I     1 10
01332 I     1 10                       FetchMemory( DisplayAddress, DisplaySegment, DisplayData, Stopcode);
01333 I     1 10                       DisplayAddress := DisplayAddress + 1;
01334 I     1 10                       FetchMemory( DisplayAddress, DisplaySegment, HighDisplayData, Stopcode);
01335 I     1 10                       DisplayAddress := DisplayAddress + 1;
01336 I     1 10                       DisplayData := uor( uand(( HighDisplayData * WordMultiplier),
01337 I     1 10                           High8Bits), DisplayData);
01338 I     1 10                       write( hex( DisplayData, 4, 4));
01339 I     1  9                   end; {PointerID}
01340 I     1  9
01341 I     1 10                   SIntegerID: begin
01342 I     1 10                       DisplayIntegerData := SignExtend7( DisplayData);
```

```
01343 I    1 10                 write( DisplayIntegerData:1);
01344 I    1  9               end; {SIntegerID}
```

```
-LINE-IDC-PL-SL-

01346 I     1 10              WordID: begin
01347 I     1 10                  FetchMemory( DisplayAddress, DisplaySegment, HighDisplayData, Stopcode);
01348 I     1 10                  DisplayAddress := DisplayAddress + 1;
01349 I     1 10                  DisplayData := uor( uand(( HighDisplayData * WordMultiplier),
01350 I     1 10                      High8Bits), DisplayData);
01351 I     1 10                  write( hex( DisplayData, 4, 4));
01352 I     1  9              end; {WordID}
01353 I     1  9
01354 I     1  8              end; {case Format.ID}
01355 I     1  8              writeln;
01356 I     1  7          end; {for DisplayLoop}
01357 I     1  6      end; {Display Command}
01358 I     1  6
01359 I     1  7      AssignmentCommand: begin
01360 I     1  7          AssignSegment := StartSegment;
01361 I     1  7          AssignAddress := StartAddress;
01362 I     1  7          DataArrayPointer := StartOfData;
01363 I     1  8          for AssignLoop := 1 to int( OperatingLength) do begin
01364 I     1  9              case Format.ID of
01365 I     1  9
01366 I     1 10                  BooleanID, ByteID, SIntegerID: begin
01367 I     1 10                      GetAssignData( AssignData, DataArrayPointer, DataArray,
01368 I     1 10                          DataArrayEntries);
01369 I     1 10                      AssignDataLow := uand( AssignData, Low8Bits);
01370 I     1 10                      StoreMemory( AssignAddress, AssignSegment, AssignDataLow, Stopcode);
01371 I     1 10                      AssignAddress := AssignAddress + 1;
01372 I     1  9                  end; {single byte assignments}
01373 I     1  9
01374 I     1 10                  IntegerID, WordID: begin
01375 I     1 10                      GetAssignData( AssignData, DataArrayPointer, DataArray,
01376 I     1 10                          DataArrayEntries);
01377 I     1 10                      AssignDataLow := uand( AssignData, Low8Bits);
01378 I     1 10                      StoreMemory( AssignAddress, AssignSegment, AssignDataLow, Stopcode);
01379 I     1 10                      AssignAddress := AssignAddress + 1;
01380 I     1 10
01381 I     1 10                      AssignDataHigh := MaskShiftRight( AssignData,
01382 I     1 10                          High8Bits, WordMultiplier);
01383 I     1 10                      StoreMemory( AssignAddress, AssignSegment, AssignDataHigh, Stopcode);
01384 I     1 10                      AssignAddress := AssignAddress + 1;
01385 I     1  9                  end; {double byte assignments}
```

```
-LINE-IDC-PL-SL-

01387 I     1 10              PointerID: begin
01388 I C   1 10                {Segment value}
01389 I     1 10                  GetAssignData( AssignData, DataArrayPointer, DataArray,
01390 I     1 10                      DataArrayEntries);
01391 I     1 10                  AssignDataLow := uand( AssignData, Low8Bits);
01392 I     1 10                  StoreMemory( AssignAddress, AssignSegment, AssignDataLow, Stopcode);
01393 I     1 10                  AssignAddress := AssignAddress + 1;
01394 I     1 10
01395 I     1 10                  AssignDataHigh := MaskShiftRight( AssignData,
01396 I     1 10                      High8Bits, WordMultiplier);
01397 I     1 10                  StoreMemory( AssignAddress, AssignSegment, AssignDataHigh, Stopcode);
01398 I     1 10                  AssignAddress := AssignAddress + 1;
01399 I     1 10
01400 I C   1 10                {Address Value}
01401 I     1 10                  GetAssignData( AssignData, DataArrayPointer, DataArray,
01402 I     1 10                      DataArrayEntries);
01403 I     1 10                  AssignDataLow := uand( AssignData, Low8Bits);
01404 I     1 10                  StoreMemory( AssignAddress, AssignSegment, AssignDataLow, Stopcode);
01405 I     1 10                  AssignAddress := AssignAddress + 1;
01406 I     1 10
01407 I     1 10                  AssignDataHigh := MaskShiftRight( AssignData,
01408 I     1 10                      High8Bits, WordMultiplier);
01409 I     1 10                  StoreMemory( AssignAddress, AssignSegment, AssignDataHigh, Stopcode);
01410 I     1 10                  AssignAddress := AssignAddress + 1;
01411 I     1  9              end; {quad byte asignments}
01412 I     1  9
01413 I     1  8              end; {case Format.ID}
01414 I     1  7          end; {for AssignLoop}
01415 I     1  6      end; {Assignment Command}
01416 I     1  6
01417 I     1  6      ErrorCommand: ;
01418 I     1  6
01419 I     1  6      otherwise writeln( 'Invalid memory access command');
01420 I     1  6
01421 I     1  5      end; {case Command}
01422 I     1  5    end {if EOLNFlag = false}
01423 I     1  4    else writeln( 'Command is missing a memory address' );
01424 I     1  3  end; {'M'}
```

```
-LINE-IDC-PL-SL-

01426       1  3   BlankClass: ; {eliminate blank errors}
01427       1  3
01428       1  3   %include 'StackClass.subpas/list'
01429 I     1  4   StackClass: begin {display stack}
01430 I     1  4       FetchRegPtr( SSWidth, SSID, SSValue, StopCode);
01431 I     1  4       FetchRegPtr( SPWidth, SPID, SPValue, StopCode);
01432 I     1  4       if (EOLNFlag = true) then StackDisplayLength := 2
01433 I     1  5       else begin
01434 I     1  5           SyntaxInterpreter( StackDisplay, SourceLine, SourcePointer,
01435 I     1  5               SourceLength, EOLNFlag);
01436 I     1  5           StackDisplayLength := int( StackDisplay.Value * 2);
01437 I     1  4       end;
01438 I     1  4
01439 I     1  4       StackDisplayIndex := int( SPValue);
01440 I     1  5       while (StackDisplayIndex < (int( SPValue) + StackDisplayLength)) do begin
01441 I     1  5           write( ' ', hex( SSValue, 4, 4));
01442 I     1  5           write(  :', hex( StackDisplayIndex, 4, 4));
01443 I     1  5           FetchMemory( uint(StackDisplayIndex), SSValue, LowStackData, StopCode);
01444 I     1  5           StackDisplayIndex := StackDisplayIndex + 1;
01445 I     1  5           FetchMemory( uint(StackDisplayIndex), SSValue, HighStackData, StopCode);
01446 I     1  5           StackDisplayIndex := StackDisplayIndex + 1;
01447 I     1  5           StackData := uor((HighStackData * WordMultiplier), LowStackData);
01448 I     1  5           write( '  ', hex( StackData, 4, 4));
01449 I     1  5           writeln;
01450 I     1  4       end; {while StackDisplayIndex ... begin}
01451 I     1  3   end;
```

```
-LINE-IDC-PL-SL-

01454       1  3   %include 'RegisterClass.subpas/list'
01455 I     1  4   RegisterClass: begin {set / display register}
01456 I     1  4       RegisterWidth := Keyword.Width;
01457 I     1  4       RegisterID := Keyword.ID;
01458 I     1  4       Identifier := (RegisterWidth * 8) + RegisterID;
01459 I     1  5       if (EOLNFlag = false) then begin
01460 I     1  5           SyntaxInterpreter( Operator, SourceLine, SourcePointer,
01461 I     1  5               SourceLength, EOLNFlag);
01462 I     1  5           if ((Operator.Class = OperatorClass)
01463 I     1  6               and (Operator.ID = AssignmentID)) then begin
01464 I     1  6               SyntaxInterpreter( NewRegister, SourceLine,
01465 I     1  6                   SourcePointer, SourceLength, EOLNFlag);
01466 I     1  7               case Identifier of
01467 I     1  7               0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19:
01468 I     1  7                 StoreRegPtr( RegisterWidth, RegisterID, NewRegister.Value, StopCode);
01469 I     1  7               20: IP := NewRegister.Value;
01470 I     1  8               21: with Flags do begin {flag}
01471 I     1  8                 AssembledFlags := NewRegister.Value;
01472 I     1  8                 if (uand(AssembledFlags,CarryFlag) = 0)
01473 I     1  8                   then Carry := Clear
01474 I     1  8                   else Carry := SetHigh;
01475 I     1  8                 if (uand(AssembledFlags,ParityFlag) = 0)
01476 I     1  8                   then Parity := Clear
01477 I     1  8                   else Parity := SetHigh;
01478 I     1  8                 if (uand(AssembledFlags,AuxCarryFlag) = 0)
01479 I     1  8                   then AuxCarry := Clear
01480 I     1  8                   else AuxCarry := SetHigh;
01481 I     1  8                 if (uand(AssembledFlags,ZeroFlag) = 0)
01482 I     1  8                   then Zero := Clear
01483 I     1  8                   else Zero := SetHigh;
01484 I     1  8                 if (uand(AssembledFlags,SignFlag) = 0)
01485 I     1  8                   then Sign := Clear
01486 I     1  8                   else Sign := SetHigh;
01487 I     1  8                 if (uand(AssembledFlags,TrapFlag) = 0)
01488 I     1  8                   then Trap := Clear
01489 I     1  8                   else Trap := SetHigh;
01490 I     1  8                 if (uand(AssembledFlags,InterruptFlag) = 0)
01491 I     1  8                   then Interrupt := Clear
01492 I     1  8                   else Interrupt := SetHigh;
01493 I     1  8                 if (uand(AssembledFlags,DirectionFlag) = 0)
01494 I     1  8                   then Direction := Clear
01495 I     1  8                   else Direction := SetHigh;
01496 I     1  8                 if (uand(AssembledFlags,OverflowFlag) = 0)
01497 I     1  8                   then Overflow := Clear
```

```
01498 I    1  8                    else Overflow := SetHigh;
01499 I    1  7                end; {with Flags}
01500 I    1  7                otherwise writeln( 'Invalid Register: "', Keyword.Name, '"');
01501 I    1  6                end; {case Identifier}
01502 I    1  6            end {if OperatorClass and Assignment}
01503 I    1  6            else begin
01504 I    1  6                writeln( 'Invalid Operator: "', Operator.Name, '"');
01505 I    1  5            end;
01506 I    1  5        end {if EOLNFlag = false}
```

```
-LINE-IDC-PL-SL-

01508 I     1  5       else begin
01509 I     1  6           case Identifier of
01510 I     1  6
01511 I     1  7           0: begin
01512 I     1  7             FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01513 I     1  7             writeln( 'AL = ', hex( DisplayRegister.Value, 2, 2));
01514 I     1  6           end;
01515 I     1  6
01516 I     1  7           1: begin
01517 I     1  7             FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01518 I     1  7             writeln( 'CL = ', hex( DisplayRegister.Value, 2, 2));
01519 I     1  6           end;
01520 I     1  6
01521 I     1  7           2: begin
01522 I     1  7             FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01523 I     1  7             writeln( 'DL = ', hex( DisplayRegister.Value, 2, 2));
01524 I     1  6           end;
01525 I     1  6
01526 I     1  7           3: begin
01527 I     1  7             FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01528 I     1  7             writeln( 'BL = ', hex( DisplayRegister.Value, 2, 2));
01529 I     1  6           end;
01530 I     1  6
01531 I     1  7           4: begin
01532 I     1  7             FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01533 I     1  7             writeln( 'AH = ', hex( DisplayRegister.Value, 2, 2));
01534 I     1  6           end;
01535 I     1  6
01536 I     1  7           5: begin
01537 I     1  7             FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01538 I     1  7             writeln( 'CH = ', hex( DisplayRegister.Value, 2, 2));
01539 I     1  6           end;
01540 I     1  6
01541 I     1  7           6: begin
01542 I     1  7             FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01543 I     1  7             writeln( 'DH = ', hex( DisplayRegister.Value, 2, 2));
01544 I     1  6           end;
01545 I     1  6
01546 I     1  7           7: begin
01547 I     1  7             FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01548 I     1  7             writeln( 'BH = ', hex( DisplayRegister.Value, 2, 2));
01549 I     1  6           end;
01550 I     1  6
01551 I     1  7           8: begin
```

```
01552 I    1  7            FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01553 I    1  7            writeln( 'AX = ', hex( DisplayRegister.Value, 4, 4));
01554 I    1  6          end;
01555 I    1  6
01556 I    1  7          9: begin
01557 I    1  7            FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01558 I    1  7            writeln( 'CX = ', hex( DisplayRegister.Value, 4, 4));
01559 I    1  6          end;
```

```
-LINE-IDC-PL-SL-

01561 I     1  7          10: begin
01562 I     1  7            FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01563 I     1  7            writeln( 'DX = ', hex( DisplayRegister.Value, 4, 4));
01564 I     1  6          end;
01565 I     1  6
01566 I     1  7          11: begin
01567 I     1  7            FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01568 I     1  7            writeln( 'BX = ', hex( DisplayRegister.Value, 4, 4));
01569 I     1  6          end;
01570 I     1  6
01571 I     1  7          12: begin
01572 I     1  7            FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01573 I     1  7            writeln( 'SP = ', hex( DisplayRegister.Value, 4, 4));
01574 I     1  6          end;
01575 I     1  6
01576 I     1  7          13: begin
01577 I     1  7            FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01578 I     1  7            writeln( 'BP = ', hex( DisplayRegister.Value, 4, 4));
01579 I     1  6          end;
01580 I     1  6
01581 I     1  7          14: begin
01582 I     1  7            FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01583 I     1  7            writeln( 'SI = ', hex( DisplayRegister.Value, 4, 4));
01584 I     1  6          end;
01585 I     1  6
01586 I     1  7          15: begin
01587 I     1  7            FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01588 I     1  7            writeln( 'DI = ', hex( DisplayRegister.Value, 4, 4));
01589 I     1  6          end;
01590 I     1  6
01591 I     1  7          16: begin
01592 I     1  7            FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01593 I     1  7            writeln( 'ES = ', hex( DisplayRegister.Value, 4, 4));
01594 I     1  6          end;
01595 I     1  6
01596 I     1  7          17: begin
01597 I     1  7            FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01598 I     1  7            writeln( 'CS = ', hex( DisplayRegister.Value, 4, 4));
01599 I     1  6          end;
01600 I     1  6
01601 I     1  7          18: begin
01602 I     1  7            FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01603 I     1  7            writeln( 'SS = ', hex( DisplayRegister.Value, 4, 4));
01604 I     1  6          end;
```

```
01605 I    1  6
01606 I    1  7          19: begin
01607 I    1  7            FetchRegPtr( RegisterWidth, RegisterID,DisplayRegister.Value, StopCode);
01608 I    1  7            writeln( 'DS = ', hex( DisplayRegister.Value, 4, 4));
01609 I    1  6          end;
```

```
-LINE-IDC-PL-SL-

01611 I     1  6          20: writeln( 'IP =  , hex( IP, 4, 4));
01612 I     1  6
01613 I     1  7          21: begin {flags}
01614 I     1  7            AssembledFlags := ClearFlags;
01615 I     1  8            with Flags do begin
01616 I     1  8              if (Carry = SetHigh)
01617 I     1  8                then AssembledFlags := uor(AssembledFlags,CarryFlag);
01618 I     1  8              if (Parity = SetHigh)
01619 I     1  8                then AssembledFlags := uor(AssembledFlags,ParityFlag);
01620 I     1  8              if (AuxCarry = SetHigh)
01621 I     1  8                then AssembledFlags := uor(AssembledFlags,AuxCarryFlag);
01622 I     1  8              if (Zero = SetHigh)
01623 I     1  8                then AssembledFlags := uor(AssembledFlags,ZeroFlag);
01624 I     1  8              if (Sign = SetHigh)
01625 I     1  8                then AssembledFlags := uor(AssembledFlags,SignFlag);
01626 I     1  8              if (Trap = SetHigh)
01627 I     1  8                then AssembledFlags := uor(AssembledFlags,TrapFlag);
01628 I     1  8              if (Interrupt = SetHigh)
01629 I     1  8                then AssembledFlags := uor(AssembledFlags,InterruptFlag);
01630 I     1  8              if (Direction = SetHigh)
01631 I     1  8                then AssembledFlags := uor(AssembledFlags,DirectionFlag);
01632 I     1  8              if (Overflow = SetHigh)
01633 I     1  8                then AssembledFlags := uor(AssembledFlags,OverflowFlag);
01634 I     1  7            end; {with Flags}
01635 I     1  7            writeln( 'RF = ', hex( AssembledFlags, 4, 4));
01636 I     1  6          end;
```

```
-LINE-IDC-PL-SL-

01638 I    1  7          24: begin {all registers}
01639 I    1  7            FetchRegPtr( AXWidth, AXId, DisplayValue, StopCode);
01640 I    1  7            write( 'AX = ', hex( DisplayValue, 4, 4), '  ');
01641 I    1  7            FetchRegPtr( BXWidth, BXId, DisplayValue, StopCode);
01642 I    1  7            write( 'BX = ', hex( DisplayValue, 4, 4), '  ');
01643 I    1  7            FetchRegPtr( CXWidth, CXId, DisplayValue, StopCode);
01644 I    1  7            write( 'CX = ', hex( DisplayValue, 4, 4), '  ');
01645 I    1  7            FetchRegPtr( DXWidth, DXId, DisplayValue, StopCode);
01646 I    1  7            writeln( 'DX = ', hex( DisplayValue, 4, 4), '  ');
01647 I    1  7            FetchRegPtr( CSWidth, CSId, DisplayValue, StopCode);
01648 I    1  7            write( 'CS = ', hex( DisplayValue, 4, 4), '  ');
01649 I    1  7            FetchRegPtr( SSWidth, SSId, DisplayValue, StopCode);
01650 I    1  7            write( 'SS = ', hex( DisplayValue, 4, 4), '  ');
01651 I    1  7            FetchRegPtr( DSWidth, DSId, DisplayValue, StopCode);
01652 I    1  7            write( 'DS = ', hex( DisplayValue, 4, 4), '  ');
01653 I    1  7            FetchRegPtr( ESWidth, ESId, DisplayValue, StopCode);
01654 I    1  7            writeln( 'ES = ', hex( DisplayValue, 4, 4), '  ');
01655 I    1  7            write( 'IP = ', hex( IP, 4, 4), '  ');
01656 I    1  7            FetchRegPtr( SPWidth, SPId, DisplayValue, StopCode);
01657 I    1  7            write( 'SP = ', hex( DisplayValue, 4, 4), '  ');
01658 I    1  7            FetchRegPtr( SIWidth, SIId, DisplayValue, StopCode);
01659 I    1  7            write( 'SI = ', hex( DisplayValue, 4, 4), '  ');
01660 I    1  7            FetchRegPtr( DIWidth, DIId, DisplayValue, StopCode);
01661 I    1  7            writeln( 'DI = ', hex( DisplayValue, 4, 4), '  ');
01662 I    1  7            AssembledFlags := ClearFlags;
01663 I    1  8            with Flags do begin
01664 I    1  8              if (Carry = SetHigh)
01665 I    1  8                then AssembledFlags := uor(AssembledFlags,CarryFlag);
01666 I    1  8              if (Parity = SetHigh)
01667 I    1  8                then AssembledFlags := uor(AssembledFlags,ParityFlag);
01668 I    1  8              if (AuxCarry = SetHigh)
01669 I    1  8                then AssembledFlags := uor(AssembledFlags,AuxCarryFlag);
01670 I    1  8              if (Zero = SetHigh)
01671 I    1  8                then AssembledFlags := uor(AssembledFlags,ZeroFlag);
01672 I    1  8              if (Sign = SetHigh)
01673 I    1  8                then AssembledFlags := uor(AssembledFlags,SignFlag);
01674 I    1  8              if (Trap = SetHigh)
01675 I    1  8                then AssembledFlags := uor(AssembledFlags,TrapFlag);
01676 I    1  8              if (Interrupt = SetHigh)
01677 I    1  8                then AssembledFlags := uor(AssembledFlags,InterruptFlag);
01678 I    1  8              if (Direction = SetHigh)
01679 I    1  8                then AssembledFlags := uor(AssembledFlags,DirectionFlag);
01680 I    1  8              if (Overflow = SetHigh)
01681 I    1  8                then AssembledFlags := uor(AssembledFlags,OverflowFlag);
```

```
-LINE-IDC-PL-SL-

01694       1  3   %include 'FlagClass.subpas/list'
01695 I     1  4   FlagClass: begin {set / display flags}
01696 I     1  4       FlagID := Keyword.ID;
01697 I     1  5       if (EOLNFlag = false) then begin
01698 I     1  5           SyntaxInterpreter( Operator, SourceLine, SourcePointer,
01699 I     1  5               SourceLength, EOLNFlag);
01700 I     1  5           if ((Operator.Class = OperatorClass)
01701 I     1  6               and (Operator.ID = AssignmentID)) then begin
01702 I     1  6               SyntaxInterpreter( NewFlag, SourceLine,
01703 I     1  6                   SourcePointer, SourceLength, EOLNFlag);
01704 I     1  7               case FlagID of
01705 I     1  7               0: if (NewFlag.Value = 0) then Flags.Carry := Clear
01706 I     1  7                   else Flags.Carry := SetHigh;
01707 I     1  7               1: if (NewFlag.Value = 0) then Flags.Parity := Clear
01708 I     1  7                   else Flags.Parity := SetHigh;
01709 I     1  7               2: if (NewFlag.Value = 0) then Flags.AuxCarry := Clear
01710 I     1  7                   else Flags.AuxCarry := SetHigh;
01711 I     1  7               3: if (NewFlag.Value = 0) then Flags.Zero := Clear
01712 I     1  7                   else Flags.Zero := SetHigh;
01713 I     1  7               4: if (NewFlag.Value = 0) then Flags.Sign := Clear
01714 I     1  7                   else Flags.Sign := SetHigh;
01715 I     1  7               5: if (NewFlag.Value = 0) then Flags.Trap := Clear
01716 I     1  7                   else Flags.Trap := SetHigh;
01717 I     1  7               6: if (NewFlag.Value = 0) then Flags.Interrupt := Clear
01718 I     1  7                   else Flags.Interrupt := SetHigh;
01719 I     1  7               7: if (NewFlag.Value = 0) then Flags.Direction := Clear
01720 I     1  7                   else Flags.Direction := SetHigh;
01721 I     1  7               8: if (NewFlag.Value = 0) then Flags.Overflow := Clear
01722 I     1  7                   else Flags.Overflow := SetHigh;
01723 I     1  7               otherwise writeln( 'Invalid flag identifier: "',
01724 I     1  7                   Keyword.Name, '"');
01725 I     1  6               end; {case Identifier}
01726 I     1  6           end {if OperatorClass and Assignment}
01727 I     1  6           else begin
01728 I     1  6               writeln( 'Invalid Operator: "', Operator.Name, '"');
01729 I     1  5           end;
01730 I     1  5       end {if EOLNFlag = false}
```

```
-LINE-IDC-PL-SL-

01732 I    1  5       else begin
01733 I    1  6           case FlagID of
01734 I    1  6           0: if (Flags.Carry = SetHigh) then write( 'CFL =  , FlagON:1)
01735 I    1  6             else write( 'CFL = ', FlagOFF:1);
01736 I    1  6           1: if (Flags.Parity = SetHigh) then write( 'PFL =  , FlagON:1)
01737 I    1  6             else write( 'PFL =  , FlagOFF:1);
01738 I    1  6           2: if (Flags.AuxCarry = SetHigh) then write( 'AFL =  , FlagON:1)
01739 I    1  6             else write( 'AFL = ', FlagOFF:1);
01740 I    1  6           3: if (Flags.Zero = SetHigh) then write( 'ZFL =  , FlagON:1)
01741 I    1  6             else write( 'ZFL = ', FlagOFF:1);
01742 I    1  6           4: if (Flags.Sign = SetHigh) then write( 'SFL =  , FlagON:1)
01743 I    1  6             else write( 'SFL = ', FlagOFF:1);
01744 I    1  6           5: if (Flags.Trap = SetHigh) then write( 'TFL =  , FlagON:1)
01745 I    1  6             else write( 'TFL = ', FlagOFF:1);
01746 I    1  6           6: if (Flags.Interrupt = SetHigh) then write( 'IFL =  , FlagON:1)
01747 I    1  6             else write( 'IFL = ', FlagOFF:1);
01748 I    1  6           7: if (Flags.Direction = SetHigh) then write( 'DFL =  , FlagON:1)
01749 I    1  6             else write( 'DFL = ', FlagOFF:1);
01750 I    1  6           8: if (Flags.Overflow = SetHigh) then write( 'OFL =  , FlagON:1)
01751 I    1  6             else write( 'OFL =  , FlagOFF:1);
01752 I    1  7           9: begin {all flags}
01753 I    1  7                if (Flags.Carry = SetHigh) then write( 'CFL =  ,  FlagON:1,  ,  )
01754 I    1  7                   else write( 'CFL = ', FlagOFF:1, ', ');
01755 I    1  7                if (Flags.Parity = SetHigh) then write( 'PFL =  ,  FlagON:1,  , ')
01756 I    1  7                   else write( 'PFL = ', FlagOFF:1, ',  );
01757 I    1  7                if (Flags.AuxCarry = SetHigh) then write( 'AFL =  ,  FlagON:1,  ,  )
01758 I    1  7                   else write( 'AFL = ', FlagOFF:1, ', ');
01759 I    1  7                if (Flags.Zero = SetHigh) then write( 'ZFL = ',  FlagON:1,  , ')
01760 I    1  7                   else write( 'ZFL = ', FlagOFF:1, ', ');
01761 I    1  7                if (Flags.Sign = SetHigh) then write( 'SFL =  ,  FlagON:1,  , ')
01762 I    1  7                   else write( 'SFL = ', FlagOFF:1, ', ');
01763 I    1  7                if (Flags.Trap = SetHigh) then write( 'TFL =  ,  FlagON:1,  , ')
01764 I    1  7                   else write( 'TFL = ', FlagOFF:1, ', ');
01765 I    1  7                if (Flags.Interrupt = SetHigh) then write( 'IFL =  ,  FlagON:1,  , ')
01766 I    1  7                   else write( 'IFL = ', FlagOFF:1, ', ');
01767 I    1  7                if (Flags.Direction = SetHigh) then write( 'DFL =  ,  FlagON:1,  , ')
01768 I    1  7                   else write( 'DFL = ', FlagOFF:1, ', ');
01769 I    1  7                if (Flags.Overflow = SetHigh) then write( 'OFL =  ,  FlagON:1)
01770 I    1  7                   else write( 'OFL = ', FlagOFF:1);
01771 I    1  6           end; {all flags}
01772 I    1  6           otherwise writeln( 'Invalid flag identifier: "',
01773 I    1  6               Keyword.Name, '"');
01774 I    1  5           end; {case FlagID}
01775 I    1  5       writeln;
```

```
01776 I     1  4        end; {else begin EOLNFlag = true}
01777 I     1  3    end;
```

```
-LINE-IDC-PL-SL-

01780       1  3   %include 'UtilityClass.subpas/list'
01781 I     1  4   UtilityClass: begin {utility commands}
01782 I     1  5       case Keyword.ID of
01783 I     1  5       ExitID: ExitFlag := true;
01784 I     1  5
01785 I     1  5       LogID: {manipulate journal file}
01786 I     1  6         begin
01787 I     1  6           Command := DisplayCommand;
01788 I     1  7           while (EOLNFlag = false) do begin
01789 I     1  7
01790 I     1  7               SyntaxInterpreter( Modifier, SourceLine, SourcePointer,
01791 I     1  7                   SourceLength, EOLNFlag);
01792 I     1  7
01793 I     1  8               case Modifier.Class of
01794 I     1  8
01795 I     1  8               BlankClass: ;
01796 I     1  8
01797 I     1  8
01798 I     1  9               OperatorClass: begin
01799 I     1 10                   case Modifier.ID of
01800 I     1 10
01801 I     1 11                   AssignmentID: begin
01802 I     1 11                       if (EOLNFlag = true) then Command := DeactivateCommand
01803 I     1 12                       else begin
01804 I     1 12                           ParseFilename( NullName, JournalFilename,
01805 I     1 12                               SourceLine, SourcePointer, SourceLength, EOLNFlag);
01806 I     1 12                           if (NullName = true) then Command := DeactivateCommand
01807 I     1 12                           else Command := AssignmentCommand;
01808 I     1 12                           EOLNFlag := true;
01809 I     1 11                       end;
01810 I     1 10                   end; {case Assign.ID}
01811 I     1 10
01812 I     1 11                   otherwise begin
01813 I     1 11                       writeln( 'Invalid assignment operator: "', Modifier.Name, '"');
01814 I     1 11                       Command := ErrorCommand;
01815 I     1 11                       EOLNFlag := true;
01816 I     1 10                   end; {otherwise}
01817 I     1 10
01818 I     1  9                   end; {case Modifier.ID}
01819 I     1  9
01820 I     1  8               end; {case Operator Class}
01821 I     1  8
01822 I     1  9               otherwise begin
01823 I     1  9                   writeln( 'Invalid modifier: "', Modifier.Name, '"');
```

```
01824 I    1  9                    Command := ErrorCommand;
01825 I    1  9                    EOLNFlag := true;
01826 I    1  8                end; {otherwise}
01827 I    1  8
01828 I    1  7                end; {case Modifier.Class}
01829 I    1  6            end; {while ( EOLNFlag = false)}
```

```
-LINE-IDC-PL-SL-

01831 I    1  7          case Command of
01832 I    1  7
01833 I    1  7          DisplayCommand:
01834 I    1  7              if (JournalingFlag = true) then writeln( JournalFilename)
01835 I    1  7              else writeln( 'Logging not active');
01836 I    1  7
01837 I    1  7          AssignmentCommand:
01838 I    1  7              if (JournalingFlag = true) then
01839 I    1  7                  writeln( 'Logging already active')
01840 I    1  8              else begin
01841 I    1  8                  open( Journal, JournalFilename);
01842 I    1  8                  rewrite( Journal);
01843 I    1  8                  JournalingFlag := true;
01844 I    1  7              end;
01845 I    1  7
01846 I    1  7          DeactivateCommand:
01847 I    1  7              if (JournalingFlag = false) then
01848 I    1  7                  writeln( 'Logging not active')
01849 I    1  8              else begin
01850 I    1  8                  close( Journal);
01851 I    1  8                  JournalingFlag := false;
01852 I    1  7              end;
01853 I    1  7
01854 I    1  7          ErrorCommand: ;
01855 I    1  7
01856 I    1  7          otherwise writeln( 'Invalid command');
01857 I    1  7
01858 I    1  6          end; {caseCommand}
01859 I    1  5        end; {LogID command}
```

```
-LINE-IDC-PL-SL-

01861 I    1  5       LoadID: {read load image}
01862 I    1  6         begin
01863 I    1  6           Command := DisplayCommand;
01864 I    1  7           while (EOLNFlag = false) do begin
01865 I    1  7
01866 I    1  7               SyntaxInterpreter( Modifier, SourceLine, SourcePointer,
01867 I    1  7                   SourceLength, EOLNFlag);
01868 I    1  7
01869 I    1  8               case Modifier.Class of
01870 I    1  8
01871 I    1  8               BlankClass: ;
01872 I    1  8
01873 I    1  8
01874 I    1  9               OperatorClass: begin
01875 I    1 10                   case Modifier.ID of
01876 I    1 10
01877 I    1 11                   AssignmentID: begin
01878 I    1 11                       if (EOLNFlag = true) then Command := InvalidNameCommand
01879 I    1 12                       else begin
01880 I    1 12                           ParseFilename( NullName, LoadFilename,
01881 I    1 12                               SourceLine, SourcePointer, SourceLength, EOLNFlag);
01882 I    1 12                           if (NullName = true) then Command := InvalidNameCommand
01883 I    1 13                           else begin
01884 I    1 13                               Exists := FileExists( %descr LoadFileName);
01885 I    1 13                               if (Exists = true) then Command := LoadCommand
01886 I    1 13                               else Command := InvalidNameCommand;
01887 I    1 12                           end;
01888 I    1 12                           EOLNFlag := true;
01889 I    1 11                       end;
01890 I    1 10                   end; {case Assign.ID}
01891 I    1 10
01892 I    1 11                   otherwise begin
01893 I    1 11                       writeln( 'Invalid assignment operator: "', Modifier.Name, '"');
01894 I    1 11                       Command := ErrorCommand;
01895 I    1 11                       EOLNFlag := true;
01896 I    1 10                   end; {otherwise}
01897 I    1 10
01898 I    1  9                   end; {case Modifier.ID}
01899 I    1  9
01900 I    1  8               end; {case Operator Class}
01901 I    1  8
01902 I    1  9               otherwise begin
01903 I    1  9                   writeln( 'Invalid modifier: "', Modifier.Name, '"');
01904 I    1  9                   Command := ErrorCommand;
```

```
01905 I    1  9                    EOLNFlag := true;
01906 I    1  8                end; {otherwise}
01907 I    1  8
01908 I    1  7                end; {case Modifier.Class}
01909 I    1  6            end; {while ( EOLNFlag = false)}
```

```
-LINE-IDC-PL-SL-

01911 I    1  7          case Command of
01912 I    1  7
01913 I    1  7          DisplayCommand:
01914 I    1  7              writeln( LoadFilename);
01915 I    1  7
01916 I    1  8          LoadCommand: begin
01917 I    1  8              StopCode := Normal;
01918 I    1  8              Loader( LoadFilename, DFormat, StopCode, SavedCS, SavedIP);
01919 I    1  8              if (StopCode <> Normal) then ReportBadStopCode( StopCode,
01920 I    1  8                  Execute, SavedCS, SavedIP);
01921 I    1  7          end;
01922 I    1  7
01923 I    1  7          InvalidNameCommand:
01924 I    1  7              writeln( 'Invalid file or path name');
01925 I    1  7
01926 I    1  7          ErrorCommand: ;
01927 I    1  7
01928 I    1  7          otherwise writeln( 'Invalid command');
01929 I    1  7
01930 I    1  6          end; {caseCommand}
01931 I    1  5        end; {LoadID command}
01932 I    1  4      end; {Keyword.ID Command}
01933 I    1  3  end; {UtilityClass}
```

```
-LINE-IDC-PL-SL-

01935       1  3   %include 'InputOutputClass.subpas/list'
01936 I     1  4   InputOutputClass: begin {set / display IOPort Data}
01937 I     1  4     Format := Keyword;
01938 I     1  5     if (EOLNFlag = false) then begin
01939 I     1  5       GetPortAddress( PortAddress, SourceLine, SourcePointer,
01940 I     1  5           SourceLength, EOLNFlag, NullAddress);
01941 I     1  5       PortCount := OnePort;
01942 I     1  5       Command := DisplayCommand;
01943 I     1  5
01944 I     1  6       if (NullAddress = true) then begin
01945 I     1  6           if (EOLNFlag = true) then writeln( 'Invalid input - address required');
01946 I     1  6           PortAddressPresent := true; {dummied out - let EOLNFlag handle recovery}
01947 I     1  6           PortIndex := FirstIOPortIndex; {also dummied out}
01948 I     1  6           EOLNFlag := true;
01949 I     1  6           Command := ErrorCommand;
01950 I     1  6       end
01951 I     1  5       else SearchForPortAddress( PortAddress, IOPortMap, PortAddressPresent,      PortIndex);
01952 I     1  5       if (PortAddressPresent = true) then with IOPortMap[PortIndex] do
01953 I     1  6         while (EOLNFlag = false) do begin
01954 I     1  6
01955 I     1  6           SyntaxInterpreter( Modifier, SourceLine, SourcePointer, SourceLength, EOLNFlag);
01956 I     1  7           case Modifier.Class of
01957 I     1  7
01958 I     1  7           BlankClass: ;
01959 I     1  7
01960 I     1  8           QualifierClass: begin
01961 I     1  9               case Modifier.ID of
01962 I     1  9
01963 I     1 10               LengthID: begin
01964 I     1 10                   SyntaxInterpreter( LengthInput, SourceLine, SourcePointer,
01965 I     1 10                       SourceLength, EOLNFlag);
01966 I     1 10                   PortCount := int( LengthInput.Value);
01967 I     1  9               end; {case LengthID}
01968 I     1  9
01969 I     1 10               ToID: begin
01970 I     1 10                   GetPortAddress( FinalPortAddress, SourceLine,
01971 I     1 10                       SourcePointer, SourceLength, EOLNFlag, NullAddress);
01972 I     1 11                   if (NullAddress = true) then begin
01973 I     1 11                       writeln( 'Invalid input - address required');
01974 I     1 11                       Command := ErrorCommand;
01975 I     1 11                   end
01976 I     1 10                   else PortCount := NoPorts;
01977 I     1  9               end; {case ToID}
01978 I     1  9
```

```
01979 I     1 10              otherwise begin
01980 I     1 10                  writeln( 'Invalid qualifier: "', Modifier.Name, '"');
01981 I     1 10                  Command := ErrorCommand;
01982 I     1 10                  EOLNFlag := true;
01983 I     1  9              end;
01984 I     1  9
01985 I     1  8              end; {case Modifier.ID}
01986 I     1  7          end; {case Qualifier Class}
```

```
-LINE-IDC-PL-SL-

01988 I     1  8          OperatorClass: begin
01989 I     1  9              case Modifier.ID of
01990 I     1  9
01991 I     1 10              AssignmentID: begin
01992 I     1 10                  GetData( DataArray, DataArrayEntries, SourceLine,
01993 I     1 10                      SourcePointer, SourceLength, EOLNFlag, Format.ID);
01994 I     1 10                  Command := AssignmentCommand;
01995 I     1  9              end; {case Assign.ID}
01996 I     1  9
01997 I     1 10              otherwise begin
01998 I     1 10                  writeln( 'Invalid assignment operator: "', Modifier.Name, '"');
01999 I     1 10                  Command := ErrorCommand;
02000 I     1 10                  EOLNFlag := true;
02001 I     1  9              end; {otherwise}
02002 I     1  9
02003 I     1  8              end; {case Modifier.ID}
02004 I     1  8
02005 I     1  7          end; {case Operator Class}
02006 I     1  7
02007 I     1  8          otherwise begin
02008 I     1  8              writeln( 'Invalid modifier: "', Modifier.Name, '"');
02009 I     1  8              Command := ErrorCommand;
02010 I     1  8              EOLNFlag := true;
02011 I     1  7          end; {otherwise}
02012 I     1  7
02013 I     1  6          end; {case Modifier.Class}
02014 I     1  6      end {while ( EOLNFlag = false)}
02015 I     1  6
02016 I     1  6      else begin
02017 I     1  6          Command := ErrorCommand;
02018 I     1  6          writeln( 'Specified port address  , hex( PortAddress, 4, 4),   not found in map');
02019 I     1  5      end;
```

```
-LINE-IDC-PL-SL-

02021 I     1  6       case Command of
02022 I     1  6
02023 I     1  7       DisplayCommand: begin
02024 I     1  7
02025 I     1  7           IODone := false;
02026 I     1  7
02027 I     1  8           while (not IODone) do with IOPortMap[ PortIndex] do begin
02028 I     1  8               if (InOutIndicator = OutputIndicator) then
02029 I     1  8                   TempDataIndex := DataIndex - 1
02030 I     1  8               else TempDataIndex := DataIndex;
02031 I     1  8
02032 I     1  9               case Format.Width of
02033 I     1  9
02034 I     1 10               EightBits: begin
02035 I     1 10                   write( 'Port address:  , hex( Address, 4, 4), ': ');
02036 I     1 10                   if (TempDataIndex < FirstIODataPoint) then DisplayData := 0
02037 I     1 10                   else Get8BitsofData( PortIndex, TempDataIndex, DisplayData);
02038 I     1 10                   write( hex( DisplayData, 2, 2));
02039 I     1  9               end; {Eight bits]
02040 I     1  9
02041 I     1 10               SixteenBits: if (Format.Width = PortWidth) then begin
02042 I     1 11                 if (LowerUpperIndicator = LowerPortIndicator) then begin
02043 I     1 11                   write( 'Port address:  , hex( Address, 4, 4));
02044 I     1 11                   write(   / ', hex( IOPortMap[ AuxIndex].Address, 4, 4), ': ');
02045 I     1 11                   if (TempDataIndex < FirstIODataPoint) then DisplayData := 0
02046 I     1 11                   else Get8BitsofData( AuxIndex, TempDataIndex, DisplayData);
02047 I     1 11                   write( hex( DisplayData, 2, 2));
02048 I     1 11                   if (TempDataIndex < FirstIODataPoint) then DisplayData := 0
02049 I     1 11                   else Get8BitsofData( PortIndex, TempDataIndex, DisplayData);
02050 I     1 11                   write( hex( DisplayData, 2, 2));
02051 I     1 11                 end
02052 I     1 11                 else begin
02053 I     1 11                   write( 'Port address:  , hex( IOPortMap[ AuxIndex].Address, 4, 4));
02054 I     1 11                   write(   / ', hex( Address, 4, 4), ': ');
02055 I     1 11                   if (TempDataIndex < FirstIODataPoint) then DisplayData := 0
02056 I     1 11                   else Get8BitsofData( PortIndex, TempDataIndex, DisplayData);
02057 I     1 11                   write( hex( DisplayData, 2, 2));
02058 I     1 11                   if (TempDataIndex < FirstIODataPoint) then DisplayData := 0
02059 I     1 11                   else Get8BitsofData( AuxIndex, TempDataIndex, DisplayData);
02060 I     1 11                   write( hex( DisplayData, 2, 2));
02061 I     1 10                 end;
02062 I     1 10               end {PortWidth = Format.Width]
02063 I     1 10               else begin
02064 I     1 10                   write( 'Attempting to access an 8 bit wide port ');
```

