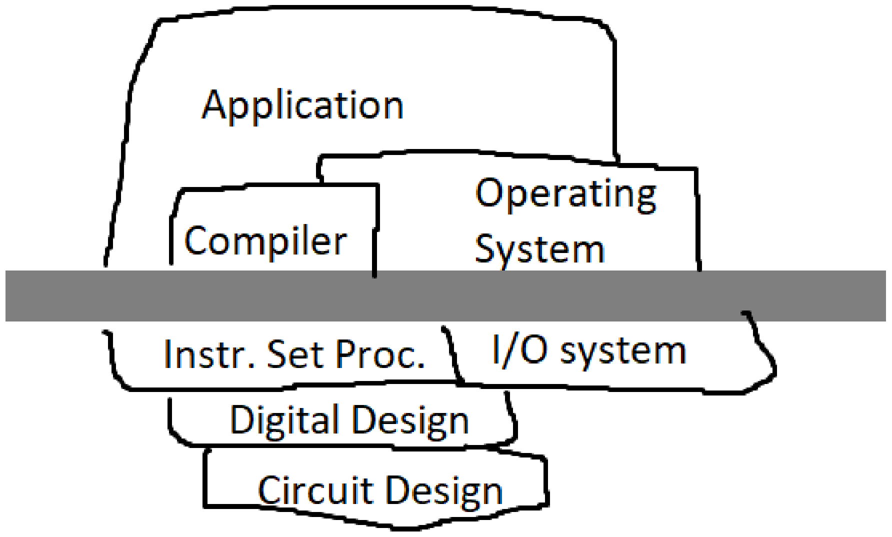
Release Date: 04/27/2020 Due Date: 05/10/2020 – 6:30pm

CSE 141 – Homework 3 (10 pts): ISA Basics

1. **(1 pt)** Explain what an instruction set architecture (ISA) is. Give an example of an existing ISA today.

   The agreed-upon interface between all the software that runs on the machine and the hardware that executes it. An example is MIPS.

2. **(2 pts)** Draw the traditional computing stack starting from application layer down to circuit design layer.



3. **(1 pt)** Explain what microarchitecture is. How is it different from instruction set architecture? Also give an example of microarchitecture.

   Microarchitecture describes how an ISA works on hardware, and includes instruction cycles, pipelining, etc. An example is an intel core i7 cpu, which can run x86 ISA.

   It is different from an ISA, where an ISA is the set of instructions supported.

4. **(1 pt)** What are some key ISA design decisions?

   How many operations, which ones to support, how many operands, their location and types, and size of instruction format

5. **(1 pt)** Your architecture supports 16 instructions and 16 registers (0-15). You have fixed width instructions which are 16 bits. How many register operands can you specify (explicitly) in an add instruction?

   <= 3

6. **(2 pts)** Which of the following statements are true (select all)?
   a. Load-store ISAs require more code relative to stack ISAs.
   b. Stack ISAs require more accesses to memory than Reg-mem ISAs.

   a) Is true

7. **(2 pts)** Take a look at MIPS ISA. Give three instruction types in MIPS. For each instruction type, specify the instruction format and give an example instruction.

   R-type – 6 bits opcode, 5 bits rs, 5 bits rt, 5 bits rd, 5 bits sa, 6 buts funct

   An example would be add

   I-type – 6 bits opcode, 5 bits rs, 5 bits rd, 16 bits immediate

   An example would be beq

   J-type – 6 bits opcode, 26 bits target

   An example would be jal

---

8. **(Bonus 1 pt).** When learning C programming for the first time, one of the most common mistakes people make is use of pointers in swap function. The classic example is:

```
/* This is wrong */
void swap(int a, int b) {
    int temp;

    temp = a;
    a = b;
    b = temp;
}


/* This is correct */
void swap(int *a, int *b) {
    int temp;

    temp = *a;
    *a = *b;
    *b = temp;
}
```

One of the reasons we study computer architecture is to get a deeper understanding of how programs execute on machines. The immediate effect you will see is better understanding of the execution model of C programs. Our goal here is to explain why the first version of the code snippet is incorrect and why second one is correct. Give your explanation however you would like. We recommend translating the code to some type of assembly language format, whether it is MIPS ISA some virtual instruction set (you can come up with simple one yourself as well).

It is incorrect as the values aren't actually swapping. That function puts the arguments in its own stack frame, and swaps those, where our passed in parameters aren't. The second one

actually takes the address in memory of where the values are supposed to swap, and swaps them.

9. **(Bonus 1 pt).** Here is another example to demonstrate the effect of computer architecture knowledge on your understanding of source programs. Traditionally, removing a node from a linked list is implemented in C like so:

```
struct list mylist {
    int data;
    struct list *next;
};

struct list *remove_ver1(struct list *head, int data)
{
    struct list *prev;
    struct list *curr;

    prev = NULL;
    curr = head;
    while (curr->data != data) {
        prev = curr;
        curr = curr->next;
    }

    if (!prev)
        head = curr->next;
    else
        prev->next = curr->next;

    return head;
}
```

However, here is another version that performs the same functionality.

```
void remove_ver2(struct list **head, int data)
{
    struct list *entry;

    while ((*head)->data != data)
        head = &(*head)->next;
    entry = *head;
    *head = (*head)->next;
    free(entry);
}
```

Second version eliminates the need for having the extra if statement that first version has. It may seem cryptic at first, but having architectural knowledge will help with understanding this code. Explain how each version can be translated to an assembly language of your choice, as you did in the previous question, and explain why second version is also correct.

SUBMISSION INSTRUCTIONS (READ CAREFULLY):

1. Please type your responses in a separate document and submit the PDF file to gradescope. ONLY typed PDF documents will be accepted as a valid submission. However, if there are any questions that ask you to draw a

diagram, those can be hand-drawn, but must be attached to the final PDF file. Please do not submit a separate file for hand-drawn diagrams. Only one, final PDF file will be accepted.

2. If you have any questions about the homework problems, please post your question on piazza.

3. No late submissions allowed!