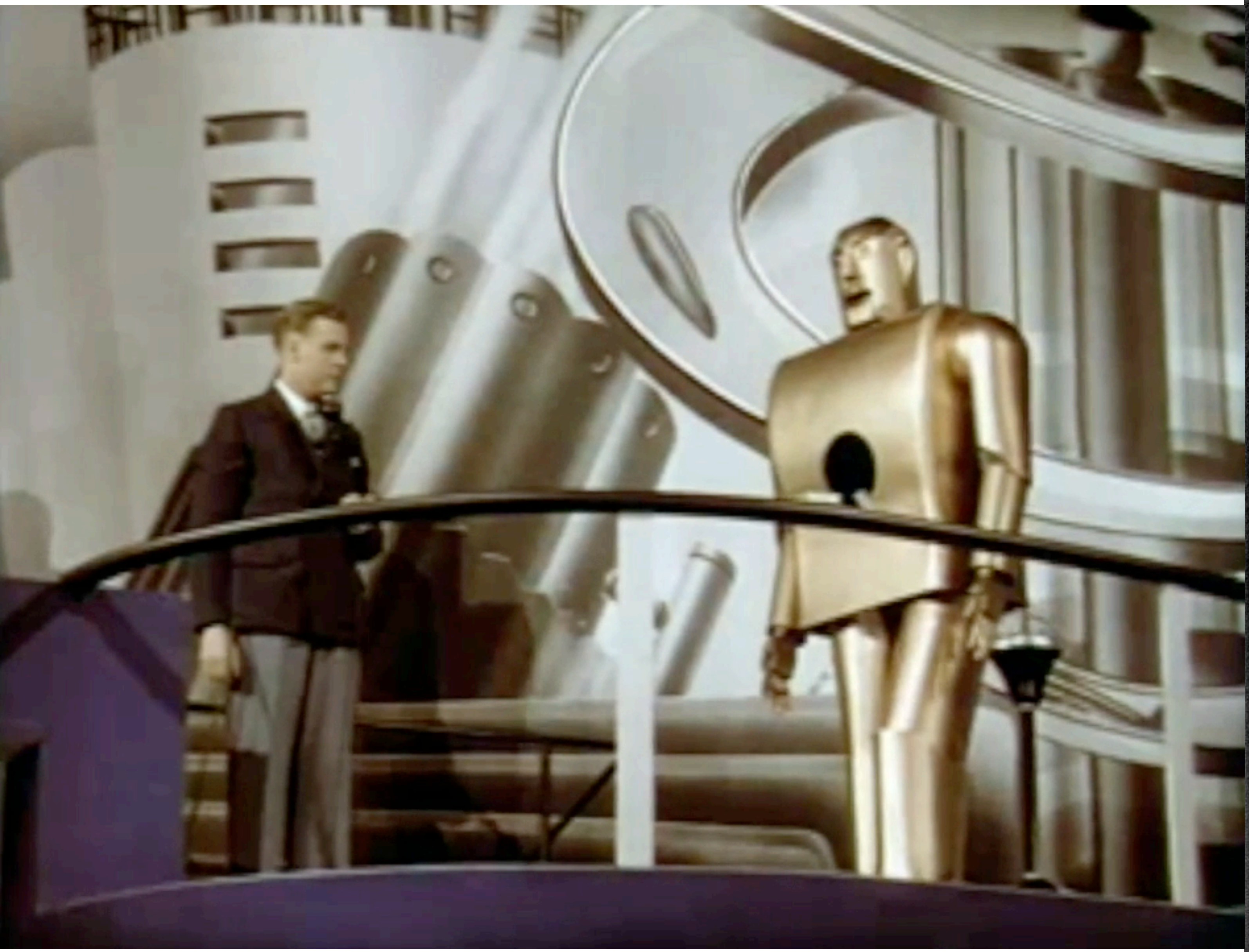
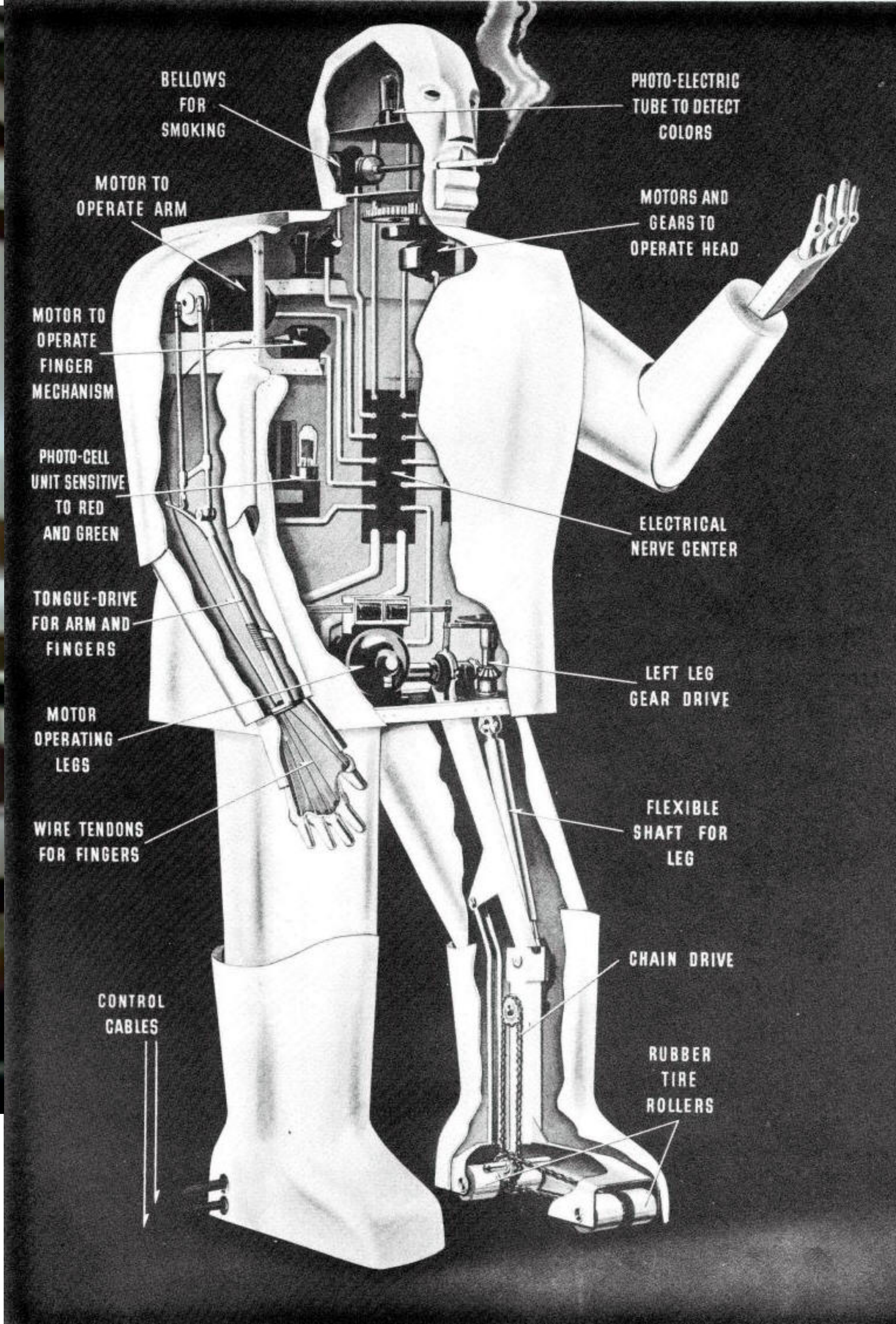
CS151: A Brief Intro to Computers

Prof Matthew Fricke

Ver: 1.0. Send corrections to mfricke@unm.edu.

By the start of the 20th century a lot of physical work had been mechanized.





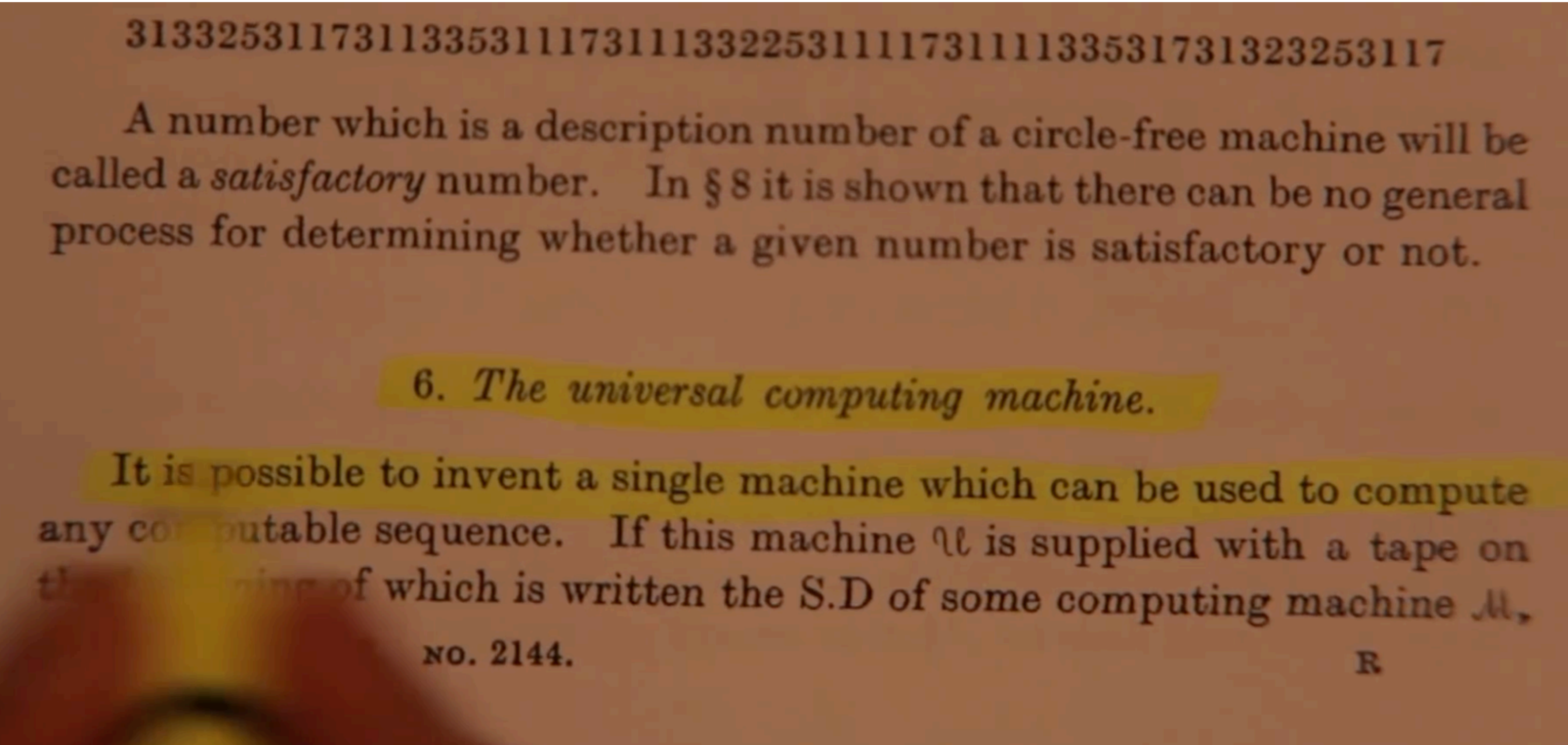
Elektro at the 1939 New York World Fair

Alan Turing was a mathematician and biologist who was interested in how the brain works.

His major breakthrough was to recognize that the human brain can solve many different kinds of problem.

He developed a theoretical machine that worked like a mathematician.

“On Computable Numbers, with an Application to the Entscheidungsproblem” - 1936



313325311731133531117311133225311117311113353173132325311 7

A number which is a description number of a circle-free machine will be called a *satisfactory* number. In § 8 it is shown that there can be no general process for determining whether a given number is satisfactory or not.

6. *The universal computing machine.*

It is possible to invent a single machine which can be used to compute any co[illegible]putable sequence. If this machine $\mathfrak{U}$ is supplied with a tape on t[illegible] of which is written the S.D of some computing machine $\mathcal{M}$,

NO. 2144. R

There were many possible machines each of which could execute a single algorithm.

You could just wire the algorithm into them. They could solve one problem.

There were many possible machines each of which could execute a single algorithm.