```
02065 I    1 10                 write( 'with WPORT is not valid');
02066 I    1  9               end; {case SixteenBits}
02067 I    1  8               end; {case Format.Width}
02068 I    1  8
02069 I    1  8               writeln;
02070 I    1  8               FindNextPort( PortIndex, PortCount, FinalPortAddress, IODone);
02071 I    1  8
02072 I    1  7         end; {while not IODone}
02073 I    1  6       end; {Display Command}
```

```
-LINE-IDC-PL-SL-

02075 I    1  7       AssignmentCommand: begin
02076 I    1  7           IODone := false;
02077 I    1  7           DataArrayPointer := StartOfData;
02078 I    1  8           while (not IODone) do with IOPortMap[ PortIndex] do begin
02079 I    1  8             GetAssignData( AssignData, DataArrayPointer, DataArray, DataArrayEntries);
02080 I    1  8             if (InOutIndicator = OutputIndicator) then
02081 I    1  8               TempDataIndex := DataIndex - 1
02082 I    1  8             else TempDataIndex := DataIndex;
02083 I    1  8
02084 I    1  9             case Format.Width of
02085 I    1  9
02086 I    1 10             EightBits: begin
02087 I    1 10               AssignDataLow := uand( AssignData, Low8Bits);
02088 I    1 10               if (TempDataIndex >= FirstIODataPoint) then
02089 I    1 10                 Put8BitsofData( PortIndex, TempDataIndex, AssignDataLow);
02090 I    1  9               end; {case EightBits}
02091 I    1  9
02092 I    1 10             SixteenBits: if (Format.Width = PortWidth) then begin
02093 I    1 10
02094 I    1 11               if (LowerUpperIndicator = LowerPortIndicator) then begin
02095 I    1 11                 AssignDataLow := uand( AssignData, Low8Bits);
02096 I    1 11                 if (TempDataIndex >= FirstIODataPoint) then
02097 I    1 11                   Put8BitsofData( PortIndex, TempDataIndex, AssignDataLow);
02098 I    1 11                 AssignDataHigh := MaskShiftRight( AssignData, High8Bits, Bit8Divisor);
02099 I    1 11                 if (TempDataIndex >= FirstIODataPoint) then
02100 I    1 11                   Put8BitsofData( AuxIndex, TempDataIndex, AssignDataHigh);
02101 I    1 11                 end
02102 I    1 11
02103 I    1 11               else begin
02104 I    1 11                 AssignDataLow := uand( AssignData, Low8Bits);
02105 I    1 11                 if (TempDataIndex >= FirstIODataPoint) then
02106 I    1 11                   Put8BitsofData( AuxIndex, TempDataIndex, AssignDataLow);
02107 I    1 11                 AssignDataHigh := MaskShiftRight( AssignData, High8Bits, Bit8Divisor);
02108 I    1 11                 if (TempDataIndex >= FirstIODataPoint) then
02109 I    1 11                   Put8BitsofData( PortIndex, TempDataIndex, AssignDataHigh);
02110 I    1 10               end; {else begin}
02111 I    1 10               end {PortWidth = Format.Width}
02112 I    1 10               else begin
02113 I    1 10                   write( 'Attempting to access an 8 bit wide port ');
02114 I    1 10                   write( 'with WPORT is not valid');
02115 I    1 10                   writeln;
02116 I    1  9               end; {case SixteenBits}
02117 I    1  9
02118 I    1  8             end; {case Format.Width]
```

```
02119 I    1  8
02120 I    1  8              FindNextPort( PortIndex, PortCount, FinalPortAddress, IODone);
02121 I    1  7            end; {while not IODone}
02122 I    1  6        end; {Assignment Command}
02123 I    1  6
02124 I    1  6        ErrorCommand: ;
02125 I    1  6
02126 I    1  6        otherwise writeln( 'Invalid IO command');
02127 I    1  6
02128 I    1  5        end; {case Command}
02129 I    1  5      end {if EOLNFlag = false}
```

```
-LINE-IDC-PL-SL-

02130 I    1  4     else writeln( 'Command is missing a Port Address');
02131 I    1  3   end; {'I'}
```

```
-LINE-IDC-PL-SL-

02133       1  3   %include 'BreakpointClass.subpas/list'
02134 I     1  4   BreakpointClass: begin {set, clear, display breakpoints}
02135 I     1  4       Command := DisplayCommand;
02136 I     1  5       while (EOLNFlag = false) do begin
02137 I     1  5
02138 I     1  5           SyntaxInterpreter( Modifier, SourceLine, SourcePointer,
02139 I     1  5               SourceLength, EOLNFlag);
02140 I     1  5
02141 I     1  6           case Modifier.Class of
02142 I     1  6
02143 I     1  6           BlankClass: ;
02144 I     1  6
02145 I     1  6
02146 I     1  7           OperatorClass: begin
02147 I     1  8               case Modifier.ID of
02148 I     1  8
02149 I     1  9               AssignmentID: begin
02150 I     1  9                   if (EOLNFlag = true) then Command := DeactivateCommand
02151 I     1 10                   else begin
02152 I     1 10                       GetAddress( BreakSegment, BreakAddress, SourceLine,
02153 I     1 10                           SourcePointer, SourceLength, EOLNFlag, NullAddress);
02154 I     1 10                       if (NullAddress = true) then Command := DeactivateCommand
02155 I     1 10                       else Command := AssignmentCommand;
02156 I     1 10                       EOLNFlag := true;
02157 I     1  9                   end;
02158 I     1  8               end; {case Assign.ID}
02159 I     1  8
02160 I     1  9               otherwise begin
02161 I     1  9                   writeln( 'Invalid assignment operator: "', Modifier.Name, '"');
02162 I     1  9                   Command := ErrorCommand;
02163 I     1  9                   EOLNFlag := true;
02164 I     1  8               end; {otherwise}
02165 I     1  8
02166 I     1  7               end; {case Modifier.ID}
02167 I     1  7
02168 I     1  6           end; {case Operator Class}
02169 I     1  6
02170 I     1  7           otherwise begin
02171 I     1  7               writeln( 'Invalid modifier: "', Modifier.Name, '"');
02172 I     1  7               Command := ErrorCommand;
02173 I     1  7               EOLNFlag := true;
02174 I     1  6           end; {otherwise}
02175 I     1  6
02176 I     1  5           end; {case Modifier.Class}
```

```
02177 I     1  4        end; {while ( EOLNFlag = false)}
```

```
-LINE-IDC-PL-SL-

02179 I     1  5        case Command of
02180 I     1  5
02181 I     1  6        DisplayCommand: begin
02182 I     1  7            case Keyword.ID of
02183 I     1  7
02184 I C   1  7            [ FirstBreakpoint..LastBreakpoint: ]
02185 I     1  7            0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15:
02186 I     1  7                DisplayBreakpoint( Keyword.ID, BreakPointTable[ Keyword.ID]);
02187 I     1  7
02188 I     1  7            AllBPIDs:
02189 I     1  7                for BreakpointLoop := FirstBreakpoint to LastBreakpoint do
02190 I     1  7                    DisplayBreakpoint( BreakpointLoop, BreakPointTable[ BreakpointLoop]);
02191 I     1  7
02192 I     1  7            otherwise writeln( 'Invalid Breakpoint identifier');
02193 I     1  7
02194 I     1  6            end; {case Keyword.ID]
02195 I     1  5        end; {DisplayCommand]
02196 I     1  5
02197 I     1  6        AssignmentCommand: begin
02198 I     1  7            case Keyword.ID of
02199 I     1  7
02200 I C   1  7            [ FirstBreakpoint..LastBreakpoint: ]
02201 I     1  7            0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15:
02202 I     1  8                with BreakPointTable[ Keyword.ID] do begin
02203 I     1  8                    Segment := BreakSegment;
02204 I     1  8                    Address := BreakAddress;
02205 I     1  8                    PhysicalAddress :=
02206 I     1  8                        (( BreakSegment * SegmentOffset) + BreakAddress);
02207 I     1  8                    if ((PhysicalAddress >= (LowMemoryLimit * BytesPerWord)) and
02208 I     1  9                        (PhysicalAddress <= (HighMemoryLimit * BytesPerWord))) then begin
02209 I     1  9                        Activated := Activate;
02210 I     1  9                        Encountered := Deactivate;
02211 I     1  9                    end
02212 I     1  8                    else writeln( 'Invalid memory address');
02213 I     1  7                end; {individual breakpoints]
02214 I     1  7
02215 I     1  7            otherwise writeln( 'Invalid assignment');
02216 I     1  7
02217 I     1  6            end; {case Keyword.ID]
02218 I     1  5        end; {AssignmentCommand]
```

```
-LINE-IDC-PL-SL-

02220 I    1  6       DeactivateCommand: begin
02221 I    1  7           case Keyword.ID of
02222 I    1  7
02223 I C  1  7           { FirstBreakpoint..LastBreakpoint: }
02224 I    1  7           0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15:
02225 I    1  8               with BreakPointTable[ Keyword.ID] do begin
02226 I    1  8                   Segment := NullSegmentValue;
02227 I    1  8                   Address := NullAddressValue;
02228 I    1  8                   PhysicalAddress := NullAddressValue;
02229 I    1  8                   Encountered := Deactivate;
02230 I    1  8                   Activated := Deactivate;
02231 I    1  7               end; {individual breakpoints}
02232 I    1  7
02233 I    1  7           AllBPIDs:
02234 I    1  7               for BreakpointLoop := FirstBreakpoint to LastBreakpoint do
02235 I    1  8               with BreakPointTable[ BreakpointLoop] do begin
02236 I    1  8                   Segment := NullSegmentValue;
02237 I    1  8                   Address := NullAddressValue;
02238 I    1  8                   PhysicalAddress := NullAddressValue;
02239 I    1  8                   Encountered := Deactivate;
02240 I    1  8                   Activated := Deactivate;
02241 I    1  7               end; {individual breakpoints}
02242 I    1  7
02243 I    1  7           otherwise writeln( 'Invalid Breakpoint identifier');
02244 I    1  7
02245 I    1  6           end; {case Keyword.ID}
02246 I    1  5       end; {DeactivateCommand}
02247 I    1  5
02248 I    1  5       ErrorCommand: ;
02249 I    1  5
02250 I    1  5       otherwise writeln( 'Invalid command');
02251 I    1  5
02252 I    1  4       end; {caseCommand}
02253 I    1  3   end; {BreakpointClass}
```

```
-LINE-IDC-PL-SL-

02255       1  3   %include 'ExecuteClass.subpas/list'
02256 I     1  4   ExecuteClass: begin {execute program segment}
02257 I     1  5       case Keyword.ID of
02258 I     1  5
02259 I     1  6       GOID: begin
02260 I     1  6           Command := GOCommand;
02261 I     1  7           while (EOLNFlag = false) do begin
02262 I     1  7               SyntaxInterpreter( Modifier, SourceLine, SourcePointer,
02263 I     1  7                   SourceLength, EOLNFlag);
02264 I     1  8               case Modifier.Class of
02265 I     1  8
02266 I     1  8               BlankClass: ;
02267 I     1  8
02268 I     1  9               OperatorClass: begin
02269 I     1 10                   case Modifier.ID of
02270 I     1 10
02271 I     1 11                   AssignmentID: begin
02272 I     1 11                       GetAddress( NewCSValue, NewIPValue, SourceLine,
02273 I     1 11                           SourcePointer, SourceLength, EOLNFlag, NullAddress);
02274 I     1 12                       if (NullAddress = true) then begin
02275 I     1 12                           writeln( 'Invalid address');
02276 I     1 12                           Command := ErrorCommand;
02277 I     1 12                           EOLNFlag := true;
02278 I     1 12                       end
02279 I     1 12                       else begin
02280 I     1 12                           IP := NewIPValue;
02281 I     1 12                           StoreRegPtr( CSWidth, CSID, NewCSValue, StopCode);
02282 I     1 12                           EOLNFlag := true;
02283 I     1 11                       end;
02284 I     1 10                   end; {AssignementID}
02285 I     1 10
02286 I     1 11                   otherwise begin
02287 I     1 11                       writeln( 'Invalid modifier ID', Modifier.ID:1,
02288 I     1 11                           '"', Modifier.Name, '"');
02289 I     1 11                       Command := ErrorCommand;
02290 I     1 11                       EOLNFlag := true;
02291 I     1 10                   end; {otherwise}
02292 I     1 10
02293 I     1  9                   end; {case Modifier.ID}
02294 I     1  9
02295 I     1  8               end; {case OperatorClass}
02296 I     1  8
02297 I     1  9               otherwise begin
02298 I     1  9                   writeln( 'Invalid modifier: "', Modifier.Name, '"');
```

```
02299 I    1  9                    Command := ErrorCommand;
02300 I    1  9                    EOLNFlag := true;
02301 I    1  8                end; {otherwise}
02302 I    1  8
02303 I    1  7                end; {case Modifier.Class}
02304 I    1  6            end; {while EOLNFlag}
02305 I    1  5        end; {case GOID}
```

```
-LINE-IDC-PL-SL-

02307 I    1  5       ContinueID: Command := ContinueCommand;
02308 I    1  5
02309 I    1  5       StepID: Command := StepCommand;
02310 I    1  5
02311 I    1  5       PStepID: Command := PStepCommand;
02312 I    1  5
02313 I    1  6       otherwise begin
02314 I    1  6           writeln( 'Invalid execution command');
02315 I    1  6           Command := ErrorCommand;
02316 I    1  6           EOLNFlag := true;
02317 I    1  5       end; {otherwise Keyword.ID}
02318 I    1  5
02319 I    1  4       end; {case Keyword.ID}
```

```
-LINE-IDC-PL-SL-

02321 I    1  5       case Command of
02322 I    1  5
02323 I    1  6       GoCommand, ContinueCommand: begin
02324 I    1  6           if (Command = GoCommand) then PStepCount := 0;
02325 I    1  6           Execute := GoContinue;
02326 I    1  7           while (Execute = GoContinue) do begin
02327 I    1  7               ExecuteStopCode := Normal;
02328 I    1  7               CtrlDStopCode := Normal;
02329 I    1  7               InstallBreakpoints( BreakPointTable, ExecuteStopCode);
02330 I    1  7               FetchRegPtr( CSWidth, CSId, SavedCS, ExecuteStopCode);
02331 I    1  7               SavedIP := IP;
02332 I    1  7               if (ExecuteStopCode <> Normal) then Execute := Stop
02333 I    1  8               else begin
02334 I    1  8                   ExecuteInstruction( StopCode);
02335 I    1  9                   case StopCode of
02336 I    1  9
02337 I    1  9                   Call, Normal, Return: ;
02338 I    1  9
02339 I    1 10                   Breakpoint: begin
02340 I    1 10                       SearchForBreakpoint( BreakPointTable, BreakPointEntry,
02341 I    1 10                           BreakPointFound, SavedCS, SavedIP);
02342 I    1 11                       if (BreakPointFound = true) then begin
02343 I    1 11                           GoodBreakpoint( BreakPointTable, BreakPointEntry,
02344 I    1 11                               Execute, SavedCS, SavedIP);
02345 I    1 11                           IP := SavedIP;
02346 I    1 11                           end
02347 I    1 10                       else UnexpectedBreakpoint( Execute, SavedCS, SavedIP);
02348 I    1  9                   end; {case Breakpoint}
02349 I    1  9
02350 I    1  9                   Halt: ProgramHalt( Execute, SavedCS, SavedIP);
02351 I    1  9
02352 I    1  9                   otherwise ReportBadStopcode( StopCode, Execute, SavedCS, SavedIP);
02353 I    1  9
02354 I    1  8                   end; {case StopCode}
02355 I    1  8                   RemoveBreakpoints( BreakPointTable, StopCode);
02356 I    1  8                   if (CtrlDStopCode <> Normal) then
02357 I    1  8                       ReportBadStopCode( ControlD, Execute, SavedCS, SavedIP);
02358 I    1  7               end;
02359 I    1  6           end; {while Execute = GoContinue}
02360 I    1  5       end; {case GoCommand, ContinueCommand}
```

```
-LINE-IDC-PL-SL-

02362 I    1  6       StepCommand: begin
02363 I    1  6
02364 I    1  6           Execute := GoContinue;
02365 I    1  7           while (Execute = GoContinue) do begin
02366 I    1  7               ExecuteStopCode := Normal;
02367 I    1  7               CtrlDStopCode := Normal;
02368 I    1  7               FetchRegPtr( CSWidth, CSId, SavedCS, ExecuteStopCode);
02369 I    1  7               SavedIP := IP;
02370 I    1  7               if (ExecuteStopCode <> Normal) then Execute := Stop
02371 I    1  8               else begin
02372 I    1  8                   write( 'This instruction at ');
02373 I    1  8                   write( hex( SavedCS, 4, 4),  :');
02374 I    1  8                   write( hex( SavedIP, 4, 4));
02375 I    1  8                   writeln;
02376 I    1  8
02377 I    1  8                   ExecuteInstruction( StopCode);
02378 I    1  8
02379 I    1  9                   case StopCode of
02380 I    1  9
02381 I    1  9                   Call, Normal, Return: Execute := Stop;
02382 I    1  9
02383 I    1  9                   Breakpoint: UnexpectedBreakpoint( Execute, SavedCS, SavedIP);
02384 I    1  9
02385 I    1  9                   Halt: ProgramHalt( Execute, SavedCS, SavedIP);
02386 I    1  9
02387 I    1  9                   otherwise ReportBadStopcode( StopCode, Execute, SavedCS, SavedIP);
02388 I    1  9
02389 I    1  8                   end; {case StopCode}
02390 I    1  8
02391 I    1  8                   if (CtrlDStopCode <> Normal) then
02392 I    1  8                       ReportBadStopCode( ControlD, Execute, SavedCS, SavedIP);
02393 I    1  8
02394 I    1  7               end; {ExecuteStopCode <> Normal}
02395 I    1  6           end; {while Execute = GoContinue}
02396 I    1  5       end; {Step command}
```

```
-LINE-IDC-PL-SL-

02398 I    1  6       PStepCommand: begin
02399 I    1  7           if (PStepCount <> 0) then begin
02400 I    1  7               write( 'Reset PStep count ?  );
02401 I    1  7               readln( Response);
02402 I    1  7               if (Response in Yes) then PStepCount := 0;
02403 I    1  6           end;
02404 I    1  6           Execute := GoContinue;
02405 I    1  6
02406 I    1  7           while (Execute = GoContinue) do begin
02407 I    1  7               ExecuteStopCode := Normal;
02408 I    1  7               CtrlDStopCode := Normal;
02409 I    1  7               InstallBreakpoints( BreakPointTable, ExecuteStopCode);
02410 I    1  7               FetchRegPtr( CSWidth, CSID, SavedCS, ExecuteStopCode);
02411 I    1  7               SavedIP := IP;
02412 I    1  7
02413 I    1  7               if (ExecuteStopCode <> Normal) then Execute := Stop
02414 I    1  8               else begin
02415 I    1  9                   if (PStepCount = 0) then begin
02416 I    1  9                       write( 'This instruction at  );
02417 I    1  9                       write( hex( SavedCS, 4, 4),  :');
02418 I    1  9                       write( hex( SavedIP, 4, 4));
02419 I    1  9                       writeln;
02420 I    1  8                   end;
02421 I    1  8                   ExecuteInstruction( StopCode);
02422 I    1  8
02423 I    1  9                   case StopCode of
02424 I    1  9
02425 I    1  9                   Normal: ;
02426 I    1  9
02427 I    1  9                   Call: PStepCount := PStepCount + 1;
02428 I    1  9
02429 I    1  9                   Return: PStepCount := PStepCount - 1;
02430 I    1  9
02431 I    1 10                   Breakpoint: begin
02432 I    1 10                       SearchForBreakpoint( BreakPointTable, BreakPointEntry,
02433 I    1 10                           BreakPointFound, SavedCS, SavedIP);
02434 I    1 11                       if (BreakPointFound = true) then begin
02435 I    1 11                           GoodBreakpoint( BreakPointTable, BreakPointEntry,
02436 I    1 11                               Execute, SavedCS, SavedIP);
02437 I    1 11                           IP := SavedIP;
02438 I    1 11                       end
02439 I    1 10                       else UnexpectedBreakpoint( Execute, SavedCS, SavedIP);
02440 I    1  9                   end; {case Breakpoint}
```

```
-LINE-IDC-PL-SL-

02442 I     1  9                  Halt: ProgramHalt( Execute, SavedCS, SavedIP);
02443 I     1  9
02444 I     1  9                  otherwise ReportBadStopcode( StopCode, Execute, SavedCS, SavedIP);
02445 I     1  9
02446 I     1  8                  end; {case StopCode}
02447 I     1  8                  RemoveBreakpoints( BreakPointTable, StopCode);
02448 I     1  8                  if (CtrlDStopCode <> Normal) then
02449 I     1  8                      ReportBadStopCode( ControlD, Execute, SavedCS, SavedIP);
02450 I     1  7              end; {else ExecuteStopCode <> Normal}
02451 I     1  7              if (PStepCount = 0) then Execute := Stop;
02452 I     1  6          end; {while Execute = GoContinue}
02453 I     1  6
02454 I     1  5      end; {PStep command}
02455 I     1  5
02456 I     1  5      ErrorCommand: ;
02457 I     1  5
02458 I     1  5      otherwise writeln( 'Invalid Execution command');
02459 I     1  5
02460 I     1  4      end; {case Command}
02461 I     1  4
02462 I     1  3  end; {Execute 'X'}
02463       1  3
02464       1  3
02465       1  3
02466       1  3
02467       1  3  otherwise writeln( 'Bad keyword class: "', Keyword.Class, '"');
02468       1  2  end; {Keyword.Class}
02469       1  1    end;
02470       0  0 end; {Debugger procedure}
02471       0  0 end. {Debugger Module}
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	24506	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	3426	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

ENVIRONMENT STATISTICS

File	Symbols Total	Symbols Loaded	Symbols Percent
SYS$COMMON:[SYSLIB]STARLET.PEN;4	20797	48	0

COMMAND QUALIFIERS

```
  PAS/LIS DEBUGGER.PAS

  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
  /OPTIMIZE
  /STANDARD=NONE
  /TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
  /USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
  /NOANALYSIS_DATA
  /NOENVIRONMENT
  /LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]DEBUGGER.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]DEBUGGER.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	421	00:00.5	00:01.4
Source Analysis	1415	00:11.5	00:32.0
Source Listing	15	00:03.5	00:12.6
Tree Construction	474	00:02.6	00:07.3
Flow Analysis	220	00:02.3	00:04.4
Value Propagation	13	00:00.3	00:00.6
Profit Analysis	133	00:01.4	00:03.7
Context Analysis	282	00:14.0	00:28.6
Name Packing	10	00:00.5	00:01.1

Compilation Complete

```
-LINE-IDC-PL-SL-

00001    C  0  0 { Title: DisplayBreakpoint
00002    C  0  0
00003    C  0  0   Purpose: Display segment, address and status information
00004    C  0  0            for the specified Breakpoint
00005    C  0  0
00006    C  0  0   Author: William A. Chapman      Date: September 18, 1986
00007    C  0  0
00008    C  0  0   Inputs: BreakpointID
00009    C  0  0            BreakpointEntry - BreakRecord
00010    C  0  0
00011    C  0  0   Outputs: Information is displayed
00012    C  0  0
00013    C  0  0   Procedures Invoked: none
00014    C  0  0 }
00015       0  0 module DisplayBreakpoint( input, output);
00016       0  0
00017       0  0 const
00018       0  0     %include '[-]Const.defn/list'
00019 I     0  0     FirstOpcodeValue = 0;
00020 I     0  0     LastOpcodeValue = 255;
00021 I     0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00022 I     0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00023 I     0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00024 I     0  0     Bit8Multiplier = %X'100';
00025 I     0  0     Bit8Divisor    = %X'100';
00026 I     0  0     WordMultiplier = %X'100';
00027 I     0  0
00028 I     0  0     EightBits = 0;                       {width for 8 bits}
00029 I     0  0     SixteenBits = 1;                     {width for 16 bits}
00030 I     0  0
00031 I     0  0     SimpleMode = %B'0';
00032 I     0  0     SimpleRM = %B'110';
00033 I     0  0
00034 I     0  0     LowMemoryLimit = 0;                  {low limit on memory array}
00035 I     0  0     HighMemoryLimit = 2048;              {high limit on memory array}
00036 I     0  0
00037 I     0  0     FilenameLength = 40;
```

```
-LINE-IDC-PL-SL-

00039       0  0     %include '[-]DebugConst.defn/list'
00040 I     0  0     FirstKeywordEntry = 0;
00041 I     0  0     LastKeywordEntry  = 100;
00042 I     0  0     UndefinedName = 'Qqqqqqqq';
00043 I     0  0     UndefinedClass = 'z';
00044 I     0  0     UndefinedWidth = -1;
00045 I     0  0     UndefinedID = -2;
00046 I     0  0     PageSize = 55;
00047 I     0  0     FF = 12;
00048 I     0  0     MinID = -1;
00049 I     0  0     MaxID = 25;
00050 I     0  0     MinWidth = 0;
00051 I     0  0     MaxWidth = 3;
00052 I     0  0     SegmentOffset = 16;
00053 I     0  0
00054 I     0  0     BlankClass       = ' ';
00055 I     0  0     BreakpointClass  = 'B';
00056 I     0  0     ErrorClass       = 'E';
00057 I     0  0     FlagClass        = 'F';
00058 I     0  0     InputOutputClass = 'I';
00059 I     0  0     MemoryClass      = 'M';
00060 I     0  0     NumericClass     = 'N';
00061 I     0  0     OperatorClass    = 'O';
00062 I     0  0     QualifierClass   = 'Q';
00063 I     0  0     RegisterClass    = 'R';
00064 I     0  0     StackClass       = 'S';
00065 I     0  0     TrueFalseClass   = 'T';
00066 I     0  0     UtilityClass     = 'U';
00067 I     0  0     ExecuteClass     = 'X';
00068 I     0  0
00069 I     0  0     Blank  =   ';
00070 I     0  0     Elipse = '.';
00071 I     0  0
00072 I     0  0     StartOfData = 1;
00073 I     0  0     DataArraySize = 50;
00074 I     0  0
00075 I     0  0     EndofLine = 0;
00076 I     0  0     StartofLine = 1;
00077 I     0  0
00078 I     0  0     ShortStringLength = 8;
00079 I     0  0     InputLineLength = 120;
00080 I     0  0
00081 I     0  0     BooleanID  = 1;
00082 I     0  0     ByteID     = 2;
```

```
00083 I     0   0      IntegerID  = 3;
00084 I     0   0      PointerID  = 4;
00085 I     0   0      SIntegerID = 5;
00086 I     0   0      WordID     = 6;
00087 I     0   0      LengthID   = 3;
00088 I     0   0      ToID       = 5;
00089 I     0   0
00090 I     0   0      AssignmentID = 1;
00091 I     0   0      ColonID      = 2;
00092 I     0   0      PlusID       = 3;
00093 I     0   0      MinusID      = 4;
```

```
-LINE-IDC-PL-SL-

00094 I    0  0     CommaID      = 5;
00095 I    0  0
00096 I    0  0     NullEntries = 0;
00097 I    0  0
00098 I    0  0     AddressState   = 'A';
00099 I    0  0     BooleanState   = 'B';
00100 I    0  0     ErrorState     = 'E';
00101 I    0  0     HexState       = 'H';
00102 I    0  0     IntegerState   = 'I';
00103 I    0  0     SegmentState   = 'S';
00104 I    0  0     SeparatorState = ',';
00105 I    0  0     UnaryState     = '-';
00106 I    0  0     ColonState     = ':';
00107 I    0  0
00108 I    0  0     NegativeOperator = -1;
00109 I    0  0     PositiveOperator = +1;
00110 I    0  0
00111 I    0  0     NullSegmentValue = 0;
00112 I    0  0     NullAddressValue = 0;
00113 I    0  0
00114 I    0  0     FirstBreakpoint  =  0;
00115 I    0  0     LastBreakpoint   = 15;
00116 I    0  0     AllBPIDs         = 16;
00117 I    0  0     Activate         = true;
00118 I    0  0     DeActivate       = false;
00119 I    0  0     BreakpointOpcode = %X'CC';
```

```
-LINE-IDC-PL-SL-

00121       0  0 type
00122       0  0     %include  [-]Type.defn/list'
00123 I     0  0     ZeroOne = -1..1;                        {defines integer with range
00124 I C   0  0                                              of zero to one with an invalid
00125 I C   0  0                                              state of -1}
00126 I     0  0
00127 I     0  0     OpcodeLUTEntry = record                 {defines entries in opcode LUT}
00128 I     0  0   OpcodeClass: char;                        {class D, A, L, S, C, P}
00129 I     0  0   OpcodeKey: integer;                       {key to interpret opcode from
00130 I C   0  0                                              0 to 7}
00131 I     0  0   DirectionBitPresent: boolean;
00132 I     0  0   WidthBitPresent: boolean
00133 I     0  0     end;
00134 I     0  0
00135 I     0  0     OpcodeType = record
00136 I     0  0   Direction: ZeroOne;                      {to or from CPU}
00137 I     0  0   Full: unsigned;                          {full opcode}
00138 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00139 I     0  0     end;
00140 I     0  0
00141 I     0  0     RegisterType = record
00142 I     0  0   Id: integer;                             {identifier}
00143 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00144 I     0  0     end;
00145 I     0  0
00146 I     0  0     EffectiveAddressType = record
00147 I     0  0   Mode: char;                              {register or memory}
00148 I     0  0   Width: ZeroOne;                         {8 or 16 bits}
00149 I     0  0   Address: unsigned;                      {memory address or
00150 I C   0  0                                              register designation}
00151 I     0  0   Segment: integer                        {segment register to use}
00152 I     0  0     end;
00153 I     0  0
00154 I     0  0     NameType = packed array[1..10] of char;
00155 I     0  0
00156 I     0  0     Characters = set of    ..'~';
00157 I     0  0     LettersAndNumbers = set of '0'..'z';
00158 I     0  0     Letters = set of 'A'..'z';
00159 I     0  0     Numbers = set of '0'..'9';
00160 I     0  0
00161 I     0  0     FileName = packed array [1..FilenameLength] of char;
```

```
-LINE-IDC-PL-SL-

00163       0  0     %include  [-]DebugType.defn/list'
00164 I     0  0     NameString = packed array[1..ShortStringLength] of char;
00165 I     0  0
00166 I     0  0     KeywordCharacteristics = record
00167 I     0  0   Name: NameString;
00168 I     0  0   Class: char;
00169 I     0  0   Width: ZeroOne;
00170 I     0  0   ID: integer;
00171 I     0  0   Value: unsigned
00172 I     0  0     end;
00173 I     0  0
00174 I     0  0     InputPointerRange = integer;
00175 I     0  0     InputLine = packed array [1..InputLineLength] of char;
00176 I     0  0
00177 I     0  0     ShortString = packed array [1..ShortStringLength] of char;
00178 I     0  0
00179 I     0  0     DataArrayType = array [StartOfData..DataArraySize] of unsigned;
00180 I     0  0
00181 I     0  0     BreakRecord = record
00182 I     0  0   Encountered: boolean;
00183 I     0  0   Activated: boolean;
00184 I     0  0   Segment: unsigned;
00185 I     0  0   Address: unsigned;
00186 I     0  0   PhysicalAddress: unsigned;
00187 I     0  0   Code: unsigned
00188 I     0  0     end;
00189 I     0  0
00190 I     0  0     BreakPointTableType = array [FirstBreakpoint..LastBreakpoint] of BreakRecord;
00191       0  0
00192       0  0
00193       1  0 [global] procedure DisplayBreakpoint( BreakpointID: integer;
00194       1  0           BreakpointEntry: BreakRecord);
00195       1  0
00196       1  0
00197       1  0
00198       1  1 begin
00199       1  2     if (BreakpointEntry.Activated = Activate) then begin
00200       1  2   write( 'Breakpoint ', hex( BreakpointID, 1, 1),   = ');
00201       1  2   write( hex( BreakpointEntry.Segment, 4, 4), ':');
00202       1  2   write( hex( BreakpointEntry.Address, 4, 4));
00203       1  2   writeln;
00204       1  1     end;
00205       0  0 end;
00206       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	226	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS DISPLAYBREAKPOINT.PAS

  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
  /OPTIMIZE
  /STANDARD=NONE
  /TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
  /USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
  /NOANALYSIS_DATA
  /NOENVIRONMENT
  /LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]DISPLAYBREAKPOINT.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]DISPLAYBREAKPOINT.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	405	00:00.5	00:01.5
Source Analysis	238	00:00.7	00:01.8
Source Listing	17	00:00.4	00:02.5
Tree Construction	79	00:00.0	00:00.1
Flow Analysis	37	00:00.0	00:00.0
Value Propagation	10	00:00.0	00:00.0
Profit Analysis	45	00:00.0	00:00.0
Context Analysis	152	00:00.2	00:00.3
Name Packing	6	00:00.0	00:00.0
Code Selection	68	00:00.1	00:00.2
Final	73	00:00.1	00:00.6
TOTAL	1134	00:02.1	00:07.1

