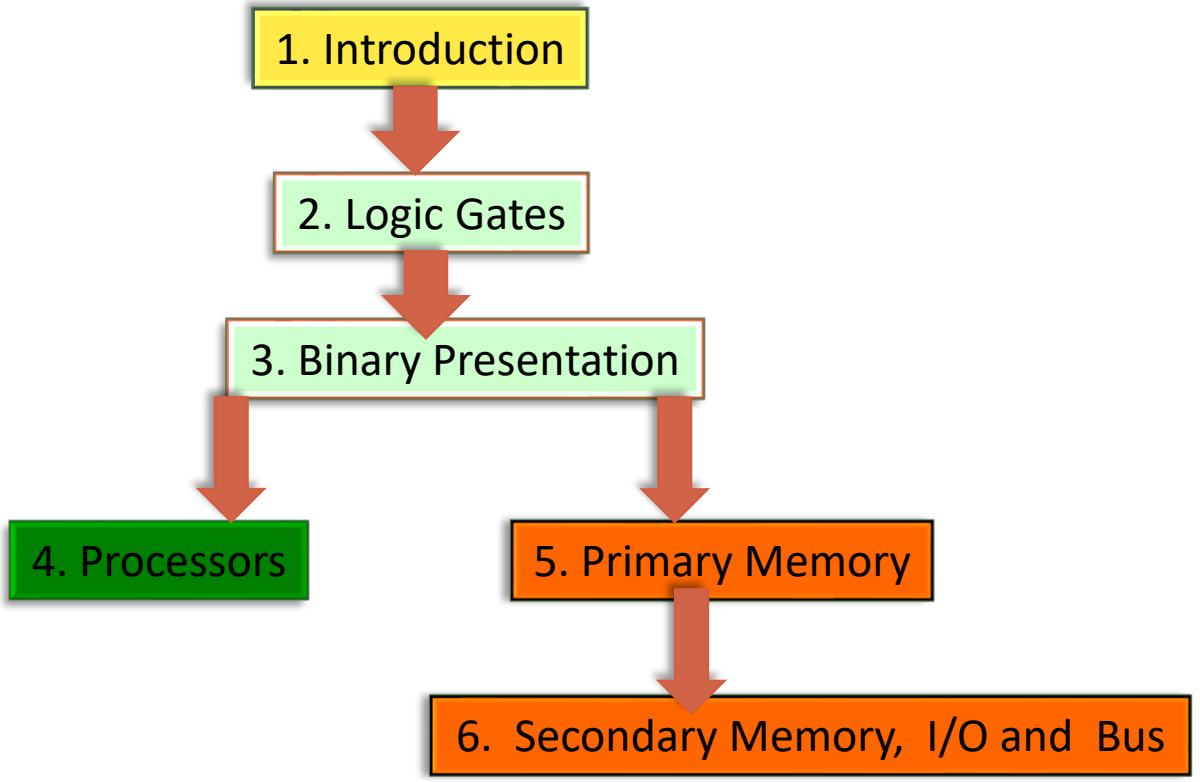
CM1205

Architecture and
Operating Systems

Part 1 Structure



"... the cost of hardware has halved every year whilst its complexity has, on average, quadrupled every three years ... the increase in complexity and decrease in cost are both of the order of one million since the mid 1960s. Equivalent progress in, for example, the motor industry, would have provided us with luxury cars requiring only half a gallon of petrol for life, having sufficient thrust to go into orbit, and a price tag of about twenty pence."

(Professor Stonham, 1987, "Digital Logic Design")

Thought For The Day

Fill In

Smaller, Faster & Cheaper

- The trend is for smaller computers, which run faster and cost less

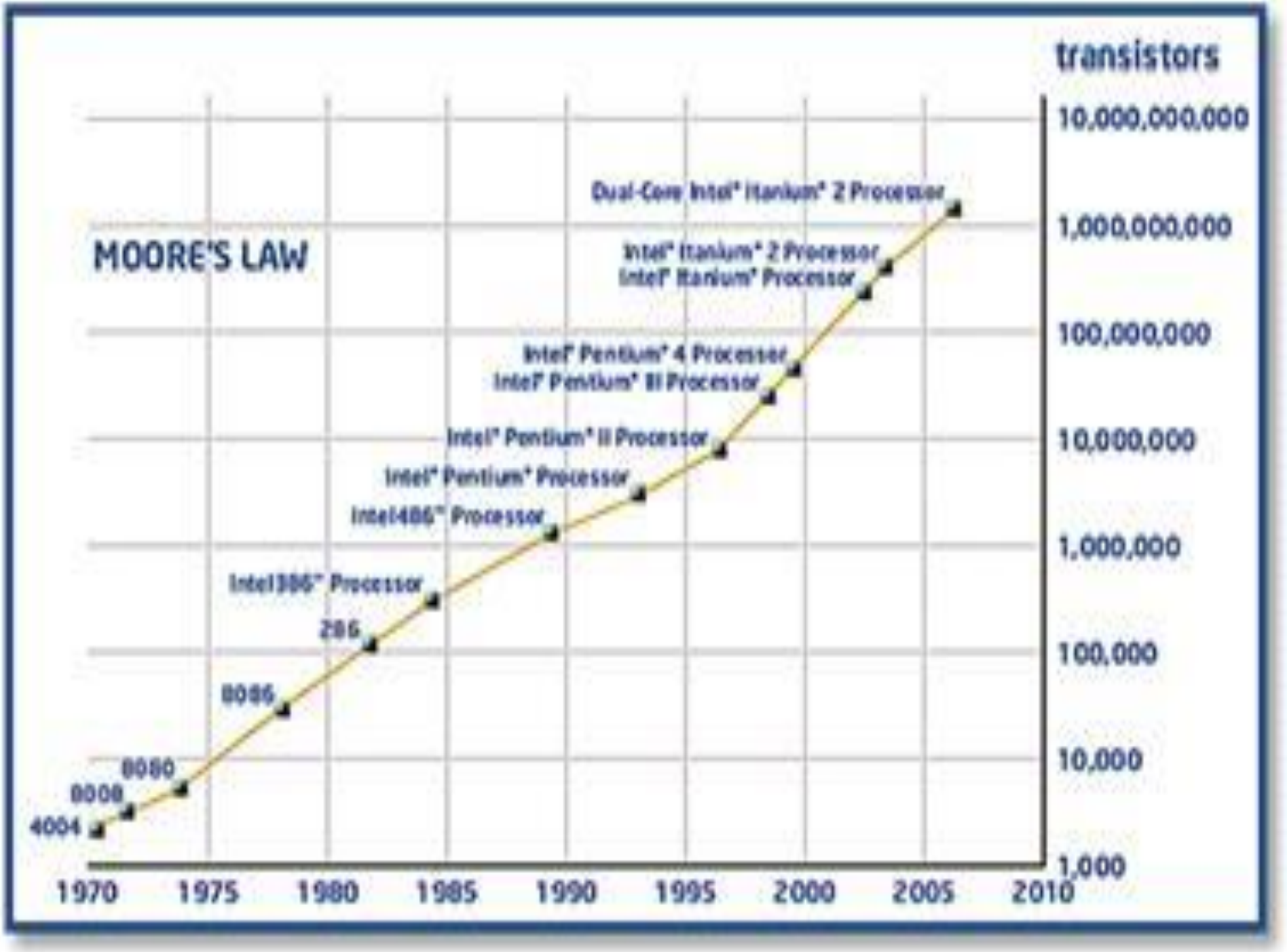
  Q: Why has this happened?

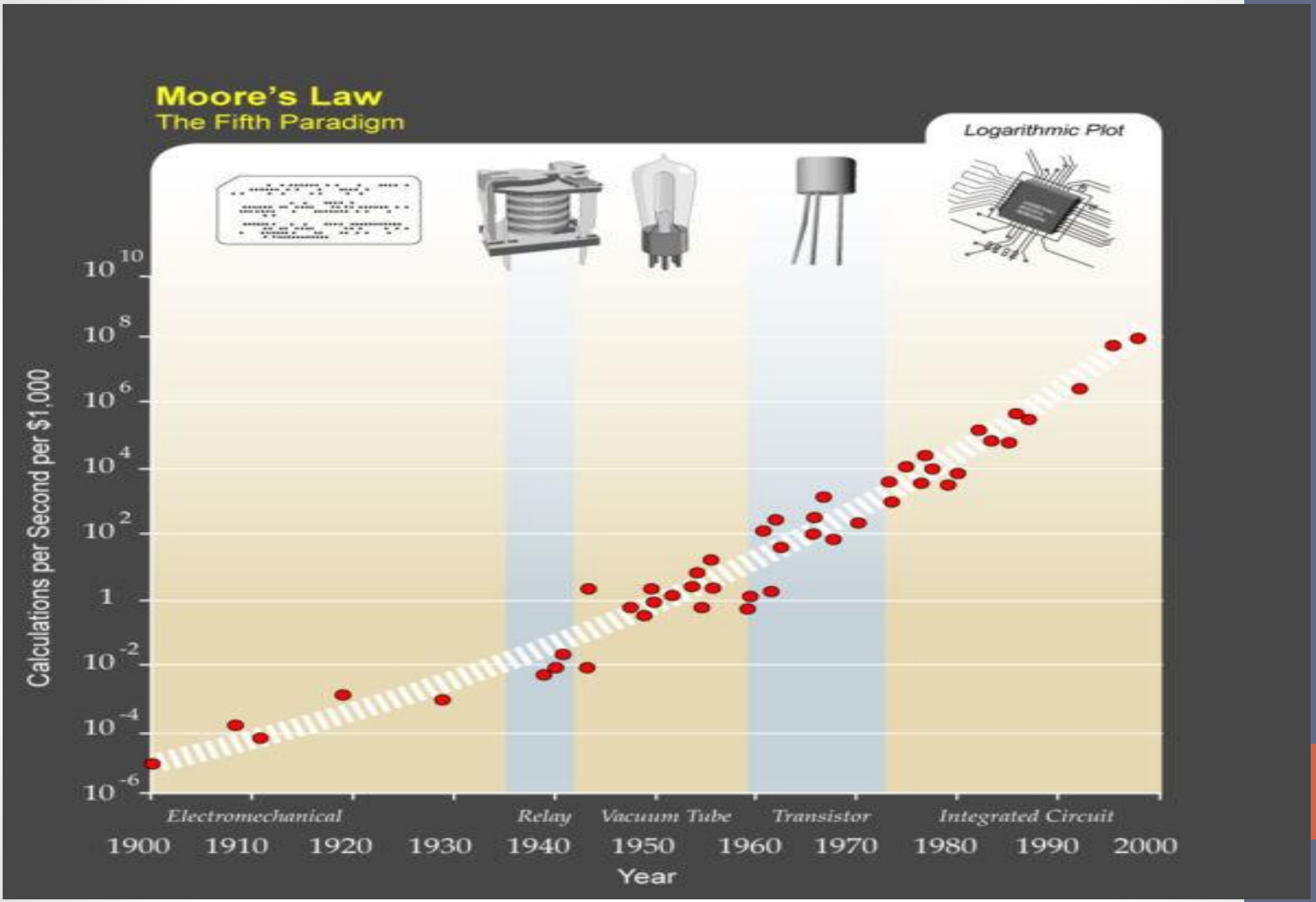
  A:

Moore's Law

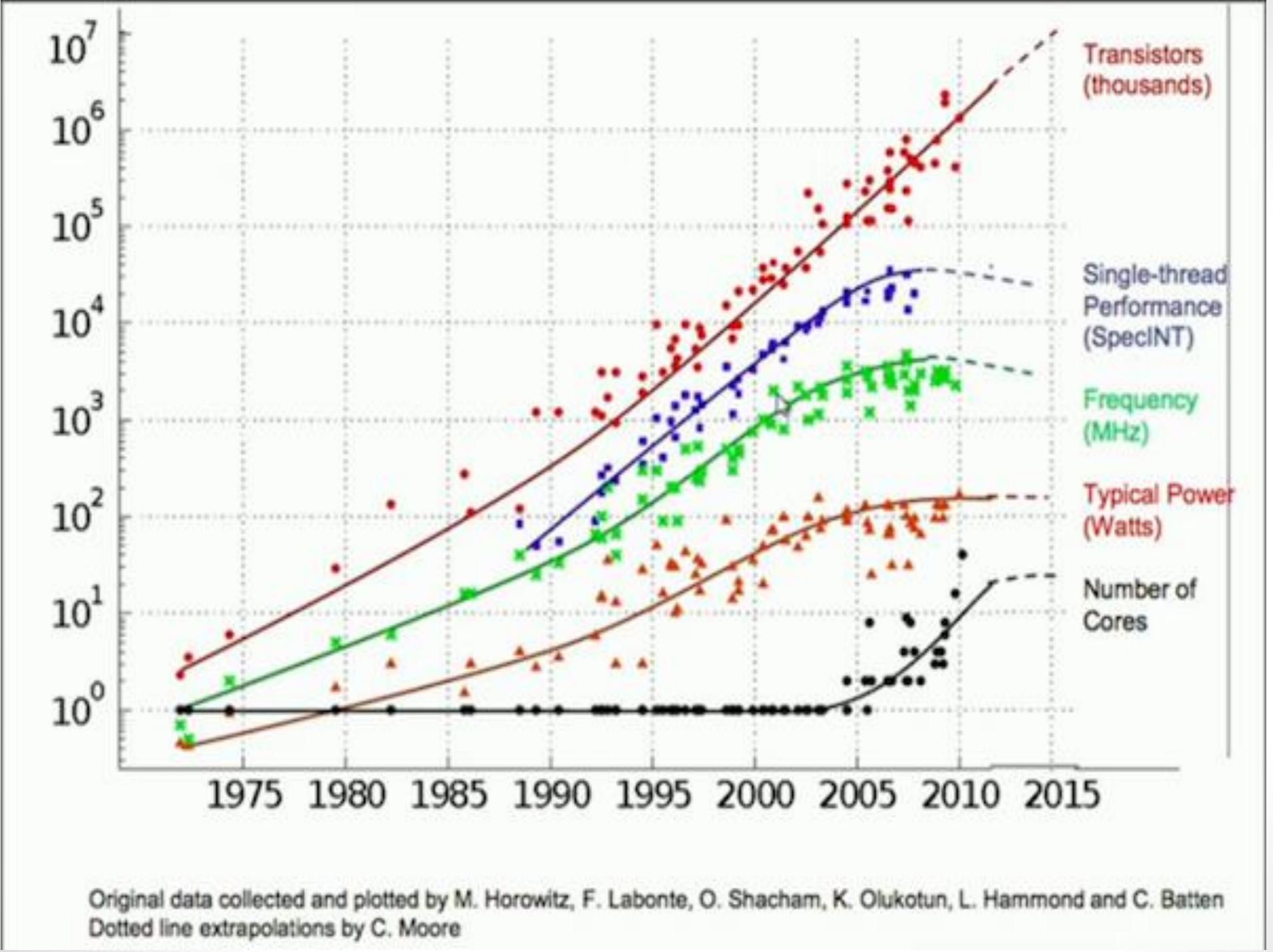
"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year.... Over the longer term, ... there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000. I believe that such a large circuit can be built on a single wafer..."

--- Moore's original statement


transistors
10,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
MOORE'S LAW
Dual-Core Intel® Itanium® 2 Processor
Intel® Itanium® 2 Processor
Intel® Itanium® Processor
Intel® Pentium® 4 Processor
Intel® Pentium® III Processor
Intel® Pentium® II Processor
Intel® Pentium® Processor
Intel486™ Processor
Intel386™ Processor
286
8086
8080
8008
4004
1970
1975
1980
1985
1990
1995
2000
2005
2010



http://en.wikipedia.org/wiki/Moore%27s_Law


10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
1975 1980 1985 1990 1995 2000 2005 2010 2015
Transistors (thousands)
Single-thread Performance (SpecINT)
Frequency (MHz)
Typical Power (Watts)
Number of Cores
Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Counting in TENS

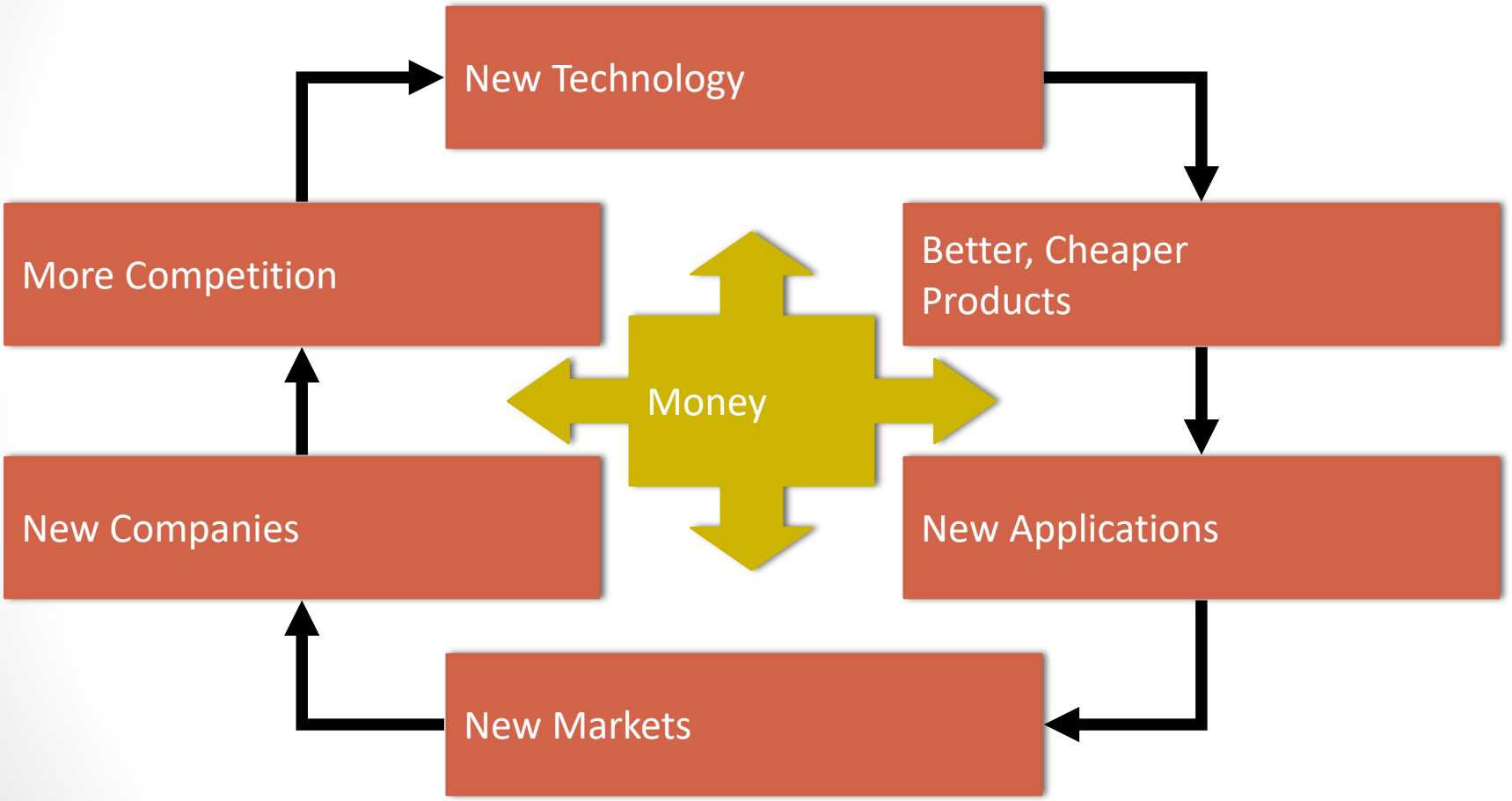
• ***One***	10^{0}	1. *"ten to the **zero**"*
• *Ten*	10^{1}	10. *"ten to the one"*
hundred	10^{2}	100. *"ten to the two"*
thousand	10^{3}	1,**000**. *"ten to the **three**" ten*
thousand	10^{4}	10,000. *"ten to the four"*
hundred thousand	10^{5}	100,000. *"ten to the five"*
• ***million***	10^{6}	1,**000,000**. *"ten to the **six**"*
ten million	10^{7}	10,000,000. *"ten to the seven"*
hundred million	10^{8}	100,000,000. *"ten to the eight"*
billion	10^{9}	1,**000,000,000**. *"ten to the **nine**"*
ten billion	10^{10}	10,000,000,000. *"ten to the ten"*
hundred billion	10^{11}	100,000,000,000. *"ten to the eleven"*
trillion	10^{12}	1,**000,000,000,000**. *"ten to the **twelve**"*
ten trillion	10^{13}	10,000,000,000,000. *"ten to the thirteen"*
hundred trillion	10^{14}	100,000,000,000,000. *"ten to the fourteen"*

Nathan's 1st Law of Software

- "*Software is a gas. It expands to fill the container holding it*"

  (Nathan Myhrvold, Microsoft)
- Example: Word processors
  - 1980 = 10-100K (Wordstar)
  - 1990 = 100K-10Mb (Wordperfect 5.1)
  - 2000 = 10Mb-100Mb (MS Office 2K)

Virtuous Economic Circle



The Evolution of the CPU Chips

Chip	Date	MHz	Transitors
8008	1972	0.108	3500
8086	1978	5-10	29000
80286	1982	8-12	134000
80386	1985	16-33	275000
80486	1989	25-100	1.2M
Pentium	1993	60-233	3.1M
Pentium II	1997	233-400	7.5M
Pentium III	1999	450-1.4G	9.5M
Pentium IV	2000	1.3G-3.8G	42M
Intel Core	2006	1.5G-2G	291M
Intel 6 core i7	2010	2.8G-3.33G	1.17G

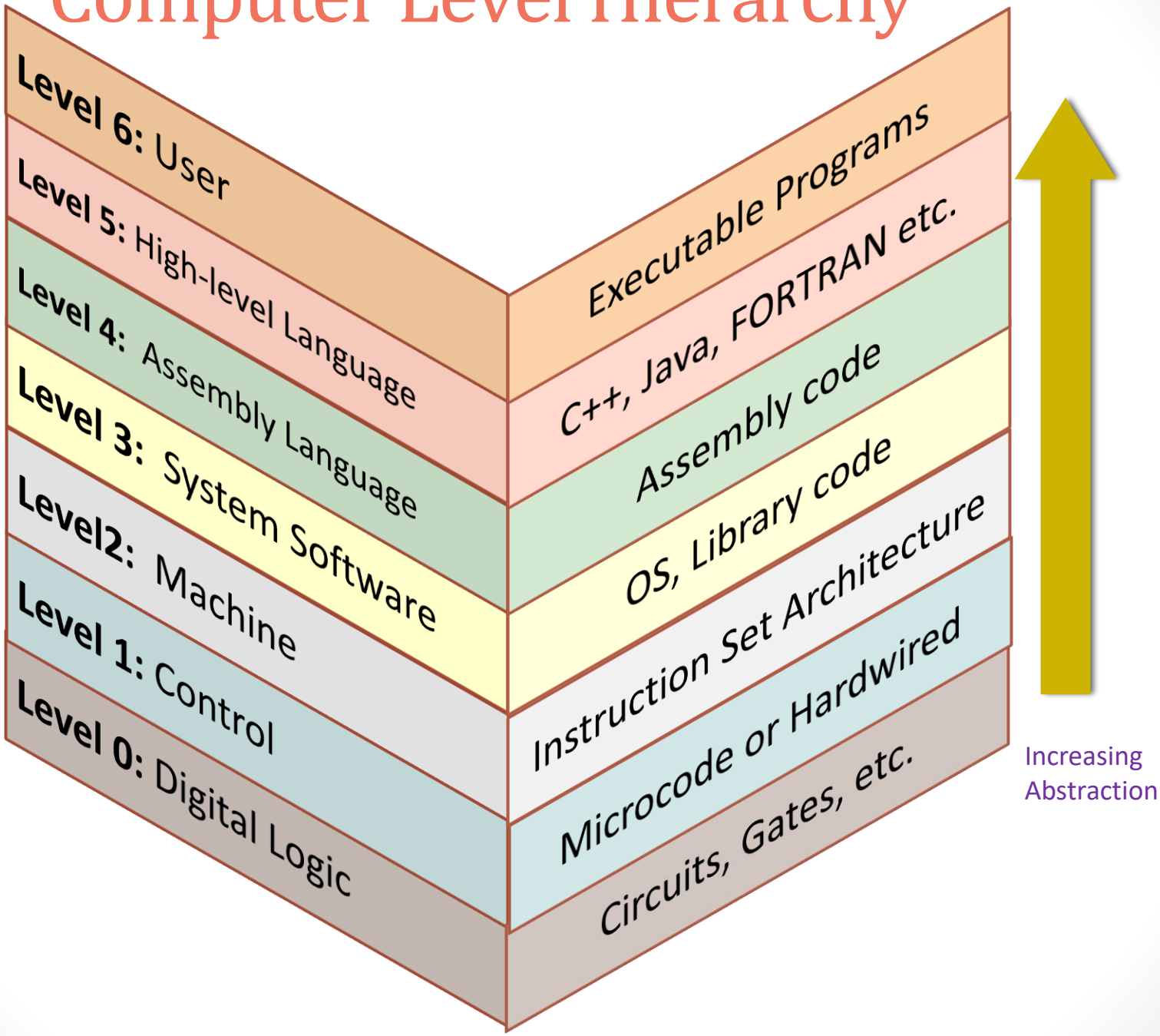
What is a Digital Computer?

