L-12 / 30.08.2023 /

Mid-term-1

~~Date~~

L-13 / 04.09.2023 /

Accumulator based CPU

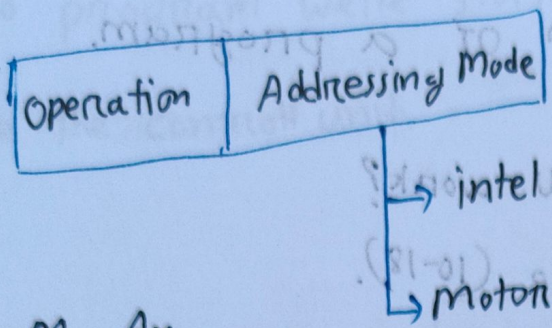
ADD R1 → Operation code

⇒ Only one operand field other operand or stone field is set by default to accumulator register.

ADI → if 'I' present in OP code → imediate → next field is not field, is a data.

	ADD M, (memory)	ADD R, (register)	ADI 45H (data itself)
Mode	memory mode	register	imediate
Processing Time	Slowest	moderate	fastest
memory access count	1+1	1+0 for instruction read	1+0

General Purpose CPU

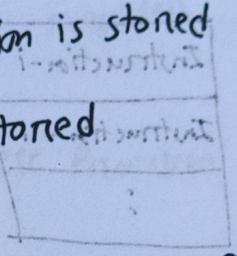


MOV M_2 Ax

→ Destination for intel processor

ADD R_1 R_2

- Address of instruction ⇒ where instruction is stored
- Address of data ⇒ where data is stored



- instruction data ⇒ Binary formate of a command.
- Data ⇒ Data is our regularly used number.
- How does the processor set the address of data?

  ⇒ from instruction
- How does the processon set the address ~~in PC,~~ ~~program counter?~~ or instruction?

  ⇒ Operating System will load this address in PC, program counter.

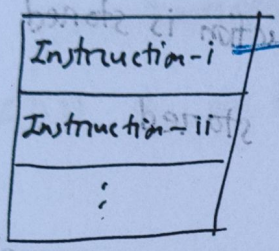
  ⇒ next address generated by a intregated circuit added with the PC.

* Sometimes updated by preceding instruction.
  during execution of a program.

* How does the CPU work?
  ⇒ read from slide (10-18).

* Basic command / instruction ⇒ micro operation.

* User Program



* Instruction Set Architecture

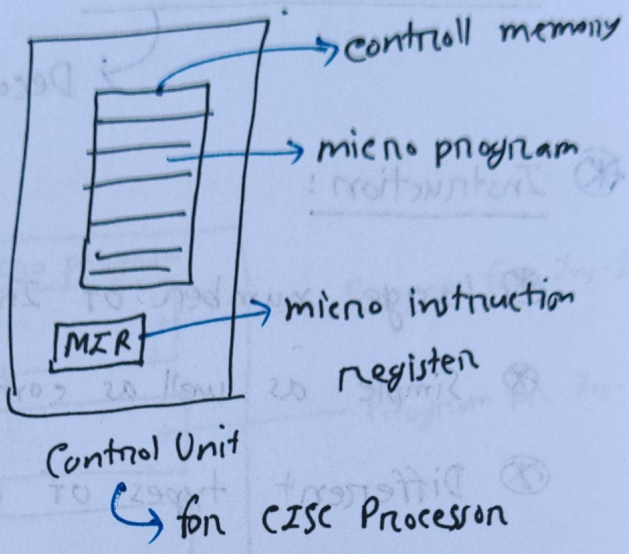
  RISC vs CISC
  → all included, flexiable.

* ADD R, M,

  1. Read R,
  2. Read from M,
  3. Activate ALU to select Adden Unit
  4. Add Operation
  5. Active R, to save result.

  } Micro Operation.

⊛ Micro program were store in controll memory inside the controll unit.



⊛ CISC Processor 70-80%.
instruction used 10-20%.
Col. of control memory unused

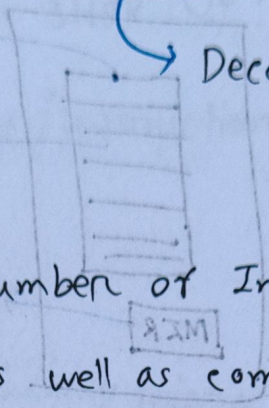
⊛ then introduced the RISC Processor
- more faster than before
- Only LOAD & STORE command can access memory.

L-14/ 11.09.2023/

* CISC Processor: → { Software base
  Micro program Control Unit

⇒ Design of control unit:

→ Decodes Instruction



* Instruction:
  * large number of Instruction
  * Simple as well as complex
  * Different types of Memory mode

Some Example

ADD R1 R2 R3 → Register Mode

ADD R1 (15) R3 → Imediate mode

ADD (M) R2 R3 → memory Mode

ADD M1 M2 M3

ADD R1 [R2] R3 → Memory indirect addressing mode

ADD R1 [R2+R3] R4 → Register pointing to memory

When, it inclosed, it means memory.

✳ Program (H.L.)

M.C

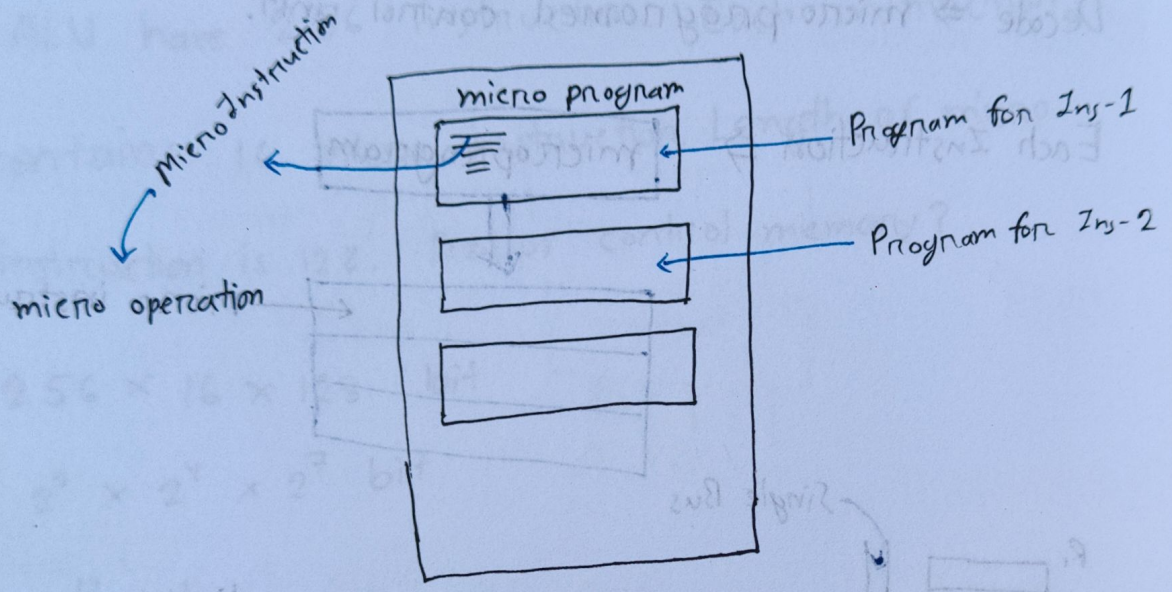
Inst - 1 → 0110110110 → C.V.