COMPILATION STATISTICS

```
  CPU Time:       00:02.1         (5886 Lines/Minute)
  Elapsed Time:   00:07.1
```

```
-LINE-IDC-PL-SL-

00001    C  0  0 { Title: Emulator86
00002    C  0  0
00003    C  0  0   Purpose: Emulate the 8086 instruction set
00004    C  0  0
00005    C  0  0   Author: William A. Chapman      Date: February 23, 1986
00006    C  0  0
00007    C  0  0   Inputs: The user is asked if any input or output ports are used.
00008    C  0  0           If so, then the map and data file names are requested.
00009    C  0  0
00010    C  0  0   Outputs: As specified by the user, possibilities include :
00011    C  0  0           Output data
00012    C  0  0           Registers
00013    C  0  0           Memory
00014    C  0  0
00015    C  0  0   Procedures Invoked:
00016    C  0  0           Debugger
00017    C  0  0           FileExists.FORTRAN
00018    C  0  0 }
00019       0  0 program Emulator86( input, output, LookUpTable,
00020       0  0           IOMapFile, IODataFile);
00021       0  0
00022       0  0 const
00023       0  0     %include '[-]Const.defn/list'
00024 I     0  0     FirstOpcodeValue = 0;
00025 I     0  0     LastOpcodeValue = 255;
00026 I     0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00027 I     0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00028 I     0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00029 I     0  0     Bit8Multiplier = %X'100';
00030 I     0  0     Bit8Divisor    = %X'100';
00031 I     0  0     WordMultiplier = %X'100';
00032 I     0  0
00033 I     0  0     EightBits = 0;                        {width for 8 bits}
00034 I     0  0     SixteenBits = 1;                      {width for 16 bits}
00035 I     0  0
00036 I     0  0     SimpleMode = %B'0';
00037 I     0  0     SimpleRM = %B'110';
00038 I     0  0
00039 I     0  0     LowMemoryLimit = 0;                   {low limit on memory array}
00040 I     0  0     HighMemoryLimit = 2048;               {high limit on memory array}
00041 I     0  0
00042 I     0  0     FilenameLength = 40;
00043       0  0
00044       0  0     %include '[-]IOConst.defn/list'
00045 I     0  0     KeyboardInPortNumber  = %X'FF';
```

```
00046 I     0   0      CRTOutPortNumber        = %X'FF';
00047 I     0   0
00048 I     0   0      FirstIOPortIndex    = 1;
00049 I     0   0      MaxNumberofIOPorts = 8;
00050 I     0   0
00051 I     0   0      FirstIODataPoint    = 1;
00052 I     0   0      MaxNumberofIOWords = 64;
00053 I     0   0      MaxNumberofIOBytes = 256;     {4 times the number of words}
00054 I     0   0
00055 I     0   0      InputIndicator  = 'I';
```

```
-LINE-IDC-PL-SL-

00056 I     0  0      OutputIndicator = 'O';
00057 I     0  0      IOIndicator     = 'B';
00058 I     0  0
00059 I     0  0      LowerPortIndicator = 'L';
00060 I     0  0      UpperPortIndicator = 'U';
00061 I     0  0
00062 I     0  0      InvalidAddress = - 1;
00063 I     0  0      InvalidLocation = - 1;
00064 I     0  0      InvalidIndex = - 1;
00065       0  0
00066       0  0      %include '[-]Emulator86.version/list'
00067 I     0  0      VersionNo = 1;
00068 I     0  0      ReleaseNo = 4;
00069       0  0
00070       0  0      LineLength = 64;
00071       0  0
00072       0  0      UndefinedSORValue = -100;
00073       0  0
00074       0  0      Standard = 'S';
00075       0  0      HexVersion = 'H';
```

```
-LINE-IDC-PL-SL-

00077       0  0 type
00078       0  0     %include  [-]Type.defn/list'
00079 I     0  0     ZeroOne = -1..1;                        {defines integer with range
00080 I C   0  0                                              of zero to one with an invalid
00081 I C   0  0                                              state of -1}
00082 I     0  0
00083 I     0  0     OpcodeLUTEntry = record                 {defines entries in opcode LUT}
00084 I     0  0   OpcodeClass: char;                        {class D, A, L, S, C, P}
00085 I     0  0   OpcodeKey: integer;                       {key to interpret opcode from
00086 I C   0  0                                              0 to 7}
00087 I     0  0   DirectionBitPresent: boolean;
00088 I     0  0   WidthBitPresent: boolean
00089 I     0  0     end;
00090 I     0  0
00091 I     0  0     OpcodeType = record
00092 I     0  0   Direction: ZeroOne;                      {to or from CPU}
00093 I     0  0   Full: unsigned;                          {full opcode}
00094 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00095 I     0  0     end;
00096 I     0  0
00097 I     0  0     RegisterType = record
00098 I     0  0   Id: integer;                             {identifier}
00099 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00100 I     0  0     end;
00101 I     0  0
00102 I     0  0     EffectiveAddressType = record
00103 I     0  0   Mode: char;                              {register or memory}
00104 I     0  0   Width: ZeroOne;                         {8 or 16 bits}
00105 I     0  0   Address: unsigned;                      {memory address or
00106 I C   0  0                                              register designation}
00107 I     0  0   Segment: integer                        {segment register to use}
00108 I     0  0     end;
00109 I     0  0
00110 I     0  0     NameType = packed array[1..10] of char;
00111 I     0  0
00112 I     0  0     Characters = set of    ..'~';
00113 I     0  0     LettersAndNumbers = set of '0'..'z';
00114 I     0  0     Letters = set of 'A'..'z';
00115 I     0  0     Numbers = set of '0'..'9';
00116 I     0  0
00117 I     0  0     FileName = packed array [1..FilenameLength] of char;
00118       0  0
00119       0  0     %include '[-]FlagType.defn/list'
00120 I     0  0     FlagType = record                       {define processor flags}
```

```
00121 I    0  0   Carry: boolean;
00122 I    0  0   Parity: boolean;
00123 I    0  0   AuxCarry: boolean;
00124 I    0  0   Zero: boolean;
00125 I    0  0   Sign: boolean;
00126 I    0  0   Trap: boolean;
00127 I    0  0   Interrupt: boolean;
00128 I    0  0   Direction: boolean;
00129 I    0  0   Overflow: boolean
00130 I    0  0     end;
00131      0  0
```

```
-LINE-IDC-PL-SL-

00132       0  0     %include  [-]MemoryType.defn/list'
00133 I     0  0     MemoryArray = array[LowMemoryLimit..HighMemoryLimit] of unsigned;
00134       0  0
00135       0  0     %include '[-]IOType.defn/list'
00136 I     0  0     PortEntry = record
00137 I     0  0   Address: unsigned;
00138 I     0  0   AuxIndex: integer;
00139 I     0  0   DataIndex: integer;
00140 I     0  0   InputIndex: integer;
00141 I     0  0   PortWidth: integer;
00142 I     0  0   LowerUpperIndicator: char;
00143 I     0  0   InOutIndicator: char
00144 I     0  0     end;
00145 I     0  0
00146 I     0  0     PortMap = array[ FirstIOPortIndex..MaxNumberOfIOPorts] of PortEntry;
00147       0  0
00148       0  0     TwoByteString = packed array[1..2] of char;
```

```
-LINE-IDC-PL-SL-

00150       0  0 var
00151       0  0     %include  [-]VarGlobal.defn/list'
00152 I     0  0     OpcodeLUT: [global] array[FirstOpcodeValue..LastOpcodeValue]
00153 I     0  0          of OpcodeLUTEntry;
00154 I     0  0
00155 I     0  0     Memory: [global] MemoryArray;
00156 I     0  0
00157 I     0  0     IP: [global] unsigned;
00158 I     0  0
00159 I     0  0     Flags: [global] FlagType;
00160 I     0  0
00161 I     0  0     SegmentOverRideCount: [global] integer;
00162 I     0  0
00163 I     0  0     SegmentOverRideValue: [global] integer;
00164       0  0
00165       0  0     %include  [-]RegGlobal.defn/list'
00166 I     0  0     AX: [global] unsigned;        [value of the AX register]
00167 I     0  0     CX: [global] unsigned;        [value of the CX register]
00168 I     0  0     DX: [global] unsigned;        [value of the DX register]
00169 I     0  0     BX: [global] unsigned;        [value of the BX register]
00170 I     0  0     SP: [global] unsigned;        {value of the SP register]
00171 I     0  0     BP: [global] unsigned;        {value of the BP register]
00172 I     0  0     SI: [global] unsigned;        {value of the SI register]
00173 I     0  0     DI: [global] unsigned;        {value of the DI register]
00174 I     0  0
00175 I     0  0     ES: [global] unsigned;        [value of the ES register]
00176 I     0  0     CS: [global] unsigned;        {value of the CS register]
00177 I     0  0     SS: [global] unsigned;        {value of the SS register]
00178 I     0  0     DS: [global] unsigned;        {value of the DS register]
00179 I     0  0
00180       0  0
00181       0  0     %include  [-]IOVarGlobal.defn/list'
00182 I     0  0     IOPortMap: [global] PortMap;
00183 I     0  0
00184 I     0  0     IOData: [global] array[FirstIOPortIndex..MaxNumberofIOPorts,
00185 I     0  0   FirstIODataPoint..MaxNumberofIOWords] of unsigned;
00186       0  0
00187       0  0     OutputGenerated: [global] boolean;
00188       0  0
00189       0  0     Punctuation: [global] Characters;
00190       0  0     Numerals: [global] Numbers;
00191       0  0     HexNumerals: [global] LettersandNumbers;
00192       0  0
00193       0  0     JournalingFlag: [global] boolean;
```

```
00194      0  0     Journal: [global] text;
00195      0  0
00196      0  0     RepeatFlag: [global] boolean;
00197      0  0     RepeatUntilZFlag: [global] boolean;
```

```
-LINE-IDC-PL-SL-

00199       0  0     LookUpTable: file of OpcodeLUTEntry;
00200       0  0
00201       0  0     LUTFilename: FileName;
00202       0  0
00203       0  0     Index: integer;
00204       0  0
00205       0  0     IOMapFileName: FileName;
00206       0  0     IOMapFile:     file of PortEntry;
00207       0  0
00208       0  0     IODataFileName: FileName;
00209       0  0     IODataFile:     file of unsigned;
00210       0  0
00211       0  0     IOPortIndex: integer;
00212       0  0     IOMapLoop:   integer;
00213       0  0     IODataLoop:  integer;
00214       0  0
00215       0  0     IOAccessed: boolean;
00216       0  0
00217       0  0     Response: char;
00218       0  0     Yes: LettersandNumbers;
00219       0  0
00220       0  0     LoaderFormat: char;
00221       0  0
00222       0  0     Exists: boolean;
```

```
-LINE-IDC-PL-SL-

00224       0  0 procedure Debugger( DFormat: char); external;
00225       0  0
00226       0  0
00227       0  0 function FileExists( FEFileName: Filename): boolean; Fortran;
00228       0  0
00229       0  0
00230       0  0 %include 'UpperCaseLetter.subpas/list'
00231 I C   0  0 { Title: UpperCaseLetter
00232 I C   0  0
00233 I C   0  0   Purpose: Convert lowercase letter to uppercase
00234 I C   0  0
00235 I C   0  0   Author: William A. Chapman      Date: October 26, 1986
00236 I C   0  0
00237 I C   0  0   Inputs: Upper or lower case letter
00238 I C   0  0
00239 I C   0  0   Outputs: All lowercase letters are converted to uppercase
00240 I C   0  0
00241 I C   0  0   Procedures Invoked: none
00242 I C   0  0 }
00243 I     1  0 procedure UpperCaseLetter( var UPLetter: char);
00244 I     1  0
00245 I     1  1 begin
00246 I     1  1     if ((( ord( UPLetter) - ord( 'a')) >= 0 ) and (( ord( UPLetter) - ord( 'z')) <= 0 ))
00247 I     1  1   then UPLetter := chr( ord( UPLetter) - (ord( 'a') - ord( 'A')));
00248 I     0  0 end;
```

```
-LINE-IDC-PL-SL-

00250       0  1 begin
00251       0  1     writeln;  writeln;
00252       0  1     writeln(               Intel 8086 Simulator');
00253       0  1     writeln(                    Version ', VersionNo:1,  . , ReleaseNo:1 );
00254       0  1     writeln;  writeln;
00255       0  1
00256       0  1     Yes := [ 'Y', 'y', '1'];
00257       0  1     SegmentOverRideCount := 0;
00258       0  1     SegmentOverRideValue := -100;
00259       0  1
00260       0  1     LUTFilename := 'OpcodeLUTFile';
00261       0  1     open( LookUpTable, LUTFilename, readonly);
00262       0  1     reset( LookUpTable);
00263       0  1     for Index := FirstOpcodeValue to LastOpcodeValue do
00264       0  1   read( LookUpTable ,OpcodeLUT[Index]);   {initialize Opcode LUT}
00265       0  1     close( LookUpTable);
00266       0  1
00267       0  1     write( 'Will this program access any IO ports [Y/N] ? ');
00268       0  1     readln( Response);
00269       0  2     if (Response in Yes) then begin
00270       0  2
00271       0  3   repeat
00272       0  3       write( 'Enter name of file containing IO port map:  );
00273       0  3       readln( IOMapFileName);
00274       0  3       Exists := FileExists( %descr IOMapFileName);
00275       0  4       if (Exists = true) then begin
00276       0  4           open( IOMapFile, IOMapFileName, old);
00277       0  4           reset( IOMapFile);
00278       0  5           for IOPortIndex := FirstIOPortIndex to MaxNumberOfIOPorts do begin
00279       0  5               read( IOMapFile, IOPortMap[ IOPortIndex]);
00280       0  5               IOPortMap[ IOPortIndex].DataIndex := FirstIODataPoint;
00281       0  4           end;
00282       0  4           close( IOMapFile);
00283       0  4       end
00284       0  3       else writeln( 'Invalid filename:  , IOMapFileName);
00285       0  2   until Exists;
00286       0  2
00287       0  3   repeat
00288       0  3       write( 'Enter name of file containing IO port data: ');
00289       0  3       readln( IODataFileName);
00290       0  3       Exists := FileExists( %DESCR IODataFileName);
00291       0  4       if (Exists = true) then begin
00292       0  4           open( IODataFile, IODataFileName, old);
00293       0  4           reset( IODataFile);
```

```
00294      0  4           for IOPortIndex := FirstIOPortIndex to MaxNumberOfIOPorts do
00295      0  4               for IODataLoop := FirstIODataPoint to MaxNumberOfIOWords do
00296      0  4                   read( IODataFile, IOData[ IOPortIndex, IODataLoop]);
00297      0  4           close( IODataFile);
00298      0  4       end
00299      0  3       else writeln( 'Invalid filename:  , IODataFileName);
00300      0  2   until Exists;
00301      0  2
00302      0  2   IOAccessed := true;
00303      0  2
00304      0  1     end;
```

```
-LINE-IDC-PL-SL-

00305       0  1
00306       0  1     OutputGenerated := false;
```

```
-LINE-IDC-PL-SL-

00308   C  0  1 [
00309   C  0  1     writeln; writeln;
00310   C  0  1     writeln( 'Do you want the Standard Intel loader format [S], ');
00311   C  0  1     write(       or the Hex loader format [H]: ');
00312   C  0  1     readln( LoaderFormat);
00313   C  0  1     UpperCaseLetter( LoaderFormat);
00314   C  0  1 ]
00315      0  1     LoaderFormat := Standard;
00316      0  1     Debugger( LoaderFormat);
00317      0  1
00318      0  2     if (IOAccessed = true) then begin
00319      0  2   open( IOMapFile, IOMapFileName, old);
00320      0  2   rewrite( IOMapFile);
00321      0  2   for IOMapLoop := FirstIOPortIndex to MaxNumberofIOPorts do
00322      0  2       write( IOMapFile, IOPortMap[ IOMapLoop]);
00323      0  2   close( IOMapFile);
00324      0  1     end; {IOAccessed]
00325      0  1
00326      0  2     if (OutputGenerated = true) then begin
00327      0  2   open( IODataFile, IODataFileName, old);
00328      0  2   rewrite( IODataFile);
00329      0  2   for IOPortIndex := FirstIOPortIndex to MaxNumberofIOPorts do
00330      0  2       for IODataLoop := FirstIODataPoint to MaxNumberofIOWords do
00331      0  2           write( IODataFile, IOData[ IOPortIndex, IODataLoop]);
00332      0  2   close( IODataFile);
00333      0  1     end; {OutputGenerated]
00334      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	1656	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	12407	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS EMULATOR86.PAS

  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
  /OPTIMIZE
  /STANDARD=NONE
  /TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
  /USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
  /NOANALYSIS_DATA
  /NOENVIRONMENT
  /LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]EMULATOR86.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]EMULATOR86.OBJ;22
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	443	00:00.5	00:01.7
Source Analysis	312	00:01.5	00:07.1
Source Listing	23	00:00.9	00:04.9
Tree Construction	155	00:00.2	00:01.0
Flow Analysis	55	00:00.1	00:00.3
Value Propagation	11	00:00.0	00:00.0
Profit Analysis	111	00:00.1	00:00.2
Context Analysis	120	00:00.7	00:03.2
Name Packing	7	00:00.1	00:00.4
Code Selection	90	00:00.2	00:00.8
Final	79	00:00.4	00:01.1
TOTAL	1410	00:04.7	00:20.6

COMPILATION STATISTICS

CPU Time: 00:04.7 (4310 Lines/Minute)

```
-LINE-IDC-PL-SL-

00001   C  0  0 { Title: Execute Instruction
00002   C  0  0
00003   C  0  0   Purpose: Select the proper instruction module to
00004   C  0  0           execute the specified opcode
00005   C  0  0
00006   C  0  0   Author: William A. Chapman      Date: January 26, 1986
00007   C  0  0
00008   C  0  0   Inputs: Opcode record
00009   C  0  0           Register record
00010   C  0  0           Effective Address record
00011   C  0  0
00012   C  0  0   Outputs: The appropriate instruction module is called
00013   C  0  0
00014   C  0  0   Procedures Invoked: DataTransfer
00015   C  0  0                       Arithmetic
00016   C  0  0                       Logic
00017   C  0  0                       String
00018   C  0  0                       ControlTransfer
00019   C  0  0                       ProcessorControl
00020   C  0  0                       FetchInstruction
00021   C  0  0 }
00022      0  0 module ExecuteInstruction(input,output);
00023      0  0
00024      0  0 const
00025      0  0     %include  [-]Const.defn/list'
00026 I    0  0     FirstOpcodeValue = 0;
00027 I    0  0     LastOpcodeValue = 255;
00028 I    0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00029 I    0  0     Low8Bits = %X'FF';                     {mask for low 8 bits}
00030 I    0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00031 I    0  0     Bit8Multiplier = %X'100';
00032 I    0  0     Bit8Divisor    = %X'100';
00033 I    0  0     WordMultiplier = %X'100';
00034 I    0  0
00035 I    0  0     EightBits = 0;                         {width for 8 bits}
00036 I    0  0     SixteenBits = 1;                       {width for 16 bits}
00037 I    0  0
00038 I    0  0     SimpleMode = %B'0';
00039 I    0  0     SimpleRM = %B'110';
00040 I    0  0
00041 I    0  0     LowMemoryLimit = 0;                    {low limit on memory array}
00042 I    0  0     HighMemoryLimit = 2048;                {high limit on memory array}
00043 I    0  0
00044 I    0  0     FilenameLength = 40;
00045      0  0
```

```
00046      0  0     %include  [-]LookUpTableClass.defn/list'
00047 I    0  0     ArithmeticClass          = 'A';
00048 I    0  0     ControlTransferClass  = 'C';
00049 I    0  0     DataTransferClass        = 'D';
00050 I    0  0     ExtendedClass   = 'E';
00051 I    0  0     InvalidClass    = 'I';
00052 I    0  0     LogicClass               = 'L';
00053 I    0  0     ProcessorControlClass = 'P';
00054 I    0  0     StringClass              = 'S';
00055      0  0
```

```
-LINE-IDC-PL-SL-

00056       0  0     WidthBit = %B'1';
00057       0  0
00058       0  0     %include '[-]StopCode.defn/list'
00059 I     0  0     Breakpoint          = 'B';
00060 I     0  0     Call          = 'C';
00061 I     0  0     ControlD            = 'D';
00062 I     0  0     BadOpcode           = 'E';
00063 I     0  0     BadCEAModeValue     = 'F';
00064 I     0  0     BadRMDModeValue     = 'G';
00065 I     0  0     Halt          = 'H';
00066 I     0  0     BadRegisterID = 'I';
00067 I     0  0     BadOpcodeKey  = 'K';
00068 I     0  0     BadMemoryAddress    = 'M';
00069 I     0  0     Normal              = 'N';
00070 I     0  0     BadPortAddress      = 'P';
00071 I     0  0     BadOpcodeClass      = 'Q';
00072 I     0  0     Return              = 'R';
00073 I     0  0     BadCheckSum         = 'S';
00074 I     0  0     BadOperandType      = 'T';
00075 I     0  0     BadMemoryWidth      = 'W';
00076 I     0  0     BadOpcodeExtension  = 'X';
00077 I     0  0     NoMemoryAccess    = 'Z';
```

```
-LINE-IDC-PL-SL-

00079       0  0 type
00080       0  0     %include '[-]Type.defn/list'
00081 I     0  0     ZeroOne = -1..1;                          {defines integer with range
00082 I C   0  0                                               of zero to one with an invalid
00083 I C   0  0                                               state of -1}
00084 I     0  0
00085 I     0  0     OpcodeLUTEntry = record                   {defines entries in opcode LUT}
00086 I     0  0   OpcodeClass: char;                         {class D, A, L, S, C, P}
00087 I     0  0   OpcodeKey: integer;                        {key to interpret opcode from
00088 I C   0  0                                               0 to 7}
00089 I     0  0   DirectionBitPresent: boolean;
00090 I     0  0   WidthBitPresent: boolean
00091 I     0  0     end;
00092 I     0  0
00093 I     0  0     OpcodeType = record
00094 I     0  0   Direction: ZeroOne;                       {to or from CPU}
00095 I     0  0   Full: unsigned;                           {full opcode}
00096 I     0  0   Width: ZeroOne                            {8 or 16 bits}
00097 I     0  0     end;
00098 I     0  0
00099 I     0  0     RegisterType = record
00100 I     0  0   Id: integer;                             {identifier}
00101 I     0  0   Width: ZeroOne                            {8 or 16 bits}
00102 I     0  0     end;
00103 I     0  0
00104 I     0  0     EffectiveAddressType = record
00105 I     0  0   Mode: char;                              {register or memory}
00106 I     0  0   Width: ZeroOne;                          {8 or 16 bits}
00107 I     0  0   Address: unsigned;                       {memory address or
00108 I C   0  0                                               register designation}
00109 I     0  0   Segment: integer                         {segment register to use}
00110 I     0  0     end;
00111 I     0  0
00112 I     0  0     NameType = packed array[1..10] of char;
00113 I     0  0
00114 I     0  0     Characters = set of    ..'~';
00115 I     0  0     LettersAndNumbers = set of '0'..'z';
00116 I     0  0     Letters = set of 'A'..'z';
00117 I     0  0     Numbers = set of '0'..'9';
00118 I     0  0
00119 I     0  0     FileName = packed array [1..FilenameLength] of char;
00120       0  0
00121       0  0 var
00122       0  0     SegmentOverrideCount: [external] integer;
```

```
00123      0  0
00124      0  0      OpcodeValue: integer;
00125      0  0      OpcodeClass: char;
00126      0  0
00127      0  0      InterimOpcodeClass: char;
00128      0  0      Opcode: OpcodeType;
00129      0  0      Register: RegisterType;
00130      0  0      EA: EffectiveAddressType;
```

```
-LINE-IDC-PL-SL-

00132      1  0 [global] procedure ExecuteInstruction( var EIStopCode: char);
00133      1  0
00134      2  0 procedure FetchInstruction( var FIOpcodeClass: char;
00135      2  0           var FIOpcode: OpcodeType;
00136      2  0           var FIRegister: RegisterType;
00137      2  0           var FIEA: EffectiveAddressType;
00138      1  0           var FIStopCode: char); external;
00139      1  0
00140      2  0 procedure DataTransfer(DTOpcode: OpcodeType;
00141      2  0           DTRegister: RegisterType;
00142      2  0           DTEA: EffectiveAddressType;
00143      1  0           var DTStopCode: char); external;
00144      1  0
00145      2  0 procedure Arithmetic(AOpcode: OpcodeType;
00146      2  0           ARegister: RegisterType;
00147      2  0           AEA: EffectiveAddressType;
00148      1  0           var AStopCode: char); external;
00149      1  0
00150      2  0 procedure Logic(LOpcode: OpcodeType;
00151      2  0           LRegister: RegisterType;
00152      2  0           LEA: EffectiveAddressType;
00153      1  0           var LStopCode: char); external;
00154      1  0
00155      2  0 procedure String(SOpcode: OpcodeType;
00156      2  0           SRegister: RegisterType;
00157      2  0           SEA: EffectiveAddressType;
00158      1  0           var SStopCode: char); external;
00159      1  0
00160      2  0 procedure ControlTransfer(CTOpcode: OpcodeType;
00161      2  0           CTRegister: RegisterType;
00162      2  0           CTEA: EffectiveAddressType;
00163      1  0           var CTStopCode: char); external;
00164      1  0
00165      2  0 procedure ProcessorControl(PCOpcode: OpcodeType;
00166      2  0           PCRegister: RegisterType;
00167      2  0           PCEA: EffectiveAddressType;
00168      1  0           var PCStopCode: char); external;
```

```
-LINE-IDC-PL-SL-

00170      1  1 begin
00171      1  1     EIStopCode := Normal;
00172      1  1
00173      1  1     FetchInstruction( InterimOpcodeClass, Opcode, Register, EA, EIStopCode);
00174      1  1     if (EIStopCode = BadOpcodeKey) then InterimOpcodeClass := InvalidClass;
00175      1  1
00176      1  2     case InterimOpcodeClass of
00177      1  2
00178      1  2          ExtendedClass: {Extended Opcodes are translated to their proper class}
00179      1  2
00180      1  3     begin
00181      1  3
00182      1  3       OpcodeValue := int(Opcode.Full);
00183      1  3
00184      1  4       case OpcodeValue of
00185      1  4
00186      1  4       %X'80',%X'82',%X'280',%X'282',%X'380',%X'382',%X'580',%X'582',
00187      1  4       %X'780',%X'782',%X'3F6',%X'4F6',%X'5F6',%X'6F6',%X'7F6',
00188      1  4       %X'0FE',%X'1FE':
00189      1  4           OpcodeClass := ArithmeticClass;
00190      1  4
00191      1  4       %X'180',%X'182',%X'480',%X'482',%X'680',%X'682',%X'0F6',%X'2F6':
00192      1  4           OpcodeClass := LogicClass;
00193      1  4
00194      1  5       %X'2FE',%X'3FE',%X'4FE',%X'5FE': begin
00195      1  5           OpcodeClass := ControlTransferClass;
00196      1  5           Opcode.Full := uor(Opcode.Full, WidthBit);
00197      1  4         end;
00198      1  4
00199      1  5       %X'6FE': begin
00200      1  5           OpcodeClass := DataTransferClass;
00201      1  5           Opcode.Full := uor(Opcode.Full, WidthBit);
00202      1  5         end
00203      1  5
00204      1  4       otherwise EIStopCode := BadOpcodeExtension;
00205      1  4
00206      1  3       end; {Case OpcodeValue}
00207      1  3
00208      1  2         end; {case 'E'}
00209      1  2
00210      1  2   otherwise OpcodeClass := InterimOpcodeClass;
00211      1  2
00212      1  1     end; {case InterimOpcodeClass}
```

```
-LINE-IDC-PL-SL-

00214       1  2     case OpcodeClass of
00215       1  2
00216       1  2   ArithmeticClass: Arithmetic( Opcode, Register, EA, EIStopCode);
00217       1  2
00218       1  2   ControlTransferClass: ControlTransfer( Opcode, Register, EA, EIStopCode);
00219       1  2
00220       1  2   DataTransferClass: DataTransfer( Opcode, Register, EA, EIStopCode);
00221       1  2
00222       1  2   LogicClass: Logic( Opcode, Register, EA, EIStopCode);
00223       1  2
00224       1  2   StringClass: String( Opcode, Register, EA, EIStopCode);
00225       1  2
00226       1  2   ProcessorControlClass: ProcessorControl( Opcode, Register, EA, EIStopCode);
00227       1  2
00228       1  2   InvalidClass: ;
00229       1  2
00230       1  2   otherwise
00231       1  2       EIStopCode := BadOpcodeClass;
00232       1  2
00233       1  2
00234       1  1     end; {case OpcodeClass}
00235       1  1
00236       1  1     if ( SegmentOverrideCount > 0) then SegmentOverrideCount := SegmentOverrideCount - 1;
00237       0  0 end;
00238       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	860	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	41	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS EXECUTEINSTRUCTION.PAS

  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
  /OPTIMIZE
  /STANDARD=NONE
  /TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
  /USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
  /NOANALYSIS_DATA
  /NOENVIRONMENT
  /LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]EXECUTEINSTRUCTION.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]EXECUTEINSTRUCTION.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	400	00:00.5	00:01.3
Source Analysis	282	00:00.8	00:01.8
Source Listing	27	00:00.4	00:02.5
Tree Construction	102	00:00.1	00:00.5
Flow Analysis	35	00:00.1	00:01.1
Value Propagation	10	00:00.0	00:00.0
Profit Analysis	43	00:00.1	00:00.1
Context Analysis	159	00:00.5	00:01.4
Name Packing	6	00:00.0	00:00.0
Code Selection	78	00:00.2	00:00.7
Final	67	00:00.3	00:00.6
TOTAL	1213	00:02.9	00:10.0

COMPILATION STATISTICS

```
  CPU Time:       00:02.9         (4976 Lines/Minute)
```

```
-LINE-IDC-PL-SL-

00001    C  0  0 { Title: Fetch Code module
00002    C  0  0
00003    C  0  0   Purpose: Obtain 8 bits of data from the store "Memory"
00004    C  0  0            using the values of the Instruction pointer and
00005    C  0  0            CS Register for address and segment values
00006    C  0  0            respectively.  Increment the Instruction Pointer.
00007    C  0  0
00008    C  0  0   Author: William A. Chapman      Date: December 24, 1985
00009    C  0  0
00010    C  0  0   Inputs: Instruction Pointer
00011    C  0  0
00012    C  0  0   Outputs: 8 bits of unsigned data are passed to the
00013    C  0  0            calling routine.
00014    C  0  0
00015    C  0  0   Procedures Invoked: FetchRegPtr
00016    C  0  0                       FetchMemory
00017    C  0  0
00018    C  0  0 }
00019       0  0 module FetchCode(input,output);
00020       0  0
00021       0  0 const
00022       0  0     %include '[-]Const.defn/list'
00023  I    0  0     FirstOpcodeValue = 0;
00024  I    0  0     LastOpcodeValue = 255;
00025  I    0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00026  I    0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00027  I    0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00028  I    0  0     Bit8Multiplier = %X'100';
00029  I    0  0     Bit8Divisor    = %X'100';
00030  I    0  0     WordMultiplier = %X'100';
00031  I    0  0
00032  I    0  0     EightBits = 0;                       {width for 8 bits}
00033  I    0  0     SixteenBits = 1;                     {width for 16 bits}
00034  I    0  0
00035  I    0  0     SimpleMode = %B'0';
00036  I    0  0     SimpleRM = %B'110';
00037  I    0  0
00038  I    0  0     LowMemoryLimit = 0;                  {low limit on memory array}
00039  I    0  0     HighMemoryLimit = 2048;              {high limit on memory array}
00040  I    0  0
00041  I    0  0     FilenameLength = 40;
00042       0  0
00043       0  0     CSWidth = 2;          {width parameter for segment registers}
00044       0  0     CSDesignator = %B'01';        {specify code segment register}
00045       0  0
```

```
00046       0   0     %include  [-]StopCode.defn/list'
00047 I     0   0     Breakpoint              = 'B';
00048 I     0   0     Call            = 'C';
00049 I     0   0     ControlD                = 'D';
00050 I     0   0     BadOpcode               = 'E';
00051 I     0   0     BadCEAModeValue         = 'F';
00052 I     0   0     BadRMDModeValue         = 'G';
00053 I     0   0     Halt            = 'H';
00054 I     0   0     BadRegisterID = 'I';
00055 I     0   0     BadOpcodeKey  = 'K';
```

```
-LINE-IDC-PL-SL-

00056 I     0  0      BadMemoryAddress      = 'M';
00057 I     0  0      Normal                = 'N';
00058 I     0  0      BadPortAddress        = 'P';
00059 I     0  0      BadOpcodeClass        = 'Q';
00060 I     0  0      Return                = 'R';
00061 I     0  0      BadCheckSum           = 'S';
00062 I     0  0      BadOperandType        = 'T';
00063 I     0  0      BadMemoryWidth        = 'W';
00064 I     0  0      BadOpcodeExtension    = 'X';
00065 I     0  0      NoMemoryAccess      = 'Z';
```

```
-LINE-IDC-PL-SL-

00067       0  0 type
00068       0  0     %include '[-]Type.defn/list'
00069 I     0  0     ZeroOne = -1..1;                        {defines integer with range
00070 I C   0  0                                              of zero to one with an invalid
00071 I C   0  0                                              state of -1}
00072 I     0  0
00073 I     0  0     OpcodeLUTEntry = record                 {defines entries in opcode LUT}
00074 I     0  0   OpcodeClass: char;                        {class D, A, L, S, C, P}
00075 I     0  0   OpcodeKey: integer;                       {key to interpret opcode from
00076 I C   0  0                                              0 to 7}
00077 I     0  0   DirectionBitPresent: boolean;
00078 I     0  0   WidthBitPresent: boolean
00079 I     0  0     end;
00080 I     0  0
00081 I     0  0     OpcodeType = record
00082 I     0  0   Direction: ZeroOne;                      {to or from CPU}
00083 I     0  0   Full: unsigned;                          {full opcode}
00084 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00085 I     0  0     end;
00086 I     0  0
00087 I     0  0     RegisterType = record
00088 I     0  0   Id: integer;                             {identifier}
00089 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00090 I     0  0     end;
00091 I     0  0
00092 I     0  0     EffectiveAddressType = record
00093 I     0  0   Mode: char;                              {register or memory}
00094 I     0  0   Width: ZeroOne;                          {8 or 16 bits}
00095 I     0  0   Address: unsigned;                      {memory address or
00096 I C   0  0                                              register designation}
00097 I     0  0   Segment: integer                         {segment register to use}
00098 I     0  0     end;
00099 I     0  0
00100 I     0  0     NameType = packed array[1..10] of char;
00101 I     0  0
00102 I     0  0     Characters = set of ' '..'~';
00103 I     0  0     LettersAndNumbers = set of '0'..'z';
00104 I     0  0     Letters = set of 'A'..'z';
00105 I     0  0     Numbers = set of '0'..'9';
00106 I     0  0
00107 I     0  0     FileName = packed array [1..FilenameLength] of char;
```

```
-LINE-IDC-PL-SL-

00109       0  0 var
00110       0  0     IP: [external] unsigned;      {instruction pointer]
00111       0  0     CSValue: unsigned;            {value of the code segment register]
00112       0  0
00113       0  0     Continue: [external] LettersAndNumbers;
00114       0  0
00115       0  0
00116       1  0 [global] procedure FetchCode(var CodeValue: unsigned;
00117       1  0           var FCStopCode: char);
00118       1  0
00119       2  0 procedure FetchRegPtr(FRPWidth: integer;
00120       2  0           FRPDesignator: integer;
00121       2  0           var FRPValue: unsigned;
00122       1  0           var FRPStopCode: char); external;
00123       1  0
00124       2  0 procedure FetchMemory(FMAddress: unsigned;
00125       2  0           FMSegment: unsigned;
00126       2  0           var FMValue: unsigned;
00127       1  0           var FMStopCode: char); external;
00128       1  0
00129       1  0
00130       1  0
00131       1  1 begin
00132       1  1     FetchRegPtr( CSWidth, CSDesignator, CSValue, FCStopCode); {get Code Segment Register]
00133       1  1     FetchMemory(IP,CSValue,CodeValue, FCStopCode); { get value from memory]
00134       1  1     if (FCStopCode in Continue) then IP := IP + 1; {update Instruction Pointer]
00135       0  0 end;
00136       0  0 end.
```

```
-LINE-IDC-PL-SL-

00001   C  0  0 { Title: Fetch Code Data
00002   C  0  0
00003   C  0  0   Purpose: Fetch immediate mode data from the code segment
00004   C  0  0
00005   C  0  0   Author: William A. Chapman      Date: January 28, 1986
00006   C  0  0
00007   C  0  0   Inputs: Width of data to acquire (8 or 16 bits)
00008   C  0  0
00009   C  0  0   Outputs: Data of specified width is returned
00010   C  0  0
00011   C  0  0   Procedures Invoked: FetchCode
00012   C  0  0 }
00013      0  0 module FetchCodeData(input, output );
00014      0  0
00015      0  0 const
00016      0  0     %include  [-]Const.defn/list'
00017 I    0  0     FirstOpcodeValue = 0;
00018 I    0  0     LastOpcodeValue = 255;
00019 I    0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00020 I    0  0     Low8Bits = %X'FF';               {mask for low 8 bits}
00021 I    0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00022 I    0  0     Bit8Multiplier = %X'100';
00023 I    0  0     Bit8Divisor    = %X'100';
00024 I    0  0     WordMultiplier = %X'100';
00025 I    0  0
00026 I    0  0     EightBits = 0;                       {width for 8 bits}
00027 I    0  0     SixteenBits = 1;                     {width for 16 bits}
00028 I    0  0
00029 I    0  0     SimpleMode = %B'0';
00030 I    0  0     SimpleRM = %B'110';
00031 I    0  0
00032 I    0  0     LowMemoryLimit = 0;                  {low limit on memory array}
00033 I    0  0     HighMemoryLimit = 2048;              {high limit on memory array}
00034 I    0  0
00035 I    0  0     FilenameLength = 40;
00036      0  0
00037      0  0     ClearData = 0;
00038      0  0
00039      0  0 type
00040      0  0     %include  [-]Type.defn/list'
00041 I    0  0     ZeroOne = -1..1;                          {defines integer with range
00042 I C  0  0                                              of zero to one with an invalid
00043 I C  0  0                                              state of -1}
00044 I    0  0
00045 I    0  0     OpcodeLUTEntry = record                   {defines entries in opcode LUT}
```

```
00046 I     0   0   OpcodeClass: char;                  {class D, A, L, S, C, P}
00047 I     0   0   OpcodeKey: integer;                 {key to interpret opcode from
00048 I C   0   0                                        0 to 7}
00049 I     0   0   DirectionBitPresent: boolean;
00050 I     0   0   WidthBitPresent: boolean
00051 I     0   0     end;
00052 I     0   0
00053 I     0   0     OpcodeType = record
00054 I     0   0   Direction: ZeroOne;                {to or from CPU}
00055 I     0   0   Full: unsigned;                     {full opcode}
```

```
-LINE-IDC-PL-SL-

00056 I    0  0   Width: ZeroOne                          {8 or 16 bits}
00057 I    0  0     end;
00058 I    0  0
00059 I    0  0     RegisterType = record
00060 I    0  0   Id: integer;                            {identifier}
00061 I    0  0   Width: ZeroOne                          {8 or 16 bits}
00062 I    0  0     end;
00063 I    0  0
00064 I    0  0     EffectiveAddressType = record
00065 I    0  0   Mode: char;                             {register or memory}
00066 I    0  0   Width: ZeroOne;                         {8 or 16 bits}
00067 I    0  0   Address: unsigned;                      {memory address or
00068 I C  0  0                                            register designation}
00069 I    0  0   Segment: integer                        {segment register to use}
00070 I    0  0     end;
00071 I    0  0
00072 I    0  0     NameType = packed array[1..10] of char;
00073 I    0  0
00074 I    0  0     Characters = set of ' '..'~';
00075 I    0  0     LettersAndNumbers = set of '0'..'z';
00076 I    0  0     Letters = set of 'A'..'z';
00077 I    0  0     Numbers = set of '0'..'9';
00078 I    0  0
00079 I    0  0     FileName = packed array [1..FilenameLength] of char;
```

```
-LINE-IDC-PL-SL-

00081       0  0 var
00082       0  0     LowData: unsigned;
00083       0  0     HighData: unsigned;
00084       0  0
00085       0  0     Continue: [external] LettersAndNumbers;
00086       0  0
00087       1  0 [global] procedure FetchCodeData(Width: integer;
00088       1  0           var ReturnedData: unsigned;
00089       1  0           var FCDStopCode: char);
00090       1  0
00091       2  0 procedure FetchCode(var FCData: unsigned;
00092       1  0           var FCStopCode: char); external;
00093       1  0
00094       1  0
00095       1  0
00096       1  1 begin
00097       1  1     HighData := ClearData;
00098       1  1     ReturnedData := ClearData;
00099       1  1     FetchCode(LowData, FCDStopCode);
00100       1  2     if (FCDStopCode in Continue) then begin
00101       1  2   if (Width = SixteenBits) then FetchCode( HighData, FCDStopCode);
00102       1  3   if (FCDStopCode in Continue) then begin
00103       1  3       HighData := uand((HighData * Bit8Multiplier), High8Bits);
00104       1  3       ReturnedData := uor(LowData, HighData);
00105       1  2  end;
00106       1  1     end;
00107       0  0 end;
00108       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	132	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	8	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

PAS/LIS FETCHCODEDATA.PAS

/CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
/DEBUG=(NOSYMBOLS,TRACEBACK)
/SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
/OPTIMIZE
/STANDARD=NONE
/TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
/USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
/NOANALYSIS_DATA
/NOENVIRONMENT
/LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FETCHCODEDATA.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FETCHCODEDATA.OBJ;1
/NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	404	00:00.4	00:03.5
Source Analysis	233	00:00.4	00:02.1
Source Listing	15	00:00.2	00:01.3
Tree Construction	92	00:00.1	00:00.2
Flow Analysis	36	00:00.0	00:00.0
Value Propagation	10	00:00.0	00:00.0
Profit Analysis	43	00:00.0	00:00.0
Context Analysis	189	00:00.2	00:00.3
Name Packing	6	00:00.0	00:00.0
Code Selection	74	00:00.1	00:00.1
Final	63	00:00.1	00:00.4
TOTAL	1169	00:01.6	00:07.8

COMPILATION STATISTICS

CPU Time: 00:01.6 (3951 Lines/Minute)

```
-LINE-IDC-PL-SL-

00001    C  0  0 { Title: Fetch Input
00002    C  0  0
00003    C  0  0   Purpose: Get datum from the input store
00004    C  0  0
00005    C  0  0   Author: William A. Chapman      Date: February 1, 1986
00006    C  0  0
00007    C  0  0   Inputs: InputData - input data
00008    C  0  0           InputPortMap - maps port number into InputData
00009    C  0  0                    and keeps track of location in array
00010    C  0  0
00011    C  0  0   Outputs: 8 or 16 bits of datum is returned
00012    C  0  0
00013    C  0  0   Procedures Invoked: LIB$STOP
00014    C  0  0           $QIOW
00015    C  0  0           (Get8BitsofData)
00016    C  0  0
00017    C  0  0   Functions Invoked: MaskShiftRight
00018    C  0  0 }
00019       0  0 [inherit ('sys$library:starlet')] module FetchInput(input,output);
00020       0  0
00021       0  0 const
00022       0  0     %include  [-]Const.defn/list'
00023 I     0  0     FirstOpcodeValue = 0;
00024 I     0  0     LastOpcodeValue = 255;
00025 I     0  0     All16Bits = %X'FFFF';         {mask for all 16 bits}
00026 I     0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00027 I     0  0     High8Bits = %X'FF00';         {mask for high 8 bits}
00028 I     0  0     Bit8Multiplier = %X'100';
00029 I     0  0     Bit8Divisor    = %X'100';
00030 I     0  0     WordMultiplier = %X'100';
00031 I     0  0
00032 I     0  0     EightBits = 0;                      {width for 8 bits}
00033 I     0  0     SixteenBits = 1;                    {width for 16 bits}
00034 I     0  0
00035 I     0  0     SimpleMode = %B'0';
00036 I     0  0     SimpleRM = %B'110';
00037 I     0  0
00038 I     0  0     LowMemoryLimit = 0;                 {low limit on memory array}
00039 I     0  0     HighMemoryLimit = 2048;             {high limit on memory array}
00040 I     0  0
00041 I     0  0     FilenameLength = 40;
00042       0  0
00043       0  0     %include  [-]IOConst.defn/list'
00044 I     0  0     KeyboardInPortNumber  = %X'FF';
00045 I     0  0     CRTOutPortNumber       = %X'FF';
```

```
00046 I     0   0
00047 I     0   0      FirstIOPortIndex   = 1;
00048 I     0   0      MaxNumberofIOPorts = 8;
00049 I     0   0
00050 I     0   0      FirstIODataPoint   = 1;
00051 I     0   0      MaxNumberofIOWords = 64;
00052 I     0   0      MaxNumberofIOBytes = 256;     {4 times the number of words}
00053 I     0   0
00054 I     0   0      InputIndicator  = 'I';
00055 I     0   0      OutputIndicator = 'O';
```

```
-LINE-IDC-PL-SL-

00056 I    0  0     IOIndicator     = 'B';
00057 I    0  0
00058 I    0  0     LowerPortIndicator = 'L';
00059 I    0  0     UpperPortIndicator = 'U';
00060 I    0  0
00061 I    0  0     InvalidAddress = - 1;
00062 I    0  0     InvalidLocation = - 1;
00063 I    0  0     InvalidIndex = - 1;
00064      0  0
00065      0  0     %include '[-]Byte.defn/list'
00066 I    0  0     Byte0Mask = %X'000000FF';
00067 I    0  0     Byte1Mask = %X'0000FF00';
00068 I    0  0     Byte2Mask = %X'00FF0000';
00069 I    0  0     Byte3Mask = %X'FF000000';
00070 I    0  0
00071 I    0  0     Byte0Divisor = %X'00000001 ;
00072 I    0  0     Byte1Divisor = %X'00000100';
00073 I    0  0     Byte2Divisor = %X'00010000';
00074 I    0  0     Byte3Divisor = %X'01000000';
00075 I    0  0
00076 I    0  0     Byte0Multiplier = %X'00000001';
00077 I    0  0     Byte1Multiplier = %X'00000100';
00078 I    0  0     Byte2Multiplier = %X'00010000';
00079 I    0  0     Byte3Multiplier = %X'01000000';
00080 I    0  0
00081 I    0  0     BytesPerWord = 4;
00082      0  0
00083      0  0     NullDatum = 0;
00084      0  0
00085      0  0     %include  [-]StopCode.defn/list'
00086 I    0  0     Breakpoint              = 'B';
00087 I    0  0     Call          = 'C';
00088 I    0  0     ControlD            = 'D';
00089 I    0  0     BadOpcode             = 'E';
00090 I    0  0     BadCEAModeValue       = 'F';
00091 I    0  0     BadRMDModeValue       = 'G';
00092 I    0  0     Halt          = 'H';
00093 I    0  0     BadRegisterID = 'I';
00094 I    0  0     BadOpcodeKey  = 'K';
00095 I    0  0     BadMemoryAddress      = 'M';
00096 I    0  0     Normal                = 'N';
00097 I    0  0     BadPortAddress        = 'P';
00098 I    0  0     BadOpcodeClass        = 'Q';
00099 I    0  0     Return                = 'R';
```

```
00100 I     0  0      BadCheckSum             = 'S';
00101 I     0  0      BadOperandType          = 'T';
00102 I     0  0      BadMemoryWidth          = 'W';
00103 I     0  0      BadOpcodeExtension      = 'X';
00104 I     0  0      NoMemoryAccess        = 'Z';
```

```
-LINE-IDC-PL-SL-

00106       0  0 type
00107       0  0     %include  [-]Type.defn/list'
00108 I     0  0     ZeroOne = -1..1;                        {defines integer with range
00109 I C   0  0                                              of zero to one with an invalid
00110 I C   0  0                                              state of -1}
00111 I     0  0
00112 I     0  0     OpcodeLUTEntry = record                 {defines entries in opcode LUT}
00113 I     0  0   OpcodeClass: char;                        {class D, A, L, S, C, P}
00114 I     0  0   OpcodeKey: integer;                       {key to interpret opcode from
00115 I C   0  0                                              0 to 7}
00116 I     0  0   DirectionBitPresent: boolean;
00117 I     0  0   WidthBitPresent: boolean
00118 I     0  0     end;
00119 I     0  0
00120 I     0  0     OpcodeType = record
00121 I     0  0   Direction: ZeroOne;                      {to or from CPU}
00122 I     0  0   Full: unsigned;                          {full opcode}
00123 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00124 I     0  0     end;
00125 I     0  0
00126 I     0  0     RegisterType = record
00127 I     0  0   Id: integer;                            {identifier}
00128 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00129 I     0  0     end;
00130 I     0  0
00131 I     0  0     EffectiveAddressType = record
00132 I     0  0   Mode: char;                              {register or memory}
00133 I     0  0   Width: ZeroOne;                         {8 or 16 bits}
00134 I     0  0   Address: unsigned;                      {memory address or
00135 I C   0  0                                              register designation}
00136 I     0  0   Segment: integer                        {segment register to use}
00137 I     0  0     end;
00138 I     0  0
00139 I     0  0     NameType = packed array[1..10] of char;
00140 I     0  0
00141 I     0  0     Characters = set of    ..'~';
00142 I     0  0     LettersAndNumbers = set of '0'..'z';
00143 I     0  0     Letters = set of 'A'..'z';
00144 I     0  0     Numbers = set of '0'..'9';
00145 I     0  0
00146 I     0  0     FileName = packed array [1..FilenameLength] of char;
00147       0  0
00148       0  0     %include  [-]IOType.defn/list'
00149 I     0  0     PortEntry = record
```

