Implementing an Instruction Set

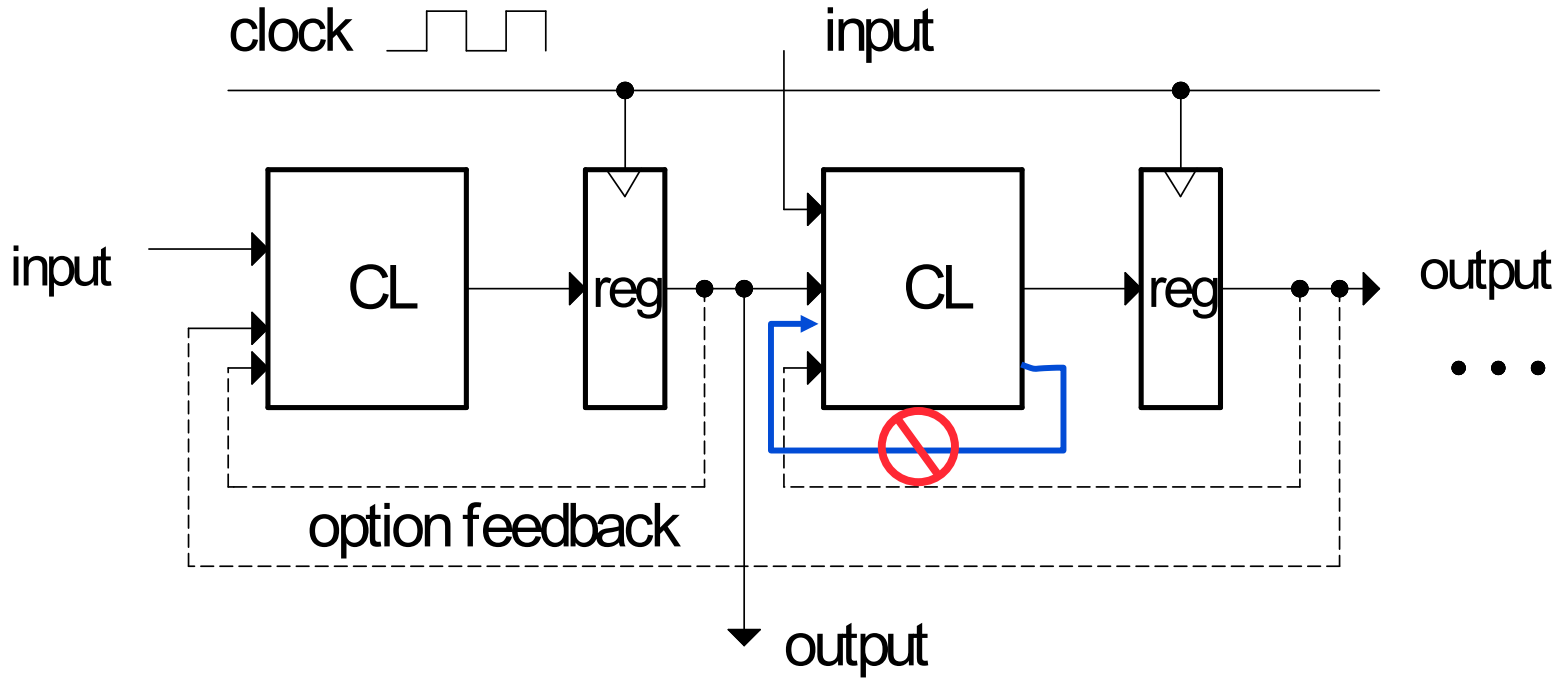
David E. Culler

CS61CL

Oct 28, 2009

Lecture 9

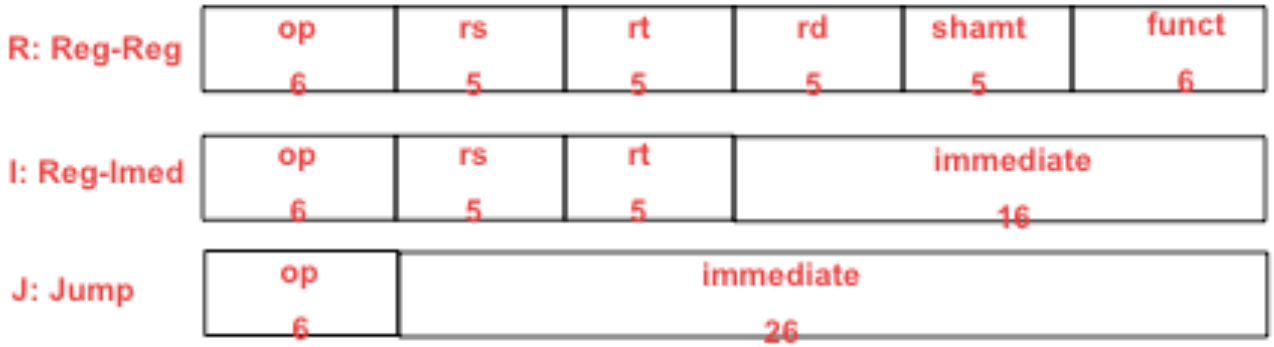
Review: Synchronous Circuit Design



- **Combinational Logic Blocks (CL)**
  - Acyclic
  - no internal state (no feedback)
  - output only a function of inputs
- **Registers (reg)**
  - collections of flip-flops
- **clock**
  - distributed to all flip-flops
- **ALL CYCLES GO THROUGH A REG!**

MIPS: Register Transfers

MIPS Instruction Format



- **Reg-Reg instructions (op == 0)**
  - add, sub, and, or, nor, xor, slt — R[rd] := R[rs] *funct* R[rt]; pc:=pc+4
  - sll. srl, sra — R[rd] := R[rt] *shft* shamt
- **Reg-Immed (op != 0)**
  - addi, andi, ori, xori, lui, — R[rt] := R[rs] op Im16
  - addiu, slti, sltiu
  - lw, lh, lhu, lb, lbu — R[rt] := Mem[R[rs] + signEx(Im16)]*
  - sw, sh, sb — Mem[R[rs] + signEx(Im16)] := R[rt]

10/21/09 cs61cl f09 lec 5 12

MIPS: Register Transfers

MIPS Instruction Format

- **Reg-Reg instructions (op == 0)**
- **Reg-Immed (op != 0)**
- **Jumps**
  - j — PC := $PC_{31..28}$ || addr || 00
  - jal — PC := $PC_{31..28}$ || addr || 00; R[31] := PC + 4
  - jr — PC := R[rs]

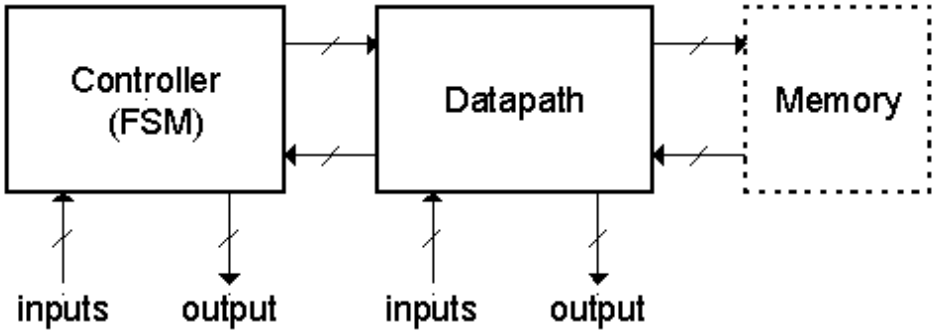
MIPS: Register Transfers

MIPS Instruction Format

- **Reg-Reg instructions (op == 0)**
- **Reg-Immed (op != 0)**
- **Jumps**
- **Branches**
  - **BEQ, BNE** PC := (R[rs] ?= R[rt]) ? PC + signEx(im16) : PC+4
  - BLT, BGT, BLE, BGTE are pseudo ops
  - Move and LI are pseudo ops too

