# Content from: https://web.archive.org/web/20051027030857/http://maven.smith.edu/~thiebaut/ArtOfAssembly/artofasm.html

## The Art of Assembly Language Programming

### Forward Why Would Anyone Learn This Stuff?

## Section One: Machine Organization

## Section Two: Basic Assembly Language

## Section Three: Intermediate Level Assembly Language Programming

## Section Four: Advanced Assembly Language Programming

## Section Five: The PC's I/O Ports

## Section Six: Optimization

## Section Seven: Appendixes

## Appendix A: ASCII/IBM Character Set

## Appendix B: Annotated Bibliography

## Appendix C: Keyboard Scan Codes

## Appendix D: Instruction Set Reference

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/fwd/fwd.html

# The Art of Assembly Language

### Forward Why Would Anyone Learn This Stuff?

# Forward Why Would Anyone Learn This Stuff?

* Your major requires a course in assembly language; i.e., you're here  
  against your will.A programmer where you work quit. Most of the source code left behind  
  was written in assembly language and you were elected to maintain it.Your boss has the audacity to insist that you write your code in assembly  
  against your strongest wishes.Your programs run just a little too slow, or are a little too large  
  and you think assembly language might help you get your project under control.You want to understand how computers actually work.You're interested in learning how to write efficient code.You want to try something new.

## 1 What's Wrong With Assembly Language

* Assembly is hard to learn.Assembly is hard to read and understand.Assembly is hard to debug.Assembly is hard to maintain.Assembly is hard to write.Assembly language programming is time consuming.Improved compiler technology has eliminated the need for assembly language.Today, machines are so fast that we no longer need to use assembly.If you need more speed, you should use a better algorithm rather than  
  switch to assembly language.Machines have so much memory today, saving space using assembly is not  
  important.Assembly language is not portable.

## 2 What's Right With Assembly Language?

* Speed. Assembly language programs are generally the fastest programs  
  around.Space. Assembly language programs are often the smallest.Capability. You can do things in assembly which are difficult or impossible  
  in HLLs.Knowledge. Your knowledge of assembly language will help you write better  
  programs, even when using HLLs.

## 3 Organization of This Text and Pedagogical Concerns

## 4 Obtaining Program Source Listings and Other Materials in This Text

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH01/CH01-1.html

## Art of Assembly Language: Chapter One

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH01/CH01-2.html

## 1.3 The Hexadecimal Numbering System

* All numeric values (regardless of their radix) begin with a decimal  
  digit.All hexadecimal values end with the letter "h", e.g., 123A4h.All binary values end with the letter "b".Decimal numbers may have a "t" or "d" suffix.

## 1.4 Arithmetic Operations on Binary and Hexadecimal Numbers

* CasioHewlett-PackardSharpTexas Instruments

## 1.5 Logical Operations on Bits

## 1.6 Logical Operations on Binary Numbers and Bit Strings

## 1.7 Signed and Unsigned Numbers

## 1.8 Sign and Zero Extension

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH01/CH01-3.html

## 1.9 Shifts and Rotates

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH02/CH02-1.html

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH02/CH02-2.html

## 2.5Simplification of Boolean Functions

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH02/CH02-3.html

## 2.6What Does This Have To Do With Computers, Anyway?

### 2.6.1Correspondence Between Electronic Circuits and Boolean Functions

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH03/CH03-1.html

## Art of Assembly Language: Chapter Three

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH03/CH03-2.html

## 3.2 System Timing

### 3.2.1 The System Clock

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH03/CH03-3.html

## 3.3 The 886, 8286, 8486, and 8686 "Hypothetical" Processors

### 3.3.1 CPU Registers

### 3.3.2 The Arithmetic & Logical Unit

* Copies the value from AX into the ALU,Sends the value five to the ALU,Instructs the ALU to add these two values together,Moves the result back into the AX register.

### 3.3.3 The Bus Interface Unit

### 3.3.4 The Control Unit and Instruction Sets

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH03/CH03-4.html

### 3.3.10 The 886 Processor

### 3.3.11 The 8286 Processor

* Fetch the instruction byte from memory.Update theipregister to   
    
    
  point at the next byte.Decode the instruction to see what it does.If required, fetch a 16-bit instruction operand from memory.If required, updateipto point beyond the operand.Compute the address of the operand, if requir  
    
    
  ed (i.e., bx+xxxx) .Fetch the operand.Store the fetched value into the destination register
* Fetch the instruction byte from memory.Decode the instruction and update ipIf required, fetch a 16-bit instruction operand from memory.Compute the address of the operand, if r  
    
    
  equired (i.e., bx+xxxx) .Fetch the operand, if required updateipto point beyond xxxx.Store the fetched value into the destination register.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH03/CH03-5.html

### 3.3.12 The 8486 Processor

### 3.3.12.1 The 8486 Pipeline

* Fetch opcode.Decode opcode and (in parallel) prefetch a possible 16-bit operand.Compute complex addressing mode (e.g.,[xxxx+bx]), if applicable.Fetch the source value from memory (if a memory operand) and the destination  
  register value (if applicable).Compute the result.Store result into destination register.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH03/CH03-6.html

## 3.4 I/O (Input/Output)

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH04/CH04-1.html

## Art of Assembly Language: Chapter Four

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH04/CH04-2.html

## 4.6 The 80x86 Addressing Modes

### 4.6.1 8086 Register Addressing Modes

### 4.6.2 8086 Memory Addressing Modes

### 4.6.2.1 The Displacement Only Addressing Mode

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH04/CH04-3.html

### 4.6.3 80386 Register Addressing Modes

### 4.6.4 80386 Memory Addressing Modes

### 4.6.4.1 Register Indirect Addressing Modes

### 4.6.4.2 80386 Indexed, Base/Indexed, and Base/Indexed/Disp Addressing Modes

### 4.6.4.3 80386 Scaled Indexed Addressing Modes

### 4.6.4.4 Some Final Notes About the 80386 Memory Addressing Modes

## 4.7 The 80x86 MOV Instruction

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH05/CH05-1.html

## Art of Assembly Language: Chapter Five

Did you find an error in *The Art of Assembly Language Programming*?  
You can let me know by using the form below to report the error to me so  
that I can correct the error for the next beta version. Thank you.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH05/CH05-2.html

