Automatic Generation of Processor Design from Instruction Set Architecture

WooCheol Jung, JunRyul Yang, and SangKyun Yun
Yonsei University, Mirae Campus
Wonju, KOREA

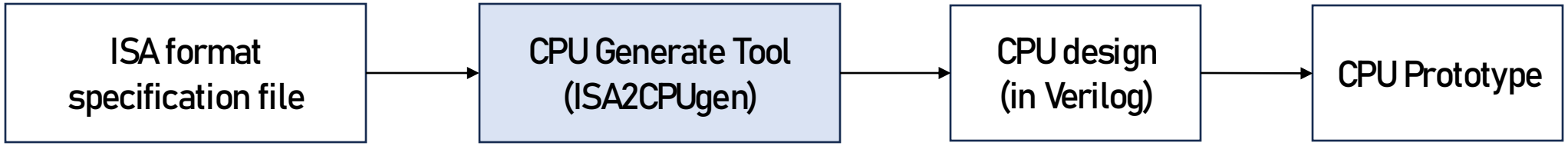
Why CPU Design?

- RISC processors like MIPS and RISC-V are widely used in education.
- Most follow simple architectures (single-cycle or 5-stage pipeline).
- FPGAs allow students to implement CPUs directly.
- Hands-on design improves understanding of computer architecture.
- Tools exist for simulator generation,
  → However, few generate real CPU hardware from ISA specs.

ISA to HDL for FPGA

- We developed a tool to generate HDL CPU designs from ISA specs.
- The design is ready for FPGA implementation.
- Works with different RISC ISAs using a shared architecture.
- Helps students quickly prototype and test CPUs on hardware.

ISA2CPUgen tool



RISC-family ISAs (without Flags)

- Single Cycle Design
- 5-stage Pipelined Design

Logic Simulation

FPGA Implementation

Instruction Set Architecture Specification File

```
// RISC-V
%register A=5 D=32 zero      // 2^5 32-bit registers (R[0] is zero register)

%instr_memory A=8 D=32 instr.hex  // 2^8x32 block RAM (init file: instr.hex)
wire [31:0] instr;  // instr : variable for instruction
PCinc = PC + 4;                   // next PC

%data_memory A=8 D=32 byteenable  // 2^8x32 block RAM, use byte-enable

%field
wire [6:0] opcode = instr[6:0];
wire [2:0] funct3 = instr[14:12];
wire [6:0] funct7 = instr[31:25];
wire [4:0] rd = instr[11:7];
wire [4:0] rs1 = instr[19:15];
wire [4:0] rs2 = instr[24:20];
wire [31:0] imm_i = {{20{instr[31]}}, instr[31:20]};
...

%operation(opcode, funct3, funct7)
ADD   (7'h33, 3'h0, 7'h00) : R[rd] = R[rs1] + R[rs2];
SUB   (7'h33, 3'h0, 7'h20) : R[rd] = R[rs1] - R[rs2];
ADDI  (7'h13, 3'h0) : R[rd] = R[rs1] + imm_i;
...
LB    (7'h03, 3'h0) : R[rd] = Mb[R[rs1] + imm_i];
LH    (7'h03, 3'h1) : R[rd] = Mh[R[rs1] + imm_i];
LW    (7'h03, 3'h2) : R[rd] = M[R[rs1] + imm_i];
LBU   (7'h03, 3'h4) : R[rd] = Mbu[R[rs1] + imm_i];
SW    (7'h23, 3'h2) : M[R[rs1] + imm_s] = R[rs2];
...
JALR  (7'h67, 3'h0) : R[rd] = PCinc, PC = R[rs1] + imm_i;
JAL   (7'h6F) : R[rd] = PCinc, PC = PC + imm_j;
BEQ   (7'h63, 3'h0) : if(R[rs1]==R[rs2]) PC = PC + imm_b;
...
```

Register file

Instruction memory

Data memory

Instruction Fields

Instruction Operations

ISA spec file – Register File and Memories

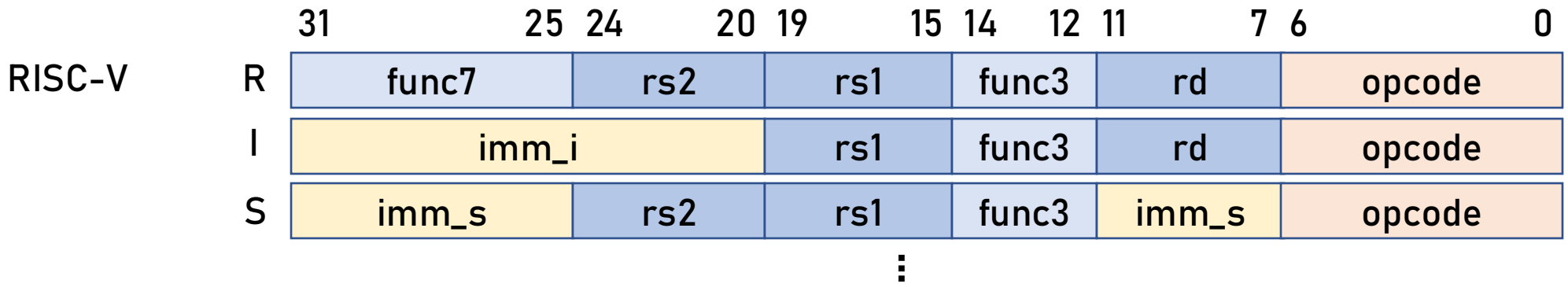
Define Address and Data size and Extra information

```
// RISC-V
%register A=5 D=32 zero    // 2^5 32-bit registers (R[0] is zero register)

%instr_memory A=8 D=32 instr.hex    // 2^8x32 block RAM (init file:instr.hex)
wire [31:0] instr;  // instr : variable for instruction
PCinc = PC + 4;     // next PC

%data_memory A=10 D=32 byteenable  // 2^10x32 block RAM, use byte-enable
```

Extra information

Register file	Instruction memory	Data memory
• Use of zero register	• program file name • instruction variable name • default Next PC variable	• Supports Byte-enable

ISA spec file - Instruction Fields




```
%field
wire [6:0] opcode = instr[6:0];
wire [2:0] funct3 = instr[14:12];
wire [6:0] funct7 = instr[31:25];
wire [4:0] rd = instr[11:7];
wire [4:0] rs1 = instr[19:15];
wire [4:0] rs2 = instr[24:20];
wire [31:0] imm_i = {{20{instr[31]}}, instr[31:20]};
…
```

- Instruction fields are defined in Verilog format

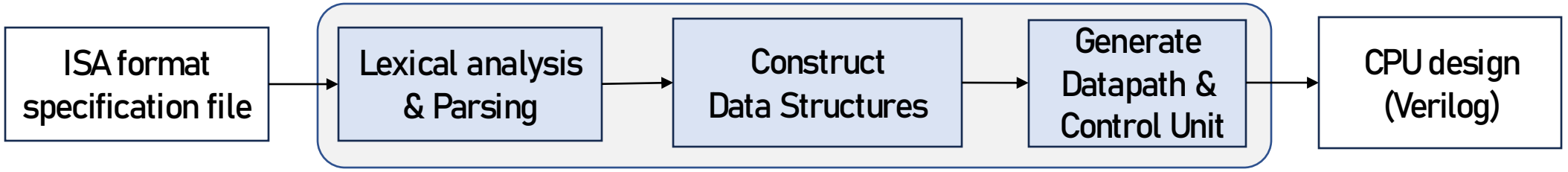
ISA spec file – Instruction Operations

Define Mnemonic, Opcode fields, Operation

```
%operation(opcode, funct3, funct7)
ADD   (7'h33, 3'h0, 7'h00) : R[rd] = R[rs1] + R[rs2];
SUB   (7'h33, 3'h0, 7'h20) : R[rd] = R[rs1] - R[rs2];
ADDI  (7'h13, 3'h0) : R[rd] = R[rs1] + imm_i;
LB    (7'h03, 3'h0) : R[rd] = Mb[R[rs1] + imm_i];
LH    (7'h03, 3'h1) : R[rd] = Mh[R[rs1] + imm_i];
LW    (7'h03, 3'h2) : R[rd] = M[R[rs1] + imm_i];
LBU   (7'h03, 3'h4) : R[rd] = Mbu[R[rs1] + imm_i];
SW    (7'h23, 3'h2) : M[R[rs1] + imm_s] = R[rs2];
JALR  (7'h67, 3'h0) : R[rd] = PCinc, PC = R[rs1] + imm_i;
JAL   (7'h6F)       : R[rd] = PCinc, PC = PC + imm_j;
BEQ   (7'h63, 3'h0) : if(R[rs1]==R[rs2]) PC = PC + imm_b;
...
```

R[ ]	Register
M..[ ]	Memory b : byte h : half bu, hu : unsigned
PC	Program Counter
If( )	Conditional Op

Organization of CPU generation tool



flex
bison

Instruction Operations

Resources
- Register file
- Instr. Mem
- Data Mem

Data input
Control signals
ALU
Instr/Data Memory
Register File
Forwarding Logic

Construct data structures

Data Structure for Instruction Operations

Opcode fields	destination	source1	source2	operator	class	extra
	• Reg addr	• Reg addr • PC	• Reg addr • Imm data		• Reg op • Mem op • Jump op • Condition • PC2Reg	• Mem op (ST/LD, S/U, size) • Jump op (Jump PC expr)

Data Structure for Resource

Register file

Instr. Memory

Data Memory

Construct data structures (MIPS)

```
ADDU  (6'h00, 6'h21) : R[rd] = R[rs] + R[rt];
```

```
ADDIU (6'h09) : R[rt] = R[rs] + SignExtImm;
```

```
LBU  (6'h24) : R[rt] = Mbu[R[rs] + SignExtImm];
```

```
BEQ  (6'h04) : if(R[rs] == R[rt]) PC = PCinc+BranchAddr;
```

Opcode fields	destination	source1	source2	operator	class	extra
6'h00, 6'h21	R[rd]	R[rs]	R[rt]	+	RegOp	
6'h09	R[rt]	R[rs]	SignExtImm	+	RegOp	
6'h24	R[rt]	R[rs]	SignExtImm	+	MemOp	LD U Byte
6'h04		R[rs]	R[rt]	==	JumpOp Condition	PCinc+BranchAddr

Generate Data Path and Control Signals

- Data structures → Data path, Control signals

- Data Path Architecture
  - Single Cycle Design
  - 5-stage Pipelined Design

- Need a generalized datapath architecture that can be used to implement many RISC-family ISAs.

