BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Course No: CSE306
Course Name: Computer Architecture Sessional

Name of the Experiment:
Assignment on 8-bit MIPS Pipelined Execution

Level/Term: 3-1
Section: B2

Group No: 02
Student ID: 1705092
1705111
1705116
1705118
1705121

Date of Submission: 04-07-2021

Introduction

In this assignment, we have designed an 8-bit processor that supports pipelining for a subset of MIPS instruction set.

Each instruction is divided into five stages: instruction fetch (IF), instruction decode (ID), execution and address calculation (EX), data memory access (MEM), and write back (WB).

The length of the clock cycle is the maximum time to execute any single stage. Therefore, each instruction takes up to five clock cycles to be executed.

The main components of the processor are as follows: instruction memory, register file, 8-bit ALU, control unit, four pipeline registers, a forwarding unit and data memory.

Instruction Set

For this experiment, we have used the same instruction set order and control codes that we used to implement the single clock cycle 8-bit MIPS. The relevant information of instruction set for this experiment is stated below,

ALUOp	Operation
000	ADD
001	SUB
010	AND
011	OR

The 16-bit control sequence,

-			Reg Write	-		-	ALUOp			-	-	-	-	-	-

MSB

The Instruction Set,

Instruction ID	**Instruction**	**Type**	**Opcode**	**Control Sequence**
A	add	R	0x0	0b 0001 0000 0000 0001 = 0x1001
E	and	R	0x2	0b 0001 0000 1000 0001 = 0x1081
C	sub	R	0x9	0b 0001 0000 0100 0001 = 0x1041
G	or	R	0xC	0b 0001 0000 1100 0001 = 0x10C1

Every instruction is 20-bit long R-type instructions.

- R-type

Opcode	Src Reg 1	Src Reg 2	Dst Reg	Shft Amnt
4-bits	4-bits	4-bits	4-bits	4-bits

The 20-bit instructions are generated from the assembly language with an assembler written in C++ language in the file "***assembler.cpp***".