Inst - 2

Inst - 3

⋮

Inst - n



✳ micro operation < micro instruction < micro program < mc < L.L < H.L

↳ at least 1 or more

✳ Micro Operation:

ADD R_1 R_2 R_3

1. CPU will read data from R_1 C_1 C_7
2. CPU will read from R_2 C_2 C_8
3. ALU to be activated for ADD C_{11}
4. Save to temp Register R_3 C_{15}
5. Temp $R_3 \rightarrow R_3$ C_6 C_9

✳ Full demonstration available in Picture

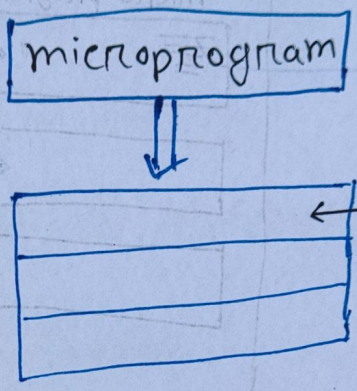
✳ Parallel micro operation can make it faster

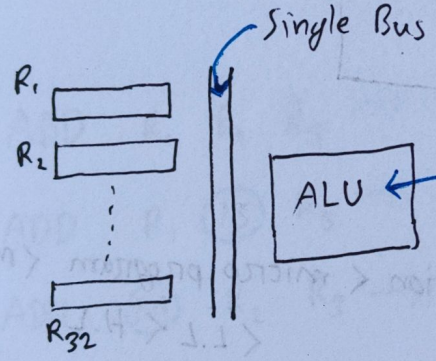
L-15 / 13.09.2023 /

- CISC
  - Number of Instruction
  - Addressing Mode
  - } More
  - May contain simple & complex instruction

Decode ⇒ micro programed control unit.

Each Instruction ⇒ microprogram





32 different operation

↳ 32 control signals to control or select ALU Operation.

- We need 32 control signal for other operations.
- for every register, we need two control signal. One for R → BUS another for BUS → R
- 64 control signals to transfer content of Register to/from BUS.

Then, total control signals 128.

C_1	C_2	C_3	.....	C_{128}
1	0	0	.....	0
0	0	1	.....	0

Then, the length of a micro instruction is 128.

* In a micro instruction, how many 1 contains, these are micro operation.

* For micro instruction we need 128 bits for this example.

* ALU have 256 instruction. Each instruction contains 16 micro instruction. Length of micro instruction is 128. Size of control memory?

$\Rightarrow 256 \times 16 \times 128$ bit

$= 2^8 \times 2^4 \times 2^7$ bit

$= 2^{19}$ bit

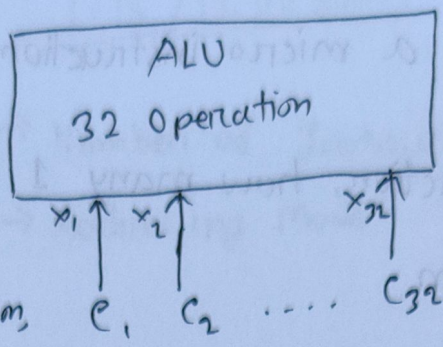
$= \frac{2^{19}}{2^3}$ byte

$= 2^{16}$ byte

$= 2^6$ kB

$= 64$ kB control memory required.

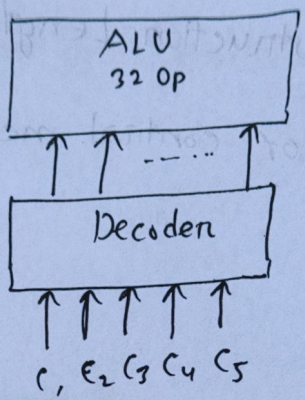
* Only one operation can be done at a time, that means in a micro instruction, there will be only one 1.



micro instruction c_1 c_2 .... c_{32}

0 1 0 0

⊛ For memory optimize, we can use decoder,

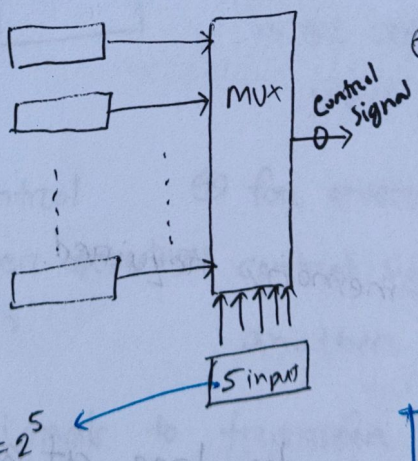


$32 = 2^5$

⇒ But there will be a delay.

⊛ For Register, we can use MUX to select Register

~~$2^6 = 64$~~



$32 = 2^5$

⊛ MUX for Register to BUS

⊛ DMUX for BUS to Register.

Next Class Quiz-2

Design of a Program

✳ CISC Processor (Easy to Use)

→ Large no of Instruction (Simple + Complex)

→ Different type of Addressing Modes

✳ 256 of instruction contains different type of addressing mode for a single instruction.

✳ CISC Processor

Instruction : 20-30%
Used : 70-80%

memory = 60%
used = 2%

✳ RISC Processor:

Simple instruction

Simple Register Mode instruction

For memory access only two instruction available
LOAD & STORE

⇒ Control Unit is hardware base
No need control memory

⇒ PIPE Lining

⇒ CPI ≅ 1 (for CISC it is 4/5 or higher)

