CprE 381 Homework 7

[Note: The first couple of questions are intended to help you familiarize yourself with pipelining. The third question will help you understand data hazards and how to mitigate them. The final question will help you prepare for the exam by looking through lecture notes and online quizzes and then thinking which questions I might ask.]

1. Pipeline Simulation

   For each of the modules from Figure 4.51 (page 304 of P&H) that are listed in the table below, specify what the inputs and outputs are in each cycle for the following code. You do not need to specify values for those modules, inputs, or outputs not listed in the table. Manually simulate (i.e., fill out the table) until the **`sw`** has completed (i.e., left the write-back stage). *[Hint: the table already has the first couple of cycles filled out. If a value depends on an instruction before or after the code below, report it as X.]*

Cycle	Instruction Memory		Register File				ALU				MemtoReg MUX			PCSrc MUX		
	Addr	Instr	Read reg 1	Read data 1	Write reg	Write data	A	B	Op (e.g, add, sub)	ALU result	1	0	s	1	0	s
1	0x00000010	addi …	X	X	X	X	X	X	X	X	X	X	X	X	0x00000014	0
2	0x00000014	lui …	0x00	0x00000000	X	X	X	X	X	X	X	X	X	X	0x00000018	0
3	0x00000018	xor…	0x00	0x00000000	X	X	0x0	0x0000003F	addi	0x0000003F	X	X	X	X	0x0000001c	0

[Pipelined MIPS – Simulation Table]

```
# Assume that $a0 = 3, $a1 = 1024, $a2 = 1023,
$a3 = -1 # at the start of your manual
simulation.
# Assume that lui is supported by the lui
operation in # the ALU and that the value
shifted for lui is the B # input of the ALU
(note that this is likely different # than your
project implementation and that's OK).
# The following instructions start at address
0x00000010:
addi $t7, $zero, 63
lui  $s7, 0x1010
xor  $t5, $a2, $a3
sub  $t6, $a0, $a1
sll  $zero, $zero, 0
ori  $s7, $s7,
0x0040 beq  $t7,
$a3, Exit sll
$zero, $zero, 0 sll
```

```
$zero, $zero, 0 sll
$zero, $zero, 0 sw
$t7, 0($s7) ...
Exit: # This label resolves to address 0x00000100.
```

Cycle	Instruction Memory		Register File				ALU				MemtoReg MUX			PCSrc MUX		
	Addr	Instr	Read reg 1	Read data 1	Write reg	Write data	A	B	Op (e.g, add, sub)	ALU result	1	0	s	1	0	s
1	0x00000010	addi …	X	X	X	X	X	X	X	X	X	X	X	X	0x00000014	0
2	0x00000014	lui …	0x00	0x00000000	X	X	X	X	X	X	X	X	X	X	0x00000018	0
3	0x00000018	xor…	0x00	0x00000000	X	X	0x00000000	0x0000003F	add	0x0000003F	X	X	X	X	0x0000001c	0
4	0x0000001c	sub…	0x06	0x000003FF	X	X	0X00000000	0x00001010	lui	0x10100000	X	X	X	0x00000110	0x00000020	0
5	0x00000020	sll…	0x04	0x00000003	0x0F	1	0x000003FF	0xFFFFFFFF	xor	0xFFFFFC00	X	0x0000003F	0	0x00004058	0x00000024	0
6	0x00000024	ori…	0x00	0x00000000	0x17	1	0x00000003	0x00000400	sub	0xFFFFFC03	X	0x10100000	0	0x0001A0B4	0x00000028	0
7	0x00000028	beq…	0x17	0x10100000	0x0D	1	0x00000000	0x00000000	Sll	0x00000000	X	0xFFFFFC00	0	0x0001C0a8	0x0000002c	0
8	0x0000002c	sll…	0x0F	0x0000003F	0x0E	1	0x10100000	0x00000040	ori	0x10100040	X	0xFFFFFC03	0	0x00000024	0x00000030	0
9	0x00000030	sll…	0x00	0x00000000	0x00	1	0x0000003F	0xFFFFFFFF	sub	0x00000040	X	0x00000000	0	0x00000128	0x00000034	0

