**DAR ES SALAAM INSTITUTE OF TECHNOLOGY**
**DEPARTMENT OF COMPUTER STUDIES**
**COU07302 MICROPROCESSOR AND COMPUTER ARCHITECTURE**
# Lecture 5 - Introduction to CPU Design
**by**
**E. Kondela**

The operation or task that must perform by CPU are:

- Fetch Instruction: The CPU reads an instruction from memory.
- Interprete Instruction: The instruction is decoded to determine what action is required.
- Fetch Data: The execution of an instruction may require reading data from memory or I/O module.
- Process data: The execution of an instruction may require performing some arithmatic or logical operation on data.
- Write data: The result of an execution may require writing data to memory or an I/O module.

To do these tasks, it should be clear that the CPU needs to store some data temporarily. It must remember the location of the last instruction so that it can know where to get the next instruction. It needs to store instructions and data temporarily while an instruction is beign executed. In other words, the CPU needs a small internal memory.
These storage location are generally referred as registers.
The major components of the CPU are an arithmatic and logic unit (ALU) and a control unit (CU). The ALU does the actual computation or processing of data. The CU controls the movement of data and instruction into and out of the CPU and controls the operation of the ALU.

The CPU is connected to the rest of the system through system bus. Through system bus, data or information gets transferred between the CPU and the other component of the system. The system bus may have three components:

**Data Bus**:
Data bus is used to transfer the data between main memory and CPU.

**Address Bus:**
Address bus is used to access a particular memory location by putting the address of the memory location.

**Control Bus:**
Control bus is used to provide the different control signal generated by CPU to different part of the system. As for example, memory read is a signal generated by CPU to indicate that a memory read operation has to be performed. Through control bus this signal is transferred to memory module to indicate the required operation.

There are three basic components of CPU: register bank, ALU and Control Unit. There are several data movements between these units and for that an internal CPU bus is used. Internal CPU bus is needed to transfer data between the various registers and the ALU, because the ALU in fact operates only on data in the internal CPU memory.

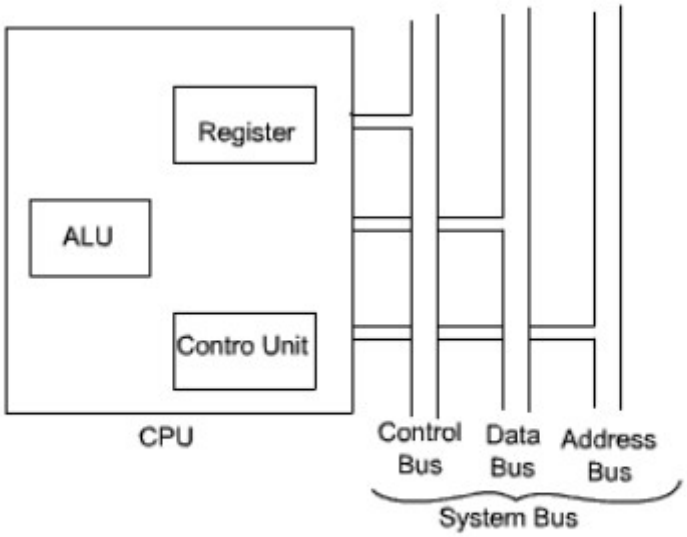



**Figure A - 1 :** CPU with the system Bus

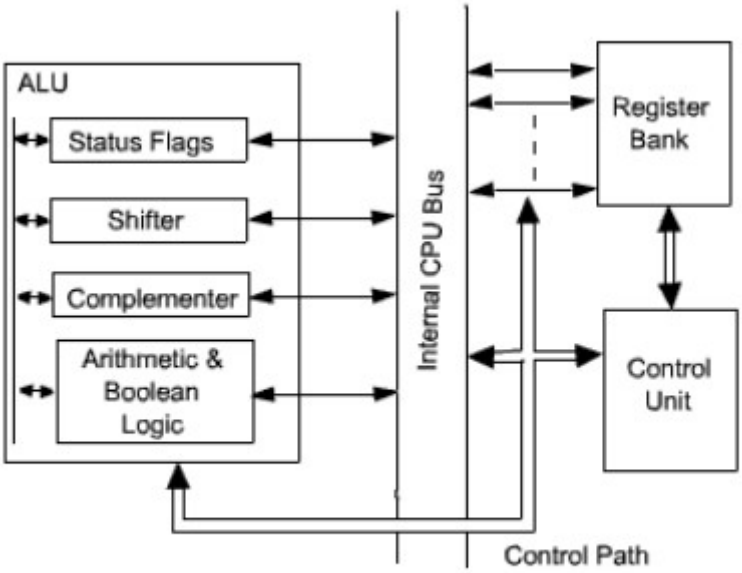



**Figure A - 2 :** Internal Structure of the CPU

**Register Organization**

A computer system employs a memory hierarchy. At the highest level of hierarchy, memory is faster, smaller and
more expensive. Within the CPU, there is a set of registers which can be treated as a memory in the highest level of
hierarchy. The registers in the CPU can be categorized into two groups:

- User-visible registers: These enables the machine - or assembly-language programmer to minimize main memory reference by optimizing use of registers.
- Control and status registers: These are used by the control unit to control the operation of the CPU.

Operating system programs may also use these in privileged mode to control the execution of program.

User-visible Registers:
The user-visible registers can be categorized as follows:

- General Purpose Registers
- Data Registers
- Address Registers
- Condition Codes

**General-purpose registers** can be assigned to a variety of functions by the programmer. In some cases, general-purpose registers can be used for addressing functions (e.g., register indirect, displacement). In other cases, there is a partial or clean separation between data registers and address registers.
**Data registers** may be used to hold only data and cannot be employed in the calculation of an operand address.
**Address registers** may be somewhat general purpose, or they may be devoted to a particular addressing mode.
Examples include the following:

- Segment pointer: In a machine with segment addressing, a segment register holds the address of the base of the segment. There may be multiple registers, one for the code segment and one for the data segment.
- Index registers: These are used for indexed addressing and may be autoindexed.
- Stack pointer: If there is user visible stack addressing, then typically the stack is in memory and there is a dedicated register that points to the top of the stack.

**Condition Codes** (also referred to as flags) are bits set by the CPU hardware as the result of the operations. For example, an arithmatic operation may produce a positive, negative, zero or overflow result. In addition to the result itself beign stored in a register or memory, a condition code is also set. The code may be subsequently be tested as part of a condition branch operation. Condition code bits are collected into one or more registers.

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

Different machines will have different register organizations and use different terminology. We will discuss here the most commonly used registers which are part of most of the machines.

