32-bit Single-Cycle RISC-V Architecture RTL & Testbench

Basma Gamal

Design Requirements

- Single-Cycle 32-bits RISC-V
- Top module divided into datapath and control logic.
- The instruction and data memories should be connected to the top module in the testbench
- reading register-file/memory data is combinational, but writing is clocked.
- Instruction memory (Imem): 1kB word-addressable ROM
- Data memory (Dmem): 1kB word-addressable RAM & connect only bits 9 to 2 in the address bus.

Design Requirements : Supported operations

R-Format: instructions using 3 register inputs : **add**, **sub** , **and** , **or**

I-Format: instructions with immediate, loads: **addi**, **ori**, **lw**, **jalr**

S-Format: store instructions: **sw**

B-Format: branch instructions: **beq**, **bne**

J-Format: the jump instruction: **jal**

Testbench Requirements

- Memory access 100 ps delay.
- Major stage in the datapath has 50 ps delay.
- Simple stages has 10 ps delay.
- Clock period 1 ns.

Assembly code

```
main: addi x2, x0, 5       # x2 = 5 0 00500113
addi x3, x0, 12            # x3 = 12 4 00C00193
addi x7, x3, -9            # x7 = (12 - 9) = 3 8 FF718393
or x4, x7, x2              # x4 = (3 OR 5) = 7 C 0023E233
and x5, x3, x4             # x5 = (12 AND 7) = 4 10 0041F2B3
add x5, x5, x4             # x5 = 4 + 7 = 11 14 004282B3
beq x5, x7, end            # shouldn't be taken 18 02728863
#slt x4, x3, x4            # x4 = (12 < 7) = 0 1C 0041A233
beq x4, x0, around         # should be taken 20 00020463
addi x5, x0, 0             # shouldn't execute 24 00000293
around: #slt x4, x7, x2    # x4 = (3 < 5) = 1 28 0023A233
add x7, x4, x5             # x7 = (1 + 11) = 12 2C 005203B3
sub x7, x7, x2             # x7 = (12 - 5) = 7 30 402383B3
sw x7, 84(x3)              # [96] = 7 34 0471AA23
lw x2, 96(x0)              # x2 = [96] = 7 38 06002103
add x9, x2, x5             # x9 = (7 + 11) = 18 3C 005104B3
jal x3, end                # jump to end, x3 = 0x44 40 008001EF
addi x2, x0, 1             # shouldn't execute 44 00100113
end: add x2, x2, x9        # x2 = (7 + 18) = 25 48
sw x2, 0x20(x3)            # [100] = 25 0221A023
done: beq x2, x2, done     # infinite loop 50 00210063
```

Specifications

- ✓ **32**-bit **datapath** **(d=32)**, Limited to integer operations
- ✓ **32**-bit **Fixed length instruction** codes
- ✓ Single-cycle needs **two separate memories** (Imem & Dmem)
- ✓ Data Memory (**Dmem**): **1Kbyte**
- ✓ Instruction memory (**Imem**): **1Kbyte**
- ✓ **Word-Aligned** memories
- ✓ The **Register file** is **32 registers**
- ✓ Address lines 10 **(n=10)**

Instruction Formats Types

R-Format: instructions using 3 register inputs : **add**, **sub** , **and** , **or**

I-Format: instructions with immediate, loads: **addi**, **ori**, **lw**, **jalr**

S-Format: store instructions: **sw**

B-Format: branch instructions: **beq**, **bne**

J-Format: the jump instruction: **jal**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Bits/Types
funct7							rs2					rs1					funct3			rd					OPcode							R-Type
imm[11:0]												rs1					funct3			rd					OPcode							I-Type
imm[11:5]							rs2					rs1					funct3			imm[4:0]					OPcode							S-Type
imm[12|10:5]							rs2					rs1					funct3			imm[4:1|11]					OPcode							B-Type
imm[20|10:1|11]												imm[19:12]								rd					OPcode							J-Type

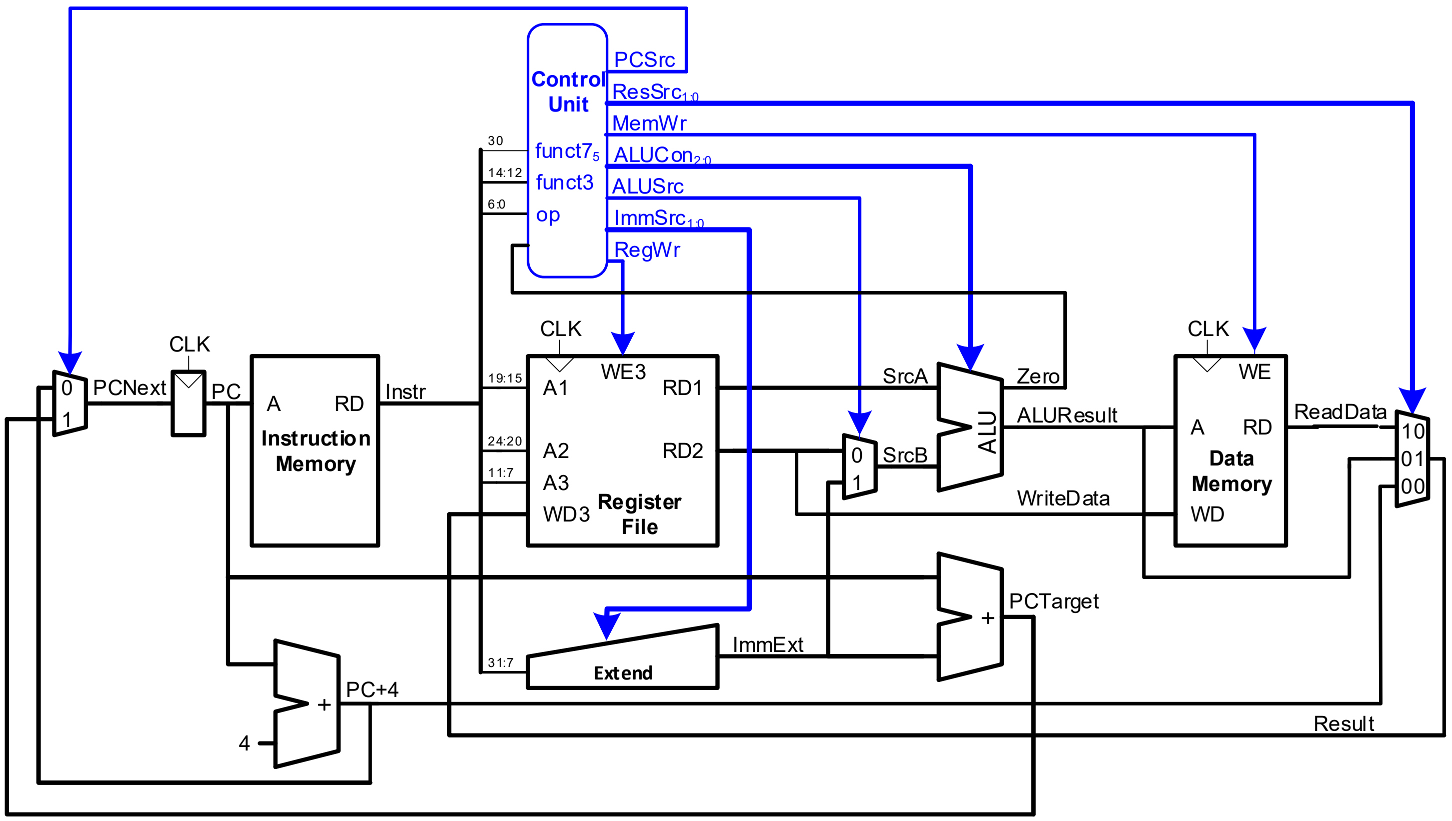
Supported Instruction

Instr.	Opcode [6:0]	Funct3 [14:12]	Funct7 [31:25]	Name	Function	Format
lw	0000_011 (03)	010	xx	Load Word	rd = dmem[rs1+ immV]	I
addi	0010_011 (19)	000	xx	ADD Imm	rd = rs1 + immV	I
ori	0010_011 (19)	110	xx	OR Imm	rd = rs1 | immV	I
sw	0100_011 (35)	010	xx	Store Word	dmem[rs1+ immV] = rs2	S
add	0110_011 (51)	000	0000_000	Addition	rd = rs1 + rs2	R
sub	0110_011 (51)	000	0100_000	Subtract	rd = rs1 - rs2	R
or	0110_011 (51)	110	0000_000	OR	rd = rs1 | rs2	R
and	0110_011 (51)	111	0000_000	AND	rd = rs1 & rs2	R
beq	1100011 (99)	000	xx	Branch if Eq	If rs1 == rs2 : pc = pc +immV	B
bnq	1100011 (99)	001	xx	Branch if not Eq	If rs1 != rs2 : pc = pc +immV	B
jalr	1100111 (103)	000	xx	Jump and Link from Register + ImmV	rd = pc +4 pc (new) = rs1 + immV	I
jal	1101111 (111)	xx	xx	Jump and Link from ImmV	rd = pc +4 pc (new) = pc + immV	J