2. Data Dependencies, Hazard Detection, and Hazard Avoidance

*[**WARNING**: The datapath described and used here is intentionally different from your project's datapath. The goal is for you to be able to understand the relationship between datapath design and hazards.]*

a. Identify all the read after write data dependencies in the following code. Use the format (reading instruction address, register read/written, writing instruction addr).

```
# The following code starts at address 0x0000010
addiu $t0, $zero, 0     # clear i ($t0) j
cond loop:
addiu $t1, $a0, $t0
```

```
lb     $t1, 0($t1)        # load value to histogram
addu   $t1, $a3, $t1      # calculate histogram bin
```

10	0x00000034	sll	0x00	0x00000000	0x17	1	0x00000000	0x00000000	sll	0x00000000	X	0x10100040	0	0x00000100	0x00000038	0
11	0x00000038	sw	0x00	0x00000000	X	0	0x00000000	0x00000000	sll	0x00000000	X	0x00000040	0	0x00000030	0x0000003c	0
12	0x0000003c	X	0x17	0x10100040	0x00	1	0x00000000	0x00000000	sll	0x00000000	X	0x00000000	0	0x00000034	0x00000040	0
13	0x00000040	X	X	X	0x00	1	0x10100040	0x00000000	add	0x10100040	X	0x00000000	0	0x00000038	0x00000044	0
14	0x00000044	X	X	X	0x00	1	X	X	X	X	X	0x00000000	0	0x0000003C	0x00000048	0
15	0x00000048	X	X	X	0x00	0	X	X	X	X	X	0x10100040	0	X	0x0000004C	X

[Pipelined MIPS – Simulation Table]

Note that the yellow highlighted elements are known, even if they are unused.

```
lb     $t2, 0($t1)        # load bin value
addiu  $t2, $t2, 1        # increment bin value sb
$t2, 0($t1)        # store bin value
addiu  $t0, $t0, 1        # increment i cond:
sltu   $t1, $t0, $a1      # loop condition check (N in $a1)
bne    $t1, $zero, loop   check:
addu   $t1, $a3, $a2      # check bin value at input ($a2
   location
lb     $t1, 0($t1)
sltiu  $t1, $t1, 100 bne
$t1, $zero,Exit
jal    detection          # jump to function (not shown)
Exit:
```

1: (0x00000018,$t0,0x00000010)
2: (0x00000018,$t0,0x00000030)
3: (0x0000001c,$t1,0x00000018)
4: (0x00000020,$t1,0x0000001c)
5: (0x00000024,$t1,0x00000020)
6: (0x00000028,$t2,0x00000024)
7: (0x0000002c,$t1,0x00000020)
8: (0x0000002c,$t2,0x00000028)
9: (0x00000030,$t0,0x00000010)*
10: (0x00000030,$t0,0x00000030)*
11: (0x00000034,$t0,0x00000030)
12: (0x00000034,$t0,0x00000010)
13: (0x00000038,$t1,0x00000034)
14: (0x00000040,$t1,0x0000003c)
15: (0x00000044,$t1,0x00000040)
16: (0x00000048,$t0,0x00000044)
*Dependency resolved without additional work needed.

b. The 5-stage MIPS pipeline described in lecture (and used for your term project) is not the only possible pipeline design. Consider the six stages described in the table below (this ordering is actually similar to the original MIPS pipeline described in a 1981 research paper). Using your answer to part 3a., please identify all possible data hazards occurring in the eight instructions following the loop label assuming the register file reads the new

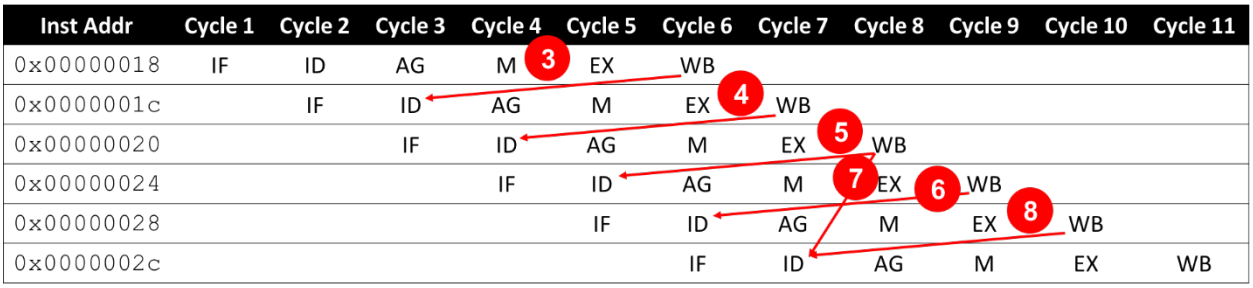
value being written into a specific register in a given cycle. Justify your answer with a pipeline diagram that shows possible backwards dependencies.