Four registers are essential to instruction execution:

**Program Counter (PC):** Contains the address of an instruction to be fetched. Typically, the PC is updated by the CPU after each instruction fetched so that it always points to the next instruction to be executed. A branch or skip instruction will also modify the contents of the PC.
**Instruction Register (IR):** Contains the instruction most recently fetched. The fetched instruction is loaded into an IR, where the opcode and operand specifiers are analyzed.
**Memory Address Register (MAR):** Containts the address of a location of main memory from where information has to be fetched or information has to be stored. Contents of MAR is directly connected to the address bus.
**Memory Buffer Register (MBR):** Contains a word of data to be written to memory or the word most recently read. Contents of MBR is directly connected to the data bus. It is also known as Memory Data Register(MDR).
Apart from these specific register, we may have some temporary registers which are not visible to the user. As such, there may be temporary buffering registers at the boundary to the ALU; these registers serve as input and output registers for the ALU and exchange data with the MBR and user visible registers.

**Processor Status Word**

All CPU designs include a register or set of registers, often known as the processor status word (PSW), that contains status information. The PSW typically contains condition codes plus other status information. Common fields or flags include the following:

- Sign : Contains the sign bit of the result of the last arithmatic operation.
- Zero : Set when the result is zero.
- Carry : Set if an operation resulted in a carry (addition) into or borrow (subtraction) out of a high order bit.
- Equal : Set if a logical campare result is equal.
- Overflow : Used to indicate arithmatic overflow.
- Interrupt enable/disable : Used to enable or disable interrupts.
- Supervisor : Indicate whether the CPU is executing in supervisor or user mode.

Certain privileged instructions can be executed only in supervisor mode, and certain areas of memory can be accessed only in supervisor mode.

Apart from these, a number of other registers related to status and control might be found in a particular CPU design. In addition to the PSW, there may be a pointer to a block of memory containing additional status information (e.g. process control blocks).

**Concept of Program Execution**

The instructions constituting a program to be executed by a computer are loaded in sequential locations in its main memory. To execute this program, the CPU fetches one instruction at a time and performs the functions specified. Instructions are fetched from successive memory locations until the execution of a branch or a jump instruction.

The CPU keeps track of the address of the memory location where the next instruction is located through the use of a dedicated CPU register, referred to as the program counter (PC). After fetching an instruction, the contents of the PC are updated to point at the next instruction in sequence.

For simplicity, let us assume that each instruction occupies one memory word. Therefore, execution of one instruction requires the following three steps to be performed by the CPU:

1. Fetch the contents of the memory location pointed at by the PC. The contents of this location are interpreted as an instruction to be executed. Hence, they are stored in the instruction register (IR). Symbolically this can be written as:

IR = [ [PC] ]

2. Increment the contents of the PC by 1.

PC = [PC] + 1

3. Carry out the actions specified by the instruction stored in the IR.

The first two steps are usually referred to as the fetch phase and the step 3 is known as the execution phase. Fetch cycle basically involves read the next instruction from the memory into the CPU and along with that update the contents of the program counter. In the execution phase, it interpretes the opcode and perform the indicated operation. The instruction fetch and execution phase together known as instruction cycle. The basic instruction cycle is shown in the figure.

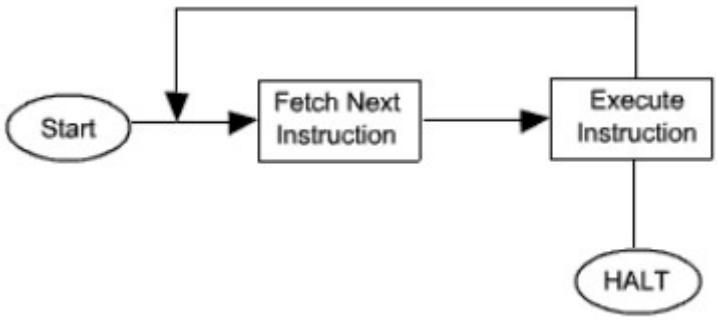



**Figure B :** Basic Instruction cycle

In cases, where an instruction occupies more than one word, step 1 and step 2 can be repeated as many times as necessary to fetch the complete instruction. In these cases, the execution of a instruction may involve one or more operands in memory, each of which requires a memory access. Further, if indirect addressing is used, then additional memory access are required.

The fetched instruction is loaded into the instruction register. The instruction contains bits that specify the action to be
performed by the processor. The processor interpretes the instruction and performs the required action. In general,
the actions fall into four categories:

- Processor-memory: Data may be transfrred from processor to memory or from memory to processor.
- Processor-I/O: Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.
- Data processing: The processor may perform some arithmatic or logic operation on data.
- Control: An instruction may specify that the sequence of execution be altered.

The main line of activity consists of alternating instruction fetch and instruction execution activities. After an instruction is fetched, it is examined to determine if any indirect addressing is involved. If so, the required operands are fetched using indirect addressing.
The execution cycle of a perticular instruction may involve more than one reference to memory. Also, instead of memory references, an instruction may specify an I/O operation. With these additional considerations the basic instruction cycle can be expanded with more details view in this figure. The figure is in the form of a state diagram.

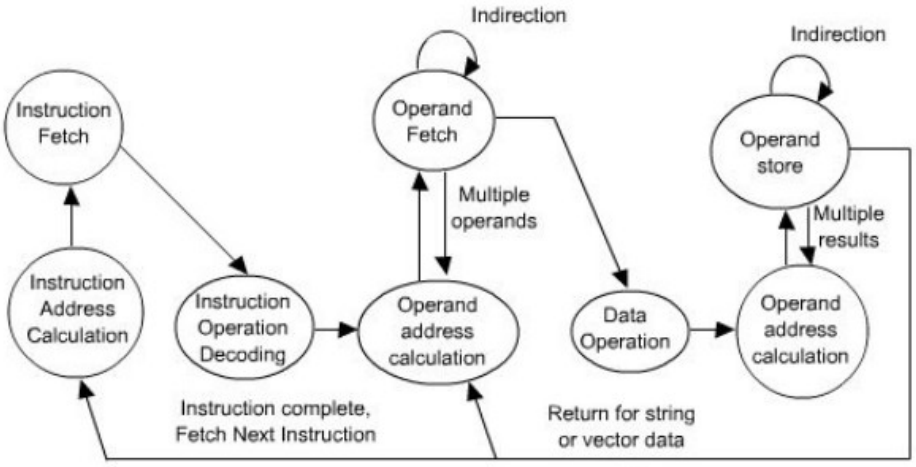



**Figure C :** Instruction cycle state diagram.

**Processor Organization**

There are several components inside a CPU, namely, ALU, control unit, general purpose register, Instruction registers etc. Now we will see how these components are organized inside CPU. There are several ways to place these components and inteconnect them. One such organization is shown in the figure A.

In this case, the arithmatic and logic unit (ALU), and all CPU registers are connected via a single common bus. This bus is internal to CPU and this internal bus is used to transfer the information between different components of the CPU. This organization is termed as single bus organization, since only one internal bus is used for transferring of information between different components of CPU. We have external bus or buses to CPU also to connect the CPU with the memory module and I/O devices. The external memory bus is also shown in the figure A connected to the CPU via the memory data and address register MDR and MAR .

