Unit 7 Advanced Microprocessors (9 Hrs.)

8086: logical block diagram and segments,

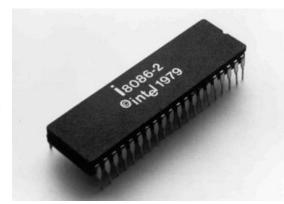
80286: Architecture, Registers, (Real/Protected mode), Privilege levels, descriptor cache, Memory access in GDT and LDT, multitasking, addressing modes, flag register

80386: Architecture, Register organization, Memory access in protected mode, Paging

Intel 8086

Introduction

- The Intel 8086 is a 16-bit microprocessor developed by Intel in 1978.
- It is the successor to the 8085 microprocessor and was a major step forward in microprocessor architecture.
- It introduced pipelining, segmented memory, and 16-bit data and address buses, enabling more powerful and efficient computing.
- It became the foundation for the x86 architecture, which is still widely used in modern computing systems.



♦ Back with of Intel 8085 and 8086

Feature	8085	8086
Data Bus	8-bit	16-bit
Address Bus	16-bit (64 KB)	20-bit (1 MB)
ALU Size	8-bit	16-bit
Pipelining	No	Yes
Registers	Fewer	More (and wider)
Operating Modes	Single mode	Min/Max mode
Memory Segmentation	No	Yes
Instruction Set	Basic	Advanced

Features of 8086

- 16-bit data bus & 20-bit address bus
- Pipelined architecture (fetch & execute simultaneously)
- 14 registers (general-purpose, segment, pointer, index, flag)
- Operating frequency: 5 MHz (8086), 8 MHz (8086-2), 10 MHz (8086-1)
- Multiplexed address/data bus (AD0–AD15)
- Supports hardware interrupts (NMI & INTR)
- Two modes of operation:
 - Minimum mode (single processor)
 - Maximum mode (multiprocessor)

Why Pipeline?

In computing, **pipelining** improves the efficiency of instruction execution by allowing multiple instructions to overlap in execution—like an assembly line in a factory.

Basic Concept:

When an application runs:

 $APP \rightarrow List \ of \ Programs \rightarrow Instructions \rightarrow Process \rightarrow Thread$

Each **instruction** (I) typically goes through four stages:

- F: Opcode Fetch
- **D**: Decode
- E: Execute
- S: Store

♦ For Single Processor (No Pipelining):

Each instruction is executed one after the other:

Time to execute =
$$I1 + I2 + I3 + I4$$

= $(F1 + D1 + E1 + S1) + (F2 + D2 + E2 + S2) +$
 $(F3 + D3 + E3 + S3) + (F4 + D4 + E4 + S4)$
= $4 \text{ ns} + 4 \text{ ns} + 4 \text{ ns}$
= $**16 \text{ ns}**$

♦ For Parallel Processor (With Pipelining):

Instructions are overlapped using pipeline stages:

Time to execute = $I1 + I2 + I3 + I4$							
P1	F1	D1	E1	S1			
P2		F2	D2	E2	S2		
P3			F3	D3	E3	S3	
P4				F4	D4	E4	S4

Time 1	Time to execute = $I1 + I2 + I3 + I4$						
P1	F1	D1	E1	S1			
P2		F2	D2	E2	S2		
Р3			F3	D3	E3	S3	
P4				F4	D4	E4	S4
	1	1	1	1	1	1	1

Total Time = 7 cycles \times 1 ns (per stage) = **7 ns**

Cycle 1: F1

Cycle 2: D1 F2

Cycle 3: E1 D2 F3

Cycle 4: S1 E2 D3 F4

Cycle 5: S2 E3 D4

Cycle 6: S3 E4

Cycle 7: S4

Total Time = 7 cycles \times 1 ns (per stage) = **7 ns**

Benefit of Pipelining:

- Non-pipelined execution: 4 instructions \rightarrow 16 ns
- **Pipelined execution:** 4 instructions \rightarrow 7 ns
- Time saved: 9 ns
- Efficiency increased by overlapping instruction stages.

Pin Diagram of the Intel 8086 Microprocessor

Intel 8086 Microprocessor, which consists of **40 pins**. Each pin serves a specific function depending on whether the processor is operating in **minimum mode** (single processor system) or **maximum mode** (multiprocessor system).

♦ Power & Clock Pins

- Vcc (Pin 40): +5V power supply.
- GND (Pins 1, 20): Ground connection.

- CLK (Pin 19): Clock input. Provides timing for internal operations.
- **RESET (Pin 21)**: Resets the processor and sets program counter to zero.

♦ Address/Data Bus (Multiplexed)

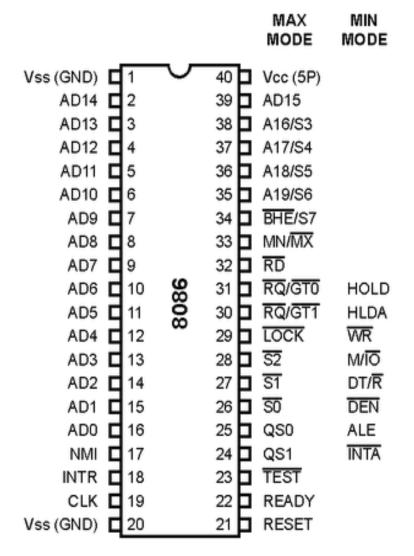
• AD0 – AD15 (Pins 2–16, 39): These are multiplexed lines used to carry address (A0–A15) during the first part of the bus cycle and data (D0–D15) during the second part.

♦ Address Bus (High)

• A16/S3 – A19/S6 (Pins 35–38): These are multiplexed with status signals (S3–S6) and used for addressing (A16–A19) in memory access.

Control & Status Pins

- ALE (Pin 25): Address Latch Enable. Indicates when AD0–AD15 contains an address.
- **DEN (Pin 26)**: Data Enable. Enables the external data bus transceivers.
- DT/R (Pin 27): Data Transmit/Receive. Controls direction of data flow.
- M/IO (Pin 28): Distinguishes memory or I/O operation.
- RD (Pin 32): Read control signal.
- WR (Pin 29): Write control signal.
- READY (Pin 22): Indicates if peripheral is ready for data transfer.



• TEST (Pin 23): Used for synchronization with external devices.

♦ Interrupt Pins

- INTR (Pin 18): Interrupt request (maskable).
- NMI (Pin 17): Non-maskable interrupt. Has higher priority than INTR.

♦ Minimum/Maximum Mode Selection

- $MN/M\overline{X}$ (Pin 33):
 - \circ Logic 1 \rightarrow Minimum mode
 - \circ Logic $0 \rightarrow$ Maximum mode

♦ Minimum Mode Specific Pins

(used only when $MN/M\overline{X} = 1$)

- INTA (Pin 24): Interrupt acknowledge.
- HOLD (Pin 31): DMA controller requests control of buses.
- HLDA (Pin 30): DMA controller granted access to buses.
- WR, RD, ALE, DEN, DT/R, M/IO \rightarrow Active in this mode.

♦ Maximum Mode Specific Pins

(used only when $MN/M\overline{X} = 0$)

- S2, S1, S0 (Pins 26–28): Status lines for controlling bus cycles.
- RQ/GT1, RQ/GT0 (Pins 34, 35): Request/Grant for bus control in multiprocessor mode.
- LOCK (Pin 29): Ensures exclusive access to shared resources.
- QS1, QS0 (Pins 24, 25): Queue status signals.

Intel 8086 architecture

The block diagram of the Intel 8086 architecture, showing the internal structure of the microprocessor. The 8086 is divided into two main units:

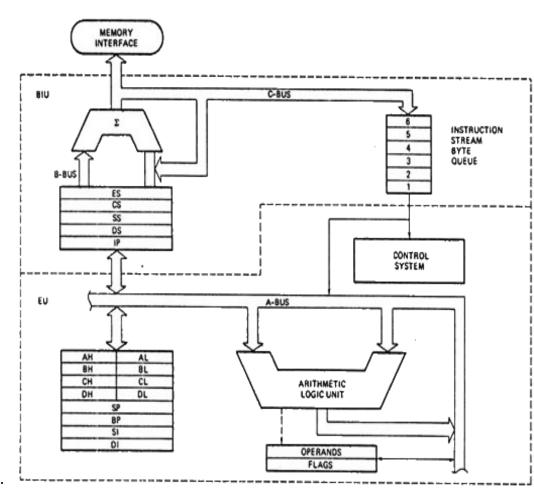
- 1. Bus Interface Unit (BIU)
- 2. Execution Unit (EU)

1. Bus Interface Unit (BIU)

Key Responsibilities:

- Handles address generation, instruction fetching, and communication with memory and I/O.
- Feeds the Execution
 Unit (EU) with a

 stream of instructions.



Components:

- Segment Registers (ES, CS, SS, DS)
 These hold segment addresses for Code (CS), Data (DS), Stack (SS), and Extra segment (ES).
- Instruction Pointer (IP)
 Points to the next instruction in the Code Segment (CS).
- Adder (Σ)
 Combines segment base and offset to form the physical address.

Instruction Queue

Pre-fetches up to **6 bytes** of instructions (using pipelining). This allows the Execution Unit to execute instructions faster.

• Memory Interface

Controls the **communication between memory and processor** using the address and data buses.

② 2. Execution Unit (EU)

Key Responsibilities:

- Decodes and executes instructions
- Performs arithmetic and logic operations
- · Controls flags and operands

Components:

• General Purpose Registers

Used for data storage and manipulation:

- AH/AL, BH/BL, CH/CL, DH/DL → Can be used as 8-bit or combined into AX, BX, CX, DX (16-bit)
- o SP (Stack Pointer), BP (Base Pointer), SI (Source Index), DI (Destination Index)
- Arithmetic and Logic Unit (ALU)

Performs all arithmetic (add, subtract, etc.) and logic (AND, OR, NOT, etc.) operations.

Operands and Flags

- Operands: Data inputs to the ALU.
- Flags: Special-purpose bits reflecting the outcome of ALU operations (e.g., Zero, Carry, Overflow).

Data Buses:

• A-Bus:

Transfers data between registers and ALU in the Execution Unit.

• C-Bus:

Transfers addresses and instructions from the BIU to memory and instruction queue.

₩ Summary:

Component	Purpose	
BIU	Handles memory interaction, instruction fetching	
Segment Registers	Holds memory segment addresses	
Instruction Pointer	Points to next instruction	
Instruction Queue	Stores pre-fetched instructions (6 bytes max)	
EU	Executes instructions using ALU	
General Registers	Temporary storage and data manipulation	
ALU	Arithmetic and logic processing	
Flags	Reflect results of operations	
A-Bus, C-Bus	Internal data and control signal pathways	

Registers in 8086

General Purpose Registers (8)

Pointer and Index Registers (4)

Segment Registers (4)

Instruction Pointer (IP)

Flag Register (1 total)

A. General Purpose Registers (8)

Used for arithmetic, logic, data transfer operations.

Register	16-bit	8-bit Parts	Usage
AX	AH + AL	AH, AL	Accumulator
BX	BH + BL	BH, BL	Base register
CX	CH + CL	CH, CL	Counter (loops, shifts)
DX	DH + DL	DH, DL	Data register (I/O, MUL/DIV)

B. Pointer and Index Registers (4)

Register	Function
SP	Stack Pointer
BP	Base Pointer
SI	Source Index (used in string operations)
DI	Destination Index

C. Segment Registers (4)

Used to access different memory segments (20-bit address space)

Register	Segment
CS	Code Segment (for IP)
DS	Data Segment
SS	Stack Segment
ES	Extra Segment

♦ These segment registers hold the base addresses used to compute the 20-bit physical address.

D. Instruction Pointer (IP)

• Holds offset of the next instruction in the **code segment (CS)**.

E. Flag Register (1 total)

• 16-bit register with 9 active flags (explained below)

Physical Address Calculation in 8086

8086 uses **Segment: Offset** addressing, combining a **16-bit segment register** and a **16-bit offset** to form a **20-bit physical address**.

Physical Address = $(Segment \times 10h) + Offset$

Example: Let's say:

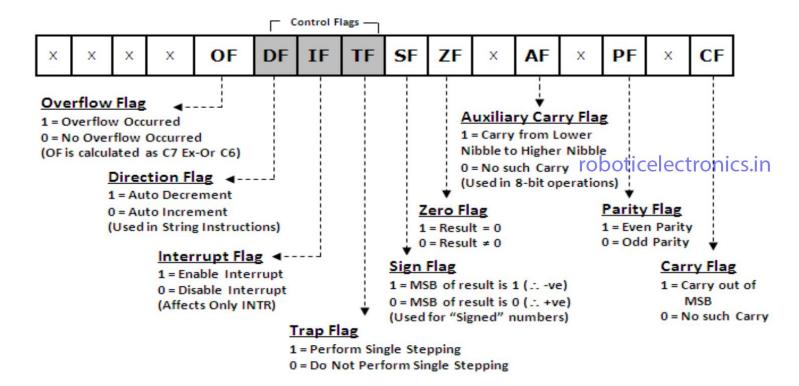
Segment = 1234h Offset = 0010h

Then,

Physical Address =
$$(1234h \times 10h) + 0010h$$

= $12340h + 0010h$
= $**12350h***$

Flags Register in 8086 (16-bit)



A. Status Flags

Indicate results of operations:

- CF Carry Flag
- **PF** Parity Flag
- **AF** Auxiliary Carry Flag
- **ZF** Zero Flag
- SF Sign Flag
- **OF** Overflow Flag

B. Control Flags

Control processor operations:

- **TF** Trap Flag (for debugging)
- **IF** Interrupt Enable Flag
- **DF** Direction Flag (string operations)

Addressing Modes in 8086

An addressing mode specifies how an operand (data) for an instruction is accessed (i.e., whether it is in a register, memory, or part of the instruction itself).

8086 supports several addressing modes to allow flexible memory and data access.

1. Immediate Addressing Mode

- The operand is directly specified in the instruction.
- Data is given as a **constant value**.

MOV AX, 1234h ; $AX \leftarrow 1234h$

Use: Simple data loading.

2. Register Addressing Mode

- The operand is located in a **register**.
- Both source and destination are registers.

$$MOVAX, BX$$
; $AX \leftarrow BX$

Use: Fast data movement within CPU.

3. Direct Addressing Mode

- The memory address of the operand is given explicitly in the instruction.
- Offset is fixed; segment is usually **DS** by default.

MOV AX, [1234h] ; AX ← contents at memory address DS:1234h

Use: Access fixed memory location.

4. Register Indirect Addressing Mode

- The **effective address** of the operand is stored in a register.
- Registers used: **BX**, **BP**, **SI**, or **DI**.
- Segment: **DS** for BX, SI, DI; **SS** for BP.

$$MOVAX$$
, $[BX]$; $AX \leftarrow contents at DS:BX$

Use: Access variable memory locations dynamically.

5. Based Addressing Mode

- A base register holds the offset (BX or BP).
- Optional displacement can be added.
- Segment: **DS** (for BX), **SS** (for BP)

```
MOVAX, [BP]; AX \leftarrow contents at SS:BP
```

Use: Access stack or data memory.

6. Indexed Addressing Mode

- Uses index registers: SI or DI
- Effective address = contents of index register
- Segment default: **DS**

Use: Access array elements.

7. Based-Indexed Addressing Mode

- Combines base register (BX or BP) and index register (SI or DI).
- Effective address = base + index
- Segment default: **DS** or **SS** depending on base

```
MOV AX, [BX + SI]; AX ← contents at DS:(BX+SI)
```

Use: Multidimensional arrays or struct access.

8. Based-Indexed with Displacement

- Most flexible addressing mode.
- Effective address = base + index + displacement
- Allows full offset control.

MOV AX,
$$[BX + SI + 10]$$
; AX \leftarrow contents at DS: $(BX + SI + 10)$

Use: Array element + offset access.

9. Relative Addressing Mode

Used only with jump (JMP) and branch instructions.

• The target address is relative to the **current instruction pointer (IP)**.

JMP SHORT LABEL ; IP \leftarrow IP + displacement to LABEL

Use: Efficient code branching.

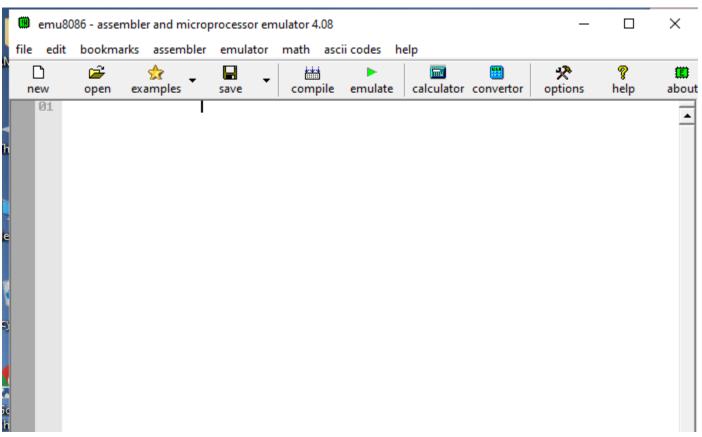
Mode	Syntax Example	Effective Address Formula
Immediate	MOV AX, 1234h	Operand is in instruction
Register	MOV AX, BX	Operand in register
Direct	MOV AX, [1234h]	EA = 1234h
Register Indirect	MOV AX, [BX]	EA = contents of BX
Based	MOV AX, [BP]	EA = BP
Indexed	MOV AX, [SI]	EA = SI
Based + Indexed	MOV AX, [BX + SI]	EA = BX + SI
Based + Indexed + Disp	MOV AX, [BX + SI + 5]	EA = BX + SI + 5
Relative (Jumps only)	JMP SHORT LABEL	New IP = Old IP + displacement

Programming with 8085

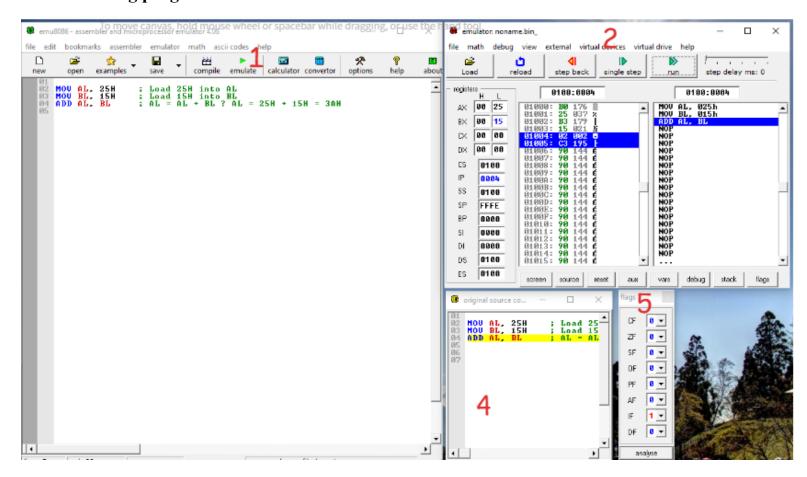
Software Simulator:

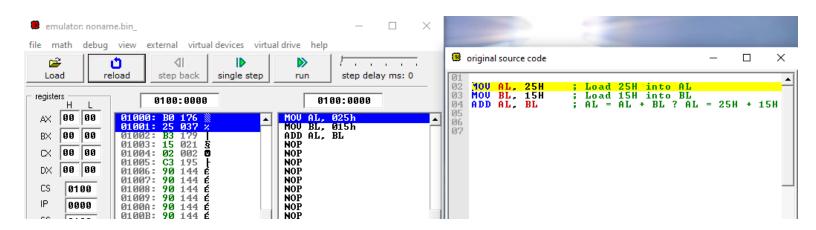
https://github.com/sanjeevlcc/notes_2081/blob/main/Microprocessor_and_CA/TU%20BSC-CSIT%20Microprocessor/emu8086v408.zip

