Progress Report

Datapath: David

Control: Luca

Arbiter: Irene

Report and roadmap: Irene

FUNCTIONALITIES:

For this checkpoint we implemented a basic pipeline that can handle all of the RV32I instructions. There is a module for each stage (IF, ID, EX, MEM, WB). The cpu was taken from previous MPs. As of now this pipeline cannot yet handle any control or data hazards.
We also created data forwarding, hazard detection and arbiter design for the next checkpoint.

TESTING STRATEGY:

Our testing strategy was to run the provided test code instruction by instruction, then examine the simulations to ensure that each instruction executed correctly.

Roadmap

Hazard detection unit: Luca, David

Forwarding unit: Luca, David

Branch prediction: Irene

Arbiter: Irene

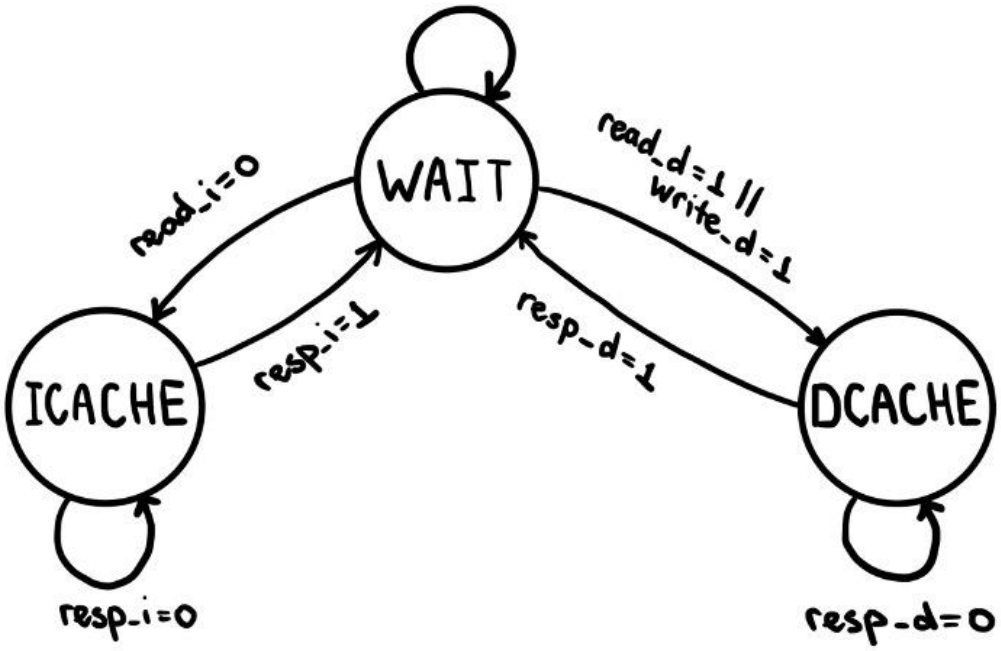
Report and roadmap: Irene

FEATURES AND FUNCTIONALITIES

For checkpoint 2, we will implement hazard detection and forwarding unit based on the design we created. In addition to this, we will also implement static-not-taken branch prediction for all control hazards. Another feature would be arbiter, that splits I-Cache and D-Cache. Instruction cache will have higher priority since it will be used more often.

Arbiter

STATE MACHINE



STATE DESCRIPTION

1. WAIT: idle state

Actions: none

Transitions:

if (read_i) next_state = icache;

else if ((read_d) || (write_d)) next_state = dcache;

else next_state = waiting;

2. ICACHE: instruction cache

Actions:

read = 1;

address = address_i;

rdata_i = rdata;

resp_i = resp;

Transitions:

if (resp == 1'b1) next_state = waiting;

else next_state = icache;

3. DCACHE: data cache

Actions:

address = address_d;

resp_d = resp;

if (read_d)

read = 1;

rdata_d = rdata;

if (write_d)

write = 1;

wdata = wdata_d;

