Discussion 4

Wei-Hsiang Ma

2009/10/06

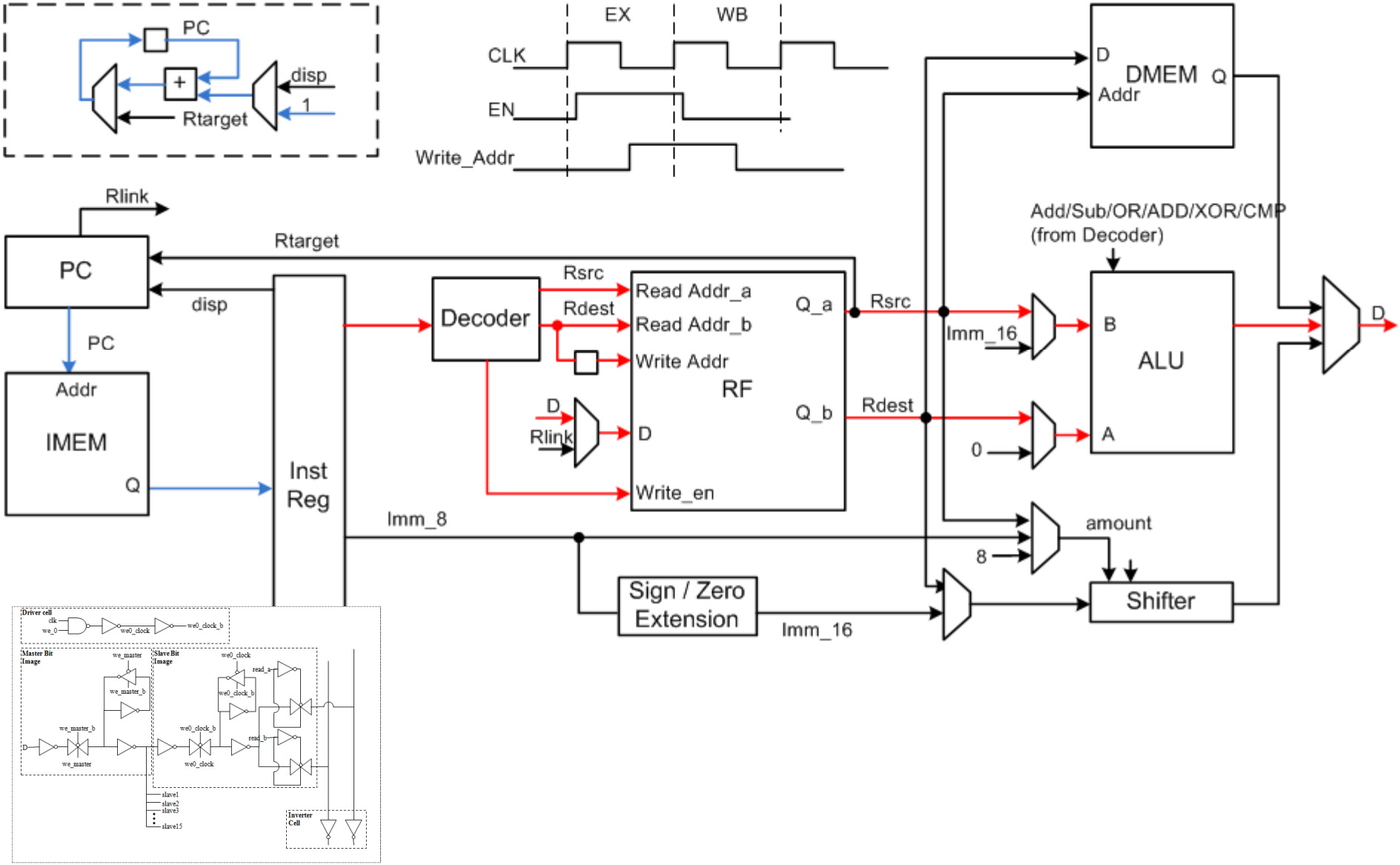
Today

- Baseline Architecture
- Synthesis Flow
- Verilog

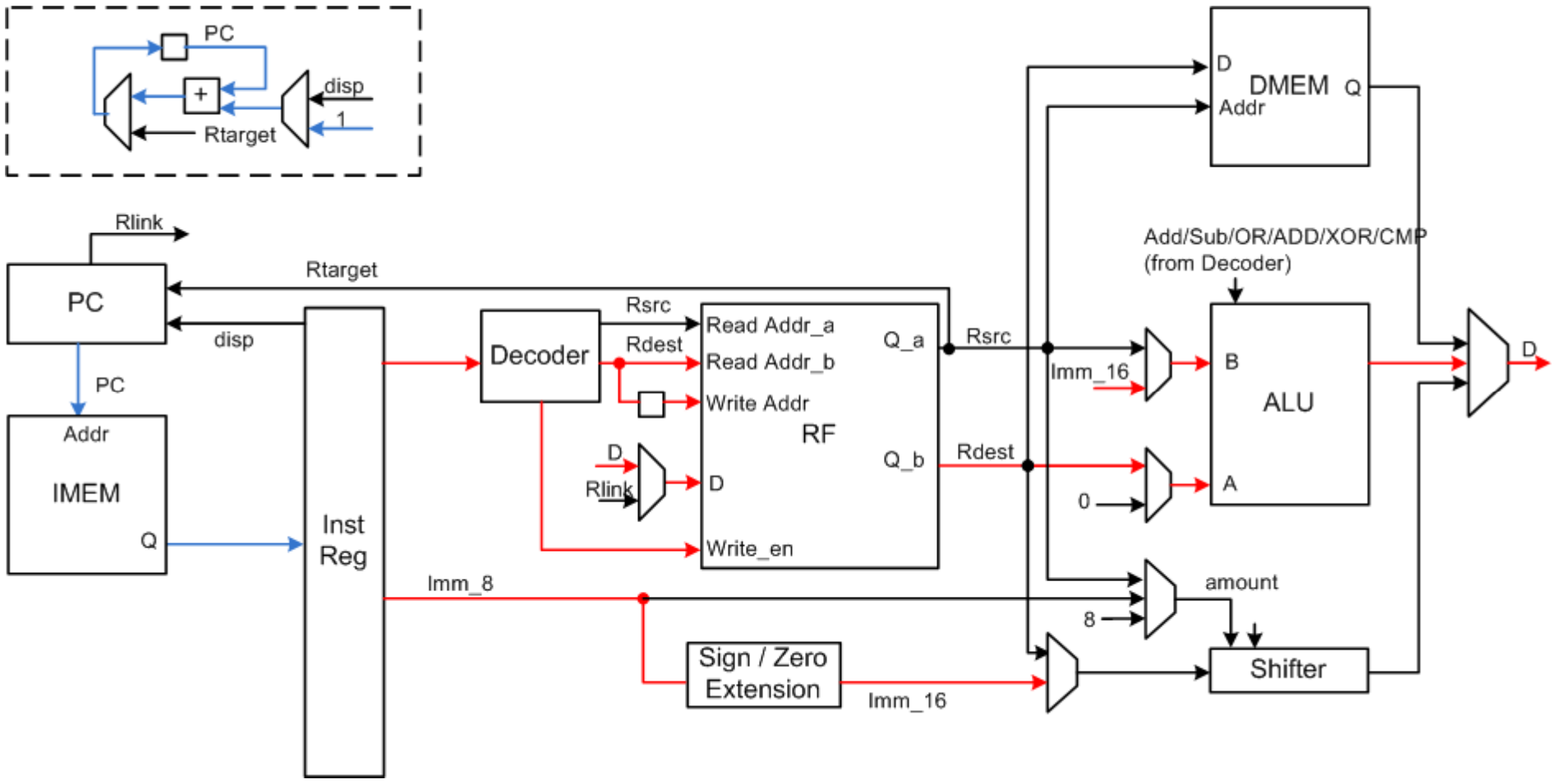
Architecture

- The architecture I am going to show you is the baseline architecture for EECS427
- Feel free to modify it, but it has to be working
- The slides are newly made by your GSI, if you see something wrong, please tell me.
- Hope you have a better understanding about the architecture.

