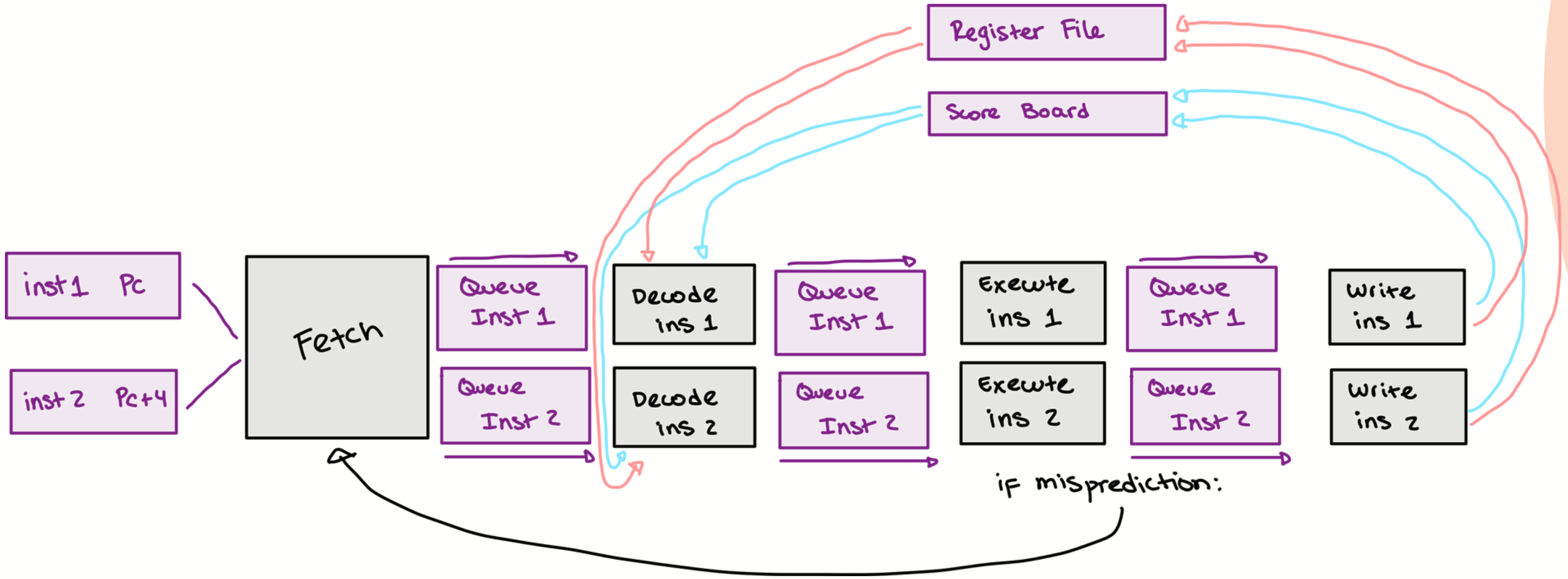
SUPERSCALAR
&
2-WAY CACHES

Ana Camba Gomes
&
Fabiana Gonzalez

Superscalar background

- Tries to execute multiple instructions during a single clock cycle if possible.
- Our implementation: Two instructions
- Potentially increasing the performance of the processor without increasing its clock speed

Diagram for Superscalar



Fetch Structure

- Two Caches and Two interfaces:

```
interface Cache32_data;
    method Action putFromProc(CacheReq e); //  deli
    method ActionValue#(Word) getToProc(); // shoul
    method ActionValue#(MainMemReq) getToMem(); //
    method Action putFromMem(MainMemResp e); // wil
endinterface

interface Cache32_ins;
    method Action putFromProc(CacheReq e); //  deli
    method ActionValue#(OneOrTwoWords) getToProc();
    method ActionValue#(MainMemReq) getToMem(); //
    method Action putFromMem(MainMemResp e); // wil
```

```
module mkCacheInterface(CacheInterface);
    let verbose = True;
    MainMem mainMem <- mkMainMem();
    Cache512 cacheL2 <- mkCache;
    Cache32_ins cacheI <- mkCache32_ins;
    Cache32_data cacheD <- mkCache32_data;
```