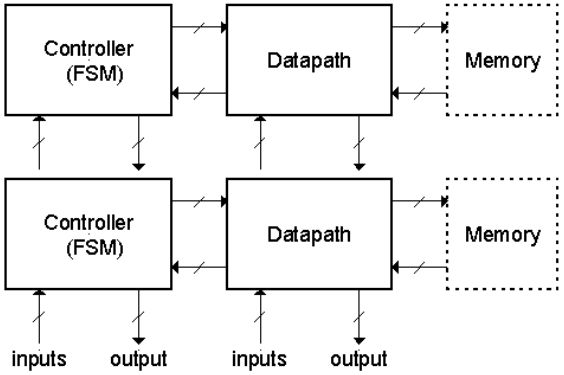
10/27/09 cs61cl f09 lec 5 20

A Standard High-level Organization

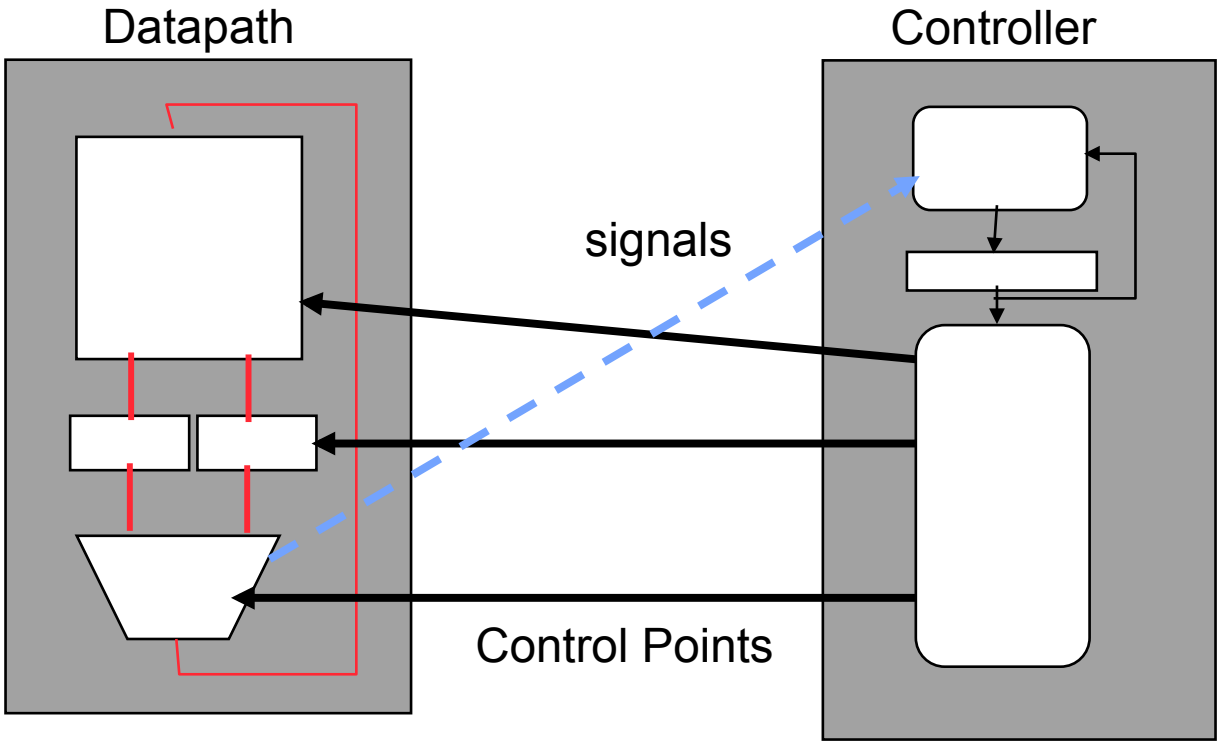


- **Controller**
  - accepts external and control input, generates control and external output and sequences the movement of data in the datapath.
- **Datapath**
  - is responsible for data manipulation. Usually includes a limited amount of storage.
- **Memory**
  - optional block used for long term storage of data structures.

- **Standard model for CPUs, micro-controllers, many other digital sub-systems.**
- **Usually *not* nested.**
- **Often cascaded:**



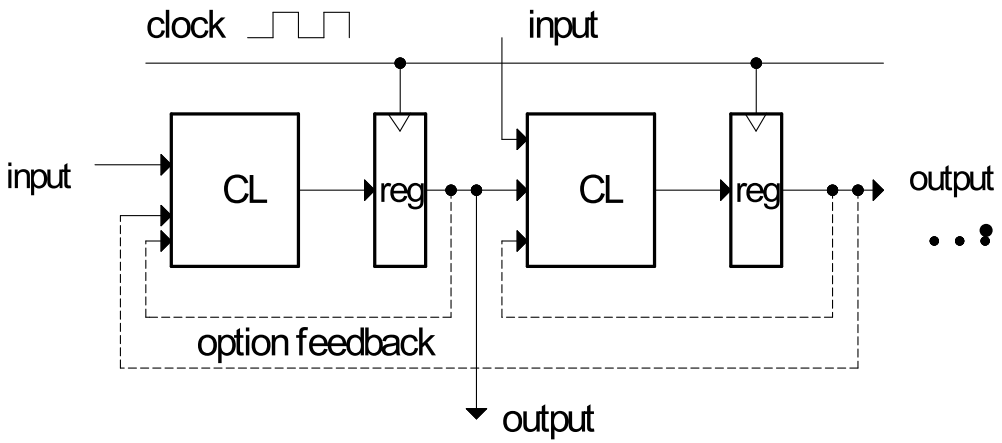
Datapath vs Control



- **Datapath: Storage, FU, interconnect sufficient to perform the desired functions**
  - **Inputs are Control Points**
  - **Outputs are signals**
- **Controller: State machine to orchestrate operation on the data path**
  - **Based on desired function and signals**

Register Transfer Level Descriptions

- A standard high-level representation for describing systems.
- It follows from the fact that all synchronous digital system can be described as a set of state elements connected by combination logic (CL) blocks:



- RTL comprises a set of *register transfers* with optional operators as part of the transfer.
- Example:

  regA ← regB

  regC ← regA + regB

  if (start==1) regA ← regC

- Personal style:
  - use ";" to separate transfers that occur on separate cycles.
  - Use "," to separate transfers that occur on the same cycle.
- Example (2 cycles):

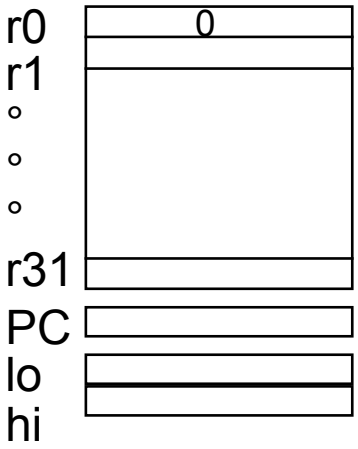
  regA ← regB, regB ← 0;

  regC ← regA;

The Methodology

1. Identify state and operations visible in the ISA
   - register transfer level operations of the ISA
2. Select a set of storage elements, function units, and interconnections that
   - realize all the "architected" state & operations
   - plus internal state and operations
3. Map each instruction to register transfers on the data path
4. Identify the control points that cause the specified data transfers
5. Implement a controller that asserts those control points
   - as a function of instruction, controller state, and signals from the data path

MIPS R3000 – tiny subset



Programmable storage

- 2^32 x bytes
- 31 x 32-bit GPRs (R0=0)
- 32 x 32-bit FP regs (paired DP)
- HI, LO, PC

Arithmetic logical

Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU,
AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI, *LUI*
SLL, SRL, SRA, SLLV, SRLV, SRAV

