COMPUTER ORGANIZATION AND ARCHITECTURE (IT 2202)

Lecture 3-4

Basic Operation Of A Computer

- We will discuss the basic mechanism through which an instruction gets executed.
- ALU has different kind of registers
- General purpose registers
- Special purpose registers
- Temporary Registers

We will discuss the function of these registers.

Interfacing with Primary Memory

- Instructions and data are stored in memory. From memory, the data and the instruction are brought to the processors and then instruction will be executed.
- For this purpose, we require two special purpose registers
  - Memory Address Register (MAR)
  - Memory Data Register (MDR)
- Memory address register (MAR) holds the address of a memory location to be accessed. MAR holds the address of memory location that is to be accessed. We can access a memory location for reading an instruction or we can access a memory location for reading a data and we can also access a memory location for writing back the data. MAR holds the address of the instruction that is to be read or it holds the address of the data that is to be read from the memory or the address of the memory where the data is to be written.

Interfacing with Primary Memory

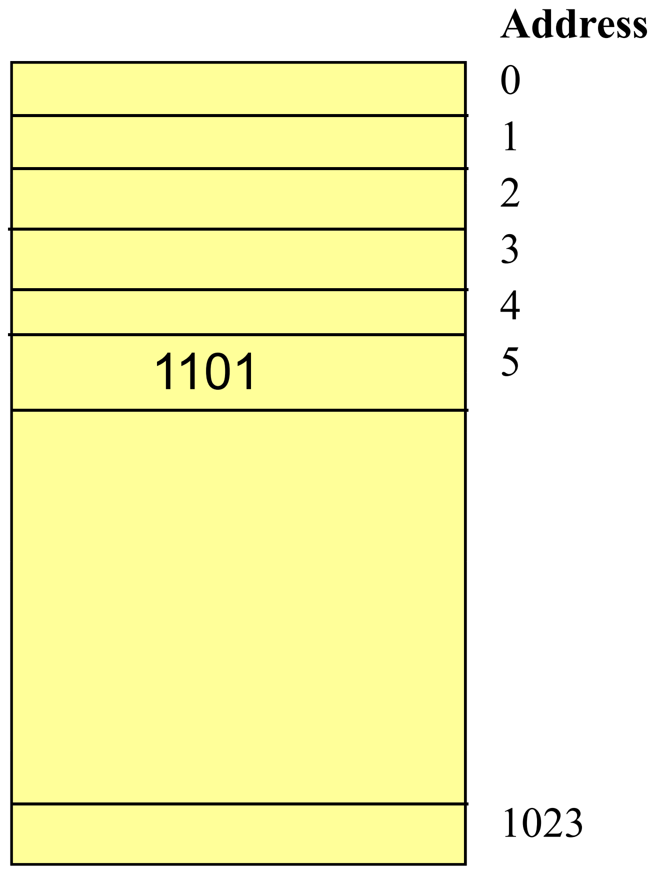
- Memory Data Register (MDR) holds the data that will be written into memory or the data that we will receive when read out from the memory location. When we write, we always write a data into the memory. But when we read, we can read an instruction or we can read a data.

- MDR will contain the instruction that is to be read from the memory or the data that is to be read from the memory or the data that is to be written into the memory. MDR contain any of these three.

- A memory is considered as a linear array of storage locations, bytes or words, each with a unique address.

Interfacing with Primary Memory

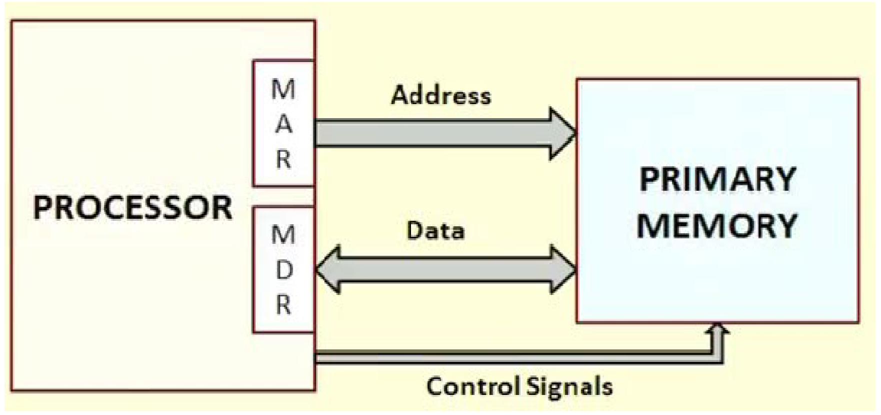
- Address is spanning from 0 to 1023.
- Number of memory locations that we can address is starting from 0 to 1023, are 1024.
- A memory is considered as a linear array of storage locations,.
- Each location which forms a unique address stores bytes or words.
- These address of the memory location starts from 0 and is incremented one by one and it has got 1024 locations.



- Memory address register will hold one of such address that is either 0, 1, 3,4 or 5 or anything
- 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.

Interfacing with Primary Memory

- This diagram shows the connection between processor and main memory.
- MAR is connected with primary memory through address bus.
- MDR is connected with primary memory through data bus. Some control signals are also required.



- When a particular data will be read from the memory or the particular data will be written into the memory, we need to specify the location from where, we need to read a data or in where, we need to write back data. These operations will be synchronized by control signals.
- First, we provide the address in the address bus that hits to the primary memory, then we provide the control signals either Read or Write depending on that, a particular value is read from that particular address and it comes to MDR.
- If we want to write a value, we have to put that value in MDR and the address where we want to write in MAR, and then we activate the write control signal. According to read or write control, a word is read or written from or into memory.

Interfacing with Primary Memory

- To read the data from memory, we first load the memory address into MAR then we issue the control signals i.e., read then the data from memory is read and it is stored into MDR.
- To write into memory, we have to load the memory address into MAR, the data that is to be written must be loaded into MDR and then we issue write control signal.

Handling Program/Instructions

- For this purpose, we have two special purpose registers.
- One is called Program Counter (PC) that holds the memory address of the next instruction to be executed. Automatically, it is incremented to point to the next instruction when an instruction is fetched and about to be executed. So, PC holds the memory address of the next instruction to be executed. So, once we read and execute an instruction, PC must point to the next instruction that we have to execute next and so on.

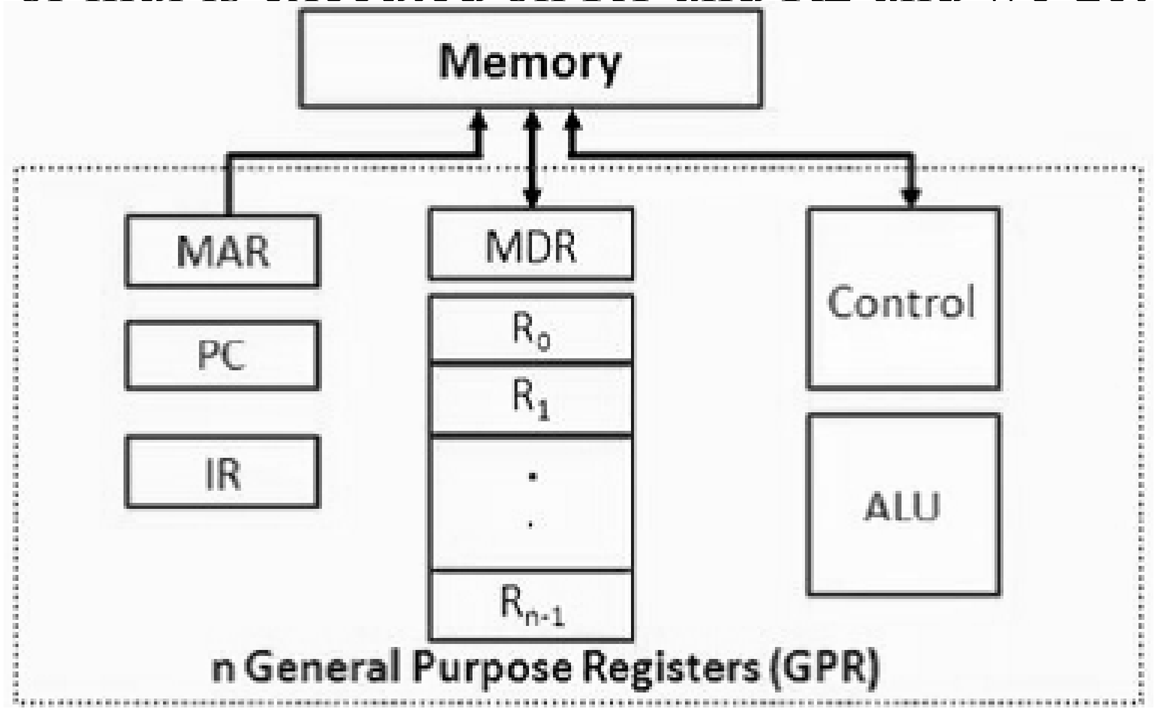
- Another register is the Instruction Register. Actually, PC will contain the location from where we have to read the instruction. Now, we have to read this instruction and store it somewhere. The register where it will get stored is known as Instruction Register. Suppose, according PC content, the location which contains ADD R1, R2 is identified, where this entire instruction ADD R1,R2 is fetched and will be stored in IR. So, IR temporarily holds an instruction that has been fetched from memory. Now, once an instruction is fetched from memory, we need to decode that instruction.

Handling Program/Instructions

- As per instruction, the computer will perform the activity.
- ADD R1,R2 instruction is saying that it adds the content of register R1 with R2, and store back in R1. Here, the instruction is decoded and then the computer understands it and then executes the instruction that adds the content of R1 and R2 and, store back the result in R1.
- The instruction register temporarily holds an instruction that has been fetched from memory and is decoded to find out the instruction type, which contains information about the location of the data.
- IR contains the entire instruction. After we get this instruction, it is decoded to know that this is add, and we also know locations of R1, R2 registers.

Architecture of Processor: Example

- The architecture of this processor shows how memory system is connected with the processor.
- The processor consists of special purpose registers: MAR, MDR, PC and IR.
- It contains some general purpose registers, control unit and ALU.
- In ADD R1,R2 instruction, R1 and R2 are registers that are present inside the processor.
- These registers (R1 and R2) are brought to ALU and then the particular instruction like add or mul is executed on R1 and R2 and we get the result.



Architecture of Processor: Example

- We shall illustrate the process of instruction execution with the help of the following two instructions:-

- ADD R1, MEM

  Add the contents of memory location (MEM) (i.e., address of the memory location is MEM) to the contents of The processor consists of register R1.

  $R1 \leftarrow R1+memory[MEM]$

- ADD R1,R2

  Add the contents of register R2 to the contents of register R1.

  $R1 \leftarrow R1+R2$

Architecture of Processor: Example

- **We will explain the execution of a instruction say, ADD R1, 5000.**
- First, this instruction is stored in some memory location. We assume here that the instruction is stored in location 1000, and the initial value of R1 is 50, and MEM value is 5000. Before the instruction is executed, PC contains the value 1000. Now, the content of PC is transferred to MAR as we need to read the instruction. To read the instruction from memory, the address needs to be loaded in MAR, and we need to activate the read control.
- Hence 1000 now should be transferred to MAR, then a read request is issued to memory unit. After the reading is performed, the instruction is in MDR; and then from MDR, it should be transferred to IR Instruction Register. And while doing these steps, we have to increment the PC to point to the next instruction.
- In computer, there will be sequential execution of instruction and the instructions are stored one by one. Here, first PC was pointing to 1000, we fetch the instruction from location 1000 and then the PC points to the next location that is 1001. Next, instruction read from memory is loaded in MDR and from MDR, it is transferred to IR. In IR, the instruction is decoded and then it is executed.

Architecture of Processor: Example

- First, PC value is transferred to MAR. Read signal is activated. The content specified by the memory location (MAR) is read and it is stored in MDR. From MDR, it is transferred to IR and at the same time PC is incremented to 4.

