MIPS Assembler Specifications

The assembler source code as well as a relevant readme file can be found in CPUS/cpu/bin directory.

# Standard instructions

The implemented assembler will translate the following instructions:

### ADD – *Add (with overflow)*

|  |  |
| --- | --- |
| Description: | Adds two registers and stores the result in a register |
| Operation: | $d = $s + $t; advance\_pc (4); |
| Syntax: | add $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0000 |

### ADDI -- *Add immediate (with overflow)*

|  |  |
| --- | --- |
| Description: | Adds a register and a sign-extended immediate value and stores the result in a register |
| Operation: | $t = $s + imm; advance\_pc (4); |
| Syntax: | addi $t, $s, imm |
| Encoding: | 0010 00ss ssst tttt iiii iiii iiii iiii |

### AND -- *Bitwise and*

|  |  |
| --- | --- |
| Description: | Bitwise ands two registers and stores the result in a register |
| Operation: | $d = $s & $t; advance\_pc (4); |
| Syntax: | and $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0100 |

### ANDI -- *Bitwise and immediate*

|  |  |
| --- | --- |
| Description: | Bitwise ands a register and an immediate value and stores the result in a register |
| Operation: | $t = $s & imm; advance\_pc (4); |
| Syntax: | andi $t, $s, imm |
| Encoding: | 0011 00ss ssst tttt iiii iiii iiii iiii |

### BEQ -- *Branch on equal*

|  |  |
| --- | --- |
| Description: | Branches if the two registers are equal |
| Operation: | if $s == $t advance\_pc (offset << 2)); else advance\_pc (4); |
| Syntax: | beq $s, $t, offset |
| Encoding: | 0001 00ss ssst tttt iiii iiii iiii iiii |

### BNE -- *Branch on not equal*

|  |  |
| --- | --- |
| Description: | Branches if the two registers are not equal |
| Operation: | if $s != $t advance\_pc (offset << 2)); else advance\_pc (4); |
| Syntax: | bne $s, $t, offset |
| Encoding: | 0001 01ss ssst tttt iiii iiii iiii iiii |

### DIV -- *Divide*

|  |  |
| --- | --- |
| Description: | Divides $s by $t and stores the quotient in $LO and the remainder in $HI |
| Operation: | $LO = $s / $t; $HI = $s % $t; advance\_pc (4); |
| Syntax: | div $s, $t |
| Encoding: | 0000 00ss ssst tttt 0000 0000 0001 1010 |

### J -- *Jump*

|  |  |
| --- | --- |
| Description: | Jumps to the calculated address |
| Operation: | PC = nPC; nPC = (PC & 0xf0000000) | (target << 2); |
| Syntax: | j target |
| Encoding: | 0000 10ii iiii iiii iiii iiii iiii iiii |

### JAL -- *Jump and link*

|  |  |
| --- | --- |
| Description: | Jumps to the calculated address and stores the return address in $31 |
| Operation: | $31 = PC + 8 (or nPC + 4); PC = nPC; nPC = (PC & 0xf0000000) | (target << 2); |
| Syntax: | jal target |
| Encoding: | 0000 11ii iiii iiii iiii iiii iiii iiii |

### JR -- *Jump register*

|  |  |
| --- | --- |
| Description: | Jump to the address contained in register $s |
| Operation: | PC = nPC; nPC = $s; |
| Syntax: | jr $s |
| Encoding: | 0000 00ss sss0 0000 0000 0000 0000 1000 |

### LB -- *Load byte*

|  |  |
| --- | --- |
| Description: | A byte is loaded into a register from the specified address. |
| Operation: | $t = MEM[$s + offset]; advance\_pc (4); |
| Syntax: | lb $t, offset($s) |
| Encoding: | 1000 00ss ssst tttt iiii iiii iiii iiii |

### LUI -- *Load upper immediate*

|  |  |
| --- | --- |
| Description: | The immediate value is shifted left 16 bits and stored in the register. The lower 16 bits are zeroes. |
| Operation: | $t = (imm << 16); advance\_pc (4); |
| Syntax: | lui $t, imm |
| Encoding: | 0011 11-- ---t tttt iiii iiii iiii iiii |

### LW -- *Load word*

|  |  |
| --- | --- |
| Description: | A word is loaded into a register from the specified address. |
| Operation: | $t = MEM[$s + offset]; advance\_pc (4); |
| Syntax: | lw $t, offset($s) |
| Encoding: | 1000 11ss ssst tttt iiii iiii iiii iiii |

### MFHI -- *Move from HI*

|  |  |
| --- | --- |
| Description: | The contents of register HI are moved to the specified register. |
| Operation: | $d = $HI; advance\_pc (4); |
| Syntax: | mfhi $d |
| Encoding: | 0000 0000 0000 0000 dddd d000 0001 0000 |

### MFLO -- *Move from LO*

|  |  |
| --- | --- |
| Description: | The contents of register LO are moved to the specified register. |
| Operation: | $d = $LO; advance\_pc (4); |
| Syntax: | mflo $d |
| Encoding: | 0000 0000 0000 0000 dddd d000 0001 0010 |

### MULT -- *Multiply*

|  |  |
| --- | --- |
| Description: | Multiplies $s by $t and stores the result in $LO. |
| Operation: | $LO = $s \* $t; advance\_pc (4); |
| Syntax: | mult $s, $t |
| Encoding: | 0000 00ss ssst tttt 0000 0000 0001 1000 |

### NOR -- *Bitwise nor*

|  |  |
| --- | --- |
| Description: | Bitwise logical nors two registers and stores the result in a register |
| Operation: | $d = !($s | $t); advance\_pc (4); |
| Syntax: | nor $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0111 |

### OR -- *Bitwise or*

|  |  |
| --- | --- |
| Description: | Bitwise logical ors two registers and stores the result in a register |
| Operation: | $d = $s | $t; advance\_pc (4); |
| Syntax: | or $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0101 |

### ORI -- *Bitwise or immediate*

|  |  |
| --- | --- |
| Description: | Bitwise ors a register and an immediate value and stores the result in a register |
| Operation: | $t = $s | imm; advance\_pc (4); |
| Syntax: | ori $t, $s, imm |
| Encoding: | 0011 01ss ssst tttt iiii iiii iiii iiii |

### SB -- *Store byte*

|  |  |
| --- | --- |
| Description: | The least significant byte of $t is stored at the specified address. |
| Operation: | MEM[$s + offset] = (0xff & $t); advance\_pc (4); |
| Syntax: | sb $t, offset($s) |
| Encoding: | 1010 00ss ssst tttt iiii iiii iiii iiii |

### SLL -- *Shift left logical*

|  |  |
| --- | --- |
| Description: | Shifts a register value left by the shift amount listed in the instruction and places the result in a third register. Zeroes are shifted in. |
| Operation: | $d = $t << h; advance\_pc (4); |
| Syntax: | sll $d, $t, h |
| Encoding: | 0000 00ss ssst tttt dddd dhhh hh00 0000 |

### SLT -- *Set on less than (signed)*

|  |  |
| --- | --- |
| Description: | If $s is less than $t, $d is set to one. It gets zero otherwise. |
| Operation: | if $s < $t $d = 1; advance\_pc (4); else $d = 0; advance\_pc (4); |
| Syntax: | slt $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 1010 |

### SLTI -- *Set on less than immediate (signed)*

|  |  |
| --- | --- |
| Description: | If $s is less than immediate, $t is set to one. It gets zero otherwise. |
| Operation: | if $s < imm $t = 1; advance\_pc (4); else $t = 0; advance\_pc (4); |
| Syntax: | slti $t, $s, imm |
| Encoding: | 0010 10ss ssst tttt iiii iiii iiii iiii |

### SRA -- *Shift right arithmetic*

|  |  |
| --- | --- |
| Description: | Shifts a register value right by the shift amount (shamt) and places the value in the destination register. The sign bit is shifted in. |
| Operation: | $d = $t >> h; advance\_pc (4); |
| Syntax: | sra $d, $t, h |
| Encoding: | 0000 00-- ---t tttt dddd dhhh hh00 0011 |

### SRL -- *Shift right logical*

|  |  |
| --- | --- |
| Description: | Shifts a register value right by the shift amount (shamt) and places the value in the destination register. Zeroes are shifted in. |
| Operation: | $d = $t >> h; advance\_pc (4); |
| Syntax: | srl $d, $t, h |
| Encoding: | 0000 00-- ---t tttt dddd dhhh hh00 0010 |

### SUB -- *Subtract*

|  |  |
| --- | --- |
| Description: | Subtracts two registers and stores the result in a register |
| Operation: | $d = $s - $t; advance\_pc (4); |
| Syntax: | sub $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0010 |

### SW -- *Store word*

|  |  |
| --- | --- |
| Description: | The contents of $t is stored at the specified address. |
| Operation: | MEM[$s + offset] = $t; advance\_pc (4); |
| Syntax: | sw $t, offset($s) |
| Encoding: | 1010 11ss ssst tttt iiii iiii iiii iiii |

### XOR -- *Bitwise exclusive or*

|  |  |
| --- | --- |
| Description: | Exclusive ors two registers and stores the result in a register |
| Operation: | $d = $s ^ $t; advance\_pc (4); |
| Syntax: | xor $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d--- --10 0110 |

### XORI -- *Bitwise exclusive or immediate*

|  |  |
| --- | --- |
| Description: | Bitwise exclusive ors a register and an immediate value and stores the result in a register |
| Operation: | $t = $s ^ imm; advance\_pc (4); |
| Syntax: | xori $t, $s, imm |
| Encoding: | 0011 10ss ssst tttt iiii iiii iiii iiii |

# Testing instructions to implement

For testing purposes, the assembler will also convert these three instructions to machine code.

### ASRT -- *Assert*

|  |  |
| --- | --- |
| Description: | Assert that a given register is equal to another given register |
| Operation: | Assert($t, $s) |
| Syntax: | asrt $t, $s |
| Encoding: | 0101 00ss ssst tttt 0000 0000 0000 0000 |

### ASRTI -- *Assert immediate*

|  |  |
| --- | --- |
| Description: | Assert that a given register is equal to an immediate value |
| Operation: | Assert($t, i) |
| Syntax: | asrti $t, i |
| Encoding: | 0101 0100 000t tttt iiii iiii iiii iiii |

### HALT -- *Halt*

|  |  |
| --- | --- |
| Description: | Halt the processor |
| Operation: | Halt |
| Syntax: | halt |
| Encoding: | 0101 1000 0000 0000 0000 0000 0000 0000 |