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. 1

Command Procedures

and

Source Code

ROCHESTER INSTITUTE OF TECHNOLOGY

BUILD PROCEDURES

Module Management System Procedures

BuildEM86.com

Emulator86.mms

IOHandling.mms

KeywordLUTGen.mms

OpcodeLUTGen.mms

Command Procedures

BuildEmulator86.com

CompileEmulator86.com

BuildIO86.com

BuildKeywordLUTGen86.com

BuildOpcodeLUTGen86.com

SOURCE CODE MODULES

Arithmetic

ComputeEA

ControlTransfer

DataTransfer

Debugger

DisplayBreakpoint

Emulator86

ExecuteInstruction

FetchCode

FetchCodeData

FetchInput

FetchInputLine

FetchInstruction

FetchMemory

FetchOperand

FetchRegPtr

FileExists

Functions

GetAddress

GetData

HexIn

InstallBreakpoints

Interrupt

IOMapgen

KeywordLUTGen

Loader

Logic

OpcodeLUTGen

ParseFilename

PopFlags

PopFromStack

PopIntersegRA

ProcessorControl

PushFlags

PushIntersegRA

PushOnStack

RemoveBreakpoints

ReviewIOData

RMDecoder

SearchForBreakpoint

StoreData

StoreMemory

StoreOutput

StoreRegPtr

String

SyntaxInterpreter

UnpackHex

UpdateFlags

Complete Build Procedure

```
!
! Inquire if PASCAL and FORTRAN should produce listings
!
$ inquire listing "Do you want a listing(s) ? [N]"
$ if (listing .eqs. "Y") then goto generate
$ pas = "pas/nolis/usage=unused"
$ for = "for/nolis"
$ goto procede
!
$ generate:
$ pas = "pas/lis/usage=unused"
$ for = "for/lis"
!
$ procede:
$ set verify
$ show symbol pas
$ show symbol for
$ !
$ ! Build the Emulator
$ !
$ mms/des=emulator86.mms emulator86.exe
$ !
$ ! Build the I/O Handling
$ !
$ mms/des=iohandling.mms iohandling.dummy
$ !
$ ! Build the Opcode LUT Generator
$ !
$ mms/des=opcodelutgen.mms opcodelutgen.exe
$ !
$ ! Build the Keyword LUT Generator
$ !
$ mms/des=keywordlutgen.mms keywordlutgen.exe
$ set noverify
$ exit
```

Emulator86.mms

```
emulator86.exe : emulator.olb( arithmetic.obj),-
    emulator.olb( computeea.obj),-
    emulator.olb( controltransfer.obj),-
    emulator.olb( datatransfer.obj),-
    emulator.olb( debugger.obj),-
    emulator.olb( displaybreakpoint.obj),-
    emulator.olb( executeinstruction.obj),-
    emulator.olb( fetchcode.obj),-
    emulator.olb( fetchcodedata.obj),-
    emulator.olb( fetchinput.obj),-
    emulator.olb( fetchinputline.obj),-
    emulator.olb( fetchinstruction.obj),-
    emulator.olb( fetchmemory.obj),-
    emulator.olb( fetchoperand.obj),-
    emulator.olb( fetchregptr.obj),-
    emulator.olb( fileexists.obj),-
    emulator.olb( functions.obj),-
    emulator.olb( getaddress.obj),-
    emulator.olb( getdata.obj),-
    emulator.olb( hexin.obj),-
    emulator.olb( installbreakpoints.obj),-
    emulator.olb( interrupt.obj),-
    emulator.olb( loader.obj)-
    emulator.olb( logic.obj),-
    emulator.olb( parsefilename.obj),-
    emulator.olb( popflags.obj),-
    emulator.olb( popfromstack.obj),-
    emulator.olb( popintersegra.obj),-
    emulator.olb( processorcontrol.obj),-
    emulator.olb( pushflags.obj),-
    emulator.olb( pushintersegra.obj),-
    emulator.olb( pushonstack.obj),-
    emulator.olb( removebreakpoints.obj),-
    emulator.olb( rmdecoder.obj),-
    emulator.olb( searchforbreakpoint.obj),-
    emulator.olb( storedata.obj),-
    emulator.olb( storememory.obj),-
    emulator.olb( storeoutput.obj),-
    emulator.olb( storeregptr.obj),-
    emulator.olb( string.obj),-
    emulator.olb( syntaxinterpreter.obj),-
    emulator.olb( unpackhex.obj),-
    emulator.olb( updateflags.obj),-
    emulator86.obj

    link emulator86, emulator.olb/lib
    purge/log emulator86.*
```

```
emulator86.obj : emulator86.pas,-
    [-]const.defn,-
    [-]ioconst.defn,-
    [-]type.defn,-
    [-]flagtype.defn,-
    [-]memorytype.defn,-
    [-]iotype.defn,-
    [-]varglobal.defn,-
    [-]regglobal.defn,-
    [-]iovarglobal.defn,-
    uppercaseletter.subpas

    pas/usage=unused emulator86.pas

emulator.olb( arithmetic.obj) : arithmetic.pas,-
    [-]const.defn,-
    [-]stopcode.defn,-
    [-]flags.defn,-
    [-]regid.defn,-
    [-]type.defn,-
    [-]flagtype.defn

    pas/usage=unused arithmetic.pas
    lib/replace emulator arithmetic
    purge/log arithmetic.*
    del/log arithmetic.obj;*

emulator.olb( computeea.obj) : computeea.pas,-
    [-]const.defn,-
    [-]stopcode.defn,-
    [-]regid.defn

    pas/usage=unused computeea.pas
    lib/replace emulator computeea
    purge/log computeea.*
    del/log computeea.obj;*

emulator.olb( controltransfer.obj) : -
    controltransfer.pas,-
    [-]const.defn,-
    [-]stopcode.defn,-
    [-]regid.defn,-
    [-]flags.defn,-
    [-]type.defn,-
    [-]flagtype.defn

    pas/usage=unused controltransfer.pas
    lib/replace emulator controltransfer
    purge/log controltransfer.*
    del/log controltransfer.obj;*
```

```
emulator.olb( datatransfer.obj) : datatransfer.pas,-     -
    [-]const.defn,-
    [-]stopcode.defn,-
    [-]flags.defn,-
    [-]regid.defn,-
    [-]type.defn,-
    [-]flagtype.defn

    pas/usage=unused datatransfer.pas
    lib/replace emulator datatransfer
    purge/log datatransfer.*
    del/log datatransfer.obj;*

emulator.olb( debugger.obj) : debugger.pas,-
    [-]const.defn,-
    [-]ioconst.defn,-
    [-]stopcode.defn,-
    [-]type.defn,-
    [-]iotype.defn,-
    [-]flags.defn,-
    [-]flagtype.defn,-
    [-]debugconst.defn,-
    [-]debugtype.defn,-
    [-]regid.defn,-
    [-]regexternal.defn,-
    [-]iovarexternal.defn,-
    get8bitsofdata.subpas,-
    put8bitsofdata.subpas,-
    searchforportaddress.subpas,-
    memoryclass.subpas,-
    stackclass.subpas,-
    registerclass.subpas,-
    flagclass.subpas,-
    utilityclass.subpas,-
    inputoutputclass.subpas,-
    breakpointclass.subpas,-
    executeclass.subpas

    pas/usage=unused debugger.pas
    lib/replace emulator debugger
    purge/log debugger.*
    purge/log get8bitsofdata.subpas
    purge/log put8bitsofdata.subpas
    purge/log searchforportaddress.subpas
    purge/log memoryclass.subpas
    purge/log stackclass.subpas
    purge/log registerclass.subpas
    purge/log flagclass.subpas
    purge/log utilityclass.subpas
    purge/log inputoutputclass.subpas
    purge/log breakpointclass.subpas
    purge/log executeclass.subpas
    del/log debugger.obj;*
```

```
emulator.olb( displaybreakpoint.obj) : -
    displaybreakpoint.pas,-
    [-]const.defn,-
    [-]debugconst.defn,-
    [-]type.defn,-
    [-]debugtype.defn

    pas/usage=unused displaybreakpoint.pas
    lib/replace emulator displaybreakpoint
    purge/log displaybreakpoint.*
    del/log displaybreakpoint.obj;*

emulator.olb( executeinstruction.obj) : -
    executeinstruction.pas,-
    [-]const.defn,-
    [-]lookuptableclass.defn,-
    [-]stopcode.defn,-
    [-]type.defn

    pas/usage=unused executeinstruction.pas
    lib/replace emulator executeinstruction
    purge/log executeinstruction.*
    del/log executeinstruction.obj;*

emulator.olb( fetchcode.obj) : fetchcode.pas,-
    [-]const.defn,-
    [-]stopcode.defn,-
    [-]type.defn

    pas/usage=unused fetchcode.pas
    lib/replace emulator fetchcode
    purge/log fetchcode.*
    del/log fetchcode.obj;*

emulator.olb( fetchcodedata.obj) : fetchcodedata.pas,-
    [-]const.defn,-
    [-]stopcode.defn,-
    [-]type.defn

    pas/usage=unused fetchcodedata.pas
    lib/replace emulator fetchcodedata
    purge/log fetchcodedata.*
    del/log fetchcodedata.obj;*
```

```
emulator.olb( fetchinput.obj) : fetchinput.pas,-
    [-]const.defn,-
    [-]byte.defn,-
    [-]ioconst.defn,-
    [-]stopcode.defn,-
    [-]type.defn,-
    [-]iotype.defn,-
    [-]iovarexternal.defn,-
    get8bitsofdata.subpas

    pas/usage=unused fetchinput.pas
    lib/replace emulator fetchinput
    purge/log get8bitsofdata.subpas
    purge/log fetchinput.*
    del/log fetchinput.obj;*

emulator.olb( fetchinputline.obj) : fetchinputline.pas,-
    [-]const.defn,-
    [-]debugconst.defn,-
    [-]type.defn,-
    [-]debugtype.defn

    pas/usage=unused fetchinputline.pas
    lib/replace emulator fetchinputline
    purge/log fetchinputline.*
    del/log fetchinputline.obj;*

emulator.olb( fetchinstruction.obj) : -
    fetchinstruction.pas,-
    [-]const.defn,-
    [-]stopcode.defn,-
    [-]type.defn,-
    [-]memorytype.defn,-
    [-]flagtype.defn-
    [-]varexternal.defn

    pas/usage=unused fetchinstruction.pas
    lib/replace emulator fetchinstruction
    purge/log fetchinstruction.*
    del/log fetchinstruction.obj;*

emulator.olb( fetchmemory.obj) : fetchmemory.pas,-
    [-]const.defn,-
    [-]stopcode.defn,-
    [-]type.defn,-
    [-]byte.defn,-
    [-]memorytype.defn

    pas/usage=unused fetchmemory.pas
    lib/replace emulator fetchmemory
    purge/log fetchmemory.*
    del/log fetchmemory.obj;*
```

```
emulator.olb( fetchoperand.obj) : fetchoperand.pas,-
    [-]const.defn,-
    [-]stopcode.defn

    pas/usage=unused fetchoperand.pas
    lib/replace emulator fetchoperand
    purge/log fetchoperand.*
    del/log fetchoperand.obj;*

emulator.olb( fetchregptr.obj) : fetchregptr.pas,-
    [-]const.defn,-
    [-]regid.defn,-
    [-]stopcode.defn,-
    [-]regexternal.defn

    pas/usage=unused fetchregptr.pas
    lib/replace emulator fetchregptr
    purge/log fetchregptr.*
    del/log fetchregptr.obj;*

emulator.olb( fileexists.obj) : fileexists.for

    for fileexists.for
    lib/replace emulator fileexists
    purge/log fileexists.*
    del/log fileexists.obj;*

emulator.olb( functions.obj) : functions.pas

    pas/usage=unused functions.pas
    lib/replace emulator functions
    purge/log functions.*
    del/log functions.obj;*

emulator.olb( getaddress.obj) : getaddress.pas,-
    [-]const.defn,-
    [-]debugconst.defn,-
    [-]type.defn,-
    [-]debugtype.defn

    pas/usage=unused getaddress.pas
    lib/replace emulator getaddress
    purge/log getaddress.*
    del/log getaddress.obj;*
```

```
emulator.olb( getdata.obj) : getdata.pas,-
    [-]const.defn,-
    [-]debugconst.defn,-
    [-]type.defn,-
    [-]debugtype.defn

    pas/usage=unused getdata.pas
    lib/replace emulator getdata
    purge/log getdata.*
    del/log getdata.obj;*

emulator.olb( hexin.obj) : hexin.pas

    pas/usage=unused hexin.pas
    lib/replace emulator hexin
    purge/log hexin.*
    del/log hexin.obj;*

emulator.olb( installbreakpoints.obj) : -
    installbreakpoints.pas,-
    [-]const.defn,-
    [-]debugconst.defn,-
    [-]type.defn,-
    [-]debugtype.defn

    pas/usage=unused installbreakpoints.pas
    lib/replace emulator installbreakpoints
    purge/log installbreakpoints.*
    del/log installbreakpoints.obj;*

emulator.olb( interrupt.obj) : interrupt.pas,-
    [-]const.defn,-
    [-]regid.defn,-
    [-]flagtype.defn,-
    [-]flags.defn

    pas/usage=unused interrupt.pas
    lib/replace emulator interrupt
    purge/log interrupt.*
    del/log interrupt.obj;*

emulator.olb( loader.obj) : loader.pas,-
    [-]const.defn,-
    [-]stopcode.defn,-
    [-]type.defn,-
    [-]byte.defn,-
    [-]regexternal.defn

    pas/usage=unused loader.pas
    lib/replace emulator loader
    purge/log loader.*
    del/log loader.obj;*
```

```
emulator.olb( logic.obj) : logic.pas,-
    [-]const.defn,-
    [-]stopcode.defn,-
    [-]flags.defn,-
    [-]regid.defn,-
    [-]type.defn,-
    [-]flagtype.defn

    pas/usage=unused logic.pas
    lib/replace emulator logic
    purge/log logic.*
    del/log logic.obj;*

emulator.olb( parsefilename.obj) : parsefilename.pas,-
    [-]const.defn,-
    [-]debugconst.defn,-
    [-]type.defn,-
    [-]debugtype.defn

    pas/usage=unused parsefilename.pas
    lib/replace emulator parsefilename
    purge/log parsefilename.*
    del/log parsefilename.obj;*

emulator.olb( popflags.obj) : popflags.pas,-
    [-]flags.defn,-
    [-]flagtype.defn

    pas/usage=unused popflags.pas
    lib/replace emulator popflags
    purge/log popflags.*
    del/log popflags.obj;*

emulator.olb( popfromstack.obj) : popfromstack.pas,-
    [-]regid.defn

    pas/usage=unused popfromstack.pas
    lib/replace emulator popfromstack
    purge/log popfromstack.*
    del/log popfromstack.obj;*

emulator.olb( popintersegra.obj) : popintersegra.pas,-
    [-]regid.defn

    pas/usage=unused popintersegra.pas
    lib/replace emulator popintersegra
    purge/log popintersegra.*
    del/log popintersegra.obj;*
```

```
emulator.olb( processorcontrol.obj) : -
    processorcontrol.pas,-
    [-]const.defn,-
    [-]stopcode.defn,-
    [-]flags.defn,-
    [-]type.defn,-
    [-]flagtype.defn

    pas/usage=unused processorcontrol.pas
    lib/replace emulator processorcontrol
    purge/log processorcontrol.*
    del/log processorcontrol.obj;*

emulator.olb( pushflags.obj) : pushflags.pas,-
    [-]flags.defn,-
    [-]flagtype.defn

    pas/usage=unused pushflags.pas
    lib/replace emulator pushflags
    purge/log pushflags.*
    del/log pushflags.obj;*

emulator.olb( pushintersegra.obj) : pushintersegra.pas,-  -  -
    [-]regid.defn

    pas/usage=unused pushintersegra.pas
    lib/replace emulator pushintersegra
    purge/log pushintersegra.*
    del/log pushintersegra.obj;*

emulator.olb( pushonstack.obj) : pushonstack.pas,-
    [-]regid.defn

    pas/usage=unused pushonstack.pas
    lib/replace emulator pushonstack
    purge/log pushonstack.*
    del/log pushonstack.obj;*

emulator.olb( removebreakpoints.obj) : -
    removebreakpoints.pas,-
    [-]const.defn,-
    [-]debugconst.defn,-
    [-]type.defn,-
    [-]debugtype.defn

    pas/usage=unused removebreakpoints.pas
    lib/replace emulator removebreakpoints
    purge/log removebreakpoints.*
    del/log removebreakpoints.obj;*
```

```
emulator.olb( rmdecoder.obj) : rmdecoder.pas,-
    [-]const.defn,-
    [-]stopcode.defn,-
    [-]regid.defn

    pas/usage=unused rmdecoder.pas
    lib/replace emulator rmdecoder
    purge/log rmdecoder.*
    del/log rmdecoder.obj;*

emulator.olb( searchforbreakpoint.obj) : -
    searchforbreakpoint.pas,-
    [-]const.defn,-
    [-]debugconst.defn,-
    [-]type.defn,-
    [-]debugtype.defn

    pas/usage=unused searchforbreakpoint.pas
    lib/replace emulator searchforbreakpoint
    purge/log searchforbreakpoint.*
    del/log searchforbreakpoint.obj;*

emulator.olb( storedata.obj) : storedata.pas,-
    [-]const.defn,-
    [-]stopcode.defn,-
    [-]stopcode.defn

    pas/usage=unused storedata.pas
    lib/replace emulator storedata
    purge/log storedata.*
    del/log storedata.obj;*

emulator.olb( storememory.obj) : storememory.pas,-
    [-]const.defn,-
    [-]stopcode.defn,-
    [-]byte.defn,-
    [-]memorytype.defn

    pas/usage=unused storememory.pas
    lib/replace emulator storememory
    purge/log storememory.*
    del/log storememory.obj;*
```

```
emulator.olb( storeoutput.obj) : storeoutput.pas,-
    [-]const.defn,-
    [-]byte.defn,-
    [-]ioconst.defn,-
    [-]stopcode.defn,-
    [-]type.defn,-
    [-]iotype.defn,-
    [-]iovarexternal.defn,-
    put8bitsofdata.subpas

    pas/usage=unused storeoutput.pas
    lib/replace emulator storeoutput
    purge/log storeoutput.*
    purge/log put8bitsofdata.subpas
    del/log storeoutput.obj;*

emulator.olb( storeregptr.obj) : storeregptr.pas,-
    [-]const.defn,-
    [-]regid.defn,-
    [-]stopcode.defn,-
    [-]regexternal.defn

    pas/usage=unused storeregptr.pas
    lib/replace emulator storeregptr
    purge/log storeregptr.*
    del/log storeregptr.obj;*

emulator.olb( string.obj) : string.pas,-
    [-]const.defn,-
    [-]flags.defn,-
    [-]regid.defn,-
    [-]stopcode.defn,-
    [-]type.defn,-
    [-]flagtype.defn

    pas/usage=unused string.pas
    lib/replace emulator string
    purge/log string.*
    del/log string.obj;*

emulator.olb( syntaxinterpreter.obj) : -
    syntaxinterpreter.pas,-
    [-]const.defn,-
    [-]debugconst.defn,-
    [-]type.defn,-
    [-]debugtype.defn,-
    uppercaseletter.subpas

    pas/usage=unused syntaxinterpreter.pas
    lib/replace emulator syntaxinterpreter
    purge/log syntaxinterpreter.*
    del/log syntaxinterpreter.obj;*
```

```
emulator.olb( unpackhex.obj) : unpackhex.pas,-
    [-]debugconst.defn

    pas/usage=unused unpackhex.pas
    lib/replace emulator unpackhex
    purge/log unpackhex.*
    del/log unpackhex.obj;*

emulator.olb( updateflags.obj) : updateflags.pas,-
    [-]const.defn,-
    [-]flags.defn,-
    [-]flagtype.defn

    pas/usage=unused updateflags.pas
    lib/replace emulator updateflags
    purge/log updateflags.*
    del/log updateflags.obj;*
```

