Sequencing Logic control me

→ Computer is Six level machine

(1) Level 5 — problem Oriented language level (High level language)

→ Applications were working only in one machine, that app is called "Desktop App"

↓ Translation (compiler)

(2) Level 4 — Assembly language level

↓ Translation (Assembler)

(3) Level 3 — Operating System Machine level

↓ partial Interpretation (O.S.)

(4) Level 2 — Instruction set Architecture level (ISA)

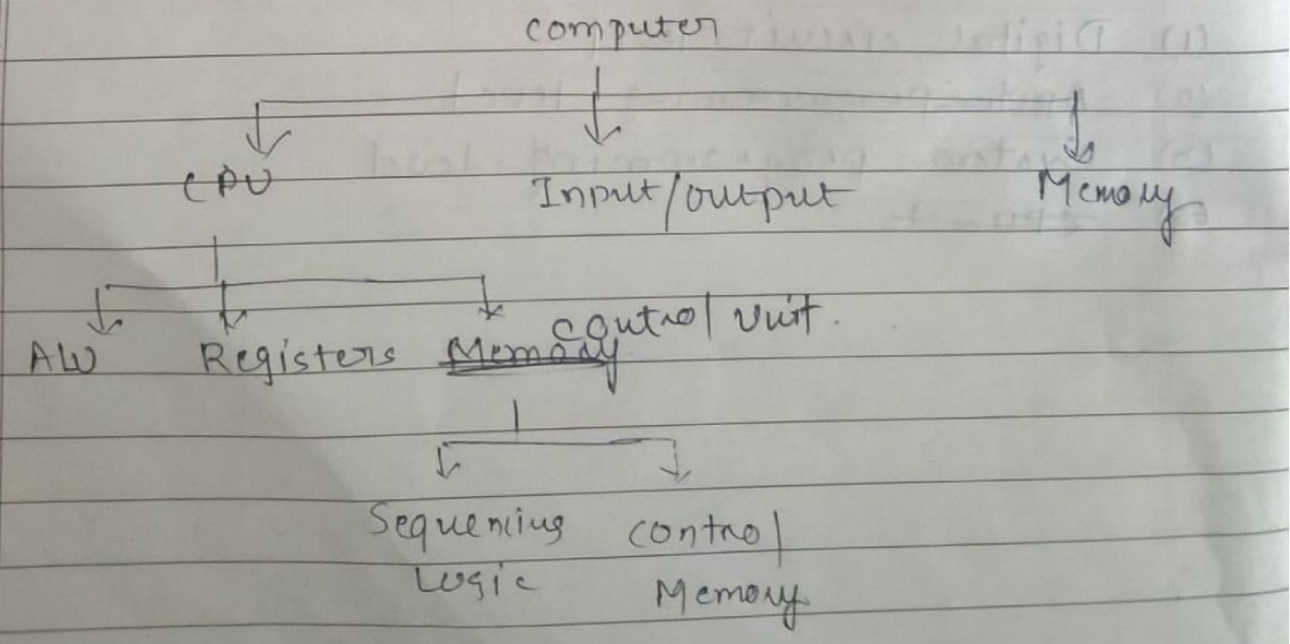
↓ Interpretation (Micro program)
↓ Direct execution (Hard wired)

(5) Level 1 — Micro Architecture level (MAL)

↓ Hardware

(6) Level 0 — Digital Logic level

⇒



⇒ Computer functions -

→ Data Movement
→ Data control

⇒ The Basic Computer Model —

⇒ The von-Neumann Model —

→ " Fetch-decode Execution cycle "

⇒ Instruction set Architecture —

C code ⟷ (compiler) Assembly code → (Assembler) 0... 1... Machine code

→ compiler makes code machine independent.

