

EEC 483 Computer Organization (Spring 2006)

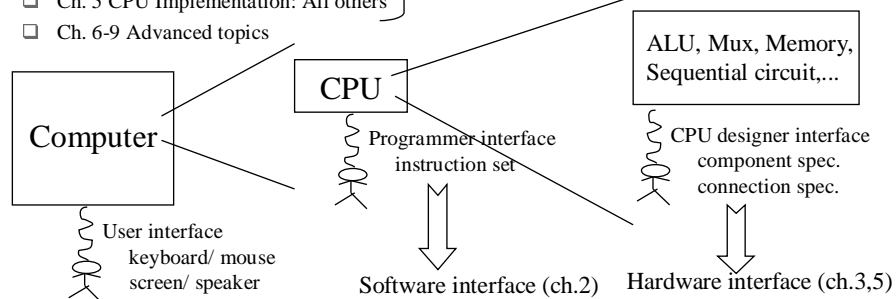
Chapter 5. The Processor: Datapath and Control

Chansu Yu

Cleveland State University

Table of Contents

- Ch.1, 4 Introduction & Performance
 - Ch. 2 Instruction: Machine Language
 - Ch. 3 CPU Implementation: Arithmetic
 - Ch. 5 CPU Implementation: All others
 - Ch. 6-9 Advanced topics
- Key parts of this course



Cleveland State
University

c.yu91@csuohio.edu

Table of Contents

- Ch.1, 4 Introduction & Performance
- Ch. 2 Instruction: Machine Language
 - 2.1 Introduction
 - 2.2 Operations (arithmetic, memory operations)
 - 2.3 Operands
 - 2.4 Representing instructions
 - 2.5 Logical operations
 - 2.6 Control flow operations
 - 2.7 Supporting procedures
 - 2.8 Beyond numbers
 - 2.9 MIPS addressing
 - 2.10 Starting a program
- Ch. 3 CPU Implementation: Arithmetic
 - 3.1 Introduction
 - 3.2 Signed and unsigned numbers
 - 3.3 Addition and subtraction
 - 3.4 Multiplication
 - 3.5 Division
 - 3.6 Floating point
 - Appendix B Constructing an Arithmetic Logic Unit (ALU)
- Ch. 5 CPU Implementation: All others
 - 5.1 Introduction
 - 5.2 Logic Design Conventions
 - 5.3 Building a Datapath
 - 5.4 A Simple Implementation Scheme
 - 5.5 A Multicycle Implementation
 - 5.6 Exceptions
 - 5.7-5.13 Etc.
- Ch. 6-9 Advanced topics

Key parts
of this course

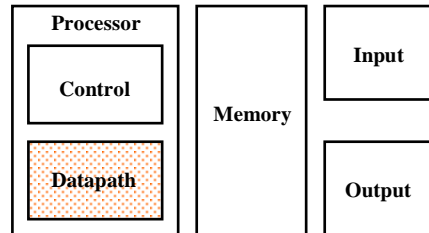
We have building blocks:
ALU, Register file, Memory.
Now, how to connect and
control them?

Implementation of the MIPS

- We're ready to look at an implementation of the MIPS
- Simplified to contain only:
 - memory-reference instructions: `lw, sw`
 - arithmetic-logical instructions: `add, sub, and, or, slt`
 - control flow instructions: `beq, j`
- Generic Implementation:
 - use the program counter (PC) to supply instruction address
 - get the instruction from memory
 - read registers
 - use the instruction to decide exactly what to do