```
00150 I     0  0    Address: unsigned;
00151 I     0  0    AuxIndex: integer;
00152 I     0  0    DataIndex: integer;
00153 I     0  0    InputIndex: integer;
00154 I     0  0    PortWidth: integer;
00155 I     0  0    LowerUpperIndicator: char;
00156 I     0  0    InOutIndicator: char
00157 I     0  0      end;
00158 I     0  0
00159 I     0  0      PortMap = array[ FirstIOPortIndex..MaxNumberOfIOPorts] of PortEntry;
00160       0  0
```

```
-LINE-IDC-PL-SL-

00161       0  0     QuadWord = [word] 0..65535;
00162       0  0     StatusBlock = record
00163       0  0   IOStatus: QuadWord;
00164       0  0   Count: QuadWord;
00165       0  0   DeviceInfo: integer
00166       0  0     end;
00167       0  0
00168       0  0
00169       0  0
00170       0  0
00171       0  0 var
00172       0  0     %include  [-]IOVarExternal.defn/list'
00173 I     0  0     IOPortMap: [external] PortMap;
00174 I     0  0
00175 I     0  0     IOData: [external] array[FirstIOPortIndex..MaxNumberofIOPorts,
00176 I     0  0   FirstIODataPoint..MaxNumberofIOWords] of unsigned;
00177       0  0
00178       0  0     LowDatum: unsigned;
00179       0  0     HighDatum: unsigned;
00180       0  0
00181       0  0     IAIndex: integer;
00182       0  0     Found: boolean;
00183       0  0     Done: boolean;
00184       0  0
00185       0  0     ReadInChar: [volatile] char;
00186       0  0
00187       0  0     SysStat: integer;
00188       0  0
00189       0  0     ChannelID: [external, volatile] array[1..2] of QuadWord;
00190       0  0
00191       0  0     IOBlock: StatusBlock;
00192       0  0
00193       0  0
00194       0  0
00195       1  0 [global] procedure FetchInput(FInWidth: integer;
00196       1  0           FInPortNumber: unsigned;
00197       1  0           var FInDatum: unsigned;
00198       1  0           var FIStopCode: char);
00199       1  0
00200       2  0 function MaskShiftRight(MSRValue: unsigned;
00201       2  0           MSRMask: unsigned;
00202       1  0           MSRDivisor:integer): unsigned; external;
00203       1  0
00204       1  0
```

```
00205       1  0 [asynchronous] procedure lib$stop( %immed L$SStopValue: integer); external;
00206       1  0
```

```
-LINE-IDC-PL-SL-

00208       1  0     %include 'Get8BitsOfData.subpas/list'
00209 I C   1  0 { Title: Get8BitsofData
00210 I C   1  0
00211 I C   1  0   Purpose: Retreive 8 bits of data from the IO data buffer
00212 I C   1  0           and return to the calling routine.
00213 I C   1  0
00214 I C   1  0   Author: William A. Chapman      Date: May 16, 1986
00215 I C   1  0
00216 I C   1  0   Inputs: IO data is read from IO data buffer.
00217 I C   1  0
00218 I C   1  0   Outputs: Data is returned to calling routine.
00219 I C   1  0
00220 I C   1  0   Procedures Invoked: MaskShiftRight
00221 I C   1  0 }
00222 I     2  0 procedure Get8BitsofData( PortPointer: integer;
00223 I     2  0           Location: integer;
00224 I     2  0           var Value: unsigned);
00225 I     2  0
00226 I     2  0
00227 I     2  0 var
00228 I     2  0     WordPointer: integer;
00229 I     2  0     BytePointer: integer;
00230 I     2  0
00231 I     2  0
00232 I     2  0
00233 I     2  1 begin
00234 I     2  1     WordPointer := (((Location - 1) div BytesPerWord) + 1);
00235 I     2  1     BytePointer := ((Location - 1)  rem BytesPerWord);
00236 I     2  1     Value := NullDatum;
00237 I     2  1
00238 I     2  2     case BytePointer of
00239 I     2  2
00240 I     2  2     0: Value:= MaskShiftRight( IOData[ PortPointer, WordPointer],
00241 I     2  2           Byte0Mask, Byte0Divisor);
00242 I     2  2
00243 I     2  2     1: Value:= MaskShiftRight( IOData[ PortPointer, WordPointer],
00244 I     2  2           Byte1Mask, Byte1Divisor);
00245 I     2  2
00246 I     2  2     2: Value:= MaskShiftRight( IOData[ PortPointer, WordPointer],
00247 I     2  2           Byte2Mask, Byte2Divisor);
00248 I     2  2
00249 I     2  2     3: Value:= MaskShiftRight( IOData[ PortPointer, WordPointer],
00250 I     2  2           Byte3Mask, Byte3Divisor);
00251 I     2  2
```

```
00252 I    2  2     otherwise writeln('Get8BitsofData: Invalid byte pointer:  , BytePointer:1);
00253 I    2  2
00254 I    2  1     end; {case BytePointer}
00255 I    2  1
00256 I    1  0 end; {procedure Get8BitsofData}
```

-LINE-IDC-PL-SL-

```
00258      1  1 begin
00259      1  1     HighDatum := NullDatum;
00260      1  1     LowDatum := NullDatum;
00261      1  1     IAIndex := FirstIOPortIndex;
00262      1  1     Found := false;
00263      1  1     Done := false;
00264      1  1
00265      1  2     if (FInPortNumber = KeyboardInPortNumber) then begin
00266      1  2
00267      1  2   SysStat := $QIOW( efn := 2, chan := ChannelID[2], func := IO$_ReadVBlk,
00268      1  2       iosb := IOBlock, p1 := %ref ReadInChar, p2 := 1);
00269      1  2   if (not odd( SysStat)) then Lib$Stop( SysStat);
00270      1  2
00271      1  2   LowDatum := uand( uint( ReadInChar), Low8Bits);
00272      1  2     end
00273      1  2     else begin
00274      1  3   while (not Done) do begin
00275      1  4       if (FInPortNumber = IOPortMap[IAIndex].Address) then begin
00276      1  4           Found := true;
00277      1  4           Done := true;
00278      1  4       end
00279      1  3       else IAIndex := IAIndex + 1;
00280      1  4       if (IAIndex > MaxNumberofIOPorts) then begin
00281      1  4           FIStopCode := BadPortAddress;
00282      1  4           Done := true;
00283      1  3       end;
00284      1  2   end; {while not Done]
00285      1  2
00286      1  3   if (Found = true) then begin
00287      1  3       Get8BitsofData( IAIndex, IOPortMap[IAIndex].DataIndex, LowDatum);
00288      1  3       if (IOPortMap[IAIndex].InOutIndicator = InputIndicator) then
00289      1  3         IOPortMap[IAIndex].DataIndex := IOPortMap[IAIndex].DataIndex + 1;
00290      1  4       if (FInWidth = SixteenBits) then begin
00291      1  4           IAIndex := IOPortMap[IAIndex].AuxIndex;
00292      1  4           Get8BitsofData( IAIndex, IOPortMap[IAIndex].DataIndex, HighDatum);
00293      1  4           if (IOPortMap[IAIndex].InOutIndicator = InputIndicator) then
00294      1  4             IOPortMap[IAIndex].DataIndex := IOPortMap[IAIndex].DataIndex + 1;
00295      1  3       end; {if (FInWidth = SixteenBits)]
00296      1  2   end; {Found = true]
00297      1  1     end; {else begin]
00298      1  1
00299      1  1     FInDatum := uor(LowDatum, uand((HighDatum * Bit8Multiplier),High8Bits));
00300      0  0 end;
00301      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	770	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	28	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

ENVIRONMENT STATISTICS

File	Symbols Total	Symbols Loaded	Symbols Percent
SYS$COMMON:[SYSLIB]STARLET.PEN;4	20797	32	0

COMMAND QUALIFIERS

```
  PAS/LIS FETCHINPUT.PAS

  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
  /OPTIMIZE
  /STANDARD=NONE
  /TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
  /USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
  /NOANALYSIS_DATA
  /NOENVIRONMENT
  /LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FETCHINPUT.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FETCHINPUT.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	427	00:00.5	00:05.1
Source Analysis	788	00:04.7	00:27.7
Source Listing	13	00:00.7	00:03.3
Tree Construction	115	00:00.3	00:01.0
Flow Analysis	38	00:00.2	00:00.6
Value Propagation	10	00:00.0	00:00.0
Profit Analysis	101	00:00.2	00:00.2
Context Analysis	168	00:01.1	00:04.9
Name Packing	6	00:00.0	00:00.2

Compilation Complete

```
-LINE-IDC-PL-SL-

00001   C  0  0 { Title: FetchInputLine
00002   C  0  0
00003   C  0  0   Purpose: Get a line of input from the Operator
00004   C  0  0
00005   C  0  0   Author: William A. Chapman      Date: July 29, 1986
00006   C  0  0
00007   C  0  0   Inputs: Input line provided by the operator
00008   C  0  0
00009   C  0  0   Outputs: Input Line
00010   C  0  0            Pointer to the beginning of the input line
00011   C  0  0            Length of the input line
00012   C  0  0            Input line is written to a journal file
00013   C  0  0
00014   C  0  0   Procedures Invoked: none
00015   C  0  0 }
00016      0  0 module FetchInputLine( input, output );
00017      0  0
00018      0  0 const
00019      0  0     %include '[-]Const.defn/list'
00020 I    0  0     FirstOpcodeValue = 0;
00021 I    0  0     LastOpcodeValue = 255;
00022 I    0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00023 I    0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00024 I    0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00025 I    0  0     Bit8Multiplier = %X'100';
00026 I    0  0     Bit8Divisor    = %X'100';
00027 I    0  0     WordMultiplier = %X'100';
00028 I    0  0
00029 I    0  0     EightBits = 0;                      {width for 8 bits}
00030 I    0  0     SixteenBits = 1;                    {width for 16 bits}
00031 I    0  0
00032 I    0  0     SimpleMode = %B'0';
00033 I    0  0     SimpleRM = %B'110';
00034 I    0  0
00035 I    0  0     LowMemoryLimit = 0;                 {low limit on memory array}
00036 I    0  0     HighMemoryLimit = 2048;             {high limit on memory array}
00037 I    0  0
00038 I    0  0     FilenameLength = 40;
```

```
-LINE-IDC-PL-SL-

00040       0  0     %include '[-]DebugConst.defn/list'
00041 I     0  0     FirstKeywordEntry = 0;
00042 I     0  0     LastKeywordEntry  = 100;
00043 I     0  0     UndefinedName = 'Qqqqqqqq';
00044 I     0  0     UndefinedClass = 'z';
00045 I     0  0     UndefinedWidth = -1;
00046 I     0  0     UndefinedID = -2;
00047 I     0  0     PageSize = 55;
00048 I     0  0     FF = 12;
00049 I     0  0     MinID = -1;
00050 I     0  0     MaxID = 25;
00051 I     0  0     MinWidth = 0;
00052 I     0  0     MaxWidth = 3;
00053 I     0  0     SegmentOffset = 16;
00054 I     0  0
00055 I     0  0     BlankClass       =    ;
00056 I     0  0     BreakpointClass  = 'B';
00057 I     0  0     ErrorClass       = 'E';
00058 I     0  0     FlagClass        = 'F';
00059 I     0  0     InputOutputClass = 'I';
00060 I     0  0     MemoryClass      = 'M';
00061 I     0  0     NumericClass     = 'N';
00062 I     0  0     OperatorClass    = 'O';
00063 I     0  0     QualifierClass   = 'Q';
00064 I     0  0     RegisterClass    = 'R';
00065 I     0  0     StackClass       = 'S';
00066 I     0  0     TrueFalseClass   = 'T';
00067 I     0  0     UtilityClass     = 'U';
00068 I     0  0     ExecuteClass     = 'X';
00069 I     0  0
00070 I     0  0     Blank  =    ;
00071 I     0  0     Elipse =  .';
00072 I     0  0
00073 I     0  0     StartOfData = 1;
00074 I     0  0     DataArraySize = 50;
00075 I     0  0
00076 I     0  0     EndofLine = 0;
00077 I     0  0     StartofLine = 1;
00078 I     0  0
00079 I     0  0     ShortStringLength = 8;
00080 I     0  0     InputLineLength = 120;
00081 I     0  0
00082 I     0  0     BooleanID  = 1;
00083 I     0  0     ByteID     = 2;
```

```
00084 I    0  0     IntegerID  = 3;
00085 I    0  0     PointerID  = 4;
00086 I    0  0     SIntegerID = 5;
00087 I    0  0     WordID     = 6;
00088 I    0  0     LengthID   = 3;
00089 I    0  0     ToID       = 5;
00090 I    0  0
00091 I    0  0     AssignmentID = 1;
00092 I    0  0     ColonID      = 2;
00093 I    0  0     PlusID       = 3;
00094 I    0  0     MinusID      = 4;
```

```
-LINE-IDC-PL-SL-

00095 I    0  0     CommaID      = 5;
00096 I    0  0
00097 I    0  0     NullEntries = 0;
00098 I    0  0
00099 I    0  0     AddressState   = 'A';
00100 I    0  0     BooleanState   = 'B';
00101 I    0  0     ErrorState     = 'E';
00102 I    0  0     HexState       = 'H';
00103 I    0  0     IntegerState   = 'I';
00104 I    0  0     SegmentState   = 'S';
00105 I    0  0     SeparatorState =  ,';
00106 I    0  0     UnaryState     = '-';
00107 I    0  0     ColonState     =  :';
00108 I    0  0
00109 I    0  0     NegativeOperator = -1;
00110 I    0  0     PositiveOperator = +1;
00111 I    0  0
00112 I    0  0     NullSegmentValue = 0;
00113 I    0  0     NullAddressValue = 0;
00114 I    0  0
00115 I    0  0     FirstBreakpoint  =  0;
00116 I    0  0     LastBreakpoint   = 15;
00117 I    0  0     AllBPIDs         = 16;
00118 I    0  0     Activate         = true;
00119 I    0  0     DeActivate       = false;
00120 I    0  0     BreakpointOpcode = %X'CC';
00121      0  0     Prompt = '*';
```

```
-LINE-IDC-PL-SL-

00123       0  0 type
00124       0  0     %include  [-]Type.defn/nolist'
00164   C   0  0 {   ZeroOne = -1..1;}
00165       0  0
00166       0  0     %include  [-]DebugType.defn/list'
00167 I     0  0     NameString = packed array[1..ShortStringLength] of char;
00168 I     0  0
00169 I     0  0     KeywordCharacteristics = record
00170 I     0  0   Name: NameString;
00171 I     0  0   Class: char;
00172 I     0  0   Width: ZeroOne;
00173 I     0  0   ID: integer;
00174 I     0  0   Value: unsigned
00175 I     0  0     end;
00176 I     0  0
00177 I     0  0     InputPointerRange = integer;
00178 I     0  0     InputLine = packed array [1..InputLineLength] of char;
00179 I     0  0
00180 I     0  0     ShortString = packed array [1..ShortStringLength] of char;
00181 I     0  0
00182 I     0  0     DataArrayType = array [StartOfData..DataArraySize] of unsigned;
00183 I     0  0
00184 I     0  0     BreakRecord = record
00185 I     0  0   Encountered: boolean;
00186 I     0  0   Activated: boolean;
00187 I     0  0   Segment: unsigned;
00188 I     0  0   Address: unsigned;
00189 I     0  0   PhysicalAddress: unsigned;
00190 I     0  0   Code: unsigned
00191 I     0  0     end;
00192 I     0  0
00193 I     0  0     BreakPointTableType = array [FirstBreakpoint..LastBreakpoint] of BreakRecord;
00194       0  0
00195       0  0 var
00196       0  0     InputCharacter: char;
00197       0  0     Loop: integer;
00198       0  0
00199       0  0     Journal: [external] text;
00200       0  0     JournalingFlag: [external] boolean;
00201       0  0
00202       1  0 [global] procedure FetchInputLine( var SourceLine: InputLine;
00203       1  0           var SourcePointer: InputPointerRange;
00204       1  0           var SourceLength: InputPointerRange);
```

```
-LINE-IDC-PL-SL-

00206       1  1 begin
00207       1  1     write( Prompt, Blank);
00208       1  1     for Loop := StartofLine to InputLineLength do
00209       1  1   SourceLine[ Loop] := Blank;
00210       1  1     SourcePointer := StartofLine;
00211       1  1     SourceLength := StartofLine;
00212       1  2     while ((not eoln) and (SourcePointer <= InputLineLength)) do begin
00213       1  2   read( InputCharacter);
00214       1  2   if (JournalingFlag = true) then write( Journal, InputCharacter);
00215       1  2   SourceLine[SourcePointer] := InputCharacter;
00216       1  2   SourceLength := SourceLength + 1;
00217       1  2   SourcePointer := SourcePointer + 1;
00218       1  1     end;
00219       1  1     readln;
00220       1  1     if (JournalingFlag = true) then writeln( Journal);
00221       1  1     SourcePointer := StartofLine;
00222       0  0 end;
00223       0  0 end.
```

```
00046       0  0     DirectionDivisor = %B'10';
00047       0  0
00048       0  0     NotPresent = -1;
00049       0  0
00050       0  0     ModeMask    = %B'11000000';
00051       0  0     ModeDivisor = %B'1000000';
00052       0  0
00053       0  0     RegMask = %B'00111000';
00054       0  0     RegDivisor =  %B'1000';
00055       0  0     RegWidth = 1;
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	262	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	8	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS FETCHINPUTLINE.PAS

  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
  /OPTIMIZE
  /STANDARD=NONE
  /TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
  /USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
  /NOANALYSIS_DATA
  /NOENVIRONMENT
  /LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FETCHINPUTLINE.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FETCHINPUTLINE.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	405	00:00.5	00:01.2
Source Analysis	249	00:00.7	00:01.6
Source Listing	17	00:00.4	00:01.1
Tree Construction	102	00:00.1	00:00.1
Flow Analysis	40	00:00.0	00:00.0
Value Propagation	11	00:00.0	00:00.0
Profit Analysis	46	00:00.0	00:00.2
Context Analysis	237	00:00.5	00:01.1
Name Packing	6	00:00.0	00:00.0
Code Selection	81	00:00.1	00:00.1
Final	73	00:00.2	00:00.6
TOTAL	1271	00:02.5	00:06.2

COMPILATION STATISTICS

```
  CPU Time:       00:02.5         (5268 Lines/Minute)
```

```
-LINE-IDC-PL-SL-

00001    C  0  0 { Title: Fetch Instruction
00002    C  0  0
00003    C  0  0   Purpose: Fetch instruction and decode opcode, register and
00004    C  0  0            effective address records for that instruction
00005    C  0  0
00006    C  0  0   Author: William A. Chapman      Date: January 12, 1986
00007    C  0  0
00008    C  0  0   Inputs: 8086 code stored in "Memory"
00009    C  0  0            Opcode Look - Up - Table (LUT)
00010    C  0  0
00011    C  0  0   Outputs: Opcode Record - completed
00012    C  0  0            Register Record - completed
00013    C  0  0            Effective Address Record - completed
00014    C  0  0            Opcode Class
00015    C  0  0
00016    C  0  0   Procedures Invoked: IntMaskShiftRight
00017    C  0  0            RMDecoder
00018    C  0  0            FetchCode
00019    C  0  0 }
00020       0  0 module FetchInstruction(input,output);
00021       0  0
00022       0  0 const
00023       0  0     %include  [-]Const.defn/list'
00024 I     0  0     FirstOpcodeValue = 0;
00025 I     0  0     LastOpcodeValue = 255;
00026 I     0  0     All16Bits = %X'FFFF';         {mask for all 16 bits}
00027 I     0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00028 I     0  0     High8Bits = %X'FF00';         {mask for high 8 bits}
00029 I     0  0     Bit8Multiplier = %X'100';
00030 I     0  0     Bit8Divisor    = %X'100';
00031 I     0  0     WordMultiplier = %X'100';
00032 I     0  0
00033 I     0  0     EightBits = 0;                      {width for 8 bits}
00034 I     0  0     SixteenBits = 1;                    {width for 16 bits}
00035 I     0  0
00036 I     0  0     SimpleMode = %B'0';
00037 I     0  0     SimpleRM = %B'110';
00038 I     0  0
00039 I     0  0     LowMemoryLimit = 0;                 {low limit on memory array}
00040 I     0  0     HighMemoryLimit = 2048;             {high limit on memory array}
00041 I     0  0
00042 I     0  0     FilenameLength = 40;
00043       0  0
00044       0  0     WidthMask = %B'1';
00045       0  0     DirectionMask = %B'10';
```

```
00046      0  0     DirectionDivisor = %B'10';
00047      0  0
00048      0  0     NotPresent = -1;
00049      0  0
00050      0  0     ModeMask   = %B'11000000';
00051      0  0     ModeDivisor = %B'1000000';
00052      0  0
00053      0  0     RegMask = %B'00111000';
00054      0  0     RegDivisor =  %B'1000';
00055      0  0     RegWidth = 1;
```

```
-LINE-IDC-PL-SL-

00056       0  0
00057       0  0     RMMask = %B'00000111';
00058       0  0     RMDivisor    =  %B'1';
00059       0  0
00060       0  0     SRMask = %B'00011000';
00061       0  0     SRDivisor  = %B'1000';
00062       0  0     SRWidth = 2;
00063       0  0
00064       0  0     AltRegMask = %B'00000111 ;
00065       0  0     AltRegDivisor     = %B'1';
00066       0  0
00067       0  0     ExtOpcodeMask = %B'00111000';
00068       0  0     ExtOpcodeDivisor  = %B'1000';
00069       0  0     ExtOpcodeMultiplier = %X'100';
00070       0  0
00071       0  0     UndefinedMode = 'U';
00072       0  0     UndefinedAddress = %X'0F0F';
00073       0  0     UndefinedSegment = %X'F0F0';
00074       0  0     UndefinedRegisterID = -1;
00075       0  0
00076       0  0     %include  [-]StopCode.defn/list'
00077 I     0  0     Breakpoint           = 'B';
00078 I     0  0     Call          = 'C';
00079 I     0  0     ControlD            = 'D';
00080 I     0  0     BadOpcode             = 'E';
00081 I     0  0     BadCEAModeValue       = 'F';
00082 I     0  0     BadRMDModeValue       = 'G';
00083 I     0  0     Halt          = 'H';
00084 I     0  0     BadRegisterID = 'I';
00085 I     0  0     BadOpcodeKey  = 'K';
00086 I     0  0     BadMemoryAddress      = 'M';
00087 I     0  0     Normal                = 'N';
00088 I     0  0     BadPortAddress        = 'P';
00089 I     0  0     BadOpcodeClass        = 'Q';
00090 I     0  0     Return                = 'R';
00091 I     0  0     BadCheckSum           = 'S';
00092 I     0  0     BadOperandType        = 'T';
00093 I     0  0     BadMemoryWidth        = 'W';
00094 I     0  0     BadOpcodeExtension    = 'X';
00095 I     0  0     NoMemoryAccess      = 'Z';
```

```
-LINE-IDC-PL-SL-

00097       0  0 type
00098       0  0     %include  [-]Type.defn/list'
00099 I     0  0     ZeroOne = -1..1;                          {defines integer with range
00100 I C   0  0                                               of zero to one with an invalid
00101 I C   0  0                                               state of -1}
00102 I     0  0
00103 I     0  0     OpcodeLUTEntry = record                   {defines entries in opcode LUT}
00104 I     0  0   OpcodeClass: char;                         {class D, A, L, S, C, P}
00105 I     0  0   OpcodeKey: integer;                        {key to interpret opcode from
00106 I C   0  0                                               0 to 7}
00107 I     0  0   DirectionBitPresent: boolean;
00108 I     0  0   WidthBitPresent: boolean
00109 I     0  0     end;
00110 I     0  0
00111 I     0  0     OpcodeType = record
00112 I     0  0   Direction: ZeroOne;                       {to or from CPU}
00113 I     0  0   Full: unsigned;                           {full opcode}
00114 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00115 I     0  0     end;
00116 I     0  0
00117 I     0  0     RegisterType = record
00118 I     0  0   Id: integer;                             {identifier}
00119 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00120 I     0  0     end;
00121 I     0  0
00122 I     0  0     EffectiveAddressType = record
00123 I     0  0   Mode: char;                              {register or memory}
00124 I     0  0   Width: ZeroOne;                          {8 or 16 bits}
00125 I     0  0   Address: unsigned;                       {memory address or
00126 I C   0  0                                               register designation}
00127 I     0  0   Segment: integer                         {segment register to use}
00128 I     0  0     end;
00129 I     0  0
00130 I     0  0     NameType = packed array[1..10] of char;
00131 I     0  0
00132 I     0  0     Characters = set of    ..'~';
00133 I     0  0     LettersAndNumbers = set of '0'..'z';
00134 I     0  0     Letters = set of 'A'..'z';
00135 I     0  0     Numbers = set of '0'..'9';
00136 I     0  0
00137 I     0  0     FileName = packed array [1..FilenameLength] of char;
```

```
-LINE-IDC-PL-SL-

00139       0  0     %include '[-]MemoryType.defn/list'
00140 I     0  0     MemoryArray = array[LowMemoryLimit..HighMemoryLimit] of unsigned;
00141       0  0
00142       0  0     %include '[-]FlagType.defn/list'
00143 I     0  0     FlagType = record                          {define processor flags}
00144 I     0  0   Carry: boolean;
00145 I     0  0   Parity: boolean;
00146 I     0  0   AuxCarry: boolean;
00147 I     0  0   Zero: boolean;
00148 I     0  0   Sign: boolean;
00149 I     0  0   Trap: boolean;
00150 I     0  0   Interrupt: boolean;
00151 I     0  0   Direction: boolean;
00152 I     0  0   Overflow: boolean
00153 I     0  0     end;
```

```
-LINE-IDC-PL-SL-

00155       0  0 var
00156       0  0     %include  [-]VarExternal.defn/list'
00157 I     0  0     OpcodeLUT: [external] array[FirstOpcodeValue..LastOpcodeValue]
00158 I     0  0           of OpcodeLUTEntry;
00159 I     0  0
00160 I     0  0     Memory: [external] MemoryArray;
00161 I     0  0
00162 I     0  0     IP: [external] unsigned;
00163 I     0  0
00164 I     0  0     Flags: [external] FlagType;
00165 I     0  0
00166 I     0  0     SegmentOverRideCount: [external] integer;
00167 I     0  0
00168 I     0  0     SegmentOverRideValue: [external] integer;
00169       0  0
00170       0  0     Width: ZeroOne;                                [8 or 16 bits]
00171       0  0     SecondByte: unsigned;                 {2nd byte of instruction}
00172       0  0     ExtOpcode: unsigned;                  {extended part of opcode}
00173       0  0
00174       0  0     Instruction: unsigned;
00175       0  0     Mode: integer;
00176       0  0     RM: integer;
00177       0  0     CS: [external] unsigned;
```

```
-LINE-IDC-PL-SL-

00179      1  0 [global] procedure FetchInstruction( var FIOpcodeClass: char;
00180      1  0           var Opcode: OpcodeType;
00181      1  0           var Register: RegisterType;
00182      1  0           var EA: EffectiveAddressType;
00183      1  0           var FIStopCode: char);
00184      1  0
00185      2  0 function IntMaskShiftRight(IMSRValue: unsigned;
00186      2  0           IMSRMask: unsigned;
00187      1  0           IMSRDivisor: integer):integer; external;
00188      1  0
00189      2  0 procedure RMDecoder(var RMDAddressMode: char;
00190      2  0           var RMDAddressValue: unsigned;
00191      2  0           var RMDSegmentId: integer;
00192      2  0           RMDModeValue: integer;
00193      2  0           RMDRMValue: integer;
00194      1  0           var RMDStopCode: char); external;
00195      1  0
00196      2  0 procedure FetchCode(var FCValue: unsigned;
00197      1  0           var FCStopCode: char); external;
```

```
-LINE-IDC-PL-SL-

00199       1  1 begin
00200       1  1
00201       1  1     FetchCode( Instruction, FIStopCode);
00202       1  1
00203       1  2     with OpcodeLUT[int(Instruction)] do begin     {access look - up - table}
00204    C  1  2   {this makes OpcodeClass, OpcodeKey, DirectionBitPresent, and
00205    C  1  2    WidthBitPresent available within the loop}
00206       1  2
00207       1  2   Opcode.Full := Instruction;     {add extended opcode later if needed}
00208       1  2
00209       1  3   if (WidthBitPresent) then begin
00210       1  3       Width := int(uand(Opcode.Full,WidthMask));
00211       1  3       Opcode.Full := uand(Opcode.Full,unot(WidthMask));
00212       1  3   end
00213       1  2   else Width := NotPresent;
00214       1  2
00215       1  2   Opcode.Width := Width;
00216       1  2   Register.Width := Width;
00217       1  2   EA.Width := Width;
00218       1  2
00219       1  3   if (DirectionBitPresent) then begin
00220       1  3       Opcode.Direction :=
00221       1  3           IntMaskShiftRight(Opcode.Full,DirectionMask,DirectionDivisor);
00222       1  3       Opcode.Full := uand(Opcode.Full,unot(DirectionMask));
00223       1  3   end
00224       1  2   else Opcode.Direction := NotPresent;
00225       1  2
00226       1  2   EA.Segment := UndefinedSegment; {flag this as undefined for now}
00227       1  2   EA.Mode := UndefinedMode;       {flag these fields}
00228       1  2   EA.Address := UndefinedAddress; {as undefined}
00229       1  2   Register.Id := UndefinedRegisterId;     {and this also}
```

```
-LINE-IDC-PL-SL-

00231       1  3   case OpcodeKey of
00232       1  3
00233       1  3       0: ;
00234       1  3
00235       1  4       1: begin {md reg R/M in 2nd byte}
00236       1  4           FetchCode( SecondByte, FIStopCode);
00237       1  4           Mode := IntMaskShiftRight(SecondByte,ModeMask,ModeDivisor);
00238       1  4           Register.Id := IntMaskShiftRight(SecondByte,RegMask,RegDivisor);
00239       1  4           RM := IntMaskShiftRight(SecondByte,RMMask,RMDivisor);
00240       1  4           RMDecoder(EA.Mode,EA.Address,EA.Segment,Mode,RM, FIStopCode);
00241       1  3         end;
00242       1  3
00243       1  4       2: begin {md xxx R/M in 2nd byte}
00244       1  4           FetchCode( SecondByte, FIStopCode);
00245       1  4           Mode := IntMaskShiftRight(SecondByte,ModeMask,ModeDivisor);
00246       1  4           RM := IntMaskShiftRight(SecondByte,RMMask,RMDivisor);
00247       1  4           RMDecoder(EA.Mode,EA.Address,EA.Segment,Mode,RM, FIStopCode);
00248       1  3         end;
00249       1  3
00250       1  4       3: begin {md 0 SR R/M in 2nd byte}
00251       1  4           FetchCode( SecondByte, FIStopCode);
00252       1  4           Mode := IntMaskShiftRight(SecondByte,ModeMask,ModeDivisor);
00253       1  4           RM := IntMaskShiftRight(SecondByte,RMMask,RMDivisor);
00254       1  4           Register.Id := IntMaskShiftRight(SecondByte,SRMask,SRDivisor);
00255       1  4           RMDecoder(EA.Mode,EA.Address,EA.Segment,Mode,RM, FIStopCode);
00256       1  4           Register.Width := SRWidth;      {for segment register}
00257       1  4           EA.Width := RegWidth;           {for R/M register
00258   C   1  4                                            Opcode Width is left undefined
00259   C   1  4                                            because of this disparity}
00260       1  3         end;
00261       1  3
00262       1  4       4: begin
00263       1  4           Register.Id := IntMaskShiftRight(Opcode.Full,SRMask,SRDivisor);
00264   C   1  4                                           {SR in bits 3 & 4 of byte one}
00265       1  4           Opcode.Full := uand(Opcode.Full,unot(SRMask));
00266       1  4           Register.Width := SRWidth;      {for Segment Register}
00267       1  4           Opcode.Width := SRWidth;        {EA Width is undefined}
00268       1  3         end;
```

```
-LINE-IDC-PL-SL-

00270       1  4       5: begin
00271       1  4           Register.Id :=
00272       1  4               IntMaskShiftRight(Opcode.Full,AltRegMask,AltRegDivisor);
00273    C  1  4                                   {reg in bits 0, 1 & 2 of byte one}
00274       1  4           Opcode.Full := uand(Opcode.Full,unot(AltRegMask));
00275       1  4           Opcode.Width := RegWidth;       {all are 16 bit operations}
00276       1  4           Register.Width := RegWidth;
00277       1  4           EA.Width := RegWidth;
00278       1  3         end;
00279       1  3
00280       1  3       6: FetchCode( SecondByte, FIStopCode); {2 byte format where 2nd byte not used}
00281       1  3
00282       1  4       7: begin {extended opcode}
00283       1  4           FetchCode( SecondByte, FIStopCode);
00284       1  4           Mode := IntMaskShiftRight(SecondByte,ModeMask,ModeDivisor);
00285       1  4           RM := IntMaskShiftRight(SecondByte,RMMask,RMDivisor);
00286       1  4           RMDecoder(EA.Mode,EA.Address,EA.Segment,Mode,RM, FIStopCode);
00287       1  4           ExtOpcode :=
00288       1  4              IntMaskShiftRight(SecondByte,ExtOpcodeMask,ExtOpcodeDivisor);
00289       1  4           Opcode.Full :=
00290       1  4              uor(Opcode.Full, uint(ExtOpcode * ExtOpcodeMultiplier));
00291       1  3         end;
00292       1  3
00293       1  3       8: {forced address}
00294       1  3           RMDecoder(EA.Mode,EA.Address,EA.Segment,SimpleMode,
00295       1  3               SimpleRM, FIStopCode);
00296       1  3
00297       1  3       otherwise FIStopCode := BadOpcodeKey;
00298       1  3
00299       1  2   end; {case Opcode Key}
00300       1  2
00301       1  2   if (SegmentOverRideCount > 0) then EA.Segment := SegmentOverRideValue;
00302       1  2
00303       1  2   FIOpcodeClass := OpcodeClass;
00304       1  2
00305       1  1     end; {with OpcodeLUT}
00306       0  0 end;
00307       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	1085	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	24	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS FETCHINSTRUCTION.PAS

  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
  /OPTIMIZE
  /STANDARD=NONE
  /TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
  /USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
  /NOANALYSIS_DATA
  /NOENVIRONMENT
  /LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FETCHINSTRUCTION.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FETCHINSTRUCTION.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	402	00:00.4	00:01.0
Source Analysis	276	00:01.1	00:04.6
Source Listing	26	00:00.6	00:02.0
Tree Construction	154	00:00.3	00:00.3
Flow Analysis	47	00:00.1	00:00.3
Value Propagation	10	00:00.0	00:00.0
Profit Analysis	38	00:00.1	00:00.1
Context Analysis	214	00:02.2	00:04.4
Name Packing	7	00:00.1	00:00.1
Code Selection	90	00:00.3	00:00.4
Final	60	00:00.4	00:01.1
TOTAL	1328	00:05.5	00:14.4

COMPILATION STATISTICS

CPU Time: 00:05.5 (3355 Lines/Minute)

```
-LINE-IDC-PL-SL-

00001    C  0  0 { Title: Fetch Memory
00002    C  0  0
00003    C  0  0   Purpose: Obtain the value of the memory location specified from
00004    C  0  0            the store "Memory"
00005    C  0  0
00006    C  0  0   Author: William A. Chapman      Date: December 23, 1984
00007    C  0  0
00008    C  0  0   Inputs: specified address value
00009    C  0  0            specified segment value
00010    C  0  0
00011    C  0  0   Outputs: returns the 8 bits of data at the specified location
00012    C  0  0
00013    C  0  0   Procedures Invoked: none
00014    C  0  0
00015    C  0  0 }
00016       0  0 module FetchMemory(input,output);
00017       0  0
00018       0  0 const
00019       0  0     %include '[-]Const.defn/list'
00020 I     0  0     FirstOpcodeValue = 0;
00021 I     0  0     LastOpcodeValue = 255;
00022 I     0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00023 I     0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00024 I     0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00025 I     0  0     Bit8Multiplier = %X'100';
00026 I     0  0     Bit8Divisor    = %X'100';
00027 I     0  0     WordMultiplier = %X'100';
00028 I     0  0
00029 I     0  0     EightBits = 0;                        {width for 8 bits}
00030 I     0  0     SixteenBits = 1;                      {width for 16 bits}
00031 I     0  0
00032 I     0  0     SimpleMode = %B'0';
00033 I     0  0     SimpleRM = %B'110';
00034 I     0  0
00035 I     0  0     LowMemoryLimit = 0;                   {low limit on memory array}
00036 I     0  0     HighMemoryLimit = 2048;               {high limit on memory array}
00037 I     0  0
00038 I     0  0     FilenameLength = 40;
00039       0  0
00040       0  0     %include  [-]Byte.defn/list'
00041 I     0  0     Byte0Mask = %X'000000FF';
00042 I     0  0     Byte1Mask = %X'0000FF00';
00043 I     0  0     Byte2Mask = %X'00FF0000';
00044 I     0  0     Byte3Mask = %X'FF000000';
00045 I     0  0
```

```
00046 I    0  0     Byte0Divisor = %X'00000001 ;
00047 I    0  0     Byte1Divisor = %X'00000100';
00048 I    0  0     Byte2Divisor = %X'00010000';
00049 I    0  0     Byte3Divisor = %X'01000000';
00050 I    0  0
00051 I    0  0     Byte0Multiplier = %X'00000001 ;
00052 I    0  0     Byte1Multiplier = %X'00000100';
00053 I    0  0     Byte2Multiplier = %X'00010000';
00054 I    0  0     Byte3Multiplier = %X'01000000';
00055 I    0  0
```

```
-LINE-IDC-PL-SL-

00056 I     0  0     BytesPerWord = 4;
00057       0  0
00058       0  0     %include  [-]StopCode.defn/list'
00059 I     0  0     Breakpoint              = 'B';
00060 I     0  0     Call            = 'C';
00061 I     0  0     ControlD              = 'D';
00062 I     0  0     BadOpcode               = 'E';
00063 I     0  0     BadCEAModeValue         = 'F';
00064 I     0  0     BadRMDModeValue         = 'G';
00065 I     0  0     Halt            = 'H';
00066 I     0  0     BadRegisterID = 'I';
00067 I     0  0     BadOpcodeKey  = 'K';
00068 I     0  0     BadMemoryAddress        = 'M';
00069 I     0  0     Normal                  = 'N';
00070 I     0  0     BadPortAddress          = 'P';
00071 I     0  0     BadOpcodeClass          = 'Q';
00072 I     0  0     Return                  = 'R';
00073 I     0  0     BadCheckSum             = 'S';
00074 I     0  0     BadOperandType          = 'T';
00075 I     0  0     BadMemoryWidth          = 'W';
00076 I     0  0     BadOpcodeExtension      = 'X';
00077 I     0  0     NoMemoryAccess        = 'Z';
00078       0  0
00079       0  0     NullValue = 0;
```

```
-LINE-IDC-PL-SL-

00081       0  0 type
00082       0  0     %include  [-]Type.defn/list'
00083 I     0  0     ZeroOne = -1..1;                          {defines integer with range
00084 I C   0  0                                               of zero to one with an invalid
00085 I C   0  0                                               state of -1}
00086 I     0  0
00087 I     0  0     OpcodeLUTEntry = record                   {defines entries in opcode LUT}
00088 I     0  0   OpcodeClass: char;                         {class D, A, L, S, C, P}
00089 I     0  0   OpcodeKey: integer;                        {key to interpret opcode from
00090 I C   0  0                                               0 to 7}
00091 I     0  0   DirectionBitPresent: boolean;
00092 I     0  0   WidthBitPresent: boolean
00093 I     0  0     end;
00094 I     0  0
00095 I     0  0     OpcodeType = record
00096 I     0  0   Direction: ZeroOne;                        {to or from CPU}
00097 I     0  0   Full: unsigned;                            {full opcode}
00098 I     0  0   Width: ZeroOne                            {8 or 16 bits}
00099 I     0  0     end;
00100 I     0  0
00101 I     0  0     RegisterType = record
00102 I     0  0   Id: integer;                              {identifier}
00103 I     0  0   Width: ZeroOne                            {8 or 16 bits}
00104 I     0  0     end;
00105 I     0  0
00106 I     0  0     EffectiveAddressType = record
00107 I     0  0   Mode: char;                               {register or memory}
00108 I     0  0   Width: ZeroOne;                           {8 or 16 bits}
00109 I     0  0   Address: unsigned;                        {memory address or
00110 I C   0  0                                               register designation}
00111 I     0  0   Segment: integer                          {segment register to use}
00112 I     0  0     end;
00113 I     0  0
00114 I     0  0     NameType = packed array[1..10] of char;
00115 I     0  0
00116 I     0  0     Characters = set of    ..'~';
00117 I     0  0     LettersAndNumbers = set of '0'..'z';
00118 I     0  0     Letters = set of 'A'..'z';
00119 I     0  0     Numbers = set of '0'..'9';
00120 I     0  0
00121 I     0  0     FileName = packed array [1..FilenameLength] of char;
00122       0  0
00123       0  0     %include  [-]MemoryType.defn/list'
00124 I     0  0     MemoryArray = array[LowMemoryLimit..HighMemoryLimit] of unsigned;
```

