CSC3050 Project4 Report

1. Blueprint

   This project aims to design a 5-stage pipelined CPU. The design idea consists of 5 parts:

   1. Analyze the need for each instruction, and derive the demond for the datapath.
   2. Choose the proper component for the datapath.
   3. Connect the component and form the datapath.
   4. Analysis the need for each instruction, so as to determine the control signals
   5. Integrate control signals to form a complete data path.

2. Analysis all instruction:

   - addu R[rd] <- R[rs] + R[rt] ; PC <- PC + 4 ;
   - add R[rd] <- R[rs] + R[rt] ; PC <- PC + 4 ; set OF = 1 if overflow
   - subu R[rd] <- R[rs] - R[rt] ; PC <- PC + 4 ;
   - sub R[rd] <- R[rs] - R[rt] ; PC <- PC + 4 ; set OF = 1 if overflow
   - addi R[rt] <- R[rs] + SignExt(imm); PC <- PC + 4; set OF = 1 if overflow
   - addiu R[rt] <- R[rs] + SignExt(imm); PC <- PC + 4
   - and R[rd] <- R[rs] & R[rt] ; PC <- PC + 4;
   - nor R[rd] <- R[rs] ~| R[rt] ; PC <- PC + 4;
   - or R[rd] <- R[rs] | R[rt] ; PC <- PC + 4;
   - xor R[rd] <- R[rs] ^ R[rt] ; PC <- PC + 4;
   - andi R[rt] <- R[rs] & ZeroExt(imm); PC <- PC + 4
   - ori R[rt] <- R[rs] | ZeroExt(imm); PC <- PC + 4
   - xori R[rt] <- R[rs] ^ ZeroExt(imm); PC <- PC + 4
   - sll R[rd] <- R[rt] << shamt (logic); PC <- PC + 4
   - sllv R[rd] <- R[rt] << R[rs] [4:0] (logic); PC <- PC + 4
   - srl R[rd] <- R[rt] >> shamt (logic); PC <- PC + 4
   - srlv R[rd] <- R[rt] >> R[rs] [4:0] (logic); PC <- PC+4
   - sra R[rd] <- R[rt] >> shamt (arithmetic) ; PC <- PC+4
   - srav R[rd] <- R[rt] >> R[rs] [4:0] (arithmetic) ; PC <- PC + 4

   - slt R[rd] <- (R[rs] < R[rt]) ; PC <- PC+4
   - beq ZF <- (R[rs] - R[rt] == 0) ; if ZF = 1, PC = PC + 4 + SignExt(imm16 << 2), else PC = PC + 4
   - bne ZF <- (R[rs] - R[rt] == 0) ; if ZF = 0, PC = PC + 4 + SignExt(imm16 << 2), else PC = PC + 4
   - jr PC <- R[rs]
   - j PC <- {(PC+4)[31:28], target, 00}
   - jal R[ra] <- PC + 8; PC <- {(PC+4)[31:28],target, 00}

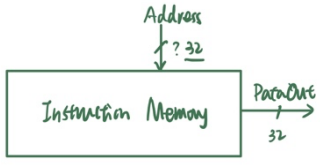
   - lw R[rt] <- a word in MEM[ddress]; PC <= PC + 4;
     - address = R[rs] + SignExt(1mm16)
   - sw MEM[address] (a word ) <- R[rt]; PC <= PC +4;

3. Devise important components:

   a). Instruction Memory

   The instruction memory has two functions: The first is to load instructions, and the second is to fetch the instruction given PC address. Initially, the module will load the machine code and store them in InstructionRAM. Each Instruction will be stored in 4 blocks. Each time the input PCF update, the InstructionRAM will automatically fetch the instruction using the address PCF.

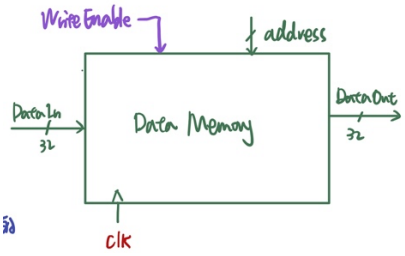
   The instruction memory is read-only, therefore, it only has an input wire and an output wire for input address and output instruction respectively.

