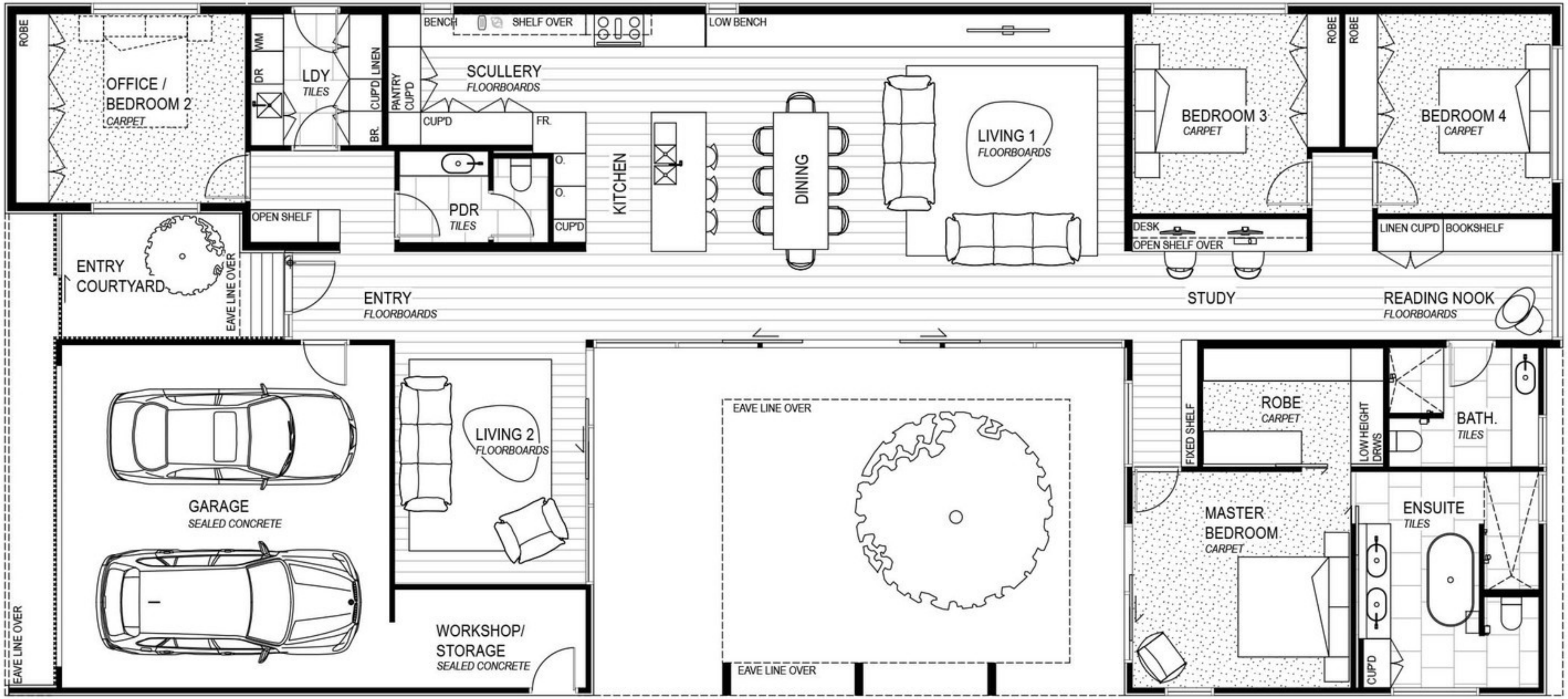
Computer architecture & Digital number systems

ET2223 Microprocessors, Microcontrollers, and Embedded Systems

Partially based on
Computer Architecture and Organization by Dr. Senaka Amarakeerthi, University of Sri Jayewardenepura
Number Systems by Dr. Paul Beame, University of Washington
Number Systems: Negative Numbers by Dr. Chung-Kuan Cheng, UC San Diego

Computer architecture

What is computer architecture?



