COMPUTER ORGANIZATION AND ARCHITECTURE (IT 2202)

Lecture 4

Prof Hafizur Rahaman
Indian Institute of Engineering Science and Technology, Shibpur
(Formerly, Bengal Engineering and Science University, Shibpur)
Howrah 711 103, West Bengal, India

Memory Addressing and Languages

- Memory is one of the most important subsystems of a computer that determines the overall performance.
- We are storing the instruction and data in the memory. If the memory is slower, then loading the data from the memory will be slower. So, in that case we need to have a high speed memory. The memory is an array of storage locations with each storage location having a unique address.
- We have first location as 0000, next location as 0001 and so on. The last location may be 1111. So, it is an array of storage location each with a unique address. So, these are individual locations and this is the address associated with each location. And each storage location can hold a fixed amount of information, which can be multiple of bits which is the basic unit of data storage.

Memory Addressing and Languages

- A memory system with M locations and N bits per location is referred to as an M x N memory, where both M and N are typically some powers of 2. An example: 1024 x8=2^{10}x8.
- We have 10-bit in the address, and each location is having 8-bit.

Overview of Memory Organization

- Some terminologies about memory.
- A bit is a single binary digit either 0 or 1.
- Nibble is a collection of 4 bits.
- Byte is a collection of 8 bits.

- Word does not have a unique definition because we can either have a 32 bit word length or 64 bit word length. So, word does not have a unique definition.

- Memory is often byte organized. Here, each byte is having an address, that means every byte of the memory has a unique address. Multiple bytes of a data can be accessed by an instruction.

