Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

2.1 – Microprocessor Basics

CSU11021 – Introduction to Computing I

Dr Jonathan Dukes | jdukes@tcd.ie
School of Computer Science and Statistics

Simple Model of a Microprocessor System

A **processing unit** (or **processor or CPU)** which performs operations on data

Memory, which stores:

Data: representing text, images, videos, sensor readings, π, audio, etc. …

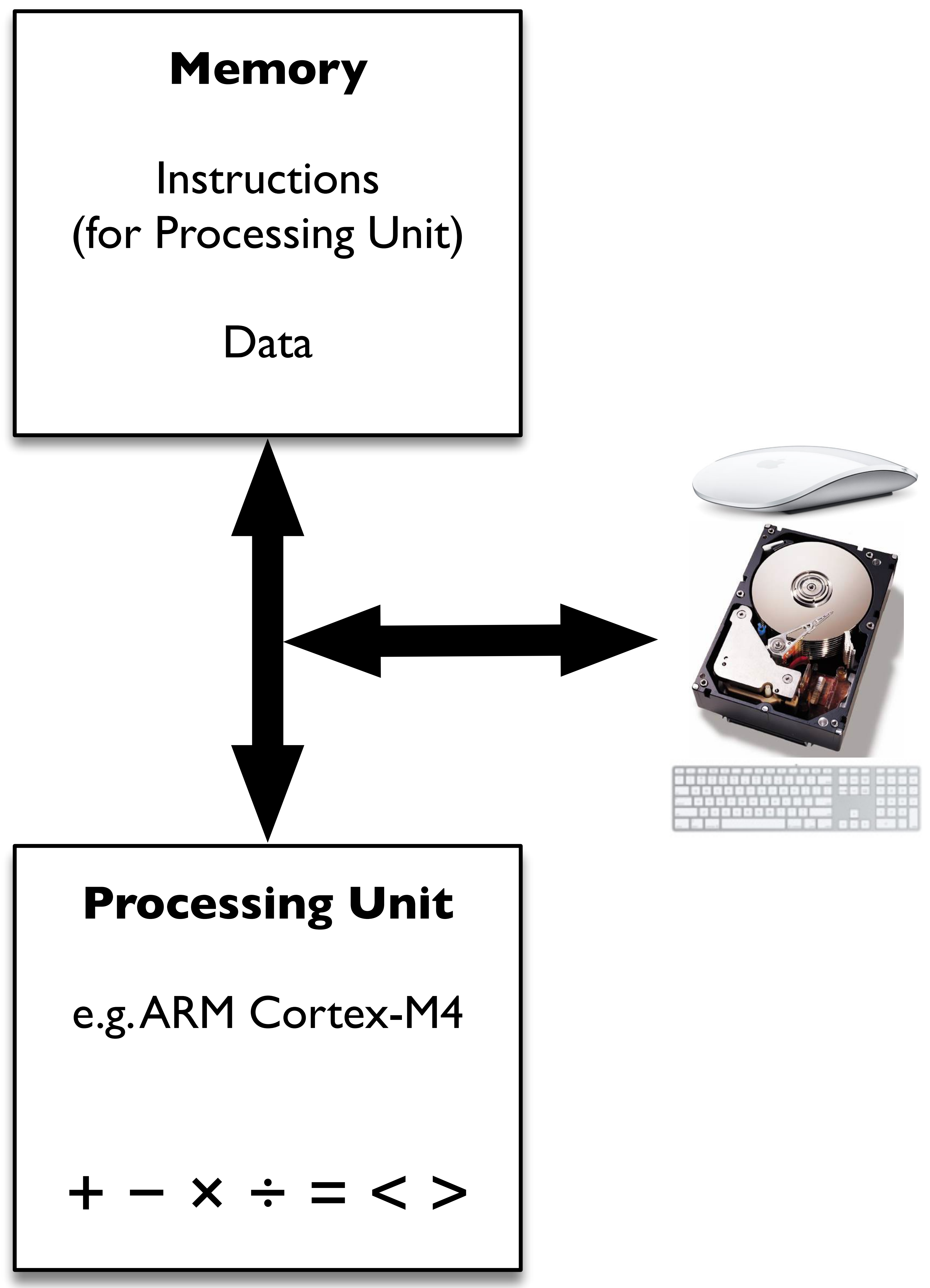
Instructions: Programs are composed of sequences of instructions that control the actions of the processing unit

Instructions typically describe very simple operations, e.g.

Add two values together

Move a value from one place to another

Compare two values



Simple Model of a Microprocessor System

Memory can be viewed as a series of equally-sized“**locations**”, each of which stores a small piece of information

Each location has a unique “**address**”

