EE309 Project Report

May 2024

1 Team

- Team ID: 28
- Ashwajit Singh (22b1227)
- Aditya Vema Reddy Kesari(22b3985)
- Suchet Gopal(22b1814)
- Raunak Mukherjee (22b3955)

2 Stages and components of the microprocessor

2.1 Instruction Fetch

- The instruction is fetched from instruction memory using the current value of program counter (stored in R0.) The program counter value and the corresponding instruction are passed to the Instruction Decode stage through the IF-ID pipeline register.
- This stage also contains an ALU to increment PC by 2. It is connected back to R0 through a mux in the OR stage.

2.2 Instruction Decode

- The instruction decoder receives the instruction from the IF-ID pipeline registers, and performs two functions: it generates control signals for the subsequent stages, and it passes on the bottom 12 bits of the instruction to Operand Read for obtaining immediate if required.
- The control signals generated are as follows:
  - Type of instruction (R, I or J) : 2 bits
  - C/Z Dependency (for instructions conditional on C or Z) : 2 bits
  - ALU code, which determines the operation the ALU in the EX stage should perform : 4 bits

  - ALU select, which determines the operands from OR stage to be passed on to the EX stage for each of the ALU inputs : 4 bits
  - Hazard bits, which are used to determine the possibility of and type of hazard : 3 bits
  - RF write, which is high if the instruction writes to the register file : 1 bit
  - Memory write, which is high if the instruction writes to memory : 1 bit
  - Register write, which determines which register is to be written to : 3 bits
  - Register read, where each bit is high if the corresponding register is read from (for dependencies) : 8 bits

- The instruction decoder also contains a controller for Load Multiple and Store Multiple instructions with 9 states. It is by default in the reset state, and every time an LM or SM instruction is encountered, it leaves the reset state, and gives as an output whether to write to memory/RF, and which state it is currently in.

2.3 Operand Read

- Operand read contains the Register File, and is also responsible for passing on the required operands to the EX stage.
- The register file allows two registers to be read simultaneously, and can write to a single register, in addition to PC (R0.) In case of a clash in writing to R0, writing to R0 as a register is given priority over writing to PC.
- The stage also contains 2 muxes to decide each of the ALU inputs for different instructions, and a sign extender that sign extends the immediate received from the ID stage.
- In case of a hazard, the required data needs to be used in place of the data from RF or from the previous stage, and so there is another set of muxes that switch inputs whenever there is a hazard.
- The Operand Read stage therefore passes on the 2 ALU inputs, and RegA and RegB as obtained from the register addresses in the instruction. This is needed for checking for branches, e.g. in BEQ

2.4 Execute

- The execute stage primarily consists of an ALU that performs the required operation on the inputs depending on the ALU code control signals, and updates the required flags (C and/or Z.)

- It also checks for branches and gives a branch control output signal that is used by the branch predictor.
- The EX stage also contains single bit registers storing the carry and zero flags. It also updates the RF write control signal for conditional arithmetic-logical instructions like ACW.

2.5 Memory Access

- This stage is used in load and store instructions, where we write to and read from memory. It consists of the byte addressable data memory, which can either be read from or written to in one cycle.
- It uses the ALU output for the address, and gives the data stored at that address as an output to the next stage.

2.6 Write Back

- This stage first determines what data is to be written back, and then writes back to the register file.
- It consists of a mux that decides whether to write from memory, from ALU or from PC, and it then writes to the address in RF given along with the control signals.

2.7 Jumping, Branching and Branch Prediction

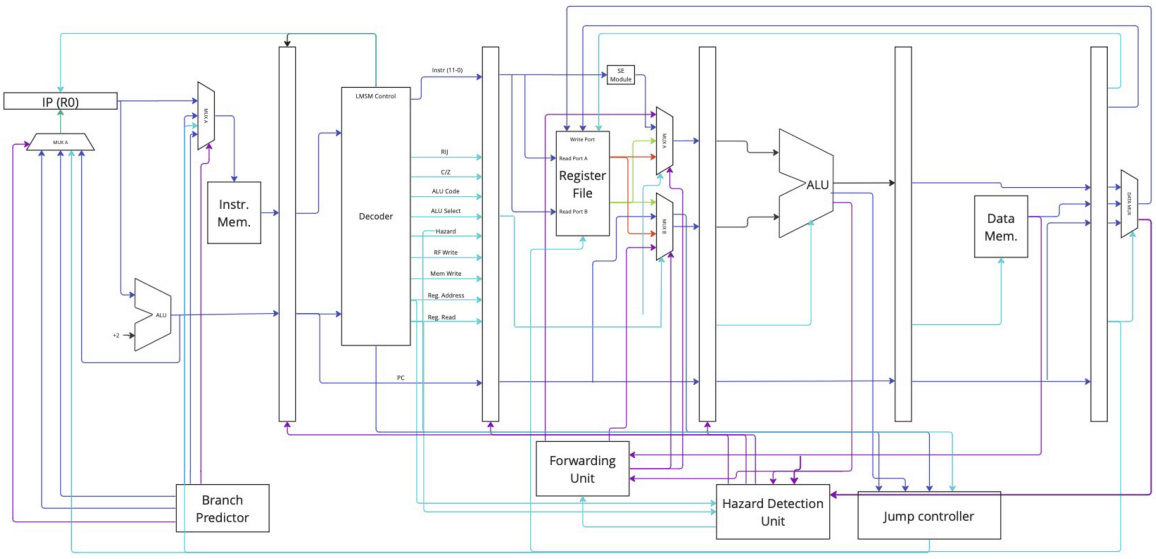
- For jumping, we have three 3 different instructions that give the address of jump in three different stages. JAL in decode stage itself so no cycles are wasted. JLR in OR stage, so 1 cycle is wasted and JRI in execute stage so 2 cycles are wasted.
- For branching all of them give their output of whether to branch or no in the execute stage itself. We then disable the write access to pipeline registers accordingly to go to correct address if there was a jump or a branch and to not execute the wrongly loaded instructions in between.
- For branch prediction, we use a 2 bit branch predictor without a look up table.

2.8 Dependencies and Hazards

- Dependencies happen when an instruction that writes into a register is followed closely by an instruction that reads from the same register. So, we split the instruction into hazcodes such they seperatee the instructions that only write, only read, read and write.

- Based on this, we identified whether we have to forward the data or no and this required 3 muxes, 1 from each of memory access, execute and WB stage to the OR stage. We also had to consider that WB, memory acccess and execute will have priority in that order as that is the order in which instructions were loaded.

3 Final Circuit Diagram



4 Work Distribution

- **Aditya:**
  - Memory and Write Back
  - Hazard Detection and Redressal
  - Debugging
- **Ashwajit:**
  - Operand Read
  - Top Level Implementation
- **Raunak:**
  - Execute stage
  - Hazard Detection and Redressal
  - Debugging
- **Suchet:**
  - Instruction Fetch, Instruction Decode stages
  - Control signals
  - Debugging