# CA HW4 handwritten part

## Problem1

Let's think about the benefits of pipelining. Consider a simple processor that has five pipeline stages, IF - ID - EX - MEM - WB (just like those introduced in the textbook). Suppose each stage has a latency as the following:

| IF | ID | EX | MEM | WB |
|---|---|---|---|---|
| 250ps | 360ps | 150ps | 280ps | 200ps |

### (a)

Determine the clock cycle time of this processor. Also, determine the clock cycle time if this is a single-cycle processor.

### (b)

What's the latency of an `sw` instruction? Also, what's the latency of it under the single-cycle design?

> For (c) and (d), suppose we execute a large program to test the performance, and there are no stalls or hazards occurring.

### (c)

Compare the performance of the pipeline and the single-cycle design.

### (d)

If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split? How much is the performance improvement compared to the original five stage design?

### (e)

Write down one scenario that can cause bubbles in the pipeline. How can we handle it?

## Problem2

For the following code:

```
c = a + b;
c = tool[c];
d = tool[a];
e = tool[b] + a;
a = e;
```

We have **a, b, c, d, e** 5 variable and an array **tool** with the table:

| variable | a | b | c | d | e | tool's based address |
|---|---|---|---|---|---|---|
| rigister | x6 | x7 | x8 | x9 | x10 | x5 |

And x12, x13 for temporary used.
We can transform the code naively into such RISC-V code:

```
# c = a + b
add x8, x6, x7

# c = tool[c]
add x12, x8, x0
slli x12, x12, 2
add x13, x5, x12
lw x8, 0(x13)

# d = tool[a]
add x12, x6, x0
slli x12, x12, 2
add x13, x5, x12
lw x9, 0(x13)

# e = tool[b] + a
add x12, x7, x0
slli x12, x12, 2
add x13, x5, x12
lw x12, 0(x13)
add x10, x12, x6

# a = e
add x6, x10, x0
```

Please answer the following question:

## (a)

For five stage pipeline, please list out all the dependency happend in the RISC-V code with categorized into RAW / WAW / WAR.
（only needs to consider the dependency nearby 1 instrction）
**Example format:**

```
RAW:
x8  in line 2 & 5 -> you need to write this
x12 in line 5 & 7 -> no need to consider
...
WAW:
...
WAR:
...
```

## (b)

Supposed there is no cache miss, and all I/s can be done in one cycle.

How many cycles dose the code needs to execute with / without forwarding unit?

(Write-back occurs in the first half of the cycle, while register read takes place in the second half.)

## (c)

If unfortunely we don't have forwarding unit, we still can furthur optimized the code by handling the dependency, such as renaming, reordering, etc..

Please use **renaming & reordering** technique to achieve **at least 2x performance improvement** by wirting down the code after optimization and do some explaination to your code.

You can use extra register, with introducing how you use them.

(Write-back occurs in the first half of the cycle, while register read takes place in the second half.)