Question 2 - Instruction Set 2 – x295+

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

Considering the result we obtained from evaluating the **`x295`** instruction set architecture in Question 1, if we wanted to reduce the number of memory accesses, we could improve our instruction set architecture by reducing the size of each of the machine code instructions. Since these are made of an opcode and three operands (memory addresses), we could achieve such size reduction by using registers as operands instead of memory addresses. These registers could contain memory addresses, yet they may require fewer bits to be uniquely identified in an instruction (assembly instruction and machine code instruction).

With this improvement in mind, we create a second 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**: 8 registers
   - **Size of register**: Each register has a length of 16 bits, i.e., a word size.
   - **Number identifying each register**:

Register	Number
`r0`	0
`r1`	1
`r2`	2
`r3`	3
`r4`	4
`r5`	5
`r6`	6
`r7`	7

3. **Instruction set (assembly instructions and machine code instructions)**
   - **Maximum number of instructions**: 16 instructions - Same as **`x295`**
   - **Memory addressing mode**: Direct (i.e., absolute) - Same as **`x295`**
   - **Operand Model**:
     - *3-operand* model – an operand can be a register or a memory address.

- **Assembly instructions**
  - 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, rB, rC` Meaning: `rC <- rA + rB`
    - `SUB rA, rB, rC` Meaning: `rC <- rA - rB`
    - `MUL rA, rB, rC` Meaning: `rC <- rA * rB`
    - `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)	Src1 (3 bits)	Src2 (3 bits)	XXX

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 and MUL assembly instructions.

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

Format 2:

Opcode (4 bits)	Dest (3 bits)	XXX	XXXXXX	padding	Memory address of Src (12 bits)

This format is made of 32 bits (2 words) and it can be used to form the machine code instructions corresponding to the LOAD assembly instruction.

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

Format 3:

Opcode (4 bits)	XXX	Src (3 bits)	XXXXXX	padding	Memory address of Dest (12 bits)

This format is made of 32 bits (2 words) and it can be used to form the machine code instructions corresponding to the STORE assembly instruction

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

Decoding the opcode for a LOAD instruction will inform the microprocessor where to find the field **Dest** and the field **Src** in the machine code instruction itself (**Dest** -> the first field after the opcode in the first word of the machine code instruction and **Src** -> the memory address located in the least significant 12 bits of the second word composing this machine code instruction). The microprocessor will also know to ignore all other bits (fields with X's and padding) in the machine code instruction.

Decoding the opcode for a STORE instruction will inform the microprocessor where to find the field **Src** and the field **Dest** in the machine code instruction itself (**Src** -> the second field after the opcode in the first word of the machine code instruction and **Dest** -> the memory address located in the least significant 12 bits of the second word composing this machine code instruction). The microprocessor will also know to ignore all other bits (fields with X's and padding field) in the machine code instruction.

ISA Design Guidelines:

Notice that, when creating formats to encode an instruction set (IS), we created the least number of formats as possible in order to express all our instructions and we positioned their fields that have the same purpose (such as **Opcode**, **Dest** and **Src**) at the same location in as many of the formats as possible. This helps simplify the design of the microprocessor (its datapath).

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

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

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

1. First, download the file **`A7_Q2_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 of **Table 1** found in the file **`A7_Q2_Tables.pdf - or docx`** - you downloaded) 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**.

3. Note that the step of "assembling" (by hand) our **`x295+`** assembly program into a machine code program has already been done for us and can be found in the right column of **Table 1**. Feel free to use this provided **`x295+`** machine code program to guide your "compiling" of the C program into a **`x295+`** assembly program (the above step).

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_Q2_Tables.pdf (or docx)** you downloaded, adding the assembly instructions under "**Assembly 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!