College of Electronics Engineering

Systems & Control Engineering Department

Microprocessors I

Lecture 1
(Basic Structure of Computers)

Zeyad T. Shareef

Functional Units

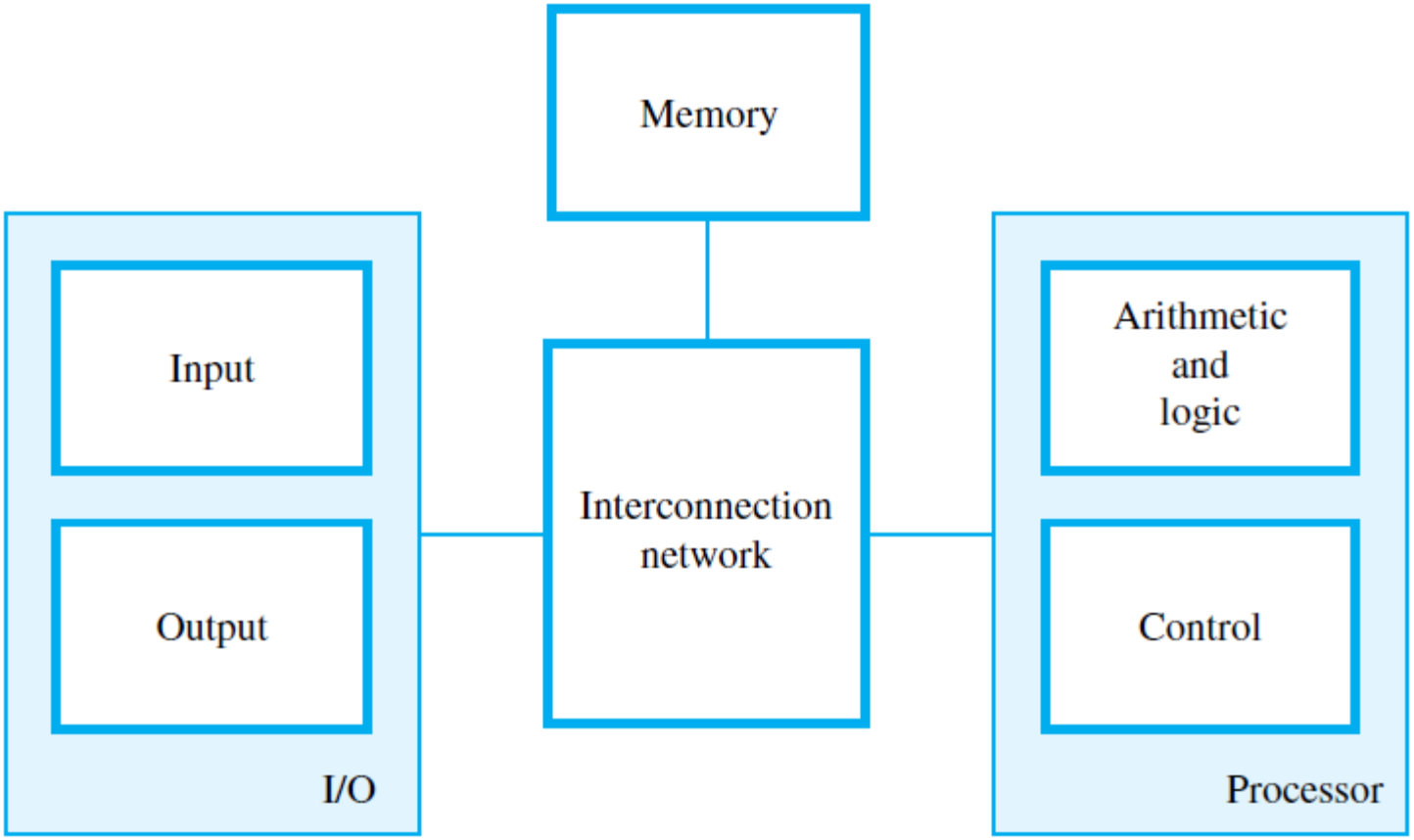


Figure 1.1. Basic functional units of a computer.

Functional Units

- **An interconnection network** provides the means for the functional units to exchange information and coordinate their actions.
- We refer to the **Arithmetic and logic circuits**, in conjunction with the **main control circuits**, as the ***processor.***

Information Handled by a Computer

- It is convenient to categorize the information handled by a computer as either **instructions** or **data**
- ***Instructions*, or *machine instructions***, are explicit commands that:
  - Govern the transfer of information within a computer as well as between the computer and its I/O devices.
  - Specify the arithmetic and logic operations to be performed.
- A ***program*** is a list of instructions which performs a task, programs are stored in the memory
- The instructions and data handled by a computer must be encoded in a suitable format.
- Each instruction, number, or character is encoded as a string of binary digits called *bits*.

Input Unit

- Computers accept coded information through input units. The most common input device is the keyboard.
- Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted to the processor.
- Many other kinds of input devices for human-computer interaction are available, including the touchpad, mouse, joystick, and trackball.

Memory Unit

- Memory units store programs and data.
- There are two classes of storage called primary and secondary storage, that will be discussed later.

Arithmetic and Logic Unit (ALU)

- Most computer operations are executed in the *arithmetic and logic unit (ALU) of the* processor.

- Any arithmetic or logic operation, such as addition, subtraction, multiplication, division, or comparison of numbers, is initiated by bringing the required operands into the processor, where the operation is performed by the ALU.

Output Unit

- The output unit is the counterpart of the input unit.
- Its function is to send processed results to the outside world.
- A familiar example of such a device is a *printer*. However, printers are quite slow mechanical devices compared to the electronic speed of a processor.
- Some units, such as graphic displays, provide both an output function, showing text and graphics, and an input function, through touchscreen capability.
- The dual role of such units is the reason for using the single name *input/output (I/O) unit in many cases*.

Control Unit

- The operations of the memory, arithmetic and logic, and I/O units are coordinated and controlled by the control unit.
- Control circuits are responsible for generating the *timing signals* that determine when a given action is to take place.
- In practice, however, much of the control circuitry is physically distributed throughout the computer.

The operation of a computer can be summarized as follows:

- The computer accepts information in the form of programs and data through an input unit and stores it in the memory.
- Information stored in the memory is fetched under program control into an arithmetic and logic unit, where it is processed.
- Processed information leaves the computer through an output unit.
- All activities in the computer are directed by the control unit.

College of Electronics Engineering

Systems & Control Engineering Department

Microprocessors I

Lecture 2
(Introduction to Microprocessors)

Zeyad T. Shareef

Introduction

- A **computer** is a programmable machine that receives input, stores and manipulates data//information, and provides output in a useful format.

DIAGRAM OF A COMPUTER SYSTEM

A **computer** is a programmable machine that receives input, stores and manipulates data//information, and provides output in a useful format.

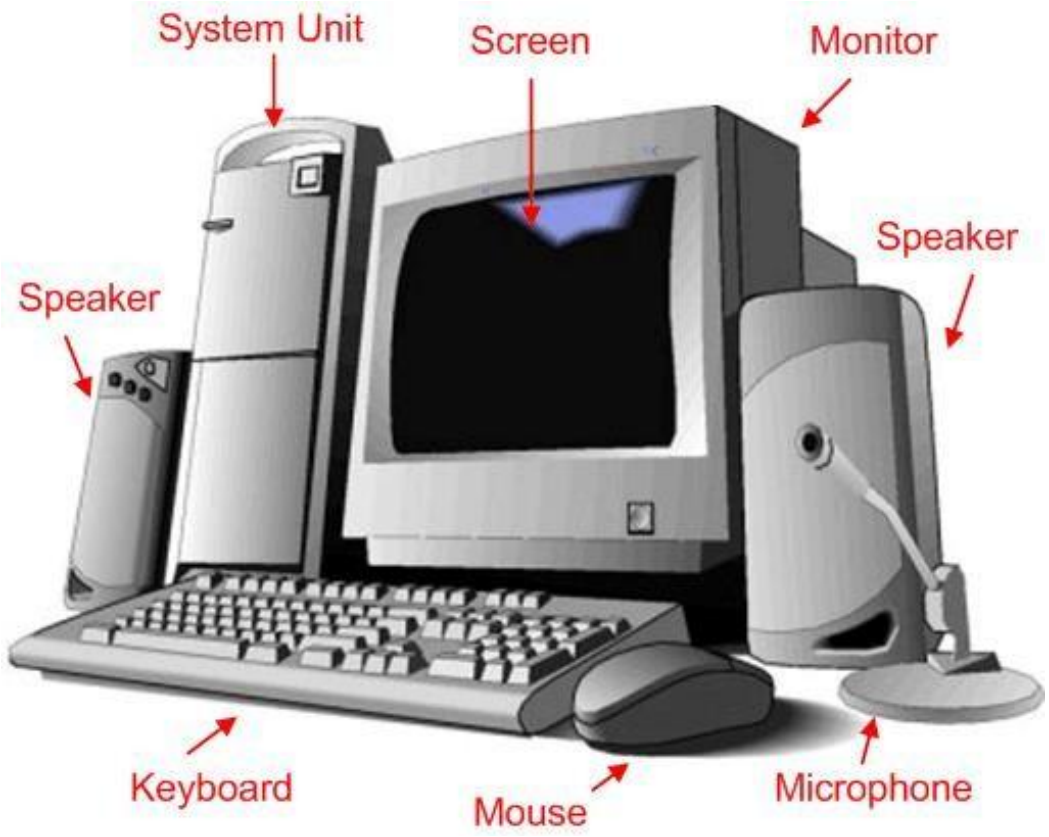
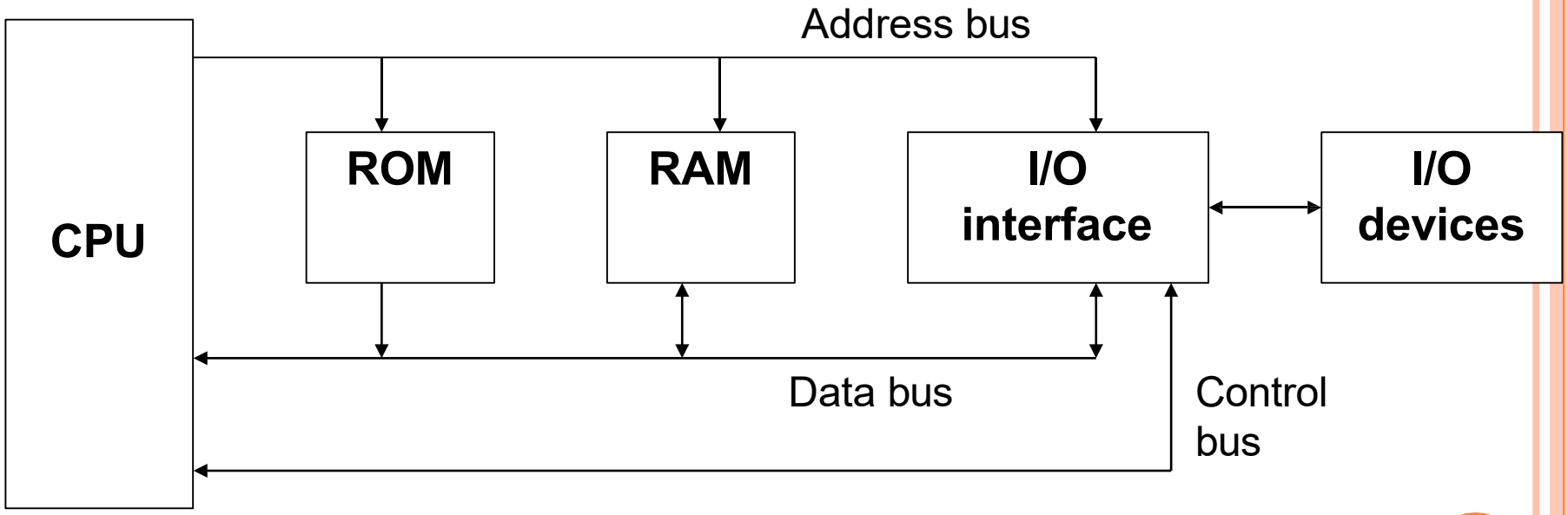


Diagram Of A Computer System

BLOCK DIAGRAM OF A BASIC COMPUTER SYSTEM

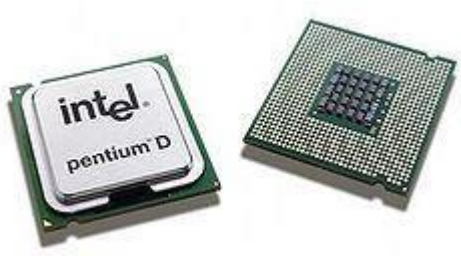
Basic computer system consist of a Central processing unit (CPU), memory (RAM and ROM), input/output (I/O) unit.



Block diagram of a basic computer system

BASIC COMPONENT OF MICROCOMPUTER

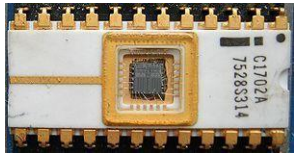
1. CPU - Central Processing Unit
   - the portion of a computer system that carries out the instructions of a computer program
   - the primary element carrying out the computer's functions. It is the unit that reads and executes program instructions.
   - The data in the instruction tells the processor what to do.



Pentium D dual core processors

2. Memory

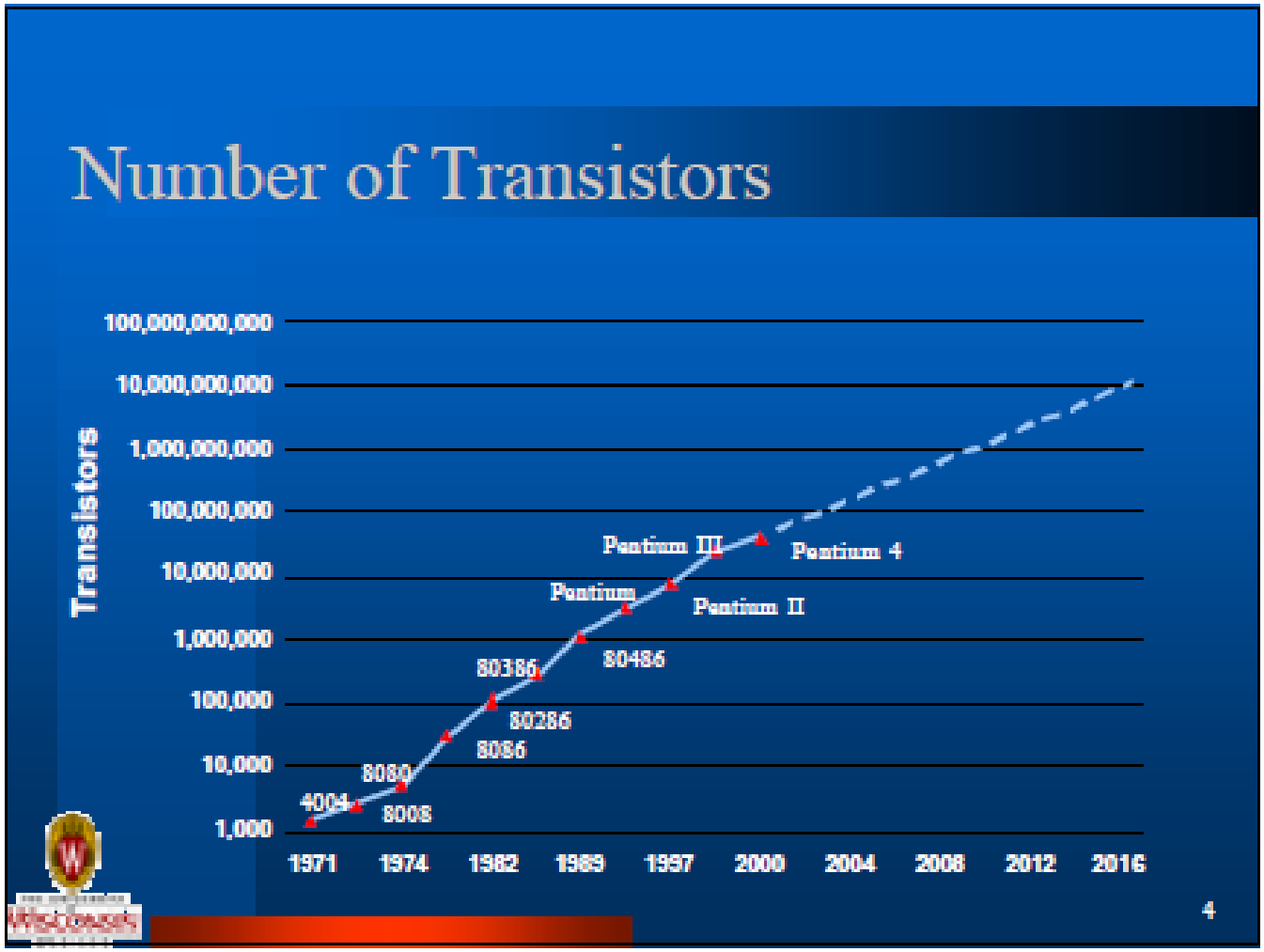
- physical devices used to store data or programs (sequences of instructions) on a temporary or permanent basis for use in an electronic digital computer.
- Computer main memory comes in two principal varieties: random-access memory (RAM) and read-only memory (ROM).
- RAM can be read and written to anytime the CPU commands it, but ROM is pre-loaded with data and software that never changes, so the CPU can only read from it.
- ROM is typically used to store the computer's initial start-up instructions.
- In general, the contents of RAM are erased when the power to the computer is turned off, but ROM retains its data indefinitely.
- In a PC, the ROM contains a specialized program called the BIOS that orchestrates loading the computer's operating system from the hard disk drive into RAM whenever the computer is turned on or reset.



