From last lecture:

- Computers work using internal switches (transistors) that are either on or off.
- Transistors are used to make logic gates: AND, OR, NOT, NAND, NOR, XOR
- The Arithmetic and Logical Unit in the CPU uses gates in order to perform operations, like addition.
- A Half Adder and Full Adders are used in the ALU to add numbers

From last lecture (continued)

- Since gates work with on/off switches, then we can use off = 0 and on = 1, and use logic for gates assuming that 0=False and 1=True.
- All data stored in the memory of a computer is in binary.
- The binary system uses two symbols (0 and 1) to represent numbers.

From last lecture (continued)

- Binary numbers are positional since the actual value of a digit depends on its position in the number:

1	**0**	**1**	**0**	**1**	**1**	**← Binary Number**
5	**4**	**3**	**2**	**1**	**0**	**← Position**
32	**0**	**8**	**0**	**2**	**1**	**← Value**

32+8+2+1 = 43 ← Decimal equivalent

Notice that those digits in positions 2 and 4 have a value of 0 because the original binary number has 0 in those places.

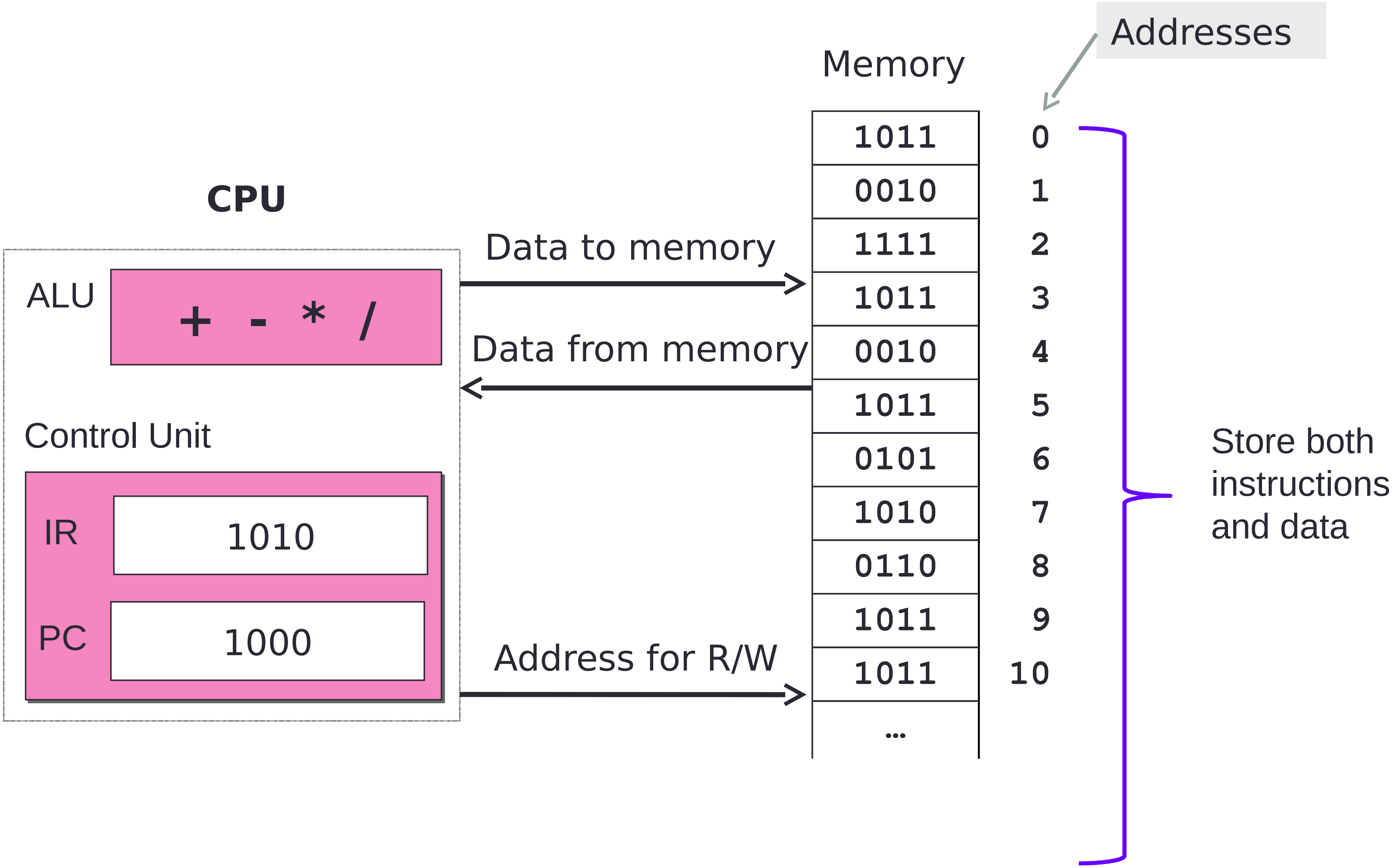
How Many Binary Patterns from N Bits

A single binary digit is called a bit, and is the smallest possible unit of information.
A bit can be either 0 or 1

Number of Bits	Number of Patterns	Number of Patterns as Power of Two
1	2	2^1
2	4	2^2
3	8	2^3
4	16	2^4
…	…	…
10	1024	2^{10}

