

15/09/2015

L-34

## PIPE line Modeling (Part 2)

### A more complex example

Q) Consider the pipeline that carries out the following stage operations

- i) Inputs:
- i) Three register addresses  
rs1      register source 1, 2  
rs2  
rd      register destination
  - ii) ALU function (func)
  - iii) and memory address (addr)

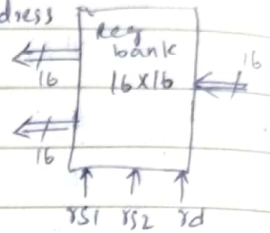
ii) Stage 1: read two 16-bit numbers from the registers specified by "rs1" and "rs2" and store them in "A" & "B"

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

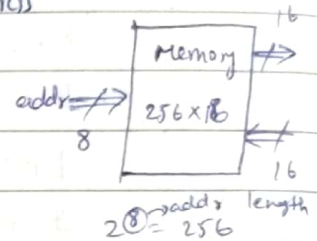
- iv) stage 3: write the value of "Z" in the register specified by "rd"
- v) stage 4: Also write the same value of "Z" in memory whose location is specified by "addr"

### 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"

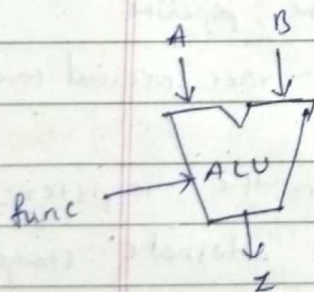


- 2) Memory organised as 256x16
  - 8-bits are required to specify memory address
  - Each location contains 16 bit of data which can be read in a single cycle
  - Memory address specified by "addr"



- 3) ALU functions selected by 4bit "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 :  $A | B$

SLA :  $A \ll 1$

MUL :  $A * B$

XOR :  $A \wedge B$

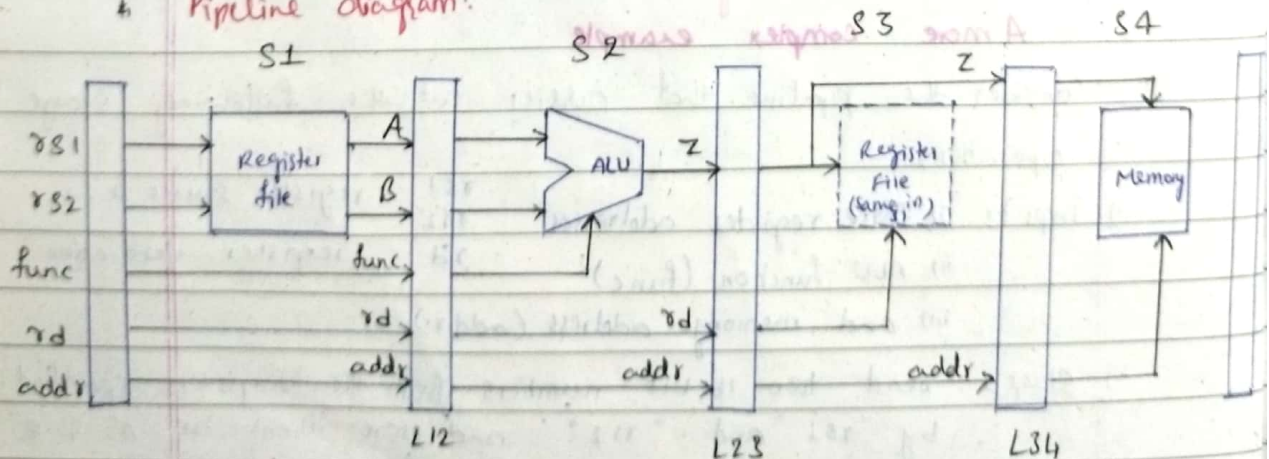
NEGA :  $\sim A$

SELA :  $Z = A$

SELB :  $Z = B$

NEG B :  $\sim B$

### Pipeline diagram:

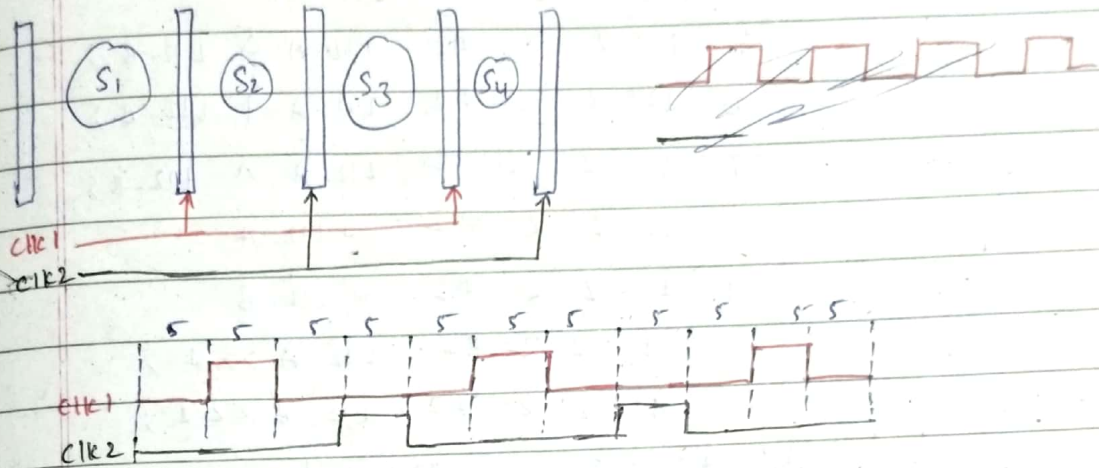


## Clocking Issue in Pipeline

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

\* Two options

- use master slave FF in latches to avoid race condition
- 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
end
    
```

// stage 2  
always @(negedge 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;  
4 : L23-Z <= #2 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 clk1)

begin

regbank [L23-rd] <= #2 L23-Z;

L34-Z <= #2 L23-Z;

L34-addr <= #2 L23-addr;

end

// stage 4

always @(negedge clk2)

begin

mem [L34-addr] <= #2 L34-Z;

end

endmodule

**Test bench**

module pipe2-test;

wire [15:0] Z;

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

reg [7:0] addr;

reg clk1, clk2;

integer k;

pipe\_ex2 M4 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 clk1 = 0 ;

#5 clk2 = 1 ; #5 clk2 = 0 ;

end

end

// initializing registers

initial

for (K=0 ; K<16 ; K=K+1)

MYPIPE.regbank[K] = K ;

// testing o/p with various i/p's

initial

begin

#5 r31=3 ; r32=5 ; rd=10 ; hunc=0 ; addr=125 ; // A00

#20 3 ; 8 ; 12 ; 2 ; 126 ; // 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 [%3d] = %3d", K, MYPIPE.mem[K]);

end

initial

begin

\$dumpfile ("pipe2.vcd");

\$dumpvars (0, pipe2-test);

\$monitor ("Time : %3d , F = %3d", \$time, z);

#300 \$finish;

end

endmodule.

Simulation Result:

Time : 0, F = x

27, 8

47, 24

67, 3

87, 14

107, 3

127, 38

Mem [127] = 14

[128] = 3

[129] = 3

[130] = 38

Mem [125] = 8

[126] = 24