3. I/O Unit

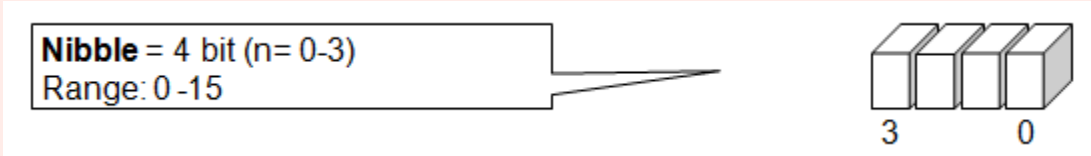
- **Input/output (I/O)**, refers to the communication between an information processing system (such as a computer), and the outside world possibly a human, or another information processing system.
- Inputs are the signals or data received by the system, and outputs are the signals or data sent from it
- Devices that provide input or output to the computer are called peripherals
- On a typical personal computer, peripherals include input devices like the keyboard and mouse, and output devices such as the display and printer. Hard disk drives, floppy disk drives and optical disc drives serve as both input and output devices. Computer networking is another form of I/O.

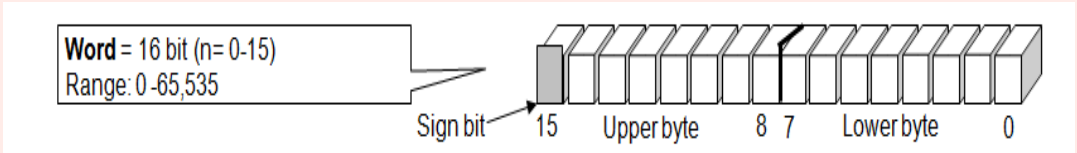
Evolution of Microprocessor

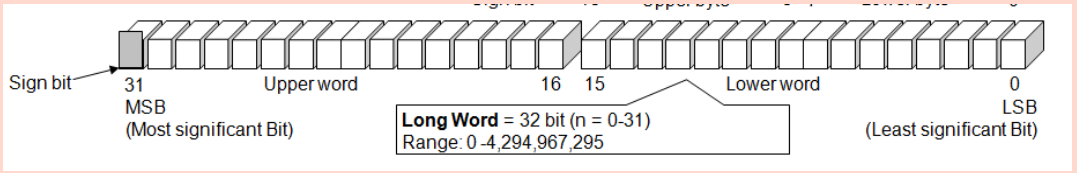


DATA SIZE

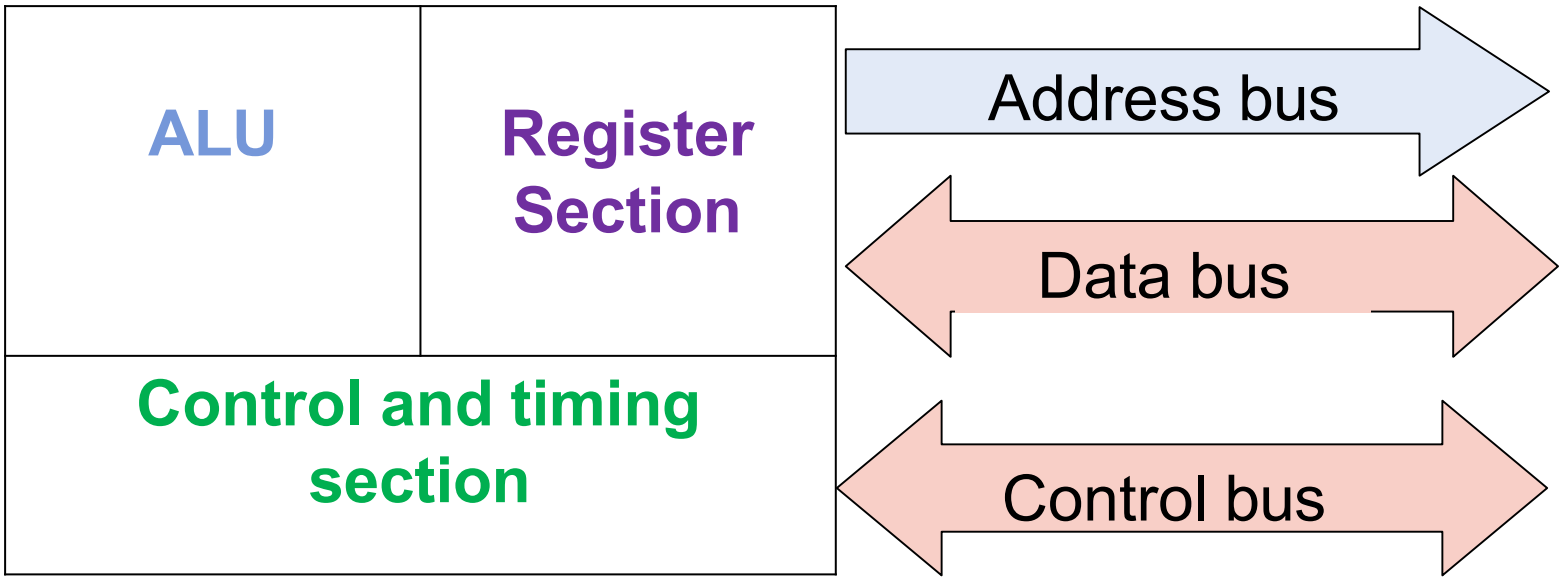
Nibble	**4 bit**
Byte	**8 bit**
Word	**16 bit**
Long word	**32 bit**







Internal structure and basic operation of microprocessor

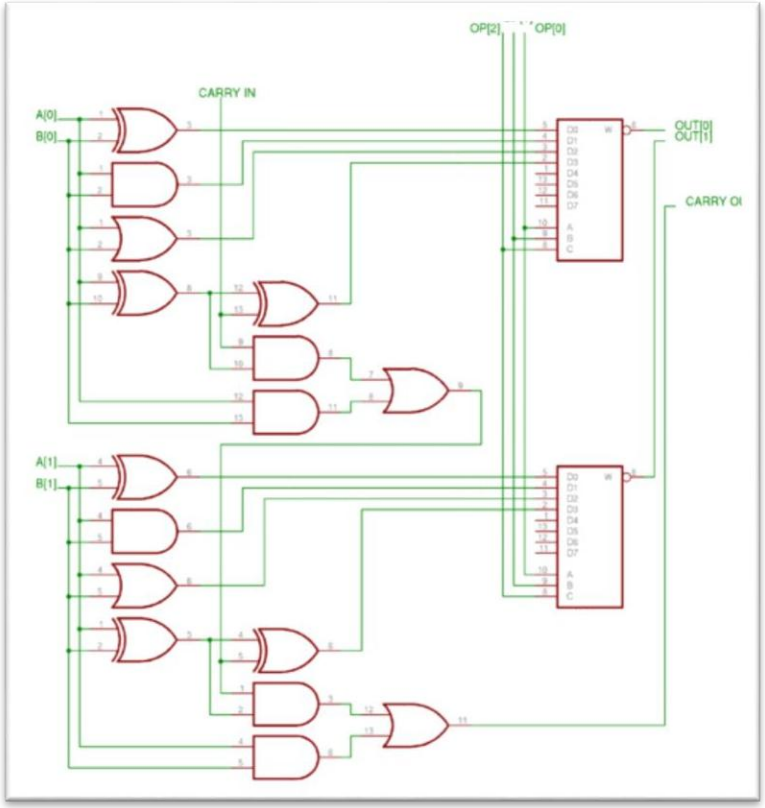


Block diagram of a microprocessor

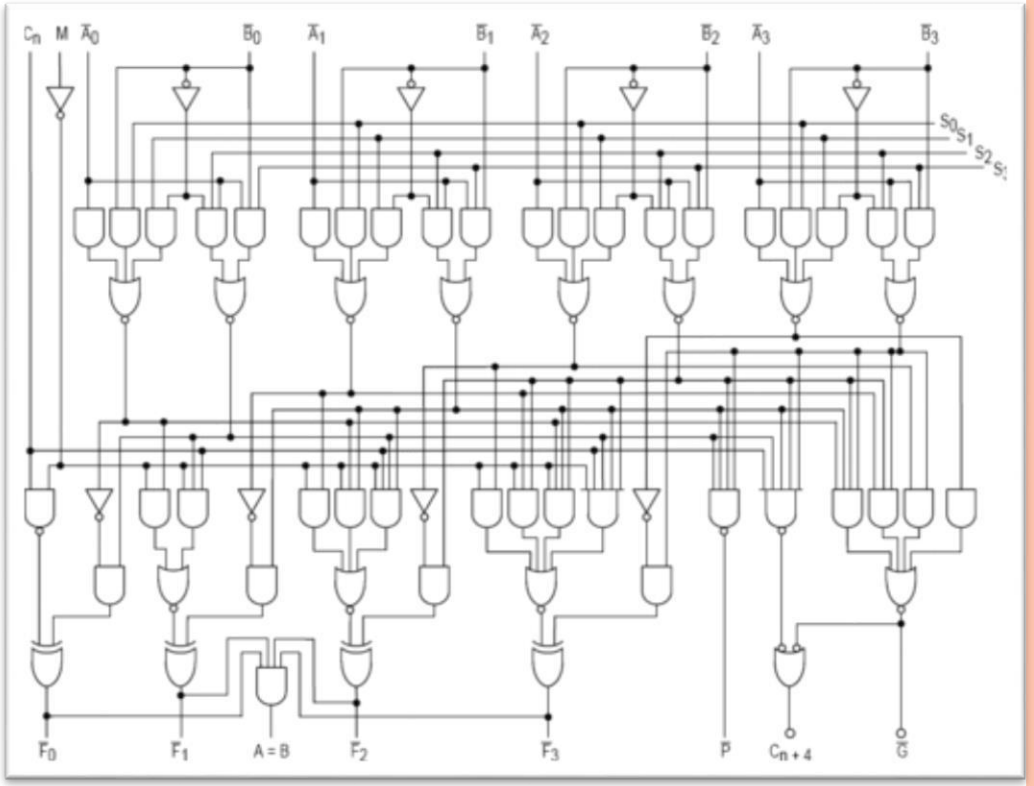
Arithmetic and logic unit (ALU)

- The component that performs the arithmetic and logical operations
- the most important components in a microprocessor, and is typically the part of the processor that is designed first.
- able to perform the basic logical operations (AND, OR), including the addition operation.
- The inclusion of inverters on the inputs enables the same ALU hardware to perform the subtraction operation (adding an inverted operand), and the operations NAND and NOR.

Internal structure of ALU



2 bits of ALU



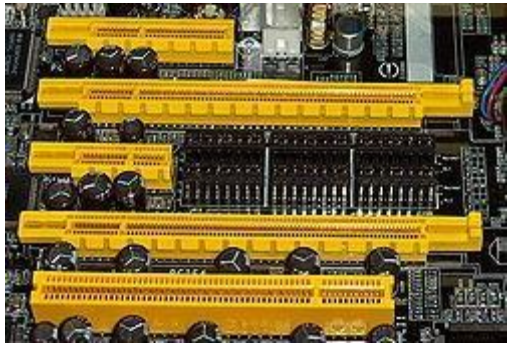
4 bits of ALU

Control unit

- The circuitry that controls the flow of information through the processor, and coordinates the activities of the other units within it.
- In a way, it is the "brain within the brain", as it controls what happens inside the processor, which in turn controls the rest of the PC.
- On a regular processor, the control unit performs the tasks of fetching, decoding, managing execution and then storing results.

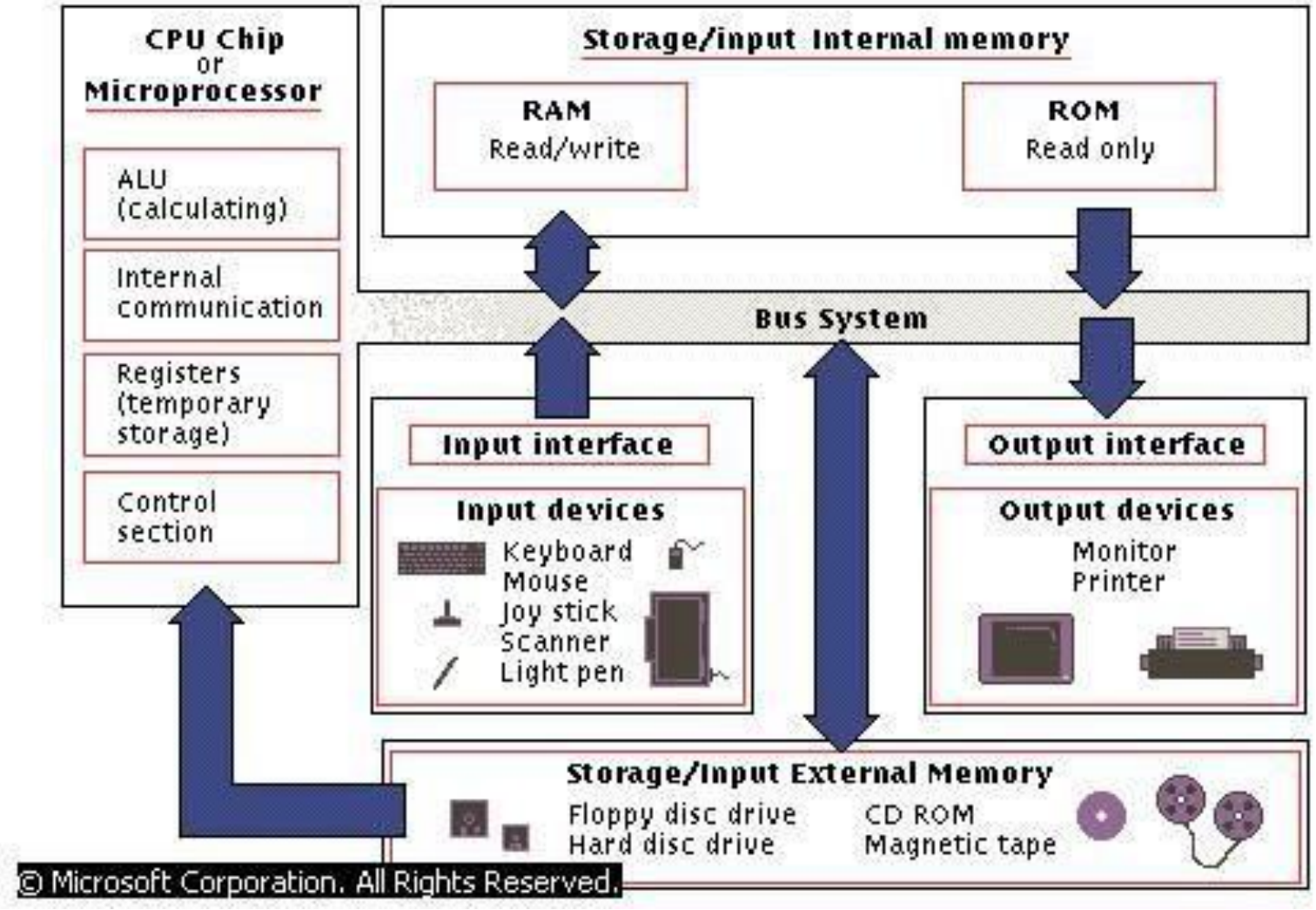
Bus system

- a subsystem that transfers data between computer components inside a computer or between computers.



4 PCI Express bus card slots (from top to bottom: x4, x16, x1 and x16), compared to a traditional 32-bit PCI bus card slot (very bottom).

Bus system connection



Data bus

- The data bus is 'bi-directional'
  - data or instruction codes from memory or input/output.are transferred into the microprocessor
  - the result of an operation or computation is sent out from the microprocessor to the memory or input/output.
- Depending on the particular microprocessor, the data bus can handle 8 bit or 16 bit data.

Address bus

- The address bus is '**unidirectional**', over which the microprocessor sends an address code to the memory or input/output.
- The size (width) of the address bus is specified by the number of bits it can handle.
- The more bits there are in the address bus, the more memory locations a microprocessor can access.
- A 16 bit address bus is capable of addressing 65,536 (64K) addresses.

Control bus

- The control bus is used by the microprocessor to send out or receive timing and control signals in order to coordinate and regulate its operation and to communicate with other devices, i.e. memory or input/output.

