# BUILDING A RISC-V CORE

Computer Organization – IUST Spring 2023

IUST
Iran University of
Science and Technology

LinkedIn : arvin-delavari faraz-ghoreishy

E-Mail : arvin7807@gmail.com farazghoreishy@gmail.com

iustCompOrg+4012@gmail.com

## Building a CPU

1 – **Choosing Instruction Set Architecture(ISA)**
* In RISC-V we should chose extensions too!

2 – **Defining Microarchitecture**
* Block diagrams and structure of the CPU

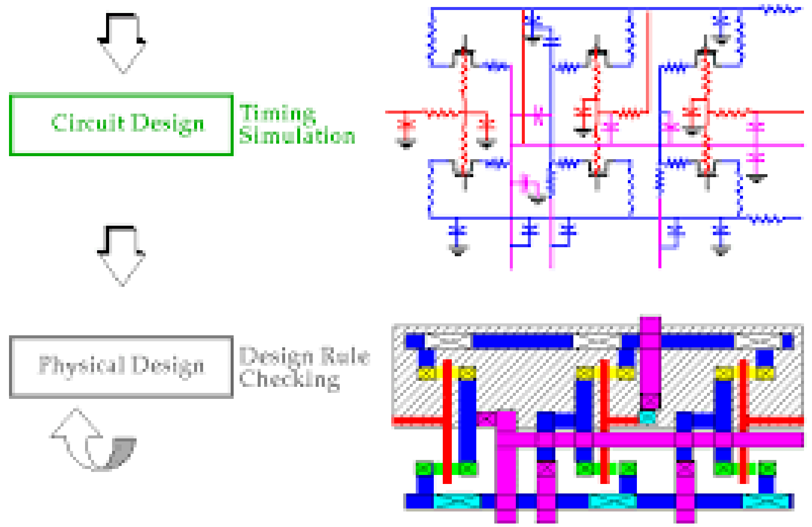
3 – **Creating a clear Datapath**
* Pipelining can be implemented here!

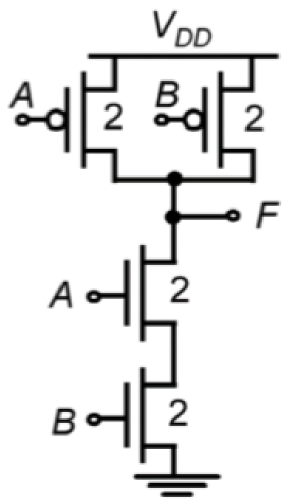
4 – **RTL programming, synthetize and simulations**
* We'll use Verilog HDL ,but won't implement on FPGA (Only simulation)

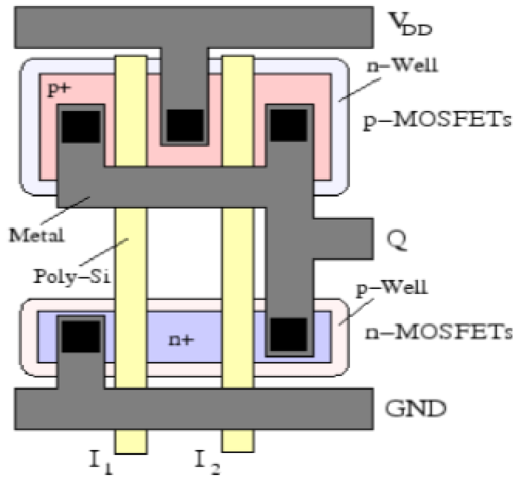
# Building a CPU

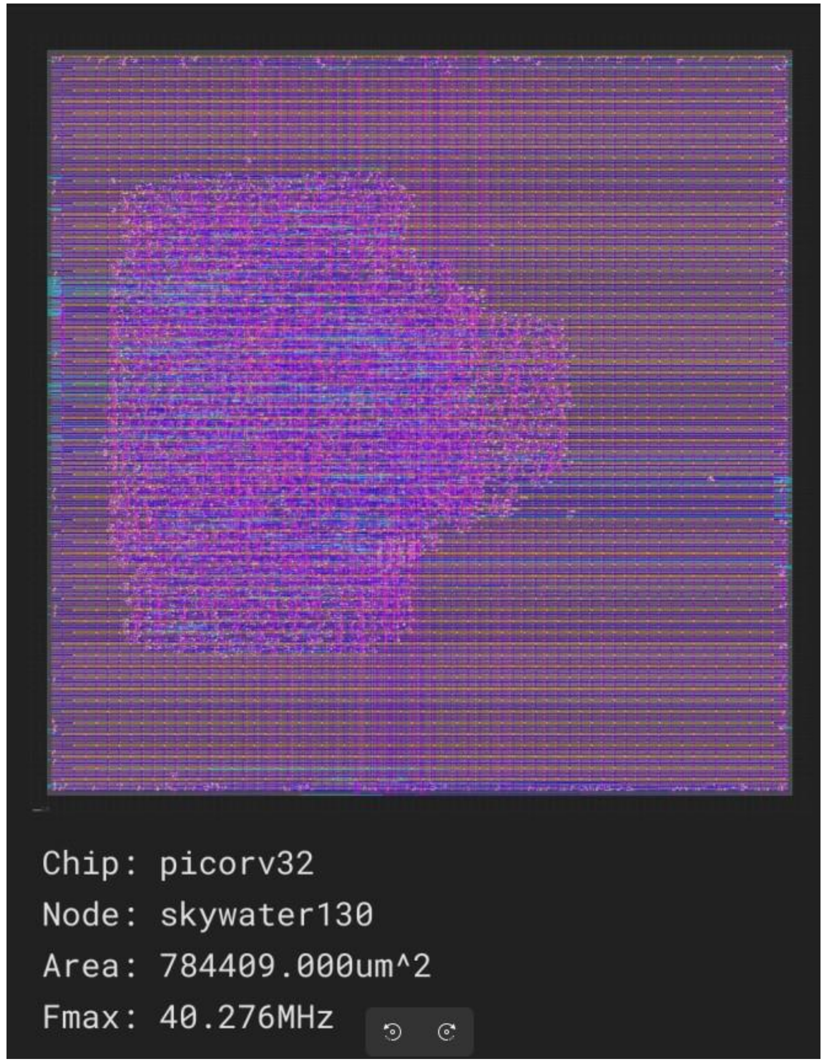
## 5 – After the logic and RTL design → Netlist output → VLSI

* We are not covering this part in this course

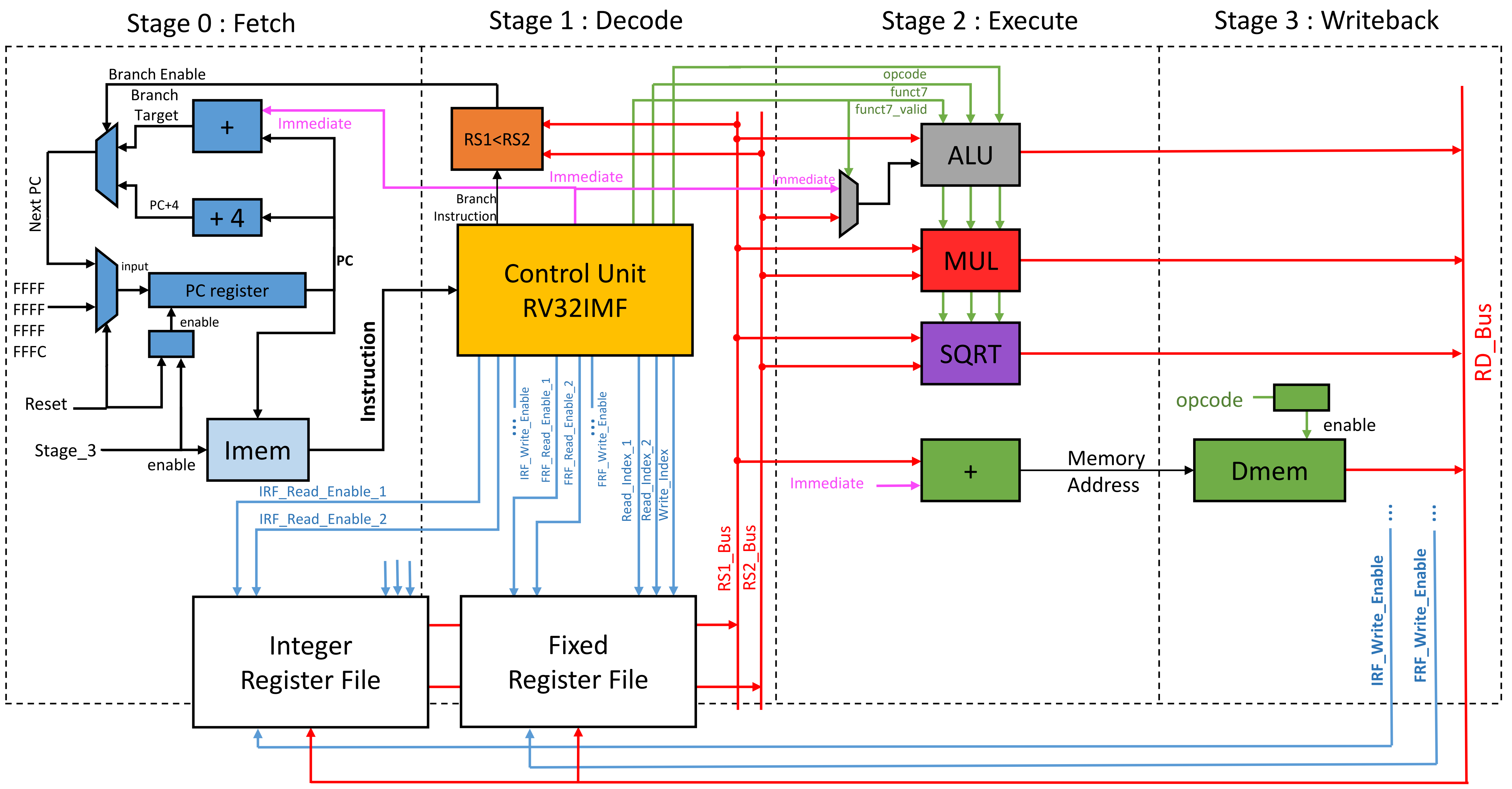


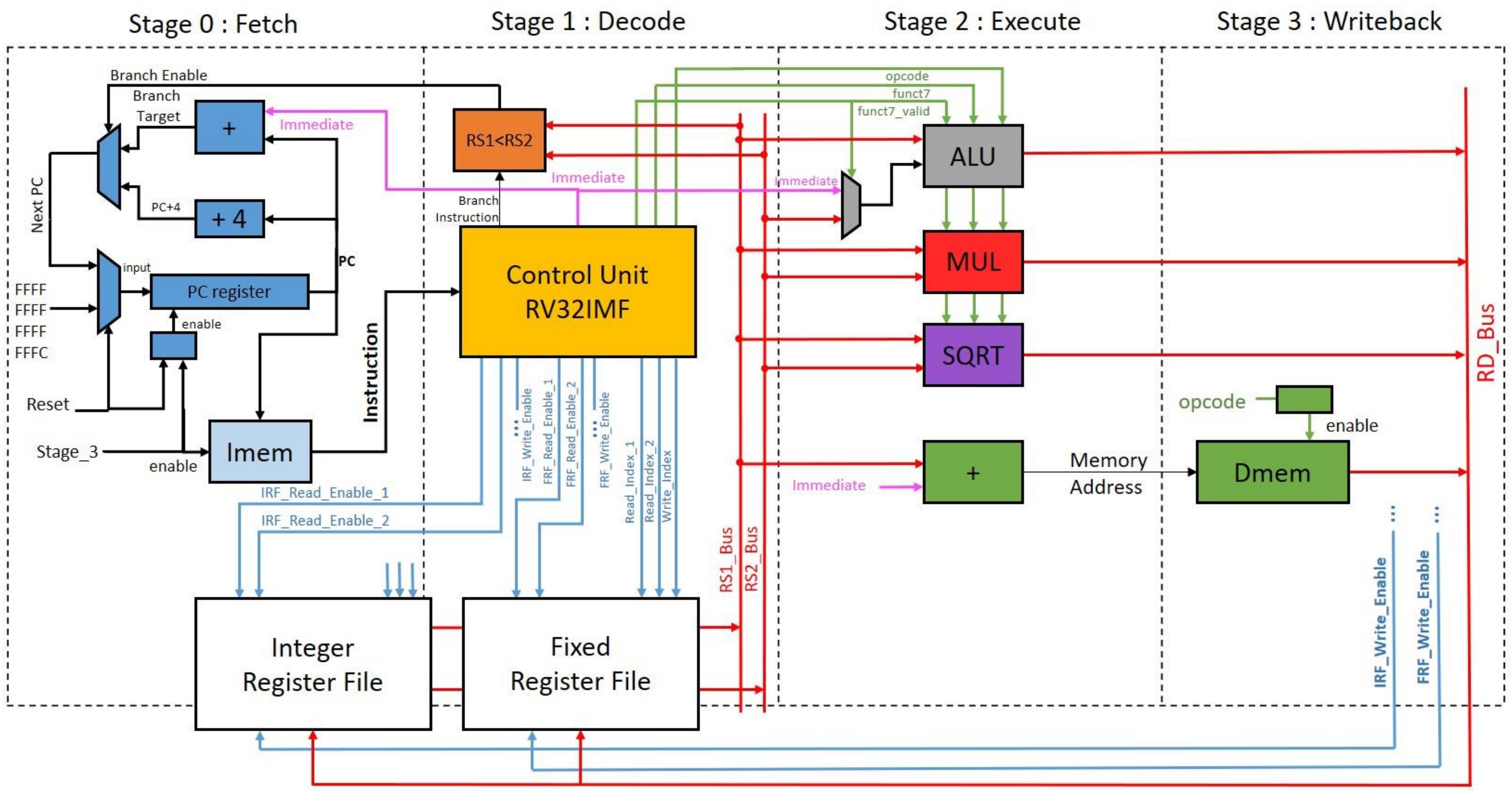






