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. 3

Command Procedures

and

Source Code

ROCHESTER INSTITUTE OF TECHNOLOGY

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

00001   C  0  0 { Title: Logic
00002   C  0  0
00003   C  0  0   Purpose: Execute the logical opcodes
00004   C  0  0
00005   C  0  0   Author: William A. Chapman      Date: April 9, 1986
00006   C  0  0
00007   C  0  0   Inputs: Opcode record
00008   C  0  0           Register Record
00009   C  0  0           Effective Address Record
00010   C  0  0
00011   C  0  0   Outputs: Memory, registers and condition codes are manipulated
00012   C  0  0           as requested for each instruction
00013   C  0  0
00014   C  0  0   Procedures Invoked: FetchRegPtr
00015   C  0  0           FetchMemory
00016   C  0  0           StoreRegPtr
00017   C  0  0           FetchOperand
00018   C  0  0           StoreData
00019   C  0  0           FetchCodeData
00020   C  0  0           UpdateZeroFlag
00021   C  0  0           UpdateSignFlag
00022   C  0  0           UpdateParityFlag
00023   C  0  0           MaskShiftRight
00024   C  0  0 }
00025      0  0 module Logic(input, output );
00026      0  0
00027      0  0
00028      0  0
00029      0  0 const
00030      0  0     %include  [-]Const.defn/list'
00031 I    0  0     FirstOpcodeValue = 0;
00032 I    0  0     LastOpcodeValue = 255;
00033 I    0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00034 I    0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00035 I    0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00036 I    0  0     Bit8Multiplier = %X'100';
00037 I    0  0     Bit8Divisor    = %X'100';
00038 I    0  0     WordMultiplier = %X'100';
00039 I    0  0
00040 I    0  0     EightBits = 0;                        {width for 8 bits}
00041 I    0  0     SixteenBits = 1;                      {width for 16 bits}
00042 I    0  0
00043 I    0  0     SimpleMode = %B'0';
00044 I    0  0     SimpleRM = %B'110';
00045 I    0  0
```

```
00046 I     0  0     LowMemoryLimit = 0;                    {low limit on memory array}
00047 I     0  0     HighMemoryLimit = 2048;                {high limit on memory array}
00048 I     0  0
00049 I     0  0     FilenameLength = 40;
00050       0  0
00051       0  0     %include '[-]Flags.defn/list'
00052 I     0  0     SetHigh = true;
00053 I     0  0     Clear = false;
00054 I     0  0
00055 I     0  0     CarryFlag     = %B'000000000001';
```

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

00056 I    0  0     ParityFlag    = %B'000000000100';
00057 I    0  0     AuxCarryFlag  = %B'000000010000';
00058 I    0  0     ZeroFlag      = %B'000001000000';
00059 I    0  0     SignFlag      = %B'000010000000';
00060 I    0  0     TrapFlag      = %B'000100000000';
00061 I    0  0     InterruptFlag = %B'001000000000';
00062 I    0  0     DirectionFlag = %B'010000000000';
00063 I    0  0     OverflowFlag  = %B'100000000000';
00064      0  0
00065      0  0     %include  [-]RegID.defn/list'
00066 I    0  0     ALId = %B'000';
00067 I    0  0     ALWidth = 0;
00068 I    0  0
00069 I    0  0     CLId = %B'001';
00070 I    0  0     CLWidth = 0;
00071 I    0  0
00072 I    0  0     DLId = %B'010';
00073 I    0  0     DLWidth = 0;
00074 I    0  0
00075 I    0  0     BLId = %B'011 ;
00076 I    0  0     BLWidth = 0;
00077 I    0  0
00078 I    0  0     AHId = %B'100';
00079 I    0  0     AHWidth = 0;
00080 I    0  0
00081 I    0  0     CHId = %B'101';
00082 I    0  0     CHWidth = 0;
00083 I    0  0
00084 I    0  0     DHId = %B'110';
00085 I    0  0     DHWidth = 0;
00086 I    0  0
00087 I    0  0     BHId = %B'111';
00088 I    0  0     BHWidth = 0;
00089 I    0  0
00090 I    0  0     AXId = %B'000';
00091 I    0  0     AXWidth = 1;
00092 I    0  0     ALorAXId = %B'000';
00093 I    0  0
00094 I    0  0     CXId = %B'001';
00095 I    0  0     CXWidth = 1;
00096 I    0  0
00097 I    0  0     DXId = %B'010';
00098 I    0  0     DXWidth = 1;
00099 I    0  0
```

```
00100 I     0  0     BXId = %B'011 ;
00101 I     0  0     BXWidth = 1;
00102 I     0  0
00103 I     0  0     SPId = %B'100';
00104 I     0  0     SPWidth = 1;
00105 I     0  0
00106 I     0  0     BPId = %B'101';
00107 I     0  0     BPWidth = 1;
00108 I     0  0
00109 I     0  0     SIId = %B'110';
00110 I     0  0     SIWidth = 1;
```

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

00111 I     0  0
00112 I     0  0      DIId = %B'111 ;
00113 I     0  0      DIWidth = 1;
00114 I     0  0
00115 I     0  0      SRWidth = 2;
00116 I     0  0
00117 I     0  0      ESId = %B'00';
00118 I     0  0      ESWidth = 2;
00119 I     0  0
00120 I     0  0      CSId = %B'01 ;
00121 I     0  0      CSWidth = 2;
00122 I     0  0
00123 I     0  0      SSId = %B'10';
00124 I     0  0      SSWidth = 2;
00125 I     0  0
00126 I     0  0      DSId = %B'11 ;
00127 I     0  0      DSWidth = 2;
00128 I     0  0
00129       0  0
00130       0  0      %include  [-]StopCode.defn/list'
00131 I     0  0      Breakpoint              = 'B';
00132 I     0  0      Call            = 'C';
00133 I     0  0      ControlD              = 'D';
00134 I     0  0      BadOpcode               = 'E';
00135 I     0  0      BadCEAModeValue         = 'F';
00136 I     0  0      BadRMDModeValue         = 'G';
00137 I     0  0      Halt            = 'H';
00138 I     0  0      BadRegisterID = 'I';
00139 I     0  0      BadOpcodeKey  = 'K';
00140 I     0  0      BadMemoryAddress        = 'M';
00141 I     0  0      Normal                  = 'N';
00142 I     0  0      BadPortAddress          = 'P';
00143 I     0  0      BadOpcodeClass          = 'Q';
00144 I     0  0      Return                  = 'R';
00145 I     0  0      BadCheckSum             = 'S';
00146 I     0  0      BadOperandType          = 'T';
00147 I     0  0      BadMemoryWidth          = 'W';
00148 I     0  0      BadOpcodeExtension      = 'X';
00149 I     0  0      NoMemoryAccess        = 'Z';
```

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

00151       0  0     Register_is_Source = 0;
00152       0  0     Register_is_Destination = 1;
00153       0  0
00154       0  0     CountFlag = %B'10';
00155       0  0     DefaultCount = 1;
00156       0  0
00157       0  0     HighOrderBitforByte = %X'80';
00158       0  0     HighOrderBitforWord = %X'8000';
00159       0  0
00160       0  0     NexttoHighOrderBitforByte = %X'40';
00161       0  0     NexttoHighOrderBitforWord = %X'4000';
00162       0  0
00163       0  0     LowOrderBit = %B'1';
00164       0  0     LowOrderBitMask = %B'1' ;
00165       0  0
00166       0  0     AllBitsClear = 0;
```

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

00168       0  0 type
00169       0  0     %include '[-]Type.defn/list'
00170 I     0  0     ZeroOne = -1..1;                          {defines integer with range
00171 I C   0  0                                               of zero to one with an invalid
00172 I C   0  0                                               state of -1}
00173 I     0  0
00174 I     0  0     OpcodeLUTEntry = record                   {defines entries in opcode LUT}
00175 I     0  0   OpcodeClass: char;                         {class D, A, L, S, C, P}
00176 I     0  0   OpcodeKey: integer;                        {key to interpret opcode from
00177 I C   0  0                                               0 to 7}
00178 I     0  0   DirectionBitPresent: boolean;
00179 I     0  0   WidthBitPresent: boolean
00180 I     0  0     end;
00181 I     0  0
00182 I     0  0     OpcodeType = record
00183 I     0  0   Direction: ZeroOne;                       {to or from CPU}
00184 I     0  0   Full: unsigned;                           {full opcode}
00185 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00186 I     0  0     end;
00187 I     0  0
00188 I     0  0     RegisterType = record
00189 I     0  0   Id: integer;                             {identifier}
00190 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00191 I     0  0     end;
00192 I     0  0
00193 I     0  0     EffectiveAddressType = record
00194 I     0  0   Mode: char;                              {register or memory}
00195 I     0  0   Width: ZeroOne;                         {8 or 16 bits}
00196 I     0  0   Address: unsigned;                      {memory address or
00197 I C   0  0                                            register designation}
00198 I     0  0   Segment: integer                        {segment register to use}
00199 I     0  0     end;
00200 I     0  0
00201 I     0  0     NameType = packed array[1..10] of char;
00202 I     0  0
00203 I     0  0     Characters = set of    ..'~';
00204 I     0  0     LettersAndNumbers = set of '0'..'z';
00205 I     0  0     Letters = set of 'A'..'z';
00206 I     0  0     Numbers = set of '0'..'9';
00207 I     0  0
00208 I     0  0     FileName = packed array [1..FilenameLength] of char;
00209       0  0
00210       0  0     %include  [-]FlagType.defn/list'
00211 I     0  0     FlagType = record                         {define processor flags}
```

```
00212 I     0  0   Carry: boolean;
00213 I     0  0   Parity: boolean;
00214 I     0  0   AuxCarry: boolean;
00215 I     0  0   Zero: boolean;
00216 I     0  0   Sign: boolean;
00217 I     0  0   Trap: boolean;
00218 I     0  0   Interrupt: boolean;
00219 I     0  0   Direction: boolean;
00220 I     0  0   Overflow: boolean
00221 I     0  0     end;
```

```
-LINE- IDC PL SL-

00223       0  0 var
00224       0  0     Flags: [external] FlagType;
00225       0  0
00226       0  0     OpcodeKey: integer;
00227       0  0
00228       0  0     Operand1: unsigned;
00229       0  0     Operand2: unsigned;
00230       0  0     Result: unsigned;
00231       0  0     OperandMask: unsigned;
00232       0  0
00233       0  0     Count: unsigned;
00234       0  0     Counter: integer;
00235       0  0     RotationCount: integer;
00236       0  0
00237       0  0     TempCarryValue: unsigned;
00238       0  0
00239       0  0     CarryFlagValue: unsigned;
00240       0  0     OverflowFlagTest1: unsigned;
00241       0  0     OverflowFlagTest2: unsigned;
00242       0  0     TestState: boolean;
00243       0  0
00244       0  0     SignBit: unsigned;
00245       0  0
00246       0  0     HighOrderBit: unsigned;
00247       0  0     HighOrderBitMask: unsigned;
00248       0  0     NexttoHighOrderBitMask: unsigned;
```

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

00250      1  0 [global] procedure Logic(Opcode: OpcodeType;
00251      1  0           Register: RegisterType;
00252      1  0           EA: EffectiveAddressType;
00253      1  0           var LStopCode: char);
00254      1  0
00255      2  0 procedure FetchRegPtr(FRPWidth: integer;
00256      2  0           FRPDesignator: integer;
00257      2  0           var FRPValue: unsigned;
00258      1  0           var FRPStopCode: char); external;
00259      1  0
00260      2  0 procedure FetchMemory(FMAddress: unsigned;
00261      2  0           FMSegment: unsigned;
00262      2  0           var FMValue: unsigned;
00263      1  0           var FMStopCode: char); external;
00264      1  0
00265      2  0 procedure StoreRegPtr(SRPWidth: integer;
00266      2  0           SRPDesignator: integer;
00267      2  0           SRPValue: unsigned;
00268      1  0           var SRPStopCode: char); external;
00269      1  0
00270      2  0 procedure FetchOperand(FOType: char;
00271      2  0           FOWidth: integer;
00272      2  0           FOAddress: unsigned;
00273      2  0           FOSegment: integer;
00274      2  0           var FOValue: unsigned;
00275      1  0           var FOStopCode: char); external;
00276      1  0
00277      2  0 procedure StoreData(SDType: char;
00278      2  0           SDWidth: integer;
00279      2  0           SDAddress: unsigned;
00280      2  0           SDSegment: integer;
00281      2  0           SDValue: unsigned;
00282      1  0           var SDStopCode: char); external;
00283      1  0
00284      2  0 procedure FetchCodeData(FCDWidth: integer;
00285      2  0           var FCDValue: unsigned;
00286      1  0           var FCDStopCode: char); external;
00287      1  0
00288      2  0 function UpdateZeroFlag(UFWidth: integer;
00289      1  0           UFResult: unsigned): boolean; external;
00290      1  0
00291      2  0 function UpdateSignFlag(UFWidth: integer;
00292      1  0           UFResult: unsigned): boolean; external;
00293      1  0
```

```
00294      2  0 function UpdateParityFlag(UFWidth: integer;
00295      1  0           UFResult: unsigned): boolean; external;
00296      1  0
00297      2  0 function MaskShiftRight(MSRValue: unsigned;
00298      2  0           MSRMask: unsigned;
00299      1  0           MSRDivisor: integer): unsigned; external;
```

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

00301       1  1 begin
00302       1  1     OpcodeKey := int(Opcode.Full);
00303       1  1
00304       1  2     case OpcodeKey of
00305       1  2
00306       1  2   %X'8': {or register / memory with register to either}
00307    C  1  2   {A = undefined, C = 0, O = 0, P, S, Z}
00308       1  3     with Opcode do begin
00309       1  3       FetchRegPtr(Register.Width, Register.Id, Operand1, LStopCode);
00310       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Operand2, LStopCode);
00311       1  3       Result := uor(Operand1, Operand2);
00312       1  3       if (Direction = Register_is_Destination) then
00313       1  3           StoreRegPtr(Register.Width, Register.Id, Result, LStopCode)
00314       1  3       else StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00315       1  4       with Flags do begin
00316       1  4           Carry := Clear;
00317       1  4           Overflow := Clear;
00318       1  4           Parity := UpdateParityFlag(Width, Result);
00319       1  4           Sign := UpdateSignFlag(Width, Result);
00320       1  4           Zero := UpdateZeroFlag(Width, Result);
00321       1  3       end; {with Flags}
00322       1  2     end; {with opcode %X'8'}
00323       1  2
00324       1  2   %X'C': {or immediate to accumulator}
00325    C  1  2   {A = undefined, C = 0, O = 0, P, S, Z}
00326       1  3     with Opcode do begin
00327       1  3       FetchRegPtr(Width, ALorAXID, Operand1, LStopCode);
00328       1  3       FetchCodeData(Width, Operand2, LStopCode);
00329       1  3       Result := uor(Operand1, Operand2);
00330       1  3       StoreRegPtr(Width, ALorAXID, Result, LStopCode);
00331       1  4       with Flags do begin
00332       1  4           Carry := Clear;
00333       1  4           Overflow := Clear;
00334       1  4           Parity := UpdateParityFlag(Width, Result);
00335       1  4           Sign := UpdateSignFlag(Width, Result);
00336       1  4           Zero := UpdateZeroFlag(Width, Result);
00337       1  3       end; {with Flags}
00338       1  2     end; {with opcode %X'4'}
```

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

00340       1  2   %X'20': {and register / memory with register to either}
00341    C  1  2   {A = undefined, C = 0, O = 0, P, S, Z}
00342       1  3     with Opcode do begin
00343       1  3       FetchRegPtr(Register.Width, Register.Id, Operand1, LStopCode);
00344       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Operand2, LStopCode);
00345       1  3       Result := uand(Operand1, Operand2);
00346       1  3       if (Direction = Register_is_Destination) then
00347       1  3           StoreRegPtr(Register.Width, Register.Id, Result, LStopCode)
00348       1  3       else StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00349       1  4       with Flags do begin
00350       1  4           Carry := Clear;
00351       1  4           Overflow := Clear;
00352       1  4           Parity := UpdateParityFlag(Width, Result);
00353       1  4           Sign := UpdateSignFlag(Width, Result);
00354       1  4           Zero := UpdateZeroFlag(Width, Result);
00355       1  3       end; {with Flags}
00356       1  2     end; {with opcode %X'20'}
00357       1  2
00358       1  2   %X'24': {and immediate to accumulator}
00359    C  1  2   {A = undefined, C = 0, O = 0, P, S, Z}
00360       1  3     with Opcode do begin
00361       1  3       FetchRegPtr(Width, ALorAXID, Operand1, LStopCode);
00362       1  3       FetchCodeData(Width, Operand2, LStopCode);
00363       1  3       Result := uand(Operand1, Operand2);
00364       1  3       StoreRegPtr(Width, ALorAXID, Result, LStopCode);
00365       1  4       with Flags do begin
00366       1  4           Carry := Clear;
00367       1  4           Overflow := Clear;
00368       1  4           Parity := UpdateParityFlag(Width, Result);
00369       1  4           Sign := UpdateSignFlag(Width, Result);
00370       1  4           Zero := UpdateZeroFlag(Width, Result);
00371       1  3       end; {with Flags}
00372       1  2     end; {with opcode %X'24'}
00373       1  2
00374       1  2   %X'30': {xor register / memory with register to either}
00375    C  1  2   {A = undefined, C = 0, O = 0, P, S, Z}
00376       1  3     with Opcode do begin
00377       1  3       FetchRegPtr(Register.Width, Register.Id, Operand1, LStopCode);
00378       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Operand2, LStopCode);
00379       1  3       Result := uxor(Operand1, Operand2);
00380       1  3       if (Direction = Register_is_Destination) then
00381       1  3           StoreRegPtr(Register.Width, Register.Id, Result, LStopCode)
00382       1  3       else StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00383       1  4       with Flags do begin
```

```
00384      1  4           Carry := Clear;
00385      1  4           Overflow := Clear;
00386      1  4           Parity := UpdateParityFlag(Width, Result);
00387      1  4           Sign := UpdateSignFlag(Width, Result);
00388      1  4           Zero := UpdateZeroFlag(Width, Result);
00389      1  3       end; {with Flags}
00390      1  2     end; {with opcode %X'30'}
```

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

00392       1  2   %X'34': {xor immediate to accumulator}
00393    C  1  2   {A = undefined, C = 0, O = 0, P, S, Z}
00394       1  3     with Opcode do begin
00395       1  3       FetchRegPtr(Width, ALorAXID, Operand1, LStopCode);
00396       1  3       FetchCodeData(Width, Operand2, LStopCode);
00397       1  3       Result := uxor(Operand1, Operand2);
00398       1  3       StoreRegPtr(Width, ALorAXID, Result, LStopCode);
00399       1  4       with Flags do begin
00400       1  4           Carry := Clear;
00401       1  4           Overflow := Clear;
00402       1  4           Parity := UpdateParityFlag(Width, Result);
00403       1  4           Sign := UpdateSignFlag(Width, Result);
00404       1  4           Zero := UpdateZeroFlag(Width, Result);
00405       1  3       end; {with Flags}
00406       1  2     end; {with opcode %X'34'}
00407       1  2
00408       1  2   %X'84': {test register / memory with register to either}
00409    C  1  2   {A = undefined, C = 0, O = 0, P, S, Z}
00410       1  3     with Opcode do begin
00411       1  3       FetchRegPtr(Register.Width, Register.Id, Operand1, LStopCode);
00412       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Operand2, LStopCode);
00413       1  3       Result := uand(Operand1, Operand2);
00414       1  4       with Flags do begin
00415       1  4           Carry := Clear;
00416       1  4           Overflow := Clear;
00417       1  4           Parity := UpdateParityFlag(Width, Result);
00418       1  4           Sign := UpdateSignFlag(Width, Result);
00419       1  4           Zero := UpdateZeroFlag(Width, Result);
00420       1  3       end; {with Flags}
00421       1  2     end; {with opcode %X'84'}
00422       1  2
00423       1  2   %X'A8': {test immediate to accumulator}
00424    C  1  2   {A = undefined, C = 0, O = 0, P, S, Z}
00425       1  3     with Opcode do begin
00426       1  3       FetchRegPtr(Width, ALorAXID, Operand1, LStopCode);
00427       1  3       FetchCodeData(Width, Operand2, LStopCode);
00428       1  3       Result := uand(Operand1, Operand2);
00429       1  4       with Flags do begin
00430       1  4           Carry := Clear;
00431       1  4           Overflow := Clear;
00432       1  4           Parity := UpdateParityFlag(Width, Result);
00433       1  4           Sign := UpdateSignFlag(Width, Result);
00434       1  4           Zero := UpdateZeroFlag(Width, Result);
00435       1  3       end; {with Flags}
```

```
00436      1  2     end; {with opcode %X'A8'}
```

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

00438       1  2   %X'0F6': {test immediate to register / memory}
00439    C  1  2     {A = undefined, C = 0, O = 0, P, S, Z}
00440       1  3     with Opcode do begin
00441       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Operand1, LStopCode);
00442       1  3       FetchCodeData(Width, Operand2, LStopCode);
00443       1  3       Result := uand(Operand1, Operand2);
00444       1  4       with Flags do begin
00445       1  4           Carry := Clear;
00446       1  4           Overflow := Clear;
00447       1  4           Parity := UpdateParityFlag(Width, Result);
00448       1  4           Sign := UpdateSignFlag(Width, Result);
00449       1  4           Zero := UpdateZeroFlag(Width, Result);
00450       1  3       end; {with Flags}
00451       1  2     end; {with opcode %X'0F6'}
00452       1  2
00453       1  2   %X'0D0',%X'0D2': {rotate left}  {C, O}
00454       1  3     with Opcode do begin
00455       1  3
00456       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00457       1  3
00458       1  4       if (uand(Full, CountFlag) = CountFlag) then begin
00459       1  4           FetchRegPtr(CLWidth, CLID, Count, LStopCode);
00460       1  4           RotationCount := int(Count);
00461       1  4       end
00462       1  3       else RotationCount := DefaultCount;
00463       1  3
00464       1  4       if (Width = EightBits) then begin
00465       1  4           HighOrderBitMask := HighOrderBitforByte;
00466       1  4           OperandMask := Low8Bits;
00467       1  4       end
00468       1  4       else begin
00469       1  4           HighOrderBitMask := HighOrderBitforWord;
00470       1  4           OperandMask := All16Bits;
00471       1  3       end;
00472       1  3
00473       1  4       for Counter := 1 to RotationCount do begin
00474       1  4           CarryFlagValue := uand(Result, HighOrderBitMask);
00475       1  4           if (CarryFlagValue = AllBitsClear) then Flags.Carry := Clear
00476       1  4           else Flags.Carry := SetHigh;
00477       1  4           Result := uand((Result * 2), OperandMask);
00478       1  4           if (Flags.Carry = SetHigh) then Result := uor(Result, LowOrderBit);
00479       1  3       end; {for Counter}
00480       1  3
00481       1  4       if (RotationCount = DefaultCount) then begin
```

```
00482     1  4           OverflowFlagTest1 := uand(Result, HighOrderBitMask);
00483     1  4           if (OverflowFlagTest1 = AllBitsClear) then TestState := Clear
00484     1  4           else TestState := SetHigh;
00485     1  4           if (TestState <> Flags.Carry) then Flags.Overflow := SetHigh
00486     1  4           else Flags.Overflow := Clear;
00487     1  3       end; {if RotationCount = DefaultCount}
00488     1  3
00489     1  3       StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00490     1  3
00491     1  2     end; {with Opcode %X'0D0',%X'0D2'}
```

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

00493       1  2   %X'1D0',%X'1D2': {rotate right} {C, O}
00494       1  3     with Opcode do begin
00495       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00496       1  3
00497       1  4       if (uand(Full, CountFlag) = CountFlag) then begin
00498       1  4           FetchRegPtr(CLWidth, CLID, Count, LStopCode);
00499       1  4           RotationCount := int(Count);
00500       1  4       end
00501       1  3       else RotationCount := DefaultCount;
00502       1  3
00503       1  4       if (Width = EightBits) then begin
00504       1  4           OperandMask := Low8Bits;
00505       1  4           HighOrderBit := HighOrderBitforByte;
00506       1  4           HighOrderBitMask := HighOrderBitforByte;
00507       1  4           NexttoHighOrderBitMask := NexttoHighOrderBitforByte;
00508       1  4       end
00509       1  4       else begin
00510       1  4           OperandMask := All16Bits;
00511       1  4           HighOrderBit := HighOrderBitforWord;
00512       1  4           HighOrderBitMask := HighOrderBitforWord;
00513       1  4           NexttoHighOrderBitMask := NexttoHighOrderBitforWord;
00514       1  3       end;
00515       1  3
00516       1  4       for Counter := 1 to RotationCount do begin
00517       1  4           CarryFlagValue := uand(Result, LowOrderBitMask);
00518       1  4           if (CarryFlagValue = AllBitsClear) then Flags.Carry := Clear
00519       1  4           else Flags.Carry := SetHigh;
00520       1  4           Result := Result div 2;
00521       1  4           if (Flags.Carry = SetHigh) then
00522       1  4               Result := uor(Result, HighOrderBit);
00523       1  3       end; {for Counter}
00524       1  3
00525       1  4       if (RotationCount = DefaultCount) then begin
00526       1  4           OverflowFlagTest1 := uand(Result, HighOrderBitMask);
00527       1  4           OverflowFlagTest1 := (OverflowFlagTest1 div 2);
00528       1  4           OverflowFlagTest2 := uand(Result, NexttoHighOrderBitMask);
00529       1  4           if ( OverflowFlagTest1 <> OverflowFlagTest2 ) then
00530       1  4               Flags.Overflow := SetHigh
00531       1  4           else Flags.Overflow := Clear;
00532       1  3       end; {if RotationCount = DefaultCount}
00533       1  3
00534       1  3       StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00535       1  3
00536       1  2     end; {with Opcode %X'1D0',%X'1D2'}
```

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

00538       1  2   %X'2D0',%X'2D2': {rotate left through carry}  {C, O}
00539       1  3     with Opcode do begin
00540       1  3
00541       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00542       1  3
00543       1  4       if (uand(Full, CountFlag) = CountFlag) then begin
00544       1  4           FetchRegPtr(CLWidth, CLID, Count, LStopCode);
00545       1  4           RotationCount := int(Count);
00546       1  4       end
00547       1  3       else RotationCount := DefaultCount;
00548       1  3
00549       1  4       if (Width = EightBits) then begin
00550       1  4           HighOrderBitMask := HighOrderBitforByte;
00551       1  4           OperandMask := Low8Bits;
00552       1  4       end
00553       1  4       else begin
00554       1  4           HighOrderBitMask := HighOrderBitforWord;
00555       1  4           OperandMask := All16Bits;
00556       1  3       end;
00557       1  3
00558       1  4       for Counter := 1 to RotationCount do begin
00559       1  4           if (Flags.Carry = Clear) then TempCarryValue := AllBitsClear
00560       1  4           else TempCarryValue := LowOrderBit;
00561       1  4           CarryFlagValue := uand(Result, HighOrderBitMask);
00562       1  4           if (CarryFlagValue = AllBitsClear) then Flags.Carry := Clear
00563       1  4           else Flags.Carry := SetHigh;
00564       1  4           Result := uand((Result * 2), OperandMask);
00565       1  4           Result := uor(Result, TempCarryValue);
00566       1  3       end; {for Counter}
00567       1  3
00568       1  4       if (RotationCount = DefaultCount) then begin
00569       1  4           OverflowFlagTest1 := uand(Result, HighOrderBitMask);
00570       1  4           if (OverflowFlagTest1 = AllBitsClear) then TestState := Clear
00571       1  4           else TestState := SetHigh;
00572       1  4           if (TestState <> Flags.Carry) then Flags.Overflow := SetHigh
00573       1  4           else Flags.Overflow := Clear;
00574       1  3       end; {if RotationCount = DefaultCount}
00575       1  3
00576       1  3       StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00577       1  3
00578       1  2     end; {with Opcode %X'2D0',%X'2D2'}
```

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

00580       1  2   %X'3D0',%X'3D2': {rotate right through carry} {C, O}
00581       1  3     with Opcode do begin
00582       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00583       1  3
00584       1  4       if (uand(Full, CountFlag) = CountFlag) then begin
00585       1  4           FetchRegPtr(CLWidth, CLID, Count, LStopCode);
00586       1  4           RotationCount := int(Count);
00587       1  4       end
00588       1  3       else RotationCount := DefaultCount;
00589       1  3
00590       1  4       if (Width = EightBits) then begin
00591       1  4           OperandMask := Low8Bits;
00592       1  4           HighOrderBit := HighOrderBitforByte;
00593       1  4           HighOrderBitMask := HighOrderBitforByte;
00594       1  4           NexttoHighOrderBitMask := NexttoHighOrderBitforByte;
00595       1  4       end
00596       1  4       else begin
00597       1  4           OperandMask := All16Bits;
00598       1  4           HighOrderBit := HighOrderBitforWord;
00599       1  4           HighOrderBitMask := HighOrderBitforWord;
00600       1  4           NexttoHighOrderBitMask := NexttoHighOrderBitforWord;
00601       1  3       end;
00602       1  3
00603       1  4       for Counter := 1 to RotationCount do begin
00604       1  4           if (Flags.Carry = Clear) then TempCarryValue := AllBitsClear
00605       1  4           else TempCarryValue := HighOrderBit;
00606       1  4           CarryFlagValue := uand(Result, LowOrderBitMask);
00607       1  4           if (CarryFlagValue = AllBitsClear) then Flags.Carry := Clear
00608       1  4           else Flags.Carry := SetHigh;
00609       1  4           Result := Result div 2;
00610       1  4           Result := uor(Result, TempCarryValue);
00611       1  3       end; {for Counter}
00612       1  3
00613       1  4       if (RotationCount = DefaultCount) then begin
00614       1  4           OverflowFlagTest1 := uand(Result, HighOrderBitMask);
00615       1  4           OverflowFlagTest1 := (OverflowFlagTest1 div 2);
00616       1  4           OverflowFlagTest2 := uand(Result, NexttoHighOrderBitMask);
00617       1  4           if (OverflowFlagTest1 <> OverflowFlagTest2) then
00618       1  4               Flags.Overflow := SetHigh
00619       1  4           else Flags.Overflow := Clear;
00620       1  3       end; {if RotationCount = DefaultCount}
00621       1  3
00622       1  3       StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00623       1  3
```

```
00624      1  2     end; {with Opcode %X'3D0',%X'3D2'}
```

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

00626       1  2   %X'4D0',%X'4D2': {shift arithmetic left, shift logical left}
00627    C  1  2   {A = undefined, C, O, P, S, Z}
00628       1  3     with Opcode do begin
00629       1  3
00630       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00631       1  3
00632       1  4       if (uand(Full, CountFlag) = CountFlag) then begin
00633       1  4           FetchRegPtr(CLWidth, CLID, Count, LStopCode);
00634       1  4           RotationCount := int(Count);
00635       1  4       end
00636       1  3       else RotationCount := DefaultCount;
00637       1  3
00638       1  4       if (Width = EightBits) then begin
00639       1  4           HighOrderBitMask := HighOrderBitforByte;
00640       1  4           OperandMask := Low8Bits;
00641       1  4       end
00642       1  4       else begin
00643       1  4           HighOrderBitMask := HighOrderBitforWord;
00644       1  4           OperandMask := All16Bits;
00645       1  3       end;
00646       1  3
00647       1  4       for Counter := 1 to RotationCount do begin
00648       1  4           CarryFlagValue := uand(Result, HighOrderBitMask);
00649       1  4           if (CarryFlagValue = AllBitsClear) then Flags.Carry := Clear
00650       1  4           else Flags.Carry := SetHigh;
00651       1  4           Result := uand((Result * 2), OperandMask);
00652       1  3       end; {for Counter}
00653       1  3
00654       1  4       with Flags do begin
00655       1  5           if (RotationCount = DefaultCount) then begin
00656       1  5               OverflowFlagTest1 := uand(Result, HighOrderBitMask);
00657       1  5               if (OverflowFlagTest1 = AllBitsClear) then
00658       1  5                   TestState := Clear
00659       1  5               else TestState := SetHigh;
00660       1  5               if (TestState <> Carry) then Overflow := SetHigh
00661       1  5               else Overflow := Clear;
00662       1  4           end; {if RotationCount = DefaultCount}
00663       1  4
00664       1  4           Parity := UpdateParityFlag(Width, Result);
00665       1  4           Sign := UpdateSignFlag(Width, Result);
00666       1  4           Zero := UpdateZeroFlag(Width, Result);
00667       1  3       end; {with Flags}
00668       1  3
00669       1  3       StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
```

```
00670       1  3
00671       1  2     end; {with Opcode %X'4D0',%X'4D2']
```

