Unit-2

Part-1

PROCESSOR ORGANIZATION

Classification of Processors

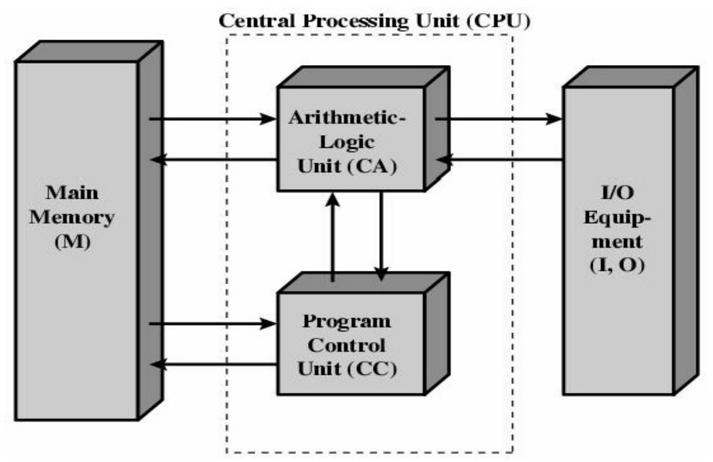
- **Categorized by memory organization**
  - Von-Neumann architecture
  - Harvard architecture
- **Categorized by instruction type**
  - CISC
  - RISC
  - VLIW

Von Neumann Model

- In 1946, John von Neumann and his colleagues began the design of a new stored program computer referred to as the IAS (Institute for Advanced Study) computer or Instruction Set Architecture (ISA).
- Stores program and data in same memory.
- It was designed to overcome the limitation of previous proposed computers.
- The limitation of old computers

  The task of entering and altering programs was extremely tedious.

Structure of IAS/ISA computer


Central Processing Unit (CPU)
Main Memory (M)
Arithmetic-Logic Unit (CA)
Program Control Unit (CC)
I/O Equip-ment (I, O)

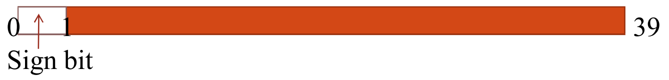
Structure of IAS/ISA computer

IAS/ISA consists of-

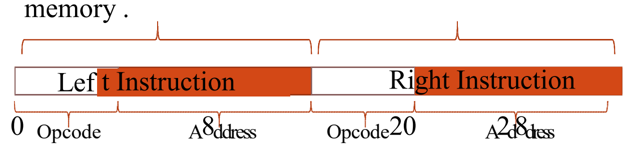
- A main memory, which stores both data and instructions
- An ALU capable of operating on binary data
- A control unit, which interprets the instructions in memory and causes them to be executed
- I/O equipment operated by the control unit

IAS Memory Formats

- The memory of IAS consists of 1000 storage locations, called words, of 40 binary digits(bits) each.
- Both data and instructions are stored there.
- Each number is represented by a sign bit and a 39-bit value.



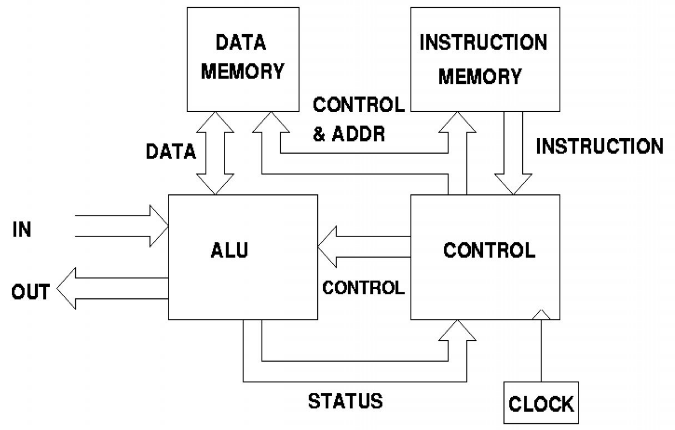
- A word may also contain two 20-bit instructions, with each instruction consisting of an 8-bit operation code(opcode) specifying the operation to be performed and a12-bit address designating one of the words in memory .



