FSEN 102

8086 Assembly Language Programming

A Complete Tutorial Guide — From Beginner to Pro

FSEN 102 | by Resolutefemi

Renance Technology | Federal University of Technology, Akure

Table of Contents

1. Introduction to the 8086

The Intel 8086 is a 16-bit microprocessor released in 1978. It is one of the most important processors in computing history — its instruction set (x86) still powers modern PCs today. The 8086 has a 16-bit data bus, 20 address lines (can address 1 MB of memory), and operates at clock speeds of 4.77 to 10 MHz.

Why Learn 8086 Assembly?

- It teaches you how the CPU actually executes instructions
- It helps you understand memory, registers, and addressing at a hardware level
- It is a required topic in FSEN 102 (Software Experimental Lab II)
- Assembly is still used in embedded systems, device drivers, and performance-critical code

8086 Architecture Overview

The 8086 is divided into two independent processing units:

- **Bus Interface Unit (BIU)** — fetches instructions, reads/writes data, calculates addresses. Contains segment registers, IP, and 6-byte prefetch queue.
- **Execution Unit (EU)** — decodes and executes instructions. Contains general-purpose registers, ALU, and FLAGS register.

> *The BIU prefetches instructions while the EU executes them — this is called pipelining.*

2. Registers

The 8086 has 14 registers, each 16 bits wide, divided into four categories.

General-Purpose Registers

Register	High/Low	Primary Use
AX	AH/AL	Accumulator — math, I/O, data transfer
BX	BH/BL	Base — memory pointer [BX]
CX	CH/CL	Counter — loops (LOOP), string ops (REP)
DX	DH/DL	Data — high word in MUL/DIV, I/O port

Table 1 — General-purpose registers

Segment Registers

Reg	Name	Purpose
CS	Code Segment	Points to executing code (CS:IP)
DS	Data Segment	Points to data/variables (default for [BX])
SS	Stack Segment	Points to stack (SS:SP)
ES	Extra Segment	Extra data segment (string dest [DI])

Table 2 — Segment registers

The FLAGS Register

Flag	Abbr	When Set
Carry	CF	Carry/borrow in unsigned arithmetic
Zero	ZF	Result is zero
Sign	SF	Result is negative (MSB=1)
Overflow	OF	Signed overflow occurred
Direction	DF	String ops auto-decrement
Interrupt	IF	Maskable interrupts enabled
Trap	TF	Single-step mode (debugging)

Table 3 — Key FLAGS bits

3. Memory Addressing

Physical Address = Segment Register x 16 + Offset

Example: CS=1000h, IP=2000h -> Physical = 1000h x 10h + 2000h = 12000h.

Addressing Modes

Mode	Example	Description
Register	MOV AX, BX	Both operands are registers
Immediate	MOV AX, 1234h	Value embedded in instruction
Direct	MOV AX, [1234h]	Offset specified directly
Register Indirect	MOV AX, [BX]	Address held in register

Mode	Example	Description
Based	MOV AX, [BX+8]	Base register + displacement
Indexed	MOV AX, [SI+10]	Index register + displacement
Based Indexed	MOV AX, [BX+SI]	Base + index register
Based Idx+Disp	MOV AX, [BX+SI+8]	Base + index + displacement

Table 4 — Addressing modes

4. Your First Program — Hello World

hello.asm

```
.MODEL SMALL
.STACK 100h

.DATA
    msg DB 'Hello, World!$'

.CODE
main PROC
    MOV AX, @DATA
    MOV DS, AX
    MOV AH, 09h
    MOV DX, OFFSET msg
    INT 21h
    MOV AH, 4Ch
    MOV AL, 0
    INT 21h
main ENDP
END main
```

Line by Line

- **.MODEL SMALL** — one code segment + one data segment
- **.STACK 100h** — reserves 256 bytes for stack
- **msg DB 'Hello, World!$'** — declares string ending with $
- **MOV AH, 09h** — DOS function to print string
- **INT 21h** — calls DOS to execute the function
- **MOV AH, 4Ch** — DOS function to exit program

5. Instruction Set — Opcodes with Examples

Data Transfer

MOV — Move data

```
MOV AX, 1234h    ; Immediate to register
MOV BX, AX       ; Register to register
MOV CX, [BX]     ; Memory to register
MOV [SI], DX     ; Register to memory
```

MOV cannot move memory to memory directly. Use a register as intermediary.

