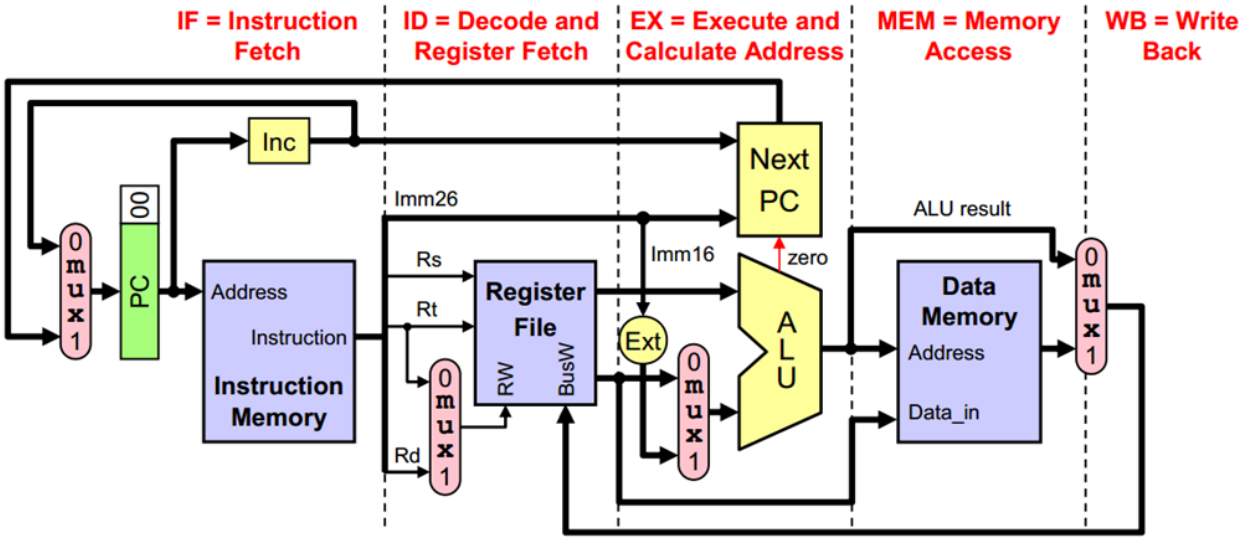
Team: William Sun

Introduction:

1. Architecture Name: FECES (Forward Error Correction Efficient Style)
2. Load-Store: Use fixed-encoding architecture like MIPS

Architectural Overview:

Use MIPS as reference for now.



Machine Specification:

1. Instruction Formats (2 bits)
    a. Memory, Comparison, and Jump Operations (00)
        i. opcode(2'b) specifier(3'b) $rt(2'b) $rd(2'b)
        ii. ex: move $t, $s
    b. 1 Variable Operations (01)
        i. opcode(2'b) specifier(1'b) imm(6'b)
        ii. opcode(2'b) specifier(1'b) $rt(2'b) extra(4'b)
        iii. ex: set i
    c. 2 Variable Operations (10)
        i. opcode(2'b) specifier(3'b) $rt(2'b) $rd(2'b)
        ii. ex: add $s, $t
    d. Branch Operations (11)
        i. opcode(2'b) specifier(1'b) $rs(2'b) $rt(2'b) $ru(2'b)
        ii. ex: beq $s, label
2. Operations
    a. Memory, Comparison, Jump, and Extra Operations (00)
        i. 00_000 → get $t $s; $t = REGISTER[$s]
        ii. 00_001 → put $t $s; REGISTER[$t] = $s

iii. 00_010 → lw $t $s; $t = MEM[$s]
iv. 00_011 → sw $t $s; MEM[$t] = $s
v. 00_100 → sgt $s $t; $s = ($s > $t)

b. 1 Variable (01)
i. 01_0 → set i; $r3 = SE(i) // NOTE: hardcode r3
ii. 01_1 → not $d; $d = ~($d)

c. 2 Variable (10)
i. 10_000 → add $s $t; $s = $s + $t
ii. 10_010 → sub $s $t; $s = $s - $t
iii. 10_011 → and $s $t; $s = $s & $t
iv. 10_100 → or $s $t; $s = $s | $t
v. 10_101 → xor $s $t; $s = ^($t)
vi. 10_110 → sleft $t $s; $t = $t << $s
vii. 10_111 → sright $t $s; $t = $t >> $s

d. Branch (11)
i. 11_0 → beq $s $t $u; if ($s == $t) pc = $u
ii. 11_1 → bne $s $t $u; if ($s != $t) pc = $u

