Problem Description:

Given a 32-bit binary number b31b30b29…….b2b1b0 that represents a MIPS instruction, identify the following fields from it.

a. The format of the instruction (e.g., R-Format/I-Format/J-Format)

b. Operation (e.g., add, sub, and, or, etc.)

c. The Source register number/s (in decimal, separated by comma if two operands exist)

d. The destination register number (in decimal)

e. Shift amount (in decimal)

f. Constant/Offset (in decimal)

If a field does not exist for a given input, print “none” for that field. You may only consider the add, addi, sub, and, or, slt, lw, sw, #### and beq instructions of the MIPS architecture. You must show your outputs for the following test cases. You will not get any

points even if you miss a single test case.

Test cases (inputs):

00000010001100100100000000100000

00100010001010000000000000000101

00000010010100111000100000100010

10001110010100010000000001100100

00000001010010111001000000100101

00000001001010101001100000100100

00000001010010110100100000101010

10101110001100110000000001100100

00010010001100100000000001100100

In [126…

```python
# putting the input in list
inputList = ['00000010001100100100000000100000',
             '00100010001010000000000000000101',
             '00000010010100111000100000100010',
             '10001110010100010000000001100100',
             '00000001010010111001000000100101',
             '00000001001010101001100000100100',
             '00000001010010110100100000101010',
             '10101110001100110000000001100100',
             '00010010001100100000000001100100']
```

Various R-Format instructions:

Mnemonic	Meaning	Opcode	Funct
add	Add	0	32
addu	Add unsigned	0	33
and	Bitwise and	0	36
nor	Bitwise nor	0	39
or	Bitwise or	0	37
slt	Set on less than	0	42
sltu	Set on less than unsigned	0	43
sll	Shift left logical	0	0
srl	Shift right logical	0	2
sra	Shift right arithmetic	0	3
sub	Subtract	0	34
subu	Subtract unsigned	0	35

In [128…

```python
def r_type_func(entry):
    if (int(entry[-6:], 2)) == 32:
        print('Operation: add')
    elif (int(entry[-6:], 2)) == 33:
        print('Operation: addu')
    elif (int(entry[-6:], 2)) == 36:
        print('Operation: and')
    elif (int(entry[-6:], 2)) == 39:
        print('Operation: nor')
    elif (int(entry[-6:], 2)) == 37:
        print('Operation: or')
    elif (int(entry[-6:], 2)) == 42:
        print('Operation: slt')
    elif (int(entry[-6:], 2)) == 43:
        print('Operation: sltu')
    elif (int(entry[-6:], 2)) == 0:
        print('Operation: sll')
    elif (int(entry[-6:], 2)) == 2:
        print('Operation: srl')
    elif (int(entry[-6:], 2)) == 3:
        print('Operation: sra')
    elif (int(entry[-6:], 2)) == 34:
        print('Operation: sub')
    elif (int(entry[-6:], 2)) == 35:
        print('Operation: subu')
```

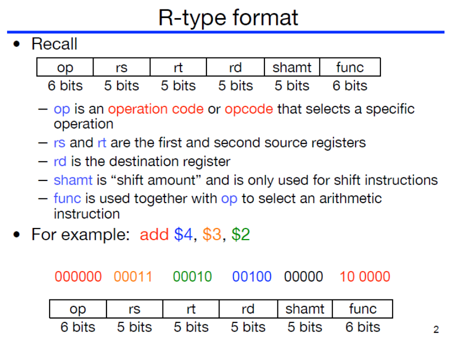
Various I-Format instructions:

Mnemonic	Meaning	Opcode
addi	Add immediate	8
addiu	Add immediate unsigned	9
andi	Bitwise and immediate	12
beq	Branch if equal	4
bne	Branch if not equal	5
lui	Load upper immediate	15
lw	Load word	35
ori	Bitwise or immediate	13
slti	Set on less than immediate	10
sltiu	Set on less than immediate unsigned	11
sw	Store word	43

In [129…

```python
def i_type_func(entry):
    if (int(entry[0:6], 2)) == 8:
        print('Operation: addi')
    elif (int(entry[0:6], 2)) == 9:
        print('Operation: addiu')
    elif (int(entry[0:6], 2)) == 12:
        print('Operation: andi')
    elif (int(entry[0:6], 2)) == 4:
        print('Operation: beq')
    elif (int(entry[0:6], 2)) == 5:
        print('Operation: bne')
    elif (int(entry[0:6], 2)) == 15:
        print('Operation: lui')
    elif (int(entry[0:6], 2)) == 35:
        print('Operation: lw')
    elif (int(entry[0:6], 2)) == 13:
        print('Operation: ori')
    elif (int(entry[0:6], 2)) == 10:
        print('Operation: slti')
    elif (int(entry[0:6], 2)) == 11:
        print('Operation: sltiu')
    elif (int(entry[0:6], 2)) == 43:
        print('Operation: sw')
```

For R-Type Operations:



R-type format

- Recall

op	rs	rt	rd	shamt	func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

  - op is an operation code or opcode that selects a specific operation
  - rs and rt are the first and second source registers
  - rd is the destination register
  - shamt is “shift amount” and is only used for shift instructions
  - func is used together with op to select an arithmetic instruction
- For example: add $4, $3, $2

000000 00011 00010 00100 00000 10 0000

op	rs	rt	rd	shamt	func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

2

Functions for R-Type Operations:

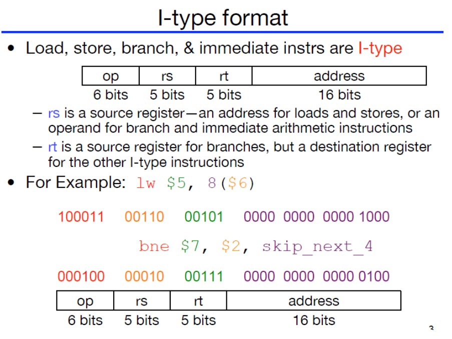
In [130…

```python
def r_source_reg(entry):
    print(f"Source Registers: {int(entry[6:11], 2)}, {int(entry[11:16], 2)}")

def r_destination_reg(entry):
    print(f"Destination Register: {int(entry[16:21], 2)}")

def r_shamt(entry):
    print(f"Shift amount: {int(entry[21:26], 2)}")
```

For I-Type Operations:



I-type format

- Load, store, branch, & immediate instrs are I-type

op	rs	rt	address
6 bits	5 bits	5 bits	16 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: lw $5, 8($6)

100011 00110 00101 0000 0000 0000 1000

bne $7, $2, skip_next_4

000100 00010 00111 0000 0000 0000 0100

op	rs	rt	address
6 bits	5 bits	5 bits	16 bits

3

Functions for I-Type Operations:

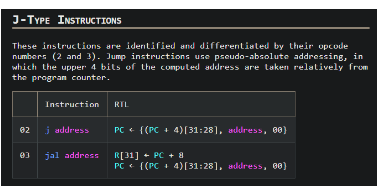
In [131…

```python
def i_source_reg(entry):
    print(f"Source Register: {int(entry[6:11], 2)}")

def i_destination_reg(entry):
    print(f"Destination Register: {int(entry[11:16], 2)}")

def i_constant_or_offset(entry):
    print(f"Constant/Offset: {int(entry[-16:], 2)}")
```

For J-Type Operations:



J-TYPE INSTRUCTIONS

These instructions are identified and differentiated by their opcode numbers (2 and 3). Jump instructions use pseudo-absolute addressing, in which the upper 4 bits of the computed address are taken relatively from the program counter.

	Instruction	RTL
02	j address	PC ← {(PC + 4)[31:28], address, 00}
03	jal address	R[31] ← PC + 8 PC ← {(PC + 4)[31:28], address, 00}

Function for J-Type Operation:

In [132…

```python
def j_type_func(entry):
    if (int(entry[0:6], 2)) == 2:
        print('Operation: jump')
    elif (int(entry[0:6], 2)) == 3:
        print('Operation: jump and link')
```

Main Function:

In [135…

```python
for entry in inputList:
    print(f"For the bit string {entry}:")
    if int(entry[0:6], 2) == 0:
        print("Instruction Format: R")
        r_type_func(entry)
        r_source_reg(entry)
        r_destination_reg(entry)
        r_shamt(entry)
        print("Constant/Offset: none")
    elif int(entry[0:6], 2) == 2 or int(entry[0:6], 2) == 3:
        print("Instruction Format: J")
        j_type_func(entry)
    else:
        print("Instruction Format: I")
        i_type_func(entry)
        i_source_reg(entry)
        i_destination_reg(entry)
        print("Shift amount: none")
        i_constant_or_offset(entry)

    print('\n')
```

```
For the bit string 00000010001100100100000000100000:
Instruction Format: R
Operation: add
Source Registers: 17, 18
Destination Register: 8
Shift amount: 0
Constant/Offset: none


For the bit string 00100010001010000000000000000101:
Instruction Format: I
Operation: addi
Source Register: 17
Destination Register: 8
Shift amount: none
Constant/Offset: 5


For the bit string 00000010010100111000100000100010:
Instruction Format: R
Operation: sub
Source Registers: 18, 19
Destination Register: 17
Shift amount: 0
Constant/Offset: none


For the bit string 10001110010100010000000001100100:
Instruction Format: I
Operation: lw
Source Register: 18
Destination Register: 17
Shift amount: none
Constant/Offset: 100


For the bit string 00000001010010111001000000100101:
Instruction Format: R
Operation: or
Source Registers: 10, 11
Destination Register: 18
Shift amount: 0
Constant/Offset: none


For the bit string 00000001001010101001100000100100:
Instruction Format: R
Operation: and
Source Registers: 9, 10
Destination Register: 19
Shift amount: 0
Constant/Offset: none


For the bit string 00000001010010110100100000101010:
Instruction Format: R
Operation: slt
Source Registers: 10, 11
Destination Register: 9
Shift amount: 0
Constant/Offset: none


For the bit string 10101110001100110000000001100100:
Instruction Format: I
Operation: sw
Source Register: 17
Destination Register: 19
Shift amount: none
Constant/Offset: 100


For the bit string 00010010001100100000000001100100:
Instruction Format: I
Operation: beq
Source Register: 17
Destination Register: 18
Shift amount: none
Constant/Offset: 100
```

In [ ]: