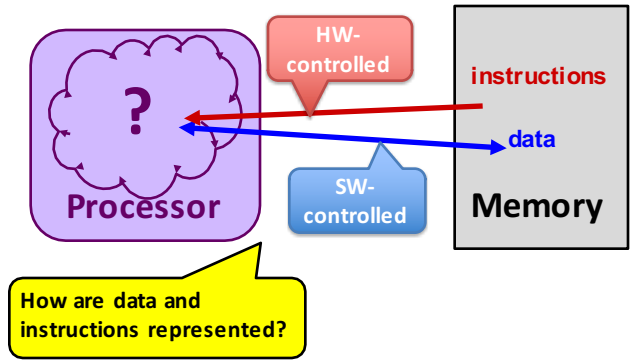
representing data with bits

bits, bytes, numbers, and notation

1930s | 1940s | 1950s | 1960s | 1970s | 1980s | 1990s | 2000s | 2010s

Modern Digital Computer

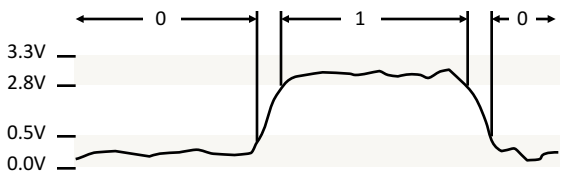
"von Neumann" model



---

bit = binary digit = 0 or 1

Electronically: high voltage vs. low voltage



Basis of all digital representations

ints, floats, chars, strings, booleans, etc.

machine instructions

---

positional number representation

fancy name, familiar concept

2	4	0
100	10	1
10^2	10^1	10^0
2	1	0

$= 2 \times 10^2 + 4 \times 10^1 + 0 \times 10^0$

weight

position

- **Base** determines:
  - Maximum digit (base – 1). Minimum digit is 0.
  - Weight of each position.
- Each position holds a digit.
- Represented value = sum of all position values
  - Position value = digit value x base$^{\text{position index}}$

base 2 (binary)

1	0	1	1
8	4	2	1
2^3	2^2	2^1	2^0
3	2	1	0

$= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

weight

position

When ambiguous, subscript with base:

101_{10} Dalmatians (movie)

101_2-Second Rule (folk wisdom about dropped food)

5

conversion and arithmetic

$19_{10} = ?_2$ One strategy:

Subtract largest power of 2 that is <= 19 from 19_{10} and at it to 0_2.
Repeat with remaining$_{10}$ and sum$_2$ until 0_{10} remains.

$240_{10} = ?_2$

$11010011_2 = ?_{10}$

$101_2 + 1011_2 = ?_2$

$1001011_2 \times 2_{10} = ?_2$

6

numbers and wires

- One wire per bit
- How many wires do we need…?
- Fixed-size encodings to build hardware.

What do you call 4 bits?

byte = 8 bits

a.k.a. octet

Smallest unit of data
used by a typical modern computer

Binary 00000000_2 -- 11111111_2
Decimal 000_{10} -- 255_{10}
Hexadecimal 00_{16} -- FF_{16}

Byte = 2 hex digits!

Programmer's hex notation (C, etc.):
0xB4 = $B4_{16}$

Octal (base 8) also useful.

Why do 240 students often confuse Halloween and Christmas?

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

8

word |wərd|, n.

Natural unit of data used by processor.

- Fixed size (e.g. 32 bits, 64 bits)
  - depends on ISA: Instruction Set Architecture
- machine instructions usually operate on words
- word size = register size = address size

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Java/C int = 4 bytes: 11,501,584

9

C programming language

- Invented in 1970s to help build UNIX operating system
  - OS manages hardware, C close to machine model
- Simple pieces look like Java:
  - if, while, for, local variables, assignment, etc.
- Other pieces do not:
  - no objects, no methods, no array bounds checks
  - addresses, pointers, structs, functions, weak type system
- Important language, still widely used,
  but many good PL ideas have come along since.

fixed-size data representations

Java Data Type	C Data Type	(size in bytes) 32-bit	64-bit
boolean		1	1
byte	char	1	1
char		2	2
short	short int	2	2
int	int	4	4
float	float	4	4
	long int	4	8
double	double	8	8
long	long long	8	8
	long double	8	16

Depends on word size!

11

boolean algebra, C notation

AND: A&B

&	0	1
0	0	0
1	0	1

OR: A|B

|	0	1
0	0	1
1	1	1

XOR: A^B

^	0	1
0	0	1
1	1	0

NOT: ~A

~	
0	1
1	0

12

general boolean algebras

Operate on **bit vectors** with **bitwise operators**.

```
  01101001      01101001      01101001
& 01010101    | 01010101    ^ 01010101    ~ 01010101
  01000001
```

Laws of boolean algebra apply.

```
  01010101
^ 01010101
```

e.g., DeMorgan's Law: ~(A | B) = ~A & ~B

How does this relate to set operations?

13

sets as bit vectors

Representation

w-bit vector represents subsets of {0, …, *w*–1}.

$a_i = 1 \equiv i \in A$

```
01101001        { 0, 3, 5, 6 }
76543210

01010101        { 0, 2, 4, 6 }
76543210
```

Operations

&	Intersection	01000001	{ 0, 6 }
|	Union	01111101	{ 0, 2, 3, 4, 5, 6 }
^	Symmetric difference	00111100	{ 2, 3, 4, 5 }
~	Complement	10101010	{ 1, 3, 5, 7 }

14

bitwise operations in C

`& | ^ ~` apply to any "integral" data type
`long, int, short, char, unsigned`

Examples (`char`)

```
~0x41 =

~0x00 =

0x69 & 0x55 =

0x69 | 0x55 =
```

Many bit-twiddling puzzles in first assignment

15

logical operations in C

`&&  ||  !` apply to any "integral" data type
`long, int, short, char, unsigned`

0 is false nonzero is true
result always 0 or 1
Early termination a.k.a. **short-circuit evaluation**

Examples (`char`)

```
 !0x41 =
 !0x00 =
!!0x41 =

0x69 && 0x55 =
0x69 || 0x55 =
```

16