無料でLSIが焼けるらしいので、皆でRISC-Vパイプラインプロセッサを作った話

sussikitigai8

開発の背景

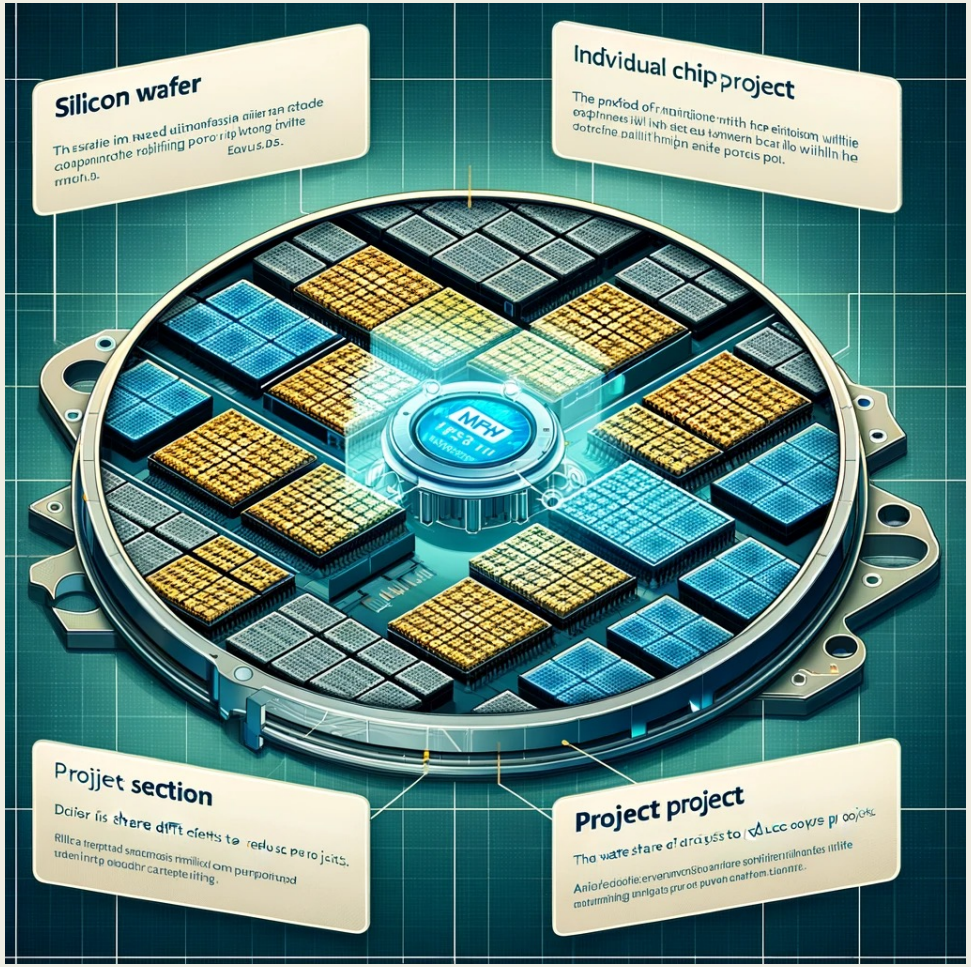
- 発端は、高専の同期との輪読会
- 当時読んでいた本は、ディジタル回路設計とコンピュータアーキテクチャ [RISC-V版]
- 第七章ではプロセッサを設計する段に入るので、実際にHDLで書いてシミュレーションした。

このプロセッサを
実際にチップとして
製造できればいいのになあ。

開発の背景

Open MPWに
作ったプロセッサ
を提出しよう！

無料でLSIを焼く方法



■ MPW(Multi Project Wafer)

・一枚のシリコンウエハ上に複数のプロジェクト(チップ)を配置

■ OpenMPW

・オープンソースの半導体設計を対象にEfabless社が開催しているチップ製造シャトル

・製造費・輸送費共に無料

・Googleが後援で資金提供

・応募数が多いので、どのチップが焼かれるかは抽選で決まる

提出するまでの流れ

HDLでRTL(Register Transfer level)コードを
記述し、シミュレーションする

OpenLANEを用いて, RTLコードをGDSIIに変換

提出するまでの流れ

👉 HDLでRTL(Register Transfer level)コードを
記述し、シミュレーションする

OpenLANEを用いて, RTLコードをGDSIIに変換

RISC-V 32bit命令セットアーキテクチャ

- 32bitの固定長命令によるISA(Instruction Set Architecture)
- x0~x31の32個のレジスタを持つ。
- 命令フォーマットは6個の形式に分かれる。
- プロセッサは、下位7bitのオペコードでどの形式かを判断する。

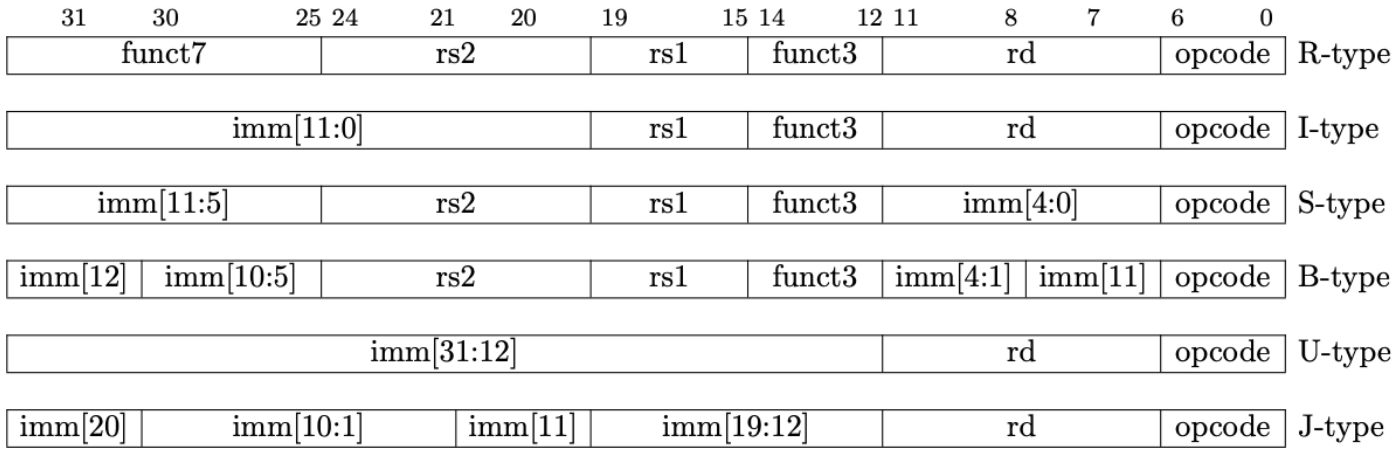
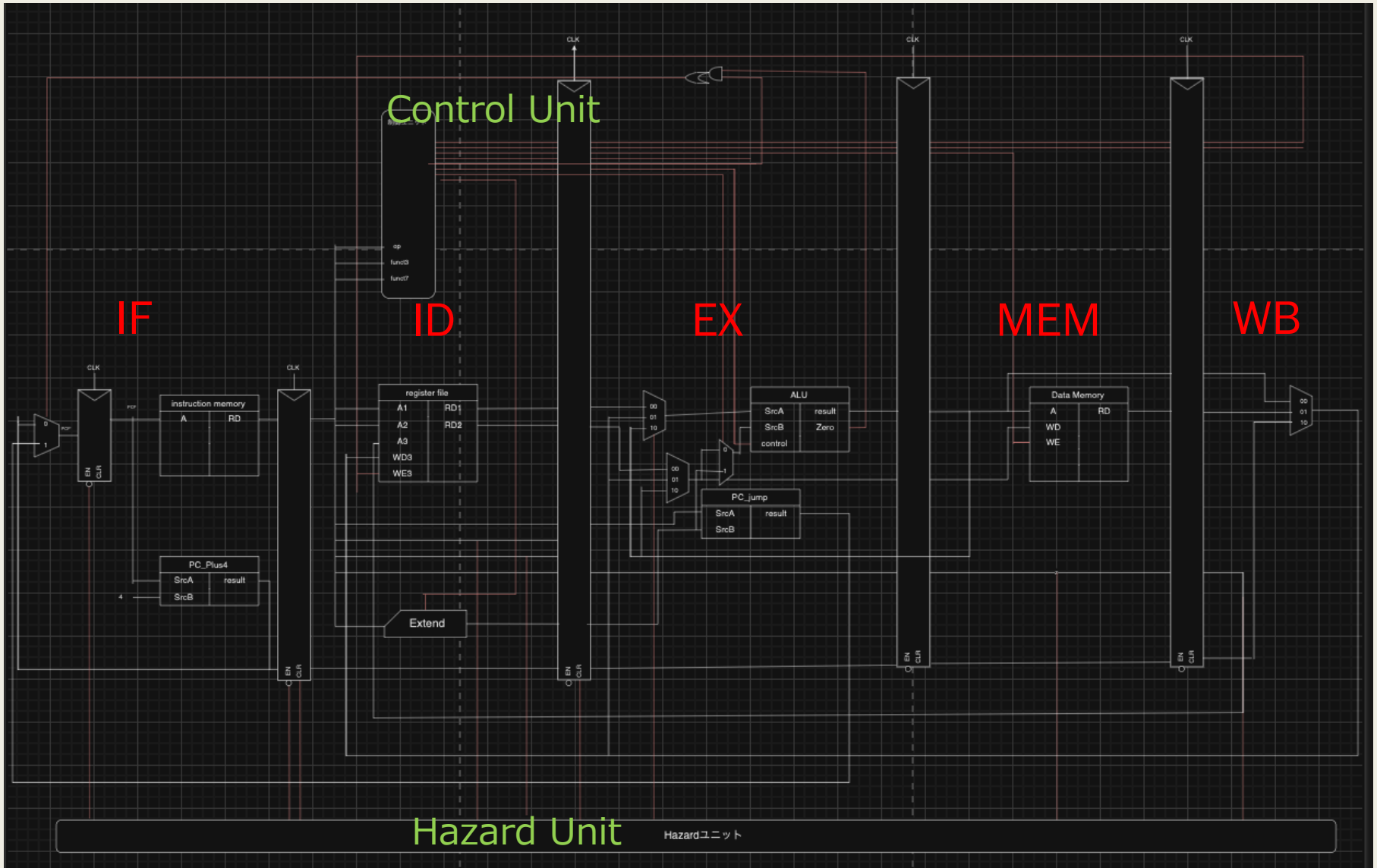


Figure 2.3: RISC-V base instruction formats showing immediate variants.

引用：The RISC-V Instruction Set Manual https://five-embeddev.com/riscv-isa-manual/latest/rv32.html

パイプラインプロセッサの回路図



パイプラインレジスタ

- 各ステージ間には、次のステージに値を渡すためのレジスタが配置されている。
- クロックが立ち上がるごとに次のステージにデータを渡す。
- always_ff文は、@マーク以下の信号の変化に同期して動作する回路を記述する。
- always_ff @(posedge clk)は、クロックの立ち上がりに同期して動作するという意味
- IF/IDレジスタでは、フェッチした命令(instr_f)とプログラムカウンタ(pc_f, pc_plus_4_f)をIDステージに受け渡す。

IF/IDパイプラインレジスタのHDL記述

```
always_ff @(posedge clk) begin
    if (rst | flush_d) begin
        instr_d <= 0;
        pc_d <= 0;
        pc_plus_4_d <= 0;
    end
    else begin
        if (!stall_d && valid && !stall_read) begin
            instr_d <= instr_f;
            pc_d <= pc_f;
            pc_plus_4_d <= pc_plus_4_f;
        end
    end
end
```

「addi x1, x0, 1」が実行されるまで

addi

add immediate

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[11:0]			rs1	000	rd	00100	11