The Big Picture

❑ The Five Classic Components of a Computer

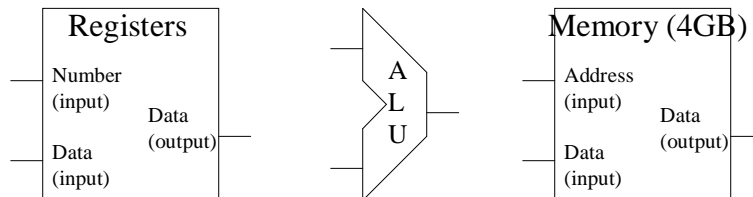


❑ Today's Topic: Designing the Datapath

Datapath of a Processor

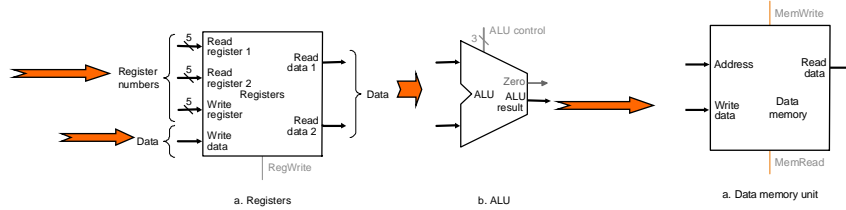
❑ Datapath

- All necessary data connection among the building blocks



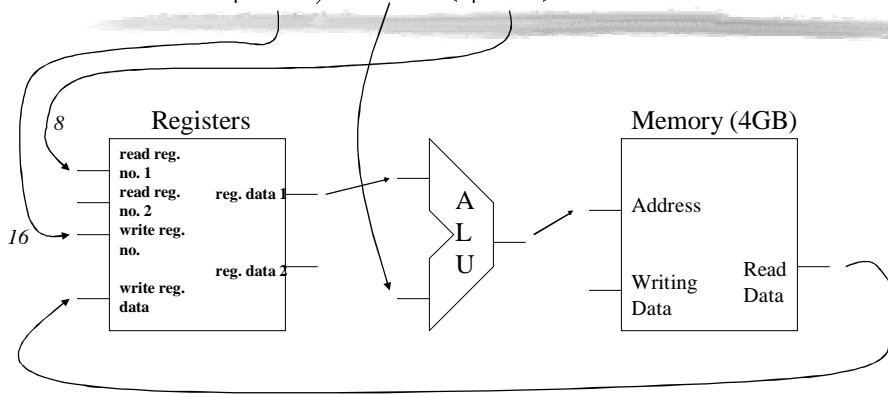
- How many bits for each connection ?

Register File, ALU and Memory

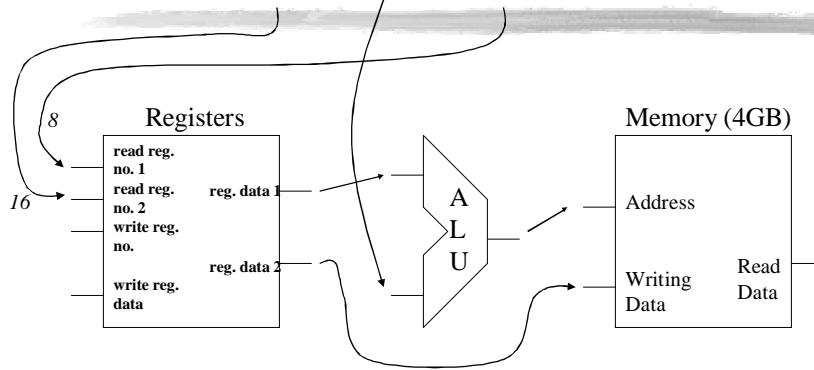


- ❑ Why register block has three inputs for register number and two outputs for register data ?
- ❑ ALU output can be fed into
 - Register block as a data input : Why ? And why not register number input ?
 - Memory block as an address : Why ? And why not data input ?
- ❑ What if it is not load/store architecture
 - add \$s0, \$s1, 96(\$t0)

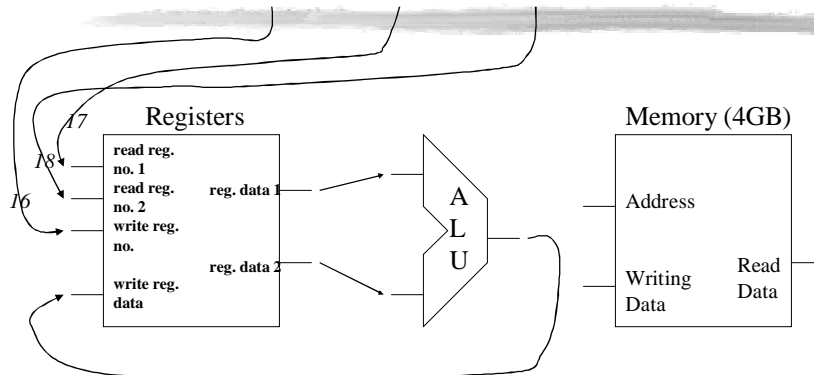
Ex: lw \$s0, 100(\$t0)



