PROCESSOR DESIGN

A DISSERTATION SUBMITTED IN PARTIAL FULFILMENT
FOR THE COURSE OF

E3 245 PROCESSOR SYSTEM DESIGN

In

Master of Technology

In The
Faculty Of Engineering

By

Ganesh Kumar Shaw

Guided By

Prof. Kuruvilla Varghese

Department of Electronic Systems Engineering

Indian Institute Of Science, Bangalore

AUGUST 2022

Notations

Contents

List of Figures

,

Chapter 1

Introduction

1.1 What is CPU?

CPU stands for Central Processing Unit, and it's often called the brain of the computer. The CPU sits at the center of everything in the computer and handles all of the computation needed to turn inputs from memory, like a photo from a hard drive, into the output on the peripherals, like an image on the monitor.

CPUs are a general purpose flexible architecture that takes in a stream of instructions from all types of workload and compute or process information based on those instructions. Simply put, CPUs do what we tell them or program them to do.

The way we use to command the CPUs is called Instruction Set and the art of designing Instruction Set is called Instruction Set Architecture.

1.2 Instruction Set Architecture

[1]Instruction set architecture plays a very crucial role and has been defined as a contract between the software and the hardware, or between the program and the machine. By having the ISA as a contract, programs and machines can be developed independently. Programs can be developed that target the ISA without requiring knowledge of the actual machine implementation. Similarly, machines can be designed that implement the ISA without concern for what programs will run on them. Any program written for a particular ISA should be able to run on any machine implementing that same ISA. The notion of maintaining the same ISA across multiple implementations of that ISA was first introduced with the IBM S/360 line of computers

[Amdahl et al., 1964].

Besides serving as a reference targeted by software developers or compilers, ***ISA serves as the specification for processor designers.*** Microprocessor design starts with the ISA and produces a microarchitecture that meets this specification.

Chapter 2

Study

2.1 Instruction Set Processor Design

The focus of this project is on designing instruction set single cycle processors. Critical to an instruction set processor is the instruction set architecture, which specifies the functionality that must be implemented by the instruction set processor. The ISA plays several crucial roles in instruction set processor design.

2.2 Instruction Formates

In the base ISA, there are four core instruction formats (R/I/S/U), as shown in Figure **??**. All are fixed 32 bits in length and must be aligned on a four-byte boundary in memory. An instruction address misaligned exception is generated on a taken branch or unconditional jump if the target address is not four-byte aligned. No instruction fetch misaligned exception is generated for a conditional branch that is not taken.

2.3 RISC-V Registers

- RISC-V consists of 32 General Purpose Registers.
  - *x0 always contains 0.*
  - all other registers are treated identically by the ISA.

- There is also a program counter(PC).
- *The RISC-V processor doesn't contain any status register.* Instead, the branch instruction in RISC-V will perform both the test as well as conditional jump.
- **RISC-V uses 32-bit addresses, and memory is byte-addressed.**

2.4 RISC-V Instruction Encoding

- Instruction width is 32-bits.
- The least significant 2 bits always indicate instruction type(32-bits or 16 bits). 11 pattern indicates it is 32 bits instruction set while the other pattern(00 01 10) indicates compressed instruction. *Therefore 30-bits is the effective width of the instruction.*

2.5 RISC-V Instruction Formate

2.5.1 R-Type

- In R-Type instruction there are two source registers(rs1, rs2) and one destination register(rd). These registers work as operands but source registers are meant for loading the inputs while destination register is for loading the output.

31 25	24 20	19 15	14 12	11 7	6 0
func7	rs2	rs1	func3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Table 2.1: R-Type Formate

- func7(7 bits), func3(3 bits), and opcode(6-2) together combination can make a different function(Arithmetic, logic etc) of R-Type.
- Instruction Example:
  - add R1, R2, R3 $\backslash\backslash R3 = R1 + R2$

2.5.1.1 Few RV32 R-formate instructions

31 25	24 20	19 15	14 12	11 7	6 0	
func7	rs2	rs1	func3	rd	opcode	Operation
000_0000	rs2	rs1	000	rd	011_0011	ADD
010_0000	rs2	rs1	000	rd	011_0011	SUB
000_0000	rs2	rs1	100	rd	011_0011	XOR
000_0000	rs2	rs1	110	rd	011_0011	OR
000_0000	rs2	rs1	111	rd	011_0011	AND
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	

Table 2.2: R-Type Formate Instructions

2.5.2 I-Type

- I-Type meant for immediate value means if we want to write the operand data in the ISA itself instead of fetching from memory. It could be operand data like use for addition($R1 = R2 + 50$) or bias value in case of load and store but there is another class of ISA called U-Type for loading a direct value into the register.

31 20	19 15	14 12	11 7	6 0
immediate	rs1	func3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

Table 2.3: I-Type Formate

- Instruction Example:
  - addi rd, rs1, imm $\backslash\backslash\, rd = rs1 + imm$
  - ld rd, imm(rs1) $\backslash\backslash\, rd = mem[rs1 + imm]$ rs1 is the base register and imm is the offset. Their sum results in the address of memory, from where data is being fetched and will be stored into rs1.

2.5.2.1 Few RV32 I-formate Instructions

31 20	19 15	14 12	11 7	6 0	
immediate	rs1	func3	rd	opcode	Operation
immediate	rs1	001	rd	001_0011	**LD**
immediate	rs1	000	rd	001_0011	ADI
12 bits	5 bits	3 bits	5 bits	7 bits	

Table 2.4: I-Type Formate Instructions

2.5.3 U-Type

- In a U-Type instruction we can provide 20 bits of data(operand, address, bias, etc) into a register.

31 12	11 7	6 0
immediate	rd	opcode
20 bits	5 bits	7 bits

Table 2.5: U-Type Formate

- Instruction Example:
  - lui x1, 0x12345 $\backslash\backslash x1 = 0x12345$
  - Since, registers are 32 bits then either it needs to be extended the sign or insert 0.

2.5.3.1 Few RV32 U-formate Instructions

31 12	11 7	6 0	
immediate	rd	opcode	Operation
immediate	rd	011_0111	LDI
20 bits	5 bits	7 bits	

Table 2.6: U-Type Formate Instructions

2.5.4 S-Type

- S-Type instruction consists of opcode, two registers, function, and immediate value.

31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	rs2	rs1	func3	imm[4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Table 2.7: S-Type Formate

- Instruction Example:
  - sd rs2, imm(rs1) $\backslash\backslash mem[rs1 + imm] = rs2$
  - rs1 is the base register and imm here is the offset value, together providing the memory address where rs1 is being stored. This is called store instruction.

2.5.4.1 Few RV32 S-formate Instructions

31 25	24 20	19 15	14 12	11 7	6 0	
imm[11:5]	rs2	rs1	func3	imm[4:0]	opcode	Operation
imm[11:5]	rs2	rs1	010	imm[4:0]	010_0011	STR
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	

Table 2.8: S-Type Formate Instructions

2.5.5 SB-Type

- SB-Type stands for ”Branch”. In SB-Type ISA there is an ’opcode’ to specify the branch, ’two registers’ to compare the values, ’func’ for different types of branches (eq, neq, etc), and an immediate value of 12 bits.
- Since there are only two registers declared in the ISA so in this case, we have to use PC as base register in case the branch is true. But the problem is we can not change the PC value directly we have to use the existing PC value and modify the PC value according to demand. Suppose our required target is less than the PC value then we have to go back which is not possible if we use the immediate value as unsigned. Therefore we have to use 2’s complement for immediate value.

- There are 12 bits available for immediate so the range would be $\pm\ 2^{11}$
- Considering PC value as the base register is preferable rather than using an extra register for the base address.
- RISC-V uses 32-bits addresses, and *memory is byte addressed* so if in case branch true: $PC\ =\ PC\ +\ 4\ +\ imm*4$
- So all we need to shift imm left by 2 bits to get multiplication by 4.
- Therefore, in case of true branch: $PC = PC+4+IMM$ \\\\ where $IMM = \{imm[11:0], 2'b00\}$
- imm[0] is always going to be zero so we don't need to mention it expecitly. we can manage while performing operations.

31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	rs2	rs1	func3	imm[4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Table 2.9: SB-Type Formate

- Instruction Example:
  - beq x18, x10, imm \\\\ if $x18 = x10$ then $PC = PC+4+IMM$ where $IMM = \{imm[11:0], 2'b00\}$

31 25	24 20	19 15	14 12	11 7	6 0	
imm[11:5]	rs2	rs1	func3	imm[4:0]	opcode	Operation
imm[11:5]	rs2	rs1	000	imm[4:0]	110_0011	BEQ
imm[11:5]	rs2	rs1	001	imm[4:0]	110_0011	BNQ
imm[11:5]	rs2	rs1	100	imm[4:0]	110_0011	BLT
imm[11:5]	rs2	rs1	101	imm[4:0]	110_0011	BGE
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	

Table 2.10: SB-Type Formate Instructions

2.5.5.1 Few RV32 SB-formate Instructions

2.5.6 UJ-Type

- UJ-Type stand for jump and link.
- This instruction is used to perform a subroutine.
- Since the PC value will be used as the base register so the immediate value must be used in the form of 2's complement.
- The purpose of 'rd' is to store the current PC value into 'rd' so that the PC can get that value again at the end of the subroutine operation.

31 12	11 7	6 0
immediate	rd	opcode
20 bits	5 bits	7 bits

Table 2.11: UJ-Type Formate

- Instruction Example:
  - jal x10, imm \\\\ $rd = PC + 4$ and $PC = PC + 4 + IMM$
    where $IMM = \{imm[19:0], 2'b00\}$

Chapter 3

Design

3.1 Single Cycle Design

3.1.1 Block diagram for single cycle design

- In this following section Block diagram will be covered of the RISC-V Processor.
- First, Starting with R-Type instruction then one by one class of ISA will be covered in the block diagram.

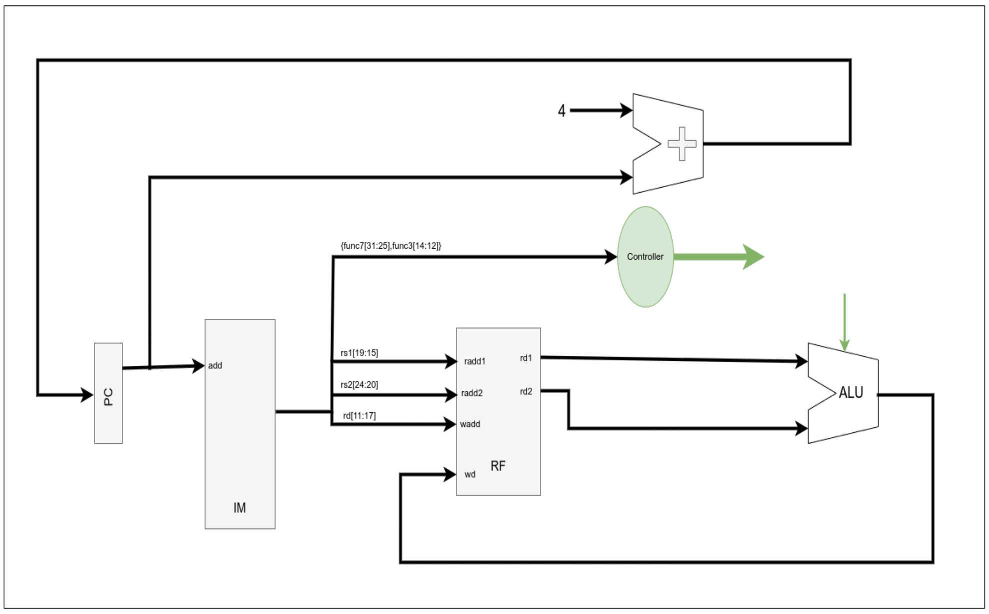


Figure 3.1: Block Diagram for R-Type instruction

- Now, the I-Type instruction would be inserted.

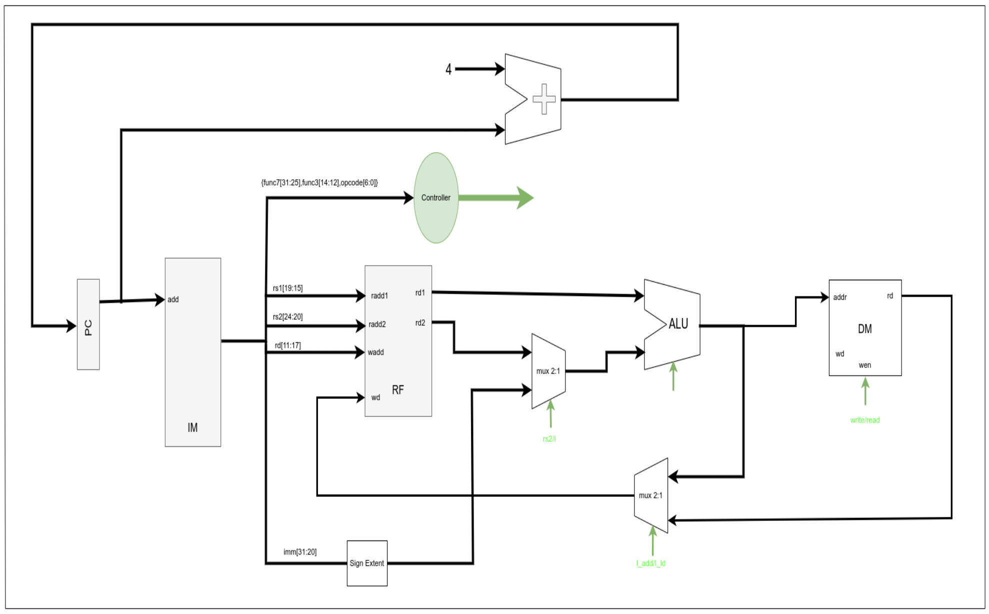


Figure 3.2: Block Diagram for R-Type & I-Type instruction

- Now, the U-Type instruction would be inserted.

