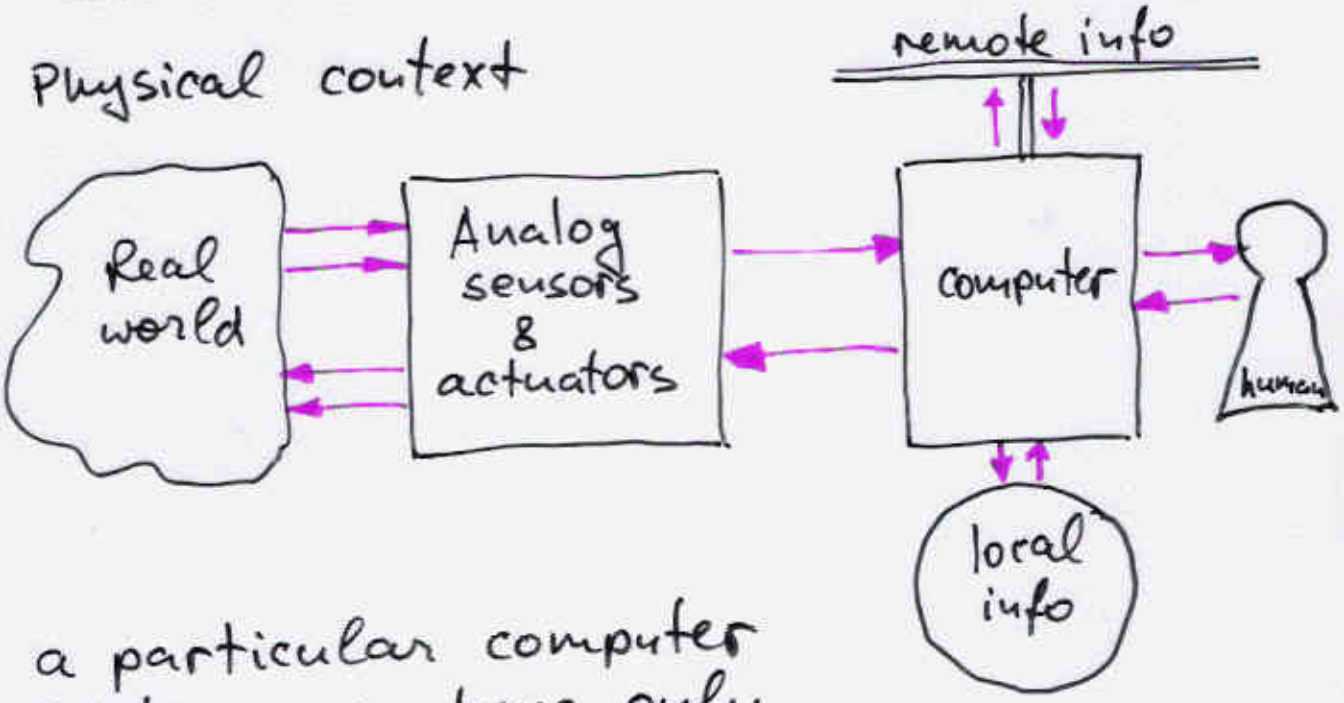
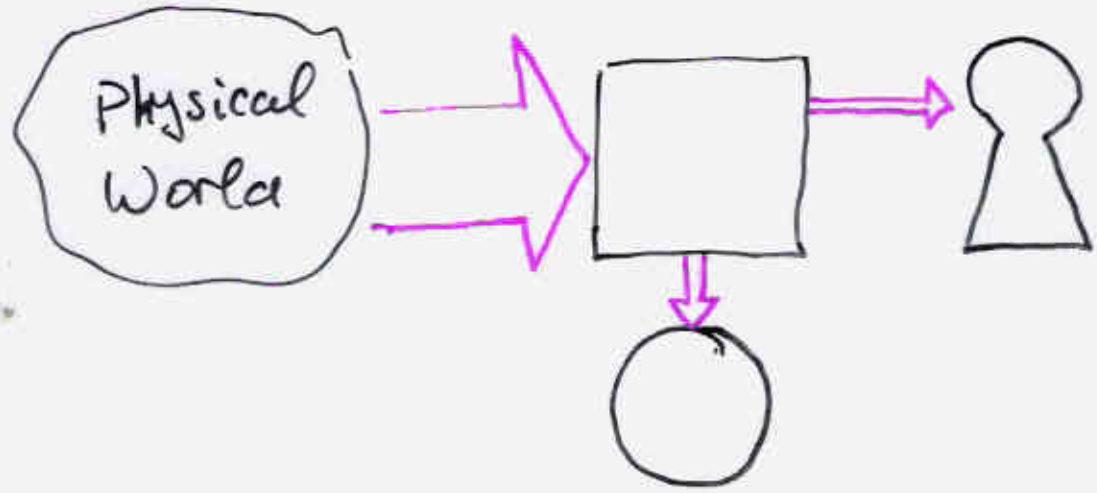
Introduction to microcomputers

- Physical context



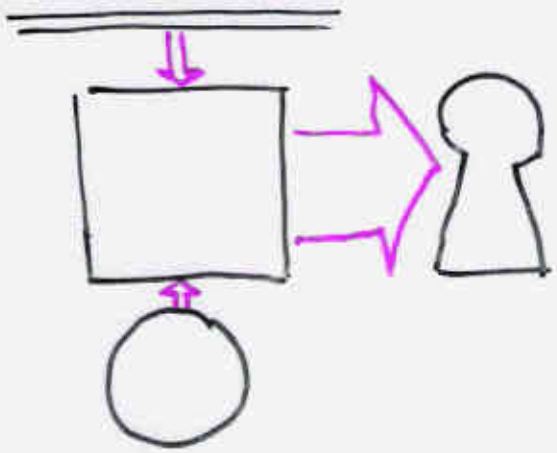
a particular computer system may have only one or all of these "links"

