ECE 552 Final Project Report

Team 9: Alex Moy, Nick Cook, Marshall Lang

Overview

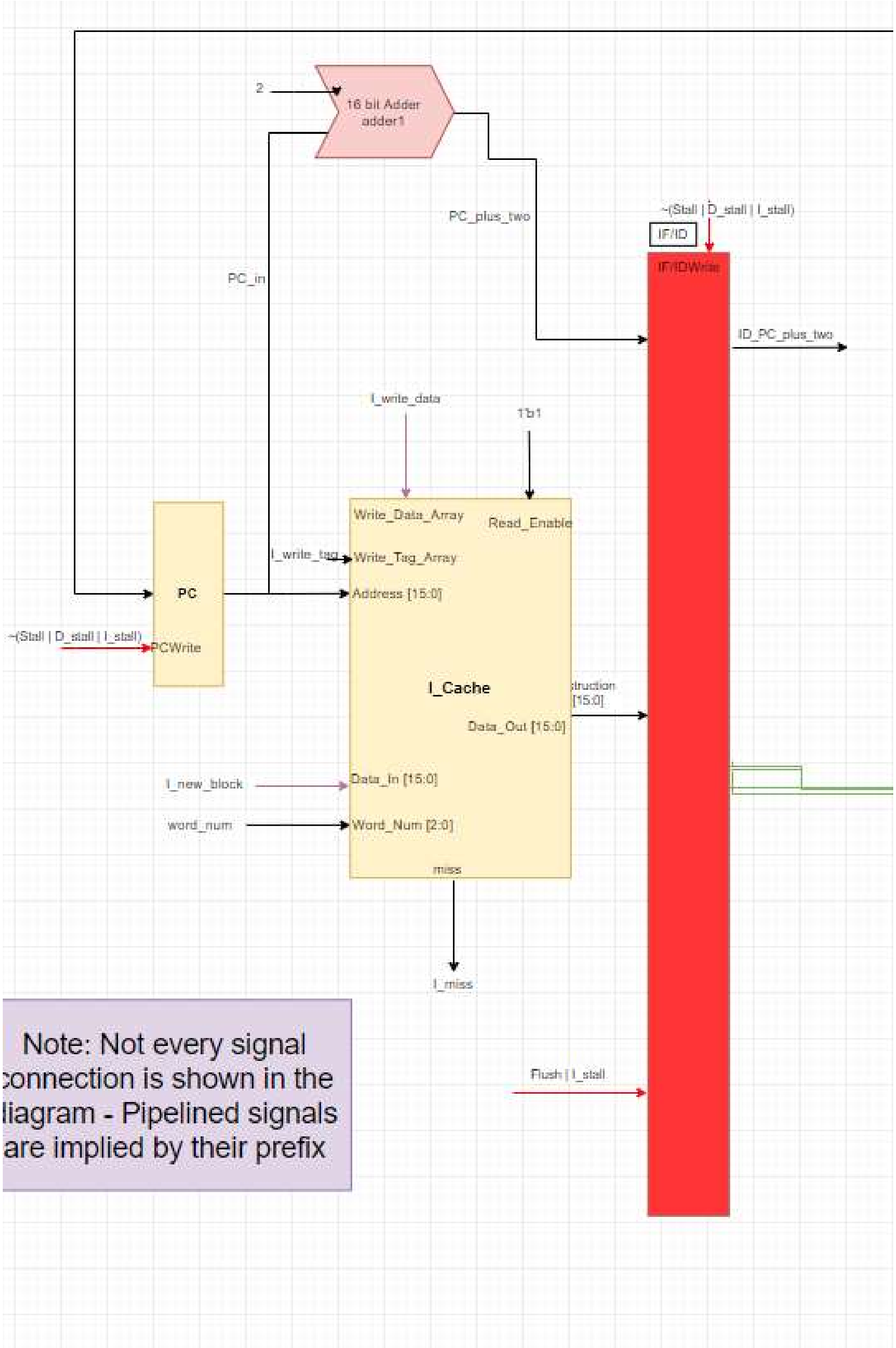
Our design is similar to the appearance of the one presented in the textbook. We tried to keep the CPU module as clean as possible and have more of the lower level logic inside of its respective modules wherever possible. The main CPU module contains instantiations of lower level modules, multiplexing for inputs of modules, and some other basic logic like sign extension. The lower level modules instantiated by the CPU module are:

- Four pipeline registers
- A program counter
- Instruction and data caches
- A register file
- An ALU
- A register to hold stable ALU flags
- An adder for incrementing the program counter
- An adder for relative branching
- A module for computing control signals based on the instruction
- A module for determining if the program counter branches based on ALU flags
- A hazard detection unit
- A data forwarding unit
- A main memory

- A cache fill FSM
- An interface for moving data between the caches, the main memory, and the FSM

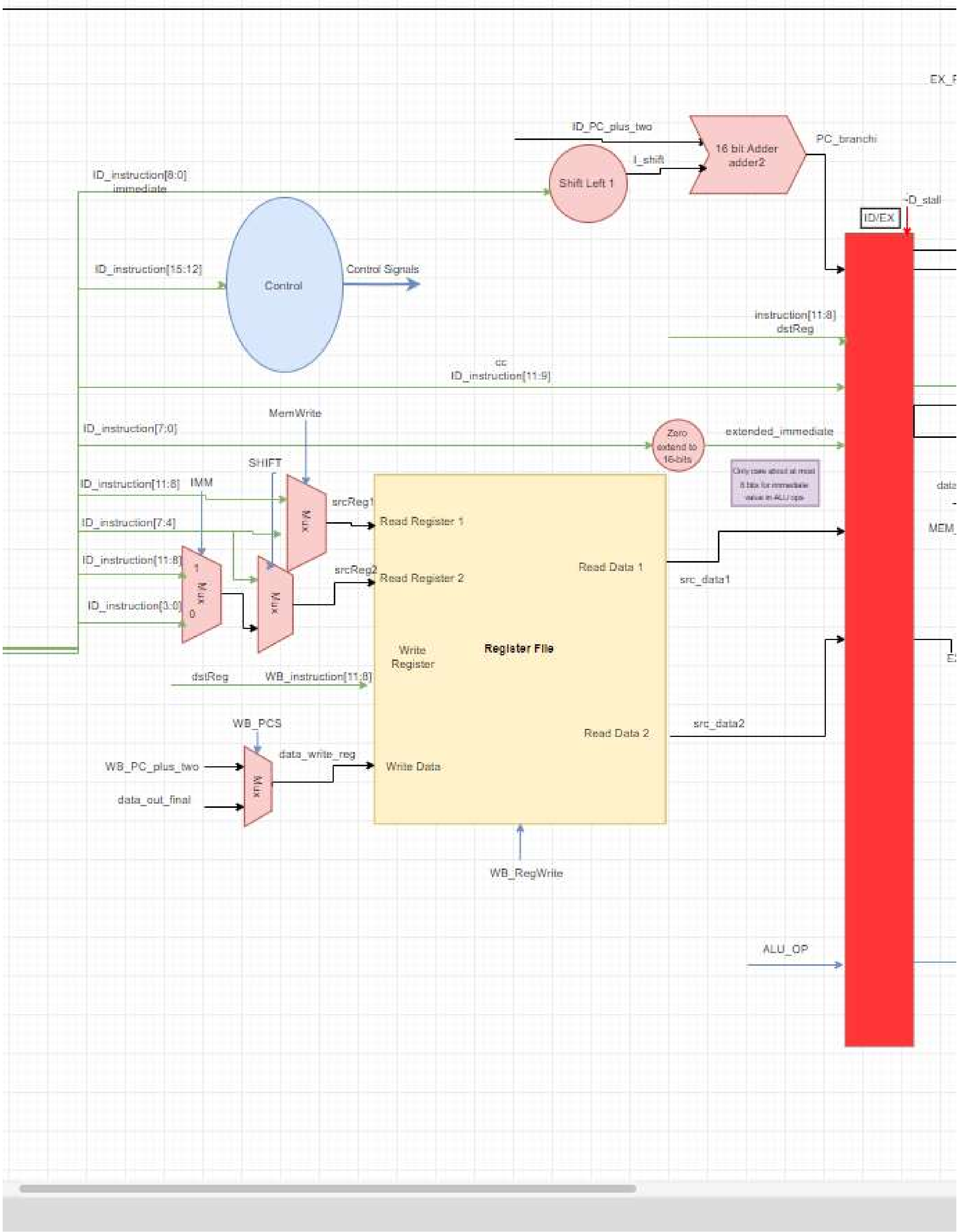
Datapath

IF/ID Stage

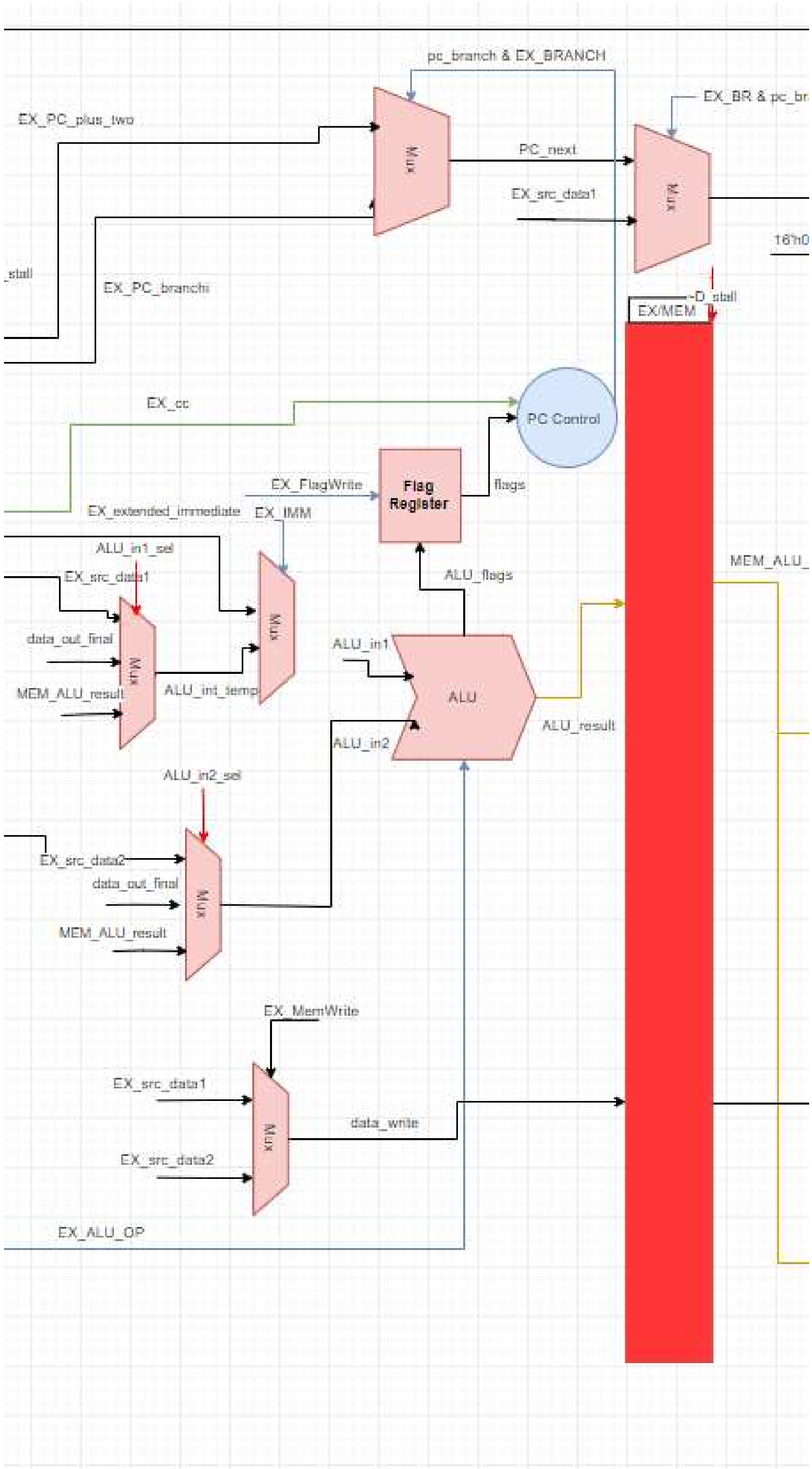


Note: Not every signal connection is shown in the iagram - Pipelined signals are implied by their prefix