Memory Access

LB, LBU, LH, LHU, LW, LWL,LWR
SB, SH, SW, SWL, SWR

Control

J, JAL, JR, JALR
BEQ, BNE, BLEZ,BGTZ,BLTZ,BGEZ,BLTZAL,BGEZAL

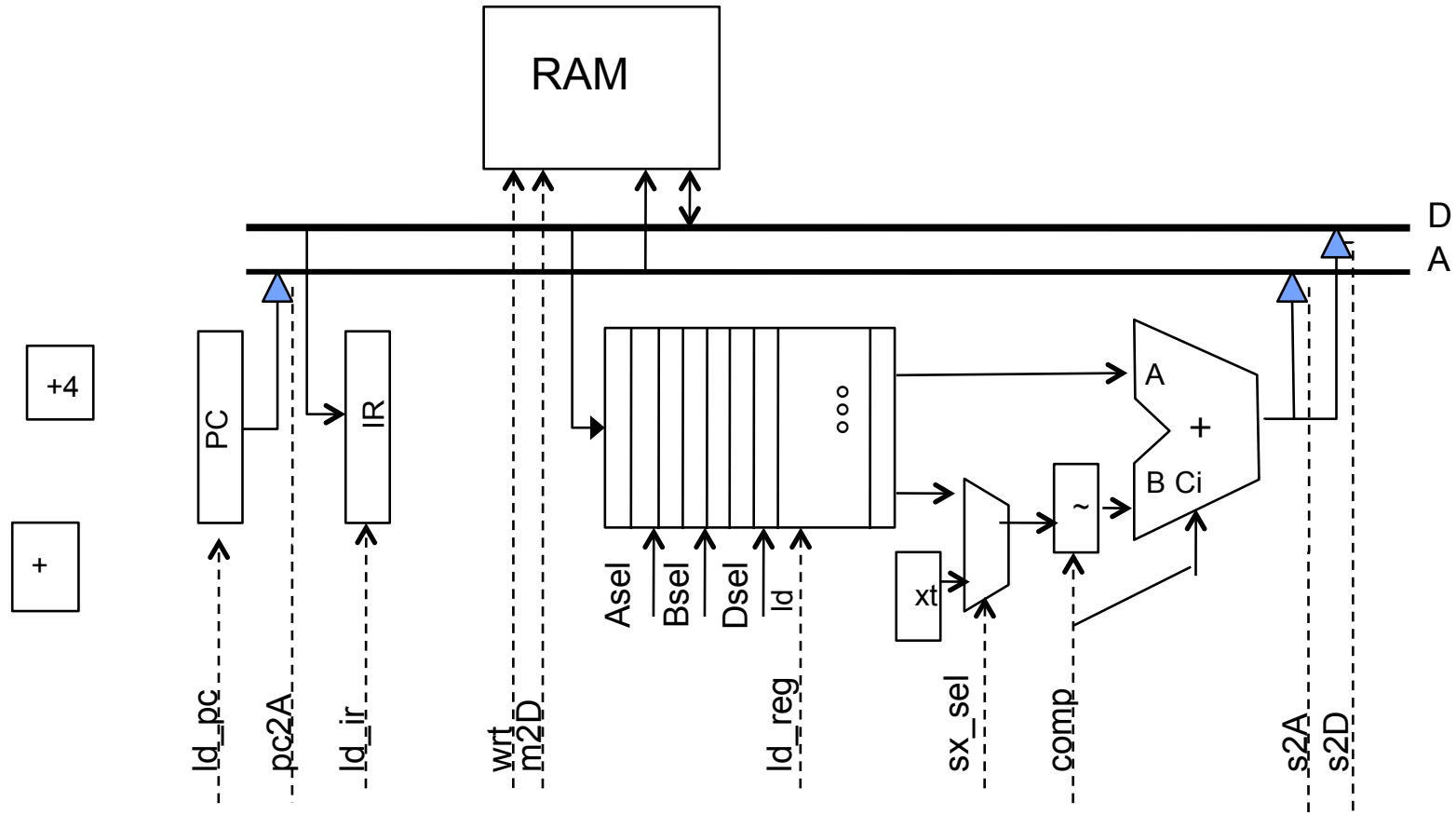
TinyMIPS

- **Reg-Reg instructions (op == 0)**
  - addu — R[rd] := R[rs] **+** R[rt]; pc:=pc+4
  - **subu** — **R[rd] := R[rs] - R[rt]; pc:=pc+4**
- **Reg-Immed (op != 0)**
  - lw — R[rt] := Mem[R[rs] + signEx(Im16)]
  - sw — Mem[R[rs] + signEx(Im16)] := R[rt]
- **Jumps**
  - **j** — PC := $PC_{31..28}$ || addr || 00
  - **jr** — PC := R[rs**]**
- **Branches**
  - **BEQ** — PC := (R[rs] == R[rt]) ? PC + signEx(im16) : PC+4
  - **BLTZ** — PC := (R[rs] < 0) ? PC + signEx(im16) : PC+4

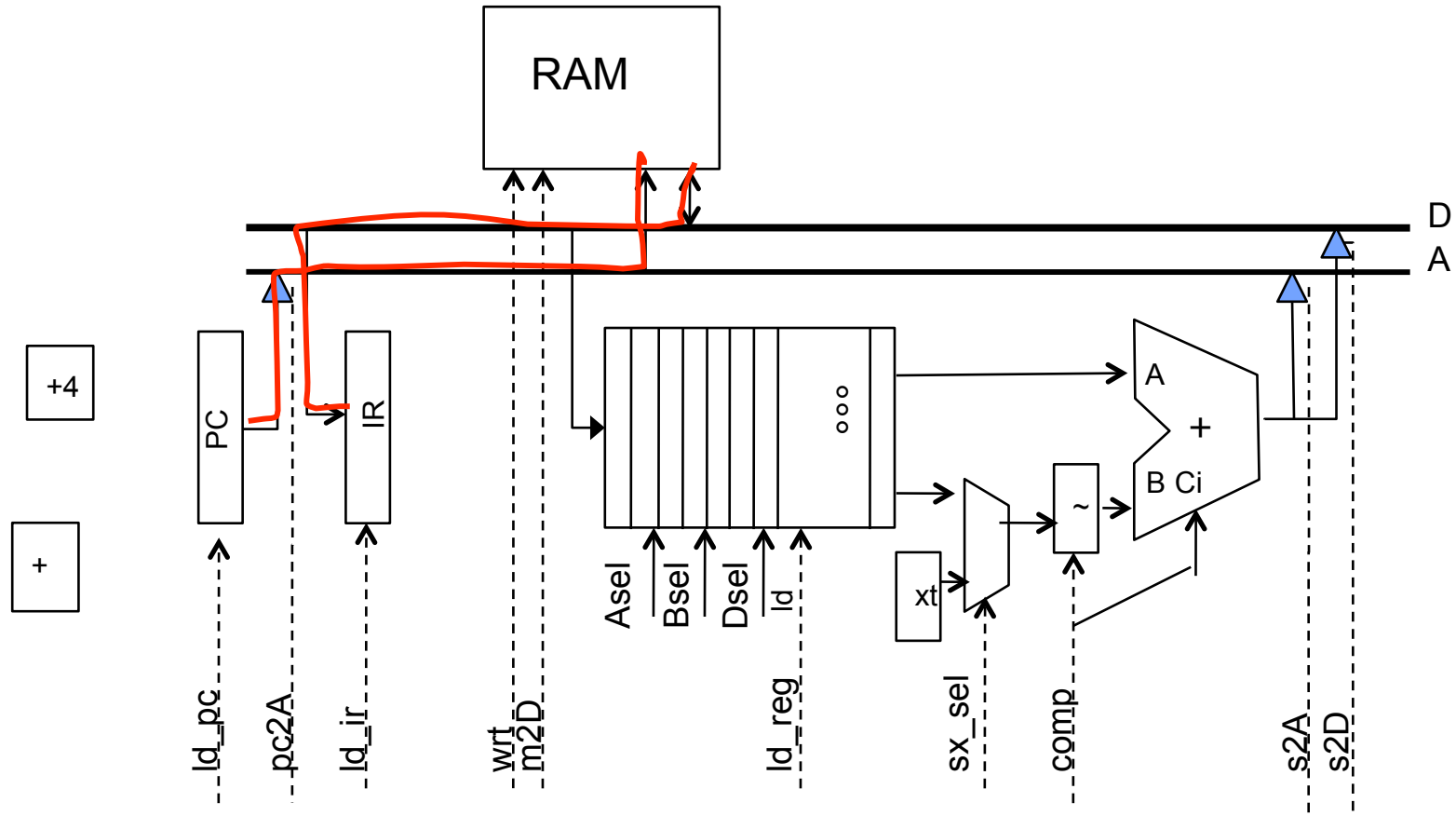
Administration

- Project 3 is out – due Sun 11/15
- Homework 7 is out – due Wed 11/4
- Midterm 2 is Monday 11/9

