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 interconnect them. One such organization is shown in the Figure 5.6.

In this case, the arithmetic 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 5.6 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 transparent to the user. Programmer cannot 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.

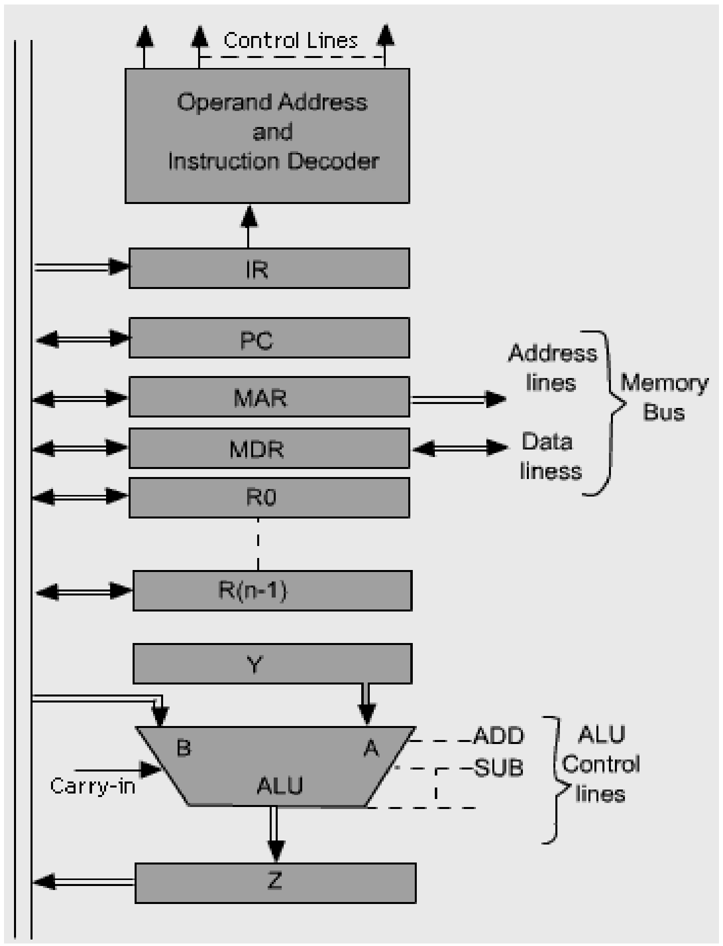

Figure 5.6: Single bus organization of the data path inside the CPU

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 pre-specified 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 arithmetic 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 transferred 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.

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 transferred 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 $\leftarrow$ [R2]   2. Read
3. Wait for MFC signal   4. R1 $\leftarrow$ [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 synchronized 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 (continuously 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 $\leftarrow$ [R2]
2. MDR $\leftarrow$ [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. During 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 $R_{i\ in}$ set to 1 the data available in the common bus is loaded into R_i . Similarly when, $R_{i\ out}$ is set to 1, the contents of the register R_i 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 R_1 to register R_2, the following actions are needed:

- Enable the output gate of register R_1 by setting R_{1out} to 1.
  -- This places the contents of R_1 on the CPU bus.
- Enable the input gate of register R_2 by setting $R_{2\ in}$ to 1.
  -- This loads data from the CPU bus into the register R_2.

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 carry 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. This organization is shown in the Figure 5.7.

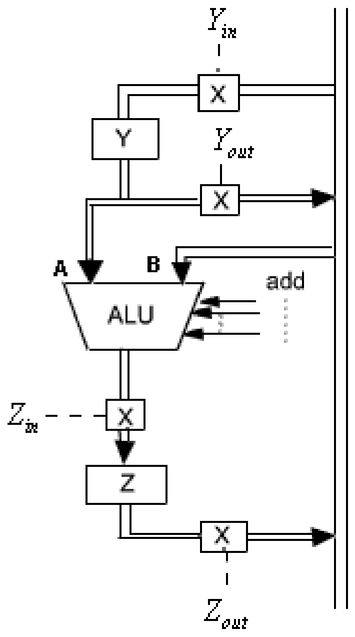

Figure 5.7: Organization for Arithmetic & Logic Operation.