You could just wire the algorithm into them. They could solve one problem.

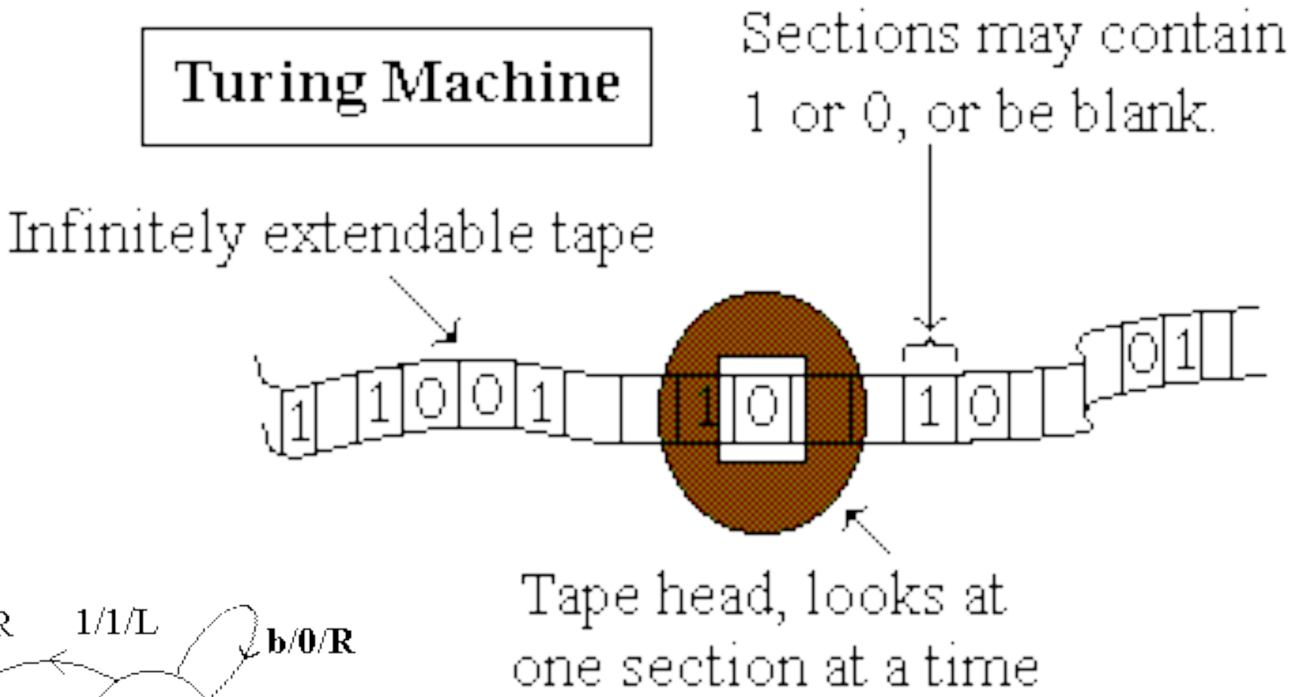
Turing realized he could build a machine that could simulate any other machine.

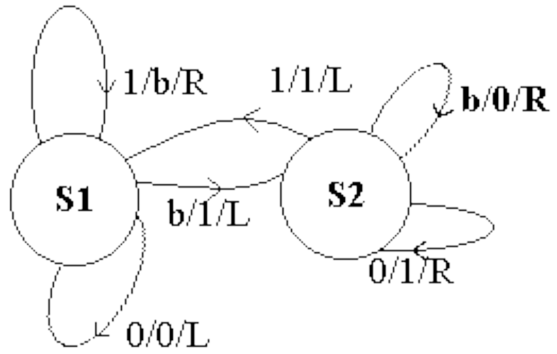
There were many possible machines each of which could execute a single algorithm.

You could just wire the algorithm into them. They could solve one problem.

Turing realized he could build a machine that could simulate any other machine.

Turing Machine


Sections may contain 1 or 0, or be blank.
Infinitely extendable tape
1 1 0 0 1 1 0 1 0 0 1
Tape head, looks at one section at a time



Transition State Diagram of Turing Machine (corresponds to transition state table)

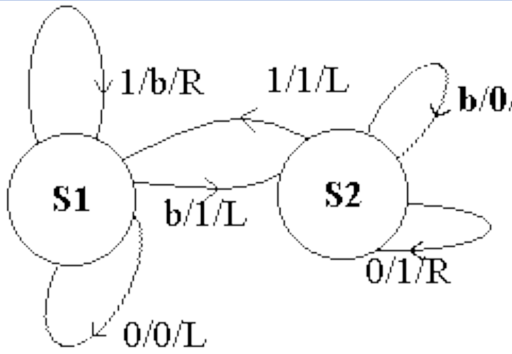
State	Read	Write	Move	Next State
S1	0	0	L	S1
	blank	1	L	S2
	1	blank	R	S1
S2	0	1	R	S2
	blank	0	R	S2
	1	1	L	S1

State Transition Table for a Turing Machine

The essential elements are

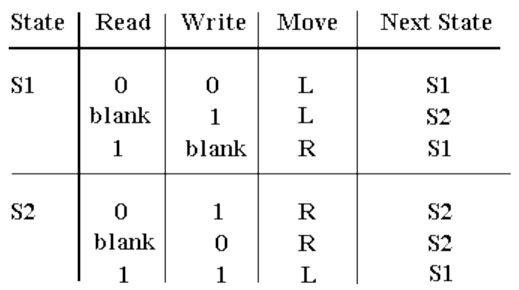
1)Being able to read symbols from a storage
2)Change state depending on the symbol you read
3)Write symbols to the storage
4)Move where you are reading and writing from

5)This is just like a human being



Transition State Diagram of Turing Machine (corresponds to transition state table)

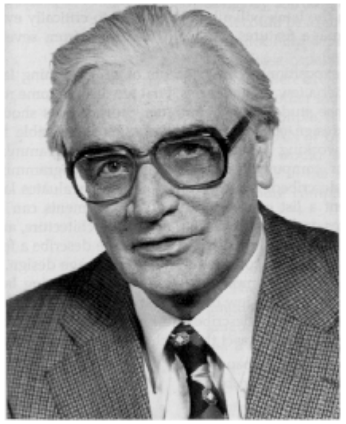
Tape head, looks at one section at a time

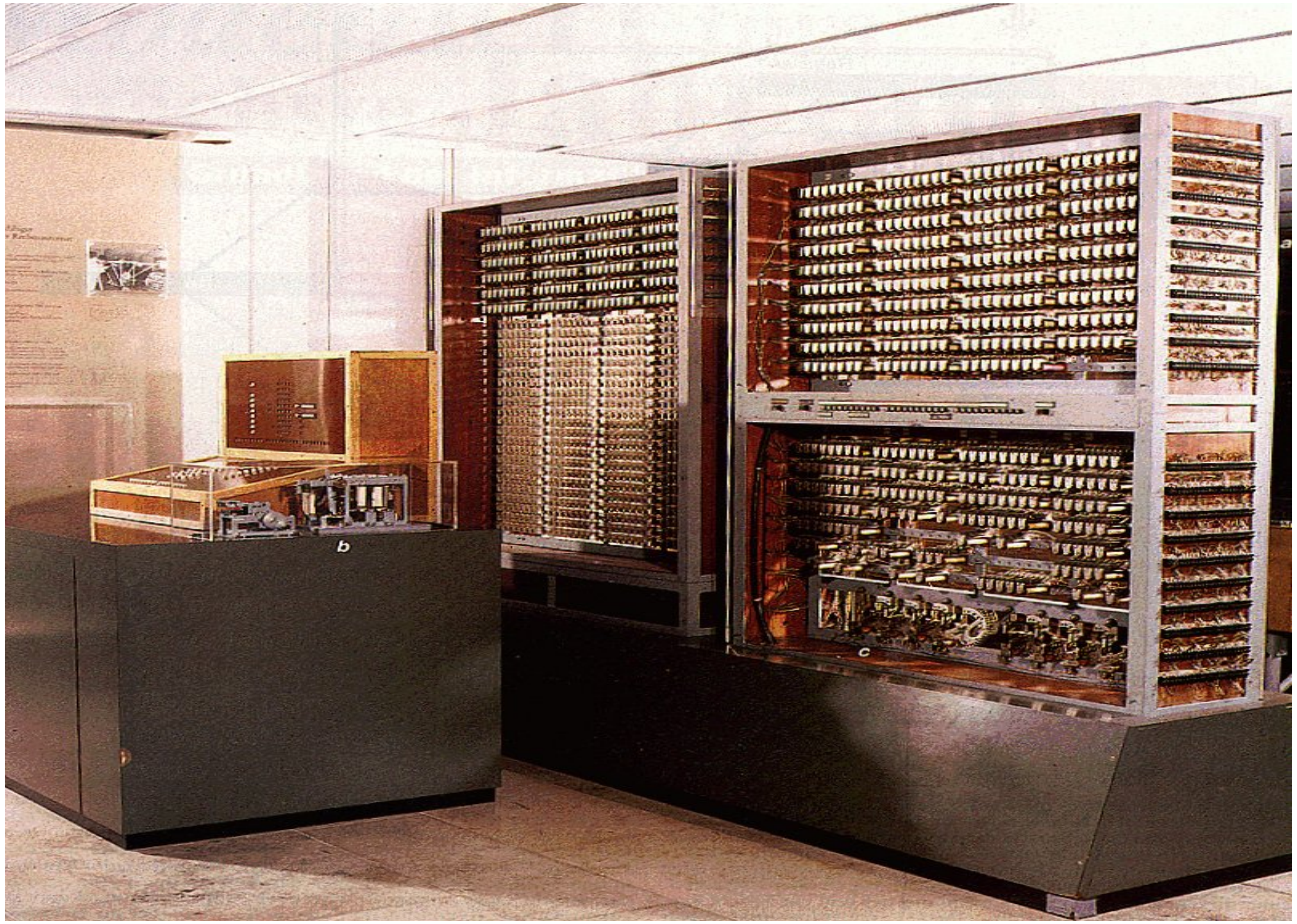


State	Read	Write	Move	Next State
S1	0	0	L	S1
	blank	1	L	S2
	1	blank	R	S1
S2	0	1	R	S2
	blank	0	R	S2
	1	1	L	S1

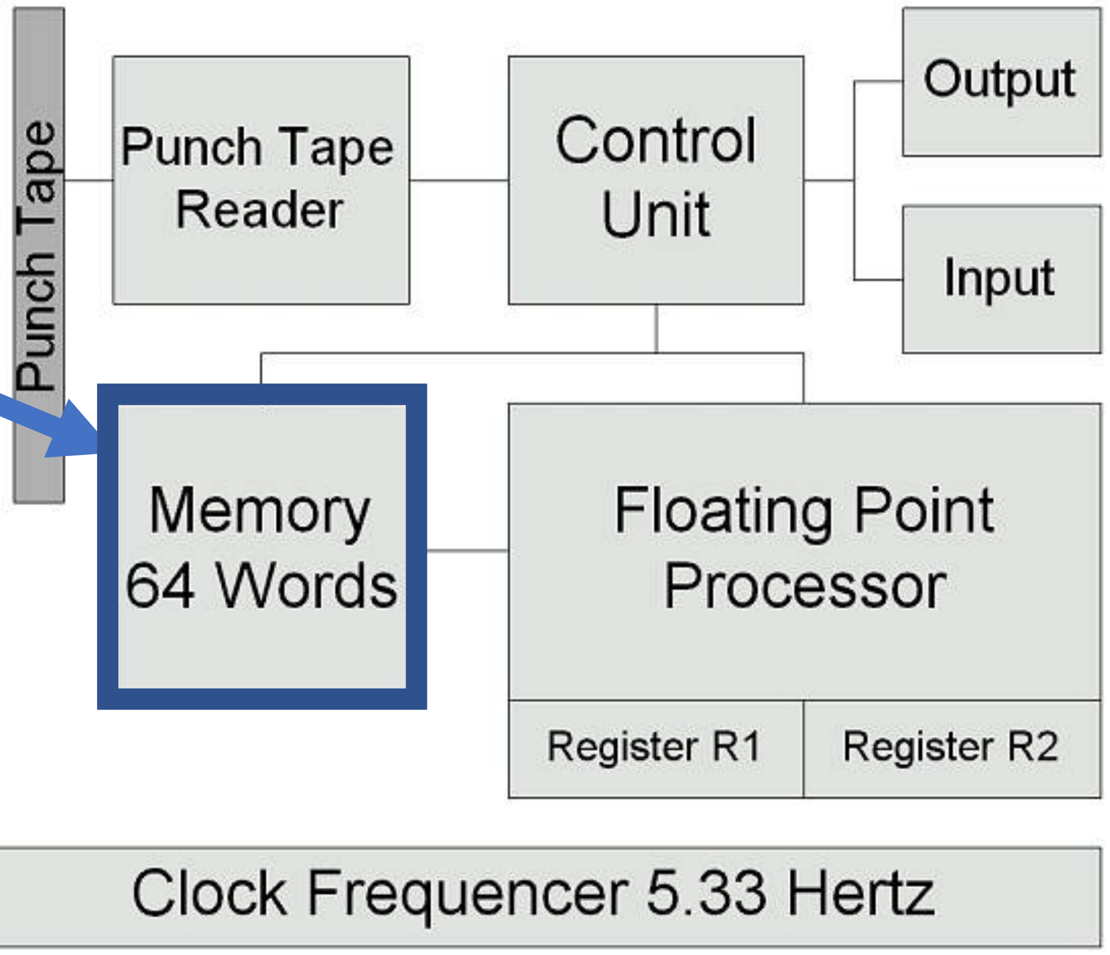
State Transition Table for a Turing Machine

