supplementary material on the Tiny Register Machine Implementation

not be covered in class

4.3 BEHIND THE SCENES OF ACTIVE CELLS

TRM Architectural State

- PC
- 8 registers
- flag registers
- Memory (configurable)
  - nK * 36 bits instruction memory (1k = 1024)
  - nk * 32 bits data memory

PL vs. HDL

Programming Language

- Sequential execution
- No notion of time

```
var a,b,c: integer;

a := 1;
b := 2;
c := a + b;
```

unknown mapping to machine cycles

Hardware Description Language

- Continuous execution (combinational logic)

```
wire [31:0] a,b,c;

assign a=1;
assign b=2;
assign c=a+b;
```

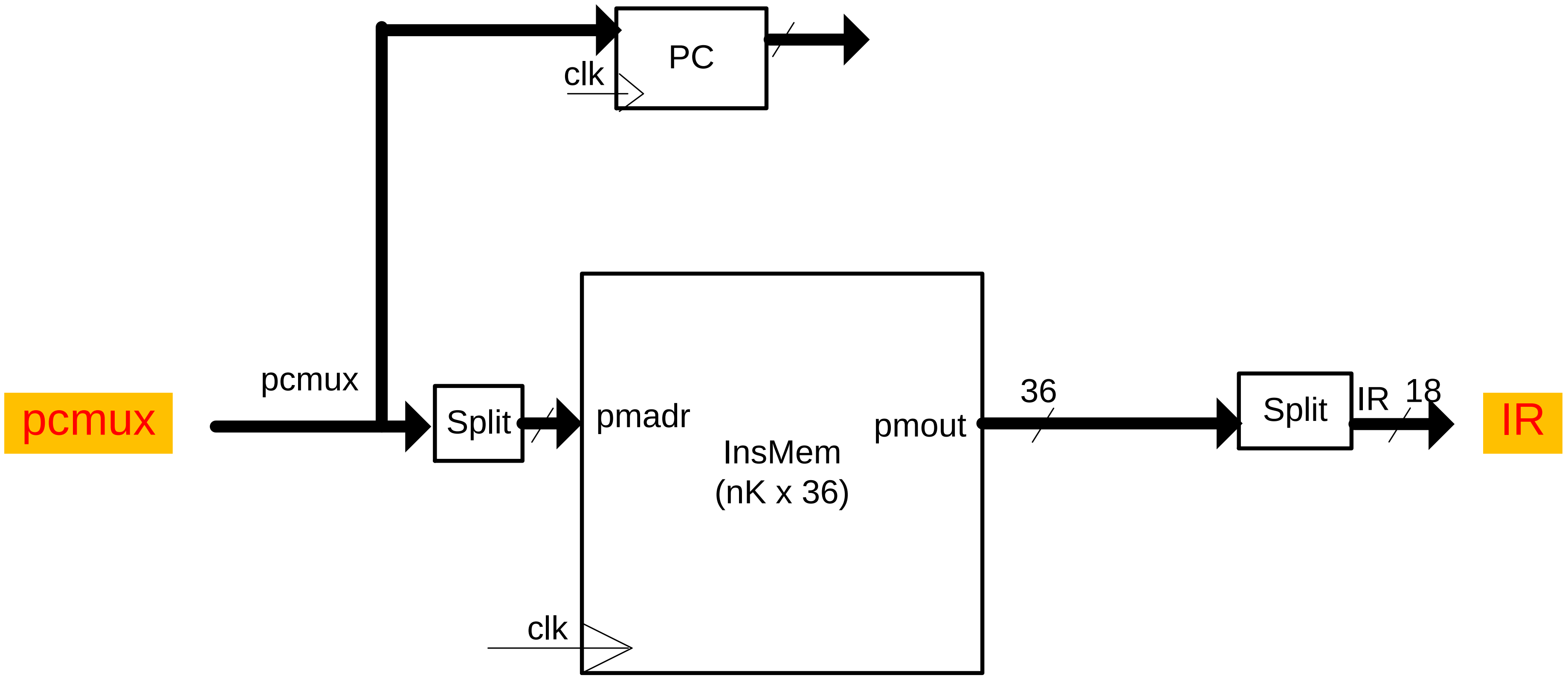
no memory associated

- Synchronous execution (register transfer)

```
reg [31:0] a,b,c;

always @ (posedge clk)
begin
  a <= 1;
  b <= 2;
  c <= a+b;
end;
```

synchronous at rising edge of the clock

Single-Cycle Datapath:
arithmetical logical instruction fetch



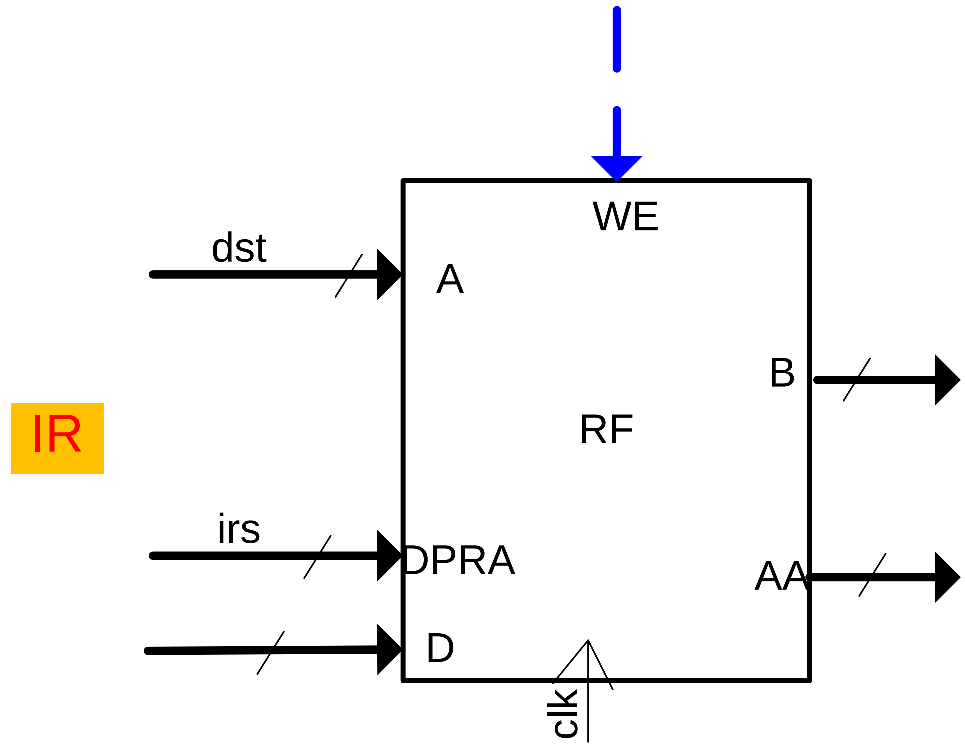
from Lectures on Reconfigurable Computing, *Dr. Ling Liu,* ETH Zürich