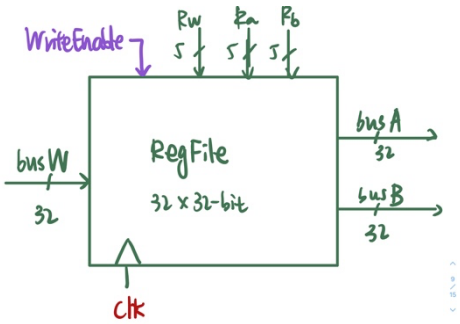


b). DataMemory

The Data Memory is used to store the data. MIPS instructions only access the data memory for two special instruction types, i.e. load and store.

The Data Memory is able to read and write data. The address and data in this DataMem is both 32-bit. Read occurs when input signal "address" changed, then DataMemory will read the data store in the address "address" and then pass the data information to the output wire "DataOut". Write, however, is clock sensitive. At the rising edge of the clock, if control signal "WriteEnable" then the DataMemory will write the data "DataIn" to the address "address".



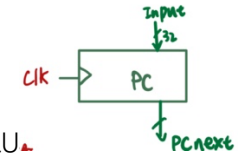




c). RegFile:

The RegFile contains 32 32-bit general registers. And the component supports two read wires and one write wire in the same time. So there will be two 5-bit wires "Ra", "Rb" as input to indicate the address of registers to read respectively, and one 5-bit wire "Rw" as input to indicate the address of the register to write. In addition, two 32-bit wire "busA", "busB" as output to show the registers intended to read, one 32-bit wire "busW" as input to show the registers intended to write in this RegFile

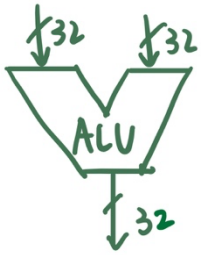
d) PC

PC is a special 32-bit register that store the current instruciton address. At each rising edge of clock, PC will update based on the input wire signal.



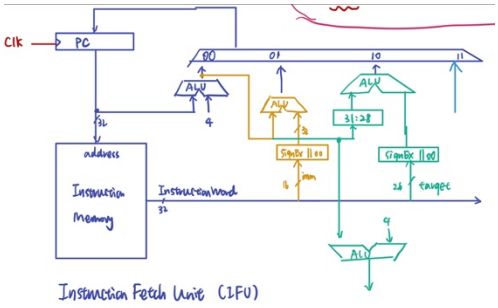

e) ALU

The ALU component support arithmetic/logic/compare opeartions. The two input operands are both 32 bit. They can be from the registers or the extended immediate value.

