CS M151B: Homework 5

Jonathan Woong

804205763

Spring 2017

Discussion 1A

Monday 8th May, 2017

Problem 1. Consider the pipelined MIPS implementation shown in Figure 4.51 (page 304). The value of the `PCSrc` signal is computed in the MEM stage. Could the `PCSrc` signal be computed in the EX stage instead? If so, what would be the advantages of making this change? Also, what would be the disadvantages of making this change?

Yes, the `PCSrc` signal can be computed in the EX stage. This will give the advantage of reducing branch stall to one cycle. The disadvantage is that additional forwarding and hazard detection is required, and we must also implement the ability to flush instructions in the IF stage.

Problem 2. Consider the pipelined MIPS implementation shown in Figure 4.51 (page 304). Consider each of the following two instructions separately. For each of these instructions (separately), what is the value of the `PCSrc` signal when the instruction is in the MEM pipeline stage? Explain your answer.

a. `add $5,$15,$24`

The value of the `PCSrc` signal at the MEM stage is 0 because the Branch signal for R-format instructions is set to 0, meaning the the next sequential PC address will be PC + 4.

b. `beq $1,$1,200`

The value of the `PCSrc` signal at the MEM stage is 1 because the Branch signal is always asserted for `beq` instructions, and the Zero signal from the ALU will be 1 due to the fact that register $1 is always equal to itself. This will set the PC to be the result of the ALU address computation.

Problem 3. Consider the following code on the pipelined datapath of Figure 4.60 on page 316:

```
add     $2, $3, $1
sub     $4, $3, $5
add     $5, $3, $7
add     $7, $6, $1
add     $8, $2, $6
```

During the fifth cycle, which registers of the register file are read and which register of the register file is written?

Instruction	CC 1	CC 2	CC 3	CC 4	CC 5
add	IF	ID	EX	MEM	WB
sub		IF	ID	EX	MEM
add			IF	ID	EX
add				IF	ID
add					IF

$\boxed{Read : \$6, \$1}$

$\boxed{Written : \$2}$

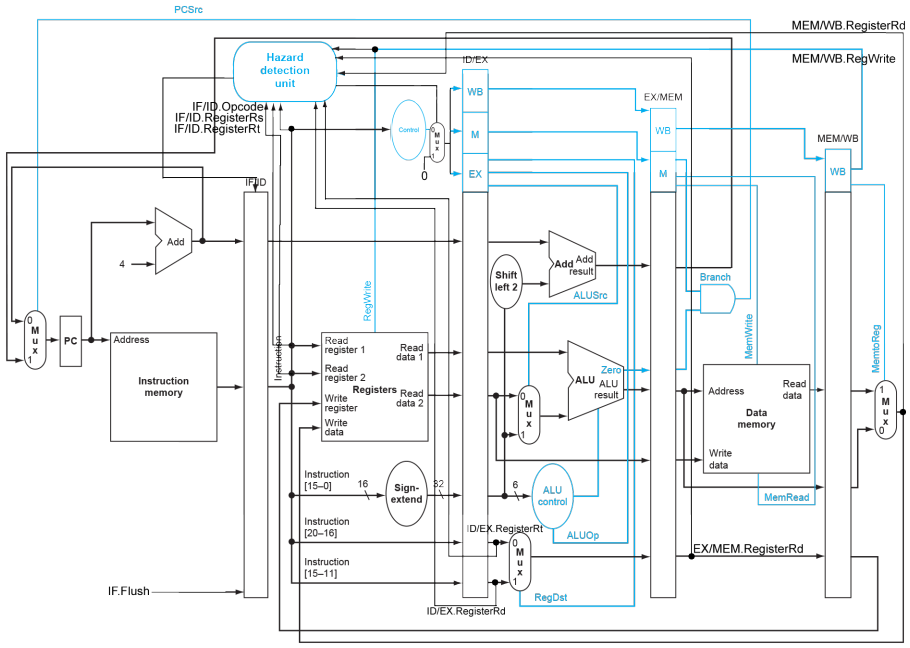
Problem 4. Consider the pipelined MIPS implementation shown in Figure 4.51 (page 304). Figure 4.51 includes support for the `beq` instruction. However, Figure 4.51 ignores the existence of control dependencies. Specifically, Figure 4.51 does not include the required control hazard detection and stall implementation. Your task is to add control hazard detection and stall implementation to Figure 4.51. Do not use any branch prediction mechanism. Do not modify the basic datapath (you cannot use the modified datapath shown in Figure 4.62). As a starting point, consider the hazard detection and stall implementation shown in Figure 4.60 (page 316), with the hazard detection unit logic described on pages 313-314. Modify this circuit as necessary and add the possibly modified circuit to Figure 4.51. Using the same style as in page 314, specify the logic of the hazard detection unit.