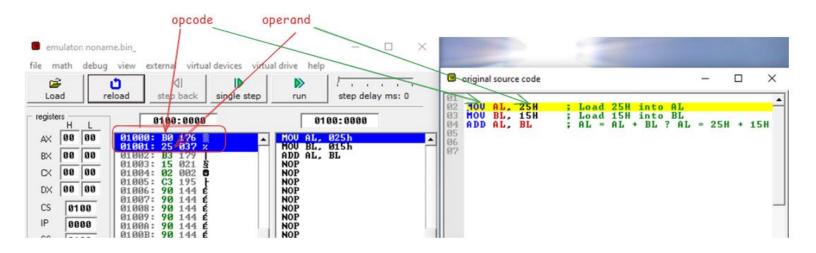


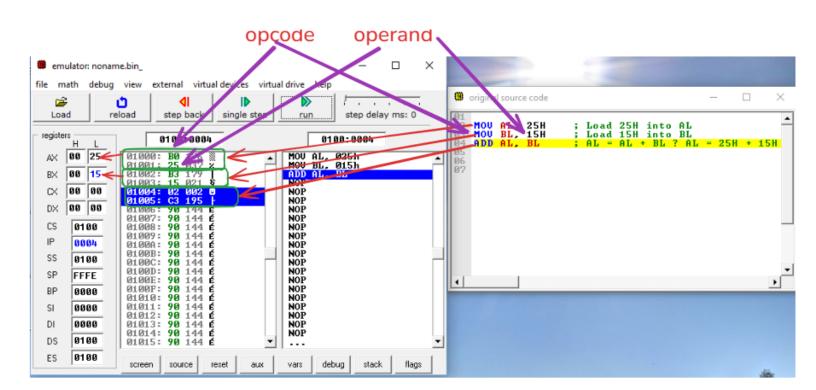


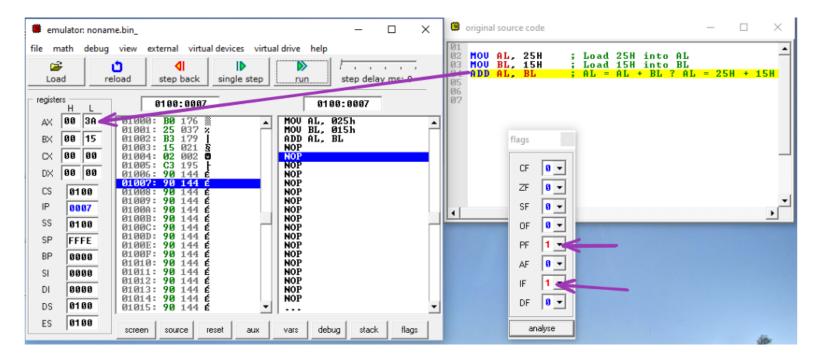
After Running program











Sample 8086 Addition

8 bit addition 12 + 0C eg 1 2 0001 0010 ---- 8 bit ----12 + 0C = 1E (0001 1110) 16 bit addition 1234 + 4321 = 5555 eg 1 2 3 4 0001 0010 0011 0100 -----16 bit ------1234 + 4321 = 5555

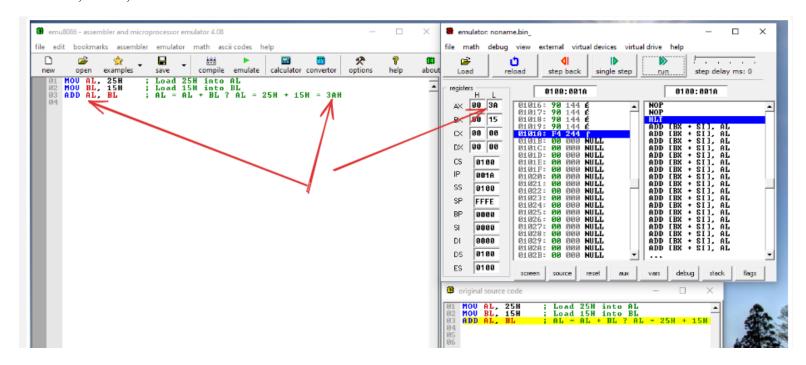
0101 0101 0101 0101

Addition (8-bit)

MOV AL, 25H; Load 25H into AL

MOV BL, 15H; Load 15H into BL

ADD AL, BL ; $AL = AL + BL \rightarrow AL = 25H + 15H = 3AH$

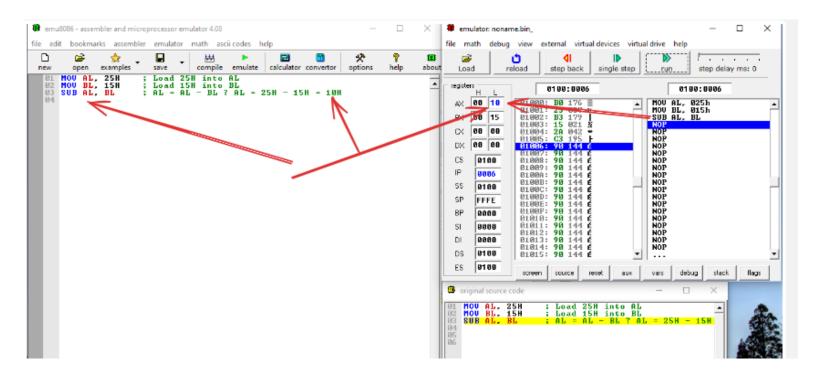


Subtraction (8-bit)

MOV AL, 25H; Load 25H into AL

MOV BL, 15H; Load 15H into BL

SUB AL, BL ; $AL = AL - BL \rightarrow AL = 25H - 15H = 10H$

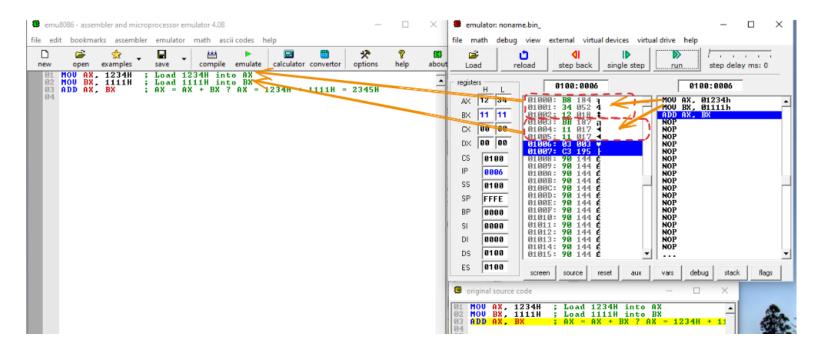


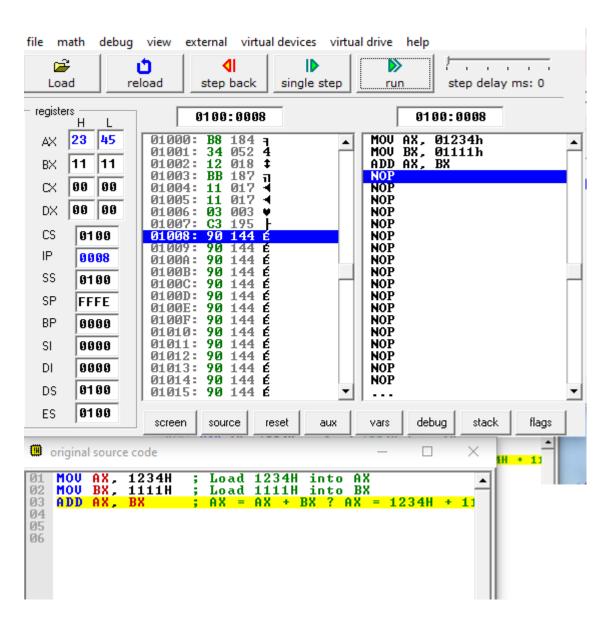
Addition (16-bit)