- Information flow - data reduction



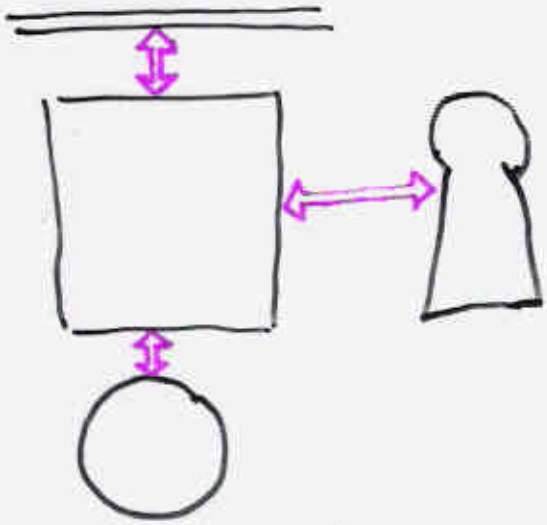
e.g. experimental data acquisition, a measurement

- Information flow – data generation



e.g. a video game; scientific visualization

- Information flow – computation



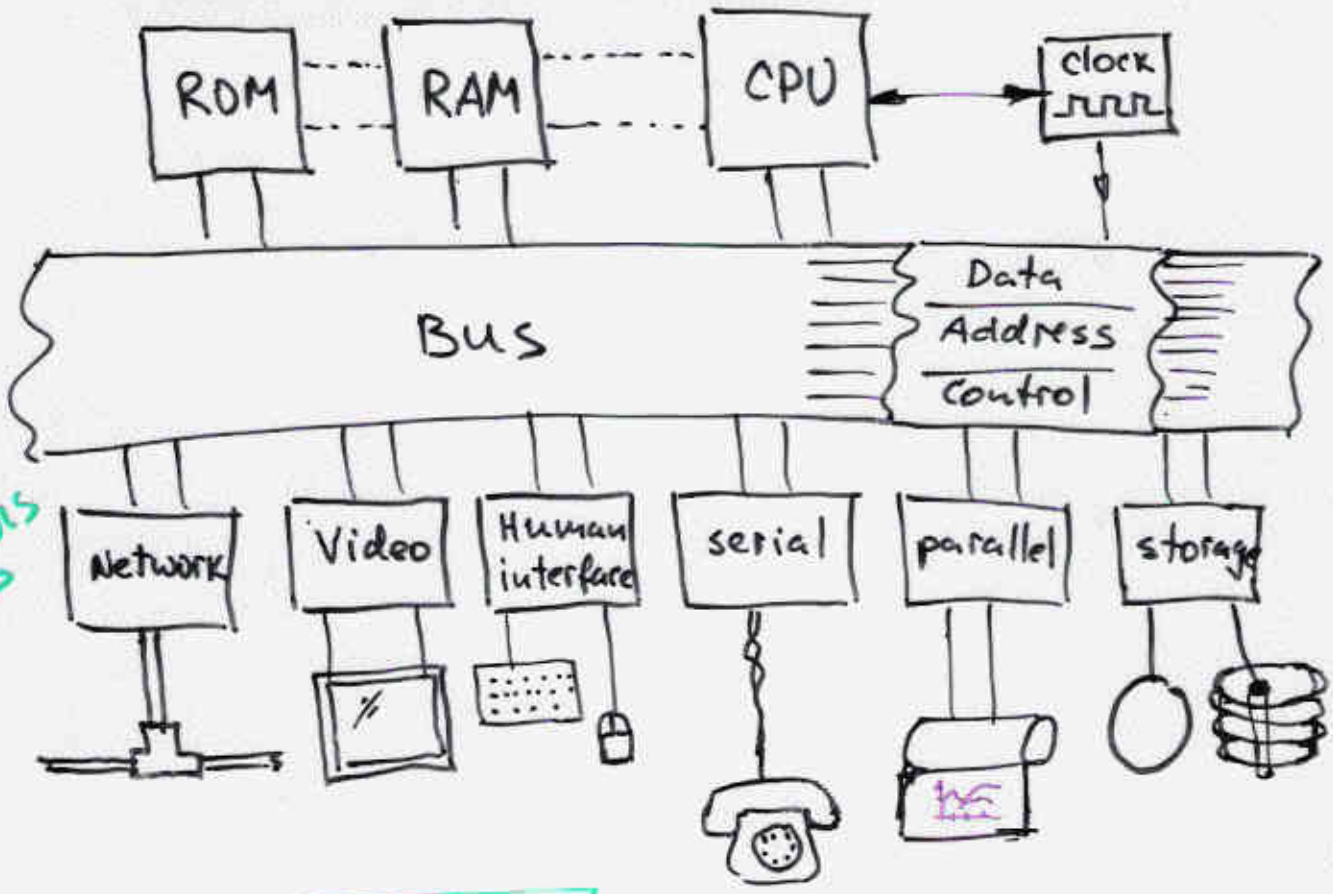
e.g. a scientific calculation

Matching computer architecture to the specific physical and/or informational context involves compromises in:

- complexity, cost (custom/mass-produced)
- serviceability (modular/field-upgrade/hot-plug)
- marketability ("upward" and "downward" compatible/branding)
- etc.