Ex: sw \$s0, 100(\$t0)

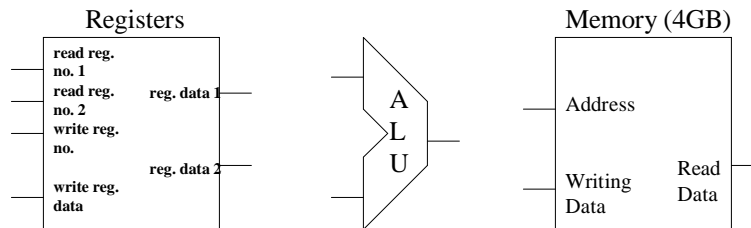


Ex: add \$s0, \$s1, \$s2



Memory is "not" used at all !!!
=>Load/store architecture: memory access is allowed only on load/store instruction

Ex: add \$s0, \$s1, 100(\$s2)

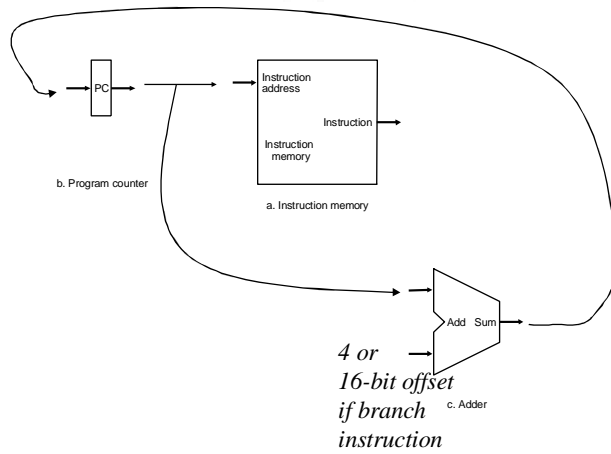


If the above instruction is a legitimate one, how to connect the blocks to implement it?

Input/Output for the Blocks

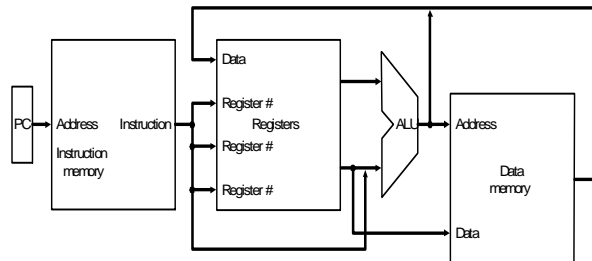
- ❑ Consider instructions: “previous four slides”
 - lw \$s0, 96(\$t0)
 - sw \$t3, 4(\$s2)
 - add \$s0, \$s1, \$s2
- ❑ MIPS addressing: “next slide”
 - $96(\$t0) = 96 + \$t0$
 - new \$pc = old \$pc + 4
 - new \$pc = old \$pc + immediate (PC-relative addressing)
- ❑ All instructions use the ALU after reading the registers
Why? memory-reference? arithmetic? control flow?

Instruction Memory, PC, Adder



Implementation Details

Abstract / Simplified View:



Two types of functional units:

- elements that operate on data values (combinational)
- elements that contain state (sequential)

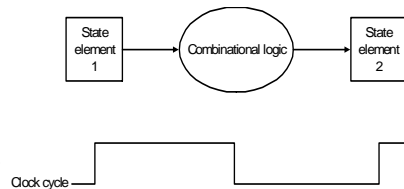
Requiring Clocked Operation

- ❑ Control sequential circuit
 - Combination circuits will provide result after some gate delays
 - Sequential circuits will provide result upon an input (maybe from combination circuits)
- ❑ add \$t0, \$zero, \$zero
add \$s0, \$s0, \$s2
 - Assume \$s0=10, \$s2=20
 - When \$s0, \$s2 are read, writing to \$s0 must be prohibited
 - because the original value \$s0 (10) must not be destroyed
 - because the result "30" is not ready yet (actually, the data line fed into the register file carries "0" that was the result of the previous calculation.)
- ❑ Who controls when to read and when to write safely?
 - Synchronous operation based on clock
 - Sequential circuits will provide result upon an input and "clock"

Our Implementation

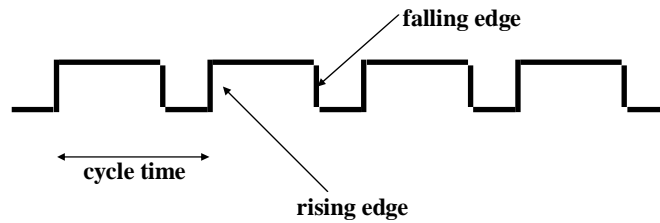
What a processor (program) essentially does is to change the contents of state element, where state element consists of memory and register file

- ❑ An edge triggered methodology
 - Every input to update state element must be stable at the active clock edge
 - Feedback cannot occur within the same clock cycle (no race)
 - The contents of state element change every clock cycle
- ❑ Typical execution (ex: add \$s0, \$s1, \$s2):
 - read contents of some state elements (\$s1, \$s2)
 - send values through some combinational logic (adder in ALU)
 - write results to one or more state elements (\$s0)



State Elements

- ❑ Unlocked vs. Clocked
- ❑ Clocks used in synchronous logic
 - when should an element that contains state be updated?



Latches and Flip-flops

- ❑ Output is equal to the stored value inside the element
(don't need to ask for permission to look at the value)
- ❑ Change of state (value) is based on the clock
- ❑ Latches: whenever the inputs change, and the clock is asserted
- ❑ Flip-flop: state changes only on a clock edge
(edge-triggered methodology)

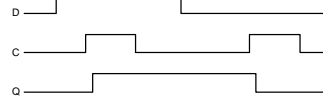
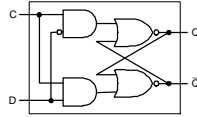
"logically true",
— could mean electrically low

A clocking methodology defines when signals can be read and written
— wouldn't want to read a signal at the same time it was being written

D-Latch & D Flip-flop

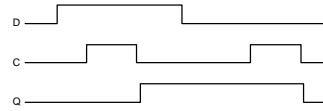
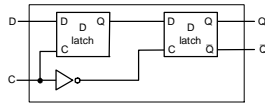
□ D-latch

- Two inputs:
 - the data value to be stored (D)
 - the clock signal (C) indicating when to read & store D
- Two outputs: the value of the internal state (Q) and its complement



□ D flip-flop

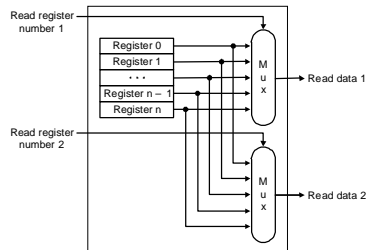
- Output changes only on the clock edge



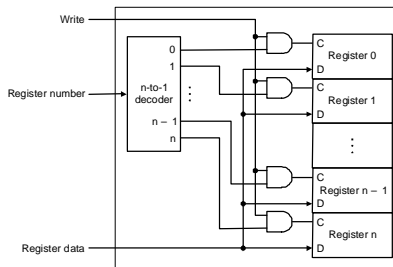
Register File

□ Built using D flip-flops

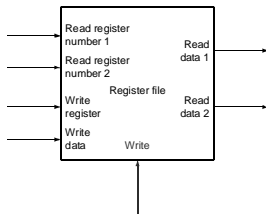
(read)



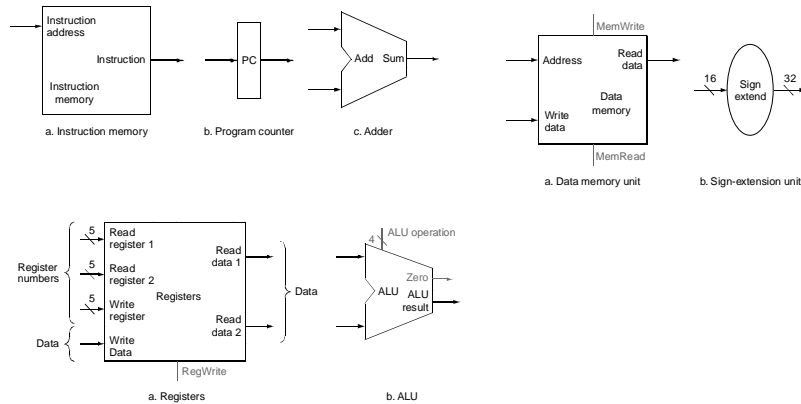
(write)



Register File



All Functional Units



Datapath with Multiplexers