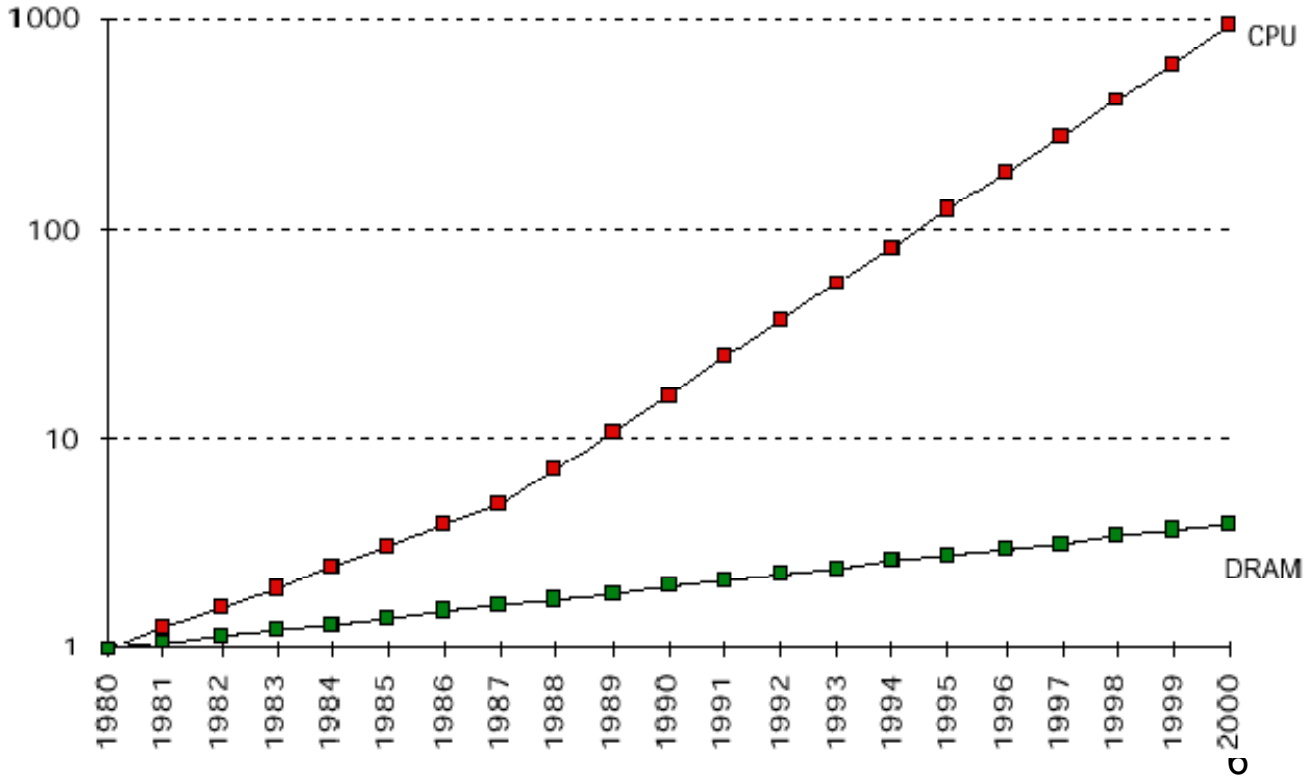
Overview of Memory Organization

- Example: ADD R1, Location. It is depending on how many bits, the ADD instruction will have, how many bits, R1 register will have, and how many bits the location will have; this will define that how many words this instruction will have or how many bytes this instruction will have.
- So, how many bytes this instruction will take is dependent on various other factors like the total number of instructions available in the computer. The total number of registers presents in the computer, and also the number of locations, we are having based on which we can determine the number of bytes required to represent this particular instruction.
- For higher data transfer rate, memory is often organized such that multiple bytes can be read or written simultaneously. This is basically needed to bridge the processor memory speed gap.
- As the processor speed is increasing memory speed is also increasing, but not at this rate the processor is increasing.

Processor- Memory Performance Gap

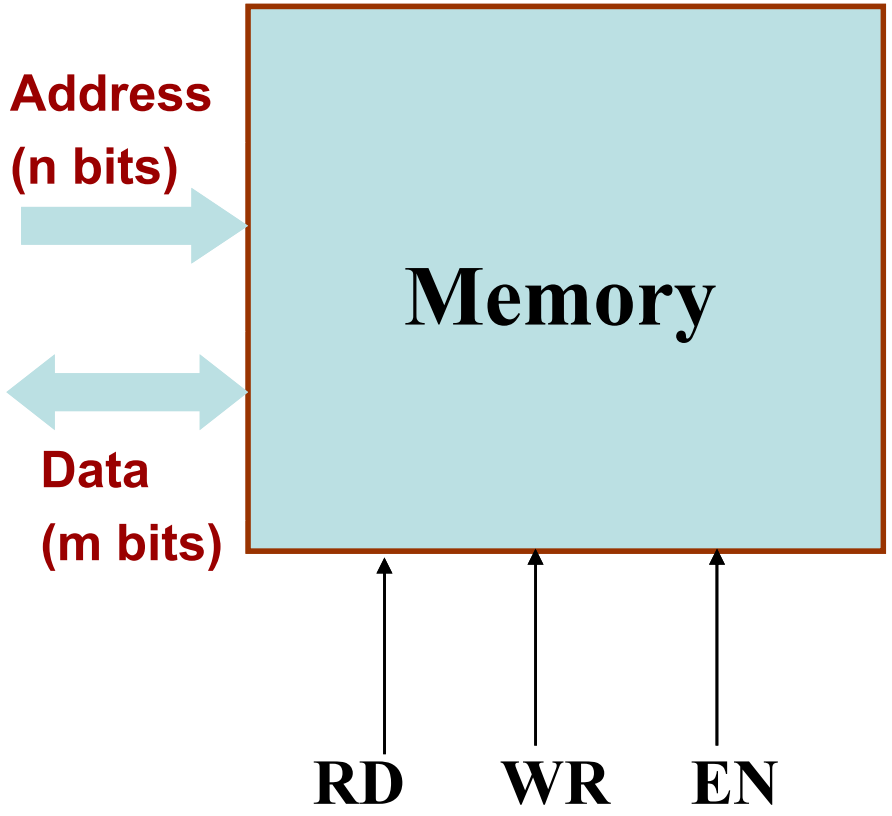
- With technological advancements, both processor and are becoming faster.
- However, the speed gap is steadily increasing.
- Performance gap grows exponentially. Although the disparity between microprocessor and memory speed is currently a problem, it will increase in the next few years. This increasing processor-memory performance gap is now the primary obstacle to improved computer system performance.

- Special techniques are thus needed to bridge this gap.
  - Cache memory
  - Memory interleaving



Memory Location

- If there are *n* bits in the address, the maximum number of storage locations can be 2^n,
  - For n=8, 256 memory locations
  - For n=16, 64k memory locations
  - For n=20, 1M memory locations
  - For n=32, 4G memory locations
- Now memory chips can store several Gigabits of data.
  - Dynamic RAM (DRAM)



- Memory data register will hold the content of that location.
- If it is 5, whatever content will be there in that particular location that will be present in MDR.

