EL501 Lab Activity

RISC V 32bit 5 Stage pipelined processor

|  |  |
| --- | --- |
| **Name** | **Student ID** |
| Parmar Rajdeep | 202011042 |
| Pandya Abhishek | 202011037 |

**Stage 1**

**Fetch**

**Fetch Unit**comprises a word (32 bits) addressable instruction memory. It takes PC as input and instructs an output. PC is also incremented by

PC=PC+1.

Instruction Memory holds different instructions to be performed by the processor.

**Stage 2**

**Decode**

**Decode Unit**reads the fetched instruction and decodes the address of two

source operands and destination register for register-based arithmetic, logical and shift instructions, immediate data(data or effective address) for immediate data-based instructions like arithmetic, logical, and branch type instructions.

**Register Bank**reads the value of source operands (Rs1 and Rs2) at the negative edge of a clock and writes the data in the destination register (Rd) at the positive edge of the clock.

The forwarding unit forwards the data from the next stages(e.g. execute, Memory) and decides whether to take operand for ALU operation from ALU result or Memory Data to eliminate hazard like RAW(Read after Write), WAR (Write after Read), WAW(Write after Write).

**Stage 3**

**Execute**

**ALU Module**uses the aluop generated by ALU Control Unit to operate on source operands.

It generates the result and stores it in the ex\_alu\_result register.

Operand Selector unit decides which data to take in the operand to perform different operations.

PSR(Program status register) shows the status according to the operations performed on the operands.

The branch unit decides whether to take the branch according to the status of PSR.

**Stage 4**

**Memory**

**Data Memory**: It stores the data of the processor.

**Memory Unit** will either write to the data memory or will read from the data memory.

**Stage 5**

**Write Back**

Write back unit will select the data and write the data to register bank.

**Instruction Format and Description**

**OPCODE**

R-type = 0110011

I-type = 0010011

L-type = 0000011

S-type = 0100011

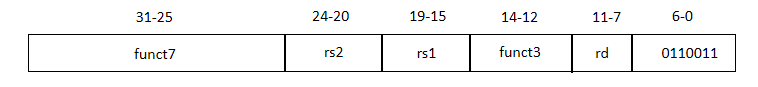
B-type = 1100011

1. **R-type Instruction:**

**Assembly(e.g. register-register operation)**

Operation rd, rs1, rs2

**Machine Encoding**



**Semantics**

GPR[rd] = GPR[rs1] operation GPR[rs2]

PC = PC + 1

**Variations**

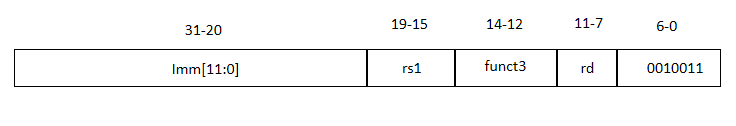
|  |  |  |  |
| --- | --- | --- | --- |
| **FUNC7** | **FUNC3** | **Instruction name** | **aluop** |
| 0000000 | 000 | add | 0000 |
| 0100000 | 000 | sub | 1000 |
| 0000000 | 001 | sll | 0001 |
| 0000000 | 010 | slt | 0010 |
| 0000000 | 011 | sltu | 0011 |
| 0000000 | 100 | xor | 0100 |
| 0000000 | 101 | srl | 0101 |
| 0100000 | 101 | sra | 1101 |
| 0000000 | 110 | or | 0110 |
| 0000000 | 111 | and | 0111 |

1. **I-type Instruction:**

**Assembly(e.g. register-Immediate operations)**

Operation rd, rs1, imm12

**Machine Encoding**



**Semantics**

GPR[rd] = GPR[rs1] operation Sign-extend(imm)

PC =PC + 1

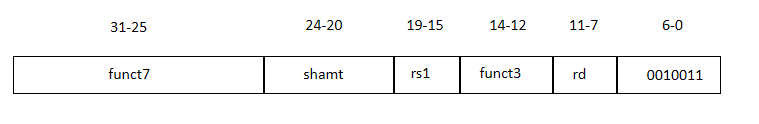
**Variations**

|  |  |  |
| --- | --- | --- |
| **FUNC3** | **Instruction name** | **aluop** |
| 000 | addi | x000 |
| 010 | slti | x010 |
| 011 | sltiu | x011 |
| 100 | xori | x100 |
| 110 | ori | x110 |
| 111 | andi | x111 |

