Module No-1:

Basics of Computer Organization:

Basic organization of the stored program computer and operation sequence for execution of a program, Von Neumann & Harvard Architecture. RISC vs. CISC based architecture.

Fetch, decode and execute cycle, Concept of registers and storage, Instruction format, Instruction sets and addressing modes.

NOTES

Computer Organization and Architecture

Computer technology has made incredible improvement in the past half century. In the early part of computer evolution, there were no stored-program computer, the computational power was less and on the top of it the size of the computer was a very huge one.

Today, a personal computer has more computational power, more main memory, more disk storage, smaller in size and it is available in affordable cost.

This rapid rate of improvement has come both from advances in the technology used to build computers and from innovation in computer design. In this course we will mainly deal with the innovation in computer design.

The task that the computer designer handles is a complex one: Determine what attributes are important for a new machine, then design a machine to maximize performance while staying within cost constraints.

This task has many aspects, including instruction set design, functional organization, logic design, and implementation.

While looking for the task for computer design, both the terms computer organization and computer architecture come into picture.

It is difficult to give precise definition for the terms Computer Organization and Computer Architecture. But while describing computer system, we come across these terms, and in literature, computer scientists try to make a distinction between these two terms.

Computer architecture refers to those parameters of a computer system that are visible to a programmer or those parameters that have a direct impact on the logical execution of a program. Examples of architectural attributes include the instruction set, the number of bits used to represent different data types, I/O mechanisms, and techniques for addressing memory.

Computer organization refers to the operational units and their interconnections that realize the architectural specifications. Examples of organizational attributes include those hardware details transparent to the programmer, such as control signals, interfaces between the computer and peripherals, and the memory technology used.

In this course we will touch upon all those factors and finally come up with the concept how these attributes contribute to build a complete computer system.

The model of a computer can be described by four basic units in high level abstraction which is shown in figure 1.1. These basic units are:

- **Central Processor Unit**
- **Input Unit**
- **Output Unit**
- **Memory Unit**

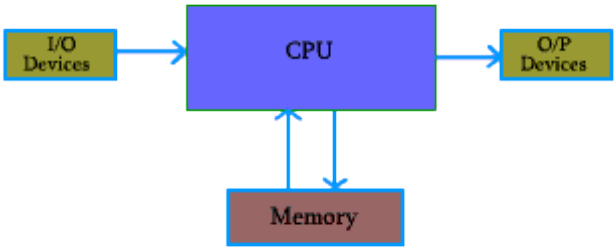


Figure 1.1: Basic Unit of a Computer

Basic Computer Model and different units of Computer

A. Central Processor Unit (CPU) :

Central processor unit consists of two basic blocks :

- The program control unit has a set of registers and control circuit to generate control signals.
- The execution unit or data processing unit contains a set of registers for storing data and an Arithmatic and Logic Unit (ALU) for execution of arithmatic and logical operations.

In addition, CPU may have some additional registers for temporary storage of data.

B. Input Unit :

With the help of input unit data from outside can be supplied to the computer. Program or data is read into main storage from input device or secondary storage under the control of CPU input instruction.

Example of input devices: Keyboard, Mouse, Hard disk, Floppy disk, CD-ROM drive etc.

C. Output Unit :

With the help of output unit computer results can be provided to the user or it can be stored in stograge device permanently for future use. Output data from main storage go to output device under the control of CPU output instructions.

Example of output devices: Printer, Monitor, Plotter, Hard Disk, Floppy Disk etc.

D. Memory Unit : Memory unit is used to store the data and program. CPU can work with the information stored in memory unit. This memory unit is termed as primary memory or main memory module. These are basically semi conductor memories.

There ate two types of semiconductor memories -

- Volatile Memory : RAM (Random Access Memory).
- Non-Volatile Memory : ROM (Read only Memory), PROM (Programmable ROM) EPROM (Erasable PROM), EEPROM (Electrically Erasable PROM).

Secondary Memory :

There is another kind of storage device, apart from primary or main memory, which is known as secondary memory. Secondary memories are non volatile memory and it is used for permanent storage of data and program.

Example of secondary memories:

Hard Disk, Floppy Disk, Magenetic Tape	------	These are magnetic devices,
CD-ROM	------	is optical device
Thumb drive (or pen drive)	------	is semiconductor memory.

Basic Working Principle of a Computer

Before going into the details of working principle of a computer, we will analyse how computers work with the help of a small hypothetical computer.

In this small computer, we do not consider about Input and Output unit. We will consider only CPU and memory module. Assume that somehow we have stored the program and data into main memory. We will see how CPU can perform the job depending on the program stored in main memory.

P.S. - Our assumption is that students understand common terms like program, CPU, memory etc. without knowing the exact details.

Consider the Arithmetic and Logic Unit (ALU) of Central Processing Unit :

Consider an ALU which can perform four arithmetic operations and four logical operations
To distingish between arithmetic and logical operation, we may use a signal line,

0 - in that signal, represents an arithmetic operation and

1 - in that signal, represents a logical operation.

In the similar manner, we need another two signal lines to distinguish between four arithmatic operations.

The different operations and their binary code is as follows:

Arithmatic		**Logical**	
000	ADD	100	OR
001	SUB	101	AND
010	MULT	110	NAND
011	DIV	111	NOR

Consider the part of control unit, its task is to generate the appropriate signal at right moment.

There is an instruction decoder in CPU which decodes this information in such a way that computer can perform the desired task

The simple model for the decoder may be considered that there is three input lines to the decoder and correspondingly it generates eight output lines. Depending on input combination only one of the output signals will be generated and it is used to indicate the corresponding operation of ALU.

In our simple model, we use three storage units in CPU,

Two -- for storing the operand and
one -- for storing the results.

These storage units are known as register.

But in computer, we need more storage space for proper functioning of the Computer.

Some of them are inside CPU, which are known as register. Other bigger chunk of storage space is known as primary memory or main memory. The CPU can work with the information available in main memory only.

To access the data from memory, we need two special registers one is known as Memory Data Register (MDR) and the second one is Memory Address Register (MAR).

Data and program is stored in main memory. While executing a program, CPU brings instruction and data from main memory, performs the tasks as per the instruction fetch from the memory. After completion of operation, CPU stores the result back into the memory.

Main Memory Organization

Main memory unit is the storage unit, There are several location for storing information in the main memory module.

The capacity of a memory module is specified by the number of memory location and the information stored in each location.

A memory module of capacity 16 X 4 indicates that, there are 16 location in the memory module and in each location, we can store 4 bit of information.

We have to know how to indicate or point to a specific memory location. This is done by address of the memory location.

We need two operation to work with memory.

READ Operation:	This operation is to retrive the data from memory and bring it to CPU register
WRITE Operation:	This operation is to store the data to a memory location from CPU register

We need some mechanism to distinguish these two operations READ and WRITE.

With the help of one signal line, we can differentiate these two operations. If the content of this signal line is

0, we say that we will do a READ operation; and if it is
1, then it is a WRITE operation.

