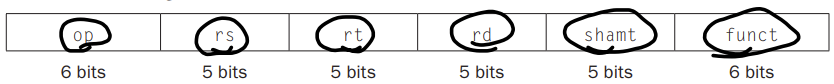
# Things to figure out:

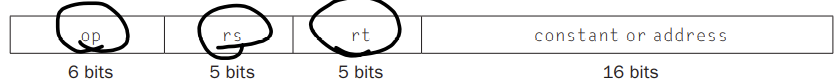
* High level overview
* Avalon bus/stalls/timing logic
* How to execute each instruction, components needed for each
* Decide Verilog modules and assign them to people to implement and test

# Instructions and the components they use:

**R-type:**

****

**I-type:**

****

**GOD LEVEL DOCUMENT:**

<https://www.cs.cmu.edu/afs/cs/academic/class/15740-f97/public/doc/mips-isa.pdf>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Instruction | Meaning | Example and equation | R, I, J | ALU, JB, or LS |  |
| ADDIU | Add immediate unsigned (no overflow) | ADDIU rt, rs, immediate  rt ← rs + immediate | I | ALU |  |
| ADDU | Add unsigned (no overflow) | ADDU rd, rs, rt  rd ← rs + rt | R | ALU |  |
| AND | Bitwise and | AND rd, rs, rt  rd ← rs AND rt | R | ALU |  |
| ANDI | Bitwise and immediate | ANDI rt, rs, immediate  rt ← rs AND immediate | I | ALU |  |
| BEQ | Branch on equal | BEQ rs, rt, offset  if (rs = rt) then branch | I | JB |  |
| BGEZ | Branch on greater than or equal to 0 | BGEZ rs, offset  if (rs ≥ 0) then branch | I | JB |  |
| BGEZAL | Branch on non-negative and link | BGEZAL rs, offset  if (rs ≥ 0) then procedure\_call | I | JB |  |
| BGTZ | Branch on greater than zero | BGTZ rs, offset  if (rs > 0) then branch | I | JB |  |
| BLEZ | Branch on less than or equal to 0 | BLEZ rs, offset  if (rs ≤ 0) then branch | I | JB |  |
| BLTZ | Branch on less than zero | BLTZ rs, offset  if (rs < 0) then branch | I | JB |  |
| BLTZAL | Branch on less than zero and link | BLTZAL rs, offset  if (rs < 0) then procedure\_call | I | JB |  |
| BNE | Branch on not equal | BNE rs, rt, offset  if (rs ≠ rt) then branch | I | JB |  |
| DIV | Divide | DIV rs, rt  (LO, HI) ← rs / rt | R | ALU |  |
| DIVU | Divide Unsigned | DIVU rs, rt  (LO, HI) ← rs / rt | R | ALU |  |
| J | Jump | J target | J | JB |  |
| JALR | Jump and link register | JALR rd, rs  rd ← return\_addr, PC ← rs | R | JB |  |
| JAL | Jump and link | JAL target | J | JB |  |
| JR | Jump register |  | R | JB |  |
| LB | Load byte |  | I | LS |  |
| LBU | Load byte unsigned |  | I | LS |  |
| LH | Load half-word |  | I | LS |  |
| LHU | Load half-word unsigned |  | I | LS |  |
| LUI | Load upper immediate |  | I | LS |  |
| LW | Load word |  | I | LS |  |
| LWL | Load word left |  | I | LS |  |
| LWR | Load word right |  | I | LS |  |
| MTHI | Move to HI |  | R | ALU |  |
| MTLO | Move to LO |  | R | ALU |  |
| MULT | Multiply |  | R | ALU |  |
| MULTU | Multiply unsigned |  | R | ALU |  |
| OR | Bitwise or |  | R | ALU |  |
| ORI | Bitwise or immediate |  | I | ALU |  |
| SB | Store byte |  | I | LS |  |
| SH | Store half-word |  | I | LS |  |
| SLL | Shift left logical |  | R | ALU |  |
| SLLV | Shift left-logical variable |  | R | ALU |  |
| SLT | Set on the less than (signed) |  | R | ALU/Next |  |
| SLTI | Set on less than immediate (signed) |  | I | ALU/Next |  |
| SLTIU | Set on less than immediate unsigned |  | I | ALU/Next |  |
| SLTU | Set on less than unsigned |  | R | ALU/Next |  |
| SRA | Shift right arithmetic |  | R | ALU |  |
| SRAV | Shift right arithmetic variable |  | R | ALU |  |
| SRL | Shift right logical |  | R | ALU |  |
| SRLV | Shift right logical variable |  | R | ALU |  |
| SUBU | Subtract unsigned |  | R | ALU |  |
| SW | Store Word |  | I | LS |  |
| XOR | Bitwise exclusive or |  | R | ALU |  |
| XORI | Bitwise exclusive or immediate |  | I | ALU |  |

# Components:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Component | Inputs | Outputs | Instructions implemented | Group member |
| ALU | Instruction (32)  Input 1 (32)  Input 2(32) | Output (32) |  |  |
| Load/Store Block | Clk (1) |  |  |  |
| Register File |  |  |  |  |
| Next Instruction | Register input |  |  |  |
| Instruction Decode | Instruction (32) |  |  |  |
| State Machine | Waitrequest (1) | State (2) |  |  |

## Instruction Decode Block

## ALU:

### Inputs:

* Input 1 (32-bit)
* Input 2 (32-bit)
* Instruction (32-bit)

### Outputs:

* Output (32-bit)

## PC:

### Input:

* Address (32-bit)

### Output:

* Next address (32 bit)

## Next instruction:

### Inputs:

* Instruction (32-bit)
* PC (32-bit)
* Register input 1 (32-bit)
* Register input 2 (32-bit)

## Output