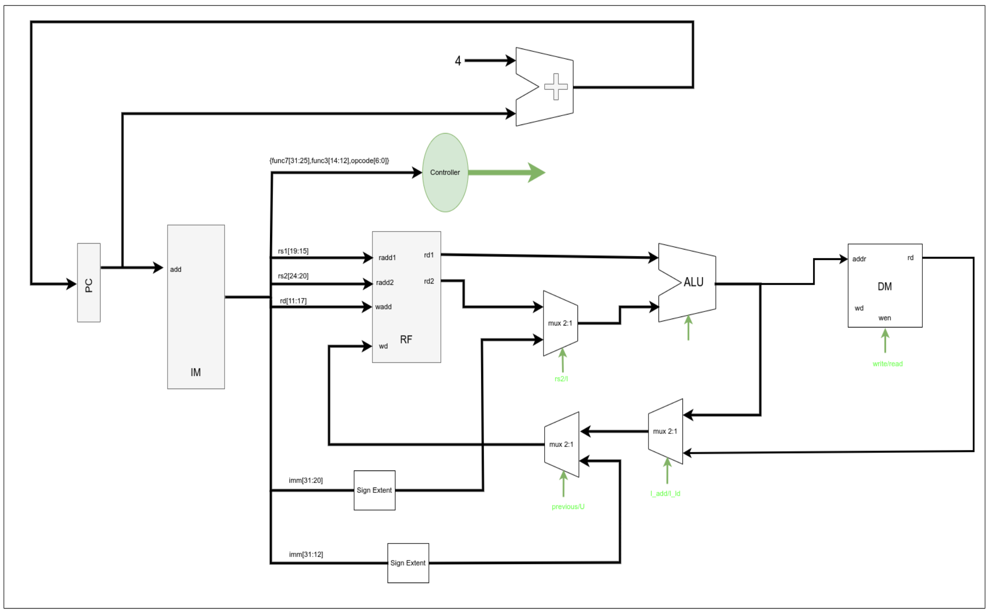


Figure 3.3: Block Diagram for R, I & U-Type instruction

- Now, the S-Type instruction would be inserted.

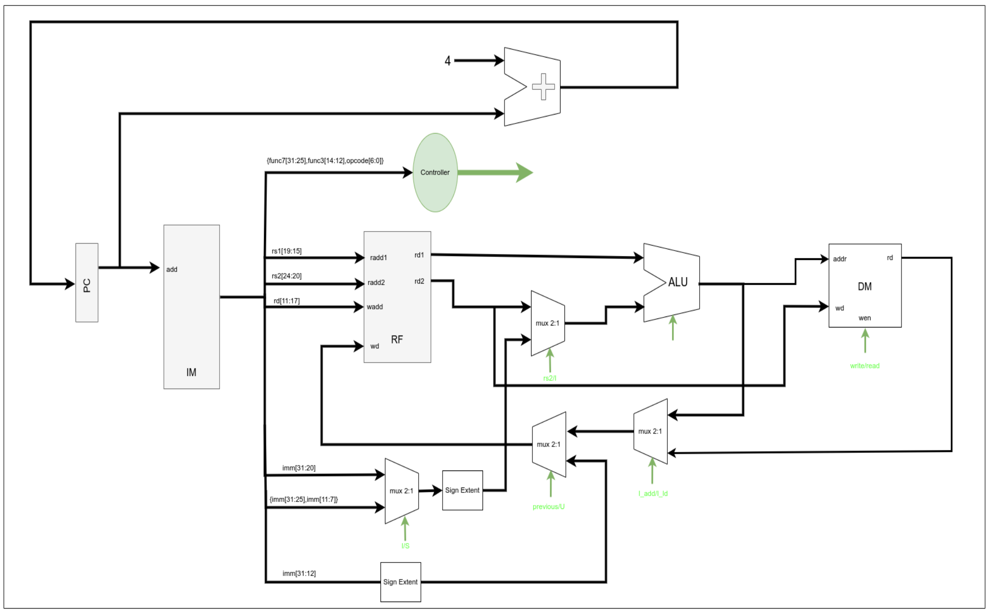


Figure 3.4: Block Diagram for R, I, U & S-Type instruction

- Now, the SB-Type instruction would be inserted.

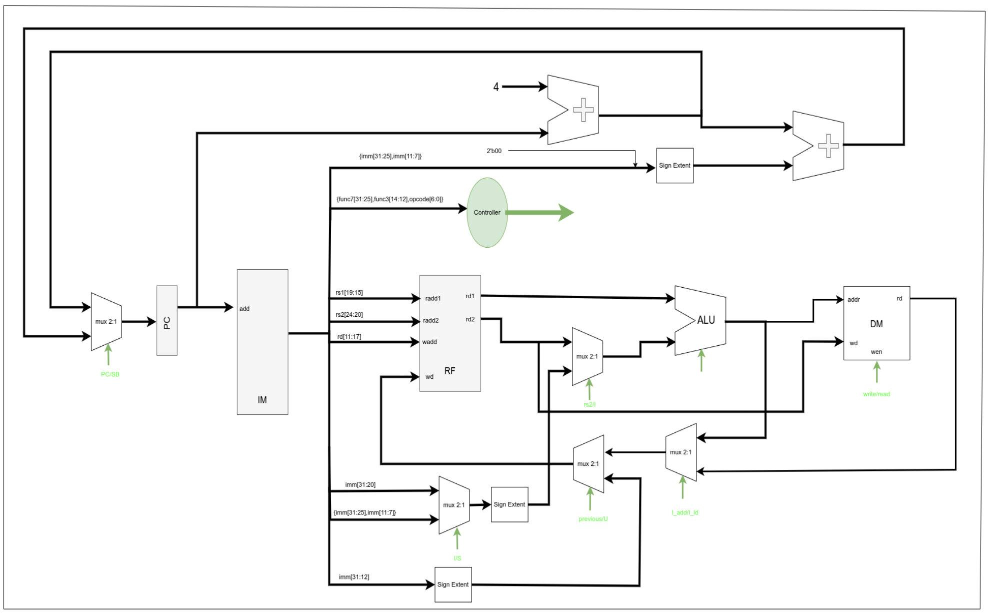


Figure 3.5: Block Diagram for R, I, U, S & SB-Type instruction

- Now, the UJ-Type instruction would be inserted.

