2 Machine Instruction & Measuring CPU Performance

Stack CPU:

- In this organisation, ALU operations are performed only on stack data, which means both of the operands are always required to be in the stack.
  After the data processing, the result is also placed in the stack.
- In the stack, insertion and deletion operation takes place at the same end called as top of the stack (TOS), so TOS becomes a default location. Therefore, the address is not required in the instruction.
- Compatible instruction format is:

Opcode – – – 0 address instruction format

Example:

(A * B) + C ; variables are in the memory.

Code:

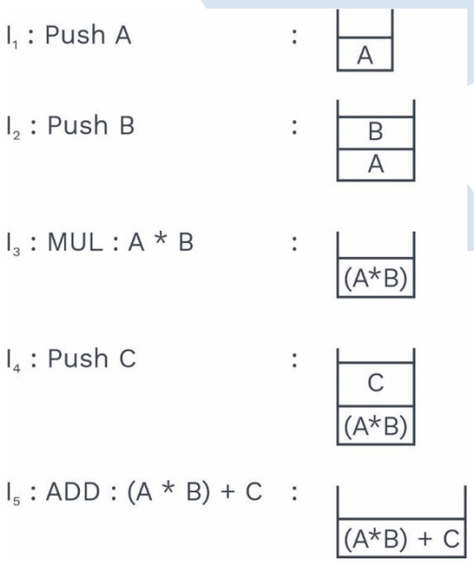
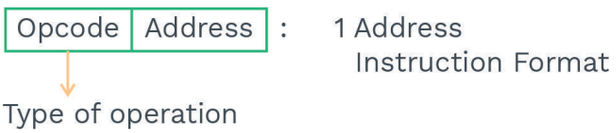


Fig. 2.1 Operations in Stack

Accumulator CPU:

- In this organisation, for an ALU operation, the first operand is always required in the accumulator and the second operand is present either in the register or in the memory.
  After the data processing, the result is placed into the accumulator.
- In the CPU design, only 1 accumulator is present so it becomes a default location. Its address is not required in the instruction.

- Compatible instruction format is



Example: (A * B) + C, variables are in the memory

Code:

I_1: Load A ; Accumulator ← M[A]

I_2: MUL B ; Accumulator ← Accumulator * M[B]

I_3: Add C ; Accumulator ← Accumulator + M[C]

General register CPU:

- Based on the number of registers possible in the CPU, architecture is of two types:
  - **i)** Register to memory reference CPU (CPU with less registers)
  - **ii)** Register to Register reference CPU (CPU with more registers)

Register to memory reference CPU:

- In this organisation, ALU first operand is always required in the register and the second operand is present either in the register or in the memory. After the data processing, the result will be placed into source$_1$ (first operand) location.
- Compatible instruction format is:

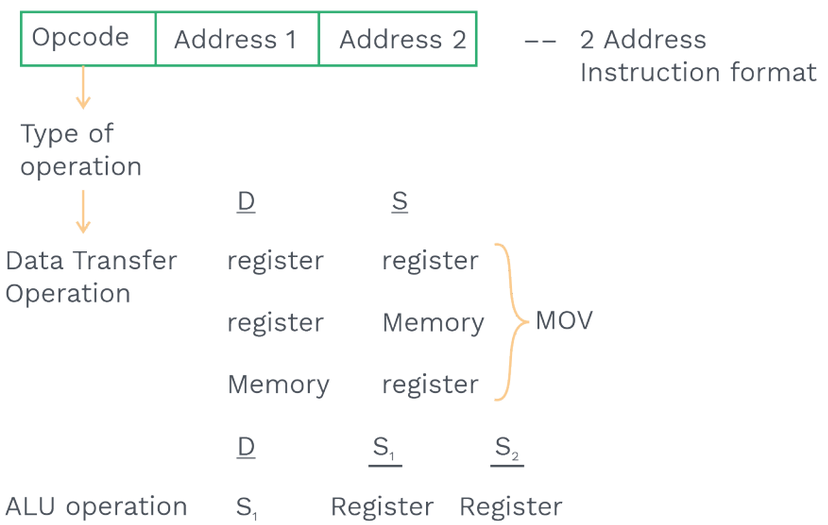


Fig. 2.2 2-Address Instruction Format

D: Destination
S= source
S_1= source 1
S_2 = source 2

Example: (A * B) + C ; variables are in the memory.

Codes:

		Action
I_1 : MOV r_0 , A	:	$r_0 \longleftarrow$ M[A]
I_2 : MUL r_0 , B	:	$r_0 \longleftarrow r_0$ * M[B]
I_3 : ADD r_0 , C	:	$r_0 \longleftarrow r_0$ + M[C]

Register to register reference CPU:

- In this organisation, ALU operations are performed only on register data, so both the operands are always present in the register. After the data processing result is also placed in a register.
- Compatible instruction format is:

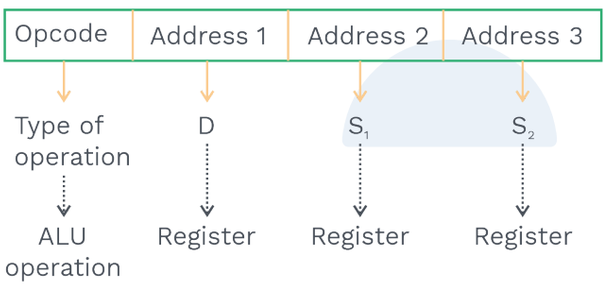


Fig. 2.3 3 Address Instruction Format

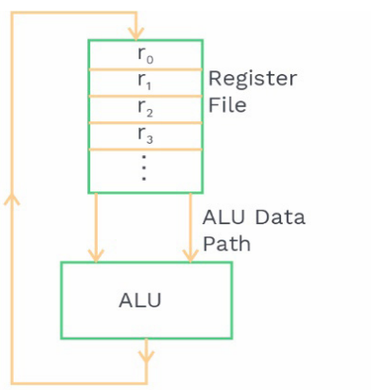


Fig. 2.4 Register-Register Reference CPU

Example: (A * B) + C: Variables are in the memory

Code:

I_1: Load r_0, A
I_2: Load r_1, B
I_3: MUL r_2, r_0, r_1
I_4: Load r_3, C
I_5: Add r_4, r_2, r_3

4 address instruction format:

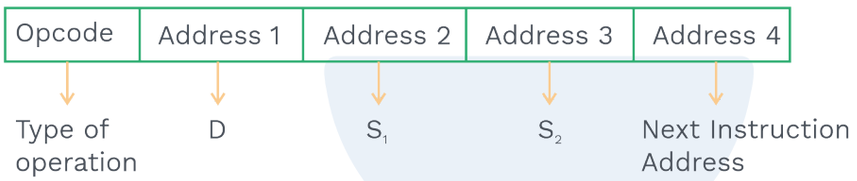


Fig. 2.5 4-Address Instruction Format

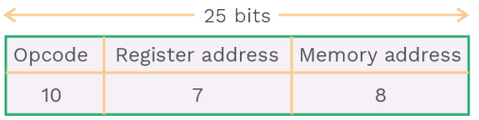
Note:
Program Counter (PC) is the mandatory register utilised by the CPU, which holds the address of the next instruction during the accomplishment of the current instruction. Therefore, this format (4 address instruction format) is not in use.

PRACTICE QUESTIONS

Q1 **A processor has 700 distinct instructions and 80 general purpose registers. A 25-bit instruction word has an opcode, one register operand and a memory operand. How many bits are reserved for the memory operand field?**
a) 5 **b) 10** **c) 8** **d) 6**
Sol: c)

Sol: Given, that we have 700 distinct instructions, so it requires $\lceil \log_2 700 \rceil$ bits = 10 bits.

- 80 general purpose registers requires $\lceil \log_2 80 \rceil$ bits = 7 bits.
- Size of the instruction = 25 bits



- Number of bits available for the memory operand field = 25 – (10+7)
  = 25 – 17
  = 8

Previous Years' Question

A processor has 40 distinct instructions and 24 general purpose registers. A 32–bit instruction word has an opcode, two register operands and an immediate operand. The number of bits available for the immediate operand field is _____ .

a) 16 bits
b) 17 bits
c) 18 bits
d) 19 bits

Sol: a) **(GATE-2016 (Set-2)**

ADDRESSING MODES

- Addressing mode shows the location of a required object in the computer.
- Object may be a data or instruction.
- Output of an addressing mode is the effective address (EA).
- EA is the actual address of an object,

  Object = [EA] content of

- Addressing mode can be implemented in the instruction in 2 ways:

1) Implicit: Here, opcode itself specifies the type of addressing mode used.
Instruction:

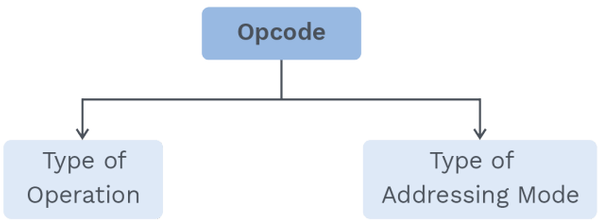


Table 2.1 Implicit Addressing Mode

2) Explicit: Mode field is used in the instruction to specify the type of addressing mode used.
Instruction:

- Symbols used for specifying the different addressing modes (AMs) in the user program are as follows:

Type of addressing mode	Symbol
1) Immediate AM	#
2) Register AM	Register name
3) Direct AM	[ ]
4) Indirect AM	@ , ( )
5) Indexed AM	Index register name
6) Auto indexed AM	+ , –