- OneorTwoWords struct : instruction cache takes two instructions instead of one if we are inside the boundary

```
typedef struct {
    Word ins1;
    Maybe#(Word) ins2;
    }
    OneOrTwoWords deriving (Eq, FShow, Bits);
```

```
function Bool notLineBoundary(Bit#(32) pcf);
    Bool check_line;

    let offset = parseAddress(pcf).offset;

    if (offset == ~0) begin
        check_line = False;  // on a boundary
    end
    else begin
        check_line = True;
    end

    return check_line;
endfunction
```

Fetch Bugs

- Wrong checking in the boundary function. Used ‘index’ instead of ‘offset’ to determine if in boundary or not

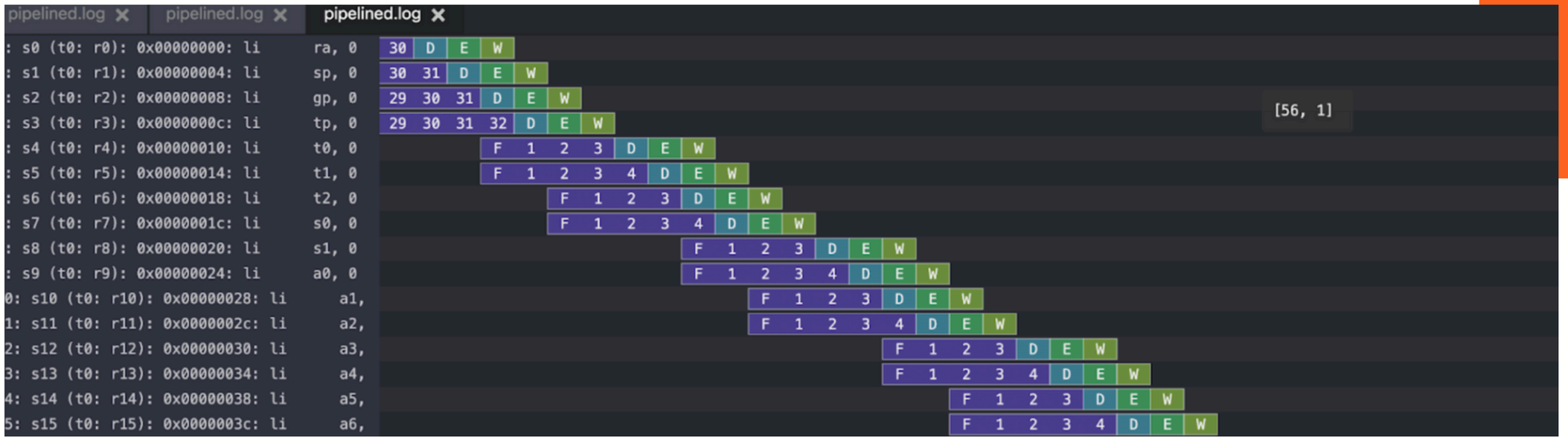
- Konata Id’s. Modifications on the Konata so we can see the two fetch stages

```
function ActionValue#(KonataId) fetch1Konata(File f, Reg#(KonataId) konataCtr,ThreadId tid);
    // Include the declaration of the instr
    actionvalue
        konataCtr <= konataCtr + 1;
        $fdisplay(f,"I\t%d\t%d\t%d",konataCtr,konataCtr,tid);
        $fdisplay(f,"S\t%d\t%d\t%s",konataCtr,0,"F");
        return konataCtr;
    endactionvalue
endfunction

function ActionValue#(KonataId) nfetchKonata(File f, Reg#(KonataId) konataCtr,ThreadId tid, Integer k);
    // Return the first id of the consecutive k id allocated
    actionvalue
        konataCtr <= konataCtr + fromInteger(k);
        for (Integer j = 0; j < k; j = j + 1) begin
            let n = konataCtr + fromInteger(j);
            $fdisplay(f,"I\t%d\t%d\t%d",n,n,tid);
            $fdisplay(f,"S\t%d\t%d\t%s", n ,0,"F");
        end
        return konataCtr;
    endactionvalue
endfunction
```

```
let numFetched = notLineBoundary(pc_fetched) ? 2 : 1;

let iid <- nfetchKonata(lfh, fresh_id, 0, numFetched);
```

```
f2d.enq1(F2D{pc: pc_fetched, ppc: pc_fetched+4, epoch: epoch[2], k_id: iid });

if(notLineBoundary(pc_fetched)) begin

    f2d.enq2(F2D{pc: pc_fetched+4, ppc: pc_fetched+8, epoch: epoch[2], k_id: iid +1 });
```

