By Thomas Bonatti

On behalf of: Josh, Ben Effron, Jeff, Thomas

**Addressable Registers**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Register** | **Address** | **Purpose** | **Register** | **Address** | **Purpose** |
| $M | 000 | Main accumulator. Used for arguments and returned values. | $t0 | 100 | General use. |
| $ra | 001 | Return address | $t1 | 101 | General use. Recommended to be used for flags |
| $sp | 010 | Stack pointer | $t2 | 110 | General use. Recommended for use as a second argument or return value. |
| $at | 011 | Pseudo instructions | $s1 | 111 | Safe use |

**Non-Addressable Registers:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Register** | **Purpose** | **Register** | **Purpose** |
| PC | Points to current instruction. | ALUout | Store output of the ALU for one cycle. |
| InstructionOut | Stores Instructions for multi-cycling. | ALUinA | Stores first input for the ALU. |
| ALUinB | Stores second input for the ALU. |  |  |

**I-Type instructions (Immediate):**

I-type instructions will use the main accumulator register, and a 12 bit immediate. Most logic and arithmetic commands will use this.

|  |  |
| --- | --- |
| Op Code [15:12] | Immediate [11:0] |

**R-type (move):**

Operations using registers

|  |  |  |  |
| --- | --- | --- | --- |
| Op Code [15:12] | Function code [11:6] | rt (register) [5:3] | rs (register) [2:0] |

**IR-Type(Immediate and Register):**

Branches and load word store word

|  |  |  |
| --- | --- | --- |
| Op code [15:12] | Reg [11:9] | Immediate [8:0] |

**JR-Type [Jump Register]:**

Used for jump register command

|  |  |  |  |
| --- | --- | --- | --- |
| Op Code [15:12] | Func Code [11:8] | Reg [7:5] | Unused [4:0] |

Op codes:

|  |  |  |  |
| --- | --- | --- | --- |
| **Op Code** | **Operation** | **Op Code** | **Operation** |
| 0000 | Use function code\* | 1000 | li (load immediate) |
| 0001 | addi (add immediate) | 1001 | ori (or immediate) |
| 0010 | beq (brach equal) | 1010 | andi (and immediate) |
| 0011 | bne (branch not equal) | 1011 | nori (nor immediate) |
| 0100 | J (jump) | 1100 | sll (shift left logical) |
| 0101 | lw (load word) | 1101 | srl (shift right logical) |
| 0110 | sw (store word) | 1110 | sra (shift right arithmetic) |
| 0111 | lui (load upper immediate) | 1111 | slti (set less than) |

**Function Codes:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Func Code** | **Operation** | **Func Code** | **Operation** |
| 0000 | add | 1000 | move |
| 0001 | and (logic) | 1001 | slt (set less than) |
| 0010 | or (logic) | 1010 | sub (subtract) |
| 0011 | xor (logic) | 1011 | brqz (branch if zero) |
| 0100 | nor (logic) | 1100 | bnez (branch not zero) |
| 0101 | swap | 1101 |  |
| 0110 | copy | 1110 |  |
| 0111 | jr | 1111 |  |

**Instruction Descriptions**

**Check function code** (op = 0000)

If the op code is 0000, the operation to be executed will be determiner by a 4 bit function code [11:8]

**Addi** **– add immediate** (op 0001, func = xxxx)

signature: addi <12 bit imm.>

I-type

Addi is an I-type operation that adds the given immediate to the accumulator register.

**Beq – Branch equal**  (op = 0010, func = xxxx)

signature: beq $r, <9 bit imm.>

IR-type

Beq branches from PC + 1 if the given register $r and the main accumulator are equal. Can branch +/- 256 lines up or down.

**Bne** - **brach not equal**(op = 0011, func = xxxx)   
signature: beq $r, <9 bit imm.>

IR-type

Bne branches if the given register $r and the main accumulator are not equal. Can branch ± 256 lines from PC+1.

**J – jump** (op = 0100, func = xxxx)

Signature: j <12 bit imm.>

I-type

The 4 most significant digits in the PC will be concatenated with a 12 bit immediate and used as the new PC.

**lw – load word** (op = 0101, func = xxxx)

signature: lw $r[ <9 bit imm.> ]

IR-type

Loads a 16 bit chunk of data from memory at the given address incremented by the immediate.