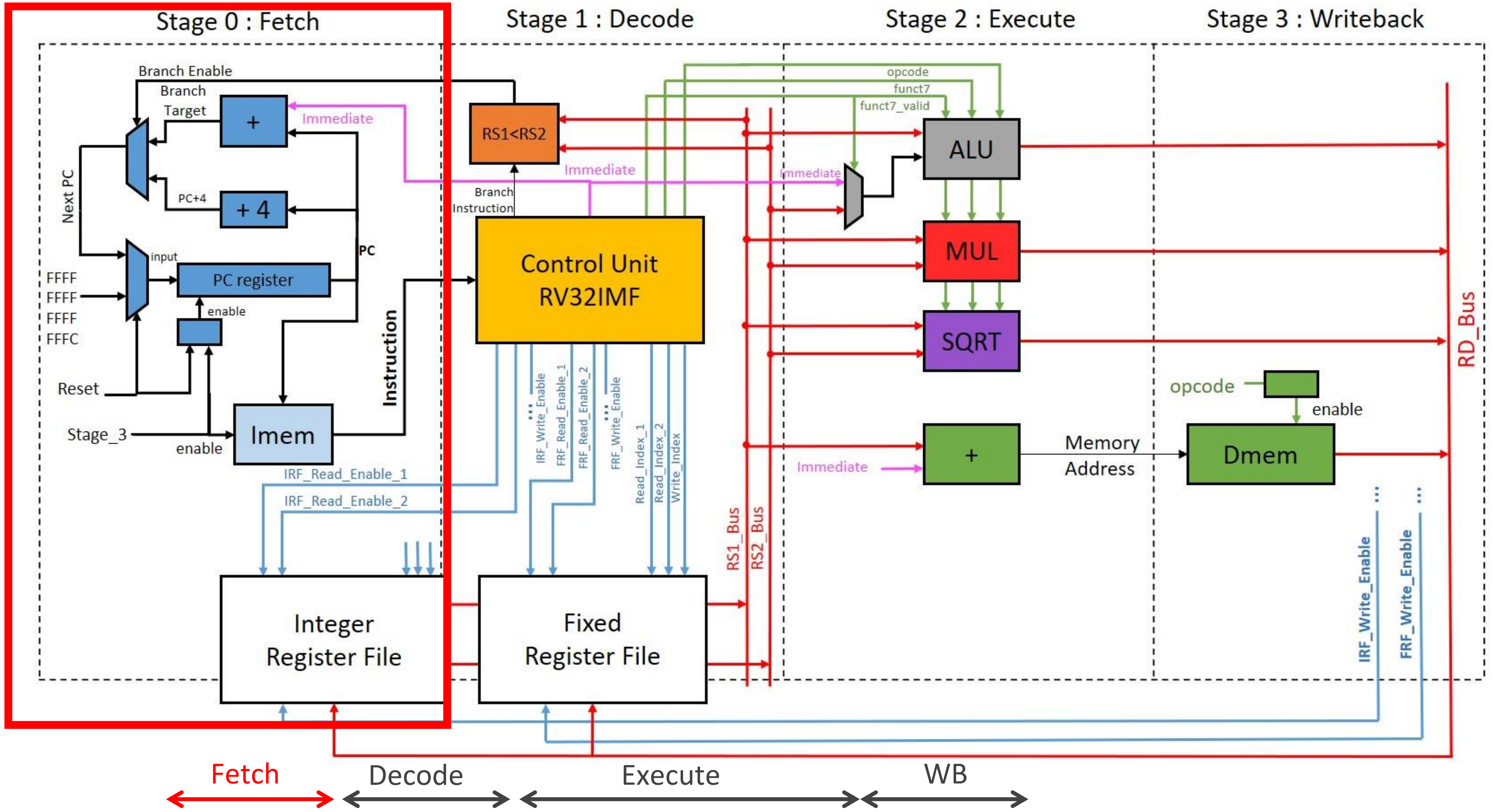
```
Chip: picorv32
Node: skywater130
Area: 784409.000um^2
Fmax: 40.276MHz
```


