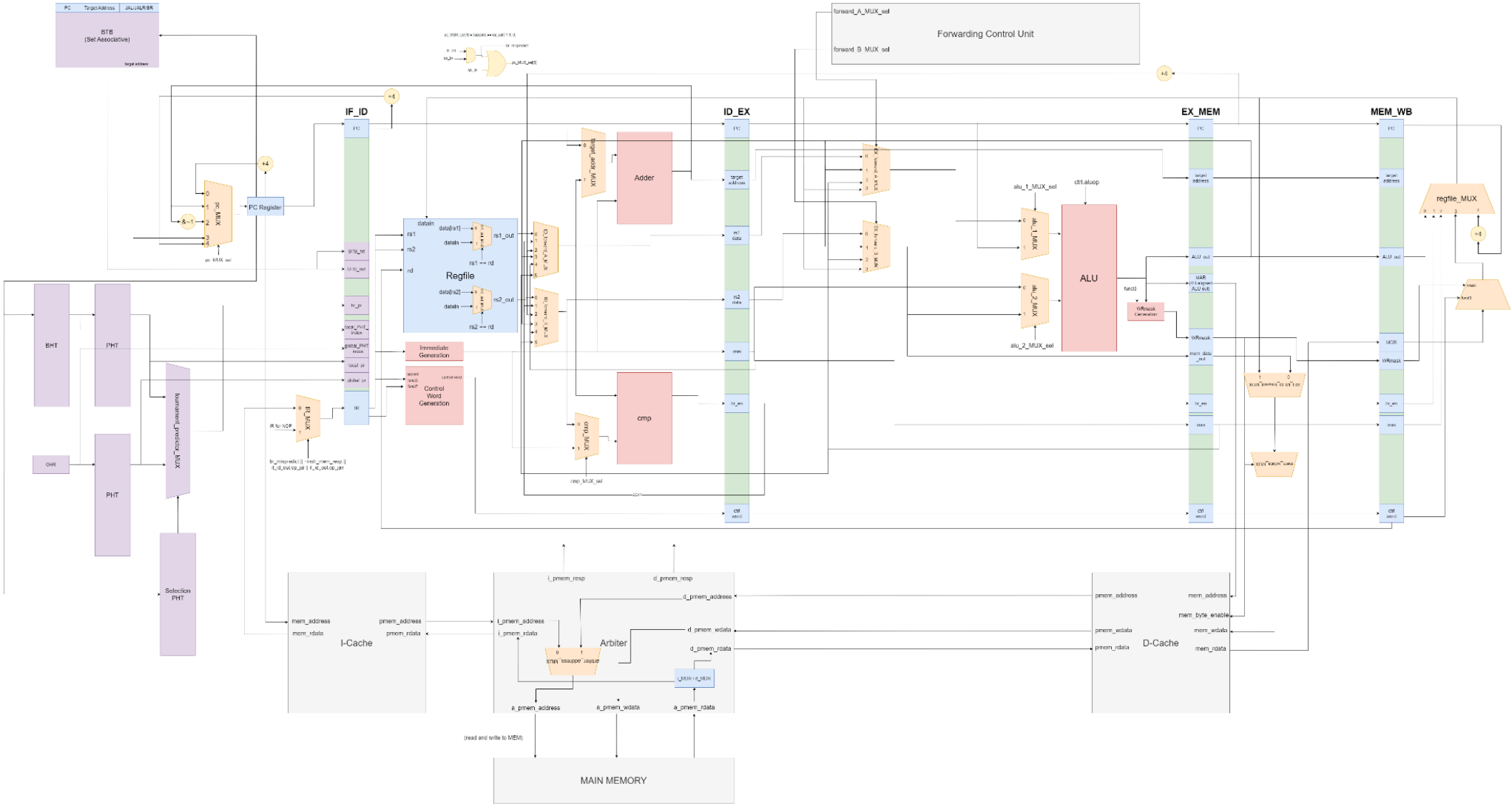
CP2 Design Document: Advanced Features

HighRISCHighReward

Datapath Diagram



Branch Predictor

(Local branch history table [2], Global 2-level branch history table [3], Tournament branch predictor [5], Branch target buffer, support for jumps [1], 4-way set associative or higher BTB [3])

```
/* Drive pc_MUX_sel by IF stage first */
if (btb_hit == 1)
begin
	if (if_id_in.btb_out.br_jal_jalr == op_br)
	begin
		if (br_pr == 1)
			pc_MUX_sel = btb_out;
		else
			pc_MUX_sel = pc_plus4;
	end

	else if (if_id_in.btb_out.br_jal_jalr == op_jal)
		pc_MUX_sel = btb_out;
	else if (if_id_in.btb_out.br_jal_jalr == op_jalr)
		pc_MUX_sel = btb_out;
end
else
	pc_MUX_sel = pc_plus4;
```

```
/* Drive pc_MUX_sel by ID stage  (Overwrite pc_MUX_sel) */
if(if_id_out.opcode == op_br)
begin
	if(btb_hit == 1 && (br_pr != br_en))
		if_id_reg_flush = 1;
		if(br_en == 1)
```

```
                pc_MUX_sel = adder_out;
            if(br_en == 0)
                pc_MUX_sel = if_id_out_pc_plus4;  // predicted T, but resolved NT, should be PC of instr + 4
        if(btb_hit == 0 && br_en == 1)  // whenever BTB is a miss, always fetch PC+4
        begin
            if_id_reg_flush = 1;
            update_BTB(if_id_out.PC, id_ex_in.target_address, op_br);
            pc_MUX_sel = adder_out;
        end
end
else if(if_id_out.opcode == op_jal)
begin
        if(btb_hit == 0)
        begin
            if_id_reg_flush = 1;
            update_BTB(if_id_out.PC, id_ex_in.target_address, op_jal);
            pc_MUX_sel = adder_out;
        end
end
else if(if_id_out.opcode == op_jalr)
begin
        if(btb_hit == 1 && (if_id_out.btb_out.target_address != id_ex_in.target_address))
        begin
            if_id_reg_flush = 1;
            update_BTB(if_id_out.PC, id_ex_in.target_address & ~1, op_jalr);
            pc_MUX_sel = adder_mod2;
