

NORTH SOUTH UNIVERSITY

Department of Electrical & Computer Engineering

**Fall 2018**

**Course ID: CSE332**

Course Title: Computer Organization and Architecture

Project: Instruction Set Architecture (ISA)

Section: 1

Group No: 6

Group Members:

|  |  |
| --- | --- |
| **Name** | **Student ID** |
| 1. Pritom Kumar Saha | 1712318642 |
| 2. Shadman S. Khan | 1611918042 |
| 3. Novera Tasnova Chowdhury | 1520378042 |
| 4. Sakib Monir | 1530964042 |
| 5. Gazi Shafayet Hossain | 1430364042 |

**Introduction:**

Our task was to design 10 bit single-cycle (RISC) CPU that has separate Data and Instruction Memory.

**Objective:**

We have to design our ISA focusing on the following three categories of programs:

a) Simple arithmetic & logic operations

b) Programs that require checking conditions

c) Loop type of programs

**Operands:**

We’ve used 8 (depends on ISA format) operands in our ISA.

We’ve used 8 registers: 3 saved registers, 1 zero register, 2 temporary registers, 1 accumulator and the last register is reserved for input and output. The Register we have selected are given below:

|  |  |
| --- | --- |
| Name | Value Assigned |
| $zero | 000 |
| $s0 | 001 |
| $s1 | 010 |
| $s2 | 011 |
| $t0 | 100 |
| $t1 | 101 |
| $io | 110 |
| $acc | 111 |

**Operations:**

There are 16 different operations. Syntax Rules:

1. There must be at least a single **space** between the opcode and the trailing contents.
2. If there are multiple operands, there must a **comma** after the first operand.
3. System allocated register name always starts with the **‘$’** -sign.
4. Constant starts with the **‘#’** – sign.
5. There can not be any space after the register and constant signs.
6. The instructions are **not case-sensitive**.

The details are given below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Category** | **Operation** | **Syntax** | **Meaning** | **Comments** |
| Arithmetic | add | add $s1, $s2 | $acc = $s1 + $s2 | Add two registers and keep the result in the accumulator |
| subtract | sub $s1, $s2 | $acc = $s1 - $s2 | Subtract a register from another and keep the result in the accumulator |
| add immediate | addi $s1, #const | $acc = $s1 + constant | Add an immediate value with a register and keep the result inside the accumulator |
| Data Transfer | load accumulator | lda $s1, #const | $acc = Memory [variable + offset] | Loads from a memory position to the accumulator |
| store accumulator | sta $s1, #const | Memory [variable + offset] = $acc | Stores into the memory position from the accumulator |
| Move register value | mov $s1, $s2 | $s1 = $s2 | Copies the value of reg2 to reg1 |
| Move immediate value | movi $s1, #const | $s1 = constant | Copies the constant into reg2 |
| Logical | AND | and $s1, $s2 | $acc = $s1 & $s2 | And operation |
| OR | or $s1, $s2 | $acc = $s1 | $s2 | Or operation |
| NOR | nor $s1, $s2 | $acc = ~($d | $s) | Nor operation |
| Conditional  branch | Jump if equal | jeq $s1, #target | If ($acc == $s1)  Jump to Target | If the register value is equal to the accumulator then jump to target |
| Jump if not equal | jne $s1, #target | If ($acc != $s1)  Jump to Target | If the register value is not equal to the accumulator then jump to target |
| Jump if less than | jlt $s1, #target | If ($s1 < $acc)  Jump to Target | If the register value is less than the accumulator then jumps to target |
| Unconditional  Jump | Jump | jmp #target | Jump to target address | Can jump anywhere within the first 64 lines of code |
| External Communication | In | in $io | Take user input into the io register | Perform input operations |
| Out | out $io | Outputs the data from io register | Perform output operations |

**Format:**

The format and op code for 3 different ISA type are given below:

**RA-Type List:**

8 operations. Used for register-register arithmetic operations and moving.

|  |  |  |
| --- | --- | --- |
| Operation | Op-code | ALU Operation (A’-B’-Op) |
| and | 0000 | AND (0-0-00) |
| or | 0001 | OR (0-0-01) |
| add | 0010 | ADD (0-0-10) |
| mov | 0011 | N/A |
| nor | 0100 | NOR using AND (1-1-00) |
| in | 0101 | N/A |
| sub | 0110 | SUB using ADD (0-1-10) |
| out | 0111 | N/A |

**RA-Type ISA format:**

|  |  |  |
| --- | --- | --- |
| Op-code | rs | rt |
| Bits: 4 bits | 3 bits | 3 bits |

**IB-Type List:**

7 operations. Used for immediate operations and Branching.

|  |  |  |
| --- | --- | --- |
| Operation | Op code | ALU Operation (A’-B’-Op) |
| addi | 1000 | ADD (0-0-10) |
| sta | 1001 | ADD (0-0-10) |
| lda | 1010 | ADD (0-0-10) |
| movi | 1011 | N/A |
| jeq | 1100 | SUB using ADD (0-1-10) |
| jne | 1101 | SUB using ADD (0-1-10) |
| jlt | 1110 | LESS (0-1-11) |

**IB-Type ISA format:**

|  |  |  |
| --- | --- | --- |
| Op code | rs | Immediate |
| Bits: 4 bits | 3 bits | 3 bits |

**J-TYPE LIST:**

1 operation. Used for Jump.

|  |  |
| --- | --- |
| Operation | Op code |
| jmp | 1111 |

**J-Type ISA format:**

|  |  |
| --- | --- |
| Op-code | Target |
| Bits: 4 bits | 6 bits |