Fetch


pipelined.log
pipelined.log
pipelined.log
: s0 (t0: r0): 0x00000000: li ra, 0
: s1 (t0: r1): 0x00000004: li sp, 0
: s2 (t0: r2): 0x00000008: li gp, 0
: s3 (t0: r3): 0x0000000c: li tp, 0
: s4 (t0: r4): 0x00000010: li t0, 0
: s5 (t0: r5): 0x00000014: li t1, 0
: s6 (t0: r6): 0x00000018: li t2, 0
: s7 (t0: r7): 0x0000001c: li s0, 0
: s8 (t0: r8): 0x00000020: li s1, 0
: s9 (t0: r9): 0x00000024: li a0, 0
0: s10 (t0: r10): 0x00000028: li a1,
1: s11 (t0: r11): 0x0000002c: li a2,
2: s12 (t0: r12): 0x00000030: li a3,
3: s13 (t0: r13): 0x00000034: li a4,
4: s14 (t0: r14): 0x00000038: li a5,
5: s15 (t0: r15): 0x0000003c: li a6,
[56, 1]

Decode Structure

- Decode:

```
rule decode if (!starting);
```

- Decode2:

```
rule decode2 if (!starting && (decode1_done[1] == 1) && fromImem.deqReadyN(2) );
```

- Changed getIResp to accept the response of the cache of OneOrTwoWords

```
method Action getIResp(OneOrTwoWords a);
    // fromImem superscalar enq1 enq2 depending if validity
    fromImem.enq1(a.ins1);

    if (isValid(a.ins2)) begin
        fromImem.enq2(fromMaybe(?,a.ins2));
    end

endmethod
```

- Checked if both can dequeue

```
method Bool deqReadyN(Bit#(2) n);  // valid 1 or 2
return case (n) matches
    0: False;
    1: (can_deq1 || can_deq2);
    2: (can_deq1 && can_deq2);
    default: False;
endcase;
endmethod
```

Decode Bugs

- Scoreboard manipulations so the second instruction can see the changes that the first instruction made
  - Decode:

```
if(scoreboard[rs1_idx][4] == 0 && scoreboard[rs2_idx][4] == 0)begin

    if (decodedInst.valid_rd && rd_idx !=0) begin
        scoreboard[rd_idx][4] <= scoreboard[rd_idx][4] + 1;

    end
```

  - Execute:

```
else begin // SQUASH INSTRUCTION. EPOCH DID NOT MATCH SO WE DO NOT WANT THIS INSTRUCTION
    squashed.enq(from_decode.k_id);
    if (from_decode.dinst.valid_rd && getInstFields(from_decode.dinst.inst).rd != 0) begin
        scoreboard[getInstFields(from_decode.dinst.inst).rd][2] <= scoreboard[getInstFields(from_decode.dinst.inst).rd][2] - 1;
    end

end
```

  - Writeback:

```
if (from_execute.dinst.valid_rd) begin
    let rd_idx = fields.rd;
    if (from_execute.dinst.valid_rd && rd_idx != 0) begin
        scoreboard[rd_idx][0] <= scoreboard[rd_idx][0] - 1;
    end
    if (rd_idx != 0) begin rf[rd_idx][0] <=data; end
end
```

Decode Bugs

- Implementation

```
Ehr#(2, Bit#(1)) decode1_done <- mkEhr(0);
```

- Instruction pass without stalling in Decode 1

```
if(scoreboard[rs1_idx][4] == 0 && scoreboard[rs2_idx][4] == 0)begin

    if (decodedInst.valid_rd && rd_idx !=0) begin
        scoreboard[rd_idx][4] <= scoreboard[rd_idx][4] + 1;

    end

    d2e.enq1(D2E{dinst: decodedInst, pc: from_fetch.pc, ppc: from_fetch.ppc, epoch: from_fetch.epoch, rv1: rs1, rv2: rs2, k_id: from_fetch.k_id });

    f2d.deq1();

    decode1_done[0] <= 1;
```