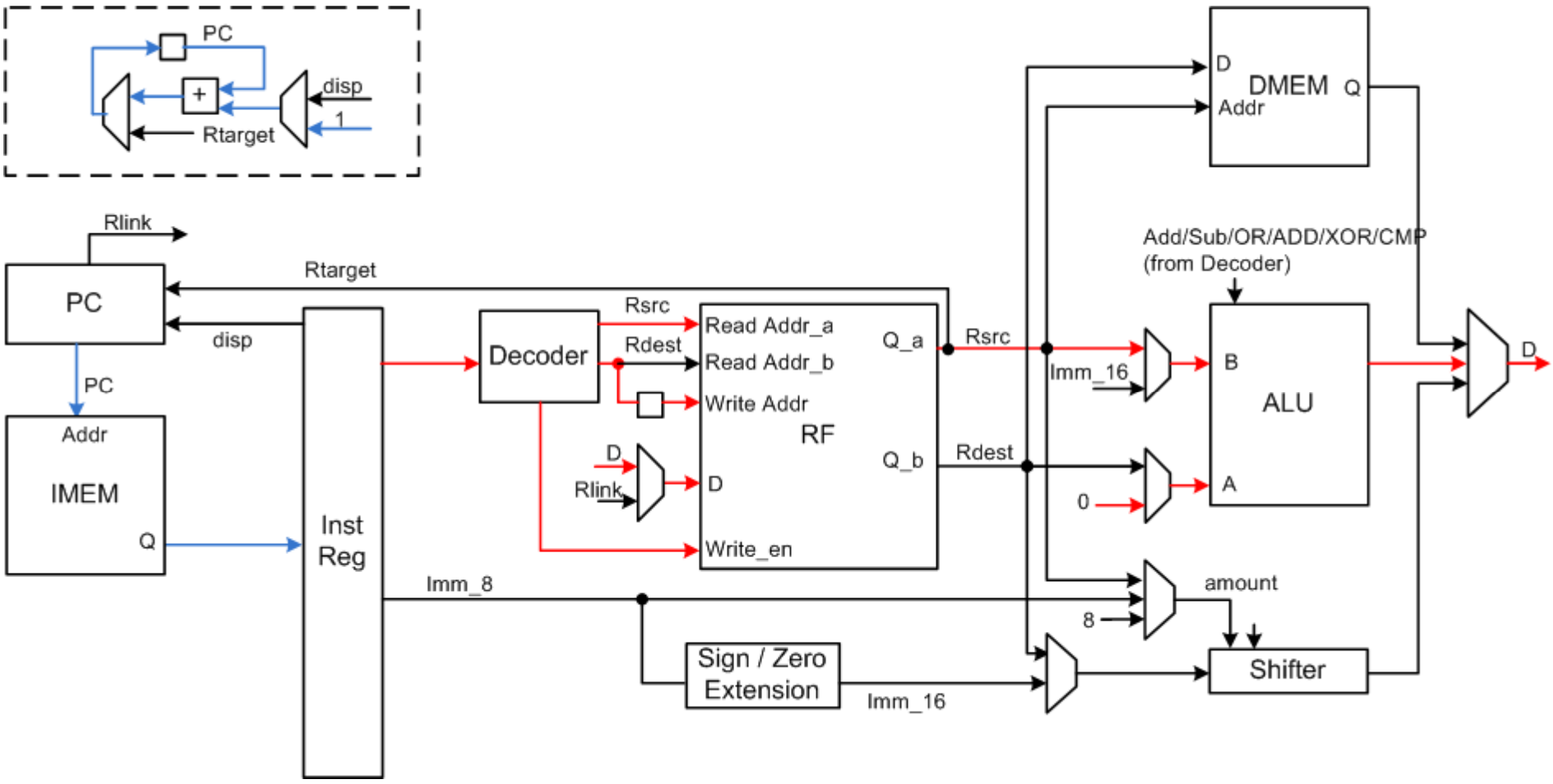
ADD/SUB/CMP/AND/OR/XOR



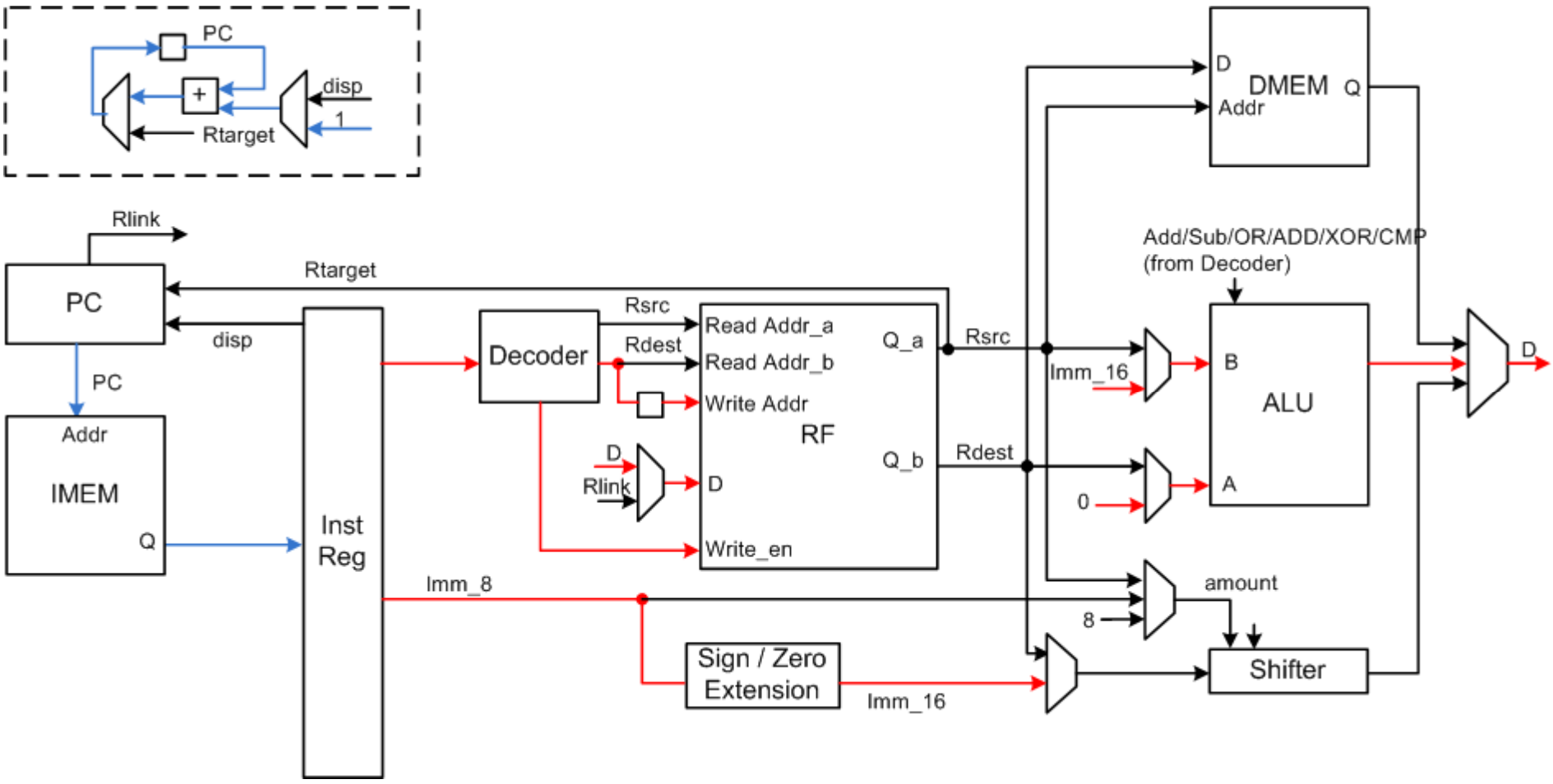
ADDI/SUBI/CMPI/ANDI/ORI/XORI



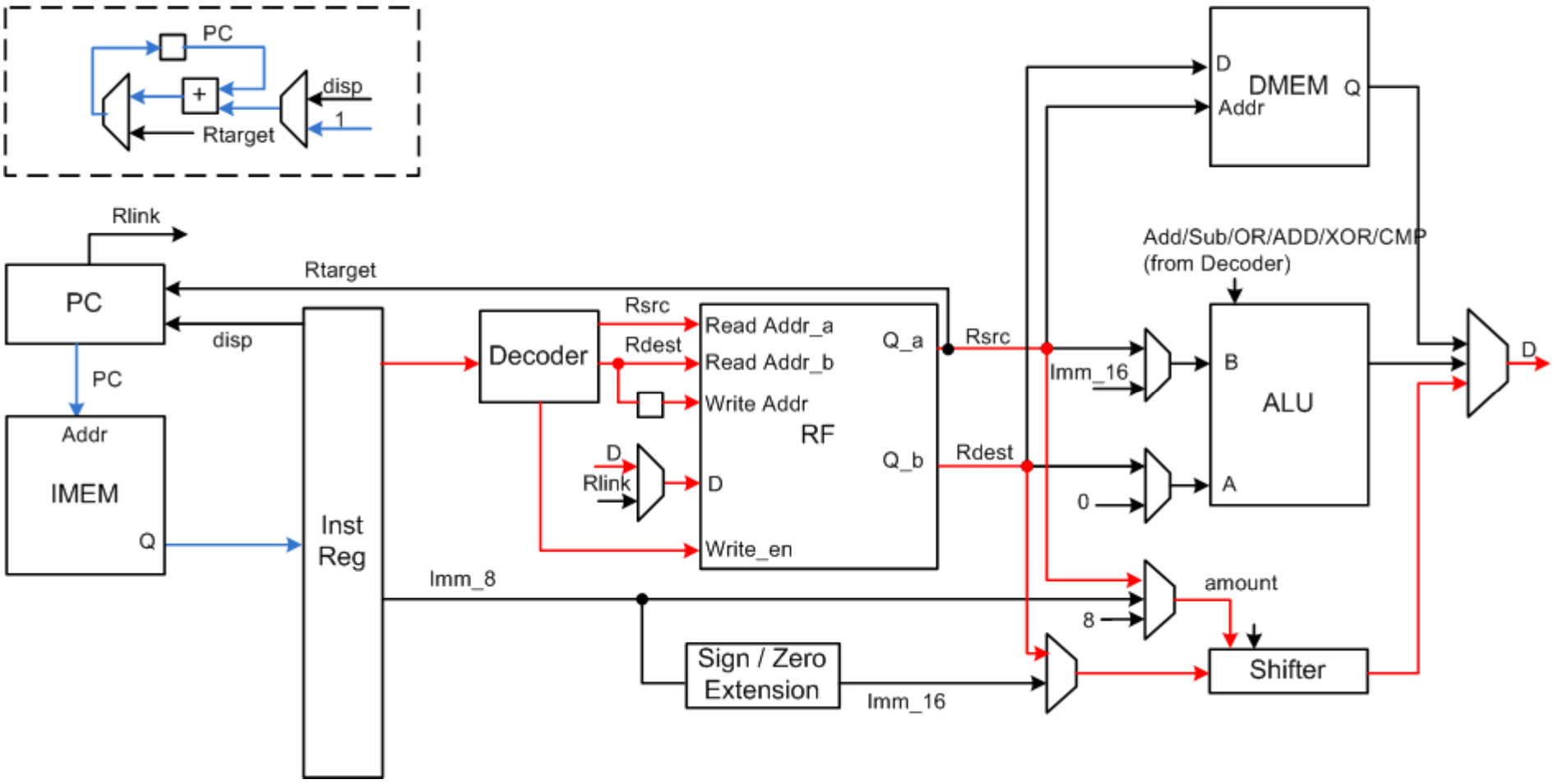
MOV


PC
+
disp
1
Rtarget
Rlink
PC
Rtarget
disp
PC
Addr
IMEM
Q
Inst
Reg
Decoder
Rsrc
Rdest
Read Addr_a
Read Addr_b
Write Addr
RF
Write_en
D
Rlink
D
Q_a
Q_b
Rsrc
Rdest
Imm_8
Sign / Zero
Extension
Imm_16
D
DMEM
Q
Addr
Add/Sub/OR/ADD/XOR/CMP
(from Decoder)
Imm_16
B
ALU
A
0
8
amount
Shifter
D

