Project 1: Pipelined CPU using Verilog

2018.11.21

Project 1

- 最多三人一組, 請在 NTU COOL 成員>小組 確認分組，分組有問題或是需要幫忙湊組員的話請寄信給助教
- Deadline:
  - Deadline: **12/11(二) (3 weeks) 11:59 PM**
  - **Please upload to ”NTU COOL”**
- Demo:
  - 另行公佈時段給各組填寫（約在繳交後一週左右）
  - 內容：執行程式給助教檢查，回答數個相關問題
  - 需全員到齊

Require

- Required Instruction Set：
  - and
  - or
  - add
  - sub
  - mul
  - addi
  - lw
  - sw
  - beq

Instruction	ALUOp	operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
ld	00	load doubleword	XXXXXXX	XXX	add	0010
sd	00	store doubleword	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001

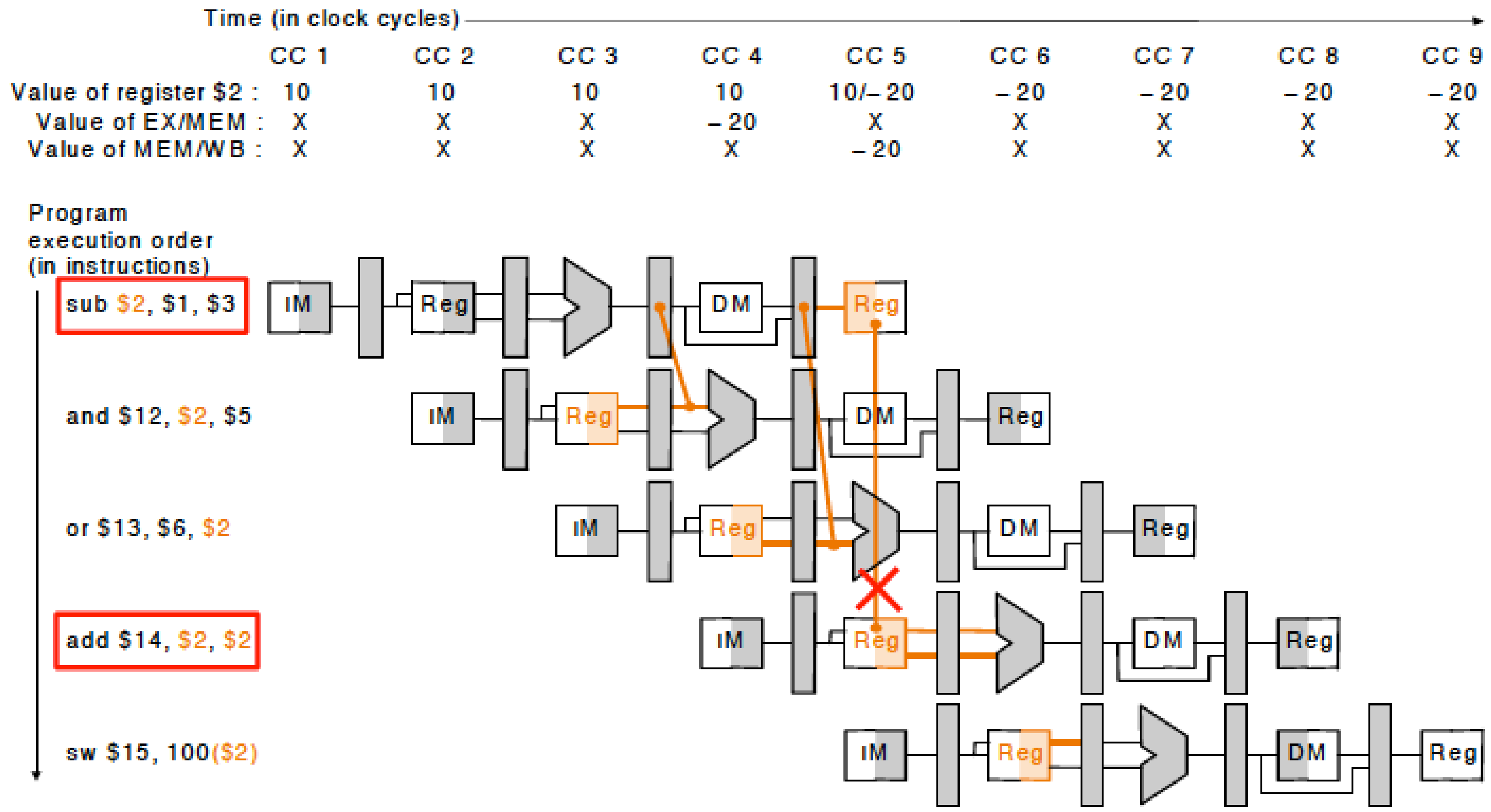
Instruction	Execution/address calculation stage control lines		Memory access stage control lines			Write-back stage control lines	
	ALUOp	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	10	0	0	0	0	1	0
ld	00	1	0	1	0	1	1
sd	00	1	0	0	1	0	X
beq	01	0	1	0	0	0	X