Examples of micro processor

- Intel 8085
- Intel 8086

College of Electronics Engineering

Systems & Control Engineering Department

Microprocessors I

Lecture 3
(Integer Numbers Representation & Arithmetic Operations)

Zeyad T. Shareef

Integer Numbers Representation

For signed integers, the leftmost bit is used to indicate the sign:

0 for positive

1 for negative

There are three ways to represent signed integers:

- Sign and magnitude
- 1's complement
- 2's complement

B	Values represented		
$b_3 b_2 b_1 b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

2’s-complement integers

2’s-complement representation is used in current computers

Consider a four-bit signed integer example,

where the value +5 is represented as:

0 1 0 1

To form the value -5, complement all bits of

0 1 0 1 to obtain 1 0 1 0

and then add 1 to obtain

1 0 1 1

Addition

Two examples of adding four-bit numbers:

0 0 0 1	+1	0 1 0 0	+4
+ 0 1 0_1 1	+ +5	+ 1 0 1 0	+ -6
---	---	---	---
0 1 1 0	+6	1 1 1 0	-2

The answers fall within the representable range of −8 through +7
No overflow occurs

Subtraction

Form the 2's-complement of the subtrahend and then perform normal addition

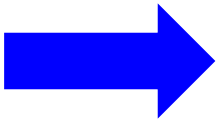
Overflow

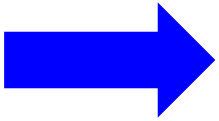
When answers do not fall within the representable range, we say that *arithmetic overflow* has occurred

0 1 1 0 +6	1 1 1 0 -2
+ $0_1$1 0 0 + +4	$_1$1 0 0 1 + -7
---------- ------	---------- ------
1 0 1 0 +10	0 1 1 1 -9

Sign extension

Replicate the sign bit to extend
4-bit signed integers to 8-bit signed integers

0 1 0 1  0 0 0 0 0 1 0 1

1 1 1 0  1 1 1 1 1 1 1 0

College of Electronics Engineering

Systems & Control Engineering Department

Microprocessors I

Lecture 4
(Internal Microprocessor Architecture)

Zeyad T. Shareef

INTRODUCTION

- Before a program is written or instruction investigated, internal configuration of the microprocessor must be known.
- In a multiple core microprocessor, each core contains the same programming model.

Processor components

In addition to the ALU and the control circuitry, the processor contains a number of registers used for several different purposes; generally the registers in the processor can be categorized into:

- **User-visible registers:**

  Registers are used during programming and specified by the instructions. These registers typically are 16, 32 and 64, each of which hold one word of operand data.

Processor components

- **Control and status registers:**

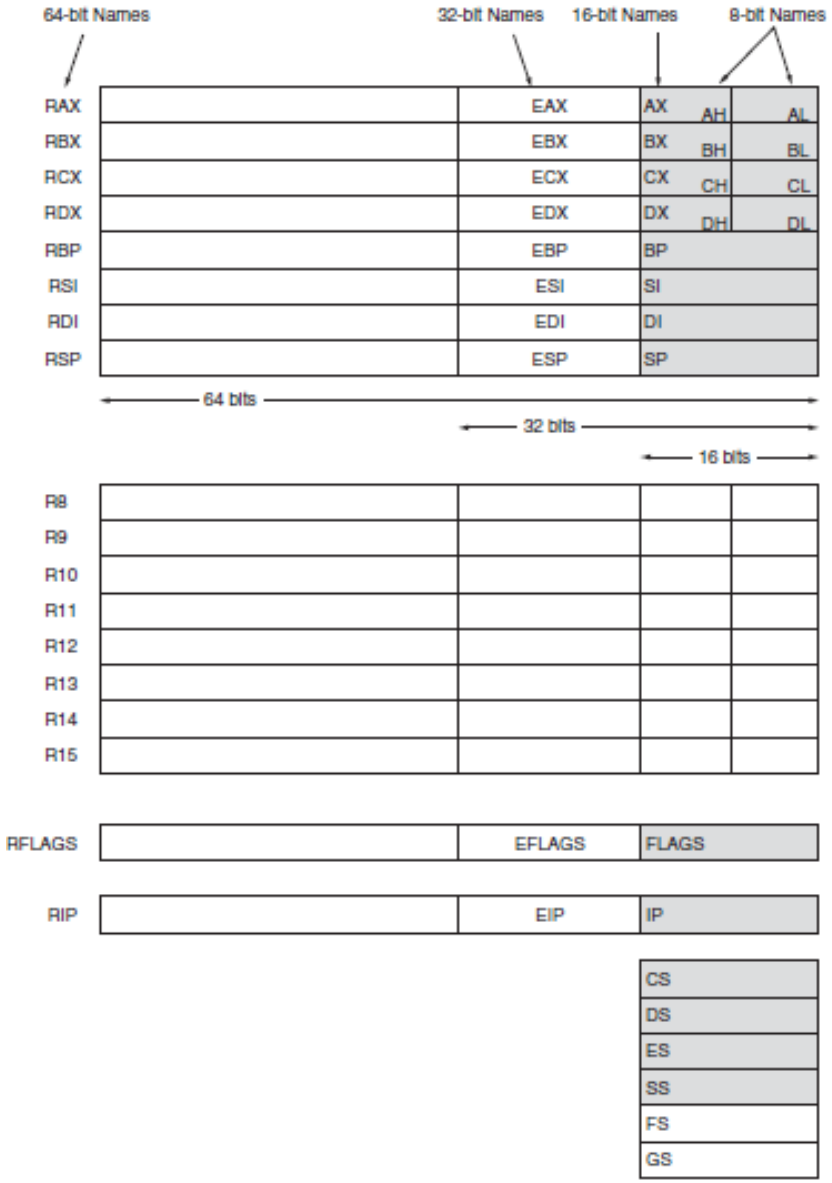
  There are a variety of processor registers that are employed to control the operation of the processor. Most of these, on most machines, are not visible to the user. As example of these registers:

  - The program counter (PC) register holds the memory address of the next instruction to be executed.
  - The instruction register (IR) holds the current instruction that is currently being executed.

The Programming Model

Figure 3.1 illustrates The programming model of the 16, 32 and 64 bits microprocessor architecture.

Figure 3.1 The programming model of the 8086 through the Core2 microprocessor including the 64-bit extensions.



Multipurpose Registers

- **RAX -** a 64-bit register (RAX), a 32-bit register **(accumulator)** (EAX), a 16-bit register (AX), or as either of two 8-bit registers (AH and AL).

- The accumulator is used for instructions such as multiplication, division, and some of the adjustment instructions.

- **RBX,** addressable as RBX, EBX, BX, BH, BL.
  - BX register **(base index)** sometimes holds offset address of a location in the memory system in all versions of the microprocessor (memory addressing).
- **RCX**, as RCX, ECX, CX, CH, or CL.
  - a **(count)** general-purpose register that also holds the count for various instructions, In the 386 or better it can also address the data memory.
- **RDX,** as RDX, EDX, DX, DH, or DL.
  - a **(data)** general-purpose register
  - holds a part of the result from a multiplication or part of dividend before a division, In the 386 or better the EDX register can also address the data memory.

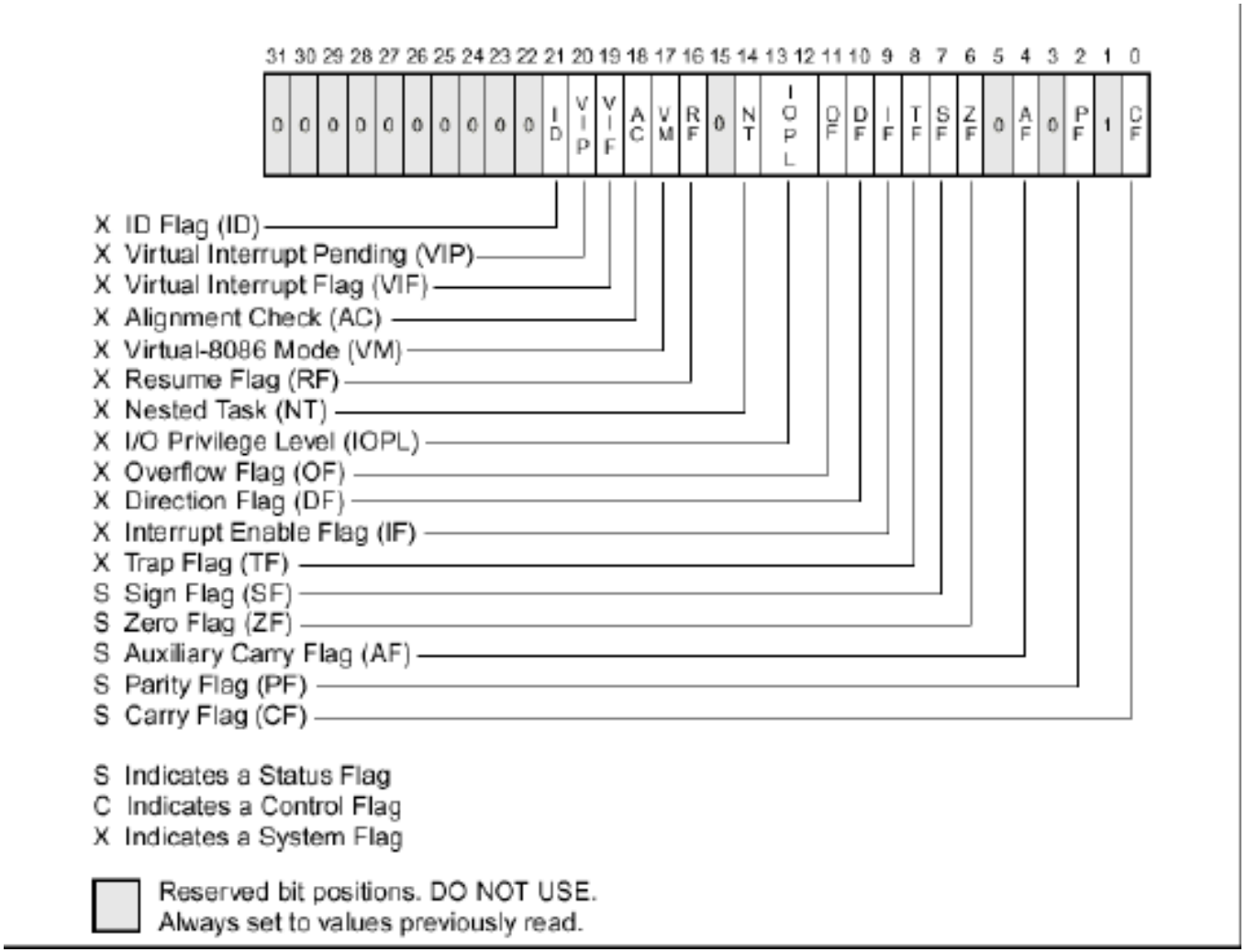
- **RBP,** as RBP, EBP, or BP.
  - points to a memory **(base pointer)** location for memory data transfers.
- **RDI** addressable as RDI, EDI, or DI.
  - often addresses **(destination index)** string destination data for the string instructions.
- **RSI** used as RSI, ESI, or SI.
  - the **(source index)** register addresses source string data for the string instructions.
  - like RDI, RSI also functions as a general-purpose register.
- **R8 - R15** found in the Pentium 4 and Core2 if 64-bit extensions are enabled.
  - data are addressed as 64-, 32-, 16-, or 8-bit sizes and considered as general purpose registers.

Special-Purpose Registers

- Include RIP, RSP, and RFLAGS.
- Segment registers include CS, DS, ES, SS, FS, and GS.
- **RIP** addresses the next instruction in a section of memory.
  - defined as **(instruction pointer)** a code segment.
- **RSP** addresses the memory area that is used for the stack.

  The stack is that memory zone of the last-in first-out type where the return addresses of the subroutines and the content of the registers (POP and PUSH instructions) are saved.

- **EFLAGS or RFLAGS:** this register keeps the state of the microprocessor, in addition to controlling its operations.
- The figure below shows its structure:



- **C (carry)** It stores the carry in the operations of addition or the borrow in the operations of subtraction. It is also used to show error conditions in some procedures.

- **I (interrupt)** It controls the operations of the input pin INTR. If I=1 the pin is enabled to generate interrupts, if it is 0 the interrupts are blocked.

- **Z (zero)** It indicates if the result of an operation is zero. It is 1 if the result is 0, it is 0 in the other cases.

- **S (sign)** It indicates the sign of the result in a mathematical or logic operation. It is 1 if the sign is negative, while it is 0 if the sign is positive.

Segment Registers

- Four or six segment registers in various versions of the microprocessor.

- Following is a list of each segment register, along with its function in the system.

- **CS (code)** It individuates a memory section where there is the code that is relevant to programs and procedures that the microprocessor must execute.

- **DS (data)** It individuates a memory section where there are the data that are used by the program.

- **ES (extra)** an additional data segment used by some instructions to hold destination data.

- **SS (stack)** defines the area of memory used for the stack.
  - stack entry point is determined by the stack segment and stack pointer registers.

- **FS & GS**: are supplemental segment registers available in 80386–Core2 microprocessors.
  - allow two additional memory segments to be accessed by programs.

College of Electronics Engineering

Systems & Control Engineering Department

Microprocessors I

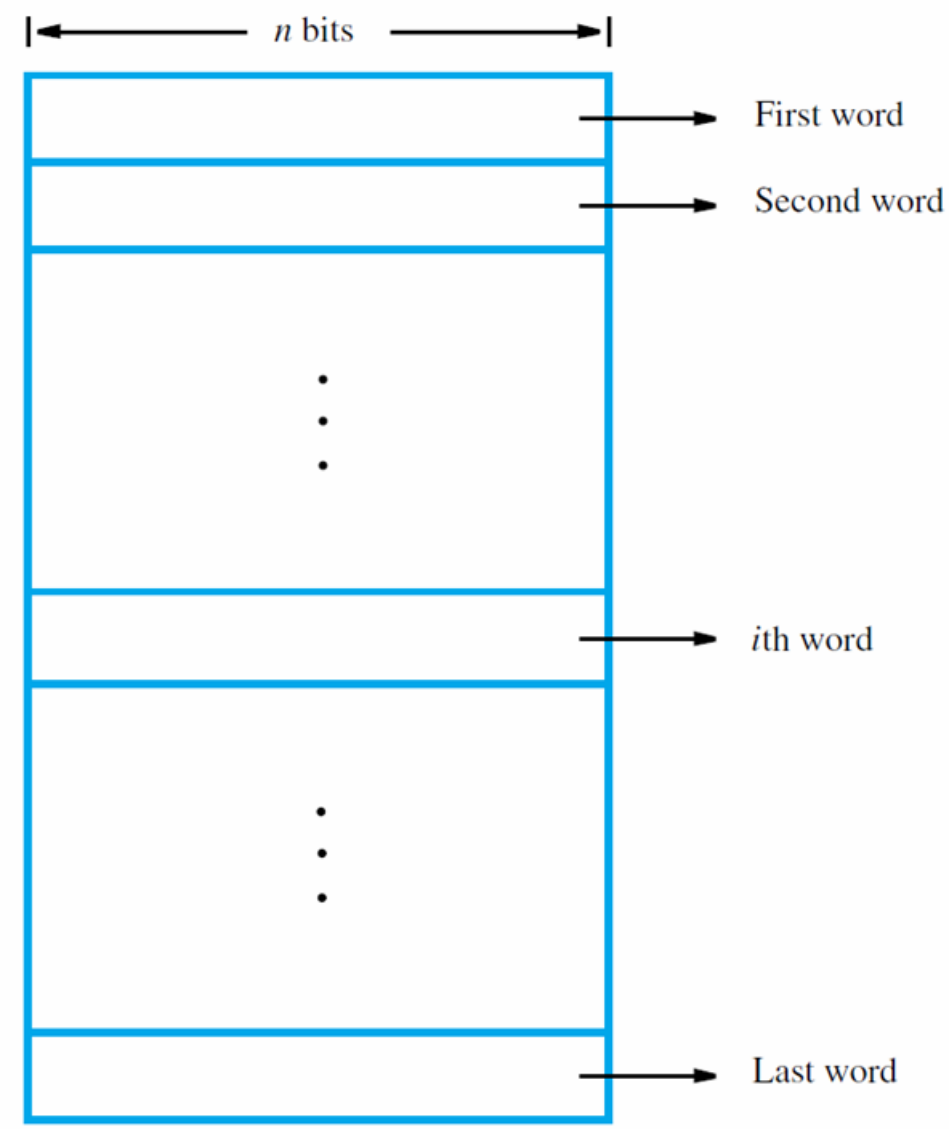
Lecture 5
(Memory Locations and Addressing)

Zeyad T. Shareef

Memory Organization

- Memory consists of many millions of cells
- Each cell holds a bit of information, 0 or 1
- Information is usually handled in larger units
- A word is a group of *n* bits
- Word length can be 16 to 64 bits
- Memory is a collection of consecutive words of the size specified by the word length

