C h a p t e r **12 PROCESSOR STRUCTURE & FUNCTION**

➢ Internal Structure of CPU

To understand the organization of the processor, let us consider the requirements placed on the processor, the things that it must do:

• **Fetch instruction:** The processor reads an instruction from memory (register, cache, main memory).

• **Interpret 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 an I/O module.

• **Process data:** The execution of an instruction may require performing some arithmetic or logical operation on data.

• **Write data:** The results of an execution may require writing data to memory or an I/O module.

To do these things, it should be clear that the processor 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.

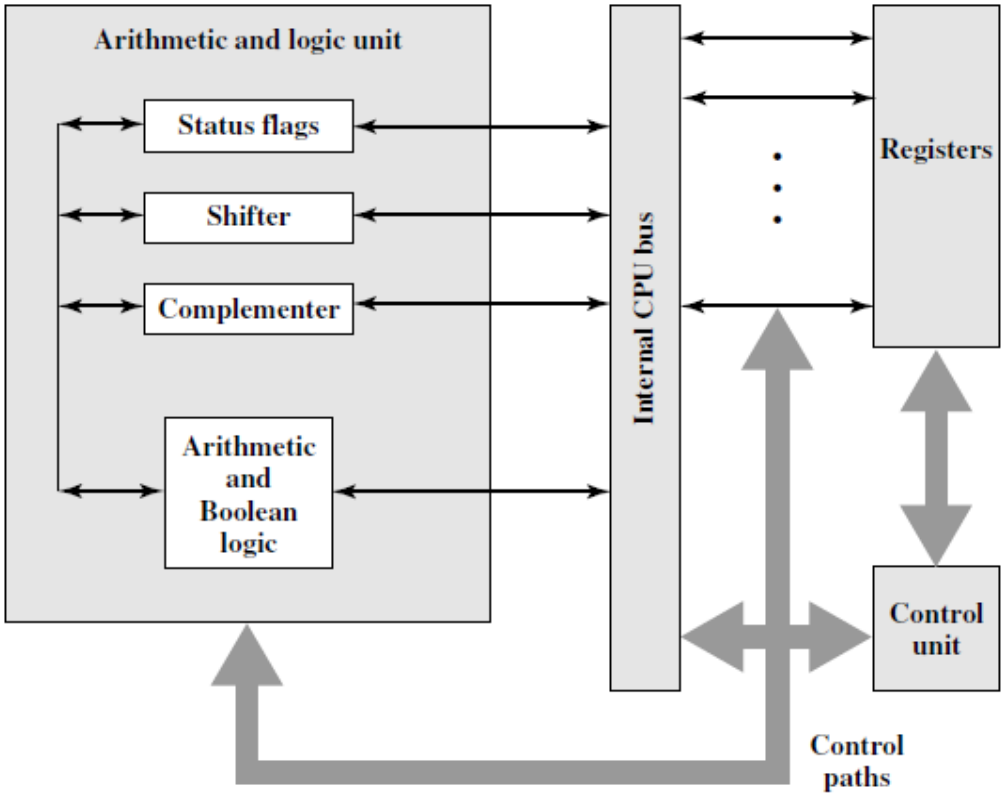


Figure 12.2 Internal Structure of the CPU

Register Organization

The registers in the processor perform two roles:

• **User-visible registers:** Enable the machine- or assembly language programmer to minimize main memory references by optimizing use of registers.

• **Control and status registers:** Used by the control unit to control the operation of the processor and by privileged, operating system programs to control the execution of programs.

User-Visible Registers

A user-visible register is one that may be referenced by means of the machine language that the processor executes. We can characterize these in the following categories:

• General purpose
• Data
• Address
• Condition codes

Data registers may be used only to hold data and cannot be employed in the calculation of an operand address.

Address registers may themselves be somewhat general purpose, or they may be devoted to a particular addressing mode.

Control and Status Registers

There are a variety of processor registers that are employed to control the operation of the processor. Most of these, on most machines, are not visible to the user. Some of them may be visible to machine instructions executed in a control or operating system mode.

Four registers are essential to instruction execution:

• **Program counter (PC):** Contains the address of an instruction to be fetched
• **Instruction register (IR):** Contains the instruction most recently fetched
• **Memory address registers (MAR):** Contains the address of a location in memory
• **Memory buffer register (MBR):** Contains a word of data to be written to memory or the word most recently read.

Common fields or flags include the following:

• **Sign:** Contains the sign bit of the result of the last arithmetic operation.
• **Zero:** Set when the result is 0.
• **Carry:** Set if an operation resulted in a carry (addition) into or borrow (subtraction) out of a high-order bit. Used for multiword arithmetic operations.
• **Equal:** Set if a logical compare result is equality.
• **Overflow:** Used to indicate arithmetic overflow.
• **Interrupt Enable/Disable:** Used to enable or disable interrupts.
• **Supervisor:** Indicates whether the processor 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.

- **Instruction Cycle**

Processor's instruction cycle (Figure 3.9).To recall, an instruction cycle includes the following stages:

• **Fetch:** Read the next instruction from memory into the processor.
• **Execute:** Interpret the op-code and perform the indicated operation.

• **Interrupt:** If interrupts are enabled and an interrupt has occurred, save the current process state and service the interrupt.

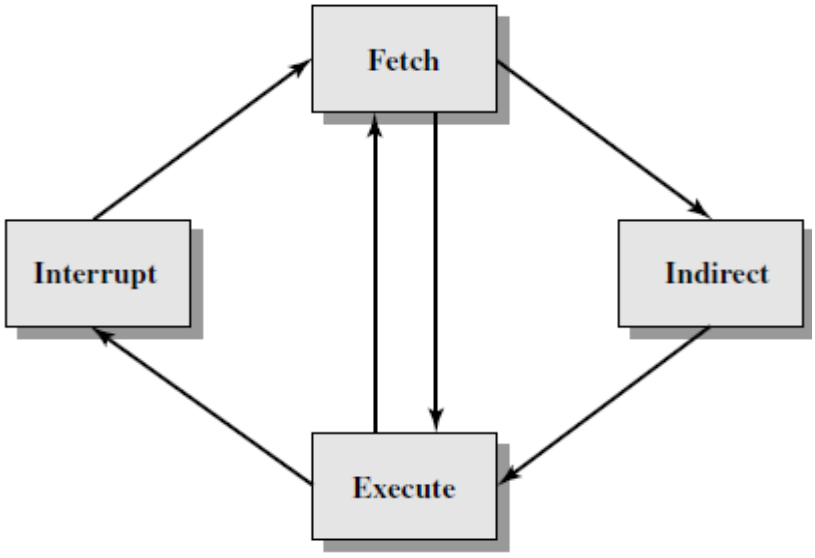
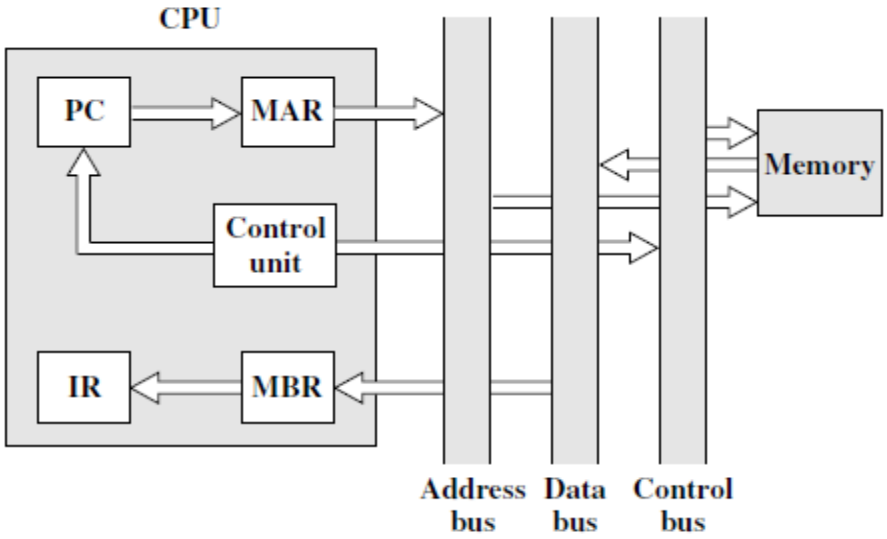


Figure 12.4 The Instruction Cycle

- **Fetch Cycle**

Let us assume that a processor that employs a memory address register (MAR), a memory buffer register (MBR), a program counter (PC), and an instruction register (IR).

During the fetch cycle, an instruction is read from memory. Figure 12.6 shows the flow of data during this cycle. The PC contains the address of the next instruction to be fetched. This address is moved to the MAR and placed on the address bus.



MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Figure 12.6 Data Flow, Fetch Cycle

The control unit requests a memory read, and the result is placed on the data bus and copied into the MBR and then moved to the IR. Meanwhile, the PC is incremented by 1, preparatory for the next fetch.

- **Indirect Cycle**

Once the fetch cycle is over, the control unit examines the contents of the IR to determine if it contains an operand specified using indirect addressing. If so, an indirect cycle is performed. As shown in Figure 12.7, this is a simple cycle.

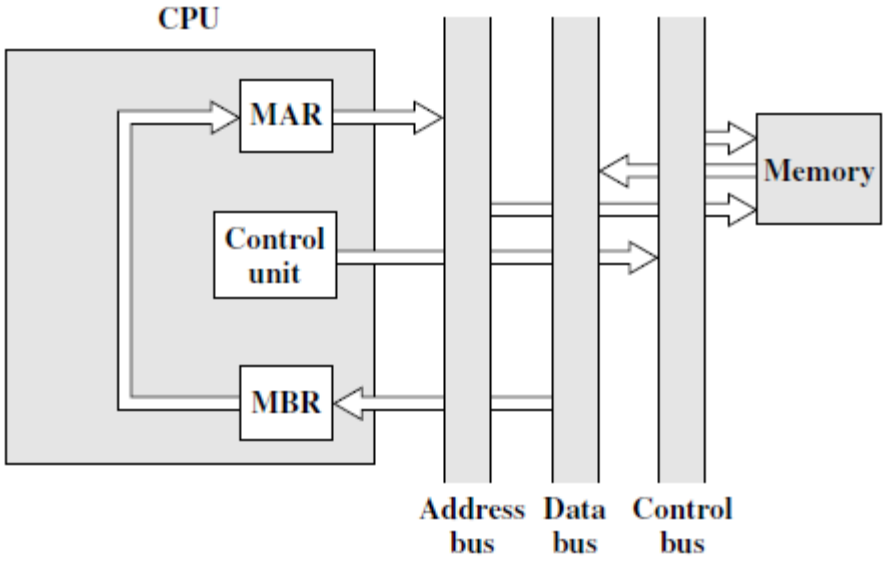


Figure 12.7 Data Flow, Indirect Cycle

The rightmost N bits of the MBR, which contain the address reference, are transferred to the MAR. Then the control unit requests a memory read, to get the desired address of the operand into the MBR. The fetch and indirect cycles are simple and predictable.

- **Interrupt Cycle**

The execute cycle takes many forms; the form depends on which of the various machine instructions is in the IR. This cycle may involve transferring data among registers, read or write from memory or I/O, and/or the invocation of the ALU.

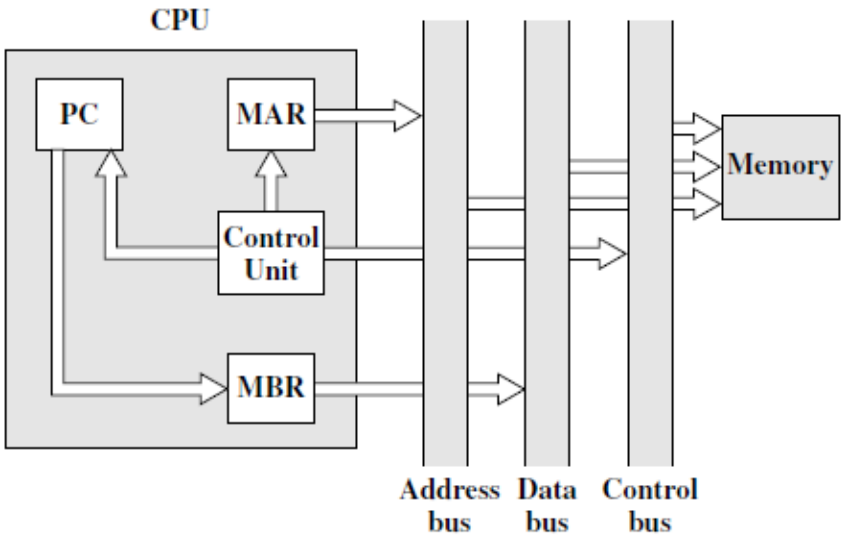


Figure 12.8 Data Flow, Interrupt Cycle

Like the fetch and indirect cycles, the interrupt cycle is simple and predictable (Figure 12.8). The current contents of the PC must be saved so that the processor can resume normal activity after the interrupt. Thus, the contents of the PC are transferred to the MBR to be written into memory. The special memory location reserved for this purpose is loaded into the MAR from the control unit.

- **Instruction pipelining**

Instruction pipelining is similar to the use of an assembly line in a manufacturing plant. An assembly line takes advantage of the fact that a product goes through various stages of production. By laying the production process out in an assembly line, products at various stages can be worked on simultaneously. This process is also referred to as pipelining; because, as in a pipeline, new inputs are accepted at one end before previously accepted inputs appear as outputs at the other end.

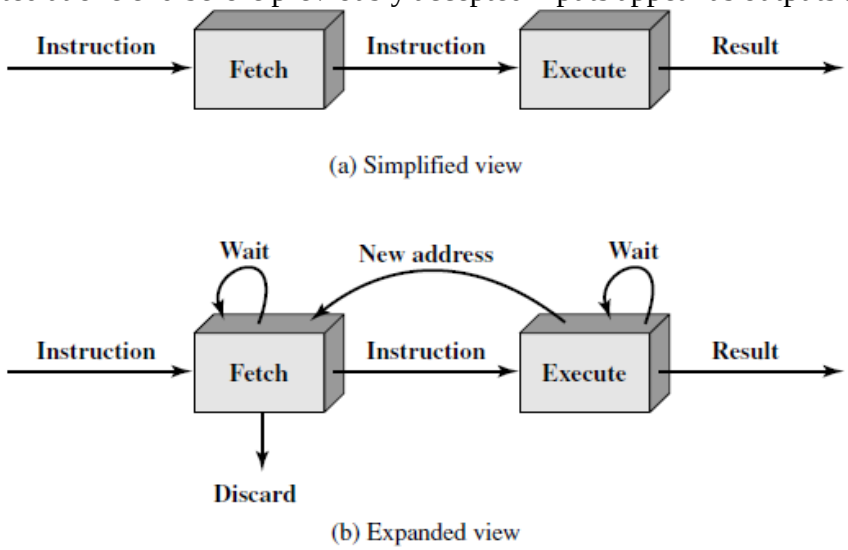


Figure 12.9 Two-Stage Instruction Pipeline

- **Dealing with Branches**

One of the major problems in designing an instruction pipeline is assuring a steady flow of instructions to the initial stages of the pipeline. The primary impediment, as we have seen, is the

conditional branch instruction. Until the instruction is actually executed, it is impossible to determine whether the branch will be taken or not.

A variety of approaches have been taken for dealing with conditional branches:
- Multiple streams
- Pre-fetch branch target
- Loop buffer
- Branch prediction
- Delayed branch

MULTIPLE STREAMS A simple pipeline suffers a penalty for a branch instruction because it must choose one of two instructions to fetch next and may make the wrong choice.
There are two problems with this approach:
- With multiple pipelines there are contention delays for access to the registers and to memory.
- Additional branch instructions may enter the pipeline (either stream) before the original branch decision is resolved. Each such instruction needs an additional stream.

PREFETCH BRANCH TARGET When a conditional branch is recognized, the target of the branch is pre-fetched, in addition to the instruction following the branch. This target is then saved until the branch instruction is executed. If the branch is taken, the target has already been pre-fetched. The IBM 360/91 uses this approach.

LOOP BUFFER A loop buffer is a small, very-high-speed memory maintained by the instruction fetch stage of the pipeline and containing the n most recently fetched instructions, in sequence. If a branch is to be taken, the hardware first checks whether the branch target is within the buffer. If so, the next instruction is fetched from the buffer.

BRANCH PREDICTION Various techniques can be used to predict whether a branch will be taken. Among the more common are the following:
- Predict never taken
- Predict always taken
- Predict by op-code
- Taken/not taken switch
- Branch history table

DELAYED BRANCH It is possible to improve pipeline performance by automatically rearranging instructions within a program, so that branch instructions occur later than actually desired.

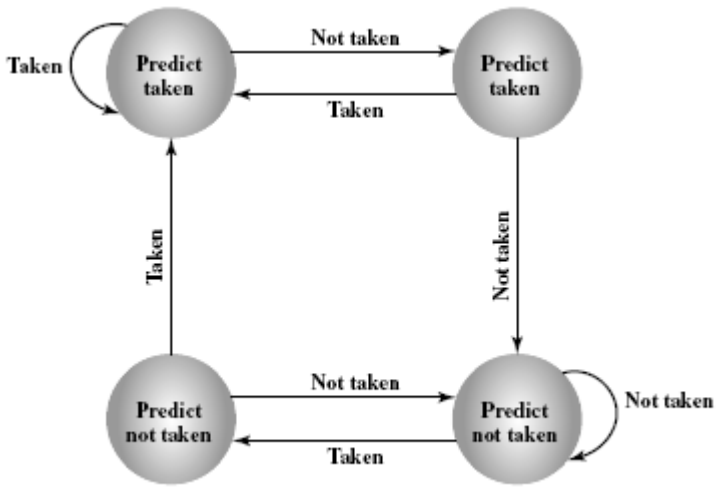


Figure 12.19 Branch Prediction State Diagram

The decision process can be represented more compactly by a finite-state machine, shown in Figure 12.19. The finite-state machine representation is commonly used in the literature.

The use of history bits, as just described, has one drawback: If the decision is made to take the branch, the target instruction cannot be fetched until the target address, which is an operand in the conditional branch instruction, is decoded. Greater efficiency could be achieved if the instruction fetch could be initiated as soon as the branch decision is made.