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

00673       1  2   %X'5D0',%X'5D2': {shift logical right}
00674    C  1  2   {A = undefined, C, O, P, S, F}
00675       1  3     with Opcode do begin
00676       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00677       1  3
00678       1  4       if (uand(Full, CountFlag) = CountFlag) then begin
00679       1  4           FetchRegPtr(CLWidth, CLID, Count, LStopCode);
00680       1  4           RotationCount := int(Count);
00681       1  4       end
00682       1  3       else RotationCount := DefaultCount;
00683       1  3
00684       1  4       if (Width = EightBits) then begin
00685       1  4           OperandMask := Low8Bits;
00686       1  4           HighOrderBit := HighOrderBitforByte;
00687       1  4           HighOrderBitMask := HighOrderBitforByte;
00688       1  4           NexttoHighOrderBitMask := NexttoHighOrderBitforByte;
00689       1  4       end
00690       1  4       else begin
00691       1  4           OperandMask := All16Bits;
00692       1  4           HighOrderBit := HighOrderBitforWord;
00693       1  4           HighOrderBitMask := HighOrderBitforWord;
00694       1  4           NexttoHighOrderBitMask := NexttoHighOrderBitforWord;
00695       1  3       end;
00696       1  3
00697       1  4       for Counter := 1 to RotationCount do begin
00698       1  4           CarryFlagValue := uand(Result, LowOrderBitMask);
00699       1  4           if (CarryFlagValue = AllBitsClear) then Flags.Carry := Clear
00700       1  4           else Flags.Carry := SetHigh;
00701       1  4           Result := uand((Result div 2), OperandMask);
00702       1  3       end; {for Counter}
00703       1  3
00704       1  4       with Flags do begin
00705       1  5           if (RotationCount = DefaultCount) then begin
00706       1  5               OverflowFlagTest1 := uand(Result, HighOrderBitMask);
00707       1  5               OverflowFlagTest1 := (OverflowFlagTest1 div 2);
00708       1  5               OverflowFlagTest2 := uand(Result, NexttoHighOrderBitMask);
00709       1  5               if (OverflowFlagTest1 <> OverflowFlagTest2) then
00710       1  5                   Overflow := SetHigh
00711       1  5               else Overflow := Clear;
00712       1  4           end; {if RotationCount = DefaultCount}
00713       1  4
00714       1  4           Parity := UpdateParityFlag(Width, Result);
00715       1  4           Sign := UpdateSignFlag(Width, Result);
00716       1  4           Zero := UpdateZeroFlag(Width, Result);
```

```
00717      1  3       end; {with Flags}
00718      1  3
00719      1  3       StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00720      1  3
00721      1  2     end; {with Opcode %X'5D0',%X'5D2'}
```

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

00723       1  2   %X'7D0',%X'7D2': {shift arithmetic right}
00724    C  1  2   {A = undefined, C, O, P, S, F}
00725       1  3     with Opcode do begin
00726       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00727       1  3
00728       1  4       if (uand(Full, CountFlag) = CountFlag) then begin
00729       1  4           FetchRegPtr(CLWidth, CLID, Count, LStopCode);
00730       1  4           RotationCount := int(Count);
00731       1  4       end
00732       1  3       else RotationCount := DefaultCount;
00733       1  3
00734       1  4       if (Width = EightBits) then begin
00735       1  4           OperandMask := Low8Bits;
00736       1  4           HighOrderBitMask := HighOrderBitforByte;
00737       1  4           NexttoHighOrderBitMask := NexttoHighOrderBitforByte;
00738       1  4       end
00739       1  4       else begin
00740       1  4           OperandMask := All16Bits;
00741       1  4           HighOrderBitMask := HighOrderBitforWord;
00742       1  4           NexttoHighOrderBitMask := NexttoHighOrderBitforWord;
00743       1  3       end;
00744       1  3
00745       1  4       for Counter := 1 to RotationCount do begin
00746       1  4           CarryFlagValue := uand(Result, LowOrderBitMask);
00747       1  4           if (CarryFlagValue = AllBitsClear) then Flags.Carry := Clear
00748       1  4           else Flags.Carry := SetHigh;
00749       1  4           SignBit := uand( Result, HighOrderBitMask);
00750       1  4           Result := MaskShiftRight( Result, OperandMask, 2);
00751       1  4           Result := uand( uor( Result, SignBit), OperandMask);
00752       1  3       end; {for Counter}
00753       1  3
00754       1  4       with Flags do begin
00755       1  5           if (RotationCount = DefaultCount) then begin
00756       1  5               OverflowFlagTest1 := uand(Result, HighOrderBitMask);
00757       1  5               OverflowFlagTest1 := (OverflowFlagTest1 div 2);
00758       1  5               OverflowFlagTest2 := uand(Result, NexttoHighOrderBitMask);
00759       1  5               if (OverflowFlagTest1 <> OverflowFlagTest2) then
00760       1  5                   Overflow := SetHigh
00761       1  5               else Overflow := Clear;
00762       1  4           end; {if RotationCount = DefaultCount}
00763       1  4
00764       1  4           Parity := UpdateParityFlag(Width, Result);
00765       1  4           Sign := UpdateSignFlag(Width, Result);
00766       1  4           Zero := UpdateZeroFlag(Width, Result);
```

```
00767     1  3       end; {with Flags}
00768     1  3
00769     1  3       StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00770     1  3
00771     1  2     end; {with Opcode %X'7D0',%X'7D2'}
```

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

00773       1  2   %X'180': {or immediate to register / memory}
00774    C  1  2     {A = undefined, C = 0, O = 0, P, S, Z}
00775       1  3     with Opcode do begin
00776       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Operand1, LStopCode);
00777       1  3       FetchCodeData(Width, Operand2, LStopCode);
00778       1  3       Result := uor(Operand1, Operand2);
00779       1  3       StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00780       1  4       with Flags do begin
00781       1  4           Carry := Clear;
00782       1  4           Overflow := Clear;
00783       1  4           Parity := UpdateParityFlag(Width, Result);
00784       1  4           Sign := UpdateSignFlag(Width, Result);
00785       1  4           Zero := UpdateZeroFlag(Width, Result);
00786       1  3       end; {with Flags}
00787       1  2     end; {with opcode %X'180'}
00788       1  2
00789       1  2   %X'2F6': {not} {no flags effected}
00790       1  3     with Opcode do begin
00791       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Operand1, LStopCode);
00792       1  3       Result := unot(Operand1);
00793       1  3       StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00794       1  2     end; {with opcode %X'2F6'}
00795       1  2
00796       1  2   %X'480': {and immediate to register / memory}
00797    C  1  2     {A = undefined, C = 0, O = 0, P, S, Z}
00798       1  3     with Opcode do begin
00799       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Operand1, LStopCode);
00800       1  3       FetchCodeData(Width, Operand2, LStopCode);
00801       1  3       Result := uand(Operand1, Operand2);
00802       1  3       StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00803       1  4       with Flags do begin
00804       1  4           Carry := Clear;
00805       1  4           Overflow := Clear;
00806       1  4           Parity := UpdateParityFlag(Width, Result);
00807       1  4           Sign := UpdateSignFlag(Width, Result);
00808       1  4           Zero := UpdateZeroFlag(Width, Result);
00809       1  3       end; {with Flags}
00810       1  2     end; {with opcode %X'480'}
```

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

00812       1  2   %X'680': {xor immediate to register / memory}
00813    C  1  2     {A = undefined, C = 0, O = 0, P, S, Z}
00814       1  3     with Opcode do begin
00815       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Operand1, LStopCode);
00816       1  3       FetchCodeData(Width, Operand2, LStopCode);
00817       1  3       Result := uxor(Operand1, Operand2);
00818       1  3       StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Result, LStopCode);
00819       1  4       with Flags do begin
00820       1  4           Carry := Clear;
00821       1  4           Overflow := Clear;
00822       1  4           Parity := UpdateParityFlag(Width, Result);
00823       1  4           Sign := UpdateSignFlag(Width, Result);
00824       1  4           Zero := UpdateZeroFlag(Width, Result);
00825       1  3       end; {with Flags}
00826       1  2     end; {with opcode %X'680'}
00827       1  2
00828       1  2   otherwise LStopCode := BadOpcode;
00829       1  1     end; {case OpcodeKey}
00830       1  1
00831       0  0 end;
00832       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	5912	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	68	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS LOGIC.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]LOGIC.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]LOGIC.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	410	00:00.5	00:05.6
Source Analysis	356	00:02.9	00:18.8
Source Listing	14	00:01.1	00:08.9
Tree Construction	281	00:01.0	00:04.6
Flow Analysis	146	00:01.0	00:03.1
Value Propagation	11	00:00.1	00:00.6
Profit Analysis	24	00:00.5	00:02.6
Context Analysis	158	00:06.0	00:15.6
Name Packing	6	00:00.1	00:00.5
Code Selection	125	00:01.1	00:02.9
Final	70	00:01.8	00:04.9
TOTAL	1606	00:16.1	01:08.7

COMPILATION STATISTICS

```
  CPU Time:       00:16.1         (3106 Lines/Minute)
```

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

00001    C  0  0 {
00002    C  0  0   Title: Look-Up-Table Generation
00003    C  0  0
00004    C  0  0   Purpose: Generation the opcode look-up-table
00005    C  0  0
00006    C  0  0   Author: William A. Chapman      Date: January 18, 1986
00007    C  0  0
00008    C  0  0   Inputs: An external data file "OpcodeLUTFile" for editing and printing
00009    C  0  0
00010    C  0  0   Outputs: An external file "OpcodeLUTFile" is created or updated
00011    C  0  0           An optional file (a printable version) whose name is specified
00012    C  0  0           by the user can also be generated.
00013    C  0  0
00014    C  0  0   Procedures Invoked: none
00015    C  0  0 }
00016       0  0 program OpcodeLUTGen( input, output, OpcodeLUTFile, OpcodeLUTList);
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  [-]LookUpTableClass.defn/list'
00041 I     0  0     ArithmeticClass         = 'A';
00042 I     0  0     ControlTransferClass  = 'C';
00043 I     0  0     DataTransferClass       = 'D';
00044 I     0  0     ExtendedClass   = 'E';
00045 I     0  0     InvalidClass    = 'I';
```

```
00046 I    0  0     LogicClass              = 'L';
00047 I    0  0     ProcessorControlClass = 'P';
00048 I    0  0     StringClass             = 'S';
```

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

00050       0  0     PageSize = 55;
00051       0  0     FF = 12;
00052       0  0
00053       0  0     MinOpcodeKey = 0;
00054       0  0     MaxOpcodeKey = 8;
00055       0  0
00056       0  0     UndefinedOpcodeClass = 'Q';
00057       0  0     UndefinedOpcodeKey = -1;
```

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

00059       0  0 type
00060       0  0     %include '[-]Type.defn/list'
00061 I     0  0     ZeroOne = -1..1;                          {defines integer with range
00062 I C   0  0                                               of zero to one with an invalid
00063 I C   0  0                                               state of -1}
00064 I     0  0
00065 I     0  0     OpcodeLUTEntry = record                   {defines entries in opcode LUT}
00066 I     0  0   OpcodeClass: char;                         {class D, A, L, S, C, P}
00067 I     0  0   OpcodeKey: integer;                        {key to interpret opcode from
00068 I C   0  0                                               0 to 7}
00069 I     0  0   DirectionBitPresent: boolean;
00070 I     0  0   WidthBitPresent: boolean
00071 I     0  0     end;
00072 I     0  0
00073 I     0  0     OpcodeType = record
00074 I     0  0   Direction: ZeroOne;                        {to or from CPU}
00075 I     0  0   Full: unsigned;                            {full opcode}
00076 I     0  0   Width: ZeroOne                            {8 or 16 bits}
00077 I     0  0     end;
00078 I     0  0
00079 I     0  0     RegisterType = record
00080 I     0  0   Id: integer;                              {identifier}
00081 I     0  0   Width: ZeroOne                            {8 or 16 bits}
00082 I     0  0     end;
00083 I     0  0
00084 I     0  0     EffectiveAddressType = record
00085 I     0  0   Mode: char;                               {register or memory}
00086 I     0  0   Width: ZeroOne;                           {8 or 16 bits}
00087 I     0  0   Address: unsigned;                        {memory address or
00088 I C   0  0                                               register designation}
00089 I     0  0   Segment: integer                          {segment register to use}
00090 I     0  0     end;
00091 I     0  0
00092 I     0  0     NameType = packed array[1..10] of char;
00093 I     0  0
00094 I     0  0     Characters = set of    ..'~';
00095 I     0  0     LettersAndNumbers = set of '0'..'z';
00096 I     0  0     Letters = set of 'A'..'z';
00097 I     0  0     Numbers = set of '0'..'9';
00098 I     0  0
00099 I     0  0     FileName = packed array [1..FilenameLength] of char;
```

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

00101       0  0 var
00102       0  0     OpcodeLUTFile: file of OpcodeLUTEntry;
00103       0  0     ListFile: Filename;
00104       0  0     OpcodeLUTList: text;
00105       0  0
00106       0  0     Entry: OpcodeLUTEntry;
00107       0  0     Action: char;
00108       0  0     Index: integer;
00109       0  0     Continue: boolean;
00110       0  0     Correct: boolean;
00111       0  0
00112       0  0     OpcodeLUTable: array [FirstOpcodeValue..LastOpcodeValue] of OpcodeLUTEntry;
00113       0  0
00114       0  0     ValidClasses: LettersAndNumbers;
00115       0  0     Yes: LettersAndNumbers;
00116       0  0     No: LettersAndNumbers;
00117       0  0     Numerals: Numbers;
00118       0  0
00119       0  0     Class: char;
00120       0  0     Key: integer;
00121       0  0     Direction: char;
00122       0  0     Width: char;
00123       0  0
00124       0  0     Valid: boolean;
00125       0  0     Done: boolean;
00126       0  0     Command: char;
00127       0  0
00128       0  0     KeyChar: char;
00129       0  0     NewDirectionBitPresent: boolean;
00130       0  0     NewWidthBitPresent: boolean;
00131       0  0
00132       0  0     %include 'UpperCaseLetter.subpas/list'
00133  I C  0  0 { Title: UpperCaseLetter
00134  I C  0  0
00135  I C  0  0   Purpose: Convert lowercase letter to uppercase
00136  I C  0  0
00137  I C  0  0   Author: William A. Chapman      Date: October 26, 1986
00138  I C  0  0
00139  I C  0  0   Inputs: Upper or lower case letter
00140  I C  0  0
00141  I C  0  0   Outputs: All lowercase letters are converted to uppercase
00142  I C  0  0
00143  I C  0  0   Procedures Invoked: none
00144  I C  0  0 ]
```

```
00145 I    1  0 procedure UpperCaseLetter( var UPLetter: char);
00146 I    1  0
00147 I    1  1 begin
00148 I    1  1     if ((( ord( UPLetter) - ord( 'a')) >= 0 ) and (( ord( UPLetter) - ord( 'z')) <= 0 ))
00149 I    1  1   then UPLetter := chr( ord( UPLetter) - (ord( 'a') - ord( 'A')));
00150 I    0  0 end;
```

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

00152   C  0  0 {
00153   C  0  0   Title: Hexadecimal Input module
00154   C  0  0
00155   C  0  0   Purpose: Obtain a hexadecimal keyboard input string
00156   C  0  0           and convert it to a decimal number.
00157   C  0  0
00158   C  0  0   Author: William A. Chapman      Date: December 28, 1985
00159   C  0  0
00160   C  0  0   Inputs: Keyboard string of hex characters
00161   C  0  0
00162   C  0  0   Outputs: An integer representation of the
00163   C  0  0           character hexadecimal string.
00164   C  0  0
00165   C  0  0   Procedures Invoked: none
00166   C  0  0 }
00167      0  0 var
00168      0  0     HexInChar: char;                    {character to process}
00169      0  0     Temp: integer;                      {temporary variable}
00170      0  0     IHIDone: boolean;
00171      0  0
00172      1  0 procedure IntHexIn(var IntHIValue: integer;
00173      1  0           var IHIValid: boolean);
00174      1  0
00175      1  0
00176      1  0
00177      1  1 begin
00178      1  1     IntHIValue := 0;
00179      1  1     IHIValid := true;
00180      1  1     IHIDone := false;
00181      1  2     while (not IHIDone) do begin
00182      1  2   if (eoln(input)) then IHIDone := true
00183      1  3   else begin
00184      1  3       read(HexInChar);
00185      1  3       Temp := ord(HexInChar) - ord('0');
00186      1  3       if (HexInChar = ' ') then Temp := 0;
00187      1  3       if (Temp > 10) then Temp := Temp + ord('0') - ord('A') + 10;
00188      1  3       if (Temp > 15) then Temp := Temp + ord('A') - ord('a');
00189      1  4       if ((Temp < 0) or (Temp > 15)) then begin
00190      1  4           writeln;
00191      1  4           write('HexIn: Invalid character for conversion: ');
00192      1  4           writeln('[',HexInChar,']');
00193      1  4           writeln('Value of Temp is: ',Temp:1);
00194      1  4           IHIValid := false;
00195      1  4           while (not( eoln( input))) do read( HexInChar);
```

```
00196      1  4           IntHIValue := 0;
00197      1  4       end {invalid character}
00198      1  3       else IntHIValue := (IntHIValue * 16) + Temp;
00199      1  2   end;
00200      1  1     end;
00201      1  1     readln;
00202      0  0 end;
```

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

00204       0  1 begin
00205       0  1     writeln('Opcode Look-Up-Table Manipulation Program');
00206       0  1     ValidClasses := [ ArithmeticClass, ControlTransferClass, DataTransferClass,
00207       0  1   ExtendedClass, LogicClass, ProcessorControlClass, StringClass];
00208       0  1     Yes := ['Y','y','T','t','P','p','1'];
00209       0  1     No := ['N','n','F','f','0'];
00210       0  1     Numerals := ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'];
00211       0  1
00212       0  1     Done := false;
00213       0  2     while (not Done) do begin
00214       0  2   writeln; writeln;
00215       0  2   write('Create (C), Edit (E), Print (P), Exit (X) ? ');
00216       0  2   readln(Action);
00217       0  2
00218       0  3   case Action of
00219       0  3
00220       0  4   'C','c': begin
00221       0  4       write(   'This will destroy any existing information in the ');
00222       0  4       writeln( 'Opcode-Lookup-Table array');
00223       0  4       write(     Are you sure you want to do this ? ');
00224       0  4       readln( Command);
00225       0  5       if (Command in Yes) then begin
00226       0  5           Entry.OpcodeClass := UndefinedOpcodeClass;
00227       0  5           Entry.OpcodeKey := UndefinedOpcodeKey;
00228       0  5           Entry.DirectionBitPresent := true;
00229       0  5           Entry.WidthBitPresent := true;
00230       0  5           for Index := FirstOpcodeValue to LastOpcodeValue do
00231       0  5               OpcodeLUTable[ Index] := Entry;
00232       0  4       end;
00233       0  3   end; {case C}
```

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

00235       0  4   'P','p': begin
00236       0  4       write( 'Enter name of file to contain listing:  );
00237       0  4       readln( ListFile);
00238       0  4       open( OpcodeLUTList, ListFile, new);
00239       0  4       rewrite(OpcodeLUTList);
00240       0  4       reset(OpcodeLUTFile);
00241       0  5       for Index := FirstOpcodeValue to LastOpcodeValue do begin
00242       0  6           if ((Index rem PageSize) = 0) then begin
00243       0  6               if (Index >= PageSize) then writeln(OpcodeLUTList,chr(FF));
00244       0  6               writeln(OpcodeLUTList,'Index Hex Class Key  Dir  Width');
00245       0  5           end;
00246       0  5           read(OpcodeLutFile,Entry);
00247       0  5           write(OpcodeLUTList,Index:4,   ');
00248       0  5           write(OpcodeLUTList, hex(Index,3,2),    ');
00249       0  5           write(OpcodeLUTList,Entry.OpcodeClass,    ');
00250       0  5           write(OpcodeLUTList,Entry.OpcodeKey:2,     ');
00251       0  5           if (Entry.DirectionBitPresent)
00252       0  5               then write(OpcodeLUTList,'P      )
00253       0  5               else write(OpcodeLUTList,'N      );
00254       0  5           if (Entry.WidthBitPresent)
00255       0  5               then write(OpcodeLUTList,'P')
00256       0  5               else write(OpcodeLUTList,'N');
00257       0  5           writeln(OpcodeLUTList);
00258       0  4       end; {for Index}
00259       0  4       close( OpcodeLUTList);
00260       0  3   end; {case P}
```

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

00262       0  4   'E','e': begin
00263       0  4       reset(OpcodeLUTFile);
00264       0  4       for Index := FirstOpcodeValue to LastOpcodeValue do
00265       0  4           read(OpcodeLUTFile,OpcodeLUTable[Index]);
00266       0  4       Continue := true;
00267       0  5       while Continue do begin
00268       0  5           writeln;  writeln;
00269       0  5           writeln( 'Enter opcode value (in hex) to edit');
00270       0  5           write(       number greater than "FF" to exit: ');
00271       0  5           IntHexIn( Index, Valid);
00272       0  5           if ((Index <= LastOpcodeValue) and (Valid = true)) then
00273       0  6               with OpcodeLUTable[Index] do begin
00274       0  6                   write('Class: ',OpcodeClass);
00275       0  6                   write(' Key:  ,OpcodeKey:1);
00276       0  6                   write(' DirectionBitPresent:  ,DirectionBitPresent);
00277       0  6                   writeln(' WidthBitPresent: ',WidthBitPresent);
00278       0  6                   Correct := false;
00279       0  7                   while (not Correct) do begin
00280       0  7                       Correct := true;
00281       0  7
00282       0  7                       write('Enter  new value for class (');
00283       0  7                       write(OpcodeClass,'): ');
00284       0  7                       readln(Class);
00285       0  7                       UpperCaseLetter( Class);
00286       0  8                       if (not (Class in ValidClasses)) then begin
00287       0  8                           Correct := false;
00288       0  8                           writeln('Bad class');
00289       0  7                       end;
00290       0  7
00291       0  7                       write('Enter  new  value for key (');
00292       0  7                       write(OpcodeKey:2,'): ');
00293       0  7                       readln(KeyChar);
00294       0  8                       if (not (KeyChar in Numerals)) then begin
00295       0  8                           Correct := false;
00296       0  8                           writeln( 'Bad key character');
00297       0  8                       end
00298       0  8                       else begin
00299       0  8                           Key := ord( KeyChar) - ord( '0');
00300       0  8                           if ((Key < MinOpcodeKey) and (Key > MaxOpcodeKey))
00301       0  9                           then begin
00302       0  9                               Correct := false;
00303       0  9                               writeln('Bad key');
00304       0  8                           end;
00305       0  7                       end;
```

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

00307       0  7                    write('Is direction bit present ?');
00308       0  7                    if (DirectionBitPresent) then write(' (P): ')
00309       0  7                        else write(' (N): ');
00310       0  7                    readln(Direction);
00311       0  7                    if (Direction in Yes)
00312       0  7                        then NewDirectionBitPresent := true
00313       0  7                        else if (Direction in No)
00314       0  7                            then NewDirectionBitPresent := false
00315       0  8                            else begin
00316       0  8                                Correct := false;
00317       0  8                                writeln('Bad direction');
00318       0  7                            end;
00319       0  7
00320       0  7                    write('Is  width  bit  present  ?');
00321       0  7                    if (WidthBitPresent) then write(' (P): ')
00322       0  7                        else write(' (N): ');
00323       0  7                    readln(Width);
00324       0  7                    if (Width in Yes)
00325       0  7                        then NewWidthBitPresent := true
00326       0  7                        else if (Width in No)
00327       0  7                            then NewWidthBitPresent := false
00328       0  8                            else begin
00329       0  8                                Correct := false;
00330       0  8                                writeln('Bad Width');
00331       0  7                            end;
00332       0  6                end; {while not Correct}
00333       0  6
00334       0  6                OpcodeClass := Class;
00335       0  6                OpcodeKey := Key;
00336       0  6                DirectionBitPresent := NewDirectionBitPresent;
00337       0  6                WidthBitPresent := NewWidthBitPresent;
00338       0  6
00339       0  6            end {with OpcodeLUTable}
00340       0  5        else if (Valid = true) then Continue := false;
00341       0  4    end; {while Continue}
00342       0  4    rewrite(OpcodeLUTFile);
00343       0  4    for Index := FirstOpcodeValue to LastOpcodeValue do
00344       0  4        write(OpcodeLUTFile,OpcodeLUTable[Index]);
00345       0  3  end; {case E}
00346       0  3
00347       0  3  'X','x': Done := true;
00348       0  3
00349       0  3  otherwise writeln('Bad command');
00350       0  3
```

```
00351      0  2   end; {case Action}
00352      0  1     end; {while not Done}
00353      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	3589	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 OPCODELUTGEN.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]OPCODELUTGEN.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]OPCODELUTGEN.OBJ;4
/NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	416	00:00.5	00:02.8
Source Analysis	323	00:01.1	00:04.7
Source Listing	14	00:00.5	00:03.2
Tree Construction	161	00:00.3	00:02.8
Flow Analysis	77	00:00.3	00:00.5
Value Propagation	11	00:00.1	00:00.3
Profit Analysis	35	00:00.2	00:00.4
Context Analysis	283	00:01.9	00:06.5
Name Packing	6	00:00.1	00:00.1
Code Selection	110	00:00.5	00:01.0
Final	67	00:00.8	00:02.8
TOTAL	1507	00:06.3	00:25.1

COMPILATION STATISTICS

CPU Time: 00:06.3 (3389 Lines/Minute)

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

00001    C  0  0 { Title: ParseFilename
00002    C  0  0
00003    C  0  0   Purpose: Isolate a filename and extension if present
00004    C  0  0
00005    C  0  0   Author: William A. Chapman      Date: August 18, 1986
00006    C  0  0
00007    C  0  0   Inputs: Input from operator
00008    C  0  0           Pointer to current position in the input string
00009    C  0  0           Length of the input string
00010    C  0  0
00011    C  0  0   Outputs: A filename is returned if present
00012    C  0  0           Pointer is updated to point to the next character
00013    C  0  0
00014    C  0  0   Procedures Invoked: none
00015    C  0  0 }
00016       0  0 module ParseFileName( 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';
```

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

00122       0  0 type
00123       0  0     %include  [-]Type.defn/list'
00124 I     0  0     ZeroOne = -1..1;                        {defines integer with range
00125 I C   0  0                                              of zero to one with an invalid
00126 I C   0  0                                              state of -1}
00127 I     0  0
00128 I     0  0     OpcodeLUTEntry = record                 {defines entries in opcode LUT}
00129 I     0  0   OpcodeClass: char;                        {class D, A, L, S, C, P}
00130 I     0  0   OpcodeKey: integer;                       {key to interpret opcode from
00131 I C   0  0                                              0 to 7}
00132 I     0  0   DirectionBitPresent: boolean;
00133 I     0  0   WidthBitPresent: boolean
00134 I     0  0     end;
00135 I     0  0
00136 I     0  0     OpcodeType = record
00137 I     0  0   Direction: ZeroOne;                      {to or from CPU}
00138 I     0  0   Full: unsigned;                          {full opcode}
00139 I     0  0   Width: ZeroOne                          {8 or 16 bits}
00140 I     0  0     end;
00141 I     0  0
00142 I     0  0     RegisterType = record
00143 I     0  0   Id: integer;                            {identifier}
00144 I     0  0   Width: ZeroOne                          {8 or 16 bits}
00145 I     0  0     end;
00146 I     0  0
00147 I     0  0     EffectiveAddressType = record
00148 I     0  0   Mode: char;                            {register or memory}
00149 I     0  0   Width: ZeroOne;                        {8 or 16 bits}
00150 I     0  0   Address: unsigned;                     {memory address or
00151 I C   0  0                                           register designation}
00152 I     0  0   Segment: integer                       {segment register to use}
00153 I     0  0     end;
00154 I     0  0
00155 I     0  0     NameType = packed array[1..10] of char;
00156 I     0  0
00157 I     0  0     Characters = set of    ..'~';
00158 I     0  0     LettersAndNumbers = set of '0'..'z';
00159 I     0  0     Letters = set of 'A'..'z';
00160 I     0  0     Numbers = set of '0'..'9';
00161 I     0  0
00162 I     0  0     FileName = packed array [1..FilenameLength] of char;
```

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

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;
00192       0  0
00193       0  0
00194       0  0
00195       0  0 var
00196       0  0     Punctuation: [external] Characters;
00197       0  0     Numerals: [external] Numbers;
00198       0  0     HexNumerals: [external] LettersandNumbers;
00199       0  0
00200       0  0     JournalingFlag: [external] boolean;
00201       0  0     Journal: [external] text;
00202       0  0
00203       0  0     FilenameIndex: integer;
00204       0  0
00205       1  0 [global] procedure ParseFilename( var NullFlag: boolean;
00206       1  0           var NameOfFile: Filename;
00207       1  0           OperatorInput: InputLine;
```

```
00208      1  0          var Pointer: InputPointerRange;
00209      1  0          InputLength: InputPointerRange;
00210      1  0          var EndOfLineFlag: boolean);
```

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

00212       1  1 begin
00213       1  1     for FilenameIndex := 1 to FilenameLength do
00214       1  1   NameOfFile[ FilenameIndex] := Blank;
00215       1  1     FilenameIndex := 1;
00216       1  1     EndOfLineFlag := true;
00217       1  1     NullFlag := true;
00218       1  1
00219       1  1     while (( Pointer <= InputLength) and
00220       1  2   ( OperatorInput[ Pointer] = Blank)) do begin
00221       1  2   Pointer := Pointer + 1;
00222       1  1     end; {while blank]
00223       1  1
00224    C  1  1     { Get filename}
00225       1  2     while ( Pointer <= InputLength) do begin
00226       1  2   NameOfFile[ FilenameIndex] := OperatorInput[ Pointer];
00227       1  2   FileNameIndex := FileNameIndex + 1;
00228       1  2   Pointer := Pointer + 1;
00229       1  2   NullFlag := false;
00230       1  1     end; [while not in Punctuation]
00231       1  1
00232       1  1     if ( Pointer >= InputLength) then Pointer := EndofLine;
00233       0  0 end; {ParseFilename}
00234       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	181	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	4	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
PAS/LIS PARSEFILENAME.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]PARSEFILENAME.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]PARSEFILENAME.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	00:05.3
Source Analysis	230	00:00.7	00:03.8
Source Listing	17	00:00.5	00:05.4
Tree Construction	91	00:00.1	00:00.2
Flow Analysis	38	00:00.0	00:00.0
Value Propagation	11	00:00.0	00:00.2
Profit Analysis	46	00:00.0	00:00.0
Context Analysis	252	00:00.7	00:02.7
Name Packing	7	00:00.0	00:00.0
Code Selection	81	00:00.1	00:01.0
Final	71	00:00.1	00:01.1
TOTAL	1249	00:02.8	00:19.8

COMPILATION STATISTICS

CPU Time: 00:02.8 (4944 Lines/Minute)

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

00001    C  0  0 { Title: Pop Flags
00002    C  0  0
00003    C  0  0   Purpose: Flags are popped off the stack and disassembled
00004    C  0  0
00005    C  0  0   Author: William A. Chapman      Date: January 21, 1986
00006    C  0  0
00007    C  0  0   Inputs: Stack pointers and data
00008    C  0  0
00009    C  0  0   Outputs: Flags are popped off stack and disassembled
00010    C  0  0
00011    C  0  0   Procedures Invoked: PopFromStack
00012    C  0  0 }
00013       0  0 module PopFlags(input, output );
00014       0  0
00015       0  0 const
00016       0  0     %include  [-]Flags.defn/list'
00017 I     0  0     SetHigh = true;
00018 I     0  0     Clear = false;
00019 I     0  0
00020 I     0  0     CarryFlag     = %B'000000000001';
00021 I     0  0     ParityFlag    = %B'000000000100';
00022 I     0  0     AuxCarryFlag  = %B'000000010000';
00023 I     0  0     ZeroFlag      = %B'000001000000';
00024 I     0  0     SignFlag      = %B'000010000000';
00025 I     0  0     TrapFlag      = %B'000100000000';
00026 I     0  0     InterruptFlag = %B'001000000000';
00027 I     0  0     DirectionFlag = %B'010000000000';
00028 I     0  0     OverflowFlag  = %B'100000000000';
00029       0  0
00030       0  0 type
00031       0  0     %include  [-]Flagtype.defn/list'
00032 I     0  0     FlagType = record                          {define processor flags}
00033 I     0  0   Carry: boolean;
00034 I     0  0   Parity: boolean;
00035 I     0  0   AuxCarry: boolean;
00036 I     0  0   Zero: boolean;
00037 I     0  0   Sign: boolean;
00038 I     0  0   Trap: boolean;
00039 I     0  0   Interrupt: boolean;
00040 I     0  0   Direction: boolean;
00041 I     0  0   Overflow: boolean
00042 I     0  0     end;
00043       0  0
00044       0  0 var
00045       0  0     AssembledFlags: unsigned;
```

