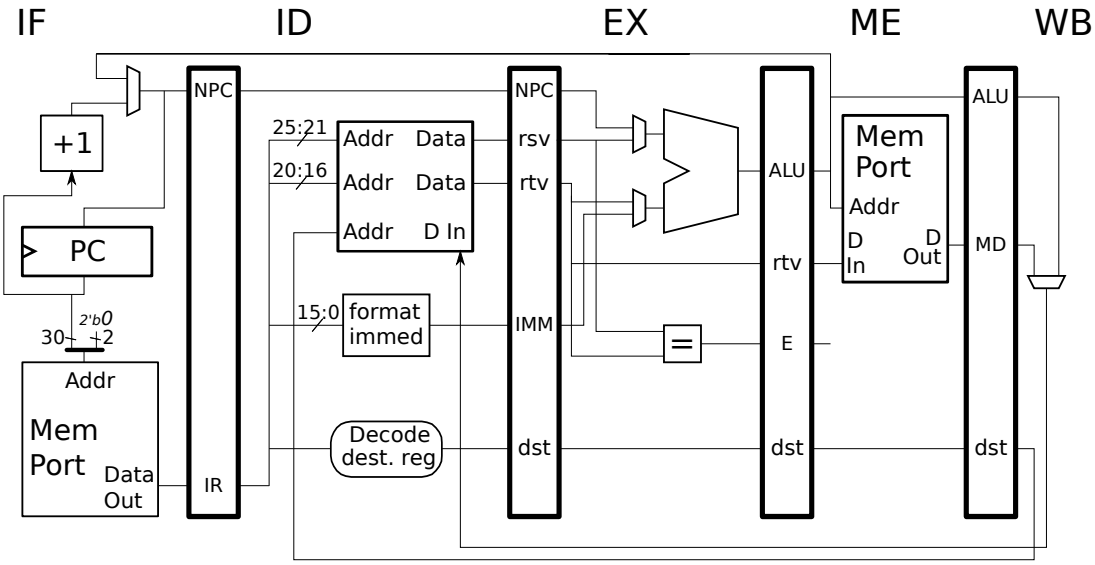
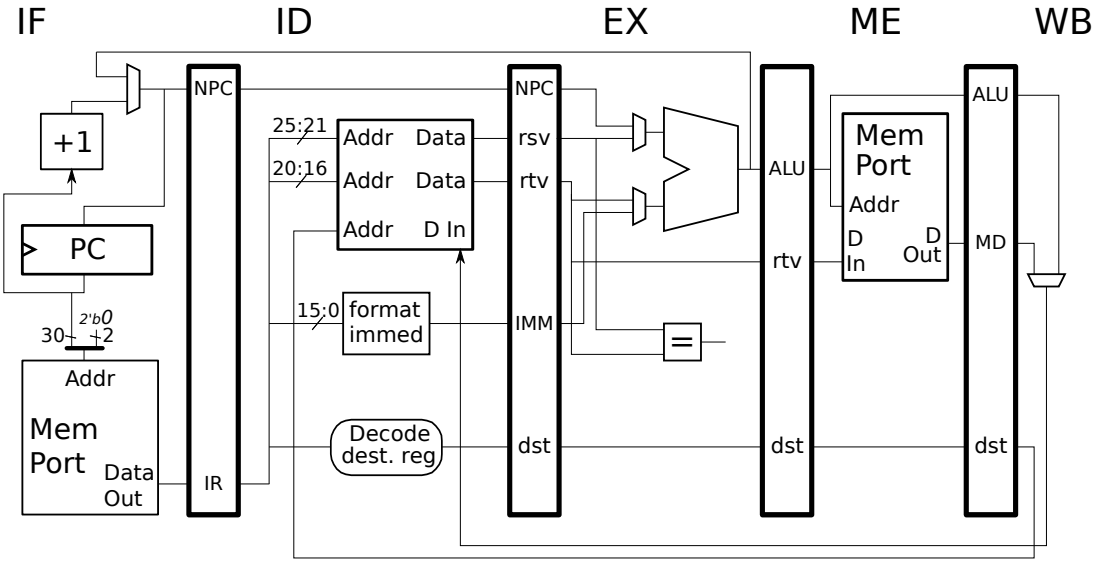
LSU EE 4720 **Homework 3** **Due: 5 March 2018**

Problem 1: Appearing below are two MIPS implementations, *The First Implementation* is taken from Homework 2 Problem 4. Branches suffer a two-cycle penalty on this implementation since they resolve in `ME`. On the *The Second Implementation* branches resolve in `EX` reducing the penalty to one cycle. For convenience for those using 2-sided printers the same implementations are shown again on the next page.