Stage 0 : Fetch
Stage 1 : Decode
Stage 2 : Execute
Stage 3 : Writeback
Branch Enable
Branch
Target
+
Immediate
PC+4
+ 4
Next PC
input
PC register
PC
enable
FFFF
FFFF
FFFF
FFFC
Reset
Stage_3
enable
Imem
Instruction
RS1<RS2
Branch
Instruction
Immediate
Control Unit
RV32IMF
opcode
funct7
funct7_valid
Immediate
ALU
MUL
SQRT
+
Immediate
Memory
Address
opcode
enable
Dmem
RD_Bus
IRF_Write_Enable
FRF_Read_Enable_1
FRF_Read_Enable_2
FRF_Write_Enable
Read_Index_1
Read_Index_2
Write_Index
IRF_Read_Enable_1
IRF_Read_Enable_2
RS1_Bus
RS2_Bus
Integer
Register File
Fixed
Register File
IRF_Write_Enable
FRF_Write_Enable

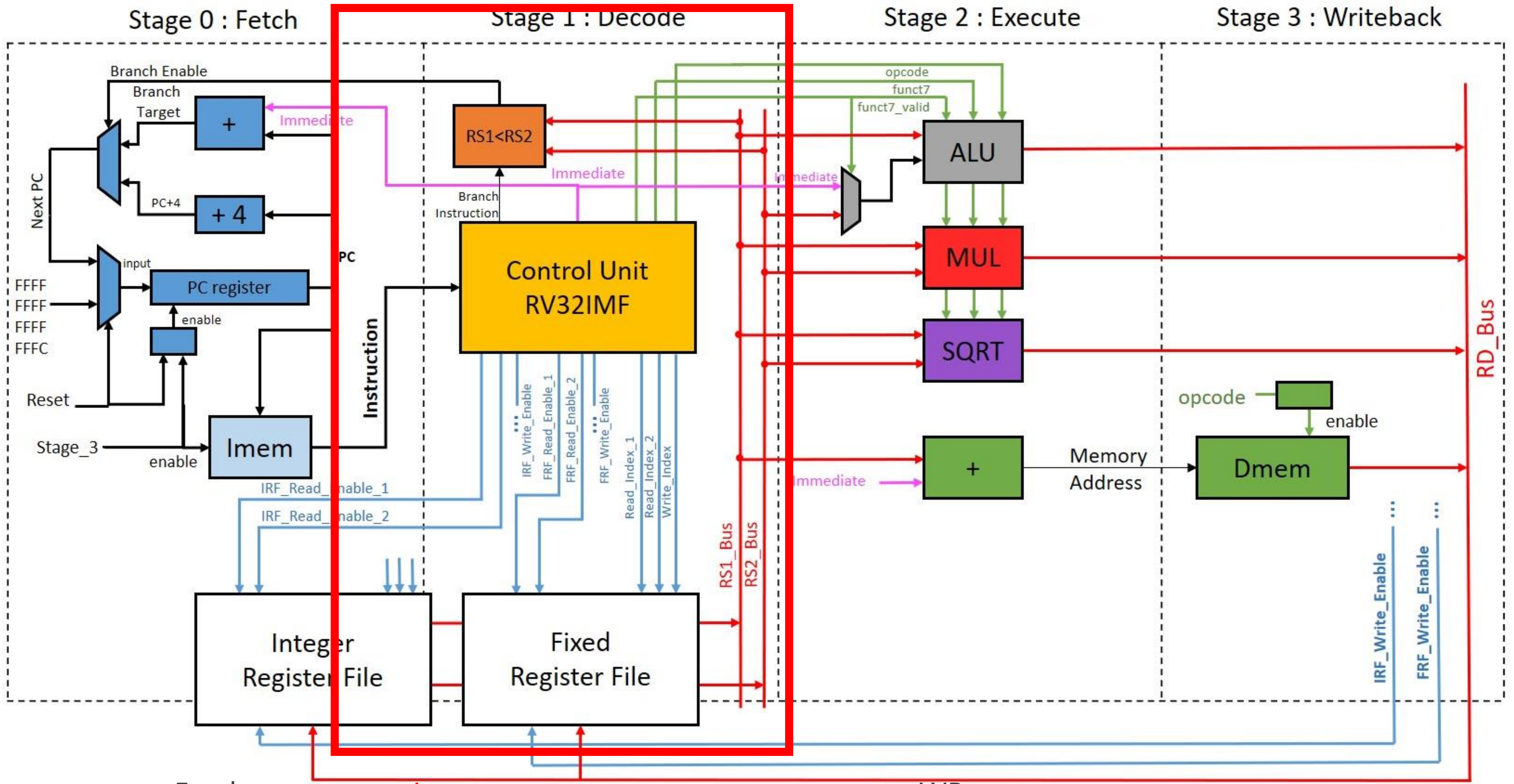


Our **RV32IMF** (F stands for fixed-point, not floating point) datapath have 4 stages:

**1 – Fetch** **2 – Decode** **3 – Execute** **4 – Writeback**

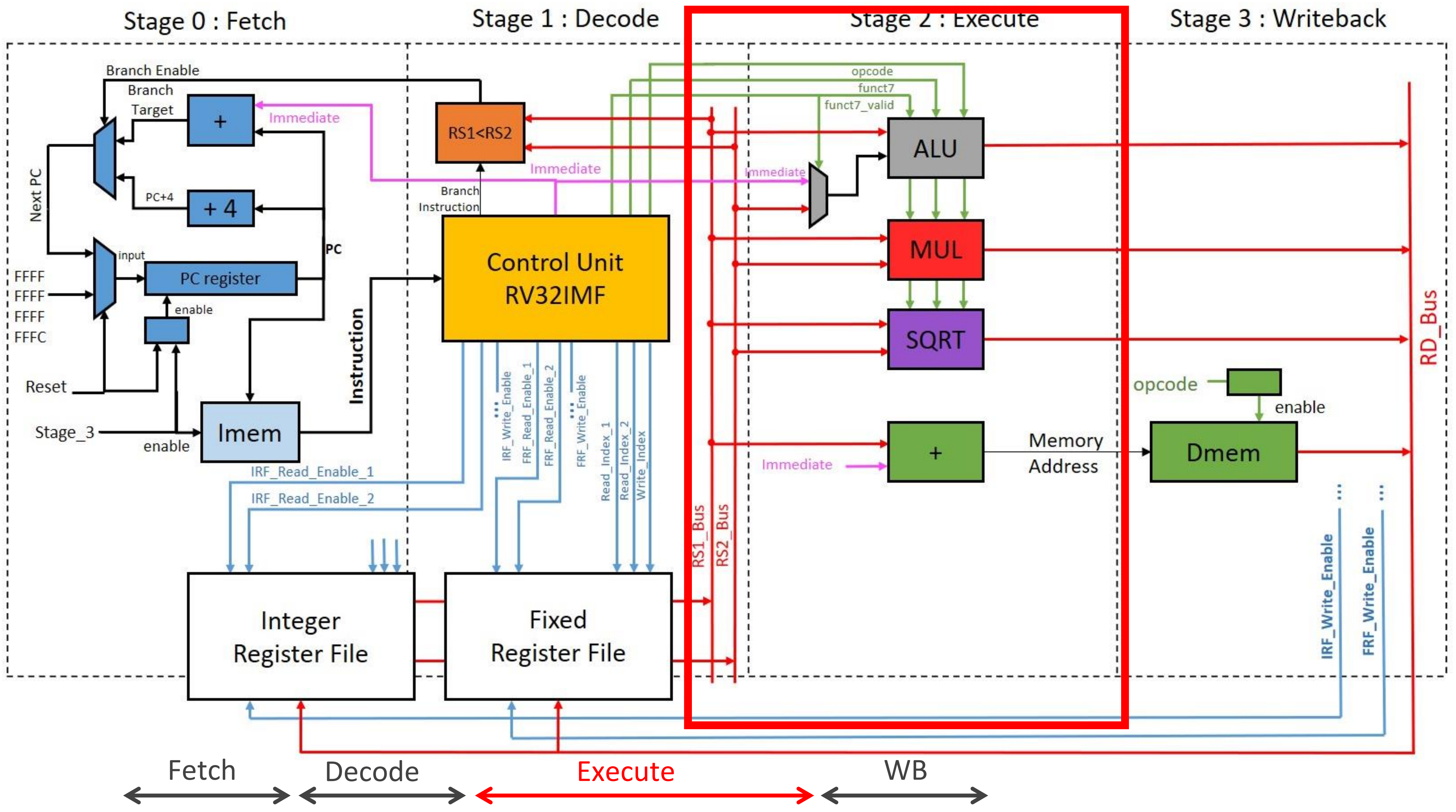


Fetch 32-bit instruction from memory + Increment PC = PC + 4

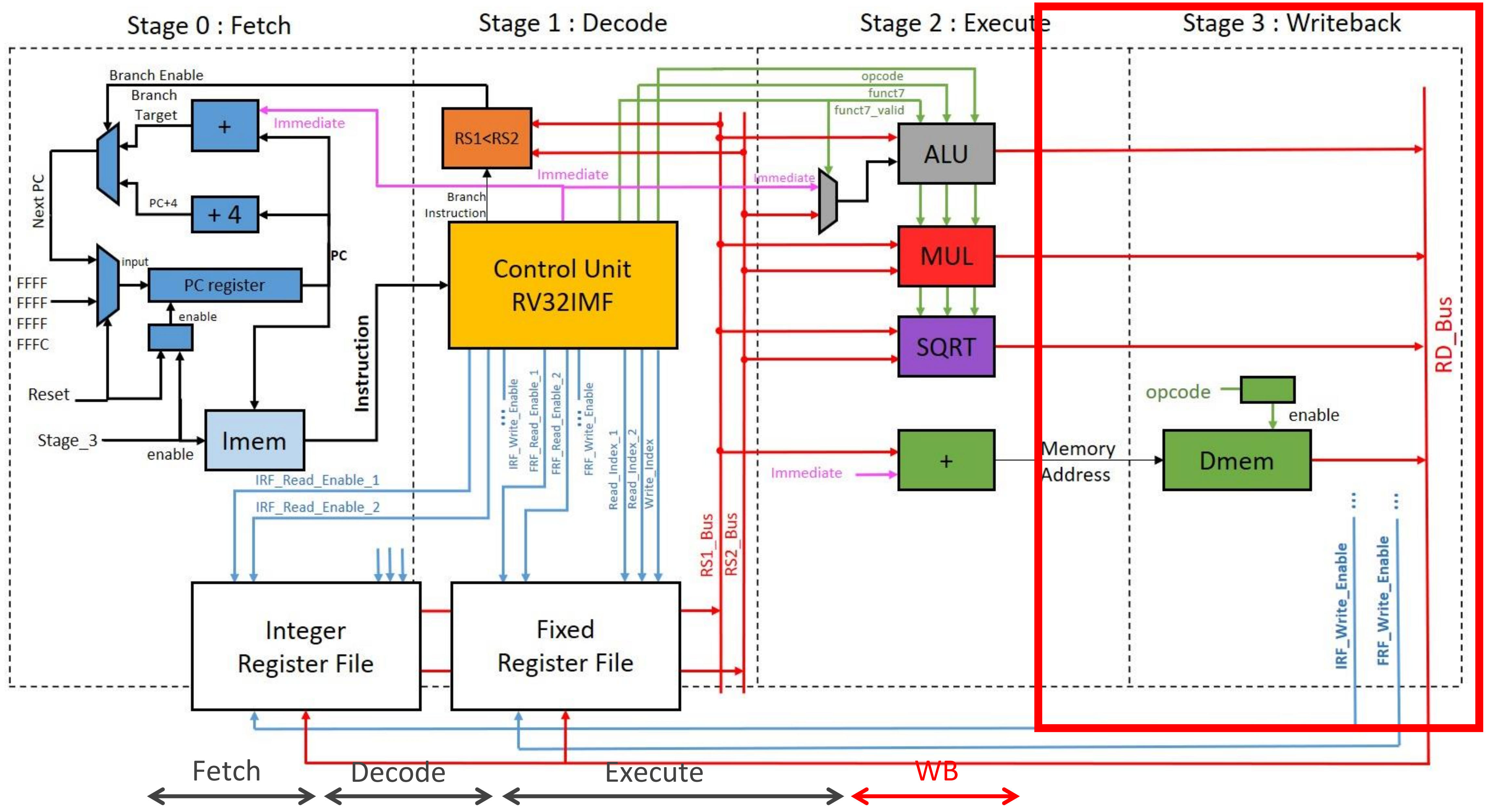


Fetch
Decode
Execute
WB

- Gather data from the instruction + handle branch instructions
- Read opcode; determine instruction type, field lengths
- Read in data from register files (F and I Reg Files)



