CA LAB1 Report

1 Module Explanation

- MUX32, MUX32_4:
  Both of them are multiplexers. MUX32 takes 1-bit selector and can have up to 2 input data, while MUX32_4 takes 2-bit selector and can have up to 4 input data.

- Imm_Gen:
  Sign_Extend module reads the 32-bit instruction as input and outputs 32-bit immidate data following the below machine code.

funct7	rs2	rs1	funct3	rd	opcode	function
0000000	rs2	rs1	111	rd	0110011	and
0000000	rs2	rs1	100	rd	0110011	xor
0000000	rs2	rs1	001	rd	0110011	sll
0000000	rs2	rs1	000	rd	0110011	add
0100000	rs2	rs1	000	rd	0110011	sub
0000001	rs2	rs1	000	rd	0110011	mul
imm[11:0]		rs1	000	rd	0010011	addi
0100000	imm[4:0]	rs1	101	rd	0010011	srai
imm[11:0]		rs1	010	rd	0000011	lw
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	sw
imm[12,10:5]	rs2	rs1	000	imm[4:1,11]	1100011	beq

- Control:
  Control module takes the 7-bit opcode and NoOp as input, and outputs corresponding control signals, e.g. RegWrite, ALUOp. When NoOp is set, the control signals will be set to 0.

- ALU_Control:
  ALU_Control module takes funct7|funct3 and 2-bit ALUOp as input, and outputs 3-bit ALUCtrl. ***lw*** and ***sw*** should perform ***add*** operation in ALU.

- ALU:
  ALU module takes data1, data2, ALUCtrl as input and outputs data, Zero. It identifies the operator with ALUCtrl and outputs the arithmetic result to data. Zero is not used in this assignment, so I don't perform any operation on Zero.

- Adder:
  Adder module takes data1, data2 as input and outputs the result of data1 + data2. It is used to update PC in this assignment.

- IF_ID, ID_EX, EX_MEM, MEM_WB:
  Those are pipeline latches. They will separate the CPU into different stages. They will update the data and signals to the next stages at the positive edge of the clock signal. IF_ID will takes additional stall and flush input sinal to determine whether the instruction should be stalled of flushed.

- Forward_Unit:
  The module is used to detect data dependency and forward the data if needed. The implementation follows the below guide in the slides.

Forwarding Control

1. EX hazard:

```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd == ID/EX.RegisterRs1)) ForwardA = 10

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 10
```

2. MEM hazard:

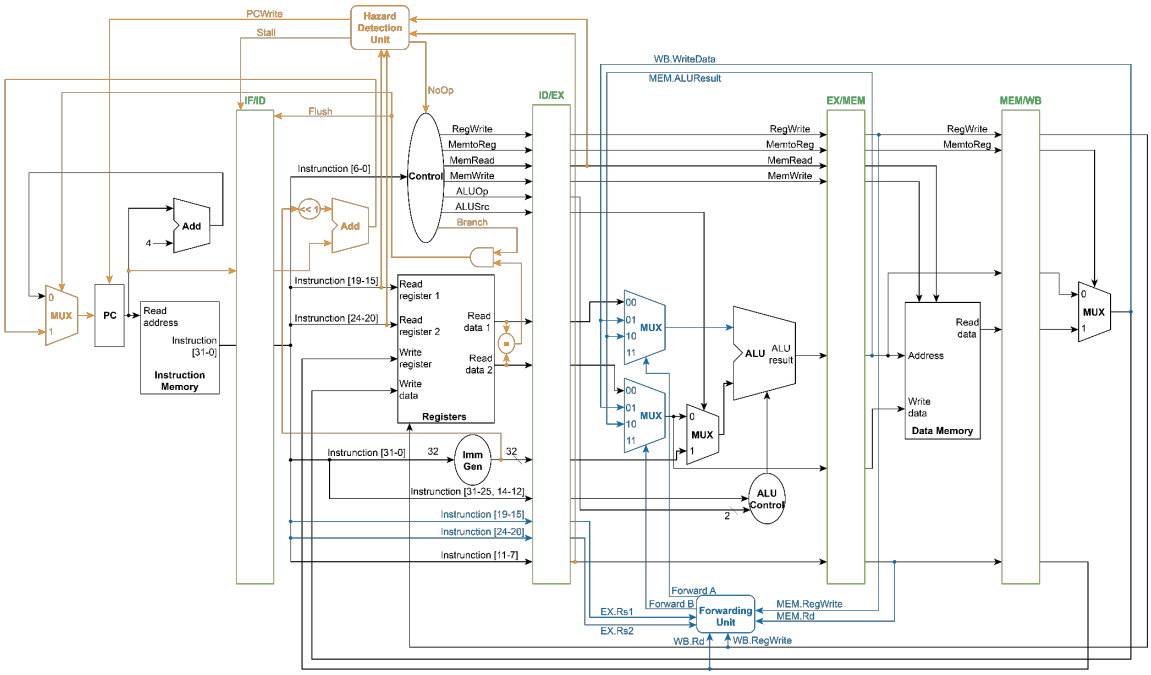
```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd != 0)
and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0)
        and (EX/MEM.RegisterRd = ID/EX.RegisterRs1))
and (MEM/WB.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 01

if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd != 0)
and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0)
        and (EX/MEM.RegisterRd = ID/EX.RegisterRs2))
and (MEM/WB.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 01
```

- Hazard_Unit:
  It detects whether the instruction should be stalled or not. It will stall the instruction if the register is used right after the register data is loaded from memory.

- CPU:
  CPU module is the core module in this assignment. It integrates different components and specifies data flows to be the below diagram. CPU module reads clock, reset and start signals. Those signals will be forwarded to some modules of the CPU, e.g. latches, PC.

CA LAB1 Report



2 Difficulties Encountered and Solutions in This Lab

- Complex datapath
  Compared to HW3, the datapath is more complex. This is the most obvious challenge in this assignment. The difficulty would be less challenging with proper variable naming, e.g. specify input as _i and output as _o, use the wire name shown in the above figure.

- x is written to registers at some points
  I found that some data of the registers would become x after executing for a while. I used gtkwave and found that some control signals and instructions were not initialized properly. After proper initialization, the instruction can be be executed correctly.

3 Development Environment

- OS: Ubuntu20.04
- Compiler: iverilog