RISC-V Architecture

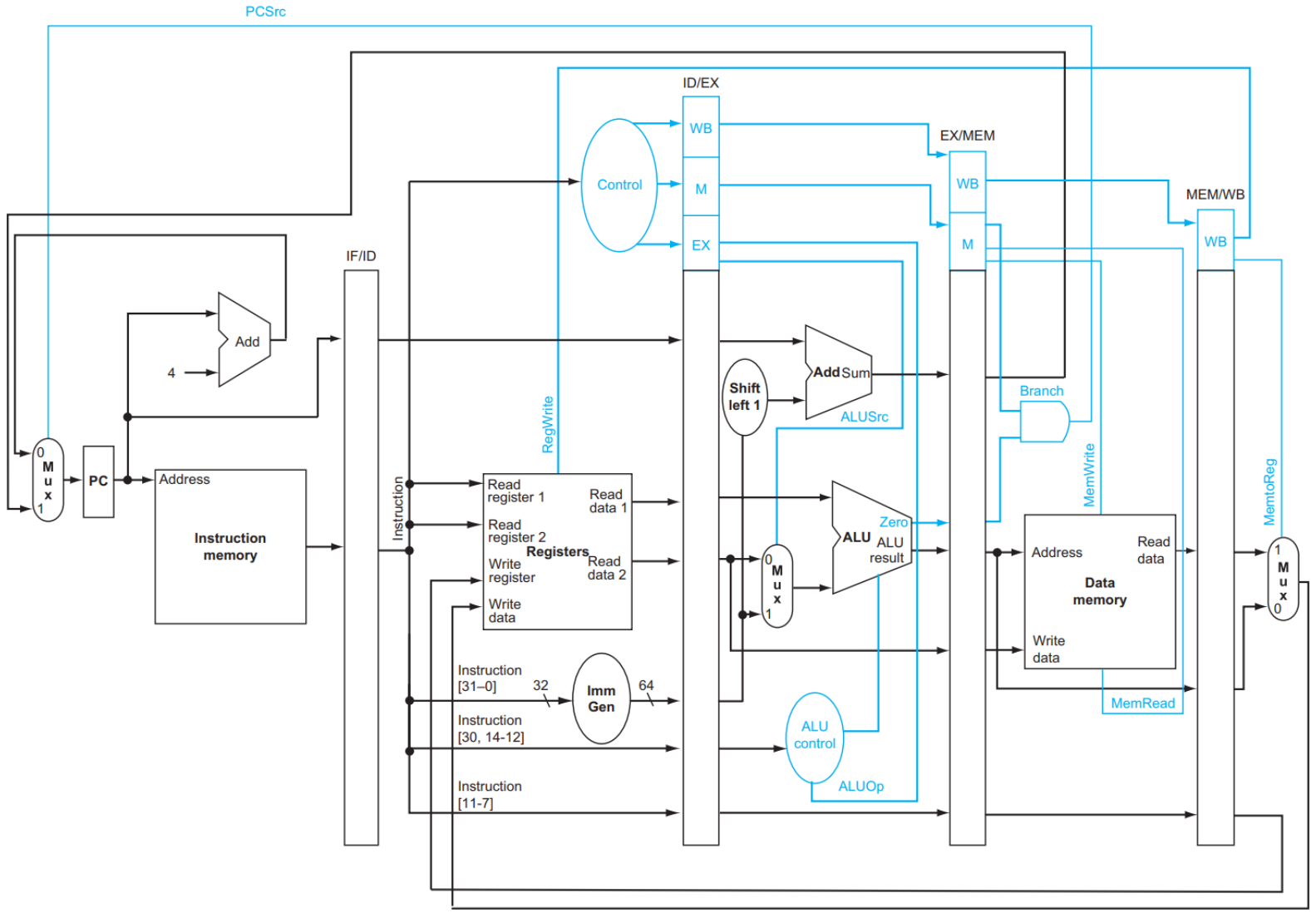


Image source: Patterson, David A., and John L. Hennessy. Computer organization and design RISC-V edition: the hardware software interface. Morgan kaufmann, 2018.

