## CMPU 224 Quiz 3 Practice Solutions

**Problem 1** Given the HCL code below, fill in the missing HCL code for the decode stage in our sequential processor implementation.

```hcl
## What register should be used as the A source?
word srcA = [
    icode in { IRRMOVQ, IRMMOVQ, IOPQ, IPUSHQ } : rA;
    icode in { IPOPQ, IRET } : RRSP;
    1 : RNONE; # Don't need register
];

## What register should be used as the B source?
word srcB = [
    icode in { IOPQ, IRMMOVQ, IMRMMOVQ, IIADDQ } : rB;
    icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;
    1 : RNONE; # Don't need register
];
```

**Problem 2** Determine the machine-level byte encoding of the Y86-64 instructions below. Write your answer in hexadecimal.

<table>
<thead>
<tr>
<th>Y86-64 Assembly</th>
<th>Y86-64 Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>subq %rax, %rdi</td>
<td>6107</td>
</tr>
<tr>
<td>iaddq $-1, %rcx</td>
<td>c0 ffffffff0000000000000000</td>
</tr>
<tr>
<td>mrmovq 5(%rsi), %rbp</td>
<td>50 56 05000000000000000000000000</td>
</tr>
<tr>
<td>ret</td>
<td>90</td>
</tr>
<tr>
<td>irmovq $42, %rbx</td>
<td>30 f3 2a0000000000000000000000</td>
</tr>
</tbody>
</table>
**Problem 3** Determine the **Y86-64** instructions from the machine-level byte encoding below. All bytes are specified as hexadecimal numbers.

<table>
<thead>
<tr>
<th>Y86-64 Machine Code</th>
<th>Y86-64 Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>803000000000000000</td>
<td>call 48 or call 0x30</td>
</tr>
<tr>
<td>714200000000000000</td>
<td>jle 66 or jle 0x42</td>
</tr>
<tr>
<td>2289</td>
<td>cmovl %r8, %r9</td>
</tr>
<tr>
<td>b0cf</td>
<td>popq %r12</td>
</tr>
<tr>
<td>00</td>
<td>halt</td>
</tr>
</tbody>
</table>

**Problem 4** Using Y86-64 code, show an example of a RAW (read after write) hazard.

Any code where a destination register is used as a source in the next instruction.

addq %rax, %rcx
addq %rcx, %rdx

**Problem 5** Using Y86-64 code, show an example of a Load/Use hazard.

Any code where the destination register of the mrmovq or popq instruction is being used as a source operand in the next instruction.

mrmovq (%rax), %rcx
addq %rcx, %rdx

**Problem 6** A three-stage pipeline requires 20 ps to load its pipeline registers. Stage A takes 100 ps to complete, stage B takes 60 ps to complete, and stage C takes 80 ps to complete. What is the overall latency of this pipeline?

The clock rate of the pipeline is limited by its slowest stage. The slowest stage takes 100 ps to complete, plus 20 ps to load the pipeline registers, giving a total of 120 ps for each stage. There are three stages to the pipeline, so the overall latency of the pipeline is 120 * 3 = 360 ps.
Problem 7 Translate the following C code into the equivalent Y86-64 assembly language function. Briefly comment your Y86 assembly language function.

### C Function

```c
// Takes an array of longs and returns a checksum of the input. This
// is done by taking the exclusive or (xor) of each long in the array.

// lst - array of longs
// len - length of the array
long cksum(long lst[], long len) {
    long i;
    long cksum = 0;
    for (i = 0; i < len; i++) {
        cksum = cksum ^ lst[i];
    }
    return cksum;
}
```

### Y86-64 Assembly Function

```assembly
cksum:
    xorq %rax, %rax          # cksum = 0
    xorq %rdx, %rdx          # i = 0
    jmp test                # loop test

loop:
    mrmovq (%rdi), %r10     # get *lst
    xorq %r10, %rax         # xor cksum
    iaddq $8, %rdi          # move to next element
    iaddq $1, %rdx          # i++

    test:
        rrmovq %rdx, %rcx     # save i
        subq %rsi, %rcx       # i:len
        jl loop               # if (i < len) loop
        ret                   # Return

    cksum:
        xorq %rax, %rax       # cksum = 0
        andq %rsi, %rsi       # Set CC based on len
        je done               # stop if len is 0

    loop:
        mrmovq (%rdi), %r10  # Get *lst
        xorq %r10, %rax       # xor cksum
        iaddq $8, %rdi        # move to next element
        iaddq $-1, %rsi       # len--
        jg done               # Stop when 0

    done:
        ret                   # Return
```