**Instruction Set Architecture (ISA)**

**Objectives:** Design a 16-bit RISC type CPU which is capable of conducting arithmetic operations, logical operations, branching and handling loops.

**Operations:** The designed ISA would be able to perform a total number of 10 operations. At least 4 bits would be needed to identify each opcode, thus total number of 16 combinations would be available. But the first 10 combinations would be enough to uniquely identify each instruction.

**Types of Operations:** There will be total **five** different types of operations;

* Arithmetic
* Logical
* Data Transfer
* Conditional Branch
* Unconditional Jump

**Operands:**

Three types of operands would be used to construct the CPU;

* Two register based operands
* One memory based operands

**Format:**

Three types of formats would be used in the ISA to design the CPU;

* Register Type – (R-type)
* Immediate Type – (I-type)
* Jump Type – (J-type)

**R-type:**

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | RD | RS | RT |
| 4-bit | 4-bit | 4-bit | 4-bit |

**I-type:**

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | RD | RS | Immediate |
| 4-bit | 4-bit | 4-bit | 4-bit |

**J-type:**

|  |  |
| --- | --- |
| Opcode | Target |
| 4-bit | 12-bit |

**Registers:**

|  |  |  |  |
| --- | --- | --- | --- |
| Register Number | Conventional Name | Usage | Binary |
| 0 | $r0 | General purpose | 0000 |
| 1 | $r1 | General purpose | 0001 |
| 2 | $r2 | General purpose | 0010 |
| 3 | $r3 | General purpose | 0011 |
| 4 | $r4 | General purpose | 0100 |
| 5 | $r5 | General purpose | 0101 |
| 6 | $r6 | General purpose | 0110 |
| 7 | $r7 | General purpose | 0111 |
| 8 | $r8 | General purpose | 1000 |
| 9 | $r9 | General purpose | 1001 |
| 10 | $r10 | General purpose | 1010 |
| 11 | $r11 | General purpose | 1011 |
| 12 | $r12 | General purpose | 1100 |
| 13 | $r13 | General purpose | 1101 |
| 14 | $r14 | General purpose | 1110 |
| 15 | $r15 | General purpose | 1111 |

**Operations:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instruction Type | Operation | Name | Type | Opcode |
| Arithmetic | Subtraction | sub | R | 0000 |
| Arithmetic | Addition | add | R | 0001 |
| Arithmetic | Addition with an immediate | addi | I | 0010 |
| Unconditional | Jump | jmp | J | 0011 |
| Logical | Bit-by-bit AND | and | R | 0100 |
| Data Transfer | Load Word | lw | I | 0101 |
| Conditional | Check for equality | beq | I | 0110 |
| Logical | Shift Left | sll | R | 0111 |
| Conditional | Compares less than | slt | R | 1000 |
| Data Transfer | Store Word | sw | I | 1001 |

**Details of each Operations & Opcodes:**

|  |  |  |
| --- | --- | --- |
| Operation Details | Syntax | Comments |
| sub: 0000  Contents of the source register 1 (RS) is subtracted from the contents of the source register 2(RT), and stored in the destination register (RD). | sub $r2 $r2 $r2  op RD RS RT | $r2 = $r2 - $r2 |
| add: 0001  Contents of the source register 1 (RS) is added with the contents of the source register 2(RT), and stored in the destination register (RD). | add $r2 $r4 $r6  op RD RS RT | $r2 = $r4 + $r6 |
| addi: 0010  Contents of the source register 1 (RS) is added with the constant (Im), and stored in the destination register (RD). | addi $r1 $r2 **5**  op RD RS Im | $r1 = $r2 - **5** |
| jmp: 0011  It jumps a relative number of instruction(s) specified by given number. | jmp 6 | Go to line 6 |
| and: 0100  Bit-by-bit logical operations between two source register RS & RT, and result stored in RD. | and $r2 $r2 $r3  op RD RS RT | $r2 = $r2 AND $r3 |
| lw: 0101  Loads the contents of the memory specified by the offset, and saves it into the destination register (RD). | lw $r0 **7**($r1)  **offset**  lw $r0 $r1 **7**  op RD RS Im | $r0 = memory[$r1 + **7**] |
| beq: 0110  Checks the content of the provided register is equal to the content of another register. If the contents are equal, then jump to a relative number of instruction. | beq $r0 $r1 **9**  beq $r0 $r1 **L**  **label** | if ($r0 == $r1) {  jump to line 9;  } else {  go to next line;  } |
| sll: 0111  Shift the content of a register (RS) to left by the content of the register (RT), and stores the value into the destination register (RD). | sll $r0 $r1 $r2  op RD RS RT | $r1 = 0111  $r2 = 0001  $r0 = 1110 |
| slt: 1000  Compares the constants of two source registers RS & RT. If the content of 1st source register (RS) is less than the 2nd source register (RT), then it sends 1 to the destination register (RD); otherwise 0 is sent to the destination register. | slt $r0 $r0 $r1  op RD RS RT | if ($r0 < $r1) {  $r0 = 1;  } else {  $r0 = 0;  } |
| sw: 1001  Stores the contents of the source register (RD) specified by the offset, and saves it into the data memory (RAM). | sw $r0 **8**($r1)  **offset**  sw $r0 $r1 **7**  op RD RS Im | memory[$r1 + **7**] = $r0 |

**Translations of high-level codes into machine code:**

|  |  |  |
| --- | --- | --- |
| High Level Code | Assembly Code | Comment |
| a = a + b | add $r1 $r1 $r2 | $r1 = a  $r2 = b  $r1 gets $r1 + $r2 |
| a = a + 8 | addi $r1 $r1 8 | $r1 = a  $r1 gets $r1 + 8 |
| a = a - b | sub $r1 $r1 $r2 | $r1 = a  $r2 = b  $r1 gets $r1 - $r2 |
| a = a AND b | sub $r1 $r1 $r2 | $r1 = a  $r2 = b  $r1 gets $r1 AND $r2 |
| b = 0;  for (i = 0; i < 2; i++) {  b = b + 4;  } | sub $r0 $r0 $r0  sub $r1 $r1 $r1  sub $r2 $r2 $r2  addi $r2 $r2 2  beq $r1 $r2 3  addi $r0 $r0 4  addi $r1 $r1 1  jmp 4  sub $r15 $r15 $r15  sw $r0 $r15 15  sub $r15 $r15 $r15  lw $r12 $r15 15 | beq: (4 - 1) = 3 [beq $r1 $r2 3]  jmp: (4)  From,  [jmp 4]  To,  [beq $r1 $r2 3] |

**Limitations:**

* In this is 16-bit CPU, only 10 operations were used and in maximum 16 operations can be used.
* For bit limitation, the ‘Function’ and ‘Shift amount’ of R-type instruction were not used in the CPU. Instead, the instructions were passed to the Opcode portion of the ISA.
* This CPU doesn’t have any error handling functionalities; thus the user must valid instructions in correct order that is supported by the CPU.