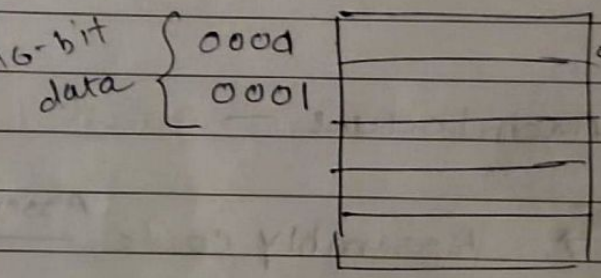
→ Instruction Set is the interface b/w hardware & software

→ Instruction set Design -
(i) Central part of any system design.
(ii) Allows abstraction, independence.

→ Machines with new ISA should allow old instruction to be execute.

→ 64-bit machine means 64-bits data will transfer through data bus at a

→ for ex. we have 16 bit machine. Then 8 bits data will be stored in one address and other 8 bits data will be stored in ~~other~~ 2nd address.



→ Memory Model -
→ Exceptional conditions -
→ Opcodes (Machine Language)

→ MAL which is the set of processor design techniques used to implement the instruction set.

⇒ Elements of an Instruction –

- Operation code (Opcode)
- Source Oprand reference
- Result Oprand reference
- Next Instruction reference

⇒ Instruction Types –

- Data processing.
- Data storage
- Data movement
- Program flow control
  (i) conditional and unconditional branches
  (ii) call and return.

⇒ ISA Architecture types –

(1) Stack architecture → Implicitly on the top of stack.
(2) Accumulator " – → one Operand is implicitly the accumulator
(3) General purpose registers architecture –

Register Memory	Load store (A memory → registe
Load R1, A	Load R1, A
Add R1, B	Load R2, B
Store C, R1	ADD R3, R2, R1
	STORE C, R3.

→ Recently we are using Load store Architecture, because we store data in Registers. So, Execution tim is less.

⇒ GPR Architecture –

(1) Register - Register (0 Memory + 3 Reg = Total 3)

Example :- ALPHA, MIPS, SPARC, Power-Pc

(2) Register- Memory (1 Memory + 1 Reg = Total 1).

example :- Intel 80x86, Motorola 68000.

(3) Memory - Memory (3 Memory + 0 Reg = Total 3)

Example: VAX. (find why VAX is still in market ?)

⇒ No. of Oprands –

Y = (a-b) / (c + (d * e)).

(1) Three Oprands instruction –

```
Sub   y, a, b.   ( 3-oprands)
Mul   t, d, e.
Add   t, t, c.
Div   y, y, t.
```

(2) Two oprands instruction –

```
Sub   a, b,
MUL   d, e.
Add   c, d.
Div   a, c.
```

⇒ One Operand instructions –

→ In fewer operands, the fetch-execution cycle is faster.

→ From 4 GB memory 2^{32} bytes can be accessed and 2^{30} words (32 bytes) can be accessed.

⇒ Big Endian – When MSB is at least address. e.g. MIPS.

⇒ Little Endian – When LSB is at least address.

→ It's a reversing of bytes not a number.

⇒ VAX and x86 ~~are~~ have CISC Instruction set.

⇒ PowerPC processors RISC Instruction set.

CISC: MOV AX, 10
MOV BX, 5
MULT AX, BX

RISC: MOV Ax, 0.
MOV bx, 10
MOV CX, 5.
Begin: Add ax, bx
Loop Begin

→ Here, for cisk (2moves x 1 cycle) + (1mult x 30 cy
= 32 cycles.

→ for RISC. (3 moves x 1 cycle) + (5 adds x 1 cycle)
+ (1 loop x 5 cycle)
= 13 cycles

⇒ MIPS (Microprocessor without Interlocked pipeline stages).

→ It has RISC ISA (Big Endinan)
→ MIPS has 3 Oprands instruction.
↓ 2 source → 1 destination.

but for A = B + C + D + E.

Add E, B, C.
Add G, D, E.
Add A, E, G.

→ F = (G + H) - (I + J)

	OR	
Add K, G, H		Add G, G, H.
Add L, I, J.		Add I, I, J.
SUB F, K, L.		SUB F, G, I

→ MIPS has a 32 x 32 bit register file.

→ If 32-bit processor, then 32-bits will move from data-bus.

→ In MIPS, there is only $t0, $t1 ... $t9 temporary values to store values.

→ and $s0, $s1 ... $s7 for saved variables.

⇒ MIPS operand - Registers —

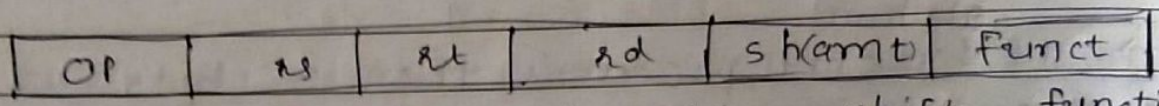
Name	Register No.	Usage
$zero	0	the constant value 0
$v0 - $v1	2-3	Values of results and expression evaluation.
$a0 - $a3	4-7	arguments
$t0 - $t7	8-15	temporaries
$s0 - $s7	16-23	Saved
$t8 - $t9	24-35	more temporaries
$gp	28	global pointer
$sp	29	stack pointer.
$fp	30	frame pointer.
$ra	31	return address.

Note: NO SUBI instruction is available in MIPS Micro-processor.

→ Register 1 ($at) is reserved for the assembler, 26-27 called $k0 - $k1 for O.S.

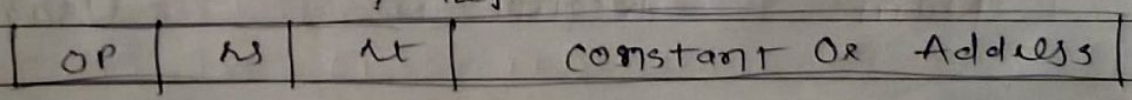
→ Require 5-bits to select one register.

⇒ R-type instruction format —



op	rs	rt	rd	shamt	funct
Opcode Operation	first register source operand	second register source operand	destination register operand	shift amount 00000 for now	function field selects variant of operation extends opcode
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

⇒ I-Type —



op	rs	rt	constant or Address
6 bits	5 bits	5 bits	16 bits

e.g. $c = c + 4$

$\$s_3 = \$s_3 + 4$

ADDI $\$s_3, \$s_3, 4$

↓

(Immediate)

⇒ Load instruction :-

e.g A = A + Array[100].

LW $t0, 100($s2)

base Address.

Loading

Add $s1, $s3, $t0

→ G = H + A[8]

LW $t0, 32($s3). (∵ each word occupies 4 bytes).

→ A[12] = H + A[8] ;

LW $t0, 32($s3)
LW $t1, 32($s4)
Add $t0, $t0, $t1

LW $t0, 32($s3)
Add $t1, $t1, $t0
SW 48($s4), $t1

LW $t0, 32($s1)
Add $t1, $t0, $s3 || $s3 = H
SW $t1, 48($s1).

where, $s1 = Base Add. of A.
$s2 = H content

$t0, $t1 = temporary registers

⇒ $s_0 = 0$ —

⇒ ~~LOO~~ $t0, $s0
nor $t1, $t0
~~Add~~ $t1, $t1, $t0
sc0 $t1, $s0.

OR and $s0, $s0, $zero

OR Sub $s0, $s0, $s0

OR addi $s0, $zero, 0 (constant).

⇒ MIPS conditional instructions —

Branch to a labeled instn if a condition is True.
Otherwise continue sequencially.

e.g. Beq rs, rt, L1
IF (rs == rt)
(If this is true then only it will go to L1 location otherwise it will go to next instn.

eg. IF (i==j) f=g+h.
else f=g-h.

Beq
~~If~~ ($s3 == $s4)
Add $s0, $s1, $s2 // f=g+h.
~~Else~~ Bne
Sub $s0, $s1, $s2 // f=g-h

Both eq.

(1) Beq $s3, $s4, LAdd
Sub $s0, $s1, $s2.

