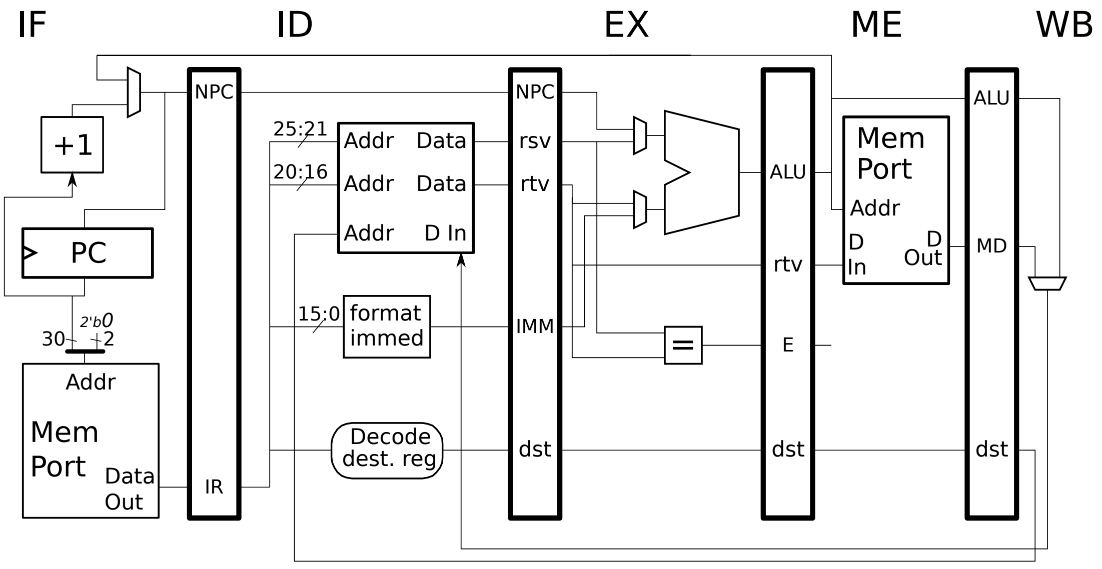
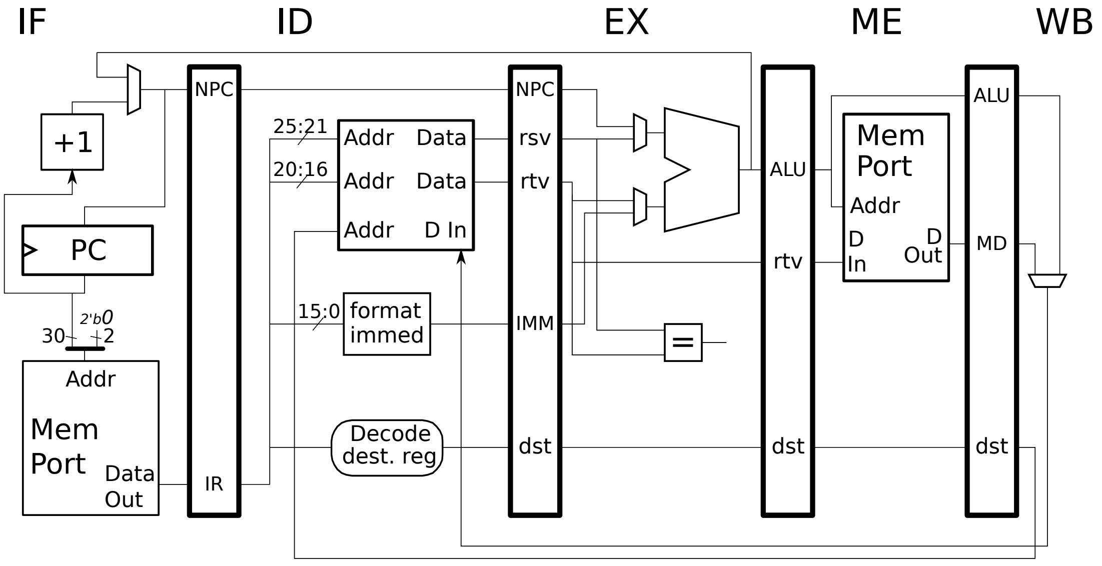
LSU EE 4720 **Homework 3** Solution **Due: 5 March 2018**

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.