Harvard Architecture

- Physically separate storage and signal pathways for instructions and data.
- Originated from the Harvard Mark I relay-based computer, which stored
  - Instructions on punched tape (24 bits wide)
  - Data in electro-mechanical counters
- In some systems, instructions can be stored in read-only memory while data memory generally requires read-write memory.
- In some systems, there is much more instruction memory than data memory.
- Used in MIPS etc.

Harvard Architecture



Register Organization

- CPU must have some working space (temporary storage) called **registers**.
- A computer system employs **a memory hierarchy**.
- At the **highest level** of hierarchy, **memory is faster, smaller and more expensive.**
- Within the CPU, there is **a set of registers** which can be treated as a memory in the **highest level of hierarchy**.

Register Organization

- The registers in the CPU can be categorized into two groups

1. User-visible registers:

- – These enables the machine - or assembly-language programmer to minimize main memory reference by optimizing use of registers.

2. Control and status registers:

- – These are used by the control unit to control the operation of the CPU.
- – Operating system programs may also use these in privileged mode to control the execution of program.

User-visible registers

- General Purpose
- Data
- Address
- Condition Codes

1. **General Purpose Registers:**
   - Used for a variety of functions by the programmer.
   - Sometimes used for holding **operands(data) of an instruction**.
   - Sometimes used for **addressing functions** (e.g., register indirect, displacement).

2. **Data registers:**
   - Used to hold **only data.**
   - **Cannot** be employed in the calculation of an operand address.

3. Address registers:

- Used exclusively for the purpose of **addressing**.
- Examples include the following:

1. Segment pointer:

- In a machine with segment addressing, a segment register holds the **address of the base of the segment**.
- There may be multiple registers, one for the code segment and one for the data segment.

2. Index registers:

- These are used for **indexed addressing** and may be **auto indexed.**

3. Stack pointer:

- A dedicated register that points to the top of the stack.
- Auto incremented or auto decremented using **PUSH** or **POP operation**

4. Condition Codes Register:

- Sets of individual bits
  - e.g. result of last operation was zero
- Can be read (implicitly) by programs
  - e.g. Jump if zero
- Can not (usually) be set by programs

Control and status registers

- **Four registers are essential for instruction execution:**

1. Program Counter (PC):

- Contains the address of an instruction to be fetched.

2. Instruction Register (IR):

- Contains the instruction most recently fetched.

3. Memory Address Register (MAR):

- Contains the address of a location of main memory from where information has to be fetched or information has to be stored.

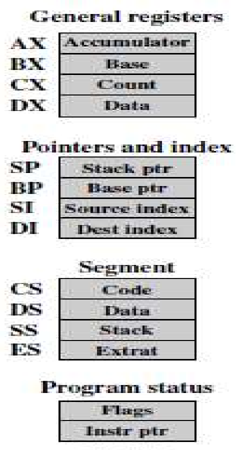
4. Memory Buffer Register (MBR):

- Contains a word of data to be written to memory or the word most recently read.

Control and status registers

- **Program Status Word (PSW)**
  - Condition code bits are collected into one or more registers, known as the **program status word (PSW),** that contains 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 zero.
    - **Carry:** Set if an operation resulted in a carry (addition) into or borrow (subtraction) out of a high order bit.
    - **Equal:** Set if a logical compare result is equal.
    - **Overflow:** Used to indicate arithmetic overflow.
    - **Interrupt enable/disable:** Used to enable or disable interrupts.