LAdd : $s0, $s1, $s2.

(2) Bne $s3, $s4, LSub
Add $s0, $s1, $s2
J Exit // to jump to LSub.
LSub: $s0, $s1, $s2
Exit: ....

conditional instructions with loop ——

Loop: g = g + A[i]
i = i + j
if (i != h) goto Loop.

: add $t1, $s6, $s6 // $t_1 = i + i = 2i$
add $t1, $t1, $t1 // $t_1 = 2i + 2i = 4i$

add $t1, $t1, $s5 // $s5 = Base Add of A.
$t1 = A[i]

lw $t0, 0($t1)
add $s1, $s1, $t0 // g = g + A[i]

add $s3, $s3, $s4 // i = i + j

bne $s3, $s2, Loop.

Que while (save[i] == k)
i = i + j ;

i ⇐ $s3, j = $s4, k = $s5. base Add = $s6.

while: Add $t1, $s3, $s3 // $t1 = i+i = 2i
Add $t1, $t1, $t1 // $t1 = 4i
Add $t1, $t1, $s6
LW $t0, 0($t1) // $t0 = save[i]

beq $t0, $s5, LAdd
Exit...

LAdd: $s3, $s3, $s4
j while // jump while loop

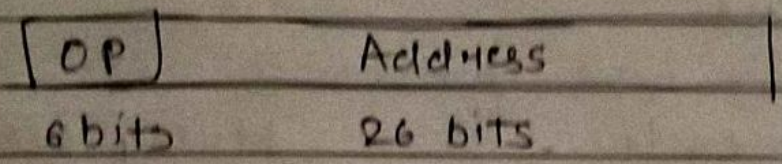
Que 1 while (save[i] == k), i+ = 1;

Que 2 switch (k) {
case 0: f = i + j ; break;
case 1: f = g + h ; break;
case 2: f = g - h ; break;
case 3: f = i - j ; break;
}

f = $s0, i = $s1, j = $s2, g = $s3, h = $s4,
k = $s5.

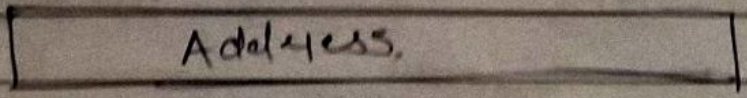
For branch instruction format we will use I-type of instruction format.

Jump (j) Addressing —



26 bits address is concated with upper bits

jal Addressing.



In I-type base Add. is for lw, sw and immediate Addressing mode is for Addi

Micro Architecture level.

GOAL: Data path and control Unit.

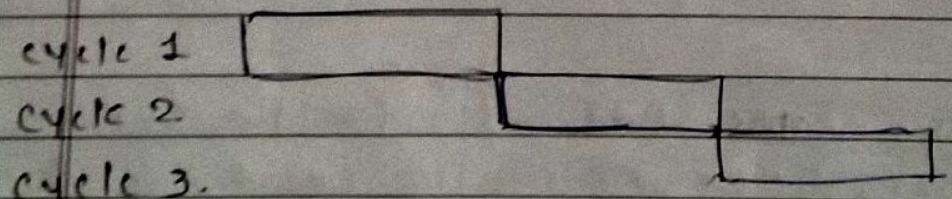
⇒ Processon implementation style —

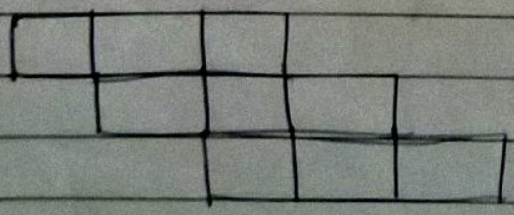
(1) Single cycle.
(2) Multy cycle.

