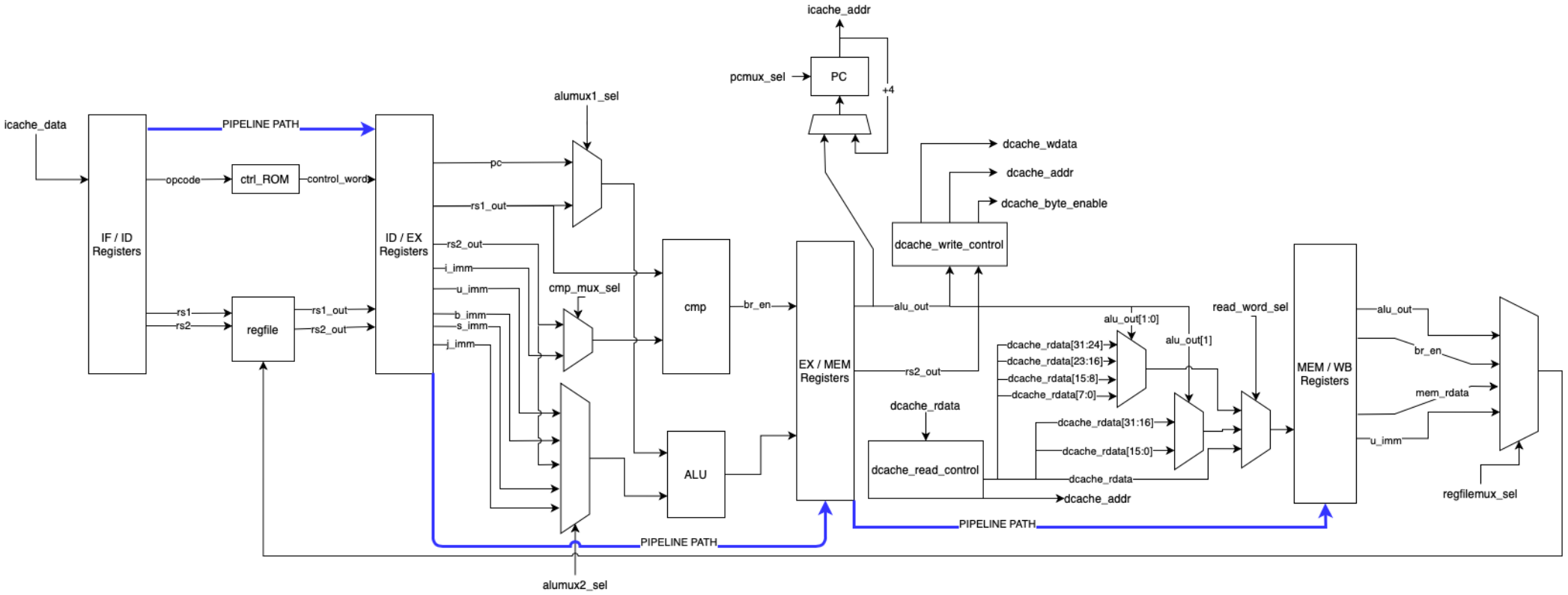
ECE 411
Team NaN

MP4: Basic Pipeline Datapath Design

Naveen Nathan (nnathan2)
Neo Vasudeva (neov2)
Anchit Rao (anchitr2)

March 23, 2021


icache_addr
pcmux_sel
PC
+4
alumux1_sel
icache_data
PIPELINE PATH
IF / ID Registers
opcode
ctrl_ROM
control_word
ID / EX Registers
pc
rs1_out
rs2_out
i_imm
u_imm
b_imm
s_imm
j_imm
cmp_mux_sel
cmp
br_en
rs1
rs2
regfile
rs1_out
rs2_out
ALU
EX / MEM Registers
alu_out
rs2_out
dcache_wdata
dcache_addr
dcache_byte_enable
dcache_write_control
dcache_rdata
dcache_read_control
dcache_addr
alu_out[1:0]
alu_out[1]
dcache_rdata[31:24]
dcache_rdata[23:16]
dcache_rdata[15:8]
dcache_rdata[7:0]
dcache_rdata[31:16]
dcache_rdata[15:0]
dcache_rdata
read_word_sel
MEM / WB Registers
alu_out
br_en
mem_rdata
u_imm
regfilemux_sel
PIPELINE PATH
PIPELINE PATH
alumux2_sel

Pipeline Control:

Default Control Signals:
pcmux_sel = pcmux::pc_plus4;
alumux1_sel = alumux::rs1_out;
alumux2_sel = alumux::i_imm;
regfilemux_sel = regfilemux::alu_out;
cmpmux_sel = cmpmux::rs2_out;
aluop = rv32i_types::alu_add;
cmpop = rv32i_types::beq;
dcache_read = 1'b0;
dcache_write = 1'b0;
dcache_byte_enable = 4'b1111;
load_regfile = 1'b0;
read_word_sel = 2'b00;

Opcode	Control Signals (in Control Word)

imm	`// slti`
	`if (funct3 == rv32i_types::slt) begin`
	`  load_regfile = 1'b1;`
	`  cmpop = rv32i_types::blt;`
	`  regfilemux_sel = regfilemux::br_en;`
	`  cmpmux_sel = cmpmux::i_imm;`
	`end`
	
	`// sltiu`
	`else if (funct3 == rv32i_types::sltu) begin`
	`  load_regfile = 1'b1;`
	`  cmpop = rv32i_types::bltu;`
	`  regfilemux_sel = regfilemux::br_en;`
	`  cmpmux_sel = cmpmux::i_imm;`
	`end`
	
