

Project 1: Pipelined CPU using Verilog

2017.11.22

Project 1

- 最多三人一組, 請在 (<https://goo.gl/UAaM3v>) 確認分組，分組有問題的話請寄信給助教
- **Deadline:**
 - Deadline: **12/11(一) (3 weeks) 11:59 PM**
 - **Please upload to "ceiba"**
- **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
 - j

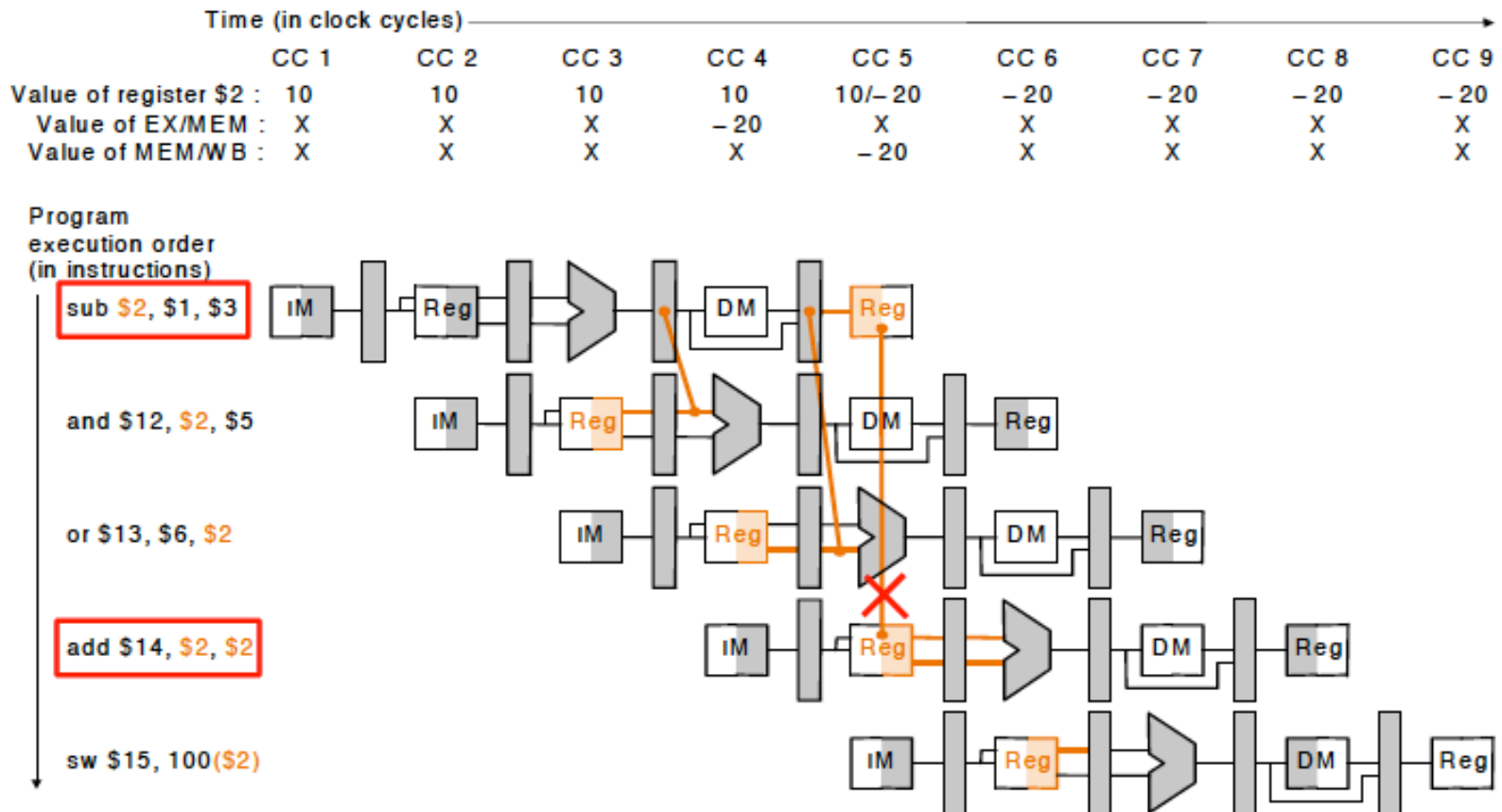
op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	ori	lw	sw	beq	jump
RegDst	1	0	0	x	x	x
ALUSrc	0	1	1	1	0	x
MemtoReg	0	0	1	x	x	x
RegWrite	1	1	1	0	0	0
MemWrite	0	0	0	1	0	0
Branch	0	0	0	0	1	0
Jump	0	0	0	0	0	1
ExtOp	x	0	1	1	x	x
ALUop<N:0>	“R-type”	Or	Add	Add	Subtract	xxx

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 !



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 \neq 0)
 and (EX/MEM.RegisterRd=ID/EX.RegisterRs))
 ForwardA = 10

if (EX/MEM.RegWrite
 and (EX/MEM.RegisterRd \neq 0)
 and (EX/MEM.RegisterRd=ID/Ex.RegisterRt))
 ForwardB = 10

2. MEM hazard

if (MEM/WB.RegWrite
 and (MEM/WB.RegisterRd \neq 0)
 and (Ex/MEM.RegisterRd \neq ID/Ex.RegisterRs)
 and (MEM/WB.RegRd=ID/Ex.RegisterRs)) ForwardA = 01

if (MEM/WB.RegWrite
 and (MEM/WB.RegRd \neq 0)
 and (Ex/MEM.RegisterRd \neq ID/Ex.RegisterRt)
 and (MEM/WB.RegRd=ID/Ex.RegisterRt)) 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 counts

CPU Start

Pipeline Stall

Pipeline Flush

cycle =

PC =

Registers

R0(r0) =

R1(at) =

R2(v0) =

R3(v1) =

R4(a0) =

R5(a1) =

R6(a2) =

R7(a3) =

Data Memory: 0x00 =

Data Memory: 0x04 =

Data Memory: 0x08 =

Data Memory: 0x0c =

Data Memory: 0x10 =

Data Memory: 0x14 =

Data Memory: 0x18 =

Data Memory: 0x1c =

0, Start = 0, Stall = 0, Flush = 0

0, R8 (t0) =

0, R9 (t1) =

0, R10(t2) =

0, R11(t3) =

0, R12(t4) =

0, R13(t5) =

0, R14(t6) =

0, R15(t7) =

0, R16(s0) =

0, R17(s1) =

0, R18(s2) =

0, R19(s3) =

0, R20(s4) =

0, R21(s5) =

0, R22(s6) =

0, R23(s7) =

0, R24(t8) =

0, R25(t9) =

0, R26(k0) =

0, R27(k1) =

0, R28(gp) =

0, R29(sp) =

0, R30(s8) =

0, R31(ra) =

0

0

0

0

0

0

0

0

Data Path & Module