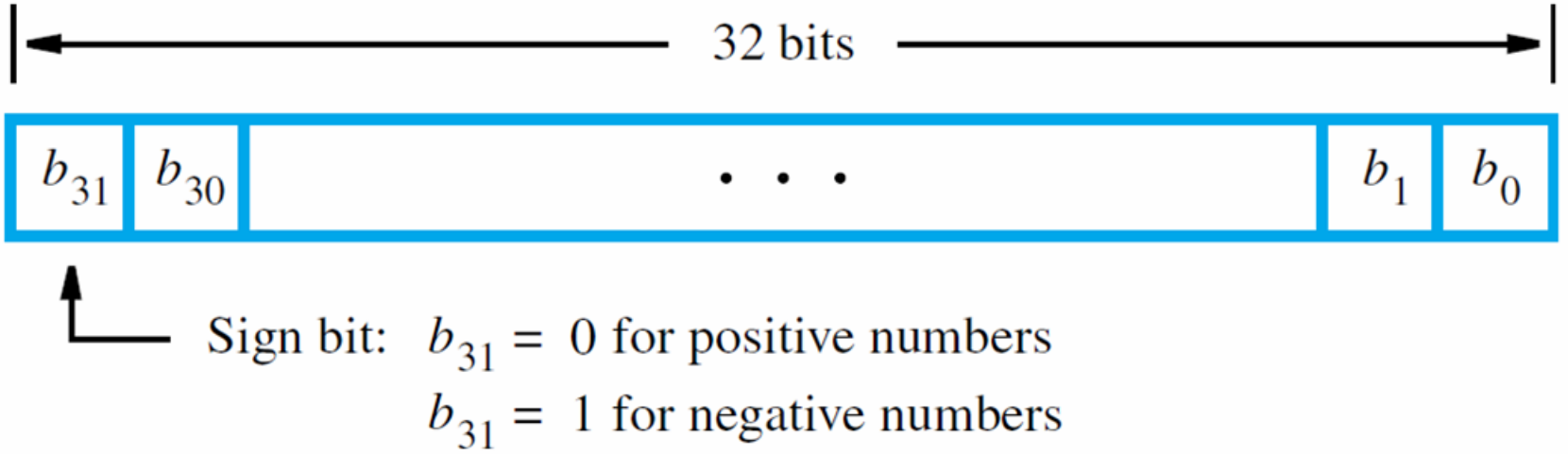
Figure 5.1 Memory words



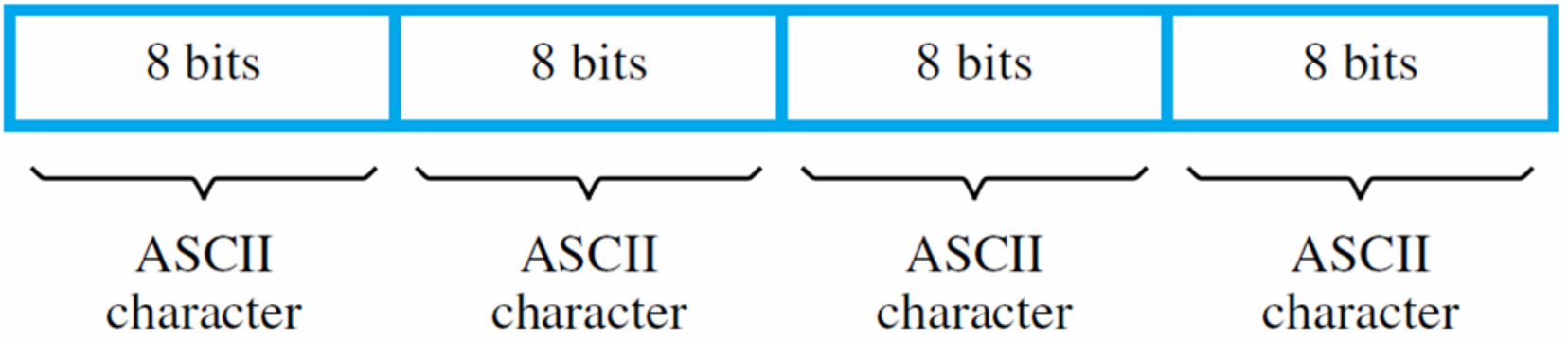
Word and Byte Encoding

- A common word length is 32 bits
- Such a word can store a 32-bit signed integer or four 8-bit bytes (e.g., ASCII characters)
- For 32-bit integer encoding, bit b_{31} is sign bit
- Words in memory may store data or machine instructions for a program
- Each machine instruction may require one or more consecutive words for encoding

Figure 5.2 Examples of encoded information in a 32-bit word



(a) A signed integer



(b) Four characters

Addresses for Memory Locations

- To store or retrieve items of information, each memory location has a distinct address
- Numbers 0 to $2^k - 1$ are used as addresses for successive locations in the memory
- The 2^k locations constitute the address space
- Memory size set by k (number of address bits)
- Examples: $k = 20 \rightarrow 2^{20}$ or 1M locations,
  $k = 32 \rightarrow 2^{32}$ or 4G locations

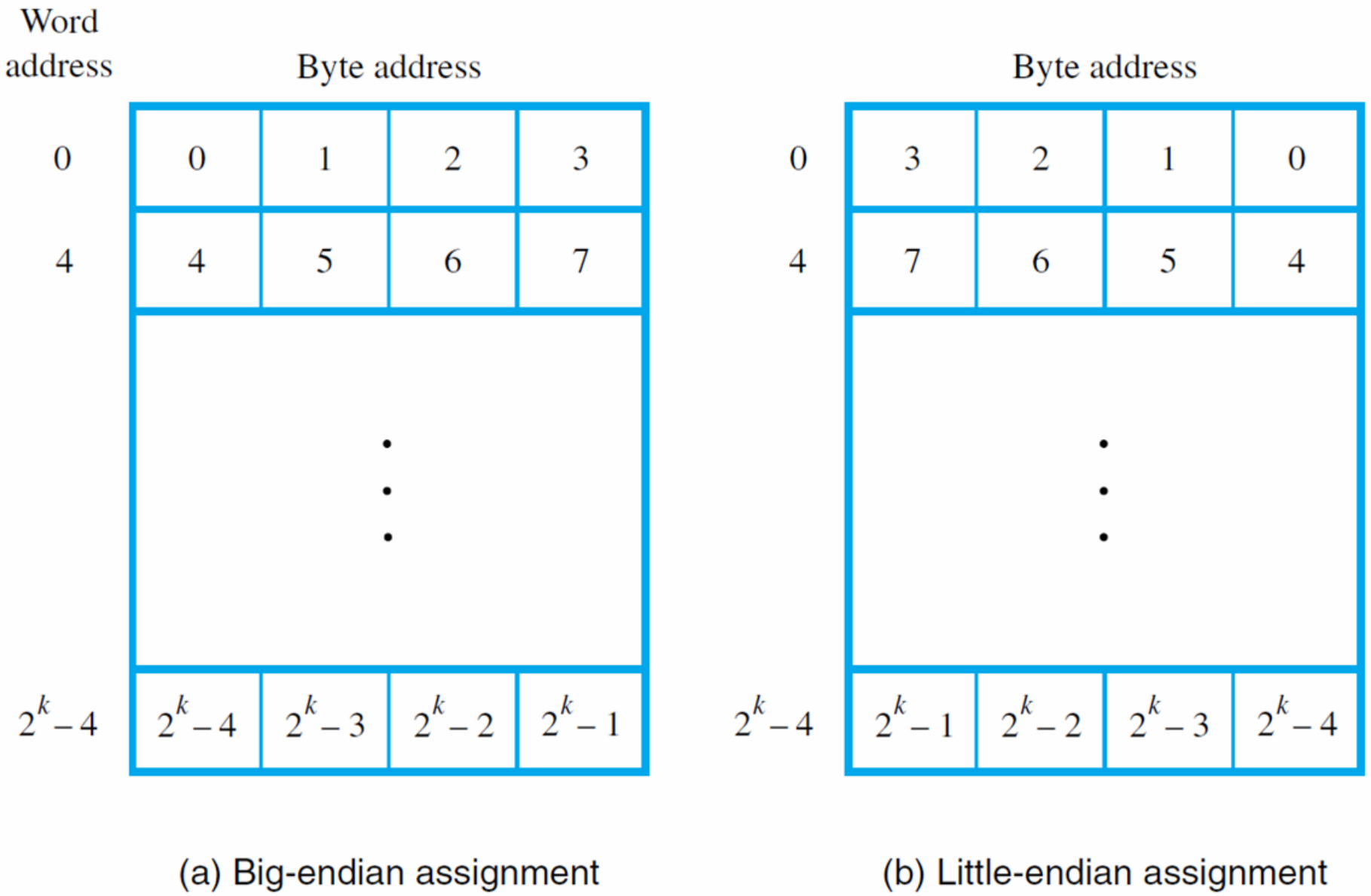
Byte Addressability

- Byte size is always 8 bits
- But word length may range from 16 to 64 bits
- Impractical to assign an address to each bit
- Instead, provide a byte-addressable memory that assigns an address to each byte
- Byte locations have addresses 0, 1, 2, …
- Assuming that the word length is 32 bits, word locations have addresses 0, 4, 8, …

Big- and Little-Endian Addressing

- Two ways to assign byte address across words:
  - Big-endian addressing assigns lower byte addresses to more significant (leftmost) bytes of the word
  - Little-endian addressing assigns lower byte addresses to less significant (rightmost) bytes of the word.
- Commercial computers use either approach, and some can support both approaches.
- Addresses for 32-bit words are still 0, 4, 8, …
- Bits in each byte labeled b_7 … b_0, left to right

Figure 5.3 Byte and word addressing



Word Alignment

- # of bytes per word is normally a power of 2
- Word locations have aligned addresses if they begin at byte addresses that are multiples of the number of bytes in a word
- Examples of aligned addresses:

  2 bytes per word → 0, 2, 4, …

  8 bytes per word → 0, 8, 16, …
- Some computers permit unaligned addresses

Real Mode Memory Addressing

- **Real mode operation** allows addressing of only the first 1M byte of memory space—even in Pentium 4 or Core2 microprocessor.
  - the first 1M byte of memory is called the **real memory**, **conventional memory**, or **DOS memory** system.
  - The DOS operating system requires that the microprocessor operates in the real mode.
  - Windows does not use the real mode.

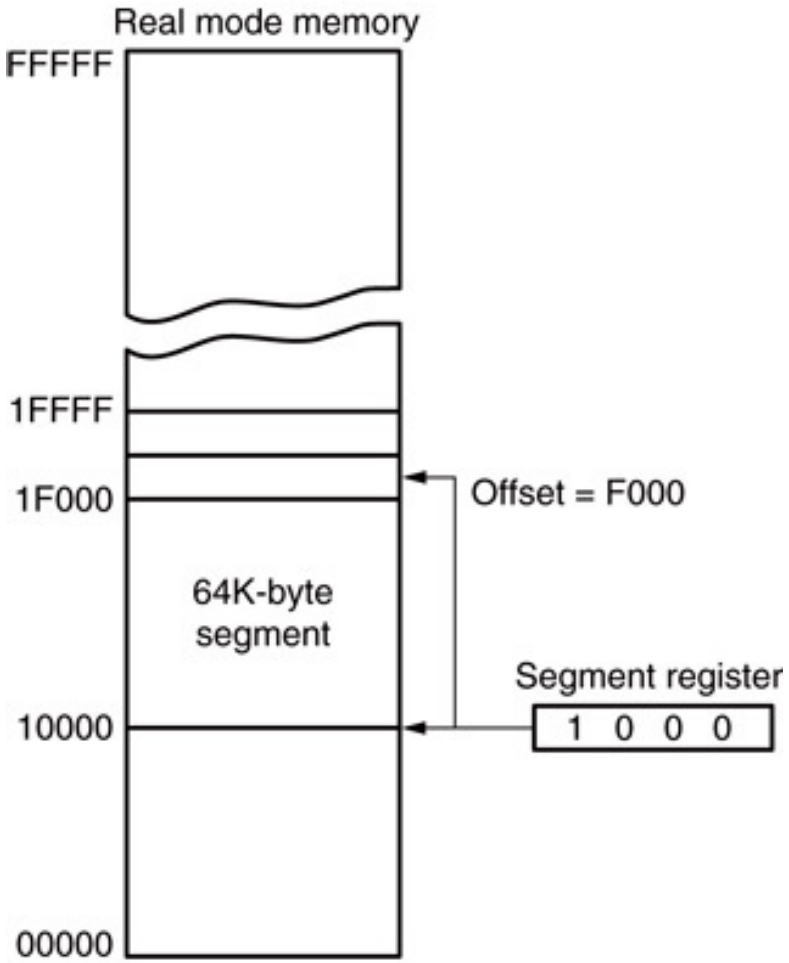
Real Mode Memory Addressing

- Only the 8086 and 8088 operate exclusively in the real mode.
- Note that if the Pentium 4 or Core2 operate in the 64-bit mode, it cannot execute real mode applications.

Segments and Offsets

- Segments in the real mode always have a length of 64K bytes.
- All real mode memory addresses must consist of a segment address plus an offset address.
  - **segment address** defines the beginning address of any 64K-byte memory segment.
  - **offset address** selects any location within the 64K byte memory segment.
- Figure 5.4 shows how the **segment plus offset** addressing scheme selects a memory location.

Figure 5.4 The real mode memory-addressing scheme, using a segment address plus an offset



- this shows a memory segment beginning at **10000H**, ending at location **1FFFFH**
  - 64K bytes in length
- also shows how an offset address, called a displacement, of **F000H** selects location **1F000H** in the memory

- The segment register in Figure 5.4 contains 1000H, yet it addresses a starting segment at location 10000H. In the real mode, each segment register is internally appended with a **0H** on its rightmost end.
- Once the beginning address is known, the **ending address** is found by adding FFFFH.
  - because a real mode segment of memory is 64K in length.
- The offset address is always added to the segment starting address to locate the data.
- Segment and offset address is sometimes written as 1000:2000.
  - a segment address of 1000H; an offset of 2000H

Example

Segment Register	Start Address	End Address
2000H	20000H	2FFFFH
2001H	20010H	3000FH
2100H	21000H	30FFFH

Default Segment and Offset Registers

- The microprocessor has rules that apply to segments whenever memory is addressed.
  - these rules define the segment and offset register combination
- For example, the **code segment** register defines the start of the code segment.
- The **instruction pointer** locates the next instruction within the code segment.

- Another of the default combinations is the **stack**.
  - stack data are referenced through the stack segment at the memory location addressed by either the stack pointer (SP/ESP) or the pointer (BP/EBP).

- The 8086–80286 microprocessors allow four memory segments and the 80386–Core2 microprocessors allow six memory segments.

Table 5.1 Default 16-bit segment and offset combinations

Segment	*Offset*	*Special Purpose*
CS	IP	Instruction address
SS	SP or BP	Stack address
DS	BX, DI, SI, an 8- or 16-bit number	Data address
ES	DI for string instructions	String destination address

Table 5.2 Default 32-bit segment and offset combinations

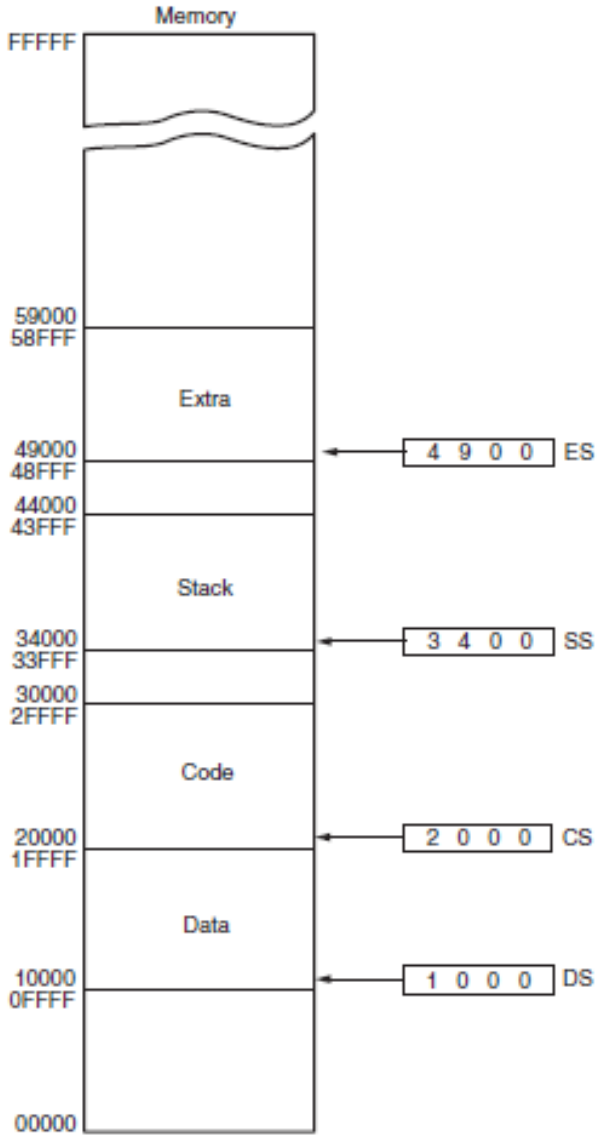
Segment	*Offset*	*Special Purpose*
CS	EIP	Instruction address
SS	ESP or EBP	Stack address
DS	EAX, EBX, ECX, EDX, ESI, EDI, an 8- or 32-bit number	Data address
ES	EDI for string instructions	String destination address
FS	No default	General address
GS	No default	General address

- A memory segment can touch or overlap if 64K bytes of memory are not required for a segment.

- Think of segments as windows that can be moved over any area of memory to access data or code.

- A program can have more than four or six segments but only access four or six segments at a time.
- Figure 5.5 shows a system that contains four memory segments.

Figure 5.5 A memory system showing the placement of four memory segments



College of Electronics Engineering

Systems & Control Engineering Department

Microprocessors I

Lecture 6
Memory Locations and Addressing (cont.)

Zeyad T. Shareef

Protected Mode Memory Addressing

- Protected mode memory addressing (80286 and above) allows access to data and programs located within & above the first 1M byte of memory.
- **Protected mode** is where Windows operates.
- The offset address is still used to access information located within the memory segment.
- In the 80386 and above, the offset address can be a 32-bit number instead of a 16-bit number in the protected mode. A 32-bit offset address allows the microprocessor to access data within a segment that can be up to 4G bytes in length.