## 5.6 Composite Data Types

### 5.6.1 Arrays

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH05/CH05-3.html

### 5.6.3 Structures

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH05/CH05-4.html

## 5.7 Sample Programs

### 5.7.1 Simple Variable Declarations

### 5.7.2 Using Pointer Variables

### 5.7.3 Single Dimension Array Access

### 5.7.4 Multidimensional Array Access

### 5.7.5 Simple Structure Access

### 5.7.6 Arrays of Structures

### 5.7.7 Structures and Arrays as Fields of Another Structure

### 5.7.8 Pointers to Structures and Arrays of Structures

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH06/CH06-1.html

## Art of Assembly Language: Chapter Six

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH06/CH06-2.html

## 6.5 Arithmetic Instructions

### 6.5.1 The Addition Instructions: ADD, ADC, INC, XADD, AAA, and DAA

### 6.5.1.1 The ADD and ADC Instructions

* The overflow flag denotes a signed arithmetic overflow.The carry flag denotes an unsigned arithmetic overflow.The sign flag denotes a negative result (i.e., the H.O. bit of the result  
  is one).The zero flag is set if the result of the addition is zero.The auxiliary carry flag contains one if a BCD overflow out of the L.O.  
  nibble occurs.The parity flag is set or cleared depending on the parity of the L.O.  
  eight bits of the result. If there are an even number of one bits in the  
  result, the ADD instructions will set the parity flag to one (to denote  
  even parity). If there are an odd number of one bits in the result, the  
  ADD instructions clear the parity flag (to denote odd parity).

### 6.5.1.2 The INC Instruction

### 6.5.1.3 The XADD Instruction

### 6.5.1.4 The AAA and DAA Instructions

### 6.5.2 The Subtraction Instructions: SUB, SBB, DEC, AAS, and DAS

* They set the zero flag if the result is zero. This occurs only if the  
  operands are equal forsubandsbb. Thedecinstruction sets the zero flag only when it decrements the value one.These instructions set the sign flag if the result is negative.These instructions set the overflow flag if signed overflow/underflow  
  occurs.They set the auxiliary carry flag as necessary for BCD/ASCII arithmetic.They set the parity flag according to the number of one bits appearing  
  in the result value.Thesubandsbbinstructions set the carry  
  flag if an unsigned overflow occurs. Note that thedecinstruction  
  does not affect the carry flag.

### 6.5.3 The CMP Instruction

### 6.5.4 The CMPXCHG, and CMPXCHG8B Instructions

### 6.5.5 The NEG Instruction

### 6.5.6 The Multiplication Instructions: MUL, IMUL, and AAM

* There isn't an 8x8 bit multiplication available (the immediate8 operands  
  simply provide a shorter form of the instruction. Internally, the CPU sign  
  extends the operand to 16 or 32 bits as necessary).These instructions do not produce a 2\*n bit result. That is, a 16x16  
  multiply produces a 16 bit result. Likewise, a 32x32 bit multiply produces  
  a 32 bit result. These instructions set the carry and overflow flags if  
  the result does not fit into the destination register.The 80286 version ofimulallows an immediate operand,  
  the standardmul/imulinstructions do not.

### 6.5.7 The Division Instructions: DIV, IDIV, and AAD

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH06/CH06-3.html

## 6.6 Logical, Shift, Rotate and Bit Instructions

### 6.6.1 The Logical Instructions: AND, OR, XOR, and NOT

* They clear the carry flag.They clear the overflow flag.They set the zero flag if the result is zero, they clear it otherwise.They copy the H.O. bit of the result into the sign flag.They set the parity flag according to the parity (number of one bits)  
  in the result.They scramble the auxiliary carry flag.

### 6.6.2 The Shift Instructions: SHL/SAL, SHR, SAR, SHLD, and SHRD

### 6.6.2.1 SHL/SAL

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH06/CH06-4.html

### 6.6.4 The Bit Operations

### 6.6.4.1 TEST

* It clears the carry flag.It clears the overflow flag.It sets the zero flag if the result is zero, they clear it otherwise.It copies the H.O. bit of the result into the sign flag.It sets the parity flag according to the parity (number of one bits)  
  in the L.O. byte of the result.It scrambles the auxiliary carry flag.

### 6.6.4.2 The Bit Test Instructions: BT, BTS, BTR, and BTC

### 6.6.4.3 Bit Scanning: BSF and BSR

### 6.6.5 The "Set on Condition" Instructions

## 6.7 I/O Instructions

## 6.8 String Instructions

* movs(move string)lods(load string element into the accumulator)stos(store accumulator into string element)scas(Scan string and check for match against the value  
  in the accumulator)cmps(compare two strings).ins(input a string from an I/O port)outs(output a string to an I/O portrep(repeat a string operation)repz(repeat while zero)repe(repeat while equal)repnz(repeat while not zero)repne(repeat while not equal)

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH06/CH06-5.html

## 6.9 Program Flow Control Instructions

### 6.9.1 Unconditional Jumps

### 6.9.2 The CALL and RET Instructions

* It pushes thecsregister onto the stack.It pushes the 16 bit offset of the next instruction following the call  
  onto the stack.It copies the 32 bit effective address into thecs:ipregisters.  
  Since thecallinstruction allows the same addressing modes  
  asjmp,callcan obtain the target address using  
  a relative, memory, or register addressing mode.Execution continues at the first instruction of the subroutine. This  
  first instruction is the opcode at the target address computed in the previous  
  step.
* It pushes the 16 bit offset of the next instruction following the call  
  onto the stack.It copies the 16 bit effective address into theipregister.  
  Since thecallinstruction allows the same addressing modes  
  asjmp,callcan obtain the target address using  
  a relative, memory, or register addressing mode.Execution continues at the first instruction of the subroutine. This  
  first instruction is the opcode at the target address computed in the previous  
  step.

