TDT 4255 Project Overview
RISCV-FiveStage Processor

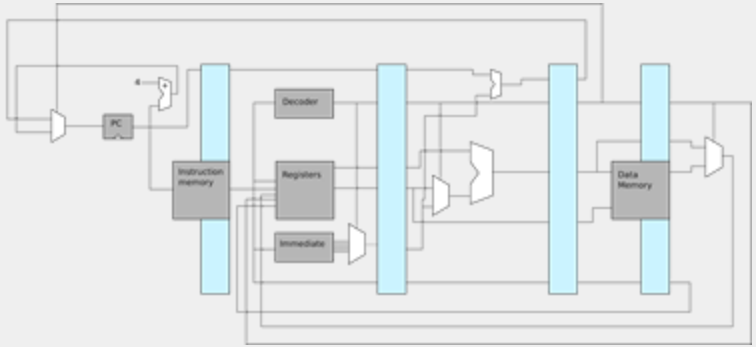
TDT 4255
Joseph Rogers, David Metz, Magnus Jahre

Announcements:

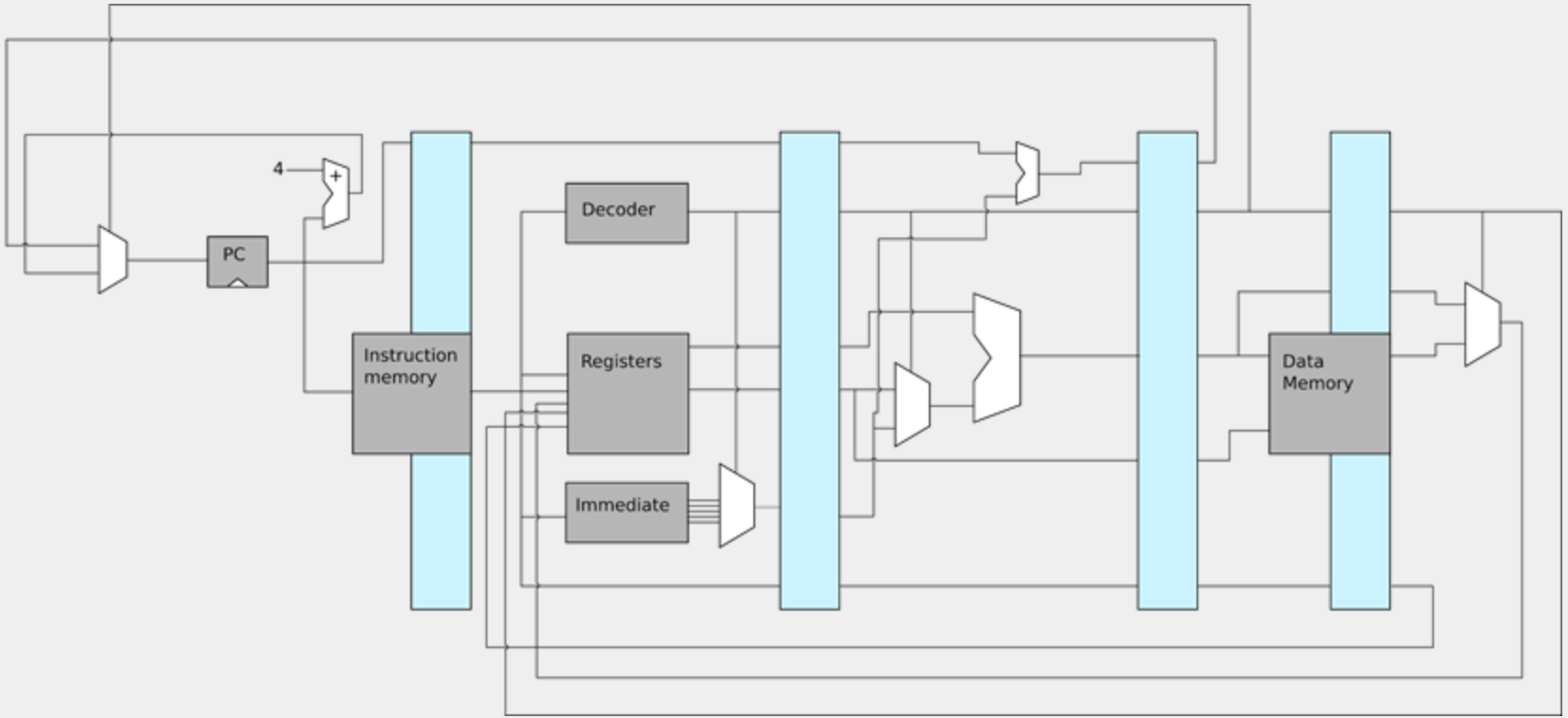
- Ex0 is due on Sep. 20th
- Show your solution on either Monday or Wednesday

Project Overview:

- Use Chisel to design a simple processor based on a real ISA! (RISCV32I).
- Use your design to run real RISC-V programs!
- Improve the design to increase performance.