The Z3 built by Konrad Zuze in Berlin 1941



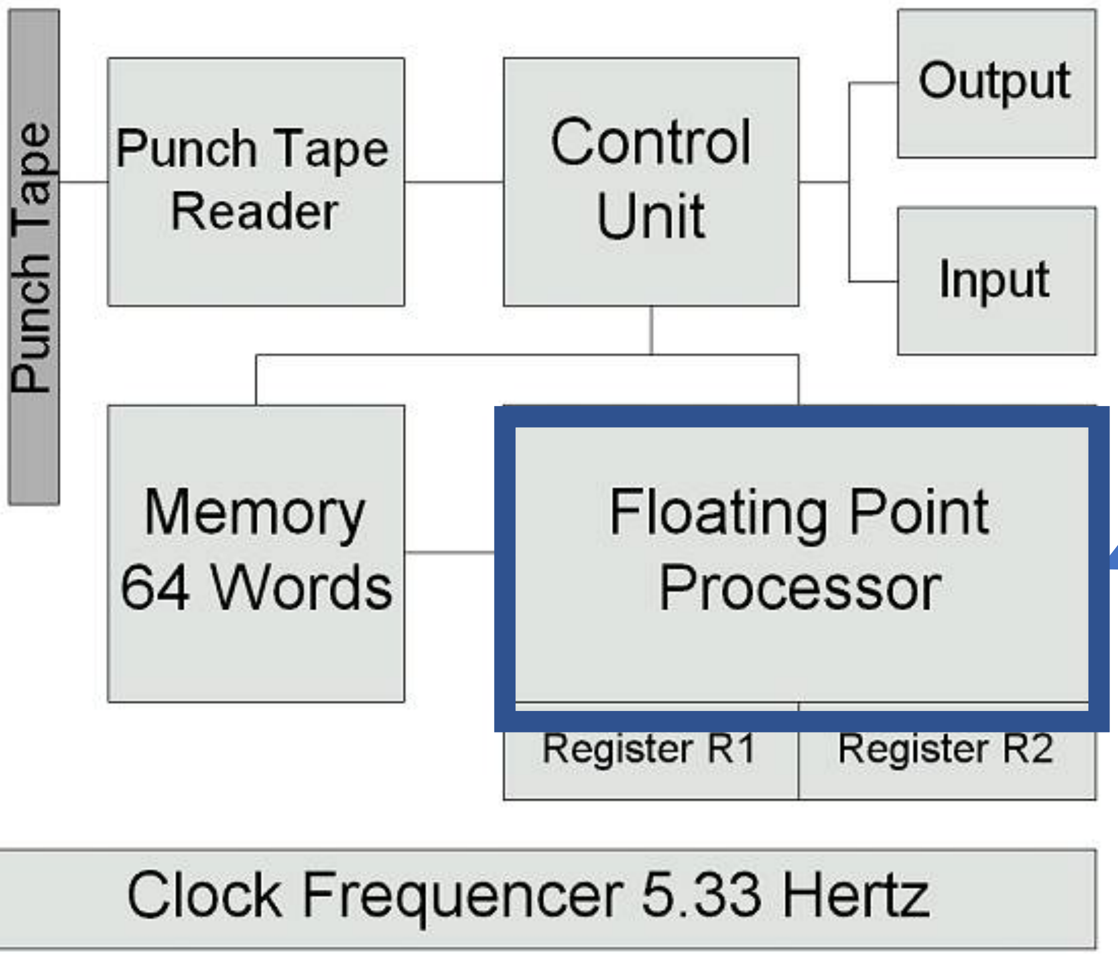


Architecture of the Z3

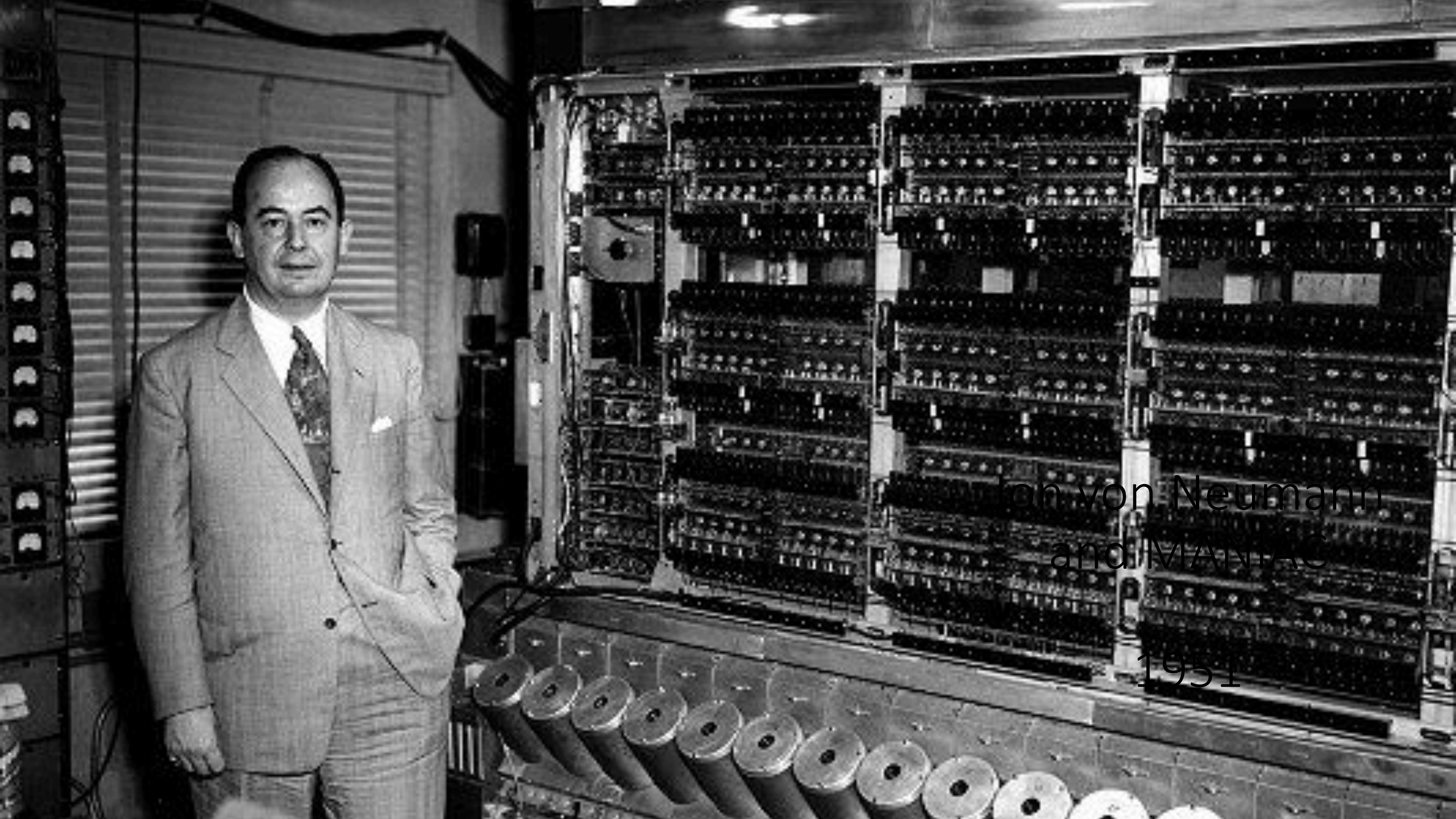


Memory is where we can read and write symbols

Architecture of the Z3



The processor says what to do for each symbol read



Jon von Neumann and MANIAC

1951

Von Neumann Architecture

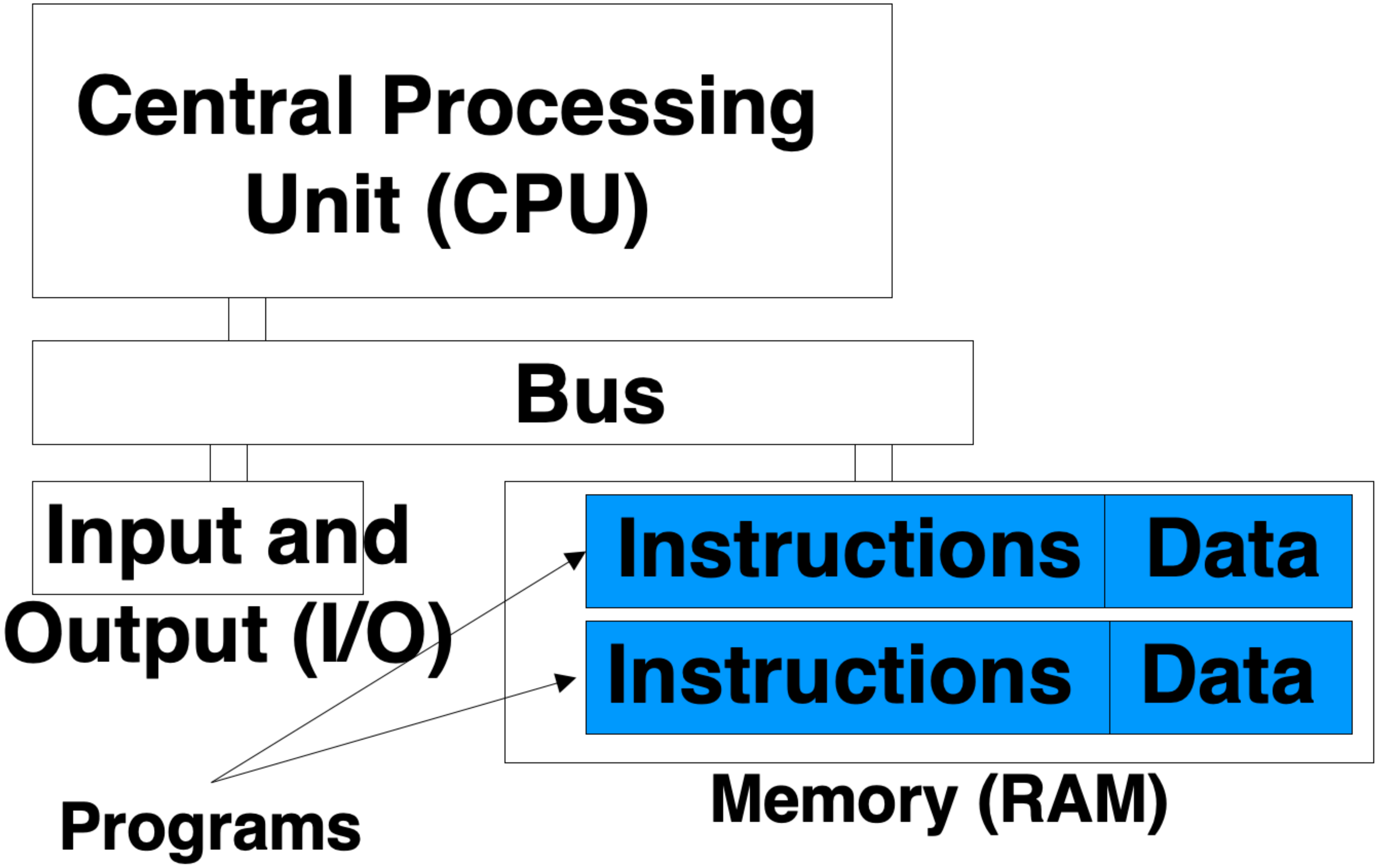
- Johann Von Neumann popularized the Stored Program computer.
- Previous computers (like the Turing’s COLLOSUS and ENIAC) had the program literally wired in.
- The Z3 had its program on punch tape.
- Stored program computers made changing the algorithm a computer was to execute trivial. (compared to rewiring it!)
- It also meant that the sequence of execution could be changed while the program was running.

