Introduction to the MIPS Architecture

Outline

What is a computer architecture?

One view: The machine language the CPU implements

Instruction set architecture (ISA)

- Built in data types (integers, floating point numbers)
- Fixed set of instructions
- Fixed set of on-processor variables (registers)
- Interface for reading/writing memory
- Mechanisms to do input/output

What is a computer architecture?

Another view: How the ISA is implemented

Microarchitecture
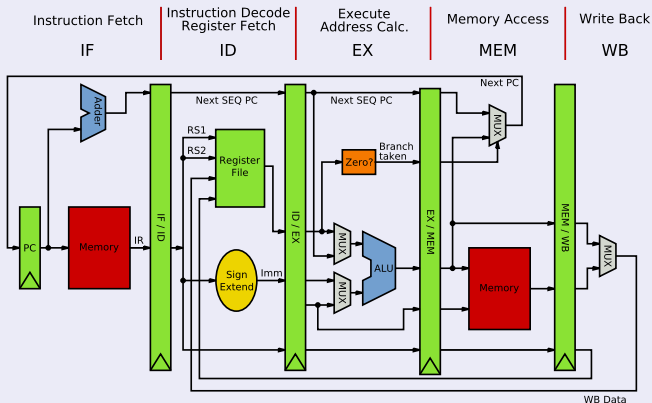
How a computer executes a program

Fetch-decode-execute cycle (FDX)

1. **fetch** the next instruction from memory
2. **decode** the instruction
3. **execute** the instruction

Decode determines:

- operation to execute
- arguments to use
- where the result will be stored

Execute:

- performs the operation
- determines next instruction to fetch (by default, next one)

Datapath and control unit

Datapath

Major hardware components of the FDX cycle
- path of instructions and data through the processor
- components connected by **buses**

Bus – parallel path for transmitting values
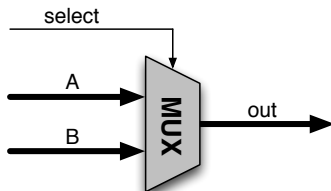- in MIPS, usually 32 bits wide

Datapath and control unit

Control unit

Controls the components of the datapath

- determines how data moves through the datapath
- receives **condition signals** from the components
- sends **control signals** to the components
- switches between buses with **multiplexers**

Multiplexer – component for choosing between buses

Outline

Components of the MIPS architecture

Major components of the datapath:

- program counter (PC)
- instruction register (IR)
- register file
- arithmetic and logic unit (ALU)
- memory

Control unit

http://www.cise.ufl.edu/~mssz/CompOrg/Figure4.3-MIPSarch2.gif

Memory: text segment vs. data segment

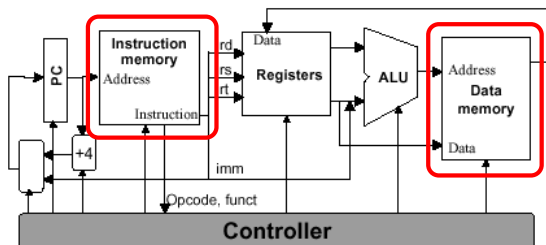In MIPS, programs are separated from data in memory

Text segment

- "instruction memory"
- part of memory that stores the program (machine code)
- **read only**

Data segment

- "data memory"
- part of memory that stores data manipulated by program
- **read/write**

Memory: text segment vs. data segment

Distinction may or may not be reflected in the hardware:

- von Neumann architecture – single, shared memory
- Harvard architecture – physically separate memories

Memory addressing in MIPS

For reading/writing the data segment

Base address plus displacement

Memory address computed as base+offset:

- base is obtained from a register
- offset is given directly as an integer

Load word (read word from memory into register):

lw $t1,8($t2) $\Rightarrow$ **$t1 := Memory[$t2+8]**

Store word (write word from register into memory):

sw $t1,4($t2) $\Rightarrow$ **Memory[$t2+4] := $t1**
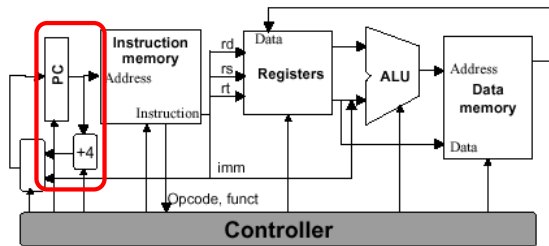
We'll talk about addressing in the text segment later

Program counter (PC)

Program: a sequence of machine instructions
in the text segment

```
0x8d0b0000
0x8d0c0004
0x016c5020
0xad0a0008
0x21080004
0x2129ffff
0x1d20fff9
```


Program counter

Register that stores the **address** of the next instruction to fetch

- also called the instruction pointer (IP)

Incrementing the PC

In MIPS, each instruction is exactly 32-bits long

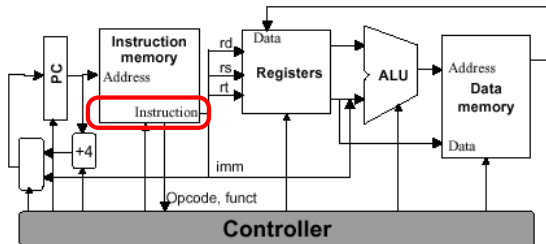What is the address of the next instruction?