IOHandling.mms

```
    iohandling.dummy : iomapgen.exe,-
        reviewiodata.exe

        show time

    iomapgen.exe : iomapgen.pas,-
        [-]const.defn,-
        [-]ioconst.defn,-
        [-]type.defn,-
        [-]iotype.defn,-
        searchforportaddress.subpas,-
        uppercaseletter.subpas,-
        emulator.olb(hexin.obj)

        pas/usage=unused iomapgen
        link iomapgen,emulator/lib
        purge/log iomapgen.*
        purge/log searchforportaddress.subpas
        purge/log uppercaseletter.subpas


    reviewiodata.exe : reviewiodata.pas,-
        [-]const.defn,-
        [-]ioconst.defn,-
        [-]byte.defn,-
        [-]type.defn,-
        [-]iotype.defn,-
        get8bitsofdata.subpas,-
        put8bitsofdata.subpas,-
        searchforportaddress.subpas,-
        emulator.olb(functions.obj),-
        emulator.olb(hexin.obj)

        pas/usage=unused reviewiodata
        link reviewiodata,emulator/lib
        purge/log reviewiodata.*
        purge/log get8bitsofdata.subpas
        purge/log put8bitsofdata.subpas
        purge/log searchforportaddress.subpas

    emulator.olb(hexin.obj) : hexin.pas

        pas/usage=unused hexin
        lib/replace emulator hexin
        purge/log hexin.*
        del hexin.obj;*
```

```
    emulator.olb(functions.obj) : functions.pas

        pas/usage=unused functions
        lib/replace emulator functions
        purge/log functions.*
        del functions.obj;*
```

KeywordLUTGen.mms

```
    keywordlutgen.exe : keywordlutgen.pas,-
        [-]const.defn,-
        [-]debugconst.defn,-
        [-]type.defn,-
        [-]debugtype.defn,-
        uppercaseletter.subpas

        pas/usage=unused keywordlutgen.pas
        link keywordlutgen
        purge/log keywordlutgen.*
        purge/log uppercaseletter.subpas
```

OpcodeLUTGen.mms

```
    opcodelutgen.exe : opcodelutgen.pas,-
        [-]const.defn,-
        [-]lookuptableclass.defn,-
        [-]type.defn,-
        uppercaseletter.subpas

        pas/usage=unused opcodelutgen.pas
        link opcodelutgen
        purge/log opcodelutgen.*
        purge/log uppercaseletter.subpas
```

Build Procedure

```
$ set noverify
$ !
$ ! Inquire if PASCAL should produce a list file
$ !
$ inquire listing "Do you want a listing(s) ? [N]"
$ if (listing .eqs. "Y") then goto generate
$ pas = "pas/nolis"
$ for = "for/nolis"
$ goto procede1
!
$ generate:
$ pas = "pas/lis"
$ for = "for/lis"
!
$ procede1:
$ show symbol pas
$ show symbol for
$ !
$ inquire compile "Do you want to compile all of the
        Emulator86 modules ? [Y/N] "
$ if (compile .nes. "Y") then goto procede2
$ set verify
$ !
$ ! Compile all Emulator86 modules
$ !
$ @compileemulator86.ritcom
$ !
$ set noverify
$ procede2:
$ !
$ inquire linkup "Do you want to link the
        Emulator86 modules ? [Y/N] "
$ if (linkup .nes. "Y") then goto procede3
$ set verify
$ !
$ link emulator86, emulator/lib
$ purge/log emulator86.exe
$ !
$ set noverify
$ !
```

```
$ procede3:
$ !
$ inquire buildio "Do you want to build the
        I/O handling programs ? [Y/N] "
$ if (buildio .nes. "Y") then goto procede4
$ set verify
$ !
$ ! Build the I/O Handling
$ !
$ @buildio86.ritcom
$ !
$ set noverify
$ procede4:
$ !
$ inquire buildopcode "Do you want to build the
        OpcodeLUTGenerator ? [Y/N] "
$ if (buildopcode .nes. "Y") then goto procede5
$ set verify
$ !
$ ! Build the Opcode LUT Generator
$ !
$ @buildopcodelutgen86.ritcom
$ !
$ set noverify
$ !
$ procede5:
$ !
$ inquire buildkeyword "Do you want to build the
        KeywordLUTGenerator? [Y/N] "
$ if (buildkeyword .nes. "Y") then goto procede6
$ set verify
$ !
$ ! Build the Keyword LUT Generator
$ !
$ @buildkeywordlutgen86.ritcom
$ set noverify
$ procede6:
$ exit
```

CompileEmulator86.com

```
$ !*****************************************************
$ !
$ set verify
$ !
$ ! Compile the Emulator86
$ !
$ ! set noverify
$ !
$ !
$ !*****************************************************
$       pas emulator86.pas
$ !
$       pas arithmetic.pas
$       lib/replace emulator arithmetic
$       purge/log arithmetic.*
$       del/log arithmetic.obj;*
$ !
$       pas computeea.pas
$       lib/replace emulator computeea
$       purge/log computeea.*
$       del/log computeea.obj;*
$ !
$       pas controltransfer.pas
$       lib/replace emulator controltransfer
$       purge/log controltransfer.*
$       del/log controltransfer.obj;*
$ !
$       pas datatransfer.pas
$       lib/replace emulator datatransfer
$       purge/log datatransfer.*
$       del/log datatransfer.obj;*
$ !
$       pas debugger.pas
$       lib/replace emulator debugger
$       purge/log debugger.*
$       purge/log get8bitsofdata.subpas
$       purge/log put8bitsofdata.subpas
$       purge/log searchforportaddress.subpas
$       purge/log memoryclass.subpas
$       purge/log stackclass.subpas
$       purge/log registerclass.subpas
$       purge/log flagclass.subpas
$       purge/log inputoutputclass.subpas
$       purge/log breakpointclass.subpas
$       purge/log executeclass.subpas
$       purge/log utilityclass.subpas
$       del/log debugger.obj;*
$ !
```

```
$       pas displaybreakpoint.pas
$       lib/replace emulator displaybreakpoint
$       purge/log displaybreakpoint.*
$       del/log displaybreakpoint.obj;*
$
$       pas executeinstruction.pas
$       lib/replace emulator executeinstruction
$       purge/log executeinstruction.*
$       del/log executeinstruction.obj;*
$ !
$       pas fetchcode.pas
$       lib/replace emulator fetchcode
$       purge/log fetchcode.*
$       del/log fetchcode.obj;*
$ !
$       pas fetchcodedata.pas
$       lib/replace emulator fetchcodedata
$       purge/log fetchcodedata.*
$       del/log fetchcodedata.obj;*
$ !
$       pas fetchinput.pas
$       lib/replace emulator fetchinput
$       purge/log get8bitsofdata.subpas
$       purge/log fetchinput.*
$       del/log fetchinput.obj;*
$ !
$       pas fetchinputline.pas
$       lib/replace emulator fetchinputline
$       purge/log fetchinputline.*
$       del/log fetchinputline.obj;*
$ !
$       pas fetchinstruction.pas
$       lib/replace emulator fetchinstruction
$       purge/log fetchinstruction.*
$       del/log fetchinstruction.obj;*
$ !
$       pas fetchmemory.pas
$       lib/replace emulator fetchmemory
$       purge/log fetchmemory.*
$       del/log fetchmemory.obj;*
$ !
$       pas fetchoperand.pas
$       lib/replace emulator fetchoperand
$       purge/log fetchoperand.*
$       del/log fetchoperand.obj;*
$ !
$       pas fetchregptr.pas
$       lib/replace emulator fetchregptr
$       purge/log fetchregptr.*
$       del/log fetchregptr.obj;*
$ !
```

```
$       pas functions.pas
$       lib/replace emulator functions
$       purge/log functions.*
$       del/log functions.obj;*
$
$       pas getaddress.pas
$       lib/replace emulator getaddress
$       purge/log getaddress.*
$       del/log getaddress.obj;*
$ !
$       pas getdata.pas
$       lib/replace emulator getdata
$       purge/log getdata.*
$       del/log getdata.obj;*
$ !
$       pas hexin.pas
$       lib/replace emulator hexin
$       purge/log hexin.*
$       del/log hexin.obj;*
$ !
$       pas installbreakpoints.pas
$       lib/replace emulator installbreakpoints
$       purge/log installbreakpoints.*
$       del/log installbreakpoints.obj;*
$ !
$       pas interrupt.pas
$       lib/replace emulator interrupt
$       purge/log interrupt.*
$       del/log interrupt.obj;*
$ !
$       pas loader.pas
$       lib/replace emulator loader
$       purge/log loader.*
$       del/log loader.obj;*
$ !
$       pas logic.pas
$       lib/replace emulator logic
$       purge/log logic.*
$       del/log logic.obj;*
$ !
$       pas parsefilename.pas
$       lib/replace emulator parsefilename
$       purge/log parsefilename.*
$       del/log parsefilename.obj;*
$ !
$       pas popflags.pas
$       lib/replace emulator popflags
$       purge/log popflags.*
$       del/log popflags.obj;*
$ !
```

```
$       pas popfromstack.pas
$       lib/replace emulator popfromstack
$       purge/log popfromstack.*
$       del/log popfromstack.obj;*
$
$       pas popintersegra.pas
$       lib/replace emulator popintersegra
$       purge/log popintersegra.*
$       del/log popintersegra.obj;*
$ !
$       pas processorcontrol.pas
$       lib/replace emulator processorcontrol
$       purge/log processorcontrol.*
$       del/log processorcontrol.obj;*
$ !
$       pas pushflags.pas
$       lib/replace emulator pushflags
$       purge/log pushflags.*
$       del/log pushflags.obj;*
$ !
$       pas pushintersegra.pas
$       lib/replace emulator pushintersegra
$       purge/log pushintersegra.*
$       del/log pushintersegra.obj;*
$ !
$       pas pushonstack.pas
$       lib/replace emulator pushonstack
$       purge/log pushonstack.*
$       del/log pushonstack.obj;*
$ !
$       pas removebreakpoints.pas
$       lib/replace emulator removebreakpoints
$       purge/log removebreakpoints.*
$       del/log removebreakpoints.obj;*
$ !
$       pas rmdecoder.pas
$       lib/replace emulator rmdecoder
$       purge/log rmdecoder.*
$       del/log rmdecoder.obj;*
$ !
$       pas searchforbreakpoint.pas
$       lib/replace emulator searchforbreakpoint
$       purge/log searchforbreakpoint.*
$       del/log searchforbreakpoint.obj;*
$ !
$       pas storedata.pas
$       lib/replace emulator storedata
$       purge/log storedata.*
$       del/log storedata.obj;*
$ !
```

```
$       pas storememory.pas
$       lib/replace emulator storememory
$       purge/log storememory.*
$       del/log storememory.obj;*
$
$       pas storeoutput.pas
$       lib/replace emulator storeoutput
$       purge/log storeoutput.*
$       purge/log put8bitsofdata.subpas
$       del/log storeoutput.obj;*
$ !
$       pas storeregptr.pas
$       lib/replace emulator storeregptr
$       purge/log storeregptr.*
$       del/log storeregptr.obj;*
$ !
$       pas string.pas
$       lib/replace emulator string
$       purge/log string.*
$       del/log string.obj;*
$ !
$       pas syntaxinterpreter.pas
$       lib/replace emulator syntaxinterpreter
$       purge/log syntaxinterpreter.*
$       del/log syntaxinterpreter.obj;*
$ !
$       pas unpackhex.pas
$       lib/replace emulator unpackhex
$       purge/log unpackhex.*
$       del/log unpackhex.obj;*
$ !
$       pas updateflags.pas
$       lib/replace emulator updateflags
$       purge/log updateflags.*
$       del/log updateflags.obj;*
$ !
$       for fileexists.for
$       lib/replace emulator fileexists
$       purge/log fileexists.*
$       del/log fileexists.obj;*
$ exit
```

BuildIO.com

```
$ !*****************************************************
$ !
$ set verify
$ !
$ ! Build the IOMapGen and ReviewIOData programs
$ !
$ !
$ !*****************************************************
$ !
$ inquire Utilities
 "Do you want to compile HEXIN and FUNCTIONS ? [N]"
$ if (Utilities .nes. "Y") then goto SkipUtilities
$ !
$       pas hexin
$       lib/replace emulator hexin
$       purge/log hexin.*
$       del/log hexin.obj;*
$ !
$       pas functions
$       lib/replace emulator functions
$       purge/log functions.*
$       del/log functions.obj;*
$ !
$ SkipUtilities:
$ !
$       pas iomapgen
$       link iomapgen,emulator/lib
$       purge/log iomapgen.*
$       purge/log searchforportaddress.subpas
$       purge/log uppercaseletter.subpas
$ !
$       pas reviewiodata
$       link reviewiodata,emulator/lib
$       purge/log reviewiodata.*
$       purge/log get8bitsofdata.subpas
$       purge/log put8bitsofdata.subpas
$       purge/log searchforportaddress.subpas
$ !
$ exit
```

BuildKeywordLUTGen.com

```
$ !*****************************************************
$ !
$ set verify
$ !
$ ! Build the Keyword LUT Generator
$ !
$ !
$ !*****************************************************
$       pas keywordlutgen.pas
$       link keywordlutgen
$       purge/log keywordlutgen.*
$ !
$ exit
```

BuildOpcodeLUTGen.com

```
$ !*****************************************************
$ !
$ set verify
$ !
$ ! Build the Opcode LUT Generator
$ !
$ !
$ !*****************************************************
$       pas opcodelutgen.pas
$       link opcodelutgen
$       purge/log opcodelutgen.*
$ !
$ exit
```

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

00001    C  0  0 { Title: Arithmetic
00002    C  0  0
00003    C  0  0   Purpose: Contains the software to execute the arithmetic instructions
00004    C  0  0
00005    C  0  0   Author: William A. Chapman      Date: February 15, 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 as
00012    C  0  0           requested for each instruction.
00013    C  0  0
00014    C  0  0   Procedures Invoked: FetchRegPtr
00015    C  0  0                       StoreRegPtr
00016    C  0  0                       FetchOperand
00017    C  0  0                       StoreData
00018    C  0  0                       FetchCodeData
00019    C  0  0                       Interrupt
00020    C  0  0
00021    C  0  0   Functions Invoked: SignExtend7
00022    C  0  0                       SignExtend15
00023    C  0  0                       UpdateFlags
00024    C  0  0 }
00025       0  0 module Arithmetic(input, output );
00026       0  0
00027       0  0 const
00028       0  0     %include  [-]Const.defn/list'
00029 I     0  0     FirstOpcodeValue = 0;
00030 I     0  0     LastOpcodeValue = 255;
00031 I     0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00032 I     0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00033 I     0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00034 I     0  0     Bit8Multiplier = %X'100';
00035 I     0  0     Bit8Divisor    = %X'100';
00036 I     0  0     WordMultiplier = %X'100';
00037 I     0  0
00038 I     0  0     EightBits = 0;                          {width for 8 bits}
00039 I     0  0     SixteenBits = 1;                        {width for 16 bits}
00040 I     0  0
00041 I     0  0     SimpleMode = %B'0';
00042 I     0  0     SimpleRM = %B'110';
00043 I     0  0
00044 I     0  0     LowMemoryLimit = 0;                     {low limit on memory array}
00045 I     0  0     HighMemoryLimit = 2048;                 {high limit on memory array}
```

```
00046 I     0   0
00047 I     0   0      FilenameLength = 40;
00048       0   0
00049       0   0      %include  [-]Flags.defn/list'
00050 I     0   0      SetHigh = true;
00051 I     0   0      Clear = false;
00052 I     0   0
00053 I     0   0      CarryFlag     = %B'000000000001';
00054 I     0   0      ParityFlag    = %B'000000000100';
00055 I     0   0      AuxCarryFlag  = %B'000000010000';
```

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

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

```
00100 I     0  0
00101 I     0  0      SPId = %B'100';
00102 I     0  0      SPWidth = 1;
00103 I     0  0
00104 I     0  0      BPId = %B'101 ;
00105 I     0  0      BPWidth = 1;
00106 I     0  0
00107 I     0  0      SIId = %B'110';
00108 I     0  0      SIWidth = 1;
00109 I     0  0
00110 I     0  0      DIId = %B'111';
```

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

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

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

00149       0  0     Register_is_Source = 0;
00150       0  0     Register_is_Destination = 1;
00151       0  0
00152       0  0     LowNibbleMask = %X'F';
00153       0  0
00154       0  0     CarrySet = 1;
00155       0  0     CarryClear = 0;
00156       0  0
00157       0  0     SignExtendMask = %B'10';
00158       0  0     WordSignExtension = %B'10';
00159       0  0
00160       0  0     ASCIIAdjustFactor = 10;
00161       0  0
00162       0  0     SingleWordMask = %X'0000FF00';
00163       0  0     SingleWordDivisor = %X'100';
00164       0  0
00165       0  0     DoubleWordMask = %X'FFFF0000';
00166       0  0     DoubleWordDivisor = %X'10000';
00167       0  0     DoubleWordMultiplier = %X'10000';
00168       0  0
00169       0  0     ByteSign = %X'80';
00170       0  0     WordSign = %X'8000';
00171       0  0     PositiveExtension = 0;
00172       0  0     NegativeExtension = -1;
00173       0  0
00174       0  0     ZeroValue = 0;
00175       0  0
00176       0  0     MaxByteQuotient = %X'FF';
00177       0  0     MaxWordQuotient = %X'FFFF';
00178       0  0
00179       0  0     MaxPositiveByteQuotient = %X'7F';
00180       0  0     MaxPositiveWordQuotient = %X'7FFF';
00181       0  0
00182       0  0     MinNegativeByteQuotient = %X'FFFFFF81';
00183       0  0     MinNegativeWordQuotient = %X'FFFF8001';
00184       0  0
00185       0  0     TypeZeroInterrupt = 0;
00186       0  0
00187       0  0     ByteOverflowMSB = %X'80';
00188       0  0     WordOverflowMSB = %X'8000';
```

