Unit-5
Addressing Modes

The different ways of specifying the location of an operand in an instruction are called as **Addressing Modes**.
The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.
Generally, the programs are written in a high-level language, as it is a convenient way to define the variables and operations that the programmer needs to perform on the variables. Later, this program is compiled to generate the machine code. The machine code has low-level instructions. The low-level instruction has **opcode** and **operands**. Addressing mode has nothing to do with the opcode part. It focuses on presenting the operand's address in the instructions.

Types of Addressing Modes:

There are following types of addressing modes-

1. Register Addressing Mode
2. Direct Addressing Mode
3. Immediate Addressing Mode
4. Register Indirect Addressing Mode
5. Index Addressing Mode
6. Auto Increment Mode
7. Auto Decrement Mode
8. Relative Addressing Mode

Before discussing the addressing modes, we must know about the term "**effective address**".

Effective Address (EA):

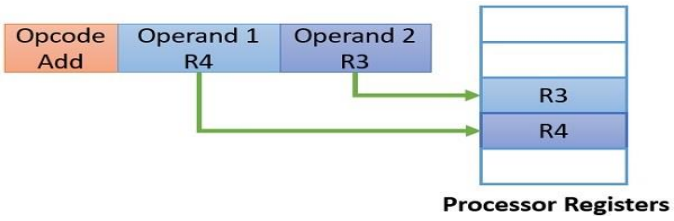
Effective address is the **address of the exact memory location** where the **value of the operand** is present.

1. Register Addressing Mode-

Every instruction includes operands; the operands can be a memory location, a processor register or an I/O device. The instruction which uses processor **registers** to represent operands is the instruction in **register addressing mode**.
Here, the effective address is a register where the value of the operand is present.
EA=R
Below we have two instructions as our examples for register addressing mode.
Add R4, R3
Load R3, R2

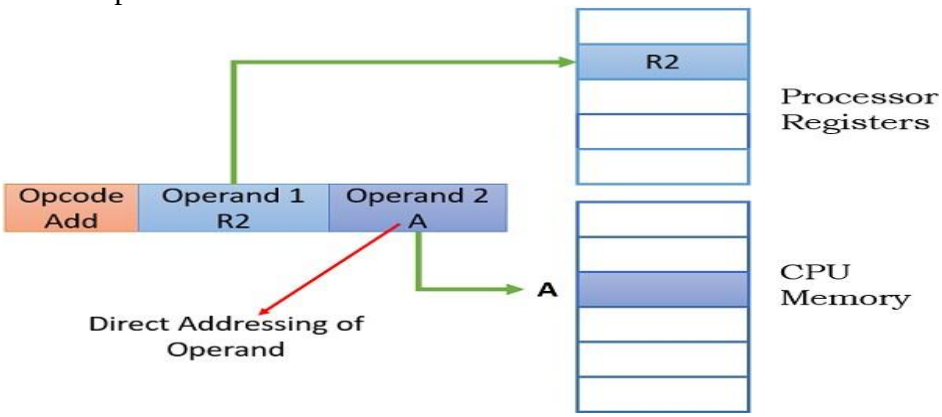
In the examples above, the Add instruction uses registers to represent both of its operands. Similarly, the Load instruction also uses registers to represent both of its operands. So, the instruction above uses register addressing mode to describe the address of the operand.

Below, we have a figure showing the Add instruction in the example above.