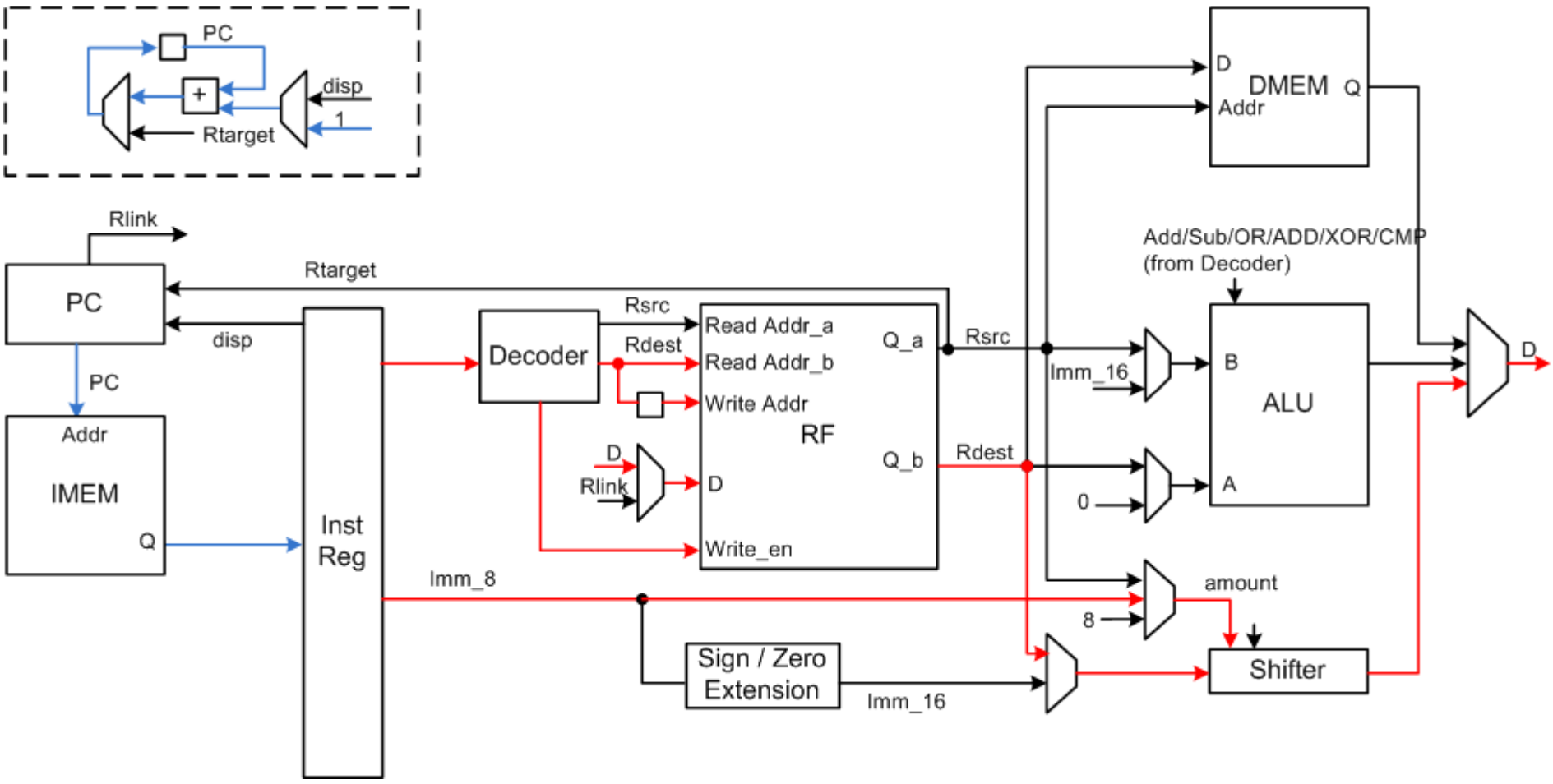
MOVI


PC
+
disp
1
Rtarget
Rlink
PC
Rtarget
disp
PC
Addr
IMEM
Q
Inst
Reg
Decoder
Rsrc
Rdest
Read Addr_a
Read Addr_b
Write Addr
RF
Write_en
D
Rlink
D
Q_a
Q_b
Rsrc
Rdest
Imm_8
Sign / Zero
Extension
Imm_16
D
DMEM
Q
Addr
Add/Sub/OR/ADD/XOR/CMP
(from Decoder)
Imm_16
B
ALU
A
0
8
amount
Shifter
D

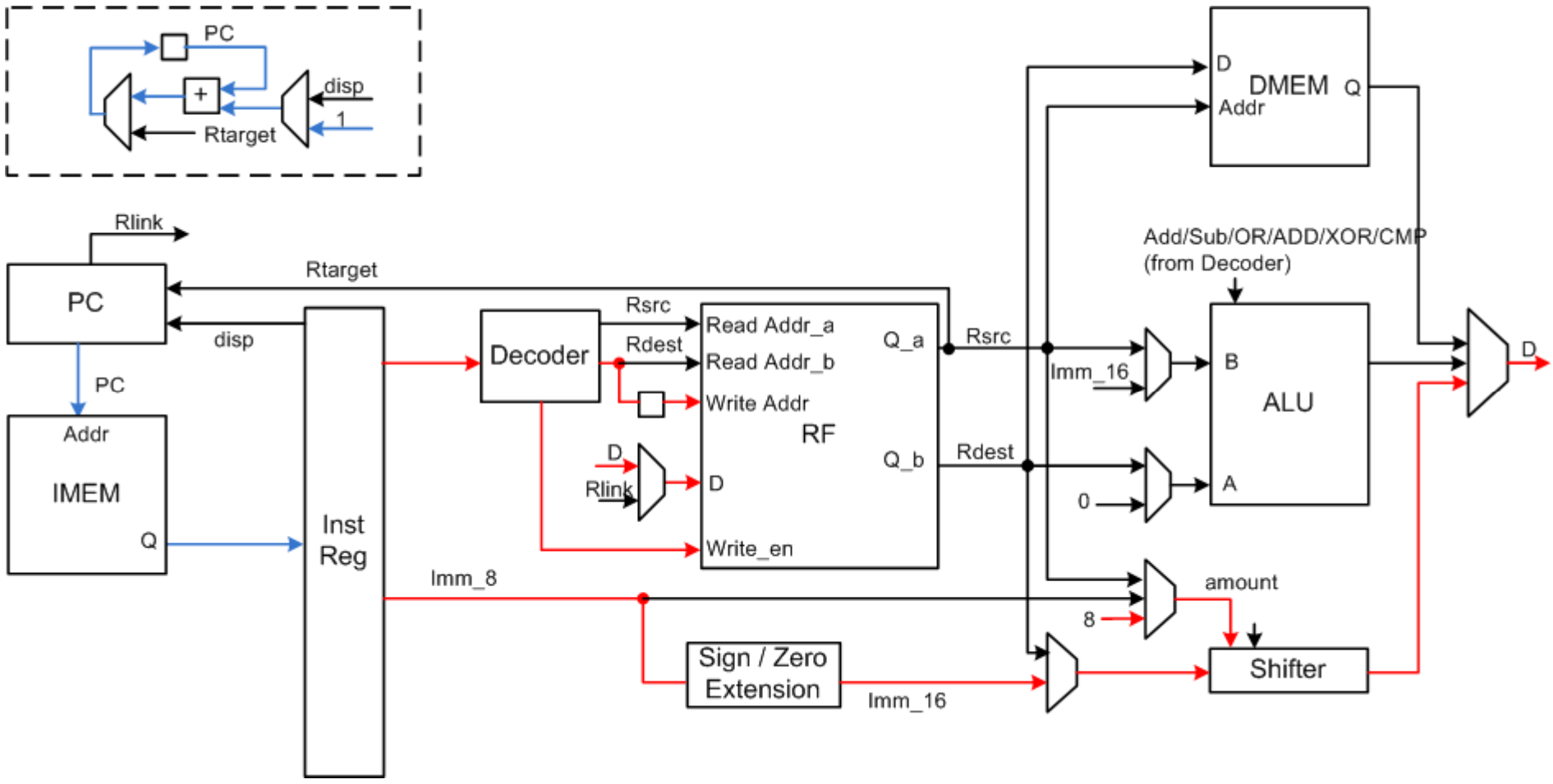
LSH/ASHU


PC
+
disp
1
Rtarget
Rlink
PC
Rtarget
disp
PC
Addr
IMEM
Q
Inst
Reg
Decoder
Rsrc
Rdest
Read Addr_a
Read Addr_b
Write Addr
RF
Q_a
Q_b
D
Rlink
D
Write_en
Imm_8
Sign / Zero
Extension
Imm_16
Rsrc
Rdest
D
DMEM
Q
Addr
Add/Sub/OR/ADD/XOR/CMP
(from Decoder)
Imm_16
B
ALU
A
0
8
amount
Shifter
D

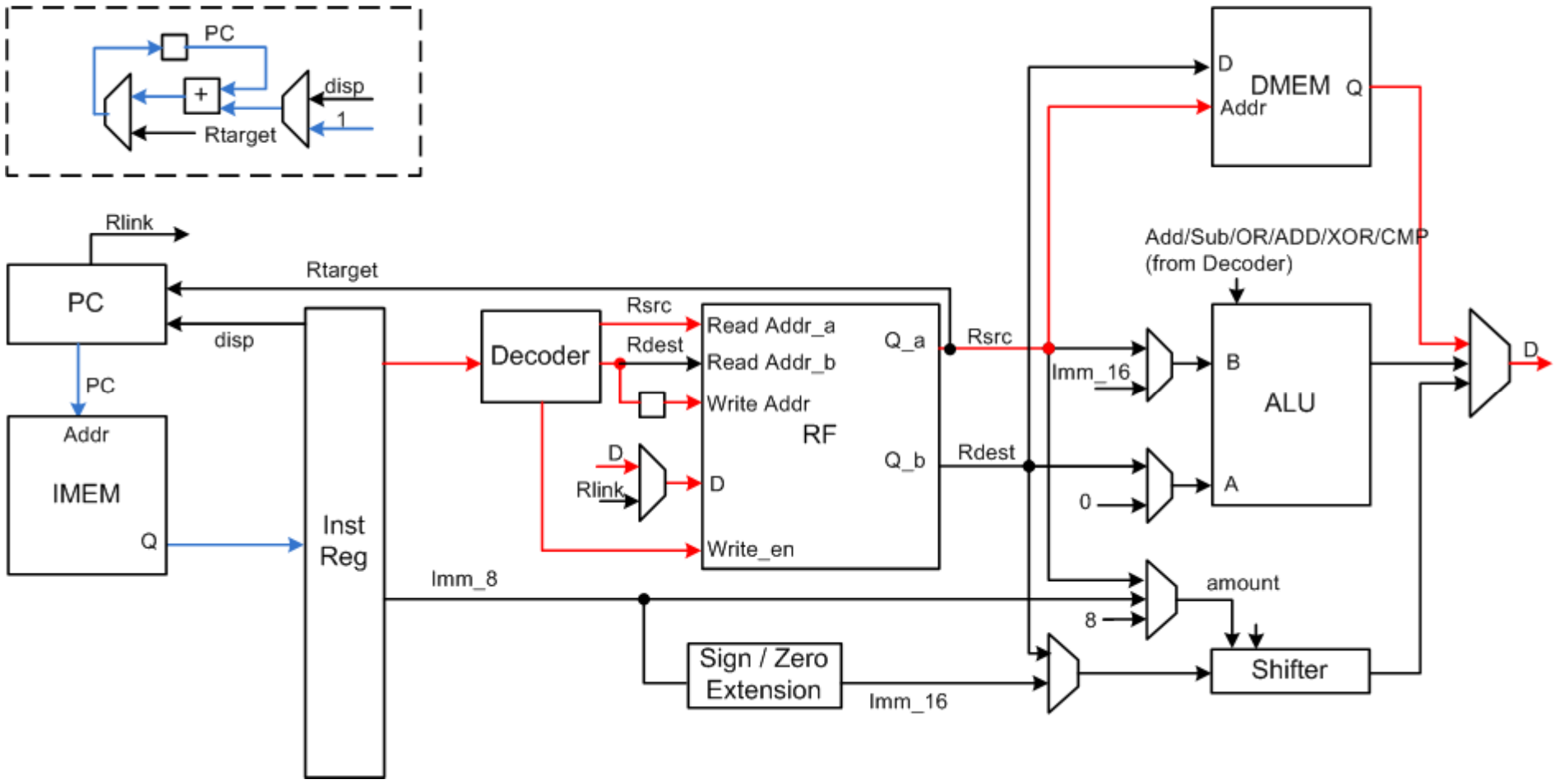
LSHI/ASHUI


PC
+
disp
1
Rtarget
Rlink
PC
Rtarget
disp
PC
Addr
IMEM
Q
Inst
Reg
Decoder
Rsrc
Rdest
Read Addr_a
Read Addr_b
Write Addr
RF
Q_a
Q_b
D
Rlink
D
Write_en
Imm_8
Sign / Zero
Extension
Imm_16
Rsrc
Rdest
D
DMEM
Q
Addr
Add/Sub/OR/ADD/XOR/CMP
(from Decoder)
Imm_16
B
ALU
A
0
8
amount
Shifter
D

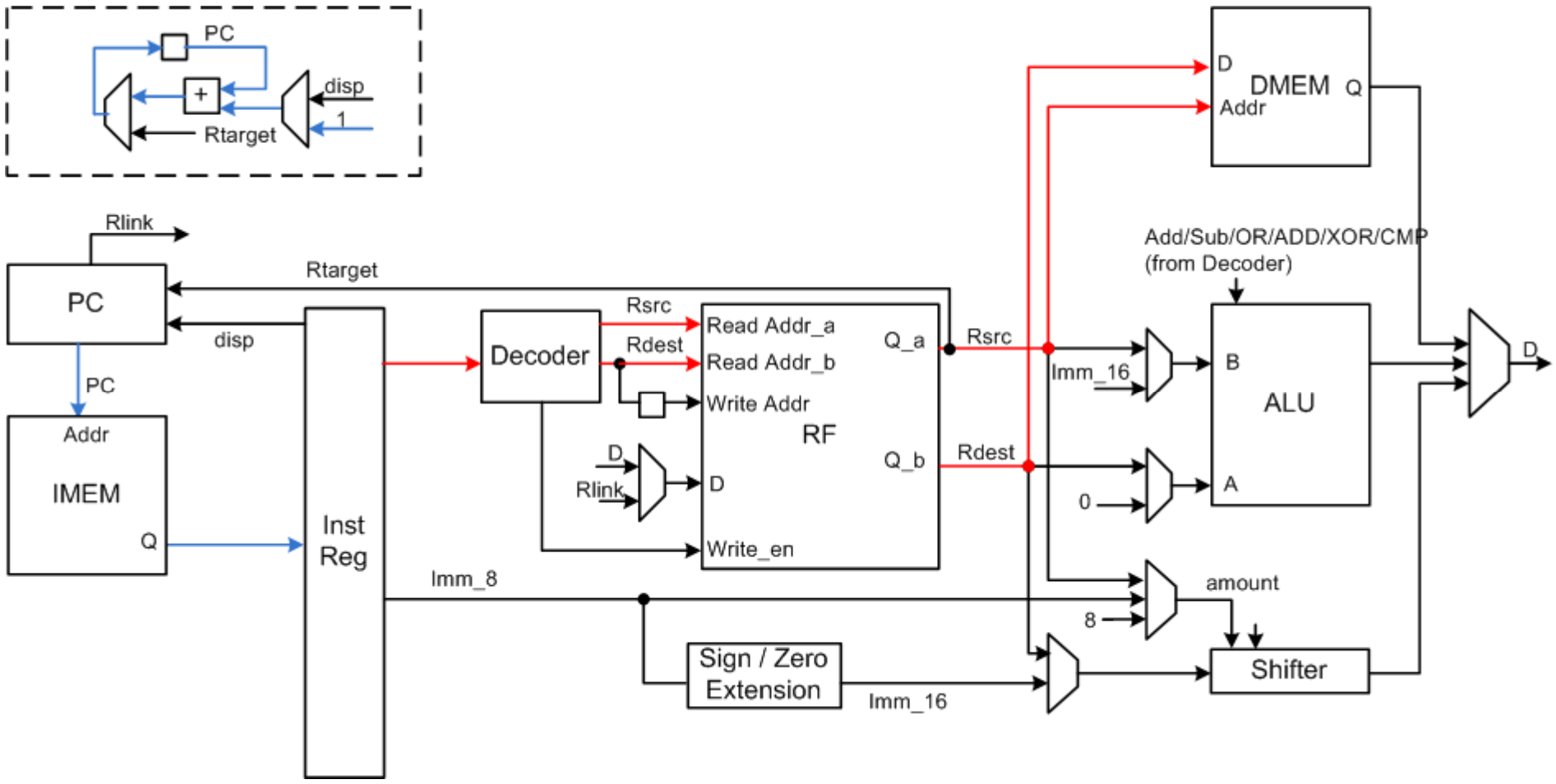
LUI


PC
+
disp
1
Rtarget
Rlink
PC
Rtarget
disp
PC
Addr
IMEM
Q
Inst
Reg
Decoder
Rsrc
Rdest
Read Addr_a
Read Addr_b
Write Addr
RF
Write_en
D
Rlink
D
Q_a
Q_b
Rsrc
Rdest
Imm_8
Sign / Zero
Extension
Imm_16
D
DMEM
Q
Addr
Add/Sub/OR/ADD/XOR/CMP
(from Decoder)
Imm_16
B
ALU
A
0
8
amount
Shifter
D

