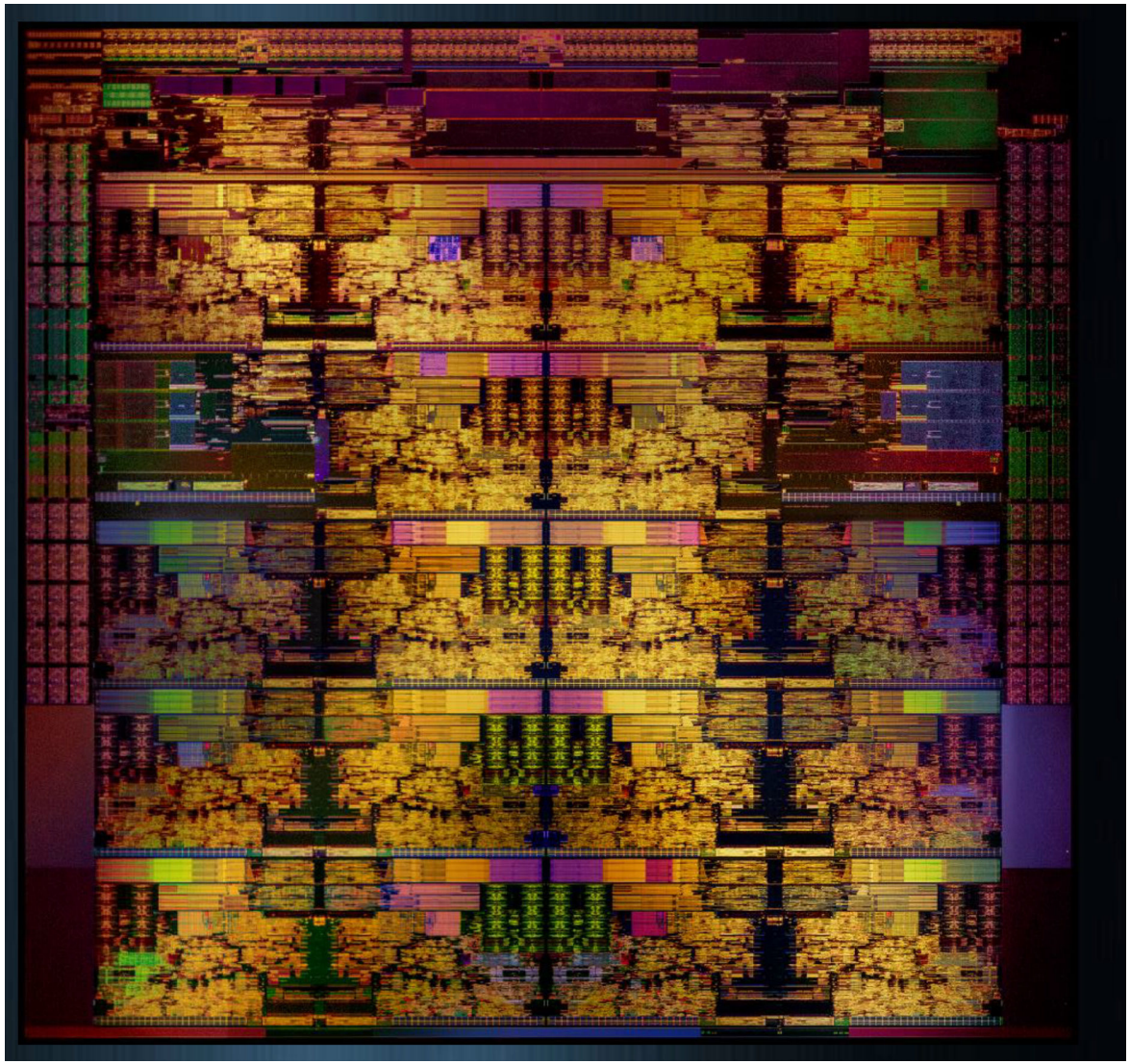
MONASH University
Information Technology

FIT3159
Computer Architecture

Dr Carlo Kopp, SMIEEE, AFAIAA, FLSS, PEng
Carlo.Kopp@monash.edu
Clayton School of Information Technology,
Monash University
Australia

18 (20) x Cores – Intel Skylake X Die (2017)



Why study this in a BSE/BCS degree?

- Performance can be affected by coding style
- Robustness can be affected by coding style
- Provides appreciation of trends and limitations of future hardware
- Proliferation of multicore machines requires knowledge of machine hardware to code effectively
- Interface to physical hardware other than computer itself – embedded realtime applications
- University provides professional education
- Impress your friends with your vast understanding
- Industry wants you to understand this [2010 Survey]

Unit Structure

- **Synopsis: The internal mechanism of computers and how they are organised and programmed. Topics include machine arithmetic, micro-programming, caches, translation look-aside buffers, RISC machines, and pipelined and parallel organisation.**
- **Assessment:**
  - Examination (3 hours): 60%
  - Laboratory/Tutorial work: 40% (5%/1.7%)

Unit Objectives:

- Provide the student with an understanding of the fundamental design features and operating principles of a modern digital computer.
- Provide the student with the ability to understand the implications of various design strategies upon hardware and computer system performance.
- Provide the student with the ability to understand the relationship between the architecture of the machine and its instruction set.
- Provide the student with basic programming skills in assembly (machine) language.

Unit overview

- Introduction and Revision
- Digital logic, combinatorial and sequential
- Machine arithmetic & data representation
- Micro-programming
- Interrupts
- Caches
- Virtual memory & translation look-aside buffers
- CISC vs RISC machines
- Pipelined and parallel organisation
- Superscalar and VLIW architecture

Lab Exercises (Assessed)

- **Lab 1**
  - Combinatorial Logic
- **Lab 2**
  - Sequential Logic
- **Lab 3**
  - Assembly language programming
- **Lab 4**
  - Assembly language programming
- **Lab 5**
  - Assembly language programming
- **Lab 6**
  - Cache performance analysis

Why Lab Exercises?

- **Logic Simulator: Reinforcement of theory taught in lectures**
- **Logic Simulator: Gain appreciation of how digital logic used in machines is designed and tested**
- **Assembly Code: Reinforcement of theory taught in lectures using an emulation of a real processor**
- **Assembly Code: Development of basic skills in Assembler coding**
- **Observations:**
  - Even if you are not developing hardware, understanding complex decision trees in large programs benefits from a good grasp of Boolean algebra and logic techniques;
  - Industry survey in 2010 ranked programming skills in Assembler above skills in Visual Basic, Fortran and COBOL, and about the same as Java language. Writing real time software or performance sensitive applications often requires assembler skills.

Text Books

- **Recommended Reading:**
  - Stallings 8th Ed.
  - Mano & Kime 4th Ed.
  - Tanenbaum A S Structured computer organisation 3rd edn, Prentice-Hall, 1990
  - Or any good book on computer architecture.
  - Please note that all examinable material will be in the unit lecture slides.

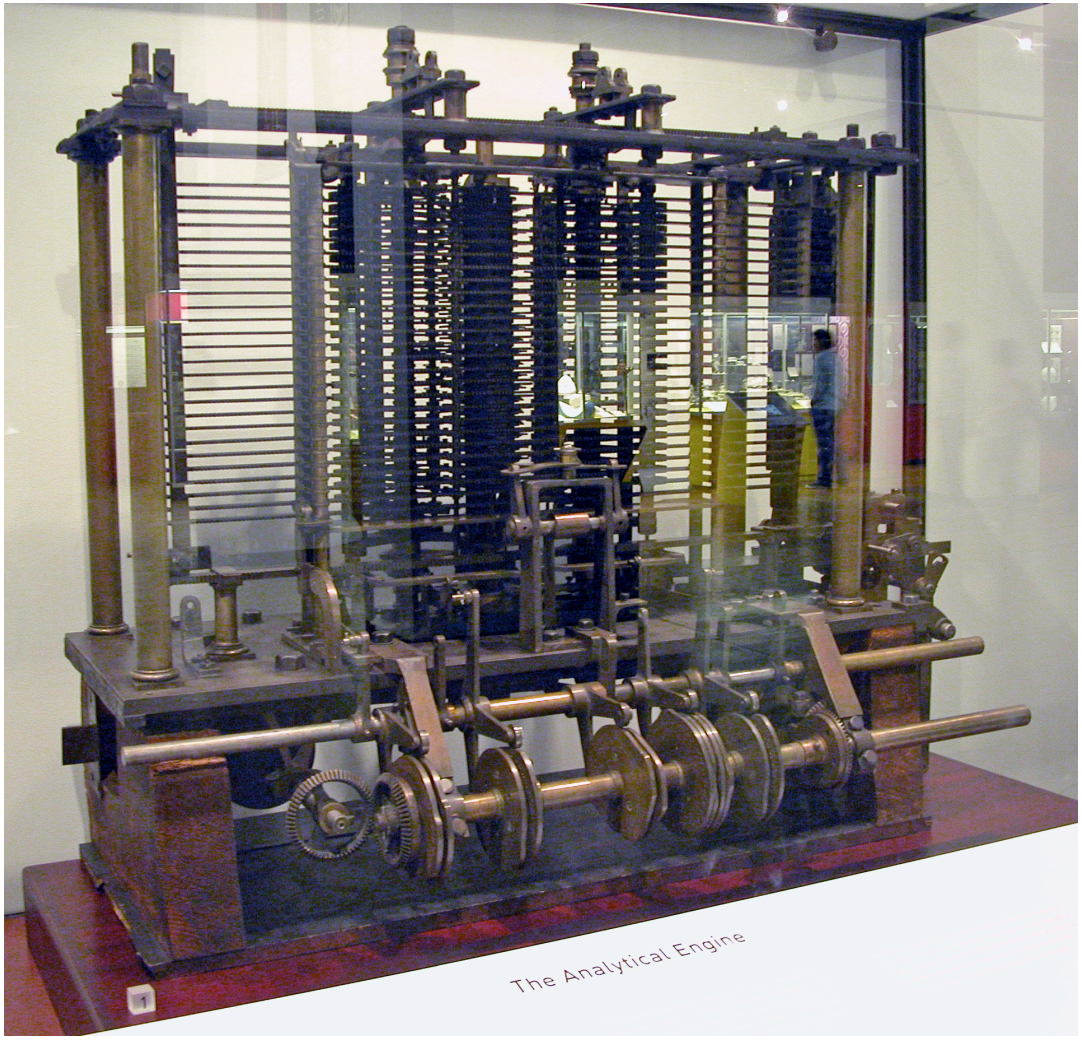
Study Technique

- Computer architecture can be challenging to study because it requires both understanding and retention of knowledge.
- Understanding is best gained and reinforced by attending lectures, asking many questions, and working through examples in the slides and the recommended textbooks.
- The material in this unit is sequential and interdependent; what this means is that to understand *Lecture N*, you have to have some understanding of *Lecture N-1, Lecture N-2* etc.
- Leaving all of the study for the end of semester is a high risk strategy, since a lot of interdependent material has to be learned, absorbed and interconnected in a very short time.
- Lab work involves a logic simulator to provide practice and reinforcement; the Assembly code tasks are also aimed at reinforcing understanding.