CISC ⇒ $\downarrow I_{100} \times \uparrow Avg\ CPI_{4} \times C_{same}$

RISC ⇒ $\uparrow I_{200} \times \downarrow Avg\ CPI_{1} \times C_{same}$

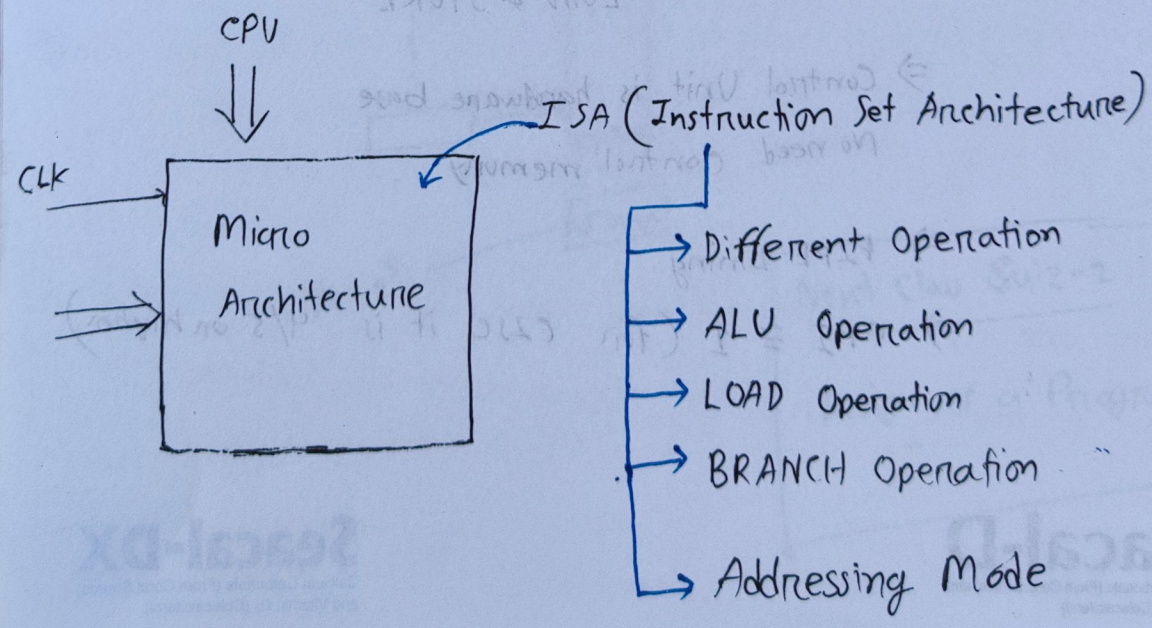
⇒ Then impact of Avg CPI is higher than instruction.

So RISC Processor is more efficient and speedy.

⇒ Instruction count of same program will be increase upto 50% in RISC.

L-16 / 18.09.2023 /

- Single Cycle Implementation
- Multi Cycle Implementation
- Pipelines (Single Cycle)



⊛ Implementation time ⇒ Varies with operation & addressing mode.

⊛ CPU-A

		delay	
LOAD →	30 μsec →	60 μsec	} Single Cycle Implementation.
STORE →	60 μsec →	60 μsec	
BRANCH →	40 μsec →	60 μsec	

⇒ maximum time will be the time of a single cycle, OR slowest instruction.

⊛ Multi Cycle implementation:

⇒ Processing of instruction split into task.

⇒ Fetch (20 μ) → Decode (8 μ) → Register Read (4 μ) → ALU Operation (8 μ) → Memory Access (20 μ) → Result Store (6 μ)

These are subtask

⊛ Simple instruction may required few subtask only.

⊛ Now we need to set the clock to the highest time.

CLK ⇒ 20 μsec

For Register Mode : Fetch, decode, Register Read, ALU, Result

= 5 CLK Cycle

For memory mode : 6 CLK Cycle

For Branch instruction : Fetch, Decode, Register Read

= 3 CLK cycle.

L-17/20.09.2023/

Instruction Set Architecture Implementation :

* Micro Architecture

Hardware is used to decode the instruction.

* Operation / Instruction

* Addressing Modes → Depends on architecture

⇒ How the operands are addressed in operands fields.

RISC → ADD R_1 R_2 R_3 → Register Mode } CISC

ADD R_1 M_1 R_3 → Memory Mode }

✳ Micro Architecture

↓

(i) Single Cycle Implementation

(ii) Multi Cycle Implementation

(iii) Pipelining Implementation

✳ Single Cycle Implementation

⇒ CLK Duration is set by the slowest Instruction time.

✳ Multiple Cycle Implementation

⇒ First we need to make a list of sub task.

→ To read machine code from RAM

⇒ Fetch → Decode → Register Read → ALU Operation → Result Store

✳ ADD R_1, R_2, R_3

⇒ Fetch → Decode → Register Read → ALU Operation → Store Result

= 5 Sub task = 5 CLK cycle

✳ LOAD R_1, M

⇒ Not in the list of Sub task.

⇒ Fetch → Decode → Register Read → ALU Operation → memory Access (for data only) → Result Store to Register

Read/Write

⇒ Now it is complete.

✱ But some subtask may not required for all Instruction.

✱ LOAD R_1, R_2(16) → memory → R_2 + 16 = memory address

⇒ Fetch → Decode → Register Read (R_2) → ALU R_2+16 → Memory Access → Register STORE

✱ Store Machine Code

Instruction RAM

Data RAM

Store Data Only

} Harvard Architecture & RISC Processor

✱ STORE R_1, R_2(16) → memory

⇒ Fetch → Decode → Register Read → ALU Operation → Memory Access

= 5 CLK Cycle

✱ ADD R_1 M_1 R_3

⇒ Not suitable for the sequence of sub task list.

⇒ Then what is the list of sub task and the sequence ?

⇒ Fetch → Decode → Register Read → Memory Access → ALU Operation → Result Store Register

→ memory

✳ ADD R_1, $[R_2 + R_3]$, R_4

⇒ Fetch → Decode → Register Read → ALU Operation → Memory Access → ALU Operation → Result Store

(memory address calculation)

= 7 cycle

✳ Fetch (20μ) → Decode (5μ) → Register Read (5μ) → ALU Operation (5μ) → Memory Access (20μ) → Result Store (5μ)

Then,

Single Cycle Implementation = 60μ
Multiple Cycle Implementation = 20μ
} CLK Speed

✳ RISC (followed Hervard Architecture)

→ Seperate Memory for Instruction & Data

⇒ Use Pipelining System.

⇒ Fetch → Decode → ALU → Memory → Result STORE

Fetch: from instruction RAM

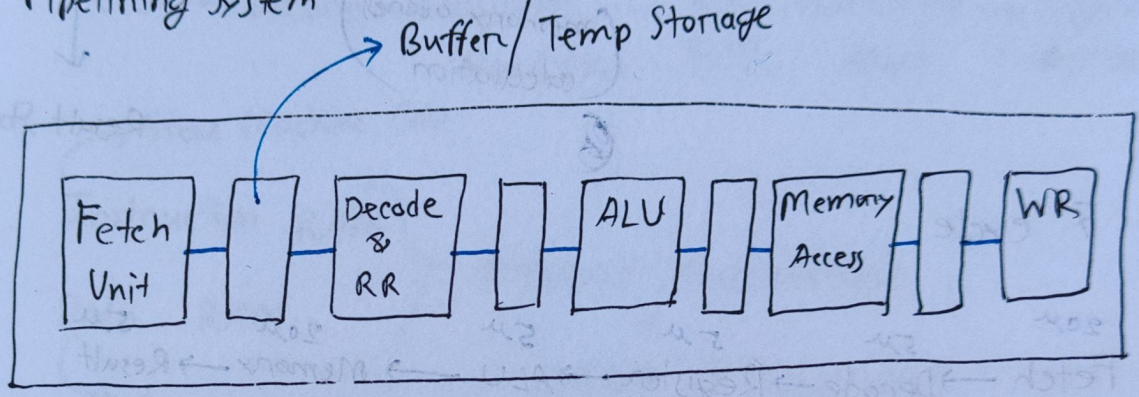
Decode & Register Read

Memory Access: Read/Write data to/from Data RAM

⇒ Then CLK cycle = 5 CLK

⇒ Processon are designed by assemble line or production line.