Therefore, the sequence of operations to add the contents of register R_1 to register R_2 and store the result in register R3 should be as follows:

1. R_{1out}, Y_{in}
2. R_{2out}, Add, Z_{in}
3. Z_{out}, $R3_{in}$

In step 2 of this sequence, the contents of register R_2 are gated to the bus, hence to input –B of the ALU which is directly connected to the bus. The contents of register Y are always available at input A of ALU. The function performed by the ALU depends on the signal applied to the ALU control lines. In this example, the Add control line of ALU is set to 1, which indicate the addition operation and the output of ALU is the sum of the two numbers at input A and B. The sum is loaded into register Z, since the input gate is enabled (Z_{in}). In step 3, the contents of register Z are transferred to the destination register R_3.

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. A simple two bus structure is shown in the Figure 5.8.

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

1. $R1_{out}$, G_{enable}, Y_{in}
2. $R2_{out}$, Add, ALU_{out}, $R3_{in}$

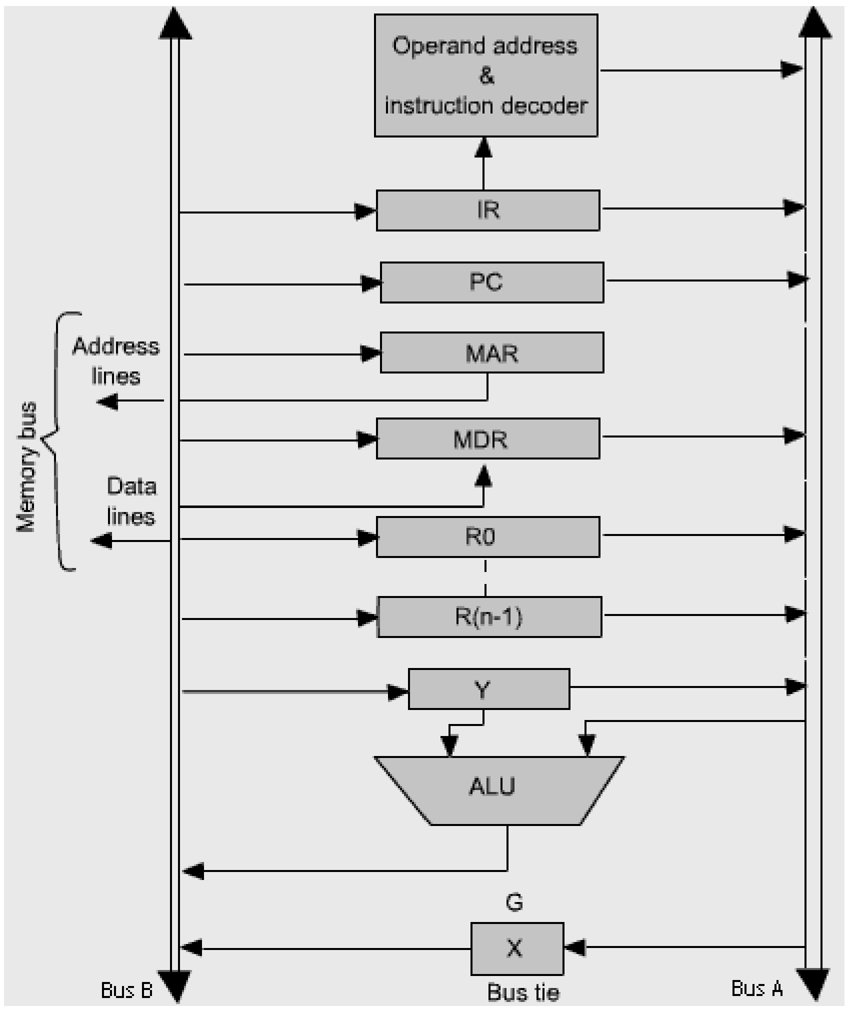

Figure 5.8: Two bus structure

In this case source register R2 and destination register R3 has to be different, because the two operations $R2_{in}$ and $R2_{out}$ cannot 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 5.9.

A multiplexer is provided at the input to each of the two working 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.

Like two bus organization, we can use bus tie to connect the input bus and output bus. When the bus tie is enabling, the information that is present in input bus is directly transferred to output bus. We may use one bus tie G1 between input data bus and ALU output bus and another bus tie G2 between register data bus and ALU output data bus.

