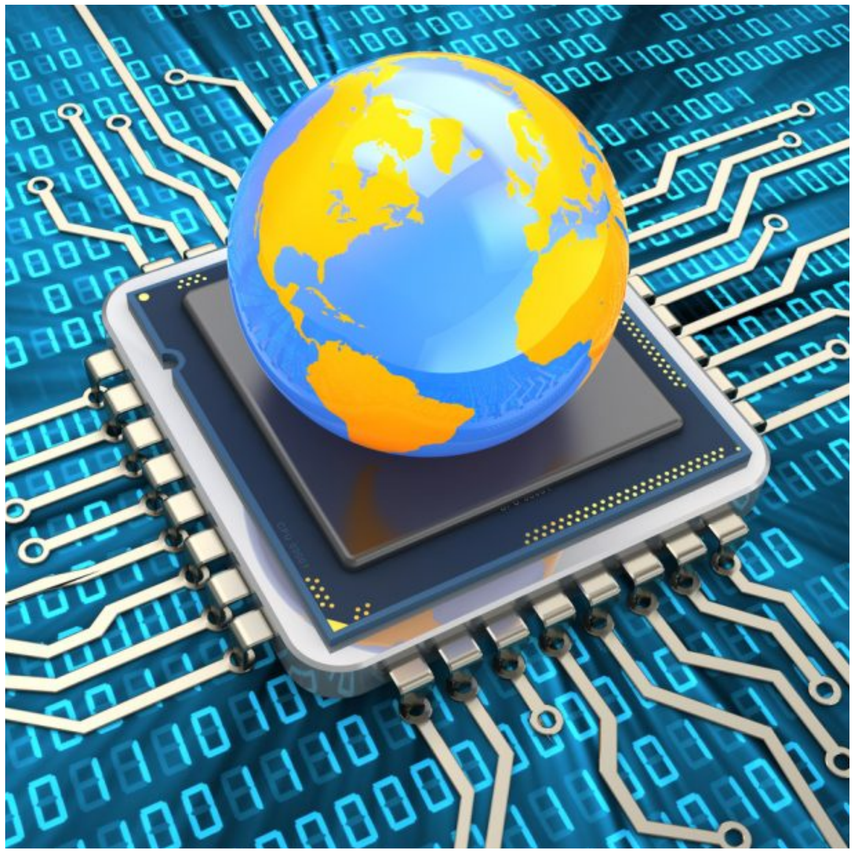
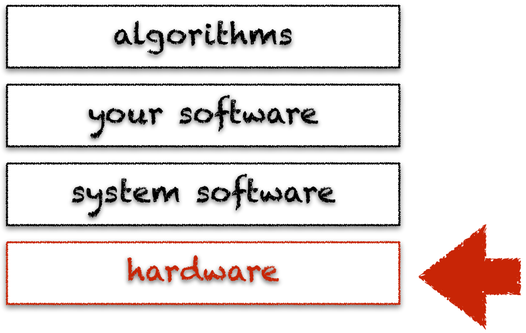
computer architecture

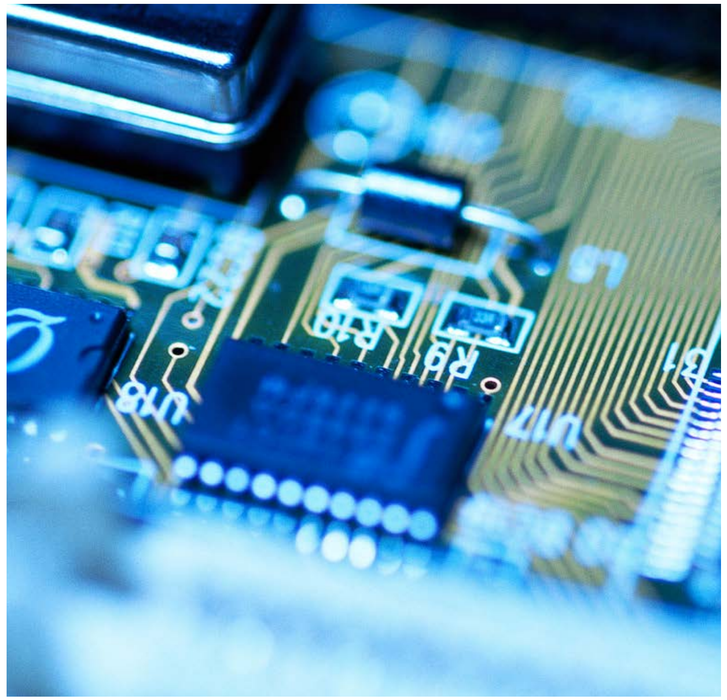


learning objectives



- understand the basics of how a computer works
- understand the basics of binary computation
- learn the basics of the Von Neumann architecture

what's a computer?



hardware

+

software

a dual origin

mathematics

Associative Rules:	$(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$	$(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$
Distributive Rules:	$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$	$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$
Idempotent Rules:	$p \wedge p \Leftrightarrow p$	$p \vee p \Leftrightarrow p$
Double Negation:	$\neg\neg p \Leftrightarrow p$	
DeMorgan's Rules:	$\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$	$\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$
Commutative Rules:	$p \wedge q \Leftrightarrow q \wedge p$	$p \vee q \Leftrightarrow q \vee p$
Absorption Rules:	$p \vee (p \wedge q) \Leftrightarrow p$	$p \wedge (p \vee q) \Leftrightarrow p$
Bound Rules:	$p \wedge F \Leftrightarrow F \quad p \wedge T \Leftrightarrow p$	$p \vee T \Leftrightarrow T \quad p \vee F \Leftrightarrow p$
Negation Rules:	$p \wedge (\neg p) \Leftrightarrow F$	$p \vee (\neg p) \Leftrightarrow T$

