CS 250 Spring 2017 Homework 07 SOLUTION
Due 11:58pm Friday, March 10, 2017
Submit your typewritten file in PDF format to Blackboard.

1. Text exercise 5.8. Assume that after the last instruction of the given code snippet is fetched, that the instructions fetched are denoted "???". Note that the pipeline of Figure 5.5 cannot read an operand that is written into the register unit on clock cycle N until clock cycle N+1. The comment to line 3 of the assembly code should read "# put 20 in register 9".
Answer:
Many forms of answer are appropriate for a problem of this type. First, mimicking the style of Figure 5.5 we could have

CC	Stage 1 IF	Stage 2 ID	Stage 3 EX	Stage 4 MEM	Stage 5 WB
1	Loadi r7,10	?	?	?	?
2	Loadi r8,15	Loadi r7,10	?	?	?
3	Loadi r9, 20	Loadi r8,15	Loadi r7,10	?	?
4	Addrr r10,r7,r8	Loadi r9, 20	Loadi r8,15	Loadi r7,10	?
5	Movr r12,r9	(Addrr r10,r7,r8)	Loadi r9, 20	Loadi r8,15	Loadi r7,10
6	(Movr r12,r9)	(Addrr r10,r7,r8)	NOP	Loadi r9, 20	Loadi r8,15
7	(Movr r12,r9)	Addrr r10,r7,r8	NOP	NOP	Loadi r9, 20
8	Movr r11,r7	Movr r12,r9	Addrr r10,r7,r8	NOP	NOP
9	Addri r14,r11,27	Movr r11,r7	Movr r12,r9	Addrr r10,r7,r8	NOP
10	Addrr r13,r12,r11	(Addri r14,r11,27)	Movr r11,r7	Movr r12,r9	Addrr r10,r7,r8
11	(Addrr r13,r12,r11)	(Addri r14,r11,27)	NOP	Movr r11,r7	Movr r12,r9
12	(Addrr r13,r12,r11)	(Addri r14,r11,27)	NOP	NOP	Movr r11,r7
13	(Addrr r13,r12,r11)	Addri r14,r11,27	NOP	NOP	NOP
14	?	Addrr r13,r12,r11	Addri r14,r11,27	NOP	NOP
15	?	?	Addrr r13,r12,r11	Addri r14,r11,27	NOP
16	?	?	?	Addrr r13,r12,r11	Addri r14,r11,27
17	?	?	?	?	Addrr r13,r12,r11
18	?	?	?	?	?

```
Fetch
Clock
Cycle     Instr.  Operands      Comment
1         loadi   r7, 10        # Put 10 in register 7, a load immediate instruction,
                                # where 10 is carried within a field of the loadi instruction
                                # and will be written into the register unit on clock cycle 5.
                                # For clarity, for each clock cycle, the pipeline state
                                # will be written in the form
                                #  [x] 1 → 2 → 3 → 4 → 5 where [x] is clock cycle x, and
                                # the current instruction being processed by each stage is
                                # listed followed by an arrow (→). ? means stage content
                                # is unknown. Content description will be abbreviated to
                                # fit  on a single line.
                                # [1] loadi r7, 10 → ? → ? → ? → ? .
```

```
2    loadi  r8, 15         # put 15 in register 8; write into reg. unit on clock cycle 6;
                           # [2] loadi r8,15 → loadi r7,10 → ? → ? → ? .
3    loadi  r9, 20         # put 20 in register 9; write into reg. unit on clock cycle 7
                           # [3] loadi r9,20 → loadi r8,15 → loadi r7,10 → ? → ? .
4    addrr  r10, r7, r8    # r10 ← r7 + r8
                           # [4] addrr r10 → loadi r9 → loadi r8 → loadi r7 → ? .
5    movr   r12, r9        # copy register r9 to register r12
                          # [5] mov r12 → (arr r10) → loadi r9 → loadi r8 →loadi r7.
                           # At this time, addrr r10,r7,r8 must stall because its
                           # operands are not yet written to the register unit. nop instr-
                           # uction must be inserted into "stage 3 ALU operation".
6    nop                   # Waste a clock cycle while r8 writes, stalling instructions
                           # in ( ) notation. r8 written during this clock cycle.
                           # [6] (mov r12) → (arr r10) → nop → loadi r9 → loadi r8.
                           # "Stage 2 fetch operands" will be able to run next clock.
7    nop                   # Stall ends, stages 1&2 start work, ( ) removed, and
                           # loadi r9 advances.
                           # [7] (mov r12,r9) → arr r10 → nop → nop → loadi r9.
8    movr   r11, r7        # all stages advance; r9 written by end of clock cycle 7, so
                           # movr r12, r9 can read r9 in stage 2 during clock cycle 8.
                           # [8] movr r11 → movr r12,r9 → addrr r10 → nop → nop.
9    addri  r14, r11, 27   #  r14 ← r11 + 27; r11 will not be written by the time it is
                           # needed, so another stall will occur
                           # [9] addri r14,r11 → mov r11 → m r12 → arr r10 → nop.
10   addrr  r13, r12, r11  # r13 ← r12 + r11; r12 and r11 will not write soon enough
                           # [10] arr r13 → ari r14,r11 → m r11 → m r12 → a r10.
11   nop                   # addri r14, r11, 27 waits for r11 to be written
                           # [11] (arr r13) → (ari r14) → nop → movr r11 → movr12.
12   nop                   # [12] (arr r13) → (ari r14,r11) → nop → nop → movr11.
13   nop                   # addri r14, r11, 27 reading operands
                           # [13] arr r13,r12, r11 → ari r14,r11 → nop → nop → nop.
14   ???                   # fetching unknown instruction ???; advancing pipeline
                           # [14] ??? → arr r13,r12, r11 → ari r14,r11 → nop → nop.
15   ???                   # fetching ???
                           # [15] ??? → ??? → arr r13,r12, r11 → ari r14,r11 → nop.
16   ???                   # [16] ??? → ??? → ??? → arr r13,r12,r11 → ari r14,r11.
17   ???                   # [17] ??? → ??? → ??? → ??? → addrr r13,r12,r11.
18   ???                   # pipeline clear of code snippet instructions
                           # [18] ??? → ??? → ??? → ??? → ???.
```