Anatomy of a Computer

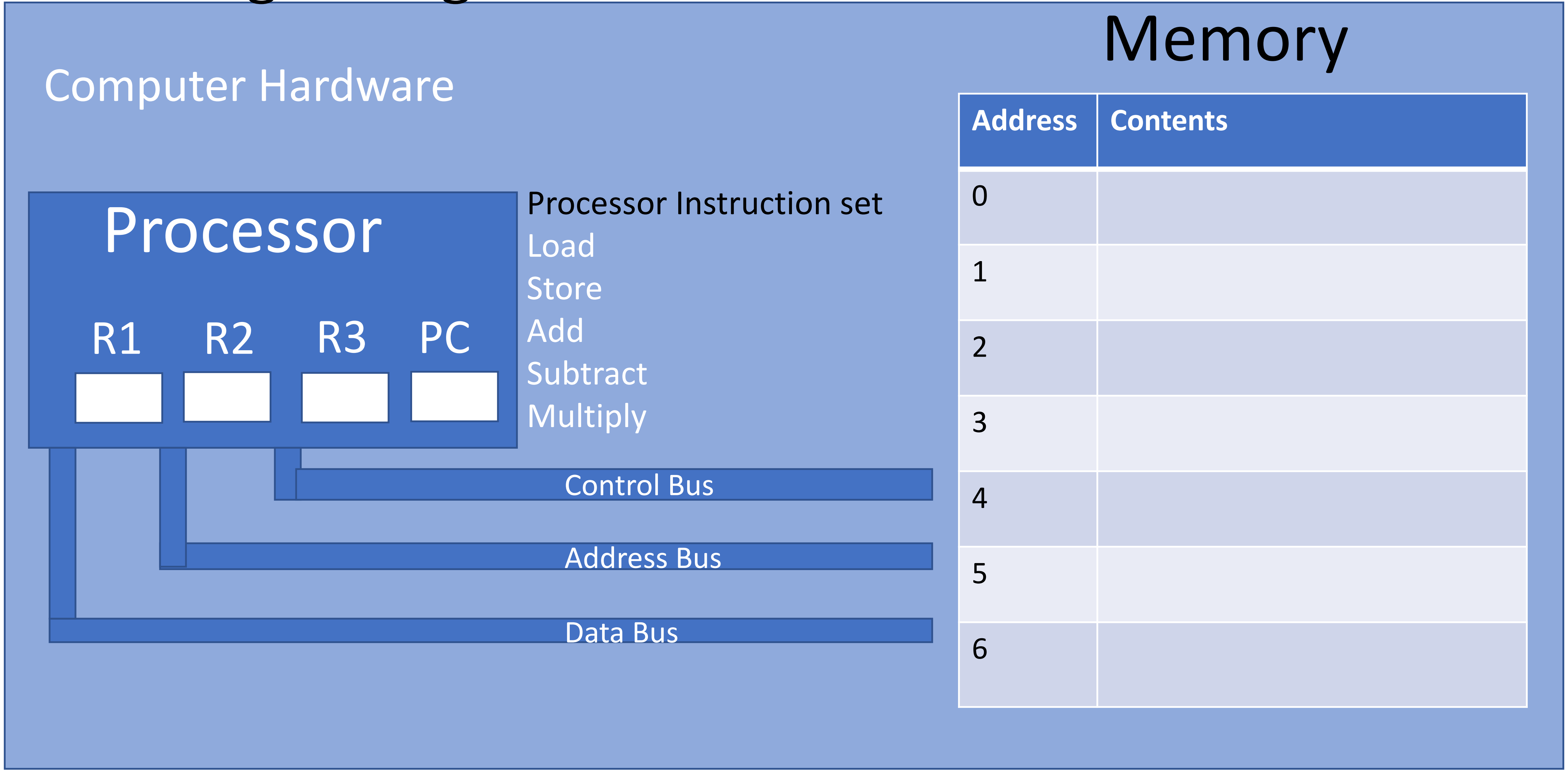
- Six logical units of modern computers
  - Input (Keyboard, mouse, etc)
  - Output (Monitor, Printer, etc)
  - Volatile Memory (RAM or main memory)
  - Long term Memory (Disk, punch tape, etc)
  - Central Processing Unit (Processor)
  - Arithmetic and Logic Unit (Processor)

(+ Bus, pipeline so all the other units can communicate)

Basic Organization of a Modern Computer



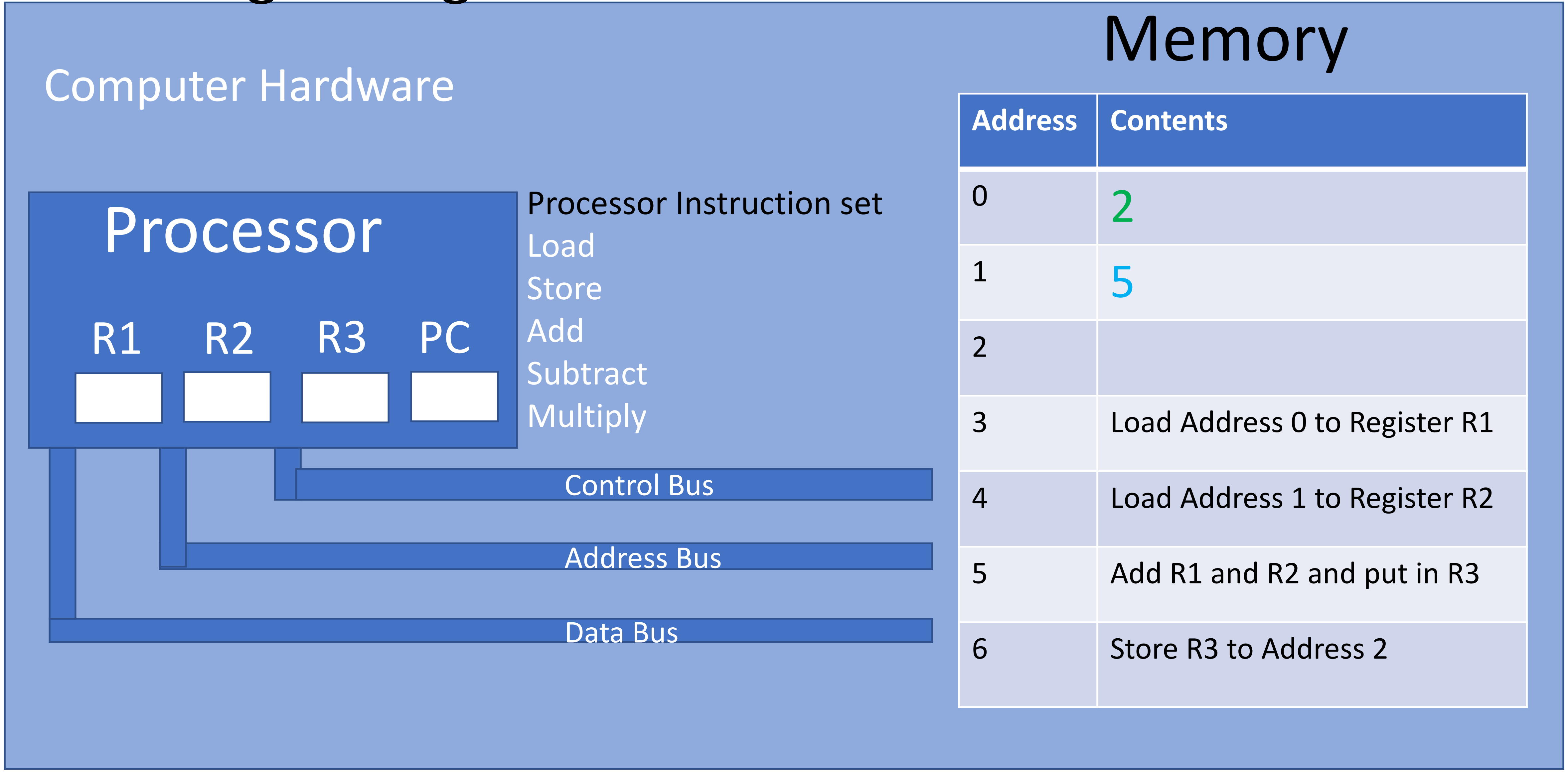
Executing a Program



Address	Contents
0	
1	
2	
3	
4	
5	
6	

Let’s add 2 and 5

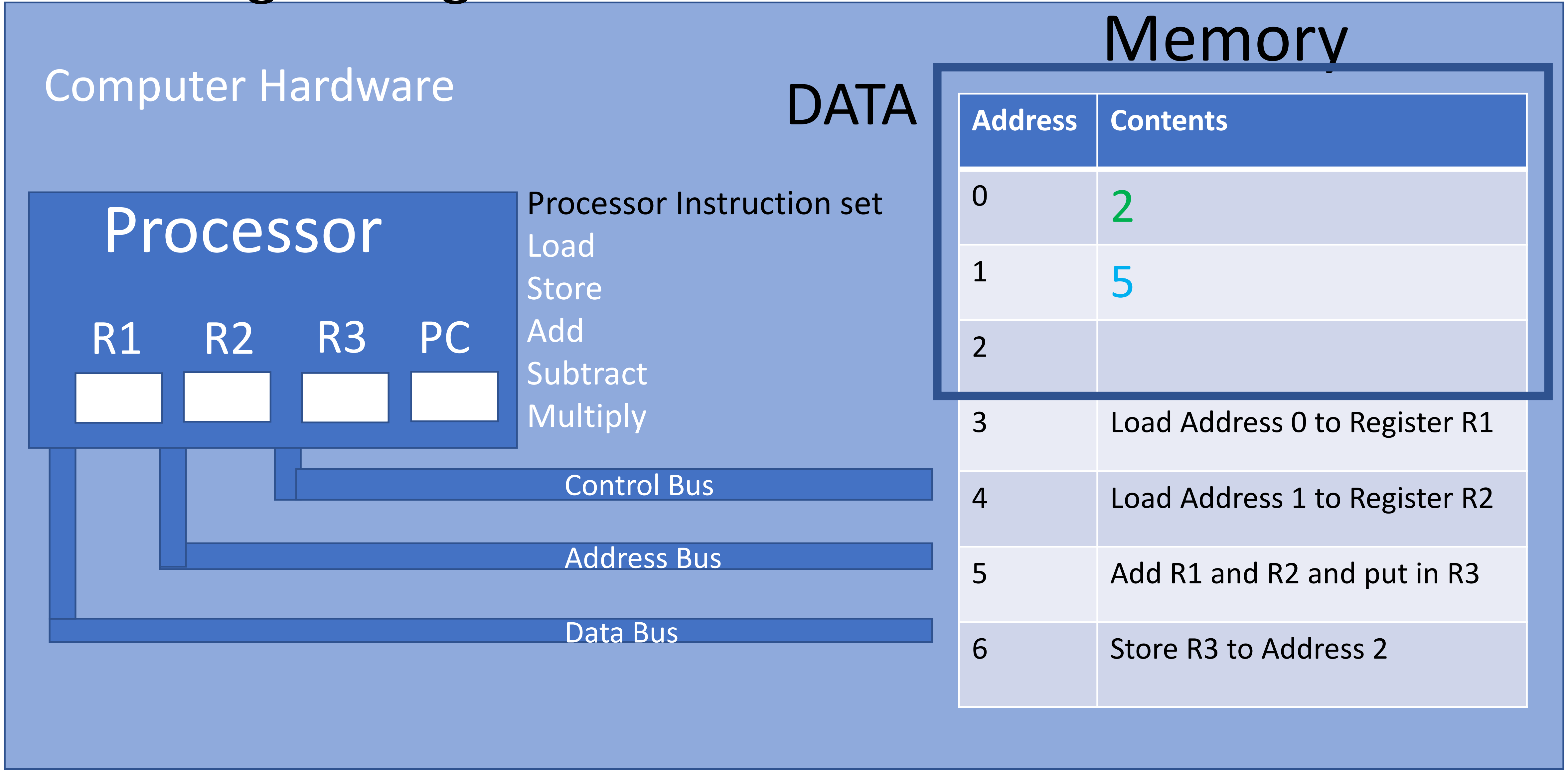
Executing a Program



Address	Contents
0	2
1	5
2	
3	Load Address 0 to Register R1
4	Load Address 1 to Register R2
5	Add R1 and R2 and put in R3
6	Store R3 to Address 2

Let’s add 2 and 5

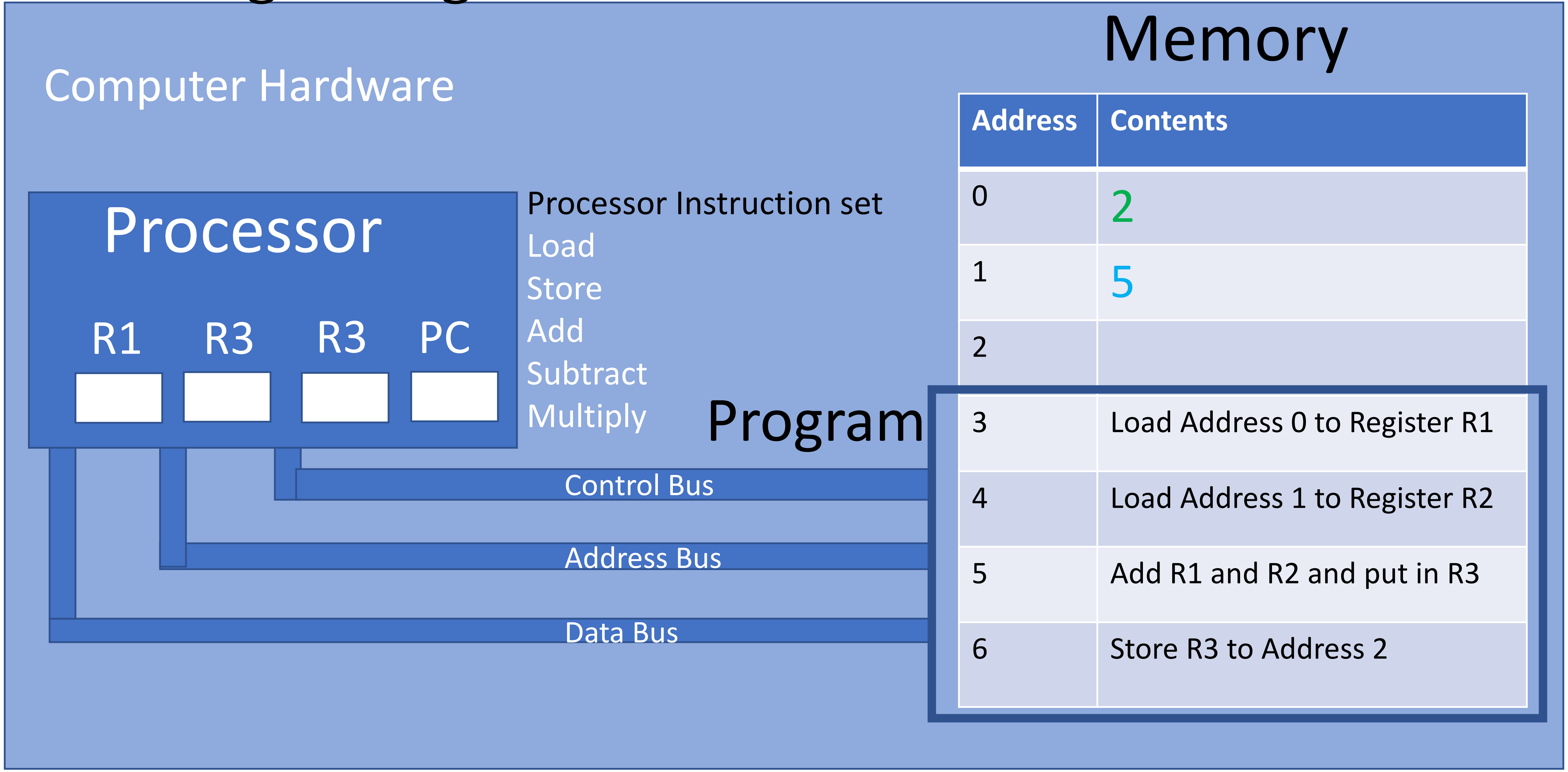
Executing a Program



Address	Contents
0	2
1	5
2	
3	Load Address 0 to Register R1
4	Load Address 1 to Register R2
5	Add R1 and R2 and put in R3
6	Store R3 to Address 2

Let’s add 2 and 5

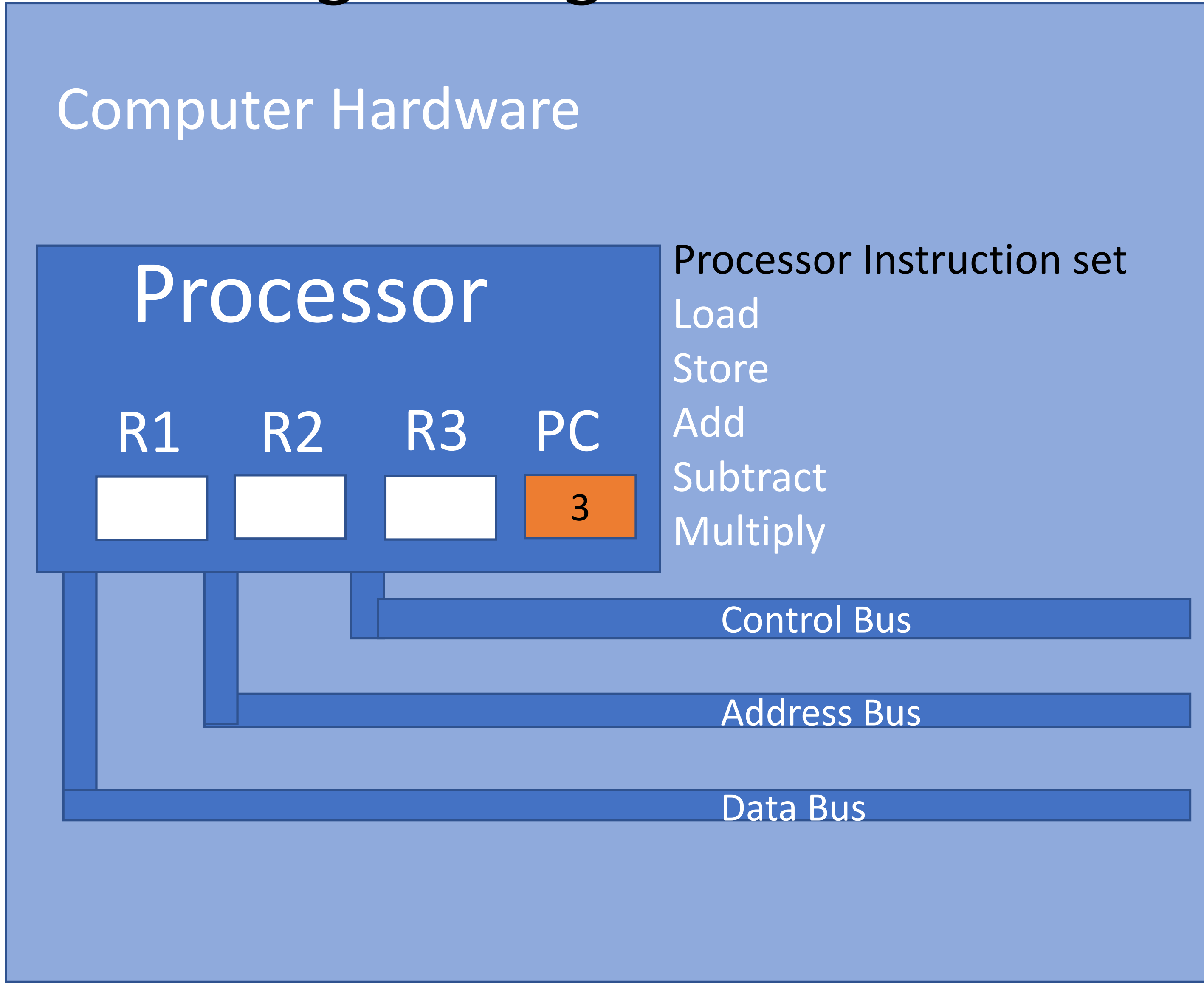
Executing a Program



Address	Contents
0	2
1	5
2	
3	Load Address 0 to Register R1
4	Load Address 1 to Register R2
5	Add R1 and R2 and put in R3
6	Store R3 to Address 2

Let’s add 2 and 5

Executing a Program

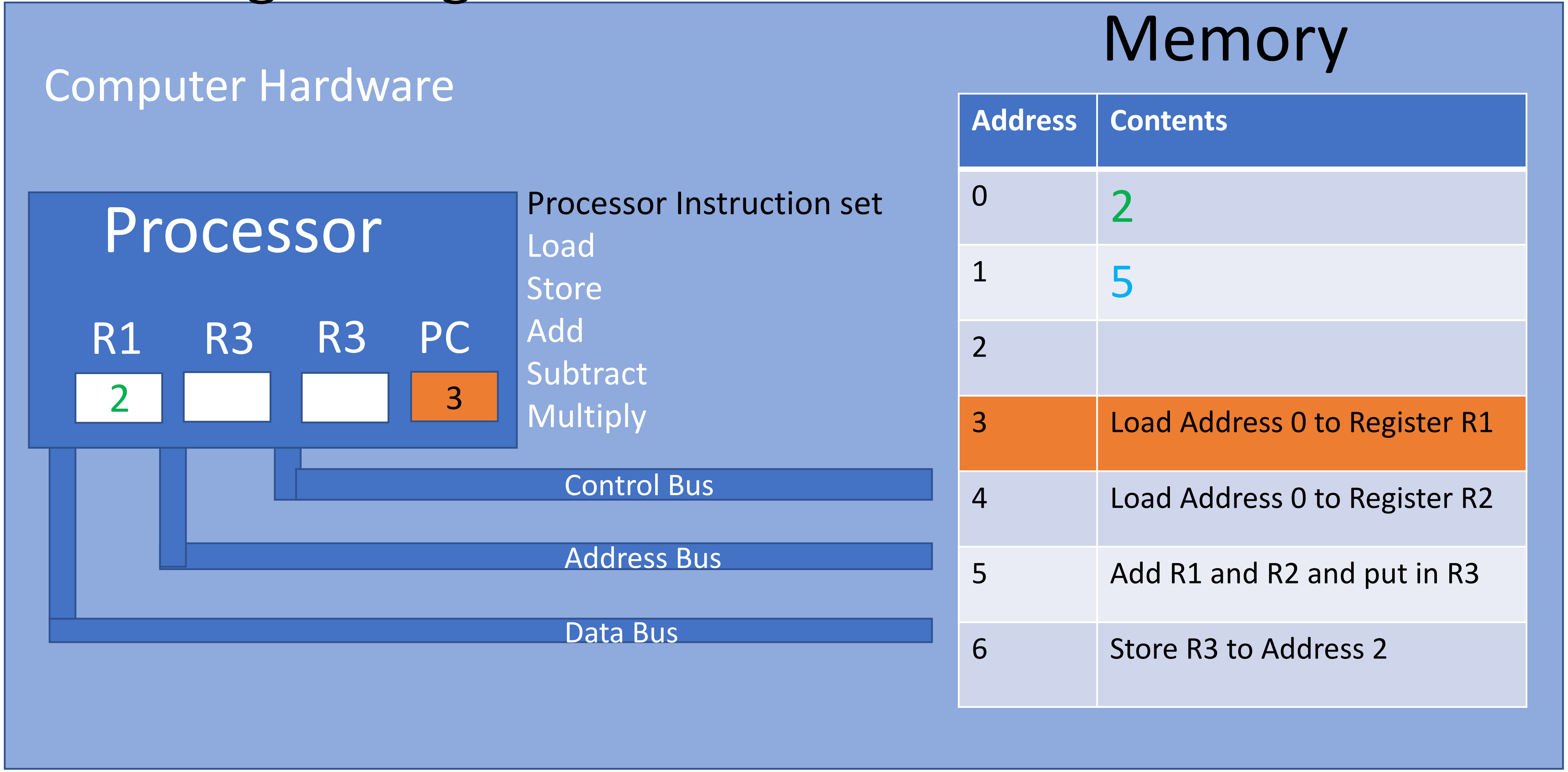


Memory

Address	Contents
0	2
1	5
2	
3	Load Address 0 to Register R1
4	Load Address 1 to Register R2
5	Add R1 and R2 and put in R3
6	Store R3 to Address 2