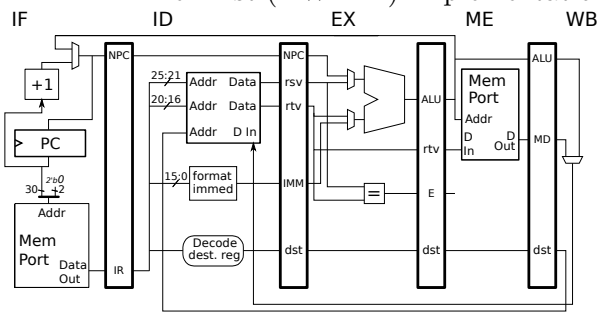
The First (HW2 P4) Implementation



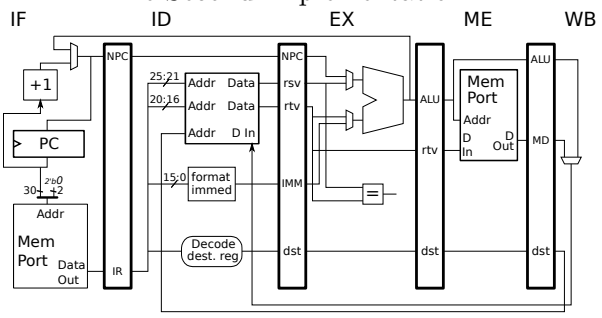
The Second Implementation



The First (HW2 P4) Implementation



The Second Implementation



The code fragment below and its execution on The First Implementation is taken from the solution to Homework 2 Problem 4. Notice that the branch suffers a two-cycle branch penalty.

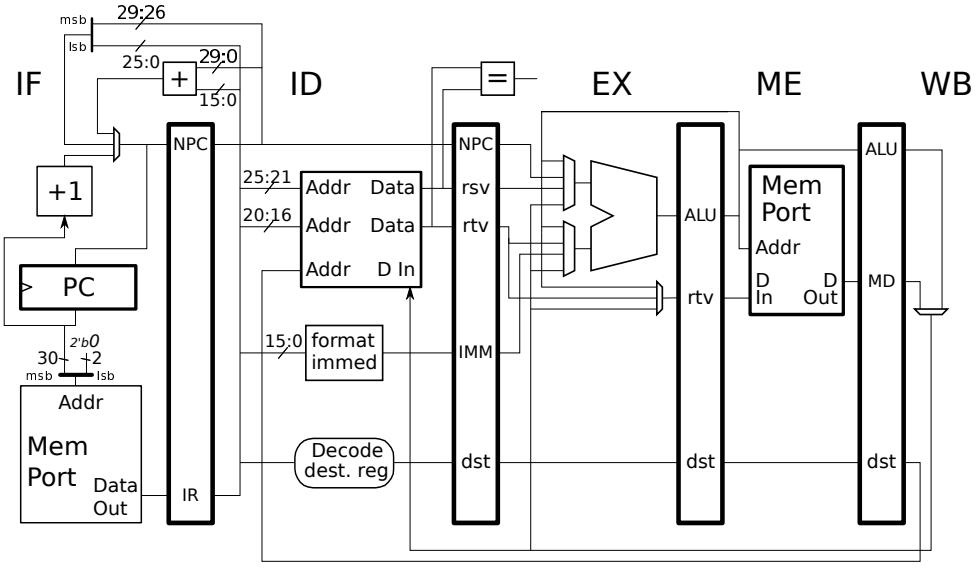
```
CLOOP:  # Cycle  0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 The 1st Implementation
 lbu $t0, 0($t4) IF ID EX ME WB
 sb $t0, 0($a1)     IF ID ----> EX ME WB
 addi $t4, $t4, 1      IF ----> ID EX ME WB
 bne $t4, $t5, CLOOP            IF ID ----> EX ME WB
 addi $a1, $a1, 1                  IF ----> ID EX ME WB
 X1                                         IF IDx
 X2                                            IFx
CLOOP:  # Cycle  0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23
 lbu $t0, 0($t4)                                  IF ID EX ME WB
 sb $t0, 0($a1)                                      IF ID ----> EX ME WB
 addi $t4, $t4, 1                                       IF ----> ID EX ME WB
 bne $t4, $t5, CLOOP                                             IF ID ----> EX ME WB
 addi $a1, $a1, 1                                                   IF ----> ID EX ME WB
 X1                                                                          IF IDx
 X2                                                                             IFx
CLOOP:  # Cycle  0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23
 lbu $t0, 0($t4)                                                                    IF ID
```