MIPS Architecture

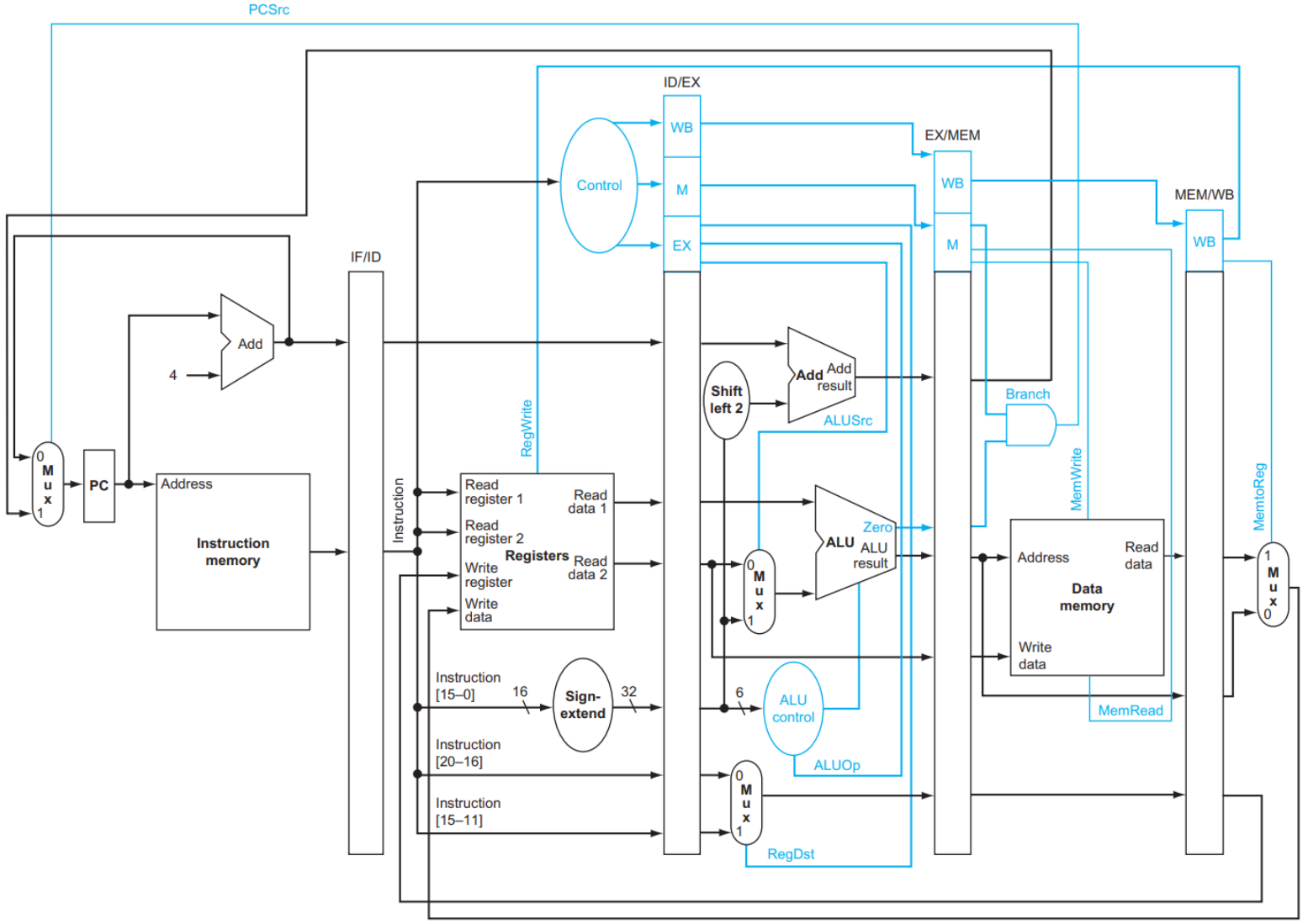


Image source: Patterson, David A., and John L. Hennessy. Computer organization and design fifth edition: the hardware software interface. Morgan kaufmann, 2014.