Transitions:

if (resp == 1'b1) next_state = waiting;

else next_state = dcache;

SIGNALS

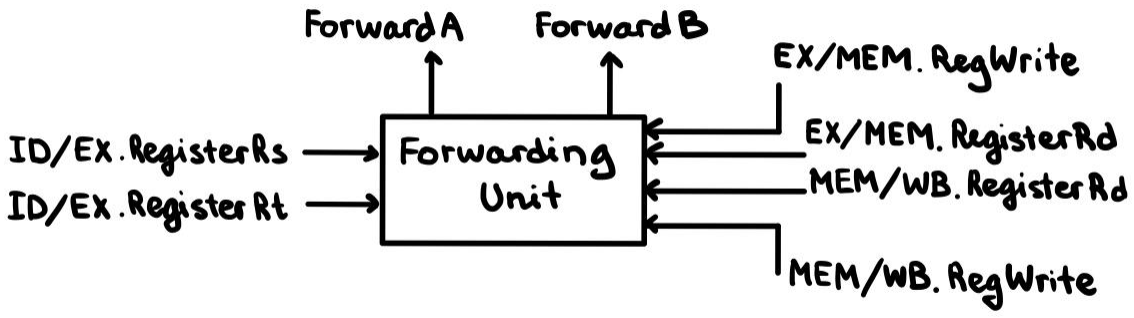
ICache: read_i, address_i, rdata_i, resp_i.

DCache: read_d, write_d, wdata_d, address_d, rdata_d, resp_d.

Cacheline-Adaptor: resp, rdata, read, write, wdata, address.

Forwarding Unit

(forwarding unit on datapath was shown during the design checkpoint)



SIGNALS:

ForwardA and ForwardB control left and right operant to ALU respectively.

Forward = 00 - ALU operand comes from regfile

Forward = 01 - ALU operand comes from data memory stage

Forward = 10 - ALU operand comes from previous ALU result

RegisterRd - destination register.

RegisterRs, RegisterRt - source registers.

FORWARDING LOGIC:

Default:

ForwardA = ForwardB = 00

if(EX/MEM.RegWrite and (EX/MEM.RegisterRd == ID/EX.RegisterRs) and (RegisterRd != 0))

ForwardA = 10

if(EX/MEM.RegWrite and (EX/MEM.RegisterRd == ID/EX.RegisterRt) and (RegisterRd != 0))

ForwardB = 10

if(MEM/WB.RegWrite and (EX/MEM.RegisterRd != ID/EX.RegisterRs) and

(MEM/WB.RegisterRd == ID/EX.RegisterRs) and (RegisterRd != 0)) Forward A = 01

if(MEM/WB.RegWrite and (EX/MEM.RegisterRd != ID/EX.RegisterRt) and (MEM/WB.RegisterRd == ID/EX.RegisterRt) and (RegisterRd != 0)) Forward B = 01

Hazard Detection Unit

(hazard detection unit on datapath was shown during the design checkpoint)

If hazard is detected, we should stall the pipeline. This can be achieved by setting MemWr and RegWr signals to 0 (stalling IF, ID), inserting a NOP (no stages after ID) and preventing PC update by setting load_PC to 0 (no additional IF stages of later instructions should appear). In addition to this, if cache miss occurs, pipeline should also be stalled.

HAZARD DETECTION LOGIC:

Hazard Detection for Rs = ((IF/ID.RegisterRs = ID/EX.RegisterRd) || (IF/ID.RegisterRs = EX/MEM.RegisterRd) || (IF/ID.RegisterRs = MEM/WB.RegisterRd)) and (RegisterRs != 0)

Hazard Detection for Rt = ((IF/ID.RegisterRt = ID/EX.RegisterRd) || (IF/ID.RegisterRt = EX/MEM.RegisterRd) || (IF/ID.RegisterRt = MEM/WB.RegisterRd)) and (RegisterRt != 0)