

# PROBLEM STATEMENT

## VICARAK TASK

- Choose an 13-bit architecture (Instruction size will be 13-bit)
- Define a specialized instruction set tailored to a specific application or a set of applications (eg. signal processing, cryptography),
- Create a detailed architectural specification, including pipeline stage, register file, ALU, and memory interfaces.

Min Ist to be added:

| ADD  $r_1, r_2, r_3$  | SUB  $r_1, r_2, r_3$  | MUL  $r_1, r_2, r_3$  | DIV  $r_1, r_2, r_3$  | INC  $r_1$  | DEC  $r_1$  |

• Logical Instructions:

| AND  $r_1, r_2, r_3$  | OR  $r_1, r_2, r_3$  | XOR  $r_1, r_2, r_3$  | NOT  $r_1, r_2$  |

• Control Flow Instructions:

- 1) JMP addr : Jump to the address given | PC = addr
- 2) BEQ  $r_1, r_2$ , addr : Branch to the address if values in  $r_1$  and  $r_2$  are equal. | if ( $r_1 == r_2$ ) PC = addr
- 3) BNE  $r_1, r_2$ , addr : Branch if not equal. | if ( $r_1 != r_2$ ) PC = addr
- 4) CALL addr : Call a subroutine at that address. | stack[SP] = PC + 1; SP = SP - 1; PC = addr
- 5) RET : return from a subroutine. | SP = SP + 1; PC = stack[SP]

• Memory Access Instructions

- 1) LD  $r_1$ , addr : Load from address to  $r_1$  |  $r_1 = \text{memory}[\text{addr}]$
- 2) ST addr,  $r_1$  : Store  $r_1$  at the address |  $\text{memory}[\text{addr}] = r_1$

• Custom Instructions

- 1) FFT  $r_1, r_2$ : Do fft at addr  $r_2$  & store at address  $r_1$ .
- 2) ENC  $r_1, r_2$ : Encrypt data at addr  $r_2$  & store at address  $r_1$ .
- 3) DEC  $r_1, r_2$ : Decrypt data at addr  $r_2$  & store at address  $r_1$ .

## APPROACH:

Current knowledge:

- Similarity with 8085 Architecture.
- Theoretically idea to implement custom instructions.  
↳ catch: no special hardware req. for this.  
So, first step complete the rest in coding.

Current Doubts:

- Implementing the Architecture in Software
- How and where to give the instruction.
- Memory size?
- How to store data to memory directly.
- Flag register implementation.
- Implementation of I/P & o/p w.r.t clock.

Plan:

Tuesday: Research on 8085 & RISC implementation.  
Wednesday: Recall complete verilog knowledge.  
Thursday: Draft first architecture clearing all doubt.  
Friday: Start coding the first draft.  
Saturday: Just frame w.r.t deadline i.e next thursday.

Similarance

- 8085 Arch
- RISC ?

# Instruction Format

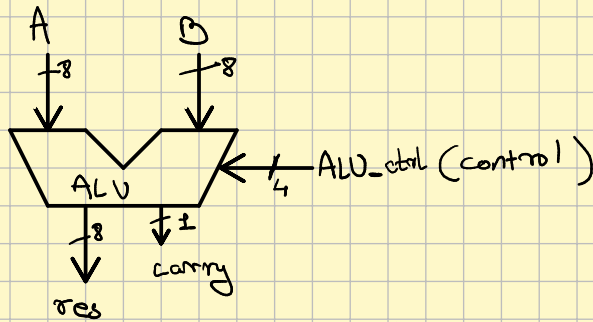
- Instruction Size: 19 bits
- Register Size in it: 4 bit each.  $\Rightarrow$  [16:5] (modular)
- Opcode Size: 5 bits  $\Rightarrow$  [4:0]
- $\star \rightarrow$  Beside operation are 2 cycle instruction. So use a NOP after using this just to be sure.

