## CS311 - COMPUTER SYSTEMS I

**Homework #5** - Due: Wednesday, October 31, at the start of class

**Emphasis**: CPU Architectural Alternatives

1. Suppose the (byte-addressable) memory of a computer contains the following values at some point in time. (All addresses and contents are given in hexadecimal).

| Address | Contents | Address | Contents | Address | Contents | Address | Contents |
|---|---|---|---|---|---|---|---|
| 1000 | 00 | 1001 | 11 | 1002 | 22 | 1003 | 33 |
| 1004 | 44 | 1005 | 55 | 1006 | 66 | 1007 | 77 |

   a. What is the word at address 1004 if the computer uses little-endian representation for 32-bit integers?

   b. What would your answer to (a) be if it used big-endian representation?

   c. What is the half-word at address 1002 if the computer uses little-endian representation for 32-bit integers?

   d. What would your answer (c) be if it used big-endian representation?

   e. Is the word address 1002 valid? If so, what is the value of the word at this address if the computer uses little-endian representation?

2. Suppose that the memory of a single accumulator computer contains the values shown below (all values given in hexadecimal), Assume that memory is byte-addressable and that the size of a word is 4 bytes.

| Address | Contents |
|---|---|
| 100 | 400 |
| ... | |
| 400 | 600 |
| ... | |
| 500 | 900 |
| ... | |
| 600 | 700 |
| ... | |
| 700 | 800 |
| ... | |
| 800 | 200 |
| ... | |

   and the instruction being executed is `LOAD WORD` *`some mode`* `100`

   what value is loaded into the accumulator if the addressing mode of the instruction is

   a. Immediate

   b. Direct

   c. Indirect

   d. Indexed with the index register containing 100

3. Write the shortest possible assembly-language programs to evaluate the following assignment statement for each of the listed architectures. (Assume standard operator precedence rules).

   • Assume the variables are stored in memory.

   • You may **not** alter the value of any variable other than X (and temporaries if you use them).

   • For some architectures you will have to use one or more temporary variables, which you may call T1, T2 etc.

   • For the load-store machine, refer to the registers as R1, R2, etc. You will not need to use registers for the other machines.

   • Be careful about the order of operands for subtraction and division!

```
X = (A + B * C) / (D - E * F)
```

   a. A zero-address (stack) machine having instructions

| | |
|---|---|
| PUSH variable | meaning push variable onto top of stack |
| POP variable | meaning pop top item from stack into variable |
| ADD | meaning pop two items; push second popped + first |
| SUB | meaning pop two items; push second popped - first |
| MUL | meaning pop two items; push second popped * first |
| DIV | meaning pop two items; push second popped / first |

   b. A load-store machine having instructions

| | |
|---|---|
| LOAD register, variable | meaning register ← variable |
| STORE register, variable | meaning variable ← register |
| ADD dest-reg, source1-reg, source2-reg | meaning dest-reg ← source1-reg + source2-reg |
| SUB dest-reg, source1-reg, source2-reg | meaning dest-reg ← source1-reg - source2-reg |
| MUL dest-reg, source1-reg, source2-reg | meaning dest-reg ← source1-reg * source2-reg |
| DIV dest-reg, source1-reg, source2-reg | meaning dest-reg ← source1-reg / source2-reg |

   c. A one-address/one-accumulator machine having instructions

| | |
|---|---|
| LOAD variable | meaning AC ← variable |
| STORE variable | meaning variable ← AC |
| ADD variable | meaning AC ← AC + variable |
| SUB variable | meaning AC ← AC - variable |
| MUL variable | meaning AC ← AC * variable |
| DIV variable | meaning AC ← AC / variable |

   d. A two-address machine having instructions

| | |
|---|---|
| MOV destination, source | meaning destination ← source |
| ADD destination, source | meaning destination ← destination + source |
| SUB destination, source | meaning destination ← destination - source |
| MUL destination, source | meaning destination ← destination * source |
| DIV destination, source | meaning destination ← destination / source |

   Note: the program will be shorter if you use X instead of a temporary when calculating the numerator of the expression, and take advantage of addition being commutative

   e. A three-address machine having instructions

| | |
|---|---|
| MOV destination, source | meaning destination ← source |
| ADD destination, source1, source2 | meaning destination ← source2 + source1 |
| SUB destination, source1, source2 | meaning destination ← source2 - source1 |
| MUL destination, source1, source2 | meaning destination ← source2 * source1 |
| DIV destination, source1, source2 | meaning destination ← source2 / source1 |

4. a. Calculate the total length of each program you wrote for the preceeding problem, under the following assumptions:

   - All instructions on the load-store machine require 32 bits
   - On the other machines, an op-code requires 8 bits, and each memory address requires 24 bits. (Thus, a stack machine instruction takes either 32 bits (PUSH, POP) or 8 bits (ADD etc.); a one-address machine instruction takes 32 bits; a two-address machine instruction takes 56 bits; and a three-address machine instruction takes either 56 (MOV) or 80 bits (all others).)
   - All of the named variables (X, A, B, C, D, E, F) are in memory.

   b. Compare the results you just got. Is a simple count of the number of instructions a sufficient way to measure the relative length of a program?

5. Suppose the two-address and three-address machines allowed operands to be either in memory or in registers, and that a register operand requires only 4 bits in the instruction.

   a. Assuming that the named variables are in memory and that registers are used for temporaries, what are the new lengths for the programs for these machines?

   b. For the two-address machine, can you write a shorter program (in terms of total bits) by using an additional instruction? (Hint: use a register instead of X as a temporary when evaluating the numerator.)