

Project 1: Pipelined CPU using Verilog

Project 1

- 最多三人一組, 請在**11/18(三)**前至 <https://goo.gl/DKEpO1> 填寫分組名單
- 填寫分組名單
 - 內文請填上組員"學號"與"姓名"
 - **11/18(三)** 半夜十二點前未填寫組員名單者, 將由助教幫你分組
 - 分組名單將於**11/19(四)** 下午兩點 公布
- **Deadline:**
 - **Deadline: 12/7(一) (3 weeks) 午夜12:00**
- **Demo:**
 - 另行公佈 (約在繳交後一週左右)

Require

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

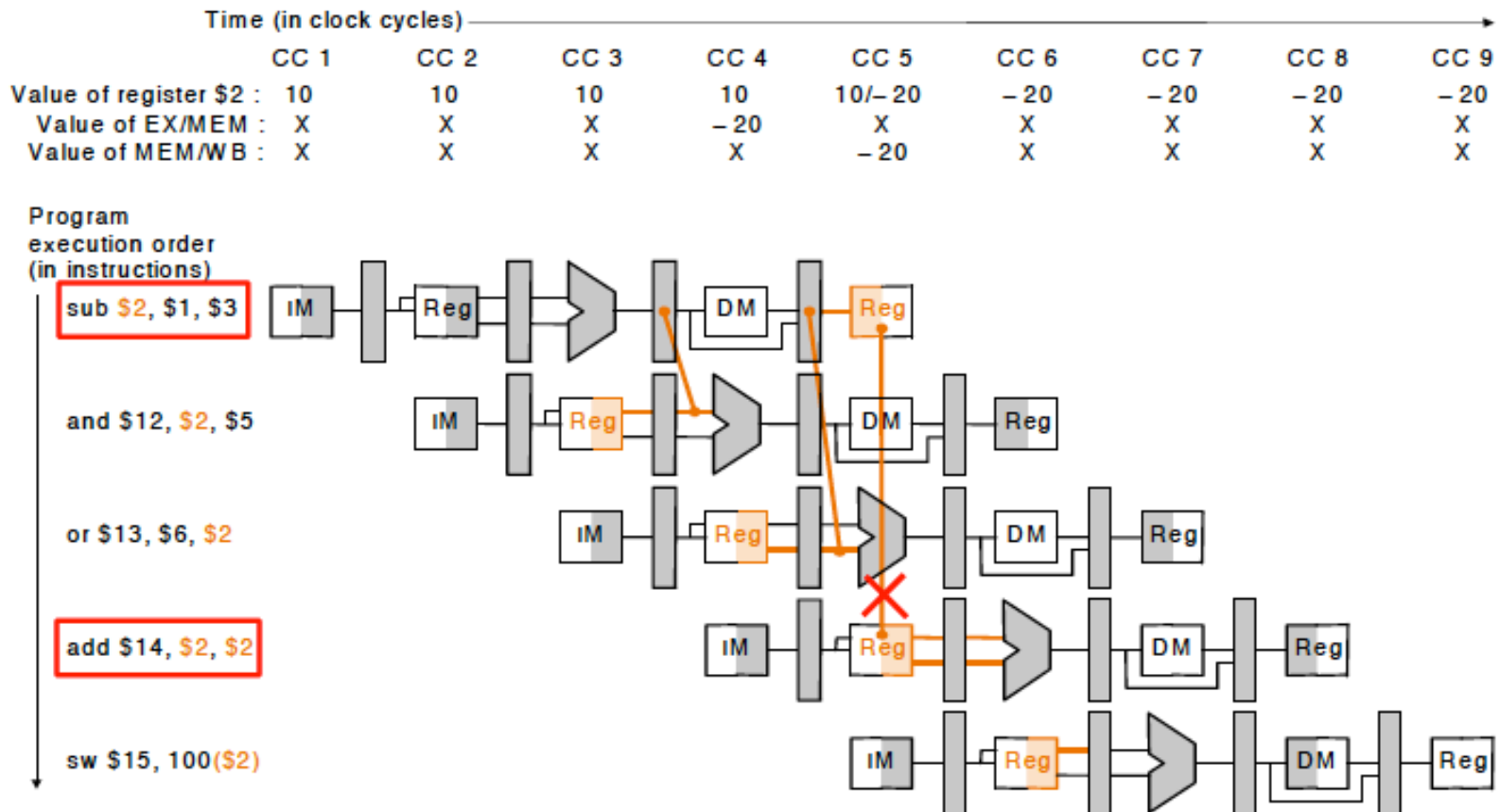
Require

- Register File : 32 Register (**Write When clock rising edge**)
- Instruction Memory : 1KB
- Data Memory : 32 Bytes
- Data Path & Module Name
- MUL OpCode :

0	rs	rt	rd	0	0x18
---	----	----	----	---	------
- 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 !



