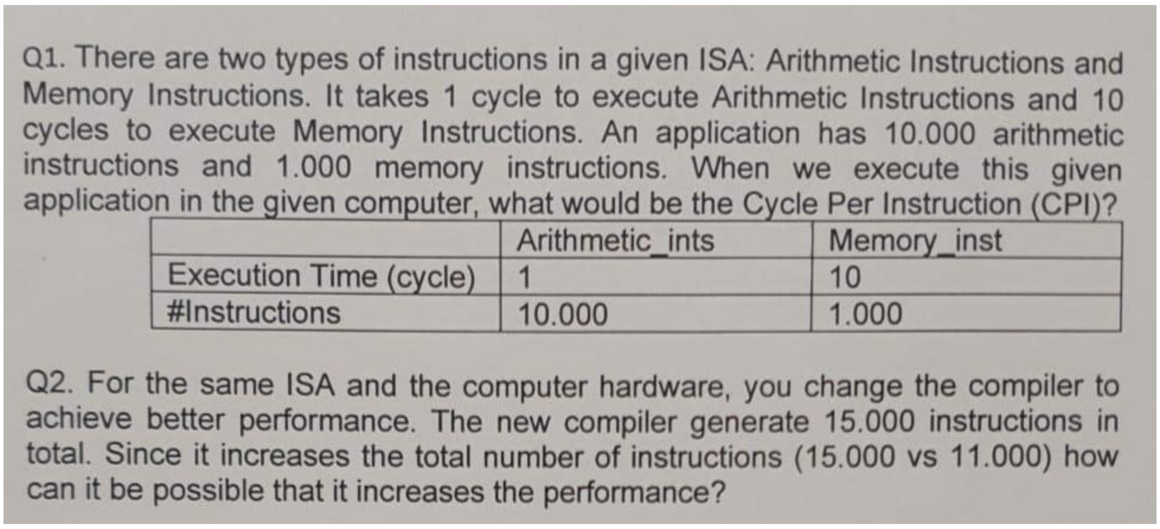
QUIZ-1



Q1. There are two types of instructions in a given ISA: Arithmetic Instructions and Memory Instructions. It takes 1 cycle to execute Arithmetic Instructions and 10 cycles to execute Memory Instructions. An application has 10.000 arithmetic instructions and 1.000 memory instructions. When we execute this given application in the given computer, what would be the Cycle Per Instruction (CPI)?

	Arithmetic_ints	Memory_inst
Execution Time (cycle)	1	10
#Instructions	10.000	1.000

Q2. For the same ISA and the computer hardware, you change the compiler to achieve better performance. The new compiler generate 15.000 instructions in total. Since it increases the total number of instructions (15.000 vs 11.000) how can it be possible that it increases the performance?

Q1-)

$$\text{CPI} = \frac{\text{Total Execution Cycles}}{\text{Total Number of Instructions}}$$

$$\text{Total Execution Cycles} = (\text{Arithmetic instructions} \times \text{Arithmetic time}) + (\text{Memory instructions} \times \text{Memory time})$$

$$\text{Total Execution Cycles} = (10,000 \times 1) + (1,000 \times 10) = 10,000 + 10,000 = 20,000$$

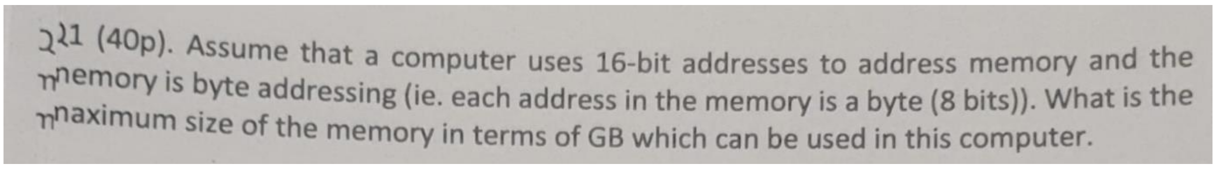
$$\text{Total Number of Instructions} = 10,000 + 1,000 = 11,000$$

$$\text{CPI} = \frac{\text{Total Execution Cycles}}{\text{Total Number of Instructions}} = \frac{20,000}{11,000} \approx 1.82$$

Q2-)

Memory instructions take 10 cycles, which is much more than arithmetic instructions (1 cycle). If the new compiler generates more arithmetic instructions and fewer memory instructions, the average CPI can decrease, improving performance.

