I] Performance and Numerical. Chapter - 1 (Patterson)

★ (1.6 Performance)

1.1) Introduction:

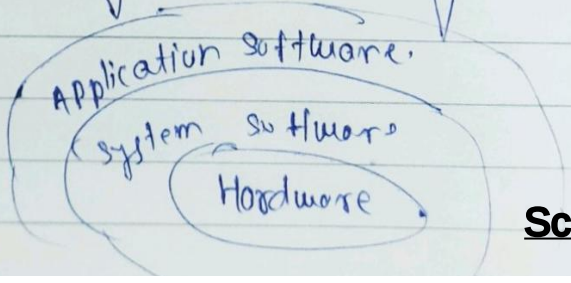
Traditional classes of computer applⁿ.

1) PC (Personal computer) — comp used under another device.
2) Servers ↑
3) ~~Supercomputer~~. Imbedded computers.

		Decimal term	Binary term.
Kilobyte	KB	10^3	2^{10}
megabyte	MB	10^6	2^{20}
Gigabyte	GB	10^9	2^{30}
Terabyte	TB	10^{12}	2^{40}

★ 8 great ideas in computer design

1) Design for moore's law.
2) use abstraction to simplify design.
3) make common case first - enhance performance
4) Performance via parallelism - open in parallel.
5) Performance via pipelining
6) Performance via prediction
7) Hierarchy of memory - cheap at bottom level
8) Dependibility via Redundancy. - keeping backup.



Integrated circuit - also called chips - they drive the advance technology.

Processor - operates at some clock rate
- also called as CPU.

5 part of computer - Input - memory
- Output - data flow path
- Control

processor has two part, datapath
control.

datapath - performs the arithmetic oper.
control - it tells datapath, memory etc. what to do.

memory is build from DRAM chips. DRAM provide allelf to any loc in memory.

Processor has one more type of memory - Cache memory

Cache memory - consists of small, fast memory that acts as buffer for DRAM.

Cache is build using SRAM - Static RAM) it is faster but less dense & is expensive.

#) Abstration interface between Hardware and lowest-level software is Instruction Set Architecture.

SA.

it is all that programmer need to know to run and write machine language correctly.

* Application Binary interface (ABI) - instruction set plus the operating system interface used by programmers is (ABI)

* each lower level Hides details from above level called abstraction.

* One key interface bwn levels of abstn is ISA.

#] Volatile memory — Hold data when power prest.

Non volatile memory - Store data.

Primary memory — Volatile. — DRAM's.

Secondary memory - Non Volatile.

↓

magnetic disk's, flash memory

#] Performance

Performance metrics are many like.

Response time also called Execution time.

Exn time. - total time required for computer to

Throughput - Bandwidth - no of task completed per unit time.

To maximize performance we want to minimize the execution time.

of computer X.

$$\text{Performance}_X = \frac{1}{\text{Execution time}_X}$$

Performance of X > Performance of Y

$$\therefore \quad \text{Performance}_X > \text{Performance}_Y$$

$$\text{Execution time}_Y > \text{Execution time}_X.$$

X is n times faster than Y

Exn time of Y is n times that of X.

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Exn time}_Y}{\text{Exn time}_X} = n.$$

Q) Computer A run in 10s.
Computer B run in 15 sec.

How much A is faster than B

$$\frac{\text{Perf A}}{\text{Perf B}} = \frac{\text{Exn time B}}{\text{Exn time A}}$$

$$= \frac{15}{10}$$

Perf A = 1.5 ~~Perf~~ Perf B

we can also say, computer B is 1.5 times slower than that of A.

#1) improve performance or improve Exn time is to decrease execution time, increase performance.

wall clock time = exn time = response time.

2) Computer tries to optimise throughput.

#1) CPU Execution time.

also called CPU time - actual time CPU spends in performing a task.

- User CPU time — CPU time spend in program itself
  System CPU time - CPU time spent in OS.

System performance — time spent on system.

CPU performance — user time spent.

*) Computers are constructed using clock, that determines when event take place in Hardware

These discrete time interval is called as Clock cycle. → (Clocks, cycles, clock ticks, clock periods.)

Clock period = length of each clock cycle. ex: Clock cycle time, (250 picoseconds)

~~Clock period~~ = Clock rate (ex 4 GHz)

$$\text{Clock rate} = \frac{1}{\text{Clock period}}$$

* CPU exn time for a program = CPU clock cycle for a program × Clock cycle time

* $$\text{CPU exn time for program} = \frac{\text{CPU clock cycle for Programe}}{\text{clock rate}}$$

$$\text{Clock cycle time} = \frac{1}{\text{clock rate}}$$

CPU clock cycles = Instruction for program × Average clock cycle per instrn

Clock cycle per instruction = CPI = Average clock cycle per instruction.

CPI - average no. of clock cycles each instrn take to execute.

CPI is average of all instrn run in program.

Exn time = CPU clock cycle × clock cycle time

Exn time = Instrn for program × CPI × clock cycle tim

Exn time = Instrn count × CPI / clock rate

CPU clock cycle = $\sum_{i=1}^{n}$ CPI$_i$ × C$_i$

Time measured in sec.

$$\text{Time} = \frac{\text{Sec}}{\text{Program}} = \frac{\text{Instrns}}{\text{program}} \times \frac{\text{Clock cycle}}{\text{Instrns}} \times \frac{\text{Second}}{\text{clock cycle}}$$

Q7] A B

$\text{Exn time}_A = 10\text{ s}$.
Clock rate = 2 GHz.

$\text{Exn time}_B = 6\text{ s}$

$\frac{\text{Clock cycle } B}{\text{cycle } A} = 1.2$ Clock cycles A.

Clock rate B = ?.

$$E = \text{cpu clock cycles} \times \text{clock cycle time}$$

$$E = \frac{\text{CPU Clock cycles.}}{\text{Clock rate}}.$$

$$\text{Clock cycle} = \text{Ex Rate.}$$

$$\frac{\text{Clock cycle } B}{\text{Clock cycle } A} = 1.2 = \frac{(EXR)_B}{(EXR)_A}$$

$$1.2 = \frac{6.4 \times R}{10 \times 2\text{ GH.}}$$

$$R_B = 4\text{ GH clock.}$$

①)

A	B
Clock cycle time $t = 250$ ps.	cct $t = 500$ ps.
CPI $= 2$	CPI $= 1.2$

$$\frac{\text{Per A}}{\text{Per B}} = \frac{\text{Ex B}}{\text{Ex A}} = \frac{(\text{Instrn cont} \times \text{CPI} \times t)_B}{(\text{Instrn cont} \times \text{CPI} \times t)_A}$$

$$= \frac{1.2 \times 500}{2 \times 250}$$

$$\frac{\text{Perf A}}{\text{Perf B}} = 1.2$$

∴ Both prog run same no of Instructions.

A is 1.2 faster as of B.

---

①) compiler designing trying to decide btwn two code seq.

all facts are given.

CPI for each instrn class.

	A	B	C
CPI	1	2	3

instrn cont for two code sequence is

Code Seq	A	B	C
1	2	1	2
2	4	1	1

↓

instrn count for each instr class

which code seq exets more instrn.

Seq 1 : 2 + 1 + 2 = 5 instrns.

Seq 2 : 4 + 1 + 1 = 6 instrns

Seq 2 exetute more instrons.

c) which will be faster.

finding total no of clock cycles in each sequence.

$$\text{Clock cycles} = \sum (CPI_i \times C_i)$$

Total Clock cycles 1 = 2×1 + 1×2 + 2×3
= 10 cycles

To Tolc. Clock cycles 2 = 4×1 + 1×2 + 1×3
= 9 cycles.

as Seq 2 has less no of clock cycles.
it will exetute faster.

d) what is CPI for each sequence.

Average CPI = Clock cycles per instrs

$$CPI = \frac{\text{Clock cycle.}}{\text{total instrn}}$$

$$CPI_1 = \frac{10}{5} = 2.0$$

$$CPI_2 = \frac{9}{6} = 1.5$$

6) 400 MHz processor was used to execute benchmark prog with following instrn.

Calculate CPI.

→

Type	A	B	C	D
Count	450000	320000	150000	80000
clock cycle	1	2	2	2

$$CPI = \frac{450000 + 2\times320000 + 2\times150000 + 2\times80000}{450000 + 320000 + 320000 + 80000}$$

$$CPI = 1.55$$

Exn time = $n \times CPI \times t$ $\quad n = 1000000$

$$= \frac{1000000 \times 1.55 \times}{400 \times 10^6}$$

Exn time = 38.75×10^{-4} sec

#7 MIPS — million instruction per second.

It is a measurement of program or execution speed. based on no. of million instruction.

MIPS is inversely proportion to Ex^n time.

Bigger MIPS means faster Computer.