[Note: Only compute any data hazards that occur from the instruction at address 0x00000018 through address 0x00000034.]

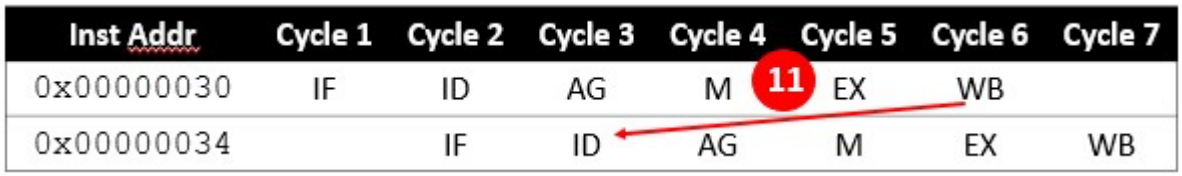
Stage Order	Stage Abbreviation	Stage Function
1	IF (Instruction Fetch)	Load Instruction from M[PC]; increment PC
2	ID (Instruction Decode)	Generate control signals for instruction; read register operands; branch and jump Next PCs determined
3	AG (Address Generation)	Calculate addresses for memory operations
4	M (Memory Access)	Access memory
5	EX (Execution)	Perform any ALU operations
6	WB (Writeback)	Write results (ALU or memory operations) back to registers

We can easily eliminate any dependencies that span more than six dynamic instructions since those instructions are never in the pipeline together (dependencies 9 and 10 above).

The following pipeline diagram shows that dependencies 3 to 8 cause data hazards.



The following pipeline diagram shows that dependency 11 causes a data hazard, while dependency 2 does not (since the register file reads the new value).



c. Pipeline Hazard Avoidance (Software)
Insert the minimum number of NOP instructions in order to allow the above program to run correctly on the 6-stage pipeline described in 3b without any hardware hazard detection or forwarding logic. Note: there are both control and data hazards that must be avoided.

```
loop:
addiu $t1, $a0, $t0
sll   $zero, $zero, 0   # Avoid data hazard 3.
sll   $zero, $zero, 0   # Avoid data hazard 3.
sll   $zero, $zero, 0   # Avoid data hazard 3.
lb    $t1, 0($t1)       # load value to histogram
sll   $zero, $zero, 0   # Avoid data hazard 4.
sll   $zero, $zero, 0   # Avoid data hazard 4.
sll   $zero, $zero, 0   # Avoid data hazard 4.
addu  $t1, $a3, $t1     # calculate histogram bin
sll   $zero, $zero, 0   # Avoid data hazard 5.
sll   $zero, $zero, 0   # Avoid data hazard 5.
sll   $zero, $zero, 0   # Avoid data hazard 5.
lb    $t2, 0($t1)       # load bin value
sll   $zero, $zero, 0   # Avoid data hazard 6.
sll   $zero, $zero, 0   # Avoid data hazard 6.
sll   $zero, $zero, 0   # Avoid data hazard 6.
addiu $t2, $t2, 1       # increment bin value
# Data hazard 7 avoided by above (5 & 6)NOPs.
sll   $zero, $zero, 0   # Avoid data hazard 8.
sll   $zero, $zero, 0   # Avoid data hazard 8.
sll   $zero, $zero, 0   # Avoid data hazard 8.
sb    $t2, 0($t1)       # store bin value
addiu $t0, $t0, 1       # increment i
sll   $zero, $zero, 0   # Avoid data hazard 11.
cond:
sll   $zero, $zero, 0   # Avoid data hazard 11.
sll   $zero, $zero, 0   # Avoid data hazard 11.
sltu  $t1, $t0, $a1     # loop condition check (N in $a1)
```

d. Pipeline Hazard Avoidance (Forwarding Logic)

Now we want to implement forwarding logic in HW to improve the CPI and # of instructions executed in our implementation. Specify the forwarding condition for the address generation module's input A (assume this is the base register input for the base offset address mode). Use the same format as we did in lecture (see Lec10.1) where the ForwardA select signal is 00 for the ID/AG pipeline register, 01 for the M/EX pipeline register, and 10 for the EX/WB pipeline register. Demonstrate that your forwarding condition holds true by drawing and annotating a pipeline diagram for the execution of the first six instructions after the `loop` label (**`addiu`**, **`lb`**, **`addu`**, **`lb`**, **`addiu`**, **`sb`**). -- Note that there is no point to forward from the AG/M pipeline register since no result will have been generated by then (no mem load or arithmetic execution). if (M/EX.RegWrite and (M/EX.RegisterRd \= 0) and (M/EX.RegisterRd = ID/AG.RegisterRs)))

-- Note that we assume that the destination register has been selected in AG so M/EX.RegisterRd is the final result register.

