**Wk4D**

**Shifting and Instructions**

Part I.

Create an eight-bit word and then perform three of the shifting operations on it.

(For ease of reading, I suggest that the word be broken into two 4-bit pieces separated by a blank space (i.e. xxxx xxxx)

The possible shifting operations include:

* + - Logical Shift Right
    - Logical Shift Left
    - Rotate Right
    - Rotate Left
    - Arithmetic Shift Right
    - Arithmetic Shift Left

Each operation should be performed independently on the original 8-bit word.

**Eight-Bit Word: 0011 1011**

|  |  |  |
| --- | --- | --- |
| Shifting Operation | Original 8-bit Word | Shifted 8-bit Word |
| Logical Shift Left by 2 | **0011 1011** | **1110 1100** |
| Rotate Right by 3 | **0011 1011** | **0110 0111** |
| Arithmetic Shift Left by 1 | **0011 1011** | **0111 0111** |

Part II

Evaluate a short algebraic expression using code with three-operand instructions.  The expression should have a minimum of three operands and 2 operators.  Operands may be alpha or numeric.

*You may only use registers A through F, plus X and T.*

*Registers A through F may not be changed, i.e. their values are fixed.*

*Register T may be used as a temporary register, and Register X must contain the final answer.*

Show the postfix for the expression, and then use a stack to evaluate the expression.

As an example, consider X=A +B + C \* D

The three-operand instructions would be:

   ADD  X, A, B

   MUL  T, C, D

   ADD  X, X, T

The Postfix would be AB+CD\*+

The stack would be:

        PUSH    / A onto the stack

        PUSH    / B onto the stack

 ADD      /pop A & B, add them, and push the answer back on the stack

        PUSH    /C

        PUSH   / D

        MUL   /pop C & D, multiply them, and push the answer back on the stack

        ADD    / pop X & T, add them, and push the answer back on the stack

         POP    /  X off the stack

*It is important to note that operational hierarchy dictates that*

*we first perform all arithmetic inside inner parentheses;*

*then inside outer parentheses;*

*then do multiplication and division operations before addition and subtraction operations.*

**Algebraic Expression: X = (A\*B- C) / (D + E)**

**MUL X, A, B**

**SUB X, X, C**

**ADD T, D, E**

**DIV X, X, T**

**Postfix Expression: X = A B \* C – D E + /**

**PUSH /A onto the stack**

**PUSH /B onto the stack**

**MUL /pop A & B, multiply them, and push the answer back on the stack**

**PUSH /C**

**SUB /pop X and C, subtract them, and push the answer back on the stack**

**PUSH /D**

**PUSH /E**

**ADD /pop D & E, add them, and push the answer back on the stack**

**DIV /pop X & T, divide them, and push the answer back on the stack**

**POP /X off the stack**

Part III

1. Suppose we have the instruction Load 1000. Given that memory and register R1 contain the values below:

|  |  |  |  |
| --- | --- | --- | --- |
| Memory | Memory |  |  |
| Address | Content |  |  |
|  |  |  |  |
| 1000 | 1200 |  |  |
|  |  | Register | Content |
| 1100 | 400 | R1 | 200 |
|  |  |  |  |
| 1200 | 1000 |  |  |
|  |  |  |  |
| 1300 | 1100 |  |  |
|  |  |  |  |
| 1400 | 1300 |  |  |

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 | **1000** |
| Direct | **1200** |
| Indirect | **1000** |
| Indexed | **1000** |

**Immediate – 1000 is loaded into AC.**

**Direct – Value at address 1000 (1200) is loaded into AC, 1200 is loaded into AC.**

**Indirect – Value at address 1000 (1200) is used as direct address, value in address 1200 (1000) is loaded into AC, 1000 is loaded into AC.**

**Indexed – 1000 is added to value in R1 (200) and is used as direct address 1200, value in address 1200 (1000) is loaded in AC, 1000 is loaded into AC.**