* $$MIPS = \frac{\text{Instruction Count}}{\text{Execution time} \times 10^6}$$

* $$MIPS = \frac{\text{Clock rate}}{CPI \times 10^6}$$

(Q) 2×10^6 instructions in code at 400MHz processor prog has 4 ports each port has seperate (P) and its %

(a1) Avg CPI

2) MIPS rate from above CPI

IT	CPI	Im %
A	1	60 %
B	2	18
C	4	12
D	8	10
		100 %

$$\text{Avg CPI} = \frac{1 \times 60 + 2 \times 18 + 4 \times 12 + 8 \times 10}{100}$$

$$\text{Avg CPI} = 2.24$$

$$\text{MIPS} = \frac{\text{clock rate}}{\text{CPI} \times 10^6}$$

$$= \frac{400 \times 10^6}{2.24 \times 10^6}$$

$$\text{MIPS} = 178.57$$

6) 3 processors P_1 P_2 P_3 executing same instrn set

P_1	Has 3 GH clock rate.	1.5 CPI
P_2	2.5 GH	1.0 CPI
P_3	4.0 GHz	2.2 CPI

→ 1) which processor has Higher instrn per sec rate.

$$\text{MIPS} = \frac{\text{clock rate}}{\text{CPI} \times 10^6}$$

$$\therefore \text{Instr/sec} = \frac{CR}{CPI}$$

1) $\frac{3}{1.5} = 2$

2) $\frac{2.5}{1} = 2.5$

3) $\frac{4}{2.2} = 1.8$

∴ P_2 Has Higher Instrn per sec.

b) processor exn time 10 sec find no of cycle & no of instrn.

→ $t_t = 10$

$$t_T = \frac{\text{clock cycle}}{\text{clock rate}}$$

clock cycle = $t_T \times$ clock rate

$$t_T = I_c \times CPI \times t$$

$$t_T = CC \times t.$$

$$t_T = \frac{CC}{CR}$$

$$CC = I_c \times CPI,$$

$$I_c = \frac{CC}{CPI}$$

$$I_{c_1} = \frac{t_T \times CR}{CPI}$$

$$I_{c_1} = \frac{10 \times 3 \times 10^9}{1.5} \qquad I_{c_2} = \frac{10 \times 2.5 \times 10^9}{1}$$

$$I_{c_3} = \frac{16 \times 4 \times 10^9}{2.2}$$

c) We are trying to reduce exn time by 30% which resu in CPI ↑ 20%.
what clock rate we must have.

→ ET ↓ 30%

∴ New ET = 7 sec.

CPI new = CPI X 1.2
=

$$ET = \frac{CC}{CR}$$

$$ET = \frac{Ic \times CPI}{CR}$$

$$CR = \frac{Ic \times CPI}{ET}$$

$$CR_1 = \frac{Ic_1 \times CPI \times 1.2}{7}$$

Some for 2 & 3.

Instruction Format General discussion.

Instruction set architecture.

Memory locn & address.

group of n bits called word,

n = word length.

many computer has word length 16 – 64 bits.

if word length of computer is 32 bits, a 32 bit word is stored. or 4 ASCII encoded character stored 8 bits each.

memory stored at address locn.

0 to $2^K - 1$ as address locn in memory.

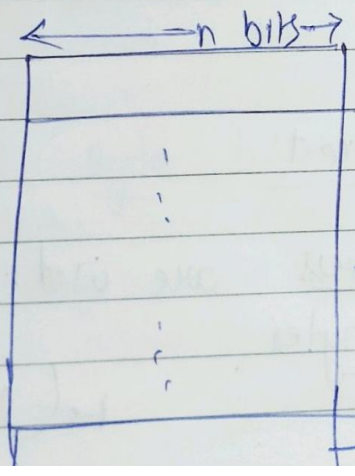
∴ for K bit addr in address. there are 2^K address line.

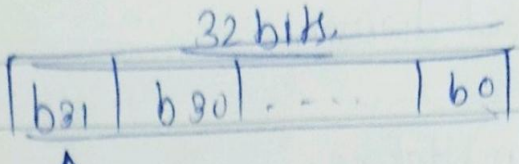
∴ 24 bit address generates 2^{24} locations.

$2^{24} = 2^4 \times 2^{20}$

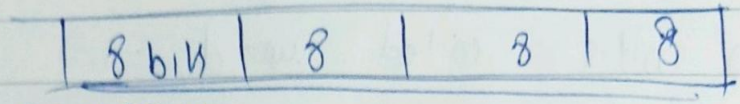
$= 16\ M$

$1M = 2^{20}$





b31 sign bit 0 +ve
1 -ve



8 bit ASCII character. 4 character 8 bits

Kilo K = 2^{10} (1024)

* Word length vary from 16-64 bits.

*) Byte addressable memory

ex if word length is 32 bit.

each word is located successive at
0, 4, 8, ...
with each word consisting of 4 bytes.

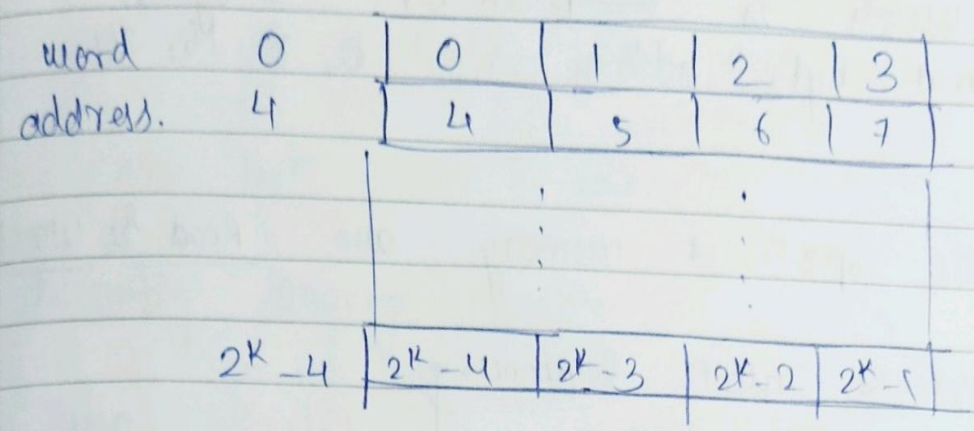
#) Big - Endian & Little - Endian Assignment

Big - endian assignment

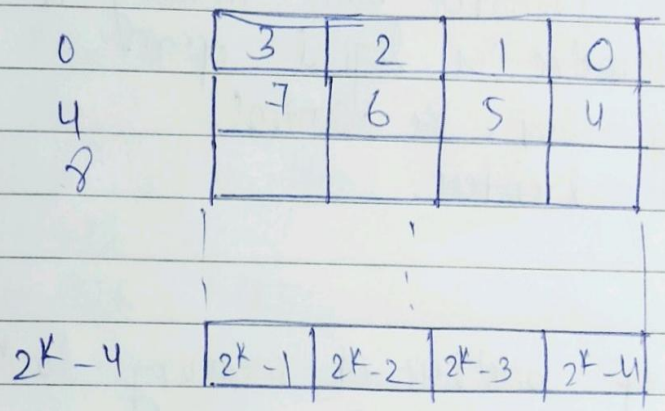
lower byte address are used. for most significant ad bytes

Little - endian address.

lower byte address used for least significant bytes
i.e. Right most.



Big - endian assignment



Little - endian assignment

This is Byte & Word addressing

v) Word alignment

if the word is 16 bit (2 bytes)
aligned byte address is of 2, 4, 6, ...

if words is ~~32 bit~~ 64-bit (8 bytes)
aligned byte address is 0, 8, 16, 24, ...

vi) 2 Basic operⁿ of memory are Read & Write.

vii) Instrn & Instrn Sequencing

4 tasks are prest a computer must have instruction to perform.

1) Data transfer bwn memory & processor
2) Arithmetic & logical operⁿ.
3) Prog seq. & control
4) I/O transfer.

Name rep address of memory locⁿ is,
LOC, PLACE, A, VAR2

Name of registers are R0, R1, R2, R3, R4, R5

Register in I/O System is DATAIN, OUTSTATUS

eg $R_2 \leftarrow [LOC]$

data at locn LOC address is transfers to R_2

eg $R_4 \leftarrow [R_2] + [R_3]$. Register Transfer Notation.

add contents at Register R_2, R_3 and store at R_4

LHS = locn RHS = value.

*) Assembly language Notation.

eg LOAD R_2, LOC loading data at memory location LOC into register R_2

Load is read from memory locn and ~~write~~ load in R_2

eg Add R_4, R_2, R_3 R_2, R_3 are source. R_4 destn.

Load - LD
Store - STR / ST

#] RISC and CISC instruction set:

These are two approaches for the design of instruction set in computers.

RISC - Reduced instruction set computers.
CISC - Complex instruction set computers.

: RISC

1) Based on the approach that each instruction should occupy exactly one word in the memory.
2) Operands need to execute an arithmetic or logic operation specified by instruction already present in registers.
3) In this approach the processor needs to process a sequence of instructions in 'pipelined' manner.
4) The activities are overlapped and total execution time is reduced.
5) The condition that each instruction must fit into the single word reduces complexity and no of instrns that can be used.

CISC.

1) This is alternative to RISC and make a more complex instructions that may span more than one word.
2) In this more complicated operations are used.
3) CISC was used prior to the RISC approach

#] RISC instruction Set

Characteristics.

