EC-310 Microprocessor & Microcontroller based Design

Dr. Taimoor Zahid

Department of Computer and Software Engineering (DC&SE)

College of Electrical and Mechanical Engineering (CEME)

National University of Sciences and Technology (NUST)

Week # 1

Instructor

Biography of Instructor

- **Name: Taimoor Zahid, Ph.D.**
- **Office: Room 3124 # (First Floor) DC&SE**
- **Email: tzahid@ceme.nust.edu.pk**

Course Syllabus (Tentative)

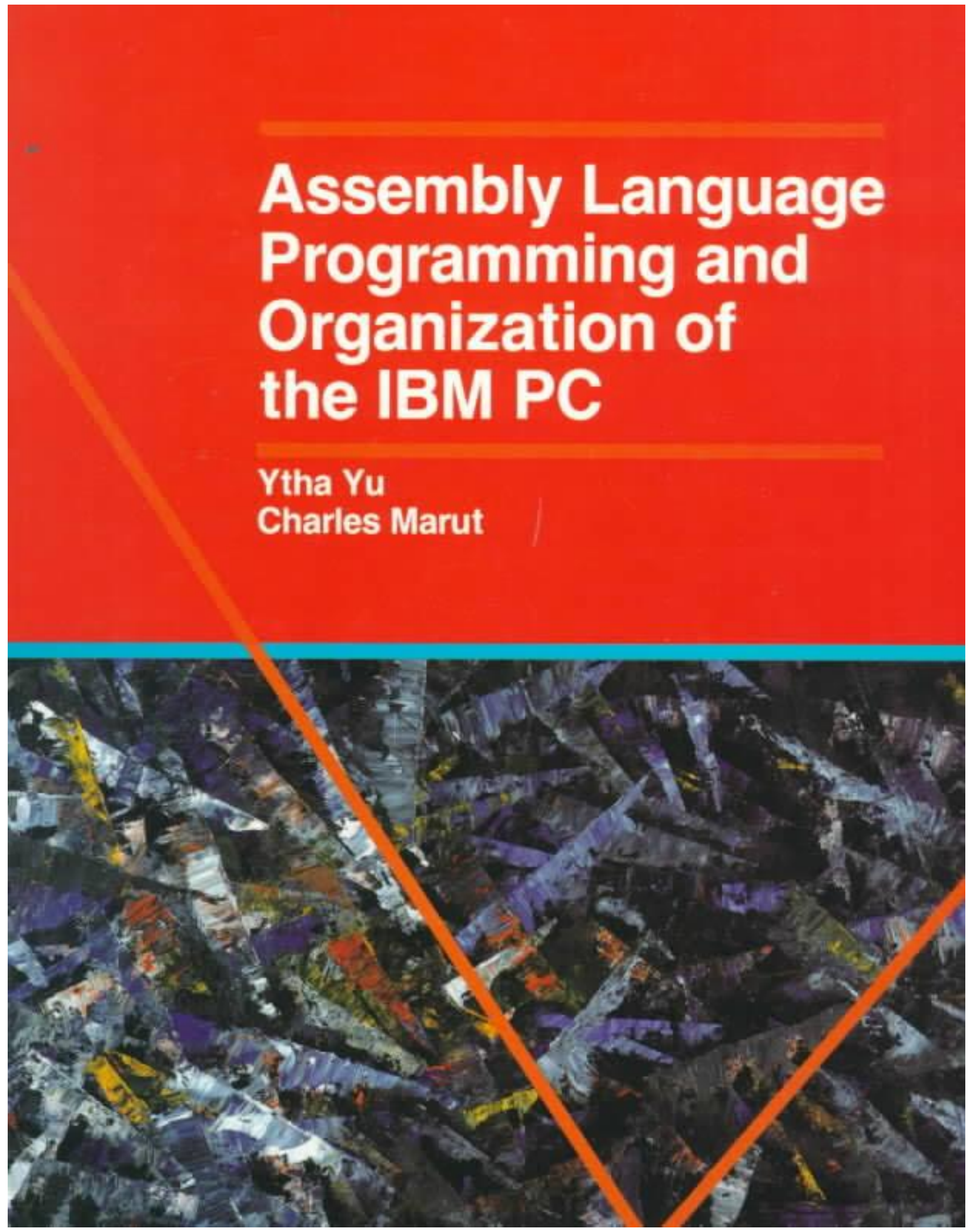
Week 1	Intel 8086 Microprocessor Architecture, Assembly Programming	
Week 2	Assembly Programming-**8086**	Assignment
Week 3	Assembly Programming-**8086**	
Week 4	Assembly Programming-**8086**	Assignment
Week 5	Assembly Programming-**8086**	Project
Week 6	**8086** Hardware Interfacing	Assignment
Week 7	**8086** Hardware Interfacing	
Week 8	**PIC18F** Hardware Interfacing	Midterm

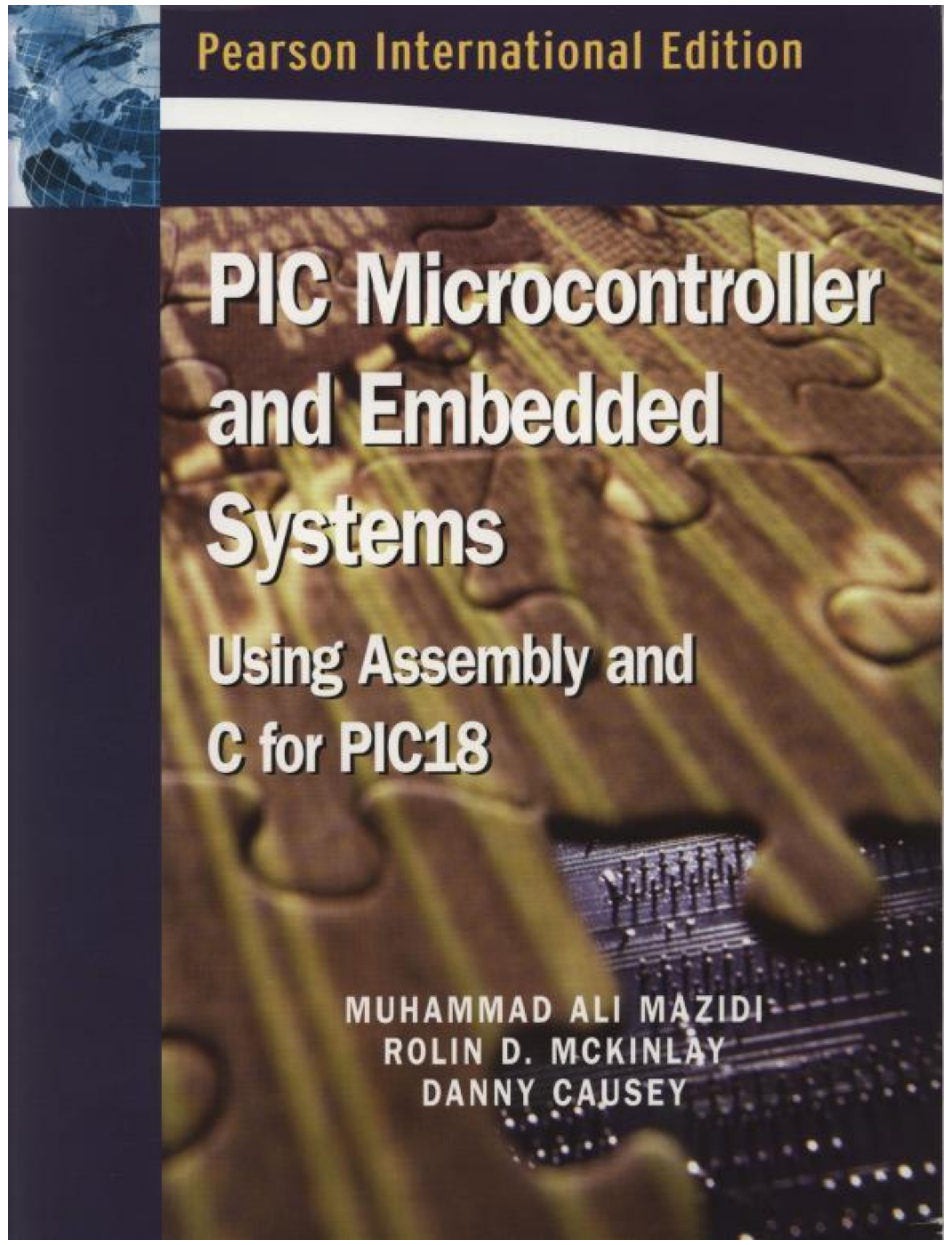
Course Syllabus (Tentative)