2. Direct Addressing Mode-

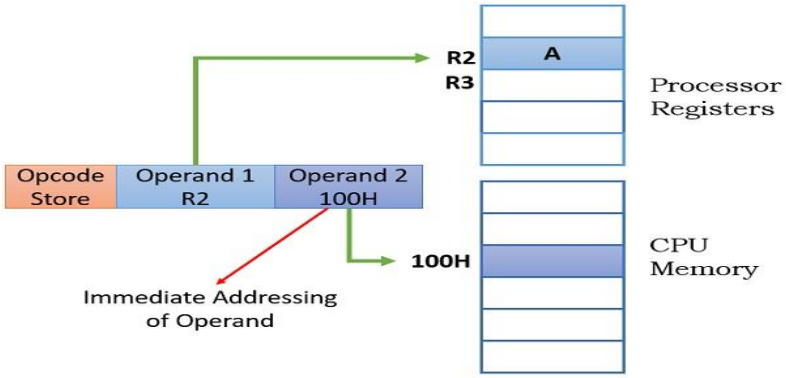
The direct addressing mode is also known as **Absolute Addressing mode**. Here, the instruction contains the address of the **location in memory** where the value of the operand is stored.
Here, the effective address is the address of memory location.
EA = A
For example, observe the examples below:
Add R2, A
Store R2, B
The Add instruction includes the memory location A which has the value to be added to the content of register R2. Similarly, the Store instruction has the address of memory location B where the content of register R2 will be stored.
Below we have a figure showing the direct addressing of the operand A in the Add instruction of the example above.



3. Immediate Addressing Mode-

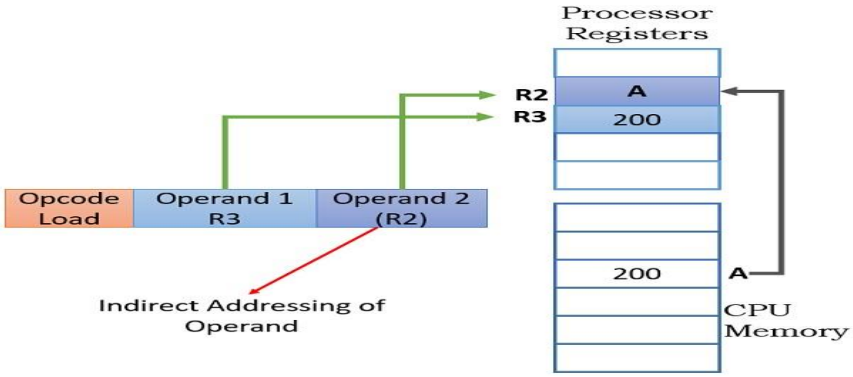
In immediate addressing mode, the value of the operand is **explicitly mentioned** in the instruction. Here, effective address is not required as the operand is explicitly defined in instruction.

Let us see the example of immediate addressing mode:
Add R2, #100
Store R2, 100H
The Add instruction, adds 100 to R2's content . The # sign in front of the value indicates the immediate value to be operated. If a value does not have # sign in front of it then it is the address of a memory location.
The next instruction Store considers the immediate value 100H as address as it does not have # sign in front of it. The Store instruction stores the content of R2 at memory location 100H.
In the figure below we have shown the Store instruction of the above examples.



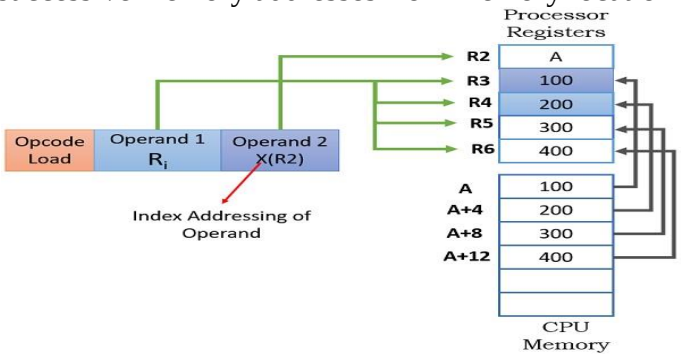
4. Register Indirect Addressing mode-

A **processor register** is used to hold the **address of a memory location** where the operand is placed. This addressing mode allows executing the same set of instructions for the different memory location. This can be done by incrementing the content of register thereby pointing the new location each time.
In higher-level language, it is referred to as pointers. The indirect mode is denoted by placing the register inside the parenthesis.
Here the effective address is the content of memory location present in the register.
EA=(R)
Now, for example:
Load R3, (R2) *// Load R2, A*
The Load instruction above will load the value present at the memory location contained by register R2 in the register R3.
The figure below shows how the register R3 gets loaded with the value stored in the memory location held by register R2.