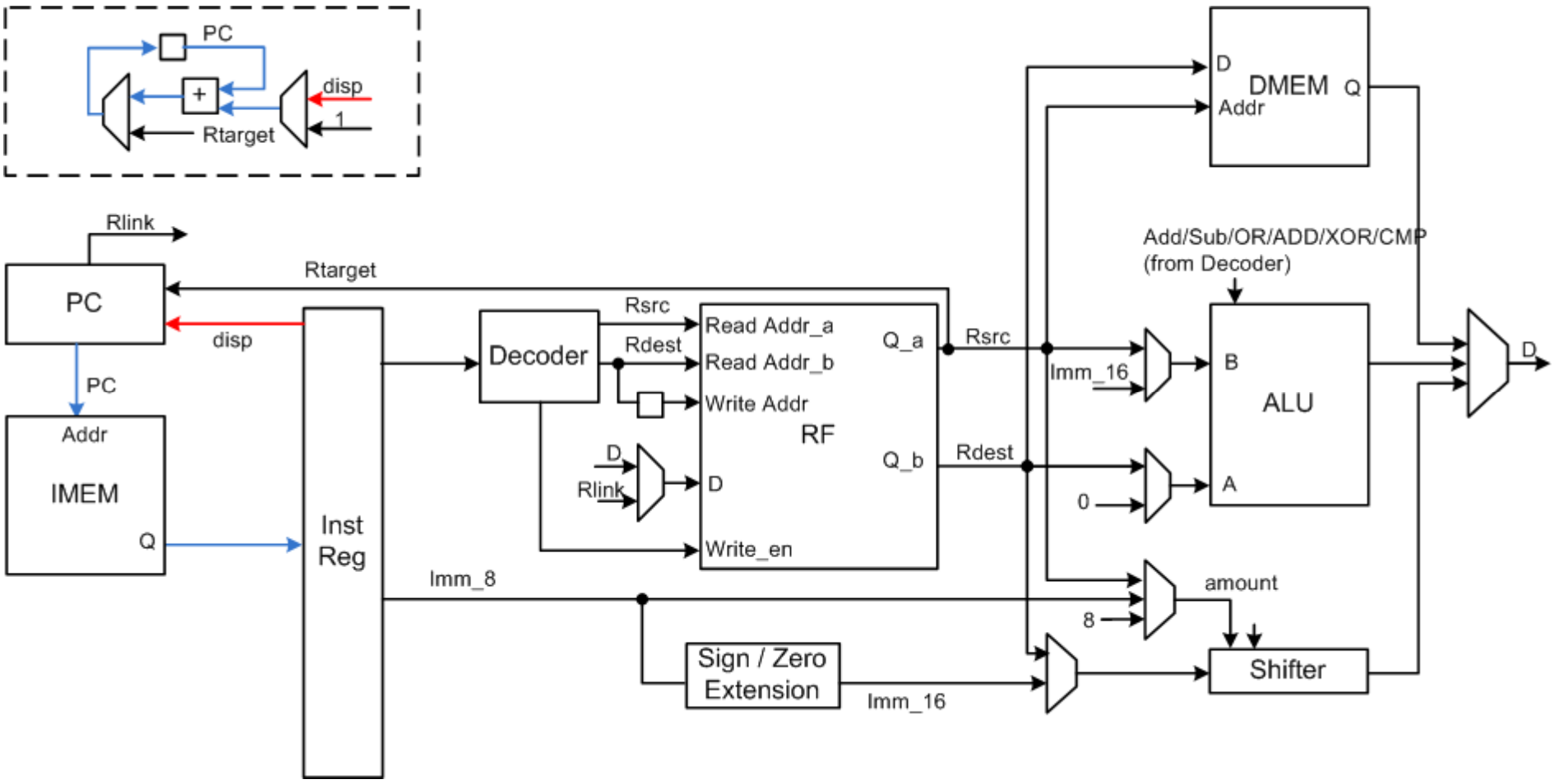
LOAD


PC
+
disp
1
Rtarget
Rlink
PC
Rtarget
disp
PC
Addr
IMEM
Q
Inst
Reg
Decoder
Rsrc
Rdest
Read Addr_a
Read Addr_b
Write Addr
RF
Write_en
Q_a
Q_b
D
Rlink
Rsrc
Rdest
Imm_8
Sign / Zero
Extension
Imm_16
D
DMEM
Q
Addr
Add/Sub/OR/ADD/XOR/CMP
(from Decoder)
Imm_16
B
ALU
A
0
8
amount
Shifter
D