MOV AX, 1234H; Load 1234H into AX

MOV BX, 1111H; Load 1111H into BX

ADD AX, BX ; $AX = AX + BX \rightarrow AX = 1234H + 1111H = 2345H$



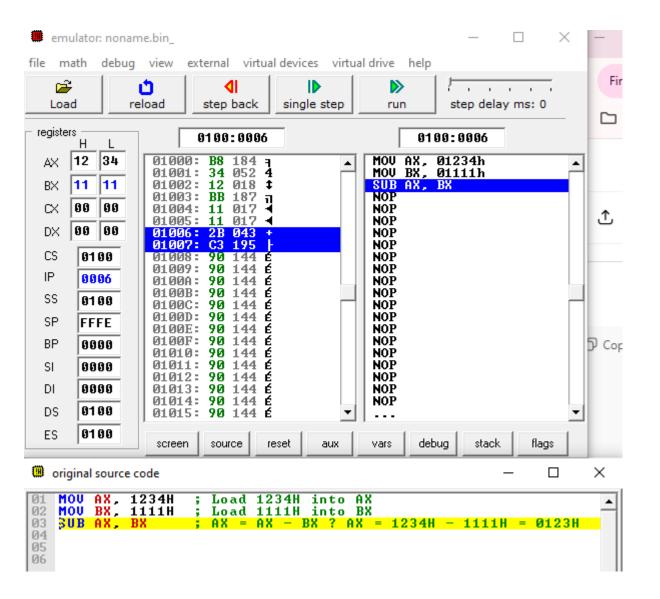


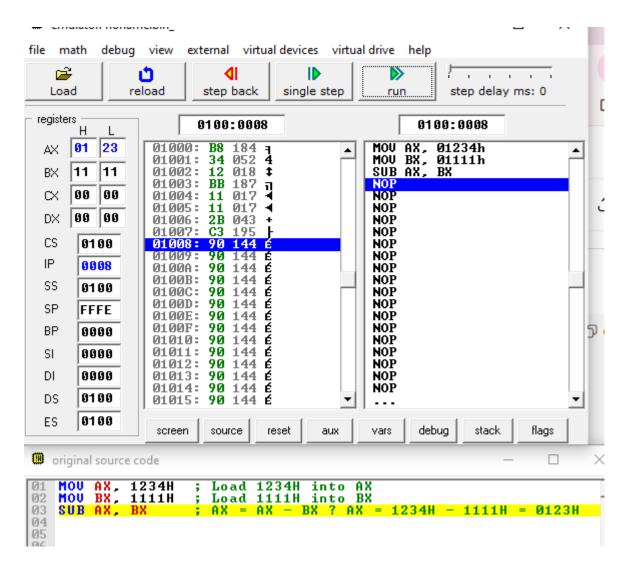
Subtraction (16-bit)

MOV AX, 1234H; Load 1234H into AX

MOV BX, 1111H; Load 1111H into BX

SUB AX, BX ; $AX = AX - BX \rightarrow AX = 1234H - 1111H = 0123H$





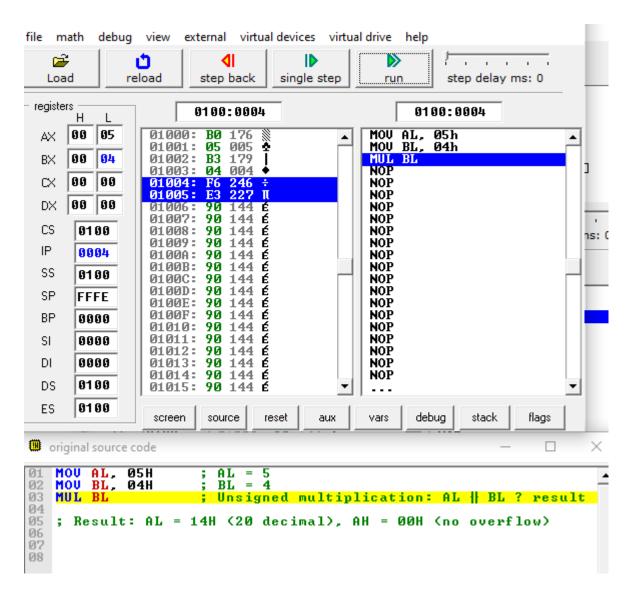
Multplication (8-bit)

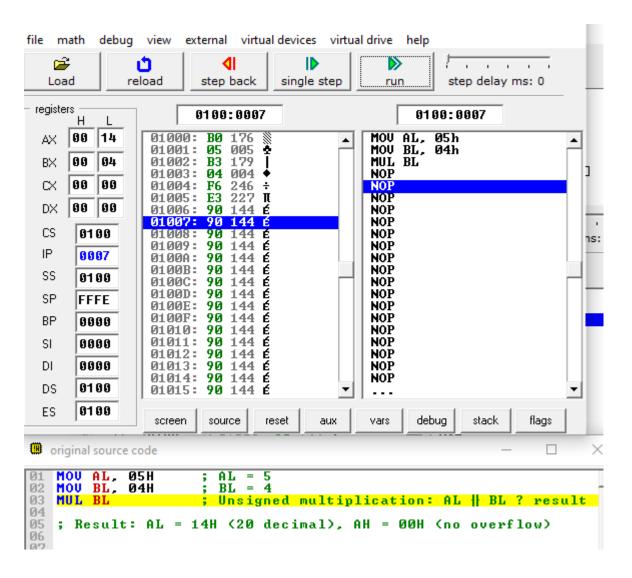
MOVAL, 05H ; AL = 5

MOV BL, 04H; BL = 4

MUL BL ; Unsigned multiplication: $AL \times BL \rightarrow result$ in AX

; Result: AL = 14H (20 decimal), AH = 00H (no overflow)





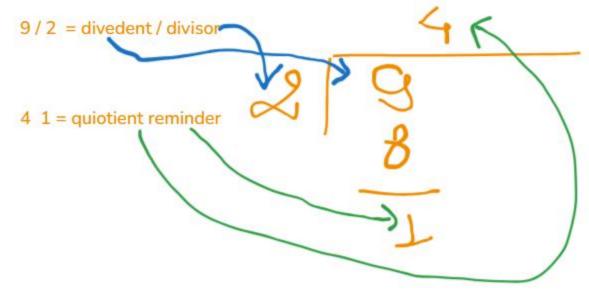
Division (8-bit)

