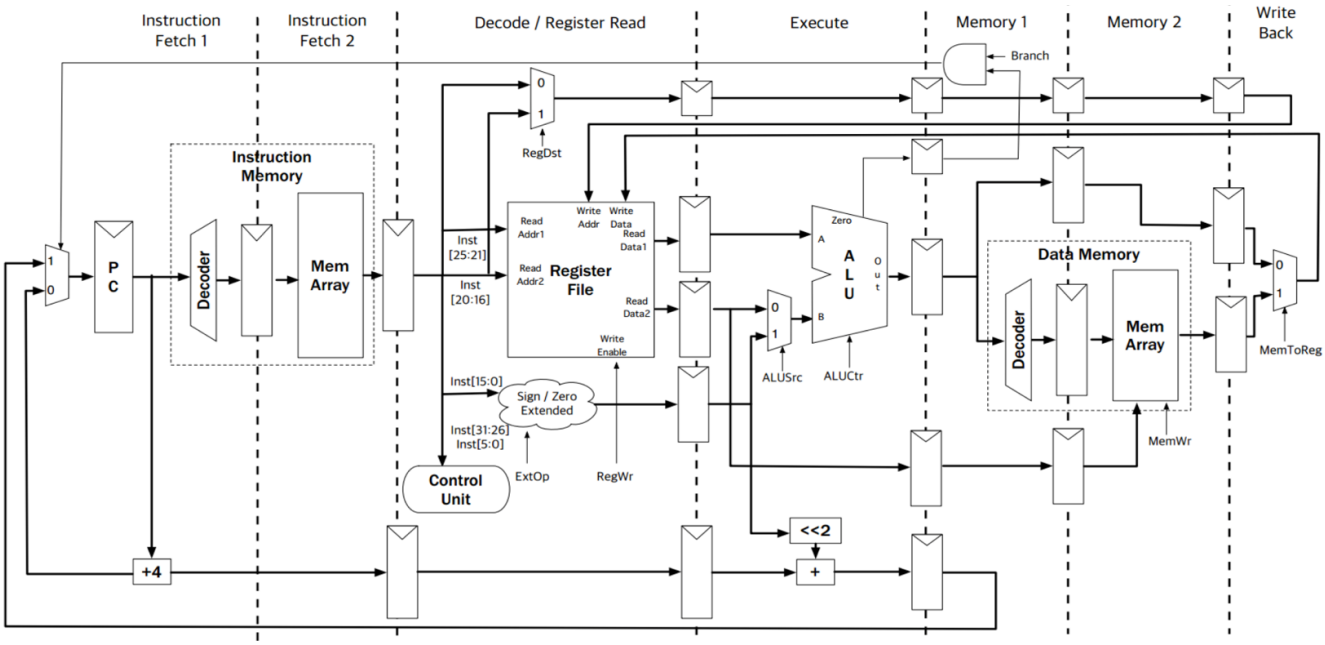
CS 61C Fall 2016
Guerrilla Section 4: MIPS CPU (Datapath & Control)

1) If this exam were a CPU, you'd be halfway through the pipeline (Sp15 Final)

We found that the instruction fetch and memory stages are the critical path of our 5-stage pipelined MIPS CPU. Therefore, we changed the IF and MEM stages to take two cycles while increasing the clock rate. You can assume that the register file is written at the falling edge of the clock.



Assume that no pipelining optimizations have been made, and that branch comparisons are made by the ALU. Here's how our pipeline looks when executing two add instructions:

Clock Cycle #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
add $t0, $t1, $t2	IF1	IF2	ID	EX	MEM1	MEM2	WB	
add $t3, $t0, $t5		IF1	IF2	ID	EX	MEM1	MEM2	WB

a) How many stalls would a data hazard between back-to-back instructions require?

3 stalls

b) How many stalls would be needed after a branch instruction?

4 stalls

c) Suppose the old clock period was 150 ns and the new clock period is now 100ns. Would our processor have a significant speedup executing a large chunk of code...

i) Without any pipelining hazards? Explain your answer in 1-2 sentences.

Yes, due to 1.5x throughput

ii) With 50% of the code containing back-to-back data hazards? Explain your answer in 1- 2 sentences.

Yes, penalty is 450 ns per hazard for a back-to-back data hazard with the old clock period, 300 ns with the new clock period, so our new processor will still have higher throughput.

