SEC204

Computer Architecture and Low Level Programming

Dr. Vasilios Kelefouras

Email: v.kelefouras@plymouth.ac.uk

Website:

https://www.plymouth.ac.uk/staff/vasilios-kelefouras

Date

23/09/2019

School of Computing

(University of Plymouth)

First Things First…

- **Please ensure that has been scanned!**

Computer Architecture and Low Level Programming

Lectures

- Week 1. Introduction to [illegible]tronics , Computer arithmetic
- Week 2. Linux (Martin Read)
- Week 3. C Programming (Martin Read)
- Week 4. Safe S/W - Buffer Overflows (Martin Read)
- Week 5. Safe S/W - Format String attacks (Martin Read)
- Week 6. Computer Architecture
- Week 7. Computer Architecture
- Week 8: Memory hierarchy and memory systems
- Week 9. Different Computer architectures
- Week 10 Bomb Lab
- Week 11. Security (Kimberly Tam) + revision
- Week 12. No Lecture

Computer Architecture and Low Level Programming

Labs

- Week 1. Introduction to c[illegible]ronics , Computer arithmetic (paper based)
- Week 2. Linux (Martin Read)
- Week 3. Linux (Martin Read)
- Week 4. Safe S/W - Buffer Overflows (Martin Read)
- Week 5. Safe S/W - Format Strings (Martin Read)
- Week 6. Computer Architecture – basics (paper based)
- Week 7. Computer Architecture (Assembly)
- Week 8: Memory hierarchy and memory systems (Assembly)
- Week 9. Computer Systems
- Week 10 Bomb Lab (Assembly) (Martin Read)
- Week 11. Revision and coursework support

Learning Outcomes

1. Recognise the ope[illegible] microprocessor core components and machine level data re[illegible]on

2. Interpret and manipulate assembly code via hardware debugging techniques

3. Apply reverse engineering techniques to identify main software flaws

4. Identify relevant countermeasures for main software flaws

Assessment

- Coursework (50%)
- In-class Test (50%)

- **How to do well?**
  - Pay attention in the Lectures and Labs
  - Self-study and practice coding
  - Follow instructions in the assignments
  - Start early: as soon as the assessment brief is advertised
  - Submit your own work (i.e. do not plagiarise) and demonstrate your understanding of the concepts

About Myself

- Dr Vasilios Kelefouras
  - Email: v.kelefouras@ .ac.uk

- Portland Square B331
  - Office hours: Any time – but please email first.

- My Research Area:
  - ✓ Optimizing Software in terms of low execution time and low energy consumption
  - ✓ High Performance Computing
  - ✓ Optimizing Compilers
  - ✓ Task mapping on Heterogeneous hardware architectures

This week - Introduction

Outline of the first half

- How computers are made?
- Logic gates
- Boolean algebra basics
- Basic circuit diagrams

- What is computer architecture
- Why do we need different computer architectures
- How to compare them – different points of view
- Comparison by generation & date

How are computers made? (1)

- It all begins with commo[illegible]ch consists mostly of silicon dioxide (quartz)
- Using chemical methods, [illegible] converted to pure silicon
- Pure silicon shines like a metal, but is breakable like a ceramic
- Silicon is a semiconductor

  - It means that we can make it conduct electricity, or make it stop conducting

  - We can switch an electrical current in silicon on or off, at will, and very, very fast (nano seconds)

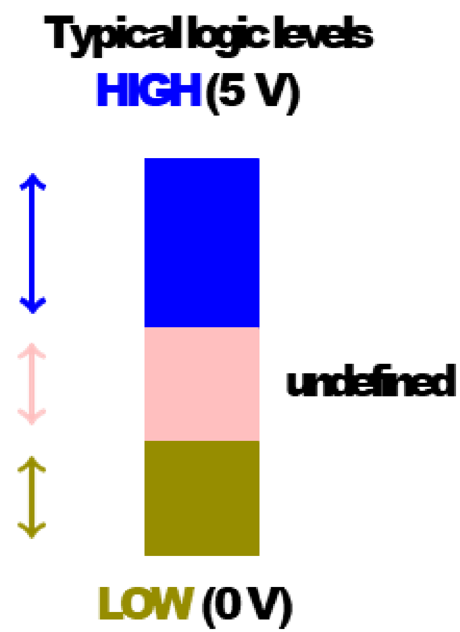
  - From silicon, we make fast switches!