⊛ Pipelining System



	1	2	3	4	5	6	7	
I_1	IF	ID	ALU	MA	WR			
I_2		IF	ID	ALU	MA	WR		
I_3			IF	ID	ALU	MA	WR	
I_4				IF	ID	ALU	MA	
I_5					IF	ID	ALU	
I_6						IF	ID	
I_7							IF	

→ t

⊛ For 1st Instruction, we need 5 CLK Cycle to get output.

⊛ For 2nd Instruction, we need 1 more CLK Cycle to get Output.

∴ 100 Instruction, How much CLK Cycle need in this Pipeline System.

$\Rightarrow 5 + (100-1) \times 1 = 104$ CLK Cycle

∴ 100 Instruction = 104 CLK Cycle

$\therefore CPI = \frac{104}{100} \cong 1$

∴ For pipelining CPI reduced to 1.

∴ If we execute in normal Non Pipeline processor

CLK Cycle need = 500

$\therefore$ Speed Up = $\frac{500}{104} \cong 5$

∴ For Pipeline, Every instruction need to be independent.

If dependency exist, then processor need to wait some CLK Cycle.

L-18/25.09.2023

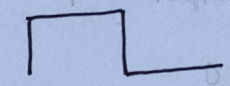
* Single Cycle Implementation
* Multi Cycle Implementation
* Pipelining Implementation

* CPU Design depends on Instruction Set.

* Single Cycle:

Assume that, there are 100 instruction and I_{45} is the most slowest and takes 20 μsec.

Then CLK Cycle will set to 20 μsec.



∴ Avg CPI = 1

* Multi Cycle Implementation:

Divide instruction in multiple task.

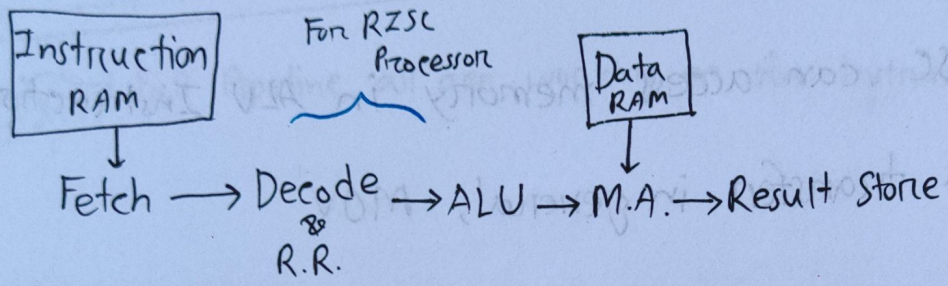
There maybe 50% of task takes less time.

Operation	Addressing Mode
ADD	R_1 R_2 R_3
ADD	R_1 M, M_3
ADD	R_1 $[R_3+R_5]$ R_6

↳ refers to memory

⊛ Processor will allow faster instruction to process faster and slowest instruction takes its time to process its operation.

⊛ Processing task is split into sub task.

⇒ 

ALU					
ADD $R_1 R_2 R_3$	✓	✓ $R_1 R_2$	✓	×	✓ R_3
LOAD $R_1 R_2(16)$	✓	✓ R_2	✓ R_2+16	✓ Read	✓ R_1
STORE $R_2 R_3(16)$	✓	✓ $R_2 R_3$	✓ R_3+16	✓ Write	×

⊛ RISC Addressing Mode:

ALU : ADD R_1 R_2 R_3 → Register Mode

ADD R_1 64 R_3 → Immediate Mode

For CPU ↔ RAM ⇒ LOAD/STORE → just for memory access

⊛ Before ALU Operation, data must be loaded in a Register.

* LOAD R_1 $R_2(16)$ → memory - always in indirect form

STORE R_2 $R_3(16)$

If we give memory address directly then instruction size will increase.

* ALU instructions cannot access RAM. Only LOAD and STORE can access memory.

* CISC can access memory in ALU Instruction. For data transfer in general, MOV.

* For CISC

ADD R_1 M, R_3

⇒ Fetch → Decode → MA → ALU → Result Store

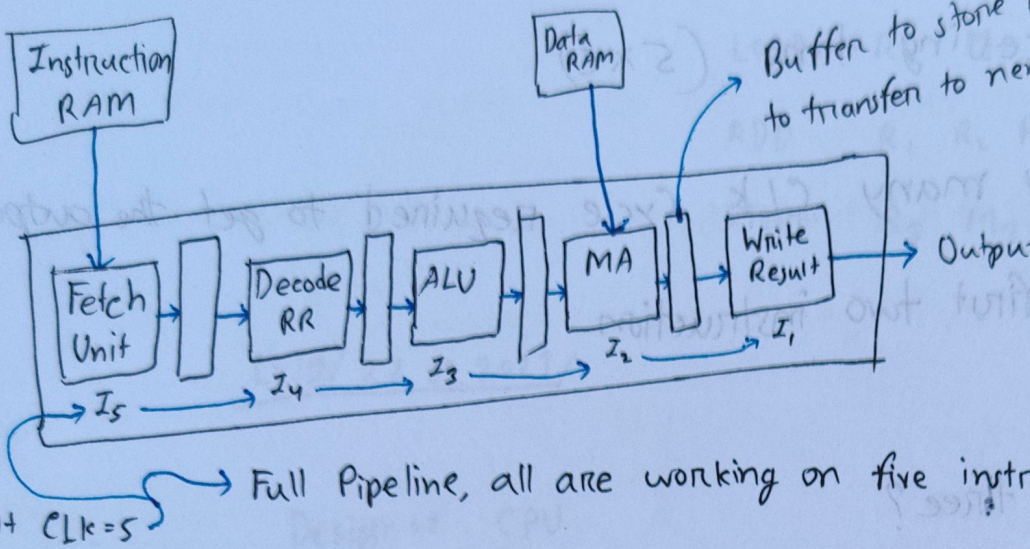
ADD R_1 $[R_2+R_3]$ R_4

⇒ Fetch → Decode (R.R.) → Calculate (M.A by ALU) → MA → ALU → Result Store

Avg CPI > 1

* Pipe Lining Implementation ~~Ampli~~ Applicable to RISC processor.

RISC



At CLK = 5 → Full Pipeline, all are working on five instruction.

Instruction \ CLK	1	2	3	4	5	6	7	8
I_1	IF	ID	ALU	MA	WR			
I_2		IF	ID	ALU	MA	WR		
I_3			IF	ID	ALU	MA	WR	
I_4				IF	ID	ALU	MA	WR
I_5					IF	ID	ALU	MA
I_6						IF	ID	ALU
I_7							IF	ID
I_8								IF

Next Class Exam

⇒ Assume all the instruction are all independent.

⊛ How many CLK cycle required for each instruction?

⇒ 5.

- Each instruction takes 5 CLk cycle.
- Processing time $(5 \times C)$
- How many CLk cycle required to get the output of first two instruction

  ⇒ 6
- First three?

  ⇒ 7
- Total CLk cycle of a program containing 100 instruction?

  ⇒ $5 + (100-1)$

  $= 5+99 = 104$ CLk Cycle
- Throughput of the System.
- What does Pipe-Line improve?

