Representing and Storing Numbers

cs4: Computer Science Bootcamp

Çetin Kaya Koç
http://koclab.cs.ucsb.edu/teaching/cs4
cetinkoc@ucsb.edu

Outline

- Representations of integers
- Binary and decimal numbers
- Storing bits: Magnetic core and semiconductor memories
- Conversion between binary and decimal
- Hexadecimal representation
- Negative and positive integer representations
- Addition and subtraction
- Representing symbols and characters

Representation of Integers

- Natural (whole, counting) numbers: $\mathcal{N} = \{0, 1, 2, 3, \ldots\}$
- (Positive and negative) integers: $\mathcal{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$
- We represent integers in decimal for human convenience
- Decimal numbers have 10 unique digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Digit locations have weights which increase by a factor of 10, from right to left (from least significant to most significant):

$$\begin{array}{ccccc} \ldots & 10^3 & 10^2 & 10^1 & 10^0 \\ \ldots & 1000 & 100 & 10 & 1 \end{array}$$

- Examples:

$$\begin{array}{ccccccc} \ldots & 10^3 & 10^2 & 10^1 & 10^0 & & \\ \ldots & 2 & 0 & 1 & 5 & \rightarrow & 2015 \\ \ldots & 6 & 7 & 2 & 3 & \rightarrow & 6723 \end{array}$$

Binary Representation

- Two unique digits: 0 and 1
- Weights of digits double from right to left, starting with 1 (from least significant to most significant):

...	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
...	128	64	32	16	8	4	2	1

- Examples:

...	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		
...	0	0	0	0	1	0	1	0	$\rightarrow$	10
...	0	1	0	0	1	0	1	0	$\rightarrow$	74
...	1	0	0	0	1	0	1	0	$\rightarrow$	138

- A <u>bi</u>nary digi<u>t</u> is called bit

Storing Bits

- Decimal representation is for human convenience: we are used to it
- Computers represent and store all numbers in binary
- To store a single bit we need a 1-bit **memory**
- To store a longer number, say n bits, we need n copies of 1-bit **memory**, n-bit memory

Storing Bits

- Conceptually a 1-bit memory is a box capable of holding the value of the bit (which is either 1 or 0) as long as we need
- A 4-bit memory holding the binary number 1010 (in decimal, ten):

1	0	1	0

- We should be able to re-write another number on top of the existing one (thus erasing the old)
- For example, we can write 1001 (in decimal, nine) over 1001

1	0	0	1

- We lose the previous number completely

Memory

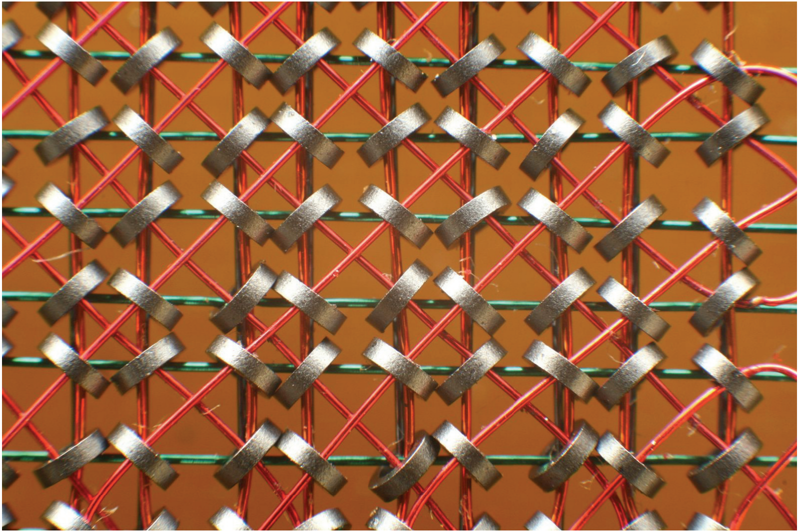
- There are different types of computer memory based on their speed, capacity, and the modes of use: registers, caches, main memory, hard disks, semiconductor drives, etc
- The speed and capacity is determined by the technology
- In today's technology the most common form of memory is based on semiconductor physics
- However, historically other memory technologies are used: electric relays, magnetic cores, etc

http://koclab.cs.ucsb.edu/teaching/cs4/docx/indian.mp4

Magnetic Core Memory

- For nearly 20 years (1955-1975) “magnetic core memory” was the predominant form of memory
- It uses tiny magnetic toroids (rings), the cores, through which wires are threaded to write and read information
- Each core represents one bit of information
- The cores can be magnetized in two different ways (clockwise or counterclockwise) and the bit stored in a core is 0 or 1 depending on that core’s magnetization direction
- The wires are arranged to allow an individual core to be set to either a 1 or a 0, and for its magnetization to be changed, by sending appropriate electric current pulses through selected wires