- In byte addressable organization, every time PC will be incremented by 4. Say, initial location is 2000, and the next location will be 2004 in case of 32-bit machine. The PC is incremented by 4. If it is a 64-bit machine, the PC will get incremented by 8.

- Now, once the instruction comes to IR, it needs to be decoded and executed.

- For example, ADD R1, 5000, after decoding, it has understood that one of the operand is in memory. 5000 is transferred from IR to MAR and this operand is read from memory location (5000) pointed by MAR and other operand will come from R1. It will be executed and addition of the content of R1 and content of memory location will be performed and the result will be saved in R1.

Micro-operations

The steps being carried out are called micro-operations

MAR ← PC

MDR ← MEM [MAR]

IR ← MDR

PC ← PC+4

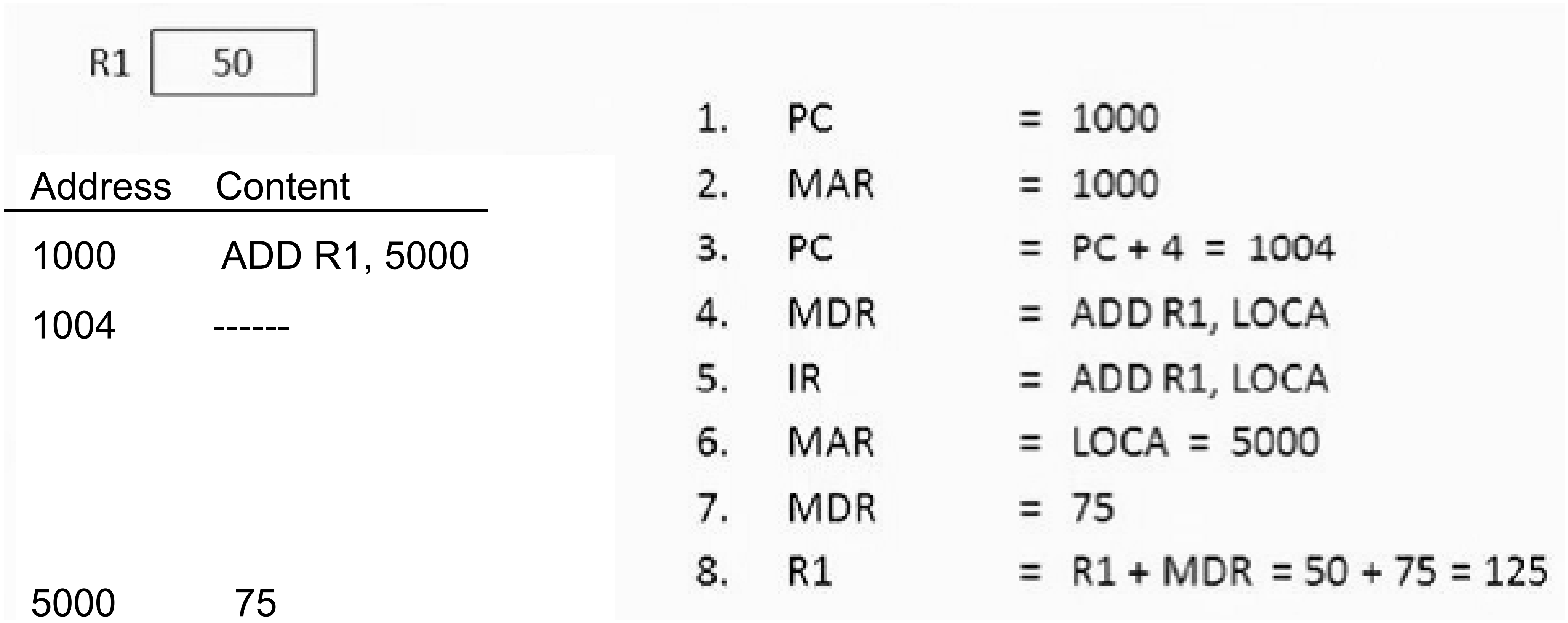
MAR ← IR [OPENAND]

MDR ← MEM[MAR]

R1 ← R1+MDR

Micro-operations: Example

Micro-operations Instruction ADD R1, 5000



R1 | 50 |

Address	Content
1000	ADD R1, 5000
1004	------
5000	75

1.	PC	=	1000
2.	MAR	=	1000
3.	PC	=	PC + 4 = 1004
4.	MDR	=	ADD R1, LOCA
5.	IR	=	ADD R1, LOCA
6.	MAR	=	LOCA = 5000
7.	MDR	=	75
8.	R1	=	R1 + MDR = 50 + 75 = 125