Processor Overview (Classic RISC Pipeline):

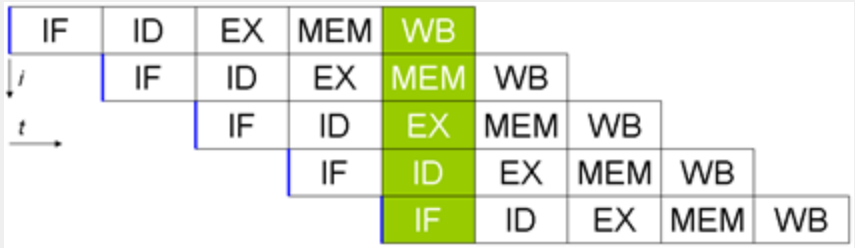


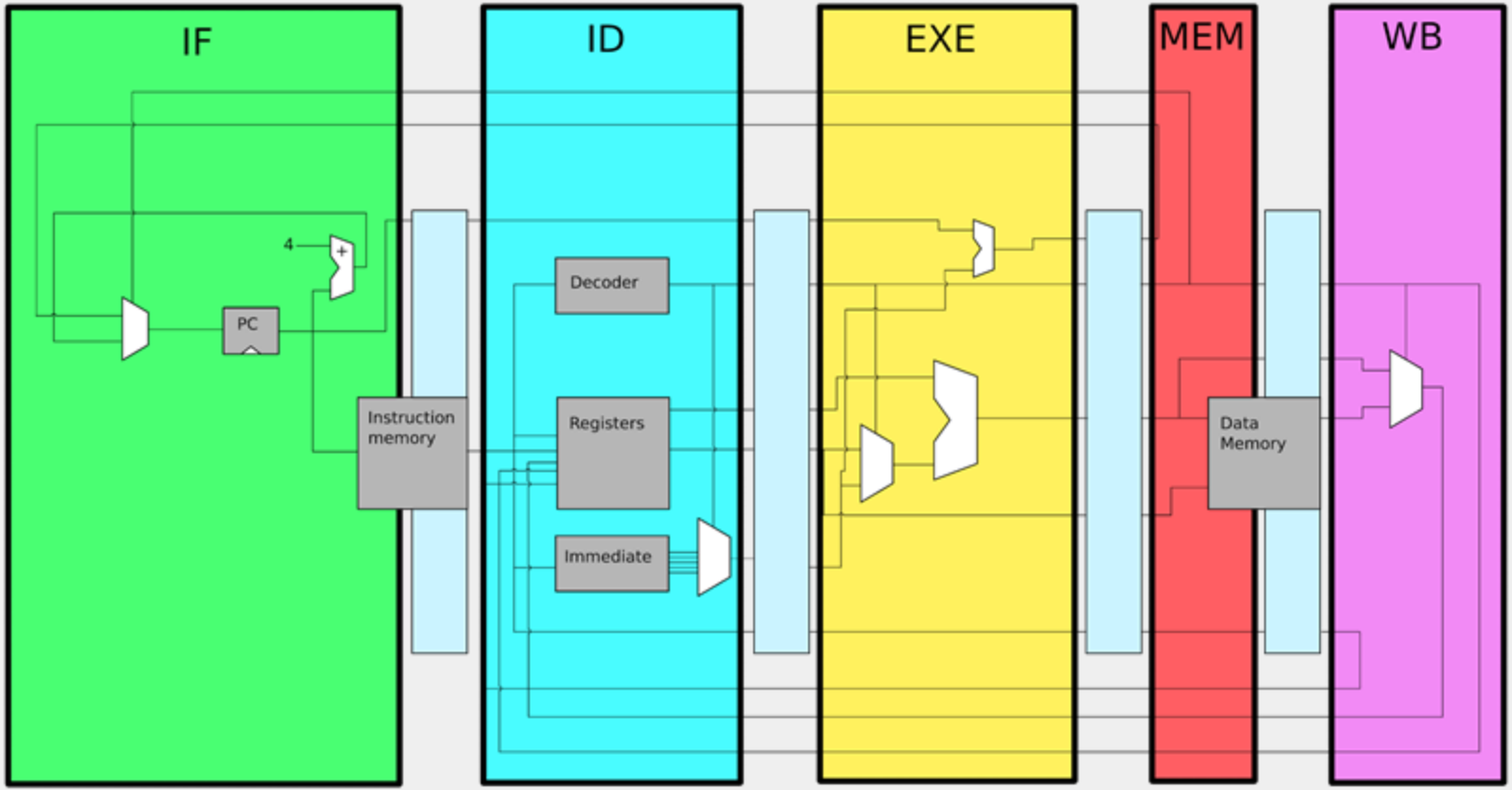
Processor Overview:

- This is a pipelined processor!
- Parts of up to five different in instructions can be being executed simultaneously.