  - A whole bunch of those switches together make a chip, which is put inside a plastic cover

- **The heart of anything electronic is those silicon switches**

How are computers made? (2)

- How do those silicon switc[illegible]y make all this happen?
  - This is called switch lo[illegible]ean logic, after George Boole (English mathematician, 1815-[illegible] was the first to think of it -- long before electronics existed!
  - A switch is either on or off – just two possible states
  - A digital signal has only two possible voltage values, usually known as logic 0 and logic 1
  - For CMOS logic gates, logic 1 is any voltage greater than 70% of the supply voltage, and logic 0 anything less than 30% of supply voltage. The in between values are not acceptable
  - Switches are called transistors



So you have switches. Now what?

- A single switch can only [illegible] "yes-no", "true-false", "1-0" (because that is the least writing...).
- But a bunch of switches [illegible]nt anything you want...

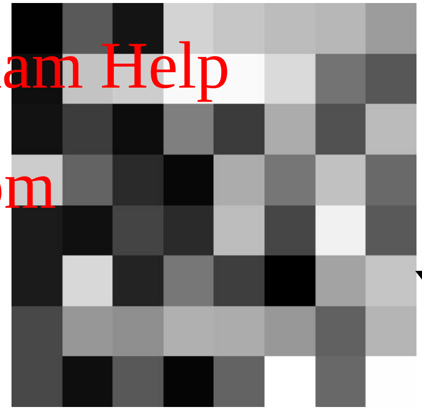
Numbers

Binary	Decimal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Text

Binary	Characters
0100 0000	@
0100 0001	A
0100 0010	B
0100 0011	C
0100 0100	D
0100 0101	E
0100 0110	F
0100 0111	G

Images



73	15	88	6	99	254	104	253
73	151	143	175	171	152	98	180
28	215	36	119	63	1	163	196
28	17	69	43	188	71	240	90
203	99	43	8	171	118	192	105
19	62	14	127	60	171	83	186
16	195	204	248	249	217	115	87
2	90	21	210	197	187	182	156

Digital computing

- The digital computers … logic: switches that can turn electricity through a se…tor ON or OFF (binary states)

- **These switches and the logics that they can adopt, are the building blocks of the computers that we use**

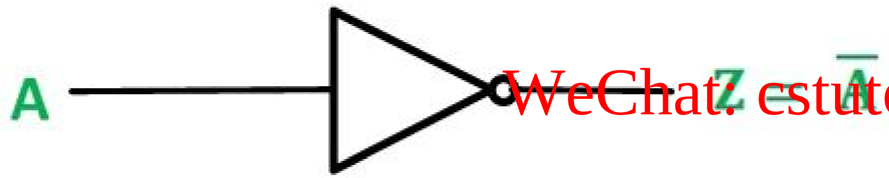
- An electronic component that can capture a particular logic is called a logic gate

- All logic gates are made from multiple transistors
  - It is easier to design hardware circuits in a gate level rather than transistor level..

- **The basic logic gates follow…**

Switch logic – NOT gate

- The NOT gate is a log ates are made from transistors) which implements logical neg



Truth table of NOT gate:

A	Z=A'
0	1
1	0

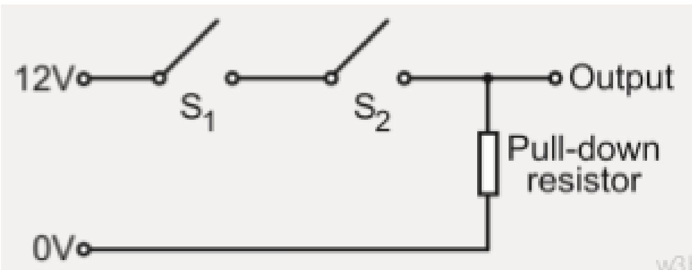
- Whatever logical state is applied to the input, the opposite state will appear at the output
- The NOT function is denoted by a horizontal bar over the value to be inverted, as shown in the figure above. In some cases a single quote mark (') may also be used for this purpose: $0' = 1$ and $1' = 0$

$A = 1 \quad A' = 0,$

$B = 0 \quad B' = 1$

Switch logic – AND gate

- The AND gate is a basic di gate that implements logical conjunct
- With the AND function, bot and B) must be 1 in order for the output (Z) to be1 – this is why it is called AND gate
- With either input at 0, the output will be held to 0



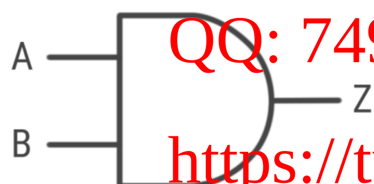
<u>AND function, «.»</u>

0 . 0 = 0

0 . 1 = 0

1 . 0 = 0

1 . 1 = 1

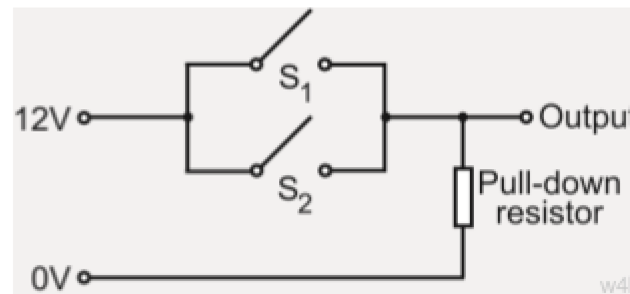


Truth table of AND gate:

A	B	Z=A . B
0	0	0
0	1	0
1	0	0
1	1	1

Switch logic – OR gate

- The OR gate is a basic dig ate that implements logical disjuncti
- The OR function allows the e true (logic 1) if any one or more of its inputs are true



<u>OR function, «+»</u>

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 1

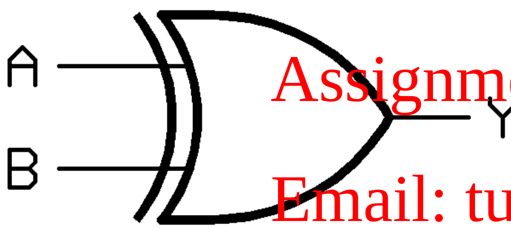
OR gate

Truth table of OR gate:

X	Y	Z=A+B
0	0	0
0	1	1
1	0	1
1	1	1

Switch logic – XOR gate

- The XOR (Exclusive-OR) digital logic gate that gives a true output only if its two inp ferent
- The XOR function is repre by $\oplus$



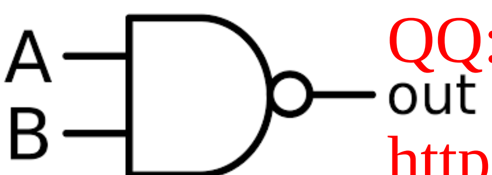
Truth table of XOR gate:

A	B	Y=A $\oplus$ B
0	0	0
0	1	1
1	0	1
1	1	0

Switch logic – NAND gate

- The NAND gate can be ge an AND gate followed by a NOT gate
- The logic symbol for the g below.

- The logic circuit of the NAND gate is shown below



Out= (A . B)’

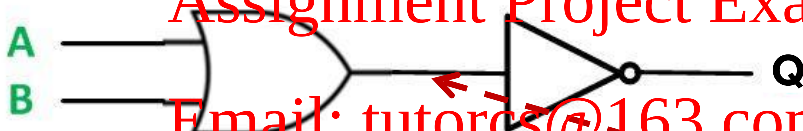
Truth table of NAND gate:

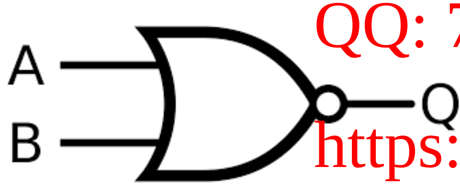
A	B	A . B	Out = (A.B)’
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Switch logic –NOR gate

- The NOR gate can be gen[illegible]n OR gate followed by a NOT gate
- The logic symbol for the g[illegible] below

- The logic circuit of the NOR gate is shown below





Out= (A + B)'

Truth table of NOR gate:

A	B	A+B	Q = (A+B)'
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

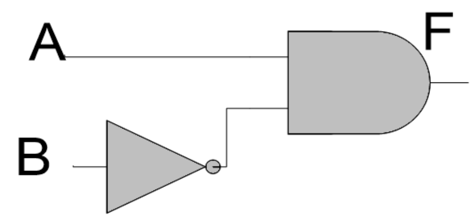
Summary of 2-input Logic Gates

- The following Truth Table compares the logical functions of the 2-input logic gates above

Inputs		Truth Table Outputs				
A	B	AND	NAND	OR	NOR	XOR
0	0	0	1	0	1	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	0	1	0	0

Exercise 1

- Write the Boolean expression of the following circuit diagram. Set up the truth table
- F = A . B’