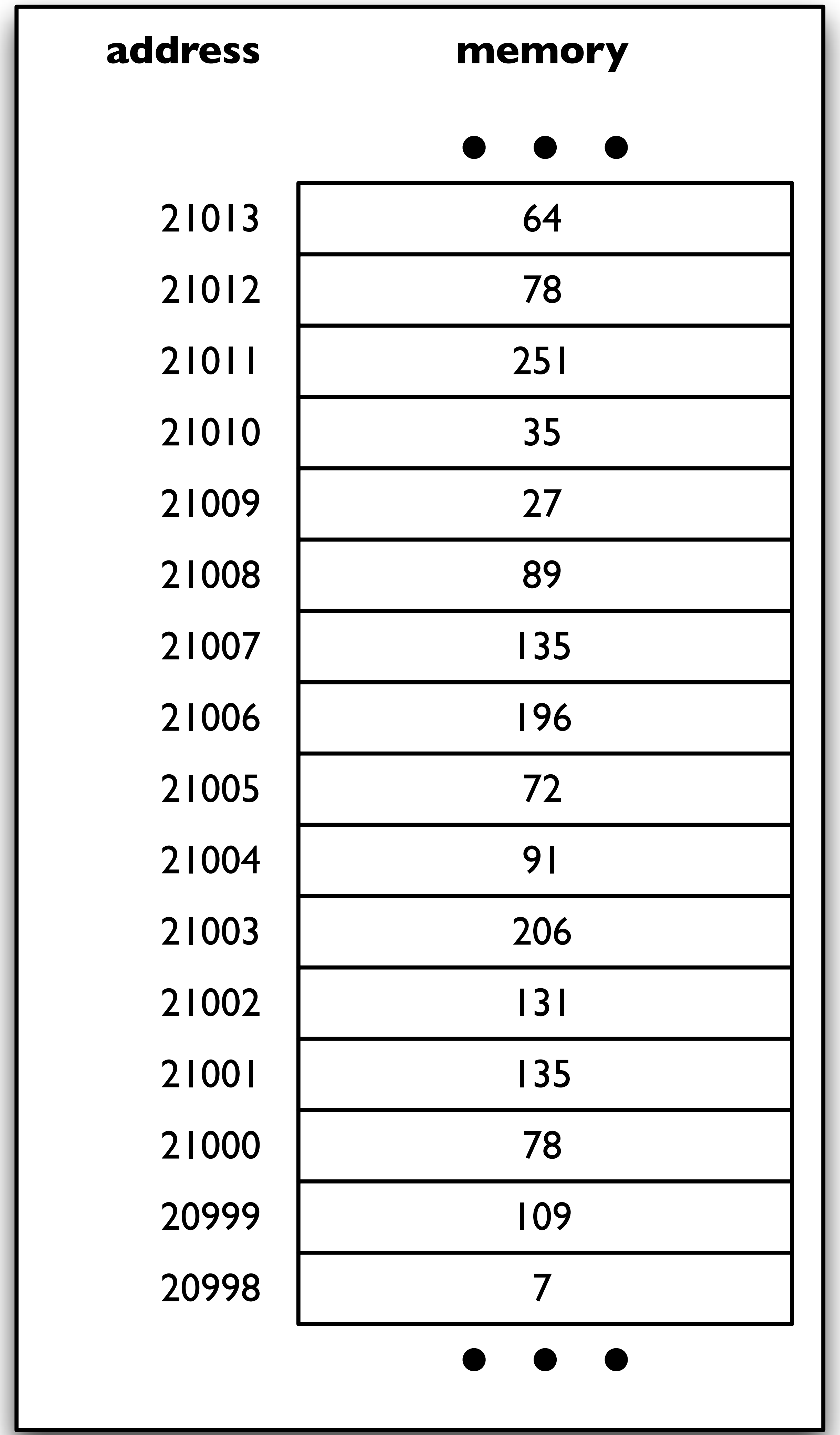
The information at each location may be

data, e.g. the value 91 or

an **instruction** that tells the processor how to manipulate the data

But instructions are also encoded as values!!

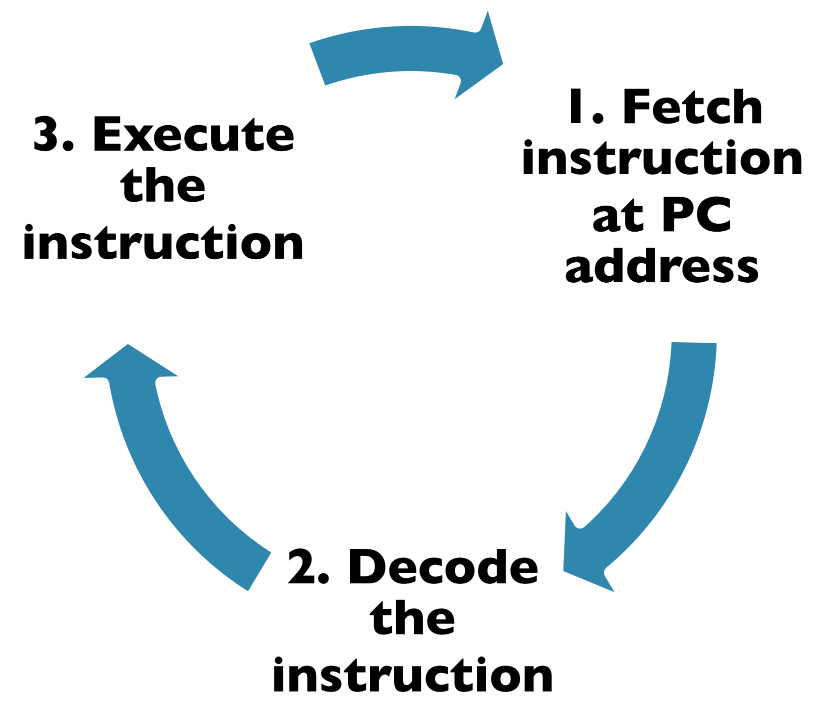
e.g. the value 91 might be a code used to tell the processing unit to add two other values together

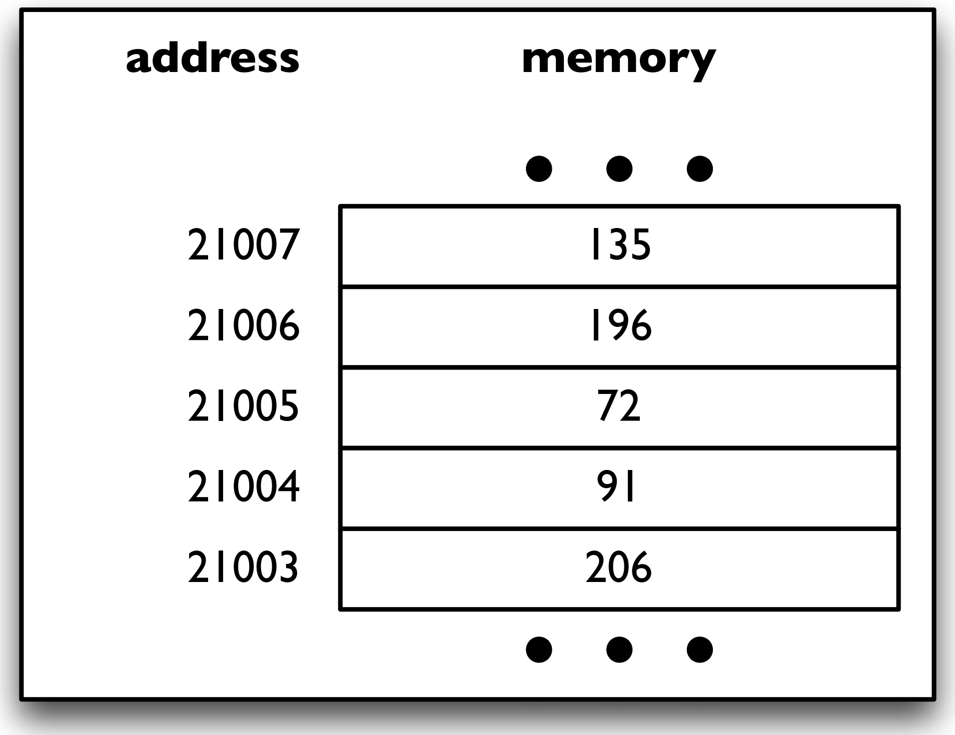


address	memory
	• • •
21013	64
21012	78
21011	251
21010	35
21009	27
21008	89
21007	135
21006	196
21005	72
21004	91
21003	206
21002	131
21001	135
21000	78
20999	109
20998	7
	• • •