Beta Architecture

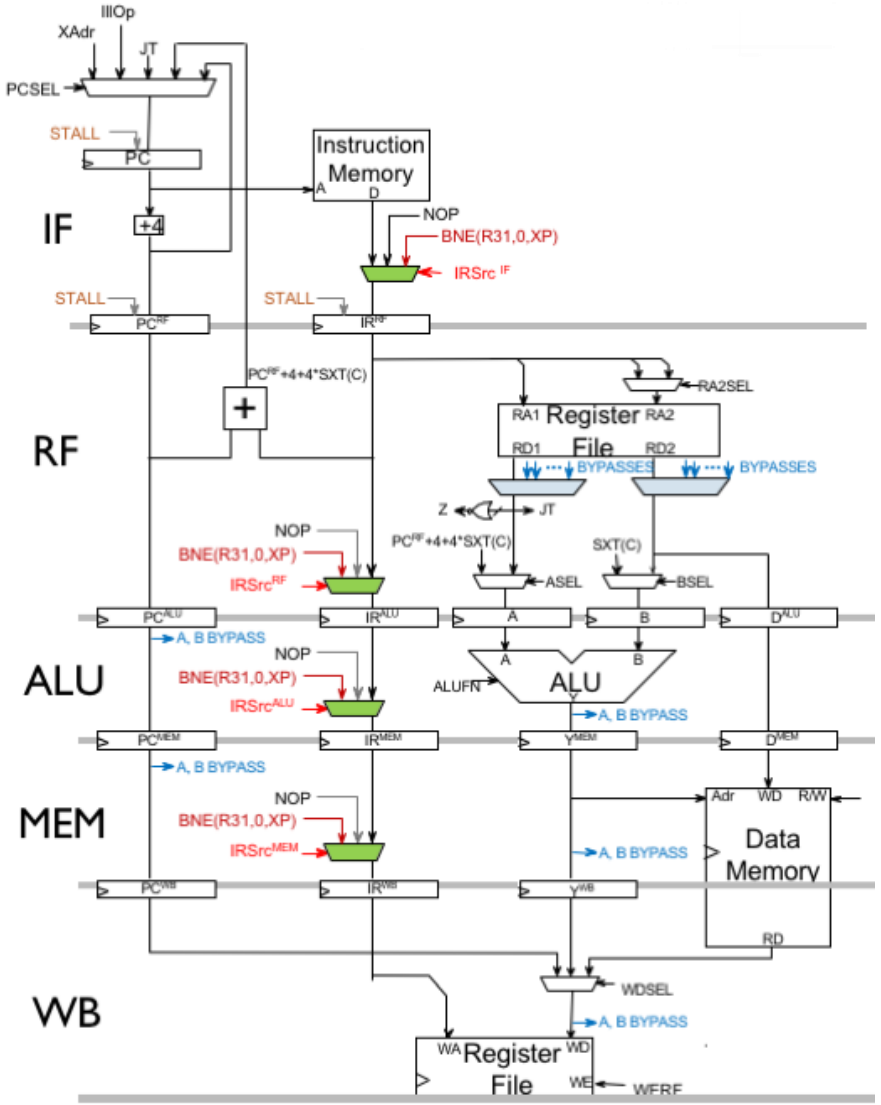
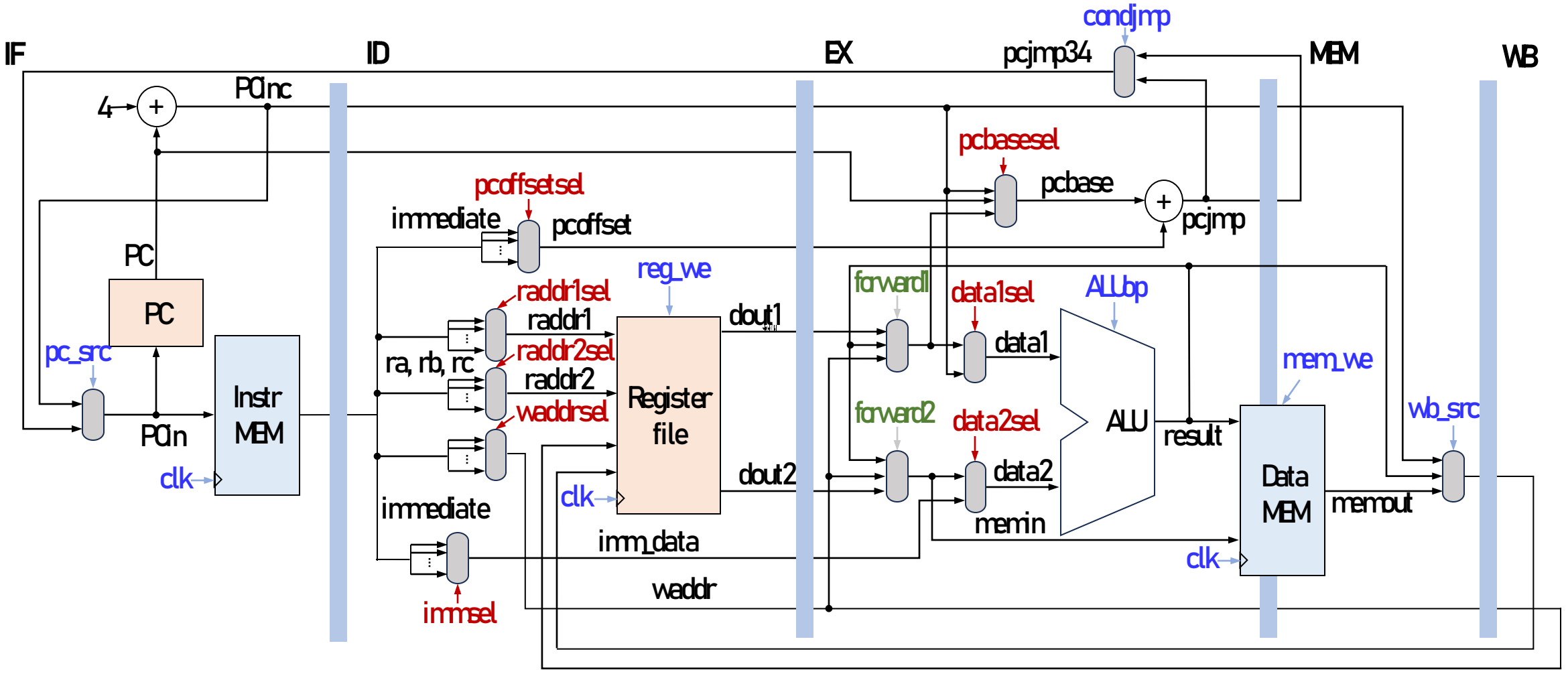


Image source: https://ocw.mit.edu/courses/6-004-computation-structures-spring-2017/pages/c15/c15s1/#47

