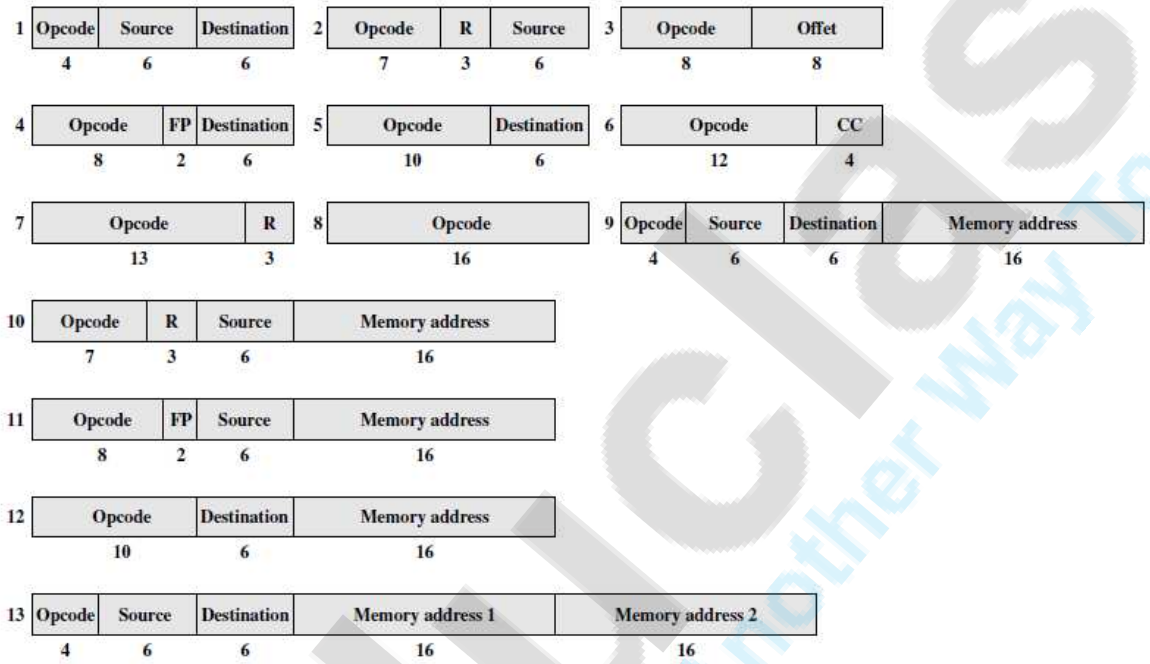
Unit 4:

Instruction Formats

An instruction format defines the layout of the bits of an instruction, in terms of its constituents parts. An instruction format must include an opcode and, implicitly or explicitly, zero or more operands. Each explicit operand is referenced using one of the addressing mode that is available for that machine. The format must, implicitly or explicitly, indicate the addressing mode of each operand. For most instruction sets, more than one instruction format is used.



Numbers below fields indicate bit length
Source and destination each contain a 3-bit addressing mode field and a 3-bit register number
FP indicates one of four floating-point registers
R indicates one of the general-purpose registers
CC is the condition code field

Figure 11.7 Instruction Formats for the PDP-11

Instruction Sets

An instruction is a command given to a computer to perform a specified operation on some given data. These instructions tell the Central Processing Unit or CPU what to do. In other words, an instruction guides the CPU to perform work accordingly. The most common fields found in the instructions are the operation code and the operands. Each field specifies different information for the computer. The two important fields of an instruction are as follows:

- Opcode
- Operand

Thus, **Instruction = Opcode + Operand**

Opcode (operation code) is an instruction field that specifies the particular operation to be performed by the instruction. Each operation has its unique opcode and may take several microoperations to accomplish. MOV, ADD and SUB are examples of Intel 8086 opcodes.

Operand fields specify where to get the source and destination operands for the operation specified by the opcode. The source/destination of operands can be the memory or one of the general-purpose registers. The complete set of opcodes for a particular microprocessor defines the instruction set for that processor.

Instruction sequencing is the method by which instructions are selected for execution, i.e., the manner in which control of the processor is transferred from one instruction to another.