## Instructions

Implementation	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
															0	0	0	0	0	HLT
do reg_wrt_en = 1 for I[4:3] = 00  R <sub>1</sub> , R <sub>2</sub> hard wired to RF also dest <sup>n</sup> reg $\uparrow$ I[3:0] opcode to ALU					Reg <sub>2</sub>			Reg <sub>1</sub>							0	0	0	0	1	ADD
															0	0	0	1	0	SUB
															0	0	0	1	1	MUL
															0	0	1	0	0	DIV
															0	0	1	0	1	AND
															0	0	1	1	0	OR
															0	0	1	1	1	XOR
Just reg_wrt_en = 1			X	X	X	X		Reg <sub>1</sub>							0	1	0	0	0	NOT } r <sub>1</sub> , r <sub>2</sub>
1st reg_wrt_en = 1			X	X	X	X		r <sub>1</sub>							0	1	0	0	1	INC } r <sub>1</sub> , r <sub>2</sub>
			X	X	X	X									0	1	0	1	0	DEC } r <sub>1</sub> , r <sub>2</sub>
ALU will do XOR if carry = 0 $\rightarrow$ keep reg_wrt_en = 0 $\rightarrow$ carry = 1 $\rightarrow$	A <sub>6</sub> A <sub>4</sub>				Reg <sub>2</sub>			Reg <sub>1</sub>							0	1	1	1	1	$\star$ BEQ } r <sub>2</sub> , r <sub>3</sub> , address
															1	0	1	1	1	$\star$ BNE }
reg_wrt_en = 0   pc_ld = address reg_wrt_en = 0   mem_addr = SP, wrt_en = 1 wrt_data = PC + 1 pc_ld = address SP = SP - 1															0	1	0	1	1	$\star$ JMP } address (8)
															0	1	1	0	0	$\star$ CALL }
mem = a_addr + SP + 1 pc_ld = mem_rdata SP = SP + 1															0	1	1	0	1	$\star$ RET
mem_addr = address, max_sel = 1 reg_wrt_en = 1   get reg 2 data from register mem_addr = address wrt_data = reg_rdata mem_wrt_en = 1					Address			Dest <sup>n</sup> Reg							0	1	1	1	0	$\star$ LD } r <sub>1</sub> , address
					Reg <sub>2</sub>			Address							1	0	0	0	1	$\star$ ST } address, r <sub>1</sub>
															0	0	0	0	0	NOP (000FF)
					$\star$ r <sub>2</sub>			r <sub>1</sub>							1	0	0	1	0	LD' r <sub>1</sub> , $\star$ r <sub>2</sub>
					r <sub>2</sub>			$\star$ r <sub>1</sub>							1	0	0	1	1	ST' $\star$ r <sub>1</sub> , r <sub>2</sub>
															1	0	1	1	0	ENC r <sub>10</sub> , r <sub>11</sub>
															1	0	1	1	1	DEC r <sub>10</sub> , r <sub>11</sub>

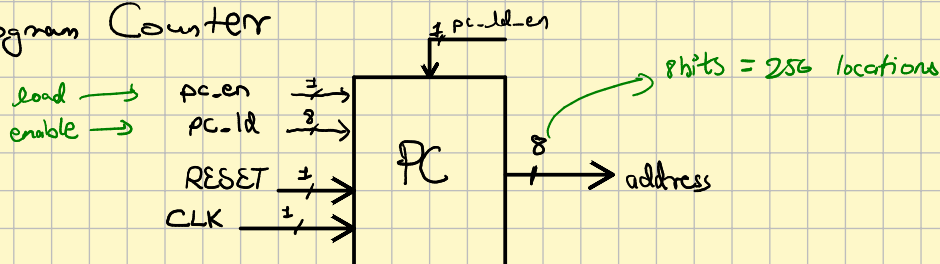
### 1) ALU Design:

takes  
0000  
to  
1001



if connections to A & B are given through signed wires it can handle -res

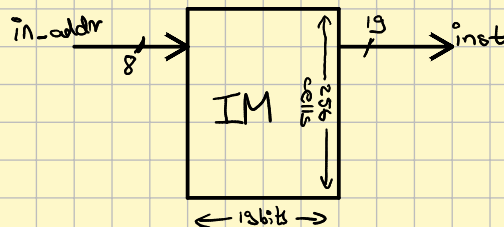
### 2) Program Counter



the load instructions only loads when it changes its value from a static value  
So if you jump to a location dont change the pc\_ld after the jump is completed.

### 3) Instruction memory

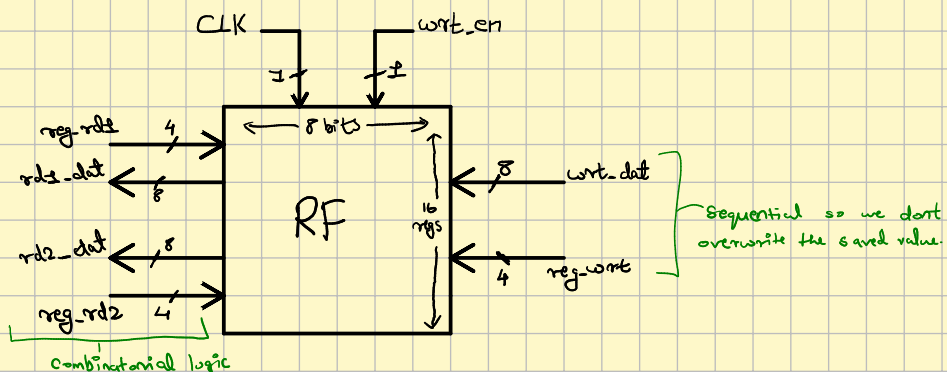
we will store the instructions in it.