Let’s add 2 and 5

Executing a Program



Processor Instruction set

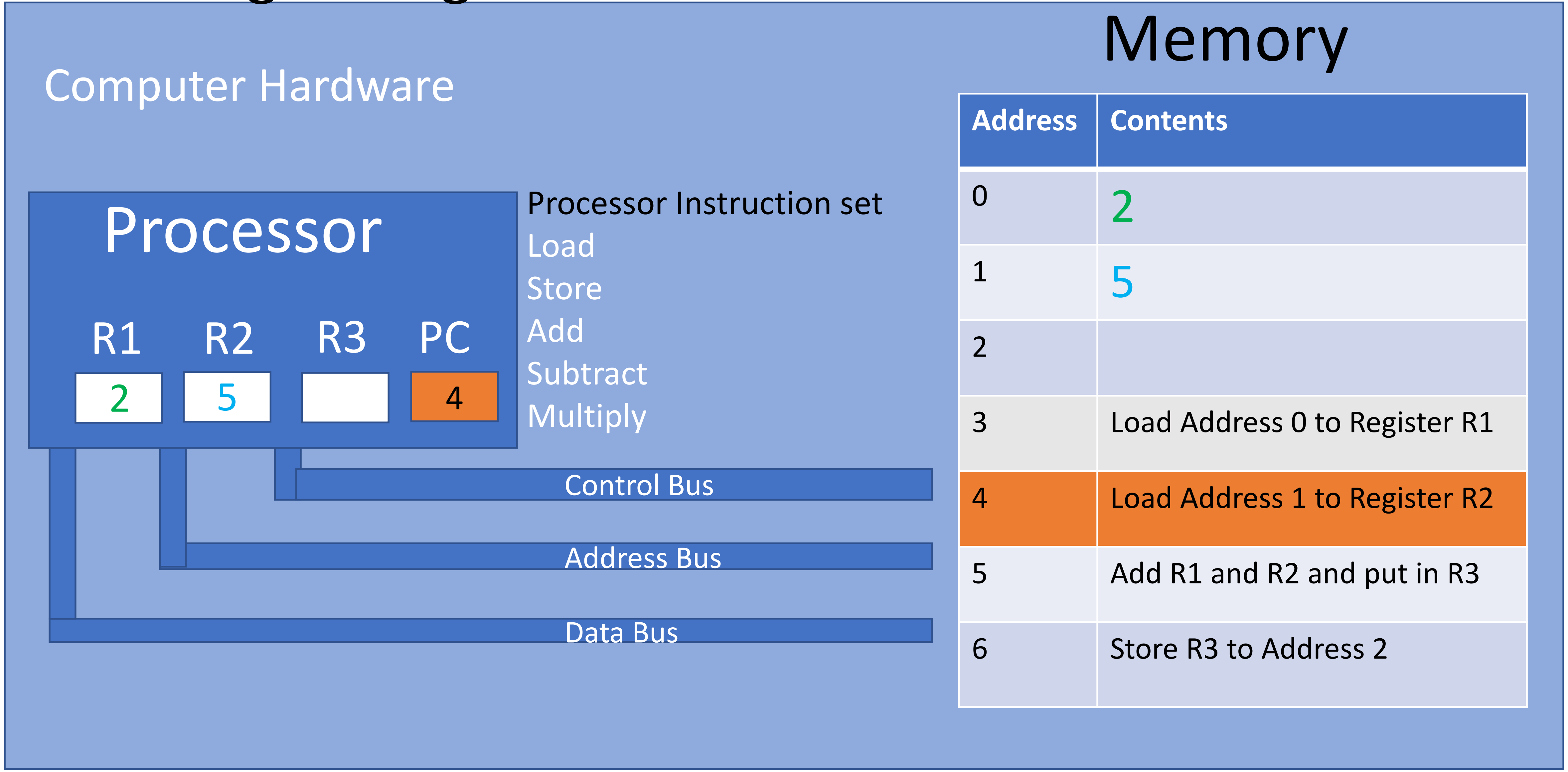
- Load
- Store
- Add
- Subtract
- Multiply

Memory

Address	Contents
0	2
1	5
2	
3	Load Address 0 to Register R1
4	Load Address 0 to Register R2
5	Add R1 and R2 and put in R3
6	Store R3 to Address 2

Let’s add 2 and 5

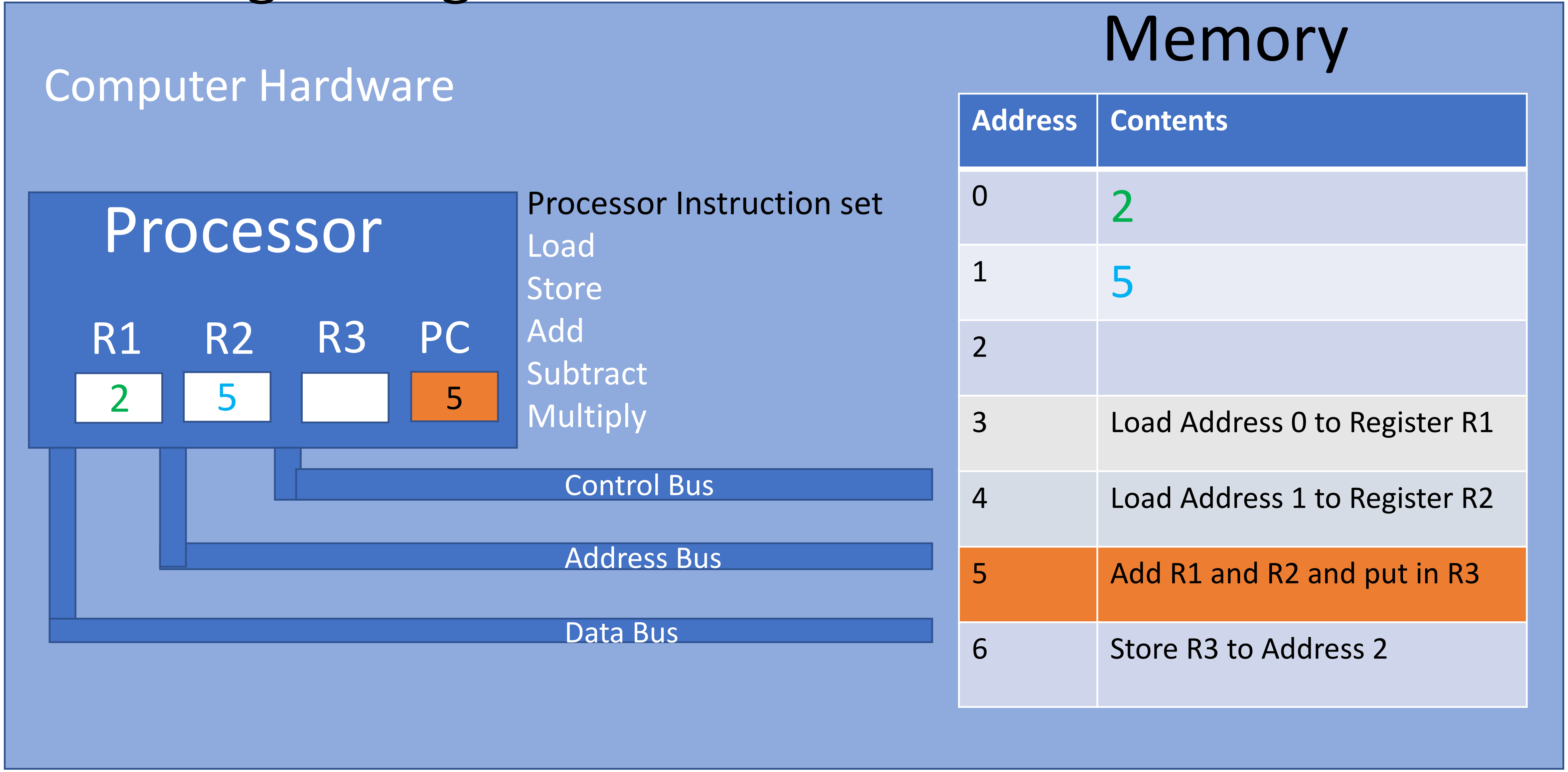
Executing a Program



Address	Contents
0	2
1	5
2	
3	Load Address 0 to Register R1
4	Load Address 1 to Register R2
5	Add R1 and R2 and put in R3
6	Store R3 to Address 2

Let’s add 2 and 5

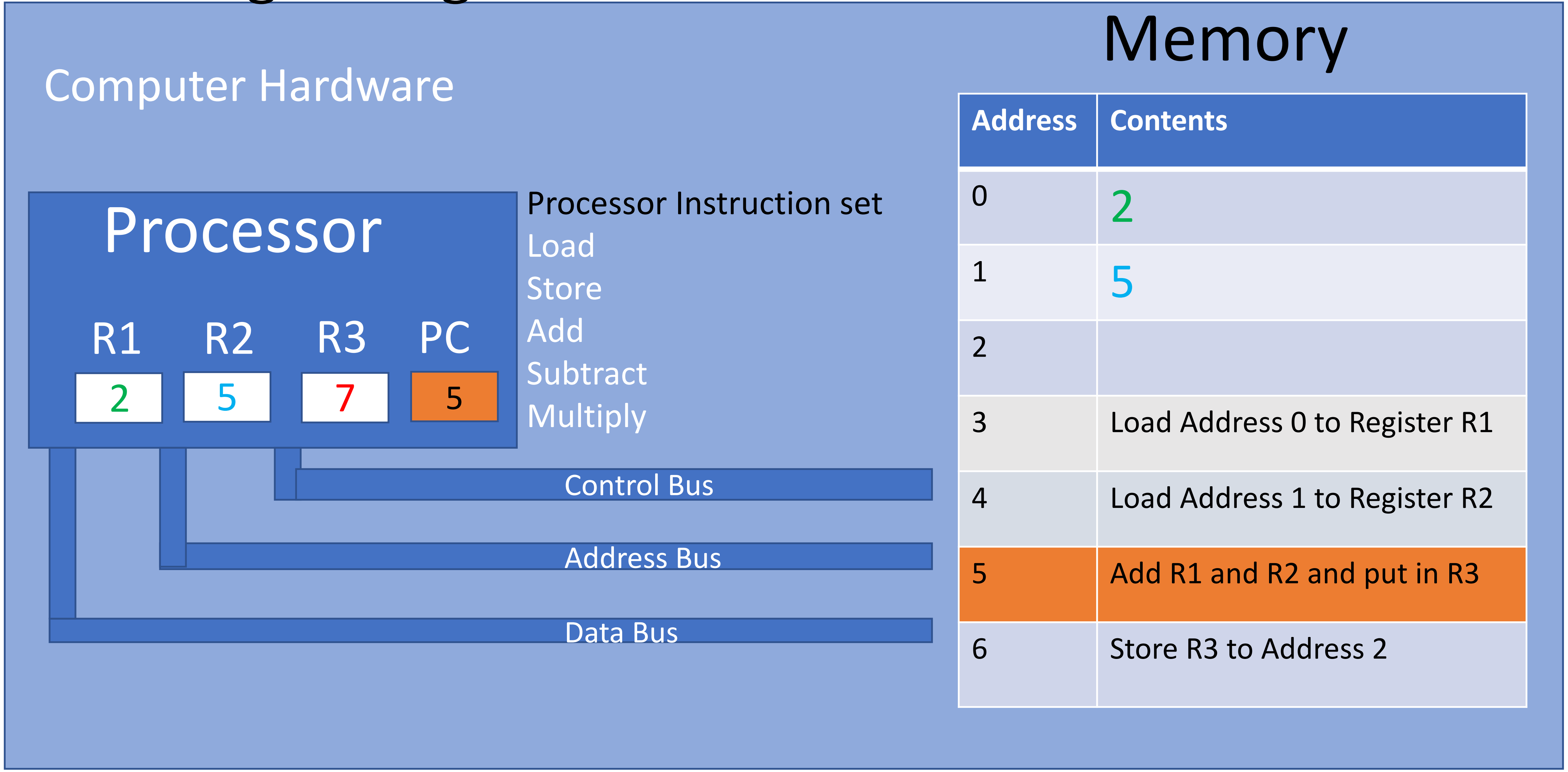
Executing a Program



Address	Contents
0	2
1	5
2	
3	Load Address 0 to Register R1
4	Load Address 1 to Register R2
5	Add R1 and R2 and put in R3
6	Store R3 to Address 2