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.doc)
 - 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.zip*

Require (cont.)

- Fibonacci
 - Set Input n into data memory at 0x00
 - CPU.Data_Memory.memory[0] = 8'h5;

Change instruction sequence if needed, to avoid ID-forwarding.

```
// Initialization
addi $a0, $r0, 0 // $a0 = 0
addi $t0, $r0, 0 // $t0 = 0
addi $t1, $r0, 1 // $t1 = 1
addi $s1, $r0, 0 // $s1 = 0
addi $s2, $r0, 1 // $s2 = 1

// Load number n from data memory
lw $s0, 0($a0)
addi $s4, $s0, 0 // save input value in $s4

// Calculate the Fibonacci Number
loop:
sub $s0, $s0, $t1 //subu $s0, 1
add $s1, $s1, $s2
addi $s3, $s2, 0 //move $s3, $s2
addi $s2, $s1, 0 //move $s2, $s1
addi $s1, $s3, 0 //move $s1, $s3
beq $s0, $t0, finish //bne $s0, $zero, loop
j loop
```

```
// Store the result
finish:
sw $s1, 4($a0)

// Calculate the input and output and Store
and $t2, $s1, $r0
or $t3, $s1, $r0
add $t4, $s1, $t4
sub $t5, $s1, $t4
mul $t6, $s1, $t4
sw $t2, 8($a0)
sw $t3, 12($a0)
sw $t4, 16($a0)
sw $t5, 20($a0)
sw $t6, 24($a0)
```