```
00046    0  0
00047    1  0 [global] procedure PopFlags(var FlagsOut: FlagType;
00048    1  0           var PFStopCode: char);
00049    1  0
00050    2  0 procedure PopFromStack(var PFSValue: unsigned;
00051    1  0           var PFSStopCode: char); external;
```

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

00053       1  1 begin
00054       1  1     PopFromStack(AssembledFlags, PFStopCode);
00055       1  2     with FlagsOut do begin
00056       1  2   if (uand(AssembledFlags,CarryFlag) = 0) then Carry := Clear
00057       1  2       else Carry := SetHigh;
00058       1  2   if (uand(AssembledFlags,ParityFlag) = 0) then Parity := Clear
00059       1  2       else Parity := SetHigh;
00060       1  2   if (uand(AssembledFlags,AuxCarryFlag) = 0) then AuxCarry := Clear
00061       1  2       else AuxCarry := SetHigh;
00062       1  2   if (uand(AssembledFlags,ZeroFlag) = 0) then Zero := Clear
00063       1  2       else Zero := SetHigh;
00064       1  2   if (uand(AssembledFlags,SignFlag) = 0) then Sign := Clear
00065       1  2       else Sign := SetHigh;
00066       1  2   if (uand(AssembledFlags,TrapFlag) = 0) then Trap := Clear
00067       1  2       else Trap := SetHigh;
00068       1  2   if (uand(AssembledFlags,InterruptFlag) = 0) then Interrupt := Clear
00069       1  2       else Interrupt := SetHigh;
00070       1  2   if (uand(AssembledFlags,DirectionFlag) = 0) then Direction := Clear
00071       1  2       else Direction := SetHigh;
00072       1  2   if (uand(AssembledFlags,OverflowFlag) = 0) then Overflow := Clear
00073       1  2       else Overflow := SetHigh;
00074       1  1     end; {with Flags}
00075       0  0 end;
00076       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	237	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	4	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
PAS/LIS POPFLAGS.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]POPFLAGS.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]POPFLAGS.OBJ;1
/NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	408	00:00.5	00:04.9
Source Analysis	212	00:00.4	00:03.7
Source Listing	13	00:00.2	00:03.4
Tree Construction	92	00:00.1	00:00.4
Flow Analysis	34	00:00.0	00:00.3
Value Propagation	10	00:00.0	00:00.1
Profit Analysis	42	00:00.0	00:00.2
Context Analysis	158	00:00.5	00:01.2
Name Packing	6	00:00.0	00:00.0
Code Selection	72	00:00.1	00:00.2
Final	60	00:00.1	00:00.5
TOTAL	1112	00:02.0	00:14.9

COMPILATION STATISTICS

CPU Time: 00:02.0 (2315 Lines/Minute)

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

00001   C  0  0 { Title: Pop from Stack
00002   C  0  0
00003   C  0  0   Purpose: Pop 16 bits of data from the stack
00004   C  0  0
00005   C  0  0   Author: William A. Chapman      Date: January 20, 1986
00006   C  0  0
00007   C  0  0   Inputs: Memory pointed to by stack pointer
00008   C  0  0           Stack Pointer
00009   C  0  0           Stack Segment register
00010   C  0  0
00011   C  0  0   Outputs: Stack pointer is incremented by 2
00012   C  0  0
00013   C  0  0   Procedures Invoked: FetchRegPtr
00014   C  0  0                      StoreRegPtr
00015   C  0  0                      FetchOperand
00016   C  0  0 }
00017      0  0 module PopFromStack(output);
00018      0  0
00019      0  0 const
00020      0  0     %include '[-]RegId.defn/nolist'
00084   C  0  0 {   SPId = %B'100';
00085   C  0  0     SPWidth = 1;    }
00086      0  0
00087      0  0     StackType = 'M';              {memory store}
00088      0  0     StackWidth = 1;               {store 16 bits of data}
00089      0  0     StackSegment = %B'10';
00090      0  0
00091      0  0 var
00092      0  0     StackPointer: unsigned;
00093      0  0
00094      1  0 [global] procedure PopFromStack(var PopData: unsigned;
00095      1  0           var PFSStopCode: char);
00096      1  0
00097      2  0 procedure FetchRegPtr(FRPWidth: integer;
00098      2  0           FRPDesignator: integer;
00099      2  0           var FRPValue: unsigned;
00100      1  0           var FRPStopCode: char); external;
00101      1  0
00102      2  0 procedure StoreRegPtr(SRPWidth: integer;
00103      2  0           SRPDesignator: integer;
00104      2  0           SRPValue: unsigned;
00105      1  0           var SRPStopCode: char); external;
00106      1  0
00107      2  0 procedure FetchOperand(FOType: char;
00108      2  0           FOWidth: integer;
```

```
00109      2  0          FOAddress: unsigned;
00110      2  0          FOSegment: integer;
00111      2  0          var FOValue: unsigned;
00112      1  0          var FOStopCode: char); external;
```

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

00114       1  1 begin
00115       1  1     FetchRegPtr(SPWidth, SPId, StackPointer, PFSStopCode);
00116       1  1     FetchOperand(StackType, StackWidth, StackPointer, StackSegment, PopData, PFSStopCode);
00117       1  1     StackPointer := StackPointer + 2;
00118       1  1     StoreRegPtr(SPWidth, SPId, StackPointer, PFSStopCode);
00119       0  0 end;
00120       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	100	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	4	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

PAS/LIS POPFROMSTACK.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]POPFROMSTACK.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]POPFROMSTACK.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	00:02.5
Source Analysis	204	00:00.3	00:02.6
Source Listing	15	00:00.2	00:02.1
Tree Construction	85	00:00.1	00:00.1
Flow Analysis	28	00:00.1	00:00.5
Value Propagation	9	00:00.0	00:00.0
Profit Analysis	43	00:00.0	00:00.0
Context Analysis	147	00:00.2	00:00.6
Name Packing	7	00:00.0	00:00.0
Code Selection	64	00:00.0	00:00.1
Final	62	00:00.1	00:00.3
TOTAL	1075	00:01.4	00:09.0

COMPILATION STATISTICS

CPU Time: 00:01.4 (5143 Lines/Minute)

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

00001   C  0  0 { Title: Pop (Intersegment) Return Address
00002   C  0  0
00003   C  0  0   Purpose: Pop the Code Segment Register value and
00004   C  0  0            Instruction Pointer value from the stack
00005   C  0  0
00006   C  0  0   Author: William A. Chapman      Date: January 20, 1986
00007   C  0  0
00008   C  0  0   Inputs: Memory pointer to by stack pointer
00009   C  0  0            Stack pointer value
00010   C  0  0
00011   C  0  0   Outputs: Code Segment Register value is updated
00012   C  0  0            InstructionPointer value is updated
00013   C  0  0            Stack pointer is incremented by 4
00014   C  0  0
00015   C  0  0   Procedures Invoked: StoreRegPtr
00016   C  0  0                       PopFromStack
00017   C  0  0 }
00018      0  0 module PopIntersegRA(input, output );
00019      0  0
00020      0  0 const
00021      0  0     %include  [-]RegId.defn/nolist'
00085   C  0  0 [   CSWidth = 2;
00086   C  0  0     CSCode = %B'01';    ]
00087      0  0
00088      0  0 var
00089      0  0     IP: [external] unsigned;
00090      0  0     CSValue: unsigned;
00091      0  0
00092      1  0 [global] procedure PopIntersegRA( var PIRAStopCode: char);
00093      1  0
00094      2  0 procedure StoreRegPtr(SRPWidth: integer;
00095      2  0           SRPDesignator: integer;
00096      2  0           SRPValue: unsigned;
00097      1  0           var SRPStopCode: char); external;
00098      1  0
00099      2  0 procedure PopFromStack(var PFSValue: unsigned;
00100      1  0           var PFSStopCode: char); external;
00101      1  0
00102      1  0
00103      1  0
00104      1  0
00105      1  0
00106      1  1 begin
00107      1  1     PopFromStack(IP, PIRAStopCode);
00108      1  1     PopFromStack(CSValue, PIRAStopCode);
```

```
00109      1  1     StoreRegPtr(CSWidth, CSId, CSValue, PIRAStopCode);
00110      0  0 end;
00111      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	63	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	4	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
PAS/LIS POPINTERSEGRA.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]POPINTERSEGRA.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]POPINTERSEGRA.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	00:02.2
Source Analysis	182	00:00.3	00:00.8
Source Listing	15	00:00.2	00:00.6
Tree Construction	84	00:00.1	00:00.1
Flow Analysis	23	00:00.0	00:00.0
Value Propagation	9	00:00.0	00:00.0
Profit Analysis	43	00:00.0	00:00.0
Context Analysis	125	00:00.1	00:00.1
Name Packing	6	00:00.0	00:00.0
Code Selection	60	00:00.0	00:00.2
Final	56	00:00.1	00:00.3
TOTAL	1011	00:01.2	00:04.3

COMPILATION STATISTICS

CPU Time: 00:01.2 (5644 Lines/Minute)

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

00001    C  0  0 { Title: Processor Control
00002    C  0  0
00003    C  0  0   Purpose: Implement the various Porcessor Control opcodes
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: Opcode Record
00008    C  0  0           Register Record
00009    C  0  0           Effective Address Record
00010    C  0  0
00011    C  0  0   Outputs: Flags are adjusted as needed
00012    C  0  0           Segment Override Flag and Value are adjusted as needed
00013    C  0  0
00014    C  0  0   Procedures Invoked: FetchOperand
00015    C  0  0 }
00016       0  0 module ProcessorControl(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 '[-]Flags.defn/list'
00041  I    0  0     SetHigh = true;
00042  I    0  0     Clear = false;
00043  I    0  0
00044  I    0  0     CarryFlag     = %B'000000000001';
00045  I    0  0     ParityFlag    = %B'000000000100';
00046  I    0  0     AuxCarryFlag  = %B'000000010000';
00047  I    0  0     ZeroFlag      = %B'000001000000';
00048  I    0  0     SignFlag      = %B'000010000000';
00049  I    0  0     TrapFlag      = %B'000100000000';
00050  I    0  0     InterruptFlag = %B'001000000000';
00051  I    0  0     DirectionFlag = %B'010000000000';
00052  I    0  0     OverflowFlag  = %B'100000000000';
00053       0  0
00054       0  0     %include  [-]StopCode.defn/list'
00055  I    0  0     Breakpoint           = 'B';
00056  I    0  0     Call          = 'C';
00057  I    0  0     ControlD            = 'D';
00058  I    0  0     BadOpcode             = 'E';
00059  I    0  0     BadCEAModeValue       = 'F';
00060  I    0  0     BadRMDModeValue       = 'G';
00061  I    0  0     Halt          = 'H';
00062  I    0  0     BadRegisterID = 'I';
00063  I    0  0     BadOpcodeKey  = 'K';
00064  I    0  0     BadMemoryAddress      = 'M';
00065  I    0  0     Normal                = 'N';
00066  I    0  0     BadPortAddress        = 'P';
00067  I    0  0     BadOpcodeClass        = 'Q';
00068  I    0  0     Return                = 'R';
00069  I    0  0     BadCheckSum           = 'S';
00070  I    0  0     BadOperandType        = 'T';
00071  I    0  0     BadMemoryWidth        = 'W';
00072  I    0  0     BadOpcodeExtension    = 'X';
00073  I    0  0     NoMemoryAccess      = 'Z';
```

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

00075       0  0 type
00076       0  0     %include  [-]Type.defn/list
00077 I     0  0     ZeroOne = -1..1;                        {defines integer with range
00078 I C   0  0                                              of zero to one with an invalid
00079 I C   0  0                                              state of -1}
00080 I     0  0
00081 I     0  0     OpcodeLUTEntry = record                 {defines entries in opcode LUT}
00082 I     0  0   OpcodeClass: char;                        {class D, A, L, S, C, P}
00083 I     0  0   OpcodeKey: integer;                       {key to interpret opcode from
00084 I C   0  0                                              0 to 7}
00085 I     0  0   DirectionBitPresent: boolean;
00086 I     0  0   WidthBitPresent: boolean
00087 I     0  0     end;
00088 I     0  0
00089 I     0  0     OpcodeType = record
00090 I     0  0   Direction: ZeroOne;                      {to or from CPU}
00091 I     0  0   Full: unsigned;                          {full opcode}
00092 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00093 I     0  0     end;
00094 I     0  0
00095 I     0  0     RegisterType = record
00096 I     0  0   Id: integer;                            {identifier}
00097 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00098 I     0  0     end;
00099 I     0  0
00100 I     0  0     EffectiveAddressType = record
00101 I     0  0   Mode: char;                            {register or memory}
00102 I     0  0   Width: ZeroOne;                        {8 or 16 bits}
00103 I     0  0   Address: unsigned;                     {memory address or
00104 I C   0  0                                          register designation}
00105 I     0  0   Segment: integer                       {segment register to use}
00106 I     0  0     end;
00107 I     0  0
00108 I     0  0     NameType = packed array[1..10] of char;
00109 I     0  0
00110 I     0  0     Characters = set of    ..'~';
00111 I     0  0     LettersAndNumbers = set of '0'..'z';
00112 I     0  0     Letters = set of 'A'..'z';
00113 I     0  0     Numbers = set of '0'..'9';
00114 I     0  0
00115 I     0  0     FileName = packed array [1..FilenameLength] of char;
00116       0  0
00117       0  0     %include  [-]FlagType.defn/list
00118 I     0  0     FlagType = record                       {define processor flags}
```

```
00119 I     0   0    Carry: boolean;
00120 I     0   0    Parity: boolean;
00121 I     0   0    AuxCarry: boolean;
00122 I     0   0    Zero: boolean;
00123 I     0   0    Sign: boolean;
00124 I     0   0    Trap: boolean;
00125 I     0   0    Interrupt: boolean;
00126 I     0   0    Direction: boolean;
00127 I     0   0    Overflow: boolean
00128 I     0   0      end;
```

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

00130       0  0 var
00131       0  0     Flags: [external] FlagType;
00132       0  0
00133       0  0     SegmentOverRideCount: [external] integer;
00134       0  0     SegmentOverRideValue: [external] integer;
00135       0  0
00136       0  0     OpcodeKey: integer;
00137       0  0
00138       0  0     DataBus: unsigned;
00139       0  0
00140       0  0
00141       0  0
00142       0  0
00143       0  0
00144       1  0 [global] procedure ProcessorControl(Opcode: OpcodeType;
00145       1  0           Register: RegisterType;
00146       1  0           EA: EffectiveAddressType;
00147       1  0           var PCStopCode: char);
00148       1  0
00149       2  0 procedure FetchOperand(FOType: char;
00150       2  0           FOWidth: integer;
00151       2  0           FOAddress: unsigned;
00152       2  0           FOSegment: integer;
00153       2  0           var FOValue: unsigned;
00154       1  0           var FOStopCode: char); external;
```

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

00156       1  1 begin
00157       1  1     OpcodeKey := int(Opcode.Full);
00158       1  1
00159       1  2     case OpcodeKey of
00160       1  2
00161       1  2     %X'26': {Segment OverRide}
00162       1  3       begin
00163       1  3   SegmentOverRideCount := 2;
00164       1  3   SegmentOverRideValue := Register.ID;
00165       1  2       end;
00166       1  2
00167       1  2     %X'9B': {Wait}
00168       1  2   writeln('Wait instruction not supported');
00169       1  2
00170       1  2     %X'D8', %X'D9', %X'DA', %X'DB', %X'DC', %X'DD', %X'DE', %X'DF': {Escape}
00171       1  2   with EA do FetchOperand(Mode, SixteenBits, Address, Segment, DataBus, PCStopCode);
00172       1  2
00173       1  2     %X'F0': {Lock}
00174       1  2   writeln('Bus Lock instruction not supported');
00175       1  2
00176       1  2     %X'F4': {Halt}
00177       1  2   PCStopCode := Halt;
00178       1  2
00179       1  2     %X'F5': {Complement Carry}
00180       1  2   Flags.Carry := not(Flags.Carry);
00181       1  2
00182       1  2     %X'F8': {Clear Carry}
00183       1  2   Flags.Carry := Clear;
00184       1  2
00185       1  2     %X'F9': {Set Carry}
00186       1  2   Flags.Carry := SetHigh;
00187       1  2
00188       1  2     %X'FA': {Clear Interrupt}
00189       1  2   Flags.Interrupt := Clear;
00190       1  2
00191       1  2     %X'FB': {Set Interrupt}
00192       1  2   Flags.Interrupt := SetHigh;
00193       1  2
00194       1  2     %X'FC': {Clear Direction}
00195       1  2   Flags.Direction := Clear;
00196       1  2
00197       1  2     %X'FD': {Set Direction}
00198       1  2   Flags.Direction := SetHigh
00199       1  2
```

```
00200       1  2     otherwise PCStopCode := BadOpcode;
00201       1  2
00202       1  1     end; {case OpcodeKey}
00203       0  0 end;
00204       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	742	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 PROCESSORCONTROL.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]PROCESSORCONTROL.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]PROCESSORCONTROL.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	00:01.4
Source Analysis	256	00:00.7	00:01.5
Source Listing	14	00:00.5	00:01.5
Tree Construction	101	00:00.1	00:00.4
Flow Analysis	52	00:00.1	00:00.1
Value Propagation	9	00:00.0	00:00.0
Profit Analysis	26	00:00.0	00:00.0
Context Analysis	149	00:00.3	00:00.7
Name Packing	7	00:00.0	00:00.2
Code Selection	71	00:00.1	00:00.1
Final	66	00:00.1	00:00.3
TOTAL	1160	00:02.4	00:06.2

COMPILATION STATISTICS

CPU Time: 00:02.4 (5037 Lines/Minute)

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

00001       0  0
00002    C  0  0 { Title: Push Flags
00003    C  0  0
00004    C  0  0   Purpose: Flags are assembled and pushed onto the stack
00005    C  0  0
00006    C  0  0   Author: William A. Chapman      Date: January 21, 1986
00007    C  0  0
00008    C  0  0   Inputs: Flags
00009    C  0  0
00010    C  0  0   Outputs: Stack is updated with flags
00011    C  0  0
00012    C  0  0   Procedures Invoked: PushOnStack
00013    C  0  0 ]
00014       0  0 module PushFlags(input, output );
00015       0  0
00016       0  0 const
00017       0  0     %include  [-]Flags.defn/list'
00018 I     0  0     SetHigh = true;
00019 I     0  0     Clear = false;
00020 I     0  0
00021 I     0  0     CarryFlag     = %B'000000000001';
00022 I     0  0     ParityFlag    = %B'000000000100';
00023 I     0  0     AuxCarryFlag  = %B'000000010000';
00024 I     0  0     ZeroFlag      = %B'000001000000';
00025 I     0  0     SignFlag      = %B'000010000000';
00026 I     0  0     TrapFlag      = %B'000100000000';
00027 I     0  0     InterruptFlag = %B'001000000000';
00028 I     0  0     DirectionFlag = %B'010000000000';
00029 I     0  0     OverflowFlag  = %B'100000000000';
00030       0  0     ClearFlags = 0;
00031       0  0
00032       0  0 type
00033       0  0     %include '[-]Flagtype.defn/list'
00034 I     0  0     FlagType = record                          {define processor flags]
00035 I     0  0   Carry: boolean;
00036 I     0  0   Parity: boolean;
00037 I     0  0   AuxCarry: boolean;
00038 I     0  0   Zero: boolean;
00039 I     0  0   Sign: boolean;
00040 I     0  0   Trap: boolean;
00041 I     0  0   Interrupt: boolean;
00042 I     0  0   Direction: boolean;
00043 I     0  0   Overflow: boolean
00044 I     0  0     end;
00045       0  0
```

```
00046     0  0 var
00047     0  0     AssembledFlags: unsigned;
00048     0  0
00049     1  0 [global] procedure PushFlags(FlagsIn: FlagType;
00050     1  0           var PFStopCode: char);
00051     1  0
00052     2  0 procedure PushOnStack(POSValue: unsigned;
00053     1  0           var POSStopCode: char); external;
```

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

00055       1  1 begin
00056       1  1     AssembledFlags := ClearFlags;
00057       1  2     with FlagsIn do begin
00058       1  2   if (Carry = SetHigh)
00059       1  2       then AssembledFlags := uor(AssembledFlags,CarryFlag);
00060       1  2   if (Parity = SetHigh)
00061       1  2       then AssembledFlags := uor(AssembledFlags,ParityFlag);
00062       1  2   if (AuxCarry = SetHigh)
00063       1  2       then AssembledFlags := uor(AssembledFlags,AuxCarryFlag);
00064       1  2   if (Zero = SetHigh)
00065       1  2       then AssembledFlags := uor(AssembledFlags,ZeroFlag);
00066       1  2   if (Sign = SetHigh)
00067       1  2       then AssembledFlags := uor(AssembledFlags,SignFlag);
00068       1  2   if (Trap = SetHigh)
00069       1  2       then AssembledFlags := uor(AssembledFlags,TrapFlag);
00070       1  2   if (Interrupt = SetHigh)
00071       1  2       then AssembledFlags := uor(AssembledFlags,InterruptFlag);
00072       1  2   if (Direction = SetHigh)
00073       1  2       then AssembledFlags := uor(AssembledFlags,DirectionFlag);
00074       1  2   if (Overflow = SetHigh)
00075       1  2       then AssembledFlags := uor(AssembledFlags,OverflowFlag);
00076       1  1     end; {with Flags}
00077       1  1     PushOnStack(AssembledFlags, PFStopCode);
00078       0  0 end;
00079       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	157	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	4	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
PAS/LIS PUSHFLAGS.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]PUSHFLAGS.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]PUSHFLAGS.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	00:01.7
Source Analysis	216	00:00.4	00:00.6
Source Listing	13	00:00.2	00:01.1
Tree Construction	97	00:00.1	00:00.2
Flow Analysis	39	00:00.1	00:00.3
Value Propagation	12	00:00.0	00:00.0
Profit Analysis	41	00:00.0	00:00.0
Context Analysis	144	00:00.3	00:00.6
Name Packing	6	00:00.0	00:00.0
Code Selection	70	00:00.1	00:00.1
Final	59	00:00.1	00:00.2
TOTAL	1105	00:01.7	00:04.9

COMPILATION STATISTICS

```
CPU Time:       00:01.7         (2805 Lines/Minute)
```

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

00001    C  0  0 { Title: Push (Intersegment) Return Address
00002    C  0  0
00003    C  0  0   Purpose: Push the Code Segment Register value and
00004    C  0  0            Instruction Pointer value on to the stack
00005    C  0  0
00006    C  0  0   Author: William A. Chapman      Date: January 20, 1986
00007    C  0  0
00008    C  0  0   Inputs: Code Segment Register value (from FetchRegptr)
00009    C  0  0            InstructionPointer value (global)
00010    C  0  0            Stack pointer (PushOnStack)
00011    C  0  0
00012    C  0  0   Outputs: Code Segment Register is pushed onto stack
00013    C  0  0            Instruction Pointer is pushed on stack
00014    C  0  0            Stack pointer is decremented by 4
00015    C  0  0
00016    C  0  0   Procedures Invoked: FetchRegPtr
00017    C  0  0                       PushOnStack
00018    C  0  0 ]
00019       0  0 module PushIntersegRA(input, output );
00020       0  0
00021       0  0 const
00022       0  0     %include  [-]RegId.defn/nolist'
00086    C  0  0 {   CSWidth = 2;
00087    C  0  0     CSCode = %B'01 ;     }
00088       0  0
00089       0  0 var
00090       0  0     IP: [external] unsigned;
00091       0  0     CSValue: unsigned;
00092       0  0
00093       1  0 [global] procedure PushIntersegRA( var PIRAStopCode: char);
00094       1  0
00095       2  0 procedure FetchRegPtr(FRPWidth: integer;
00096       2  0           FRPDesignator: integer;
00097       2  0           var FRPValue: unsigned;
00098       1  0           var FRPStopCode: char); external;
00099       1  0
00100       2  0 procedure PushOnStack(POSValue: unsigned;
00101       1  0           var POSStopCode: char); external;
00102       1  0
00103       1  0
00104       1  0
00105       1  0
00106       1  1 begin
00107       1  1     FetchRegPtr( CSWidth, CSId, CSValue, PIRAStopCode);
00108       1  1     PushOnStack( CSValue, PIRAStopCode);
```

```
00109      1  1     PushOnStack( IP, PIRAStopCode);
00110      0  0 end;
00111      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	63	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	4	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
PAS/LIS PUSHINTERSEGRA.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]PUSHINTERSEGRA.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]PUSHINTERSEGRA.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	00:01.5
Source Analysis	183	00:00.2	00:00.7
Source Listing	15	00:00.2	00:00.8
Tree Construction	84	00:00.1	00:00.1
Flow Analysis	23	00:00.0	00:00.2
Value Propagation	9	00:00.0	00:00.0
Profit Analysis	44	00:00.0	00:00.0
Context Analysis	125	00:00.1	00:00.1
Name Packing	7	00:00.0	00:00.0
Code Selection	60	00:00.0	00:00.0
Final	59	00:00.1	00:00.2
TOTAL	1014	00:01.2	00:03.8

COMPILATION STATISTICS

CPU Time: 00:01.2 (5371 Lines/Minute)

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

00001   C  0  0 { Title: Push on Stack
00002   C  0  0
00003   C  0  0   Purpose: Push 16 bits of data onto the stack
00004   C  0  0
00005   C  0  0   Author: William A. Chapman      Date: January 20, 1986
00006   C  0  0
00007   C  0  0   Inputs: Unsigned data to push
00008   C  0  0
00009   C  0  0   Outputs: Memory pointed to by stack pointer - 2 and
00010   C  0  0           stack segment register is used to store data
00011   C  0  0           - Stack pointer is decremented by 2
00012   C  0  0
00013   C  0  0   Procedures Invoked: FetchRegPtr
00014   C  0  0                      StoreRegPtr
00015   C  0  0                      StoreData
00016   C  0  0 }
00017      0  0 module PushOnStack(output);
00018      0  0
00019      0  0 const
00020      0  0     %include '[-]RegId.defn/nolist'
00084   C  0  0 {   SPId = %B'100';
00085   C  0  0     SPWidth = 1;    }
00086      0  0
00087      0  0     StackType = 'M';              {memory store}
00088      0  0     StackWidth = 1;               {store 16 bits of data}
00089      0  0     StackSegment = %B'10';
00090      0  0
00091      0  0 var
00092      0  0     StackPointer: unsigned;
00093      0  0
00094      1  0 [global] procedure PushOnStack(PushData: unsigned;
00095      1  0           var POSStopCode: char);
00096      1  0
00097      2  0 procedure FetchRegPtr(FRPWidth: integer;
00098      2  0           FRPDesignator: integer;
00099      2  0           var FRPValue: unsigned;
00100      1  0           var FRPStopCode: char); external;
00101      1  0
00102      2  0 procedure StoreRegPtr(SRPWidth: integer;
00103      2  0           SRPDesignator: integer;
00104      2  0           SRPValue: unsigned;
00105      1  0           var SRPStopCode: char); external;
00106      1  0
00107      2  0 procedure StoreData(SDType: char;
00108      2  0           SDWidth: integer;
```

```
00109      2  0          SDAddress: unsigned;
00110      2  0          SDSegment: integer;
00111      2  0          SDValue: unsigned;
00112      1  0          var SDStopCode: char); external;
```

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

00114       1  1 begin
00115       1  1     FetchRegPtr(SPWidth, SPId, StackPointer, POSStopCode);
00116       1  1     StackPointer := StackPointer - 2;
00117       1  1     StoreData(StackType, StackWidth, StackPointer, StackSegment, PushData, POSStopCode);
00118       1  1     StoreRegPtr(SPWidth, SPId, StackPointer, POSStopCode);
00119       0  0 end;
00120       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	111	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	4	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS PUSHONSTACK.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]PUSHONSTACK.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]PUSHONSTACK.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	00:00.4	00:01.2
Source Analysis	200	00:00.3	00:00.9
Source Listing	17	00:00.2	00:00.5
Tree Construction	88	00:00.1	00:00.1
Flow Analysis	29	00:00.0	00:00.0
Value Propagation	9	00:00.0	00:00.0
Profit Analysis	42	00:00.0	00:00.2
Context Analysis	149	00:00.2	00:00.4
Name Packing	6	00:00.0	00:00.0
Code Selection	66	00:00.0	00:00.1
Final	58	00:00.1	00:00.9
TOTAL	1074	00:01.4	00:04.2

COMPILATION STATISTICS

```
  CPU Time:       00:01.4         (5217 Lines/Minute)
```

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

00001   C  0  0 { Title: RemoveBreakpoints
00002   C  0  0
00003   C  0  0   Purpose: Remove all breakpoint opcodes
00004   C  0  0
00005   C  0  0   Author: William A. Chapman      Date: September 19, 1986
00006   C  0  0
00007   C  0  0   Inputs: BreakPointTable (global)
00008   C  0  0
00009   C  0  0   Outputs: Normal code is replaced
00010   C  0  0
00011   C  0  0   Procedures Invoked: StoreMemory
00012   C  0  0 }
00013      0  0 module RemoveBreakpoints( 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;
```

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

00037       0  0     %include  [-]DebugConst.defn/list'
00038 I     0  0     FirstKeywordEntry = 0;
00039 I     0  0     LastKeywordEntry  = 100;
00040 I     0  0     UndefinedName = 'Qqqqqqqq';
00041 I     0  0     UndefinedClass = 'z';
00042 I     0  0     UndefinedWidth = -1;
00043 I     0  0     UndefinedID = -2;
00044 I     0  0     PageSize = 55;
00045 I     0  0     FF = 12;
00046 I     0  0     MinID = -1;
00047 I     0  0     MaxID = 25;
00048 I     0  0     MinWidth = 0;
00049 I     0  0     MaxWidth = 3;
00050 I     0  0     SegmentOffset = 16;
00051 I     0  0
00052 I     0  0     BlankClass       =    ;
00053 I     0  0     BreakpointClass  = 'B';
00054 I     0  0     ErrorClass       = 'E';
00055 I     0  0     FlagClass        = 'F';
00056 I     0  0     InputOutputClass = 'I';
00057 I     0  0     MemoryClass   .  = 'M';
00058 I     0  0     NumericClass     = 'N';
00059 I     0  0     OperatorClass    = 'O';
00060 I     0  0     QualifierClass   = 'Q';
00061 I     0  0     RegisterClass    = 'R';
00062 I     0  0     StackClass       = 'S';
00063 I     0  0     TrueFalseClass   = 'T';
00064 I     0  0     UtilityClass     = 'U';
00065 I     0  0     ExecuteClass     = 'X';
00066 I     0  0
00067 I     0  0     Blank  = '   ;
00068 I     0  0     Elipse =  .';
00069 I     0  0
00070 I     0  0     StartOfData = 1;
00071 I     0  0     DataArraySize = 50;
00072 I     0  0
00073 I     0  0     EndofLine = 0;
00074 I     0  0     StartofLine = 1;
00075 I     0  0
00076 I     0  0     ShortStringLength = 8;
00077 I     0  0     InputLineLength = 120;
00078 I     0  0
00079 I     0  0     BooleanID  = 1;
00080 I     0  0     ByteID     = 2;
```