Problem 2: `movz` and `movnz` (Sp15 MT2)

Consider adding the following instruction to MIPS (disregard any existing definition in the green sheet):

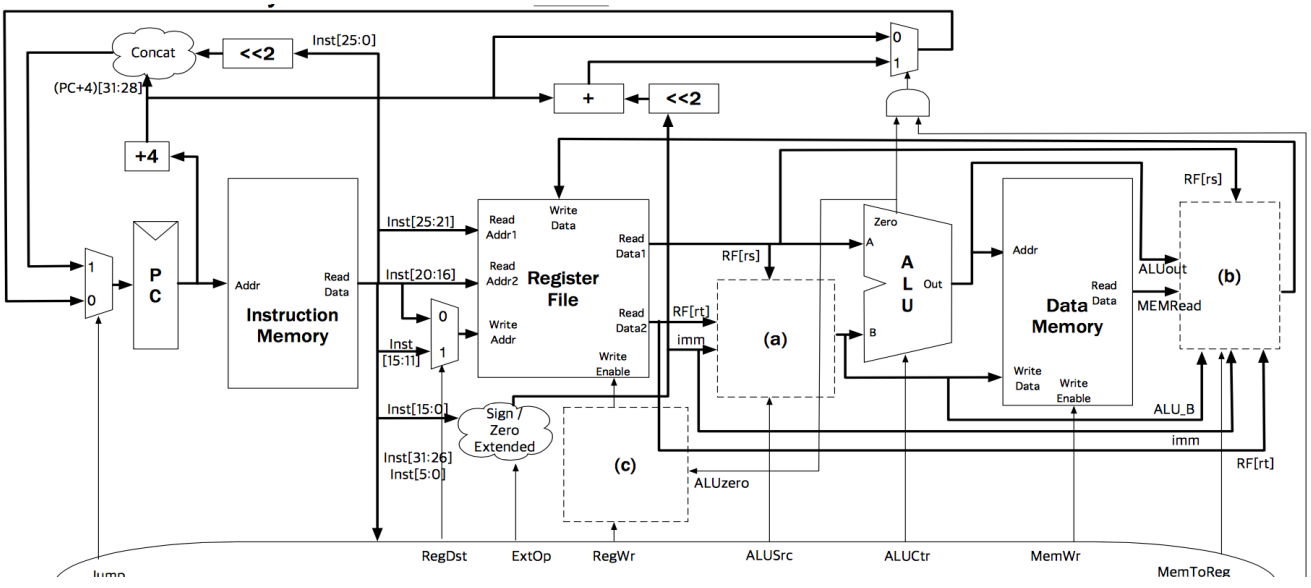
Instruction	Operation
`movz rd, rs, rt`	`if (R[rs] == 0) R[rd]<-R[rt]`
`movnz rd, rs, rt`	`if (R[rs] != 0) R[rd]<-R[rt]`

a) Translate the following C code using `movz` and `movnz`. Do not use branches.

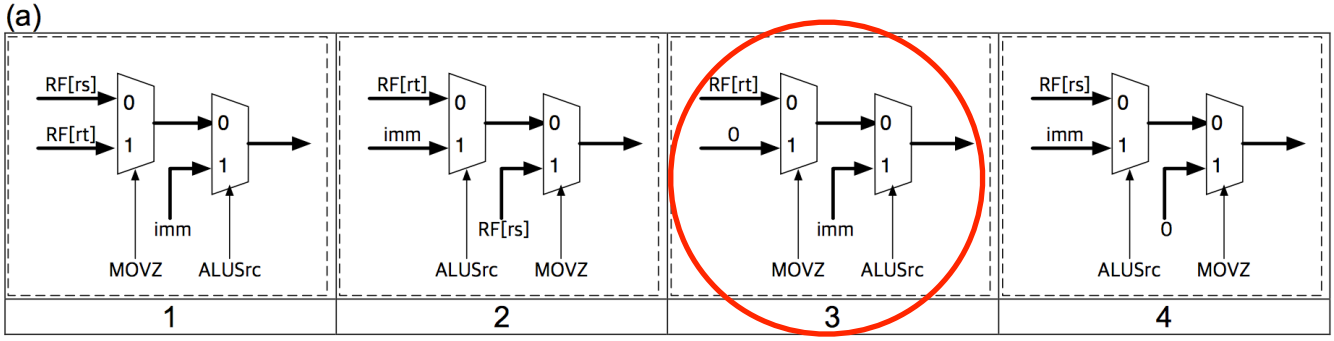
C code	MIPS

`// a -> $s0, b-> $s1, c-> $s2`
`int a = b < c ? b : c;`
	`slt $t0, $s1, $s2`
	`movnz $s0, $t0, $s1`
	`movz $s0, $t0, $s2`

