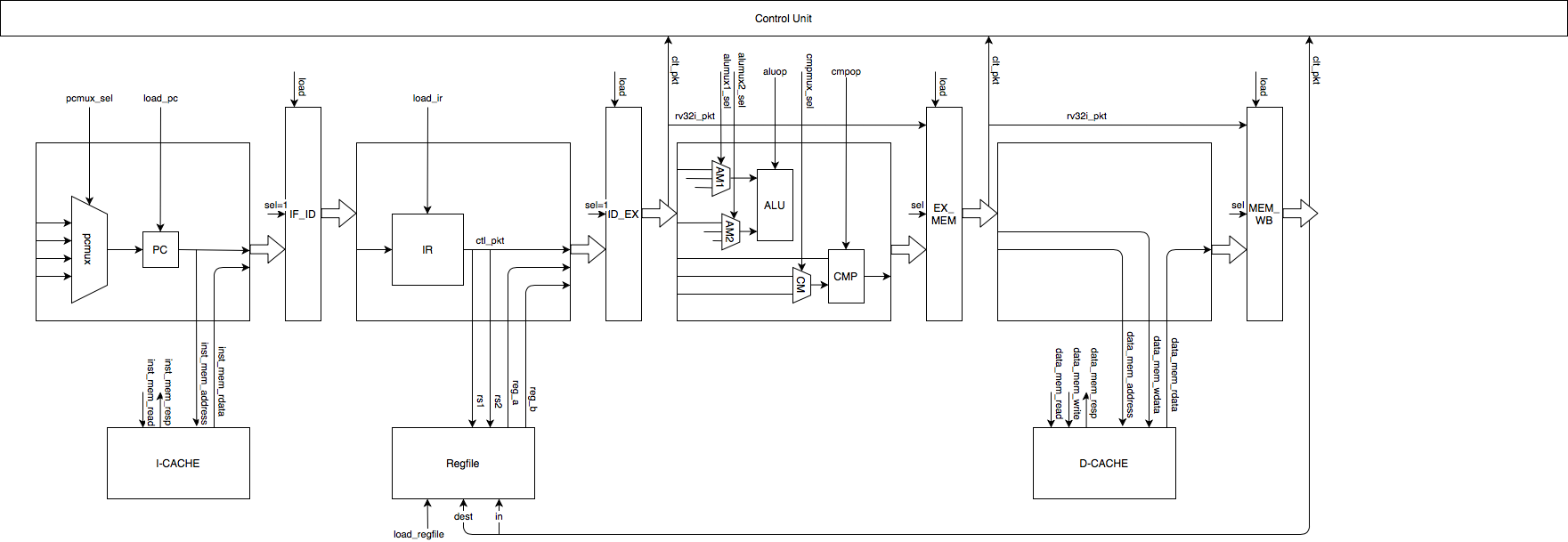
ECE411 MP4 CP1

MP4 Progress Report

Tingkai Liu, Zhi Cen

1 Design Checkpoint 0: RV32I ISA and basic pipelining

1.1 datapath



1.2 instruction analysis

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Opcode | IF | ID | EX | MEM | WB | Note |
| IMM | 1 | 1 | 1 | 0 | 1 |  |
| REG | 1 | 1 | 1 | 0 | 1 |  |
| LUI | 1 | 1 | 0 | 0 | 1 | Place u\_imm into register |
| MEM | 1 | 1 | 1 | 1 | 1/0 | Not need to write back for ST |
| AUIPC | 1 | 1 | 1 | 0 | 1 | Add something to PC and place the result into register. |
| BR | 1 | 1 | 1 | 0 | 0 | Conditional branch |
| JAL | 1 | 1 | 1 | 0 | 1 | Unconditional jump. |
| JALR | 1 | 1 | 1 | 0 | 1 | Unconditional jump. |

2 Checkpoint 1: RV32I ISA and basic pipelining

2.1 functionality and work distribution

|  |  |  |
| --- | --- | --- |
| Functionality | Notes | Work Distribution |
| a 5-stage pipeline cpu which can handle RV32I Instructions | 1. no hazard detection  2. full RV32I Instructions except FENCE\*, ECALL, EBREAK, and CSRR | Zhi: split mp2 cpu into 5 stages, unit test each stage and test the whole cpu with provided tb.  Tingkai: design control rom, assign control signals to each stage and connect all stages in datapath. |

2.2 update datapath

Diagram

Description automatically generated

2.3 update interface

2.3.1 Control ROM

module control\_rom(

input rv32i\_opcode opcode,

input logic [2:0] funct3,

input logic [6:0] funct7,

output rv32i\_ctrl\_packet\_t ctrl

);

2.3.2 Update the design of cpu\_control

|  |  |
| --- | --- |
| Before | Now |
| get decode information from each buffer and output all control signals from cpu\_control. | decode instruction by control\_rom and save them in ctrl packet. In cpu\_control, only deal with signals that can’t be determined at ID stage. |

2.3.3 testing

|  |  |
| --- | --- |
| Testing Type |  |
| Unit test | if\_unit\_test, id\_unit\_test, ex\_unit\_test, wb\_unit\_test |
| Using source code | mp4\_cp1.s |

2.3.4 bug log

|  |  |  |
| --- | --- | --- |
| Bug | Reason | Note |
| PC didn't change at branch | forgot to set ALU for EX when BR |  |
| Invalid data read from D memory | forgot to mask the low 2 bits for mem\_addr | I-cache access is always 4 byte allign so no need for that |
| When EX indicates halt, the WB for previous inst may not finish | The way for detecting halt may need to change |  |
| missing information for rvfimon | For rvfi monitor, we need to save additional information in packet | data\_mem\_rdata == mdrreg\_out |
| rd\_addr doesn’t match | all signals for rvfi monitors should be sent from wb stage | Only commit at wb.load\_buffers && wb.inst != nop |
| pc\_wdata doesn’t match at beq inst | pc\_wdata should be pc+4 or alu\_out |  |

3 Design Checkpoint 1: L1 caches, hazards and static branch prediction

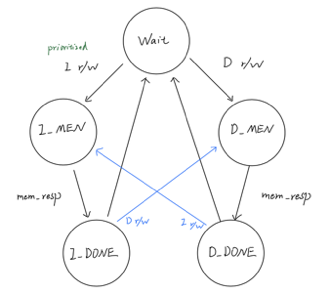
3.1 arbiter design

3.1.1 datapath

Diagram, engineering drawing

Description automatically generated

3.1.2 state machine

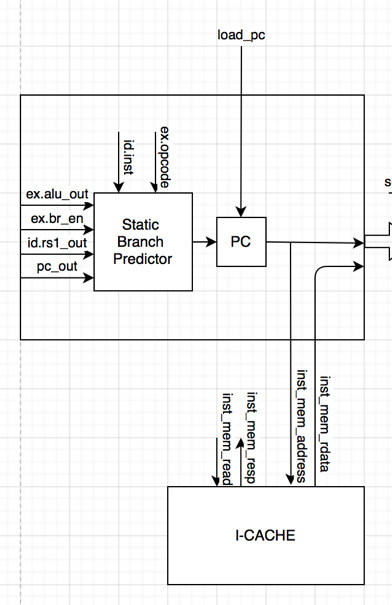


3.1.3 state description

|  |  |  |  |
| --- | --- | --- | --- |
| State | Description | Next | Output |
| WAIT | Wait until a access request happens from either I-cache or D-cache | If I-cache requests, to I\_MEM.  If D-cache requests, to D\_MEM. I-cache request is check first (Prioritized) | 0 |
| I\_MEM | Access memory according to the signals from I-cache | If mem\_resp, to I\_DONE | Pass signals of I-cache to physical memory |
| I\_DONE | Finish I-cache request, and check whether need to do D-cache request | If D-cache requests, to D\_MEM, else to WAIT | inst\_mem\_resp = 1 |
| D\_MEM | Access memory according to the signals from D-cache | If mem\_resp, to D\_DONE | Pass signals of D-cache to physical memory |
| I\_DONE | Finish D-cache request, and check whether need to do I-cache request | If I-cache requests, to I\_MEM, else to WAIT | data\_mem\_resp = 1 |

3.2 static branch predictor

3.2.1 datapath



3.2.2 pc update table

|  |  |
| --- | --- |
| Opcode | PC |
| BRANCH | PC <- PC + (br\_en? b\_imm : 4) |
| JAL | PC <- PC + j\_imm |
| JALR | PC <- {(rs1\_out + i\_imm) [31:1], 1’b0} |
| Other opcodes | PC <- PC + 4 |

3.2.3 control logic

If EX.opcode == op\_branch && br\_en

ID.pc = nop

ID.ctrl = nop

IF.pcmux\_out = EX.alu\_out

If ID.opcode == op\_jal

IF.pcmux\_out = IF.pc\_out + ID.j\_mm

If ID.opcode == op\_jalr

IF.pcmux\_out = {(ID.rs1\_out + ID.i\_imm)[31:1], 1’b0}

Else

IF.pcmux\_out = IF.pc\_out + 4

3.3 forwarding

3.3.1 datapath

Diagram

Description automatically generated

3.3.2 control