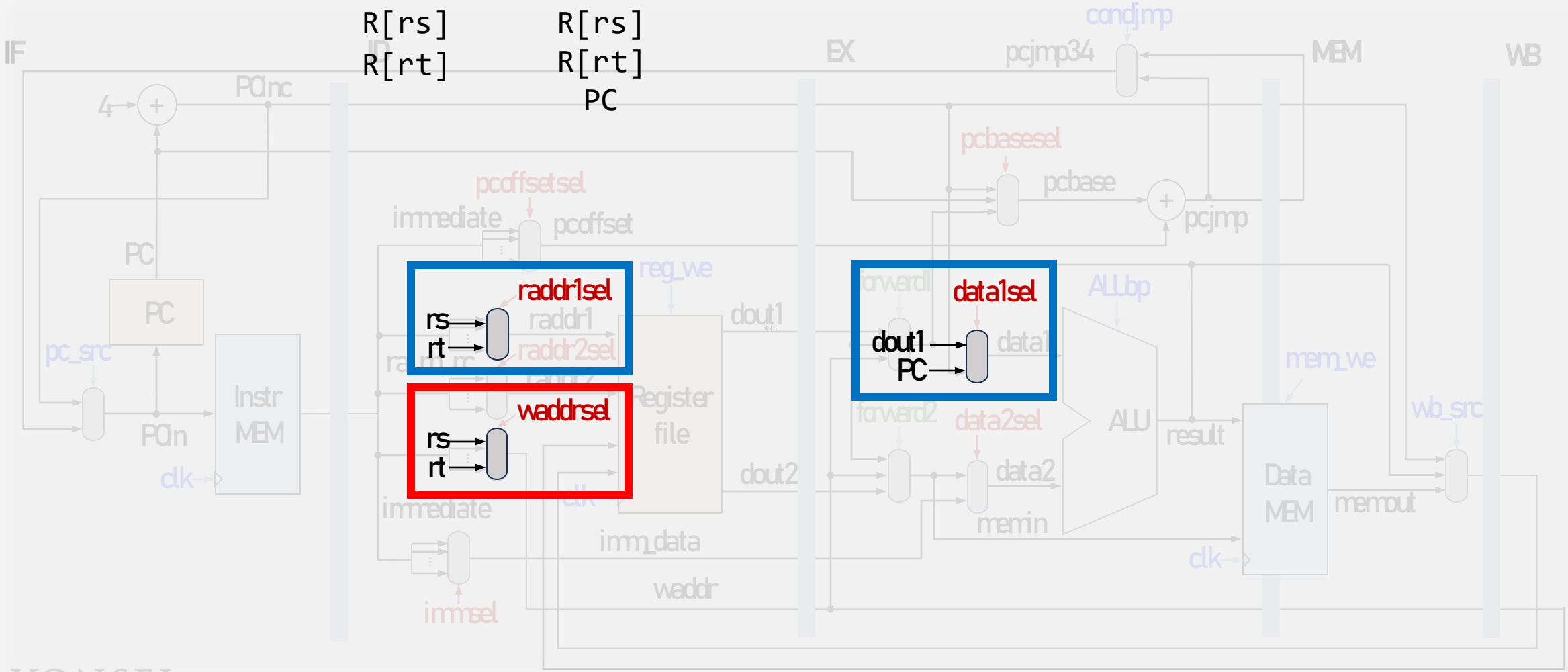
Generalized 5-stage Pipelined Architecture



Generate Data Path and Control Signals

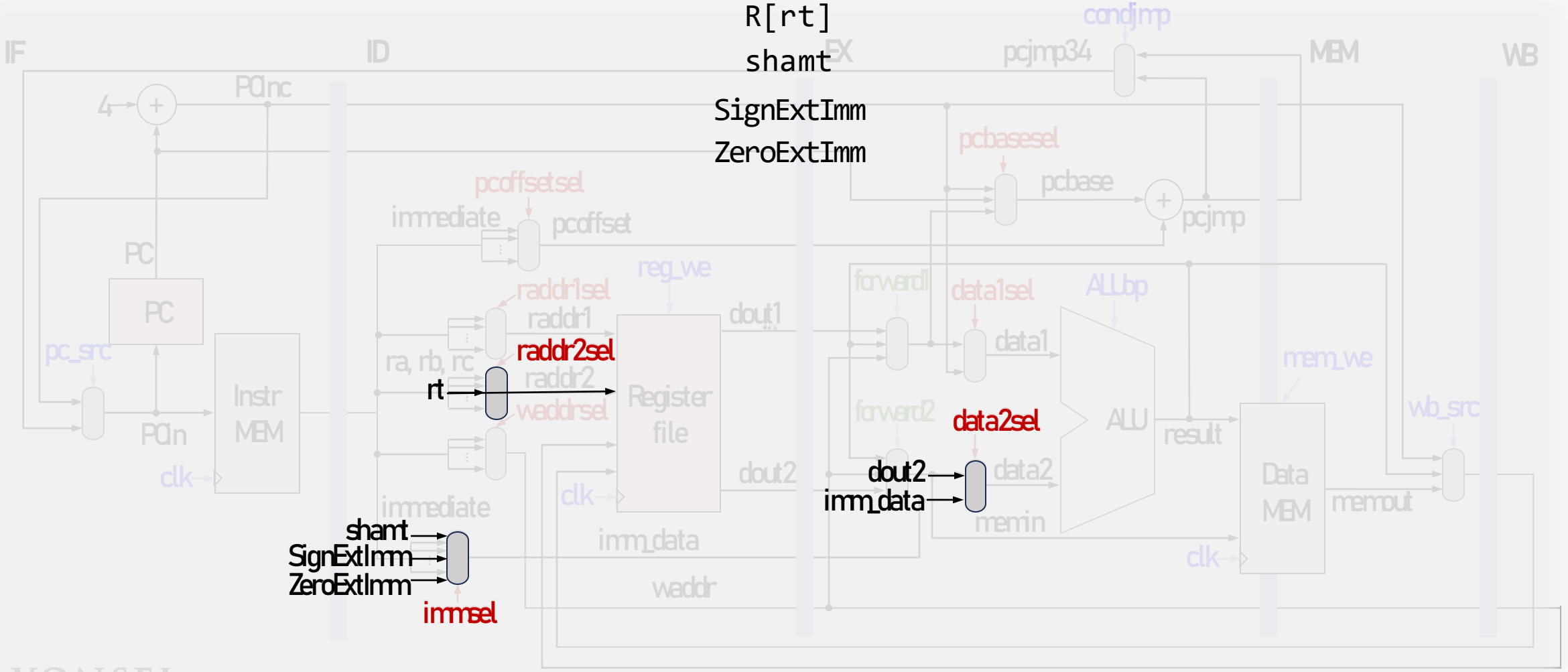
R[rs]
R[rt]