Complete Block diagram of Pipelined Datapath of 8-bit MIPS processor

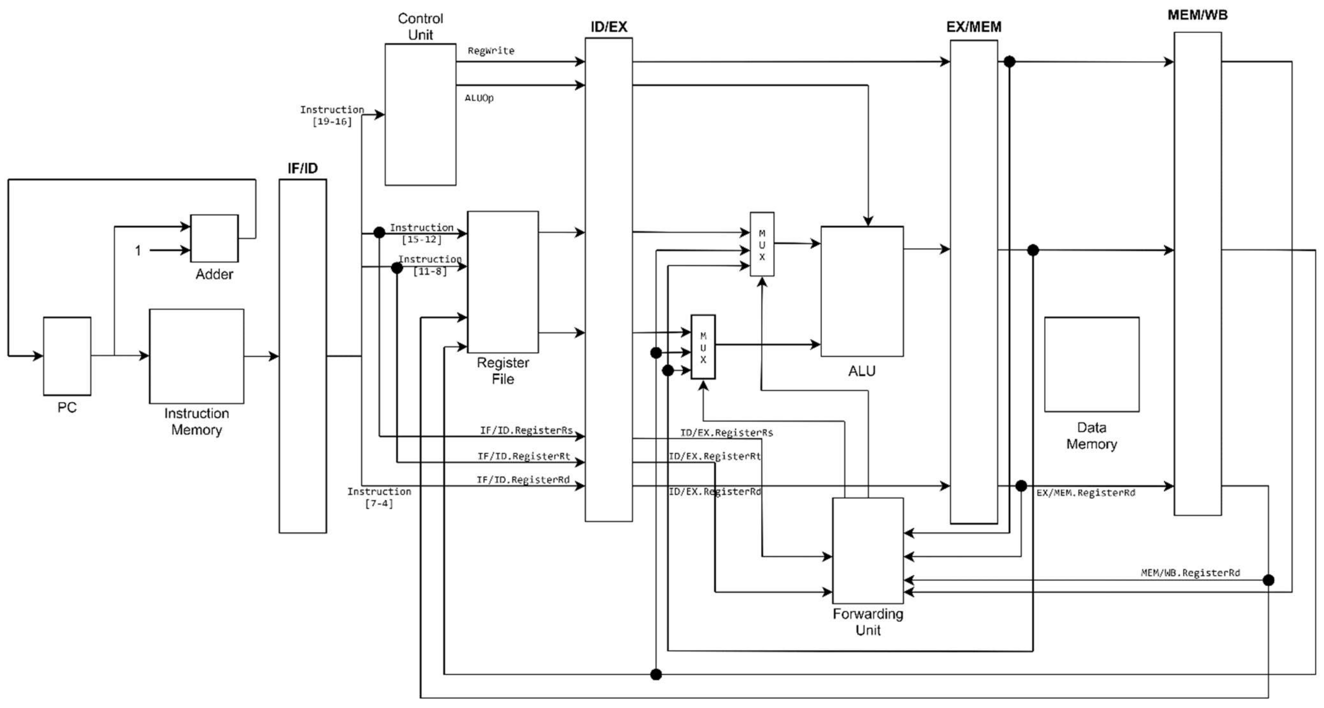

Figure: Complete Block diagram of Pipelined Datapath of 8-bit MIPS processor

Block diagram and size of Pipeline Registers

- **<u>IF/ID Register:</u>** 20 bits
  It holds the 20-bit instruction read from the instruction memory.

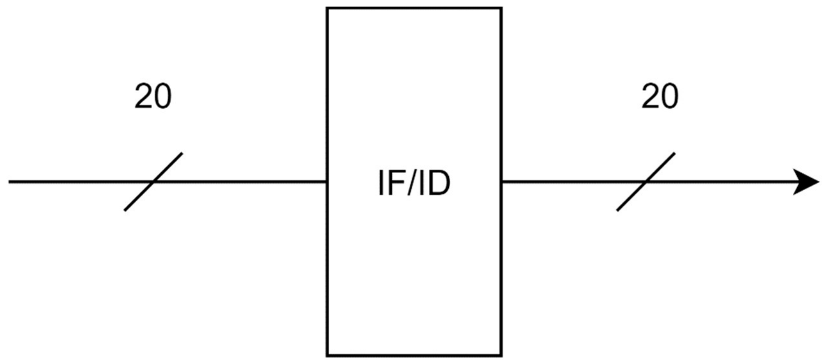

Figure: IF/ID Register

- **<u>ID/EX Register:</u>** 32 bits

RegisterRd	RegisterRt	RegisterRs	Read data 2	Read data 1	ALUOp	RegWrite

MSB

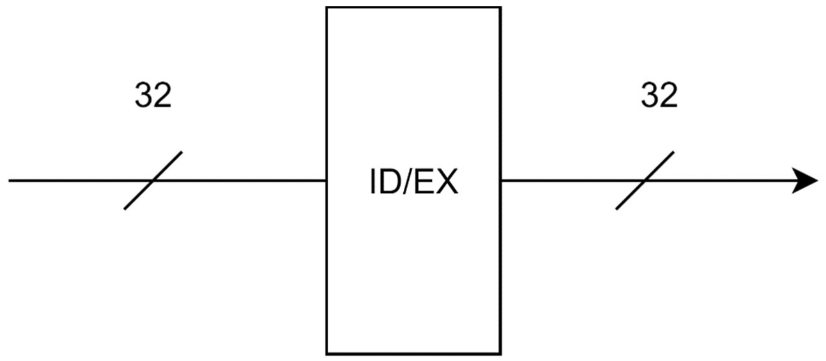

Figure: ID/EX Register

- **EX/MEM Register:** 13 bits

RegisterRd				ALU Output								Reg Write

MSB

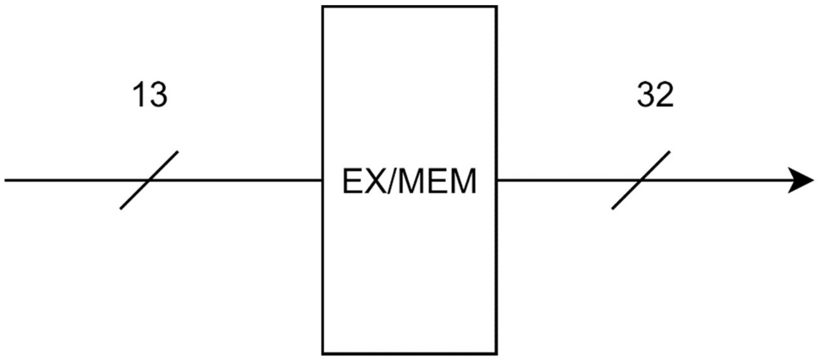

Figure: EX/MEM Register

- **MEM/WB Register:** 13 bits

RegisterRd				ALU Output								Reg Write

MSB

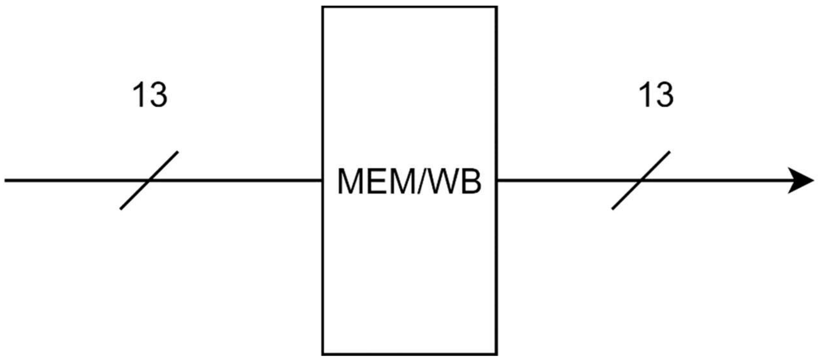