Hazard detection unit:

```
Case 1:  if (IF/ID.Opcode=000100 and
   (ID/EX.RegisterRd = IF/ID.RegisterRs or ID/EX.RegisterRd = IF/ID.RegsiterRt)
or (ID/EX.RegisterRt = IF/ID.RegisterRs or ID/EX.RegisterRt = IF/ID.RegsiterRt))
   stall 3 cycles
Case 2:  if (IF/ID.Opcode=000100 and
   (EX/MEM.RegisterRd = IF/ID.RegisterRs or EX/MEM.RegisterRd = IF/ID.RegsiterRt))
   stall 2 cycles
Case 3:  if (IF/ID.Opcode=000100 and
   MEM/WB.RegWrite=1 and (MEM/WB.RegisterRd = IF/ID.RegisterRs or MEM/WB.RegisterRd
= IF/ID.RegsiterRt))
   stall 1 cycle
Case 4:  if (MEM/WB.Branch and MEM/WB.Zero)
   flush IF/ID, ID/EX, EX/MEM
```

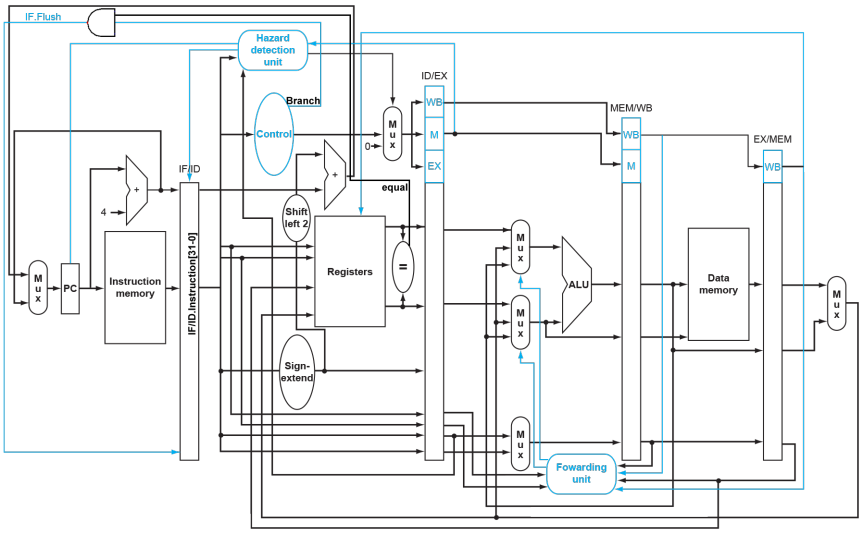
Case 1,2,3: The first line identifies the `beq` instruction, since it is the only one to use the opcode 000100. The second line identifies that either the Rs or Rt of `beq` intends to be written by a previous instruction (either `lw` or an R-type instruction). The stall is accomplished by the MUX that is controlled by the Hazard Detection Unit signal, which sets all control signals to 0. After stalling the correct number of cycles, the correct values for the Rs and Rt registers are written by the previous instruction.