Single-Cycle Datapath: instruction fetch

```
wire [PAW-1:0] pcmux, nxpc;
wire [17:0] IR;
reg [PAW-1:0] PC;

IM #(.BN(IMB)) imx(.clk(clk), .pmadr({{{32-PAW}{1'b0}},pcmux[PAW-1:1]}),
   .pmout(pmout));

assign IR = (~rst)? NOP: (PC[0]) ? pmout[35:18] : pmout[17:0];

always @ (posedge clk) begin
  if (~rst) PC <= 0;
  else if (stall0)
      PC <= PC;
  else
      PC <= pcmux;
end
```

Single-Cycle Datapath: register read

- **STEP 2:** Read source operands from register file




```
wire [2:0] rd, rs;
wire regWr;
wire [31:0] rdOut, rsOut;

//register file
…

assign irs = IR[2:0];
assign ird = IR[13:11];
assign dst = (BL)? 7: ird;
```

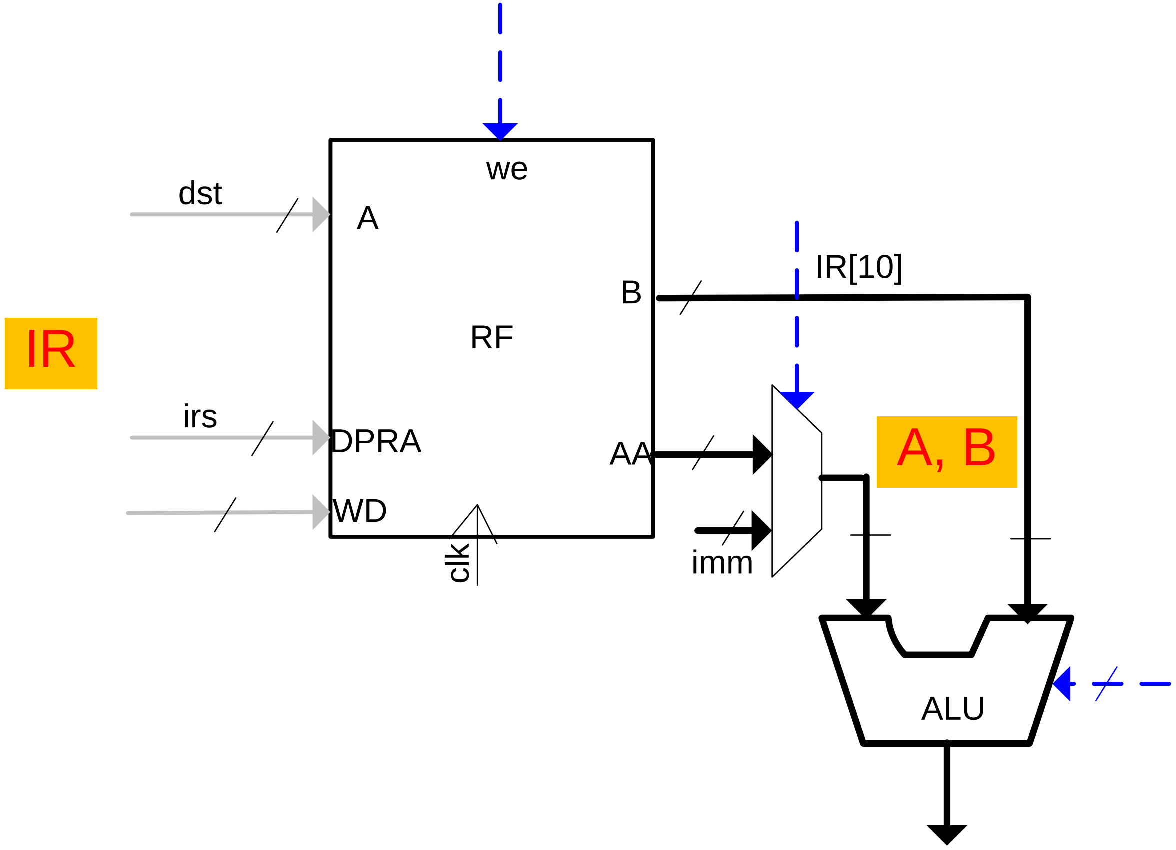
source register

destination register

from Lectures on Reconfigurable Computing, *Dr. Ling Liu,* ETH Zürich

Single-Cycle Datapath: ALU

- **STEP 3:** Compute the result via ALU




```
wire [31:0] AA, A, B, imm;
wire [32:0] aluRes;

assign A = (IR[10])? AA:
                {22’b0, imm};

assign minusA = {1‘b0, ~A} + 33‘d1;
assign aluRes =
    (MOV)? A:
    (ADD)? {1‘b0, B} + {1‘b0, A} :
    (SUB)? {1‘b0, B} + minusA :
    (AND)? B & A :
    (BIC)? B & ~A :
    (OR)? B | A :
    (XOR)? B ^ A :
    ~A;
```

