Introduction to Computer Science

Lecture 4: How computers execute programs

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

Recap

- Last time:
  - Fixed point numbers
  - Arithmetic in binary
- This time:
  - Negative numbers
  - Floating point
  - Introduction to how computers execute programs

How Computers Execute Programs

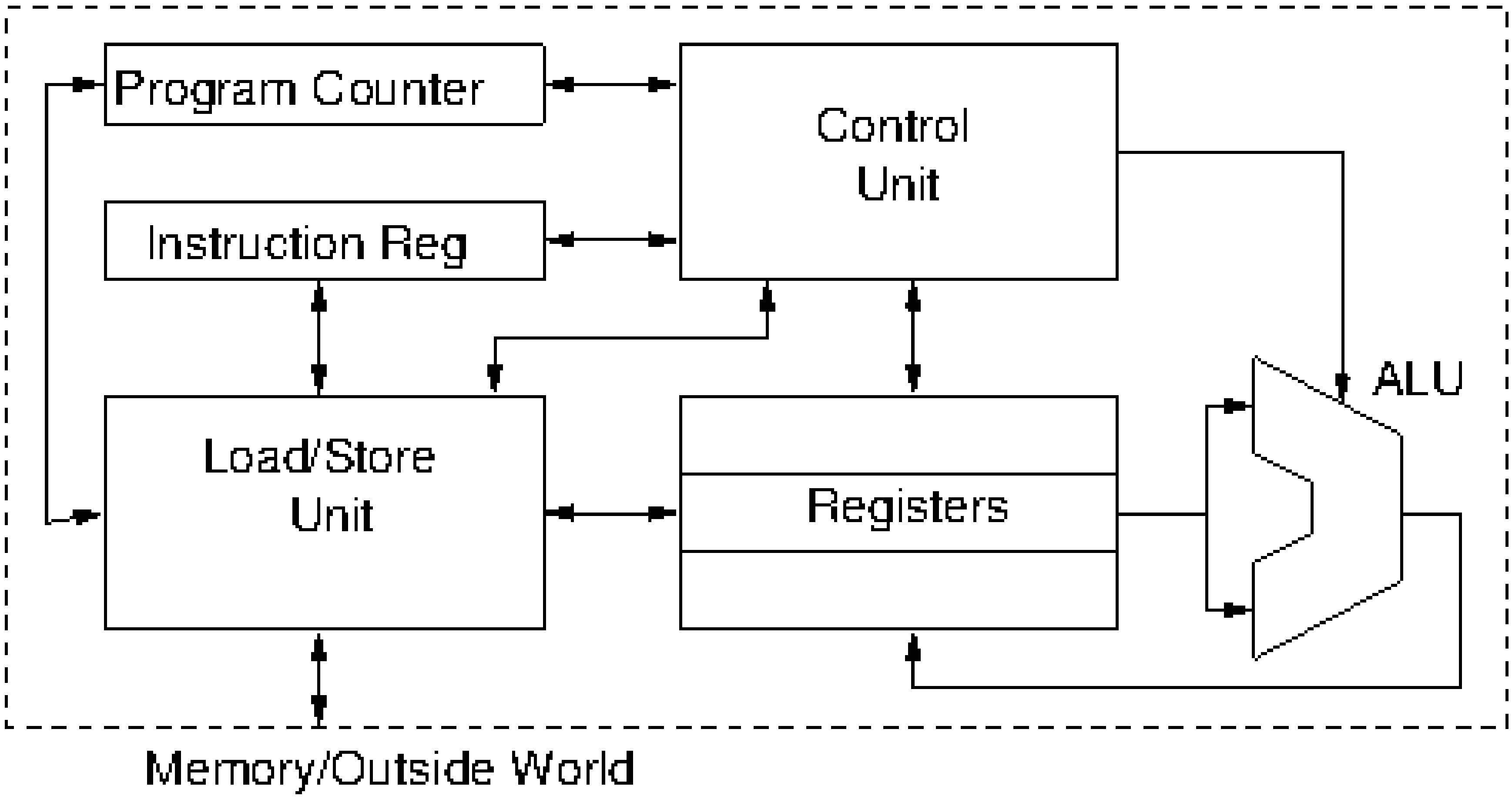
Here is a simple computer program

```
a = input(‘Enter a number’);
if a<1 exit(’Invalid input’);
b=1;
for(i=1;i<=a;i++) {
   b=b*i;
}
print (a,’! = ’,b);
```