p	$\neg p$
F	T
T	F

p	q	$p \wedge q$
F	F	F
F	T	F
T	F	F
T	T	T

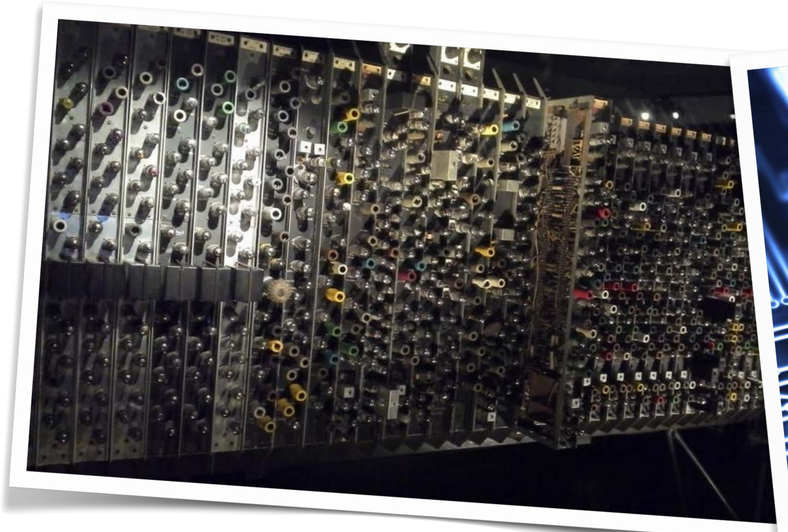
p	q	$p \vee q$
F	F	F
F	T	T
T	F	T
T	T	T

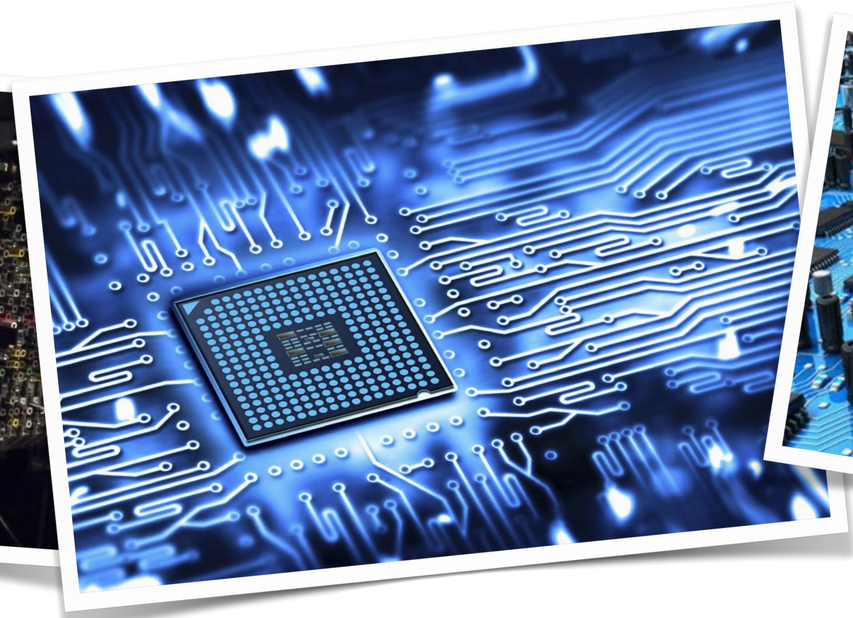
$$\left\{\frac{2^i - 1}{2^i}\right\}_{i=1}^{\infty} = \frac{1}{2}, \frac{3}{4}, \frac{7}{8}, \frac{15}{16}, \ldots,$$

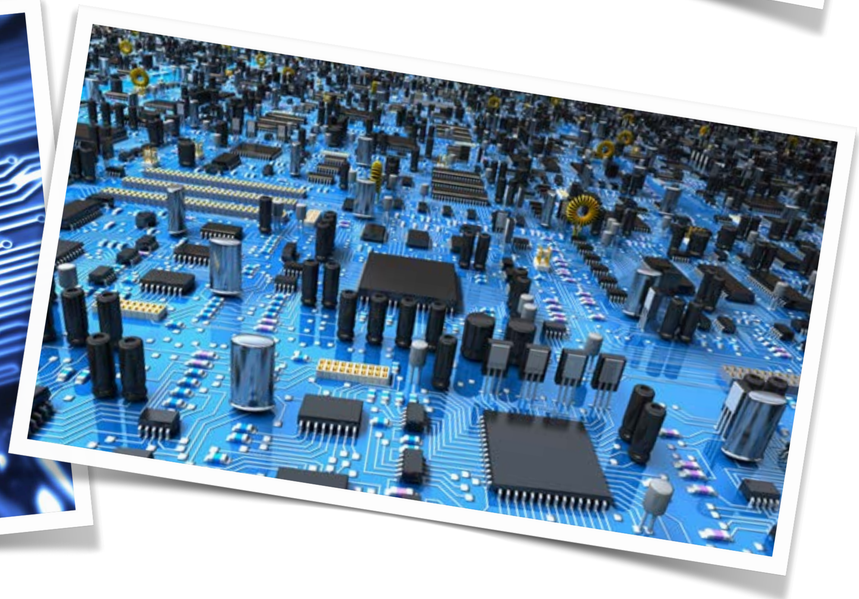
$$\left\{\frac{n+1}{n}\right\}_{i=1}^{\infty} = \frac{2}{1}, \frac{3}{2}, \frac{4}{3}, \frac{5}{4}, \ldots$$

$$\frac{a_{n+1}}{a_n} = \frac{(n+1)!}{(n+1)^{n+1}} \frac{n^n}{n!} = \frac{(n+1)!}{n!} \frac{n^n}{(n+1)^{n+1}} = \frac{n+1}{n+1} \left(\frac{n}{n+1}\right)^n = \left(\frac{n}{n+1}\right)^n < 1.$$