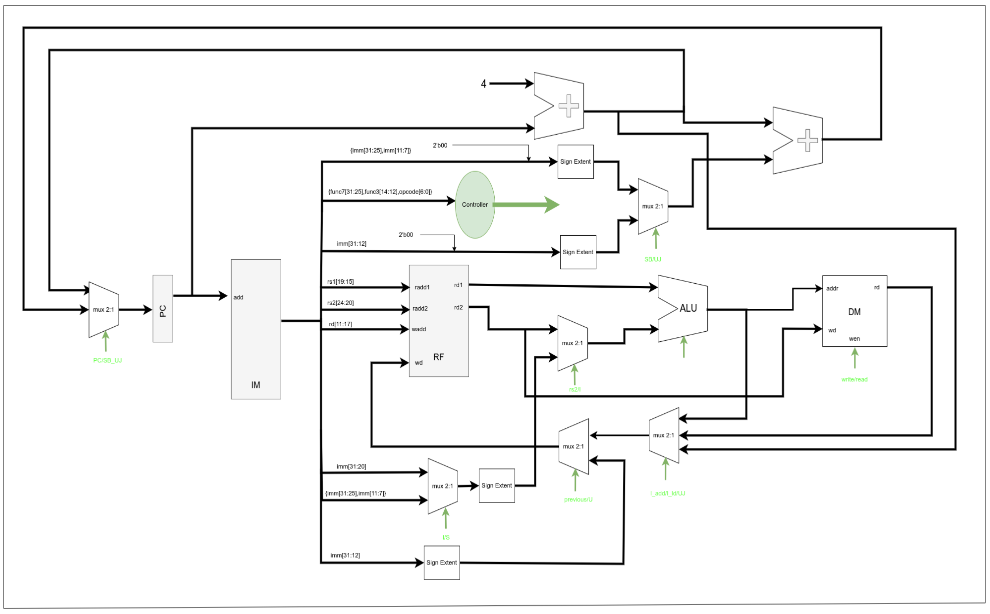


Figure 3.6: Block Diagram for R, I, U, S, SB, & UJ-Type instruction

- Modified block diagram

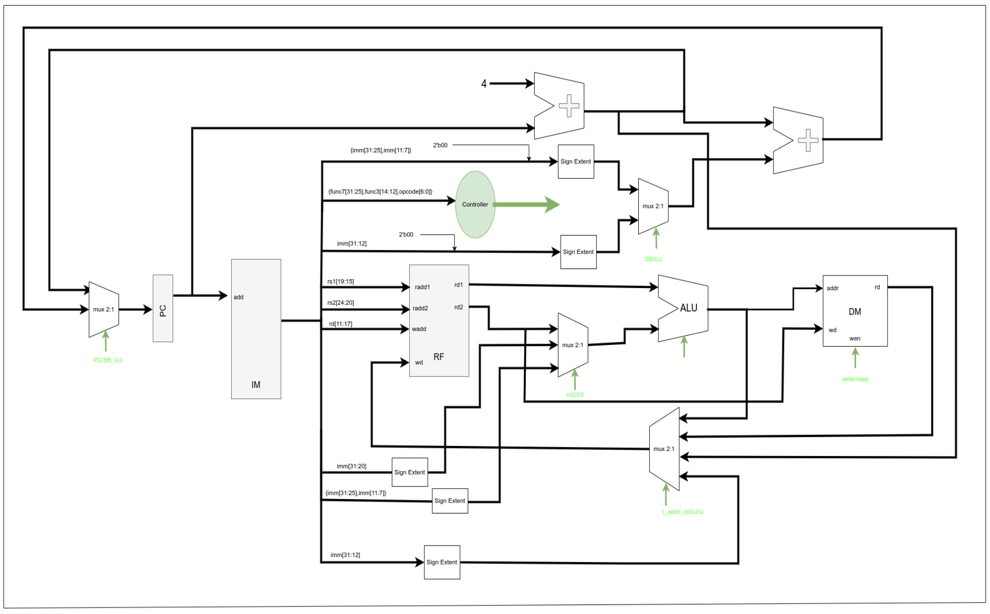


Figure 3.7: Modified

3.1.2 Problem with single cycle design

- Slowest instruction pulls down the clock frequency.

- Poor utilization of resources
- There is some instruction that is impossible to be implemented in this manner.

3.1.3 Delay analysis for Single Cycle Design

- Delay Assumption
  - Register : 0
  - Instruction memory: t_I
  - Register file : t_R
  - Data memory: t_M
  - ALU : t_A
  - ADDER : t_a
  - Bit manipulation components: 0
  - Multiplexer: 0