Program Execution

When the computer is turned on, the processing unit begins executing the instruction in the memory at the address stored in the Program Counter or PC


1. Fetch instruction at PC address
2. Decode the instruction
3. Execute the instruction



address	memory
	• • •
21007	135
21006	196
21005	72
21004	91
21003	206
	• • •

After fetching an instruction, the value of the Program Counter is changed to the address of the next instruction in the program

Processing unit keeps doing this until the computer is turned off

Program Execution

This simple model of a microprocessor system (computer) is the model used by computers familiar to us (PCs, games consoles, mobile phones, engine control units, ...)

Behaviour is predictable (deterministic)

“On two occasions I have been asked, — 'Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?' … I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.”

Charles Babbage, Passages from the Life of a Philosopher (1864), ch. 5 "Difference Engine No. 1"

The “power” of computers arises because they perform a lot of simple operations very quickly

The complexity of computers arises because useful programs are composed of many thousands or millions of simple instructions

Possibly executing in parallel on more than one processor/computer!

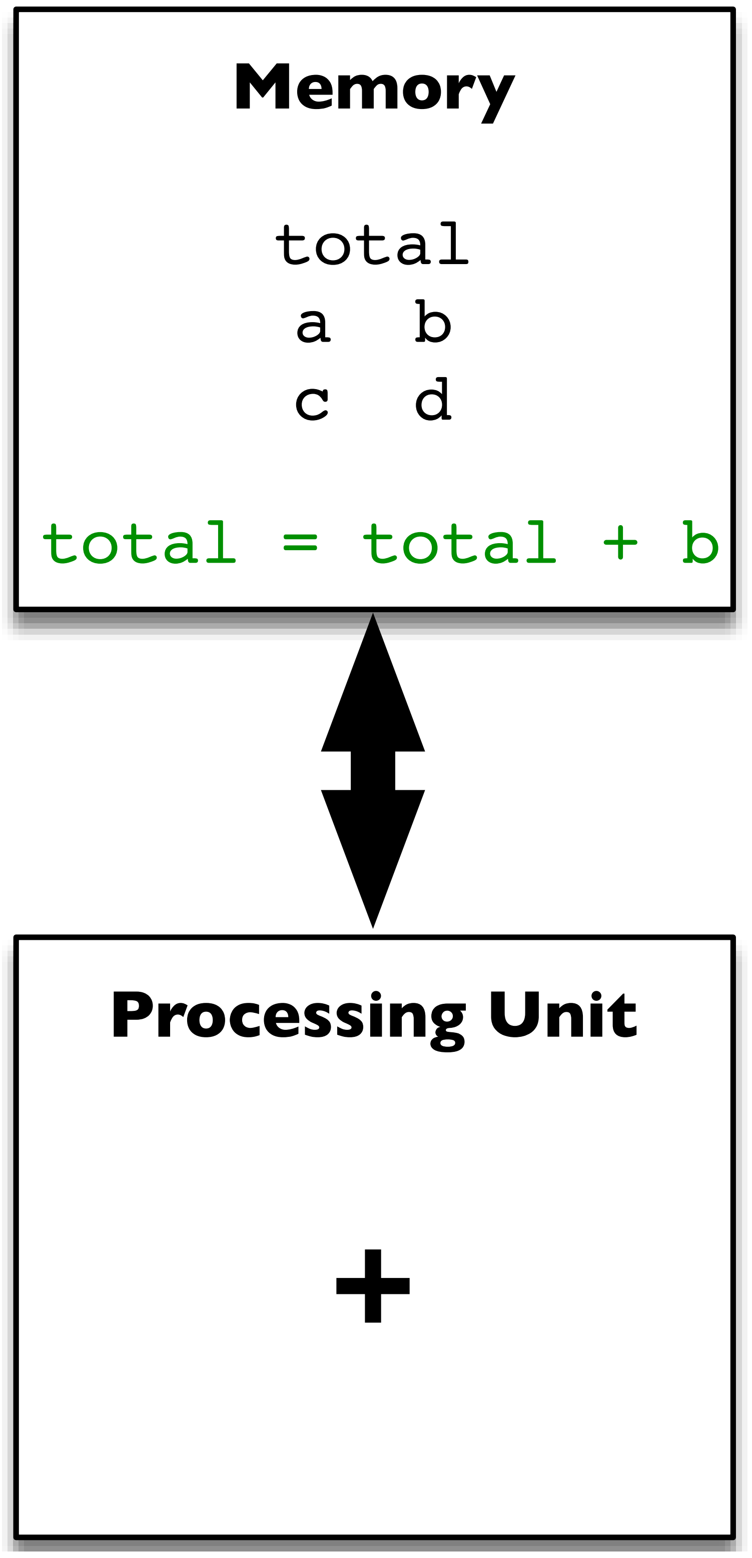
Registers

```
MOV  total, a           Make the first number the subtotal
ADD  total, total, b    Add the second number to the subtotal
ADD  total, total, c    Add the third number to the subtotal
ADD  total, total, d    Add the fourth number to the subtotal
```

Recall our simple program from Lecture #1

- Add four numbers together
- total = a + b + c + d
- total, a, b, c, and d are stored in memory
- operations (move and add) are performed in the processing unit

Question: How many memory ↔ processing unit transfers?