Table 2.2 Symbolic Representation of Addressing Modes

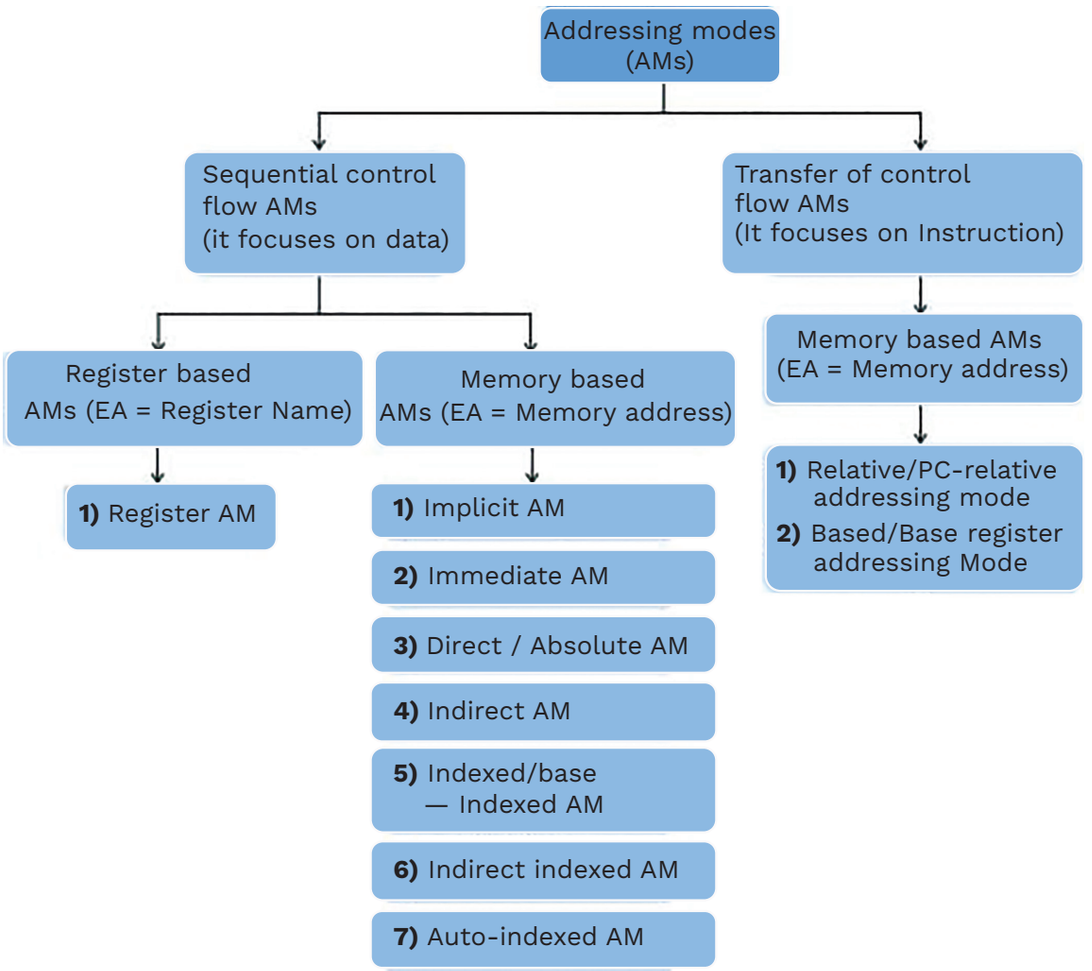


Table 2.3 Classification of Addressing Modes

Memory based:

Sequential control flow AM:

- These types of modes concentrate on the data location, so data transfer and data manipulation instructions are designed with this addressing modes.

  1) Implied addressing mode:

  In the implied addressing mode, the opcode itself specifies the data information along with the operation.

  Example: Set Carry Clear Carry

  STC CLC

  Carry = 1 Carry = 0

Example: ADD on Stack CPU

i) POP
ii) POP
iii) +
iv) Push

Note:
All zero address instructions are implied instructions, but all implied instructions are not zero address instructions.

2) Immediate addressing mode:
This mode is used for accessing the constants. In this mode, data is present in the address field of an instruction.
Instruction:

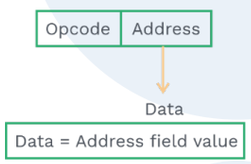


Fig. 2.6 Immediate Addressing Mode

Example:

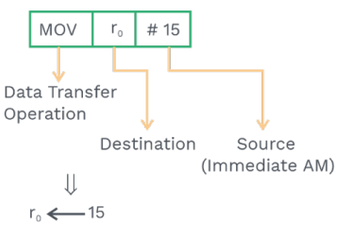


Fig. 2.7 Valid Data Transfer using Immediate Addressing Mode

Example:

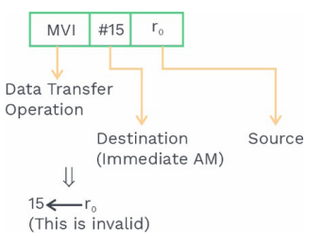


Fig. 2.8 Invalid Data Transfer using Immediate Addressing Mode

Limitation:

Range of constants is restricted by the size of the address–field, i.e., if k–bit address field, then

Range = {0 to (2^k – 1)} unsigned constants.

Range = {– (2^{k-1}) to + (2^{k-1} – 1)} signed constants.

3) Direct addressing mode:

- This mode is best utilised to refer to the static variables.
- In this mode, the address field of the instruction contains the effective address of data which is present in the memory.

Instruction:

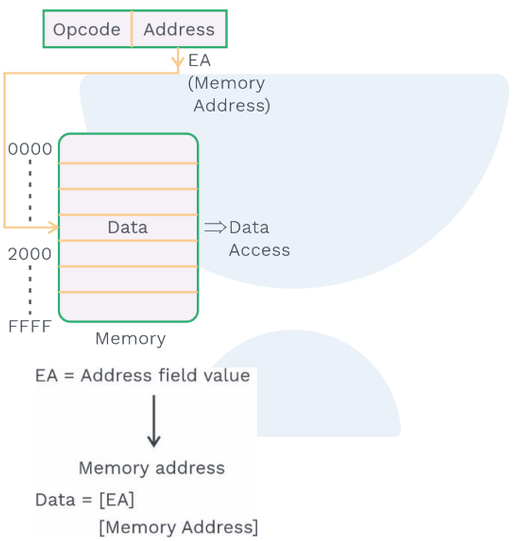


Fig. 2.9 Direct Addressing Mode

- Here, 1 memory reference is required to access the data.

Example: Consider 3 word Instruction

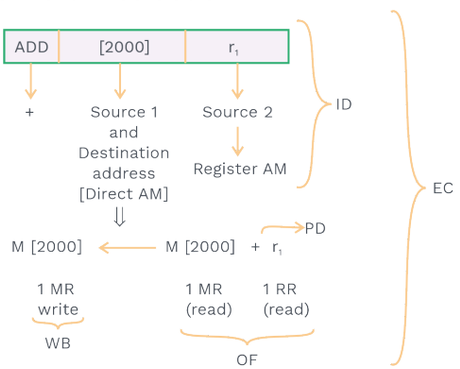


Fig. 2.10 Data Transfer using Direct Addressing Mode

Instruction cycle:

Fetch Cycle (FC)	Execution Cycle (EC)				
IF	ID	OF		PD	WB
1 MR	2 MR	S_1	S_2	1ALU	1MR
		1MR	1RR		

IF = Instruction Fetch
ID = Instruction Decode
OF = Operand Fetch
PD = Process Data (Execute)
WB = Write Back

S_1 = Source 1 register
S_2 = Source 2 register
D = Destination register
MR = Memory reference

Fig. 2.11 Instruction Cycle

4) Indirect addressing mode:

The idea of pointers is best realised using indirect addressing mode.

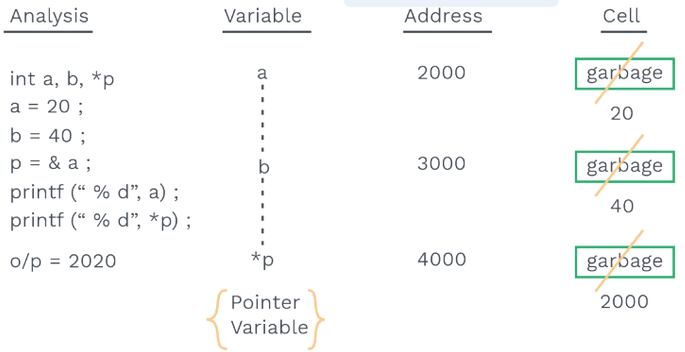


Fig. 2.12 Pointers using Indirect Addressing Mode

- The default storage class is auto, so the default value is garbage.

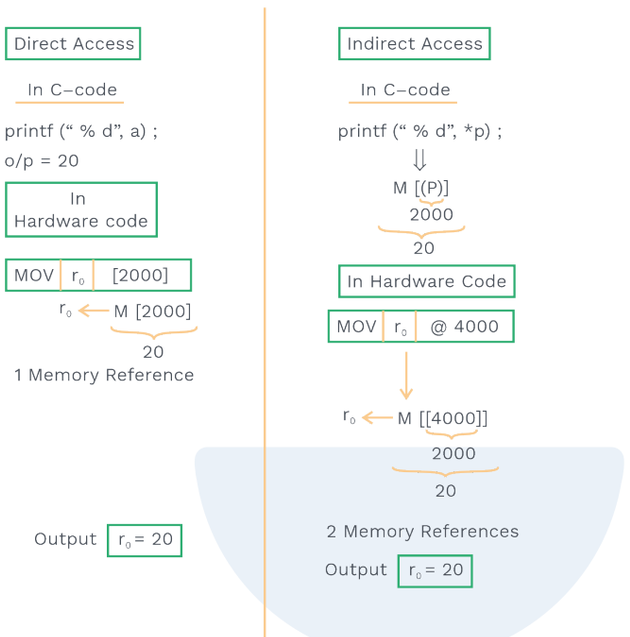


Fig. 2.13 Program Analysis using Direct and Indirect Addressing Modes

4) a) Memory indirect addressing mode:

In this mode, the effective address [EA] is present in the memory; the corresponding memory address will be maintained in the address field of instruction as an address of the Effective Address.

Instruction:

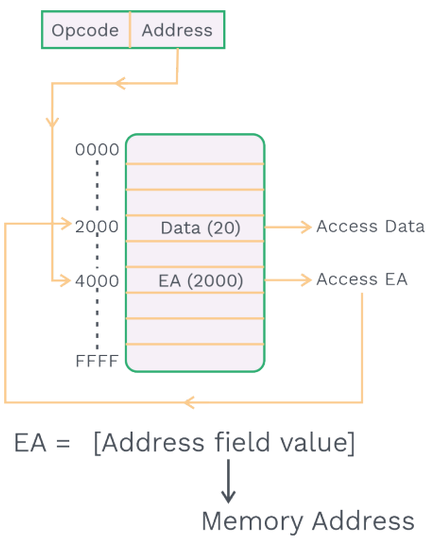


Fig. 2.14 Memory Indirect Addressing Mode

Data = [EA]
[[Memory Address]]
* 1 memory reference to access the EA.
* 1 memory reference to access the data.

Example: Consider the 4 word Instruction.