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

00190       0  0 type
00191       0  0     %include  {-}Type.defn/list'
00192 I     0  0     ZeroOne = -1..1;                        {defines integer with range
00193 I C   0  0                                              of zero to one with an invalid
00194 I C   0  0                                              state of -1}
00195 I     0  0
00196 I     0  0     OpcodeLUTEntry = record                 {defines entries in opcode LUT}
00197 I     0  0   OpcodeClass: char;                        {class D, A, L, S, C, P}
00198 I     0  0   OpcodeKey: integer;                       {key to interpret opcode from
00199 I C   0  0                                              0 to 7}
00200 I     0  0   DirectionBitPresent: boolean;
00201 I     0  0   WidthBitPresent: boolean
00202 I     0  0     end;
00203 I     0  0
00204 I     0  0     OpcodeType = record
00205 I     0  0   Direction: ZeroOne;                      {to or from CPU}
00206 I     0  0   Full: unsigned;                          {full opcode}
00207 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00208 I     0  0     end;
00209 I     0  0
00210 I     0  0     RegisterType = record
00211 I     0  0   Id: integer;                            {identifier}
00212 I     0  0   Width: ZeroOne                           {8 or 16 bits}
00213 I     0  0     end;
00214 I     0  0
00215 I     0  0     EffectiveAddressType = record
00216 I     0  0   Mode: char;                             {register or memory}
00217 I     0  0   Width: ZeroOne;                         {8 or 16 bits}
00218 I     0  0   Address: unsigned;                      {memory address or
00219 I C   0  0                                              register designation}
00220 I     0  0   Segment: integer                        {segment register to use}
00221 I     0  0     end;
00222 I     0  0
00223 I     0  0     NameType = packed array[1..10] of char;
00224 I     0  0
00225 I     0  0     Characters = set of    ..'~';
00226 I     0  0     LettersAndNumbers = set of '0'..'z';
00227 I     0  0     Letters = set of 'A'..'z';
00228 I     0  0     Numbers = set of '0'..'9';
00229 I     0  0
00230 I     0  0     FileName = packed array [1..FilenameLength] of char;
00231       0  0
00232       0  0     %include '[-]FlagType.defn/list'
00233 I     0  0     FlagType = record                       {define processor flags}
```

```
00234 I    0  0   Carry: boolean;
00235 I    0  0   Parity: boolean;
00236 I    0  0   AuxCarry: boolean;
00237 I    0  0   Zero: boolean;
00238 I    0  0   Sign: boolean;
00239 I    0  0   Trap: boolean;
00240 I    0  0   Interrupt: boolean;
00241 I    0  0   Direction: boolean;
00242 I    0  0   Overflow: boolean
00243 I    0  0     end;
```

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

00245       0  0 var
00246       0  0     Flags: [external] FlagType;
00247       0  0     OpcodeKey: integer;
00248       0  0
00249       0  0     Addend1: unsigned;
00250       0  0     Addend2: unsigned;
00251       0  0     Sum: unsigned;
00252       0  0
00253       0  0     Minuend: unsigned;
00254       0  0     Subtrahend: unsigned;
00255       0  0     Difference: unsigned;
00256       0  0
00257       0  0     Multiplicand: unsigned;
00258       0  0     Multiplier: unsigned;
00259       0  0     Product: unsigned;
00260       0  0
00261       0  0     Numerator: unsigned;
00262       0  0     Divisor: unsigned;
00263       0  0     Quotient: unsigned;
00264       0  0     Remainder: unsigned;
00265       0  0
00266       0  0     Operand: unsigned;
00267       0  0
00268       0  0     Extension: unsigned;
00269       0  0     SignExtension: unsigned;
00270       0  0
00271       0  0     OverflowMSB: unsigned;
00272       0  0
00273       0  0     MaxQuotient: unsigned;
00274       0  0     MinQuotient: unsigned;
00275       0  0
00276       0  0     AHValue: unsigned;
00277       0  0     ALValue: unsigned;
00278       0  0
00279       0  0     RegisterValue: unsigned;
00280       0  0     OldValue: unsigned;
00281       0  0
00282       0  0     IP: [external] unsigned;
```

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

00284       0  0     IntAddend1: integer;
00285       0  0     IntAddend2: integer;
00286       0  0
00287       0  0     IntMinuend: integer;
00288       0  0     IntSubtrahend: integer;
00289       0  0
00290       0  0     IntMultiplicand: integer;
00291       0  0     IntMultiplier: integer;
00292       0  0
00293       0  0     IntNumerator: integer;
00294       0  0     IntDivisor: integer;
00295       0  0
00296       0  0     IntOperand: integer;
00297       0  0
00298       0  0     IntAHValue: integer;
00299       0  0     IntALValue: integer;
00300       0  0
00301       0  0     SignExtendWidth: integer;
00302       0  0
00303       0  0     CarryValue: integer;
```

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

00305      1  0 [global] procedure Arithmetic(Opcode: OpcodeType;
00306      1  0           Register: RegisterType;
00307      1  0           EA: EffectiveAddressType;
00308      1  0           var AStopCode: char);
00309      1  0
00310      2  0 procedure FetchRegPtr(FRPWidth: integer;
00311      2  0           FRPDesignator: integer;
00312      2  0           var FRPValue: unsigned;
00313      1  0           var FRPStopCode: char); external;
00314      1  0
00315      2  0 procedure StoreRegPtr(SRPWidth: integer;
00316      2  0           SRPDesignator: integer;
00317      2  0           SRPValue: unsigned;
00318      1  0           var SRPStopCode: char); external;
00319      1  0
00320      2  0 procedure FetchOperand(FOType: char;
00321      2  0           FOWidth: integer;
00322      2  0           FOAddress: unsigned;
00323      2  0           FOSegment: integer;
00324      2  0           var FOValue: unsigned;
00325      1  0           var FOStopCode: char); external;
00326      1  0
00327      2  0 procedure StoreData(SDType: char;
00328      2  0           SDWidth: integer;
00329      2  0           SDAddress: unsigned;
00330      2  0           SDSegment: integer;
00331      2  0           SDValue: unsigned;
00332      1  0           var SDStopCode: char); external;
00333      1  0
00334      2  0 procedure FetchCodeData(FCDWidth: integer;
00335      2  0           var FCDValue: unsigned;
00336      1  0           var FCDStopCode: char); external;
00337      1  0
00338      2  0 procedure Interrupt(IInterruptType: unsigned;
00339      1  0           var IStopCode: char); external;
```

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

00341      1  0 function SignExtend7(SEValue: unsigned): integer; external;
00342      1  0
00343      1  0 function SignExtend15(SEValue: unsigned): integer; external;
00344      1  0
00345      2  0 function UpdateZeroFlag(UFWidth: integer;
00346      1  0           UFResult: unsigned): boolean; external;
00347      1  0
00348      2  0 function UpdateSignFlag(UFWidth: integer;
00349      1  0           UFResult: unsigned): boolean; external;
00350      1  0
00351      2  0 function UpdateParityFlag(UFWidth: integer;
00352      1  0           UFResult: unsigned): boolean; external;
00353      1  0
00354      2  0 function UpdateOverflowFlag_Add(UFWidth: integer;
00355      2  0           UFAddend1: unsigned;
00356      2  0           UFAddend2: unsigned;
00357      1  0           UFResult: unsigned): boolean; external;
00358      1  0
00359      2  0 function UpdateOverflowFlag_Sub(UFWidth: integer;
00360      2  0           UFMinuend: unsigned;
00361      2  0           UFSubtrehend: unsigned;
00362      1  0           UFResult: unsigned): boolean; external;
00363      1  0
00364      2  0 function UpdateCarryFlag_Add(UFWidth: integer;
00365      2  0           UFAddend1: unsigned;
00366      2  0           UFAddend2: unsigned;
00367      1  0           UFResult: unsigned): boolean; external;
00368      1  0
00369      2  0 function UpdateCarryFlag_Sub(UFWidth: integer;
00370      2  0           UFMinuend: unsigned;
00371      2  0           UFSubtrehend: unsigned;
00372      1  0           UFResult: unsigned): boolean; external;
00373      1  0
00374      2  0 function UpdateAuxCarryFlag_Add(UFWidth: integer;
00375      2  0           UFAddend1: unsigned;
00376      2  0           UFAddend2: unsigned;
00377      1  0           UFResult: unsigned): boolean; external;
00378      1  0
00379      2  0 function UpdateAuxCarryFlag_Sub(UFWidth: integer;
00380      2  0           UFMinuend: unsigned;
00381      2  0           UFSubtrehend: unsigned;
00382      1  0           UFResult: unsigned): boolean; external;
00383      1  0
00384      2  0 function MaskShiftRight(MSRValue: unsigned;
```

```
00385      2  0          MSRMask: unsigned;
00386      1  0          MSRDivisor: integer): unsigned; external;
```

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

00388      1  1 begin
00389      1  1     OpcodeKey := int(Opcode.Full);
00390      1  1
00391      1  2     case OpcodeKey of
00392      1  2
00393      1  2   %X'0': {add register / memory with register to either} {A, C, O, P, S, Z}
00394      1  3     with Opcode do begin
00395      1  3       FetchRegPtr(Register.Width, Register.Id, Addend1, AStopCode);
00396      1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Addend2, AStopCode);
00397      1  4       if (Width = EightBits) then begin
00398      1  4           IntAddend1 := SignExtend7(Addend1);
00399      1  4           IntAddend2 := SignExtend7(Addend2);
00400      1  4       end
00401      1  4       else begin
00402      1  4           IntAddend1 := SignExtend15(Addend1);
00403      1  4           IntAddend2 := SignExtend15(Addend2);
00404      1  3       end;
00405      1  3       Sum := uint(IntAddend1 + IntAddend2);
00406      1  3       if (Direction = Register_is_Destination) then
00407      1  3           StoreRegPtr(Register.Width, Register.Id, Sum, AStopCode)
00408      1  3       else StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Sum, AStopCode);
00409      1  4       with Flags do begin
00410      1  4           AuxCarry := UpdateAuxCarryFlag_Add( Width,
00411      1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00412      1  4           Carry := UpdateCarryFlag_Add( Width,
00413      1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00414      1  4           Overflow := UpdateOverflowFlag_Add( Width,
00415      1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00416      1  4           Parity := UpdateParityFlag(Width, Sum);
00417      1  4           Sign := UpdateSignFlag(Width, Sum);
00418      1  4           Zero := UpdateZeroFlag(Width, Sum);
00419      1  3       end; {with Flags}
00420      1  2     end; {with opcode %X'0'}
```

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

00422       1  2   %X'4': {add immediate to accumulator} {A, C, O, P, S, Z}
00423       1  3     with Opcode do begin
00424       1  3       FetchRegPtr(Width, ALorAXID, Addend1, AStopCode);
00425       1  3       FetchCodeData(Width, Addend2, AStopCode);
00426       1  4       if (Width = EightBits) then begin
00427       1  4           IntAddend1 := SignExtend7(Addend1);
00428       1  4           IntAddend2 := SignExtend7(Addend2);
00429       1  4       end
00430       1  4       else begin
00431       1  4           IntAddend1 := SignExtend15(Addend1);
00432       1  4           IntAddend2 := SignExtend15(Addend2);
00433       1  3       end;
00434       1  3       Sum := uint(IntAddend1 + IntAddend2);
00435       1  3       StoreRegPtr(Width, ALorAXID, Sum, AStopCode);
00436       1  4       with Flags do begin
00437       1  4           AuxCarry := UpdateAuxCarryFlag_Add( Width,
00438       1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00439       1  4           Carry := UpdateCarryFlag_Add( Width,
00440       1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00441       1  4           Overflow := UpdateOverflowFlag_Add( Width,
00442       1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00443       1  4           Parity := UpdateParityFlag(Width, Sum);
00444       1  4           Sign := UpdateSignFlag(Width, Sum);
00445       1  4           Zero := UpdateZeroFlag(Width, Sum);
00446       1  3       end; {with Flags}
00447       1  2     end; {with opcode %X'4'}
```

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

00449       1  2   %X'10': {add with carry register / memory with register to either}
00450    C  1  2     {A, C, O, P, S, Z}
00451       1  3     with Opcode do begin
00452       1  3       FetchRegPtr(Register.Width, Register.Id, Addend1, AStopCode);
00453       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Addend2, AStopCode);
00454       1  4       if (Width = EightBits) then begin
00455       1  4           IntAddend1 := SignExtend7(Addend1);
00456       1  4           IntAddend2 := SignExtend7(Addend2);
00457       1  4       end
00458       1  4       else begin
00459       1  4           IntAddend1 := SignExtend15(Addend1);
00460       1  4           IntAddend2 := SignExtend15(Addend2);
00461       1  3       end;
00462       1  3       if (Flags.Carry) then CarryValue := CarrySet
00463       1  3       else CarryValue := CarryClear;
00464       1  3       Sum := uint(IntAddend1 + IntAddend2 + CarryValue);
00465       1  3       if (Direction = Register_is_Destination) then
00466       1  3           StoreRegPtr(Register.Width, Register Id, Sum, AStopCode)
00467       1  3       else StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Sum, AStopCode);
00468       1  4       with Flags do begin
00469       1  4           AuxCarry := UpdateAuxCarryFlag_Add( Width,
00470       1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00471       1  4           Carry := UpdateCarryFlag_Add( Width,
00472       1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00473       1  4           Overflow := UpdateOverflowFlag_Add( Width,
00474       1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00475       1  4           Parity := UpdateParityFlag(Width, Sum);
00476       1  4           Sign := UpdateSignFlag(Width, Sum);
00477       1  4           Zero := UpdateZeroFlag(Width, Sum);
00478       1  3       end; {with Flags}
00479       1  2     end; {with opcode %X'10'}
```

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

00481       1  2   %X'14': {add with carry register / memory with register to either}
00482    C  1  2     {A, C, O, P, S, Z}
00483       1  3     with Opcode do begin
00484       1  3       FetchRegPtr(Width, ALorAXID, Addend1, AStopCode);
00485       1  3       FetchCodeData(Width, Addend2, AStopCode);
00486       1  4       if (Width = EightBits) then begin
00487       1  4           IntAddend1 := SignExtend7(Addend1);
00488       1  4           IntAddend2 := SignExtend7(Addend2);
00489       1  4       end
00490       1  4       else begin
00491       1  4           IntAddend1 := SignExtend15(Addend1);
00492       1  4           IntAddend2 := SignExtend15(Addend2);
00493       1  3       end;
00494       1  3       if (Flags.Carry) then CarryValue := CarrySet
00495       1  3       else CarryValue := CarryClear;
00496       1  3       Sum := uint(IntAddend1 + IntAddend2 + CarryValue);
00497       1  3       StoreRegPtr(Width, ALorAXID, Sum, AStopCode);
00498       1  4       with Flags do begin
00499       1  4           AuxCarry := UpdateAuxCarryFlag_Add( Width,
00500       1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00501       1  4           Carry := UpdateCarryFlag_Add( Width,
00502       1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00503       1  4           Overflow := UpdateOverflowFlag_Add( Width,
00504       1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00505       1  4           Parity := UpdateParityFlag(Width, Sum);
00506       1  4           Sign := UpdateSignFlag(Width, Sum);
00507       1  4           Zero := UpdateZeroFlag(Width, Sum);
00508       1  3       end; {with Flags}
00509       1  2     end; {with opcode %X'14'}
```

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

00511       1  2   %X'18': {subtract with borrow register / memory with register to either}
00512    C  1  2     {A, C, O, P, S, Z}
00513       1  3     with Opcode do begin
00514       1  4       if (Direction = Register_is_Destination) then begin
00515       1  4           FetchRegPtr(Register.Width, Register.Id, Minuend, AStopCode);
00516       1  4           FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Subtrahend, AStopCode);
00517       1  4       end
00518       1  4       else begin
00519       1  4           FetchRegPtr(Register.Width, Register.Id, Subtrahend, AStopCode);
00520       1  4           FetchOperand(EA.Mode,EA.Width, EA.Address, EA.Segment, Minuend, AStopCode);
00521       1  3       end;
00522       1  4       if (Width = EightBits) then begin
00523       1  4           IntMinuend := SignExtend7(Minuend);
00524       1  4           IntSubtrahend := SignExtend7(Subtrahend);
00525       1  4       end
00526       1  4       else begin
00527       1  4           IntMinuend:= SignExtend15(Minuend);
00528       1  4           IntSubtrahend := SignExtend15(Subtrahend);
00529       1  3       end;
00530       1  3       if (Flags.Carry) then CarryValue := CarrySet
00531       1  3       else CarryValue := CarryClear;
00532       1  3       Difference := uint(IntMinuend - IntSubtrahend - CarryValue);
00533       1  3       if (Direction = Register_is_Destination) then
00534       1  3           StoreRegPtr(Register.Width, Register.Id, Difference, AStopCode)
00535       1  3       else
00536       1  3           StoreData(EA.Mode,EA.Width, EA.Address, EA.Segment, Difference, AStopCode);
00537       1  4       with Flags do begin
00538       1  4           AuxCarry := UpdateAuxCarryFlag_Sub( Width,
00539       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00540       1  4           Carry := UpdateCarryFlag_Sub( Width,
00541       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00542       1  4           Overflow := UpdateOverflowFlag_Sub( Width,
00543       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00544       1  4           Parity := UpdateParityFlag(Width, Difference);
00545       1  4           Sign := UpdateSignFlag(Width, Difference);
00546       1  4           Zero := UpdateZeroFlag(Width, Difference);
00547       1  3       end; {with Flags}
00548       1  2     end; {with opcode %X'18'}
```

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

00550       1  2   %X'1C': {subtract with borrow immediate from accumulator}
00551    C  1  2     {A, C, O, P, S, Z}
00552       1  3     with Opcode do begin
00553       1  3       FetchRegPtr(Width, ALorAXID, Minuend, AStopCode);
00554       1  3       FetchCodeData(Width, Subtrahend, AStopCode);
00555       1  4       if (Width = EightBits) then begin
00556       1  4           IntMinuend := SignExtend7(Minuend);
00557       1  4           IntSubtrahend := SignExtend7(Subtrahend);
00558       1  4       end
00559       1  4       else begin
00560       1  4           IntMinuend:= SignExtend15(Minuend);
00561       1  4           IntSubtrahend := SignExtend15(Subtrahend);
00562       1  3       end;
00563       1  3       if (Flags.Carry) then CarryValue := CarrySet
00564       1  3       else CarryValue := CarryClear;
00565       1  3       Difference := uint(IntMinuend - IntSubtrahend - CarryValue);
00566       1  3       StoreRegPtr(Width, ALorAXID, Difference, AStopCode);
00567       1  4       with Flags do begin
00568       1  4           AuxCarry := UpdateAuxCarryFlag_Sub( Width,
00569       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00570       1  4           Carry := UpdateCarryFlag_Sub( Width,
00571       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00572       1  4           Overflow := UpdateOverflowFlag_Sub( Width,
00573       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00574       1  4           Parity := UpdateParityFlag(Width, Difference);
00575       1  4           Sign := UpdateSignFlag(Width, Difference);
00576       1  4           Zero := UpdateZeroFlag(Width, Difference);
00577       1  3       end; {with Flags}
00578       1  2     end; {with opcode %X'1C'}
00579       1  2
00580       1  2   %X'27': {Decimal adjust for add} {A, C, P, S, Z} {O is undefined}
00581       1  3     with Flags do begin
00582       1  3       FetchRegPtr(ALWidth, ALID, ALValue, AStopCode);
00583       1  4       if ((int(uand(ALValue, LowNibbleMask)) > 9) or AuxCarry) then begin
00584       1  4           ALValue := uint(SignExtend7(ALValue) + 6);
00585       1  4           AuxCarry := true;
00586       1  4       end
00587       1  3       else AuxCarry := false;
00588       1  4       if ((int(uand(ALValue, Low8Bits)) > %X'9F') or Carry) then begin
00589       1  4           ALValue := uint(SignExtend7(ALValue) + %X'60');
00590       1  4           Carry := true;
00591       1  4       end
00592       1  3       else Carry := false;
00593       1  3       StoreRegPtr(ALWidth, ALID, ALValue, AStopCode);
```