3.1.3.1 Delay to due to R-Type Instruction

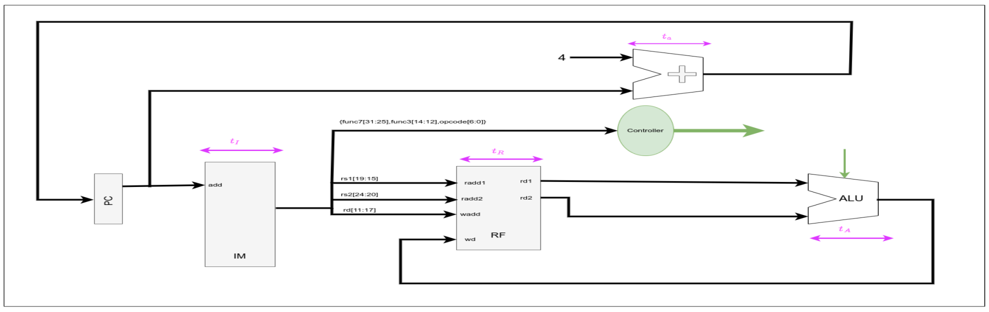


Figure 3.8: Block Diagram for R-Tpye ISA including delay

- The delay would be in this case the **maximum** among all the path

$$\begin{cases} t_a \\ t_I + t_R + t_A + t_R \end{cases}$$

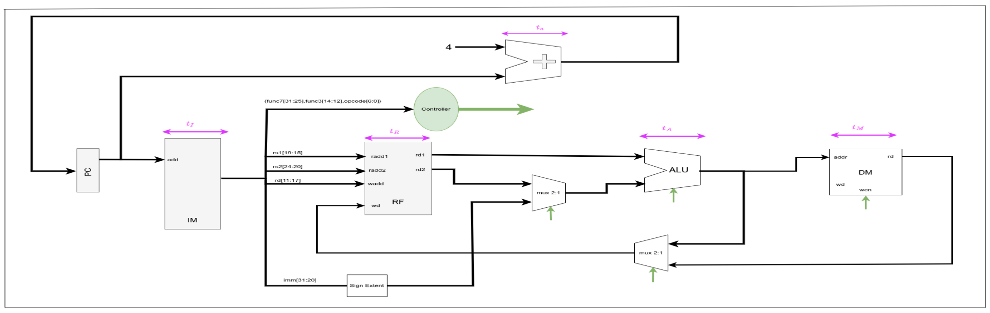


Figure 3.9: Block Diagram for I-Tpye ISA including delay

3.1.3.2 Delay to due to I-Type Instruction

- The delay would be in this case the **maximum** among all the path-

$$\begin{cases} t_a \\ t_I + t_R + t_A + t_M + t_R \end{cases}$$

3.1.3.3 Delay to due to U-Type Instruction

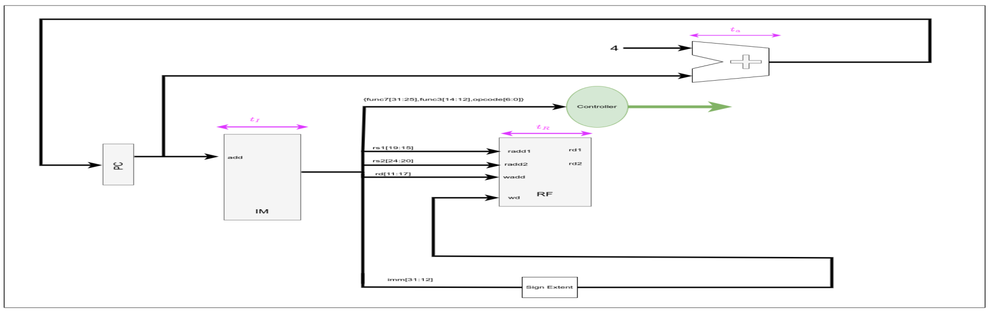


Figure 3.10: Block Diagram for U-Tpye ISA including delay

- The delay would be in this case the **maximum** among all the path-

$$\begin{cases} t_a \\ t_I + t_R \end{cases}$$

3.1.3.4 Delay to due to S-Type Instruction

- The delay would be in this case the **maximum** among all the path-

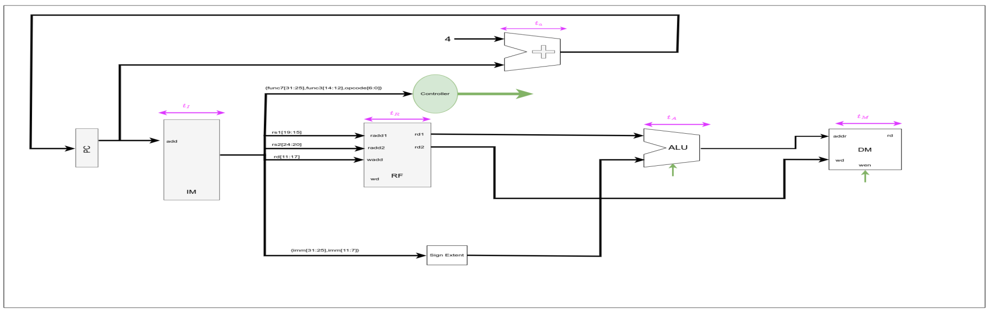


Figure 3.11: Block Diagram for S-Tpye ISA including delay

$$\begin{cases} t_a \\ t_I + t_R + t_A + t_M \end{cases}$$

3.1.3.5 Delay to due to SB-Type Instruction

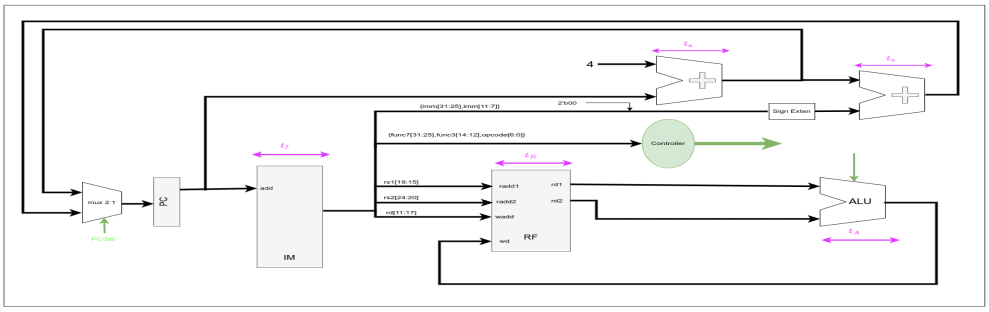


Figure 3.12: Block Diagram for SB-Tpye ISA including delay

- The delay would be in this case the **maximum** among all the path-

$$\begin{cases} t_a \\ t_I + t_R + t_A \end{cases}$$

3.1.3.6 Delay to due to UJ-Type Instruction

- The delay would be in this case the **maximum** among all the path-

$$\begin{cases} t_a + t_a \\ t_I + t_R \end{cases}$$

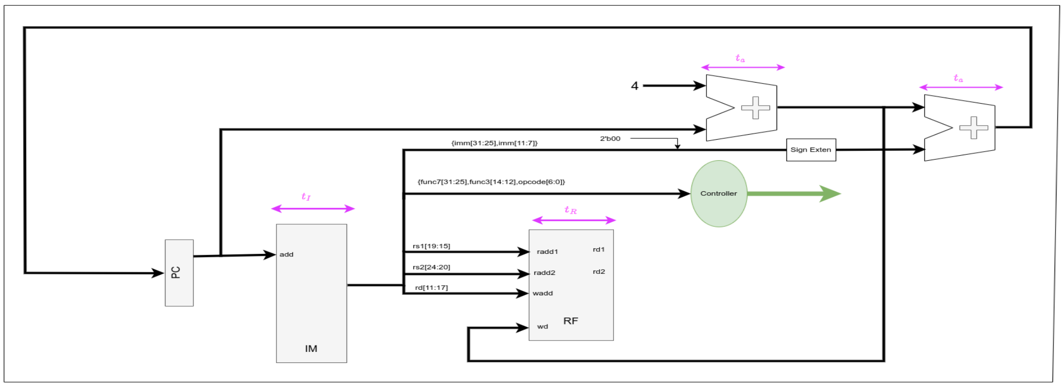


Figure 3.13: Block Diagram for UJ-Tpye ISA including delay

3.1.4 Overall clock period

- The clock period would be the maximum among all the expression got for individual instruction.

$$\begin{cases} t_I + t_R + t_A + t_M + t_R \\ t_a + t_a \\ t_I + t_a \end{cases}$$

3.1.5 Clock period in single cycle design

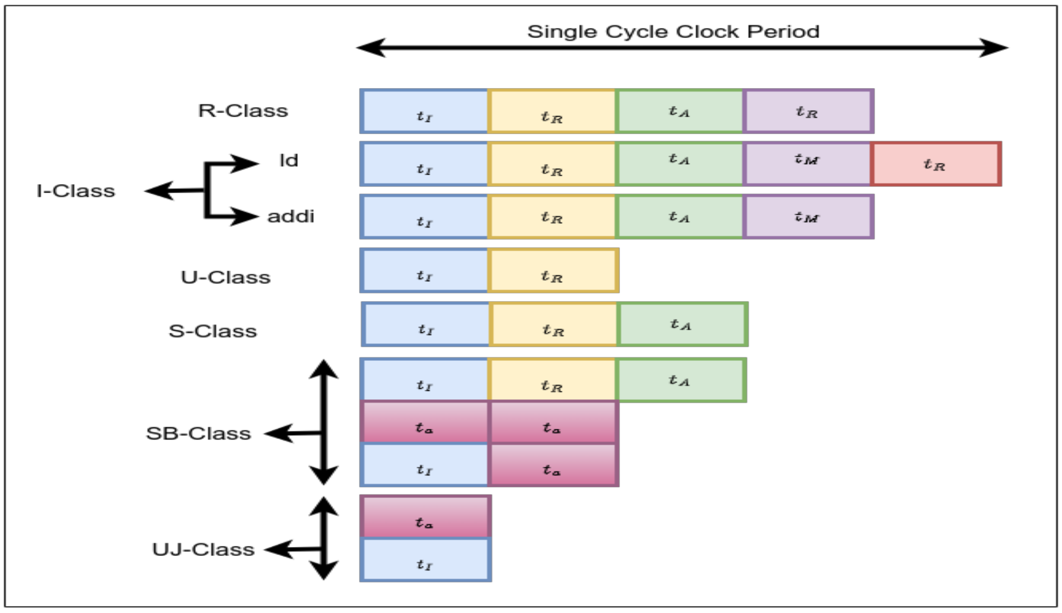


Figure 3.14: Block Diagram for analysis clock period

3.1.6 Clock period would be in multi-cycle design

- In a single cycle CPU, multiple actions are being performed in the single cycle but in the case of multi-cycle, it will take multiple cycles for multiple actions if required it can take multiple cycles for the same action to reduce the hardware.