QUIZ-2

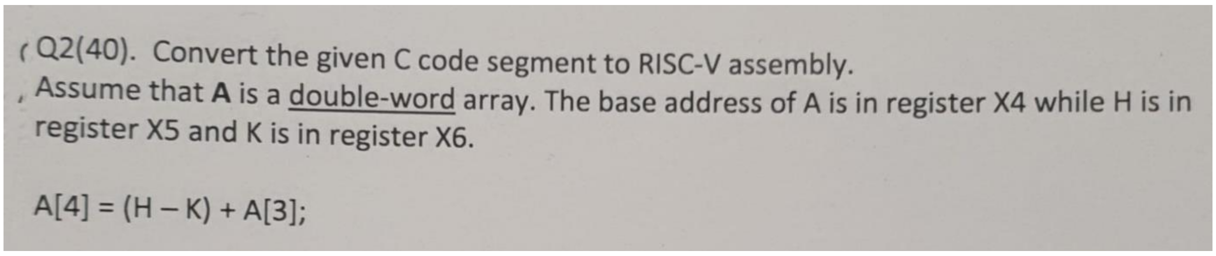


Q1 (40p). Assume that a computer uses 16-bit addresses to address memory and the memory is byte addressing (ie. each address in the memory is a byte (8 bits)). What is the maximum size of the memory in terms of GB which can be used in this computer.

For a 16-bit system, there are 2^{16} possible addresses, and each address represents a 8-bit (a byte).

Maximum Memory Size=Number of Addresses × Byte Size

Maximum Memory Size=2^{16} × 8 bits = 2^{16} x 1 bytes = 64 KB



Q2(40). Convert the given C code segment to RISC-V assembly.
Assume that **A** is a double-word array. The base address of A is in register X4 while H is in register X5 and K is in register X6.

A[4] = (H – K) + A[3];

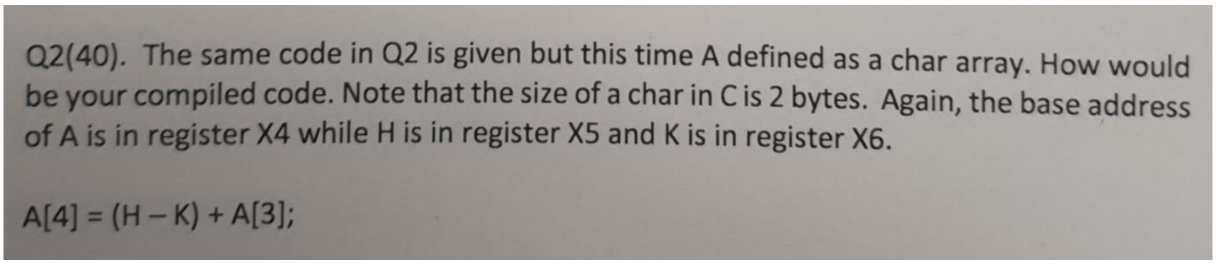
Load A[3] into a temporary register, x10

ld x10, 24(x4) -> Load A[3] into a temporary register, x10, each element is 8 bytes (double-word) so 3 * 8 = 24 bytes offset

sub x11, x5, x6 -> Calculate (H - K)

add x11, x11, x10 -> Add (H - K) to A[3] and store the result in x11

sd t4, 32(x4) # Store the result in A[4], 4 * 8 = 32 bytes offset for A[4]



Q2(40). The same code in Q2 is given but this time A defined as a char array. How would be your compiled code. Note that the size of a char in C is 2 bytes. Again, the base address of A is in register X4 while H is in register X5 and K is in register X6.