However, many general elements are found in all computers

- System architecture



controllers →

E.g. an "IBM PC"

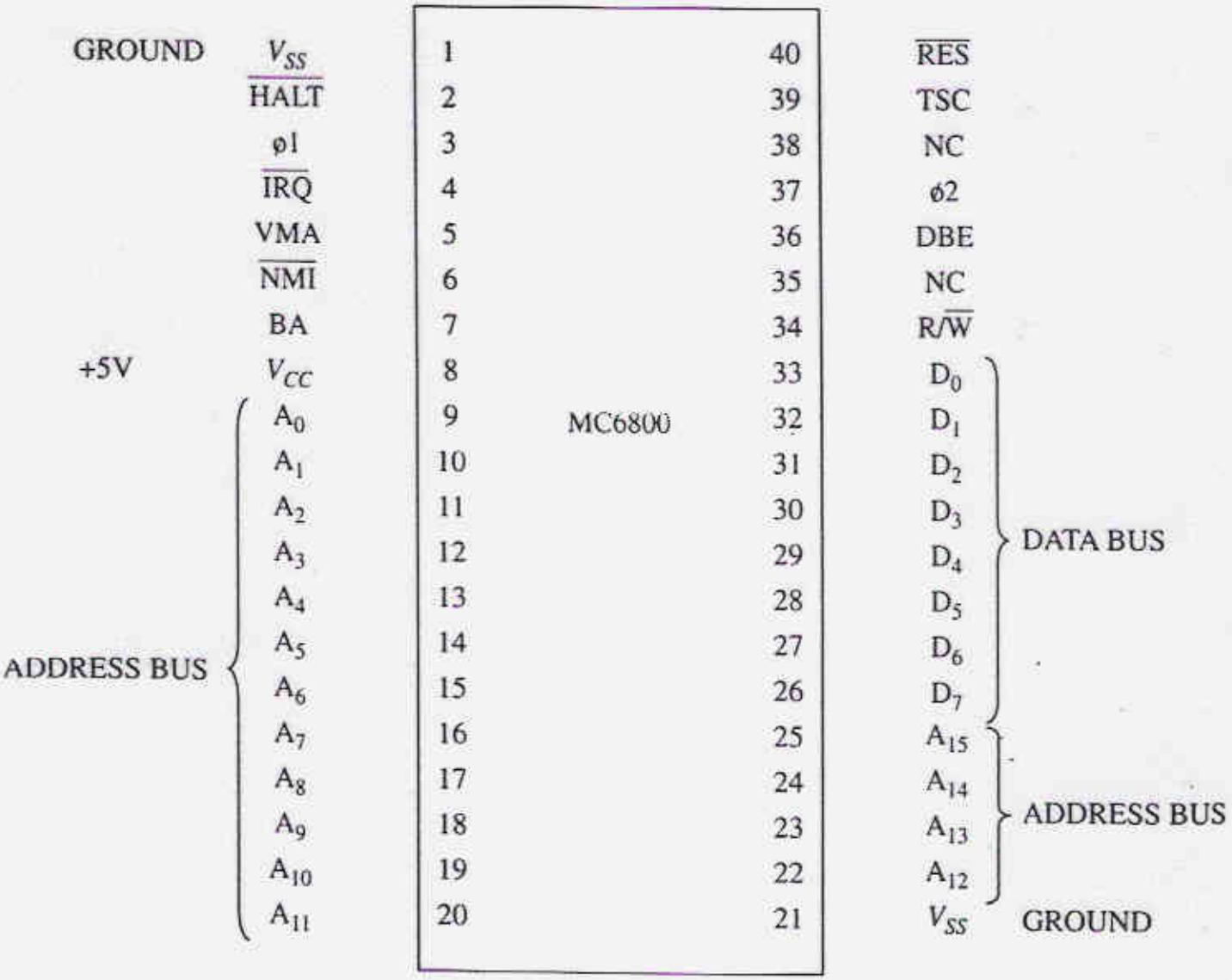
- the motherboard: CPU, RAM (BIOS), some I/O, timing circuits
- I/O Bus: expansion cards
  - address lines: which device is getting/sending data
  - auxiliary lines: timing of the transfer, handshake
  - data lines: numbers, machine instructions, parts of addresses

20
34
AD0-AD7

Operation of the computer boils down to controlled transfer and manipulation of bits (0V & 5V levels) among various components.

- Getting inside the CPU

E.g., a Motorola 6800 (Atari, Macs)



NC — no connection

FIGURE 13.11 **Pin-out of an MC6800 microprocessor.**

Motorola Corp.