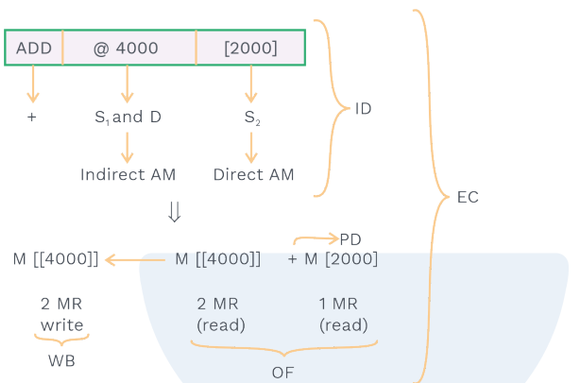


Fig. 2.15 4-Word ALU Instruction

Instruction cycle:

Fetch Cycle (FC)		Execution Cycle (EC)			
IF	ID	OF		PD	WB
1 MR	3 MR	S_1	S_2	1 ALU	D
		2MR	1MR		2 MR

4) b) Register indirect AM:
The address bits of the instruction format store the name of the register containing the effective address of the operand in the memory.
Instruction:

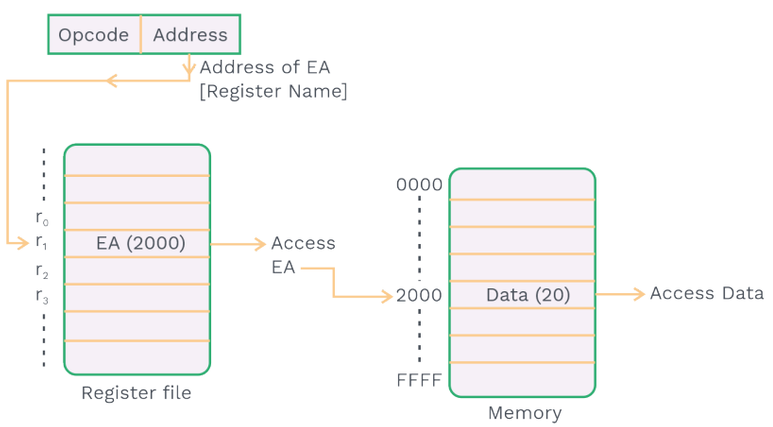


Fig. 2.16 Register Indirect Addressing Mode

Data = [EA]
[Register Name]
* 1 register reference required to access the Effective Address.
* 1 memory reference to access the data.
Example: Consider the 4 word instruction.

Instruction:

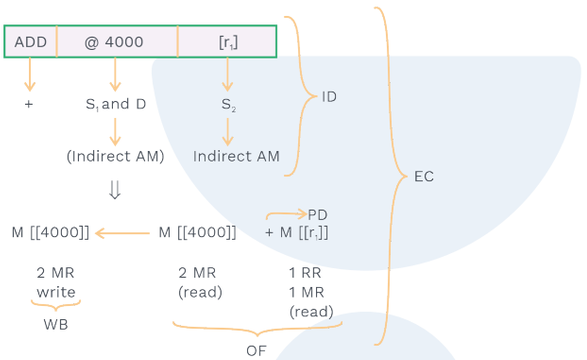


Fig. 2.17 4-Word ALU Instruction using Memory and Register Indirect Addressing Modes

Instruction cycle:

Fetch Cycle (FC)	Execution Cycle (EC)				
IF	ID	OF		PD	WB
1 MR	3 MR	S_1	S_2	1ALU	D
		2MR	1RR 1MR		2 MR

IF = Instruction Fetch
ID = Instruction Decode
OF = Operand Fetch
PD = Process Data
WB = Write Back

S_1 = Source 1 register
S_2 = Source 2 register
D = Destination register
MR = Memory reference
RR = Register reference

Fig. 2.18 Description of Instruction Cycle

5) Indexed addressing mode:

- This mode is used to access the arrays.

Analysis:

char a [10] ;

Storage:

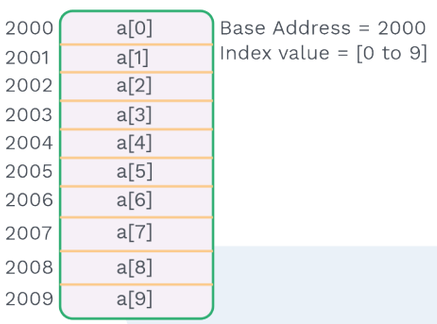


Fig. 2.19 Indexed Addressing Mode

Example: a[4]

I_1: MOV r_0, # 4 Assume r_0 as an index register

I_2:

MOV	r_1	2000 (r_0)

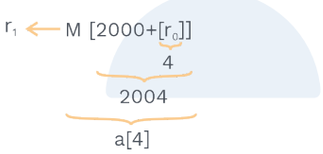


Fig. 2.20 Random Accessing of Array Elements I

r_1 = a[4]

Example: a[10]

I_1: MOV r_0, # 10

I_2:

MOV	r_1	2000 (r_0)

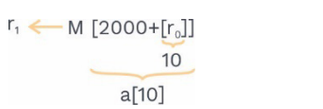


Fig. 2.21 Random Accessing of Array Elements II

r_1 = a[10]

- The above 2 examples are in the category of randomly accessing the array.

i) Pre increment:

I_1: MOV r_0, # 1 FFF

I_2: | MOV | r_1 | + (r_0) |

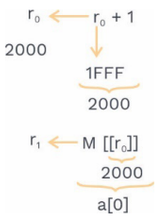


Fig. 2.22 Pre-Increment Indexed Addressing Mode

ii) Post increment:

I_1: MOV r_0, # 2000

I_2: | MOV | r_1 | (r_0) + |

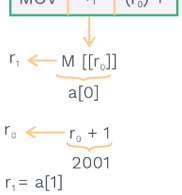


Fig. 2.23 Post-Increment Indexed Addressing Mode

iii) Pre decrement:

I_1: MOV r_0, # 200 A

I_2: | MOV | r_1 | – (r_0) |

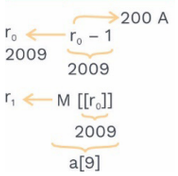


Fig. 2.24 Pre-Decrement Indexed Addressing Mode

r_1 = a[9]

iv) Post decrement:

I_1: MOV r_0, # 2009

I_2: | MOV | r_1 | (r_0) – |

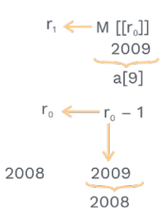


Fig. 2.25 Post-Increment Indexed Addressing Mode

r_1 = a[9]

- Above all are examples of linearly accessing the memory.
- Indexed AM is used to access the random array elements from the memory.
- Here 2 parameters are required.

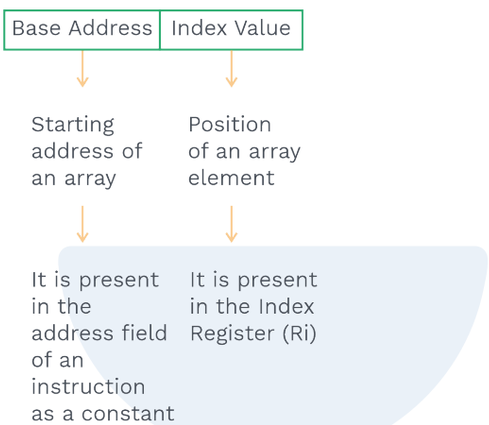


Fig. 2.26 Array Accessing Parameters

- The Effective address (EA) is obtained by adding the constant to the contents of the (Ri).

EA = Address field value + [Ri]

Data = [EA]

- Here 1 register reference, 1 memory reference and 1 ALU operation are required.

Instruction design:

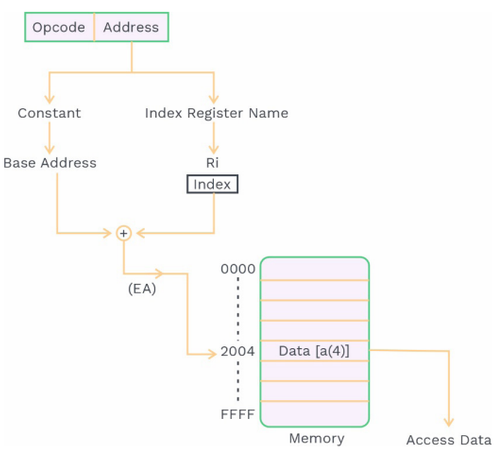
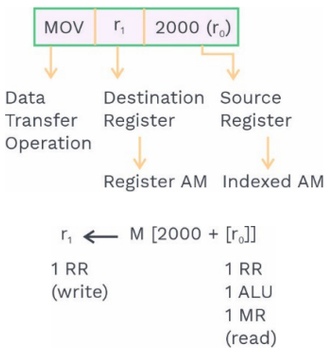


Fig. 2.27 Instruction Design

Example: Consider 3 word instruction:



Instruction fetch = 1 MR
Instruction decode = 2 MR
Operand fetch = S = 1 RR
1 ALU
1 MR
Write Back to D = 1 RR

S = Source Register
D = Destination Register

Fig. 2.28 3-Word Data Transfer Instruction using Register and Indexed Addressing Modes

6) Indirect indexed AM:

- EA is present in the memory in this addressing mode.
- In this mode, the address of the EA is calculated by adding the index value to base address.
  EA = [Address field value + (Ri)]
  Data = [EA]
- 1 RR is required to get the index
  1 ALU is required to calculate the address of EA
  1 MR is required to get the EA
  1 MR is required to access the data

Instruction design:

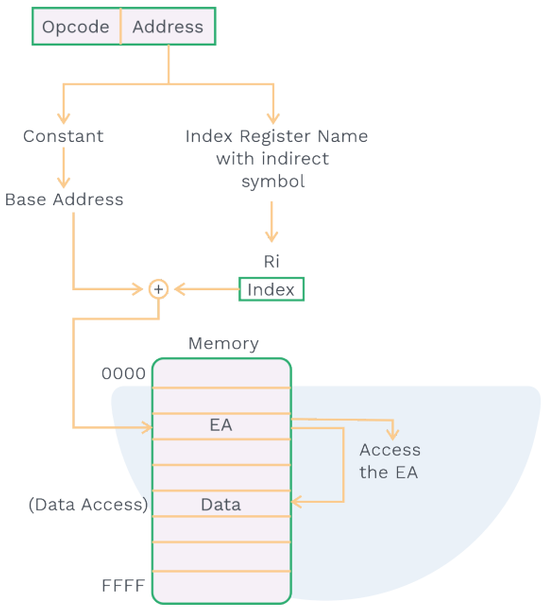


Fig. 2.29 Instruction Design using Indirect Indexed Addressing Mode

- This mode takes more time to access the data because 2 MR are required to access the EA and data, respectively.

7) Auto indexed AM:

- It is used to access the linear array of elements from memory.

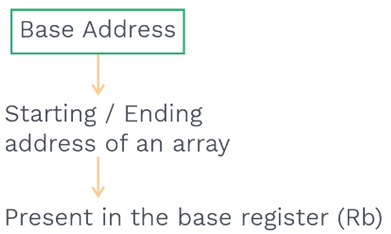


Fig. 2.30 Linear Accessing of Array Elements

- Effective address is calculated either by incrementing or decrementing the contents of Rb with some given step size.
- Step size is fixed constant, depending on the word length of the CPU.
  EA = [Rb] (+ / –) step size
  Data = [EA]

1) 1 RR is required to get the base address (Rb).
2) 1 ALU operation is required to calculate the EA.
3) 1 MR is required to access the data.