A[4] = (H – K) + A[3];

lh x10, 6(x4) -> Load A[3] into x10. Each element is 2 bytes, so the offset for A[3] = 3 * 2 = 6 bytes.

sub x11, x5, x6 -> Calculate (H - K) ->   x11 = H - K

add x11, x11, x10   # Add (H - K) to A[3]   -> x11 = (H - K) + A[3]

sh x11, 8(x4)   # Store the result in A[4]. Offset for A[4] = 4 * 2 = 8 bytes.

QUIZ-3

1. ld X2, 8(X0) - > Load the value from memory at address X0 + 8 (1000 + 8 = 1008) into register X2.

Memory[1008] = 10 → X2 = 10.

2. addi X1, X1, 12 -> Add 12 to the value of X1 and store the result in X1.

X1 = X1 + 12 = 1004 + 12 = 1016.

3. ld X3, 0(X1) -> Load the value from memory at address X1 + 0 (1016 + 0 = 1016) into register X3.

Memory[1016] = 15 → X3 = 15.

4. subi X4, X3, 100 -> Subtract 100 from the value of X3 and store the result in X4.

X4 = X3 - 100 = 15 - 100 = -85.

5. sd X4, 24(X0)-> Store the value of X4 at the memory address X0 + 24 (1000 + 24 = 1024).

X4 = -85 → Memory[1024] = -85.

6. addi X5, X4, 30 -> Add 30 to the value of X4 and store the result in X5.

X5 = X4 + 30 = -85 + 30 = -55.

7. addi X1, X1, 24 -> Add 24 to the value of X1 and store the result in X1.

X1 = X1 + 24 = 1016 + 24 = 1040.

8. sd X5, 0(X1)-> Store the value of X5 at the memory address X1 + 0 (1040 + 0 = 1040).

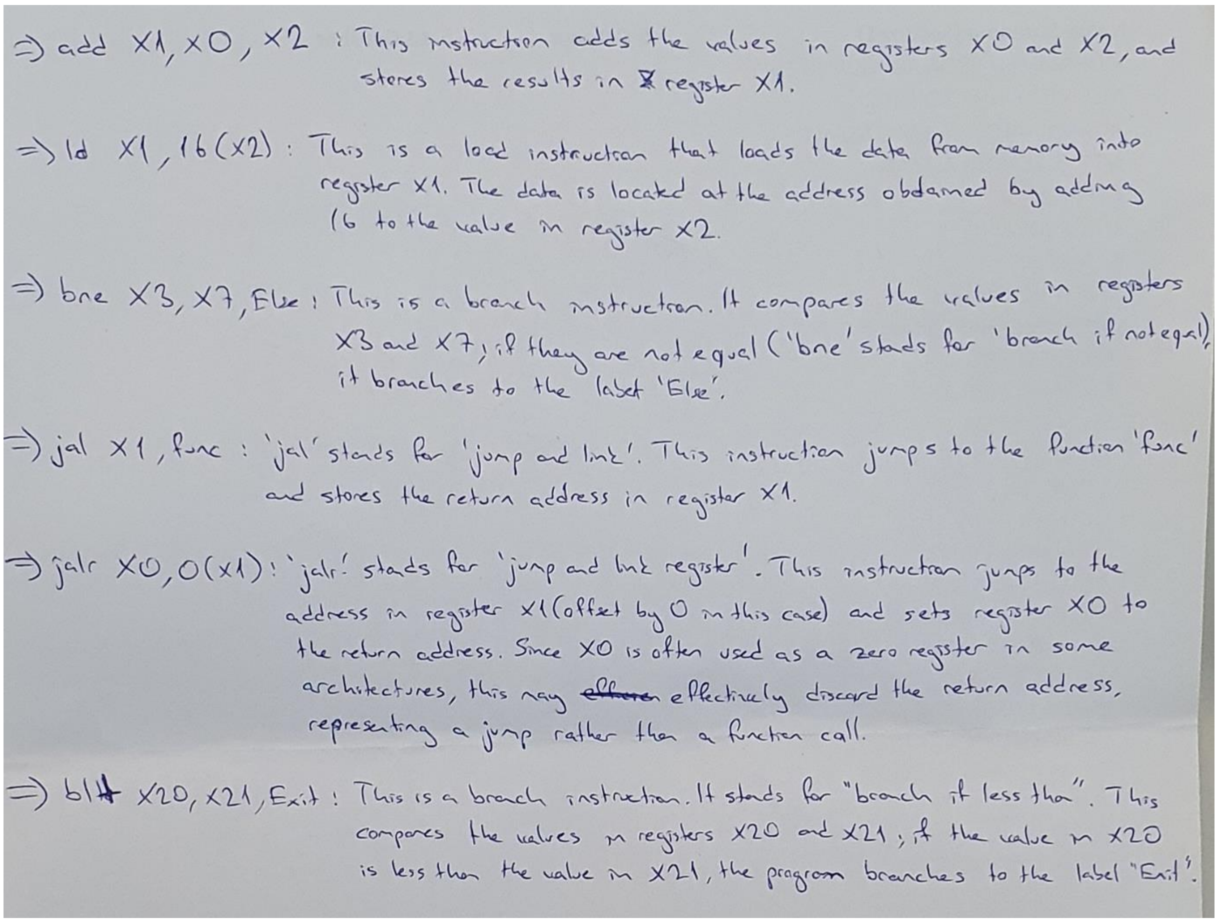
X5 = -55 → Memory[1040] = -55.

Register File	
X0	1000
X1	1040
X2	10
X3	15
X4	-85
X5	-55
X6	80

Memory	
1040	-55
1032	25
1024	-85
1016	15
1008	10
1000	5

QUIZ-4

Explain what the following instructions do with one or two sentences.



⇒ add X1, X0, X2 : This instruction adds the values in registers X0 and X2, and stores the results in ~~X~~ register X1.

⇒ ld X1, 16(X2) : This is a load instruction that loads the data from memory into register X1. The data is located at the address obtained by adding 16 to the value in register X2.

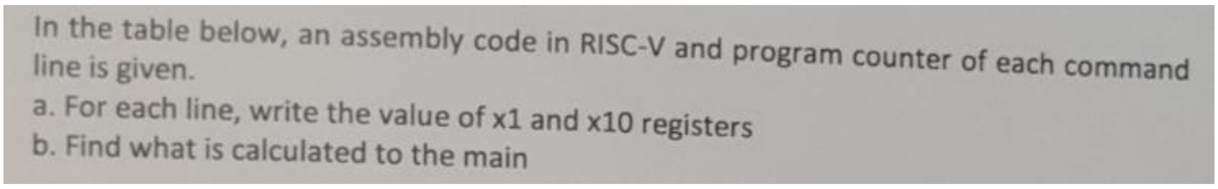
⇒ bne X3, X7, Else : This is a branch instruction. It compares the values in registers X3 and X7; if they are not equal ('bne' stands for 'branch if not equal'), it branches to the label 'Else'.

⇒ jal X1, func : 'jal' stands for 'jump and link'. This instruction jumps to the function 'func' and stores the return address in register X1.

⇒ jalr X0, 0(X1) : 'jalr' stands for 'jump and link register'. This instruction jumps to the address in register X1 (offset by 0 in this case) and sets register X0 to the return address. Since X0 is often used as a zero register in some architectures, this may ~~effecen~~ effectively discard the return address, representing a jump rather than a function call.

