Unit-5.
Central Processing Unit

1) Introduction:

⊛ Major Components of CPU:

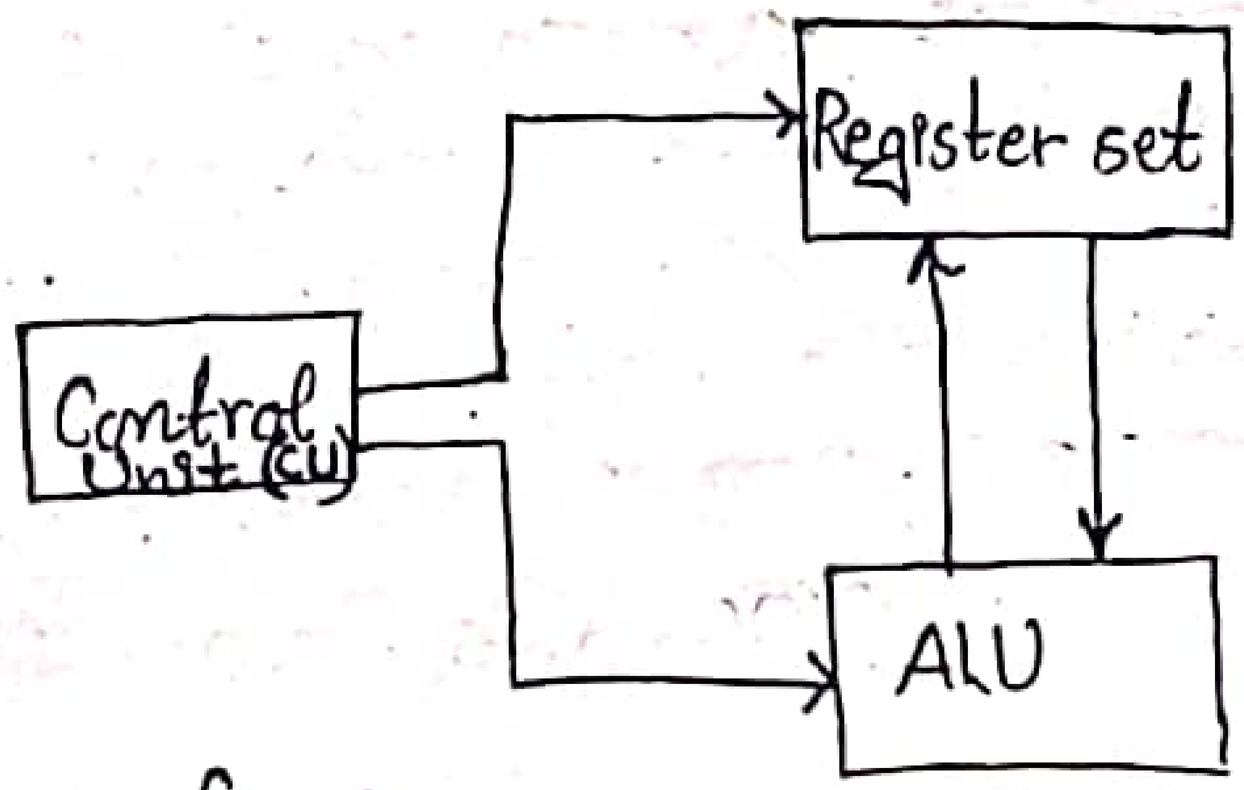


fig. Major Components of CPU

Central Processing Unit (CPU) is the brain of computer the performs data processing operations. Intermediate data is stored in the register set during the execution of the instructions. The microoperations required for executing the instructions are performed by the arithmetic logic unit whereas the control unit takes care of transfer of information among the registers and guides the ALU.

⊛ CPU Organizations:-

i) Accumulator based organization.
ii) General register organization.
iii) Stack organization.

i) Accumulator based organization:- (Less imp).

Accumulator based organization is ~~whe~~ used when we are trying to make the cheapest system. Since the cost of CPU depends on number of registers. Register are fastest and costliest memory units. The more number of registers the more will be it's cost.

In CPU organization, the first ALU operand is always stored into the accumulator and the second operand is present either in Registers or in the memory.

Accumulator is the default address thus after data manipulation the results are stored into the accumulator.
The format of instruction is;

Instriction = Opcode + Address.

Opcode indicates the type of operation to be performed. Mainly two types of operations are performed in single accumulator based organization; Data transfer operation and ALU operation. In Data transfer operation, the data is transfered from a source to a destination for e.g. LOADX. In ALU arithmetic operations are performed on data. for e.g. MULT X. where, X is the address of operand.