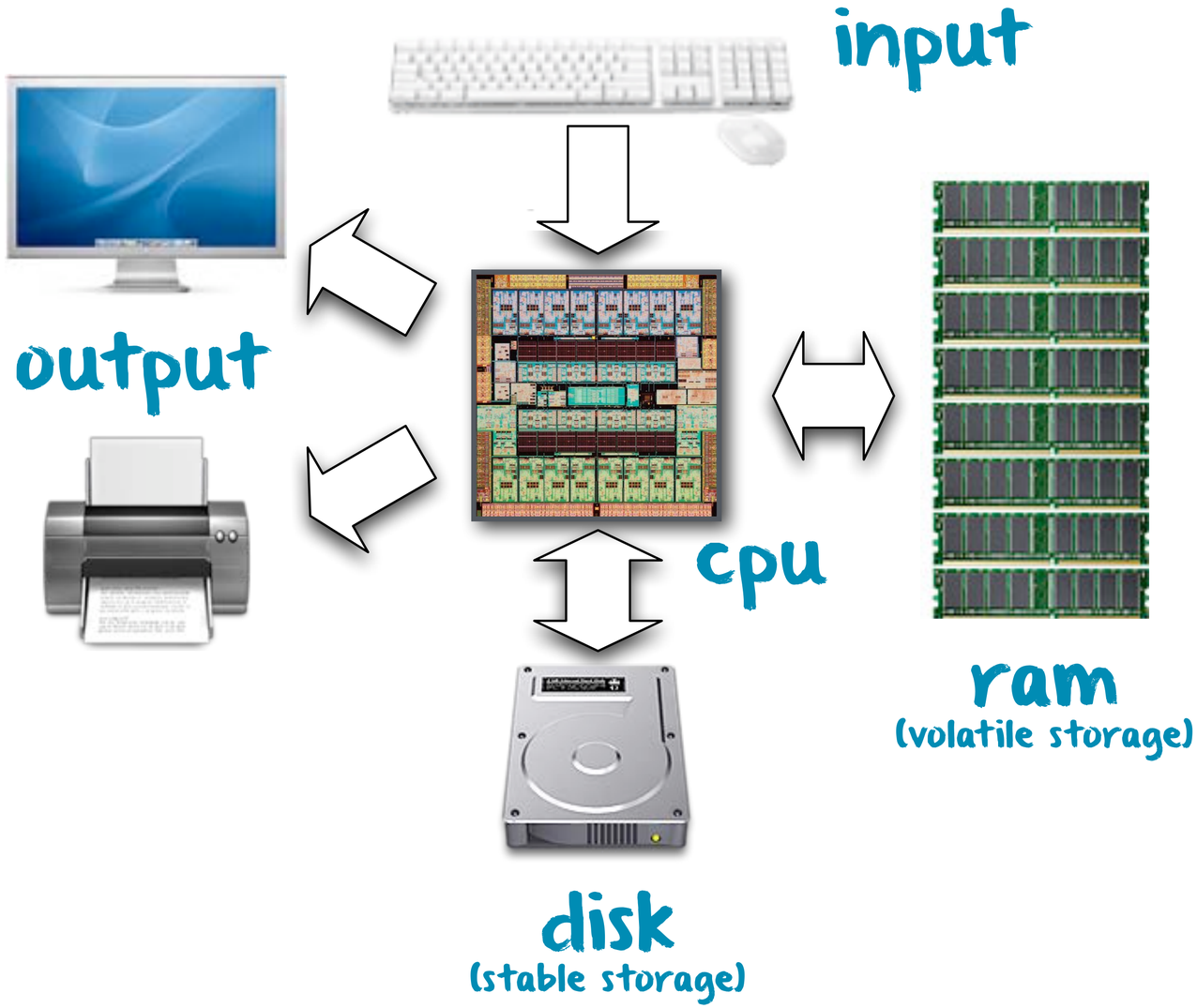
electronics





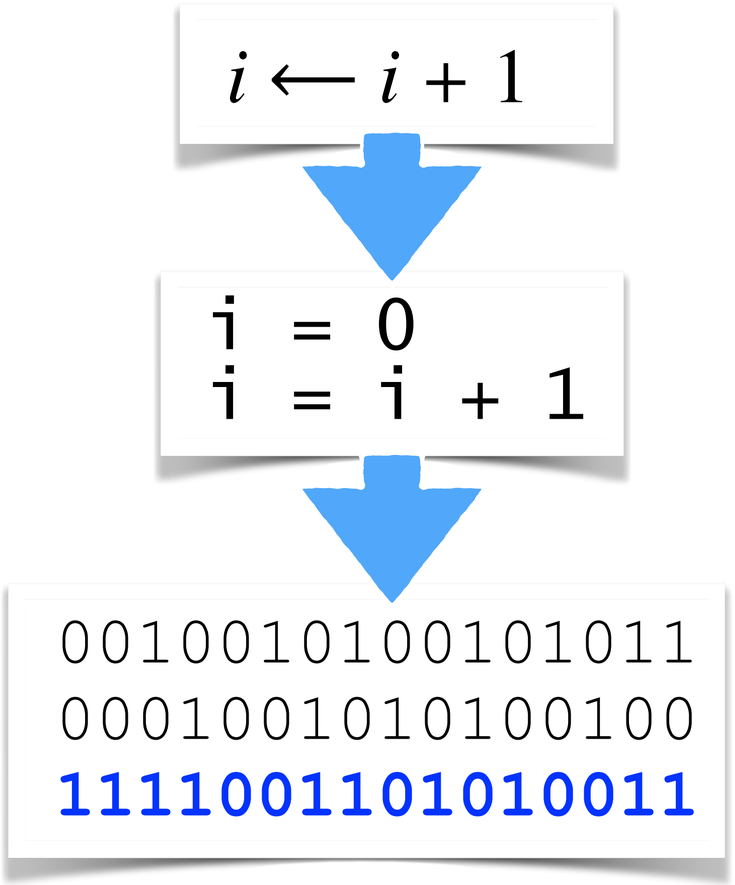


what's hardware?



cpu = central processing unit

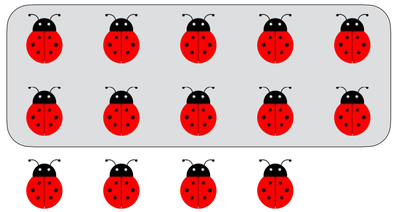
what's software?



we will come back to these transformations in the next module

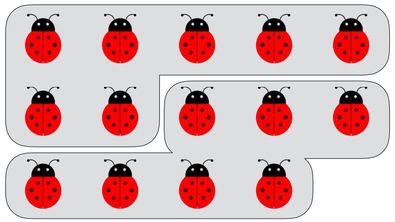
binary computation

base 10



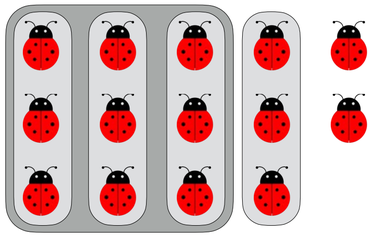
10^2	10^1	10^0
	1	4

base 7



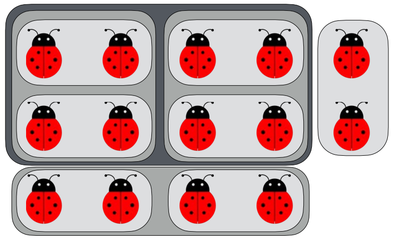
7^2	7^1	7^0
	2	0

base 3



3^2	3^1	3^0
1	1	2

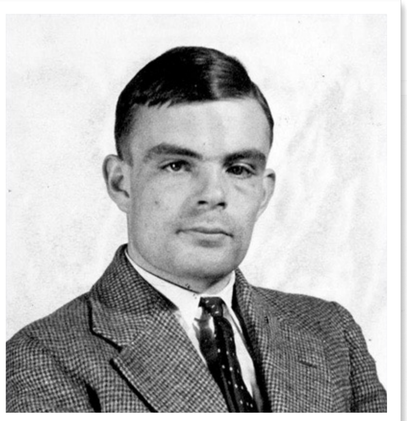
base 2

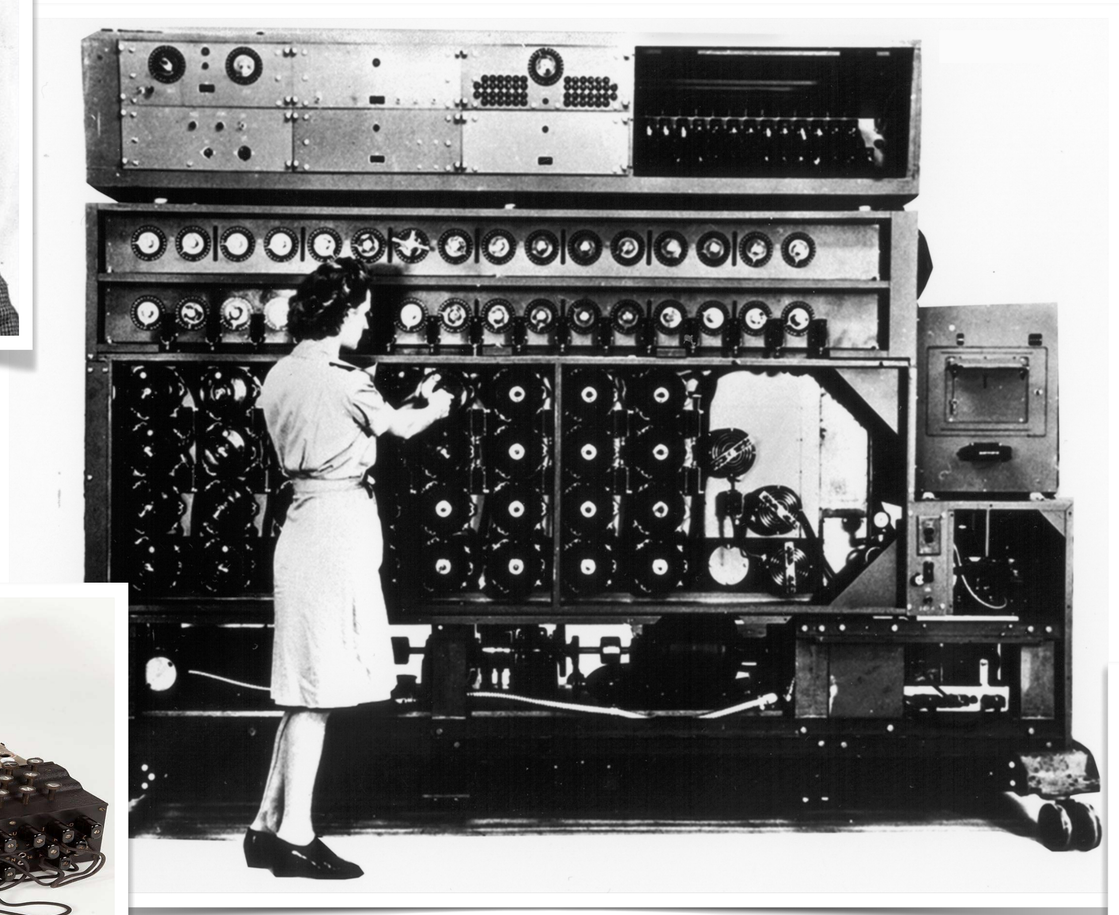


2^3	2^2	2^1	2^0
1	1	1	0

note that in a computer, binary words are not only used to represent integer numbers

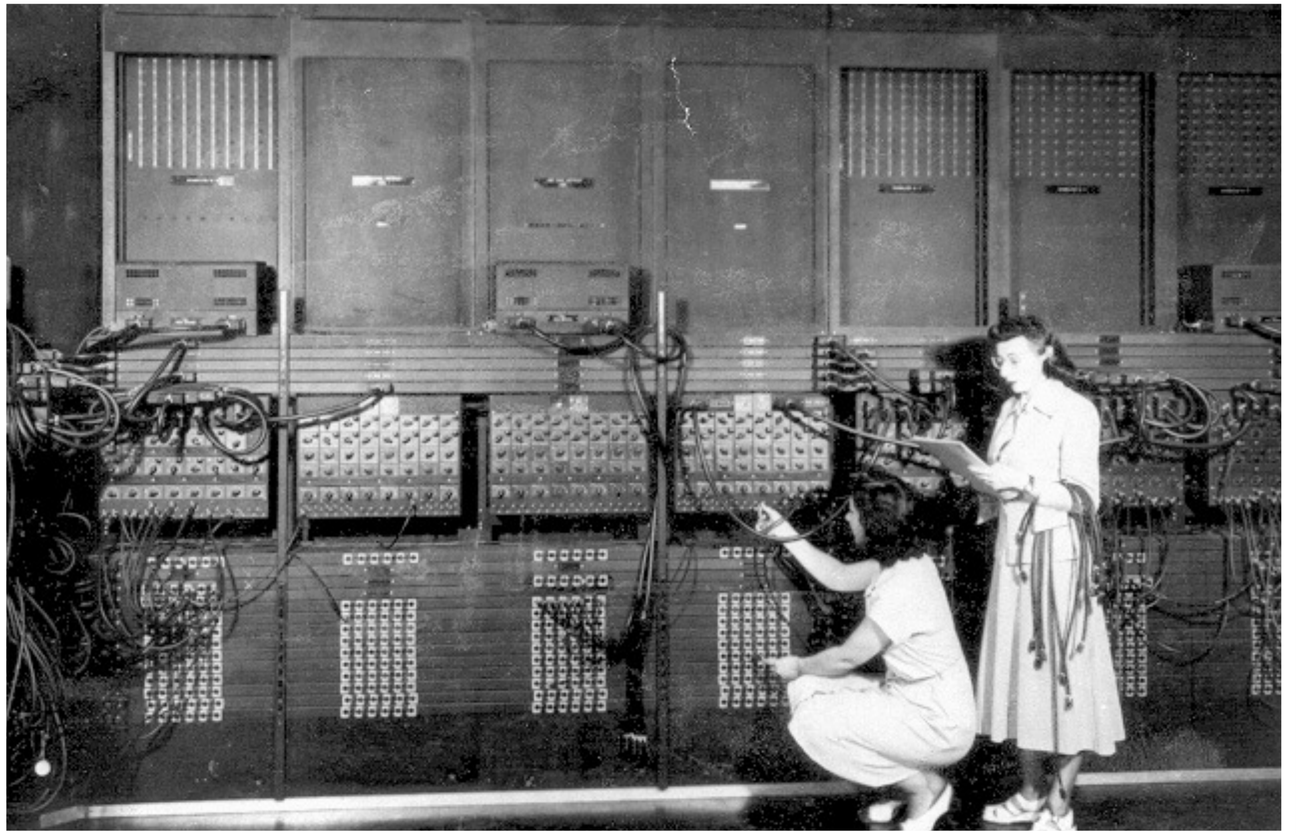
1939-40





the "bombe" was an electromechanical device designed by Alan Turing to decipher German Enigma-encrypted messages

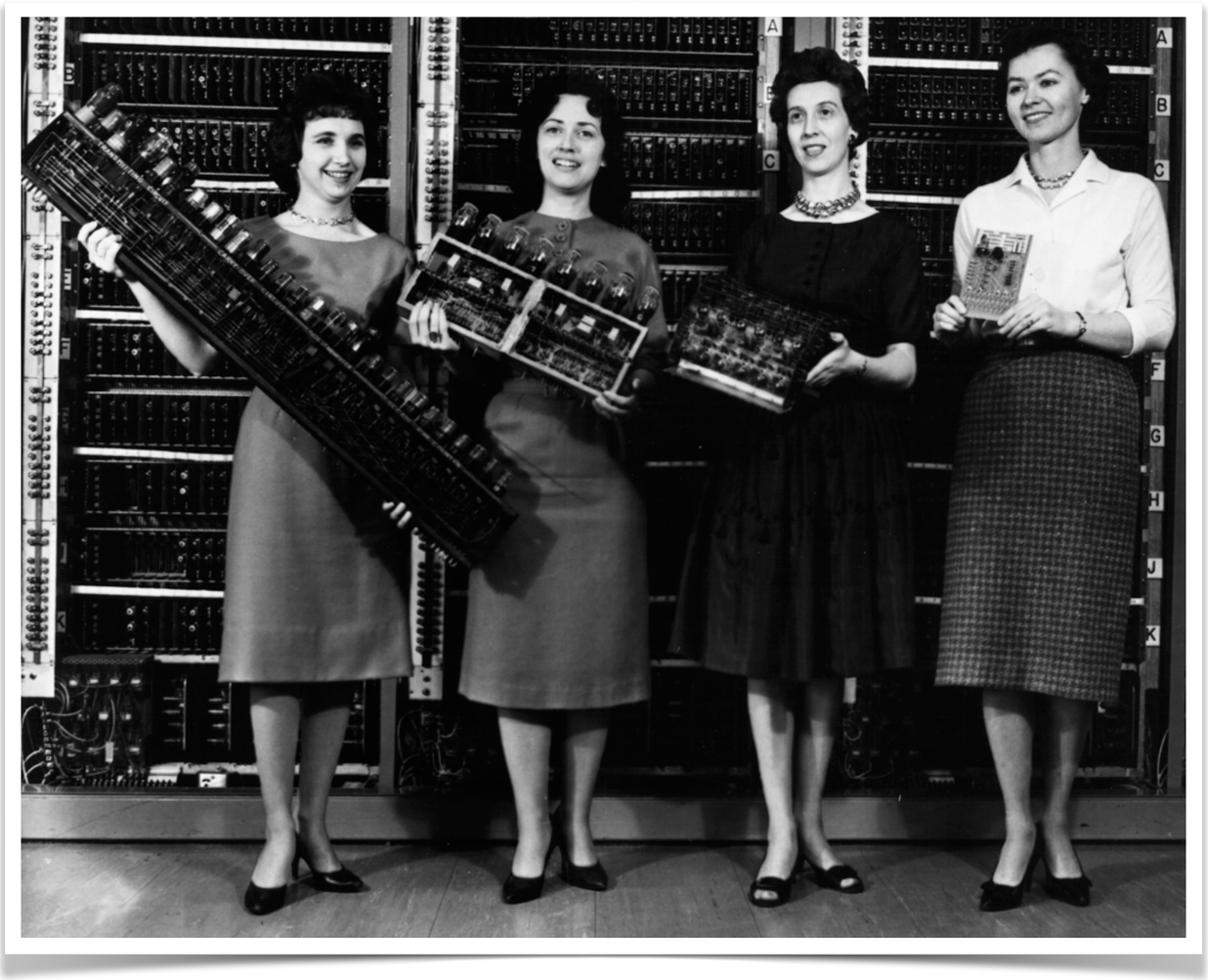
1943



the ENIAC was the first general electronic computer

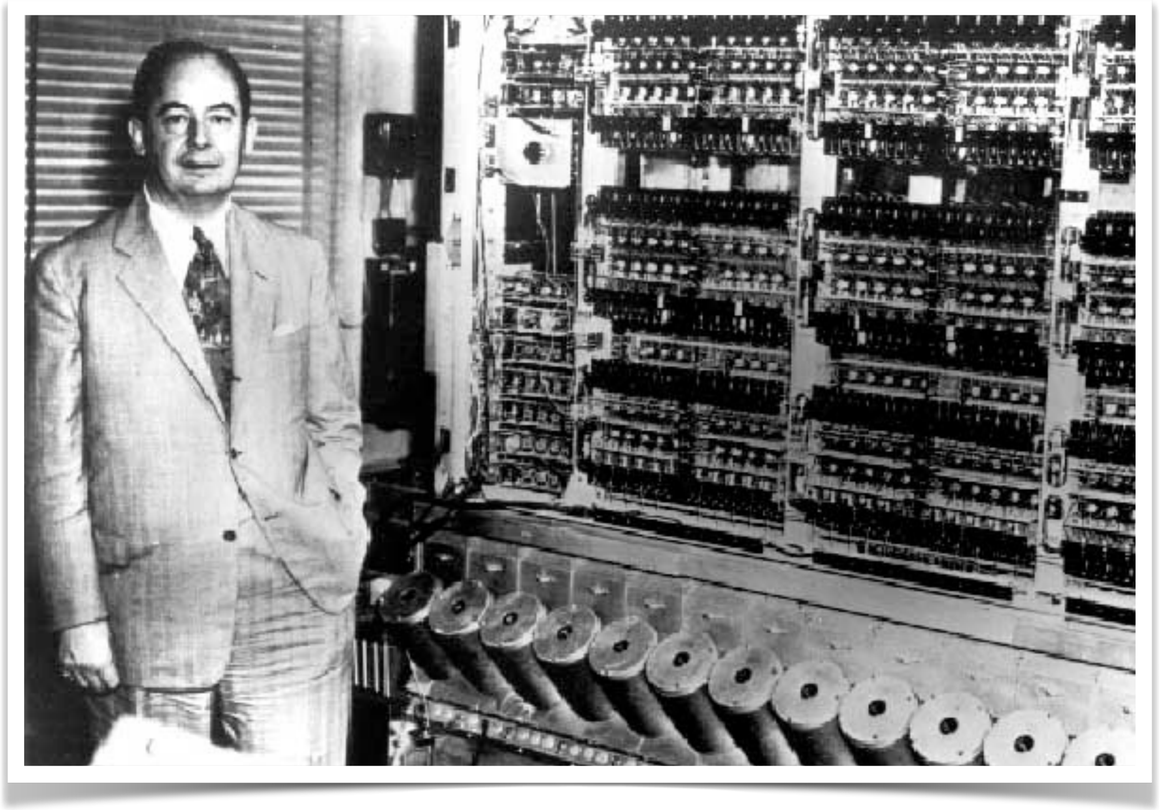
programs were hard-wired (dials & switches)

1944



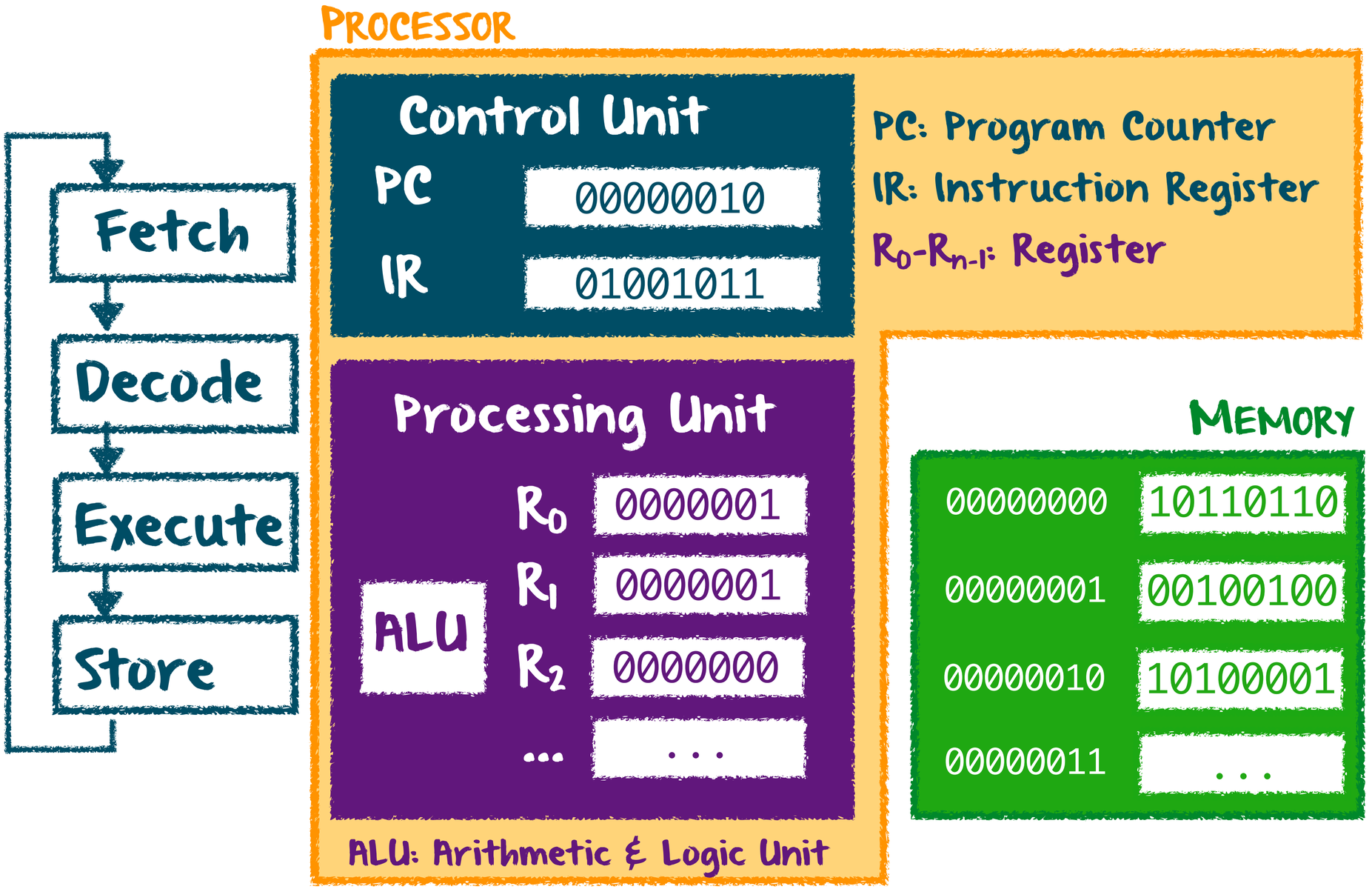
the EDVAC is the first computer to rely on programs stored in memory

