CO-224

Lab 5 Part 5

E/16/388

Weerasundara W.M.T.M.P.B.

1. Instruction encoding.

- All instructions are 32 bit wide.
- Each instruction has 4 fields with 8 bits for each.
    I. OP-CODE field identifies the instruction's operation. This should be used by the control logic to interpret the remaining fields and derive the control signals
    II. DESTINATION field specifies either the register to be written to in the register file, or an immediate value (jump or branch target offset).
    III. SOURCE 1 field specifies the 1st operand to be read from the register file.
    IV. SOURCE 2 is the 2nd operand from the register file, or an immediate value (loadi).

Below is instruction format

OPCODE(31-24)	DESTINATION(23-16)	SOURCE1(15-8)	SOURCE2(7-0)

2.OPCODES and other control signals

Opcode decode process happens in control unit module inside the cpu. It checks OPCODE and set control signals accordingly.

Control signals are,

I. ALUOP – Decides ALU operation and results
II. Mux_minus – choose between 2's complement value or the original value store in a register.
III. Mux_immediate – choose between immediate value input or the value stored in a register
IV. WRITEENABLE – choose to write to or not to the destination register at next positive clock edge
V. Branch – used for beq
VI. Jump – used for j
VII. b_notequal - used for bne

ALUOPs are,

I. 0000 – loadi
II. 0001 – add/sub
III. 0010 – bitwise &
IV. 0011 – bitwise or
V. 0100 – logical shift right
VI. 0101 - logical shift left
VII. 0110 – rotate right
VIII. 0111 – rithmatic shift right
IX. 1000 - multiplication

Add, sub, and, or, j, beq, mov, loadi instructions are implemented inside the ALU.

- Add instruction

OPCODE-8'b00000010

ALUOP = 4'b0001;

mux_minus = 0; (select positive value)

mux_immediate = 0; (select register value)

WRITEENABLE = 1;

branch = 0;

jump = 0;

b_notequal = 0;

- Sub instruction

  Opcode:8'b00000011:

  ALUOP = 4'b0001;

  mux_minus = 1;

  mux_immediate = 0;

  WRITEENABLE = 1;

  branch = 0;

  jump = 0;

  b_notequal = 0;

- And instruction

  Opcode: 8'b00000100:

  ALUOP = 4'b0010;

  mux_minus = 0;

  mux_immediate = 0;

  WRITEENABLE = 1;

  branch = 0;

  jump = 0;

b_notequal = 0;

- Or Instruction
  Opcode: 8'b00000101:

  ALUOP = 4'b0011;

  mux_minus = 0;

  mux_immediate = 0;

  branch = 0;

  jump = 0;

  b_notequal = 0;

- Mov instruction

  Opcode: 8'b00000001:

  ALUOP = 4'b0000;

  mux_minus = 0;

  mux_immediate = 0;

  WRITEENABLE = 1;

  branch = 0;

  jump = 0;

  b_notequal = 0;

- Lodi instruction
  Opcode: 8'b00000000:

  ALUOP = 4'b0000;

  mux_immediate = 1

  WRITEENABLE = 1;

  branch = 0;

  jump = 0;

  b_notequal = 0;

Beq and jump instruction has different datapaths.

- Beq instruction
  Opcode: 8'b00000111

  branch = 1;

  mux_minus = 1;

  mux_immediate = 0;

  WRITEENABLE = 0;

  jump = 0;

  ALUOP = 4'b0001;

  b_notequal = 0;

- Jump instruction
  Opcode: 8'b00000110
  register operands, ALU operation, doesn't matter. Therefore mux control signals and ALUOP don't care.
  Also do not write ALU results to register file therefore WRITEENABLE = 0.

  jump = 1;

  b_notequal = 0;

- Bne instruction
  OPCODE:8'b00001000: /

b_notequal = 1;

mux_minus = 1; need minus operand for subtract

mux_immediate = 0; select register operand as positive operand

WRITEENABLE = 0; don’t write results to reg file

jump = 0;

ALUOP = 4'b0001; for 2s complement addition

Below 4 control settings are for shift operations. They are not branch or jump instructions therefor all branch/jump signals are 0.