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

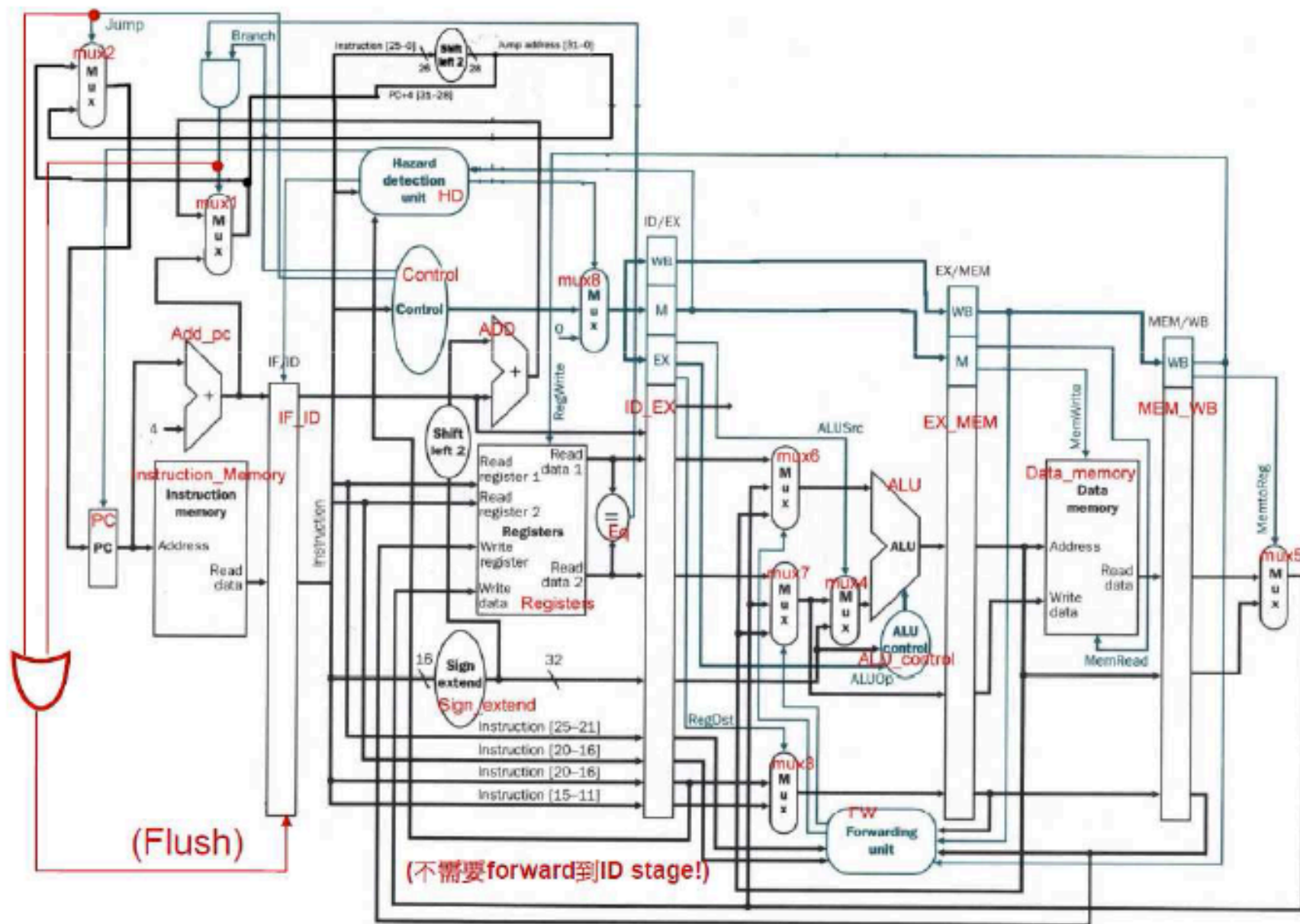
0

0

0

0

Data Path & Module



Pipeline Example

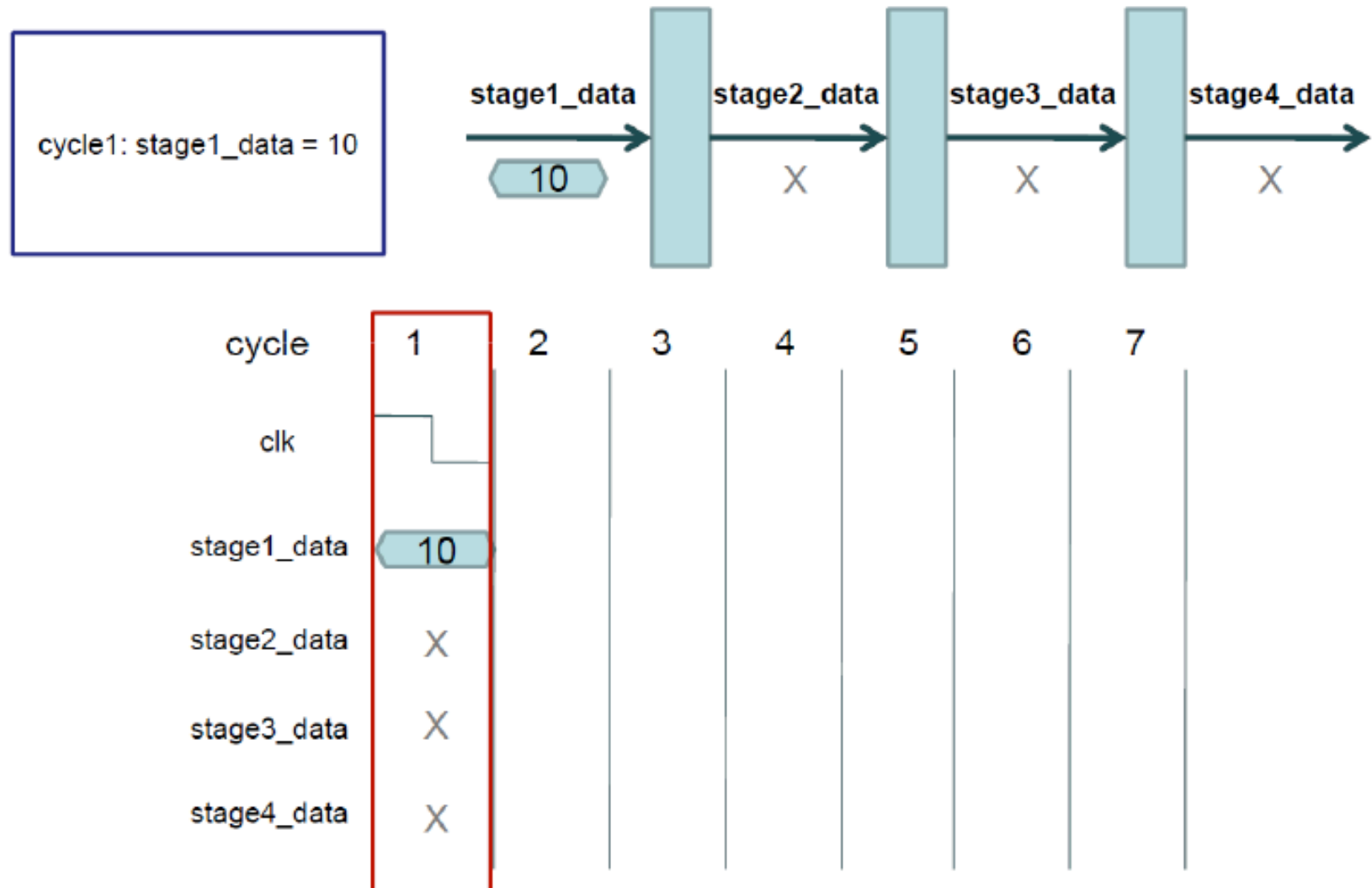


```
always@(posedge clk) begin
    stage2_data <= stage1_data;
end

always@(posedge clk) begin
    stage3_data <= stage2_data;
end

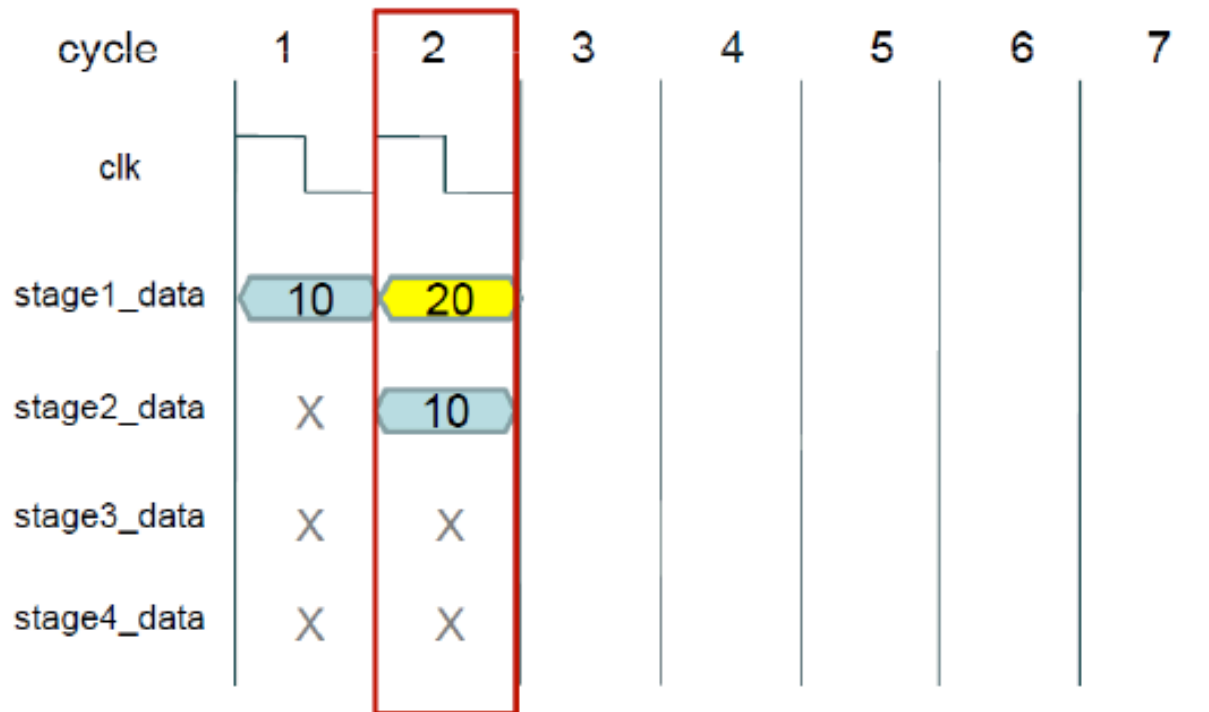
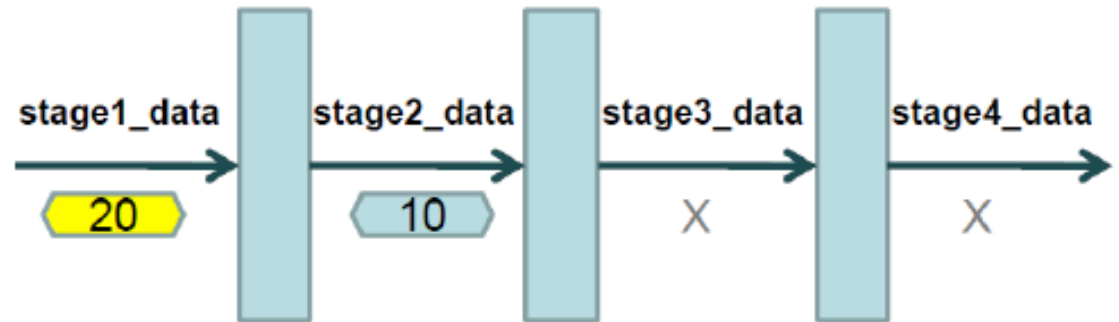
always@(posedge clk) begin
    stage4_data <= stage3_data;
end
```