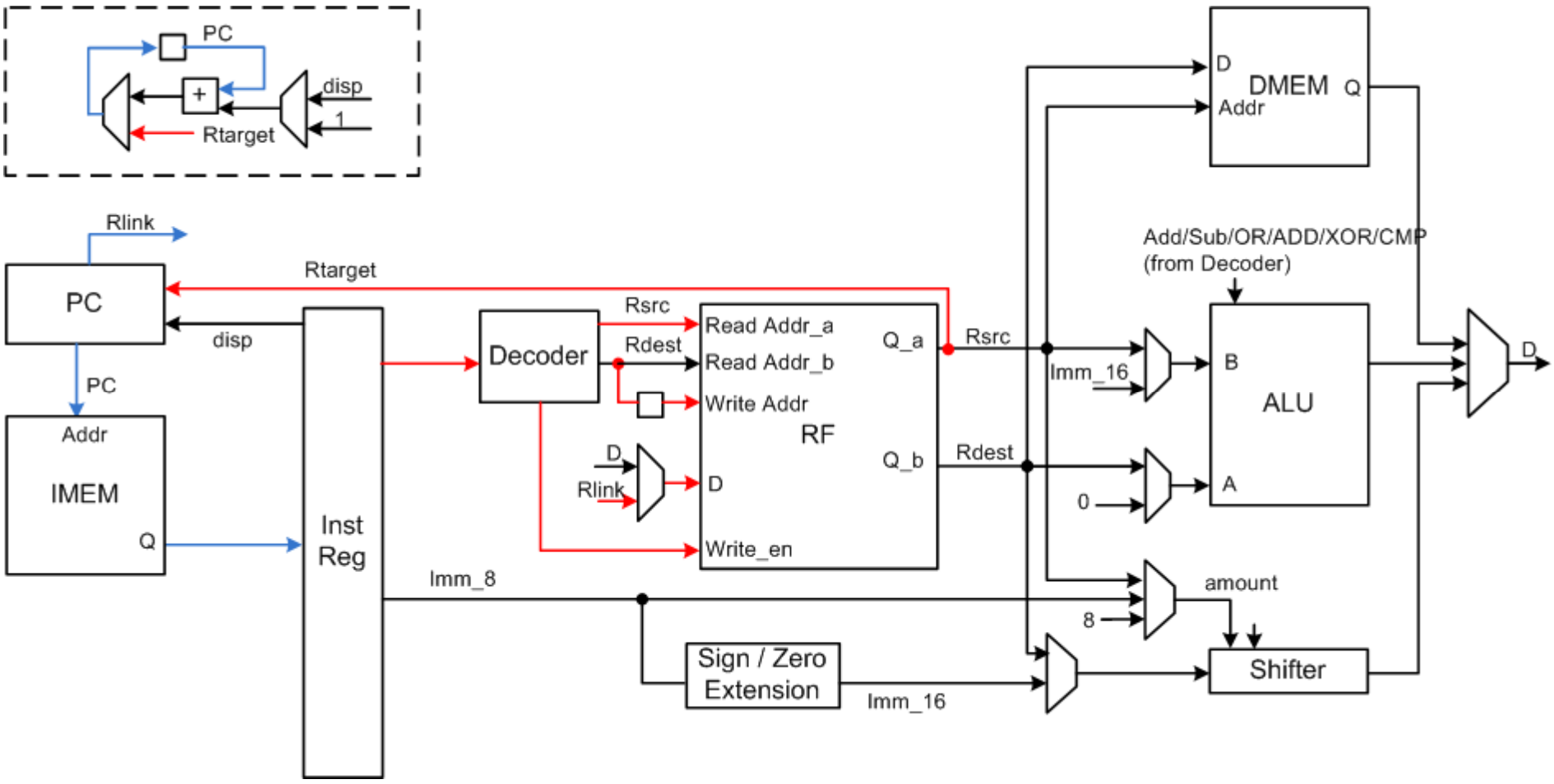
STOR


PC
+
disp
1
Rtarget
Rlink
PC
Rtarget
disp
PC
Addr
IMEM
Q
Inst
Reg
Decoder
Rsrc
Rdest
Read Addr_a
Read Addr_b
Write Addr
RF
Q_a
Q_b
D
Rlink
D
Write_en
Imm_8
Sign / Zero
Extension
Imm_16
Rsrc
Rdest
D
DMEM
Q
Addr
Add/Sub/OR/ADD/XOR/CMP
(from Decoder)
Imm_16
B
ALU
A
0
8
amount
Shifter
D

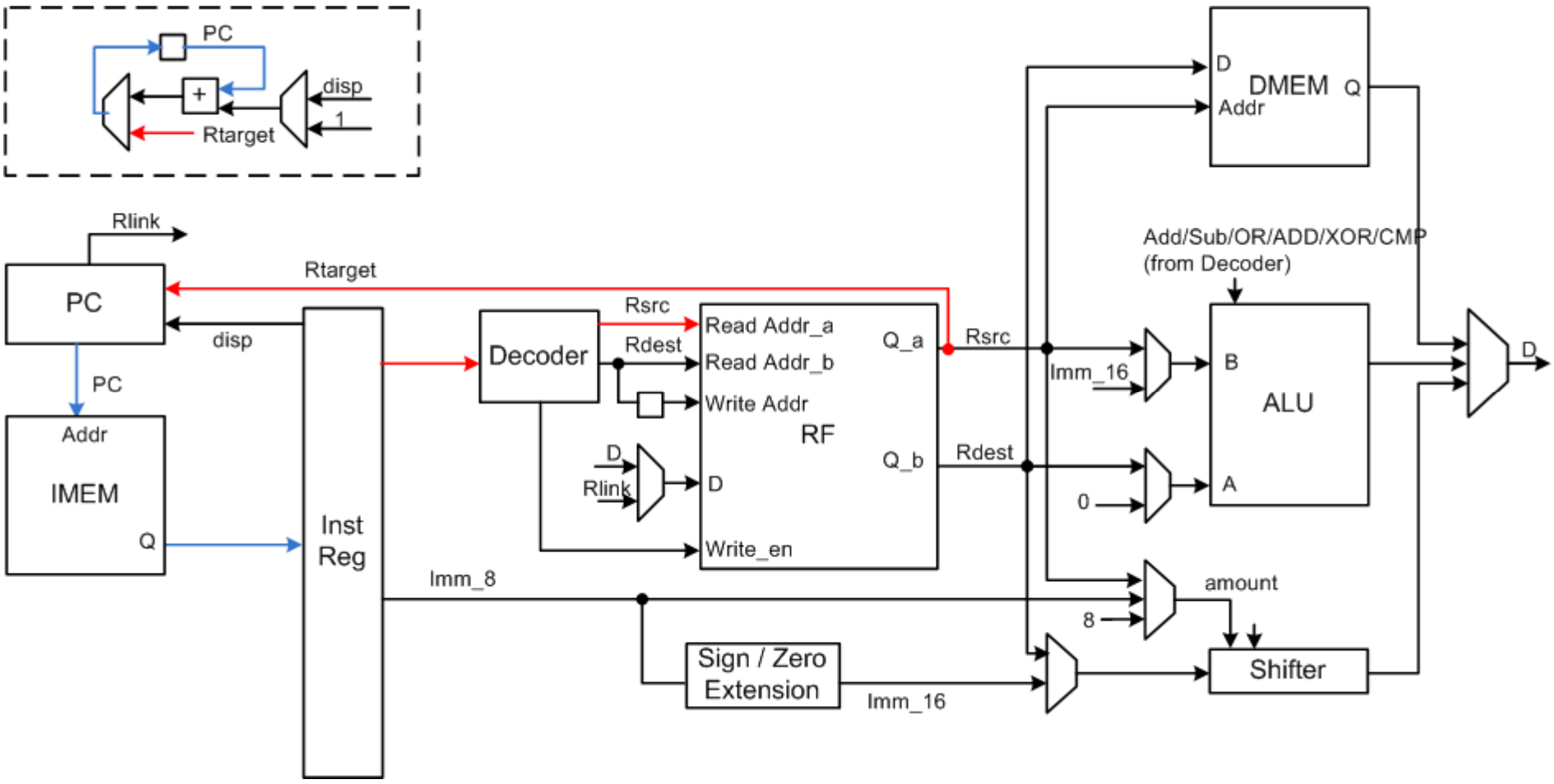
Bcond


PC
+
disp
1
Rtarget
Rlink
PC
Rtarget
disp
PC
Addr
IMEM
Q
Inst
Reg
Decoder
Rsrc
Rdest
Read Addr_a
Read Addr_b
Write Addr
RF
Q_a
Q_b
D
Rlink
D
Write_en
Rsrc
Rdest
Imm_8
Sign / Zero
Extension
Imm_16
D
DMEM
Q
Addr
Add/Sub/OR/ADD/XOR/CMP
(from Decoder)
Imm_16
B
ALU
A
0
8
amount
Shifter
D

Jcond


PC
+
disp
1
Rtarget
Rlink
PC
Rtarget
disp
PC
Addr
IMEM
Q
Inst
Reg
Decoder
Rsrc
Rdest
Read Addr_a
Read Addr_b
Write Addr
RF
Q_a
Q_b
D
Rlink
D
Write_en
Rsrc
Rdest
Imm_8
Sign / Zero
Extension
Imm_16
D
DMEM
Q
Addr
Add/Sub/OR/ADD/XOR/CMP
(from Decoder)
Imm_16
B
ALU
A
0
8
amount
Shifter
D

JAL


PC
+
disp
1
Rtarget
Rlink
PC
Rtarget
disp
PC
Addr
IMEM
Q
Inst
Reg
Decoder
Rsrc
Rdest
Read Addr_a
Read Addr_b
Write Addr
RF
Q_a
Q_b
D
Rlink
D
Write_en
Rsrc
Rdest
Imm_8
Sign / Zero
Extension
Imm_16
D
DMEM
Q
Addr
Add/Sub/OR/ADD/XOR/CMP
(from Decoder)
Imm_16
B
ALU
A
0
8
amount
Shifter
D

What else?