Memory Location

- **An 8 bit address represents 2^8 unique locations.**
- **First locations will be all 0s, and the last location will be all 1s**
- **Each of these locations again will have some content.**
- **Example: 2^8 x 16 memory, each of these locations will contains 16 bits data.**

Address	Contents
00000000	0000 0000 0000 0001
00000001	0000 0100 0101 0000
0000 0010	1010 1000 0000 0000
⋮	⋮
1111 1111	1011 0000 0000 1010

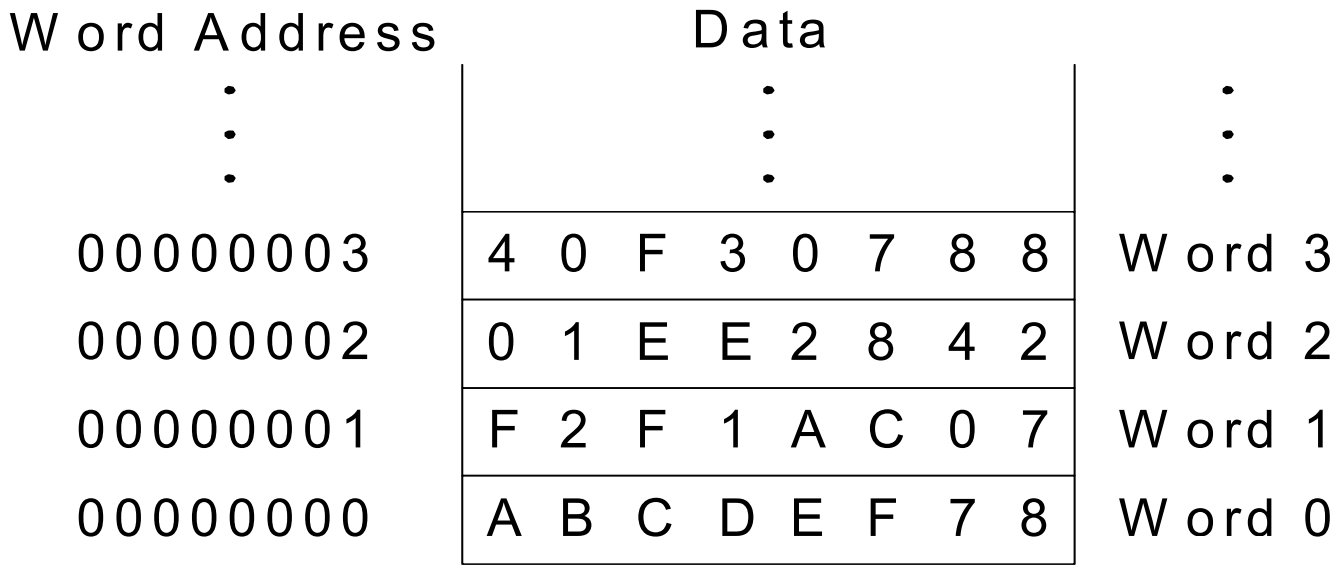
$2^8 \times 16$ memory

Memory: Example

- For a computer with 64 MB of byte addressable memory, number of bits are needed in the memory address (64 MB =) 2^{26} . 26 bits are needed to bits to represent the address.
- Example: A computer has 1 GB of memory and each word in this computer is 32 bit.
- 1 GB = 2^{30}. If each word is 32 bits, . So, total words possible will be 2^{30} / 4 = 2^{28}. So, we require 28 bits, with address from 0 to 2^{28}-1.
- If it is byte addressable, each byte can be accessed with address from 0 to 2^{30}-1.