Week 9	Microcontroller Programming & Interfacing-**PIC18F**	Assignment
Week 10	Microcontroller Programming & Interfacing-**PIC18F**	
Week 11	Microcontroller Programming & Interfacing-**PIC18F**	Project-Mid
Week 12	Microcontroller Programming & Interfacing-**PIC18F**	Assignment
Week 13	Microcontroller Programming & Interfacing-**PIC18F**	
Week 14	Microcontroller Programming & Interfacing-**PIC18F**	Assignment
Week 15	Microcontroller Programming & Interfacing-**PIC18F**	Project-Final
Week 16	Final Exam	

Text Books

- Ytha Y. Yu and Charles Marut, "Assembly Language Programming and Organization of the IBM PC".
- *M.Ali Mazidi, "PIC Microcontroller and Embedded Systems using Assembly and C for PIC18".*

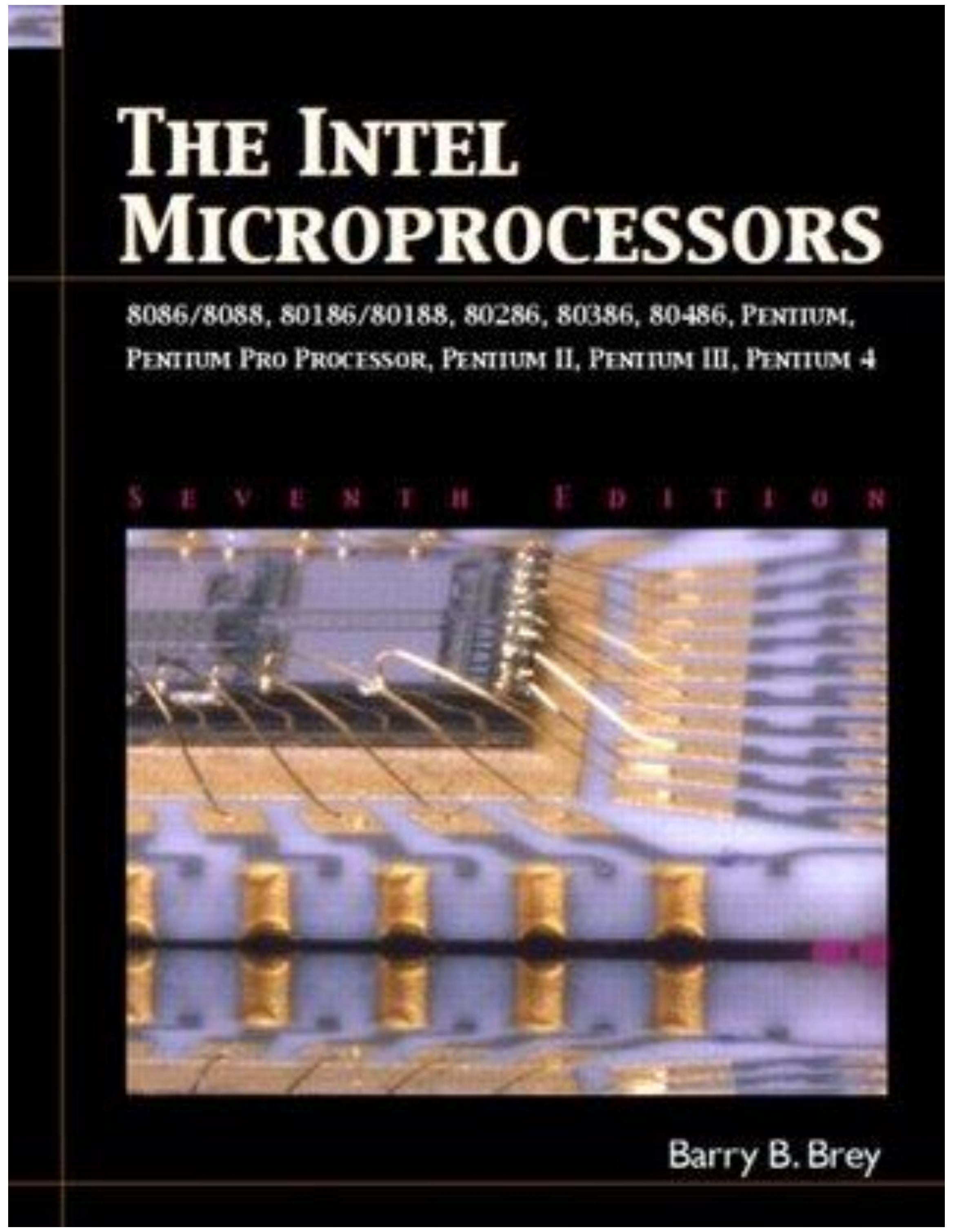




Microprocessor & Microcontroller based Design, Fall 2022

References

- Barry B. Brey, “The Intel Microprocessors 8086/8088, 80186, 80286, 80386, 80486, Pentium and Pentium Pro Processor, Pentium-II, Pentium-III, Pentium-4, Architecture, Programming and Interfacing”, 7th Edition, Prentice-Hall.

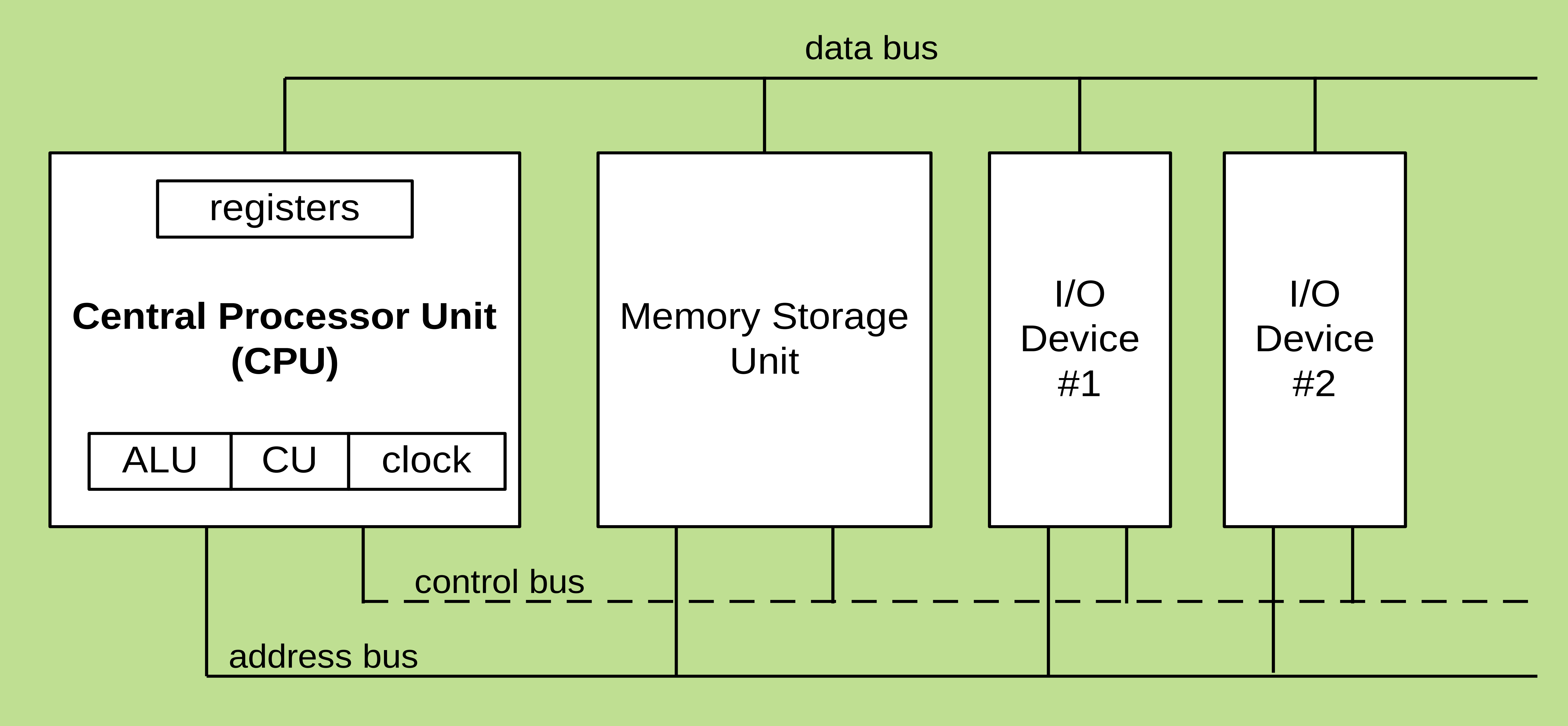


General Course Policy

- **Attendance Policy**
  - Attendance is essential to success in this class
  - Students are expected to attend every lecture
- **Office Hours**
  - Friday 8:45 am - 10:30 pm and Tuesday 10:00 am - 12:00 pm
  - Otherwise, appointment is recommended in advance
- **Assignments and Project**
  - Will be announced during lectures
- **Quizzes**
  - Quizzes will be taken during regular class time. Majority of them will be unannounced.

