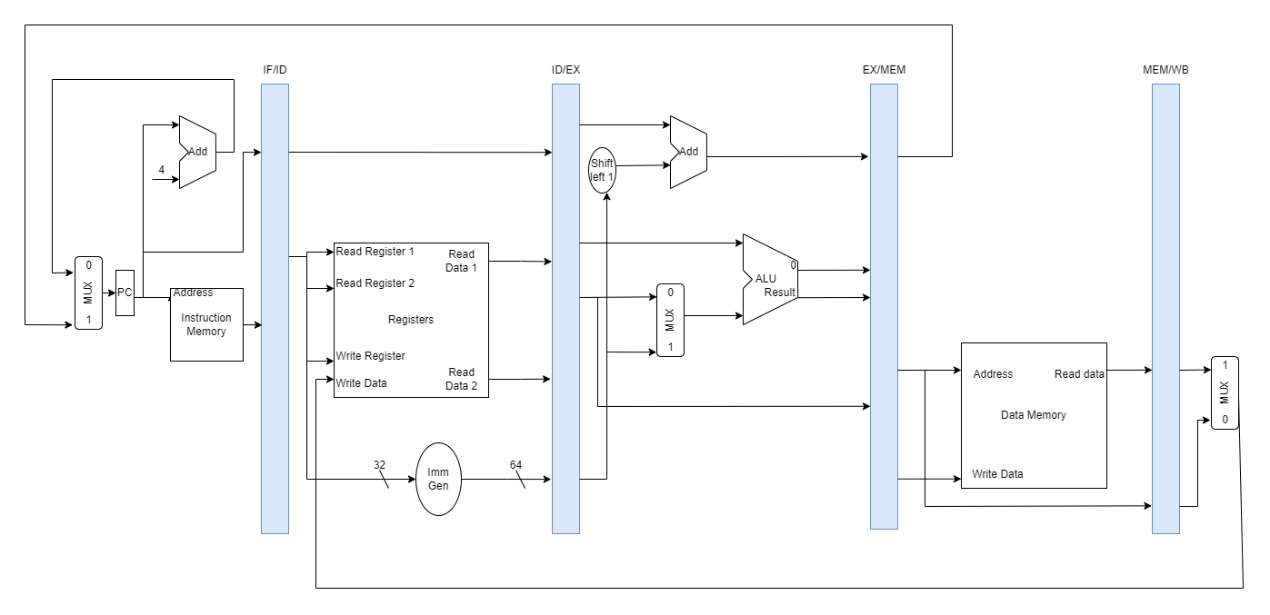
Computing Systems Architecture-Project A

**Name**: Rongze Li **Net ID**: rl4670

**Name**: Xinran Tang **Net ID**: xt2191

In this project, five different stages are implemented as: Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory Read/Write (MEM), and Write Back (WB). The processor supports five basic instructions: R-type, I-type, S-type, B-type, J-type, and a HALT instruction. The components realized by this project are Instruction Memory, ALU, ALU Control, Control Unit, Decoder and Data Memory. Both instruction and data memory are in“Big-Endian” format.

1. **schematic for a single stage processor**



As shown in the image, the processor has five stages: IF, ID, EX, MEM, and WB. When an instruction was fetched from the instruction memory, it will be parsed and get corresponding funct7, funct3, opcode, rs1, rs2, rd, and type. If the instruction has immediate value, them the Imm will also be parsed out. If the instruction was detected as a HALT instruction, all stages will be set to “nop”. Then, the parsed instruction will be sent to the ALU to calculate the results. The Control Unit will generate control signals for other essential components according to the opcode. Contains a finite state machine that enables multi-cycle execution. Sets and Outputs signals to components by stages: Instruction Fetch (IF), Instruction Decode (ID) and Execute (EX), Memory Access (MEM), Write Back (WB), and Halt (HLT). In the MEM stage, there are two main functions, memory read, and memory write. The final stage is WB, which write the result back to the registers.

1. **schematic for a five-stage pipelined processor**

图示

描述已自动生成

The five stages are similar to the five stages in the single stage processor. The difference between the single stage processor and the five stages pipelined processor is that the pipeline allows multiple instructions to be processed at the same time, without a pipeline, each instruction would have to wait for the previous one to finish before it could even be accessed. The pipeline enables the processing of each task in parallel rather than waiting for a task to finish and then moving onto another.

There are hazards in pipelined processor. In order to resolve different types of hazards (data and control), a hazard unit is implemented to keep track of different hazards and when any hazard occurs, appropriate action is taken by this unit. Two types of hazards are detected and solved:

1. **RAW Hazards**: RAW hazards are dealt with using either only forwarding (if possible) or, if not, using stalling + forwarding. Use EX-ID forwarding and MEM-ID forwarding appropriately.

* **Forwarding** – Forward the result from the Memory Stage and the Write Back stage to the Execute stage.
* **Stalling** (for load word) – Data read from memory is available only at the end of clock cycle. So, we stall the next instructions and forward the memory data from Write Back stage.

1. **Control Flow Hazards**: The branch conditions are resolved in the ID/RF stage of the pipeline.

* **Flushing** – Whenever the decision is made to take a branch in Execute stage, flush the next two instructions (Decode and Fetch Stages).

1. **average CPI, Total execution cycles, and Instructions per cycle**

* Test case 0

文本

描述已自动生成

* Test case 1

文本

描述已自动生成

* Test case 2

文本

描述已自动生成

* Test case 3
* Test case 4

1. **Compare the results from both the single stage and the five-stage pipelined processor**

According to the results obtained by the executed code from TC0 to TC 4, the performance of single stage and five stages pipelined processor is not fixed. When there are not so many instructions, the performance of single stage processor is better that five stages. By contrast, when there are many executed instructions, the five stages pipelined perform better than single stage. The reason for the results is that when there are few instructions to be processed, the role of pipeline is difficult to reflect, which will lead to processing efficiency. When there are many instructions, the pipeline can well plan the continuous processing of instructions, which will save time and efficiency.

1. **features added to improve performance**
2. When it is detected that the ID-stage needs jump instruction and the EX-stage contains the source operand required by the branch jump instruction, the ID-stage is blocked, thus breaking the critical path of EX operation -> branching judgment. It is important to note that the implementation is designed as a whole, so that this situation does not occur again, rather than organizing the execution of this situation through detection. The former avoids the critical path, while the latter reconnects it because of the detection circuit.
3. Move the branch instruction to the EX-stage. If using this method, then without a branch predictor, every time we jump, there is bound to be one beat of bubbles, which is slightly less effective than method a, but more regular.