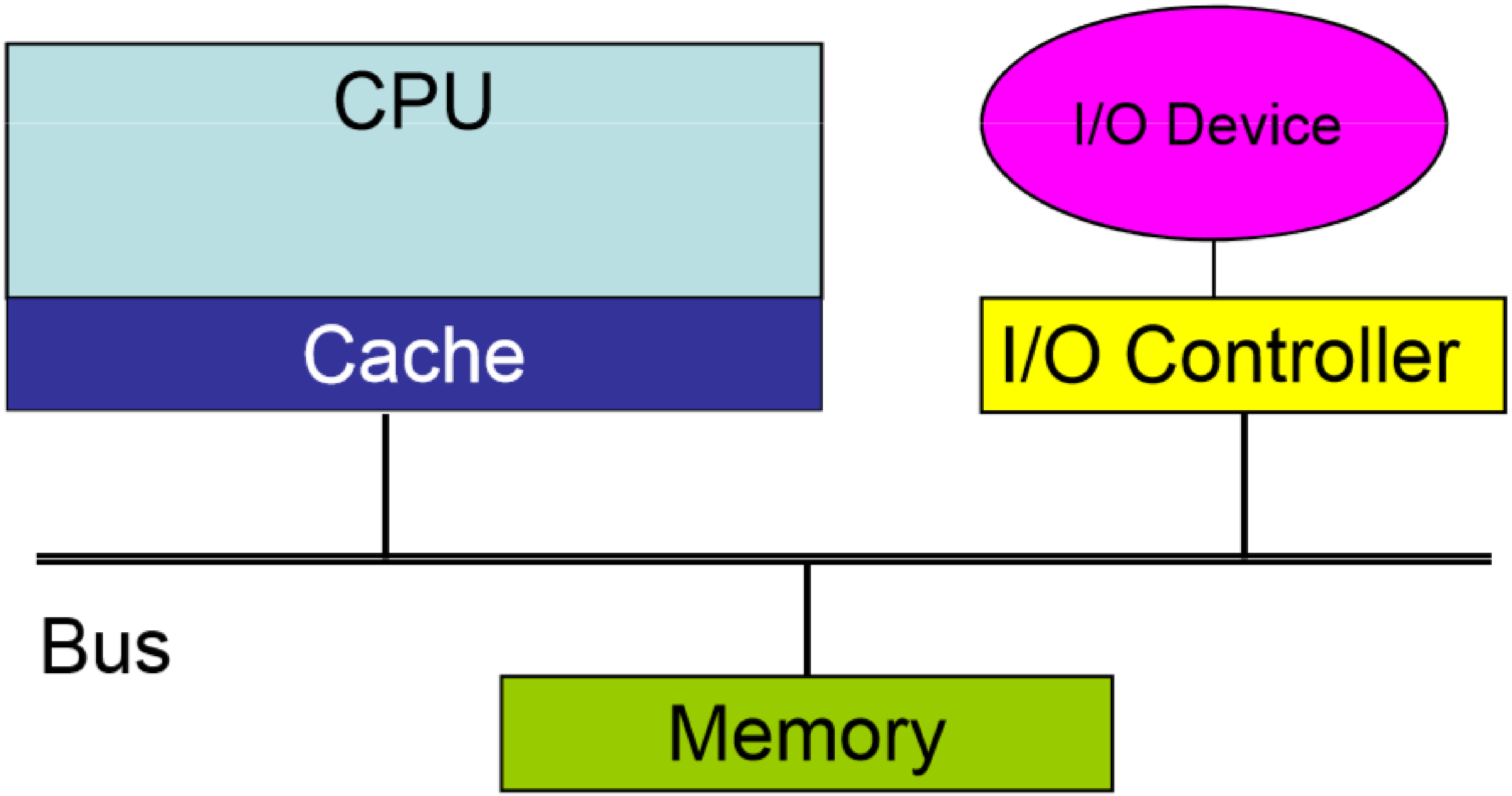
Architecture and organization

- Architecture is the design of the system visible to the assembly level programmer.
  - What instructions
  - How many registers
  - Memory addressing scheme
- Organization is how the architecture is implemented.
  - How much cache memory
  - Microcode or direct hardware
  - Implementation technology

Same architecture, different organization

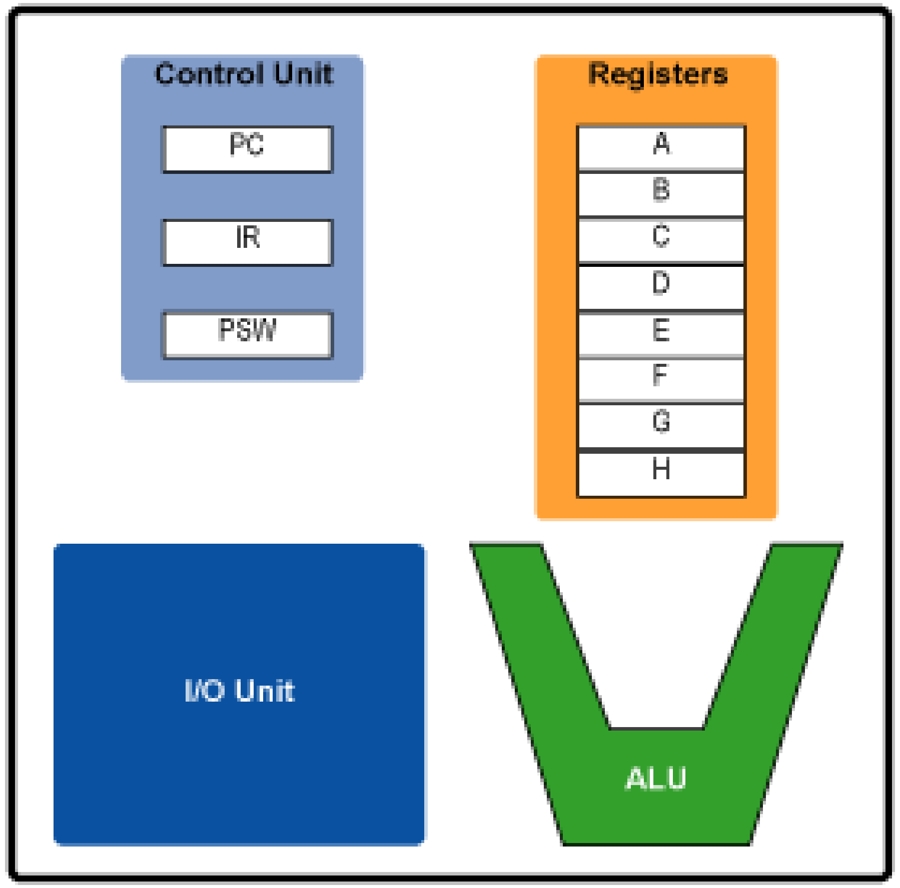
- Almost every program that can run on a Core i3 can run on a Core i5.
- All computers in the Intel Core series have the same architecture.
- Each version of the Intel Core has a different organization or implementation, speed, and price.

Basic computer components



Central Processing Unit

- Contains the control logic that initiates most activities in the computer.
- The Arithmetic Logic Units perform the math and logic calculations.
- Registers contain temporary data values.
- Program Counter contains the address of the next instruction to execute.