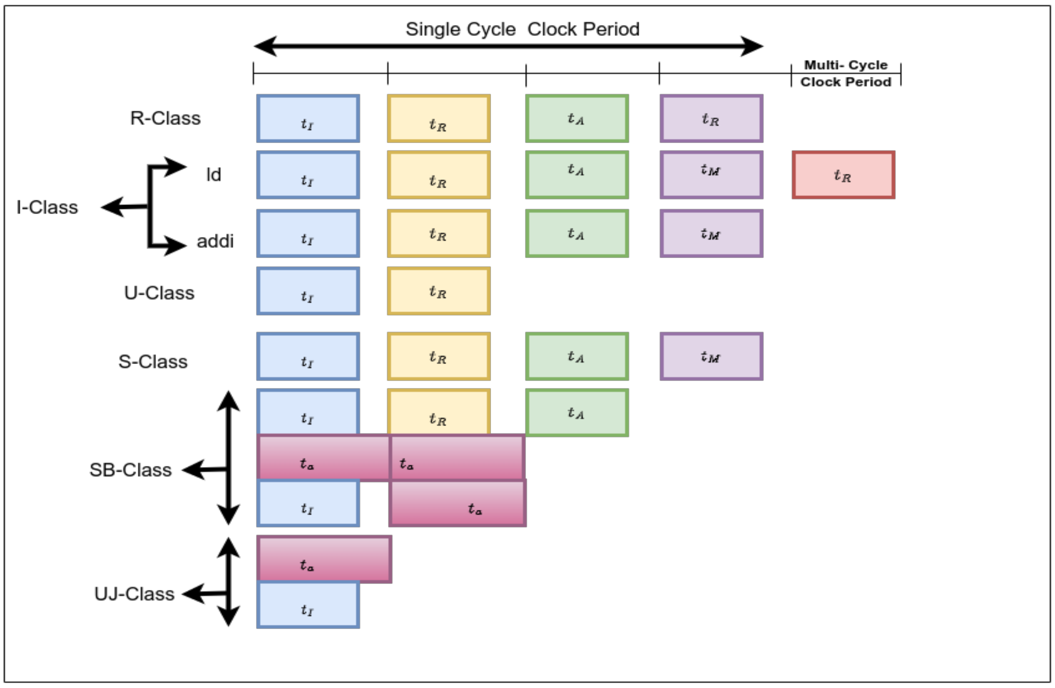


Figure 3.15: Block Diagram for analysis clock period for Multi-Cycle

3.1.7 How to achieve multi-cycle CPU design from Single cycle CPU

- By improving resources utilization
  - Elimination of two adders
  - Replace Instruction memory and Data Memory with a single memory.

3.2 Multi-cycle Processor

- The modified Block diagram from a single cycle after optimizing resources.

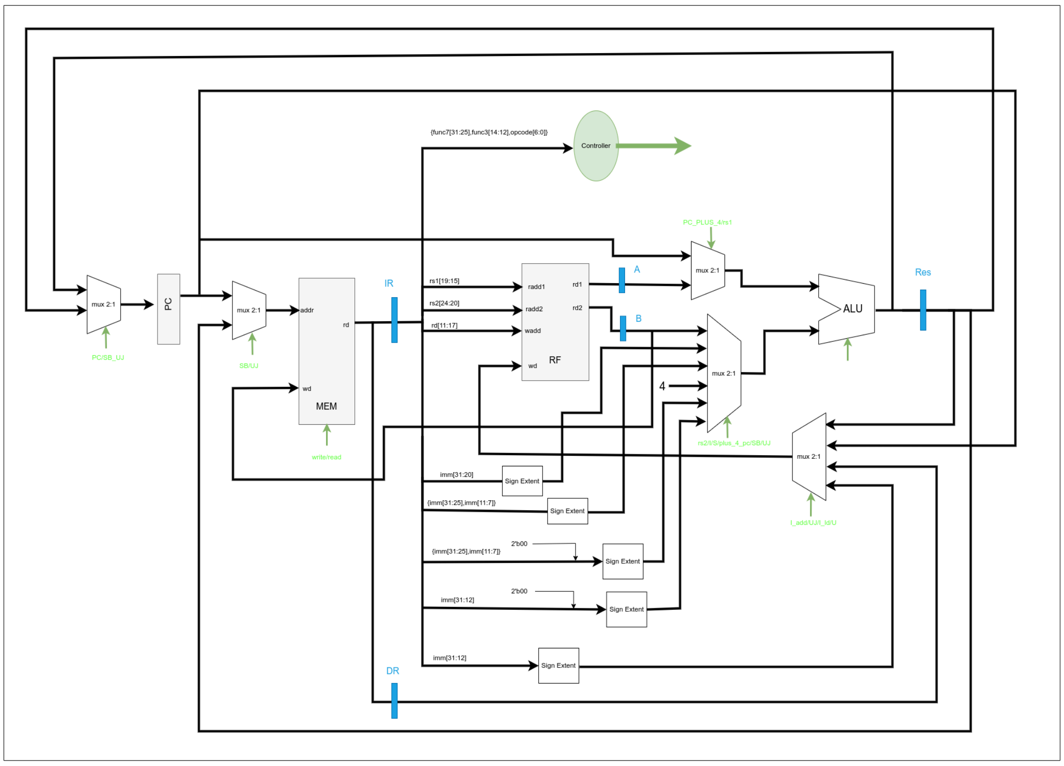
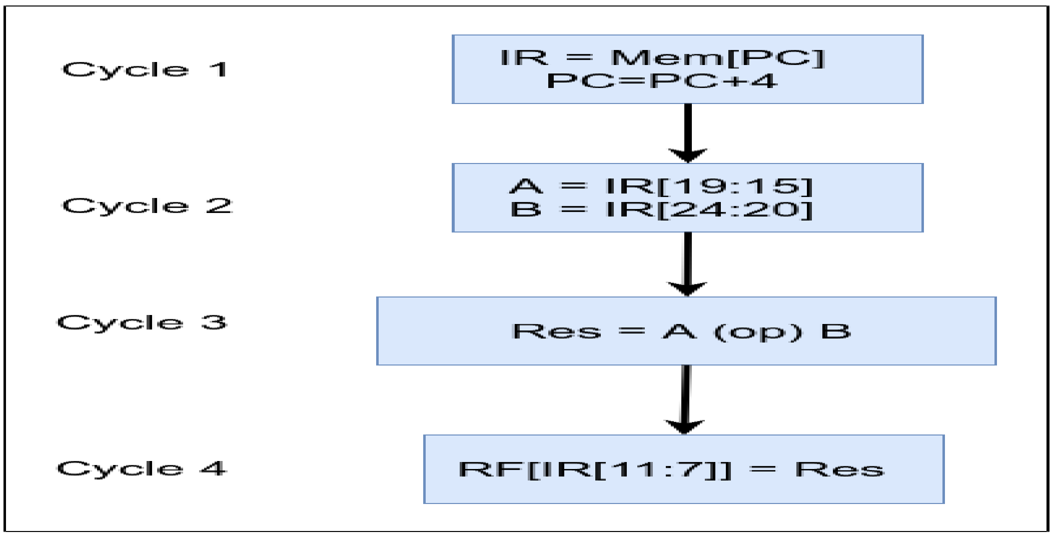