1) Each instruction fits in a single word.
2) Load / Store architecture is used.
   - Memory operands are allowed with load & store instrn.

- All operands in arithmetic or logic operⁿ must either be present in registers, or one of the two operands given explicitly within instⁿ word.

* Load destⁿ, source.

Load processor_reg, memory_locn.

ex C = A + B

Load R_2, A
Load R_3, B
Add R_4, R_2, R_3.
Store R_4, C

Store → Source, destⁿ.

Add = destⁿ, src1, src2

Address.	Contents.
i	Load R_2, A
i+4	Load R_3, B
i+8	Add R_4, R_2, R_3
i+12.	Store R_4, C.

A
B
C

Data for the program.

Program Counter (PC) Holds the address of the next instruction to be executed.

To begin the prog exn the address of the first instruction Here (i) is placed in the (PC).

At each step PC) is increased by 4) to next instruction.

Executing a given instruction is a two-phase procedure.

First phase

IF Instruction fetch : instrn is fetched from memory loc whose address is in PC.

IR Instruction Register : instrn is placed in the IR in processor.

Second phase

IE Instruction Execute : instrn in IR is examined to determine which opern is to perform

FO fetch operands : fetching operands from memory or processor registers for exn.

Store : Storing result in destn location.

After IE phase is done, the PC contains address of next instrn & new IF phase begins.

#] Branching

Branching is to make loop.

(list of n numbers, rep as NUM1, NUM2, ... NUMn.

R_2 is used as counter to determine no of times loop is executed.

Subract R_2, R_2, #1 is done after each loop.

Conditional branch causes the branch if only specified condn is followed.

if condition is not satisfied PC incremented in normal way.

Here it will be

branch_if_[R_2] > 0. loop

store

Store : Storing result in destn location.

After IE phase is done, the PC contains address of next instrn & new IF phase begins.

#] Branching

Branching is to make loop.

(list of n numbers, rep as NUM1, NUM2, ... NUMn.

R2 is used as counter to determine no of times loop is executed.

Subtract R2, R2, #1 is done after each loop.

Conditional branch causes the branch if only specified condn is satisfied.

if condition is not satisfied PC incremented in normal way.

Here it will be

branch_N_[R2] >0. loop.

```
        Load    R2, N
        Clear   R3
Load.   Load    R5, [Num1]
        Add     R3, R3, R5
        Subtrat R2, R2, #1
        Branch if [R2] > 0   loop.
        store   R3, SUM.
```

eg Branch_if_[R4] > [R5] written as

Branch_greater_than R4, R5, LOOP.

BGT R4, R5, loop.

#] Addressing modes.

Addressing modes in RISC.

Name.	Syntax	Addressing fn.
Immediate	# value	operand = value.
Register	Ri	EA = Ri
Absolute	LOC	EA = LOC
Register indirect	(Ri)	EA = [Ri]
Indexed	X(Ri)	EA = [Ri] + X
Base with index	(Ri, Rj)	EA = [Ri] + [Rj].

The obsolute address can be given only in 16 bit value not more for a 32 bit word.

1-5 bits are used to store value of
0-15 bits are used for 16 bit no.

16-32 are used to sign extension.

*) Load operation only works on memory operands.

to load a number into an register we use

move R4, #NUM1

program for adding n numbers.

```
        Load    R2, N
        Clear   R3
        move    R4, #NUM1
Loop    Load    R5, (R4)
        Add     R3, R3, R5
        Add     R4, R4, #4.
        Subtrad R2, R2, #1
        Branch_if_[R2] >0 loop
        Store   R3, Sum.
```

indirect addressing used.

*) Indexed mode.

EA generated by adding a const value to content of Register

indexing also called indexed register.

$X(R_i)$

$EA = X + [R_i]$

X defines an 'offset' i.e. address displacement from this ~~location~~ to a location specified.

v) mnemonics ST R2 SUM,

ST is an operation code opcode.

assembler translates this code & computer recognises.

vi) Subroutine,

(Block of instruction is called - subroutine)

That is executed in ~~burn~~ loun of main program.

After a subroutine is called the program must return execution to after the instn that called the subroutine.

The PC saves the by the call instn to enable to correct return to calling program.

The way in which a computer makes it possible to call and return from subroutine is referred as subroutine linkage method.

simplest method is, save the return address in specific loc^n also called link register.

* Subroutine nesting

it is to have one subroutine call another one subroutine.

Additional Instructions.

1) Logical Instruction, AND, OR, & NOT.

eg And R_4, R_2, R_3.

Bitwise and bwn R_2, R_3 and store result in R_4.

And R_4, R_2, #value.

eg And R_2, R_2, #0xFF
Move R_3, #0x5A.
Branch if $[R_2]=[R_3]$ FOUND Z.

*) Shift & Rotate instructions.

*) Logical Shift.

Shifting left - LShiftL

Shifting right - LShiftR.

general form LShiftL Ri, Rj, count.

Shifting Rj bits to left by count. giv

*) Rotate oprn.

#) Multiplication and Division

Multiply Rk Ri, Rj.

$R_k \leftarrow [R_i] \times [R_j]$

2 n bit words after mutiplying generates as large as 2n bit words to store the both n bit words, a adjacent register R(k+1) is created.

Divide Rk, Ri, Rj

$R_j \leftarrow [R_j] / [R_i]$

Here quotient stored in R1 & remainder in R(i+1)

*] CISC Instn set.

Complex instruction set computers

CISC instn are not contained to load / store architecture, arithmetic & logical operations can be performed only on operands that are in processor registers.
Instructions do not necessarily need to be fit into single word.

CISC uses two address format unlike the RISC 3 address format.

eg operation destn, source.

Add B, A

B ← [A] + [B].

Here B is both source & destn.

eg C = A + B

Move C, B
Add C, A.

Sum of numbers in CISC architecture,

```
        move    R2, N
        clear   R3
        move    R4, #NUM1
loop    Add     R3, (R4)+      ← Autoincrement to next number
        Subtract R2, #1
        Branch>0 loop
        move    SUM, R3.
```

8] RISC and CISC Styles.

1) **RISC.**
   1) Simple addressing mode.
   2) All instrn fitting in single word.
   3) few instrn in instrn set.
   4) Arithmatic & logical operations can be performed only on opeands in processor registers.
   5) fast execution. - uses pipelining
   6) load / Store instrn dont allow direct transfer of from one memory to anoth

2) **CISC**
   1) Complex addressing mode.
   2) Instrn may span in mutiple words
   3) Instrn implemt complex tasks.
   4) Arithmatic & logical operations can be performed in memory operands as well as operand in prosor
   5) Transfer from one memory to other using simple move instrn.
   6) Complex instrn req for complex task.

Dot product of vectors.

$$\text{Dot product} = \sum_{i=0}^{n-1} A(i) * B(i)$$

```
        move    R2, #AVEC.
        move    R3, #BVEC.
        load    R4, N
        clear   R5
Loop:   load    R6, (R2)
        load    R7, (R3)
        Multiply R8, R6, R7
        Add     R5, R5, R8
        Add     R2, R2, #4
        Add     R3, R3, #4.
        Sub     R4, R4, #1
        Branch if (R4) > 0  loop.
        Store   R5, DOTPROD.
```

RISC style.

New CISC Style.

```
        move    R2, #AVEC.
        move    R3, #BVEC.
        load    R4, N
        clear   R5
loop    move    R6, (R2)+,
        multiply R6, (R3)+
        Move    R5, R6
        Subtract R4, #1
        Branch > 0  loop.
        Move    DOTPROD, R5
```

v) Instruction cycle : Subcycle.

Processor does three thing.

1) Fetch instrn : read from memory

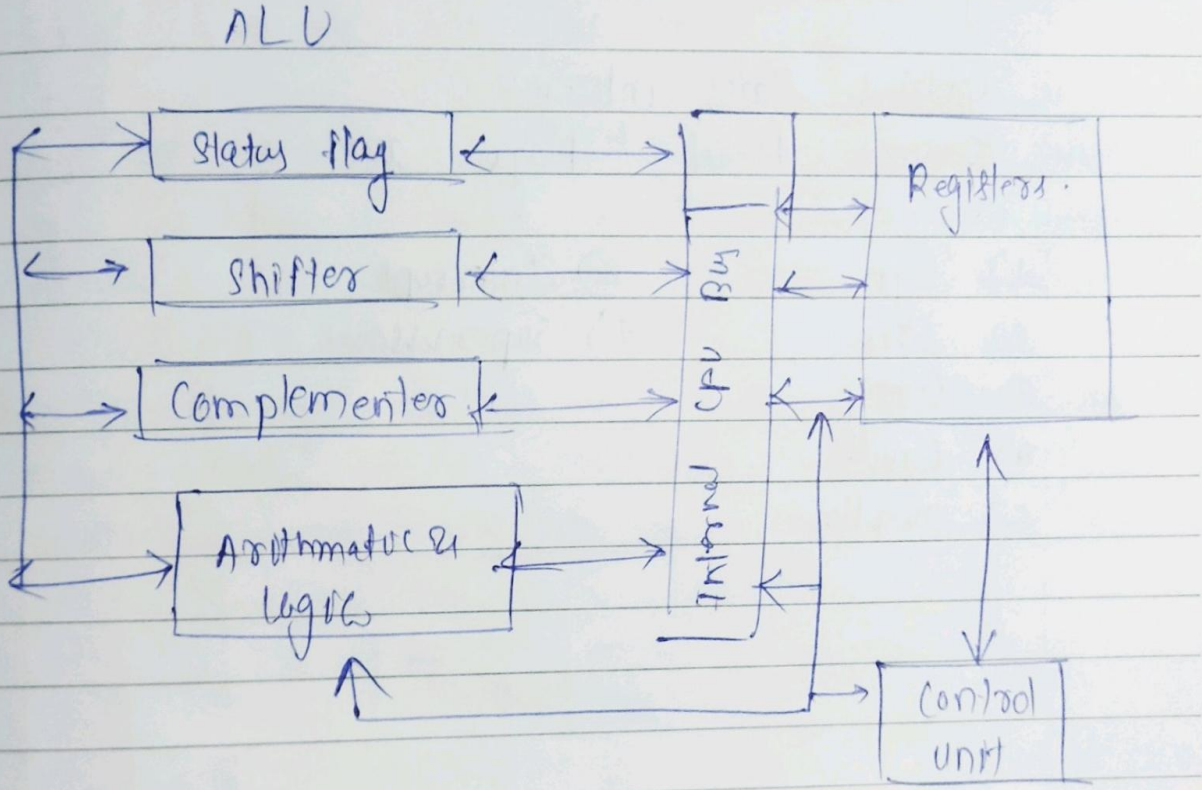
2) Intrepret instrn : instrn is decoded.

3) Fetch data : reading data from memory or I/O.

4) Procs data : perform arithmatic & logical oprn on data.

5) Write data : oprn req writing data to memory.

processor needs small internal memory for temporary storage.



Internal str^n of CPU.

v) Register organizn.

Register in procsor perform two roles.

user-Visible Register | Control & Status Reg

a) referenced by machine lang that procsor executes

b) registers employed to control the operation of procsor.

general purpose
Data register.
Address
Condition Codes (flags)

Program Counter (PC)
Instn register (IR)
Memory Address Register (MAR
Memory Buffer Register (MBR

vi) Program Status word.

Contains status informn.
Common Fields / Flags are.

1) Sign.
2) Zero.
3) Carry
4) Equal
5) Overflow
6) Interrupt
7) Supervisor.

Instruction cycle.

An instn cycle includes following stages.

1) Fetch: Read the next instn from memory
2) Execute: Interpret opcode & perform indicated operation
3) Interrupt: If interrupt is enabled and an interrupt has occured, save the current process state and service the interrupt

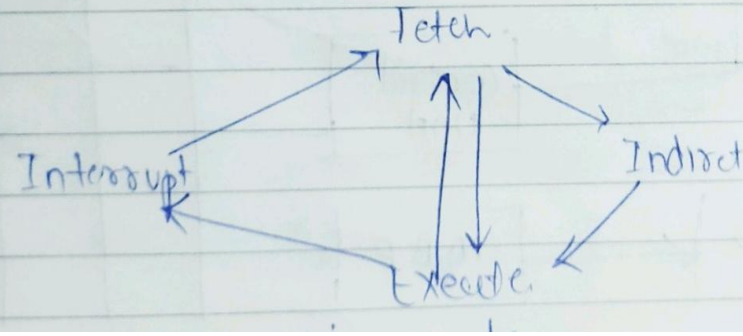
one additional state Indirect cycle.

while executing instn memory accss are required & if indirect addressing is used then additional memory accss are required.

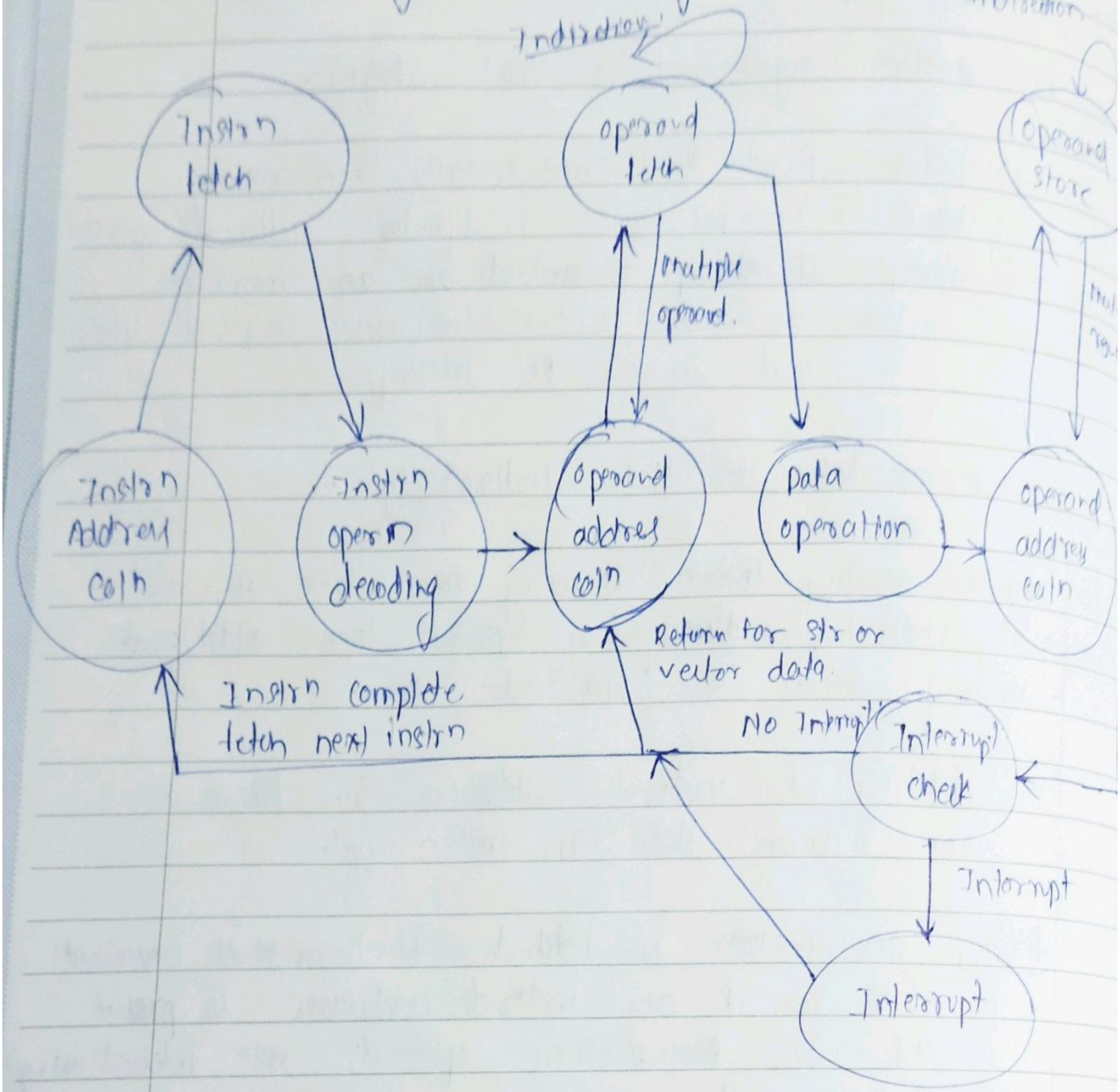
∴ fetching of indirect addresses is taken as one more state in instn cycle.

first, an instn is fetched & then it is examined to see if any indirect addresses is used. If so the req operands are fetched using Indirect cycle.

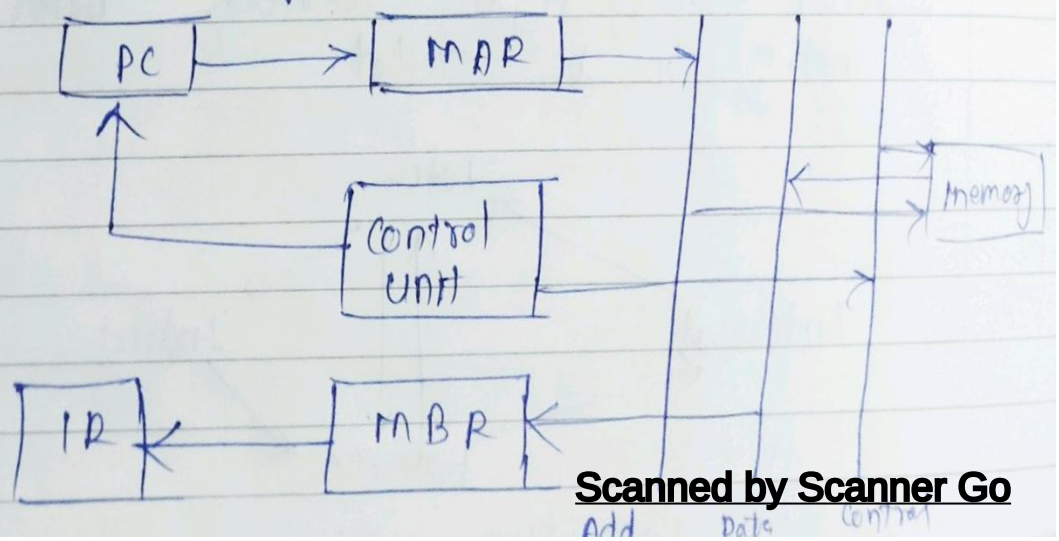
Then following execution, an interrupt may occur then it is serviced. before next instn to be exected.