To transfer the data from CPU to memory module and vice-versa, we need some connection. This is termed as DATA BUS.

The size of the data bus indicate how many bit we can transfer at a time. Size of data bus is mainly specified by the data storage capacity of each location of memory module.

We have to resolve the issues how to specify a particular memory location where we want to store our data or from where we want to retrieve the data.

This can be done by the memory address. Each location can be specified with the help of a binary address.

If we use 4 signal lines, we have 16 different combinations in these four lines, provided we use two signal values only (say 0 and 1).

To distinguish 16 location, we need four signal lines. These signal lines use to identify a memory location is termed as ADDRESS BUS. Size of address bus depends on the memory size. For a memory module of capacity of 2^n location, we need *n* address lines, that is, an address bus of size *n*.

We use a address decoder to decode the address that are present in address bus

As for example, consider a memory module of 16 location and each location can store 4 bit of information
The size of address bus is 4 bit and the size of the data bus is 4 bit
The size of address decoder is 4 X 16.

There is a control signal named R/W.
If R/W = 0, we perform a READ operation and
if R/W = 1, we perform a WRITE operation

If the contents of address bus is 0101 and contents of data bus is 1100 and R/W = 1, then 1100 will be written in location 5.

If the contents of address bus is 1011 and R/W=0, then the contents of location 1011 will be placed in data bus.

In next section, we will explain how to perform memory access operation in our small hypothetical computer.

Memory Instruction

We need some more instruction to work with the computer. Apart from the instruction needed to perform task inside CPU, we need some more instructions for data transfer from main memory to CPU and vice versa.
In our hypothetical machine, we use three signal lines to identify a particular instruction. If we want to include more instruction, we need additional signal lines.

Instruction	Code	Meaning
1000	LDAI imm	Load register A with data that is given in the program
1001	LDAA addr	Load register A with data from memory location addr
1010	LDBI imm	Load register B with data
1011	LDBA addr	Load register B with data from memory location addr
1100	STC addr	Store the value of register C in memory location addr
1101	HALT	Stop the execution
1110	NOP	No operation
1111	NOP	No operation

With this additional signal line, we can go upto 16 instructions. When the signal of this new line is 0, it will indicate the ALU operation. For signal value equal to 1, it will indicate 8 new instructions. So, we can design 8 new memory access instructions.

We have added 6 new instructions. Still two codes are unused, which can be used for other purposes. We show it as NOP means No Operation.

We have seen that for ALU operation, instruction decoder generated the signal for appropriate ALU operation.

Apart from that we need many more signals for proper functioning of the computer. Therefore, we need a module, which is known as control unit, and it is a part of CPU. The control unit is responsible to generate the appropriate signal.

As for example, for LDAI instruction, control unit must generate a signal which enables the register A to store in data into register A.

One major task is to design the control unit to generate the appropriate signal at appropriate time for the proper functioning of the computer.

Consider a simple problem to add two numbers and store the result in memory, say we want to add 7 to 5.

To solve this problem in computer, we have to write a computer program. The program is machine specific, and it is related to the instruction set of the machine.

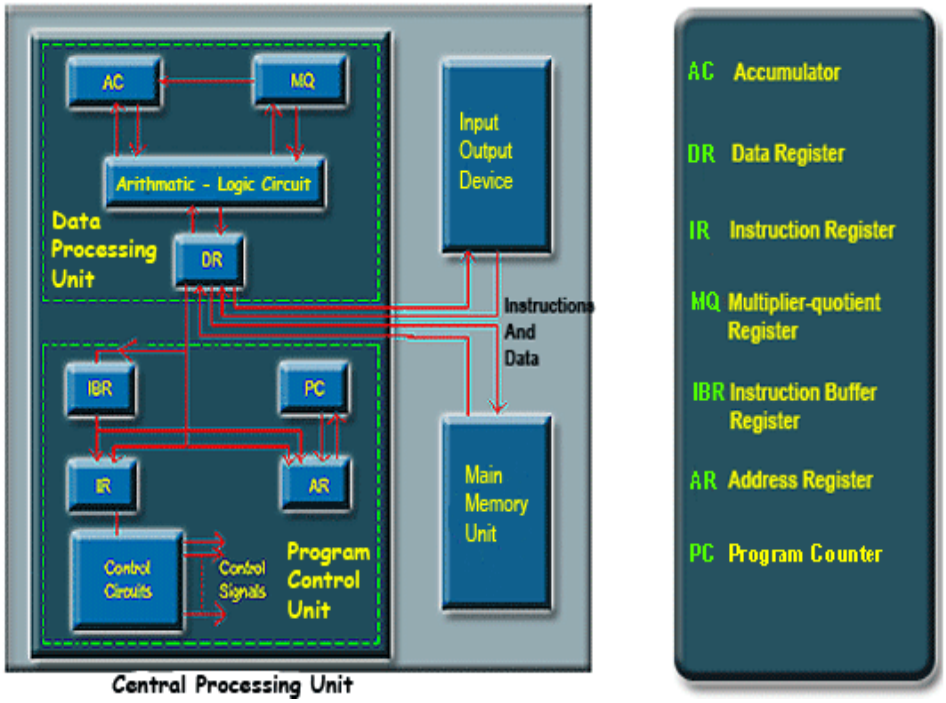
Main Memory Organization: Stored Program

The present day digital computers are based on stored-program concept introduced by Von Neumann. In this stored-program concept, programs and data are stored in separate storage unit called memories.

Central Processing Unit, the main component of computer can work with the information stored in storage unit only.

In 1946, Von Neumann and his colleagues began the design of a stored-program computer at the Institute for Advanced Studies in Princeton. This computer is referred as the IAS computer.

The structure of IAS computer is shown in Figure 1.2.



The IAS computer is having three basic units:

- The Central Processing Unit (CPU).
- The Main Memory Unit.
- The Input/Output Device.

Central Processing Unit:

This is the main unit of computer, which is responsible to perform all the operations. The CPU of the IAS computer consists of a data processing unit and a program control unit.

The data processing unit contains a high speed registers intended for temporary storage of instructions, memory addresses and data. The main action specified by instructions are performed by the arithmetic-logic circuits of the data processing unit.

The control circuits in the program control unit are responsible for fetching instructions, decoding opcodes, controlling the information movements correctly through the system, and providing proper control signals for all CPU actions.

The Main Memory Unit:

It is used for storing programs and data. The memory locations of memory unit is uniquely specified by the memory address of the location. M(X) is used to indicate the location of the memory unit M with address X.

The data transfer between memory unit and CPU takes place with the help of data register DR. When CPU wants to read some information from memory unit, the information first brings to DR, and after that it goes to appropriate position. Similarly, data to be stored to memory must put into DR first, and then it is stored to appropriate location in the memory unit.

The address of the memory location that is used during memory read and memory write operations are stored in the memory register AR.

The information fetched from the memory is a operand of an instruction, then it is moved from DR to data processing unit (either to AC or MQ). If it is an operand, then it is moved to program control unit (either to IR or IBR).