PUSH / POP — Stack operations

```
PUSH AX          ; Decrement SP by 2, store AX at SS:SP
PUSH BX
POP BX           ; Load from SS:SP, increment SP by 2
POP AX           ; Last in, first out
```

XCHG — Exchange

```
MOV AX, 10
MOV BX, 20
XCHG AX, BX    ; Now AX=20, BX=10
```

LEA — Load Effective Address

```
LEA BX, msg      ; Load offset of msg into BX
LEA SI, [BX+10]  ; Load address BX+10 into SI
```

Arithmetic

ADD / SUB

```
MOV AX, 10
ADD AX, 5       ; AX = 15
MOV BX, 20
SUB BX, 7       ; BX = 13
```

INC / DEC (do not affect CF!)

```
MOV CX, 5
INC CX          ; CX = 6
DEC CX          ; CX = 5
```

MUL / IMUL — Multiplication

```
; 8-bit: AL x source -> AX
MOV AL, 10
MOV BL, 3
MUL BL          ; AX = 30

; 16-bit: AX x source -> DX:AX
MOV AX, 1000
MOV BX, 50
MUL BX          ; DX:AX = 50000
```

DIV / IDIV — Division

```
; 8-bit: AX / source -> AL (quotient), AH (remainder)
MOV AX, 100
MOV BL, 3
DIV BL          ; AL=33, AH=1

; 16-bit: DX:AX / source -> AX (quotient), DX (remainder)
MOV DX, 0
MOV AX, 1000
MOV BX, 10
DIV BX          ; AX=100, DX=0
```

Logic Instructions

AND / OR / XOR / NOT / TEST

```
AND AX, 00FFh    ; Mask: keep lower byte
OR BX, 00FFh     ; Set: force lower byte to 1
XOR CX, CX       ; Clear CX to 0 (common trick)
NOT DX           ; Flip all bits (one's complement)
TEST AL, 01h     ; Check if AL is odd (sets ZF)
```

SHL / SHR / SAR — Shifts

```
MOV AX, 5
SHL AX, 1        ; AX = 10 (multiply by 2)
SHL AX, 2        ; AX = 40 (multiply by 4 more)

MOV BX, 160
SHR BX, 1        ; BX = 80 (unsigned divide by 2)

MOV CX, -8
SAR CX, 1        ; CX = -4 (signed divide, preserves sign)
```

ROL / ROR — Rotates

```
MOV AX, 8001h
ROL AX, 1        ; AX = 0003h (MSB wraps to LSB)
ROR AX, 1        ; AX = 8001h (back again)
```

Comparison and Jumps

CMP — Compare

```
MOV AX, 10
CMP AX, 10      ; ZF=1 (equal)
CMP AX, 5       ; ZF=0, CF=0 (AX > 5)
CMP AX, 15      ; CF=1 (AX < 15)
```

Conditional Jumps

Instruction	Condition	Type
JE/JZ	ZF=1 (equal)	Both
JNE/JNZ	ZF=0 (not equal)	Both
JG	ZF=0 and SF=OF (greater)	Signed
JL	SF!=OF (less)	Signed
JA	CF=0 and ZF=0 (above)	Unsigned
JB/JC	CF=1 (below)	Unsigned

Table 5 — Common conditional jumps

LOOP — Loop with CX

```
MOV CX, 5        ; Loop 5 times
MOV AX, 0
loop_start:
    ADD AX, CX
    LOOP loop_start  ; CX--; if CX!=0, jump
; Result: AX = 5+4+3+2+1 = 15
```

Procedures

CALL / RET

```
print_msg PROC
    MOV AH, 09h
    MOV DX, OFFSET msg
    INT 21h
    RET
print_msg ENDP

; Call it:
    CALL print_msg
```

6. String Operations

Instruction	Operation
MOVSB/MOVSW	Copy byte/word from DS:SI to ES:DI
CMPSB/CMPSW	Compare DS:SI with ES:DI
SCASB/SCASW	Scan: compare AL/AX with ES:DI
LODSB/LODSW	Load from DS:SI into AL/AX
STOSB/STOSW	Store AL/AX to ES:DI