- Instruction stalling in Decode 1

```
else begin
    // STALL
    labelKonataLeft(lfh,from_fetch.k_id, $format("instruction 1"));
    labelKonataLeft(lfh,from_fetch.k_id, $format(" rs1 idx: [%0d] content: (%x) ", rs1_idx, scoreboard[rs1_idx][2]));
    labelKonataLeft(lfh,from_fetch.k_id,$format(" rs2 idx: [%0d] content: (%x) ", rs2_idx, scoreboard[rs2_idx][2]));
    decode1_done[0] <= 0;

end
```

Strategies for debug

- Implement a rule that breaks after certain amount of cycles. Prevents Konata from lagging

```
rule tic;
    cycle_count <= cycle_count + 1;
    if (cycle_count >= 10000) begin
        $display("Capped at 10000 cycles in top_pipelined.bsv");
        $finish;
    end
endrule
```

- Comment code and check .schem file for blocking rules

```
Blocking rules: writeback
```

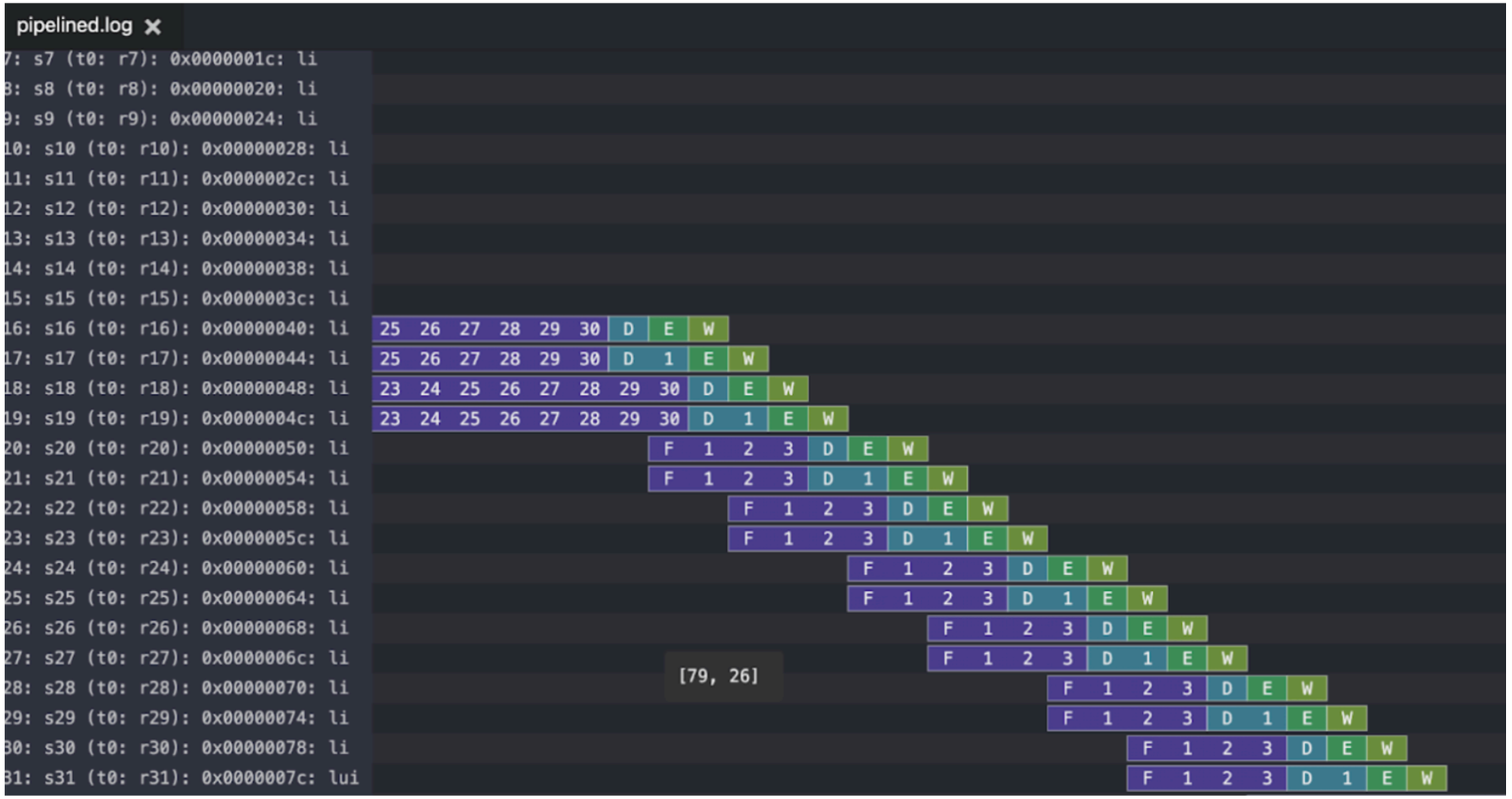
```
Blocking rules: (none)
```

- Instruction comments in Konata

```
labelKonataLeft(lfh,from_decode.k_id, $format("execute 1"));
```

```
a1, 0instruction 1execute 1
a2, 0instruction 2execute 2
```

Decode


pipelined.log
7: s7 (t0: r7): 0x0000001c: li
8: s8 (t0: r8): 0x00000020: li
9: s9 (t0: r9): 0x00000024: li
10: s10 (t0: r10): 0x00000028: li
11: s11 (t0: r11): 0x0000002c: li
12: s12 (t0: r12): 0x00000030: li
13: s13 (t0: r13): 0x00000034: li
14: s14 (t0: r14): 0x00000038: li