Case 4: The line identifies the condition that the branch must be taken. In this case, we flush the IF/ID, ID/EX, and EX/MEM register values to ensure that instructions that execute after branching are not affected by instructions that would have executed if the branch was not taken. This can be accomplished by setting all control signals to 0 (using the MUX controlled by Hazard Detection Unit signal) and asserting IF.Flush for three cycles.

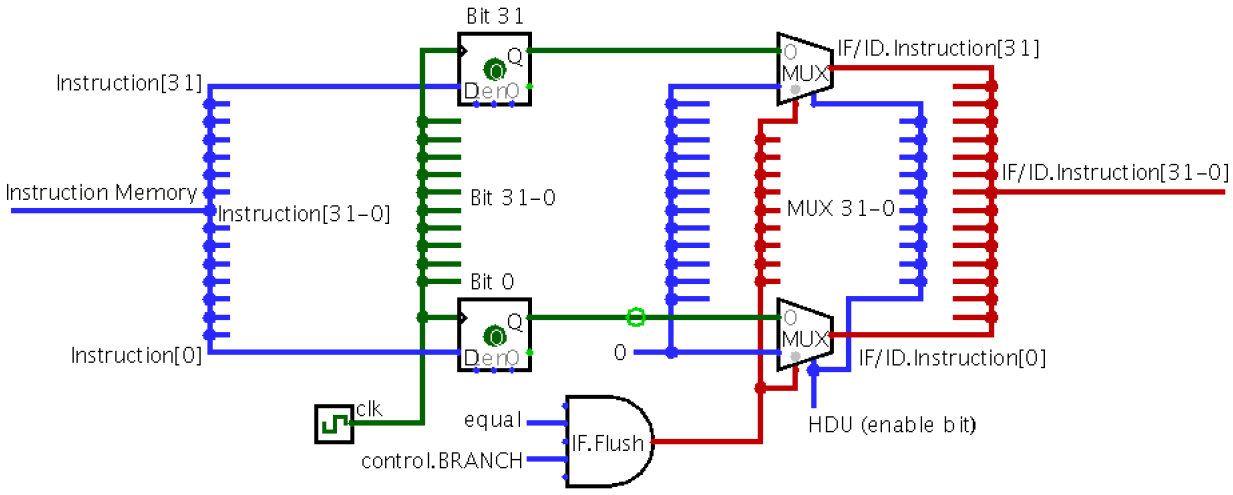


Problem 5. Figure 4.65 (page 325) shows the pipelined MIPS implementation with the details of how stalls due to the `lw` and `beq` instructions are implemented. As discussed in class, the book doesn't fully explain how the stalls for `beq` are implemented. Assume that the implementation is as shown on slide 7.54 in the class notes. As shown in Figure 4.65, the IF/ID register is a selectively-modified register. Assume that this register is implemented with flip-flops, as shown on the right side of slide 5.19 in the class notes.

A) On a copy of Figure 4.65, show the digital circuit (gate-level implementation) that generates the IF.Flush signal. Be sure that it is clear exactly how it is connected to the rest of the implementation. If it is connected to a module that generates multiple outputs, clearly identify the specific output to which your circuit is connected.



B) Given the assumption above regarding how the IF/ID register is implemented, show the digital circuit (gate-level implementation) that uses the IF.Flush signal to implement the desired functionality. Note that this part should be shown separately from the copy of Figure 4.65.



Problem 6. Refer to the following fragment of MIPS code:

```
sw      r16,12(r6)
lw      r16,8(r6)
beq     r5,r4,Label     # Assume r5!=r4
add     r5,r1,r4
slt     r5,r15,r4
```

Assume that individual pipeline stages have the following latencies:

IF	ID	EX	MEM	WB
200 ps	120 ps	150 ps	190 ps	100 ps

Assume that all branches are perfectly predicted (this eliminates all control hazards) and that no delay slots are used. If we change load/store instructions to use a register (without an offset) as the address, these instructions no longer need to use the ALU. As a result, MEM and EX stages can be overlapped and the pipeline has only 4 stages. Change this code to accomodate this changed ISA. Assuming this change does not affect clock cycle time, what speedup is achieved in this instruction sequence?