Figure: MEM/WB Register

Block diagram of Forwarding Unit

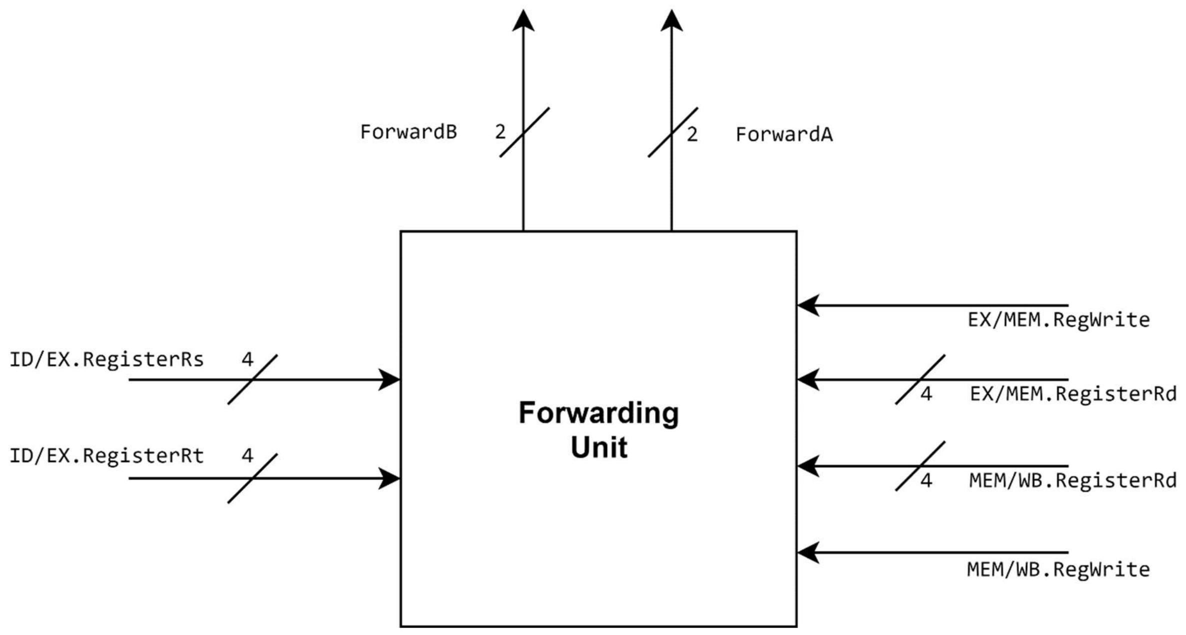

Figure: Block diagram of Forwarding Unit

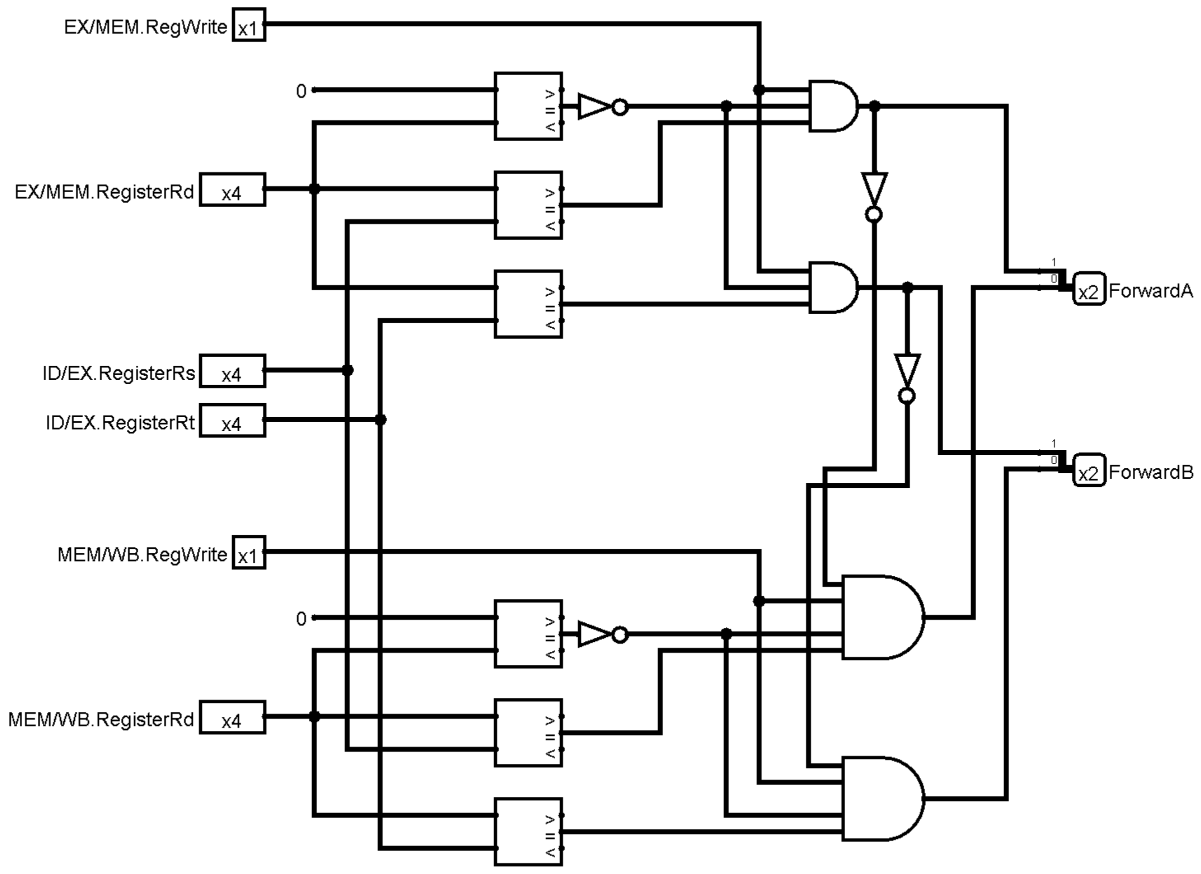

Figure: Circuit diagram of Forwarding Unit

Mechanism of Forwarding Unit

The Forwarding Unit is used for forwarding aka bypassing to resolve Data Hazard in the pipelined MIPS processor.

EX Hazard handling:
EX Hazard occurs when the dependent instruction is in the EX-stage and the prior instruction is in MEM stage.

To handle this, the data from the EX/MEM register is passed on to the multiplexers for the inputs of the ALU unit. This data will be used if the source registers used in the current instruction (ID/EX.ResgisterRs or ID/EX.ResgisterRt) is the same as the destination register of the prior instruction (EX/MEM. ResgisterRd). And if data is going to be written in the destination register, that is if EX/MEM.RegWrite flag is true. Additionally, if the destination is the $zero register we do not need to be concerned with an updated value as $zero will always holds zero. So, in that case this hazard check can be ignored.
To summarize, we can write up a pseudocode for it,

```
if (EX/MEM.RegWrite && (EX/MEM. ResgisterRd ≠ 0)
   && (EX/MEM. ResgisterRd == ID/EX.ResgisterRs))
        ForwardA = 10;

if (EX/MEM.RegWrite && (EX/MEM. ResgisterRd ≠ 0)
   && (EX/MEM. ResgisterRd == ID/EX.ResgisterRt))
        ForwardB = 10;
```

MEM Hazard handling:
MEM Hazard occurs when the dependent instruction is in the EX-stage and the prior instruction is in WB stage.

To handle this, the data from the MEM/WB register is passed on to the multiplexers for the inputs of the ALU unit. This data will be used if the source registers used in the current instruction (ID/EX.ResgisterRs or ID/EX.ResgisterRt) is the same as the destination register of the prior instruction (MEM/WB. ResgisterRd). And if data is going to be written in the destination register, that is if MEM/WB.RegWrite flag is true. Additionally, the $zero register check happens here as well.

Double Data Hazard handling:
Double Data Hazard occurs when the dependent instruction is in EX-stage and it depends on two prior instructions, one of which, is in MEM-stage, and the other is in WB-stage.

As the name suggests, Double Data hazard is the juxtaposition of the previous two data hazards. Now we should consider the latest updated register value. So, in the case of the Double Data hazard, we need to handle the EX hazard to get the updated data from the immediate prior instruction.

So, the conditions of the MEM hazard explained in the previous point, should additionally consider if this is not the case of the EX hazard.

We can finally write up a pseudocode for the MEM hazard with Double Data hazard handled as well.

```
if (MEM/WB.RegWrite && (MEM/WB. ResgisterRd ≠ 0)

   && !(EX/MEM.RegWrite && (EX/MEM. ResgisterRd ≠ 0)
   && (EX/MEM. ResgisterRd == ID/EX.ResgisterRs))

   && (MEM/WB. ResgisterRd == ID/EX.ResgisterRs))
        ForwardA = 01;


if (MEM/WB.RegWrite && (MEM/WB. ResgisterRd ≠ 0)

   && !(EX/MEM.RegWrite && (EX/MEM. ResgisterRd ≠ 0)
   && (EX/MEM. ResgisterRd == ID/EX.ResgisterRt))

   && (MEM/WB. ResgisterRd == ID/EX.ResgisterRt))
        ForwardB = 01;
```

These two branching procedures are implemented in our forwarding unit with comparators, NOT and AND gates as shown in the circuit diagram.

Simulator used

The simulation software used in this 8-bit MIPS Pipelined Execution experiment is ***logisim-win-2.7.1.exe.***

Discussion

- This assignment on Pipelined Execution of the 8-bit MIPS processor implementation is based on the idea and execution of the previous assignment on single cycle 8-bit MIPS processor.
- The same assembler written in C++ language to convert the given assembly code into 20-bit instructions was used in this assignment. This is why the opcode for the instructions and the control unit logic was kept the same as before.
- The instruction sets used in this assignment are only of R-type. So, there is no use of the Data memory unit. But it was kept as per instructions and to keep the basic 5 stages of the pipelined data path.
- As all the instructions are R-type, the registers can not have any new value from the instructions. To check values in the registers, the registers in the register file unit can be manually loaded with data in the Logisim simulator before executing the instructions.
- In R-type instructions, we were only concerned with the Data hazards, and have not taken any measures for other types of hazards. So, only forwarding aka bypassing is executed and no stalling is used.
- The pipeline registers are used as one single register of necessary data bit long (20/32/13 bits), as the Logisim simulator supports registers of arbitrary sizes. For a real life implementation, the pipeline registers can be made with multiple banks of registers of necessary bits.