Instruction design:

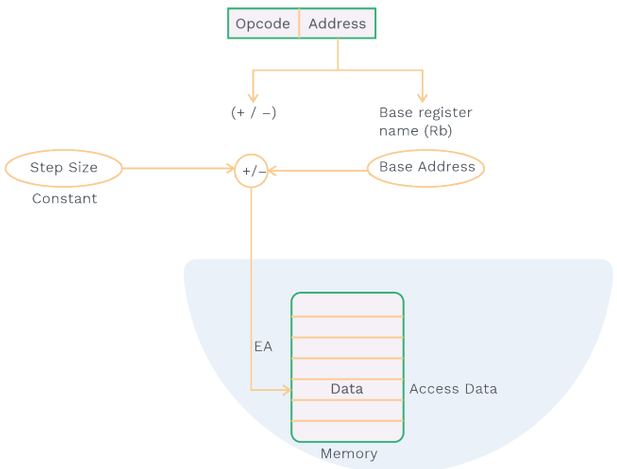


Fig. 2.31 Instruction Design using Pre-Increment/ Decrement Auto Indexed Addressing Mode

- This is pre-increment/pre decrement Auto indexed AM.

Instruction design:

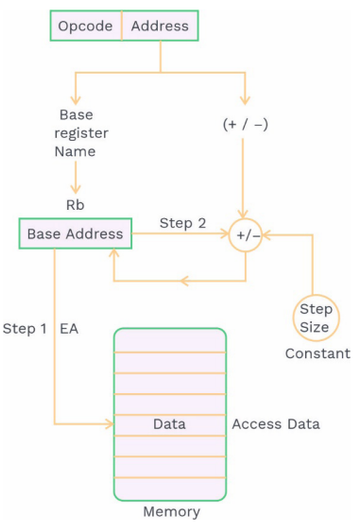


Fig. 2.32 Instruction Design using Post-Increment/ Decrement Auto Indexed Addressing Mode

- This is post-increment/post-decrement auto indexed AM.

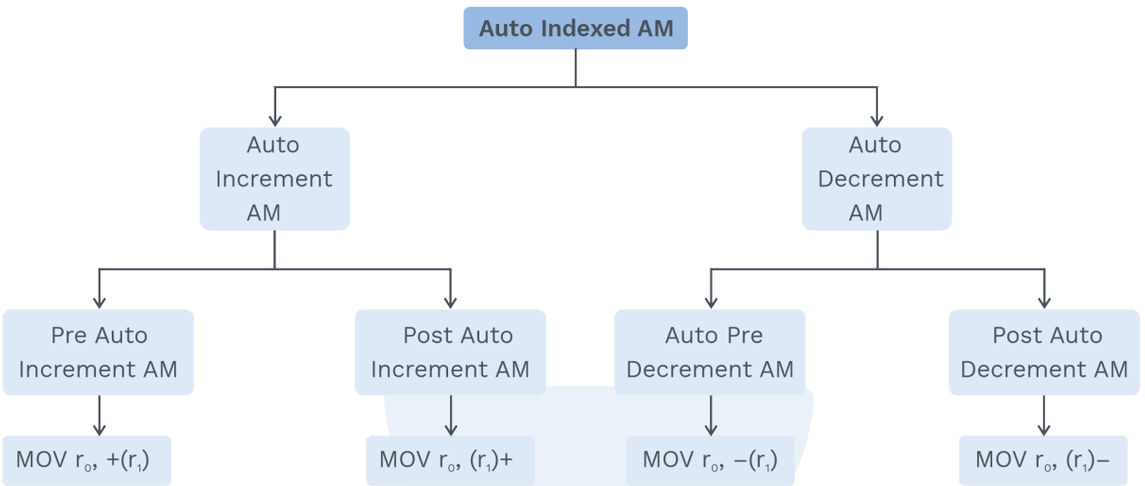


Table 2.4 Classification of Auto Indexed Addressing Modes

- Here "r_1" is the base register.
- By default, auto increment AM uses post auto increment AM.
- And by default, auto decrement AM uses pre auto decrement AM.

Register based:

Register addressing mode:

The local variables are referred to on utilising this mode. In this mode, data is present in the register, and the corresponding register name will be maintained in the address field of instruction as effective address.

Instruction:

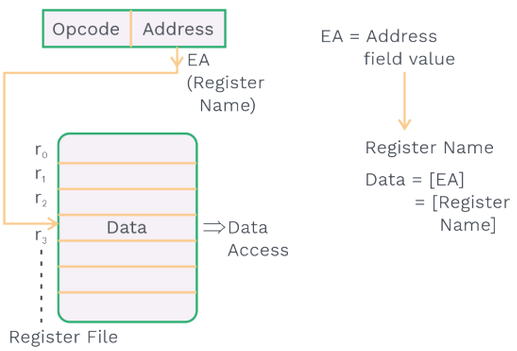


Fig. 2.33 Register Addressing Mode

- Here, 1 register reference is there to access the data.

Example: Consider 2 word instruction

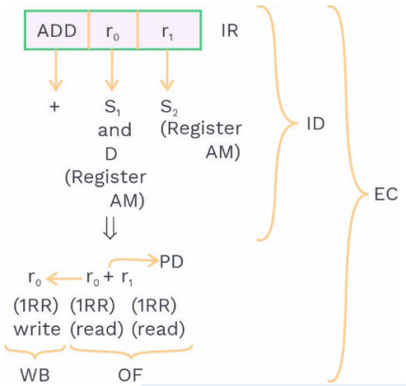


Fig. 2.34 2-Word ALU Instruction using Register Addressing Mode

Instruction cycle:

Fetch Cycle (FC)	Execution Cycle (EC)				
IF	ID	OF		PD	WB
1 MR	1 MR	S_1	S_2	1ALU	1RR
		1RR	1RR		

IF = Instruction Fetch
ID = Instruction Decode
OF = Operand Fetch
PD = Process Data (Execute)
WB = Write Back

S_1 = Source 1 register
S_2 = Source 2 register
D = Destination register
MR = Memory reference

Fig. 2.35 Description of Instruction Cycle

Transfer of control flow addressing modes:

- These addressing modes focus on the location of the next instruction.
- When the program contains control structures, then during the program execution, control will be transferred from the current location to the target location.

Code:

```
        1000 : I1
        1001 : I2
        1002 : I3   (goto l1)
        1003 : I4
          :
          :
[l1]    2000 : Branch Instruction 1 (BI1)
        2001 : Branch Instruction 2 (BI2)
```

Output sequence: $I_1 - I_2 - I_3 - BI_1 - BI_2 - - -$

- To implement the control structures in the base hardware, transfer of control (TOC) instructions are used, named JUMP (JMP) / branch/ skip.
- Transfer of control instructions are designed with the following addressing modes, used to calculate the effective address or target address or branch address.

Types of transfer of control addressing modes (AMs):

1) PC relative AM

2) Base register AM

- For analysing the above AMs, we will consider the 8086-memory organisation as a reference model.
- 8086 supports 1 MB physical memory, organized into 16 logical segments with each segment size of 64 KB.

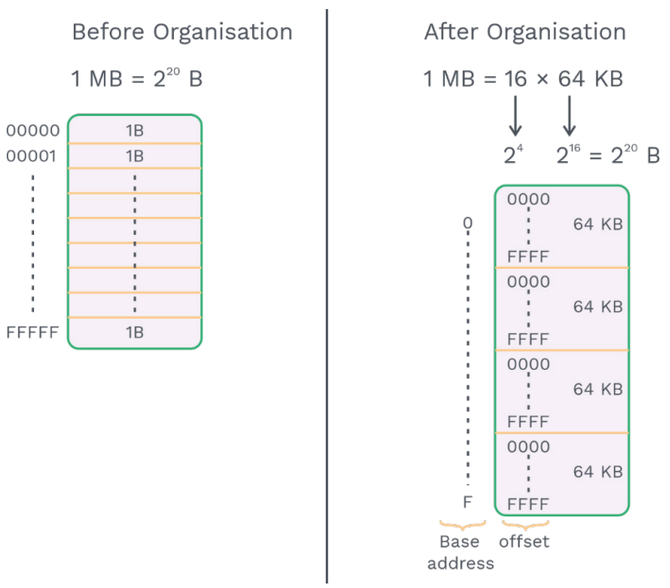


Fig. 2.36 Memory Organization of 8086 Microprocessor Chip

Intra-segment TOC:

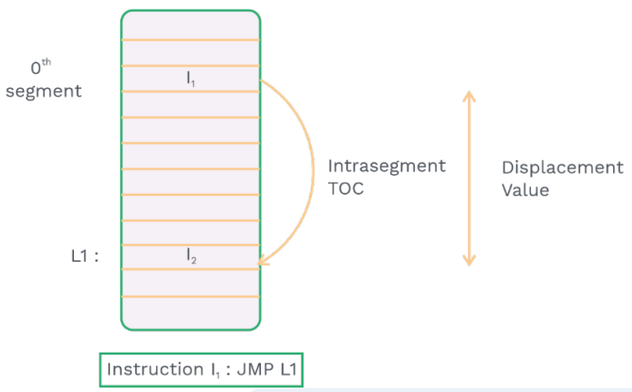


Fig. 2.37 Intra-segment TOC

Inter-segment TOC:

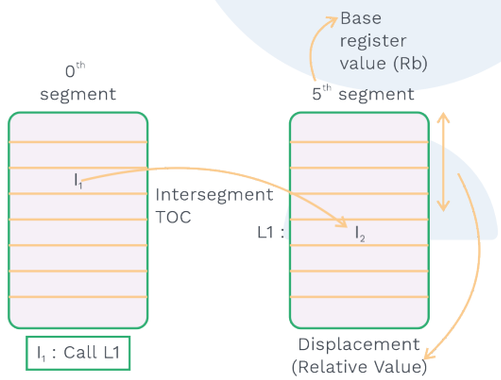


Fig. 2.38 Inter-segment TOC

1) PC relative AM:

- The targeted instruction, when available in the same segment, control is relocated within the segment while machine code execution. This procedure is termed Intra-segment TOC.
- This Addressing Mode (AM) is used to implement the above operation.
- Effective Address is obtained by adding the relative value to PC.
- Relative value means the distance between the current location and target location. It is a signed constant present in the address field of instruction.