        end
        if(btb_hit == 0)
        begin
            if_id_reg_flush = 1;
```

```
                update_BTB(if_id_out.PC, id_ex_in.target_address & ~1, op_jalr);
                pc_MUX_sel = adder_mod2;
        end
end

// GHR

if(id_ex_in.ctrl.opcode == op_br)
begin
        load_GHR = 1;
        GHR_in = id_ex_in.br_en;
end

// BHT
if(id_ex_in.ctrl.opcode == op_br)
begin
        BHT_windex = if_id_out.pc;
        load_BHT = 1;
        BHT_in = id_ex_in.br_en;
end

// local PHT
if(id_ex_in.ctrl.opcode == op_br)
begin
        //load_local_PHT = 1;
        local_PHT_windex = if_id_out.local_PHT_index;
        if(id_ex_in.br_en == 1)
                increment();
```

```
	else
		decrement();
end

// global PHT
if(id_ex_in.ctrl.opcode == op_br)
begin
	//load_global_PHT = 1;
	global_PHT_windex = if_id_out.global_PHT_index;
	if(id_ex_in.br_en == 1)
		increment();
	else
		decrement();
end

logic local_c;
logic global_c;
// selection PHT
if(id_ex_in.ctrl.opcode == op_br)
begin
	//load_select_PHT = 1;
	select_PHT_windex = if_id_out.pc;
	if(id_ex_in.br_en == if_id_out.local_pr)
		local_c = 1;
	if(id_ex_in.br_en == if_id_out.global_pr)
		global_c = 1;
	if(local_c != global_c)
	begin
```

```
            if(local_c == 1)
                  decrement();  // local predictor is RIGHT, global predictor is WRONG, and we let the number go
DOWN, so if the counter value is low, it means local is right a lot recently
            if(global_c == 1)
                  increment();
      end
end
```

Tournament Predictor:

If Selection PHT is an array of 2-bit, both-side-saturating counters, then the selection is made by using Architectural PC to index into the Selection PHT. If the counter value is 00 or 01, then use the prediction from Local Predictor. If the counter value is 10 or 11, then use the prediction from Global Predictor.