*) Instrn cycle State diagram.



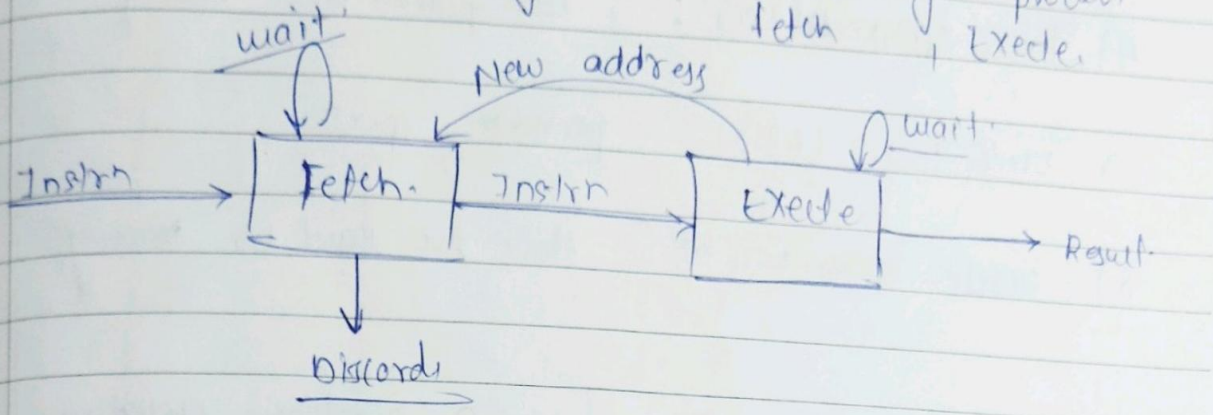
*) Data flow, fetch cycle.



Instrn pipelining

In pipelining new inputs are accepted at one end before previously accepted inputs appear at the output end.

Instrn has no. of stages, ex 2 stage process fetch + Execute.



Consider dividing the instrn processing in two stages, fetch instrn & execute instrn.

There are times in exen 2nd stage when main memory is not being used. This time can be used to fetch a new instrn while in parallel to exn of curr one.

Fetch stage fetches & give instrn to execute, while it is execting a new instrn is fetched & stored in buffer, when exn frees it is given to it.

This is called instrn prefetch.

II] More stages in the instrn cycle. helps in pipelining.

1) Fetch instrn (FI) : fetch next in buffer

2) Decode instrn (DI) : determine the opcode & operand specifiers.

3) Calculate operands (CO) : calculate address of operands

4) Fetch operands (FO) : fetch operand from memory

5) Exath Instrn (EI) : perform opern.

6) write operand (WO) : store the result in memory

Timing diagram for Instrn pipelining operation.

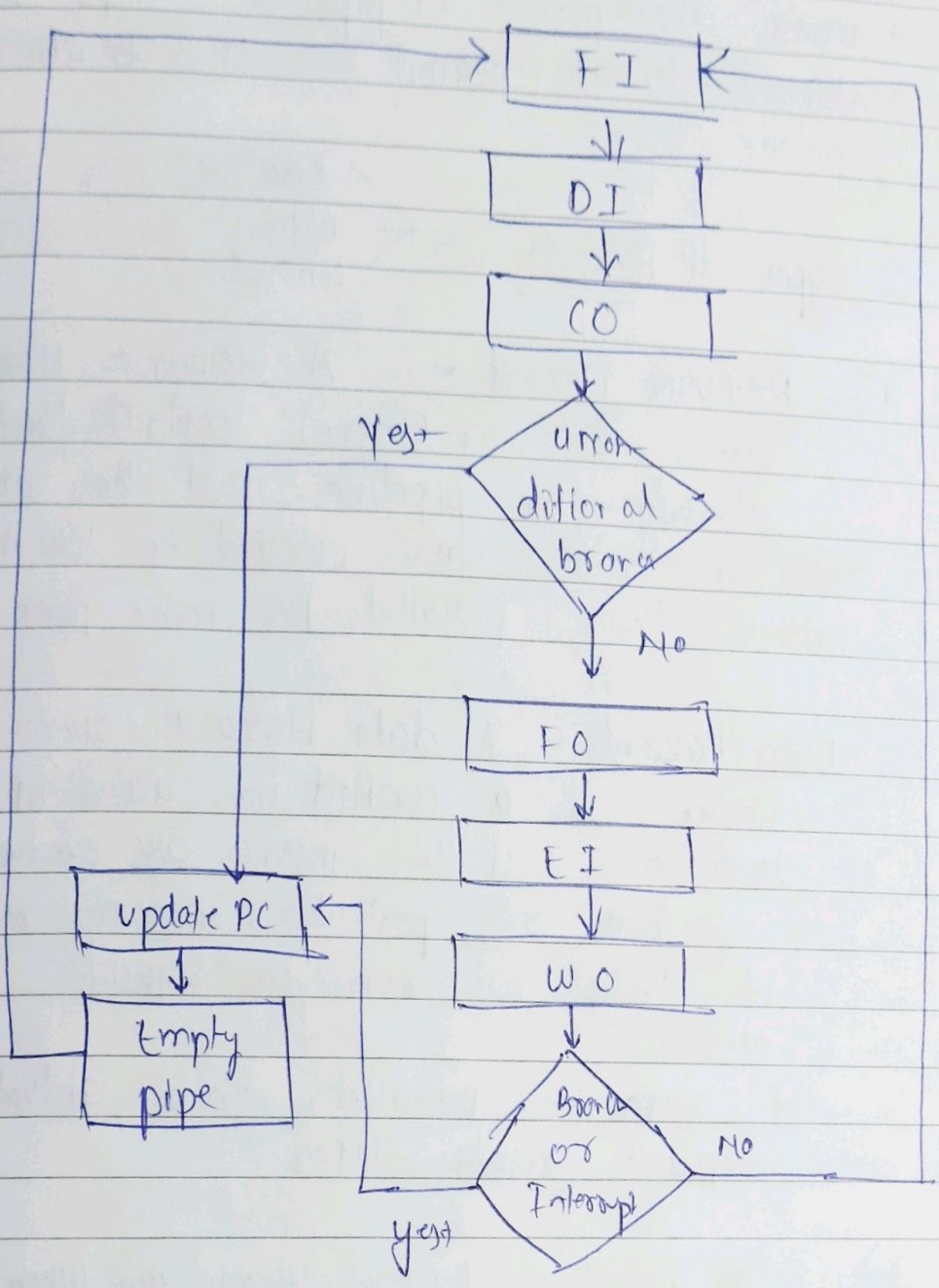
	1	2	3	4	5	6	7	8
Ins1	FI	DI	CO	FO	EI	WO		
Ins2		FI	DI	CO	FO	EI	WO	
Ins 3			FI	DI	CO	FO	EI	WO.
Ins 4								

The more stages, better pipelining but this is disturbed due to 2 point.

1) At each stage there is some overhead involving in moving data from buffer to buffer, this extends the time.

2) Amount of logic req to handle memory and registr. dependencies. increase time.

*) 6 stage instrn pipelining



Speedup factor = $\frac{nK}{K+(n-1)}$

K = no of stages
n = total instrn.

Pipelining Hazards.

it means some portion of pipeline steps because conditions do not permit. this is also called pipeline bubble.

3 types of Hazards.

- Resource
- Data
- control.

1) **Resource Hazard** - A resource Hazard occurs when two or (more) instrn that are already in pipeline need the same resource result is instrn are exected in serial ~~rat~~ rather than parallel. For some portion of pipeline.

2) **Data Hazard.** A data Hazard occur when there is a conflict in acces of operand location. If two instrn are exetud in sequence & both req particular memory or registr. If they are run in sequence, no problem occurs.
If run in parallel, then actual result & exped result differ.

3) **Control Hazard** - A control Hazard occ when the pipelining make the wrong decision on a branch prediction & brings instrn into pipelining that must subsequently be discarded.