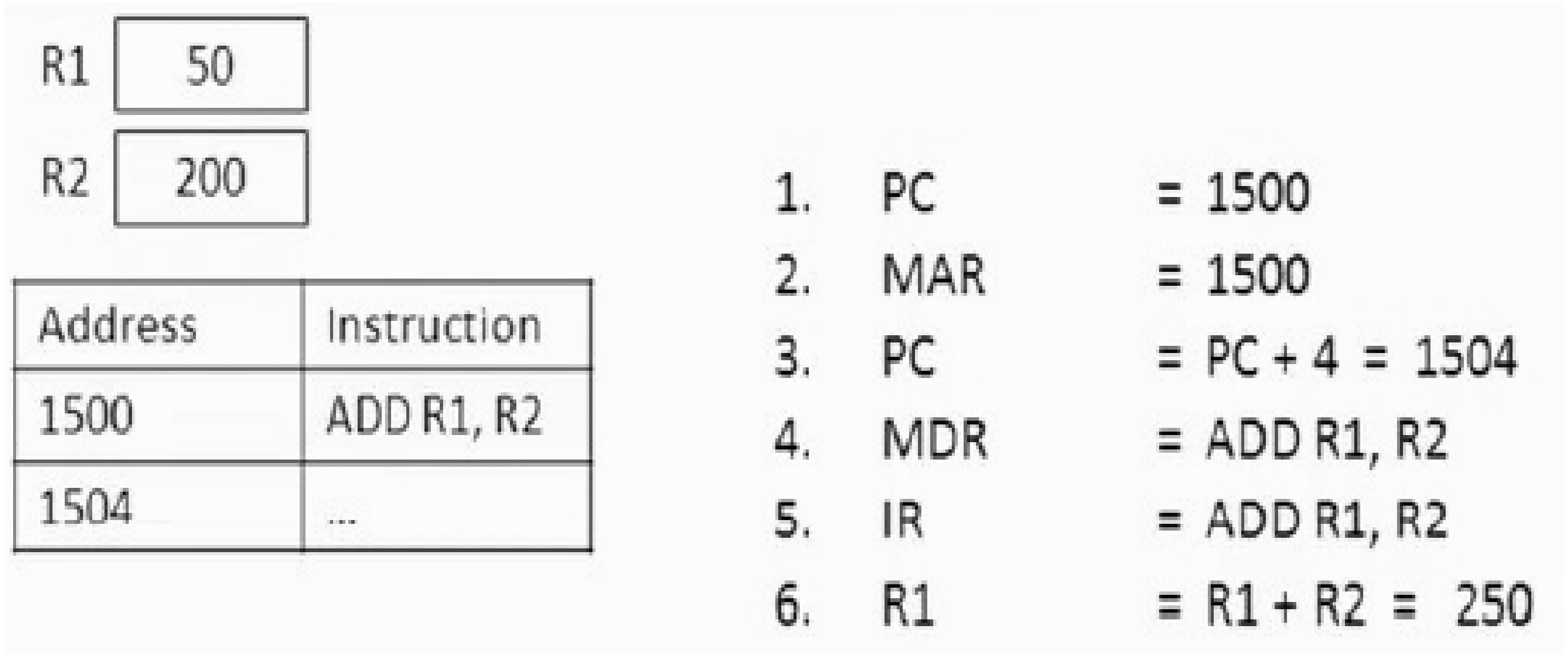
Micro-operations: Example

Micro-operations Instruction ADD R1, R2

• Suppose, ADD R1, R2 is saved in memory location 8000. Initial Value: R1=100 AND R2=150	
• Before instruction is executed, PC=8000	
• Content of PC is transferred to MAR.	**MAR ← PC**
• READ request is issued to memory.	
• Instruction is fetched to MDR.	**MDR ← MEM [MAR]**
• Content of MDR is transferred to IR.	**IR ← MDR**
• PC is incremented to point to the next instruction.	**PC ← PC+4**
• Instruction is decoded by the control unit.	
• R2 is added to r1.	**R1 ← R1+R2**

Micro-operations: Example

Micro-operations Instruction ADD R1, R2



R1	50
R2	200

Address	Instruction
1500	ADD R1, R2
1504	...

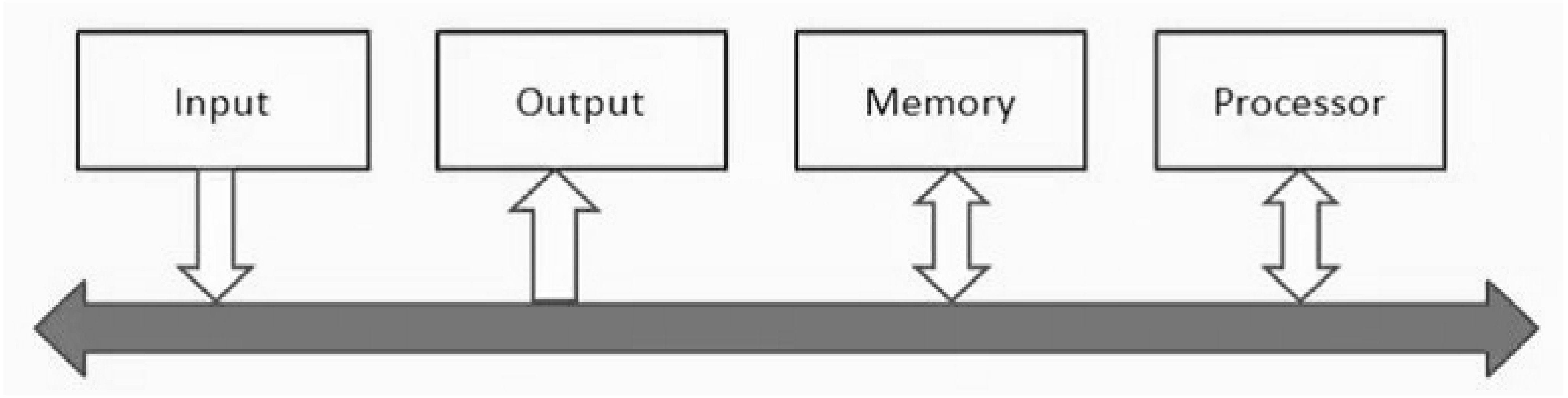
1. PC = 1500
2. MAR = 1500
3. PC = PC + 4 = 1504
4. MDR = ADD R1, R2
5. IR = ADD R1, R2
6. R1 = R1 + R2 = 250

Bus Architecture

- **Computer system consists of processor module, memory module, input output module.**
- **All these modules are communicating between each other.**
- **A communicating pathway is needed to communicate between each of these functional units. So, the different functional modules must be connected in an organized manner to form an operational system.**
- **Bus refers to a group of lines that serve as a connecting path for several devices. So, the simplest way to connect all these is through a single bus architecture, where only one data transfer will be allowed in one cycle.**
- **For multi bus architecture parallelism in data transfer is allowed.**

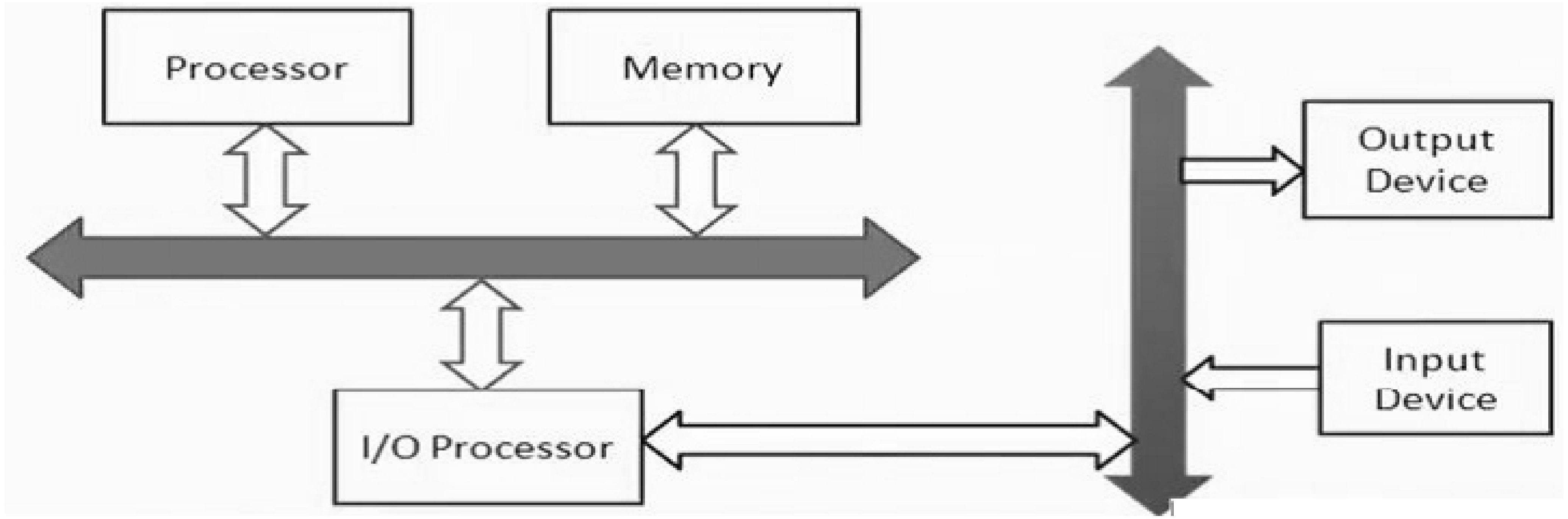
System-Level Single Bus Architecture

- **All the modules:-memory, processor, input and output are connected to a single bus. If the processor wants to communicate with memory, it has to use this bus and no other modules will be able to use at that moment.**
- **In the same way if from memory something needs to be sent to output device, no other module can communicate.**
- **This is a bottleneck of a single level system bus.**



System-Level Two-Bus Architecture

- System-level two-bus architecture
- There is a bus dedicated to processor memory and IO processor is also connected to it.
- For input and output, there is a separate bus and this bus will be communicating with the IO processor in turn and the IO processor will then be communicating with the processor or the memory as and when it is required. When there is more communication between processor and memory, then we must have a bus dedicated for this.

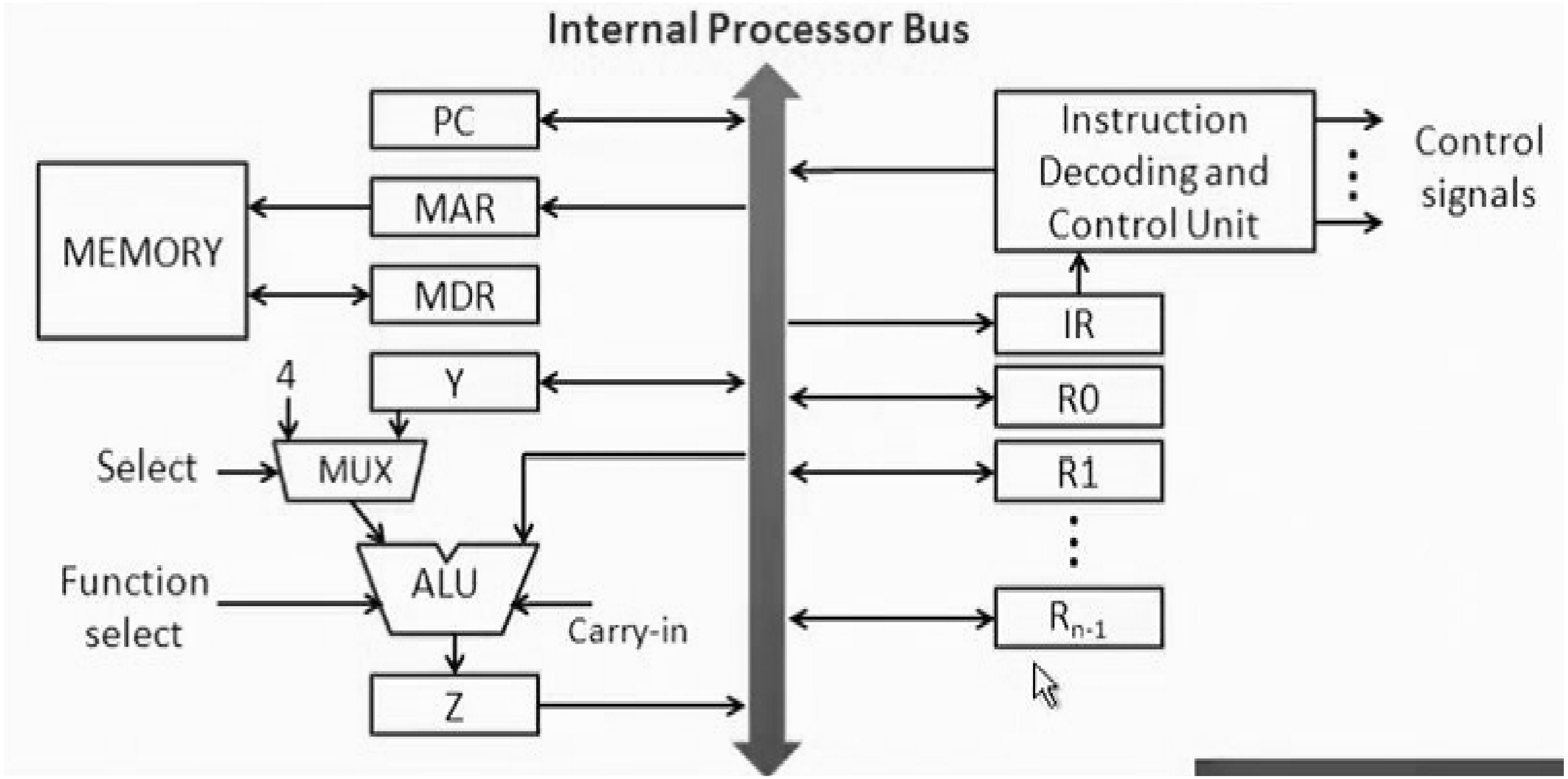


Single-Bus Architecture Inside the Processor

- There is a single bus inside the processor.
  - ALU and the registers are all connected via the single bus.
  - This bus is internal to the processor and should not be confused with the external bus that connects the processor to the memory and I/O devices.
- A typical single-bus processor architecture is shown on the next slide.
  - Two temporary registers *Y* and *Z* are also included.
  - Register *Y* temporarily holds one of the operands of the ALU.
  - Register *Z* temporarily holds the result of the ALU operation.
  - The multiplexer selects a constant operand 4 during execution of the micro-operation: $PC \leftarrow PC + 4$.

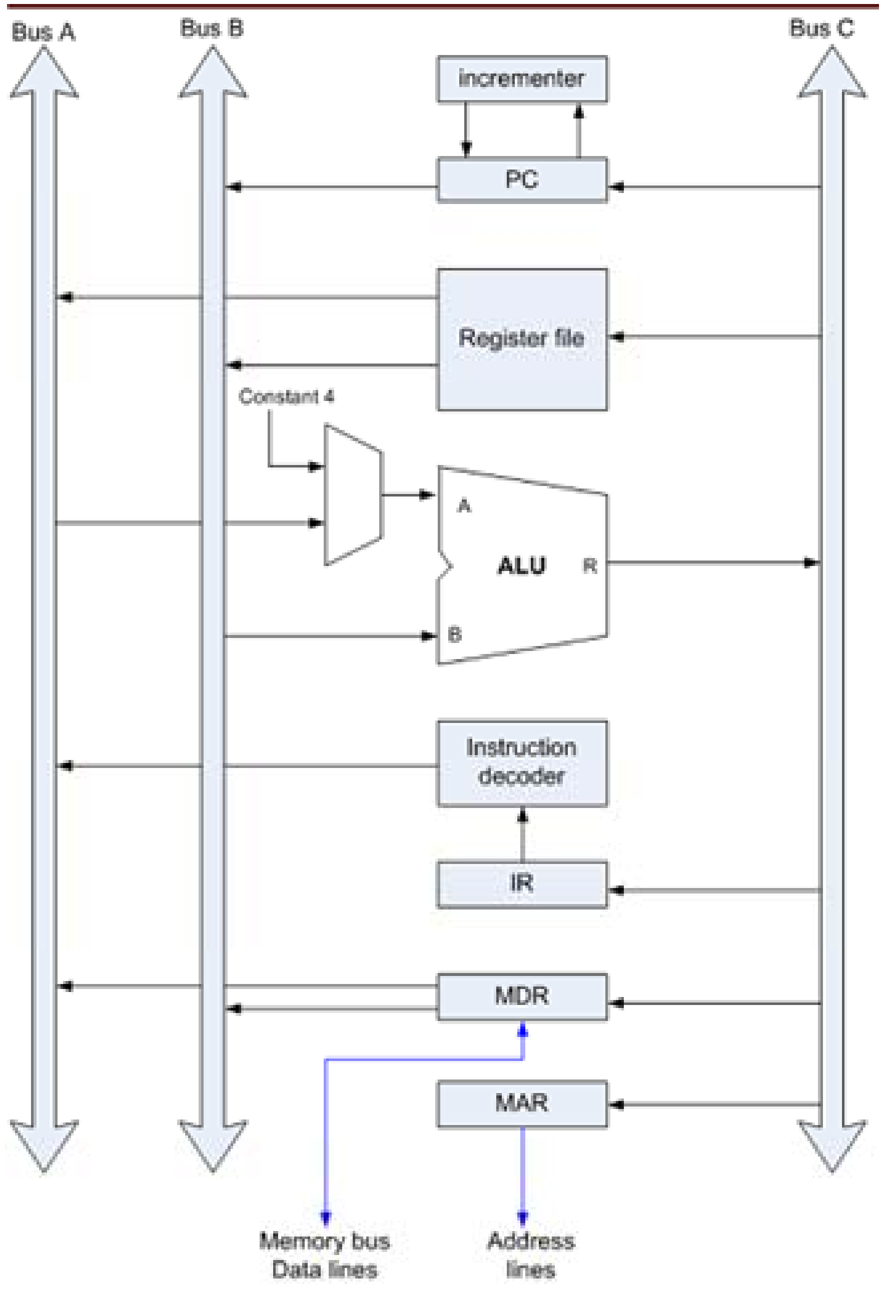
Single-Bus Architecture Inside the Processor

- This is an internal processor bus. All the elements: PC, MAR, MDR, Registers, ALU are inside the processor. Temporary registers Z, Y, general purpose registers, IR, instruction decoding and control unit are connected. Information is getting transferred within all these modules.
- A data transfer from R1 to R2 or from PC to MAR or from MDR to R1 or R2 or to ALU, will be done through this internal processes bus.
-



Multi-Bus Architecture

- Modern processors use the multi-bus architecture. Here, within the processor to communicate between various registers, we have multiple-bus.
- The advantage is that more operations can be performed and we get results much faster. So, there will be a overall improvement in instruction execution time.



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.

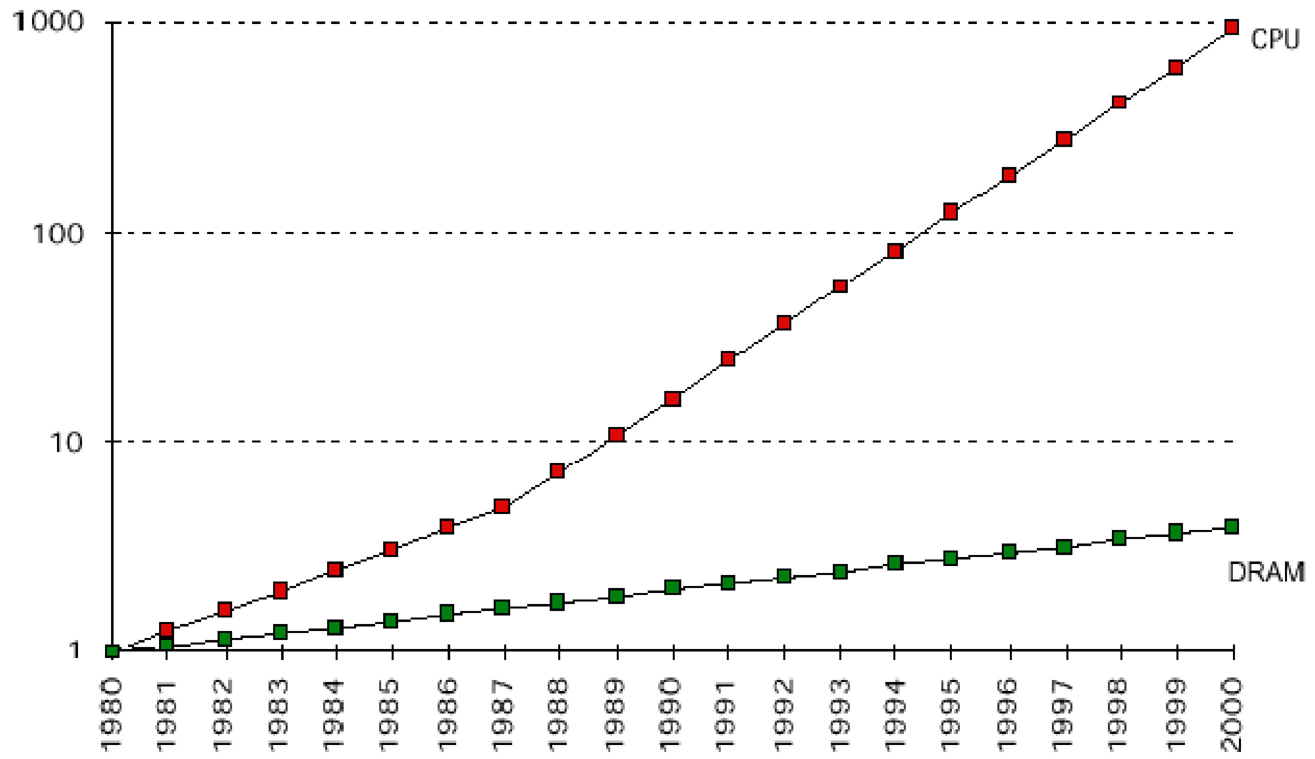
- Memory is often byte organized. Here, each byte is having an address, that means every byte of the memory has a unique address. Multiple bytes of a data can be accessed by an instruction.

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.

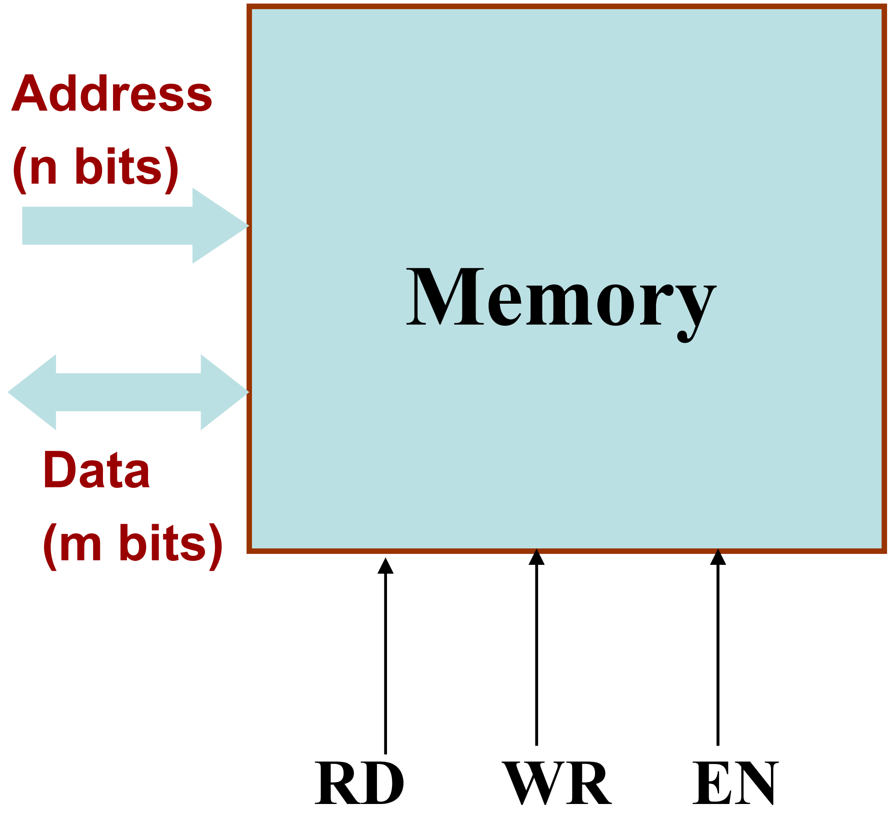
Processor- Memory Performance Gap

- With technological advancements, both processor and memory 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
- 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