1945

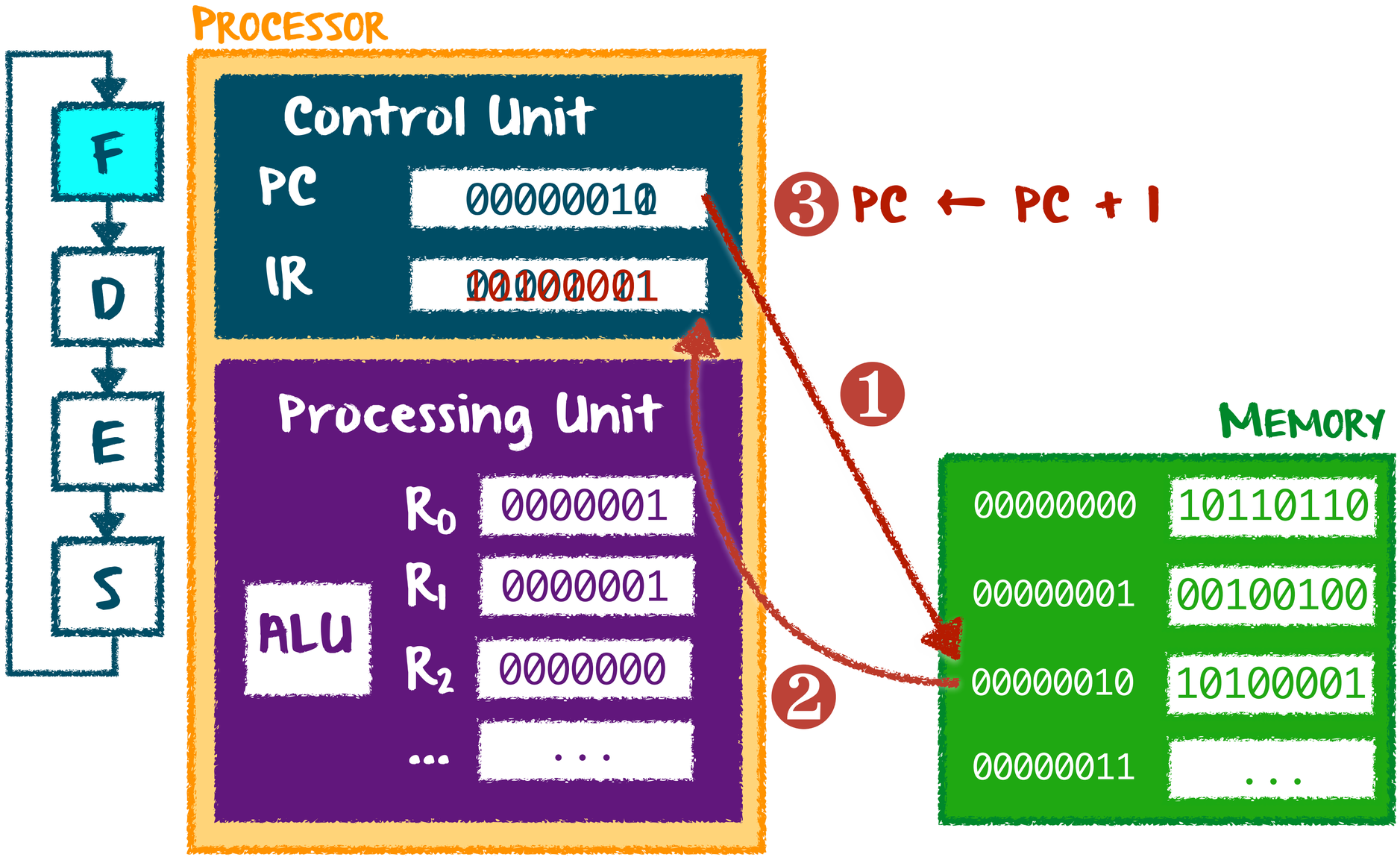


John von Neumann describes the concept of programs stored in memory in a report about the EDVAC computer: the Von Neumann architecture