Intel 80486 pipelining has 5 stages

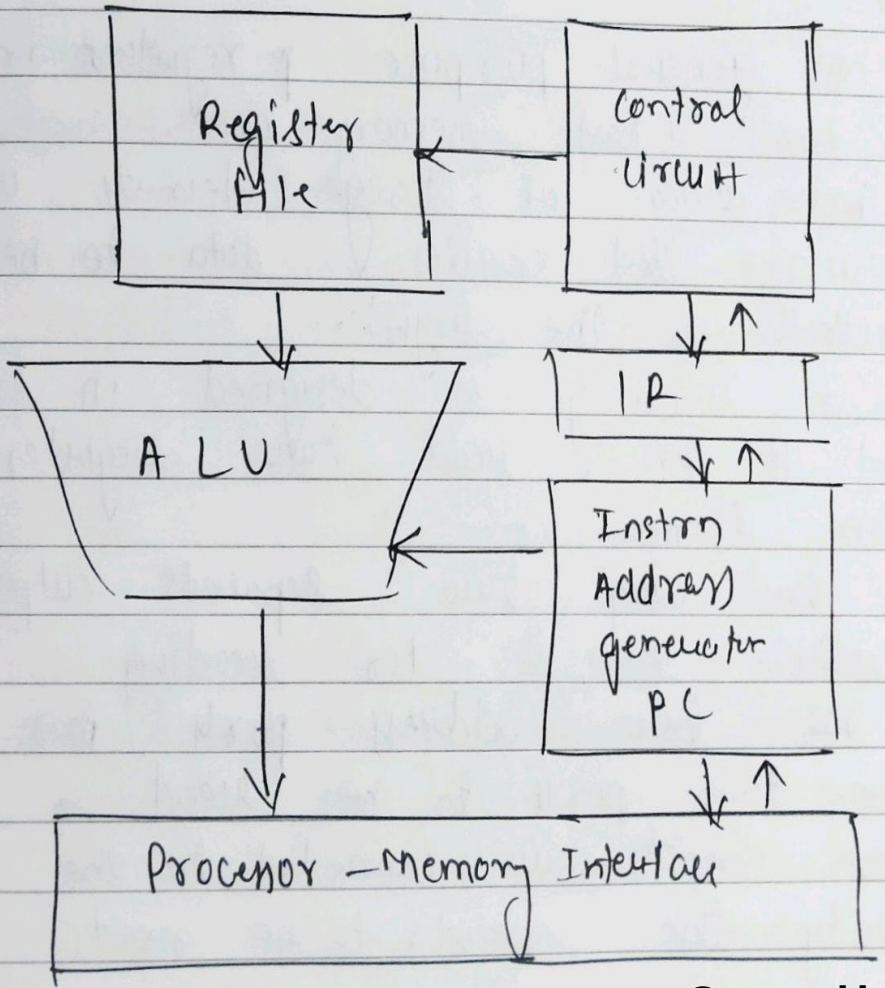
Fetch, Decode I,

161 172 175
Data path, Control signal, Hardwired.
microprogrammed control.
183

*) IR Hold the instrn until it is excuted. completely
oprn specified in instrn can be carried out by
performing one or more of following.

1) Read contet from memory locn & load into a processor register
2) Read data from one or more processor register
3) Perform arithmetic or logical oprn & store result into a processor register
4) Store data from processor register into memory locn

* Hardware componets of processor.



- Registers consists of edge triggered flip flops, hence data added at edge of clock
- Complex operⁿ in block can be broken down into simple steps. performed by subcircuits in stages
- Multi-stage structure is advantageous for pipelined operⁿ in RISC-style instr set.

Hardware components

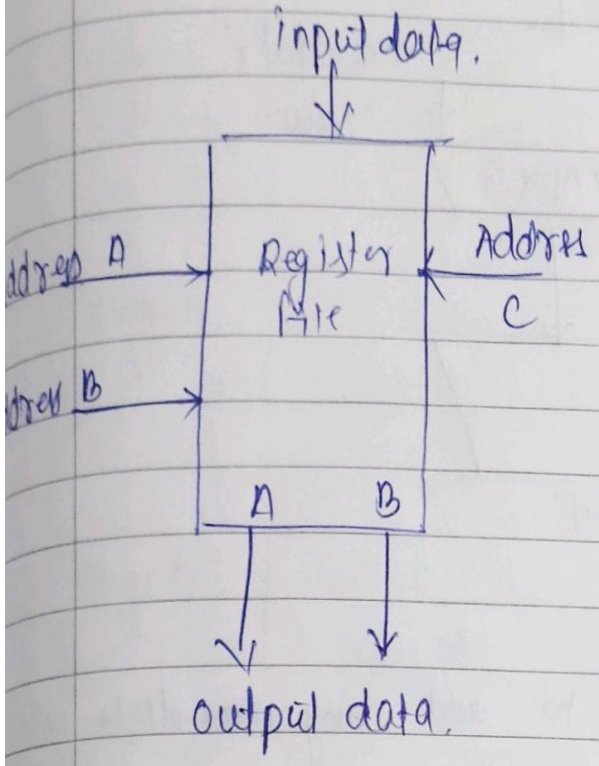
All instrn in RISC type style processor can be exeded in five-stages,
Hence Hardware may be organised in 5 stages.

1*) Register file.

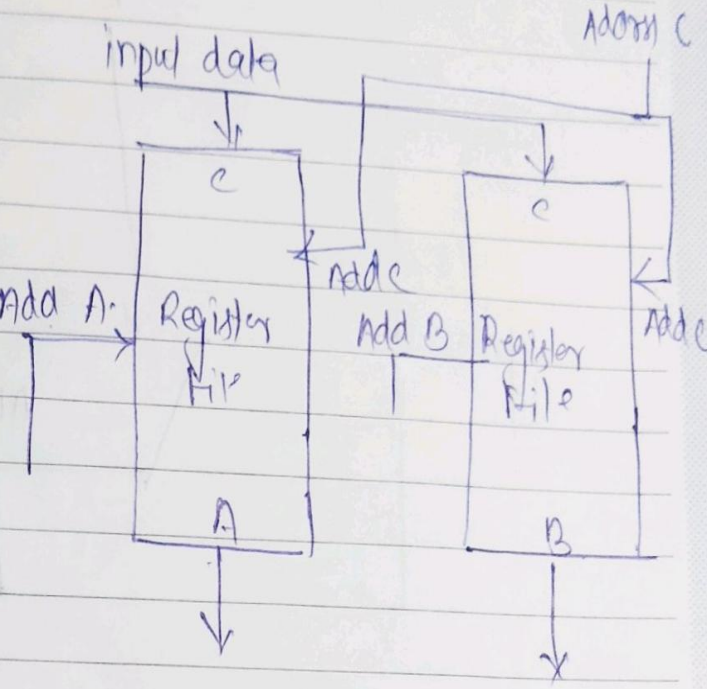
- It has general purpose registers in it, which is small & fast memory block.
- It has array of storage elements with access circuitry that enables data to be read or written in the registers.
- Access circuitry is designed in such a way that it can read two registers at same time.
- In this way two separate output ports are available A, B for reading
- It has two address ports for selecting the two ports to be read.
- These inputs are connected to the IR which specify the register to be read

- Register file has data input port C by which the data to be written in the register is entered.
- For this there is one address line connected to IR which tells where to write data.

This type of str is called as, dual-ported. as it has 2 output to read.



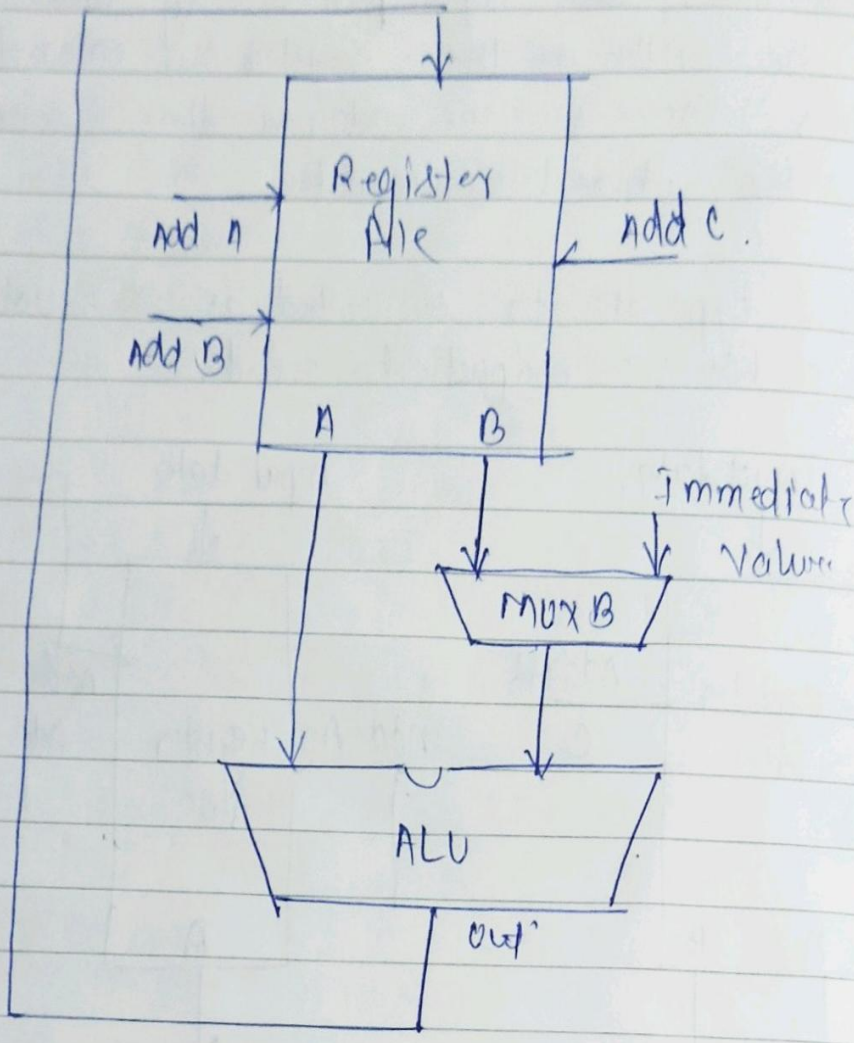
Single memory block



Two memory blocks

2) ALU.