3. Internal Operands
   a. Total: 16 registers
   b. Temporary Use: 4 registers (r0 - r3)
      i. During operation, use r0 - r3 as operands
      ii. Use mov, lw, sw to move result into general storage
   c. General Storage: 12 registers (r4 - r15)
4. Control Flow
   a. Type
      i. Branch on equals to save bits, can also use jump operation
   b. Target Address
      i. Load in address from register
   c. Maximum Distance
      i. Register data is 8 bits. Can jump either up or down, so data must be signed. Thus, range is i < 2^7 = -128 to 128, and max distance is 128.
5. Addressing Modes
   a. Register-indirect addressing
      i. Load and store data via contents of register
      ii. lw, sw

Changelog Milestone 2

- Reorganize ISA opcode based on instruction format
  - Allows for more instructions
  - Allows for larger immediate values
- Change 2 registers to 4
- Update control flow based on ISA changes
- Fix addressing modes issues

Changelog Milestone 3

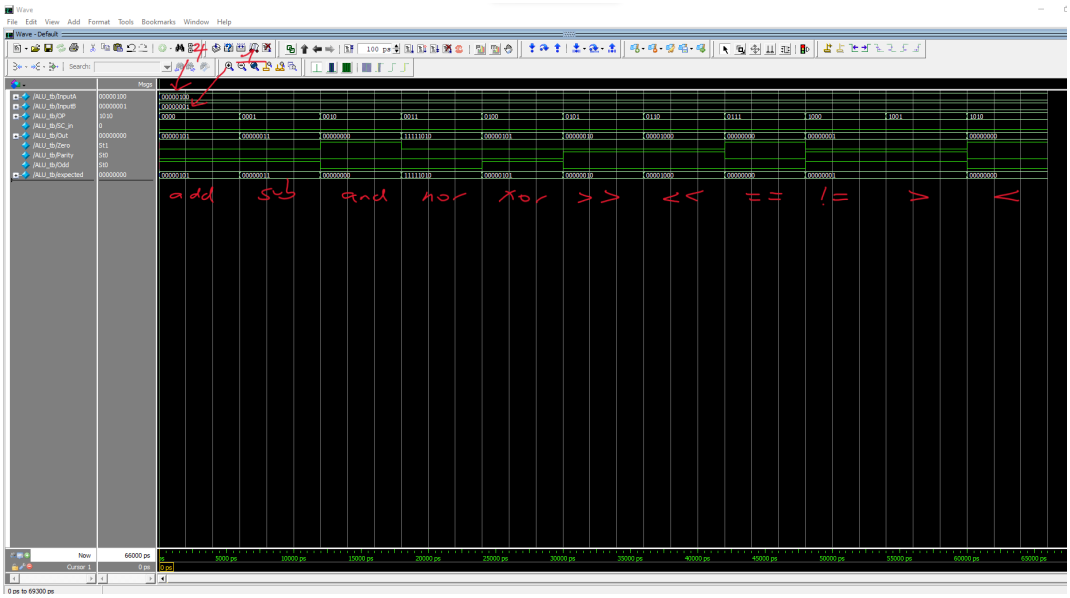
- Distinguish between moving data from registers to memory (lw, sw) and moving data between working registers and general registers (put, get)
- Removed unnecessary operations in (00)
- Replaced NOR with OR in (10)
- Change XOR into bitwise operation in (10)
- Updated BEQ in (11) with register-indirect addressing
- Added BNE in (11)
- Rename shift operations
- Maximum branch distance updated to 128
- Update addressing mode to register-indirect
- Removed PC-relative addressing

Programmer's Model:

1. The programmer should think of this machine as trying to conserve as many bits as possible. With only 9 bits total, we must split up the instruction encoding wisely. We use 2 bits to first split between major types of instructions. That gives us enough bits to set large immediate values (2^6). With variable instructions, we can use up all 9 bits, so we can split those further into specific instructions to increase the total number of instructions we can encode. Additionally, the programmer should abuse the hardware and use NOR as a universal gate to recreate other operations to save bits. Although this will increase CPI, it makes it easier to fit the 9-bit instruction limit.
2. Example
    a. C → $r0 = $r0 + $r1
    b. Assembly → add $r0, $r1
    c. Machine → 10_000 (add) 00 (r0) 01 (r1)