A little history …

- **Mechanical Computers - Babbage's Analytical Engine.**
  - Limitations: Slow, Bulky, Unreliable.
- **Electronic Computers - Turing / Zuse / Von Neumann models.**
  - Limited in density, speed and reliability by the switching technology in use.
  - Thermionic Valves and Relays (1940s - 1950s)
  - Discrete Germanium Transistors (1950s)
  - Discrete Silicon Transistors and low density integrated circuits (early 1960s) - TTL, DTL, ECL

Babbage's Analytical Engine – London Science Museum



Limitations:

Slow – all mechanical components

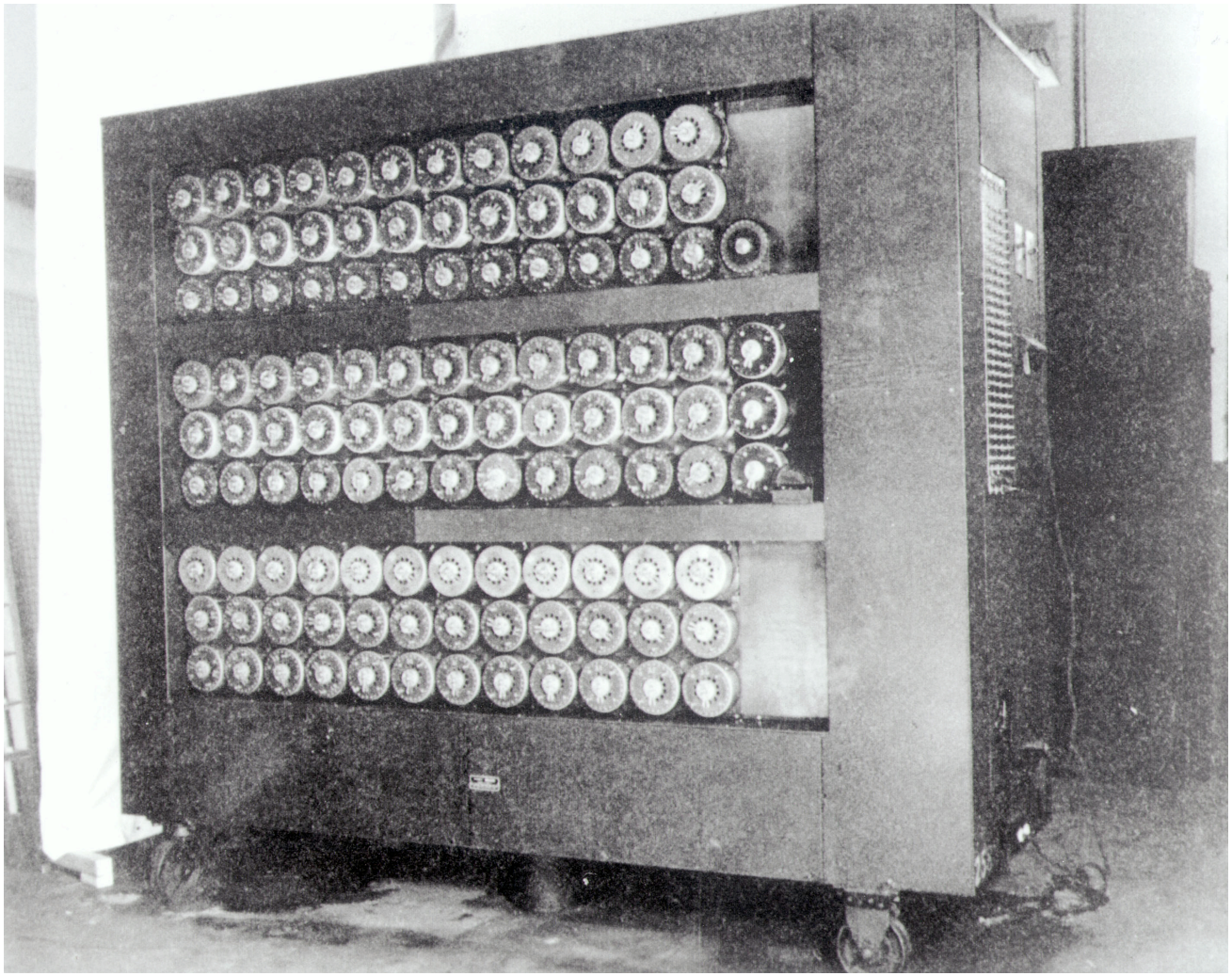
Bulky – as depicted

Unreliable – mechanical components subject to wear, lubricants required

Programs and data were held on Paper punch cards

Bruno Barral https://commons.wikimedia.org/wiki/File:AnalyticalMachine_Babbage_London.jpg

Turing's Bombe – Bletchley Park



Limitations:

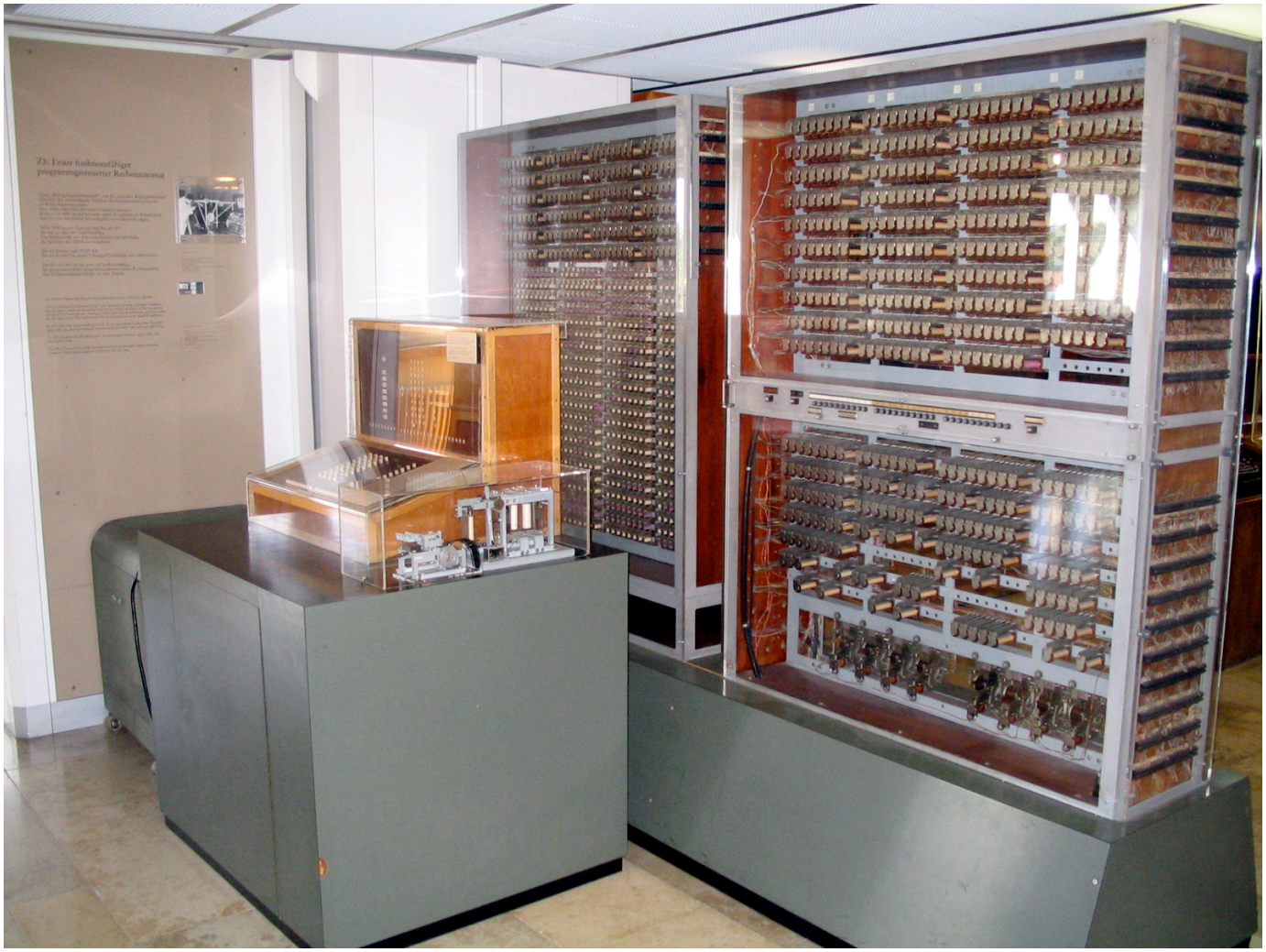
Slow – electro-mechanical components

Bulky – as depicted

Unreliable – mechanical components
subject to wear

http://www.bombe.org.uk/slide-show/

Zuse's Z3 Computer - Deutsches Museum in Munich



Limitations:

Slow – electro-mechanical components borrowed from a telephone switch

Bulky – as depicted

Unreliable – mechanical components subject to wear

https://en.wikipedia.org/wiki/Z3_(computer)

A little history …

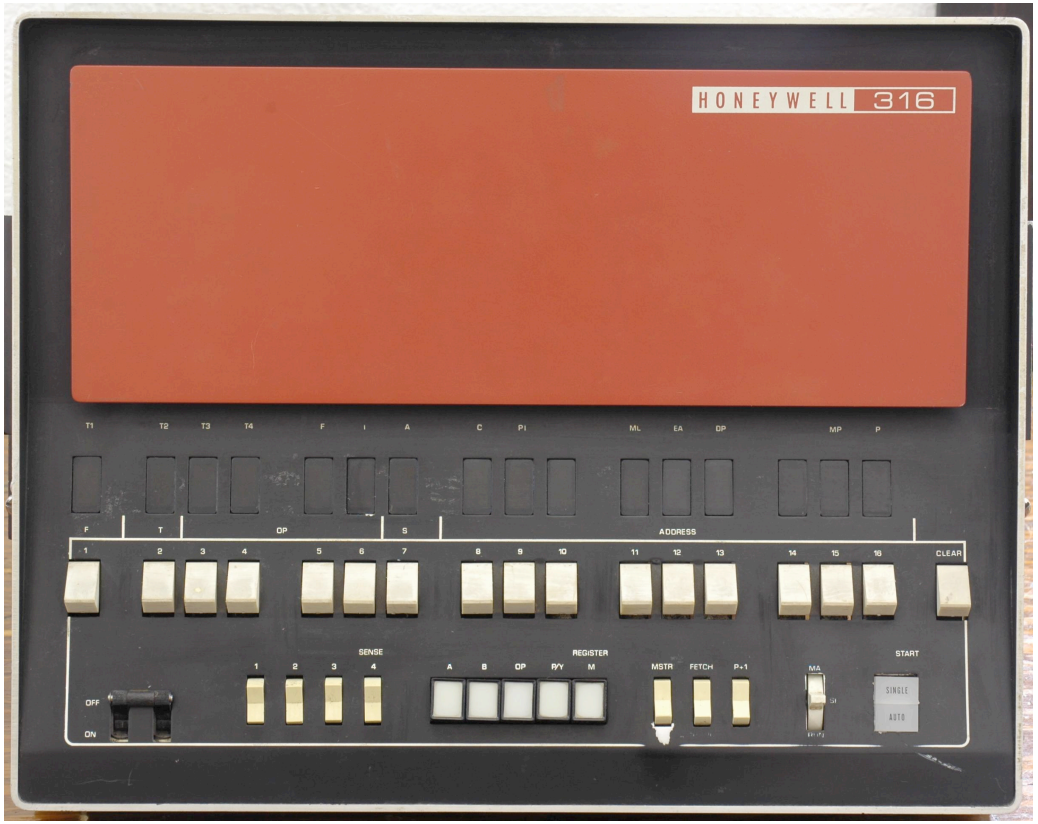
- **Electronic Computers (Cont …)**
  - Medium Scale Integration (1970s)- S-TTL, CMOS, NMOS, MPU
  - (Very) Large Scale Integration (1980s)- LS-TTL, CMOS, NMOS
  - 1990s - Microprocessors largely displace the minicomputer.
  - By Y2K, CMOS dominates, with millions of transistors per single Silicon die.
  - By 2010, CMOS technology multicore chips predominate, with 2 or 4 processors most common.
  - Future trends involve “exotic” materials and increased density and numbers of processing cores ….

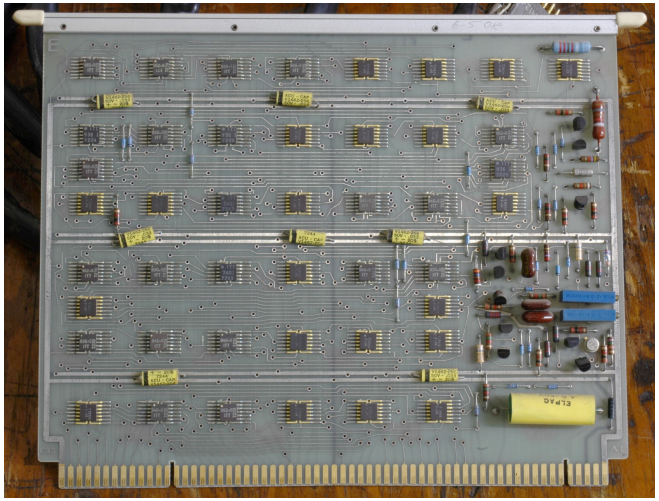
Digital Equipment Corp PDP-8 – 1965 “Minicomputer”

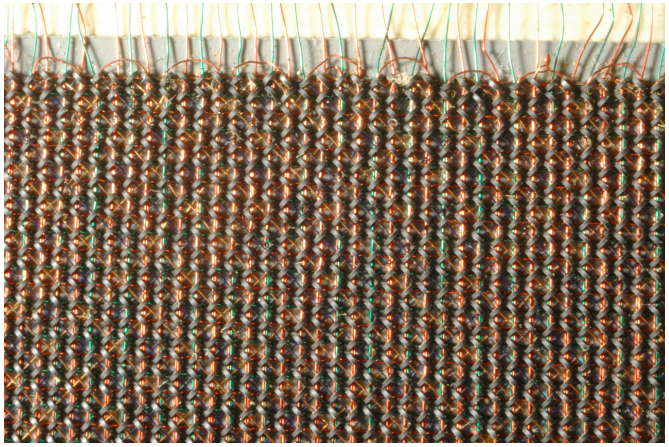
Limitations: Slow – TTL Logic, Bulky – as depicted, Unreliable – electrical connectors, cooling fans, chip failures

Online PDP-8 Home Page, Run a PDP-8: https://www.pdp8.net/

Honeywell 316 – 1969 “Embedded Minicomputer”



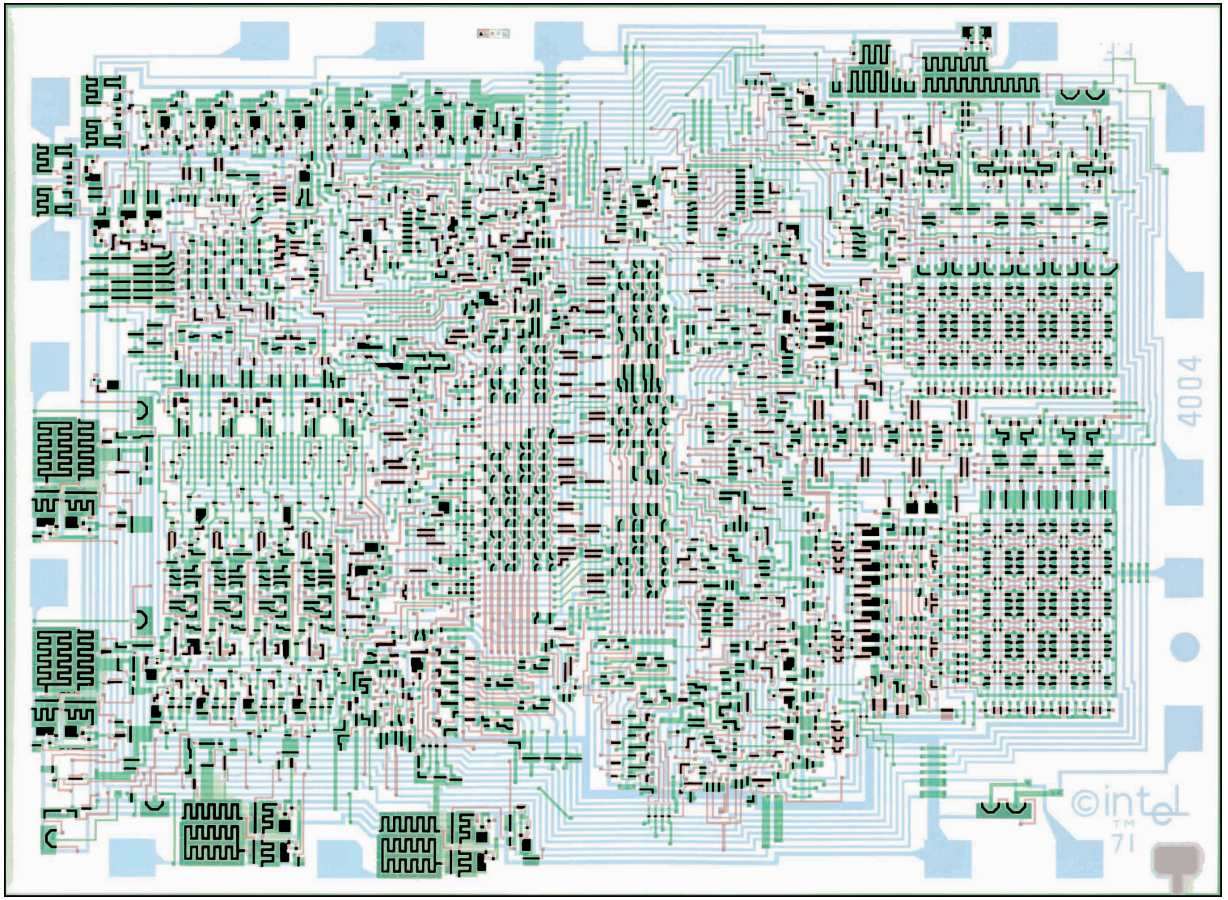


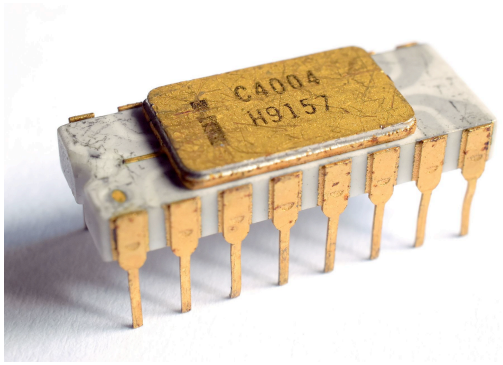


Limitations: Slow – TTL Logic, Bulky – as depicted, Unreliable – electrical connectors, cooling fans, chip failures

The Retro-Computing Society of Rhode Island, Inc.: https://www.rcsri.org/collection/honeywell-316/

Intel 4004 – 1971 “First Microprocessor”



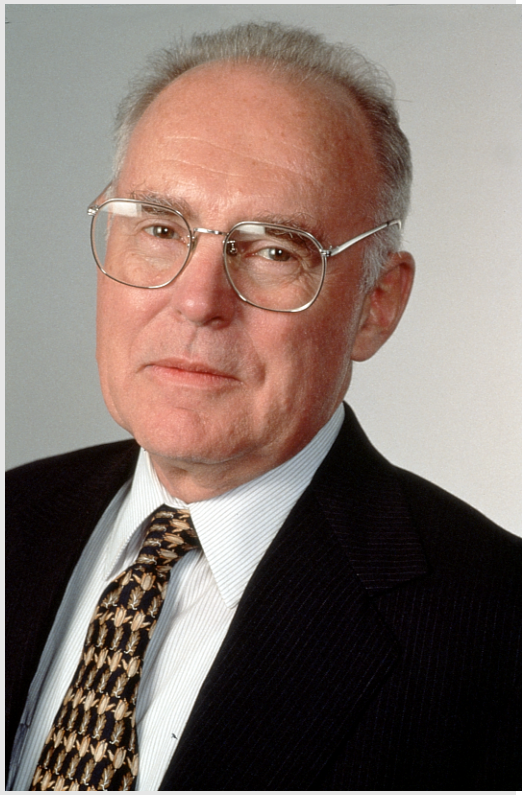


Limitations: Slow – TTL Logic 0.74 MHz Clock, 12-bit addresses, 8-bit instructions, 4-bit data words

https://en.wikipedia.org/wiki/Intel_4004

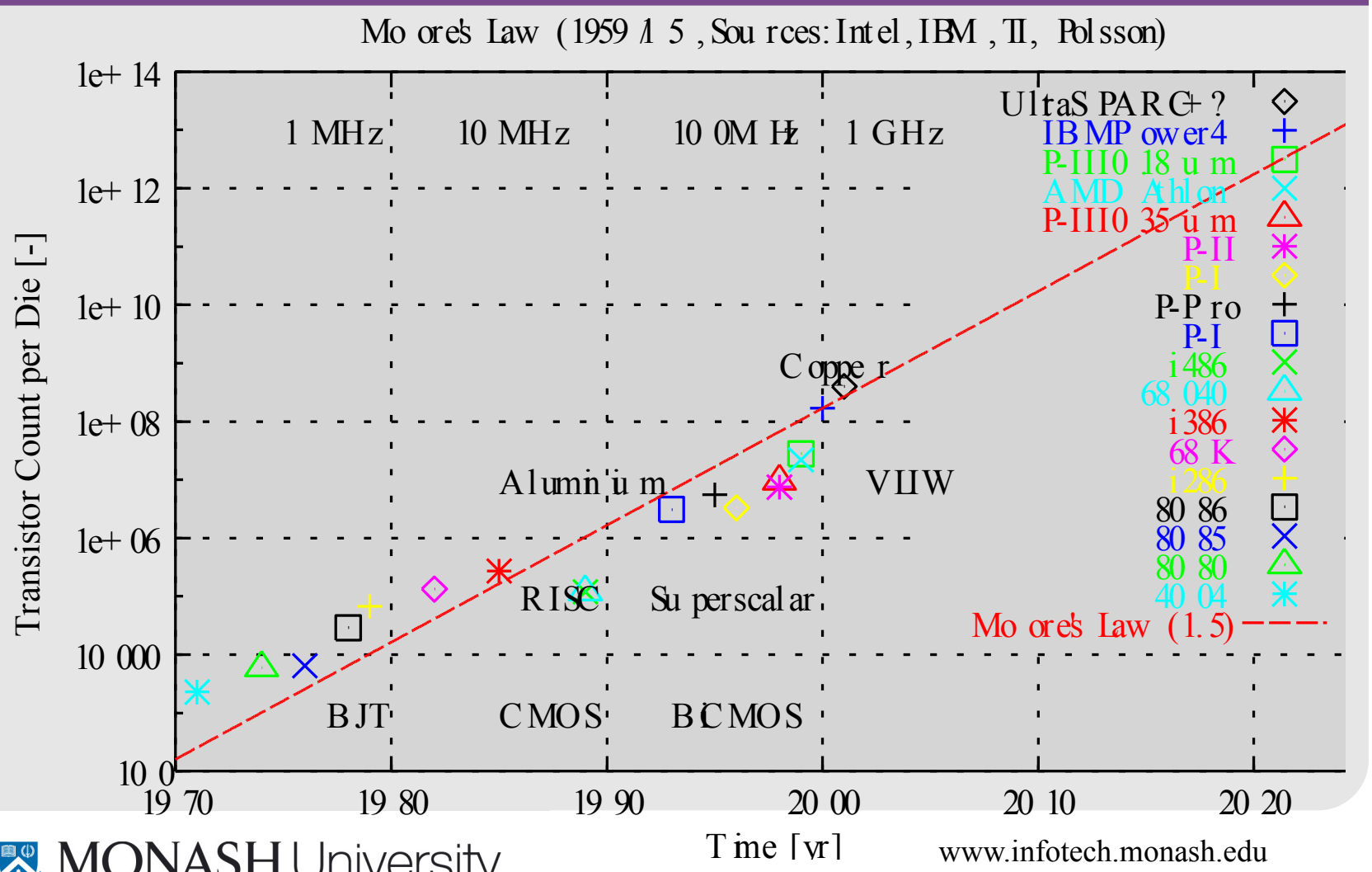
Moore's *Law*

- **Moore co-founded Intel in 1968, serving initially as Executive Vice President.**
- **Moore is widely known for "Moore's Law,"**
  - he predicted that the number of transistors the industry would be able to place on a computer chip would double every couple of years.
- **In 1995, he updated his prediction to once every two years.**
- **While originally intended as a rule of thumb in 1965, it has become the guiding principle for the industry to deliver ever-more-powerful semiconductor chips at proportionate decreases in cost.**



http://www.intel.com/pressroom/kits/bios/moore.htm

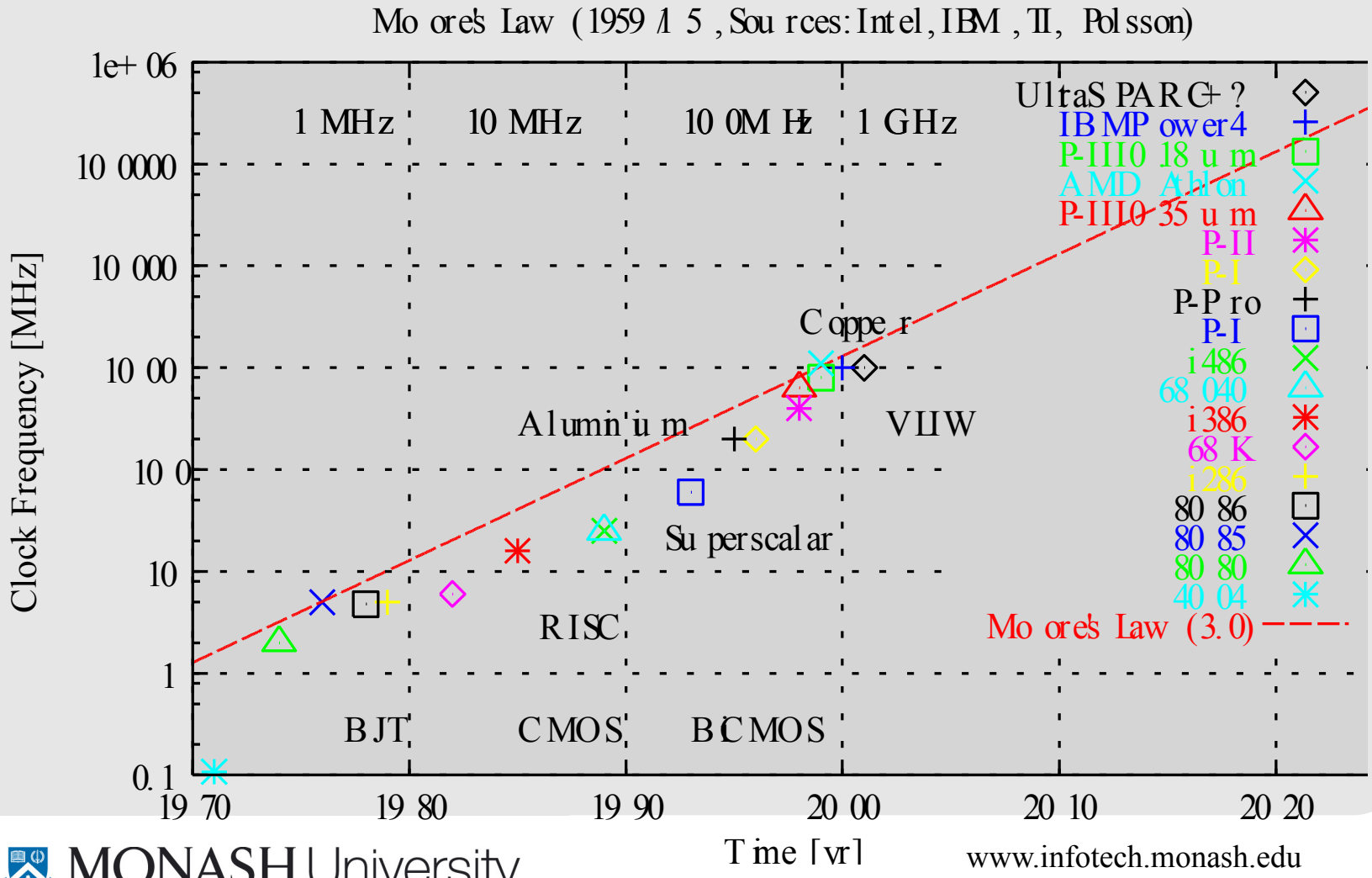
Moore's *Law* - Chip Density



www.infotech.monash.edu

MONASH University
Information Technology

Moore's *Law* - Chip Speed



MONASH University
Information Technology

What is Exponential Growth?

- **Wikipedia Definition:** ***"Exponential growth occurs when the growth rate of the value of a mathematical function is proportional to the function's current value."***

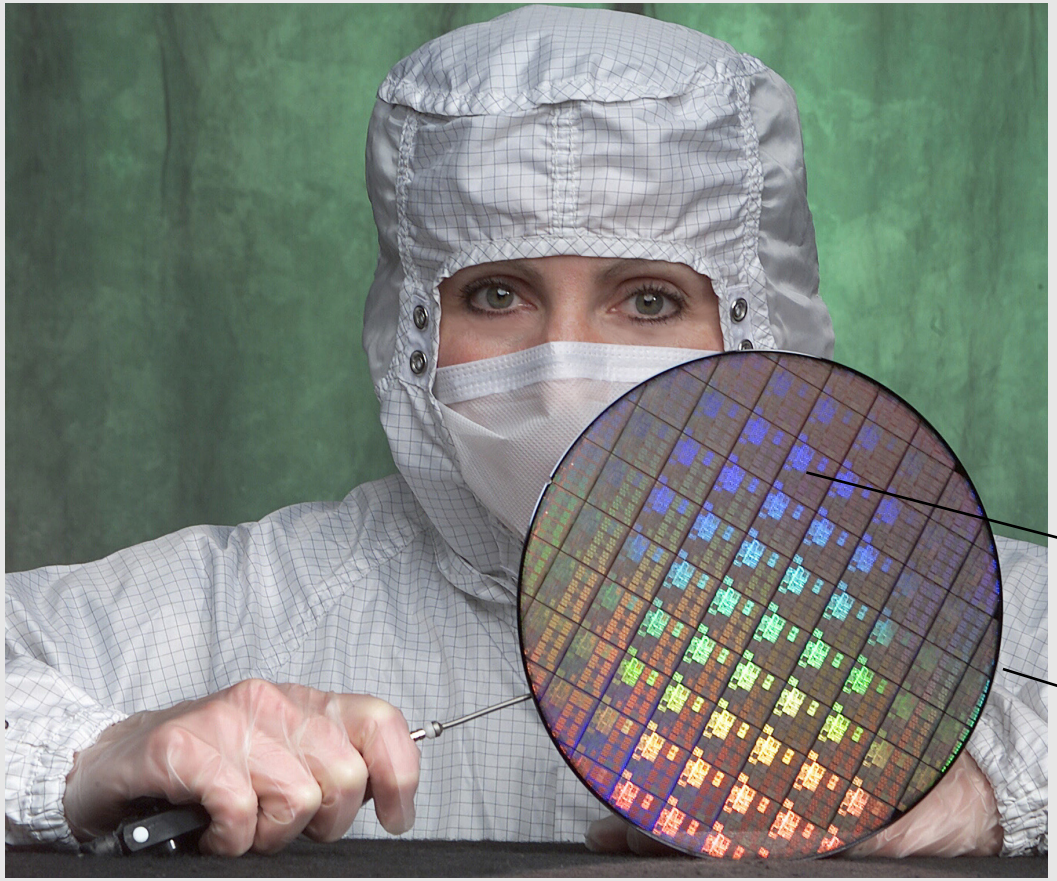
$$x_t = x_0 (1 + r)^t$$

- **Where x_0 is the value at initial time $t=0$;**
- **Commonly used forms to represent exponential growth functions include:**

$$x(t) = x(0).e^{kt} = x(0).e^{t/\tau}$$

- **Where k and τ are constants constraining the rate of growth over time;**
- **Often graphically represented as a straight line function on a logarithmic scale;**
- **Punchline:** ***Empirically observed effect in semiconductor monolithic technologies, and magnetic storage technologies – always bounded by the physics of the technology employed, and usually very "noisy" functions. Should not be called a "law", but rather an "effect".***

What's in a chip?



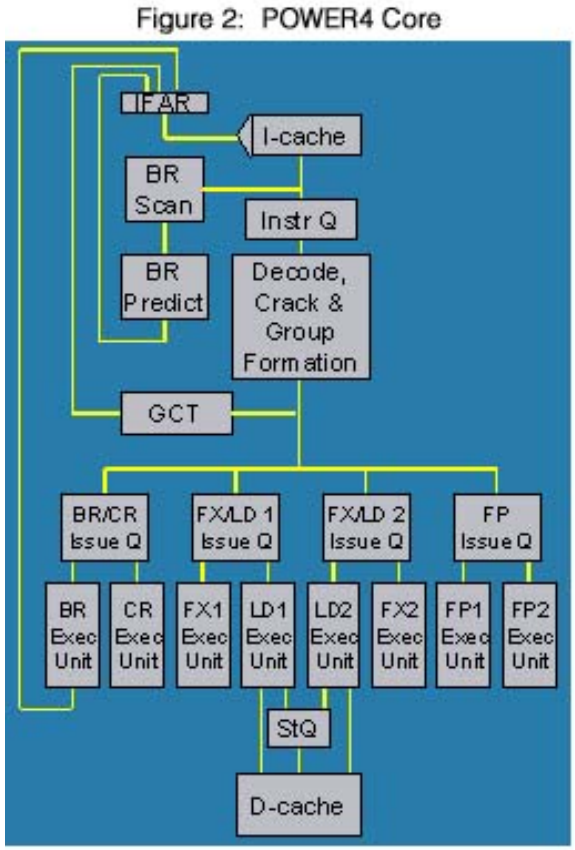
Photolithography used to etch circuits and components on to chip dies

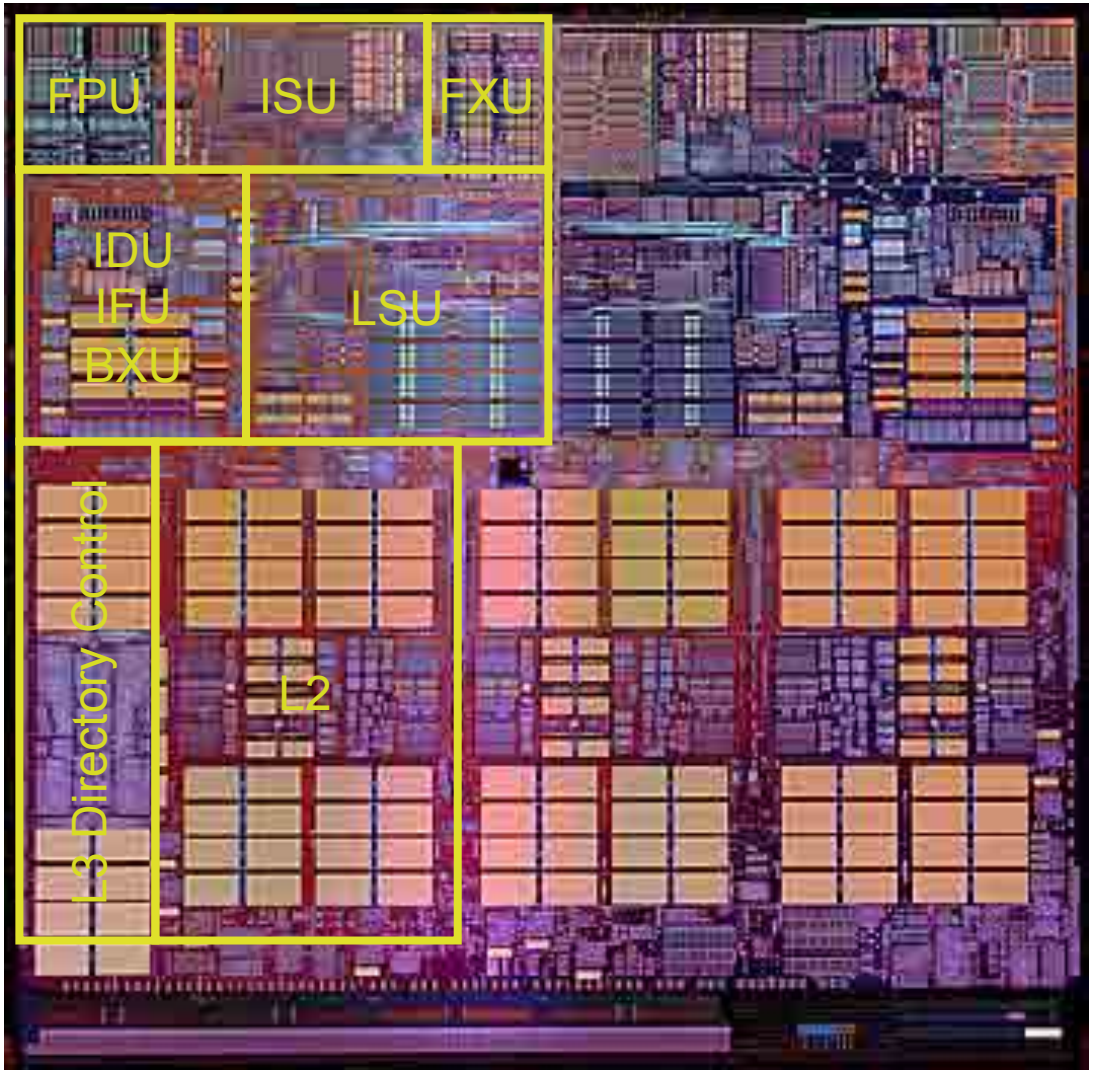
Chip Die

Wafer with multiple dies

Source IBM

Power 4 (PowerPC) Architecture



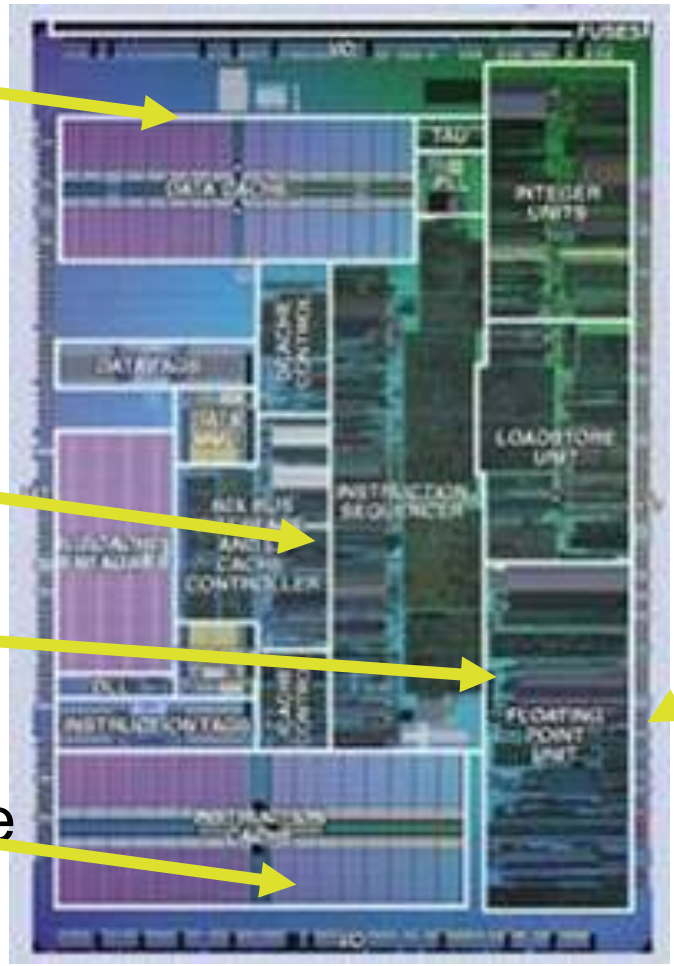


MONASH University
Information Technology

Source IBM

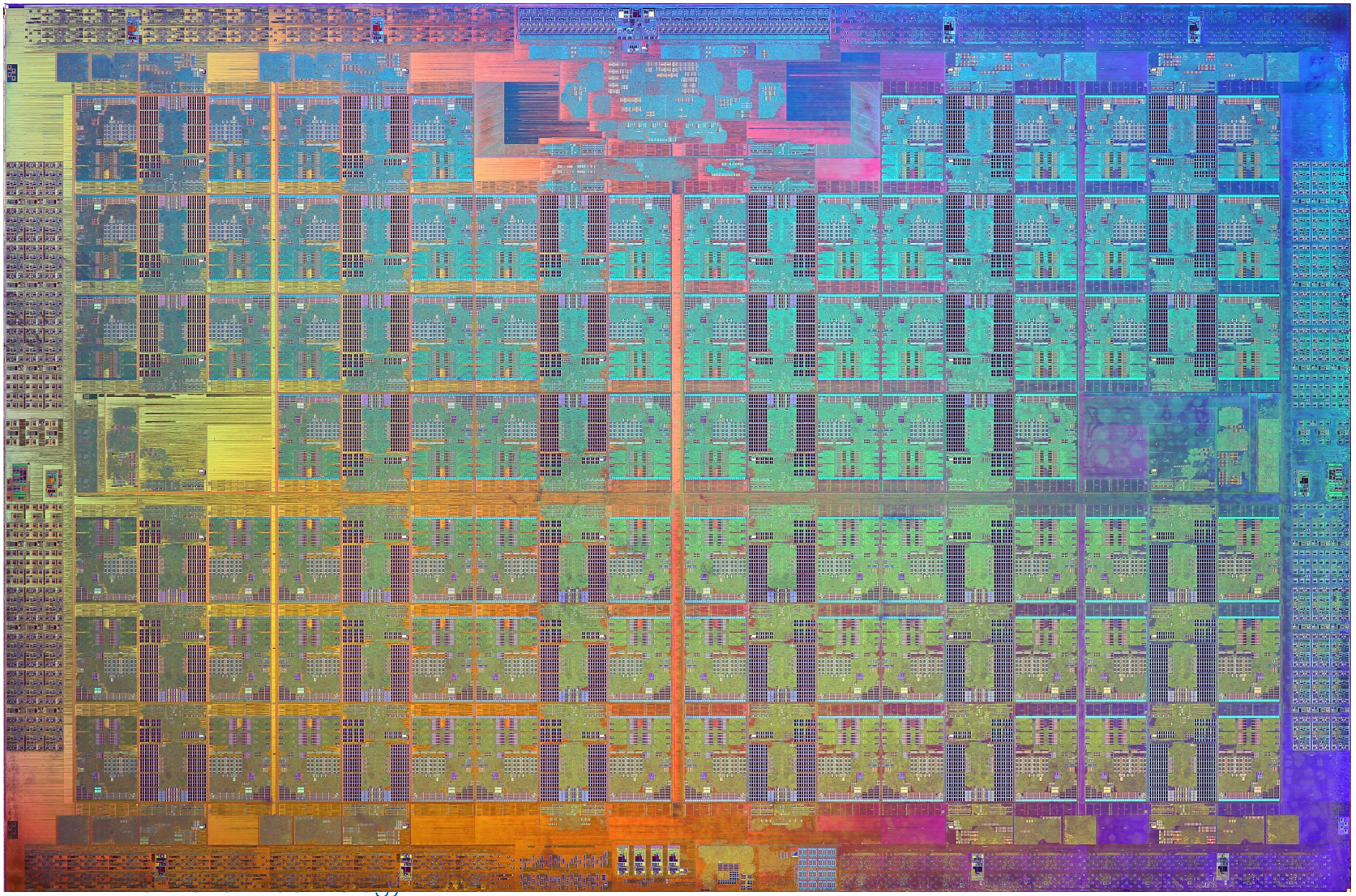
www.infotech.monash.edu

IBM/Motorola G3 Chip Layout



Source IBM

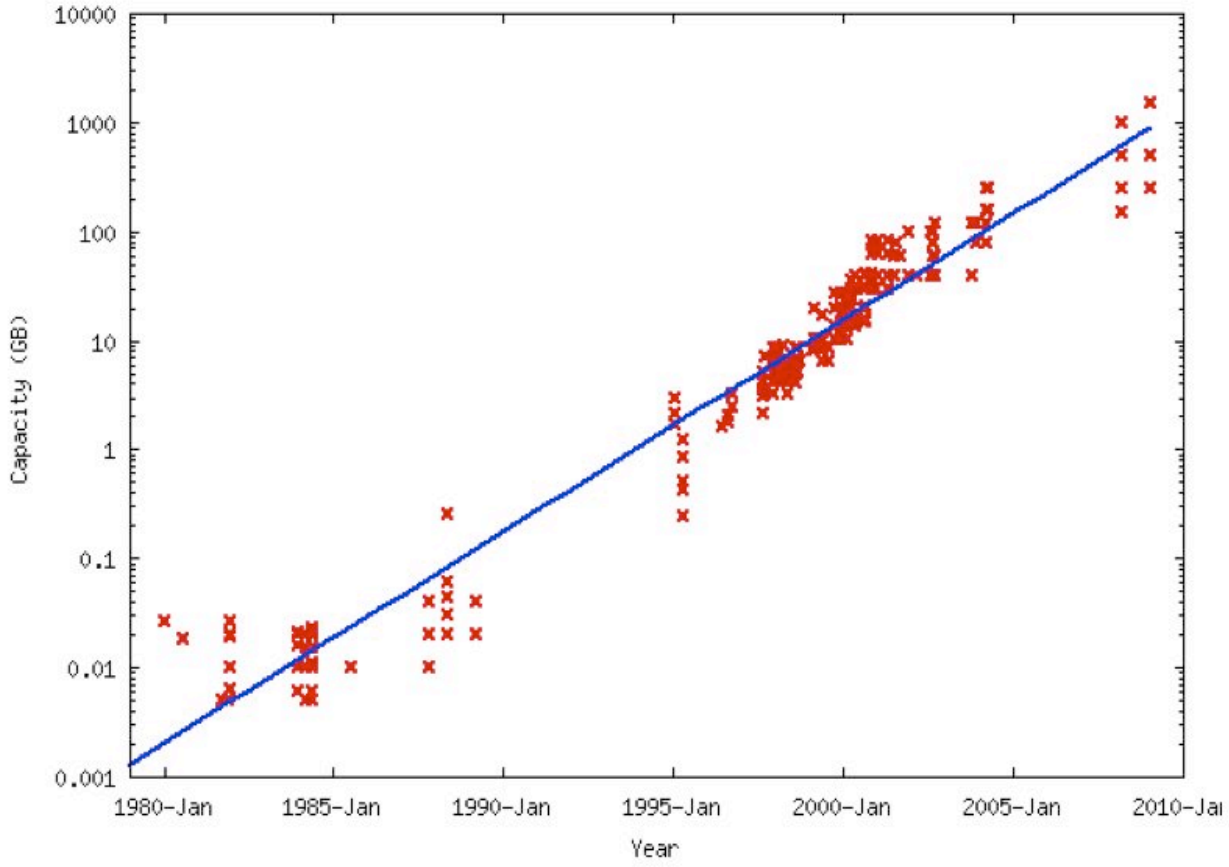
72 x Cores – Intel Xeon Phi / Knight's Landing (2016)



Exponentially Growing Mass Storage Technology

- Rotating "hard disks" continue to grow in surface storage density following "Kryder's Law", exponentially;
- While the density of rotating disks and thus transfer rate grow exponentially, improvements in access times / latency are limited due to mechanical rotation and head movements (storage topics);
- New technologies will extend "Kryder's Law", although physics bounds are being approached;
- Solid State Disks (SSD) using "Flash RAM" technology, common to USB thumbdrives and SDHC modules, are now becoming available at affordable prices of < $1/Gigabyte;
- Within the physics bounds of Flash RAM technology, SSDs will also follow the "Moore's Law" exponential growth curve;

Kryder's Law (Storage Capacity vs. Time)



SATA Bus Variant Comparison / Kryder's Law

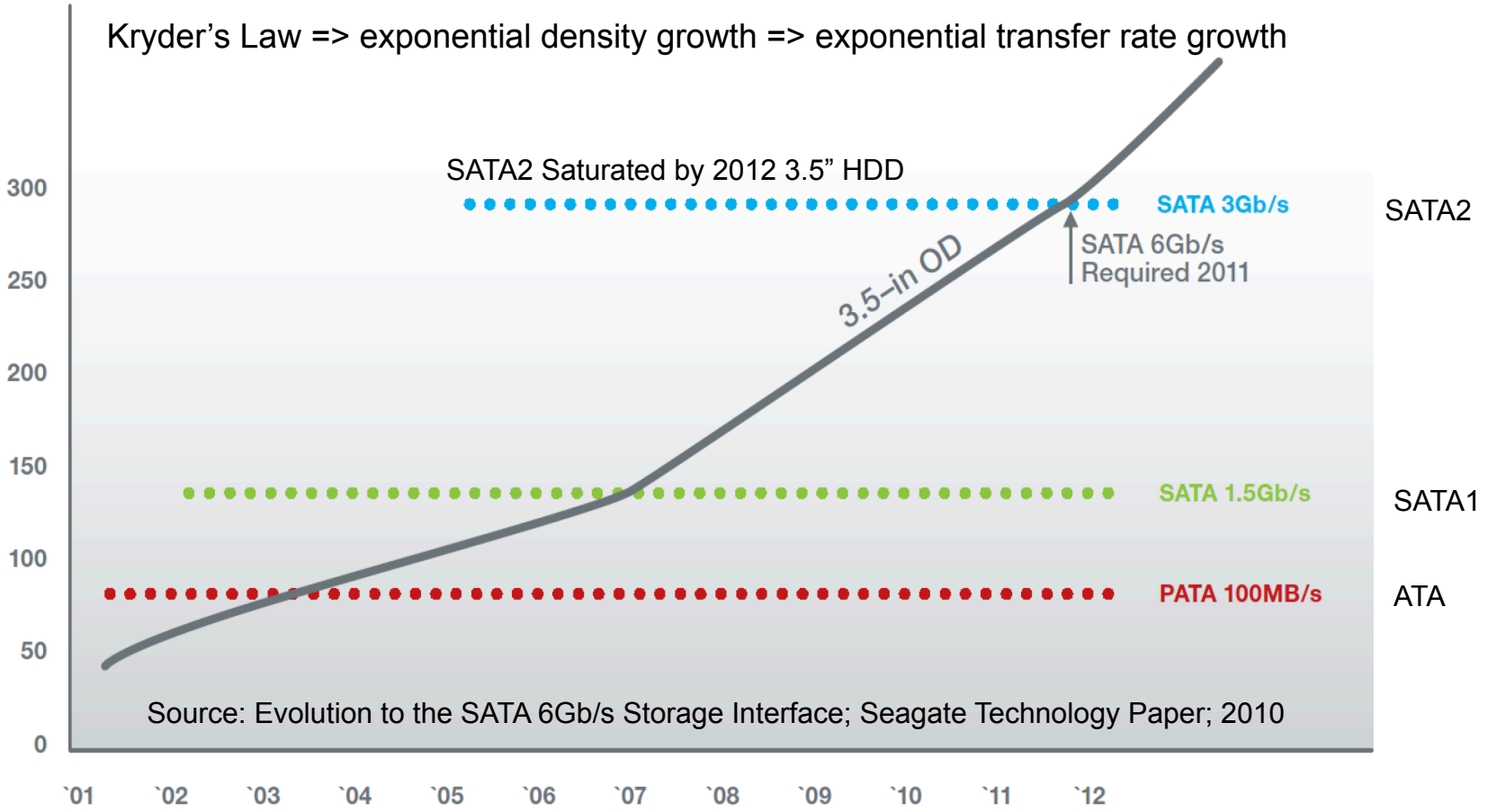


Figure 1. Sustained Data Transfer Rate Estimates

Architecture vs Implementation

- The "logical architecture and organisation" of a machine is, in principle, independent of the technology we use to implement it;
- *We could, in principle, implement the same machine architecture in a range of different types of digital logic, e.g. NMOS, CMOS or even ECL;*
- In practice the "logical architecture" is usually adapted in a manner which makes it most convenient for the hardware type we intend to use;
- Contemporary processor chips mostly use CMOS technology, due to its low power consumption compared to other logic chip families;
- While Silicon is most widely used, "exotic" materials are now being introduced.

Summary

- **Understanding computer architecture helps us design programs which are**
  - More efficient in resource use
  - Execute faster if needed
  - Execute more reliably if needed
  - Can interface to physical systems other than the computer itself
- **Computer technology evolves over time.**
- **Basic technology evolves over time.**
- **Moore's Law shows exponential growth over time.**
- **Computer performance tracks Moore's Law.**
- **Hard disk performance tracks Kryder's Law.**

MONASH University
Information Technology

FIT3159
Computer Architecture

Digital Logic Revision / Binary Numbers / Boolean Algebra
Ch.2 Mano & Kime

Logic and Computer Design Fundamentals, 4e, PPT Slides

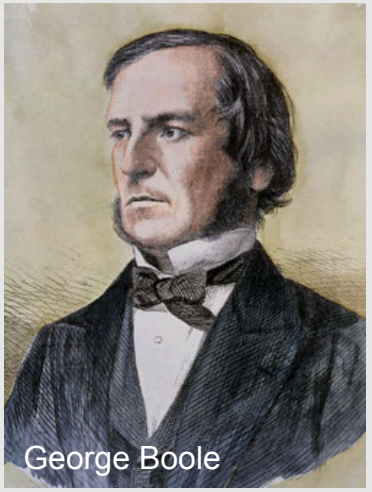
Charles Kime & Thomas Kaminski

Why should we understand Boolean logic?

- Computers are built up from digital logic circuits and components;
- Understanding how various parts of a computer work requires an understanding of Boolean logic;
- Understanding how data is represented, and how machine arithmetic is performed requires an understanding of Boolean logic – relevant for understanding errors in machine arithmetic and how to overcome or manage them (Example – test conditions in programs used for control flow);
- Optimising control flow in programs benefits from understanding Boolean logic;
- Interfacing other hardware to computers requires an understanding of Boolean logic;

George Boole (1815 – 1864) [N/E]

- George Boole was a British (Irish) philosopher and mathematician;
- *"Mathematical Analysis of Logic"* - 1847;
- *"An Investigation of the Laws of Thought, on Which are Founded the Mathematical Theories of Logic and Probabilities"* – 1854;
- Augustus De Morgan (1806 – 1871) extended Boole's work with "De Morgan's Laws" and "mathematical induction";
- William Stanley Jevons (1835 – 1882) popularised Boolean logic in the "*Elementary Lessons on Logic*"
- *Claude Shannon later (1946) used Boole's work in the development of information theory used in digital communications.*


George Boole


Augustus De Morgan

Binary Logic and Gates

- Binary variables take on one of two values.
- Logical operators operate on binary values and binary variables.
- Basic logical operators are the logic functions AND, OR and NOT.
- Logic gates implement logic functions.
- Boolean Algebra: a useful mathematical system for specifying and transforming logic functions.
- *We study Boolean algebra as a foundation for designing and analyzing digital systems!*

Binary Variables

- **Recall that the two binary values have different names:**
  - **True/False**
  - **On/Off**
  - **Yes/No**
  - **1/0**
- **We use 1 and 0 to denote the two values.**
- **Variable identifier examples:**
  - **A, B, y, z, or X_1 for now**
  - **RESET, START_IT, or ADD1 later**

Logical Operations

- The three basic logical operations are:
  - AND
  - OR
  - NOT
- AND is denoted by a dot (·).
- OR is denoted by a plus (+).
- NOT is denoted by an overbar ($\bar{}$), a single quote mark (') after, or (~) before the variable.

Notation Examples

- **Examples:**
  - $Y = A \cdot B$ is read "Y is equal to A AND B."
  - $z = x + y$ is read "z is equal to x OR y."
  - $X = \overline{A}$ is read "X is equal to NOT A."

- **Note: The statement:**

  1 + 1 = 2 (read "one plus one equals two")

  is not the same as

  1 + 1 = 1 (read "1 or 1 equals 1").

Operator Definitions

- Operations are defined on the values "0" and "1" for each operator:

AND	OR	NOT
$0 \cdot 0 = 0$	$0 + 0 = 0$	$\overline{0} = 1$
$0 \cdot 1 = 0$	$0 + 1 = 1$	$\overline{1} = 0$
$1 \cdot 0 = 0$	$1 + 0 = 1$	
$1 \cdot 1 = 1$	$1 + 1 = 1$	

Truth Tables

- *Truth table* – a tabular listing of the values of a function for all possible combinations of values on its arguments
- Example: Truth tables for the basic logic operations:

AND		
X	Y	Z = X·Y
0	0	0
0	1	0
1	0	0
1	1	1

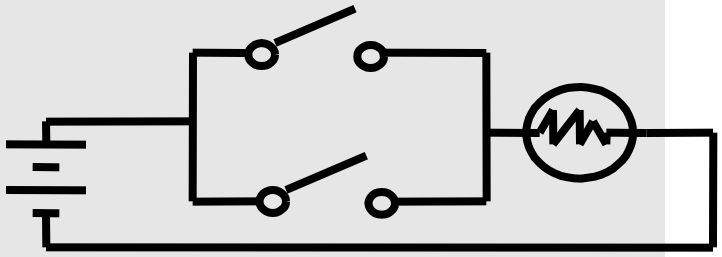
OR		
X	Y	Z = X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT	
X	$Z = \overline{X}$
0	1
1	0

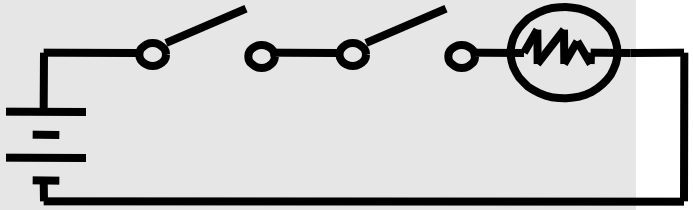
Logic Function Implementation

- **Using Switches**
  - For inputs:
    - > logic 1 is switch closed
    - > logic 0 is switch open
  - For outputs:
    - > logic 1 is light on
    - > logic 0 is light off.
  - NOT uses a switch such that:
    - > logic 1 is switch open
    - > logic 0 is switch closed

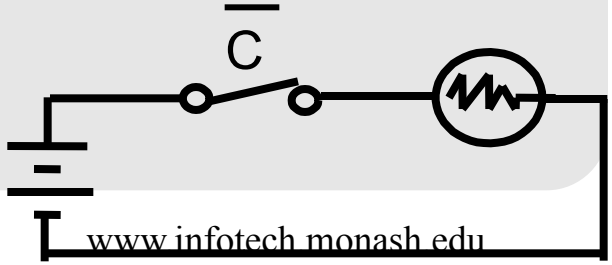
Switches in parallel => OR



Switches in series => AND

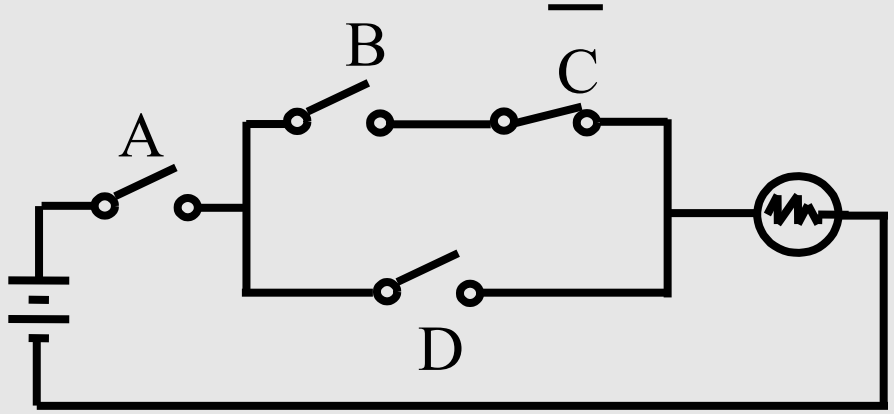


Normally-closed switch => NOT



Logic Function Implementation (Continued)

- **Example: Logic Using Switches**



- **Light is on (L = 1) for**

  $L(A, B, C, D) = A\bar{B}C + A\bar{B}D$

  and off (L = 0), otherwise.

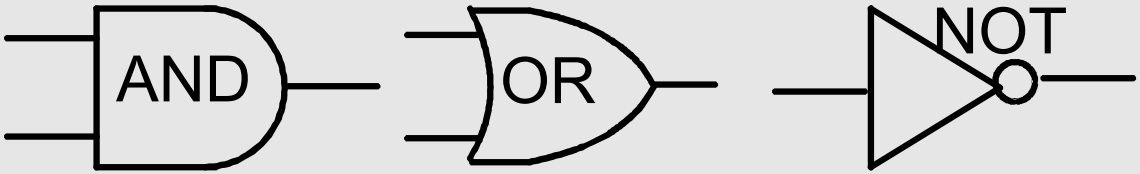
- **Useful model for relay circuits and for CMOS gate circuits, the foundation of current digital logic technology**

Logic Gates

- In the earliest computers, switches were opened and closed by magnetic fields produced by energizing coils in *relays*. The switches in turn opened and closed the current paths.
- Later, *vacuum tubes* that open and close current paths electronically replaced relays.
- Today, *transistors* are used as electronic switches that open and close current paths.
- Many different transistor types have been used, current preference is CMOS due low power drain

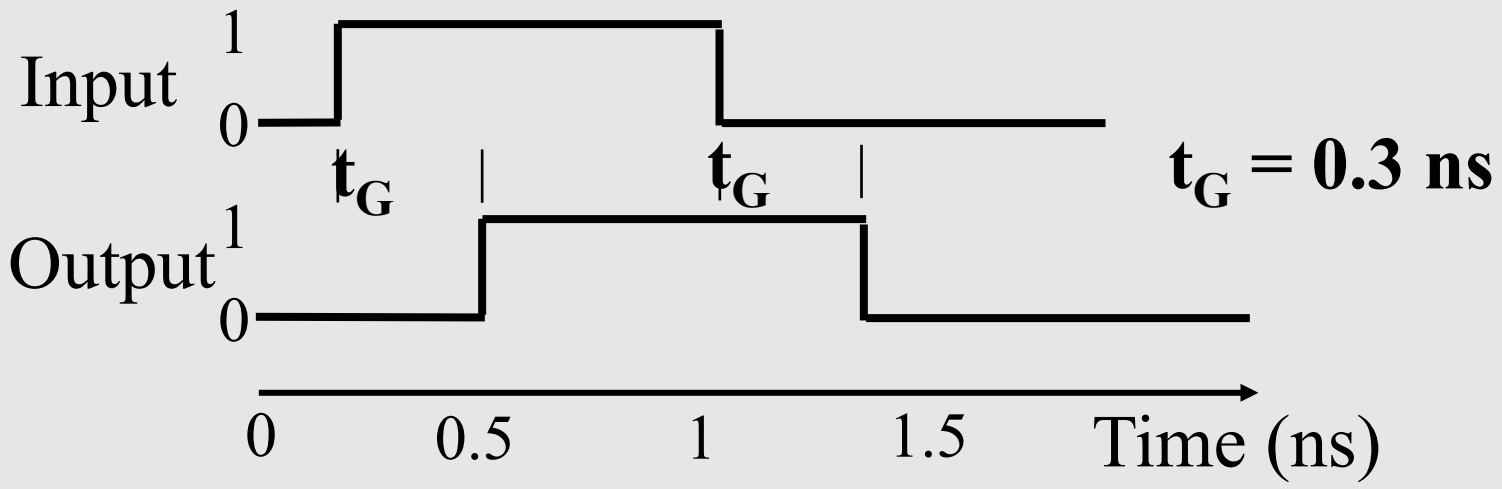
Logic Gate Symbols and Behavior

- **Logic gates have special symbols:**



Gate Delay

- In actual physical gates, if one or more input changes causes the output to change, the output change does not occur instantaneously.
- The delay between an input change(s) and the resulting output change is the *gate delay* denoted by t_G:



Logic Diagrams and Expressions

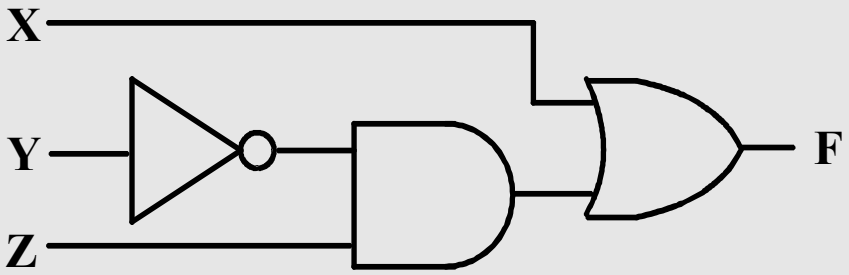
Truth Table

X Y Z	$F = X + \overline{Y} \cdot Z$
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	0
1 0 0	1
1 0 1	1
1 1 0	1
1 1 1	1

Equation

$$F = X + \overline{Y}\,Z$$

Logic Diagram



- **Boolean equations, truth tables and logic diagrams describe the same function!**
- **Truth tables are unique; expressions and logic diagrams are not. This gives flexibility in implementing functions.**

Boolean Algebra

- **An algebraic structure defined on a set of at least two elements, B, together with three binary operators (denoted +, · and ¯) that satisfies the following basic identities:**

1.	$X + 0 = X$	2.	$X \cdot 1 = X$
3.	$X + 1 = 1$	4.	$X \cdot 0 = 0$
5.	$X + X = X$	6.	$X \cdot X = X$
7.	$X + \overline{X} = 1$	8.	$X \cdot \overline{X} = 0$
9.	$\overline{\overline{X}} = X$		

10.	$X + Y = Y + X$	11.	$XY = YX$	Commutative
12.	$(X + Y) + Z = X + (Y + Z)$	13.	$(XY) Z = X(Y Z)$	Associative
14.	$X(Y + Z) = XY + XZ$	15.	$X + YZ = (X + Y) (X + Z)$	Distributive
16.	$\overline{X + Y} = \overline{X} \cdot \overline{Y}$	17.	$\overline{X \cdot Y} = \overline{X} + \overline{Y}$	DeMorgan's

Some Properties of Identities & the Algebra

- If the meaning is unambiguous, we leave out the symbol "·"
- The identities above are organized into pairs. These pairs have names as follows:

  1-4 Existence of 0 and 1 5-6 Idempotence

  7-8 Existence of complement 9 Involution

  10-11 Commutative Laws 12-13 Associative Laws

  14-15 Distributive Laws 16-17 DeMorgan's Laws

- The dual of an algebraic expression is obtained by interchanging + and · and interchanging 0's and 1's.
- The identities appear in dual pairs. When there is only one identity on a line the identity is self-dual, i. e., the dual expression = the original expression.

Boolean Operator Precedence

- **The order of evaluation in a Boolean expression is:**

  1. Parentheses
  2. NOT
  3. AND
  4. OR

- **Consequence: Parentheses appear around OR expressions**
- **Example:** $F = A(B + C)(C + \overline{D})$

Definitions

- What is a "minterm"?
- A minterm is an expression which is the logical AND or logical "product"of two or more Boolean variables, or the complements of these variables.
- Examples: A.B.C, A'.B.C or AB'C etc
- What is a "maxterm"?
- A maxterm is an expression which is the logical OR or logical "sum"of two or more Boolean variables, or the complements of these variables.
- Examples: A+B+C, A'+B+C, A+B'+C etc
- The terms "minterm" and "maxterm" appear frequently in textbooks or other literature dealing with Boolean logic or algebra.

Useful Theorems

$x \cdot y + \bar{x} \cdot y = y$	$(x + y)(\bar{x} + y) = y$	**Minimization**
$x + x \cdot y = x$	$x \cdot (x + y) = x$	**Absorption**
$x + \bar{x} \cdot y = x + y$	$x \cdot (\bar{x} + y) = x \cdot y$	**Simplification**
$x \cdot y + \bar{x} \cdot z + y \cdot z = x \cdot y + \bar{x} \cdot z$		**Consensus**
$(x + y) \cdot (\bar{x} + z) \cdot (y + z) = (x + y) \cdot (\bar{x} + z)$		
$\overline{x + y} = \bar{x} \cdot \bar{y}$	$\overline{x \cdot y} = \bar{x} + \bar{y}$	**DeMorgan's Laws**

Boolean Function Evaluation

$F1 = xy\bar{z}$

$F2 = x + \bar{y}z$

$F3 = \bar{x}\bar{y}\bar{z} + \bar{x}\,y\,z + x\bar{y}$

$F4 = x\bar{y} + \bar{x}\,z$

x	y	z	F1	F2	F3	F4
0	0	0	0	0		
0	0	1	0	1		
0	1	0	0	0		
0	1	1	0	0		
1	0	0	0	1		
1	0	1	0	1		
1	1	0	1	1		
1	1	1	0	1		

Summary

- Binary numbers
- Boolean Algebra
- Logic gate symbols and equivalence relations
- Revision materials (Annex) number system conversions

Revision Materials

- **Not examinable and intended to aid understanding.**

Binary Numbers / Special Powers of 2

2^{10} (1024) is Kilo, denoted "K"

2^{20} (1,048,576) is Mega, denoted "M"

2^{30} (1,073, 741,824) is Giga, denoted "G"

2^{40} (1,099,511,627,776) is Tera, denoted “T"

BASE CONVERSION - Positive Powers of 2

- **Useful for Base Conversion (worth memorising):**

Exponent	Value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Exponent	Value
11	2,048
12	4,096
13	8,192
14	16,384
15	32,768
16	65,536
17	131,072
18	262,144
19	524,288
20	1,048,576
21	2,097,152

Converting Binary to Decimal

- To convert to decimal, use decimal arithmetic to form:
- Σ (digit × respective power of 2).
- Example:Convert 110101_2 to N_{10}:

Converting Decimal to Binary

- **Method 1**
  - Subtract the largest power of 2 (see slide 14) that gives a positive remainder and record the power.
  - Repeat, subtracting from the prior remainder and recording the power, until the remainder is zero.
  - Place 1's in the positions in the binary result corresponding to the powers recorded; in all other positions place 0's.
- **Example: Convert 625_{10} to N_2**

Commonly Occurring Bases

Name	Radix	Digits
Binary	2	0,1
Octal	8	0,1,2,3,4,5,6,7
Decimal	10	0,1,2,3,4,5,6,7,8,9
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Numbers in Different Bases

- **Good idea to memorize!**

Decimal (Base 10)	Binary (Base 2)	Octal (Base 8)	Hexa decimal (Base 16)
00	00000	00	00
01	00001	01	01
02	00010	02	02
03	00011	03	03
04	00100	04	04
05	00101	05	05
06	00110	06	06
07	00111	07	07
08	01000	10	08
09	01001	11	09
10	01010	12	0A
11	01011	13	0B
12	01100	14	0C
13	01101	15	0D
14	01110	16	0E
15	01111	17	0F
16	10000	20	10

Conversion Between Bases

Method 2

To convert from one base to another:

1) Convert the Integer Part

2) Convert the Fraction Part

3) Join the two results with a radix point

Conversion Details

- **To Convert the Integral Part:**

  Repeatedly divide the number by the new radix and save the remainders. The digits for the new radix are the remainders in *reverse order* of their computation. If the new radix is > 10, then convert all remainders > 10 to digits A, B, …

- **To Convert the Fractional Part:**

  Repeatedly multiply the fraction by the new radix and save the integer digits that result. The digits for the new radix are the integer digits in *order* of their computation. If the new radix is > 10, then convert all integers > 10 to digits A, B, …

Example 2: Boolean Algebraic Proofs

- $AB + \overline{A}C + BC = AB + \overline{A}C$ (Consensus Theorem)

Proof Steps

Justification (identity or theorem)

$AB + \overline{A}C + BC$

$= AB + \overline{A}C + 1 \cdot B\overline{C}$?

$= AB + AC + (A + A) \cdot BC$?

$=$

Proof of Simplification

$$x \cdot y + \bar{x} \cdot y = y \qquad (x + y)(\bar{x} + y) = y$$

Proof of DeMorgan's Laws

$$\overline{x + y} = \bar{x} \cdot \bar{y} \qquad \overline{x \cdot y} = \bar{x} + \bar{y}$$