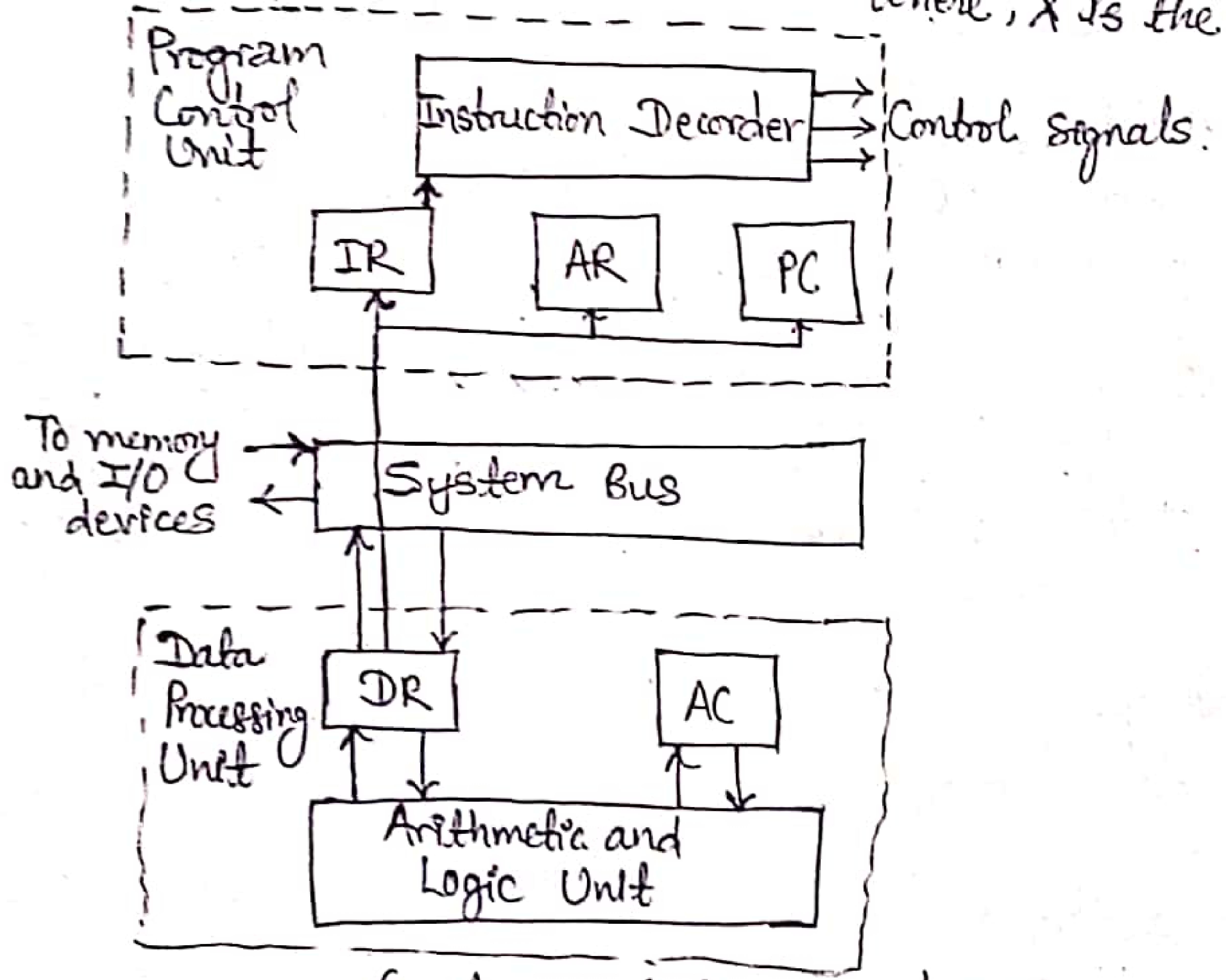


fig. Accumulator based CPU organization.

Advantage

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

Disdavantage

i) When complex expressions are computed memory size increases.
ii) Increase in number of instructions increases execution time.

General Register Organization:

General Register Organization is done when we are taking concern on speed of CPU rather than cost. In accumulator based organization we used for 1 address instruction but in general register organization we use for 2 address or 3 address instruction, since multiple registers are used in this type of organization.