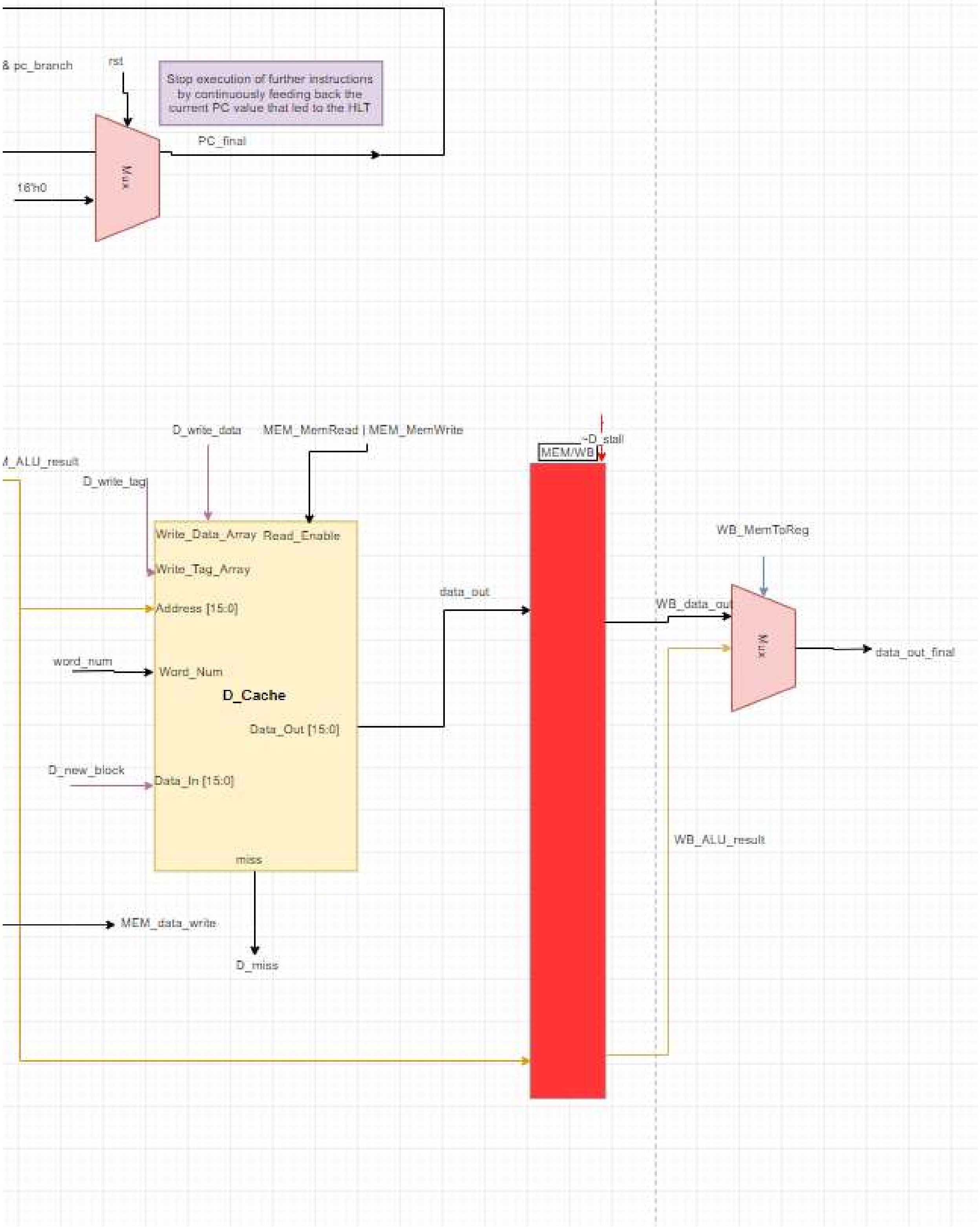
ID/EX Stage


EX_F
ID_PC_plus_two
16 bit Adder
adder2
PC_branchi
I_shift
Shift Left 1
ID_instruction[8:0]
immediate
ID_stall
ID/EX
ID_instruction[15:12]
Control
Control Signals
instruction[11:8]
dstReg
cc
ID_instruction[11:9]
MemWrite
ID_instruction[7:0]
Zero extend to 16-bits
extended_immediate
SHIFT
Only care about at most 8 bits for immediate value in ALU ops
ID_instruction[11:8]
IMM
srcReg1
Mux
data
ID_instruction[7:4]
Read Register 1
MEM_
ID_instruction[11:8]
1
Mux
0
Mux
srcReg2
Read Register 2
Read Data 1
src_data1
ID_instruction[3:0]
Write Register
Register File
E
dstReg
WB_instruction[11:8]
WB_PCS
Read Data 2
src_data2
data_write_reg
WB_PC_plus_two
Mux
Write Data
data_out_final
WB_RegWrite
ALU_OP

EX/MEM Stage

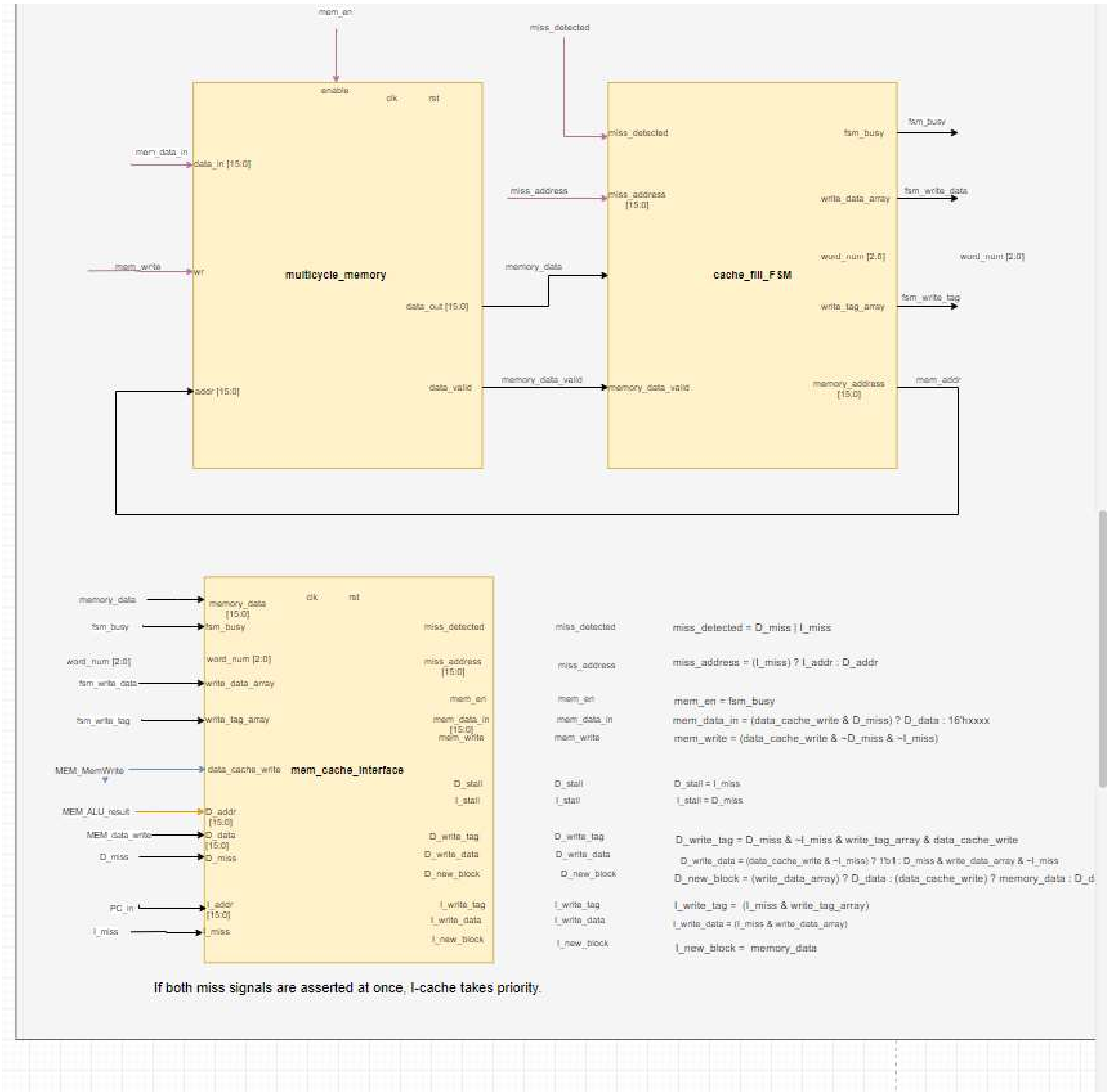

pc_branch & EX_BRANCH
EX_BR & pc_br
EX_PC_plus_two
Mux
PC_next
EX_src_data1
Mux
16'h0
_stall
EX_PC_branchi
D_stall
EX/MEM
EX_cc
PC Control
EX_FlagWrite
Flag Register
flags
EX_extended_immediate
EX_IMM
ALU_in1_sel
EX_src_data1
MEM_ALU_
data_out_final
Mux
Mux
ALU_in1_temp
MEM_ALU_result
ALU_flags
ALU_in1
ALU
ALU_result
ALU_in2
ALU_in2_sel
EX_src_data2
data_out_final
Mux
MEM_ALU_result
EX_MemWrite
EX_src_data1
Mux
data_write
EX_src_data2
EX_ALU_OP

MEM/WB Stage



Stop execution of further instructions by continuously feeding back the current PC value that led to the HLT

Cache Fill FSM / Cache-Memory Interface



```
miss_detected = D_miss | I_miss
miss_address = (I_miss) ? I_addr : D_addr
mem_en = fsm_busy
mem_data_in = (data_cache_write & D_miss) ? D_data : 16'hxxxx
mem_write = (data_cache_write & ~D_miss & ~I_miss)
D_stall = I_miss
I_stall = D_miss
D_write_tag = D_miss & ~I_miss & write_tag_array & data_cache_write
D_write_data = (data_cache_write & ~I_miss) ? 1'b1 : D_miss & write_data_array & ~I_miss
D_new_block = (write_data_array) ? D_data : (data_cache_write) ? memory_data : D_d
I_write_tag =  (I_miss & write_tag_array)
I_write_data = (I_miss & write_data_array)
I_new_block =  memory_data
```

If both miss signals are asserted at once, I-cache takes priority.

Responsibilities

Team Member	Responsibilities
Alex Moy	High level interconnectivity of design, initial datapath and modules, cache fsm, debugging, PC control, instruction decoding
Nick Cook	Cache modules and logic, CLA adder, RED module, ALU module, debugging
Marshall Lang	Stage 2 and 3 datapath, mem_cache_interface, pipeline registers, shifter module, debugging, hazard detection unit, forwarding unit

Special Features

One special feature with our design was that we abstracted our cache module so that we could instantiate both the instruction and data caches with the same module. Another thing we did was for every instruction, besides the branch and memory access instructions, we performed their computations inside of our ALU. We used a 4-bit opcode to determine the operation and had shifters, CLA adders, and all other necessary modules inside of our ALU module. We ran into problems during stage 3 when trying to hook up our cache modules. This was because the logic going to the read and write enables for the cache was cyclical, so it was causing Modelsim to stop execution of the testbench. This confused us for a while until we went through and reworked all of the logic required for those signals.