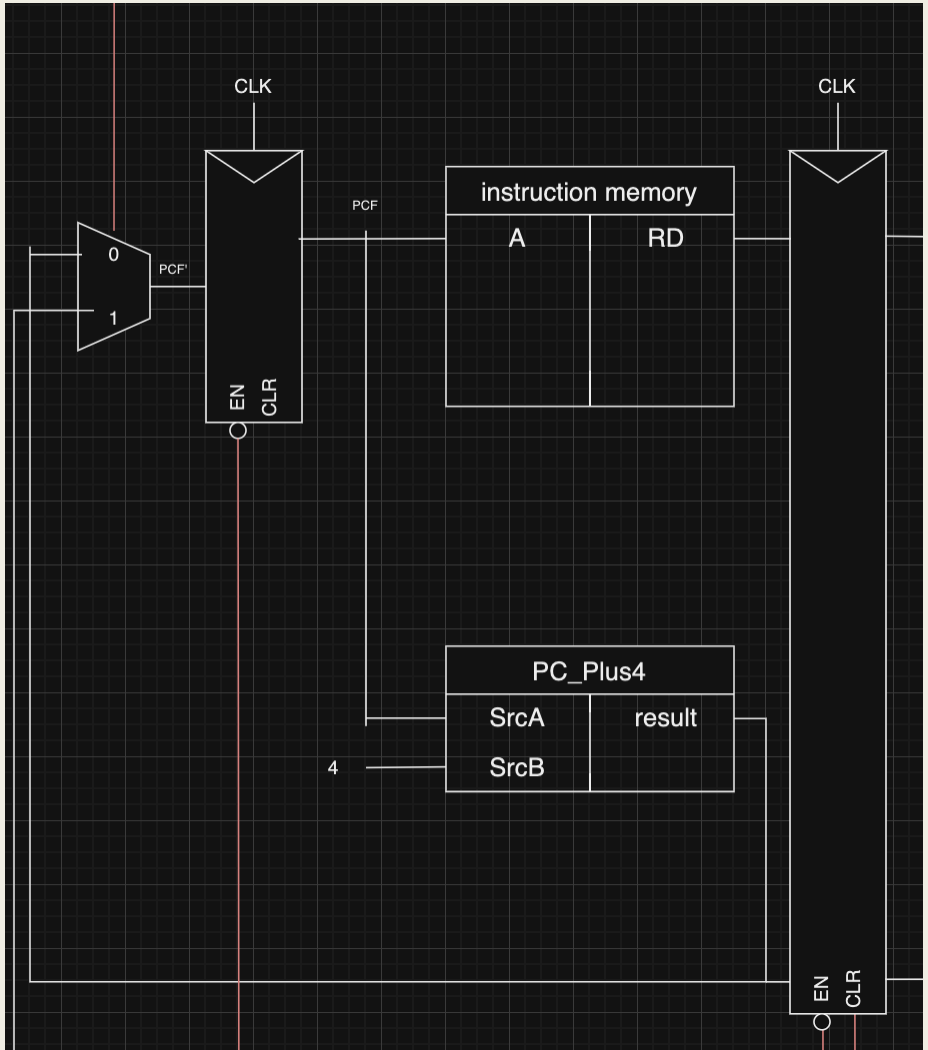
Format addi rd,rs1,imm

Description Adds the sign-extended 12-bit immediate to register rs1. Arithmetic overflow is ignored and the result is simply the low XLEN bits of the result. ADDI rd, rs1, 0 is used to implement the MV rd, rs1 assembler pseudo-instruction.

Implementation x[rd] = x[rs1] + sext(immediate)

引用： rv32i, RV64I Instructions — riscv-isa-pages documentation
https://msyksphinz-self.github.io/riscv-isadoc/html/rvi.html

1. IF(Instruction Fetch)



- マルチプレクサでプログラムカウンタを選択
- Instruction memoryがプログラムカウンタに対応する32bit命令を出力
- InstructionやPCはパイプラインレジスタに入る