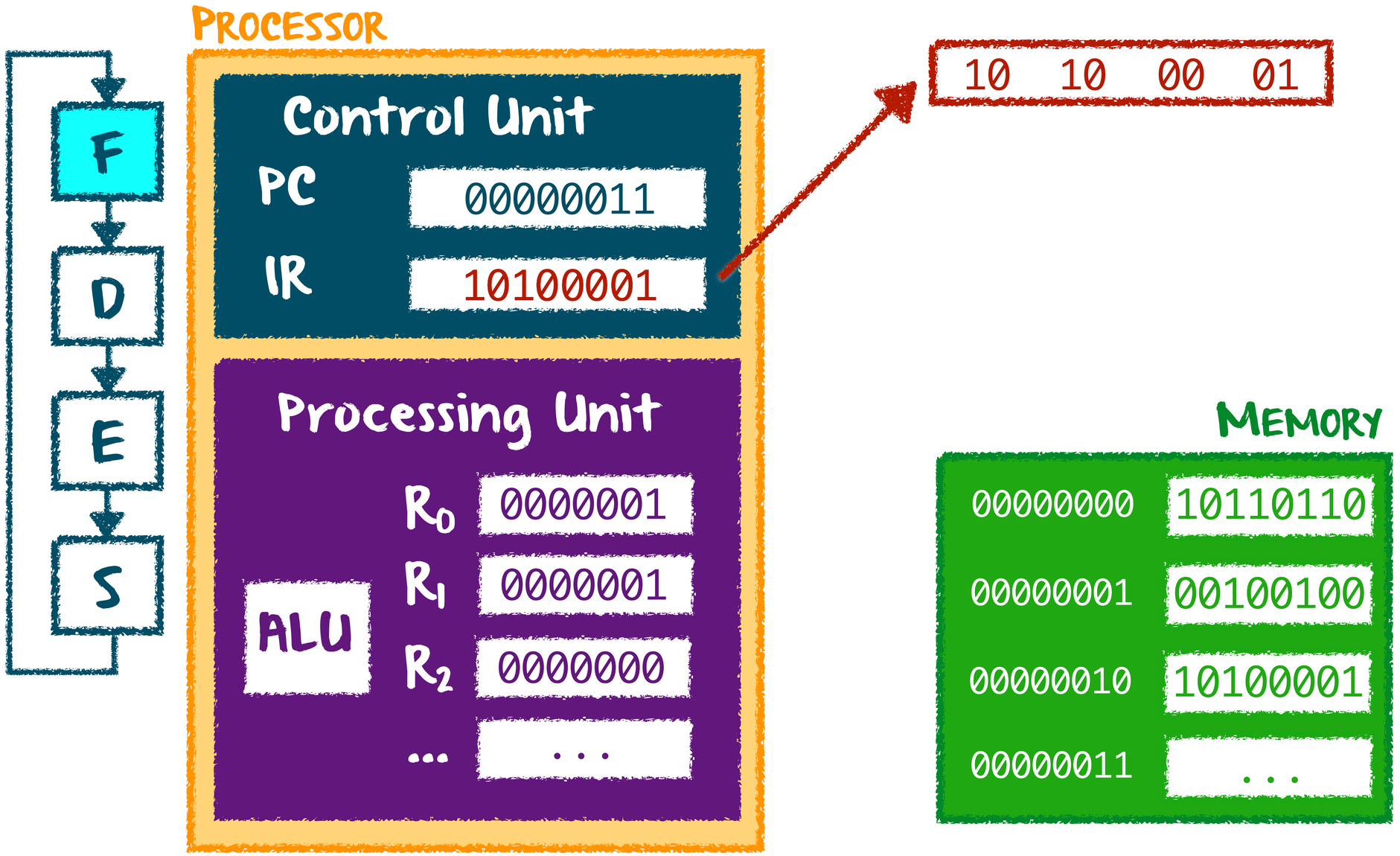
the von Neumann model



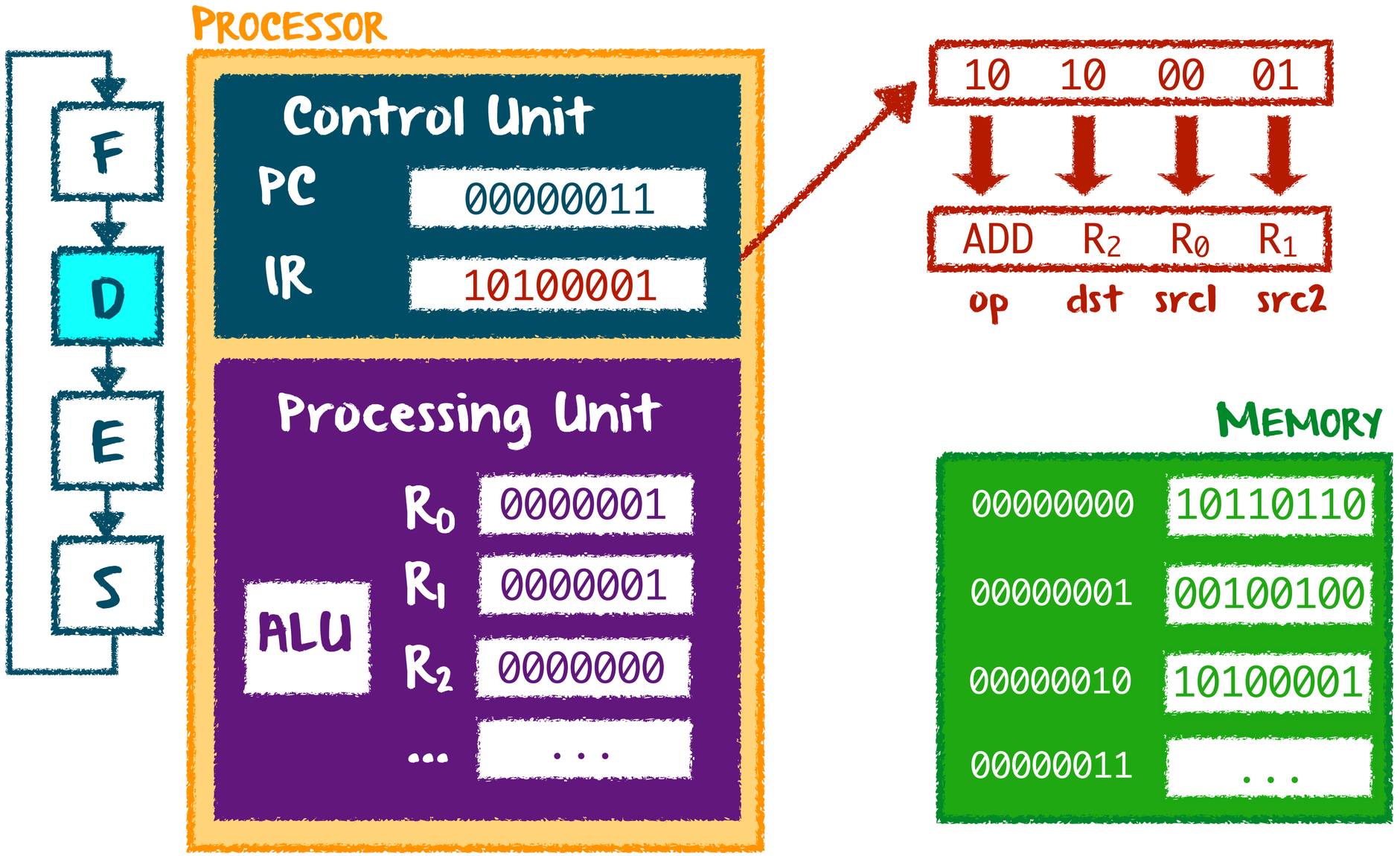
fetch instruction



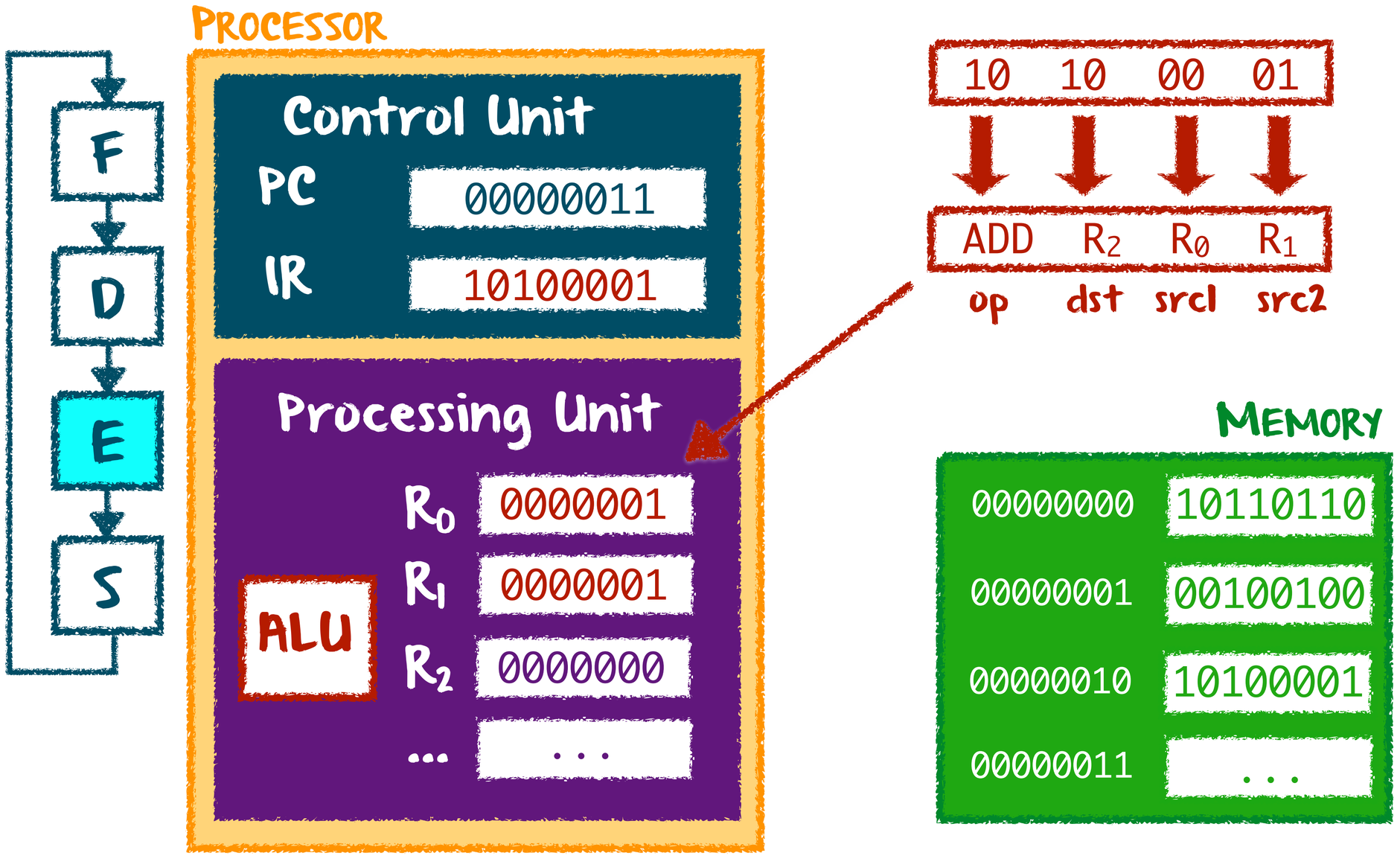
fetch instruction



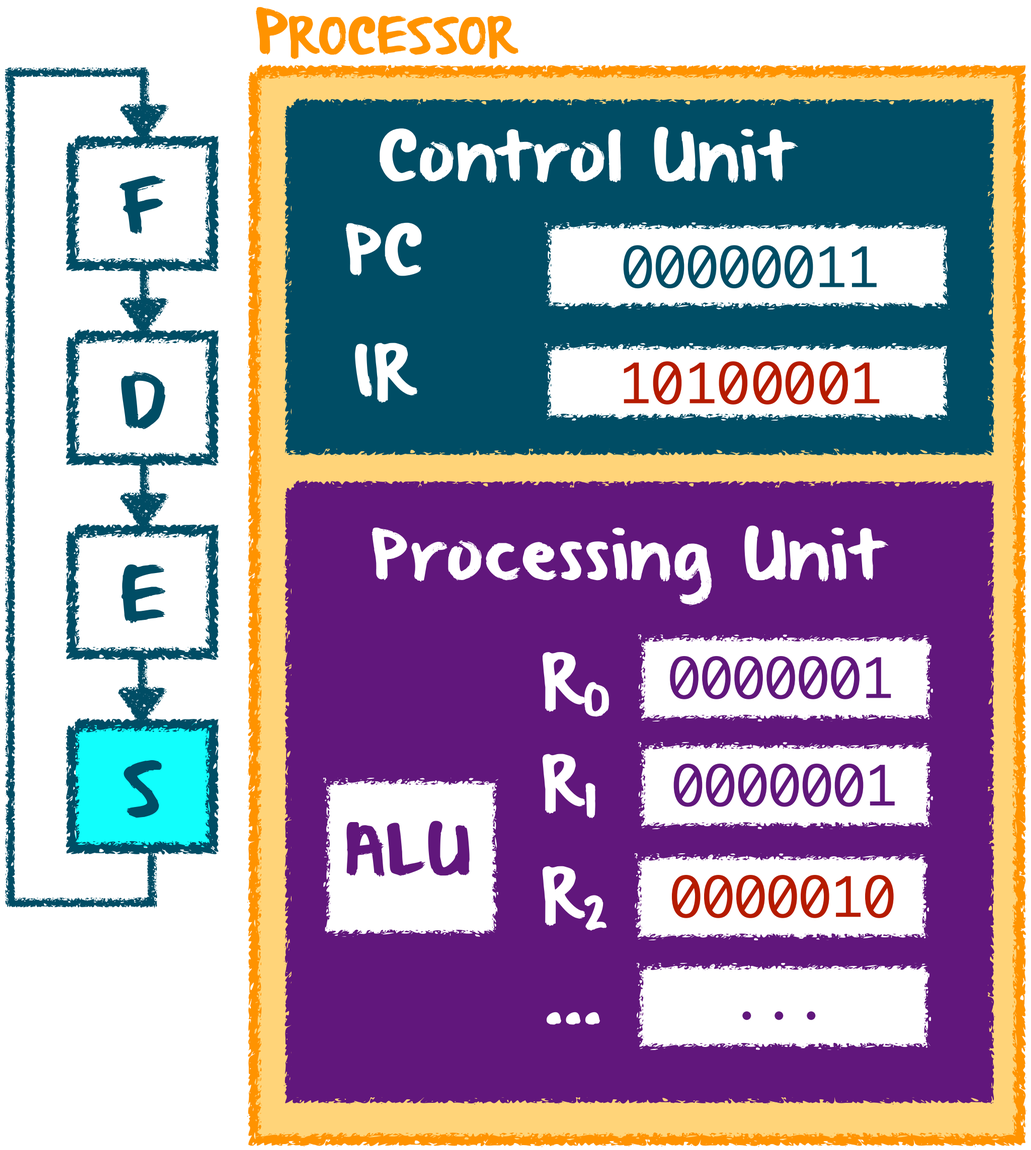
decode instruction

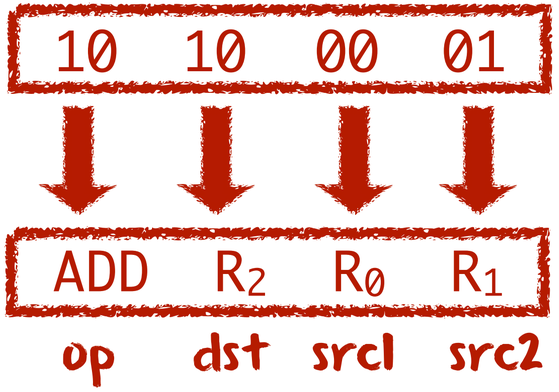


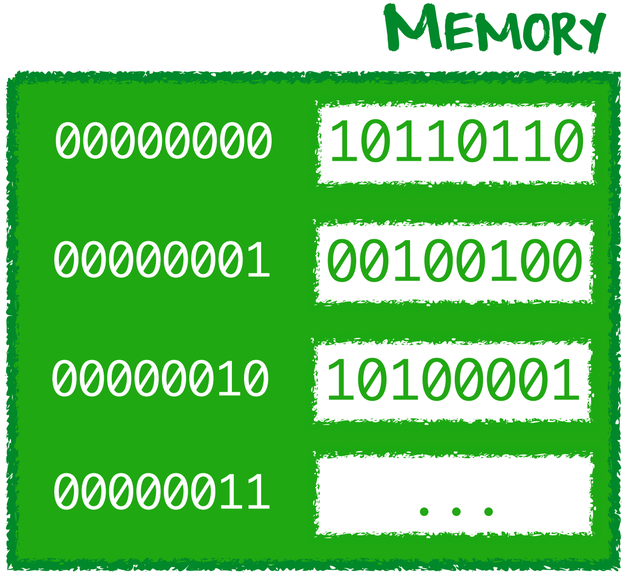
execute instruction



store result







instruction sets

ADDSD—Add Scalar Double-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
F2 0F 58 /r	ADDSD *xmm1, xmm2/m64*	Valid	Valid	Add the low double-precision floating-point value from *xmm2/m64* to *xmm1*.

Description

Adds the low double-precision floating-point values from the source operand (second operand) and the destination operand (first operand), and stores the double-precision floating-point result in the destination operand.

The source operand can be an XMM register or a 64-bit memory location. The destination operand is an XMM register. The high quadword of the destination operand remains unchanged. See Chapter 11 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for an overview of a scalar double-precision floating-point operation.

In 64-bit mode, using a REX prefix in the form of REX.R permits this instruction to access additional registers (XMM8-XMM15).

Operation

```
DEST[63:0] ← DEST[63:0] + SRC[63:0];
(* DEST[127:64] unchanged *)
```

Intel IA-64
[2007]

Motorola 68000
[1980]

ADD — Add (M68000 Family) — ADD

Operation: Source + Destination → Destination

Assembler Syntax: ADD < ea > ,Dn
ADD Dn, < ea >

Attributes: Size = (Byte, Word, Long)

Description: Adds the source operand to the destination operand using binary addition and stores the result in the destination location. The size of the operation may be specified as byte, word, or long. The mode of the instruction indicates which operand is the source and which is the destination, as well as the operand size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X — Set the same as the carry bit.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a carry is generated; cleared otherwise.

Instruction Format: