SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107
Accredited by NAAC-UGC with 'A' Grade
Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department of Information Technology

Course Name – 19IT301 Computer Organization and Aechitecture

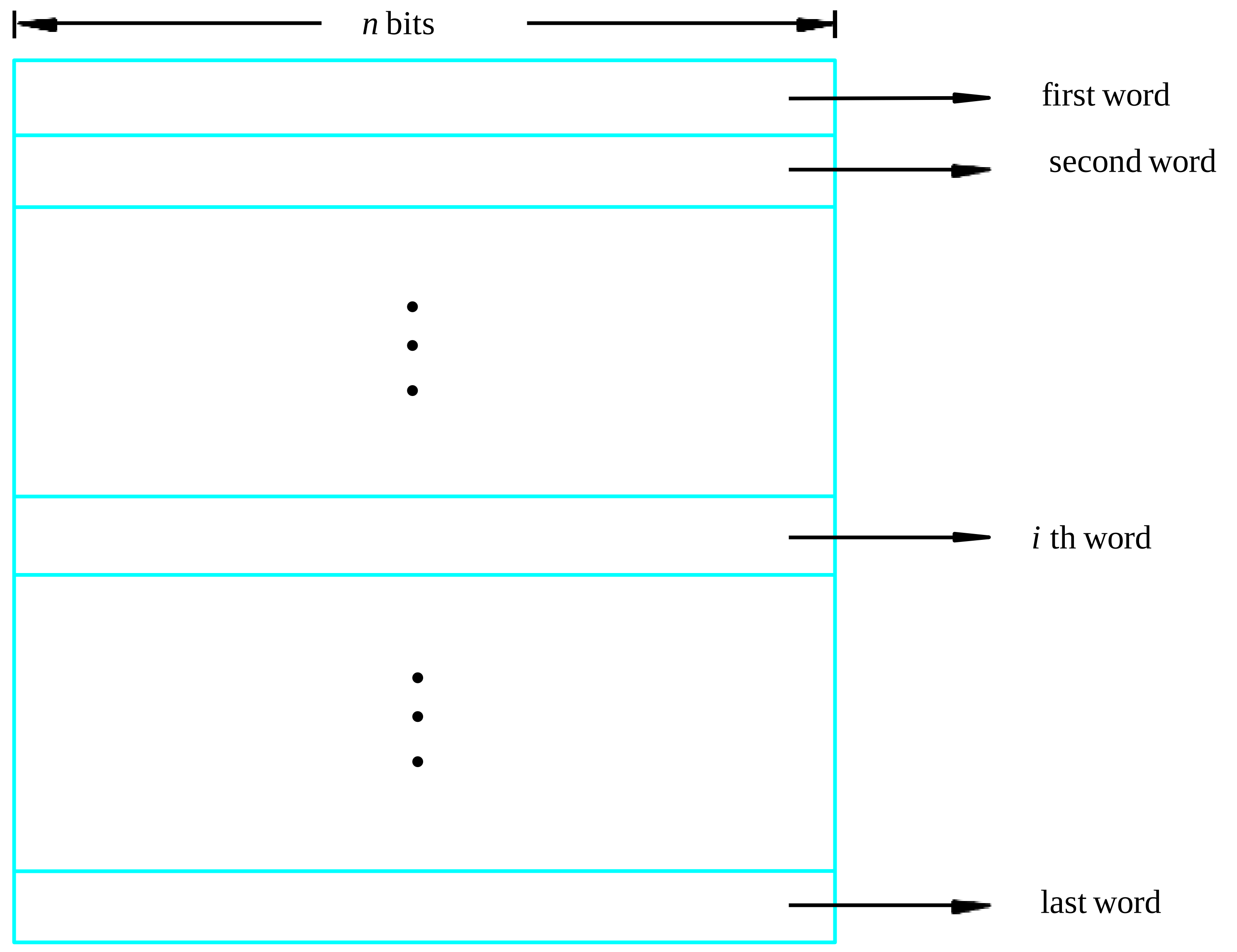
II Year / III Semester

Unit 1 – Basic Structures of Computers

Topic :Memory Location, Addresses, and Operation

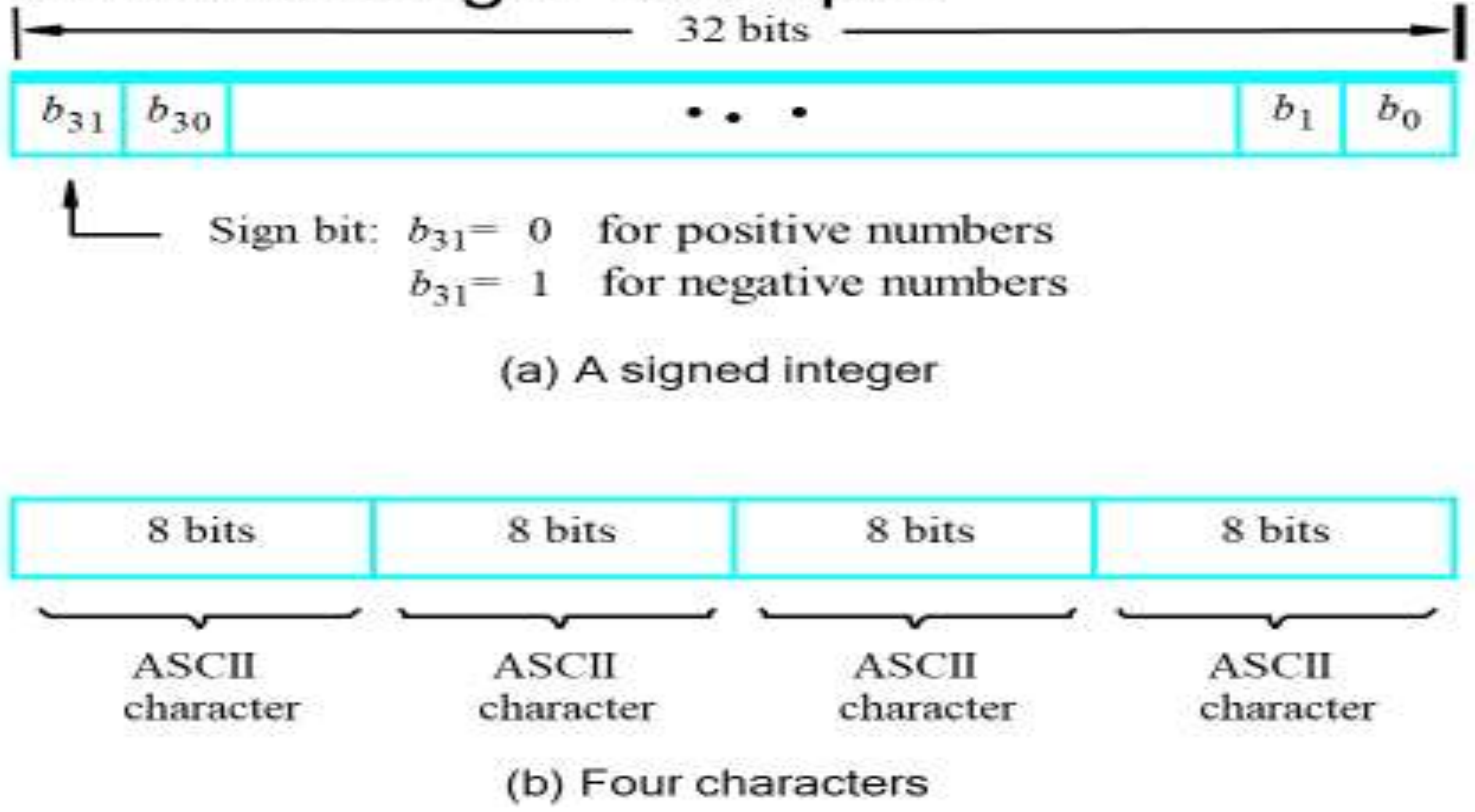
Memory

- Memory consists of many millions of storage cells, each of which can store 1 bit.
- Data is usually accessed in *n*-bit groups. *n* is called word length.