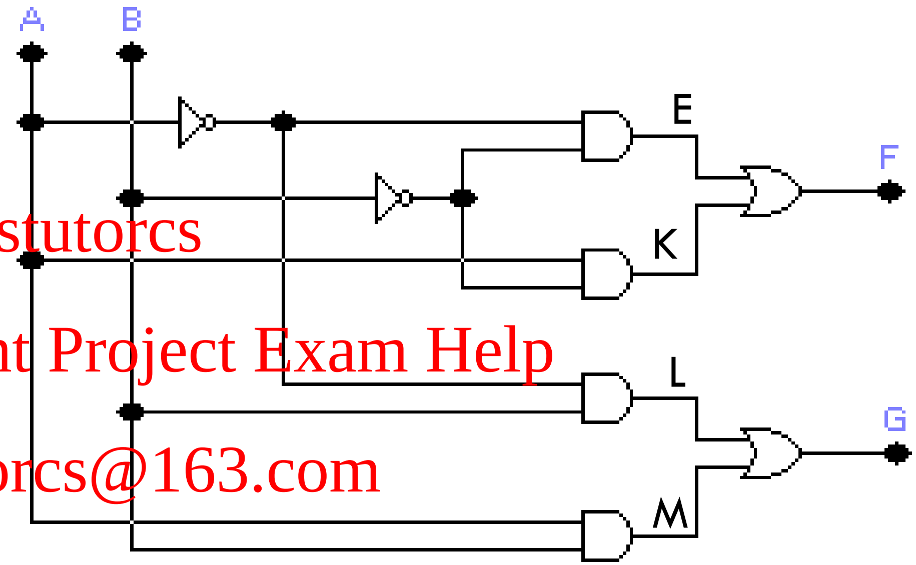


Truth table:

A	B	B’	F = A.B’
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

Exercise 2

- Write the Boolean expression for the following circuit diagram. Set up the truth table
- F = A’ . B’ + A . B’
- G = A’ . B + A . B



Truth table:

A	B	A’	B’	E=A’ . B’	K=A . B’	F=E+K	L=A’ . B	M=A . B	G=L+M
0	0	1	1	1	0	**1**	0	0	**0**
0	1	1	0	0	0	**0**	1	0	**1**
1	0	0	1	0	1	**1**	0	0	**0**
1	1	0	0	0	0	**0**	0	1	**1**

BOOLEAN AXIOMS AND THEOREMS

Identity Property

x + 0 = x

x . 1 = x

x + 1 = 1

x . 0 = 0

nt Property

= x

x . x = x

Complement Property

x + x' = 1

x . x' = 0

Involution *Property*

(x')' = x

Commutative Property

x + y = y + x

x . y = y . x

Associative Property

x + (y + z) = (x + y) + z

x . (y . z) = (x . y) . z

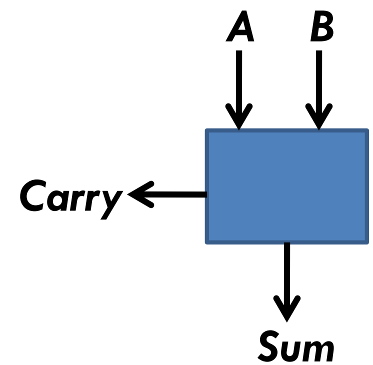
Simplification of Boolean Expressions

- Linear algebra: 2x + [illegible] = 5x – y
- Boolean algebra: (x+[illegible] y

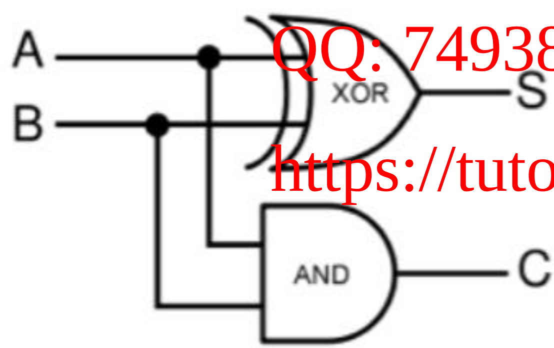
Proof	Identity Name
$(x+y)(\bar{x}+y) = x\bar{x}+xy+y\bar{x}+yy$	Distributive Law
$= 0+xy+y\bar{x}+yy$	Inverse Law
$= 0+xy+y\bar{x}+y$	Idempotent Law
$= xy+y\bar{x}+y$	Identity Law
$= y(x+\bar{x})+y$	Distributive Law (and Commutative Law)
$= y(1)+y$	Inverse Law
$= y+y$	Identity Law
$= y$	Idempotent Law

Half Adder (2 digit Adder)

- Consider the problem of a decimal digits
  - **We need two digits for**, one for the sum and one for the carry, e.g., if A=5, um=1 and Carry=1
  - The same holds when adding two binary digits too
- Consider the problem of adding two **binary** digits together
  - 0+0=0
  - 0+1=1
  - 1+0=1
  - 1+1=10



the decimal number 2 is represented by the "10" in binary. Thus two digits are needed, one for the sum and one for the carry



The Logic Diagram for a 2 digit adder (Half-Adder)

The Truth Table for a 2 digit adder (Half-Adder)

Inputs		Outputs	
A	B	S (Sum)	C (Carry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full Adder (three inputs) (1)