- The instructions can be either stored manually or using a file.

want synthesize this feature in hardware?

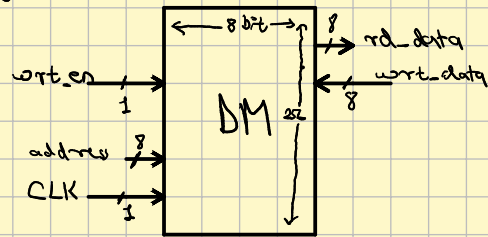
### 4) Register File:



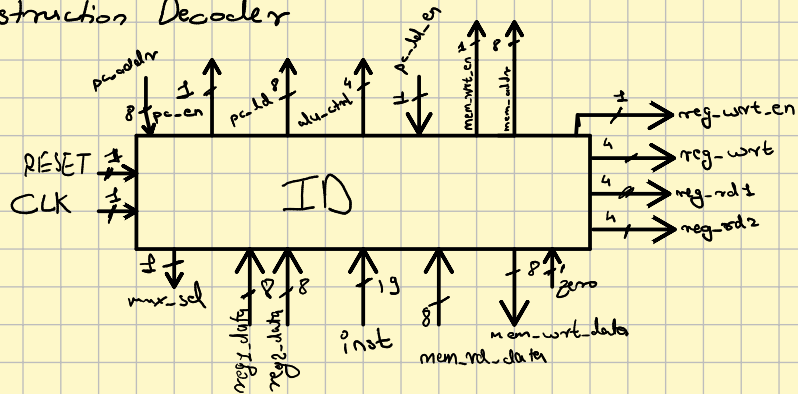
sequential so we dont overwrite the saved value.

- Total 16 registers from r1 to r16 can be selected by 0000 to 1111.
- 2 data read lines and one data write line.