**sw** **– store word** (op = 0110, func = xxxx)

signature: sw $r [ <9 bit imm.> ]

IR-type

Stores the value in the main register in memory at the given address, incremented by the immediate.

**Li – Load Immediate** (op = 1000, func = xxxx)

Signature: li <12 bit imm.>

I-type

Li stores the given immediate value in the main register.

**ori – or immediate** (op = 1001, func = xxxx)

signature: ori <12 bit imm.>

I-type

ori performs a logical or operation on the value of the main register and the immediate, then stores the value in the main register.

**andi – and immediate** (op = 1010, func = xxxx)

signature: andi <12 bit imm.>

I-type

andi performs a logical and operation on the value of the main register and the immediate, then stores the value in the main register.

**nori – nor immediate** (op = 1011, func = xxxx)

signature: nori <12 bit imm.>

I-type

nori performs a logical nor operation on the value of the main register and the immediate, then stores the value in the main register.

**sll – shift left logical** (op = 1100, func = xxxx)

IR-type

signature: sll $r, <9 bit imm.>

**srl – shift right logical** (op = 1101, func = xxxx)

IR-type

signature: srl $r, <9 bit imm.>

**sra – shift right arithmetic** (op = 1110, func = xxxx)

IR-type

signature: sra $r, <9 bit imm.>

**slti – set less than** (op = 1111, func = xxxx)

signature: slt $r, <9 bit imm.>

I-type

Sets the given register r to 1 if the main register $m is less than the 9 bit immediate. Else, r is set to 0.

**add – add** (op = 0000, func = 0000)

signature: add $r1, $r2

R-type

Add adds $r1 and $r2 together and stores in the main register. To accumulate, use add $m, $r1.

**and – and** (op = 0000, func = 0001)

signature: and $r1, $r2

R-type

Preforms a logical and operation on two registers, then stores the result in the main register.

**or – or** (op = 0000, func = 0010)

signature: or $r1, $r2

R-type

Preforms a logical or operation on two registers, then stores the result in the main register.

**xor – xor** (op = 0000, func = 0011)

signature: xor $r1, $r2

R-type

Preforms a logical xor operation on two registers, then stores the result in the main register.

**nor – nor** (op = 0000, func = 0100)

signature: nor $r1, $r2

R-type

Preforms a logical nor operation on two registers, then stores the result in the main register.

**swap – swap** (op = 0000, func = 0101)

signature: swap $r1, $r2

R-type

Swaps the values in $r1 and $r2.

**copy – copy** (op = 0000, func = 0110)

signature: copy $r1, $r2

R-type

Writes the value of $r1 into register $r2.

**jr – jump register** (op = 0000, func = 0111)

signature: jr $r

JR-type

jumps to the address in the given register.

**move – move** (op = 0000, func = 1000)

signature: move $r1, $r2

R-type

Writes the value of $r1 to $r2, then sets the value of $r1 to zero.

**slt – set less than register** (op = 0000, func = 1001)

signature: sltr $r1, $r2

R-type

If the main register is less than register $r1, a flag is set to 1 in register $r2. If the main register is equal to or greater than register $r1, a flag is set to 0 in register $r2.

**sub – subtract** (op = 0000, func = 1010)

signature: sub $r1, $r2

R-type

Does operation $m = r1 – r2.

**beqz – branch if zero** (op = 0000, func = 1011)

Signature: beqz <12 bit imm.>

I-type

If the main register equals zero, branches. This branch command can branch ±2048 instructions from PC+1

**bnez– branch if zero** (op = 0000, func = 1100)

Signature: bnez <12 bit imm.>

I-type

If the main register does not zero, branches. This branch command can branch ±2048 instructions from PC+1

**Pseudo Instructions**

**Jal – Jump and Link**

Jump and link jumps to an immediate after backing up the return address and setting a new return address in $ra.

**Instructions**:

swap $m, $ra # saves $m # 0000 0101 000 001 xx

sw $sp[0] # puts the return address on the stack. # 0110 010 000000000

swap $m, $sp # 0000 0101 000 010 xx

addi 1 # increments the stack pointer # 0001 000000000001

swap $m, $sp # gets $ra back. # 0000 0101 000 010 xx

Swap $m, $ra # gets original M back. # 0000 0101 000 001 xx

j <12 bit imm.> # jump to the given immediate # 0100 <12 bit imm.>