2. Text exercise 6.4.
   Answer: The offset field for instructions of the format used in chapter 6 includes 15 bits for an immediate value using either unsigned integer format or perhaps 2's complement representation. The decimal value 40,000 is too large to fit into 15 bits using either format, therefore, the instruction jump 40000(r15) is invalid.

3. Text exercise 6.6.
   Answer: The abstracted away clock signal controls the timing of the update of the contents of the 32-bit pgm.ctr. register, preventing wild, run-away infinite loop behavior.

4. Explain how CPU with two execution modes uses those modes to run an operating system and also execute code written by a user.
   Answer: The high-privilege and permission mode is used by the operating system so that it can manage all the resources of the computer system, including memory and input/output devices. The lower privilege, low permission mode is used by all application code, such as code written by a user, so that this code cannot accidentally or maliciously mis-manage computer system resources.

5. You are designing a microcoded version of the CPU in chapter 6 of our textbook. You have built a fully-functional circuit that includes all the components shown in Figure 6.9. What additional component do you need? To what should its input and its output be connected in Figure 6.9?
   Answer: You need a memory to store the microcode. The input of this memory should be connected to the instruction decode box in Figure 6.9. The output of this memory should be connected to the circuit in Figure 6.9 in essentially the same way as the outputs of the instruction decoder are connected to the rest of the circuit.

6. Assume that the IF (instruction fetch), ID (instruction decode), EX (instruction execute), MEM (data memory access, read or write), and WB (write-back to a register) stages of a 5-stage pipelined version of Figure 6.9 have propagation delays of 20, 30, 50, 50, and 10 nanoseconds (ns), or 10^{-9} seconds, respectively.
   a. What is the fastest clock rate for this pipeline?
      Answer: The clock rate is set by the slowest propagation delay, thus, 50 ns.

   b. Assume that the stage registers required for pipelining each have 3 ns of propagation delay to record an input. What was the propagation delay of the processor before pipelining?
      Answer: $20 - 3 + 30 - 3 + 50 - 3 + 50 - 3 + 10 = 148$ ns because the no register is added to the IF stage because the current_instruction_pointer register, more traditionally known as the program counter, is part of this portion of the processor circuit whether the circuit is pipelined or not. Also, the write-back stage has no register added to record the result of this stage. Rather, the result of write back is to write into one of the registers included in the unpipelined circuit.

   c. Ideally, how much faster is this processor once it is pipelined?
      Answer: The unpipelined circuit completes one instruction every 148 ns. Ideally, the pipelined circuit will complete one instruction every 50 ns. So the speedup is 148 ns / 50 ns = 2.96. The pipelined circuit is, ideally, 2.96 times faster than the unpipelined circuit.

7. Could the multiplexer controlled by the signal RegDst in Lecture 15, slide 52 be moved from the EX stage to the MEM stage without compromising the correct functioning of the

pipeline? Could this multiplexer be moved all the way to the WB stage and have correct operation of the pipeline be preserved? Why does the design in this slide place the multiplexer in the EX stage?
Answer: Correct operation is assured as long as the multiplexer correctly selects the input needed to point to the intended destination register. Whether this is done in the Ex stage, the MEM stage, or the WB stage is immaterial. All that is needed is a correct selection prior to writing into the register unit. This is possible regardless of the stage chosen. The reason the design shown in slide 52 chooses to locate the multiplexer in the EX stage is that this reduces the number of bits that need to be passed through the EX/MEM and the MEM/WB registers, saving 5 bits of storage circuitry for both stage registers.