# 0. Team

Qianjun Li

# Introduction

Overall, what I am planning to do is a register-register/load-store like a standard MIP processor without pipelining.

# Architectural Overview

# 

# Machine Specification

## Instruction formats

I don’t use type bits so type here is only for categorization, pt codes are 3 or 4 bits,(some peek the first bit of immediate) and all 4 bits are used in the lookup table for decoding.

|  |  |  |
| --- | --- | --- |
| **TYPE** | **FORMAT** | **CORRESPONDING INSTRUCTIONS** |
| R | 3 bits opcode, 3 bit first operand register, 3 bit second operand register, destination register is always a default register R7, | add, |
| B | 4 bits opcode, 5 bit for branch look up | beq |
| Mem | 4 bits opcode, 5 bit immediate for offset | lw, sw, lwi |
| xor | 4 bits opcode, 2 bits unused,3 bits operand | xor, xorb |
| I | 4 bits opcode (peek the first bit of the first operand), 3 immediate, 3 bit operand register, destination register is always a default register R7, | addi, sl,sr |
| Mov | 3 bits opcode, 3 bits first operand (mov to), 3 bits second operand(mov from) | mov |
| Sys | 4 bits opcode, used tell that the program has end | exit |

## Operations

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NAME** | **TYPE** | **BIT BREAKDOWN** | **EXAMPLE** | **NOTES** |
| ADD  000 | R | b8:b6 => optcode  b5:b3 => operand 1  b2:b0 => operand 2 | ADD R2 R3  R2+R3 = R7 | add will take values in registers specified by two operands and always store the result into register R7, move instruction later will help to move value in R7 to proper location for calculation afterwards. |
| XOR  0010 | xor | b8:b5 => optcode  b2:b0 => operand 2 | XOR R3  R3 = R6 ^ R7 | xor will always use value from R6 and R7 as operands, R3 is destination register  wr\_defualt is used to determine whether to use default destination R7 or the register 2:0 specified by instruction.  rd\_defualt is used to determine whether to use default operands R6, R7 or registers specified by 5:0 of instruction. This means RD1 default is r6 and RD2 default is r7  These signals are determined by control unit and pass to register files |
| XORB  0011 | xor | b8:b5 => optcode  b2:b0 => operand 2 | XORB R3    R3 = xor all bits in R7 | xorb will always use value from and R7 as operand, R3 is destination register. Similar to xor, this will use default operand register R6, R7, while passing a ALUop so that alu ignore R6 and do a bus xor on R7 |
| LW  0100 | Mem | b8:b5 => optcode  b4:b0 => Immediate | LW 7  Load from memory address stored in R6 + 7 and put the result in R7 (default register) | Always use R6(default, can’t change) + Immediate as memory address. Put result from memory in R7(default, can’t change) |
| LWI  0101 | Mem | b8:b5 => optcode  b4:b0 => Immediate | LWI 7  R7 = 7 | Used to put an immediate value in R7, immediate does go through ALU but the ALU simply ignore ALUin1 and just pass through this immediate. |
| SW  0110 | Mem | b8:b5 => optcode  b4:b0 => Immediate | SW 7  Save to memory address R6 + 7 the content in R7 (default register) | Always use R6 + Immediate as memory address, then put in what is in R7 |
| EXIT  0111 | Sys | b8:b5 => optcode  b4:b0 => Immediate | EXIT 7  exit with a code 7 | Used at the end of a program |
| BRE  1000 | Branch | b8:b5 => optcode  b4:b0 => Immediate | BRE 16  If R6 = R7, brach to result of LUT giving input of 16 | Value in R6 and R7, specifically, whether they are equal are always used for branch condition, so it is needed to put appropriate value in them before branch using MOV |
| BRNE  1001 | Brachn | b8:b5 => optcode  b4:b0 => Immediate | BRNE 10  If R6 != R7, brach to result of LUT giving input of 16 | Values in R6 and R7 are always used for branch condition, same as BRE except that it branches when they are not equal. |
| ADDI  101 | I | b8:b5 => optcode  b4:b3 => Immediate  b2:b0 => operand 2 | addi +1 R3  R7 = R3+1  addi -1 R3  R7 = R3-1 | This signed immediate will be converted to a 3 bit code using the assembler. First bit is used to denote the sign: 0 for +, 1 for -. Second to third bit is used to denote (value - 1). This is because shift 0 is a redundant code, so my code will map +1 to 000 and -1 to 100, +2 to 001, -2 to 101… This way the first bit of the immediate is also used in opt code. Its operation will always be operand +/- immediate. Thus, valid value for immediate should be [-1,-2,-3,-4,1,2,3,4]. The result always go to default register R7 |
| SHIFTI  110 | I | b8:b5 => optcode  b4:b3 => Immediate  b2:b0 => operand 2 | SHIFT +1 R3  R7 = R3<<1  SHIFT -1 R3  R7 = R3>>1 | This is the same setting as for ADDI, +k is shift left k times, -k is shift right k times. These shifts are logical shifts. |
| MOV  111 | move | b8:b6 => optcode  b5:b3 => operand 1  b2:b0 => operand 2 | MOV R2 R3  R3 = R2 | The instruction needs an extra mux to select from the typical result of ALU or mem compared to the direct value from register. See RegWriteSrc signal and mux controlled by it in picture. |

## Internal Operands

5 general purpose registers, 3 special registers : R0, R6, R7.

wr\_defualt is used to determine whether to use default destination R7 or the register 2:0 specified by instruction.

rd\_defualt is used to determine whether to use default operands R6, R7 or registers specified by 5:3 and 2:0 of instruction. This means RD1 default is r6 and RD2 default is r7

These signals are determined by the control unit and passed to register files.

R0 will always have a value 128 whenever a program starts. This is expected to be used as a base for memory location storing testing,or intermediate data.

## Control Flow (branches)

Direct branching, branches are looked up in LUT and go directly into the PC. This means in total 5 bits = 32 different branches are supported, and since we can pre program LUT, there is no limit on the distance of a branch. Programmers should fill in branch LUT as they program. For example after finishing programming, 2 branches are needed, first one to line 0, second one to line 72, both are BRE. Two branch instructions should simply be BRE 0 and BRE 1. And the LUT should have LUT[0] = 0, LUT[1] = 72. My branch LUT will function very similarly to a ROM. \\ updated, now assembler will tell you where each branch target is, just make sure to use a different branch label value when you have a new branch target. need to specify by this //branch destination 0

## Addressing Modes

Indirect addressing is calculated as base + offset, base is always value stored in R6 and result is put in R7.

# Programmer's Model [Lite]

Since we have enough registers, 6 general purpose, 2 special use, programmers should always try to load as many into registers before calculation. Since many instructions now have a default destination register R7, after such calculation, programmers should use mov to move the value of R7 to proper location for later use.

We cannot simply copy MIPS instruction

Reason one: all instructions have at most 2 operands now, we need to use mov to adjust positions.

Reason two: some of the order of the operands are different from MIPS, for example addi, shifti: immediate value needs to be put in front of the operand, next to opcode, because the first bit of it is used in opcode decoding to determine left/right shift.