Starting Point

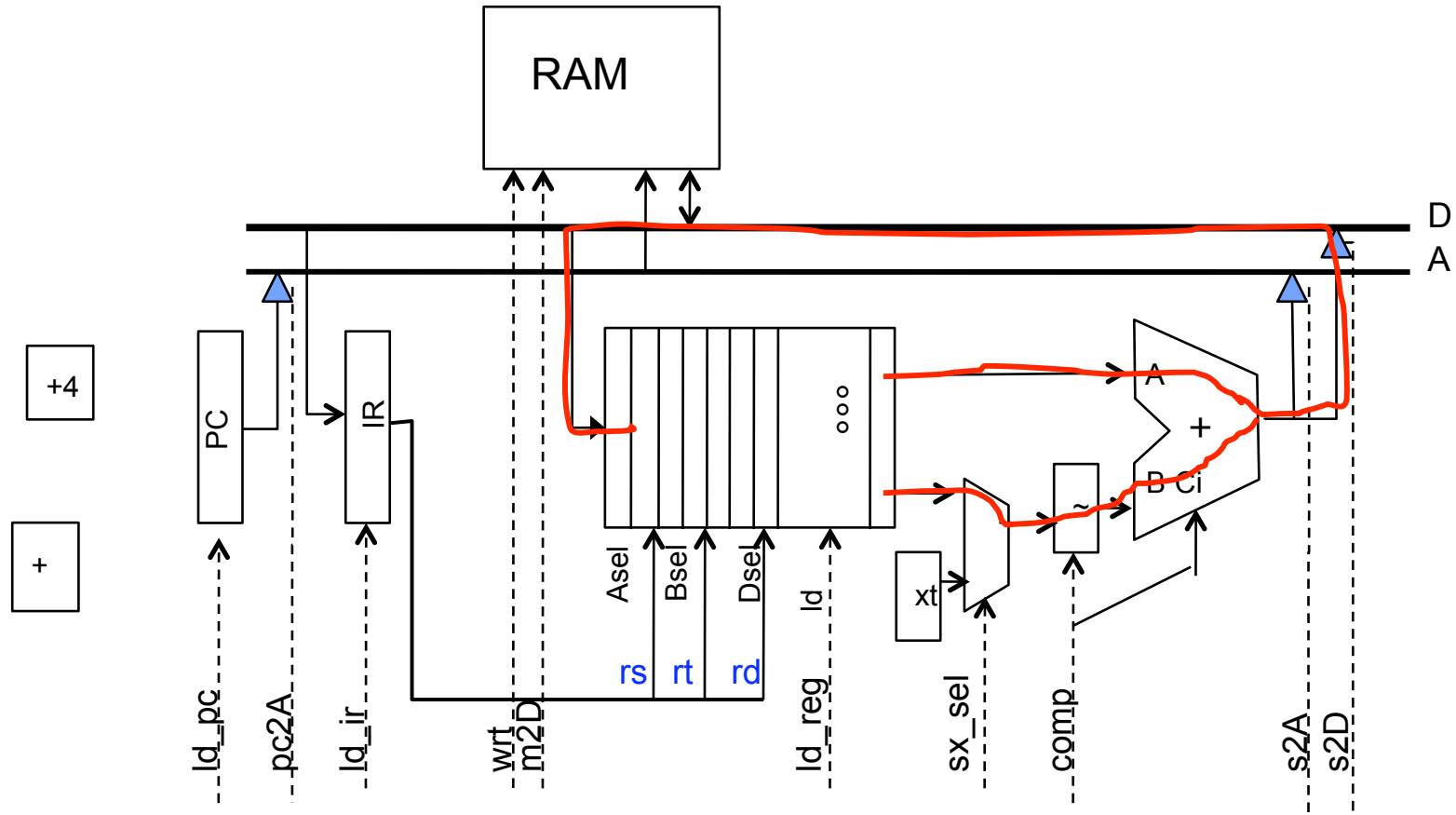


Ifetch: IR := Mem[PC]



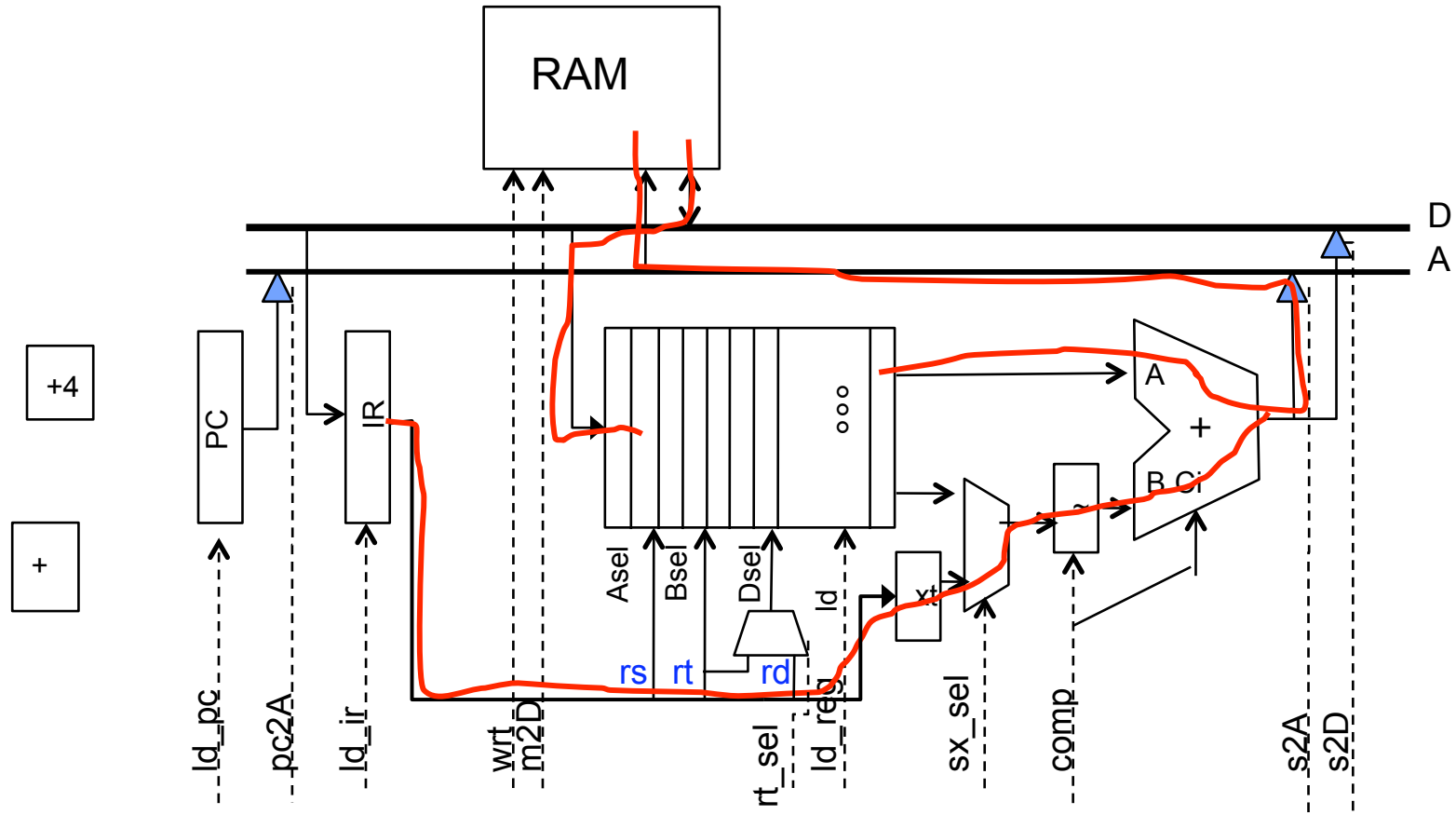
- `RAM_addr <- A <- PC;` (pc2A)
- `IR_in <- D <- RAM_data;` (m2D, ~wrt)
- `IR := IR_in;` (ld_ir)