```
00081 I     0   0      IntegerID  = 3;
00082 I     0   0      PointerID  = 4;
00083 I     0   0      SIntegerID = 5;
00084 I     0   0      WordID     = 6;
00085 I     0   0      LengthID   = 3;
00086 I     0   0      ToID       = 5;
00087 I     0   0
00088 I     0   0      AssignmentID = 1;
00089 I     0   0      ColonID      = 2;
00090 I     0   0      PlusID       = 3;
00091 I     0   0      MinusID      = 4;
```

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

00092 I     0  0      CommaID      = 5;
00093 I     0  0
00094 I     0  0      NullEntries = 0;
00095 I     0  0
00096 I     0  0      AddressState   = 'A';
00097 I     0  0      BooleanState   = 'B';
00098 I     0  0      ErrorState     = 'E';
00099 I     0  0      HexState       = 'H';
00100 I     0  0      IntegerState   = 'I';
00101 I     0  0      SegmentState   = 'S';
00102 I     0  0      SeparatorState =  ,';
00103 I     0  0      UnaryState     = '-';
00104 I     0  0      ColonState     =  :';
00105 I     0  0
00106 I     0  0      NegativeOperator = -1;
00107 I     0  0      PositiveOperator = +1;
00108 I     0  0
00109 I     0  0      NullSegmentValue = 0;
00110 I     0  0      NullAddressValue = 0;
00111 I     0  0
00112 I     0  0      FirstBreakpoint  =  0;
00113 I     0  0      LastBreakpoint   = 15;
00114 I     0  0      AllBPIDs         = 16;
00115 I     0  0      Activate         = true;
00116 I     0  0      DeActivate       = false;
00117 I     0  0      BreakpointOpcode = %X'CC';
```

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

00119       0  0 type
00120       0  0     %include '[-]Type.defn/list'
00121 I     0  0     ZeroOne = -1..1;                        {defines integer with range
00122 I C   0  0                                              of zero to one with an invalid
00123 I C   0  0                                              state of -1}
00124 I     0  0
00125 I     0  0     OpcodeLUTEntry = record                 {defines entries in opcode LUT}
00126 I     0  0   OpcodeClass: char;                        {class D, A, L, S, C, P}
00127 I     0  0   OpcodeKey: integer;                       {key to interpret opcode from
00128 I C   0  0                                              0 to 7}
00129 I     0  0   DirectionBitPresent: boolean;
00130 I     0  0   WidthBitPresent: boolean
00131 I     0  0     end;
00132 I     0  0
00133 I     0  0     OpcodeType = record
00134 I     0  0   Direction: ZeroOne;                      {to or from CPU}
00135 I     0  0   Full: unsigned;                          {full opcode}
00136 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00137 I     0  0     end;
00138 I     0  0
00139 I     0  0     RegisterType = record
00140 I     0  0   Id: integer;                            {identifier}
00141 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00142 I     0  0     end;
00143 I     0  0
00144 I     0  0     EffectiveAddressType = record
00145 I     0  0   Mode: char;                             {register or memory}
00146 I     0  0   Width: ZeroOne;                         {8 or 16 bits}
00147 I     0  0   Address: unsigned;                      {memory address or
00148 I C   0  0                                              register designation}
00149 I     0  0   Segment: integer                        {segment register to use}
00150 I     0  0     end;
00151 I     0  0
00152 I     0  0     NameType = packed array[1..10] of char;
00153 I     0  0
00154 I     0  0     Characters = set of   '..'~';
00155 I     0  0     LettersAndNumbers = set of '0'..'z';
00156 I     0  0     Letters = set of 'A'..'z';
00157 I     0  0     Numbers = set of '0'..'9';
00158 I     0  0
00159 I     0  0     FileName = packed array [1..FilenameLength] of char;
```

Missing Page

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

00161       0  0     %include  [-]DebugType.defn/list'
00162 I     0  0     NameString = packed array[1..ShortStringLength] of char;
00163 I     0  0
00164 I     0  0     KeywordCharacteristics = record
00165 I     0  0   Name: NameString;
00166 I     0  0   Class: char;
00167 I     0  0   Width: ZeroOne;
00168 I     0  0   ID: integer;
00169 I     0  0   Value: unsigned
00170 I     0  0     end;
00171 I     0  0
00172 I     0  0     InputPointerRange = integer;
00173 I     0  0     InputLine = packed array [1..InputLineLength] of char;
00174 I     0  0
00175 I     0  0     ShortString = packed array [1..ShortStringLength] of char;
00176 I     0  0
00177 I     0  0     DataArrayType = array [StartOfData..DataArraySize] of unsigned;
00178 I     0  0
00179 I     0  0     BreakRecord = record
00180 I     0  0   Encountered: boolean;
00181 I     0  0   Activated: boolean;
00182 I     0  0   Segment: unsigned;
00183 I     0  0   Address: unsigned;
00184 I     0  0   PhysicalAddress: unsigned;
00185 I     0  0   Code: unsigned
00186 I     0  0     end;
00187 I     0  0
00188 I     0  0     BreakPointTableType = array [FirstBreakpoint..LastBreakpoint] of BreakRecord;
00189       0  0
00190       1  0 procedure StoreMemory(SMAddress: unsigned;
00191       1  0           SMSegment: unsigned;
00192       1  0           SMValue: unsigned;
00193       0  0           var SMStopCode: char); external;
00194       0  0
00195       0  0
00196       1  0 [global] procedure RemoveBreakpoints( var BreakPointTable: BreakPointTableType;
00197       1  0           var RBStopCode: char);
00198       1  0
00199       1  0
00200       1  0
00201       1  0
00202       1  0 var
00203       1  0     RBBreakpointLoop: integer;
00204       1  0
```

```
00205      1  1 begin
00206      1  1     for RBBreakpointLoop := FirstBreakpoint to LastBreakpoint do
00207      1  1   with BreakPointTable[ RBBreakpointLoop] do
00208      1  1       if (Activated = true) then StoreMemory( Address, Segment, Code, RBStopCode);
00209      0  0 end;
00210      0  0 end.
```

Missing Page

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	64	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS REMOVEBREAKPOINTS.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]REMOVEBREAKPOINTS.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]REMOVEBREAKPOINTS.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	240	00:00.7	00:02.1
Source Listing	17	00:00.4	00:01.1
Tree Construction	102	00:00.1	00:00.1
Flow Analysis	34	00:00.0	00:00.0
Value Propagation	11	00:00.0	00:00.0
Profit Analysis	45	00:00.0	00:00.0
Context Analysis	179	00:00.1	00:00.2
Name Packing	7	00:00.0	00:00.0
Code Selection	73	00:00.0	00:00.0
Final	67	00:00.1	00:00.4
TOTAL	1179	00:01.9	00:05.2

COMPILATION STATISTICS

```
  CPU Time:       00:01.9         (6528 Lines/Minute)
  Elapsed Time:   00:05.2
```

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

00001    C  0  0 { Title: ReviewIOData
00002    C  0  0
00003    C  0  0   Purpose: Generate or modify the input port data
00004    C  0  0            and view input or output data
00005    C  0  0
00006    C  0  0   Author: William A. Chapman      Date: May 17, 1986
00007    C  0  0
00008    C  0  0   Inputs: The IO map must already exist in a disk file.
00009    C  0  0            The data to be input to the program is requested
00010    C  0  0            from the user and store in the apprpriate format.
00011    C  0  0
00012    C  0  0   Outputs: The data is input, modified, displayed, or printed
00013    C  0  0            and then stored on disk.
00014    C  0  0
00015    C  0  0   Procedures Invoked: HexIn
00016    C  0  0            (Get8BitsofData)
00017    C  0  0            (Put8BitsofData)
00018    C  0  0            (SearchForPortAddress)
00019    C  0  0            MaskShiftRight
00020    C  0  0            FileExists.FORTRAN
00021    C  0  0 }
00022       0  0 program ReviewIOData( input, output, IODataFile, IOMapFile, ListFile);
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  [-]IOConst.defn/list'
00047 I     0  0     KeyboardInPortNumber  = %X'FF';
00048 I     0  0     CRTOutPortNumber       = %X'FF';
00049 I     0  0
00050 I     0  0     FirstIOPortIndex   = 1;
00051 I     0  0     MaxNumberofIOPorts = 8;
00052 I     0  0
00053 I     0  0     FirstIODataPoint   = 1;
00054 I     0  0     MaxNumberofIOWords = 64;
00055 I     0  0     MaxNumberofIOBytes = 256;     {4 times the number of words}
```

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

00056 I     0  0
00057 I     0  0      InputIndicator  = 'I';
00058 I     0  0      OutputIndicator = 'O';
00059 I     0  0      IOIndicator     = 'B';
00060 I     0  0
00061 I     0  0      LowerPortIndicator = 'L';
00062 I     0  0      UpperPortIndicator = 'U';
00063 I     0  0
00064 I     0  0      InvalidAddress = - 1;
00065 I     0  0      InvalidLocation = - 1;
00066 I     0  0      InvalidIndex = - 1;
00067       0  0
00068       0  0      %include  [-]Byte.defn/list'
00069 I     0  0      Byte0Mask = %X'000000FF';
00070 I     0  0      Byte1Mask = %X'0000FF00';
00071 I     0  0      Byte2Mask = %X'00FF0000';
00072 I     0  0      Byte3Mask = %X'FF000000';
00073 I     0  0
00074 I     0  0      Byte0Divisor = %X'00000001';
00075 I     0  0      Byte1Divisor = %X'00000100';
00076 I     0  0      Byte2Divisor = %X'00010000';
00077 I     0  0      Byte3Divisor = %X'01000000';
00078 I     0  0
00079 I     0  0      Byte0Multiplier = %X'00000001';
00080 I     0  0      Byte1Multiplier = %X'00000100';
00081 I     0  0      Byte2Multiplier = %X'00010000';
00082 I     0  0      Byte3Multiplier = %X'01000000';
00083 I     0  0
00084 I     0  0      BytesPerWord = 4;
00085       0  0
00086       0  0      %include '[-]ReviewIOData.version/list'
00087 I     0  0      VersionNo = 1;
00088 I     0  0      ReleaseNo = 2;
00089       0  0
00090       0  0      DisplayEntries8Bits = 24;
00091       0  0      DisplayEntries16Bits = 14;
00092       0  0      DisplaywIndex8Bits = 8;
00093       0  0      DisplaywIndex16Bits = 6;
00094       0  0      PrintEntries8Bits = 24;
00095       0  0      PrintEntries16Bits = 14;
00096       0  0
00097       0  0      NullDatum = 0;
```

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

00099       0  0 type
00100       0  0     %include '[-]Type.defn/nolist'
00140    C  0  0 {   FileName = array[1..FilenameLength] of char;}
00141       0  0
00142       0  0     %include '[-]IOType.defn/list'
00143 I     0  0     PortEntry = record
00144 I     0  0   Address: unsigned;
00145 I     0  0   AuxIndex: integer;
00146 I     0  0   DataIndex: integer;
00147 I     0  0   InputIndex: integer;
00148 I     0  0   PortWidth: integer;
00149 I     0  0   LowerUpperIndicator: char;
00150 I     0  0   InOutIndicator: char
00151 I     0  0     end;
00152 I     0  0
00153 I     0  0     PortMap = array[ FirstIOPortIndex..MaxNumberOfIOPorts] of PortEntry;
```

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

00155       0  0 var
00156       0  0     IOData: array[FirstIOPortIndex..MaxNumberofIOPorts,
00157       0  0   FirstIODataPoint..MaxNumberofIOWords] of unsigned;
00158       0  0
00159       0  0     IOPortMap: PortMap;
00160       0  0
00161       0  0     ListFile: text;
00162       0  0     ListFilename: FileName;
00163       0  0
00164       0  0     IOMapFilename: FileName;
00165       0  0     IOMapFile: file of PortEntry;
00166       0  0     IODataFilename: FileName;
00167       0  0     IODataFile: file of unsigned;
00168       0  0
00169       0  0     PortIndex: integer;
00170       0  0     PortAddress: unsigned;
00171       0  0     PortAddressPresent: boolean;
00172       0  0
00173       0  0     ModifyCommand: char;
00174       0  0     Finished: boolean;
00175       0  0
00176       0  0     DataLoop: integer;
00177       0  0     Datum: unsigned;
00178       0  0     UpperDatum: unsigned;
00179       0  0     LowerDatum: unsigned;
00180       0  0
00181       0  0     Command: char;
00182       0  0     Done: boolean;
00183       0  0
00184       0  0     QuitResponse: char;
00185       0  0
00186       0  0     NumberOfPoints: integer;
00187       0  0
00188       0  0     AppendNumber: integer;
00189       0  0
00190       0  0     RemoveLocation: integer;
00191       0  0     RemoveNumber: integer;
00192       0  0
00193       0  0     InsertLocation: integer;
00194       0  0     InsertNumber: integer;
00195       0  0
00196       0  0     ChangeLocation: integer;
00197       0  0
00198       0  0     DisplayIndex: integer;
```

```
00199       0  0     PrintIndex: integer;
00200       0  0
00201       0  0     Exists: boolean;
```

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

00203       0  0 function FileExists( FEFileName: Filename): boolean; Fortran;
00204       0  0
00205       0  0 procedure HexIn(var HIValue: unsigned); external;
00206       0  0
00207       1  0 function MaskShiftRight( MSRValue: unsigned;
00208       1  0           MSRMask: unsigned;
00209       0  0           MSRDivisor: integer): unsigned; external;
```

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

00211       0  0     %include 'Get8BitsofData.subpas/list'
00212  I C  0  0 { Title: Get8BitsofData
00213  I C  0  0
00214  I C  0  0   Purpose: Retreive 8 bits of data from the IO data buffer
00215  I C  0  0            and return to the calling routine.
00216  I C  0  0
00217  I C  0  0   Author: William A. Chapman      Date: May 16, 1986
00218  I C  0  0
00219  I C  0  0   Inputs: IO data is read from IO data buffer.
00220  I C  0  0
00221  I C  0  0   Outputs: Data is returned to calling routine.
00222  I C  0  0
00223  I C  0  0   Procedures Invoked: MaskShiftRight
00224  I C  0  0 }
00225  I    1  0 procedure Get8BitsofData( PortPointer: integer;
00226  I    1  0           Location: integer;
00227  I    1  0           var Value: unsigned);
00228  I    1  0
00229  I    1  0
00230  I    1  0 var
00231  I    1  0     WordPointer: integer;
00232  I    1  0     BytePointer: integer;
00233  I    1  0
00234  I    1  0
00235  I    1  0
00236  I    1  1 begin
00237  I    1  1     WordPointer := (((Location - 1) div BytesPerWord) + 1);
00238  I    1  1     BytePointer := ((Location - 1)  rem BytesPerWord);
00239  I    1  1     Value := NullDatum;
00240  I    1  1
00241  I    1  2     case BytePointer of
00242  I    1  2
00243  I    1  2     0: Value:= MaskShiftRight( IOData[ PortPointer, WordPointer],
00244  I    1  2           Byte0Mask, Byte0Divisor);
00245  I    1  2
00246  I    1  2     1: Value:= MaskShiftRight( IOData[ PortPointer, WordPointer],
00247  I    1  2           Byte1Mask, Byte1Divisor);
00248  I    1  2
00249  I    1  2     2: Value:= MaskShiftRight( IOData[ PortPointer, WordPointer],
00250  I    1  2           Byte2Mask, Byte2Divisor);
00251  I    1  2                                    .
00252  I    1  2     3: Value:= MaskShiftRight( IOData[ PortPointer, WordPointer],
00253  I    1  2           Byte3Mask, Byte3Divisor);
00254  I    1  2
```

```
00255 I    1  2     otherwise writeln('Get8BitsofData: Invalid byte pointer:  , BytePointer:1);
00256 I    1  2
00257 I    1  1     end; {case BytePointer}
00258 I    1  1
00259 I    0  0 end; {procedure Get8BitsofData}
```

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

00261       0  0     %include 'Put8BitsofData.subpas/list'
00262 I C   0  0 { Title: Put8BitsofData
00263 I C   0  0
00264 I C   0  0   Purpose: Take 8 bits of data passed from the calling routine
00265 I C   0  0            and store it in the IO data buffer.
00266 I C   0  0
00267 I C   0  0   Author: William A. Chapman      Date: May 17, 1986
00268 I C   0  0
00269 I C   0  0   Inputs: Input Data is obtained from the calling routine.
00270 I C   0  0
00271 I C   0  0   Outputs: Data is stored in the IO data buffer.
00272 I C   0  0
00273 I C   0  0   Procedures Invoked: none
00274 I C   0  0 }
00275 I     1  0 procedure Put8BitsofData( PortPointer: integer;
00276 I     1  0           Location: integer;
00277 I     1  0           Value: unsigned);
00278 I     1  0
00279 I     1  0 var
00280 I     1  0     Unchanged: unsigned;
00281 I     1  0     NewDatum: unsigned;
00282 I     1  0
00283 I     1  0     WordPointer: integer;
00284 I     1  0     BytePointer: integer;
```

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

00286 I     1  1 begin
00287 I     1  1     WordPointer := (((Location - 1) div BytesPerWord) + 1);
00288 I     1  1     BytePointer := ((Location - 1)  rem BytesPerWord);
00289 I     1  1
00290 I     1  2     case BytePointer of
00291 I     1  2
00292 I     1  3     0: begin
00293 I     1  3   Unchanged := uand( IOData[PortPointer,WordPointer], unot(Byte0Mask));
00294 I     1  3   NewDatum := uand( (Value * Byte0Multiplier), Byte0Mask);
00295 I     1  3   IOData[PortPointer,WordPointer] := uor( Unchanged, NewDatum);
00296 I     1  2        end;
00297 I     1  2
00298 I     1  3     1: begin
00299 I     1  3   Unchanged := uand( IOData[PortPointer,WordPointer], unot(Byte1Mask));
00300 I     1  3   NewDatum := uand( (Value * Byte1Multiplier), Byte1Mask);
00301 I     1  3   IOData[PortPointer,WordPointer] := uor( Unchanged, NewDatum);
00302 I     1  2        end;
00303 I     1  2
00304 I     1  3     2: begin
00305 I     1  3   Unchanged := uand( IOData[PortPointer,WordPointer], unot(Byte2Mask));
00306 I     1  3   NewDatum := uand( (Value * Byte2Multiplier), Byte2Mask);
00307 I     1  3   IOData[PortPointer,WordPointer] := uor( Unchanged, NewDatum);
00308 I     1  2        end;
00309 I     1  2
00310 I     1  3     3: begin
00311 I     1  3   Unchanged := uand( IOData[PortPointer,WordPointer], unot(Byte3Mask));
00312 I     1  3   NewDatum := uand( (Value * Byte3Multiplier), Byte3Mask);
00313 I     1  3   IOData[PortPointer,WordPointer] := uor( Unchanged, NewDatum);
00314 I     1  2        end;
00315 I     1  2
00316 I     1  2     otherwise writeln( 'Put8BitsOfData: Invalid Byte Pointer:  , BytePointer:1);
00317 I     1  2
00318 I     1  1     end; [case BytePointer]
00319 I     0  0 end; [Put8BitsofData]
```

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

00321       0  0     %include 'SearchforPortAddress.subpas/list'
00322 I C   0  0 { Title: SearchForPortAddress
00323 I C   0  0
00324 I C   0  0   Purpose: Find a specified port address
00325 I C   0  0
00326 I C   0  0   Author: William A. Chapman      Date: May 11, 1986
00327 I C   0  0
00328 I C   0  0   Inputs: PortMap
00329 I C   0  0           Target Port Address
00330 I C   0  0
00331 I C   0  0   Outputs: PortFound - boolean flag
00332 I C   0  0           PortLocation - index of port
00333 I C   0  0
00334 I C   0  0   Procedures Called: none
00335 I C   0  0 }
00336 I     1  0 procedure SearchForPortAddress( SFPATargetPortAddress: unsigned;
00337 I     1  0           var SFPASearchPortMap: PortMap;
00338 I     1  0           var SFPAPortFound: boolean;
00339 I     1  0           var SFPAPortLocation: integer);
00340 I     1  0
00341 I     1  0 var
00342 I     1  0     SFPAIndex: integer;
00343 I     1  0     SFPADone: boolean;
00344 I     1  0
00345 I     1  1 begin
00346 I     1  1     SFPAIndex := FirstIOPortIndex;
00347 I     1  1     SFPAPortFound := false;
00348 I     1  1     SFPADone := false;
00349 I     1  1
00350 I     1  2     while (not SFPADone) do begin
00351 I     1  3   if (SFPASearchPortMap[ SFPAIndex].Address = SFPATargetPortAddress) then begin
00352 I     1  3       SFPAPortFound := true;
00353 I     1  3       SFPAPortLocation := SFPAIndex;
00354 I     1  3       SFPADone := true;
00355 I     1  3   end
00356 I     1  2   else SFPAIndex := SFPAIndex + 1;
00357 I     1  3   if (SFPAIndex > MaxNumberofIOPorts) then begin
00358 I     1  3       SFPAPortFound := false;
00359 I     1  3       SFPAPortLocation := InvalidLocation;
00360 I     1  3       SFPADone := true;
00361 I     1  2   end;
00362 I     1  1     end;  {while (not SFPADone)}
00363 I     0  0 end; {SearchForPortAddress}
```

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

00365       0  1 begin
00366       0  1     writeln;  writeln;
00367       0  1     writeln(              Intel 8086 Simulator - Review I/O Data Utility');
00368       0  1     writeln(                                    Version ', VersionNo:1, '.', ReleaseNo:1 );
00369       0  1     writeln;  writeln;
00370       0  1
00371       0  2     repeat
00372       0  2   write( 'Enter name of file containing map: ');
00373       0  2   readln( IOMapFilename);
00374       0  2   Exists := FileExists( %descr IOMapFileName);
00375       0  3   if (Exists = true) then begin
00376       0  3       open( IOMapFile, IOMapFilename, old);
00377       0  3       reset( IOMapFile);
00378       0  3       for PortIndex := FirstIOPortIndex to MaxNumberofIOPorts do
00379       0  3           read( IOMapFile, IOPortMap[PortIndex]);
00380       0  3       close( IOMapFile);
00381       0  3   end
00382       0  2   else writeln( 'File  , IOMapFilename,   does not exist');
00383       0  1     until Exists;
00384       0  1
00385       0  1     Done := false;
00386       0  2     while (not Done) do begin
00387       0  2   writeln;  writeln;
00388       0  2   writeln('Enter desired operation to perform on data file: ');
00389       0  2   writeln( '  Create file (C), Read file (R), Modify data (M), Input data (I),  );
00390       0  2   write(      Display data (D), Print data (P), Write (W), Quit (Q), Exit (E):  );
00391       0  2   readln(Command);
00392       0  2   writeln;
00393       0  2
00394       0  3   case Command of
```

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

00396       0  4     'I','i': begin {input data}
00397       0  4       writeln;  writeln;
00398       0  4       PortAddressPresent := false;
00399       0  5       while (not PortAddressPresent) do begin
00400       0  5         write( 'Enter port address : ');
00401       0  5         HexIn( PortAddress);
00402       0  5         SearchForPortAddress( PortAddress, IOPortMap, PortAddressPresent, PortIndex);
00403       0  5         if (PortAddressPresent) then with IOPortMap[PortIndex] do
00404       0  5          if (InOutIndicator = OutputIndicator) then
00405       0  5           writeln( 'Cannot enter data for an output port')
00406       0  6          else begin
00407       0  6           write( 'Enter number of input points (in decimal):  );
00408       0  6           readln( NumberofPoints);
00409       0  6           writeln( 'Enter each data point in hex:');
00410       0  6           write( '  (2 characters for 8 bit ports and  );
00411       0  6           write( '  4 characters for 16 bit ports.)');
00412       0  6           writeln;
00413       0  7           for DataLoop := FirstIODataPoint to NumberofPoints do begin
00414       0  7             write( '  Point Number ', DataLoop:3, ': ');
00415       0  7             HexIn( Datum);
00416       0  7             InputIndex := InputIndex + 1;
00417       0  8             case PortWidth of
00418       0  8
00419       0  9               EightBits: begin
00420       0  9                 LowerDatum := uand( Datum, Low8Bits);
00421       0  9                 Put8BitsofData( PortIndex, DataLoop, LowerDatum);
00422       0  8                 end; {case EightBits}
00423       0  8
00424       0  9               SixteenBits: begin
00425       0  9                 LowerDatum := uand( Datum, Low8Bits);
00426       0  9                 UpperDatum := MaskShiftRight( Datum, High8Bits, Bit8Divisor);
00427       0 10                 if (LowerUpperIndicator = LowerPortIndicator) then begin
00428       0 10                   Put8BitsofData( PortIndex, DataLoop, LowerDatum);
00429       0 10                   Put8BitsofData( AuxIndex, DataLoop, UpperDatum);
00430       0 10                   end
00431       0 10
00432       0 10                 else begin
00433       0 10                   Put8BitsofData( AuxIndex, DataLoop, LowerDatum);
00434       0 10                   Put8BitsofData( PortIndex, DataLoop, UpperDatum);
00435       0  9                 end; {else begin}
00436       0  9                 IOPortMap[AuxIndex].InputIndex := IOPortMap[AuxIndex].InputIndex + 1;
00437       0  8               end; {case SixteenBits}
00438       0  8
00439       0  7               end; {case}
```

```
00440      0  6           end; {for DataLoop}
00441      0  6         end {if (PortAddressPresent)}
00442      0  5         else writeln( 'Port address not found:  ,hex( PortAddress, 4, 4));
00443      0  4       end; {while (not PortAddressPresent)}
00444      0  3     end; {case I}
```

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

00446       0  4     'M','m': begin {modify}
00447       0  4       writeln;  writeln;
00448       0  4       Finished := false;
00449       0  4       write( 'Enter port address : ');
00450       0  4       HexIn( PortAddress);
00451       0  4       SearchForPortAddress( PortAddress, IOPortMap, PortAddressPresent,   PortIndex);
00452       0  4       if (PortAddressPresent) then with IOPortMap[PortIndex] do
00453       0  4          if (InOutIndicator = OutputIndicator) then
00454       0  4           writeln( 'Cannot modify data for an output port')
00455       0  5          else while (not Finished) do begin
00456       0  5           writeln; writeln;
00457       0  5           write( 'All operations are done to port  ,hex( PortAddress, 4, 4),  .');
00458       0  5           writeln( '   Do you want to :');
00459       0  5           write('  Append data (A), Insert data (I), Remove data (R),');
00460       0  5           writeln( ' Change data (C),');
00461       0  5           write(  '  Display data with index (D), or are you Finished (F) :  );
00462       0  5           readln(ModifyCommand);
00463       0  5           writeln;  writeln;
00464       0  5
00465       0  6           case ModifyCommand of
00466       0  6
00467       0  7           'C','c': begin {change}
00468       0  7             write( 'Enter the location of the point to be changed : ');
00469       0  7             readln( ChangeLocation);
00470       0  7             write( '  New value for Point Number  , ChangeLocation:3,  : ');
00471       0  7             HexIn( Datum);
00472       0  8             case PortWidth of
00473       0  8
00474       0  9             EightBits: begin
00475       0  9               LowerDatum := uand( Datum, Low8Bits);
00476       0  9               Put8BitsofData( PortIndex, ChangeLocation, LowerDatum);
00477       )  8               end; {case EightBits}
00478       0  8
00479       0  9             SixteenBits: begin
00480       0  9               LowerDatum := uand( Datum, Low8Bits);
00481       0  9               UpperDatum := MaskShiftRight( Datum, High8Bits, Bit8Divisor);
00482       0 10               if (LowerUpperIndicator = LowerPortIndicator) then begin
00483       0 10                 Put8BitsofData( PortIndex, ChangeLocation, LowerDatum);
00484       0 10                 Put8BitsofData( AuxIndex, ChangeLocation, UpperDatum);
00485       0 10                 end
00486       0 10
00487       0 10               else begin
00488       0 10                 Put8BitsofData( AuxIndex, ChangeLocation, LowerDatum);
00489       0 10                 Put8BitsofData( PortIndex, ChangeLocation, UpperDatum);
```

```
00490      0  9              end; {else begin}
00491      0  8            end; {case SixteenBits}
00492      0  8
00493      0  7            end; {case}
00494      0  6          end; {case C}
```

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

00496       0  7          'A','a': begin {append}
00497       0  7            write( 'How many data points (in decimal) do you want to append: ');
00498       0  7            readln( AppendNumber);
00499       0  7            writeln( 'Enter each data point in hex:');
00500       0  7            write( '  (2 characters for 8 bit ports and ');
00501       0  7            write( '  4 characters for 16 bit ports.)');
00502       0  7            writeln;
00503       0  7            for DataLoop := InputIndex to (InputIndex + AppendNumber - 1)
00504       0  8            do begin
00505       0  8              write( '  Point Number  , DataLoop:3, ': ');
00506       0  8              HexIn( Datum);
00507       0  8              InputIndex := InputIndex + 1;
00508       0  9              case PortWidth of
00509       0  9
00510       0 10              EightBits: begin
00511       0 10                LowerDatum := uand( Datum, Low8Bits);
00512       0 10                Put8BitsofData( PortIndex, DataLoop, LowerDatum);
00513       0  9                end; {case EightBits}
00514       0  9
00515       0 10              SixteenBits: begin
00516       0 10                LowerDatum := uand( Datum, Low8Bits);
00517       0 10                UpperDatum := MaskShiftRight( Datum, High8Bits, Bit8Divisor);
00518       0 11                if (LowerUpperIndicator = LowerPortIndicator) then begin
00519       0 11                  Put8BitsofData( PortIndex, DataLoop, LowerDatum);
00520       0 11                  Put8BitsofData( AuxIndex, DataLoop, UpperDatum);
00521       0 11                  end
00522       0 11
00523       0 11                else begin
00524       0 11                  Put8BitsofData( AuxIndex, DataLoop, LowerDatum);
00525       0 11                  Put8BitsofData( PortIndex, DataLoop, UpperDatum);
00526       0 10                end; {else begin}
00527       0 10                IOPortMap[AuxIndex].InputIndex := IOPortMap[AuxIndex].InputIndex + 1;
00528       0  9              end; {case SixteenBits}
00529       0  9
00530       0  8              end; {case}
00531       0  7            end; {for DataLoop}
00532       0  6          end; {case A}
```

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

00534       0  7          'I','i': begin {insert}
00535       0  7            write( 'Enter the location of the point(s) to be inserted : ');
00536       0  7            readln( InsertLocation);
00537       0  7            write( 'Enter the number of points to insert : ');
00538       0  7            readln( InsertNumber);
00539       0  7            for DataLoop := (InputIndex - 1) downto InsertLocation
00540       0  8            do begin
00541       0  9              case PortWidth of
00542       0  9
00543       0 10              EightBits: begin
00544       0 10                Get8BitsofData( PortIndex, DataLoop, LowerDatum);
00545       0 10                Put8BitsofData( PortIndex, (DataLoop + InsertNumber),
00546       0 10                  LowerDatum);
00547       0  9                end; {case EightBits}
00548       0  9
00549       0  9              SixteenBits:
00550       0 10                if (LowerUpperIndicator = LowerPortIndicator) then begin
00551       0 10                  Get8BitsofData( PortIndex, DataLoop, LowerDatum);
00552       0 10                  Put8BitsofData( PortIndex, (DataLoop + InsertNumber), LowerDatum);
00553       0 10                  Get8BitsofData( AuxIndex, DataLoop, UpperDatum);
00554       0 10                  Put8BitsofData( AuxIndex, (DataLoop + InsertNumber), UpperDatum);
00555       0 10                  end
00556       0 10
00557       0 10                else begin
00558       0 10
00559       0 10                  Get8BitsofData( AuxIndex, DataLoop, LowerDatum);
00560       0 10                  Put8BitsofData( AuxIndex, (DataLoop + InsertNumber),
00561       0 10                    LowerDatum);
00562       0 10                  Get8BitsofData( PortIndex, DataLoop, UpperDatum);
00563       0 10                  Put8BitsofData( PortIndex, (DataLoop + InsertNumber), UpperDatum);
00564       0  9                end; {else begin}
00565       0  9
00566       0  8              end; {case}
00567       0  7            end; {for DataLoop}
```