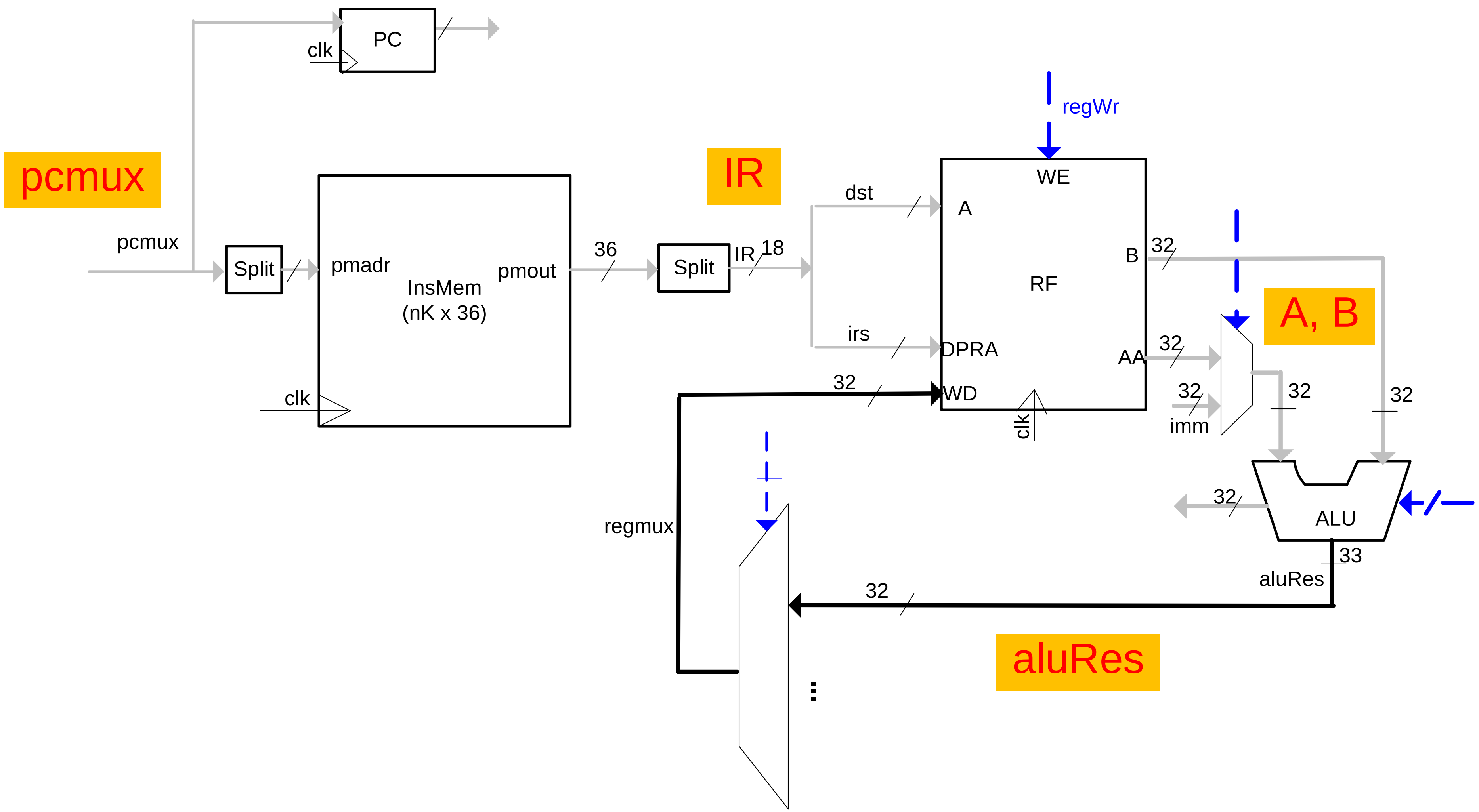
Control Path

```
assign vector = IR[10] & IR[9] & ~IR[8] & ~IR[7];
assign op = IR[17:14];

assign MOV = (op == 0);
assign NOT = (op == 1);
assign ADD = (op == 2);
assign SUB = (op == 3);
assign AND = (op == 4);
assign BIC = (op == 5);
assign OR  = (op == 6);
assign XOR = (op == 7);
assign MUL = (op == 8) & (~IR[10] | ~IR[9]);
assign ROR = (op == 10);
assign BR  = (op == 11) & IR[10] & ~IR[9];
assign LDR = (op == 12);
assign ST  = (op == 13);
assign Bc  = (op == 14);
assign BL  = (op == 15);
assign LDH = MOV & IR[10] & IR[3];
assign BLR = (op == 11) & IR[10] & IR[9];
```

$IR_{17:14}$	Function
0000	B := A
0001	B := ~A
0010	B := B + A
0011	B := B – A
0100	B := B & A
0101	B := B & ~A
0110	B := B | A
0111	B := B ^ A