Requirements of a Computing Device

- Load the program from some external device/memory
  - *interface to outside world*
- Process instructions in the correct order
  - *mechanism to keep track of progress, local storage and decoding of instructions*
- Access pieces of data in accordance with the program's instructions
  - *local storage of data*
- Perform computations
  - *a calculation "engine"*
- Take decisions according to the results of the computations
  - *mechanism for control*
- Send the results of the the computations to some external device
  - *interface to outside world*

The von Neumann Model



Executing Programmes

1) Load
   - Put the programme somewhere accessible (main memory)
2) Fetch
   - Load instructions from memory into the CPU
3) Decode
   - Determine what the instruction is supposed to do, fetch any data, configure the CPU
4) Execute
   - Perform the calculation

In the context of the von Neumann model...

Stages of Execution

1) Load
   - *Put programme in memory and **address** of first instruction into PC*
2) Fetch
3) Decode
4) Execute

Stages of Execution

1) Load
2) Fetch
   - *Fetch next instruction from memory:*
     IR←memory(PC)
   - *Update PC → next instruction*
3) Decode
4) Execute

Stages of Execution

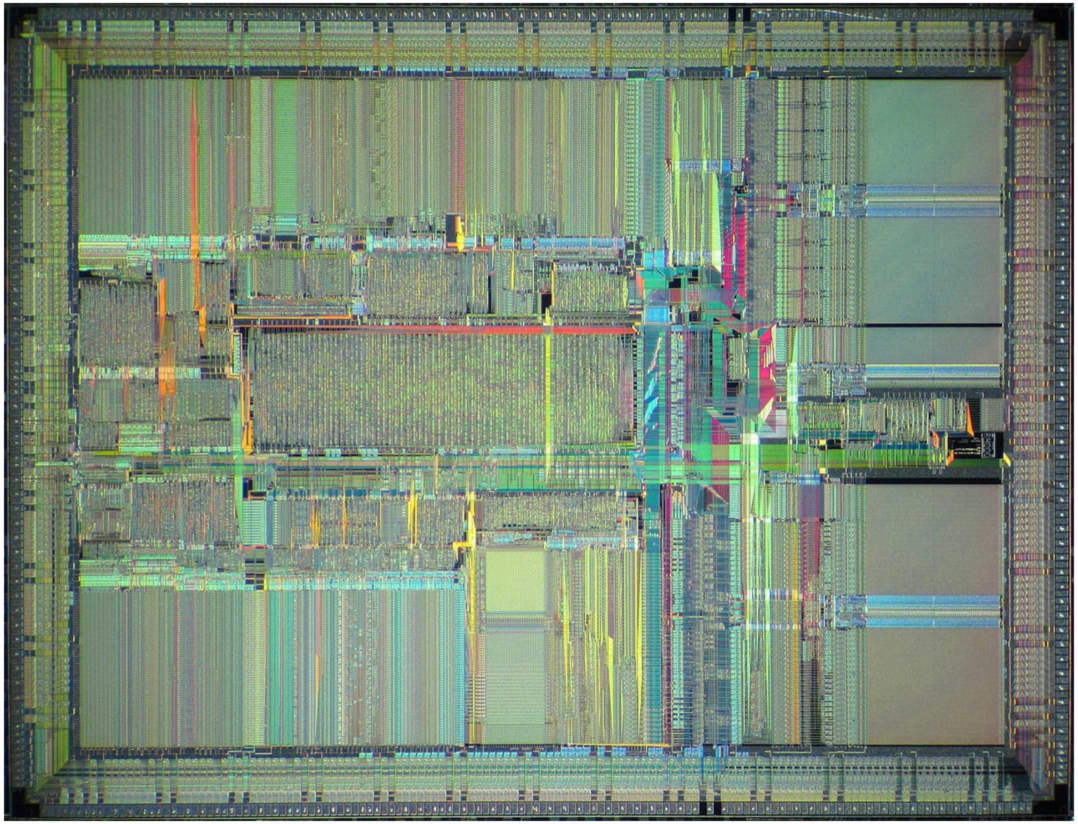
1) Load
2) Fetch
3) Decode
   - *Determine type of instruction*
   - *Calculate memory address of data*
   - *Fetch data if necessary*
4) Execute