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

00569       0  7            for DataLoop := InsertLocation to
00570       0  8              (InsertLocation + InsertNumber - 1) do begin
00571       0  8              write( '  New point number ', DataLoop:3, ': ');
00572       0  8              Hexin( Datum);
00573       0  8              InputIndex := InputIndex + 1;
00574       0  9              case PortWidth of
00575       0  9
00576       0 10              EightBits: begin
00577       0 10                LowerDatum := uand( Datum, Low8Bits);
00578       0 10                Put8BitsofData( PortIndex, DataLoop, LowerDatum);
00579       0  9                end; {case EightBits}
00580       0  9
00581       0 10              SixteenBits: begin
00582       0 10                LowerDatum := uand( Datum, Low8Bits);
00583       0 10                UpperDatum := MaskShiftRight( Datum, High8Bits, Bit Divisor);
00584       0 11                if (LowerUpperIndicator = LowerPortIndicator) then begin
00585       0 11                  Put8BitsofData( PortIndex, DataLoop, LowerDatum);
00586       0 11                  Put8BitsofData( AuxIndex, DataLoop, UpperDatum);
00587       0 11                  end
00588       0 11
00589       0 11                else begin
00590       0 11                  Put8BitsofData( AuxIndex, DataLoop, LowerDatum);
00591       0 11                  Put8BitsofData( PortIndex, DataLoop, UpperDatum);
00592       0 10                end; {else begin}
00593       0 10                IOPortMap[AuxIndex].InputIndex := IOPortMap[AuxIndex].InputIndex + 1;
00594       0  9              end; {case SixteenBits}
00595       0  9
00596       0  8              end; {case}
00597       0  7            end; {for DataLoop}
00598       0  6          end; {case I}
```

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

00600       0  7          'D','d': begin {display with index]
00601       0  7
00602       0  8              if (LowerUpperIndicator = LowerPortIndicator) then begin
00603       0  8                write('  Address: ', hex( Address, 4, 4));
00604       0  8                if (PortWidth = SixteenBits) then
00605       0  8                  write(' and  , hex( IOPortMap[ AuxIndex].Address, 4, 4),
00606       0  8                    ',  Width: 16 bits')
00607       0  8                else write(',  Width: 8 bits');
00608       0  8              end
00609       0  8              else begin
00610       0  9                if (PortWidth = SixteenBits) then begin
00611       0  9                  write('  Address: ', hex( IOPortMap[ AuxIndex].Address, 4, 4));
00612       0  9                  write(' and ', hex( Address, 4, 4), ',  Width: 16 bits');
00613       0  8                end;
00614       0  7              end;
00615       0  7              write(  ,  Entries Accessed:  , (DataIndex - 1):1,  , );
00616       0  7              writeln;
00617       0  7              write( '  Number of Entries:  , (InputIndex - 1):1,  ,');
00618       0  7              write(    Direction: ');
00619       0  8              case InOutIndicator of
00620       0  8
00621       0  8              InputIndicator: write( 'Input');
00622       0  8
00623       0  8              OutPutIndicator: write( 'Output');
00624       0  8
00625       0  8              IOIndicator: write( 'Input and Output');
00626       0  8
00627       0  7              end; {case InOutIndicator}
00628       0  7
00629       0  7              writeln;
00630       0  7
00631       0  7              if (DataIndex >= InputIndex) then DisplayIndex := DataIndex
00632       0  7              else DisplayIndex := InputIndex;
00633       0  7
00634       0  8              if (DisplayIndex <> FirstIODataPoint) then begin
00635       0  8                write('  Data: ');
00636       0  9                case PortWidth of
00637       0  9
00638       0 10                EightBits: for DataLoop := FirstIODataPoint to (DisplayIndex - 1) do begin
00639       0 10                  Get8BitsofData( PortIndex, DataLoop, Datum);
00640       0 10                  write( DataLoop:4, ': ', hex( Datum, 2, 2),'  );
00641       0 10                  if ((( DataLoop rem DisplaywIndex8Bits) = 0) and
00642       0 11                    (DataLoop > FirstIODataPoint)) then begin
00643       0 11                      writeln;  write('        ');
```

```
00644       ) 10
00645       0  9                  end;
                                end; {for DataLoop Eight bits}
```

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

00647       0 10                SixteenBits: begin
00648       0 10                  if (LowerUpperIndicator = LowerPortIndicator) then
00649       0 11                    for DataLoop := FirstIODataPoint to (DisplayIndex - 1) do begin
00650       0 11                      Get8BitsofData( AuxIndex, DataLoop, Datum);
00651       0 11                      write( DataLoop:4, ': ', hex( Datum, 2, 2));
00652       0 11                      Get8BitsofData( PortIndex, DataLoop, Datum);
00653       0 11                      write( hex( Datum, 2, 2));
00654       0 11                      if ((( DataLoop rem DisplaywIndex16Bits) = 0)
00655       0 12                        and (DataLoop > FirstIODataPoint)) then begin
00656       0 12                          writeln; write('        ');
00657       0 11                        end;
00658       0 11                    end {for DataLoop}
00659       0 11
00660       0 11                    else for DataLoop := FirstIODataPoint to (DisplayIndex - 1) do begin
00661       0 11                      Get8BitsofData( PortIndex, DataLoop, Datum);
00662       0 11                      write( DataLoop:4, ': ', hex( Datum, 2, 2));
00663       0 11                      Get8BitsofData( AuxIndex, DataLoop, Datum);
00664       0 11                      write( hex( Datum, 2, 2));
00665       0 11                      if ((( DataLoop rem DisplaywIndex16Bits) = 0)
00666       0 12                        and (DataLoop > FirstIODataPoint)) then begin
00667       0 12                          writeln; write('        ');
00668       0 11                        end;
00669       0 11                    end {for DataLoop}
00670       0 11
00671       0  9                  end; {case SixteenBits}
00672       0  8                end; {case PortWidth}
00673       0  8
00674       0  7              end; {if DisplayIndex <> FirstIODataPoint}
00675       0  7              writeln; writeln;
00676       0  7
00677       0  6          end; {case Display with Index}
```

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

00679       0  7           'R','r': begin {remove}
00680       0  7             write( 'Enter the location of the point(s) to be removed : ');
00681       0  7             readln( RemoveLocation);
00682       0  7             write( 'Enter the number of points to remove :  );
00683       0  7             readln( RemoveNumber);
00684       0  7             for DataLoop := (RemoveLocation + RemoveNumber) to InputIndex
00685       0  8             do begin
00686       0  9               case PortWidth of
00687       0  9
00688       0 10               EightBits: begin
00689       0 10                 Get8BitsofData( PortIndex, DataLoop, LowerDatum);
00690       0 10                 Put8BitsofData( PortIndex, RemoveLocation, LowerDatum);
00691       0  9                 end; {case EightBits}
00692       0  9
00693       0  9               SixteenBits:
00694       0 10                 if (LowerUpperIndicator = LowerPortIndicator) then begin
00695       0 10                   Get8BitsofData( PortIndex, DataLoop, LowerDatum);
00696       0 10                   Put8BitsofData( PortIndex, RemoveLocation, LowerDatum);
00697       0 10                   Get8BitsofData( AuxIndex, DataLoop, UpperDatum);
00698       0 10                   Put8BitsofData( AuxIndex, RemoveLocation, UpperDatum);
00699       0 10                   end
00700       0 10
00701       0 10                 else begin
00702       0 10
00703       0 10                   Get8BitsofData( AuxIndex, DataLoop, LowerDatum);
00704       0 10                   Put8BitsofData( AuxIndex, RemoveLocation, LowerDatum);
00705       0 10                   Get8BitsofData( PortIndex, DataLoop, UpperDatum);
00706       0 10                   Put8BitsofData( PortIndex, RemoveLocation, UpperDatum);
00707       0  9                 end; {else begin}
00708       0  9
00709       0  8               end; {case}
00710       0  8               RemoveLocation := RemoveLocation + 1;
00711       0  7             end; {for DataLoop}
00712       0  7             InputIndex := InputIndex - RemoveNumber;
00713       0  7             if (PortWidth = SixteenBits) then
00714       0  7               IOPortMap[AuxIndex].InputIndex := IOPortMap[AuxIndex].InputIndex - RemoveNumber;
00715       0  6           end; {case R}
00716       0  6
00717       0  6
00718       0  6           'F','f': Finished := true;  {finished}
00719       0  5           end; {case ModifyCommand}
00720       0  5         end  {while not Finished}
00721       0  5
00722       0  4     else writeln( 'Port address not found:  ,hex( PortAddress, 4, 4));
```

```
00723      0  3     end; {case M}
```

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

00725       0  4     'D','d': begin {display}
00726       0  4       writeln;  writeln;
00727       0  4       write( 'Enter port address :  );
00728       0  4       HexIn( PortAddress);
00729       0  4       SearchForPortAddress( PortAddress, IOPortMap, PortAddressPresent,   PortIndex);
00730       0  5       if (PortAddressPresent) then with IOPortMap[PortIndex] do begin
00731       0  6            if (LowerUpperIndicator = LowerPortIndicator) then begin
00732       0  6              writeln('Index: ', PortIndex:1);
00733       0  6              write('  Address: ', hex( Address, 4, 4));
00734       0  6              if (PortWidth = SixteenBits) then
00735       0  6                write(' and  , hex( IOPortMap[ AuxIndex].Address, 4, 4),
00736       0  6                  ',  Width: 16 bits')
00737       0  6              else write(',  Width: 8 bits');
00738       0  6            end
00739       0  6            else begin
00740       0  7              if (PortWidth = SixteenBits) then begin
00741       0  7                write('  Address: ', hex( IOPortMap[ AuxIndex].Address, 4, 4));
00742       0  7                write(' and ', hex( Address, 4, 4), ',  Width: 16 bits');
00743       0  6              end;
00744       0  5            end;
00745       0  5            write(  ,  Entries Accessed:  , (DataIndex - 1):1, ',');
00746       0  5            writeln;
00747       0  5            write( '  Number of Entries:  , (InputIndex - 1):1,  , );
00748       0  5            write(    Direction: ');
00749       0  6            case InOutIndicator of
00750       0  6
00751       0  6            InputIndicator: write( 'Input');
00752       0  6
00753       0  6            OutPutIndicator: write( 'Output');
00754       0  6
00755       0  6            IOIndicator: write( 'Input and Output');
00756       0  6
00757       0  5            end; {case InOutIndicator}
00758       0  5
00759       0  5            writeln;
00760       0  5
00761       0  5            if (DataIndex >= InputIndex) then DisplayIndex := DataIndex
00762       0  5            else DisplayIndex := InputIndex;
00763       0  5
00764       0  6            if (DisplayIndex <> FirstIODataPoint) then begin
00765       0  6              write('  Data: ');
00766       0  7              case PortWidth of
00767       0  7
00768       0  8                EightBits: for DataLoop := FirstIODataPoint to (DisplayIndex - 1) do begin
```

```
00769     0  8              Get8BitsofData( PortIndex, DataLoop, Datum);
00770     0  8              write( hex( Datum, 2, 2),' ');
00771     0  8              if ((( DataLoop rem DisplayEntries8Bits) = 0) and
00772     0  9                (DataLoop > FirstIODataPoint)) then begin
00773     0  9                  writeln;  write('        ');
00774     0  8                end;
00775     0  7            end; {for DataLoop Eight bits}
```

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

00777       0  8              SixteenBits: begin
00778       0  8                if (LowerUpperIndicator = LowerPortIndicator) then
00779       0  9                  for DataLoop := FirstIODataPoint to (DisplayIndex - 1) do begin
00780       0  9                    Get8BitsofData( AuxIndex, DataLoop, Datum);
00781       0  9                    write( hex( Datum, 2, 2));
00782       0  9                    Get8BitsofData( PortIndex, DataLoop, Datum);
00783       0  9                    write( hex( Datum, 2, 2),' ');
00784       0  9                    if ((( DataLoop rem DisplayEntries16Bits) = 0)
00785       0 10                      and (DataLoop > FirstIODataPoint)) then begin
00786       0 10                      writeln; write('        ');
00787       0  9                    end;
00788       0  9                  end {for DataLoop}
00789       0  9
00790       0  9                  else for DataLoop := FirstIODataPoint to (DisplayIndex - 1) do begin
00791       0  9                    Get8BitsofData( PortIndex, DataLoop, Datum);
00792       0  9                    write( hex( Datum, 2, 2));
00793       0  9                    Get8BitsofData( AuxIndex, DataLoop, Datum);
00794       0  9                    write( hex( Datum, 2, 2), ' ');
00795       0  9                    if ((( DataLoop rem DisplayEntries16Bits) = 0)
00796       0 10                      and (DataLoop > FirstIODataPoint)) then begin
00797       0 10                      writeln; write('        ');
00798       0  9                    end;
00799       0  8                  end; {for DataLoop}
00800       0  8
00801       0  7                end; {case SixteenBits}
00802       0  6              end; {case PortWidth}
00803       0  6
00804       0  5            end; {if DisplayIndex <> FirstIODataPoint}
00805       0  5            writeln; writeln;
00806       0  5          end {with IOPortMap[PortIndex]}
00807       0  4      else writeln( 'Port address not found:  ',hex( PortAddress, 4, 4));
00808       0  3    end; {case D}
```

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

00810       0  4     'P','p': begin {print}
00811       0  4       writeln;  writeln;
00812       0  4       write( 'Enter name of file to contain listing of data : ');
00813       0  4       readln( ListFilename);
00814       0  4       open( ListFile, ListFilename, new);
00815       0  4       rewrite( ListFile);
00816       0  4
00817       0  4       for PortIndex := FirstIOPortIndex to MaxNumberofIOPorts do
00818       0  4         with IOPortMap[PortIndex] do
00819       0  4           if ((Address <> InvalidAddress) and
00820       0  5           (LowerUpperIndicator = LowerPortIndicator)) then begin
00821       0  5             writeln( ListFile, 'Index: ', PortIndex:1);
00822       0  5             write( ListFile, '  Address:  , hex( Address, 4, 4));
00823       0  5             if (PortWidth = SixteenBits) then write( ListFile, ' and  ,
00824       0  5               hex( IOPortMap[AuxIndex].Address, 4, 4),
00825       0  5               ',  Width: 16 bits')
00826       0  5             else write( ListFile,  ,  Width: 8 bits');
00827       0  5             write( ListFile, ',  Entries Accessed: ', (DataIndex - 1):1,  ,');
00828       0  5             writeln( ListFile);
00829       0  5             write( ListFile, '  Number of Entries:  , (InputIndex - 1):1,  , );
00830       0  5             write( ListFile,    Direction: ');
00831       0  5
00832       0  6             case InOutIndicator of
00833       0  6
00834       0  6             InputIndicator: write( ListFile, 'Input');
00835       0  6
00836       0  6             OutPutIndicator: write( ListFile, 'Output');
00837       0  6
00838       0  6             IOIndicator: write( ListFile, 'Input and Output');
00839       0  6
00840       0  5             end; {case InOutIndicator}
00841       0  5
00842       0  5             writeln( ListFile);
00843       0  5
00844       0  5             if (DataIndex >= InputIndex) then PrintIndex := DataIndex
00845       0  5             else PrintIndex := InputIndex;
00846       0  5
00847       0  6             if (PrintIndex <> FirstIODataPoint) then begin
00848       0  6               write( ListFile, '  Data: ');
00849       0  7               case PortWidth of
00850       0  7
00851       0  8               EightBits: for DataLoop := FirstIODataPoint to (PrintIndex - 1) do begin
00852       0  8                 Get8BitsofData( PortIndex, DataLoop, Datum);
00853       0  8                 write( ListFile, hex( Datum, 2, 2),' ');
```

```
00854     0  8              if ((( DataLoop rem PrintEntries8Bits) = 0) and
00855     0  9                (DataLoop > FirstIODataPoint)) then begin
00856     0  9                  writeln( ListFile);  write( ListFile,'        ');
00857     0  8                end;
00858     0  7              end; {for DataLoop Eight bits}
```

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

00860       0  8              SixteenBits: begin
00861       0  8                if (LowerUpperIndicator = LowerPortIndicator) then
00862       0  9                  for DataLoop := FirstIODataPoint to (PrintIndex - 1) do begin
00863       0  9                    Get8BitsofData( AuxIndex, DataLoop, Datum);
00864       0  9                    write( ListFile,  hex( Datum, 2, 2));
00865       0  9                    Get8BitsofData( PortIndex, DataLoop, Datum);
00866       0  9                    write( ListFile,  hex( Datum, 2, 2), ' ');
00867       0  9                    if ((( DataLoop rem PrintEntries16Bits) = 0) and
00868       0 10                      (DataLoop > FirstIODataPoint)) then begin
00869       0 10                        writeln( ListFile);  write( ListFile,'        ');
00870       0  9                    end;
00871       0  8                  end; {for DataLoop}
00872       0  7                end; {case SixteenBits}
00873       0  6              end; {case PortWidth}
00874       0  6
00875       0  5            end; {if PrintIndex <> FirstIODataPoint}
00876       0  5            writeln( ListFile);  writeln( ListFile);
00877       0  5
00878       0  4        end; {if (Address <> InvalidAddress) then begin}
00879       0  4      close( ListFile);
00880       0  3    end; {case P}
00881       0  3
00882       0  4        'W','w': begin {write}
00883       0  4      open( IOMapFile, IOMapFilename, old);
00884       0  4      rewrite( IOMapFile);
00885       0  4      open( IODataFile, IODataFilename, old);
00886       0  4      rewrite( IODataFile);
00887       0  4
00888       0  4      for PortIndex := FirstIOPortIndex to MaxNumberofIOPorts do
00889       0  5        begin
00890       0  5          write( IOMapFile, IOPortMap[PortIndex]);
00891       0  5          for DataLoop := FirstIODataPoint to MaxNumberofIOWords do
00892       0  5              write( IODataFile, IOData[PortIndex,DataLoop]);
00893       0  4        end; {for PortIndex do begin}
00894       0  4      close( IOMapFile);
00895       0  4      close( IODataFile);
00896       0  4
00897       0  3    end; {case W}
00898       0  3
00899       0  4        'Q','q': begin {exit without writing}
00900       0  4      writeln; writeln;
00901       0  4      writeln( 'This will exit without saving the data or map files.');
00902       0  4      write(   'Are you sure you want to do this [Y/N] ? ');
00903       0  4      readln( QuitResponse);
```

```
00904      0  4       if ((QuitResponse = 'Y') or (QuitResponse = 'y')) then Done := true;  {exit}
00905      0  3     end; {case Q}
00906      0  3
00907      0  3
```

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

00909       0  4          'C','c': begin {create disk file]
00910       0  4        writeln;  writeln;
00911       0  4        write( 'Enter name of file containing data:  );
00912       0  4        readln( IODataFilename);
00913       0  4        writeln;
00914       0  4        open( IODataFile, IODataFilename, new);
00915       0  4        close( IODataFile);
00916       0  3      end; {case C]
00917       0  3
00918       0  4          'R','r': begin {read from disk]
00919       0  4        writeln;  writeln;
00920       0  4        write( 'Enter name of file containing data:  );
00921       0  4        readln( IODataFilename);
00922       0  4        writeln;
00923       0  4        Exists := FileExists( %descr IODataFileName);
00924       0  5        if (Exists = true) then begin
00925       0  5            open( IODataFile, IODataFilename, old);
00926       0  5            reset( IODataFile);
00927       0  5
00928       0  5            for PortIndex := FirstIOPortIndex to MaxNumberofIOPorts do
00929       0  5                for DataLoop := FirstIODataPoint to MaxNumberofIOWords do
00930       0  5                    read( IODataFile, IOData[PortIndex,DataLoop]);
00931       0  5            close( IODataFile);
00932       0  5        end
00933       0  4        else writeln( 'File  , IODataFilename,   does not exist');
00934       0  3      end; {case R]
00935       0  3
00936       0  3
00937       0  4          'E','e': begin
00938       0  4        Done := true;  {exit]
00939       0  4        open( IOMapFile, IOMapFilename, old);
00940       0  4        rewrite( IOMapFile);
00941       0  4        open( IODataFile, IODataFilename, old);
00942       0  4        rewrite( IODataFile);
00943       0  4
00944       0  4        for PortIndex := FirstIOPortIndex to MaxNumberofIOPorts do
00945       0  5          begin
00946       0  5            write( IOMapFile, IOPortMap[PortIndex]);
00947       0  5            for DataLoop := FirstIODataPoint to MaxNumberofIOWords do
00948       0  5                write( IODataFile, IOData[PortIndex,DataLoop]);
00949       0  4          end; {for PortIndex do begin]
00950       0  4        close( IOMapFile);
00951       0  4        close( IODataFile);
00952       0  4
```

```
00953     0  3     end; {case E}
00954     0  3
00955     0  3           otherwise writeln('Bad input.  Try again');
00956     0  3
00957     0  2           end; {case}
00958     0  1     end; {while not done}
00959     0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	12037	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	2048	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

PAS/LIS REVIEWIODATA

/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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]REVIEWIODATA.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]REVIEWIODATA.OBJ;35
/NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	409	00:00.5	00:03.2
Source Analysis	486	00:03.4	00:16.0
Source Listing	16	00:01.5	00:08.9
Tree Construction	446	00:01.5	00:04.8
Flow Analysis	223	00:01.2	00:02.0
Value Propagation	41	00:00.2	00:00.3
Profit Analysis	234	00:00.9	00:01.9
Context Analysis	273	00:08.1	00:18.4
Name Packing	9	00:00.6	00:01.8
Code Selection	187	00:01.8	00:05.9
Final	100	00:02.9	00:09.2
TOTAL	2430	00:22.6	01:12.5

COMPILATION STATISTICS

CPU Time: 00:22.6 (2551 Lines/Minute)

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

00001    C  0  0 { Title: RMDecoder
00002    C  0  0
00003    C  0  0   Purpose: Decode register and mode fields of opcode
00004    C  0  0
00005    C  0  0   Author: William A. Chapman      Date: January 12, 1986
00006    C  0  0
00007    C  0  0   Inputs: Mode Value - integer - indicates how to interpret displacement
00008    C  0  0           RMValue - integer - indicates which addressing mode to use
00009    C  0  0
00010    C  0  0   Outputs: Address Mode - register or memory flag
00011    C  0  0           Address Value - pointer to memory, or register indicator
00012    C  0  0
00013    C  0  0   Procedures Invoked: FetchCode
00014    C  0  0           ComputeEA
00015    C  0  0           (SegmentSelector)
00016    C  0  0
00017    C  0  0   Functions Invoked: SignExtend7
00018    C  0  0           SignExtend15
00019    C  0  0 }
00020       0  0 module RMDecoder(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;
```

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

00044       0  0     %include  [-]RegId.defn/nolist'
00108    C  0  0 {    SSId = %B'10';                stack segment id]
00109    C  0  0 {    DSId = %B'11';                data segment id]
00110       0  0
00111       0  0     RegisterFlag = 'R';
00112       0  0     MemoryFlag = 'M';
00113       0  0     Undefined = -1;
00114       0  0
00115       0  0     %include '[-]StopCode.defn/list'
00116 I     0  0     Breakpoint              = 'B';
00117 I     0  0     Call            = 'C';
00118 I     0  0     ControlD              = 'D';
00119 I     0  0     BadOpcode               = 'E';
00120 I     0  0     BadCEAModeValue         = 'F';
00121 I     0  0     BadRMDModeValue         = 'G';
00122 I     0  0     Halt            = 'H';
00123 I     0  0     BadRegisterID = 'I';
00124 I     0  0     BadOpcodeKey  = 'K';
00125 I     0  0     BadMemoryAddress        = 'M';
00126 I     0  0     Normal                  = 'N';
00127 I     0  0     BadPortAddress          = 'P';
00128 I     0  0     BadOpcodeClass          = 'Q';
00129 I     0  0     Return                  = 'R';
00130 I     0  0     BadCheckSum             = 'S';
00131 I     0  0     BadOperandType          = 'T';
00132 I     0  0     BadMemoryWidth          = 'W';
00133 I     0  0     BadOpcodeExtension      = 'X';
00134 I     0  0     NoMemoryAccess        = 'Z';
00135       0  0
00136       0  0
00137       0  0
00138       0  0
00139       0  0 var
00140       0  0     Displacement: integer;
00141       0  0     LoDisplacement: unsigned;
00142       0  0     HighDisplacement: unsigned;
00143       0  0
00144       1  0 [global] procedure RMDecoder(var AddressMode: char;
00145       1  0           var AddressValue: unsigned;
00146       1  0           var SegmentId: integer;
00147       1  0           ModeValue: integer;
00148       1  0           RMValue: integer;
00149       1  0           var RMDStopCode: char);
00150       1  0
```

```
00151      2  0 procedure FetchCode(var FCValue: unsigned;
00152      1  0           var FCStopCode: char); external;
00153      1  0
00154      2  0 procedure ComputeEA(CEADisplacement: integer;
00155      2  0           CEAMode: integer;
00156      2  0           var CEAAddressValue: unsigned;
00157      1  0           var CEAStopCode: char); external;
00158      1  0
00159      1  0 function SignExtend7(SE7Value: unsigned): integer; external;
00160      1  0
00161      1  0 function SignExtend15(SE15Value: unsigned): integer; external;
```

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

00162       1  0
00163       1  0
00164       1  0
00165       2  0 function SegmentSelector(Key: integer): integer;
00166       2  0
00167       2  1 begin
00168       2  2     case Key of
00169       2  2
00170       2  2     0,1,4,5,7: SegmentSelector := DSId;          {these use the data segment}
00171       2  2
00172       2  2     2,3,6: SegmentSelector := SSId;              {these use the stack segment}
00173       2  2
00174       2  1     end;
00175       1  0 end;
```

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

00177       1  1 begin
00178       1  1
00179       1  2     if ((ModeValue = SimpleMode) and (RMValue = SimpleRM)) then begin
00180       1  2   AddressMode := MemoryFlag;              {this is displacement only mode}
00181       1  2   FetchCode( LoDisplacement, RMDStopCode);
00182       1  2   FetchCode( HighDisplacement, RMDStopCode);
00183       1  2   HighDisplacement := uand((HighDisplacement * Bit8Multiplier),High8Bits);
00184       1  2   AddressValue := uor(LoDisplacement,HighDisplacement);
00185       1  2   SegmentId := DSId;
00186       1  2     end {then begin} {displacement only address mode}
00187       1  2
00188       1  2     else case ModeValue of
00189       1  2
00190       1  3   0: begin
00191       1  3        AddressMode := MemoryFlag;
00192       1  3        Displacement := 0;
00193       1  3        ComputeEA( Displacement, RMValue, AddressValue, RMDStopCode);
00194       1  3        SegmentId := SegmentSelector(RMValue);
00195       1  2      end;
00196       1  2
00197       1  3   1: begin
00198       1  3        AddressMode := MemoryFlag;
00199       1  3        FetchCode( LoDisplacement, RMDStopCode);
00200       1  3        Displacement := int(SignExtend7(LoDisplacement));
00201       1  3        ComputeEA( Displacement, RMValue, AddressValue, RMDStopCode);
00202       1  3        SegmentId := SegmentSelector(RMValue);
00203       1  2      end;
00204       1  2
00205       1  3   2: begin
00206       1  3        AddressMode := MemoryFlag;
00207       1  3        FetchCode( LoDisplacement, RMDStopCode);
00208       1  3        FetchCode( HighDisplacement, RMDStopCode);
00209       1  3        HighDisplacement :=
00210       1  3            uand((HighDisplacement * Bit8Multiplier),High8Bits);
00211       1  3        Displacement := int(SignExtend15(
00212       1  3            uor(LoDisplacement,HighDisplacement)));
00213       1  3        ComputeEA( Displacement, RMValue, AddressValue, RMDStopCode);
00214       1  3        SegmentId := SegmentSelector(RMValue);
00215       1  2      end;
00216       1  2
00217       1  3   3: begin
00218       1  3        AddressMode := RegisterFlag;
00219       1  3        AddressValue := uint(RMValue);
00220       1  3        SegmentId := Undefined;
```

```
00221      1  2     end;
00222      1  2
00223      1  2   otherwise RMDStopCode := BadRMDModeValue;
00224      1  2
00225      1  1   end; {else case ModeValue}
00226      1  1
00227      0  0 end; {procedure}
00228      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	520	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 RMDECODER.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]RMDECODER.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]RMDECODER.OBJ;1
/NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	411	00:00.5	00:02.2
Source Analysis	278	00:00.8	00:03.4
Source Listing	16	00:00.4	00:02.6
Tree Construction	110	00:00.1	00:00.5
Flow Analysis	43	00:00.1	00:00.1
Value Propagation	11	00:00.0	00:00.0
Profit Analysis	39	00:00.1	00:00.1
Context Analysis	180	00:00.7	00:01.6
Name Packing	6	00:00.0	00:00.0
Code Selection	80	00:00.2	00:00.3
Final	60	00:00.2	00:00.9
TOTAL	1238	00:03.0	00:11.6

COMPILATION STATISTICS

CPU Time: 00:03.0 (4500 Lines/Minute)

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

00001    C  0  0 { Title: SearchForBreakpoint
00002    C  0  0
00003    C  0  0   Purpose: Examine the breakpoint table for a breakpoint at this
00004    C  0  0            CS:( IP - 1)
00005    C  0  0
00006    C  0  0   Author: William A. Chapman      Date: October 5, 1986
00007    C  0  0
00008    C  0  0   Inputs: BreakPointTable
00009    C  0  0           CS
00010    C  0  0           IP
00011    C  0  0
00012    C  0  0   Outputs: BreakPointEntry - entry point into BreakPointTable
00013    C  0  0                              if entry found
00014    C  0  0           BreakPointFound - flag is set if valid entry found
00015    C  0  0
00016    C  0  0   Procedures Invoked: none
00017    C  0  0 }
00018       0  0 module SearchForBreakpoint( 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';
00123       0  0
00124       0  0      %include  [-]RegID.defn/nolist'
00188   C   0  0 (    CSId = %B'01';
00189   C   0  0      CSWidth = 2;)
```

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

00191       0  0 type
00192       0  0     %include  [-]Type.defn/list'
00193 I     0  0     ZeroOne = -1..1;                        {defines integer with range
00194 I C   0  0                                              of zero to one with an invalid
00195 I C   0  0                                              state of -1}
00196 I     0  0
00197 I     0  0     OpcodeLUTEntry = record                 {defines entries in opcode LUT}
00198 I     0  0   OpcodeClass: char;                        {class D, A, L, S, C, P}
00199 I     0  0   OpcodeKey: integer;                       {key to interpret opcode from
00200 I C   0  0                                              0 to 7}
00201 I     0  0   DirectionBitPresent: boolean;
00202 I     0  0   WidthBitPresent: boolean
00203 I     0  0     end;
00204 I     0  0
00205 I     0  0     OpcodeType = record
00206 I     0  0   Direction: ZeroOne;                      {to or from CPU}
00207 I     0  0   Full: unsigned;                          {full opcode}
00208 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00209 I     0  0     end;
00210 I     0  0
00211 I     0  0     RegisterType = record
00212 I     0  0   Id: integer;                             {identifier}
00213 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00214 I     0  0     end;
00215 I     0  0
00216 I     0  0     EffectiveAddressType = record
00217 I     0  0   Mode: char;                              {register or memory}
00218 I     0  0   Width: ZeroOne;                         {8 or 16 bits}
00219 I     0  0   Address: unsigned;                      {memory address or
00220 I C   0  0                                              register designation}
00221 I     0  0   Segment: integer                        {segment register to use}
00222 I     0  0     end;
00223 I     0  0
00224 I     0  0     NameType = packed array[1..10] of char;
00225 I     0  0
00226 I     0  0     Characters = set of     ..'~';
00227 I     0  0     LettersAndNumbers = set of '0'..'z';
00228 I     0  0     Letters = set of 'A'..'z';
00229 I     0  0     Numbers = set of '0'..'9';
00230 I     0  0
00231 I     0  0     FileName = packed array [1..FilenameLength] of char;
```

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

00233       0  0     %include '[-]DebugType.defn/list'
00234 I     0  0     NameString = packed array[1..ShortStringLength] of char;
00235 I     0  0
00236 I     0  0     KeywordCharacteristics = record
00237 I     0  0   Name: NameString;
00238 I     0  0   Class: char;
00239 I     0  0   Width: ZeroOne;
00240 I     0  0   ID: integer;
00241 I     0  0   Value: unsigned
00242 I     0  0     end;
00243 I     0  0
00244 I     0  0     InputPointerRange = integer;
00245 I     0  0     InputLine = packed array [1..InputLineLength] of char;
00246 I     0  0
00247 I     0  0     ShortString = packed array [1..ShortStringLength] of char;
00248 I     0  0
00249 I     0  0     DataArrayType = array [StartOfData..DataArraySize] of unsigned;
00250 I     0  0
00251 I     0  0     BreakRecord = record
00252 I     0  0   Encountered: boolean;
00253 I     0  0   Activated: boolean;
00254 I     0  0   Segment: unsigned;
00255 I     0  0   Address: unsigned;
00256 I     0  0   PhysicalAddress: unsigned;
00257 I     0  0   Code: unsigned
00258 I     0  0     end;
00259 I     0  0
00260 I     0  0     BreakPointTableType = array [FirstBreakpoint..LastBreakpoint] of BreakRecord;
00261       0  0
00262       0  0
00263       0  0
00264       0  0 var
00265       0  0     IP: [external] unsigned;
```