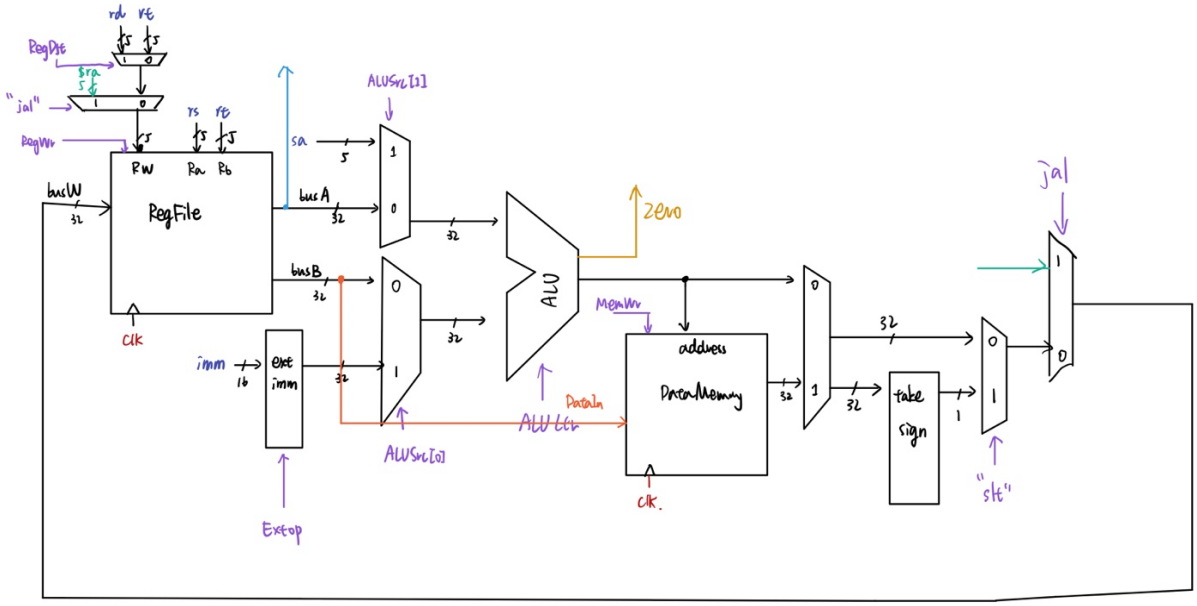



I have implemented modules for all the components mentioned above.

4. After finishing the construction of components, I need to find a data path that connect all the component. In the meantime, I need to devise and generate proper control signals so as to correctly select the method and component.




I mainly follow the datapath draw above. I need to connect all the modules based on wire and run them based on control signals.




The most important things is to device the control signals. Specifically, I deviced nPC_sel and ALUCtr based on the below diagram:

| ALUCtr | | | | | |
|---|---|---|---|---|---|
| 0000 | “ADD” | 0100 | “AND” | 1000 | “SLL” |
| 0001 | “SUB” | 0101 | “OR” | 1001 | “SLLV” |
| 0011 | “Slt” | 0110 | “XOR” | 1010 | “SRL” |
| | | 0111 | “NOR” | 1011 | “SRLV |
| | | | | 1100 | “SRA” |
| | | | | 1101 | “SEAV” |

| nPC_sel | PCupdate |
|---|---|
| 000 | PC+4 |
| 001 | j,jal |
| 010 | Branch(beq) |
| 011 | Branch(bne) |
| 1xx | jr |

5. Now I have completed a single cycled CPU. To have a pipelined CPU, I devide the datapath shown on the above into 5 stages: Fetch, Decode, Excecute, WrittenBack.
   I have finished all module mentioned above and connect them in CPU.v
   a). The Fetch stage composed of module PC and module InstrucitonRAM.
   At the rising edge of the clock, if input data PCIn signal changed, PC will update, and pass to the output wire PCF --- implemented by PC
   The output wire PCF by PC connect to the InstructionRAM as Input. PCF also

connect to an ALU as input. Once InstructionRAM detects any update on the input it will generate instruction automatically and pass to the output wire InstrD --- Implemented by InstrctionRAM.

In the same time, PCF will pass to an ALU component and increment by 4, the result PCplus4 will be as output

b).The decode stage will divide the instruction, generate the control signals, read the registers. And Extender will extend the imm16 to 32-bit. After completing stage I (FETCH) process, the output of stage I will be stored in registers Instruction and PCplus4

Instruction will be divided into function field such as opcode, func, rs, rt, rd, imm, target, sa. Based on opcode and func, the CONTROl component will generate control signals automatically given the Instruction signal changed. Based on rs, rt, the RegFile will read the registers, and pass out as wire busA, busB. Based on imm and the control signals, the extender will extend the imm using the right method, and pass out as wire extimm Based on control signal, the write address will be choose from rt and rd (or $ra), and pass out as wire Rw.

c). The execute stage will perform ALU operation so as to get the ALU result and the branch address. This part contains two ALU:

ALU1 (busA,busB,sa,Extimm --> ALUout, ZERO, (OF))

-- For Arithmetic/Logic/Shift Instructions

-- For Unconditionally Jump(beq/bne/slt) = "sub"

-- For lw/sw

ALU2 (PCplus4, target --> PCBranch)

-- For j, jal

PC update will also be implemented in this module.

d). Access Data Memory stage implements the memory access process, mainly for load and store instruction.

e). WriteBack stage implements Register Write Back process.

Finally, all the 5 stages will be integrated to realize the pipelined CPU.