```
00594      1  3       Parity := UpdateParityFlag(ALWidth, ALValue);
00595      1  3       Sign := UpdateSignFlag(ALWidth, ALValue);
00596      1  3       Zero := UpdateZeroFlag(ALWidth, ALValue);
00597      1  2     end; {with flags Case %X'27'}
```

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

00599       1  2   %X'28': {subtract register / memory with register to either}
00600    C  1  2     {A, C, O, P, S, Z}
00601       1  3     with Opcode do begin
00602       1  4       if (Direction = Register_is_Destination) then begin
00603       1  4           FetchRegPtr(Register.Width, Register.Id, Minuend, AStopCode);
00604       1  4           FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Subtrahend, AStopCode);
00605       1  4       end
00606       1  4       else begin
00607       1  4           FetchRegPtr(Register.Width, Register.Id, Subtrahend, AStopCode);
00608       1  4           FetchOperand(EA.Mode,EA.Width, EA.Address, EA.Segment, Minuend, AStopCode);
00609       1  3       end;
00610       1  4       if (Width = EightBits) then begin
00611       1  4           IntMinuend := SignExtend7(Minuend);
00612       1  4           IntSubtrahend := SignExtend7(Subtrahend);
00613       1  4       end
00614       1  4       else begin
00615       1  4           IntMinuend:= SignExtend15(Minuend);
00616       1  4           IntSubtrahend := SignExtend15(Subtrahend);
00617       1  3       end;
00618       1  3       Difference := uint(IntMinuend - IntSubtrahend);
00619       1  3       if (Direction = Register_is_Destination) then
00620       1  3           StoreRegPtr(Register.Width, Register.Id, Difference, AStopCode)
00621       1  3       else StoreData(EA.Mode,EA.Width, EA.Address, EA.Segment, Difference, AStopCode);
00622       1  4       with Flags do begin
00623       1  4           AuxCarry := UpdateAuxCarryFlag_Sub( Width,
00624       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00625       1  4           Carry := UpdateCarryFlag_Sub( Width,
00626       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00627       1  4           Overflow := UpdateOverflowFlag_Sub( Width,
00628       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00629       1  4           Parity := UpdateParityFlag(Width, Difference);
00630       1  4           Sign := UpdateSignFlag(Width, Difference);
00631       1  4           Zero := UpdateZeroFlag(Width, Difference);
00632       1  3       end; {with Flags}
00633       1  2     end; {with opcode %X'28'}
```

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

00635       1  2   %X'2C': {subtract immediate from accumulator}
00636    C  1  2     {A, C, O, P, S, Z}
00637       1  3     with Opcode do begin
00638       1  3       FetchRegPtr(Width, ALorAXID, Minuend, AStopCode);
00639       1  3       FetchCodeData(Width, Subtrahend, AStopCode);
00640       1  4       if (Width = EightBits) then begin
00641       1  4           IntMinuend := SignExtend7(Minuend);
00642       1  4           IntSubtrahend := SignExtend7(Subtrahend);
00643       1  4       end
00644       1  4       else begin
00645       1  4           IntMinuend:= SignExtend15(Minuend);
00646       1  4           IntSubtrahend := SignExtend15(Subtrahend);
00647       1  3       end;
00648       1  3       Difference := uint(IntMinuend - IntSubtrahend);
00649       1  3       StoreRegPtr(Width, ALorAXID, Difference, AStopCode);
00650       1  4       with Flags do begin
00651       1  4           AuxCarry := UpdateAuxCarryFlag_Sub( Width,
00652       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00653       1  4           Carry := UpdateCarryFlag_Sub( Width,
00654       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00655       1  4           Overflow := UpdateOverflowFlag_Sub( Width,
00656       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00657       1  4           Parity := UpdateParityFlag(Width, Difference);
00658       1  4           Sign := UpdateSignFlag(Width, Difference);
00659       1  4           Zero := UpdateZeroFlag(Width, Difference);
00660       1  3       end; {with Flags}
00661       1  2     end; {with opcode %X'2C'}
00662       1  2
00663       1  2   %X'2F': {Decimal adjust for sub} {A, C, P, S, Z} {O is undefined}
00664       1  3     with Flags do begin
00665       1  3       FetchRegPtr(ALWidth, ALID, ALValue, AStopCode);
00666       1  4       if ((int(uand(ALValue, LowNibbleMask)) > 9) or AuxCarry) then begin
00667       1  4           ALValue := uint(SignExtend7(ALValue) - 6);
00668       1  4           AuxCarry := true;
00669       1  4       end
00670       1  3       else AuxCarry := false;
00671       1  4       if ((int(uand(ALValue, Low8Bits)) > %X'9F') or Carry) then begin
00672       1  4           ALValue := uint(SignExtend7(ALValue) - %X'60');
00673       1  4           Carry := true;
00674       1  4       end
00675       1  3       else Carry := false;
00676       1  3       StoreRegPtr(ALWidth, ALID, ALValue, AStopCode);
00677       1  3       Parity := UpdateParityFlag(ALWidth, ALValue);
00678       1  3       Sign := UpdateSignFlag(ALWidth, ALValue);
```

```
00679      1  3       Zero := UpdateZeroFlag(ALWidth, ALValue);
00680      1  2     end; {with flags Case %X'2F'}
```

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

00682       1  2   %X'37': {ASCII adjust for add} {A, C} {O, P, S, Z are undefined}
00683       1  3     with Flags do begin
00684       1  3       FetchRegPtr(ALWidth, ALID, ALValue, AStopCode);
00685       1  4       if ((int(uand(ALValue, LowNibbleMask)) > 9) or AuxCarry) then begin
00686       1  4           ALValue := uint(SignExtend7(ALValue) + 6);
00687       1  4           FetchRegPtr(AHWidth, AHID, AHValue, AStopCode);
00688       1  4           AHValue := uint((SignExtend7(AHValue)) + 1);
00689       1  4           StoreRegPtr(AHWidth, AHID, AHValue, AStopCode);
00690       1  4           AuxCarry := true;
00691       1  4       end
00692       1  3       else AuxCarry := false;
00693       1  3       Carry := AuxCarry;
00694       1  3       ALValue := uand(ALValue, LowNibbleMask);
00695       1  3       StoreRegPtr(ALWidth, ALID, ALValue, AStopCode);
00696       1  2     end; {with flags Case %X'37'}
00697       1  2
00698       1  2   %X'38': {compare register / memory with register to either}
00699   C   1  2     {A, C, O, P, S, Z}
00700       1  3     with Opcode do begin
00701       1  4       if (Direction = Register_is_Destination) then begin
00702       1  4           FetchRegPtr(Register.Width, Register.Id, Minuend, AStopCode);
00703       1  4           FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Subtrahend, AStopCode);
00704       1  4       end
00705       1  4       else begin
00706       1  4           FetchRegPtr(Register.Width, Register.Id, Subtrahend, AStopCode);
00707       1  4           FetchOperand(EA.Mode,EA.Width, EA.Address, EA.Segment, Minuend, AStopCode);
00708       1  3       end;
00709       1  4       if (Width = EightBits) then begin
00710       1  4           IntMinuend := SignExtend7(Minuend);
00711       1  4           IntSubtrahend := SignExtend7(Subtrahend);
00712       1  4       end
00713       1  4       else begin
00714       1  4           IntMinuend:= SignExtend15(Minuend);
00715       1  4           IntSubtrahend := SignExtend15(Subtrahend);
00716       1  3       end;
00717       1  3       Difference := uint(IntMinuend - IntSubtrahend);
00718       1  4       with Flags do begin
00719       1  4           AuxCarry := UpdateAuxCarryFlag_Sub( Width,
00720       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00721       1  4           Carry := UpdateCarryFlag_Sub( Width,
00722       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00723       1  4           Overflow := UpdateOverflowFlag_Sub( Width,
00724       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00725       1  4           Parity := UpdateParityFlag(Width, Difference);
```

```
00726      1  4           Sign := UpdateSignFlag(Width, Difference);
00727      1  4           Zero := UpdateZeroFlag(Width, Difference);
00728      1  3         end; {with Flags}
00729      1  2       end; {with opcode %X'38'}
```

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

00731       1  2   %X'3C': {compare immediate with accumulator}
00732    C  1  2     {A, C, O, P, S, Z}
00733       1  3     with Opcode do begin
00734       1  3       FetchRegPtr(Width, ALorAXID, Minuend, AStopCode);
00735       1  3       FetchCodeData(Width, Subtrahend, AStopCode);
00736       1  4       if (Width = EightBits) then begin
00737       1  4           IntMinuend := SignExtend7(Minuend);
00738       1  4           IntSubtrahend := SignExtend7(Subtrahend);
00739       1  4       end
00740       1  4       else begin
00741       1  4           IntMinuend:= SignExtend15(Minuend);
00742       1  4           IntSubtrahend := SignExtend15(Subtrahend);
00743       1  3       end;
00744       1  3       Difference := uint(IntMinuend - IntSubtrahend);
00745       1  4       with Flags do begin
00746       1  4           AuxCarry := UpdateAuxCarryFlag_Sub( Width,
00747       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00748       1  4           Carry := UpdateCarryFlag_Sub( Width,
00749       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00750       1  4           Overflow := UpdateOverflowFlag_Sub( Width,
00751       1  4               uint(IntMinuend), uint(Subtrahend), Difference);
00752       1  4           Parity := UpdateParityFlag(Width, Difference);
00753       1  4           Sign := UpdateSignFlag(Width, Difference);
00754       1  4           Zero := UpdateZeroFlag(Width, Difference);
00755       1  3       end; {with Flags}
00756       1  2     end; {with opcode %X'3C'}
00757       1  2
00758       1  2   %X'3F': {ASCII adjust for sub} {A, C} {O, P, S, Z are undefined}
00759       1  3     with Flags do begin
00760       1  3       FetchRegPtr(ALWidth, ALID, ALValue, AStopCode);
00761       1  4       if ((int(uand(ALValue, LowNibbleMask)) > 9) or AuxCarry) then begin
00762       1  4           ALValue := uint(SignExtend7(ALValue) - 6);
00763       1  4           FetchRegPtr(AHWidth, AHID, AHValue, AStopCode);
00764       1  4           AHValue := uint((SignExtend7(AHValue)) - 1);
00765       1  4           StoreRegPtr(AHWidth, AHID, AHValue, AStopCode);
00766       1  4           AuxCarry := true;
00767       1  4       end
00768       1  3       else AuxCarry := false;
00769       1  3       Carry := AuxCarry;
00770       1  3       ALValue := uand( ALValue, LowNibbleMask);
00771       1  3       StoreRegPtr(ALWidth, ALID, ALValue, AStopCode);
00772       1  2     end; {with flags Case %X'3F'}
```

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

00774       1  2   %X'40': {increment 16 bit register} {A, O, P, S, Z}  {C is not affected}
00775       1  3     with Register do begin
00776       1  3       FetchRegPtr(Width, ID, RegisterValue, AStopCode);
00777       1  3       OldValue := RegisterValue;
00778       1  3       RegisterValue := uint(SignExtend15(RegisterValue) + 1);
00779       1  3       StoreRegPtr(Width, ID, RegisterValue, AStopCode);
00780       1  4       with Flags do begin
00781       1  4           AuxCarry := UpdateAuxCarryFlag_Add(Width, OldValue,
00782       1  4                   %X'1', RegisterValue);
00783       1  4           Overflow := UpdateOverflowFlag_Add(Width, OldValue,
00784       1  4                   %X'1', RegisterValue);
00785       1  4           Parity := UpdateParityFlag(Width, RegisterValue);
00786       1  4           Sign := UpdateSignFlag(Width, RegisterValue);
00787       1  4           Zero := UpdateZeroFlag(Width, RegisterValue);
00788       1  3       end; {with Flags}
00789       1  2   end; {with Register case %X'40'}
00790       1  2
00791       1  2   %X'48': {decrement 16 bit register} {A, O, P, S, Z}  {C is not affected}
00792       1  3     with Register do begin
00793       1  3       FetchRegPtr(Width, ID, RegisterValue, AStopCode);
00794       1  3       OldValue := RegisterValue;
00795       1  3       RegisterValue := uint(SignExtend15(RegisterValue) - 1);
00796       1  3       StoreRegPtr(Width, ID, RegisterValue, AStopCode);
00797       1  4       with Flags do begin
00798       1  4           AuxCarry := UpdateAuxCarryFlag_Sub(Width, OldValue,
00799       1  4                   %X'1', RegisterValue);
00800       1  4           Overflow := UpdateOverflowFlag_Sub(Width, OldValue,
00801       1  4                   %X'1', RegisterValue);
00802       1  4           Parity := UpdateParityFlag(Width, RegisterValue);
00803       1  4           Sign := UpdateSignFlag(Width, RegisterValue);
00804       1  4           Zero := UpdateZeroFlag(Width, RegisterValue);
00805       1  3       end; {with Flags}
00806       1  2   end; {with Register case %X'48'}
```

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

00808       1  2   %X'080',%X'082': {add immediate to register / memory}
00809    C  1  2     {A, C, O, P, S, Z}
00810       1  3     with Opcode do begin
00811       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Addend1, AStopCode);
00812       1  4       if (Width = SixteenBits) then begin
00813       1  4           if (uand(Full, SignExtendMask) = WordSignExtension)
00814       1  4           then SignExtendWidth := EightBits               {S:W = 11}
00815       1  4           else SignExtendWidth := SixteenBits;            {S:W = 01}
00816       1  4       end
00817       1  3       else SignExtendWidth := EightBits;
00818       1  3       FetchCodeData( SignExtendWidth, Addend2, AStopCode);
00819       1  4       if (Width = EightBits) then begin
00820       1  4           IntAddend1 := SignExtend7(Addend1);
00821       1  4           IntAddend2 := SignExtend7(Addend2);
00822       1  4       end
00823       1  4       else begin
00824       1  4           IntAddend1 := SignExtend15(Addend1);
00825       1  4           if (SignExtendWidth = EightBits) then
00826       1  4               IntAddend2 := SignExtend7(Addend2)
00827       1  4           else IntAddend2 := SignExtend15(Addend2);
00828       1  3       end;
00829       1  3       Sum := uint(IntAddend1 + IntAddend2);
00830       1  3       StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Sum, AStopCode);
00831       1  4       with Flags do begin
00832       1  4           AuxCarry := UpdateAuxCarryFlag_Add(Width,
00833       1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00834       1  4           Carry := UpdateCarryFlag_Add(Width,
00835       1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00836       1  4           Overflow := UpdateOverflowFlag_Add(Width,
00837       1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00838       1  4           Parity := UpdateParityFlag(Width, Sum);
00839       1  4           Sign := UpdateSignFlag(Width, Sum);
00840       1  4           Zero := UpdateZeroFlag(Width, Sum);
00841       1  3       end; {with Flags}
00842       1  2     end; {with opcode %X'080',%X'082'}
```

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

00844       1  2   %X'98': {convert byte to word} {no flags affected}
00845       1  3     begin
00846       1  3       FetchRegPtr(ALWidth, ALID, RegisterValue, AStopCode);
00847       1  3       if (RegisterValue < ByteSign) then
00848       1  3           StoreRegPtr( AHWidth, AHID, PositiveExtension, AStopCode)
00849       1  3       else StoreRegPtr( AHWidth, AHID, NegativeExtension, AStopCode);
00850       1  2     end;
00851       1  2
00852       1  2   %X'99': {convert word to double word} {no flags affected}
00853       1  3     begin
00854       1  3       FetchRegPtr(AXWidth, AXID, RegisterValue, AStopCode);
00855       1  3       if (RegisterValue < WordSign) then
00856       1  3           StoreRegPtr(DXWidth, DXID, PositiveExtension, AStopCode)
00857       1  3       else StoreRegPtr(DXWidth, DXID, NegativeExtension, AStopCode);
00858       1  2     end;
00859       1  2
00860       1  2   %X'D4': {ASCII adjust for multiplication} {P, S, Z} {A, C, O undefined}
00861       1  3     begin
00862       1  3       FetchRegPtr(ALWidth, ALID, ALValue, AStopCode);
00863       1  3       IntALValue := int( ALValue);
00864       1  3       AHValue := uint( utrunc(IntALValue / ASCIIAdjustFactor));
00865       1  3       StoreRegPtr(AHWidth, AHID, AHValue, AStopCode);
00866       1  3       ALValue := uint(IntALValue rem ASCIIAdjustFactor);
00867       1  3       StoreRegPtr(ALWidth, ALID, ALValue, AStopCode);
00868       1  4       with Flags do begin
00869       1  4           Parity := UpdateParityFlag(ALWidth, ALValue);
00870       1  4           Sign := UpdateSignFlag(ALWidth, ALValue);
00871       1  4           Zero := UpdateZeroFlag(ALWidth, ALValue);
00872       1  3       end; {with Flags}
00873       1  2   end; {case %X'D4'}
00874       1  2
00875       1  2   %X'D5': {ASCII adjust for division} {P, S, Z} {A, C, O undefined}
00876       1  3     begin
00877       1  3       FetchRegPtr(ALWidth, ALID, ALValue, AStopCode);
00878       1  3       IntALValue := int( ALValue);
00879       1  3       FetchRegPtr(AHWidth, AHID, AHValue, AStopCode);
00880       1  3       IntAHValue := int( AHValue);
00881       1  3       ALValue := uint( (IntAHValue * ASCIIAdjustFactor) + IntALValue);
00882       1  3       StoreRegPtr(ALWidth, ALID, ALValue, AStopCode);
00883       1  3       StoreRegPtr(AHWidth, AHID, ZeroValue, AStopCode);
00884       1  4       with Flags do begin
00885       1  4           Parity := UpdateParityFlag(ALWidth, ALValue);
00886       1  4           Sign := UpdateSignFlag(ALWidth, ALValue);
00887       1  4           Zero := UpdateZeroFlag(ALWidth, ALValue);
```

```
00888      1  3       end; {with Flags}
00889      1  2   end; {case %X'D5'}
```

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

00891       1  2   %X'280',%X'282': {add immediate with carry to register / memory}
00892    C  1  2     {A, C, O, P, S, Z}
00893       1  3     with Opcode do begin
00894       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Addend1, AStopCode);
00895       1  4       if (Width = SixteenBits) then begin
00896       1  4           if (uand(Full, SignExtendMask) = WordSignExtension)
00897       1  4           then SignExtendWidth := EightBits               {S:W = 11}
00898       1  4           else SignExtendWidth := SixteenBits;            {S:W = 01}
00899       1  4       end
00900       1  3       else SignExtendWidth := EightBits;
00901       1  3       FetchCodeData( SignExtendWidth, Addend2, AStopCode);
00902       1  4       if (Width = EightBits) then begin
00903       1  4           IntAddend1 := SignExtend7(Addend1);
00904       1  4           IntAddend2 := SignExtend7(Addend2);
00905       1  4       end
00906       1  4       else begin
00907       1  4           IntAddend1 := SignExtend15(Addend1);
00908       1  4           if (SignExtendWidth = EightBits) then
00909       1  4               IntAddend2 := SignExtend7(Addend2)
00910       1  4           else IntAddend2 := SignExtend15(Addend2);
00911       1  3       end;
00912       1  3       if  (Flags.Carry) then CarryValue := CarrySet
00913       1  3       else CarryValue := CarryClear;
00914       1  3       Sum := uint(IntAddend1 + IntAddend2 + CarryValue);
00915       1  3       StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Sum, AStopCode);
00916       1  4       with Flags do begin
00917       1  4           AuxCarry := UpdateAuxCarryFlag_Add(Width,
00918       1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00919       1  4           Carry := UpdateCarryFlag_Add(Width,
00920       1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00921       1  4           Overflow := UpdateOverflowFlag_Add(Width,
00922       1  4               uint(IntAddend1), uint(IntAddend2), Sum);
00923       1  4           Parity := UpdateParityFlag(Width, Sum);
00924       1  4           Sign := UpdateSignFlag(Width, Sum);
00925       1  4           Zero := UpdateZeroFlag(Width, Sum);
00926       1  3       end; {with Flags}
00927       1  2     end; {with opcode %X'080',%X'082'}
```

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

00929       1  2   %X'380',%X'382': {subtract with borrow immediate from register / memory}
00930     C 1  2     {A, C, O, P, S, Z}
00931       1  3     with Opcode do begin
00932       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Minuend, AStopCode);
00933       1  4       if (Width = SixteenBits) then begin
00934       1  4           if (uand(Full, SignExtendMask) = WordSignExtension)
00935       1  4           then SignExtendWidth := EightBits
00936       1  4           else SignExtendWidth := SixteenBits;
00937       1  4       end
00938       1  3       else SignExtendWidth := EightBits;
00939       1  3       FetchCodeData( SignExtendWidth, Subtrahend, AStopCode);
00940       1  4       if (Width = EightBits) then begin
00941       1  4           IntMinuend := SignExtend7(Minuend);
00942       1  4           IntSubtrahend := SignExtend7(Subtrahend);
00943       1  4       end
00944       1  4       else begin
00945       1  4           IntMinuend:= SignExtend15(Minuend);
00946       1  4           if (SignExtendWidth = EightBits) then
00947       1  4               IntSubtrahend := SignExtend7(Subtrahend)
00948       1  4           else IntSubtrahend := SignExtend15(Subtrahend);
00949       1  3       end;
00950       1  3       if (Flags.Carry) then CarryValue := CarrySet
00951       1  3       else CarryValue := CarryClear;
00952       1  3       Difference := uint(IntMinuend - IntSubtrahend - CarryValue);
00953       1  3       StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Difference, AStopCode);
00954       1  4       with Flags do begin
00955       1  4           AuxCarry := UpdateAuxCarryFlag_Sub( Width,
00956       1  4               uint(IntMinuend), uint(IntSubtrahend), Difference);
00957       1  4           Carry := UpdateCarryFlag_Sub( Width,
00958       1  4               uint(IntMinuend), uint(IntSubtrahend), Difference);
00959       1  4           Overflow := UpdateOverflowFlag_Sub( Width,
00960       1  4               uint(IntMinuend), uint(IntSubtrahend), Difference);
00961       1  4           Parity := UpdateParityFlag(Width, Difference);
00962       1  4           Sign := UpdateSignFlag(Width, Difference);
00963       1  4           Zero := UpdateZeroFlag(Width, Difference);
00964       1  3       end; {with Flags}
00965       1  2     end; {with opcode %X'380',%X'382'}
```

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

00967       1  2   %X'580',%X'582': {subtract immediate from register / memory}
00968    C  1  2     {A, C, O, P, S, Z}
00969       1  3     with Opcode do begin
00970       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Minuend, AStopCode);
00971       1  4       if (Width = SixteenBits) then begin
00972       1  4           if (uand(Full, SignExtendMask) = WordSignExtension)
00973       1  4           then SignExtendWidth := EightBits
00974       1  4           else SignExtendWidth := SixteenBits;
00975       1  4       end
00976       1  3       else SignExtendWidth := EightBits;
00977       1  3       FetchCodeData( SignExtendWidth, Subtrahend, AStopCode);
00978       1  4       if (Width = EightBits) then begin
00979       1  4           IntMinuend := SignExtend7(Minuend);
00980       1  4           IntSubtrahend := SignExtend7(Subtrahend);
00981       1  4       end
00982       1  4       else begin
00983       1  4           IntMinuend:= SignExtend15(Minuend);
00984       1  4           if (SignExtendWidth = EightBits) then
00985       1  4               IntSubtrahend := SignExtend7(Subtrahend)
00986       1  4           else IntSubtrahend := SignExtend15(Subtrahend);
00987       1  3       end;
00988       1  3       Difference := uint(IntMinuend - IntSubtrahend);
00989       1  3       StoreData(EA.Mode, EA.Width, EA.Address, EA.Segment, Difference, AStopCode);
00990       1  4       with Flags do begin
00991       1  4           AuxCarry := UpdateAuxCarryFlag_Sub( Width,
00992       1  4               uint(IntMinuend), uint(IntSubtrahend), Difference);
00993       1  4           Carry := UpdateCarryFlag_Sub( Width,
00994       1  4               uint(IntMinuend), uint(IntSubtrahend), Difference);
00995       1  4           Overflow := UpdateOverflowFlag_Sub( Width,
00996       1  4               uint(IntMinuend), uint(IntSubtrahend), Difference);
00997       1  4           Parity := UpdateParityFlag(Width, Difference);
00998       1  4           Sign := UpdateSignFlag(Width, Difference);
00999       1  4           Zero := UpdateZeroFlag(Width, Difference);
01000       1  3       end; {with Flags}
01001       1  2     end; {with opcode %X'580',%X'582'}
```

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

01003       1  2   %X'780',%X'782': {compare immediate with register / memory}
01004    C  1  2     {A, C, O, P, S, Z}
01005       1  3     with Opcode do begin
01006       1  3       FetchOperand(EA.Mode, EA.Width, EA.Address, EA.Segment, Minuend, AStopCode);
01007       1  4       if (Width = SixteenBits) then begin
01008       1  4           if (uand(Full, SignExtendMask) = WordSignExtension)
01009       1  4           then SignExtendWidth := EightBits
01010       1  4           else SignExtendWidth := SixteenBits;
01011       1  4       end
01012       1  3       else SignExtendWidth := EightBits;
01013       1  3       FetchCodeData( SignExtendWidth, Subtrahend, AStopCode);
01014       1  4       if (Width = EightBits) then begin
01015       1  4           IntMinuend := SignExtend7(Minuend);
01016       1  4           IntSubtrahend := SignExtend7(Subtrahend);
01017       1  4       end
01018       1  4       else begin
01019       1  4           IntMinuend:= SignExtend15(Minuend);
01020       1  4           if (SignExtendWidth - EightBits) then
01021       1  4               IntSubtrahend := SignExtend7(Subtrahend)
01022       1  4           else IntSubtrahend := SignExtend15(Subtrahend);
01023       1  3       end;
01024       1  3       Difference := uint(IntMinuend - IntSubtrahend);
01025       1  4       with Flags do begin
01026       1  4           AuxCarry := UpdateAuxCarryFlag_Sub( Width,
01027       1  4               uint(IntMinuend), uint(IntSubtrahend), Difference);
01028       1  4           Carry := UpdateCarryFlag_Sub( Width,
01029       1  4               uint(IntMinuend), uint(IntSubtrahend), Difference);
01030       1  4           Overflow := UpdateOverflowFlag_Sub( Width,
01031       1  4               uint(IntMinuend), uint(IntSubtrahend), Difference);
01032       1  4           Parity := UpdateParityFlag(Width, Difference);
01033       1  4           Sign := UpdateSignFlag(Width, Difference);
01034       1  4           Zero := UpdateZeroFlag(Width, Difference);
01035       1  3       end; {with Flags}
01036       1  2     end; {with opcode %X'780',%X'782'}
```

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

01038       1  2   %X'0FE': {increment register / memory}
01039    C  1  2     {A, O, P, S, Z}  {C is not affected}
01040       1  3     with EA do begin
01041       1  3       FetchOperand(Mode, Width, Address, Segment, Operand, AStopCode);
01042       1  3       if (Width = EightBits) then IntOperand := SignExtend7(Operand)
01043       1  3       else IntOperand := SignExtend15(Operand);
01044       1  3       OldValue := uint(IntOperand);
01045       1  3       Operand := uint(IntOperand + 1);
01046       1  3       StoreData(Mode, Width, Address, Segment, Operand, AStopCode);
01047       1  4       with Flags do begin
01048       1  4           AuxCarry := UpdateAuxCarryFlag_Add(Width, OldValue,
01049       1  4                   %X'1', Operand);
01050       1  4           Overflow := UpdateOverflowFlag_Add(Width, OldValue,
01051       1  4                   %X'1', Operand);
01052       1  4           Parity := UpdateParityFlag(Width, Operand);
01053       1  4           Sign := UpdateSignFlag(Width, Operand);
01054       1  4           Zero := UpdateZeroFlag(Width, Operand);
01055       1  3       end; {with Flags}
01056       1  2   end; {with EA case %X'0FE'}
01057       1  2
01058       1  2   %X'1FE': {decrement register / memory}
01059    C  1  2     {A, O, P, S, Z}  {C is not affected}
01060       1  3     with EA do begin
01061       1  3       FetchOperand(Mode, Width, Address, Segment, Operand, AStopCode);
01062       1  3       if (Width = EightBits) then IntOperand := SignExtend7(Operand)
01063       1  3       else IntOperand := SignExtend15(Operand);
01064       1  3       OldValue := uint(IntOperand);
01065       1  3       Operand := uint(IntOperand - 1);
01066       1  3       StoreData(Mode, Width, Address, Segment, Operand, AStopCode);
01067       1  4       with Flags do begin
01068       1  4           AuxCarry := UpdateAuxCarryFlag_Sub(Width, OldValue,
01069       1  4                   %X'1', Operand);
01070       1  4           Overflow := UpdateOverflowFlag_Sub(Width, OldValue,
01071       1  4                   %X'1', Operand);
01072       1  4           Parity := UpdateParityFlag(Width, Operand);
01073       1  4           Sign := UpdateSignFlag(Width, Operand);
01074       1  4           Zero := UpdateZeroFlag(Width, Operand);
01075       1  3       end; {with Flags}
01076       1  2   end; {with EA case %X'1FE'}
```

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

01078       1  2   %X'3F6': {negate} {A, C, O, P, S, Z}
01079       1  3     with EA do begin
01080       1  3       FetchOperand(Mode, Width, Address, Segment, Operand, AStopCode);
01081       1  4       if (Width = EightBits) then begin
01082       1  4           IntOperand := SignExtend7(Operand);
01083       1  4           OverflowMSB := ByteOverflowMSB;
01084       1  4       end
01085       1  4       else begin
01086       1  4           IntOperand := SignExtend15(Operand);
01087       1  4           OverflowMSB := WordOverflowMSB;
01088       1  3       end;
01089       1  3       Operand := unot(uint(IntOperand));
01090       1  3       OldValue := Operand;
01091       1  3       Operand := uint( int(Operand) + 1);
01092       1  3       StoreData(Mode, Width, Address, Segment, Operand, AStopCode);
01093       1  4       with Flags do begin
01094       1  4           if (uand(Operand, LowNibbleMask) = 0) then AuxCarry := Clear
01095       1  4               else AuxCarry := SetHigh;
01096       1  4           if (Operand = 0) then Carry := Clear
01097       1  4               else Carry := SetHigh;
01098       1  4           if (Operand = OverflowMSB) then Overflow := SetHigh
01099       1  4               else Overflow := Clear;
01100       1  4           Parity := UpdateParityFlag(Width, Operand);
01101       1  4           Sign := UpdateSignFlag(Width, Operand);
01102       1  4           Zero := UpdateZeroFlag(Width, Operand);
01103       1  3       end; {with Flags}
01104       1  2   end; {with EA case %X'3F6'}
```

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

01106       1  2   %X'4F6': {unsigned multiply} {C, O} {A, P, S, Z  undefined}
01107       1  3     with EA do begin
01108       1  3       FetchRegPtr(Width, ALorAXID, Multiplicand, AStopCode);
01109       1  3       FetchOperand(Mode, Width, Address, Segment, Multiplier, AStopCode);
01110       1  4       if (Width = EightBits) then begin
01111       1  4           IntMultiplicand := int( Multiplicand);
01112       1  4           IntMultiplier := int( Multiplier);
01113       1  4       end
01114       1  4       else begin
01115       1  4           IntMultiplicand := int( Multiplicand);
01116       1  4           IntMultiplier := int( Multiplier);
01117       1  3       end;
01118       1  3       Product := uint(IntMultiplicand * IntMultiplier);
01119       1  3       StoreRegPtr(AXWidth, AXID, Product, AStopCode);
01120       1  4       if (Width = SixteenBits) then begin
01121       1  4           Extension :=
01122       1  4               MaskShiftRight(Product, DoubleWordMask, DoubleWordDivisor);
01123       1  4           Extension := uand(Extension, All16Bits);
01124       1  4           StoreRegPtr(DXWidth, DXID, Extension, AStopCode);
01125       1  4       end
01126       1  3       else Extension :=
01127       1  3           MaskShiftRight(Product, SingleWordMask, SingleWordDivisor);
01128       1  3       if (Extension = 0) then Flags.Carry := Clear
01129       1  3       else Flags.Carry := SetHigh;
01130       1  3       Flags.Overflow := Flags.Carry;
01131       1  2   end; {case %X'4F6'}
```

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

01133       1  2   %X'5F6': {integer multiply} {C, O} {A, P, S, Z  undefined}
01134       1  3     with EA do begin
01135       1  3       FetchRegPtr(Width, ALorAXID, Multiplicand, AStopCode);
01136       1  3       FetchOperand(Mode, Width, Address, Segment, Multiplier, AStopCode);
01137       1  4       if (Width = EightBits) then begin
01138       1  4           IntMultiplicand := SignExtend7(Multiplicand);
01139       1  4           IntMultiplier := SignExtend7(Multiplier);
01140       1  4       end
01141       1  4       else begin
01142       1  4           IntMultiplicand := SignExtend15(Multiplicand);
01143       1  4           IntMultiplier := SignExtend15(Multiplier);
01144       1  3       end;
01145       1  3       Product := uint(IntMultiplicand * IntMultiplier);
01146       1  3       StoreRegPtr(AXWidth, AXID, Product, AStopCode);
01147       1  4       if (Width = SixteenBits) then begin
01148       1  4           Extension :=
01149       1  4               MaskShiftRight(Product, DoubleWordMask, DoubleWordDivisor);
01150       1  4           Extension := uand(Extension, All16Bits);
01151       1  4           StoreRegPtr(DXWidth, DXID, Extension, AStopCode);
01152       1  4           Product := uand(Product, All16Bits);
01153       1  4           Product := SignExtend15(Product);
01154       1  4           SignExtension :=
01155       1  4               MaskShiftRight(Product, DoubleWordMask, DoubleWordDivisor);
01156       1  4           SignExtension := uand(SignExtension, All16Bits);
01157       1  4       end
01158       1  4       else begin
01159       1  4           Extension :=
01160       1  4               MaskShiftRight(Product, SingleWordMask, SingleWordDivisor);
01161       1  4           Product := uand(Product, Low8Bits);
01162       1  4           Product := SignExtend7(Product);
01163       1  4           SignExtension :=
01164       1  4               MaskShiftRight(Product, SingleWordMask, SingleWordDivisor);
01165       1  3       end;
01166       1  3       if (Extension = SignExtension) then Flags.Carry := Clear
01167       1  3       else Flags.Carry := SetHigh;
01168       1  3       Flags.Overflow := Flags.Carry;
01169       1  2   end; {case %X'4F6'}
```

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

01171       1  2   %X'6F6': {unsigned divide} {all flags undefined}
01172       1  3     with EA do begin
01173       1  3       FetchRegPtr(AXWidth, AXID, Numerator, AStopCode);
01174       1  3       FetchOperand(Mode, Width, Address, Segment, Divisor, AStopCode);
01175       1  4       if (Width = EightBits) then begin
01176       1  4           IntNumerator := int( uand( Numerator, All16Bits));
01177       1  4           IntDivisor := int( uand( Divisor, Low8Bits));
01178       1  4           MaxQuotient := MaxByteQuotient;
01179       1  4       end
01180       1  4       else begin
01181       1  4           FetchRegPtr(DXWidth, DXID, Extension, AStopCode);
01182       1  4           IntNumerator := int(uor(uand((Extension * DoubleWordMultiplier),
01183       1  4               DoubleWordMask), Numerator));
01184       1  4           IntDivisor := int( uand( Divisor, All16Bits));
01185       1  4           MaxQuotient := MaxWordQuotient;
01186       1  3       end;
01187       1  3       if ((IntNumerator / IntDivisor) > int(MaxQuotient) )
01188       1  3           then Interrupt(TypeZeroInterrupt, AStopCode)
01189       1  4       else begin
01190       1  4           Quotient := utrunc(IntNumerator / IntDivisor);
01191       1  4           StoreRegPtr(Width, ALorAXID, Quotient, AStopCode);
01192       1  4           Remainder := uint(IntNumerator rem IntDivisor);
01193       1  4           if (Width = EightBits)
01194       1  4               then StoreRegPtr(AHWidth, AHID, Remainder, AStopCode)
01195       1  4           else StoreRegPtr(DXWidth, DXID, Remainder, AStopCode);
01196       1  3       end;
01197       1  2   end; {case %X'6F6'}
```

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

01199       1  2   %X'7F6': {integer divide} {all flags undefined}
01200       1  3     with EA do begin
01201       1  3       FetchRegPtr(AXWidth, AXID, Numerator, AStopCode);
01202       1  3       FetchOperand(Mode, Width, Address, Segment, Divisor, AStopCode);
01203       1  4       if (Width = EightBits) then begin
01204       1  4           IntNumerator := SignExtend15(Numerator);
01205       1  4           IntDivisor := SignExtend7(Divisor);
01206       1  4           MaxQuotient := MaxPositiveByteQuotient;
01207       1  4           MinQuotient := MinNegativeByteQuotient;
01208       1  4       end
01209       1  4       else begin
01210       1  4           FetchRegPtr(DXWidth, DXID, Extension, AStopCode);
01211       1  4           IntNumerator := int(uor(uand((Extension * DoubleWordMultiplier),
01212       1  4               DoubleWordMask), Numerator));
01213       1  4           IntDivisor := SignExtend15(Divisor);
01214       1  4           MaxQuotient := MaxPositiveWordQuotient;
01215       1  4           MinQuotient := MinNegativeWordQuotient;
01216       1  3       end;
01217       1  3       if ((((IntNumerator / IntDivisor) > 0) and
01218       1  3           ((IntNumerator / IntDivisor) > int(MaxQuotient))) or
01219       1  3           (((IntNumerator / IntDivisor) < 0) and
01220       1  3           ((IntNumerator / IntDivisor) < int(MinQuotient))))
01221       1  3               then Interrupt(TypeZeroInterrupt, AStopCode)
01222       1  4       else begin
01223       1  4           Quotient := utrunc(IntNumerator / IntDivisor);
01224       1  4           StoreRegPtr(Width, ALorAXID, Quotient, AStopCode);
01225       1  4           Remainder := uint(IntNumerator rem IntDivisor);
01226       1  4           if (Width = EightBits)
01227       1  4               then StoreRegPtr(AHWidth, AHID, Remainder, AStopCode)
01228       1  4           else StoreRegPtr(DXWidth, DXID, Remainder, AStopCode);
01229       1  3       end;
01230       1  3    end {case %X'7F6'}
01231       1  3
01232       1  2    otherwise AStopCode := BadOpcode;
01233       1  1      end; {case OpcodeKey}
01234       1  1
01235       0  0 end;
01236       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	11812	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	148	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

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

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	432	00:00.5	00:03.3
Source Analysis	446	00:04.9	00:15.3
Source Listing	15	00:01.5	00:08.4
Tree Construction	504	00:02.0	00:05.0
Flow Analysis	290	00:02.0	00:04.4
Value Propagation	10	00:00.2	00:00.3
Profit Analysis	97	00:00.9	00:02.9
Context Analysis	87	00:13.4	00:29.6
Name Packing	7	00:00.4	00:00.5
Code Selection	174	00:02.1	00:03.7
Final	132	00:03.1	00:06.9
TOTAL	2201	00:31.0	01:20.4

COMPILATION STATISTICS

CPU Time: 00:31.0 (2396 Lines/Minute)

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

00001   C  0  0 { Title: Compute Effective Address
00002   C  0  0
00003   C  0  0   Purpose: Compute an Effective Address by combining
00004   C  0  0            displacements, registers, indices and pointers
00005   C  0  0            as appropriate.
00006   C  0  0
00007   C  0  0   Author: William A. Chapman      Date: January 12, 1986
00008   C  0  0
00009   C  0  0   Inputs: Displacement (as a signed number)
00010   C  0  0            Mode - indicates which registers to combine
00011   C  0  0
00012   C  0  0   Outputs: Effective Address - unsigned number
00013   C  0  0
00014   C  0  0   Procedures Invoked: FetchRegPtr
00015   C  0  0 }
00016      0  0 module ComputeEA(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  [-]RegId.defn/list'
00041 I     0  0     ALId = %B'000';
00042 I     0  0     ALWidth = 0;
00043 I     0  0
00044 I     0  0     CLId = %B'001';
00045 I     0  0     CLWidth = 0;
00046 I     0  0
00047 I     0  0     DLId = %B'010';
00048 I     0  0     DLWidth = 0;
00049 I     0  0
00050 I     0  0     BLId = %B'011';
00051 I     0  0     BLWidth = 0;
00052 I     0  0
00053 I     0  0     AHId = %B'100';
00054 I     0  0     AHWidth = 0;
00055 I     0  0
00056 I     0  0     CHId = %B'101';
00057 I     0  0     CHWidth = 0;
00058 I     0  0
00059 I     0  0     DHId = %B'110';
00060 I     0  0     DHWidth = 0;
00061 I     0  0
00062 I     0  0     BHId = %B'111 ;
00063 I     0  0     BHWidth = 0;
00064 I     0  0
00065 I     0  0     AXId = %B'000';
00066 I     0  0     AXWidth = 1;
00067 I     0  0     ALorAXId = %B'000';
00068 I     0  0
00069 I     0  0     CXId = %B'001';
00070 I     0  0     CXWidth = 1;
00071 I     0  0
00072 I     0  0     DXId = %B'010';
00073 I     0  0     DXWidth = 1;
00074 I     0  0
00075 I     0  0     BXId = %B'011';
00076 I     0  0     BXWidth = 1;
00077 I     0  0
00078 I     0  0     SPId = %B'100';
00079 I     0  0     SPWidth = 1;
00080 I     0  0
00081 I     0  0     BPId = %B'101';
00082 I     0  0     BPWidth = 1;
00083 I     0  0
```

```
00084 I     0   0      SIId = %B'110';
00085 I     0   0      SIWidth = 1;
00086 I     0   0
00087 I     0   0      DIId = %B'111';
00088 I     0   0      DIWidth = 1;
00089 I     0   0
00090 I     0   0      SRWidth = 2;
00091 I     0   0
00092 I     0   0      ESId = %B'00';
00093 I     0   0      ESWidth = 2;
00094 I     0   0
```

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

00095 I     0  0     CSId = %B'01 ;
00096 I     0  0     CSWidth = 2;
00097 I     0  0
00098 I     0  0     SSId = %B'10';
00099 I     0  0     SSWidth = 2;
00100 I     0  0
00101 I     0  0     DSId = %B'11';
00102 I     0  0     DSWidth = 2;
00103 I     0  0
00104       0  0
00105       0  0     %include  [-]StopCode.defn/list'
00106 I     0  0     Breakpoint            = 'B';
00107 I     0  0     Call          = 'C';
00108 I     0  0     ControlD             = 'D';
00109 I     0  0     BadOpcode              = 'E';
00110 I     0  0     BadCEAModeValue        = 'F';
00111 I     0  0     BadRMDModeValue        = 'G';
00112 I     0  0     Halt          = 'H';
00113 I     0  0     BadRegisterID = 'I';
00114 I     0  0     BadOpcodeKey  = 'K';
00115 I     0  0     BadMemoryAddress       = 'M';
00116 I     0  0     Normal                 = 'N';
00117 I     0  0     BadPortAddress         = 'P';
00118 I     0  0     BadOpcodeClass         = 'Q';
00119 I     0  0     Return                 = 'R';
00120 I     0  0     BadCheckSum            = 'S';
00121 I     0  0     BadOperandType         = 'T';
00122 I     0  0     BadMemoryWidth         = 'W';
00123 I     0  0     BadOpcodeExtension     = 'X';
00124 I     0  0     NoMemoryAccess       = 'Z';
00125       0  0
00126       0  0
00127       0  0 var
00128       0  0     Pointer: unsigned;            {unsigned value of pointer}
00129       0  0     Register: unsigned;
00130       0  0     Index: unsigned;
00131       0  0
00132       0  0     Address: integer;             {signed accumulation of components}
00133       0  0
00134       0  0
00135       0  0
00136       1  0 [global] procedure ComputeEA(CEADisplacement: integer;
00137       1  0           CEAMode: integer;
00138       1  0           var CEAAddress: unsigned;
```

```
00139      1  0           var CEAStopCode: char);
00140      1  0
00141      2  0 procedure FetchRegPtr(FRPWidth: integer;
00142      2  0           FRPDesignator: integer;
00143      2  0           var FRPValue: unsigned;
00144      1  0           var FRPStopCode: char); external;
```

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

00146      1  1 begin
00147      1  1     Pointer := 0;         {initialize variables}
00148      1  1     Register := 0;
00149      1  1     Index := 0;
00150      1  1
00151      1  2     case CEAMode of
00152      1  3   0: begin
00153      1  3        FetchRegPtr( BXWidth, BXId, Register, CEAStopCode);
00154      1  3        FetchRegPtr( SIWidth, SIId, Index, CEAStopCode);
00155      1  2     end;
00156      1  2
00157      1  3   1: begin
00158      1  3        FetchRegPtr( BXWidth, BXId, Register, CEAStopCode);
00159      1  3        FetchRegPtr( DIWidth, DIId, Index, CEAStopCode);
00160      1  2     end;
00161      1  2
00162      1  3   2: begin
00163      1  3        FetchRegPtr( BPWidth, BPId, Pointer, CEAStopCode);
00164      1  3        FetchRegPtr( SIWidth, SIId, Index, CEAStopCode);
00165      1  2     end;
00166      1  2
00167      1  3   3: begin
00168      1  3        FetchRegPtr( BPWidth, BPId, Pointer, CEAStopCode);
00169      1  3        FetchRegPtr( DIWidth, DIId, Index, CEAStopCode);
00170      1  2     end;
00171      1  2
00172      1  2   4: FetchRegPtr(SIWidth, SIId, Index, CEAStopCode);
00173      1  2   5: FetchRegPtr(DIWidth, DIId, Index, CEAStopCode);
00174      1  2   6: FetchRegPtr(BPWidth, BPId, Pointer, CEAStopCode);
00175      1  2   7: FetchRegPtr(BXWidth, BXId, Register, CEAStopCode);
00176      1  2
00177      1  2   otherwise CEAStopCode := BadCEAModeValue;
00178      1  2
00179      1  1     end; {case CEAMode}
00180      1  1     Address := int(Pointer) + int(Register) + int(Index) + CEADisplacement;
00181      1  1     CEAAddress := uand(uint(Address),All16Bits);
00182      0  0 end;
00183      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	452	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	16	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

PAS/LIS COMPUTEEA.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]COMPUTEEA.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]COMPUTEEA.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	00:03.7
Source Analysis	226	00:00.5	00:04.0
Source Listing	14	00:00.4	00:02.6
Tree Construction	114	00:00.1	00:00.7
Flow Analysis	37	00:00.1	00:00.1
Value Propagation	9	00:00.0	00:00.0
Profit Analysis	41	00:00.1	00:00.1
Context Analysis	177	00:00.5	00:01.1
Name Packing	6	00:00.0	00:00.0
Code Selection	72	00:00.1	00:00.1
Final	60	00:00.1	00:01.4
TOTAL	1164	00:02.3	00:13.7

COMPILATION STATISTICS

CPU Time: 00:02.3 (4712 Lines/Minute)

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

00001    C  0  0 { Title: Control Transfer
00002    C  0  0
00003    C  0  0   Purpose: Manipulate the Instruction Pointer and Code Segment
00004    C  0  0           register as required
00005    C  0  0
00006    C  0  0   Author: William A. Chapman      Date: February 23, 1986
00007    C  0  0
00008    C  0  0   Inputs: Opcode record
00009    C  0  0           Register record
00010    C  0  0           Effective Address record
00011    C  0  0           Instruction Pointer (global)
00012    C  0  0           Code Segment Register
00013    C  0  0           Flags (global)
00014    C  0  0
00015    C  0  0   Outputs: Instruction Pointer is updated as required
00016    C  0  0           Code Segment Register is updated as required
00017    C  0  0
00018    C  0  0   Procedures Invoked: FetchRegPtr
00019    C  0  0           StoreRegPtr
00020    C  0  0           FetchCodeData
00021    C  0  0           PushIntersegRA
00022    C  0  0           PopIntersegRA
00023    C  0  0           PushOnStack
00024    C  0  0           PopFromStack
00025    C  0  0           PopFlags
00026    C  0  0           FetchOperand
00027    C  0  0           Interrupt
00028    C  0  0           (XOR)
00029    C  0  0
00030    C  0  0   Functions Invoked: SignExtend7
00031    C  0  0           SignExtend15
00032    C  0  0 }
00033       0  0 module ControlTransfer(input, output );
00034       0  0
00035       0  0 const
00036       0  0     %include  [-]Const.defn/list'
00037 I     0  0     FirstOpcodeValue = 0;
00038 I     0  0     LastOpcodeValue = 255;
00039 I     0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00040 I     0  0     Low8Bits = %X'FF';                {mask for low 8 bits}
00041 I     0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00042 I     0  0     Bit8Multiplier = %X'100';
00043 I     0  0     Bit8Divisor    = %X'100';
00044 I     0  0     WordMultiplier = %X'100';
00045 I     0  0
```

```
00046 I     0  0      EightBits = 0;                    {width for 8 bits}
00047 I     0  0      SixteenBits = 1;                  {width for 16 bits}
00048 I     0  0
00049 I     0  0      SimpleMode = %B'0';
00050 I     0  0      SimpleRM = %B'110';
00051 I     0  0
00052 I     0  0      LowMemoryLimit = 0;               {low limit on memory array}
00053 I     0  0      HighMemoryLimit = 2048;           {high limit on memory array}
00054 I     0  0
00055 I     0  0      FilenameLength = 40;
```

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

00056       0  0
00057       0  0     %include '[-]RegId.defn/list'
00058 I     0  0     ALId = %B'000';
00059 I     0  0     ALWidth = 0;
00060 I     0  0
00061 I     0  0     CLId = %B'001';
00062 I     0  0     CLWidth = 0;
00063 I     0  0
00064 I     0  0     DLId = %B'010';
00065 I     0  0     DLWidth = 0;
00066 I     0  0
00067 I     0  0     BLId = %B'011 ;
00068 I     0  0     BLWidth = 0;
00069 I     0  0
00070 I     0  0     AHId = %B'100';
00071 I     0  0     AHWidth = 0;
00072 I     0  0
00073 I     0  0     CHId = %B'101';
00074 I     0  0     CHWidth = 0;
00075 I     0  0
00076 I     0  0     DHId = %B'110';
00077 I     0  0     DHWidth = 0;
00078 I     0  0
00079 I     0  0     BHId = %B'111 ;
00080 I     0  0     BHWidth = 0;
00081 I     0  0
00082 I     0  0     AXId = %B'000';
00083 I     0  0     AXWidth = 1;
00084 I     0  0     ALorAXId = %B'000';
00085 I     0  0
00086 I     0  0     CXId = %B'001 ;
00087 I     0  0     CXWidth = 1;
00088 I     0  0
00089 I     0  0     DXId = %B'010';
00090 I     0  0     DXWidth = 1;
00091 I     0  0
00092 I     0  0     BXId = %B'011 ;
00093 I     0  0     BXWidth = 1;
00094 I     0  0
00095 I     0  0     SPId = %B'100';
00096 I     0  0     SPWidth = 1;
00097 I     0  0
00098 I     0  0     BPId = %B'101';
00099 I     0  0     BPWidth = 1;
```

```
00100 I     0  0
00101 I     0  0      SIId = %B'110';
00102 I     0  0      SIWidth = 1;
00103 I     0  0
00104 I     0  0      DIId = %B'111 ;
00105 I     0  0      DIWidth = 1;
00106 I     0  0
00107 I     0  0      SRWidth = 2;
00108 I     0  0
00109 I     0  0      ESId = %B'00';
00110 I     0  0      ESWidth = 2;
```

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

00111 I     0  0
00112 I     0  0      CSId = %B'01';
00113 I     0  0      CSWidth = 2;
00114 I     0  0
00115 I     0  0      SSId = %B'10';
00116 I     0  0      SSWidth = 2;
00117 I     0  0
00118 I     0  0      DSId = %B'11';
00119 I     0  0      DSWidth = 2;
00120 I     0  0
```

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

00122       0  0     %include  [-]Flags.defn/list'
00123  I    0  0     SetHigh = true;
00124  I    0  0     Clear = false;
00125  I    0  0
00126  I    0  0     CarryFlag     = %B'000000000001';
00127  I    0  0     ParityFlag    = %B'000000000100';
00128  I    0  0     AuxCarryFlag  = %B'000000010000';
00129  I    0  0     ZeroFlag      = %B'000001000000';
00130  I    0  0     SignFlag      = %B'000010000000';
00131  I    0  0     TrapFlag      = %B'000100000000';
00132  I    0  0     InterruptFlag = %B'001000000000';
00133  I    0  0     DirectionFlag = %B'010000000000';
00134  I    0  0     OverflowFlag  = %B'100000000000';
00135       0  0
00136       0  0     DisplacementWidth = 0;
00137       0  0     InterruptWidth = 0;
00138       0  0     IntraSegmentDisplacementWidth = 1;
00139       0  0
00140       0  0     CSLength = 1;
00141       0  0     IPLength = 1;
00142       0  0     PopValueLength = 1;
00143       0  0
00144       0  0     Type3Interrupt = 3;
00145       0  0     OverflowInterrupt = 4;
00146       0  0
00147       0  0     %include  [-]StopCode.defn/list'
00148  I    0  0     Breakpoint            = 'B';
00149  I    0  0     Call          = 'C';
00150  I    0  0     ControlD            = 'D';
00151  I    0  0     BadOpcode             = 'E';
00152  I    0  0     BadCEAModeValue       = 'F';
00153  I    0  0     BadRMDModeValue       = 'G';
00154  I    0  0     Halt          = 'H';
00155  I    0  0     BadRegisterID = 'I';
00156  I    0  0     BadOpcodeKey  = 'K';
00157  I    0  0     BadMemoryAddress      = 'M';
00158  I    0  0     Normal                = 'N';
00159  I    0  0     BadPortAddress        = 'P';
00160  I    0  0     BadOpcodeClass        = 'Q';
00161  I    0  0     Return                = 'R';
00162  I    0  0     BadCheckSum           = 'S';
00163  I    0  0     BadOperandType        = 'T';
00164  I    0  0     BadMemoryWidth        = 'W';
00165  I    0  0     BadOpcodeExtension    = 'X';
```

```
00166 I     0  0       NoMemoryAccess          - '%';
```

```
-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;
```

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

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;
00222       0  0
00223       0  0
00224       0  0 var
00225       0  0     IP: [external] unsigned;
00226       0  0     Flags: [external] FlagType;
00227       0  0
00228       0  0     OpcodeKey: integer;
00229       0  0
00230       0  0     CXValue: unsigned;
00231       0  0
00232       0  0     NewIP: unsigned;
00233       0  0     NewCS: unsigned;
00234       0  0
00235       0  0     StackPointer: unsigned;
00236       0  0     PopValue: unsigned;
00237       0  0
00238       0  0     InterruptType: unsigned;
00239       0  0
00240       0  0     IntraSegmentDisplacement: unsigned;
00241       0  0
00242       0  0     Displacement: unsigned;
00243       0  0     IntDisplacement: integer;
```

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

00245      1  0 [global] procedure ControlTransfer(Opcode: OpcodeType;
00246      1  0           Register: RegisterType;
00247      1  0           EA: EffectiveAddressType;
00248      1  0           var CTStopCode: char);
00249      1  0
00250      2  0 procedure FetchRegPtr(FRPWidth: integer;
00251      2  0           FRPDesignator: integer;
00252      2  0           var FRPValue: unsigned;
00253      1  0           var FRPStopCode: char); external;
00254      1  0
00255      2  0 procedure PushOnStack(POSValue: unsigned;
00256      1  0           var POSStopCode: char); external;
00257      1  0
00258      2  0 procedure PopFromStack(var PFSValue: unsigned;
00259      1  0           var PFSStopCode: char); external;
00260      1  0
00261      2  0 procedure StoreRegPtr(SRPWidth: integer;
00262      2  0           SRPDesignator: integer;
00263      2  0           SRPValue: unsigned;
00264      1  0           var SRPStopCode: char); external;
00265      1  0
00266      2  0 procedure FetchOperand(FOType: char;
00267      2  0           FOWidth: integer;
00268      2  0           FOAddress: unsigned;
00269      2  0           FOSegment: integer;
00270      2  0           var FOValue: unsigned;
00271      1  0           var FOStopCode: char); external;
00272      1  0
00273      2  0 procedure FetchCodeData(FCDWidth: integer;
00274      2  0           var FCDValue: unsigned;
00275      1  0           var FCDStopCode: char); external;
00276      1  0
00277      2  0 procedure PopFlags( PFFlags: FlagType;
00278      1  0           var PFStopCode: char); external;
00279      1  0
00280      1  0 procedure PushIntersegRA( var PIRAStopCode: char); external;
00281      1  0
00282      1  0 procedure PopIntersegRA( var PIRAStopCode: char); external;
00283      1  0
00284      2  0 procedure Interrupt(IInterruptType: unsigned;
00285      1  0           var IStopCode: char); external;
```

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

00287      1  0 function SignExtend7(SEValue: unsigned): integer; external;
00288      1  0
00289      1  0 function SignExtend15(SEValue: unsigned): integer; external;
00290      1  0
00291      2  0 function XOR(FA: boolean;
00292      2  0           FB: boolean): boolean;
00293      2  0
00294      2  1 begin
00295      2  1     if (FA = FB) then XOR := false
00296      2  1     else XOR := true;
00297      1  0 end;
```

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

00299       1  1 begin
00300       1  1     OpcodeKey := int(Opcode.Full);
00301       1  1
00302       1  2     case OpcodeKey of
00303       1  2
00304       1  2     %X'70': {Jump on Overflow}
00305       1  3       begin
00306       1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00307       1  3   if (Flags.Overflow) then
00308       1  3       IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00309       1  2       end;
00310       1  2
00311       1  2     %X'71': {Jump on not Overflow}
00312       1  3       begin
00313       1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00314       1  3   if (not Flags.Overflow) then
00315       1  3       IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00316       1  2       end;
00317       1  2
00318       1  2
00319       1  2     %X'72': {Jump on Below, Jump on not Above or Equal, Jump on Carry}
00320       1  3       begin
00321       1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00322       1  3   if (Flags.Carry) then
00323       1  3       IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00324       1  2       end;
00325       1  2
00326       1  2
00327       1  2     %X'73': {Jump on Above or Equal, Jump on not Below, Jump on not Carry}
00328       1  3       begin
00329       1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00330       1  3   if (not Flags.Carry) then
00331       1  3       IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00332       1  2       end;
00333       1  2
00334       1  2
00335       1  2     %X'74': {Jump on Equal, Jump on Zero}
00336       1  3       begin
00337       1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00338       1  3   if (Flags.Zero) then
00339       1  3       IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00340       1  2       end;
00341       1  2
00342       1  2
```

```
00343      1  2     %X'75': {Jump on not Equal, Jump on not Zero}
00344      1  3       begin
00345      1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00346      1  3   if (not Flags.Zero) then
00347      1  3       IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00348      1  2       end;
```

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

00350       1  2     %X'76': {Jump on Below or Equal, Jump on not Above}
00351       1  3       begin
00352       1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00353       1  3   if ((Flags.Carry) or (Flags.Zero)) then
00354       1  3       IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00355       1  2       end;
00356       1  2
00357       1  2
00358       1  2     %X'77': {Jump on Above, Jump on not Below or Equal}
00359       1  3       begin
00360       1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00361       1  3   if (not ((Flags.Carry) or (Flags.Zero))) then
00362       1  3       IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00363       1  2       end;
00364       1  2
00365       1  2
00366       1  2     %X'78': {Jump on Sign}
00367       1  3       begin
00368       1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00369       1  3   if (Flags.Sign) then
00370       1  3       IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00371       1  2       end;
00372       1  2
00373       1  2
00374       1  2     %X'79': {Jump on not Sign}
00375       1  3       begin
00376       1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00377       1  3   if (not Flags.Sign) then
00378       1  3       IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00379       1  2       end;
00380       1  2
00381       1  2
00382       1  2     %X'7A': {Jump on Parity, Jump on Parity Equal}
00383       1  3       begin
00384       1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00385       1  3   if (Flags.Parity) then
00386       1  3       IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00387       1  2       end;
00388       1  2
00389       1  2
00390       1  2     %X'7B': {Jump on not Parity, Jump on Parity Odd}
00391       1  3       begin
00392       1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00393       1  3   if (not Flags.Parity) then
```

```
00394      1  3        IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00395      1  2        end;
```

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

00397       1  2     %X'7C': {Jump on Less, Jump on not Greater or Equal}
00398       1  3       begin
00399       1  3    FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00400       1  3    if (XOR(Flags.Sign,Flags.Overflow)) then
00401       1  3        IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00402       1  2        end;
00403       1  2
00404       1  2
00405       1  2     %X'7D': {Jump on Greater or Equal, Jump on not Less}
00406       1  3       begin
00407       1  3    FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00408       1  3    if (Flags.Sign = Flags.Overflow) then
00409       1  3        IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00410       1  2        end;
00411       1  2
00412       1  2
00413       1  2     %X'7E': {Jump on Less or Equal, Jump on not Greater}
00414       1  3       with Flags do begin
00415       1  3    FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00416       1  3    if ( (XOR( Sign, Overflow)) or Zero) then
00417       1  3        IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00418       1  2        end;
00419       1  2
00420       1  2
00421       1  2     %X'7F': {Jump on Greater, Jump on not Less or Equal}
00422       1  3       with Flags do begin
00423       1  3    FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00424       1  3    if ( not((XOR( Sign, OverFlow)) or Zero)) then
00425       1  3        IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00426       1  2        end;
```

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

00428       1  2     %X'9A': {call Direct Intersegment}
00429       1  3       begin
00430       1  3   CTStopCode := Call;
00431       1  3   FetchCodeData(IPLength, NewIP, CTStopCode);
00432       1  3   FetchCodeData(CSLength, NewCS, CTStopCode);
00433       1  3   PushIntersegRA( CTStopCode);
00434       1  3   IP := NewIP;
00435       1  3   StoreRegPtr(CSWidth, CSID, NewCS, CTStopCode);
00436       1  2       end;
00437       1  2
00438       1  2     %X'C2': {Intra Segment Return and Add Immediate to Stack Pointer}
00439       1  3       begin
00440       1  3   CTStopCode := Return;
00441       1  3   FetchCodeData(PopValueLength, PopValue, CTStopCode);
00442       1  3   PopFromStack( IP, CTStopCode);
00443       1  3   FetchRegPtr(SPWidth, SPID, StackPointer, CTStopCode);
00444       1  3   StackPointer := uint( int(StackPointer) + SignExtend15(PopValue));
00445       1  3   StoreRegPtr(SPWidth, SPID, StackPointer, CTStopCode);
00446       1  2       end;
00447       1  2
00448       1  2     %X'C3': {Return Intrasegment}
00449       1  3       begin
00450       1  3   CTStopCode := Return;
00451       1  3   PopFromStack( IP, CTStopCode);
00452       1  2       end;
00453       1  2
00454       1  2     %X'CA': {InterSegment Return and ADD Immediate to Stack Pointer}
00455       1  3       begin
00456       1  3   CTStopCode := Return;
00457       1  3   FetchCodeData(PopValueLength, PopValue, CTStopCode);
00458       1  3   PopIntersegRA( CTStopCode);
00459       1  3   FetchRegPtr(SPWidth, SPID, StackPointer, CTStopCode);
00460       1  3   StackPointer := uint( int(StackPointer) + SignExtend15(PopValue));
00461       1  3   StoreRegPtr(SPWidth, SPID, StackPointer, CTStopCode);
00462       1  2       end;
00463       1  2
00464       1  2     %X'CB': {InterSegment Return}
00465       1  3       begin
00466       1  3   CTStopCode := Return;
00467       1  3   PopIntersegRA( CTStopCode);
00468       1  2       end;
```

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

00470       1  2     %X'CC': {Type 3 Interrupt}
00471       1  3       begin
00472       1  3   CTStopCode := BreakPoint;
00473    C  1  3 { Interrupt(Type3Interrupt);}
00474       1  2       end;
00475       1  2
00476       1  2     %X'CD': {Interrupt of Specified Type}
00477       1  3       begin
00478       1  3   FetchCodeData(InterruptWidth, InterruptType, CTStopCode);
00479       1  3   Interrupt(InterruptType, CTStopCode);
00480       1  2       end;
00481       1  2
00482       1  2     %X'CE': {Interrupt on Overflow}
00483       1  2   if (Flags.Overflow = SetHigh) then Interrupt(OverflowInterrupt, CTStopCode);
00484       1  2
00485       1  2     %X'CF': {Interrupt Return}
00486       1  3       begin
00487       1  3   PopIntersegRA( CTStopCode);
00488       1  3   PopFlags(Flags, CTStopCode);
00489       1  2       end;
```

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

00491       1  2     %X'E0': {Loop While not Zero, Loop While not Equal}
00492       1  3       begin
00493       1  3   FetchRegPtr(CXWidth, CXID, CXValue, CTStopCode);
00494       1  3   CXValue := CXValue - 1;
00495       1  3   StoreRegPtr( CXWidth, CXID, CXValue, CTStopCode);
00496       1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00497       1  3   if ((not(Flags.Zero)) and (CXValue <> 0)) then
00498       1  3       IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00499       1  2       end;
00500       1  2
00501       1  2     %X'E1': {Loop While Equal, Loop While Zero}
00502       1  3       begin
00503       1  3   FetchRegPtr(CXWidth, CXID, CXValue, CTStopCode);
00504       1  3   CXValue := CXValue - 1;
00505       1  3   StoreRegPtr( CXWidth, CXID, CXValue, CTStopCode);
00506       1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00507       1  3   if ((Flags.Zero) and (CXValue <> 0)) then
00508       1  3       IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00509       1  2       end;
00510       1  2
00511       1  2     %X'E2': {loop}
00512       1  3       begin
00513       1  3   FetchRegPtr(CXWidth, CXID, CXValue, CTStopCode);
00514       1  3   CXValue := CXValue - 1;
00515       1  3   StoreRegPtr( CXWidth, CXID, CXValue, CTStopCode);
00516       1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00517       1  3   if (CXValue <> 0) then
00518       1  3       IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00519       1  2       end;
00520       1  2
00521       1  2     %X'E3': {Jump if CX is Zero}
00522       1  3       begin
00523       1  3   FetchRegPtr(CXWidth, CXID, CXValue, CTStopCode);
00524       1  3   FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00525       1  3   if (CXValue = 0) then
00526       1  3       IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00527       1  2       end;
```

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

00529       1  2     %X'E8': {Call IntraSegmentDirect}
00530       1  3        begin
00531       1  3    CTStopCode := Call;
00532       1  3    FetchCodeData(IntraSegmentDisplacementWidth, IntraSegmentDisplacement, CTStopCode);
00533       1  3    PushOnStack(IP, CTStopCode);
00534       1  3    IntDisplacement := SignExtend15(IntraSegmentDisplacement);
00535       1  3    IP := uand( uint( int(IP) + IntDisplacement), All16Bits);
00536       1  2        end;
00537       1  2
00538       1  2     %X'E9': {Jump IntraSegmentDirect}
00539       1  3        begin
00540       1  3    FetchCodeData(IntraSegmentDisplacementWidth, IntraSegmentDisplacement, CTStopCode);
00541       1  3    IntDisplacement := SignExtend15(IntraSegmentDisplacement);
00542       1  3    IP := uand( uint( int(IP) + IntDisplacement), All16Bits);
00543       1  2        end;
00544       1  2
00545       1  2     %X'EA': {Jump InterSegment Direct}
00546       1  3        begin
00547       1  3    FetchCodeData(IPLength, NewIP, CTStopCode);
00548       1  3    FetchCodeData(CSLength, NewCS, CTStopCode);
00549       1  3    IP := NewIP;
00550       1  3    StoreRegPtr(CSWidth, CSID, NewCS, CTStopCode);
00551       1  2        end;
00552       1  2
00553       1  2     %X'EB': {Jump IntraSegmentDirect Short}
00554       1  3        begin
00555       1  3    FetchCodeData(DisplacementWidth, Displacement, CTStopCode);
00556       1  3    IP := uand( uint( int(IP) + SignExtend7(Displacement)), All16Bits);
00557       1  2        end;
```

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

00559       1  2     %X'2FF': {Call IntraSegment Indirect}
00560       1  3       with EA do begin
00561       1  3    CTStopCode := Call;
00562       1  3    PushOnStack(IP, CTStopCode);
00563       1  3    FetchOperand(Mode, SixteenBits, Address, Segment, IP, CTStopCode);
00564       1  2        end;
00565       1  2
00566       1  2     %X'3FF': {Call Indirect InterSegment}
00567       1  3       with EA do begin
00568       1  3    CTStopCode := Call;
00569       1  3    PushIntersegRA( CTStopCode);
00570       1  3    FetchOperand(Mode, SixteenBits, Address, Segment, NewIP, CTStopCode);
00571       1  3    FetchOperand(Mode, SixteenBits, (Address + 2), Segment, NewCS, CTStopCode);
00572       1  3    IP := NewIP;
00573       1  3    StoreRegPtr(CSWidth, CSID, NewCS, CTStopCode);
00574       1  2        end;
00575       1  2
00576       1  2     %X'4FF': {Jump Indirect IntraSegment}
00577       1  2       with EA do
00578       1  2    FetchOperand(Mode, SixteenBits, Address, Segment, IP, CTStopCode);
00579       1  2
00580       1  2     %X'5FF': {Jump Indirect InterSegment}
00581       1  3       with EA do begin
00582       1  3    FetchOperand(Mode, SixteenBits, Address, Segment, NewIP, CTStopCode);
00583       1  3    FetchOperand(Mode, SixteenBits, (Address + 2), Segment, NewCS, CTStopCode);
00584       1  3    IP := NewIP;
00585       1  3    StoreRegPtr(CSWidth, CSID, NewCS, CTStopCode);
00586       1  3        end
00587       1  3
00588       1  2     otherwise CTStopCode := BadOpcode;
00589       1  2
00590       1  1     end; {case OpcodeKey}
00591       0  0 end;
00592       0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	3387	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 CONTROLTRANSFER.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$PROGRAM2:[CHAPMAN.THESIS.SOURCE]CONTROLTRANSFER.LIS;1
/OBJECT=SYS$PROGRAM2:[CHAPMAN.THESIS.SOURCE]CONTROLTRANSFER.OBJ;1
/NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE /NOOLD_VERSION /WARNINGS
```

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	429	00:00.5	00:03.6
Source Analysis	314	00:02.0	00:07.8
Source Listing	14	00:00.9	00:03.4
Tree Construction	237	00:00.6	00:00.8
Flow Analysis	105	00:00.4	00:00.5
Value Propagation	12	00:00.1	00:00.1
Profit Analysis	49	00:00.2	00:00.3
Context Analysis	256	00:03.6	00:08.0
Name Packing	6	00:00.1	00:00.1
Code Selection	104	00:00.6	00:01.1
Final	62	00:01.6	00:02.1
TOTAL	1594	00:10.4	00:27.7

COMPILATION STATISTICS

CPU Time: 00:10.4 (3402 Lines/Minute)

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

00001    C  0  0 {
00002    C  0  0   Title: DataTransfer
00003    C  0  0
00004    C  0  0   Purpose: Contains the software to execute the Data Transfer instructions
00005    C  0  0
00006    C  0  0   Author: William A. Chapman      Date: January 26, 1986
00007    C  0  0
00008    C  0  0   Inputs: Opcode record
00009    C  0  0           Register record
00010    C  0  0           Effective Address record
00011    C  0  0
00012    C  0  0   Outputs: Memory and register stores are manipulated as requested
00013    C  0  0           for each instruction
00014    C  0  0
00015    C  0  0   Procedures Invoked: FetchRegPtr
00016    C  0  0           StoreRegPtr
00017    C  0  0           FetchOperand
00018    C  0  0           StoreData
00019    C  0  0           FetchCode
00020    C  0  0           FetchCodeData
00021    C  0  0           PushOnStack
00022    C  0  0           PopFromStack
00023    C  0  0           PushFlags
00024    C  0  0           PopFlags
00025    C  0  0           FetchInput
00026    C  0  0           StoreOutput
00027    C  0  0 }
00028       0  0 module DataTransfer(input, output );
00029       0  0
00030       0  0 const
00031       0  0
00032       0  0     %include  [-]Const.defn/list'
00033 I     0  0     FirstOpcodeValue = 0;
00034 I     0  0     LastOpcodeValue = 255;
00035 I     0  0     All16Bits = %X'FFFF';          {mask for all 16 bits}
00036 I     0  0     Low8Bits = %X'FF';               {mask for low 8 bits}
00037 I     0  0     High8Bits = %X'FF00';          {mask for high 8 bits}
00038 I     0  0     Bit8Multiplier = %X'100';
00039 I     0  0     Bit8Divisor    = %X'100';
00040 I     0  0     WordMultiplier = %X'100';
00041 I     0  0
00042 I     0  0     EightBits = 0;                          {width for 8 bits}
00043 I     0  0     SixteenBits = 1;                        {width for 16 bits}
00044 I     0  0
00045 I     0  0     SimpleMode = %B'0';
```

```
00046 I     0  0     SimpleRM = %B'110';
00047 I     0  0
00048 I     0  0     LowMemoryLimit = 0;                    {low limit on memory array}
00049 I     0  0     HighMemoryLimit = 2048;                {high limit on memory array}
00050 I     0  0
00051 I     0  0     FilenameLength = 40;
00052       0  0
00053       0  0     %include '[-]Flags.defn/list'
00054 I     0  0     SetHigh = true;
00055 I     0  0     Clear = false;
```

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

00056 I     0  0
00057 I     0  0      CarryFlag     = %B'000000000001';
00058 I     0  0      ParityFlag    = %B'000000000100';
00059 I     0  0      AuxCarryFlag  = %B'000000010000';
00060 I     0  0      ZeroFlag      = %B'000001000000';
00061 I     0  0      SignFlag      = %B'000010000000';
00062 I     0  0      TrapFlag      = %B'000100000000';
00063 I     0  0      InterruptFlag = %B'001000000000';
00064 I     0  0      DirectionFlag = %B'010000000000';
00065 I     0  0      OverflowFlag  = %B'100000000000';
00066       0  0      %include '[-]RegId.defn/list'
00067 I     0  0      ALId = %B'000';
00068 I     0  0      ALWidth = 0;
00069 I     0  0
00070 I     0  0      CLId = %B'001';
00071 I     0  0      CLWidth = 0;
00072 I     0  0
00073 I     0  0      DLId = %B'010';
00074 I     0  0      DLWidth = 0;
00075 I     0  0
00076 I     0  0      BLId = %B'011';
00077 I     0  0      BLWidth = 0;
00078 I     0  0
00079 I     0  0      AHId = %B'100';
00080 I     0  0      AHWidth = 0;
00081 I     0  0
00082 I     0  0      CHId = %B'101';
00083 I     0  0      CHWidth = 0;
00084 I     0  0
00085 I     0  0      DHId = %B'110';
00086 I     0  0      DHWidth = 0;
00087 I     0  0
00088 I     0  0      BHId = %B'111';
00089 I     0  0      BHWidth = 0;
00090 I     0  0
00091 I     0  0      AXId = %B'000';
00092 I     0  0      AXWidth = 1;
00093 I     0  0      ALorAXId = %B'000';
00094 I     0  0
00095 I     0  0      CXId = %B'001';
00096 I     0  0      CXWidth = 1;
00097 I     0  0
00098 I     0  0      DXId = %B'010';
00099 I     0  0      DXWidth = 1;
```

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

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

00111 I     0  0      SIWidth = 1;
00112 I     0  0
00113 I     0  0      DIId = %B'111';
00114 I     0  0      DIWidth = 1;
00115 I     0  0
00116 I     0  0      SRWidth = 2;
00117 I     0  0
00118 I     0  0      ESId = %B'00';
00119 I     0  0      ESWidth = 2;
00120 I     0  0
00121 I     0  0      CSId = %B'01';
00122 I     0  0      CSWidth = 2;
00123 I     0  0
00124 I     0  0      SSId = %B'10';
00125 I     0  0      SSWidth = 2;
00126 I     0  0
00127 I     0  0      DSId = %B'11 ;
00128 I     0  0      DSWidth = 2;
00129 I     0  0
00130       0  0      AltWidthMask = %B'1000';
00131       0  0
00132       0  0      ClearData = 0;
00133       0  0
00134       0  0      RegisterFlag = 'R';
00135       0  0      MemoryFlag = 'M';
00136       0  0
00137       0  0      %include  [-]StopCode.defn/list'
00138 I     0  0      Breakpoint              = 'B';
00139 I     0  0      Call          = 'C';
00140 I     0  0      ControlD              = 'D';
00141 I     0  0      BadOpcode               = 'E';
00142 I     0  0      BadCEAModeValue         = 'F';
00143 I     0  0      BadRMDModeValue         = 'G';
00144 I     0  0      Halt          = 'H';
00145 I     0  0      BadRegisterID = 'I';
00146 I     0  0      BadOpcodeKey  = 'K';
00147 I     0  0      BadMemoryAddress        = 'M';
00148 I     0  0      Normal                  = 'N';
00149 I     0  0      BadPortAddress          = 'P';
00150 I     0  0      BadOpcodeClass          = 'Q';
00151 I     0  0      Return                  = 'R';
00152 I     0  0      BadCheckSum             = 'S';
00153 I     0  0      BadOperandType          = 'T';
00154 I     0  0      BadMemoryWidth          = 'W';
```

```
00155 I     0  0      BadOpcodeExtension    = 'X';
00156 I     0  0      NoMemoryAccess        = 'Z';
```

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

00158       0  0 type
00159       0  0     %include  [-]Type.defn/list'
00160 I     0  0     ZeroOne = -1..1;                          {defines integer with range
00161 I C   0  0                                               of zero to one with an invalid
00162 I C   0  0                                               state of -1}
00163 I     0  0
00164 I     0  0     OpcodeLUTEntry = record                   {defines entries in opcode LUT}
00165 I     0  0   OpcodeClass: char;                         {class D, A, L, S, C, P}
00166 I     0  0   OpcodeKey: integer;                        {key to interpret opcode from
00167 I C   0  0                                               0 to 7}
00168 I     0  0   DirectionBitPresent: boolean;
00169 I     0  0   WidthBitPresent: boolean
00170 I     0  0     end;
00171 I     0  0
00172 I     0  0     OpcodeType = record
00173 I     0  0   Direction: ZeroOne;                        {to or from CPU}
00174 I     0  0   Full: unsigned;                            {full opcode}
00175 I     0  0   Width: ZeroOne                            {8 or 16 bits}
00176 I     0  0     end;
00177 I     0  0
00178 I     0  0     RegisterType = record
00179 I     0  0   Id: integer;                              {identifier}
00180 I     0  0   Width: ZeroOne                            {8 or 16 bits}
00181 I     0  0     end;
00182 I     0  0
00183 I     0  0     EffectiveAddressType = record
00184 I     0  0   Mode: char;                               {register or memory}
00185 I     0  0   Width: ZeroOne;                           {8 or 16 bits}
00186 I     0  0   Address: unsigned;                       {memory address or
00187 I C   0  0                                               register designation}
00188 I     0  0   Segment: integer                          {segment register to use}
00189 I     0  0     end;
00190 I     0  0
00191 I     0  0     NameType = packed array[1..10] of char;
00192 I     0  0
00193 I     0  0     Characters = set of    ..'~';
00194 I     0  0     LettersAndNumbers = set of '0'..'z';
00195 I     0  0     Letters = set of 'A'..'z';
00196 I     0  0     Numbers = set of '0'. '9';
00197 I     0  0
00198 I     0  0     FileName = packed array [1..FilenameLength] of char;
00199       0  0
00200       0  0     %include '[-]FlagType.defn/list
00201 I     0  0     FlagType = record                         {define processor flags}
```

```
00202 I     0  0    Carry: boolean;
00203 I     0  0    Parity: boolean;
00204 I     0  0    AuxCarry: boolean;
00205 I     0  0    Zero: boolean;
00206 I     0  0    Sign: boolean;
00207 I     0  0    Trap: boolean;
00208 I     0  0    Interrupt: boolean;
00209 I     0  0    Direction: boolean;
00210 I     0  0    Overflow: boolean
00211 I     0  0      end;
```

```
-LINE IDC-PL SL-

00213       0  0 var
00214       0  0     StackData: unsigned;
00215       0  0     TempData: unsigned;
00216       0  0
00217       0  0     SourceData: unsigned;
00218       0  0     RegisterData: unsigned;
00219       0  0
00220       0  0     AllData: unsigned;
00221       0  0
00222       0  0     ALValue: unsigned;
00223       0  0     BXValue: unsigned;
00224       0  0
00225       0  0     PortAddress: unsigned;
00226       0  0     InputData: unsigned;
00227       0  0     OutputData: unsigned;
00228       0  0
00229       0  0     Flags: [external] FlagType;
00230       0  0
00231       0  0     Address: unsigned;
00232       0  0     SegmentRegister: integer;
00233       0  0
00234       0  0     OpcodeKey: integer;
00235       0  0
00236       0  0     SegmentOverRideCount: [external] integer;
00237       0  0     SegmentOverRideValue: [external] integer;
00238       0  0
00239       0  0
00240       0  0
00241       0  0
00242       0  0
00243       1  0 [global] procedure DataTransfer(Opcode: OpcodeType;
00244       1  0                 Register: RegisterType;
00245       1  0                 EA: EffectiveAddressType;
00246       1  0                 var PTStopCode: char);
```

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

00248       2  0 procedure FetchRegPtr(FRPWidth: integer;
00249       2  0           FRPDesignator: integer;
00250       2  0           var FRPValue: unsigned;
00251       1  0           var FRPStopCode: char); external;
00252       1  0
00253       2  0 procedure PushOnStack(POSValue: unsigned;
00254       1  0           var POSStopCode: char); external;
00255       1  0
00256       2  0 procedure PopFromStack(var PFPValue: unsigned;
00257       1  0           var PFSStopCode: char); external;
00258       1  0
00259       2  0 procedure StoreRegPtr(SRPWidth: integer;
00260       2  0           SRPDesignator: integer;
00261       2  0           SRPValue: unsigned;
00262       1  0           var SRPStopCode: char); external;
00263       1  0
00264       2  0 procedure FetchOperand(FOType: char;
00265       2  0           FOWidth: integer;
00266       2  0           FOAddress: unsigned;
00267       2  0           FOSegment: integer;
00268       2  0           var FOValue: unsigned;
00269       1  0           var FOStopCode: char); external;
00270       1  0
00271       2  0 procedure StoreData(SDType: char;
00272       2  0           SDWidth: integer;
00273       2  0           SDAddress: unsigned;
00274       2  0           SDSegment: integer;
00275       2  0           SDValue: unsigned;
00276       1  0           var SDStopCode: char); external;
00277       1  0
00278       2  0 procedure FetchCodeData(FCDWidth: integer;
00279       2  0           var FCDValue: unsigned;
00280       1  0           var FCDStopCode: char); external;
00281       1  0
00282       2  0 procedure PushFlags(PFFlags: FlagType;
00283       1  0           var PFStopCode: char); external;
00284       1  0
00285       2  0 procedure PopFlags(PFFlags: FlagType;
00286       1  0           var PFStopCode: char); external;
00287       1  0
00288       2  0 procedure FetchInput(FIWidth: integer;
00289       2  0           FIPortAddress: unsigned;
00290       2  0           var FIData: unsigned;
00291       1  0           var FIStopCode: char); external;
```

```
00292     1  0
00293     2  0 procedure StoreOutput(SOWidth: integer;
00294     2  0           SOPortAddress: unsigned;
00295     2  0           SOData: unsigned;
00296     1  0           var SOStopCode: char); external;
00297     1  0
00298     2  0 procedure FetchCode(var FCData: unsigned;
00299     1  0           var FCStopCode: char); 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'6',%x'50': {push 16 bit register or segment register}
00307       1  3     with Register do begin
00308       1  3       FetchRegPtr( Width, Id, StackData, DTStopCode);
00309       1  3       PushOnStack( StackData, DTStopCode);
00310       1  2     end;
00311       1  2
00312       1  2   %X'7',%X'58': {pop 16 bit register or segment register}
00313       1  3     with Register do begin
00314       1  3       if ((Width = SRWidth) and (Id = CSId)) then DTStopCode := BadOpcode
00315       1  4       else begin
00316       1  4           PopFromStack( StackData, DTStopCode);
00317       1  4           StoreRegPtr(Width,Id,StackData, DTStopCode);
00318       1  3       end;
00319       1  2     end;
00320       1  2
00321       1  2   %X'86': {exchange R/M with register}
00322       1  3     with Register do begin
00323       1  3       FetchRegPtr(Width,Id,TempData, DTStopCode);
00324       1  3       FetchOperand( EA.Mode, EA.Width, EA.Address, EA.Segment, SourceData, DTStopCode);
00325       1  3       StoreRegPtr(Width,Id,SourceData, DTStopCode);
00326       1  3       StoreData(EA.Mode,EA.Width,EA.Address,EA.Segment,TempData, DTStopCode);
00327       1  2     end;
00328       1  2
00329       1  2   %X'88': {move register / memory  to / from register}
00330       1  2     with EA do
00331       1  3       if (Opcode.Direction = 1) then begin
00332       1  3           FetchOperand(Mode,Width,Address,Segment,SourceData, DTStopCode);
00333       1  3           StoreRegPtr(Register.Width,Register Id,SourceData, DTStopCode);
00334       1  3       end
00335       1  3       else begin
00336       1  3           FetchRegPtr(Register.Width,Register Id,SourceData, DTStopCode);
00337       1  3           StoreData(Mode, Width, Address, Segment, SourceData, DTStopCode);
00338       1  2       end; {else begin and with}
00339       1  2
00340       1  2   %X'8C': {move segment register to register / memory}
00341       1  3     with EA do begin
00342       1  3       FetchRegPtr(Register.Width,Register Id,RegisterData, DTStopCode);
00343       1  3       StoreData(Mode, Width, Address, Segment, RegisterData, DTStopCode);
00344       1  2     end;
```

```
00345     1  2
00346     1  2   %X'8D': {load Effective address to register}
00347     1  2     with EA do
00348     1  2       if (Mode = RegisterFlag) then DTStopCode := BadOpcode
00349     1  2       else StoreRegPtr(SixteenBits,Register.Id,Address, DTStopCode);
```

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

00351       1  2   %X'8E': {move register / memory to segment register}
00352       1  3     with EA do begin
00353       1  3       if (Register.Id = CSId) then DTStopCode := BadOpcode
00354       1  4       else begin
00355       1  4           FetchOperand(Mode, Width, Address, Segment, RegisterData, DTStopCode);
00356       1  4           StoreRegPtr(Register.Width, Register.Id, RegisterData, DTStopCode);
00357       1  3       end;
00358       1  2     end;
00359       1  2
00360       1  2   %X'8F': {pop register / memory from stack}
00361       1  3     with EA do begin
00362       1  3       Width := SixteenBits;       {force this to 16 bits}
00363       1  3       PopFromStack( StackData, DTStopCode);
00364       1  3       StoreData(Mode, Width, Address, Segment, StackData, DTStopCode);
00365       1  2     end;
00366       1  2
00367       1  2   %X'90': {exchange 16 bit register with accumulator}
00368       1  3     with Register do begin
00369       1  3       FetchRegPtr(Width, Id, RegisterData, DTStopCode);
00370       1  3       FetchRegPtr(AXWidth, AXId, TempData, DTStopCode);
00371       1  3       StoreRegPtr(AXWidth, AXId, RegisterData, DTStopCode);
00372       1  3       StoreRegPtr(Width, Id, TempData, DTStopCode);
00373       1  2     end;
00374       1  2
00375       1  2   %X'9C': PushFlags( Flags, DTStopCode);
00376       1  2
00377       1  2   %X'9D': PopFlags( Flags, DTStopCode);
00378       1  2
00379       1  2   %X'9E': {store AH with flags "S Z x A x P x C" }
00380       1  3     with Flags do begin
00381       1  3       FetchRegPtr(AHWidth, AHId, AHData, DTStopCode);
00382       1  3       if (uand(AHData, CarryFlag) = 0) then Carry := Clear
00383       1  3           else Carry := SetHigh;
00384       1  3       if (uand(AHData, ParityFlag) = 0) then Parity := Clear
00385       1  3           else Parity := SetHigh;
00386       1  3       if (uand(AHData, AuxCarryFlag) = 0) then AuxCarry := Clear
00387       1  3           else AuxCarry := SetHigh;
00388       1  3       if (uand(AHData, ZeroFlag) = 0) then Zero := Clear
00389       1  3           else Zero := SetHigh;
00390       1  3       if (uand(AHData, SignFlag) = 0) then Sign := Clear
00391       1  3           else Sign := SetHigh;
00392       1  2     end; {with Flags}
00393       1  2
00394       1  2   %X'9F': {load AH with flags "S Z x A x P x C"}
```

```
00395     1  3     with Flags do begin
00396     1  3       AHData := ClearData;
00397     1  3       if (Carry = SetHigh) then AHData := uor(AHData, CarryFlag);
00398     1  3       if (Parity = SetHigh) then AHData := uor(AHData, ParityFlag);
00399     1  3       if (AuxCarry = SetHigh) then AHData := uor(AHData, AuxCarryFlag);
00400     1  3       if (Zero = SetHigh) then AHData := uor(AHData, ZeroFlag);
00401     1  3       if (Sign = SetHigh) then AHData := uor(AHData, SignFlag);
00402     1  3       StoreRegPtr(AHWidth, AHId, AHData, DTStopCode);
00403     1  2     end;
```

```
-LINE-IDC PL-SL-

00405       1  2   %X'A0': {move memory to accumulator}
00406       1  3     with EA do begin
00407       1  3       FetchOperand(Mode, Width, Address, Segment, RegisterData, DTStopCode);
00408       1  3       StoreRegPtr(Register.Width, AlorAXId, RegisterData, DTStopCode);
00409       1  2     end;
00410       1  2
00411       1  2   %X'A2': {move accumulator to memory}
00412       1  3     with EA do begin
00413       1  3       FetchRegPtr(Register.Width, AlorAXId, RegisterData, DTStopCode);
00414       1  3       StoreData(Mode, Width, Address, Segment, RegisterData, DTStopCode);
00415       1  2     end;
00416       1  2
00417       1  2   %X'B0',%X'B8': {move immediate to register}
00418       1  3     with Register do begin
00419       1  3       if (uand(Opcode.Full,AltWidthMask) = 0) then Width := EightBits
00420       1  3           else Width := SixteenBits;
00421       1  3       FetchCodeData(Width, RegisterData, DTStopCode);
00422       1  3       StoreRegPtr(Width, Id, RegisterData, DTStopCode);
00423       1  2     end;
00424       1  2
00425       1  2   %X'C4': {load pointer using ES}
00426       1  3     with EA do begin
00427       1  3       if (Mode   RegisterFlag) then DTStopCode := BadOpcode
00428       1  4       else begin
00429       1  4           FetchOperand(Mode, SixteenBits, Address, Segment, RegisterData, DTStopCode);
00430       1  4           StoreRegPtr(SixteenBits, Register.Id, RegisterData, DTStopCode);
00431       1  4           FetchOperand(Mode, SixteenBits, Address + 2, Segment, RegisterData, DTStopCode);
00432       1  4           StoreRegPtr(SRWidth, ESId, RegisterData, DTStopCode);
00433       1  3       end;
00434       1  2     end;
00435       1  2
00436       1  2
00437       1  2   %X'C5': {load pointer using DS}
00438       1  3     with EA do begin
00439       1  3       if (Mode = RegisterFlag) then DTStopCode := BadOpcode
00440       1  4       else begin
00441       1  4           FetchOperand(Mode, SixteenBits, Address, Segment, RegisterData, DTStopCode);
00442       1  4           StoreRegPtr(SixteenBits, Register.Id, RegisterData, DTStopCode);
00443       1  4           FetchOperand(Mode, SixteenBits, Address + 2, Segment, RegisterData, DTStopCode);
00444       1  4           StoreRegPtr(SRWidth, DSId, RegisterData, DTStopCode);
00445       1  3       end;
00446       1  2     end;
00447       1  2
00448       1  2   %X'C6': {move immediate to register / memory}
```

```
00449      1  3     with EA do begin
00450      1  3       FetchCodeData(Width, SourceData, DTStopCode);
00451      1  3       StoreData(Mode, Width, Address, Segment, SourceData, DTStopCode);
00452      1  2     end;
```

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

00454       1  2   %X'D7': {translate AL}
00455       1  3     begin
00456       1  3       FetchRegPtr(ALWidth, ALID, ALValue, DTStopCode);
00457       1  3       FetchRegPtr(BXWidth, BXID, BXValue, DTStopCode);
00458       1  3       Address := uint(int(BXValue) + int(ALValue));
00459       1  3       if ( SegmentOverRideCount > 0) then SegmentRegister := SegmentOverRideValue
00460       1  3       else SegmentRegister := DSID;
00461       1  3       FetchOperand(MemoryFlag,ALWidth, Address, SegmentRegister, ALValue, DTStopCode);
00462       1  3       StoreRegPtr(ALWidth, ALID, ALValue, DTStopCode);
00463       1  2     end;
00464       1  2
00465       1  2   %X'E4': {input fixed port}
00466       1  3     with Opcode do begin
00467       1  3       FetchCode(PortAddress, DTStopCode);
00468       1  3       FetchInput(Width, PortAddress, InputData, DTStopCode);
00469       1  3       StoreRegPtr(Width, AlorAXId, InputData, DTStopCode);
00470       1  2     end;
00471       1  2
00472       1  2   %X'E6': {output fixed port}
00473       1  3     with Opcode do begin
00474       1  3       FetchCode(PortAddress, DTStopCode);
00475       1  3       FetchRegPtr(Width, AlorAXId, OutputData, DTStopCode);
00476       1  3       StoreOutput(Width, PortAddress, OutputData, DTStopCode);
00477       1  2     end;
00478       1  2
00479       1  2   %X'EC': {input variable port}
00480       1  3     with Opcode do begin
00481       1  3       FetchRegPtr(SixteenBits, DXId, PortAddress, DTStopCode);
00482       1  3       FetchInput(Width, PortAddress, InputData, DTStopCode);
00483       1  3       StoreRegPtr(Width, AlorAXId, InputData, DTStopCode);
00484       1  2     end;
00485       1  2
00486       1  2   %X'EE': {output variable port}
00487       1  3     with Opcode do begin
00488       1  3       FetchRegPtr(SixteenBits, DXId, PortAddress, DTStopCode);
00489       1  3       FetchRegPtr(Width, AlorAXId, OutputData, DTStopCode);
00490       1  3       StoreOutput(Width, PortAddress, OutputData, DTStopCode);
00491       1  2     end;
00492       1  2
00493       1  2   %X'6FF': {push register / memory}
00494       1  3     with EA do begin
00495       1  3       FetchOperand(Mode, SixteenBits, Address, Segment, StackData, DTStopCode);
00496       1  3       PushOnStack( StackData, DTStopCode);
00497       1  3     end
```

```
00498      1  3
00499      1  2   otherwise DTStopCode := BadOpcode;
00500      1  2
00501      1  1     end; {case Opcode.Full}
00502      0  0 end;
00503      0  0 end.
```

PSECT SUMMARY

Name	Bytes	Attributes
$CODE	2464	NOVEC,NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
$LOCAL	52	NOVEC, WRT, RD,NOEXE,NOSHR, LCL, REL, CON, PIC,ALIGN(2)

COMMAND QUALIFIERS

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

COMPILER INTERNAL TIMING

Phase	Faults	CPU Time	Elapsed Time
Initialization	422	00:00.5	00:01.3
Source Analysis	315	00:01.7	00:06.3
Source Listing	14	00:00.8	00:02.7
Tree Construction	185	00:00.5	00:00.7
Flow Analysis	77	00:00.3	00:00.7
Value Propagation	12	00:00.1	00:00.1
Profit Analysis	35	00:00.2	00:00.6
Context Analysis	173	00:02.3	00:05.3
Name Packing	7	00:00.0	00:00.0
Code Selection	98	00:00.5	00:01.6
Final	63	00:01.0	00:03.4
TOTAL	1405	00:07.9	00:22.7

COMPILATION STATISTICS

```
  CPU Time:       00:07.9         (3840 Lines/Minute)
```