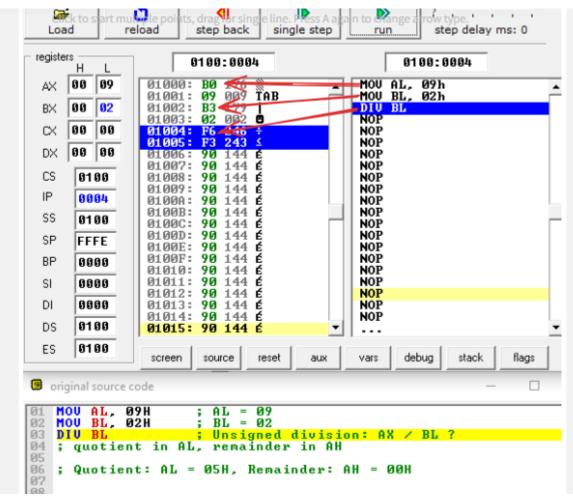
MOV AL, 14H; AL = 20

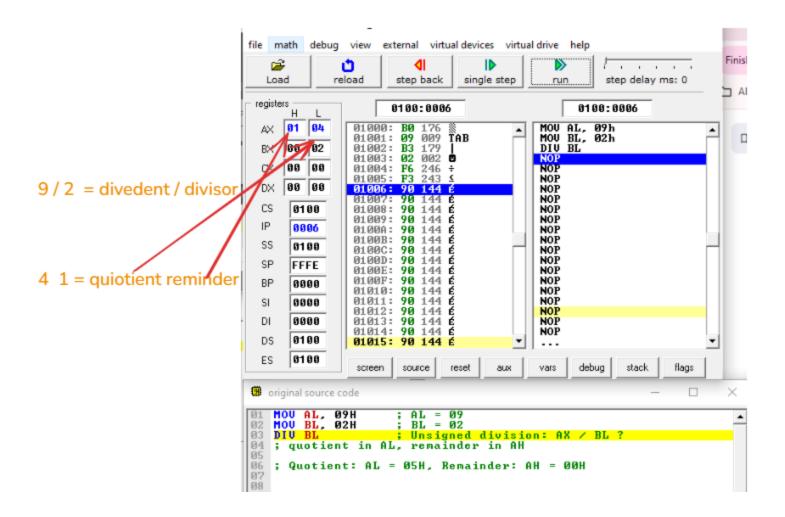
MOV BL, 04H; BL = 4

DIV BL; Unsigned division: $AX / BL \rightarrow$ quotient in AL, remainder in AH

; Quotient: AL = 05H, Remainder: AH = 00H







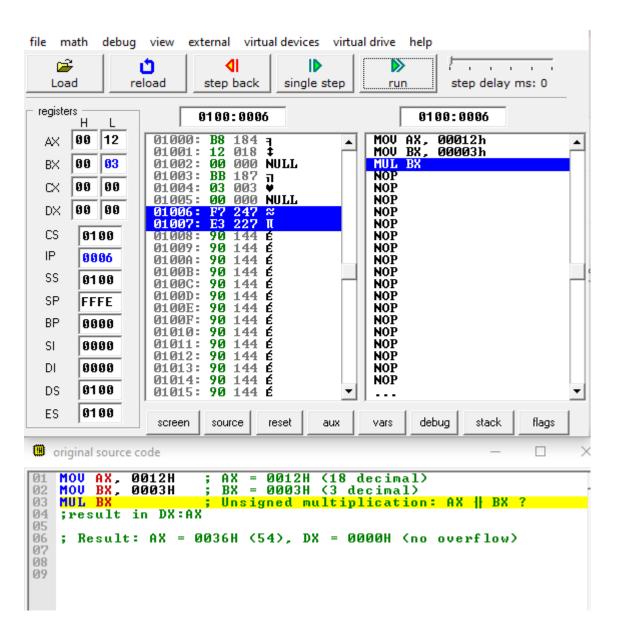
Multplication (16-bit)

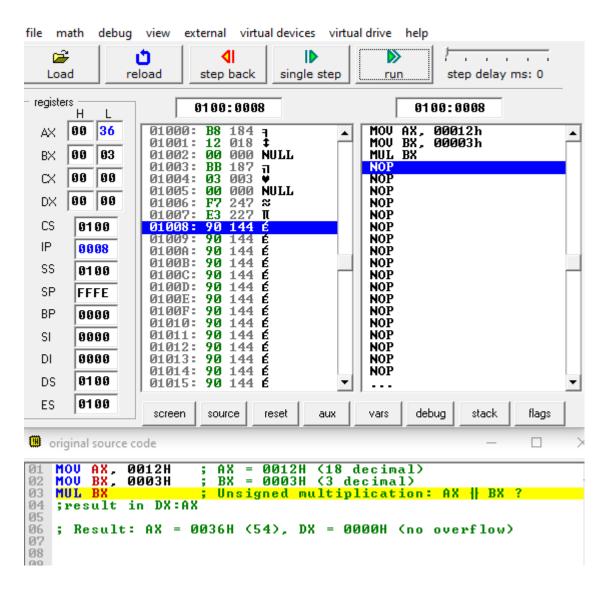
MOV AX, 0012H; AX = 0012H (18 decimal)

MOV BX, 0003H; BX = 0003H (3 decimal)

MUL BX ; Unsigned multiplication: $AX \times BX \rightarrow result$ in DX:AX

; Result: AX = 0036H (54), DX = 0000H (no overflow)





Division (16-bit)

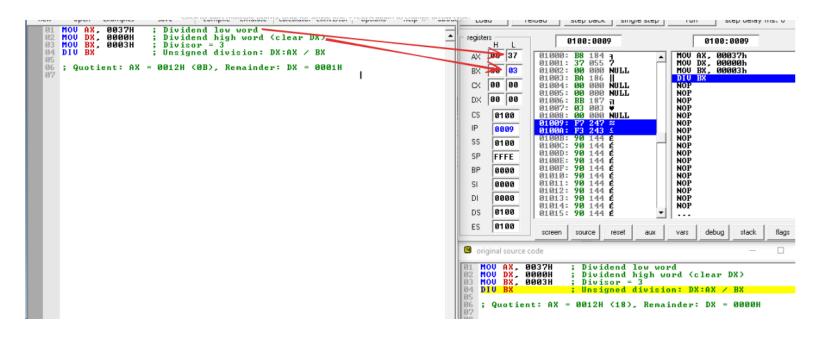
MOV AX, 0037H; Dividend low word

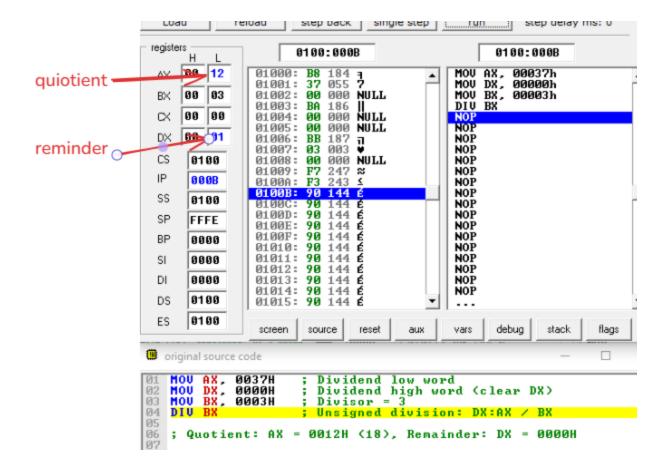
MOV DX, 0000H; Dividend high word (clear DX)

MOV BX, 0003H; Divisor = 3

DIV BX; Unsigned division: DX:AX / BX

; Quotient: AX = 0012H (0B), Remainder: DX = 0001H





Simple "Hello World" Program

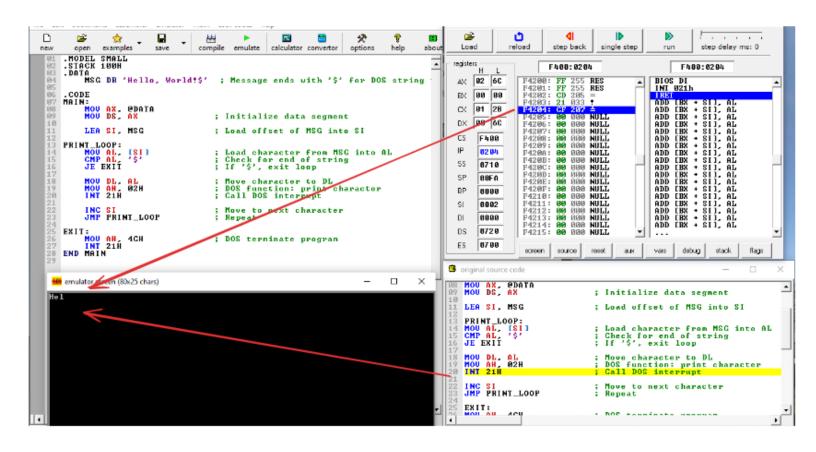
```
.MODEL SMALL
.STACK 100H
.DATA
  MSG DB 'Hello, World!$'; Message ends with '$' for DOS string termination
.CODE
MAIN:
  MOV AX, @DATA
  MOV DS, AX
                     ; Initialize data segment
                     ; Load offset of MSG into SI
  LEA SI, MSG
PRINT LOOP:
  MOV AL, [SI]
                    ; Load character from MSG into AL
  CMPAL, '$'
                   ; Check for end of string
                  ; If '$', exit loop
  JE EXIT
  MOV DL, AL
                     ; Move character to DL
                      ; DOS function: print character
  MOV AH, 02H
                  ; Call DOS interrupt
  INT 21H
  INC SI
                 ; Move to next character
  JMP PRINT LOOP
                         ; Repeat
```