### 6.9.3 The INT, INTO, BOUND, and IRET Instructions

* The 80x86 pushes the flags register onto the stack;The 80x86 pushescsand theniponto the stack;The 80x86 uses the interrupt number (intois interrupt  
  #4,boundis interrupt #5) times four as an index into the  
  interrupt vector table and copies the double word at that point in the table  
  intocs:ip.

### 6.9.4 The Conditional Jump Instructions

* Whatever jump you're using, switch to its opposite form. (given in the  
  tables above).Once you've selected the opposite branch, use it to jump over ajmpinstruction whose target address is the original target address.
* If the second letter of thejccinstruction is not an "n",  
  insert an "n" after the "j". E.g.,jebecomesjneandjlbecomesjnl.If the second letter of thejccinstruction is an "n",  
  then remove that "n" from the instruction. E.g.,jngbecomesjg,jnebecomesje.

### 6.9.5 The JCXZ/JECXZ Instructions

### 6.9.6 The LOOP Instruction

### 6.9.7 The LOOPE/LOOPZ Instruction

### 6.9.8 The LOOPNE/LOOPNZ Instruction

## 6.10 Miscellaneous Instructions

* clcClears the carry flagstcSets the carry flagcmcComplements the carry flagcldClears the direction flagstdSets the direction flagcliClears the interrupt enable/disable flagstiSets the interrupt enable/disable flag

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH06/CH06-6.html

## 6.11 Sample Programs

### 6.11.1 Simple Arithmetic I

### 6.11.2 Simple Arithmetic II

### 6.11.3 Logical Operations

### 6.11.4 Shift and Rotate Operations

### 6.11.5 Bit Operations and SETcc Instructions

### 6.11.6 String Operations

### 6.11.7 Conditional Jumps

### 6.11.8 CALL and INT Instructions

### 6.11.9 Conditional Jumps I

### 6.11.10 Conditional Jump Instructions II

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH07/CH07-1.html

## Art of Assembly Language: Chapter Seven

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH07/CH07-2.html

## 7.2 Sample Programs

### 7.2.1 Stripped SHELL.ASM File

### 7.2.2 Numeric I/O

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH08/CH08-1.html

## Art of Assembly Language: Chapter Eight

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH08/CH08-2.html

## 8.8 Segments

### 8.8.1 Segment Names

### 8.8.2 Segment Loading Order

### 8.8.3 Segment Operands

### 8.8.3.1 The ALIGN Type

If segments one and two are declared as below, and segment #2 is word aligned,  
the segments appear in memory as shown below:

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH08/CH08-3.html

### 8.8.4 The CLASS Type

* (1) The assembler combines all public segments that have the same name.(2) Once combined, the segments are output to the object code file in  
  the order of their appearance in the source file. If a segment name appears  
  twice within a source file (and it's public), then the combined segment  
  will be output to the object code file at the position denoted by the first  
  occurrence of the segment within the source file.(3) The linker reads the object code file produced by the assembler  
  and rearranges the segments when creating the executable file. The linker  
  begins by writing the first segment found in the object code file to the  
  .EXE file. Then it searches throughout the object code file for every segment  
  with the same class name. Such segments are sequentially written to the  
  .EXE file.(4) Once all the segments with the same class name as the first segment  
  are emitted to the .EXE file, the linker scans the object code file for  
  the next segment which doesn't belong to the same class as the previous  
  segment(s). It writes this segment to the .EXE file and repeats step (3)  
  for each segment belonging to this class.(5) Finally, the linker repeats step (4) until it has linked all the  
  segments in the object code file.

### 8.8.5 The Read-only Operand

### 8.8.6 The USE16, USE32, and FLAT Options

### 8.8.7 Typical Segment Definitions

### 8.8.8 Why You Would Want to Control the Loading Order

### 8.8.9 Segment Prefixes

### 8.8.10 Controlling Segments with the ASSUME Directive

### 8.8.11 Combining Segments: The GROUP Directive

* If MASM doesn't know that a segment register is pointing at the symbol's  
  segment or a group containing that segment, MASM generates an error.If anassumedirective associates the segment name with  
  a segment register but does not associate a segment register with the group  
  name, then MASM uses the offset of the symbol within its segment.If anassumedirective associates the group name with a  
  segment register but does not associate a segment register with the symbol's  
  segment name, MASM uses the offset of the symbol with the group.If anassumedirective provides segment register association  
  with both the symbol's segment and its group, MASM will pick the offset  
  that would not require a segment override prefix. For example, if the assume  
  directive specifies thatdspoints at the group name andespoints at the segment name, MASM will use the group offset if the default  
  segment register would bedssince this would not require MASM  
  to emit a segment override prefix opcode. If either choice results in the  
  emission of a segment override prefix, MASM will choose the offset (and  
  segment override prefix) associated with the symbol's segment.