Let’s add 2 and 5

Executing a Program



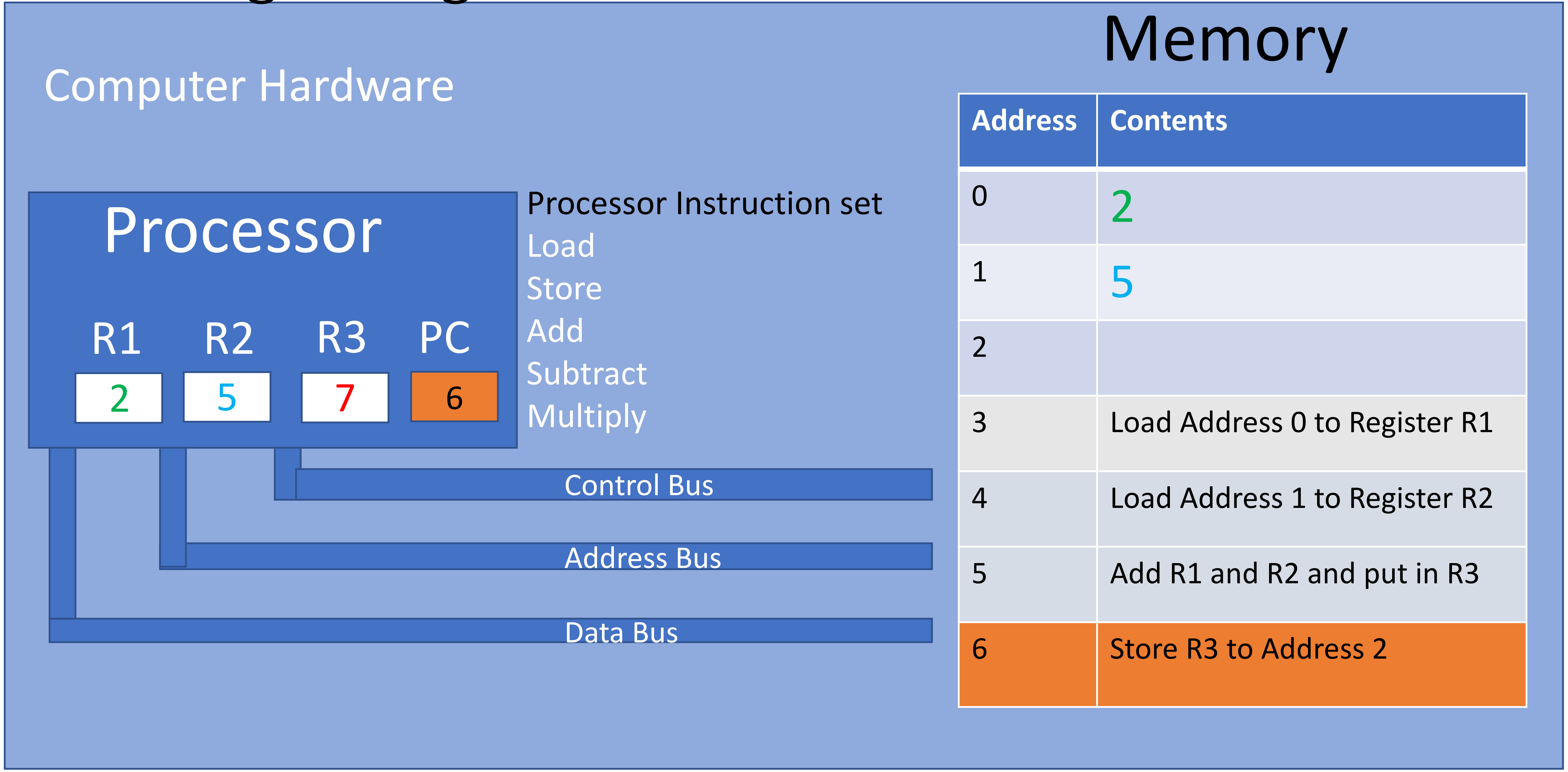
Processor Instruction set
Load
Store
Add
Subtract
Multiply

Memory

Address	Contents
0	2
1	5
2	
3	Load Address 0 to Register R1
4	Load Address 1 to Register R2
5	Add R1 and R2 and put in R3
6	Store R3 to Address 2

Let's add 2 and 5

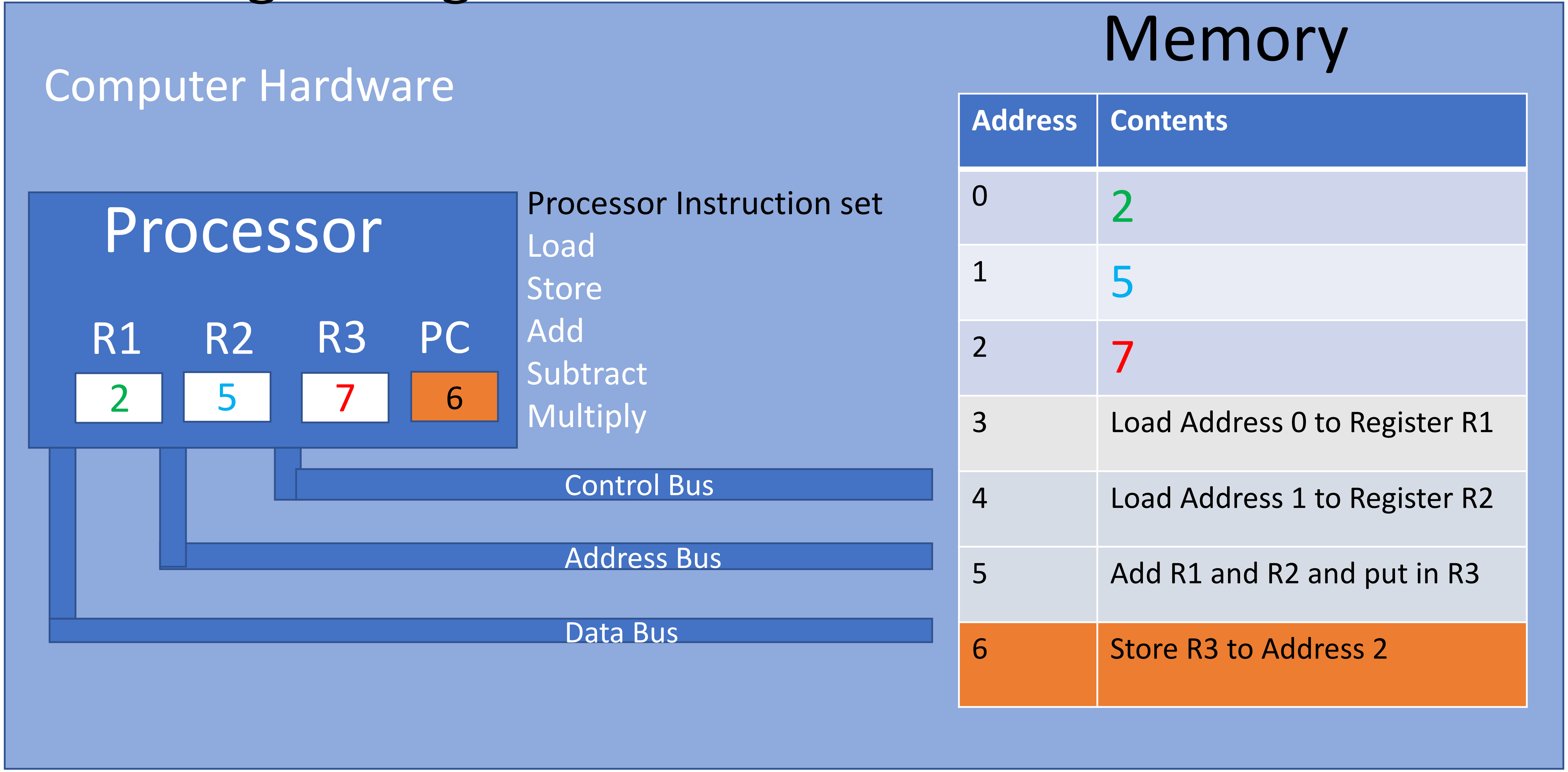
Executing a Program



Address	Contents
0	2
1	5
2	
3	Load Address 0 to Register R1
4	Load Address 1 to Register R2
5	Add R1 and R2 and put in R3
6	Store R3 to Address 2