Completeness

For our stage 1 submission, we didn’t have the correct results for tests 2 and 3, so before we started stage 2 we went and debugged ours with the correct outputs for those tests that were provided. Once our stage 1 worked, we moved on to the pipelined implementation of our design. We got two points off of test 3, we believe due to some intermediate values being written to registers, but otherwise our stage 2 design was working well. For stage 3, we obtained the correct results for each test during the demo, but our PC value was off for test 4 and the instruction counts for each test were abnormally large. Because of this, we went and debugged again as well as connecting the correct signals to the testbench so that the plog and ptrace files would match up with what we would be graded against.

Testing

For testing our design, we would run the testbench and look at each module we created, figuring out what the correct outputs should be and determining whether or not we got those results. If we did, then we would move on to the next module and verify each one until determining if we got the correct results on a higher level. If we didn't get the correct results then we would rethink our logic for that module and fix it before moving on. We also manually computed the results for most of the tests to match the outputs we were getting to what we expected to get.

Results

Test #	Results	Cycle Count
1	Correct	42
2	Correct	53
3	Correct	54

4	Correct	105

We also had some trouble connecting the appropriate signals to the test bench because those signals did not exist in our design, so modifying the existing signals to fit those expectations was challenging. We reach the correct result for test 4, but our PC got messed up at some point due to a bad branch. We were unable to determine the actual cause of this problem.

Extra Credit

We did not have time to implement any extra credit.