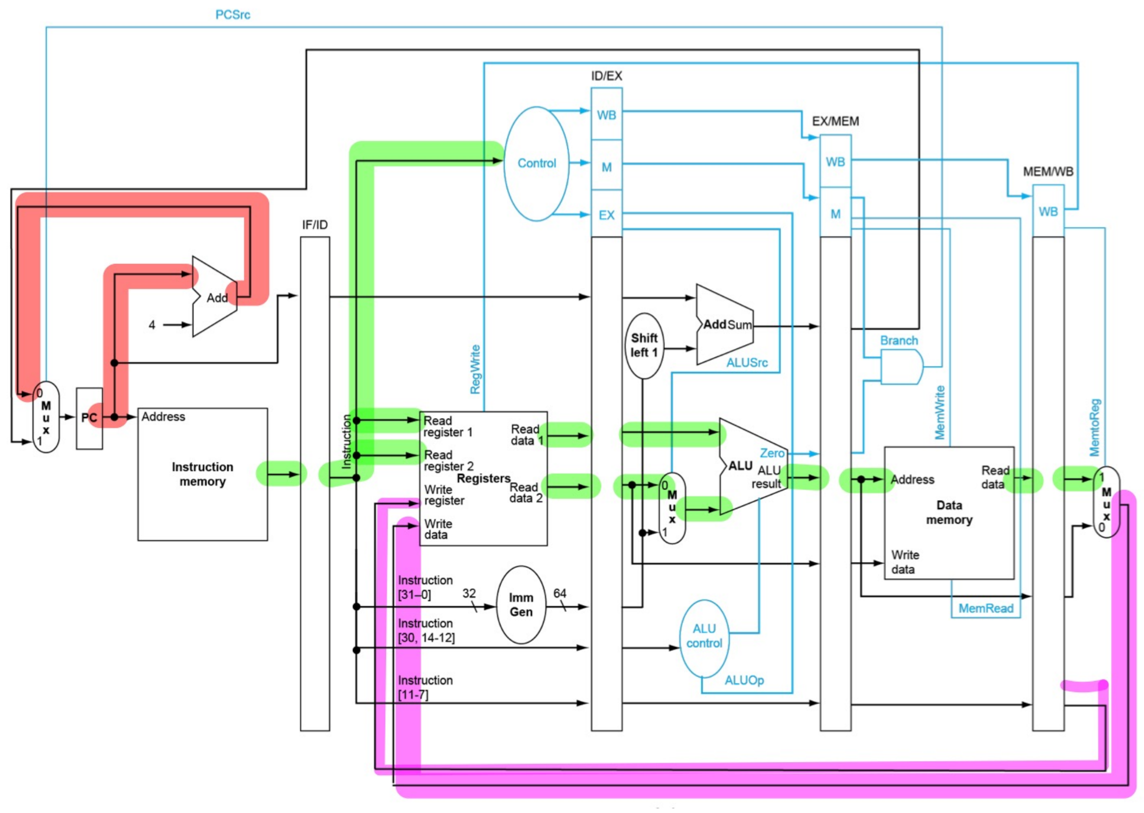
Project A - Phase II

ECE 6913 Computer System Architecture

Jingjing He (jh8244@nyu.edu), Ziqi Gao (zg2346@nyu.edu)