Figure 3.16: Block Diagram for Multi-Cycle Processor

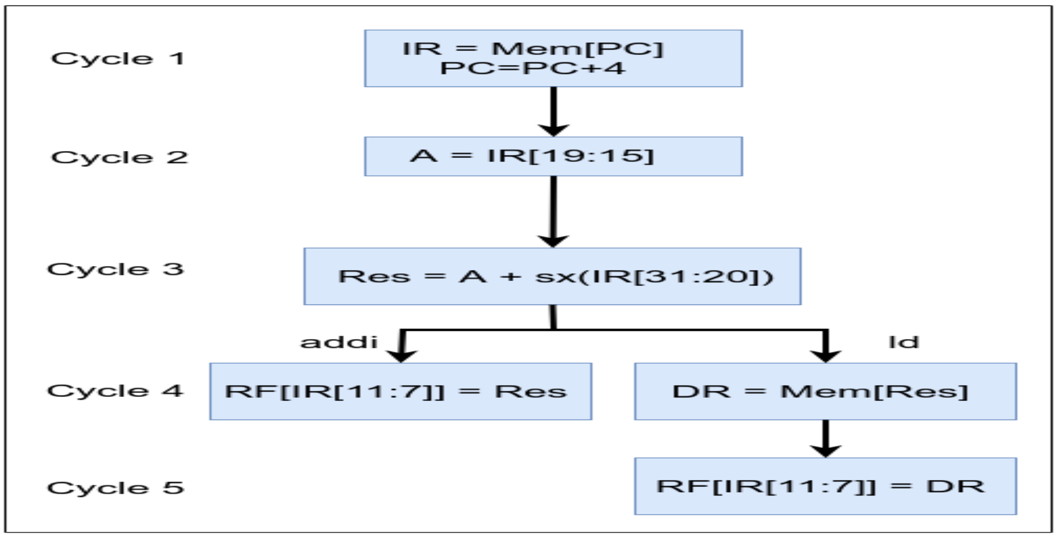
3.2.1 Control Design

3.2.1.1 Break Instructions Execution into Cycles

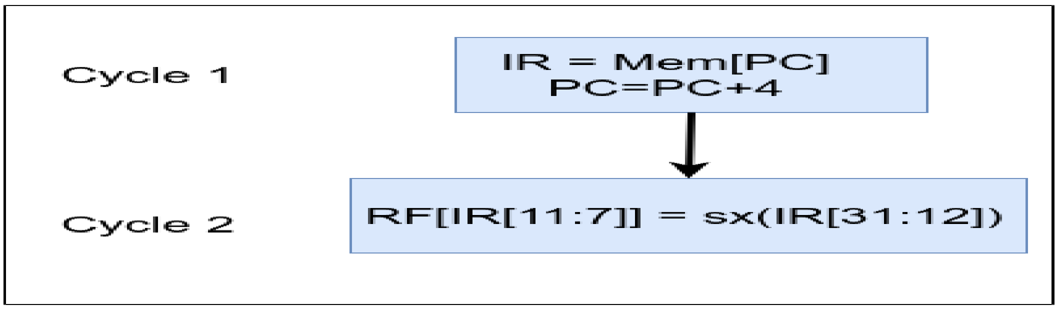
- R-Class Instruction Flow



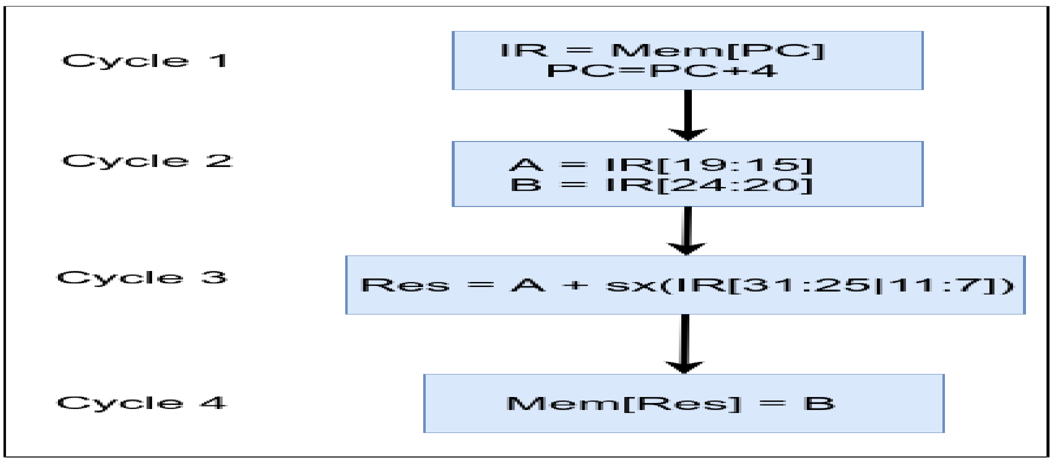
- I-Class Instruction Flow



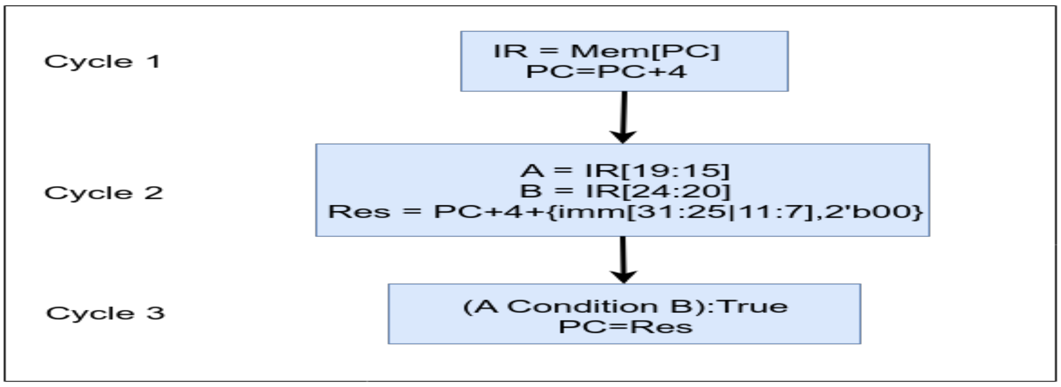
- U-Class Instruction Flow



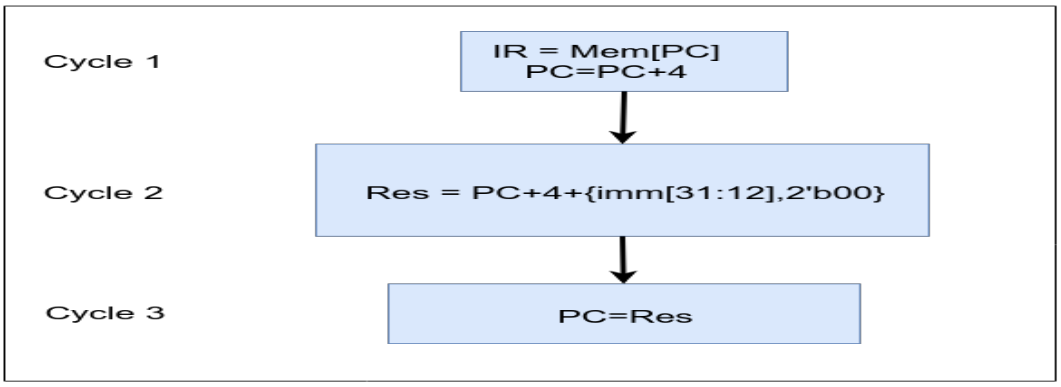
- S-Class Instruction Flow



- SB-Class Instruction Flow



- UJ-Class Instruction Flow



3.2.1.2 Combine Sequence of all Class

- After joining

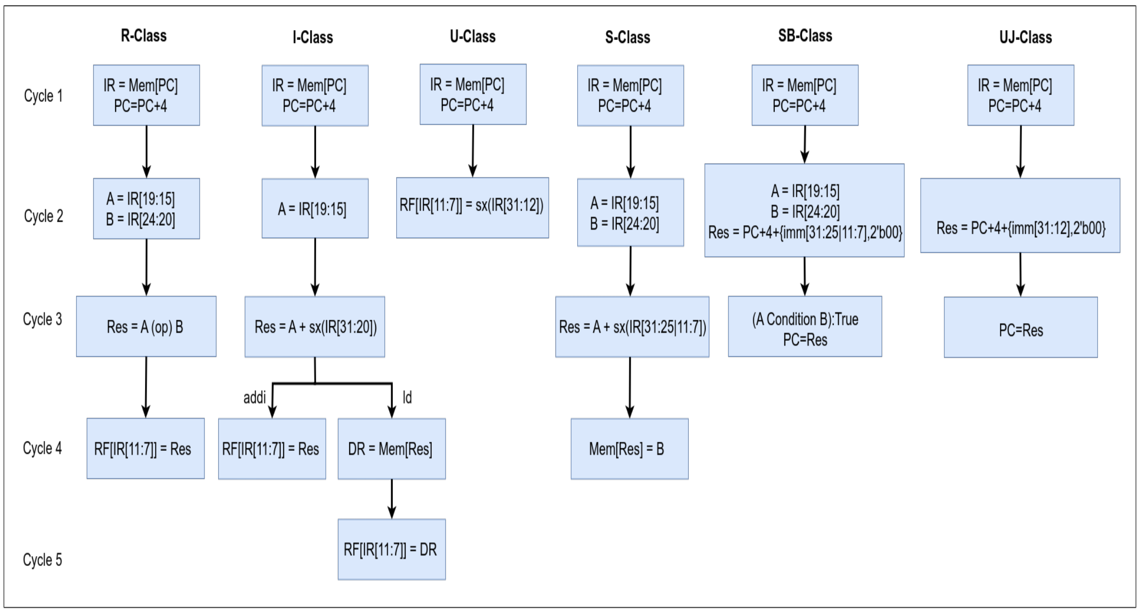


Figure 3.17: Combine Sequence

- As it can be seen in the figure 3.17 that all classes have the same operation in the first cycle which is fetched, so they can be merged to get more optimization.

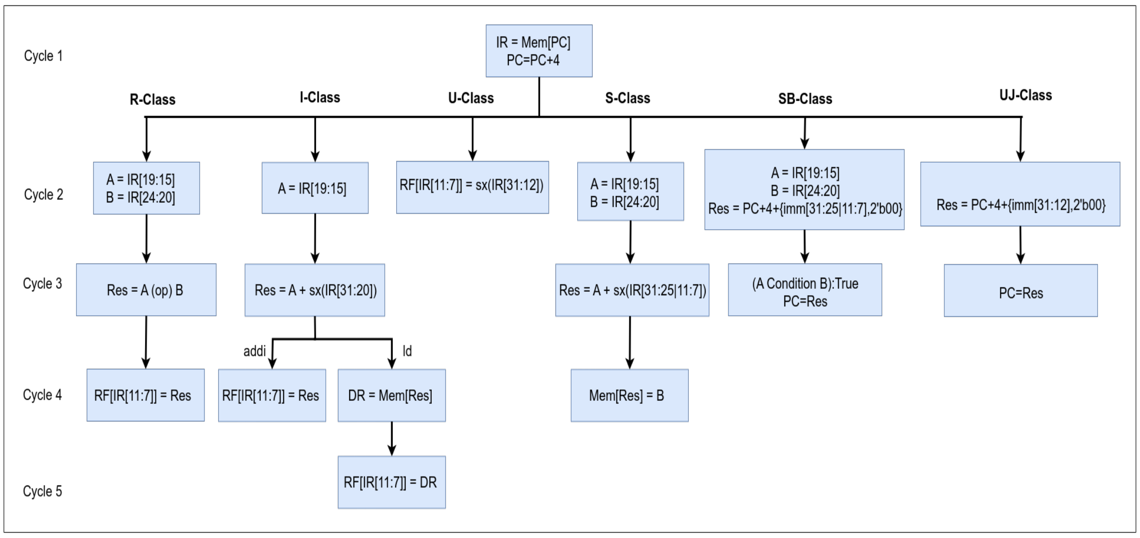


Figure 3.18: With a common Fetch cycle