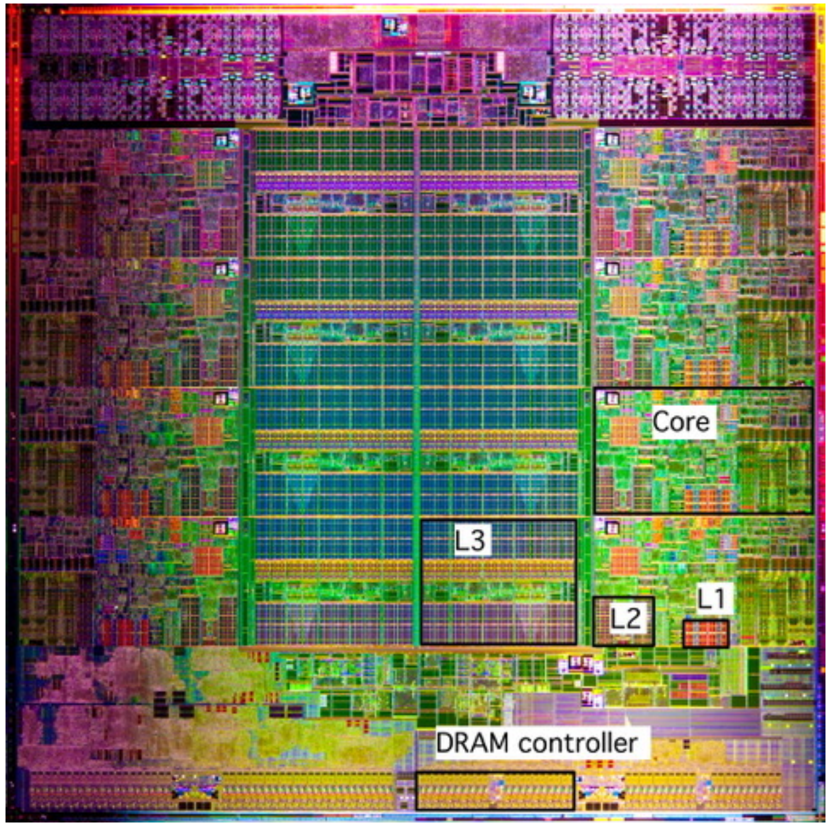
Let’s add 2 and 5

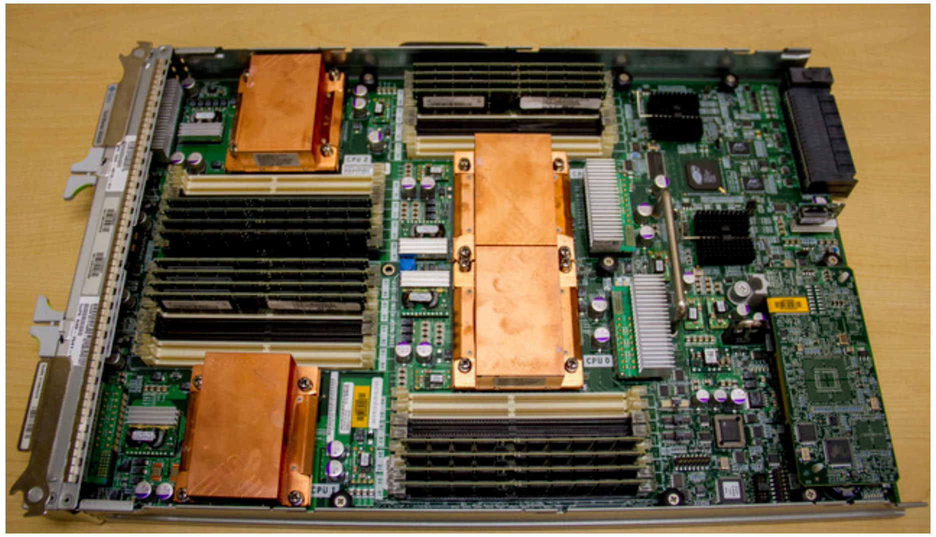
Executing a Program

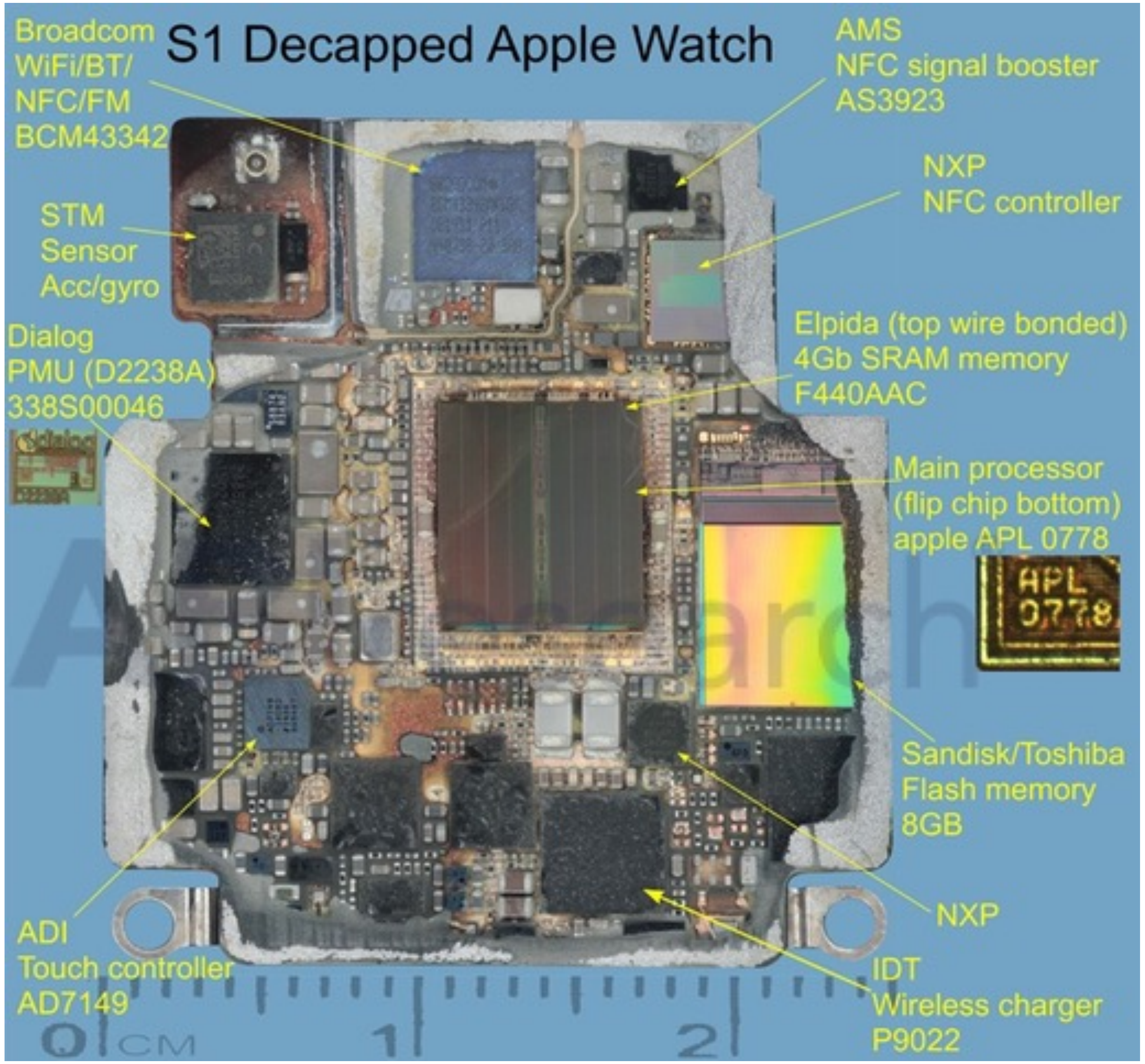


Address	Contents
0	2
1	5
2	7
3	Load Address 0 to Register R1
4	Load Address 1 to Register R2
5	Add R1 and R2 and put in R3
6	Store R3 to Address 2

Let’s add 2 and 5


Core
L3
L2
L1
DRAM controller




S1 Decapped Apple Watch
Broadcom WiFi/BT/ NFC/FM BCM43342
STM Sensor Acc/gyro
Dialog PMU (D2238A) 338S00046
AMS NFC signal booster AS3923
NXP NFC controller
Elpida (top wire bonded) 4Gb SRAM memory F440AAC
Main processor (flip chip bottom) apple APL 0778
APL 0778
Sandisk/Toshiba Flash memory 8GB
NXP
IDT Wireless charger P9022
ADI Touch controller AD7149
0 CM
1
2

Who invented the Universal Computer

Alan Turing

Ada Lovelace

John von Neuman

Who is widely considered the first programmer?

Anatomy of a Computer - Volatile Memory

- Volatile memory (RAM) consists of numerous (typically millions or billions) of binary digits (“bits”).
- A bit can hold the value 1 (the bit is set) or zero (the bit is unset).
- A collection of eight bits can have 256 different states and so can be used to represent different things. For example 256 different
- Eight bits is called a byte
- Characters (A,B, C, … a, b, c, … &, *, $) can be represented with a single byte by defining each different sequence of 1s and 0s to represent a different character.

Anatomy of a Computer - Volatile Memory

- There are other terms for various numbers of bits – for example, half a byte (4 bits) is a nibble, four bytes are called a “long” or a “dword” (double word), 1024 bytes is a kilobyte.
- The more bits a division has the more states it can represent but the more memory it uses.

Anatomy of a Computer - CPU

- The CPU has several local “registers.” Registers are memory locations just like RAM.
- The CPU is able to “fetch” values from RAM and place them into its registers.
- Data in the registers can then be operated on by the ALU and CPU.

Anatomy of a Computer

- Secondary storage is mapped out in much the same way as main memory (RAM). The media for storing the bits is non-volatile and so does not need constant power. This is where we save our programs when they are not running.
- Input devices can be as diverse as keyboards, scanners, and temperature sensors.
- Output devices are typically monitors, sound cards, and printers.
- Many devices are used for both input and output, such as network cards.

Languages – Machine Language

- Every CPU has a list of actions that it is capable of performing (the instruction set).
- These instructions must be given to the CPU in binary code for it to understand them.
- A typical instruction might be:

  00000001 01100101 10010010 01

Languages – Machine Language, cont

A typical instruction might be:

00000001 01100101 10010010 01

Where 00000001 means “load” (copy), “01100101 10010010” is the address of a word in memory, and 01 is the address of a register.

So this instruction tells the computer to load the value held at a certain address into the first register.

Languages - Assembly Language

Machine language was too difficult to work with so programmers added a layer of abstraction.

They wrote programs to translate keywords into the appropriate machine language.

Now they could write “load 77E814F1 esi” and the “assembler” would translate the code into the 1s and 0s of machine language.

Typical commands are things like load, add, jump, add1, poke, and, xor, etc.

Languages – Assembly Language

Typical snapshot of assembly language

Address	Instruction	Register or RAM Address
77E814EE	mov	esi,dword ptr [edi+8]
77E814F1	mov	dword ptr [ebp+64h],0Ah
77E814F8	add	esi,4Ah
77E814FB	jmp	77E7E91A
77E81500	push	ebx
77E81501	xor	ebx,ebx
77E81503	cmp	ecx,ebx
77E81505	push	esi
77E81506	push	edi

A jmp and a cmp allow conditional logic: in other words an if statement

Languages – High Level Languages

- A lot of sophisticated software was written using machine and assembly language but it was still too difficult to understand.
- High level languages have added a further level of abstraction such that a single command in Fortran, Java, or C++ might be translated into hundreds or thousands of assembly language instructions.

Languages – High Level Languages

- High level languages also allow the programmer to define new commands in terms of old ones.
- This means that most high level languages are unlimited in expressive power and in their potential to abstract away complexity.
- Once you have one high level language it become easy to use it to write other high level languages.

Languages - High Level Languages

- The first high level language was created by John Backus at IBM, in 1954.
- It was originally called SpeedCoding but later the name was changed to the Formula Translation Language or Fortran.
- In 1958 John McCarthy developed the List Processing Language or Lisp. Algol, Fortran III, Flow-Matic which became Cobol all came out the same year.
- There are now well over 200 major high level programming languages in 26 different groups.

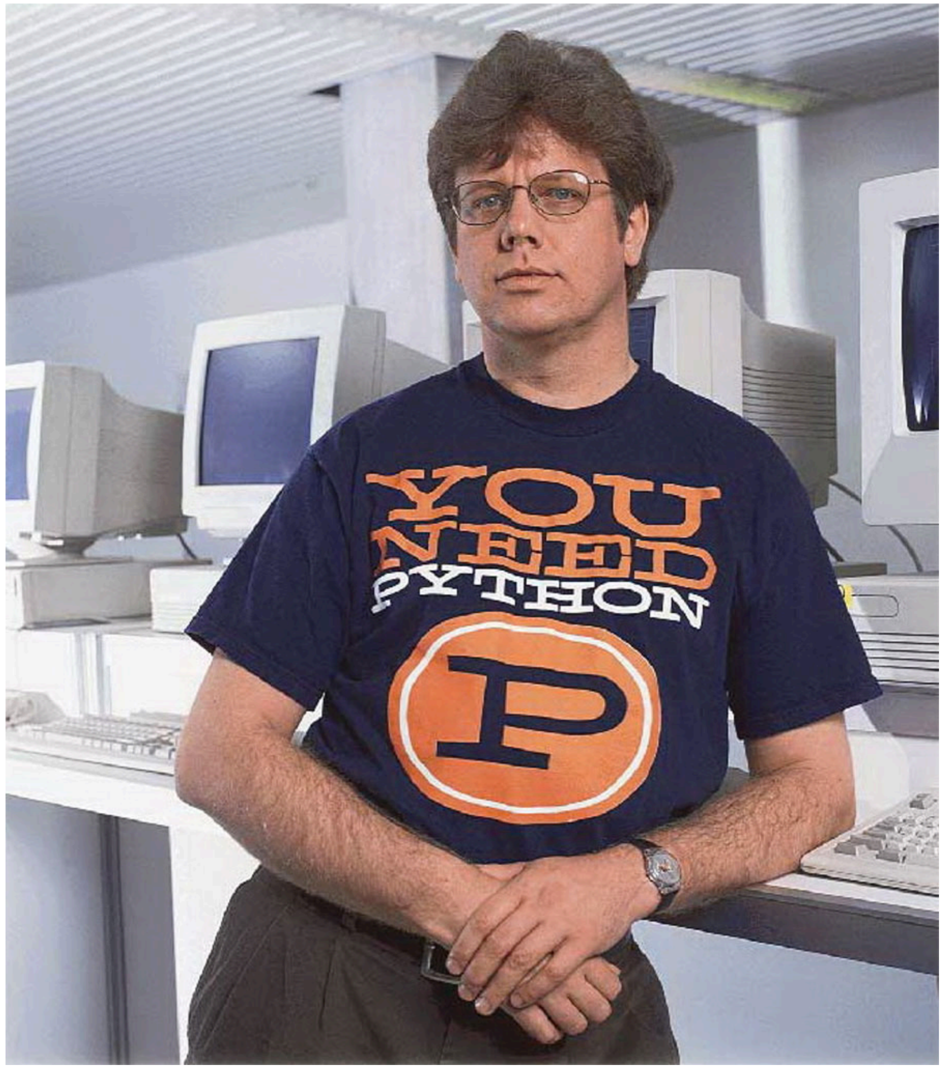
Languages - High Level Languages

Python

Invented by Guido van Rossum

(Dutch Programmer) in the 90s.

Is is relatively easy to use but powerful.



IBM Quantum Computer

Does not store data as bits that are 1s or 0s

The data is instead a probability of being between 0 and 1.

The state of the whole system collapses to the solution

For more info see:

https://plus.maths.org/content/how-does-quantum-commuting-work

Next time: Our first program

```
(base) polychrome:~ matthew$ python -V
Python 3.7.3
(base) polychrome:~ matthew$ python
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
```