(*a*) On The Second Implementation the branch penalty would only be one cycle. But, as we discussed in class, moving branch resolution from ME to EX might impact the critical path. Let $\phi_1 =$ 1 GHz denote the clock frequency on The First Implementation and call the clock frequency on The Second Implementation ϕ_2. For what value of ϕ_2 would the performance of the two implementations be the same when executing the code above for a large number of iterations?

Show your work.

Problem 2: The code below is taken from the solution to Homework 2 Problem 1. Sharp students might remember that the loop can be entered at four places: the `COPY_LOOPd4` label (which is the normal way to enter such a loop), the second `lb`, the third `lb`, or the fourth `lb`. For this problem assume that the loop can only be entered at the `COPY_LOOPd4` label.

```
COPY_LOOPd4:
    lb $t0, 0($t4)  # First lb
    sb $t0, 0($a1)
    lb $t0, 1($t4)  # Second lb
    sb $t0, 1($a1)
    lb $t0, 2($t4)  # Third lb
    sb $t0, 2($a1)
    lb $t0, 3($t4)  # Fourth lb
    sb $t0, 3($a1)
    addi $t4, $t4, 4
    bne $t4, $t5, COPY_LOOPd4
    addi $a1, $a1, 4
```



(*a*) Schedule the code (rearrange the instructions) so that it executes without a stall on the implementation shown above.

Problem 3: Perhaps some students have already wondered why, if the goal were to reduce dynamic instruction count, the previous occurrence loop (the subject of the first two problems and of Homework 2) wasn't written using `lw` and `sw` instructions since they handle four times as much data. Such a loop appears below. Alas, the loop won't work for every situation, for one reason due to MIPS' alignment restrictions.

Let a_p denote the address of the previous text occurrence (the value is in `t4`), let a_o denote the address of the next character to write into the output buffer (the value is in `a1`), and let L denote the length of the previous occurrence to copy. (Register `t5` is $a_p + L$.)

```
COPY_LOOP44:
        lw $t0, 0($t4)
        sw $t0, 0($a1)
        addi $t4, $t4, 4
        bne $t4, $t5, COPY_LOOP44
        addi $a1, $a1, 4
        j LOOP
        nop
```

(*a*) In terms of a_p, a_o, and L, specify the conditions under which the loop above will run correctly. Also show that the loop would work for about only 1 out of 64 copies assuming that the values of a_p, a_o, and L, are uniformly distributed over some large range. For this part don't assume any special code added before or after.

(*b*) Suppose one added *prologue code* before the loop to copy the first few characters and *epilogue code* after the loop to copy the last few characters, with the goal of being able to use the loop for more than $\frac{1}{64}$th (or $\frac{100}{64}\%$) of copies.

In terms of a_p, a_o, and L, specify the conditions under which the loop will run correctly and show that the fraction of copies that the loop can handle is about $\frac{1}{4}$.

Also show the number of characters that should be copied by the prologue code and the number of characters that should be copied by the epilogue code.