**Assembly(e.g. register-Immediate operations)**

Operation rd, rs1, shamt

**Machine Encoding**



**Semantics**

GPR[rd] = GPR[rs1] operation Shamt

PC =PC + 1

**Shift instruction variations:**

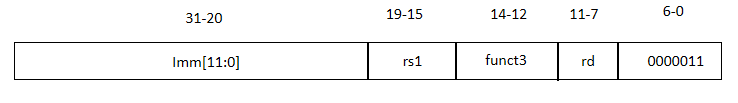
|  |  |  |  |
| --- | --- | --- | --- |
| **FUNC7** | **FUNC3** | **Instruction name** | **aluop** |
| 0000000 | 001 | slli | 0001 |
| 0000000 | 101 | srli | 0101 |
| 0100000 | 101 | srai | 1101 |

1. **L-type instruction:**

**Assembly (e.g. Load 4/2/1-byte)**

Operation rd, offset12(base)

**Machine Encoding**



**Semantics**

Byte\_address32 = sign-extend(offset12) + GPR[base]

GPR[rd] = MEM32[byte\_address]

PC = PC + 1

**Variations**

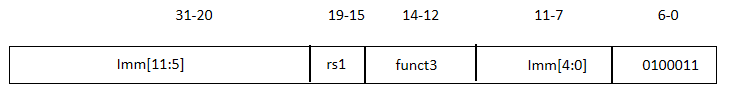
|  |  |  |
| --- | --- | --- |
| **FUNC3** | **Instruction name** | **aluop** |
| 000 | lb | x000 |
| 001 | lh | x001 |
| 010 | lw | x010 |

1. **S-type instruction:**

**Assembly(e.g. Store 4/2/1-byte)**

Operation rs2,offset12(base)

**Machine Encoding**



**Semantics**

byte\_address32 = sign-extend(offset12) + GPR[base]

MEM32[byte\_address] = GPR[rs2]

PC = PC + 1

**Variations**

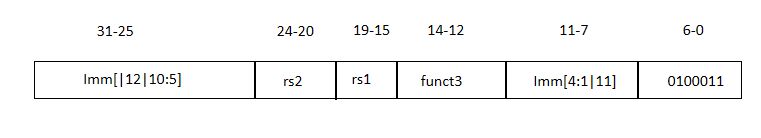
|  |  |  |
| --- | --- | --- |
| **FUNC3** | **Instruction name** | **aluop** |
| 000 | sb | x000 |
| 001 | sh | x001 |
| 010 | sw | x010 |

1. **B-Type instruction:**

**Assembly(e.g. branch if equal)**

Branch\_operation rs1, rs2, imm13

**Machine Encoding**



**Semantics**

Target = PC + sign-extend(imm13)

If GPR[rs1] operation GPR[rs2] then PC = target

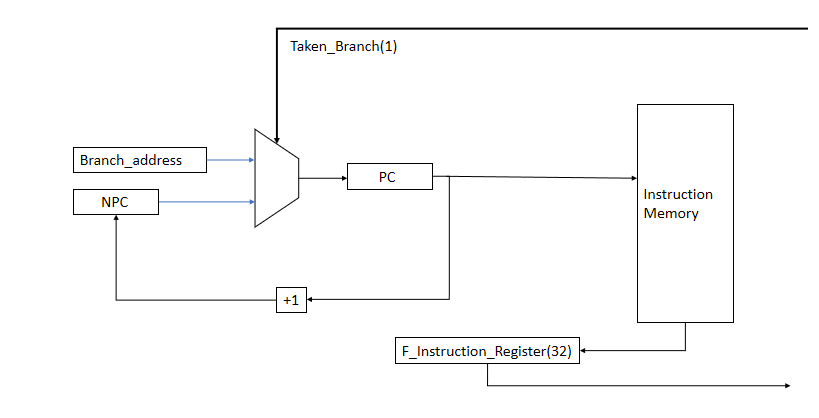
Else PC = PC + 1

**Variation:**

|  |  |  |
| --- | --- | --- |
| **FUNC3** | **Instruction name** | **aluop** |
| 000 | beq | x000 |
| 001 | bne | x001 |
| 100 | blt | x100 |
| 101 | bge | X101 |

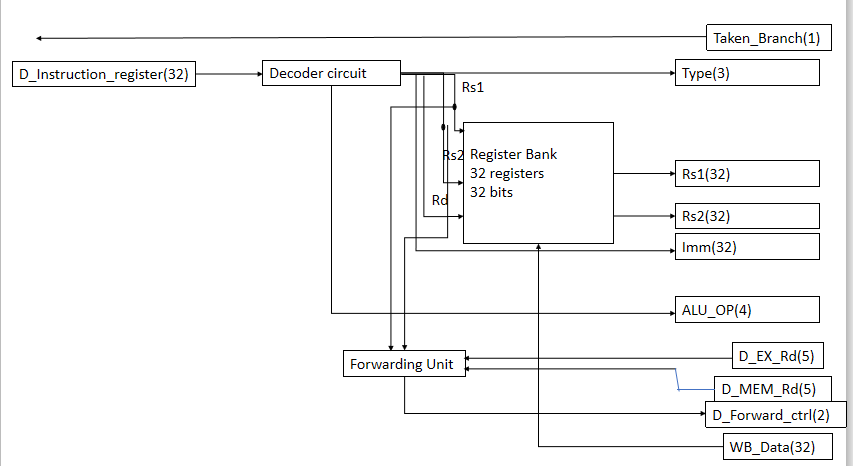
**Description**

**Fetch stage:**



This stage fetches the instruction from instruction memory and stores it in f\_instruction\_register. And after fetching the instruction, it increases the program counter by 1. In the next clock cycle, PC takes value according to the taken\_branch signal. If taken\_branch = 1, PC = branch\_address generated by the branch unit, otherwise PC = NPC.

**Decode Stage:**



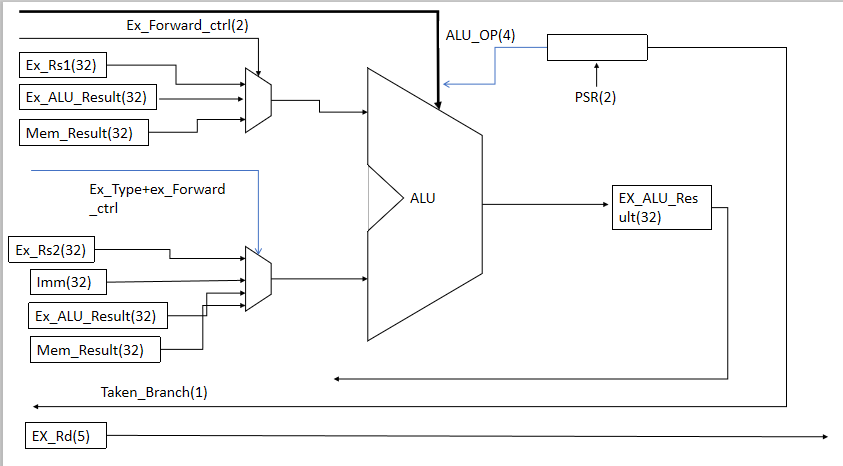
This stage decodes the instruction available in the instruction register and generates signals(i.e. alu\_op, type, etc.).

decoder circuit will give source register addresses (i.e. rs1\_address and rs2\_address), destination register address (i.e. rd\_address), and immediate data.

The forwarding unit will compare different destination register addresses from the next stages, with source operands address, and based on this it will generate d\_forward\_ctrl, which will be used in the operand selector.

alu\_op will decide the operation to be performed in ALU and the type signal will define the type of instruction.

