ECE 152
Introduction to Computer Architecture

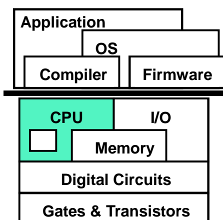
Processor Design: Datapath and Control
Copyright 2012 Daniel J. Sorin
Duke University

Slides are derived from work by
Amir Roth (Penn)
Spring 2012

Where We Are in This Course Right Now

- So far:
  - We know what a computer architecture is
  - We know what kinds of instructions it might execute
  - We know how to perform arithmetic and logic in an ALU
- Now:
  - We learn how to design a processor in which the ALU is just one component
  - Processor must be able to fetch instructions, decode them, and execute them
  - There are many ways to do this, even for a given ISA
- Next:
  - We learn how to use pipelining to get better performance out of this processor

This Unit: Processor Design



- Datapath components and timing
  - Registers and register files
  - Memories (RAMs)
- Mapping an ISA to a datapath
- Control
- Exceptions

Readings

- Patterson and Hennessy
  - Chapter 4: Sections 4.1-4.4
- Read this chapter carefully
  - It has many more examples than I can cover in class

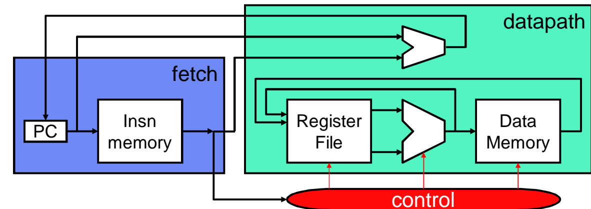
So You Have an ALU…

- Important reminder: a processor is just a big finite state machine (FSM) that interprets some ISA
- Start with one instruction

```
add $3,$2,$4
```

  - ALU performs just a small part of execution of instruction
  - You have to read and write registers
  - You have have to fetch the instruction to begin with
- What about loads and stores?
  - Need some sort of memory interface
- What about branches?
  - Need some hardware for that, too

Datapath and Control



- **Datapath**: registers, memories, ALUs (computation)
- **Control**: which registers read/write, which ALU operation
- **Fetch**: get instruction, translate into control
- Processor Cycle: **Fetch** → **Decode** → **Execute**

Building a Processor for an ISA

- Fetch is pretty straightforward
  - Just need a register (called the Program Counter or PC) to hold the next address to fetch from instruction memory
  - Provide address to instruction memory → instruction memory provides instruction at that address
- Let's start with the datapath
  1. Look at ISA
  2. Make sure datapath can implement every instruction

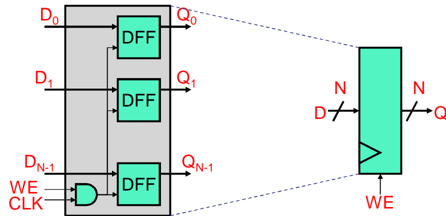
Datapath for MIPS ISA

- Consider only the following instructions

```
add $1,$2,$3
addi $1,2,$3
lw $1,4($3)
sw $1,4($3)
beq $1,$2,PC_relative_target
j Absolute_target
```

- Why only these?
  - Most other instructions are similar from datapath viewpoint
  - I leave the other ones for you to figure out

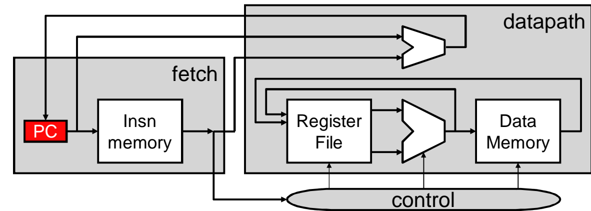
Review (ECE 52 & Project Part1): Register



- **Register**: DFF array with shared clock, write-enable (WE)
  - Notice: both a clock and a WE (DFF_{WE} = clock & $register_{WE}$)
  - Convention I: clock represented by wedge
  - Convention II: if no WE, DFF is written on every clock

© 2012 Daniel J. Sorin
from Roth

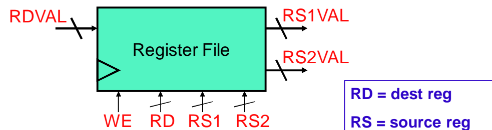
Uses of Registers



- A single register is good for some things
  - PC: program counter
  - Other things which aren't the ISA registers
    - ICQ: other examples from within the ALU, mult, div?

© 2012 Daniel J. Sorin
from Roth

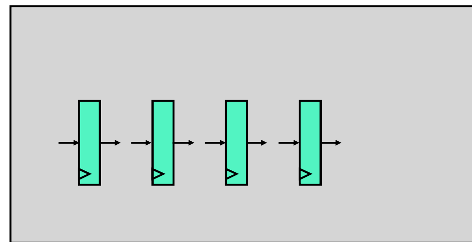
What About the ISA Registers?