  ⇒ Processing time of a program.

  Because, each instruction is still same.
- Avg CPI $= \frac{104}{100} \cong 1$

RZSC : $\uparrow I \times \underset{\cong 1}{\downarrow Avg\ CPI} \times C_{same}$

CZSC : $\downarrow I \times \underset{>1}{\uparrow Avg\ CPI} \times C_{same}$

Processing time:

RZSC < CZSC

↳ Avg CPI most effective

CISC

ADD M_1 M_2 M_3 ⟶ RISC ⟶ LOAD R_1 M_1

LOAD R_2 M_2

ADD R_1 R_2 R_3

STORE R_3 M_3

L-19/ 27.09.2023/

Design of CPU

⊛ Micro Architecture Implementation:

⊛ Instruction:

ADD R_1 R_2 R_3

ADD → Operation; R_1 R_2 → Operands; R_3 → Result field

SUB R_4 R_5 R_6

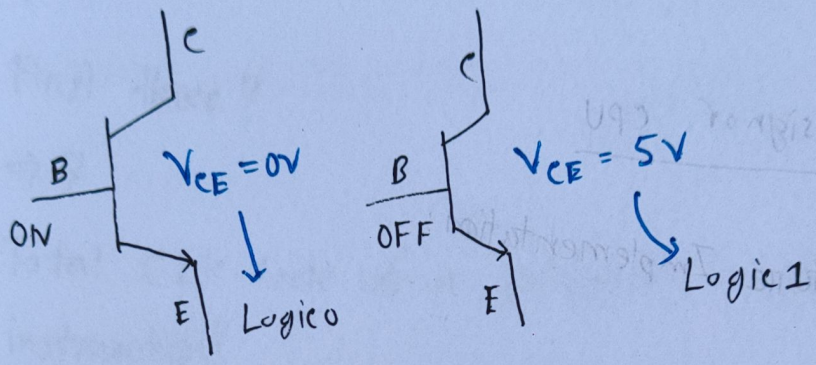
⊛ Lets CPU has 32 Register each of 32 bits.

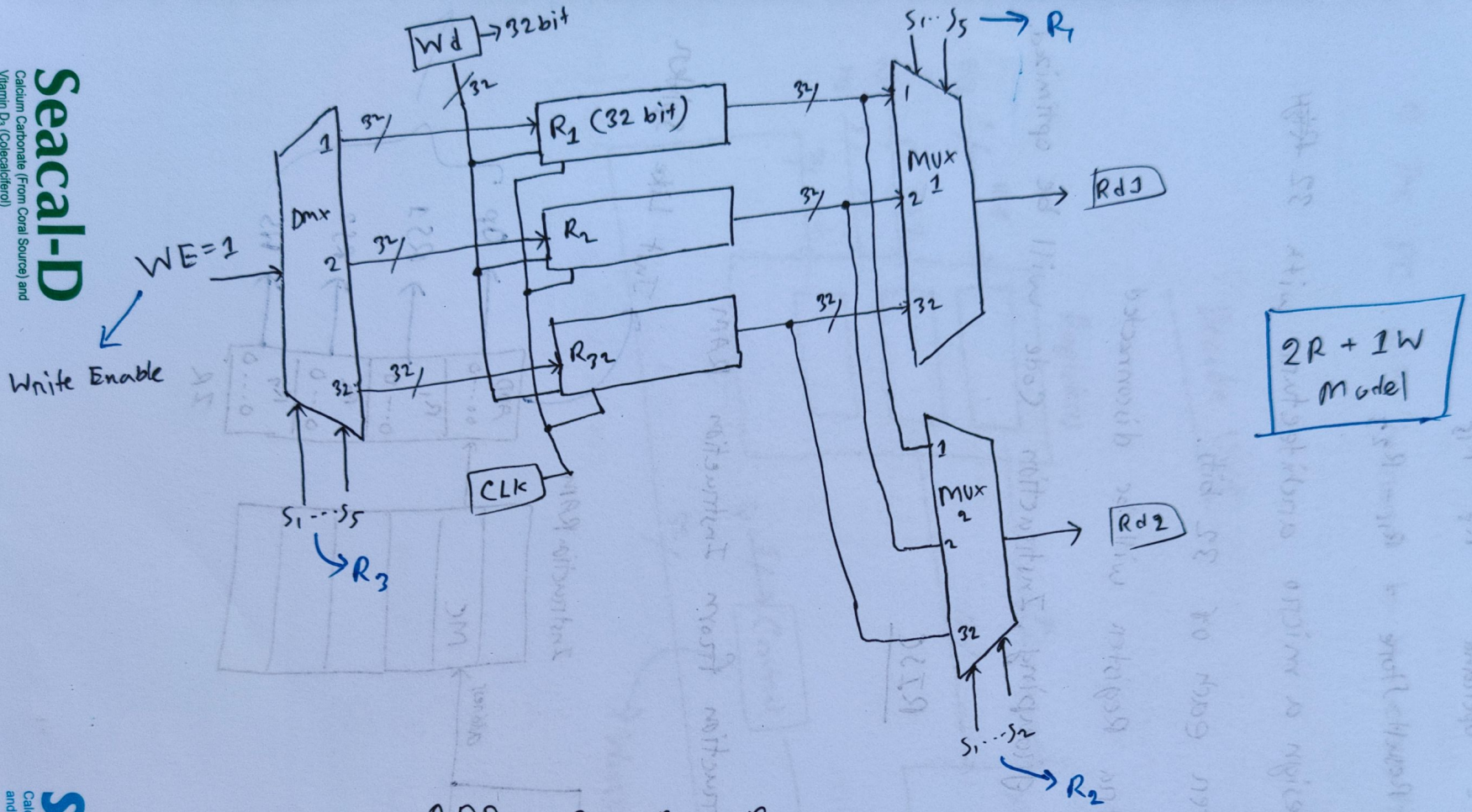
that means there are 32- D-Flip Flop with 32 input and output Line.

⊛ We can read two Register at a time.

(*) We can save to one Register at a time.

Flip Flop used only for data storie. that's why we use D Flip-Flop. Flip Flop used transistor to store data (A number of transistor)





2R + 1W Model

ADD R_1 R_2 R_3

⊛ If Grouped Like ⇒

1st Operand = $R_0 - R_7$

2nd Operand = $R_8 - R_{15}$

Result Store = $R_{16} - R_{23}$

Now design a micro architecture with 32 ~~Regi~~ Registers each of 32 bits.