**Execute stage:**



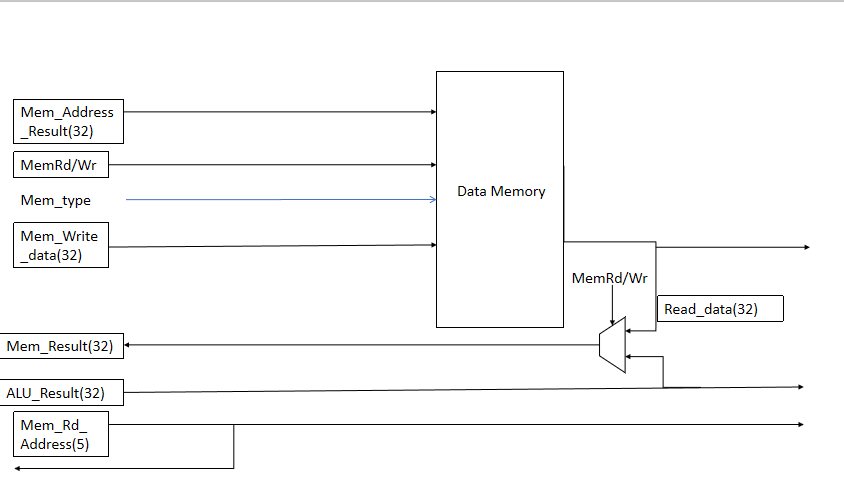
This stage executes different instructions according to the alu\_op signal in the ALU unit. Operand1 to the ALU will be selected according to the forward\_ctrl signal and Operand2 will be selected according to the type and forward\_ctrl signals.

ex\_rsk(k = 1,2 ) defines data of rsk register, ex\_alu\_result defines data of previously executed instruction, mem\_result defines data from the memory that is received in previous instruction and imm defines immediate data from the instruction.

PSR is a program status register. It contains 2 bits, which will be used to decide the taken\_branch signal in conditional branch instructions.

|  |  |
| --- | --- |
| **PSR bits(2)** | **Condition** |
| 00 | rs1 = rs2 |
| 01 | rs1 != rs2 |
| 10 | rs1 >= rs2 |
| 11 | rs1 < rs2 |

**Memory stage :**



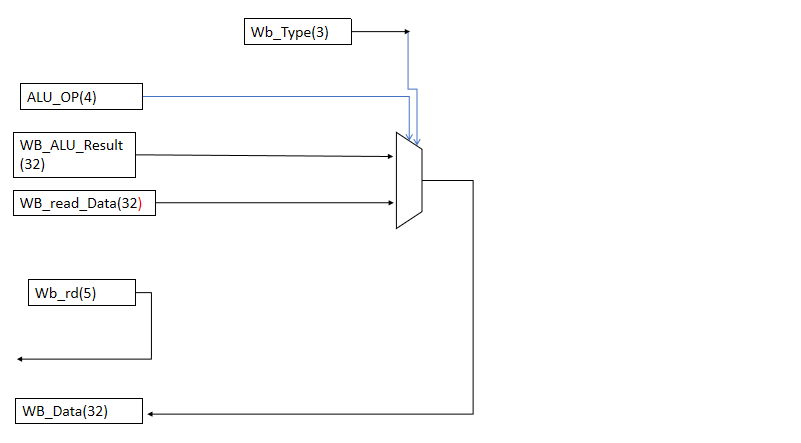
This stage is used to load and store data from register\_bank to data\_memory and vice-versa.

mem\_rd\_wr signal defines load or store operation(i.e. mem\_rd\_wr = 0 , load and mem\_rd\_wr = 1, store).

mem\_address\_result will contain either result or address. mem\_type defines the type of instruction. mem\_write\_data register defines the data that must be written in data\_memory in a store instruction.

mem\_result and mem\_rd\_adderess are used for data forwarding.

**Write-Back stage**



This stage is used for writing the data in the register bank. alu\_op signal is used for defining the number of Bytes to be moved in the register bank. wb\_alu\_result is the data to be written in the register bank which is the result from the ALU and wb\_read\_data is the data to be written in the register bank which is from the memory stage (i.e. load).