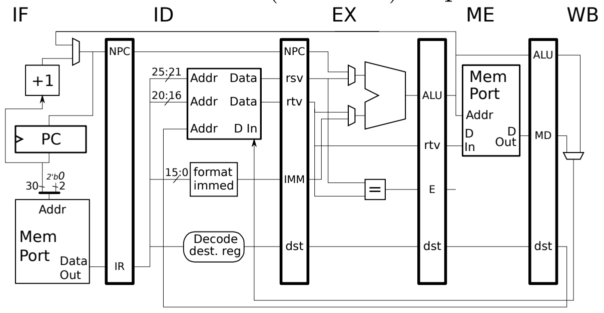
The First (HW2 P4) Implementation



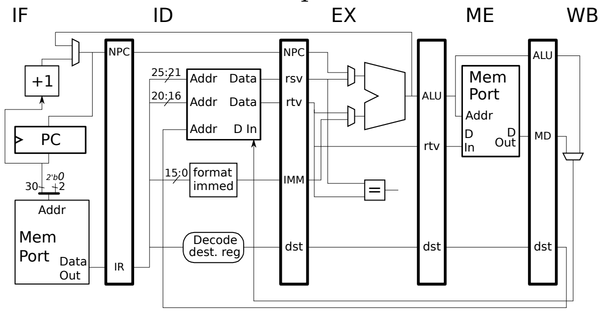
The Second Implementation



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
```

```
CLOOP:  # Cycle  0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 The 2nd 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                                         IFx
 X2
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                                                                       IFx
 X2
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
```

(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 $\phi_1 = 1\,\text{GHz}$ 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.

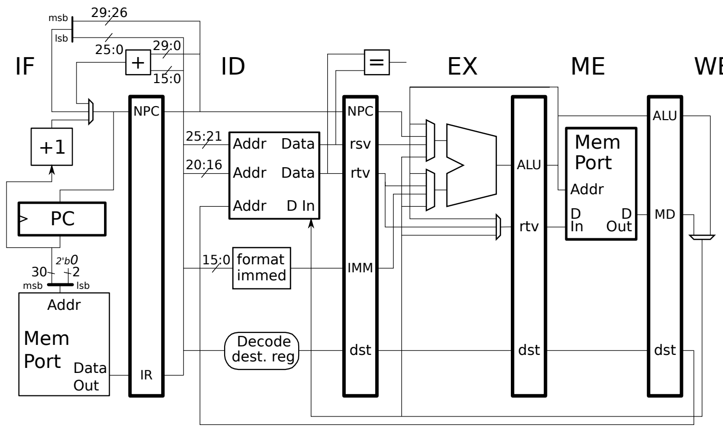
Short answer: $\phi_2 = \phi_1 \frac{10}{11} = 909.09\,\text{MHz}$.

Explanation: One loop iteration on The First Implementation takes 11 clock cycles. (The duration of loop iteration x can be found by subtracting the fetch time of the first instruction of iteration x from the fetch time of the first instruction of iteration $x+1$. In the loop above iteration 0 takes $11 - 0 = 11\,\text{cyc}$ and iteration 1 takes $22 - 11 = 11\,\text{cyc}$. There's no guarantee that iteration 0 and 1 will take the same amount of time. However we can expect iteration 2 to take as much time as iteration 1 because the state of the pipeline is identical at cycles 11 and 22: `lbu` in `IF`, `addi` in `ME`, and `bne` in `WB`.) By similar reasoning one loop iteration on The Second Implementation takes 10 cycles. Dividing cycles by clock frequency gives time, so 11 cycles on The First Implementation takes $\frac{11}{\phi_1} = 11\,\text{ns}$. To find the clock frequency for The Second Implementation at which the two perform equally solve $\frac{11}{\phi_1} = \frac{10}{\phi_2}$ for ϕ_2, $\phi_2 = \phi_1 \frac{10}{11} = 909.09\,\text{MHz}$.

Grading Note: A common mistake was to assume that in The Second Implementation the `sb` and `bne` suffer only 1-cycle stalls rather than the 2-cycles that they suffer in The First Implementation. That's not a good mistake to make because the only change between the two implementations is where the branch resolves. The two-cycle stall starting in cycle 3 is due to the `sb` waiting for the value produced by the `lbu`, the stall starting in cycle 7 is due to the `bne` waiting for the value of `t4` written by `addi`. The change does not effect either stall.

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.

The solution appears below. The **sb** instructions have been moved away from the **lb** instructions, avoiding **lb/sb** stalls. The increment of **t4** has been moved up, avoiding the need for the branch to stall. Note that the destination register of all but the first **lb** and the source of all but the first **sb** had to be changed to avoid the second **lb** clobbering the value loaded by the first **lb**, etc.

Grading Note: One common mistake was to leave the **lb** and **sb** registers unchanged.