Single-Cycle Datapath: write back to Rd

- **STEP 4:** Write result back to Rd



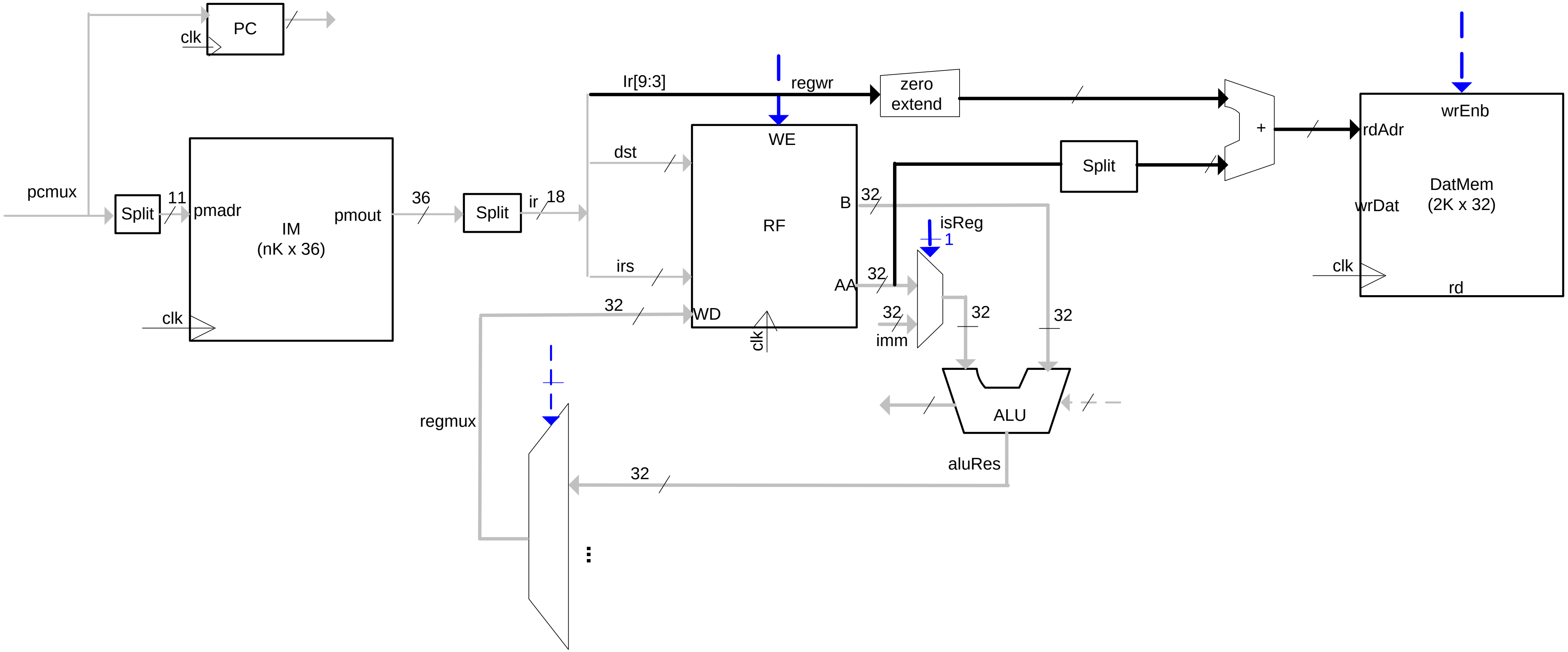
from Lectures on Reconfigurable Computing, *Dr. Ling Liu,* ETH Zürich

Single-Cycle Datapath: write back to Rd

```
wire [31:0] regmux;
wire regwr;

...

assign regwr = (BL | BLR | LDR & ~IR[10] |
               ~(IR[17] & IR[16]) & ~BR & ~vector)) & ~stall0;

assign regmux =
    (BL | BLR) ? {{{32-PAW}{1'b0}}, nxpc} :
    (LDR & ~IoenbReg) ? dmout :
    (LDR & IoenbReg)? InbusReg: //from IO
    (MUL) ? mulRes[31:0] :
    (ROR) ? s3 :
    (LDH) ? H :
    aluRes;
```

Single-Cycle Datapath: LD

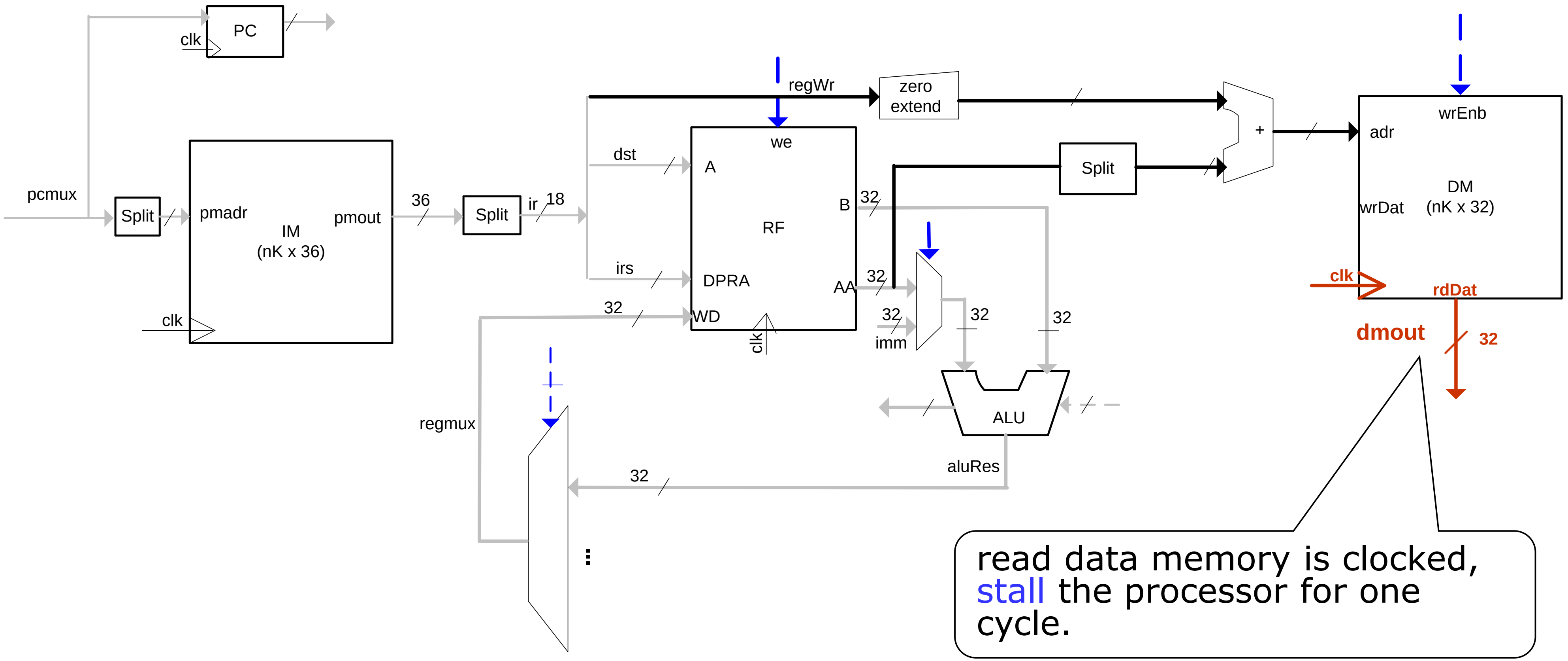
- **STEP 1:** Fetch instruction
- **STEP 2:** Read source operand from the register file
- **STEP 3:** Compute the memory address