The number and function of registers R0 to R(n-1) vary considerably from one machine to another. They may be given for general-purpose for the use of the programmer. Alternatively, some of them may be dedicated as special-purpose registers, such as index register or stack pointers .

In this organization, two registers, namely Y and Z are used which are transperant to the user. Programmer can not directly access these two registers. These are used as input and output buffer to the ALU which will be used in ALU operations. They will be used by CPU as temporary storage for some instructions.

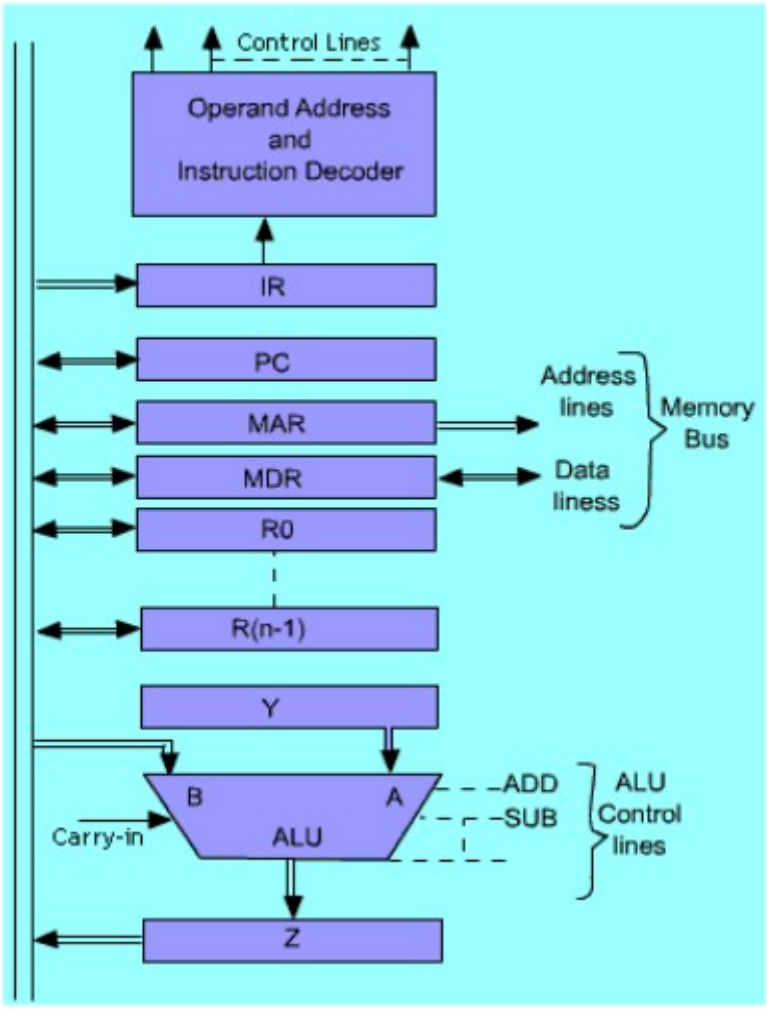



**Figure A :** Single bus organization of the data path inside the CPU

For the execution of an instruction, we need to perform an instruction cycle. An instruction cycle consists of two
phase,

- Fetch cycle and
- Execution cycle.

Most of the operation of a CPU can be carried out by performing one or more of the following functions in some prespecified sequence:

1. Fetch the contents of a given memory location and load them into a CPU register.
2. Store a word of data from a CPU register into a given memory location.
3. Transfer a word of data from one CPU register to another or to the ALU.
4. Perform an arithmatic or logic operation, and store the result in a CPU register.

Now we will examine the way in which each of the above functions is implemented in a computer.
Fetching a Word from Memory:
Information is stored in memory location indentified by their address. To fetch a word from memory, the CPU has to specify the address of the memory location where this information is stored and request a Read operation. The information may include both, the data for an operation or the instruction of a program which is available in main memory.

To perform a memory fetch operation, we need to complete the following tasks:
The CPU transfers the address of the required memory location to the Memory Address Register (MAR).
The MAR is connected to the memory address line of the memory bus, hence the address of the required word is transfered to the main memory.
Next, CPU uses the control lines of the memory bus to indicate that a Read operation is initiated. After issuing this request, the CPU waits until it receives an answer from the memory, indicating that the requested operation has been completed.
This is accomplished by another control signal of memory bus known as Memory-Function-Complete (MFC).
The memory set this signal to 1 to indicate that the contents of the specified memory location are available in memory data bus.
As soon as MFC signal is set to 1, the information available in the data bus is loaded into the Memory Data Register (MDR) and this is available for use inside the CPU.

As an example, assume that the address of the memory location to be accessed is kept in register R2 and that the
memory contents to be loaded into register R1. This is done by the following sequence of operations:
1. MAR← [R2]
2. Read
3. Wait for MFC signal
4. R1← [MDR]

The time required for step 3 depends on the speed of the memory unit. In general, the time required to access a word from the memory is longer than the time required to perform any operation within the CPU.
The scheme that is used here to transfer data from one device (memory) to another device (CPU) is referred to as an asynchronous transfer.
This asynchronous transfer enables transfer of data between two independent devices that have different speeds of operation. The data transfer is synchronised with the help of some control signals. In this example, Read request and MFC signal are doing the synchronization task.
An alternative scheme is synchronous transfer. In this case all the devices are controlled by a common clock pulse (continously running clock of a fixed frequency). These pulses provide common timing signal to the CPU and the main memory. A memory operation is completed during

every clock period. Though the synchronous data transfer scheme leads to a simpler implementation, it is difficult to accommodate devices with widely varying speed. In such cases, the duration of the clock pulse will be synchronized to the slowest device. It reduces the speed of all the devices to the slowest one.

**Storing a word into memory**

The procedure of writing a word into memory location is similar to that for reading one from memory. The only difference is that the data word to be written is first loaded into the MDR, the write command is issued. As an example, assumes that the data word to be stored in the memory is in register R1 and that the memory address is in register R2. The memory write operation requires the following sequence:

1. MAR← [R2]
2. MDR→ [R1]
3. Write
4. Wait for MFC

- In this case step 1 and step 2 are independent and so they can be carried out in any order. In fact, step 1 and 2 can be carried out simultaneously, if this is allowed by the architecture, that is, if these two data transfers (memory address and data) do not use the same data path.

In case of both memory read and memory write operation, the total time duration depends on wait for the MFC signal, which depends on the speed of the memory module.

There is a scope to improve the performance of the CPU , if CPU is allowed to perform some other operation while waiting for MFC signal. D uring the period, CPU can perform some other instructions which do not require the use of MAR and MDR.

**Register Transfer Operation**

Register transfer operations enable data transfer between various blocks connected to the common bus of CPU . We have several registers inside CPU and it is needed to transfer information from one register another. As for example during memory write operation data from appropriate register must be moved to MDR. Since the input output lines of all the register are connected to the common internal bus, we need appropriate input output gating. The input and output gates for register R i are controlled by the signal R i in and R i out respectively.

Thus, when Riin set to 1 the data available in the common bus is loaded into Ri. Similarly when, Riout is set to 1, the contents of the register Ri are placed on the bus. To transfer data from one register to other register, we need to generate the appropriate register gating signal.

For example, to transfer the contents of register R1 to register R2 , the following actions are needed:

- Enable the output gate of register R1 by setting R1out to 1.
  -- This places the contents of R1 on the CPU bus.
- Enable the input gate of register R2 by setting R2in to 1.
  -- This loads data from the CPU bus into the register R2 .

**Performing the arithmetic or logic operation:**

Generally, ALU is used inside CPU to perform arithmetic and logic operation. ALU is a combinational logic circuit which does not have any internal storage.

Therefore, to perform any arithmetic or logic operation (say binary operation) both the input should be made available at the two inputs of the ALU simultaneously. Once both the inputs are available then appropriate signal is generated to perform the required operation.

We may have to use temporary storage (register) to carry out the operation in ALU .

The sequence of operations that have to carried out to perform one ALU operation depends on the organization of the CPU. Consider an organization in which one of the operand of ALU is stored in some temporary register Y and other operand is directly taken from CPU internal bus. The result of the ALU operation is stored in another temporary register Z.

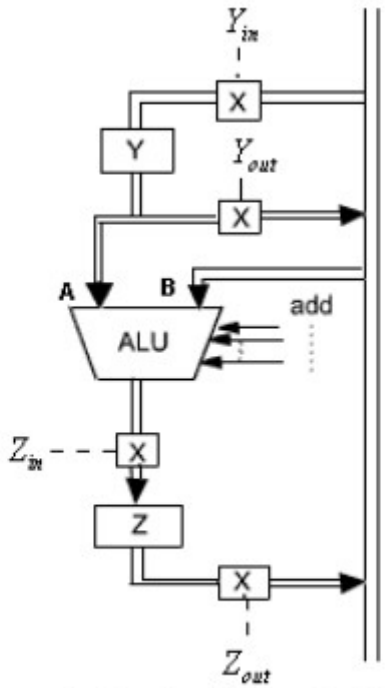



**Figure B :** Organization for Arithmatic & Logic Operation.

**Multiple Bus Organization**

Till now we have considered only one internal bus of CPU. The single-bus organization, which is only one of the possibilities for interconnecting different building blocks of CPU.

An alternative structure is the two bus structure, where two different internal buses are used in CPU. All register outputs are connected to bus A, add all registered inputs are connected to bus B.

There is a special arrangement to transfer the data from one bus to the other bus. The buses are connected through the bus tie G. When this tie is enabled data on bus A is transfer to bus B. When G is disabled, the two buses are electrically isolated.

Since two buses are used here the temporary register Z is not required here which is used in single bus organization to store the result of ALU. Now result can be directly transferred to bus B, since one of the inputs is in bus A. With the bus tie disabled, the result can directly be transferred to destination register.

For example, for the operation, [R3] ← [R1] + [R2] can now be performed as

1. R1 out , G enable , Y in
2. R2 out , Add, ALU out , R3 in

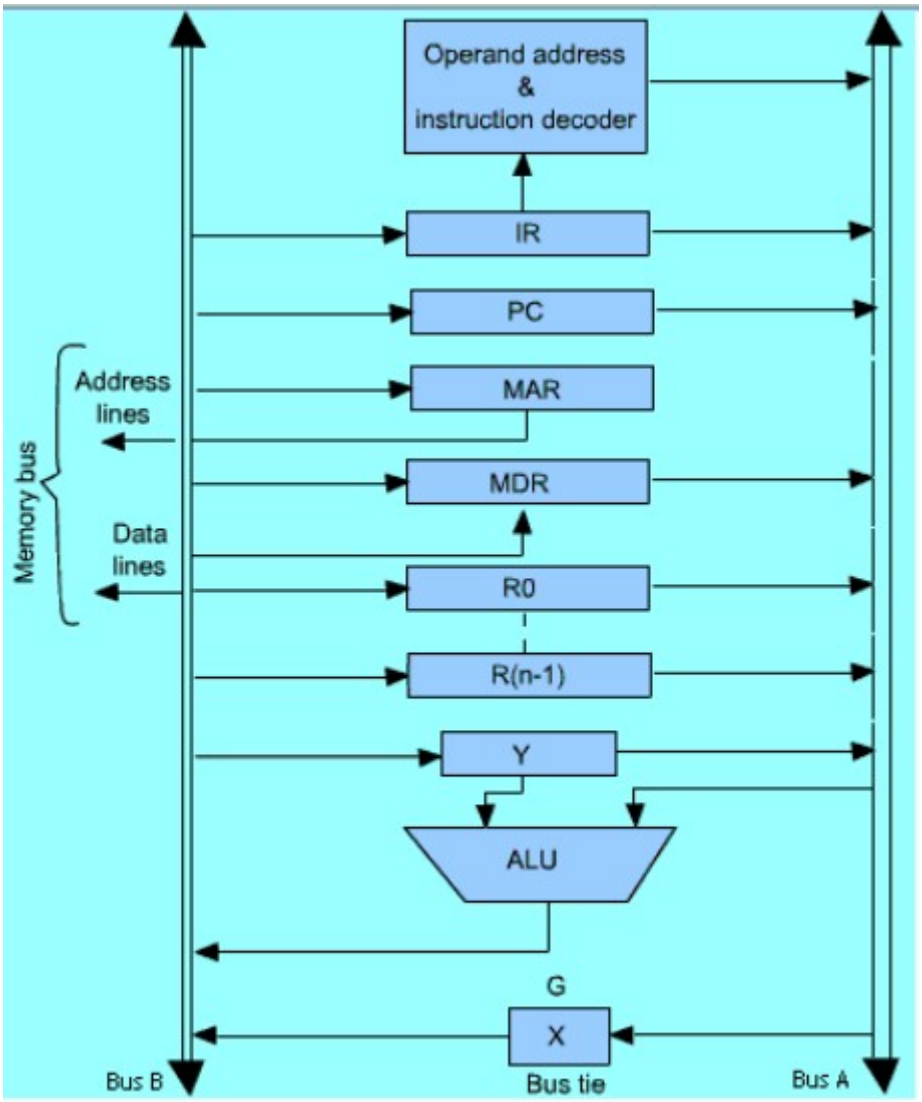



**Figure C :** Two bus structure

In this case source register R2 and destination register R3 has to be different, because the two operations R2 in and R2 out can not be performed together. If the registers are made of simple latches then only we have the restriction.

We may have another CPU organization, where three internal CPU buses are used. In this organization each bus connected to only one output and number of inputs. The elimination of the need for connecting more than one output to the same bus leads to faster bus transfer and simple control. A simple three-bus organization is shown in the figure D.

A multiplexer is provided at the input to each of the two work registers A and B, which allow them to be loaded from either the input data bus or the register data bus. In the diagram, a possible interconnection of three-bus organization is presented, there may be different interconnections possible. In this three bus organization, we are keeping two input data buses instead of one that is used in two bus organization.

Two separate input data buses are present – one is for external data transfer, i.e. retrieving from memory and the second one is for internal data transfer that is transferring data from general purpose register to other building block inside the CPU.

**Execution of a Complete Instructions:**
We have discussed about four different types of basic operations:

- Fetch information from memory to CPU
- Store information to CPU register to memory
- Transfer of data between CPU registers.
- Perform arithmetic or logic operation and store the result in CPU registers.

To execute a complete instruction we need to take help of these basic operations and we need to execute these operation in some particular order to execute an instruction.
As for example, consider the instruction : "Add contents of memory location NUM to the contents of register R1 and store the result in register R1." For simplicity, assume that the address NUM is given explicitly in the address field of the instruction .That is, in this instruction, direct addressing mode is used.
Execution of this instruction requires the following action :

1. Fetch instruction
2. Fetch first operand (Contents of memory location pointed at by the address field of the instruction)
3. Perform addition
4. Load the result into R1.

Instruction execution proceeds as follows:
**In Step1:**
The instruction fetch operation is initiated by loading the contents of the PC into the MAR and sending a read request to memory.
To perform this task first of all the contents of PC have to be brought to internal bus and then it is loaded to MAR.To perform this task control circuit has to generate the PC out signal and MAR in signal. After issuing the read signal, CPU has to wait for some time to get the MFC signal. During that time PC is updated by 1 through the use of the ALU. This is accomplished by setting one of the inputs to the ALU (Register Y) to 0 and the other input is available in bus which is current value of PC. At the same time, the carry-in to the ALU is set to 1 and an add operation is specified.

**In Step 2:**
The updated value is moved from register Z back into the PC. Step 2 is initiated immediately after issuing the memory Read request without waiting for completion of memory function. This is possible, because step 2 does not use the memory bus and its execution does not depend on the memory read operation.

**In Step 3:**
Step3 has been delayed until the MFC is received. Once MFC is received, the word fetched from the memory is transfered to IR (Instruction Register), Because it is an instruction. Step 1 through 3 constitute the instruction fetch phase of the control sequence.
The instruction fetch portion is same for all instructions. Next step onwards, instruction execution phase takes place.

As soon as the IR is loaded with instruction, the instruction decoding circuits interprets its contents. This
enables the control circuitry to choose the appropriate signals for the remainder of the control sequence, step 4 to 8,
which we referred to as the execution phase. To design the control sequence of execution phase, it is needed to
have the knowledge of the internal structure and instruction format of the PU. Secondly , the length of instruction

phase is different for different instruction.
In this example , we have assumed the following instruction format :

| | | |
|---|---|---|
| opcode | M | R |

i.e., opcode: Operation Code
M: Memory address for source
R: Register address for source/destination

**In Step 5 :**
The destination field of IR, which contains the address of the register R1, is used to transfer the contents of register R1 to register Y and wait for Memory function Complete. When the read operation is completed, the memory operand is available in MDR.

**In Step 6 :**
The result of addition operation is performed in this step.

**In Step 7:**
The result of addition operation is transferred from temporary register Z to the destination register R1 in this step.

**In step 8 :**
It indicates the end of the execution of the instruction by generating End signal. This indicates completion of execution of the current instruction and causes a new fetch cycle to be started by going back to step 1.

**Branching**
With the help of branching instruction, the control of the execution of the program is transfered from one particular position to some other position, due to which the sequence flow of control is broken. Branching is accomplished by replacing the current contents of the PC by the branch address, that is, the address of the instruction to which branching is required.
Consider a branch instruction in which branch address is obtained by adding an offset X, which is given in the address field of the branch instruction, to the current value of PC.

Consider the following unconditional branch instruction
JUMP X

i.e., the format is

| | |
|---|---|
| op-code | Offset of jump |

**Design of Control Unit**

To execute an instruction, the control unit of the CPU must generate the required control signal in the proper sequence. As for example, during the fetch phase, CPU has to generate PC out signal along with other required signal in the first clock pulse. In the second clock pulse CPU has to generate PC in signal along with other required signals.

So, during fetch phase, the proper sequence for generating the signal to retrieve from and store to PC is PCout and PCin .

To generate the control signal in proper sequence, a wide variety of techniques exist. Most of these techniques, howeve, fall into one of the two categories,

1. Hardwired Control
2. Microprogrammed Control.

**Hardwired Control**

In this hardwired control techniques, the control signals are generated by means of hardwired circuit. The main objective of control unit is to generate the control signal in proper sequence. Consider the sequence of control signal required to execute the ADD instruction that is explained in previous lecture. It is obvious that eight non-overlapping time slots are required for proper execution of the instruction represented by this sequence. Each time slot must be at least long enough for the function specified in the corresponding step to be completed.

Since, the control unit is implemented by hardwire device and every device is having a propagation delay, due to which it requires some time to get the stable output signal at the output port after giving the input signal. So, to find out the time slot is a complicated design task.

For the moment, for simplicity, let us assume that all slots are equal in time duration. Therefore the required controller may be implemented based upon the use of a counter driven by a clock.

Each state, or count, of this counter corresponds to one of the steps of the control sequence of the instructions of the CPU.

In the previous lecture, we have mentioned control sequence for execution of two instructions only (one is for add and other one is for branch). Like that we need to design the control sequence of all the instructions.

By looking into the design of the CPU, we may say that there are various instruction for add operation. As for example,

ADD NUM R1 Add the contents of memory location specified by NUM to the contents of register R1 .

ADD R2 R1 Add the contents of register R 2 to the contents of register R 1 .

The control sequence for execution of these two ADD instructions are different. Of course, the fetch phase of all the instructions remain same.

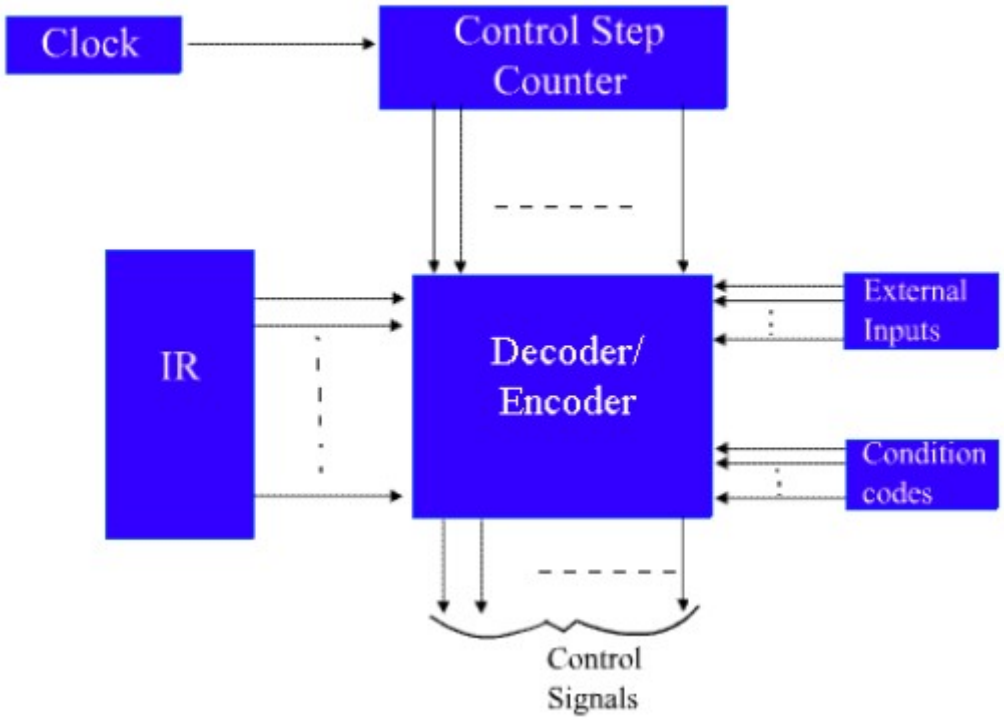
It is clear that control signals depend on the instruction, i.e., the contents of the instruction register. It is also observed that execution of some of the instructions depend on the contents of condition code or status flag register, where the control sequence depends in conditional branch instruction.

Hence, the required control signals are uniquely determined by the following information:

- Contents of the control counter.
- Contents of the instruction register.
- Contents of the condition code and other status flags.

The external inputs represent the state of the CPU and various control lines connected to it, such as MFC status signal. The condition codes/ status flags indicates the state of the CPU. These includes the status flags like carry, overflow, zero, etc.

## Control Unit Organization




The structure of control unit can be represented in a simplified view by putting it in block diagram. The detailed hardware involved may be explored step by step. The simplified view of the control unit is given in the figure above (A) The decoder/encoder block is simply a combinational circuit that generates the required control outputs depending on the state of all its input.

The decoder part of decoder/encoder part provide a separate signal line for each control step, or time slot in the control sequence. Similarly, the output of the instructor decoder consists of a separate line for each machine instruction loaded in the IR, one of the output line INS 1 to INS m is set to 1 and all other lines are set to 0.

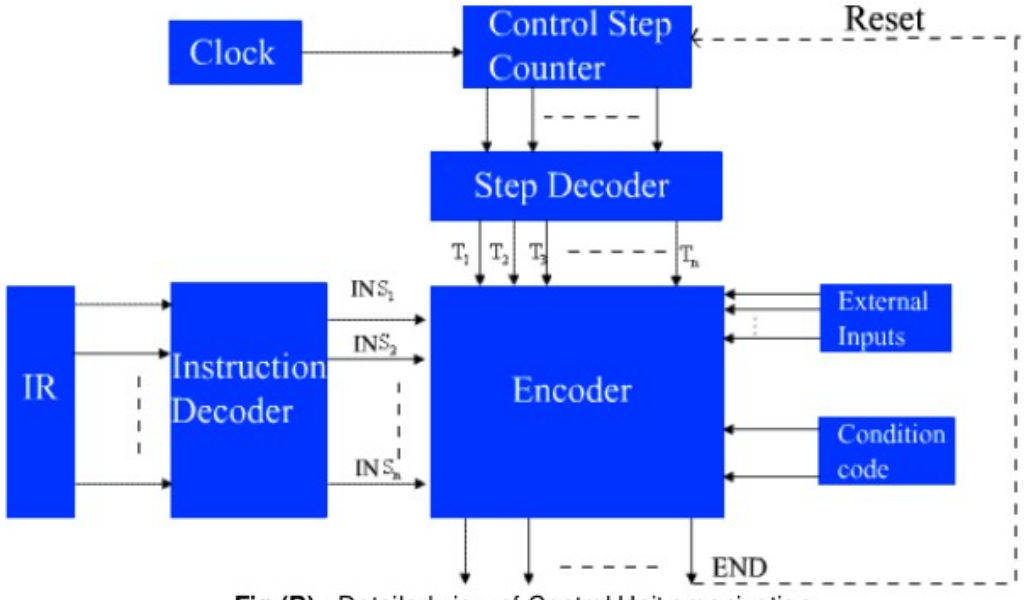



**Fig (B) :** Detailed view of Control Unit organization

All input signals to the encoder block should be combined to generate the individual control signals. In the previous section, we have mentioned the control sequence of the instruction,
"Add contents of memory location address in memory direct made to register R 1 ( ADD_MD)",
"Control sequence for an unconditional branch instruction (BR)",
also, we have mentioned about Branch on negative (BRN).
Consder those three CPU instruction ADD_MD, BR, BRN.
It is required to generate many control signals by the control unit. These are basically coming out from the encoder circuit of the control signal generator. The control signals are: PC in , PC out , Z in , Z out , MAR in , ADD, END, etc.

By looking into the above three instructions, we can write the logic function for Z in as :

Z in = T 1 + T 6 . ADD_MD + T 5 . BR + T 5 . BRN + . . . . . .

For all instructions, in time step1 we need the control signal Z in to enable the input to register Z in time cycle T 6 of ADD_MD instruction, in time cycle T 5 of BR instruction and so on.

Similarly, the Boolean logic function for ADD signal is

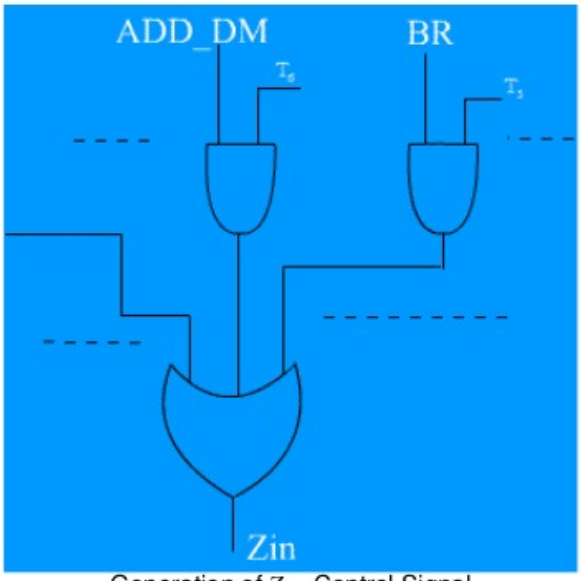
ADD = T 1 + T 6 . ADD_MD + T 5 . BR + . . . . . .