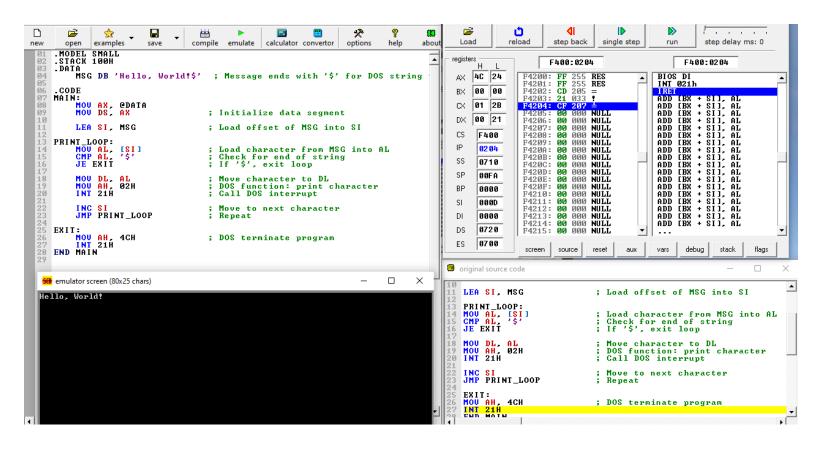
EXIT:

MOV AH, 4CH ; DOS terminate program

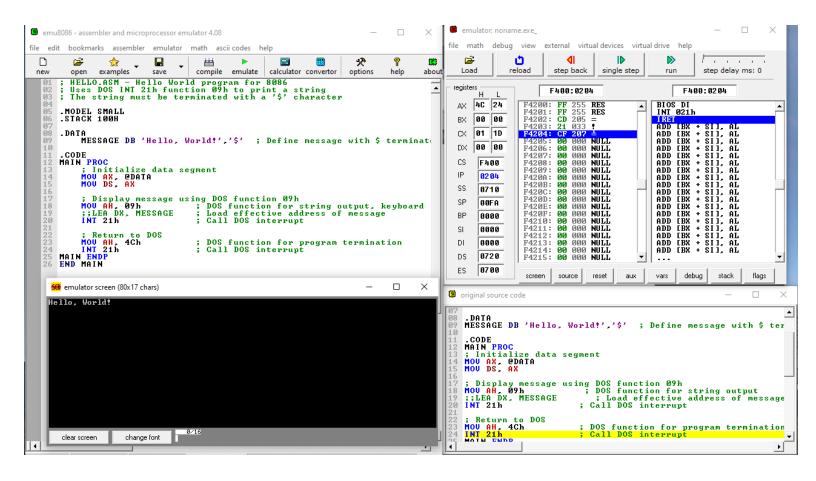
INT 21H

END MAIN





OR



- ; HELLO.ASM Hello World program for 8086
- ; Uses DOS INT 21h function 09h to print a string
- ; The string must be terminated with a '\$' character

.MODEL SMALL

.STACK 100H

.DATA

MESSAGE DB 'Hello, World!','\$'; Define message with \$ terminator

.CODE

MAIN PROC

; Initialize data segment

MOV AX, @DATA

MOV DS, AX

; Display message using DOS function 09h

MOV AH, 09h ; DOS function for string output, keyboard

;;LEA DX, MESSAGE ; Load effective address of message

INT 21h ; Call DOS interrupt

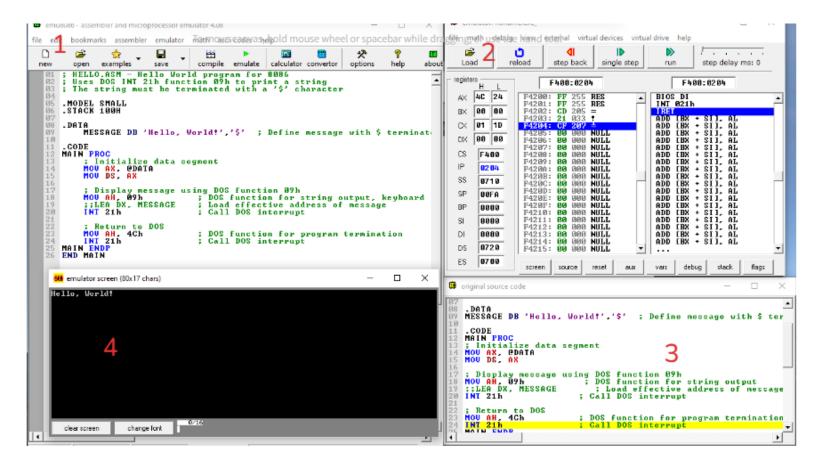
; Return to DOS

MOV AH, 4Ch ; DOS function for program termination

INT 21h ; Call DOS interrupt

MAIN ENDP

END MAIN



Palindrome

.MODEL SMALL

.STACK 100H

.DATA

; Define the test string (you can change this)

STRING DB 'MADAM', '\$' ; String to check (must be uppercase)

STR LEN EQU 5 ; Length of the string (excluding \$)

```
; Output messages
MSG_YES DB 'Palindrome$', 0
MSG_NO DB 'Not Palindrome$', 0
.CODE
MAIN PROC
; Set up segments
MOVAX, @DATA
MOV DS, AX
; Setup SI and DI to compare characters
LEA SI, STRING
                   ; SI -> start of the string
LEA DI, STRING
ADD DI, STR_LEN - 1 ; DI -> end of the string (before $)
MOV CX, STR_LEN/2 ; Number of comparisons (half length)
MOV BX, 0
           ; BX = mismatch flag
CHECK_LOOP:
MOVAL, [SI]
MOV DL, [DI]
CMP AL, DL
JNE NOT_PALINDROME
INC SI
DEC DI
LOOP CHECK_LOOP
; If no mismatches found
LEA DX, MSG_YES
```

JMP PRINT_RESULT

NOT_PALINDROME:

LEA DX, MSG_NO

PRINT_RESULT:

MOVAH, 09H; DOS function to print string

INT 21H

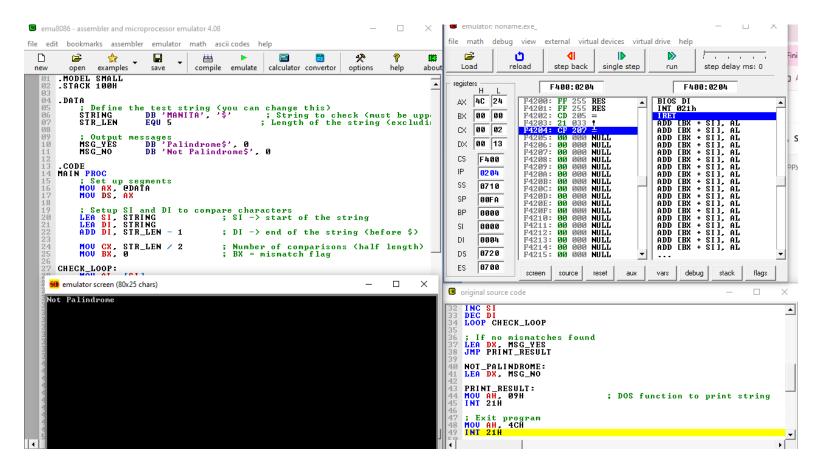
; Exit program

MOV AH, 4CH

INT 21H

MAIN ENDP

END MAIN



Subtraction (16-bit)