- **Register file**: the ISA ("architectural", "visible") registers
  - Two read "ports" + one write "port"
    - Maximum number of reads/writes in single instruction (R-type)
- **Port**: wires for accessing an array of data
  - Data bus: width of data element (MIPS: 32 bits)
  - Address bus: width of $\log_2$ number of elements (MIPS: 5 bits)
  - Write enable: if it's a write port
  - M ports = M parallel and independent accesses

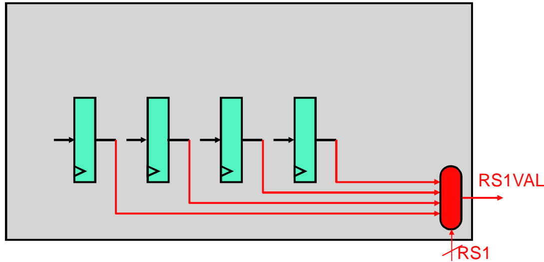
© 2012 Daniel J. Sorin
from Roth

A Register File With Four Registers



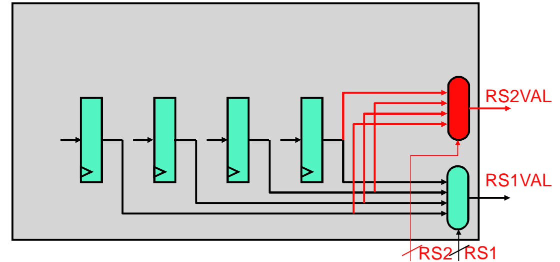
© 2012 Daniel J. Sorin
from Roth

Add a Read Port for RS1



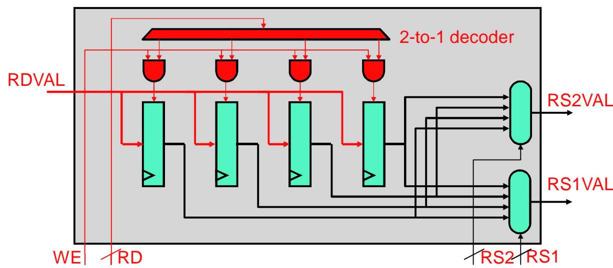
- Output of each register into 4to1 mux (RS1VAL)
  - RS1 is select input of RS1VAL mux

Add Another Read Port for RS2



- Output of each register into another 4to1 mux (RS2VAL)
  - RS2 is select input of RS2VAL mux

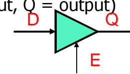
Add a Write Port for RD



- Input RDVAL into each register
  - Enable only one register's WE: (Decoded RD) & (WE)
- What if we needed two write ports?

Another Read Port Implementation

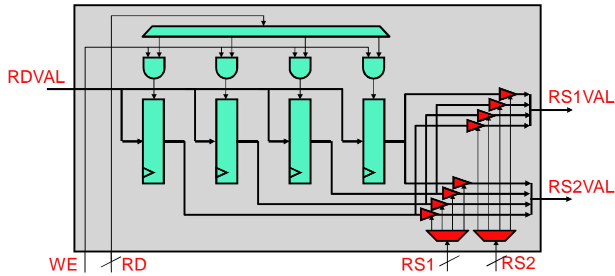
- A read port that uses muxes is fine for 4 registers
  - Not so good for 32 registers (32-to-1 mux is very slow)
- Alternative implementation uses **tri-state buffers**
  - Truth table (E = enable, D = input, Q = output)



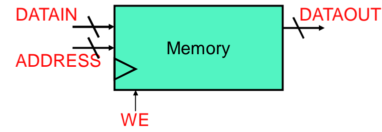
E D → Q
1 D → D
0 D → **Z**

  - **Z**: "high impedance" state, no current flowing
  - Mux: connect multiple tri-stated buses to one output bus
  - Key: only one input "driving" at any time, all others must be in "Z"
    - Else, all hell breaks loose (electrically)

Register File With Tri-State Read Ports



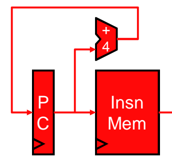
Another Useful Component: Memory



- **Memory**: where instructions and data reside
  - One read/write "port": one access per cycle, either read **or** write
    - One address bus
  - One input data bus for writes, one output data bus for reads

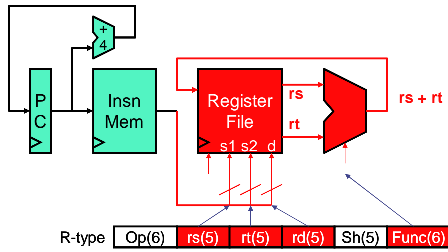
Let's Build A MIPS-like Datapath

Start With Fetch



- PC and instruction memory
- A +4 incrementer computes default next instruction PC
  - Why +4 (and not +1)? What will it be for 32-bit Duke 152/32?

First Instruction: add $rd, $rs, $rt



- Add register file and ALU