Single-cycle datapath

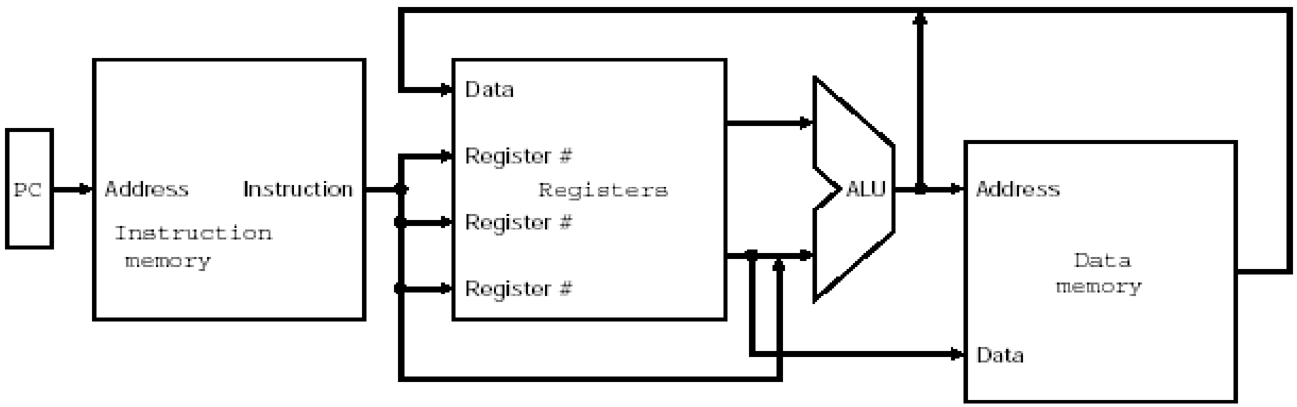


Fig. 5.1

Single-cycle implementation

- Assume each instruction is executed in 1 clock cycle
- Each component (memory, ALU, etc.) can be used only once
  - Reason for assuming separate instruction and data memories
- Advantage: simpler to design
- Disadvantage: speed of machine is determined by time for longest path
  - Memory access is much slower than register access,
    - but most instructions use only registers
- Better: each instruction type can take different number of clock cycles

Common elements for all instructions

- Instruction fetch, PC update
- Access 1 or 2 registers

Instruction fetch

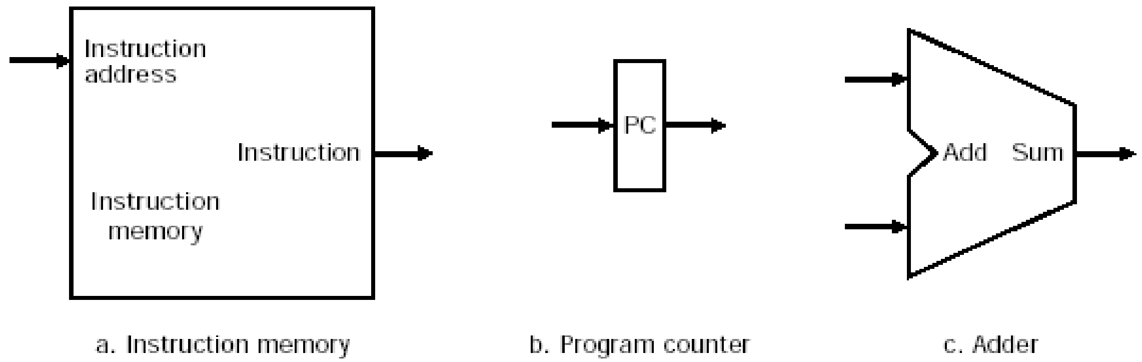


Fig. 5.4

Instruction fetch: used by all instructions

Memory

- **Input: instruction address**
- **Output: instruction**
- **how to build: defer until later**