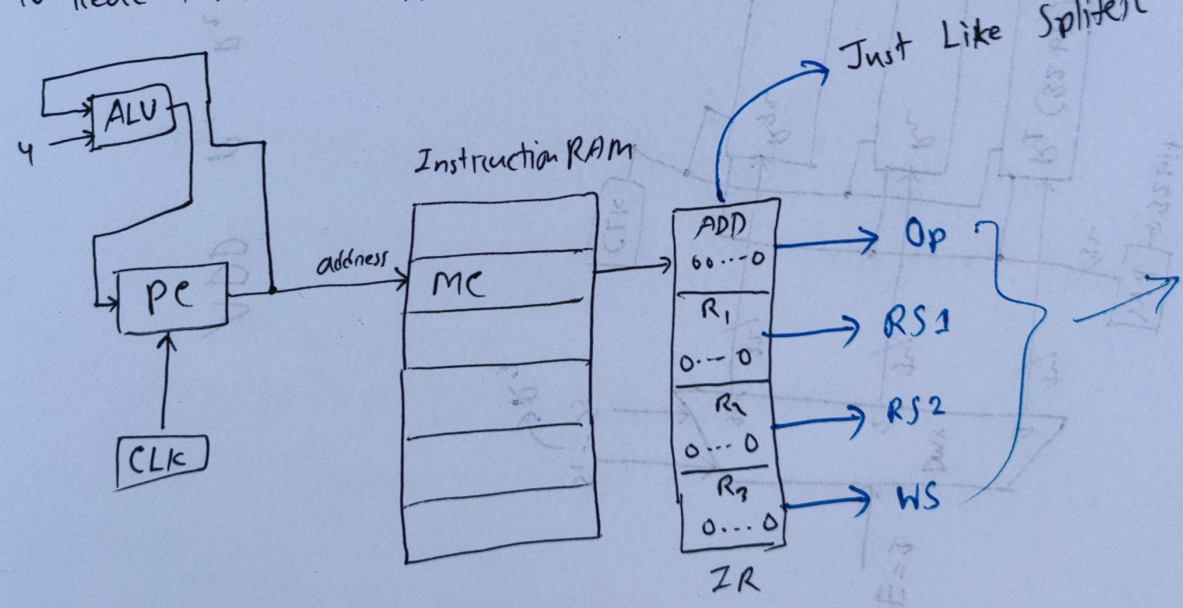
⊛ Extra Registers will be disconnected

⊛ For grouping, Instruction Code will be optimized.

RISC

Fetch Unit

To read instruction from Instruction RAM

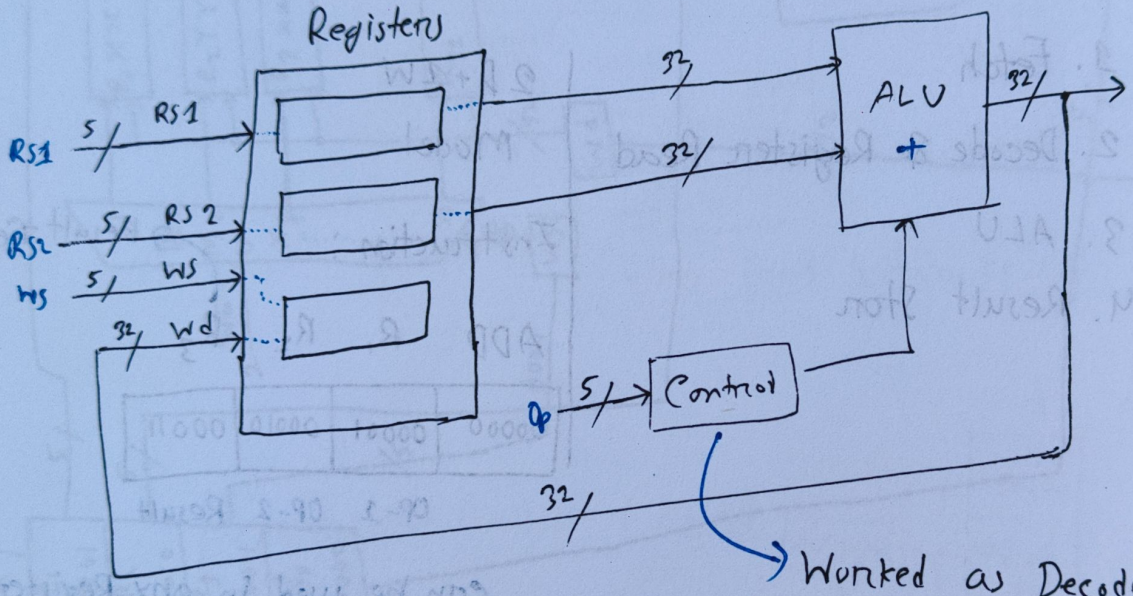


⊗ Machine Code 32 bits, RAM is Byte addressable.

⇒ then we need 4 memory location to store MC.

⇒ The PC will increase by 4 every time.

Decode Unit



L-20/02.10.2023/

⊗ Data Path Design for a CPU that support register addressing ALU instruction:

⇒ 32 Register

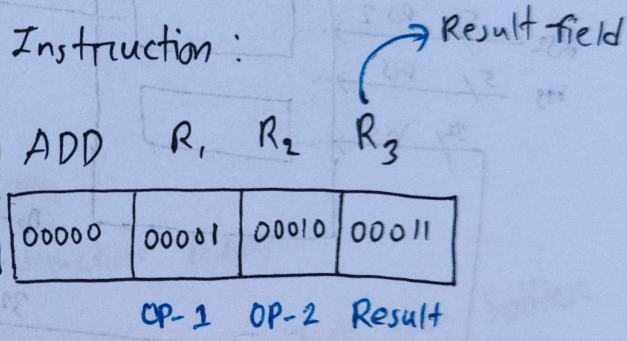
⇒ Any Register can be used for any operand