Selectors and Descriptors

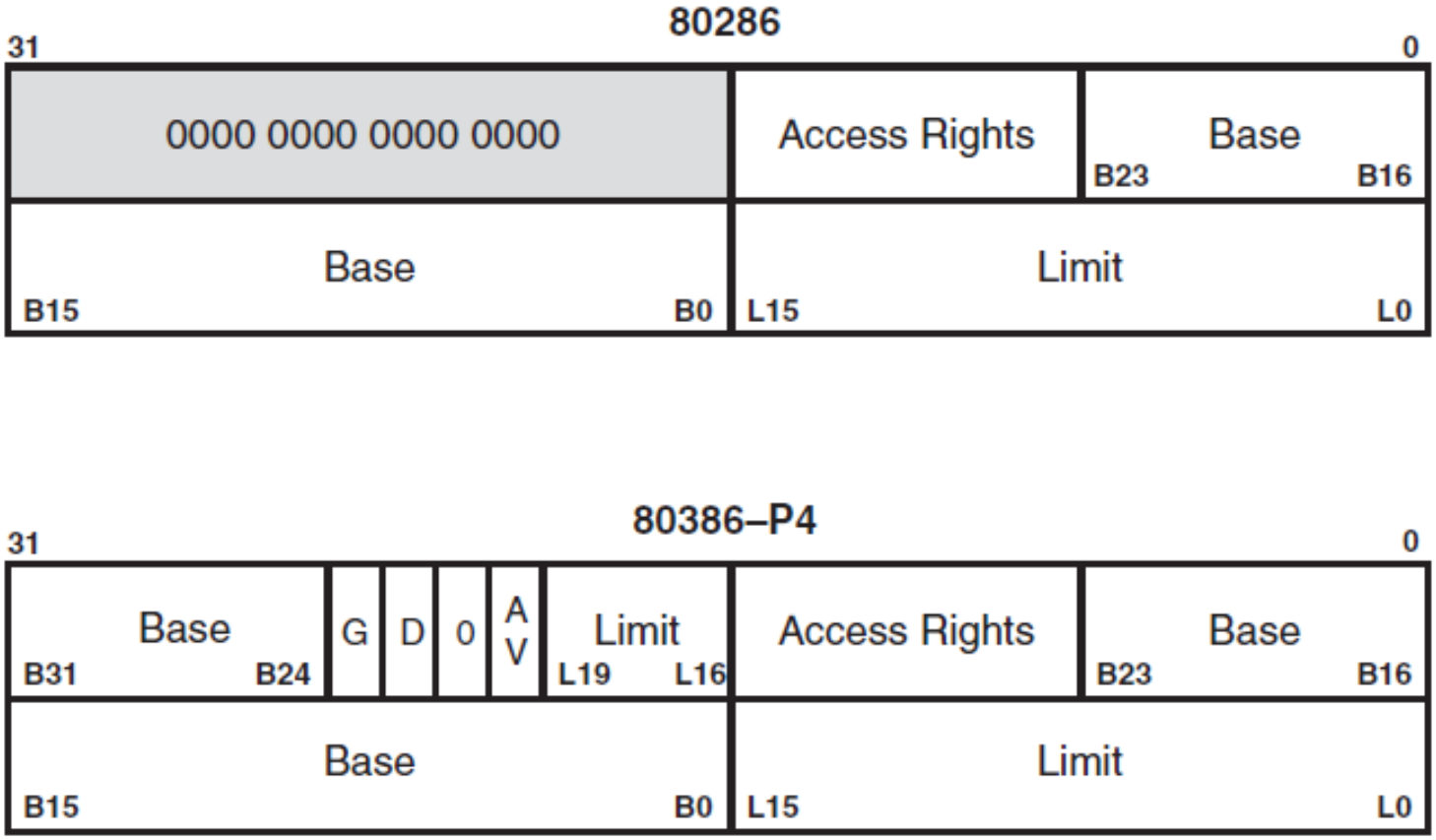
- Indirectly, the register still selects a memory segment, but not directly as in real mode.
- In place of a segment address, the segment register contains a **selector** that selects a **descriptor** from a descriptor table.
- The **descriptor** describes the memory segment's location, length, and access rights.
  - it selects one of 8192 descriptors from one of two tables of descriptors
- There are two descriptor tables used with the segment registers: a global descriptor (**system descriptor)** and a local descriptor (**application descriptor).**

- Each descriptor table contains 8192 descriptors, so a total of 16,384 descriptors are available to an application at any time.

- Because the descriptor describes a memory segment, this allows up to 16,384 memory segments to be described for each application.

- Figure 5.1 shows the format of a descriptor for the 80286 and 80386-P4 as examples. Descriptors for the 80286 and the 80386–Core2 differ slightly.
  - each descriptor is 8 bytes in length

Figure 5.1 The 80286 and 80386 descriptors



- The base address of the descriptor indicates the starting location of the memory segment. The base address is a 24-bit address for the 80286 microprocessor, and a 32-bit address for the 80386 and above.

- The segment limit contains the last offset address found in a segment. For example, if a segment begins at memory location F00000H and ends at location F000FFH, the base address is F00000H and the limit is FFH.

  For the 80286 microprocessor, the base address is F00000H and the limit is 00FFH. For the 80386 and above, the base address is 00F00000H and the limit is 000FFH.

- Notice that the 80286 has a 16-bit limit and the 80386 through the Pentium 4 have a 20-bit limit.

- The **access rights byte** controls access to the protected mode segment.

- If the segment grows beyond its limit, the operating system is interrupted, indicating a general protection fault.

- Note that in 64-bit mode there is only a code segment and no other segment descriptor types.

- There is another feature found in the 80386 through the Pentium 4 descriptor that is not found in the 80286 descriptor: the **G bit**, or **granularity bit**.

- If G=0, the limit specifies a segment limit of 00000H to FFFFFH.

- If G=1, the value of the limit is multiplied by 4K bytes (appended with FFFH). The limit is then 00000FFFH to FFFFFFFFH

EXAMPLE 1

```
Base = Start = 10000000H
G = 0
End = Base + Limit = 10000000H + 001FFH = 100001FFH
```

Example 1 shows the segment start and end if the base address is 10000000H, the limit is 001FFH, and the G bit = 0.

EXAMPLE 2

```
Base = Start = 10000000H
G = 1
End = Base + Limit = 10000000H + 001FFFFFH = 101FFFFFH
```

Example 2 uses the same data as Example 1 , except that the G bit = 1. Notice that the limit is appended with FFFH to determine the ending segment address.

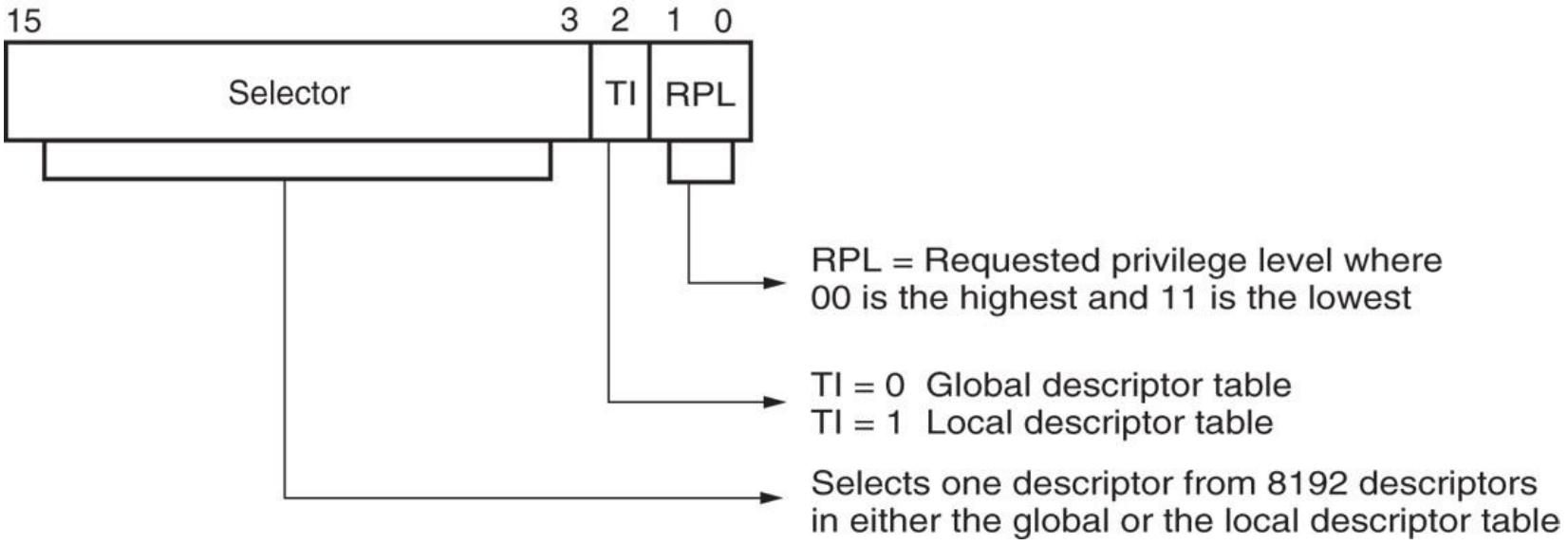
- Descriptors are chosen from the descriptor table by the segment register.
- The segment register contains a 13-bit selector field, a table selector bit, and a requested privilege level field.

  -The 13-bit **selector** chooses one of the 8192 descriptors from the descriptor table as mentioned before.

  - The **TI bit** selects either the global or the local descriptor table.

  - **Requested Privilege Level** (RPL) requests the access privilege level of a memory segment.

Figure 5.2 The contents of a segment register during protected mode operation of the 80286 through Core2 microprocessors



- Figure 5.3 shows how the segment register, containing a selector, chooses a descriptor from the global descriptor table.

- The entry in the global descriptor table selects a segment in the memory system.

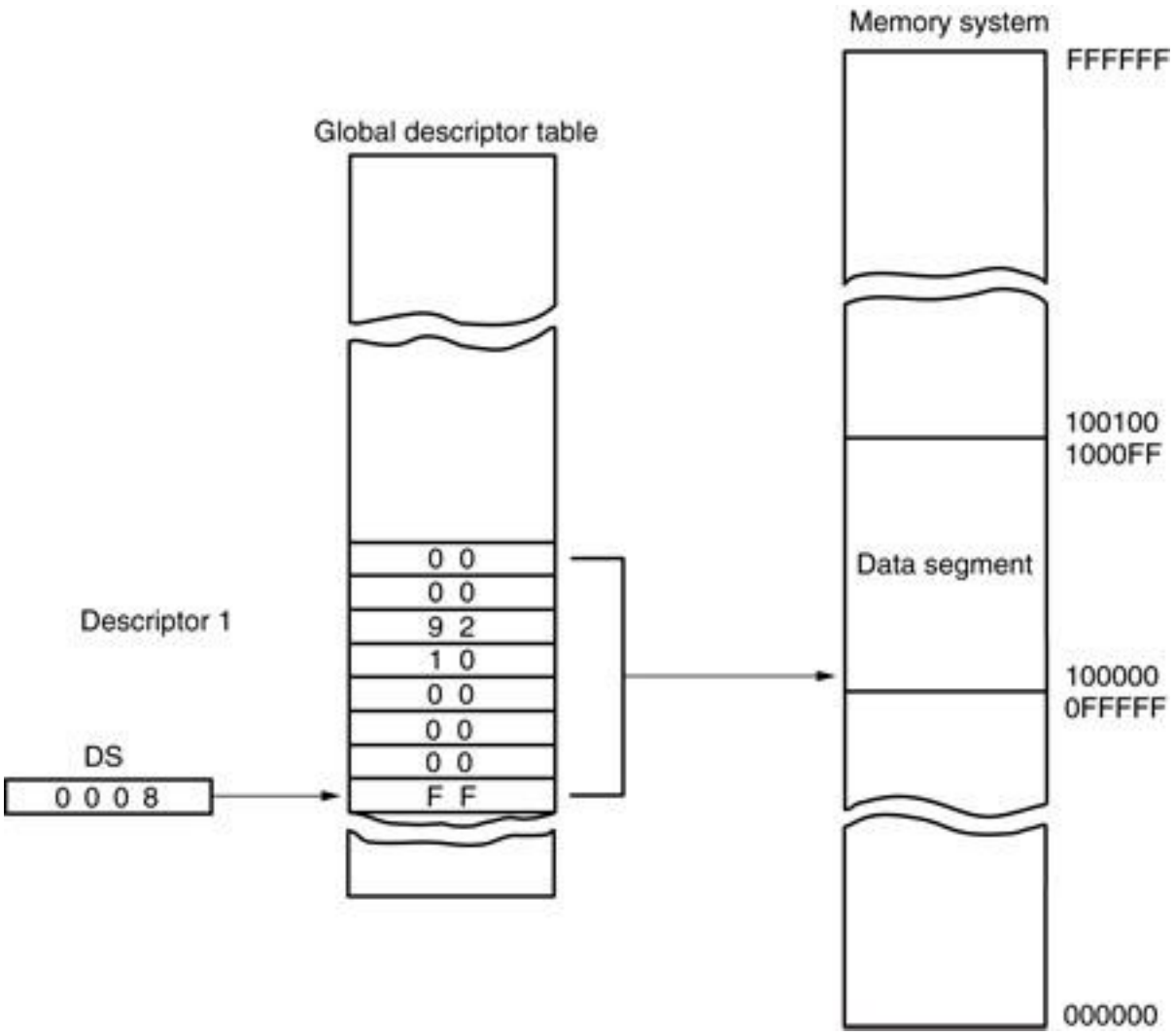
- In this illustration, DS contains 0008H, which accesses the descriptor number 1 from the global descriptor table using a requested privilege level of 00.

- Descriptor number 1 contains a descriptor that defines the base address as 00100000H with a segment limit of 000FFH

- This means that a value of 0008H loaded into DS causes the microprocessor to use memory locations 00100000H–001000FFH

- Descriptor zero is called the null descriptor, must contain all zeros, and may not be used for accessing memory.

Figure 5.3 Using the DS register to select a description from the global descriptor table. In this example, the DS register accesses memory locations 00100000H–001000FFH as a data segment



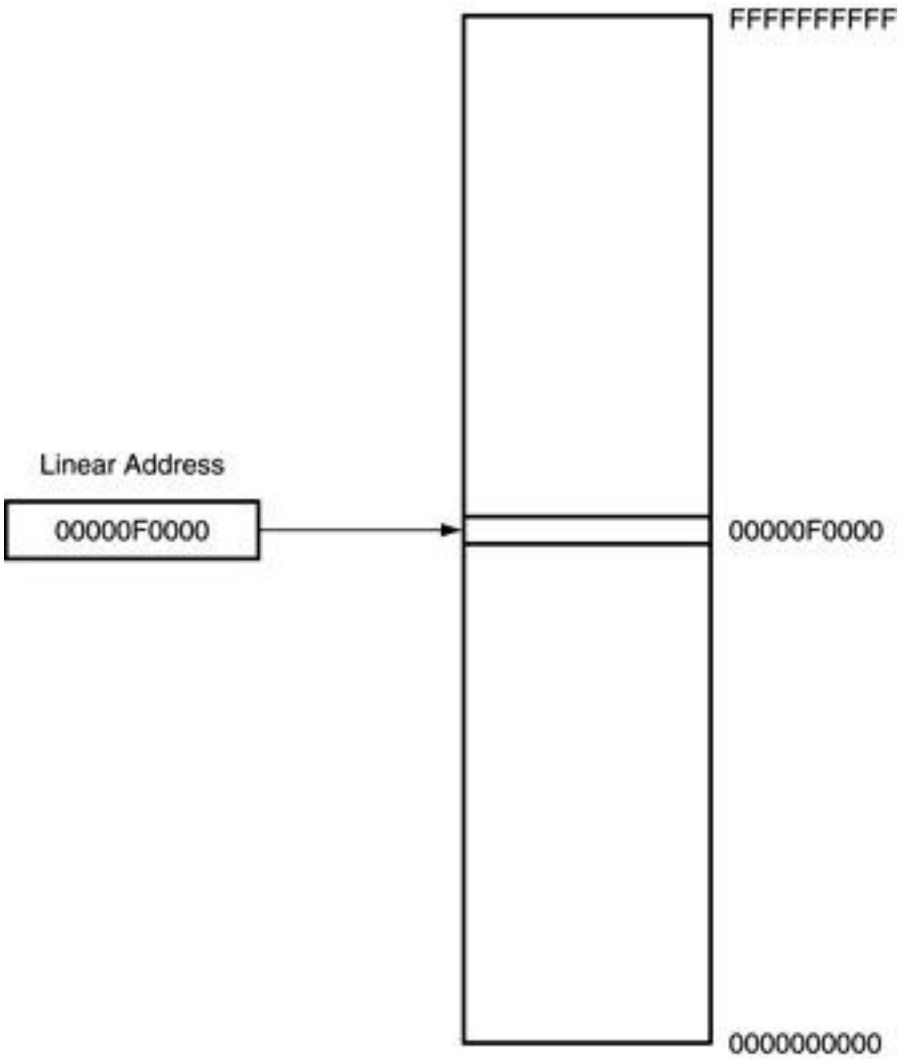
Flat Mode Memory Addressing

- A *flat mode memory* system is one in which there is no segmentation.
  - does not use a segment register to address a location in the memory.