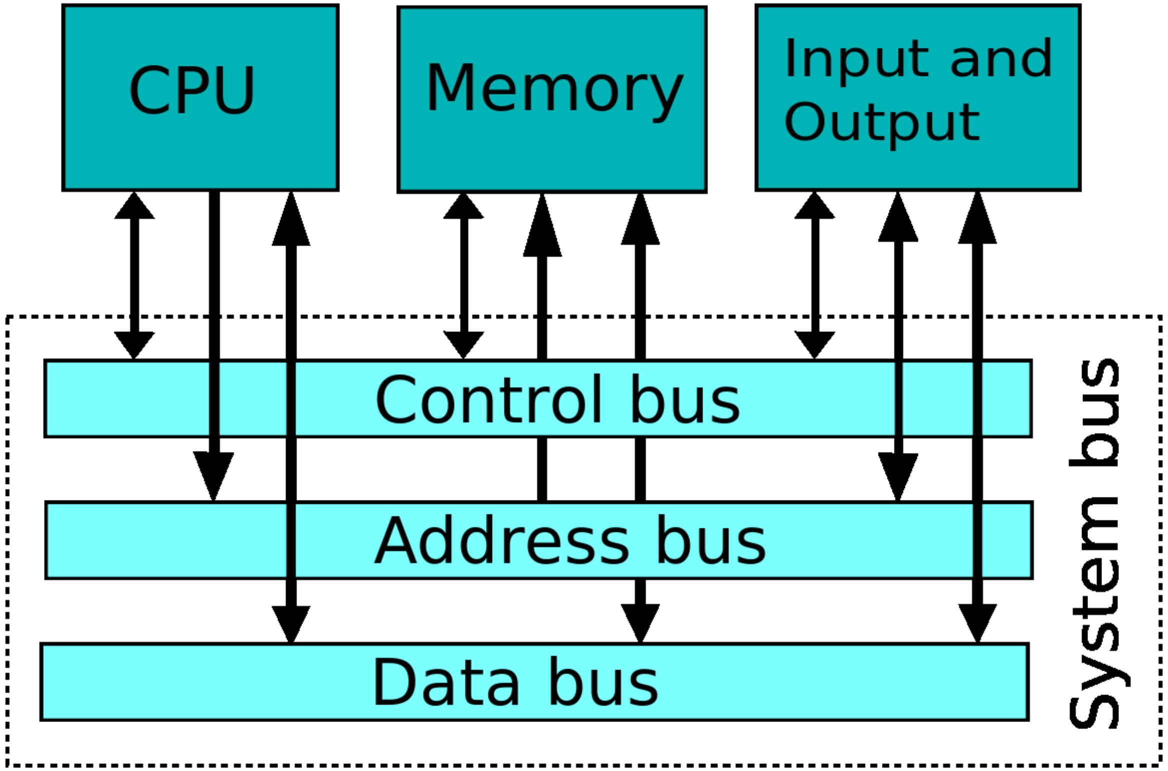


Registers

- The CPU has registers to temporarily hold data being acted upon.
- Different architectures have different number of registers.
- Some registers are available for the user programs to use directly.
- Some registers are used indirectly (such as the program counter).
- Some registers are used only by the operating system (i.e. program status reg)

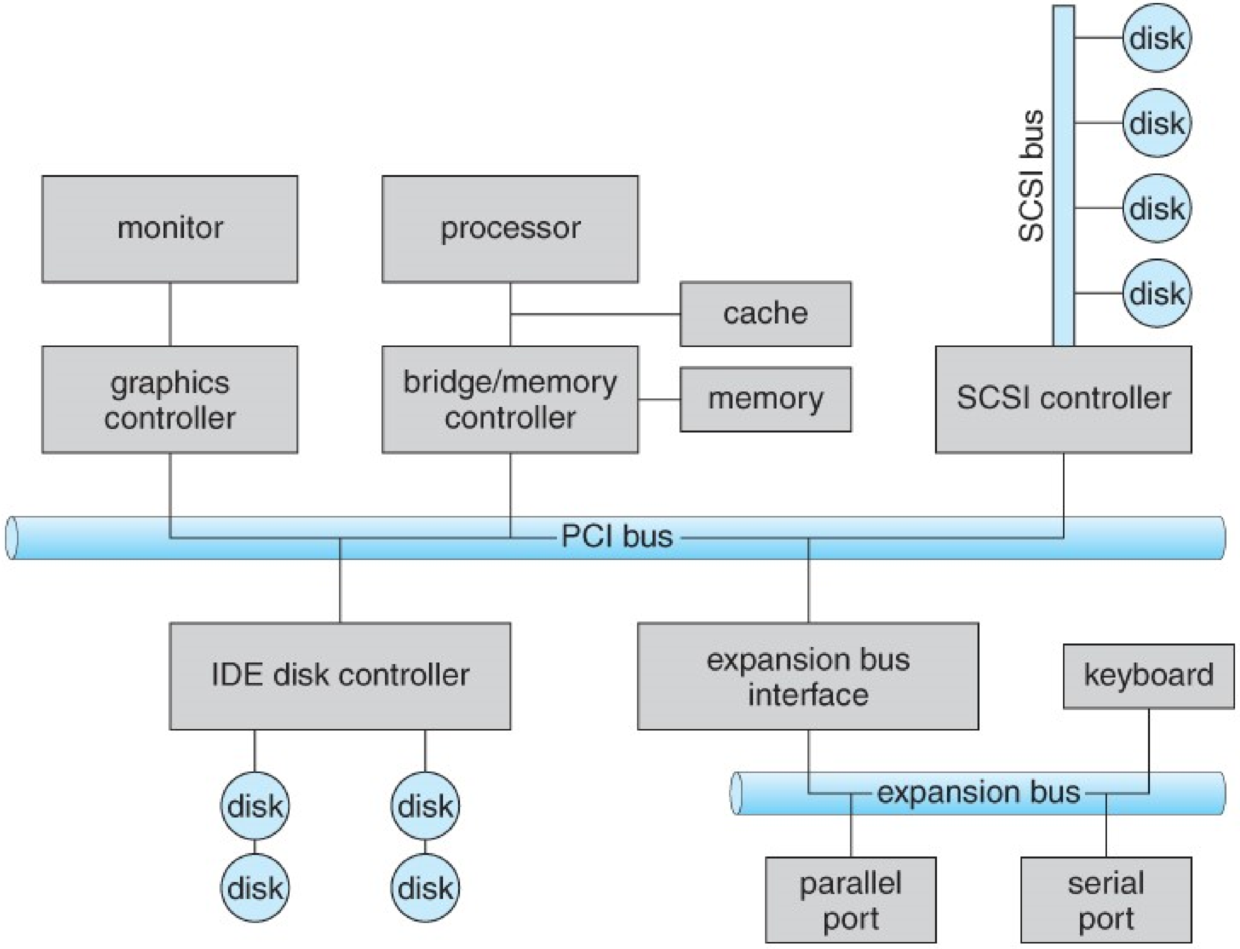
Bus

- The bus is a set of parallel wires that connect the CPU, memory and I/O controllers.
- It has logic (the chipset) to determine who can use the bus at any given instant.
- The width of the bus determines the maximum memory configuration