• Inside a 6800

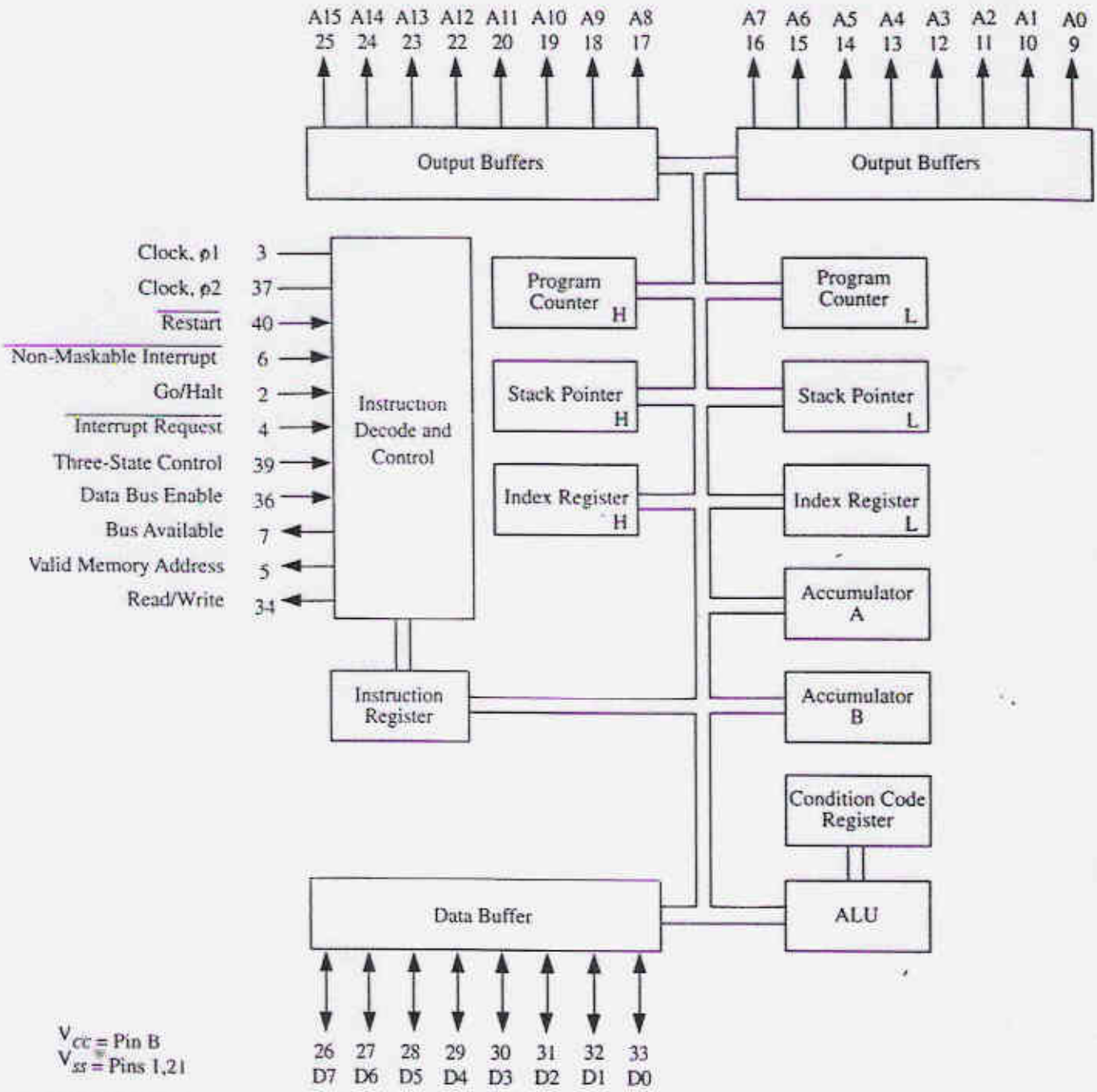


FIGURE 13.12 **Block diagram of MC6800 microprocessor with pin-out connections. Adapted from *The Complete Motorola Microcomputer Data Library*, Motorola 1978.**

• Inside Intel 8080 (IBM PC)