Register organization of INTEL 8086 processor



(b) 8086

Register organization of INTEL 8086 processor

- **16-bit flags, Instruction Pointer**
- **General Purpose Registers, 16 bits**
  - AX – Accumulator, favored in calculations
  - BX – Base, normally holds an address of a variable or func
  - CX – Count, normally used for loops
  - DX – Data, normally used for multiply/divide
- **Segment, 16 bits**
  - SS – Stack, base segment of stack in memory
  - CS – Code, base location of code
  - DS – Data, base location of variable data
  - ES – Extra, additional location for memory data

Register organization of INTEL 8086 processor

- **Index, 16 bits**
  - BP – Base Pointer, offset from SS for locating subroutines
  - SP – Stack Pointer, offset from SS for top of stack
  - SI – Source Index, used for copying data/strings
  - DI – Destination Index, used for copy data/strings

INSTRUCTION FORMAT

- The operation of the computer system are determined by **the instructions** executed by the central processing unit.
- These instructions are known as **machine instruction** and are in the form of **binary codes**.
- Each instruction of the CPU has specific **information field** which are required to execute it.
- These information field of instructions are called **elements of instruction**.

➢Elements of Instruction

1. **Operation Code:**

   Binary code that specifies which operation to be performed.
2. **Source operand address:**

   Specifies one or more source operands
3. **Destination operand address:**

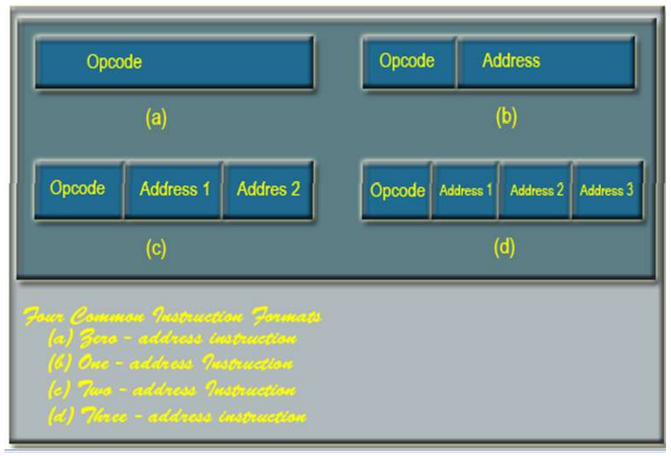
   The operation executed by the CPU may produce result which is stored in the destination address.
4. **Next instruction address:**

   Tells the CPU from where to fetch the next instruction after completion of execution of current instruction.

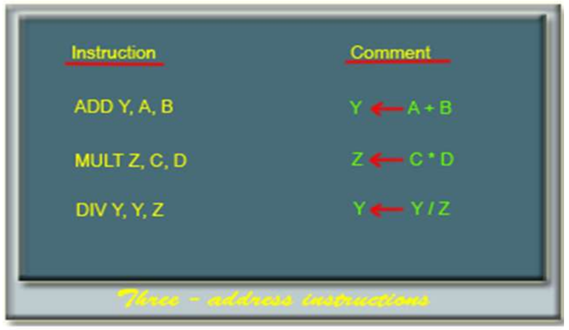
➢Pictorial Representation of Instruction

Opcode	Operand address1	Operand address2

➢Instruction Types According to Number of Addresses


Opcode
(a)
Opcode
Address
(b)
Opcode
Address 1
Addres 2
(c)
Opcode
Address 1
Address 2
Address 3
(d)
Four Common Instruction Formats
(a) Zero - address instruction
(b) One - address Instruction
(c) Two - address Instruction
(d) Three - address instruction

Three Address Instruction

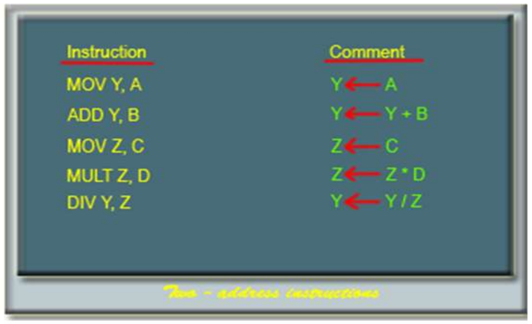


Instruction	Comment
ADD Y, A, B	Y ⟵ A + B
MULT Z, C, D	Z ⟵ C * D
DIV Y, Y, Z	Y ⟵ Y / Z