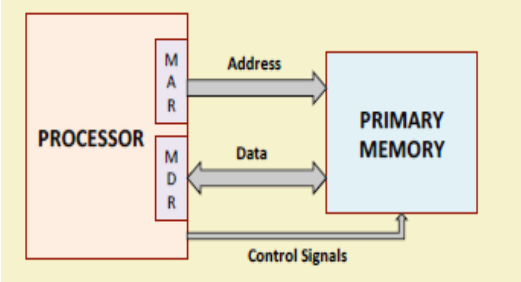
Two additional registers for the temporary storage of operands and results are included in data processing units: the accumulator AC and the multiplier-quotient register MQ.

Two instructions are fetch simultaneously from M and transferred to the program control unit. The instruction that is not to be executed immediately is placed in the instruction buffer register IBR. The opcode of the other instruction is placed in the instruction register IR where it is decoded.

In the decoding phase, the control circuits generate the required control signals to perform the specified operation in the instruction.

The program counter(PC) is used to store the address of the next instruction to be fetched from memory.

Interfacing with the Primary Memory



- Two special-purpose registers are used:
- – *Memory Address Register (MAR): Holds the* address of the memory location to be accessed.
- – *Memory Data Register (MDR): Holds the data* that is being written into memory, or will receive the data being read out from memory.
- Memory considered as a linear array of storage locations (bytes or words) each

with unique address.

The memory has 4096 words in it

4096 = 2^{12}, so it takes 12 bits to select a word in memory

Each word is 16 bits long

Size of MAR=12 bits

Size of MDR=16bits

- To read data from memory
  - a) Load the memory address into MAR.
  - b) Issue the control signal *READ.*
  - c) The data read from the memory is stored into MDR.

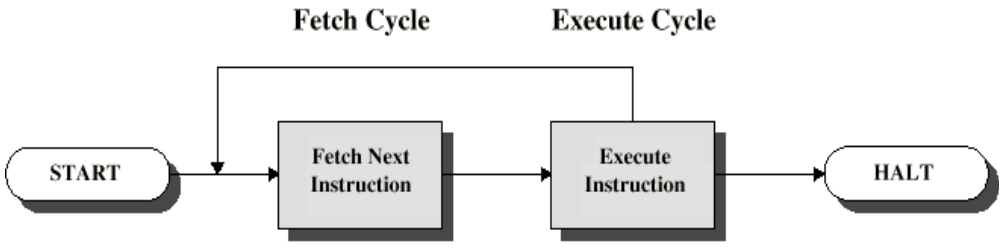
- To write data into memory
    - a) Load the memory address into MAR.
    - b) Load the data to be written into MDR.
    - c) Issue the control signal *WRITE.*

For Keeping Track of Program / Instructions

- Two special-purpose registers are used:
- *Program Counter (PC): Holds the memory address of the next* instruction to be executed.
- Automatically incremented to point to the next instruction when an instruction is being executed.
- *Instruction Register (IR): Temporarily holds an instruction that has* been fetched from memory.
- Need to be decoded to find out the instruction type.
- Also contains information about the location of the data.

Instruction Cycle

- Two steps:
    - Fetch
    - Execute



Fetch Cycle

- Program Counter (PC) holds address of next instruction to fetch
- Processor fetches instruction from memory location pointed to by PC
- Increment PC
    - Unless told otherwise

- Instruction loaded into Instruction Register (IR)
- Processor interprets instruction and performs required actions

Execute Cycle

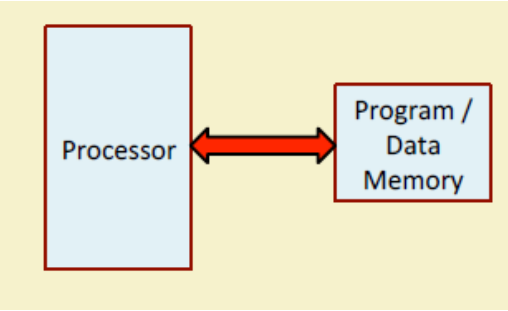
- Processor-memory
  - data transfer between CPU and main memory
- Processor I/O
  - Data transfer between CPU and I/O module
- Data processing
  - Some arithmetic or logical operation on data
- Control
  - Alteration of sequence of operations
  - e.g. jump
  - Combination of above

Input Output Device :

Input devices are used to put the information into computer. With the help of input devices we can store information in memory so that CPU can use it. Program or data is read into main memory from input device or secondary storage under the control of CPU input instruction.

Output devices are used to output the information from computer. If some results are evaluated by computer and it is stored in computer, then with the help of output devices, we can present it to the user. Output data from the main memory go to output device under the control of CPU output instruction.

von-Neumann Architecture



- Stored Program concept
- Main memory storing programs and data
- ALU operating on binary data
- Control unit interpreting instructions from memory and executing
- Input and output equipment operated by control unit
- Princeton Institute for Advanced Studies
  - IAS Completed 1952

• Instructions and data are stored in the same memory module.
• More flexible and easier to implement.
• Suitable for most of the general purpose processors.
• General Disadvantage:
– The processor-memory bus acts as the bottleneck.

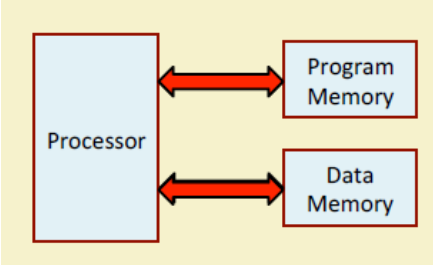
– All instructions and data are moved back and forth through the pipe

How can this be reduced?

This performance problem is reduced by using cache memory.(details in memory part)
Using RISC architecture as it uses less number of memory reference instruction and uses large number of registers.

Harvard Architecture

• Separate memory for program and data.
– Instructions are stored in program memory and data are stored in data memory.
• Instruction and data accesses can be done in parallel.
• Some microcontrollers and pipelines with separate instruction and data caches follow this concept.
• The processor-memory bottleneck remains.



CISC AND RISC ARCHITECTURE

• Computer architectures have evolved over the years.
– Features that were developed for mainframes and supercomputers in the 1960s and 1970s have started to appear on a regular basis on later generation microprocessors.
• Two broad classifications of ISA:
a) Complex Instruction Set Computer (CISC)
b) Reduced Instruction Set Computer (RISC)

Complex Instruction Set Computer (CISC)

– More traditional approach.
– Main features:
• Complex instruction set
• Large number of addressing modes (R-R, R-M, M-M, indexed, indirect, etc.)
• Special-purpose registers and Flags (sign, zero, carry, overflow, etc.)
• Variable-length instructions / Complex instruction encoding
• Ease of mapping high-level language statements to machine instructions
• Instruction decoding / control unit design more complex
• Pipeline implementation quite complex

– CISC Examples:
• IBM 360/370 (1960-70)
• VAX-11/780 (1970-80)
• Intel x86 / Pentium (1985-present)

Only CISC instruction set that survived over generations.
• Desktop PC's / Laptops use these.
• The volume of chips manufactured is so high that there is enough motivation to pay the extra design cost.
• Sufficient hardware resources available today to translate from CISC to RISC internally.