Pipeline (Cycle 1)



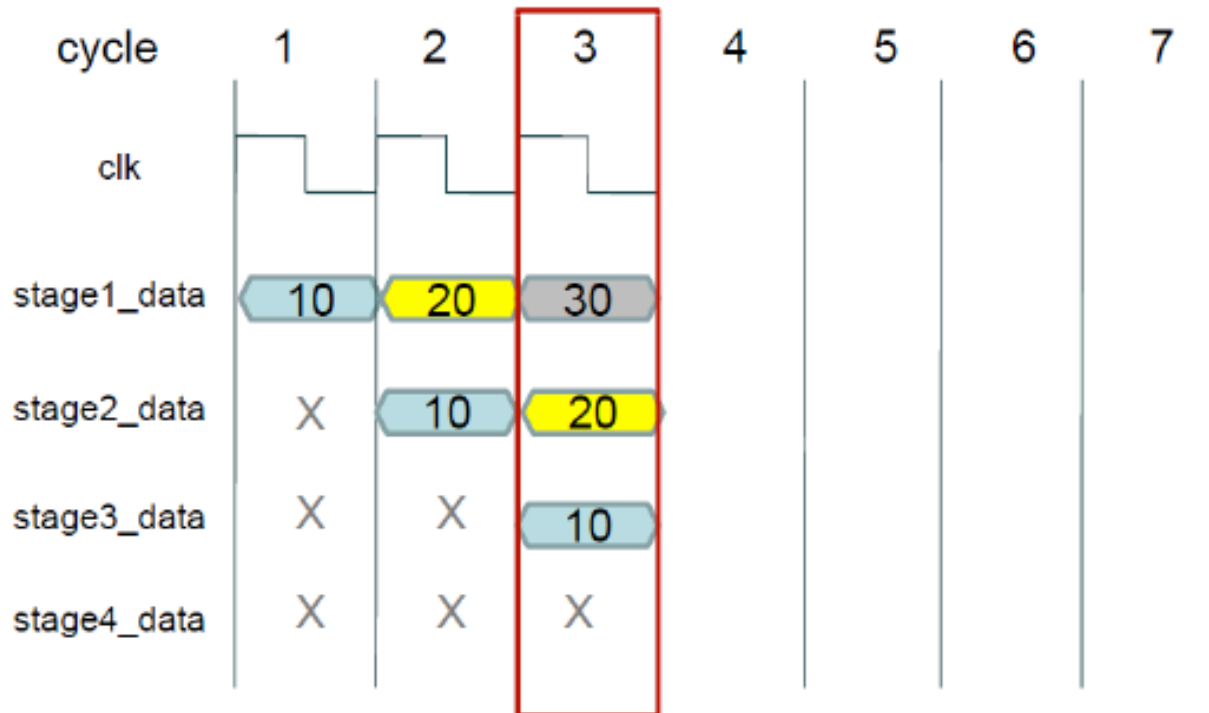
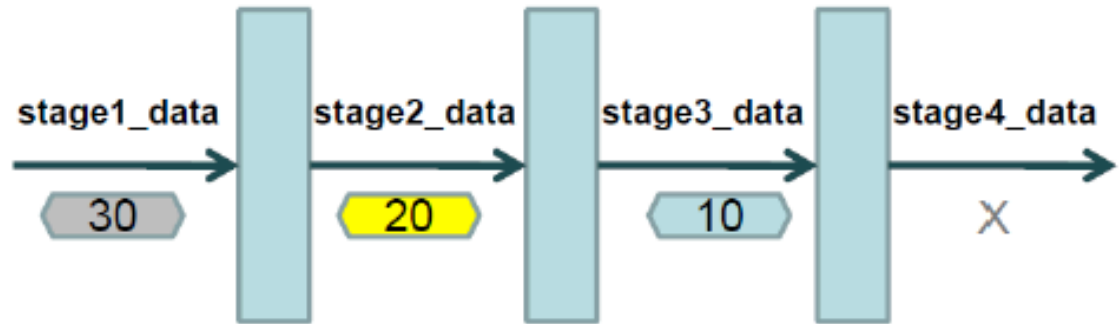
Pipeline (Cycle 2)