	`// sr (srai, srli)`
	`else if (funct3 == rv32i_types::sr) begin`
	`  load_regfile = 1'b1;`
	`  regfilemux_sel = regfilemux::alu_out;`
	
	`  // srai/srli`
	`  if (funct7 == 7'b0100000)`
	`  aluop = rv32i_types::alu_sra;`
	`  else`
	`  aluop = rv32i_types::alu_srl;`
	`end`
	
	`// other immediate instructions`
	`else begin`

	load_regfile = 1'b1;
	aluop = alu_ops'(funct3);
	end

lui	load_regfile = 1'b1; regfilemux_sel = regfilemux::u_imm;

reg_arith	// other control signals
	load_regfile = 1'b1;
	alumux1_sel = alumux::rs1_out;
	alumux2_sel = alumux::rs2_out;
	regfilemux_sel = regfilemux::alu_out;
	
	// add/sub
	if (funct3 == rv32i_types::add) begin
	if (funct7 == 7'b0000000)
	aluop = rv32i_types::alu_add;
	else
	aluop = rv32i_types::alu_sub;
	end
	
	// srl/sra
	else if (funct3 == rv32i_types::sr) begin
	if (funct7 == 7'b0000000)
	aluop = rv32i_types::alu_srl;
	else
	aluop = rv32i_types::alu_sra;
	end
	
	// sll
	else if (funct3 == rv32i_types::sll)
	aluop = rv32i_types::alu_sll;
	
	// slt
	else if (funct3 == rv32i_types::slt) begin
	cmpop = rv32i_types::blt;
	regfilemux_sel = regfilemux::br_en;
	cmpmux_sel = cmpmux::rs2_out;
	end
	
	// sltu
	else if (funct3 == rv32i_types::sltu) begin
	cmpop = rv32i_types::bltu;
	regfilemux_sel = regfilemux::br_en;
	cmpmux_sel = cmpmux::rs2_out;
	end
	
	// xor
	else if (funct3 == rv32i_types::axor)
	aluop = rv32i_types::alu_xor;

	`// or`
	`else if (funct3 == rv32i_types::aor)`
	`  aluop = rv32i_types::alu_or;`
	
	`// and`
	`else if (funct3 == rv32i_types::aand)`
	`  aluop = rv32i_types::alu_and;`

auipc	`alumux1_sel = alumux::pc_out;` `alumux2_sel = alumux::u_imm;`

br	`pcmux_sel = br_en;`
	`alumux1_sel = alumux::pc_out;`
	`alumux2_sel = alumux::b_imm;`
	`cmpop = funct3;`

load	`dcache_read = 1'b1;`
	`load_regfile = 1'b1;`
	`unique case (funct3)`
	`  rv32i_types::lw: begin`
	`  regfilemux_sel = regfilemux::lw;`
	`  read_word_sel = 2'b00;`
	`  end`
	`  rv32i_types::lb: begin`
	`  regfilemux_sel = regfilemux::lb;`
	`  read_word_sel = 2'b01;`
	`  end`
	`  rv32i_types::lbu: begin`
	`  regfilemux_sel = regfilemux::lbu;`
	`  read_word_sel = 2'b01;`
	`  end`
	`  rv32i_types::lh: begin`
	`  regfilemux_sel = regfilemux::lh;`
	`  read_word_sel = 2'b10;`
	`  end`
	`  rv32i_types::lhu: begin`
	`  regfilemux_sel = regfilemux::lhu;`
	`  read_word_sel = 2'b10;`
	`  end`
	`  default: ;`
	`endcase`

store	`alumux2_sel = alumux::s_imm;`
	`dcache_write = 1'b1;`
	`unique case (funct3)`
	`  rv32i_types::sw: begin`
	`  dcache_byte_enable = 4'b1111;`
	`  end`
	`  rv32i_types::sb: begin`
	`  dcache_byte_enable = 4'b0011 <<`
	`{mem_address_unaligned[1], 1'b0};`

	end rv32i_types::sh: begin dcache_byte_enable = 4'b0001 << mem_address_unaligned[1:0]; end default: ; endcase

jal	pcmux_sel = pcmux::alu_mod2;
	regfilemux_sel = regfilemux::pc_plus4;
	alumux1_sel = alumux::pc_out;
	alumux2_sel = alumux::j_imm;

jalr	pcmux_sel = pcmux::alu_mod2; regfilemux_sel = regfilemux::pc_plus4;

Control Word Struct Definition:

```
struct packed {
        logic alumux1_sel;
        logic [2:0] alumux2_sel;
        logic [3:0] regfilemux_sel;
        logic cmpmux_sel;
        logic [2:0] cmpop;
        logic [2:0] aluop;
        logic load_regfile;
        logic [1:0] read_word_sel;
        logic dcache_read;
        logic dcache_write;
        logic [3:0] dcache_byte_enable;
} ctrl_word_t
```

IF_ID Register Definition:

```
struct packed {
        logic [31:0] instruction;
        logic [31:0] PC;
} IF_ID
```

ID_EX Register Definition:

```
struct packed {
        logic [31:0] instruction;
        logic [31:0] PC;
        ctrl_word_t control_word;
} ID_EX
```

EX_MEM Register Definition:

```
struct packed {
        logic [31:0] alu_out;
        logic [31:0] rs2_out;
        logic [31:0] rd;
        logic br_en;
        logic [31:0] u_imm;
        ctrl_word_t control_word;
} EX_MEM
```

MEM_WB Register Definition:

```
struct packed {
        logic [31:0] alu_out;
        logic [31:0] rd;
        logic br_en;
        logic [31:0] u_imm;
        ctrl_word_t control_word;
        logic [31:0] mdrreg_out;
} MEM_WB
```