Reduced Instruction Set Computer (RISC)

– Very widely used among many manufacturers today.
– Also referred to as *Load-Store Architecture*.
• Only LOAD and STORE instructions access memory.

• All other instructions operate on processor registers.
– Main features:
• Simple architecture for the sake of efficient pipelining.
• Simple instruction set with very few addressing modes.
• Large number of general-purpose registers; very few special-purpose.
• Instruction length and encoding uniform for easy instruction decoding.
• Compiler assisted scheduling of pipeline for improved performance.

– RISC Examples:
• CDC 6600 (1964)
• MIPS family (1980-90)
• SPARC
• ARM microcontroller family

• Almost all the computers today use a RISC based pipeline for efficient implementation.
• RISC based computers use compilers to translate into RISC instructions.
• CISC based computers (e.g. x86) use hardware to translate into RISC instructions.

Instruction Set & Addressing

1. **Various addressing modes**
2. **Machine Instruction**
3. **Instruction Format**

- **Program**
  - **A sequence of (machine) instructions**
- **(Machine) Instruction**
  - **A group of bits that tell the computer to *perform a specific operation* (a sequence of micro-operation)**
- **The instructions of a program, along with any needed data are stored in memory**
- **The CPU reads the next instruction from memory**
- **It is placed in an *Instruction Register* (IR)**
- **Control circuitry in control unit then translates the instruction into the sequence of micro operations necessary to implement it**

Instruction Set Architecture (ISA)

- **Serves as an interface between software and hardware.**

- **Typically consists of information regarding the programmer's view of the architecture (i.e. the registers, address and data buses, etc.).**
- **Also consists of the instruction set.**
- **Many ISA's are not specific to a particular computer architecture.**
- **They survive across generations.**
- **Classic examples: IBM 360 series, Intel x86 series, etc.**

Instruction Formats

- **Layout of bits in an instruction**
- **Includes opcode**
- **Includes (implicit or explicit) operand(s)**
- **Usually more than one instruction format in an instruction set**

Instruction Length

- **Affected by and affects:**
  - **Memory size**
  - **Memory organization**
  - **Bus structure**
  - **CPU complexity**
  - **CPU speed**

Instruction Addressing

We have examined the types of *operands* and *operations* that may be specified by *machine instructions*. Now we have to see how is the address of an operand specified, and how are the *bits* of an instruction organized to define the *operand addresses* and operation of that instruction.

Addressing Modes:

The most common addressing techniques are:

- **Immediate**
- **Direct**
- **Indirect**
- **Register**
- **Register Indirect**
- **Displacement**
- **Stack**

All computer architectures provide more than one of these *addressing modes*. The question arises as to how the *control unit* can determine which addressing mode is being used in a particular instruction. Several approaches are used. Often, different *opcodes* will use different addressing modes. Also, one or more bits in the *instruction format* can be used as a *mode field*. The value of the mode field determines which addressing mode is to be used.

What is the interpretation of *effective address*. In a system without virtual memory, the effective address will be either a main memory address or a register. In a virtual memory system, the effective address is a virtual address or a register. The actual mapping to a physical address is a function of the paging mechanism and is invisible to the programmer.

To explain the addressing modes, we use the following notation:

A	=	contents of an address field in the instruction that refers to a memory
R	=	contents of an address field in the instruction that refers to a register
EA	=	actual (effective) address of the location containing the referenced operand
(X)	=	contents of location X

Immediate Addressing:

The simplest form of addressing is immediate addressing, in which the operand is actually present in the instruction:

OPERAND = A

This mode can be used to define and use constants or set initial values of variables. The advantage of immediate addressing is that no memory reference other than the instruction fetch is required to obtain the operand. The disadvantage is that the size of the number is restricted to the size of the address field, which, in most instruction sets, is small compared with the world length.

Figure 4.1: Immediate Addressing Mode

The instruction format for Immediate Addressing Mode is shown in the Figure 4.1.

Direct Addressing:

A very simple form of addressing is direct addressing, in which the address field contains the effective address of the operand:

EA = A

It requires only one memory reference and no special calculation.

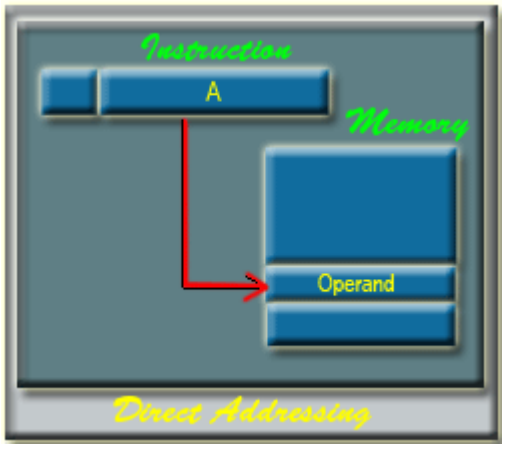


Figure 4.2: Direct Addressing Mode

The fetching of data from the memory location in case of direct addressing mode is shown in the Figure 4.2. Here, 'A' indicates the memory address field for the operand.

Indirect Addressing:

With direct addressing, the length of the address field is usually less than the word length, thus limiting the address range. One solution is to have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand. This is know as indirect addressing:

EA = (A)

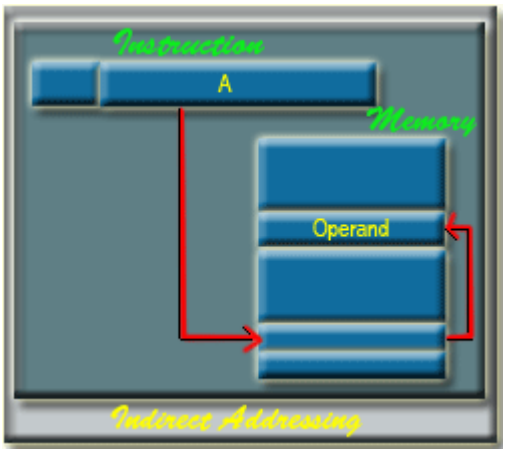


Figure 4.3: Indirect Addressing Mode

The exact memory location of the operand in case of indirect addressing mode is shown in the Figure 4.2. Here 'A' indicates the memory address field of the required operands.

Register Addressing:

Register addressing is similar to direct addressing. The only difference is that the address field referes to a register rather than a main memory address:

EA = R

The advantages of register addressing are that only a small address field is needed in the instruction and no memory reference is required. The disadvantage of register addressing is that the address space is very limited.

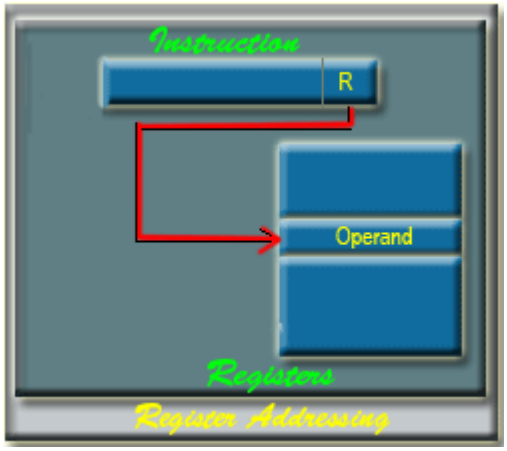


Figure 4.4: Register Addressing Mode.

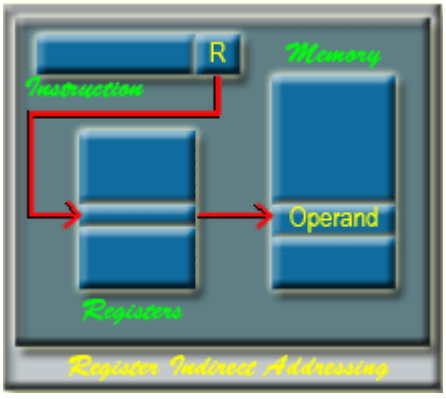
The exact register location of the operand in case of Register Addressing Mode is shown in the Figure 34.4. Here, 'R' indicates a register where the operand is present.

Register Indirect Addressing:

Register indirect addressing is similar to indirect addressing, except that the address field refers to a register instead of a memory location.
It requires only one memory reference and no special calculation.

EA = (R)

Register indirect addressing uses one less memory reference than indirect addressing. Because, the first information is available in a register which is nothing but a memory address. From that memory location, we use to get the data or information. In general, register access is much more faster than the memory access.



Diaplacement Addressing:

A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing, which is broadly categorized as displacement addressing:

EA = A + (R)

Displacement addressing requires that the instruction have two address fields, at least one of which is explicit. The value contained in one address field (value = A) is used directly. The other address field, or an implicit reference based on opcode, refers to a register whose contents are added to A to produce the effective address. The general format of Displacement Addressing is shown in the Figure 4.6.

Three of the most common use of displacement addressing are:

- Relative addressing
- Base-register addressing
- Indexing

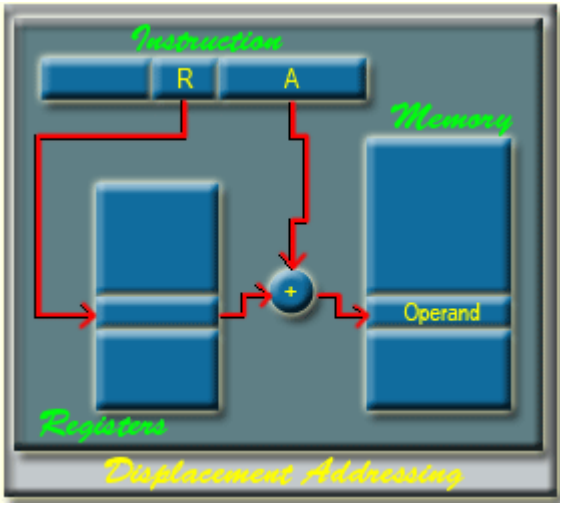


Figure 4.6: Displacement Addressing

Relative Addressing:

For relative addressing, the implicitly referenced register is the *program counter* (PC). That is, the current instruction address is added to the address field to produce the EA. Thus, the effective address is a displacement relative to the address of the instruction.

Base-Register Addressing:

The reference register contains a memory address, and the address field contains a displacement from that address. The register reference may be *explicit* or *implicit*.

In some implimentation, a single segment/base register is employed and is used implicitly. In others, the programmer may choose a register to hold the base address of a segment, and the instruction must reference it explicitly.

Indexing:

The address field references a main memory address, and the reference register contains a positive displacement from that address. In this case also the register reference is sometimes explicit and sometimes implicit.

Generally index register are used for iterative tasks, it is typical that there is a need to increment or decrement the index register after each reference to it. Because this is such a common operation, some system will automatically do this as part of the same instruction cycle.

This is known as *auto-indexing.* We may get two types of auto-indexing:

-- one is auto-incrementing and the other one is
-- auto-decrementing.

If certain registers are devoted exclusively to indexing, then auto-indexing can be invoked implicitly and automatically. If general purpose register are used, the autoindex operation may need to be signaled by a bit in the instruction.

Auto-indexing using *increment* can be depicted as follows:

EA = A + (R)
R = (R) + 1

Auto-indexing using *decrement* can be depicted as follows:

EA = A + (R)
R = (R) - 1

In some machines, both *indirect addressing* and *indexing* are provided, and it is possible to employ both in the same instruction. There are two possibilities: The indexing is performed either before or after the indirection.

If indexing is performed after the indirection, it is termed *postindexing*

EA = (A) + (R)

First, the contents of the address field are used to access a memory location containing an address. This address is then indexed by the register value.

With preindexing, the indexing is performed before the indirection:

EA = (A + (R))

An address is calculated, the calculated address contains not the operand, but the address of the operand.

Stack Addressing:

A stack is a linear array or list of locations. It is sometimes referred to as a *pushdown list* or *last-in-first-out queue*. A 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. 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.

Table 2.1 Generic addressing modes

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute (Direct)	LOC	EA = LOC
Indirect	(R*i*)	EA = [R*i*]
	(LOC)	EA = [LOC]
Index	X(R*i*)	EA = [R*i*] + X
Base with index	(R*i*,R*j*)	EA = [R*i*] + [R*j*]
Base with index and offset	X(R*i*,R*j*)	EA = [R*i*] + [R*j*] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(R*i*)+	EA = [R*i*]; Increment R*i*
Autodecrement	−(R*i*)	Decrement R*i*; EA = [R*i*]

EA = effective address
Value = a signed number

Refer Hamacher for details

Machine Instruction The operation of a CPU is determine by the instruction it executes, referred to as machine instructions or computer instructions. The collection of different instructions is referred as the *instruction set of the CPU*.

Each instruction must contain the information required by the CPU for execution. The elements of an instruction are as follows:

Operation Code:
Specifies the operation to be performed (e.g., add, move etc.). The operation is specified by a binary code, know as the *operation code* or *opcode*.

Source operand reference:
The operation may involve one or more source operands; that is, operands that are inputs for the operation.

Result operand reference:
The operation may produce a result.

Next instruction reference:
This tells the CPU where to fetch the next instruction after the execution of this instruction is complete.

The next instruction to be fetched is located in main memory. But in case of virtual memory system, it may be either in main memory or secondary memory (disk). In most cases, the next instruction to be fetched immediately follow the current instruction. In those cases, there is no explicit reference to the next instruction.When an explicit reference is needed, then the main memory or virtual memory address must be given.

Source and result operands can be in one of the three areas:

- main or virtual memory,
- CPU register or
- I/O device.

The steps involved in instruction execution is shown in the Figure 4.7-

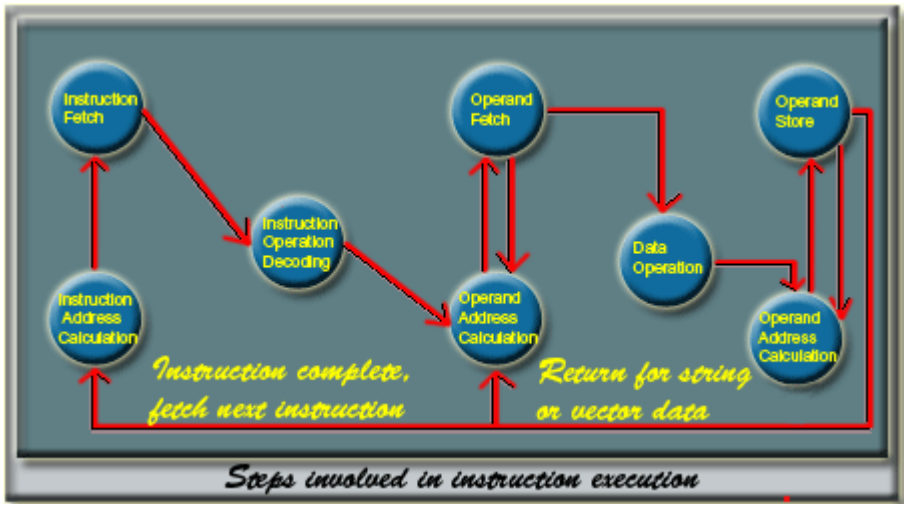


Figure 4.7: Steps involved in instruction execution

Instruction Representation

Within the computer, each instruction is represented by a sequence of bits. The instruction is divided into fields, corresponding to the constituent elements of the instruction. The instruction format is highly machine specific and it mainly depends on the machine architecture. A simple example of an instruction format is shown in the Figure 4.8. It is assume that it is a 16-bit CPU. 4 bits are used to provide the operation code. So, we may have to 16 (2^4 = 16) different set of instructions. With each instruction, there are two operands. To specify each operands, 6 bits are used. It is possible to provide 64 (2^6 = 64) different operands for each operand reference.

It is difficult to deal with binary representation of machine instructions. Thus, it has become common practice to use a symbolic representation of machine instructions.

Opcodes are represented by abbreviations, called *mnemonics*, that indicate the operations. Common examples include:

ADD Add
SUB Subtract
MULT Multiply
DIV Division
LOAD Load data from memory to CPU
STORE Store data to memory from CPU.

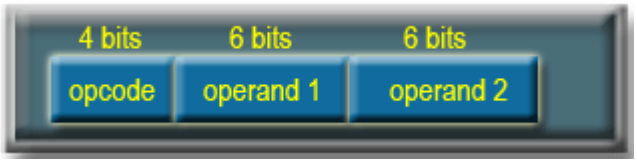


Figure 4.8: A simple instruction format.

Operands are also represented symbolically. For example, the instruction

```
MULT  R, X ; R ← R * X
```

may mean multiply the value contained in the data location X by the contents of register R and put the result in register R

In this example, X refers to the address of a location in memory and R refers to a particular register.

Thus, it is possible to write a machine language program in symbolic form. Each symbolic opcode has a fixed binary representation, and the programmer specifies the location of each symbolic operand.

Instruction Types

The instruction set of a CPU can be categorized as follows:

Data Processing:

Arithmatic and Logic instructions Arithmatic instructions provide computational capabilities for processing numeric data. Logic (Boolean) instructions operate on the bits of a word as bits rather than as numbers. Logic instructions thus provide capabilities for processing any other type of data. There operations are performed primarily on data in CPU registers.

Data Storage:

Memory instructions are used for moving data between memory and CPU registers.

Data Movement:

I/O instructions are needed to transfer program and data into memory from storage device or input device and the results of computation back to the user.

Control:

Test and branch instructions
Test instructions are used to test the value of a data word or the status of a computation. Branch instructions are then used to branch to a different set of instructions depending on the decision made.

Number of Addresses

What is the maximum number of addresses one might need in an instruction? Most of the arithmantic and logic operations are either *unary* (one operand) or *binary* (two operands). Thus we need a maximum of two addresses to reference operands. The result of an operation must be stored, suggesting a third address. Finally after completion of an instruction, the next instruction must be fetched, and its address is needed.

This reasoning suggests that an instruction may require to contain *four address references:* two operands, one result, and the address of the next instruction. In practice, four address instructions are rare. Most instructions have one, two or three operands addresses, with the address of the next instruction being implicit (obtained from the program counter).

Instruction Set Design

One of the most interesting, and most analyzed, aspects of computer design is instruction set design. The instruction set defines the functions performed by the CPU. The instruction set is the programmer's means of controlling the CPU. Thus programmer requirements must be considered in designing the instruction set.

Most important and fundamental design issues:

Operation repertoire	:	How many and which operations to provide, and how complex operations should be.
Data Types	:	The various type of data upon which operations are performed.
Instruction format	:	Instruction length (in bits), number of addresses, size of various fields and so on.
Registers	:	Number of CPU registers that can be referenced by instructions and their use.
Addressing	:	The mode or modes by which the address of an operand is specified.

Types of Operands

Machine instructions operate on data. Data can be categorized as follows :

Addresses: It basically indicates the address of a memory location. Addresses are nothing but the unsigned integer, but treated in a special way to indicate the address of a memory location. Address arithmatic is somewhat different from normal arithmetic and it is related to machine architecture.

Numbers: All machine languages include numeric data types. Numeric data are classified into two broad categories: integer or fixed point and floating point.

Characters: A common form of data is text or character strings. Since computer works with bits, so characters are represented by a sequence of bits. The most commonly used coding scheme is ASCII (American Standard Code for Information Interchange) code.

Logical Data: Normally each word or other addressable unit (byte, halfword, and so on) is treated as a single unit of data. It is sometime useful to consider an *n*-bit unit as consisting of *n* 1-bit items of data, each

item having the value 0 or 1. When data are viewed this way, they are considered to be logical data. Generally 1 is treated as true and 0 is treated as false.

Types of Opearations

The number of different *opcodes* and their types varies widely from machine to machine. However, some general type of operations are found in most of the machine architecture. Those operations can be categorized as follows:

- Data Transfer
- Arithmatic
- Logical
- Conversion
- Input Output [I/O]
- System Control
- Transfer Control

Data Transfer:

The most fundamental type of machine instruction is the data transfer instruction. The data transfer instruction must specify several things. First, the location of the source and destination operands must be specified. Each location could be memory, a register, or the top of the stack. Second, the length of data to be transferred must be indicated. Third, as with all instructions with operands, the mode of addressing for each operand must be specified.

The CPU has to perform several task to accomplish a data transfer operation. If both source and destination are registers, then the CPU simply causes data to be transferred from one register to another; this is an operation internal to the CPU.

If one or both operands are in memory, then the CPU must perform some or all of the following actions:

a) Calculate the memory address, based on the addressing mode.
b) If the address refers to virtual memory, translate from virtual to actual memory address.
c) Determine whether the addressed item is in cache.
d) If not, issue a command to the memory module.

Commonly used data transfer operation:

Operation Name	Description
Move (Transfer)	Transfer word or block from source to destination
Store	Transfer word from processor to memory

Load (fetch)	Transfer word from memory to processor
Exchange	Swap contents of source and destination
Clear (reset)	Transfer word of 0s to destination
Set	Transfer word of 1s to destination
Push	Transfer word from source to top of stack
Pop	Transfer word from top of stack to destination

Arithmatic:

Most machines provide the basic arithmetic operations like add, subtract, multiply, divide etc. These are invariably provided for signed integer (fixed-point) numbers. They are also available for floating point number.

The execution of an arithmetic operation may involve data transfer operation to provide the operands to the ALU input and to deliver the result of the ALU operation.

Commonly used data transfer operation:

Operation Name	Description
Add	Compute sum of two operands
Subtract	Compute difference of two operands
Multiply	Compute product of two operands
Divide	Compute quotient of two operands
Absolute	Replace operand by its absolute value
Negate	Change sign of operand
Increment	Add 1 to operand
Decrement	Subtract 1 from operand

Logical:

Most machines also provide a variety of operations for manipulating individual bits of a word or other addressable units.

Most commonly available logical operations are:

Operation Name	Description
AND	Performs the logical operation AND bitwise

OR	Performs the logical operation OR bitwise
NOT	Performs the logical operation NOT bitwise
Exclusive OR	Performs the specified logical operation Exculsive-OR bitwise
Test	Test specified condition; set flag(s) based on outcome
Compare	Make logical or arithmatic comparison Set flag(s) based on outcome
Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control etc.
Shift	Left (right) shift operand, introducing constant at end
Rotate	Left (right) shift operation, with wraparound end

Conversion:

Conversion instructions are those that change the format or operate on the format of data. An example is converting from decimal to binary.

Input/Output :

Input/Output instructions are used to transfer data between input/output devices and memory/CPU register.

Commonly available I/O operations are:

Operation Name	Description
Input (Read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
Output (Write)	Transfer data from specified source to I/O port or device.
Start I/O	Transfer instructions to I/O processor to initiate I/O operation.
Test I/O	Transfer status information from I/O system to specified destination

System Control:

System control instructions are those which are used for system setting and it can be used only in privileged state. Typically, these instructions are reserved for the use of operating systems. For example, a system control instruction may read or alter the content of a control register. Another instruction may be to read or modify a storage protection key.

Transfer of Control:

In most of the cases, the next instruction to be performed is the one that immediately follows the current instruction in memory. Therefore, program counter helps us to get the next instruction. But sometimes it is required to change the sequence of instruction execution and for that instruction set should provide instructions to accomplish these tasks. For these instructions, the operation performed by the CPU is to upload the program counter to contain the address of some instruction in memory. The most common transfer-of-control operations found in instruction set are: branch, skip and procedure call.

Branch Instruction

A branch instruction, also called a jump instruction, has one of its operands as the address of the next instruction to be executed. Basically there are two types of branch instructions: Conditional Branch instruction and unconditional branch instruction. In case of unconditional branch instruction, the branch is made by updating the program counter to address specified in operand. In case of conditional branch instruction, the branch is made only if a certain condition is met. Otherwise, the next instruction in sequence is executed. There are two common ways of generating the condition to be tested in a conditional branch instruction **First** most machines provide a 1-bit or multiple-bit condition code that is set as the result of some operations. As an example, an arithmetic operation could set a 2-bit condition code with one of the following four values: zero, positive, negative and overflow. On such a machine, there could be four different conditional branch instructions:

```
BRP    X ;    Branch to location  X   if result is positive
BRN    X ;   Branch to location  X   if result is negative
BRZ    X ;    Branch to location  X   is result is zero
BRO   X ;    Branch to location  X   if overflow occurs
```

In all of these cases, the result referred to is the result of the most recent operation that set the condition code.

Another approach that can be used with three address instruction format is to perform a comparison and specify a branch in the same instruction.

For example,

```
BRE   R1,   R2,   X  ;  Branch to   X      if contents of R1 = Contents of R2.
```

Skip Instruction

Another common form of transfer-of-control instruction is the skip instruction. Generally, the skip imples that one instruction to be skipped; thus the implied address equals the address of the next instruction plus one instruction length. A typical example is the increment-and-skip-if-zero (ISZ) instruction. For example,

```
ISZ    R1
```

This instruction will increment the value of the register R1. If the result of the increment is zero, then it will skip the next instruction.

Procedure Call Instruction

A procedure is a self contained computer program that is incorporated into a large program. At any point in the program the procedure may be invoked, or called. The processor is instructed to go and execute the entire procedure and then return to the point from which the call took place.

The procedure mechanism involves two basic instructions: a call instruction that branches from the present location to the procedure, and a return instruction that returns from the procedure to the place from which it was called. Both of these are forms of branching instructions.
Some important points regarding procedure call:

- A procedure can be called from more than one location.
- A procedure call can appear in a procedure. This allows the nesting of procedures to an arbitrary depth.
- Each procedure call is matched by a return in the called program.

Since we can call a procedure from a variety of points, the CPU must somehow save the return address so that the return can take place appropriately. There are three common places for storing the return address:

- Register
- Start of procedure
- Top of stack

Consider a machine language instruction CALL X, which stands for call procedure at location X. If the register apprach is used, CALL X causes the following actions:

RN ← PC + IL

PC ← X

where RN is a register that is always used for this purpose, PC is the program counter and IL is the instruction length. The called procedure can now save the contents of RN to be used for the later return.

A second possibilities is to store the return address at the start of the procedure. In this case, CALL X causes

X ← PC + IL

PC ← X + 1

Both of these approaches have been used. The only limitation of these approaches is that they prevent the use of reentrant procedures. A reentrant procedure is one in which it is possible to have several calls open to it at the same time.

A more general approach is to use stack. When the CPU executes a call, it places the return address on the stack. When it executes a return, it uses the address on the stack.

It may happen that, the called procedure might have to use the processor registers. This will overwrite the contents of the registers and the calling environment will lose the information. So, it is necessary to preserve the contents of processor register too along with the return address. The stack is used to store the contents of processor register. On return from the procedure call, the contents of the stack will be popped out to appropriate registers.

In addition to provide a return address, it is also often necessary to pass parameters with a procedure call. The most general approach to parameter passing is the stack. When the processor executes a call, it not only stacks the return address, it stacks parameters to be passed to the called procedures. The called procedure can access the parameters from the stack.Upon return, return parameters can also be placed on the stack.

The entire set of parameters, including return address, that is stored for a procedure invocation is referred to as stack frame.

Most commonly used transfer of control operation:

Operation Name	Description
Jump (branch)	Unconditional transfer, load PC with specific address
Jump conditional	Test specific condition; either load PC with specific address or do nothing, based on condition
Jump to subroutine	Place current program control information in known location; jump to specific address
Return	Replace contents of PC and other register from known location
Skip	Increment PC to skip next instruction
Skip Conditional	Test specified condition; either skip or do nothing based on condition
Halt	Stop program execution

Instruction Format:

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 explit operand is referenced using one of the addressing mode that is available for that machine. The format must, implicitly or explictly, indicate the addressing mode of each operand. For most instruction sets, more than one instruction format is used. Four common instruction format are shown in the Figure 4.9.

Figure 4.9: Four common Instruction formats

Instruction Length:

On some machines, all instructions have the same length; on others there may be many different lengths. Instructions may be shorter than, the same length as, or more than the word length. Having all the instructions be the same length is simpler and make decoding easier but often wastes space, since all instructions then have to be as long as the longest one. Possible relationship between instruction length and word length is shown in the Figure 4.10.

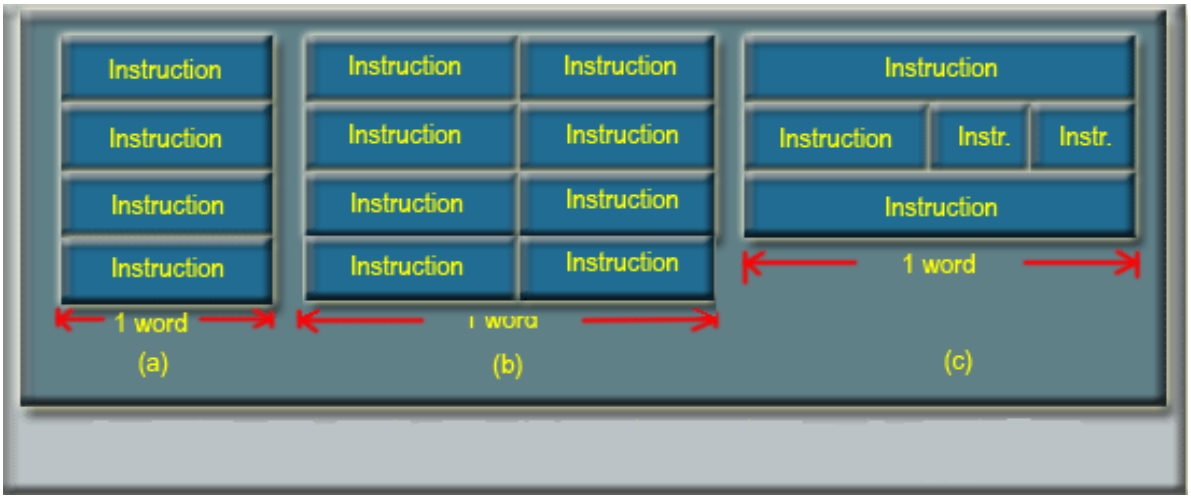


Figure 4.10: Some Possible relationship between instructions and word length

Generally there is a correlation between memory transfer length and the instruction length. Either the instruction length should be equal to the memory transfer length or one should be a multiple of the other. Also in most of the case there is a correlation between memory transfer length and word length of the machine.

Allocation of Bits:

For a given instruction length, there is a clearly a trade-off between the number of opcodes and the power of the addressing capabilities. More opcodes obviously mean more bits in the opcode field. For an instruction format of a given length, this reduces the number of bits available for addressing.
The following interrelated factors go into determining the use of the addressing bits:

Number of Addressing modes:

Sometimes as addressing mode can be indicated implicitly. In other cases, the addressing mode must be explicit, and one or more bits will be needed.

Number of Operands:

Typical instructions on today's machines provide for two operands. Each operand address in the instruction might require its own mode indicator, or the use of a mode indicator could be limited to just one of the address field.

Register versus memory:

A machine must have registers so that data can be brought into the CPU for processing. With a single user-visible register (usually called the accumulator), one operand address is implicit and consumes no instruction bits. Even with multiple registers, only a few bits are needed to specify the register. The more that registers can be used for operand references, the fewer bits are needed.

Number of register sets:

A number of machines have one set of general purpose registers, with typically 8 or 16 registers in the set. These registers can be used to store data and can be used to store addresses for displacement addressing. The trend recently has been away from one bank of general purpose registers and toward a collection of two or more specialized sets (such as data and displacement).

Address range:

For addresses that reference memory, the range of addresses that can be referenced is related to the number of address bits. With displacement addressing, the range is opened up to the length of the address register.

Address granularity:

In a system with 16- or 32-bit words, an address can reference a word or a byte at the designer's choice. Byte addressing is convenient for character manipulation but requires, for a fixed size memory, more address bits.

Variable-Length Instructions:

Instead of looking for fixed length instruction format, designer may choose to provide a variety of instructions formats of different lengths. This tectic makes it easy to provide a large repertoire of opcodes, with different opcode lengths. Addressing can be more flexible, with various combinations of register and memory references plus addressing modes. With variable length instructions, many variations can be provided efficiently and compactly. The principal price to pay for variable length instructions is an increase in the complexity of the CPU.

Number of addresses :

The processor architecture is described in terms of the number of addresses contained in each instruction. Most of the arithmetic and logic instructions will require more operands. All arithmatic and logic operations are either unary
(one source operand, e.g. NOT) or binary (two source operands, e.g. ADD).

Thus, we need a maximum of two addresses to reference source operands. The result of an operation must be stored, suggesting a third reference.

Three address instruction formats are not common because they require a relatively long instruction format to hold the three address reference.

With two address instructions, and for binary operations, one address must do double duty as both an operand and a result.

In one address instruction format, a second address must be implicit for a binary operation. For implicit reference, a processor register is used and it is termed as accumulator(AC). the accumulator contains one of the operands and is used to store the result.

Consider a simple arithmatic expression to evalute:
Y= (A + B) / (C * D)

The evaluation of this expression in three address instruction format, two address instruction format and one address instruction format is shown in the Figure 4.11, Figure 4.12 and Figure 4.13 respectively.

Figure 4.11: Three address instructions

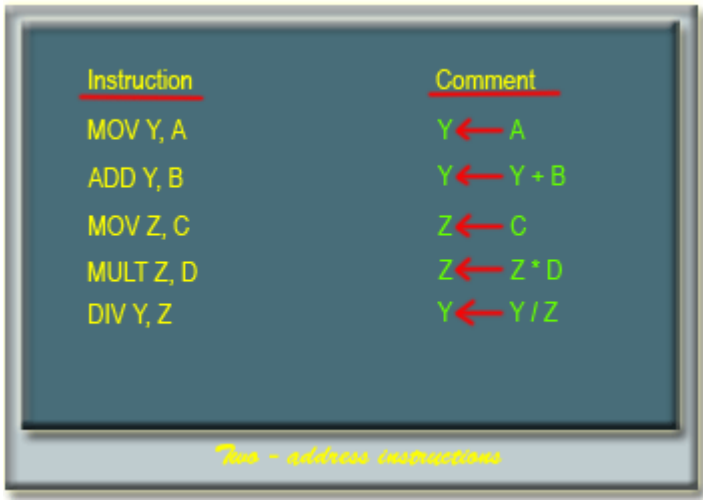


Figure 4.12: Two address instructions

Figure 4.13: One address instructions