	Single cycle	Multy cycle
	←	
lw	1 cycle	4 cycle
Add	1 cycle	3 cycle
sw	1 cycle	2 cycle
	Cycle time - 20ns	Cycle time - 5ns

↳ max time require that will be cycle time

⇒ Unpipelined. —



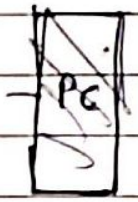


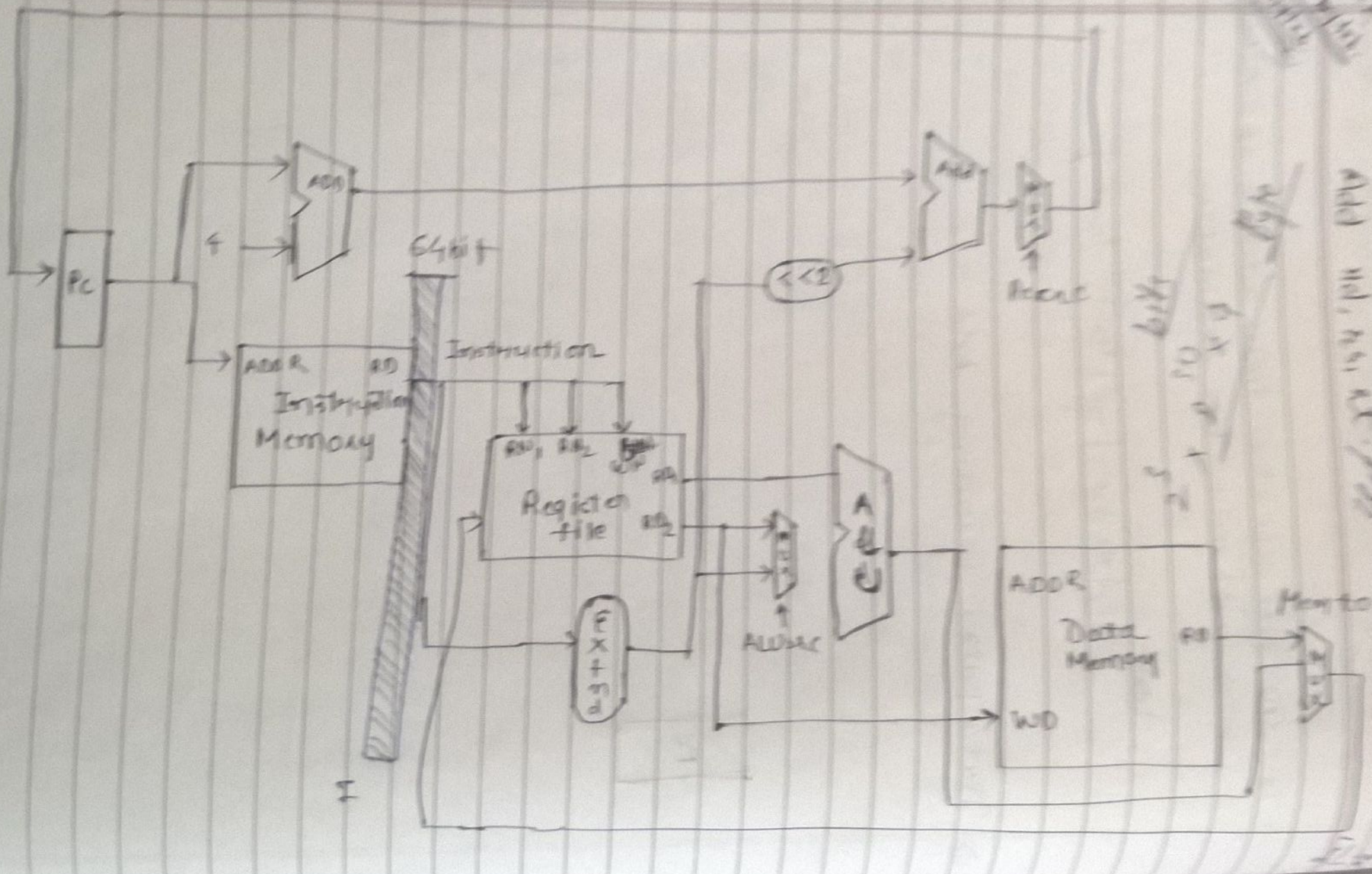
→ Skip topics which are titled with "red" colour.