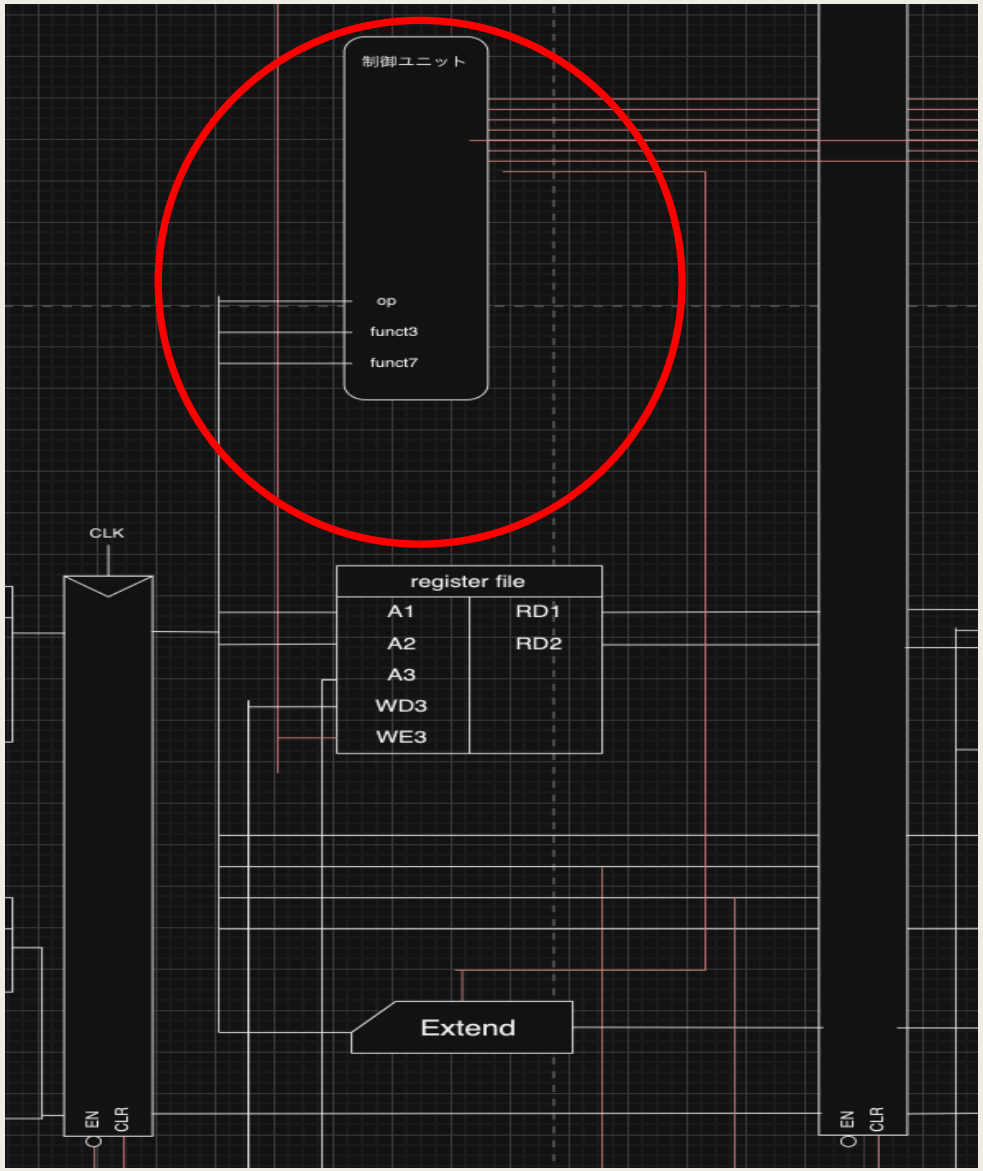
```
always_ff @(posedge clk) begin
    if (rst | flush_d) begin
        instr_d <= 0;
        pc_d <= 0;
        pc_plus_4_d <= 0;
    end
    else begin
        if (!stall_d && valid && !stall_read) begin
            instr_d <= instr_f;
            pc_d <= pc_f;
            pc_plus_4_d <= pc_plus_4_f;
        end
    end
end
```

2. ID(Instruction Decode, Register read)

命令を分割して、制御ユニットに入力

```
assign op      = instr_d[6:0];
assign funct7  = instr_d[31:25];
assign funct3_d = instr_d[14:12];
```

```
Decoder decoder(
    .op(op),
    .funct3(funct3_d),
    .funct7(funct7),

    .reg_write(reg_write_d),
    .result_src(result_src_d),
    .mem_write(mem_write_d),
    .jump(jump_d),
    .branch(branch_d),
    .alu_control(alu_control_d),
    .alu_src(alu_src_d),
    .imm_src(imm_src_d),

    .pc_alu_src(pc_alu_src_d),
    .src_a_src(src_a_src_d)
);
```



制御ユニットでは、入力信号を元にその命令に対応する制御信号を出力
以降、制御信号はパイプラインレジスタによって各ステージに伝播していく

```
always_comb begin
    case (op)

        (中略)

        //addi,slli,slti,sltiu,xori,srli,srai,ori,andi
        7'b0010011 : begin
            reg_write = 1; //レジスタの書き込みを行うか
            imm_src = 3‘b00; //即値のフォーマットを選択
            alu_src = 1; // ALUの入力ソースを選択するマルチプレクサの制御信号
            mem_write = 0; //メモリへの書き込みを行うか
            result_src = 0;   // WBステージのresult信号を選択するマルチプレクサの制御信号
            alu_op = 2‘b10; // ALUの演算方法を選択するための一要素
            branch = 1’b0; // branch命令かどうか
            jump = 1‘b0;   // jump命令かどうか
            pc_alu_src = 1’b0; // プログラムカウンタを計算するALUに関する
            src_a_src = 2’b1; // ALUのsrcaに関する制御信号

        (中略)

    end
```

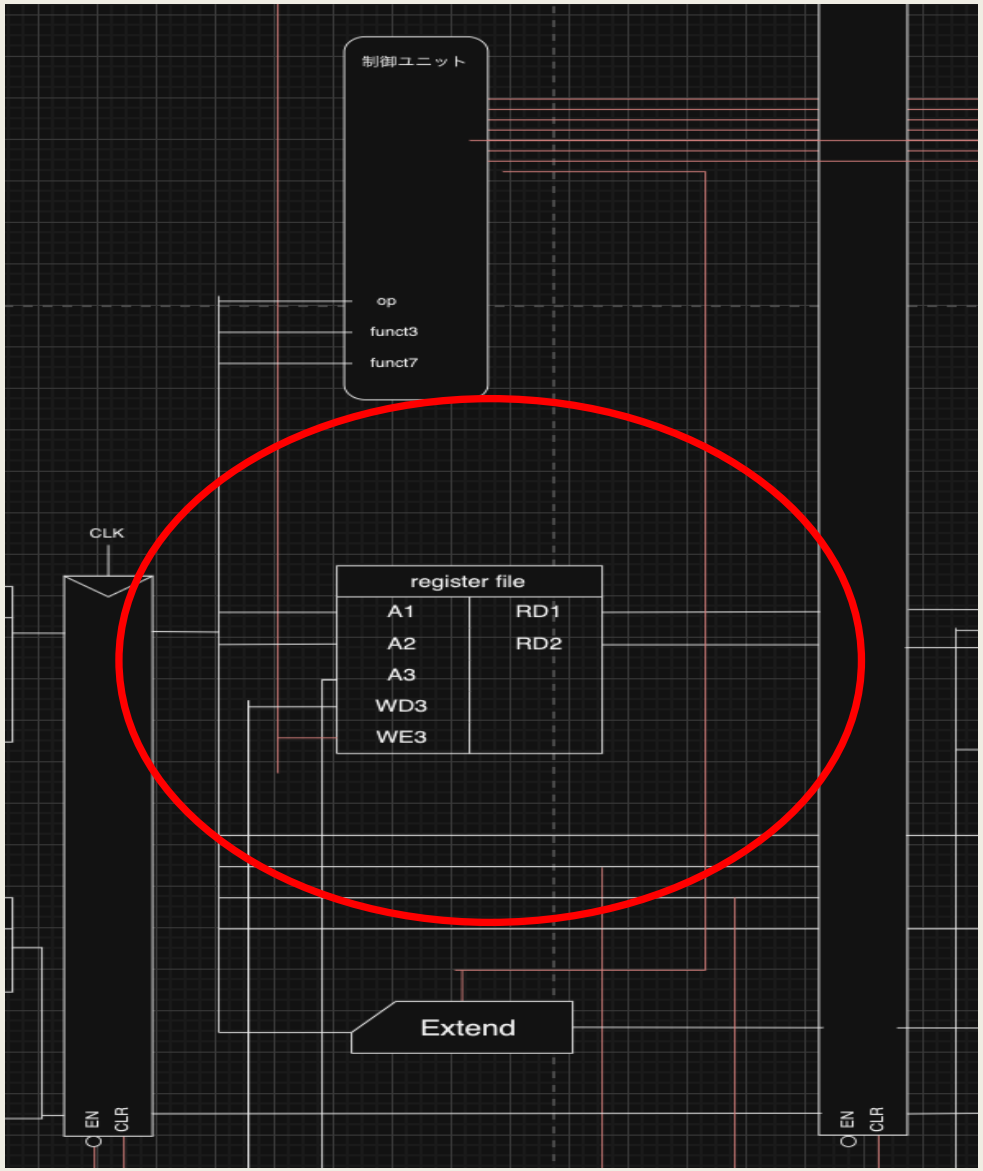
2. ID(Instruction Decode, Register read)

命令を分割してレジスタファイルに入力

```
Regfile reg_file(
    .clk(clk),
    .rst(rst),

    .addr1(instr_d[19: 15]),
    .rd1(rd1_d),
    .addr2(instr_d[24: 20]),
    .rd2(rd2_d),

    .addr3(rd_w),
    .wd3(wd3_w),
    .we3(reg_write_w)
);
```

レジスタファイルでは、入力された
addressに対応するレジスタの値を出力

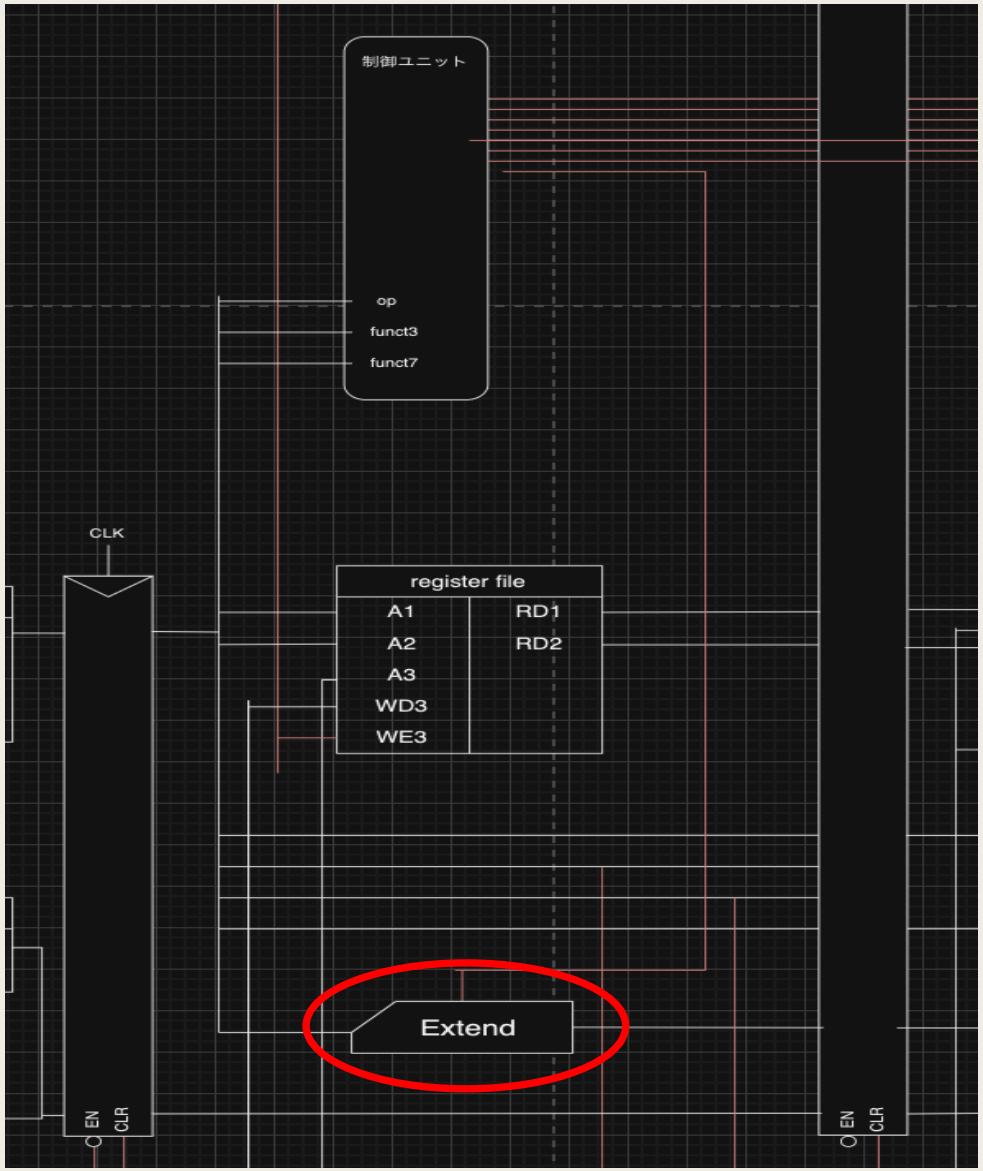
```
always_comb begin
    // rd1にaddr1の中身を出力
    rd1 = regfile[addr1];

    // rd2にaddr2の中身を出力
    rd2 = regfile[addr2];
end
```



2. ID(Instruction Decode, Register read)

命令から即値を読み込む

```
function [31: 0] extend(input [2:0] imm_src, input [31:0] instr);
	case(imm_src)
		// I-Type
		3'b000: extend = 32'(signed'(instr[31 -: 12]));
		// S-Type
		3'b001: extend = 32'(signed'({instr[31 -: 7], instr[7 +:
		5]}));
		// B-Type
		3'b010: extend = 32'(signed'({instr[31], instr[7],
		instr[30:25], instr[11:8],1'b0}));
		// J-Type
		3'b011: extend = 32'(signed'({instr[31], instr[19:12],
		instr[20], instr[30:21], 1'b0}));
		//U-Type
		3'b100: extend = 32'(signed'(instr[31 : 12])) << 12;
		default: extend = 32'hdeadbeef;
	endcase
endfunction
```

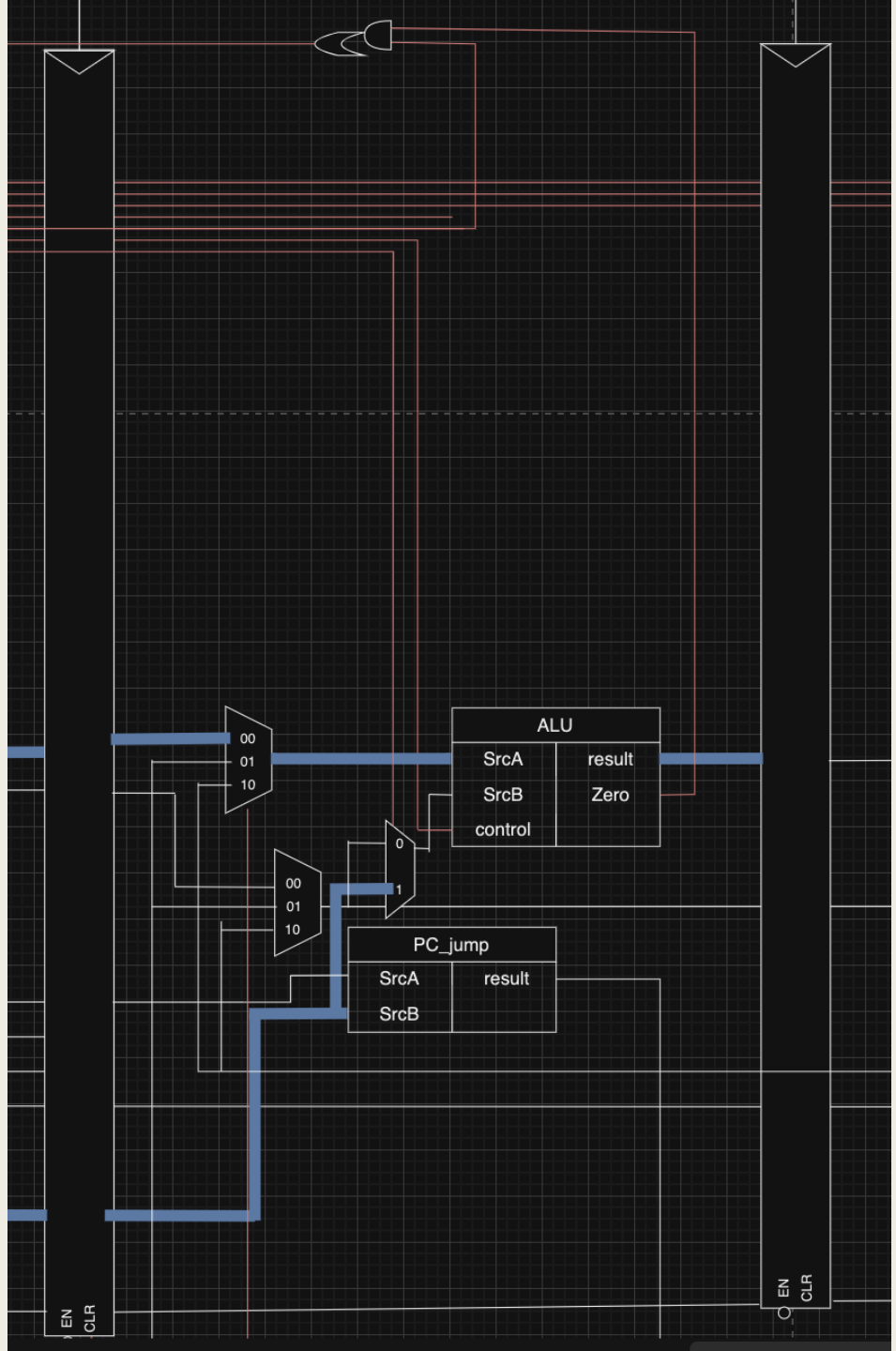


3. EX(Execution)

ALUの入力信号
(srcA,srcB,alu_control)を
マルチプレクサによって選択

今回は、
srcAにx0から読み込んだ値(0)
srcBに即値(x1)、
alu_controlは加算が選択される。

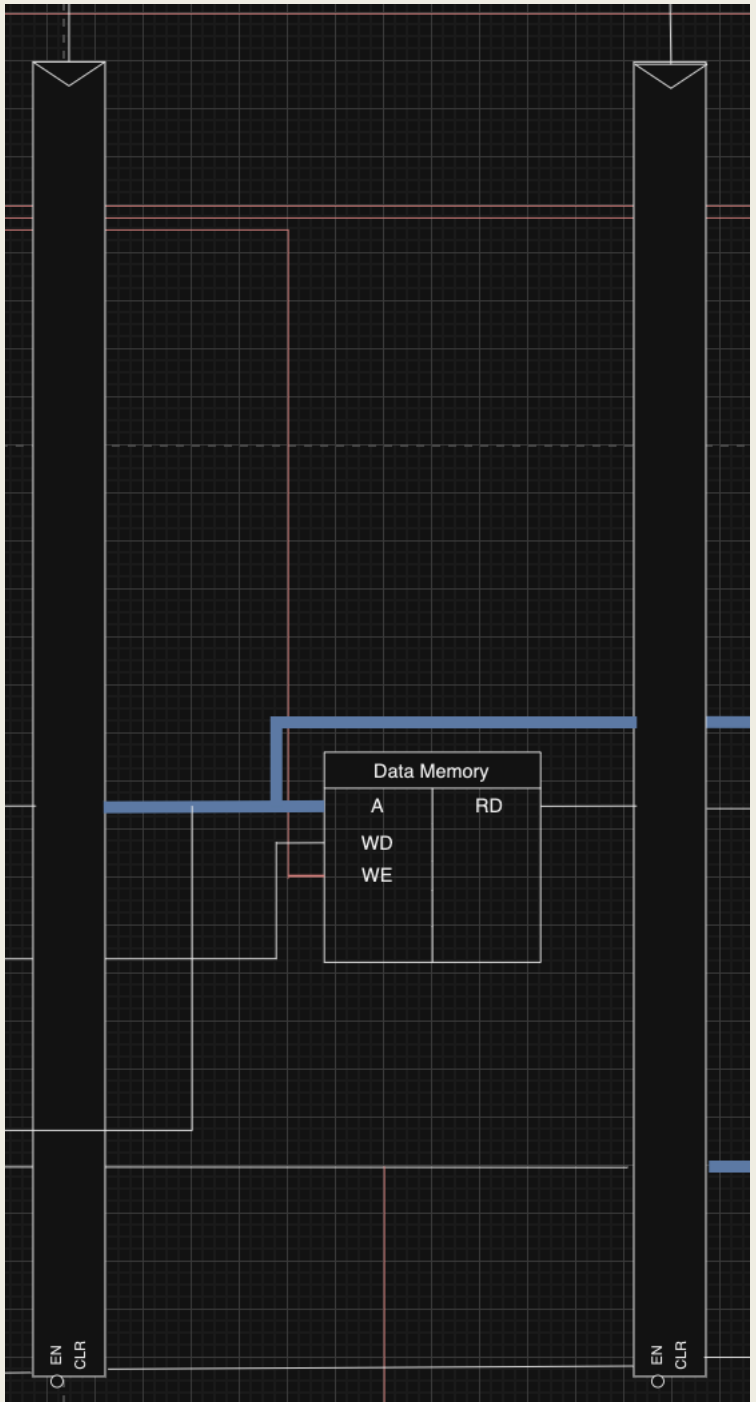
ALUはsrcA+srcBを出力



4. MEM(Memory access)

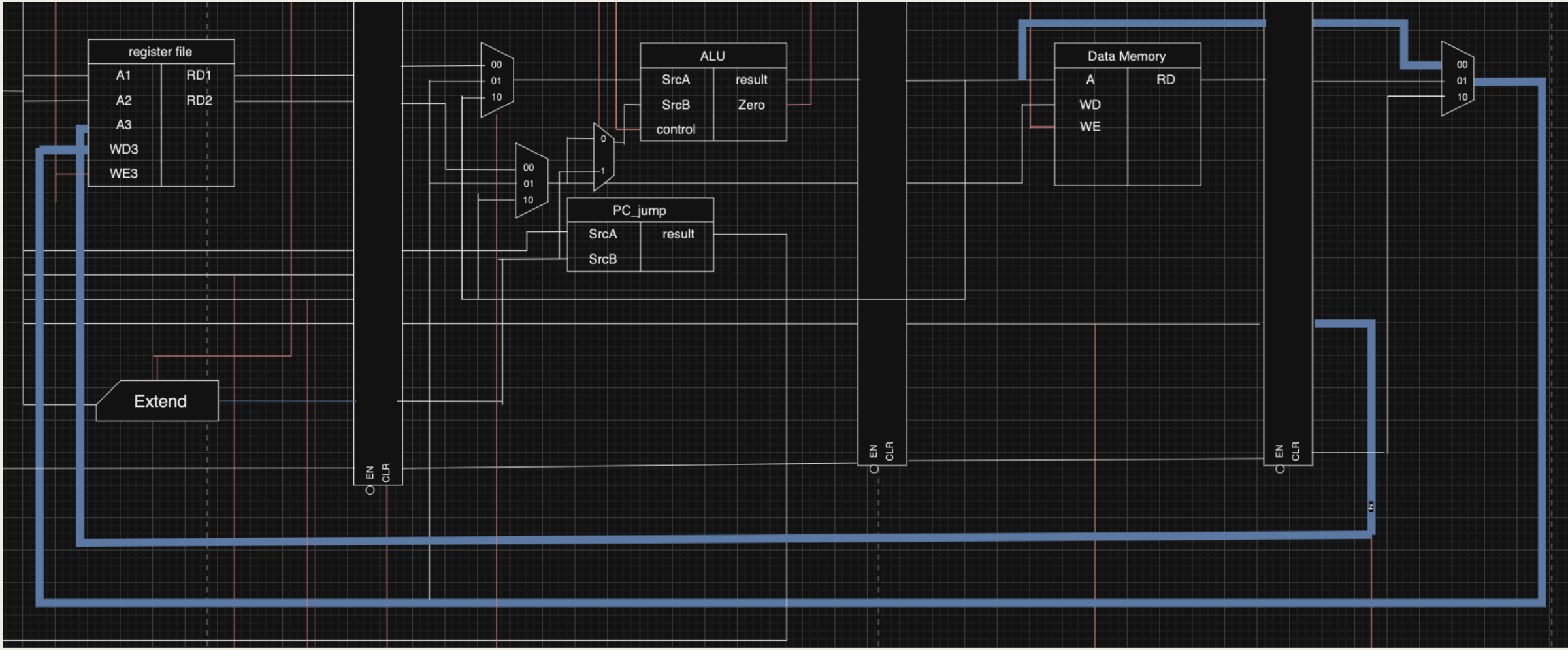
今回はデータメモリの読み書きを
行わない。

ALUの出力結果をMEM/WBパイプ
ラインレジスタに渡す。



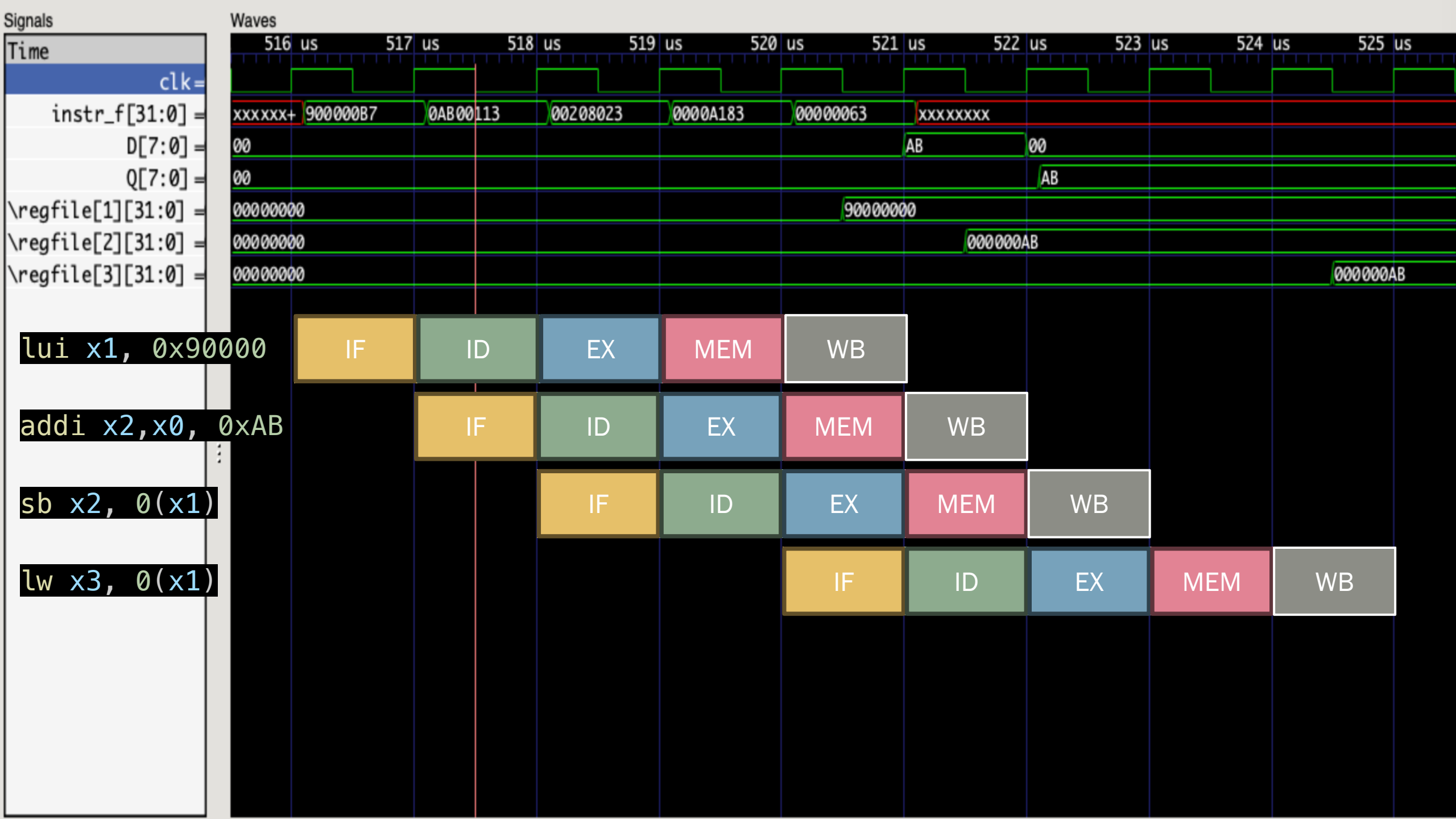
5.WB(Write Back)

レジスタにALUの出力結果とrd(register destination)アドレスを渡す。



シミュレーション波形の表示

```
.section .text
.global _start
_start:
# データメモリのベースアドレスをs0に設定
lui x1, 0x90000
# x2 = 0xAB
addi x2,x0, 0xAB
# x2の下位8bitをメモリに書き込む
sb x2, 0(x1)
# x2の下位8bitをメモリから読み込む
lw x3, 0(x1)

.end:
beq x0, x0, .end
```


Signals
Waves
Time
clk
instr_f[31:0]
D[7:0]
Q[7:0]
\regfile[1][31:0]
\regfile[2][31:0]
\regfile[3][31:0]
516 us
517 us
518 us
519 us
520 us
521 us
522 us
523 us
524 us
525 us
xxxxxx+
900000B7
0AB00113
00208023
0000A183
00000063
xxxxxxxx
00
AB
00
00
AB
00000000
90000000
00000000
000000AB
00000000
000000AB
lui x1, 0x90000
addi x2,x0, 0xAB
sb x2, 0(x1)
lw x3, 0(x1)
IF
ID
EX
MEM
WB

今後の方針

- GPIOと割り込み命令を実装する
- OpenLANEでRTLからGDSIIを変換する工程を完了させる。
- 当初は12/11締め切りのOpenMPWに応募しようとしていたが間に合わなかった。年に数回開催しているらしいので、次回の開催までに完成させる。

参考資料

- ディジタル回路設計とコンピュータアーキテクチャ [RISC-V版] サラ・L・ハリス (著), デイビッド・ハリス (著), 天野英晴/鈴木 貢 (翻訳), 中條拓伯/永松礼夫 (翻訳)
- rv32i, RV64I Instructions — riscv-isa-pages documentation https://msyksphinz-self.github.io/riscv-isadoc/html/rvi.html
- The RISC-V Instruction Set Manual https://five-embeddev.com/riscv-isa-manual/latest/rv32.html