PC+4

(Each address refers to one byte, and $32/8 = 4$)

Instruction register (IR)

> ## Instruction register
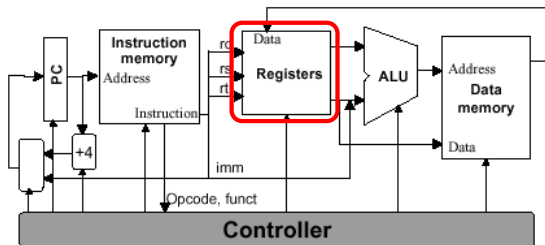> Register that holds the instruction currently being decoded

Note condition signals from the IR to the control unit!

Register file

Register: component that stores a 32-bit value

MIPS register file contains 32 registers

Register names and conventions

| Number	Name	Usage	Preserved?
`$0`	`$zero`	constant `0x00000000`	N/A
`$1`	`$at`	assembler temporary	✗
`$2`–`$3`	`$v0`–`$v1`	function return values	✗
`$4`–`$7`	`$a0`–`$a3`	function arguments	✗
`$8`–`$15`	`$t0`–`$t7`	temporaries	✗
`$16`–`$23`	`$s0`–`$s7`	saved temporaries	✓
`$24`–`$25`	`$t8`–`$t9`	more temporaries	✗
`$26`–`$27`	`$k0`–`$k1`	reserved for OS kernel	N/A
`$28`	`$gp`	global pointer	✓
`$29`	`$sp`	stack pointer	✓
`$30`	`$fp`	frame pointer	✓
`$31`	`$ra`	return address	✓

Arithmetic and logic unit (ALU)

Implements binary arithmetic and logic operations

Inputs:

- operands – 2 × 32-bit
- operation – control signal
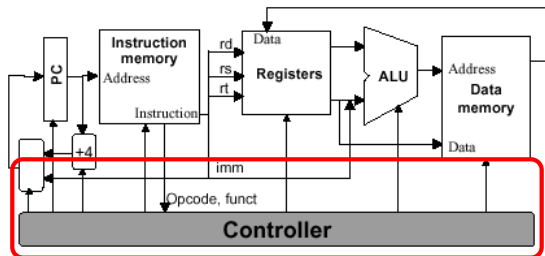
Outputs:

- result – 1 × 64-bit
  (usually just use 32 bits of this)
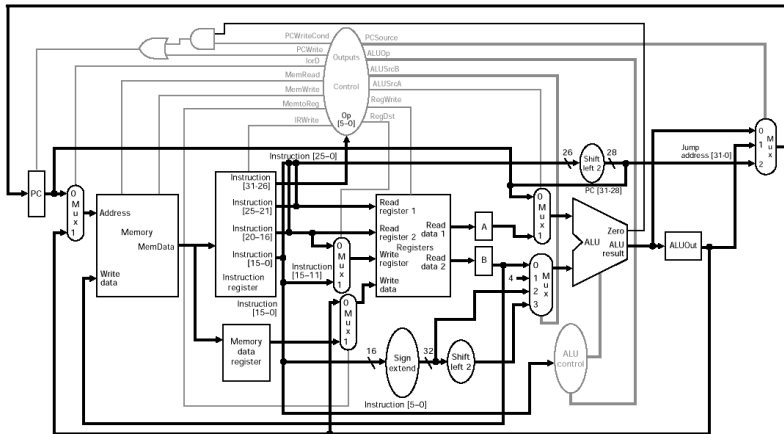- status – condition signals
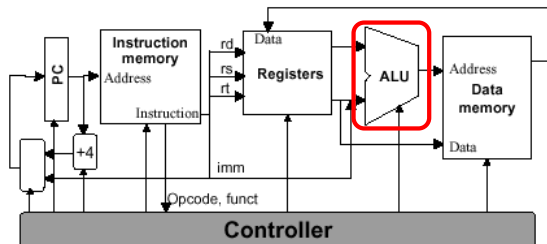
Control unit

Controls components of datapath to implement FDX cycle

- **Inputs**: condition signals
- **Outputs**: control signals

Implemented as a finite state machine

MIPS data path with control signals

http://fourier.eng.hmc.edu/e85/lectures/figures/MIPS_datapath_control.gif

Control unit

Condition signals

- from IR – decode operation, arguments, result location
- from ALU – overflow, divide-by-zero, . . .

Control signals

- to multiplexors – buses to select
- to each register – load new value
- to ALU – operation to perform
- to all – **clock signal**

Clock signal

Each component is implemented as an electrical circuit

- when inputs change, outputs change – not instantaneous!
- clock signal ensures we don't use outputs until ready

Clock generator produces the clock signal
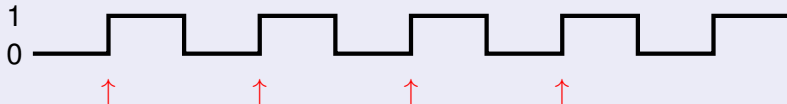
- synchronizes the components in the data path
- the faster the clock, the faster the program executes
  - clock rate is limited by the slowest component!