Pipeline MIPS Architecture

Assumptions

1. Setting Reset signal on high will reset PC (=0).
2. Instruction Memory has 16 Memory Locations each memory location is 32-bit wide.
3. Data Memory has 16 Memory Locations each memory location is 32-bit wide.
4. Whenever the Program Counter is out of bound i.e. greater than 16. NOP is executed, to go back to initial state Reset should be made ON.

Instructions Implemented

a. NOP

b. R-Type: AND, OR, ADD, SUB, SLT, NOR

c. LW, SW

d. I-Type: ADDI, ANDI, LUI, ORI, SLTI, d. Branch: BEQ

Datapath

(diagram has been spilt into three parts for better understanding)

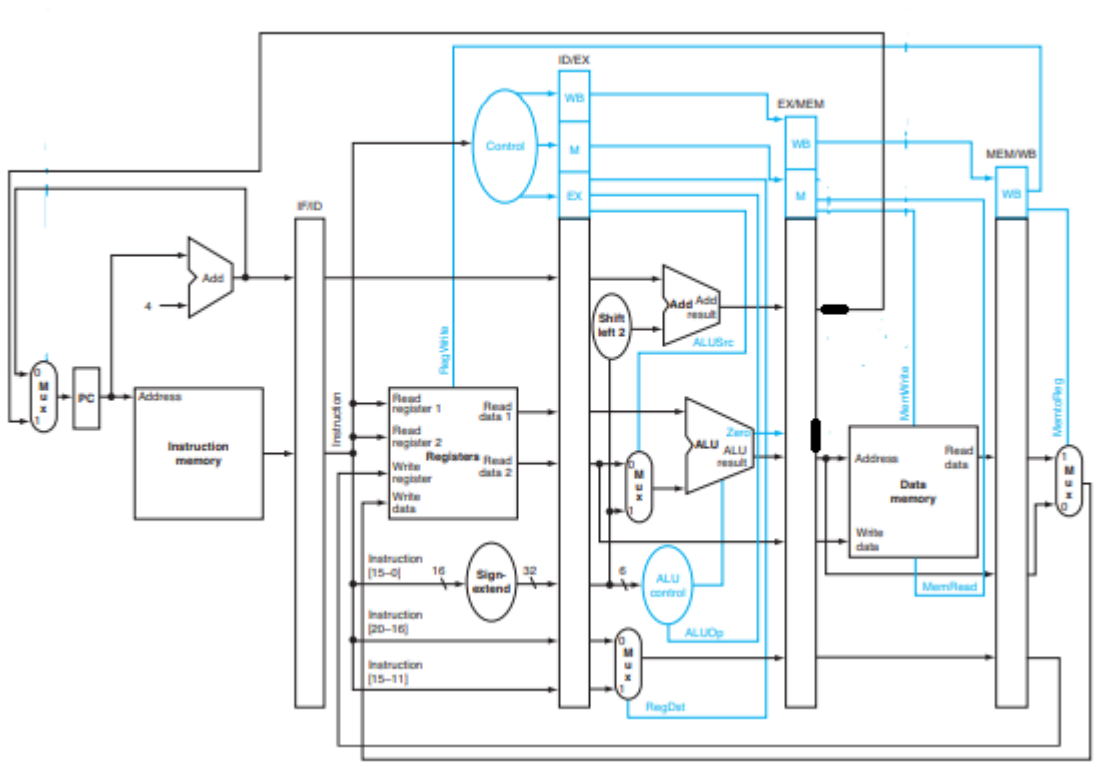

Fig.1 Pipelined Datapath

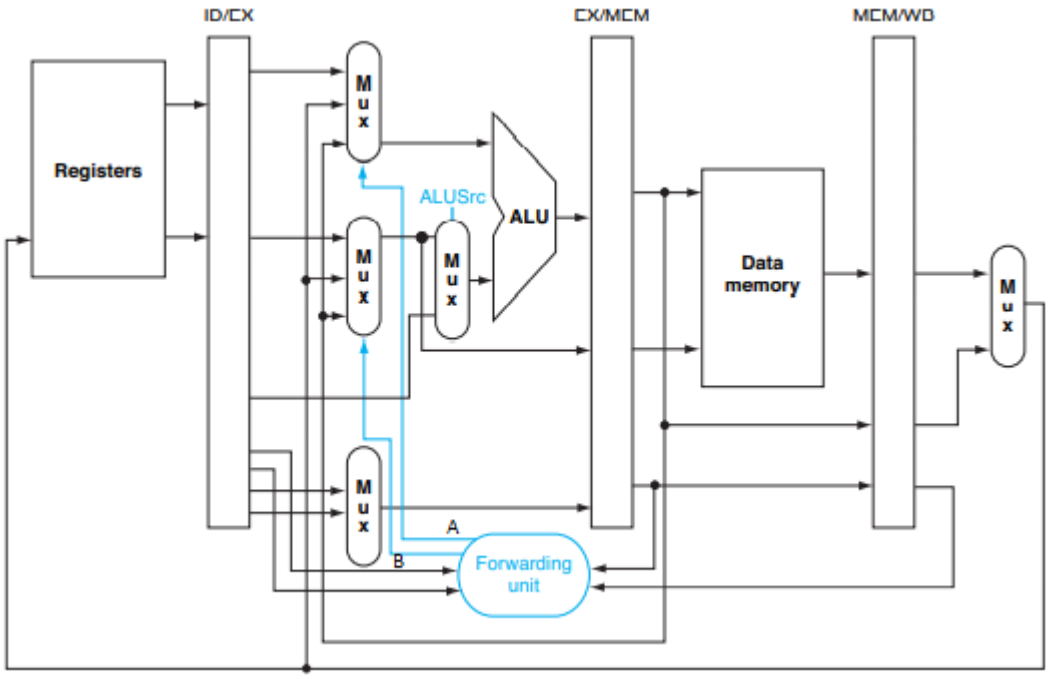

Fig.2 Datapath with Forwarding Unit

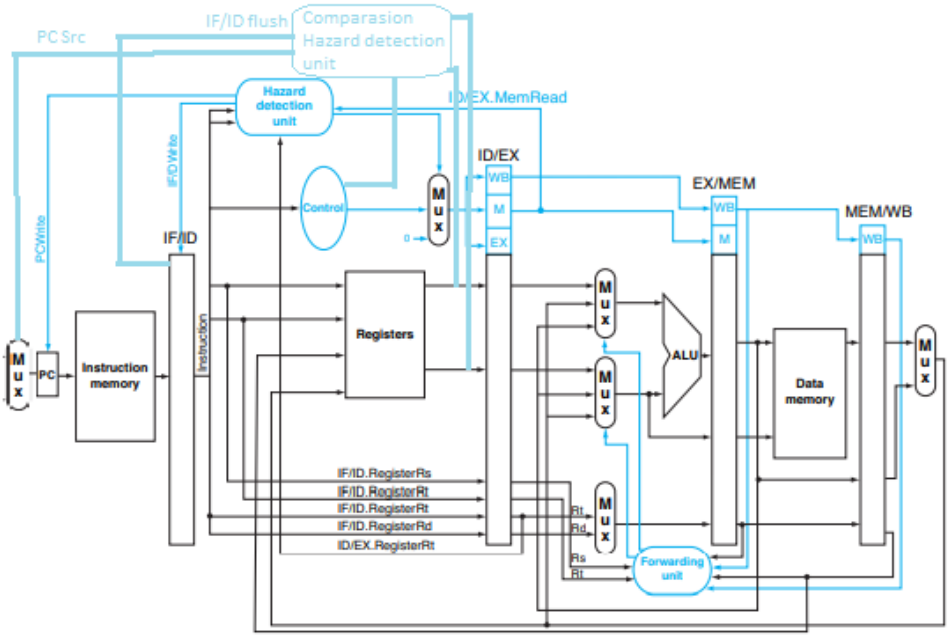

Fig.3 Datapath with Hazard Detection

Modules and their Functions

1. mux_32x2: A 32bit X 2-input multiplexer with 1-bit control and one output.

2. mux_32x3: A 32-bit X 3-input multiplexer with 2-bit control and one output.

3. mux_11x2: A 11-bit X 2-input multiplexer with 1-bit control and one output.

4. mux_5x2: A 5-bit X 2-input multiplexer with 1-bit control one output.

5. simple_alu_32bit: A simple 32-bit ALU with only one operation i.e. addition. It is used in calculation of next PC. Firstly, to calculate PC+1 and then for PC+1+Offset.

6. instr_mem: instruction memory which space for 16 32-bit instructions and it can be expanded further very easily. It takes PC as input and provides the instruction corresponding to it.

7. hazard_detect_unit: A hazard detection unit for generating signals to cope with hazards caused by LW instruction. It takes ID/EX.MemRead, ID/EX.RegisterRt, IF/ID.RegisterRs, and IF/ID.RegisterRt as input and provide output Control for 11 bit MUX used, IF/IDWrite, PCWrite.

8. compare_hazard: A hazard detection unit for generating signals to cope with hazards caused by BEQ instruction. It takes ReadData1, ReadData2 and Branch as input and provides IF/IDFlush and PCSrc as output.

9. ctrl_unit: Main control unit to generate signals for 32-bit 2-input multiplexers, 5-bit 2-input multiplexer and main ALU. It takes IF/ID.Opcode as input and provides control signals (aluOp, regDst, aluSrc, memToReg, regWrite, memWrite, memRead, branch)

10. registers_file: This module is used for 32 32-bit registers with ability to read or change their content.

11. sign_extend: It sign extends a 16-bit binary number to a 32-bit binary number. Used by LW and SW instruction to generate a 32-bit offset.

12. forwarding_unit: This module generates control signals, required by forwarding multiplexers, depending on different conditions.

13. alu_ctrl_unit: This module generates control signals required for main ALU w.r.t inputs (which are ALUOp and Function)

14. alu_32bit: This module is for the main ALU. It can perform add, sub, and, or, nor, less than, equal to.

15. data_mem: This module is the RAM of the system. It can store up to 16 32-bit of data which can be extended further very easily.

16. pipeline_mips_32bit: This module is the main one. It combines and completes all the connections required by the above modules to work and form a Pipeline MIPS Architecture.

17. testbench: this unit is the testbench for pipeline_mips_32bit module.

Examples

Initial Content of Register File.

0000001f	00000000000000000000000000000000 00000000000000000000000000000000
0000001d	00000000000000000000000000000000 00000000000000000000000000000000
0000001b	00000000000000000000000000000000 00000000000000000000000000000000
00000019	00000000000000000000000000000000 00000000000000000000000000000000
00000017	00000000000000000000000000001111 00000000000000000000000000001101
00000015	00000000000000000000000000001101 00000000000000000000000000001100
00000013	00000000000000000000000000001011 00000000000000000000000000001010
00000011	00000000000000000000000000001001 00000000000000000000000000001000
0000000f	00000000000000000000000000000111 00000000000000000000000000000110
0000000d	00000000000000000000000000000101 00000000000000000000000000000100
0000000b	00000000000000000000000000000011 00000000000000000000000000000010
00000009	00000000000000000000000000000001 00000000000000000000000000000000
00000007	00000000000000000000000000000000 00000000000000000000000000000000
00000005	00000000000000000000000000000000 00000000000000000000000000000000
00000003	00000000000000000000000000000000 00000000000000000000000000000000
00000001	00000000000000000000000000000000 00000000000000000000000000000000

Initial Content of Data Memory

0000000f	00000000000000000000000000000000 00000000000000000000000000000000
0000000d	00000000000000000000000000000000 00000000000000000000000000000000
0000000b	00000000000000000000000000000000 00000000000000000000000000000000
00000009	00000000000000000000000000000000 00000000000000000000000000000000
00000007	00000000000000000000000000001000 00000000000000000000000000000111
00000005	00000000000000000000000000000101 00000000000000000000000000000100
00000003	00000000000000000000000000000011 00000000000000000000000000000010
00000001	00000000000000000000000000000001 00000000000000000000000000000000

1.

```
rom[0] = 32'b00000001010010110100100000100010;  //sub t1 t2 t3;
rom[1] = 32'b00010010101101100000000000000010;  //beq s5 s6 0x2;
rom[2] = 32'b00000001100011010101100000100000;  //add t3 t4 t5;
rom[3] = 32'b00000001101011100110000000100000;  //add t4 t5 t6;
rom[4] = 32'b00000001110011111011010000100000;  //add t5 t6 t7;
rom[5] = 32'b10001110010100010000000000000000;  //lw s1 0x0 s2;
```

After implementing the instructions

Register File -

0000001f	00000000000000000000000000000000 00000000000000000000000000000000
0000001d	00000000000000000000000000000000 00000000000000000000000000000000
0000001b	00000000000000000000000000000000 00000000000000000000000000000000
00000019	00000000000000000000000000000000 00000000000000000000000000000000
00000017	00000000000000000000000000001111 00000000000000000000000000001101
00000015	00000000000000000000000000001101 00000000000000000000000000001100
00000013	00000000000000000000000000001011 00000000000000000000000000000001
00000011	00000000000000000000000000000011 00000000000000000000000000001000
0000000f	00000000000000000000000000000111 00000000000000000000000000000110
0000000d	00000000000000000000000000000101 00000000000000000000000000000100
0000000b	00000000000000000000000000000011 00000000000000000000000000000010
00000009	00000000000000000000000000000001 00000000000000000000000000000000
00000007	00000000000000000000000000000000 00000000000000000000000000000000
00000005	00000000000000000000000000000000 00000000000000000000000000000000
00000003	00000000000000000000000000000000 00000000000000000000000000000000
00000001	00000000000000000000000000000000 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Data Memory -

0000000f	00000000000000000000000000000000 00000000000000000000000000000000
0000000d	00000000000000000000000000000000 00000000000000000000000000000000
0000000b	00000000000000000000000000000000 00000000000000000000000000000000
00000009	00000000000000000000000000000000 00000000000000000000000000000000
00000007	00000000000000000000000000001000 00000000000000000000000000000111
00000005	00000000000000000000000000000101 00000000000000000000000000000100
00000003	00000000000000000000000000000011 00000000000000000000000000000010
00000001	00000000000000000000000000000001 00000000000000000000000000000000