R[rs]
R[rt]
PC



Generate Data Path and Control Signals

R[rt]
shamt
SignExtImm
ZeroExtImm

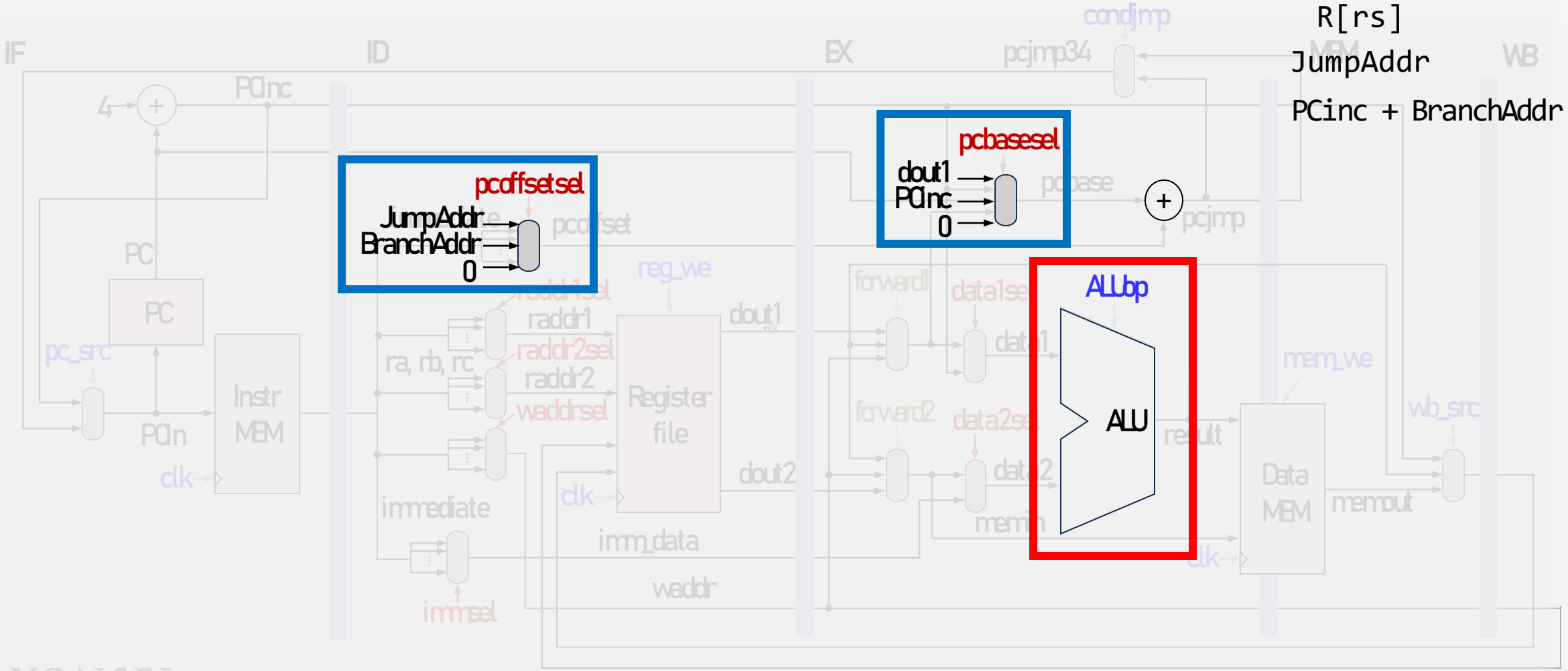


Generate Data Path and Control Signals

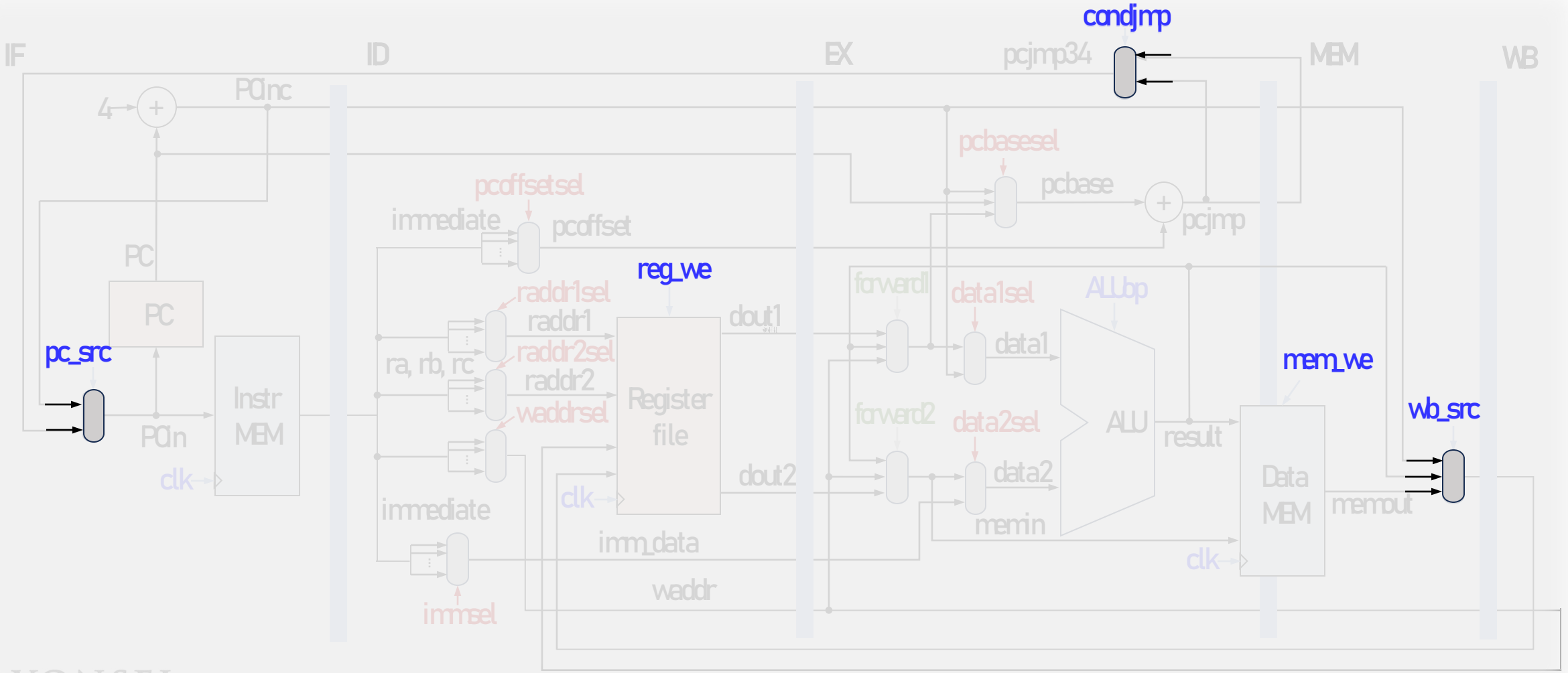
R[rs]

JumpAddr

PCinc + BranchAddr



Generate Data Path and Control Signals



Result – Generated Verilog Code (datapath)

```
//waddr MUX
always @(*) begin
    case(waddrsel)
     1: waddr = rt;
     2: waddr = 31;
     default: waddr = rd;
    endcase
end
//raddr1 MUX
always @(*) begin
    if (raddr1sel == 1) raddr1 = rt;
    else raddr1 = rs;
end
//raddr2 assign
assign raddr2 = rt;
//pcoffset MUX
always @(*) begin
    case(pcoffsetsel)
     1: pcoffset = 0;
     2: pcoffset = BranchAddr;
     default: pcoffset = JumpAddr;
    endcase
end
```

```
//data1 MUX
always @(*) begin
    if (data1sel_3 == 1) data1 = PC_3;
    else data1 = dout1_3;
end
//imm MUX
always @(*) begin
    case(immsel)
     1: imm = imm_s;
     2: imm = imm_u;
     default: imm = imm_i;
    endcase
end
//data2 MUX
always @(*) begin
    if (data2sel_3 == 1) data2 = dout2_3;
    else data2 = imm_3;
end
//pcbase MUX
always @(*) begin
    if (pcbasesel_3 == 1) pcbase = dout1_3;
    else pcbase = PC_3;
end
```

Result – Generated Verilog Code (control signals)

```
//data1sel
always @(*) begin
    if( (opcode == 6'h03) ) data1sel = 1;
    else data1sel = 0;
end
//immsel
always @(*) begin
    case(opcode)
     6'h0C, 6'h0D: immsel = 2;
     6'h03: immsel = 3;
     6'h0F: immsel = 4;
     6'h00:
     case(funct)
      6'h00, 6'h02: immsel = 1;