Number of possible patterns of N bits = 2^N

Von Neumann Model: closer look



CPU

ALU

+ - * /

Control Unit

IR 1010

PC 1000

Data to memory

Data from memory

Address for R/W

Memory

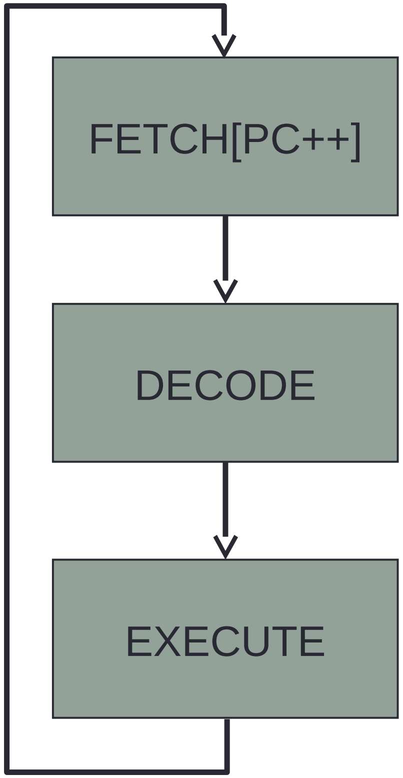
Addresses

Memory	Addresses
1011	0
0010	1
1111	2
1011	3
0010	4
1011	5
0101	6
1010	7
0110	8
1011	9
1011	10
…	

Store both instructions and data

CPU Fetch-and-Execute Cycle

- Programs
  - Written in a high level language
  - Translated into machine language that can be executed by the CPU
- CPU executing a program
  - Program is in main memory



Arithmetic: +, -, *, /
Logic: bre, jmp

Memory:

Memory is just a numbered list of binary numbers:

ADDRESS	CONTENTS
0	01001100
1	11001100
2	10101010
3	11001001
4	10110101
5	01010101
6	11111111
7	01010011
8	11001100
9	11001010
...	

One byte = 8 bits

How much memory do we have?

- 2^{10} bytes = 1024 bytes = 1 Kilobyte
- 2^{20} bytes = 2^{10} x 2^{10} = 1024 Kilobytes (1 Megabytes)
- 2^{30} bytes = 2^{10} x 2^{20} = 1024 Megabytes (1 Gigabytes)
- 2^{40} bytes = 2^{10} x 2^{30} = 1024 Gigabytes (1 Terabytes)
- 2^{50} bytes = 2^{10} x 2^{40} = 1024 Terabytes (1 Petabytes)

What can we store in memory?

Basic data:

- Machine language instructions
- Numbers (integers, floating point, unsigned integers, etc.)
- Letters (text)

To represent letters the ASCII (American Standard Code for Information Interchange) code was used for a long time:

binary	decimal	symbol
0100 0001	65	A
0100 0001	66	B
0100 0001	67	C
0100 0001	68	D
0100 0001	69	E
0100 0001	70	F
0100 0001	71	G
0100 0001	72	H
0100 0001	73	I
0100 0001	74	J
0100 0001	75	K
0100 0001	76	L
...	...	...

Newer formats like Unicode and UTF-8 are backwards compatible with ASCII and support many more characters.

Building Blocks of Programs

- Data: Variables and Types
  - a *variable* is just a memory location
  - a variable has a *type* to indicate what sort of data it can hold
- Instructions: Control Structures and Subroutines
  - *control structures* can change the flow of control
    - branches and loops
  - *subroutines* are a group of instructions that together perform some task

Primitive Data Types

A variable in Java can hold only one type of data

Data Type	Bytes	Range	Example
byte	1	-128 to 127	byte a = 130;
short	2	-32768 to 32767	short a = 1230;
int	4	-2147483648 to 2147483647	int a = 331;
long	8	-9223372036854775808 to 9223372036854775807	long a = 23;
float	4	10^{38}	float a = 23.1;
double	8	10^{308}	float a = 56.2;
char	2		char a = ‘A’;
boolean	1	true or false	boolean a = true;

Program: Fahrenheit To Celsius Conversion

1. Analysis
   - Input: temperature in Fahrenheit
   - Output: temperature in Celsius
   - Error conditions: input less than -459.67 (absolute zero)

2. Algorithm Construction

```
print "Please, enter the temperature in Fahrenheit"
tempF ← read number
tempC ← (tempF – 32) / 9 * 5
print tempC
```

When execution gets here, it waits for user to enter data, then reads the value entered, and stores it into a memory location called tempF

The right hand side is computed, using value retrieved from the memory location tempF, the result is stored in a memory location called tempC

The value in the memory location tempC is retrieved and printed.
Retrieval does NOT wipe out the values – they are still there, and can be reused as many times as needed.

Program: Fahrenheit To Celsius Conversion with Error Checking

1. Analysis: same as before
2. Algorithm Construction

```
print “Please, enter the temperature in Fahrenheit”
tempF ← read number
if tempF < -459.67
    print “Not a valid temperature”
    halt
tempC ← (tempF – 32) / 9 * 5
print tempC
```

Indentation expresses conditionality

4. Testing

Input	Expected Output	Output
32	0	0
100	37.78	37.78
-600	error	error

When making a code change, run all tests again.

Same Problem, Different Solutions

```
print “Please, enter the temperature in Fahrenheit”
tempF ← read number
if tempF < -459.67
    print “Not a valid temperature”
    halt
tempC ← (tempF – 32) / 9 * 5
print tempC
```

1-way decision

See F2C.java

```
print “Please, enter the temperature in Fahrenheit”
tempF ← read number
if tempF < -459.67
    print “Not a valid temperature”
else
    tempC ← (tempF – 32) / 9 * 5
    print tempC
```

2-way decision

See F2C_v2.java