- An electronic machine which solves problems by carrying out a sequence of instructions
- The instructions are its *program,* and are usually very simple:
  - Add two numbers
  - Check to see if something is zero
  - Copy data from A to B

Computer Level Hierarchy

- Through the principle of abstraction, we can view the machine to be built from a hierarchy of levels
  - Each level has a specific function and can be seen as a distinct abstract virtual machine.
  - Each level's virtual machine executes its own set of instructions, calling upon machines at lower levels.
- Application programmers particularly interested in the top two levels

Computer Level Hierarchy

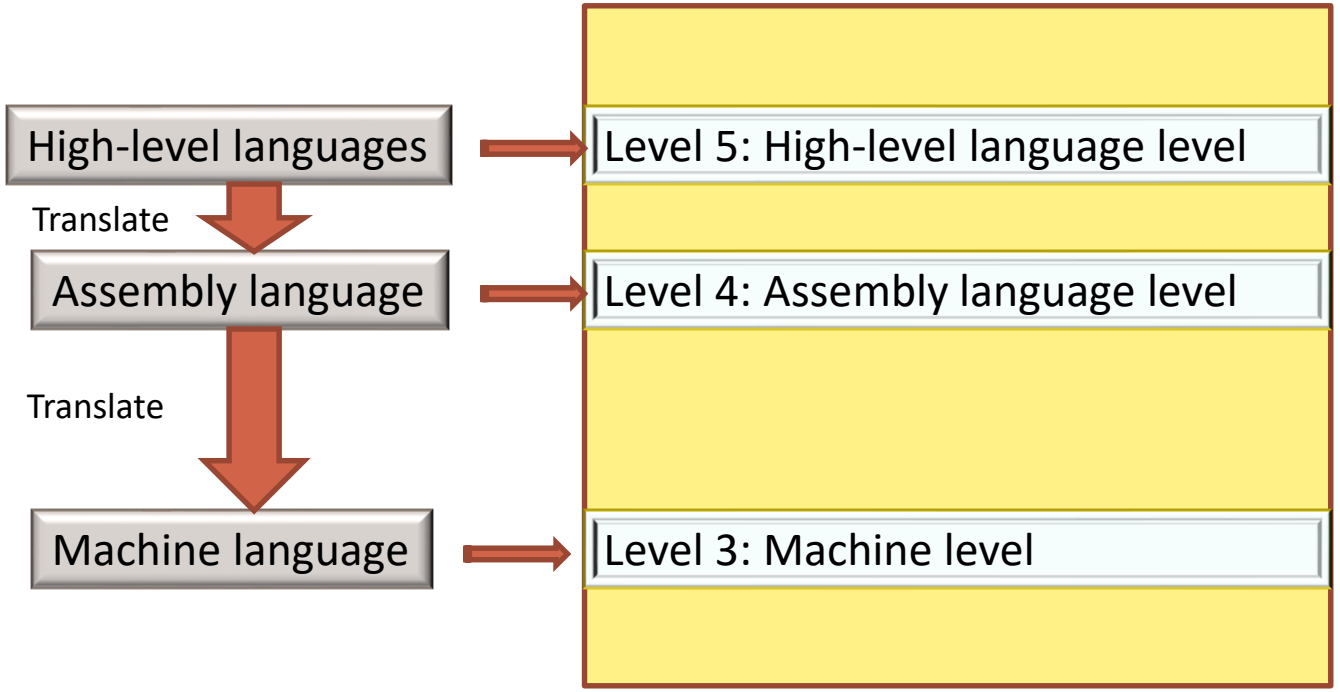


Level 6: User Level

- Is composed of applications
  - Word processors
  - Games
  - Browsers
  - Mailers

Level 5: High-Level language level

- It consists of high-level languages such as Java, C/C++, Fortran, Prolog etc.
  - These languages must be translated to a language the machine can understand.



Level 4: Assembly Language Level

- This is really just a human-readable form of one of the underlying languages
  - Reduce the semantic gap between machine language and high-level languages.
- Based on the Machine (ISA) level
- Programs in assembler need to be translated into machine language before execution
  - One-to-one translation: one assembly language instruction to one machine instruction

Level 3: System Software Level

- Deals with operation system instructions
- Extends the ISA level with extra functions for:
  - Memory management
  - Process control
  - Interprocess communication
  - Multiprogramming
- We'll see more later…

Level 2: Machine Level

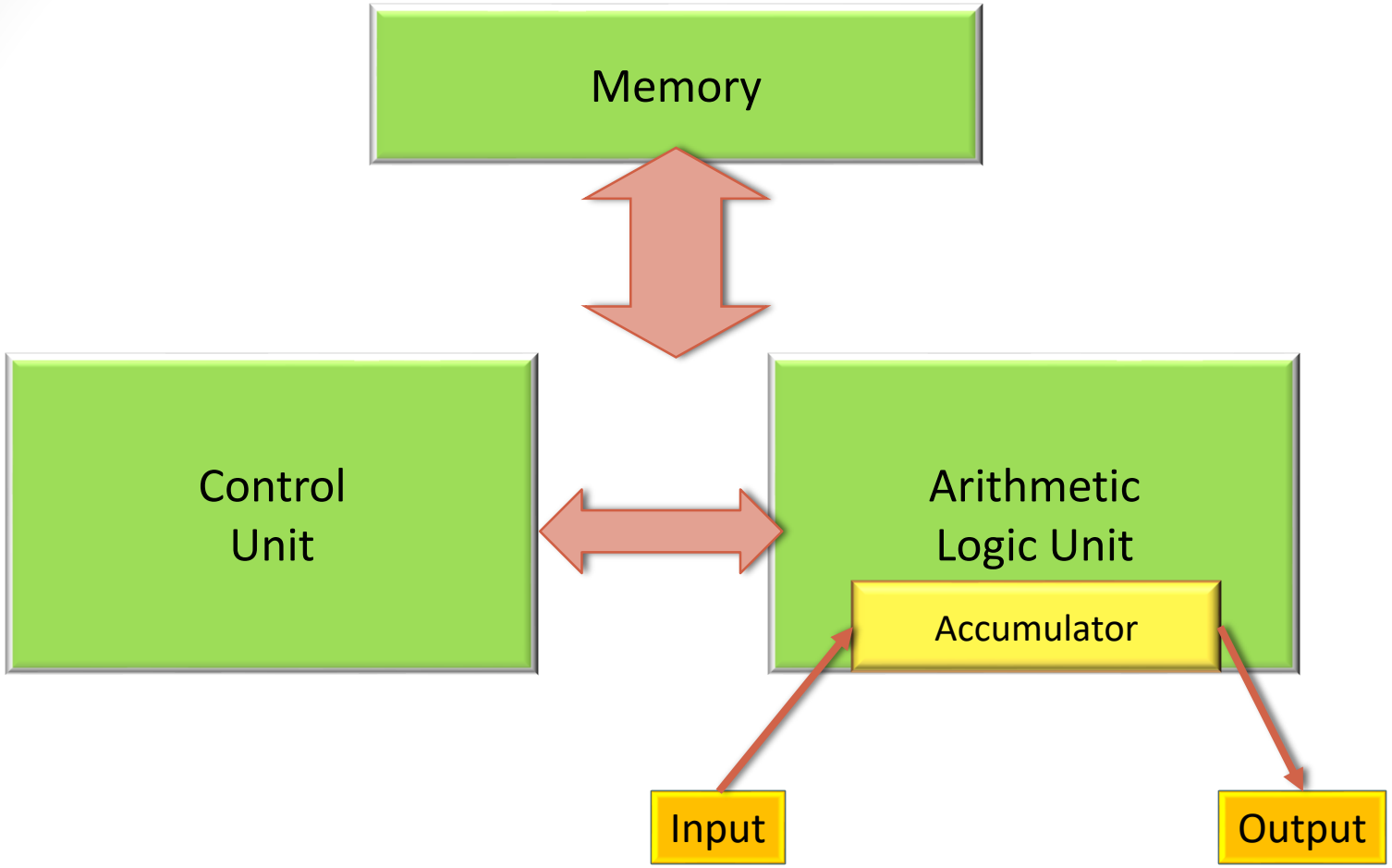
- It is also called Instruction Set Architecture (ISA) Level
- Consists of machine language
- An instruction may be carried out in hardwired control unit (direct execution), or microprogrammed control unit (in Level 1).

Level 1: Control Level

- This level is where a control unit interprets the machine instructions passed to it, one at a time, from the level above, causing the required actions to take place.
- Hardwired control units and microprogrammed control units
  - Hardwired: control signals emanate from blocks of digital logic components
    - Fast but difficult to modify
  - Microprogrammed: machine code is implemented directly by the hardware.
    - Slower but modifiable

Level 0: Digital Logic Level

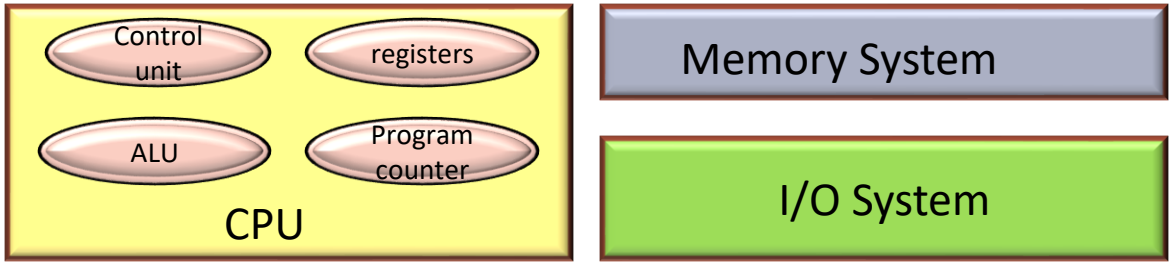
- Digital circuits constructed from *logic gates*
- Each gate computes a simple function based on a small number of digital inputs.
- Clusters of logic gates are combined to make *registers*, *control circuits* and other computational devices



The von Neumann Machine

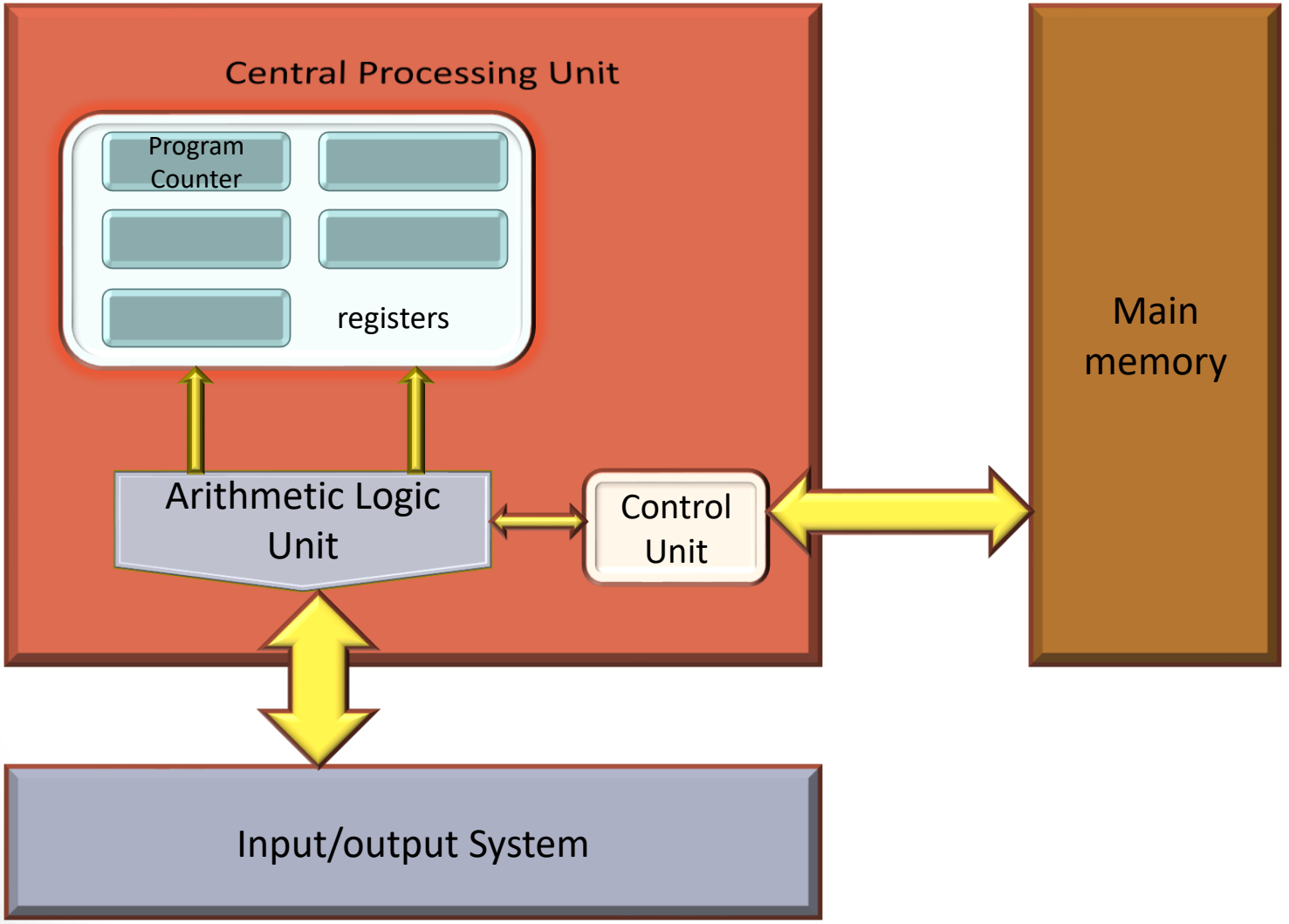
Modern Von Neumann Architecture

- Today's version of the stored-program machine architecture satisfies at least the following characteristics
  - Consists three hardware systems:

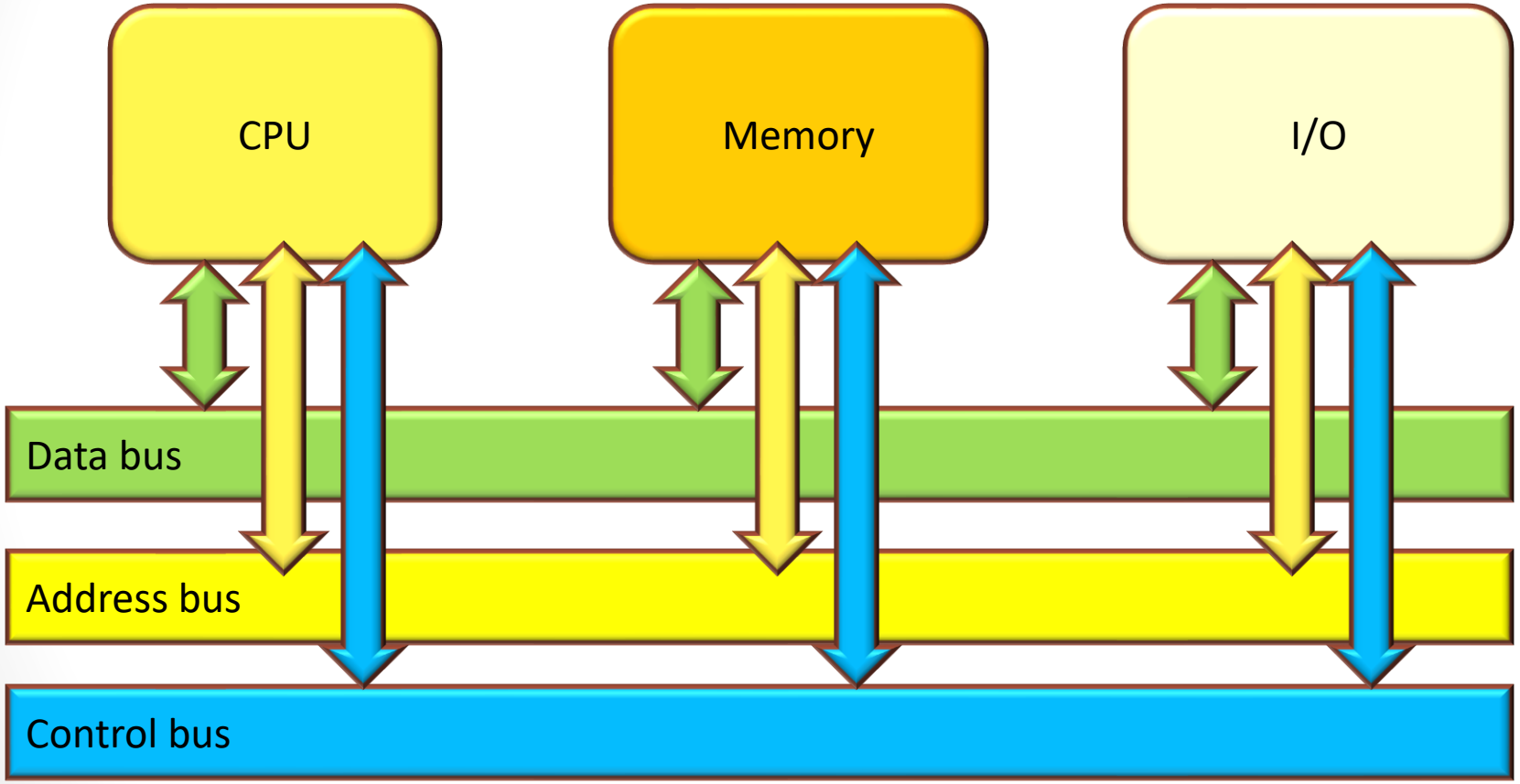


  - Can carry out sequential instruction processing
  - Von Neumann bottleneck
    - It contains a single path between CPU and memory, forcing alternation of memory accessing and CPU execution cycle.
    - The amount of time a CPU spends waiting data to be fetched outpace the time a CPU spends doing actual work
      - A faster CPU is no longer translated to a faster computer.

The Modern Von Neumann Architecture



Modified Von Neumann Architecture



Non-Von Neumann Models

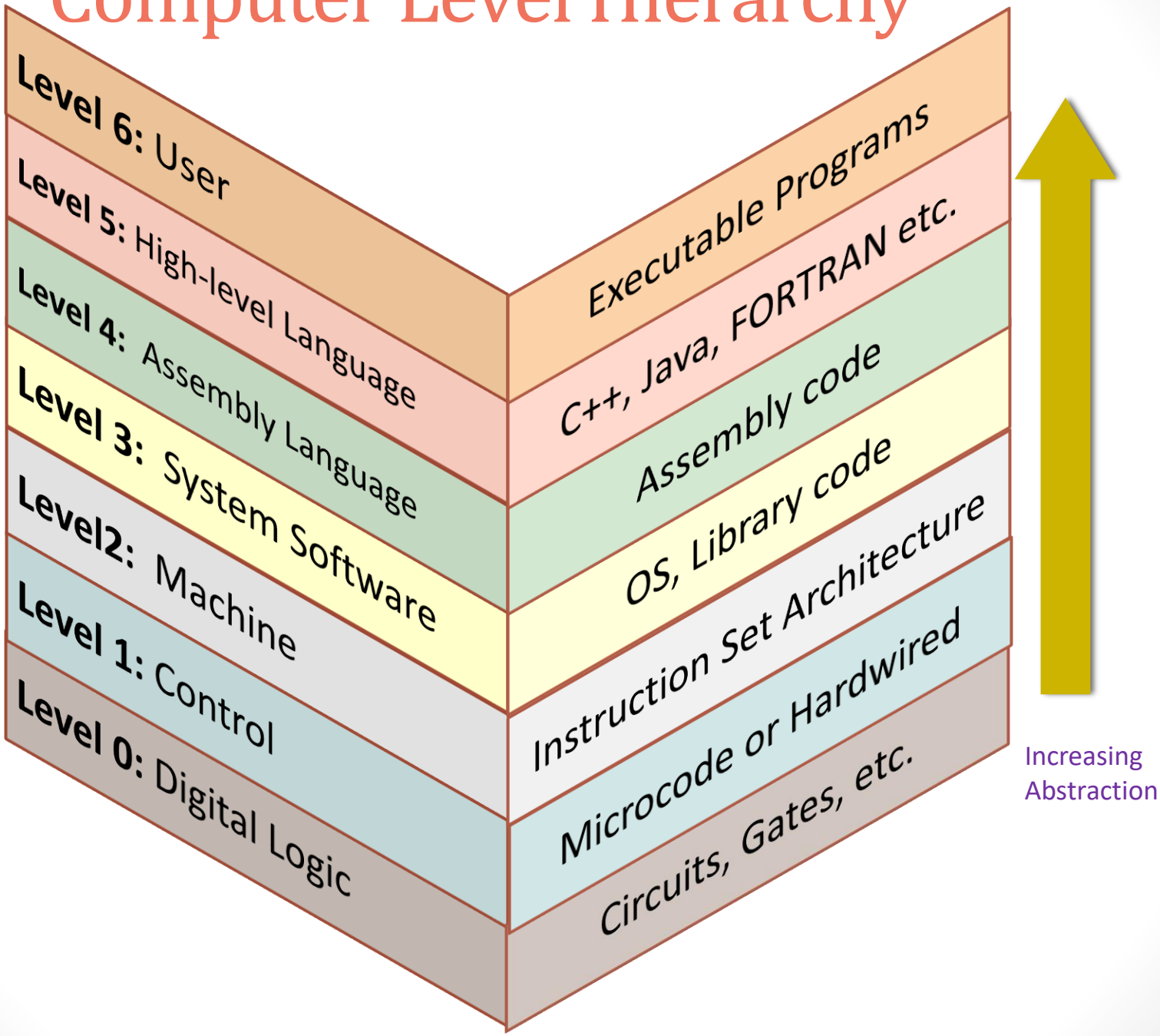
- Most general-purpose computers follow Von Neumann design.
  - Von Neumann model suits sequential processing
  - Difficulty with Von Neumann bottleneck
- Non-Von Neumann Models
  - May not store programs and data in memory
    - Neural network architecture
    - Cellular automata :

    Cellular automata is a discrete model studied in computability theory, mathematics, physics, complexity science, theoretical biology and microstructure modeling.
  - May not process a program sequentially
    - Parallel processing:
      - Multicore architecture ,
      - Quantum computing etc.

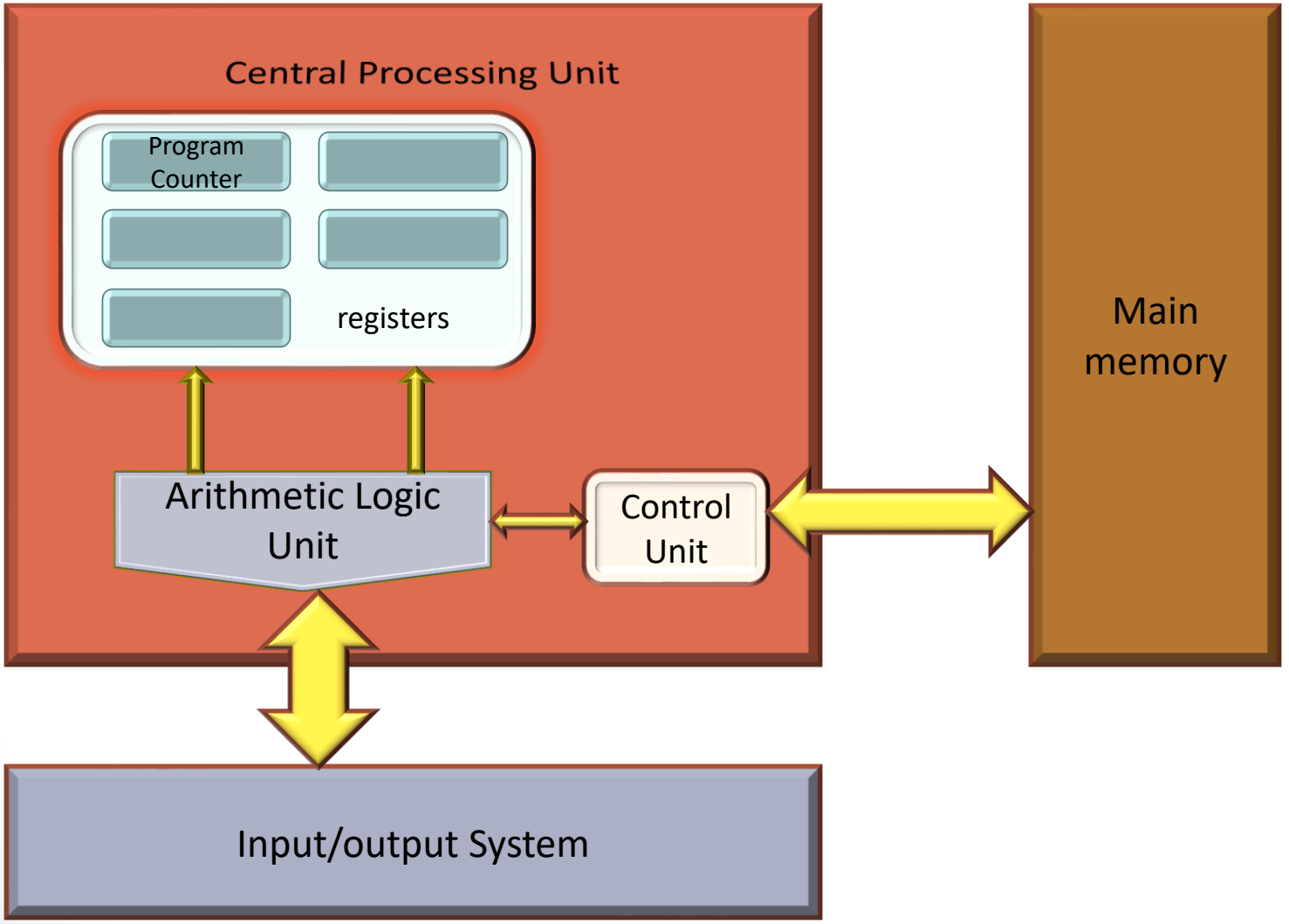
Summary

- Multilevel machines
  - Computers are designed as a series of levels, each built upon the last
  - Knowing how computers work makes programming a lot easier
- 4 generations of innovation
  - The von Neumann Machine
  - General trend towards faster, cheaper, smaller and more reliable computers
    - Moore's Law
  - Progress in the computer-industry is self-catalyzing
    - Virtuous Circle
- Modern Von Neumann Models
  - Characteristics
  - Bottleneck

Computer Level Hierarchy



The Modern Von Neumann Architecture



Logic Gates

Boolean Algebra, Truth Table

Logic Gates, Arithmetic Circuits, Decoder and ALU

Boolean Algebra

- *Boolean algebra* is the algebra of 2 values: 1(True) and 0(False)
- Basic Boolean functions

Conventions

NOT	¬A	Not A	A
AND	A ∧ B	A and B	AB
OR	A ∨ B	A or B	A + B
XOR (exclusive-or)	A⊕B	A XOR B	A x B

Truth Table

- is used to describe Boolean functions
  - lists all possible values of the inputs to the function

A	$\overline{A}$
1	0
0	1

NOT

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

AND

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

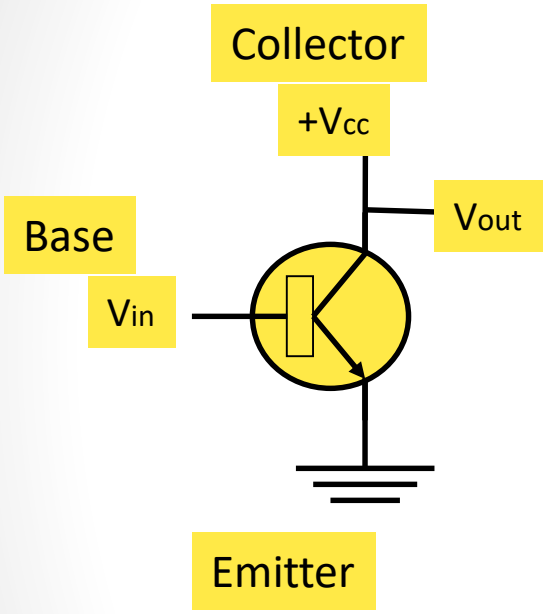
OR

A	B	AXB
0	0	0
0	1	1
1	0	1
1	1	0

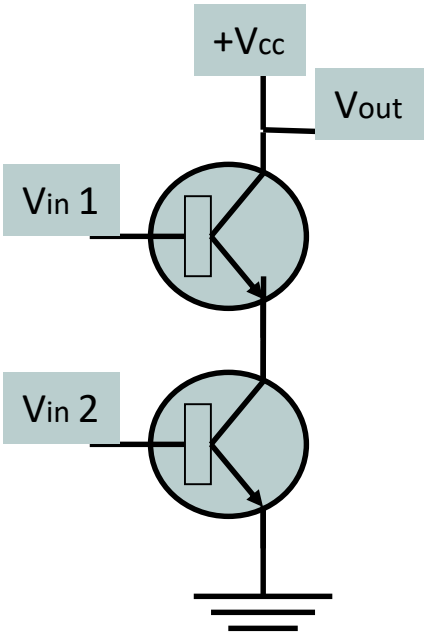
XOR

- Need 2^n rows in a truth-table for a function of n variables

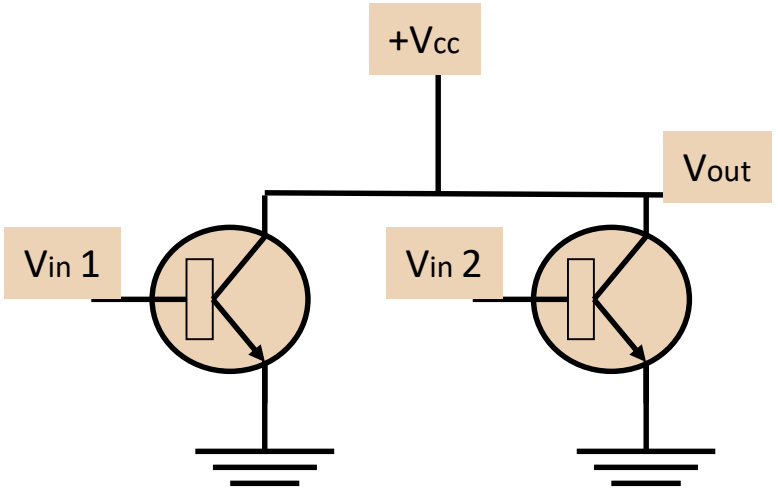
Transistors as Logic Gates



NOT Gate

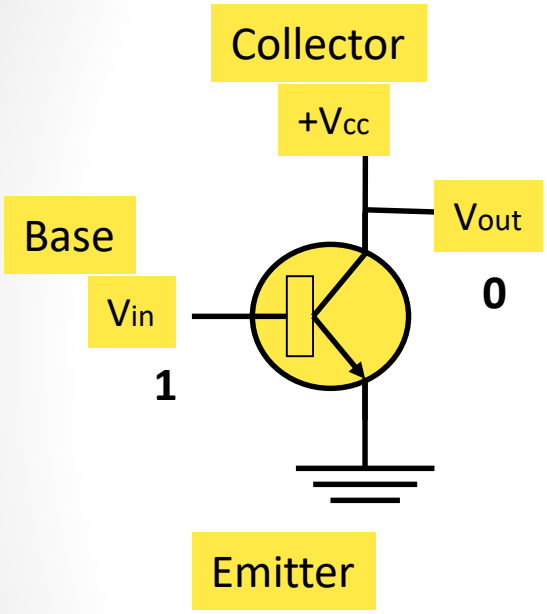


NAND Gate

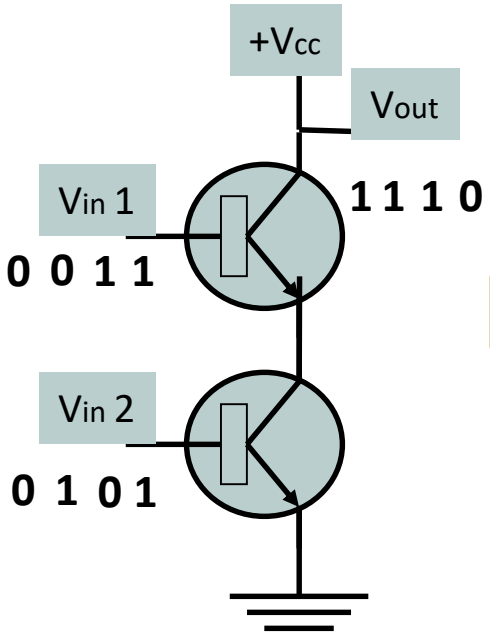


NOR Gate

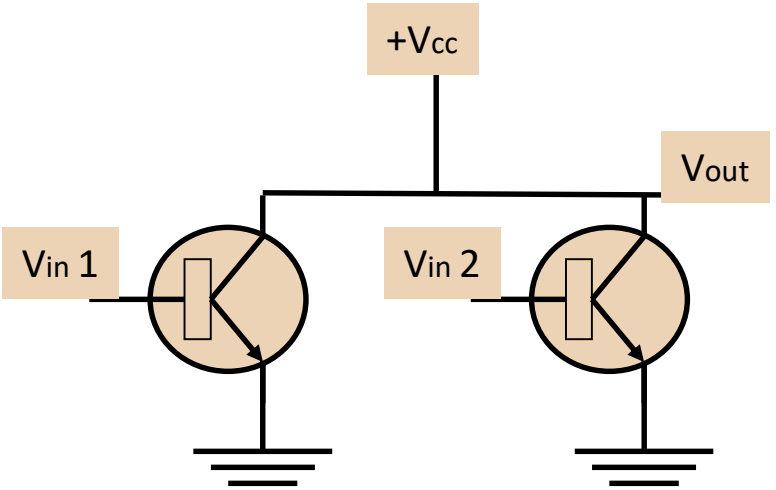
Transistors as Logic Gates



NOT Gate

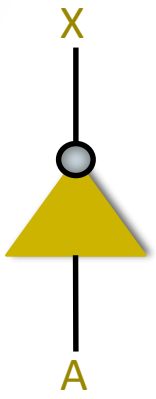


NAND Gate



NOR Gate

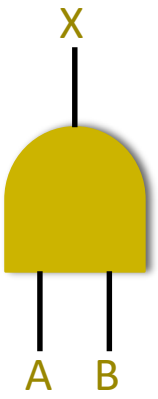
Logic Gates



A	X
0	1
1	0

X=A

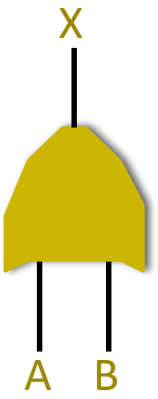
NOT



A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

X=AB

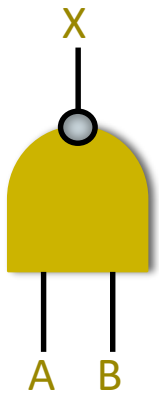
AND



A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

X=A+B

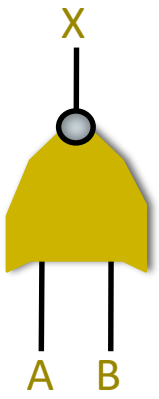
OR



A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

X= AB

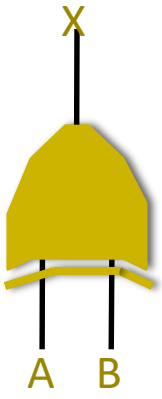
NAND



A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

X= A+B

NOR



A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

X=AxB

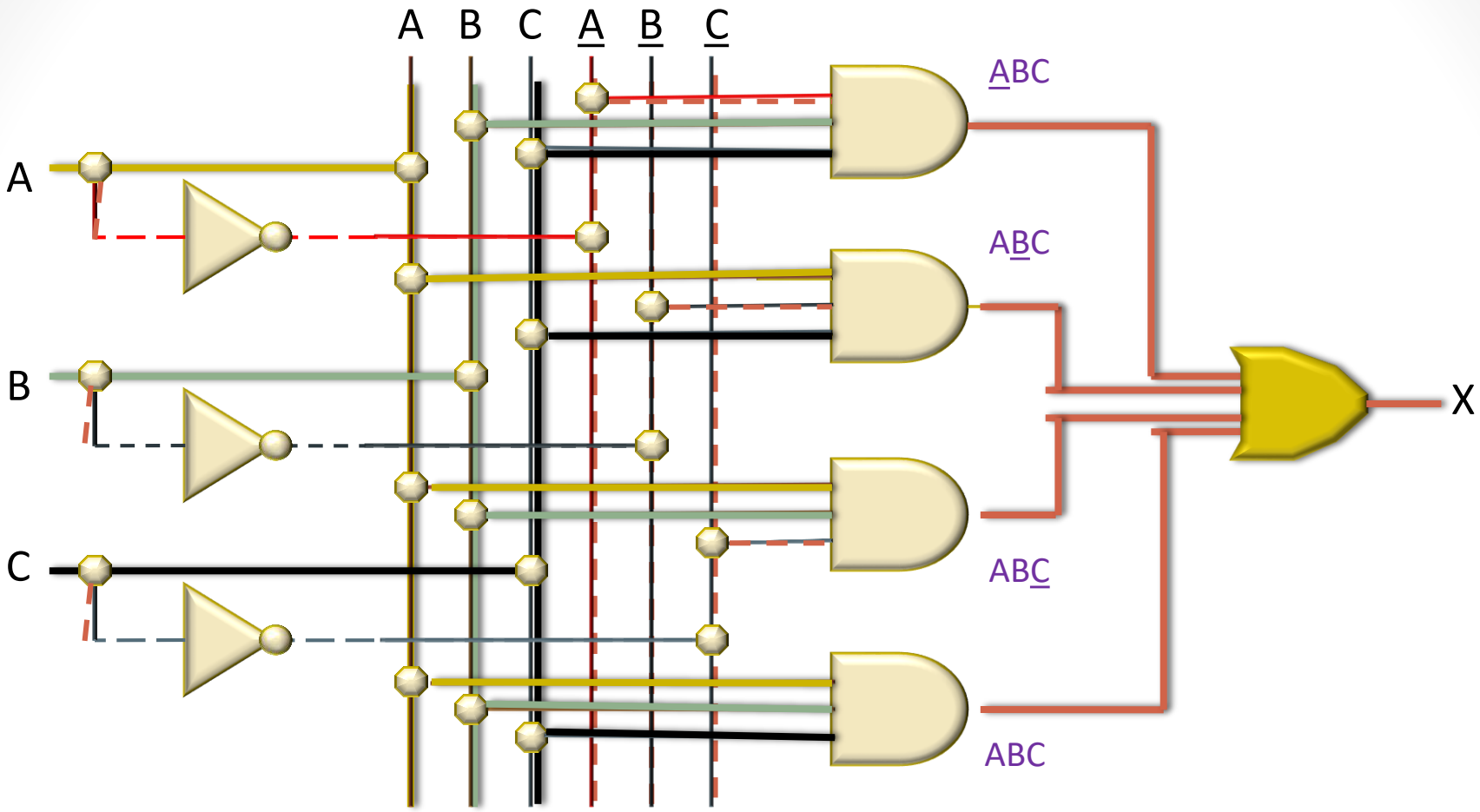
XOR

Fill in

A	B	C	X
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

- If there are more inputs which are **1** than **0**, then output **1**. Otherwise output **0**
-

Example: The Majority Function



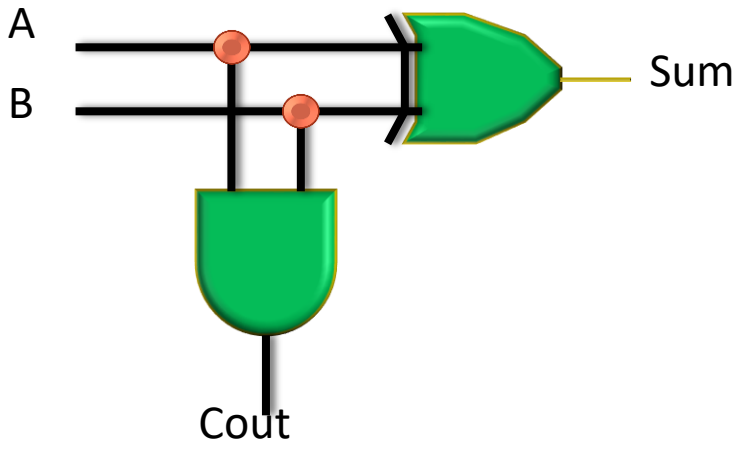
The Majority Function as a Digital Circuit

Arithmetic Circuits

- We can easily build circuits which perform *Boolean operations* on binary words at the bit-level
- Binary arithmetic can also be thought of as a *Boolean operations*
- We can build circuits which add, subtract, multiply and divide in a similar way

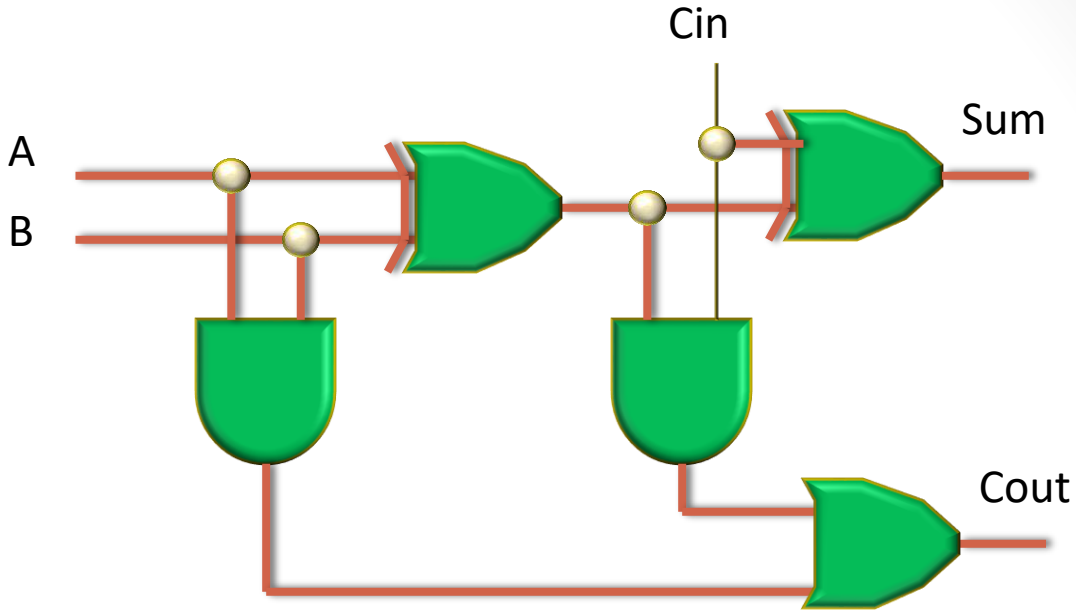
A Half-Adder

A	B	Sum	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



- A half-adder performs 1-bit addition
- But it will not work for a bit in the middle of a word
  - It needs a "carry" input

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

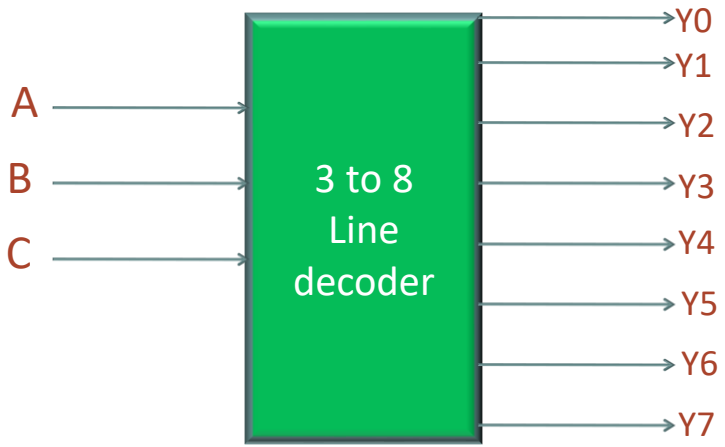


- Constructed from 2 half-adders
- Can replicate this circuit to build adders for any word size
  - Need to propagate the carry

A Full-Adder

Decoders

A	B	C	0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



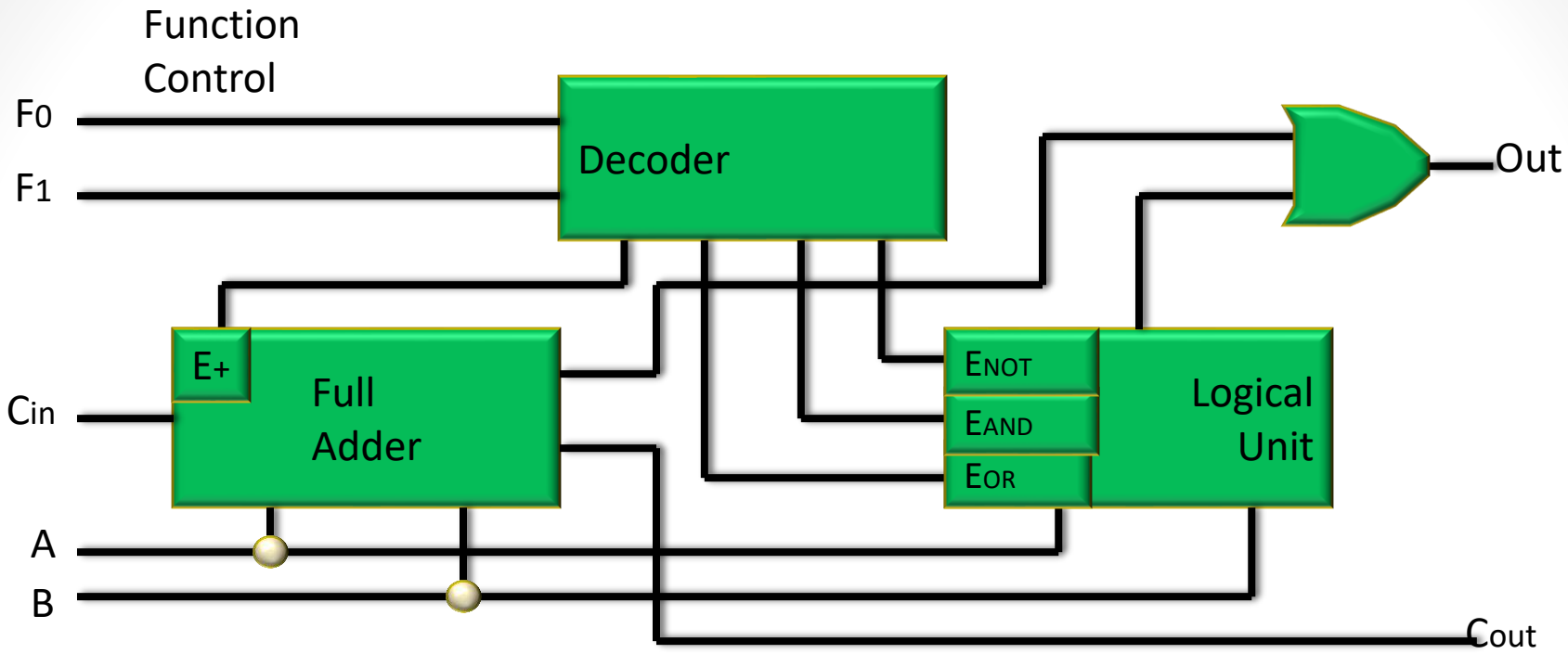
SN74F138 3 to 8 line decoder

- More output than input lines
- Select one output line at a time
- Useful in address decoders

  E.g could be used to select amongst 8 memory chips by taking the top three bits of an address.

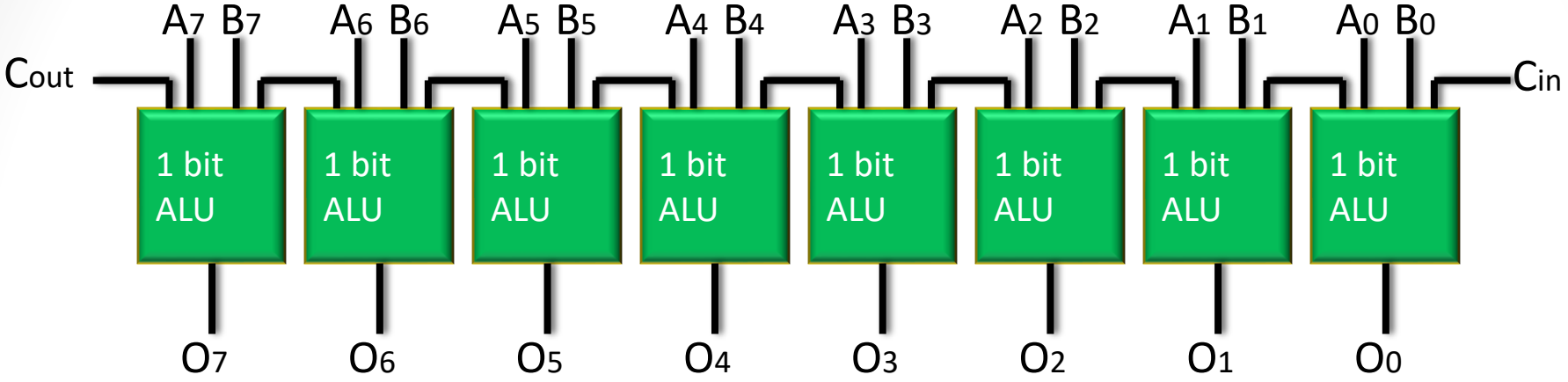
ALU(Arithmetic Logic Unit)

- A single, complex hardware unit to carry out Boolean operations such as AND, OR, NOT and bit operations such as left and right shift and addition



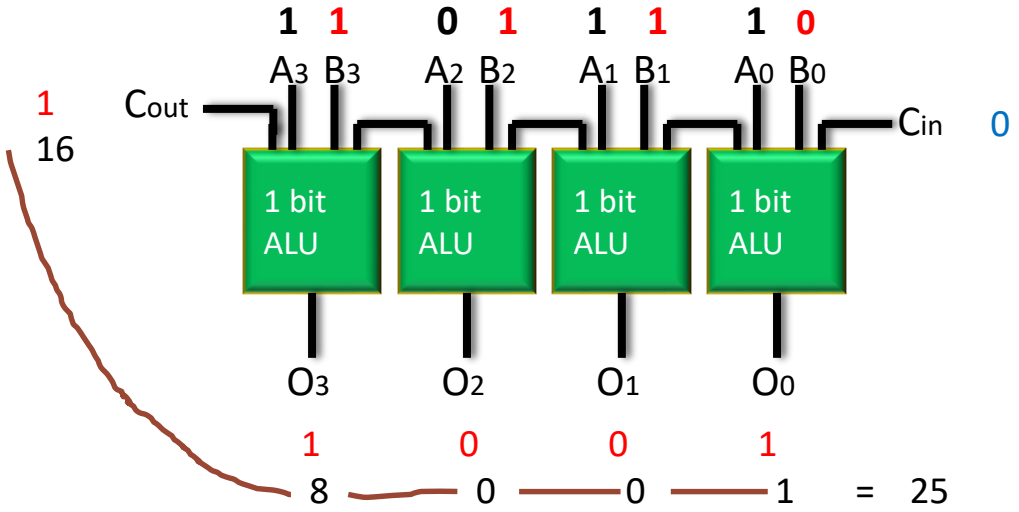
- It uses a *decoder* to select the active function

A 1-bit ALU



- The 8-bit ALU is constructed by replicating a 1-bit ALU

An 8-bit ALU



Consider 5-bit addition

A = 1011(eleven), B = 1110 (fourteen)

Summary

- Boolean Algebra
  - Truth table
  - Basic Boolean functions: *not, and, or, xor*
- Logic gates and their truth table
  - Built from transistors
- Digital Circuits
  - Design your own circuit
    - The majority function example
  - Decoders
  - Adders
- The design of an ALU
  - 1-bit and 8-bit

Chapter 3

Binary Representation

Bits,

binary-to-decimal and decimal-to-binary converting,

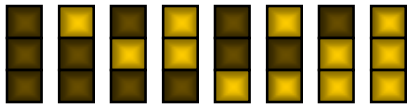
representations of negative integers and real numbers

Why *Binary* Arithmetic?

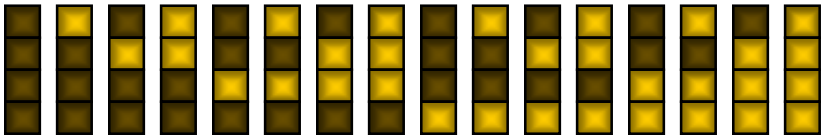
- Early computers often used decimal arithmetic
- Binary arithmetic is the most robust
  - Only need to distinguish between two different voltage levels
- A decimal computer would need to distinguish ten different levels
  - Would need a much lower noise level to operate reliably – expensive
- Another possibility is an *analogue* computer
  - But the presence of noise means low accuracy

1 bits—2 combinations

2 bits—4 combinations



3 bits—8 combinations



4 bits --16 combinations

- Each bit we add increases the number of possible combinations by 2
- With *n* bits, we have 2^n combinations
- Hence 8 bits (1 byte) gives $2^8 = 256$ different combinations

8 bits, 1 byte, 256 values

How many bits in a date?

- Using *binary* values means we must divide our world into powers of 2
  - We usually pick the nearest power of 2
- For the date format "DD/MM/YYYY"
  - 28 to 31 days in a month: $2^5 = 32$
  - 12 months in a year: $2^4 = 16$
  - 9999 possible years: $2^{14} = 16384$
- So we would need 5+4+14=23 bits in total to represent a date like this
  - This leaves many illegal bit combinations
    - E.g. "32/16/9999"

Numbers

- Decimal
  - Base 10
    - 0-9
    - i.e. 19 in decimal is (9 + 10)
- Hexadecimal
  - Base 16
    - 0-9
    - A-F
    - i.e. 13 in Hexadecimal is (16 + 3)
- Binary
  - Base 2
    - 0 or 1
    - i.e. 00010011 in binary (2^0 + 2^1 + 2^4)

Binary	Decimal
000	$0 = 0x2^2 + 0x2^1 + 0x2^0$
001	$1 = 0x2^2 + 0x2^1 + 1x2^0$
010	$2 = 0x2^2 + 1x2^1 + 0x2^0$
011	$3 = 0x2^2 + 1x2^1 + 1x2^0$
100	$4 = 1x2^2 + 0x2^1 + 0x2^0$
101	$5 = 1x2^2 + 0x2^1 + 1x2^0$
110	$6 = 1x2^2 + 1x2^1 + 0x2^0$
111	$7 = 1x2^2 + 1x2^1 + 1x2^0$

- By convention, the right-most bit is the least-significant
- Each subsequent bit is worth twice the one before
  - We can build this idea into an algorithm

Binary-to-Decimal

Decimal-to-Binary

- This involves the successive division of the decimal number by 2
- The *remainder* at each stage is the next digit of the binary expansion
  - *E.g. Converting 58 into binary*

*Divide 58 by 2 = 29 remainder **0***
*Divide 29 by 2 = 14 remainder **1***
*Divide 14 by 2 = 7 remainder **0***
*Divide 7 by 2 = 3 remainder **1***
*Divide 3 by 2 = 1 remainder **1***
*Divide 1 by 2 = 0 remainder **1***

  - *So 58 is **111010** in binary*

Binary to Hexadecimal

MSB LSB

128	64	32	16	8	4	2	1
1	1	0	1	0	1	0	1

Binary Number

SPLIT into
Two nibbles

High Nibble

8	4	2	1
1	1	0	1

Add 8 + 4 +1 = 13

13 in Hex = D

Low Nibble

8	4	2	1
0	1	0	1

Add 4 + 1 = 5

5 in Hex = 5

ANSWER 11010101 binary = D5H

Hexadecimal to Decimal

Weighting	16^2	16^1	16^0
Digits	2	F	A
2FA =	$2 * 16^2$ +	$F * 16^1$ +	$A * 16^0$
=	2 * 256 +	15 * 16 +	10
=	512 +	240 +	10

ANSWER 2FA in HEX = 762 Decimal

SUMARY bit fields

Decimal	128	64	32	16	8	4	2	1
Hexdecimal	268435456 10000000H	16777216 1000000H	1048576 100000H	65536 10000H	4096 1000H	256 100H	16 10H	1 1

Representations of Negative Integers

- There are four main ways of representing negative *m*-bit binary numbers
  - Sign and magnitude
  - One's Complement
  - Two's Complement
  - Excess N
- All of them have problems
- But all of them work well with binary addition and subtraction

Fill In

Sign and Magnitude

- Allocate 1 sign bit, leaving the remainder bits for magnitude

00010110 == 22

10010110 == -22

  - The range is therefore –1
  - Two representations for zero

00000000 == 0

10000000 == -0

One's Complement

Fill In

- Also has a sign bit
- When negating a number, we simply flip every 0 to a 1 and visa versa

00010110 == 22

11101001 == -22

- The range is –127 to 127

- Two representations for zero

00000000 == 0

11111111 == -0

Two's Complement

- Like One's Complement, but to negate a number we negate all the bits and add 1 to the result

	00011100 == 28
Invert	*11100011*
	00000001 +
	11100100 == -28

- This gives us a range of –128 to 127
- Only one representation of zero
- But we still have a problem:
  - *–(-128) = -(10000000) = 10000000 = -128*

Fill In

Excess N

- Using a pre-defined number N(= 2^{m-1}) as a basing value, each number is stored as its intended value plus N
  - E.g. Using an 8-bit binary representation (called 8-bit excess-127)

Binary Value	excess-127	unsigned
01000101	-58	69
01101001	-2	
01111110	-	
01111111	0	127
10000000		
10010110	2	
10111010	5	

- The range is –

Excess N (Let's simplify)

- First select the pattern length to be used then write down all the different bit patterns.
- Next we observe that the first pattern with a 1 as its most significant bit appears approx. halfway down the list. We pick this pattern to represent ZERO.
- The following slide shows EXCESS 8 notation to illustrate the concept for simplicity.
- In each case you will see that the binary interpretation of the bit pattern exceeds the value computed by a factor of 8.
- i.e. 1111 = 15 on the table this represents 7 which is 8 less.

Excess 8 example

BIT PATTERN	VALUE REPRESENTED
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	-1
0110	-2
0101	-3
0100	-4
0011	-5
0010	-6
0001	-7
0000	-8

Fixed-Point Arithmetic

- It is useful to be able to represent *real numbers*,
  - e.g. *1.23, 3.141, 0.0000123 …*
- One way we can do this is to reserve a certain number of digits to be the fractional part:
  - *E.g.* **1010.1100** *= 10.75*
    (".1100" = $2^{-1} + 2^{-2}$ = 0.5 + 0.25 = 0.75)
- Given *m* bits before and *n* bits after the decimal point
  - *m* determines the *range*
  - *n* determines the *precision*

Problems with Fixed-point Arithmetic

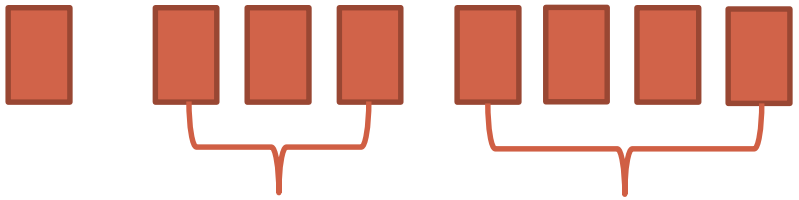
- Fixed-point arithmetic is useful, but limited
- No good for extremely large or small numbers
  - The mass of the Sun is 2×10^{33} grams
  - The mass of an electron is 9×10^{-28} grams
- To use both these amounts in a calculation we would need over 60 decimal digits, most of which would be irrelevant
- We need *floating-point* arithmetic

Floating Point Arithmetic

- Borrow the familiar scientific notation to represent numbers:
  - $3.14 = 0.314 \times 10^{1}$
  - $0.00005 = 0.5 \times 10^{-4}$
- Use powers of 2 instead of 10:
  - N = *mantissa* x $2^{exponent}$ (-1 < *mantissa* < 1)
  - Range depends on *exponent*
  - Precision depends on *mantissa*
- This allows a huge range of numbers to be covered
  - With some loss in accuracy
- IEEE floating-point standard 754:
  - Single precision: 1 sign bit, 8 exponent, 23 mantissa
  - Double precision: 1 sign bit, 11 exponent, 52 mantissa
  - Reserved values for *infinity* and *NaN* (not a number)

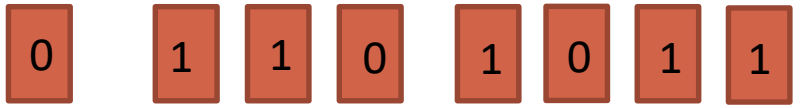
Floating Point example

- This where the idea of using the excess method.
- In our example we will use 8-bits to represent our floating point number.
  - First we designate the high order bit as the sign (1 represents negative)
  - Next we divide the remaining 7-bits into our into two groups, or fields, the **exponent field** and the **mantissa field**.
  - Let's use 3-bits to represent the **exponent** and 4-bits for the **mantissa**



Floating Point example con't

- Using the following bit pattern:



- The sign is 0 therefore it's a positive number.
- The exponent is 110 and the mantissa is 1011
  - To decode the byte we extract the mantissa and put a radix point onits far left obtaining **.1011**
  - Next we extract the contents of the exponent field (110) and interpret it as an integer that is stored using the **3-bit excess method**. Using the table opposite we see this represents the positive value 2. This tells us to move the radix in our solution 2 bits to the right, (a negative value would move it two places left).
  - Giving **10.11** Which is the binary representation for $2_{3/4}$. WHY?

Bit Pattern	Value
111	3
110	2
101	1
100	0
011	-1
010	-2
001	-3
000	-4

Fractional values in binary

- To extend binary notation to accommodate fractional values, we use a radix point as seen e.g. 10.11
- Digits to the right of the radix point are assigned fractional values, the first quantity to the right is a ½ (which is 2-1) the next is a ¼ the next an eighth and so forth.

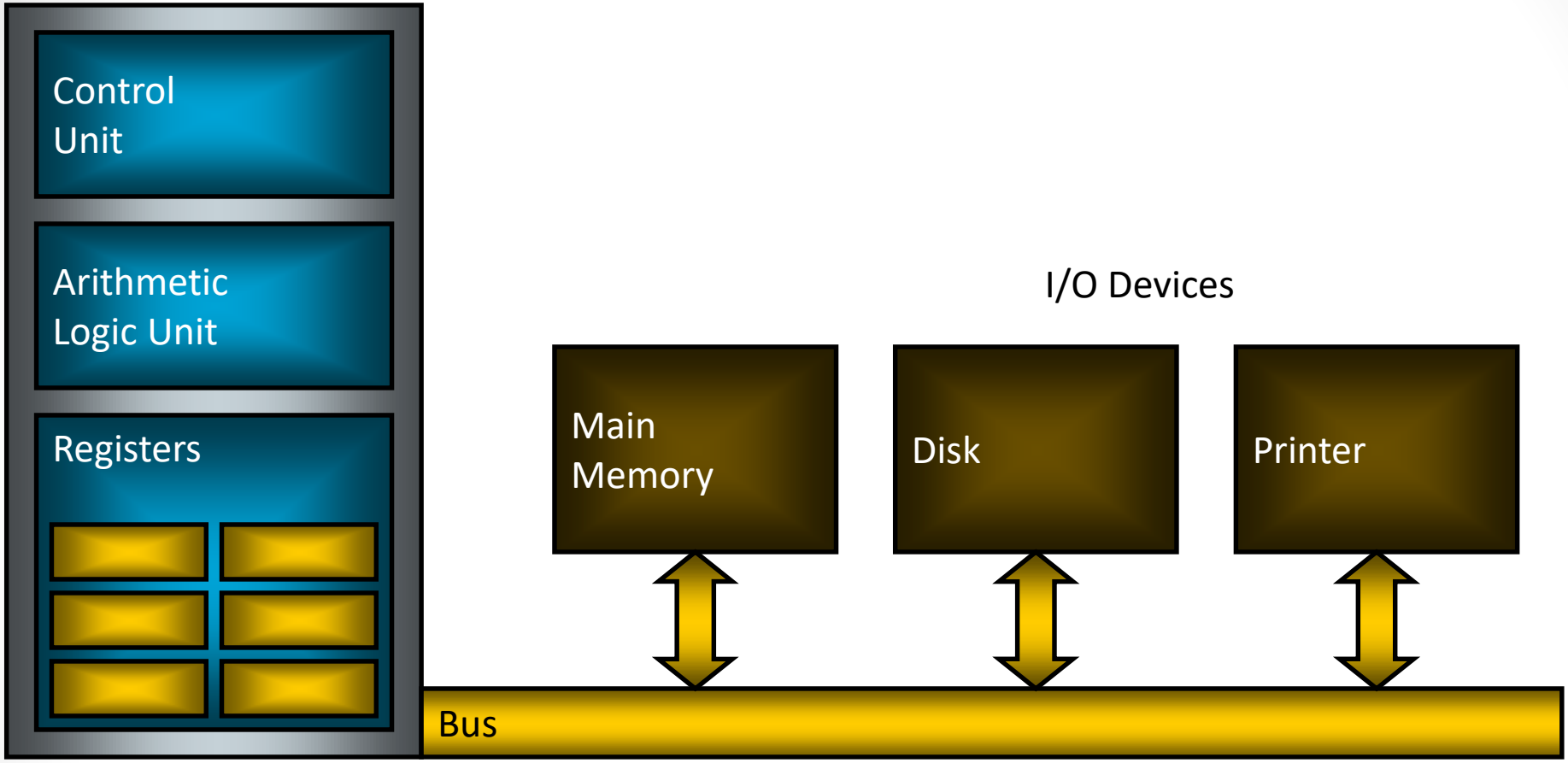
- So in our example above .11 equiv the ½ + ¼ = ¾

Summary

- Computers use a *binary* representation for numbers
- Representing *negative* numbers is something of a challenge
  - No system is ideal
- We can represent real numbers using fixed-point or floating-point arithmetic

Processors

CPU: fetch-execute cycle, clock, instruction set, Registers
CISC and RISC
Parallelism: pipelining, multi-core

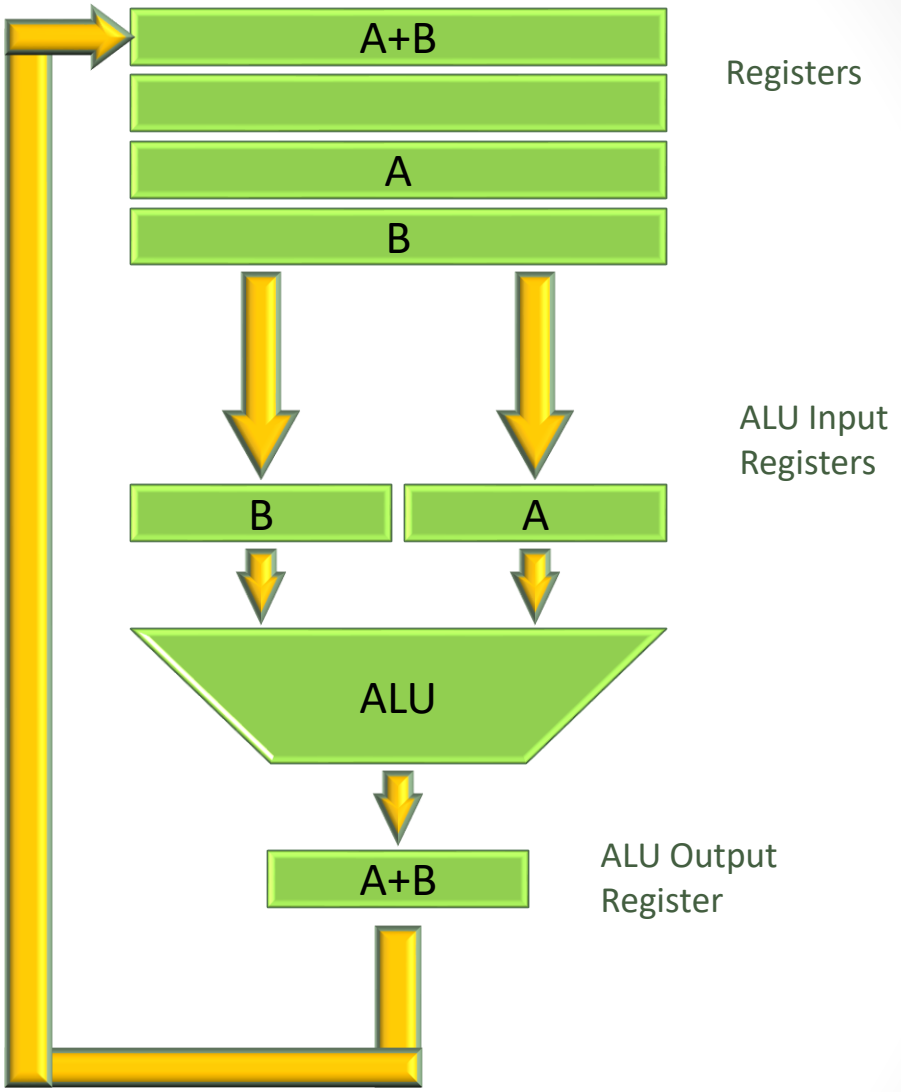


A Typical Layout

What is a CPU?

- Central Processing Unit = “brain”
- Executes programs by:
  - Fetching the next instruction from memory
  - Examine it
  - Execute it
- Consists of:
  - Control Unit
  - Arithmetic Logic Unit (ALU)
  - Registers (high-speed memory)
    - Program Counter (PC)
    - Instruction Register (IR)

The Data Path of a von Neumann Machine



Control: Getting started and Stopping

- The processor begins executing program from a fixed location
  - Location zero in memory
  - ROM
- The fetch-execute cycle never stops unless the computer is powered down
  - Loop is used to delay (wait for I/O completion)
  - An operating system (OS) runs when no application is running
  - When no application is running, the OS enters a loop to wait for input.

Instruction Set

- Refer to the operations the hardware recognizes and performs.
  - One instruction executed per *fetch-execute cycle*.
  - Representation of an instruction
    - Binary format for hardware
    - For software – 3 parts
      - Opcodes, operands, results

Typical Instruction Format

- Each instruction contains three parts
  - *Opcode*—the operation to be performed
  - *Oprands* – the value(s) to be used
  - *Results* – where to place the result(s)
- Binary format

General-Purpose Registers

- They are used to hold an operand or the result of an instruction
- A register is a high-speed hardware device that
  - has fixed size
  - supports two operations
    - *Store*
    - *Fetch*

Programming with Registers

- For a particular task, a series of instructions might be required to move values between memory and the registers
  - Eg. To add two integers X and Y and place the result in memory M. (suppose registers R3, R6 are available)

```
mov   R3   X
mov   R6   Y
add    R6  R3
mov   M   R6
```

Clock

- A **clock** emits an alternative sequence of 0 and 1 values at a regular rate

- The speed of a clock is measured in **Hertz(Hz)**
  - Hertz --- the number of times per second the clock cycles through a 1 followed by a 0.
  - E.g. 3GHz=$3x10^9$ cycles per second

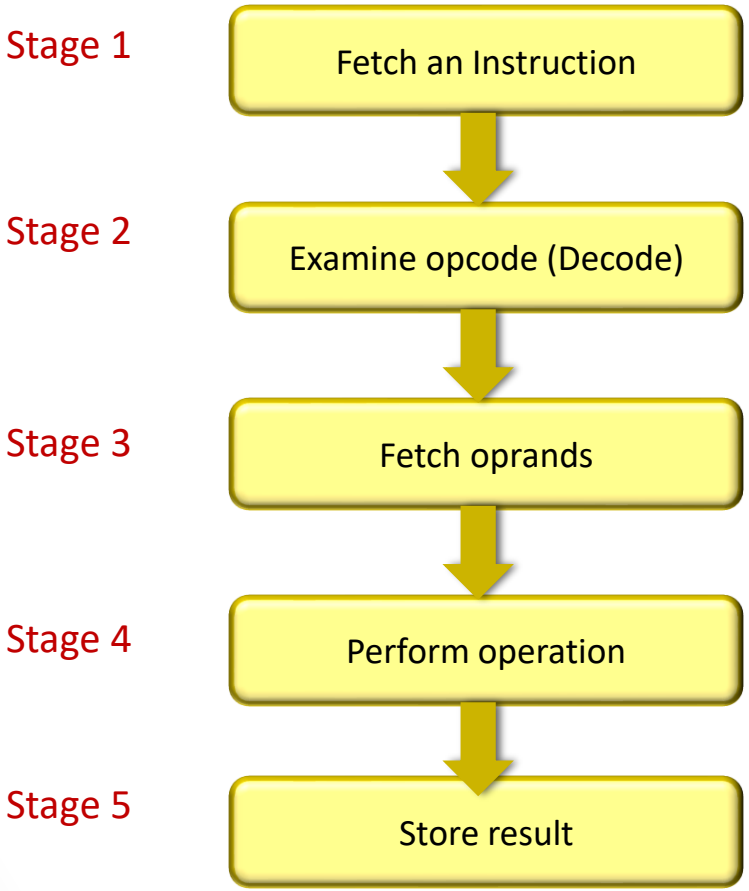
Clock Rate and Instruction Rate

- Generally a faster clock rate will produce higher performance.
- However the fetch-execute cycle doesn't not proceed at a fixed rate given by the clock.
  - The time taken to execute an instruction depends on the operation be performed
  - Some operations require more time (more clock cycles) than the others.

Fetch-Execute Cycle

- Get current instruction, and put it in the Instruction Register (IR)
- Increment the Program Counter (PC)
- Determine type of instruction fetched
- If instruction needs a word from memory, determine where it is
- Fetch the word, if needed, and put it into a register
- Execute the instruction
- Repeat

5 steps in a fetch-execute cycle



A fetch-execute cycle

Solution: Use parallel hardware

Can a CPU executes one instruction in one clock cycle?

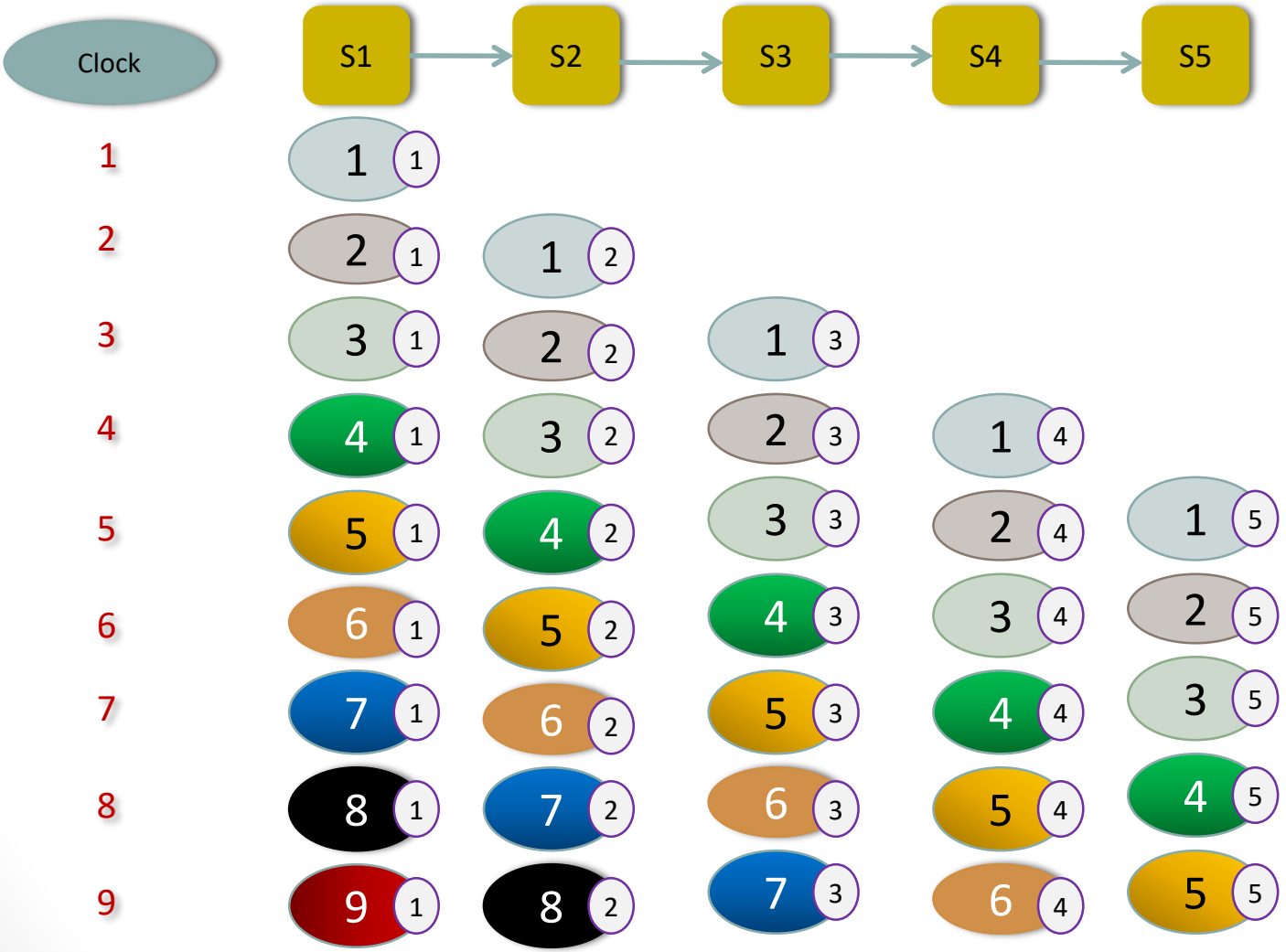
The fact is that executing an instruction takes longer than one clock cycle.

The Multistage Pipelines (Cont.)

- To enable high speed, we use parallel hardware units in a processor, where each unit performs one step

  - Whenever the clock ticks, all stages simultaneously pass the instruction to the right
  - At any time, the pipeline contains 5 instructions
  - One instruction is completed on each clock cycle

The Multistage Pipelines



How to Decide the Operation Sets

- Choosing a set of operation sets represents a complex trade-off
  - The Cost of the hardware
  - The convenience for a programmer
  - Engineering considerations
    - Chip size
    - Power consumption
    - Heat dissipation

Complex and Reduced Instruction Sets (CISC and RISC)

- A **CISC** processor includes a large set (hundreds) of instructions, many of which perform complex computation.
  - Complex instructions are slow
- A **RISC** includes a minimum set of instructions (typically <50)
  - To achieve highest possible speed
    - Fixed-size instruction
    - Is designed to complete one instruction in each clock cycle.

Multi-Core Processor

- One integrated circuit which has two or more processors (called cores)
  - Dual-, quad-, hexa- core etc.
- Implements multiprocessing in a single physical package
- Use message-passing or shared memory inter-core communication methods
- Several tens of cores may require a Network-on-Chip(NoC)
  - Applies networking theory and methods to on-chip communication between cores

Summary

- CPU
  - Fetch-execute cycle
  - Clock and the speed of clock
  - Instruction, instruction format and instruction set
  - Registers
- Tradeoff between Complex and Reduced Instruction Sets (CISC and RISC)
- Higher speeds can be obtained through the use of parallelism
  - Instruction-level pipelining
  - Processor-level
    - Multi-core processor

INTEL ASSEMBLER

Let us now pause at this stage and consider the Assembly code for INTEL processors.