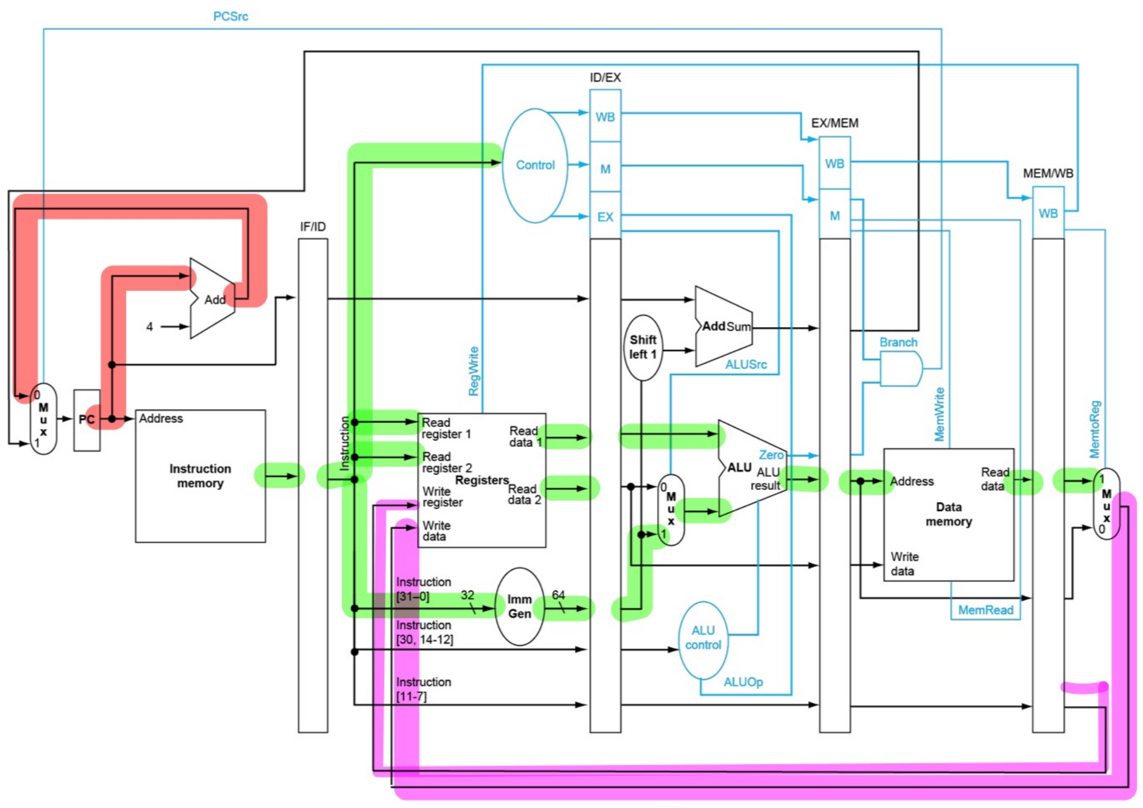
Schematic and Datapath for Single Cycle Processor

R-Type



- Instruction is fetched from the instruction memory, and the PC is incremented.
- Two registers' values, are decoded and read from the register file (instruction[19-15], instruction[24-20]). ALUSrc, RegWrite are set in the control unit. ALUOp1 =1, ALUOp0 = 0
- ALU Control are set based on Instruction [14-12] (func3) with ALUOp1 =1 ALUOp0 = 0.
- The ALU operates on the data read from the register file, using ALU Control's output to generate the ALU function.
- The result from the ALU is written into the destination register (instruction[11-7]) in the register file.