These logic functions can be implemented by a two level combinational circuit of AND and OR gates. Similarly, the END control signal is generated by the logic function :

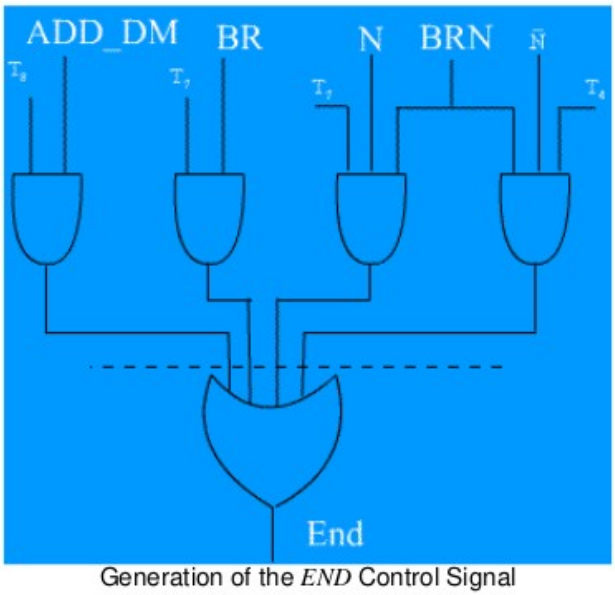
END = T 8 . ADD_MD + T 7 . BR + ( T 7 . N + T 4 . N^) . BRN + . . . . . .

This END signal indicates the end of the execution of an instruction, so this END signal can be used to start a new instruction fetch cycle by resetting the control step counter to its starting value.

The circuit diagram (Partial) for generating $Z_{in}$ and *END* signal is shown in the diagram.



Generation of $Z_{in}$ Control Signal



Generation of the *END* Control Signal

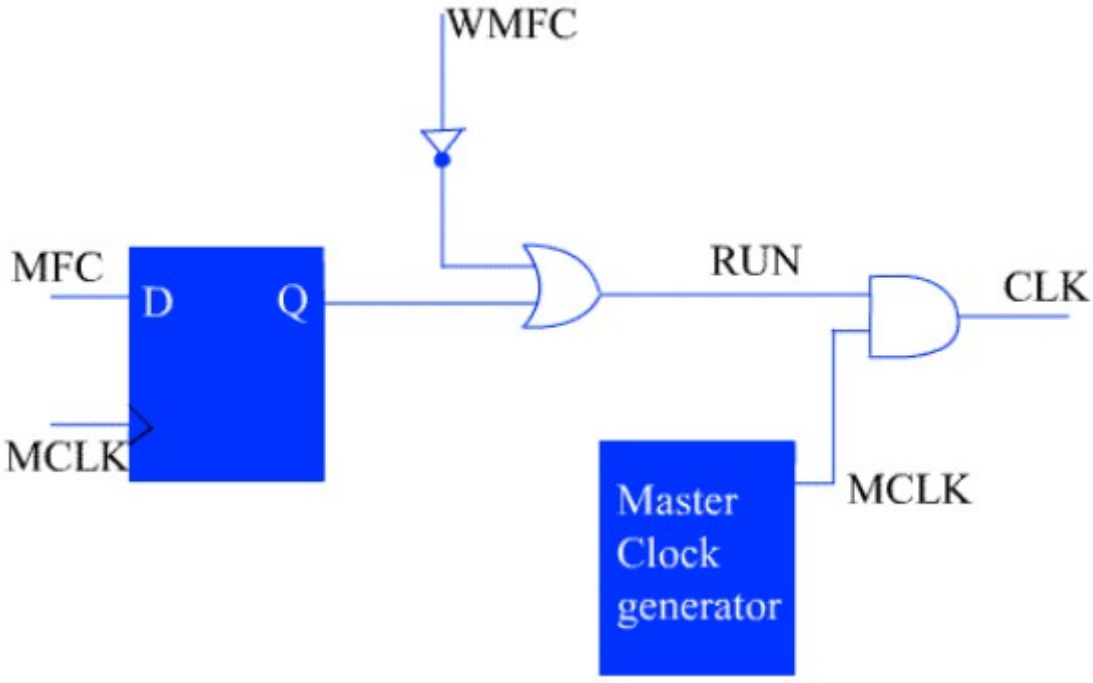
The signal ADD_MD, BR, BRN etc. are coming from instruction decoder circuits which depends on the contents of IR.

The signal T 1 , T 2 , T 3 etc are coming out from step decoder depends on control step counter.
The signal N (Negative) is coming from condition code register.

When wait for MFC (WMFC) signal is generated, then CPU does not do any works and it waits for an MFC signal from memory unit. In this case, the desired effect is to delay the initiation of the next control step until the MFC signal is received from the main memory. This can be incorporated by inhibiting the advancement of the control step counter for the required period.
Let us assume that the control step counter is controlled by a signal called RUN.
By looking at the control sequence of all the instructions, the WMFC signal is generated as:

WMFC = T 2 + T 5 . ADD_MD + . . . . . .

The RUN signal is generated with the help of WMFC signal and MFC signal. The arrangement is shown in the figure.

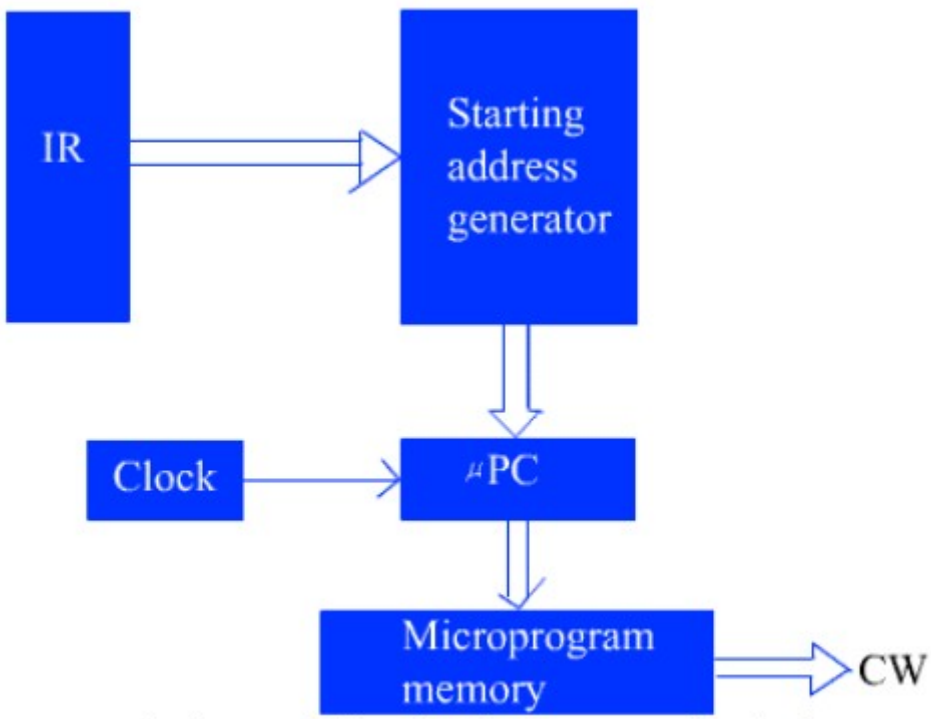


The MFC signal is generated by the main memory whose operation is independent of CPU clock. Hence MFC is an asynchronous signal that may arrive at any time relative to the CPU clock. It is possible to synchronized with CPU clock with the help of a D flip-flop.
When WMFC signal is high, then RUN signal is low. This run signal is used with the master clock pulse through an AND gate. When RUN is low, then the CLK signal remains low, and it does not allow to progress the control step counter.
When the MFC signal is received, the run signal becomes high and the CLK signal becomes same with the MCLK signal and due to which the control step counter progresses. Therefore, in the next control step, the WMFC signal goes low and control unit operates normally till the next memory access signal is generated.

