Processor Architecture #1 **4190.308 Computer Architecture**

Due Date: Thursday, November 6, 2014, 23:59 **Solution**

Submission: in paper form.
There will be a drop off box in class and in front of the CSAP Lab in building 301, room 419.

Question 1

Harvard and Von Neumann architecture

Explain difference between *Harvard* and *Von Neumann* architecture.

Under pure von Neumann architecture the CPU can be either reading an instruction or reading/writing data from/to the memory. Both cannot occur at the same time since the instructions and data use the same bus system. In a computer using the Harvard architecture, the CPU can both read an instruction and perform a data memory access at the same time, even without a cache. A Harvard architecture computer can thus be faster for a given circuit complexity because instruction fetches and data access do not contend for a single memory pathway.

Question 2
The Y86 Instruction Set Architecture

For each byte sequences listed, determine the Y86 instruction sequence it encodes. If there is some invalid byte in the sequence, show the instruction sequence up to that point and indicate where the invalid value occurs. For each sequence, we show the starting address, then a colon, and then byte sequence.

A. `0x100:30f3fcffffff40630008000000`

```
0x100: 30f3fcffffff |    irmovl $-4,%ebx
0x106: 406300080000 |    rmmovl %esi, 0x800(%ebx)
0x10c: 00           |    halt
```

B. `0x200:a06f80080200000030f30a00000090`

```
0x200: a06f         |    pushl %esi
0x202: 8008020000   |    call proc
0x207: 00           |    halt
0x208:              | proc:
0x208: 30f30a000000 |    irmovl $10, %ebx
0x20e: 90           |    ret
```

C. `0x300:505407000000 10f0b01f`

```
0x300: 505407000000 |    mrmovl 7(%esp), %ebp
0x306: 10           |    nop
0x307: f0           | .byte 0xf0  # invalid instruction code
0x308: b01f         |    popl %ecx
```

D. `0x400:6113730004000000`

```
0x400:              | loop:
0x400: 6113         |    subl %ecx, %ebx
0x402: 7300040000   |    je loop
0x407: 00           |    halt
```

E. `0x500:6362a0f0`

```
0x500: 6362         |    xorl %esi, %edx
0x502: a0           |    .byte 0xa0  # pushl instruction code
0x503: f0           |    .byte 0xf0  # Invalid register specifier byte
```

Question 3

The Y86 Instruction Set Architecture

Write Y86 code on the right cell to implement a recursive sum function `rSum`, based on the following C code:

```
int rSum(int *Start, int Count)
{
        if (Count <= 0)
                return 0;
        return *Start + rSum(Start+1,
Count-1);
}
```

```
# int Sum(int *Start, int Count)
rSum: pushl %ebp
      rrmovl %esp,%ebp
      pushl %ebx            # Save value of %ebx
      mrmovl 8(%ebp),%ebx   # Get Start
      mrmovl 12(%ebp),%eax # Get Count
      andl %eax,%eax       # Test value of Count
      jle L38
      irmovl $-1,%edx
      addl %edx,%eax        # Count--
      pushl %eax            # Push Count
      irmovl $4,%edx
      rrmovl %ebx,%eax
      addl %edx,%eax
      pushl %eax            # Push Start+1
      call rSum           # Sum(Start+1,Count-1)
      mrmovl (%ebx),%edx
      addl %edx,%eax        # Add *Start
      jmp L39               # goto done
L38:  xorl %eax,%eax        # zreturn;
L39:  mrmovl -4(%ebp),%ebx # done: Restore %ebx
      rrmovl %ebp,%esp # Deallocate stack frame
      popl %ebp             # restore %ebp
      ret
```

You might find it helpful to compile the C code on an IA32 machine and then translate the instructions to Y86.

Question 4
Logic Design and the Hardware Control Language HCL

Write HCL code describing a circuit that for word inputs **A**, **B**, and **C** selects the *median* of the three values. That is, the output equals the word lying between the minimum and maximum of the three inputs.

```
int Med3 = [
    A <= B && B <= C : B;
    C <= B && B <= A : B;
    B <= A && A <= C : A;
    C <= A && A <= B : A;
    1                : C;
];
```