Linh Nguyen

B04

ENCM 369 – Computer Organization

Lab 9

Ex. A) Part I

1. The critical path in the Fetch stage only goes through the Instruction Memory.

2. fc = 3.333 GHz -> Tc = 1/fc = 300.03 ps. The Instruction Memory will need to speed up:

Part II

1. The critical path in the Execute stage goes through the multiplexer and the ALU

The time constraint is satisfied, therefore no modifications need to be made to the Execute stage.

Part III

1. The critical path that provides input to the PC register goes through the adder and the mux in the F stage:

The tpd through the AND gate in the Memory stage and the mux in the F stage is:

Therefore, in the worst-case scenario, the PC input will be ready in:

The minimum clock period will be:

1. The critical path only involves the Data Memory.
2. Tc = 300.03 ps. The Data Memory will need to speed up:

Part IV

1. Setup-time constraint for the half-cycle before an R-file update:
2. Setup-time constraint for the half-cycle before an update to the register:
3. Tc = 300.03 ps which satisfies both time constraints, therefore nothing about the designs of the Writeback and Decode stages will prevent the use of a 3.333 GHz clock.

Part V

1. Using all the constraints found:

For reliable operation of all five pipeline stages, the minimum clock period will be the greatest *Tc* constraint from above to satisfy the rest, which is

1. There were no changes needed to allow a clock frequency of 3.333 GHz in the Execute, Writeback, or Decode stage. However, in both the Fetch and Memory stage the changes needed were:

Both D-Mem and I-Mem need to speed up to 253.03 ps or less to allow a clock frequency of 3.333 GHz.

Ex B)

1. At *t = 45.5 ns*, 0x0040\_00a8 is written to the PC. Shortly after that, the values for InstrD and PCPlus4D is

InstrD = 0x1240\_fffa

PCPlus4D = 0x0040\_00a8

1. At *t = 46.0 ns*, 0x0040\_00ac is written to the PC. Shortly after that, the values for InstrD and PCPlus4E is

InstrD = 0x0319\_4022

PCPlus4E = 0x0040\_00a8

1. At *t = 46.5 ns*, 0x0040\_00b0 is written to the PC. Shortly after that, the values for InstrD, PCBranchM, and ZeroM is

InstrD = 0x0319\_482a

PCBranchM = PCPlus4E + SignImmE\*4

= 0x0040\_00a8 + 0xffff\_fffa\*4

= 0x0040\_00a8 + 0xffff\_ffe8

= 0x0040\_0090

ZeroM = 1

1. At *t = 47.0 ns*, 0x0040\_0090 is written to the PC. Shortly after that, the value for InstrD is

InstrD = 0x0319\_5024

1. At *t = 47.5 ns*, 0x0040\_0094 is written to the PC. Shortly after that, the value for InstrD is

InstrD = 0x8e12\_0000

Ex C) Part I

1. Register $25 on lines 4 and 6.

During the Execute stage of the ADDi instruction in line 6, the Hazard Unit detects that RsE (110012 for $25) matches WriteRegW (also 110012 for $25) and that RegWriteW = 1.

It responds by setting ForwardAE= 01 so that ResultW, which is the result of the ADD instruction on line 4, is passed to the A input of the ALU.

1. Register $24 on lines 5 and 7.

During the Execute stage of the LW instruction in line 7, the Hazard Unit detects that RsE (110002 for $24) matches WriteRegW (also 110002 for $24) and that RegWrite = 1. It responds by setting ForwardAE = 01 so that ResultW, which is the result of the SUB instruction on line 5, is passed to the A input of the ALU.

1. Register $10 on lines 7 and 9.

During the Execute stage of the SW instruction on line 10, the Hazard Unit detects that RtE (010102 for $10) matches WriteRegW (also 010102 for $10) and that RegWrite = 1. It responds by setting ForwardBE = 01 so that ResultW, which is the result of the LW instruction, is passed to the E/M pipeline register WriteDataE.

Part II

The circuit fails to handle the hazard correctly and SW will use some other wrong data value because there are no forwarding muxes in the Memory stage and no logic to stall. In the Execute stage of the SW instruction, the Hazard Unit detects that RtE and WriteRegW match, both being 010102 for $10, and RegWriteM is 1. Therefore, ForwardBE will be 10 and WriteDataE will be ALUOutM. Thus, SW incorrectly uses the address of LW as data.

Ex D)

#Register allocation

# $a0 p

# $a1 past\_last

# $t0 sum

L1:

lw $t1, ($a0) # $t1 = \*p

slt $t2, $t1, $zero # $t2 = (\*p < 0)

movz $t1, $zero, $t2 # if($t2 == 0) $t1 = 0

addiu $a0, $a0, 4 # p++

bne $a0, $a1, L1 # if(p != past\_last) goto L1

addu $t0, $t0, $t1 # sum += $t1