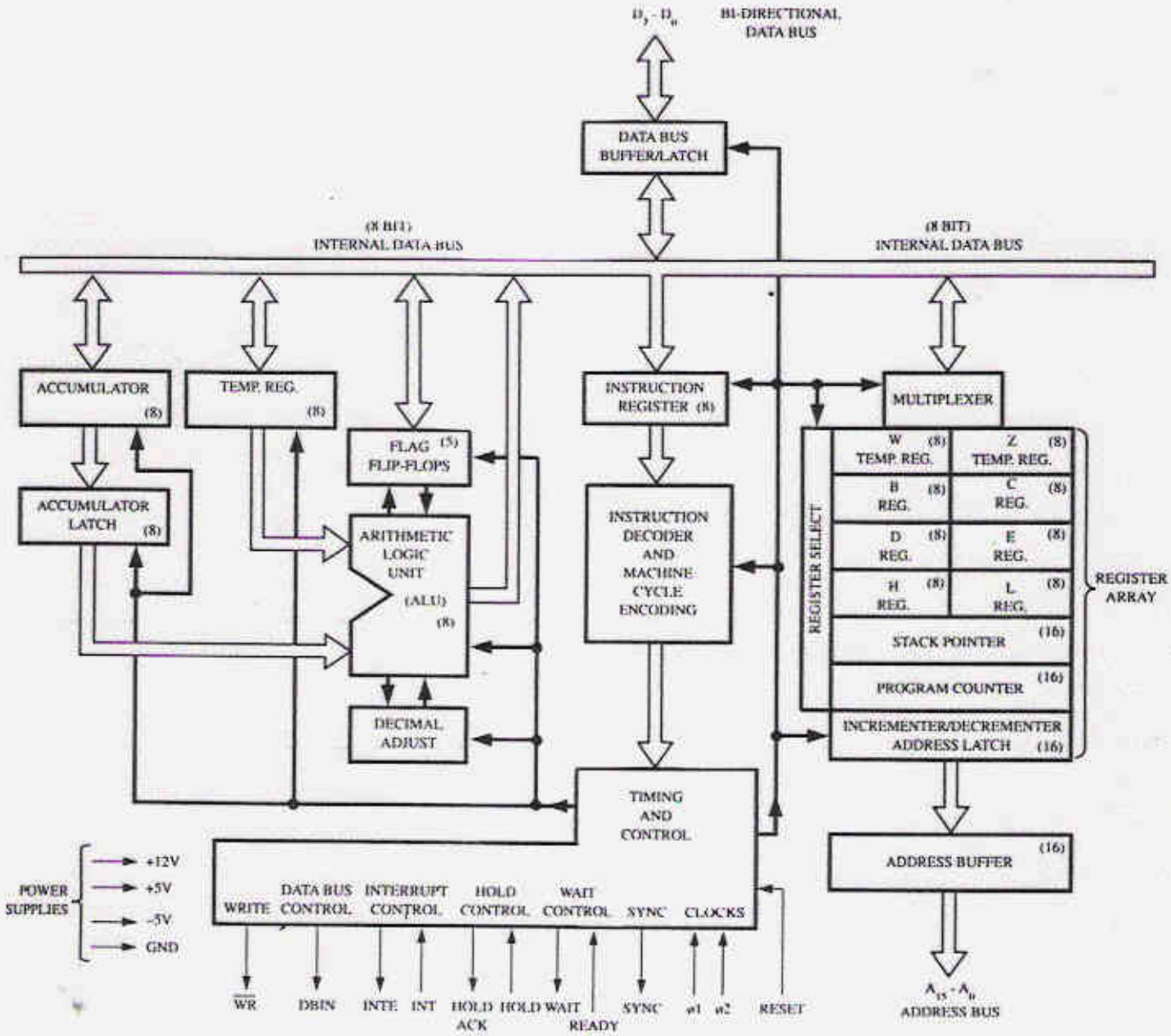
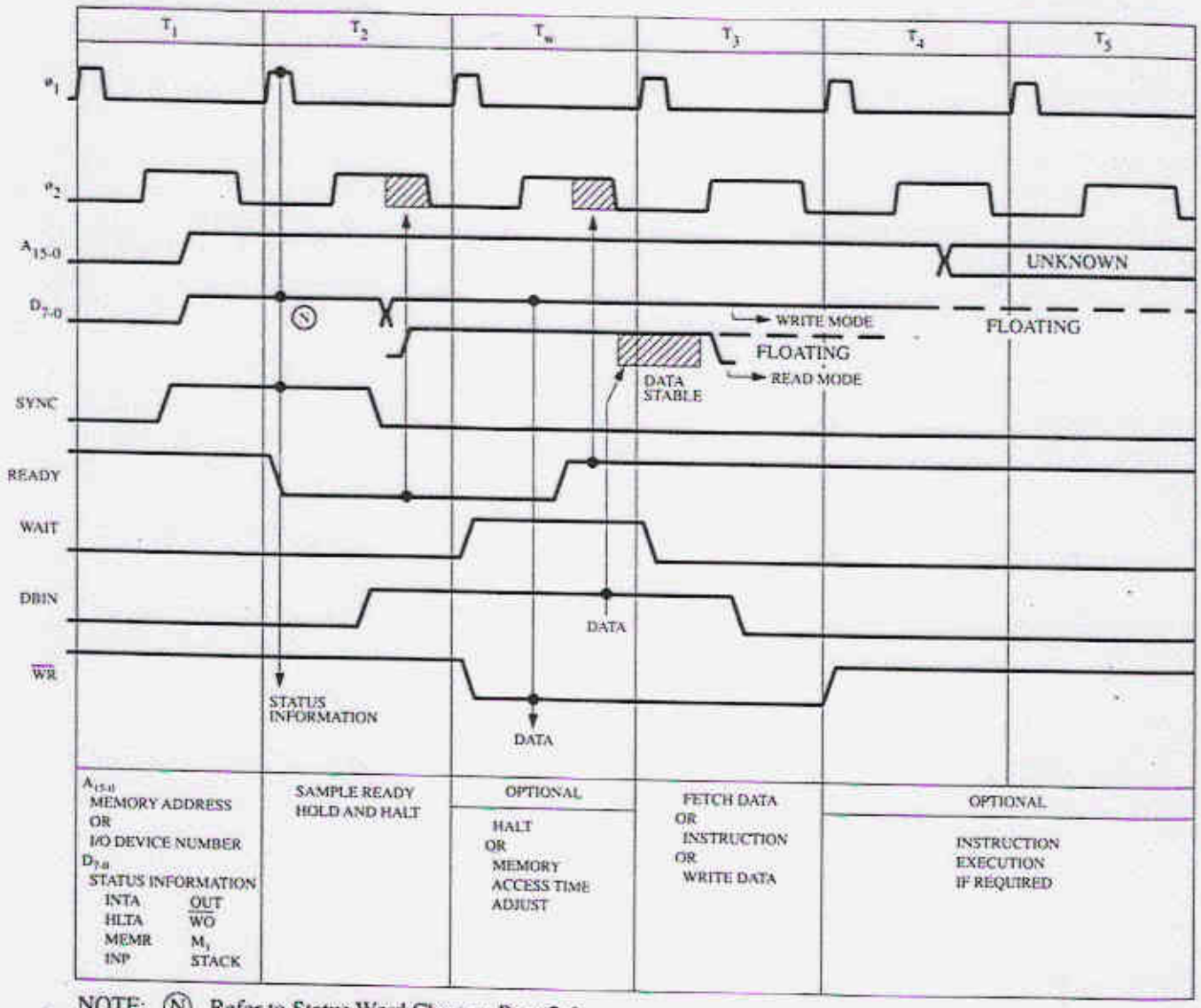


FIGURE 13.17 **The Intel 8080 organization. (Courtesy of Intel Corp.)**

• "controlled transfer and manipulation" of electrical levels



NOTE: (N) Refer to Status Word Chart on Page 2-6.

FIGURE 13.19 **The timing cycle for the 8080. (Courtesy of Intel Corp.)**

- Architectural considerations

1970-80s : how many operands?

$C = A + B$ is a 3-operand instruction

⇒ many clock cycles, and much hardware is needed to perform:

- load value at address A; and
- load value at address B; and
- add; and
- write value to address C

D.E. Knuth in 1971 : 80% of code is of the form :

$A = A \text{ (operation) } B$ or $A = B$ } 2 operands at most!

cf. RPN calculators

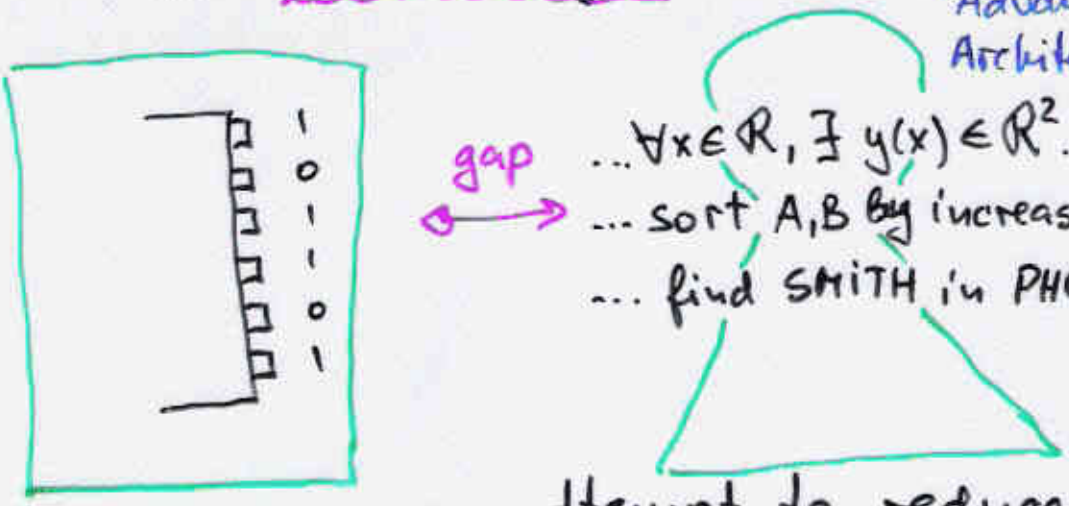
- CISC : complex instruction set computers