15: s15 (t0: r15): 0x0000003c: li
16: s16 (t0: r16): 0x00000040: li
17: s17 (t0: r17): 0x00000044: li
18: s18 (t0: r18): 0x00000048: li
19: s19 (t0: r19): 0x0000004c: li
20: s20 (t0: r20): 0x00000050: li
21: s21 (t0: r21): 0x00000054: li
22: s22 (t0: r22): 0x00000058: li
23: s23 (t0: r23): 0x0000005c: li
24: s24 (t0: r24): 0x00000060: li
25: s25 (t0: r25): 0x00000064: li
26: s26 (t0: r26): 0x00000068: li
27: s27 (t0: r27): 0x0000006c: li
28: s28 (t0: r28): 0x00000070: li
29: s29 (t0: r29): 0x00000074: li
30: s30 (t0: r30): 0x00000078: li
31: s31 (t0: r31): 0x0000007c: lui
25 26 27 28 29 30 D E W
25 26 27 28 29 30 D 1 E W
23 24 25 26 27 28 29 30 D E W
23 24 25 26 27 28 29 30 D 1 E W
F 1 2 3 D E W
F 1 2 3 D 1 E W
F 1 2 3 D E W
F 1 2 3 D 1 E W
F 1 2 3 D E W
F 1 2 3 D 1 E W
F 1 2 3 D E W
F 1 2 3 D 1 E W
[79, 26]
F 1 2 3 D E W
F 1 2 3 D 1 E W
F 1 2 3 D E W
F 1 2 3 D 1 E W

Execute Structure

- Vector to check if we have a memory / control instruction

```
Vector#(2,Ehr#(2, Bit#(1))) instruction_bool <- replicateM(mkEhr(1));
```

- Decode 1

```
if(isMemoryInst(decodedInst) || isControlInst(decodedInst)) begin

    instruction_bool[0][0] <= 0;

end
```

- Decode 2

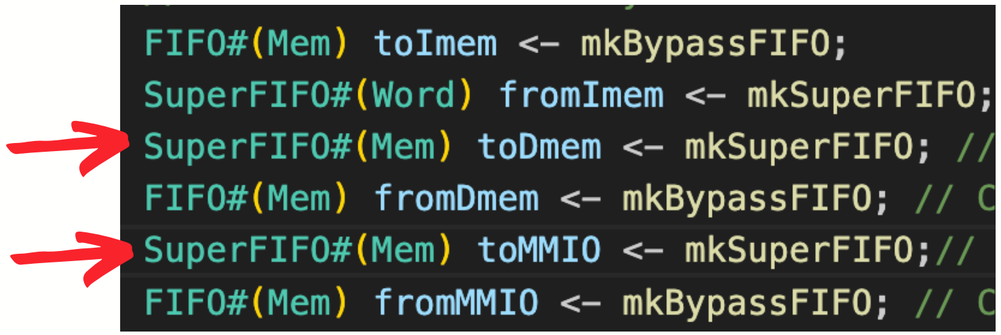
```
if(isMemoryInst(decodedInst) || isControlInst(decodedInst)) begin

    instruction_bool[1][1] <= 0;
end
```