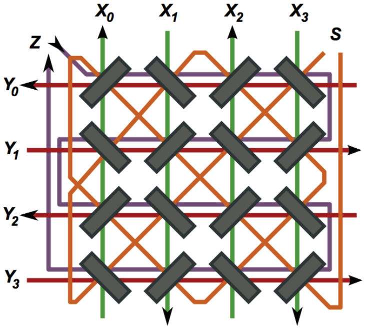
Magnetic Core Memory



Magnetic Core Memory

The distance between the rings is roughly 1 mm (0.04 in).

The green and brown wires (X and Y) are for selection. The sense wires are diagonal, colored orange, and the inhibit wires are vertical twisted pairs.

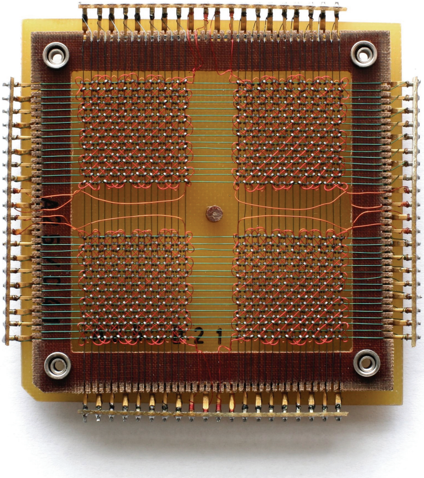


http://www.nzeldes.com/HOC/CoreMemory.htm

Magnetic Core Memory

$4 \times 16 \times 16$ bits $= 1024$ bits
1024 bits is 1K (Kilo) bits

Various logical organizations:
$1 \times 32 \times 32$
$2 \times 16 \times 32$
$4 \times 2 \times 8 \times 16$
1×1024
2×512
4×256



Magnetic Core Memory Size

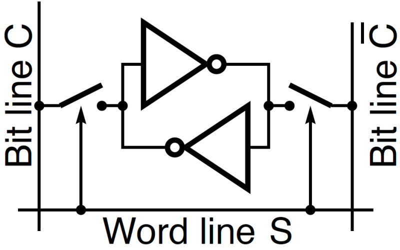
- A 1-bit magnetic core takes approximately 1 mm2 space
- The 1024-bit core would be about 1024 mm2 $\approx$ 10 cm2
- Your iPhone may have 16G (Giga) bytes, which is $16 \times 8 \times 2^{30}$ bits
- In core memory, this means an area of $128 \times 2^{20} \times 10$ cm2
- This is equivalent to 134,217 m2
- A football field is about 110m by 48m, which is 5,280 m2
- 16 Gigabytes of magnetic core memory takes the space of 25 fields!

- iPhone with magnetic core memory would be unthinkable! :)

5 MByte IBM Hard Drive in 1956

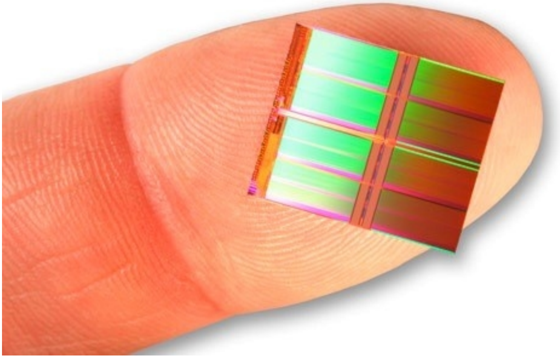
Semiconductor Memory Size

- Current memory technology is based on semiconductors
- The geometry of 1-bit memory is measured by nanometers (10^{-9}m)
- The main concept of 1-bit (CMOS) memory:

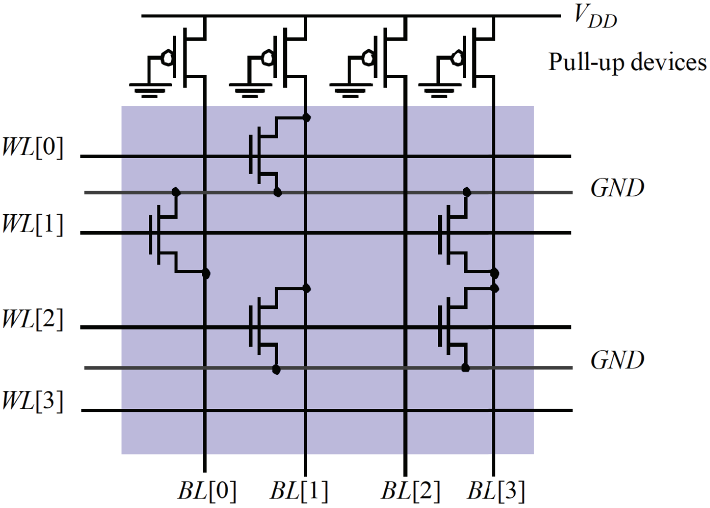


- The size of memory chip holding several gigabits is only a few mm2

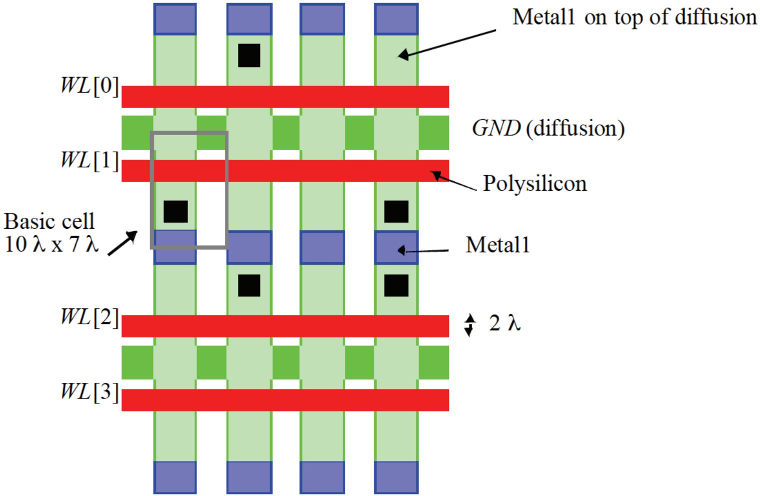
128 GBit Flash Chip 2012



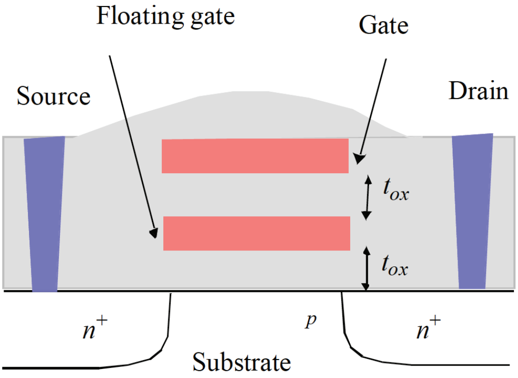
Semiconductor Memory - Logical View



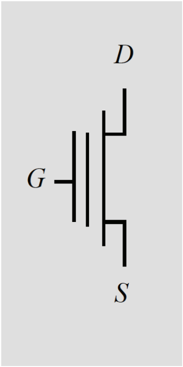
Semiconductor Memory - Chip Mask



Semiconductor Memory - Cross-Section View

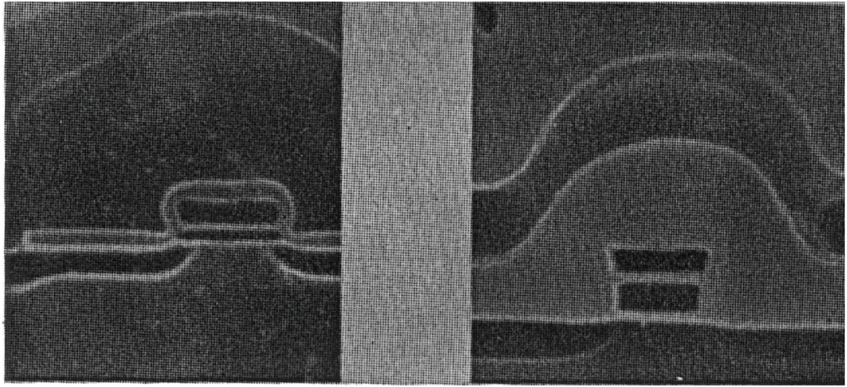


(a) Device cross-section



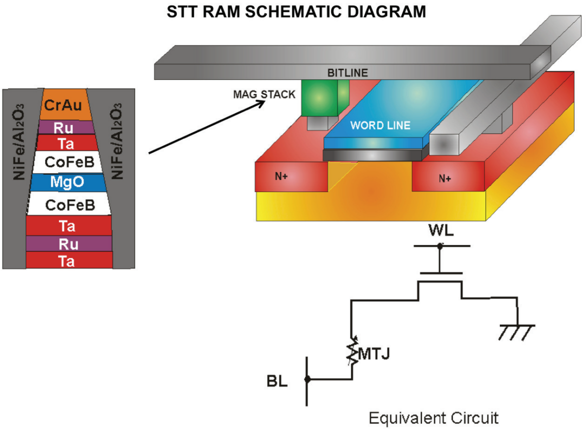
(b) Schematic symbol

Semiconductor Memory - Physical Device

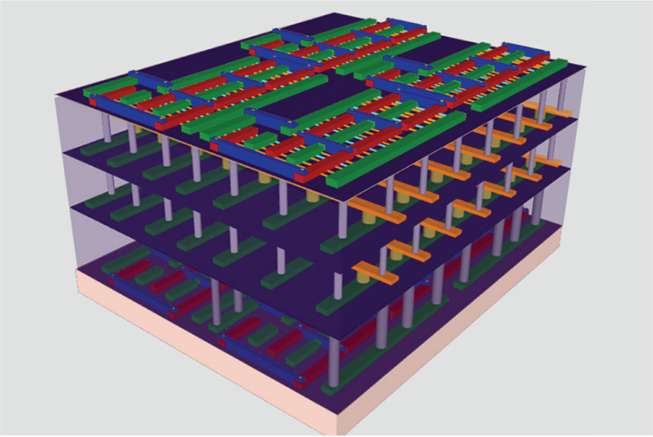


Flash *Courtesy Intel* **EPROM**

Semiconductor Memory - Future Devices



Semiconductor Memory - Future Devices



Binary and Decimal Representation

- Decimal representation is for human convenience
- It is hard to read binary numbers on paper: 11111011111 ?
- If represented in decimal, it looks familiar: 2015
- Sometimes we need to convert binary numbers to decimal or vice versa

Binary2Decimal Conversion

- Sum the powers of 2 for which the bit is 1, ignore others
- $(0000\ 1010) = 8 + 2 = 10$
- $(0100\ 1010) = 64 + 8 + 2 = 74$
- $(1000\ 1010) = 128 + 8 + 2 = 138$

Decimal2Binary Conversion

- Subtract the largest power of 2 (keeping the remainder positive) from the number, and set the corresponding bit to 1
- The other bits are zero
- For example, if the number is 138, the largest power of 2 that is closest to 138 would be $128 = 2^7$, and thus:
- $138 - 2^7 = 10 \quad \rightarrow$ the bit 7 is set to 1
- $10 - 2^3 = 2 \quad \rightarrow$ the bit 3 is set to 1
- $2 - 2^1 = 0 \quad \rightarrow$ the bit 1 is set to 1

$$\begin{array}{cccccccccc} & & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 138 & \rightarrow & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{array}$$

Hexadecimal Representation

- A convenient "on-paper" representation of binary numbers
- The binary number is grouped into 4 bits, starting from right
- If the number of bits is not a multiple of 4, leftmost 0s are added
- 2015 $\rightarrow$ 11111011111 $\rightarrow$ 0111 1101 1111
- Each 4-bit group is viewed as a digit, and represented using a separate symbol
- Since there are $2^4 = 16$ distinct numbers, we need 16 symbols
- We could select any set of 16 distinct symbols: $\alpha, \beta, \gamma, \delta, \ldots$
- However, there is a conventional (and more logical) method
- We use decimal digits for the first ten and the first six letters of the alphabet for the rest