```
COPY_LOOPd4:  SOLUTION
    lb $t0, 0($t4)     IF ID EX ME WB
    lb $t1, 1($t4)        IF ID EX ME WB
    lb $t2, 2($t4)           IF ID EX ME WB
    lb $t9, 3($t4)
    addi $t4, $t4, 4
    sb $t0, 0($a1)
    sb $t1, 1($a1)
    sb $t2, 2($a1)
    sb $t9, 3($a1)
    bne $t4, $t5, COPY_LOOPd4
    addi $a1, $a1, 4
```

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.

The loop will only run correctly if a_p, a_o, and L are all multiples of 4 and if $L \geq 4$. That is $a_p \bmod 4 = 0$, $a_o \bmod 4 = 0$, $L \bmod 4 = 0$, and $L \geq 4$.

Suppose that previous occurrence lengths are uniformly distributed over range $[1, 4M]$, for some integer M. Because they are uniformly distributed the probability of each value in the range is $P(L = x) = \frac{1}{4M}$ for $1 \leq x \leq 4M$. Since $L \bmod 4 = 0$ for M values in the range, those values occur $\frac{1}{4}$ of the time. The same argument can be made for the address of the previous occurrence and the current buffer location. If each is a multiple of four $\frac{1}{4}$ of the time and assuming their values are independent, all of them are multiples of four $\left(\frac{1}{4}\right)^3 = \frac{1}{64}$ of the time.

(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.

If the only requirement were that L be a multiple of 4, then prologue code could copy the first $L \bmod 4$ characters and the loop would handle the remaining $L' = L - L \bmod 4$ characters. But `t4` and `a1` must both also be a multiple of 4 when the `COPY_LOOP44` loop is entered. Both!

Suppose $(a_p \bmod 4) = (a_o \bmod 4)$ and let $m = (a_o \bmod 4)$ and $b = \begin{cases} 0, & \text{if } m = 0; \\ 4 - m, & \text{otherwise.} \end{cases}$. If the prologue loop executes b iterations (and advances `t4` and `a1`) then when the `COPY_LOOP44` is entered both `t4` and `a1` will be multiples of 4. Execute $L' = \lfloor (L - b)/4 \rfloor$ iterations in `COPY_LOOP44` and then execute $L - 4L' - b$ iterations in the epilogue code.

If $a_p \bmod 4 \neq a_o \bmod 4$ then `COPY_LOOP44` cannot work.

Suppose a_p, a_0, and L are uniformly distributed and independent, as considered in the second part. Then $a_p \bmod 4 = a_o \bmod 4$ occurs $\frac{1}{4}$ of the times because there are 4 possible values of $a_p \bmod 4$ and 4 values for $a_o \bmod 4$, for a total of 16 pairs. Each pair is equally likely with probability $\frac{1}{16}$. For four of those pairs $a_p \bmod 4 = a_o \bmod 4$, the probability of any of those favorable pairs occurring is $4\frac{1}{16} = \frac{1}{4}$. The value of L does not matter so long as it is $L \geq 10$ since the epilogue will handle the last 1-3 characters.