RISC-V Architecture

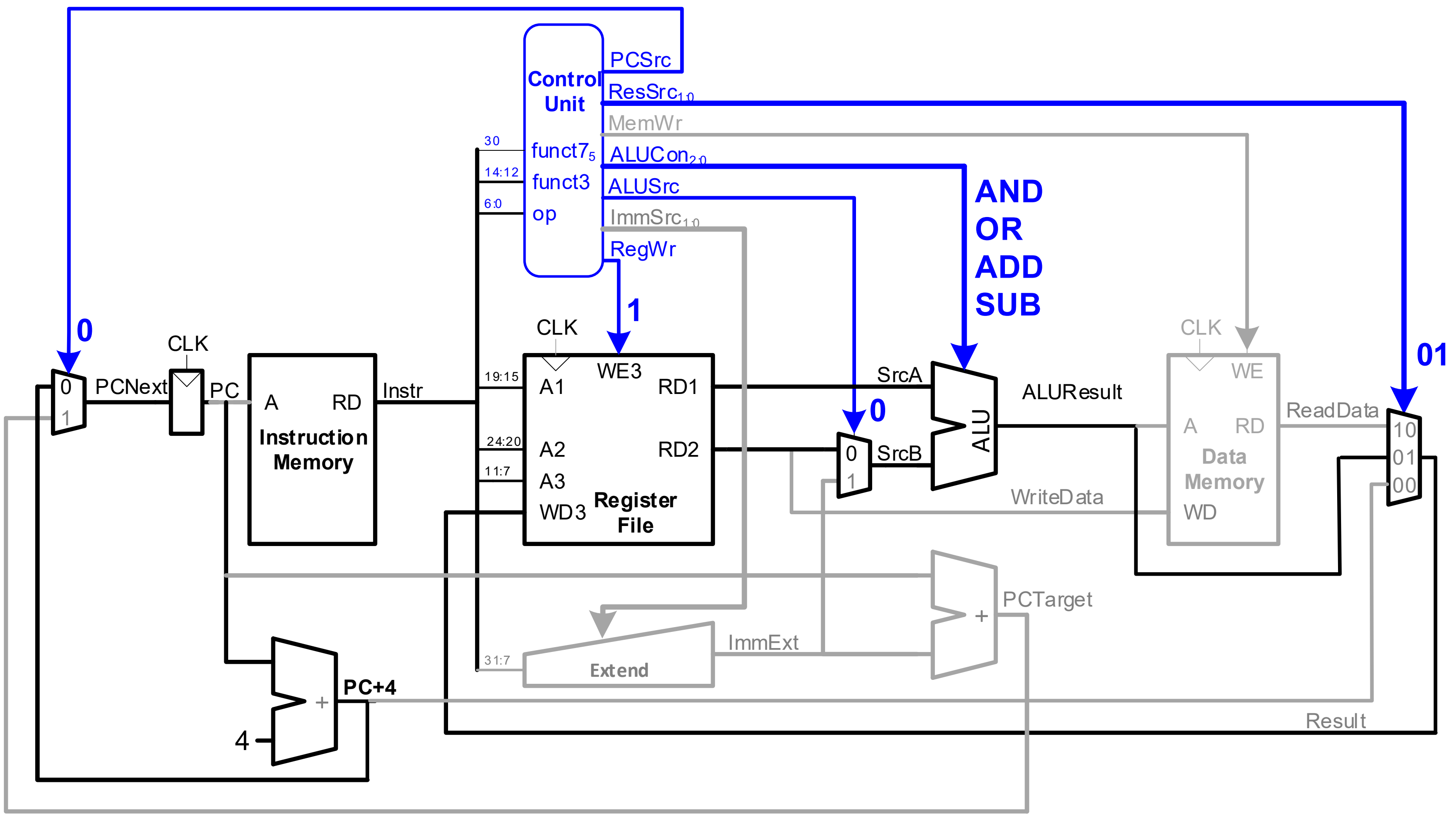
Instr.	Supported
lw	Yes
addi	Yes
ori	Yes
andi	Yes
sw	Yes
add	Yes
sub	Yes
xor	Yes
or	Yes
and	Yes
beq	Yes
bnq	Yes
jalr	NO
jal	Yes

RISC-V architecture adopted from the ref*

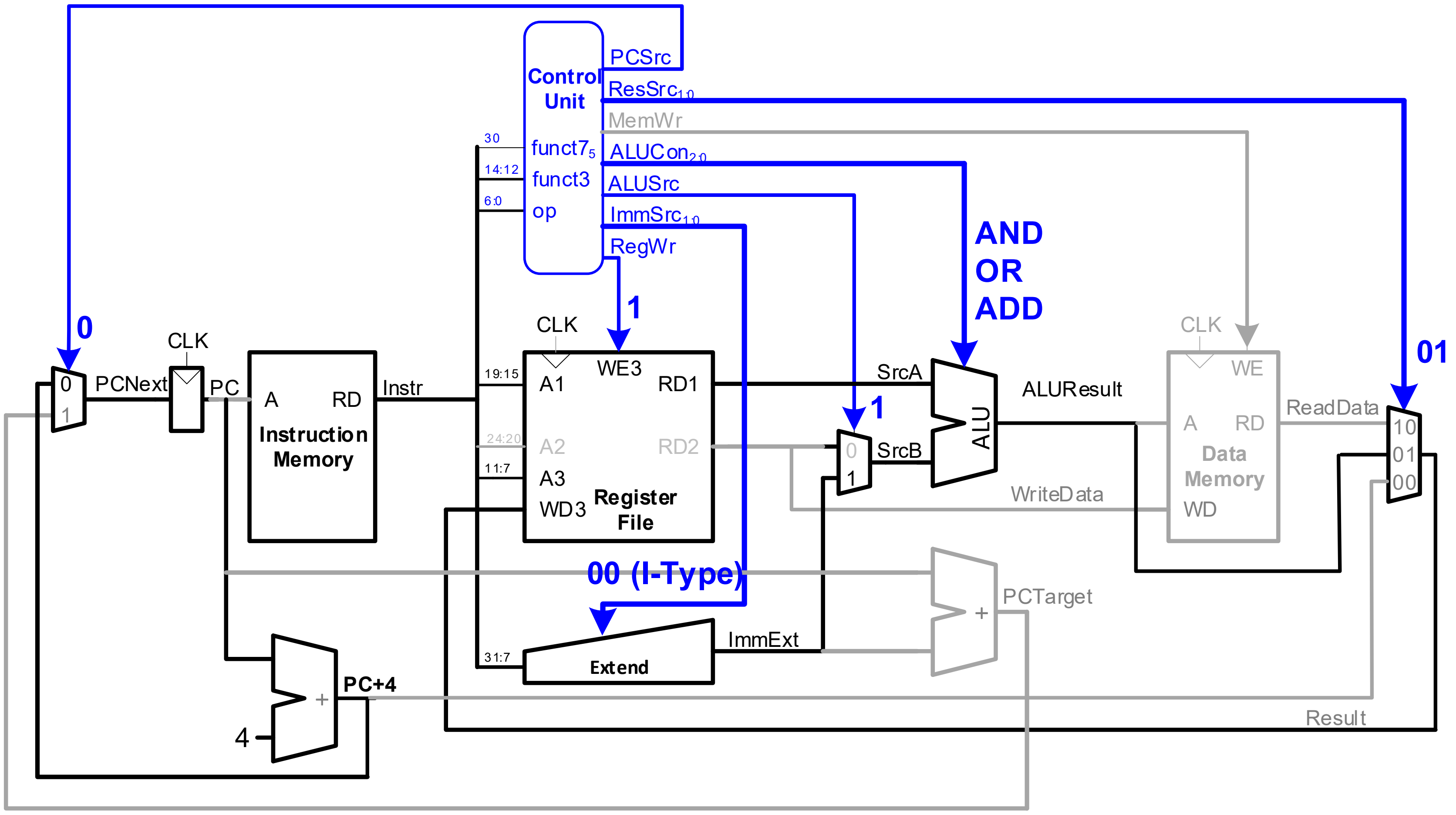


*Digital Design and Computer Architecture: RISC-V Edition Harris & Harris © 2020 Elsevier

