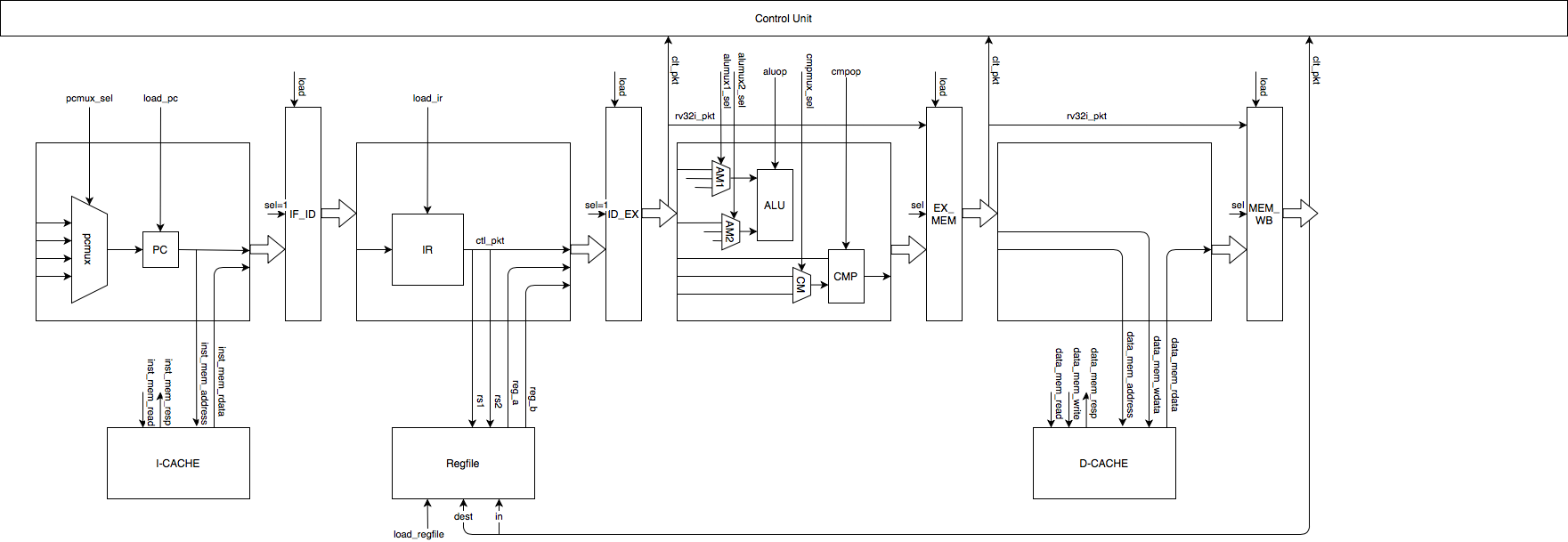
ECE411 MP4 CP1

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 design

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. |

2.4 testing and bug log

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

|  |  |  |
| --- | --- | --- |
| 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

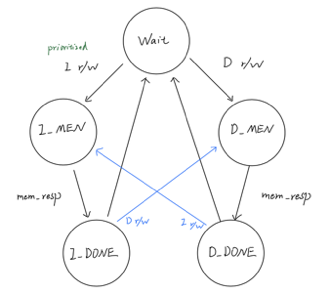
3.1 arbiter design

3.1.1 datapath

Diagram, engineering drawing

Description automatically generated

3.1.2 state machine

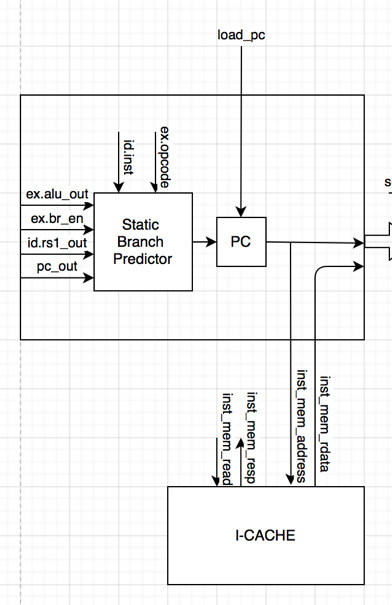


3.1.3 state description

|  |  |  |  |
| --- | --- | --- | --- |
| 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