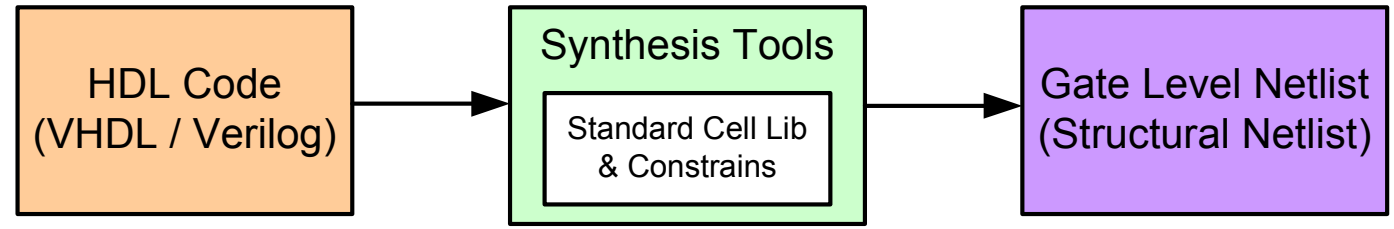
- Data Hazard

  Inst0: r1 + r3 => r1

  Inst1: r1 + r5 => r1

- Scan
  - Scan out some intermediate data
  - Scan in instructions I-MEM
- Global Reset
  - How does your processor start?

Synthesis Flow



add2.v

```
module add2 (Y, A, B);
input [1:0] A, B;
output [1:0] Y;

assign Y[1:0] = A[1:0] + B[1:0];

endmodule
```

Std Cells:

INVX1TR
INVX2TR
AND2X1TR
AND2X2TR
OR2X1TR
OR2X2TR

Constrains:

set input delay
set output delay
set output load
set clock period

add2.nl.v

```
module add2 (Y, A, B);
input [1:0] A, B;
output [1:0] Y;

XORX2X1TR I1 (.Y(Y[0]), .A(A[0]), .B(B[0]));
AND2X2TR I0 (.Y(C0), .A(A[0]), .B(B[0]));
XOR3X2TR I2 (.Y(Y[1]), .A(A[1]), .B(B[1]), . C(C0));

endmodule
```

- Synthesis tools will choose the size of the std cell according to the design constrains

Verilog (Sequential Logic)

DFF without RESET

```
module DFF (Q, Q_b, D, CK);
input D, CK;
output Q, Q_b;

reg Q, Q_b;

always @(posedge CK)
begin
Q <= D;
Q_b <= ~D;
end

endmodule
```

DFF with Sync RESET

```
module DFF (Q, Q_b, D, CK, R);
input D, CK,R;
output Q, Q_b;

reg Q, Q_b;

always @(posedge CK)
begin
  if (R == 1'b0)
  begin
  Q <= 1'b0;
  Q_b <= 1'b1;
  end

  else
  begin
  Q <= D;
  Q_b <= ~D;
  end
end

endmodule
```

DFF with Async RESET

```
module DFF (Q, Q_b, D, CK, R);
input D, CK,R;
output Q, Q_b;

reg Q, Q_b;

always @(posedge CK or negedge R)
begin
  if (R == 1'b0)
  begin
  Q <= 1'b0;
  Q_b <= 1'b1;
  end

  else
  begin
  Q <= D;
  Q_b <= ~D;
  end
end

endmodule
```

- Tips:
  - Declare Q, Q_b as register
  - Use Non-Blocking assignment “<=“
  - All non-blocking assignments happen at the same time

Verilog (Combinational Logic)

Use Assign	**Use Always block**
module mult (Y, A, B); input [1:0] A, B; output [3:0] Y; **assign Y = A * B;** endmodule	module mult (Y, A, B); input [1:0] A, B; output [3:0] Y; **reg [3:0] Y;** **always@ *** **begin** **Y = A * B;** **end** endmodule

- Tips for using always block
  - Declare Y as register, even though it is not.
  - Use Blocking assignment “=“
  - Blocking assignments are processed in order (C-style)

Verilog (Seq + Com Logic)

```
module mult (Y, A, B, CK);
input [1:0] A, B;
input CK;
output [3:0] Y;

reg [3:0] Y;
wire [3:0] temp;

assign temp = A * B;

always @(posedge CK)
begin
Y <= temp;
end
endmodule
```

FSM

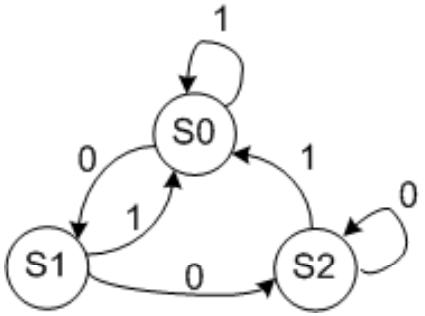
```
module FSM (A, clk, reset, state);

input A;
input clk, reset;
output [1:0] state;

reg [1:0] state;

always@(posedge clk)
begin
  if (reset == 1'b0)
  begin
        state <= 2'b00;
  end
  else
  begin
        case (state)
        2'b00:
        begin
                if (A == 1'b1) state <=2'b00;
                else state <= 2'b01;
        end
        2'b01:
        begin
                if (A == 1'b1) state <=2'b00;
                else state <= 2'b10;
        end
        2'b10:
        begin
                if (A == 1'b1) state <=2'b00;
                else state <= 2'b10;
        end
        endcase
  end
end

endmodule
```



FSM

```
module FSM (A, clk, reset, state);

input A;
input clk, reset;
output [1:0] state;

reg [1:0] state;

always@(posedge clk)
begin
  if (reset == 1'b0)
  begin
        state <= 2'b00;
  end
  else
  begin
        case (state)
        2'b00:
        begin
                if (A == 1'b1) state <=2'b00;
                else state <= 2'b01;
        end
        2'b01:
        begin
                if (A == 1'b1) state <=2'b00;
                else state <= 2'b10;
        end
        2'b10:
        begin
                if (A == 1'b1) state <=2'b00;
                else state <= 2'b10;
        end
        endcase
  end
end

endmodule
```

```
reg we_en0, we_en1, we_en2;
always@ *
begin
        case (state)
        2'b00:
        begin
                we_en0 = 1'b1;
                we_en1= 1'b0;
                we_en2= 1'b1;
        end
        2'b01:
        begin
                we_en0 = 1'b1;
                we_en1= 1'b0;
                we_en2= 1'b0;
        end
        2'b10:
        begin
                we_en0= 1'b0;
                we_en1= 1'b0;
                we_en2= 1'b1;
        end
        endcase
end
```

Others

- While, forever, initial,… => only used in testbench

Verilog Simulation

- Create “**functional**” view
- Paste your behavioral code into the “**functional**” view
- Create an empty “**schematic**” view
- When doing NCverilog simulation, choose “**functional**” instead of “**schematic**”
- Modify your testbench and simulate it

Demo

- Verilog simulation