

15/09/2015
L-34

PIPE Line Modeling (Part 2)

A more complex example

- i) Consider the pipeline that carries out the following stage operations

ii) Inputs:

- i) Three register addresses

rs1 register source 1, 2

rs2

rd register destination

- ii) ALU function (func)

- iii) and memory address (addr)

iii) Stage 1: read two 16-bit numbers from the registers specified by "rs1" and "rs2" and store them in "A" & "B"

iv) Stage 2: perform ALU operation on A & B specified by "func", and store in "Z"

- iv) Stage 3: write the value of "z" in the register specified by "rd"
 v) Stage 4: also write the same value of z in memory whose location is specified by "addr"

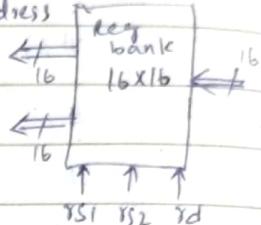
Design The Assumptions

1) There is a register bank containing 16 16-bit registers

→ 4 bits are required to specify a register address

→ 2 register reads & 1 register write can be performed at a time every clock cycle.

→ Register addresses are "rs1", "rs2" & "rd"



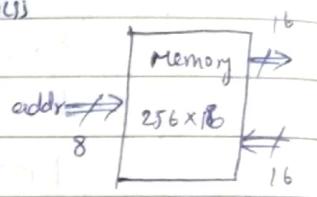
2) Memory organised as 256×16

→ 8-bits are required to specify memory address

→ Each location contains 16 bit of data

which can be read in a single cycle

→ Memory address specified by "addr"



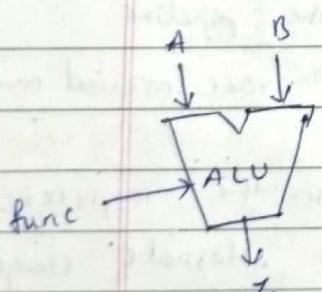
3) ALU functions selected by 4-bit "func"

0000 : ADD 0100 : SELB 1000 : NEGA

0001 : SUB 0101 : AND 1001 : NEG_B

0010 : MUL 0110 : OR 1010 : SRA

0011 : SELA 0111 : XOR 1011 : SLA



ADD : $Z = A + B$ AND : $A \& B$ SRA : $A \gg 1$

SUB : $Z = A - B$

OR : $A | B$

SRA : $A \gg 1$

MUL : $A * B$

XOR : $A ^ B$

SLA : $A \ll 1$

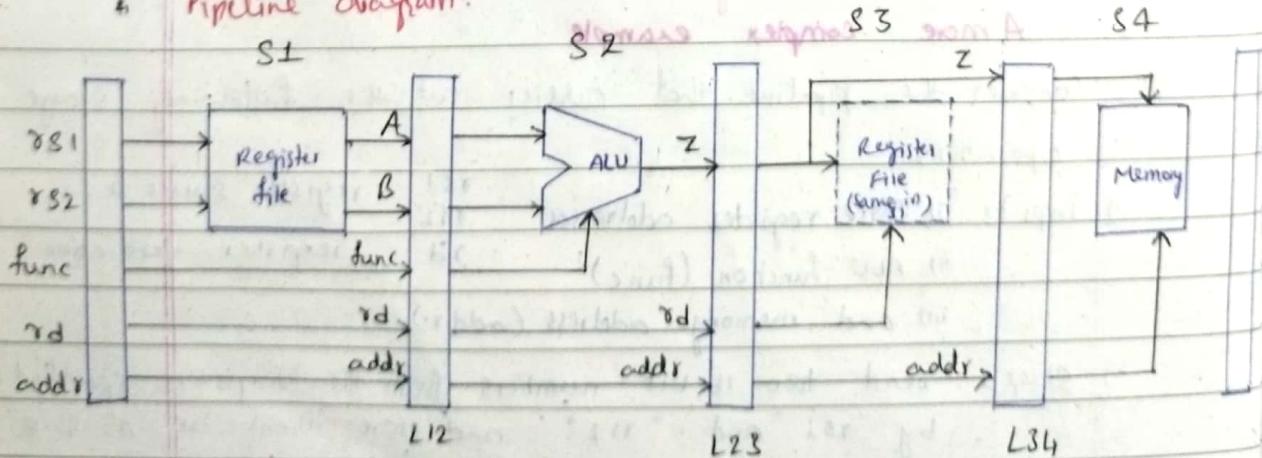
SELA : $Z = A$

NEGA : $\sim A$

SELB : $Z = B$

NEG_B : $\sim B$

Pipeline diagram:



Clocking Issue in Pipeline

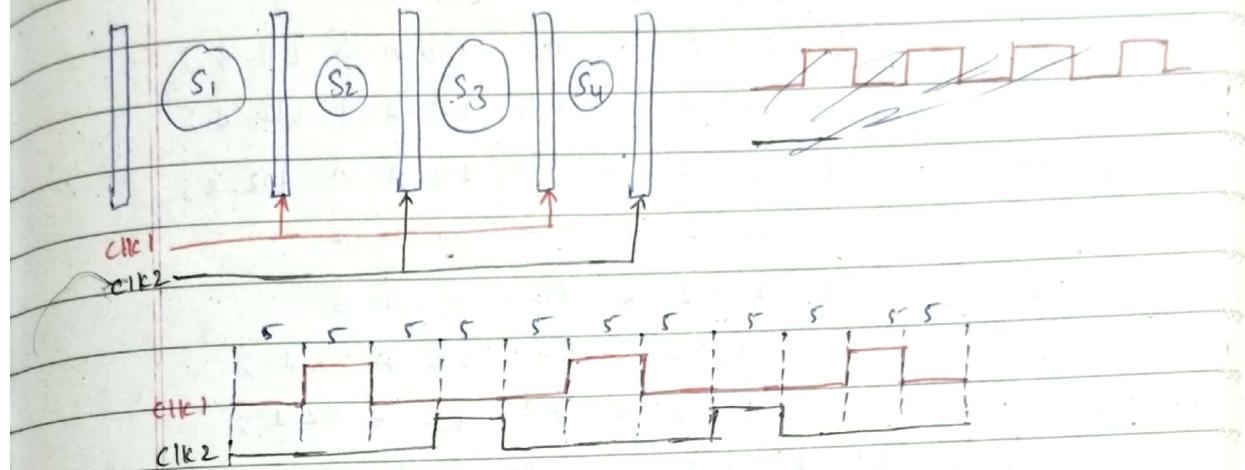
- * It is very important that the consecutive stages be applied suitable clocks for correct operation.

Page No.:

Date:

- * TWO options

- i) use master slave FF in latches to avoid race condition
- ii) use non-overlapping two-phase clock for the consecutive pipeline stages



Pipeline Modeling Verilog code

```
module pipe_ex2 (zout, rs1, rs2, rd, func, addr, clk1, clk2);  
    input [3:0] rs1, rs2, rd, func;  
    input [7:0] addr;  
    output [15:0] zout;  
    input clk1, clk2; // Two phase clock  
    reg [15:0] L12_A, L12_B, L23_Z, L34_Z;  
    reg [3:0] L12_rd, L12_func, L23_rd;  
    reg [7:0] L12_addr, L23_addr, L34_addr;  
    reg [15:0] regbank [0:15]; // Register bank  
    reg [15:0] mem [0:255]; // Memory  
    assign zout = L34_Z;  
    // Stage 1  
    always @ (posedge clk1)  
        begin  
            L12_A <= #2 regbank[rs1];  
            L12_B <= #2 regbank[rs2];  
            L12_rd <= #2 rd;  
            L12_func <= #2 func;  
            L12_addr <= #2 addr;  
        end
```

// Stage 2

always @ (posedge clk2)

begin

case (func)

0 : L23_Z <= #2 L12_A + L12_B;

1 : L23_Z <= #2 L12_A - L12_B;

2 : L23_Z <= #2 L12_A * L12_B;

3 : L23_Z <= #2 L12_A ^ L12_B;

4 : L23_Z <= #2 L12_A & L12_B;

5 : L23_Z <= #2 L12_A &~ L12_B;

6 : L23_Z <= #2 L12_A | L12_B;

7 : L23_Z <= #2 L12_A ^~ L12_B;

8 : L23_Z <= #2 ~ L12_A;

9 : L23_Z <= #2 ~ L12_B;

10 : L23_Z <= #2 L12_A >> 1;

11 : L23_Z <= #2 L12_B << 1;

default : L23_Z <= #2 16'hxxxx;

endcase

L23_rd <= #2 L12_rd;

L23_addr <= #2 L12_addr;

end

// Stage 3

always @ (posedge clk2)

begin

reg bank [L23_rd] <= #2 L23_Z;

L34_Z <= #2 L23_Z;

L34_addr <= #2 L23_addr;

end

// Stage 4

always @ (posedge clk2)

begin

mem [L34_addr] <= #2 L34_Z;

end

endmodule

Test bench

module pipe2_test;

wire [5:0] z;

reg [3:0] rs1, rs2, rd, func;

reg [7:0] addr;

reg clk1, clk2;

integer k;

pipe_ex2 # MY_PIPE (z, rs1, rs2, rd, func, addr, clk1, clk2);

// generating two phase clock

```
initial
begin
    clk1 = 0 ; clk2 = 0 ;
repeat(20)
begin
    #5 clk1 = 1 ; #5 clk2 = 0 ;
    #5 clk2 = 1 ; #5 clk1 = 0 ;
end
end
```

Page No.:

Date:

// initializing registers

```
initial
for (k=0; k<16; k=k+1)
    MYPipe.regbank[k] = k;
```

// testing o/p with various ip's

```
initial
begin
    #5 y31=3; y32=5; rd=10; func=0; addrs=125; // A00
    #20 3; 8; 12; 2; 16; // MUL
    #20 10; 5; 14; 1; 128; // SUB
    #20 7; 3; 13; 11; 127; // SLA
    #20 10; 5; 15; 1; 129; // SUB
    #20 12; 13; 16; 0; 130; // ADD
    #60 for (k=125; k<131; k=k+1)
        $display ("Mem [%d.d] = %d", k, MYPipe.mem[k])
end
```

initial

```
begin
    $dumpfile ("pipe2.vcd");
    $dumpvars (0, pipe2-test);
    $monitor ("Time: %3d, f=%3d", $time, z);
    #300 ($finish); noflush;
end
```

end module.

Simulation Result:

Time : 0, f = x	Mem [127] = 14
27, 8	[128] = 3
47, 24	[129] = 3
67, 3	[130] = 38
87, 14	
107, 3	
127, 38	

Mem [125] = 8

[126] = 24