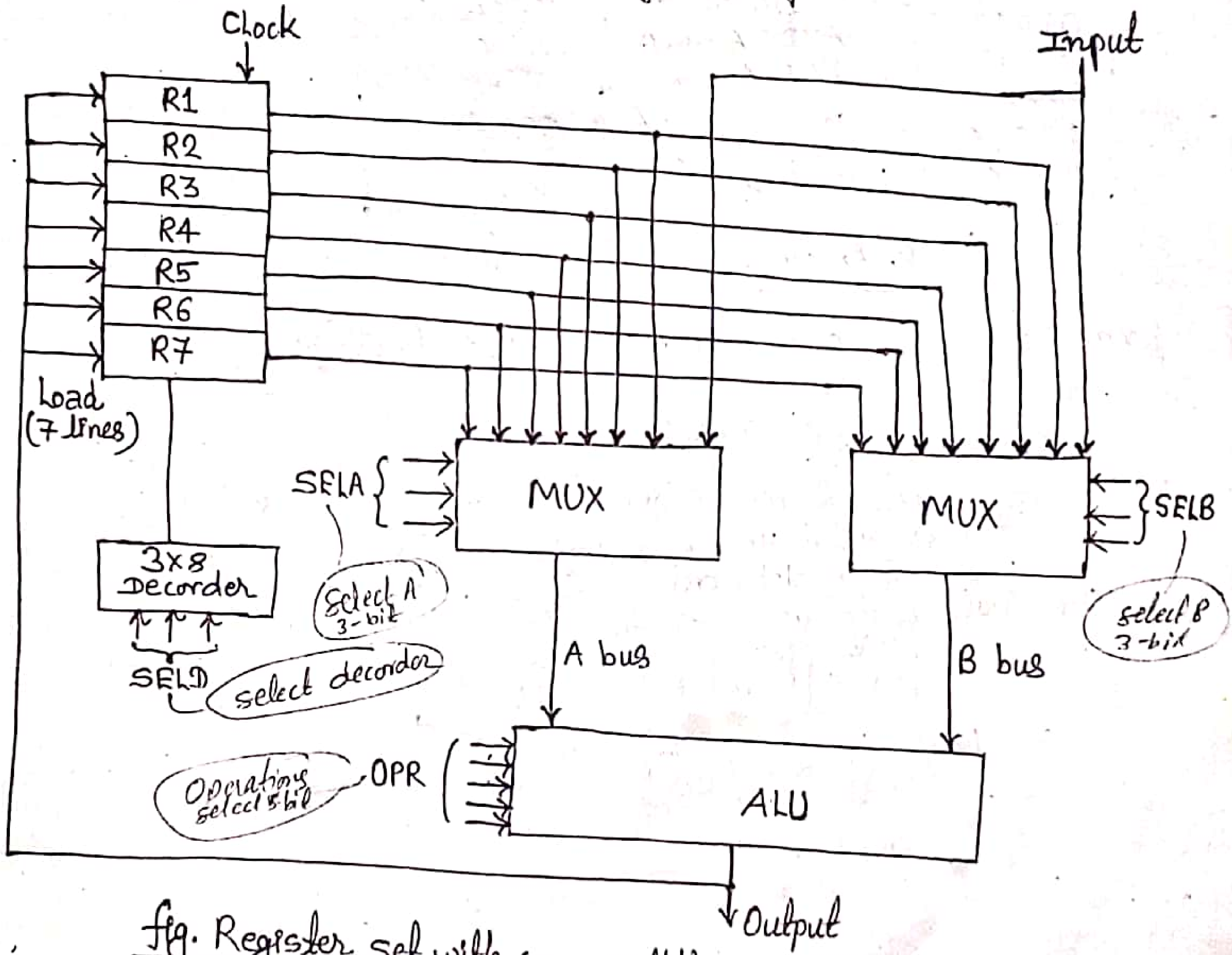


fig. Register set with common ALU

Control Word:

3	3	3	5
SELA	SELB	SELD	OPR

Binary code	SELA	SELB	SELD
000	Input	Input	None
001	R_1	R_1	R_1
010	R_2	R_2	R_2
011	R_3	R_3	R_3
100	R_4	R_4	R_4
101	R_5	R_5	R_5
110	R_6	R_6	R_6
111	R_7	R_7	R_7

← Table for encoding of register selection field.

OPR Select	Operation	Symbol.
00000	Transfer A	TSFA
00001	Increment A	INCA
00010 .	Add A+B .	ADD
00101	Subtract A−B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA.

since OPR is of 5-bits

Table: Encoding of ALU operations

Example:- Write control to execute $R_1 \leftarrow R_2 + R_3$.

Solution,

Given, $R_1 \leftarrow R_2 + R_3$

Since, $R_2 + R_3$ means operation show we look select value for addition in above operation table.

So, $R_2 + R_3$ denotes 00010 in operation field.

We have control word as follows:

3	3	3	5
SELA	SELB	SELD	OPR

Now,

$$\begin{tikzcd}
R_1 & \leftarrow R_2 & + & R_3 & 00010 \text{ (from \sout{oper} OPR table)} \arrow[ll, bend right] \\
001 \arrow[u] & 010 \arrow[u] & & 011 \arrow[u]
\end{tikzcd}$$

$\underbrace{\hspace{6em}}_{\text{from encoding table}}$

Hence control to execute is as follows:-

010	011	001	00010

iii) Stack Organization:-

→ A stack is a storage device that stores information in such the manner that the item stored lased is the first item retrived. (i.e, LIFO list).

→ The register that holds the address for the stack is called stack pointer (SP), because its value always pointsat the top item in the stack.

→ Two operations are performed during stack organization
ⓐ Push operation/insertion ⓑ Pop operation/deletion.

→ It is implemented either by <u>register stack</u> or by <u>memory stack</u>.

ⓐ Register stack:

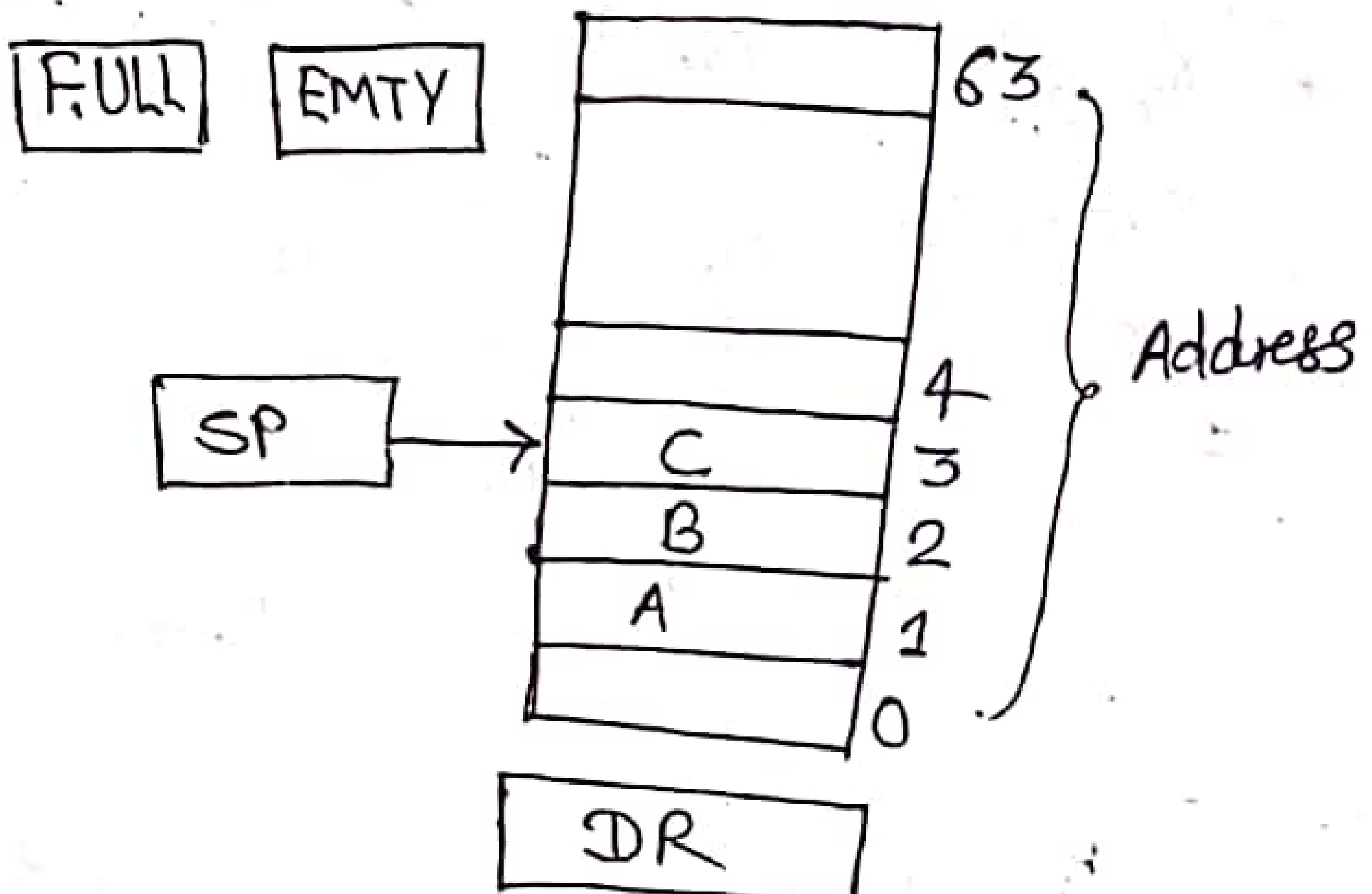


fig. Block diagram of a 64-word stack

The stack pointer (SP) contains a binary number whose value is equal to the address of word that is currently on top of the stack. The three items; A, B and C are placed in the stack in that order. Item 3 is now on top of the stack so that content of SP is now 3. To remove the top item we take Pop operation and decrement content of SP. Now item B is on top of stack and holds address 2. Similarly to insert an item we take Push operation and increment content of SP. Now the top item is C and holds address 3.

In 64-word stack, the stack pointer contains 6 bits because $2^6 = 64$. Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary). When 63 is incremented by 1 the result is 0 since 111111 + 1 = 1000000 in binary but SP can store only six bits (ie, becomes 000000). Similarly, when 000000 is decremented by 1 the result is 111111. One bit register FULL is set to 1 when stack is full and EMTY set to 1 when stack is empty. DR is data register that holds binary data to be read/write out of stack.

PUSH operation:

If the stack is not full (i.e FULL=0), a new item is inserted with a push operation. The push operation consists of following sequence of microoperations:

$SP \leftarrow SP+1$	Increment stackpointer.
$M[SP] \leftarrow DR$	WRITE ITEM ON TOP OF THE STACK.
IF $(SP=0)$ then $(FULL \leftarrow 1)$	Check is stack is full
$EMTY \leftarrow 0$	Mark the stack not empty.

POP operation:

If the stack is not empty (i.e, if EMTY=0), a new item is deleted from stack. The pop operation consists of the following sequence of microoperations:-

$DR \leftarrow M[SP]$	Read item on top of stack.
$SP \leftarrow SP-1$	decrement stack pointer.
IF $(SP=0)$ then $(EMTY \leftarrow 1)$	Check if stack is empty.
$FULL \leftarrow 0$	Mark the stack not full.

(b) Memory Stack:

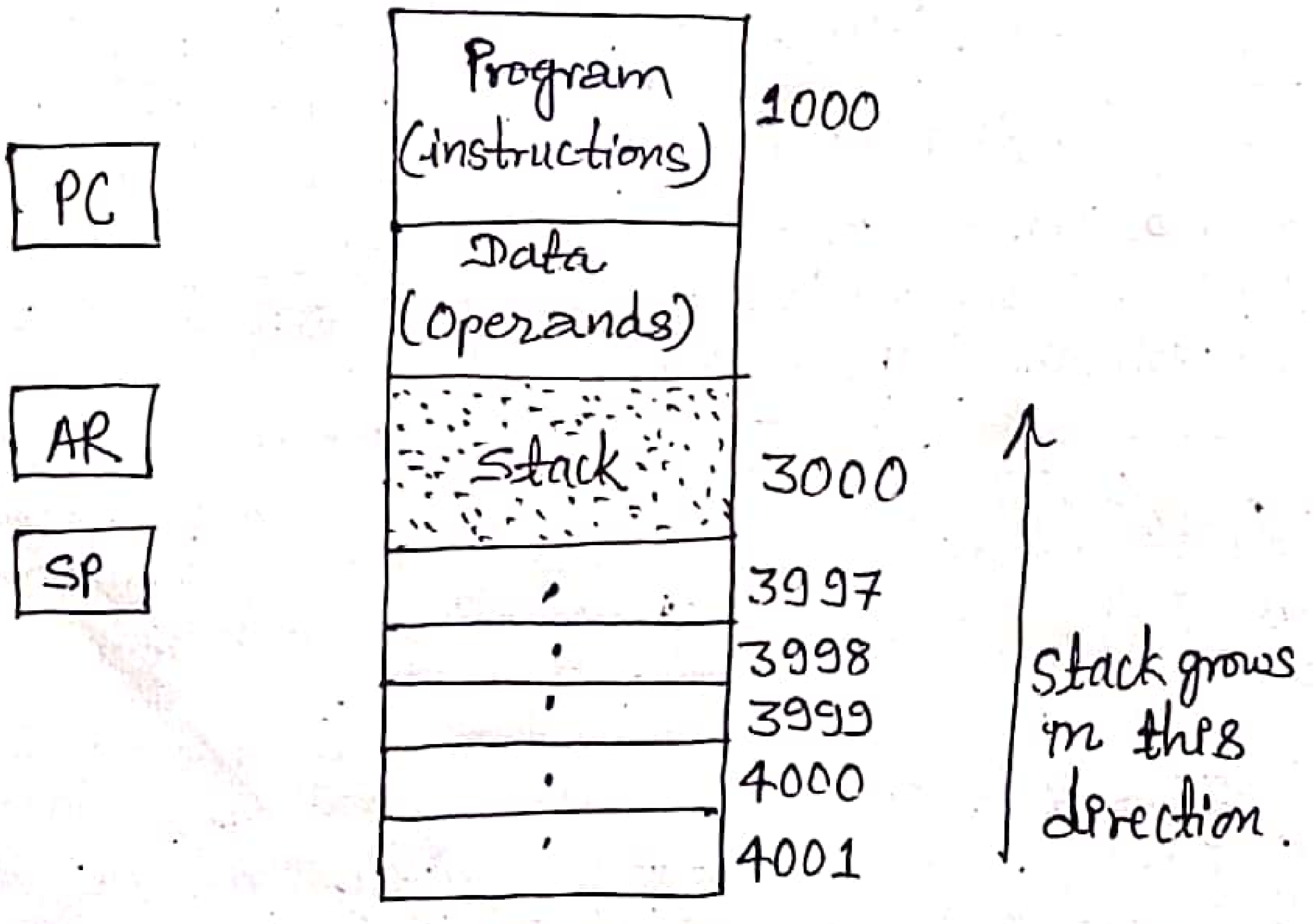


fig. Computer memory stack

The program counter (PC) points to the address of next instruction in the program. The address register (AR) points to an array of data which is used during the execute phase to read an operand. The stack pointer (SP) points at the top of the stack which is used to push or pop items into or from the stack.

PUSH operation:

A new item is inserted with the push operation as follows:

$SP \leftarrow SP-1$ Stack pointer is decremented

$M[SP] \leftarrow DR$ A memory write operation inserts the word from DR into the top of the stack.

POP operation:

A new item is deleted with a pop operation as follows;

$DR \leftarrow M[SP]$ The top item is read from stack into DR.

$SP \leftarrow SP+1$ The stack pointer is then incremented.

2> CPU Instructions:-

⊛. Instruction Formats:- The bits of the instructions are divided into groups called fields. The most common fields found in the instruction format are:-

a> Opcode field.
b> Address field
c> Mode field.

On the basis of number of address field found in instruction formats, instructions are categorised as;

a> One address instruction [ADD, SUB, DIV, MUL, LOAD, STORE]
e.g:- ADD X , $AC \leftarrow AC+M[X]$;

b> Two address instruction [ADD, SUB, DIV, MUL, MOV]
eg:- ADD R_1, R_2, $R_1 \leftarrow R_1+R_2$;

c> Three address instruction [ADD, SUB, DIV, MUL].
e.g:- ADD R_1, R_2, R_3, $R_1 \leftarrow R_2+R_3$;

d> Zero address instruction [PUSH, POP, ADD, SUB, MUL, DIV]
e.g:- ADD B , $AC \leftarrow AC+M[B]$;

V. Imp

Q. Evaluate the following arithmetic expression using zero, one, two and three address instruction.

ⓐ $y = (A+B) * (C-D)$

ⓑ $y = \frac{(A+B) * (C-D)}{(A+D)}$

ⓒ $y = A[(B+C)/(C-D)] + AB$

ⓓ $y = (A+B)/(C-D)$

But Not in micro-syllabus and not asked in model and past year. It is in exercise question and has possibility.

ⓐ Solution: Using zero address instruction

```
PUSH A ;  TOS ← A
PUSH B ;  TOS ← B
ADD ;     TOS ← A+B
PUSH C ;  TOS ← C
PUSH D ;  TOS ← D
SUB ;     TOS ← C-D
MUL ;     TOS ← (A+B) * (C-D)
POP y ;   y ← TOS
```

Using One address instruction

```
LOAD A ;  AC ← M[A]
ADD B ;   AC ← AC + M[B]
STORE T ; M[T] ← AC
LOAD C ;  AC ← M[C]
SUB D ;   AC ← AC - M[D]
MUL T ;   AC ← AC * M[T]
STORE y ; M[y] ← AC
```

Using two address instruction

```
MOV R1, A ;   R1 ← M[A]
ADD R1, B ;   R1 ← R1 + M[B]
MOV R2, C ;   R2 ← M[C]
SUB R2, D ;   R2 ← R2 - M[D]
MUL R1, R2 ;  R1 * R2
MOV y, R1 ;   M[y] ← R1.
```

Using Three address instruction

ADD R_1, A, B ; $R_1 \leftarrow M[A] + M[B]$
SUB R_2, C, D ; $R_2 \leftarrow M[C] + M[D]$
MUL y, R_1, R_2 ; $y \leftarrow R_1 * R_2$.

Similarly we can solve for b, c, d.

⊛ Addressing modes: The way in which the operant of an instruction is specified are called addressing modes. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.

Purpose

→ To give programming flexibility to the user.
→ To use the bits in the address field of the instruction efficiently.

Types of addressing modes:-

ⓐ Implied mode:

→ Address of the operands are specified implicitly, in the definition of the instruction.
→ No need to specify address in the instruction.
Effective address → EA = AC , or EA = Stack [SP]

ⓑ Immediate mode:

→ Operand is specified in the instruction itself.
→ No need to specify address in the instruction.
→ However operand itself needs to be specified.
→ Fast to acquire an operand.

ⓒ Register mode:-

→ Operands are within registers the reside within the CPU.
→ Shorter address than the memory address
→ Saving address field in the instruction.
→ Faster to acquire an operand than the memory addressing.
→ EAA = IR(R) (IR(R) denotes register field of IR).

(d) Register indirect mode:-

→ Instruction specifies a register which contains the memory address of the operand.

→ Slower to acquire an operand than both the register addressing or memory addressing.

→ EA = [IR(R)] ∵ ([X]: Content of X).

(e). Auto-increment or Auto-decrement mode:

→ It is same as the Register Indirect, but: When the address in the register is used to access memory, the value in the register is incremented or decremented by 1.(after or before the execution of the instruction).

(f). Direct address mode:

→ Instruction specifies the memory address which can be used directly to the physical memory.

→ faster than the other memory addressing modes.

(g). Indirect addressing mode:

→ The address field of an instruction specifies the address of a memory location that contains address of the operand.

→ Slow to acquire an operand because of an additional memory access.

(h) Relative addressing mode:

→ The address fields of an instruction specifies the part of the address which can be used along with designated register to calculate address of the operand.

→ Address field of instruction is short.

(i). Indexed addressing mode:

$EA = XR + IR$ where, XR: Index Register
IR: Instruction Register (address)

(j). Base register addressing mode:

$EA = BAR + IR$ where, BAR: Base address register
IR: Instruction Register (address)

Imp ⊛ Addressing Modes – Examples [Numerical Example for addressing modes]:

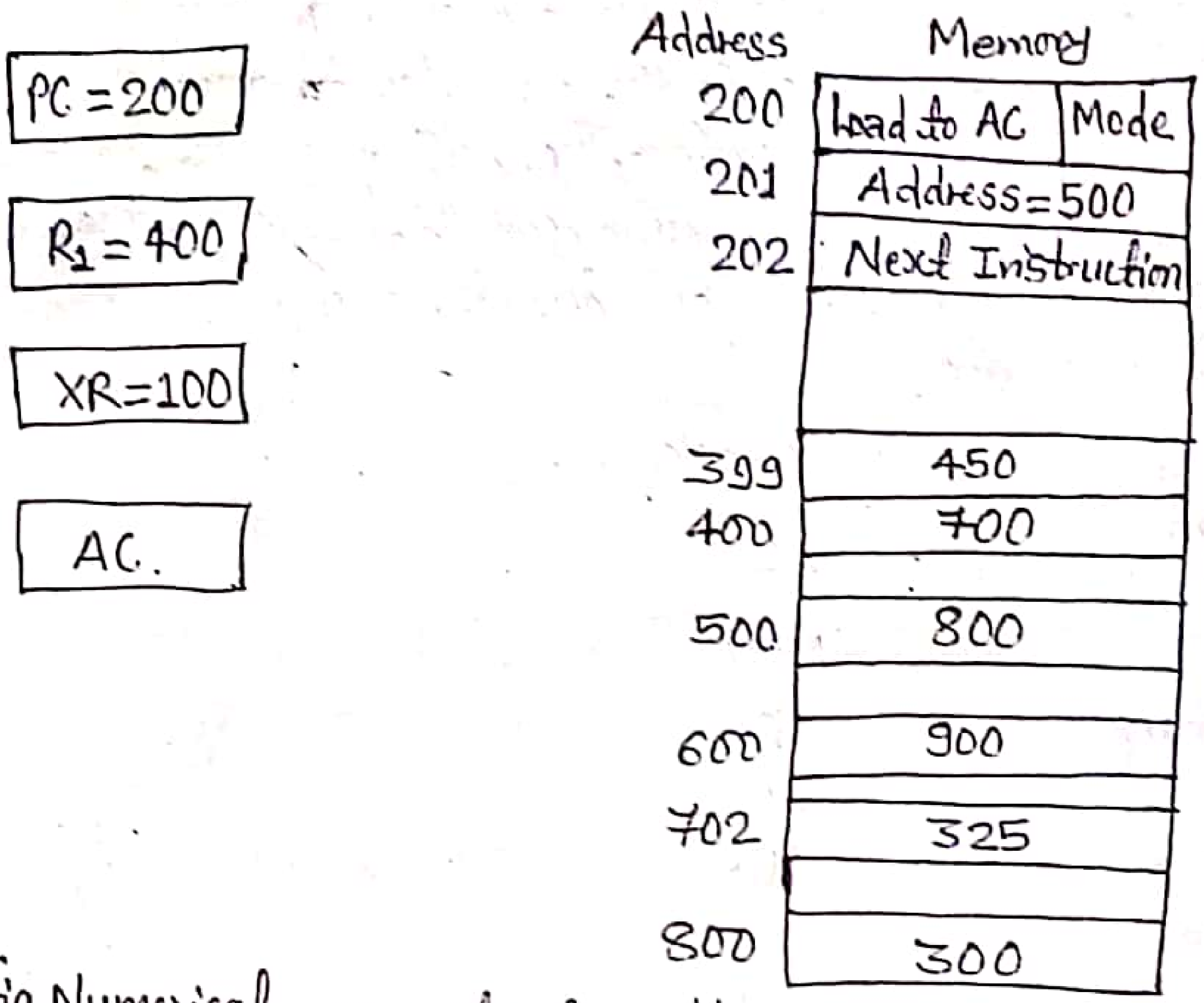


fig. Numerical example for addressing modes

Mode Address	Addressing Effective		Content of AC
Direct address	500	$AC \leftarrow (500)$	800
Immediate operand	—	$AC \leftarrow 500$	500
Indirect address	800	$AC \leftarrow ((500))$	300
Relative address	702	$AC \leftarrow (PC+500)$	325
Indexed address	600	$AC \leftarrow (XR+500)$	900
Register	—	$AC \leftarrow R_1$	400
Register indirect	400	$AC \leftarrow (R_1)$	700
Auto increment	400	$AC \leftarrow (R_1)+$	700
Auto decrement	399	$AC \leftarrow (R)$	450

⊛ Types of instructions on the basis of types of operations:

i) **Data transfer instructions:** Data transfer instructions move data from one place in the computer to another without changing the data content. The most common transfers are between memory and processor registers, between processor registers and input or output and between the processor register themselves.

Name	Mnemonic.
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

ii) **Data Manipulation Instructions:** Data manipulation instructions perform operations on data and provide computational capibilities for the computers. There are three basic types:

ⓐ Arithmetic instructions
ⓑ Logical instructions & bit mainupulation instructions
ⓒ Shift instructions.

Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with Carry	ADDC
Subtract with Borrow	SUBB
Negate (2's complement)	NEG

Logical and Bit manipulation instructions:

Name	Mnemonic.
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI

Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC.

iii) Program Control Instructions:

Unconditional Branching Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RTN
Compare	CMP
Test	TST

Conditional Branching Instructions

Mnemonic	Branch condition	tested condition
BZ	Branch if zero	Z=1
BNZ	Branch if not zero	Z=0
BC	Branch if carry	C=1
BNC	Branch if no carry	C=0

⊛ Subroutine Call and Return:-

We have discussed about description of subroutine already in unit 4.

· Subroutine Call: A call subroutine instruction consists of an operation code along with an address that specify the begining address of a subroutine. This instruction is executed by performing two operations;

i) The address of next instruction (PC) is stored in memory stack.

ii) The PC (Program Counter) is located with the starting address of the subroutine.

List of microoperations performed during subroutine call are as follows:-

SP ← SP-1
M[SP] ← PC
PC ← EA (starting address of subroutine)

Subroutine Return: The last instruction from subroutine causes a return to the address stored in stack. Save the return address to get the address of the location in the calling program. Locations for storing Return address are as;

→ Fixed location in the subroutine (Memory).
→ In a Register.
→ In a memory stack etc.

List of microoperations performed during subroutine Return are as follows:-

PC ← M[SP]
SP ← SP+1.

⊛. Program Interrupt:-

Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request. Control returns to the original program after the service program gets executed.

Types of interrupts:-

There are three major types of interrupts that cause a break in the normal execution of a program. They can be classified as;

Ⓐ **External interrupts** → External interrupts come from I/O devices, from a timing device, from a circuit monitoring the power supply, or from any other external source. Examples that cause external interrupts are I/O device requesting transfer of data, I/O device finished transfer of data, power failure etc. Timeout interrupt may result from a program that is in an endless loop and thus exceeded its time allocation. External interrupts are asynchronous. External interrupts depend on external conditions that are independent of program being executed at the time.

ⓑ. Internal interrupts:→ Internal interrupts arise from illegal or incorrect use of instruction or data. Internal interrupts are also called traps. Examples of internal interrupts are attempt to divide by zero, an invalid operation code, register overflow, stack overflow, protection violation etc. Internal interrupts are synchronous with the program. If the program is return, the internal interrupts will occur in the same place each time.

ⓒ. Software interrupts:→ A software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program. The most common use of software interrupt is associated with a supervisor call instruction.

3> RISC and CISC:-

RISC VS CISC

Complex Instruction Set Computer (CISC)	Reduced Instruction Set Computer (RISC)
i) It gives emphasis on hardware	i) It gives emphasis on software.
ii) It has multiple instruction sizes and formats.	ii) It has instructions of same set with few formats.
iii) It uses less registers	iii) It uses more registers.
iv) It has more addressing modes.	iv) It has fewer addressing modes.
v) Pipeling is difficult in CISC	v) Pipelining is easy in RISC.
vi) As it consists of complex instructions, it takes multiple cycles to execute.	vi) It consists of simple instructions that take single cycle to execute.
vii) Coding in CISC processor is simple.	vii) Coding in RISC processor requires more number of lines.

Advantages of CISC:

i) Microprogramming is easy to implement and much less expensive than hard wiring a control unit.
ii) It is easy to add new commands into the chip without changing the structure of instruction set as the architecture uses general-purpose hardware to carry out commands.
iii) This architecture makes the efficient use of main memory.
iv) The compiler need not be very complicated, as the microprogram instruction sets can be written to match the constructs of high level languages.