Stages of Execution

1) Load
2) Fetch
3) Decode
4) Execute
   - *Do calculation*
   - *Store result*
   - *Update PC if a control instruction*

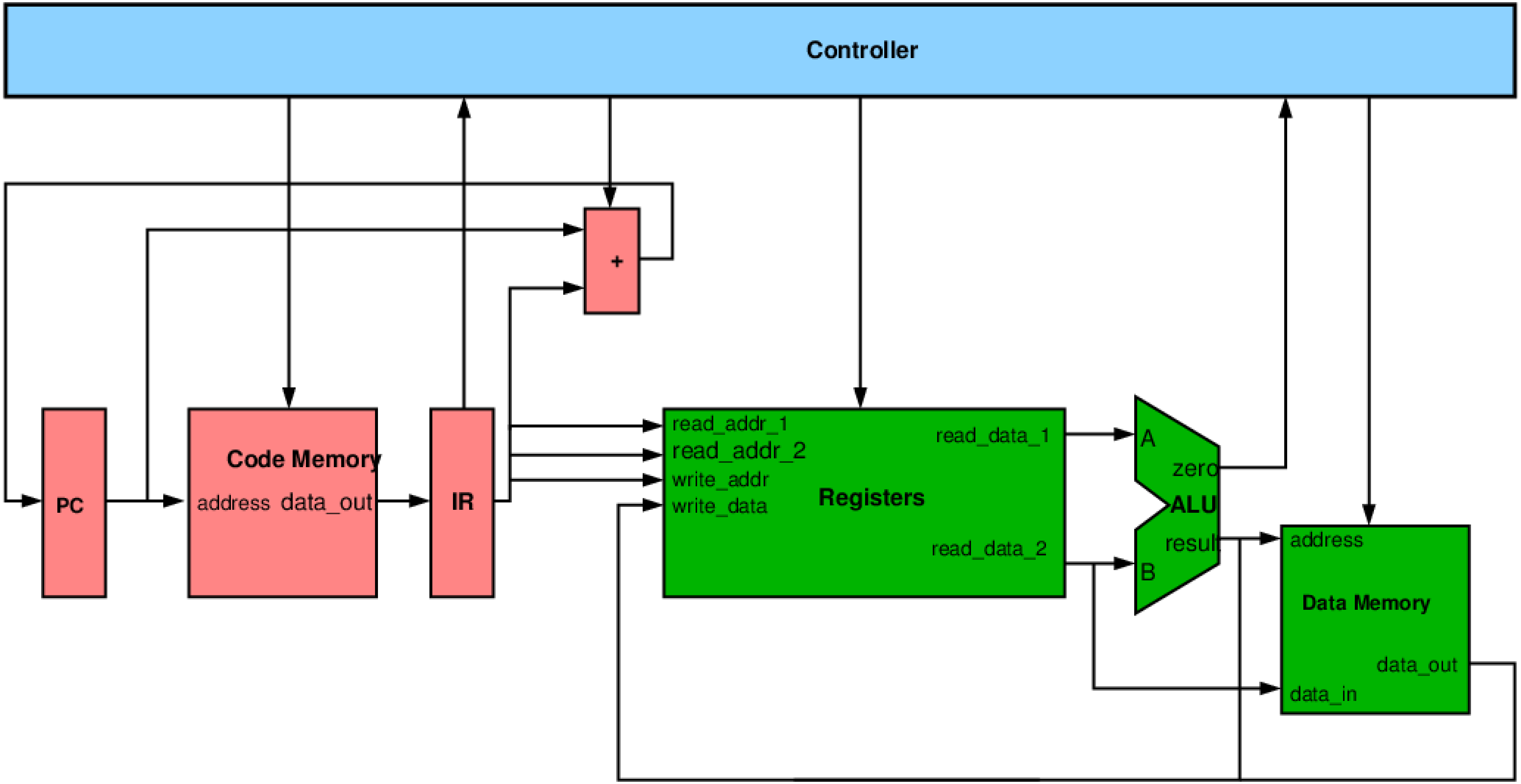
Case Study: MIPS R4000

- 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

Summary

- We've considered the requirements of a simple computer program
- Studied at an outline design that could execute it.
- Looked at a real example
- Next time: code that the computer can understand