Instruction

- 32-bit word length example



Memory Location, Addresses, and Operation

- To retrieve information from memory, either for one word or one byte (8-bit), addresses for each location are needed.
- A *k*-bit address memory has 2^k memory locations, namely $0 - 2^k-1$, called memory space.
- 24-bit memory: $2^{24} = 16,777,216 = 16M$ ($1M=2^{20}$)
- 32-bit memory: $2^{32} = 4G$ ($1G=2^{30}$) 1K(kilo)=2^{10}
- 1T(tera)=2^{40}

Memory Location, Addresses, and Operation

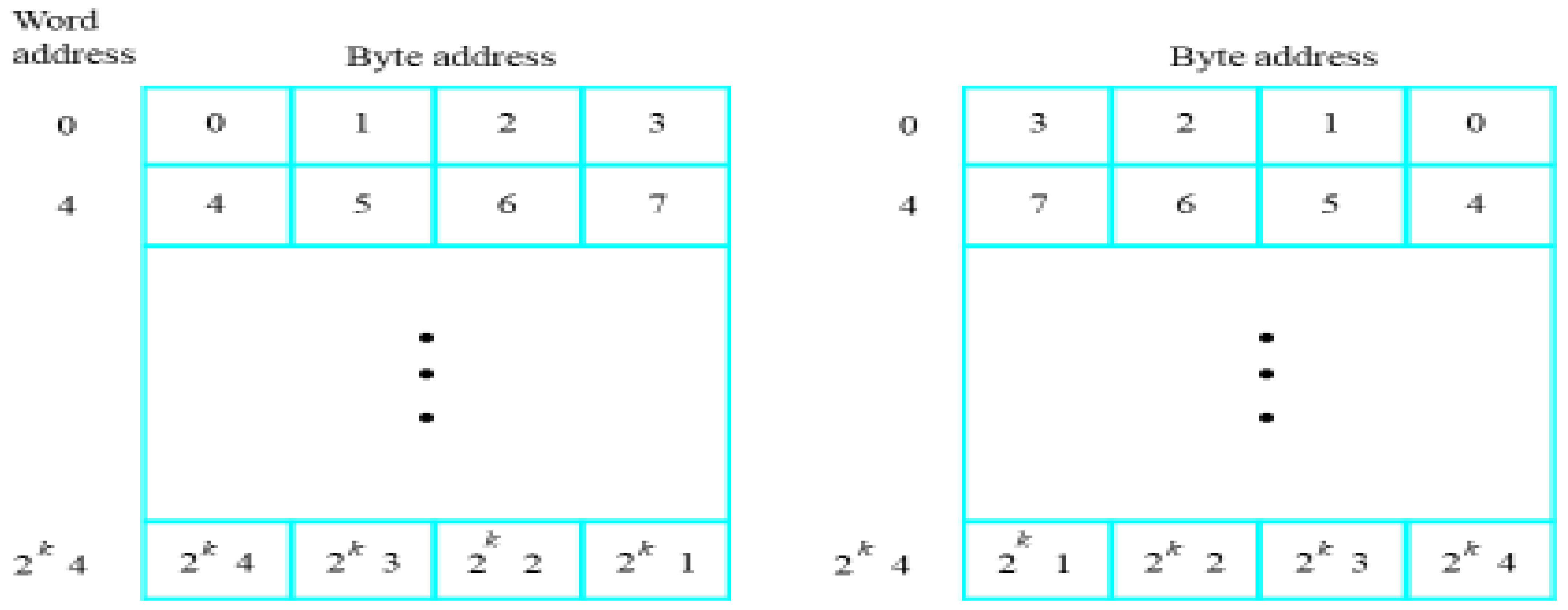
- It is impractical to assign distinct addresses to individual bit locations in the memory.
- The most practical assignment is to have successive addresses refer to successive byte locations in the memory – byte- addressable memory.
- Byte locations have addresses 0, 1, 2, … If word length is 32 bits, they successive words are located at addresses 0, 4, 8,…

Big-endian and Little-Endian Assignments

Big-Endian and Little-Endian Assignments

Big-Endian: lower byte addresses are used for the most significant bytes of the word

Little-Endian: opposite ordering. lower byte addresses are used for the less significant bytes of the word



Word address	Byte address			
0	0	1	2	3
4	4	5	6	7
	• • •			
2^k 4	2^k 4	2^k 3	2^k 2	2^k 1

(a) Bigendian assignment

Word address	Byte address			
0	3	2	1	0
4	7	6	5	4
	• • •			
2^k 4	2^k 1	2^k 2	2^k 3	2^k 4

(b) Littleendian assignment

Figure 2.7. Byte and word addressing.

Memory Location, Addresses, and Operation

- Address ordering of bytes
- Word alignment
  - Words are said to be aligned in memory if they begin at a byte addr. that is a multiple of the num of bytes in a word.

16-bit word: word addresses: 0, 2, 4,….
32-bit word: word addresses: 0, 4, 8,….
64-bit word: word addresses: 0, 8,16,….

Memory Operation

Load (or Read or Fetch)

➢Copy the content. The memory content doesn’t change. Address – Load

➢Registers can be used

Store (or Write)

➢Overwrite the content in memory

➢Address and Data – Store

➢Registers can be used

Register Transfer Notation

- Identify a location by a symbolic name standing for its hardware binary address (LOC, R0,…)
- Contents of a location are denoted by placing square brackets around the name of the location ($R1 \leftarrow [LOC]$, $R3 \leftarrow [R1]+[R2]$)
- Register Transfer Notation (RTN)

Assembly Language Notation

- Represent machine instructions and programs.
- Move LOC, R1 = R1$\leftarrow$[LOC]
- Add R1, R2, R3 = R3 $\leftarrow$[R1]+[R2]

CPU ORGANIZATION

- Single Accumulator
  - ➢ Result usually goes to the Accumulator
  - ➢ Accumulator has to be saved to memory quite often

General Register

- ➢ Registers hold operands thus reduce memory traffic
- ➢ Register bookkeeping

- Stack
  - Operands and result are always in the stack

Instruction format

- Three-Address Instructions

  ADD R1, R2, R3 R1 ← R2 + R3
- Two-Address Instructions

  ADD R1, R2 R1 ← R1 + R2
- One-Address Instructions

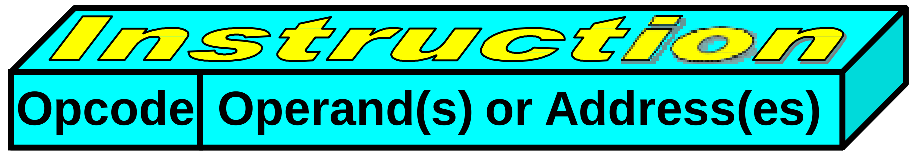
  ADD M AC ← AC + M[AR]

Zero-Address Instructions

ADD TOS ← TOS + (TOS – 1)

RISC Instructions

Lots of registers. Memory is restricted to Load & Store



Instruction Format Example

Example: Evaluate (A+B) * (C+D)

ADD R1, A, B	;R1 ← M[A] + M[B]
ADD R2, C, D	; R2 ← M[C] + M[D]
MUL X, R1, R2	; M[X] ← R1 * R2

Two Address Format

```
MOV   R1, A    ; R1 ← M[A]
ADD   R1, B    ; R1 ← R1 + M[B]
MOV   R2, C    ; R2 ← M[C]
ADD   R2, D    ; R2 ← R2 + M[D]
MUL   R1, R2   ; R1 ← R1 □ R2
MOV   X, R1    ; M[X] ← R1
```

One address Process

```
LOAD  A        ; AC ← M[A]
ADD   B        ; AC ← AC + M[B]
STORE T        ; M[T] ← AC
LOAD  C        ; AC ← M[C]
 ADD  D        ; AC ← AC + M[D]
MUL   T        ; AC ← AC □ M[T]
STORE X        ; M[X] ← AC
```

Zero Address Format

```
PUSH   A               ;TOS ← A
PUSH   B               ; TOS ← B
ADD                    ; TOS ← (A + B)
 PUSH  C               ; TOS ← C
PUSH   D               ; TOS ← D
MUL                    ; TOS ← (C + D)
 (C+D)*(A+B)           ; TOS ←
POP    X
ADD                    ; M[X] ← TOS
```

THANK YOU