ALU is to perform arithmatic & logical operⁿ. ALU take, output of Register file A & B as input to the ALU unit & output of ALU unit, an input data C is entered in to the Register.

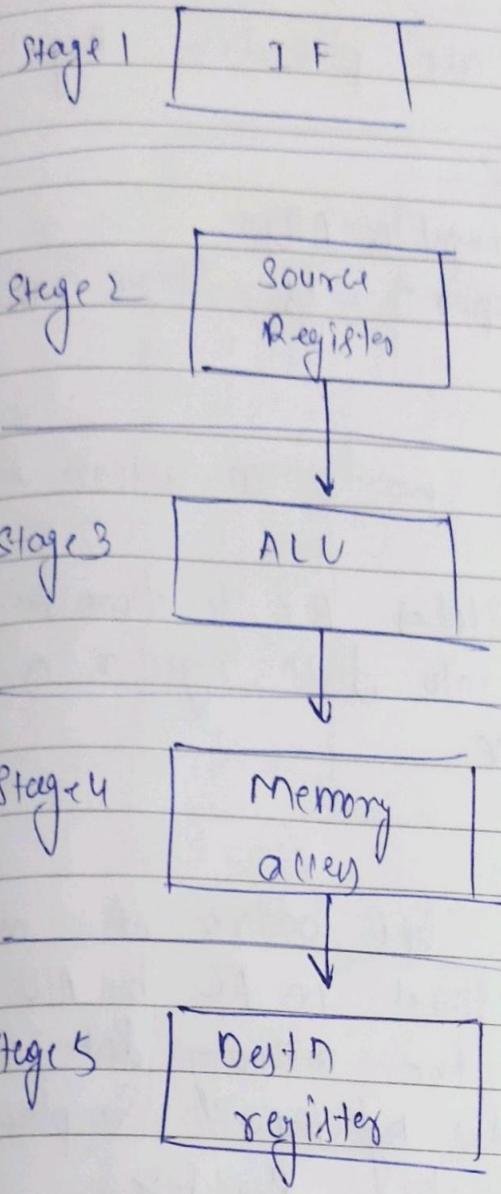


Immediate value is used to add any immediate value to the port output port B.
if B has a value then i.e. immediate then that value is taken.

#] Datapath.

Instrn processing is two stage / phase, fetch phase & exection phase

The section that fetches instrn is also reponsible to decode and generate control signal that causes appropriate action to be taken



instrn fetched in IR Stage 1 & place in IR.
It is decoded and source register is read in Stg 2

Info in IR is used to generate control signal in all steps.

It is necessary to have inter stage register to store info intermediately.

The Stage 2 to 5 is called as datapath.

- Data from registers are placed in Register RA & RB.

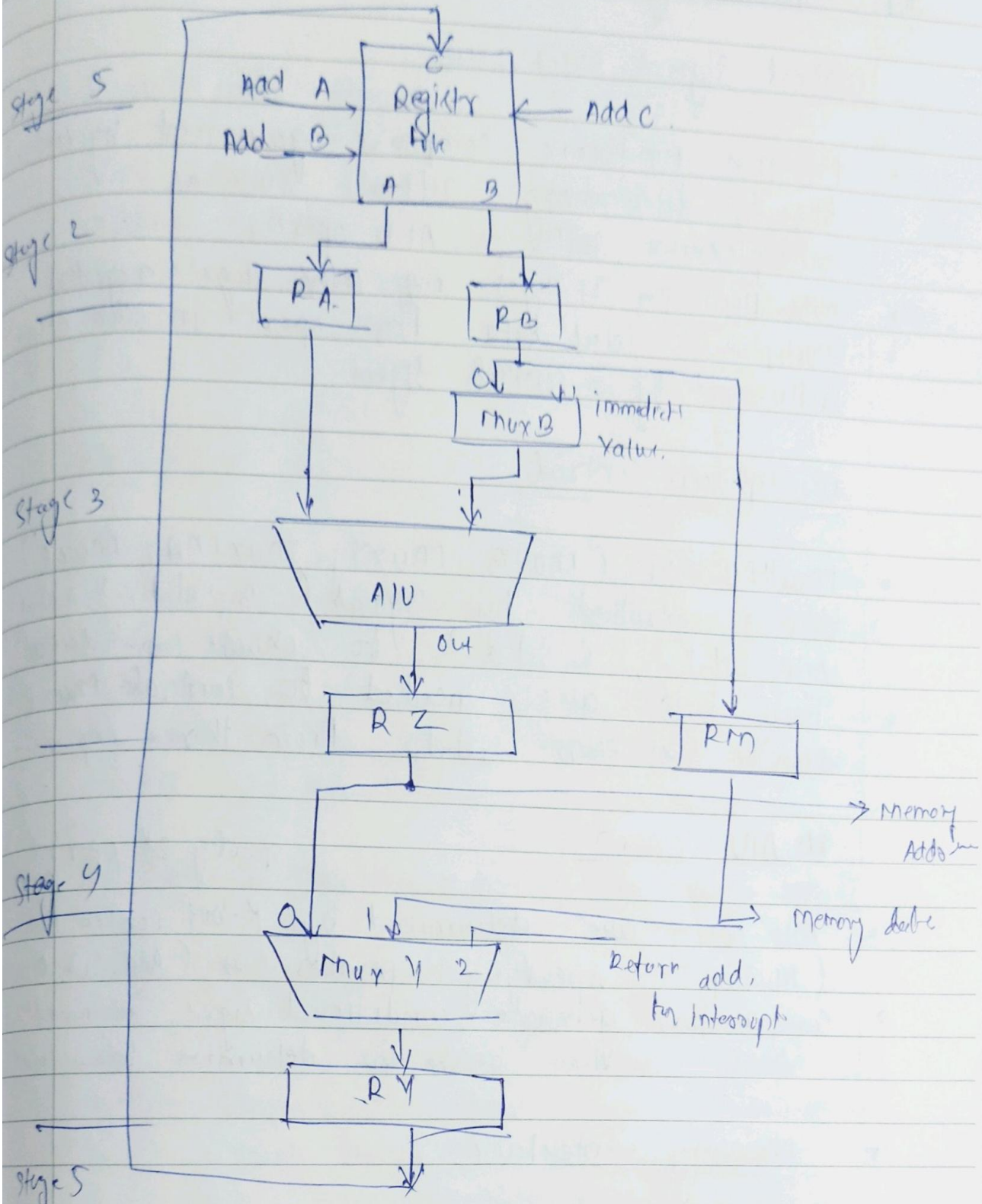
- This RA & RB gives input to ALU. ALU solves the opern & store result in RZ.

- for action such as Add no action is taken in Stage 4
At this the mbr4 selects RZ & transfer data to RY & loaded into data register in Stage 5 to Register file

- for load / store instrn eff addrs of memory operand is computed & stored in RZ by ALU
In case of load instrn the memory data read from memory is selected by mar4 & placed in reg RY to transfer to Reg file.
The data to be stored is placed in Reg RM from RB

- Interrupt processing they return address to be saved. another Reg RA is use for this purpose.
for this another I/p input called for return address. it can be sent to Register file thro RY.

Datapath in processor.



Control Signal

control signal overview

- Processor hardware component governed by control signal, determining actions such as multiplexer selⁿ, ALU opern.
- Data flow in 4 stages req inter-stage registers.
- Multiplexer select data for opern in each stage influenced by control signal.

multiplexer control

- Multiplexers (muxB, muxY, muxMA, muxC) are controlled by signals (B_Select, Y_select, MA_Select, C_Select) to choose input data.
- Two bits are needed to control muxC & muxY as each selects from three inputs.

ALU opern

specify 2^k oprns.

- ALU opern are determined by K-bit control code (ALU_op) specifying operations as (Add, Sub, AND, OR)
- Comparators generate conditional signal, examined during condition branch to determine branch cond^n.

Memory - Interface

- MEM_read & MEM_write are used to initiate the memory opern.
- IR_enable is used to allowing loading new instrn into registers.

- MIC signal is used when requested oprn is excuted.

- Instrn add generator

- INC_Selct is used to determine the value to be added to PC (constal 4 or branch offset)
- PC_Selct is select signal Updates address or content of register.

Hardwired Control

There are 2 ways for generating control signals. Hard wired control, & microprogrammed control.

Step counter is used keep track of the proces of exection.

- Setting of control signal is depends on

1) Content of Step counter.
2) Content of instrn Register.
3) Result of compute or comparison oprn
4) External input signals such as interrupt req.

- The instrn decoder interprets op code, & addressing mode into setting corres IMS signal
- Step counter increments at every clock cycle, indicates the current exection step ($T_1 - T_5$)

control signal generator generates necessary control signal based on inputs

ex) New instrn is fetched from memory identified by signal T_1.
MA-select signal set to 1,
Selecting the PC as memory address source &
MEMread is activated.

IR is loaded with data from memory
PC incremented by 4 preparing for next instrn fetch

*) Datapath control signal

- Instrn decoder interprets IR contents & source register contents become available at A & B
- Other storage registers are also used. like RA RB etc.

2] Microprogrammed Control

Microprogrammed control is an alternative to Hardwired approach

Control signals are determined by program stored in a special memory called microprogram memory.

- Each step task has its own microprogram repp. by series of bits, called microinstrn

- microinstrn are step by step instrn to conduct
- Microprogram memory
  - microprogram are stored in small & fast memory called microprogrm memory
  - This memory has all instrns needed for diff steps in various tasks
  - Execution of prgm depend the microinstrn
  - micro instrn addrs generator helps find the proper microinstrn in memory.
  - microroutine is the sequence of exeding the instrn.
  - This approach is slower than Hardwired approch. It is more flexible than Hardwired.

RISC-V uses Hardwired control

Control unit operations. ch 20, 21 Stallings

*) Micro operations.

Each instrn cycle is made up of smaller units as fetch, indirect, execute, interrupt. These are the micro-operations.

each these steps involve procssor registers. each instrn is exected in an instrn cyle. & each cycle has subcycles as fetch, indirect etc.

*) Fetch Cycle

It is at the start of instrn to be exected.
It has 4 Registers.

1) MAR Memory add. reg. (connected to address lines)
2) MBR (Memory Buffer reg.) (connected to data lines)
3) Program Counter (PC) (Hold the address of next instr to be fetched)
4) Instrn Register (IR) (Hold the last instrn fetched)

They work in 3 steps.

T_1 : MAR ← PC
T_2 : MBR ← Memory
PC ← (PC) + I.
T_3 : IR ← (MBR)

a) Add of next instrn to be executed is in PC,
step 1 is to move the content of PC to MAR
step 2 the address in MAR is put on ~~data~~ address lines, & instrn is fetched in MBR by Read. Instrn

- The content in memory appears on data lines and put on MBR
- Meanwhile the PC is incremented for next instr to execute.
- Third step is move data from MBR to IR & free up the MBR.

3] Indirect cycle.

If instrn specifies an indirect address then an indirect cyle runs before execute cycle

This indirect add is transferred to MAR, in T_1
Then memory from memory is send on MBR in T_2

Now in T_3 the MBR address is stored in IR.

T_1	MAR	←	IR(Address)
T_2	MBR	←	Memory
T_3	IR(Add)	←	MBR(Add)

*) Interrupt cycle.

- At end of execute cycle, a check is made to see if any intr. occurred,

it has 3 steps.

T_1 : MBR ← (PC) } Saving curr. state of prog.
T_2 : MAR ← save Add. ↓ start of interrup
PC ← Routine Add.
T_3 : Memory ← MBR

*) Execute cycle

EC can vary for instrn to instrn, many varities

So control unit examines the opcode of instrn & generate the sequence of micro-operns
This is instrn decoding.

eg ADD R1, X

add content of loc X into Register R1

T_1 : MAR ← IR (Addr)
T_2 : MBR ← Memory
T_3 : R_1 ← (R1) + (MBR)

#) Instruction cycle code.

(Instrn cycle code) H is a 2bit register which decides, in which sub cycle the instrn is,

00 Fetch.
01 Indirect
10 Execute
11

Control of processor

Functional requirements of the control unit are the fns that the CU must perform.

3 step process leads to characterize of of CU.

1) Define basic elements of processor
2) Describe micro operations
3) Determine fns of m CU.

1) Basic functional elements

ALU, Register, data path, CU.

2) micro operns.

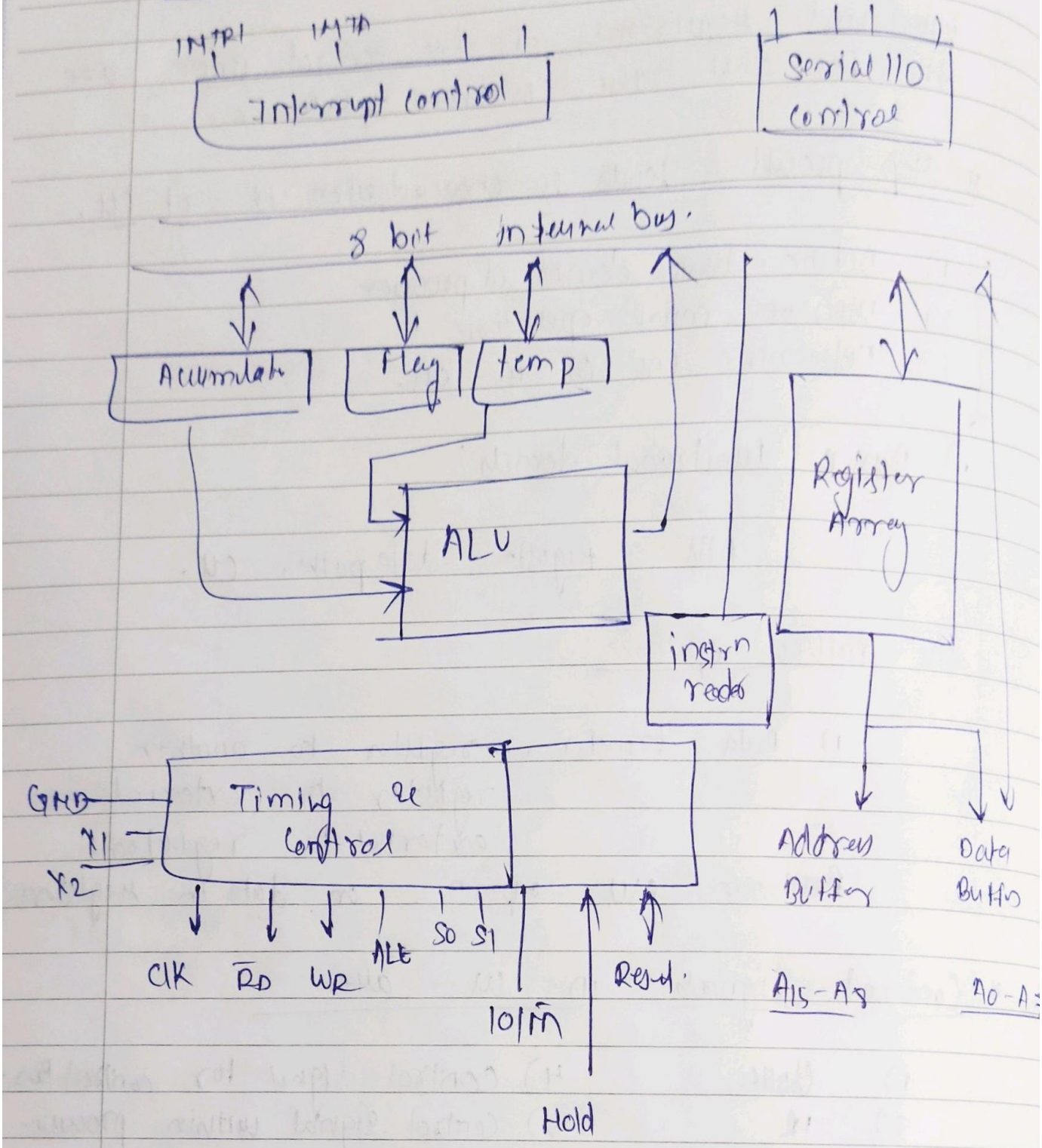
1) Data transfer, register to another register to external, external to register.

Perform ALU opern on data in registers

* Control signals in CU are.

1) Clock,
2) IR
3) Flags
4) Control signal for control Bus.
5) Control signal within processor
6) Control signal to control Bus.

*) Intel 8085 organizn.



Data path, control signal for RISC-V instrn
ADD, LOAD, etc

Ch-4 Pattersen 4.1 to 4.4

*) Introdn.

core RISC-V imp instrn set.

1) memory reference instrn load doubleword (ld)
store doubleword (sd)

2) arithmatic logic oprn add, sub, and, or.

3) conditional branch instrn branch if equal (beq)

*) Every instrn has two basic steps.

1) Send the PC to the memory that has code & fetch the instrn.

2) Read one or two registers using fields of the instrn to select the registers.

*) After this there are 3 instruction classes.
(memory-reference, Arithmetic-logic, branches)

Studying the datapath flow done previously for better grip

#) RISC - V instrn set architecture.

DO Chapter Patterson 4.1 — 4.4,