CDA 3103 Computer Organization
Homework #7 Solution Set

1 Problems

1. Write a MARIE assembly program for the following algorithm where the subroutine takes two numbers and returns their product. Your assembly program needs to implement and use such a subroutine.

```
X = 3;
Y = 4;
X = mult(X, Y);
```

Answer: Assume X and Y are unsigned positive numbers. Two parameters are stored in locations X and Y, and the return value is stored in rtn.

```
          Load    X
          Store   arg1
          Load    Y
          Store   arg2
          JnS     mult      / Call the subroutine mult
          Load    rtn
          Store   X
          Halt
mult,     Hex     0
          Clear
          Store   rtn
Loop      Load    arg2
          Skipcond  800      / Check if [AC] > 0
          JumpI   mult       / Return to the main program
          Subt    One
          Store   arg2
          Load    rtn
          Add     arg1
          Store   rtn
          Jump    Loop
arg1,     Hex     0
arg2,     Hex     0
rtn,      Hex     0
X,        Hex     3
Y,        Hex     4
ONE,      Hex     1
```

2. Write a MARIE assembly program to compute the quotient of $\frac{X}{Y}$ for two given numbers X and Y, and store the result in Z. A quotient is the result of dividing one number by another. For example, if $X = 9$ and $Y = 3$, the quotient of $\frac{X}{Y}$ is 3. Or if $X = 11$ and $Y = 3$, the quotient of $\frac{X}{Y}$ is still 3.

Answer: Assume X and Y are unsigned positive numbers. Let Z be the memory location storing the quotient, One the memory location storing the constant 1.

```
Loop,    Load       X
         Subt       Y
         SkipCond   000    / If X-Y < 0, terminate.
         Jump       Inc
End,     Halt
Inc,     Store      X      / X = X-Y
         Load       Z
         Add        One
         Store      Z      / Z = Z+1
         Jump       Loop
X,       Dec        11
Y,       Dec        3
Z,       Dec        0
One,     Dec        1
```

3. Complete the following problems in the exercises section at the end of Chapter 5.

   2. Show how the following values would be stored by byte-addressable machines with 32-bit words, using little endian and then big endian format. Assume each value starts at address 10_{16}. Draw a diagram of memory for each, placing the appropriate values in the correct (and labeled) memory locations.

      a. $456789A1_{16}$
      b. $0000058A_{16}$
      c. 14148888_{16}

Ans.

a.

Address ⟶	**10_{16}**	**11_{16}**	**12_{16}**	**13_{16}**
Big Endian	45	67	89	A1
Little Endian	A1	89	67	45

b.

Address ⟶	**10_{16}**	**11_{16}**	**12_{16}**	**13_{16}**
Big Endian	00	00	05	8A
Little Endian	8A	05	00	00

c.

Address ⟶	**10_{16}**	**11_{16}**	**12_{16}**	**13_{16}**
Big Endian	14	14	88	88
Little Endian	88	88	14	14

9 There are reasons for machine designers to want all instructions to be the same length. Why is this not a good idea on a stack machine?

Answer: The only instructions on a stack machine that need to address memory are push and pop. So an operand field is required, which implies the instruction field must be divided into an opcode and an operand. However, the other instructions need not access memory, and can thus consist of only the opcode. To make them all the same length, these instructions would need to be "artificially lengthened", which can cause waste of memory for storing program instructions.

13 Convert the following expressions from reverse Polish notation to the infix notation.

a) $12\ 8\ 3\ 1\ +\ -/$ $12/(8-(3+1))$
b) $5\ 2+2\times 1+2\times$ $(((5+2)\times 2+1)\times 2)$
c) $3\ 5\ 7+2\ 1-\times 1++$ $3+((5+7)\times(2-1)+1)$

15 Explain how a stack is used to evaluate the RPN expressions from Exercise 13.

a) 12 8 3 1 + – /
12, 8, 3 and 1 are pushed onto the stack. The plus operator adds 3 + 1, pops them from the stack, and pushes 4. The minus operator takes 8 – 4, pops them from the stack, and pushes 4. The divide operator takes 12/4, pops them from the stack, and pushes 3.
b) 5 2 + 2 × 1 + 2 ×
5 and 2 are pushed onto the stack. The plus operator adds 5+2, pops them from the stack, and pushes 7. 2 is pushed, and then the times operator takes 7 X 2, pops them from the stack, and pushes 14. 1 is pushed, and then the plus operator adds 14 + 1, pops them, and pushes 15. 2 is pushed, and then the times operators multiples 15 by 2, pops them, and pushes 30.
c) 3 5 7 + 2 1 – × 1 + +
3, 5 and 7 are pushed. The plus operator adds 5 + 7, pops them, and pushes 12. Then 2 and 1 are pushed. The minus operator subtracts 1 from 2, pops them, and pushes 1. The times operator multiplies 12 by 1, pops them, and pushes 12. The 1 is pushed, then the plus operator adds 12 plus 1, pops them, and pushes 13. The plus operator them adds 3 + 13, pops them, and pushes 16.

16

a) Write the following expression in postfix (Reverse Polish) notation. Remember the rules of precedence for arithmetic operators!

$$X = \frac{A - B + C \times (D \times E - F)}{G + H \times K}$$

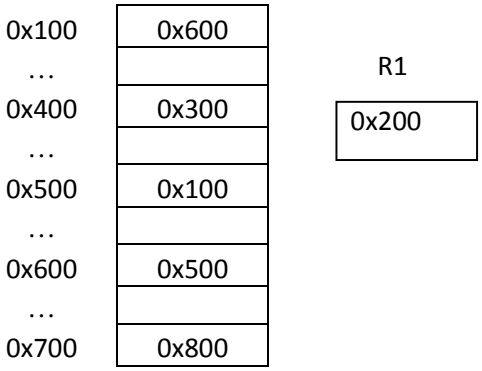
The RPN expression is $A\ B - C\ D\ E \times F - \times + G\ H\ K \times +/$

b) Write a program to evaluate the above arithmetic statement using a stack organized computer with zero-address instructions (so only pop and push can access memory).

```
Push A
Push B
Subtract
Push C
Push D
Push E
Mult
Push F
Subtract
Mult
Add
Push G
Push H
```

```
Push K
Mult
Add
Div
Pop X
```

22 Suppose we have the instruction Load 500. Given memory and register R1 contain the values below.



and assuming R1 is implied in the indexed addressing mode, determine the actual value loaded into the accumulator and fill in the table below:

Mode	Value Loaded into AC
Immediate	
Direct	
Indirect	
Indexed	

Ans.

Mode	Value
Immediate	500
Direct	100
Indirect	600
Indexed	800

4. Use a few sentences to answer each of the following questions. The answers can be found in section 5.1 to 5.4 in the textbook.

   (a) Explain the difference between register-to-register, register-to-memory, and memory-to-memory instructions.

   (b) What does the term "endian" mean? Why does "endian-ness" matter?

   (c) Which of the following programs would be longer: a program written for a zero-address architecture, a program written for a one-address architecture, or a program written for a two address architecture? Why?

   (d) What are addressing modes?

   The answers to the above questions can be found in seciton 5.1 - 5.4.