→ In lw rs will be base address of Array and rd will be destination.

op	rs	rt	rd	shamt	funct

⇒ Data Path Executing Add –




PC
ADD
4
64bit
ADDR
RD
Instruction Memory
Instruction
Register file
Extend
<<2
ADD
ALUSrc
ALU
ADDR
Data Memory
WD
RD
MemtoReg

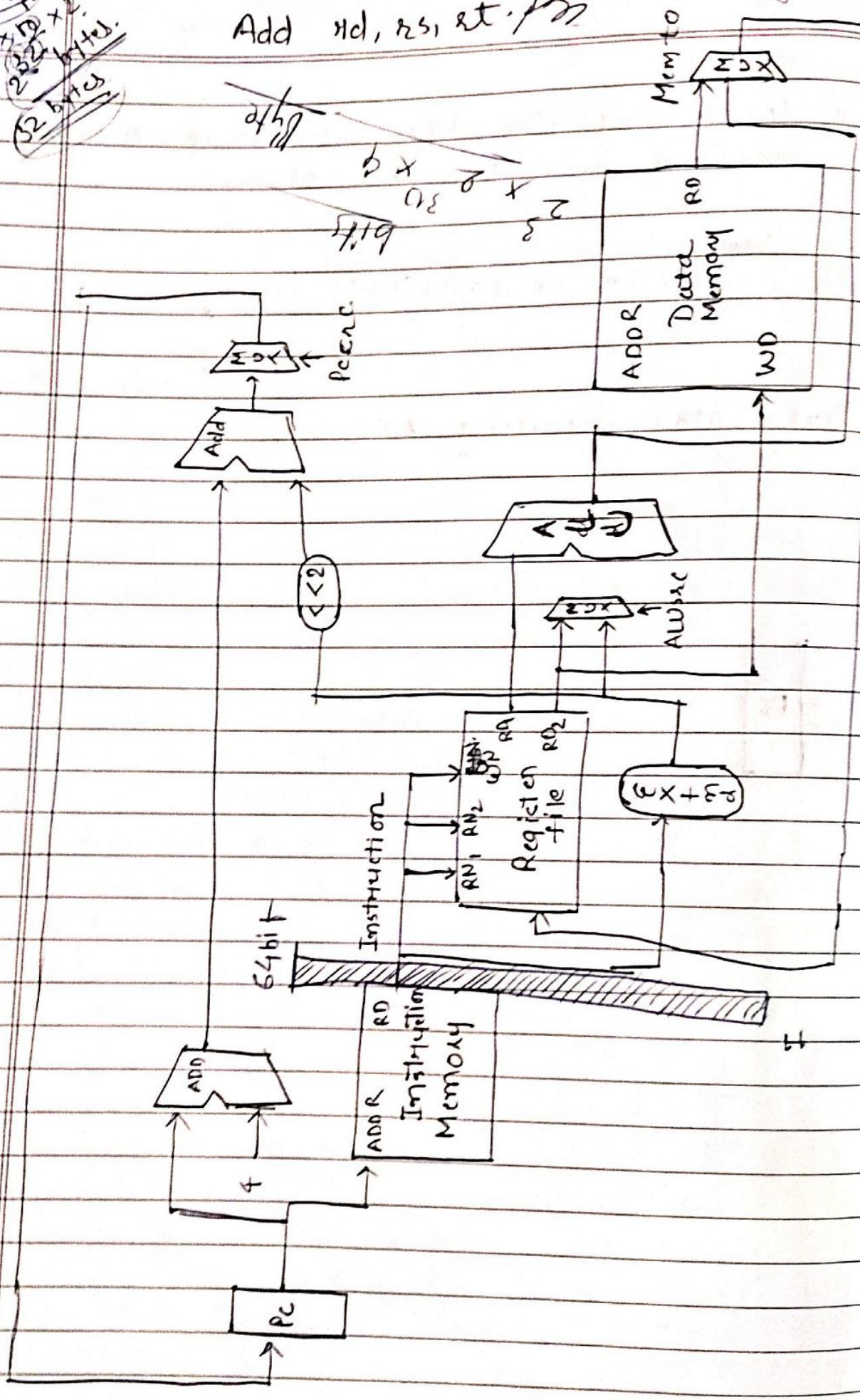
4 GB = 4 × 1000 × 1000 × 1000

= 4 × 2^10 × 2^10 × 2^10 bytes

= 2^32 bytes

32 bits

Add rd, rs1, rs2



⇒ Control Signals –

	Signal Name	Effect when Deasserted	Effect when asserted.
(1)	Reg Dst	The reg destination number for the write reg. comes for the rt field.	The register dest number for the write reg. comes from the rd field.
(2)	Reg write	None	The reg. on the write reg. input is written with the value on the write data input.
(3)	ALUSrc	The second ALU operand comes from the second reg. file output.	The second ALU operand is the sign-extended, lower 16-bits of the instruction.
(4)	PCSrc	The pc is replaced by the output of the adder that computes the value of PC+4	The pc is replaced by the output of the adder that computes the branch target.

(5) MemRead None Data memory contents designated by the add. input are put on the first Read data output.

(6) MemWrite None Data

⇒

⇒

Instruction name	Reg. Dst.	Reg. Write	ALU Src	PC Src	Memory Read	Memory Write	Mem to Reg.
R-type.	1	1	0	0	0	0	0
LW	0	1	1	0	1	0	1
SW	~~X~~ (MUX is not in Use)	0	1	0	0	1	X
I-type (Beq, Bne)	X	0	0	1	0	0	X

Multi-cycle Processor.

⇒ Disadvantages of Single-cycle pro. —

(1) Clock cycle must have the same length for every instruction.
(2) Several inst could fit in a shorter clock-cycle

⇒ Goal: Break inst. into steps and apply same clock cycle to each step

→ functional units can be shared between diff. cycles within one instruction.

⇒ Breaking instructions into steps —

→ Each step takes one clock cycle.

IF (1) Instruction fetch and Pc increment
ID (2) Instru. decode and register fetch
EX (3) Execution, memory computation, branch
MEM (4) Memory access on R-type instru. comp
WB (5) Memory read completion (write Back)

~~Steps~~

Step-3 R type: ALUOut = A op B.
branch: IF (A == B), Pc = ALUOut.
Jump:
Memory Ref: ALUout + Signextended (IR(15-0))

In 3rd step branch and jump instru. are over.

Read from 45 (Page) in PPT
~~to~~ skip topics from b/w pages.

→ for LW, we require all 5 steps.

→ We have to write all the control signals for all steps.

Pipelining

Goal: → clock cycle per instruction CPI <= 1.
→ Want 100% hardware Utilization.
→ cycle time is not dependent on ~~the~~ slowest instruction.

→ MIPS in RISC Type of Micro-processor so, Most instru are of same length.

⇒ Data flowing Right to left may cause "Hazard."

⇒ Graphical Representation of Pipelining —

tion

(1) Data Hazard
(2) Structure Hazard.
(3) Control Hazard.

→ Clock ~~cy~~ cycle = No. of instructions + pipeline stages − 1.

(1) → When Same registers are used for different steps like EX, FD ... then it will create a problem is called a "Data Hazard"

e.g. Add $s0, $s1, $s3
Add $s2, $s0, $s5
Add $s0, $s7, $s8.

(2) → In Multicycle Processor we can't use same Memory with data and instruction.

e.g. LW $s0, 16($s3)
Add $s4, $s1, $s5
Add $s2, $s5, $s6
Add $s7, $s5, $s6.

(3)