Microprocessor Implementation using Xilinx FPGA

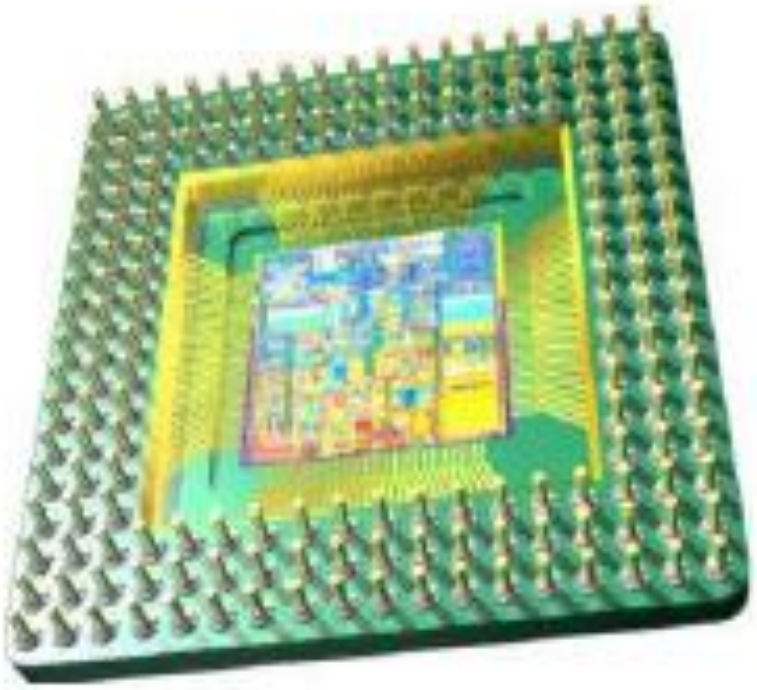
Agenda

- ✓ introduction
- ✓ Architecture
- ✓ Microarchitecture
- ✓ Data path design
- ✓ Control unit design

Agenda

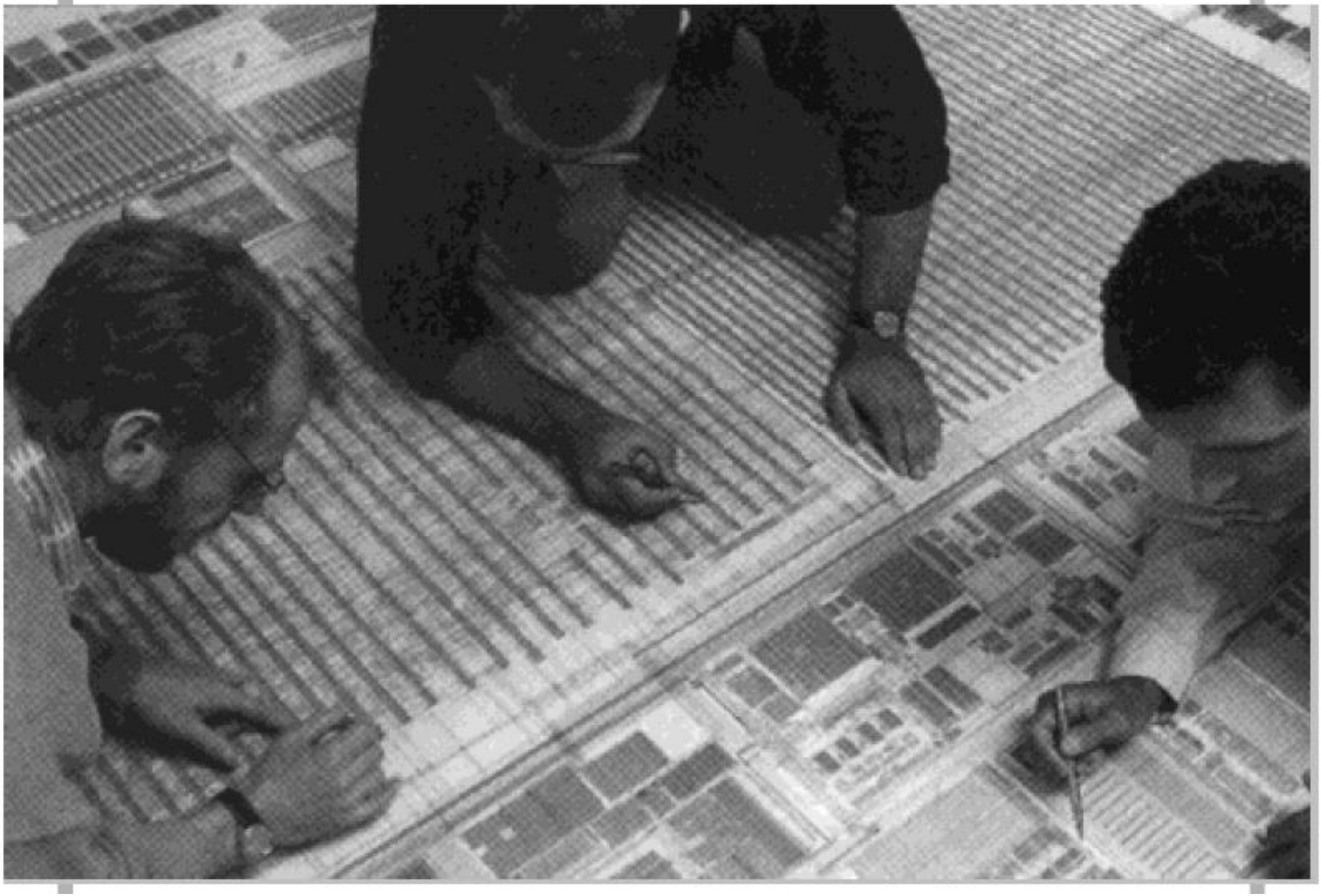
- ✓ introduction
- ✓ Architecture
- ✓ Microarchitecture
- ✓ Data path design
- ✓ Control unit design

How To Design This ?



Microprocessors

Early,



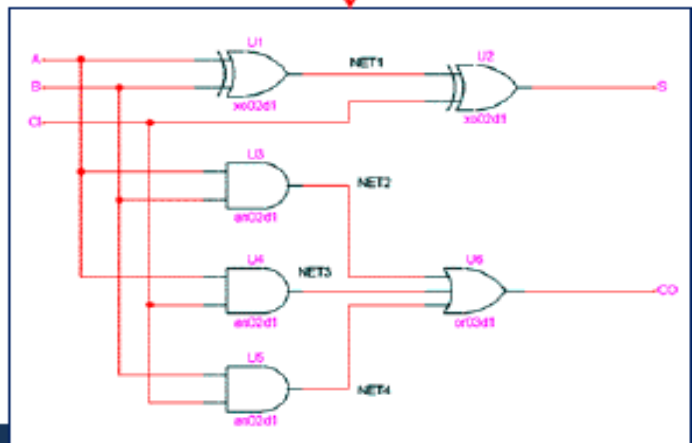
Now, using CAD Tools (Front End)

```
module seq1101_mealy(x, y, CLK, RESET);
    input x;
    input CLK;
    input RESET;
    output y;
    reg y;

    parameter start = 2'b00, got1 = 2'b01, got

    reg [1:0] Q;     // state variables
    reg [1:0] D;     // next state logic output

    // next state logic
    always @(x or Q)
    begin
        y = 0;
        case(Q)
        start: D = x ? got1  : start;
```

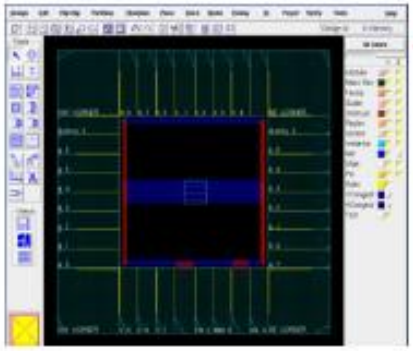
Translation

$$P = a\,x + a\,y + c$$
$$Q = x + y + z$$

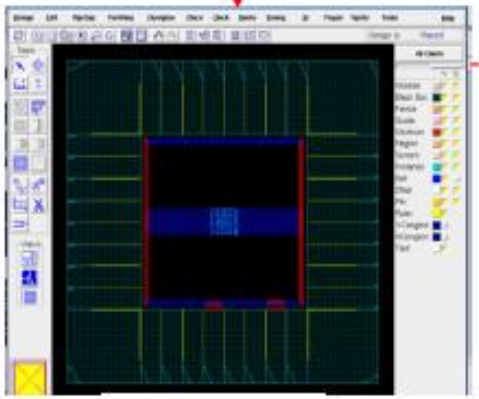
Mapping



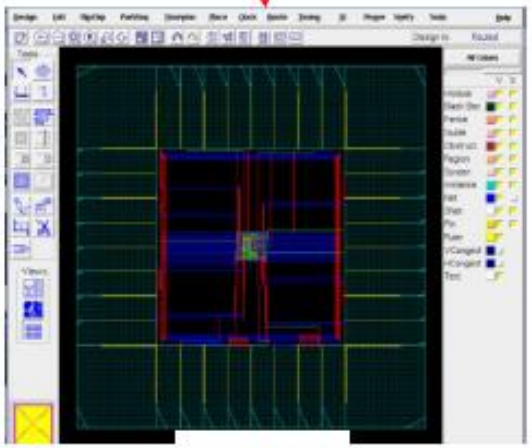
And (Back End)



Floor planning

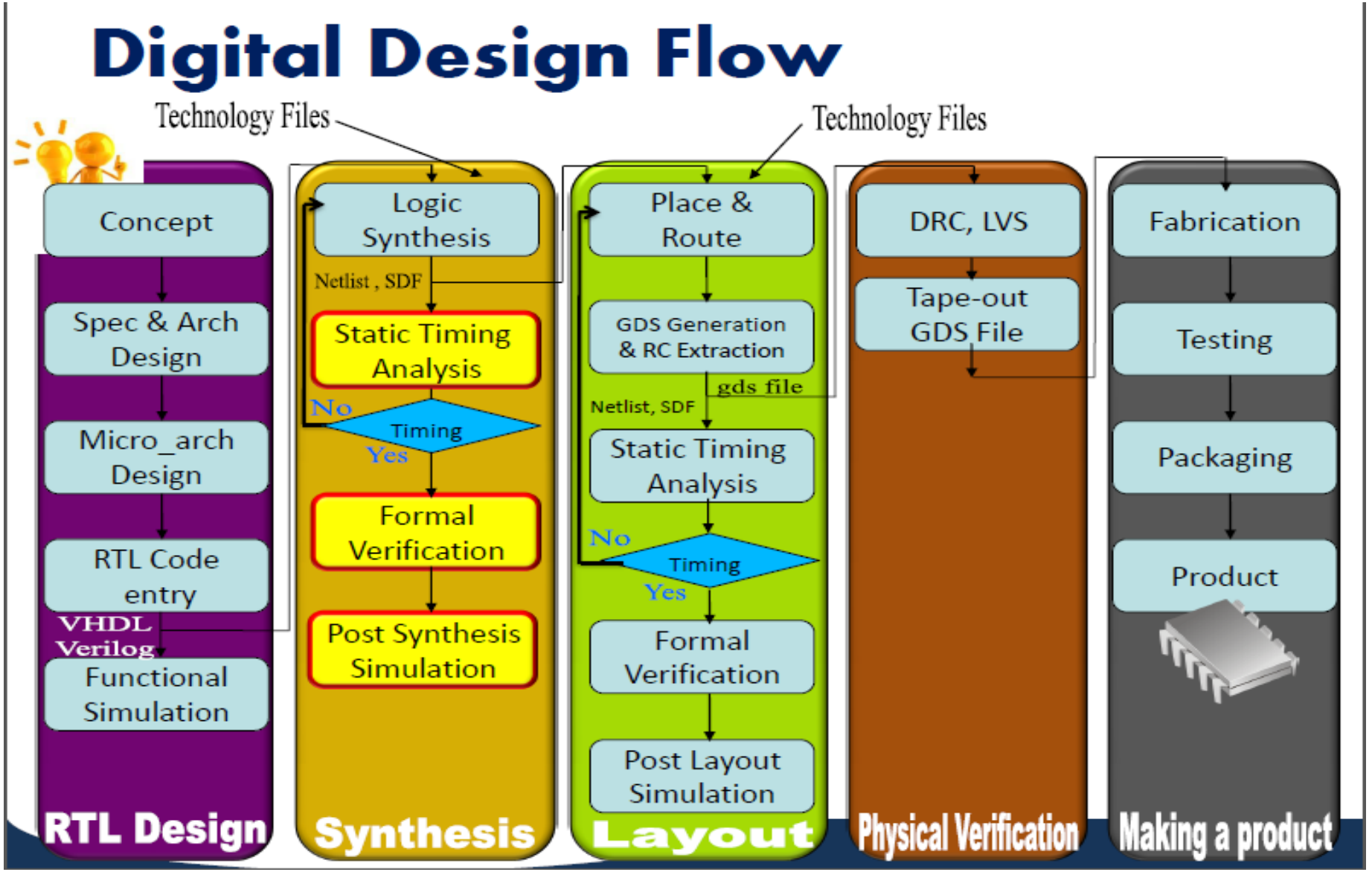


Placement

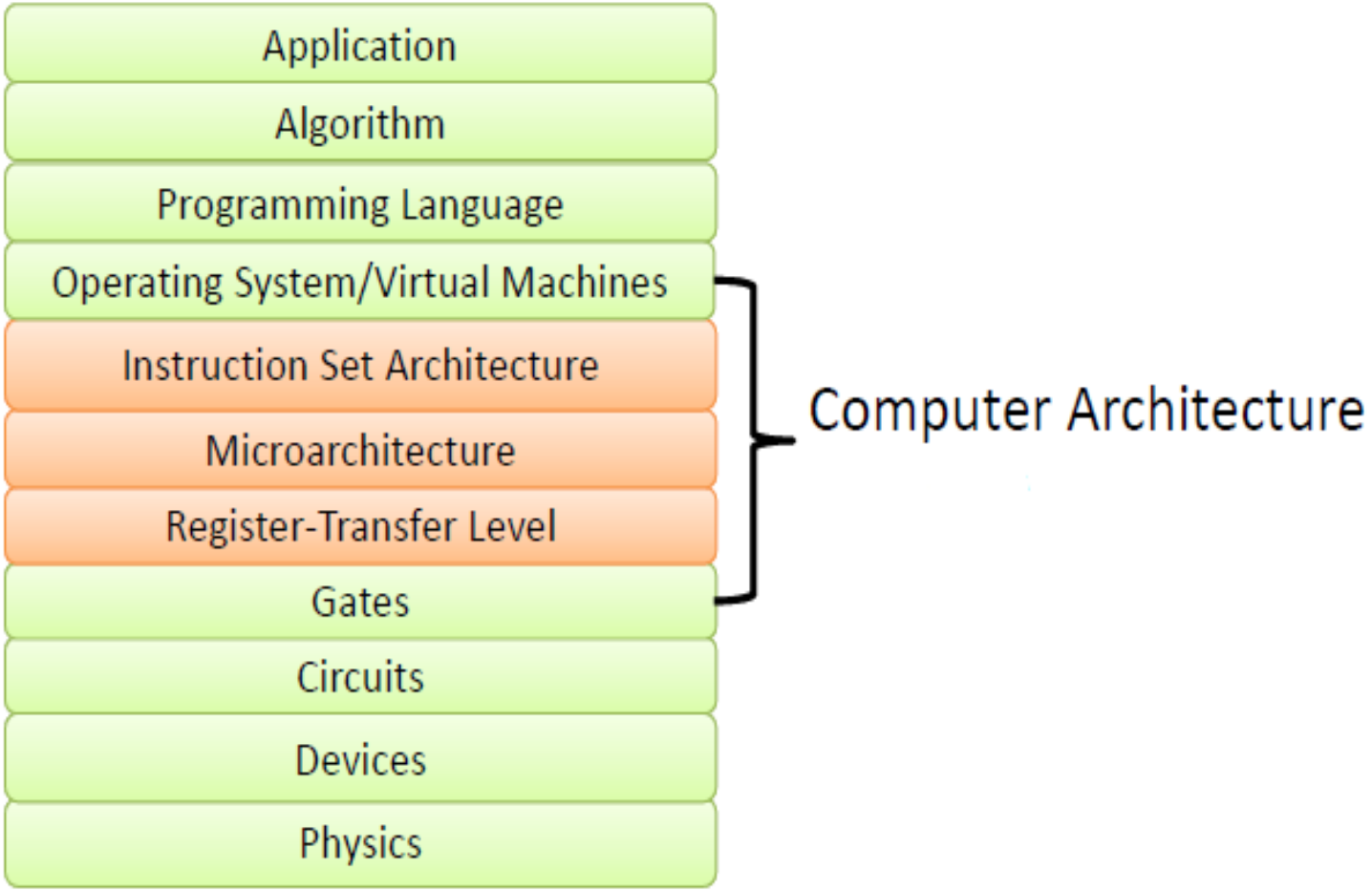


Routing

- Microprocessor is the main application of the digital design .
  So, let's talk about digital design flow .



Abstractions in Modern Computing Systems



Architecture vs. Microarchitecture

"Architecture"/Instruction Set Architecture:

- Programmer visible state (Memory & Register)
- Operations (Instructions and how they work)
- Execution Semantics (interrupts)
- Input/Output
- Data Types/Sizes

Microarchitecture/Organization:

- Tradeoffs on how to implement ISA for some metric (Speed, Energy, Cost)
- Examples: Pipeline depth, number of pipelines, cache size, silicon area, peak power, execution ordering, bus widths, ALU widths

Same Architecture
Different Microarchitecture

AMD Phenom X4

- X86 Instruction Set
- Quad Core
- 125W
- Decode 3 Instructions/Cycle/Core
- 64KB L1 I Cache, 64KB L1 D Cache
- 512KB L2 Cache
- Out-of-order
- 2.6GHz

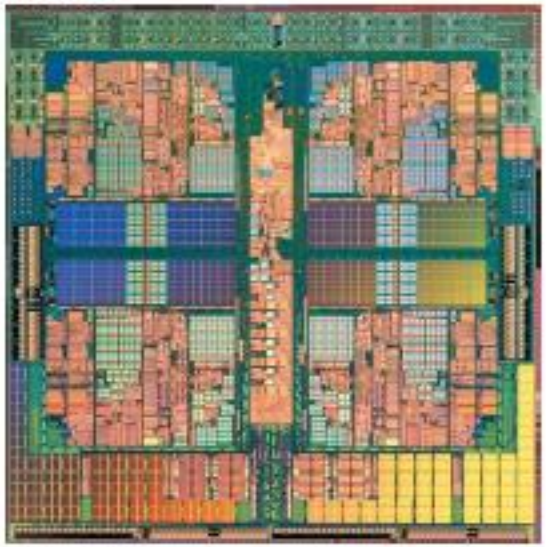


Image Credit: AMD

Intel Atom

- X86 Instruction Set
- Single Core
- 2W
- Decode 2 Instructions/Cycle/Core
- 32KB L1 I Cache, 24KB L1 D Cache
- 512KB L2 Cache
- In-order
- 1.6GHz

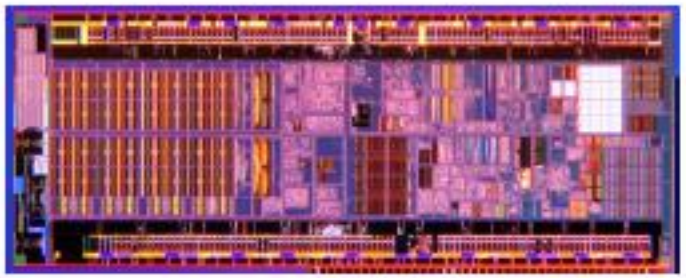


Image Credit: Intel

Agenda

- ✓ Introduction
- ✓ Architecture
- ✓ Microarchitecture
- ✓ Data path design
- ✓ Control unit design

FEE SPARC

- FEE SPARC is a 32-bit single cycle Microprocessor which is based on SPARC Architecture.
- SPARC stands for Scalable Processor ARChitecture which is designed by Berkeley University.
- SPARC derived from Reduced Instruction Set Computer "RISC".

Features

- 32-bit address space.
- Few and simple instruction formats.
- Few addressing modes
- Triadic register addresses
- A large “windowed” register file

Registers

General-purpose registers "IU *r* Registers"

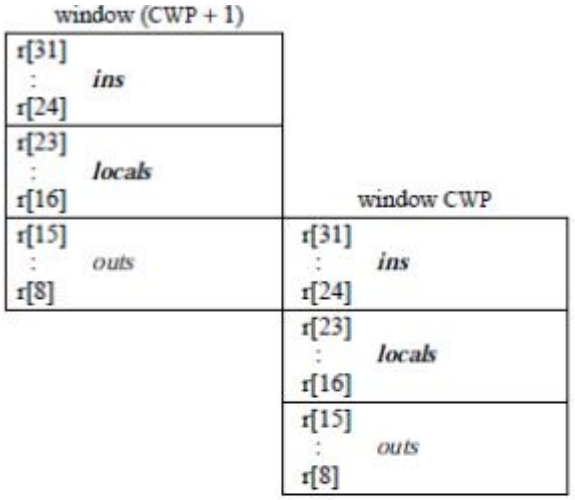
- The IU contains 40 general-purpose 32-bit ***r*** registers.
- They are partitioned into 8 ***global*** registers, plus 2-register ***sets***. A register set is further partitioned into 8 ***in*** registers and 8 ***local*** registers.

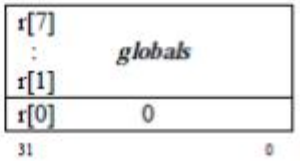
Window Addressing

Windowed Register Address	*r* Register Address
in[0] – in[7]	r[24] – r[31]
local[0] – local[7]	r[16] – r[23]
out[0] – out[7]	r[8] – r[15]
global[0] – global[7]	r[0] – r[7]

Registers

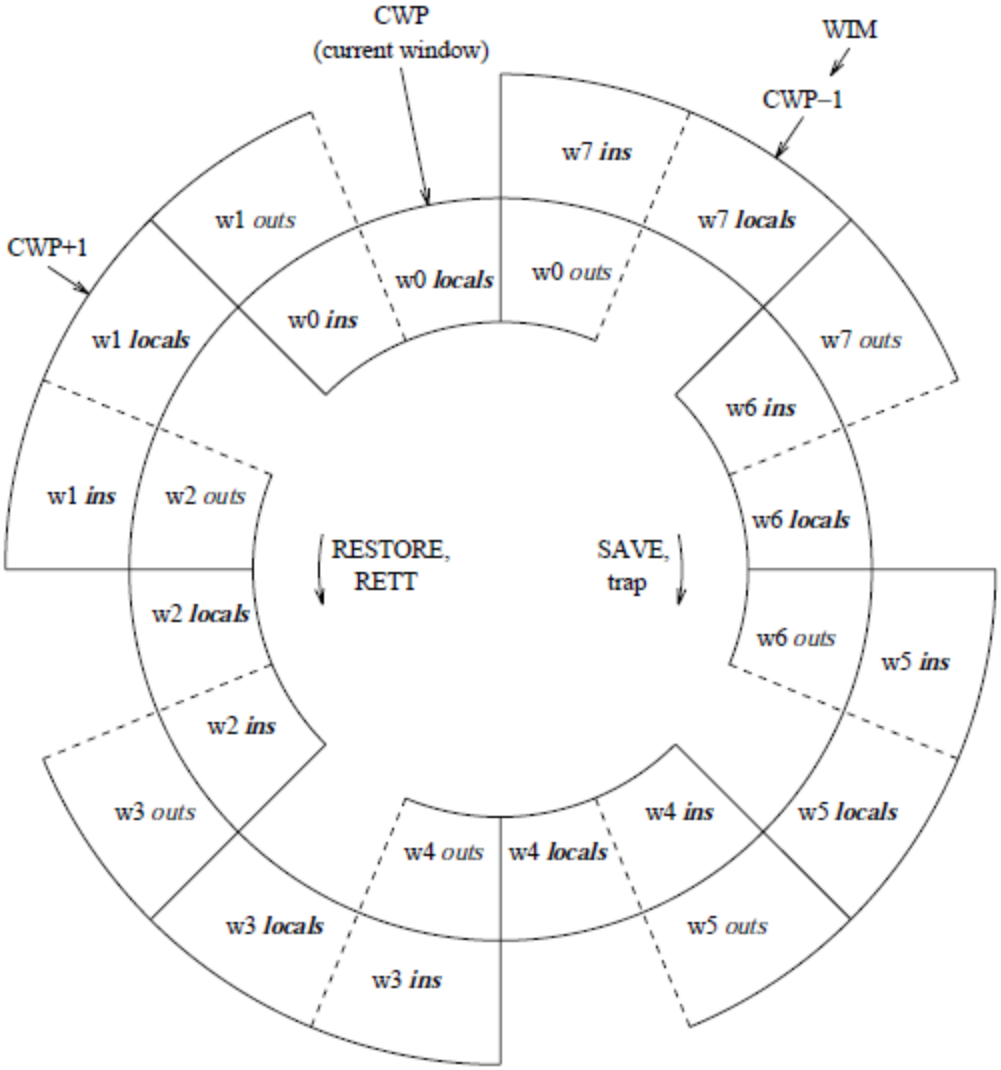
Windowed *r* Registers





Registers

Overlapping of windows



The Windowed r Registers

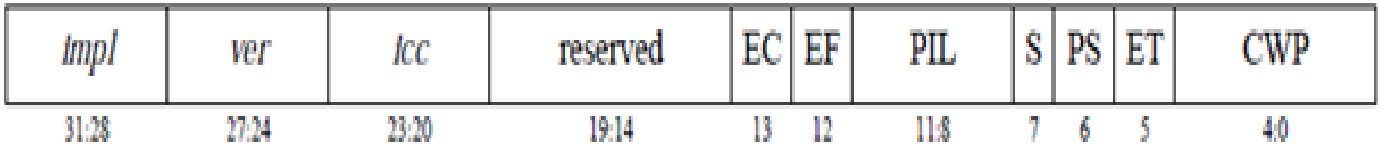
❑ IU Control/Status Registers

The 32-bit IU control/status registers include the Processor State Register (PSR), the Window Invalid Mask register (WIM), the Trap Base Register (TBR), the program counters (PC), and optional, implementation-dependent Ancillary State Registers (ASRs).

❑ Processor State Register (PSR)

The 32-bit PSR contains various fields that control the processor and hold status information. It can be modified by the SAVE, RESTORE, Ticc, and RETT instructions, and by all instructions that modify the condition codes. The privileged RDPSR and WRPSR instructions read and write the PSR directly.

PSR Fields



PSR_integer_cond_codes (*icc*)

Bits 23 through 20 are the IU's condition codes. These bits are modified by the arithmetic and logical instructions whose names end with the letters **cc** (e.g. ANDcc), and by the WRPSR instruction.

The Bicc and Ticc instructions cause a transfer of control based on the value of these bits, which are defined as follows:

Integer Condition Codes (icc) Fields of the PSR

n	*z*	*v*	*c*
23	22	21	20

PSR_negative (*n*)

Bit 23 indicates whether the 32-bit 2's complement ALU result was negative for the last instruction that modified the *icc* field. 1 = negative, 0 = not negative.

PSR_zero (*z*)

Bit 22 indicates whether the 32-bit ALU result was zero for the last instruction that modified the *icc* field. 1 = zero, 0 = nonzero.

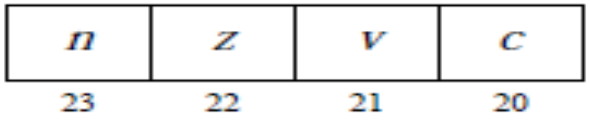
PSR_overflow (*v*)

Bit 21 indicates whether the ALU result was within the range of (was representable in) 32-bit 2's complement notation for the last instruction that modified the *icc* field.
1 = overflow, 0 = no overflow.

PSR_carry (*c*)

Bit 20 indicates whether a 2's complement carry out (or borrow) occurred for the last instruction that modified the *icc* field. Carry is set on addition if there is a carry out of bit 31. Carry is set on subtraction if there is borrow into bit 31. 1 =carry, 0 = no carry.

Integer Condition Codes (icc) Fields of the PSR



❑ Instructions

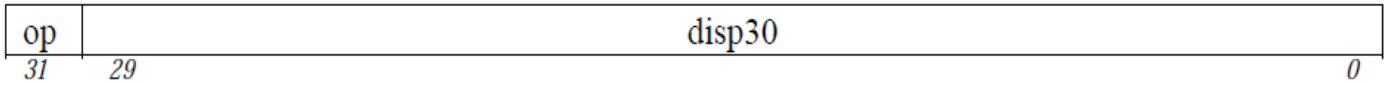
Instructions are encoded in three 32-bit formats and can be partitioned into four general categories.
There are 72 basic instruction operations.

Instructions fall into four basic categories:
1) Load/store
2) Arithmetic/logical/shift
3) Control transfer
4) Read/write control register

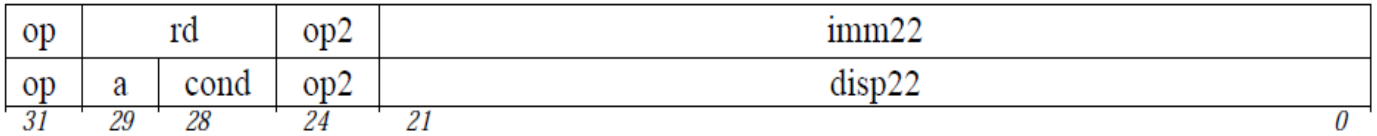
❖ Instruction Formats

Instructions are encoded in three major 32-bit formats.

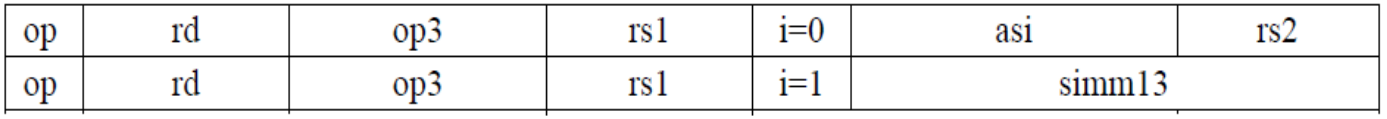
Format 1 (*op* = 1): CALL



Format 2 (*op* = 0): SETHI & Branches (Bicc, FBfcc, CBccc)



Format 3 (*op* = 2 or 3): Remaining instructions



Agenda

- ✓ Introduction
- ✓ Architecture
- ✓ Microarchitecture
- ✓ Data path design
- ✓ Control unit design

❑ Concept

As our implementation is considered as IP so, I will explain the concept to build a micro architecture on a MIPS microprocessor.

❑ Design Process

We will divide our microarchitectures into two interacting parts: the *data path* and the *control*.
we develop three microarchitectures for the MIPS processor architecture: single-cycle, multicycle, and pipelined

MIPS is a 32-bit architecture, so we will use a 32-bit data path.

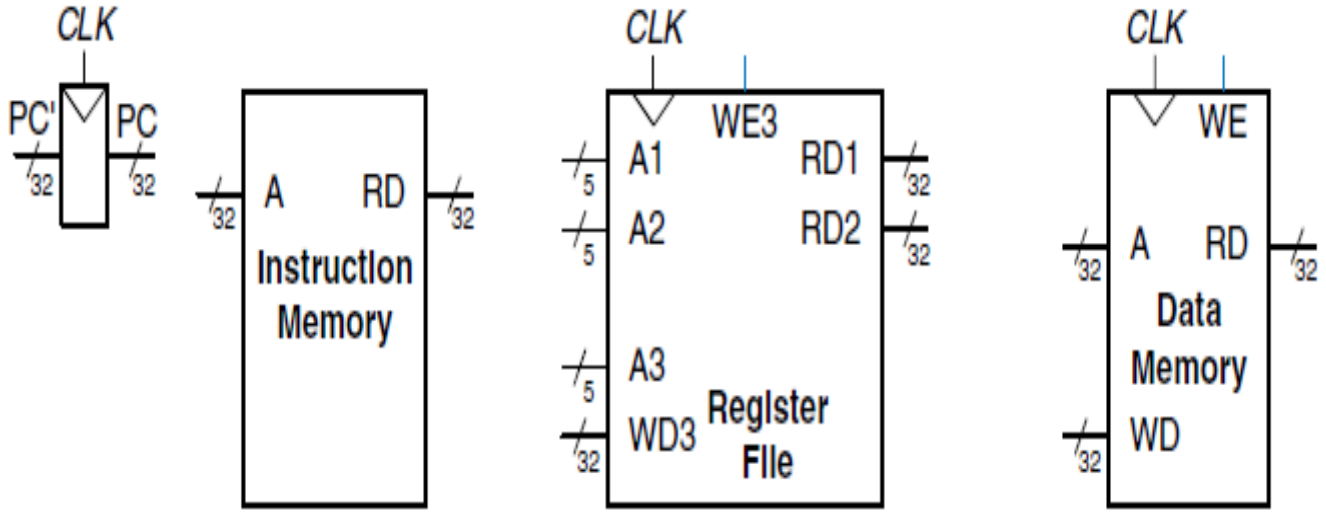
- It contains structures such as memories, registers, ALUs, and multiplexers.

- The control unit receives the current instruction from the data path and tells the data path how to execute that instruction.
- the control unit produces multiplexer select, register enable, and memory write signals To control the operation of the data path.

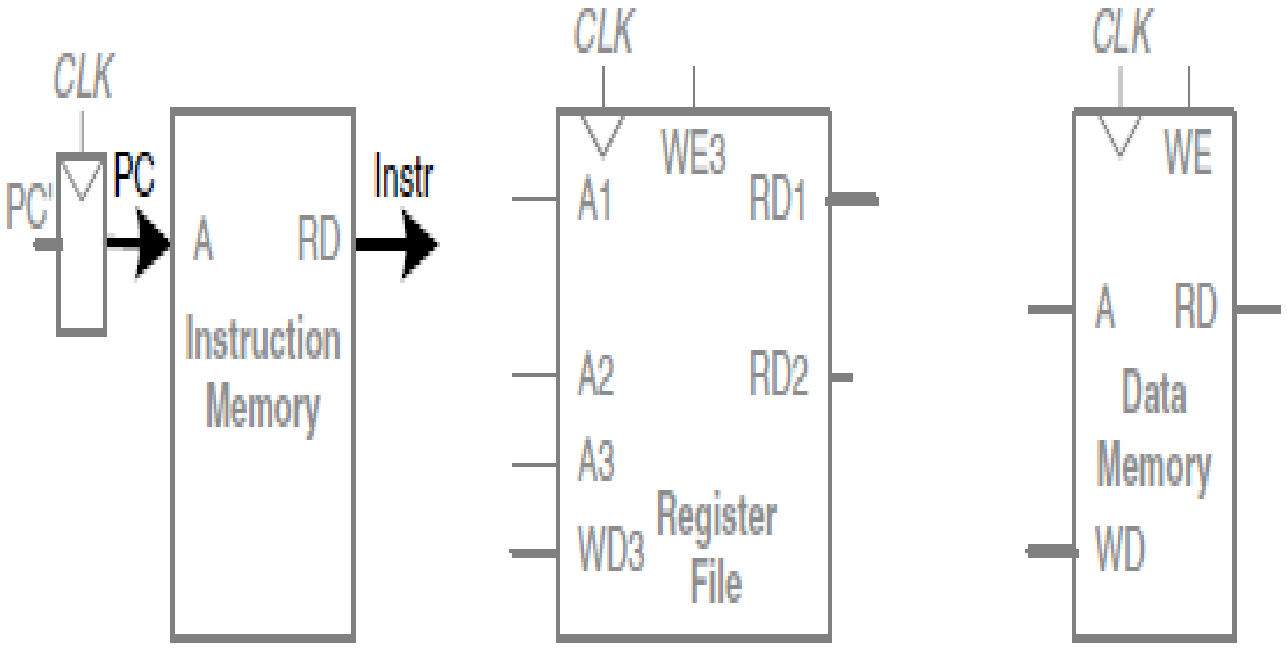
How to manage complexity?

A good way to design a complex system is to start with hardware containing the state elements.

These elements include the memories and the architectural state (the program counter and registers).

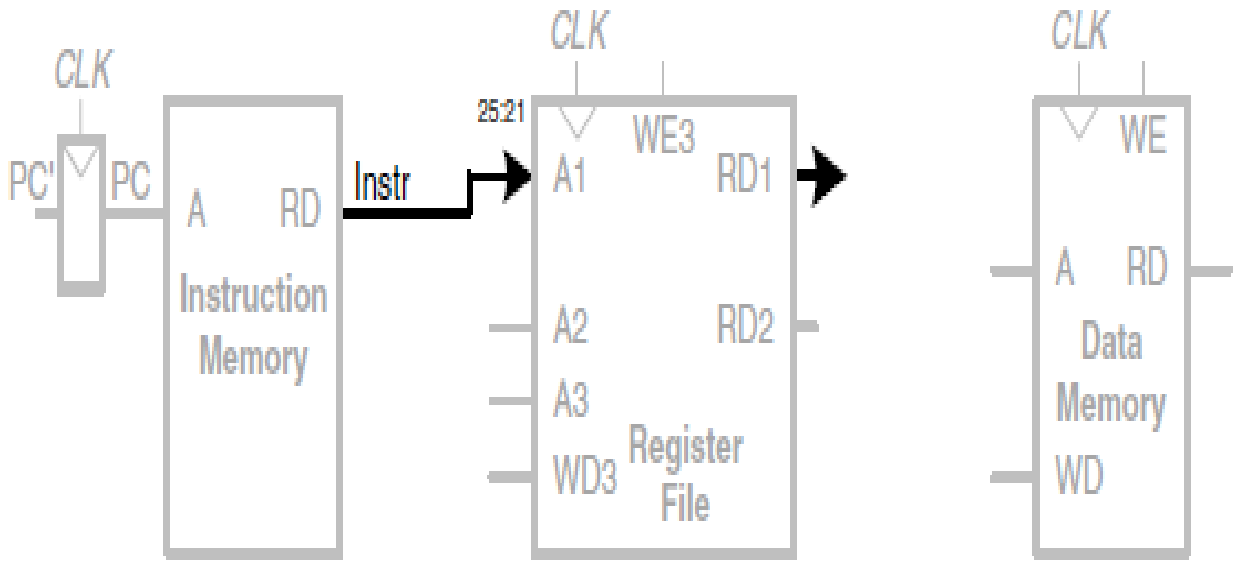


❑ Single-Cycle Data path

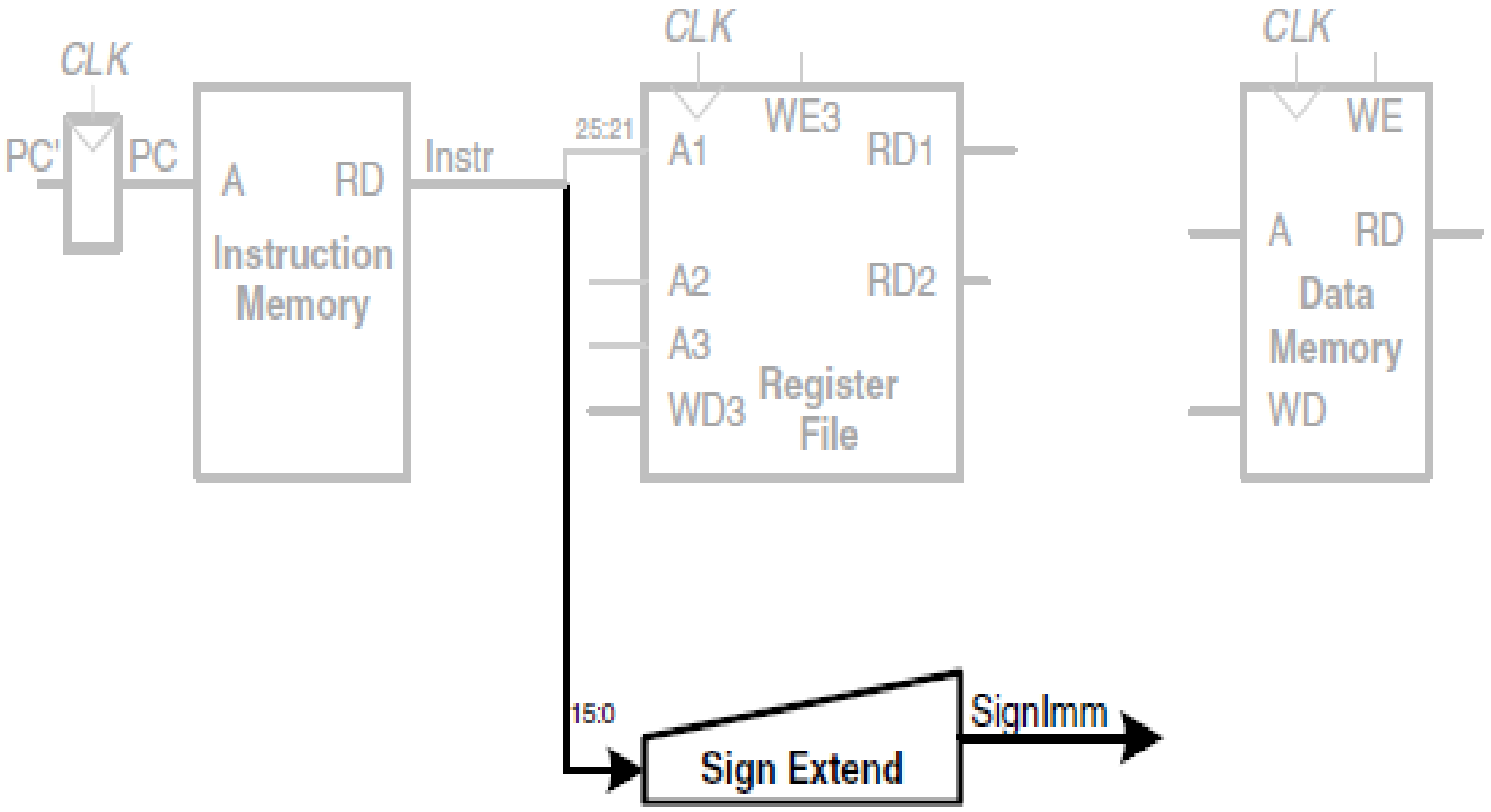


Fetch instruction from memory

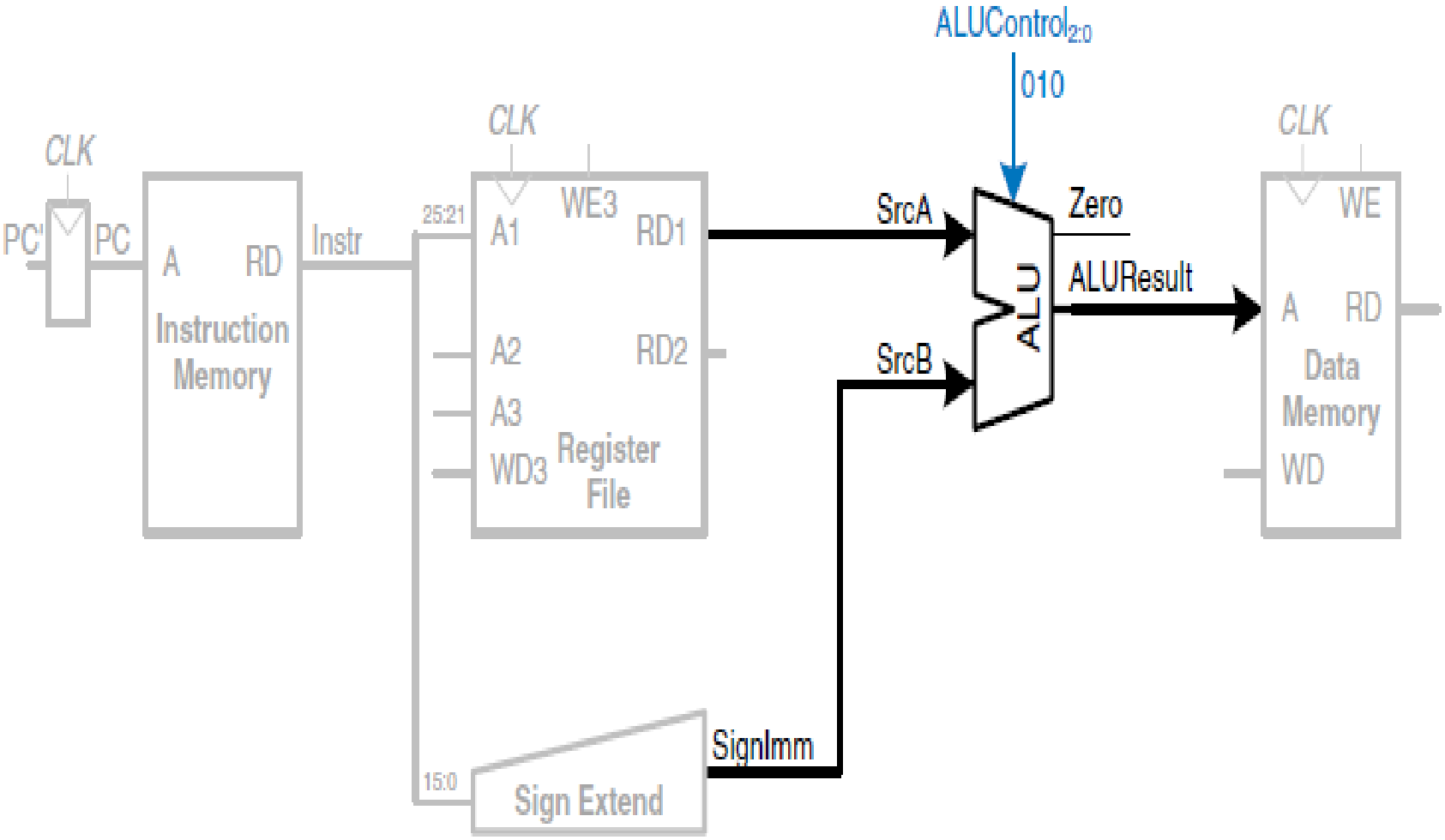
Load instruction



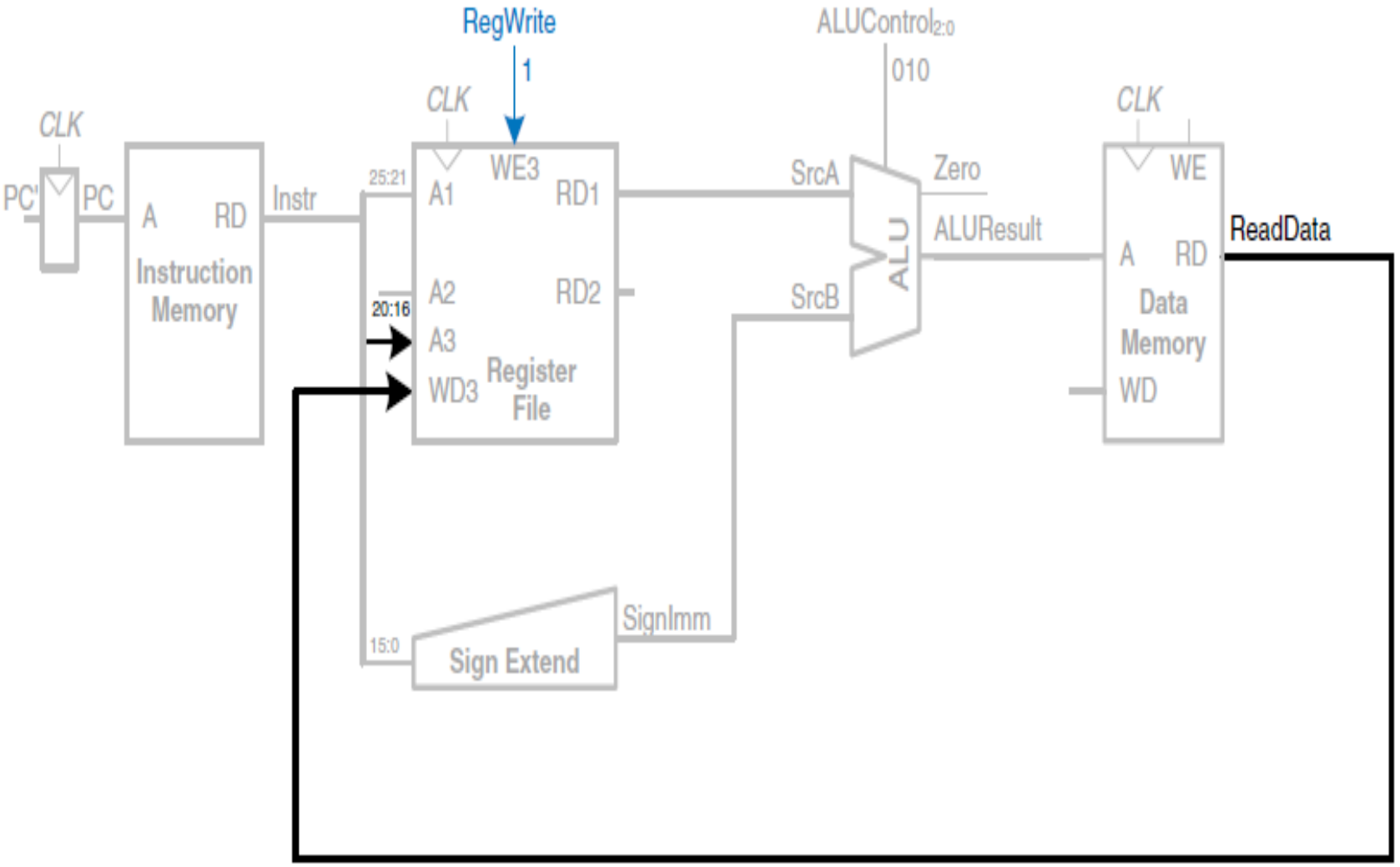
Read source operand from register file



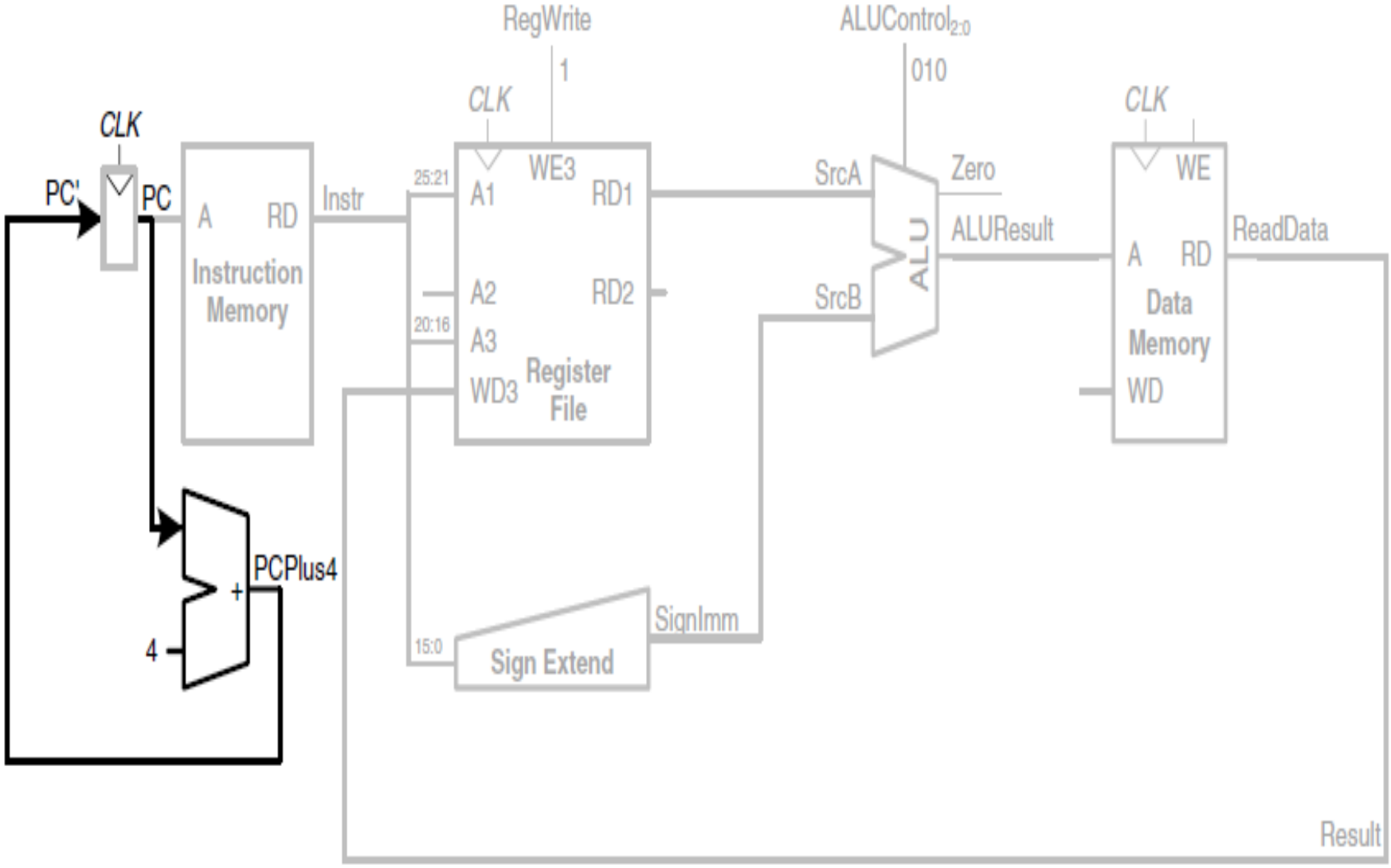
Sign-extend the immediate



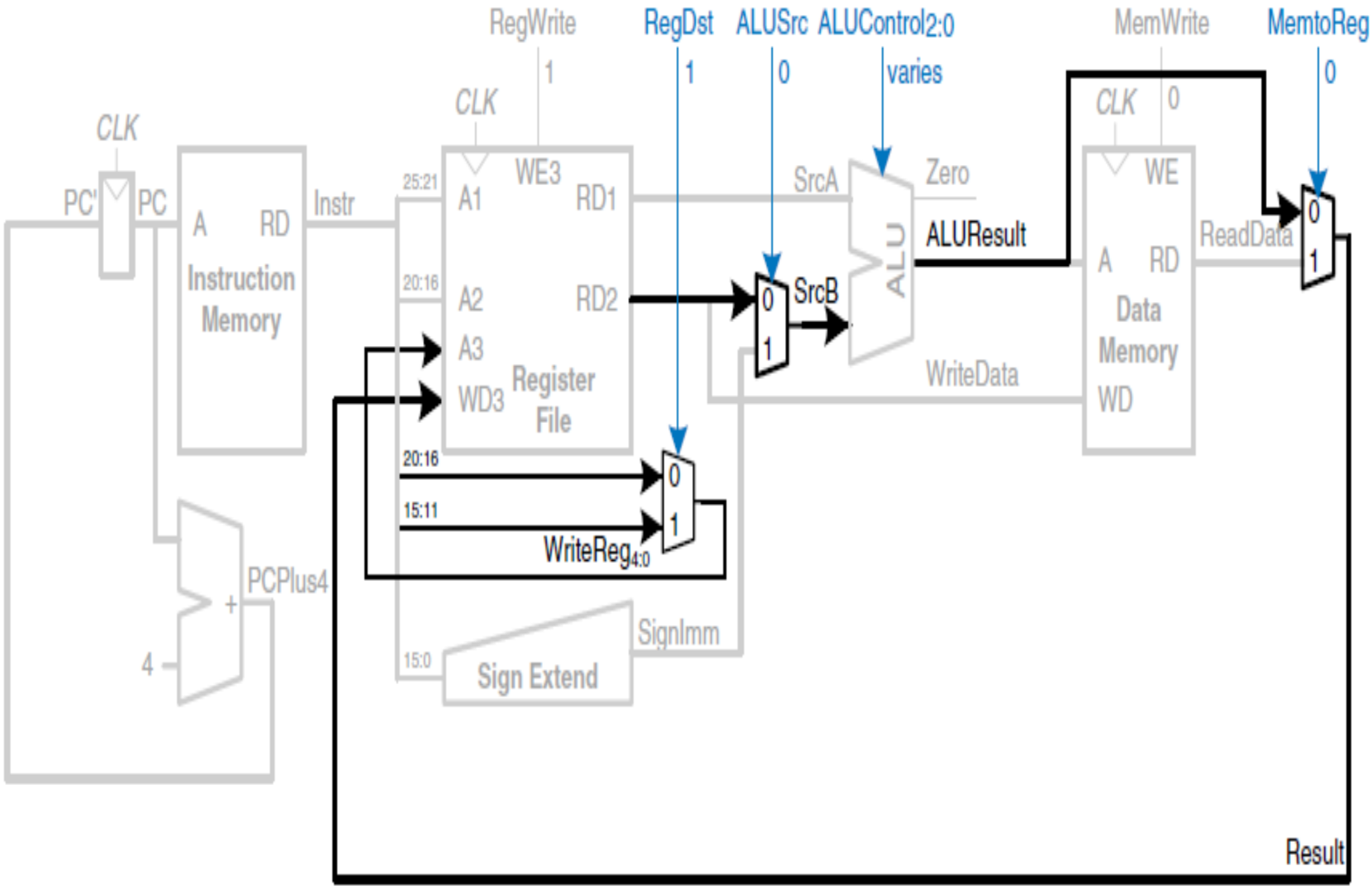
Compute memory address



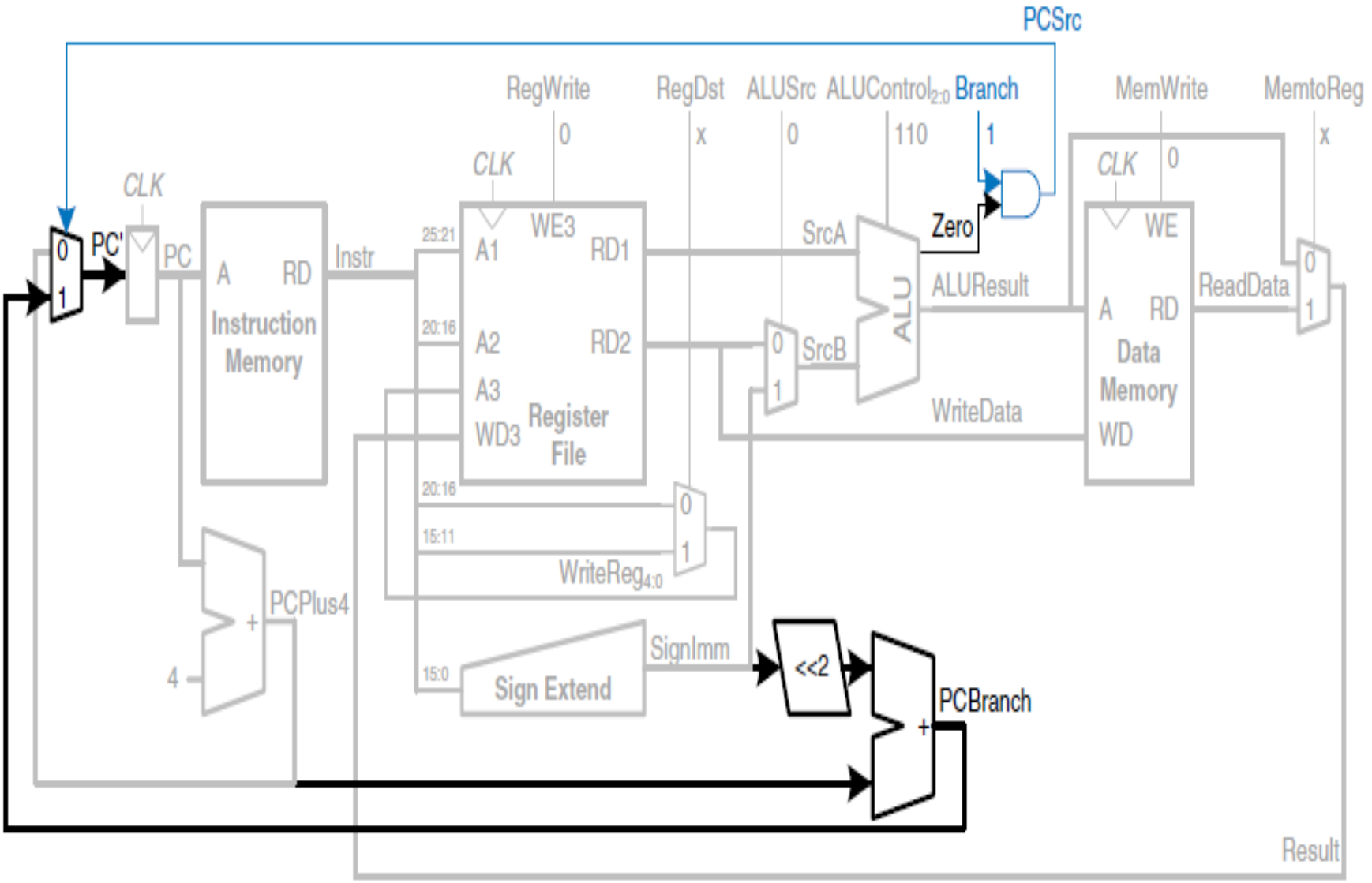
Write data back to register file



Determine address of next instruction for PC

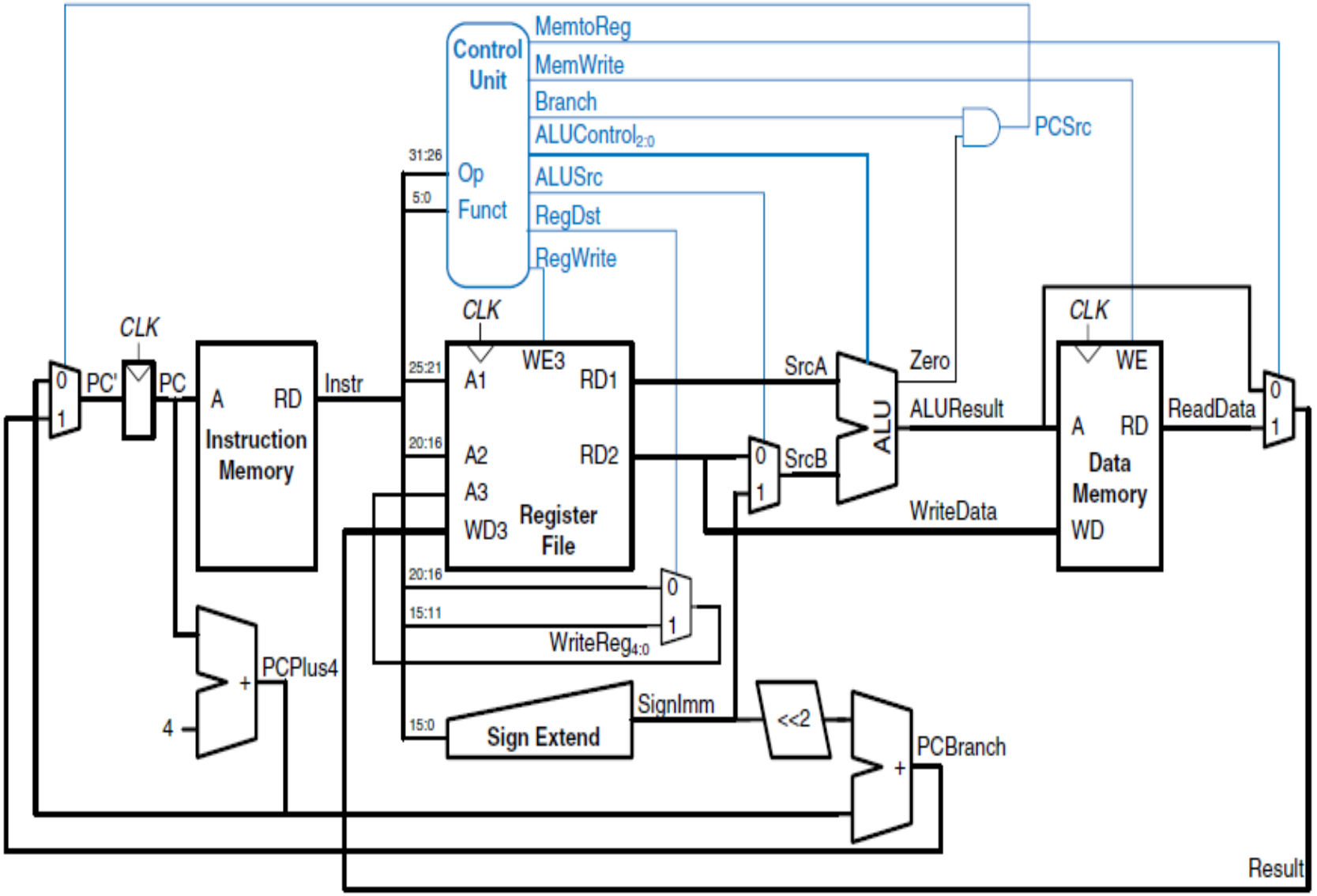


Datapath enhancements for R-type instruction

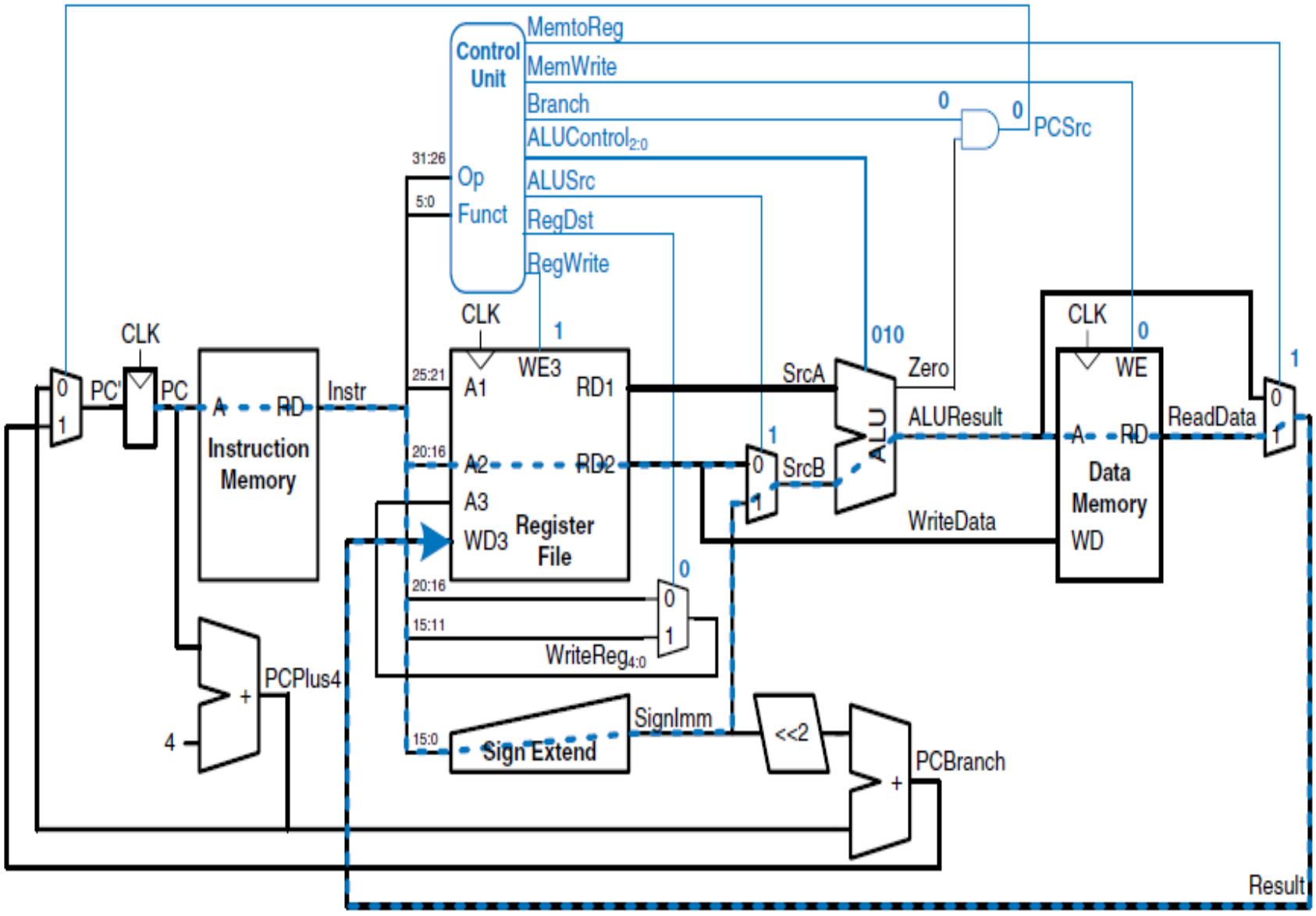


Datapath enhancements for beq instruction

❑ Single-Cycle Control



Complete single-cycle MIPS processor



Critical path for lw instruction