⇒ Multi-cycle implementation

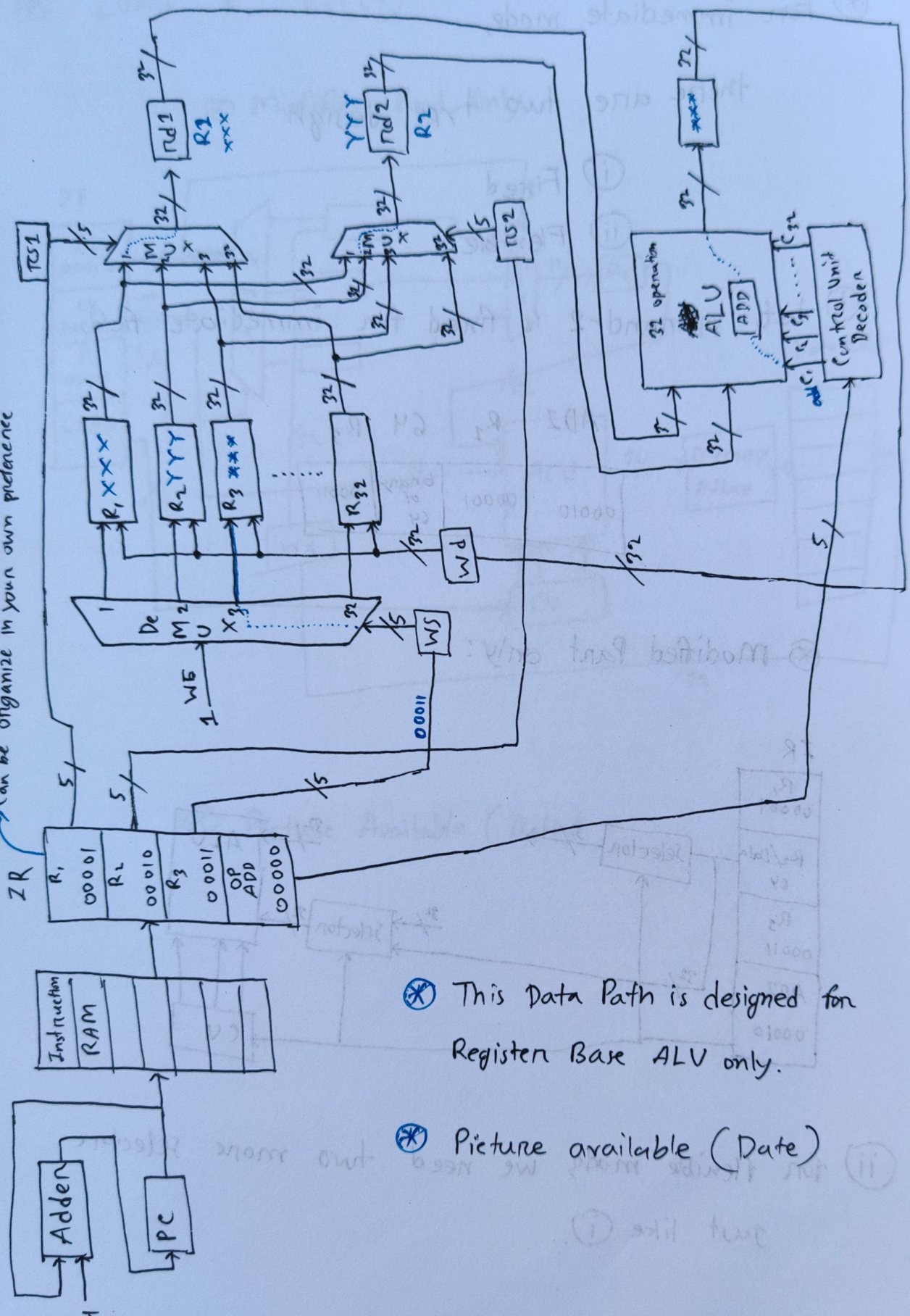
⇒ 1. Fetch
2. Decode & Register Read
3. ALU
4. Result Store

2D+1W Model

Instruction:



can be used in any Register.



* This Data Path is designed for Register Base ALV only.

* Picture available (Date)

⊛ For immediate mode,

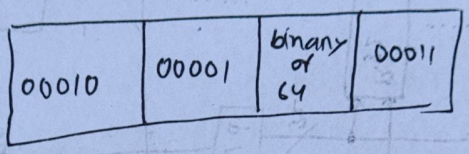
there are two type design

(i) Fixed

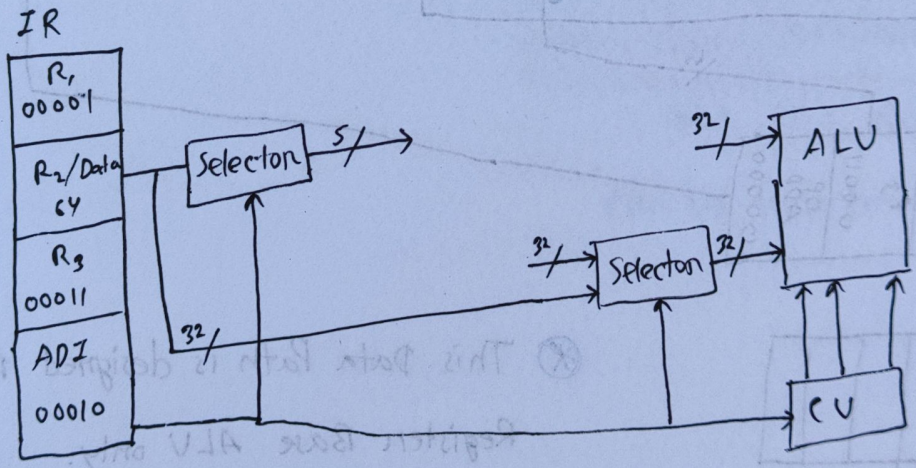
(ii) Flexible

(i) lets operand-2 is fixed for immediate field.

ADI R_1 64 R_3



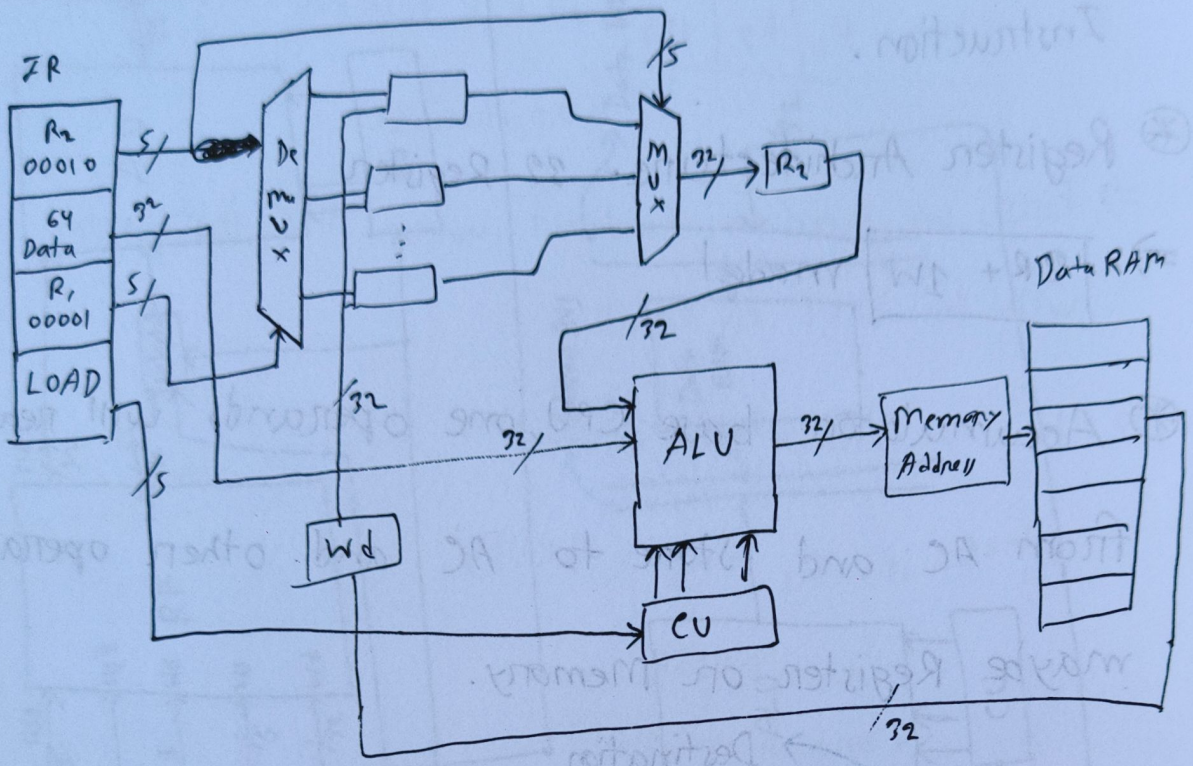
⊛ Modified Part only:



(ii) for flexible mode we need two more selectors just like (i).