**Microprogrammed Control**
In hardwired control, we saw how all the control signals required inside the CPU can be generated using a state counter and a PLA circuit.
There is an alternative approach by which the control signals required inside the CPU can be generated . This alternative approach is known as microprogrammed control unit. In microprogrammed control unit, the logic of the control unit is specified by a microprogram. A microprogram consists of a sequence of instructions in a microprogramming language. These are instructions that specify microoperations.
A microprogrammed control unit is a relatively simple logic circuit that is capable of (1) sequencing through microinstructions and (2) generating control signals to execute each microinstruction.
The concept of microprogram is similar to computer program. In computer program the complete instructions of the program is stored in main memory and during execution it fetches the instructions from main memory one after another. The sequence of instruction fetch is controlled by program counter (PC) . Microprogram are stored in microprogram memory and the execution is controlled by microprogram counter (uPC).

Microprogram consists of microinstructions which are nothing but the strings of 0's and 1's. In a particular instance, we read the contents of one location of microprogram memory, which is nothing but a microinstruction. Each output line ( data line ) of microprogram memory corresponds to one control signal. If the contents of the memory cell is 0, it indicates that the signal is not generated and if the contents of memory cell is 1, it indicates to generate that control signal at that instant of time.
**Control Word (CW) :**
Control word is defined as a word whose individual bits represent the various control signal. Therefore each of the control steps in the control sequence of an instruction defines a unique combination of 0s and 1s in the CW. A sequence of control words (CWs) corresponding to the control sequence of a machine instruction constitutes the microprogram for that instruction.
The individual control words in this microprogram are referred to as microinstructions. The microprograms corresponding to the instruction set of a computer are stored ina aspecial memory which will be referred to as the microprogram memory. The control words related to an instructions are stored in microprogram memory.

The control unit can generate the control signals for any instruction by sequencially reading the CWs of the corresponding microprogram from the microprogram memory. To read the control word sequentially from the microprogram memory a microprogram counter ( PC) is needed.
The basic organization of a microprogrammed control unit is shown in the figure. The "starting address generator" block is responsible for loading the starting address of the microprogram into the PC everytime a new instruction is loaded in the IR. The PC is then automatically incremented by the clock, and it reads the successive microinstruction from memory.

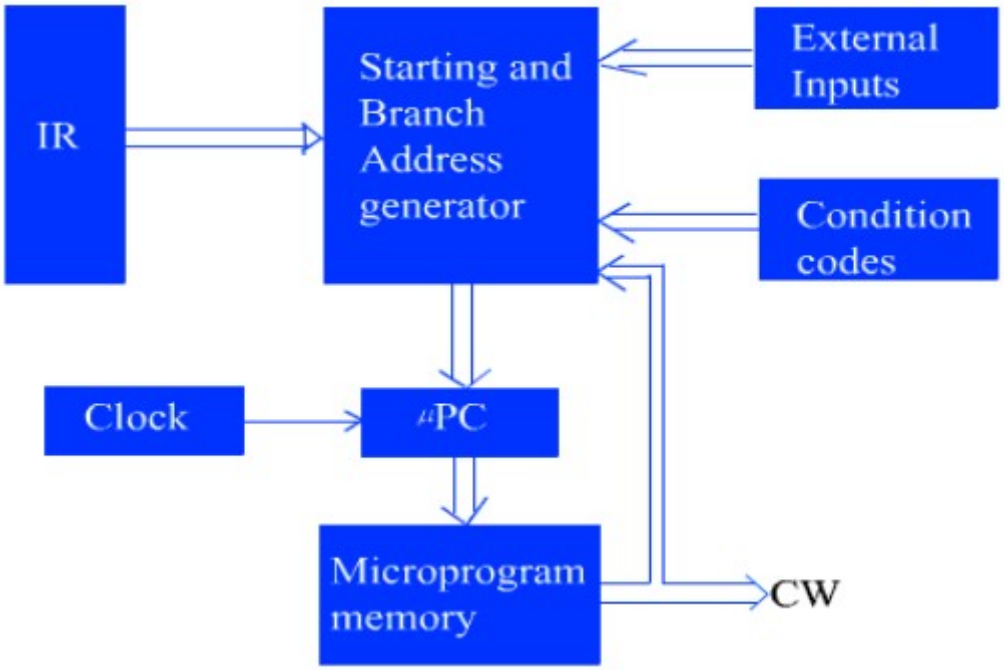


Basic organization of a microprogrammed control

Each
microinstruction basically provides the required control signal at that time step. The microprogram counter ensures that the control signal will be delivered to the various parts of the CPU in correct sequence.
We have some instructions whose execution depends on the status of condition codes and status flag, as for example, the branch instruction. During branch instruction execution, it is required to take the decision between the alternative action. To handle such type of instructions with microprogrammed control, the design of control unit is based on the concept of conditional branching in the microprogram. For that it is required to include some conditional branch

microinstructions. In conditional microinstructions, it is required to specify the address of the microprogram memory to which the control must direct. It is known as branch address. Apart from branch address, these microinstructions can specify which of the states flags, condition codes, or possibly, bits of the instruction register should be checked as a condition for branching to take place.
To support microprogram branching, the organization of control unit should be modified to accommodate the branching decision.
To generate the branch address, it is required to know the status of the condition codes and status flag.
To generate the starting address, we need the instruction which is present in IR. But for branch address generation we have to check the content of condition codes and status flag.

The organization of control unit to enable conditional branching in the microprogram is shown in the figure.




The control bits of the microinstructions word which specify the branch conditions and address are fed to the "Starting and branch address generator" block.
This block performs the function of loading a new address into the satisfied. PC when the condition of branch instruction is In a computer program we have seen that execution of every instruction consists of two part - fetch phase and execution phase of the instruction. It is also observed that the fetch phase of all instruction is same.
In microprogrammed controlled control unit, a common microprogram is used to fetch the instruction. This microprogram is stored in a specific location and execution of each instruction start from that memory location.

At the end of fetch microprogram, the starting address generator unit calculate the appropriate starting address of the microprogram for the instruction which is currently present in IR. After the PC controls the execution of microprogram which generates the appropriate control signal in proper sequence.

During the execution of a microprogram, the PC is always incremented everytime a new microinstruction is fetched from the microprogram memory, except in the following situations :

1. When an End instruction is encountered, uPC is loaded with the address of the first CW in the microprogram for the instruction fetch cycle.
2. When a new instruction is loaded into the IR, the PC is loaded with the starting address of the microprogram for that instruction.
3. When a branch microinstruction is encountered, and the branch condition is satisfied, the PC is loaded with the branch address.