Registers

total = total + b

Load instruction from Memory to Processing Unit

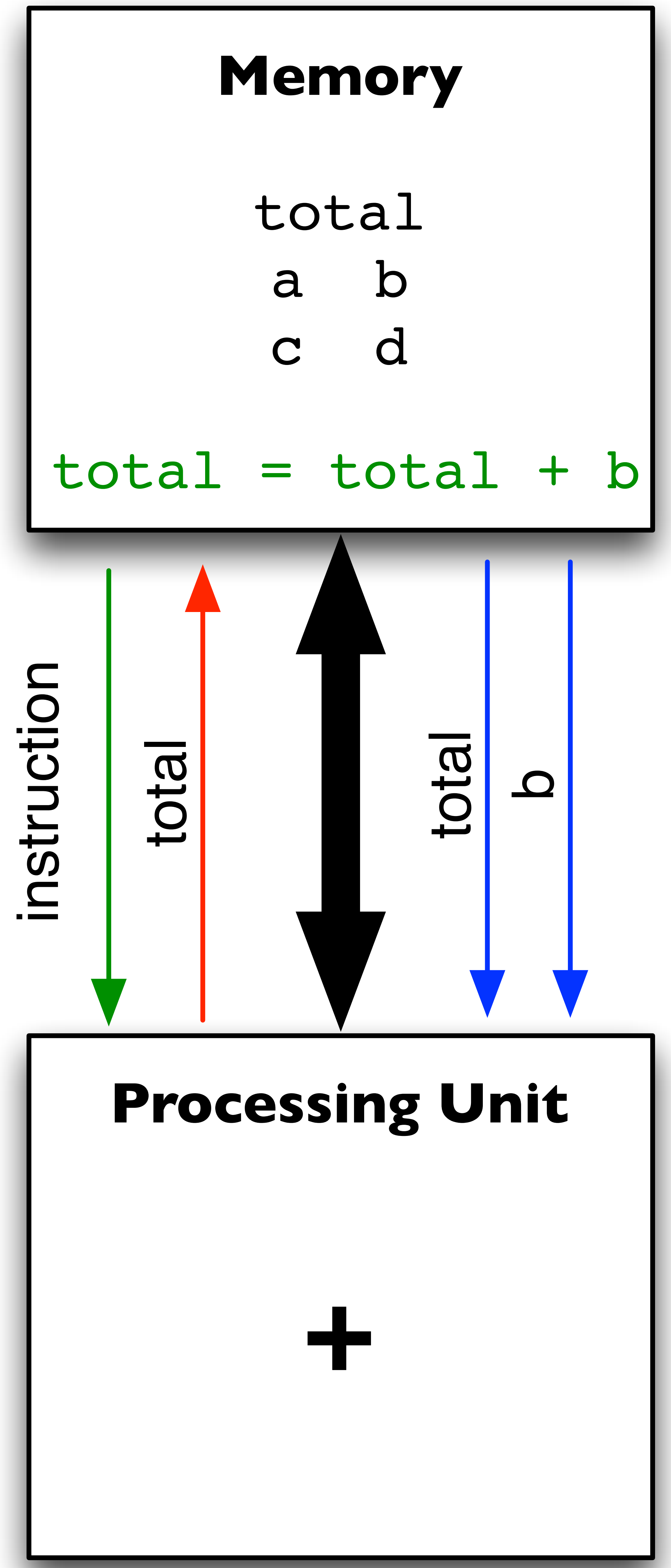
Load total from Memory to Processing Unit

Load b from Memory to Processing Unit

Compute total + b

Store total from Processing Unit to Memory

Accessing memory is slow relative to the speed at which the processor can execute instructions



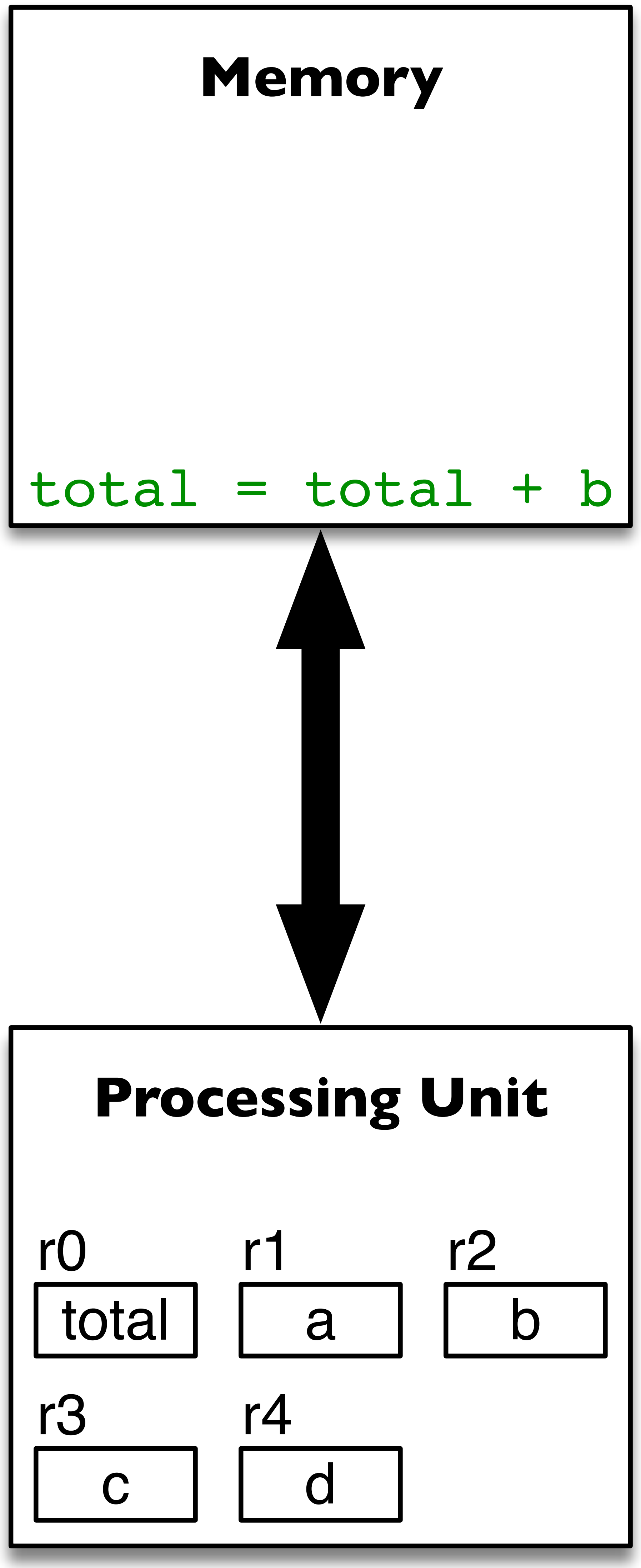
Registers

Processors use small fast internal storage to temporarily store values – called <u>registers</u>

ARM microprocessors have 16 registers

- Labelled R0, R1, …, R15
- R15 is special – the Program Counter
- R13 and R14 are also special (you should avoid using them for now)

Question: How many memory ↔ Processing Unit transfers if total, a, b, c, and d are stored in registers?



Registers

Processors use small fast internal storage to temporarily store values – called <u>registers</u>

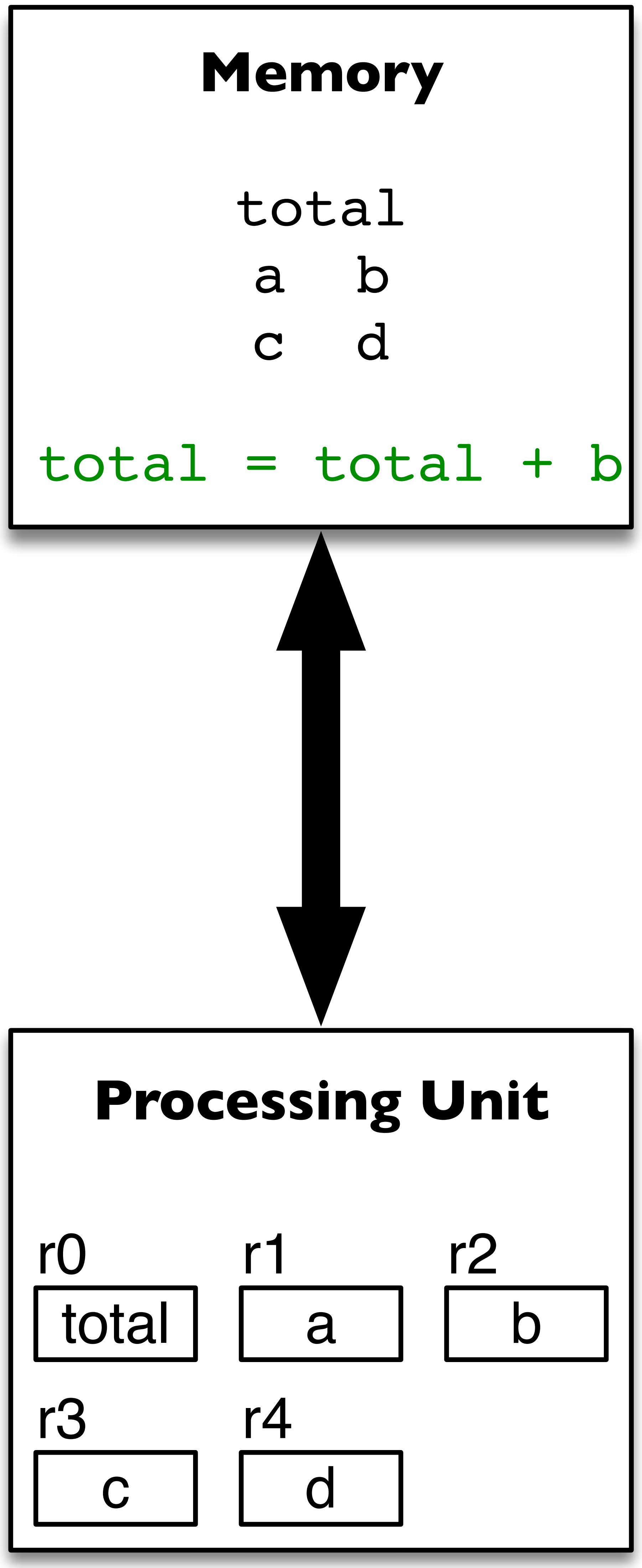
ARM microprocessors have 16 registers

Labelled R0, R1, …, R15

R15 is special – the Program Counter

R13 and R14 are also special
(you should avoid using them for now)

Question: How many memory ↔ Processing Unit transfers if total, a, b, c, and d are stored in registers?



Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

2.2 – Machine Code and Assembly Language

CSU11021 – Introduction to Computing I

Dr Jonathan Dukes | jdukes@tcd.ie
School of Computer Science and Statistics

Machine Code

A program (any program, originally written using any language) is composed of a sequence of **machine code instructions** that are stored in memory

Instructions determine the operations performed by the processor (e.g. add, move, multiply, subtract, compare, …)

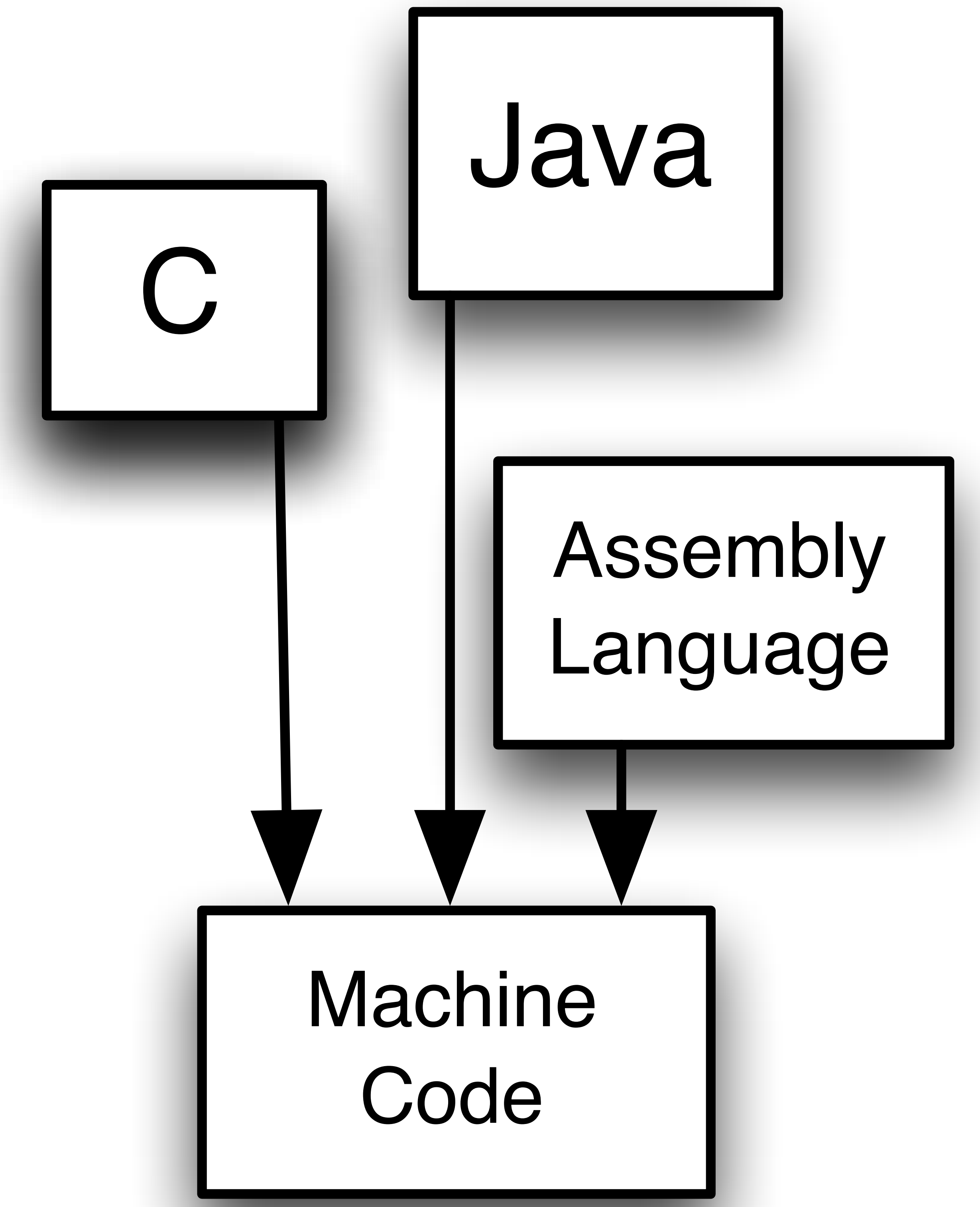
A single instruction is composed of

- an **operator** (the operation to perform, e.g. ✖, ➗, ➖, ➕)
- zero, one or more **operands** (e.g. R1, R2, R3, R4)

a b c d

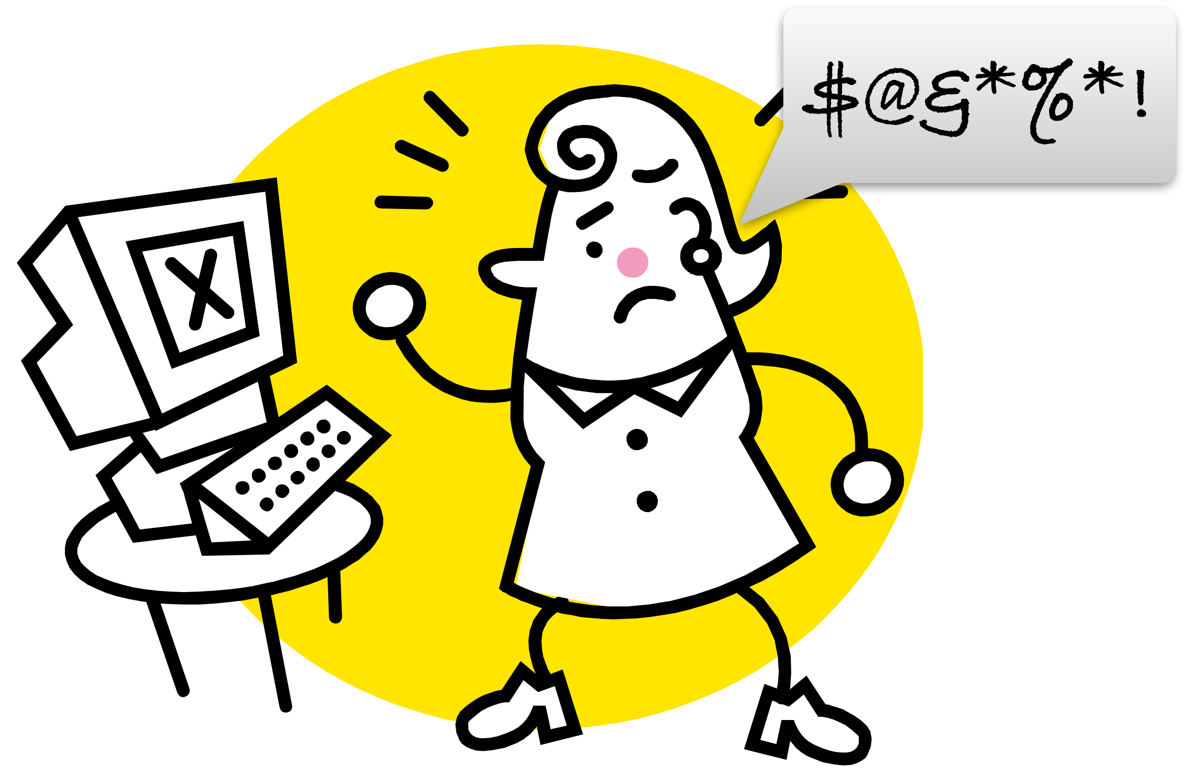
Each instruction and its operands are encoded using a numerical value

- e.g. 3766550530 is the machine code that causes the processor to add the values in R1 and R2 and store the result in R3 (`ADD R3,R1,R2`)