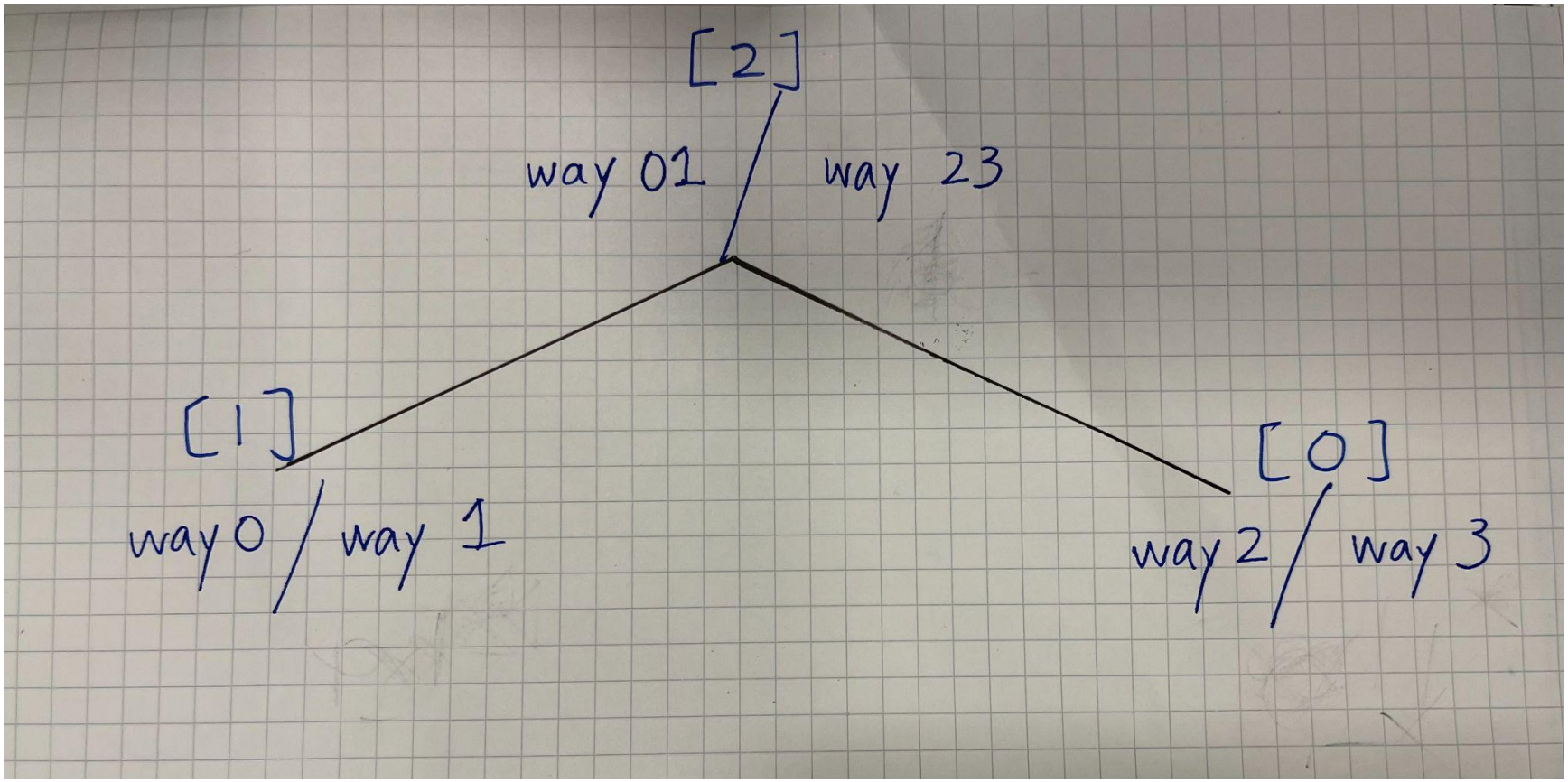
Local and Global branch predictor designs are copied from https://www.cs.utah.edu/~rajeev/cs7810/papers/mcfarling93.pdf (this is the paper linked from the lecture slides).

