Introduction to Computer Science

Lecture 5: Instructions, Assembly Language, and Machine Code

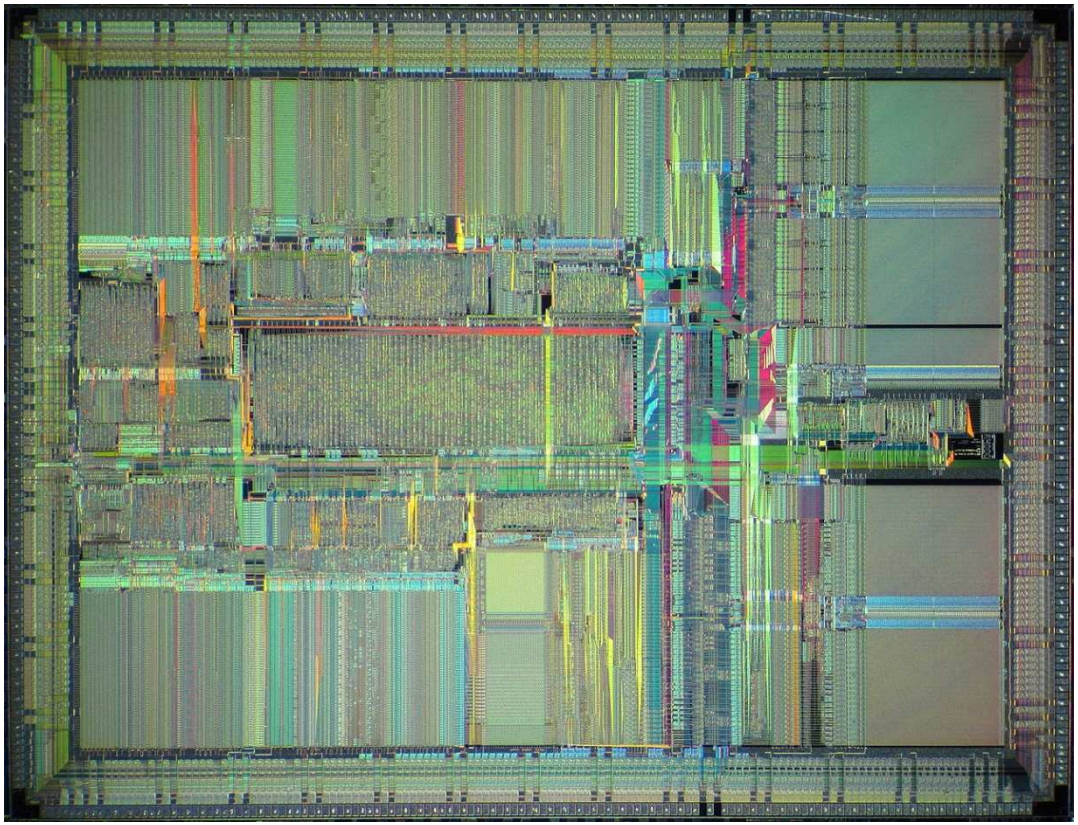
Iain Styles
I.B.Styles@cs.bham.ac.uk

Overview

- Last time – the von Neumann architecture
- This time – MIPS R4000 in detail
  - Structure
  - Features
  - Instruction Set

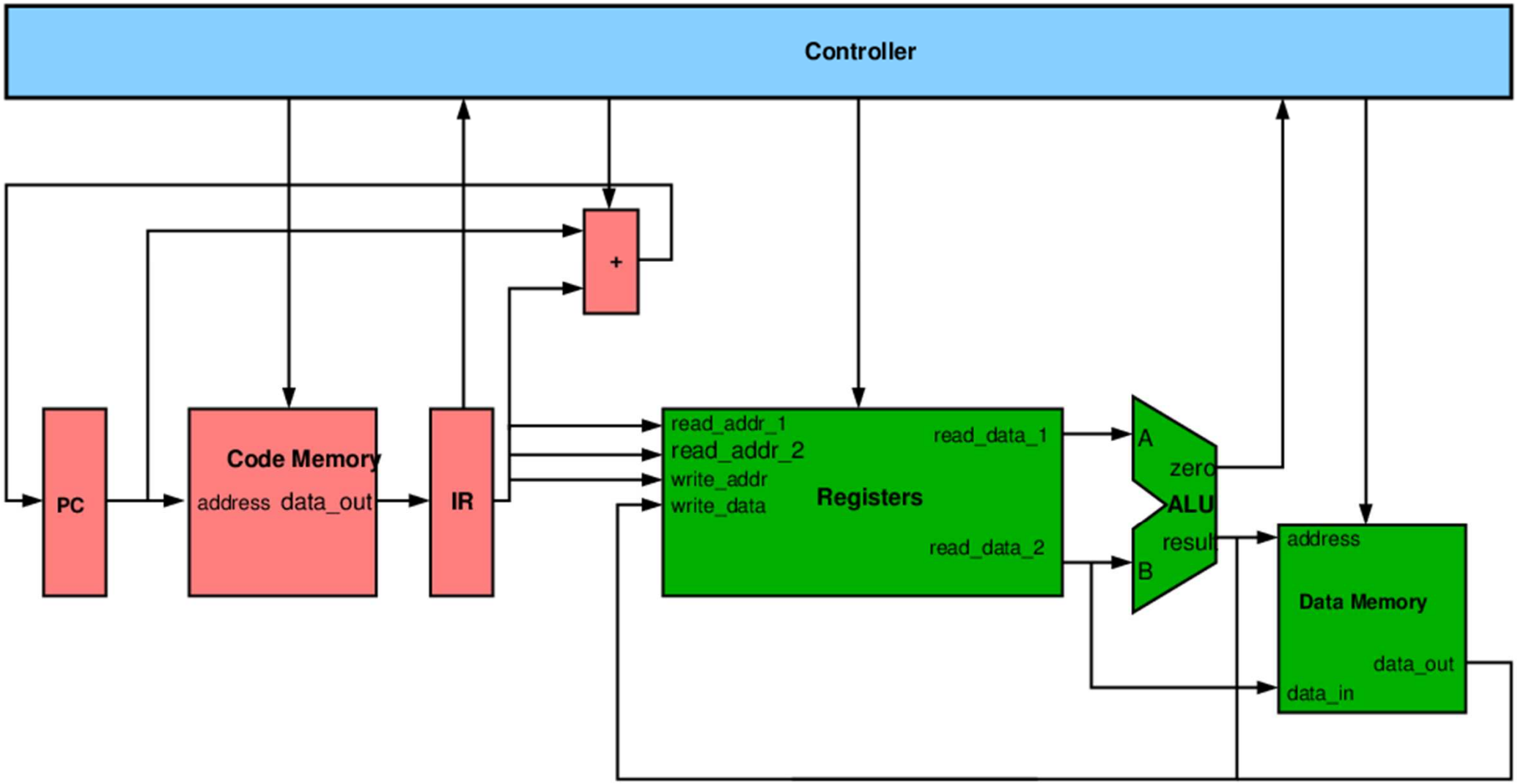
MIPS R4000

- Similar to von Neumann but separate interfaces to instructions and data
  - Modified ***Harvard*** architecture



Case Study: MIPS R4000

- Simple, von Neumann-like architecture
- Simple, well-behaved programming model
- Compact set of instructions with nice regular format
- “Easy” to translate from high-level code to machine language


Controller
+
PC
Code Memory
address
data_out
IR
read_addr_1
read_addr_2
write_addr
write_data
Registers
read_data_1
read_data_2
A
zero
ALU
result
B
address
Data Memory
data_out
data_in

Key Features

- Program counter, instruction register, control unit
- 32 x 32-bit registers, denoted $0-$31
  - $0 always stores 000...000
- Arithmetic and Logic unit
  - Takes input from up to two registers
  - Sends output to registers only
    - One exception: also used to calculate memory addresses

Instruction Sets

- Computers do not understand Java by themselves
  - Several layers of software between your code and the machine itself
- Java: machine independent high-level language
  - Runs on anything with a JVM
  - Language has many sophisticated features
  - Can do complex things with little code
- Machine Level: programs composed of complex combinations of simple “instructions”
- Each machine has a unique “instruction set”

Instruction Sets

- The instruction set is what the programmer “sees” and “uses”
- Set of operations that that CPU can execute
- Each MIPS instruction has 32-bit representation
  - Binary representation – **machine code**
  - Human readable **assembly language**
- High-level program, describe “what” and “how”
- Low-level program, also specify “where”

Very Simple Example

- How to add two numbers?
- In Java: a = b + c;
- In MIPS assembly language, need to know:
  - Where is b?
  - Where is c?
  - Where to put the result (where is a)?
  - What instruction(s) should be used?

Very Simple Example

- a = b + c
- We need to
  - Put b and c into the registers
    - **Load** them from the memory
  - Add them together
    - The result will go into a register
  - Put the result into the memory location of a
    - **Store** it

Very Simple Example

Load b:	lw $8, &b
Load c:	lw $9, &c
Add:	add $10, $8, $9
Store as a:	sw $10, &a

Another Example

a = b + c - d

Load b:	lw $8, &b
Load c:	lw $9, &c
Load c:	lw $10, &c
Add:	add $11, $8, $9
Sub:	sub $11, $11, $10
Store as a:	sw $11, &a

A Bit More Complex

```
                                  if(a==1) b=0;
                                  else b=1;

Load a:                           lw $8, &a
Value 1:                          li $9, 1
If a!=1:                          bne $8, $9, L1
b=0:                              li $10,0
jump:                             j L2
b=1:                    L1:       li $10,1
end:                    L2:       sw $10,&b
```

Summary

- We’ve seen some more detail on MIPS
- And programmed it in its own language
- Next, more examples