5. Index Addressing Mode

Index addressing mode is helpful when the instructions in the program are accessing the **array or the large range of memory addresses**. In this mode, the effective address is generated by **adding a constant to the register's content**. The content of the register does not change.
The symbolic representation of index addressing mode is denoted as:
X(R)
And the effective address is denoted by
EA = X + (R)
For example, consider the instruction below:
Load R2, A
Load R3, (R2)
Load R4, 4(R2)
Load R5, 8(R2)
Load R6, 12(R2)
The above instructions will load the register R3, R4, R5, R6 with the contents, present at the successive memory addresses from memory location A correspondingly.



6. Auto Increment Addressing Mode

In auto-increment addressing mode once the content of the register is **accessed** by the instruction the register's content is **incremented** to refer the next operand.
Symbolically it is represented as below:
(R)+
Here, the effective address is content of the register as it is enclosed by parenthesis. The content of register which is referring to a memory location is incremented so that it could point the next

memory location where the next operand is stored.

7. Auto Decrement Addressing Mode-

It is just opposite of auto-increment mode. In auto decrement mode the content of the register is **decremented initially** and then the decremented content of the register is used as effective address.
Symbolically it is presented as:
-(R)
The auto-increment and decrement mode help to implement the **stack structure**.

8. Relative Addressing Mode-

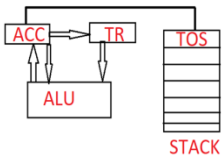
In the content above we have discussed the **index addressing mode**. There we were adding a constant to the register content to refer the next operand address. In some computer instead of a register, the **program counter** is used.
The symbolic representation of relative address mode is
X(PC)
The effective address for it would be:
EA = X + (PC)
As here the operand addresses are found relative to the program counter. That's why it is referred to as relative address mode.

Instruction Format

Instruction format describes the internal structures (layout design) of the bits of an instruction, in terms of its constituent parts. It must include an opcode, and address is dependent on an availability of particular operands.
Instruction formats are classified into 5 types based on the type of the CPU organization. CPU organization is divided into three types based on the availability of the ALU operands, which are as follows here:

1) STACK CPU:

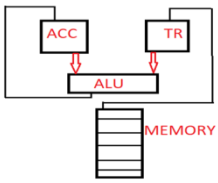
In this organization, ALU operands are performed only on a stack data. This means that both of the ALU operations are always required in the stack. The same stack is also used as the destination. In the stack, we can perform insert and deletion operation at only one end which is called as the top of a stack. So in this format, there is no need of address because in this TOS becomes the default location.



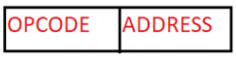
In this organization, only the ALU operands are zero address operation whereas data transfer instructions are not a zero address instruction. The computable instruction format of STACK CPU is **Zero Address Instruction Format**.

2) Accumulator CPU:

In this organization, one of the ALU operands is always present in the accumulator. The same accumulator is also used as the destination. Another ALU operand is present either in the register or in memory. In processor design, only one accumulator is present so it becomes the default location.



The computable instruction format of Accumulator CPU is **One Address Instruction Format**.