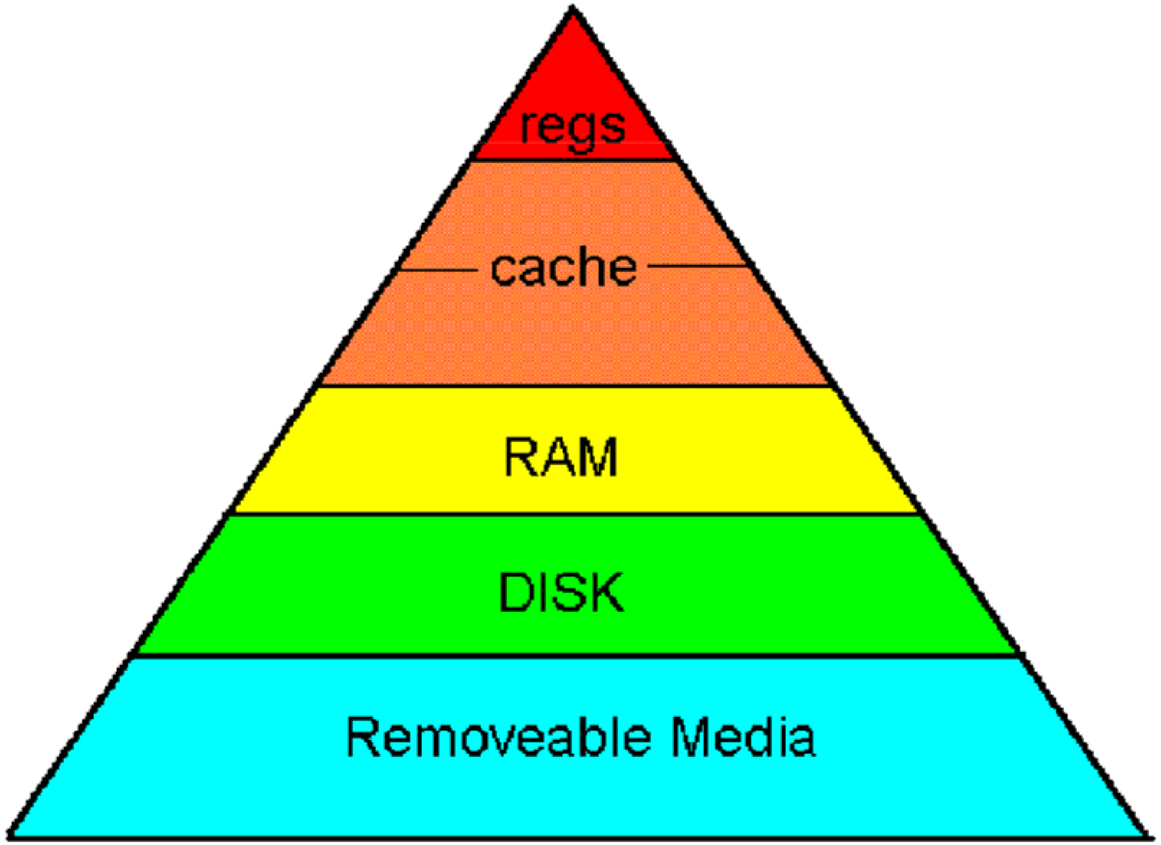
I/O controllers

- Direct the flow of data to and from I/O devices.
- CPU sends a request to the I/O controller to initiate I/O.
- I/O controllers run independently and in parallel with the CPU
- I/O controllers may interrupt the CPU upon completion of request or error.



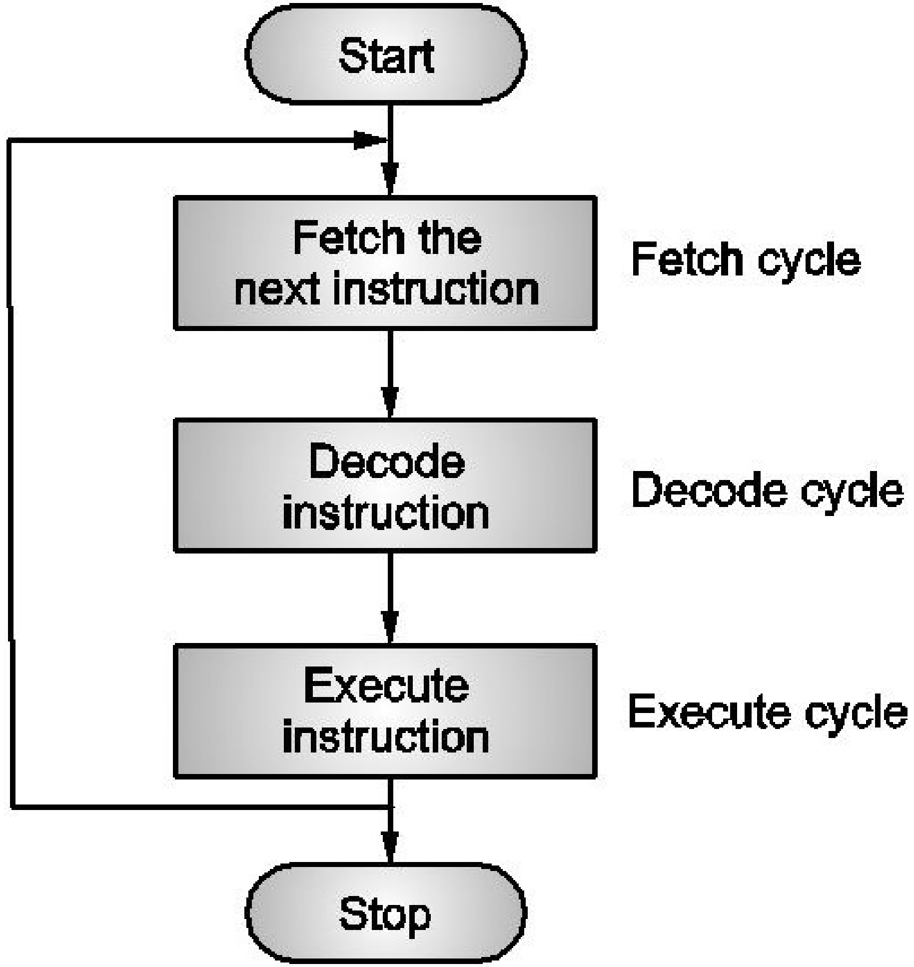
Memory hierarchy

- The internal memory is Random Access Memory (RAM).
- Both data and program instructions are kept in RAM.
- Instructions must be in RAM to be executed.