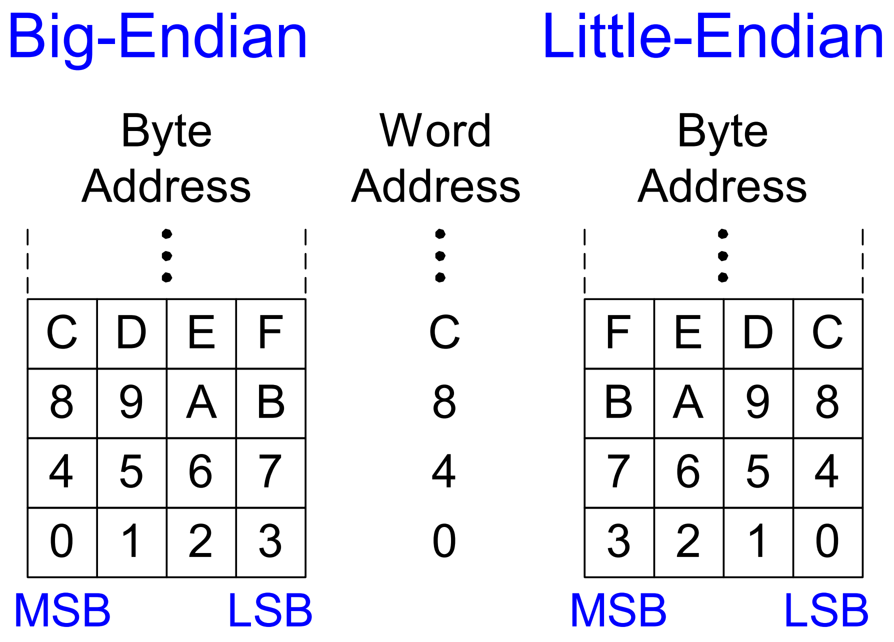
Word Address	Data	
⋮	⋮	⋮
00000003	4 0 F 3 0 7 8 8	Word 3
00000002	0 1 E E 2 8 4 2	Word 2
00000001	F 2 F 1 A C 0 7	Word 1
00000000	A B C D E F 7 8	Word 0

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 assembly language program to machine language.

Compiler

- Translate a high level language program 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

	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:

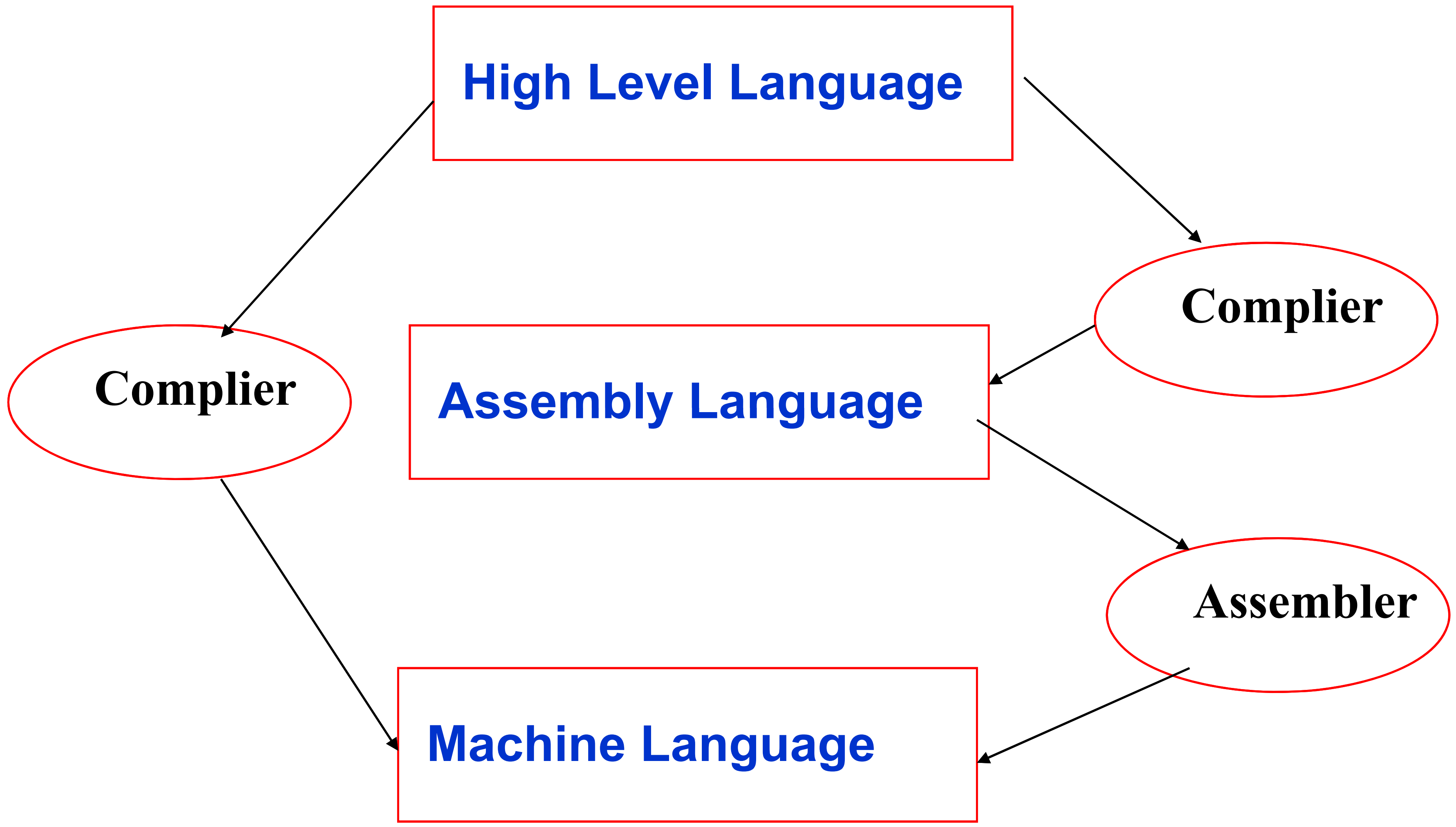
add rd, rs, rt

- 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;
}
```

$$\begin{tikzcd}\text{C program} \arrow[r] & \boxed{\text{Compiler}} \arrow[r] & \text{Assembly program} \arrow[d] \\ & \text{Binary program} & \boxed{\text{Assembler}} \arrow[l]\end{tikzcd}$$

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
```

Binary Language Program (For MIPS)

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```