e.g. DEC (Digital) VAX-11

had hundreds of "elementary" instructions, reflecting the desire for high-level language support : integer, floating-point, string, array operations.

- the "semantic gap"

Glenford Meyers "Advances in Computer Architecture", Wiley 1978



CISC was an attempt to reduce this semantic gap.

1987: National Semiconductor 32000series implemented essentially the entire VAX-11 CISC instruction set on a chip.

- RISC: reduced instruction set computers

since mid-1980s, the winning strategy (and marketing tool!) is to reduce and simplify the instruction set, but speed up the performance of the individual instruction, ideally in 1 clock cycle

NO — floating-point, string, array — operations

Instead: - pipelining (pre-fetching data for ALU)
- fast CACHE memory (most of the time, the instructions that follow in sequence are at nearby memory locations, pre-load the whole bunch)

What was seen as a major advantage of CISC – the ease and compactness of programming – has been largely restored with better software development tools.

Marketing? "X-RISC" :)

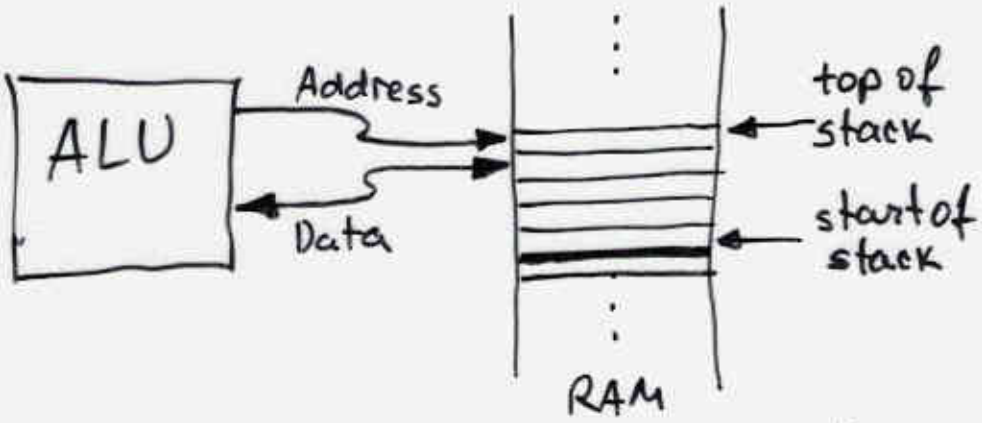
However, large-scale integration means that today's RISCs are not that R, as the instruction-set complexity is creeping up. e.g. MMX on Pentium chips.

- specialized architectures
  - parallel and vector processors: (late 1980's – early 90s)
    a single instruction applied to many data: $A = A + B \rightarrow \vec{A} = \vec{A} + \vec{B}$
    Efficient for a narrow range of applications
    e.g. graphics processing, spectral analysis
  - variable-word-length processors
    (Kendall Square Research, mid-1990s)

	op.	operands
sometimes	+	$\lvert A \rvert B \rvert$
	+	$\lvert A_1 \rvert A_2 \rvert \ldots \lvert A_N \rvert B_1 \rvert B_2 \rvert \ldots \lvert B_N \rvert$

- ALU = the bit-pushing heart of the CPU

  - stack architecture : zero explicit operands per instruction



C = A + B

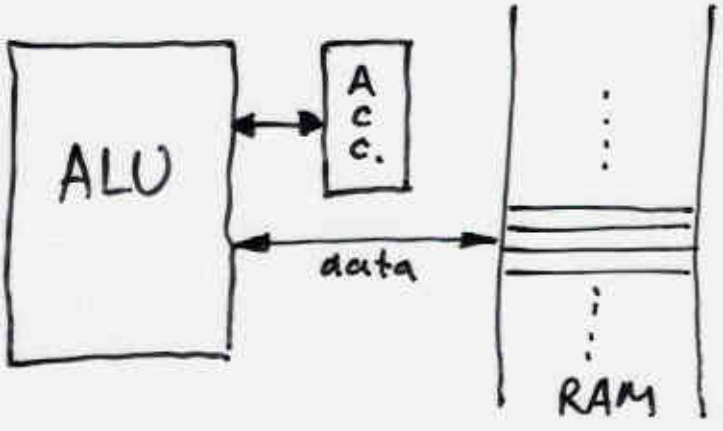
push A
push B
ADD
pop C

source: top two elements on stack ("pop")

result: top of stack ("push")

e.g. HP3000

  - accumulator architecture : one explicit operands per instruction



C = A + B

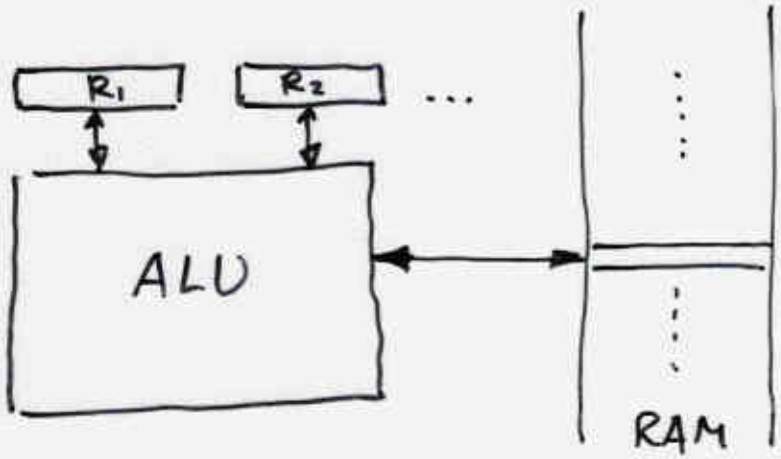
LoadAcc A
Add Acc B
StoreAcc C

source: — (1) accumulator
— (1) memory location

result: accumulator (Acc = Acc + Mi)