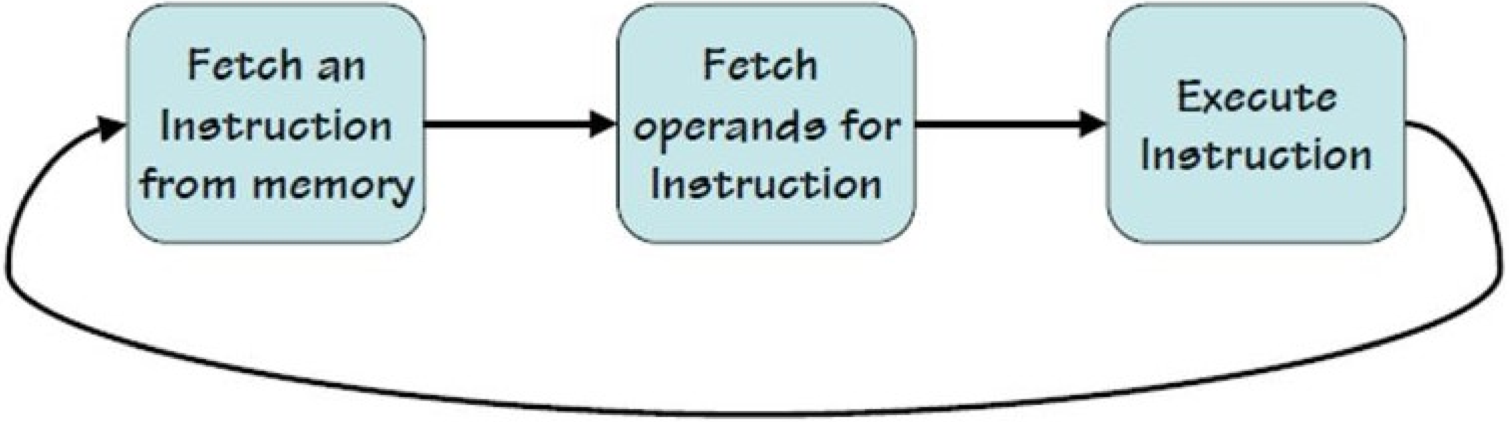
Instruction cycle

- Fetching the instruction from memory and executing the instruction
  1. Fetch the instruction from the memory address in the Program Counter register
  2. Increment the Program Counter
  3. Decode the type of instruction
  4. Fetch the operands
  5. Execute the instruction
  6. Store the results



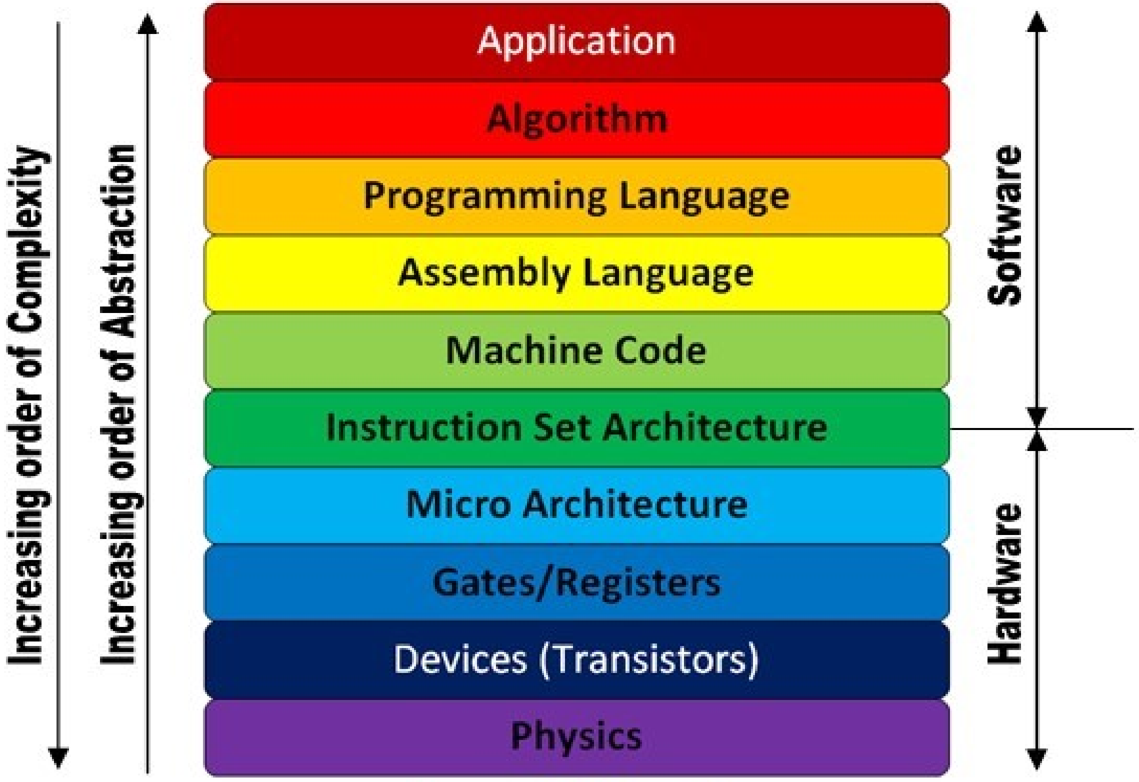
Simple model of execution

- Instruction sequence is determined by a simple conceptual control point.
- Each instruction is completed before the next instruction starts.
- One instruction is executed at a time.



Layers

- You can consider computer operation at many different levels.
  - Applications
  - Middleware
  - High level languages
  - Machine Language
  - Microcode
  - Logic circuits
  - Gates
  - Transistors
  - Silicon structures



Layers of Abstraction

Digital number systems

Digital

- Digital = discrete
  - Binary codes (example: Binary Coded Decimal)
- Binary codes
  - Represent symbols using binary digits (bits)
- Digital computers:
  - I/O is digital
    - ASCII, decimal, etc.
  - Internal representation is binary
    - Process information in bits

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

Digital numbers

- Binary numbers
  - Computers work with 0’s and 1’s; binary is like the alphabets of a language
- Base conversion
  - For convenience, people use other bases (like decimal, hexdecimal)
  - Need to know how to convert from one to another
- Number systems
  - There are more than one way to express a number in binary.
  - So 1010 could be 10, -2, -5 or -6 and need to know which one.
- A/D and D/A conversion
  - Real world signals come in continuous/analog format
  - It is good to know how they become 0’s and 1’s (and vice versa)