- Execute 2 where we check to see if we can do two alu instructions at a time

```
rule execute2 if (!starting  && (instruction_bool[1][0]==1) && (instruction_bool[1][1]==1));
```

Execute Bugs

- Blocking Rule Prevention



```
FIFO#(Mem) toImem <- mkBypassFIFO;
SuperFIFO#(Word) fromImem <- mkSuperFIFO;
SuperFIFO#(Mem) toDmem <- mkSuperFIFO; //
FIFO#(Mem) fromDmem <- mkBypassFIFO; // C
SuperFIFO#(Mem) toMMIO <- mkSuperFIFO;//
FIFO#(Mem) fromMMIO <- mkBypassFIFO; // C
```

```
method ActionValue#(Mem) getDReq();
    toDmem.deq1();
    return toDmem.first1();
endmethod
```

```
method ActionValue#(Mem) getMMIOReq();
    toMMIO.deq1();
    return toMMIO.first1();
endmethod
```

- Debug Display Statements

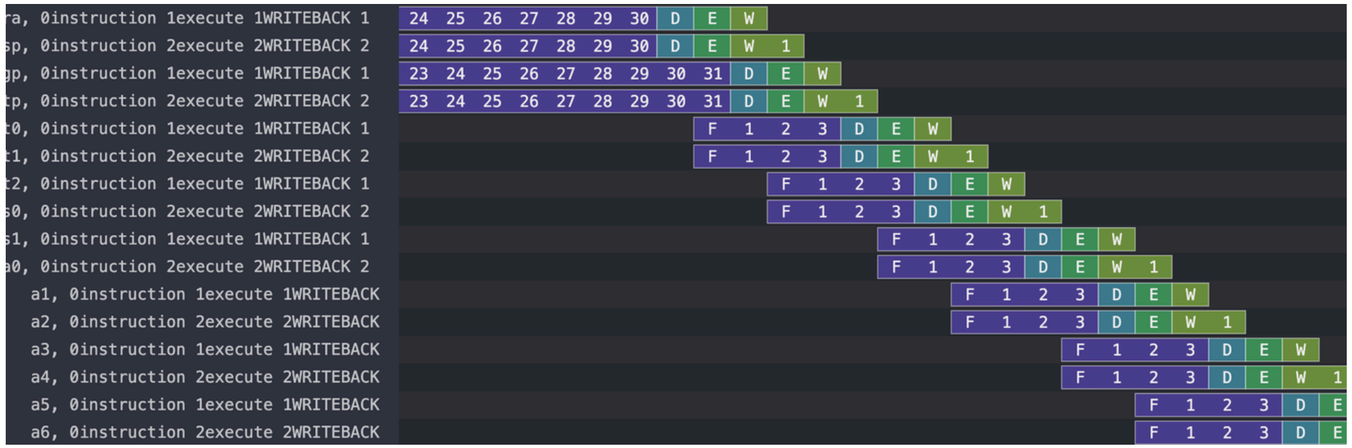
```
0: s20 (t0: r20): 0x00000050: li      s5, 0instruction 1execute 1
1: s21 (t0: r21): 0x00000054: li      s6, 0instruction 2execute 2
2: s22 (t0: r22): 0x00000058: li      s7, 0instruction 1execute 1
3: s23 (t0: r23): 0x0000005c: li      s8, 0instruction 2execute 2
```