⊛ LOAD R, R_2(64)

⊛ Modified Part Only



⊛ Picture Available (Date)

L-21 / 04.10.2023

* Data Path Design for LOAD/STORE and BRANCH Instruction.

* Register Architecture - 32 Register

⇒ 2R + 1W model

* Accumulator base CPU, one operands will read from AC and store to AC and other operand maybe Register or Memory.

* LOAD R_1 R_2(128)

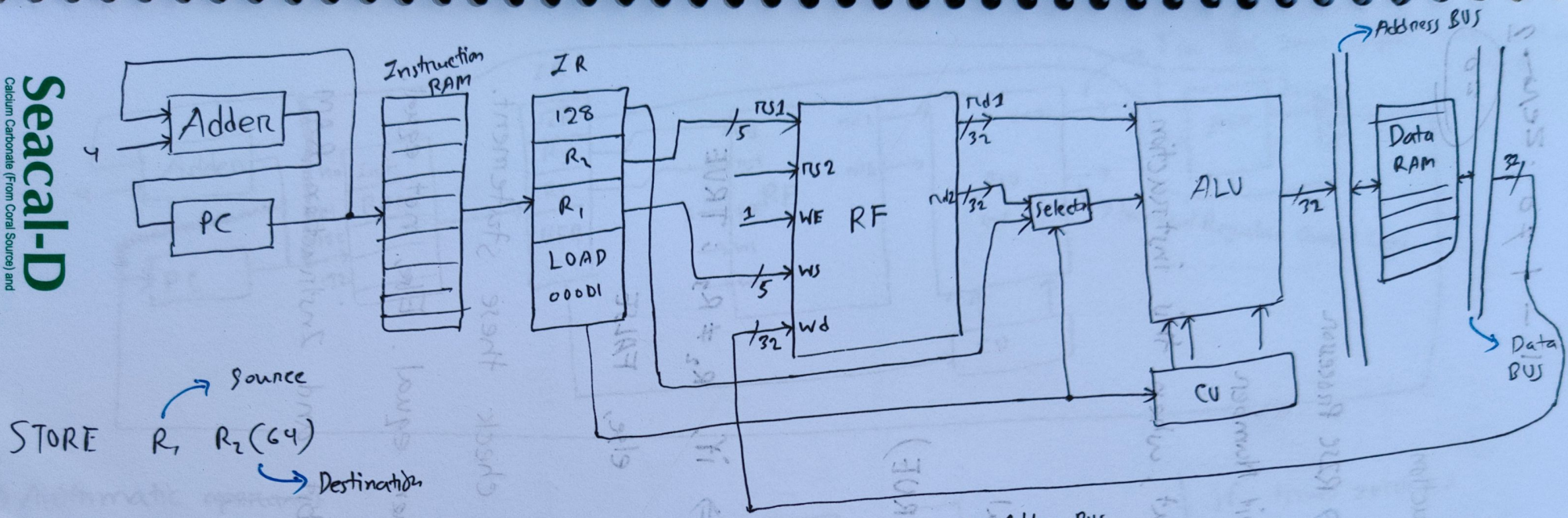
R_1 → Destination

R_2(128) → Source (Memory)

Next Page

Seacal-D

Calcium Carbonate (From Coral Source) and Vitamin D_3 (Colecalciferol)



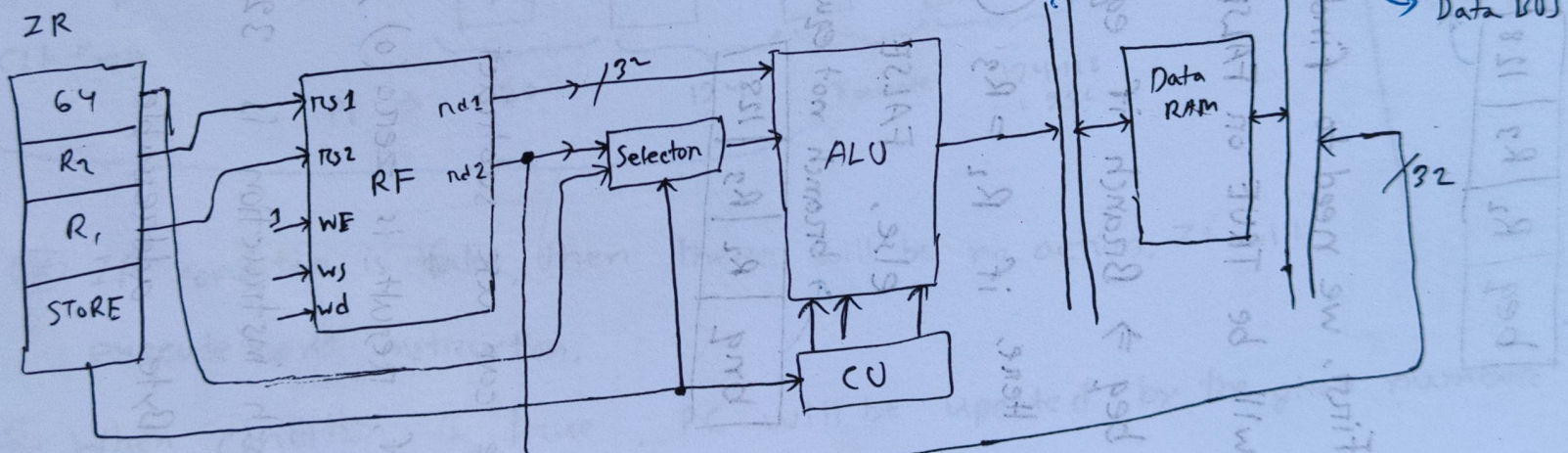
STORE R_1 $R_2(64)$

Source

Destination

Seacal-DX

Calcium Carbonate (From Coral Source) and Vitamin D_3 (Colecalciferol)

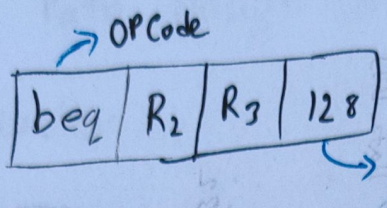


1 - 1 ≠ 0 : zero = 0

⊛ BRANCH Instruction

⇒ Conditional Branch Instruction

⇒



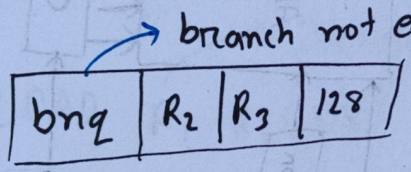
⇒ RISC Processor

⇒ First, we need to find out, when this instruction will be TRUE or FALSE.

⊛ beq ⇒ Branch if equal

Here, if $R_2 = R_3$ (TRUE)

else, FALSE



⇒ if, $R_2 \neq R_3$; TRUE

else, FALSE

⊛ We can use subtract to check these statement. If, result is zero (0), then equal. Else, not equal.

⊛ Each instruction is 32 bit and Instruction RAM is Byte addressable.