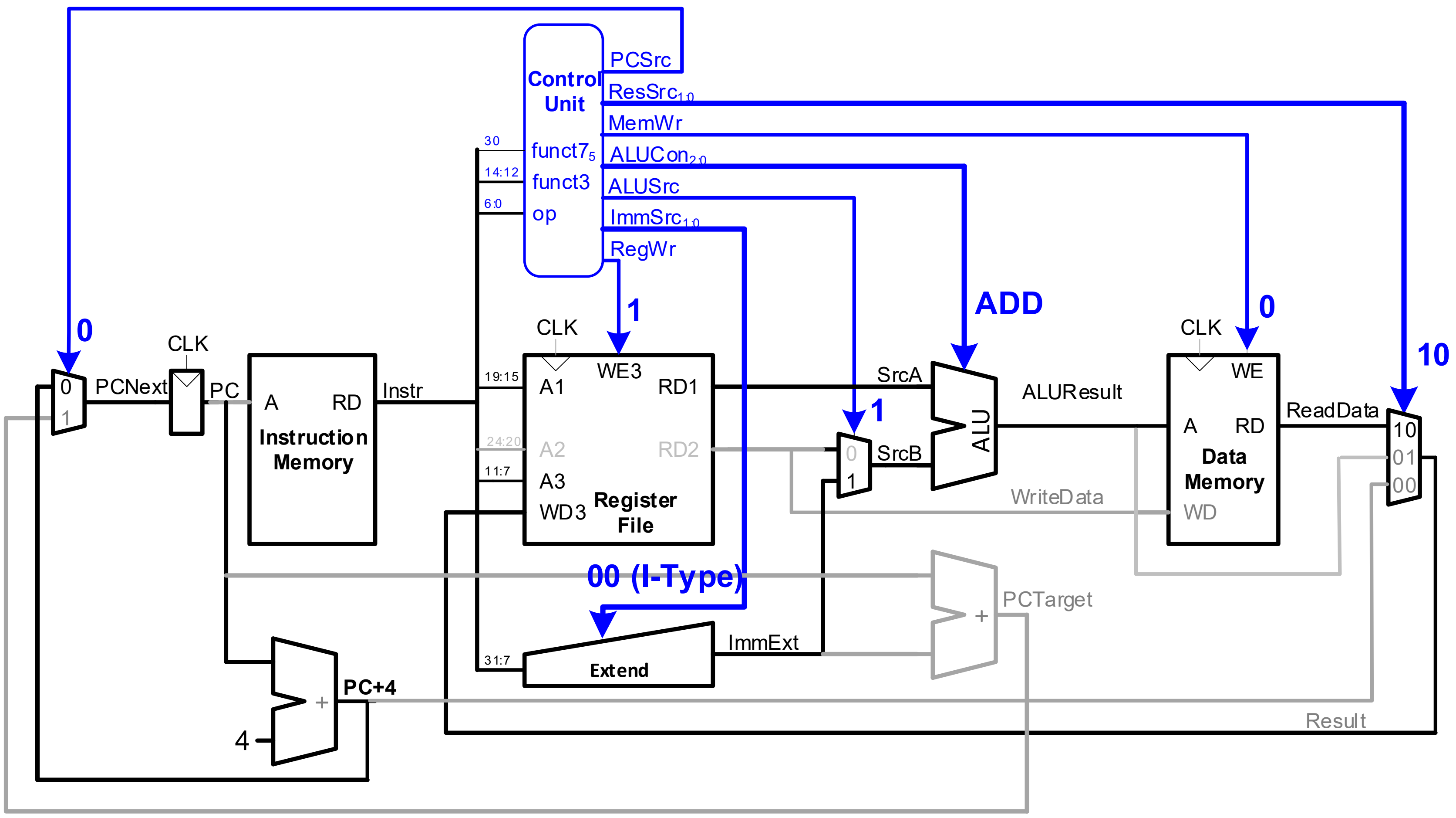
and – or – xor - add – sub



addi – ori – andi

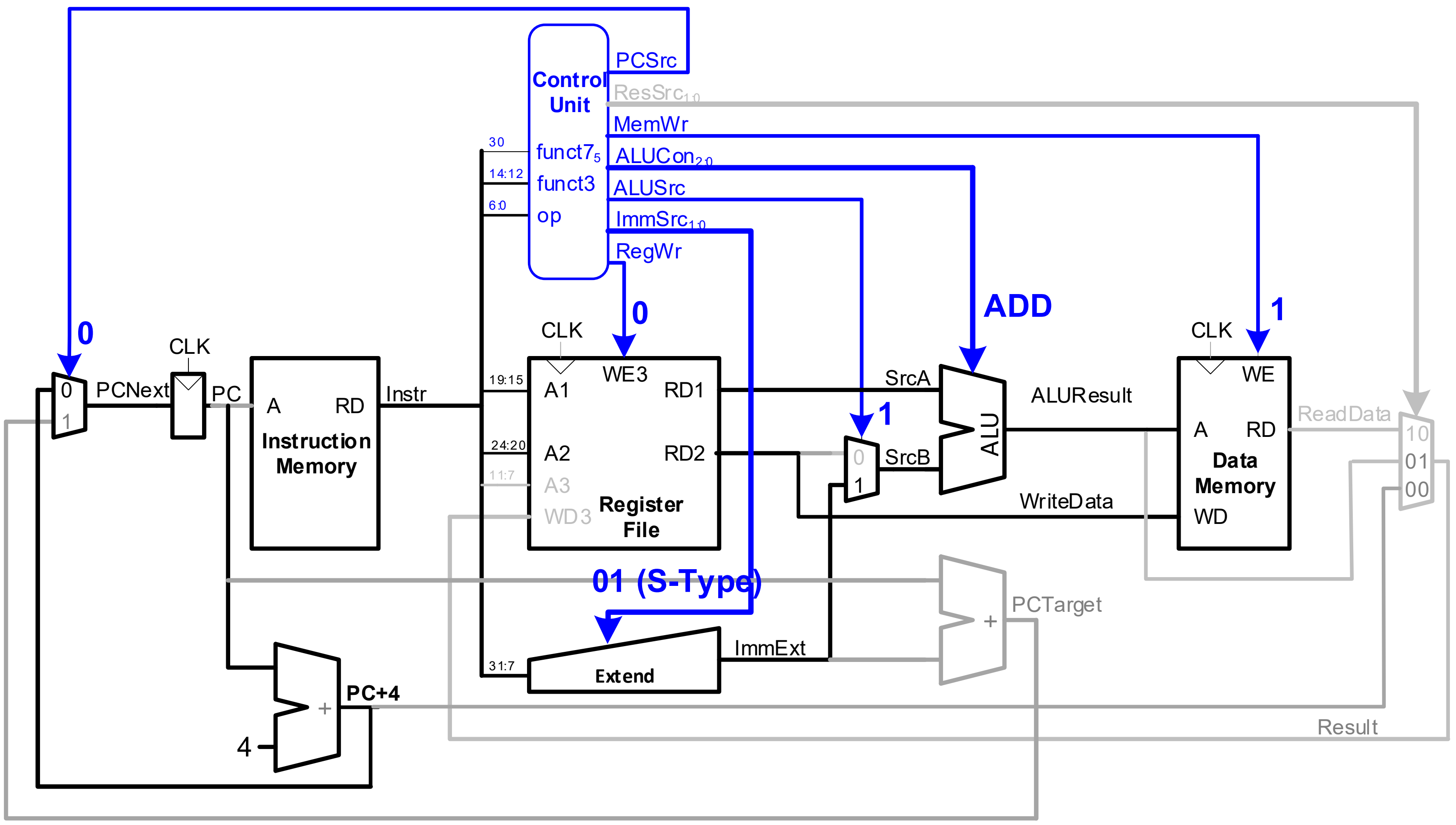


lw



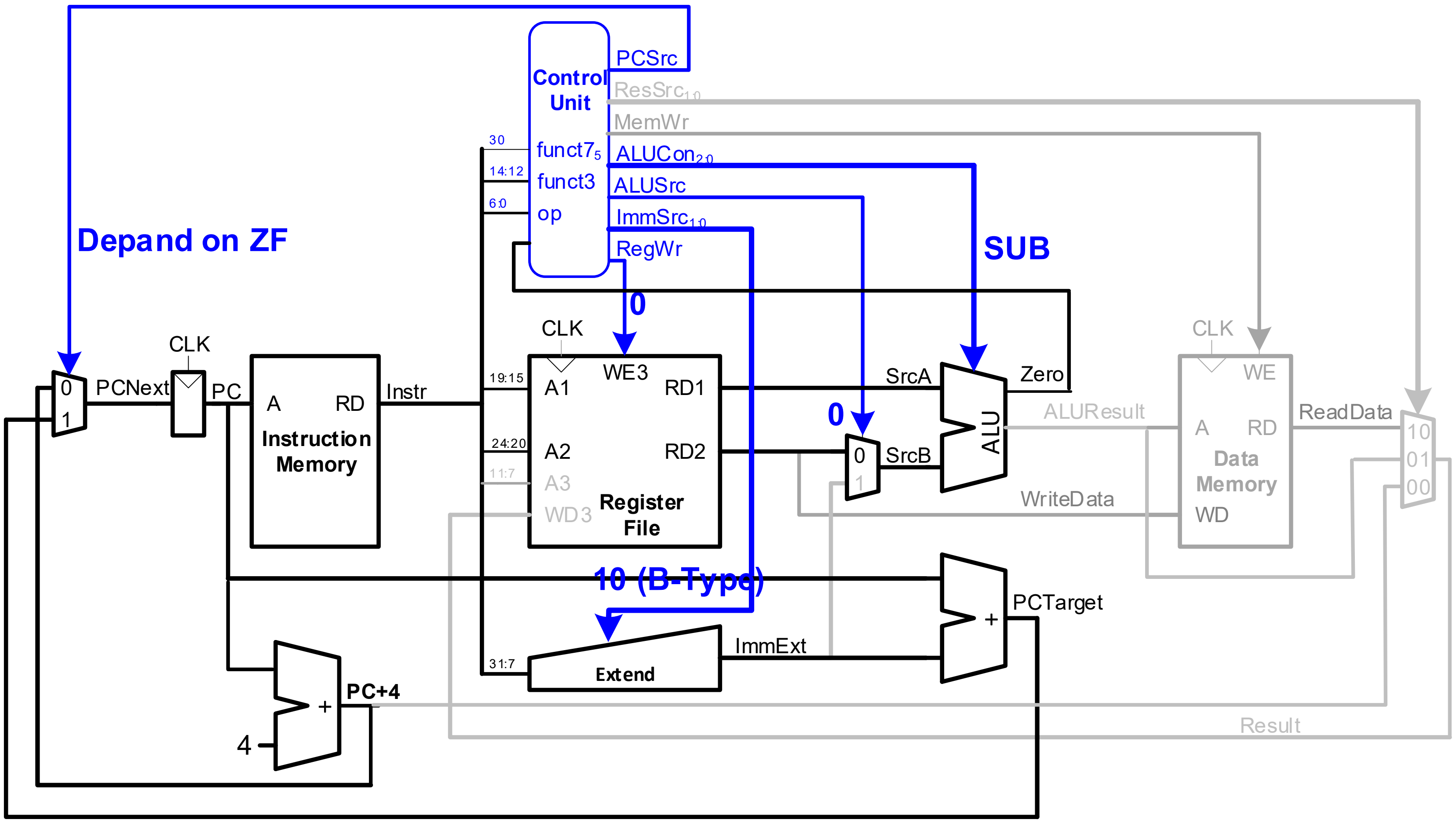
Control Unit
PCSrc
ResSrc1:0
MemWr
ALUCon2:0
ALUSrc
ImmSrc1:0
RegWr
30
14:12
6:0
funct75
funct3
op
0
CLK
PCNext
PC
A
RD
Instruction Memory
Instr
19:15
24:20
11:7
31:7
CLK
1
WE3
A1
A2
A3
WD3
RD1
RD2
Register File
SrcA
SrcB
1
0
ADD
ALU
ALUResult
WriteData
CLK
0
WE
A
RD
Data Memory
WD
ReadData
10
10
01
00
00 (I-Type)
Extend
ImmExt
PCTarget
PC+4
4
Result