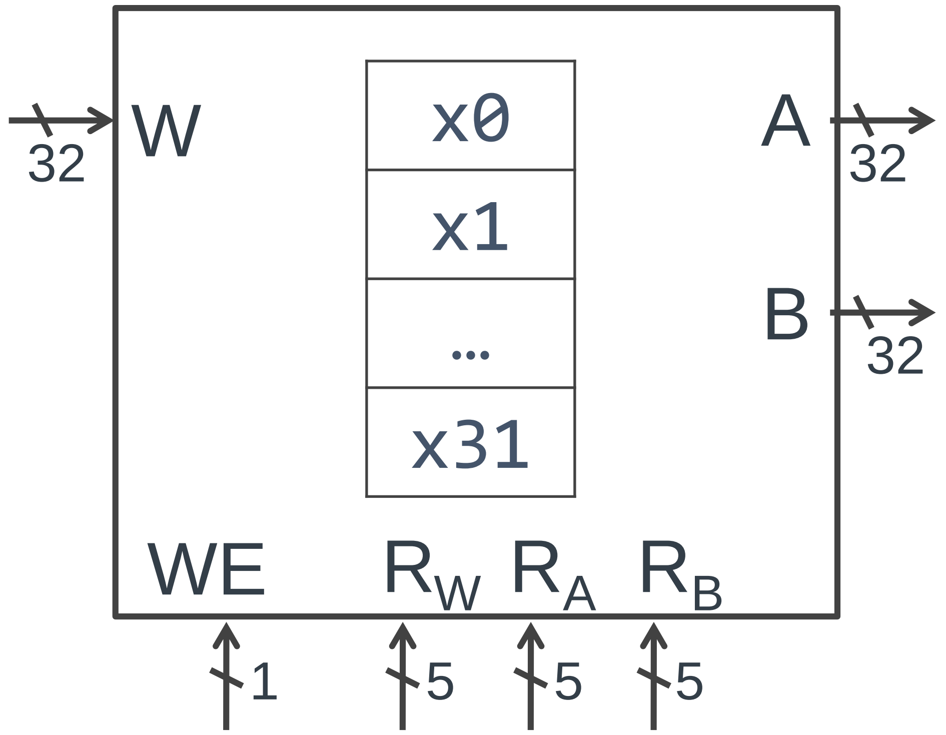
- Useful work done here (+, -), shift, logic operation, …
- In our CPU : SQRT and MUL are separate modules form ALU
- We have FADD, and FCVRT functions too!



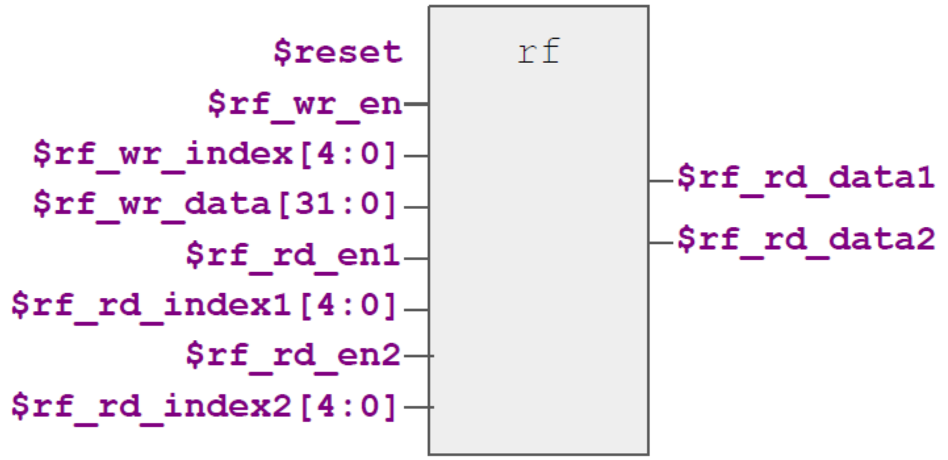
- We have load unit in this part!
- Writeback to register files
- IRF and FRF Write_Enable signals are generated in this part!

## RISC-V register file

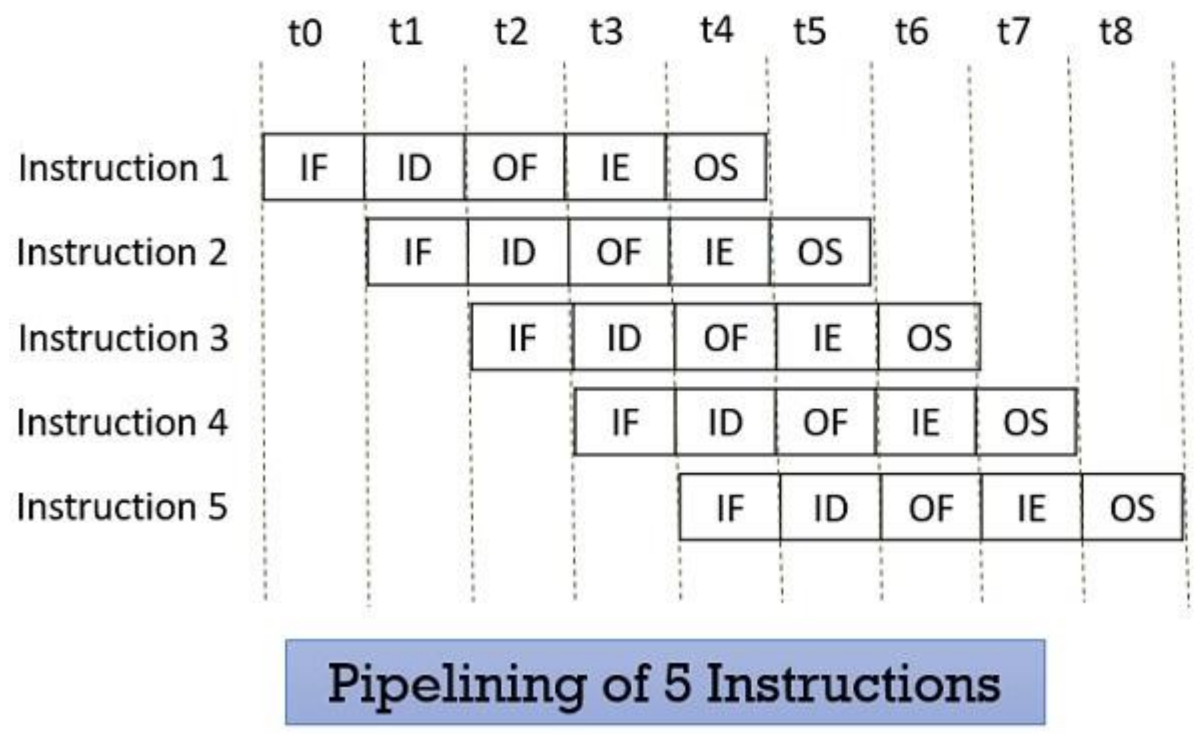
- 32 registers, 32-bits each
- x0 wired to zero
- Write port indexed via $R_W$
  - on falling edge when WE=1
- Read ports indexed via $R_A$, $R_B$
- Numbered from 0 to 31
- Can be referred by number: x0, x1, x2, … x31
- Convention, each register also has a name:
  - x10 – x17 → a0 – a7, x28 – x31 → t3 – t6