3) General Register CPU:

Based on the number of the registers possible in the processors, the architecture is divided into two types:

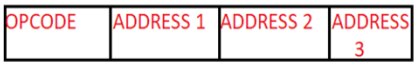
i. **Register-Memory references CPU**
ii. **Register-Register references CPU**

i) Register-Memory Reference CPU-

In this architecture, processors support less number of registers. Therefore register file size is small. In this organization, the first ALU operand is always required in the register. The same register can also be used as the destination. The second ALU operand is present either in a register or in memory. The computable instruction format of the register to memory reference CPU is **Two Address Instruction Format**.

ii) Register-Register Reference CPU-

In this architecture, processors support number of registers, therefore, register file size is large. In this organization, ALU operands are performed only on a registers data that means both of the ALU operands are required in the register. Due to more number of register present in the CPU, the separate register is used to store the result. The computable instruction format of Register-Register Reference CPU is Three Address Instruction Format.



Four Address instruction format

This format contains the 4 different address fields with an opcode. Since PC is used as the mandatory register in the CPU design which is used to hold the next instruction address. So four instruction format is not in the use.

Unit-5

Types of CPU Organization

➢ General Register based CPU Organization:

When we are using multiple general purpose registers, instead of single accumulator register, in the CPU Organization then this type of organization is known as General register based CPU Organization.
In this type of organization, computer uses two or three address fields in their instruction format. Each address field may specify a general register or a memory word.
For example:

```
MULT R1, R2, R3
```

This is an instruction of an arithmatic multiplication written in assembly language. It uses three address fields R1, R2 and R3.

The meaning of this instruction is:

```
R1 <-- R2 * R3
```

This instruction also can be written using only two address fields as:

```
MULT R1, R2
```

In this instruction, the destination register is the same as one of the source registers.

This means the operation

```
R1 <-- R1 * R2
```

The use of large number of registers results in short program with limited instructions.

Some examples of General register based CPU Organization are **IBM 360 and PDP- 11**.

Advantages of General register based CPU organization –

- Efficiency of CPU increases as there are large number of registers are used in this organization.
- Less memory space is used to store the program since the instructions are written in compact way.

Disadvantages of General register based CPU organization –

- Care should be taken to avoid unnecessary usage of registers. Thus, compilers need to be more intelligent in this aspect.
- Since large number of registers are used, thus extra cost is required in this organization.

➢ Stack based CPU Organization:

The computers which use Stack-based CPU Organization are based on a data structure called **stack**. The stack is a list of data words. It uses **Last In First Out (LIFO)** access method which is the most popular access method in most of the CPU. A register is used to store the address of the topmost element of the stack which is known as **Stack pointer (SP)**.
In this organisation, ALU operations are performed on stack data. It means both the operands are always required on the stack. After manipulation, the result is placed in the stack.
The main two operations that are performed on the operators of the stack are **Push** and **Pop**. These two operations are performed from one end only.

1. **Push –**

   This operation results in inserting one operand at the top of the stack and it decrease the stack pointer register.
   The format of the PUSH instruction is:

   ```
   PUSH
   ```

   It inserts the data word at specified address to the top of the stack.

It can be implemented as:

```
//decrement SP by 1
SP <-- SP - 1


//store the content of specified memory address
//into SP; i.e, at top of stack
SP <-- (memory address)
```

2. **Pop –**
   This operation results in deleting one operand from the top of the stack and it increase the stack pointer register.
   The format of the POP instruction is:

```
POP
```

It deletes the data word at the top of the stack to the specified address.

It can be implemented as:

```
//transfer the content of  SP (i.e, at top most data)
//into specified memory location
(memory address) <-- SP


//increment SP by 1
SP <-- SP + 1
```

Operation type instruction does not need the address field in this CPU organization. This is because the operation is performed on the two operands that are on the top of the stack.

For example:

```
SUB
```

This instruction contains the opcode only with no address field. It pops the two top data from the stack, subtracting the data, and pushing the result into the stack at the top.

PDP-11, Intel's 8085 and HP 3000 are some of the examples of the stack organized computers.

Advantages of Stack based CPU organization –

- Efficient computation of complex arithmetic expressions.
- Execution of instructions is fast because operand data are stored in consecutive memory locations.
- Length of instruction is short as they do not have address field.

Disadvantages of Stack based CPU organization –

- The size of the program increases.