- The memory system in a Pentium-based computer (Pentium 4 or Core2) that uses the 64-bit extensions uses a flat mode memory system.

- First byte address is at 00 0000 0000H; the last location is at FF FFFF FFFFH.
  - address is 40-bits
- The segment register still selects the privilege level of the software.

Figure 5.4 The 64-bit flat mode memory model



College of Electronics Engineering

Systems & Control Engineering Department

Microprocessors I

Lectures 7 and 8 (Addressing Modes)

Zeyad T. Shareef

DATA ADDRESSING MODES

- MOV instruction is a common and flexible instruction.
  - provides a basis for explanation of data-addressing modes
- Figure 6.1 illustrates the MOV instruction and defines the direction of data flow.
- **Source** is to the right and **destination** to the left, next to the opcode MOV.
  - an **opcode**, or operation code, tells the microprocessor which operation to perform.

Figure 6.1 The MOV instruction showing the source, destination, and direction of data flow.

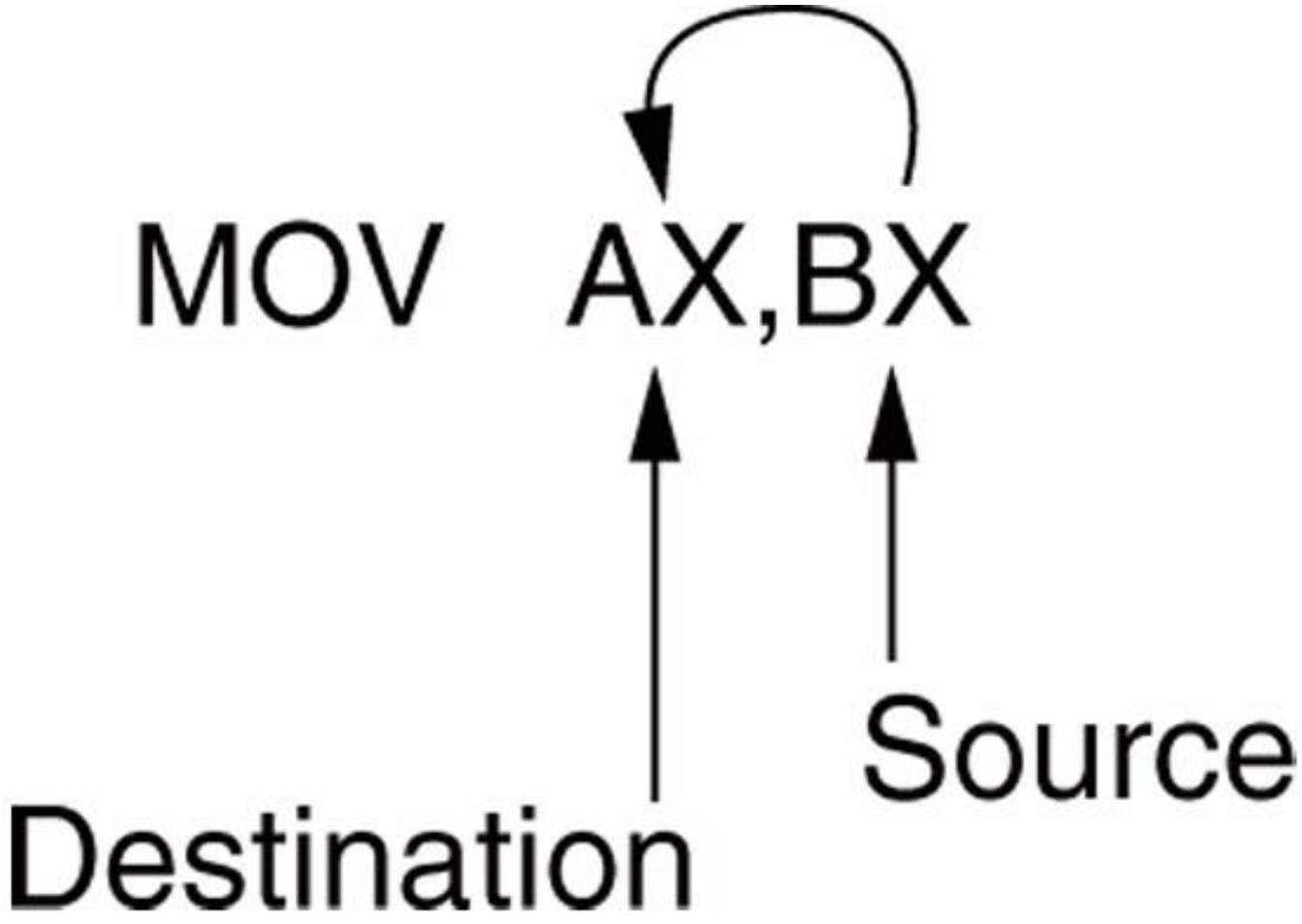
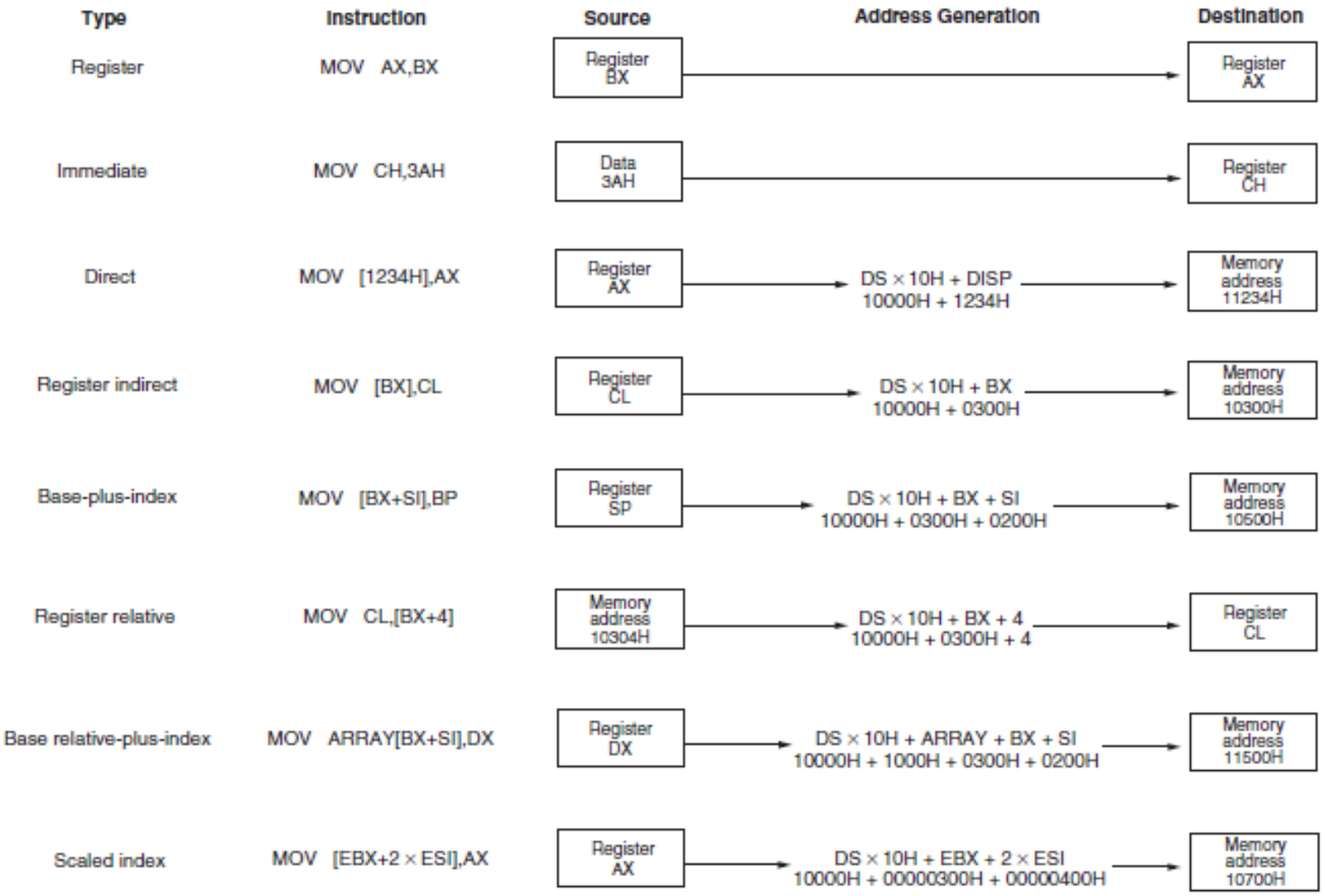


Figure 6.2 8086–Core2 data-addressing modes



Notes: EBX = 00000300H, ESI = 00000200H, ARRAY = 1000H, and DS = 1000H

Register Addressing

- The most common form of data addressing.
- The microprocessor contains these 8-bit register names used with register addressing: AH, AL, BH, BL, CH, CL, DH, and DL.
- 16-bit register names: AX, BX, CX, DX, SP, BP, SI, and DI.

- In 80386 & above, extended 32-bit register names are: EAX, EBX, ECX, EDX, ESP, EBP, EDI, and ESI.
- 64-bit mode register names are: RAX, RBX, RCX, RDX, RSP, RBP, RDI, RSI, and R8 through R15.
- Important for instructions to use registers that are the same size.
  - *never* mix an 8-bit \with a 16-bit register, an 8- or a 16-bit register with a 32-bit register
  - this is not allowed by the microprocessor and results in an error when assembled.
- Note that a few instructions are exceptions to this rule.

- Figure 6.3 shows the operation of the MOV BX, CX instruction.
- The source register's contents do not change.
  - the destination register's contents do change.
- The contents of the destination register or destination memory location change for all instructions except the CMP and TEST instructions.
- The MOV BX, CX instruction does not affect the leftmost 16 bits of register EBX.

Figure 6.3 The effect of executing the MOV BX, CX instruction at the point just before the BX register changes. Note that only the rightmost 16 bits of register EBX change

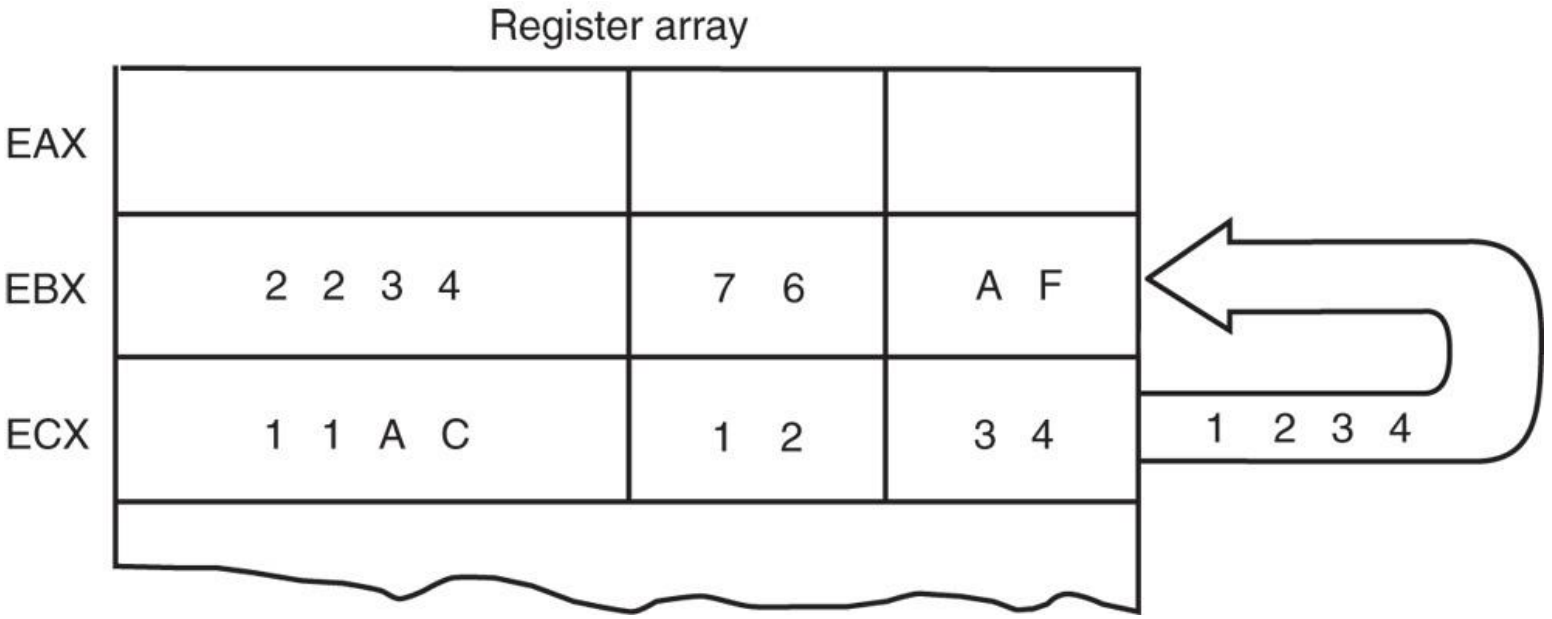


Table 6.1 Examples of register addressing

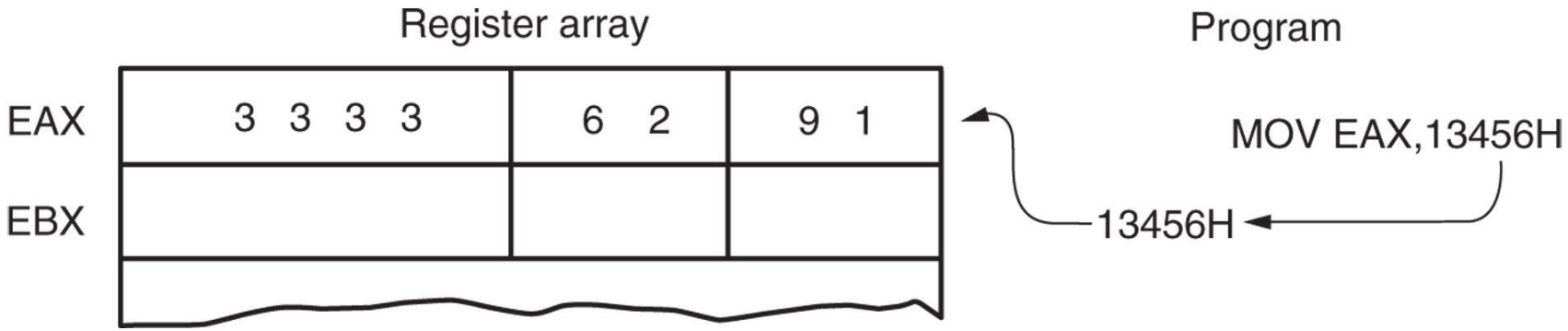
Assembly Language	*Size*	*Operation*
MOV AL,BL	8 bits	Copies BL into AL
MOV CH,CL	8 bits	Copies CL into CH
MOV R8B,CL	8 bits	Copies CL to the byte portion of R8 (64-bit mode)
MOV R8B,CH	8 bits	Not allowed
MOV AX,CX	16 bits	Copies CX into AX
MOV SP,BP	16 bits	Copies BP into SP
MOV DS,AX	16 bits	Copies AX into DS
MOV BP,R10W	16 bits	Copies R10 into BP (64-bit mode)
MOV SI,DI	16 bits	Copies DI into SI
MOV BX,ES	16 bits	Copies ES into BX
MOV ECX,EBX	32 bits	Copies EBX into ECX
MOV ESP,EDX	32 bits	Copies EDX into ESP
MOV EDX,R9D	32 bits	Copies R9 into EDX (64-bit mode)
MOV RAX,RDX	64 bits	Copies RDX into RAX
MOV DS,CX	16 bits	Copies CX into DS
MOV ES,DS	—	Not allowed (segment-to-segment)
MOV BL,DX	—	Not allowed (mixed sizes)
MOV CS,AX	—	Not allowed (the code segment register may not be the destination register)

Immediate Addressing

- Term *immediate* implies that data immediately follow the hexadecimal opcode in the memory.
  - immediate data are constant data.
  - data transferred from a register or memory location are variable data.
- Figure 6.4 shows the operation of a

MOV EAX,13456H instruction.

Figure 6.4 The operation of the MOV EAX,13456H instruction. This instruction copies the immediate data (13456H) into EAX



- As with the MOV instruction illustrated in Figure 6.3, the source data overwrites the destination data.