The basics: Binary numbers

- Bases we will use
  - Binary: Base 2
    - 0,1
  - Octal: Base 8
    - 0,1,2,3,4,5,6,7
  - Decimal: Base 10
    - 0,1,2,3,4,5,6,7,8,9
  - Hexadecimal: Base 16
    - 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- Positional number system
  - $101_2 = 1\times2^2 + 0\times2^1 + 1\times2^0$
  - $63_8 = 6\times8^1 + 3\times8^0$
  - $A1_{16} = 10\times16^1 + 1\times16^0$
- Addition and subtraction

$$\begin{array}{r} 1011 \\ +\ 1010 \\ \hline 10101 \end{array} \qquad \begin{array}{r} 1011 \\ -\ 0110 \\ \hline 0101 \end{array}$$

Binary → hex/decimal/octal conversion

- Conversion from binary to octal/hex
  - Binary : 10011110001
  - Octal : 10 | 011 | 110 | 001=2361_8
  - Hex : 100 | 1111 | 0001=$4F1_{16}$
- Conversion from binary to decimal
  - $101_2 = 1\times2^2 + 0\times2^1 + 1\times2^0 = 5_{10}$
  - $63.4_8 = 6\times8^1 + 3\times8^0 + 4\times8^{-1} = 51.5_{10}$
  - $A1_{16} = 10\times16^1 + 1\times16^0 = 161_{10}$

Decimal→ binary/octal/hex conversion

Binary

	Quotient	Remainder
56÷2=	28	0
28÷2=	14	0
14÷2=	7	0
7÷2=	3	1
3÷2=	1	1
1÷2=	0	1

Octal

	Quotient	Remainder
56÷8=	7	0
7÷8=	0	7

$56_{10}=111000_2$

$56_{10}=70_8$

- Why does this work?
  - $N=56_{10}=111000_2$
  - Q=N/2=56/2=111000/2=11100 remainder 0
- Each successive divide liberates an LSB (least significant bit)

Number systems

- How do we write negative binary numbers?
- Historically: 3 approaches
  - Sign-and-magnitude
  - Ones-complement
  - Twos-complement
- For all 3, the most-significant bit (MSB) is the sign digit
  - 0 ≡ positive
  - 1 ≡ negative
- Twos-complement is the important one
  - Simplifies arithmetic
  - Used almost universally

Sign-and-magnitude

- The most-significant bit (MSB) is the sign digit
  - 0 ≡ positive
  - 1 ≡ negative
- The remaining bits are the number's magnitude
- Problem 1: Two representations for zero
  - 0 = 0000 and also –0 = 1000
- Problem 2: Arithmetic is cumbersome

Add		Subtract			Compare and subtract		
4 + 3	0100 + 0011	4 – 3	0100 + 1011	0100 – 0011	– 4 + 3	1100 + 0011	1100 – 0011
= 7	= 0111	= 1	≠ 1111	= 0001	– 1	≠ 1111	= 1001

Ones-complement & Twos-complement

- Signed system: Simple. Just flip the sign bit
    - 0 = positive
    - 1 = negative
- One’s complement: Replace subtraction with addition
  - Easy to derive (Just flip every bit)
- Two’s complement: Replace subtraction with addition
  - Addition of one’s complement and one produces the two’s complement.

For positive integer x, represent -x

- 1's complement:
  - Formula: $2^n - 1 - x$
    - i.e. n=4, $2^4 - 1 - x = 15 - x$
    - In binary: $(1\ 1\ 1\ 1) - (b_3\ b_2\ b_1\ b_0)$
    - Just flip all the bits.
- 2's complement:
  - Formula: $2^n - x$
    - i.e. n=4, $2^4 - x = 16 - x$
    - In binary: $(1\ 0\ 0\ 0\ 0) - (0\ \ b_3\ b_2\ b_1\ b_0)$
    - Just flip all the bits and add 1.

Ones-complement

- Negative number: Bitwise complement positive number
  - 0111 ≡ 7_{10}
  - 1000 ≡ $–7_{10}$
- Solves the arithmetic problem

Add		Invert, add, add carry		Invert and add	
4	0100	4	0100	– 4	1011
+ 3	+ 0011	– 3	+ 1100	+ 3	+ 0011
= 7	= 0111	= 1	1 0000	– 1	1110
		add carry:	+1		
			= 0001		

- Remaining problem: Two representations for zero
  - 0 = 0000 and also –0 = 1111

Why ones-complement works?

- The ones-complement of an 8-bit positive y is 11111111_2 – y
- What is 11111111_2 ?
  - 1 less than 1 00000000_2 ≡ 28 ≡ 25610
  - So in ones-complement –y is represented by (28 -1) – y
- Adding representations of x and –y where x, y are positive: we get (28 -1) + x – y
  - If x < y then x - y < 0 there is no carry and get –ve number
    - Just add the representations if no carry
  - If x > y then x - y > 0 there is a carry and get +ve number
    - Need to add 1 and ignore the 28, i.e. “add the carry”
  - If x = y then answer should be 0, get 28-1 =11111111_2

Arithmetic Operations: 1's Complement

Input: two positive integers x & y,

1. We represent the operands in one's complement.
2. We sum up the two operands.
3. We delete 2^n-1 if there is carry out at left.
4. The result is the solution in one's complement.

Arithmetic	1's complement
$x + y$	$x + y$
$x - y$	$x + (2^n - 1 - y) = 2^n - 1 + (x - y)$
$-x + y$	$(2^n - 1 - x) + y = 2^n - 1 + (-x + y)$
$-x - y$	$(2^n - 1 - x) + (2^n - 1 - y) = 2^n - 1 + (2^n - 1 - x - y)$

Arithmetic Operations: Example: 4 – 3 = 1

$4_{10} = 0100_2$

$3_{10} = 0011_2$ $-3_{10} \rightarrow 1100_2$ in one's complement

  0100 (4 in decimal)
+ 1100 (12 in decimal or 15-3)
1,0000 (16 in decimal or 15+1)
  0001(after deleting 2^n-1)

We discard the extra 1 at the left which is 2^n and add one at the first bit.

