BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 3

DCA2103

COMPUTER ORGANIZATION

Unit 5

Control Unit Design

Table of Contents

1. INTRODUCTION

In the previous unit, we studied the various aspects of the processor. The unit started with the discussion of parallelism and computer arithmetic. Then we discussed the 8086 processor and the registers organization of the 8086 processors and other concepts like Instruction Cycles, Read Write cycles, and Addressing Modes. In this unit, we focus on the most complex aspect of ALU and control unit which are the main components of the processing unit. We have seen in the preceding units the task that the CPU must-do for the execution of an instruction. That is it fetch instruction, interprets it, fetches data, Processes data, and finally writes the result into the appropriate location. An 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 memory of the CPU that is registers.

An algorithm written to solve a problem consists of a number of steps which are to be carried out in a specific sequence. To implement this algorithm, these steps are broken down into smaller steps, where each smaller step represents one machine instruction. The resulting sequence of instructions is called a machine language program which represents the algorithm. Each of these machine instructions is executed by carrying out a sequence of rudimentary operations. These operations are the means by which they are generated, and how each individual operation is caused to happen. The control unit is the portion of the processor that actually causes things to happen. The control unit issues control signals:

- Internal to the processor to move data between registers, to cause ALU to perform a specified function, and to regulate other internal operations,
- External to the processor to cause data exchange with memory and I/O modules.

1.1 Objectives:

After studying this unit, you should be able to:

- *List and explain the different micro-operations that occur in the execution of an instruction.*
- *List and explain the registers that are involved in the micro-operation of a fetch cycle.*
- *Define indirect and interrupt the cycle.*
- *Explain the possible units of control unit*
- *With neat sketches explain the one bus structure*
- *Discuss hardwired control unit*

2. MICRO OPERATIONS

We have already come across the fact that the execution of one instruction consists of one or more fetch and one execute cycle. The time or clock for the execution of a single instruction is called an instruction cycle. Thus a program execution that has various numbers of instructions consists of a sequence of instruction cycles with one machine instruction per cycle. The sequence of instruction cycles is not necessarily the same for every instruction.

Each instruction cycle is considered to be made up of smaller units. The most common smaller units of the instruction cycle are:

- Fetch
- Indirect
- Execute
- Interrupt

Fetch and execute cycles always occurs in every instruction cycle. To design a control unit we need to break the descriptions down further. Each smaller cycle involves a series of steps called micro-operations. These microoperations involve the CPU registers. Figure 5.1 shows the elements of a program execution.

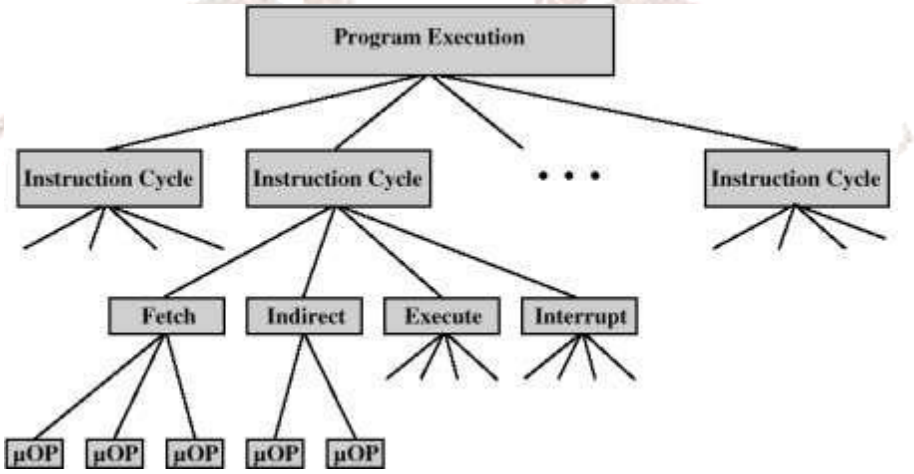

Fig 5.1: Elements of a program execution

2.1 Micro Operations of Fetch Cycle

Fetch cycle occurs at the beginning of every instruction. Four main registers that are involved are

- Memory Address Register (MAR):
  - Connected to address bus
  - Specifies address for read or write operation
- Memory Buffer Register (MBR)
  - Connected to the data bus
  - Holds data to write or last data read
- Program Counter (PC)
  - Holds the address of next instruction to be fetched (not connected to address bus)
- Instruction Register (IR)
  - Holds last instruction fetched (not connected to data bus)

Symbolically the Sequence of events in the fetch cycle is given below:

- T1: MAR ← PC
- T2: MBR ← memory
  PC ← PC +1
- T3: IR ← MBR

These are sequences listed from the point of view of its effect on the CPU registers. An example is given in figure 5.2.

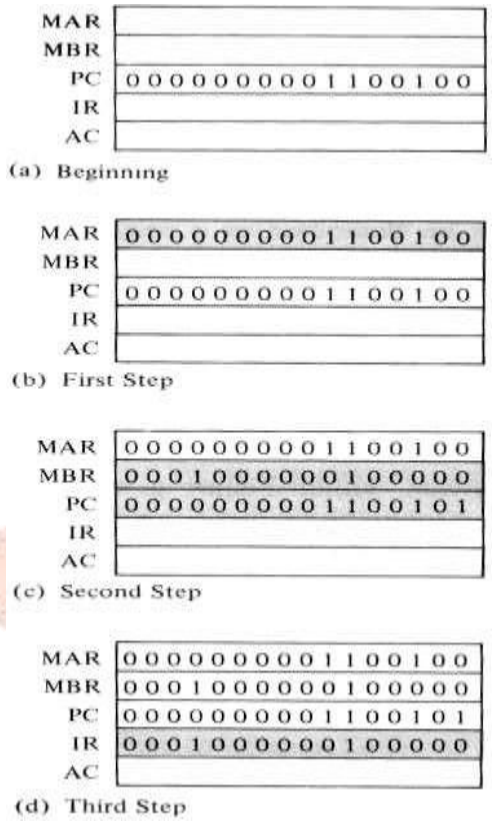

Fig 5.2: Contents of CPU registers for various steps

At the beginning of the fetch cycle, the address of the next instruction is in the program counter (PC). In this case it 1100100.

- The first step is to move that address to the MAR since it is the only register connected to the address lines of the system bus.
- The second step is to bring in the instruction. Hence the desired address is placed on the address bus (MAR) and the control unit issues a READ command on the control bus, and the result (data from memory) appears on the data bus which is then copied into MBR. We also need the next instruction address which is there in the next adjacent location of the previous instruction and hence we need to increment the PC incremented by 1 (in parallel with data fetch from memory) to get ready for the next instruction.
- The third step is to move the Data (instruction) from MBR to IR. And MBR is now free for further data fetches

2.2 Indirect Cycle

Once the instruction is fetched, the next step is to fetch source operands. Using the same example let us assume a one-byte instruction, with direct or indirect addressing. Now if the instruction uses direct addressing then the operand is implicitly specified in the instruction only. But if the instruction uses indirect addressing then the indirect cycle much precedes the execution cycle. Now considering the indirect addressing, the data flow, therefore, consists of the following micro-operations.

- T4: $MAR \leftarrow (IR_{ADDRESS})$
- T5: $MBR \leftarrow$ memory
- T6: $IR_{ADDRESS} \leftarrow MBR_{ADDRESS}$

The address field of the instruction is transferred to the MAR. This address is then used to fetch the address of the operand. Now the address of IR is updated from the MBR, and it contains a direct rather than indirect address. The IR is not in the same state as if indirect addressing is not been used, and ready for the execution cycle. We will see the execution cycle after the interrupt cycle.

The Interrupt cycle

At the completion of executing cycle, a test is made to determine whether any enabled interrupts have occurred. If so, the interrupt cycle occurs. The nature of this cycle varies greatly from one machine to another. Let us consider the following sequence of events:

- T1: $MBR \leftarrow (PC)$
- T2: $MAR \leftarrow$ save-address
  $PC \leftarrow$ routine-address
- T3: memory $\leftarrow (MBR)$

In the first step, the contents of the PC are transferred to MBR, so that they can be saved for return from the interrupt. Then the MAR is loaded with the address at which the contents of the PC are to be saved, and the PC is loaded with the address of the start of the interrupt processing routine. These two actions may each be a single micro-operation. However, since most CPUs provide multiple types and/or levels of interrupts, it may take one or more additional micro-operations to obtain the save address and the routine address before they can be transferred to the MAR and PC, respectively. In any case, once this is over the final

step is to store MBR which contains the old value of the PC, into memory. The CPU is now ready to begin the next instruction cycle.

2.3 The Execute Cycle

The fetch, indirect, and interrupt cycles are simple. Each of these involves a small, fixed sequence of micro-operations which have predictable. The same micro-operations are repeated each time around. It is not the same with the execution cycle.

For a machine with N different op codes, there are N different sequences of micro-operations that can occur. Let us discuss these micro-operations with several different illustrative examples.

Example 1: Let us consider an ADD instruction given below:

ADD R1, X

This instruction adds the contents of location X to register R1. The following sequence of micro-operations might occur:

- T6: MAR ← ($IR_{ADDRESS}$)
- T7: MBR ← (memory)
- T8: R1 ← R1 + (MBR)

We start with the IR containing the ADD instruction. In the first step, the address portion of the MBR is loaded into the MAR. Then the referenced memory location is read. Finally, the contents of R1 and MBR are added by the ALU. This was a very simple example, there may be an additional micro operation required if register reference is to be extracted from the IR and perhaps to stage to the ALU inputs or outputs in some intermediate registers.

Example 2: Now let us take another example which is complex

ISZ X

This instruction is called an increment and skips if zero. The content of location X is incremented by 1. If the result is 0, the next instruction is skipped. A possible sequence of micro-operations is given below:

- t1: MAR ← ($IR_{address}$)
- t2: MBR ← (memory)
- t3: MBR ← (MBR) + 1

- t4: memory ← (MBR)
- if (MBR) == 0 thenPC ← (PC) + 1

The new feature in this example was conditional action. The PC is incremented if MBR = 0. This test and action can be implemented as one micro-operation. This micro-operation is performed during which the updated value in MBR is stored back in memory.

Example 3: Let us consider a subroutine call instruction which is a branch and save the address.

BSA X

The address of the instruction that follows the BSA instruction is saved in location X and execution continues at location X+1. The saved address will later be used for the return. The following is the sequence of possible micro operations:

- T1: MAR ← ($IR_{ADDRESS}$)
  MBR <- (PC)
- T2: PC ← ($IR_{ADDRESS}$)
  memory ← (MBR)
- T3: PC ← (PC) + 1

The address in the PC at the start of the instruction is the address of the next instruction in sequence. This is saved at the address designated IR. The latter address is also incremented to provide the address of instruction for the next instruction cycle.

2.4 The Instruction Cycle

Each phase of the instruction cycle can be decomposed into a sequence of elementary operations called micro-operations. Thus, there is one sequence of micro-operation in every phase for every opcode. The flow chart of all the sequence of micro-operations is tied up together and is as shown in figure 5.3. Here we assume a new 2-bit registers called instruction cycle code (ICC). The ICC values give the state of the CPU and indicate in which state it is. (Refer table 5.1).

Table 5.1: State of CPU

Code for ICC	State
00	Fetch
01	Indirect
10	Execute
11	Interrupt

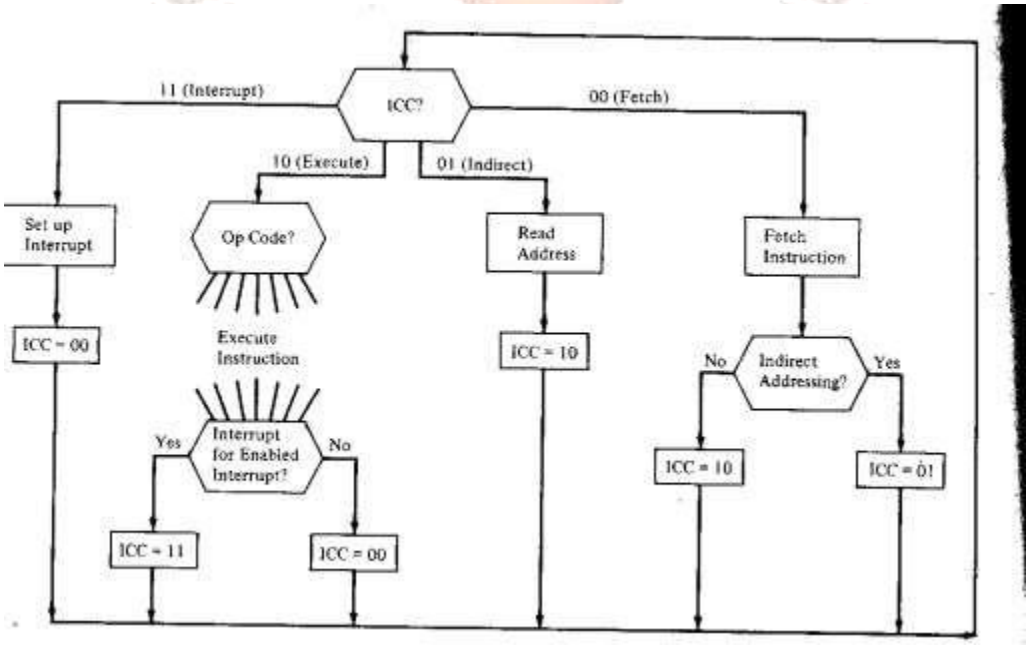

Fig 5.3: Flowchart for instruction cycle

Self-Assessment Questions – 1

1. The time or clock for execution of a single instruction is called an _____ .
2. _____cycles always occurs in every instruction cycle.
3. Each smaller cycles involves a series of steps called_____.
4. _____holds address of next instruction to be fetched.

3. CONTROL OF THE CPU

In the previous section, we have seen a reduction in the operation of the CPU to its most fundamental level. Using these reductions we will be to define what exactly it is that the control unit must cause to happen. Here in this section first we define the functional requirements for the control unit.

3.1 Functional Requirements

The functional requirements of the control unit are those functions that the control unit must perform. And these are the basis for the design and implementation of the control unit.

A three-step process that leads to the characterization of the Control unit:

- Define the basic elements of the processor
- Describe the micro-operations that the processor performs
- Determine the functions that the control unit must perform to cause the micro-operations to be performed.

1. Basic Elements of Processor

The following are the basic functional elements of a CPU:

- ALU: is the functional essence of the computer.
- Registers: are used to store data internal to the CPU.

As discussed earlier, some registers also contain status information needed to manage instruction sequencing ex: program status words. Some registers contain data for transfer between ALU, memory, and I/O modules.

- Internal data paths: are used to move data between the registers and ALU
- External data paths: are used to link registers and I/O modules, often by a system bus.
- Control Unit: The control unit causes operations to happen within the CPU.

2. Types of Micro-operation

The execution of a program as seen earlier consists of operations involving these CPU elements. These operations consist of a sequence of micro-operations. All micro instructions fall into one of the following categories:

- Transfer data between registers
- Transfer data from the register to external

- Transfer data from external to register
- Perform arithmetic or logical ops

All micro-operations need to perform one instruction cycle.

3. Functions of Control Unit

Now we define more explicitly the function of the control unit. The control unit performs two tasks:

- Sequencing: The control unit causes the CPU to step through a series of micro-operations in proper sequence based on the program being executed.
- Execution: The control unit causes each micro-operation to be performed.

These tasks are accomplished using Control Signals

3.2 Control Signals

For the control unit to perform its function, it must have inputs that allow it to determine the state of the system and outputs that allow it to control the behavior of the system. These are the external specifications of the control unit. Internally, the control unit must have the logic required to perform sequencing and execution functions.

Let us take a close look at the general model of the control unit given in figure 5.4 (a). The model is indicated with all of its inputs and outputs. As from the earlier section, we know that the function of the control unit is sequencing and then execution of the instruction. Hence the control unit organization can be considered as shown in figure 5.4(b).

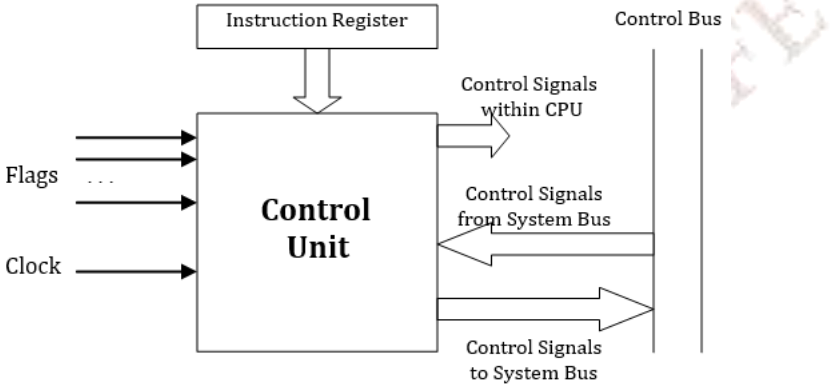

Fig 5.4 (a): Model of the Control Unit

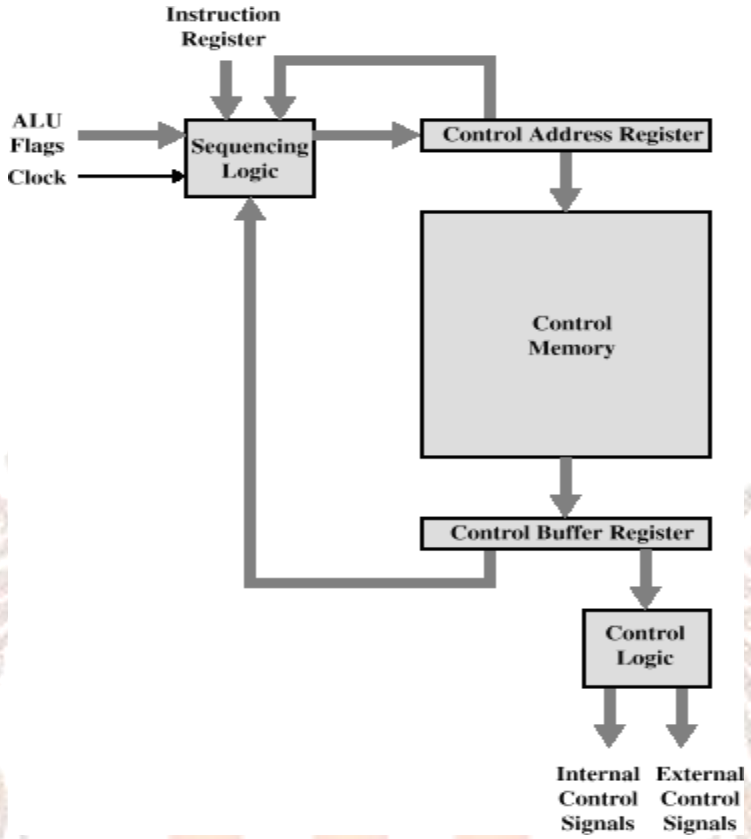

Fig 5.4 (b): Control Unit Organization

Control unit Inputs

The possible inputs for the control units are:

Clock: The control unit uses the clock to maintain the timings. The control unit causes one micro-instruction (or set of parallel micro-instructions) per clock cycle. This is sometimes referred to as the processor clock time or the clock cycle time.

Instruction register: Op-code of the current instruction is used to determine which micro-instructions to be performed during the execution cycle.

Flags: These are needed by the control unit to determine the status of the CPU and the outcome of previous ALU operations.

Example: As seen earlier the instruction ISZ, which is increment and skip if zero, the control unit will increment the PC if the zero flag is set.

Control signals from the control bus: the control bus portion of the system bus provides signals to the control unit, such as interrupt signals and acknowledgments.

Control Signals – output

The following are the control signals which are the output of the control unit:

Control signals within CPU: There are two types

1. Signals that cause data to be moved from one register to another.
2. Signals that activate specific ALU functions

Control signals to control bus: There are two types

1. signals to memory
2. signals to I/O modules

Control unit operation

Let us consider an example again a fetch cycle to study how the control unit maintains the control during this fetch cycle.

The control unit keeps track of where it is in the instruction cycle. At a given point, it knows that the fetch cycle is to be performed next. The first step is to transfer the contents of the PC to the MAR. The control unit does this by activating the control signal that opens the gate between the bits of the PC and the bits of the MAR. the next step is to read a word from memory into MBR and increment the PC. The control unit does this by sending the following control signals simultaneously.

1. A control signal that opens the gate allows the contents of the MAR onto the address bus.
2. A memory read control signal on the control bus.
3. A control signal that opens gates allows the contents of the data bus to be stored in the MBR.
4. Control signals to logic that adds 1 to the contents of the PC and stores the result back in the PC.

Following this, the control unit opens the gates between the MBR and the IR. This completes the fetch cycle. Now the control unit must decide whether to perform an indirect cycle or an execute cycle next. To decide it examines the IR to see if any indirect memory reference is made. The indirect and interrupt cycle work in a similar manner. For the execute cycle, the control unit begins by examining the opcode and on the basis of that decides which sequence of micro-operations to perform for that execute cycle.

3.3 Data Paths and Control Signals

To illustrate the data paths and control signals we will consider an example as shown in figure 5.5. This is a simple machine whose CPU has a single accumulator.

The control unit receives inputs from the clock, the IR, and the flags. With each clock cycle, the control unit reads all of its inputs and emits a set of control signals. Control signals go to three separate destinations.

- ALU: the control unit controls the operation of ALU by a set of control signals. This signal activates the various logic devices and gates within the ALU.
- Data paths: the control unit controls the internal flow of data. For each data path to be controlled there is a gate indicated by a circle in the figure 5.5. A control signal temporarily opens the gate to let data pass.

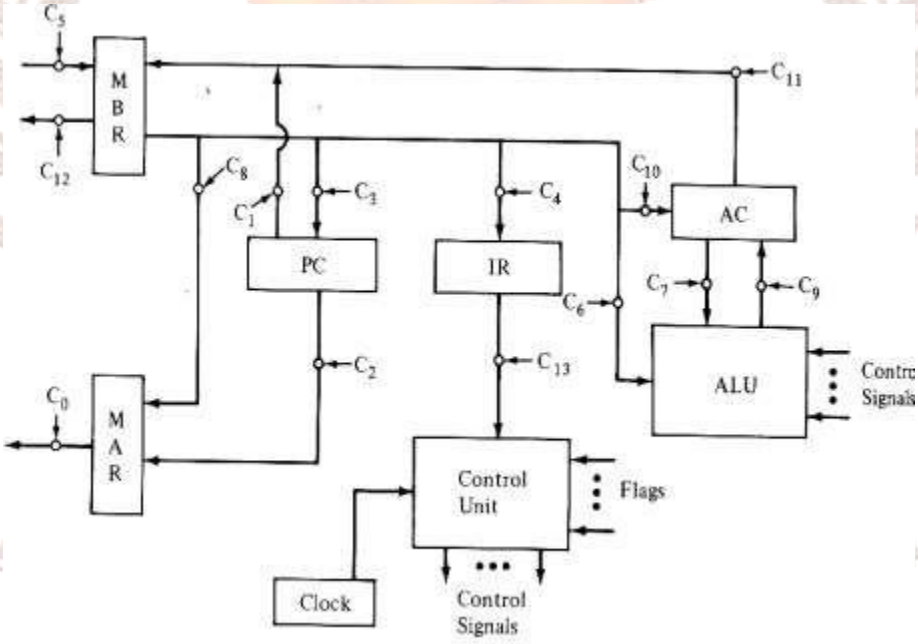

Fig 5.5: Data paths and control signals

Table 5.2: Micro-Operations and Control Signals

Micro-Operations	Timing	Active Control Signals
Fetch:	t1: MAR <-- PC t2: memor <-- MAR t3: MBR <-- memory t4: IR <- MBR t5: PC <- PC +1	C_2 C_5 , C_R C_4
Indirect:	t1: MAR <- ($IR_{address}$) t2: MBR <- (Memory) t3: IR($_{address}$)<- ($MBR_{address}$)	C_8 C_5 , C_RC_4
Interrupt:	t1: MBR <- (PC) MAR <- save-address t2: PC <- routine-address memory <- (MBR) t3:	C_1 C_{12} , C_W
Where C_R = Read control signal to system bus C_W = Write control signal to system bus		

Example: In the instruction fetch, as shown in table 5.2 the contents of MBR are transferred to the IR. And this transfer is activated by the control signal C4 given in table 5.2 and shown in figure 5.5. Thus control unit controls everything with few control signals sent to registers or modules (ALU) which are within the CPU and few control signals sent to the system bus.

Self-Assessment Questions – 2

5. _____are used to move data between the registers and ALU.
6. The control unit uses_____to maintain the timings.
7. _____are needed by the control unit to determine the status of the CPU.

4. DATA PATH INSIDE A CPU

In the earlier section, we have seen the data paths and the control signal for a very simple machine. The complexity of the organization depends on the CPU bus also. Here we examine the main data paths inside a CPU with three number of internal Bus. They are:

- CPU with Single Bus Structure
- CPU with Two Bus Structure
- CPU with Three Bus Structure

4.1 Single Bus Structure

Figure 5.6 illustrates the single bus organization of the data paths inside the CPU. In this case, ALU and all CPU registers are connected through a common bus. This bus is internal to the CPU and is different from the external buses which are used to connect the CPU with other units of the computer system.

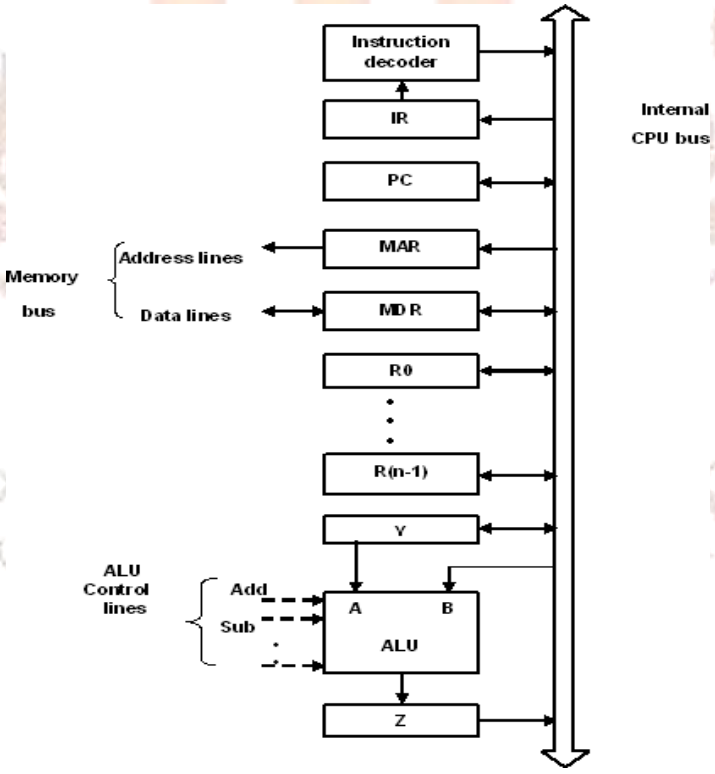

Fig 5.6: Single Bus Organization of Data Path inside CPU

MAR and MDR registers communicate with the main memory through the address bus and data bus. The registers R0 to R(n–1) are provided for general purposes. Some of them can be used as special-purpose registers such as stack pointers or index registers. Registers Y and Z are used only by the CPU for temporary storage during the execution of some instructions. They are never used for storing the data generated by one instruction for later use by another instruction.

Most of the CPU operations mentioned above in steps 1 to 3 can be carried out by performing one or more of the following functions in some pre-specified sequence.

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

Now we will discuss in detail how the above functions are implemented in a typical computer.

Fetching a word from the memory

To fetch a word of information from the memory CPU must specify the address of the memory location where this word is located and request a read operation. This includes whether the information to be fetched represents an instruction in a program or an operand specified by the instruction. CPU transfers the address of the required information to the Memory Address Register (MAR) from where it is transferred to the main memory through the address lines of the main memory. In the same period CPU uses the control lines of the memory bus to indicate that a read operation is required. After issuing this request CPU waits until it receives feedback from the memory indicating that the requested function has been completed. This is done using another control signal on the memory bus, referred to as Memory Function Completed (MFC).

The memory sets this signal to 1 to indicate that the contents of the specified location in the memory have been read and are available on the data lines of the memory bus and thus available for use inside the CPU. This completes the memory fetch operation. The transfer mechanism where one device initiates the transfer and waits until the other device response

is called asynchronous transfer. This mechanism enables the transfer of data between two independent devices that have different speeds of operation.

Consider an example, assume that the address of the memory location to be accessed is in register R0 and data is to be loaded into register R1. The following sequence of operations is used to achieve this.

1. MAR ← [R0]
2. Read
3. Wait for MFC signal
4. R1 ← [MDR]

The duration of step 3 depends on the memory access time. Usually, the access time of the memory is longer than the time required for performing any single CPU operation. Hence the overall time of instruction can be reduced by organizing a sequence of operations to be performed in the CPU while waiting for the memory to respond. Functions which do not require the use of MDR and MAR can be carried out during this time. For example, PC can be incremented while waiting for the MFC signal.

In many computers, an alternative Synchronous Transfer is used. In this case, one of the control lines of the bus carries clock pulses, which provide timing signals to the CPU and the main memory. A memory operation can be completed in one clock cycle. The synchronous bus scheme leads to simpler implementation, but it cannot accommodate devices of widely varying speed. So the speed of all the devices must be reduced to that of the slowest one.

Storing a Word into the Memory

First, the address is loaded into the MAR and the data to be stored is loaded into the MDR register before or at the same time as the write command is issued.

The following sequence illustrates the write operation assuming that the data to be stored in the memory is in R1 and the memory address is in R0.

1. MAR ← [R0]
2. MDR ← [R1], write.
3. Wait for the MFC signal.

Here also during the wait period operations which do not require registers of MDR and MAR can be performed. Steps 1 and 2 can be carried out in a two-bus structure if the two transfers do not use the same data path.

Register Transfers

For transferring the data between the blocks of CPU, input and output gating must be provided. Signals Riin and Riout control the input and output gates for register Ri. When Riin is set to 1, the data available on the common bus is loaded into Ri and when Riout is set to 1, the contents of Ri are placed on the bus. When Riout is set to 0, bus can be used for transferring the data from other registers.

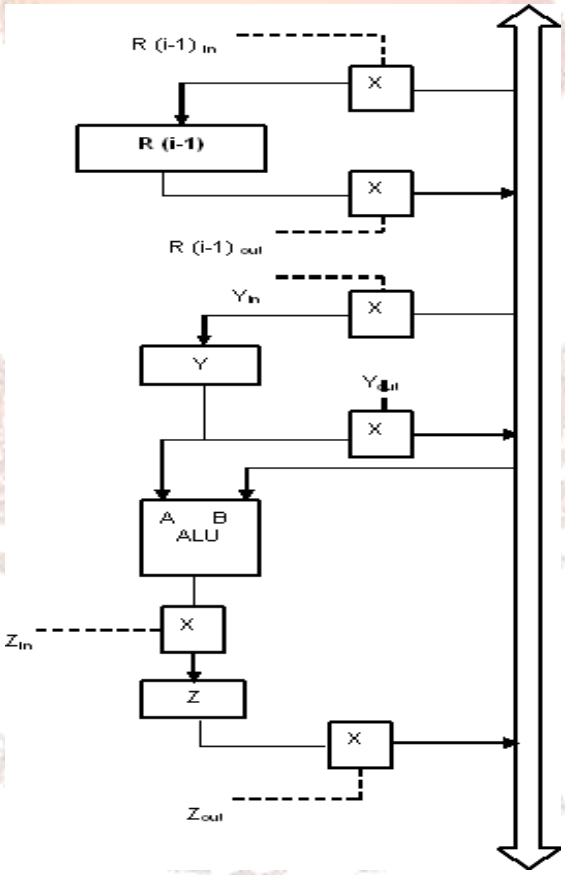

Fig 5.7: Input-Output Gating for the Register

As an example consider a data transfer from register R0 to R1. To achieve this following sequence of operations are to be performed.

- Set R0out to 1, which enables the output gate of register R0. This places the contents of R0 on the CPU bus.

- Set R1 into 1, which enables the input gate of register R1. This loads the content of R0 into the register R1from the CPU bus.

Performing an Arithmetic or Logic Operation

The ALU performs arithmetic and logic operations. It has no internal storage. Therefore, to perform an operation two operands should be brought at two inputs of the ALU.

Registers Y and Z are used for this purpose. Register Y is used to hold one of the two operands while the other is gated onto the bus. The result is stored in register Z.

Consider an add operation to add content register R0 and register R1 and store the result in register R2. The sequence of operations is as follows:

Step	Action
1	$R0_{out}$, Y_{in}
2	$R1_{out}$, add,Z_{in}
3	Z_{out}, $R2_{in}$

The signals for the corresponding speed names are set to 1, for the signal corresponding to that step and all other signals are inactive. In step 1 output gate of register, R0, and gate of register Yin are enabled, causing the content of R0 to be transferred to Y. now the content of Y is available at input A. In step 2, the contents of register R1 are gated onto the bus and are available at input B. Add line is set to 1, to add the contents of inputs A and B. Input gate of Z is enabled and the sum is loaded into temporary register Z. In step 3, the contents of register Z are transferred to the destination register R2.

Register Input-Output Gating

Now we will discuss in detail about the implementation of input and output gating required for transferring data to and from the common bus. Consider a case where each bit of register shown in figure 5.6 and figure 5.7 consists of a flip-flop as shown in figure 5.8. The flip-flop is assumed to be one of the bits of register Z. when the control input is set to 1, the state of the flip-flop changes corresponding to the data on the bus and data stored in the flip-flop. When the state of Z changes from 1 to 0, this data is locked until Z is again set to 1. Thus two input gates of the flip-flop form a control switch as shown in figure 5.7.

Conceptually the output switches act as mechanical switches. When the switch is in on state it transfers the contents of its corresponding register to the bus. When it is in off state register output is electrically disconnected from the bus, allowing another register to place data on the bus. Thus output of the register switch can be in three states i.e. 0, 1, or open circuit state.

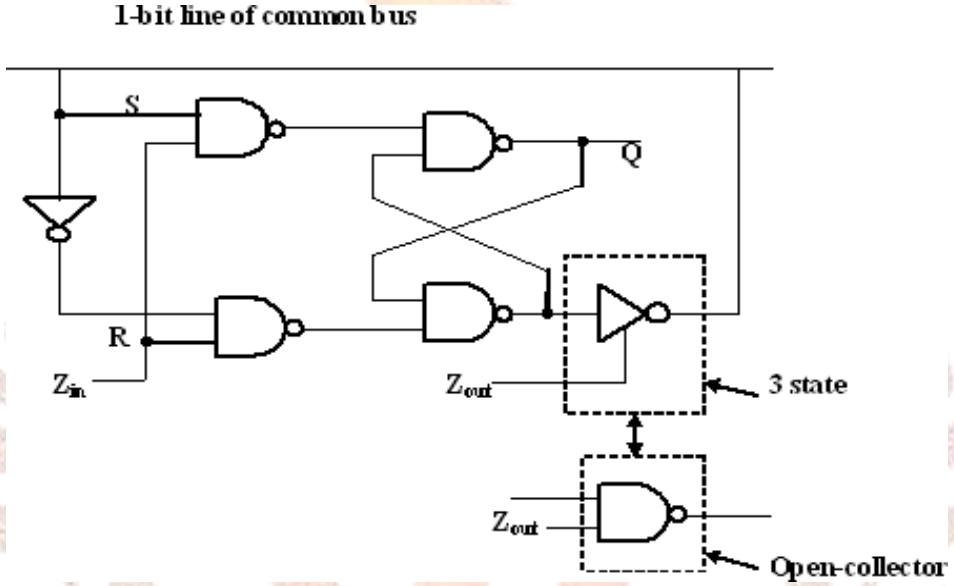

Fig 5.8: Input and Output Gating for One Register Bit

In practical implementations, electronic switches are used. Since it supports 3 state possibilities it is said to have a three-state output. A separate control unit is used either to enable gate output to drive the bus to 0 or 1, or to put a high impedance (electrically disconnected) state.

An alternative design for the common bus makes use of an open-controller for a bipolar or open-drain for MOS gates. It does not require output switches. If an open controller arrangement is used, the three-state output gates of the figure.

5.8 is replaced by an open controller NAND gate. When Zout is high the bit stored in the latch is fed to the bus, when Zout is low, the data from another register can be transferred to the bus. The three-state designs enable faster data transfers than the open controller approach.

Timing of Data Transfers between Registers

Now we will consider some aspects of the timing of data transfer inside the CPU. For example, consider the addition operation discussed in the previous section. After the signal R1out is set to 1 following delays will be observed.

- A finite delay is encountered for the gate to open.
- For the data to travel along the bus to the input of ALU.
- ALU adder circuits.
- For the result to be properly stored in register Z, data must be maintained on the bus for an additional period of time equal to the setup and hold times for this register.

The timing diagram is shown in the figure. 5.9. The sum of five delay times is the minimum duration of the signal R1out.

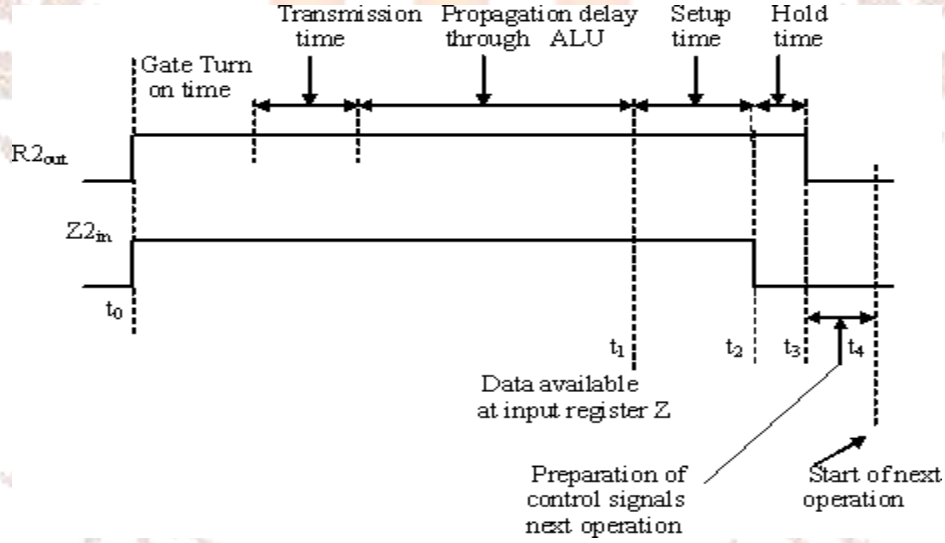

Fig 5.9: Timing of the Control Signals during the Add Step

4.2 Two Bus Structure

An alternative arrangement for the single bus organization is the two bus structure as shown in figure 5.10. Here all the register outputs are connected to bus A, and all register inputs are connected to bus B. The bus tie G connects two buses together. When G is enabled, it transfers data on bus A to bus B, and when it is disabled two buses are electrically disconnected. The temporary register Z is not required in this organization because, when bus tie G is disabled, the output of ALU can be transferred directly to the destination register.

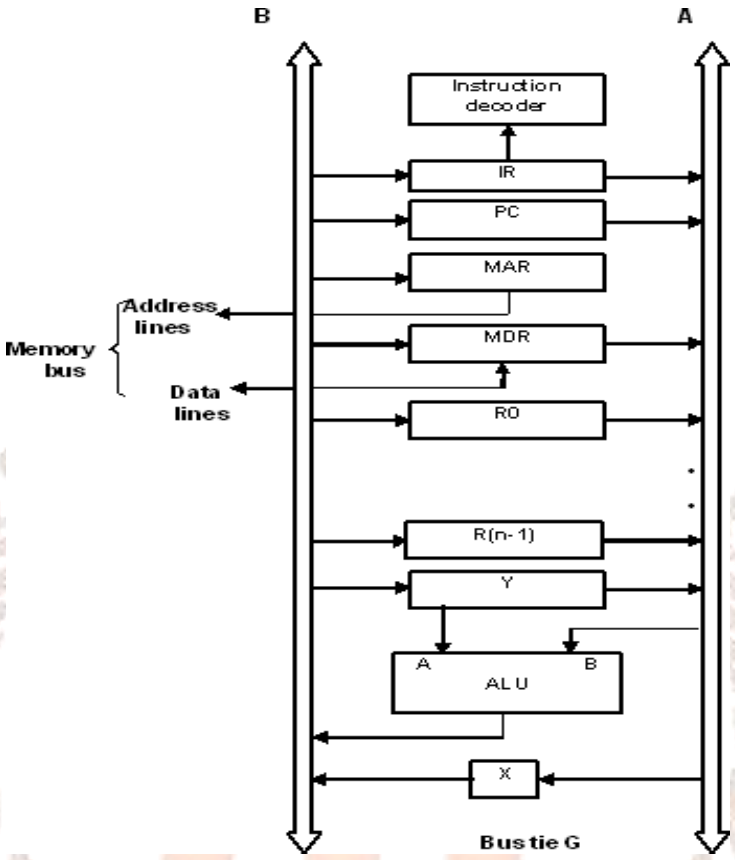

Fig 5.10: Two Bus Structure

Now the addition operation discussed in the previous section can be performed as follows:

1. If the registers are simple latches, the destination register in the above sequence must be different from R1, because two operations R1in and R1out cannot be done at the same time. However, by interchanging R0out and R1out and replacing R2in by R1in in step 2, the operation R1 ← [R0] + [R1] can be performed
2. R0out, Genable,Yin
3. R1out, add, ALUout, R2in

4.3 Three Bus Structure

Consider one more CPU organization having three bus structures as shown in figure 5.11. Here each bus is connected to only one output and a number of inputs.

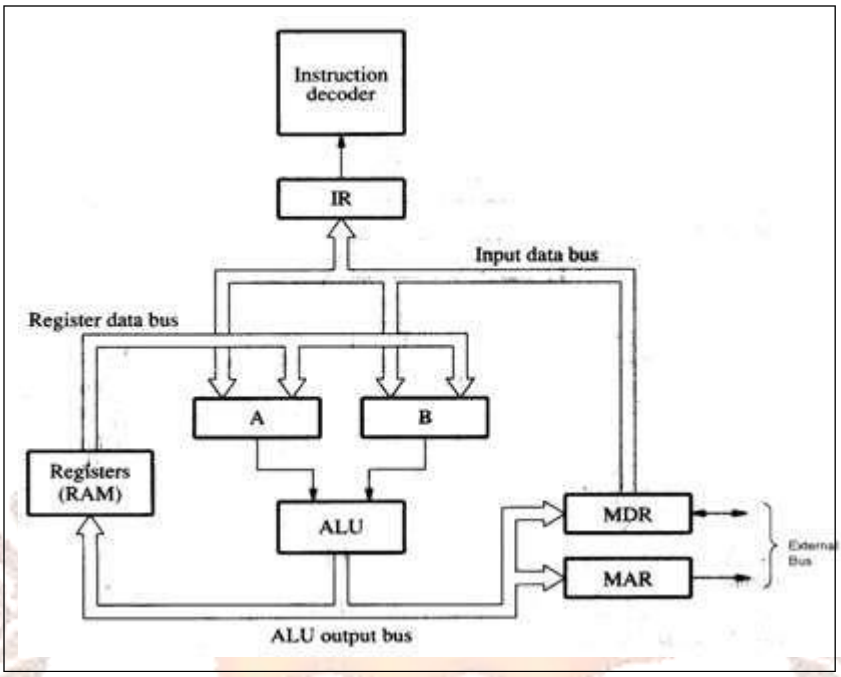

Fig 5.11: Three Bus Structure

Two registers A and B are provided, which can be loaded from either the input data bus or the register data bus. Three data transfers can take place at the same time. Components connected to each bus are less when compared to two bus structure. Hence data transfer will be faster in this case.

In this case, general-purpose registers of the CPU are shown in a single block and they are assumed to be implemented using a RAM unit. These RAMs are different from the main memory RAMS that constitute the main memory of the computer.

4.4 Execution of A Complete Instruction

Now we will try to put together the sequence of elementary operations required to execute one instruction.

For example, consider the instruction to add the content of memory location NUM to register R0. Let the address NUM is given explicitly in the address field of instruction. Table 5.3 illustrates the control sequence for the execution of the instruction.

Table 5.3: Control Sequence for the Execution of the Instruction

Step	Action
1	PC_{out} , MAR_{in}, read, clear Y,
	Set carry-in to ALU, add, Z_{in}
2	Z_{out}, PC_{in}, wait for MFC
3	MDR_{out}, IR_{in}
4	Address field of IR_{out}, MAR_{in}, read
5	$R0_{out}$,Y_{in}, wait for MFC
6	MDR_{out}, add, Z_{in}
7	Z_{out},$R0_{in}$, end

To execute this instruction following actions are performed

1. Fetch the instruction.
2. Fetch the content of the memory location pointed by the address field of the instruction.
3. Perform the addition.
4. Load the result in R0.

The instruction execution procedure is explained below:

Step 1: Contents of PC are loaded into MAR and the read request is sent to the main memory. While waiting for the MFC signal PC is incremented by setting one of the current values in the PC. Meanwhile, carry-in to ALU is set to 1 and an add operation is specified.

Step 2: The updated value from register Z is sent back to PC. This step can be started immediately after issuing the read request without having to wait for the MFC signal.

Step 3: Wait for the MFC signal. Here word fetched from the memory is loaded into the IR. This completes the fetch phase of the operation.

Step 4: The instruction decoding circuit interprets the contents of IR at the beginning which enables the control circuitry to choose the appropriate signals for the remainder of the control sequence. Here address field of IR, which contains the address NUM, is gated to the MAR and memory read operation is initiated.

Step 5: Transfer the content of R0 to register Y and wait for the MFC signal.

Step 6: Now the memory operand is available on register MDR. The addition operation is performed.

Step 7: The result is transferred to R0. This completes the execution phase of the operation. The End Signal indicates that this is the last step of the current instruction and causes a new fetch cycle to be started going back to step1.

4.5 Branching

Branching is achieved by replacing the current contents of the PC by the branch address. The branch address is obtained by adding an offset X given in the address field of the branch instruction, to the current value of PC. Table 5.4 shows the control sequence for an unconditional branch using a single bus organization.

Table 5.4: Control Sequence for an Unconditional Branch Instruction

Step	Action
1	PC_{out} , MAR_{in}, read, clear Y, set carry-in to ALU, add, Z_{in}
2	Z_{out}, PC_{in}, wait for MFC
3	MDR_{out}, IR_{in}
4	PC_{out}, Y_{in}
5	Address field of IR_{out}, Add, Z_{in}
6	Z_{out}, PC_{in}, End

In step 4, IR will be decoded in the beginning. To execute branch instruction, the contents of the PC are transferred to register Y.

In step 5, the offset X is gated onto the bus, and the addition operation is performed which forms the branch address.

In step 6, the branch address is loaded into the PC.

Here PC is incremented during the fetch phase, before knowing the type of instruction being executed. At the same time, the offset X is added to the contents of the PC. Hence offset X is the difference between the branch address and the address immediately following the branch instruction.

For example, if the branch instruction is at location 3000 and it is required to branch to location 3050, the value of the X must be 49.

Now consider the conditional branching. In this case, the status of the condition codes must be checked before loading a new value into the PC. For example, for a branch on negative instruction, step 4 is replaced with

PC_{out}, Y_{in}, if N then End

As earlier, the contents of the PC are copied into register Y, just in case, they will be needed to compute the branch operation. Meanwhile, the N bit is checked. If it is zero the execution is terminated using End Signal. If N is 1, the remaining steps are executed.

Self-Assessment Questions – 3

8. _____communicate with main memory through address bus anddata bus.
9. _____are never used for storing the data generated by oneinstruction for later use by another instruction.
10. In _____ one of the control lines of the bus carries clock pulses,which provide timing signals to the CPU and the main memory.
11. The _____performs arithmetic and logic operations.
12. ALU has _____ internal storage
13. Contents of PC are loaded into MAR is the _____ step of instruction execution

5. SEQUENCING OF CONTROL SIGNALS

As discussed in section 5.3.1, the functional requirement of the control unit is sequencing and then execution. Sequencing is very necessary to execute the instructions. That is the CPU must generate the control signals in the proper sequence.

Two techniques to solve this problem are:

1. Hardwired control
2. Micro programmed control

For example: For the sequence of control signals given in table 5.3, seven non-overlapping signals are required to execute the instruction. Each time the slot must be long enough for the functions specified in the corresponding step to be completed.

5.1 Hardwired Control Unit

In a hardwired implementation, the control unit is essentially a combinatorial circuit. Its input logic signals are transformed into a set of output logic signals, which are the control signals.

The control unit organization is as shown in figure 5.12.

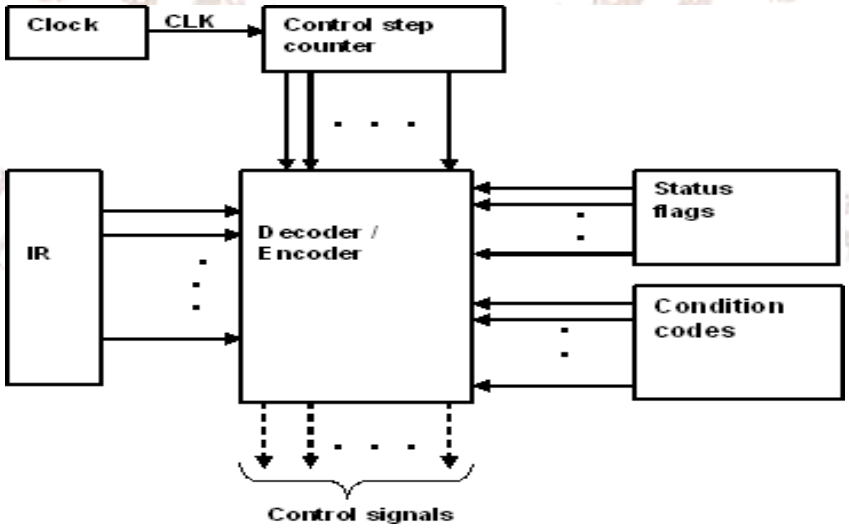

Fig 5.12: Control Unit Organization

The key inputs to the control unit are IR, clock, Flags, and control bus. The required control signals are determined based on the following information.

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

We will see the inputs one by one in the following discussion.

First, consider the instruction register. The control unit makes use of the opcode and will perform actions accordingly.

Instruction register

- Op-code causes different control signals for each different instruction
- Unique logic for each op-code
- The decoder takes encoded input and produces a single output
- *n* binary inputs and 2n outputs

Here the 2n output pattern of the decoder will activate a single unique output. In most flags and control bus signals are directly useful to the control unit.

Flags and control bus

- Each bit of these register has some meaning which is used by the control unit

Now we will see the clock of the control unit. The function of the clock module is given below;

Clock

- It issues a repetitive sequence of pulses
- Useful for measuring the duration of micro-ops
- Must be long enough to allow signal propagation
- Different control signals at different times within the instruction cycle
- Need a counter with different control signals for t1, t2 etc.

Next, we will see the function of Control unit signals are emitted at different time units within a single instruction cycle. The counter as input also would be of use for getting different control signal being at different times.

Hardwired Implementation

Now we will separate decoding and encoding functions to understand the structure of the control unit more clearly. Figure 5.13 shows the separation of encoding and decoding units.

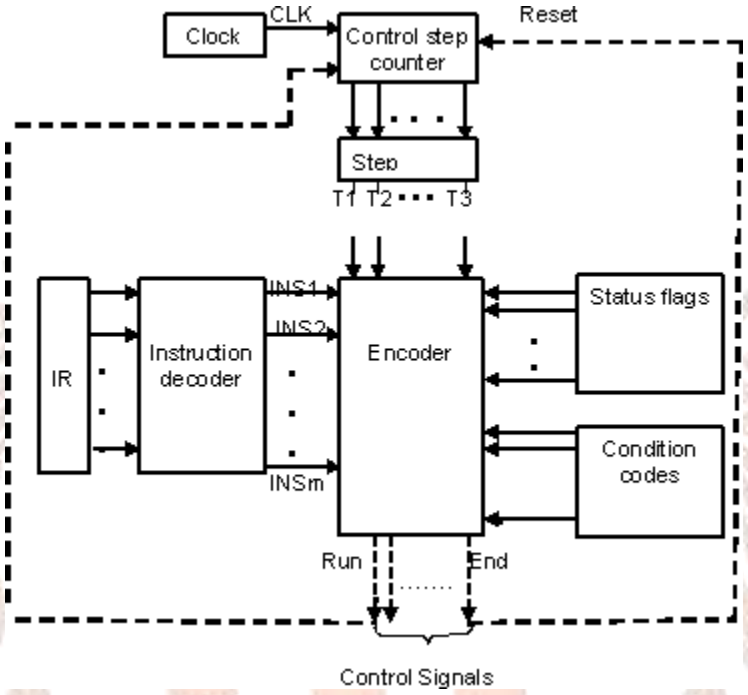

Fig 5.13: Separation of the Encoding and Decoding Functions

The step encoder provides a separate signal line for each time slot in the control sequence. Similarly, the output of the instruction decoder consists of separate lines for each instruction. When the instruction is loaded into IR, one of the output lines INS1 to INSm is set to 1 and all the other lines are set to 0. All the input signals to the encoder are combined to generate control signals such as Yin, PCout, add, end and etc.

1. The control signal Zin is generated by the logic function

   Zin= T1+ T6.ADD + T5.BR +…

Zin is turned on during the time slot T1 for all instructions; during T6 for add instructions, and so on. Time T1 is common to all instructions because it occurs during the fetch phase. The Combinational circuit for the generation of this signal is shown in figure 5.14

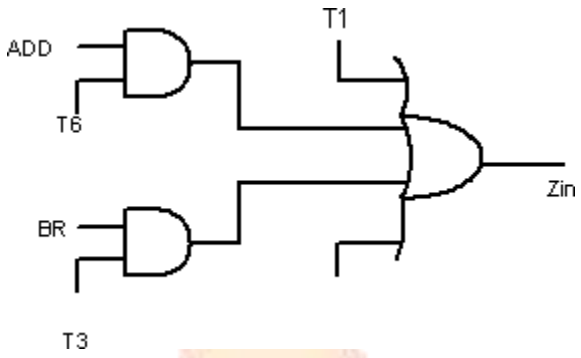

Fig 5.14: Generation of Zin Control Signal

2. Similarly following logic function generates the End Function.

End = T7. ADD +T6.BR + (T6.N + T4. $\bar{N}$).BRN+·

Gating required to generate End are shown in figure 5.15

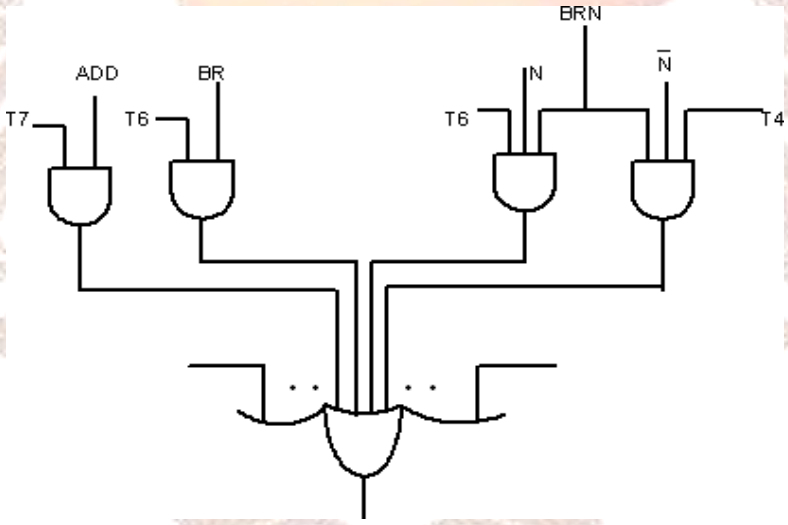

Fig 5.15: Generation of the End Signal

Problems with Hardwired Designs

- Complex sequencing & micro-operation logic
- Difficult to design and test
- Inflexible design
- Difficult to add new instructions

5.2 Micro-Programmed Control

An alternative approach which is quite common is Micro-Programmed Control unit. The control signals and each micro operation is described using a programming language called micro programming language. Micro program is midway between the hardware and software.

Definition of some frequently used terms

Control Word (CW): A word whose individual bits represent the control signals. Therefore a control step in the control sequence of an instruction is a unique combination of 1s and 0s in the control word.

Micro routine: A sequence of control words corresponding to the control sequence of a machine instruction constitutes the micro routine for that instruction.

Micro program Memory: it is a special memory in which the micro routines corresponding to the instruction set of a computer are stored. The control unit can sequentially generate the control signals for any instruction and also the control words of the corresponding micro routine.

Micro program Counter: This is used to read the control words sequentially from the microprogram memory.

Basic Concepts

The basic structure of microprogrammed control unit is shown in Figure 5.16 (a) and

detailed micro-programmed control unit is shown in figure 5.16 (b).

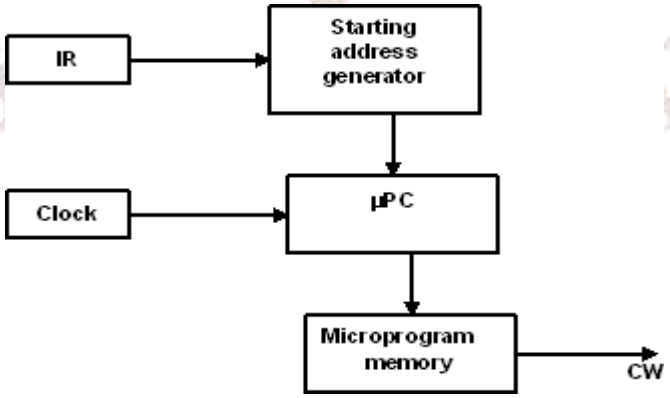

Fig 5.16 (a): Basic Organization of a Microprogrammed Control Unit

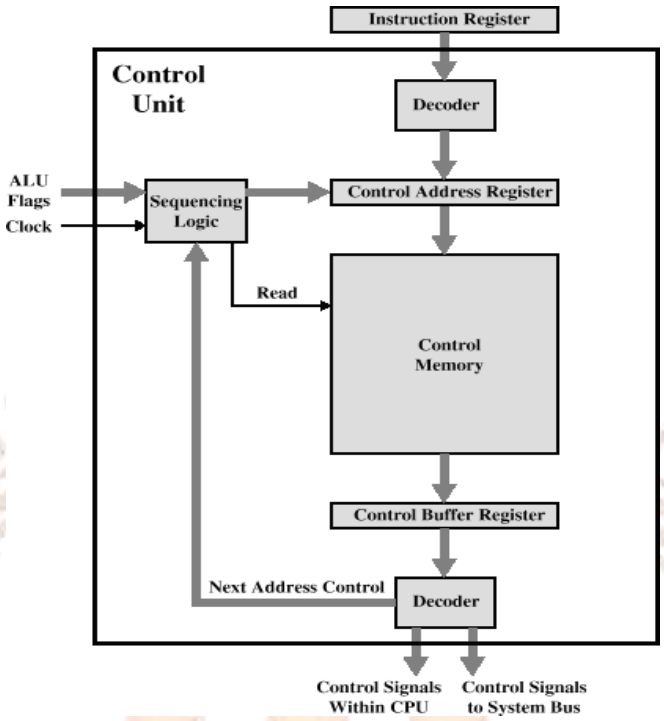

Fig 5.16 (b): Microprogrammed Control Unit

When a new instruction is loaded into the IR, the starting address is loaded into the PC from the starting address generator block. Automatically, the PC is incremented by the clock, causing successive microinstructions to be read from the memory. Thus, the control signals are loaded into various parts of the CPU in the correct sequence.

Another important function of the control unit is to check for the status of the condition codes or status flags while branching. In the case of hardwired control, this situation is handled by including an appropriate logic function in the encoder circuitry whereas, in the case of microprogrammed control, the microinstruction set is expanded to include some conditional branch microinstructions. These microinstructions can also specify the status flags, conditions for branching, and the branch address.

Example: The control words corresponding to steps 5, 6, and 7 of Table 5.4 discussed in section 4.5 Branching, can be written as shown in Table 5.5.

Table 5.5: Example Microinstructions for table 5.4

Step	Ro_{in}	R_{out}	Y_{in}	Y_{out}	Z_{in}	Z_{out}	MDR_{in}	MDR_{out}	WMFC	Add	End
5	0	1	1	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	1	0	1	0
7	1	0	0	0	0	1	0	0	0	0	1

Now consider table 5.6 that shows the implementation of a micro routine for a branch on negative instruction.

Table 5.6: Micro routine for a branch on negative instruction.

ADDRESS	MICROINSTRUCTION
0	PC_{out}, MAR_{in}, read, clear Y, set carry-in to ALU, add, Z_{in}
1	Z_{out}, PC_{in}, wait for MFC
2	MDR_{out}, IR
3	Branch to starting address of appropriate microroutine.
	.....
32	PC_{out}, Y_{in}, if N then branch to 0.
33	Address field of IR_{out}, ADD, Z_{in}
34	Z_{out}, PC_{in}, End

Figure 5.17 shows the modification required to enable conditional branching in the microprogram. The branch conditions bits of the microinstruction word, contents of IR, status flags, and branch address are fed to the starting and branch address generator block. This block loads a new address into the PC when instructed to do so by a microinstruction. Therefore PC is incremented every time a new microinstruction is fetched from the microprogram memory, except for the following situations.

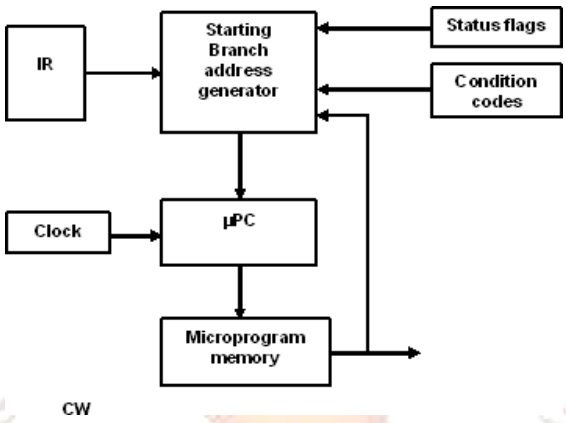

Fig 5.17: Organization of the Control Unit to Enable Conditional

Branching in the Microprogram

1. When an End Microinstruction is encountered, the PC is loaded with the address of the first CW in the micro routine 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 micro routine for that instruction.
3. When a branch instruction is encountered and the branch condition is satisfied, the PC is loaded with the branch address.

Some important points to be noted regarding microprogrammed control are:

1. The microprogram defines the instruction set of a computer. Therefore instruction set can be changed simply by changing the contents of the microprogram memory which offers considerable flexibility in to design a computer.
2. Since the contents of the microprogram memory are not changed at all, a read-only memory can be used for that purpose.
3. Since the execution of any machine instruction involves a number of fetches from the microprogram memory, the speed of this memory plays a major role in determining the overall speed of the computer.
4. If the entire CPU is fabricated as a single chip, the microprogram ROM is a part of that chip.

Self-Assessment Questions – 4

14. The required control signals are determined based on the content of _____
15. Each bit of_____register has some meaning which is used by thecontrol unit
16. Clock must be long enough to allow_____.
17. The micro program defines the_____of a computer.

6. SUMMARY

In this unit we studied the most complex aspects of ALU and control unit. The control unit is the portion of the processor that actually causes things to happen. The execution of one instruction consists of one or more fetch and one execute cycle. In the fetch phase the contents of the memory location pointed to by the program counter are fetched and loaded into the instruction register. In the execution phase, the actions specified by the instruction in the IR are carried out. The various blocks of CPU can be organized and interconnected in three different ways i.e. single bus organization; two-bus organization and three-bus organization. We have also introduced the two approaches used to implement the control units of a CPU are hardwired control and microprogrammed control.

7. TERMINAL QUESTIONS

1. What are the Micro-operations? Explain the Micro operations of the Fetch cycle.
2. What are the functions of the control unit? Explain.
3. Discuss the significance of Data Path and Control Signal.
4. Explain the process of fetching a word from the memory.
5. Differentiate Hardwired Control and Micro-programmed Controls.

8. ANSWERS

Self-Assessment Questions

1. instruction cycle
2. Fetch and execute
3. micro operations
4. Program Counter (PC)
5. Internal data paths
6. clock
7. Flags
8. MAR and MDR registers
9. Temporary registers
10. Synchronous Transfer
11. ALU
12. no
13. first
14. control counter, IR and flags
15. Flags and control bus
16. signal propagation
17. instruction set

Terminal Questions

1. Fetch and execute cycles always occurs in every instruction cycle. Each smaller cycle involves a series of steps called micro-operations. Refer to section 2 for more details.
2. Sequencing and Execution are the two most important functions of control unit. Refer to section 3.1 for more details.
3. The control unit receives inputs from the clock, the IR, and the flags. With each clock cycle, the control unit reads all of its inputs and emits a set of control signals. Refer to section 3.3 for more details.
4. To fetch a word of information from the memory CPU must specify the address of the memory location where this word is located and request a read operation. Refer to section 4.1 for more details.

5. In a hardwired implementation, the control unit is essentially a combinatorial circuit. An alternative approach which is quite common is Micro-Programmed Control unit. Micro program is midway between the hardware and software. Refer to sub-sections 5.1 and 5.2 for more details