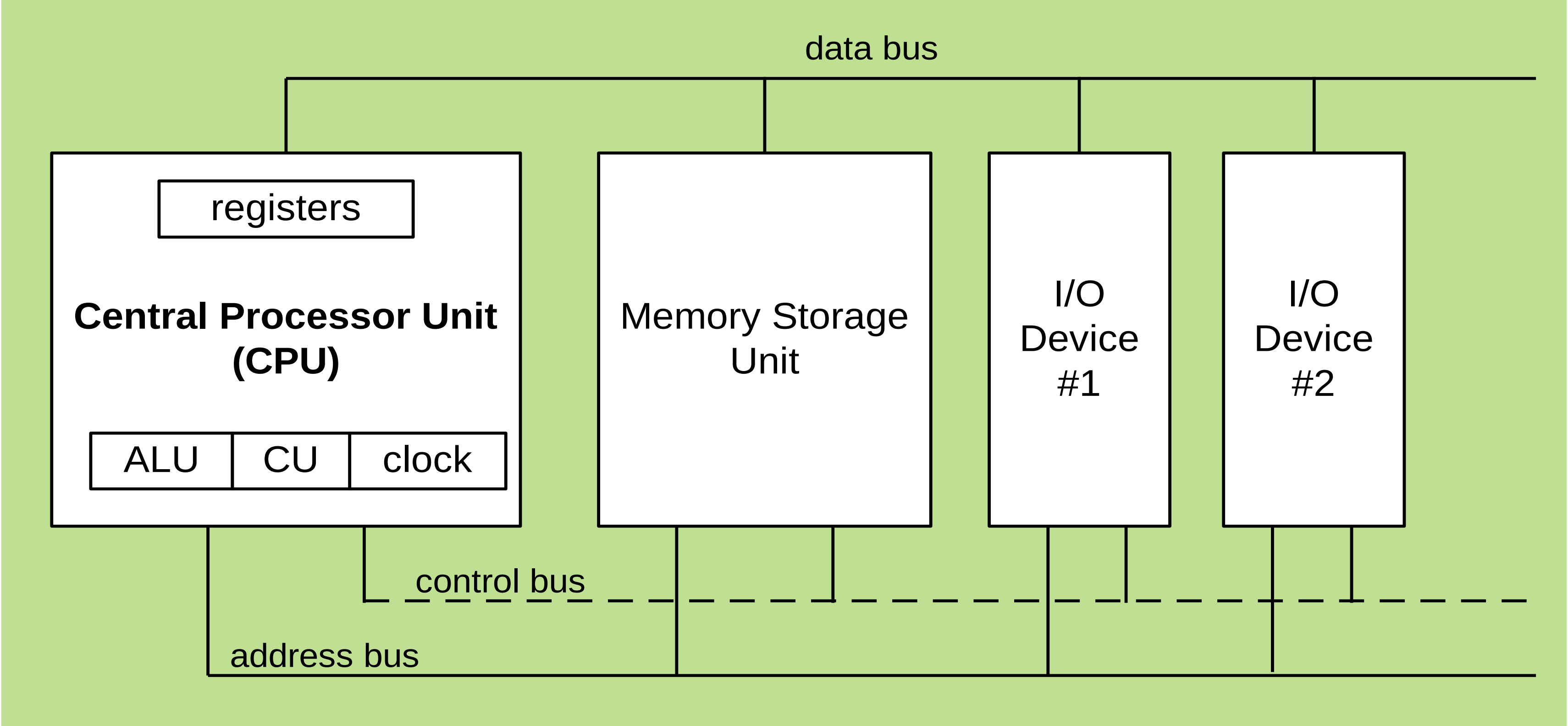
Components of a Microcomputer System

- **Central Processing Unit (CPU)**
- **Memory**
- **I/O Devices or Peripherals**



- The clock synchronizes the internal operations of the CPU with other system components.
- The control unit (CU) coordinates the sequencing of steps involved in executing machine instructions.
- The arithmetic logic unit (ALU) performs arithmetic operations such as addition and subtraction, and logical operations such as AND, OR, and NOT.

Components of a Microcomputer System

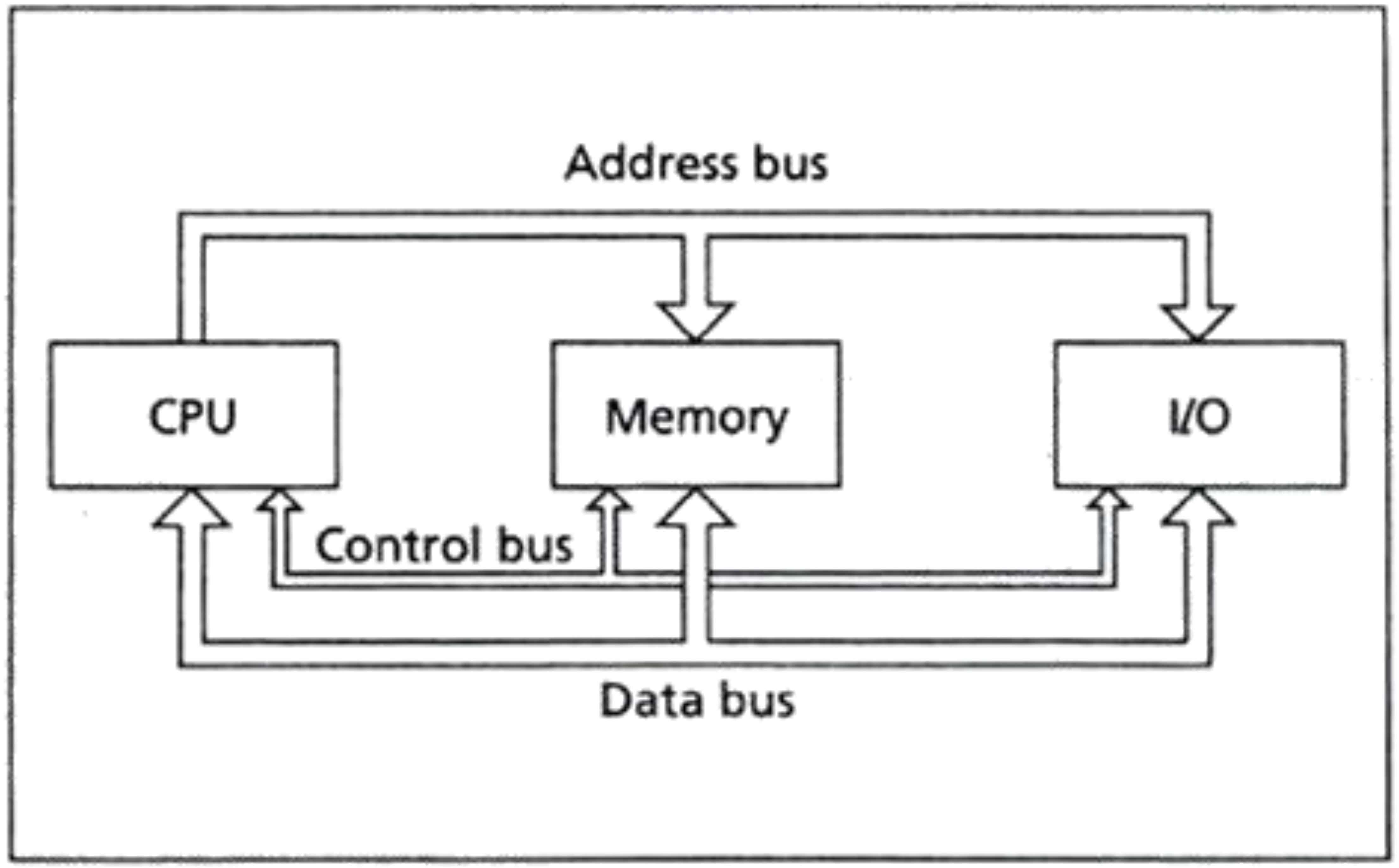


Bus Connections of Microcomputer

Buses – Set of wires or Interconnections b/w CPU, Memory, I/Os

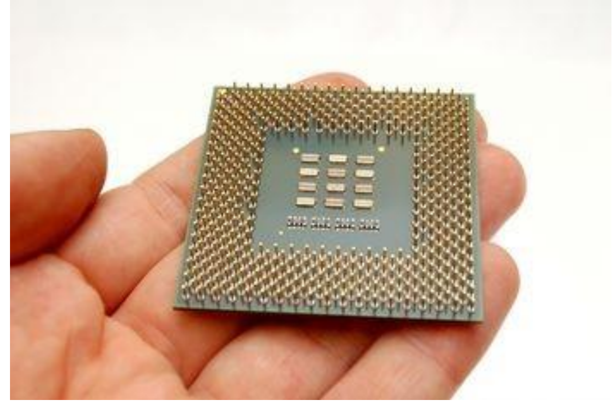
- Address Bus
- Data Bus
- Control

  (contains binary signals to sync. The actions of all)



Bus

Components of a Microcomputer System



Central Processing Unit (CPU)

- **Single-chip processor called microprocessor**
- **Brain of the computer**
- **Controls all operations**

Intel 8086 Microprocessor as CPU

- **Execution Unit (EU)**
  - Executes Instructions
  - Arithmetic Logic Unit (+,-,/,x) (and, or, not)
- **Bus Interface Unit (BIU)**
  - Facilitates communication b/w CPU, Memory, I/Os
  - EU and BIU connected through Internal Bus
  - Instruction Prefetch

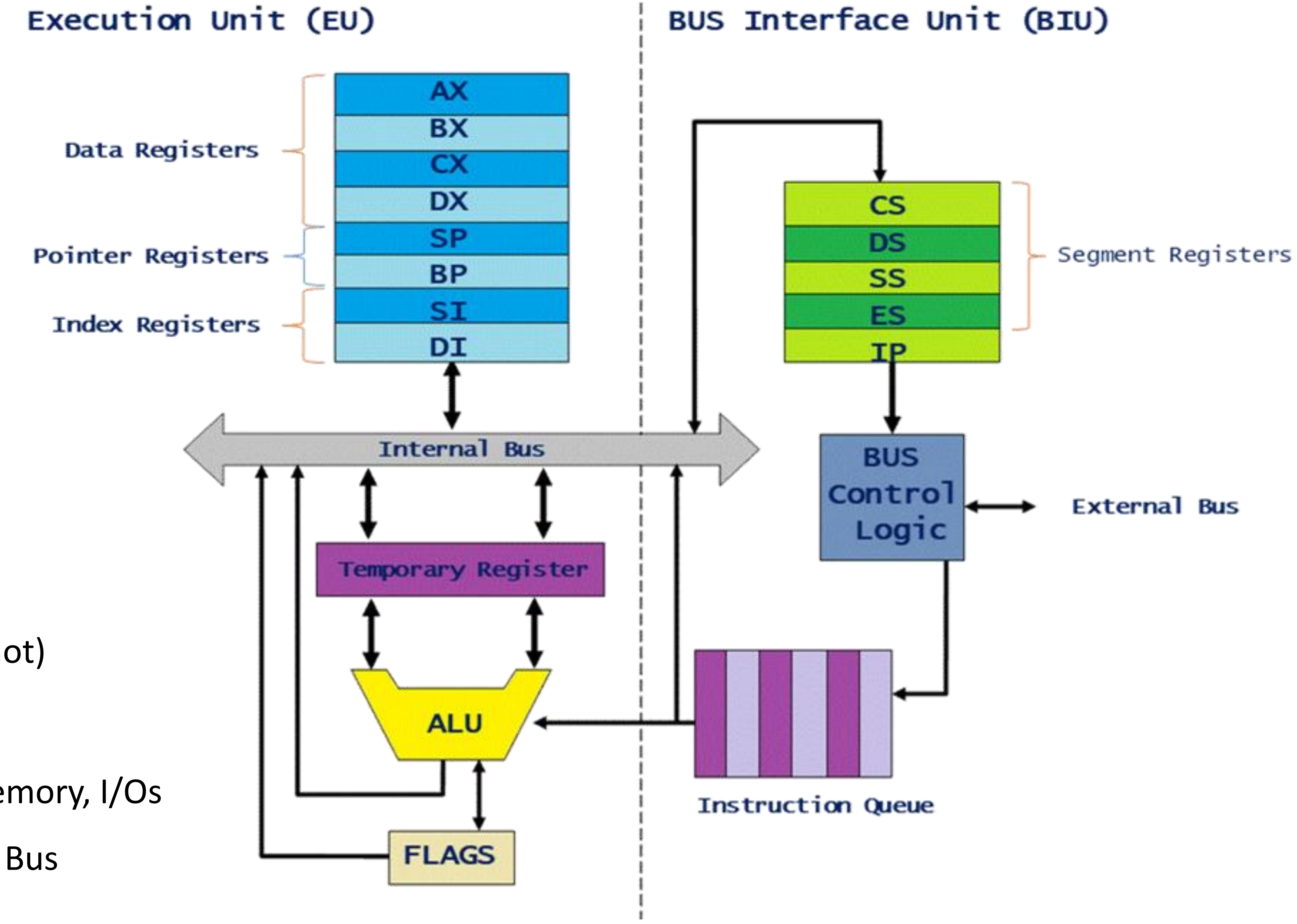
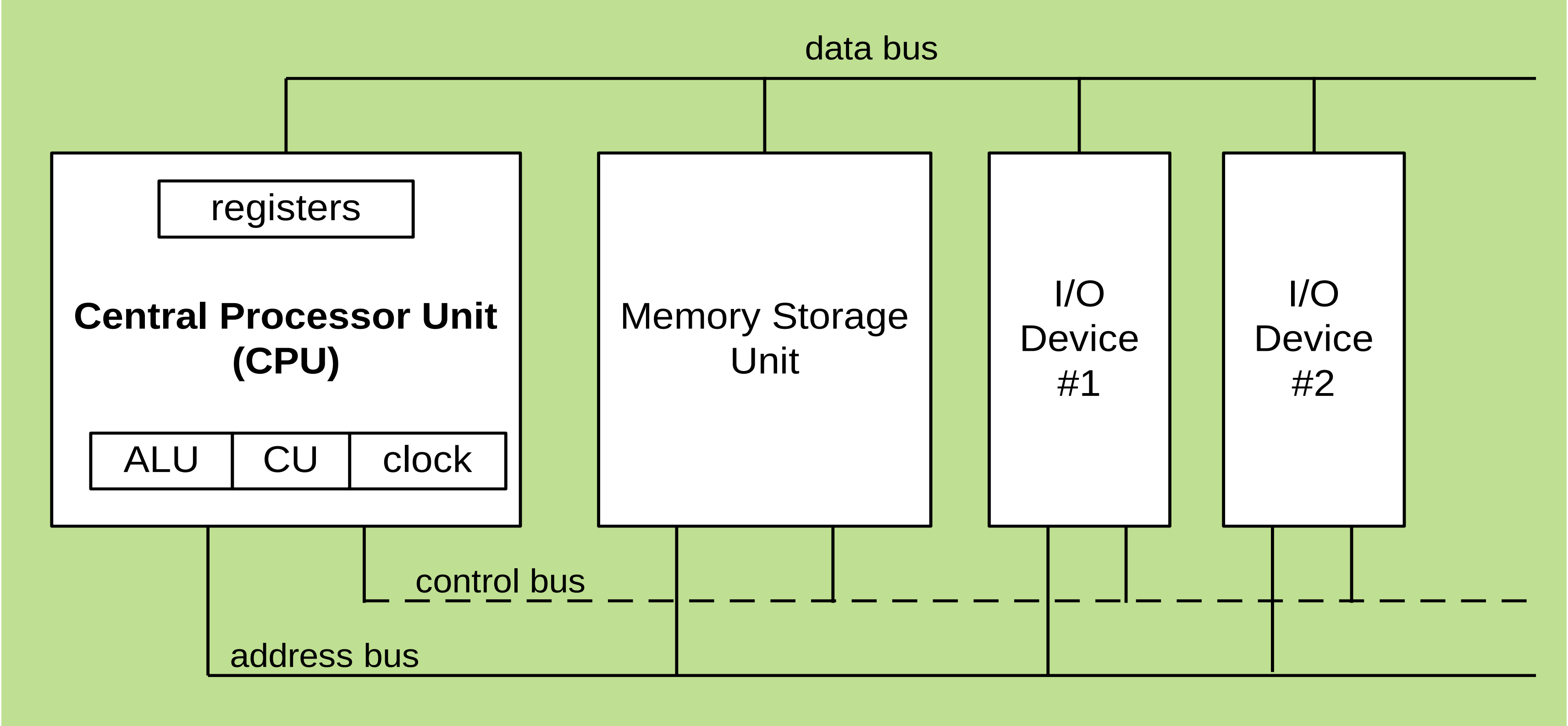