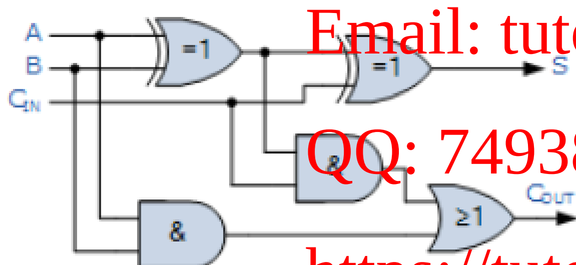
- The half-adder is a v[illegible] circuit and not really very useful because it [illegible] add two bits together
- **There is no provision [illegible] "carry-in" from the previous circuit when adding together multiple data bits**
- **We need a circuit that allows three inputs (x, y, and Carry In), and two outputs (Sum and Carry Out)**
- However, we can extend this adder to a circuit that allows the addition of larger binary numbers

235
+789
1024 Sum
11 Carry

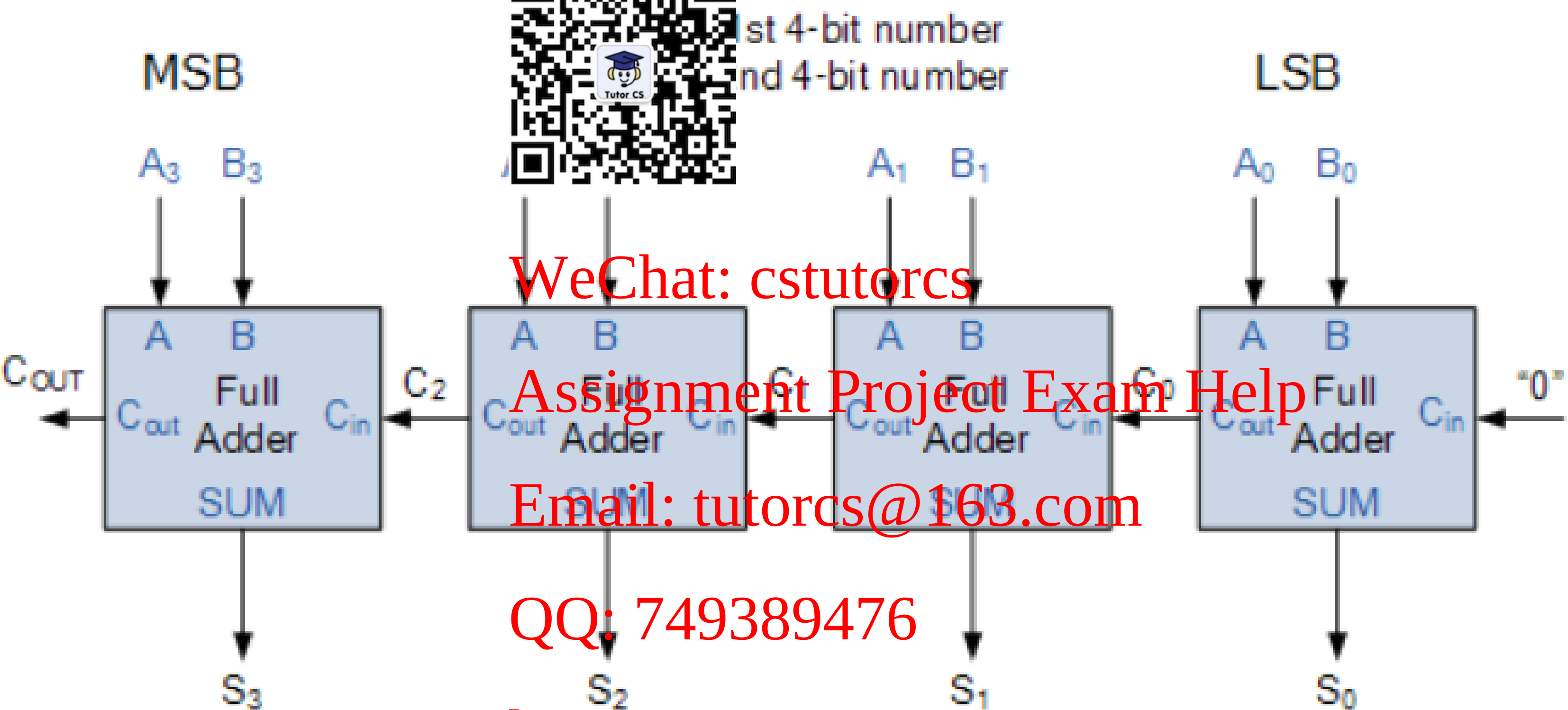
0 0 1 1 +
0 1 0 1
1 0 0 0 **Sum**
(0 1 1 1 **carry**)

Full Adder (three inputs) (2)

- In many ways, the full add[illegible]hought of **as two half adders connected together,** with the first half [illegible]ng its carry to the second half adder as shown