Assembly Language

Writing programs using machine code is possible …
… but not practical



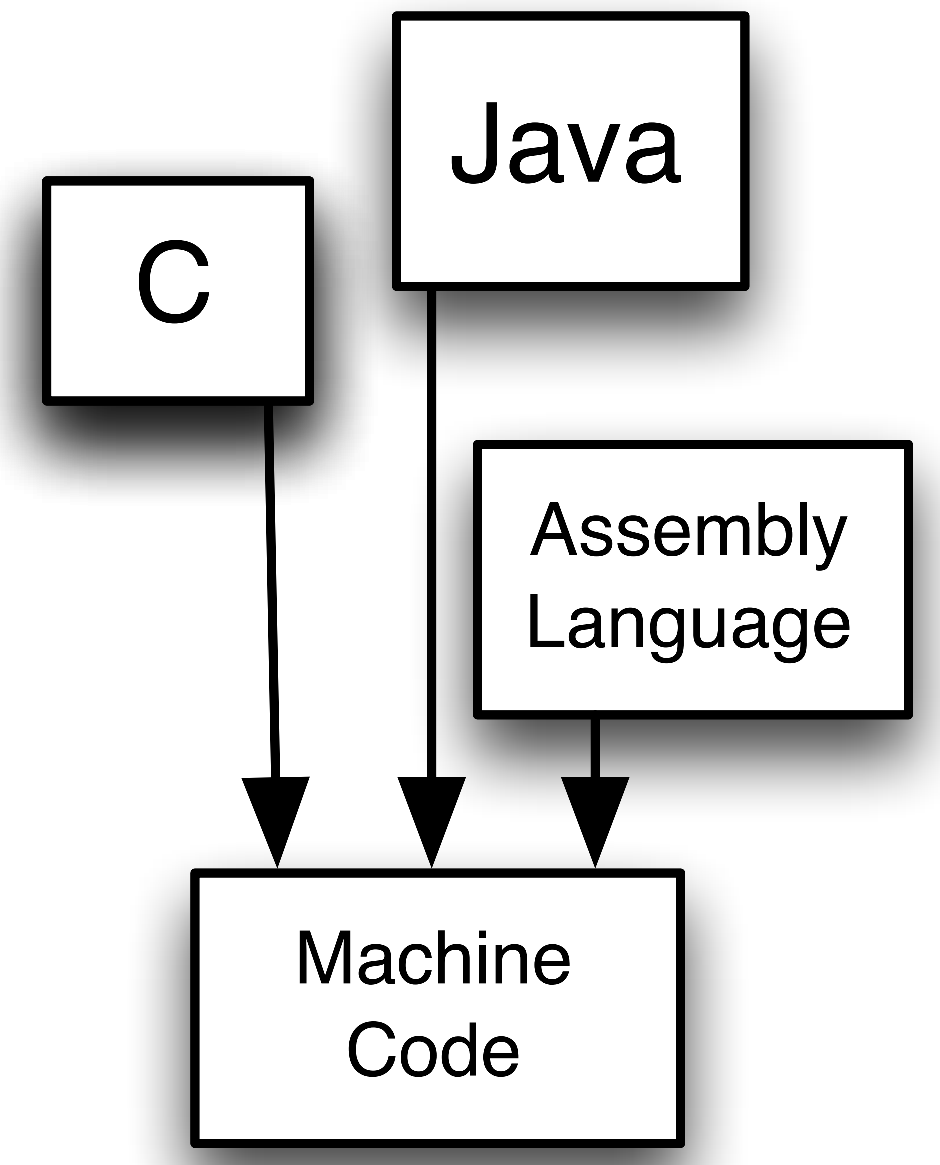
This is you writing a machine code program!!

Instead, we will write programs using assembly language

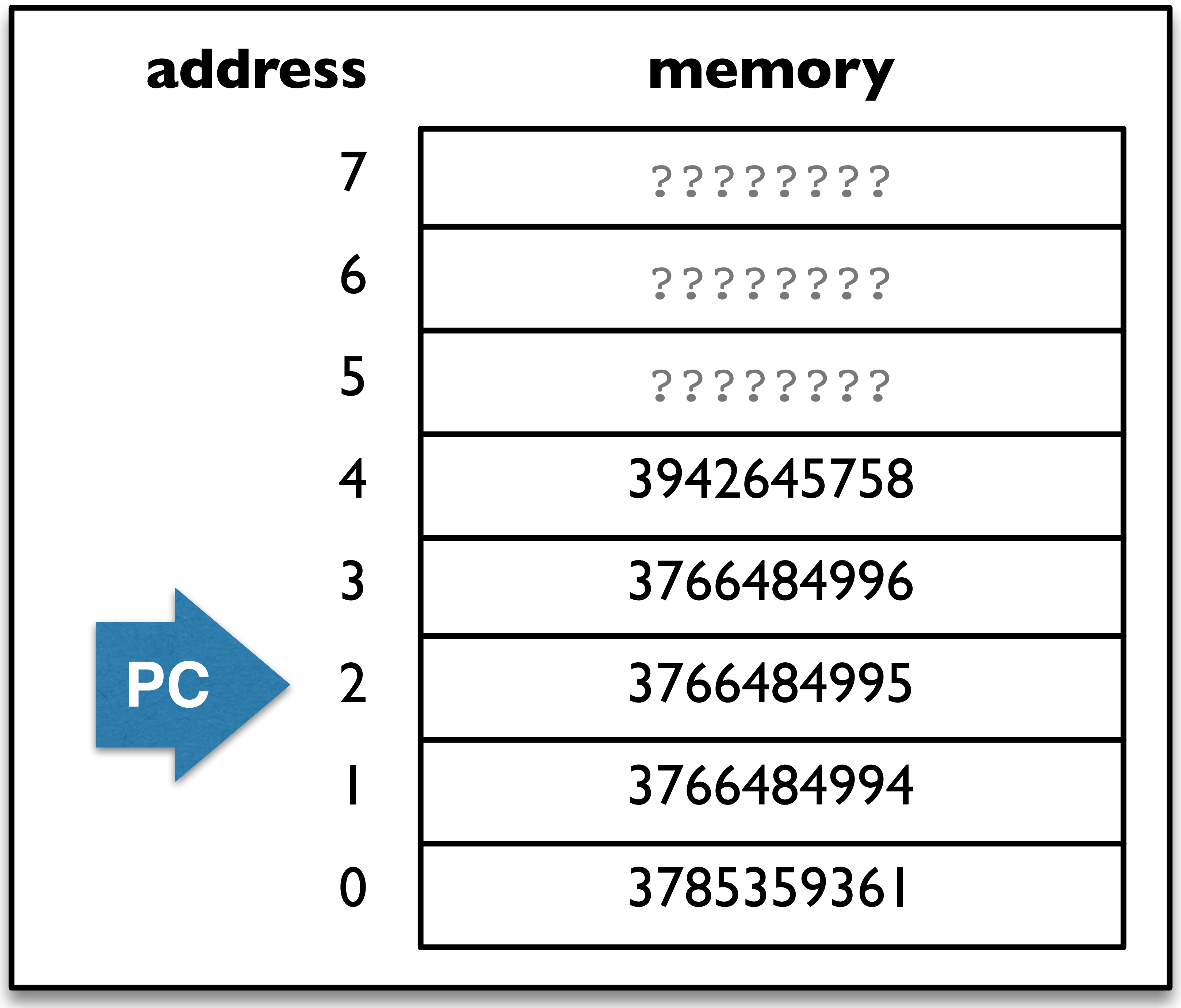
- Instructions are expressed using mnemonics
- e.g. the word "ADD" instead of the machine code 3766550530
- e.g. the expression "R2" to refer to register number two

Assembly language must still be translated into machine code

- This is done using a program called an assembler
- Machine code produced by the assembler is stored in memory and executed by the processor (the **fetch – decode – execute** cycle)