SW



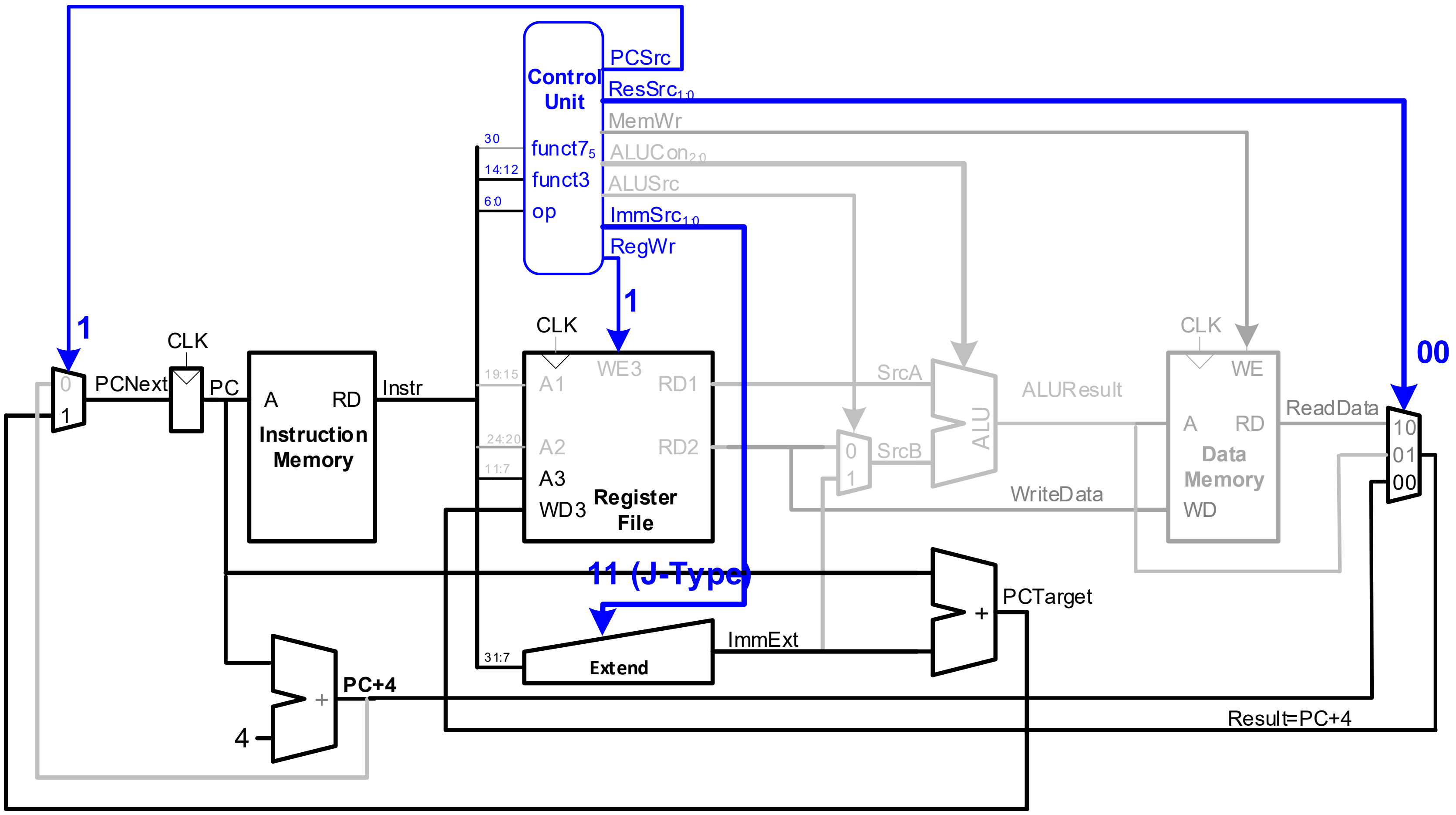
Control Unit
PCSrc
ResSrc1:0
MemWr
ALUCon2:0
ALUSrc
ImmSrc1:0
RegWr
funct75
funct3
op
30
14:12
6:0
0
0
1
ADD
1
CLK
PCNext
PC
A
RD
Instruction Memory
Instr
19:15
24:20
11:7
31:7
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
ALUResult
WriteData
WE
A
RD
WD
Data Memory
ReadData
10
01
00
01 (S-Type)
Extend
ImmExt
PCTarget
PC+4
4
Result

beq- bnq



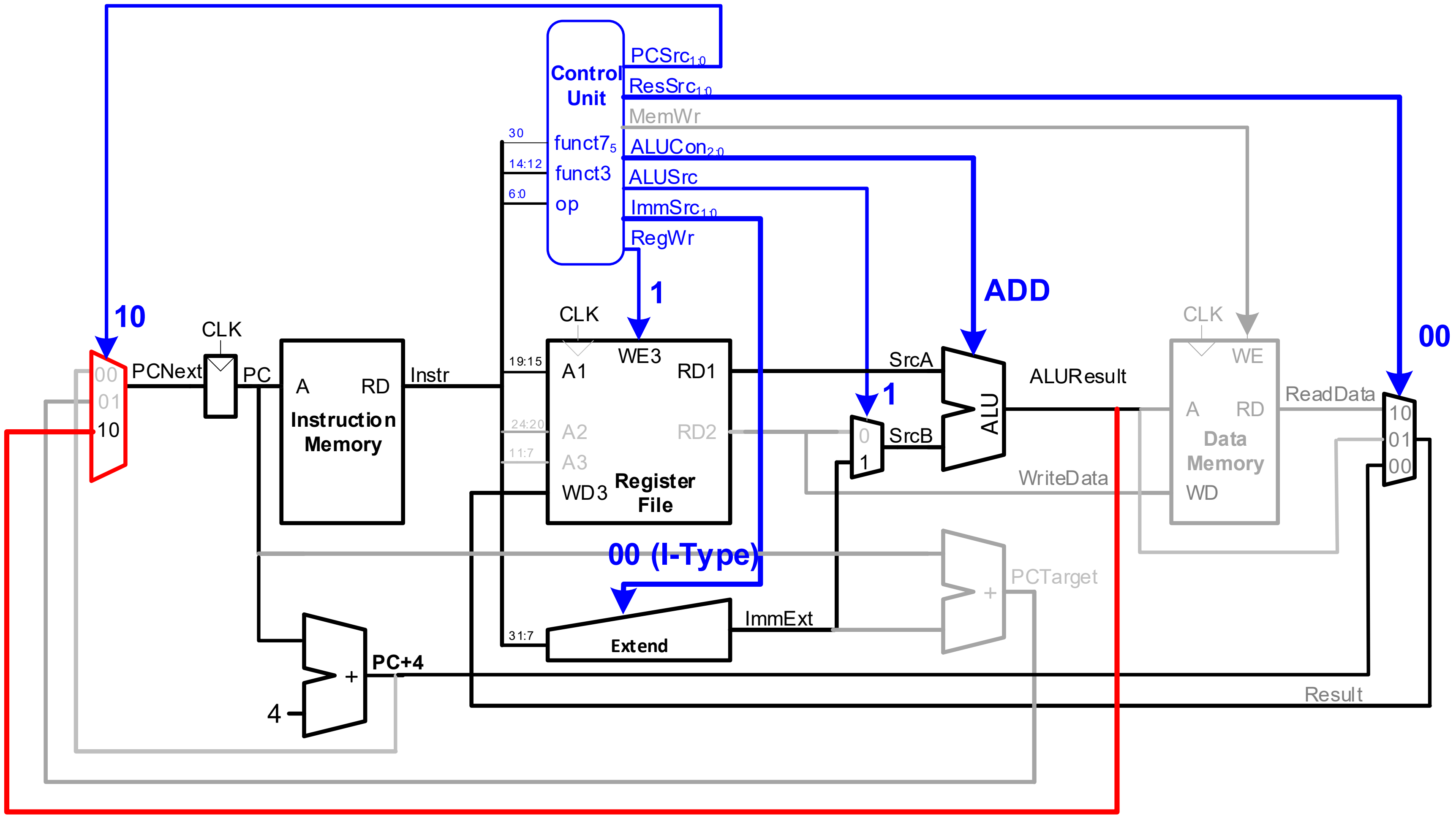
Depand on ZF
SUB
0
0
10 (B-Type)
Control Unit
PCSrc
ResSrc1:0
MemWr
ALUCon2:0
ALUSrc
ImmSrc1:0
RegWr
funct75
funct3
op
30
14:12
6:0
CLK
PCNext
PC
A
RD
Instruction Memory
Instr
19:15
24:20
11:7
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
Data Memory
WE
WD
ReadData
10
01
00
PCTarget
ImmExt
Extend
31:7
PC+4
4
Result

jal



PCSrc
ResSrc1:0
MemWr
ALUCon2:0
ALUSrc
ImmSrc1:0
RegWr
Control Unit
funct75
funct3
op
30
14:12
6:0
1
1
00
11 (J-Type)
CLK
PCNext
PC
A
RD
Instruction Memory
Instr
19:15
24:20
11:7
31:7
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
ALUResult
WriteData
A
RD
WE
Data Memory
WD
ReadData
10
01
00
0
1
Extend
ImmExt
PCTarget
PC+4
4
Result=PC+4