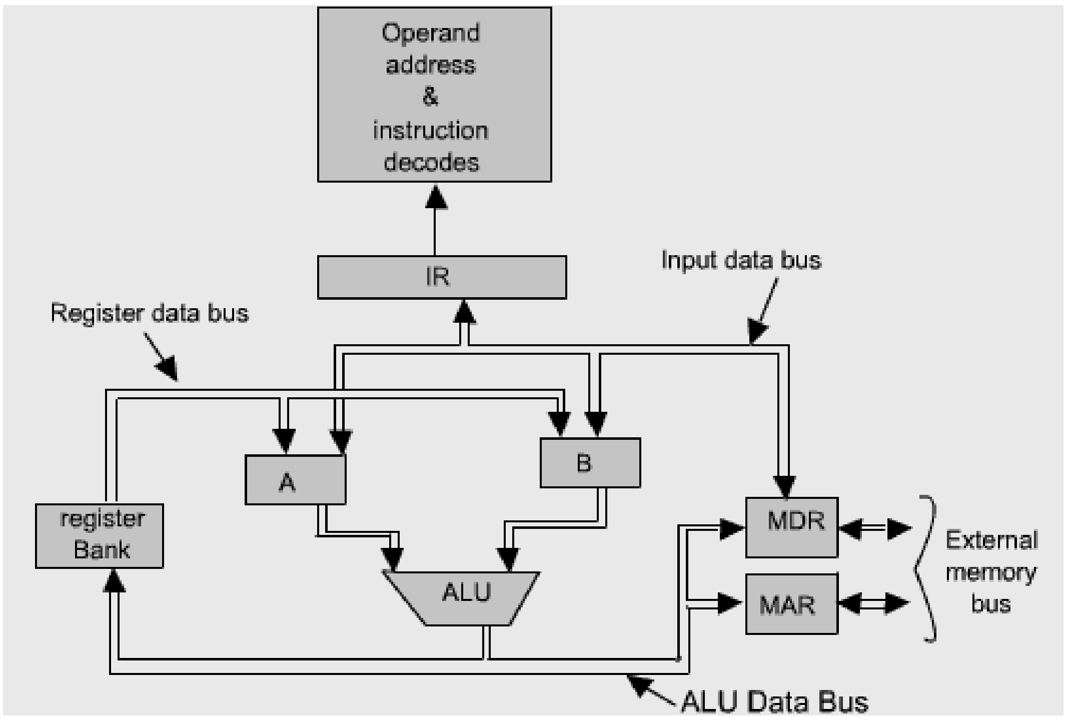

Figure 5.9 : Three Bus structure

Execution of a Complete Instruction:

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.

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.

Following sequence of control steps are required to implement the above operation for the single-bus architecture that we have discussed in earlier section.

Steps	Actions
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.	$R1_{out}$, Y_{in}, Wait for MFC
6.	MDR_{out}, Add, Z_{in}
7.	Z_{out}, $R1_{in}$
8.	END

Instruct ruction execution proceeds as follows:

In Step 1: 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 transferred to IR (Instruction Register), because it is an instruction. Step 1 through 3 constitutes 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 transferred 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

The control sequence that enables execution of an unconditional branch instruction using the single - bus organization is as follows:

Steps	**Actions**
1.	PCout, 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}
7.	End

Execution starts as usual with the fetch phase, ending with the instruction being loaded into the IR in step 3. To execute the branch instruction, the execution phase starts in step 4.

In Step 4: The contents of the PC are transferred to register Y.

In Step 5: The offset **X** of the instruction is gated to the bus and the addition operation is performed.

In Step 6: The result of the addition, which represents the branch address, is loaded into the PC.

In Step 7: It generates the End signal to indicate the end of execution of the current instruction.

Consider now the conditional branch instruction instead of unconditional branch. In this case, we need to check the status of the condition codes, between step 3 and 4. i.e., before adding the offset value to the PC contents.

For example, if the instruction decoding circuitry interprets the contents of the IR as a branch on Negative (BRN) instruction, the control unit proceeds as follows: First the condition code register is checked. If bit N (negative) is equal to 1, the control unit proceeds with step 4 through step 7 of control sequence of unconditional branch instruction.

If, on the other hand, N is equal to 0, and End signal is issued.

This in effect, terminates execution of the branch instruction and causes the instruction immediately following in the branch instruction to be fetched when a new fetch operation is performed.

Therefore, the control sequence for the conditional branch instruction BRN can be obtained from the control sequence of an unconditional branch instruction by replacing the step 4 by

4. If $\overline{N}$ then End
   If N then PC_{out}, y_{in}

Most commonly need conditional branch instructions are

BNZ	: Branch on not Zero
BZ	: Branch on positive
BP	: Branch on Positive
BNP	: Branch on not Positive
BO	: Branch on overflow