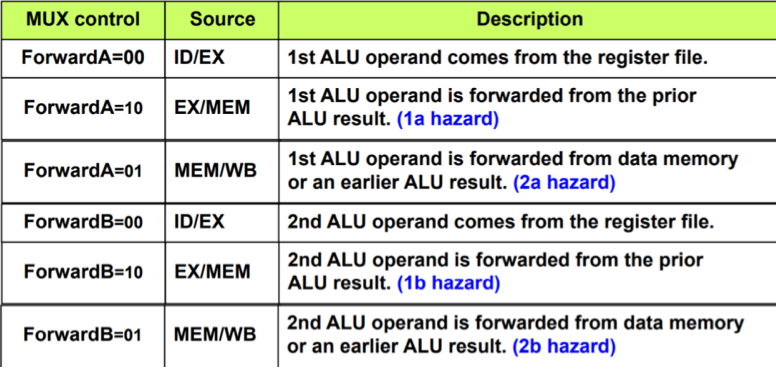
3.3 forwarding

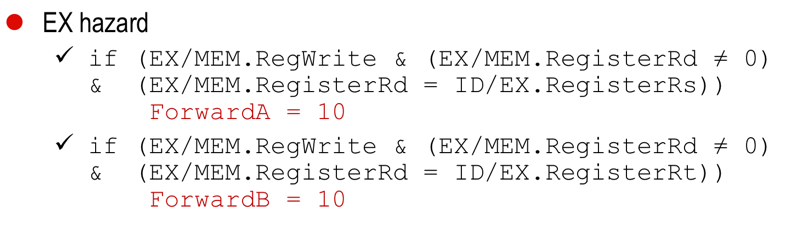
3.3.1 datapath

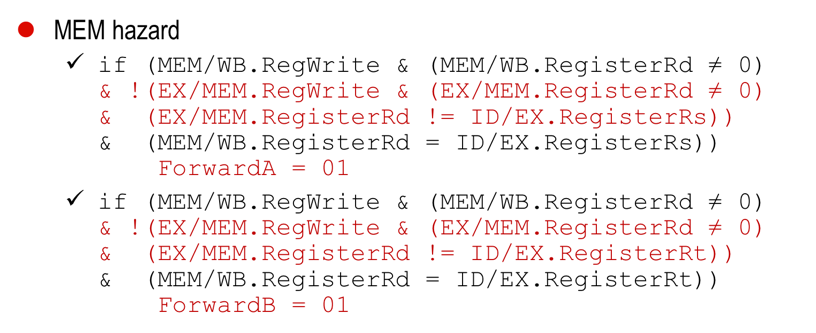
Diagram

Description automatically generated

3.3.2 control





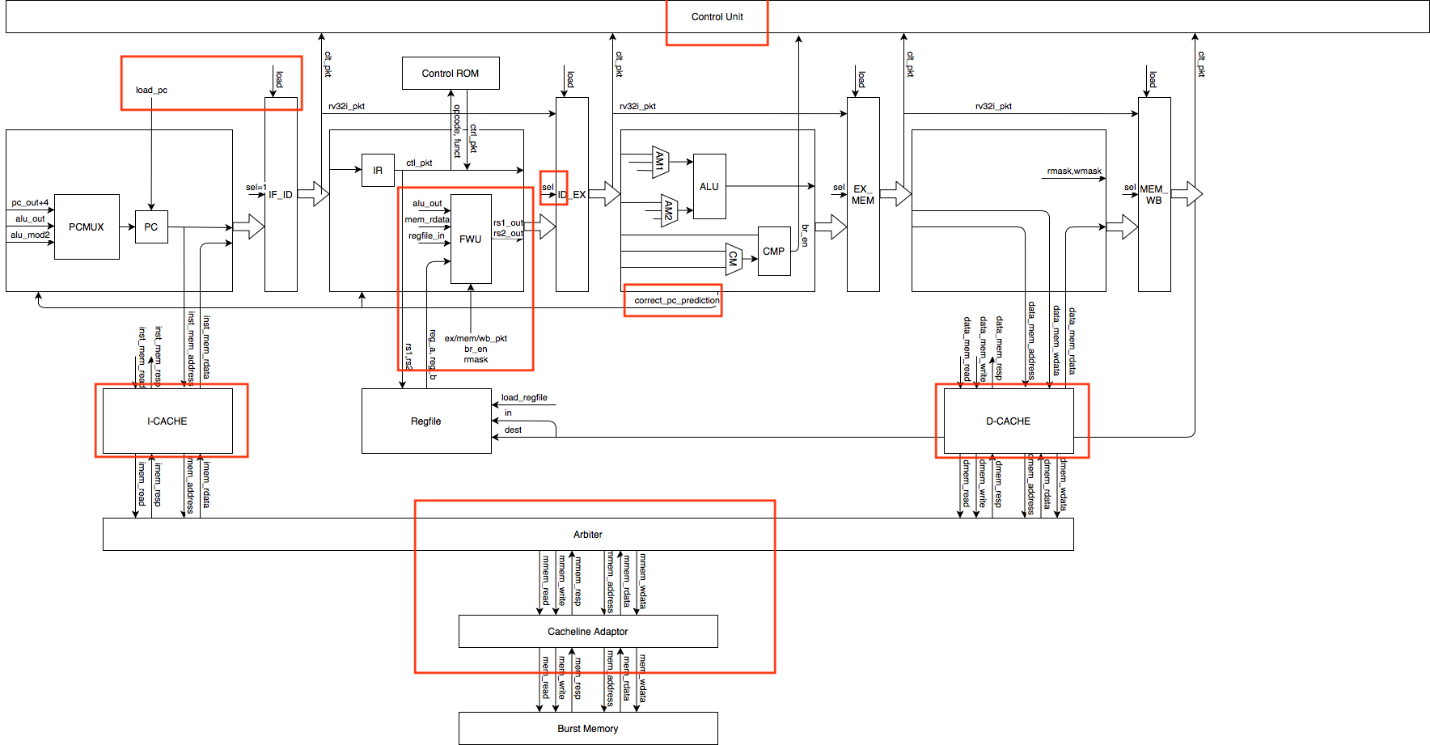


4 Checkpoint 2: L1 caches, hazards and static branch prediction

4.1 functionality and work distribution

|  |  |  |
| --- | --- | --- |
| Functionality | Notes | Work Distribution |
| Hazard, static branch prediction | 1. data hazard: forwarding  2. control hazard: static-not-taken branch prediction | Tingkai |
| L1 cache, arbiter | 1. arbiter connected to I-Cache, D-cache and cacheline adaptor  2. no cache coherence | Zhi |