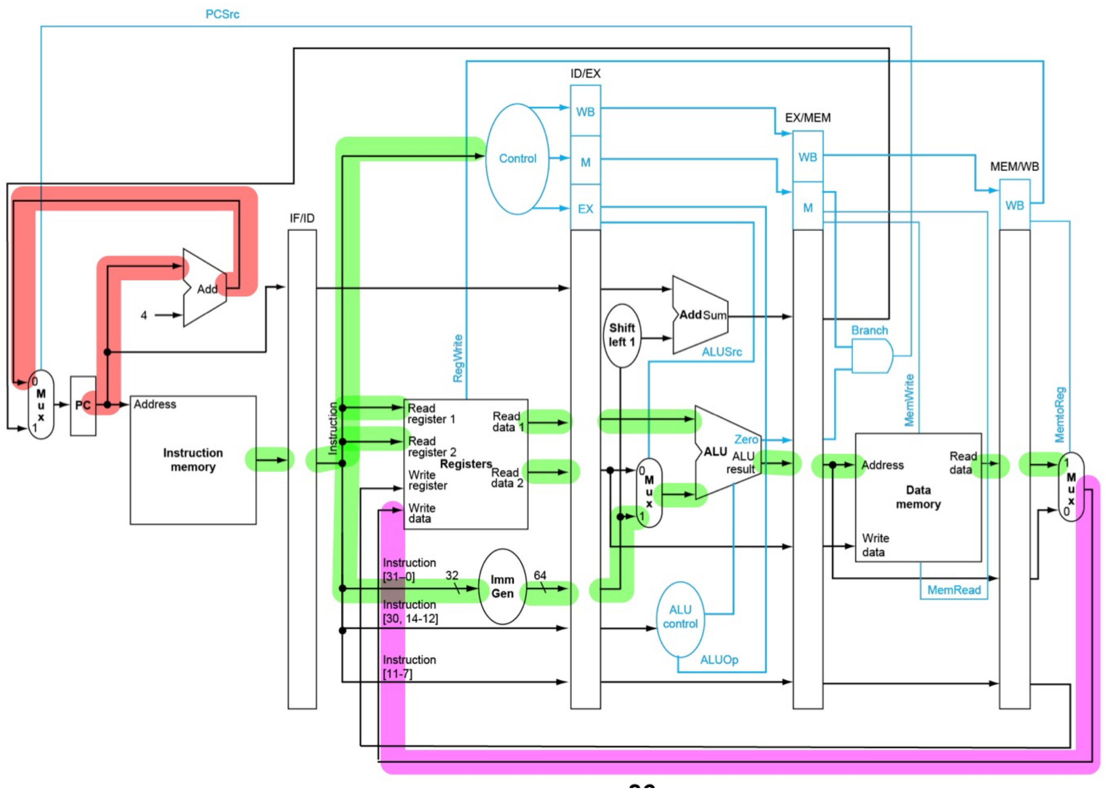
I-Type



- 86

- Instruction is fetched from the instruction memory, and the PC is incremented.
- Instruction is decoded and the register (instruction[19-15]) is read from the register file. RegWrite are set in the control unit.
- ALU Control are set based on Instruction [14-12] (func3) with ALUOp1 =1 ALUOp0 = 0. And the immediate value is sign-extended.
- The ALU operates on the data read from the register file and the sign-extended immediate value, using ALU Control's output to generate the ALU function.
- The result from the ALU is written into the destination register (instruction[11-7]) in the register file.

LW/SW



- Instruction is fetched from the instruction memory, and the PC is incremented.
- Instruction is decoded and the register (instruction[19-15]) is read from the register file. RegWrite are set in the control unit. rd_mem is set for LW, and wrt_mem is set for SW. For LW, only reg 1 need to be read, but for SW, both reg1 and reg2 should be used. Wrt_reg_addr, is set for LW. is_I_type is set, so the multiplexer can choose the sign extended immediate value.
- ALU Control are set based on Instruction [14-12] (func3) with ALUOp1 =1 ALUOp0 = 0. And the immediate value is sign-extended.
- The ALU operates on the data read from the register file and the sign-extended immediate value, using ALU Control's output to generate the ALU function. The final result is written into EX/MEM Register.
- For, LW, data is read from the memory and write back to the register. For SW, data is written into memory.

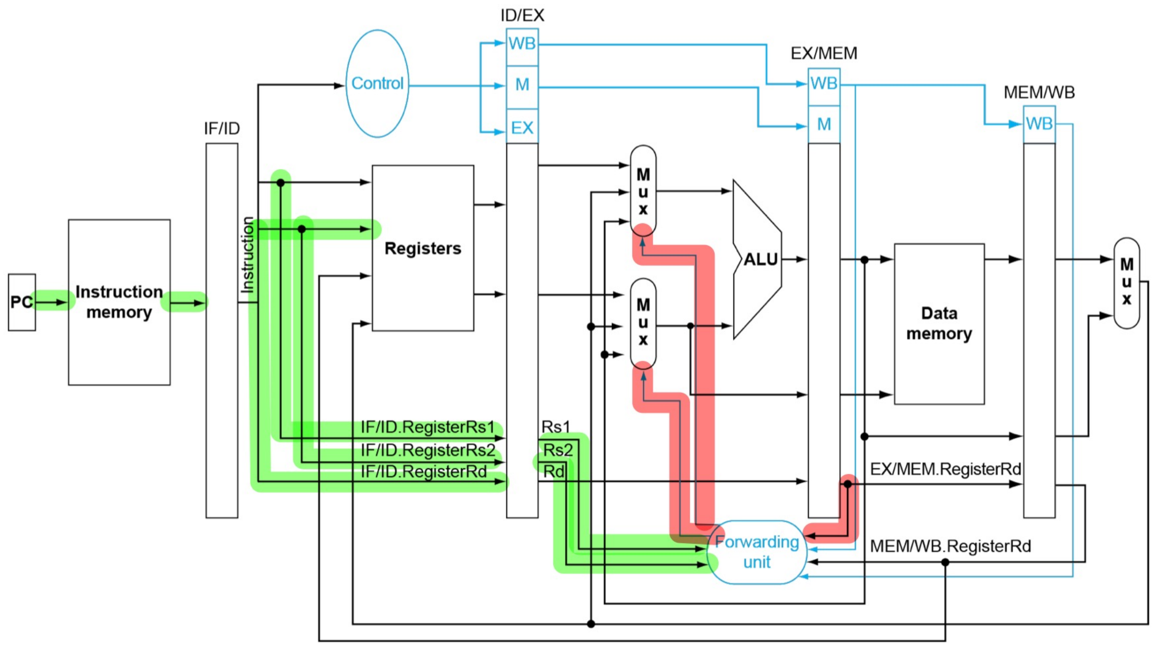
Branch/Jump:

Introduced in Hazard part.

Data Hazard

Raw Hazard

EX Hazard



We need forward the data under the following condition:

- The destination of the first instruction is used immediately in the next instruction. e.g.

```
sub x2, x1, x3
and x12, x2, x5
```

- The previous instruction need to write the register.
- The Write back register of the first instruction is not x0.
- The register to be written is the same as one of the rs1/rs2 in the later instruction.

MEM Hazard

We need forward the data under the following condition:

- The destination of the first instruction is used immediately in the next next instruction. e.g.

```
sub x2, x1, x3
add x13, x6, x7
and x12, x2, x5
```

  It is the same if the first instruction is `ld`:

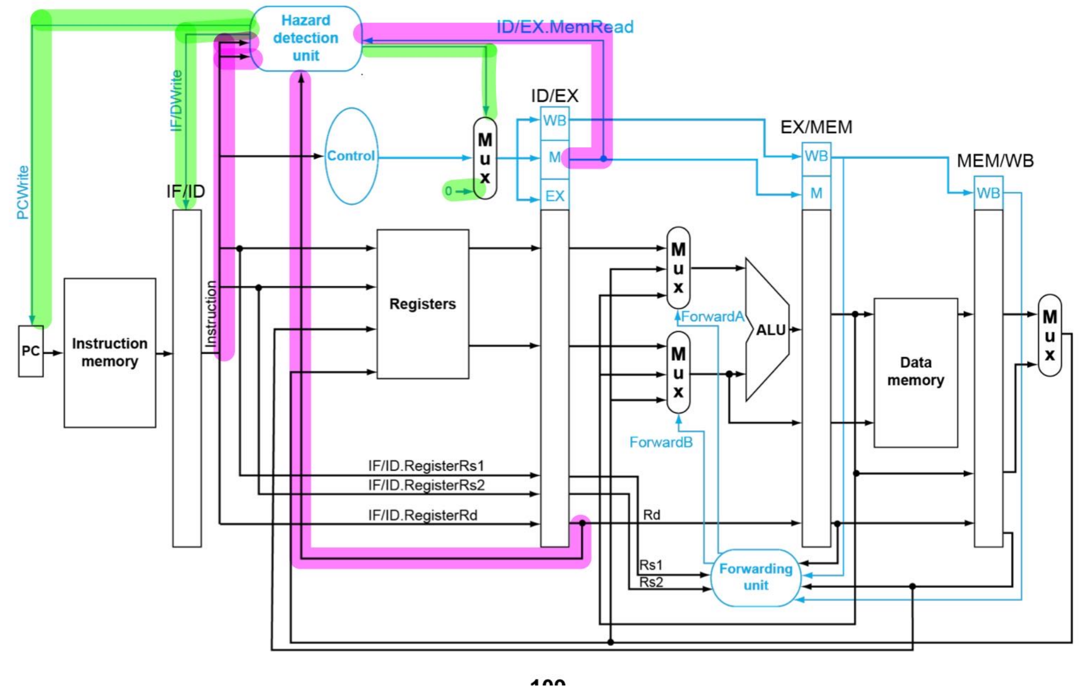
```
ld x2, 0(x7)
add x13, x6, x7
and x12, x2, x5
```

- The previous instruction need to write the register.
- The Write back register of the first instruction is not x0.
- The register to be written is the same as one of the rs1/rs2 in the latest instruction.
- There is no double data hazard like this:

```
add x1, x1, x2
add x1, x1, x3
add x1, x1, x4
```

In this case, we will not forward the data in the MEM/WB stage but forward that in the EX/MEM stage.

Load-use Data Hazard:



- This situations happens when we load the memory into a register rd, and the rd is immediately used in the next instruction. e.g.

```
ld x2, 20(x1)
and x4, x2, x5
```

- **When `ld` in its ID/EX stage**, we can check whether Load-Use Data Hazard happens. The `rd` used in `ld` should be equal to one of the `rs1/rs2` used in the next instruction.
- In this case we need to firstly stall the next instructions by unsetting the `IF/IDWrite` and `PCWrite` to stall the following instructions for one cycle.
- Also, ID/EX is set to all 0 through the multiplexer to stall.
- At the same cycle the ALU does the calculation as usual.

Then in the next cycle **(`ld` in the EX/MEM stage)**:

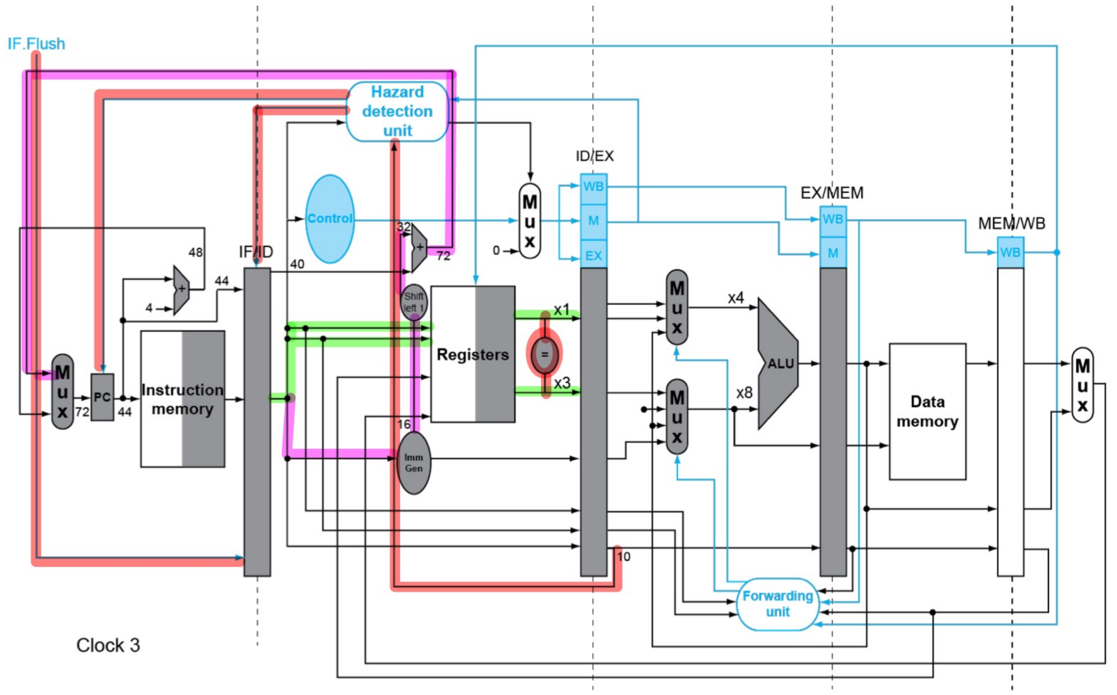
- Read the data from memory, and write it to `MEM/WB` register.
- Because `ID/EX` is all set to 0, nothing to be done in this cycle.
- `IF` and `ID/RR` stage running again, with the same parameters as the last cycle.

Then the problem turns into the `MEM Hazard`. And no more extra actions need to be done.

Branch Hazard (Can be applied to JAL)

We always assume that the branch will not be taken. And therefore, when a branch is taken, some actions need to be done.

Whether to take the branch is decided in `IF/ID stage` with two ALUs and to check the equality of `rs1.val` and `rs2.val` and calculate the new address (For J-type, we just assume the branch will always be taken without checking rs1.val and rs2.val):



Now we are in the following situation:

- The branch is decided to be taken in `IF/ID` stage.
- The instruction next to the branch instruction will write its info into `IF/ID` register. We need to set the signal `IF.Flush` to flush it.
- In the same cycle, the `ID/EX` is set to 0 to stall.

In the next cycle:

- The `ID/EX` do nothing for its register is set to 0.
- The `IF/ID` do nothing for its register is also set to 0.
- The new instruction is set to the new address got from the extra ALU.

Comparison between Five Stage Processor and Single Stage Processor:

Single-Stage Processor:
In a single-stage processor, instructions are executed in a single stage or step, meaning that the entire process (fetch, decode, execute, memory access, and writeback) occurs sequentially. This processor's performance is limited due to the linear nature of its execution.

Advantages:

- Simple design: The single-stage processor is easy to design and implement.
- Low latency: Each instruction is executed in a single cycle, resulting in low latency for individual instructions.

Disadvantages:

- Low throughput: The processor can execute only one instruction at a time, leading to a low throughput rate.
- Poor resource utilization: Since the processor's components are used sequentially, they often remain idle during different stages of instruction execution.
- Low Clock Frequency: The clock must be long enough to complete the slowest instruction, which has less flexibility than 5-stage processor.

Five-Stage Pipelined Processor:
In a pipelined processor, instruction execution is divided into multiple stages, allowing for multiple instructions to be processed simultaneously. A typical five-stage pipeline includes the following stages: instruction fetch (IF), instruction decode (ID), execute (EX), memory access (MEM), and writeback (WB).

Advantages:

- High throughput: As multiple instructions are processed concurrently, the processor's throughput is significantly improved. The clock frequency is high and it uses one clock cycle to finish only one stage instead of the whole instruction.
- Better resource utilization: By dividing instruction execution into stages, the processor's components are utilized more efficiently, reducing idle time.
- High potential for optimization: Through good design of compiler or other software, the scheduling and pre-process of code may improve the performance of 5-stage pipeline even more. (This can also be considered as disadvantage, since the performance depends on software optimization).

Disadvantages:

- Increased complexity: Implementing a pipelined processor requires more complex design and control logic.
- Hazards handling: Because of the pipeline design, hazards (data/control) should be approached appropriately. And stalls need to be inserted in some situations, which will affect the performance.
- Problem brought by high clock frequency: It will increase the power consumption,

Optimizations to improve performance:

- Increase the pipeline depth. Instead of 5-stage, 6-stage, 7-stage... may be used to have higher clock frequency.
- Try to use dynamic branch prediction instead using "Always Not Taken" principle.
- Compiler optimizations like better scheduling algorithm, pre-processing algorithm to avoid the frequency of hazards. Also, multithreading may also help to reduce the frequency of hazards and exploit parallelism at the same time.