cycle1: stage1_data = 10
cycle2: stage1_data = 20



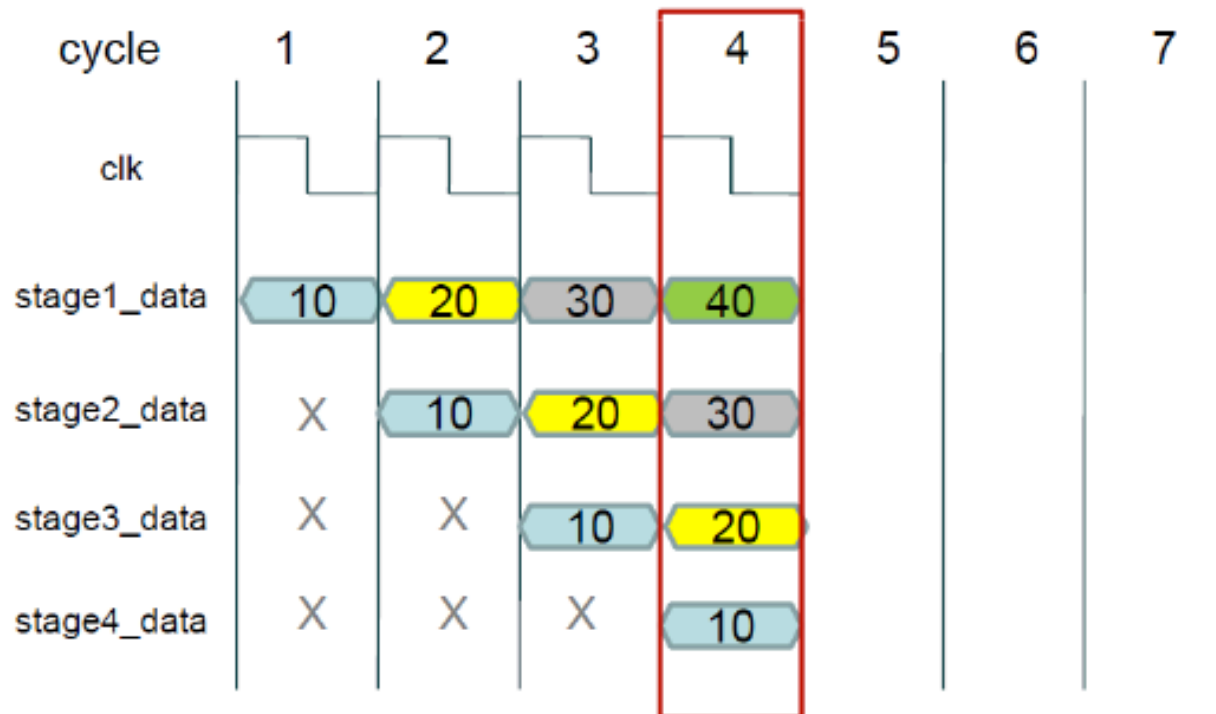
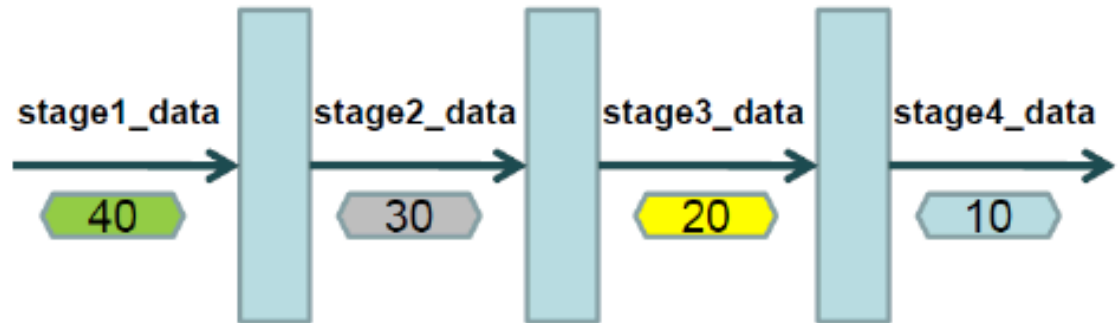
Pipeline (Cycle 3)

cycle1: stage1_data = 10
cycle2: stage1_data = 20
cycle3: stage1_data = 30



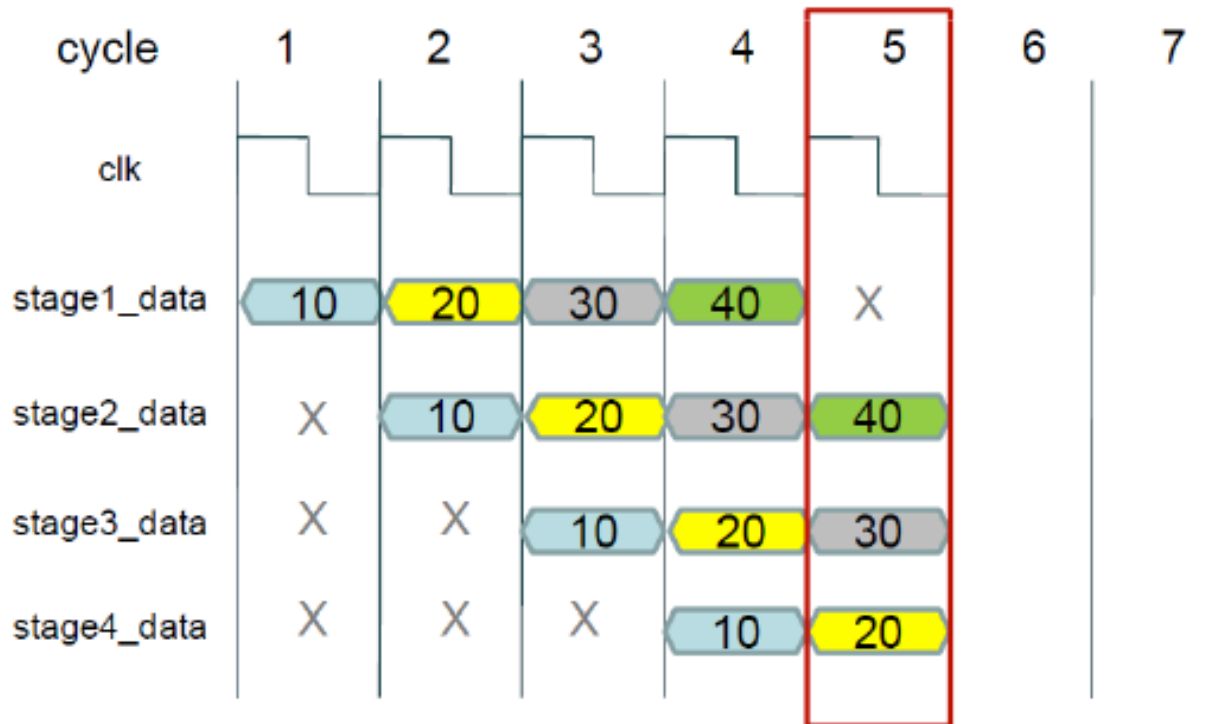
Pipeline (Cycle 4)

cycle1: stage1_data = 10
cycle2: stage1_data = 20
cycle3: stage1_data = 30
cycle4: stage1_data = 40



Pipeline (Cycle 5)

cycle1: stage1_data = 10
cycle2: stage1_data = 20
cycle3: stage1_data = 30
cycle4: stage1_data = 40



Pipeline (Cycle 6)

cycle1: stage1_data = 10
cycle2: stage1_data = 20
cycle3: stage1_data = 30
cycle4: stage1_data = 40



cycle	1	2	3	4	5	6	7
clk							
stage1_data	10	20	30	40	X	X	
stage2_data	X	10	20	30	40	X	
stage3_data	X	X	10	20	30	40	
stage4_data	X	X	X	10	20	30	

Pipeline (Cycle 7)

cycle1: stage1_data = 10
cycle2: stage1_data = 20
cycle3: stage1_data = 30
cycle4: stage1_data = 40