from Lectures on Reconfigurable Computing, *Dr. Ling Liu,* ETH Zürich

Single-Cycle Datapath: LD

- **STEP 3:** Compute the memory address
- **STEP 4:** Read data from data memory



read data memory is clocked, stall the processor for one cycle.

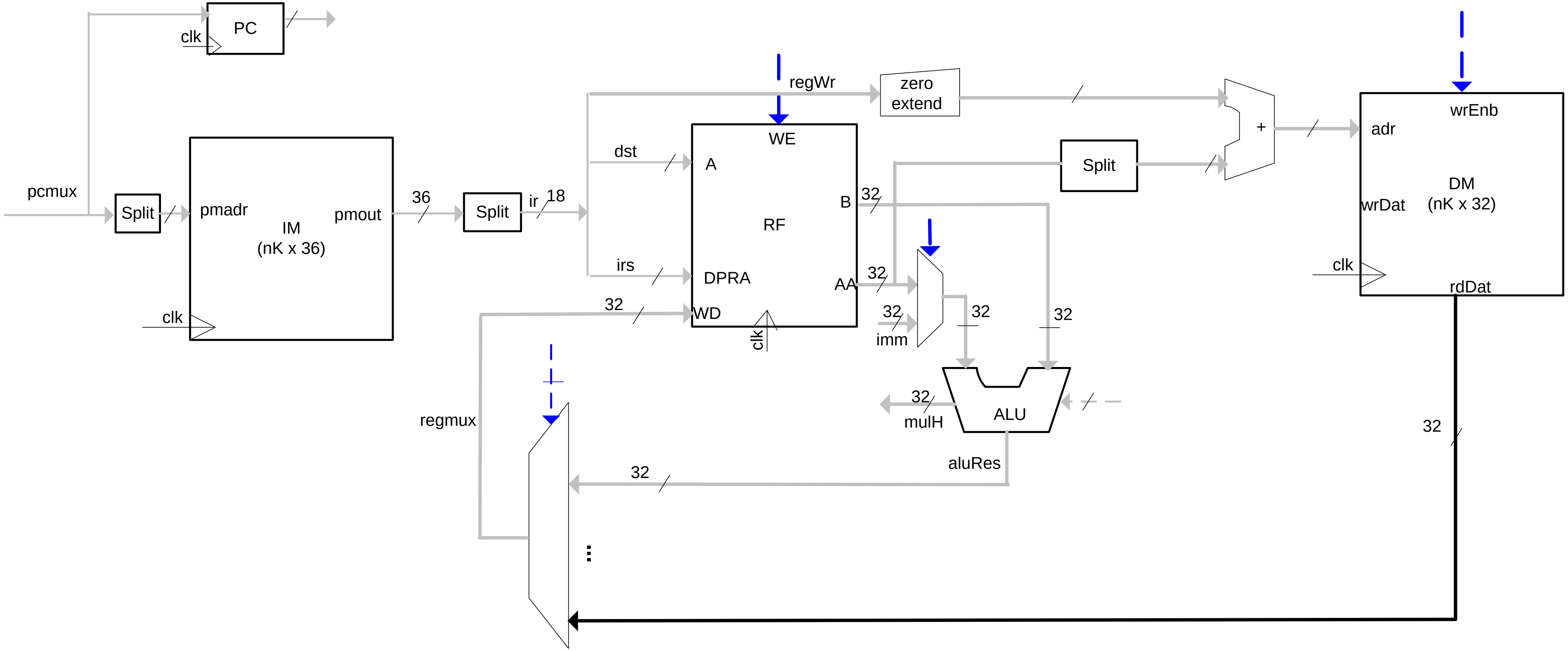
from Lectures on Reconfigurable Computing, *Dr. Ling Liu,* ETH Zürich

TRM Stalling

- stop fetching next instruction, pcmux keeps the current value
- disable register file write enable and memory write enable signals to avoid changing the state of the processor.
  - only LD and MUL instructions stall the processor.
  - dmwe signal is not affected.
  - regwr signal is affected.

