01.How many operands?

Answer: 3 operands [e.g. : a = b + c]

02.Types of operand? (Register based?? Memory based? Mixed?

Answer: Mixed.

03.How many Operations?

Answer: 13 Operations

04.Types of operations? (Arithmetic, logical, branch type?? How many from each category? How many from each category? List the opcodes and respective binary values)

Answer: 5 types

Operation Type	Operations	Register Type
Arithmetic	ADD,SUB	(R-Type)
Logical	AND, OR, XOR	(R-Type)
Data Transfer	ADDI, LW, SW	(I-Type)
Conditional Branch	BEQ,BNE	(I-Type)
Unconditional Jump	J	(J-Type)

05.No. of format of instruction (How many formats would you choose?)

Answer: 3 Formats. R-Type, I-Type, J-Type

06. Describe each of the format (fields and field length)

Answer:

R-Type Format: -

• Op (Opcode) is 4 bits

• RS, RT, RD are 3 bits

• OP is an operation code or opcode that selects a specific operation

• RS is the source register and RT is second source registers

• RD is the destination register

• SA is 1 bit shift amount register

For Example: ADD $t0, $s1, $s2

Op (4 bits)	RD (3 bits)	RS (3 bits)	RT (3 bits)	SA (1 bit)

I-Type Format: -

• Load word, store word, branch type, & immediate type are I-type

•Opcode 4 bits, RT and RS are 3 bits, and Immediate 4 bits

• RS is a source register, an address for loads and stores, or an operand for branch and immediate arithmetic instructions

• RT is a source register for branches, but a destination register for the other I-type instructions

For Example: ADDI $t0, $t1, 4

OP (4 bits)	RD (3 bits)	RS (3 bits)	Address/Immediate (4 bits)

J-Type Format: -

- Jump type is J-Type
- Target address
- OP (Opcode) 4 bits , Target 10 bits

For Example: J EXIT

OP (4 bits)	Target (10 bits)

Format (fields and field length)

R-Type

Operations	Opcode (OP) 4 bit				Destination (RD) 3 bit			Source Reg (RS) 3 bit			Target Reg (RT) 3 bit			Shift Amount (SA) 1 bits
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	0	RD			RS			RT			SA
SUB	0	0	0	1	RD			RS			RT			SA
SRL	0	0	1	0	RD			RS			RT			SA
SLL	0	0	1	1	RD			RS			RT			SA
AND	0	1	0	0	RD			RS			RT			SA
OR	0	1	0	1	RD			RS			RT			SA
XOR	0	1	1	0	RD			RS			RT			SA

I-Type

Operations	Opcode (OP) 4 bits				Destination (RD) 3 bits	Source Reg (RS) 3 bits	Immediate 4 bits
ADDI	0	1	1	1	RD	RS	Immediate
LW	1	0	0	0	RD	RS	Immediate
SW	1	0	0	1	RD	RS	Immediate
BEQ	1	0	1	0	RD	RS	Immediate
BNE	1	0	1	1	RD	RS	Immediate

J-Type

Operations	Opcode (OP) 4 bits				Target 10 bits
J	1	1	0	0	target

List of Registers:

Name of the Registers	Registers Number	Value Assigned(3 bits)
$zero	0	000
$t0	1	001
$t1	2	010
$t2	3	011
$t3	4	100
$s0	5	101
$s1	6	110
$s2	7	111

Instruction Description:

ADD: It adds two registers and stores the result in destination register.
Operation: $s2(Destination) = $t1(Source) + $t2(Target)
Syntax: ADD $s2,$t1,$t2

SUB: It substruct two register's data and stores the result in destination register. It cannot handle negative value.
Operation: $s2(Destination) = $t1(Source) - $t2(Target)
Syntax: SUB $s2,$t1, $t2

SRL: Basically it does right shift.
Operation: $s2(Destination)=$t1(Source)>>im.v
Syntax: SRL $s2,$t1,1

SLL: Basically it does left shift.
Operation: $s2(Destination)=$s1(Source)<<im.v
Syntax: SLL $s2,$t1,1

AND: It AND's two register values and stores the result in destination register. Basically, it sets some bits to 0.
Operation: $s2(Destination)=$t1(Source) && $t2(Target)
Syntax: AND $s2,$t1,$t2

OR: It OR's two register values and stores the result in destination register. Basically, it sets some bits to 1.
Operation: $s2(Destination)=$t1(Source)||$t2(Target)
Syntax: OR $s2,$t1,$t2

XOR: Exclusive ors two registers and stores the result in a register
Operation: XOR: Rd = Rs ^ Rt
Syntax: XOR $s2, $t1, $t2

ADDI: It adds a value from register with an integer value and stores the result in destination register.
Operation: $s2(Destination)=$t1(Source)+constant
Syntax: ADDI $s2,$t1,4

LW: It loads required value from the memory and write it back into the register.
Operation: $s2(Destination) = MEM[offset+($t0(base address))]
Syntax: LW $s2, offset($t0)

SW: It stores specific value from register to memory.
Operation: MEM[offset+($t0(base address))] = $s2(Destination)
Syntax: SW $s2, offset($t0)

BEQ: It checks whether the values of two register s are same or not. If it's same it performs the operation located in the offset value
Operation: if ($s1==$t1) jump to level.
Syntax: BEQ $s1, $t1, offset

BNE: It checks whether the values of two registers are same or not. If it's not same, it performs the operation located in the address at offset value.
Syntax: BNE $s2, $t1, offset
Operation: if ($s2!=$t1) jump to offset else go to next line

J: It directly jumps to the target without any conditions.
Operation: Jump to the offset
Syntax: J exit

Summary:

• This is a 14-bit RISC Type CPU. As an ISA designer, I have chosen 3 mixed type operands, 9 opcodes, 5 types of operations,3 types of instruction format, and finally I have given names and values of 8 different registers.

• This is a 14-bit CPU as a result the registers length is 14 bits. For the design, I believe simplicity favors regularity. So, I have assigned 4bits to the opcodes and it is same for every type of formats. For R-type I have assigned 3 bits each for RD, RS, RT and 1 bit for SA. For I-type I have assigned 3 bits each for RT, RS and 4 bits for immediate. For J-type 10 bits for target.

• Smaller is faster. Though it is only a 14-bit CPU, it will handle very common operations. In this CPU, I tried to use as many as operations it can handle also keeping register's values in my mind.