Hexadecimal Representation

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Therefore, 2015 would be represented as

$$(2015)_{decimal} = (0111\ 1101\ 1111)_{binary} = (7DF)_{hex}$$

Negative Numbers

- So far we considered only positive numbers
- In fact, we did not think about the sign at all
- All numbers we have seen so far are **unsigned** integers
- To represent signed numbers, we need to reserve a separate bit for the sign, and also treat the rest of the bits somewhat differently
- There are 3 methods: Sign-Magnitude, One's-Complement, and Two's-Complement
- These representations allow both the negative and positive numbers in a unified way
- We will only study Two's-Complement which is predominantly used

Two's-Complement Representation

- First we fix the number of bits to a particular value, such as 4
- We use all 4 bits to represent a number even if it does not need it
- The leftmost bit is the sign of the number, positive or negative
- 0 stands for positive and 1 stands for negative
- The negative of a number is obtained by taking the bitwise complement of the number and then adding 1 to it
- For example, $+6 = (0110)$ implies $-6 = (1001) + (0001) = (1010)$

4-bit Two's-Complement Representation

0000	$\rightarrow$	+0	1000	$\rightarrow$	-8
0001	$\rightarrow$	+1	1001	$\rightarrow$	-7
0010	$\rightarrow$	+2	1010	$\rightarrow$	-6
0011	$\rightarrow$	+3	1011	$\rightarrow$	-5
0100	$\rightarrow$	+4	1100	$\rightarrow$	-4
0101	$\rightarrow$	+5	1101	$\rightarrow$	-3
0110	$\rightarrow$	+6	1110	$\rightarrow$	-2
0111	$\rightarrow$	+7	1111	$\rightarrow$	-1

Negation in Two's-Complement Representation

- Given a positive integer x, how do we find $-x$?
- For example, given $3 = (0011)$, how do we obtain -3?
- Algorithm: Complement every bit and add 1
- $0011 \rightarrow 1100 \rightarrow 1100 + 1 \rightarrow 1101$

- Similarly, given $-4 = (1100)$, how do we obtain $+4$?
- $1100 \rightarrow 0011 \rightarrow 0011 + 1 \rightarrow 0100$

Addition in Two's-Complement Representation

- Add the numbers (including their signs) and ignore the carry out
- For example: (-2)+(+7)=(1110)+(0111) is performed as

(1)	(1)	(1)	(0)		carry bits
	1	1	1	0	(-2)
	0	1	1	1	$(+7)$
	0	1	0	1	$(+5)$

Subtraction in Two's-Complement Representation

- Subtraction is performed by negating the second operand and then performing addition: (+5)-(+7) = (+5)+(-7)
- We already learned the negation algorithm: Complement every bit of the number and then adding 1 to it
- Since $x = +7 = (0111)$, then $-x = (1000) + 1 = (1001) = -7$
- Now, add the numbers (including their signs) and ignore the carry out
- (+5)+(-7)=(0101)+(1001) is performed as

(1)	(0)	(0)	(1)		carry bits
	0	1	0	1	$(+5)$
	1	0	0	1	(-7)
	1	1	1	0	(-2)

Overflow

- If the result is beyond the range of numbers representable, there will be overflow (in all 3 representations)
- In 4-bit Two's-Complement we can represent only $\{-8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7\}$
- If we attempt to add 4 and 5 in 4-bit Two's-Complement representation, the result (which is 9) is not representable, and thus we will not be able to obtain the correct sum

$$\begin{array}{ccccc|l} (0) & (1) & (0) & (0) & & \text{carry bits} \\ & 0 & 1 & 0 & 0 & (+4) \\ & 0 & 1 & 0 & 1 & (+5) \\ \hline & 1 & 0 & 0 & 1 & (-7)? \end{array}$$

- Solution: Allow more bits

Representing Symbols and Characters

- Representation method of integers should take arithmetic operations into account
- In general, any representation method of a particular class of objects should mind the use of objects in a computational setting or their relationship to one another
- In some cases (such as integers), the computational requirements are very important
- In some other cases (such as characters, letters, text), there are fewer or less stringent computational requirements
- We just need to assign a “code” to the symbol (character or letter) and remember it when we need it