Pipeline Stages:

- IF -- Instruction Fetch
- ID -- Instruction Decode
- EX -- Execute
- MEM -- Memory Access
- WB -- Writeback




IF
ID
EXE
MEM
WB
4
+
PC
Instruction memory
Decoder
Registers
Immediate
Data Memory

Project Milestones:

Milestone 1, Due Oct. 11th: Partial CPU functionality, all “basic” tests passed.

- ALU + Load/Store

Milestone 2, Due Oct. 18th: Basic CPU complete, all tests passed with NOPs inserted.

- Branches

Milestone 3, Due Nov. 1st: Fully pipelined complete CPU, all tests passed.

- Forwarding / Hazards

Final Submission, Due Nov. 15th: CPU fully functional, along with at least one performance upgrade.

Milestone 1, Basic Datapath:

- Fill out the code for each of the basic components for each stage of the pipeline.
- Hook the components together so that “basic” programs function.
- 4 NOP instructions are inserted between each fetch. This will stop all hazards.

Milestone 2, Completing the Datapath:

- Finish implementing all of the RISCV instructions so that the basic datapath works on all tests with NOPs included.
- Hook the components together so that basic programs function.

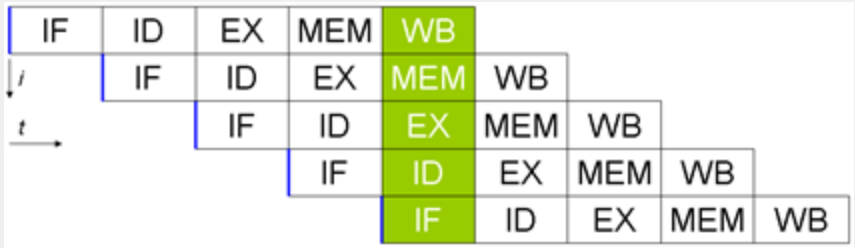
Milestone 3, Pipelining the CPU:

- Turn of the NOPs, start working on true pipelining.
- **RAW Hazards** - Implement support for handling RAW data hazards so that the processor can be correctly pipelined.
- **Control Hazards** - Make it so that branches do not always have to stall.

Pipeline Hazards:

- Initially, four NOPs were inserted in between each “real” instruction.
- This is done to avoid hazards/data dependencies in the current design.
- Improvements to the processor made during exercise 2 will fix this.

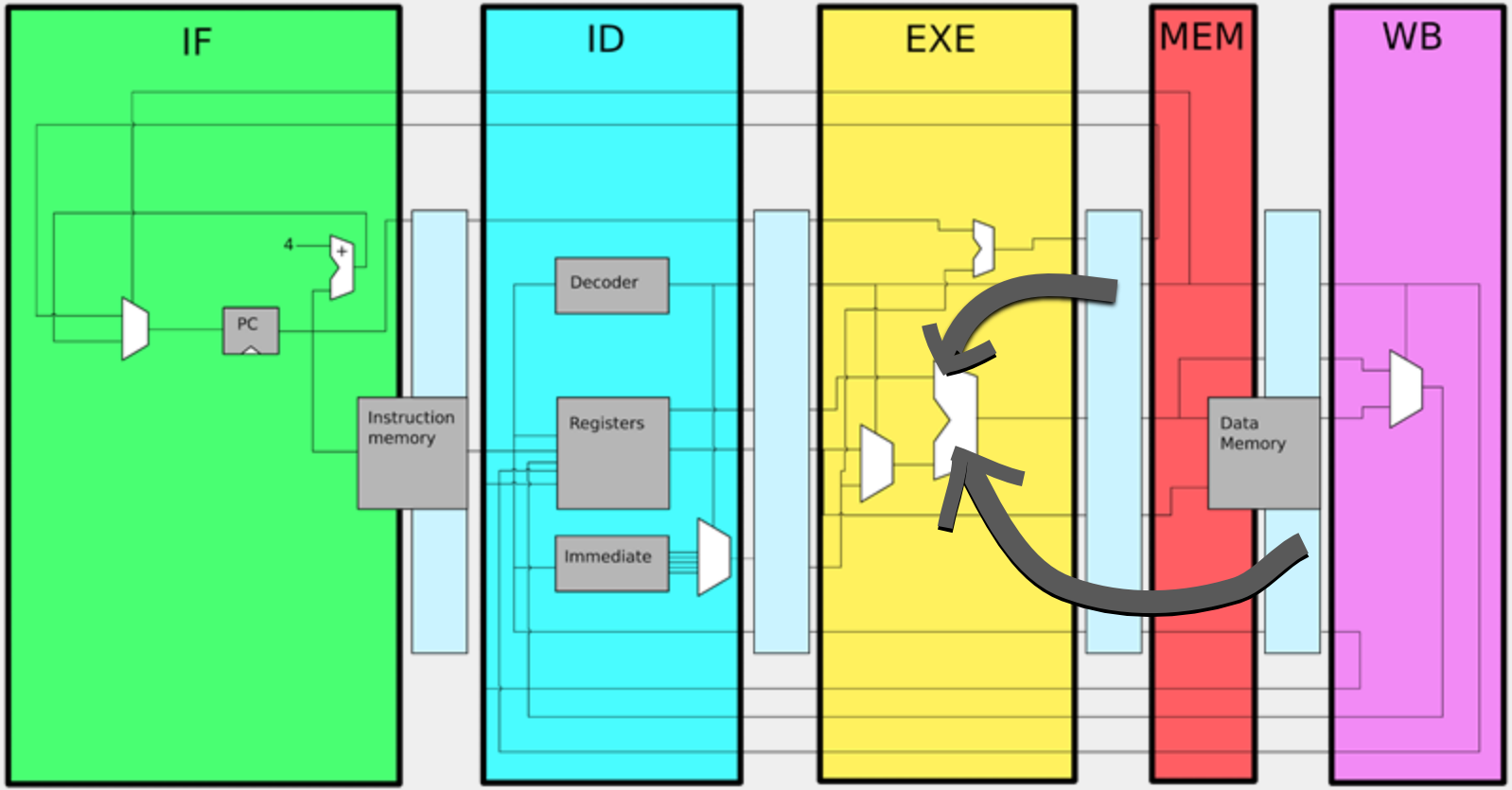
```
for i in range(1, 10):
        array[i] = array[i-1]*2

/* This could create a RAW
Hazard */
```