Note:Stack based CPU organisation uses zero address instruction.

➢ Single Accumulator based CPU organization

In this type of CPU organization, the accumulator register is used implicitly for processing all instructions of a program and store the results into the accumulator. The instruction format that is used by this CPU Organisation is **One address field**. Due to this the CPU is known as **One Address Machine**.
The main points about Single Accumulator based CPU Organisation are:

1. In this CPU Organization, the first ALU operand is always stored into the Accumulator and the second operand is present either in Registers or in the Memory.
2. Accumulator is the default address thus after data manipulation the results are stored into the accumulator.
3. One address instruction is used in this type of organization.

The format of instruction is: Opcode + Address

Opcode indicates the type of operation to be performed.

Mainly two types of operation are performed in single accumulator based CPU organization:

1. **Data transfer operation –**
   In this type of operation, the data is transferred from a source to a destination.
   For ex: LOAD X, STORE Y

   Here LOAD is memory read operation that is data is transfer from memory to accumulator and STORE is memory write operation that is data is transfer from accumulator to memory.
2. **ALU operation –**
   In this type of operation, arithmetic operations are performed on the data.
   For ex: MULT X

   where X is the address of the operand. The MULT instruction in this example performs the operation,

   AC <-- AC * M[X]

   AC is the Accumulator and M[X] is the memory word located at location X.

This type of CPU organization is first used in **PDP-8 processor** and is used for process control and laboratory applications. It has been totally replaced by the introduction of the new general register based CPU.

Advantages –

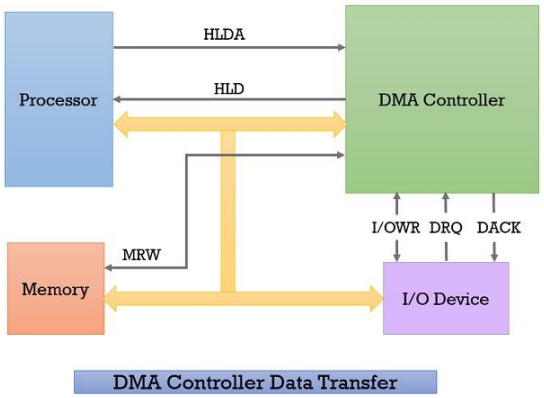
- One of the operands is always held by the accumulator register. This results in short instructions and less memory space.
- Instruction cycle takes less time because it saves time in instruction fetching from memory.

Disadvantages –

- When complex expressions are computed, program size increases due to the usage of many short instructions to execute it. Thus memory size increases.
- As the number of instructions increases for a program, the execution time increases.

Direct memory access (DMA)

Direct Memory Access (DMA) transfers the block of data between the memory and peripheral devices of the system, without the participation of the processor. The unit that controls the activity of accessing memory directly is called a DMA controller.
So, the DMA controller can accomplish the task of data transfer via the system bus.



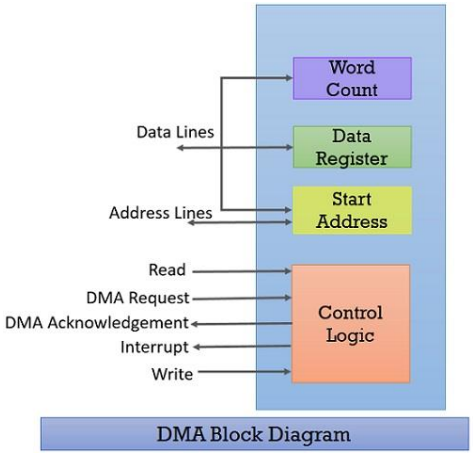
DMA Controller Data Transfer

1. Whenever an I/O device wants to transfer the data to or from memory, it sends the DMA request (**DRQ**) to the DMA controller. DMA controller accepts this DRQ and asks the CPU to hold for a few clock cycles by sending it the Hold request (**HLD**).
2. CPU receives the Hold request (HLD) from DMA controller and relinquishes the bus and sends the Hold acknowledgement (**HLDA**) to DMA controller.
3. After receiving the Hold acknowledgement (HLDA), DMA controller acknowledges I/O device **(DACK)** that the data transfer can be performed and DMA controller takes the charge of the system bus and transfers the data to or from memory.

