Description of Our Design Implementation

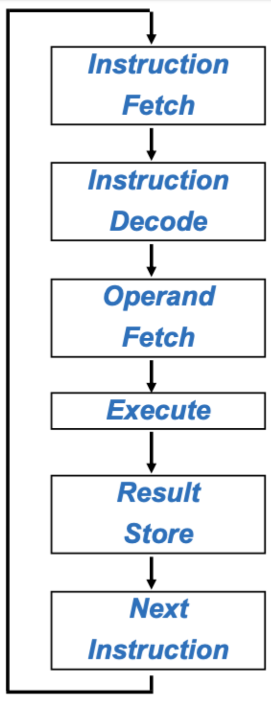
- ECE 550 Project Checkpoint5
- Name: Rui Cao, netID: rc384
- Name: Yanxin Li, netID: yl869

1. Overall Design

- The overall design is consistent with checkpoint 4. Compared with checkpoint 4, only the implementation of 7 additional instructions in processor.v and some control signals have been added in control.v.
- Processor: processor.v
  - including steps of The von Neumann Model
  - using dffe.v, control.v, and signExtend.v
- Skeleton: skeleton.v
  - output four clocks (imem_clock, dmem_clock, processor_clock, and regfile_clock)
- PC which outputs the address of instructions in Imem: dffe.v
  - 32-bit DFFE for PC
- Get control signals from instruction machine code : control.v
- Sign-extend module for immediate: signExtend.v
- Clock divider by 4: clock_divider_by2.v
- imem.v and dmem.v
  - generates the dmem and imem files by generating Quartus syncram components
- alu.v and regfile.v
  - provided by Resource folder on Sakai

2. Detailed Description of Processor

Our processor module is implemented according to steps of The von Neumann Model (the figure below).



- Instruction Fetch:
  Read instruction bits from memory
- Decode:
  Figure out what those bits mean
- Operand Fetch:
  Read registers (+ mem to get sources)
- Execute:
  Do the actual operation (e.g., add the numbers)
- Result Store:
  Write result to register or memory
- Next Instruction:
  Figure out mem addr of next insn, repeat

2.1. Instruction Fetch:

- First, we get the address of instructions in Imem from 32-bit DFFE (32-bit PC).
- Second, compute the address of the next instruction.
  - PC + 1
  - or T: jal T or bex T (& $rstatus != 0) or j T
  - or PC + 1 + N: bne $rd, $rs, N (& $rd != $rs) or blt $rd, $rs, N (& $rd < $rs)
  - or $rd: jr $rd
- Third, get the instruction address in Imem (address_imem) from the first 12 bits of 32-bit PC.

2.2. Instruction Decode:

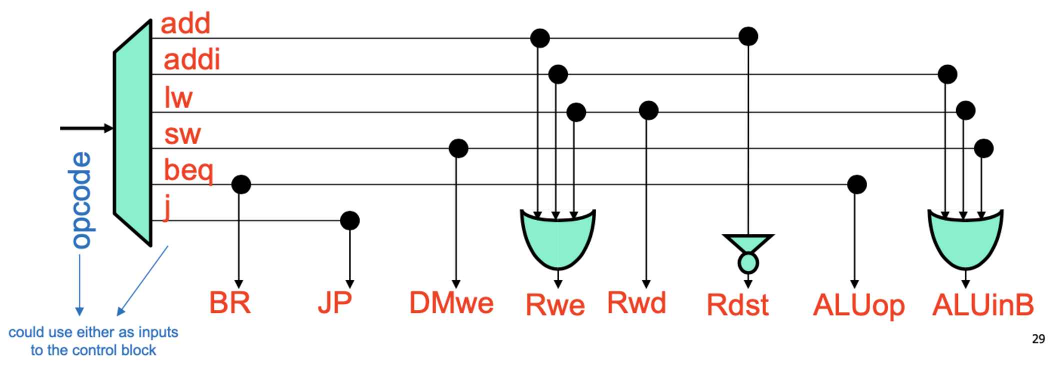
- According to the Instruction Machine Code Format of PDF, figure out what those bits in Instruction Machine Code (q_imem) mean.

Instruction Type	Instruction Format						
R	Opcode [31:27]	$rd [26:22]	$rs [21:17]	$rt [16:12]	shamt [11:7]	ALU op [6:2]	Zeroes [1:0]
I	Opcode [31:27]	$rd [26:22]	$rs [21:17]	Immediate (N) [16:0]			
JI	Opcode [31:27]	Target (T) [26:0]					
JII	Opcode [31:27]	$rd [26:22]	Zeroes [21:0]				

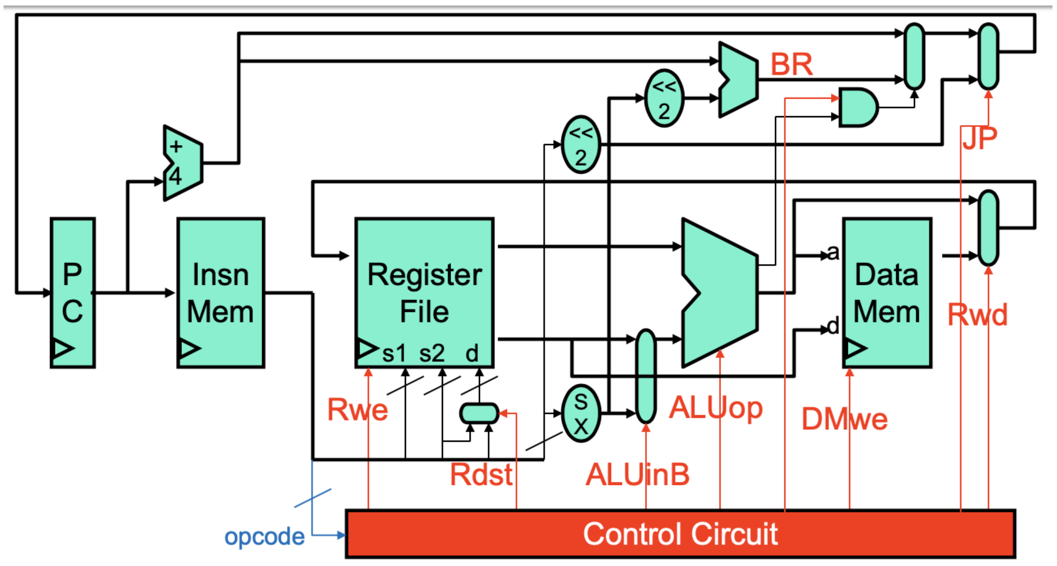
- Besides, r30 (rstatus) is $30. r0 is $0. r31 is $31

2.3. Control Circuit (using control.v):

- We implement control circuit using “Random Logic” provided by our PPT 07 Page 29 (as shown in the figure below).



- First, get 1-bit typeR (whether this instruction is type R or not) using 5-bit opcode and AND gate. The same method is used for isAddi, isSw, isLw, isJ, isJal, isBlt, isBne, isJr, isSetx. Second, get 1-bit isAdd and isSub using 5-bit aluOp, 1-bit typeR, and AND gate. Third, get control signals, such as Rwe, DMwe, and Rdst, using results obtained above and OR gate.
- The usage of each control signal is shown in the figure below (The following steps will use them).



2.4. Operand Fetch:

- ctrl_readRegA:
    - $rd: bne or jr or blt
    - or $rstatus($r30): bex
    - or $rs
- ctrl_readRegB:
    - $rt
    - $rd: sw
    - or $r0 (0): bex
    - or $rs: bne or blt

2.5. Execution (using alu.v and signExtend.v):

- 17-bit immediate will become 32-bit after using signExtend module. data_operandA is directly the input data_readRegA, while data_operandB dependends on the control signal ALUinB and it is either 32-bit immediate or the input data_readRegB.
- Besides, ctrl_ALUopcode dependends on whether the instruction is type R.

2.6. Result Store:

- Write result to register:
    - The output ctrl_writeEnable is the control signal Rwe.
    - Note: overflow will affect ctrl_writeReg and data_writeReg. If overflow occurs, ctrl_writeReg will be r30. ctrl_writeReg:
        - $rd
        - or $r30: overflow or setx
        - or $r31: jal
    - When the instruction is lw, data_writeReg is the output of dmem (q_dmem). data_writeReg:
        - overflow data: 1 or 2 or 3
        - or data_result from alu
        - or q_dmem: lw
        - or T: setx

        - or PC + 1: jal

- Write result to memory:
    - The data which is written to dmem is data_readRegB.
    - The output wren is the control signal DMwe.