### 8.8.12 Why Even Bother With Segments?

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH08/CH08-4.html

## 8.9 The END Directive

## 8.10 Variables

## 8.11 Label Types

### 8.11.1 How to Give a Symbol a Particular Type

### 8.11.2 Label Values

### 8.11.3 Type Conflicts

## 8.12 Address Expressions

### 8.12.1 Symbol Types and Addressing Modes

* NEAR if the variable was declared withword/sword/dwFAR if the variable was declared withdword/sdword/dd

### 8.12.2 Arithmetic and Logical Operators

### 8.12.3 Coercion

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH08/CH08-5.html

### 8.12.4 Type Operators

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH08/CH08-6.html

## 8.13 Conditional Assembly

### 8.13.1 IF Directive

### 8.13.2 IFE directive

### 8.13.3 IFDEF and IFNDEF

### 8.13.4 IFB, IFNB

### 8.13.5 IFIDN, IFDIF, IFIDNI, and IFDIFI

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH08/CH08-7.html

## 8.14 Macros

### 8.14.1 Procedural Macros

### 8.14.2 Macros vs. 80x86 Procedures

* encounters thecallinstruction,pushes the return address onto the stack,jumps toProc\_1,executes the code therein,pops the return address off the stack, andreturns to the calling code.

### 8.14.3 The LOCAL Directive

### 8.14.4 The EXITM Directive

### 8.14.5 Macro Parameter Expansion and Macro Operators

### 8.14.6 A Sample Macro to Implement For Loops

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH08/CH08-8.html

### 8.14.7 Macro Functions

### 8.14.8 Predefined Macros, Macro Functions, and Symbols

### 8.14.9 Macros vs. Text Equates

### 8.14.10 Macros: Good and Bad News

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH08/CH08-9.html

## 8.15 Repeat Operations

## 8.16 The FOR and FORC Macro Operations

## 8.17 The WHILE Macro Operation

## 8.18 Macro Parameters

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH08/CH08-10.html

## 8.19 Controlling the Listing

### 8.19.1 The ECHO and %OUT Directives

### 8.19.2 The TITLE Directive

### 8.19.3 The SUBTTL Directive

### 8.19.4 The PAGE Directive

### 8.19.5 The .LIST, .NOLIST, and .XLIST Directives

### 8.19.6 Other Listing Directives

## 8.20 Managing Large Programs

### 8.20.1 The INCLUDE Directive

### 8.20.2 The PUBLIC, EXTERN, and EXTRN Directives

### 8.20.3 The EXTERNDEF Directive

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH08/CH08-11.html

## 8.22 Sample Program

### 8.22.1 EX8.MAK

### 8.22.2 Matrix.A

### 8.22.3 EX8.ASM

### 8.22.4 GETI.ASM

### 8.22.5 GetArray.ASM

### 8.22.6 XProduct.ASM

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH09/CH09-1.html

## Art of Assembly Language: Chapter Nine

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH09/CH09-2.html

## 9.2 Logical (Boolean) Expressions

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH09/CH09-3.html

## 9.3 Multiprecision Operations

### 9.3.1 Multiprecision Addition Operations

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH09/CH09-4.html

### 9.3.4 Extended Precision Multiplication

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH09/CH09-5.html

### 9.3.6 Extended Precision NEG Operations

### 9.3.7 Extended Precision AND Operations

### 9.3.8 Extended Precision OR Operations

### 9.3.9 Extended Precision XOR Operations

### 9.3.10 Extended Precision NOT Operations

### 9.3.11 Extended Precision Shift Operations

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH09/CH09-6.html

## 9.5 Machine and Arithmetic Idioms

### 9.5.1 Multiplying Without MUL and IMUL

### 9.5.2 Division Without DIV and IDIV

### 9.5.3 Using AND to Compute Remainders

### 9.5.4 Implementing Modulo-n Counters with AND

### 9.5.5 Testing an Extended Precision Value for 0FFFF..FFh

### 9.5.6 TEST Operations

### 9.5.7 Testing Signs with the XOR Instruction

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH09/CH09-7.html

## 9.6 Masking Operations

### 9.6.1 Masking Operations with the AND Instruction

### 9.6.2 Masking Operations with the OR Instruction

## 9.7 Packing and Unpacking Data Types

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH09/CH09-8.html

## 9.9 Sample Programs

### 9.9.1 Converting Arithmetic Expressions to Assembly Language

### 9.9.2 Boolean Operations Example

### 9.9.3 64-bit Integer I/O

### 9.9.4 Packing and Unpacking Date Data Types

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH10/CH10-1.html

## Art of Assembly Language: Chapter Ten

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH10/CH10-2.html

## 10.3 CASE Statements

## 10.4 State Machines and Indirect Jumps

## 10.5 Spaghetti Code

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH10/CH10-3.html

## 10.6 Loops

### 10.6.1 While Loops

### 10.6.2 Repeat..Until Loops

### 10.6.3 LOOP..ENDLOOP Loops

### 10.6.4 FOR Loops

## 10.7 Register Usage and Loops

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH10/CH10-4.html

## 10.8 Performance Improvements

### 10.8.1 Moving the Termination Condition to the End of a Loop

### 10.8.2 Executing the Loop Backwards

### 10.8.3 Loop Invariant Computations

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH10/CH10-5.html

## 10.9 Nested Statements

## 10.10 Timing Delay Loops

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH10/CH10-6.html

## 10.11 Sample Program

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH11/CH11-1.html

## Art of Assembly Language: Chapter Eleven

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH11/CH11-2.html

## 11.5 Parameters

* where is the data coming from?how do you pass and return data?what is the amount of data to pass?
* pass by value,pass by reference,pass by value/returned,pass by result, andpass by name.pass by lazy evaluation
* in registers,in global memory locations,on the stack,in the code stream, orin a parameter block referenced via a pointer.