Arithmetic Operations: Example: -4 +3 = -1

$4_{10} = 0100_2$ $-4_{10} \rightarrow$ Using one's comp. $\rightarrow 1011_2$

(Invert bits)

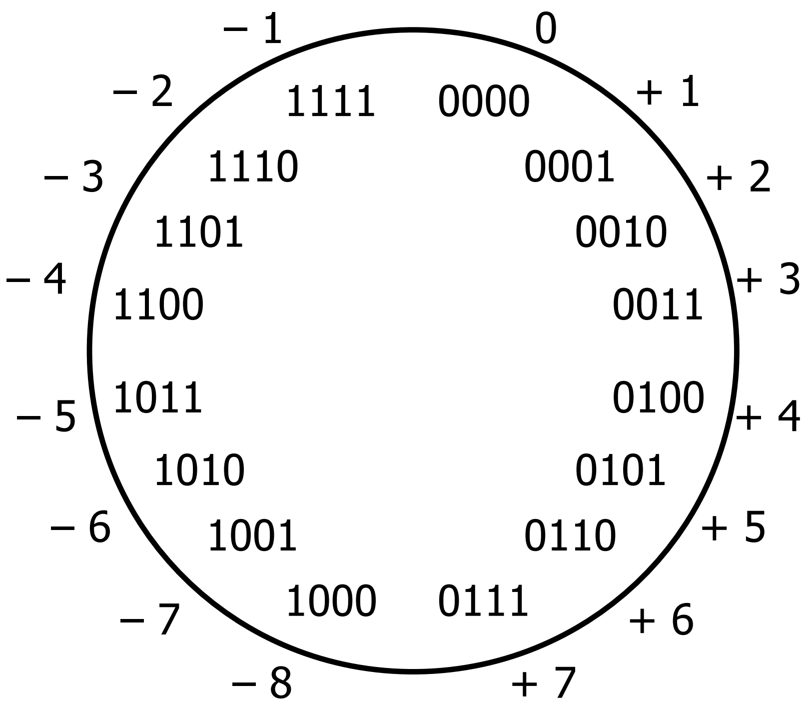
$3_{10} = 0011_2$

  1011 (11 in decimal or 15-4)
+ 0011 (3 in decimal)
  1110 (14 in decimal or 15-1)

If the left-most bit is 1, it means that we have a negative number.

Twos-complement

- Negative number: Bitwise complement plus one
  - $0111 \equiv 7_{10}$
  - $1001 \equiv -7_{10}$
- Number wheel
  - Only one zero!
  - MSB is the sign digit
  - 0 ≡ positive
  - 1 ≡ negative



Twos-complement

- Complementing a complement the original number
- Arithmetic is easy
  - Subtraction = negation and addition
  - Easy to implement in hardware

Add		Invert and add		Invert and add	
4	0100	4	0100	– 4	1100
+ 3	+ 0011	– 3	+ 1101	+ 3	+ 0011
= 7	= 0111	= 1	1 0001	– 1	1111
		drop carry	= 0001		

Why twos-complement works better

- Recall:
  - The ones-complement of a b-bit positive y is $(2^b-1) – y$
- Adding 1 to get the twos-complement represents $–y$ by $2^b – y$
  - So $-y$ and $2^b – y$ are equal mod 2^b
    (leave the same remainder when divided by 2^b)
  - Ignoring carries is equivalent to doing arithmetic mod 2^b
- Adding representations of x and $–y$ yields $2^b + x – y$
  - If there is a carry then that means $x \geq y$ and dropping the carry yields $x-y$
  - If there is no carry then $x < y$ and then we can think of it as $2^b – (y-x)$

Arithmetic Operations: 2’s Complement

Input: two positive integers x & y,

1. We represent the operands in two’s complement.
2. We sum up the two operands and ignore bit n.
3. The result is the solution in two’s complement.

Arithmetic	2’s complement
$x + y$	$x + y$
$x - y$	$x + (2^n - y) = 2^n+(x-y)$
$-x + y$	$(2^n - x) + y =2^n+(-x+y)$
$-x - y$	$(2^n - x) + (2^n - y) = 2^n+2^n-x-y$

Arithmetic Operations: Example: 4 – 3 = 1

$4_{10} = 0100_2$

$3_{10} = 0011_2$ $\quad$ $-3_{10} \rightarrow 1101_2$

$$
\begin{array}{r}
0100 \\
\underline{+\ 1101} \\
{\color{red}1}0001
\end{array}
\rightarrow 1 \text{ (after discarding extra bit)}
$$

We discard the extra 1 at the left which is 2^n from 2's complement of -3. Note that bit b_{n-1} is 0. Thus, the result is positive.

Arithmetic Operations: Example: -4 +3 = -1

$4_{10} = 0100_2$ $-4_{10} \rightarrow$ Using two's comp. $\rightarrow$ $1011 + 1 = 1100_2$

(Invert bits)

$3_{10} = 0011_2$

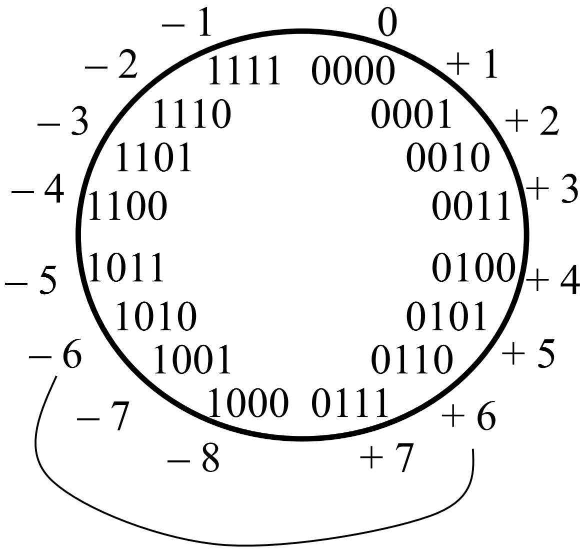
$$\begin{array}{r} 1100 \\ \underline{+\ 0011} \\ 1111 \end{array}$$

$\rightarrow$ Using two's comp. $\rightarrow$ 0000 + 1 = 1, so our answer is -1

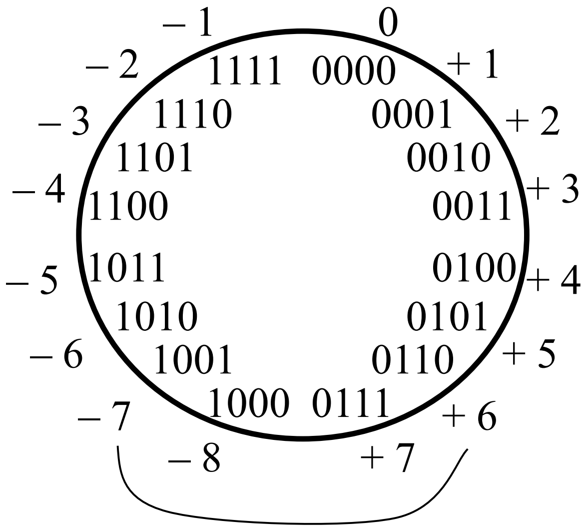
If left-most bit is 1, it means that we have a negative number.