Require

- Register File : 32 Register (Write When clock rising edge)
- Instruction Memory : 1KB
- Data Memory : 32 Bytes
- Data Path & Module Name
- Hazard handling
  - Data hazard
    - Implement the Forwarding Unit to reduce or avoid the stall cycles.
    - The data dependency instruction follow "lw" must stall 1 cycle.
    - No need forwarding to ID stage
  - Control hazard :
    - The instruction follow 'beq' or 'j' instruction may need stall 1 cycle.
    - Pipeline Flush

Data hazard

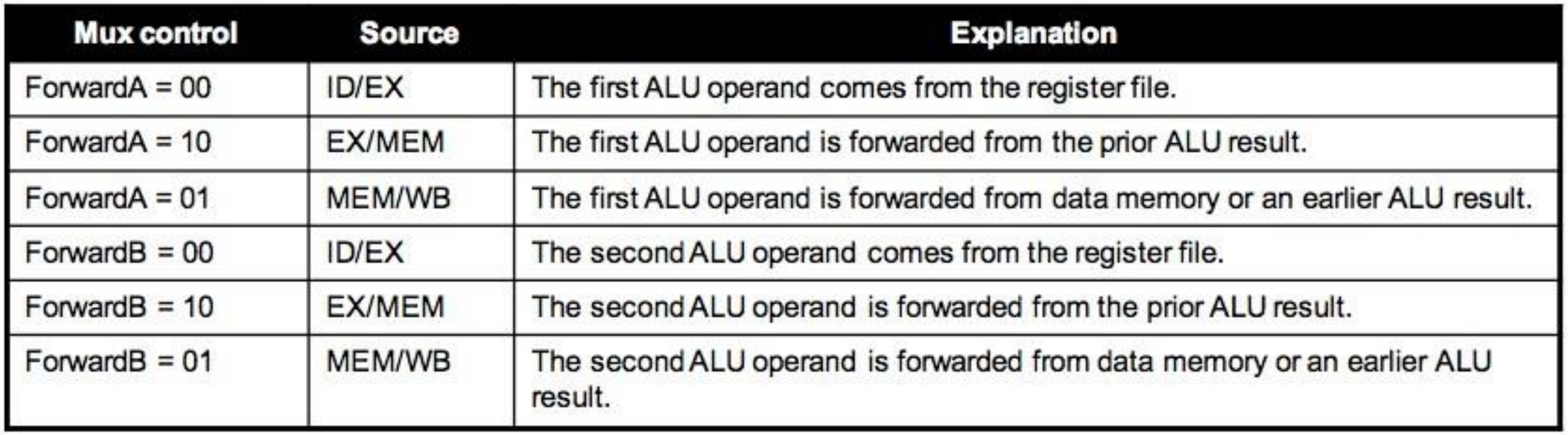
- 不需要forwarding 到 ID stage !



Time (in clock cycles)

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2 :	10	10	10	10	10/– 20	– 20	– 20	– 20	– 20
Value of EX/MEM :	X	X	X	– 20	X	X	X	X	X
Value of MEM/WB :	X	X	X	X	– 20	X	X	X	X