Dealing with RAW Data Hazards:

```
main:
   add x1, x1, x2
   add x1, x1, x1
   add x1, x1, x2
```

- Add hardware (a forwarding unit) in the EX stage that can select the ALU input from another source.
- Sources are register bank, MEM, or WB stages.
- Selection priority based on newest result, **MEM > WB > REG.**


IF
ID
EXE
MEM
WB
4
+
PC
Instruction memory
Decoder
Registers
Immediate
Data Memory

Delay After Load

```
main:
  lw x1, 0(x2)
  add x1, x1, x1
  add x1, x1, x1
```

- Op2 would load from the register, but x1 is not ready yet.
- Loading takes a cycle, so the forwarder must stall the pipeline.
- Issue a single to the barriers, telling them not to update.

Control Hazards:

```
main:
  beq zero, zero, target
  add x1, x1, x1
  add x1, x1, x1
  j main
target:
  sub x1, x2, x2
  sub x2, x2, x2
```

- The add instructions could be fetched before the beq jump happens.
- When a branch happens, incorrect fetches must be flushed.
- Waiting until each branch is resolved is undesirable because it guarantees a stall.
- More details in the assignment description…

Final Submission, Performance Improvements:

Branch Prediction: Add some kind of branch prediction strategy! Either 1-bit, 2-bit, or something else.

Fast Branch Handling: Certain branches like BEQ and BNE can be done faster than BGE or BLE.

Value Prediction: If lots of values are being calculated in a predictable pattern then they could be speculatively generated to save cycles.

Data Cache (Only if you want): Only try this if you have already done one of the others. More details on Github…

Instructions:

- Clone https://github.com/EECS-NTNU/RISCV-FiveStage
- Follow the instructions to make the processor!
- Instructions are contained in the README.org, introduction.org, instructions.org, and exercise.org files
- When finished, present your work during one of the lab sessions in R90.
- For the final CPU submission there will be a Blackboard submission link so that your code can be reviewed.
- At the end of the final project, please turn in your entire project folder as a compressed archive.
- **Get started now!**

Recommendations:

- Use peekpoke testers.
- Use waveforms https://github.com/EECS-NTNU/tdt4255-chisel-intro/blob/master/waveforms.org
- Write your own tests for incremental functionality.

Resources:

- https://github.com/EECS-NTNU/RISCV-FiveStage -- Project Github page.
- https://riscv.org//wp-content/uploads/2017/05/riscv-spec-v2.2.pdf -- Official specs for the RISCV32I ISA (and several others). This is pretty dense, but it can server as a good reference.
- https://en.wikipedia.org/wiki/Classic_RISC_pipeline -- Decent overview of a classic RISC pipeline, similar in concept to this assignment.
- **Get started now!!**