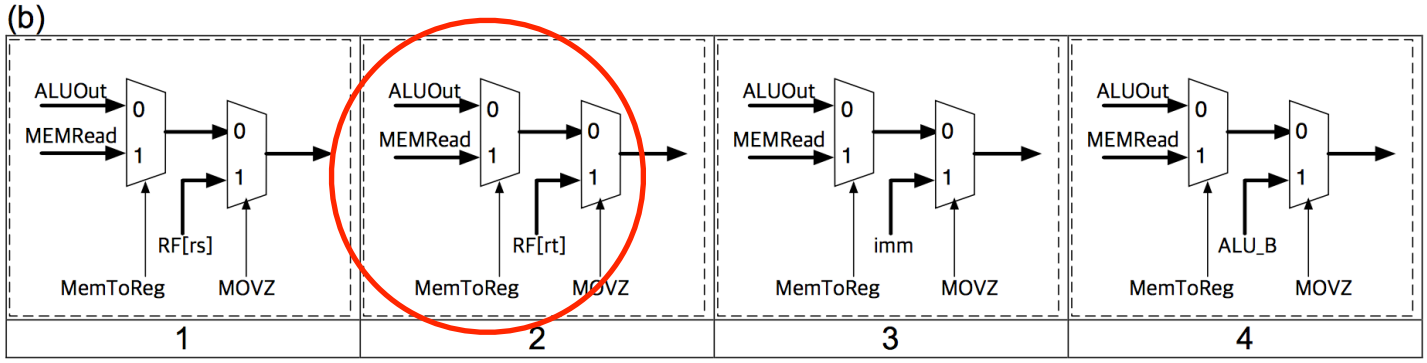
b) Implement **`movz`** (but not `movnz`) in the datapath. Choose the correct implementation for (a), (b), and (c). Note that you do not need to use all the signals provided to each box, and the control signal `MOVZ` is 1 if and only if the instruction is **`movz`**.



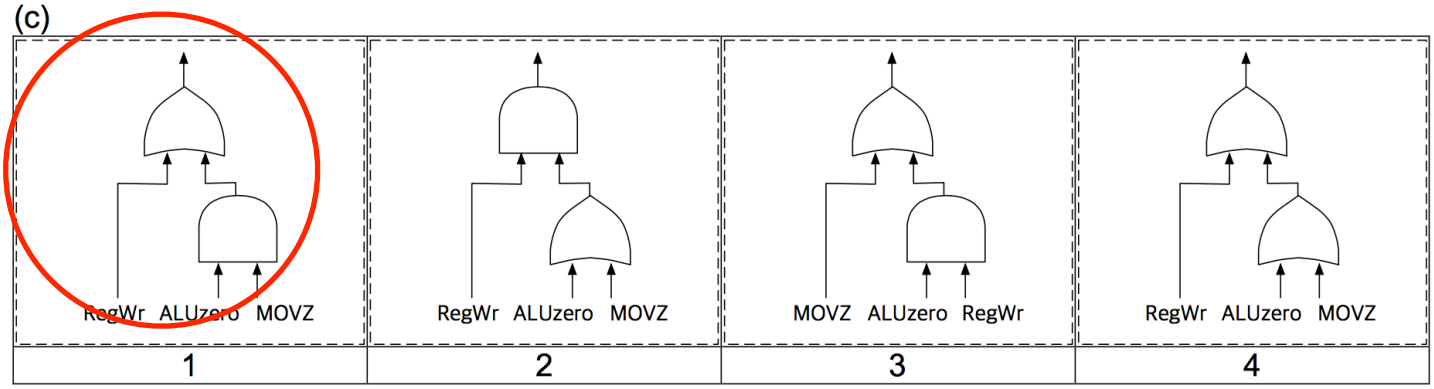
(a)



(b)



(c)



c) Generate the control signals for **`movz`**. The values should be 0, 1, or X (don't care) terms. You must use don't care terms where possible.

