

EECS150 - Digital Design

Lecture 9- CPU Microarchitecture

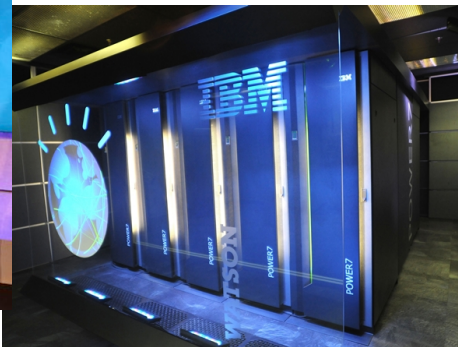
Feb 15, 2011
John Wawrzynek

Spring 2011

EECS150 - Lec09-cpu

Page 1

Watson: Jeopardy-playing Computer



Watson is made up of a cluster of ninety IBM Power 750 servers (plus additional I/O, network and cluster controller nodes in 10 racks) for a total of 2880 POWER7 processor cores and 16 Terabytes of RAM. Each Power 750 server uses a 3.5 GHz POWER7 eight core processor, with four threads per core, and it still takes ~15 seconds a question.

Each core can do 8 double-precision FLOPS/cycle. So total is $2880 \times 3.5 \times 8 > 80$ TFLOPS

Spring 2011

EECS150 - Lec09-cpu

Page 2

Processor Microarchitecture Introduction

Microarchitecture: how to implement an architecture in hardware

Good examples of how to put principles of digital design to practice.

Introduction to final project.

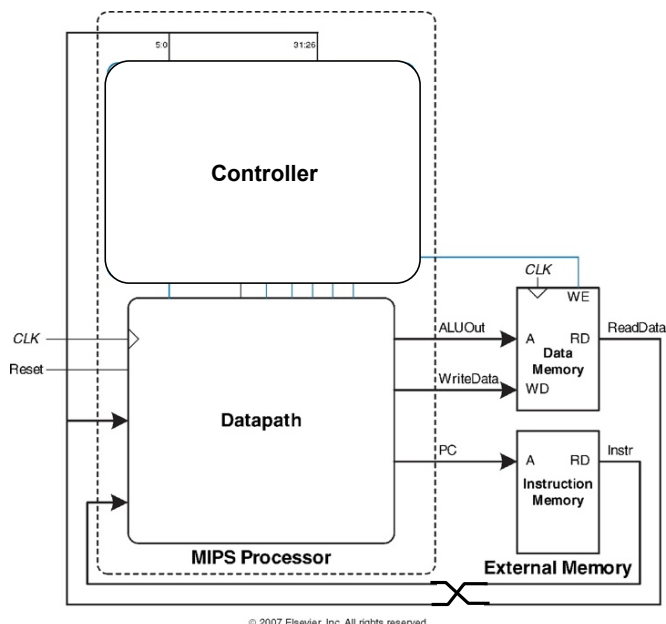
| | |
|----------------------|---------------------------|
| Application Software | programs |
| Operating Systems | device drivers |
| Architecture | instructions registers |
| Micro-architecture | datapaths controllers |
| Logic | adders memories |
| Digital Circuits | AND gates NOT gates |
| Analog Circuits | amplifiers filters |
| Devices | transistors diodes |
| Physics | electrons |

MIPS Processor Architecture

- For now we consider a subset of MIPS instructions:
 - R-type instructions: **and**, **or**, **add**, **sub**, **slt**
 - Memory instructions: **lw**, **sw**
 - Branch instructions: **beq**
- Later we'll add **addi** and **j**

MIPS Micrarchitecture Organization

Datapath + Controller + External Memory



Spring 2011

© 2007 Elsevier, Inc. All rights reserved.

Page 5

How to Design a Processor: step-by-step

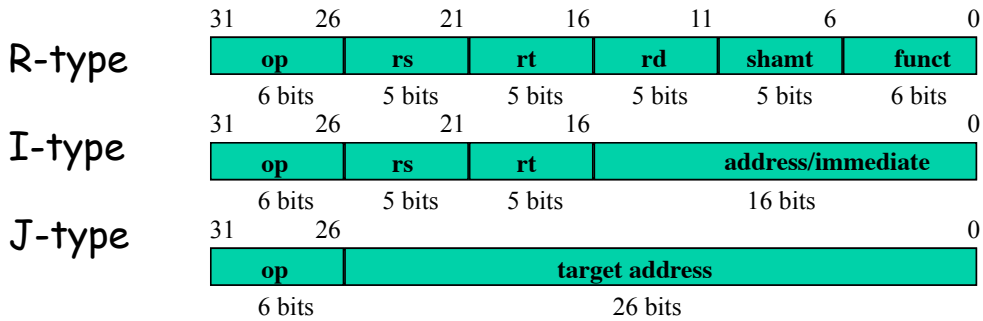
1. Analyze instruction set architecture (ISA) \Rightarrow datapath [requirements](#)
 - meaning of each instruction is given by the *data transfers (register transfers)*
 - datapath must include storage element for ISA registers
 - datapath must support each data transfer
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting requirements
4. Analyze implementation of each instruction to determine setting of control points that effects the data transfer.
5. Assemble the control logic.

Spring 2011

EECS150 - Lec09-cpu

Page 6

Review: The MIPS Instruction



The different fields are:

op: operation ("opcode") of the instruction

rs, rt, rd: the source and destination register specifiers

shamt: shift amount

funct: selects the variant of the operation in the "op" field

address / immediate: address offset or immediate value

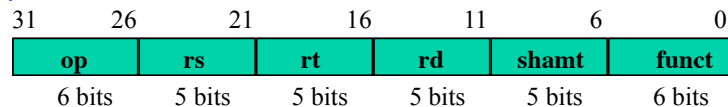
target address: target address of jump instruction

Subset for Lecture

add, sub, or, slt

• **addu rd,rs,rt**

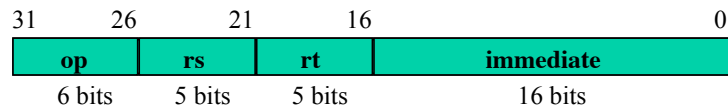
• **subu rd,rs,rt**



lw, sw

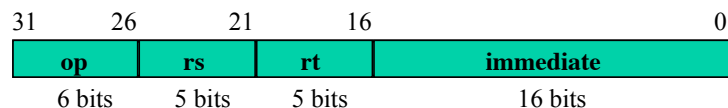
• **lw rt,rs,imm16**

• **sw rt,rs,imm16**



beq

• **beq rs,rt,imm16**



Register Transfer Descriptions

All start with instruction fetch:

{op , rs , rt , rd , shamt , funct} ← IMEM[PC] OR
{op , rs , rt , Imm16} ← IMEM[PC] THEN

inst Register Transfers

add $R[rd] \leftarrow R[rs] + R[rt];$ $PC \leftarrow PC + 4$
sub $R[rd] \leftarrow R[rs] - R[rt];$ $PC \leftarrow PC + 4$
or $R[rd] \leftarrow R[rs] | R[rt];$ $PC \leftarrow PC + 4$
slt $R[rd] \leftarrow (R[rs] < R[rt]) ? 1 : 0;$ $PC \leftarrow PC + 4$
lw $R[rt] \leftarrow DMEM[R[rs] + \text{sign_ext}(Imm16)];$ $PC \leftarrow PC + 4$
sw $DMEM[R[rs] + \text{sign_ext}(Imm16)] \leftarrow R[rt];$ $PC \leftarrow PC + 4$
beq if ($R[rs] == R[rt]$) then $PC \leftarrow PC + 4 + \{\text{sign_ext}(Imm16), 00\}$
 else $PC \leftarrow PC + 4$

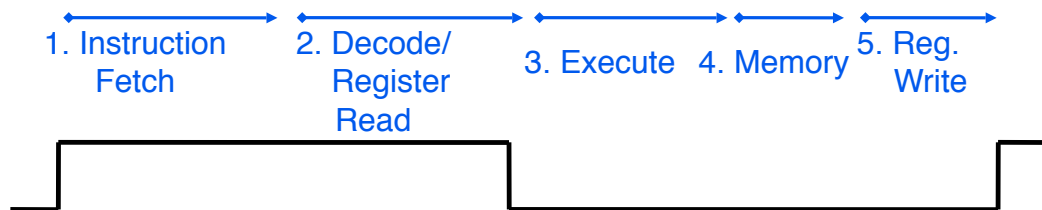
Microarchitecture

Multiple implementations for a single architecture:

- Single-cycle
 - Each instruction executes in a single clock cycle.
- Multicycle
 - Each instruction is broken up into a series of shorter steps with one step per clock cycle.
- Pipelined (variant on "multicycle")
 - Each instruction is broken up into a series of steps with one step per clock cycle
 - Multiple instructions execute at once.

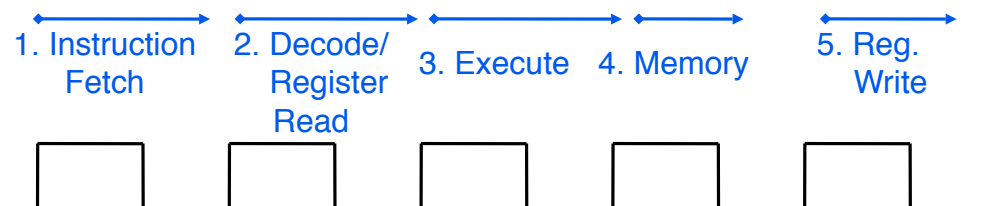
CPU clocking (1/2)

- Single Cycle CPU: All stages of an instruction are completed within one *long* clock cycle.
 - The clock cycle is made sufficient long to allow each instruction to complete all stages without interruption and within one cycle.



CPU clocking (2/2)

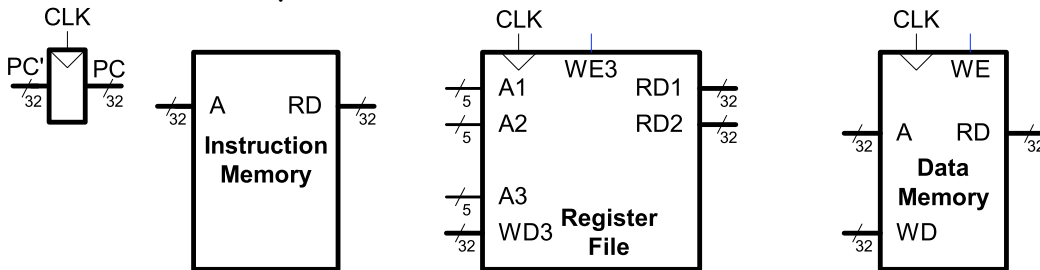
- Multiple-cycle CPU: Only one stage of instruction per clock cycle.
 - The clock is made as long as the slowest stage.



Several significant advantages over single cycle execution: Unused stages in a particular instruction can be skipped OR instructions can be pipelined (overlapped).

MIPS State Elements

- Determines everything about the execution status of a processor:
 - PC register
 - 32 registers
 - Memory



Note: for these state elements, clock is used for write but not for read (asynchronous read, synchronous write).

Single-Cycle Datapath: 1w fetch

- First consider executing 1w

$$R[rt] \leftarrow \text{DMEM}[R[rs] + \text{sign_ext}(\text{Imm16})]$$

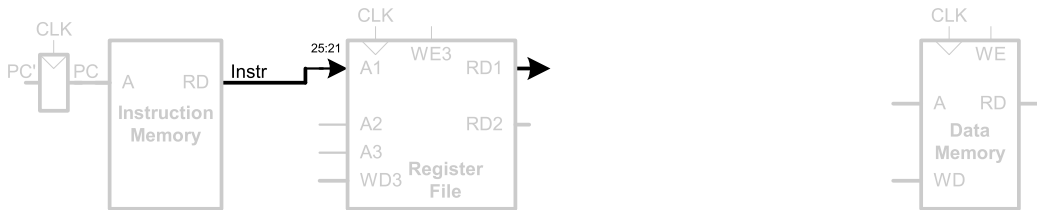
- **STEP 1: Fetch instruction**



Single-Cycle Datapath: lw register read

$$R[rt] \leftarrow DMEM[R[rs] + \text{sign_ext}(Imm16)]$$

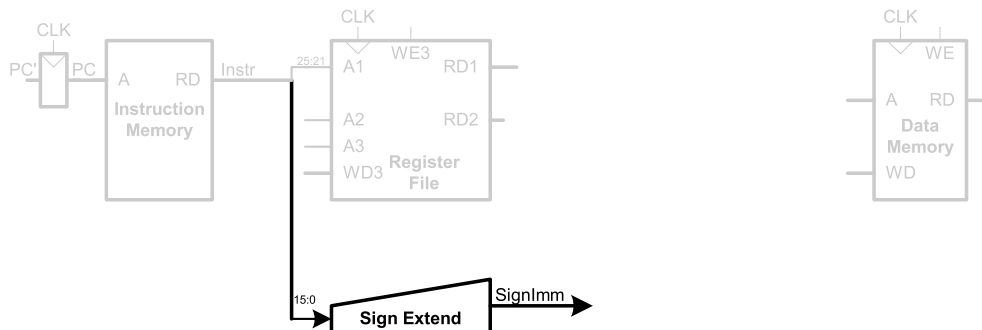
- STEP 2: Read source operands from register file



Single-Cycle Datapath: lw immediate

$$R[rt] \leftarrow DMEM[R[rs] + \text{sign_ext}(Imm16)]$$

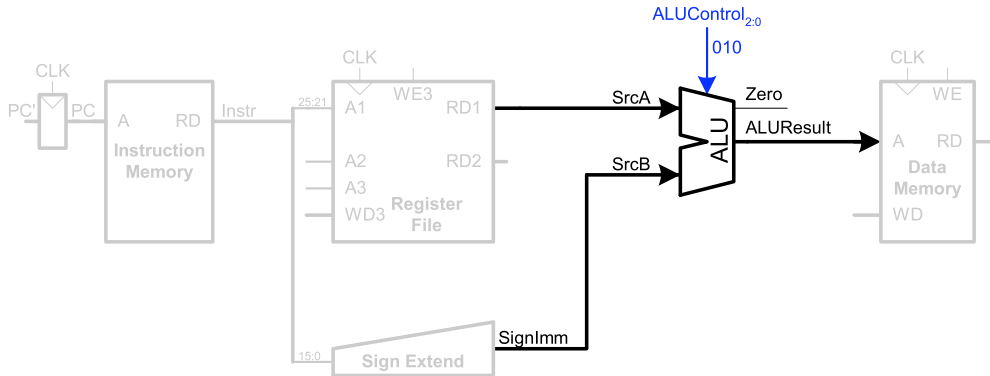
- STEP 3: Sign-extend the immediate



Single-Cycle Datapath: 1w address

$$R[rt] \leftarrow DMEM[R[rs] + \text{sign_ext}(Imm16)]$$

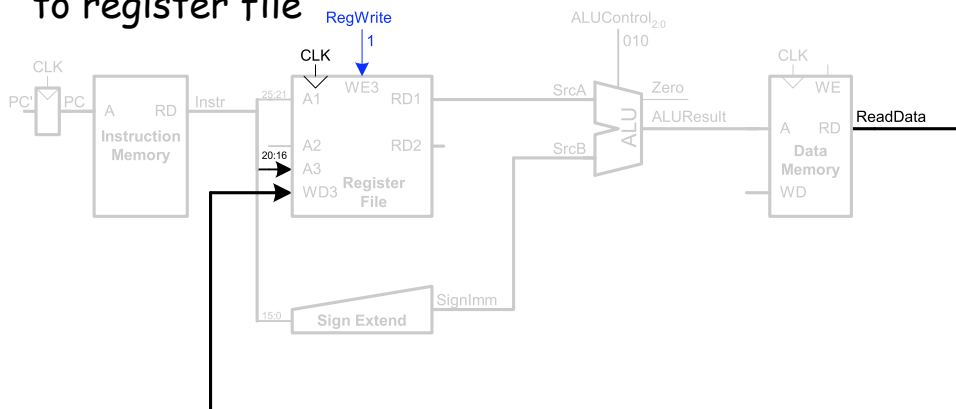
- **STEP 4:** Compute the memory address



Single-Cycle Datapath: 1w memory read

$$R[rt] \leftarrow DMEM[R[rs] + \text{sign_ext}(Imm16)]$$

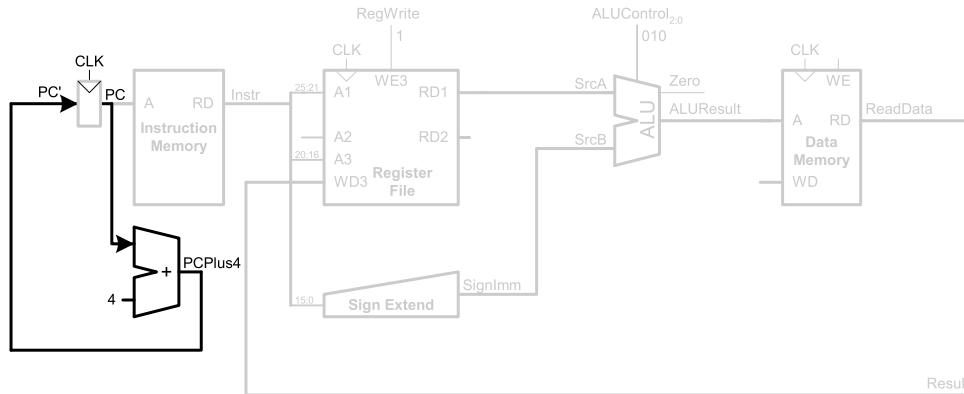
- **STEP 5:** Read data from memory and write it back to register file



Single-Cycle Datapath: lw PC increment

- STEP 6: Determine the address of the next instruction

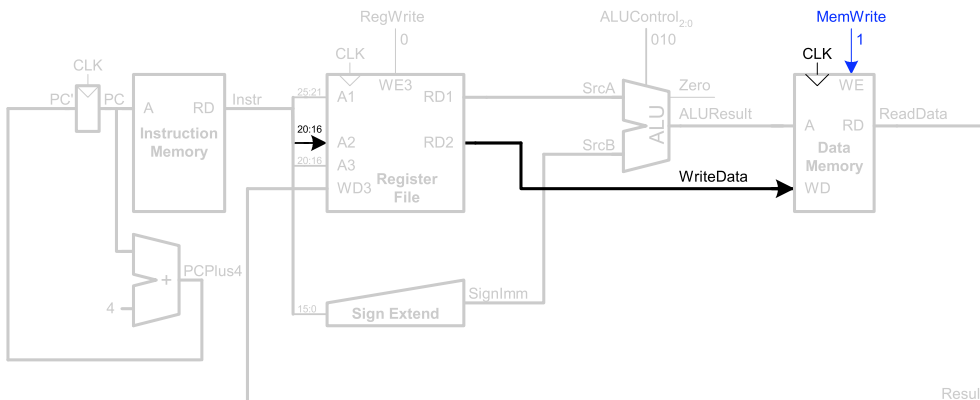
$$PC \leftarrow PC + 4$$



Single-Cycle Datapath: sw

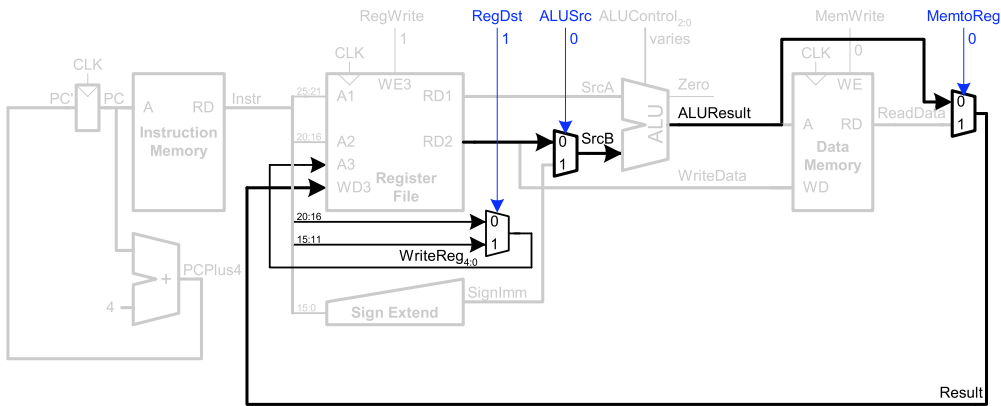
$$DMEM[R[rs] + \text{sign_ext}(Imm16)] \leftarrow R[rt]$$

- Write data in `rt` to memory



Single-Cycle Datapath: R-type instructions

- Read from rs and rt $R[rd] \leftarrow R[rs] \text{ op } R[rt]$
- Write $ALUResult$ to register file
- Write to rd (instead of rt)

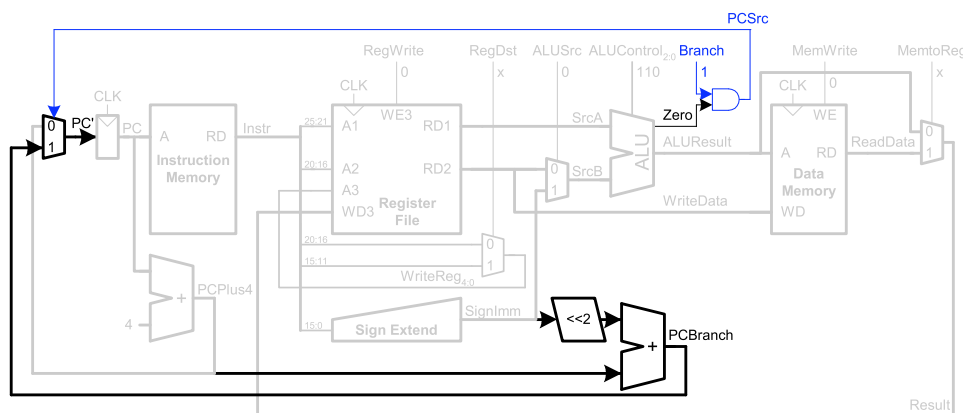


Single-Cycle Datapath: beq

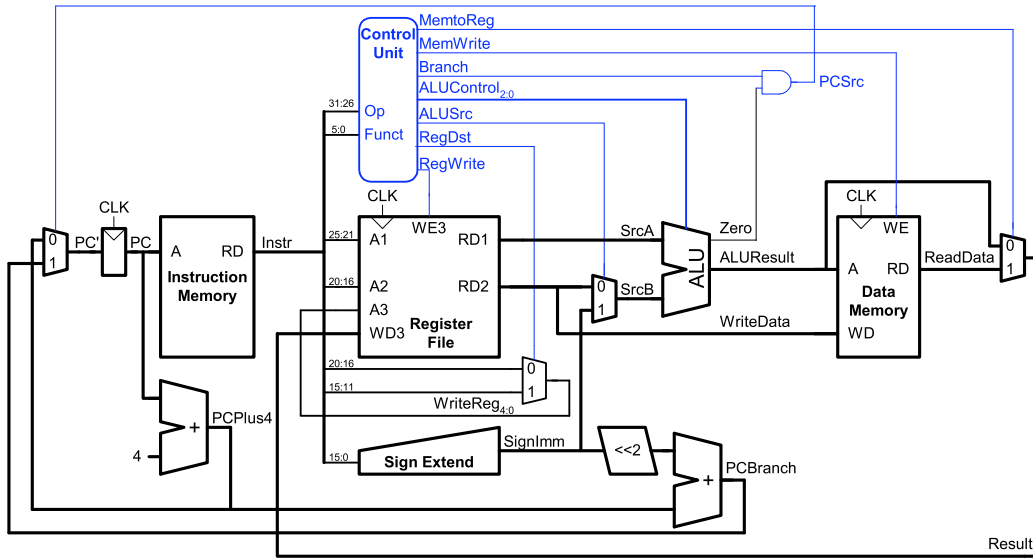
$\text{if } (R[rs] == R[rt]) \text{ then } PC \leftarrow PC + 4 + \{\text{sign_ext}(\text{Imm16}), 00\}$

- Determine whether values in rs and rt are equal
- Calculate branch target address:

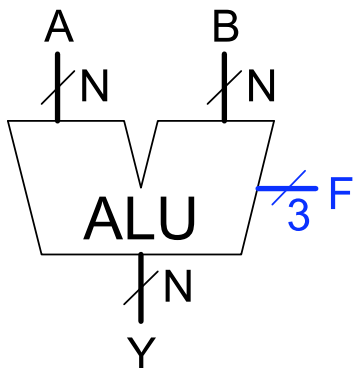
$$\text{BTA} = (\text{sign-extended immediate} \ll 2) + (\text{PC}+4)$$



Complete Single-Cycle Processor

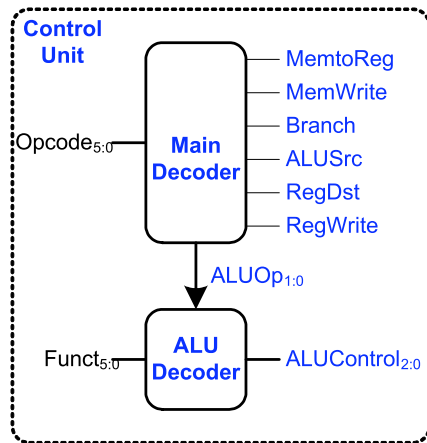


Review: ALU



| $F_{2:0}$ | Function |
|-----------|--------------|
| 000 | A & B |
| 001 | A B |
| 010 | A + B |
| 011 | not used |
| 100 | A & \sim B |
| 101 | A \sim B |
| 110 | A - B |
| 111 | SLT |

Control Unit



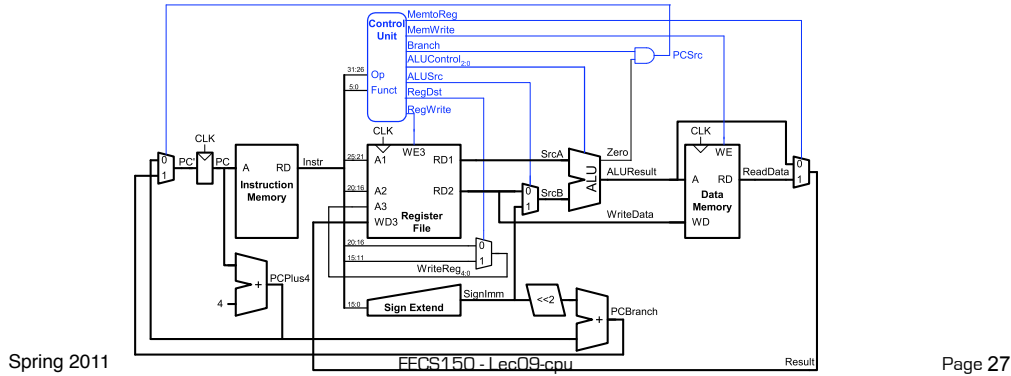
Control Unit: ALU Decoder

| ALUOp_{1:0} | Meaning |
|----------------------------|----------------|
| 00 | Add |
| 01 | Subtract |
| 10 | Look at Funct |
| 11 | Not Used |

| ALUOp_{1:0} | Funct | ALUControl_{2:0} |
|----------------------------|--------------|---------------------------------|
| 00 | XXXXXX | 010 (Add) |
| X1 | XXXXXX | 110 (Subtract) |
| 1X | 100000 (add) | 010 (Add) |
| 1X | 100010 (sub) | 110 (Subtract) |
| 1X | 100100 (and) | 000 (And) |
| 1X | 100101 (or) | 001 (Or) |
| 1X | 101010 (slt) | 111 (SLT) |

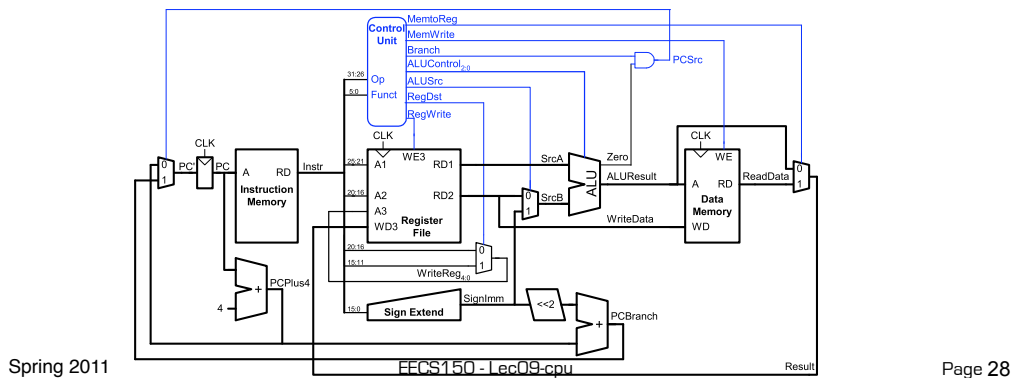
Control Unit: Main Decoder

| Instruction | Op _{5:0} | RegWrite | RegDst | AluSrc | Branch | MemWrite | MemtoReg | ALUOp _{1:0} |
|-------------|-------------------|----------|--------|--------|--------|----------|----------|----------------------|
| R-type | 000000 | | | | | | | |
| lw | 100011 | | | | | | | |
| sw | 101011 | | | | | | | |
| beq | 000100 | | | | | | | |

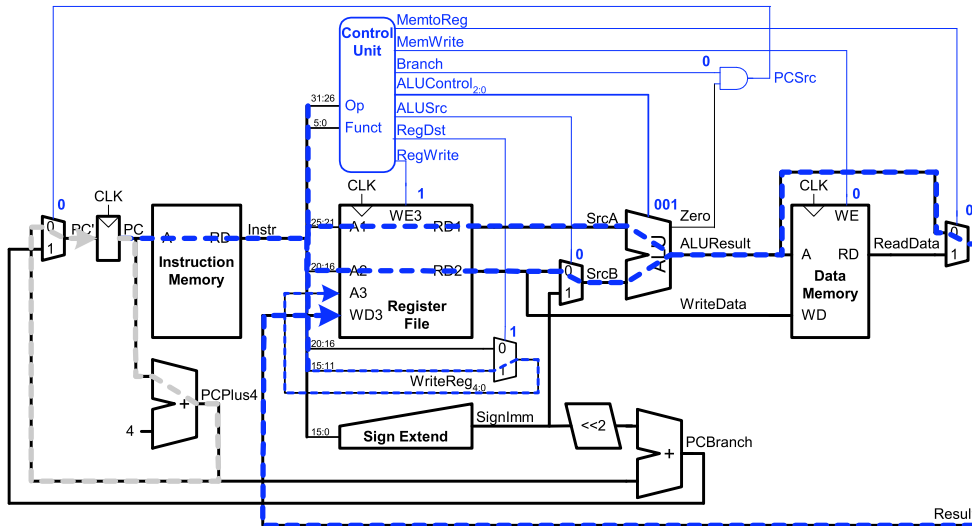


Control Unit: Main Decoder

| Instruction | Op _{5:0} | RegWrite | RegDst | AluSrc | Branch | MemWrite | MemtoReg | ALUOp _{1:0} |
|-------------|-------------------|----------|--------|--------|--------|----------|----------|----------------------|
| R-type | 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 10 |
| lw | 100011 | 1 | 0 | 1 | 0 | 0 | 0 | 00 |
| sw | 101011 | 0 | X | 1 | 0 | 1 | X | 00 |
| beq | 000100 | 0 | X | 0 | 1 | 0 | X | 01 |

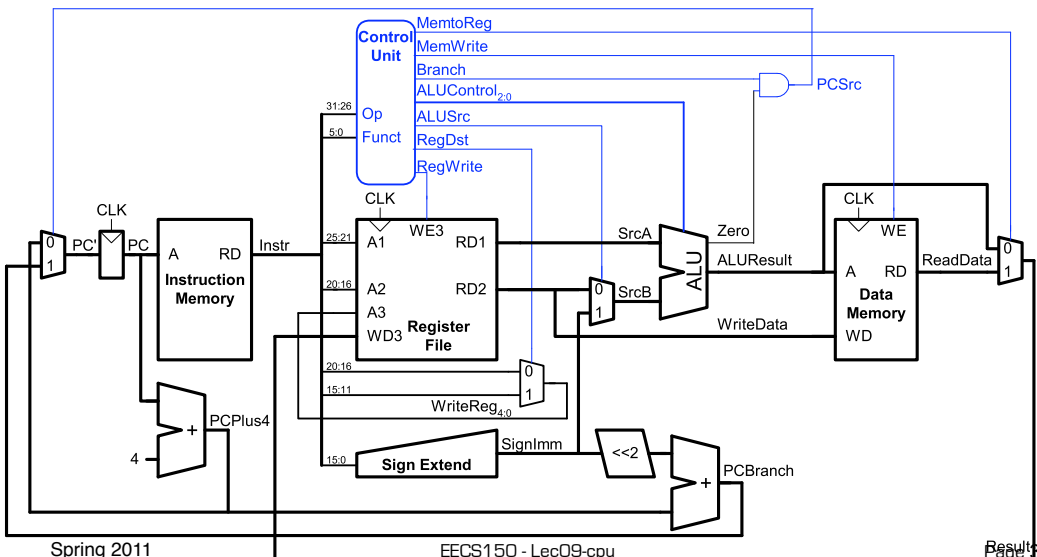


Single-Cycle Datapath Example: or



Extended Functionality: addi

- No change to datapath



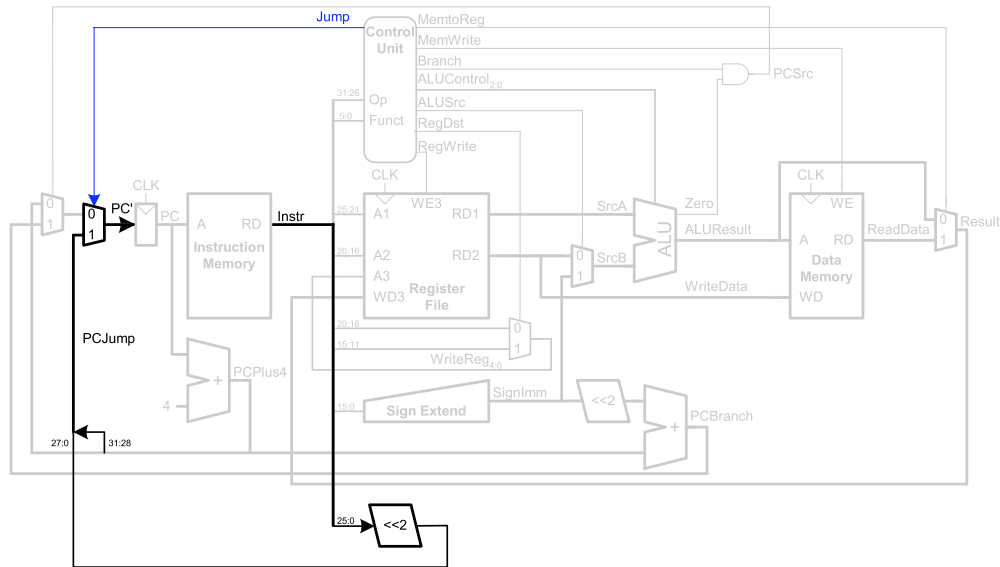
Control Unit: addi

| Instruction | Op _{5:0} | RegWrite | RegDst | AluSrc | Branch | MemWrite | MemtoReg | ALUOp _{1:0} |
|-------------|-------------------|----------|--------|--------|--------|----------|----------|----------------------|
| R-type | 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 10 |
| lw | 100011 | 1 | 0 | 1 | 0 | 0 | 1 | 00 |
| sw | 101011 | 0 | X | 1 | 0 | 1 | X | 00 |
| beq | 000100 | 0 | X | 0 | 1 | 0 | X | 01 |
| addi | 001000 | | | | | | | |

Control Unit: addi

| Instruction | Op _{5:0} | RegWrite | RegDst | AluSrc | Branch | MemWrite | MemtoReg | ALUOp _{1:0} |
|-------------|-------------------|----------|----------|----------|----------|----------|----------|----------------------|
| R-type | 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 10 |
| lw | 100011 | 1 | 0 | 1 | 0 | 0 | 1 | 00 |
| sw | 101011 | 0 | X | 1 | 0 | 1 | X | 00 |
| beq | 000100 | 0 | X | 0 | 1 | 0 | X | 01 |
| addi | 001000 | 1 | 0 | 1 | 0 | 0 | 0 | 00 |

Extended Functionality: j



Control Unit: Main Decoder

| Instruction | Op _{5,0} | RegWrite | RegDst | AluSrc | Branch | MemWrite | MemtoReg | ALUOp _{1,0} | Jump |
|-------------|-------------------|----------|--------|--------|--------|----------|----------|----------------------|------|
| R-type | 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 10 | 0 |
| lw | 100011 | 1 | 0 | 1 | 0 | 0 | 1 | 00 | 0 |
| sw | 101011 | 0 | X | 1 | 0 | 1 | X | 00 | 0 |
| beq | 000100 | 0 | X | 0 | 1 | 0 | X | 01 | 0 |
| j | 000100 | | | | | | | | |

Control Unit: Main Decoder

| Instruction | Op _{5:0} | RegWrite | RegDst | AluSrc | Branch | MemWrite | MemtoReg | ALUOp _{1:0} | Jump |
|-------------|-------------------|----------|----------|----------|----------|----------|----------|----------------------|----------|
| R-type | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 10 | 0 |
| lw | 100011 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| sw | 101011 | 0 | X | 1 | 0 | 1 | X | 0 | 0 |
| beq | 100 | 0 | X | 0 | 1 | 0 | X | 1 | 0 |
| j | 100 | 0 | X | X | X | 0 | X | XX | 1 |

Review: Processor Performance

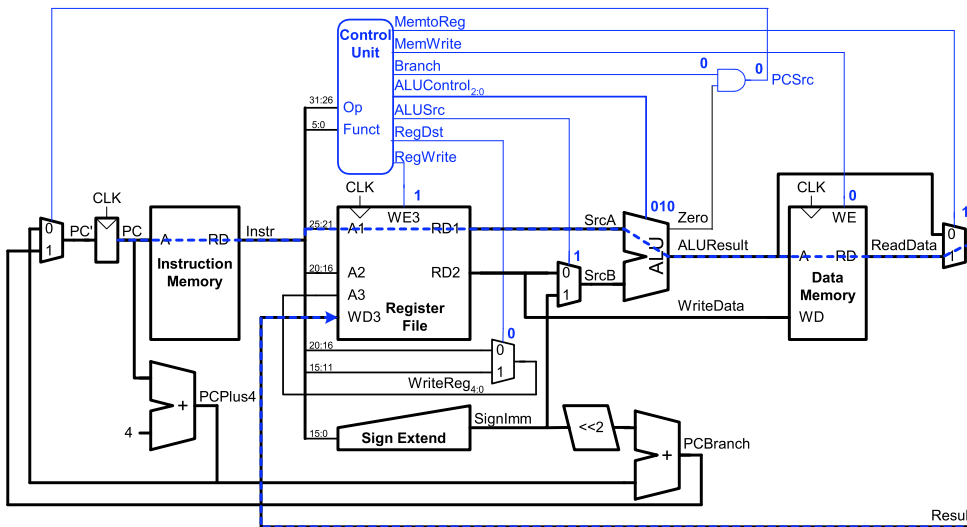
Program Execution Time

$$= (\# \text{ instructions})(\text{cycles/instruction})(\text{seconds/cycle})$$

$$= \# \text{ instructions} \times \text{CPI} \times T_C$$

Single-Cycle Performance

- T_c is limited by the critical path (**1w**)



Single-Cycle Performance

- Single-cycle critical path:

$$T_c = t_{pcq_PC} + t_{mem} + \max(t_{RFread}, t_{sext} + t_{mux}) + t_{ALU} + t_{mem} + t_{mux} + t_{RFsetup}$$
- In most implementations, limiting paths are:
 - memory, ALU, register file.
 - $T_c = t_{pcq_PC} + 2t_{mem} + t_{RFread} + t_{mux} + t_{ALU} + t_{RFsetup}$

Single-Cycle Performance Example

| Element | Parameter | Delay (ps) |
|---------------------|---------------|------------|
| Register clock-to-Q | t_{pcq_PC} | 30 |
| Register setup | t_{setup} | 20 |
| Multiplexer | t_{mux} | 25 |
| ALU | t_{ALU} | 200 |
| Memory read | t_{mem} | 250 |
| Register file read | t_{RFread} | 150 |
| Register file setup | $t_{RFsetup}$ | 20 |

$$T_c =$$

Single-Cycle Performance Example

| Element | Parameter | Delay (ps) |
|---------------------|---------------|------------|
| Register clock-to-Q | t_{pcq_PC} | 30 |
| Register setup | t_{setup} | 20 |
| Multiplexer | t_{mux} | 25 |
| ALU | t_{ALU} | 200 |
| Memory read | t_{mem} | 250 |
| Register file read | t_{RFread} | 150 |
| Register file setup | $t_{RFsetup}$ | 20 |

$$\begin{aligned} T_c &= t_{pcq_PC} + 2t_{mem} + t_{RFread} + t_{mux} + t_{ALU} + t_{RFsetup} \\ &= [30 + 2(250) + 150 + 25 + 200 + 20] \text{ ps} \\ &= 925 \text{ ps} \end{aligned}$$

Single-Cycle Performance Example

- For a program with 100 billion instructions executing on a single-cycle MIPS processor,

Execution Time =

Single-Cycle Performance Example

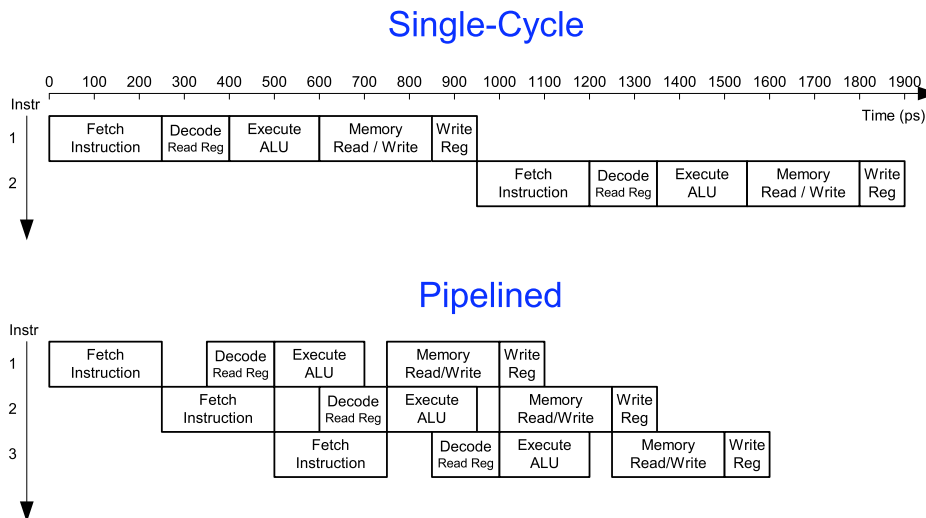
- For a program with 100 billion instructions executing on a single-cycle MIPS processor,

$$\begin{aligned}\text{Execution Time} &= \# \text{ instructions} \times \text{CPI} \times T_C \\ &= (100 \times 10^9)(1)(925 \times 10^{-12} \text{ s}) \\ &= 92.5 \text{ seconds}\end{aligned}$$

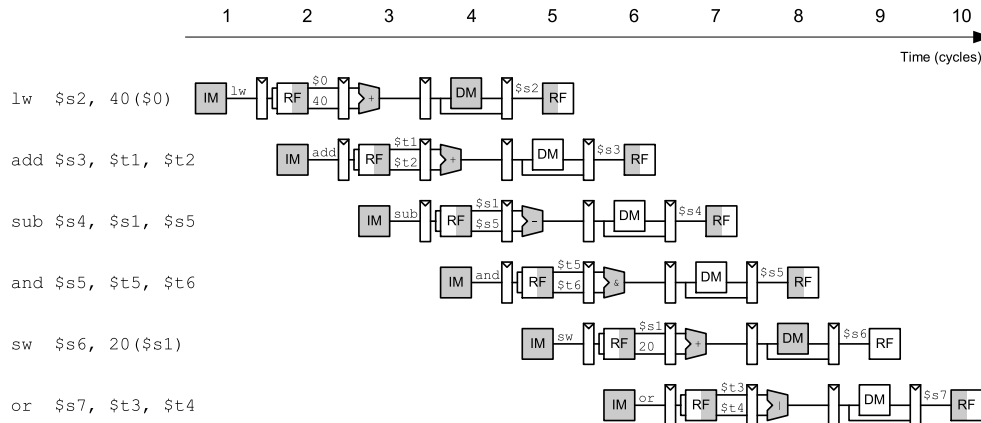
Pipelined MIPS Processor

- Temporal parallelism
- Divide single-cycle processor into 5 stages:
 - Fetch
 - Decode
 - Execute
 - Memory
 - Writeback
- Add pipeline registers between stages

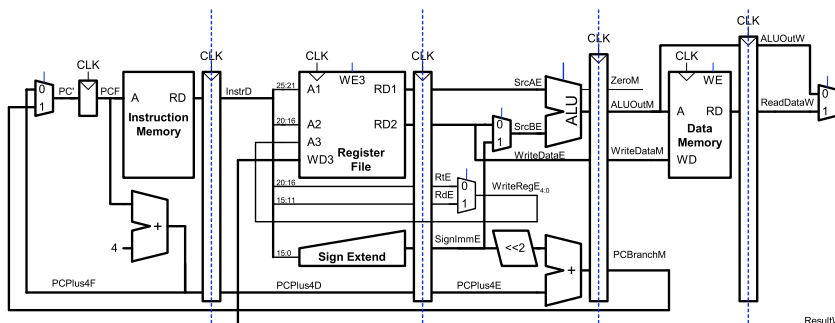
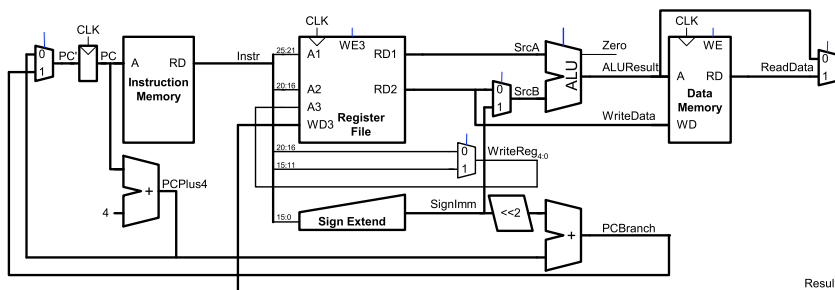
Single-Cycle vs. Pipelined Performance



Pipelining Abstraction

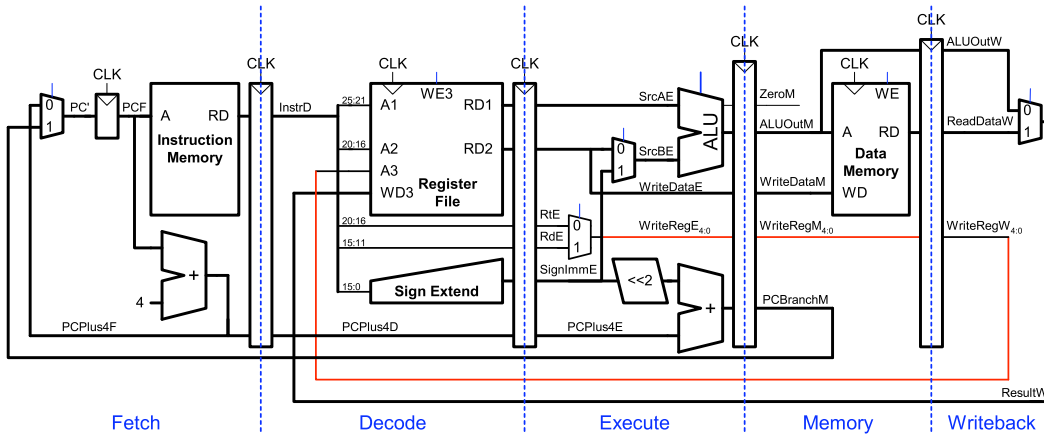


Single-Cycle and Pipelined Datapath

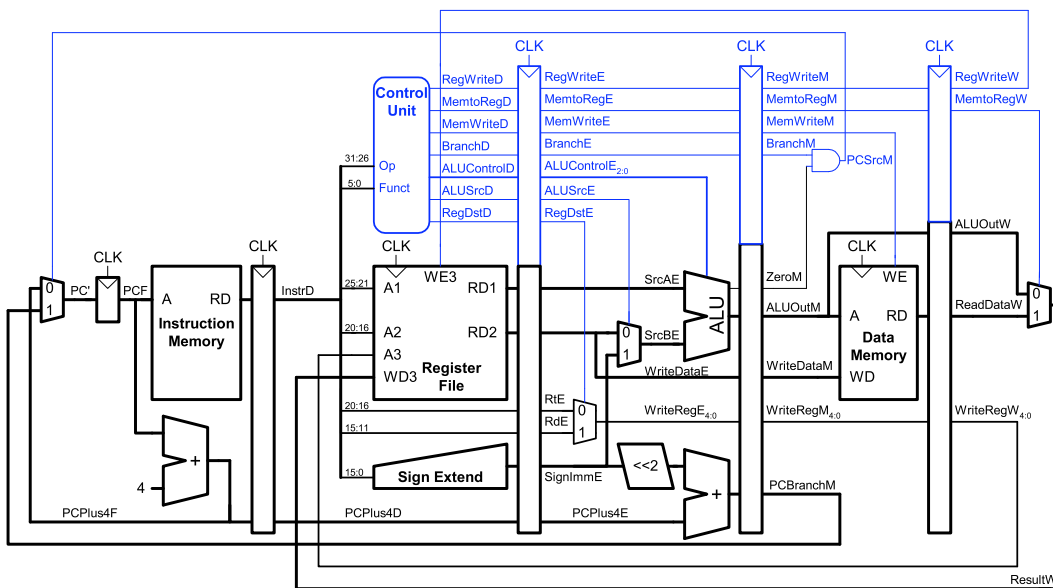


Corrected Pipelined Datapath

- WriteReg must arrive at the same time as Result



Pipelined Control



Same control unit as single-cycle processor

Control delayed to proper pipeline stage

Pipeline Hazards

- Occurs when an instruction depends on results from previous instruction that hasn't completed.
- Types of hazards:
 - **Data hazard:** register value not written back to register file yet
 - **Control hazard:** next instruction not decided yet (caused by branches)