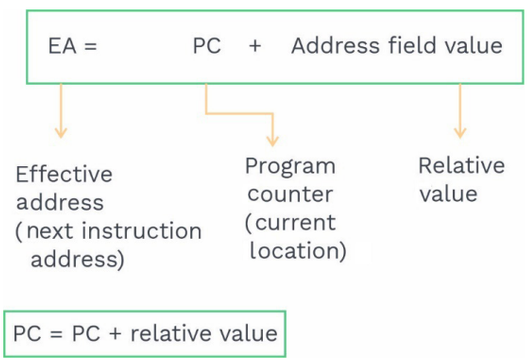


Fig. 2.39 PC Relative Addressing Mode

2) Base register AM:

- The targeted instruction, when available in a distinct segment, control is relocated from one segment to another segment while machine code execution. This procedure is termed Inter-segment TOC.
- Base Register AM is best suitable for the implementation of the above concept.
- Effective Address (EA) is obtained by adding the relative value to the content of the base register.

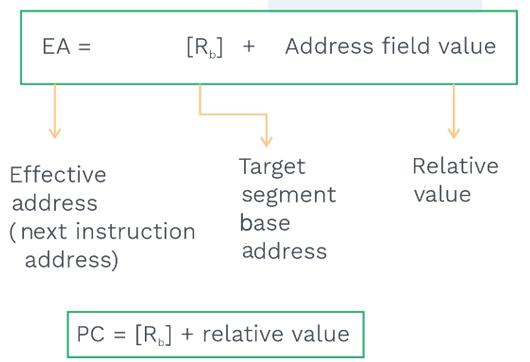


Fig. 2.40 Base Register Addressing Mode

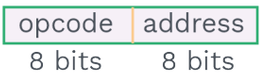
Note:

i) Runtime program relocation is best supported by both PC relative and base register addressing modes.

ii) Position independent codes framing is best supported by a base register addressing mode.

Consider the following data for the examples (2 to 8):

- A 2 byte instruction is present in memory starting at $(550)_{10}$, and the instruction format is given below:

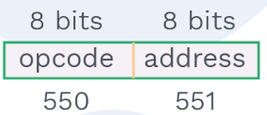


- Address field contains the value $(400)_{10}$
- M[400] = 600
- Register R1 contains the value $(320)_{10}$
- R1 has an address of 800.

PRACTICE QUESTIONS

Q2 **Calculate the effective address if immediate addressing mode is utilised.**
Sol: 551

Sol:



Value of address field = 400

- In immediate addressing mode, the value in the address field is used as data.
- 400 is used as data to opcode in immediate addressing mode.

400 is stored at address 551.
so, effective address = 551

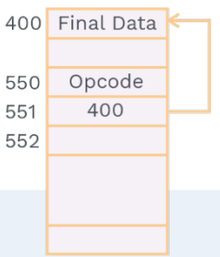
Q3 **Calculate the effective address if register addressing mode is used.**
Sol: 800

Sol: Register R1 contains the value of 320, and it is stored at address 800. So, effective address = 800

Note:
Here we are assuming only one register R1 is given.

Q4 **What will be the effective address using direct addressing mode?**
Sol: 400

Sol: In direct addressing mode, the value at the address field is considered the actual address of the data. So, here effective address = 400
Final data resides at address location 400.



Q5 **What is the effective address using register indirect addressing mode?**
Sol: 320

Sol: In register indirect addressing mode, whatever the data contained in register will be considered as the final address.
Here R1 contains the value of 320.

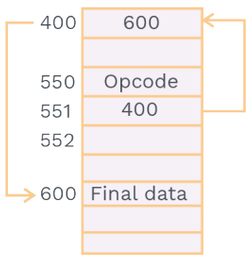
R1 | 320 |
800

so, effective address = 320

- At memory location 320, the final data resides.

Q6 **Calculate the effective address using memory indirect addressing mode.**
Sol: 600

Sol: This addressing mode works as a pointer
M[400] = 600
In memory location 600, the final data resides.



so the effective address = 600

Note:
The effective address is the final address where data resides.

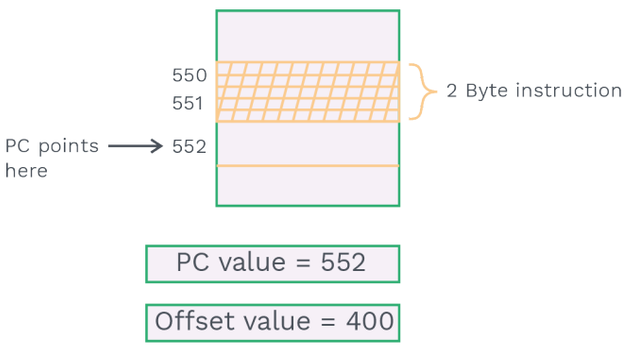
Q7 **Calculate the effective address using indexed–addressing mode (R1 =index register).**
Sol: 720

Sol: In indexed–addressing mode, the value in the address field will be used as the base address.

- Here address field value = 400
- Index value = Value in R1 = 320
- Effective address = Base register + Index value
  = 400 + 320 = 720

Q8 **Calculate the effective address using the PC–relative addressing mode.**
Sol: 952

Sol: In PC–relative addressing mode, the value in the address field will be used as an offset.



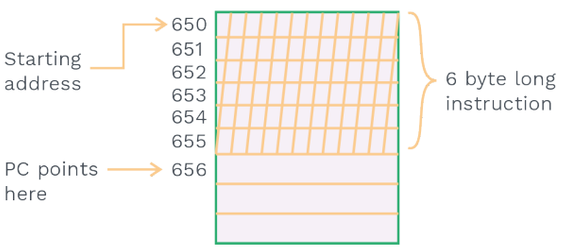
Effective address = PC value + Offset value
= 552 + 400
= 952

Note:
For PC–relative addressing, we are assuming that we have JUMP like instruction.

- **Consider the following data for examples (9 to 11):**
A 6 Byte long PC–relative jump instruction started at $(650)_{10}$ onwards, and (–88) is the signed displacement value which is present in the address field of the instruction.

Q9 **What is the branch address to which it will branch (or the effective address)?**
Sol: 568

Sol:



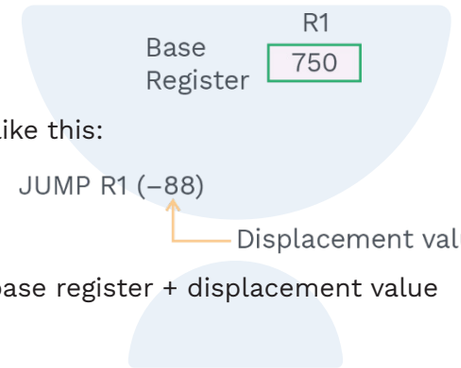
- Relative address is also known as displacement value. This field has a value of (–88).

PC value = 656

effective (Branch) address = PC value + Relative (displacement value)
= 656 + (−88)
= 568

Q10 **If base−register R1 contains $(750)_{10}$, what will be the branch address (or the effective address) if base−register addressing mode is used?**
Sol: 662

Sol: Suppose the base register is R1



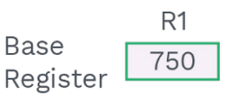
Instruction will look like this:

Effective address = base register + displacement value
= 750 + (−88)
= 662

Note:
The base register always contains the first address of the segment. The displacement field contains displacement within the segment.

Q11 **Calculate the branch address (the effective address) if based−indexed addressing mode is used, if index−value = 92.**
Sol: 754

Sol:

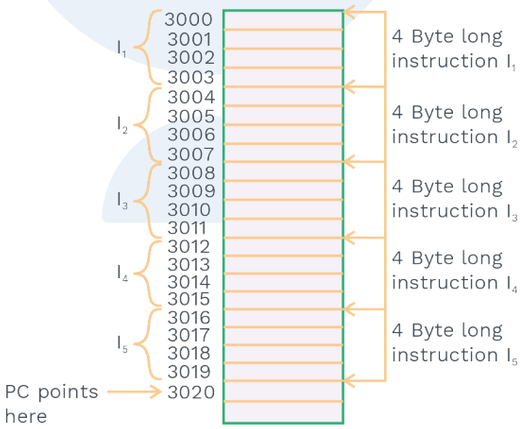


index value = 92
Displacement value = (−88)
Effective address = Base register value

+
Index–value
+
Displacement value
= 750 + 92 – 88
= 754

Q12 **Assume that there are 5 instructions in the loop body; starting memory locations are 3000, 3004, 3008, 3012 and 3016, respectively. Assume each instruction takes 4 bytes of memory and memory is byte addressable. Calculate the offset value needed to return to the loop (starting of the loop). Sol: 20**

Sol: Suppose five instructions are I_1, I_2, I_3, I_4 and I_5.



- PC value at the end of loop (or after instruction I_5) = 3020.
- To return to starting of the loop, the effective address should be equal to starting address.

Effective address = PC value + Offset value
Effective address = starting address of the loop
Effective address = 3000
3000 = PC value + Offset value
3000 = 3020 + Offset value
Offset value = –20

PC = Program Counter

Q13 **In a certain processor, a 4 Byte jump instruction is encountered at memory address $(3603)_{10}$; the jump instruction is in PC relative mode. The instruction is JMP(–13), where (–13) is a signed byte. Determine the branch target address. (Assume byte addressable memory).**

a) 3594 **b) 3590** **c) 3616** **d) 3620**

Sol: a)

Sol: The jump instruction is at address $(3603)_{10}$, and the instruction is 4 byte long.

Hence ; PC (Program Counter) points to 3603 + 4 = 3607

Effective (branch) address = PC value + Displacement (relative) value

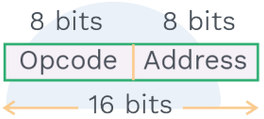
Here relative (displacement) value = –13

Effective address = 3607 + (–13)

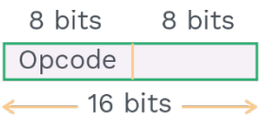
= 3594

Expand opcode–technique:

- Assume a microprocessor with fixed–length instructions (16 bits). This microprocessor has both 1–address instructions and 0–address instructions.

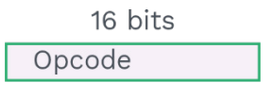


- This is a 1–address instruction format; here, opcode = 8 bits and address field = 8 bits. So, 1–address instruction perfectly fits inside fixed length instruction size.



0–address instruction format

- Here, opcode is 8 bits, but the issue is with the remaining 8 bits because the microprocessor has a fixed length instruction of size 16 bits. So, the question arises, what do we have to do with these 8 bits which are remaining.
- So, now we will use this method of expanding the opcode. In the remaining 8 bits, we have the scope of filling 00000000 to 11111111. So, instead of wasting these 8 bits, we will expand the opcode to use all 16 bits.