Exec: Arithmetic



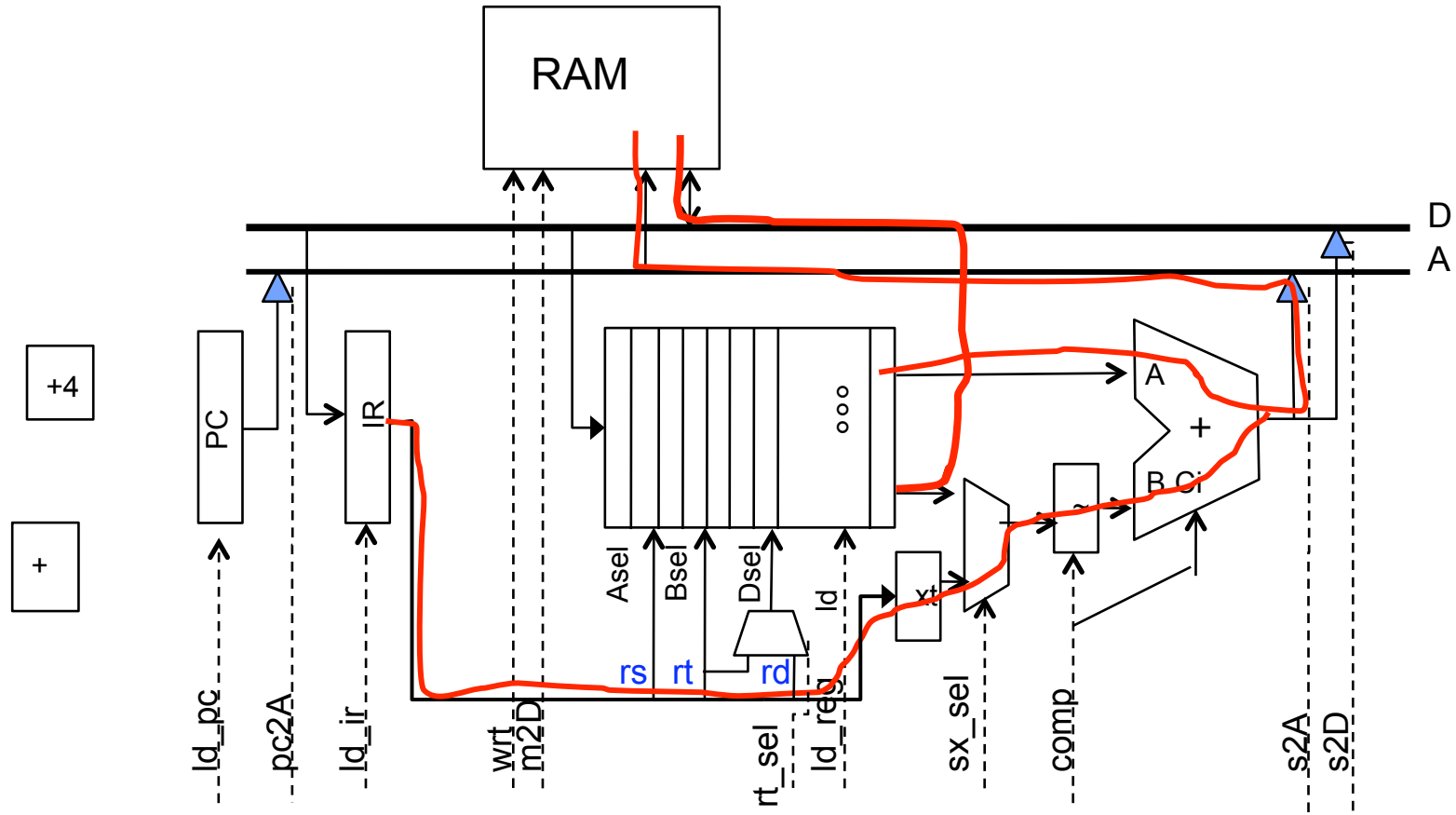
- `R[rd] := R[rs] + R[rt];(~sx_sel,~comp,s2D,ld_reg,~wrt)`
- `R[rd] := R[rs] – R[rt];(~sx_sel, comp,s2D,ld_reg,~wrt)`

Exec: LW



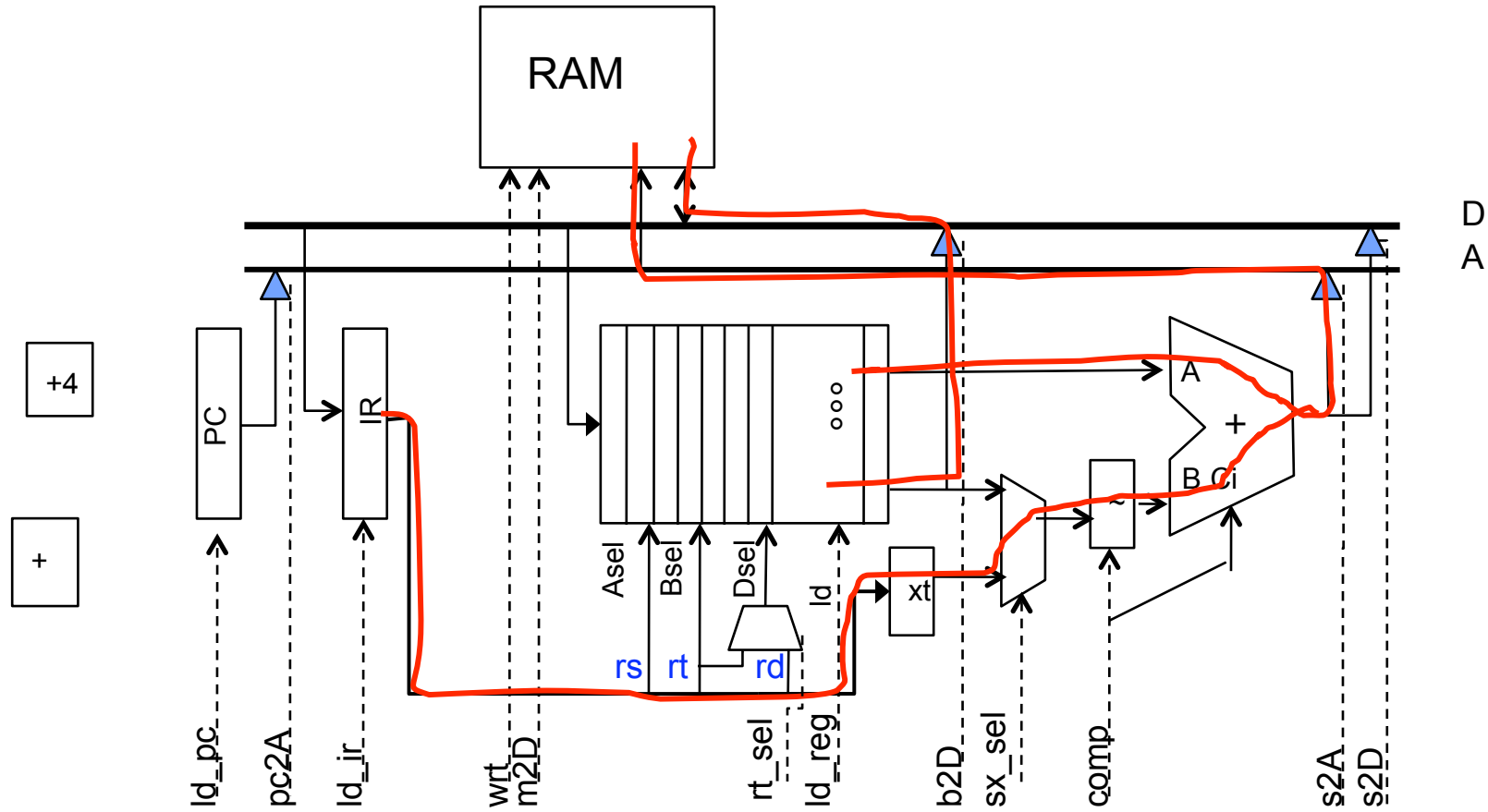
- `R[rt]:= Mem[R[rs]+SXim16];`
- `sx_sel,~comp,s2A,~pc2A,~wrt,m2D,ld_reg`
- `rt_sel`