4-Way Set Associative Pseudo-LRU

(4-way set associative cache [2])

(This logic description is written for cache. But the same logic applies to BTB as well. But BTB does not write back.)

We employ a tree-based Pseudo-LRU scheme using 3 bits per set to maintain which cache line to replace. For N-way set associative, we use N-1 bits.




```
// Decide if we need to WB or not.
if(hit == 1'b1)
```

```
begin
    next_state = DEFAULT;
end
else if(v_array_0_dataout == 1'b0 || v_array_1_dataout == 1'b0 || v_array_2_dataout == 1'b0 || v_array_3_dataout
== 1'b0)
begin
    next_state = NO_WB_1;
end
else
begin
    if(LRU_array_dataout[2] == 1'b0)
    begin
        if(LRU_array_dataout[0] == 1'b0)
        begin
            if(d_array_3_dataout == 1'b0)
                next_state = NO_WB_1;
            else
                next_state = WRITE_BACK;
        end
        else
        begin
            if(d_array_2_dataout == 1'b0)
                next_state = NO_WB_1;
            else
                next_state = WRITE_BACK;
        end
    end
    else
```

```
    begin
        if(LRU_array_dataout[1] == 1'b0)
        begin
            if(d_array_1_dataout == 1'b0)
                next_state = NO_WB_1;
            else
                next_state = WRITE_BACK;
        end
        else
        begin
            if(d_array_0_dataout == 1'b0)
                next_state = NO_WB_1;
            else
                next_state = WRITE_BACK;
        end
    end
end

// At WRITE_BACK state, just look at LRU to decide which way to write back
// since we already know (no free cacheline) + (LRU is dirty)


// At NO_WB_2, decide which way to allocate.

if(v_array_0_dataout == 1'b0)
begin
    // Alloc way 0
end
```

```
else if(v_array_1_dataout == 1'b0)
begin
    // Alloc way 1
end
else if(v_array_2_dataout == 1'b0)
begin
    // Alloc way 2
end
else if(v_array_3_dataout == 1'b0)
begin
    // Alloc way 3
end
else
begin
    if(LRU_array_dataout[2] == 1'b0)
    begin
        if(LRU_array_dataout[0] == 1'b0)
        begin
            // Alloc way 3
        end
        else
        begin
            // Alloc way 2
        end
    end
    else
    begin
        if(LRU_array_dataout[1] == 1'b0)
```

```
        begin
            // Alloc way 1
        end
        else
        begin
            // Alloc way 0
        end
    end
end


// Update LRU array when there is a hit.
if(way_0_hit)
begin
    LRU_array_datain[2:0] = {1'b0, 1'b0, LRU_array_dataout[0]};
end
else if(way_1_hit)
begin
    LRU_array_datain[2:0] = {1'b0, 1'b1, LRU_array_dataout[0]};
end
else if(way_2_hit)
begin
    LRU_array_datain[2:0] = {1'b1, LRU_array_dataout[1], 1'b0};
end
else if(way_3_hit)
begin
    LRU_array_datain[2:0] = {1'b1, LRU_array_dataout[1], 1'b1};
end
```

L2+ Cache System

(L2+ cache system [2])

The L2 Cache will be a unified cache sitting between the Arbiter and the Cache Line Adaptor. It will be ported from our MP3 cache but modified to be 4-way set associative. We will also parameterize the number of sets.

Multiplier

(RISC-V M Extension: A basic multiplier design is worth [3] while an advanced multiplier is worth [5])

Wallace Tree Multiplier:

Sum partial products using a selection of full and half adders until only two values are left, and then sum those up, giving the result.

We will also need to add additional decoding logic in the ID stage to provide the proper control word for multiple instructions.

The multiplier will need to be integrated with the ALU in the execute stage. The Wallace Tree Multiplier should be done in 1 cycle so no additional stalling control is required (all ALU ops still take one cycle).