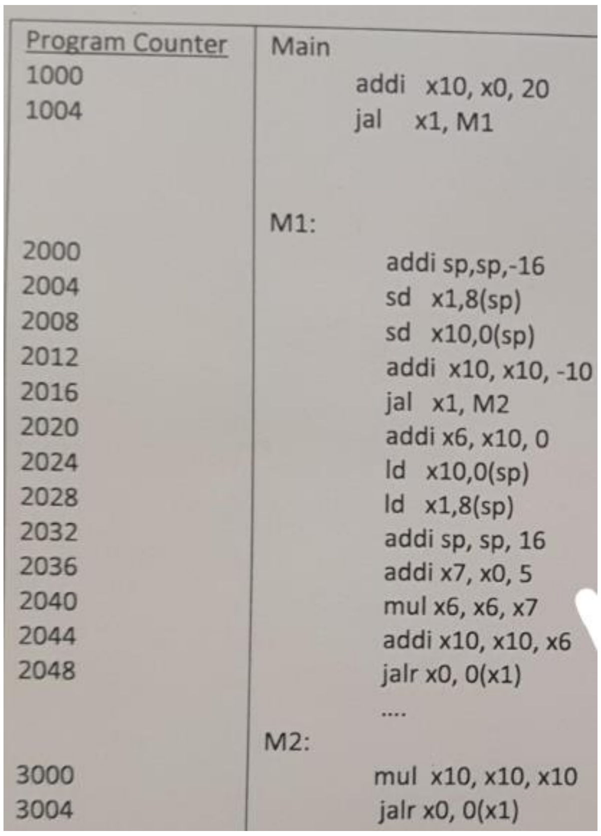
⇒ blt X20, X21, Exit : This is a branch instruction. It stands for "branch if less than". This compares the values in registers X20 and X21; if the value in X20 is less than the value in X21, the program branches to the label "Exit".

QUIZ-5


In the table below, an assembly code in RISC-V and program counter of each command line is given.

a. For each line, write the value of x1 and x10 registers

b. Find what is calculated to the main



Program Counter	Main
1000	addi x10, x0, 20
1004	jal x1, M1
	M1:
2000	addi sp,sp,-16
2004	sd x1,8(sp)
2008	sd x10,0(sp)
2012	addi x10, x10, -10
2016	jal x1, M2
2020	addi x6, x10, 0
2024	ld x10,0(sp)
2028	ld x1,8(sp)
2032	addi sp, sp, 16
2036	addi x7, x0, 5
2040	mul x6, x6, x7
2044	addi x10, x10, x6
2048	jalr x0, 0(x1)
	....
	M2:
3000	mul x10, x10, x10
3004	jalr x0, 0(x1)

Main Function (Program Counter: 1000–1004)

1. **Instruction: addi x10, x0, 20 (PC: 1000)**
   - Add 20 to x0 (which is 0).
   - **Result:** x10 = 20
2. **Instruction: jal x1, M1 (PC: 1004)**
   - Jump to label M1 (PC = 2000) and store the return address (1008) in x1.
   - **Result:** x1 = 1008

M1 Function (Program Counter: 2000–2048)

3. **Instruction: addi sp, sp, -16 (PC: 2000)**
   - Decrease the stack pointer (sp) by 16 to allocate stack space.
   - **Result:** sp = sp - 16
4. **Instruction: sd x1, 8(sp) (PC: 2004)**
   - Store the value of x1 (1008) at memory location sp + 8.
   - **Result:** Memory[sp + 8] = 1008
5. **Instruction: sd x10, 0(sp) (PC: 2008)**
   - Store the value of x10 (20) at memory location sp + 0.