Exec: SW



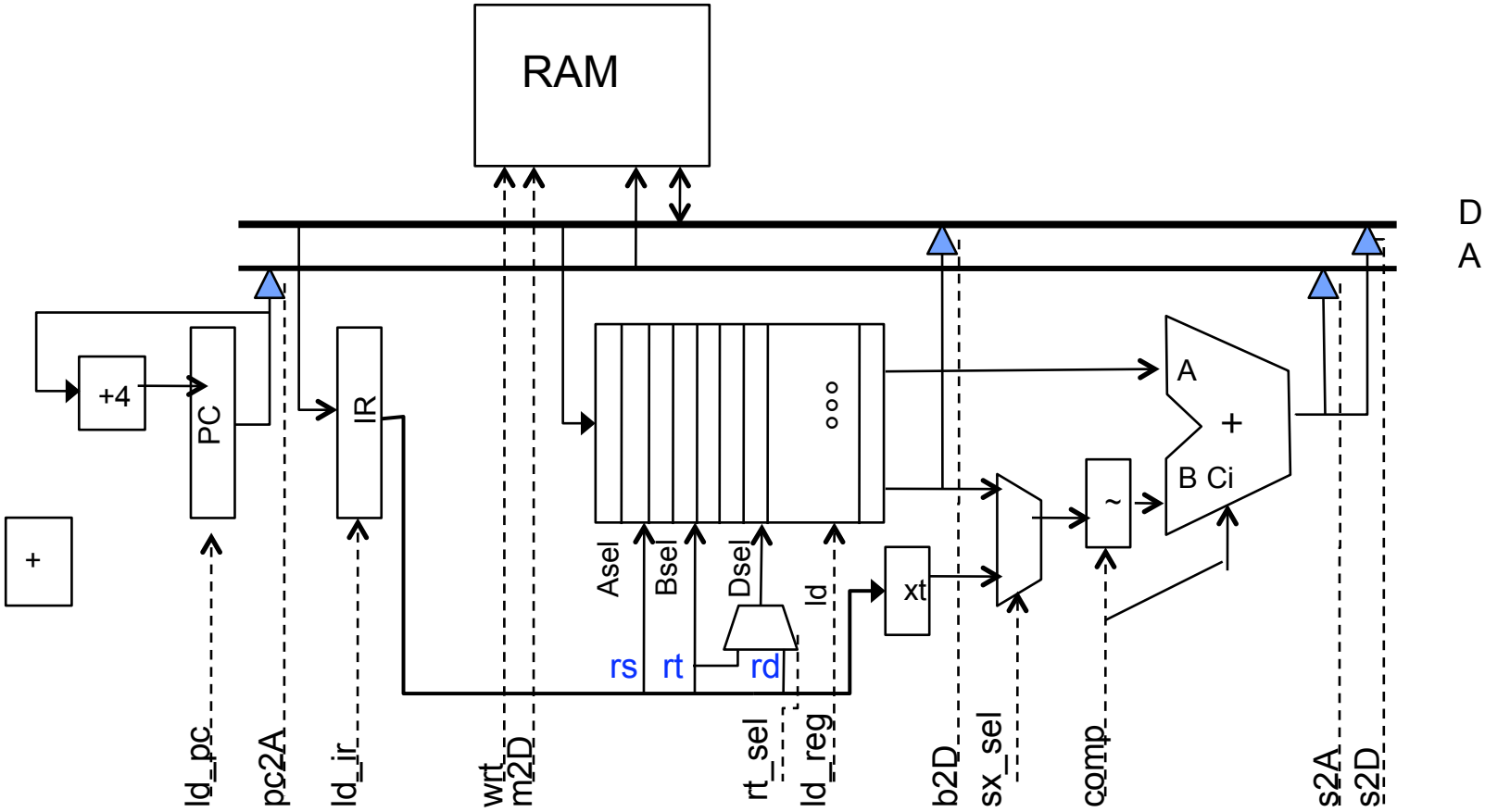
- **`Mem[R[rs]+SXim16]:= R[rt];`**

Exec: SW



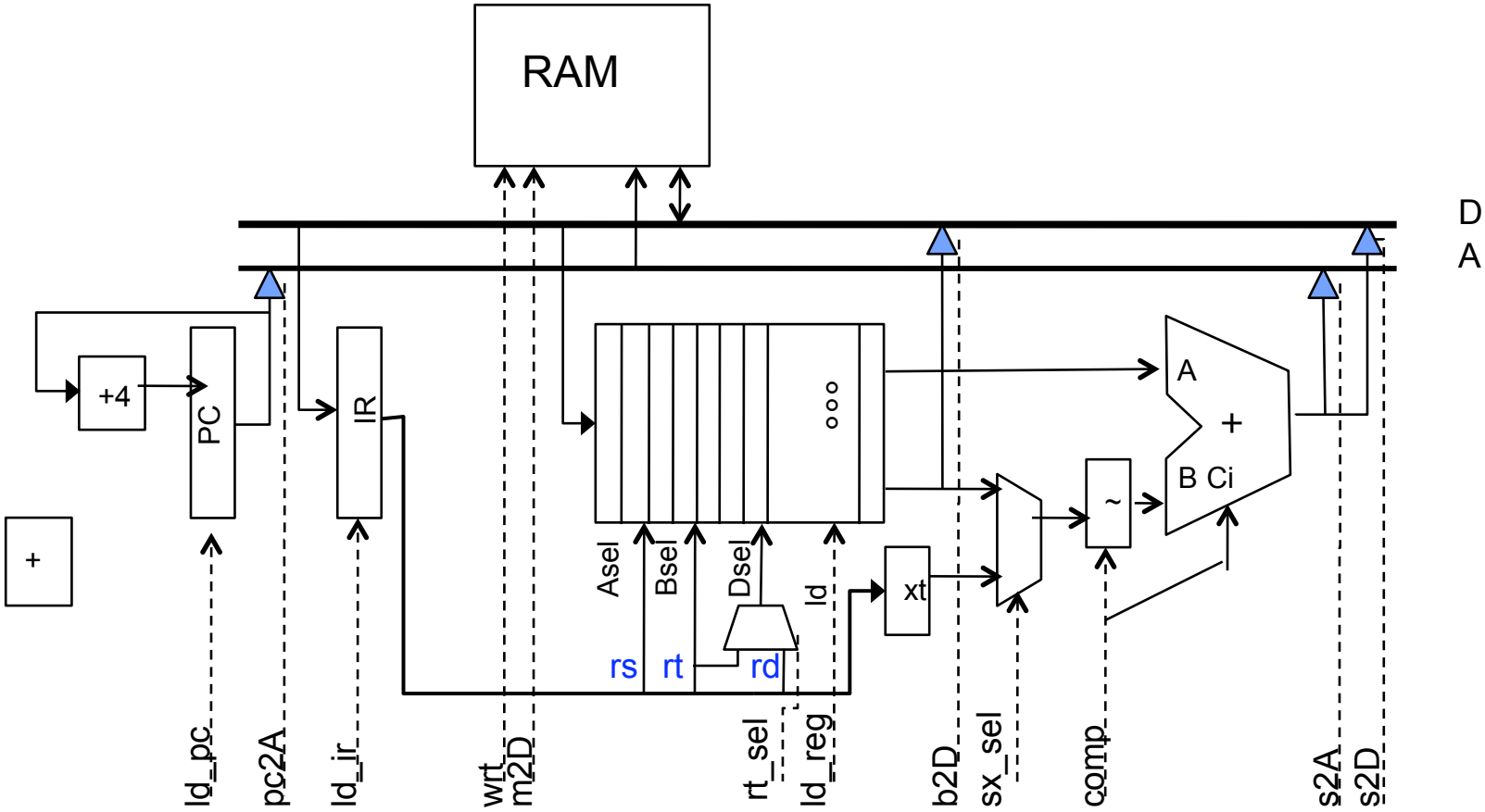
- **`Mem[R[rs]+SXim16]:= R[rt];`**
- **`b2D, sx_sel, ~comp, s2A, ~s2D, ~m2D, wrt, ~ld_reg, ~pc2A`**