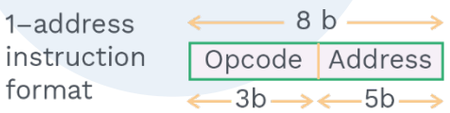


0–address instruction format

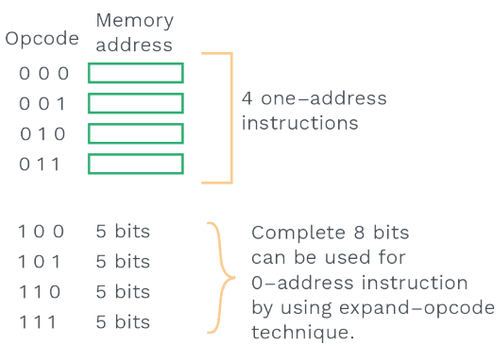
- In this way, the size of the opcode field is increased.

Q14 **A microprocessor supports both 0–address instructions and 1–address instructions. The memory size is 32 words. This microprocessor has fixed–sized instructions of 8 bits. Suppose there are 4 one–address instructions. Calculate the number of possible zero–address instructions.**
Sol: 128

Sol: Given fixed instruction size of 8 bits.



There are 32 words in memory, hence address field of 1–address instruction will take $\lceil \log_2 32 \rceil = 5$ bits. So the remaining 3 bits are there for the opcode.
Total number of 1–address instructions possible = $2^3 = 8$.
But we are using only 4 one–address instructions, the remaining 8 – 4 = 4 can be used for 0–address instructions.



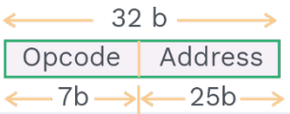
So total, we can have 4×2^5 zero–address instructions.
$2^2 \times 2^5 = 2^7 = 128$ zero–address instructions

Q15 **Consider a microprocessor having both 0–address and 1–address instruction and a fixed size instruction of 32 bits. The main memory is 32 MW in size. If there are 'x' 1–address instructions, calculate the number of 0–address instructions.**

Sol: Memory size = 32 MW

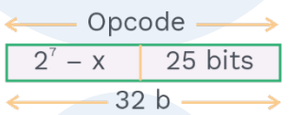
$= 2^5 \times 2^{20} = 2^{25}$ words

so 25 bits will be used for the address field.



1–address instruction format

Total opcodes possible 2^7, 'x' is used for 1–address instructions, remaining 2^7–x can be used for 0–address instructions.



0–address instruction format

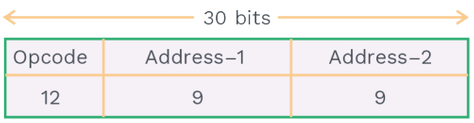
- Number of 0–address instructions possible $= \boxed{(2^7 - x) \times 2^{25}}$ by using expand opcode technique.

Q16 **A computer system supports two–address, and one–address instructions, and the word size are 30 bits. The main memory is word addressable, and it supports 512 words. If there are 160 two address instructions, then how many one address instructions are there in the system?**
a) 2,015,232 b) 1, 007, 616 c) 503, 808 d) 4, 030, 464
Sol: a)

Sol: Given instruction size = 30 bits

address size = 512 words, it requires $\lceil \log_2 512 \rceil$ = 9 bits

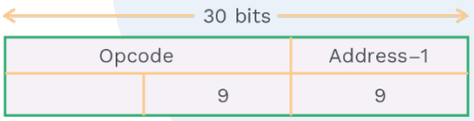
2–address instruction format:



2–Address Instruction Format

Number of 2–address instructions possible = 2^{12} = 4096
But we are using only 160, 2–address instructions,
so remaining opcodes = 2^{12} – 160
= 4096 – 160
= 3936

- We can use these opcodes in 1–address instructions by using expand opcode technique.



One–Address Instruction Format

- So total number of one address instruction = 3936 × 2^9
= 3936 × 512
= 2,015,232

Performance measuring of CPU:

- Performance can be measured on the basis of two criterias.

i) Execution time (ET):

Performance is inversely proportional to the execution time.

$$P(\text{Performance}) \propto \frac{1}{ET}$$

If there are two microprocessors A and B. If A is taking 5 ns and B is taking 10 ns. As $P \propto \frac{1}{ET}$, we can say that A has higher performance than B.

ii) Throughput:

It is defined as the number of jobs completed (executed) per unit time.

Speed–up–factor (S):

If there are two microprocessors X and Y, we want to evaluate how fast is X as compared to Y. As we know $P(\text{Performance}) \propto \frac{1}{ET}$.

$$(\text{Speedup})\, S = \frac{P_X}{P_Y} = \frac{\frac{1}{ET_X}}{\frac{1}{ET_Y}} = \frac{ET_Y}{ET_X}$$

- If X executes a process in 5 ns and Y executes the same process in 20 ns. Then

  $$\text{speedup}(S) = \frac{ET_Y}{ET_X} = \frac{20\,\text{ns}}{5\,\text{ns}}$$

  $$\text{speedup}(S) = 4.$$

  X is 4 times faster than microprocessor Y.

Amdahl's law:

It is a formula which gives the theoretical speedup in latency of the execution of a task at a fixed workload that can be expected from the system whose resources are improved.

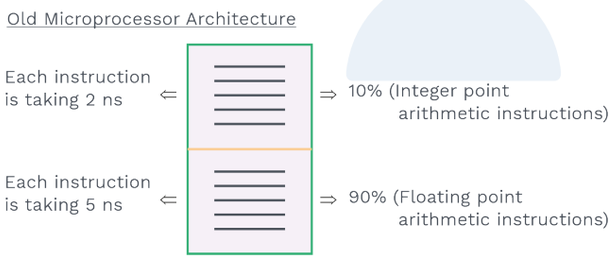


Fig. 2.41 Old Microprocessor Architecture

Expected execution time (old architecture)

ET_{old} = 0.1 × 2 + 0.9 × 5

= 0.2 + 4.5 = 4.7 ns

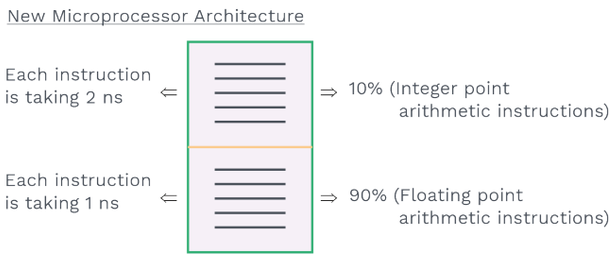


Fig. 2.42 New Microprocessor Architecture

Expected execution time (New architecture)

$ET_{new} = 0.1 \times 2 + 0.9 \times 1$

$= 1.1$ ns

- In Amdahl's law, common case (more frequently occurring instructions) is made fast. For example, floating point instructions are made fast in above scenario.

$$\text{overall speedup} = \frac{ET_{old}}{ET_{new}} = \frac{4.7\,ns}{1.1ns}$$

$$= 4.27$$

Generalised case:

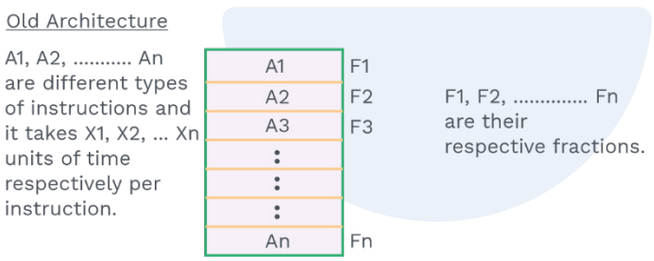


Fig. 2.43 Generalised Case of Old Architecture

Expected/average execution time

$$ET_{old} = \sum_{i=1}^{n} XiFi$$

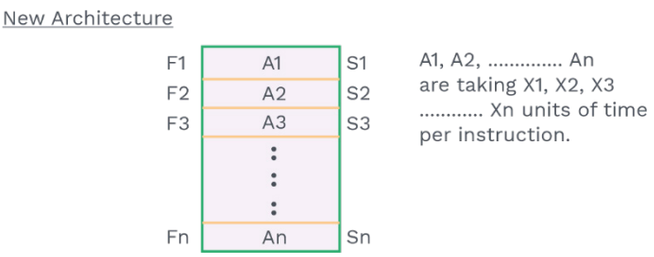


Fig. 2.44 Generalised Case of New Architecture

- S1, S2, S3 ..... Sn are the speedup of A1, A2, A3 ....., An types of instructions respectively, where each S1, S2 ..... Sn are greater than or equal to 1 ($Si \geq 1$) ($i = 1$ to $i = n$)

Expected/Average execution time

$$ET_{new} = \sum_{i=1}^{n} Fi\left(\frac{Xi}{Si}\right)$$

$$\text{Overall speedup} = \frac{ET_{old}}{ET_{new}}$$

Deriving the Amdahl's formula:

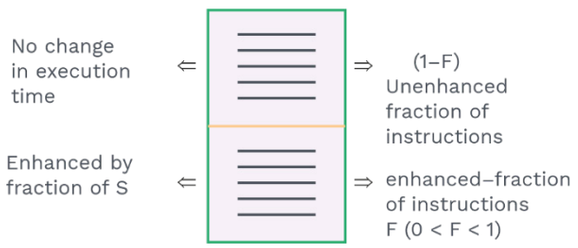


Fig. 2.45 Derivation of Amdahl's Formula

- Assume each instruction initially takes the same time; let's take 1 ns for simplicity.
  ET_{old} = 1 ns

$$\text{Overall speedup } (S_{overall}) = \frac{ET_{old}}{ET_{new}}$$

$$= \frac{1}{1\times(1-F)+\frac{1}{S}\times F}$$

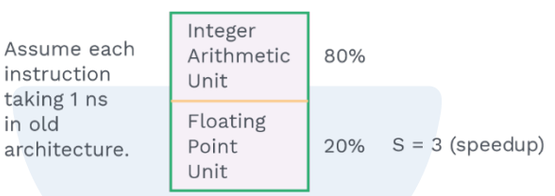
$$= \frac{1}{(1-F)+\frac{F}{S}}$$

$$= \frac{1}{1-(\text{Fraction})_{enhanced}+\frac{(\text{Fraction})_{enhanced}}{(\text{Speedup})_{enhanced}}}$$

PRACTICE QUESTIONS

Q17 **Assume a microprocessor given which is widely used for scientific applications. It has both integer and floating point instructions. Now floating point unit is enhanced and made 3 times faster than before, the integer point unit is still unenhanced. If there are 20% of floating point instructions in the program. Calculate the overall speedup achieved.**

Sol:



$ET_{old} = 1$ ns

$$ET_{new} = 0.8 \times 1 + \frac{0.2}{3}$$

$$= \frac{3 \times 0.8 + 0.2}{3}$$

$$= 0.867$$

$$S_{overall} = \frac{ET_{old}}{ET_{new}} = \frac{1}{0.867} = \boxed{1.153}$$

- Now by using formula–based approach:

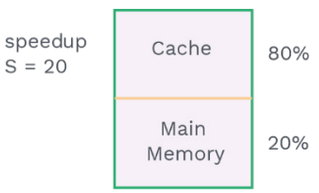
$$= \frac{1}{(1-F) + \frac{F}{S}} \quad \text{Here F = 20\%}$$

$$= \frac{1}{(1-0.2) + \frac{0.2}{3}}$$

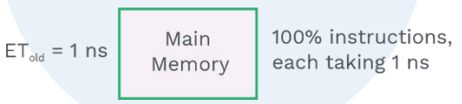
$$= \frac{3}{3 \times 0.8 + 0.2} = \frac{3}{2.6} = \boxed{1.153}$$

Q18 **Assume in a hypothetical system cache is 20 times faster than main memory and CPU refers to the cache 80% of the time. Calculate the overall speedup of this system.**

Sol:



- If we assume that in an old system, only the main memory was there, and each instruction takes 1 ns.



$$ET_{new} = (1-F) + \frac{F}{S}$$

F = 80%

$$S_{overall} = \frac{1}{(1-F) + \frac{F}{S}}$$

$$= \frac{1}{(1-0.8) + \frac{0.8}{20}}$$

$$= \frac{1}{0.2 + 0.04} = \frac{1}{0.24}$$

$$= \boxed{4.167}$$

Q19 **In a hypothetical system, floating point instructions are improved to run five times as fast, but only 40% of the time was spent on these instructions originally. How much speedup is achieved by the new machine?**
a) 1.85 b) 1.47 c) 1.39 d) 1.22
Sol: b)

Sol: By using Amdahl's law:

$$\text{Overall speedup} = \frac{\text{Old execution time}}{\text{New execution time}}$$

$$= \frac{1}{1 - (\text{Fraction})_{\text{enhanced}} + \frac{(\text{Fraction})_{\text{enhanced}}}{(\text{Speedup})_{\text{enhanced}}}}$$

$$= \frac{1}{\left((1 - 0.4) + \frac{0.4}{5}\right)}$$

$$= \frac{1}{(0.6 + 0.08)}$$

$$\frac{1}{0.68} = \boxed{1.47}$$

Processor–clock cycle time and frequency

- **t = clock cycle time,** it is a time interval between two consecutive rising edges or 2 consecutive falling edges.
- **Frequency (f) =** $\frac{1}{t}$ (It is measured in cycles per unit time). Its SI unit is Hertz (Hz).
- $\boxed{\text{CPU time}}$ = Instruction count × CPI × Clock cycle time

CPI = Clocks per instruction
Here, we are assuming each instruction takes the same amount/number of cycles.

- For different instructions, CPU takes a different number of instructions in the real world, e.g., data—transfer instruction, data–manipulation instructions and transfer of control instructions take a different number of cycles.

$$\text{CPU time} = \left[\sum_{i=1}^{K} IC_i \times CPI_i\right] \times t$$

IC_i = Instruction count for i^{th} type of instruction
CPI_i = Clocks per instruction for each i^{th} type of instruction
t = clock cycle time.

PRACTICE QUESTIONS

Q20 **Assume the frequency (clock rate) of CPU is given as 4.5 GHz. Assume we have different type of instructions which have different number of instruction count (IC) and different CPI. Now calculate i) average instruction execution time ii) MIPS–rate iii) Program execution time (ET).**

Instruction type	Instruction count	CPI
Load	**100**	**10**
Store	**200**	**8**
Arithmetic	**300**	**5**
Logical	**150**	**4**
Shift	**250**	**3**
Branch	**400**	**2**

Sol: Clock rate (frequency) = 4.5 GHz

$$t = \text{clock cycle time} = \frac{1}{4.5 \times 10^9}\,\text{sec}$$
$$= 0.22\ \text{ns}$$

a) Average–instruction Execution time (ET) $= \dfrac{\left[\sum_{i=1}^{K} Ic_i \times CPI_i\right] \times t}{\text{Total instruction count}}$

$$= \frac{(100 \times 10 + 200 \times 8 + 300 \times 5 + 150 \times 4 + 250 \times 3 + 400 \times 2) \times 0.22\,\text{ns}}{100 + 200 + 300 + 150 + 250 + 400}$$

$$= \frac{(6250) \times 0.22\,ns}{1400} = 0.9821ns$$

b) MIPS–rate
MIPS = Millions of instructions per second
On average, 1 instruction is taking = 0.9821 ns
In 1 sec, we can execute,

$$\frac{1}{0.9821 \times 10^{-9}}$$

$$= \frac{10^9}{0.9821} = 1.018 \times 10^9 \text{ instructions}$$

$= 1018 \times 10^6$

$= \boxed{1018\,MIPS}$

c) Program execution time
= Total instruction × Average instruction execution time
= 1400 × 0.9821 ns
= 1374.94 ns

Q21 **In a certain processor, we have a program which takes 1 million instructions to execute with processor whose speed is given as 2GHz. Suppose 40% of the instruction takes 2 clock cycles, 30% of the instructions take 3 clock cycles and the remaining 30% takes 5 clock cycles for their respective execution. What is the total execution time for the program (in milliseconds)?**
a) 3.2 **b) 4.8** **c) 1.6** **d) 2.4**
Sol: c)

Sol: We have a total instructions = 10^6
Frequency = 2GHz

$$\text{Clock cycle time} = \frac{1}{\text{Frequency}}$$

$$= \frac{1}{2 \times 10^9} \text{ sec}$$

$$= \boxed{0.5 \times 10^{-9} \text{ sec}}$$

Total execution time = Total CPI × Number of instructions × Clock cycle time

$\boxed{\text{CPI} = \text{Clocks per instruction}}$

Number of clock cycles	Percentage of instructions	Multiplication
2	40% = 0.40	0.40 × 2 = 0.8
3	30% = 0.30	0.30 × 3 = 0.9
5	30% = 0.30	0.30 × 5 = 1.5

Total CPI = 0.8 + 0.9 + 1.5
= 3.2
Total execution time = $3.2 \times 10^{6} \times 0.5 \times 10^{-9}$
= 1.6×10^{-3}
= 1.6 milliseconds
= 1.6 ms

Q22 **Consider a 2 GHz processor which is used to execute a benchmark program with the following instruction and clock cycle count:**

Instuction	Clock cycle count
80000	2
40000	3
5000	3
10000	4

What is the effective CPI of the program and total execution time?
a) CPI = 2.48 **b) Execution time = 2.5 ns**
c) CPI = 3.20 **d) Execution time = 1.24 ns**
(ns = Nanoseconds) **(CPI = Clocks per instruction)**
Sol: a), d)

Sol: Total instruction counts
= 80000 + 40000 + 5000 + 10000
= 135000

$$\text{effective CPI} = \frac{\Sigma\left(\underset{I_i}{\text{Instruction count}}\right) \times \left(\underset{C_i}{\text{Clock cycle count}}\right)}{\text{Total instruction count}}$$

I_i = Instruction count of i^{th} instruction
C_i = Clock cycle of i^{th} instruction

$$= \frac{(80000 \times 2) + (40000 \times 3) + (5000 \times 3) + (10000 \times 4)}{135000}$$

$$= \frac{335000}{135000} = \boxed{2.48}$$

Program execution time = Effective CPI × Clock cycle time

$$\text{Clock cycle time} = \frac{1}{\text{Frequency}} = \frac{1}{2 \times 10^9} \text{ sec}$$

$$\text{Program execution time} = 2.48 \times \frac{1}{2 \times 10^9} \text{ sec}$$

$= 1.24 \times 10^{-9}$ sec = $\boxed{1.24\,\text{ns}}$

CISC and RISC processors:

- CISC = Complex Instruction Set Computer
- It has a large number of instructions

e.g.:

i) MUL [1080] [1050]
ii) MUL r_0, r_1
iii) MUL r_0, [1080]

} Different types of multiplication instruction

- For multiplication, large number of instructions are present:
  - **i)** Multiplication between 2 memory location data.
  - **ii)** Multiplication between 2 register data.
  - **iii)** Multiplication between register and memory location data.
- It has more complex circuitry because of a large number of instructions, so it will consume more power.
- It has fewer registers because most of the space is occupied by the circuitry on the physical chip; hence less space is available for registers.
- Instructions in this architecture have a variable number of cycles to execute.

For example: Some instructions have a long execution time; these are the instructions that copy an entire block from one part of the memory to another part of the memory. Others copy multiple registers to and from memory.

- **i)** MUL [2080] [1050]
- **ii)** MUL r_0, [1080]

Both are multiplication instructions only but take a different number of cycles for execution because

- **i) instruction Involves only memory operands**
- **ii) instruction involves both register and memory operand.**

- Examples of CISC are: Pentium i3, i5, i7.

RISC: Reduced instruction set computer:

- It has fewer instructions.
  MUL R_0, R_1

It can only multiply data in 2 registers.

MUL[1080][1050] will be translated into these four instructions.

LOAD R_0, [1050]
LOAD R_1, [1080]
MUL R_0, R_1
STORE [1080], R_0

- It has less number of instructions, so it has simple circuitry.
- As it has simple circuitry, it will consume less power; for example smart phones and i–pads prefer this (RISC) architecture so that they have longer battery life.
- As it has simpler circuitry, it can have more number registers on the same chip as compared to CISC based architecture.
- It has a fixed number of cycles per instruction (mostly 1 cycle per instruction).
  LOAD R_0, [1050] $\rightarrow$ 1 cycle
  LOAD R_1, [1080] $\rightarrow$ 1 cycle
  MUL R_0, R_1 $\rightarrow$ 1 cycle
  STORE [1080], R_0 $\rightarrow$ 1 cycle
- All these can be executed in 1 cycle.
- Examples of RISC architecture-based devices are Mobile, i–pads and ARM processors for laptops.
- Key differences between RISC and CISC

S. No.	RISC	CISC
1)	Simple instructions taking one cycle	Complex instructions taking multiple cycles
2)	Supports fixed format instructions	Supports variable format instructions
3)	Contains multiple register sets	Contains single register set
4)	Few instructions	Many instructions
5)	Highly pipelined	Not pipelined or less pipelined
6)	Few addressing modes, and most instructions have register to register addressing mode	Many addressing modes

S. No.	RISC	CISC
7)	Consumes less power	Consumes more power
8)	Uses hardwired control unit	Uses microprogrammed control unit
9)	Very few instructions refer to memory	Most of the instructions may refer to memory

Table 1.5 Comparison between RISC and CISC Architectures

Register–organisation in RISC:

$\boxed{G(g)}$ G = Global register file

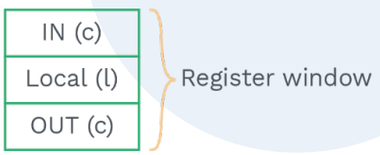


Fig. 2.46 Layout of Register Window

Register window = Collection of registers, having IN, OUT (Common registers) and local registers (l).

g = number of global registers
l = number of local registers per window
c = number of common registers per window
w = number of windows

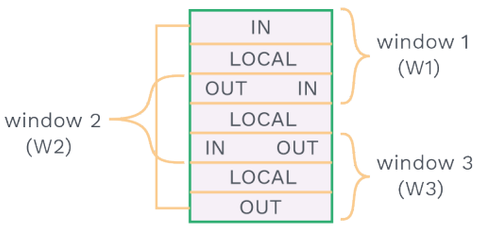


Fig. 2.47 Overlapped Register Window

- OUT of W1 is shared with IN of W2; similarly OUT of W2 is shared with IN of W3 and OUT of W3 is shared with IN of W1.