4. When the data transfer is accomplished, the DMA raise an **interrupt** to let know the processor that the task of data transfer is finished and the processor can take control over the bus again and start processing where it has left.

Direct Memory Access Diagram:

We have a block diagram of DMA controller.



Whenever a processor is requested to read or write a block of data, i.e. transfer a block of data, it instructs the DMA controller by sending the following information.

1. The first information is whether the data has to be read from memory or the data has to be written to the memory. It passes this information via **read or write control lines** that is between the processor and DMA controllers **control logic unit**.
2. The processor also provides the **starting address** of/ for the data block in the memory, from where the data block in memory has to be read or where the data block has to be written in memory. DMA controller stores this in its **address register**. It is also called the **starting address register**.
3. The processor also sends the **word count**, i.e. how many words are to be read or written. It stores this information in the **data count** or the **word count** register.
4. The most important is the **address of I/O device** that wants to read or write data. This information is stored in the **data register.**

Types of DMA transfer:

DMA performs data transfer operation. The different DMA transfer modes are as follows:-

- Burst or block transfer DMA
- Cycle steal or single byte transfer DMA.

1) Burst or block transfer DMA

- It is the fastest DMA mode. In this two or more data bytes are transferred continuously.

- Processor is disconnected from system bus during DMA transfer. N number of machine cycles are adopted into the machine cycles of the processor where N is the number of bytes to be transferred.
- DMA sends HOLD signal to processor to request for system bus and waits for HLDA signal.
- After receiving HLDA signal, DMA gains control of system bus and transfers one byte. After transferring one byte, it increments memory address, decrements counter and transfers next byte.
- In this way, it transfer all data bytes between memory and I/O devices. After transferring all data bytes, the DMA controller disables HOLD signal & enters into slave mode.

2) Cycle steal or single byte transfer DMA.

- In this mode only one byte is transferred at a time. This is slower than burst DMA.
- DMA sends HOLD signal to processor and waits for HLDA signal on receiving HLDA signal, it gains control of system bus and executes only one DMA cycle.
- After transfer one byte, it disables HOLD signal and enters into slave mode.
- Processor gains control of system bus and executes next machine cycle. If count is not zero and data is available then the DMA controller sends HOLD signal to the processor and transfer next byte of data block.

Difference between Burst Mode and Cycle Stealing Mode of DMA

Parameter	Burst Mode of DMA	Cycle Stealing Mode of DMA
Definition	It is the DMA data transfer technique in which no. of data words are transferred continuously until whole data is not transferred.	It is the data transfer technique in which one data word is transferred and then control is returned to CPU.
Data Transfer	Data transfer Continues until whole data is not transferred.	Data is transferred Only when CPU is idle.
Speed	This is very fast data transfer technique and is used to transfer data for fast speed devices.	It is the slow data transfer technique as data is transferred only when CPU is idle
CPU Utilization	Low CPU Utilization because CPU remains idle until whole data is not transferred.	High CPU utilization because data is transferred when CPU has no task to perform.

Extra Overhead	No need to check CPU idleness	Extra Overhead because every time CPU has to be monitored for idle periods or slots.

Advantages of DMA:

1. Transferring the data without the involvement of the processor will **speed up** the read-write task.
2. DMA **reduces the clock cycle** requires to read or write a block of data.
3. Implementing DMA also **reduces the overhead** of the processor.

Disadvantages of DMA

1. As it is a hardware unit, it would **cost** to implement a DMA controller in the system.
2. Cache **coherence** problem can occur while using DMA controller