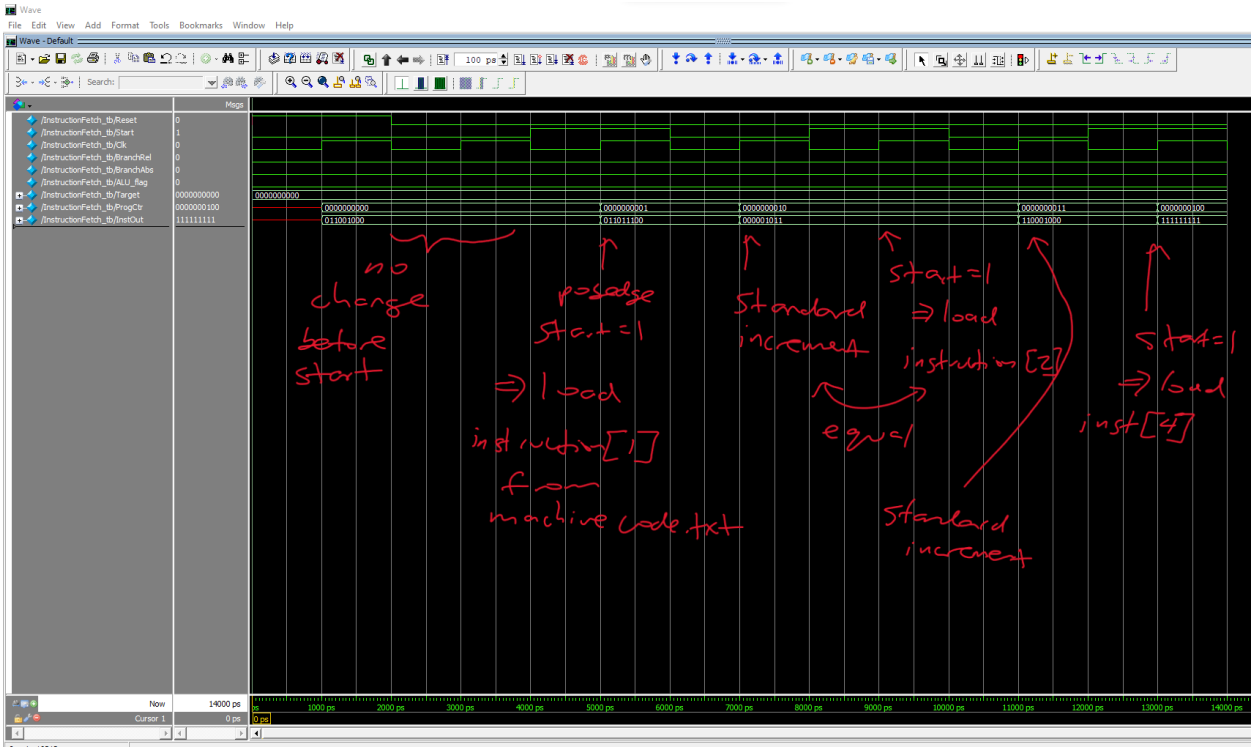
Working ALU:

```
VSIM 15> run -all
#                1000 YAY!! inputs = 04 01, OPcode = 0000, Zero 0
#                7000 YAY!! inputs = 04 01, OPcode = 0001, Zero 0
#               13000 YAY!! inputs = 04 01, OPcode = 0010, Zero 1
#               19000 YAY!! inputs = 04 01, OPcode = 0011, Zero 0
#               25000 YAY!! inputs = 04 01, OPcode = 0100, Zero 0
#               31000 YAY!! inputs = 04 01, OPcode = 0101, Zero 0
#               37000 YAY!! inputs = 04 01, OPcode = 0110, Zero 0
#               43000 YAY!! inputs = 04 01, OPcode = 0111, Zero 1
#               49000 YAY!! inputs = 04 01, OPcode = 1000, Zero 0
#               55000 YAY!! inputs = 04 01, OPcode = 1001, Zero 0
#               61000 YAY!! inputs = 04 01, OPcode = 1010, Zero 1
```



Working Instruction Fetch:

```
VSIM 7> run -all
# Check Reset
# Check nothing happens before Start
# Check first program was loaded
# Check second program was loaded
# Check third program was loaded
```



Milestone 2 Questions:

1. ALU Demonstration
    a. Demonstrated all 2 variable (opcode 10) and comparison (opcode 00) operations
2. Register File
    a. Using starter code
    b. Updated it to initialize 16 registers
3. ALU Non-Arithmetic Instructions
    a. No, my ALU only does arithmetic