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

00267      1  0 [global] procedure SearchForBreakpoint(
00268      1  0           var SFBBreakPointTable: BreakPointTableType;
00269      1  0           var SFBBreakPointEntry: integer;
00270      1  0           var SFBBreakPointFound: boolean;
00271      1  0           SFBCS: unsigned;
00272      1  0           SFBIP: unsigned);
00273      1  0
00274      1  0 var
00275      1  0     SFBDone: boolean;
00276      1  0     SFBAddress: unsigned;
00277      1  0
00278      1  1 begin
00279      1  1     SFBAddress := ((SFBCS * SegmentOffset) + SFBIP);
00280      1  1
00281      1  1     SFBDone := false;
00282      1  1     SFBBreakPointEntry := FirstBreakPoint - 1;
00283      1  1     SFBBreakPointFound := false;
00284      1  2     while (not SFBDone) do begin
00285      1  2   SFBBreakPointEntry := SFBBreakPointEntry + 1;
00286      1  2   if (SFBBreakPointTable[ SFBBreakPointEntry].PhysicalAddress =
00287      1  3       SFBAddress) then begin
00288      1  3       SFBDone := true;
00289      1  3       SFBBreakPointFound := true;
00290      1  2   end;
00291      1  2   if (SFBBreakPointEntry = LastBreakPoint) then SFBDone := true;
00292      1  1     end; {while not SFBDone}
00293      1  1
00294      0  0 end;
00295      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	85	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
PAS/LIS SEARCHFORBREAKPOINT.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]SEARCHFORBREAKPOINT.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]SEARCHFORBREAKPOINT.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	00:00.4	00:04.2
Source Analysis	240	00:00.8	00:04.2
Source Listing	18	00:00.5	00:02.7
Tree Construction	86	00:00.1	00:00.3
Flow Analysis	38	00:00.0	00:00.1
Value Propagation	10	00:00.0	00:00.0
Profit Analysis	43	00:00.0	00:00.2
Context Analysis	204	00:00.5	00:00.8
Name Packing	7	00:00.0	00:00.0
Code Selection	81	00:00.1	00:00.1
Final	72	00:00.1	00:01.1
TOTAL	1210	00:02.6	00:13.8

COMPILATION STATISTICS

```
CPU Time:       00:02.6         (6887 Lines/Minute)
Elapsed Time:   00:13.8
```

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

00001    C  0  0 { Title: Store Data module
00002    C  0  0
00003    C  0  0   Purpose: Store the value specified in the store "Memory"
00004    C  0  0           or "Registers Pointers" as specified.
00005    C  0  0
00006    C  0  0   Author: William A. Chapman      Date: December 28, 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           Value - data to be stored
00014    C  0  0
00015    C  0  0   Outputs: The 8 or 16 bit value of the variable is stored in the
00016    C  0  0           store "Memory" or "Registers Pointers" as required.
00017    C  0  0
00018    C  0  0   Procedures Invoked: FetchRegPtr
00019    C  0  0           StoreRegPtr
00020    C  0  0           StoreMemory
00021    C  0  0
00022    C  0  0   Functions Invoked: MaskShiftRight
00023    C  0  0 }
00024       0  0 module StoreData(output);
00025       0  0
00026       0  0 const
00027       0  0     %include  [-]Const.defn/list'
00028 I     0  0     FirstOpcodeValue = 0;
00029 I     0  0     LastOpcodeValue = 255;
00030 I     0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00031 I     0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00032 I     0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00033 I     0  0     Bit8Multiplier = %X'100';
00034 I     0  0     Bit8Divisor    = %X'100';
00035 I     0  0     WordMultiplier = %X'100';
00036 I     0  0
00037 I     0  0     EightBits = 0;                      {width for 8 bits}
00038 I     0  0     SixteenBits = 1;                    {width for 16 bits}
00039 I     0  0
00040 I     0  0     SimpleMode = %B'0';
00041 I     0  0     SimpleRM = %B'110';
00042 I     0  0
00043 I     0  0     LowMemoryLimit = 0;                 {low limit on memory array}
00044 I     0  0     HighMemoryLimit = 2048;             {high limit on memory array}
00045 I     0  0
```

```
00046 I     0  0     FilenameLength = 40;
00047       0  0
00048       0  0     MemoryPointer = 2;                    {width variable to access
00049    C  0  0                                        segment registers}
00050       0  0
00051       0  0     %include  [-]StopCode.defn/list'
00052 I     0  0     Breakpoint            = 'B';
00053 I     0  0     Call          = 'C';
00054 I     0  0     ControlD              = 'D';
00055 I     0  0     BadOpcode             = 'E';
```

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

00056 I     0  0      BadCEAModeValue       = 'F';
00057 I     0  0      BadRMDModeValue       = 'G';
00058 I     0  0      Halt            = 'H';
00059 I     0  0      BadRegisterID = 'I';
00060 I     0  0      BadOpcodeKey  = 'K';
00061 I     0  0      BadMemoryAddress      = 'M';
00062 I     0  0      Normal                = 'N';
00063 I     0  0      BadPortAddress        = 'P';
00064 I     0  0      BadOpcodeClass        = 'Q';
00065 I     0  0      Return                = 'R';
00066 I     0  0      BadCheckSum           = 'S';
00067 I     0  0      BadOperandType        = 'T';
00068 I     0  0      BadMemoryWidth        = 'W';
00069 I     0  0      BadOpcodeExtension    = 'X';
00070 I     0  0      NoMemoryAccess      = 'Z';
00071       0  0
00072       0  0
00073       0  0
00074       0  0
00075       0  0 var
00076       0  0      LowByte: unsigned;                {low 8 bits of 16}
00077       0  0      HighByte: unsigned;               {high 8 bits of 16}
00078       0  0      Segment: unsigned;                {value of the desired segment register}
```

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

00080       1  0 [global] procedure StoreData(SDType: char;
00081       1  0           SDWidth: integer;
00082       1  0           SDAddress: unsigned;
00083       1  0           SDSegment: integer;
00084       1  0           SDValue: unsigned;
00085       1  0           var SDStopCode: char);
00086       1  0
00087       2  0 procedure FetchRegPtr(FRPWidth: integer;
00088       2  0           FRPDesignator:integer;
00089       2  0           var FRPValue: unsigned;
00090       1  0           var FRPStopCode: char); external;
00091       1  0
00092       2  0 procedure StoreRegPtr(SRPWidth: integer;
00093       2  0           SRPDesignator:integer;
00094       2  0           SRPValue: unsigned;
00095       1  0           var SRPStopCode: char); external;
00096       1  0
00097       2  0 procedure StoreMemory(SMAddress: unsigned;
00098       2  0           SMSegment: unsigned;
00099       2  0           SMValue: unsigned;
00100       1  0           var SMStopCode: char); external;
00101       1  0
00102       2  0 function MaskShiftRight(MSRValue: unsigned;
00103       2  0           MSRMask: unsigned;
00104       1  0           MSRDivisor: integer): unsigned; external;
```

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

00106       1  1 begin
00107       1  2     case SDType of        {sort Register from Memory accesses}
00108       1  2
00109       1  2   'r','R': StoreRegPtr( SDWidth, int( SDAddress), SDValue, SDStopCode);
00110       1  2
00111       1  3   'm','M': begin
00112       1  3       FetchRegPtr( MemoryPointer, SDSegment, Segment, SDStopCode);
00113    C  1  3                   {get value of desired segment register}
00114       1  4       case SDWidth of     {sort into 8 or 16 bits}
00115       1  4
00116       1  4           0: StoreMemory( SDAddress, Segment, SDValue, SDStopCode);
00117    C  1  4                   {store 8 bits directly}
00118       1  5           1: begin
00119       1  5                LowByte := uand(SDValue,Low8Bits);  {mask off unwanted bits}
00120       1  5                HighByte := MaskShiftRight(SDValue ,High8Bits, Bit8Divisor);
00121    C  1  5                    {shift high 8 bits down and mask off unwanted bits}
00122       1  5                StoreMemory( SDAddress, Segment, LowByte, SDStopCode); {store low byte}
00123       1  5                StoreMemory( (SDAddress + 1), Segment, HighByte, SDStopCode);
00124    C  1  5                    {store high byte}
00125       1  4              end; {case 1}
00126       1  4
00127       1  4           otherwise SDStopCode := BadMemoryWidth;
00128       1  4
00129       1  3       end; {case SDWidth}
00130       1  2   end; {case "M"}
00131       1  2
00132       1  2   otherwise SDStopCode := BadOperandType;
00133       1  2
00134       1  1     end; {case SDType}
00135       0  0 end;
00136       0  0 end. {module}
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	334	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 STOREDATA.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]STOREDATA.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]STOREDATA.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	00:02.1
Source Analysis	221	00:00.5	00:01.9
Source Listing	14	00:00.3	00:00.9
Tree Construction	97	00:00.1	00:00.1
Flow Analysis	53	00:00.1	00:00.2
Value Propagation	9	00:00.0	00:00.0
Profit Analysis	43	00:00.1	00:00.1
Context Analysis	141	00:00.4	00:01.0
Name Packing	6	00:00.0	00:00.0
Code Selection	72	00:00.1	00:00.5
Final	64	00:00.2	00:00.5
TOTAL	1128	00:02.2	00:07.3

COMPILATION STATISTICS

CPU Time: 00:02.2 (3795 Lines/Minute)

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

00001    C  0  0 { Title: Store Memory
00002    C  0  0
00003    C  0  0   Purpose: Deposit the specified data in the memory location
00004    C  0  0   specified in 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           specified data value
00011    C  0  0
00012    C  0  0   Outputs: stores the 8 bits of data at the specified location
00013    C  0  0
00014    C  0  0   Procedures Invoked: none
00015    C  0  0 }
00016       0  0 module StoreMemory(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';
```

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

00079       0  0 type
00080       0  0     %include '[-]MemoryType.defn/list'
00081 I     0  0     MemoryArray = array[LowMemoryLimit..HighMemoryLimit] of unsigned;
00082       0  0
00083       0  0 var
00084       0  0     Memory: [external] MemoryArray;
00085       0  0     Address: integer;                       {pointer into memory array}
00086       0  0     WordPointer: integer;
00087       0  0     BytePointer: integer;
00088       0  0
00089       0  0     Unchanged: unsigned;           {value of memory array which
00090    C  0  0                                     will remain unchanged}
00091       0  0     NewDatum: unsigned;                     {new value of memory location}
00092       0  0
00093       1  0 [global] procedure StoreMemory(SMAddress: unsigned;
00094       1  0           SMSegment: unsigned;
00095       1  0           SMValue: unsigned;
00096       1  0           var SMStopCode: char);
```

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

00098       1  1 begin
00099       1  1     Address := int(SMAddress + (SMSegment * 16)); [form array index]
00100       1  1     if ((Address < (LowMemoryLimit * BytesPerWord))
00101       1  1   or (Address > ((HighMemoryLimit * BytesPerWord) + 1))) then
00102       1  1       SMStopCode := BadMemoryAddress
00103       1  1
00104       1  2     else begin
00105       1  2   WordPointer := Address div BytesPerWord;
00106       1  2   BytePointer := Address rem BytesPerWord;
00107       1  2
00108       1  3   case BytePointer of
00109       1  3
00110       1  4   0: begin
00111       1  4       Unchanged := uand( Memory[WordPointer], unot(Byte0Mask));
00112       1  4       NewDatum := uand( (SMValue * Byte0Multiplier), Byte0Mask);
00113       1  4       Memory[WordPointer] := uor( Unchanged, NewDatum);
00114       1  3   end;
00115       1  3
00116       1  4   1: begin
00117       1  4       Unchanged := uand( Memory[WordPointer], unot(Byte1Mask));
00118       1  4       NewDatum := uand( (SMValue * Byte1Multiplier), Byte1Mask);
00119       1  4       Memory[WordPointer] := uor( Unchanged, NewDatum);
00120       1  3   end;
00121       1  3
00122       1  4   2: begin
00123       1  4       Unchanged := uand( Memory[WordPointer], unot(Byte2Mask));
00124       1  4       NewDatum := uand( (SMValue * Byte2Multiplier), Byte2Mask);
00125       1  4       Memory[WordPointer] := uor( Unchanged, NewDatum);
00126       1  3   end;
00127       1  3
00128       1  4   3: begin
00129       1  4       Unchanged := uand( Memory[WordPointer], unot(Byte3Mask));
00130       1  4       NewDatum := uand( (SMValue * Byte3Multiplier), Byte3Mask);
00131       1  4       Memory[WordPointer] := uor( Unchanged, NewDatum);
00132       1  3   end;
00133       1  3
00134       1  3   otherwise SMStopCode := BadMemoryAddress;
00135       1  3
00136       1  2   end; [case BytePointer]
00137       1  2
00138       1  1     end;
00139       1  1
00140       0  0 end; [procedure StoreMemory]
00141       0  0 end. [program]
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	388	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	20	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS STOREMEMORY.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]STOREMEMORY.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]STOREMEMORY.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	408	00:00.5	00:02.2
Source Analysis	218	00:00.6	00:03.4
Source Listing	14	00:00.4	00:02.6
Tree Construction	103	00:00.1	00:00.2
Flow Analysis	43	00:00.1	00:00.3
Value Propagation	10	00:00.0	00:00.1
Profit Analysis	113	00:00.1	00:00.3
Context Analysis	113	00:00.6	00:01.1
Name Packing	7	00:00.0	00:00.0
Code Selection	88	00:00.1	00:00.1
Final	82	00:00.1	00:00.8
TOTAL	1203	00:02.7	00:11.1

COMPILATION STATISTICS

```
  CPU Time:       00:02.7         (3192 Lines/Minute)
```

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

00001    C  0  0 { Title: Store Ouput datum
00002    C  0  0
00003    C  0  0   Purpose: Put datum into the output store
00004    C  0  0
00005    C  0  0   Author: William A. Chapman      Date: February 2, 1986
00006    C  0  0
00007    C  0  0   Inputs: OutputDatum - datum to be output
00008    C  0  0           OutputPortMap - maps port number into OutputData
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: (Put8BitsOfData)
00014    C  0  0           LIB$STOP
00015    C  0  0           $QIOW
00016    C  0  0
00017    C  0  0   Functions Invoked: MaskShiftRight
00018    C  0  0 }
00019       0  0 [inherit ('sys$library:starlet')] module StoreOutput(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      %include  [-]StopCode.defn/list'
00084 I     0  0      Breakpoint              = 'B';
00085 I     0  0      Call          = 'C';
00086 I     0  0      ControlD              = 'D';
00087 I     0  0      BadOpcode               = 'E';
00088 I     0  0      BadCEAModeValue         = 'F';
00089 I     0  0      BadRMDModeValue         = 'G';
00090 I     0  0      Halt          = 'H';
00091 I     0  0      BadRegisterID = 'I';
00092 I     0  0      BadOpcodeKey  = 'K';
00093 I     0  0      BadMemoryAddress        = 'M';
00094 I     0  0      Normal                  = 'N';
00095 I     0  0      BadPortAddress          = 'P';
00096 I     0  0      BadOpcodeClass          = 'Q';
00097 I     0  0      Return                  = 'R';
00098 I     0  0      BadCheckSum             = 'S';
00099 I     0  0      BadOperandType          = 'T';
```

```
00100 I     0  0      BadMemoryWidth         = 'W';
00101 I     0  0      BadOpcodeExtension     = 'X';
00102 I     0  0      NoMemoryAccess       = 'Z';
00103       0  0
00104       0  0      ASCIIMask = %X'7F';
00105       0  0
```

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

00107       0  0 type
00108       0  0     %include  {-]Type.defn/list'
00109 I     0  0     ZeroOne = -1..1;                        {defines integer with range
00110 I C   0  0                                              of zero to one with an invalid
00111 I C   0  0                                              state of -1}
00112 I     0  0
00113 I     0  0     OpcodeLUTEntry = record                 {defines entries in opcode LUT}
00114 I     0  0   OpcodeClass: char;                        {class D, A, L, S, C, P}
00115 I     0  0   OpcodeKey: integer;                       {key to interpret opcode from
00116 I C   0  0                                              0 to 7}
00117 I     0  0   DirectionBitPresent: boolean;
00118 I     0  0   WidthBitPresent: boolean
00119 I     0  0     end;
00120 I     0  0
00121 I     0  0     OpcodeType = record
00122 I     0  0   Direction: ZeroOne;                      {to or from CPU}
00123 I     0  0   Full: unsigned;                          {full opcode}
00124 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00125 I     0  0     end;
00126 I     0  0
00127 I     0  0     RegisterType = record
00128 I     0  0   Id: integer;                            {identifier}
00129 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00130 I     0  0     end;
00131 I     0  0
00132 I     0  0     EffectiveAddressType = record
00133 I     0  0   Mode: char;                             {register or memory}
00134 I     0  0   Width: ZeroOne;                         {8 or 16 bits}
00135 I     0  0   Address: unsigned;                      {memory address or
00136 I C   0  0                                              register designation}
00137 I     0  0   Segment: integer                        {segment register to use}
00138 I     0  0     end;
00139 I     0  0
00140 I     0  0     NameType = packed array[1..10] of char;
00141 I     0  0
00142 I     0  0     Characters = set of    ..'~';
00143 I     0  0     LettersAndNumbers = set of '0'..'z';
00144 I     0  0     Letters = set of 'A'..'z';
00145 I     0  0     Numbers = set of '0'..'9';
00146 I     0  0
00147 I     0  0     FileName = packed array [1..FilenameLength] of char;
00148       0  0
00149       0  0     %include '[-]IOType.defn/list'
00150 I     0  0     PortEntry = record
```

```
00151 I     0  0   Address: unsigned;
00152 I     0  0   AuxIndex: integer;
00153 I     0  0   DataIndex: integer;
00154 I     0  0   InputIndex: integer;
00155 I     0  0   PortWidth: integer;
00156 I     0  0   LowerUpperIndicator: char;
00157 I     0  0   InOutIndicator: char
00158 I     0  0     end;
00159 I     0  0
00160 I     0  0     PortMap = array[ FirstIOPortIndex..MaxNumberOfIOPorts] of PortEntry;
00161       0  0
```

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

00162       0  0     QuadWord = [word] 0..65535;
00163       0  0     StatusBlock = record
00164       0  0   IOStatus: QuadWord;
00165       0  0   Count: QuadWord;
00166       0  0   DeviceInfo: integer
00167       0  0     end;
00168       0  0
00169       0  0
00170       0  0
00171       0  0
00172       0  0 var
00173       0  0     %include '[-]IOVarExternal.defn/list'
00174 I     0  0     IOPortMap: [external] PortMap;
00175 I     0  0
00176 I     0  0     IOData: [external] array[FirstIOPortIndex..MaxNumberofIOPorts,
00177 I     0  0   FirstIODataPoint..MaxNumberofIOWords] of unsigned;
00178       0  0
00179       0  0     LowDatum: unsigned;
00180       0  0     HighDatum: unsigned;
00181       0  0
00182       0  0     OAIndex: integer;
00183       0  0     Found: boolean;
00184       0  0     Done: boolean;
00185       0  0
00186       0  0     OutputGenerated: [external] boolean;
00187       0  0
00188       0  0     ASCIIChar: char;
00189       0  0
00190       0  0     SysStat: integer;
00191       0  0
00192       0  0     ChannelID: [external, volatile] array[1..2] of QuadWord;
00193       0  0
00194       0  0     IOBlock: StatusBlock;
00195       0  0
00196       0  0
00197       0  0
00198       0  0
00199       0  0
00200       1  0 [global] procedure StoreOutput(SOutWidth: integer;
00201       1  0                    SOutPortAddress: unsigned;
00202       1  0                    SOutDatum: unsigned;
00203       1  0                    var SOStopCode: char);
00204       1  0
00205       2  0 function MaskShiftRight(MSRValue: unsigned;
```

```
00206      2  0                    MSRMask: unsigned;
00207      1  0                    MSRDivisor:integer): unsigned; external;
00208      1  0
00209      1  0 [asynchronous] procedure lib$stop( %immed L$SStopValue: integer); external;
```

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

00211       1  0     %include 'Put8BitsOfData.subpas/list'
00212 I C   1  0 { Title: Put8BitsofData
00213 I C   1  0
00214 I C   1  0   Purpose: Take 8 bits of data passed from the calling routine
00215 I C   1  0           and store it in the IO data buffer.
00216 I C   1  0
00217 I C   1  0   Author: William A. Chapman      Date: May 17, 1986
00218 I C   1  0
00219 I C   1  0   Inputs: Input Data is obtained from the calling routine.
00220 I C   1  0
00221 I C   1  0   Outputs: Data is stored in the IO data buffer.
00222 I C   1  0
00223 I C   1  0   Procedures Invoked: none
00224 I C   1  0 }
00225 I     2  0 procedure Put8BitsofData( PortPointer: integer;
00226 I     2  0           Location: integer;
00227 I     2  0           Value: unsigned);
00228 I     2  0
00229 I     2  0 var
00230 I     2  0     Unchanged: unsigned;
00231 I     2  0     NewDatum: unsigned;
00232 I     2  0
00233 I     2  0     WordPointer: integer;
00234 I     2  0     BytePointer: integer;
```

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

00236 I    2  1 begin
00237 I    2  1     WordPointer := (((Location - 1) div BytesPerWord) + 1);
00238 I    2  1     BytePointer := ((Location - 1)  rem BytesPerWord);
00239 I    2  1
00240 I    2  2     case BytePointer of
00241 I    2  2
00242 I    2  3     0: begin
00243 I    2  3   Unchanged := uand( IOData[PortPointer,WordPointer], unot(Byte0Mask));
00244 I    2  3   NewDatum := uand( (Value * Byte0Multiplier), Byte0Mask);
00245 I    2  3   IOData[PortPointer,WordPointer] := uor( Unchanged, NewDatum);
00246 I    2  2       end;
00247 I    2  2
00248 I    2  3     1: begin
00249 I    2  3   Unchanged := uand( IOData[PortPointer,WordPointer], unot(Byte1Mask));
00250 I    2  3   NewDatum := uand( (Value * Byte1Multiplier), Byte1Mask);
00251 I    2  3   IOData[PortPointer,WordPointer] := uor( Unchanged, NewDatum);
00252 I    2  2       end;
00253 I    2  2
00254 I    2  3     2: begin
00255 I    2  3   Unchanged := uand( IOData[PortPointer,WordPointer], unot(Byte2Mask));
00256 I    2  3   NewDatum := uand( (Value * Byte2Multiplier), Byte2Mask);
00257 I    2  3   IOData[PortPointer,WordPointer] := uor( Unchanged, NewDatum);
00258 I    2  2       end;
00259 I    2  2
00260 I    2  3     3: begin
00261 I    2  3   Unchanged := uand( IOData[PortPointer,WordPointer], unot(Byte3Mask));
00262 I    2  3   NewDatum := uand( (Value * Byte3Multiplier), Byte3Mask);
00263 I    2  3   IOData[PortPointer,WordPointer] := uor( Unchanged, NewDatum);
00264 I    2  2       end;
00265 I    2  2
00266 I    2  2     otherwise writeln( 'Put8BitsOfData: Invalid Byte Pointer:  , BytePointer:1);
00267 I    2  2
00268 I    2  1     end; {case BytePointer]
00269 I    1  0 end; {Put8BitsofData]
```

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

00271      1  1 begin
00272      1  1     OAIndex := FirstIOPortIndex;
00273      1  1     Found := false;
00274      1  1     Done := false;
00275      1  1
00276      1  2     if (SOutPortAddress = CRTOutPortNumber) then begin
00277      1  2   ASCIIChar := chr( uand( SOutDatum, ASCIIMask));
00278      1  2
00279      1  2   SysStat := $QIOW( efn := 2, chan := ChannelID[2], func := IO$_WriteVBlk,
00280      1  2       iosb := IOBlock, p1 := %ref ASCIIChar, p2 := 1);
00281      1  2   if (not odd( SysStat)) then Lib$Stop( SysStat);
00282      1  2
00283      1  2     end  {if (SOutPortAddress = CRTOutPortNumber)}
00284      1  2
00285      1  2     else begin
00286      1  3   while (not Done) do begin
00287      1  4       if (SOutPortAddress = IOPortMap[OAIndex].Address) then begin
00288      1  4           Found := true;
00289      1  4           Done := true;
00290      1  4       end
00291      1  3       else OAIndex := OAIndex + 1;
00292      1  4       if (OAIndex > MaxNumberofIOPorts) then begin
00293      1  4           SOStopCode := BadPortAddress;
00294      1  4           Done := true;
00295      1  3       end;
00296      1  2   end; {while not Done}
00297      1  3   if (Found = true) then begin
00298      1  3       OutputGenerated := true;
00299      1  3       LowDatum := uand(SOutDatum, Low8Bits);
00300      1  3       Put8BitsofData( OAIndex, IOPortMap[OAIndex].DataIndex, LowDatum);
00301      1  3       if (IOPortMap[OAIndex].InOutIndicator <> InputIndicator) then
00302      1  3         IOPortMap[OAIndex].DataIndex := IOPortMap[OAIndex].DataIndex + 1;
00303      1  4       if (SOutWidth = SixteenBits) then begin
00304      1  4           OAIndex := IOPortMap[OAIndex].AuxIndex;
00305      1  4           HighDatum := MaskShiftRight(SOutDatum, High8Bits, Bit8Divisor);
00306      1  4           Put8BitsofData( OAIndex, IOPortMap[OAIndex].DataIndex, HighDatum);
00307      1  4           if (IOPortMap[OAIndex].InOutIndicator <> InputIndicator) then
00308      1  4             IOPortMap[OAIndex].DataIndex := IOPortMap[OAIndex].DataIndex + 1;
00309      1  3       end; {if SOutWidth}
00310      1  2   end; {if Found = true}
00311      1  1     end; {else begin}
00312      1  1
00313      0  0 end;
00314      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	794	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 STOREOUTPUT.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]STOREOUTPUT.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]STOREOUTPUT.OBJ;1
/NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	415	00:00.5	00:03.4
Source Analysis	869	00:04.9	00:19.2
Source Listing	43	00:00.7	00:04.6
Tree Construction	133	00:00.3	00:00.4
Flow Analysis	48	00:00.3	00:00.5
Value Propagation	10	00:00.0	00:00.0
Profit Analysis	135	00:00.3	00:00.5
Context Analysis	153	00:01.3	00:04.6
Name Packing	8	00:00.0	00:00.3

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

00001    C  0  0 { Title: StoreRegPtr
00002    C  0  0
00003    C  0  0   Purpose: Store the value specified into the appropriate
00004    C  0  0            register or pointer in the store "Registers Pointers"
00005    C  0  0
00006    C  0  0   Author: William A. Chapman      Date: December 22, 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            Value - the value of the desired register or pointer
00012    C  0  0
00013    C  0  0   Outputs: The store is updated with the desired value
00014    C  0  0
00015    C  0  0   Procedures Invoked: FetchRegPtr
00016    C  0  0 }
00017       0  0 module StoreRegPtr(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;
00040       0  0
00041       0  0     %include  [-]RegID.defn/list'
00042  I    0  0     ALId = %B'000';
00043  I    0  0     ALWidth = 0;
00044  I    0  0
00045  I    0  0     CLId = %B'001';
```

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

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

00056 I    0  0
00057 I    0  0     CHId = %B'101';
00058 I    0  0     CHWidth = 0;
00059 I    0  0
00060 I    0  0     DHId = %B'110';
00061 I    0  0     DHWidth = 0;
00062 I    0  0
00063 I    0  0     BHId = %B'111';
00064 I    0  0     BHWidth = 0;
00065 I    0  0
00066 I    0  0     AXId = %B'000';
00067 I    0  0     AXWidth = 1;
00068 I    0  0     ALorAXId = %B'000';
00069 I    0  0
00070 I    0  0     CXId = %B'001';
00071 I    0  0     CXWidth = 1;
00072 I    0  0
00073 I    0  0     DXId = %B'010';
00074 I    0  0     DXWidth = 1;
00075 I    0  0
00076 I    0  0     BXId = %B'011';
00077 I    0  0     BXWidth = 1;
00078 I    0  0
00079 I    0  0     SPId = %B'100';
00080 I    0  0     SPWidth = 1;
00081 I    0  0
00082 I    0  0     BPId = %B'101';
00083 I    0  0     BPWidth = 1;
00084 I    0  0
00085 I    0  0     SIId = %B'110';
00086 I    0  0     SIWidth = 1;
00087 I    0  0
00088 I    0  0     DIId = %B'111';
00089 I    0  0     DIWidth = 1;
00090 I    0  0
00091 I    0  0     SRWidth = 2;
00092 I    0  0
00093 I    0  0     ESId = %B'00';
00094 I    0  0     ESWidth = 2;
00095 I    0  0
00096 I    0  0     CSId = %B'01';
00097 I    0  0     CSWidth = 2;
00098 I    0  0
00099 I    0  0     SSId = %B'10';
```

```
00100 I     0   0      SSWidth = 2;
00101 I     0   0
00102 I     0   0      DSId = %B'11';
00103 I     0   0      DSWidth = 2;
00104 I     0   0
```

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

00106       0  0     %include  [-]StopCode.defn/list'
00107 I     0  0     Breakpoint              = 'B';
00108 I     0  0     Call          = 'C';
00109 I     0  0     ControlD              = 'D';
00110 I     0  0     BadOpcode               = 'E';
00111 I     0  0     BadCEAModeValue         = 'F';
00112 I     0  0     BadRMDModeValue         = 'G';
00113 I     0  0     Halt          = 'H';
00114 I     0  0     BadRegisterID = 'I';
00115 I     0  0     BadOpcodeKey  = 'K';
00116 I     0  0     BadMemoryAddress        = 'M';
00117 I     0  0     Normal                  = 'N';
00118 I     0  0     BadPortAddress          = 'P';
00119 I     0  0     BadOpcodeClass          = 'Q';
00120 I     0  0     Return                  = 'R';
00121 I     0  0     BadCheckSum             = 'S';
00122 I     0  0     BadOperandType          = 'T';
00123 I     0  0     BadMemoryWidth          = 'W';
00124 I     0  0     BadOpcodeExtension      = 'X';
00125 I     0  0     NoMemoryAccess        = 'Z';
```

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

00127       0  0 var
00128       0  0     %include '[-]RegExternal.defn/list'
00129 I     0  0     AX: [external] unsigned;      [value of the AX register]
00130 I     0  0     CX: [external] unsigned;      [value of the CX register]
00131 I     0  0     DX: [external] unsigned;      [value of the DX register]
00132 I     0  0     BX: [external] unsigned;      [value of the BX register]
00133 I     0  0     SP: [external] unsigned;      [value of the SP register]
00134 I     0  0     BP: [external] unsigned;      [value of the BP register]
00135 I     0  0     SI: [external] unsigned;      [value of the SI register]
00136 I     0  0     DI: [external] unsigned;      [value of the DI register]
00137 I     0  0
00138 I     0  0     ES: [external] unsigned;      [value of the ES register]
00139 I     0  0     CS: [external] unsigned;      [value of the CS register]
00140 I     0  0     SS: [external] unsigned;      [value of the SS register]
00141 I     0  0     DS: [external] unsigned;      [value of the DS register]
00142 I     0  0
00143       0  0     LowValue: unsigned;
00144       0  0     HighValue: unsigned;
00145       0  0     Unchanged: unsigned;
00146       0  0
00147       0  0
00148       1  0 [global] procedure StoreRegPtr(SRPWidth: integer;
00149       1  0           SRPDesignator: integer;
00150       1  0           SRPValue: unsigned;
00151       1  0           var SRPStopCode: char);
00152       1  0
00153       2  0 procedure FetchRegPtr(FRPWidth: integer;
00154       2  0           FRPDesignator: integer;
00155       2  0           var FRPValue: unsigned;
00156       1  0           var FRPStopCode: char); external;
```

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

00158       1  1 begin
00159       1  2     case SRPWidth of
00160       1  2
00161       1  3   0:  case SRPDesignator of       {8 bit registers}
00162       1  3
00163       1  4           ALId: begin
00164       1  4                   FetchRegPtr( AHWidth, AHId, HighValue, SRPStopCode);
00165       1  4                   Unchanged := uand( ( HighValue * 256), High8Bits); {get rid of low 8 bits}
00166       1  4                   AX := uor( uand( SRPValue, Low8Bits), Unchanged); {be sure its only 8 bits
00167   C   1  4                            and or it with the upper 8 bits}
00168       1  3               end; {case ALId}
00169       1  3
00170       1  4           CLId: begin
00171       1  4                   FetchRegPtr( CHWidth, CHId, HighValue, SRPStopCode);
00172       1  4                   Unchanged := uand( ( HighValue * 256), High8Bits); {get rid of low 8 bits}
00173       1  4                   CX := uor( uand( SRPValue, Low8Bits), Unchanged);
00174       1  3               end; {case CLId}
00175       1  3
00176       1  4           DLId: begin
00177       1  4                   FetchRegPtr( DHWidth, DHId, HighValue, SRPStopCode);
00178       1  4                   Unchanged := uand( ( HighValue * 256), High8Bits); {get rid of low 8 bits}
00179       1  4                   DX := uor( uand( SRPValue, Low8Bits), Unchanged);
00180       1  3               end; {case DLId}
00181       1  3
00182       1  4           BLId: begin
00183       1  4                   FetchRegPtr( BHWidth, BHId, HighValue, SRPStopCode);
00184       1  4                   Unchanged := uand( ( HighValue * 256), High8Bits); {get rid of low 8 bits}
00185       1  4                   BX := uor( uand( SRPValue, Low8Bits), Unchanged);
00186       1  3               end; {case BLId}
00187       1  3
00188       1  4           AHId: begin
00189       1  4                   FetchRegPtr( ALWidth, ALId, LowValue, SRPStopCode);
00190       1  4                   AX := uor( uand( (SRPValue * 256), High8Bits), LowValue);
00191       1  3               end; {case AHId}
00192       1  3
00193       1  4           CHId: begin
00194       1  4                   FetchRegPtr( CLWidth, CLId, LowValue, SRPStopCode);
00195       1  4                   CX := uor( uand( (SRPValue * 256), High8Bits), LowValue);
00196       1  3               end; {case CHId}
00197       1  3
00198       1  4           DHId: begin
00199       1  4                   FetchRegPtr( DLWidth, DLId, LowValue, SRPStopCode);
00200       1  4                   DX := uor( uand( (SRPValue * 256), High8Bits), LowValue);
00201       1  3               end; {case DHId}
```

```
00202     1  3
00203     1  4          BHId: begin
00204     1  4                  FetchRegPtr( BLWidth, BLId, LowValue, SRPStopCode);
00205     1  4                  BX := uor( uand( (SRPValue * 256), High8Bits), LowValue);
00206     1  4              end {case BHId}
00207     1  4
00208     1  3          otherwise SRPStopCode := BadRegisterID;
00209     1  3
00210     1  2      end; {case 0 - SRPWidth}
```

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

00212       1  3   1:  case SRPDesignator of       {16 bit registers}
00213       1  3
00214       1  3           AXId: AX := uand( SRPValue, All16Bits);
00215       1  3           CXId: CX := uand( SRPValue, All16Bits);
00216       1  3           DXId: DX := uand( SRPValue, All16Bits);
00217       1  3           BXId: BX := uand( SRPValue, All16Bits);
00218       1  3           SPId: SP := uand( SRPValue, All16Bits);
00219       1  3           BPId: BP := uand( SRPValue, All16Bits);
00220       1  3           SIId: SI := uand( SRPValue, All16Bits);
00221       1  3           DIId: DI := uand( SRPValue, All16Bits);
00222       1  3           otherwise SRPStopCode := BadRegisterID;
00223       1  2       end; {case 1 - SRPWidth}
00224       1  2
00225       1  3         2: case SRPDesignator of  {16 bit pointers}
00226       1  3           ESID: ES := uand( SRPValue, All16Bits);
00227       1  3           CSId: CS := uand( SRPValue, All16Bits);
00228       1  3           SSId: SS := uand( SRPValue, All16Bits);
00229       1  3           DSId: DS := uand( SRPValue, All16Bits);
00230       1  3           otherwise SRPStopCode := BadRegisterID;
00231       1  3       end {case 2 - SRPWidth}
00232       1  3
00233       1  2   otherwise SRPStopCode := BadRegisterID;
00234       1  2
00235       1  1     end; {SRPWidth}
00236       0  0 end; {procedure StoreRegPtr}
00237       0  0 end. {program}
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	820	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 STOREREGPTR.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]STOREREGPTR.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]STOREREGPTR.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	00:03.3
Source Analysis	266	00:00.9	00:07.1
Source Listing	21	00:00.5	00:03.6
Tree Construction	116	00:00.2	00:00.6
Flow Analysis	33	00:00.1	00:00.2
Value Propagation	9	00:00.0	00:00.1
Profit Analysis	25	00:00.1	00:00.3
Context Analysis	153	00:00.9	00:05.5
Name Packing	6	00:00.0	00:00.0
Code Selection	74	00:00.2	00:00.4
Final	64	00:00.3	00:02.2
TOTAL	1174	00:03.7	00:23.2

COMPILATION STATISTICS

CPU Time: 00:03.7 (3864 Lines/Minute)

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

00001    C  0  0 { Title: String
00002    C  0  0
00003    C  0  0   Purpose: Execute the string opcodes
00004    C  0  0
00005    C  0  0   Author: William A. Chapman      Date: December 22, 1986
00006    C  0  0
00007    C  0  0   Inputs: Opcode Record
00008    C  0  0           Register Record
00009    C  0  0           Effective Address Record
00010    C  0  0
00011    C  0  0   Outputs: Registers,flags and memory locations are updated as required.
00012    C  0  0
00013    C  0  0   Procedures Invoked: FetchRegPtr
00014    C  0  0           StoreRegPtr
00015    C  0  0           FetchOperand
00016    C  0  0           StoreData
00017    C  0  0           SignExtend7
00018    C  0  0           SignExtend15
00019    C  0  0           UpdateZeroFlag
00020    C  0  0           UpdateSignFlag
00021    C  0  0           UpdateParityFlag
00022    C  0  0           UpdateOverflowFlag_Sub
00023    C  0  0           UpdateCarryFlag_Sub
00024    C  0  0           UpdateAuxCarryFlag_Sub
00025    C  0  0 }
00026       0  0 module String(input, output );
00027       0  0
00028       0  0 const
00029       0  0     %include  [-]Const.defn/list'
00030 I     0  0     FirstOpcodeValue = 0;
00031 I     0  0     LastOpcodeValue = 255;
00032 I     0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00033 I     0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00034 I     0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00035 I     0  0     Bit8Multiplier = %X'100';
00036 I     0  0     Bit8Divisor    = %X'100';
00037 I     0  0     WordMultiplier = %X'100';
00038 I     0  0
00039 I     0  0     EightBits = 0;                       {width for 8 bits}
00040 I     0  0     SixteenBits = 1;                     {width for 16 bits}
00041 I     0  0
00042 I     0  0     SimpleMode = %B'0';
00043 I     0  0     SimpleRM = %B'110';
00044 I     0  0
00045 I     0  0     LowMemoryLimit = 0;                  {low limit on memory array}
```

```
00046 I     0  0      HighMemoryLimit = 2048;              {high limit on memory array}
00047 I     0  0
00048 I     0  0      FilenameLength = 40;
```

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

00050       0  0     %include  [-]Flags.defn/list'
00051 I     0  0     SetHigh = true;
00052 I     0  0     Clear = false;
00053 I     0  0
00054 I     0  0     CarryFlag     = %B'000000000001';
00055 I     0  0     ParityFlag    = %B'000000000100';
00056 I     0  0     AuxCarryFlag  = %B'000000010000';
00057 I     0  0     ZeroFlag      = %B'000001000000';
00058 I     0  0     SignFlag      = %B'000010000000';
00059 I     0  0     TrapFlag      = %B'000100000000';
00060 I     0  0     InterruptFlag = %B'001000000000';
00061 I     0  0     DirectionFlag = %B'010000000000';
00062 I     0  0     OverflowFlag  = %B'100000000000';
00063       0  0
00064       0  0     %include  [-]RegID.defn/list'
00065 I     0  0     ALId = %B'000';
00066 I     0  0     ALWidth = 0;
00067 I     0  0
00068 I     0  0     CLId = %B'001 ;
00069 I     0  0     CLWidth = 0;
00070 I     0  0
00071 I     0  0     DLId = %B'010';
00072 I     0  0     DLWidth = 0;
00073 I     0  0
00074 I     0  0     BLId = %B'011';
00075 I     0  0     BLWidth = 0;
00076 I     0  0
00077 I     0  0     AHId = %B'100';
00078 I     0  0     AHWidth = 0;
00079 I     0  0
00080 I     0  0     CHId = %B'101';
00081 I     0  0     CHWidth = 0;
00082 I     0  0
00083 I     0  0     DHId = %B'110';
00084 I     0  0     DHWidth = 0;
00085 I     0  0
00086 I     0  0     BHId = %B'111';
00087 I     0  0     BHWidth = 0;
00088 I     0  0
00089 I     0  0     AXId = %B'000';
00090 I     0  0     AXWidth = 1;
00091 I     0  0     ALorAXId = %B'000';
00092 I     0  0
00093 I     0  0     CXId = %B'001';
```

```
00094 I     0   0      CXWidth = 1;
00095 I     0   0
00096 I     0   0      DXId = %B'010';
00097 I     0   0      DXWidth = 1;
00098 I     0   0
00099 I     0   0      BXId = %B'011 ;
00100 I     0   0      BXWidth = 1;
00101 I     0   0
00102 I     0   0      SPId = %B'100';
00103 I     0   0      SPWidth = 1;
00104 I     0   0
```

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

00105 I     0  0      BPId = %B'101' ;
00106 I     0  0      BPWidth = 1;
00107 I     0  0
00108 I     0  0      SIId = %B'110';
00109 I     0  0      SIWidth = 1;
00110 I     0  0
00111 I     0  0      DIId = %B'111';
00112 I     0  0      DIWidth = 1;
00113 I     0  0
00114 I     0  0      SRWidth = 2;
00115 I     0  0
00116 I     0  0      ESId = %B'00';
00117 I     0  0      ESWidth = 2;
00118 I     0  0
00119 I     0  0      CSId = %B'01' ;
00120 I     0  0      CSWidth = 2;
00121 I     0  0
00122 I     0  0      SSId = %B'10';
00123 I     0  0      SSWidth = 2;
00124 I     0  0
00125 I     0  0      DSId = %B'11';
00126 I     0  0      DSWidth = 2;
00127 I     0  0
00128       0  0
00129       0  0      %include '[-]StopCode.defn/list'
00130 I     0  0      Breakpoint              = 'B';
00131 I     0  0      Call          = 'C';
00132 I     0  0      ControlD              = 'D';
00133 I     0  0      BadOpcode               = 'E';
00134 I     0  0      BadCEAModeValue         = 'F';
00135 I     0  0      BadRMDModeValue         = 'G';
00136 I     0  0      Halt          = 'H';
00137 I     0  0      BadRegisterID = 'I';
00138 I     0  0      BadOpcodeKey  = 'K';
00139 I     0  0      BadMemoryAddress        = 'M';
00140 I     0  0      Normal                  = 'N';
00141 I     0  0      BadPortAddress          = 'P';
00142 I     0  0      BadOpcodeClass          = 'Q';
00143 I     0  0      Return                  = 'R';
00144 I     0  0      BadCheckSum             = 'S';
00145 I     0  0      BadOperandType          = 'T';
00146 I     0  0      BadMemoryWidth          = 'W';
00147 I     0  0      BadOpcodeExtension      = 'X';
00148 I     0  0      NoMemoryAccess        = 'Z';
```

```
00149       0  0
00150       0  0      MemoryAccess = 'M';
```

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

00152       0  0 type
00153       0  0     %include '[-]Type.defn/list'
00154 I     0  0     ZeroOne = -1..1;                        {defines integer with range
00155 I C   0  0                                             of zero to one with an invalid
00156 I C   0  0                                             state of -1}
00157 I     0  0
00158 I     0  0     OpcodeLUTEntry = record                 {defines entries in opcode LUT}
00159 I     0  0   OpcodeClass: char;                        {class D, A, L, S, C, P}
00160 I     0  0   OpcodeKey: integer;                       {key to interpret opcode from
00161 I C   0  0                                             0 to 7}
00162 I     0  0   DirectionBitPresent: boolean;
00163 I     0  0   WidthBitPresent: boolean
00164 I     0  0     end;
00165 I     0  0
00166 I     0  0     OpcodeType = record
00167 I     0  0   Direction: ZeroOne;                      {to or from CPU}
00168 I     0  0   Full: unsigned;                          {full opcode}
00169 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00170 I     0  0     end;
00171 I     0  0
00172 I     0  0     RegisterType = record
00173 I     0  0   Id: integer;                             {identifier}
00174 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00175 I     0  0     end;
00176 I     0  0
00177 I     0  0     EffectiveAddressType = record
00178 I     0  0   Mode: char;                              {register or memory}
00179 I     0  0   Width: ZeroOne;                         {8 or 16 bits}
00180 I     0  0   Address: unsigned;                      {memory address or
00181 I C   0  0                                             register designation}
00182 I     0  0   Segment: integer                        {segment register to use}
00183 I     0  0     end;
00184 I     0  0
00185 I     0  0     NameType = packed array[1..10] of char;
00186 I     0  0
00187 I     0  0     Characters = set of    ..'~';
00188 I     0  0     LettersAndNumbers = set of '0'..'z';
00189 I     0  0     Letters = set of 'A'..'z';
00190 I     0  0     Numbers = set of '0'..'9';
00191 I     0  0
00192 I     0  0     FileName = packed array [1..FilenameLength] of char;
00193       0  0
00194       0  0     %include '[-]FlagType.defn/list'
00195 I     0  0     FlagType = record                       {define processor flags}
```

```
00196 I    0  0   Carry: boolean;
00197 I    0  0   Parity: boolean;
00198 I    0  0   AuxCarry: boolean;
00199 I    0  0   Zero: boolean;
00200 I    0  0   Sign: boolean;
00201 I    0  0   Trap: boolean;
00202 I    0  0   Interrupt: boolean;
00203 I    0  0   Direction: boolean;
00204 I    0  0   Overflow: boolean
00205 I    0  0     end;
```

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

00207       0  0 var
00208       0  0     Flags: [external] FlagType;
00209       0  0
00210       0  0     SegmentOverRideCount: [external] integer;
00211       0  0     SegmentOverRideValue: [external] integer;
00212       0  0
00213       0  0     Continue: [external] LettersAndNumbers;
00214       0  0
00215       0  0     RepeatFlag: [external] boolean;
00216       0  0     RepeatUntilZFlag: [external] boolean;
00217       0  0
00218       0  0     SourceAddress: unsigned;
00219       0  0     DestinationAddress: unsigned;
00220       0  0     MinuendAddress: unsigned;
00221       0  0     SubtrahendAddress: unsigned;
00222       0  0
00223       0  0     LoadValue: unsigned;
00224       0  0     StoreValue: unsigned;
00225       0  0     MoveValue: unsigned;
00226       0  0
00227       0  0     Minuend: unsigned;
00228       0  0     Subtrahend: unsigned;
00229       0  0     Difference: unsigned;
00230       0  0     IntMinuend: integer;
00231       0  0     IntSubtrahend: integer;
00232       0  0
00233       0  0     Delta: integer;
00234       0  0
00235       0  0     OpcodeKey: integer;
00236       0  0
00237       0  0     LoopCount: unsigned;
00238       0  0     IntLoopCount: integer;
```

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

00240      1  0 [global] procedure String( Opcode: OpcodeType;
00241      1  0           Register: RegisterType;
00242      1  0           EA: EffectiveAddressType;
00243      1  0           var SStopCode: char);
00244      1  0
00245      2  0 procedure FetchRegPtr( FRPWidth: integer;
00246      2  0           FRPDesignator: integer;
00247      2  0           var FRPValue: unsigned;
00248      1  0           var FRPStopCode: char); external;
00249      1  0
00250      2  0 procedure StoreRegPtr( SRPWidth: integer;
00251      2  0           SRPDesignator: integer;
00252      2  0           SRPValue: unsigned;
00253      1  0           var SRPStopCode: char); external;
00254      1  0
00255      2  0 procedure FetchOperand( FOType: char;
00256      2  0           FOWidth: integer;
00257      2  0           FOAddress: unsigned;
00258      2  0           FOSegment: integer;
00259      2  0           var FOValue: unsigned;
00260      1  0           var FOStopCode: char); external;
00261      1  0
00262      2  0 procedure StoreData( SDType: char;
00263      2  0           SDWidth: integer;
00264      2  0           SDAddress: unsigned;
00265      2  0           SDSegment: integer;
00266      2  0           SDValue: unsigned;
00267      1  0           var SDStopCode: char); external;
00268      1  0
00269      1  0 function SignExtend7( SEValue: unsigned): integer; external;
00270      1  0
00271      1  0 function SignExtend15( SEValue: unsigned): integer; external;
00272      1  0
00273      2  0 function UpdateZeroFlag(UFWidth: integer;
00274      1  0           UFResult: unsigned): boolean; external;
00275      1  0
00276      2  0 function UpdateSignFlag(UFWidth: integer;
00277      1  0           UFResult: unsigned): boolean; external;
00278      1  0
00279      2  0 function UpdateParityFlag(UFWidth: integer;
00280      1  0           UFResult: unsigned): boolean; external;
```

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

00282       2  0 function UpdateOverflowFlag_Sub(UFWidth: integer;
00283       2  0           UFMinuend: unsigned;
00284       2  0           UFSubtrehend: unsigned;
00285       1  0           UFResult: unsigned): boolean; external;
00286       1  0
00287       2  0 function UpdateCarryFlag_Sub(UFWidth: integer;
00288       2  0           UFMinuend: unsigned;
00289       2  0           UFSubtrehend: unsigned;
00290       1  0           UFResult: unsigned): boolean; external;
00291       1  0
00292       2  0 function UpdateAuxCarryFlag_Sub(UFWidth: integer;
00293       2  0           UFMinuend: unsigned;
00294       2  0           UFSubtrehend: unsigned;
00295       1  0           UFResult: unsigned): boolean; external;
00296       1  0
00297       1  0
00298       1  0
00299       1  0
00300    C  1  0 { Title: FetchStringDatum
00301    C  1  0
00302    C  1  0   Purpose: Set up proper value of Segment Override for call to FetchOperand
00303    C  1  0
00304    C  1  0   Author: William A. Chapman      Date: January 2, 1987
00305    C  1  0
00306    C  1  0   Inputs: Width
00307    C  1  0           Address
00308    C  1  0           SegmentID
00309    C  1  0           SegmentOverrideCount
00310    C  1  0           SegmentOverrideValue
00311    C  1  0           MemoryAccess
00312    C  1  0
00313    C  1  0   Outputs: Datum
00314    C  1  0           SStopCode
00315    C  1  0
00316    C  1  0   Procedures Invoked: FetchOperand
00317    C  1  0 ]
00318       2  0 procedure FetchStringDatum( FSDWidth: integer;
00319       2  0           FSDAddress: unsigned;
00320       2  0           FSDSegment: integer;
00321       2  0           var FSDDatum: unsigned;
00322       2  0           var FSDStopCode: char);
00323       2  0
00324       2  1 begin
00325       2  1     if ( SegmentOverRideCount > 0) then FSDSegment := SegmentOverRideValue;
```

```
00326      2  1     FetchOperand( MemoryAccess, FSDWidth, FSDAddress, FSDSegment, FSDDatum, FSDStopCode);
00327      1  0 end;
```

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

00329      1  1 begin
00330      1  1     OpcodeKey := int(Opcode.Full);
00331      1  1
00332      1  1     Delta := Opcode.Width + 1;
00333      1  1     if (Flags.Direction = SetHigh) then Delta := -Delta;
00334      1  1
00335      1  2     case OpcodeKey of
00336      1  2
00337      1  2   %X'A4': [move string 3-118]
00338      1  3     with Opcode do begin
00339      1  3       FetchRegPtr( CXWidth, CXId, LoopCount, SStopCode);
00340      1  3       IntLoopCount := int( LoopCount);
00341      1  4       repeat
00342      1  4           FetchRegPtr( SIWidth, SIId, SourceAddress, SStopCode);
00343      1  4           FetchRegPtr( DIWidth, DIId, DestinationAddress, SStopCode);
00344      1  4           FetchStringDatum( Width, SourceAddress, DSId, MoveValue, SStopCode);
00345      1  4           if (SStopCode in Continue) then StoreData( MemoryAccess, Width,
00346      1  4               DestinationAddress, ESId, MoveValue, SStopCode);
00347      1  5           if (SStopCode in Continue) then begin
00348      1  5               SourceAddress := uint( int( SourceAddress) + Delta);
00349      1  5               StoreRegPtr( SIWidth, SIId, SourceAddress, SStopCode);
00350      1  5               DestinationAddress := uint( int( DestinationAddress) + Delta);
00351      1  5               StoreRegPtr( DIWidth, DIId, DestinationAddress, SStopCode);
00352      1  5           end
00353      1  4           else RepeatFlag := Clear;
00354      1  5           if ( RepeatFlag = SetHigh) then begin
00355      1  5               IntLoopCount := IntLoopCount - 1;
00356      1  5               StoreRegPtr( CXWidth, CXID, uint( IntLoopCount), SStopCode);
00357      1  5               if ( IntLoopCount = 0) then RepeatFlag := Clear;
00358      1  4           end;
00359      1  3       until ( RepeatFlag = Clear);
00360      1  2     end; [case 'A4']
```

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

00362       1  2    %X'A6': [compare string 3-64]
00363       1  3      with Opcode do begin
00364       1  3        FetchRegPtr( CXWidth, CXId, LoopCount, SStopCode);
00365       1  3        IntLoopCount := int( LoopCount);
00366       1  4        repeat
00367       1  4            FetchRegPtr( SIWidth, SIId, MinuendAddress, SStopCode);
00368       1  4            FetchStringDatum( Width, MinuendAddress, DSId, Minuend, SStopCode);
00369       1  4            FetchRegPtr( DIWidth, DIId, SubtrahendAddress, SStopCode);
00370       1  4            FetchOperand( MemoryAccess, Width, SubtrahendAddress, ESId,
00371       1  4                Subtrahend, SStopCode);
00372       1  5            if (Width = EightBits) then begin
00373       1  5                IntMinuend := SignExtend7(Minuend);
00374       1  5                IntSubtrahend := SignExtend7(Subtrahend);
00375       1  5            end
00376       1  5            else begin
00377       1  5                IntMinuend:= SignExtend15(Minuend);
00378       1  5                IntSubtrahend := SignExtend15(Subtrahend);
00379       1  4            end;
00380       1  4            Difference := uint(IntMinuend - IntSubtrahend);
00381       1  5            with Flags do begin
00382       1  5                AuxCarry := UpdateAuxCarryFlag_Sub( Width,
00383       1  5                    uint(IntMinuend), uint(Subtrahend), Difference);
00384       1  5                Carry := UpdateCarryFlag_Sub( Width,
00385       1  5                    uint(IntMinuend), uint(Subtrahend), Difference);
00386       1  5                Overflow := UpdateOverflowFlag_Sub( Width,
00387       1  5                    uint(IntMinuend), uint(Subtrahend), Difference);
00388       1  5                Parity := UpdateParityFlag(Width, Difference);
00389       1  5                Sign := UpdateSignFlag(Width, Difference);
00390       1  5                Zero := UpdateZeroFlag(Width, Difference);
00391       1  4            end; {with Flags}
00392       1  4
00393       1  5            if (SStopCode in Continue) then begin
00394       1  5                MinuendAddress := uint( int( MinuendAddress) + Delta);
00395       1  5                StoreRegPtr( SIWidth, SIId, MinuendAddress, SStopCode);
00396       1  5                SubtrahendAddress := uint( int( SubtrahendAddress) + Delta);
00397       1  5                StoreRegPtr( DIWidth, DIId, SubtrahendAddress, SStopCode);
00398       1  5            end
00399       1  4            else RepeatFlag := Clear;
00400       1  5            if ( RepeatFlag = SetHigh) then begin
00401       1  5                IntLoopCount := IntLoopCount - 1;
00402       1  5                StoreRegPtr( CXWidth, CXID, uint( IntLoopCount), SStopCode);
00403       1  5                if ( IntLoopCount = 0) then RepeatFlag := Clear;
00404       1  5                if ( RepeatUntilZFlag = Flags.Zero) then RepeatFlag := Clear;
00405       1  4            end;
```

```
00406      1  3       until ( RepeatFlag = Clear);
00407      1  2     end; {case 'A6'}
```

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

00409       1  2   %X'AA': {store string 3-160}
00410       1  3     with Opcode do begin
00411       1  3       FetchRegPtr( CXWidth, CXId, LoopCount, SStopCode);
00412       1  3       IntLoopCount := int( LoopCount);
00413       1  4       repeat
00414       1  4           FetchRegPtr( Width, ALorAXId, StoreValue, SStopCode);
00415       1  4           FetchRegPtr( DIWidth, DIId, DestinationAddress, SStopCode);
00416       1  5           if (SStopCode in Continue) then begin
00417       1  5               StoreData( MemoryAccess, Width, DestinationAddress, ESId,
00418       1  5                   StoreValue, SStopCode);
00419       1  5               DestinationAddress := uint( int( DestinationAddress) + Delta);
00420       1  5               StoreRegPtr( DIWidth, DIId, DestinationAddress, SStopCode);
00421       1  5           end
00422       1  4           else RepeatFlag := Clear;
00423       1  5           if ( RepeatFlag = SetHigh) then begin
00424       1  5               IntLoopCount := IntLoopCount - 1;
00425       1  5               StoreRegPtr( CXWidth, CXID, uint( IntLoopCount), SStopCode);
00426       1  5               if ( IntLoopCount = 0) then RepeatFlag := Clear;
00427       1  4           end;
00428       1  3       until ( RepeatFlag = Clear);
00429       1  2     end; {case 'AA'}
00430       1  2
00431       1  2   %X'AC': {load string 3-112}
00432       1  3     with Opcode do begin
00433       1  3       FetchRegPtr( CXWidth, CXId, LoopCount, SStopCode);
00434       1  3       IntLoopCount := int( LoopCount);
00435       1  4       repeat
00436       1  4           FetchRegPtr( SIWidth, SIId, SourceAddress, SStopCode);
00437       1  4           FetchStringDatum( Width, SourceAddress, DSId, LoadValue, SStopCode);
00438       1  5           if (SStopCode in Continue) then begin
00439       1  5               StoreRegPtr( Width, ALorAXId, LoadValue, SStopCode);
00440       1  5               SourceAddress := uint( int( SourceAddress) + Delta);
00441       1  5               StoreRegPtr( SIWidth, SIId, SourceAddress, SStopCode);
00442       1  5           end
00443       1  4           else RepeatFlag := Clear;
00444       1  5           if ( RepeatFlag = SetHigh) then begin
00445       1  5               IntLoopCount := IntLoopCount - 1;
00446       1  5               StoreRegPtr( CXWidth, CXID, uint( IntLoopCount), SStopCode);
00447       1  5               if ( IntLoopCount = 0) then RepeatFlag := Clear;
00448       1  4           end;
00449       1  3       until ( RepeatFlag = Clear);
00450       1  2     end; {case 'AC'}
```

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

00452       1  2   %X'AE': {scan string 3-154]
00453       1  3     with Opcode do begin
00454       1  3       FetchRegPtr( CXWidth, CXId, LoopCount, SStopCode);
00455       1  3       IntLoopCount := int( LoopCount);
00456       1  4       repeat
00457       1  4           FetchRegPtr( Width, ALorAXId, Minuend, SStopCode);
00458       1  4           FetchRegPtr( DIWidth, DIId, SubtrahendAddress, SStopCode);
00459       1  4           FetchOperand( MemoryAccess, Width, SubtrahendAddress, ESId,
00460       1  4               Subtrahend, SStopCode);
00461       1  5           if (Width = EightBits) then begin
00462       1  5               IntMinuend := SignExtend7(Minuend);
00463       1  5               IntSubtrahend := SignExtend7(Subtrahend);
00464       1  5           end
00465       1  5           else begin
00466       1  5               IntMinuend:= SignExtend15(Minuend);
00467       1  5               IntSubtrahend := SignExtend15(Subtrahend);
00468       1  4           end;
00469       1  4           Difference := uint(IntMinuend - IntSubtrahend);
00470       1  5           with Flags do begin
00471       1  5               AuxCarry := UpdateAuxCarryFlag_Sub( Width,
00472       1  5                   uint(IntMinuend), uint(Subtrahend), Difference);
00473       1  5               Carry := UpdateCarryFlag_Sub( Width,
00474       1  5                   uint(IntMinuend), uint(Subtrahend), Difference);
00475       1  5               Overflow := UpdateOverflowFlag_Sub( Width,
00476       1  5                   uint(IntMinuend), uint(Subtrahend), Difference);
00477       1  5               Parity := UpdateParityFlag(Width, Difference);
00478       1  5               Sign := UpdateSignFlag(Width, Difference);
00479       1  5               Zero := UpdateZeroFlag(Width, Difference);
00480       1  4           end; {with Flags}
00481       1  4
00482       1  5           if (SStopCode in Continue) then begin
00483       1  5               SubtrahendAddress := uint( int( SubtrahendAddress) + Delta);
00484       1  5               StoreRegPtr( DIWidth, DIId, SubtrahendAddress, SStopCode);
00485       1  5           end
00486       1  4           else RepeatFlag := Clear;
00487       1  5           if ( RepeatFlag = SetHigh) then begin
00488       1  5               IntLoopCount := IntLoopCount - 1;
00489       1  5               StoreRegPtr( CXWidth, CXID, uint( IntLoopCount), SStopCode);
00490       1  5               if ( IntLoopCount = 0) then RepeatFlag := Clear;
00491       1  5               if ( RepeatUntilZFlag = Flags.Zero) then RepeatFlag := Clear;
00492       1  4           end;
00493       1  3       until ( RepeatFlag = Clear);
00494       1  2     end; {case 'AE']
```

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

00496       1  2   %X'F2': {repeat 3-140}
00497       1  3     begin
00498       1  3       RepeatFlag := SetHigh;
00499       1  3       RepeatUntilZFlag := SetHigh;
00500       1  2     end;
00501       1  2
00502       1  2   %X'F3': {repeat 3-140}
00503       1  3     begin
00504       1  3       RepeatFlag := SetHigh;
00505       1  3       RepeatUntilZFlag := Clear;
00506       1  2     end;
00507       1  2
00508       1  2   otherwise SStopCode := BadOpcode;
00509       1  1     end; {case OpcodeKey}
00510       0  0 end;
00511       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	2498	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	64	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS STRING.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]STRING.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]STRING.OBJ;1
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	420	00:00.6	00:04.3
Source Analysis	305	00:01.8	00:06.8
Source Listing	14	00:00.8	00:03.1
Tree Construction	194	00:00.5	00:01.9
Flow Analysis	77	00:00.4	00:01.7
Value Propagation	10	00:00.1	00:00.3
Profit Analysis	34	00:00.2	00:01.3
Context Analysis	194	00:03.1	00:10.7
Name Packing	6	00:00.1	00:00.1
Code Selection	100	00:00.4	00:02.2
Final	63	00:00.7	00:01.8
TOTAL	1421	00:08.5	00:34.3

COMPILATION STATISTICS

```
  CPU Time:       00:08.5         (3594 Lines/Minute)
```

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

00001    C  0  0 { Title: SyntaxInterpreter
00002    C  0  0
00003    C  0  0   Purpose: Get operator input and isolate words, numbers and punctuation
00004    C  0  0            so that it can be intelligently interpreted
00005    C  0  0
00006    C  0  0   Author: William A. Chapman      Date: July 29, 1986
00007    C  0  0
00008    C  0  0   Inputs: Operator input
00009    C  0  0
00010    C  0  0   Outputs: Keyword characteristics are provided for valid tokens
00011    C  0  0            including numeric values
00012    C  0  0
00013    C  0  0   Procedures Invoked: (Parser)
00014    C  0  0            (KeywordLookUp)
00015    C  0  0            (Convert)
00016    C  0  0            (UpperCase)
00017    C  0  0 }
00018       0  0 module SyntaxInterpreter( 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';
00123       0  0
00124       0  0      UndefinedValue = %X'00A5';
```

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

00126       0  0 type
00127       0  0     %include  [-]Type.defn/nolist'
00167   C   0  0 {   ZeroOne = -1..1;}
00168       0  0
00169       0  0     %include  [-]DebugType.defn/list'
00170 I     0  0     NameString = packed array[1..ShortStringLength] of char;
00171 I     0  0
00172 I     0  0     KeywordCharacteristics = record
00173 I     0  0   Name: NameString;
00174 I     0  0   Class: char;
00175 I     0  0   Width: ZeroOne;
00176 I     0  0   ID: integer;
00177 I     0  0   Value: unsigned
00178 I     0  0     end;
00179 I     0  0
00180 I     0  0     InputPointerRange = integer;
00181 I     0  0     InputLine = packed array [1..InputLineLength] of char;
00182 I     0  0
00183 I     0  0     ShortString = packed array [1..ShortStringLength] of char;
00184 I     0  0
00185 I     0  0     DataArrayType = array [StartOfData..DataArraySize] of unsigned;
00186 I     0  0
00187 I     0  0     BreakRecord = record
00188 I     0  0   Encountered: boolean;
00189 I     0  0   Activated: boolean;
00190 I     0  0   Segment: unsigned;
00191 I     0  0   Address: unsigned;
00192 I     0  0   PhysicalAddress: unsigned;
00193 I     0  0   Code: unsigned
00194 I     0  0     end;
00195 I     0  0
00196 I     0  0     BreakPointTableType = array [FirstBreakpoint..LastBreakpoint] of BreakRecord;
```