4.2 update datapath



4.3 update design

4.3.1 updates in L1 cache

|  |  |
| --- | --- |
| Before | Now |
| mp3 cache takes two cycles for cache hit | Now our cache takes one cycle for cache hit by changing read to combinational logic |

4.3.2 updates in arbiter

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| State | Description | Next(Before) | Next(Now) | Output |
| 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 | WAIT | inst\_mem\_resp = 1 |
| 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 | WAIT | data\_mem\_resp = 1 |

4.3.3 updates in forwarding

Key idea of forwarding: At ID stage, when reading the register values, the latest version of register value may not be write-back. As a result, we design the forwarding to be getting the latest value of register at ID stage if necessary, which means instead of connecting forwarded value to EX, MEM, WB, we only forward the value to ID stage and store the forward value in the packet and pass it along the pipeline.

Note that for register dependency happens when the previous instruction is a load instruction, the pipeline need to stall since the next instruction needs the value at EX at the same time MEM is getting the value. The way we solve that is when such cases is detected, the IF, IF\_ID, and ID are stalled by keeping the load to be 0, and an invalid instruction is inserted in ID\_EX and keep ID\_EX, EX, EX\_MEM, MEM, MEM\_WB, WB moving.

4.3.4 updates in control hazard

Key idea for solving control hazard: mark the mis-fetched instruction to be invalid and stop them from updating memory and registers. Since we are using static no-taken branch prediction, when we see a br/jal/jalr at EX stage calculated the correct branch target which is not pc+4, we will mark the instruction in IF\_ID and ID\_EX to be invalid since they are mis-fetched. After those instructions are marked as invalid, they won’t be able to write to memory or registers.

4.4 testing and bug log

|  |  |
| --- | --- |
| Testing Type |  |
| Unit test | cache\_unit\_test, arbiter\_unit\_test, cache\_top\_unit\_test |
| Using source code | mp4\_cp2.s, factorial.s, mp4\_cp3.s |

|  |  |  |
| --- | --- | --- |
| Bug | Reason | Note |
| iteration limit reached | For one-cycle hit cache, inst\_mem\_read and inst\_mem\_resp are circular assigned. |  |
| cannot switch to inst/data\_mem\_state after inst/data\_fin\_state | TODO | TODO |
| stall at one instruction | When the inst at mem is invalid, forgot to change the condition for done and load\_buffer is never asserted | also need to change if branch prediction is added! |
| write nonzero data to x0 | When write to x0, should write 0 anyway |  |
| Some instructions didn’t get the forwarded value from last but two instruction. | It was ignored that for the instructions at MEM, its result from EX is still needed for forwarding. |  |
| Stalling didn't get the forwarded value | Forgot the forward setting (only) for lw |  |

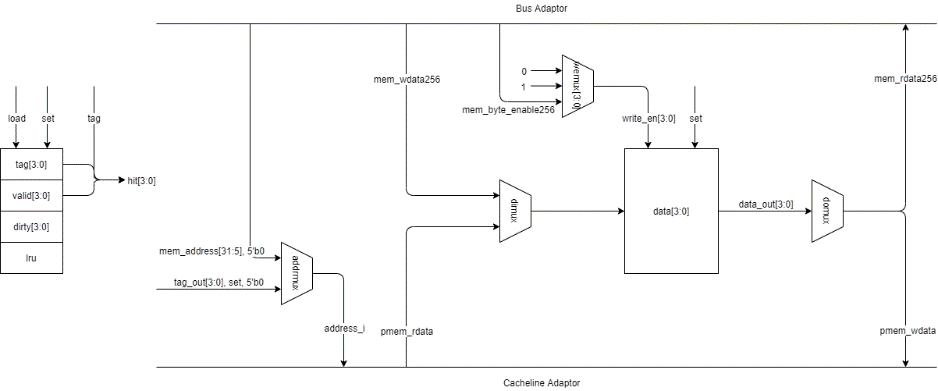
5 Design Checkpoint 2: Advanced Design Options

5.1 Roadmap

|  |  |  |  |
| --- | --- | --- | --- |
| Functionality | Points | Notes | Work Distribution |
| 1. 4-way cache [2]  2. Global 2-level branch history table [3]  3. Non-blocking L1 cache [8] | 13 |  | Zhi |
| 1. Tomasulo [20] | 20 |  | Tingkai |

5.2 4-way cache design

5.2.1 datapath



5.3 global 2-level branch predictor

5.3.1 datapath