- Mainly opcode starting performing operation after 2nd cycle so some optimization can be done in 2nd cycle as well.
- In the fig 3.19 it can see that most of the classes have the same operation and those which have a different action, By writing a pseudo action(which is not affecting real action) it can be made symmetric.
- Example: By including A=IR[19:15], B=IR[24:20] in the UJ-Class in the 2nd cycle block, it can be made symmetric.
- **By considering SB-Class 2nd operation as a center of focus**, all 2nd cycle actions of the class except U and UJ-Class, can be merged after including pseudo action with respect to their class.

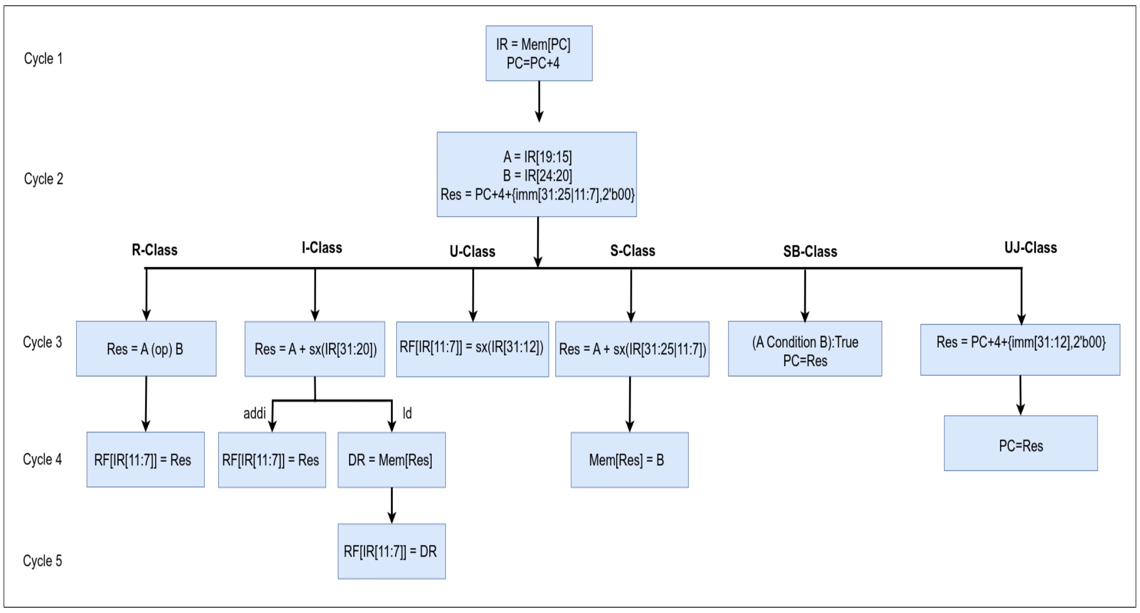


Figure 3.19: With a common Decoding cycle

Bibliography

[1] John Paul Shen and Mikko H Lipasti. *Modern processor design: fundamentals of superscalar processors*. Waveland Press, 2013.