LSU EE 4720 **Homework 2** **Due: 16 February 2018**

The solution to several of the problems in this assignment requires material about to be covered in class, in particular, stalling instructions to avoid hazards. For coverage of this material see slide set six, `http://www.ece.lsu.edu/ee4720/2018/lsli06.pdf`. *For a solved problem see 2014 Homework 1 Problem 3. Feel free to look through old homework and exams for other similar problems, but when doing so make sure that the MIPS implementation matches the one in this problem: the muxen at the ALU inputs should each have just 2 inputs.*

Problem 1: Recall that in the `unsqw` program from Homework 1 there was a loop that had to copy the prior occurrence of a piece of text to the output buffer. That loop from the solution appears below, and again re-written to improve performance, at least that was the goal.

```
# Original Code ------------------------------------------------------------
        #  Copy the prior occurrence of text from some part of the
        #  output buffer to the end of the output buffer.

        #  At this point in code:
        #    Reference marker is in $t0.
        #    Length is in register $t3.
        #    Distance is in register $t4.
        #
        sub $t4, $a1, $t4  # Compute starting address of prior occurrence.
        add $t5, $t4, $t3  # Compute ending address of prior occurrence.
        addi $a0, $a0, 1

COPY_LOOP:
        lb $t0, 0($t4)     # Load character of prior occurrence ..
        sb $t0, 0($a1)     # .. and write it to the end of the output buffer.
        addi $t4, $t4, 1
        bne $t4, $t5, COPY_LOOP
        addi $a1, $a1, 1
        j LOOP
        nop
```

```
# Improved Code ------------------------------------------------------------
        #  Copy the prior occurrence of text to the end of the output buffer.

        #  At this point in code:
        #    Reference marker is in $t0.
        #    Length is in register $t3.
        #    Distance is in register $t4.
        #
        sub $t4, $a1, $t4  # Compute starting address of prior occurrence.
        addi $a0, $a0, 1

        # Round length (L) up to a multiple of 4.
        #
        addi $t7, $t3, 3
        andi $t8, $t7, 0xfffc   # Note: this only works if L < 65536
        sub $t6, $t8, $t3
        #
        # At this point:
        #   $t8: L', rounded-up length.
        #   $t6: Amount added to L to round it up. That is, L' - L
        #        t6 can be 0, 1, 2, or 3.
        #        If t6 is 0, then L' = L;
        #        if t6 is 1, then L' = L + 1; etc.

        # Decrement prior-occurrence and output-buffer pointers.
        #
        sub $t4, $t4, $t6
        sub $a1, $a1, $t6

        # Jump to one of the four lb instructions in the copy loop.
        #
        la $t7, COPY_LOOPd4   # Get address of first lb instruction.
        sll $t6, $t6, 3       # Compute offset to lb that we want to start at.
        add $t7, $t7, $t6     # Compute address of starting lb ..
        jr $t7                # .. and jump to it.
        add $t5, $t4, $t8     # Don't forget to compute stop address.

COPY_LOOPd4:
        lb $t0, 0($t4)
        sb $t0, 0($a1)
        lb $t0, 1($t4)
        sb $t0, 1($a1)
        lb $t0, 2($t4)
        sb $t0, 2($a1)
        lb $t0, 3($t4)
        sb $t0, 3($a1)
        addi $t4, $t4, 4
        bne $t4, $t5, COPY_LOOPd4
        addi $a1, $a1, 4
        j LOOP
        nop
```

Let L denote the length of the prior occurrence of text to copy.

(a) Determine the number of instructions executed by the original code in terms of L. Include the copy loop and the instructions before it shown above. State any assumptions.

(b) Determine the number of instructions executed by the improved code in terms of L. Include the copy

loop and the instructions before it shown above. State any assumptions.

(*c*) What is the minimum value of L for the improved method to actually be faster?

(*d*) What is it about the improved code that helps performance?

Problem 2: *Note: The following problem is identical to 2016 Homework 1 Problem 1. Try to solve it without looking at the solution.* Answer each MIPS code question below. Try to answer these by hand (without running code).

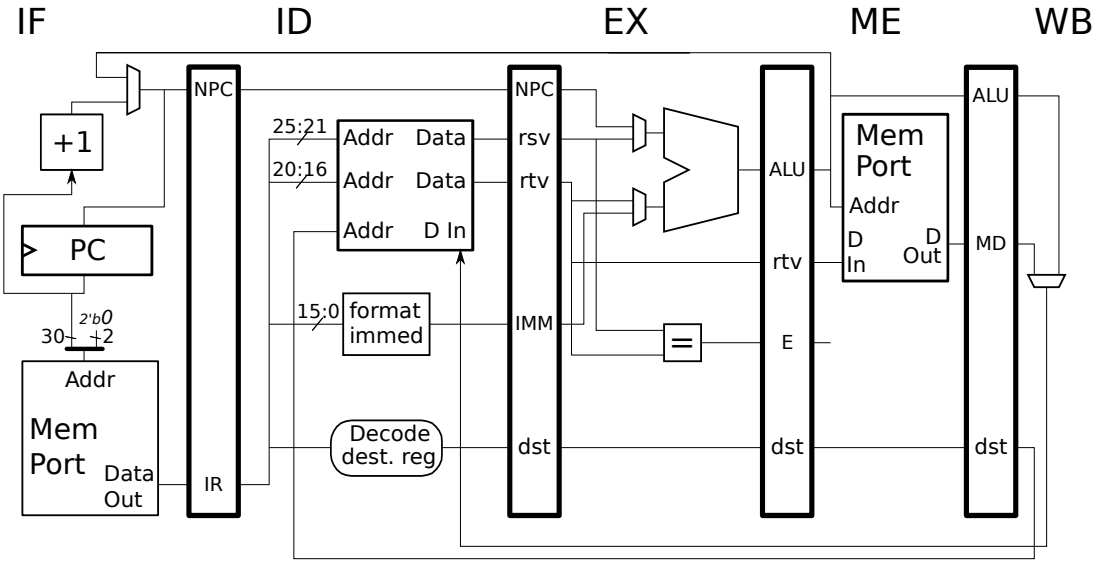
(*a*) Show the values assigned to registers `t1` through `t8` (the lines with the tail comment `Val:`) in the code below. Refer to the MIPS review notes and MIPS documentation for details.

```
        .data
myarray:
        .byte 0x10, 0x11, 0x12, 0x13
        .byte 0x14, 0x15, 0x16, 0x17
        .byte 0x18, 0x19, 0x1a, 0x1b
        .byte 0x1c, 0x1d, 0x1e, 0x1f


        .text
        la $s0, myarray   # Load $s0 with the address of the first value above.
                          # Show value retrieved by each load below.
        lbu $t1, 0($s0)   # Val:
        lbu $t2, 1($s0)   # Val:
        lbu $t2, 5($s0)   # Val:
        lhu $t3, 0($s0)   # Val:
        lhu $t4, 2($s0)   # Val:

        addi $s1, $0, 3

        add $s3, $s0, $s1
        lbu $t5, 0($s3)   # Val:

        sll $s4, $s1, 1
        add $s3, $s0, $s4
        lhu $t6, 0($s3)   # Val:

        sll $s4, $s1, 2
        add $s3, $s0, $s4
        lhu $t7, 0($s3)   # Val:
        lw $t8, 0($s3)    # Val:
```

(*b*) The last two instructions in the code above load from the same address. Given the context, one of those instructions looks wrong. Identify the instruction and explain why it looks wrong. (Both instructions should execute correctly, but one looks like it's not what the programmer intended.)

(*c*) Explain why the following answer to the question above is wrong for the MIPS 32 code above: *"The* `lw` *instruction should be a* `lwu` *to be consistent with the others."*

Problem 3: *Note: The following problem was assigned in each of the last three years, and its solution is available. DO NOT look at the solution unless you are lost and can't get help elsewhere. Even in that case just glimpse.* Appearing below are **incorrect** executions on the illustrated implementation. For each one explain why it is wrong and show the correct execution.