2-read, 1-write register file:

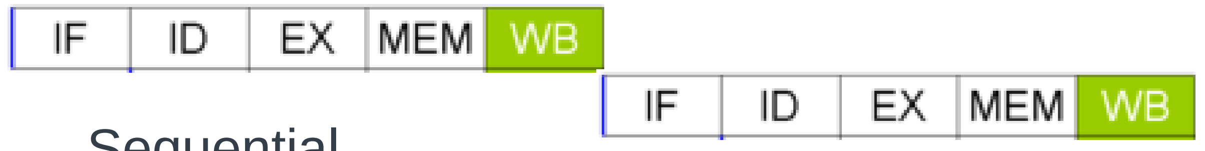


## Pipelining

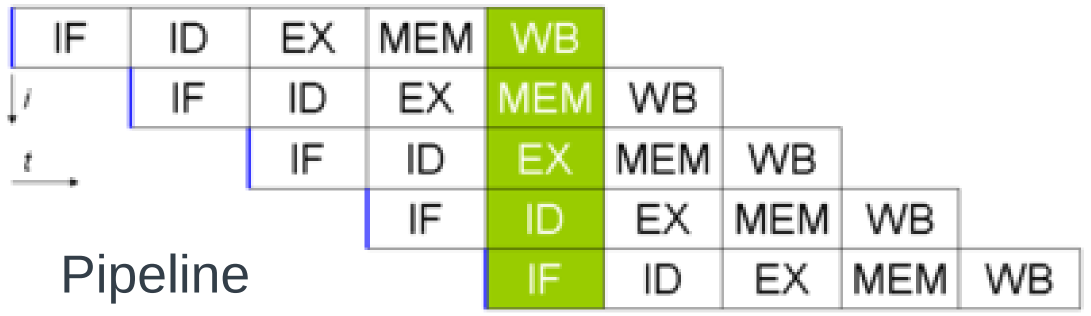


|  | t0 | t1 | t2 | t3 | t4 | t5 | t6 | t7 | t8 |
|---|---|---|---|---|---|---|---|---|---|
| Instruction 1 | IF | ID | OF | IE | OS |  |  |  |  |
| Instruction 2 |  | IF | ID | OF | IE | OS |  |  |  |
| Instruction 3 |  |  | IF | ID | OF | IE | OS |  |  |
| Instruction 4 |  |  |  | IF | ID | OF | IE | OS |  |
| Instruction 5 |  |  |  |  | IF | ID | OF | IE | OS |

Pipelining of 5 Instructions

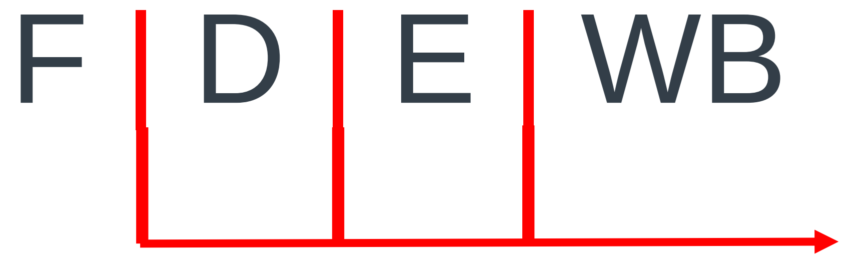


Sequential



Pipeline

## Pipelining



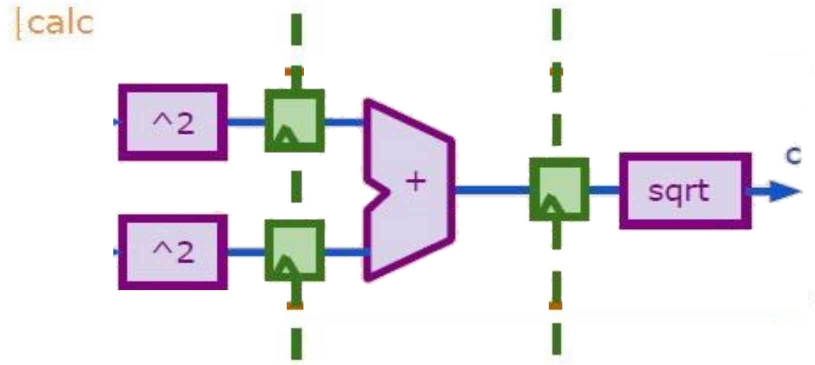
Latch : keeps data for next operation

$$Speed\ Up = \frac{T_s\ (Sequential)}{T_p\ (Pipeline)}$$

$$\tau = \frac{1}{f}(clk) \rightarrow \frac{NK\tau}{K\tau + (N-1)\underset{Max\ \tau_i}{\boxed{\tau}}} = \frac{NK}{N+K-1},\ \lim_{N\to\infty} S = K$$

$$\mathrm{N} = \text{Number of instruction}\ ,\mathrm{K} = \text{stages of pipeline}\ ,\tau = \text{clk}$$

