Department of Electrical & Computer Engineering
North South University

Project Title: A 12-bit Custom RISC-V ISA

Course: CSE 332 - Computer Organization & Architecture

Section – 06

Summer 2022

Submitted by:

Zobaer Ahammod Zamil

ID: 2021796042

Submitted To:

Dr. Mainul Hossain (MHo1)

Department of Electrical & Computer Engineering,
North **S**outh **U**niversity

Design of A 12-bit Custom RISC-V ISA

Objectives:

- I will design a 12-bit ISA which will perform some arithmetic, logical operations, branching and jumping.

Here is the analysis of my designed 12-bit ISA.

1. How many types of instruction?

- I will use 3 types of instruction. Those are:
  i. R-type
  ii. I-type
  iii. J-type

2. Describe each of the formats (fields and field length)?

R-type Format (Register):

Op code (3-bit)	Rs (3-bit)	Rt (3-bit)	Rd (3-bit)

I-type Format (Immediate):

Op code (3-bit)	Rs (3-bit)	Rd (3-bit)	Immediate (3-bit)

J-type Format (Jump):

Op code (3-bit)	Target (9-bit)

Op code ---- Operation Code (Determine what operation will be performed)
Rs ----- Source Register 1
Rt ----- Source Register 2 (for R-type) / Destination Register (for I-type)
Rs ----- Destination Register 1
Immediate --- Immediate, Branch or Address Displacement
Target ---- Jump to target address

3. How many operands?

- There are 3 operands, those are Rs, Rt, Rd.

4. How many operations?

- Here I assign 3-bit for opcode, so $\mathbf{2^3 = 8}$ different operation can be executed by using this ISA.

5. Type of operations?

- Arithmetic
- Logical
- Branch
    - Conditional
    - Unconditional
- Data Transfer

Decimal Opcode	Binary Opcode	Operation	Name	Type	Syntax	Category
0	000	Addition	add	R	add $rd, $rs, $rt	Arithmetic
1	001	Subtraction	sub	R	sub $rd, $rs, $rt	Arithmetic
2	010	Immediate Addition	addi	I	addi $rd, $rs, const	Arithmetic
3	011	Load data	lw	I	lw $rd, offset($rs)	Data Transfer
4	100	Store data	sw	I	sw $rd, offset($rs)	Data Transfer
5	101	AND Logical	and	R	and $rd, $rs, $rt	Logical
6	110	Branch Equal	beq	I	jmp $rs, $rt, offset	Conditional
7	111	Jump	jmp	J	jmp offset	Unconditional

Register Table:

- I assign 3-bit for register, so there is total $\mathbf{2^3 = 8}$ register in my register file. Each one can be used for saving values in hexadecimal form.

Register Name	Register Number	Binary value
$R0	0	000
$R1	1	001
$R2	2	010
$R3	3	011
$R4	4	100
$R5	5	101
$R6	6	110
$R7	7	111

Example Instructions:

Here is some sample of high-level language that will run in CUP using my 12-bit ISA.

Addition (add): It will add two register and store in destination register

Syntax: add $rd, $rs, $rt

Problem: a = b + c;

In ISA: add $rd, $rs, $rt # $rd = $rs + $rt

Subtraction (sub): It will subtract two register and store in destination register

Syntax: sub $rd, $rs, $rt

Problem: a = b - c;

In ISA: sub $rd, $rs, $rt # $rd = $rs - $rt

Addition of Immediate (addi): It will add one register with a constant and store in destination register

Syntax: addi $rd, $rs, const

Problem: a = b + 4;

In ISA: addi $rd, $rs, 4 # $rd = $rs + 4

Load data (lw): Value from a location of memory will load into a register for calculation

Syntax: lw $rd, offset($rs)

Problem: a[1] = 5; g = 5 + a[1];

In ISA: lw $rd, 4($rs) # $rd = MEM[$rs + 4] ; &a[0] = $rs

addi $rd, $rd, 5 # $rd = $rd + 4

Store data (lw): Value from register will store in memory location

Syntax: sw $rd, offset($rs)

Problem: g[0] = a + 1;

In ISA: addi $rd, $rs, 1 # $rd = $rs + 1

sw $rd, 0($rs) # MEM[$rs + 0] = $rd ; &g[0] = $rs

AND Logical (and): It will check the bits of 2 binary values, and match the bits. If both bits of same position are 1 the it will be 1, otherwise it will be 0. And this output value will store in third register.

Syntax: and $rd, $rs, $rt

Branch Equal (beq): If the two values are same then program will goto the offset location

Syntax: beq $rs, $rt, offset

Problem: if (g = = a)

In ISA: $rd = = $rs jump to offset location

Jump (jmp): It will goto the offset location

Syntax: jmp offset

Problem: calling a function from another function

In ISA: jump to offset location

High-Level Language to Assembly Instruction (ISA format):

(i) c = a + b;

```
add $R1, $R2, $R3        # $R3 = $R1 + $R2
```

(ii) d = c – a;

```
sub $R3, $R1, $R4        # $R4 = $R3 - $R1
```

(iii) A[0] = d;

```
store $R0, $R4, 0        # M[$R0 + 0] = $R4
```

(iv) C code:

```
1  int i =  0;
2
3  while(i !=  5){
4
5      i++;
6  }
7
8
```

MIPS Code:

```
start:
   addi $r0, $r1, 0
   addi $r0, $r2, 5
loop:
    beq $r2, $r1, exit      # jump to line 7
    addi $r1, $r1, 1
    j loop                  # jump back to line 3
exit:
```

Limitation:

As I use 3-bit for opcode, so I cannot perform more than 8 operations. As a result, I cannot perform other logical operations which can be performed in R-type, I-type ISA format.