(*a*) Explain error and show correct execution.

```
LOOP: # Cycles     0  1  2  3  4  5  6  7
 lw r2, 0(r4)      IF ID EX ME WB
 add r1, r2, r7       IF ID EX ME WB
LOOP: # Cycles     0  1  2  3  4  5  6  7
```

(*b*) Explain error and show correct execution.

```
LOOP: # Cycles     0  1  2  3  4  5  6  7
 add r1, r2, r3    IF ID EX ME WB
 lw r1, 0(r4)         IF ID -> EX ME WB
LOOP: # Cycles     0  1  2  3  4  5  6  7
```

(*c*) Explain error and show correct execution.

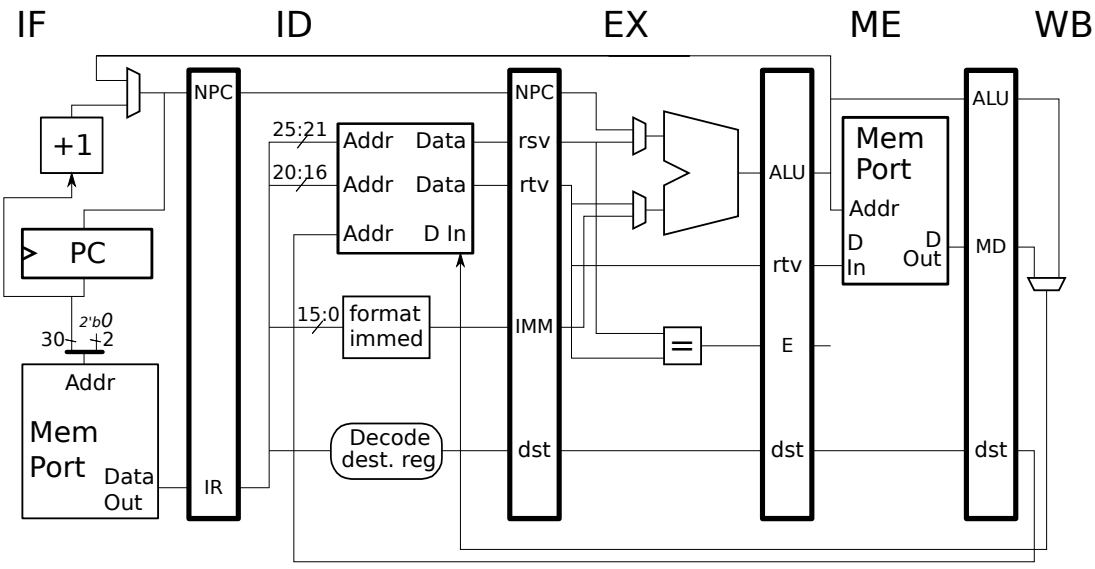
```
LOOP: # Cycles     0  1  2  3  4  5  6  7
 add r1, r2, r3    IF ID EX ME WB
 sw r1, 0(r4)         IF ID -> EX ME WB
LOOP: # Cycles     0  1  2  3  4  5  6  7
```

(*d*) Explain error and show correct execution.

```
LOOP: # Cycles     0  1  2  3  4  5  6  7
 add r1, r2, r3    IF ID EX ME WB
 xor r4, r1, r5       IF ----> ID EX ME WB
LOOP: # Cycles     0  1  2  3  4  5  6  7
```

Problem 4: The MIPS code below is taken from the solution to 2018 Homework 1. Show the execution of this MIPS code on the illustrated implementation for two iterations. The register file is designed so that if the same register is simultaneously written and read, the value that will be read will be the value being written. (In class we called such a register file *internally bypassed*.)

- Check carefully for dependencies.
- Focus on when the branch target is fetched and on when wrong-path instructions are squashed.
- Be sure to stall when necessary.




```
CLOOP:
 lbu $t0, 0($t4)
 sb $t0, 0($a1)
 addi $t4, $t4, 1
 bne $t4, $t5, CLOOP
 addi $a1, $a1, 1
```