- The letter **H** appends hexadecimal data.
- Decimal data don’t require any special codes or adjustments.
  - an example is the 100 decimal in the MOV AL,100 instruction

- An ASCII-coded character or characters may be depicted in the immediate form if the ASCII data are enclosed in apostrophes.
  - be careful to use the apostrophe (‘) for ASCII data and not the single quotation mark.
- Binary data are represented if the binary number is followed by the letter **B**.
  - in some assemblers, the letter **Y.**

Table 6.2 Examples of immediate addressing

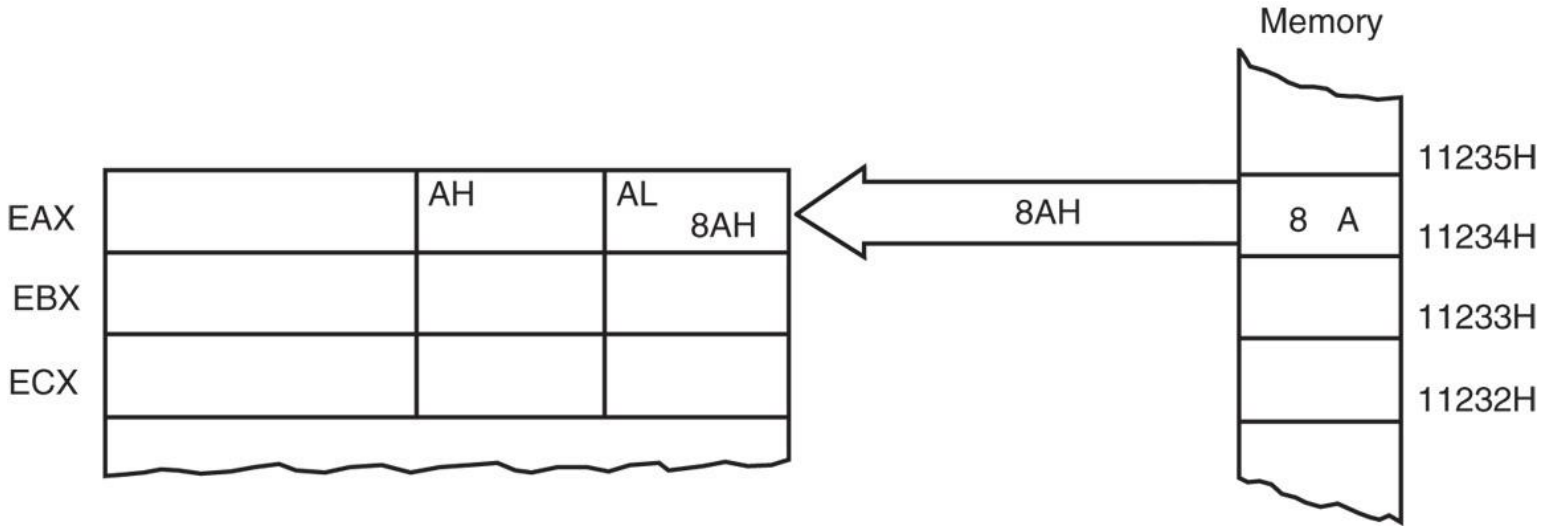
Assembly Language	*Size*	*Operation*
MOV BL,44	8 bits	Copies 44 decimal (2CH) into BL
MOV AX,44H	16 bits	Copies 0044H into AX
MOV SI,0	16 bits	Copies 0000H into SI
MOV CH,100	8 bits	Copies 100 decimal (64H) into CH
MOV AL,'A'	8 bits	Copies ASCII A into AL
MOV AH,1	8 bits	Not allowed in 64-bit mode, but allowed in 32- or 16-bit modes
MOV AX,'AB'	16 bits	Copies ASCII BA* into AX
MOV CL,11001110B	8 bits	Copies 11001110 binary into CL
MOV EBX,12340000H	32 bits	Copies 12340000H into EBX
MOV ESI,12	32 bits	Copies 12 decimal into ESI
MOV EAX,100B	32 bits	Copies 100 binary into EAX
MOV RCX,100H	64 bits	Copies 100H into RCX

*Note: This is not an error. The ASCII characters are stored as BA, so exercise care when using word-sized pairs of ASCII characters.

Direct Data Addressing

- Applied to many instructions in a typical program.
- Address is formed by adding the displacement to the default data segment address or an alternate segment address.

Figure 6.5 The operation of the MOV AL,[1234H] instruction when DS=1000H

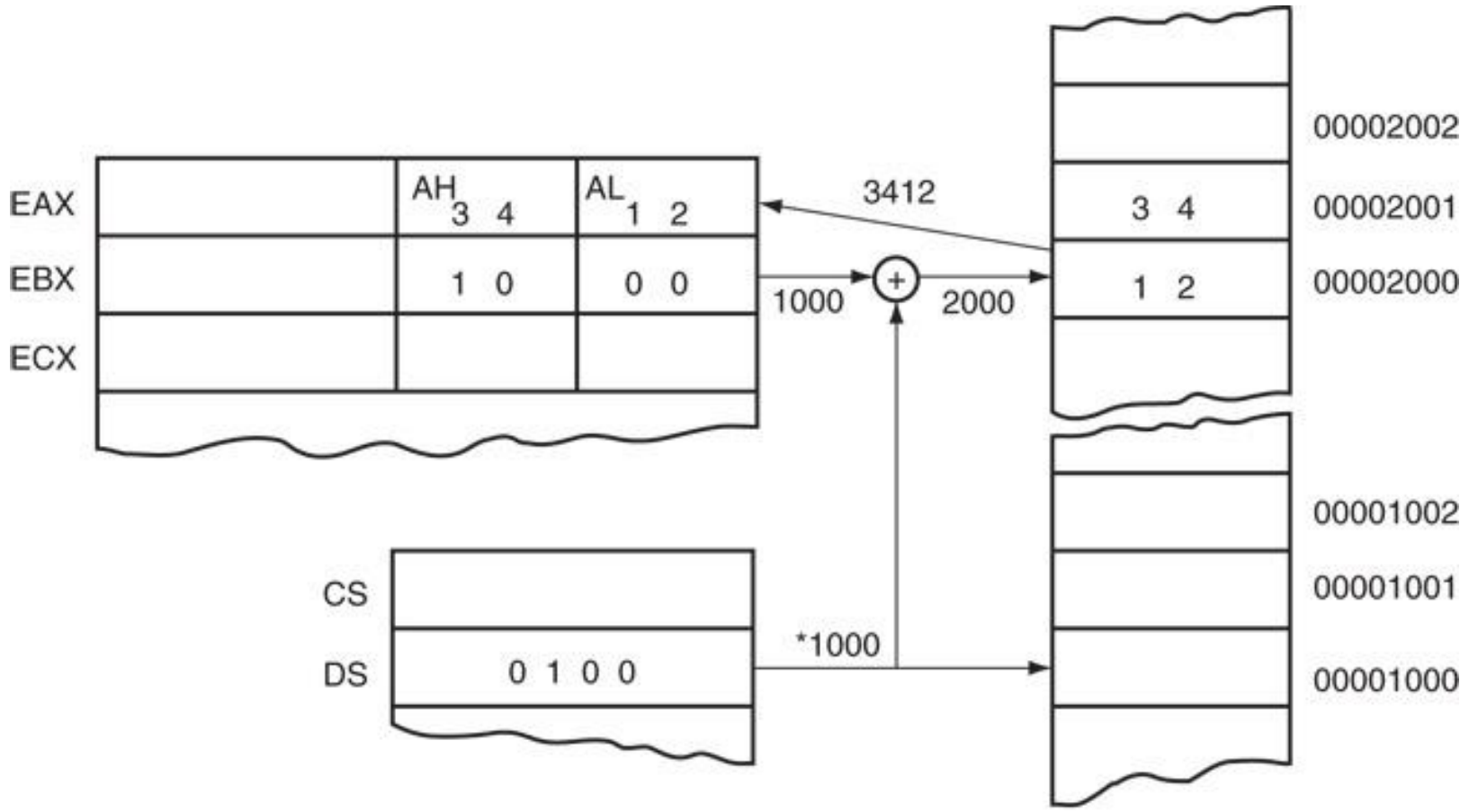


- This instruction transfers a copy contents of memory location 11234H into AL.
- the effective address is formed by adding 1234H (the offset address) and 10000H (the data segment address of 1000H times 10H) in a system operating in the real mode.

Register Indirect Addressing

- Allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI, and SI.

Figure 6.6 The operation of the MOV AX,[BX] instruction when BX = 1000H and DS = 0100H. Note that this instruction is shown after the contents of memory are transferred to AX



*After DS is appended with a 0.

- The **data segment** is used by default with register indirect addressing or any other mode that uses BX, DI, or SI to address memory.
- If the BP register addresses memory, the **stack segment** is used by default.
  - these settings are considered the default for these four index and base registers

Table 6.3 Examples of register indirect addressing

Assembly Language	*Size*	*Operation*
MOV CX,[BX]	16 bits	Copies the word contents of the data segment memory location addressed by BX into CX
MOV [BP],DL*	8 bits	Copies DL into the stack segment memory location addressed by BP
MOV [DI],BH	8 bits	Copies BH into the data segment memory location addressed by DI
MOV [DI],[BX]	—	Memory-to-memory transfers are not allowed except with string instructions
MOV AL,[EDX]	8 bits	Copies the byte contents of the data segment memory location addressed by EDX into AL
MOV ECX,[EBX]	32 bits	Copies the doubleword contents of the data segment memory location addressed by EBX into ECX
MOV RAX,[RDX]	64 bits	Copies the quadword contents of the memory location address by the linear address located in RDX into RAX (64-bit mode)

Base-Plus-Index Addressing

- Similar to indirect addressing because it indirectly addresses memory data.
- The base register often holds the beginning location of a memory array.
  - the index register holds the relative position of an element in the array.
  - whenever BP addresses memory data, both the stack segment register and BP generate the effective address.

- Figure 6.7 shows how data are addressed by the MOV DX,[BX + DI] instruction when the microprocessor operates in the real mode.

Figure 6.7 An example showing how the base-plus-index addressing mode functions for the MOV DX,[BX + DI] instruction. Notice that memory address 02010H is accessed because DS=0100H, BX=1000H and DI=0010H

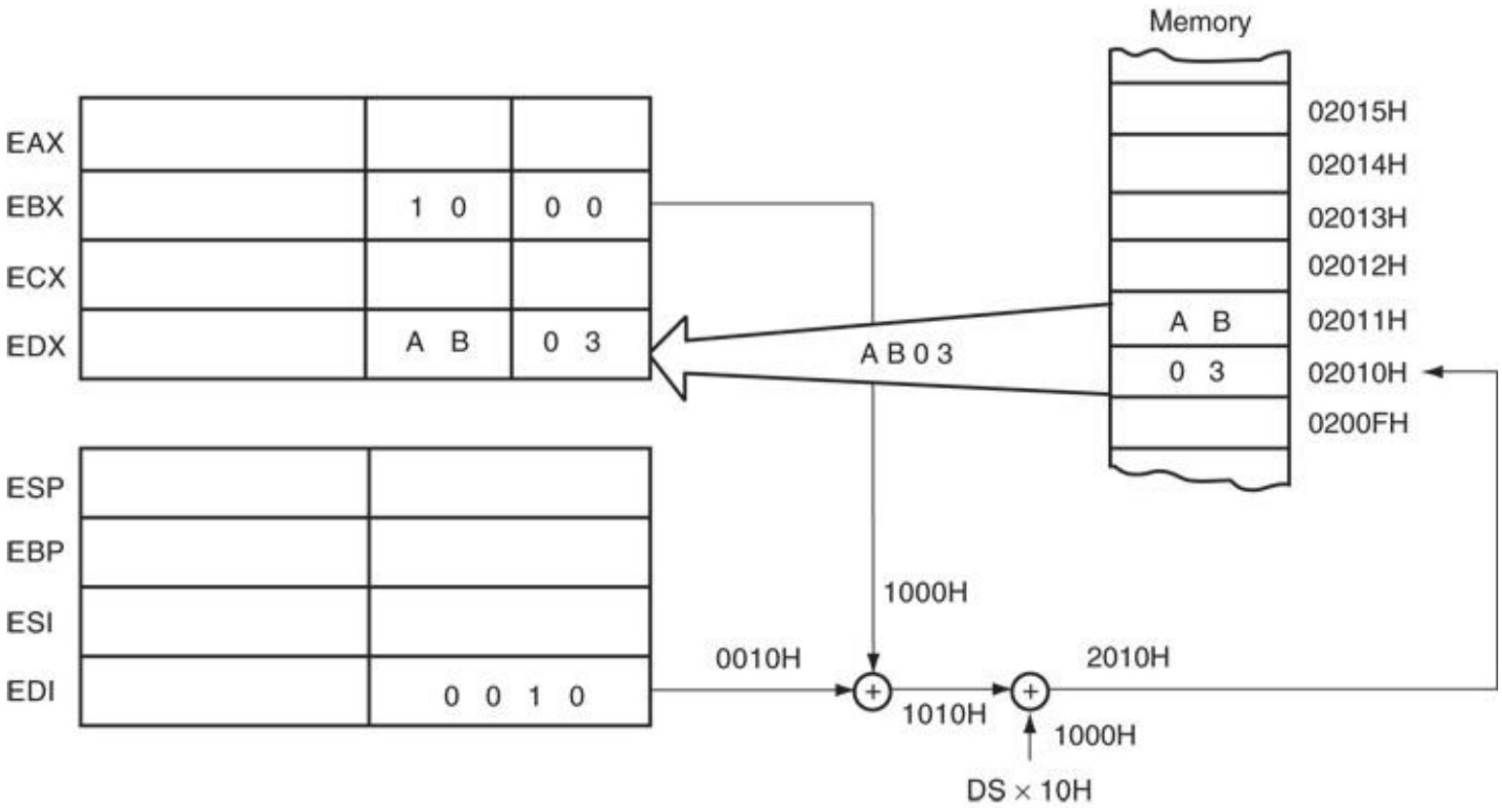


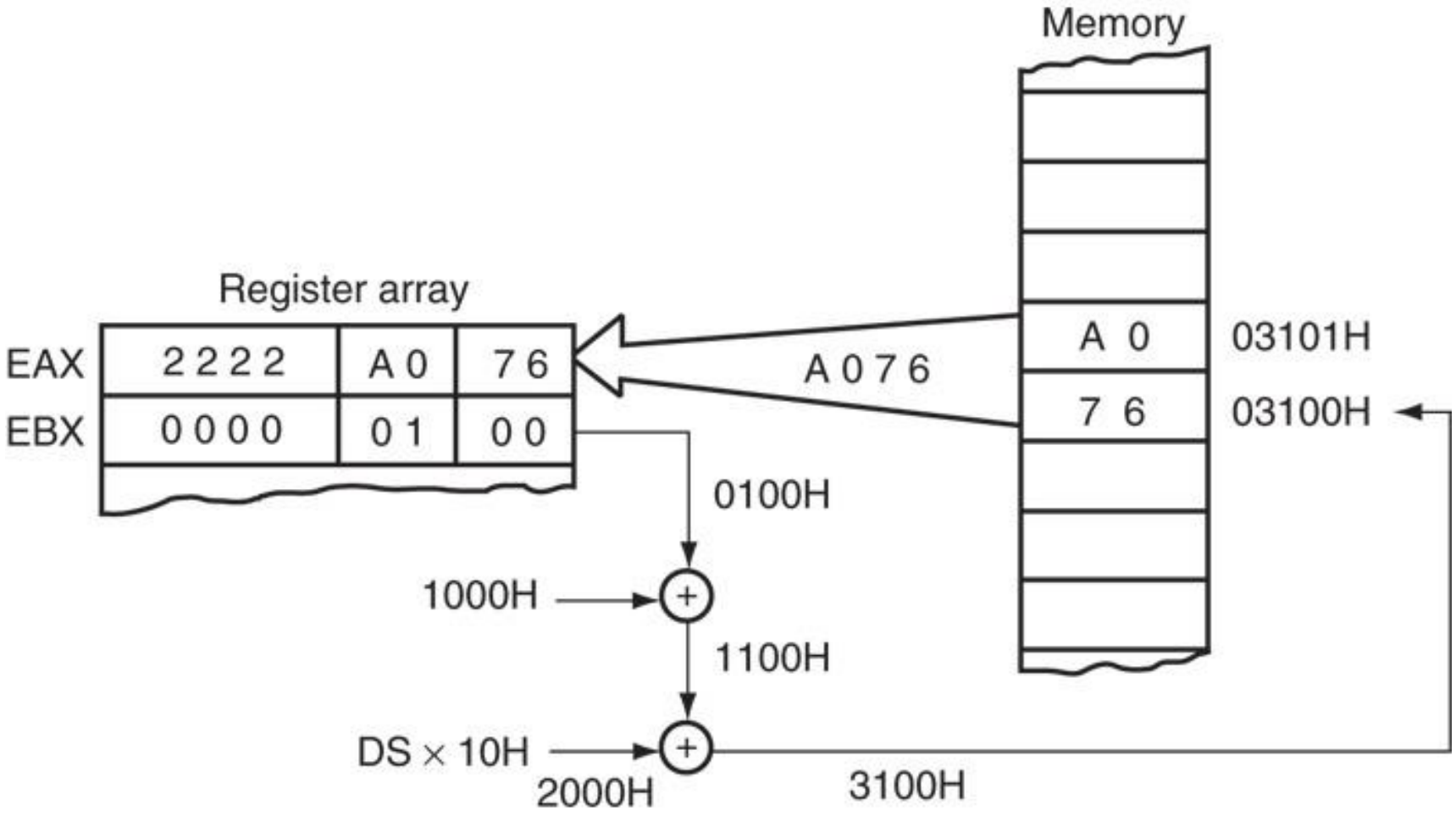
Table 6.4 Examples of base-plus index addressing

Assembly Language	*Size*	*Operation*
MOV CX,[BX+DI]	16 bits	Copies the word contents of the data segment memory location addressed by BX plus DI into CX
MOV CH,[BP+SI]	8 bits	Copies the byte contents of the stack segment memory location addressed by BP plus SI into CH
MOV [BX+SI],SP	16 bits	Copies SP into the data segment memory location addressed by BX plus SI
MOV [BP+DI],AH	8 bits	Copies AH into the stack segment memory location addressed by BP plus DI
MOV CL,[EDX+EDI]	8 bits	Copies the byte contents of the data segment memory location addressed by EDX plus EDI into CL
MOV [EAX+EBX],ECX	32 bits	Copies ECX into the data segment memory location addressed by EAX plus EBX
MOV [RSI+RBX],RAX	64 bit	Copies RAX into the linear memory location addressed by RSI plus RBX (64-bit mode)

Register Relative Addressing

- Similar to base-plus-index addressing and displacement addressing.
  - data in a segment of memory are addressed by adding the displacement to the contents of a base or an index register (BP, BX, DI, or SI)
- Figure 6.8 shows the operation of the MOV AX,[BX+1000H] instruction.
- Remember that a real mode segment is 64K bytes long.