      default: immsel = 0;
     endcase
     default: immsel = 0;
    endcase
end
```

```
//mem_we
always @(*) begin
    if( (opcode == 6'h28)  ||
        (opcode == 6'h29)  ||
        (opcode == 6'h2B) ) mem_we = 1;
    else mem_we = 0;
end
always @(posedge clk) begin
    pc_src_3 <= jump_next ? 1'b0 : pc_src;
end
//pc_src
always @(*) begin
    if( (opcode == 6'h00) && (funct == 6'h08)  ||
        (opcode == 6'h02)  ||
        (opcode == 6'h03) ) pc_src = 1;
    else pc_src = 0;
end
```

Result – Quartus Compile

Flow Summary

<<Filter>>

Flow Status	Successful - Thu Jun 19 17:59:50 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	mips
Top-level Entity Name	mips
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	1,308 / 114,480 (1 %)
Total registers	342
Total pins	34 / 529 (6 %)
Total virtual pins	0
Total memory bits	12,288 / 3,981,312 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Conclusions

- ISA2CPUgen Tool generates CPU design from ISA specification
  - 5-stage pipelined design
  - single cycle design
- Generated design can be implemented and operated on FPGA board
- The tool is applicable to RISC-family ISAs (without Flags)
- Instructions can be added or removed into the ISA.
- Can implement Prototype of processors with new ISA

ACKNOWLEDGMENTS

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the National Program in Medical AI Semiconductor)(2024-0-0097) supervised by the IITP(Institute of Information & Communications Technology Planning&Evaluation) in 2025.

Thank you

Q&A