Question 3 - Instruction Set 3 – x295++

A. Description of x295++ instruction set architecture (ISA)

Would reducing the number of operands in the instructions of our **x295+** instruction set architecture decrease the number of memory accesses the microprocessor does when fetching, decoding and executing our program?

Would introducing another instruction, namely **`COPY`**, in our instruction set also decrease the number of memory accesses the microprocessor does when fetching. decoding and executing our program?

With the above two questions in mind, we create a third instruction set architecture called **`x295++`**. This instruction set architecture is just like **`x295+`**, except for the following modifications:

1. **Memory model**: Same as **`x295+`**
2. **Registers**
   - **Data type**: Same as **`x295+`**
   - **Number of registers**: Same as **`x295+`**
   - **Size of register**: Same as **`x295+`**
   - **Number (and bit encoding) associated with each register**: Same as **`x295+`**
3. **Instruction set (assembly instructions and machine code instructions)**
   - **Maximum number of instructions**: Same as **`x295+`**
   - **Memory addressing mode**: Same as **`x295+`**
   - **Operand Model**:
     - *2-operand* model - an operand can be a register or a memory address.
   - **Assembly instructions**:
     - Note: Even though there is a maximum of 16 instructions in this instruction set, in this assignment we shall only define and use a subset of these 16 instructions:

       List of this subset of instructions with their format (syntax) and meaning:

▪ `ADD rA, rC`	Meaning: `rC <- rA + rC`
▪ `SUB rA, rC`	Meaning: `rC <- rA - rC`
▪ `MUL rA, rC`	Meaning: `rC <- rA * rC`

- `COPY rA, rC` Meaning: `rC <- rA`
- `LOAD a, rC` Meaning: `rC <- M[a]`
- `STORE rA, c` Meaning: `M[c] <- rA`

- **Machine code instructions**
  - **Format:**

Format 1:

Opcode (4 bits)	Dest (3 bits)	Src (3 bits)	XXXXXX

The Format 1 above replaces the Format 1 defined for **`x295+`**.

This format is made of 16 bits (1 word) and it can be used to form the machine code instructions corresponding to the ADD, SUB, MUL and COPY assembly instructions.

About field XXXXXX in the above format: These X's can be either 0's or 1's – they are not used by the microprocessor **.**

`x295++` uses Format 2 and Format 3 defined for **`x295+`**.

- **Opcode encoding (bit pattern)**:

Opcode (instruction)	**Encoding Bit pattern (4 bits)**
padding	0000
ADD	0001
SUB	0010
MUL	0011
COPY	**1001**
LOAD	1010
STORE	1011
…	…

B. Compiling and assembling a C program using our x295++ instruction set

1. First, download the file **`A7_Q3_Tables.pdf (or docx)`** from our course web site (below Assignment 7) and open it. It contains two tables called **Table 1** and **Table 2**.

2. Then, "compile" (by hand) the C program (in the left column) into a **`x295++`** assembly program using the assembly instructions defined in our **`x295++`** instruction set and write it in the middle column of **Table 1** found in the file **`A7_Q3_Tables.pdf (or docx`**) you downloaded.

3. Lastly, "assemble" (by hand) our **x295++** assembly program into a **x295++** machine code program using the machine code instructions defined in our **x295++** instruction set and write it in the right column of **Table 1** found in the file **A7_Q3_Tables.pdf (or docx**) you downloaded.

C. Evaluating our x295++ instruction set

The next step is to evaluate our **x295++** instruction set by executing (hand tracing) our **x295++** assembly program (or its corresponding **x295++** machine code program although it is easier to use the assembly program) and using the criteria called ***memory traffic***, by counting the number of memory accesses our **x295++** assembly program makes during its execution. In other words, we count how many time the execution of our **x295++** assembly program required a word (16 bits) to be read from or written to memory.

Perform this evaluation …

1. By completing the left column of **Table 2** found in the file **A7_Q3_Tables.pdf (or docx**) you downloaded, adding the assembly instructions under "**Assembly instruction:**" and the machine code instructions under "**Machine code instruction:**".

2. By completing the middle and right columns of **Table 2**, adding your memory access count as well as explaining how you obtain this count and lastly …

3. By totaling your counts in the last row of the table. Don't forget the units!

Challenge Question - Instruction Set 3 – x295++

1. Would modifying or simplifying our test C program improve the performance of the resulting **x295++** executable? Would our modified C program lead to an **x295++** executable with fewer instructions? Would our modified C program lead to an **x295++** executable performing fewer memory accesses?

2. Support your claim ("it will" or "it will not") by completing the **Table 1 – Challenge** and the **Table 2 – Challenge** found in the downloaded file **A7_Q3_Tables.docx (or pdf)**.

3. In doing so, make sure that your modified/simplified test C program still produce the same result as the original one.