Twos-complement overflow

- Answers only correct mod 2b
  - Summing two positive numbers can give negative result
  - Summing two negative numbers can give a positive result



$6 + 4 \Rightarrow -6$



$-7 - 3 \Rightarrow +6$

Miscellaneous

- Twos-complement of non-integers
  - $1.6875_{10} = 01.1011_2$
  - $-1.6875_{10} = 10.0101_2$
- Sign extension
  - Write +6 and –6 as 2's complement
    - 0110 and 1010
  - Sign extend to 8-bit bytes
    - 00000110 and 11111010

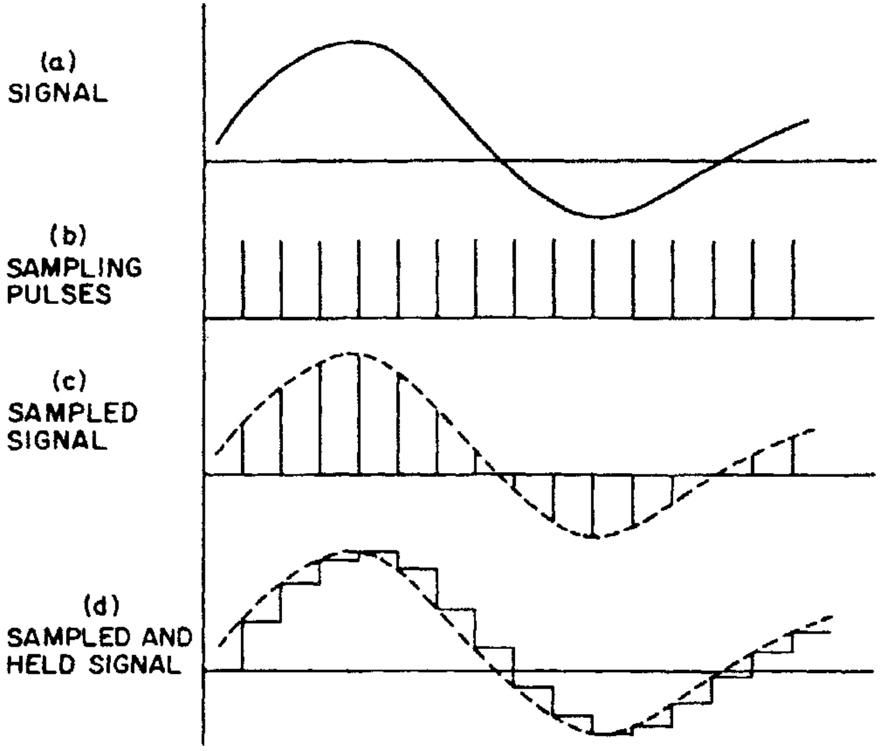
- Can't infer a representation from a number
  - 11001 is 25 (unsigned)
  - 11001 is –9 (sign magnitude)
  - 11001 is –6 (ones complement)
  - 11001 is –7 (twos complement)

Digital and analog

- The physical world is analog
- Digital systems need to
  - Measure analog quantities
    - Speech waveforms, etc
  - Control analog systems
    - Drive motors, etc
- How do we connect the analog and digital domains?
  - Analog-to-digital converter (A/D)
    - Example: CD recording
  - Digital-to-analog converter (D/A)
    - Example: CD playback

Sampling

- Quantization
  - Conversion from analog to discrete
- Quantizing a signal
  - We sample it

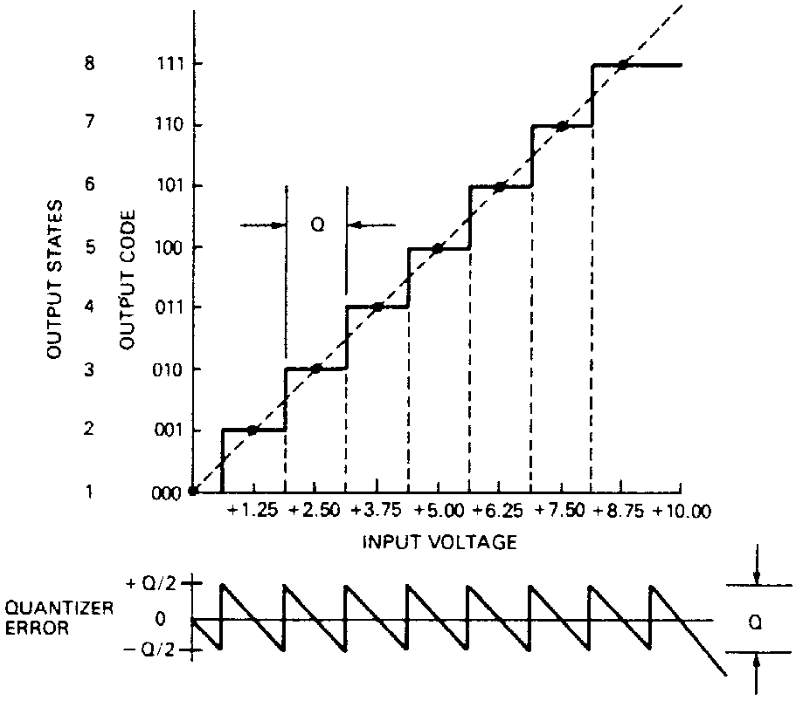


Signal Sampling

Datel Data Acquisition and
Conversion Handbook

Conversion

- Encoding
  - Assigning a digital word to each discret
- Encoding a quantized signal
  - Encode the samples
  - Typically Gray or binary codes



Transfer Function of Ideal 3-Bit Quantizer

Datel Data Acquisition and Conversion Handbook