Program counter

- **Register containing address of current instruction ("hidden")**

Adder: ALU with only one operation

- **Combinational circuit**
- **Input: 2 operands**
- **Output: sum**

Instruction fetch

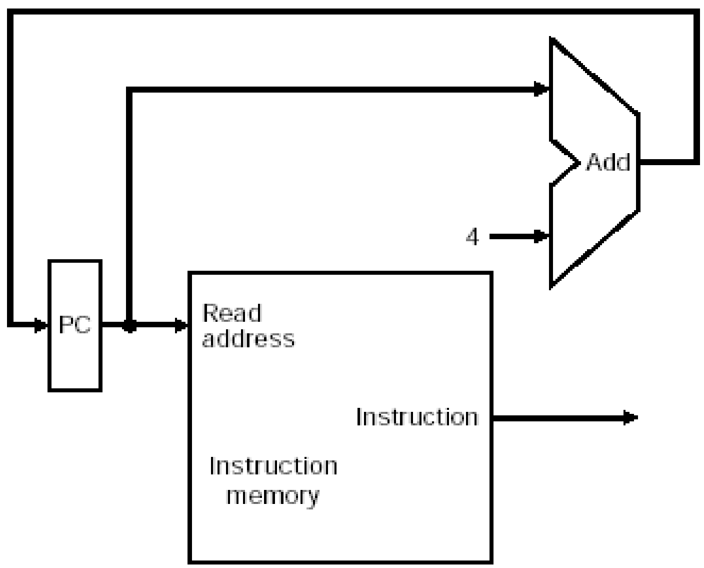


Fig. 5.5

Instruction fetch

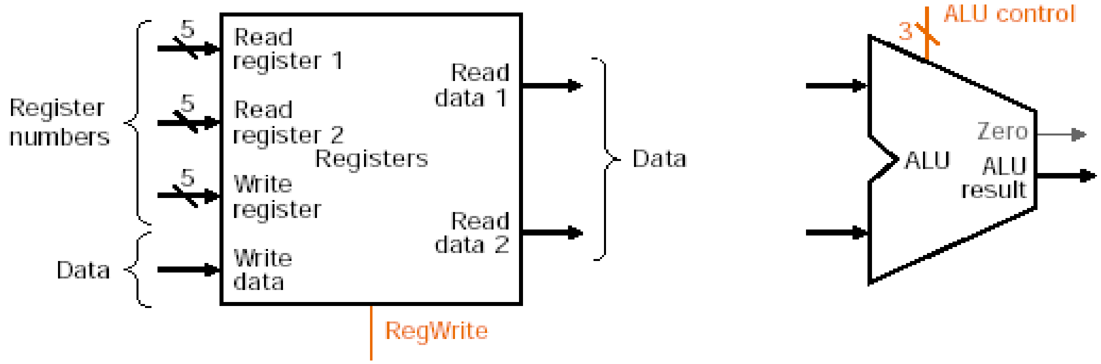
- **PC gives address to instruction memory**
- **Memory outputs instruction contents**

Increment PC

- **PC value is first operand input to adder**
- **Constant 4 is second operand input to adder**
- **PC + 4 is stored back in PC**

Repeat once each clock cycle

Register access: R-type



Components

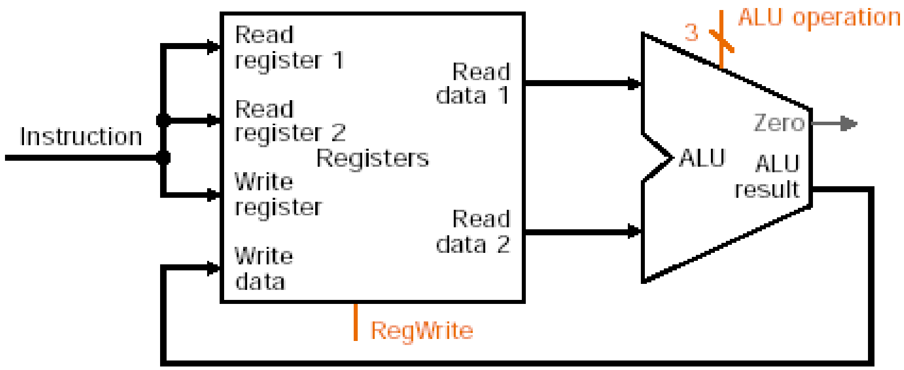
Register file: 32 registers **Fig. 5.6**