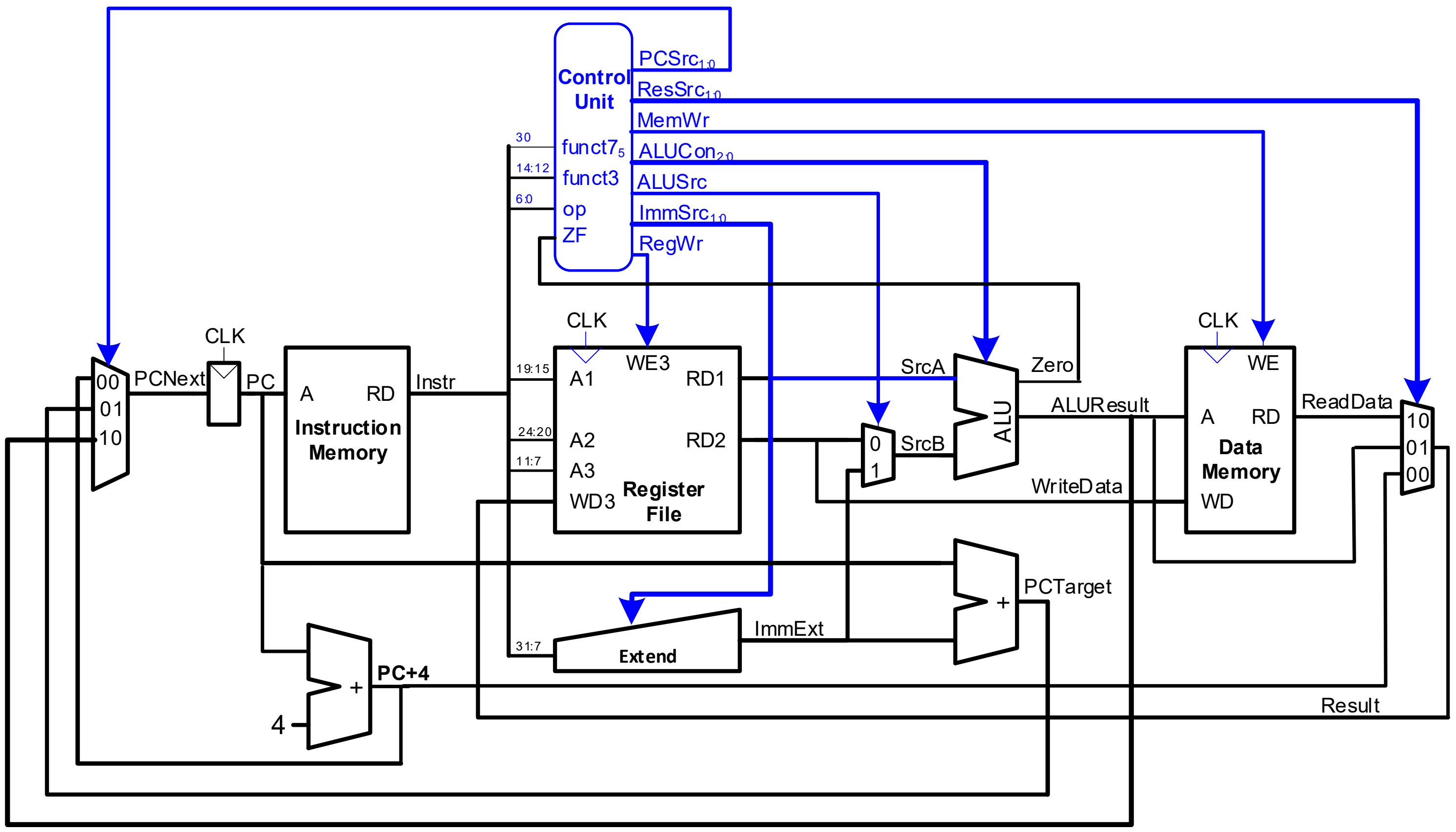
jalr



PCSrc1:0
ResSrc1:0
MemWr
ALUCon2:0
ALUSrc
ImmSrc1:0
RegWr
Control Unit
funct7_5
funct3
op
30
14:12
6:0
10
00
01
10
PCNext
CLK
PC
A
RD
Instruction Memory
Instr
1
CLK
WE3
19:15
A1
RD1
24:20
A2
RD2
11:7
A3
WD3
Register File
ADD
SrcA
1
0
1
SrcB
ALU
ALUResult
WriteData
CLK
WE
A
RD
Data Memory
WD
ReadData
00
10
01
00
00 (I-Type)
31:7
Extend
ImmExt
PCTarget
+
PC+4
+
4
Result

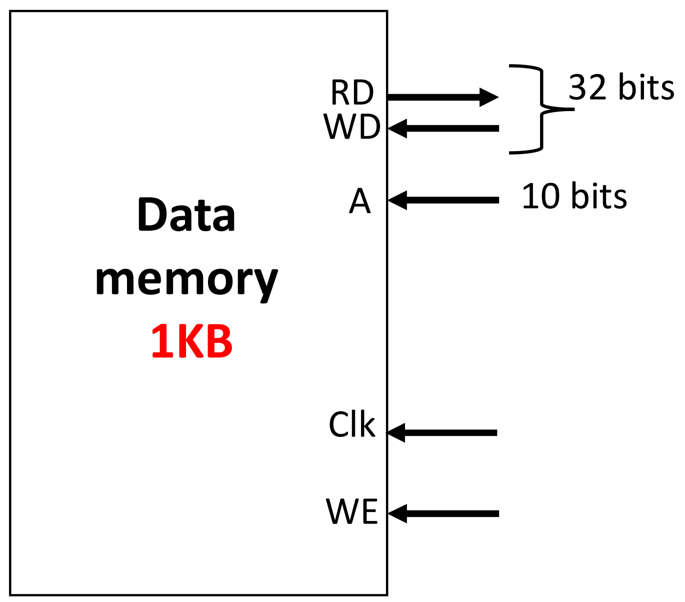
Instr.	Supported
lw	Yes
addi	Yes
ori	Yes
andi	Yes
sw	Yes
add	Yes
sub	Yes
xor	Yes
or	Yes
and	Yes
beq	Yes
bnq	Yes
jalr	Yes
jal	Yes

Final RISC-V architecture Design



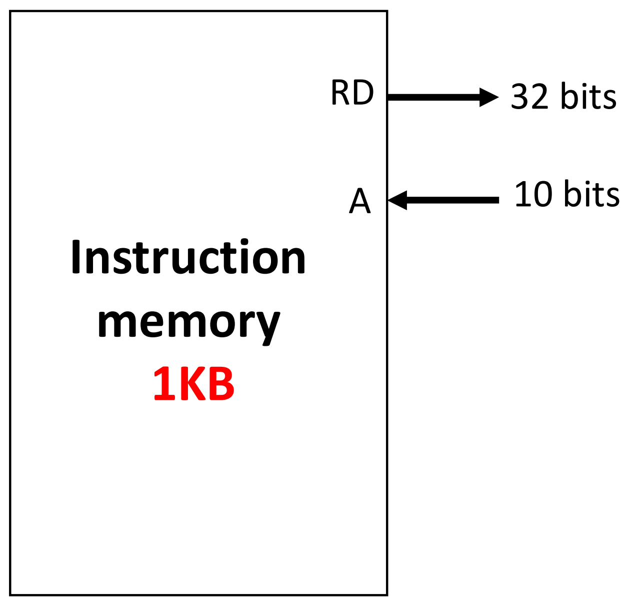
RISC-V Verilog Codes

Data Memory



```
// 1KB 32-bit aligned RAM with Asynchronous Read
module Dmem #(parameter n=10 , d=32)
            (input clk , we , [d-1:0] wd , [n-1:0] a,
             output  [d-1:0] rd);
reg [d-1:0] mem [(2**n)-1:0];

always @(posedge(clk))
// Synchronous write
if (we)  mem[a[9:2]] = wd ;

// asynchronous Read
assign rd = mem[a[9:2]] ;
endmodule
```

