EE/CSE469 Review Problem 0

- As you wait for class to start, answer the following question:
  - What is important in a computer? What features do you look for when buying one?

MHz/GHz
Memory size/Type
Battery life/Power efficiency
Size
I/O: USB/HD/TV/Screen
Price
Durability
EE/CSE 469: Computer Design and Organization

Professor Scott Hauck, 307Q, hauck@uw.edu
Office hours: email w/schedule

TA: Hongwei Ding (hwding@uw.edu)
    Ailing Piao (alpiao@uw.edu)

Office hours: (EEB-361) up-to-date times on website

Book:


Grading (approximate):

20% - Homeworks  35% - Design Project  20% - Midterm  25% - Final Exam
Prerequisites

Basic Logic Design and Boolean Algebra
  AND, OR, NAND, NOR gates
  Boolean Algebra
  D flip-flops, registers, and memories
  Binary numbers, 2’s complement, negation, overflows

Verilog

C/C++/Java programming

If you don’t know this material, **DO NOT TAKE THE CLASS**

If you don’t remember this material, **REVIEW NOW**.
Joint Work Policy

The processor design and homeworks will be done in groups of 1-2.
Groups may not collaborate on the specifics of homework or on the projects.
Let me know if you need help forming groups.

OK:
- Studying together for exams
- Discussing lectures or readings
- Talking about general approaches
- Help in debugging, CAD tools peculiarities, etc.

Not OK:
- Developing a design between groups
- Implementing the CPU between groups
- Checking homework answers between groups

Violation of these rules is at minimum:
- Loss of twice the points of that assignment.
- Report of Academic Misconduct to Dean’s Level.
- Potentially fail class, be expelled from UW.
Late Policy

All assignments due by the end of the class period

Late penalties;
-10% for the first 24 hours
-20% for the second 24 hours (total –30%)
-30% for the third 24 hours (total –60%)
-40% for all additional hours (total –100%)
Computer Architecture

Readings: 1.1-1.4

Instruction Set Architecture

Interaction between hardware and software
Hardware sets realities, requirements
Area, power, performance
Software places demands on hardware
Processor only as good as software it runs
Implementing Software – The Compilation Process

*/ Swap the ith and (i+1)th element of an array */
swap(int v[], int k) {
    int temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}

SWAP:
LSL   X9, X1, #3
ADD   X9, X0, X9   // Compute address of v[k]
LDUR  X10, [X9, #0] // get v[k]
LDUR  X11, [X9, #8] // get v[k+1]
STUR  X11, [X9,#0] // save new value to v[k]
STUR  X10, [X9, #8] // save new value to v[k+1]
BR    X30          // return from subroutine

CPU  Memory

Memory contents:

11010011011 00000 000011 000011 01001
10001011000 01001 000000 00000 01001
11111000010 00000000 00 01001 01010
11111000010 000001000 00 01001 01011
11111000000 00000000 00 01001 01011
11111000000 000001000 00 01001 01010
11010110000 00000 000000 00000 11110
Computer Organization

Five classic components

- **Computer**
  - Processor
    - Control
    - Datapath
  - Memory
    - Instruction
    - Date
  - Devices
    - Input
    - Output

Memory: Store instructions, data
Datapath: Perform operations (Add, subtract, ...)
Control: Orchestrate operations (who does what when)
Input: Get information from the outside world
Output: Provide results
Execution cycle

- **Instruction Fetch**: Obtain instruction from program storage
- **Instruction Decode**: Determine required actions and instruction size
- **Operand Fetch**: Locate and obtain operand data
- **Execute**: Compute result value or status
- **Result Store**: Deposit results in storage for later use
- **Next Instruction**: Determine successor instruction
Review Problem 1

- Programming languages have many instructions, but they fall under a few basic types. One is arithmetic (+, -, *, /, etc). What are the others?

  Logical (shift, & , |)

  Control flow (if ... then ... else, goto, procedure call)

  Iteration (for, while, ...)

  Data structures (variables, malloc/new)

  Libraries: I/O printing, user input
Grade vs. Lecture Attendance

% Lectures Attended

Grade
Assembly Language

Readings: 2.1-2.7, 2.9-2.10, 2.14
Green reference card

Assembly language
Simple, regular instructions – building blocks of C, Java & other languages
Typically one-to-one mapping to machine language

Our goal
Understand the basics of assembly language
Help figure out what the processor needs to be able to do

Not our goal to teach complete assembly/machine language programming
Floating point
Procedure calls
Stacks & local variables
Aside: C/C++ Primer

```c
struct coord { int x, y; } ;  /* Declares a type */
struct coord start;          /* Object with two slots, x and y */
start.x = 1;                  /* For objects "." accesses a slot */
struct coord *myLoc;         /* "*" is a pointer to objects */
myLoc = &start;              /* "&" returns thing's location */
myLoc->y = 2;                /* "->" is "*" plus "." */
```

```
myLoc: 4000
```

```
int scores[8];
scores[1]=5;
int *index = scores;
index++;
(*index)++;
index = &(scores[3]);   /* Points to scores[0] */
*index = 9;             /* Next scores location */
```

```
Scores: 0 1 2 3 4 5 6 7
```

```
Start:
```

```
x 1
y 2
```

```
index: 4000
```
ARM Assembly Language

The basic instructions have four components:
  Operator name
  Destination
  1st operand
  2nd operand

ADD <dst>, <src1>, <src2> // <dst> = <src1> + <src2>
SUB <dst>, <src1>, <src2> // <dst> = <src1> - <src2>

Simple format: easy to implement in hardware

More complex: A = B + C + D - E

ADD t1, B, C // t1 = B + C
SUB t2, D, E // t2 = D - E
ADD A, t1, t2
Operands & Storage

For speed, CPU has 32 general-purpose registers for storing most operands
For capacity, computer has large memory (multi-GB)

Load/store operation moves information between registers and main memory
All other operations work on registers
## Registers

32x 64-bit registers for operands

<table>
<thead>
<tr>
<th>Register</th>
<th>Function</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>X0-X7</td>
<td>Function arguments/Results</td>
<td></td>
</tr>
<tr>
<td>X8</td>
<td>Result, if a pointer</td>
<td></td>
</tr>
<tr>
<td>X9-X15</td>
<td>Volatile Temporaries</td>
<td>Not saved on call</td>
</tr>
<tr>
<td>X16-X17</td>
<td>Linker scratch registers</td>
<td>Don’t use them</td>
</tr>
<tr>
<td>X18</td>
<td>Platform register</td>
<td>Don’t use this</td>
</tr>
<tr>
<td>X19-X27</td>
<td>Temporaries (saved across calls)</td>
<td>Saved on call</td>
</tr>
<tr>
<td>X28</td>
<td>Stack Pointer</td>
<td></td>
</tr>
<tr>
<td>X29</td>
<td>Frame Pointer</td>
<td></td>
</tr>
<tr>
<td>X30</td>
<td>Return Address</td>
<td></td>
</tr>
<tr>
<td>X31</td>
<td>Always 0</td>
<td>No-op on write</td>
</tr>
</tbody>
</table>