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

00198       0  0 var
00199       0  0     Punctuation: [external] Characters;
00200       0  0     Numerals: [external] Numbers;
00201       0  0     HexNumerals: [external] LettersandNumbers;
00202       0  0
00203       0  0     KeywordLUT: [external] array[FirstKeywordEntry..LastKeywordEntry]
00204       0  0   of KeywordCharacteristics;
00205       0  0
00206       0  0     UndefinedKeyword: KeywordCharacteristics;
00207       0  0
00208       0  0     Token: ShortString;
00209       0  0     ValidInput: boolean;
00210       0  0
00211       0  0     Success: boolean;
00212       0  0     ConvertValue: unsigned;
00213       0  0
00214       0  0
00215       0  0
00216       1  0 [global] procedure SyntaxInterpreter( var Keyword: KeywordCharacteristics;
00217       1  0           var OperatorInput: InputLine;
00218       1  0           var InputPointer: InputPointerRange;
00219       1  0           var InputLength: InputPointerRange;
00220       1  0           var EOLNFlag: boolean);
```

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

00222   C  1  0 { Title: Parser
00223   C  1  0
00224   C  1  0   Purpose: Isolate words, numbers or punctuation marks in the
00225   C  1  0            input string which are separated by punctuation characters
00226   C  1  0
00227   C  1  0   Author: William A. Chapman      Date: July 29, 1986
00228   C  1  0
00229   C  1  0   Inputs: Input from operator
00230   C  1  0            Pointer to current position in the input string
00231   C  1  0            Length of the input string
00232   C  1  0
00233   C  1  0   Outputs: A TOKEN is returned (word, number or punctuation mark)
00234   C  1  0            Pointer is updated to point to the next character
00235   C  1  0
00236   C  1  0   Procedures Invoked: none
00237   C  1  0 }
00238      2  0 procedure Parser( var Token: ShortString;
00239      2  0                   OperatorInput: InputLine;
00240      2  0                   var Pointer: InputPointerRange;
00241      2  0                   InputLength: InputPointerRange;
00242      2  0                   var EndOfLineFlag: boolean);
00243      2  0
00244      2  0 var
00245      2  0     TokenIndex: integer;
00246      2  0     Done: boolean;
```

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

00248       2  1 begin
00249       2  1     for TokenIndex := 1 to ShortStringLength do Token[ TokenIndex] := Blank;
00250       2  1     TokenIndex := 1;
00251       2  1     Done := false;
00252       2  2     while (not Done) do begin
00253       2  2   while (( Pointer <= InputLength) and
00254       2  3       ( OperatorInput[ Pointer] = Blank)) do begin
00255       2  3       Pointer := Pointer + 1;
00256       2  2   end; {while blank}
00257       2  2
00258       2  2   if (( Pointer <= InputLength) and
00259       2  3       ( OperatorInput[ Pointer] in Punctuation)) then begin
00260       2  3       Token[ TokenIndex] := OperatorInput[ Pointer];
00261       2  3       TokenIndex := TokenIndex + 1;
00262       2  3       Pointer := Pointer + 1;
00263       2  3       Done := true;
00264       2  3   end {then begin}
00265       2  3
00266       2  3   else begin
00267       2  3       while (( Pointer <= InputLength) and
00268       2  4           not( OperatorInput[ Pointer] in Punctuation)) do begin
00269       2  4           Token[ TokenIndex] := OperatorInput[ Pointer];
00270       2  4           TokenIndex := TokenIndex + 1;
00271       2  4           Pointer := Pointer + 1;
00272       2  3       end; {while not in Punctuation}
00273       2  3       Done := true;
00274       2  2   end; {else begin}
00275       2  2
00276       2  3   if ( Pointer >= InputLength) then begin
00277       2  3       EndOfLineFlag := true;
00278       2  3       Pointer := EndofLine;
00279       2  3       Done := true;
00280       2  3   end {if Pointer > EndofLine}
00281       2  2   else EndOfLineFlag := false;
00282       2  2
00283       2  1     end; {while (not Done)}
00284       1  0 end; {Parser}
```

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

00286       1  0     %include 'UpperCaseString.subpas'
00287 I C   1  0 { Title: UpperCase
00288 I C   1  0
00289 I C   1  0   Purpose: Convert lowercase letters to uppercase
00290 I C   1  0
00291 I C   1  0   Author: William A. Chapman      Date: July 28, 1986
00292 I C   1  0
00293 I C   1  0   Inputs: A string is input (max ShortStringLength characters)
00294 I C   1  0
00295 I C   1  0   Outputs: All lowercase letters are converted to uppercase
00296 I C   1  0
00297 I C   1  0   Procedures Invoked: none
00298 I C   1  0 }
00299 I     2  0 procedure UpperCase( var SourceString: ShortString);
00300 I     2  0
00301 I     2  0 const
00302 I     2  0     StringStart = 1;
00303 I     2  0
00304 I     2  0 var
00305 I     2  0     Loop: integer;
00306 I     2  0     Offset: integer;
00307 I     2  0
00308 I     2  1 begin
00309 I     2  1     Offset := ord( 'a') - ord( 'A');
00310 I     2  1     for Loop := StringStart to ShortStringLength do
00311 I     2  1   if ((( ord( SourceString[Loop]) - ord( 'a')) >= 0 ) and
00312 I     2  1       (( ord( SourceString[Loop]) - ord( 'z')) <= 0 ))
00313 I     2  1   then SourceString[Loop] := chr( ord( SourceString[ Loop]) - Offset);
00314 I     1  0 end;
```

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

00316   C  1  0 { Title: Convert
00317   C  1  0
00318   C  1  0   Purpose: Convert string of numeric digits to an unsigned number
00319   C  1  0
00320   C  1  0   Author: William A. Chapman      Date: August 7, 1986
00321   C  1  0
00322   C  1  0   Inputs: ShortString of hex numerals
00323   C  1  0
00324   C  1  0   Outputs: Unsigned number
00325   C  1  0
00326   C  1  0   Procedures Invoked: none
00327   C  1  0 }
00328      2  0 procedure Convert( var CVTSuccess: boolean;
00329      2  0   var CVTValue: unsigned;
00330      2  0   CVTString: ShortString);
00331      2  0
00332      2  0 const
00333      2  0     CVTStringLength = 10;
00334      2  0
00335      2  0 var
00336      2  0     CVTPointer: integer;
00337      2  0     Temp: integer;
00338      2  0     Loop: integer;
00339      2  0     NumeralString: packed array [1..CVTStringLength] of char;
00340      2  0
00341      2  0
00342      2  1 begin
00343      2  1     CVTSuccess := true;
00344      2  1     for Loop := 1 to CVTStringLength do NumeralString[ Loop] := Blank;
00345      2  1     for Loop := 1 to ShortStringLength do
00346      2  1   NumeralString[ Loop] := CVTString[ Loop];
00347      2  1
00348      2  1     CVTValue := 0;
00349      2  1     CVTPointer := 1;
00350      2  1     while (( CVTPointer <= ShortStringLength) and
00351      2  2   ( NumeralString[ CVTPointer] = Blank)) do begin
00352      2  2   CVTPointer := CVTPointer + 1;
00353      2  1     end; {while blank}
00354      2  1
00355      2  1     while (( CVTPointer <= ShortStringLength) and
00356      2  2   ( NumeralString[ CVTPointer] in HexNumerals)) do begin
00357      2  2   Temp := ord( NumeralString[ CVTPointer]) - ord('0');
00358      2  2   if (Temp > 10) then Temp := Temp + ord('0') - ord('A') + 10;
00359      2  2   if (Temp > 15) then Temp := Temp + ord('A') - ord('a');
```

```
00360     2  3   if ((Temp < 0) or (Temp > 15)) then begin
00361     2  3       CVTSuccess := false;
00362     2  3       CVTValue := 0;
00363     2  3       Temp := 0;
00364     2  3       CVTPointer := ShortStringLength + 1;
00365     2  3   end {invalid character}
00366     2  2   else CVTValue := (CVTValue * 16) + Temp;
00367     2  2   CVTPointer := CVTPointer + 1;
00368     2  1     end;
00369     2  1
00370     2  1     while (( CVTPointer <= ShortStringLength) and
```

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

00371       2  2   ( NumeralString[ CVTPointer] = Blank)) do begin
00372       2  2   CVTPointer := CVTPointer + 1;
00373       2  1     end; {while blank]
00374       1  0 end;
```

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

00376   C  1  0 { Title: KeywordLookUp
00377   C  1  0
00378   C  1  0   Purpose: Find the keyword in the KeywordLUT and return its
00379   C  1  0            pertinent characteristics
00380   C  1  0
00381   C  1  0   Author: William A. Chapman      Date: July 29, 1986
00382   C  1  0
00383   C  1  0   Inputs: Keyword Look Up Table
00384   C  1  0           Token to look up
00385   C  1  0
00386   C  1  0   Outputs: Keyword characteristics
00387   C  1  0            Status of look up effort
00388   C  1  0
00389   C  1  0   Procedures Invoked: UpperCase
00390   C  1  0            Convert
00391   C  1  0 }
00392      2  0 procedure KeywordLookUp( var KeywordValue: KeywordCharacteristics;
00393      2  0           Token: ShortString;
00394      2  0           var GoodInput: boolean);
00395      2  0
00396      2  0 var
00397      2  0     Pattern: ShortString;
00398      2  0     Loop: integer;
00399      2  0     Found: boolean;
00400      2  0     Done: boolean;
```

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

00402      2  1 begin
00403      2  1     Pattern := substr( Token, 1, 3);
00404      2  1     UpperCase( Pattern);
00405      2  1     Found := false;
00406      2  1     Done := false;
00407      2  1
00408      2  2     if ( Token[1]  in Numerals) then begin
00409      2  2   Convert( Success, ConvertValue, Token);
00410      2  3   with KeywordValue do begin
00411      2  3       Name := Token;
00412      2  3       Class := NumericClass;
00413      2  3       Width := UndefinedWidth;
00414      2  3       ID := UndefinedID;
00415      2  3       Value := ConvertValue;
00416      2  2   end;
00417      2  2   Found := true;
00418      2  2   GoodInput := true;
00419      2  2   Done := true;
00420      2  1     end;
00421      2  1
00422      2  1     Loop := FirstKeywordEntry;
00423      2  2     while ( not Done) do begin
00424      2  2   if (substr( KeywordLUT[ Loop].Name, 1, 3) = substr( Pattern, 1, 3))
00425      2  3   then begin
00426      2  3       KeywordValue := KeywordLUT[ Loop];
00427      2  3       Found := true;
00428      2  3       GoodInput := true;
00429      2  3       Done := true;
00430      2  3   end
00431      2  3   else begin
00432      2  3       Loop := Loop + 1;
00433      2  3       if ( Loop > LastKeywordEntry) then Done := true;
00434      2  2   end;
00435      2  1     end;
00436      2  2     if (not Found) then begin
00437      2  2   Convert( Success, ConvertValue, Token);
00438      2  3   if (Success = true) then begin
00439      2  4       with KeywordValue do begin
00440      2  4           Name := Token;
00441      2  4           Class := NumericClass;
00442      2  4           Width := UndefinedWidth;
00443      2  4           ID := UndefinedID;
00444      2  4           Value := ConvertValue;
00445      2  3       end;
```

```
00446      2  3        GoodInput := true;
00447      2  3    end
00448      2  3    else begin
00449      2  3        writeln( 'Invalid keyword: "', Token: ShortStringLength, '"');
00450      2  3        GoodInput := false;
00451      2  3        KeywordValue := UndefinedKeyword;
00452      2  2    end;
00453      2  1      end;
00454      1  0 end;
```

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

00456       1  1 begin
00457       1  1     Parser( Token, OperatorInput, InputPointer, InputLength, EOLNFlag);
00458       1  1     KeywordLookUp( Keyword, Token, ValidInput);
00459       0  0 end;
00460       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	1094	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	40	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
  PAS/LIS SYNTAXINTERPRETER.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]SYNTAXINTERPRETER.LIS;1
  /OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]SYNTAXINTERPRETER.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	00:04.7
Source Analysis	361	00:01.3	00:03.8
Source Listing	36	00:00.7	00:04.0
Tree Construction	139	00:00.4	00:01.0
Flow Analysis	46	00:00.3	00:00.5
Value Propagation	11	00:00.0	00:00.0
Profit Analysis	94	00:00.2	00:00.7
Context Analysis	306	00:02.3	00:06.5
Name Packing	7	00:00.1	00:00.2
Code Selection	125	00:00.3	00:01.0
Final	62	00:00.5	00:01.2
TOTAL	1595	00:06.5	00:23.6

COMPILATION STATISTICS

```
  CPU Time:       00:06.5         (4233 Lines/Minute)
```

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

00001   C  0  0 { Title: UnpackHex
00002   C  0  0
00003   C  0  0   Purpose: Convert a hex number to decimal, and check for illegal
00004   C  0  0            decimal characters
00005   C  0  0
00006   C  0  0   Author: William A. Chapman      Date: September 16, 1986
00007   C  0  0
00008   C  0  0   Inputs: Unsigned hexadecimal input
00009   C  0  0            Sign of input
00010   C  0  0
00011   C  0  0   Outputs: Decimal integer representation of number
00012   C  0  0
00013   C  0  0   Procedures Invoked: none
00014   C  0  0 ]
00015      0  0 module UnpackHex( input, output);
00016      0  0
00017      0  0 const
00018      0  0     %include '[-]DebugConst.defn/list'
00019 I    0  0     FirstKeywordEntry = 0;
00020 I    0  0     LastKeywordEntry  = 100;
00021 I    0  0     UndefinedName = 'Qqqqqqqq';
00022 I    0  0     UndefinedClass = 'z';
00023 I    0  0     UndefinedWidth = -1;
00024 I    0  0     UndefinedID = -2;
00025 I    0  0     PageSize = 55;
00026 I    0  0     FF = 12;
00027 I    0  0     MinID = -1;
00028 I    0  0     MaxID = 25;
00029 I    0  0     MinWidth = 0;
00030 I    0  0     MaxWidth = 3;
00031 I    0  0     SegmentOffset = 16;
00032 I    0  0
00033 I    0  0     BlankClass       =   ;
00034 I    0  0     BreakpointClass  = 'B';
00035 I    0  0     ErrorClass       = 'E';
00036 I    0  0     FlagClass        = 'F';
00037 I    0  0     InputOutputClass = 'I';
00038 I    0  0     MemoryClass      = 'M';
00039 I    0  0     NumericClass     = 'N';
00040 I    0  0     OperatorClass    = 'O';
00041 I    0  0     QualifierClass   = 'Q';
00042 I    0  0     RegisterClass    = 'R';
00043 I    0  0     StackClass       = 'S';
00044 I    0  0     TrueFalseClass   = 'T';
00045 I    0  0     UtilityClass     = 'U';
```

```
00046 I    0  0     ExecuteClass     = 'X';
00047 I    0  0
00048 I    0  0     Blank  =    ;
00049 I    0  0     Elipse =  .';
00050 I    0  0
00051 I    0  0     StartOfData = 1;
00052 I    0  0     DataArraySize = 50;
00053 I    0  0
00054 I    0  0     EndofLine = 0;
00055 I    0  0     StartofLine = 1;
```

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

00056 I     0  0
00057 I     0  0      ShortStringLength = 8;
00058 I     0  0      InputLineLength = 120;
00059 I     0  0
00060 I     0  0      BooleanID  = 1;
00061 I     0  0      ByteID     = 2;
00062 I     0  0      IntegerID  = 3;
00063 I     0  0      PointerID  = 4;
00064 I     0  0      SIntegerID = 5;
00065 I     0  0      WordID     = 6;
00066 I     0  0      LengthID   = 3;
00067 I     0  0      ToID       = 5;
00068 I     0  0
00069 I     0  0      AssignmentID = 1;
00070 I     0  0      ColonID      = 2;
00071 I     0  0      PlusID       = 3;
00072 I     0  0      MinusID      = 4;
00073 I     0  0      CommaID      = 5;
00074 I     0  0
00075 I     0  0      NullEntries = 0;
00076 I     0  0
00077 I     0  0      AddressState   = 'A';
00078 I     0  0      BooleanState   = 'B';
00079 I     0  0      ErrorState     = 'E';
00080 I     0  0      HexState       = 'H';
00081 I     0  0      IntegerState   = 'I';
00082 I     0  0      SegmentState   = 'S';
00083 I     0  0      SeparatorState = ',';
00084 I     0  0      UnaryState     = '-';
00085 I     0  0      ColonState     = ':';
00086 I     0  0
00087 I     0  0      NegativeOperator = -1;
00088 I     0  0      PositiveOperator = +1;
00089 I     0  0
00090 I     0  0      NullSegmentValue = 0;
00091 I     0  0      NullAddressValue = 0;
00092 I     0  0
00093 I     0  0      FirstBreakpoint  =  0;
00094 I     0  0      LastBreakpoint   = 15;
00095 I     0  0      AllBPIDs         = 16;
00096 I     0  0      Activate         = true;
00097 I     0  0      DeActivate       = false;
00098 I     0  0      BreakpointOpcode = %X'CC';
```

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

00100       0  0 var
00101       0  0     ConvertValue: integer;
00102       0  0     Loop: integer;
00103       0  0     ConvertString: packed array[1..ShortStringLength] of char;
00104       0  0     ConvertTemp: integer;
00105       0  0
00106       0  0
00107       1  0 [global] function UnpackHex( PackedHex: unsigned;
00108       1  0           SignValue: integer): integer;
00109       1  0
00110       1  0
00111       1  0
00112       1  1 begin
00113       1  1     ConvertString := hex( PackedHex, ShortStringLength, ShortStringLength);
00114       1  1     ConvertValue := 0;
00115       1  2     for Loop := 1 to ShortStringLength do begin
00116       1  2   ConvertTemp := ord( ConvertString[ Loop]) - ord('0');
00117       1  3   if (ConvertTemp > 9) then begin
00118       1  3       writeln( 'Invalid integer character: "', ConvertString[ Loop], '"');
00119       1  3       ConvertTemp := 0;
00120       1  2   end; {ConvertTemp > 9}
00121       1  2   ConvertValue := (ConvertValue * 10) + ConvertTemp;
00122       1  1     end; {for Loop}
00123       1  1     UnpackHex := ConvertValue * SignValue;
00124       0  0 end;
00125       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	251	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	20	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

```
PAS/LIS UNPACKHEX.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]UNPACKHEX.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]UNPACKHEX.OBJ;1
/NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	425	00:00.5	00:02.7
Source Analysis	221	00:00.4	00:01.4
Source Listing	16	00:00.3	00:01.1
Tree Construction	95	00:00.1	00:00.3
Flow Analysis	45	00:00.1	00:00.1
Value Propagation	11	00:00.0	00:00.0
Profit Analysis	104	00:00.0	00:00.4
Context Analysis	116	00:00.3	00:00.3
Name Packing	6	00:00.0	00:00.0
Code Selection	81	00:00.1	00:00.1
Final	74	00:00.1	00:00.5
TOTAL	1198	00:01.8	00:07.0

COMPILATION STATISTICS

CPU Time: 00:01.8 (4190 Lines/Minute)

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

00001    C  0  0 { Title: Update Flags
00002    C  0  0
00003    C  0  0   Purpose: A collection of functions to update the Processor Status Flags
00004    C  0  0
00005    C  0  0   Author: William A. Chapman      Date: February 8, 1986
00006    C  0  0
00007    C  0  0   Inputs: Width
00008    C  0  0           Addend1 for addition only (for carry, overflow, auxcarry)
00009    C  0  0           Addend2 for addition only (for carry, overflow, auxcarry)
00010    C  0  0           Minuend for subtraction only (for carry, overflow, auxcarry)
00011    C  0  0           Subtrahend for subtraction only (for carry, overflow, auxcarry)
00012    C  0  0           Result
00013    C  0  0
00014    C  0  0   Outputs: True or False value to be loaded into the appropriate flag
00015    C  0  0
00016    C  0  0   Procedures Invoked: MaskShiftRight
00017    C  0  0 }
00018       0  0 module UpdateFlags(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;
00041       0  0     %include '[-]Flags.defn/list'
00042 I     0  0     SetHigh = true;
00043 I     0  0     Clear = false;
00044 I     0  0
00045 I     0  0     CarryFlag     = %B'000000000001';
```

```
00046 I     0  0     ParityFlag    = %B'000000000100';
00047 I     0  0     AuxCarryFlag  = %B'000000010000';
00048 I     0  0     ZeroFlag      = %B'000001000000';
00049 I     0  0     SignFlag      = %B'000010000000';
00050 I     0  0     TrapFlag      = %B'000100000000';
00051 I     0  0     InterruptFlag = %B'001000000000';
00052 I     0  0     DirectionFlag = %B'010000000000';
00053 I     0  0     OverflowFlag  = %B'100000000000';
00054       0  0
00055       0  0     ByteSignMask = %X'80';
```

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

00056       0  0     WordSignMask = %X'8000';
00057       0  0
00058       0  0     ByteCarryMask = %X'80';
00059       0  0     WordCarryMask = %X'8000';
00060       0  0     AuxCarryMask  = %X'8';
00061       0  0
00062       0  0     Positive = false;
00063       0  0     Negative = true;
00064       0  0
00065       0  0     ParityMask = 1;
00066       0  0     Limit = 8;
00067       0  0     RotateRightOne = 2;
```

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

00069       0  0 type
00070       0  0     %include  [-]FlagType.defn/list'
00071 I     0  0     FlagType = record                          {define processor flags}
00072 I     0  0   Carry: boolean;
00073 I     0  0   Parity: boolean;
00074 I     0  0   AuxCarry: boolean;
00075 I     0  0   Zero: boolean;
00076 I     0  0   Sign: boolean;
00077 I     0  0   Trap: boolean;
00078 I     0  0   Interrupt: boolean;
00079 I     0  0   Direction: boolean;
00080 I     0  0   Overflow: boolean
00081 I     0  0     end;
00082       0  0
00083       0  0 var
00084       0  0     Flags: [external] FlagType;
00085       0  0
00086       0  0     DataMask: unsigned;
00087       0  0     SignMask: unsigned;
00088       0  0     SignAddend1: unsigned;
00089       0  0     SignAddend2: unsigned;
00090       0  0     SignMinuend: unsigned;
00091       0  0     SignSubtrahend: unsigned;
00092       0  0
00093       0  0     CarryMask: unsigned;
00094       0  0     CarryAddend1: unsigned;
00095       0  0     CarryAddend2: unsigned;
00096       0  0     CarryMinuend: unsigned;
00097       0  0     CarrySubtrahend: unsigned;
00098       0  0     CarryResult: unsigned;
00099       0  0
00100       0  0     AuxCarryAddend1: unsigned;
00101       0  0     AuxCarryAddend2: unsigned;
00102       0  0     AuxCarryMinuend: unsigned;
00103       0  0     AuxCarrySubtrahend: unsigned;
00104       0  0     AuxCarryResult: unsigned;
00105       0  0
00106       0  0     Counter: integer;
00107       0  0     ParityCount: integer;
```

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

00109       1  0 function MaskShiftRight(MSRValue: unsigned;
00110       1  0           MSRMask: unsigned;
00111       0  0           MSRDivisor: integer): unsigned; external;
00112       0  0
00113       0  0
00114       0  0
00115       0  0
00116       1  0 [global] function UpdateZeroFlag(Width: integer;
00117       1  0           Result: unsigned): boolean;
00118       1  0
00119       1  1 begin
00120       1  1     if (Width = EightBits) then DataMask := Low8Bits
00121       1  1     else DataMask := All16Bits;
00122       1  1
00123       1  1     if (uand(DataMask, Result) = 0) then UpdateZeroFlag := true
00124       1  1     else UpdateZeroFlag := false;
00125       0  0 end;
00126       0  0
00127       0  0
00128       0  0
00129       1  0 [global] function UpdateSignFlag(Width: integer;
00130       1  0           Result: unsigned): boolean;
00131       1  0
00132       1  1 begin
00133       1  1     if (Width = EightBits) then SignMask := ByteSignMask
00134       1  1     else SignMask := WordSignMask;
00135       1  1
00136       1  1     if (uand(SignMask, Result) = 0) then UpdateSignFlag := Positive
00137       1  1     else UpdateSignFlag := Negative;
00138       0  0 end;
00139       0  0
00140       0  0
00141       0  0
00142       1  0 [global] function UpdateParityFlag(Width: integer;
00143       1  0           Result: unsigned): boolean;
00144       1  0
00145       1  0
00146   C   1  0 { NOTE: Result must NOT be var
00147   C   1  0 or it will screw up the calling procedure's value]
00148       1  0
00149       1  1 begin
00150       1  1     ParityCount := 0;
00151       1  2     for Counter := 1 to Limit do begin
00152       1  2   ParityCount := ParityCount + int(uand(Result, ParityMask));
```

```
00153     1  2   Result := MaskShiftRight(Result, Low8Bits, RotateRightOne);
00154     1  1     end; {for Counter}
00155     1  1     UpdateParityFlag := not( odd(ParityCount));
00156     0  0 end;
```

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

00158      1  0 [global] function UpdateOverflowFlag_Add(Width: integer;
00159      1  0           Addend1: unsigned;
00160      1  0           Addend2: unsigned;
00161      1  0           Result: unsigned): boolean;
00162      1  0
00163      1  0
00164      1  1 begin
00165      1  1     if (Width = EightBits) then SignMask := ByteSignMask
00166      1  1     else SignMask := WordSignMask;
00167      1  1
00168      1  1     SignAddend1 := uand(Addend1, SignMask);
00169      1  1     SignAddend2 := uand(Addend2, SignMask);
00170      1  1
00171      1  2     if (SignAddend1 = SignAddend2) then begin
00172      1  2   if (SignAddend1 <> uand(SignMask, Result))
00173      1  2       then UpdateOverflowFlag_Add := true
00174      1  2   else UpdateOverflowFlag_Add := false;
00175      1  2     end
00176      1  1     else UpdateOverflowFlag_Add := false;
00177      0  0 end;
00178      0  0
00179      0  0
00180      0  0
00181      0  0
00182      1  0 [global] function UpdateOverflowFlag_Sub(Width: integer;
00183      1  0           Minuend: unsigned;
00184      1  0           Subtrahend: unsigned;
00185      1  0           Result: unsigned): boolean;
00186      1  0
00187      1  1 begin
00188      1  1     if (Width = EightBits) then SignMask := ByteSignMask
00189      1  1     else SignMask := WordSignMask;
00190      1  1
00191      1  1     SignMinuend := uand(Minuend, SignMask);
00192      1  1     SignSubtrahend := uand(Subtrahend, SignMask);
00193      1  1
00194      1  2     if (SignMinuend <> SignSubtrahend) then begin
00195      1  2   if (SignMinuend <> uand(SignMask, Result))
00196      1  2       then UpdateOverflowFlag_Sub := true
00197      1  2   else UpdateOverflowFlag_Sub := false;
00198      1  2     end
00199      1  1     else UpdateOverflowFlag_Sub := false;
00200      0  0 end;
```

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

00202      1  0 [global] function UpdateCarryFlag_Add(Width: integer;
00203      1  0           Addend1: unsigned;
00204      1  0           Addend2: unsigned;
00205      1  0           Result: unsigned): boolean;
00206      1  0
00207      1  0
00208      1  1 begin
00209      1  1     if (Width = EightBits) then CarryMask := ByteCarryMask
00210      1  1     else CarryMask := WordCarryMask;
00211      1  1
00212      1  1     CarryAddend1 := uand(Addend1, CarryMask);
00213      1  1     CarryAddend2 := uand(Addend2, CarryMask);
00214      1  1     CarryResult  := uand(Result, CarryMask);
00215      1  1
00216      1  1     if ((uand(CarryAddend1, CarryAddend2) <> 0) or
00217      1  1   (uand(CarryAddend2, unot(CarryResult)) <> 0) or
00218      1  1   (uand(CarryAddend1, unot(CarryResult)) <> 0))
00219      1  1       then UpdateCarryFlag_Add := true
00220      1  1     else UpdateCarryFlag_Add := false;
00221      0  0 end;
00222      0  0
00223      0  0
00224      0  0
00225      0  0
00226      0  0
00227      0  0
00228      1  0 [global] function UpdateCarryFlag_Sub(Width: integer;
00229      1  0           Minuend: unsigned;
00230      1  0           Subtrahend: unsigned;
00231      1  0           Result: unsigned): boolean;
00232      1  0
00233      1  0
00234      1  1 begin
00235      1  1     if (Width = EightBits) then CarryMask := ByteCarryMask
00236      1  1     else CarryMask := WordCarryMask;
00237      1  1
00238      1  1     CarryMinuend := uand(Minuend, CarryMask);
00239      1  1     CarrySubtrahend := uand(Subtrahend, CarryMask);
00240      1  1     CarryResult  := uand(Result, CarryMask);
00241      1  1
00242      1  1     if ((uand(unot(CarryMinuend), CarrySubtrahend) <> 0) or
00243      1  1   (uand(CarrySubtrahend, CarryResult) <> 0) or
00244      1  1   (uand(unot(CarryMinuend), CarryResult) <> 0))
00245      1  1       then UpdateCarryFlag_Sub := true
```

```
00246     1  1     else UpdateCarryFlag_Sub := false;
00247     0  0 end;
00248     0  0
00249     0  0
00250     0  0
```

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

00252      1  0 [global] function UpdateAuxCarryFlag_Add(Width: integer;
00253      1  0           Addend1: unsigned;
00254      1  0           Addend2: unsigned;
00255      1  0           Result: unsigned): boolean;
00256      1  0
00257      1  0
00258      1  1 begin
00259      1  1     AuxCarryAddend1 := uand(Addend1, AuxCarryMask);
00260      1  1     AuxCarryAddend2 := uand(Addend2, AuxCarryMask);
00261      1  1     AuxCarryResult  := uand(Result, AuxCarryMask);
00262      1  1
00263      1  1     if ((uand(AuxCarryAddend1, AuxCarryAddend2) <> 0) or
00264      1  1   (uand(AuxCarryAddend2, unot(AuxCarryResult)) <> 0) or
00265      1  1   (uand(AuxCarryAddend1, unot(AuxCarryResult)) <> 0))
00266      1  1       then UpdateAuxCarryFlag_Add := true
00267      1  1     else UpdateAuxCarryFlag_Add := false;
00268      0  0 end;
00269      0  0
00270      0  0
00271      0  0
00272      0  0
00273      1  0 [global] function UpdateAuxCarryFlag_Sub(Width: integer;
00274      1  0           Minuend: unsigned;
00275      1  0           Subtrahend: unsigned;
00276      1  0           Result: unsigned): boolean;
00277      1  0
00278      1  0
00279      1  1 begin
00280      1  1     AuxCarryMinuend := uand(Minuend, AuxCarryMask);
00281      1  1     AuxCarrySubtrahend := uand(Subtrahend, AuxCarryMask);
00282      1  1     AuxCarryResult  := uand(Result, AuxCarryMask);
00283      1  1
00284      1  1     if ((uand(unot(AuxCarryMinuend), AuxCarrySubtrahend) <> 0) or
00285      1  1   (uand(AuxCarrySubtrahend, AuxCarryResult) <> 0) or
00286      1  1   (uand(unot(AuxCarryMinuend), AuxCarryResult) <> 0))
00287      1  1       then UpdateAuxCarryFlag_Sub := true
00288      1  1     else UpdateAuxCarryFlag_Sub := false;
00289      0  0 end;
00290      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	1066	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	76	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

PAS/LIS UPDATEFLAGS.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]UPDATEFLAGS.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]UPDATEFLAGS.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	00:01.3
Source Analysis	403	00:01.0	00:02.6
Source Listing	29	00:00.5	00:02.3
Tree Construction	111	00:00.2	00:00.9
Flow Analysis	43	00:00.3	00:00.6
Value Propagation	11	00:00.1	00:00.8
Profit Analysis	62	00:00.2	00:00.4
Context Analysis	197	00:02.2	00:06.3
Name Packing	7	00:00.1	00:00.5
Code Selection	109	00:00.4	00:01.1
Final	66	00:00.5	00:02.3
TOTAL	1447	00:05.9	00:19.3

COMPILATION STATISTICS

CPU Time: 00:05.9 (2969 Lines/Minute)