Exec: PC := PC+4



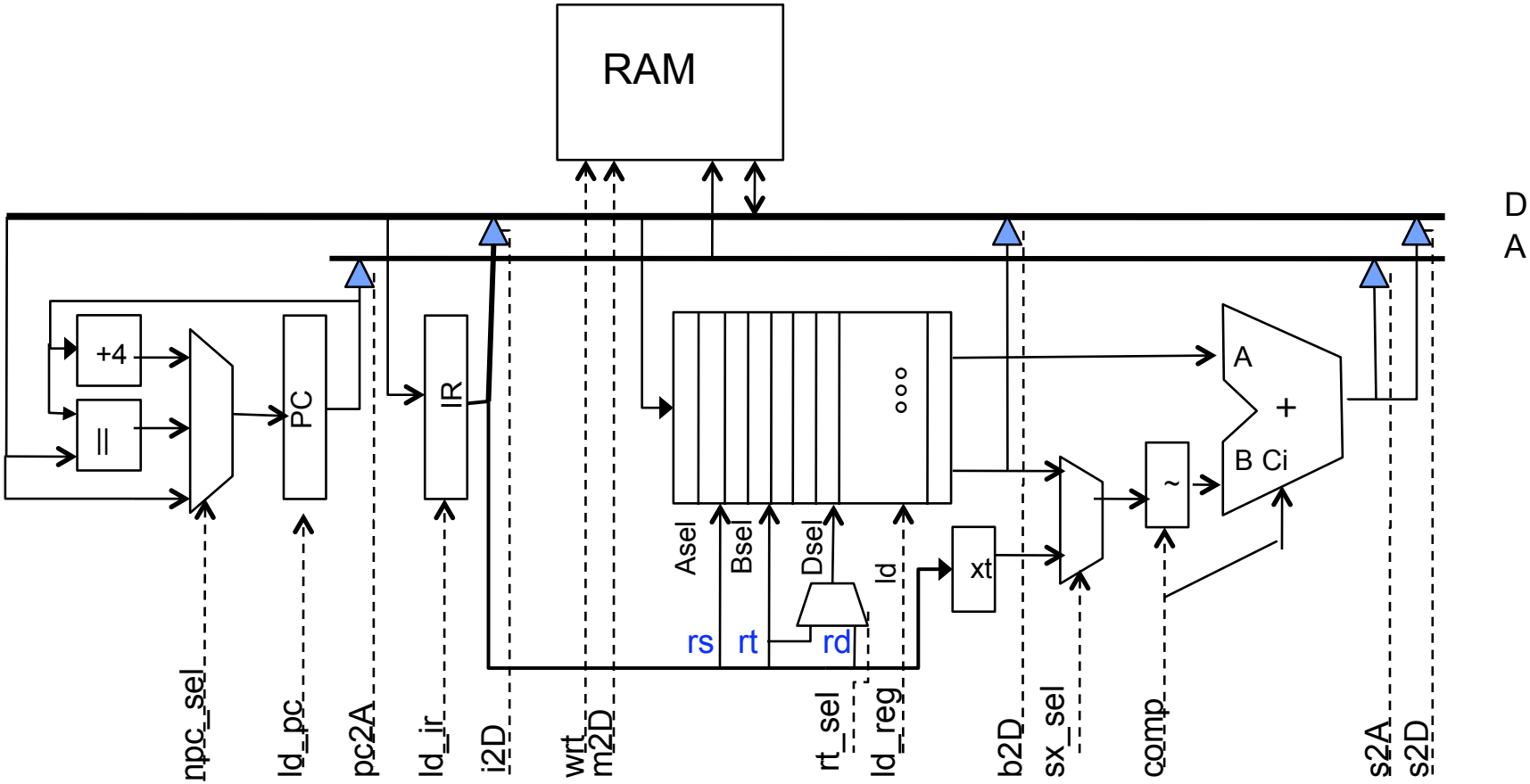
- **`ld_pc`**

Exec: PC := PC+4



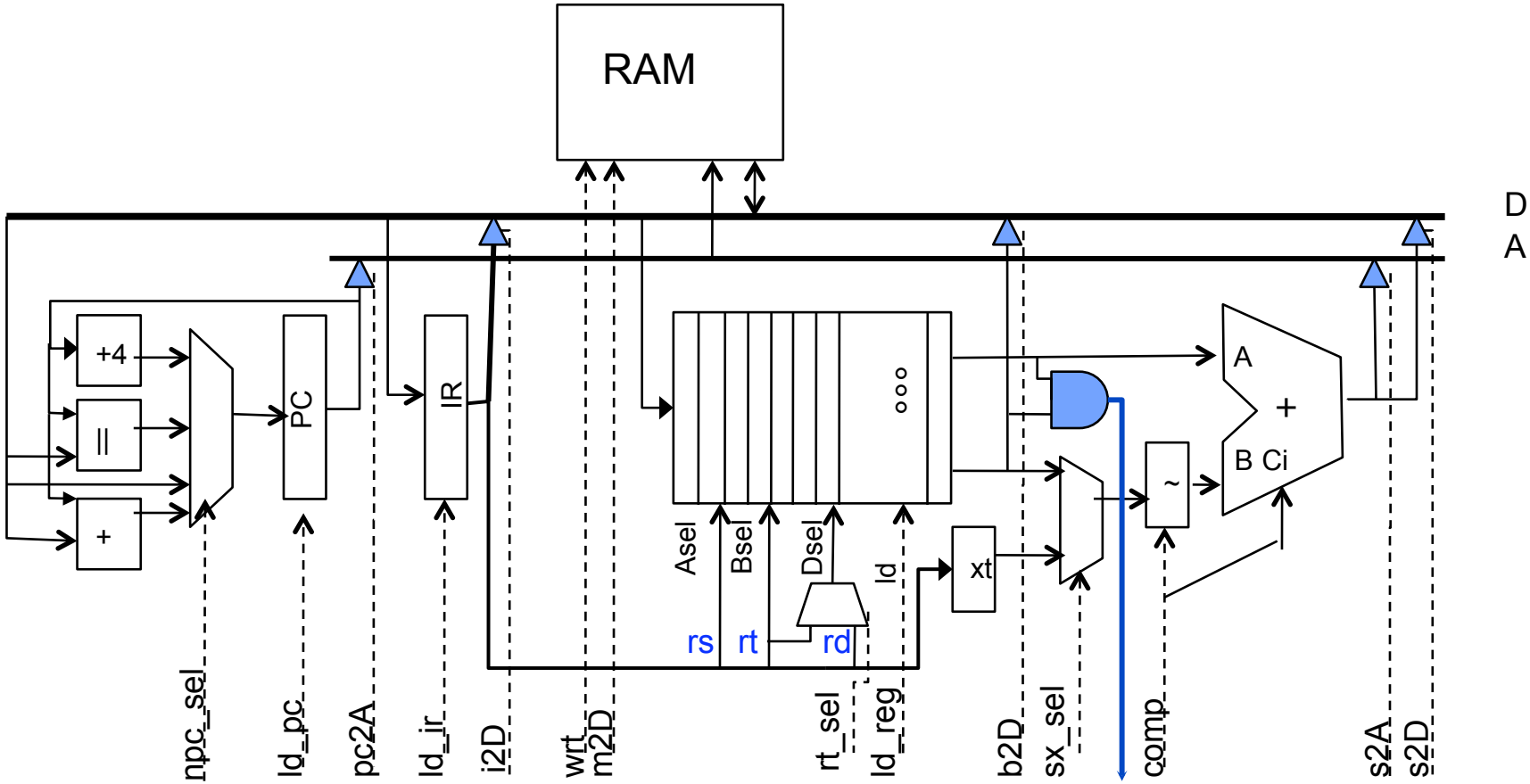
- **`ld_pc`**

Exec: Jump



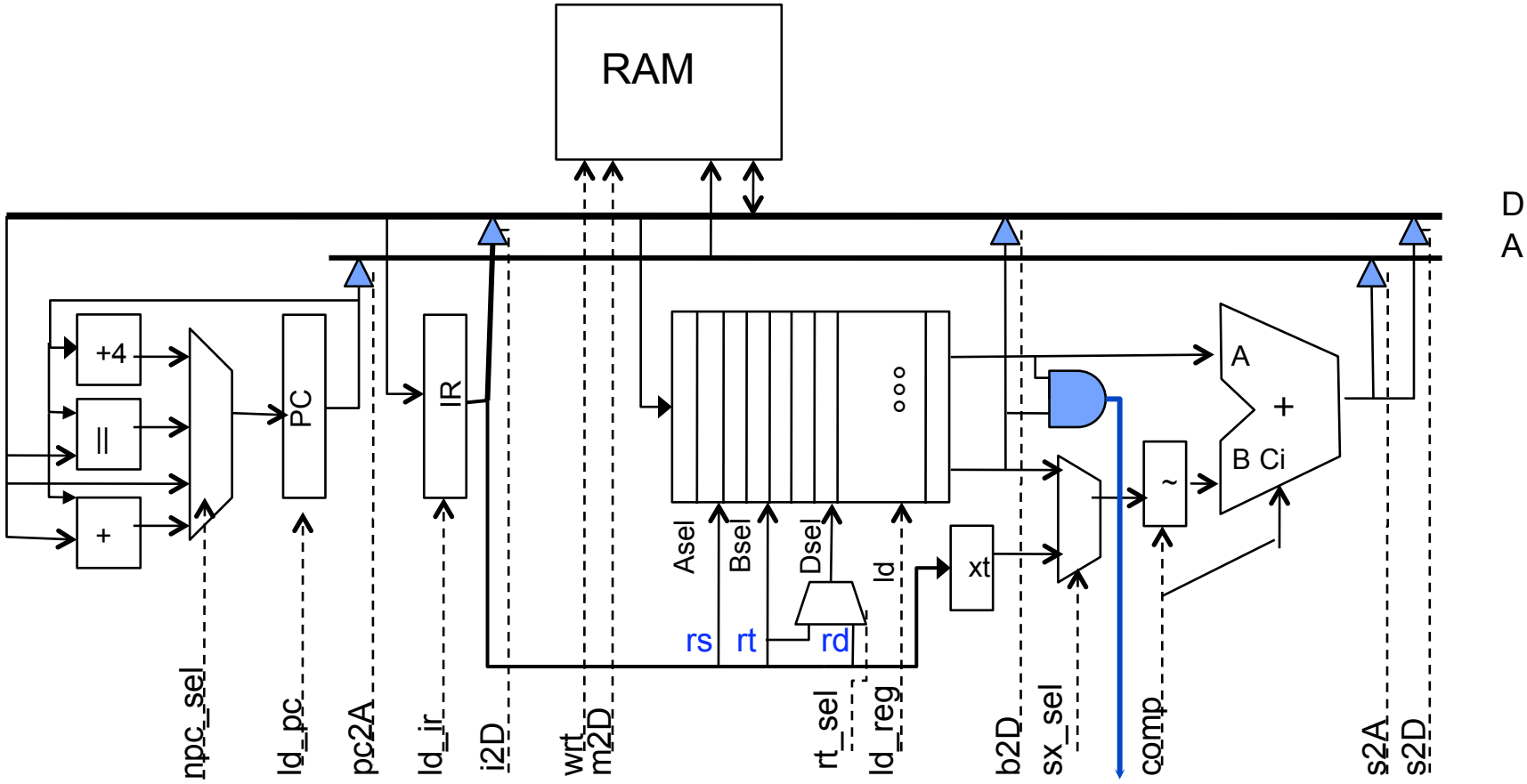
- **j** PC := $PC_{31..28}$ || addr || 00
- **jr** PC := R[rs]

Exec: Branch



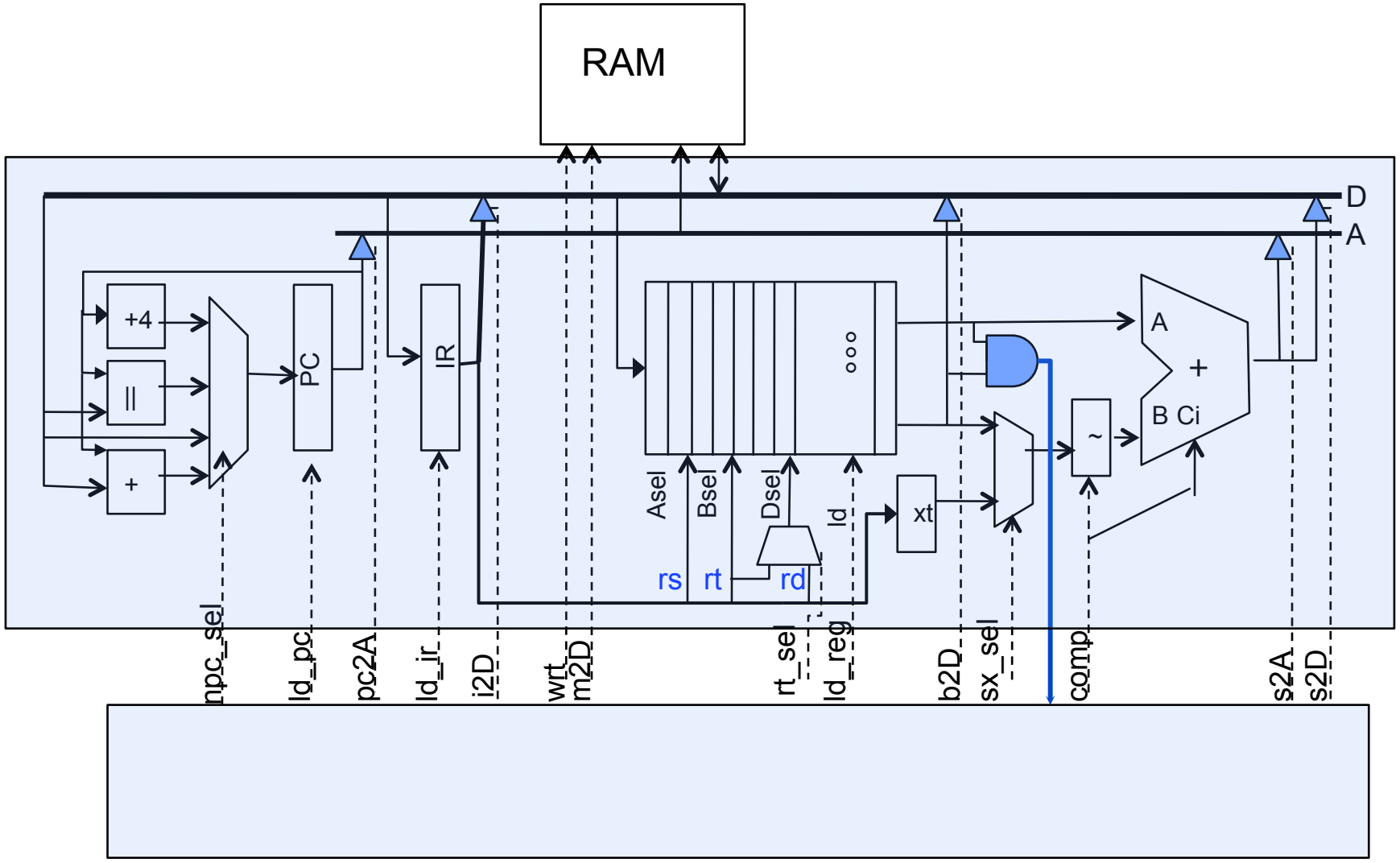
- **BEQ** PC := (R[rs] == R[rt]) ? PC + signEx(im16) : PC+4
- **BLTZ** PC := (R[rs] < 0) ? PC + signEx(im16) : PC+4

Exec: Branch



- PC := (R[rs] == R[rt]) ? PC + signEx(im16) : PC+4
  - nPC_sel = EQ ? pcAdd : PCInc
- **BLTZ** PC := (R[rs] < 0) ? PC + signEx(im16) : PC+4

DataPath + Control



Levels of Design Representation