LSU EE 4720 **Homework 1** **Due: 12 February 2016**

Problem 1: Answer each MIPS code question below. Try to answer these by hand (without running code).

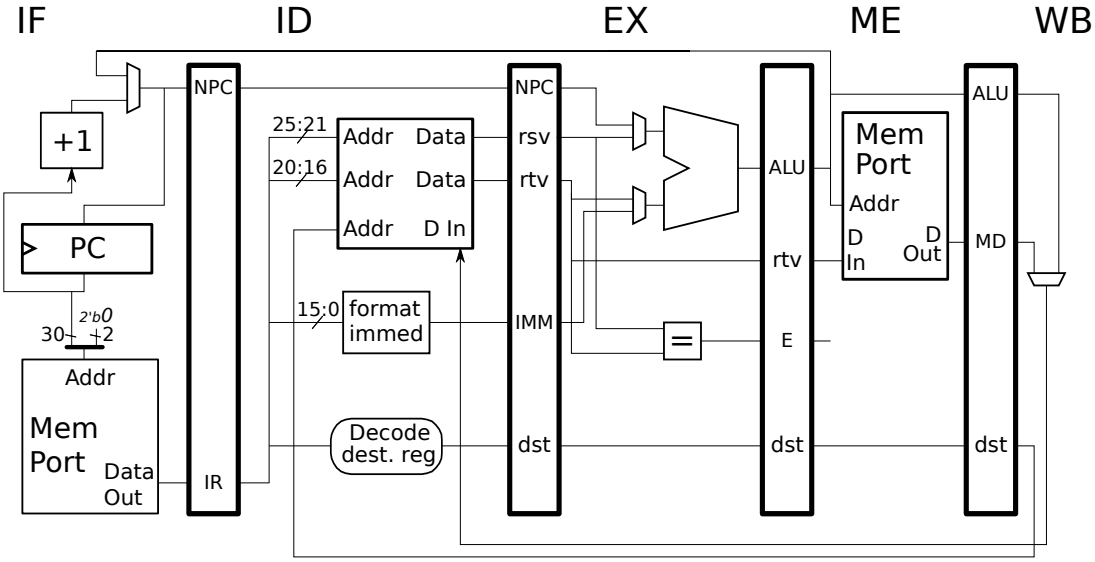
(*a*) Show the values assigned to registers `t1` through `t8` (the lines with the tail comment `Val:`) in the code below. Refer to the MIPS review notes and MIPS documentation for details.

```
        .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 2: *Note: The following problem was assigned last year and two years ago 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.

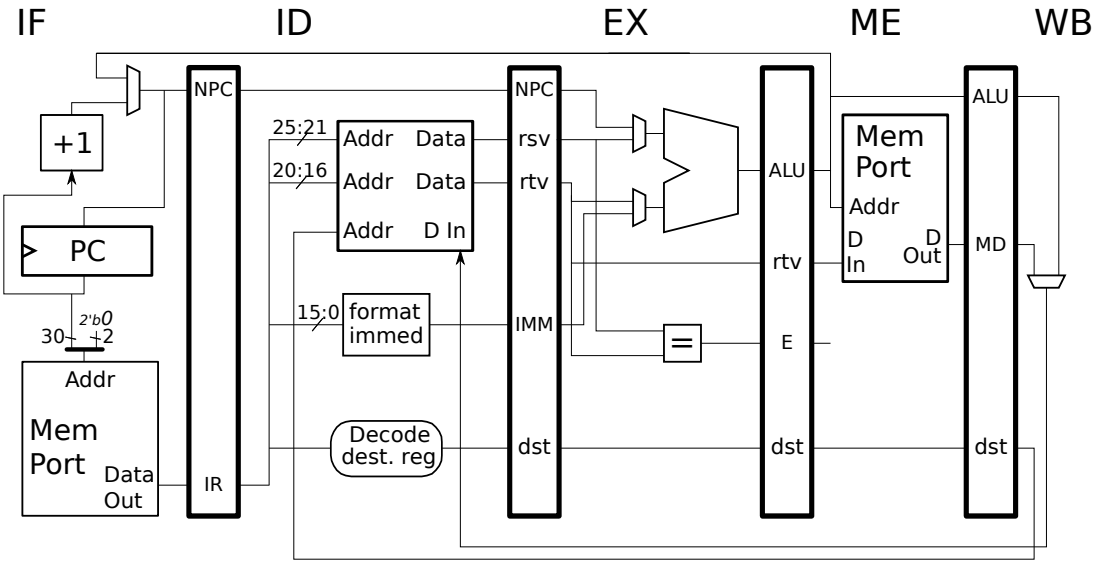
```
LOOP: # Cycles     0  1  2  3  4  5  6  7
 add r1, r2, r3    IF ID EX ME WB
 sw r1, 0(r4)         IF ID -> EX ME WB
LOOP: # Cycles     0  1  2  3  4  5  6  7
```

(*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 3: Show the execution of the MIPS code below on the illustrated implementation. The register file is designed so that if the same register is simultaneously written and read, the value that will be read will be value being written. (In class we called such a register file *internally bypassed*.)

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




```
 add r1, r2, r3

 sub r4, r1, r5

 beq r1, r1, SKIP

 lw r6, 0(r4)

 xor r7, r8, r9

SKIP:

 ori r9, r10, 11
```