ForwardA = 01

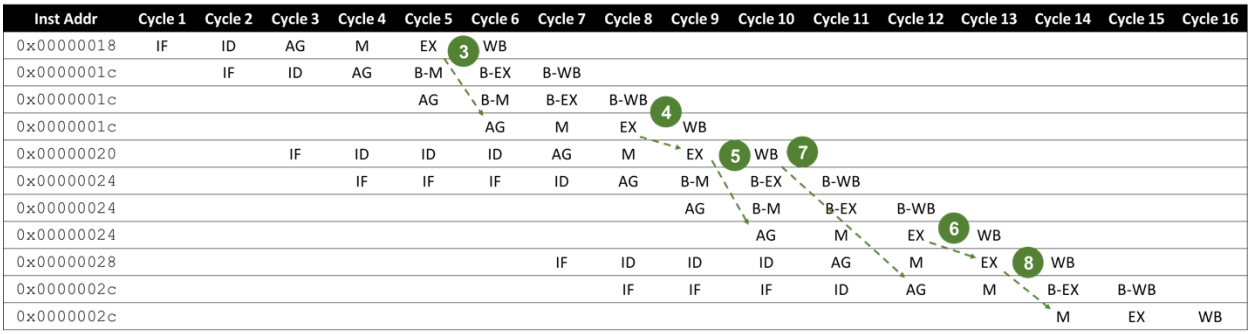
elseif (EX/WB.RegWrite and (EX/WB.RegisterRd \= 0) and (EX/WB.RegisterRd = ID/AG.RegisterRs))

-- Note that the elseif implies no forwarding from the M/EX pipeline register.

ForwardA = 10

else

ForwardA = 00



All that our forwarding needs to worry about is the AG stage, so dependencies 3, 5, and 7 (others are shown just to work out stalling). Note that B-<Stage ID> indicates that a bubble has been inserted into the pipeline. A repeated occupancy of an instruction in a stage for multiple cycles is a stall.

We see that the address generation stage of the 0x0000001c lb instruction requires a two cycle stall for the previous value to be calculated (a disadvantage of this pipeline organization). Then the first condition of the forwarding logic evaluates to true and we forward from the EX/WB pipeline register (10).

We next see that the address generation stage of the 0x00000024 lb instruction requires another two cycle stall for the previous value to be calculated. Afterwards, the first condition of the forwarding logic evaluates to true and we forward from the EX/WB pipeline register (10).

Finally, we see that the address generation stage of the 0x0000002c sb instruction uses a value that has been written back to the register file two cycles before. In this case, we see that neither of the first two conditions are true, so the forwarding logic evaluates to the ID/AG pipeline register (00) which contains the value read from the register file.

e. Why such a pipeline?

What would be a possible advantage of having the above six-stage pipeline (think about this pipeline implementing a more CISC-like ISA)? What would your average CPI be? (answer qualitatively rather than quantitatively)

While there are several disadvantages of the above pipeline in terms of being able to forward, etc., it would allow memory load + computation instructions to execute on the pipeline which would lower the number of instructions in the program.

In terms of CPI, there are several observations one could make, but one obvious one is that, compared to the 5-stage pipeline, the 6-stage pipeline (assuming fully-forwarded in both cases) has a worse (i.e., higher) CPI due to the execute stage being so late in the pipeline impacting the calculation of addresses for memory operations.

3. Exam Question

Develop your own exam question (roughly 10-15 points) from MIPS arithmetic, single- cycle processor design, performance analysis, pipelining, or data hazards. Your question shouldn't simply ask students to recall information, but should ask for an application of a concept or require understanding of a concept or need analysis of a processor/application. You must include a correct and complete solution to your question. This question should be your own work and not copied from a book or an old exam. ***Post your question and solution to the Exam2 channel for others to use.***

Will post some of the best questions for exam review.