Three - address instructions

Three address instructions

Two Address Instruction

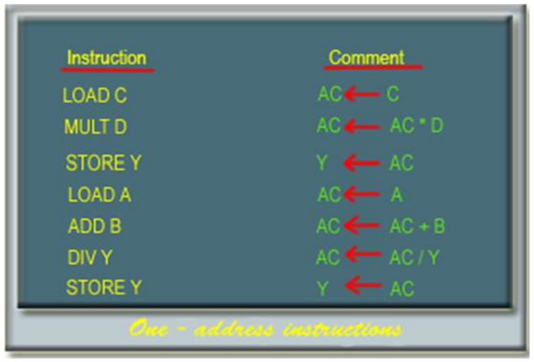


Instruction	Comment
MOV Y, A	Y ← A
ADD Y, B	Y ← Y + B
MOV Z, C	Z ← C
MULT Z, D	Z ← Z * D
DIV Y, Z	Y ← Y / Z

Two - address instructions

Two address instructions

One Address Instruction



Instruction	Comment
LOAD C	AC ← C
MULT D	AC ← AC * D
STORE Y	Y ← AC
LOAD A	AC ← A
ADD B	AC ← AC + B
DIV Y	AC ← AC / Y
STORE Y	Y ← AC

One - address instructions

One address instructions

Zero Address Instruction

- The location of the operands are defined implicitly
- For implicit reference, a processor register is used and it is termed as accumulator(AC).
- E.g. CMA //complements the content of accumulator
- i.e. $AC \longleftarrow AC$

➢Instruction Format Design Issues:

- An instruction consists of **an opcode and one or more operands**, implicitly or explicitly.
- Each explicit operand is referenced using one of **the addressing mode** that is available for that machine.
- **An instruction format is used to define the layout of the bits allocated to these elements of instructions.**
- Some of issues effecting instruction design are:
  1. **Instruction Length**
  2. **Allocation of bits for different fields in an instruction**
  3. **Variable length instruction**

1. Instruction Length

- A longer instruction means take more time in fetching an instruction.
- For e.g. an instruction of length 32 bit on a machine with word size of 16 bit will need two memory fetch to bring the instruction.
- Programmer desires:
  - **More opcode and operands** in a instruction as it will reduce the program length.
  - **More addressing mode** for greater flexibility in accessing various types of data.

Factors for deciding the instruction length:

A. Memory Size

- More bits are required in address field to access larger memory range.

B. Memory Organization

- If the system supports virtual memory then memory range is larger than the physical memory. Hence required the more number of addressing bits.

C. Bus Structure

- The instruction length should be equal to data bus length or multiple of it.

D. Processor Speed

- The data transfer rate from the memory should be equal to the processor speed.

2. Allocation of Bits

- More opcodes obviously mean more bits in the opcode field.
- Factors which are considered for selection of addressing bits are:

A. Number of Addressing modes:

- More addressing modes, more bits will be needed.

B. Number of Operands:

- More operands – more number of bits needed

C. Register versus memory:

- If more and more registers can be used for operand reference then the fewer bits are needed
- As number of register are far less than memory size.

2. Allocation of Bits

D. Number of Register Sets:

- Assume that A machine has 16 general purpose registers, a register address require 4 bits.
- However if these 16 registers are divided into two groups, then one of the 8 register of a group will need 3 bits for register addressing.

E. Address Range:

- 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.

F. 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.

3. Variable length Instruction

- Instead of looking for fixed length instruction format, designer may choose to provide a variety of instructions formats of different lengths.
- Addressing can be more flexible, with various combinations of register and memory references plus addressing modes.
- **Disadvantage:** an increase in the complexity of the CPU.

Concept of Program Execution

- The instructions constituting a program to be executed by a computer are loaded in sequential locations in its main memory.
- Processor fetches one instruction at a time and perform the operation specified.
- Instructions are fetched from successive memory locations until a branch or a jump instruction is encountered.
- Processor keeps track of the address of the memory location containing the next instruction to be fetched using Program Counter (PC).
- Instruction Register (IR)

Executing an Instruction

1. Fetch the contents of the memory location pointed to by the PC. The contents of this location are loaded into the IR (fetch phase).

   $IR \leftarrow [[PC]]$
2. Assuming that the memory is byte addressable, increment the contents of the PC by 4 (fetch phase).

   $PC \leftarrow [PC] + 4$
3. Carry out the actions specified by the instruction in the IR (execution phase).

Addressing Modes

- The term addressing mode refers to the mechanism employed for **specifying operands**.
- An operand can be specified as **part of the instruction or reference of the memory locations** can be given.
- An operand could also be **an address of CPU register**.
- The most common addressing techniques are:
  - **Immediate**
  - **Direct**
  - **Indirect**
  - **Register**
  - **Register Indirect**
  - **Displacement**
  - **Stack**

Addressing Modes

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

A=contents of an address field in the instruction that refers to a memory

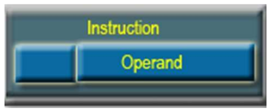
R=contents of an address field in the instruction that refers to a register

EA=actual (effective) address of the location containing the referenced operand

(X)=contents of memory location X or register X

1. Immediate Addressing:

- The operand is actually present in the instruction
- OPERAND = A

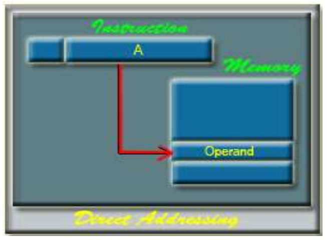


- This mode can be used to define and use constants or set initial values of variables.
- The advantage of immediate addressing is that no memory reference other than the instruction fetch is required to obtain the operand.
- e.g. **MOVE R0,300** ← Immediate addressing

2. Direct Addressing

- The address field contains the effective address of the operand: EA= A
- It requires only one memory reference and no special calculation.
- Here, 'A' indicates the memory address field for the operand.
- e.g. MOVE R1, 1001

Direct addressing

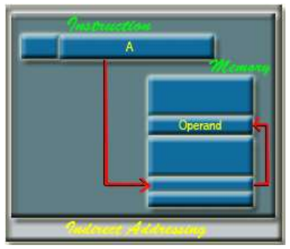


3. Indirect Addressing

- The effective address of the operand is stored in the memory and the instruction contains **the address of the memory containing the address of the data**.
- This is know as indirect addressing:

  EA = (A)
- Here 'A' indicates the memory address field of the required Operands.
- E.g. MOVE R0,(1000)

Indirect addressing



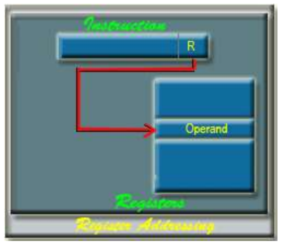
4. Register Addressing

- The instruction specifies the address of the register containing the operand.
- The instruction contains the name of the a CPU register.

  EA=R ← indicates a register where the operand is present.

- E.g. MOVE R1, 1010

Register addressing

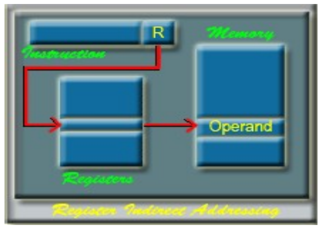


5. Register Indirect Addressing

- The effective address of the operand is stored in a register and instruction contains **the address of the register containing the address of the data.**
- EA = (R)
- Here 'R' indicates the memory address field of the required Operands.
- E.g. MOVE R0,(R1)

Register addressing

Register indirect addressing



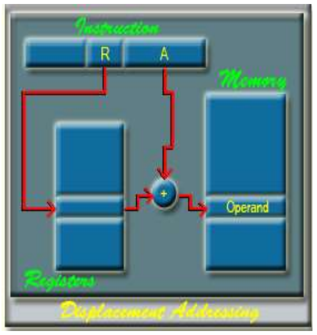
6. Displacement Addressing

- A combination of both direct addressing and register indirect addressing modes.

  EA = A + (R)
- The value contained in one address field (value = A) is used directly. The other address field refers to a register whose contents are added to A to produce the effective address.

6. Displacement Addressing

- Three of the most common use of displacement addressing are:
- ➔**Relative addressing**
- ➔**Base-register addressing**
- ➔**Indexing**



7. Relative addressing

- For relative addressing, the implicitly referenced register is the program counter (PC).
- 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.
- e.g. 1001 JC X1

```
1001        JC  X1


1050 X1:   ADD  R1,5
```

- X1=address of the target instruction-address of the current instruction

  =1050-1001=49

8. Base Register Addressing

- The **base register(reference register) contains a memory address**, and the **address field contains a displacement** from that base address specified by the base register.

  EA=A+(B)

9. Indexing or Indexed Addressing

- Used to access **elements an array** which are stored in consecutive location of memory.

  EA = A+(R)
- Address field **A gives main memory address** and **R contains positive displacement** with respect to base address.
- The displacement can be specified either **directly** in the instruction or through another **registers**.
- E.g. MOVE R1, (BR+5)   MOVE R0,(BR+R1)

  Starting address   Offset(index)   Starting address   Offset(index)

10. Auto Indexing

- 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.**
- Two types of auto-indexing
  1. **auto-incrementing**
  2. **auto-decrementing.**

a. Auto Increment Mode

- If register R contains the address of the operand
- After accessing the operand, the contents of register R is incremented to point to the next item in the list.
- Auto-indexing using increment can be depicted as follows:
  EA = A+ (R) or EA = (R)+
  (R) = (R) + 1
- E.g. MOVE R1,1010 /*starting Memory location 1010 is stored in R1*/
- ADD AC,(R1)+ /*contents of 1010 ML are added to AC and the contents of R1 is incremented by 1*/

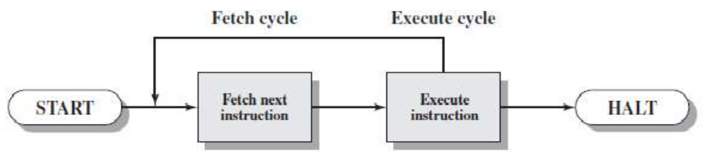
b. Auto Decrement Mode

- The contents of register specified in the instruction are decremented and these contents are then used as the effective address of the operand.
- Auto-indexing using decrement can be depicted as follows:
  EA = A– (R) or EA = –(R)
  (R) = (R) – 1
- The contents of the register are to be decremented before used as the effective address.
- E.g. ADD R1,-(R2)

11. Stack Addressing

- A stack is a **linear array or list of locations**.
- Sometimes referred to as a **pushdown list** or **last-in-first-out queue**.
- 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.
- E.g. **PUSH and POP**

Basic Instruction Cycle



•**Fetch cycle** basically involves read the next instruction from the memory into the CPU and along with that update the contents of the program counter.

• In the **execution phase**, it interprets the opcode and perform the indicated operation.

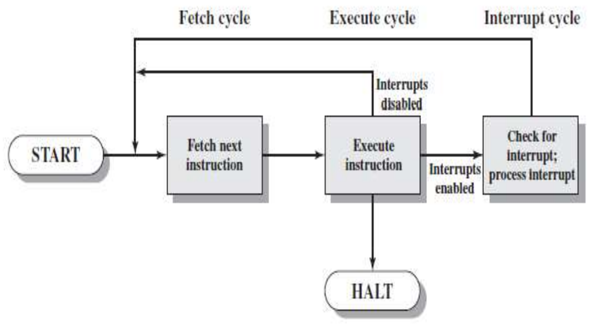
• The instruction fetch and execution phase together known as **instruction cycle**.

Basic Instruction Cycle with Interrupt

An instruction cycle includes the following sub cycles:

1. **Fetch:** Read the next instruction from memory into the processor.
2. **Execute:** Interpret the opcode and perform the indicated operation.
3. **Interrupt:** If interrupts are enabled and an interrupt has occurred, save the current process state and service the interrupt.

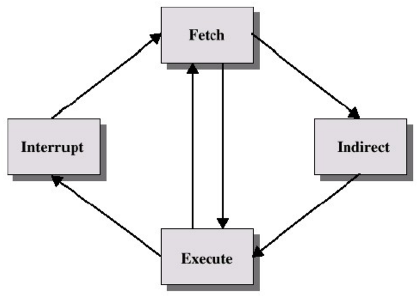
Basic Instruction Cycle with Interrupt



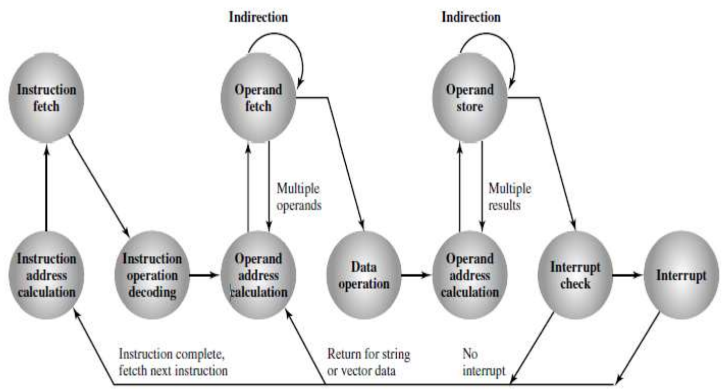
The Indirect Cycle

- 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.
- For fetching the indirect addresses as one more instructions subcycle are required.
- 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.

The Indirect Cycle



Instruction Cycle State Diagram


Indirection
Indirection
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetch next instruction
Return for string or vector data
No interrupt

- **Instruction address calculation (iac):** Determine the address of the next instruction to be executed. Usually, this involves adding a fixed number to the address of the previous instruction.
- **Instruction fetch (if):** Read instruction from its memory location into the processor.
- **Instruction operation decoding (iod):** Analyze instruction to determine type of operation to be performed and operand(s) to be used.
- **Operand address calculation (oac):** If the operation involves reference to an operand in memory or available via I/O, then determine the address of the operand.
- **Operand fetch (of):** Fetch the operand from memory or read it in from I/O.
- **Data operation (do):** Perform the operation indicated in the instruction.
- **Operand store (os):** Write the result into memory or out to I/O.

Instruction Interpretation and Sequencing

- Every processor has some basic type of instructions like data transfer instruction, arithmetic and logical instruction, branch instruction and so on.
- To perform a particular task on the computer it is programmers job **to select and write appropriate instructions one after the other**. This job of programmer is known as **instruction sequencing**.
- Two types:
  1. Straight line sequencing
  2. Branch instructon

1.Straight line sequencing

- Processor executes a program with the help of **Program Counter(PC)** which holds the address of the next instruction to be executed.
- To begin execution of a program, **the address of the its first instruction is placed into the PC**

memory address specified by the PC and executes instruction, one at a time.

- At the same time **the content of PC is incremented** so as to point to the address of next instruction.
- This is called as **straight line sequencing.**

2. Branch Instruction

- After executing decision making instruction, processor have to follow one of the two program sequence.
- Branch instruction transfer the program control from one straight line sequence to another straight line sequence instruction.
- In branch instruction, the new address called **target address or branch target** is loaded into PC and instruction is fetched from the new address.

Thank You