Before change:

Instruction	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9	CC 10	CC 11	Total
sw	IF	ID	EX	MEM	WB							
lw		IF	ID	EX	MEM	WB						
beq			IF	ID	EX	MEM	WB					
add						IF	ID	EX	MEM	WB		
slt							IF	ID	EX	MEM	WB	
time	200	200	200	190	190	200	200	150	190	190	100	2010

After change:

Instruction	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9	Total
sw	IF	ID	EX/MEM	WB						
lw		IF	ID	EX/MEM	WB					
beq			IF	ID	EX/MEM	WB				
add					IF	ID	EX/MEM	WB		
slt						IF	ID	EX/MEM	WB	
time	200	200	200	190	200	200	190	190	100	1670

```
sw      r16,r6
lw      r16,r6
beq     r5,r4,Label     # Assume r5!=r4
add     r5,r1,r4
slt     r5,r15,r4
```

Total speedup $= 2010 - 1670 = \boxed{340\ ps} = 1 - \frac{1670}{2010} = 0.1691 = \boxed{16.91\%}$

Problem 7. Assuming stall-on-branch and no delay slots, what speedup is achieved on this code if branch outcomes are determined in the ID stage, relative to the execution where branch outcomes are determined in the EX stage?

After change:

Instruction	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9	CC 10	Total
sw	IF	ID	EX	MEM	WB						
lw		IF	ID	EX	MEM	WB					
beq			IF	ID	EX	MEM	WB				
add					IF	ID	EX	MEM	WB		
slt						IF	ID	EX	MEM	WB	
time	200	200	200	190	200	200	150	190	190	100	1820

Total speedup $= 2010 - 1820 = \boxed{190\ ps} = 1 - \frac{1820}{2010} = 0.0945 = \boxed{9.45\%}$

Problem 8. An ISA that is very similar to the MIPS ISA supports 95 opcodes and 64 registers. Instructions are 32 bits wide. There is a class of instructions in this ISA that specify two register arguments and one signed (2's complement) immediate. What is the maximum possible range of values of this immediate?

opcode	Rs	Rt	Immediate
7 bits	6 bits	6 bits	13 bits

Range $= (-2^{12}, 2^{12} - 1) = (-4096, 4095)$

Problem 9. Implement the following C code in MIPS using indexed addressing and in PowerPC using update addressing:

```
for (i=0; i != 10; i++) { a[2i] = b[i] + i; }
```

Assume that the base address at `a[]` is stored in `$a0 ($4)`, and the base address of `b[]` is stored in `$a1 ($5)`.

```
MIPS:   addi    $1,$0,0         # $1 = i ← 0
        addi    $6,$1,40        # $6 = $a2 ← 40
loop:   beq     $1,$6,end       # if (i == 10): exit loop
        lw      $9,$5+$1        # $9 = $t1 ← M[$a1 + i] = b[i]
        add     $9,$9,$1        # $t1 ← b[i] + i
        sll     $7,$1,1         # $7 = $a3 ← 2i
        sw      $9,$4+$7        # M[$a0 + 2i] ← $t1 ⇔ a[2i] ← b[i] + i
        addi    $1,$1,4         # i ← i + 1
        j       loop            # repeat loop
end:

PowerPC:addi    $1,$0,0         # $1 = i ← 0
        addi    $6,$1,40        # $6 = $a2 ← 40
        addi    $8,$5,$1        # $8 = $t0 ← $a1 + i
        lw      $8,0($8)        # $t0 ← M[$a1 + i] = b[i]
loop:   beq     $1,$6,end       # if (i == 10): exit loop
        add     $9,$8,$1        # $t1 ← b[i] + i
        sll     $10,$1,1        # $10 = $t2 ← 2i
        add     $7,$4,$10       # $7 = $a3 ← $a0 + 2i
        sw      $9,0($7)        # M[$a0 + 2i] = a[2i] ← b[i] + i
        lwu     $8,4($1)        # $8 ← M[$8 + 4]; i ← i + 4
        j       loop
end:
```