MOVZ	RegDst	ExpOp	RegWr	ALUSrc	ALUCtr	MEMWr	MemToReg	Jump	Branch
1	1	X	0	0	0001, 0010, or 0110	0	X	0	0

This table shows the ALUCtr values for each operation of the ALU:

Operation	AND	OR	ADD	SUB	SLT	NOR
ALUCtr	0000	0001	0010	0110	0111	1100

Problem 3) Watch Your (Time) Step! (Su12 Final)

Consider the following difference equation (i.e. differential equation that is discretized in time):

```
x[n+1] - x[n] = -alpha*x[n]
x[n+1] = (1-alpha)*x[n]
```

Here we restrict ourselves to integer values of alpha and use the following integration function:

```
# $a0 -> addr of array to store x[i] (assume x[0] is already set)
# $a1 -> length of array/integration
# $a2 -> alpha
```

IF	ID	EX	MEM	WB
200 ps	100 ps	400 ps	200 ps	100 ps

```
     integrate:
1            beq   $a1,$0,exit
2            sub   $t0,$0,$a2          # $t0 = -alpha
3            addi  $t0,$t0,1      # $t0 = 1-alpha
4            lw    $t1,0($a0)          #$t1=x[n]
5            mult  $t0,$t1
6            mflo  $t1            # $t1 = (1-alpha)*x[n]
7            sw    $t1,4($a0)          # x[n+1] = (1-alpha)*x[n]
8            addiu $a0,$a0,4
9            addiu $a1,$a1,-1
10           j     integrate
11    exit:  jr    $ra
```

We are using a 5-stage MIPS pipelined datapath with separate I$ and D$ that can read and write to registers in a single cycle. Assume no other optimizations (no forwarding, etc.). The default behavior is to stall when necessary. Multiplication and branch checking are done during EX and the HI and LO registers are read during ID and written during WB.

a) As a reminder, Tc stands for "time between completions of instructions." Given the following datapath stage times, what is the ratio Tc,single-cycle/Tc,pipelined?

1000/400=5/2

b) Count the number of the different types of potential hazards found in the code above:

Structural:_____0_____ Data:_____6_____ Control:_____3_____

None @ 2-3, 3-5, 4-5, 5-6, 6-7, 9-1 @ 1, 10, 11

For the following questions, examine a SINGLE ITERATION of the loop (do not consider the jr).

c) With no optimizations, how many clock cycles does our 5-stage pipelined datapath take in one loop iteration (end your count exactly on completion of j)?

24 = 10 instructions + 4 for draining pipeline + 2*(# data hazards – 2) + 2 for beq control hazard. Beq hazard is 2 stalls because branch comparison done in EX stage.

data hazards minus 2 because 3-5 and 9-1 hazards taken care of by other stalls.

d) How many clock cycles LESS would be taken if we had FORWARDING?

7= 2*(# of data hazards – 1) – 1 because of load hazard. Again, you can ignore the 3-5 hazard here.

e) Assuming that we introduce both FORWARDING and DELAY SLOTS, describe independent changes to integrate that will reduce the total clock cycles taken. Changes include replacing or moving instructions. You may not need all spaces given.

Instr	**Change**
___3____	Move after 4 (or move instr 4 before instr 3)_____
___8/9__	Move after 10_____

OR

____9___	Move after 4_____
____8___	Move after 10_____

F4) Please Pass Professor's Pretty Pipeline-pourri Problem… (30 pts, 42 min)

Instruction	Cycle 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
(1) add $s0 $s1 $t0	F	Dt0s1	E	M	Ws0														
(2) addi $t0 $t0 4		F	Dt0	E	M	Wt0													
(3) sw $s0 0($t1)			F	F	Ds0t1	E	M	WM											
(4) and $t1 $t1 $t2					F	Dt1t2	E	M	Wt1										
(5) lw $t2 4($t1)						F	F	F	Dt1	E	M	Wt2							
(6) bne $t2 $t1 -6 #goto 1									F	F	F	Dt1t2	E	M	W				
(7) sub $s0 $s0 $t2																			

↓ Instr / Cycle # →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
sll $v1 $v1 8	F	D	E	M	W													
xor $v0 $a1 $a2		F	D	E	M	W												
addu $a1 $s3 $t1			F	D	E	M	W											
andi $v0 $v0 1				F	D	D	E	M	W									
addu $t0 $a0 $t1					F	F	D	E	M	W								
lw $s0 0($t0)							F	D	D	D	E	M	W					
bne $s0 $0 End								F	F	F	D	D	D	E	M	W		
j Taketwo											F	F	F	D	E	M	W	