```
00125     0  0
00126     0  0
00127     0  0
00128     0  0 var
00129     0  0
00130     0  0     Memory: [external] MemoryArray;
00131     0  0     Address: integer;                      {pointer into memory array}
00132     0  0     WordPointer: integer;
00133     0  0     BytePointer: integer;
00134     0  0
00135     0  0     Continue: [external] LettersAndNumbers;
```

```
-LINE-IDC-PL-SL-

00137       1  0 [global] procedure FetchMemory(FMAddress: unsigned;
00138       1  0           FMSegment: unsigned;
00139       1  0           var FMValue: unsigned;
00140       1  0           var FMStopCode: char);
00141       1  0
00142       2  0 function MaskShiftRight(MSRValue: unsigned;
00143       2  0           MSRMask: unsigned;
00144       1  0           MSRDivisor: integer): unsigned; external;
00145       1  0
00146       1  0
00147       1  0
00148       1  0
00149       1  1 begin
00150       1  1     FMValue := NullValue;
00151       1  1     Address := int(FMAddress + (FMSegment * 16)); {form array index]
00152       1  1     if ((Address < (LowMemoryLimit * BytesPerWord))
00153       1  1   or (Address > ((HighMemoryLimit * BytesPerWord) + 1))) then
00154       1  1       FMStopCode := BadMemoryAddress
00155       1  2     else begin
00156       1  2
00157       1  2   WordPointer := Address div BytesPerWord;
00158       1  2   BytePointer := Address rem BytesPerWord;
00159       1  2
00160       1  3   case BytePointer of
00161       1  3
00162       1  3   0: FMValue := MaskShiftRight(Memory[WordPointer], Byte0Mask, Byte0Divisor);
00163       1  3
00164       1  3   1: FMValue := MaskShiftRight(Memory[WordPointer], Byte1Mask, Byte1Divisor);
00165       1  3
00166       1  3   2: FMValue := MaskShiftRight(Memory[WordPointer], Byte2Mask, Byte2Divisor);
00167       1  3
00168       1  3   3: FMValue := MaskShiftRight(Memory[WordPointer], Byte3Mask, Byte3Divisor);
00169       1  3
00170       1  3   otherwise FMStopCode := BadMemoryAddress;
00171       1  3
00172       1  2   end; {case BytePointer]
00173       1  2
00174       1  1     end;
00175       1  1
00176       0  0 end; {procedure FetchMemory]
00177       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	304	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	12	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS FETCHMEMORY.PAS

  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
  /OPTIMIZE
  /STANDARD=NONE
  /TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
  /USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
  /NOANALYSIS_DATA
  /NOENVIRONMENT
  /LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FETCHMEMORY.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FETCHMEMORY.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	407	00:00.4	00:03.2
Source Analysis	249	00:00.7	00:04.5
Source Listing	14	00:00.4	00:02.7
Tree Construction	109	00:00.1	00:00.2
Flow Analysis	54	00:00.1	00:00.1
Value Propagation	10	00:00.0	00:00.0
Profit Analysis	27	00:00.0	00:00.0
Context Analysis	203	00:00.5	00:00.9
Name Packing	6	00:00.0	00:00.0
Code Selection	87	00:00.1	00:00.6
Final	66	00:00.2	00:00.6
TOTAL	1236	00:02.5	00:12.8

COMPILATION STATISTICS

```
  CPU Time:       00:02.5         (4198 Lines/Minute)
```

```
-LINE-IDC-PL-SL-

00001    C  0  0 { Title: Fetch Operand module
00002    C  0  0
00003    C  0  0   Purpose: Obtain the operand specified from the store "Memory"
00004    C  0  0           or "Registers, Pointers, Flags"
00005    C  0  0
00006    C  0  0   Author: William A. Chapman      Date: December 24, 1985
00007    C  0  0
00008    C  0  0   Inputs: Type of operand - Register (R), or Memory (M)
00009    C  0  0           Width - 8 or 16 bits
00010    C  0  0           Address for Memory or Register Designator for register
00011    C  0  0           Segment - Register Designator for the segment register
00012    C  0  0           to be used
00013    C  0  0
00014    C  0  0   Outputs: The 8 or 16 bit value of the variable requested
00015    C  0  0
00016    C  0  0   Procedures Invoked: Fetch RegPtr
00017    C  0  0           Fetch Memory
00018    C  0  0 }
00019       0  0 module FetchOperand(output);
00020       0  0
00021       0  0 const
00022       0  0     %include  [-]Const.defn/list'
00023  I    0  0     FirstOpcodeValue = 0;
00024  I    0  0     LastOpcodeValue = 255;
00025  I    0  0     All16Bits = %X'FFFF';         {mask for all 16 bits}
00026  I    0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00027  I    0  0     High8Bits = %X'FF00';         {mask for high 8 bits}
00028  I    0  0     Bit8Multiplier = %X'100';
00029  I    0  0     Bit8Divisor    = %X'100';
00030  I    0  0     WordMultiplier = %X'100';
00031  I    0  0
00032  I    0  0     EightBits = 0;                        {width for 8 bits}
00033  I    0  0     SixteenBits = 1;                      {width for 16 bits}
00034  I    0  0
00035  I    0  0     SimpleMode = %B'0';
00036  I    0  0     SimpleRM = %B'110';
00037  I    0  0
00038  I    0  0     LowMemoryLimit = 0;                   {low limit on memory array}
00039  I    0  0     HighMemoryLimit = 2048;               {high limit on memory array}
00040  I    0  0
00041  I    0  0     FilenameLength = 40;
00042       0  0
00043       0  0     MemoryPointer = 2;                    {width variable to access
00044    C  0  0                                      segment registers}
00045       0  0     %include  [-]StopCode.defn/list'
```

```
00046 I     0   0     Breakpoint              = 'B';
00047 I     0   0     Call          = 'C';
00048 I     0   0     ControlD              = 'D';
00049 I     0   0     BadOpcode               = 'E';
00050 I     0   0     BadCEAModeValue         = 'F';
00051 I     0   0     BadRMDModeValue         = 'G';
00052 I     0   0     Halt          = 'H';
00053 I     0   0     BadRegisterID = 'I';
00054 I     0   0     BadOpcodeKey  = 'K';
00055 I     0   0     BadMemoryAddress        = 'M';
```

```
-LINE-IDC-PL-SL-

00056 I    0  0     Normal                = 'N';
00057 I    0  0     BadPortAddress        = 'P';
00058 I    0  0     BadOpcodeClass        = 'Q';
00059 I    0  0     Return                = 'R';
00060 I    0  0     BadCheckSum           = 'S';
00061 I    0  0     BadOperandType        = 'T';
00062 I    0  0     BadMemoryWidth        = 'W';
00063 I    0  0     BadOpcodeExtension    = 'X';
00064 I    0  0     NoMemoryAccess      = 'Z';
00065      0  0
00066      0  0 var
00067      0  0     LowByte: unsigned;                  {low 8 bits of 16}
00068      0  0     HighByte: unsigned;                 {high 8 bits of 16}
00069      0  0     Segment: unsigned;                  {value of the desired segment register}
00070      0  0
00071      0  0
00072      0  0
00073      0  0
00074      1  0 [global] procedure FetchOperand(FOType: char;
00075      1  0           FOWidth: integer;
00076      1  0           FOAddress: unsigned;
00077      1  0           FOSegment: integer;
00078      1  0           var FOValue: unsigned;
00079      1  0           var FOStopCode: char);
00080      1  0
00081      2  0 procedure FetchRegPtr(FRPWidth: integer;
00082      2  0           FRPDesignator:integer;
00083      2  0           var FRPValue: unsigned;
00084      1  0           var FRPStopCode: char); external;
00085      1  0
00086      2  0 procedure FetchMemory(FMAddress: unsigned;
00087      2  0           FMSegment: unsigned;
00088      2  0           var FMValue: unsigned;
00089      1  0           var FMStopCode: char); external;
```

```
-LINE-IDC-PL-SL-

00091       1  1 begin
00092       1  2     case FOType of        {sort Register from Memory accesses}
00093       1  2
00094       1  2   'r','R': FetchRegPtr( FOWidth, int(FOAddress), FOValue, FOStopCode);
00095       1  2
00096       1  3   'm','M': begin
00097       1  3       FetchRegPtr(MemoryPointer, FOSegment, Segment, FOStopCode);
00098   C   1  3                   {get value of desired segment register}
00099       1  4       case FOWidth of     {sort into 8 or 16 bits}
00100       1  4
00101       1  4           0: FetchMemory(FOAddress,Segment,FOValue, FOStopCode);
00102   C   1  4                   {get 8 bits directly}
00103       1  5           1: begin
00104       1  5               FetchMemory( FOAddress, Segment, LowByte, FOStopCode); {get low byte}
00105       1  5               FetchMemory(( FOAddress + 1), Segment, HighByte, FOStopCode);
00106   C   1  5                   {get high byte}
00107       1  5               FOValue := uor(LowByte,uand((HighByte * 256),High8Bits));
00108   C   1  5                   {position the high byte, "or" in the low byte}
00109       1  5               FOValue := uand(FOValue,All16Bits); {limit it to 16 bits}
00110       1  5               end {case 1}
00111       1  5
00112       1  4           otherwise FOStopCode := BadMemoryWidth;
00113       1  4
00114       1  3       end; {case FOWidth}
00115       1  3   end {case "M"}
00116       1  3
00117       1  2   otherwise FOStopCode := BadOperandType;
00118       1  2
00119       1  1     end; {case FOType}
00120       0  0 end;
00121       0  0 end. {module}
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	313	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	12	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS FETCHOPERAND.PAS

  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
  /OPTIMIZE
  /STANDARD=NONE
  /TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
  /USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
  /NOANALYSIS_DATA
  /NOENVIRONMENT
  /LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FETCHOPERAND.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FETCHOPERAND.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	403	00:00.5	00:04.9
Source Analysis	225	00:00.4	00:02.9
Source Listing	14	00:00.2	00:01.8
Tree Construction	97	00:00.1	00:00.2
Flow Analysis	51	00:00.1	00:00.1
Value Propagation	9	00:00.0	00:00.0
Profit Analysis	42	00:00.1	00:00.1
Context Analysis	147	00:00.5	00:01.0
Name Packing	7	00:00.0	00:00.0
Code Selection	73	00:00.1	00:00.1
Final	59	00:00.1	00:00.4
TOTAL	1132	00:02.0	00:11.5

COMPILATION STATISTICS

```
  CPU Time:       00:02.0         (3576 Lines/Minute)
```

```
-LINE-IDC-PL-SL-

00001   C  0  0 { Title: Fetch Registers and Pointers
00002   C  0  0
00003   C  0  0   Purpose: Obtain the value of the register or pointer
00004   C  0  0            specified from the store "Registers Pointers"
00005   C  0  0
00006   C  0  0   Author: William A. Chapman      Date: December 19, 1985
00007   C  0  0
00008   C  0  0   Inputs: Width - indicating 8 or 16 bits
00009   C  0  0            Designator - indicating which register or pointer
00010   C  0  0                    is to be accessed
00011   C  0  0
00012   C  0  0   Outputs: Value - the value of the desired register or pointer
00013   C  0  0
00014   C  0  0   Functions Invoked: MaskShiftRight
00015   C  0  0 }
00016      0  0 module FetchRegPtr(output);
00017      0  0
00018      0  0 const
00019      0  0     %include  [-]Const.defn/list'
00020 I    0  0     FirstOpcodeValue = 0;
00021 I    0  0     LastOpcodeValue = 255;
00022 I    0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00023 I    0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00024 I    0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00025 I    0  0     Bit8Multiplier = %X'100';
00026 I    0  0     Bit8Divisor    = %X'100';
00027 I    0  0     WordMultiplier = %X'100';
00028 I    0  0
00029 I    0  0     EightBits = 0;                        {width for 8 bits}
00030 I    0  0     SixteenBits = 1;                      {width for 16 bits}
00031 I    0  0
00032 I    0  0     SimpleMode = %B'0';
00033 I    0  0     SimpleRM = %B'110';
00034 I    0  0
00035 I    0  0     LowMemoryLimit = 0;                   {low limit on memory array}
00036 I    0  0     HighMemoryLimit = 2048;               {high limit on memory array}
00037 I    0  0
00038 I    0  0     FilenameLength = 40;
00039      0  0
00040      0  0     HighByteDivisor = %X'100';
00041      0  0
00042      0  0     %include  [-]RegID.defn/list'
00043 I    0  0     ALId = %B'000';
00044 I    0  0     ALWidth = 0;
00045 I    0  0
```

```
00046 I     0   0      CLId = %B'001';
00047 I     0   0      CLWidth = 0;
00048 I     0   0
00049 I     0   0      DLId = %B'010';
00050 I     0   0      DLWidth = 0;
00051 I     0   0
00052 I     0   0      BLId = %B'011';
00053 I     0   0      BLWidth = 0;
00054 I     0   0
00055 I     0   0      AHId = %B'100';
```

```
-LINE-IDC-PL-SL-

00056 I     0  0      AHWidth = 0;
00057 I     0  0
00058 I     0  0      CHId = %B'101 ;
00059 I     0  0      CHWidth = 0;
00060 I     0  0
00061 I     0  0      DHId = %B'110';
00062 I     0  0      DHWidth = 0;
00063 I     0  0
00064 I     0  0      BHId = %B'111';
00065 I     0  0      BHWidth = 0;
00066 I     0  0
00067 I     0  0      AXId = %B'000';
00068 I     0  0      AXWidth = 1;
00069 I     0  0      ALorAXId = %B'000';
00070 I     0  0
00071 I     0  0      CXId = %B'001';
00072 I     0  0      CXWidth = 1;
00073 I     0  0
00074 I     0  0      DXId = %B'010';
00075 I     0  0      DXWidth = 1;
00076 I     0  0
00077 I     0  0      BXId = %B'011';
00078 I     0  0      BXWidth = 1;
00079 I     0  0
00080 I     0  0      SPId = %B'100';
00081 I     0  0      SPWidth = 1;
00082 I     0  0
00083 I     0  0      BPId = %B'101 ;
00084 I     0  0      BPWidth = 1;
00085 I     0  0
00086 I     0  0      SIId = %B'110';
00087 I     0  0      SIWidth = 1;
00088 I     0  0
00089 I     0  0      DIId = %B'111';
00090 I     0  0      DIWidth = 1;
00091 I     0  0
00092 I     0  0      SRWidth = 2;
00093 I     0  0
00094 I     0  0      ESId = %B'00';
00095 I     0  0      ESWidth = 2;
00096 I     0  0
00097 I     0  0      CSId = %B'01';
00098 I     0  0      CSWidth = 2;
00099 I     0  0
```

```
00100 I     0  0      SSId = %B'10';
00101 I     0  0      SSWidth = 2;
00102 I     0  0
00103 I     0  0      DSId = %B'11';
00104 I     0  0      DSWidth = 2;
00105 I     0  0
00106       0  0
00107       0  0      %include '[-]StopCode.defn/list'
00108 I     0  0      Breakpoint              = 'B';
00109 I     0  0      Call          = 'C';
00110 I     0  0      ControlD                = 'D';
```

```
-LINE-IDC-PL-SL-

00111 I     0  0     BadOpcode              = 'E';
00112 I     0  0     BadCEAModeValue        = 'F';
00113 I     0  0     BadRMDModeValue        = 'G';
00114 I     0  0     Halt          = 'H';
00115 I     0  0     BadRegisterID = 'I';
00116 I     0  0     BadOpcodeKey  = 'K';
00117 I     0  0     BadMemoryAddress       = 'M';
00118 I     0  0     Normal                 = 'N';
00119 I     0  0     BadPortAddress         = 'P';
00120 I     0  0     BadOpcodeClass         = 'Q';
00121 I     0  0     Return                 = 'R';
00122 I     0  0     BadCheckSum            = 'S';
00123 I     0  0     BadOperandType         = 'T';
00124 I     0  0     BadMemoryWidth         = 'W';
00125 I     0  0     BadOpcodeExtension     = 'X';
00126 I     0  0     NoMemoryAccess       = 'Z';
00127       0  0
00128       0  0
00129       0  0
00130       0  0 var
00131       0  0     %include  [-]Regexternal.defn/list'
00132 I     0  0     AX: [external] unsigned;      {value of the AX register}
00133 I     0  0     CX: [external] unsigned;      {value of the CX register}
00134 I     0  0     DX: [external] unsigned;      {value of the DX register}
00135 I     0  0     BX: [external] unsigned;      {value of the BX register}
00136 I     0  0     SP: [external] unsigned;      {value of the SP register}
00137 I     0  0     BP: [external] unsigned;      {value of the BP register}
00138 I     0  0     SI: [external] unsigned;      {value of the SI register}
00139 I     0  0     DI: [external] unsigned;      {value of the DI register}
00140 I     0  0
00141 I     0  0     ES: [external] unsigned;      {value of the ES register}
00142 I     0  0     CS: [external] unsigned;      {value of the CS register}
00143 I     0  0     SS: [external] unsigned;      {value of the SS register}
00144 I     0  0     DS: [external] unsigned;      {value of the DS register}
00145 I     0  0
00146       0  0
00147       0  0
00148       0  0
00149       1  0 [global] procedure FetchRegPtr(FRPWidth: integer;
00150       1  0           FRPDesignator: integer;
00151       1  0           var FRPValue: unsigned;
00152       1  0           var FRPStopCode: char);
00153       1  0
00154       2  0 function MaskShiftRight(MSRValue: unsigned;
```

```
00155      2  0          MSRMask: unsigned;
00156      1  0          MSRDivisor: integer): unsigned; external;
```

```
-LINE-IDC-PL-SL-

00158      1  1 begin
00159      1  2     case FRPWidth of
00160      1  2
00161      1  3   0:  case FRPDesignator of       {8 bit registers}
00162      1  3           ALId: FRPValue := uand( AX, Low8Bits);
00163      1  3           CLId: FRPValue := uand( CX, Low8Bits);
00164      1  3           DLId: FRPValue := uand( DX, Low8Bits);
00165      1  3           BLId: FRPValue := uand( BX, Low8Bits);
00166      1  3           AHId: FRPValue := MaskShiftRight( AX, High8Bits, HighByteDivisor);
00167      1  3           CHId: FRPValue := MaskShiftRight( CX, High8Bits, HighByteDivisor);
00168      1  3           DHId: FRPValue := MaskShiftRight( DX, High8Bits, HighByteDivisor);
00169      1  3           BHId: FRPValue := MaskShiftRight( BX, High8Bits, HighByteDivisor);
00170      1  3           otherwise FRPStopCode := BadRegisterID;
00171      1  2       end; {case 0 - 8 bit registers}
00172      1  2
00173      1  3   1:  case FRPDesignator of       {16 bit registers}
00174      1  3           AXId: FRPValue := uand( AX, All16Bits);
00175      1  3           CXId: FRPValue := uand( CX, All16Bits);
00176      1  3           DXId: FRPValue := uand( DX, All16Bits);
00177      1  3           BXId: FRPValue := uand( BX, All16Bits);
00178      1  3           SPId: FRPValue := uand( SP, All16Bits);
00179      1  3           BPId: FRPValue := uand( BP, All16Bits);
00180      1  3           SIId: FRPValue := uand( SI, All16Bits);
00181      1  3           DIId: FRPValue := uand( DI, All16Bits)
00182      1  3           otherwise FRPStopCode := BadRegisterID;
00183      1  2       end; {case 1 - 16 bit registers}
00184      1  2
00185      1  3         2: case FRPDesignator of  {16 bit pointers}
00186      1  3           ESId: FRPValue := uand( ES, All16Bits);
00187      1  3           CSId: FRPValue := uand( CS, All16Bits);
00188      1  3           SSId: FRPValue := uand( SS, All16Bits);
00189      1  3           DSId: FRPValue := uand( DS, All16Bits)
00190      1  3           otherwise FRPStopCode := BadRegisterID;
00191      1  3       end {case 2 - Pointers}
00192      1  3
00193      1  2   otherwise FRPStopCode := BadRegisterID;
00194      1  2
00195      1  1     end; {case FRPWidth}
00196      0  0 end; {procedure FetchRegPtr}
00197      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	488	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
PAS/LIS FETCHREGPTR.PAS

/CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
/DEBUG=(NOSYMBOLS,TRACEBACK)
/SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
/OPTIMIZE
/STANDARD=NONE
/TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
/USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
/NOANALYSIS_DATA
/NOENVIRONMENT
/LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FETCHREGPTR.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FETCHREGPTR.OBJ;1
/NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	404	00:00.5	00:04.4
Source Analysis	237	00:00.7	00:04.0
Source Listing	16	00:00.4	00:02.7
Tree Construction	115	00:00.1	00:00.3
Flow Analysis	32	00:00.1	00:00.8
Value Propagation	9	00:00.0	00:00.0
Profit Analysis	26	00:00.1	00:00.1
Context Analysis	151	00:00.9	00:01.7
Name Packing	6	00:00.0	00:00.0
Code Selection	70	00:00.1	00:00.1
Final	63	00:00.2	00:00.7
TOTAL	1133	00:03.1	00:14.8

COMPILATION STATISTICS

```
CPU Time:       00:03.1         (3850 Lines/Minute)
Elapsed Time:   00:14.8
```

```
0001    C Title: Exists
0002    C
0003    C Purpose: Inquire if a file exists
0004    C
0005    C Author: William A. Chapman      Date: December 14, 1986
0006    C
0007    C Inputs: Filename
0008    C
0009    C Outputs: True if file exists
0010    C         False otherwise
0011    C
0012    C Procedures Invoked: none
0013    C
0014    C Compliments of: J. Schueckler
0015    C
0016
0017      Logical Function FileExists(NAME)
0018
0019      Character Name*40
0020      Logical*1 Existence
0021
0022      Inquire( FILE=NAME, Exist=Existence )
0023      FileExists = Existence
0024
0025      Return
0026      End
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	30	PIC CON REL LCL SHR EXE RD NOWRT LONG
2	$LOCAL	36	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
	Total Space Allocated	66	

ENTRY POINTS

Address	Type	Name
0-00000000	L*4	FILEEXISTS

VARIABLES

Address	Type	Name	Address	Type	Name
2-00000000	L*1	EXISTENCE	AP-00000004@	CHAR	NAME

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name
	FOR$INQUIRE

COMMAND QUALIFIERS

```
FOR/LIS FILEEXISTS.FOR

/CHECK=(NOBOUNDS,OVERFLOW,NOUNDERFLOW)
/DEBUG=(NOSYMBOLS,TRACEBACK)
/STANDARD=(NOSYNTAX,NOSOURCE_FORM)
/SHOW=(NOPREPROCESSOR,NOINCLUDE,MAP,NODICTIONARY,SINGLE)
/WARNINGS=(GENERAL,NODECLARATIONS,NOULTRIX)
/CONTINUATIONS=19  /NOCROSS_REFERENCE  /NOD_LINES  /NOEXTEND_SOURCE  /F77
/NOG_FLOATING  /I4  /NOMACHINE_CODE  /OPTIMIZE  /NOANALYSIS
```

COMPILATION STATISTICS

```
Run Time:           0.70 seconds
Elapsed Time:       3.34 seconds
Page Faults:        706
Dynamic Memory:     566 pages
```

```
-LINE-IDC-PL-SL-

00001       0  0 module Functions(input,output);                    {a collection of functions}
00002    C  0  0 {
00003    C  0  0   Title: MaskShiftRight
00004    C  0  0
00005    C  0  0   Purpose: Mask off the specified bits and shift them right
00006    C  0  0            per the specified count
00007    C  0  0
00008    C  0  0   Author: William A. Chapman      Date: January 8, 1986
00009    C  0  0
00010    C  0  0   Inputs: Value: to be shifted (unsigned)
00011    C  0  0           Mask: to mask off unwanted bits (unsigned)
00012    C  0  0           Divisor: amount to shift (integer)
00013    C  0  0
00014    C  0  0   Outputs: Value: masked and shifted as specified (unsigned)
00015    C  0  0
00016    C  0  0   Procedures Invoked: none
00017    C  0  0 }
00018       1  0 [global] function MaskShiftRight(Value: unsigned;
00019       1  0                                  Mask: unsigned;
00020       1  0                                  Divisor: integer): unsigned;
00021       1  0
00022       1  1 begin
00023       1  1
00024       1  1     MaskShiftRight := utrunc(uand(Value,Mask) / Divisor);
00025       1  1
00026       0  0 end; {MaskShiftRight}
```

```
-LINE-IDC-PL-SL-

00028   C  0  0 {
00029   C  0  0   Title: IntMaskShiftRight
00030   C  0  0
00031   C  0  0   Purpose: Convert the masked, shifted number to an integer
00032   C  0  0
00033   C  0  0   Author: William A. Chapman      Date: January 8, 1986
00034   C  0  0
00035   C  0  0   Inputs: Value: to be shifted (unsigned)
00036   C  0  0           Mask: mask to be used (unsigned)
00037   C  0  0           Divisor: amount to shift (integer)
00038   C  0  0
00039   C  0  0   Outputs: Shifted value is converted to an integer
00040   C  0  0
00041   C  0  0   Procedures Invoked: MaskShiftRight
00042   C  0  0 }
00043      1  0 [global] function IntMaskShiftRight(Value: unsigned;
00044      1  0                                     Mask: unsigned;
00045      1  0                                     Divisor: integer): integer;
00046      1  0
00047      1  1 begin
00048      1  1
00049      1  1     IntMaskShiftRight := int( MaskShiftRight(Value,Mask,Divisor));
00050      1  1
00051      0  0 end; {IntMaskShiftRight}
```

```
-LINE-IDC-PL-SL-

00053    C  0  0 {
00054    C  0  0   Title: SignExtend7
00055    C  0  0
00056    C  0  0   Purpose: Sign extend to 32 bits the number specified
00057    C  0  0            based on the state of bit 7
00058    C  0  0
00059    C  0  0   Author: William A. Chapman      Date: January 8, 1986
00060    C  0  0
00061    C  0  0   Inputs: Value: to be extended (unsigned)
00062    C  0  0
00063    C  0  0   Outputs: A 32 bit integer equivalent of the input number
00064    C  0  0
00065    C  0  0   Procedures Invoked: none
00066    C  0  0 }
00067       1  0 [global] function SignExtend7(Value: unsigned): integer;
00068       1  0
00069       1  0 const
00070       1  0     Bit7Mask = %B'10000000';
00071       1  0     HighByteOn = %X'FFFFFF00';
00072       1  0
00073       1  1 begin
00074       1  1
00075       1  1     if (uand(Value,Bit7Mask) = 0) then SignExtend7 := int(Value)
00076       1  1     else SignExtend7 := int(uor(Value,HighByteOn));
00077       1  1
00078       0  0 end; {SignExtend7}
```

```
-LINE-IDC-PL-SL-

00080    C  0  0 {
00081    C  0  0   Title: SignExtend15
00082    C  0  0
00083    C  0  0   Purpose: Sign extend to 32 bits the number specified
00084    C  0  0           based on the state of bit 15
00085    C  0  0
00086    C  0  0   Author: William A. Chapman      Date: January 8, 1986
00087    C  0  0
00088    C  0  0   Inputs: Value: to be extended (unsigned)
00089    C  0  0
00090    C  0  0   Outputs: A 32 bit integer equivalent of the input number
00091    C  0  0
00092    C  0  0   Procedures Invoked: none
00093    C  0  0 }
00094       1  0 [global] function SignExtend15(Value: unsigned): integer;
00095       1  0
00096       1  0 const
00097       1  0     Bit15Mask = %X'8000';
00098       1  0     HighWordOn = %X'FFFF0000';
00099       1  0
00100       1  1 begin
00101       1  1
00102       1  1     if (uand(Value,Bit15Mask) = 0) then SignExtend15 := int(Value)
00103       1  1     else SignExtend15 := int(uor(Value,HighWordOn));
00104       1  1
00105       0  0 end; [SignExtend15]
00106       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	185	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS FUNCTIONS.PAS

  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
  /OPTIMIZE
  /STANDARD=NONE
  /TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
  /USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
  /NOANALYSIS_DATA
  /NOENVIRONMENT
  /LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FUNCTIONS.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]FUNCTIONS.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	403	00:00.5	00:05.5
Source Analysis	234	00:00.2	00:00.7
Source Listing	13	00:00.2	00:01.8
Tree Construction	108	00:00.1	00:00.3
Flow Analysis	38	00:00.1	00:00.1
Value Propagation	10	00:00.0	00:00.0
Profit Analysis	45	00:00.1	00:00.2
Context Analysis	162	00:00.6	00:01.7
Name Packing	7	00:00.0	00:00.0
Code Selection	100	00:00.1	00:00.4
Final	60	00:00.1	00:00.8
TOTAL	1184	00:02.0	00:11.9

COMPILATION STATISTICS

```
  CPU Time:       00:02.0         (3164 Lines/Minute)
  Elapsed Time:   00:11.9
```

```
-LINE-IDC-PL-SL-

00001    C  0  0 { Title: GetAddress
00002    C  0  0
00003    C  0  0   Purpose: Isolate segment and address values from the input stream
00004    C  0  0
00005    C  0  0   Author: William A. Chapman      Date: August 27, 1986
00006    C  0  0
00007    C  0  0   Inputs: OperatorInput
00008    C  0  0           Current input pointer
00009    C  0  0           Length of input
00010    C  0  0
00011    C  0  0   Outputs: Segment value
00012    C  0  0           Address value
00013    C  0  0
00014    C  0  0   Procedures Invoked: SyntaxInterpreter
00015    C  0  0           FetchRegPtr
00016    C  0  0 ]
00017       0  0 module GetAddress( input, output);
00018       0  0
00019       0  0 const
00020       0  0     %include  [-]Const.defn/list'
00021 I     0  0     FirstOpcodeValue = 0;
00022 I     0  0     LastOpcodeValue = 255;
00023 I     0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00024 I     0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00025 I     0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00026 I     0  0     Bit8Multiplier = %X'100';
00027 I     0  0     Bit8Divisor    = %X'100';
00028 I     0  0     WordMultiplier = %X'100';
00029 I     0  0
00030 I     0  0     EightBits = 0;                        {width for 8 bits}
00031 I     0  0     SixteenBits = 1;                      {width for 16 bits}
00032 I     0  0
00033 I     0  0     SimpleMode = %B'0';
00034 I     0  0     SimpleRM = %B'110';
00035 I     0  0
00036 I     0  0     LowMemoryLimit = 0;                   {low limit on memory array}
00037 I     0  0     HighMemoryLimit = 2048;               {high limit on memory array}
00038 I     0  0
00039 I     0  0     FilenameLength = 40;
```

```
-LINE-IDC-PL-SL-

00041       0  0     %include '[-]DebugConst.defn/list'
00042 I     0  0     FirstKeywordEntry = 0;
00043 I     0  0     LastKeywordEntry  = 100;
00044 I     0  0     UndefinedName = 'Qqqqqqqq';
00045 I     0  0     UndefinedClass = 'z';
00046 I     0  0     UndefinedWidth = -1;
00047 I     0  0     UndefinedID = -2;
00048 I     0  0     PageSize = 55;
00049 I     0  0     FF = 12;
00050 I     0  0     MinID = -1;
00051 I     0  0     MaxID = 25;
00052 I     0  0     MinWidth = 0;
00053 I     0  0     MaxWidth = 3;
00054 I     0  0     SegmentOffset = 16;
00055 I     0  0
00056 I     0  0     BlankClass       =    ;
00057 I     0  0     BreakpointClass  = 'B';
00058 I     0  0     ErrorClass       = 'E';
00059 I     0  0     FlagClass        = 'F';
00060 I     0  0     InputOutputClass = 'I';
00061 I     0  0     MemoryClass      = 'M';
00062 I     0  0     NumericClass     = 'N';
00063 I     0  0     OperatorClass    = 'O';
00064 I     0  0     QualifierClass   = 'Q';
00065 I     0  0     RegisterClass    = 'R';
00066 I     0  0     StackClass       = 'S';
00067 I     0  0     TrueFalseClass   = 'T';
00068 I     0  0     UtilityClass     = 'U';
00069 I     0  0     ExecuteClass     = 'X';
00070 I     0  0
00071 I     0  0     Blank  =    ;
00072 I     0  0     Elipse =  .';
00073 I     0  0
00074 I     0  0     StartOfData = 1;
00075 I     0  0     DataArraySize = 50;
00076 I     0  0
00077 I     0  0     EndofLine = 0;
00078 I     0  0     StartofLine = 1;
00079 I     0  0
00080 I     0  0     ShortStringLength = 8;
00081 I     0  0     InputLineLength = 120;
00082 I     0  0
00083 I     0  0     BooleanID  = 1;
00084 I     0  0     ByteID     = 2;
```

```
-LINE-IDC-PL-SL-

00096 I    0  0     CommaID      = 5;
00097 I    0  0
00098 I    0  0     NullEntries = 0;
00099 I    0  0
00100 I    0  0     AddressState   = 'A';
00101 I    0  0     BooleanState   = 'B';
00102 I    0  0     ErrorState     = 'E';
00103 I    0  0     HexState       = 'H';
00104 I    0  0     IntegerState   = 'I';
00105 I    0  0     SegmentState   = 'S';
00106 I    0  0     SeparatorState =  ,';
00107 I    0  0     UnaryState     = '-';
00108 I    0  0     ColonState     =  :';
00109 I    0  0
00110 I    0  0     NegativeOperator = -1;
00111 I    0  0     PositiveOperator = +1;
00112 I    0  0
00113 I    0  0     NullSegmentValue = 0;
00114 I    0  0     NullAddressValue = 0;
00115 I    0  0
00116 I    0  0     FirstBreakpoint  =  0;
00117 I    0  0     LastBreakpoint   = 15;
00118 I    0  0     AllBPIDs         = 16;
00119 I    0  0     Activate         = true;
00120 I    0  0     DeActivate       = false;
00121 I    0  0     BreakpointOpcode = %X'CC';
00122      0  0
00123      0  0     %include  [-]RegID.defn/nolist'
00187   C  0  0 {
00188   C  0  0     CSId = %B'01';
00189   C  0  0     CSWidth = 2;
00190   C  0  0 }
```

```
-LINE-IDC-PL-SL-

00192       0  0 type
00193       0  0     %include '[-]Type.defn/list'
00194 I     0  0     ZeroOne = -1..1;                        {defines integer with range
00195 I C   0  0                                              of zero to one with an invalid
00196 I C   0  0                                              state of -1}
00197 I     0  0
00198 I     0  0     OpcodeLUTEntry = record                 {defines entries in opcode LUT}
00199 I     0  0   OpcodeClass: char;                        {class D, A, L, S, C, P}
00200 I     0  0   OpcodeKey: integer;                       {key to interpret opcode from
00201 I C   0  0                                              0 to 7}
00202 I     0  0   DirectionBitPresent: boolean;
00203 I     0  0   WidthBitPresent: boolean
00204 I     0  0     end;
00205 I     0  0
00206 I     0  0     OpcodeType = record
00207 I     0  0   Direction: ZeroOne;                      {to or from CPU}
00208 I     0  0   Full: unsigned;                          {full opcode}
00209 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00210 I     0  0     end;
00211 I     0  0
00212 I     0  0     RegisterType = record
00213 I     0  0   Id: integer;                             {identifier}
00214 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00215 I     0  0     end;
00216 I     0  0
00217 I     0  0     EffectiveAddressType = record
00218 I     0  0   Mode: char;                              {register or memory}
00219 I     0  0   Width: ZeroOne;                          {8 or 16 bits}
00220 I     0  0   Address: unsigned;                      {memory address or
00221 I C   0  0                                              register designation}
00222 I     0  0   Segment: integer                        {segment register to use}
00223 I     0  0     end;
00224 I     0  0
00225 I     0  0     NameType = packed array[1..10] of char;
00226 I     0  0
00227 I     0  0     Characters = set of    ..'~';
00228 I     0  0     LettersAndNumbers = set of '0'..'z';
00229 I     0  0     Letters = set of 'A'..'z';
00230 I     0  0     Numbers = set of '0'..'9';
00231 I     0  0
00232 I     0  0     FileName = packed array [1..FilenameLength] of char;
```

```
-LINE-IDC-PL-SL-

00234       0  0     %include  [-]DebugType.defn/list'
00235 I     0  0     NameString = packed array[1..ShortStringLength] of char;
00236 I     0  0
00237 I     0  0     KeywordCharacteristics = record
00238 I     0  0   Name: NameString;
00239 I     0  0   Class: char;
00240 I     0  0   Width: ZeroOne;
00241 I     0  0   ID: integer;
00242 I     0  0   Value: unsigned
00243 I     0  0     end;
00244 I     0  0
00245 I     0  0     InputPointerRange = integer;
00246 I     0  0     InputLine = packed array [1..InputLineLength] of char;
00247 I     0  0
00248 I     0  0     ShortString = packed array [1..ShortStringLength] of char;
00249 I     0  0
00250 I     0  0     DataArrayType = array [StartOfData..DataArraySize] of unsigned;
00251 I     0  0
00252 I     0  0     BreakRecord = record
00253 I     0  0   Encountered: boolean;
00254 I     0  0   Activated: boolean;
00255 I     0  0   Segment: unsigned;
00256 I     0  0   Address: unsigned;
00257 I     0  0   PhysicalAddress: unsigned;
00258 I     0  0   Code: unsigned
00259 I     0  0     end;
00260 I     0  0
00261 I     0  0     BreakPointTableType = array [FirstBreakpoint..LastBreakpoint] of BreakRecord;
00262       0  0
00263       0  0 var
00264       0  0     TempPointer: InputPointerRange;
00265       0  0     Separator: KeywordCharacteristics;
00266       0  0
00267       0  0     IP: [external] unsigned;
00268       0  0
00269       0  0     LocalStopCode: char;
```

```
-LINE-IDC-PL-SL-

00271      1  0 [global] procedure GetAddress( var GASegmentValue: unsigned;
00272      1  0           var GAAddressValue: unsigned;
00273      1  0           GASourceLine: InputLine;
00274      1  0           var GASourcePointer: InputPointerRange;
00275      1  0           GASourceLength: InputPointerRange;
00276      1  0           var GAEOLNFlag: boolean;
00277      1  0           var GANullAddress: boolean);
00278      1  0
00279      2  0 procedure SyntaxInterpreter( var SIKeyword: KeywordCharacteristics;
00280      2  0           var SIOperatorInput: InputLine;
00281      2  0           var SIInputPointer: InputPointerRange;
00282      2  0           var SIInputLength: InputPointerRange;
00283      1  0           var SIEndofLine: boolean); external;
00284      1  0
00285      2  0 procedure FetchRegPtr( FRPWidth: integer;
00286      2  0           FRPDesignator: integer;
00287      2  0           var FRPValue: unsigned;
00288      1  0           var FRPStopCode: char); external;
```

```
-LINE-IDC-PL-SL-

00290   C  1  0 { Title: GetAddressComponent
00291   C  1  0
00292   C  1  0   Purpose: Translate operator input from register designation or
00293   C  1  0            numeric value into an address component
00294   C  1  0
00295   C  1  0   Author: William A. Chapman      Date: August 29, 1986
00296   C  1  0
00297   C  1  0   Inputs: SourceLine
00298   C  1  0           SourcePointer
00299   C  1  0           SourceLength
00300   C  1  0           EOLNFlag
00301   C  1  0
00302   C  1  0   Outputs: A numeric value is returned as part of the address
00303   C  1  0            SourcePointer is updated
00304   C  1  0            EndOfLineFlag is determined
00305   C  1  0            NullAddress is determined
00306   C  1  0
00307   C  1  0   Procedures Invoked: SyntaxInterpreter
00308   C  1  0                      FetchRegPtr
00309   C  1  0 }
00310      2  0 procedure GetAddressComponent( var GACAddressComponent: unsigned;
00311      2  0                                GACSourceLine: InputLine;
00312      2  0                                var GACSourcePointer: InputPointerRange;
00313      2  0                                GACSourceLength: InputPointerRange;
00314      2  0                                var GACEOLNFlag: boolean;
00315      2  0                                var GACNullAddress: boolean);
00316      2  0
00317      2  0 var
00318      2  0     AddressRecord: KeywordCharacteristics;
00319      2  0     LocalStopCode: char;
00320      2  0
00321      2  1 begin
00322      2  1     GACNullAddress := false;
00323      2  1
00324      2  1     SyntaxInterpreter( AddressRecord, GACSourceLine, GACSourcePointer,
00325      2  1    GACSourceLength, GACEOLNFlag);
00326      2  1
00327      2  2     case AddressRecord.Class of
00328      2  2
00329      2  2     BlankClass: GACNullAddress := true;
00330      2  2
00331      2  2     NumericClass: GACAddressComponent := AddressRecord.Value;
00332      2  2
00333      2  2     RegisterClass: with AddressRecord do
```

```
00334      2  2   if (((Width * 8) + ID) = 20) then GACAddressComponent := IP
00335      2  2   else FetchRegPtr( Width, ID, GACAddressComponent, LocalStopCode);
00336      2  2
00337      2  2     otherwise writeln( 'Invalid Address Component: "', AddressRecord.Name, '"');
00338      2  2
00339      2  1     end; {case AddressRecord.Class}
00340      1  0 end;
```

```
-LINE-IDC-PL-SL-

00342      1  1 begin
00343      1  1     GetAddressComponent( GASegmentValue, GASourceLine, GASourcePointer,
00344      1  1   GASourceLength, GAEOLNFlag, GANullAddress);
00345      1  2     if (GANullAddress = true) then begin
00346      1  2   GAAddressValue := NullAddressValue;
00347      1  2   GASegmentValue := NullSegmentValue;
00348      1  2     end
00349      1  2     else if (GAEOLNFlag = true) then begin
00350      1  2       GAAddressValue := GASegmentValue;
00351      1  2       FetchRegPtr( CSWidth, CSID, GASegmentValue, LocalStopCode);
00352      1  2   end
00353      1  2   else begin
00354      1  2       TempPointer := GASourcePointer;
00355      1  2       SyntaxInterpreter( Separator, GASourceLine, GASourcePointer,
00356      1  2           GASourceLength, GAEOLNFlag);
00357      1  2       if ((Separator.Class = OperatorClass) and (Separator.ID = ColonID)) then
00358      1  2           GetAddressComponent( GAAddressValue, GASourceLine,
00359      1  2               GASourcePointer, GASourceLength, GAEOLNFlag, GANullAddress)
00360      1  3       else begin {no Segment Value, only Address}
00361      1  3           GASourcePointer := TempPointer;
00362      1  3           GAAddressValue := GASegmentValue;
00363      1  3           FetchRegPtr( CSWidth, CSID, GASegmentValue, LocalStopCode);
00364      1  2       end;
00365      1  1    end;
00366      0  0 end;
00367      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	542	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	26	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS GETADDRESS.PAS

  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
  /OPTIMIZE
  /STANDARD=NONE
  /TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
  /USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
  /NOANALYSIS_DATA
  /NOENVIRONMENT
  /LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]GETADDRESS.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]GETADDRESS.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	406	00:00.5	00:03.7
Source Analysis	321	00:01.0	00:04.6
Source Listing	26	00:00.6	00:05.6
Tree Construction	101	00:00.2	00:00.5
Flow Analysis	39	00:00.1	00:00.2
Value Propagation	10	00:00.0	00:00.1
Profit Analysis	60	00:00.1	00:00.7
Context Analysis	234	00:00.8	00:03.0
Name Packing	6	00:00.0	00:00.2
Code Selection	77	00:00.2	00:01.1
Final	66	00:00.2	00:00.9
TOTAL	1350	00:03.6	00:20.6

COMPILATION STATISTICS

