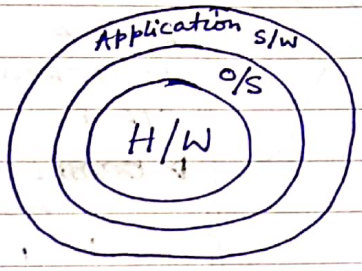
CO

Digital Computer Organization

In this course, we will try to find out that what are the resources available in a computer; specifically the H/W resources and how those resources are interconnected, or how the data paths, that are connected go from one resource to another resource to complete a particular task.

High level view of computer



So, whenever a prog is to be executed, that prog remains in the application s/w level. Now as the prog is to be executed, then it is the OS which gives an interface between the application s/w and the h/w resources of the computer. So, Basic job of the OS, which is in between the application s/w and the H/W, is to manage the H/W resources, so that, the same H/W resource can be used by one or more application level s/w s at the same time. In our course we will concentrate on the H/W resources, we will not talk much about the OS or application s/w.

What are the H/W resources that we will need in computer

Let us take a very very simple s/w

example —

```
int a,b,c;
scanf ("%d %d",&a,&b);
c = a+b;
printf ("%d",c);
```

now you know computer is not capable of executing this high level language prog. So, we have to compile the prog

and generate executable code and only the executable code can be executed by the computer.

Let us try to convert this piece of code into assembly level code →

```
int a, b, c;
scanf ("%d %d", &a, &b);
c = a+b;
printf ("%d", c);
```

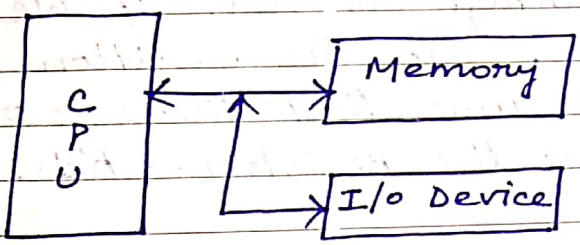
```
IN PORTA
STA A      → (store)
IN PORTA
STA B      → (store)
LDA A      → (load)
ADM B      → (Add memory)
STA C      → (store)
OUT PORTB.
```

so the content of the memory location B has to be added with A.
And we assume after addition the result will be available at ACC → accumulator.

- These line of codes in assembly level language points to the resources which are needed in the CPU of a computer or in a computer system as a whole. The first line → **IN PORTA** → for this in operation the data is read from portA and it is saved in the ACCUMULATOR. i.e. the CPU of the computer must have a registor called an **ACCUMULATOR**. The second line → **STA A** → similarly we are saving the data to a memory location which is allocated to this variable A. So, which again tells we have to have some **memory associated** with the computer. and along with this registor in this particular case it is ACCUMULATOR and the memory unit, we also have

some I/P, O/P devices. Because you find that the values of the variables are input from a port, similarly the O/P of C is output to another port. So, as a whole in a computer we must have a CPU – which executes the program, we must have atleast one registor (later we will see 1 registor may not be sufficient) – accumulator which is a special purpose registor, and we also have to have some I/P – O/P devices.

So, in a high level view →



Now the connections between the CPU to memory is shown as bidirectional. Because, the data may come from the memory to CPU, in case of reading a value from memory location, the data may also move from the CPU to memory, in the case of writing a data from the CPU to memory location. Similarly, if you want to read from an I/P device, then the data moves from the I/P device to the CPU, and if you want to write a data to the O/P device – the data moves from CPU to the O/P device.

To have an overview about the components of the CPU or the resources available to CPU, let us go back to the programing example.

Now We know, initially the data is read from I/P device or PORTA and saved in ACCUMULATOR – so this is one of the registors.

And next operation we have performed is — from the ACCUMULATOR we have written the data to a memory location A — allocated to the variable A.
Likewise the next operation data is read from PORTA and it goes to the ACCUMULATOR, then again from the ACCUMULATOR the data is written to registor B. Now, we have to perform these write back operations to the memory because we have assumed that we have only a single special purpose registor in the CPU till now. Let us see what advantage we have if we have more than one registor. So, in addition to the ACCUMULATOR if we assume that there is one more registor — Let us name the Registor — R1. then —

```
IN PORTA
MOV R1, ACC
```
→ the data is moved from ACC to the Registor R1, R1 is an additional registor with in the CPU.

NOW when our value of A is stored in R1 then we dont need to move the value of B from the ACC. So, what will happen —


```
IN PORTA
MOV R1, ACC
[redacted]
IN PORTA
ADD R1
```

Because the value of B is already available in the ACC which is done by this 2nd I/P statement. The instruction count is reduced by having one more registor in CPU.

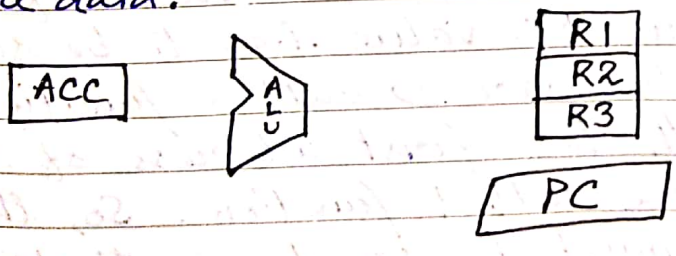
2nd advantage is that we dont need to move, or write the data from CPU to memory but we are just moving it from ACC to R1. ACC and R1 both are situated in CPU. So, it is taking much less time.

3rd Advantage is that if we dont want to use the value of C in some other place then we can discard the line STAC. Because ADD R1 results the O/P in the ACC itself. So, the code will be —

```
IN PORTA
MOV R1, ACC
IN PORTA
ADD R1
OUT PORTB
```

which is more efficient than the previous code.

- NOW, we have two different operations for addition. Previously it was ADMB and then ADD R1. So, this particular unit which will perform the addition operation or the logical operation is called ALU (Arithmetic Logical Unit).
- So, In a CPU we must have an ALU, an ACC, some more registors or general purpose registors for storage of the data.



[The no of general purpose registors can be increased. So, we can write more efficient progs. But it must have a limit]

- Let us take the programing Example again. In this prog when an instruction is executed, the CPU must know

2013

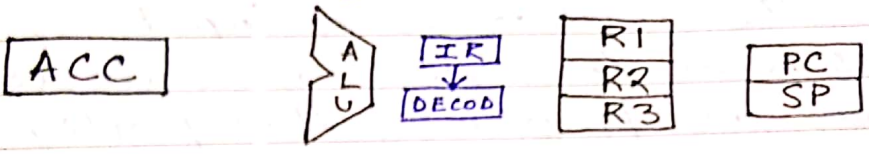
which is the next instruction to be executed. Now how these instructions are stored in the memory? Each of the instruction will be converted to a m/c. language code which is a binary number, sometimes it is represented as a hexadecimal number. Those instructions will be read or m/c level instructions will be read from the memory, then the instructions will be decoded and corresponding action will be taken.

Once a particular instruction will be executed, then the next instruction has to be executed. So, I must have a resource which acts as a pointer to the next instruction, I must execute. So, that particular resource in the CPU is called a PROGRAM COUNTER

I may need some more registors. Say for example, when you execute a program and if you write a structured program, in that case you may find for execution of a particular task, you have to call a function. So, that function can be written as the module of a s/w. And when you call any particular function, all the previous intermidiate results must be saved somewhere. Similarly, if I call a function in that case the previous PC value has to be saved somewhere because whenever I call a function the PC is loaded with the first address of the first instruction of the called function. So, the previous value of PC must be saved, so that when execution of a particular function is complete and I come back to the calling function I must able to

restore the value of PC to complete the whole execution. So, to save all these informations I must have some space. These informations are stored in some memory location known as STACKS and the addresses of those stacks are given by another registors — STACK POINTER (SP)

Now we can say CPU is like this →



Now another important part of the computer is instruction interpretation. Before execution of the instruction the CPU must know that what that instruction is supposed to perform, i.e. the CPU has to decode the instruction then the OPCODE - the code of the instruction which is the m/c level code has to be read from the memory and it is put in another registor in the CPU — called Instruction registor (IR).

Now from this IR the instruction goes to an Instruction Decoder — which decodes the instruction and after that only the CPU knows what this instruction is supposed to do.

Let us take a very simple Example →

MOV R1, R2 $R_1 \leftarrow R_2$

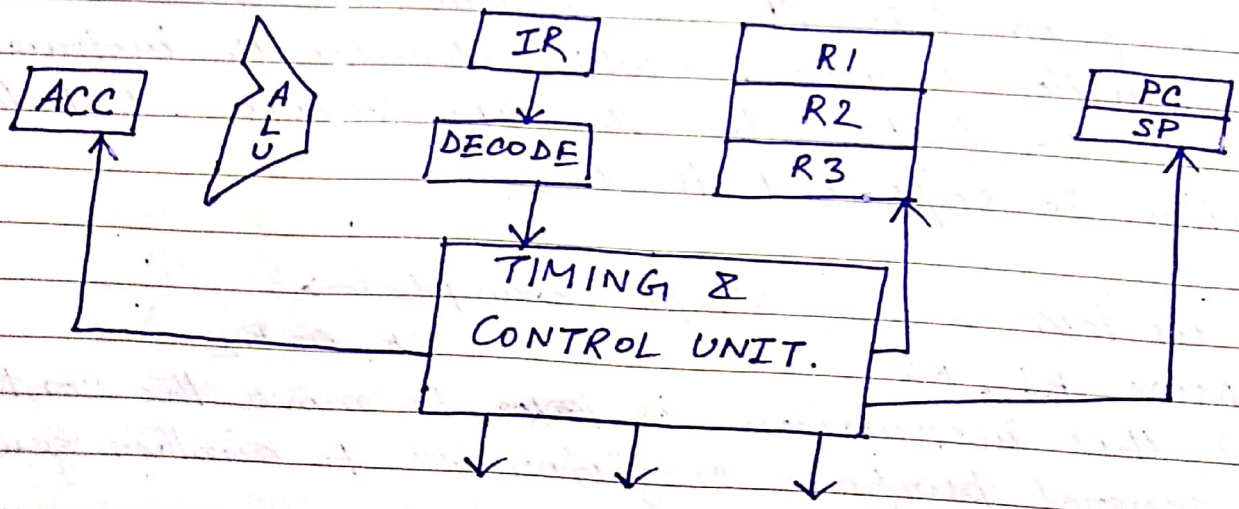
job of this instruction is to move the content of a general purpose registor R2 to another general purpose registor R1.

SEPTEMBER
OCTOBER
NOVEMBER
2013

For performing this particular operation this registor R1 must have a control input called Load or Latch. and R2 must have a control input called output enable. So, when this O/P enable of registor R2 is active that data from registor R2 is ready to be transferred to some other destination and when you make the Latch or Load input of R1 active — R1 is ready to take any available input.

So, for performing this particular operation I have to activate the O/P of R2 and at the same time I have to activate the Load I/P of R1. These two are called the control signals. So, all these registors will have different such control signals and these control signals are to be generated in proper way so that the specified task can be executed.

So, in CPU we must have another unit known as timing and control unit.



So, this timing and control unit generates all the control signals in a proper sequence so, that a specified task can be executed.

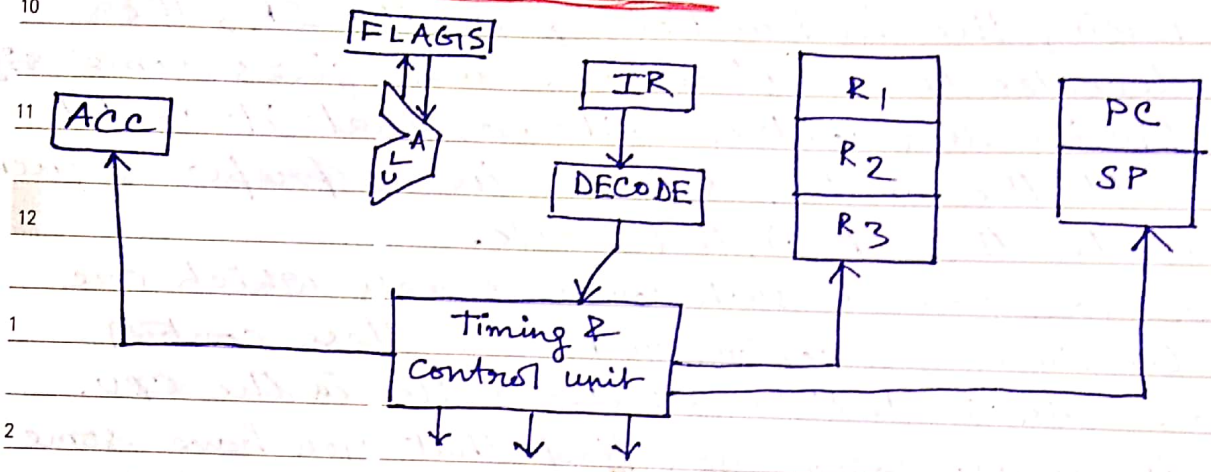
After taking the instruction from the IR, the decoder decodes the instruction and gives some signa to the timing and control unit so, that it could generate all the control signals in a proper sequence to execute the specified task.

So, there are various such control signals which are generated by timing and control unit and these control signals are distributed to all other units in the CPU.

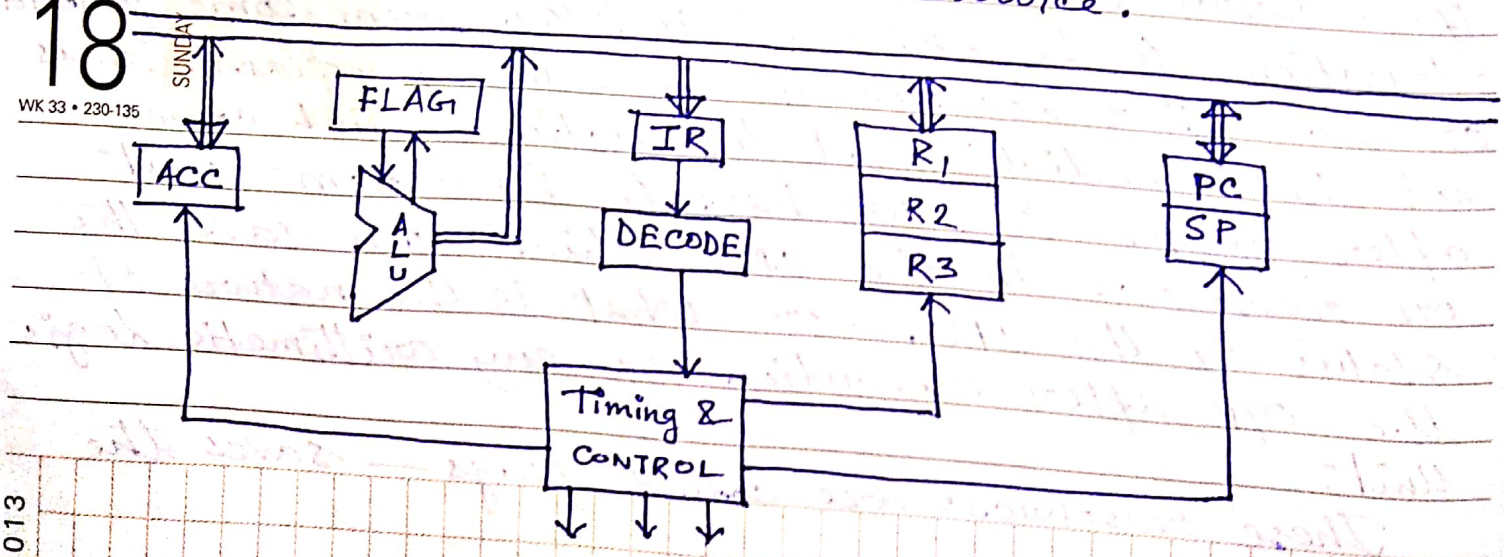
Now, in many cases you find that you have some special instructions or jump instructions — used to decide that whether to take a jump to some other instruction or whether to call a subroutine — these decision depends upon the result of computation. Say for example we give one instruction like jump with carry. So, after doing some arithmetic or logical operation if a carry is generated in that case the next instruction to be executed is taken from some function or some other location in the same program. It is not the immediate next instruction which comes after jump. So, we have to have some unit or resource in the CPU which can save the status of the CPU — or what is the nature of the o/p after execution of any arithmetic logic unit.

These resources are called flags — saves the

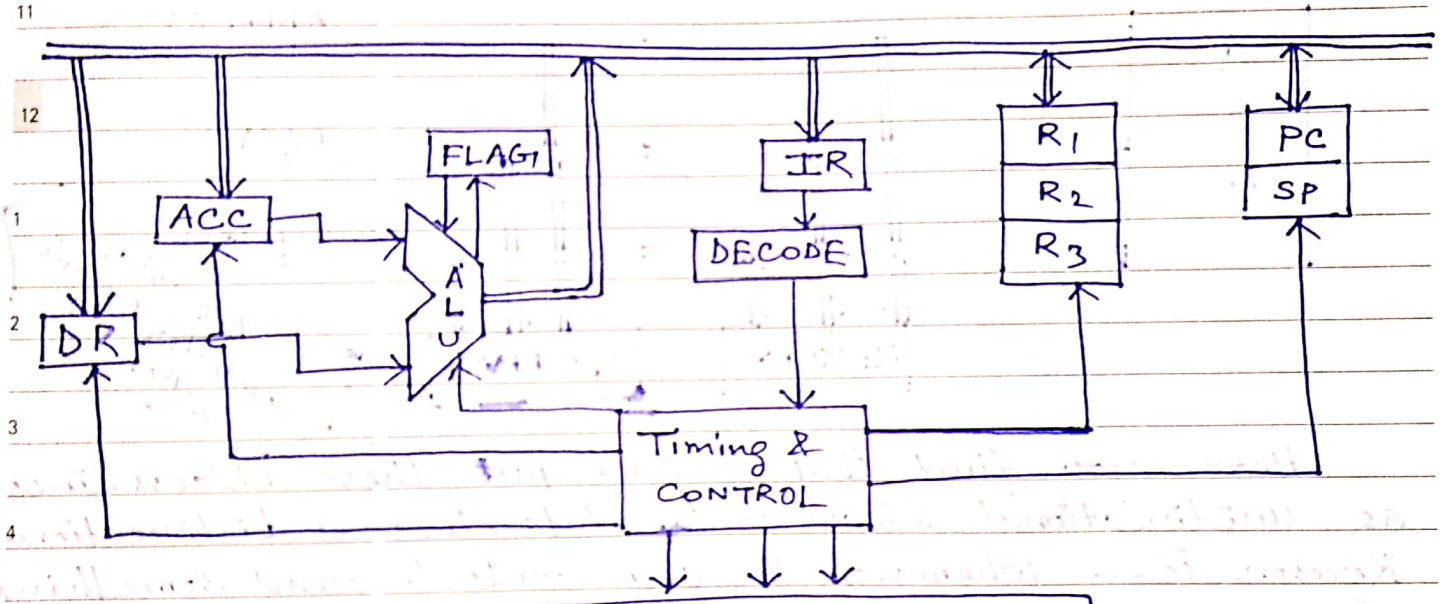
the status of execution of some arithmetic or logical operations. These also sometimes known as — Processor Status Word.



Now these are the resources that we need in a CPU for execution of any particular task. In addition to the resources we also must have some agent which can move the data from one resource to another resource. So, I have to have a data path, using which the data can be moved from resource to resource.



Now coming back to this ALU, it needs two inputs for operations in most of the cases. So, one of the I/P to ALU comes from the ACC, and the other I/P to the ALU comes from Data Registor (DR). This DR should also have a connection to the Data path.

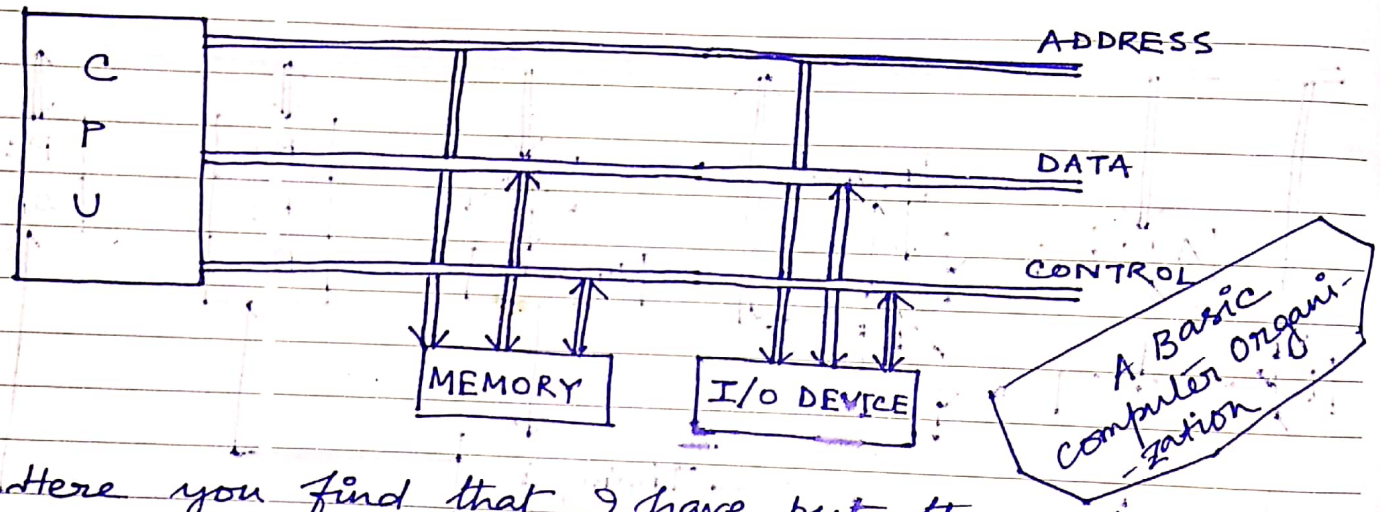


INTERNAL ARCHITECTURE OF CPU — INTEL 8085

Now next part which comes is that once we have this internal architecture of CPU and if I take this CPU as a Block, then what I have to see is how this CPU has to be interfaced with the external Memory and I/O devices.

For that some of the interfacing signals which are taken out of the CPU. So, if we represent this CPU by a single block it will have a no of pins using which we can interface the CPU with external

memory and I/O devices. A set of such pins are called the Address Lines or Address Bus, another is called Data Lines or Data Bus. And we have some more pins which actually generates the control signals known as CONTROL BUS.



Here you find that I have put these address lines as unidirectional whereas the data lines as bidirectional. Reason is — whenever the CPU wants to read something from any memory location, it is the responsibility of the CPU to specify the Address of the memory location from where the CPU wants to read the data. Same case for writing also. for I/O devices we can understand. Same logic works.

For data lines it must be bidirectional because sometimes we read data from memory or I/O device to CPU and sometimes we write data to memory or I/O device from CPU.

Now, coming to the control lines that we would like to have in case of CPU's control signal generation You find that we have two basic operations that have to be performed with the memory or with the I/O devices — Read & Write.

Now, to identify what kind of operation is to be performed wheather it is READ operation or Write operation, the CPU has to generate control signals. For Read the signal generated is RD. In most of the cases it comes in '-'ve logic → $\overline{RD}$.

For Write → $\overline{WR}$.

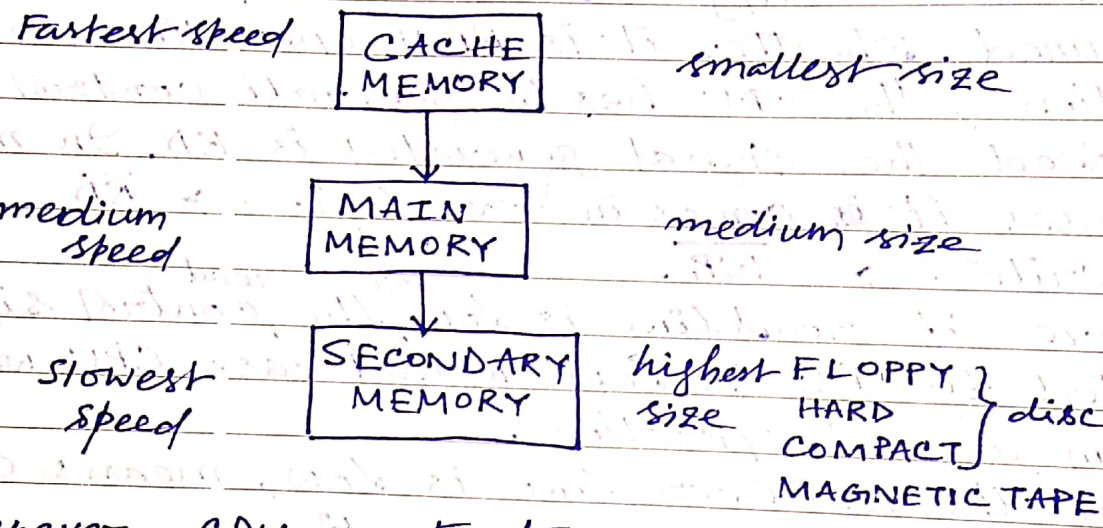
Here if read line is low, or the read control signal generated by CPU is low in that case CPU wants to perform a read operation.

If the write Bar Line is low, means CPU wants to perform a write operation.

Obviously both $\overline{RD}$ & $\overline{WR}$ cannot be low simaltaniously because in that case what operation is to be performed will not be defined. So, which one among these two signal is to be low will be decided by the CPU and it depends upon the instruction i.e. going to be executed.

So, as we have said that in case of CPU we have a timing and control unit which will generate the corresponding control signals after it gets an info from the decoder.

Now, coming to the Memory, we know that we have various kinds of Memory — cache, Primary, Secondary memory. So, this memory is actually a hierarchy of different types of Memory.



Whenever CPU wants to access something it first tries to find it in the cache memory. If it is not available in the cache memory then CPU tries to find it out in Main Memory.

Now this cache memory and main memory both are directly accessed by CPU. But the CPU does not have the direct access to the secondary memory. If the data is neither available in cache memory nor in the main memory then data has to be read from the secondary memory and then it has to be put in main memory or cache memory and then only the CPU can access it.

Now, sizewise the cache memory is the smallest among all of them. But it is fastest — just like you already know about the memory hierarchy.

So, we find the nature of access of these different kinds of memory are very much different. because the CPU has direct access over the cache or main memory. So, from these two memories the data can be accessed or written byte by byte. Whereas for secondary memory, it is accessed or written as a block.