Total number of registers:

$$g + w \times l + \frac{w(2 \times c)}{2}$$

$$= \boxed{g + w(l + c)}$$

PRACTICE QUESTIONS

Q23 **A RISC processor has 400 registers. Each window has 4 inputs, 20 local and 4 output registers. The number of register windows is 10. What is the total number of global registers?**

a) 180 **b) 140** **c) 160** **d) 200**

Sol: c)

Sol: Given,

L = Local register = 20

G = Global register = G

C = Common register = 4

W = Number of windows = 10

- In RISC, overlapping of registers is there, 'Input' register of one window overlaps with 'output' register of other. So, only one among the 'input' and 'output' registers is counted.

Total registers = 400

Number of registers in RISC processor

$400 = G + W (L + C)$

$400 = G + 10 (20 + 4)$

$400 = G + 10 (24)$

$400 = G + 240$

$\boxed{G = 160}$

Previous Years' Question

Consider the following processor design characteristics.

1) Register–to–register arithmetic operations only

2) Fixed–length instruction format

3) Hardwired control unit

Which of the characteristics above are used in the design of a RISC processor?

a) 1 and 2 only

b) 2 and 3 only

c) 1 and 3 only

d) 1, 2 and 3

Sol: d) **(GATE-2018)**

Instruction set:

i) Arithmetic Instructions:

a) INC (Increment) instruction:

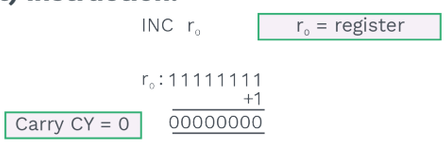


Fig. 2.48 Increment Instruction

b) DEC r_0 (Decrement) instruction:

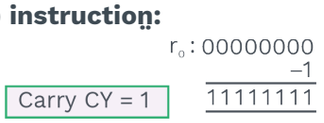


Fig. 2.49 Decrement Instruction

- If the initial value of the carry flag is 1, then after the above operation, it will still remain the same, it will not be 0 as happens in a typical SUB operation. It would not change.
- So, in increment and decrement instructions carry value will not change; it will remain the same as before in most microprocessors.

ii) Logical Instructions:

1) AND, OR and EX–OR:

Binary number	AND	OR	EX–OR
0 0	0	0	0
0 1	0	1	1
1 0	0	1	1
1 1	1	1	0

Table 1.6 AND, OR and EX-OR Operations

- Every microprocessor provides these logical instructions.

2) CMP (Comparison) instruction:

CMP r_0 r_1 (Comparison of register r_0 and r_1)

- Internally ALU is used and ($r_0 - r_1$) is performed through subtraction, the and result is stored in accumulator.

CMP	CY (Carry flag)	Z (Zero flag)
$r_0 = r_1$	0	1
$r_0 > r_1$	0	0
$r_0 < r_1$	1	0

Table 1.7 Status of CY and Z Flags in CMP Instruction

- We can compare r_0 and r_1 on the basis of carry (CY) and zero flags (Z), which are stored in the program status word (PSW).

iii) Shift operations:

1) Logical shift right:

Least Significant Bit (LSB) will fall off, and MSB will be filled with '0'.

Example:

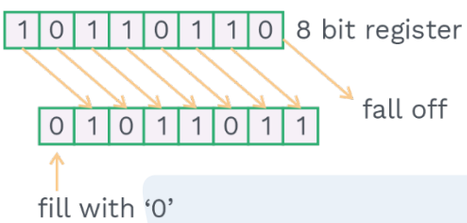


Fig. 2.50 Logical Shift Right Operation

- We can shift many times in the same manner.

2) Logical shift left:

It is opposite to the right shift. The most significant bit (MSB) will fall off, and LSB (Least significant bit) will be filled with '0'.

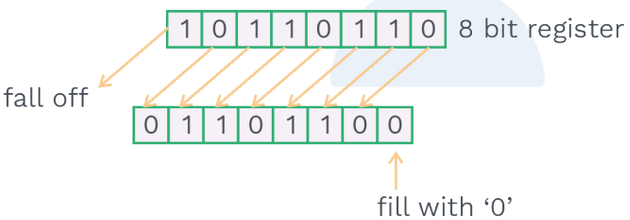


Fig. 2.51 Logical Shift Left Operation

- We can shift as many times as asked to shift.

3) Arithmetic right shift:

(LSB) Least Significant Bit will fall off, and MSB (Most Significant Bit) will be filled with the same bit value of MSB as before.

Example: 1

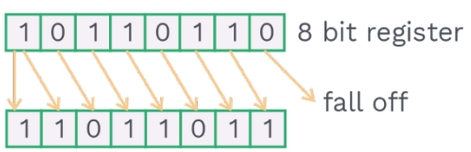


Fig. 2.52 Arithmetic Shift Right Operation I

Example: 2

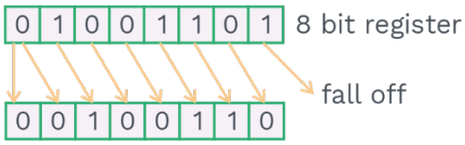


Fig. 2.53 Arithmetic Shift Right Operations II

If we consider these numbers like 2's complement number, then a negative number will remain negative, and a positive number will remain positive in the arithmetic right shift because MSB bit is not changed; hence the sign remains the same.

4) Arithmetic left shift:

It is equivalent to a logical left shift. So we don't need two different circuits and two different instructions for this operation.

5) Rotate right/left:

Here bit will not fall, rather it will enter into the register from another side , i.e. rotation.

i)

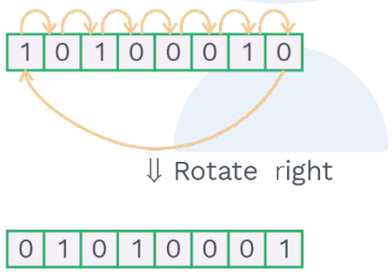


Fig. 1.54 ROR Operation

ii)

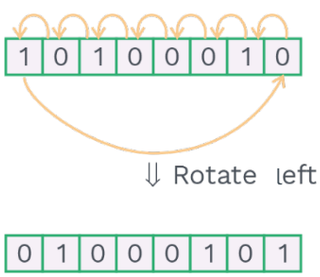


Fig. 2.55 ROL Instruction

6) Rotate right/left through carry:

In this operation, one more bit is appended, i.e. carry flag.

(Think it as ⑨ bit register)

c) Carry flag

Fig. 2.56 Register With CY Flag

i)

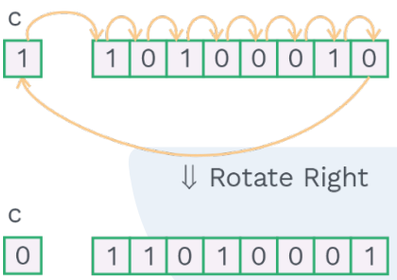


Fig. 2.57 RCR Operation

ii)

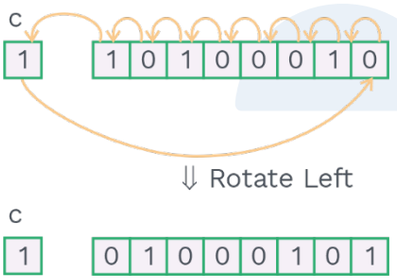


Fig. 2.58 RCL Operation

Chapter Summary

Types of CPU organisation:

1) **Stack CPU:** ALU operations are performed only on stack data.
2) **Accumulator CPU:** ALU first operand required in an accumulator.
3) **General register CPU:** Architecture is of two kinds:
   a) Register to memory reference: ALU first operand is always required in register, and the second operand can be in memory/register.
   b) Register to register: Both operands are required in a register.

Instruction format:

- An instruction format defines layout of bits of an instruction in terms of its constituent parts.

Elements of machine instruction:

1) Operation Code
2) Source Operand Reference
3) Result Operand Reference
4) Next Instruction Reference

Number of addresses:

1) Zero address instruction
2) One address instruction
3) Two address instruction
4) Three address instruction

Addressing modes: Shows the location of a required object on computer.

Sequential control flow AMs:

1) Register AM
2) Implied/Implicit AM
3) Immediate AM
4) Direct/Absolute AM
5) Indirect AM
6) Indexed/Base–Indexed AM
7) Indirect Indexed AM
8) Auto–Indexed AM

Transfer Flow of Control flow AMs

1) Relative/PC – relative Addressing Mode
2) Based/Base Register Addressing Mode

Chapter Summary

Expand–Opcode technique:
The address field of primitive instruction is joined with the free opcodes to generate derived opcodes.

Performance measuring of CPU:
1) Execution Time
2) Throughput
3) speedup
Amdahl's law:

$$= \frac{1}{1 - (\text{Fraction})_{\text{enhanced}} + \frac{(\text{Fraction})_{\text{enhanced}}}{(\text{Speedup})_{\text{enhanced}}}}$$

Processor clock cycle time and frequency:
Clock cycle time: Interval between 2 consecutive rising or falling edges.

$$\textbf{Frequency} = \frac{1}{\text{clock cycle time}}$$

CISC architecture and RISC architecture:
Key difference:

S. No.	RISC	CISC
1)	Few instructions	Many instructions
2)	Highly pipelined	Less pipelined or not pipelined
3)	Fixed format instruction	Variable format instruction
4)	Uses hardwired control unit	Uses microprogrammed control unit

Register–organisation in RISC:

Total number of registers = $\boxed{g + w(l + c)}$