```
  CPU Time:       00:03.6         (6066 Lines/Minute)
```

```
-LINE-IDC-PL-SL-

00001   C  0  0 { Title: GetData
00002   C  0  0
00003   C  0  0   Purpose: Parse data off of operator input string
00004   C  0  0
00005   C  0  0   Author: William A. Chapman      Date: September 4, 1986
00006   C  0  0
00007   C  0  0   Inputs: SourceLine - operator input
00008   C  0  0           SourcePointer - current position in operator input
00009   C  0  0
00010   C  0  0   Outputs: Data array is loaded with data pattern
00011   C  0  0           DataArrayEntries is set to the number of entries
00012   C  0  0           SourcePointer is updated
00013   C  0  0           EOLNFlag is updated
00014   C  0  0
00015   C  0  0   Procedures Invoked: SyntaxInterpreter
00016   C  0  0           UnpackHex
00017   C  0  0 }
00018      0  0 module GetData( input, output);
00019      0  0
00020      0  0 const
00021      0  0     %include '[-]Const.defn/list'
00022 I    0  0     FirstOpcodeValue = 0;
00023 I    0  0     LastOpcodeValue = 255;
00024 I    0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00025 I    0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00026 I    0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00027 I    0  0     Bit8Multiplier = %X'100';
00028 I    0  0     Bit8Divisor    = %X'100';
00029 I    0  0     WordMultiplier = %X'100';
00030 I    0  0
00031 I    0  0     EightBits = 0;                        {width for 8 bits}
00032 I    0  0     SixteenBits = 1;                      {width for 16 bits}
00033 I    0  0
00034 I    0  0     SimpleMode = %B'0';
00035 I    0  0     SimpleRM = %B'110';
00036 I    0  0
00037 I    0  0     LowMemoryLimit = 0;                   {low limit on memory array}
00038 I    0  0     HighMemoryLimit = 2048;               {high limit on memory array}
00039 I    0  0
00040 I    0  0     FilenameLength = 40;
```

```
-LINE-IDC-PL-SL-

00042       0  0     %include  [-]DebugConst.defn/list'
00043  I    0  0     FirstKeywordEntry = 0;
00044  I    0  0     LastKeywordEntry  = 100;
00045  I    0  0     UndefinedName = 'Qqqqqqqq';
00046  I    0  0     UndefinedClass = 'z';
00047  I    0  0     UndefinedWidth = -1;
00048  I    0  0     UndefinedID = -2;
00049  I    0  0     PageSize = 55;
00050  I    0  0     FF = 12;
00051  I    0  0     MinID = -1;
00052  I    0  0     MaxID = 25;
00053  I    0  0     MinWidth = 0;
00054  I    0  0     MaxWidth = 3;
00055  I    0  0     SegmentOffset = 16;
00056  I    0  0
00057  I    0  0     BlankClass       =   ';
00058  I    0  0     BreakpointClass  = 'B';
00059  I    0  0     ErrorClass       = 'E';
00060  I    0  0     FlagClass        = 'F';
00061  I    0  0     InputOutputClass = 'I';
00062  I    0  0     MemoryClass      = 'M';
00063  I    0  0     NumericClass     = 'N';
00064  I    0  0     OperatorClass    = 'O';
00065  I    0  0     QualifierClass   = 'Q';
00066  I    0  0     RegisterClass    = 'R';
00067  I    0  0     StackClass       = 'S';
00068  I    0  0     TrueFalseClass   = 'T';
00069  I    0  0     UtilityClass     = 'U';
00070  I    0  0     ExecuteClass     = 'X';
00071  I    0  0
00072  I    0  0     Blank  = '  ;
00073  I    0  0     Elipse =  .';
00074  I    0  0
00075  I    0  0     StartOfData = 1;
00076  I    0  0     DataArraySize = 50;
00077  I    0  0
00078  I    0  0     EndofLine = 0;
00079  I    0  0     StartofLine = 1;
00080  I    0  0
00081  I    0  0     ShortStringLength = 8;
00082  I    0  0     InputLineLength = 120;
00083  I    0  0
00084  I    0  0     BooleanID  = 1;
00085  I    0  0     ByteID     = 2;
```

```
00086 I    0  0     IntegerID  = 3;
00087 I    0  0     PointerID  = 4;
00088 I    0  0     SIntegerID = 5;
00089 I    0  0     WordID     = 6;
00090 I    0  0     LengthID   = 3;
00091 I    0  0     ToID       = 5;
00092 I    0  0
00093 I    0  0     AssignmentID = 1;
00094 I    0  0     ColonID      = 2;
00095 I    0  0     PlusID       = 3;
00096 I    0  0     MinusID      = 4;
```

```
-LINE-IDC-PL-SL-

00097 I    0  0     CommaID      = 5;
00098 I    0  0
00099 I    0  0     NullEntries = 0;
00100 I    0  0
00101 I    0  0     AddressState   = 'A';
00102 I    0  0     BooleanState   = 'B';
00103 I    0  0     ErrorState     = 'E';
00104 I    0  0     HexState       = 'H';
00105 I    0  0     IntegerState   = 'I';
00106 I    0  0     SegmentState   = 'S';
00107 I    0  0     SeparatorState =  ,';
00108 I    0  0     UnaryState     = '-';
00109 I    0  0     ColonState     = ':';
00110 I    0  0
00111 I    0  0     NegativeOperator = -1;
00112 I    0  0     PositiveOperator = +1;
00113 I    0  0
00114 I    0  0     NullSegmentValue = 0;
00115 I    0  0     NullAddressValue = 0;
00116 I    0  0
00117 I    0  0     FirstBreakpoint  =  0;
00118 I    0  0     LastBreakpoint   = 15;
00119 I    0  0     AllBPIDs         = 16;
00120 I    0  0     Activate         = true;
00121 I    0  0     DeActivate       = false;
00122 I    0  0     BreakpointOpcode = %X'CC';
```

```
-LINE-IDC-PL-SL-

00124       0  0 type
00125       0  0     %include '[-]Type.defn/list'
00126 I     0  0     ZeroOne = -1..1;                        {defines integer with range
00127 I C   0  0                                              of zero to one with an invalid
00128 I C   0  0                                              state of -1}
00129 I     0  0
00130 I     0  0     OpcodeLUTEntry = record                 {defines entries in opcode LUT}
00131 I     0  0   OpcodeClass: char;                        {class D, A, L, S, C, P}
00132 I     0  0   OpcodeKey: integer;                       {key to interpret opcode from
00133 I C   0  0                                              0 to 7}
00134 I     0  0   DirectionBitPresent: boolean;
00135 I     0  0   WidthBitPresent: boolean
00136 I     0  0     end;
00137 I     0  0
00138 I     0  0     OpcodeType = record
00139 I     0  0   Direction: ZeroOne;                      {to or from CPU}
00140 I     0  0   Full: unsigned;                          {full opcode}
00141 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00142 I     0  0     end;
00143 I     0  0
00144 I     0  0     RegisterType = record
00145 I     0  0   Id: integer;                             {identifier}
00146 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00147 I     0  0     end;
00148 I     0  0
00149 I     0  0     EffectiveAddressType = record
00150 I     0  0   Mode: char;                              {register or memory}
00151 I     0  0   Width: ZeroOne;                          {8 or 16 bits}
00152 I     0  0   Address: unsigned;                       {memory address or
00153 I C   0  0                                              register designation}
00154 I     0  0   Segment: integer                         {segment register to use}
00155 I     0  0     end;
00156 I     0  0
00157 I     0  0     NameType = packed array[1..10] of char;
00158 I     0  0
00159 I     0  0     Characters = set of    ..'~';
00160 I     0  0     LettersAndNumbers = set of '0'..'z';
00161 I     0  0     Letters = set of 'A'..'z';
00162 I     0  0     Numbers = set of '0'..'9';
00163 I     0  0
00164 I     0  0     FileName = packed array [1..FilenameLength] of char;
```

```
-LINE-IDC-PL-SL-

00166       0  0     %include  [-]DebugType.defn/list'
00167 I     0  0     NameString = packed array[1..ShortStringLength] of char;
00168 I     0  0
00169 I     0  0     KeywordCharacteristics = record
00170 I     0  0   Name: NameString;
00171 I     0  0   Class: char;
00172 I     0  0   Width: ZeroOne;
00173 I     0  0   ID: integer;
00174 I     0  0   Value: unsigned
00175 I     0  0     end;
00176 I     0  0
00177 I     0  0     InputPointerRange = integer;
00178 I     0  0     InputLine = packed array [1..InputLineLength] of char;
00179 I     0  0
00180 I     0  0     ShortString = packed array [1..ShortStringLength] of char;
00181 I     0  0
00182 I     0  0     DataArrayType = array [StartOfData..DataArraySize] of unsigned;
00183 I     0  0
00184 I     0  0     BreakRecord = record
00185 I     0  0   Encountered: boolean;
00186 I     0  0   Activated: boolean;
00187 I     0  0   Segment: unsigned;
00188 I     0  0   Address: unsigned;
00189 I     0  0   PhysicalAddress: unsigned;
00190 I     0  0   Code: unsigned
00191 I     0  0     end;
00192 I     0  0
00193 I     0  0     BreakPointTableType = array [FirstBreakpoint..LastBreakpoint] of BreakRecord;
00194       0  0
00195       0  0 var
00196       0  0     Datum: KeywordCharacteristics;
00197       0  0     GDState: char;
00198       0  0     GDArrayPointer: StartOfData..DataArraySize;
00199       0  0     GDIntegerValue: integer;
00200       0  0     UnaryOperator: NegativeOperator..PositiveOperator;
```

```
-LINE-IDC-PL-SL-

00202      1  0 [global] procedure GetData( var GDArray: DataArrayType;
00203      1  0           var GDArrayEntries: integer;
00204      1  0           GDSourceLine: InputLine;
00205      1  0           var GDSourcePointer: InputPointerRange;
00206      1  0           GDSourceLength: InputPointerRange;
00207      1  0           var GDEOLNFlag: boolean;
00208      1  0           GDFormatIdent: integer);
00209      1  0
00210      2  0 procedure SyntaxInterpreter( var SIKeyword: KeywordCharacteristics;
00211      2  0           var SIOperatorInput: InputLine;
00212      2  0           var SIInputPointer: InputPointerRange;
00213      2  0           var SIInputLength: InputPointerRange;
00214      1  0           var SIEndofLine: boolean); external;
00215      1  0
00216      2  0 function UnpackHex( UPPackedHex: unsigned;
00217      1  0           UHSignValue: integer): integer; external;
```

```
-LINE-IDC-PL-SL-

00219      1  1 begin
00220      1  1     GDArrayPointer := StartOfData;
00221      1  1     GDArrayEntries := NullEntries;
00222      1  1
00223      1  2     case GDFormatIdent of
00224      1  2
00225      1  3     BooleanID: begin
00226      1  3   GDState := BooleanState;
00227      1  4   while (GDEOLNFlag = false) do begin
00228      1  4       SyntaxInterpreter( Datum, GDSourceLine, GDSourcePointer, GDSourceLength,
00229      1  4           GDEOLNFlag);
00230      1  5       case Datum.Class of
00231      1  5
00232      1  5       BlankClass: ; {do nothing}
00233      1  5
00234      1  6       TrueFalseClass: begin
00235      1  7           if (GDState = BooleanState) then begin
00236      1  7               GDArray[ GDArrayPointer] := uint( Datum.ID);
00237      1  7               GDArrayPointer := GDArrayPointer + 1;
00238      1  7               GDArrayEntries := GDArrayEntries + 1;
00239      1  7               GDState := SeparatorState;
00240      1  7           end {if GDState = BooleanState}
00241      1  7           else begin
00242      1  7               GDState := ErrorState;
00243      1  7               writeln( 'Invalid input (must be "true" or "false"): "',
00244      1  7                    Datum.Name, '"');
00245      1  6           end; {else GDState <> BooleanState}
00246      1  5       end; {TrueFalseClass}
00247      1  5
00248      1  6       OperatorClass: begin
00249      1  6           if ((GDState = SeparatorState) and (Datum.ID = CommaID)) then
00250      1  6               GDState := BooleanState
00251      1  7           else begin
00252      1  7               GDState := ErrorState;
00253      1  7               writeln( 'Invalid punctuation: "', Datum.Name, '"');
00254      1  6           end; {else GDState = SeparatorState and Datum.ID = CommanID}
00255      1  5       end; {OperatorClass}
00256      1  5
00257      1  6       otherwise begin
00258      1  6           writeln( 'Invalid keyword: "', Datum.Name, '"');
00259      1  6           GDState := ErrorState;
00260      1  5       end; {otherwise}
00261      1  4       end; {case Datum.Class}
00262      1  4
```

```
00263      1  4       if (GDState = ErrorState) then GDEOLNFlag := true;
00264      1  3   end; {while GDEOLNFlag is false}
00265      1  2     end; {case BooleanID}
```

```
-LINE-IDC-PL-SL-

00267      1  3     ByteID, WordID: begin
00268      1  3   GDState := HexState;
00269      1  4   while (GDEOLNFlag = false) do begin
00270      1  4       SyntaxInterpreter( Datum, GDSourceLine, GDSourcePointer, GDSourceLength,
00271      1  4           GDEOLNFlag);
00272      1  5       case Datum.Class of
00273      1  5
00274      1  5       BlankClass: ; {do nothing}
00275      1  5
00276      1  6       NumericClass: begin
00277      1  7           if (GDState = HexState) then begin
00278      1  7               GDArray[ GDArrayPointer] := Datum.Value;
00279      1  7               GDArrayPointer := GDArrayPointer + 1;
00280      1  7               GDArrayEntries := GDArrayEntries + 1;
00281      1  7               GDState := SeparatorState;
00282      1  7           end {if GDState = HexState}
00283      1  7           else begin
00284      1  7               GDState := ErrorState;
00285      1  7               writeln( 'Invalid input (must be hexadecimal): "',
00286      1  7                    Datum.Name, '"');
00287      1  6           end; {else GDState <> HexState}
00288      1  5       end; {NumericClass}
00289      1  5
00290      1  6       OperatorClass: begin
00291      1  6           if ((GDState = SeparatorState) and (Datum.ID = CommaID)) then
00292      1  6               GDState := HexState
00293      1  7           else begin
00294      1  7               GDState := ErrorState;
00295      1  7               writeln( 'Invalid punctuation: "', Datum.Name, '"');
00296      1  6           end; {else GDState = SeparatorState and Datum.ID = CommanID}
00297      1  5       end; {OperatorClass}
00298      1  5
00299      1  6       otherwise begin
00300      1  6           writeln( 'Invalid keyword: "', Datum.Name, '"');
00301      1  6           GDState := ErrorState;
00302      1  5       end; {otherwise}
00303      1  4       end; {case Datum.Class}
00304      1  4
00305      1  4       if (GDState = ErrorState) then GDEOLNFlag := true;
00306      1  3   end; {while GDEOLNFlag is false}
00307      1  2     end; {case ByteID, WordID}
```

```
-LINE-IDC-PL-SL-

00309      1  3     IntegerID, SIntegerID: begin
00310      1  3   GDState := IntegerState;
00311      1  3   UnaryOperator := PositiveOperator;
00312      1  4   while (GDEOLNFlag = false) do begin
00313      1  4       SyntaxInterpreter( Datum, GDSourceLine, GDSourcePointer, GDSourceLength,
00314      1  4           GDEOLNFlag);
00315      1  5       case Datum.Class of
00316      1  5
00317      1  5       BlankClass: ; {do nothing}
00318      1  5
00319      1  6       NumericClass: begin
00320      1  7           case GDState of
00321      1  7
00322      1  8           IntegerState: begin
00323      1  8               GDIntegerValue := UnpackHex( Datum.Value, UnaryOperator);
00324      1  8               GDArray[ GDArrayPointer] := uint( GDIntegerValue);
00325      1  8               GDArrayPointer := GDArrayPointer + 1;
00326      1  8               GDArrayEntries := GDArrayEntries + 1;
00327      1  8               GDState := SeparatorState;
00328      1  7           end; {case IntegerState}
00329      1  7
00330      1  8           otherwise begin
00331      1  8               GDState := ErrorState;
00332      1  8               writeln( 'Invalid input (must be numeric): "',
00333      1  8                    Datum.Name, '"');
00334      1  7           end; {otherwise}
00335      1  7
00336      1  6           end; {case GDState}
00337      1  5       end; {NumericClass}
```

```
-LINE-IDC-PL-SL-

00339       1  6       OperatorClass: begin
00340       1  7           case GDState of
00341       1  7
00342       1  8           SeparatorState: begin
00343       1  8               if (Datum.ID = CommaID) then GDState := IntegerState
00344       1  9               else begin
00345       1  9                   writeln( 'Invalid punctuation: "', Datum.Name, '"');
00346       1  9                   GDState := ErrorState;
00347       1  8               end; {else Datum.ID = CommaID}
00348       1  7           end; {SeparatorState}
00349       1  7
00350       1  8           IntegerState: begin
00351       1  9               case Datum.ID of
00352       1  9
00353       1  9               MinusID: UnaryOperator := UnaryOperator * NegativeOperator;
00354       1  9
00355       1  9               PlusID: UnaryOperator := UnaryOperator * PositiveOperator;
00356       1  9
00357       1 10               otherwise begin
00358       1 10                   writeln( 'Invalid unary operator: "', Datum.Name, '"');
00359       1 10                   GDState := ErrorState;
00360       1  9               end; {otherwise}
00361       1  9
00362       1  8               end; {case Datum.ID}
00363       1  7           end; {IntegerState}
00364       1  7
00365       1  8           otherwise begin
00366       1  8               GDState := ErrorState;
00367       1  8               writeln( 'Invalid punctuation: "', Datum.Name, '"');
00368       1  7           end; {otherwise}
00369       1  6           end; {case GDState}
00370       1  5       end; {OperatorClass}
00371       1  5
00372       1  6       otherwise begin
00373       1  6           writeln( 'Invalid keyword: "', Datum.Name, '"');
00374       1  6           GDState := ErrorState;
00375       1  5       end; {otherwise}
00376       1  5
00377       1  4       end; {case Datum.Class}
00378       1  4
00379       1  4       if (GDState = ErrorState) then GDEOLNFlag := true;
00380       1  3   end; {while GDEOLNFlag is false}
00381       1  2     end; {case IntegerID, SIntegerID}
```

```
-LINE-IDC-PL-SL-

00383       1  3     PointerID: begin
00384       1  3   GDState := SegmentState;
00385       1  4   while (GDEOLNFlag = false) do begin
00386       1  4       SyntaxInterpreter( Datum, GDSourceLine, GDSourcePointer, GDSourceLength,
00387       1  4           GDEOLNFlag);
00388       1  5       case Datum.Class of
00389       1  5
00390       1  5       BlankClass: ; {do nothing}
00391       1  5
00392       1  6       NumericClass: begin
00393       1  7           case GDState of
00394       1  7
00395       1  8           SegmentState: begin
00396       1  8               GDArray[ GDArrayPointer] := Datum.Value;
00397       1  8               GDArrayPointer := GDArrayPointer + 1;
00398       1  8               GDArrayEntries := GDArrayEntries + 1;
00399       1  8               GDState := ColonState;
00400       1  7           end; {case SegmentState}
00401       1  7
00402       1  8           AddressState: begin
00403       1  8               GDArray[ GDArrayPointer] := Datum.Value;
00404       1  8               GDArrayPointer := GDArrayPointer + 1;
00405       1  8               GDArrayEntries := GDArrayEntries + 1;
00406       1  8               GDState := SeparatorState;
00407       1  7           end; {case AddressState}
00408       1  7
00409       1  8           otherwise begin
00410       1  8               GDState := ErrorState;
00411       1  8               writeln( 'Invalid input (must be hexadecimal): "',
00412       1  8                   Datum.Name, '"');
00413       1  7           end; {otherwise}
00414       1  7
00415       1  6           end; {case GDState}
00416       1  5       end; {NumericClass}
```

```
-LINE-IDC-PL-SL-

00418      1  6       OperatorClass: begin
00419      1  7           case GDState of
00420      1  7
00421      1  8           SeparatorState: begin
00422      1  8               if (Datum.ID = CommaID) then GDState := SegmentState
00423      1  9               else begin
00424      1  9                   writeln( 'Invalid punctuation: "', Datum.Name, '"');
00425      1  9                   GDState := ErrorState;
00426      1  8               end; {else Datum.ID = CommaID}
00427      1  7           end; {SeparatorState}
00428      1  7
00429      1  8           ColonState: begin
00430      1  8               if (Datum.ID = ColonID) then GDState := AddressState
00431      1  9               else begin
00432      1  9                   writeln( 'Invalid punctuation: "', Datum.Name, '"');
00433      1  9                   GDState := ErrorState;
00434      1  8               end; {else Datum.ID = ColonID}
00435      1  7           end; {ColonState}
00436      1  7
00437      1  8           otherwise begin
00438      1  8               GDState := ErrorState;
00439      1  8               writeln( 'Invalid punctuation: "', Datum.Name, '"');
00440      1  7           end; {otherwise}
00441      1  6           end; {case GDState}
00442      1  5       end; {OperatorClass}
00443      1  5
00444      1  6       otherwise begin
00445      1  6           writeln( 'Invalid keyword: "', Datum.Name, '"');
00446      1  6           GDState := ErrorState;
00447      1  5       end; {otherwise}
00448      1  4       end; {case Datum.Class}
00449      1  4
00450      1  4       if (GDState = ErrorState) then GDEOLNFlag := true;
00451      1  3   end; {while GDEOLNFlag is false}
00452      1  3
00453      1  3   if (GDState <> ErrorState) then
00454      1  3       if (GDState <> SeparatorState) then
00455      1  3           writeln( 'Mismatched Segment:Address information');
00456      1  2     end; {case PointerID}
00457      1  2
00458      1  2     otherwise writeln( 'Invalid Format Identifier: "', GDFormatIdent:1, '"');
00459      1  1     end; {Case GDFormatIdent}
00460      0  0 end;
00461      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	3299	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	36	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS GETDATA.PAS

  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
  /OPTIMIZE
  /STANDARD=NONE
  /TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
  /USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
  /NOANALYSIS_DATA
  /NOENVIRONMENT
  /LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]GETDATA.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]GETDATA.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	403	00:00.5	00:07.1
Source Analysis	350	00:01.3	00:09.2
Source Listing	19	00:00.7	00:05.9
Tree Construction	160	00:00.4	00:01.1
Flow Analysis	59	00:00.3	00:00.6
Value Propagation	10	00:00.0	00:00.1
Profit Analysis	33	00:00.1	00:00.6
Context Analysis	250	00:02.3	00:10.5
Name Packing	7	00:00.0	00:00.0
Code Selection	101	00:00.4	00:00.8
Final	65	00:00.8	00:03.6
TOTAL	1461	00:06.8	00:39.5

COMPILATION STATISTICS

```
  CPU Time:       00:06.8         (4044 Lines/Minute)
```

```
-LINE-IDC-PL-SL-

00001   C  0  0 {
00002   C  0  0   Title: Hexadecimal Input module
00003   C  0  0
00004   C  0  0   Purpose: Obtain a hexadecimal keyboard input string
00005   C  0  0            and convert it to a decimal number.
00006   C  0  0
00007   C  0  0   Author: William A. Chapman      Date: December 28, 1985
00008   C  0  0
00009   C  0  0   Inputs: Keyboard string of characters
00010   C  0  0
00011   C  0  0   Outputs: An unsigned integer representation of the
00012   C  0  0            character hexadecimal string.
00013   C  0  0
00014   C  0  0   Procedures Invoked: none
00015   C  0  0 }
00016      0  0 module HexIn(input,output);
00017      0  0
00018      0  0 var
00019      0  0     HexInChar: char;                    {character to process}
00020      0  0     Temp: integer;                      {temporary variable}
00021      0  0     Done: boolean;
00022      0  0
00023      1  0 [global] procedure HexIn(var HIValue: unsigned);
00024      1  0
00025      1  0
00026      1  0
00027      1  1 begin
00028      1  1     HIValue := 0;
00029      1  1     Done := false;
00030      1  2     while (not Done) do begin
00031      1  2   if (eoln(input)) then Done := true
00032      1  3   else begin
00033      1  3       read(HexInChar);
00034      1  3       Temp := ord(HexInChar) - ord('0');
00035      1  3       if (HexInChar = ' ') then Temp := 0;
00036      1  3       if (Temp > 10) then Temp := Temp + ord('0') - ord('A') + 10;
00037      1  3       if (Temp > 15) then Temp := Temp + ord('A') - ord('a');
00038      1  4       if ((Temp < 0) or (Temp > 15)) then begin
00039      1  4           writeln;
00040      1  4           write( 'HexIn: Invalid character for conversion: ');
00041      1  4           writeln( '"', HexInChar, '"');
00042      1  4           HIValue := 0;
00043      1  4           Temp := 0;
00044      1  4           Done := true;
00045      1  4       end {invalid character]
```

```
00046      1  3       else HIValue := (HIValue * 16) + Temp;
00047      1  2   end;
00048      1  1     end;
00049      1  1     readln;
00050      0  0 end;
00051      0  0 end. {module}
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	363	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	9	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS HEXIN.PAS

  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
  /OPTIMIZE
  /STANDARD=NONE
  /TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
  /USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
  /NOANALYSIS_DATA
  /NOENVIRONMENT
  /LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]HEXIN.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]HEXIN.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	406	00:00.5	00:08.2
Source Analysis	175	00:00.2	00:00.6
Source Listing	13	00:00.1	00:03.0
Tree Construction	100	00:00.1	00:00.5
Flow Analysis	42	00:00.1	00:00.1
Value Propagation	10	00:00.0	00:00.0
Profit Analysis	43	00:00.0	00:00.0
Context Analysis	232	00:00.5	00:02.7
Name Packing	6	00:00.0	00:00.0
Code Selection	79	00:00.1	00:00.1
Final	73	00:00.2	00:00.9
TOTAL	1183	00:01.7	00:16.0

COMPILATION STATISTICS

```
  CPU Time:       00:01.7         (1821 Lines/Minute)
```

```
-LINE-IDC-PL-SL-

00001   C  0  0 { Title: InstallBreakpoints
00002   C  0  0
00003   C  0  0   Purpose: Install breakpoint opcode
00004   C  0  0
00005   C  0  0   Author: William A. Chapman      Date: September 18, 1986
00006   C  0  0
00007   C  0  0   Inputs: BreakPointTable (global)
00008   C  0  0
00009   C  0  0   Outputs: Breakpoint opcode is installed for all activated breakpoints
00010   C  0  0
00011   C  0  0   Procedures Invoked: FetchMemory
00012   C  0  0                       StoreMemory
00013   C  0  0 }
00014      0  0 module InstallBreakpoints( input, output);
00015      0  0
00016      0  0 const
00017      0  0     %include '[-]Const.defn/list'
00018 I    0  0     FirstOpcodeValue = 0;
00019 I    0  0     LastOpcodeValue = 255;
00020 I    0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00021 I    0  0     Low8Bits = %X'FF';                     {mask for low 8 bits}
00022 I    0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00023 I    0  0     Bit8Multiplier = %X'100';
00024 I    0  0     Bit8Divisor    = %X'100';
00025 I    0  0     WordMultiplier = %X'100';
00026 I    0  0
00027 I    0  0     EightBits = 0;                         {width for 8 bits}
00028 I    0  0     SixteenBits = 1;                       {width for 16 bits}
00029 I    0  0
00030 I    0  0     SimpleMode = %B'0';
00031 I    0  0     SimpleRM = %B'110';
00032 I    0  0
00033 I    0  0     LowMemoryLimit = 0;                    {low limit on memory array}
00034 I    0  0     HighMemoryLimit = 2048;                {high limit on memory array}
00035 I    0  0
00036 I    0  0     FilenameLength = 40;
```

```
-LINE-IDC-PL-SL-

00038       0  0     %include  [-]DebugConst.defn/list'
00039 I     0  0     FirstKeywordEntry = 0;
00040 I     0  0     LastKeywordEntry  = 100;
00041 I     0  0     UndefinedName = 'Qqqqqqqq';
00042 I     0  0     UndefinedClass = 'z';
00043 I     0  0     UndefinedWidth = -1;
00044 I     0  0     UndefinedID = -2;
00045 I     0  0     PageSize = 55;
00046 I     0  0     FF = 12;
00047 I     0  0     MinID = -1;
00048 I     0  0     MaxID = 25;
00049 I     0  0     MinWidth = 0;
00050 I     0  0     MaxWidth = 3;
00051 I     0  0     SegmentOffset = 16;
00052 I     0  0
00053 I     0  0     BlankClass       =    ;
00054 I     0  0     BreakpointClass  = 'B';
00055 I     0  0     ErrorClass       = 'E';
00056 I     0  0     FlagClass        = 'F';
00057 I     0  0     InputOutputClass = 'I';
00058 I     0  0     MemoryClass      = 'M';
00059 I     0  0     NumericClass     = 'N';
00060 I     0  0     OperatorClass    = 'O';
00061 I     0  0     QualifierClass   = 'Q';
00062 I     0  0     RegisterClass    = 'R';
00063 I     0  0     StackClass       = 'S';
00064 I     0  0     TrueFalseClass   = 'T';
00065 I     0  0     UtilityClass     = 'U';
00066 I     0  0     ExecuteClass     = 'X';
00067 I     0  0
00068 I     0  0     Blank  =    ;
00069 I     0  0     Elipse =  . ;
00070 I     0  0
00071 I     0  0     StartOfData = 1;
00072 I     0  0     DataArraySize = 50;
00073 I     0  0
00074 I     0  0     EndofLine = 0;
00075 I     0  0     StartofLine = 1;
00076 I     0  0
00077 I     0  0     ShortStringLength = 8;
00078 I     0  0     InputLineLength = 120;
00079 I     0  0
00080 I     0  0     BooleanID  = 1;
00081 I     0  0     ByteID     = 2;
```

```
00082 I    0  0     IntegerID  = 3;
00083 I    0  0     PointerID  = 4;
00084 I    0  0     SIntegerID = 5;
00085 I    0  0     WordID     = 6;
00086 I    0  0     LengthID   = 3;
00087 I    0  0     ToID       = 5;
00088 I    0  0
00089 I    0  0     AssignmentID = 1;
00090 I    0  0     ColonID      = 2;
00091 I    0  0     PlusID       = 3;
00092 I    0  0     MinusID      = 4;
```

```
-LINE-IDC-PL-SL-

00093 I    0  0     CommaID      = 5;
00094 I    0  0
00095 I    0  0     NullEntries = 0;
00096 I    0  0
00097 I    0  0     AddressState   = 'A';
00098 I    0  0     BooleanState   = 'B';
00099 I    0  0     ErrorState     = 'E';
00100 I    0  0     HexState       = 'H';
00101 I    0  0     IntegerState   = 'I';
00102 I    0  0     SegmentState   = 'S';
00103 I    0  0     SeparatorState =  , ;
00104 I    0  0     UnaryState     = '-';
00105 I    0  0     ColonState     = ': ;
00106 I    0  0
00107 I    0  0     NegativeOperator = -1;
00108 I    0  0     PositiveOperator = +1;
00109 I    0  0
00110 I    0  0     NullSegmentValue = 0;
00111 I    0  0     NullAddressValue = 0;
00112 I    0  0
00113 I    0  0     FirstBreakpoint  =  0;
00114 I    0  0     LastBreakpoint   = 15;
00115 I    0  0     AllBPIDs         = 16;
00116 I    0  0     Activate         = true;
00117 I    0  0     DeActivate       = false;
00118 I    0  0     BreakpointOpcode = %X'CC';
```

```
-LINE-IDC-PL-SL-

00120       0  0 type
00121       0  0     %include '[-]Type.defn/list'
00122 I     0  0     ZeroOne = -1..1;                          {defines integer with range
00123 I C   0  0                                               of zero to one with an invalid
00124 I C   0  0                                               state of -1}
00125 I     0  0
00126 I     0  0     OpcodeLUTEntry = record                   {defines entries in opcode LUT}
00127 I     0  0   OpcodeClass: char;                         {class D, A, L, S, C, P}
00128 I     0  0   OpcodeKey: integer;                        {key to interpret opcode from
00129 I C   0  0                                               0 to 7}
00130 I     0  0   DirectionBitPresent: boolean;
00131 I     0  0   WidthBitPresent: boolean
00132 I     0  0     end;
00133 I     0  0
00134 I     0  0     OpcodeType = record
00135 I     0  0   Direction: ZeroOne;                        {to or from CPU}
00136 I     0  0   Full: unsigned;                            {full opcode}
00137 I     0  0   Width: ZeroOne                            {8 or 16 bits}
00138 I     0  0     end;
00139 I     0  0
00140 I     0  0     RegisterType = record
00141 I     0  0   Id: integer;                              {identifier}
00142 I     0  0   Width: ZeroOne                            {8 or 16 bits}
00143 I     0  0     end;
00144 I     0  0
00145 I     0  0     EffectiveAddressType = record
00146 I     0  0   Mode: char;                               {register or memory}
00147 I     0  0   Width: ZeroOne;                           {8 or 16 bits}
00148 I     0  0   Address: unsigned;                        {memory address or
00149 I C   0  0                                               register designation}
00150 I     0  0   Segment: integer                          {segment register to use}
00151 I     0  0     end;
00152 I     0  0
00153 I     0  0     NameType = packed array[1..10] of char;
00154 I     0  0
00155 I     0  0     Characters = set of    ..'~';
00156 I     0  0     LettersAndNumbers = set of '0'..'z';
00157 I     0  0     Letters = set of 'A'..'z';
00158 I     0  0     Numbers = set of '0'..'9';
00159 I     0  0
00160 I     0  0     FileName = packed array [1..FilenameLength] of char;
```

```
-LINE-IDC-PL-SL-

00162       0  0     %include '[-]DebugType.defn/list'
00163 I     0  0     NameString = packed array[1..ShortStringLength] of char;
00164 I     0  0
00165 I     0  0     KeywordCharacteristics = record
00166 I     0  0   Name: NameString;
00167 I     0  0   Class: char;
00168 I     0  0   Width: ZeroOne;
00169 I     0  0   ID: integer;
00170 I     0  0   Value: unsigned
00171 I     0  0     end;
00172 I     0  0
00173 I     0  0     InputPointerRange = integer;
00174 I     0  0     InputLine = packed array [1..InputLineLength] of char;
00175 I     0  0
00176 I     0  0     ShortString = packed array [1..ShortStringLength] of char;
00177 I     0  0
00178 I     0  0     DataArrayType = array [StartOfData..DataArraySize] of unsigned;
00179 I     0  0
00180 I     0  0     BreakRecord = record
00181 I     0  0   Encountered: boolean;
00182 I     0  0   Activated: boolean;
00183 I     0  0   Segment: unsigned;
00184 I     0  0   Address: unsigned;
00185 I     0  0   PhysicalAddress: unsigned;
00186 I     0  0   Code: unsigned
00187 I     0  0     end;
00188 I     0  0
00189 I     0  0     BreakPointTableType = array [FirstBreakpoint..LastBreakpoint] of BreakRecord;
00190       0  0
00191       1  0 procedure FetchMemory(FMAddress: unsigned;
00192       1  0           FMSegment: unsigned;
00193       1  0           var FMValue: unsigned;
00194       0  0           var FMStopCode: char); external;
00195       0  0
00196       1  0 procedure StoreMemory(SMAddress: unsigned;
00197       1  0           SMSegment: unsigned;
00198       1  0           SMValue: unsigned;
00199       0  0           var SMStopCode: char); external;
```

```
-LINE-IDC-PL-SL-

00201      1  0 [global] procedure InstallBreakpoints( var BreakPointTable: BreakPointTableType;
00202      1  0           var IBStopCode: char);
00203      1  0
00204      1  0 var
00205      1  0     IBBreakpointLoop: integer;
00206      1  0
00207      1  1 begin
00208      1  1     for IBBreakpointLoop := FirstBreakpoint to LastBreakpoint do
00209      1  1   with BreakPointTable[ IBBreakpointLoop] do
00210      1  1       if (Activated = Activate) then
00211      1  1           if (Encountered = Activate) then Encountered := Deactivate
00212      1  2           else begin
00213      1  2               FetchMemory( Address, Segment, Code, IBStopCode);
00214      1  2               StoreMemory( Address, Segment, BreakpointOpcode, IBStopCode);
00215      1  1           end;
00216      0  0 end;
00217      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	94	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
PAS/LIS INSTALLBREAKPOINTS.PAS

/CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
/DEBUG=(NOSYMBOLS,TRACEBACK)
/SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
/OPTIMIZE
/STANDARD=NONE
/TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
/USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
/NOANALYSIS_DATA
/NOENVIRONMENT
/LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]INSTALLBREAKPOINTS.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]INSTALLBREAKPOINTS.OBJ;1
/NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	401	00:00.5	00:05.9
Source Analysis	245	00:00.7	00:05.2
Source Listing	20	00:00.5	00:04.6
Tree Construction	103	00:00.1	00:00.1
Flow Analysis	37	00:00.0	00:00.0
Value Propagation	11	00:00.0	00:00.0
Profit Analysis	44	00:00.0	00:00.1
Context Analysis	189	00:00.2	00:00.8
Name Packing	7	00:00.0	00:00.2
Code Selection	74	00:00.1	00:00.2
Final	60	00:00.1	00:00.9
TOTAL	1195	00:02.2	00:18.0