Instruction Memory

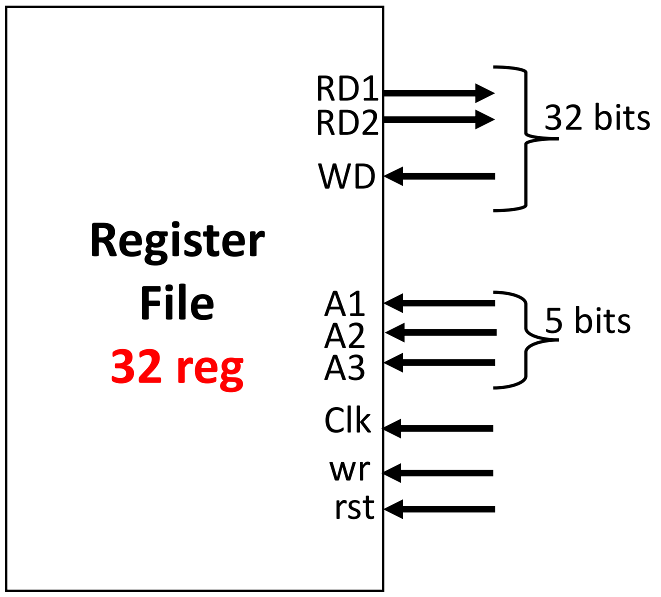


```
// 1KB 32-bit ROM Combinational Read
module Imem #(parameter n=10 , d=32)
            (input  [n-1:0] a,
             output [d-1:0] rd);

reg [d-1:0] mem [(2**n)-1:0];

initial
$readmemh("hexcode.txt",mem);

assign rd = mem[a[n-1:2]];
endmodule
```

hexcode.txt

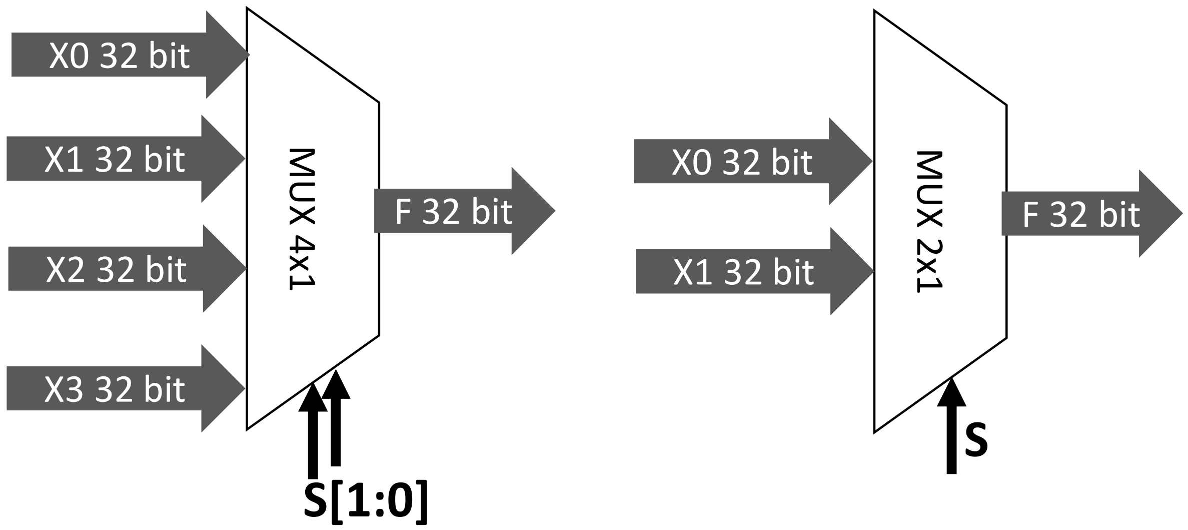
```
00500113
00C00193
FF718393
0023E233
0041F2B3
004282B3
02728463
00020463
00000293
005203B3
402383B3
0471AA23
06002103
005104B3
008001EF
00100113
00910133
0221A023
00210063
```

Data Path: Register file



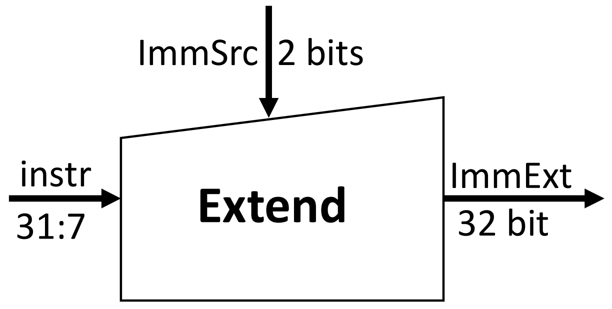
```
module reg_file #(parameter d = 32)(input clk , rst , wr
,[d-1:0]wd, [4:0]a1, a2, a3, output [d-1:0] rd1 , rd2);

reg [d-1:0] regs [0: 31];
integer i ;

// asynchronous read + R0 is hardwired to 0
assign rd1 = (a1) ? regs [a1] : 0;
assign rd2 = (a2) ? regs [a2] : 0;

always @(posedge(clk)) begin
// Initialize
if (rst)
   for (i=0 ; i< 32 ; i=i+1)
    regs[i]=0 ;
// Synchronous Write
else if (wr && a3)
  regs[a3] = wd;
end
endmodule
```

Data Path: 32-bits Multiplexers



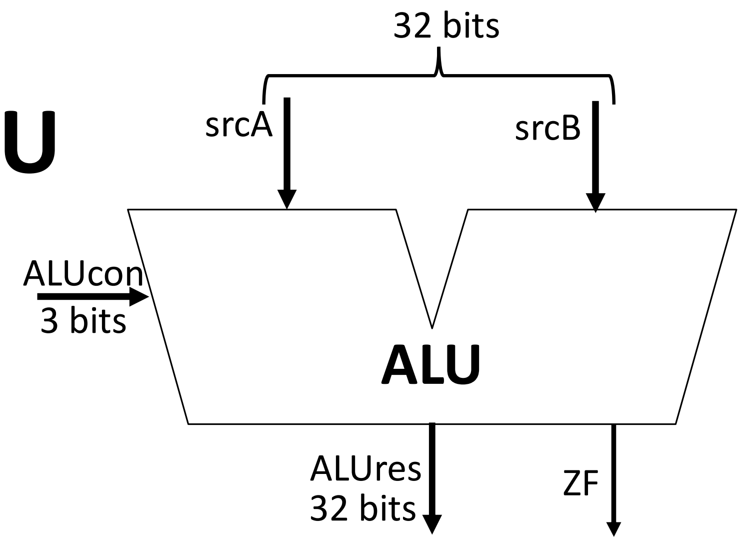
```
module mux2x1 #(parameter n = 32) (input [n-1:0] x0 , x1 , input s , output [n-1:0] f);
    assign f = (s)? x1 : x0 ;
endmodule

module mux4x1 #(parameter n =32 ) (input [n-1:0] x0 , x1 , x2 , x3 , input [1:0] s ,
output [n-1:0] f);
     assign f = (s==2'b00) ? x0 : (s==2'b01) ? x1 : (s==2'b10) ? x2 : x3 ;
endmodule
```

Data Path: Extend



```
// 00 I-Type , 01 S-Type , 10 B-Type , 11 J-Type
module extend (input [1:0] ImmSrc , [31:7] instr , output [31:0] ImmExt);
assign ImmExt=(ImmSrc==0)? {{20{instr[31]}}, instr[31:20]}:
              (ImmSrc==1)? {{20{instr[31]}}, instr[31:25], instr[11:7]} :
              (ImmSrc==2)? {{19{instr[31]}}, instr[31], instr[7], instr[30:25], instr[11:8], 1’b0}:
              {{12{instr[31]}}, instr[19:12], instr[20],instr[30:21], 1'b0} ;
endmodule
```