Figure 6.8 The operation of the MOV AX, [BX+1000H] instruction, when BX=0100H and DS=0200H



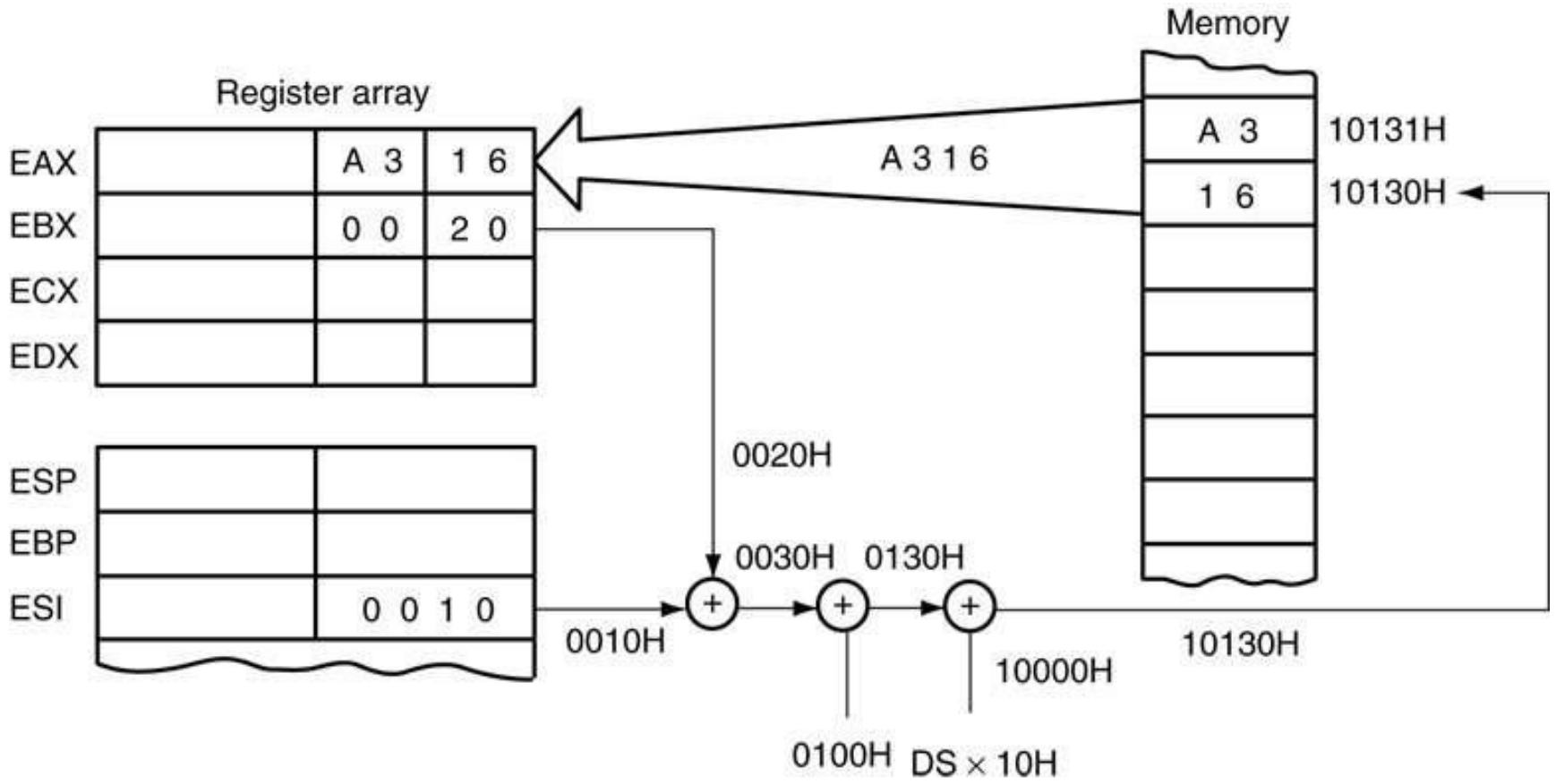
Base Relative-Plus-Index Addressing

- Similar to base-plus-index addressing.
  - adds a displacement
  - uses a base register and an index register to form the memory address.

- Least-used addressing mode.

- Figure 6.9 shows how data are referenced if the instruction executed by the microprocessor is MOV AX,[BX + SI + 100H].
  - displacement of 100H adds to BX and SI to form the offset address within the data segment
- This addressing mode is too complex for frequent use in programming.

Figure 6.9 An example of base relative-plus-index addressing using a MOV AX,[BX+SI+100H] instruction. Note: DS=1000H, BX=20H, SI=10H

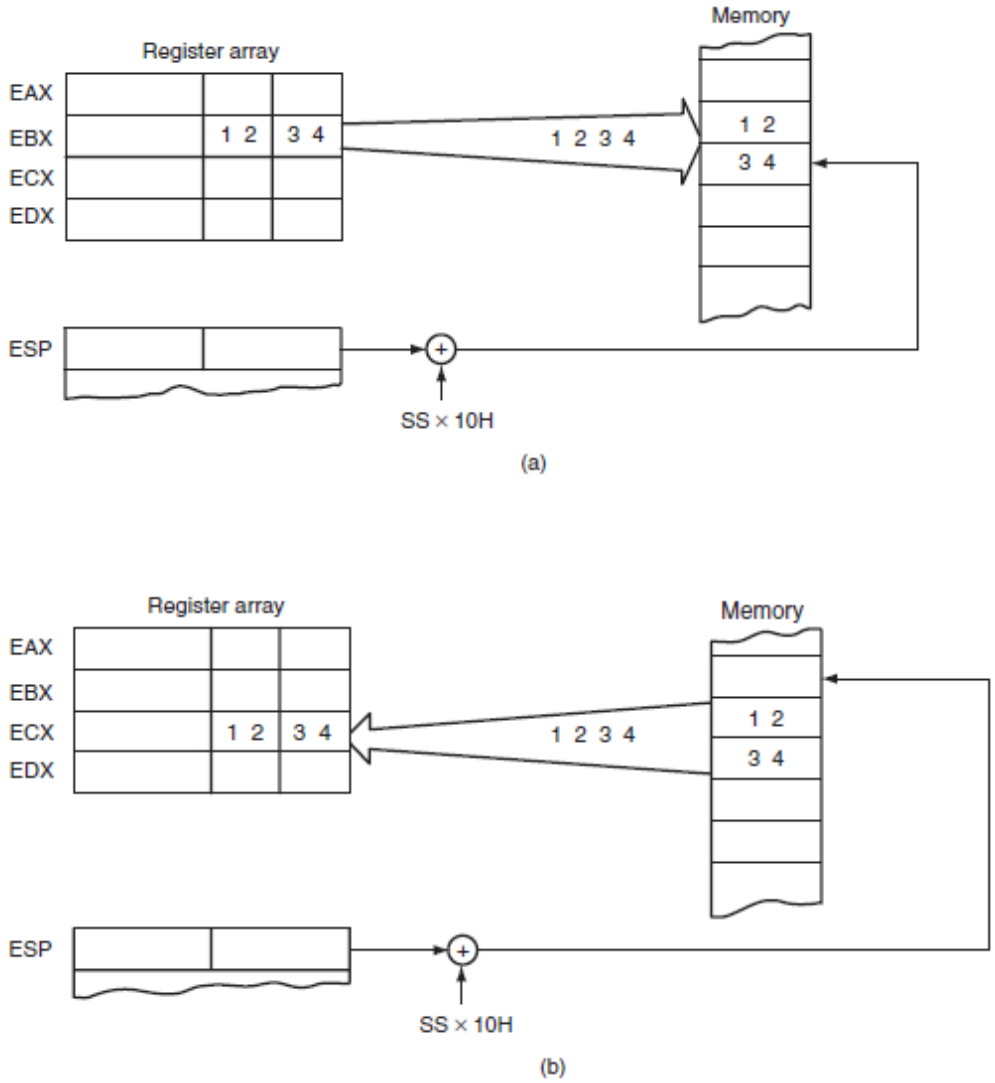


Stack Memory-Addressing Modes

- The stack plays an important role in all microprocessors.
  - holds data temporarily and stores return addresses used by procedures.
- Stack memory is LIFO (**last-in, first-out**) memory.

- Data are placed on the stack with a **PUSH instruction**; removed with a **POP instruction.**
- Stack memory is maintained by two registers:
  - the stack pointer (SP or ESP)
  - the stack segment register (SS)

Figure 6.10 The PUSH and POP instructions: (a) PUSH BX places the contents of BX onto the stack; (b) POP CX removes data from the stack and places them into CX. Both instructions are shown after execution.



- Note that PUSH and POP store or retrieve words of data—never bytes—in 8086 - 80286.
- 80386 and above allow words or double-words to be transferred to and from the stack.
- Data may be pushed onto the stack from any 16-bit register or segment register.
  - in 80386 and above, from any 32-bit extended register
- Data may be popped off the stack into any register or any segment register except CS.

College of Electronics Engineering

Systems & Control Engineering Department

Microprocessors I

Lecture 9
(Instruction Set Architecture)

Zeyad T. Shareef

Machine Instruction

- An instruction (or command) is encoded as a bit pattern recognizable by the CPU.
- Machine Instruction types:
  - Data Transfer: copy data from one location to another
  - Arithmetic/Logic: use existing bit patterns to compute a new bit patterns
  - Control: direct the execution of the program

Memory Operations

- Memory contains data & program instructions
- Control circuits initiate transfer of data and instructions between memory and processor.
- ***Read* operation**: memory retrieves contents at address location given by processor
- ***Write* operation**: memory overwrites contents at given location with given data

Register Transfer Notation

- Register transfer notation is used to describe hardware-level data transfers and operations
- Use [...] to denote contents of a location.
- Use $\leftarrow$ to denote transfer to a destination
- Example: R2 $\leftarrow$ [LOC]
  (transfer from LOC in memory to register R2)

Assembly-Language Notation

- Another notation is needed to represent machine instructions & programs using them;

  Assembly language is used for this purpose.
- Examples in this chapter will use English words for the operations (e.g., Load, Store, and Add)
- In the assembly language instructions of actual (commercial) processors, mnemonics are used, usually abbreviations (e.g., LD, ST, and ADD).
- Mnemonics differ from processor to processor.

RISC and CISC Instruction Sets

- Nature of instructions distinguishes computers.

- Two fundamentally different approaches:
  - Reduced Instruction Set Computers (RISC)
    - ➢ have one-word instructions
    - ➢ Few, simple, efficient, and fast instructions
    - ➢ Examples: PowerPC from Apple/IBM/Motorola and ARM

  - Complex Instruction Set Computers (CISC)
    - ➢ have multi-word instructions
    - ➢ Many, convenient, and powerful instructions
    - ➢ Example: Intel

RISC Instruction Sets

- Each RISC instruction fits in a single word.
- A load/store architecture is used, meaning:
  - only Load and Store instructions are used to access memory operands
  - operands for arithmetic/logic instructions must be in registers, or one of them may be given explicitly in instruction word.

RISC Instruction Sets

- For simplicity, lets assume that A, B, and C are labels representing memory word addresses; R2, R3 and R4 are processor registers.

- To add the contents of locations A and B, then store the result in location C, the following steps are needed:

  Fetch contents of locations A and B, compute sum, and transfer result to location C.

RISC Instruction Sets

- Sequence of simple RISC instructions for this task:

```
Load    R2, A
Load    R3, B
Add     R4, R2, R3
Store   R4, C
```

- **Load** instruction transfers data to register.
- **Store** instruction transfers data to the memory.

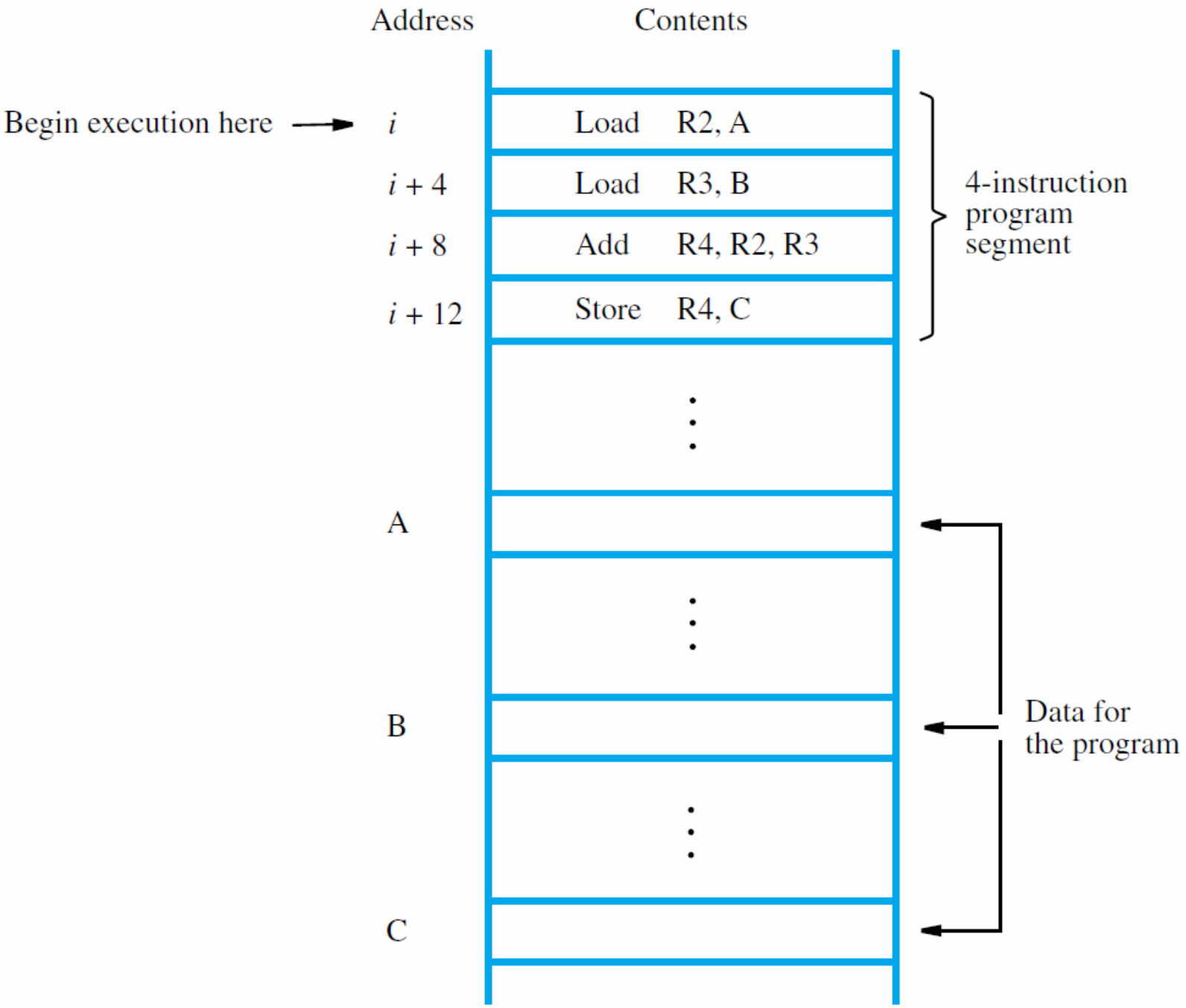
Fetching and executing instructions

- Example: Load R2, LOC

  The processor control circuits do the following:
  - Send address in PC to memory; issue **Read**
  - Load instruction from memory into IR
  - Increment PC to point to next instruction
  - Send address LOC to memory; issue **Read**
  - Load word from memory into register R2

A Program in the Memory

- Consider the preceding 4-instruction program
- How is it stored in the memory?
  (32-bit word length, byte-addressable)
- Place first RISC instruction word at address *i*
- Remaining instructions are at $i + 4$, $i + 8$, $i + 12$


Address
Contents
Begin execution here
i
i + 4
i + 8
i + 12
Load R2, A
Load R3, B
Add R4, R2, R3
Store R4, C
4-instruction program segment
A
B
C
Data for the program

Instruction Execution/Sequencing

- How is the previous program executed?
- Processor has program counter (PC) register
- Address *i* for first instruction placed in PC
- Control circuits fetch and execute instructions, one after another $\rightarrow$ straight-line sequencing
- During execution of each instruction, PC register is incremented by 4
- PC contents are $i + 16$ after Store is executed