## Pipelining



System Verilog

```
// Calc Pipeline
logic [31:0] a_C1;
logic [31:0] b_C1;
logic [31:0] a_sq_C1,
             a_sq_C2;
logic [31:0] b_sq_C1,
             b_sq_C2;
logic [31:0] c_sq_C2,
             c_sq_C3;
logic [31:0] c_C3;
always_ff @(posedge clk) a_sq_C2 <= a_sq_C1;
always_ff @(posedge clk) b_sq_C2 <= b_sq_C1;
always_ff @(posedge clk) c_sq_C3 <= c_sq_C2;
// Stage 1
assign a_sq_C1 = a_C1 * a_C1;
assign b_sq_C1 = b_C1 * b_C1;
// Stage 2
assign c_sq_C2 = a_sq_C2 + b_sq_C2;
// Stage 3
assign c_C3 = sqrt(c_sq_C3);
```

# Pipelining Hazards :

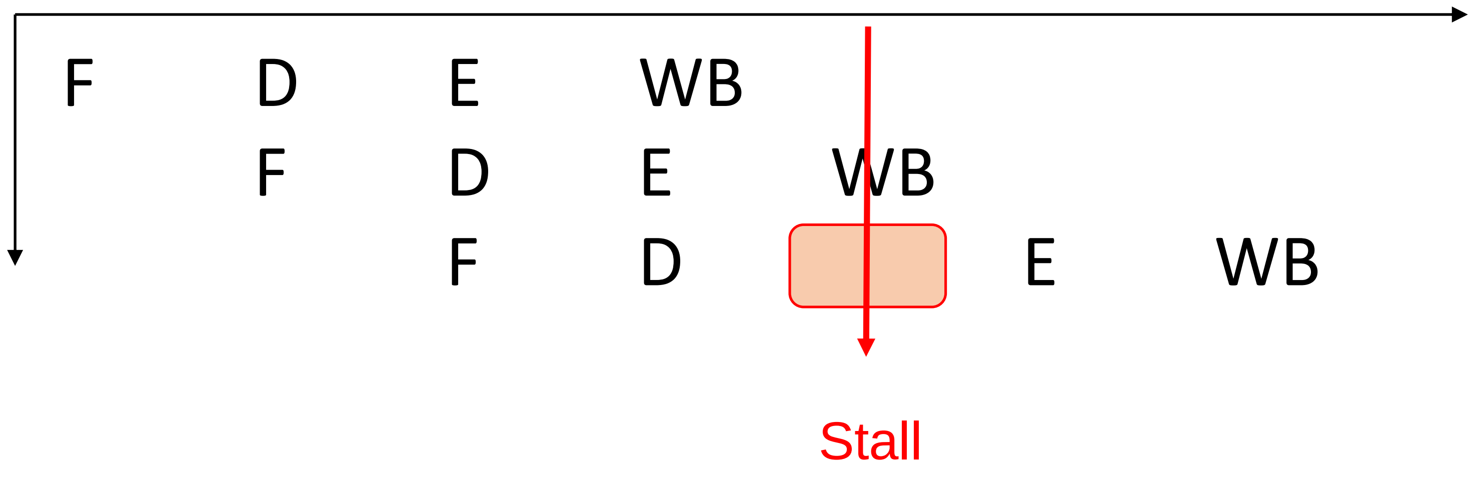
Hazards and Bubbles in pipelining:

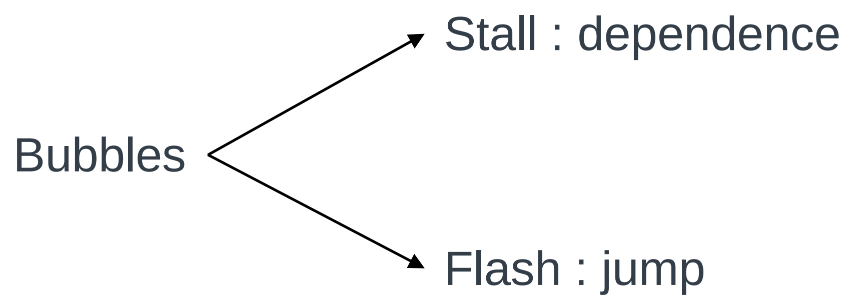
I1 : R1 ← 40

I2 : R2 ← 41

I3 : R3 ← R1 + R2

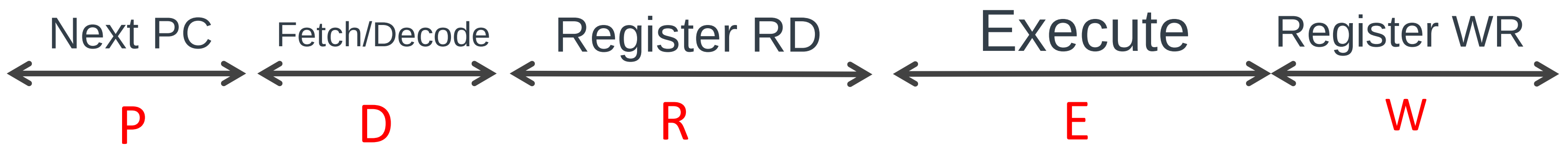
I4 : 42 ← R3





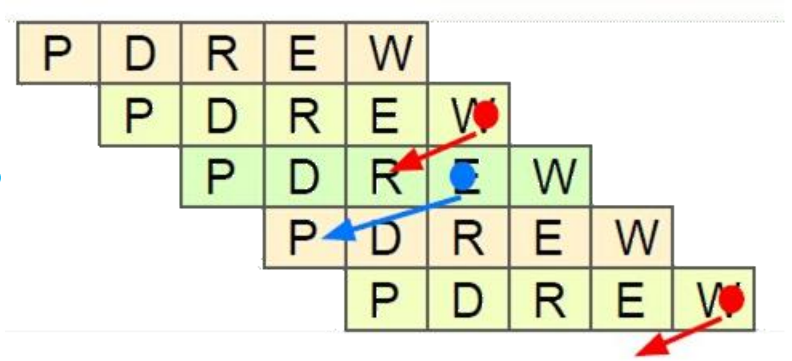
## Pipelining

- In computer engineering, instruction pipelining is a technique for implementing instruction-level parallelism within a single processor.



- RISC-V Waterfall diagram and hazards

```
loop : add a4,a3,a4
       addi a3,a3,1
       blt a3,a2,loop
loop : add a4,a3,a4
       addi a3,a3,1
       ...
```


P D R E W
P D R E W
P D R E W
P D R E W
P D R E W