The simplest method of controlling the sequence of instruction execution is to have each instruction explicitly specify the address of the next instruction to be run. However, explicit inclusion of instruction addresses in all the instructions is disadvantageous as the instruction length increases. This results in increased of cost of memory where the instructions are to be stored.

Instruction Execution

The sequence of operations performed by the CPU in processing an instruction is known as an instruction cycle. The time required to complete one instruction is called execution time.

To execute an instruction, the following three steps are required:

- Fetch step, during which a new instruction is read from the memory.
- Decode step, during which the instruction is decoded.
- Execute step, during which the operations specified by the instruction are executed.

The instruction fetch operation is initiated by loading the contents of the Program Counter (PC) into the Address Register (AR) and it sends a read request to the memory. The contents of the PC is the address of the instruction to be tun. The instruction rend from the memory is then placed in the Instruction Register (JR) and the content of the PC is incremented so that it contains the address of the next instruction in the program. After this, the instruction is decoded to determine the type of instruction that was just read. Finally, the instruction is executed to perform the operation specified by the instruction.

Let us consider an instruction, which adds the content of a memory location specified by register RO to the content of register R2 and the result is to be stored in R2. The execution of this instruction is performed in the following steps:

Step 1: Fetch and decode the instruction.

Step 2: Fetch the operand.

Step 3: Perform the operation (addition).

Step 4: Store the result in R2.

Addressing Modes - Examples with Assembly Language[8085/8086 CPU]

The address field or fields in a typical instruction format are relatively small. We would like to be able to reference a large range of locations in main memory or, for some systems, virtual memory. To achieve this objective, a variety of addressing techniques has been employed. They all involve some trade-off between address range and/or addressing flexibility, on the one hand, and the number of memory references in the instruction and/or the complexity of address calculation, on the other. The most common addressing techniques:

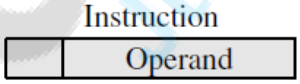
- Immediate
- Direct
- Indirect
- Register
- Register indirect
- Displacement
- Stack

Table 11.1 Basic Addressing Modes

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

Immediate Addressing mode

In this mode. the operand is indicated in the instruction itself, i.e., in the immediate addressing mode, the instruction has an operand field rather than an address field. The operand field contains the actual operand. This mode is useful for initializing registers to a constant value.



Following examples show the immediate addressing mode:

MVI 06 Move 06 to the accumulator.

ADD 05 Add 05 to the content of the AC.

Merits: The operand is available in the instruction as soon as the instruction fetch is over. Hence, the instruction executes quickly.

Demerits: The length of the operand field in instruction limits the value of the operand.

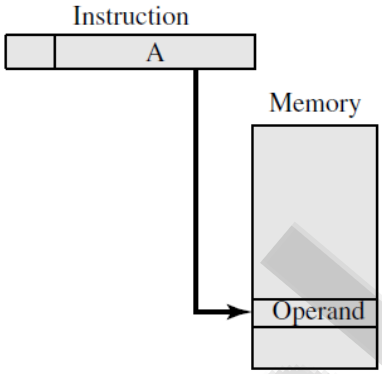
Direct Addressing Mode

The direct addressing mode is also known as absolute addressing mode. In this mode, the address of data, i.e. the operand is specified in the instruction itself. In other words, the operand resides in the memory and its address is given directly by the address field of the instruction.

Opcode	Memory Address = Operand

Following example shows the direct addressing mode.

LD ADR AC <— M[ADR]



(b) Direct

Merits: The instruction itself contains the operand address and hence, there is no need to find the effective address of the operand. Therefore, the instruction executes fast.

Demerits: The number of bits for specifying operand address is limited.

Indirect Addressing Mode

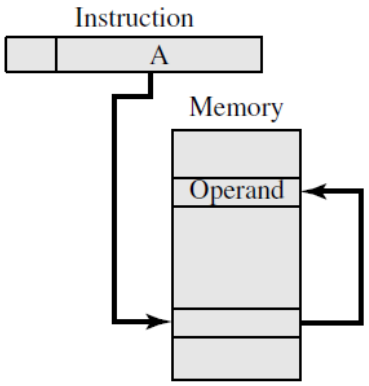
In this mode, the address field of the instruction gives the address where the operand is stored in the memory. Figure illustrates the indirect addressing mode, where the address field of the instruction refers to the address of a word in memory which in turn contains the full length address of the operand.

Opcode	Memory Address = Operand Address

Following example shows the indirect addressing mode:

LD @ ADR

AC <— M[M[ADR])



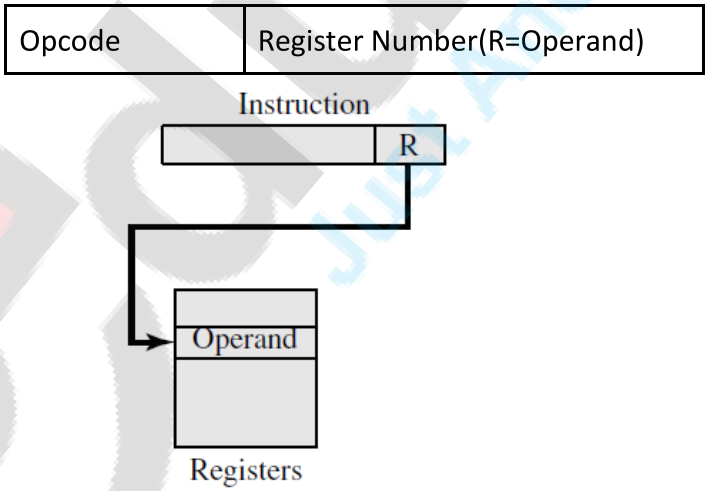
(c) Indirect

Merits: The merit of this mode is that for the word length of N, an address space of 2N can be addressed.

Demerits: The demerit is that instruction execution requires two memory reference to fetch the operand multilevel or cascaded indirect addressing can also be used.

Register Addressing mode

In the register-addressing method, the operands are in registers that exist within the CPU. i.e.. the contents of the register are the operand itself. The instruction contains the register number that has the operand. This addressing mode is very useful in a long program for storing the intermediate results in registers rather than in memory. Figure illustrates the register addressing mode.



(d) Register

Following examples show the register addressing mode:

MOV RI, R2 RI ← R2 Transfer the contents of register R2 to that of register RI

ADD RI AC ←AC + R1 Add the contents of register RI to the accumulator

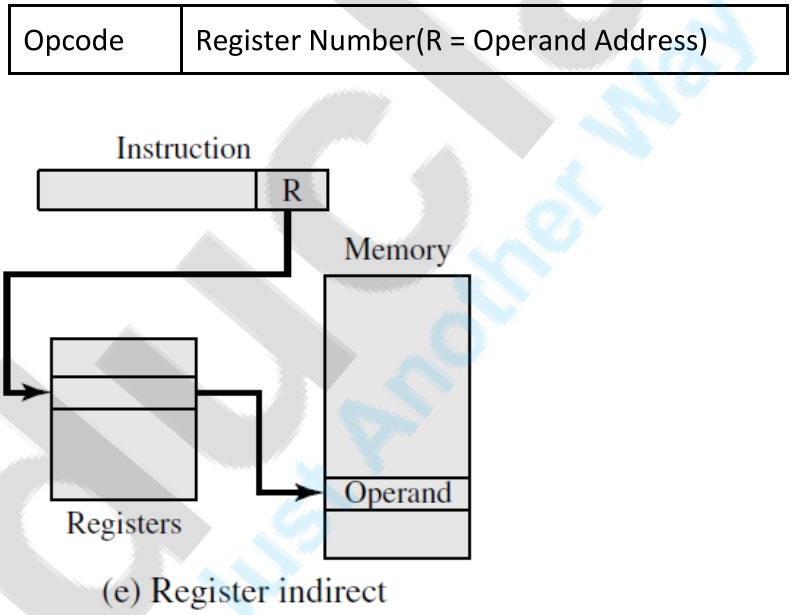
LD RI AC ← R1 Load the content of register to the accumulator

Merits: Operands are fetched very fast without using memory.

Demerits: Since the number of registers is limited, the utilization of these registers must be done very carefully.

Register Indirect Addressing Mode

The command identifies a register in the CPU whose contents offer the address of the memory location where the operand is stored, i.e., the selected register comprises the address of the operand rather than the operand itself. In this mode, the register acts as the memory address register. Figure illustrates register indirect addressing mode.



(e) Register indirect

Merits: Since the register number is specified with bits, the length of the instruction must be decided very carefully.

Displacement Addressing Mode

These addressing modes require that the address field of the instruction be added to the content of a specific register in the CPU to get the effective address.

Thus,

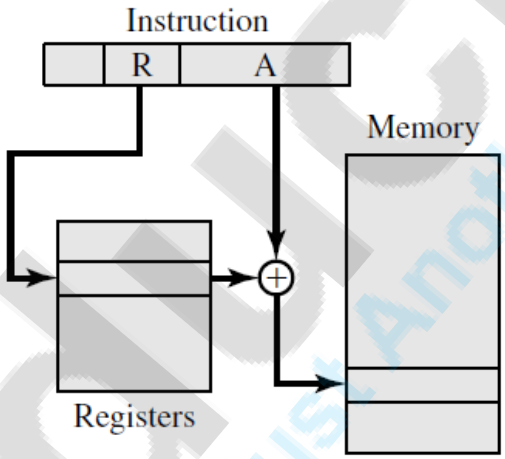
Effective address = Address part of instruction + Content of the CPU register

The CPU register used in computing the effective address may be a program counter, an index register or a base register. Displacement addressing modes are of the following three types:

- Relative Addressing Mode: In the relative addressing mode for calculating the effective address, the contents of the program counter and the address pan of the instruction are added.

For example, let the program counter contains 750 and the address part of the instruction contains the number 35. The instruction at the memory location 750 is read from memory during the fetch phase and the program counter is incremented by one to 751. The effective address computation for the relative address mode is 751 + 35 = 786.

- Index Addressing Mode: In the index addressing mode to calculate the effective address, the contents of the index register and the address part of the instruction are added.
- Base Register Addressing Mode: In the base register addressing mode for calculating the effective address, the content of the base register and the address part of the instruction are added.

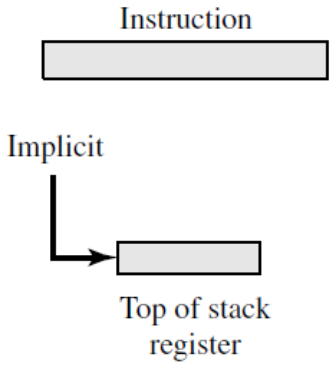


(f) Displacement

Stack Addressing Mode

A stack is a linear array of locations. It is sometimes referred to as a pushdown list or last-in-first-out queue. The stack is a reserved block of locations. Items are appended to the top of the stack so that, at any given time, the block is partially filled. Associated with the stack is a pointer whose value is the address of the top of the stack. Alternatively, the top two elements of the stack may be in processor registers, in which case the stack pointer references the third element of the stack. The stack pointer is maintained in a register. Thus, references to stack locations in memory are in fact register indirect addresses. The stack mode of addressing is a form of implied

addressing. The machine instructions need not include a memory reference but implicitly operate on the top of the stack.



(g) Stack

Processor Organization

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. It needs to store instructions and data temporarily while an instruction is being executed. In other words, the processor needs a small internal memory.

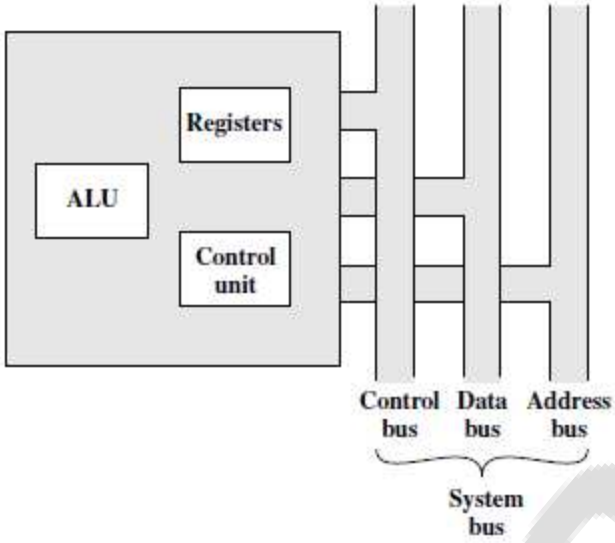


Figure 12.1 The CPU with the System Bus

Figure 12.1 is a simplified view of a processor, indicating its connection to the rest of the system via the system bus. A similar interface would be needed for any of the interconnection structures described in Chapter 3.The reader will recall that the major components of the processor are an arithmetic and logic unit (ALU) and a control unit (CU).The ALU does the actual computation or processing of data.The control unit controls the movement of data and instructions into and out of the processor and controls the operation of the ALU. In addition, the figure shows a minimal internal memory, consisting of a set of storage locations, called registers.

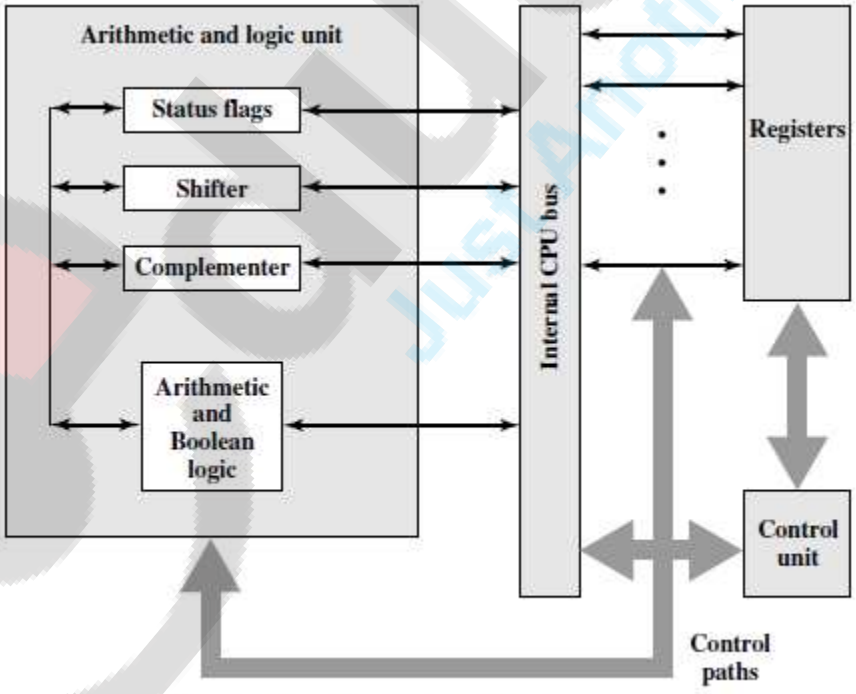


Figure 12.2 Internal Structure of the CPU

Figure 12.2 is a slightly more detailed view of the processor.The data transfer and logic control paths are indicated, including an element labeled internal processor bus. This element is needed to transfer data between the various registers and the ALU because the ALU in fact operates only on data in the internal processor memory.The figure also shows typical basic elements of the ALU. Note the similarity between the internal structure of the computer as a whole and the internal structure of the processor. In both cases, there is a small collection of major elements (computer: processor, I/O, memory; processor: control unit,ALU, registers) connected by data paths.

Structure and Function

Register Organization

A computer system employs a memory hierarchy. At higher levels of the hierarchy, memory is faster, smaller, and more expensive (per bit). Within the processor, there is a set of registers that function as a level of memory above main memory and cache in the hierarchy. 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 Code

General-purpose registers can be assigned to a variety of functions by the programmer. Sometimes their use within the instruction set is orthogonal to the operation. That is, any general-purpose register can contain the operand for any opcode. This provides true general-purpose register use. Often, however, there are restrictions. For example, there may be dedicated registers for floating-point and stack operations.

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 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. Examples include the following:

- **Segment pointers:** In a machine with segmented addressing (see Section 8.3), a segment register holds the address of the base of the segment. There may be multiple registers: for example, one for the operating system and one for the current process.
- **Index registers:** These are used for indexed addressing and may be auto indexed.
- **Stack pointer:** If there is user-visible stack addressing, then typically there is a dedicated register that points to the top of the stack. This allows implicit addressing; that is, push, pop, and other stack instructions need not contain an explicit stack operand.

Condition codes are bits set by the processor hardware as the result of operations. For example, an arithmetic operation may produce a positive, negative, zero, or overflow result. In addition to the result itself being stored in a register or memory, a condition code is also set.The code may subsequently be tested as part of a conditional branch operation. Condition code bits are collected into one or more registers. Usually, they form part of a control register. Generally, machine instructions allow these bits to be read by implicit reference, but the programmer cannot alter them.

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. Of course, different machines will have different register organizations and use different terminology.We list here a reasonably complete list of register types. 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 register (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

The processor updates the PC after each instruction fetch so that the PC always points to the next instruction to be executed. A branch or skip instruction will also modify the contents of the PC.The fetched instruction is loaded into an IR, where the opcode and operand specifiers are analyzed. Data are exchanged with memory using the MAR and MBR. In a bus-organized system,

the MAR connects directly to the address bus, and the MBR connects directly to the data bus. User visible registers, in turn, exchange data with the MBR.

Many processor designs include a register or set of registers, often known as the program status word (PSW), that contain 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 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

An instruction cycle includes the following stages:

Fetch: Read the next instruction from memory into the processor.

Execute: Interpret the opcode and perform the indicated operation.

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

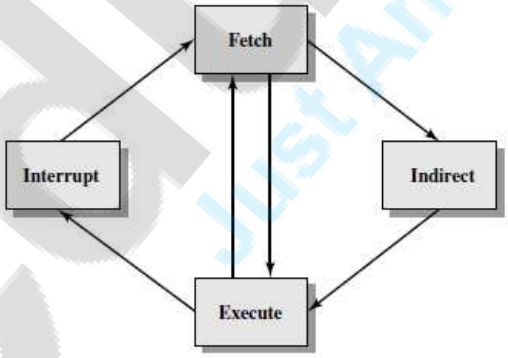


Figure 12.4 The Instruction Cycle

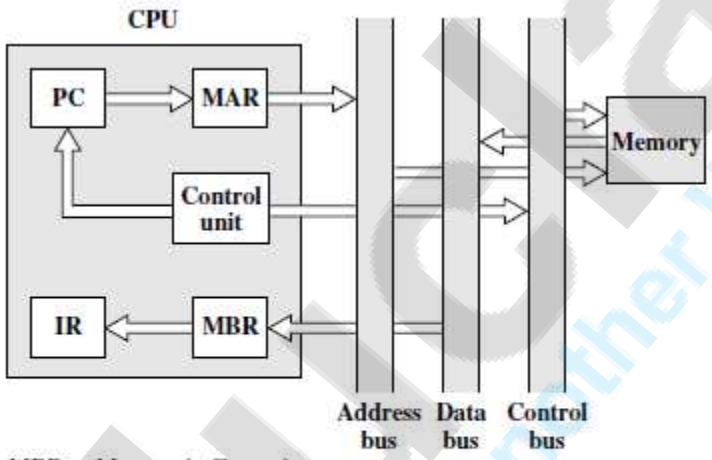
The Indirect Cycle

The execution of an 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 accesses are required. We can think of the fetching of indirect addresses as one more instruction stages.The result is shown in Figure 12.4.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. Following execution, an interrupt may be processed before the next instruction fetch.

Data Flow

The exact sequence of events during an instruction cycle depends on the design of the processor.We can, however, indicate in general terms what must happen. 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

Instruction Pipelining

Introduction to RISC and CISC Architecture

CISC	RISC
Emphasis on hardware	Emphasis on software
Includes multi-clock	Single-clock
complex instructions	reduced instruction only
Memory-to-memory: "LOAD" and "STORE" incorporated in instructions	Register to register: "LOAD" and "STORE" are independent instructions
high cycles per second, Small code sizes	Low cycles per second, large code sizes
Transistors used for storing complex instructions	Spends more transistors on memory registers

Instruction Level Parallelism and Superscalar Processors

Design Issues