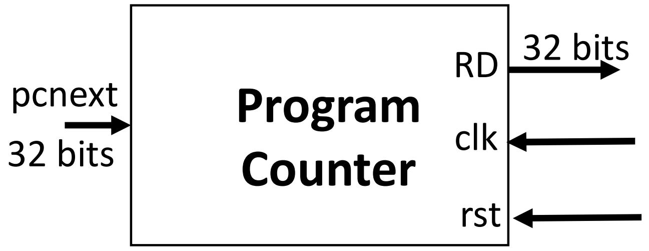
Data Path: ALU



Operation	ALUcon
AND	000
OR	001
ADD	010
SUB	110

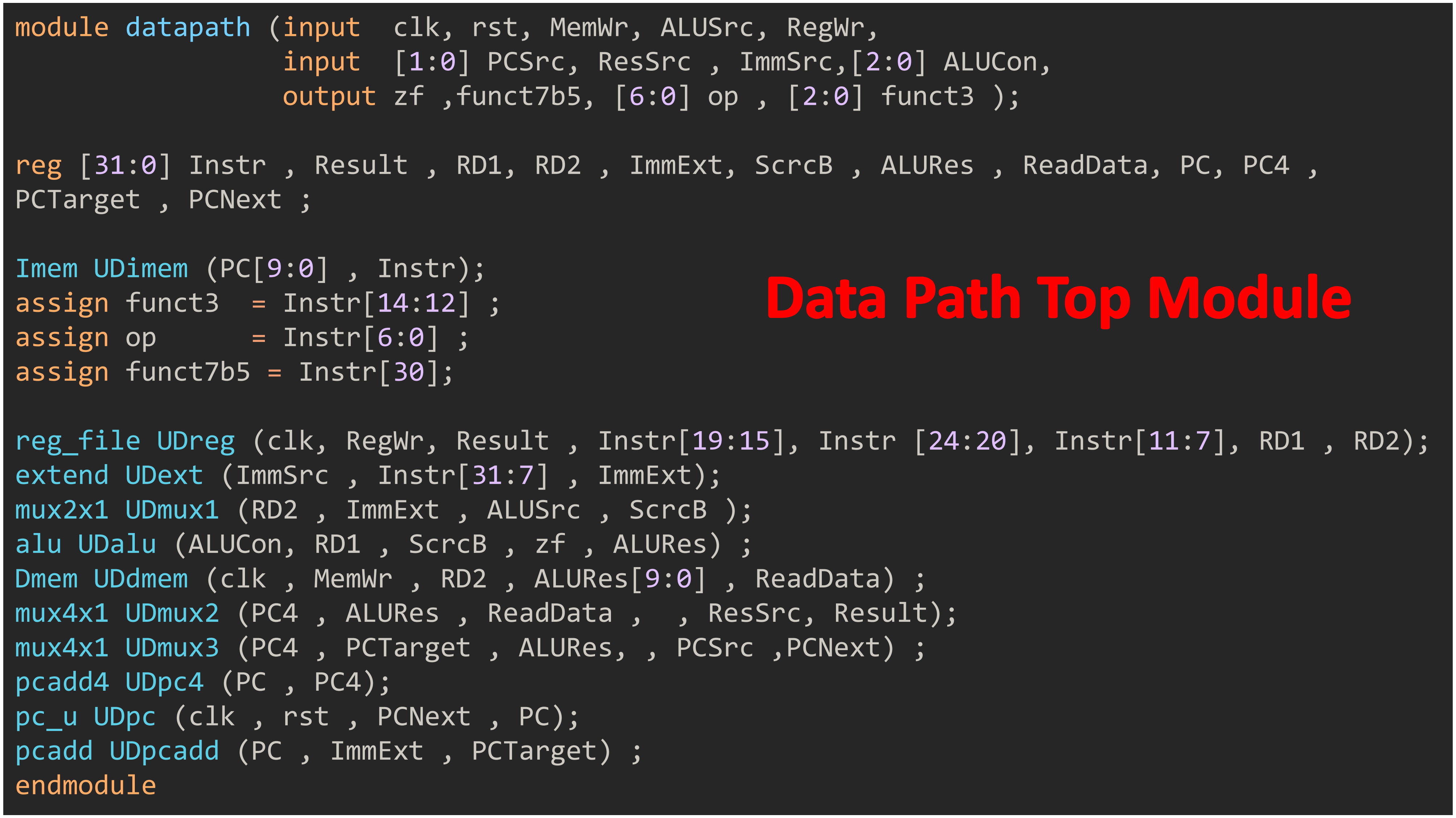
```
module alu #(parameter d = 32) (input [2:0] ALUcon , [d-1:0] srcA, srcB, output zf , reg
[d-1:0] ALUres);
assign zf = (!ALUres) ? 1 : 0 ;
always @(*)
case (ALUcon)
3'b000  : ALUres = srcA & srcB ;
3'b001  : ALUres = srcA | srcB ;
3'b010  : ALUres = srcA + srcB ;
3’b110  : ALUres = srcA - srcB ;
default : ALUres = {d{1'bx}} ;
endcase
endmodule
```

Data Path: Program Counter



```
module pc_u #(parameter n = 32) (input clk , rst , input [n-1:0] pcnext ,
                                output reg [n-1:0] pc);
always @(posedge clk )
    if (rst) pc = 0;
    else     pc = pcnext ;
endmodule
```

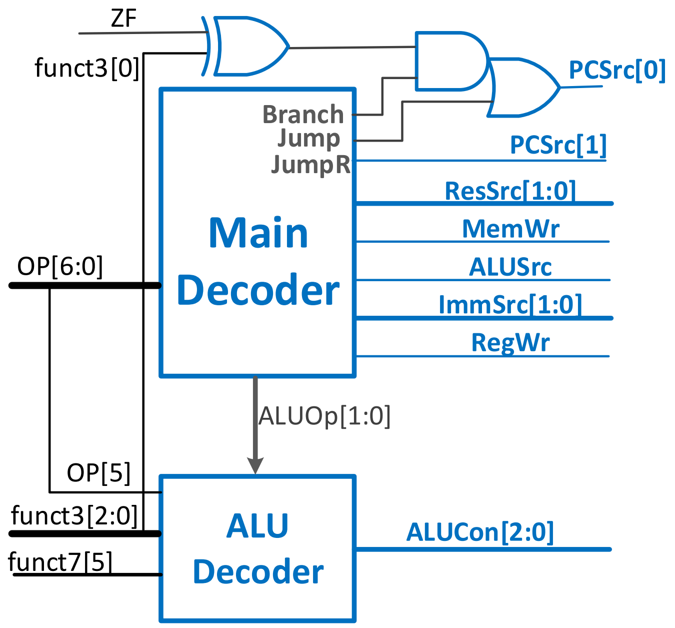
Data Path Top Module



```
module datapath (input  clk, rst, MemWr, ALUSrc, RegWr,
                 input  [1:0] PCSrc, ResSrc , ImmSrc,[2:0] ALUCon,
                 output zf ,funct7b5, [6:0] op , [2:0] funct3 );

reg [31:0] Instr , Result , RD1, RD2 , ImmExt, ScrcB , ALURes , ReadData, PC, PC4 ,
PCTarget , PCNext ;

Imem UDimem (PC[9:0] , Instr);
assign funct3  = Instr[14:12] ;
assign op      = Instr[6:0] ;
assign funct7b5 = Instr[30];

reg_file UDreg (clk, RegWr, Result , Instr[19:15], Instr [24:20], Instr[11:7], RD1 , RD2);
extend UDext (ImmSrc , Instr[31:7] , ImmExt);
mux2x1 UDmux1 (RD2 , ImmExt , ALUSrc , ScrcB );
alu UDalu (ALUCon, RD1 , ScrcB , zf , ALURes) ;
Dmem UDdmem (clk , MemWr , RD2 , ALURes[9:0] , ReadData) ;
mux4x1 UDmux2 (PC4 , ALURes , ReadData ,  , ResSrc, Result);
mux4x1 UDmux3 (PC4 , PCTarget , ALURes, , PCSrc ,PCNext) ;
pcadd4 UDpc4 (PC , PC4);
pc_u UDpc (clk , rst , PCNext , PC);
pcadd UDpcadd (PC , ImmExt , PCTarget) ;
endmodule
```

Control Unit Design

Op	funct7[5]	zf	funct3	Regwr	ImmSrc	ALuSrc	MemWr	ResSrc	PCSrc	ALUCon
3	x	x	xxx	1	00	1	0	10	10	010
19	x	x	000	1	00	1	x	01	00	010
19	x	x	110	1	00	1	x	01	00	001
35	x	x	xxx	0	01	1	1	xx	00	010
51	x	x	111	1	xx	0	x	01	00	000
99	x	1	xx0	0	10	0	x	xx	01	110
99	x	0	xx1	0	10	0	x	xx	01	110
103	0	x	xxx	1	00	1	x	00	10	010
111	xx	x	xxx	1	11	x	x	00	01	xxx



Control Unit : Instruction Decoder

Op	Branch	Jump	JumpR	Regwr	ImmSrc	ALuSrc	MemWr	ResSrc	ALUop
3	0	0	0	1	00	1	0	10	00
19	0	0	0	1	00	1	x	01	10
35	0	0	0	0	01	1	1	xx	00
51	0	0	0	1	xx	0	x	01	10
99	1	0	0	0	10	0	x	xx	01
103	0	0	1	1	00	1	x	00	00
111	0	1	0	1	11	x	x	00	xx

```
module main_dec (input [6:0] op ,
                 output branch , jump , jumpr ,
                 output reg MemWr,ALUSrc ,Regwr,
                 output reg [1:0] ResSrc , [1:0] ImmSrc , [1:0] ALUOp);

assign branch = (op==7'd99) ? 1'b1 : 1'b0 ;
assign jump   = (op==7'd111)? 1'b1 : 1'b0 ;
assign jumpr  = (op==7'd103)? 1'b1 : 1'b0 ;
```

```
always@(*)
case (op)

7'd3 : begin
Regwr  = 1'b1;
ImmSrc = 2'b00;
ALUSrc = 1'b1;
MemWr  = 1'b0;
ResSrc = 2'b10;
ALUOp  = 2'b00;
end

7'd19  : begin
Regwr  = 1'b1;
ImmSrc = 2'b00;
ALUSrc = 1'b1;
MemWr  = 1'bx;
ResSrc = 2'b01;
ALUOp  = 2'b10;
end

7'd51 : begin
Regwr  = 1'b1;
ImmSrc = 2'bxx;
ALUSrc = 1'b0;
MemWr  = 1'bx;
ResSrc = 2'b01;
ALUOp  = 2'b10;
end

7'd99 : begin
Regwr  = 1'b0;
ImmSrc = 2'b10;
ALUSrc = 1'b0;
MemWr  = 1'bx;
ResSrc = 2'bxx;
ALUOp  = 2'b01;
end

7'd103 : begin
Regwr  = 1'b1;
ImmSrc = 2'b00;
ALUSrc = 1'b1;
MemWr  = 1'bx;
ResSrc = 2'b00;
ALUOp  = 2'b00;
end

7'd111 : begin
Regwr  = 1'b1;
ImmSrc = 2'b11;
ALUSrc = 1'bx;
MemWr  = 1'bx;
ResSrc = 2'b00;
ALUOp  = 2'bxx;
end

default: begin
Regwr  = 1'bx;
ImmSrc = 2'bxx;
ALUSrc = 1'bx;
MemWr  = 1'bx;
ResSrc = 2'bxx;
ALUOp  = 2'bxx;
end

endcase
endmodule
```