Execute


: s8 (t0: r8): 0x00000020: li s1, 0instruction 1execute 1
: s9 (t0: r9): 0x00000024: li a0, 0instruction 2execute 2
0: s10 (t0: r10): 0x00000028: li a1, 0instruction 1execute 1
1: s11 (t0: r11): 0x0000002c: li a2, 0instruction 2execute 2
2: s12 (t0: r12): 0x00000030: li a3, 0instruction 1execute 1
3: s13 (t0: r13): 0x00000034: li a4, 0instruction 2execute 2
4: s14 (t0: r14): 0x00000038: li a5, 0instruction 1execute 1
5: s15 (t0: r15): 0x0000003c: li a6, 0instruction 2execute 2
6: s16 (t0: r16): 0x00000040: li a7, 0instruction 1execute 1
7: s17 (t0: r17): 0x00000044: li s2, 0instruction 2execute 2
8: s18 (t0: r18): 0x00000048: li s3, 0instruction 1execute 1
9: s19 (t0: r19): 0x0000004c: li s4, 0instruction 2execute 2
0: s20 (t0: r20): 0x00000050: li s5, 0instruction 1execute 1
1: s21 (t0: r21): 0x00000054: li s6, 0instruction 2execute 2
2: s22 (t0: r22): 0x00000058: li s7, 0instruction 1execute 1
3: s23 (t0: r23): 0x0000005c: li s8, 0instruction 2execute 2
4: s24 (t0: r24): 0x00000060: li s9, 0instruction 1execute 1
5: s25 (t0: r25): 0x00000064: li s10, 0instruction 2execute 2
6: s26 (t0: r26): 0x00000068: li s11, 0instruction 1execute 1
7: s27 (t0: r27): 0x0000006c: li t3, 0instruction 2execute 2
8: s28 (t0: r28): 0x00000070: li t4, 0instruction 1execute 1
9: s29 (t0: r29): 0x00000074: li t5, 0instruction 2execute 2
0: s30 (t0: r30): 0x00000078: li t6, 0instruction 1execute 1
1: s31 (t0: r31): 0x0000007c: lui sp, 0x10000instruction 2execute 2
2: s32 (t0: r32): 0x00000080: addi sp, sp, -16instruction 1execute 1
3: s33 (t0: r33): 0x00000084: jal pc + 0x1018instruction 2execute 1Thi
4: s34 (t0: r0): 0x00000088: li a0, 0instruction 1execute 1
5: s35 (t0: r0): 0x0000008c: jal pc + 0xfd8instruction 2execute 1
29 30 D E W
29 30 D E 1 W
27 28 29 30 D E W
27 28 29 30 D E 1 W
F 1 2 3 D E W
F 1 2 3 D E 1 W
F 1 2 3 D E W
F 1 2 3 D E 1 W
F 1 2 3 D E W
F 1 2 3 D E 1 W
F 1 2 3 D E W
F 1 2 3 D E 1 W
F 1 2 3 D E W
F 1 2 3 D E 1 W
F 1 2 3 D E W
F 1 2 3 D E 1 W
F 1 2 3 4 5 6 7
F 1 2 3 4 5 6 7
F 1 2 3 4 5
F 1 2 3 4 5

Writeback Structure

- If ALU instructions

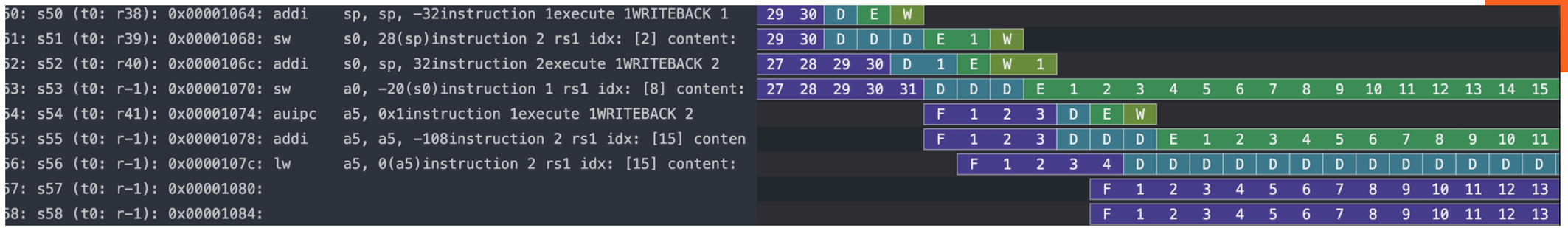


- Second writeback is only for non-memory instructions

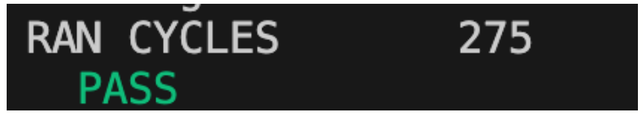
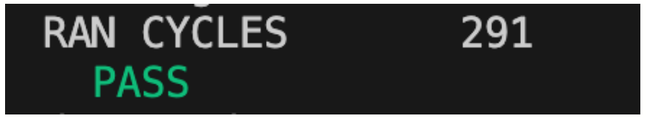
```
rule writeback2 if (!starting);
    let from_execute = e2w.first2();


    if (isMemoryInst(from_execute.dinst)) begin
        //STALL


    end
```

Current Writeback Bugs

- Crashes with Memory instructions in Writeback

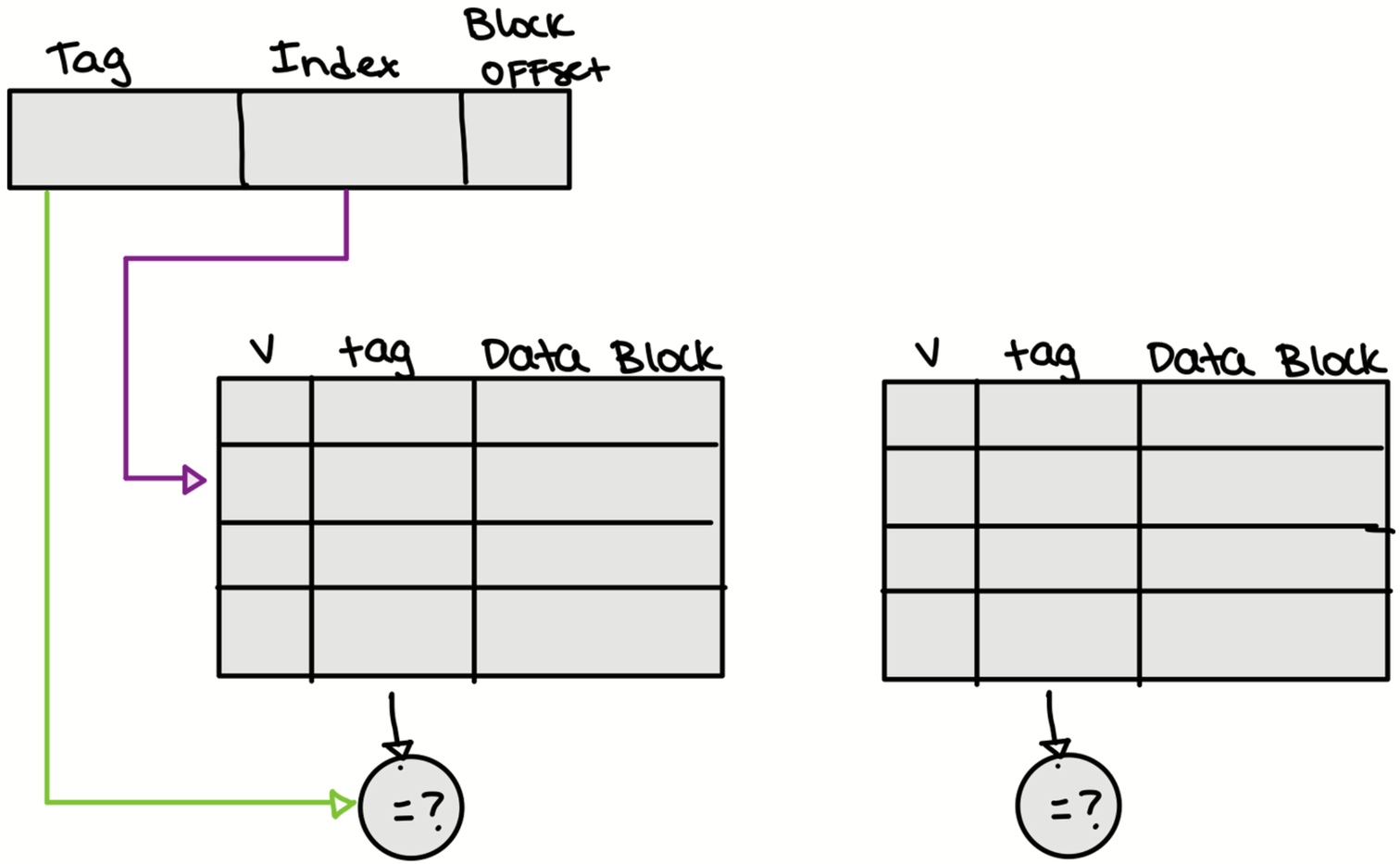


Test Updates

	SUPERSCALAR	LAB 3B
• Add32		
• Mul32		

Next step

- Two way cache instead of direct mapped cache



Thank you for listening!