Forwarding Control



Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

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 (MEM/WB.RegisterRd = ID/EX.RegisterRs1))
ForwardA = 01
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs2))
ForwardB = 01
```

Stall & Flush

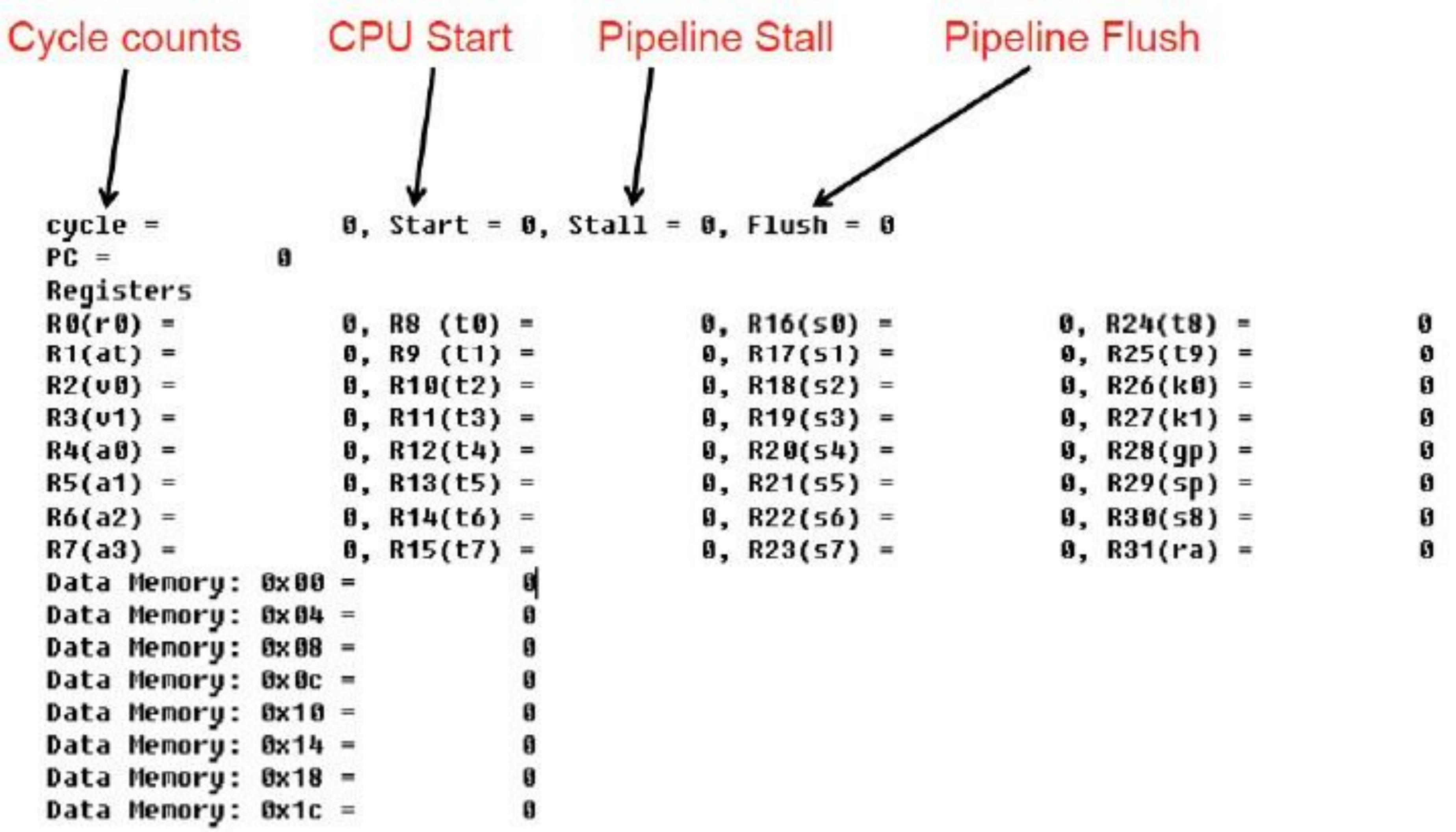
- In testbench.v
- Can be changed depends on your own design.

```
// put in your own signal to count stall and flush

   if (CPU.HazzardDetection.mux8_o == 1 && CPU.Control.Jump_o == 0
&& CPU.Control.Branch_o == 0)
            stall = stall + 1;

   if(CPU.HazzardDetection.Flush_o == 1)
            flush = flush + 1;
```

Require (cont.)

- (80%) Source code (put all .v file into “code” directory)
  - CPU module
    - Basic (40%)
    - Data forwarding (20%)
    - Data hazard (lw stall) (10%)
    - Control hazard (flush) (10%)
  - TestBench (may need to modified)
    - Initialize storage units
    - Load instruction.txt into instruction memory
    - Create clock signal
    - Output cycle count in each cycle
    - Output Register File & Data Memory in each cycle
    - Print result to output.txt
  - TestData
    - Fibonacci
- (20%) Report (project1_teamXX.pdf)
  - Members & Team Work
    - 須註明組員工作分配比例
  - How do you implement this Pipelined CPU.
  - Explain the implementation of each module.
  - Problems and solution of this project.
- Put all files and directory into *project1_teamXX_V0*

Output Example