Control Unit : ALU Decoder

ALUop	funct3	Op[5]	funct7[5]	ALUCon
00	xxx	x	x	010
01	xxx	x	x	110
10	111	1	0	000
10	110	1	1	001
10	000	1	0	010
10	000	1	1	110
10	000	0	x	010

```
module alu_dec (input  funct7b5 , op_5, [2:0]
funct3 , [1:0] ALUOp, output reg [2:0] ALUCon );

always@(*)
case (ALUOp)
2'b00:  ALUCon = 3'b010 ;
2'b01:  ALUCon = 3'b110 ;
2'b10:if(funct3==3'b111) ALUCon = 3'b000 ;
else if(funct3==3'b110)  ALUCon = 3'b001 ;
else if(funct3==3'b000 &  op_5 & !funct7b5)
ALUCon = 3'b010 ;
else if(funct3==3'b000 &  op_5 &  funct7b5)
ALUCon = 3'b110 ;
else if(funct3==3'b000 & !op_5) ALUCon = 3'b010;
default: ALUCon = 3'bxxx ;
endcase
endmodule
```

Control Unit: Top Module

```
module con (input  zf, funct7b5 , [6:0] op , [2:0] funct3,
            output MemWr, ALUSrc , Regwr ,
            output [1:0] PCSrc , [1:0]ResSrc ,  [1:0] ImmSrc , [2:0] ALUCon );


wire branch , jump , jumpr  ;
wire [1:0] ALUOp ;

assign PCSrc[1] = jumpr ;
assign PCSrc [0] = ( (zf ^ funct3[0]) & branch ) | jump ;

main_dec UC0 (op, branch, jump, jumpr, MemWr, ALUSrc , Regwr , ResSrc , ImmSrc , ALUOp );
alu_dec  UC1 (funct7b5 , op[5] , funct3 , ALUOp , ALUCon);

endmodule
```

Finial RISC-V

```verilog
module top_risc (input clk , rst);

wire zf ,  funct7b5 , MemWr , ALUSrc , RegWr ;
wire [6:0] op ;
wire [2:0] funct3 , ALUCon ;
wire [1:0] PCSrc , ResSrc , ImmSrc ;

con UC ( zf, funct7b5 , op , funct3, MemWr, ALUSrc , RegWr , PCSrc , ResSrc
, ImmSrc , ALUCon );
datapath UD ( clk, rst, PCSrc, ResSrc , ImmSrc, MemWr, ALUSrc, RegWr,
ALUCon, zf , funct7b5,  op , funct3 );

endmodule
```

RISC-V Test bench

RISC-V Test bench Plan

Clk#	PC	Event	Expected output	Test
1	0x00000000	Reset	PC=0	Yes
2	0x00000000	addi x2 x0 5	x2= (0+5) = 5	Yes
3	0x00000004	addi x3 x0 12	x3= (0+12) = 12	Yes
4	0x00000008	addi x7 x3 -9	x7= (12-9) = 3	Yes
5	0x0000000C	or x4 x7 x2	x4= (3 or 5) = 7	Yes
6	0x00000010	and x5 x3 x4	x5= (12 AND 7) = 4	skip
7	0x00000014	add x5 x5 x4	x5= (5+7) = 11	skip
8	0x00000018	beq x5 x7 40	not jump	skip
9	0x0000001C	beq x4 x0 8	not jump	skip
10	0x00000020	addi x5 x0 0	x5= (0 + 0) 0	skip
11	0x00000024	add x7 x4 x5	x7= (7+0) = 7	skip
12	0x00000028	sub x7 x7 x2	x7 = (7-5) = 2	skip
13	0x0000002C	sw x7 84(x3)	dmem[84d + 12d = 96d = 60h]= 2	skip
14	0x00000030	lw x2 96(x0)	Address= 96d = 60h , align to = 24d , x2= dmem[24] = 2	Yes
15	0x00000034	add x9 x2 x5	x9= (2+0) = 2	skip
16	0x00000038	jal x3 8	x3 = h3C , PC = 40	skip
17	0x0000003C	addi x2 x0 1	not executed	skip
18	0x00000040	add x2 x2 x9	x2 = (2+2)=4	skip
19	0x00000044	sw x2 32(x3)	Address= 32d+3Ch=5C , align to = 23d , dmem[23]=4	Yes
20	0x00000048	beq x2 x2 0	Stop	

RISC-V Test bench

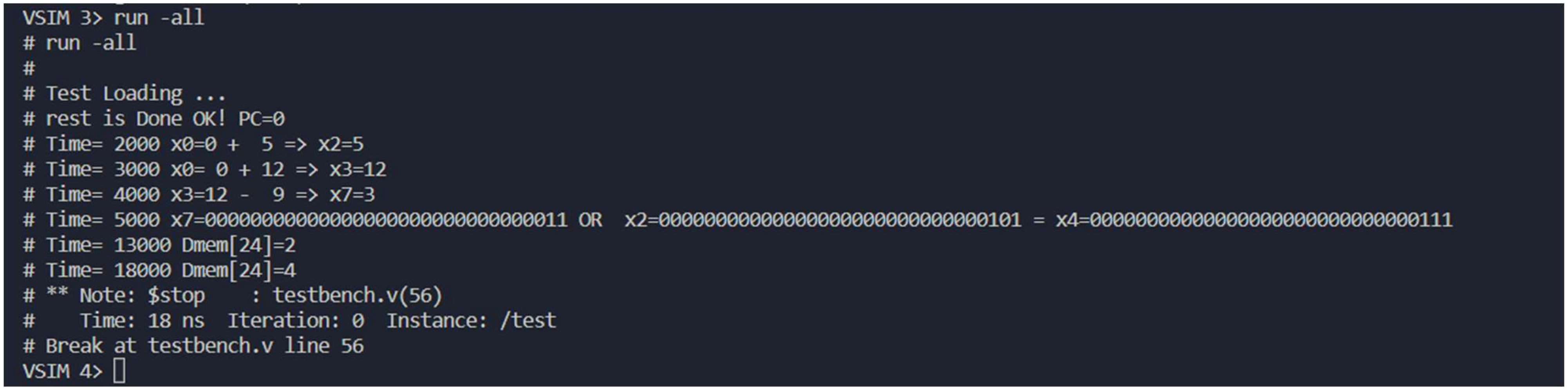
```
`timescale 1ps/1ps
`include "top_risc.v"
`include "control.v"
`include "con_alu_dec.v"
`include "con_main_dec.v"
`include "datapath.v"
`include "alu.v"
`include "pc.v"
`include "reg_file.v"
`include "Dmem.v"
`include "Imem.v"
`include "extend.v"
`include "mux2x1.v"
`include "mux4x1.v"
`include "pcadd.v"
`include "pcadd4.v"
```

Clk#	Expected output	Test
1	PC=0	Yes
2	x2= (0+5) = 5	Yes
3	x3= (0+12) = 12	Yes
4	x7= (12-9) = 3	Yes
5	x4= (3 or 5) = 7	Yes
14	Address= 96d = 60h , align to = 24d , x2= dmem[24] = 2	Yes
19	Address= 32d+3Ch=5C , align to = 23d , dmem[23]=4	Yes
20	Stop	

```verilog
module test ();
parameter Tclk =1000; // 1ns
reg clk = 1'b0 , rst;
top_risc DUT (clk , rst );

initial
forever #(Tclk/2) clk = ~ clk ;

initial begin
$display ("Test Loading ... ");
rst = 1'b1 ; //1
#Tclk
if (DUT.UD.UDpc.pc ==0) $display ("rest is Done OK! PC=0 ");
else $display ("rest Failed!");
rst = 1'b0 ; //2
#Tclk
$display ("Time= %0d x0=%0d +  5 => x2=%0d",$time, DUT.UD.UDreg.regs[0],
DUT.UD.UDreg.regs[2]); //3
#Tclk
$display ("Time= %0d x0= %0d + 12 => x3=%0d",$time, DUT.UD.UDreg.regs[0],
DUT.UD.UDreg.regs[3]); //4
```

```
#Tclk
$display ("Time= %0d x3=%0d -  9 => x7=%0d",$time, DUT.UD.UDreg.regs[3],
DUT.UD.UDreg.regs[7]); //5
#Tclk
$display ("Time= %0d x7=%b OR  x2=%b = x4=%b",$time, DUT.UD.UDreg.regs[7],
DUT.UD.UDreg.regs[2], DUT.UD.UDreg.regs[4]); // 6 to 14
#(8*Tclk)
$display ("Time= %0d Dmem[24]=%0d",$time, DUT.UD.UDdmem.mem[24]); // 15 to 19
#(5*Tclk)
$display ("Time= %0d Dmem[24]=%0d",$time, DUT.UD.UDdmem.mem[23]);
$stop ; end
endmodule
```

RISC-V Test bench Result



```
VSIM 3> run -all
# run -all
#
# Test Loading ...
# rest is Done OK! PC=0
# Time= 2000 x0=0 +  5 => x2=5
# Time= 3000 x0= 0 + 12 => x3=12
# Time= 4000 x3=12 -  9 => x7=3
# Time= 5000 x7=00000000000000000000000000000011 OR  x2=00000000000000000000000000000101 = x4=00000000000000000000000000000111
# Time= 13000 Dmem[24]=2
# Time= 18000 Dmem[24]=4
# ** Note: $stop    : testbench.v(56)
#    Time: 18 ns  Iteration: 0  Instance: /test
# Break at testbench.v line 56
VSIM 4> 
```