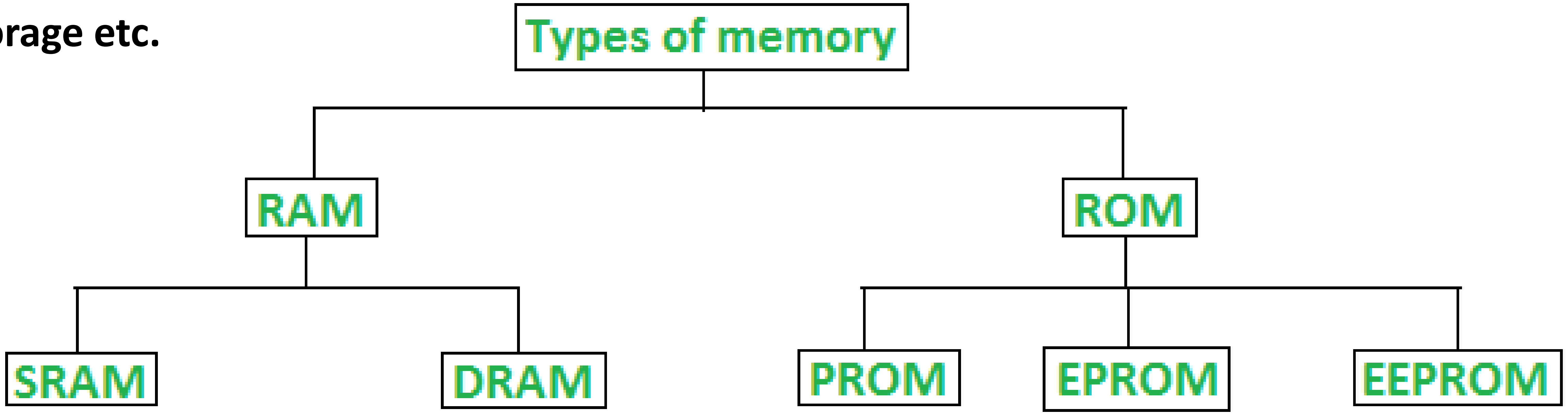
Fig. Internal Architecture of Intel 8086 Microprocessor

Components of a Microcomputer System



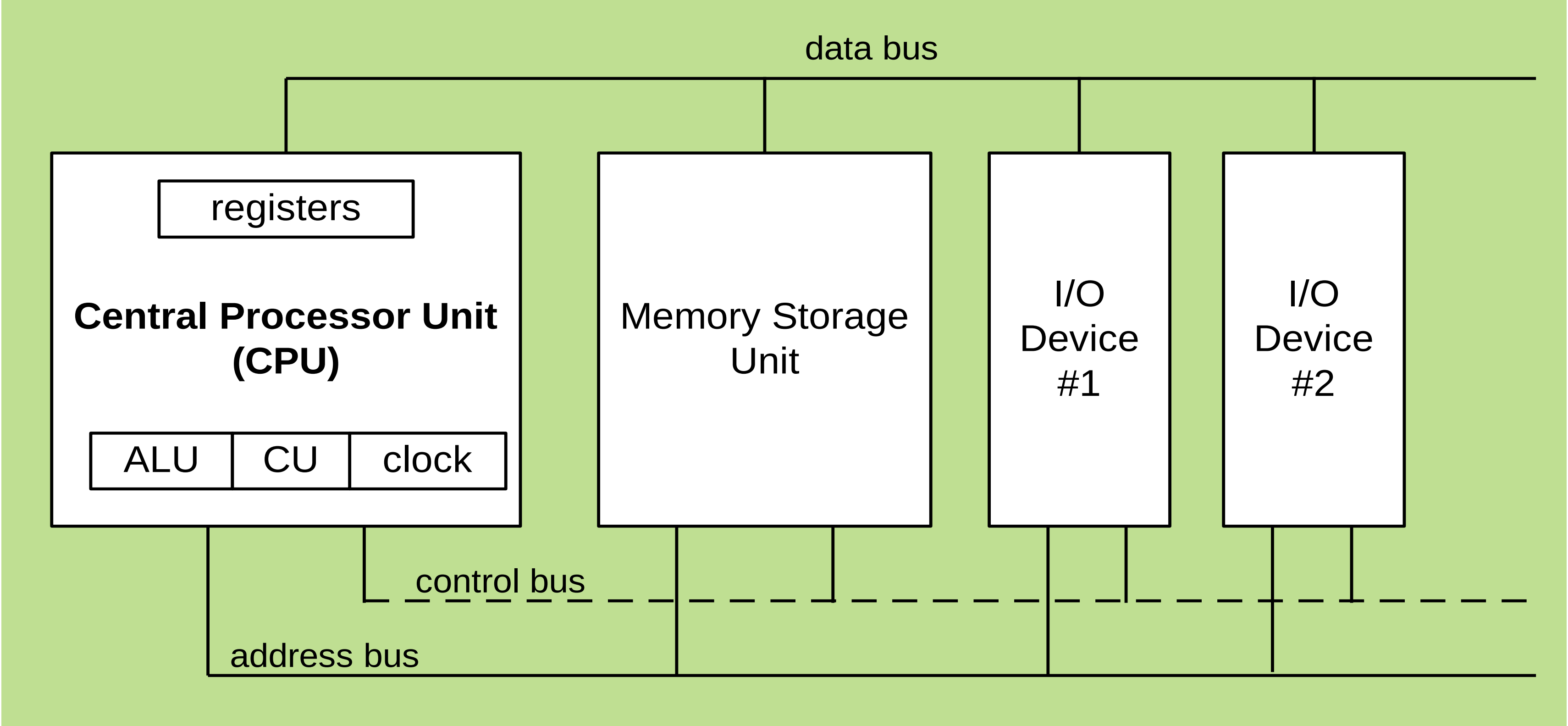
Memory Units

- **RAM (Random Access Memory)**
  - Primary / Main memory, Volatile, Read/Write, Fast
- **ROM (Read Only Memory)**
  - Non-volatile, Read only, Stores manufacturers or startup programs / firmware's
- **Disk Drives, SSDs, Flash, Network/Cloud Storage etc.**