```
cycle =          0, Start = 0, Stall = 0, Flush = 0
PC =         0
Registers
R0(r0) =         0, R8 (t0) =         0, R16(s0) =         0, R24(t8) =         0
R1(at) =         0, R9 (t1) =         0, R17(s1) =         0, R25(t9) =         0
R2(v0) =         0, R10(t2) =         0, R18(s2) =         0, R26(k0) =         0
R3(v1) =         0, R11(t3) =         0, R19(s3) =         0, R27(k1) =         0
R4(a0) =         0, R12(t4) =         0, R20(s4) =         0, R28(gp) =         0
R5(a1) =         0, R13(t5) =         0, R21(s5) =         0, R29(sp) =         0
R6(a2) =         0, R14(t6) =         0, R22(s6) =         0, R30(s8) =         0
R7(a3) =         0, R15(t7) =         0, R23(s7) =         0, R31(ra) =         0
Data Memory: 0x00 =         0
Data Memory: 0x04 =         0
Data Memory: 0x08 =         0
Data Memory: 0x0c =         0
Data Memory: 0x10 =         0
Data Memory: 0x14 =         0
Data Memory: 0x18 =         0
Data Memory: 0x1c =         0
```

Data Path & Module

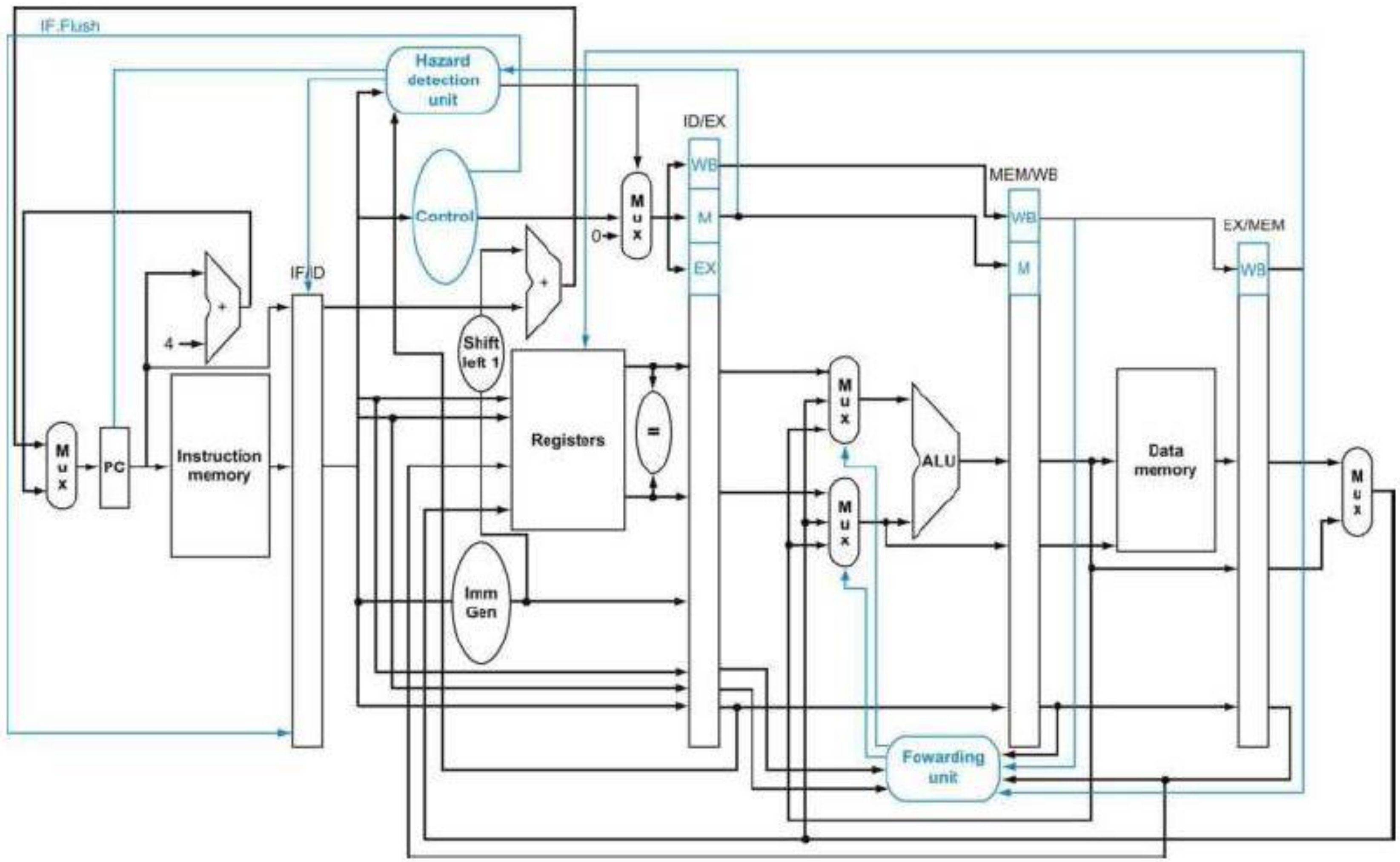


FIGURE 4.62 The final datapath and control for this chapter.