Disadvantages of CISC:

i) Chip-hardware and instruction set became complex with each generation of processor.
ii) The overall performance of machine is reduced due to different amount of clock time required by different instructions.
iii) This architecture requires on-chip hardware to be continiously reprogrammed.
iv) The complexity of hardware and on-chip software increases.

Advantages of RISC

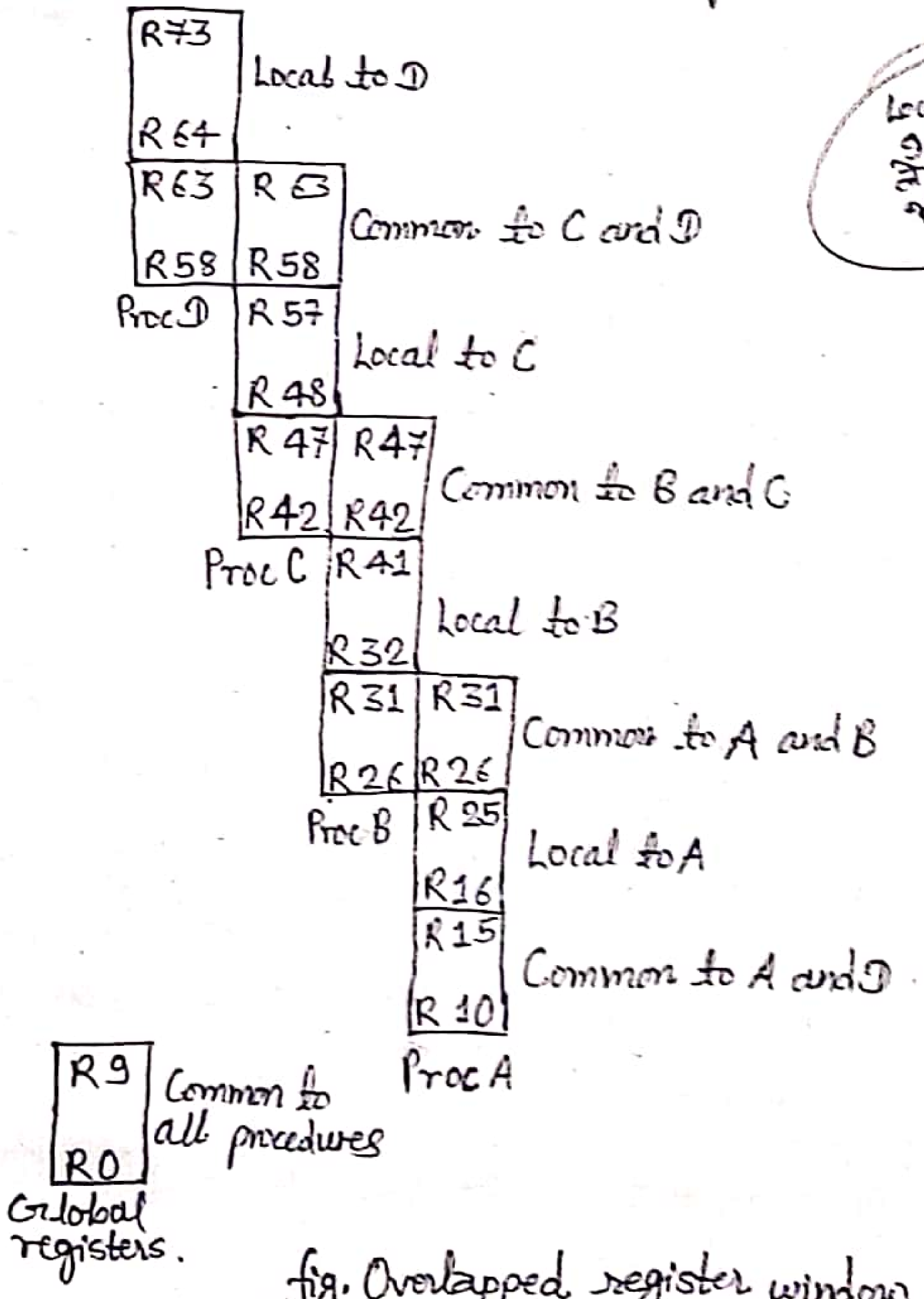
i) The performance of RISC processor is two to four times better than CISC.
ii) The ~~architecut~~ architecture uses less chip space due to reduced instruction set.
iii) RISC processors can be designed more quickly than CISC processors.
iv) The execution of instructions is high in RISC due to use of many registers.

Disadvantages of RISC:

i) When compiler makes poor job of sheduling instructions the processor spends much time waiting for first instruction result.
ii) RISC processors require very fast memory systems to feed various instructions.

⊛. Register Overlapped Windows:

If we use multiple small sets of registers (windows), each assigned to a different procedure, a procedure call automatically switches the CPU to use a different window of registers rather than saving registers in memory. Windows for adjcent procedures are overlapped to allow parameter passing.



Local भएको ठाउँमा 9 को difference ~~[illegible]~~ common भएको ठाउमा 5 को।

fig. Overlapped register window

There are three classes of register windows.

i) Global Registers, G → Available to all functions
ii) Window Local registers, L → Variables local to the function
iii) Window shared registers, C → Permit data to be shared needing to copy it. Window size, W = L+2C+G & Total no of registers = (L+C)W+G.