Single-Cycle Datapath: LD

- **STEP 4:** Read data from data memory
- **STEP 5:** Write data back into the register file



from Lectures on Reconfigurable Computing, *Dr. Ling Liu,* ETH Zürich

TRM: LD

LD rd, [rs+offset]

Verilog code in TRM module

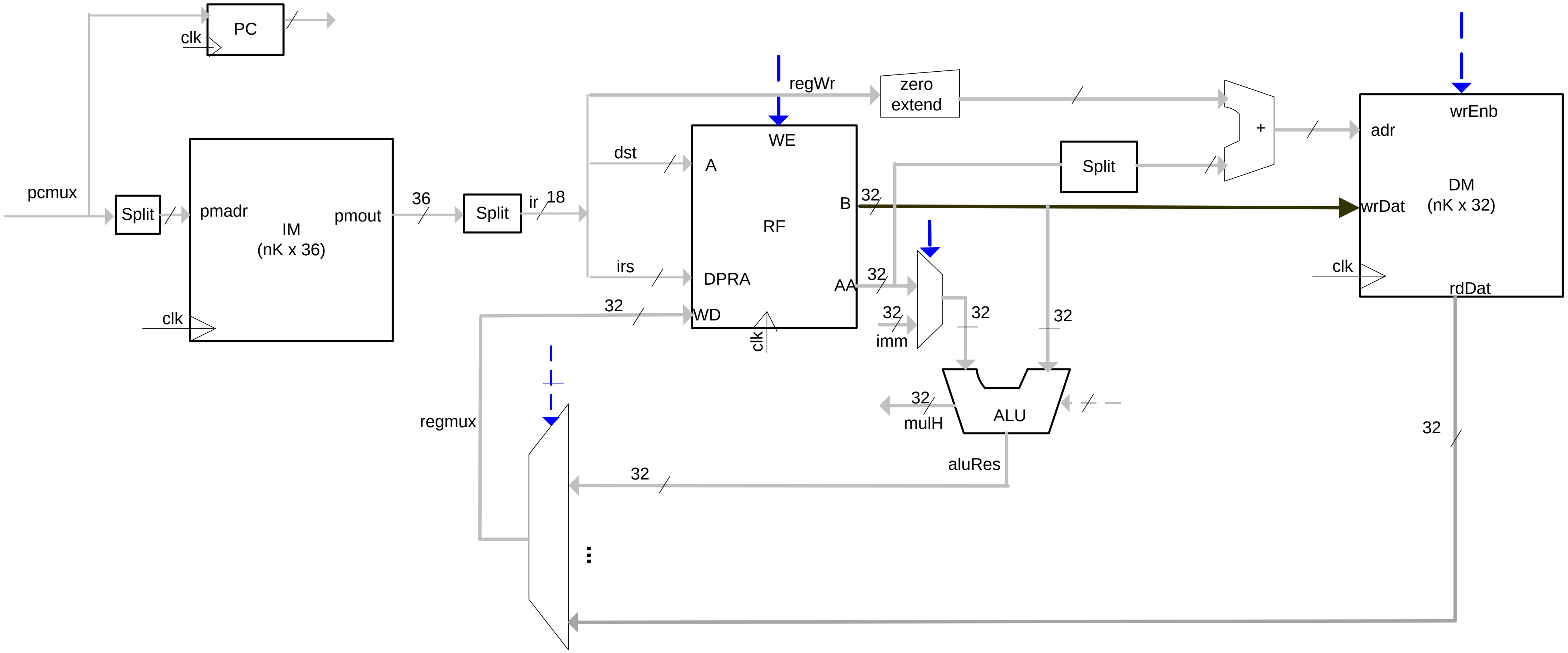
```
wire [31:0] dmout;
wire [DAW:0] dmadr;
wire [6:0] offset;
reg IoenbReg;


//register file
…
Assign dmadr = (irs == 7) ? {{{DAW-6}{1'b0}}, offset} :(AA[DAW:0] + {{{DAW-6}{1'b0}}, offset});
assign ioenb = &(dmadr[DAW:6]);
assign rfWd = ...
              (LDR & ~IoenbReg)? dmout:
             (LDR & IoenbReg)? InbusReg: //from IO
              ...;
always @(posedge clk)
    IoenbReg <= ioenb;
```

Src register = lr ignored (Harvard architecture!)

I/O space: Uppermost 2^6 bytes in data memory

Single-Cycle Datapath: ST

- **STEP 1:** Fetch instruction
- **STEP 2:** Read source operand from the register file
- **STEP 3:** Compute the memory address
- **STEP 4:** Write data into the data memory



from Lectures on Reconfigurable Computing, *Dr. Ling Liu*, ETH Zürich

Single-Cycle Datapath: ST

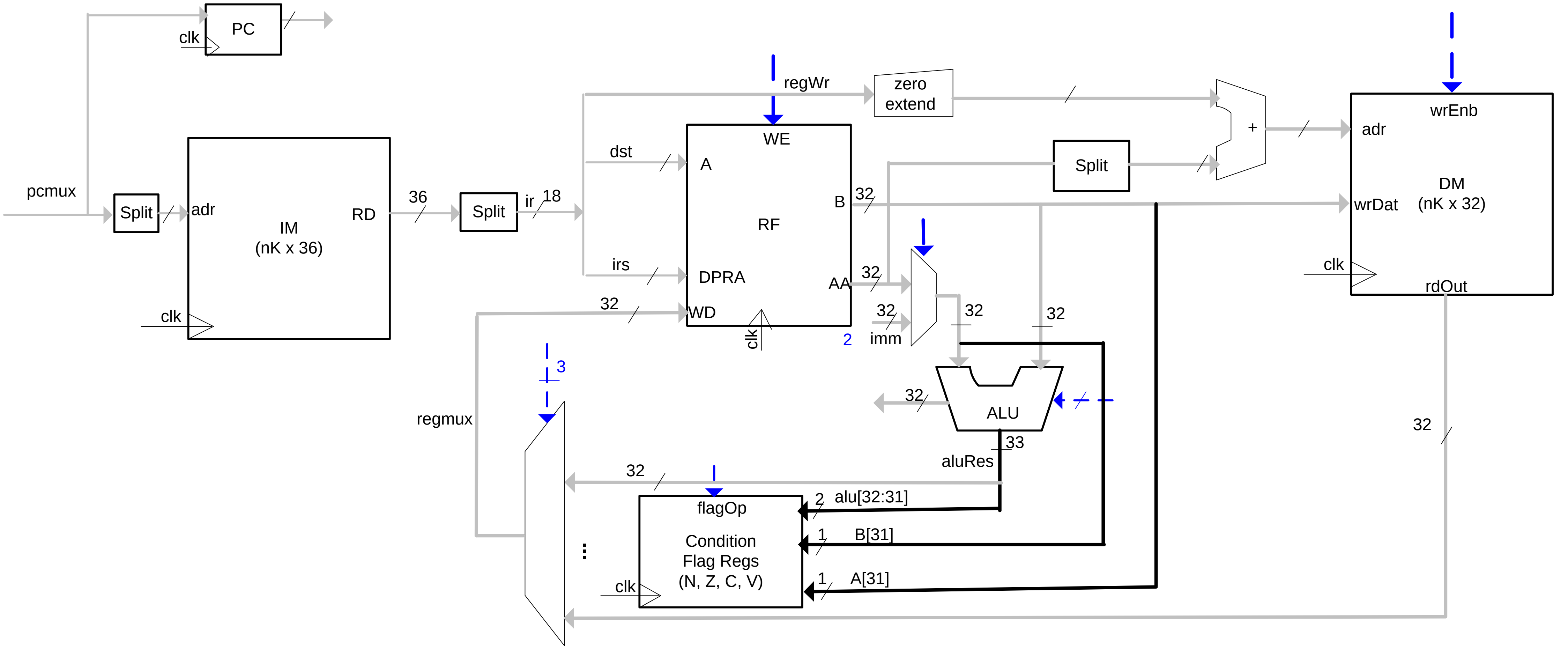
- **STEP 3:** Compute the memory address
- **STEP 4:** Write data into the data memory

```
wire [31:0] dmin;
wire dmwr;

//register file
…
DM #(.BN(DMB)) dmx (.clk(clk),
   .wrDat(dmin),
     .wrAdr({{{31-DAW}{1'b0}},dmadr}),
     .rdAdr({{{31-DAW}{1'b0}},dmadr}),
     .wrEnb(dmwe),
     .rdDat(dmout));

Assign dmwe = ST & ~IR[10] & ~ioenb;
assign dmin = B;
```

Single-Cycle Datapath: set flag registers

```
always @ (posedge clk, negedge rst) begin  // flags
    handling
  if (~rst) begin N <= 0; Z <= 0; C <= 0; V <= 0; end
  else begin
    if (regwr) begin
        N <= aluRes[31];
        Z <= (aluRes[31:0] == 0);
        C <= (ROR & s3[0]) | (~ROR & aluRes[32]);
        V <= ADD & ((~A[31] & ~B[31] & aluRes[31])
               | (A[31] & B[31] & ~aluRes[31]))
               | SUB & ((~B[31] & A[31] & aluRes[31])
               | (B[31] & ~A[31] & ~aluRes[31]));
    end
  end
end
```

Single-Cycle Datapath: set flag registers

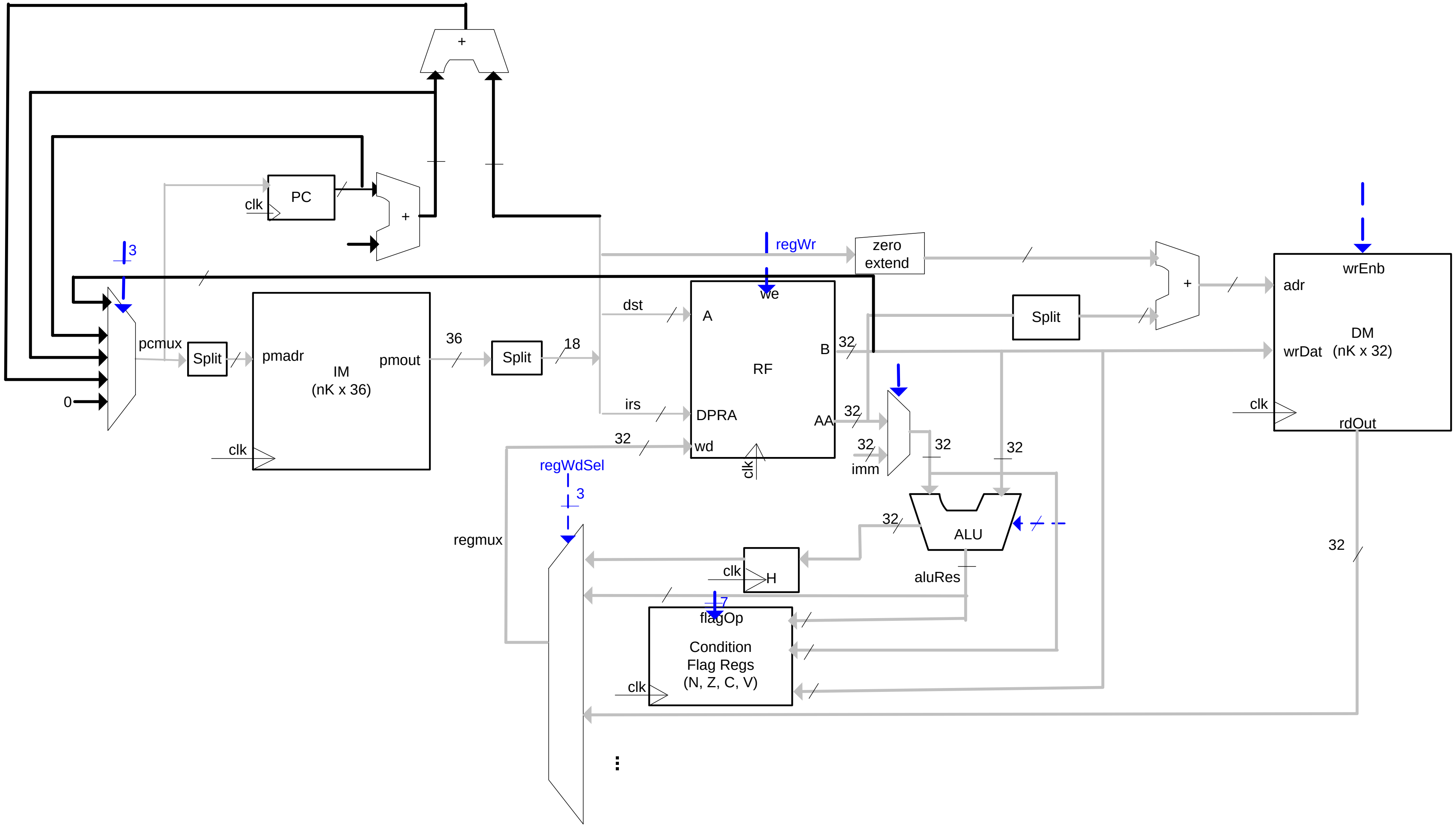


from Lectures on Reconfigurable Computing, *Dr. Ling Liu,* ETH Zürich

Single-Cycle Datapath: Branch instructions

- Type c instructions, BR instruction, BL instruction
  - PC <= PC + 1 + off
  - PC <= Rs
  - **PC <= PC + 1 (by default)**
  - **PC <= PC (if stall)**
  - **PC <= 0 (reset)**

Single-Cycle Datapath: Branch instructions

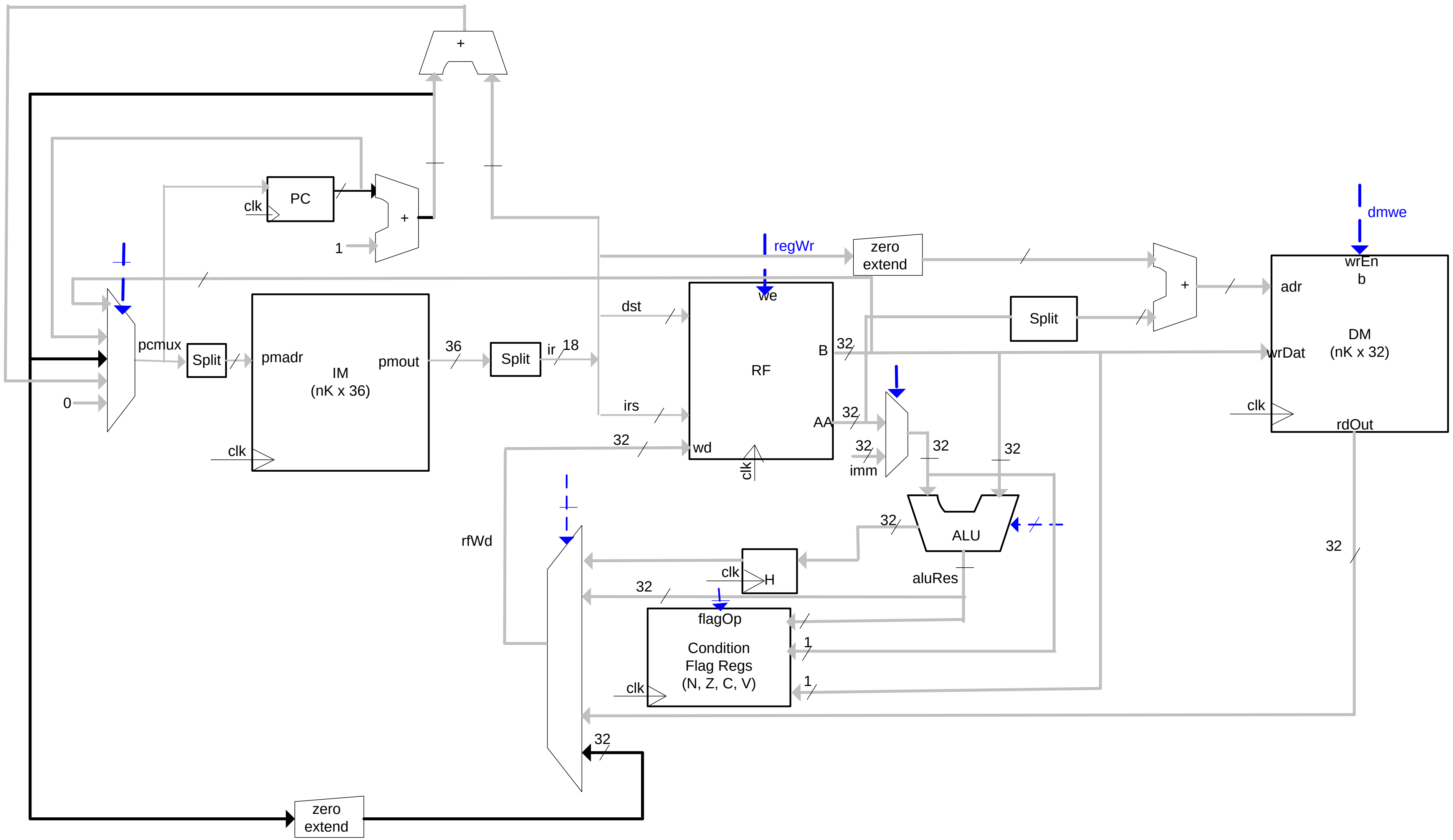


from Lectures on Reconfigurable Computing, *Dr. Ling Liu,* ETH Zürich

Single-Cycle Datapath: Branch instructions

```
//pcmux logic
assign pcmux =
  (~rst) ? 0 :
  (stall0) ? PC:
  (BL)? {{10{IR[BLS-1]}},IR[BLS-1: 0]}+ nxpc :
  (Bc & cond) ? {{{PAW-10}{IR[9]}}, IR[9:0]} + nxpc :
  (BLR | BR ) ? A[PAW-1:0] :
  nxpc;
```

Complete single-cycle datapath



from Lectures on Reconfigurable Computing, *Dr. Ling Liu,* ETH Zürich