Word-Addressable Memory

- Each 32-bit data word has a unique address

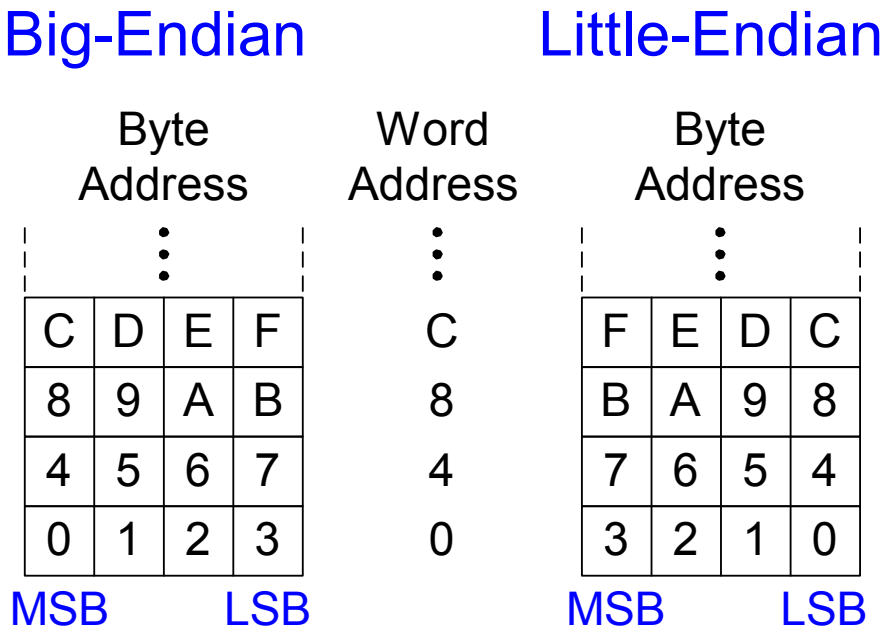


Words and Bytes

- 2^{32} bytes : byte addresses from 0 to 2^{32}-1
- 2^{30} words : byte addresses 0, 4, 8, ... 2^{32}-4

Big-Endian and Little-Endian Memory

- Word address is the same for big- or little-endian
- Little-endian: byte numbers start at the little (least significant) end
- Big-endian: byte numbers start at the big (most significant) end



Big Endian Addressing

- With Big Endian addressing, the byte binary address

  x . . . x00

  is in the most significant position (big end) of a 32 bit word (IBM, Motorolla, Sun, HP).

MSB			LSB
0	1	2	3
4	5	6	7

Little Endian Addressing

- With Little Endian addressing, the byte binary address

  x . . . x00

  is in the least significant position (little end) of a 32 bit word (DEC, Intel).

M S B			L S B
3	2	1	0
7	6	5	4

- Programmers/protocols should be careful when transferring binary data between Big Endian and Little Endian machines

Little and Big Endian Addressing: Example

- Represent the following 32-bit number in both Little Endian and Big-Endian in memory from 2000 onwards:

  01010101 00110011 00001111 11001100

	Big Endian	Little Endian
Address	Data	Data
2000	11001100	01010101
2001	00001111	00110011
2002	00110011	00001111
2003	01010101	11001100

Memory Access by Instructions

- **The program instructions and data are stored in memory.**
  - **In von-neumann architecture, they are stored in the same meory.**
  - **In Harvard architecture, they are stored in different memories.**

- **For executing the program, two basic operations are required.**

Load- The contents of a specified memory location is read into processor register.

LOAD R1, 8000

Store: The contents of a processor register is written into a specified memory location.

STORE 8000, R1

Memory Access by Instructions:Example

Compute X=(A+B)-(C-D)

```
LOAD R1, A
LOAD R2, B
ADD R3,R1,R2
LOAD R1, C
LOAD R2, D
SUB R4,R1,R2
SUB R3,R3,R4
STORE X,S3
```

Machine, Assembly and High level Language

- **Machine Language**
  - **Closure to the processor executed directly by hardware**
  - **Instructions consist of binary bits: 1's and 0'1**
- **Assembly Language**
  - **Low level symbolic version of machine language**
  - **One to one correspondence to the machine language.**
  - **Pseudo Instruction known as mnemonic are used that are more readable and easy to use.**
- **High level language**
  - **Programming languages like C, C++, Java etc.**
  - **More readable and closure to human language.**

Assembler and Compilers

- **Assembler**
- **Translates an assemly language program to machine language**

Assemblers and Compilers

- Assembler
  - Translates an assembly language program to machine language.
- Compiler
  - Translate a high-level language programs to assembly/machine language.
  - The translation is done by the compiler directly, or
  - The compiler first translates to assembly language and then the assembler converts it to machine code.

Assembler and Compilers

- **Assembler**
  - **Translates an assembly language program to machine language.**

Assembly Code

Field Values

Assembly Code	op	rs	rt	rd	shamt	funct
add $s0, $s1, $s2	0	17	18	16	0	32
sub $t0, $t3, $t5	0	11	13	8	0	34
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Order of registers in the assembly code:

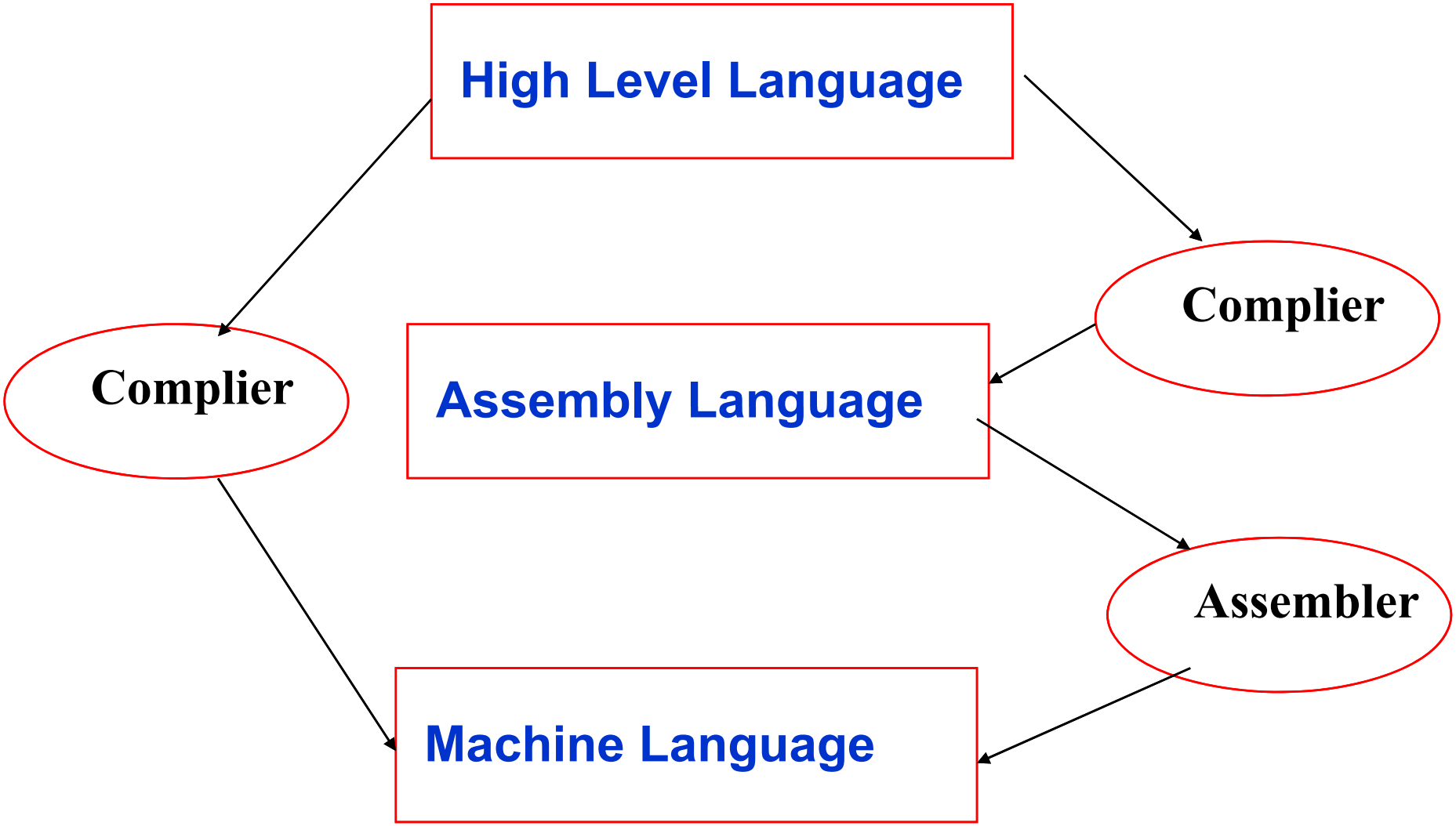
add rd, rs, rt

- Instructions are 32 bits long, registers have numbers 0 .. 31, e.g., $t0=8, $t1=9, $s0=16, $s1=17 etc.

Assembler and Compilers

- **Compiler**
  - **Translates a high language program to assembly/machine language**
  - **Translation is done by the complier directly or**
  - **Complier first translates to assembly language and then assembler converts it to machine code.**

Assembler and Compilers



Software abstraction

Higher Level Language Program (C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Compiler

Assembly Language Program (For MIPS)

```
swap:
        muli  $2, $5,4
        add   $2, $4,$2
        lw    $15, 0($2)
        lw    $16, 4($2)
        sw    $16, 0($2)
        sw    $15, 4($2)
        jr    $31
```

Assembler

Binary Language Program (For MIPS)

```
00000000101000010000000000011000
00000000000110000011000000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```