Classification of computer memory

Components of a Microcomputer System



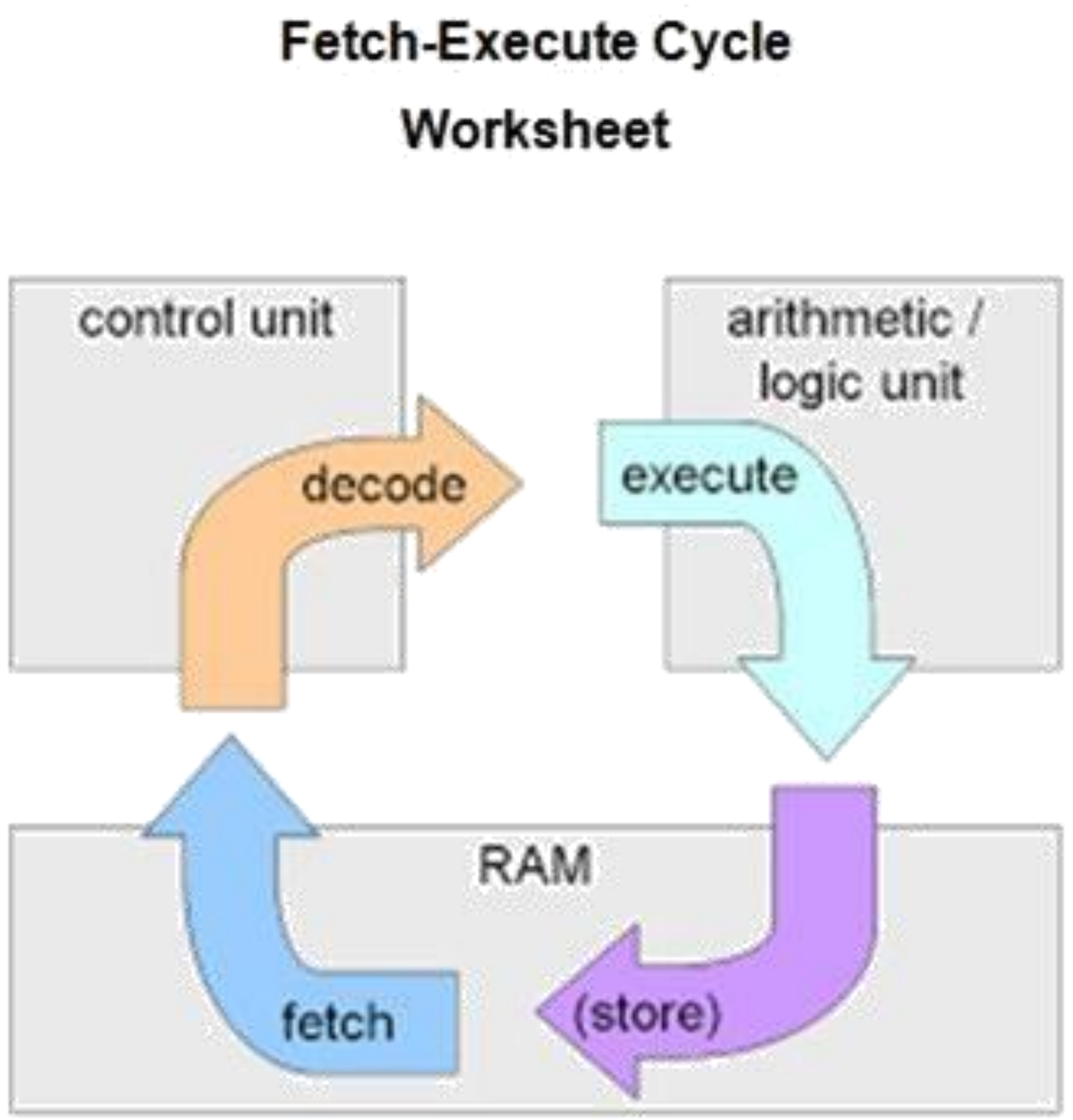
I/O Devices or Peripherals

- I/O Devices are connected to computer through I/O circuits.
- I/O circuits have several Registers called I/O ports.
- Like Memory locations, I/O ports (registers) have addresses and are connected to bus system.
- **Data transfer modes b/w I/O port and I/O device:**
  - **Serial Port:** 1 bit a time e.g. Slow devices i.e. keyboard
  - **Parallel port:** 8 or 16 bits at a time e.g. disk drives
- All the components of microcomputer are hosted by Mother board or system board

Instruction Execution

Fetch-Execute Cycle

- **Fetch**
  - Fetch an instruction from memory
  - Decode the instruction to determine the operation
  - Fetch data from memory if necessary
- **Execute**
  - Perform the operation on the data
  - Store the result in memory if needed



Programming Languages

Low to High Level Languages

1) **Machine Language**
   - Bit strings giving machine specific instructions
   - CPU only understands and executes machine language instructions
2) **Assembly Language**
   - English-like abbreviations or symbolic names to represent operations, registers, and memory
   - A program called **Assembler** translates each assembly language instructions into machine language **(1-to-1 mapping)**
3) **High-Level Language**
   - Codes similar to natural language – English e.g. C, C++, Java, Python etc.
   - Translation of complex mathematical expressions and natural language commands into machine language instructions via **Compilers (1-to many mapping)**

Programming Languages

Machine Code

10100001 00000000 00000000 ; Fetch memory word 0 and place it in register AX
00000101 00000100 00000000 ; Add 4 to AX
10100011 00000000 00000000 ; Store AX's contents in memory word 0

Assembly Code

MOV AX, A ; Fetch contents of location A and place it in register AX
ADD AX, 4 ; Add 4 to AX
MOV A, AX ; Store AX's contents into location A

C/C++ (High level lang.) Code

Int A;
A = A+4;

Pros of Assembly Language

- Efficiency
- Easy access to specific memory or I/O ports read/write
- Provides better understanding of how the computer "thinks" and works.

Number Systems

Numbering System		
System	Base	Digits
Binary	2	0, 1
Octal	8	0,1,2,3,4,5,6,7
Decimal	10	0,1,2,3,4,5,6,7,8,9
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Decimal Base-10	Binary Base-2	Octal Base-8	Hexa Decimal Base-16
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Conversion b/w Number Systems

Decimal to Binary or Hexadecimal System

25d = 2*12 + **1**

12d = 2*6 + **0**

6d = 2*3 + **0**

3d = 2*1 + **1**

1d = 2*0 + **1**

25d = 11001b

35d = 16*2 + 3

2d = 16*0 + 2

35d = 23h

Conversion b/w Number Systems

Binary or Hexadecimal to Decimal system

1 0 1 0 1 1b = $1*2^0 + 1*2^1 + 0*2^2 + 1*2^3 + 0*2^4 + 1*2^5$

= 1 + 2 + 0 + 8 + 0 + 32

= **43d**

1 2 3h = $3*16^0 + 2*16^1 + 1*16^2$

= 3 + 32 + 256

= 291d

Conversion b/w Number Systems

Binary <-> Hexadecimal system

1 0 1 1 | 1 1 0 0b =>

1 1 0 0b = 12d = 0Ch

1 0 1 1b = 11d = 0Bh

10111100b = 0BCh

- Direct translation of nibbles (4 bits) to hex digits and vice versa.
- More convenient to use hex digits

Integer Representations in Computer

Unsigned Integer

- Positive numbers, represent magnitude

0 <= 8 bit <= 255

0 <= 16bit <= 65535

0 <= N bit <= 2^N - 1

127

128	64	32	16	8	4	2	1
1	1	1	1	1	1	1	1
MSB				Byte Memory			LSB

Signed Integer

- Positive (MSB = 0) or Negative (MSB = 1)
- Negative integers are stored in computer in 2's complement form

00000110 -> 6d
11111001
+1
11111010 -> -6d

00000001 -> 1d
11111110
+1
11111111 -> -1d

-128 <= 8 bit <= 127

-32768 <= 16bit <= 32767

-(2^{N-1}) <= N bit <= 2^{N-1} - 1

Character Representations

ASCII Codes (American Standard Code for Information Interchange)

- Most popular encoding scheme
- Only 7 bits are used for ASCII codes -> 128 ASCII Codes
- 0-31 and 127 are for communication control purposes
- 32-126 -> 95 ASCII codes are printable
- E.g. A = 41h , Z = 59h, a = 61h, Space = 20h etc.

Intel 8086 Family of Microprocessors

TABLE 1–6 The Intel family of microprocessor bus and memory sizes.