80286: Architecture, Registers, (Real/Protected mode), Privilege levels, descriptor cache, Memory access in GDT and LDT, multitasking, addressing modes, flag register

✓ Summary Table

Bus Type	Width	Direction	Purpose
Address	24-bit	Unidirectional	Specifies memory/I/O addresses
Data	16-bit	Bidirectional	Transfers data and instructions
Control	Varies	Mixed	Controls operations like read/write

Feature	Description
Address Bus	24-bit → 16MB memory
Real Mode	8086-compatible, 1MB memory, no protection
Protected Mode	Enables multitasking, memory protection
GDT/LDT	Define memory segments for system and tasks
Multitasking	Through Task State Segment and task descriptors
Descriptor Cache	Internal registers holding descriptor info
Privilege Levels	0 (kernel) to 3 (user) security levels
Flags Register	Status and control flags for arithmetic, execution

1. Address Bus

Definition:

• The Address Bus is used by the processor to specify memory addresses or I/O ports to read from or write to.

Key Features:

• Width: 24-bit address bus

• Addressable Memory: 2²⁴ = 16 MB of physical memory

Unidirectional: Data flows only from CPU to memory/I/O

Use:

• Sends memory addresses to RAM, ROM, or I/O devices.

2. Data Bus

Definition:

• The Data Bus carries actual data values between the CPU, memory, and I/O devices.

\rightarrow Key Features:

• Width: 16-bit data bus

• **Bidirectional:** Data flows **both ways** (to and from the CPU)

Use:

• Transfers instructions, operand values, and I/O data.

• 16 bits at a time \rightarrow suitable for word operations.

✓ 3. Control Bus

Definition:

• The Control Bus carries control signals that manage and coordinate all CPU activities.

Signal	Description
RD	Read – Memory or I/O read operation
WR	Write – Memory or I/O write operation
ALE	Address Latch Enable – Separates address/data
DT/R	Data Transmit/Receive – Bus direction
DEÑ	Data Enable – Enables external data bus
READY	Wait state insertion from slow devices
INTĀ	Interrupt Acknowledge
NMI	Non-maskable interrupt
LOCK	Locks system bus for atomic operations
HOLD/HLDA	Bus control for DMA

1. Architecture Overview

- Introduced by Intel in **1982**, 80286 (or iAPX 286) is a **16-bit** microprocessor.
- Clock speed: 6 MHz 25 MHz.
- 24-bit address bus → Can address up to 16 MB of physical memory.
- Supports **Real Mode** (8086-compatible) and **Protected Mode** (advanced features).



2. Registers of 80286

♦ General-Purpose Registers (16-bit):

• AX (Accumulator)

• BX (Base)

• CX (Count)

• DX (Data)

Segment Registers:

- CS (Code Segment)
- DS (Data Segment)
- SS (Stack Segment)
- ES (Extra Segment)

♦ Pointer and Index Registers:

- SP (Stack Pointer), BP (Base Pointer)
- SI (Source Index), DI (Destination Index)

♦ Instruction Pointer and Flags:

• IP (Instruction Pointer)

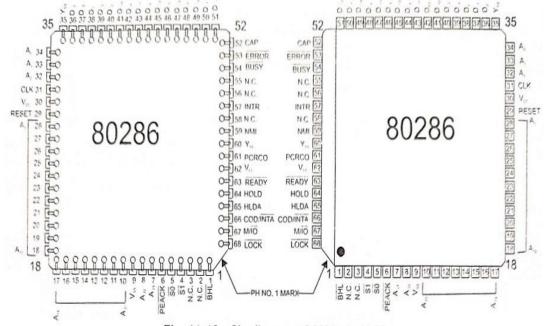


Fig. 11.18 Pin diagram of 80286 in PLCC

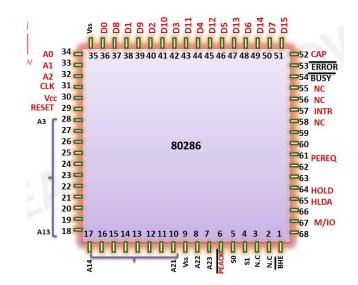
• FLAGS (Status and control bits)

✓ 3. Real Mode vs Protected Mode

Feature	Real Mode	Protected Mode
Addressable Memory	1 MB (20-bit addressing)	16 MB physical, 1 GB virtual memory
Mode Type	Compatible with 8086	Advanced features enabled
Multitasking	Not supported	Supported
Protection	None	Memory protection, privilege levels
Switching	Default on reset	Entered by setting PE (CR0 bit 0)

✓ 4. Privilege Levels

- 4 Privilege Levels (Ring 0 to Ring 3):
 - o Ring 0: Most privileged (Kernel/OS)
 - o Ring 3: Least privileged (User apps)
- Used for memory and I/O protection.
- CPL (Current Privilege Level) vs DPL (Descriptor Privilege Level).



✓ 5. Descriptor Cache (Shadow Registers)

- Internal registers that store segment descriptor info (base, limit, access rights) from memory.
- When a segment is loaded (e.g., MOV DS, AX), the CPU fetches the **descriptor from GDT/LDT** and stores in **descriptor cache**.
- Fast access to segment properties without repeatedly accessing memory.

✓ 6. Memory Access with GDT and LDT

- **♦** Global Descriptor Table (GDT)
 - System-wide table with **segment descriptors**.
 - Each entry defines: Base address, Limit, Access rights.
 - Used for code, data, stack segments, etc.
- **♦** Local Descriptor Table (LDT)
 - Specific to each task/process.
 - Defines memory accessible to that task.
 - Enables task isolation and memory protection.

7. Multitasking

- 80286 supports hardware-level multitasking using the Task State Segment (TSS).
- Each task has:
 - Own register set, LDT, stack, flags.
 - o Described using a **Task Descriptor** in GDT.
- Task Switching involves:
 - Saving current task state to its TSS.
 - Loading new task state from next TSS.
 - Updating Task Register and control flags.

✓ 8. Addressing Modes in 80286

Supports all 8086 addressing modes:

- **♦** Register and Immediate Modes:
 - MOV AX, BX
 - ADD AX, 1234H

♦ Memory Addressing Modes:

• **Direct**: [1234H]

• Register Indirect: [BX], [SI], [DI]

• Based: [BX+SI], [BX+DI]

• **Based Indexed + Displacement**: [BX+SI+10H]

→ Uses Effective Address (EA) computation:

EA = Base + Index + Displacement

9. Flag Register (FLAGS)

Bit	Flag Name	Description
0	CF (Carry)	Carry out from MSB
2	PF (Parity)	Set if even number of 1's in LSB
4	AF (Auxiliary)	Carry from bit 3 to bit 4
6	ZF (Zero)	Set if result is zero
7	SF (Sign)	Set if result is negative
8	TF (Trap)	Enables single-step mode
9	IF (Interrupt)	Enables/disables hardware interrupts
10	DF (Direction)	Controls string operation direction
11	OF (Overflow)	Set if signed overflow occurs

80386:

Architecture, Register organization, Memory access in protected mode, Paging