## 5) Data Memory:

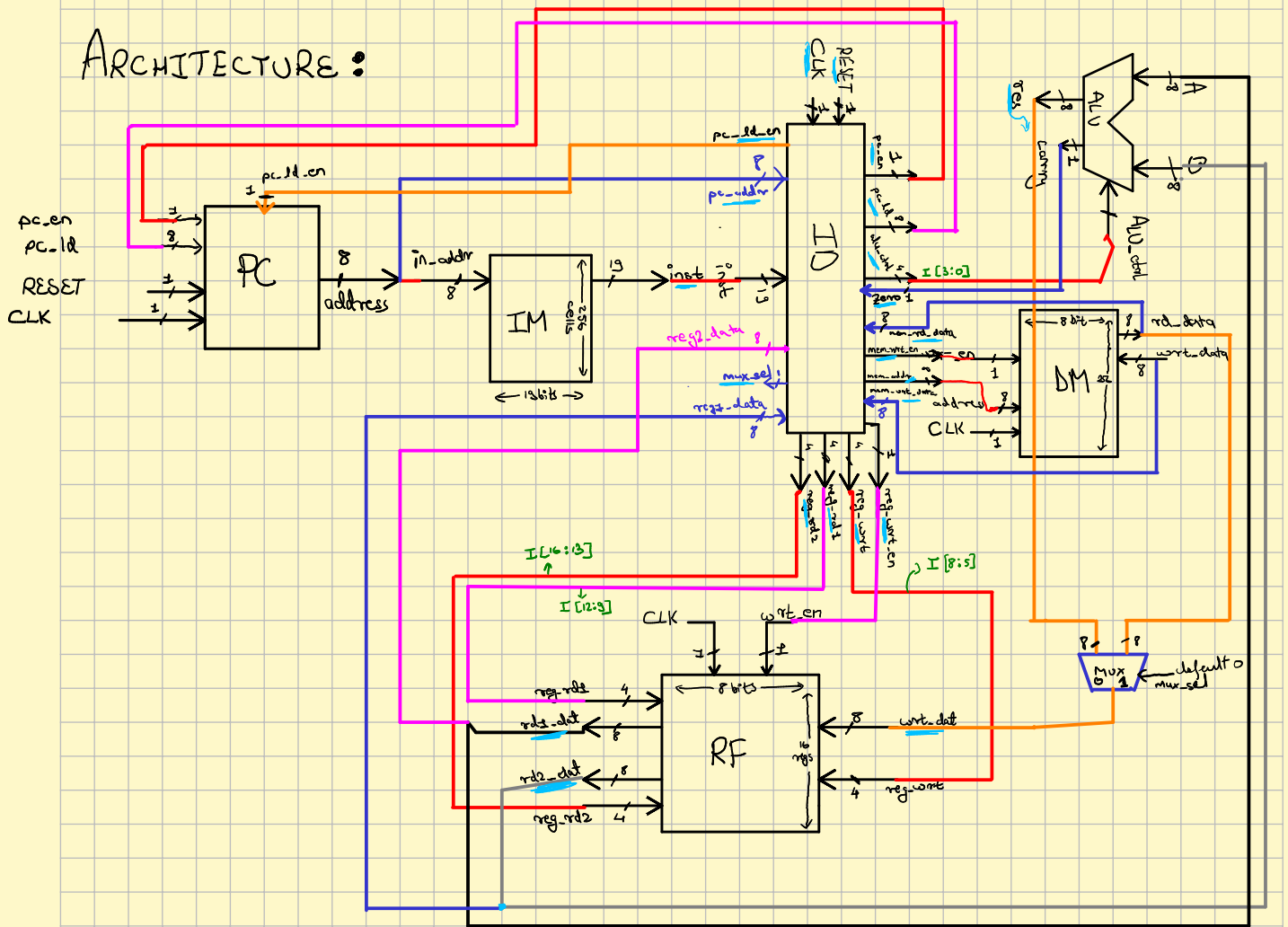


## 6) Control/Instruction Decoder



★ Name of wire in CPU\_top

# ARCHITECTURE :





$R_0 = 12, R_1 = 14$ 

ADDR

0	ADD $r_2, r_0, r_1$	$r_2 = 16$	02041
1	SUB $r_3, r_0, r_1$	$r_3 = 8$	02062
2	ST 00, $r_1$	00 = 4	02011
3	NOP		000FF
4	MUL $r_4, r_2, r_3$	$r_4 = 128$	06483
5	BNE $r_1, r_2, 20$	Jump to 20	24237
6	NOP		000FF
7	AND $r_6, r_1, r_0$	$r_6 = 4$	002C5
8	JMP 14	Jump to 14	001CB
9	NOP		000FF
10	LD $r_9, 00$	$r_9 = 4$	0012E
11	NOP		000FF
12	JMP 24 <del>CALL 124 BRC <math>r_2, r_1</math></del>	Jump to 24 <del>to 124</del>	0030B <del>0044A</del>
13	NOP		000FF
14	OR $r_7, r_3, r_6$	$r_7 = 12$	0C6E6
15	JMP 03	Jump to 03	0012B
16	NOP		000FF
17	NOP		000FF
18	<del>0000</del>		0000
19	0000		0000
20	DIV $r_5, r_4, r_2$	$r_5 = 8$	048A4
21	BEQ $r_1, r_1, 07$	Jump to 07	022EF
22	NOP		000FF
23	NOP		000FF
24	NOT $r_{10}, r_9$	$r_{10} = 111$	01148
25	INC $r_6, r_6$	$r_6 = 5$	00CC9
26	CALL 30		003CC
27	NOP		000FF
28	0000	<u>halt</u>	00000
29	0000		00000
30	INC $r_1, r_1$	$r_1 = 5$	00223
31	RET	back to 27	0000D
32	NOP		000FF



NOP  
ENC  $r_{10}, r_{11}$   
NOP  
NOP  
SUB  $r_{11}, r_{10}, r_{08}$   
INC  $r_{10}, r_{10}$   
ADD  $r_{14}, r_8, r_{15}$   
NOP  
DEC  $r_{10}, r_{11}$   
NOP

$r_8 = 49$  000FF  
 $r_9 =$  01754  
 $r_{10} = 80$  000FF  
 $r_{11} = 00$  000FF  
 $r_{12} =$  11562  
 $r_{13} = BC$  015430  
 $r_{14} = 49$  1F1C1  
 $r_{15} = 0030$  000FF  
900 01755  
900 000FF

63 JMP 78  
64 NOP  
65 LD'  $r_9, r_{11}$   
66 NOP  
67 XOR  $r_{12}, r_9, r_{13}$   
68 ~~ST'  $r_{10}, r_{12}$~~  NOP  
69 ST'  $r_{10}, r_{12}$   
70 NOP  
71 INC  $r_{10}, r_{10}$   
72 INC  $r_{11}, r_{11}$   
73 DEC  $r_{14}, r_{14}$   
74 NOP  
75 BNE  $r_{14}, r_{15}$   
76 NOP  
77 RET  
78 NOP

009CB  
000FF  
01732  
000FF  
1B387  
000FF  
01953  
000FF  
01543  
01769  
01DCA  
000FF  
~~7FE88~~ 7FDF7  
000FF  
0000D  
000FF