e.g. MC6800, Intel 8080

- register set architecture:
  2 or 3 explicit operands per instruction



C = A + B

Load R_1, A
Add R_1, B
Store C, R_1

source: one of registers (or ports), memory
result: ——11—— 11 ——11——

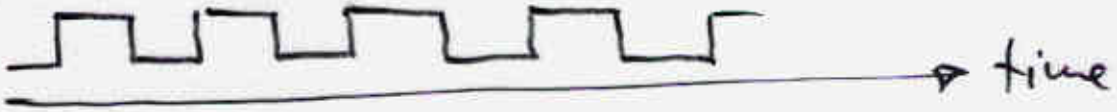
e.g. IBM360, DEC VAX-11, MC 680x0, ...

- hybrid architectures
  e.g. Z80, a cross between accumulator and register set architectures

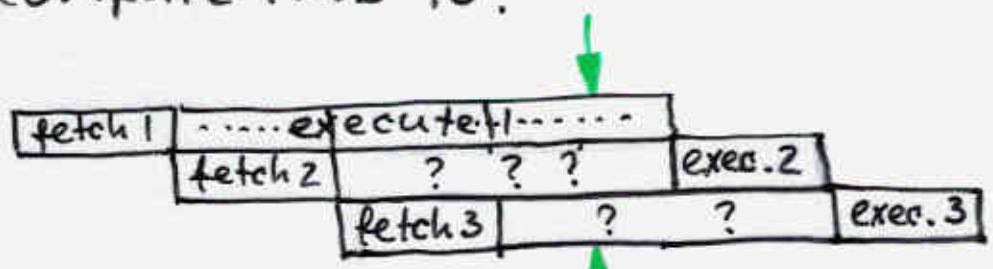
- a historical note
  - stack and accumulator architectures popular early on, while memory/registers are expensive
  - register set architecture dominates today: code size not so important, high-speed large-scale logic arrays available, use of registers reduces execution time (no memory fetches)

- streamlining RISC

- one cycle per instruction:



compare this to:



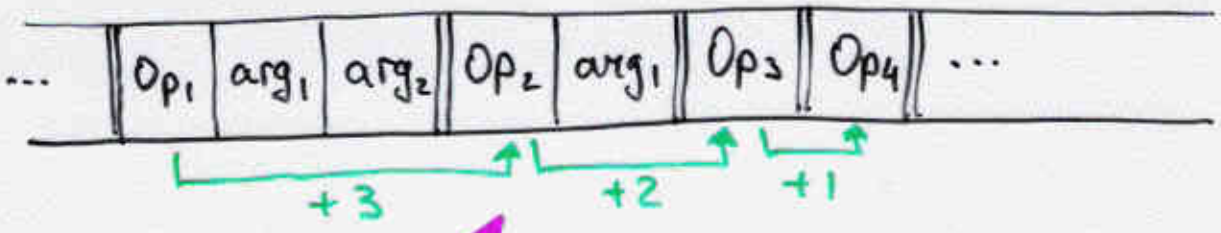
multiple instructions are pending! (a varying number)

- separate data and program memory

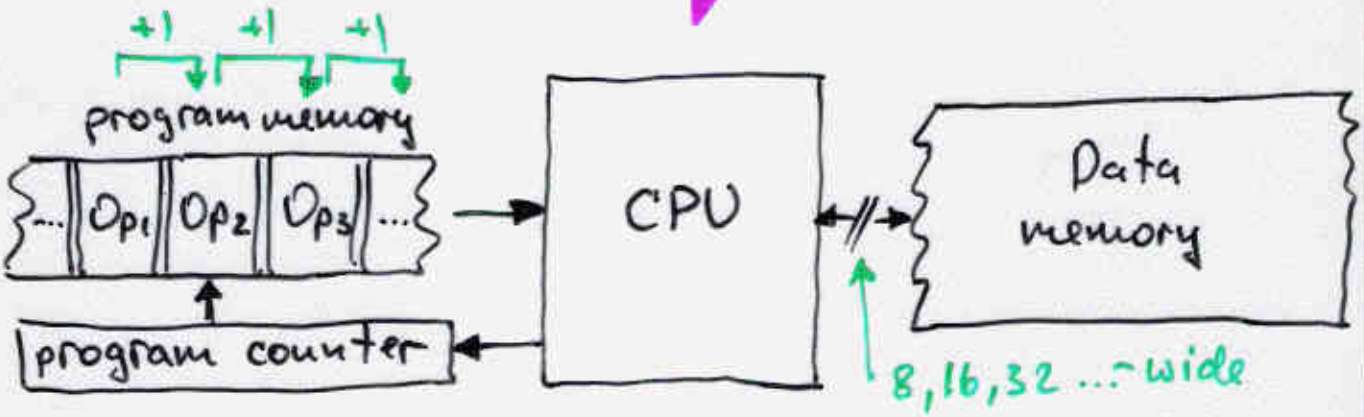
8 bits: $\left\{\begin{array}{l} \text{256 possible addresses} \\ \text{"} \quad \text{"} \quad \text{operations codes} \\ \text{"} \quad \text{"} \quad \text{data values} \end{array}\right.$

eg. PIC only has 35 instructions

Using the same-size bus means that when an op.code is on the bus some bits may go unused, or that some operations may require multiple addresses. Thus the next operation may be in the next address or (1, 2, 3...) addresses later, depending on the number of operands



von Neumann architecture vs. Harvard architecture

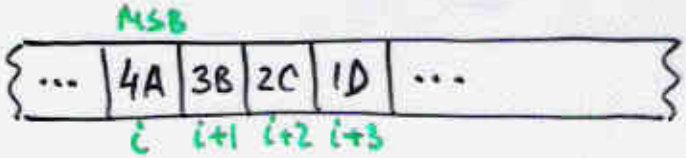


- little- vs. big-endian or byte order in multibyte data:

Jonathan Swift "Gulliver's Travels" (end of eggs)

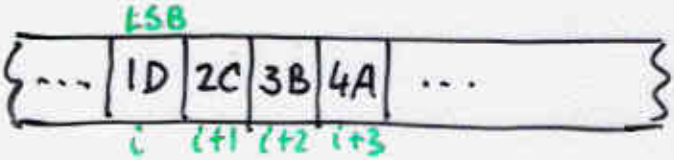
e.g: MSB 4A 3B 2C 1D LSB = a four-byte datum

- Big-endian:



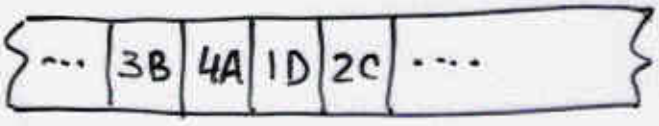
68000 IBM/370

- little-endian:



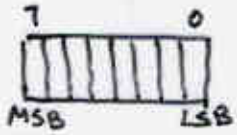
6502 Intel x86 DEC VAX

- mixed-endian: (middle-endian)



PDP-11 VAX f.p.

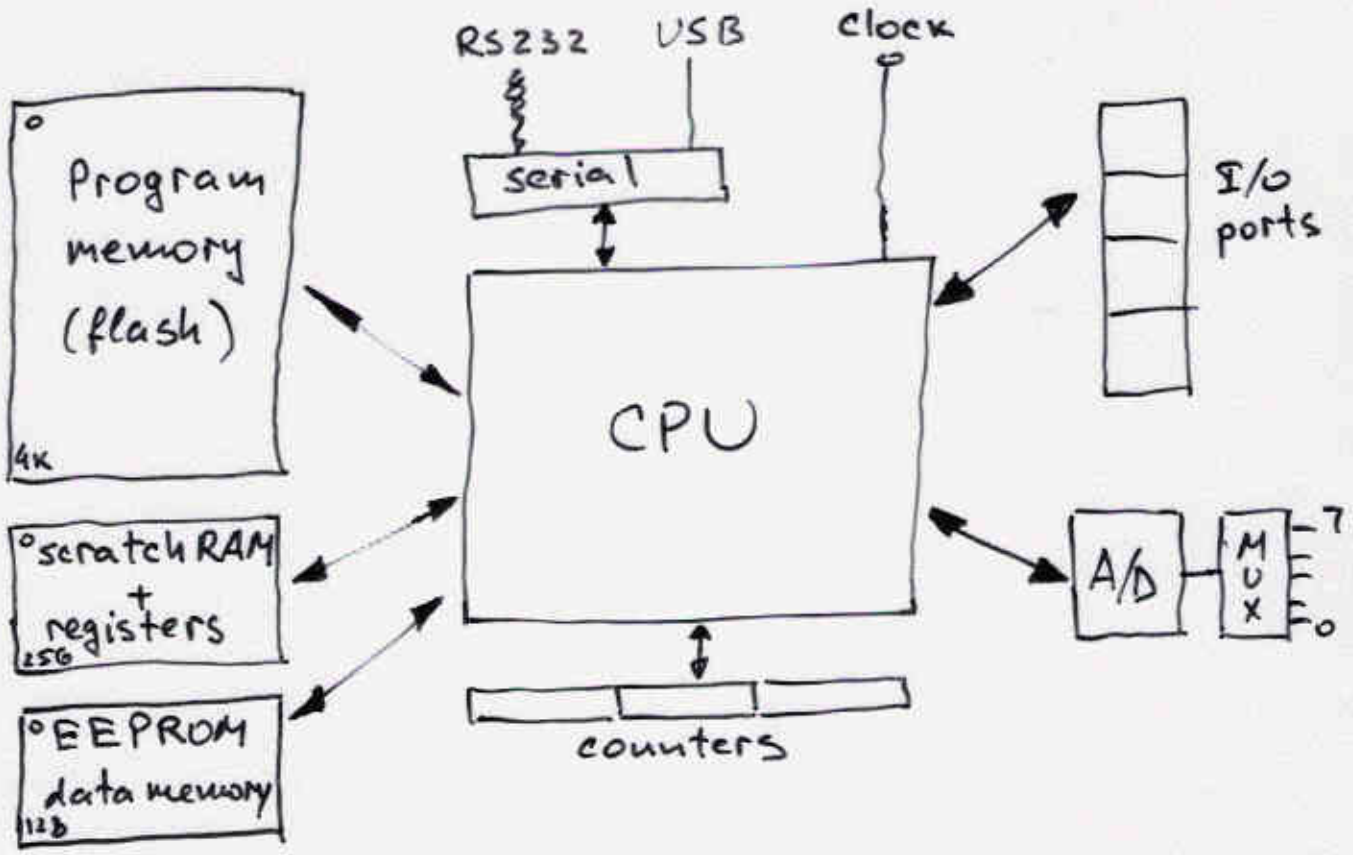
- byte = 8 bits



- Microcontrollers

  – high-integration, single-IC, a "computer on a chip"

e.g. PIC 16F874



* flash: retained when power is off
  ~ 10,000 re-writes under external control

* EEPROM: retained when power is off
  ∞ re-writes under program control