   - **Result:** Memory[sp + 0] = 20
6. **Instruction: addi x10, x10, -10 (PC: 2012)**
   - Subtract 10 from x10.
   - **Result:** x10 = 20 - 10 = 10
7. **Instruction: jal x1, M2 (PC: 2016)**
   - Jump to label M2 (PC = 3000) and store the return address (2020) in x1.
   - **Result:** x1 = 2020

M2 unction (Program Counter: 3000–3004)

8. **Instruction: mul x10, x10, x10 (PC: 3000)**
   - Multiply x10 by itself.
   - **Result:** x10 = 10 * 10 = 100
9. **Instruction: jalr x0, 0(x1) (PC: 3004)**
   - Return to the address stored in x1 (2020).
   - **Result:** Jump back to PC = 2020

Back to M1 Function (Program Counter: 2020–2048)

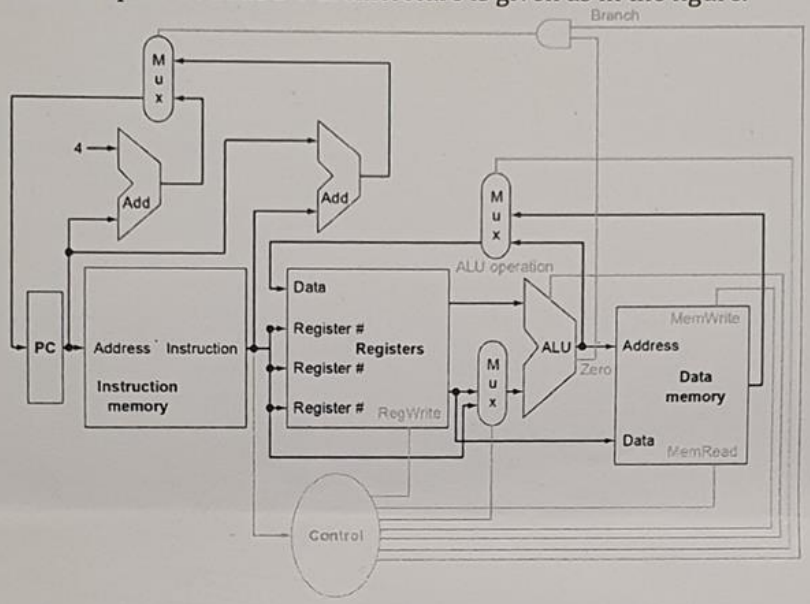
10. **Instruction: addi x6, x10, 0 (PC: 2020)**
    - Copy the value of x10 (100) into x6.
    - **Result:** x6 = 100
11. **Instruction: ld x10, 0(sp) (PC: 2024)**
    - Load the value from memory at sp + 0 (20) into x10.
    - **Result:** x10 = 20
12. **Instruction: ld x1, 8(sp) (PC: 2028)**
    - Load the value from memory at sp + 8 (1008) into x1.
    - **Result:** x1 = 1008
13. **Instruction: addi sp, sp, 16 (PC: 2032)**
    - Restore the stack pointer (sp) by adding 16.
    - **Result:** sp = sp + 16
14. **Instruction: addi x7, x0, 5 (PC: 2036)**
    - Add 5 to x0 (0) and store in x7.
    - **Result:** x7 = 5
15. **Instruction: mul x6, x6, x7 (PC: 2040)**
    - Multiply x6 (100) by x7 (5).
    - **Result:** x6 = 100 * 5 = 500
16. **Instruction: addi x10, x10, x6 (PC: 2044)**
    - Add x6 (500) to x10 (20).
    - **Result:** x10 = 20 + 500 = 520
17. **Instruction: jalr x0, 0(x1) (PC: 2048)**
    - Return to the address stored in x1 (1008).
    - **Result:** Jump back to main.

Final Register Values:

- x1 = 1008 (return address to main).
- x10 = 520.

QUIZ-6

The datapath in a RISC-V architecture is given as in the figure.



ALU control	0000	0001	0010	0110
Function	AND	OR	ADD	SUBTRACT

Write down the value of each signal for given instructions in the table.

Instruction	Branch	MemRead	MemWrite	ALU operation	RegWrite
ADD X1, X2, X3	0	0	0	0010	1
BNE X1, X2, Exit	1	0	0	0110	0
LD X1, 8(X2)	0	1	0	0010	1
SD X1, 4(X5)	0	0	1	0010	0
AND X1, X2, X3	0	0	0	0000	1