Microprocessor	Data Bus Width	Address Bus Width	Memory Size
8086	16	20	1M
8088	8	20	1M
80186	16	20	1M
80188	8	20	1M
80286	16	24	16M
80386SX	16	24	16M
80386DX	32	32	4G
80386EX	16	26	64M
80486	32	32	4G
Pentium	64	32	4G
Pentium Pro–Pentium 4	64	32	4G
Pentium Pro–Pentium 4 (if extended addressing is enabled)	64	36	64G

Memory Addressing Modes

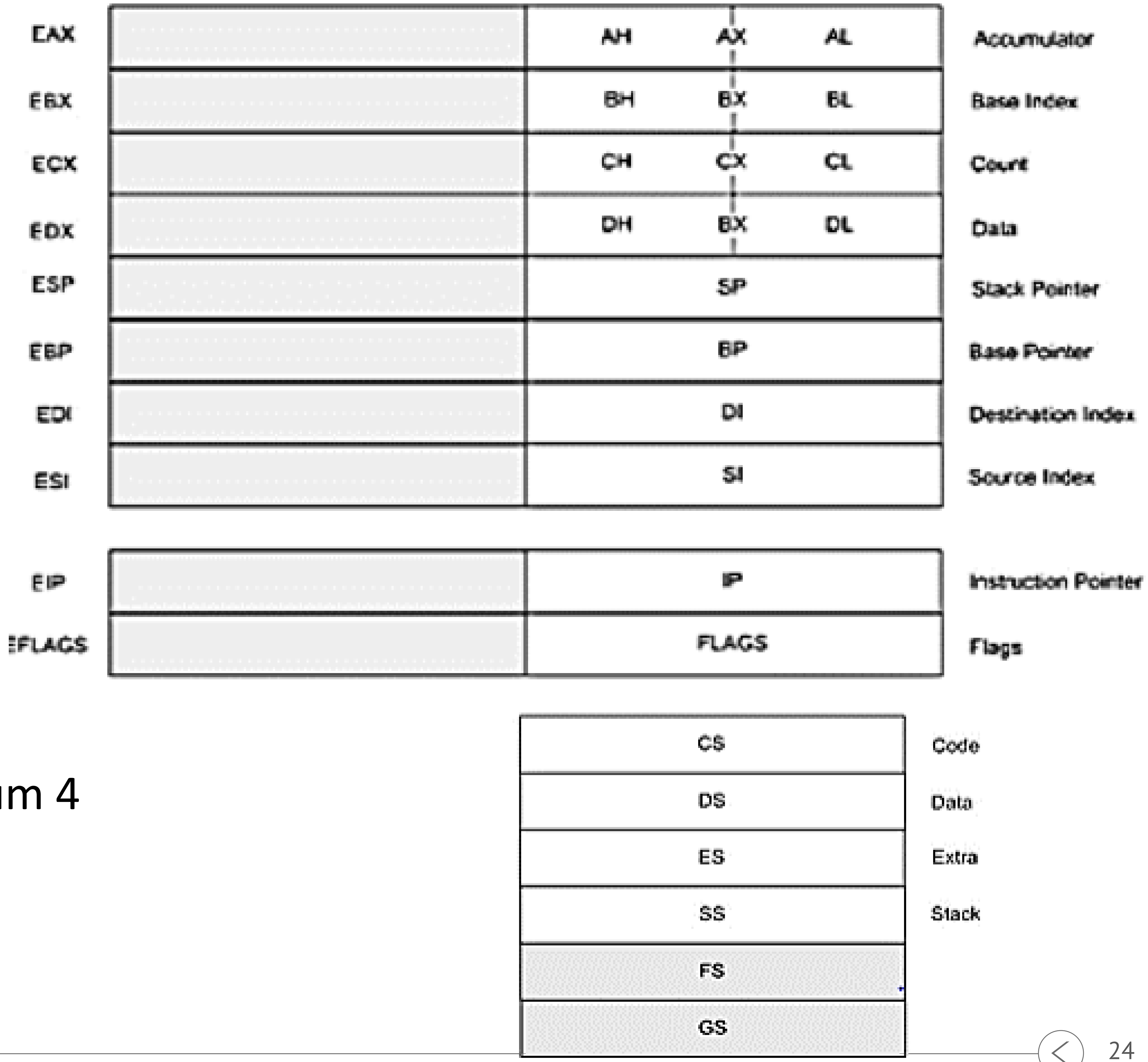
Real Addressing Mode

- Real mode memory operation: 1MB accessible only even by Pentium 4
- This 1MB of real memory access also termed as conventional memory or DOS memory system.

Protected Virtual Address Mode

- More memory is accessible, Bigger programs can run.
- Multitasking through memory protection among different programs.
- 80286 onwards supports both modes.

The programming model: Intel 8086 to Pentium 4



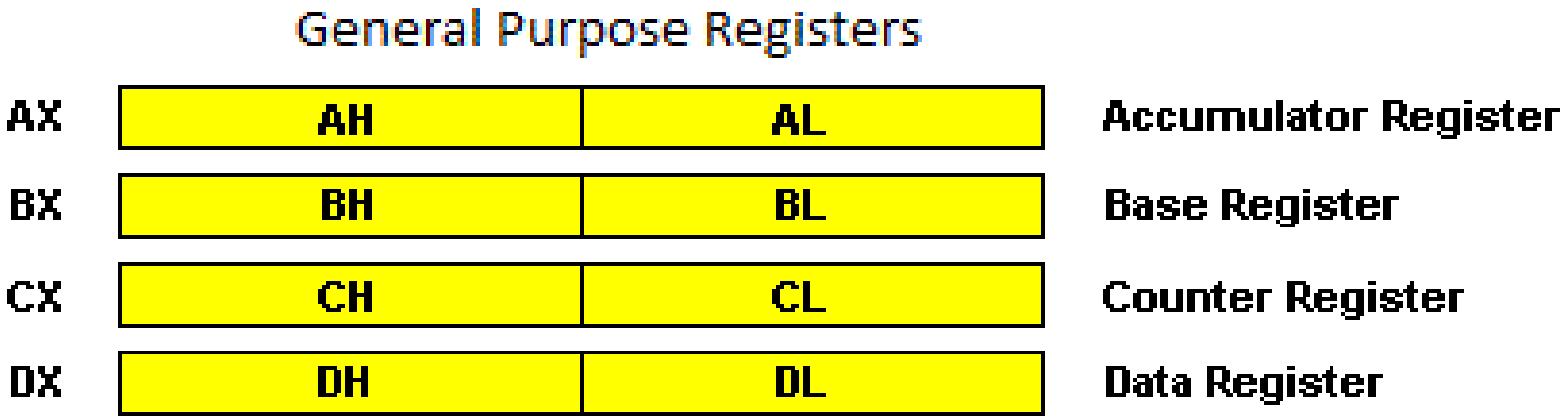
- Shaded Registers exist only from 80386 to Pentium 4
- FS & GS registers have no special name

Data Registers

- Used for arithmetic, logic, and data transfer instructions
- AX: Mul and Div instructions
- BX: Xlat (Translate)

CX: Rep instruction – loop counter

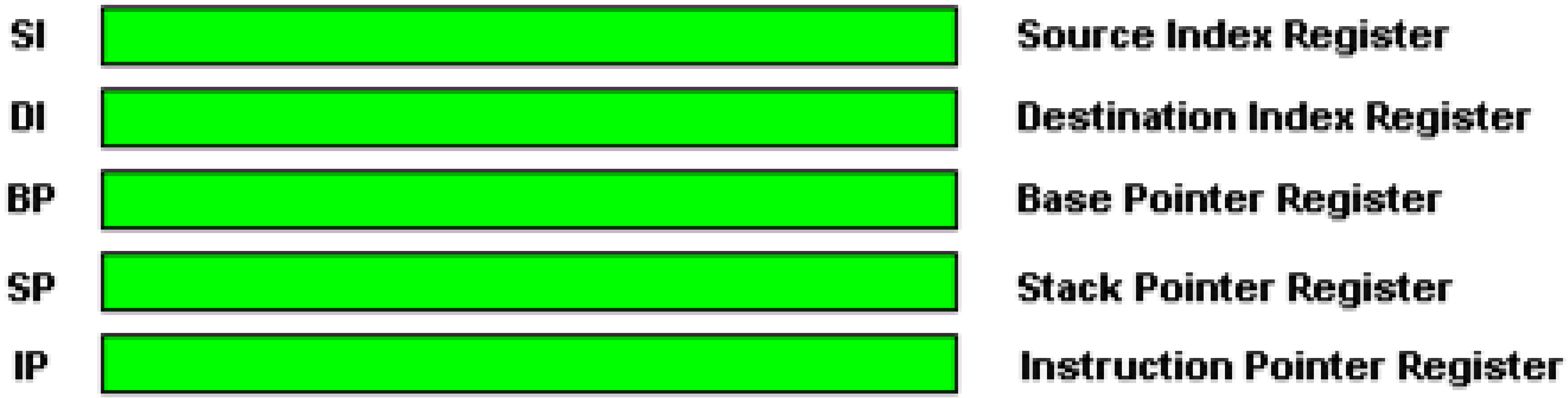
DX: Mul, Div and I/O instructions



Segment Registers

- Used for storing addresses of instructions and data
- 8086 address bus: 20 bit Whereas Registers are of 16 bits. **PROBLEM!!**
- **Segment : Offset** Address
- **Physical Address = Segment Address x 10 h + offset**

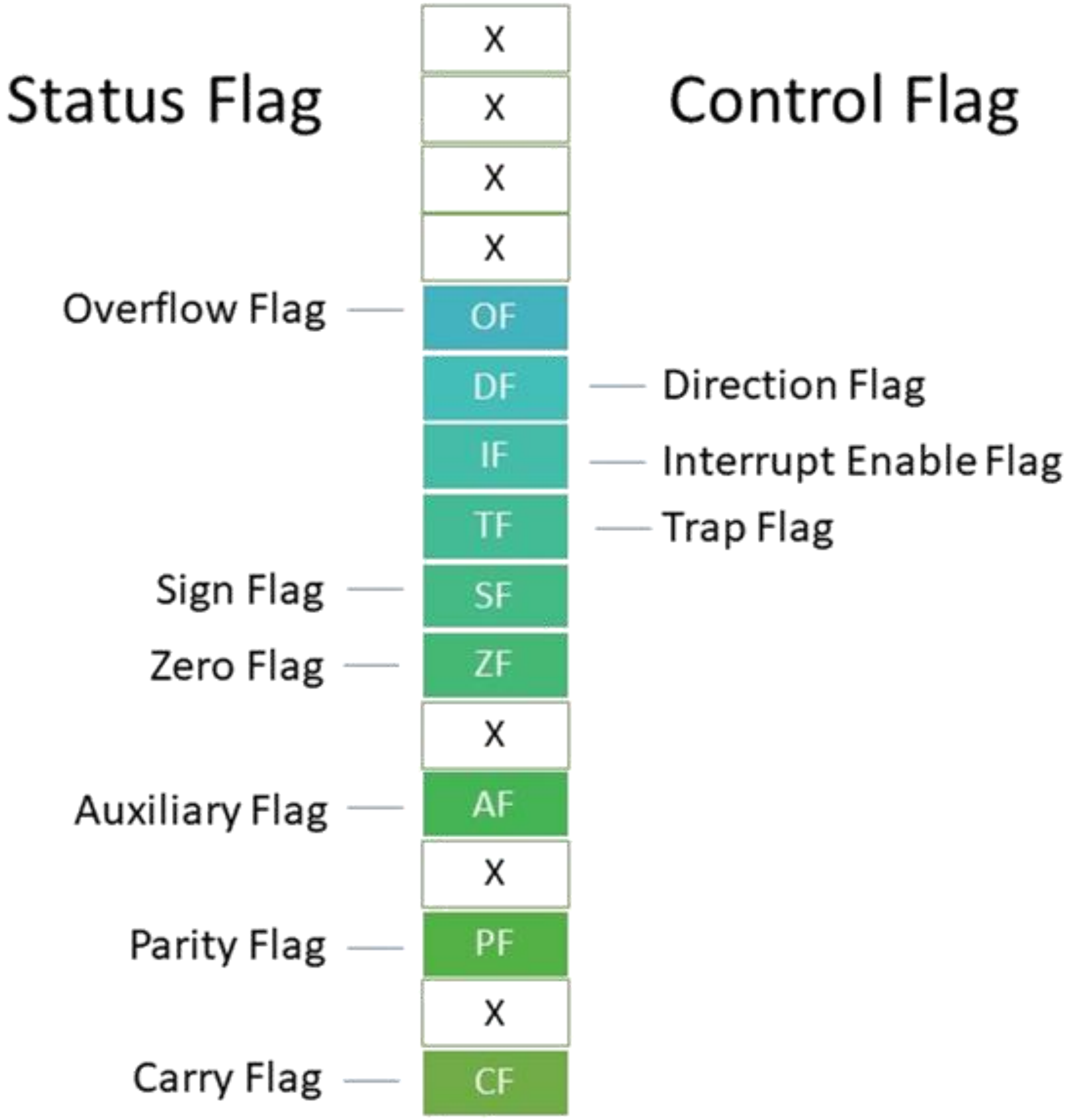
Pointer and Index Registers



Segment	Offset	Special Purpose
CS	IP	Instruction address
SS	SP or BP	Stack address
DS	BX, DI, SI	Data address
ES	DI for string instr.	String destination address

FLAG Register of 8086

- Status Flags
- Control Flags



Assembly Language Syntax

Name	operation	operand(s)	comment

Assembly program consists of statements that are either,

- Instruction (assembler translates to machine code)
- Assembler Directives (instructions for assembler e.g. allocating memory)

Name Field

- Used for instruction labels, procedure names and variable names
- Up to 31 characters long, consists of letters, digits and special characters (? . @ _ $ %)
- Examples:

@sum $1000   .test   done?   X_y   total value   7even   X.   Y&m

Assembly Language Syntax

Name	**operation**	**operand(s)**	**comment**

Operation Field

- **Instructions** -> symbolic operation code (Opcode) -> translated to machine opcode
  - Examples: MOV, ADD, SUB, NEG etc.
- **Assembler Directives** -> pseudo operation code (pseudo-op) -> not translated to machine code
  - Examples: PROC to create a procedure

Assembly Language Syntax

Name\Label	**operation**	**operand(s)**	**;comment**

Operand Field

- **Instructions** -> specifies the data for the operation
  - Examples: NOP (no operand), INC AX (one operand), ADD AX , DX (two- Destination, Src)

- **Assembler Directives** -> contains more information about the directive

EXAMPLE

```
TITLE Add and Subtract              (AddSub.asm)

; This program adds and subtracts 32-bit integers.

INCLUDE Irvine32.inc
.code
main PROC

    mov eax,10000h                 ; EAX = 10000h
    add eax,40000h                 ; EAX = 50000h
    sub eax,20000h                 ; EAX = 30000h
    call DumpRegs                  ; display registers

    exit
main ENDP
END main
```

Microprocessor & Microcontroller based Design, Fall 2022

Variables or Defining Data

Data Defining Pseudo-ops

- Each variable has a data type and a memory address assigned to it
- Pseudo-ops can be used to generate one or more data items

Pseudo-op	Stands for	Size
DB	Define byte	8 bits
DW	Define word	2 bytes
DD	Define double word	4 bytes
DQ	Define quad word	8 bytes
DT	Define ten bytes	10 bytes

Variables or Defining Data

Byte Variables

- Format: **name DB Initial Value** Range: -128~127 **OR** 0~255
- Examples: Alpha DB 4 Beta DB ?

Word Variables

- Format: **name DW Initial Value** Range: -32768~32767 **OR** 0~65535
- Examples: Alpha DW 1234H Beta DW ?

	Offset	**Contents**
Alpha	0000h	34
Alpha + 1	0001h	12

Variables or Defining Data

Arrays

- Sequence of memory bytes or words
- Examples:
  - B_Array   DB 10H,20H,30H
  - W_Array DW 1000,50, 4568, 30
  - Letters DB   ‘ABC’
  - LETTERS DB 41h, 42h, 43h
  - MSG DB ‘HELLO’, 0AH, 0DH, ‘$’

Byte Offset

Offset	Contents
0000h	1000d
0002h	50d
0004h	4568d
0006h	30d

Named Constants

EQU (Equates)

- To assign name to a constant
- No memory is allocated for EQU names
- Examples:
    - LF EQU 0AH
    - CR EQU 0DH
    - PROMT EQU 'TYPE YOUR NAME'

Data Transfer Instructions

MOV, XCHG

- MOV Destination, Source
- E.g. MOV AX, BX
- XCHG AX, Word1

Legal Combinations of operands for MOV

Source operand	Destination Operand: General register	Segment register	Memory location	Constant
General register	yes	yes	yes	no
Segment register	yes	no	yes	no
Memory location	yes	yes	no	no
Constant	yes	no	yes	no

- Illegal: MOV WORD1, WORD2
- Legal: MOV AX, WORD2
  MOV WORD1, AX
- Illegal: MOV DS, CS
- Legal: MOV AX, CS
  MOV DS, AX

➢ **Both operands can't be memory locations or segment registers**

Arithmetic Instructions

ADD, SUB

- ADD destination, Source — ADD Word1, AX
- SUB destination, Source — SUB AX,DX ; AX = AX-DX

➢**Both operands can't be memory locations**

INC, DEC

- Increment or decrement by 1 in the contents of memory location or register
- INC Word1 DEC Byte1

NEG

- Negate the contents of destination by taking 2's complement. E.g. NEG BX