1) Write the control signals of the single-bus processor below for the following instructions (be careful with the addressing modes).

a. (mem1) + (mem2) = R1
b. (mem1) + (R1) = (mem2)
c. Conditional Jump N:

(mem1) + (R1) = mem2

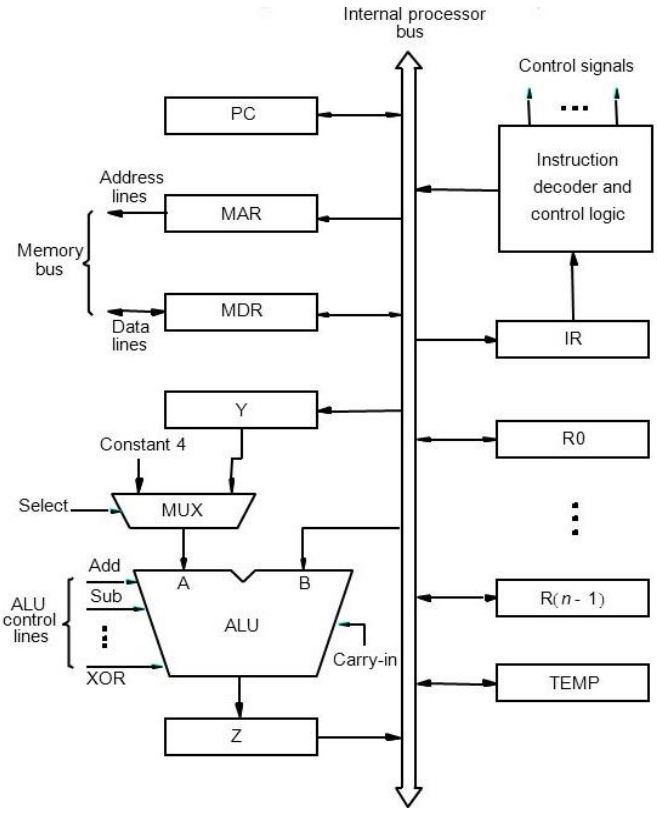
d. Addressing Mode: Absolute

OPCODE | 200 (PC = 200)

e. Addressing Mode: Relative

OPCODE | 200, (PC = PC + 200)

f. Conditional Jump relative (PC + offset in N and END in ~~N~~)



2) Answer the following questions:

a. Describe "BUS multiplexing".

b. What is ISA? What are the main differences between instruction sets such as Intel x86, ARM, RISC-V and MIPS?

3) In the following assembly code, how many times instruction memory, data memory and register bank have been accessed? Fill the table and write a full description about every line of code.

```
LOAD R0, M40        (Loads the content of M40 in R1)
LOAD R1, M41        (Loads the content of M41 in R2)
ADDI R2, R0, R1     (Add the contents of R1 and R0 -> R2)
STORE M42, R2               (Stores the contents of R2 in M42)
HALT
```

Code lines	Register bank	Instruction Memory	Data Memory
Line 1			
Line 2			
Line 3			
Line 4			
Line 5			

4) Write -1.75 in IEEE-754 standard (both 32 bits and 64 bits).

5) Write an equivalent C code for the following RISC-V assembly code. Define variables f, g, h, i, j as x5, x6, x7, x27, x29 registers. The base address of A and B arrays are stored in registers x11, x12.

```
slli  x30, x5, 3          // x30 =
add   x30, x10, x30       // x30 =
slli  x31, x6, 3          // x31 =
add   x31, x11, x31       // x31 =
ld    x5, 0(x30)          // f =
addi  x12, x30, 8         //
ld    x30, 0(x12)         //
add   x30, x30, x5        //
sd    x30, 0(x31)         //
```

6 – Bonus point) Write RISC-V assembly code for the following equation.

$$S = \sum_{k=1}^{n} k^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2$$

If you have any questions regarding this assignment, feel free to contact us.

Please submit your homework, simulations and projects in the following format:

Name_StudentNumber_HW1 (BillGates_12345678_HW1)

Good Luck!

RISC-V assembly cheat sheet:

Arithmetic Operation

Mnemonic	Instruction	Type	Description
ADD rd, rs1, rs2	Add	R	rd ← rs1 + rs2
SUB rd, rs1, rs2	Subtract	R	rd ← rs1 - rs2
ADDI rd, rs1, imm12	Add immediate	I	rd ← rs1 + imm12
SLT rd, rs1, rs2	Set less than	R	rd ← rs1 < rs2 ? 1 : 0
SLTI rd, rs1, imm12	Set less than immediate	I	rd ← rs1 < imm12 ? 1 : 0
SLTU rd, rs1, rs2	Set less than unsigned	R	rd ← rs1 < rs2 ? 1 : 0
SLTIU rd, rs1, imm12	Set less than immediate unsigned	I	rd ← rs1 < imm12 ? 1 : 0
LUI rd, imm20	Load upper immediate	U	rd ← imm20 << 12
AUIP rd, imm20	Add upper immediate to PC	U	rd ← PC + imm20 << 12

Logical Operations

Mnemonic	Instruction	Type	Description
AND rd, rs1, rs2	AND	R	rd ← rs1 & rs2
OR rd, rs1, rs2	OR	R	rd ← rs1 | rs2
XOR rd, rs1, rs2	XOR	R	rd ← rs1 ^ rs2
ANDI rd, rs1, imm12	AND immediate	I	rd ← rs1 & imm12
ORI rd, rs1, imm12	OR immediate	I	rd ← rs1 | imm12
XORI rd, rs1, imm12	XOR immediate	I	rd ← rs1 ^ imm12
SLL rd, rs1, rs2	Shift left logical	R	rd ← rs1 << rs2
SRL rd, rs1, rs2	Shift right logical	R	rd ← rs1 >> rs2
SRA rd, rs1, rs2	Shift right arithmetic	R	rd ← rs1 >> rs2
SLLI rd, rs1, shamt	Shift left logical immediate	I	rd ← rs1 << shamt
SRLI rd, rs1, shamt	Shift right logical imm.	I	rd ← rs1 >> shamt
SRAI rd, rs1, shamt	Shift right arithmetic immediate	I	rd ← rs1 >> shamt

Mnemonic	Instruction	Base instruction(s)
LI rd, imm12	Load immediate (near)	ADDI rd, zero, imm12
LI rd, imm	Load immediate (far)	LUI rd, imm[31:12] ADDI rd, rd, imm[11:0]
LA rd, sym	Load address (far)	AUIPC rd, sym[31:12] ADDI rd, rd, sym[11:0]
MV rd, rs	Copy register	ADDI rd, rs, 0
NOT rd, rs	One's complement	XORI rd, rs, -1
NEG rd, rs	Two's complement	SUB rd, zero, rs
BGT rs1, rs2, offset	Branch if rs1 > rs2	BLT rs2, rs1, offset
BLE rs1, rs2, offset	Branch if rs1 ≤ rs2	BGE rs2, rs1, offset
BGTU rs1, rs2, offset	Branch if rs1 > rs2 (unsigned)	BLTU rs2, rs1, offset
BLEU rs1, rs2, offset	Branch if rs1 ≤ rs2 (unsigned)	BGEU rs2, rs1, offset
BEQZ rs1, offset	Branch if rs1 = 0	BEQ rs1, zero, offset
BNEZ rs1, offset	Branch if rs1 ≠ 0	BNE rs1, zero, offset
BGEZ rs1, offset	Branch if rs1 ≥ 0	BGE rs1, zero, offset
BLEZ rs1, offset	Branch if rs1 ≤ 0	BGE zero, rs1, offset
BGTZ rs1, offset	Branch if rs1 > 0	BLT zero, rs1, offset
J offset	Unconditional jump	JAL zero, offset
CALL offset12	Call subroutine (near)	JALR ra, ra, offset12
CALL offset	Call subroutine (far)	AUIPC ra, offset[31:12] JALR ra, ra, offset[11:0]
RET	Return from subroutine	JALR zero, 0(ra)
NOP	No operation	ADDI zero, zero, 0

Branching

Mnemonic	Instruction	AVR	AVR Description
`BEQ rs1, rs2, imm12`	Branch equal	`BREQ imm7`	`if Z == 1` `  pc ← pc + imm7`
`BNE rs1, rs2, imm12`	Branch not equal	`BRNE imm7`	`if Z == 0` `  pc ← pc + imm7`
`BGE rs1, rs2, imm12`	Branch greater than or equal	`BRGE imm7`	`if N ^ V == 0` `  pc ← pc + imm7`
`BGEU rs1, rs2, imm12`	Branch greater than or equal unsigned	`BRSH imm7`	`if C == 0` `  pc ← pc + imm12`
`BLT rs1, rs2, imm12`	Branch less than	`BRLT imm7`	`if rs1 < rs2` `  pc ← pc + imm12`
`BLTU rs1, rs2, imm12`	Branch less than unsigned	`BRLO imm7`	`if rs1 < rs2` `  pc ← pc + imm12 << 1`
`JALR zero, imm12(zero)`	Jump	`JMP imm16`	`pc ← imm16`
`JAL zero, imm20`	Relative jump	`RJMP imm12`	`pc ← pc + imm12`
`JALR zero, imm12(rs1)`	Indirect jump	`IJMP`	`pc ← r31:r30`
`JAL rd, imm20`	Jump and link	`RCALL imm12`	`stack ← pc + 2` `pc ← pc + imm12`
`JALR rd, imm12(zero)`	Long call	`CALL imm16`	`stack ← pc + 2` `pc ← imm16`
`JALR rd, imm12(rs1)`	Jump and link register	`RET`	`pc ← stack`
`JALR rd, imm12(rs1)`	Jump and link register	`ICALL`	`stack ← pc + 2` `pc ← r31:r30`

32-bit instruction format