- **Inputs**
  - **2 read register numbers (5 bits each)**
  - **1 write register number (5 bits)**
  - **write data (32 bits)**
- **Outputs: 2 read data values (32 bits each)**
- **Control: RegWrite determines whether to write data to target register (1 bit)**

ALU: performs arithmetic/logical operations

- **Inputs: 2 data values (32 bits each)**
- **Outputs**
  - **Result of operation (32 bits)**
  - **Zero: result is equal to 0 (1 bit)**
- **Control: ALU control selects operation (3 bits)**

Register access: R-type



Components
Register file: 32 registers
ALU: performs arithmetic/logical operations

Fig. 5.7

Register read

- **Instruction gives addresses of 2 read registers and 1 write register to register file**
- **2 data read values are given by register file to inputs of ALU**

R-type operation

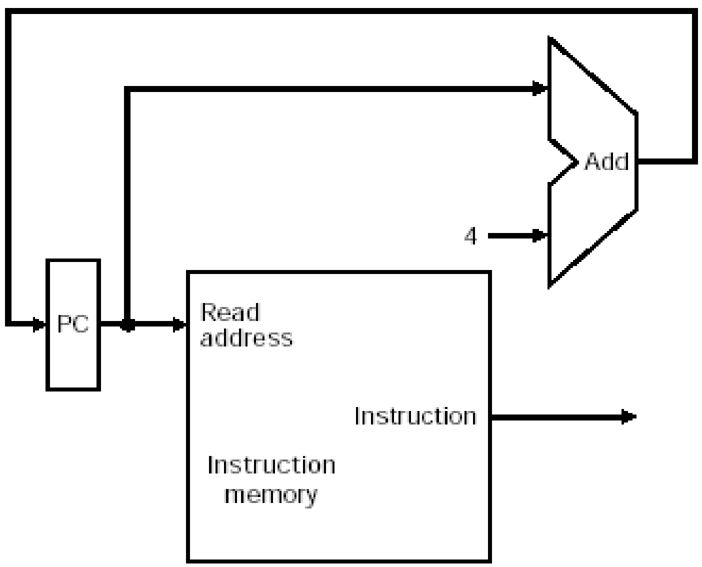
- **3 ALU operation control bits are used to determine what operation is required**
- **ALU result is returned to write data input of register file**
- **ALU also has an output called Zero (indicates whether result of operation equals 0)**
- **Note that RegWrite control input is used to determine if result is written to the write register**

Single-cycle datapath

The story so far:

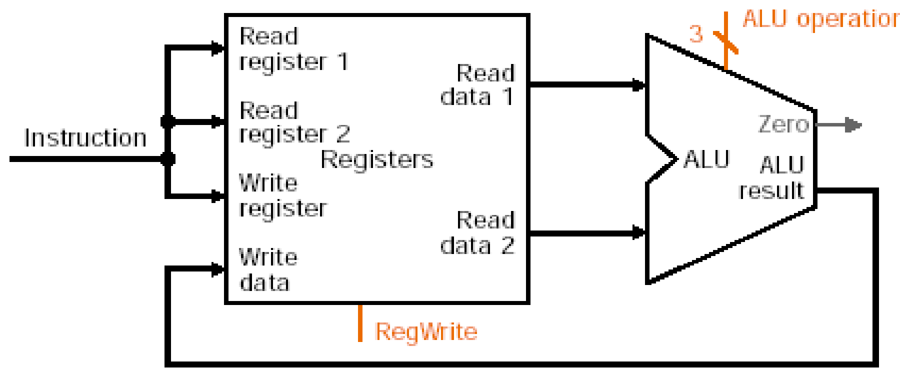
- **Implementing R-type, memory access, and branch/jump instructions**
- **Single-cycle datapath: each instruction takes 1 clock cycle**

Common elements:



Instruction fetch and PC update

Register access



R-type operation: ALU

Load and store

Load and store instructions

```
lw  $rt, offset($rs)
sw  $rt, offset($rs)
```

Requirements

- **Add 16-bit offset value to contents of base register $rs**
  - **extend 16-bit value to 32 bits for addition**
- **For load instruction, write value from memory into register $rt**
- **For store instruction, read data value from register $rt**

Load and store

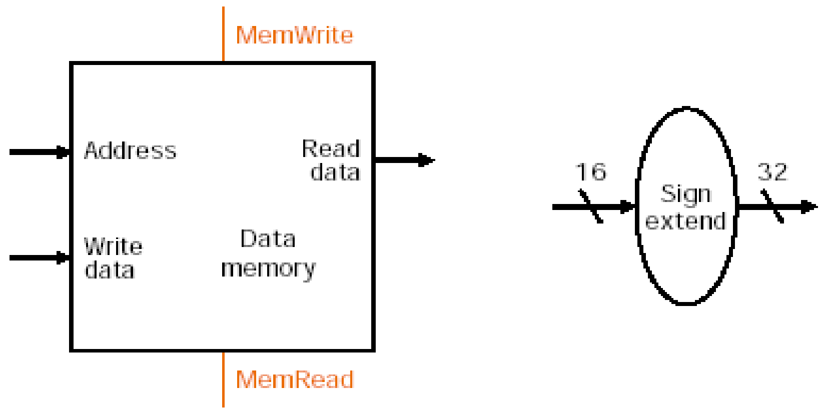


Fig. 5.8

Components

- **Data memory**
  - **Inputs**
    - **Address (32 bits)**
    - **Write data (32 bits)**
  - **Output: Read data (32 bits)**
  - **Controls**
    - **MemRead**
    - **MemWrite**
- **Sign extend**
  - **Input: 16 bit data**
  - **Output: 32 bit data with sign bit repeated 16 times**

Load and store

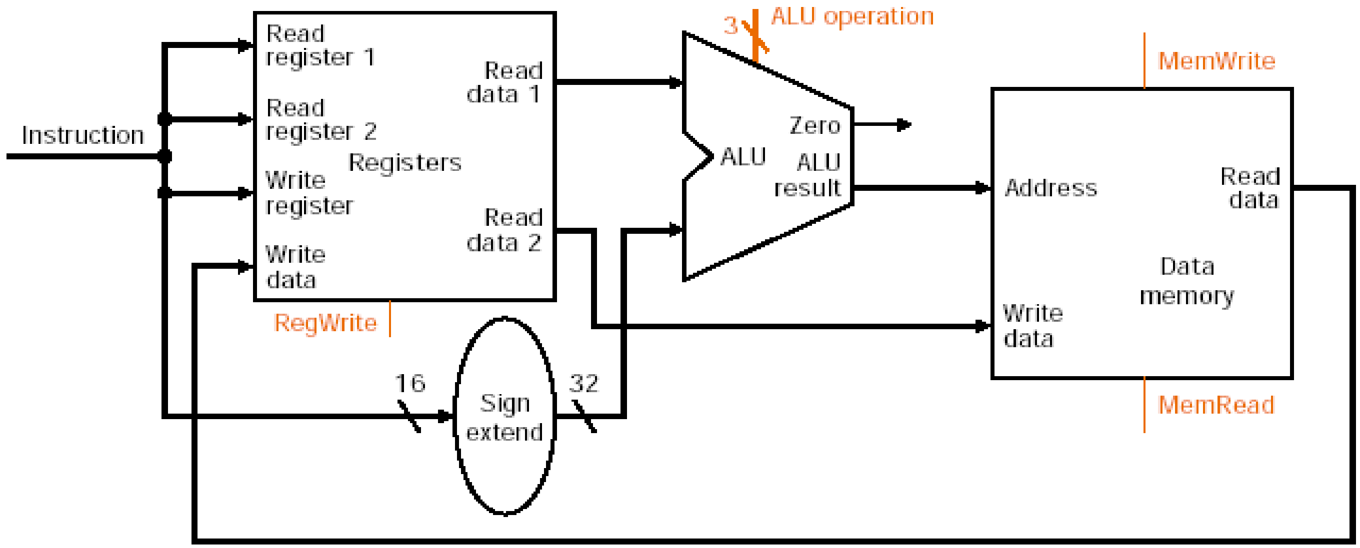


Fig. 5.9

Instruction provides read and write register numbers, 16-bit offset

Register file provides

- Read data 1: base register value to ALU
- Read data 2: data value to be stored in data memory

Sign extend provides 32-bit offset value to ALU for addition

ALU generates sum of base and offset as address input to data memory

MemRead, MemWrite controls determine whether to read or write data memory

- For load (read), data memory provides data for register write
- For store (write), data memory writes data to memory location given by address

Branch

Branch on equal

`beq  $rs, $rt, offset`

Requirements

- Compare contents of 2 registers
- Shift 16-bit offset left by 2 bits to get word address
- Add shifted offset value to value of PC + 4 to get branch target address
- Update PC with branch target if operands are equal (branch is taken)

Two operations: compare and add

- Also modify instruction fetch datapath to allow PC to be updated with new value

Jump requires different address calculation

- Replace lower 28 bits of PC with 26 bits from instruction, shifted left 2 bits
- To be added later

Branch

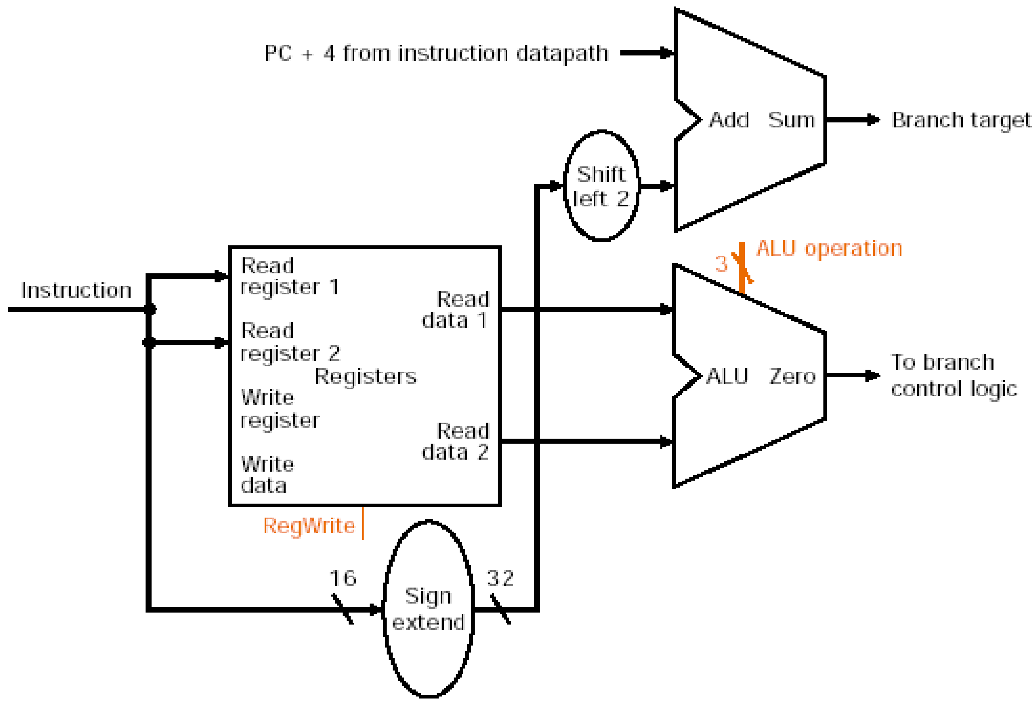


Fig. 5.10

Instruction provides read register numbers to register file, offset to sign extend

Registers give operand input to ALU for comparison (which ALU operation?)

ALU generates Zero output (what value?) to branch control

Sign extend provides 32-bit value to shifter, which shifts left by 2 bits

Adder computes branch target for possible PC update using:

- **Offset from shifter**
- **PC + 4 from instruction datapath**