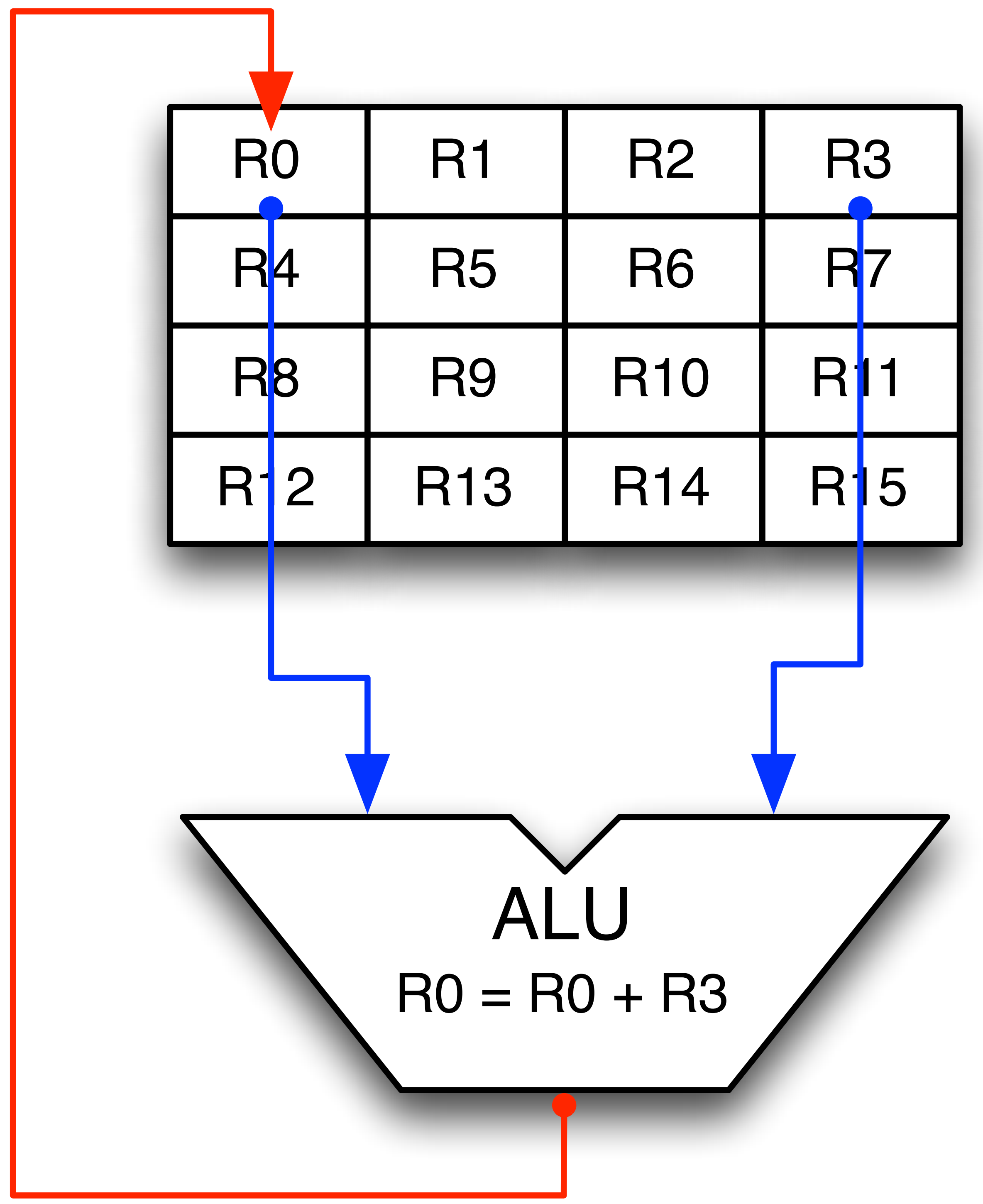


Instruction Execution



address	memory
7	????????
6	????????
5	????????
4	3942645758
3	3766484996
2	3766484995
1	3766484994
0	3785359361

<u>3766484995</u>
operation: add
source R*n*: R0
source R*m*: R3
destination R*d*: R0



Fetch next instruction from memory at the address contained in the Program Counter (PC ≡ R15)

PC advances (increments) to next instruction

Machine Code instruction is decoded to determine operation and source / destination operands

Instruction is executed. (In this example the **ALU** adds the values in R0 and R3, storing the result back in R0.)

ARM Assembly Language Instructions

General form of an ARM Assembly Language instruction:

```
OP    destination, source_1, source_2
```

e.g. Add the values in R2 and R3, store the result in R1

```
ADD    R1, R2, R3        @ R1 = R2 + R3
```

Some ARM instructions have just one input operand

```
MOV    R4, R5        @ R4 = R5
```

Immediate Operands

Register Operands

```
ADD     R0, R1, R2
MOV     R5, R2
```

Register Operand

Often we want to use constant values, instead of registers (variables)

e.g. move the value 0 (zero) into register R3

```
MOV     R0, #0
```

e.g. set R1 = R2 + 1

```
ADD     R1, R2, #1
```

Immediate Operands

Simple Arithmetic

Write an ARM Assembly Language program to compute $4x^2+3x$ if x is stored in register R1. Store the result in register R0.

Cannot use MUL to multiply by a constant value
(a.k.a. immediate operand)

R1 is unmodified by our solution … which may be something we want … or maybe we don't care

```
@ Write a program to compute 4x^2+3x
MUL    R0, R1, R1       @ result = x * x
MOV    R2, #4           @ tmp = 4
MUL    R0, R2, R0       @ result = 4 * x * x

MOV    R2, #3           @ tmp = 3
MUL    R2, R1, R2       @ tmp = x * tmp

ADD    R0, R0, R2       @ result = result + tmp
```

LoaD Register (LDR)

`MOV Rx, #y` can only be used to load certain small values into Registers (Which values and how small? Long story for another day …)

Loa**D** **R**egister instruction can be used to load any* value into a register

```
        LDR     R4, =45673857   ; tmp = 45673857
```

Note use of **=x** syntax instead of **#x** with LDR instruction

```
        LDR     R2, =3          ; tmp = 3
        MUL     R2, R1, R2      ; tmp = x * tmp
```

Note use of **=** again in operand **=3**

If in doubt, use `LDR Rx, =y.`

* well, not quite any

Assembly Language Programming Guidelines

Provide meaningful comments and assume someone else will be reading your code

```
MUL     R2, R1, R2      @ r2 = r1 * r2              ✗

MUL     R2, R1, R2      @ tmp = x * tmp             ✓
```

Break your programs into small blocks separated by white space

While starting out, keep programs simple

Pay attention to initial values in registers (and memory)