Table 6 — String instructions

REP — Repeat String Operations

```
; Copy 100 bytes
MOV CX, 100
LEA SI, source
LEA DI, dest
CLD             ; Forward direction
REP MOVSB       ; Copy 100 bytes

; Fill 50 bytes with zero
MOV CX, 50
MOV AL, 0
LEA DI, buffer
REP STOSB       ; Fill with zeros
```

7. Interrupts and DOS Services

AH	Function	Input	Output
01h	Read char (echo)	None	AL = char
02h	Display char	DL = char	None
09h	Display string	DS:DX = $-terminated	None
0Ah	Read string	DS:DX = buffer	Buffer filled
4Ch	Exit program	AL = return code	None

Table 7 — Common INT 21h functions

8. Practical Examples

Example 1: Adding Two Numbers

add.asm

```
.MODEL SMALL
.STACK 100h
.DATA
    num1 DB 15
    num2 DB 25
    result DB ?
.CODE
main PROC
    MOV AX, @DATA
    MOV DS, AX
    MOV AL, num1
    ADD AL, num2
    MOV result, AL
    MOV AH, 4Ch
    INT 21h
main ENDP
END main
```

Example 2: Find the Larger Number

larger.asm

```
.MODEL SMALL
.STACK 100h
.DATA
    a DW 30
    b DW 45
    larger DW ?
.CODE
main PROC
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, a
    CMP AX, b
    JGE a_is_larger
    MOV AX, b
a_is_larger:
    MOV larger, AX
    MOV AH, 4Ch
    INT 21h
main ENDP
END main
```

Example 3: Count 1 to 10

count10.asm

```
.MODEL SMALL
.STACK 100h
.CODE
main PROC
    MOV CX, 1
print_loop:
    MOV DL, CL
    ADD DL, 30h
    MOV AH, 02h
    INT 21h
    MOV DL, 0Ah
    INT 21h
    INC CX
    CMP CX, 10
    JLE print_loop
    MOV AH, 4Ch
    INT 21h
main ENDP
END main
```

Example 4: String Copy

strcpy.asm

```
.MODEL SMALL
.STACK 100h
.DATA
    source DB 'Hello!', 0
    dest DB 20 DUP(?)
.CODE
main PROC
    MOV AX, @DATA
    MOV DS, AX
    MOV ES, AX
    LEA SI, source
    LEA DI, dest
    CLD
copy_loop:
    LODSB
    STOSB
    CMP AL, 0
    JNE copy_loop
    MOV AH, 4Ch
    INT 21h
main ENDP
END main
```

Example 5: Factorial (Recursion)

factorial.asm

```
.MODEL SMALL
.STACK 200h
.DATA
    n DW 5
    result DW ?
.CODE
main PROC
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, n
    CALL factorial
    MOV result, AX
    MOV AH, 4Ch
    INT 21h
main ENDP
factorial PROC
    CMP AX, 1
    JLE base_case
    PUSH AX
    DEC AX
    CALL factorial
    POP BX
    MUL BX
    RET
base_case:
    MOV AX, 1
    RET
factorial ENDP
END main
```

9. Summary and Quick Reference

- **8086** = 16-bit, 1 MB memory, 20 address lines, 40 pins
- **BIU** = fetches, **EU** = executes (pipelining)
- **14 registers**: AX, BX, CX, DX, SI, DI, BP, SP, CS, DS, SS, ES, IP, FLAGS
- **Physical address** = Segment x 16 + Offset
- **MOV** = move, **ADD/SUB** = arithmetic, **AND/OR/XOR** = logic
- **JMP** = unconditional, **JE/JNE/JG/JL** = conditional
- **LOOP** = decrement CX, jump if CX!=0
- **CALL/RET** = subroutine call/return
- **INT 21h** = DOS API (AH=09h print, AH=4Ch exit)
- **String ops**: MOVSB, CMPSB, SCASB with REP prefix
- **Stack** grows down, PUSH decrements SP, POP increments SP

End of FSEN 102 8086 Assembly Programming Guide. Best of luck!