COMPILATION STATISTICS

```
CPU Time:       00:02.2         (6000 Lines/Minute)
Elapsed Time:   00:18.0
```

```
-LINE-IDC-PL-SL-

00001    C  0  0 { Title: Interrupt
00002    C  0  0
00003    C  0  0   Purpose: Perform the necessary actions to respond to an interrupt
00004    C  0  0
00005    C  0  0   Author: William A. Chapman      Date: February 22, 1986
00006    C  0  0
00007    C  0  0   Inputs: Interrupt type
00008    C  0  0
00009    C  0  0   Outputs: Flags are pushed onto the stack
00010    C  0  0            Interrupt and Trap Flags are cleared
00011    C  0  0            New values for IP and CS are obtained from the vector
00012    C  0  0                    space indexed by the Interrupt Type
00013    C  0  0
00014    C  0  0   Procedures Invoked: PushFlags
00015    C  0  0            PushIntersegRA
00016    C  0  0            FetchMemory
00017    C  0  0            StoreRegPtr
00018    C  0  0 }
00019       0  0 module Interrupt(input, output );
00020       0  0
00021       0  0 const
00022       0  0     %include  [-]Const.defn/list'
00023 I     0  0     FirstOpcodeValue = 0;
00024 I     0  0     LastOpcodeValue = 255;
00025 I     0  0     All16Bits = %X'FFFF';         {mask for all 16 bits}
00026 I     0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00027 I     0  0     High8Bits = %X'FF00';         {mask for high 8 bits}
00028 I     0  0     Bit8Multiplier = %X'100';
00029 I     0  0     Bit8Divisor    = %X'100';
00030 I     0  0     WordMultiplier = %X'100';
00031 I     0  0
00032 I     0  0     EightBits = 0;                        {width for 8 bits}
00033 I     0  0     SixteenBits = 1;                      {width for 16 bits}
00034 I     0  0
00035 I     0  0     SimpleMode = %B'0';
00036 I     0  0     SimpleRM = %B'110';
00037 I     0  0
00038 I     0  0     LowMemoryLimit = 0;                   {low limit on memory array}
00039 I     0  0     HighMemoryLimit = 2048;               {high limit on memory array}
00040 I     0  0
00041 I     0  0     FilenameLength = 40;
```

```
-LINE-IDC-PL-SL-

00043       0  0     %include '[-]RegId.defn/list'
00044 I     0  0     ALId = %B'000';
00045 I     0  0     ALWidth = 0;
00046 I     0  0
00047 I     0  0     CLId = %B'001';
00048 I     0  0     CLWidth = 0;
00049 I     0  0
00050 I     0  0     DLId = %B'010';
00051 I     0  0     DLWidth = 0;
00052 I     0  0
00053 I     0  0     BLId = %B'011';
00054 I     0  0     BLWidth = 0;
00055 I     0  0
00056 I     0  0     AHId = %B'100';
00057 I     0  0     AHWidth = 0;
00058 I     0  0
00059 I     0  0     CHId = %B'101';
00060 I     0  0     CHWidth = 0;
00061 I     0  0
00062 I     0  0     DHId = %B'110';
00063 I     0  0     DHWidth = 0;
00064 I     0  0
00065 I     0  0     BHId = %B'111';
00066 I     0  0     BHWidth = 0;
00067 I     0  0
00068 I     0  0     AXId = %B'000';
00069 I     0  0     AXWidth = 1;
00070 I     0  0     ALorAXId = %B'000';
00071 I     0  0
00072 I     0  0     CXId = %B'001';
00073 I     0  0     CXWidth = 1;
00074 I     0  0
00075 I     0  0     DXId = %B'010';
00076 I     0  0     DXWidth = 1;
00077 I     0  0
00078 I     0  0     BXId = %B'011 ;
00079 I     0  0     BXWidth = 1;
00080 I     0  0
00081 I     0  0     SPId = %B'100';
00082 I     0  0     SPWidth = 1;
00083 I     0  0
00084 I     0  0     BPId = %B'101';
00085 I     0  0     BPWidth = 1;
00086 I     0  0
```

```
00087 I     0  0      SIId = %B'110';
00088 I     0  0      SIWidth = 1;
00089 I     0  0
00090 I     0  0      DIId = %B'111';
00091 I     0  0      DIWidth = 1;
00092 I     0  0
00093 I     0  0      SRWidth = 2;
00094 I     0  0
00095 I     0  0      ESId = %B'00';
00096 I     0  0      ESWidth = 2;
00097 I     0  0
```

```
-LINE-IDC-PL-SL-

00098 I    0  0     CSId = %B'01';
00099 I    0  0     CSWidth = 2;
00100 I    0  0
00101 I    0  0     SSId = %B'10';
00102 I    0  0     SSWidth = 2;
00103 I    0  0
00104 I    0  0     DSId = %B'11 ;
00105 I    0  0     DSWidth = 2;
00106 I    0  0
00107      0  0     %include  [-]Flags.defn/list'
00108 I    0  0     SetHigh = true;
00109 I    0  0     Clear = false;
00110 I    0  0
00111 I    0  0     CarryFlag     = %B'000000000001';
00112 I    0  0     ParityFlag    = %B'000000000100';
00113 I    0  0     AuxCarryFlag  = %B'000000010000';
00114 I    0  0     ZeroFlag      = %B'000001000000';
00115 I    0  0     SignFlag      = %B'000010000000';
00116 I    0  0     TrapFlag      = %B'000100000000';
00117 I    0  0     InterruptFlag = %B'001000000000';
00118 I    0  0     DirectionFlag = %B'010000000000';
00119 I    0  0     OverflowFlag  = %B'100000000000';
00120      0  0
00121      0  0     BaseSegment = 0;
```

```
-LINE-IDC-PL-SL-

00123       0  0 type
00124       0  0     %include '[-]FlagType.defn/list'
00125 I     0  0     FlagType = record                              [define processor flags]
00126 I     0  0   Carry: boolean;
00127 I     0  0   Parity: boolean;
00128 I     0  0   AuxCarry: boolean;
00129 I     0  0   Zero: boolean;
00130 I     0  0   Sign: boolean;
00131 I     0  0   Trap: boolean;
00132 I     0  0   Interrupt: boolean;
00133 I     0  0   Direction: boolean;
00134 I     0  0   Overflow: boolean
00135 I     0  0     end;
00136       0  0
00137       0  0 var
00138       0  0     IP: [external] unsigned;
00139       0  0     Flags: [external] FlagType;
00140       0  0     LowValue: unsigned;
00141       0  0     HighValue: unsigned;
00142       0  0     Address: unsigned;
00143       0  0     NewCS: unsigned;
00144       0  0
00145       0  0
00146       0  0
00147       0  0
00148       0  0
00149       0  0
00150       1  0 [global] procedure Interrupt(InterruptType: unsigned;
00151       1  0           var IStopCode: char);
00152       1  0
00153       2  0 procedure StoreRegPtr(SRPWidth: integer;
00154       2  0           SRPDesignator: integer;
00155       2  0           SRPValue: unsigned;
00156       1  0           var SRPStopCode: char); external;
00157       1  0
00158       2  0 procedure FetchMemory(FMAddress: unsigned;
00159       2  0           FMSegment: unsigned;
00160       2  0           var FMValue: unsigned;
00161       1  0           var FMStopCode:char); external;
00162       1  0
00163       2  0 procedure PushFlags(PFFlags: FlagType;
00164       1  0           var PFStopCode: char); external;
00165       1  0
00166       1  0 procedure PushIntersegRA( var PIRAStopCode: char); external;
```

```
-LINE-IDC-PL-SL-

00168      1  1 begin
00169      1  1
00170      1  1 writeln('Interrupt Type  ,InterruptType:1);
00171      1  1
00172      1  1     PushFlags(Flags, IStopCode);
00173      1  1     Flags.Interrupt := Clear;
00174      1  1     Flags.Trap := Clear;
00175      1  1     PushIntersegRA( IStopCode);
00176      1  1     Address := InterruptType * 4;
00177      1  1     FetchMemory(Address, BaseSegment, LowValue, IStopCode);
00178      1  1     FetchMemory((Address + 1), BaseSegment, HighValue, IStopCode);
00179      1  1     IP := uor( uand(( HighValue * WordMultiplier), High8Bits), LowValue);
00180      1  1     FetchMemory((Address + 2), BaseSegment, LowValue, IStopCode);
00181      1  1     FetchMemory((Address + 3), BaseSegment, HighValue, IStopCode);
00182      1  1     NewCS := uor( uand(( HighValue * WordMultiplier), High8Bits), LowValue);
00183      1  1     StoreRegPtr(CSWidth, CSID, NewCS, IStopCode);
00184      0  0 end;
00185      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	343	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	16	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS INTERRUPT.PAS

  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
  /OPTIMIZE
  /STANDARD=NONE
  /TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
  /USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
  /NOANALYSIS_DATA
  /NOENVIRONMENT
  /LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]INTERRUPT.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]INTERRUPT.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	406	00:00.5	00:04.3
Source Analysis	240	00:00.7	00:03.8
Source Listing	14	00:00.4	00:02.4
Tree Construction	96	00:00.1	00:00.1
Flow Analysis	33	00:00.0	00:00.0
Value Propagation	9	00:00.0	00:00.0
Profit Analysis	41	00:00.0	00:00.3
Context Analysis	169	00:00.4	00:00.7
Name Packing	6	00:00.0	00:00.0
Code Selection	72	00:00.1	00:00.4
Final	68	00:00.1	00:02.1
TOTAL	1158	00:02.3	00:14.2

COMPILATION STATISTICS

```
  CPU Time:       00:02.3         (4826 Lines/Minute)
```

```
-LINE-IDC-PL-SL-

00001   C  0  0 { Title: IOMapGen
00002   C  0  0
00003   C  0  0   Purpose: Generate a map correlating the Input and Output Port Addresses
00004   C  0  0           with the data array indices.
00005   C  0  0
00006   C  0  0   Author: William A. Chapman      Date: May 11, 1986
00007   C  0  0
00008   C  0  0   Inputs: The operator is asked to supply the total number of
00009   C  0  0           input and output ports and then the individual port addresses.
00010   C  0  0
00011   C  0  0   Outputs: The map is stored on disk for the emulator.
00012   C  0  0
00013   C  0  0   Procedures Invoked: HexIn
00014   C  0  0           FileExists.FOR
00015   C  0  0 }
00016      0  0 program IOMapGen(input, output, ASCIIMapFile, IOMapFile);
00017      0  0
00018      0  0 const
00019      0  0     %include '[-]Const.defn/list'
00020 I    0  0     FirstOpcodeValue = 0;
00021 I    0  0     LastOpcodeValue = 255;
00022 I    0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00023 I    0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00024 I    0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00025 I    0  0     Bit8Multiplier = %X'100';
00026 I    0  0     Bit8Divisor    = %X'100';
00027 I    0  0     WordMultiplier = %X'100';
00028 I    0  0
00029 I    0  0     EightBits = 0;                       {width for 8 bits}
00030 I    0  0     SixteenBits = 1;                     {width for 16 bits}
00031 I    0  0
00032 I    0  0     SimpleMode = %B'0';
00033 I    0  0     SimpleRM = %B'110';
00034 I    0  0
00035 I    0  0     LowMemoryLimit = 0;                  {low limit on memory array}
00036 I    0  0     HighMemoryLimit = 2048;              {high limit on memory array}
00037 I    0  0
00038 I    0  0     FilenameLength = 40;
00039      0  0
00040      0  0     %include '[-]IOConst.defn/list'
00041 I    0  0     KeyboardInPortNumber  = %X'FF';
00042 I    0  0     CRTOutPortNumber       = %X'FF';
00043 I    0  0
00044 I    0  0     FirstIOPortIndex   = 1;
00045 I    0  0     MaxNumberofIOPorts = 8;
```

```
00046 I    0  0
00047 I    0  0     FirstIODataPoint   = 1;
00048 I    0  0     MaxNumberofIOWords = 64;
00049 I    0  0     MaxNumberofIOBytes = 256;     {4 times the number of words}
00050 I    0  0
00051 I    0  0     InputIndicator  = 'I';
00052 I    0  0     OutputIndicator = 'O';
00053 I    0  0     IOIndicator     = 'B';
00054 I    0  0
00055 I    0  0     LowerPortIndicator = 'L';
```

```
-LINE-IDC-PL-SL-

00056 I     0  0     UpperPortIndicator = 'U';
00057 I     0  0
00058 I     0  0     InvalidAddress = - 1;
00059 I     0  0     InvalidLocation = - 1;
00060 I     0  0     InvalidIndex = - 1;
00061       0  0
00062       0  0     %include  [-]IOMapGen.version/list'
00063 I     0  0     VersionNo = 1;
00064 I     0  0     ReleaseNo = 2;
00065       0  0
00066       0  0     FF = 12;
00067       0  0
00068       0  0 type
00069       0  0     %include  [-]Type.defn/nolist'
00109   C   0  0 {   FileName = packed array [1..FilenameLength] of char;
00110   C   0  0     Letters = set of 'A'..'z';]
00111       0  0
00112       0  0     %include  [-]IOType.defn/list'
00113 I     0  0     PortEntry = record
00114 I     0  0   Address: unsigned;
00115 I     0  0   AuxIndex: integer;
00116 I     0  0   DataIndex: integer;
00117 I     0  0   InputIndex: integer;
00118 I     0  0   PortWidth: integer;
00119 I     0  0   LowerUpperIndicator: char;
00120 I     0  0   InOutIndicator: char
00121 I     0  0     end;
00122 I     0  0
00123 I     0  0     PortMap = array[ FirstIOPortIndex..MaxNumberOfIOPorts] of PortEntry;
```

```
-LINE-IDC-PL-SL-

00125      0  0 var
00126      0  0     IOPortMap: [global] PortMap;
00127      0  0
00128      0  0     NumberOfIOPorts: integer;
00129      0  0
00130      0  0     PortIndex: integer;
00131      0  0
00132      0  0     PortAddressToDelete: unsigned;
00133      0  0     PortAddressToReset:  unsigned;
00134      0  0     PortAddressPresent:  boolean;
00135      0  0     PortAdded: boolean;
00136      0  0
00137      0  0     ResetIndex:     char;
00138      0  0     TempDataIndex:  integer;
00139      0  0     TempInputIndex: integer;
00140      0  0
00141      0  0     DeleteUpperIndex: integer;
00142      0  0
00143      0  0     ModifyCommand: char;
00144      0  0     Valid:         boolean;
00145      0  0     Finished:      boolean;
00146      0  0
00147      0  0     Command: char;
00148      0  0     Done:    boolean;
00149      0  0
00150      0  0     QuitResponse: char;
00151      0  0
00152      0  0     Loop: integer;
00153      0  0
00154      0  0     IOMapFile: file of PortEntry;
00155      0  0
00156      0  0     ASCIIMapFile:    text;
00157      0  0     IOMapFileName:   FileName;
00158      0  0     ListFileName:    FileName;
00159      0  0
00160      0  0     IODesignator: char;
00161      0  0     IOClasses: Letters;
00162      0  0
00163      0  0     MapAccessed:boolean;
00164      0  0
00165      0  0     Exists: boolean;
00166      0  0
00167      0  0
00168      0  0 function FileExists( FEFileName: Filename): boolean; Fortran;
```

```
00169      0  0
00170      0  0 procedure HexIn(var HIValue: unsigned); external;
```

```
-LINE-IDC-PL-SL-

00172       0  0 %include 'SearchForPortAddress.subpas/list'
00173 I C   0  0 { Title: SearchForPortAddress
00174 I C   0  0
00175 I C   0  0   Purpose: Find a specified port address
00176 I C   0  0
00177 I C   0  0   Author: William A. Chapman      Date: May 11, 1986
00178 I C   0  0
00179 I C   0  0   Inputs: PortMap
00180 I C   0  0           Target Port Address
00181 I C   0  0
00182 I C   0  0   Outputs: PortFound - boolean flag
00183 I C   0  0           PortLocation - index of port
00184 I C   0  0
00185 I C   0  0   Procedures Called: none
00186 I C   0  0 }
00187 I     1  0 procedure SearchForPortAddress( SFPATargetPortAddress: unsigned;
00188 I     1  0           var SFPASearchPortMap: PortMap;
00189 I     1  0           var SFPAPortFound: boolean;
00190 I     1  0           var SFPAPortLocation: integer);
00191 I     1  0
00192 I     1  0 var
00193 I     1  0     SFPAIndex: integer;
00194 I     1  0     SFPADone: boolean;
00195 I     1  0
00196 I     1  1 begin
00197 I     1  1     SFPAIndex := FirstIOPortIndex;
00198 I     1  1     SFPAPortFound := false;
00199 I     1  1     SFPADone := false;
00200 I     1  1
00201 I     1  2     while (not SFPADone) do begin
00202 I     1  3   if (SFPASearchPortMap[ SFPAIndex].Address = SFPATargetPortAddress) then begin
00203 I     1  3       SFPAPortFound := true;
00204 I     1  3       SFPAPortLocation := SFPAIndex;
00205 I     1  3       SFPADone := true;
00206 I     1  3   end
00207 I     1  2   else SFPAIndex := SFPAIndex + 1;
00208 I     1  3   if (SFPAIndex ) MaxNumberofIOPorts) then begin
00209 I     1  3       SFPAPortFound := false;
00210 I     1  3       SFPAPortLocation := InvalidLocation;
00211 I     1  3       SFPADone := true;
00212 I     1  2   end;
00213 I     1  1     end;  {while (not SFPADone)}
00214 I     0  0 end; {SearchForPortAddress}
```

```
-LINE-IDC-PL-SL-

00216   C  0  0 { Title: FindFreePort
00217   C  0  0
00218   C  0  0   Purpose: Locate first available free port
00219   C  0  0
00220   C  0  0   Author: William A. Chapman      Date: May 11, 1986
00221   C  0  0
00222   C  0  0   Inputs: PortMap
00223   C  0  0
00224   C  0  0   Outputs: FreePortAvailable - status
00225   C  0  0            FreePortLocation
00226   C  0  0
00227   C  0  0   Procedures Invoked: none
00228   C  0  0
00229   C  0  0 }
00230      1  0 procedure FindFreePort( var FFPFreePortAvailable: boolean;
00231      1  0           var FFPFreePortLocation: integer;
00232      1  0           FFPFindPortMap: PortMap);
00233      1  0
00234      1  0 var
00235      1  0     FFPDone: boolean;
00236      1  0
00237      1  1 begin
00238      1  1     FFPFreePortLocation := FirstIOPortIndex;
00239      1  1     FFPDone := false;
00240      1  1
00241      1  2     while (not FFPDone) do begin
00242      1  3   if (FFPFindPortMap[ FFPFreePortLocation].Address = InvalidAddress) then begin
00243      1  3       FFPFreePortAvailable := true;
00244      1  3       FFPDone := true;
00245      1  3   end
00246      1  2   else FFPFreePortLocation := FFPFreePortLocation + 1;
00247      1  3   if (FFPFreePortLocation > MaxNumberofIOPorts) then begin
00248      1  3       FFPFreePortAvailable := false;
00249      1  3       FFPFreePortLocation := InvalidLocation;
00250      1  3       FFPDone := true;
00251      1  2   end;
00252      1  1     end; {while (not FFPDone)}
00253      0  0 end; {procedure FindFreePort}
```

```
-LINE-IDC-PL-SL-

00255   C  0  0 [ Title: AddAPort
00256   C  0  0
00257   C  0  0   Purpose: Verify that the 8 bit port address is not already assigned
00258   C  0  0            and if it is not, then allocate first available free port.
00259   C  0  0
00260   C  0  0   Author: William A. Chapman      Date: May 11, 1986
00261   C  0  0
00262   C  0  0   Inputs: First port index
00263   C  0  0            Address of Port to be added
00264   C  0  0            Width of port to be added
00265   C  0  0
00266   C  0  0   Outputs: Port address added to map if not already present and
00267   C  0  0            sufficient room
00268   C  0  0
00269   C  0  0   Procedures Invoked: SearchForPortAddress
00270   C  0  0            FindFreePort
00271   C  0  0 }
00272      1  0 procedure AddAPort( var AAPAddPortMap: PortMap;
00273      1  0            AAPAddPortAddress: unsigned;
00274      1  0            var AAPFreePortIndex: integer;
00275      1  0            var AAPPortAdded: boolean);
00276      1  0
00277      1  0
00278      1  0 var
00279      1  0     AAPPortAvailable: boolean;
00280      1  0
00281      1  0
00282      1  1 begin
00283      1  1     AAPPortAdded := false;
00284      1  1     FindFreePort( AAPPortAvailable, AAPFreePortIndex, AAPAddPortMap);
00285      1  2     if (AAPPortAvailable) then begin
00286      1  2   AAPAddPortMap[ AAPFreePortIndex].Address := AAPAddPortAddress;
00287      1  2   AAPPortAdded := true;
00288      1  2     end
00289      1  2     else begin
00290      1  2   writeln('Output port map is full.  No room for additional ports. );
00291      1  2   writeln('See operators manual for additional instructions.');
00292      1  1     end;
00293      0  0 end;
```

```
-LINE-IDC-PL-SL-

00295       0  0     %include 'UpperCaseLetter.subpas/list'
00296 I C   0  0 { Title: UpperCaseLetter
00297 I C   0  0
00298 I C   0  0   Purpose: Convert lowercase letter to uppercase
00299 I C   0  0
00300 I C   0  0   Author: William A. Chapman      Date: October 26, 1986
00301 I C   0  0
00302 I C   0  0   Inputs: Upper or lower case letter
00303 I C   0  0
00304 I C   0  0   Outputs: All lowercase letters are converted to uppercase
00305 I C   0  0
00306 I C   0  0   Procedures Invoked: none
00307 I C   0  0 }
00308 I     1  0 procedure UpperCaseLetter( var UPLetter: char);
00309 I     1  0
00310 I     1  1 begin
00311 I     1  1     if ((( ord( UPLetter) - ord( 'a')) >= 0 ) and (( ord( UPLetter) - ord( 'z')) <= 0 ))
00312 I     1  1   then UPLetter := chr( ord( UPLetter) - (ord( 'a') - ord( 'A')));
00313 I     0  0 end;
```

```
-LINE-IDC-PL-SL-

00315   C  0  0 { Title: AddPortToMap
00316   C  0  0
00317   C  0  0   Purpose: Gather information about a port and add port to map
00318   C  0  0            if the information is valid.
00319   C  0  0
00320   C  0  0   Author: William A. Chapman      Date: October 26, 1986
00321   C  0  0
00322   C  0  0   Inputs: IOPortMap
00323   C  0  0
00324   C  0  0   Outputs: IOPortMap is updated
00325   C  0  0
00326   C  0  0   Procedures Invoked: AddAPort
00327   C  0  0 }
00328      1  0 procedure AddPortToMap( var APTMPortMap: PortMap;
00329      1  0           var APTMPortIndex: integer;
00330      1  0           var APTMPortAdded: boolean);
00331      1  0
00332      1  0 var
00333      1  0     NewPortAddress: unsigned;
00334      1  0     NewPortWidth: integer;
00335      1  0     NewPortIndex: integer;
00336      1  0     LowerPortIndex: integer;
00337      1  0     UpperPortIndex: integer;
00338      1  0     WidthValid: boolean;
00339      1  0
00340      1  0     LowerNewPortAddress: unsigned;
00341      1  0     LowerPortAdded: boolean;
00342      1  0     UpperNewPortAddress: unsigned;
00343      1  0     UpperPortAdded: boolean;
00344      1  0
00345      1  0     APTMPortAddressAlreadyThere: boolean;
00346      1  0     APTMPortLocation: integer;
00347      1  0
00348      1  1 begin
00349      1  1     WidthValid := false;
00350      1  2     while (not WidthValid) do begin
00351      1  2
00352      1  2   writeln;
00353      1  2   write( '   Enter address of new port (in hex): ');
00354      1  2   HexIn( NewPortAddress);
00355      1  2   SearchForPortAddress( NewPortAddress, APTMPortMap,
00356      1  2       APTMPortAddressAlreadyThere, APTMPortLocation);
00357      1  2   if (APTMPortAddressAlreadyThere) then writeln('Port address already assigned. )
00358      1  3   else begin
```

```
00359      1  3
00360      1  3       write(       Is this an 8 bit or 16 bit  port  ?  );
00361      1  3       readln(NewPortWidth);
00362      1  3       writeln( '   Is this an Input port (I), ');
00363      1  3       write(        an Output port (O) or Both  (B)  ? ');
00364      1  3       readln( IODesignator);
00365      1  3       UpperCaseLetter( IODesignator);
00366      1  4       if (IODesignator in IOClasses) then begin
```

```
-LINE-IDC-PL-SL-

00368       1  5           case NewPortWidth of
00369       1  5
00370       1  6           8: begin {8 bit wide port}
00371       1  6               AddAPort( APTMPortMap, NewPortAddress, NewPortIndex, APTMPortAdded);
00372       1  6               WidthValid := true;
00373       1  7               if (APTMPortAdded = true) then begin
00374       1  7                   APTMPortIndex := APTMPortIndex + 1;
00375       1  7                   APTMPortMap[ NewPortIndex].InOutIndicator := IODesignator;
00376       1  6               end;
00377       1  5           end; {8 bit wide port}
00378       1  5
00379       1  6           16: begin {16 bit wide port}
00380       1  6               LowerNewPortAddress := NewPortAddress;
00381       1  6               UpperNewPortAddress := NewPortAddress + 1;
00382       1  6               AddAPort( APTMPortMap, LowerNewPortAddress, LowerPortIndex, LowerPortAdded);
00383       1  7               if (LowerPortAdded = true) then begin
00384       1  7                   AddAPort( APTMPortMap, UpperNewPortAddress, UpperPortIndex, UpperPortAdded);
00385       1  8                   if (UpperPortAdded = true) then begin
00386       1  8                       APTMPortMap[ LowerPortIndex].PortWidth := SixteenBits;
00387       1  8                       APTMPortMap[ UpperPortIndex].PortWidth := SixteenBits;
00388       1  8                       APTMPortMap[ LowerPortIndex].AuxIndex := UpperPortIndex;
00389       1  8                       APTMPortMap[ UpperPortIndex].AuxIndex := LowerPortIndex;
00390       1  8                       APTMPortMap[ UpperPortIndex].LowerUpperIndicator := UpperPortIndicator;
00391       1  8                       APTMPortMap[ LowerPortIndex].InOutIndicator := IODesignator;
00392       1  8                       APTMPortMap[ UpperPortIndex].InOutIndicator := IODesignator;
00393       1  8                       APTMPortIndex := APTMPortIndex + 2;
00394       1  7                   end;
00395       1  6               end;
00396       1  6               WidthValid := true;
00397       1  5           end; {16 bit wide port}
00398       1  5
00399       1  5           otherwise writeln('Invalid port width. Try again.');
00400       1  5
00401       1  4           end; {case NewPortWidth}
00402       1  4
00403       1  4       end
00404       1  3       else writeln( 'Invalid IO Designator. Try again. );
00405       1  2   end; {else begin AddressPortAlreadyPresent}
00406       1  2
00407       1  1     end; {while (not WidthValid)}
00408       0  0 end; {AddPortToMap}
```

```
-LINE-IDC-PL-SL-

00410       0  1 begin
00411       0  1     writeln;  writeln;
00412       0  1     writeln(              Intel 8086 Simulator - I/O Map Generation Utility');
00413       0  1     writeln(                                Version ', VersionNo:1, '.', ReleaseNo:1 );
00414       0  1     writeln;  writeln;
00415       0  1
00416       0  1     Done := false;
00417       0  1     IOClasses := ['B', 'I', 'O', 'b', 'i', 'o'];
00418       0  1     MapAccessed := false;
00419       0  1
00420       0  1     writeln;
00421       0  1     writeln('Generate or Modify IO Map');
00422       0  2     while (not Done) do begin
00423       0  2   writeln; writeln;
00424       0  2   writeln( 'Enter desired operation:  );
00425       0  2   writeln(    Create (C), Read (R), Modify (M), Display (D)');
00426       0  2   write(      Print (P), Write (W), Quit (Q), Exit (E): ');
00427       0  2   readln(Command);
00428       0  2   writeln; writeln;
00429       0  2
00430       0  3   case Command of
00431       0  4     'C','c': begin {create]
00432       0  4       write('Enter name of file to contain map:  );
00433       0  4       readln( IOMapFilename);
00434       0  4       open( IOMapFile, IOMapFilename, new);       {this is the only place an]
00435       0  4       close( IOMapFile);  {open "new" should be issued on this data file]
00436       0  4       writeln;  writeln;
00437       0  4
00438       0  4       Valid := false;
00439       0  5       while (not Valid) do begin
00440       0  5           writeln( 'Enter number of IO ports desired');
00441       0  5           writeln(     Each 16 bit port counts as 2 port');
00442       0  5           write(       Total number of ports must be ');
00443       0  5           write( MaxNumberofIOPorts:1,' or less : ');
00444       0  5           readln( NumberofIOPorts);
00445       0  5           if (NumberofIOPorts <= MaxNumberofIOPorts) then Valid := true
00446       0  5           else writeln('Too many ports requested.  Try again.');
00447       0  4       end; {while Valid]
00448       0  4
00449       0  4       for PortIndex := FirstIOPortIndex to MaxNumberofIOPorts do
00450       0  5           with IOPortMap[PortIndex] do begin
00451       0  5               Address := InvalidAddress;
00452       0  5               AuxIndex := InvalidIndex;
00453       0  5               DataIndex := FirstIODataPoint;
```

```
00454    0  5              InputIndex := FirstIODataPoint;
00455    0  5              PortWidth := EightBits;
00456    0  5              LowerUpperIndicator := LowerPortIndicator;
00457    0  5              InOutIndicator := InputIndicator;
00458    0  4          end; {with IOPortMap[PortIndex]}
00459    0  4
00460    0  4      writeln; writeln;
00461    0  4      writeln('Enter ',NumberofIOPorts :1,  port IDs');
00462    0  4      PortIndex := FirstIOPortIndex;
00463    0  5      while (PortIndex <= NumberofIOPorts) do begin
00464    0  5          AddPortToMap( IOPortMap, PortIndex, PortAdded);
```

```
-LINE-IDC-PL-SL-

00465      0  5          writeln;
00466      0  4        end; {while PortIndex <= NumberofPorts}
00467      0  4        MapAccessed := true;
00468      0  3      end; {case C}
```

```
-LINE-IDC-PL-SL-

00470       0  4     'M','m': begin {modify}
00471       0  4       Finished := false;
00472       0  5       while (not Finished) do begin
00473       0  5           writeln; writeln;
00474       0  5           writeln('Do you want to :');
00475       0  5           writeln( '   Add  (A)  a  port, Delete (D) a port  );
00476       0  5           write(       Reset (R) an index, or are you Finished (F) : ');
00477       0  5           readln(ModifyCommand);
00478       0  5           writeln;
00479       0  5
00480       0  6           case ModifyCommand of
00481       0  6             'A','a': {Add a port}
00482       0  6               AddPortToMap( IOPortMap, PortIndex, PortAdded);
00483       0  6
00484       0  7             'D','d': begin {delete a port}
00485       0  7               write('Enter address of port to be deleted (in hex):  );
00486       0  7               HexIn(PortAddressToDelete);
00487       0  7               SearchForPortAddress( PortAddressToDelete, IOPortMap,
00488       0  7                   PortAddressPresent, PortIndex);
00489       0  7               if (PortAddressPresent) then
00490       0  8                   with IOPortMap[PortIndex] do begin
00491       0  8                       Address := InvalidAddress;
00492       0  8                       DataIndex := FirstIODataPoint;
00493       0  8                       InputIndex := FirstIODataPoint;
00494       0  9                       if (PortWidth = SixteenBits) then begin
00495       0  9                           DeleteUpperIndex := AuxIndex;
00496       0 10                           with IOPortMap[DeleteUpperIndex] do begin
00497       0 10                               Address := InvalidAddress;
00498       0 10                               DataIndex := FirstIODataPoint;
00499       0 10                               InputIndex := FirstIODataPoint;
00500       0 10                               AuxIndex := InvalidIndex;
00501       0 10                               PortWidth := EightBits;
00502       0 10                               LowerUpperIndicator := LowerPortIndicator;
00503       0 10                               InOutIndicator := InputIndicator;
00504       0  9                           end; {with IOPortMap[DeleteUpperIndex]}
00505       0  8                       end; {if (Width = SixteenBits)}
00506       0  8                       AuxIndex := InvalidIndex;
00507       0  8                       PortWidth := EightBits;
00508       0  8                       LowerUpperIndicator := LowerPortIndicator;
00509       0  8                       InOutIndicator := InputIndicator;
00510       0  8                   end {with IOPortMap}
00511       0  8               else begin
00512       0  8                   write('Port address  ,hex( PortAddressToDelete, 4,4));
00513       0  8                   writeln(' not present.');
```

```
00514      0  7              end; {else begin}
00515      0  6            end; {case D - delete a port}
```

```
-LINE-IDC-PL-SL-

00517       0  7            'R','r': begin {reset data index for a port}
00518       0  7              write('Enter address of port to be reset (in hex): ');
00519       0  7              HexIn(PortAddressToReset);
00520       0  7              SearchForPortAddress( PortAddressToReset,IOPortMap, PortAddressPresent, PortIndex);
00521       0  8              if (PortAddressPresent) then begin
00522       0  9                  with IOPortMap[PortIndex] do begin
00523       0  9                    writeln( '    Do you want to reset a Data index (D), );
00524       0  9                    write(       an Input (I) index, or Both (B) : ');
00525       0  9                    readln( ResetIndex);
00526       0  9
00527       0 10                    case ResetIndex of
00528       0 10
00529       0 11                    'D','d': begin {reset Data index}
00530       0 11                      write( 'Enter new value for Data Index (in decimal): ');
00531       0 11                      readln( TempDataIndex);
00532       0 11                      if ((TempDataIndex >= FirstIODataPoint) and
00533       0 12                        (TempDataIndex <= MaxNumberofIOBytes)) then begin
00534       0 12                          DataIndex := TempDataIndex;
00535       0 12                          if (PortWidth = SixteenBits) then
00536       0 12                            IOPortMap[ AuxIndex].DataIndex := TempDataIndex;
00537       0 11                      end;
00538       0 10                    end; {case reset Data index}
00539       0 10
00540       0 11                    'I','i': begin {reset Input index}
00541       0 11                      write( 'Enter new value for Input Index (in decimal): ');
00542       0 11                      readln( TempInputIndex);
00543       0 11                      if ((TempInputIndex >= FirstIODataPoint) and
00544       0 12                        (TempInputIndex <= MaxNumberofIOBytes)) then begin
00545       0 12                          InputIndex := TempInputIndex;
00546       0 12                          if (PortWidth = SixteenBits) then
00547       0 12                            IOPortMap[ AuxIndex].InputIndex := TempInputIndex;
00548       0 11                      end;
00549       0 10                    end; {case reset Input index}
```

```
-LINE-IDC-PL-SL-

00551       0 11                    'B','b': begin {reset both Data and Input index}
00552       0 11                      write( 'Enter new value for Data Index (in decimal):  );
00553       0 11                      readln( TempDataIndex);
00554       0 11                      if ((TempDataIndex >= FirstIODataPoint) and
00555       0 12                        (TempDataIndex <= MaxNumberofIOBytes)) then begin
00556       0 12                          DataIndex := TempDataIndex;
00557       0 12                          if (PortWidth = SixteenBits) then
00558       0 12                            IOPortMap[ AuxIndex].DataIndex := TempDataIndex;
00559       0 11                      end;
00560       0 11
00561       0 11                      write( 'Enter new value for Input Index (in decimal):  );
00562       0 11                      readln( TempInputIndex);
00563       0 11                      if ((TempInputIndex >= FirstIODataPoint) and
00564       0 12                        (TempInputIndex <= MaxNumberofIOBytes)) then begin
00565       0 12                          InputIndex := TempInputIndex;
00566       0 12                          if (PortWidth = SixteenBits) then
00567       0 12                            IOPortMap[ AuxIndex].InputIndex := TempInputIndex;
00568       0 11                      end;
00569       0 10                    end; {case reset Both Data and Input index]
00570       0 10
00571       0 10                    otherwise writeln( 'Bad input. Try again.');
00572       0 10
00573       0  9                    end; {case ResetIndex]
00574       0  8                  end; {with IOPortMap]
00575       0  8                end
00576       0  8                else begin
00577       0  8                    write('Port address  ,hex( PortAddressToReset, 4,4));
00578       0  8                    writeln(' not present.');
00579       0  7                end; {else begin]
00580       0  6              end; {case R - reset a port]
00581       0  6
00582       0  6              'F','f': Finished := true;  {finished]
00583       0  5            end; {case ModifyCommand]
00584       0  4      end;  {while not Finished]
00585       0  3    end; {case M}
```

```
-LINE-IDC-PL-SL-

00587      0  4     'R','r': begin [read]
00588      0  4       write('Enter name of file containing map:  );
00589      0  4       readln( IOMapFilename);
00590      0  4       Exists := FileExists( %descr IOMapFileName);
00591      0  5       if (Exists = true) then begin
00592      0  5           open( IOMapFile, IOMapFilename, old);
00593      0  5           reset( IOMapFile);
00594      0  5           for Loop := FirstIOPortIndex to MaxNumberofIOPorts do
00595      0  5               read( IOMapFile, IOPortMap[Loop]);
00596      0  5           MapAccessed := true;
00597      0  5           close( IOMapFile);
00598      0  5       end
00599      0  4       else writeln( 'File ', IOMapFilename,   does not exist');
00600      0  3     end; {case R]
00601      0  3
00602      0  3
00603      0  4           'D','d': begin {Display]
00604      0  4       writeln( chr( FF));
00605      0  4       writeln( 'Description of IO Port Map');
00606      0  4       writeln;  writeln;
00607      0  4       writeln( 'Port   Port   Port  Upper   Aux   Data   Input I / O');
00608      0  4       writeln( 'Index Address Width Lower  Index  Index  Index State');
00609      0  4       for PortIndex := FirstIOPortIndex to MaxNumberofIOPorts do
00610      0  5       with IOPortMap[PortIndex] do begin
00611      0  5           write( PortIndex:3);
00612      0  5           write('    ',hex( Address, 4,4),      );
00613      0  5           if (PortWidth = EightBits) then write(' 8')
00614      0  5           else write('16');
00615      0  5           write('     ', LowerUpperIndicator);
00616      0  5           write('    ', AuxIndex:3);
00617      0  5           write('    ', DataIndex:4);
00618      0  5           write('   ', InputIndex:4);
00619      0  5           write('   ');
00620      0  6           case InOutIndicator of
00621      0  6
00622      0  6           'I','i': write( 'Input');
00623      0  6
00624      0  6           'O','o': write( 'Output');
00625      0  6
00626      0  6           'B','b': write( 'Both');
00627      0  6
00628      0  5           end; [case InOutIndicator]
00629      0  5           writeln;
00630      0  4       end;
```

```
00631      0  3     end; {case D]
```

```
-LINE-IDC-PL-SL-

00633       0  4           'W','w': begin {write]
00634       0  5         if (MapAccessed = true) then begin
00635       0  5             open( IOMapFile, IOMapFileName, old);
00636       0  5             rewrite( IOMapFile);
00637       0  5             for Loop := FirstIOPortIndex to MaxNumberofIOPorts do
00638       0  5                 write( IOMapFile, IOPortMap[Loop]);
00639       0  5             close( IOMapFile);
00640       0  5         end
00641       0  4         else writeln( 'IO map file never accessed');
00642       0  3       end; {case W]
00643       0  3
00644       0  4           'P','p': begin {print]
00645       0  4         write( 'Enter name of file to contain map:  );
00646       0  4         readln( ListFilename);
00647       0  4         open( ASCIIMapFile, ListFilename, new);
00648       0  4         rewrite( ASCIIMapFile);
00649       0  4
00650       0  4         writeln( ASCIIMapFile,'Description of IO Port Map');
00651       0  4         writeln( ASCIIMapFile);
00652       0  4         writeln( ASCIIMapFile);
00653       0  4
00654       0  4         writeln( ASCIIMapFile,'Port   Port   Port  Upper   Aux   Data   Input I / O');
00655       0  4         writeln( ASCIIMapFile,'Index Address Width Lower  Index  Index  Index State');
00656       0  4         for PortIndex := FirstIOPortIndex to MaxNumberofIOPorts do
00657       0  5         with IOPortMap[PortIndex] do begin
00658       0  5             write( ASCIIMapFile, PortIndex:3);
00659       0  5             write( ASCIIMapFile, '    ',hex( Address, 4,4),     ');
00660       0  5             if (PortWidth = EightBits) then write( ASCIIMapFile, ' 8')
00661       0  5             else write( ASCIIMapFile, '16');
00662       0  5             write( ASCIIMapFile, '     ', LowerUpperIndicator);
00663       0  5             write( ASCIIMapFile,       , AuxIndex:3);
00664       0  5             write( ASCIIMapFile,     ', DataIndex:4);
00665       0  5             write( ASCIIMapFile,     ', InputIndex:4);
00666       0  5             write( ASCIIMapFile,      );
00667       0  6             case InOutIndicator of
00668       0  6
00669       0  6             'I','i': write( ASCIIMapFile, 'Input');
00670       0  6
00671       0  6             'O','o': write( ASCIIMapFile, 'Output');
00672       0  6
00673       0  6             'B','b': write( ASCIIMapFile, 'Both');
00674       0  6
00675       0  5             end; {case InOutIndicator]
00676       0  5             writeln( ASCIIMapFile);
```

```
00677      0  4       end;
00678      0  4       close( ASCIIMapFile);
00679      0  3     end; {case P}
```

```
-LINE-IDC-PL-SL-

00681       0  4          'Q','q': begin {exit without writing]
00682       0  4        writeln; writeln;
00683       0  4        writeln( 'This will exit without saving the map file.');
00684       0  4        write(   'Are you sure you want to do this [Y/N] ? ');
00685       0  4        readln( QuitResponse);
00686       0  4        if ((QuitResponse = 'Y') or (QuitResponse = 'y')) then Done := true;  {exit]
00687       0  3      end; {case Q]
00688       0  3
00689       0  4          'E','e': begin {exit]
00690       0  4        Done := true;
00691       0  5        if (MapAccessed = true) then begin
00692       0  5            open( IOMapFile, IOMapFileName, old);
00693       0  5            rewrite( IOMapFile);
00694       0  5            for Loop := FirstIOPortIndex to MaxNumberofIOPorts do
00695       0  5                write( IOMapFile, IOPortMap[Loop]);
00696       0  5            close( IOMapFile);
00697       0  5        end
00698       0  4        else writeln( 'IO map file never accessed');
00699       0  3      end; {case E]
00700       0  3
00701       0  3          otherwise writeln('Bad input try again');
00702       0  2          end; {case]
00703       0  1     end; {while not done]
00704       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	8053	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	212	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

PAS/LIS IOMAPGEN

/CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
/DEBUG=(NOSYMBOLS,TRACEBACK)
/SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
/OPTIMIZE
/STANDARD=NONE
/TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
/USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
/NOANALYSIS_DATA
/NOENVIRONMENT
/LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]IOMAPGEN.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]IOMAPGEN.OBJ;22
/NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	408	00:00.6	00:05.4
Source Analysis	480	00:02.2	00:12.1
Source Listing	14	00:01.1	00:06.4
Tree Construction	214	00:00.9	00:02.7
Flow Analysis	84	00:00.5	00:01.6
Value Propagation	11	00:00.1	00:00.9
Profit Analysis	99	00:00.4	00:01.4
Context Analysis	257	00:04.2	00:14.4
Name Packing	7	00:00.3	00:00.9
Code Selection	126	00:01.1	00:04.8
Final	110	00:02.0	00:09.4
TOTAL	1821	00:13.2	01:00.2

COMPILATION STATISTICS

CPU Time: 00:13.2 (3193 Lines/Minute)

```
-LINE-IDC-PL-SL-

00001    C  0  0 { Title: Keyword Look-Up-Table Generator for the Debugger
00002    C  0  0
00003    C  0  0   Purpose: Generate and maintain the Debugger Keyword Look-Up-Table
00004    C  0  0
00005    C  0  0   Author: William A. Chapman      Date: July 17, 1986
00006    C  0  0
00007    C  0  0   Inputs: Provided by the user
00008    C  0  0
00009    C  0  0   Outputs: An external file "KeywordLUTFile" is generated.
00010    C  0  0           An optional file "KeywordLUTList" (a printable version)
00011    C  0  0           can also be generated.
00012    C  0  0
00013    C  0  0   Procedures Invoked: none
00014    C  0  0 }
00015       0  0 program KeywordLUTGen(input, output,KeywordLUTFile, KeywordLUTList );
00016       0  0
00017       0  0 const
00018       0  0     %include  [-]Const.defn/list'
00019 I     0  0     FirstOpcodeValue = 0;
00020 I     0  0     LastOpcodeValue = 255;
00021 I     0  0     All16Bits = %X'FFFF';         {mask for all 16 bits}
00022 I     0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00023 I     0  0     High8Bits = %X'FF00';         {mask for high 8 bits}
00024 I     0  0     Bit8Multiplier = %X'100';
00025 I     0  0     Bit8Divisor    = %X'100';
00026 I     0  0     WordMultiplier = %X'100';
00027 I     0  0
00028 I     0  0     EightBits = 0;                      {width for 8 bits}
00029 I     0  0     SixteenBits = 1;                    {width for 16 bits}
00030 I     0  0
00031 I     0  0     SimpleMode = %B'0';
00032 I     0  0     SimpleRM = %B'110';
00033 I     0  0
00034 I     0  0     LowMemoryLimit = 0;                 {low limit on memory array}
00035 I     0  0     HighMemoryLimit = 2048;             {high limit on memory array}
00036 I     0  0
00037 I     0  0     FilenameLength = 40;
```

```
-LINE-IDC-PL-SL-

00039       0  0     %include '[-]DebugConst.defn/list'
00040 I     0  0     FirstKeywordEntry = 0;
00041 I     0  0     LastKeywordEntry  = 100;
00042 I     0  0     UndefinedName = 'Qqqqqqqq';
00043 I     0  0     UndefinedClass = 'z';
00044 I     0  0     UndefinedWidth = -1;
00045 I     0  0     UndefinedID = -2;
00046 I     0  0     PageSize = 55;
00047 I     0  0     FF = 12;
00048 I     0  0     MinID = -1;
00049 I     0  0     MaxID = 25;
00050 I     0  0     MinWidth = 0;
00051 I     0  0     MaxWidth = 3;
00052 I     0  0     SegmentOffset = 16;
00053 I     0  0
00054 I     0  0     BlankClass       =    ;
00055 I     0  0     BreakpointClass  = 'B';
00056 I     0  0     ErrorClass       = 'E';
00057 I     0  0     FlagClass        = 'F';
00058 I     0  0     InputOutputClass = 'I';
00059 I     0  0     MemoryClass      = 'M';
00060 I     0  0     NumericClass     = 'N';
00061 I     0  0     OperatorClass    = 'O';
00062 I     0  0     QualifierClass   = 'Q';
00063 I     0  0     RegisterClass    = 'R';
00064 I     0  0     StackClass       = 'S';
00065 I     0  0     TrueFalseClass   = 'T';
00066 I     0  0     UtilityClass     = 'U';
00067 I     0  0     ExecuteClass     = 'X';
00068 I     0  0
00069 I     0  0     Blank  =    ;
00070 I     0  0     Elipse =  .';
00071 I     0  0
00072 I     0  0     StartOfData = 1;
00073 I     0  0     DataArraySize = 50;
00074 I     0  0
00075 I     0  0     EndofLine = 0;
00076 I     0  0     StartofLine = 1;
00077 I     0  0
00078 I     0  0     ShortStringLength = 8;
00079 I     0  0     InputLineLength = 120;
00080 I     0  0
00081 I     0  0     BooleanID  = 1;
00082 I     0  0     ByteID     = 2;
```

```
00083 I    0  0     IntegerID  = 3;
00084 I    0  0     PointerID  = 4;
00085 I    0  0     SIntegerID = 5;
00086 I    0  0     WordID     = 6;
00087 I    0  0     LengthID   = 3;
00088 I    0  0     ToID       = 5;
00089 I    0  0
00090 I    0  0     AssignmentID = 1;
00091 I    0  0     ColonID      = 2;
00092 I    0  0     PlusID       = 3;
00093 I    0  0     MinusID      = 4;
```

```
-LINE-IDC-PL-SL-

00094 I    0  0     CommaID      = 5;
00095 I    0  0
00096 I    0  0     NullEntries = 0;
00097 I    0  0
00098 I    0  0     AddressState   = 'A';
00099 I    0  0     BooleanState   = 'B';
00100 I    0  0     ErrorState     = 'E';
00101 I    0  0     HexState       = 'H';
00102 I    0  0     IntegerState   = 'I';
00103 I    0  0     SegmentState   = 'S';
00104 I    0  0     SeparatorState = ',';
00105 I    0  0     UnaryState     = '-';
00106 I    0  0     ColonState     = ':';
00107 I    0  0
00108 I    0  0     NegativeOperator = -1;
00109 I    0  0     PositiveOperator = +1;
00110 I    0  0
00111 I    0  0     NullSegmentValue = 0;
00112 I    0  0     NullAddressValue = 0;
00113 I    0  0
00114 I    0  0     FirstBreakpoint  =  0;
00115 I    0  0     LastBreakpoint   = 15;
00116 I    0  0     AllBPIDs         = 16;
00117 I    0  0     Activate         = true;
00118 I    0  0     DeActivate       = false;
00119 I    0  0     BreakpointOpcode = %X'CC';
```

```
-LINE-IDC-PL-SL-

00121       0  0 type
00122       0  0     %include  [-]Type.defn/nolist'
00162   C   0  0 (   ZeroOne = -1..1;]
00163       0  0
00164       0  0     %include '[-]DebugType.defn/list'
00165 I     0  0     NameString = packed array[1..ShortStringLength] of char;
00166 I     0  0
00167 I     0  0     KeywordCharacteristics = record
00168 I     0  0   Name: NameString;
00169 I     0  0   Class: char;
00170 I     0  0   Width: ZeroOne;
00171 I     0  0   ID: integer;
00172 I     0  0   Value: unsigned
00173 I     0  0     end;
00174 I     0  0
00175 I     0  0     InputPointerRange = integer;
00176 I     0  0     InputLine = packed array [1..InputLineLength] of char;
00177 I     0  0
00178 I     0  0     ShortString = packed array [1..ShortStringLength] of char;
00179 I     0  0
00180 I     0  0     DataArrayType = array [StartOfData..DataArraySize] of unsigned;
00181 I     0  0
00182 I     0  0     BreakRecord = record
00183 I     0  0   Encountered: boolean;
00184 I     0  0   Activated: boolean;
00185 I     0  0   Segment: unsigned;
00186 I     0  0   Address: unsigned;
00187 I     0  0   PhysicalAddress: unsigned;
00188 I     0  0   Code: unsigned
00189 I     0  0     end;
00190 I     0  0
00191 I     0  0     BreakPointTableType = array [FirstBreakpoint..LastBreakpoint] of BreakRecord;
```

```
-LINE-IDC-PL-SL-

00193       0  0 var
00194       0  0     ValidClasses: LettersAndNumbers;
00195       0  0
00196       0  0     ListFile: Filename;
00197       0  0     KeywordLUTFile: file of KeywordCharacteristics;
00198       0  0     KeywordLUTList: text;
00199       0  0
00200       0  0     Keyword: KeywordCharacteristics;
00201       0  0     KeywordLUTable: array [FirstKeywordEntry..LastKeywordEntry]
00202       0  0   of KeywordCharacteristics;
00203       0  0
00204       0  0     NewName: NameString;
00205       0  0     NewID: integer;
00206       0  0     NewClass: char;
00207       0  0     NewWidth: integer;
00208       0  0     Index: integer;
00209       0  0
00210       0  0     Command: char;
00211       0  0     Continue: boolean;
00212       0  0     Correct: boolean;
```

```
-LINE-IDC-PL-SL-

00214       0  0     %include 'UpperCaseString.subpas/list'
00215 I C   0  0 { Title: UpperCase
00216 I C   0  0
00217 I C   0  0   Purpose: Convert lowercase letters to uppercase
00218 I C   0  0
00219 I C   0  0   Author: William A. Chapman      Date: July 28, 1986
00220 I C   0  0
00221 I C   0  0   Inputs: A string is input (max ShortStringLength characters)
00222 I C   0  0
00223 I C   0  0   Outputs: All lowercase letters are converted to uppercase
00224 I C   0  0
00225 I C   0  0   Procedures Invoked: none
00226 I C   0  0 ]
00227 I     1  0 procedure UpperCase( var SourceString: ShortString);
00228 I     1  0
00229 I     1  0 const
00230 I     1  0     StringStart = 1;
00231 I     1  0
00232 I     1  0 var
00233 I     1  0     Loop: integer;
00234 I     1  0     Offset: integer;
00235 I     1  0
00236 I     1  1 begin
00237 I     1  1     Offset := ord( 'a') - ord( 'A');
00238 I     1  1     for Loop := StringStart to ShortStringLength do
00239 I     1  1   if ((( ord( SourceString[Loop]) - ord( 'a')) >= 0 ) and
00240 I     1  1       (( ord( SourceString[Loop]) - ord( 'z')) <= 0 ))
00241 I     1  1   then SourceString[Loop] := chr( ord( SourceString[ Loop]) - Offset);
00242 I     0  0 end;
```

```
-LINE-IDC-PL-SL-

00244       0  1 begin
00245       0  1     writeln('Keyword Look-Up-Table Maintenance Program');
00246       0  1     ValidClasses := [BlankClass, BreakpointClass, ErrorClass, FlagClass,
00247       0  1   InputOutputClass, TrueFalseClass, MemoryClass, NumericClass,
00248       0  1   OperatorClass, QualifierClass,  RegisterClass, StackClass,
00249       0  1   UtilityClass, ExecuteClass];
00250       0  1     write( 'Create (C), Invalidate (I), Edit (E), Print (P) ? ');
00251       0  1     readln( Command);
00252       0  2     case Command of
00253       0  3   'C','c': with Keyword do begin
00254       0  3       rewrite( KeywordLUTFile);
00255       0  3       Name := UndefinedName;
00256       0  3       Class := UndefinedClass;
00257       0  3       Width := UndefinedWidth;
00258       0  3       ID := UndefinedID;
00259       0  3       for Index := FirstKeywordEntry to LastKeywordEntry do
00260       0  3           write( KeywordLUTFile, Keyword);
00261       0  2   end; {case C]
00262       0  2
00263       0  3   'P','p': with Keyword do begin
00264       0  3       reset( KeywordLUTFile);
00265       0  3       write( 'Enter name of file to contain listing: ');
00266       0  3       readln( ListFile);
00267       0  3       open( KeywordLUTList, ListFile, new);
00268       0  3       rewrite( KeywordLUTList);
00269       0  4       for Index := FirstKeywordEntry to LastKeywordEntry do begin
00270       0  5           if ((Index rem PageSize) = 0) then begin
00271       0  5               if (Index >= PageSize) then writeln( KeywordLUTList, chr(FF));
00272       0  5               writeln( KeywordLUTList,
00273       0  5                   'Index  KeyWord   ID   Class  Width');
00274       0  5               writeln( KeywordLUTList);
00275       0  4           end;
00276       0  4           read( KeywordLUTFile, Keyword);
00277       0  4           write( KeywordLUTList, ' ', Index:3,     );
00278       0  4           write( KeywordLUTList, Name:8, '  ');
00279       0  4           write( KeywordLUTList, ID:3, '     ');
00280       0  4           write( KeywordLUTList, Class:1,        );
00281       0  4           write( KeywordLUTList, Width:3);
00282       0  4           writeln( KeywordLUTList);
00283       0  3       end; {for Index]
00284       0  2   end; {case P]
```

```
-LINE-IDC-PL-SL-

00286      0  3   'U','u','I','i : with Keyword do begin {convert to undefined entry}
00287      0  3       writeln( 'Convert an entry to undefined status');
00288      0  3       reset( KeywordLUTFile);
00289      0  3       for Index := FirstKeywordEntry to LastKeywordEntry do
00290      0  3           read( KeywordLUTFile, KeywordLUTable[ Index]);
00291      0  3       Continue := true;
00292      0  4       while Continue do begin
00293      0  4           writeln;  writeln;
00294      0  4           writeln( 'Enter index for entry to be invalidated');
00295      0  4           write(      (number greater than ', LastKeywordEntry:1);
00296      0  4           write(   ' to exit): ');
00297      0  4           readln( Index);
00298      0  4           if (Index <= LastKeywordEntry) then
00299      0  5           with KeywordLUTable[Index] do begin
00300      0  5               Name := UndefinedName;
00301      0  5               Class := UndefinedClass;
00302      0  5               Width := UndefinedWidth;
00303      0  5               ID := UndefinedID;
00304      0  5           end {with KeywordLUTable}
00305      0  4           else Continue := false;
00306      0  3       end; {while Continue}
00307      0  3       rewrite( KeywordLUTFile);
00308      0  3       for Index := FirstKeywordEntry to LastKeywordEntry do
00309      0  3           write( KeywordLUTFile, KeywordLUTAble[ Index]);
00310      0  2   end; {case U, I}
```

```
-LINE-IDC-PL-SL-

00312       0  3   'E','e': with Keyword do begin
00313       0  3       reset( KeywordLUTFile);
00314       0  3       for Index := FirstKeywordEntry to LastKeywordEntry do
00315       0  3           read( KeywordLUTFile, KeywordLUTable[ Index]);
00316       0  3       Continue := true;
00317       0  4       while Continue do begin
00318       0  4           writeln;  writeln;
00319       0  4           writeln( 'Enter index for entry to be edited');
00320       0  4           write(      (number greater than ', LastKeywordEntry:1);
00321       0  4           write(   ' to exit): ');
00322       0  4           readln( Index);
00323       0  4           if (Index <= LastKeywordEntry) then
00324       0  5           with KeywordLUTable[Index] do begin
00325       0  5               write( 'Keyword Name: ', Name:8);
00326       0  5               write(    ID: ', ID:3);
00327       0  5               write(    Class: ', Class:1);
00328       0  5               write( '  Width: ', Width:3);
00329       0  5               writeln;
00330       0  5               Correct := false;
00331       0  6               while (not Correct) do begin
00332       0  6                   Correct := true;
00333       0  6
00334       0  6                   write( 'Enter  new Name: ');
00335       0  6                   readln( NewName);
00336       0  6                   NewName := pad(NewName,    , 8);
00337       0  6                   Name := substr(NewName, 1, 8);
00338       0  6                   UpperCase( Name);
00339       0  6
00340       0  6                   write( 'Enter  new   ID: ');
00341       0  6                   readln( NewID);
00342       0  6                   if ((NewID >= MinID) and (NewID <= MaxID))
00343       0  6                       then ID := NewID
00344       0  7                       else begin
00345       0  7                           Correct := false;
00346       0  7                           writeln( 'Bad ID');
00347       0  6                       end;
00348       0  6
00349       0  6                   write( 'Enter new Class: ');
00350       0  6                   readln( NewClass);
00351       0  6                   if ((( ord( NewClass) - ord( 'a')) >= 0) and
00352       0  6                       (( ord( NewClass) - ord( 'z')) <= 0))
00353       0  6                       then NewClass :=
00354       0  6                           chr( ord( NewClass) - ord( 'a') + ord( 'A'));
00355       0  6                   if (NewClass in ValidClasses)
```

```
00356      0  6                    then Class := NewClass
00357      0  7                    else begin
00358      0  7                        Correct := false;
00359      0  7                        writeln( 'Bad Class');
00360      0  6                    end;
```

```
-LINE-IDC-PL-SL-

00362       0  6                  write( 'Enter new Width: ');
00363       0  6                  readln( NewWidth);
00364       0  6                  if ((NewWidth >= MinWidth) and (NewWidth <= MaxWidth))
00365       0  6                      then Width := NewWidth
00366       0  7                      else begin
00367       0  7                          Correct := false;
00368       0  7                          writeln( 'Bad Width');
00369       0  6                      end;
00370       0  5                  end; {while not Correct}
00371       0  5
00372       0  5              end {then with KeywordLUTAble}
00373       0  4              else Continue := false;
00374       0  3          end; {while Continue}
00375       0  3          rewrite( KeywordLUTFile);
00376       0  3          for Index := FirstKeywordEntry to LastKeywordEntry do
00377       0  3              write( KeywordLUTFile, KeywordLUTAble[ Index]);
00378       0  2      end; {case E}
00379       0  2
00380       0  2      otherwise writeln('Bad command');
00381       0  1        end; {case Command}
00382       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	2832	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
PAS/LIS KEYWORDLUTGEN.PAS

/CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
/DEBUG=(NOSYMBOLS,TRACEBACK)
/SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
/OPTIMIZE
/STANDARD=NONE
/TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
/USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
/NOANALYSIS_DATA
/NOENVIRONMENT
/LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]KEYWORDLUTGEN.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]KEYWORDLUTGEN.OBJ;11
/NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	404	00:00.5	00:02.4
Source Analysis	311	00:01.3	00:05.5
Source Listing	30	00:00.6	00:04.2
Tree Construction	190	00:00.3	00:01.1
Flow Analysis	55	00:00.2	00:00.5
Value Propagation	11	00:00.0	00:00.3
Profit Analysis	104	00:00.2	00:00.4
Context Analysis	223	00:01.5	00:03.9
Name Packing	8	00:00.1	00:00.4
Code Selection	111	00:00.4	00:00.8
Final	62	00:00.6	00:03.5
TOTAL	1513	00:05.7	00:22.9

COMPILATION STATISTICS

```
CPU Time:       00:05.7         (4035 Lines/Minute)
Elapsed Time:   00:22.9
```

```
-LINE-IDC-PL-SL-

00001   C  0  0 { Title: Loader
00002   C  0  0
00003   C  0  0   Purpose: Place the located Intel program image into "memory"
00004   C  0  0           using the "Standard" intel format, or a "Hex" format
00005   C  0  0
00006   C  0  0   Author: William A. Chapman      Date: April 7, 1986
00007   C  0  0
00008   C  0  0   Inputs: Intel's Absolute (Located) Object File
00009   C  0  0                          OR
00010   C  0  0           An ASCII file containing 2 hex characters per line
00011   C  0  0           which is a representation of the located image.
00012   C  0  0
00013   C  0  0   Outputs: The file is loaded into "memory"
00014   C  0  0
00015   C  0  0   Procedures Invoked: StoreMemory
00016   C  0  0                       MaskShiftRight
00017   C  0  0                       IntMaskShiftRight
00018   C  0  0                       HexIn
00019   C  0  0                       HexConvert (local)
00020   C  0  0 }
00021      0  0 module Loader(input, output, Image1, Image2);
00022      0  0
00023      0  0 const
00024      0  0     %include '[-]Const.defn/list'
00025 I    0  0     FirstOpcodeValue = 0;
00026 I    0  0     LastOpcodeValue = 255;
00027 I    0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00028 I    0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00029 I    0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00030 I    0  0     Bit8Multiplier = %X'100';
00031 I    0  0     Bit8Divisor    = %X'100';
00032 I    0  0     WordMultiplier = %X'100';
00033 I    0  0
00034 I    0  0     EightBits = 0;                        {width for 8 bits}
00035 I    0  0     SixteenBits = 1;                      {width for 16 bits}
00036 I    0  0
00037 I    0  0     SimpleMode = %B'0';
00038 I    0  0     SimpleRM = %B'110';
00039 I    0  0
00040 I    0  0     LowMemoryLimit = 0;                   {low limit on memory array}
00041 I    0  0     HighMemoryLimit = 2048;               {high limit on memory array}
00042 I    0  0
00043 I    0  0     FilenameLength = 40;
00044      0  0
00045      0  0     StartMainMask = %B'11000000';
```

```
00046       0  0     StartMainDivisor = %B'01000000';
00047       0  0
00048       0  0     %include  [-]StopCode.defn/list'
00049 I     0  0     Breakpoint              = 'B';
00050 I     0  0     Call          = 'C';
00051 I     0  0     ControlD              = 'D';
00052 I     0  0     BadOpcode               = 'E';
00053 I     0  0     BadCEAModeValue         = 'F';
00054 I     0  0     BadRMDModeValue         = 'G';
00055 I     0  0     Halt          = 'H';
```

```
-LINE-IDC-PL-SL-

00056 I    0  0     BadRegisterID = 'I';
00057 I    0  0     BadOpcodeKey  = 'K';
00058 I    0  0     BadMemoryAddress      = 'M';
00059 I    0  0     Normal                = 'N';
00060 I    0  0     BadPortAddress        = 'P';
00061 I    0  0     BadOpcodeClass        = 'Q';
00062 I    0  0     Return                = 'R';
00063 I    0  0     BadCheckSum           = 'S';
00064 I    0  0     BadOperandType        = 'T';
00065 I    0  0     BadMemoryWidth        = 'W';
00066 I    0  0     BadOpcodeExtension    = 'X';
00067 I    0  0     NoMemoryAccess      = 'Z';
00068      0  0
00069      0  0     LineLength = 64;
00070      0  0
00071      0  0     Standard = 'S';
00072      0  0
00073      0  0     None = false;
00074      0  0     Success = true;
```

```
-LINE-IDC-PL-SL-

00076       0  0 type
00077       0  0     %include '[-]Type.defn/list'
00078 I     0  0     ZeroOne = -1..1;                        {defines integer with range
00079 I C   0  0                                              of zero to one with an invalid
00080 I C   0  0                                              state of -1}
00081 I     0  0
00082 I     0  0     OpcodeLUTEntry = record                 {defines entries in opcode LUT}
00083 I     0  0   OpcodeClass: char;                        {class D, A, L, S, C, P}
00084 I     0  0   OpcodeKey: integer;                       {key to interpret opcode from
00085 I C   0  0                                              0 to 7}
00086 I     0  0   DirectionBitPresent: boolean;
00087 I     0  0   WidthBitPresent: boolean
00088 I     0  0     end;
00089 I     0  0
00090 I     0  0     OpcodeType = record
00091 I     0  0   Direction: ZeroOne;                      {to or from CPU}
00092 I     0  0   Full: unsigned;                          {full opcode}
00093 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00094 I     0  0     end;
00095 I     0  0
00096 I     0  0     RegisterType = record
00097 I     0  0   Id: integer;                             {identifier}
00098 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00099 I     0  0     end;
00100 I     0  0
00101 I     0  0     EffectiveAddressType = record
00102 I     0  0   Mode: char;                              {register or memory}
00103 I     0  0   Width: ZeroOne;                         {8 or 16 bits}
00104 I     0  0   Address: unsigned;                      {memory address or
00105 I C   0  0                                              register designation}
00106 I     0  0   Segment: integer                        {segment register to use}
00107 I     0  0     end;
00108 I     0  0
00109 I     0  0     NameType = packed array[1..10] of char;
00110 I     0  0
00111 I     0  0     Characters = set of    ..'~';
00112 I     0  0     LettersAndNumbers = set of '0'..'z';
00113 I     0  0     Letters = set of 'A'..'z';
00114 I     0  0     Numbers = set of '0'..'9';
00115 I     0  0
00116 I     0  0     FileName = packed array [1..FilenameLength] of char;
00117       0  0
00118       0  0     RecordBlock = array [1..128] of unsigned;
00119       0  0
```

```
00120      0  0     TwoByteString = packed array[1..2] of char;
```

```
-LINE-IDC-PL-SL-

00122      0  0 var
00123      0  0     Image1: text;
00124      0  0
00125      0  0     Address: unsigned;
00126      0  0
00127      0  0     Value: unsigned;
00128      0  0
00129      0  0     LoadData: TwoByteString;
00130      0  0
00131      0  0     Image2: file of RecordBlock;
00132      0  0
00133      0  0     FileBlock: RecordBlock;
00134      0  0     BlockPointer: integer;
00135      0  0
00136      0  0     FileByte: unsigned;
00137      0  0     ByteCounter: integer;
00138      0  0
00139      0  0     UnsignedRecordType: unsigned;
00140      0  0     RecordType: integer;
00141      0  0
00142      0  0     UnsignedRecordLength: unsigned;
00143      0  0     RecordLength: integer;
00144      0  0
00145      0  0     ByteData: unsigned;
00146      0  0     Dummy: unsigned;
00147      0  0     Checksum: integer;
00148      0  0
00149      0  0     Index: integer;
00150      0  0
00151      0  0     ModType: unsigned;
00152      0  0     ModuleAttribute: integer;
00153      0  0
00154      0  0     IP: [external] unsigned;
00155      0  0
00156      0  0     Done: boolean;
00157      0  0
00158      0  0     MemoryAccess: boolean;
```

```
-LINE-IDC-PL-SL-

00160       0  0     %include  [-]RegExternal.defn/list' {CS: [external] unsigned;}
00161 I     0  0     AX: [external] unsigned;      {value of the AX register}
00162 I     0  0     CX: [external] unsigned;      {value of the CX register}
00163 I     0  0     DX: [external] unsigned;      {value of the DX register}
00164 I     0  0     BX: [external] unsigned;      {value of the BX register}
00165 I     0  0     SP: [external] unsigned;      {value of the SP register}
00166 I     0  0     BP: [external] unsigned;      {value of the BP register}
00167 I     0  0     SI: [external] unsigned;      {value of the SI register}
00168 I     0  0     DI: [external] unsigned;      {value of the DI register}
00169 I     0  0
00170 I     0  0     ES: [external] unsigned;      {value of the ES register}
00171 I     0  0     CS: [external] unsigned;      {value of the CS register}
00172 I     0  0     SS: [external] unsigned;      {value of the SS register}
00173 I     0  0     DS: [external] unsigned;      {value of the DS register}
00174 I     0  0
00175       0  0
00176       0  0
00177       0  0
00178       1  0 [global] procedure Loader( NameOfFile: FileName;
00179       1  0           LFormat: char;
00180       1  0           var LStopCode: char;
00181       1  0           var LSegment: unsigned;
00182       1  0           var LAddress: unsigned);
00183       1  0
00184       2  0 procedure StoreMemory(SMAddress: unsigned;
00185       2  0           SMSegment: unsigned;
00186       2  0           SMValue: unsigned;
00187       1  0           var SMStopCode: char); external;
00188       1  0
00189       2  0 function MaskShiftRight(var MSRValue: unsigned;
00190       2  0           MSRMask: unsigned;
00191       1  0           MSRDivisor: unsigned): unsigned; external;
00192       1  0
00193       2  0 function IntMaskShiftRight(var IMSRValue: unsigned;
00194       2  0           IMSRMask: unsigned;
00195       1  0           IMSRDivisor: unsigned): integer; external;
00196       1  0
00197       1  0 procedure HexIn(var HIValue: unsigned); external;
```

```
-LINE-IDC-PL-SL-

00199   C  1  0 { Title: Convert Hexadecimal Input module
00200   C  1  0
00201   C  1  0   Purpose: Convert a hexadecimal string to a hexadecimal number
00202   C  1  0
00203   C  1  0   Author: William A. Chapman      Date: March 5,1986
00204   C  1  0
00205   C  1  0   Inputs: String of characters
00206   C  1  0
00207   C  1  0   Outputs: An unsigned integer representation of the
00208   C  1  0            character hexadecimal string.
00209   C  1  0
00210   C  1  0   Procedures Invoked: none
00211   C  1  0 }
00212      2  0 procedure HexConvert(HCString: TwoByteString;
00213      2  0           var HCValue: unsigned);
00214      2  0
00215      2  0 var
00216      2  0     HexInChar: char;                    {character to process}
00217      2  0     Index: integer;
00218      2  0     Temp: integer;                      {temporary variable}
00219      2  0
00220      2  0
00221      2  1 begin
00222      2  1     HCValue := 0;
00223      2  2     for Index := 1 to 2 do begin
00224      2  2   HexInChar := HCString[Index];
00225      2  2   Temp := ord(HexInChar) - ord('0');
00226      2  2   if (HexInChar = ' ') then Temp := 0;
00227      2  2   if (Temp > 10) then Temp := Temp + ord('0') - ord('A') + 10;
00228      2  2   if (Temp > 15) then Temp := Temp + ord('A') - ord('a');
00229      2  3   if ((Temp < 0) or (Temp > 15)) then begin
00230      2  3       writeln;
00231      2  3       write( 'HexConvert: Invalid character for conversion:  );
00232      2  3       writeln( '"', HexInChar, '"');
00233      2  3       HCValue := 0;
00234      2  3       Temp := 0;
00235      2  3   end {invalid character}
00236      2  2   else HCValue := (HCValue * 16) + Temp;
00237      2  1     end;
00238      1  0 end;
```

```
-LINE-IDC-PL-SL-

00240   C  1  0 { Title: GetByte
00241   C  1  0
00242   C  1  0   Purpose: Get a byte of data from input file
00243   C  1  0
00244   C  1  0   Author: William A. Chapman      Date: April 7, 1986
00245   C  1  0
00246   C  1  0   Inputs: Intel's Absolute (Located) Object File
00247   C  1  0           Modulo pointer for next byte
00248   C  1  0
00249   C  1  0   Outputs: Temporary storage of the 4 bytes obtained for each read
00250   C  1  0           Next byte of data from file
00251   C  1  0
00252   C  1  0 ]
00253      2  0 procedure GetByte(var Datum: unsigned);
00254      2  0
00255      2  0 const
00256      2  0     %include '[-]Byte.defn/list'
00257 I    2  0     Byte0Mask = %X'000000FF';
00258 I    2  0     Byte1Mask = %X'0000FF00';
00259 I    2  0     Byte2Mask = %X'00FF0000';
00260 I    2  0     Byte3Mask = %X'FF000000';
00261 I    2  0
00262 I    2  0     Byte0Divisor = %X'00000001';
00263 I    2  0     Byte1Divisor = %X'00000100';
00264 I    2  0     Byte2Divisor = %X'00010000';
00265 I    2  0     Byte3Divisor = %X'01000000';
00266 I    2  0
00267 I    2  0     Byte0Multiplier = %X'00000001 ;
00268 I    2  0     Byte1Multiplier = %X'00000100';
00269 I    2  0     Byte2Multiplier = %X'00010000';
00270 I    2  0     Byte3Multiplier = %X'01000000';
00271 I    2  0
00272 I    2  0     BytesPerWord = 4;
00273      2  0
00274      2  0 var
00275      2  0     BytePointer: integer;
```

```
-LINE-IDC-PL-SL-

00277       2  1 begin
00278       2  1     BytePointer := ByteCounter rem BytesPerWord;
00279       2  1
00280       2  2     case BytePointer of
00281       2  2
00282       2  2     0:    Datum := MaskShiftRight(FileByte, Byte0Mask, Byte0Divisor);
00283       2  2
00284       2  2     1:    Datum := MaskShiftRight(FileByte, Byte1Mask, Byte1Divisor);
00285       2  2
00286       2  2     2:    Datum := MaskShiftRight(FileByte, Byte2Mask, Byte2Divisor);
00287       2  2
00288       2  2     3:    Datum := MaskShiftRight(FileByte, Byte3Mask, Byte3Divisor);
00289       2  2
00290       2  3     otherwise begin
00291       2  3   write('Loader: GetByte: Invalid Byte Pointer:  );
00292       2  3   writeln(BytePointer:1);
00293       2  2       end; {otherwise}
00294       2  2
00295       2  1     end; {case BytePointer}
00296       2  1
00297       2  1
00298       2  1     ByteCounter := ByteCounter + 1;
00299       2  1
00300       2  2     if (BlockPointer < 128) then begin
00301       2  3   if (BytePointer >= 3) then begin
00302       2  3       BlockPointer := BlockPointer + 1;
00303       2  3       FileByte := FileBlock[BlockPointer];
00304       2  2   end; {if BytePointer >= 3}
00305       2  2     end
00306       2  2     else if (BytePointer >= 1) then begin
00307       2  2   BlockPointer := 1;
00308       2  2   read(Image2, FileBlock);
00309       2  2   ByteCounter := 0;
00310       2  2   FileByte := FileBlock[BlockPointer];
00311       2  1     end; {if BytePointer >= 1}
00312       2  1
00313       1  0 end; {procedure GetByte}
```

```
-LINE-IDC-PL-SL-

00315   C  1  0 { Title: GetBytewithChecksum
00316   C  1  0
00317   C  1  0   Purpose: Get a byte from file and accumulate the checksum
00318   C  1  0
00319   C  1  0   Author: William A. Chapman      Date: April 7, 1986
00320   C  1  0
00321   C  1  0   Inputs: Previous checksum value
00322   C  1  0
00323   C  1  0   Outputs: Data byte is returned
00324   C  1  0            Checksum is updated (modulo 256)
00325   C  1  0
00326   C  1  0   Procedures Invoked: GetByte
00327   C  1  0 }
00328      2  0 procedure GetBytewithChecksum(var ByteValue: unsigned;
00329      2  0           var ChecksumValue: integer);
00330      2  0
00331      2  0 const
00332      2  0     ChecksumBase = 256;
00333      2  0
00334      2  1 begin
00335      2  1     GetByte(ByteValue);
00336      2  1
00337      2  1     ChecksumValue := ChecksumValue + int( ByteValue);
00338      2  1     ChecksumValue := ChecksumValue mod ChecksumBase;
00339      1  0 end;
```

```
-LINE-IDC-PL-SL-

00341   C  1  0 { Title: GetWordwithChecksum
00342   C  1  0
00343   C  1  0   Purpose: Get 16 bits of data and accumulate the checksum
00344   C  1  0
00345   C  1  0   Author: William A. Chapman      Date: April 7,1986
00346   C  1  0
00347   C  1  0   Inputs: Checksum Value
00348   C  1  0           File Data
00349   C  1  0
00350   C  1  0   Outputs: Checksum Value is updated
00351   C  1  0           Sixteen bits of data is returned
00352   C  1  0
00353   C  1  0   Procedures Invoked: GetBytewithChecksum
00354   C  1  0 }
00355      2  0 procedure GetWordwithChecksum(var WordValue: unsigned;
00356      2  0           var ChecksumValue: integer);
00357      2  0
00358      2  0 const
00359      2  0     Byte1Multiplier = %X'100';
00360      2  0     SixteenBitMask = %X'FFFF';
00361      2  0
00362      2  0 var
00363      2  0     Byte0: unsigned;
00364      2  0     Byte1: unsigned;
00365      2  0
00366      2  1 begin
00367      2  1     GetBytewithChecksum(Byte0, ChecksumValue);
00368      2  1     GetBytewithChecksum(Byte1, ChecksumValue);
00369      2  1     WordValue := uand( uor( Byte0, (Byte1 * Byte1Multiplier)), SixteenBitMask);
00370      1  0 end;
```

```
-LINE-IDC-PL-SL-

00372      1  1 begin {Loader]
00373      1  2   if (LFormat = Standard) then begin
00374      1  2     MemoryAccess := None;
00375      1  2     open(Image2, NameOfFile, old);
00376      1  2     reset (Image2);
00377      1  2     read(Image2, FileBlock);
00378      1  2
00379      1  2     BlockPointer := 1;
00380      1  2     FileByte := FileBlock[BlockPointer];
00381      1  2     ByteCounter := 0;
00382      1  2     Done := false;
00383      1  3     while (not Done) do begin
00384      1  3   Checksum := 0;
00385      1  3   GetBytewithChecksum(UnsignedRecordType, Checksum);
00386      1  3   RecordType := int(UnsignedRecordType);
00387      1  3   GetWordwithChecksum(UnsignedRecordLength, Checksum);
00388      1  3   RecordLength := int(UnsignedRecordLength);
00389      1  4   case RecordType of
00390      1  5   %X'84': begin {physical enumerated data]
00391      1  5       GetWordwithChecksum( LSegment, Checksum);
00392      1  5       GetBytewithChecksum( LAddress, Checksum);
00393      1  5       Index := 1;
00394      1  6       while ((Index <= (RecordLength - 4)) and (LStopCode = Normal)) do begin
00395   C  1  6           {the -4 accounts for the Segment (2), Address (1)
00396   C  1  6           and the Checksum (1) read at "if Checksum <>"]
00397      1  6           GetBytewithChecksum(ByteData, Checksum);
00398      1  6           StoreMemory( LAddress, LSegment, ByteData, LStopCode);
00399      1  6           MemoryAccess := Success;
00400      1  6           if (LStopCode <> Normal) then Done := true;
00401      1  6           Index := Index + 1;
00402      1  6           LAddress := LAddress + 1;
00403      1  5       end; {for Index]
00404      1  4   end; {case %X'84']
00405      1  4
00406      1  5   %X'8A': begin {module end]
00407      1  5       GetBytewithChecksum(ModType, Checksum);
00408      1  5       ModuleAttribute := IntMaskShiftRight( ModType, StartMainMask,
00409      1  5           StartMainDivisor);
00410      1  5
00411      1  6       case ModuleAttribute of
00412      1  6       0,2: ;
00413      1  7       1: begin
00414      1  7           GetWordwithChecksum(Dummy, Checksum);
00415      1  7           GetWordwithChecksum(Dummy, Checksum);
```

```
00416      1  6         end;
00417      1  7       3: begin
00418      1  7            GetWordwithChecksum(CS, Checksum);
00419      1  7            GetWordwithChecksum(IP, Checksum);
00420      1  6         end;
00421      1  5       end; {case ModuleAttribute}
00422      1  5       Done := true;
00423      1  4   end; {case %X'8A'}
```

```
-LINE-IDC-PL-SL-

00425       1  4   otherwise
00426       1  4       for Index := 1 to (RecordLength - 1) do
00427       1  4           GetBytewithChecksum(Dummy, Checksum);
00428       1  4
00429       1  3   end; {case RecordType}
00430       1  3
00431       1  3   GetBytewithChecksum(Dummy, Checksum);
00432       1  4   if (Checksum <> 0) then begin
00433       1  4       LStopCode := BadCheckSum;
00434       1  4       Done := true;
00435       1  3   end; {if Checksum}
00436       1  3
00437       1  2     end;  {while not eof}
00438       1  2
00439       1  2     close( Image2);
00440       1  2     if (MemoryAccess = None) then
00441       1  2   if (LStopCode = Normal) then LStopCode := NoMemoryAccess;
00442       1  2   end {Standard Format}
```

```
-LINE-IDC-PL-SL-

00444       1  2   else begin
00445       1  2     writeln;  writeln;
00446       1  2     write( 'Enter segment value (in Hex):  );
00447       1  2     HexIn( LSegment);
00448       1  2     write( 'Enter starting address of image (in Hex): ');
00449       1  2     HexIn( Address);
00450       1  2     writeln;  writeln;
00451       1  2
00452       1  2     LAddress := Address;
00453       1  2
00454       1  2     open(Image1, NameOfFile, old);
00455       1  2     reset(Image1);
00456       1  2
00457       1  2     writeln('Reading load image from disk');
00458       1  3     while ( (not eof(Image1)) and (LStopCode = Normal)) do begin
00459       1  3   readln(Image1, LoadData);
00460       1  3   HexConvert(LoadData, Value);
00461       1  3   StoreMemory(LAddress, LSegment, Value, LStopCode);
00462       1  3   LAddress := LAddress + 1;
00463       1  2     end;
00464       1  2     close( Image1);
00465       1  1   end;
00466       0  0 end; {Loader]
00467       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	2116	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	1126	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS LOADER.PAS

  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /SHOW=(DICTIONARY,INCLUDE,NOINLINE,HEADER,SOURCE,STATISTICS,TABLE_OF_CONTENTS)
  /OPTIMIZE
  /STANDARD=NONE
  /TERMINAL=(NOFILE_NAME,NOROUTINE_NAME,NOSTATISTICS)
  /USAGE=(NOUNUSED,UNINITIALIZED,NOUNCERTAIN)
  /NOANALYSIS_DATA
  /NOENVIRONMENT
  /LIST=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]LOADER.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]LOADER.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	404	00:00.5	00:05.1
Source Analysis	377	00:01.4	00:10.3
Source Listing	14	00:00.7	00:04.2
Tree Construction	143	00:00.5	00:01.4
Flow Analysis	74	00:00.3	00:00.6
Value Propagation	12	00:00.1	00:00.1
Profit Analysis	52	00:00.2	00:00.9
Context Analysis	332	00:03.1	00:10.0
Name Packing	7	00:00.1	00:00.1
Code Selection	122	00:00.4	00:00.8
Final	66	00:00.6	00:03.9
TOTAL	1607	00:07.9	00:37.6

COMPILATION STATISTICS

CPU Time: 00:07.9 (3556 Lines/Minute)