### 11.5.1 Pass by Value

### 11.5.2 Pass by Reference

### 11.5.3 Pass by Value-Returned

### 11.5.4 Pass by Result

### 11.5.5 Pass by Name

### 11.5.6 Pass by Lazy-Evaluation

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH11/CH11-3.html

### 11.5.7 Passing Parameters in Registers

### 11.5.8 Passing Parameters in Global Variables

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH11/CH11-4.html

### 11.5.9 Passing Parameters on the Stack

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH11/CH11-5.html

### 11.5.10 Passing Parameters in the Code Stream

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH11/CH11-6.html

### 11.5.11 Passing Parameters via a Parameter Block

## 11.6 Function Results

### 11.6.1 Returning Function Results in a Register

### 11.6.2 Returning Function Results on the Stack

### 11.6.3 Returning Function Results in Memory Locations

## 11.7 Side Effects

* Always properly document the input and output conditions of a procedure.  
  Never rely on any other entry or exit conditions other than these documented  
  operations.Partition your procedures so that they compute a single value or execute  
  a single operation. Subroutines that do two or more tasks are, by definition,  
  producing side effects unless every invocation of that subroutine requires  
  all the computations and operations.When updating the code in a procedure, make sure that it still obeys  
  the entry and exit conditions. If not, either modify the program so that  
  it does or update the documentation for that procedure to reflect the new  
  entry and exit conditions.Avoid passing information between routines in the CPU's flag register.  
  Passing an error status in the carry flag is about as far as you should  
  ever go. Too many instructions affect the flags and it's too easy to foul  
  up a return sequence so that an important flag is modified on return.Always save and restore all registers a procedure modifies.Avoid passing parameters and function results in global variables.Avoid passing parameters by reference (with the intent of modifying  
  them for use by the calling code).

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH11/CH11-7.html

## 11.8 Local Variable Storage

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH11/CH11-8.html

## 11.10 Sample Program

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH12/CH12-1.html

## Art of Assembly Language: Chapter Twelve

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH12/CH12-2.html

### 12.1.3 Static Links

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH12/CH12-3.html

### 12.1.5 The Display

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH12/CH12-4.html

## 12.2 Passing Variables at Different Lex Levels as Parameters.

### 12.2.1 Passing Parameters by Value in a Block Structured Language

### 12.2.2 Passing Parameters by Reference, Result, and Value-Result in a Block Structured Language

### 12.2.3 Passing Parameters by Name and Lazy-Evaluation in a Block Structured Language

## 12.3 Passing Parameters as Parameters to Another Procedure

### 12.3.1 Passing Reference Parameters to Other Procedures

### 12.3.2 Passing Value-Result and Result Parameters as Parameters

### 12.3.3 Passing Name Parameters to Other Procedures

### 12.3.4 Passing Lazy Evaluation Parameters as Parameters

### 12.3.5 Parameter Passing Summary

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH12/CH12-5.html

## 12.4 Passing Procedures as Parameters

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH12/CH12-6.html

## 12.5 Iterators

### 12.5.1 Implementing Iterators Using In-Line Expansion

### 12.5.2 Implementing Iterators with Resume Frames

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH12/CH12-7.html

## 12.6 Sample Programs

### 12.6.1 An Example of an Iterator

### 12.6.2 Another Iterator Example

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH13/CH13-1.html

## Art of Assembly Language: Chapter Thirteen

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH13/CH13-2.html

## 13.1 The IBM PC BIOS

## 13.2 An Introduction to the BIOS' Services

### 13.2.1 INT 5- Print Screen

### 13.2.2 INT 10h - Video Services

### 13.2.3 INT 11h - Equipment Installed

### 13.2.4 INT 12h - Memory Available

### 13.2.5 INT 13h - Low Level Disk Services

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH13/CH13-3.html

### 13.2.6 INT 14h - Serial I/O

### 13.2.6.1 AH=0: Serial Port Initialization

### 13.2.6.2 AH=1: Transmit a Character to the Serial Port

### 13.2.6.3 AH=2: Receive a Character from the Serial Port

### 13.2.6.4 AH=3: Serial Port Status

### 13.2.7 INT 15h - Miscellaneous Services

### 13.2.8 INT 16h - Keyboard Services

### 13.2.8.1 AH=0: Read a Key From the Keyboard

### 13.2.8.2 AH=1: See if a Key is Available at the Keyboard

### 13.2.8.3 AH=2: Return Keyboard Shift Key Status

### 13.2.9 INT 17h - Printer Services

### 13.2.9.1 AH=0: Print a Character

### 13.2.9.2 AH=1: Initialize Printer

### 13.2.9.3 AH=2: Return Printer Status

### 13.2.10 INT 18h - Run BASIC

### 13.2.11 INT 19h - Reboot Computer

### 13.2.12 INT 1Ah - Real Time Clock

### 13.2.12.1 AH=0: Read the Real Time Clock

### 13.2.12.2 AH=1: Setting the Real Time Clock

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH13/CH13-4.html

## 13.3 An Introduction to MS-DOS'

### 13.3.1 MS-DOS Calling Sequence

### 13.3.2 MS-DOS Character Oriented Functions

### 13.3.3 MS-DOS Drive Commands

### 13.3.4 MS-DOS "Obsolete" Filing Calls

### 13.3.5 MS-DOS Date and Time Functions

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH13/CH13-5.html

### 13.3.6 MS-DOS Memory Management Functions

### 13.3.6.1 Allocate Memory

### 13.3.6.2 Deallocate Memory

### 13.3.6.3 Modify Memory Allocation

### 13.3.6.4 Advanced Memory Management Functions

### 13.3.7 MS-DOS Process Control Functions

### 13.3.7.1 Terminate Program Execution

### 13.3.7.2 Terminate, but Stay Resident

### 13.3.7.3 Execute a Program

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH13/CH13-6.html

### 13.3.8 MS-DOS "New" Filing Calls

### 13.3.8.1 Open File

### 13.3.8.2 Create File

### 13.3.8.3 Close File

### 13.3.8.4 Read From a File

### 13.3.8.5 Write to a File

### 13.3.8.6 Seek (Move File Pointer)

### 13.3.8.7 Set Disk Transfer Address (DTA)

### 13.3.8.8 Find First File

### 13.3.8.9 Find Next File

### 13.3.8.10 Delete File

### 13.3.8.11 Rename File

### 13.3.8.12 Change/Get File Attributes

### 13.3.8.13 Get/Set File Date and Time

### 13.3.8.14 Other DOS Calls

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH13/CH13-7.html

### 13.3.9 File I/O Examples

### 13.3.9.1 Example #1: A Hex Dump Utility

### 13.3.9.2 Example #2: Upper Case Conversion

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH13/CH13-8.html

### 13.3.10 Blocked File I/O

### 13.3.11 The Program Segment Prefix (PSP)

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH13/CH13-9.html

### 13.3.12 Accessing Command Line Parameters

### 13.3.13 ARGC and ARGV

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH13/CH13-10.html

## 13.4 UCR Standard Library File I/O Routines

### 13.4.1 Fopen

### 13.4.2 Fcreate

### 13.4.3 Fclose

### 13.4.4 Fflush

### 13.4.5 Fgetc

### 13.4.6 Fread

### 13.4.7 Fputc

### 13.4.8 Fwrite

### 13.4.9 Redirecting I/O Through the StdLib File I/O Routines

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH13/CH13-11.html

### 13.4.10 A File I/O Example

## 13.5 Sample Program

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH14/CH14-1.html

## Art of Assembly Language: Chapter Fifteen

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH14/CH14-2.html

## 14.3 The UCR Standard Library Floating Point Routines

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH14/CH14-3.html

## 14.4 The 80x87 Floating Point Coprocessors

### 14.4.1 FPU Registers

### 14.4.1.1 The FPU Data Registers

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH14/CH14-4.html

### 14.4.3 The FPU Instruction Set

### 14.4.4 FPU Data Movement Instructions

### 14.4.4.1 The FLD Instruction

### 14.4.4.2 The FST and FSTP Instructions

### 14.4.4.3 The FXCH Instruction

### 14.4.5 Conversions

### 14.4.5.1 The FILD Instruction

### 14.4.5.2 The FIST and FISTP Instructions

### 14.4.5.3 The FBLD and FBSTP Instructions

### 14.4.6 Arithmetic Instructions

### 14.4.6.1 The FADD and FADDP Instructions

### 14.4.6.2 The FSUB, FSUBP, FSUBR, and FSUBRP Instructions

### 14.4.6.3 The FMUL and FMULP Instructions

### 14.4.6.4 The FDIV, FDIVP, FDIVR, and FDIVRP Instructions

### 14.4.6.5 The FSQRT Instruction

### 14.4.6.6 The FSCALE Instruction

### 14.4.6.7 The FPREM and FPREM1 Instructions

### 14.4.6.8 The FRNDINT Instruction

### 14.4.6.9 The FXTRACT Instruction

### 14.4.6.10 The FABS Instruction

### 14.4.6.11 The FCHS Instruction

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH14/CH14-5.html

### 14.4.7 Comparison Instructions

### 14.4.7.1 The FCOM, FCOMP, and FCOMPP Instructions

### 14.4.7.2 The FUCOM, FUCOMP, and FUCOMPP Instructions

### 14.4.7.3 The FTST Instruction

### 14.4.7.4 The FXAM Instruction

### 14.4.8 Constant Instructions

### 14.4.9 Transcendental Instructions

### 14.4.9.1 The F2XM1 Instruction

### 14.4.9.2 The FSIN, FCOS, and FSINCOS Instructions

### 14.4.9.3 The FPTAN Instruction

### 14.4.9.4 The FPATAN Instruction

### 14.4.9.5 The FYL2X and FYL2XP1 Instructions

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH14/CH14-6.html

## 14.5 Sample Program: Additional Trigonometric Functions

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH15/CH15-1.html

## Art of Assembly Language: Chapter Fifteen

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH15/CH15-2.html

### 15.1.4 The MOVS Instruction

### 15.1.5 The CMPS Instruction

* The direction flag must be cleared before comparing the strings.Use thecmpsbinstruction to compare the strings on a byte  
  by byte basis. Even if the strings contain an even number of characters,  
  you cannot use thecmpswinstruction. It does not compare strings  
  in lexicographical order.Thecxregister must be loaded with the length of the smaller  
  string.Use therepeprefix.Theds:siandes:diregisters must point at  
  the very first character in the two strings you want to compare.

### 15.1.6 The SCAS Instruction

### 15.1.7 The STOS Instruction

### 15.1.8 The LODS Instruction

### 15.1.9 Building Complex String Functions from LODS and STOS

### 15.1.10 Prefixes and the String Instructions

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH15/CH15-3.html

## 15.2 Character Strings

### 15.2.1 Types of Strings

### 15.2.2 String Assignment

### 15.2.3 String Comparison

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH15/CH15-4.html

## 15.3 Character String Functions

### 15.3.1 Substr

* DestStris the name of the string variable where you want  
  to store the substring,SrcStris the name of the source string (from which the  
  substring is to be taken),Indexis the starting character position within the string  
  (1..length(SrcStr)), andLength is the length of the substring you want to copy intoDestStr.
* The index parameter (Index) is less than one.Indexis greater than the length of the string.TheSubstrlength parameter (Length) is greater  
  than the length of the string.The sum ofIndexandLengthis greater than  
  the length of the string.
* The index parameter (Index) is less than one. There are  
  two ways to handle this error condition. One way is to automatically set  
  theIndexparameter to one and return the substring beginning  
  with the first character of the source string. The other alternative is  
  to return the empty string, a string of length zero, as the substring. Variations  
  on this theme are also possible. You might return the substring beginning  
  with the first character if the index is zero and an empty string if the  
  index is negative. Another alternative is to use unsigned numbers. Then  
  you've only got to worry about the case whereIndexis zero.  
  A negative number, should the calling code accidentally generate one, would  
  look like a large positive number.The index is greater than the length of the string. If this is the case,  
  then theSubstrfunction should return an empty string. Intuitively,  
  this is the proper response in this situation.TheSubstrlength parameter (Length) is greater  
  than the length of the string. -or-The sum ofIndexandLengthis greater than  
  the length of the string. Points three and four are the same problem, the  
  length of the desired substring extends beyond the end of the source string.  
  In this event,Substrshould return the substring consisting  
  of those characters starting atIndexthrough the end of the  
  source string.
* The substring, at locationes:di.Substrclears the carry flag if there were no errors.Substrsets the carry flag if there was an error.Substrpreserves all the registers.
* If theIndexparameter (ch) is zero,Substruses one instead.TheIndexandLengthparameters are both unsigned  
  byte values, therefore they are never negative.If theIndexparameter is greater than the length of the  
  source string,Substrreturns an empty string.If the sum of theIndexandLengthparameters  
  is greater than the length of the source string,Substrreturns  
  only those characters fromIndexthrough the end of the source  
  string. The following code realizes the substring function.

### 15.3.2 Index

### 15.3.3 Repeat

### 15.3.4 Insert

### 15.3.5 Delete

### 15.3.6 Concatenation

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH15/CH15-5.html

## 15.4 String Functions in the UCR Standard Library

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH15/CH15-6.html

## 15.5 The Character Set Routines in the UCR Standard Library

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH15/CH15-7.html

## 15.7 Sample Programs

### 15.7.1 Find.asm

### 15.7.2 StrDemo.asm

### 15.7.3 Fcmp.asm

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH16/CH16-1.html

## Art of Assembly Language: Chapter Sixteen

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH16/CH16-2.html

### 16.1.2.2 Nondeterministic Finite State Automata (NFAs)

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH16/CH16-3.html

### 16.1.2.5 Deterministic Finite State Automata (DFAs)

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH16/CH16-4.html

### 16.1.3 Context Free Languages

### 16.1.4 Eliminating Left Recursion and Left Factoring CFGs

### 16.1.5 Converting REs to CFGs

### 16.1.6 Converting CFGs to Assembly Language

### 16.1.7 Some Final Comments on CFGs

### 16.1.8 Beyond Context Free Languages

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH16/CH16-5.html

## 16.2 The UCR Standard Library Pattern Matching Routines

## 16.3 The Standard Library Pattern Matching Functions

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH16/CH16-6.html

## 16.4 Designing Your Own Pattern Matching Routines

## 16.5 Extracting Substrings from Matched Patterns

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH16/CH16-7.html

## 16.6 Semantic Rules and Actions

## 16.7 Constructing Patterns for the MATCH Routine

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH16/CH16-8.html

## 16.8 Some Sample Pattern Matching Applications

### 16.8.1 Converting Written Numbers to Integers

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH16/CH16-9.html

### 16.8.2 Processing Dates

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH16/CH16-10.html

### 16.8.3 Evaluating Arithmetic Expressions

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH16/CH16-11.html

### 16.8.4 A Tiny Assembler

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH16/CH16-12.html

### 16.8.5 The "MADVENTURE" Game

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH17/CH17-1.html

## Art of Assembly Language: Chapter Seventeen

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH17/CH17-2.html

## 17.3 Exceptions

### 17.3.1 Divide Error Exception (INT 0)

### 17.3.2 Single Step (Trace) Exception (INT 1)

### 17.3.3 Breakpoint Exception (INT 3)

### 17.3.4 Overflow Exception (INT 4/INTO)

### 17.3.5 Bounds Exception (INT 5/BOUND)

### 17.3.6 Invalid Opcode Exception (INT 6)

### 17.3.7 Coprocessor Not Available (INT 7)

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH17/CH17-3.html

## 17.4 Hardware Interrupts

### 17.4.1 The 8259A Programmable Interrupt Controller (PIC)

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH17/CH17-4.html

## 17.5 Chaining Interrupt Service Routines

## 17.6 Reentrancy Problems

## 17.7 The Efficiency of an Interrupt Driven System

### 17.7.1 Interrupt Driven I/O vs. Polling

### 17.7.2 Interrupt Service Time

### 17.7.3 Interrupt Latency

### 17.7.4 Prioritized Interrupts

## 17.8 Debugging ISRs

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH18/CH18-1.html

## Art of Assembly Language: Chapter Eighteen

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH18/CH18-2.html

## 18.2 Active vs. Passive TSRs

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH18/CH18-3.html

## 18.3 Reentrancy

### 18.3.1 Reentrancy Problems with DOS

### 18.3.2 Reentrancy Problems with BIOS

### 18.3.3 Reentrancy Problems with Other Code

## 18.4 The Multiplex Interrupt (INT 2Fh)

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH18/CH18-4.html

## 18.5 Installing a TSR

## 18.6 Removing a TSR

## 18.7 Other DOS Related Issues

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH18/CH18-5.html

## 18.8 A Keyboard Monitor TSR

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH18/CH18-6.html

## 18.9 Semiresident Programs

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH19/CH19-1.html

## Art of Assembly Language: Chapter Nineteen

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH19/CH19-2.html

### 19.1.2 Exception Handling in DOS: The Break Handler

### 19.1.3 Exception Handling in DOS: The Critical Error Handler

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH19/CH19-3.html

### 19.1.4 Exception Handling in DOS: Traps

### 19.1.5 Redirection of I/O for Child Processes

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH19/CH19-4.html

## 19.2 Shared Memory

### 19.2.1 Static Shared Memory

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH19/CH19-5.html

### 19.2.2 Dynamic Shared Memory

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH19/CH19-6.html

## 19.3 Coroutines

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH19/CH19-7.html

### 19.3.1 AMAZE.ASM

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH19/CH19-8.html

### 19.3.2 32-bit Coroutines

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH19/CH19-9.html

## 19.4 Multitasking

### 19.4.1 Lightweight and HeavyWeight Processes

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH19/CH19-10.html

### 19.4.2 The UCR Standard Library Processes Package

### 19.4.3 Problems with Multitasking

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH19/CH19-11.html

### 19.4.4 A Sample Program with Threads

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH19/CH19-12.html

## 19.5 Synchronization

### 19.5.1 Atomic Operations, Test & Set, and Busy-Waiting

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH19/CH19-13.html

### 19.5.2 Semaphores

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH19/CH19-14.html

### 19.5.3 The UCR Standard Library Semaphore Support

### 19.5.4 Using Semaphores to Protect Critical Regions

### 19.5.5 Using Semaphores for Barrier Synchronization

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH19/CH19-15.html

## 19.6 Deadlock

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH20/CH20-1.html

## Art of Assembly Language: Chapter Twenty

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH20/CH20-2.html

## 20.2The Keyboard Hardware Interface

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH20/CH20-3.html

## 20.3The Keyboard DOS Interface

## 20.4The Keyboard BIOS Interface

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH20/CH20-4.html

## 20.5The Keyboard Interrupt Service Routine

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH20/CH20-5.html

## 20.6Patching into the INT 9 Interrupt Service Routine

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH20/CH20-6.html

## 20.7Simulating Keystrokes

### 20.7.1Stuffing Characters in the Type Ahead Buffer

### 20.7.2Using the 80x86 Trace Flag to Simulate IN AL, 60H Instructions

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH20/CH20-7.html

### 20.7.3Using the 8042 Microcontroller to Simulate Keystrokes

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH21/CH21-1.html

TOC   
Copyright 1996 by Randall Hyde   
  
All rights reserved.   
  
Duplication other than for immediate display through a browser is prohibited  
by U.S. Copyright Law.   
  
This material is provided on-line as a beta-test of this text. It is for  
the personal use of the reader only. If you are interested in using this  
material as part of a course, please contact   
  
rhyde@cs.ucr.edu   
  
Supporting software and other materials are available via anonymous ftp  
from ftp.cs.ucr.edu. See the "/pub/pc/ibmpcdir" directory for  
details. You may also download the material from "Randall Hyde's Assembly  
Language Page" at URL:  
  
http://webster.ucr.edu  
  
Notes:   
  
This document does not contain the laboratory exercises, programming assignment\  
s,  
exercises, or chapter summary. These portions were omitted for several reasons:  
either they wouldn't format properly, they contained hyperlinks that were  
too much work to resolve, they were under constant revision, or they were  
not included for security reasons. Such omission should have very little  
impact on the reader interested in learning this material or evaluating  
this document.   
  
This document was prepared using Harlequin's Web Maker 2.2 and Quadralay's  
Webworks Publisher. Since HTML does not support the rich formatting options  
available in Framemaker, this document is only an approximation of the actual  
chapter from the textbook.   
  
If you are absolutely dying to get your hands on a version other than HTML,  
you might consider having the UCR Printing a Reprographics Department run  
you off a copy on their Xerox machines. For details, please read the following  
EMAIL message I received from the Printing and Reprographics Department:  
  
We are currently working on ways to publish this text in a form other than  
HTML (e.g., Postscript, PDF, Frameviewer, hard copy, etc.). This, however,  
is a low-priority project. Please do not contact Randall Hyde concerning  
this effort. When something happens, an announcement will appear on "Randa\  
ll  
Hyde's Assembly Language Page." Please visit this WEB site at http://webst\  
er.ucr.edu for the latest scoop.  
  
  
  
  
Did you find an error in *The Art of Assembly Language Programming*?  
You can let me know by using the form below to report the error to me so  
that I can correct the error for the next beta version. Thank you.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH21/CH21-2.html

Art of Assembly: Chapter Twenty-One

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH21/CH21-3.html

Art of Assembly: Chapter Twenty-One

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH22/CH22-1.html

## Art of Assembly Language: Chapter Twenty-Two

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH22/CH22-2.html

## 22.2 The UCR Standard Library Serial Communications Support Routines

## 22.3 Programming the 8250 (Examples from the Standard Library)

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH23/CH23-1.html

## Art of Assembly Language: Chapter Twenty-Three

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH24/CH24-1.html

## Art of Assembly Language: Chapter Twenty-Four

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH24/CH24-2.html

### 24.5.13 An SGDI Driver for the Standard Game Adapter Card

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH24/CH24-3.html

## 24.6 An SGDI Driver for the CH Products' Flight Stick Pro'

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH24/CH24-4.html

## 24.7 Patching Existing Games

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH25/CH25-1.html

## Art of Assembly Language: Chapter Twenty-Five

### Art of Assembly Bug Report Submissions

Please provide your name and e-mail address so I can contact you if  
I have any questions regarding your submission.

# Content from: https://web.archive.org/web/20051027030857/http:/maven.smith.edu/~thiebaut/ArtOfAssembly/CH25/CH25-2.html

## 25.5Improving the Implementation of an Algorithm