Results are written to reg file therefore WRITEENABLE = 1 for all shift operations. They require unique ALUOP signals to decide ALU functionality.Mux to select minus value is don’t care because we use immediate value sprightly.

Also the shift operations listed below are implemented inside the ALU.

- logical shift right instruction
  Opcode: 8'b00001001:

  ALUOP = 4'b0100;

  mux_immediate = 1; to get immediate value (shift amount)

  WRITEENABLE = 1;

  branch = 0;

jump = 0;

b_notequal = 0;

- logical shift left

  Opcode: 8'b00001010

  ALUOP = 4'b0101;

  mux_immediate = 1; to get immediate value (shift amount)

  WRITEENABLE = 1;

  branch = 0;

  jump = 0;

  b_notequal = 0;

- Rotate right instruction
  Opcode: 8'b00001011

  ALUOP = 4'b0110;

  mux_immediate = 1; to get immediate value (rotate amount)

  WRITEENABLE = 1;

  branch = 0;

  jump = 0;

  b_notequal = 0;

- Arithmetic shift right operation
  Opcode: 8'b00001100

ALUOP = 4'b0111;

mux_immediate = 1; to get immediate value (shift amount)

WRITEENABLE = 1;

branch = 0;

jump = 0;

b_notequal = 0;

- multiplication

Multiplication is done inside the ALU using submodule multiplier. It uses register operands and write results to regfile. Don't do any jumps or branches.

Opcode8'b00001101

ALUOP = 4'b1000;

mux_immediate = 0; use original register operands

mux_minus = 0;

WRITEENABLE = 1; writes the results to destination

branch = 0;

jump = 0;

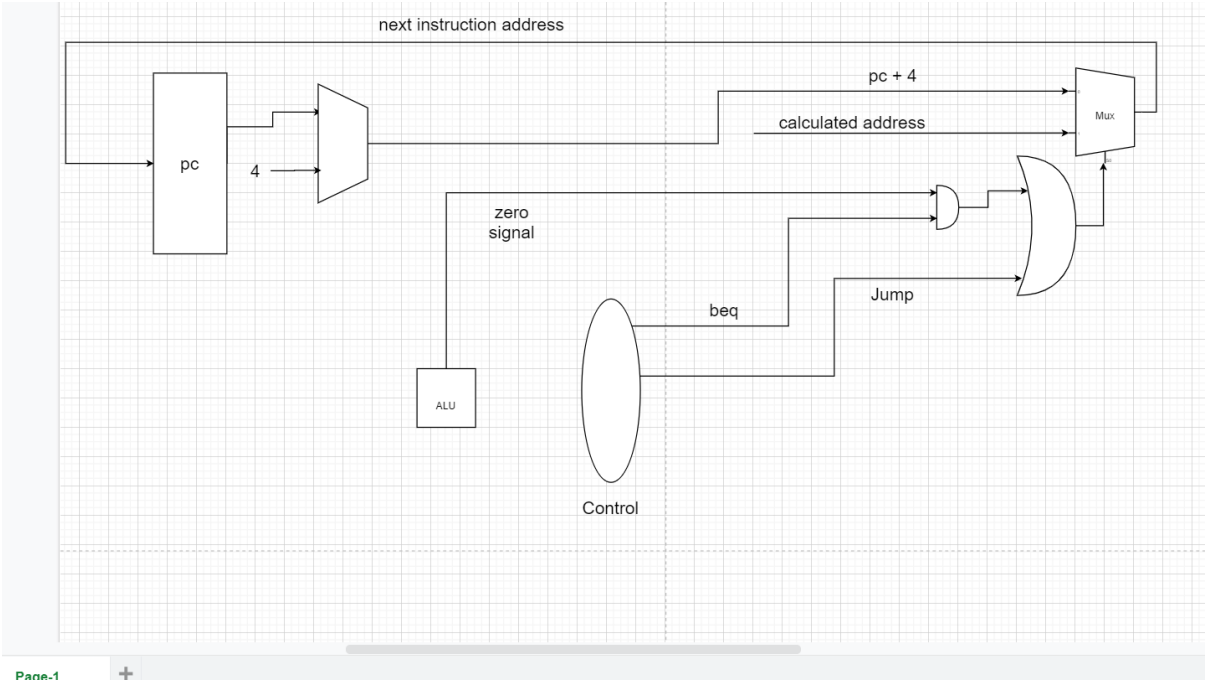
b_notequal = 0;

2. Changes made to data path and control

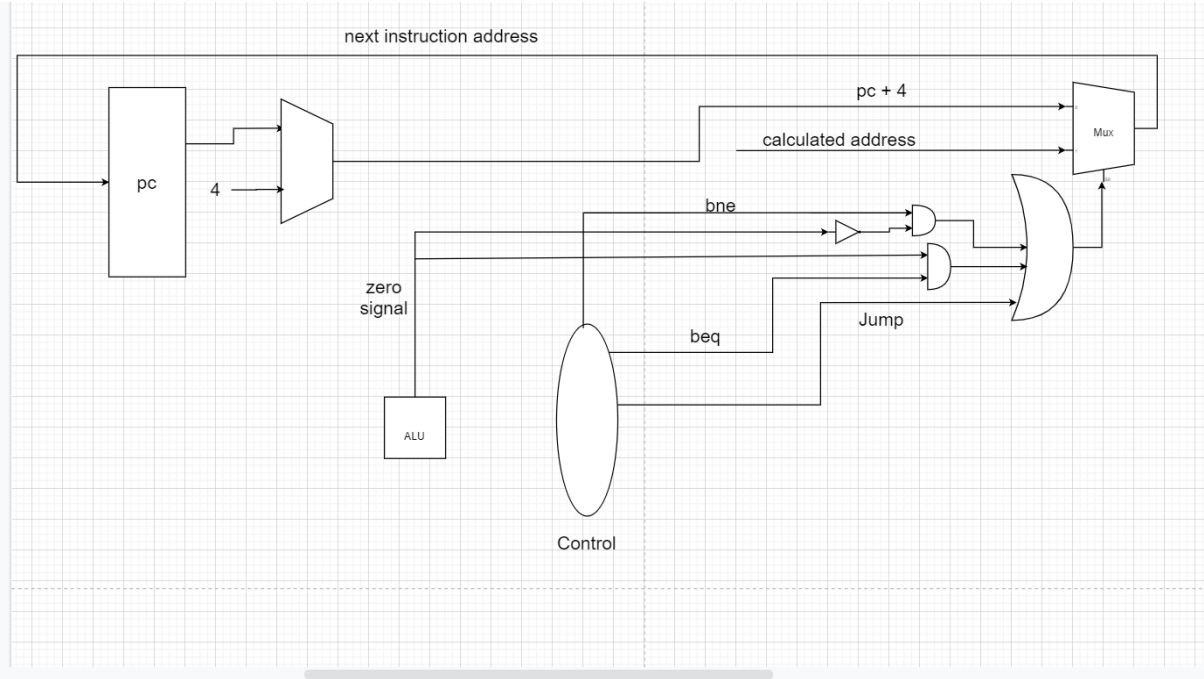
For part 5, only because of bne, data paths were updated. All the shift operations and multiplication operation was implemented inside the ALU module. Therefore they didn't affect the data path.

- Before adding bne data path

  Branch/ jump target depends on j/branch control signal and zero calue.
  Select = jump + (beq & zero)



- After implementing bne instruction



Mux output get affected due to bne control signal.

Select = jump + (beq & zero) + (bne & ~(zero))

These are the changes made to the data path in this step.

For a bne, both zero signal from ALU and control signal bne is required.

Functionality is opposite to previously implemented beq.

- Changes made to control

Before implement part 5

I. ALUOP – Decides ALU operation and results
II. Mux_minus – choose between 2's complement value or the original value store in a register.
III. Mux_immediate – choose between immediate value input or the value stored in a register
IV. WRITEENABLE – choose to write to or not to the destination register at next positive clock edge
V. Branch – used for beq
VI. Jump – used for j

These control signals were used.
Then bne signal was added.

ALUOP signals before part 5

I. 0000 – loadi
II. 0001 – add/sub
III. 0010 – bitwise &
IV. 0011 – bitwise or

After implementing, these signals were added

I. 0100 – logical shift right
II. 0101 - logical shift left
III. 0110 – rotate right
IV. 0111 – arithmatic shift right
V. 1000 – multiplication

Zero signal from the ALU didn't change.