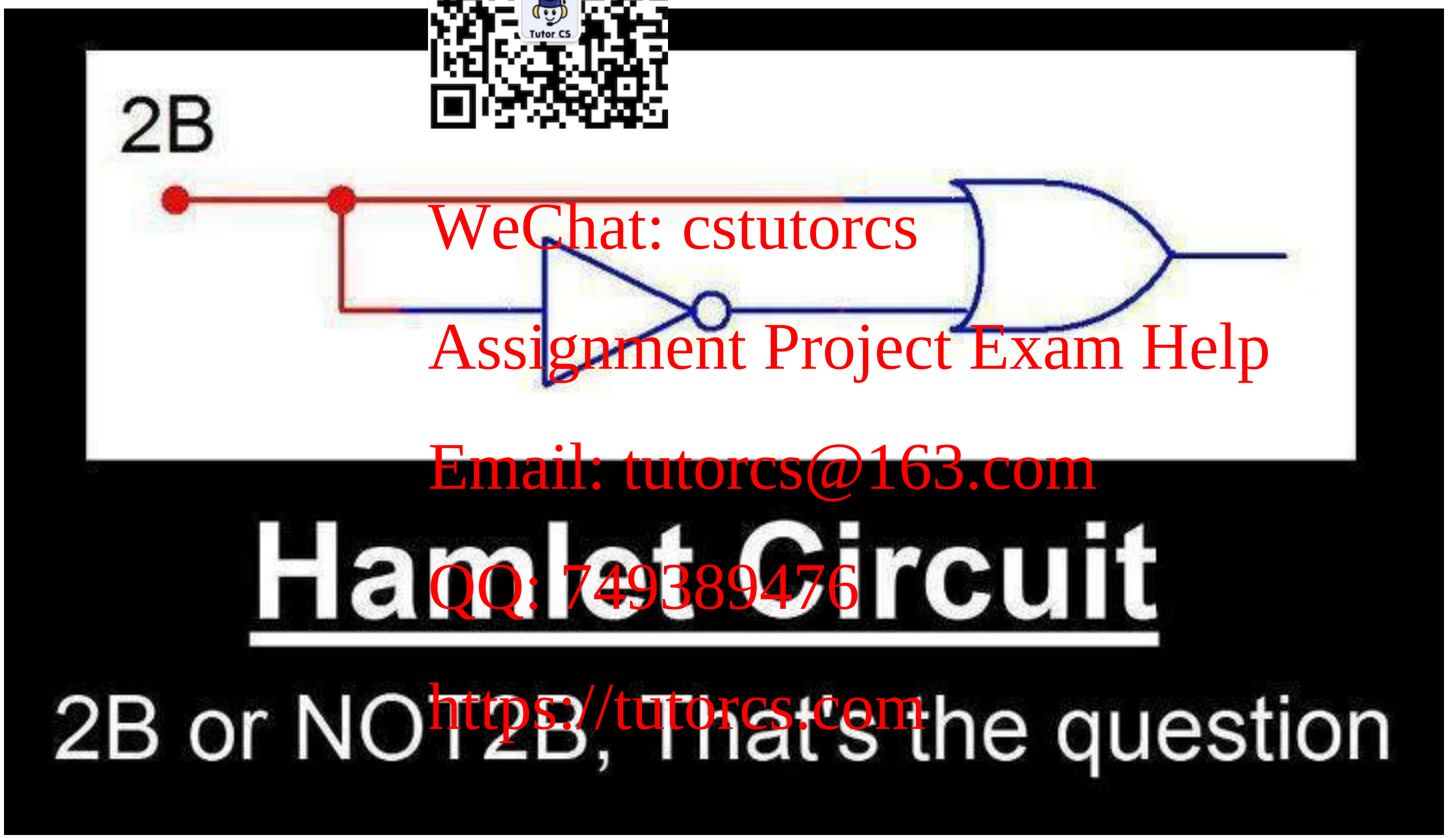
Sym[illegible]	Truth Table				
	C-in	B	A	Sum	C-out
	0	0	0	0	0
	0	0	1	1	0
	0	1	0	1	0
	0	1	1	0	1
	1	0	0	1	0
	1	0	1	0	1
	1	1	0	0	1
	1	1	1	1	1

A 4-bit Ripple Carry Adder



If A=0111 and B=1001 what would be the S and Cout?

For more information you can visit
https://www.electronics-tutorials.ws/combination/comb_7.html


2B
Hamlet Circuit
2B or NOT 2B, That's the question

Any questions?

What is computer architecture?

- The science and art of and interconnecting hardware components to create that meet functional, performance and cost goals
- A set of disciplines that describes a computer system by specifying its parts and their relations
- Simple words: **how parts are put together to achieve some overall goal**
  - Parts are transistors, logic gates, SRAM memory etc
  - A goal can be high performance, low cost, energy efficiency etc

Why do we need different computer architectures?

- To improve
  - ✓ Performance, e.g., Scientific applications, computer games
  - ✓ Power/energy consumption, battery life, e.g., Embedded Systems, Mobile Phones
  - ✓ Cost
  - ✓ Computer size and weight, e.g., tablet, laptop
  - ✓ Chip area, e.g., Brain implants
  - ✓ Abilities, e.g., Security, 3D graphics, Debugging Support

Hardware and Software

- Hardware refers to the physical elements that make up a computer or electronic system and everything else involved that is physically tangible.
  - This includes the monitor, hard drive, memory and the CPU.

- Software is a set of instructions or programs instructing a computer to do specific tasks

- **Any task done by software can also be done using hardware, and any operation performed directly by hardware can be done using software**

  - Hardware executes a function faster and by consuming less energy

Computer Architectures – need for classification

Too many puzzling words:

- x86, RISC, CISC, EPIC, VLIW, Harvard
- SIMD, SISD, MISD, MIMD
- Microcontrollers, ASIC, ASIP, FPGA, GPU, DSP
- Pipeline, vector processing, superscalar, hyper-threading, multi-threading
- UMA, NUMA, CUMA
- cluster, grid, cloud,

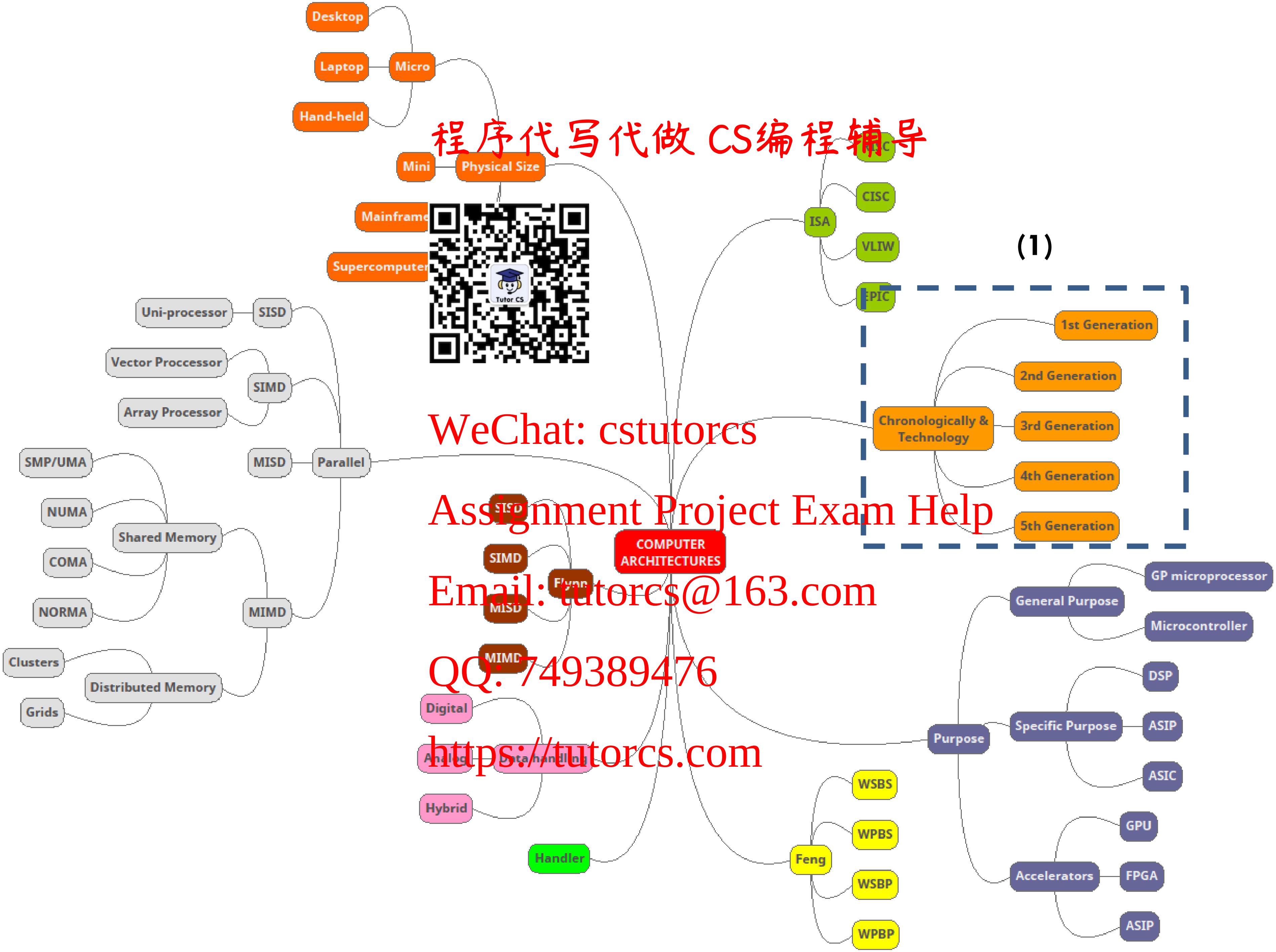
How to classify & compare all different computer architectures?

- Chronologically?
- ISA (Instruction Set Arc
- Purpose?
- Functionality?
- Performance?
- Power consumption?
- Cost?
- Flynn classification?
- Feng classification?
- Handler classification?

- Physical size?
- Parallelism?
- Architecture features?
- Memory access mode?
- Technology?
- Chip area?
- Flexibility?
- User Friendly?
- Data handling?

There is no single classification

We can make a classification for each bullet


COMPUTER ARCHITECTURES
Physical Size
Micro
Desktop
Laptop
Hand-held
Mini
Mainframe
Supercomputer
ISA
CISC
VLIW
EPIC
(1)
Chronologically & Technology
1st Generation
2nd Generation
3rd Generation
4th Generation
5th Generation
Parallel
SISD
Uni-processor
SIMD
Vector Proccessor
Array Processor
MISD
MIMD
Shared Memory
SMP/UMA
NUMA
COMA
NORMA
Distributed Memory
Clusters
Grids
Flynn
SISD
SIMD
MISD
MIMD
Data handling
Digital
Analog
Hybrid
Handler
Feng
WSBS
WPBS
WSBP
WPBP
Purpose
General Purpose
GP microprocessor
Microcontroller
Specific Purpose
DSP
ASIP
ASIC
Accelerators
GPU
FPGA
ASIP

Different computer architectures – classified chronologically & technologically

- 1st generation computers – Vacuum Tubes (1945-1955)
- 2nd generation computers – Transistors (1955-1965)
- 3rd generation computers – Integrated circuits (1965-1980)
- 4th generation computers – Very Large Scale Integration (VLSI) (1980-today)
- 5th generation computers – Low-power and invisible computers (present and beyond)

1 st generation computers – vacuum Tubes (1945-1955)

- Vacuum tubes for circuit
  magnetic drums for me
  little storage available)
- Programmed in machine language
- Often programmed by physical connection (hardwiring)
- **Big, Slow, Unreliable, Expensive**

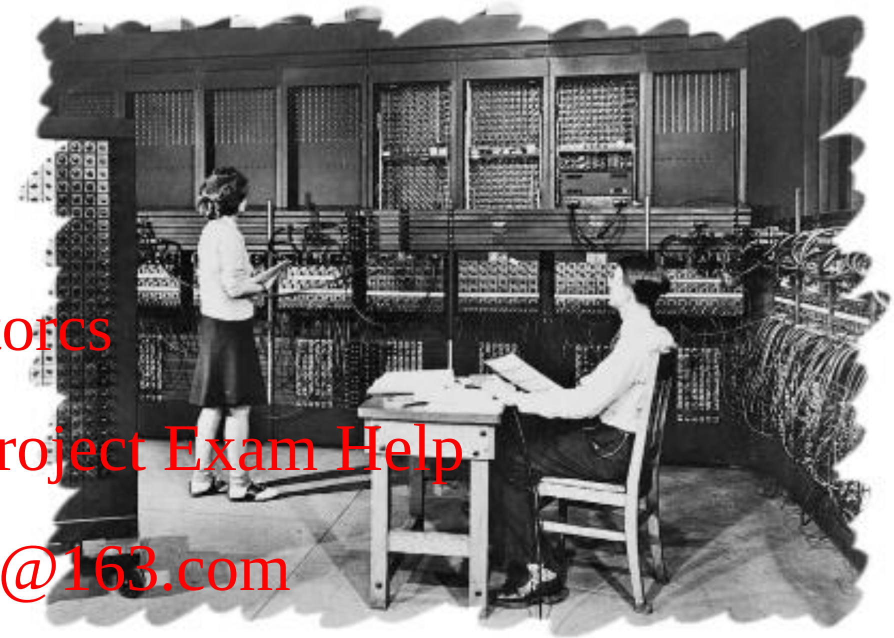


Fig.1. The ENIAC –the first programmable electronic computer – 1946.
17468 vacuum tubes,
1800 square feet, 30 tons

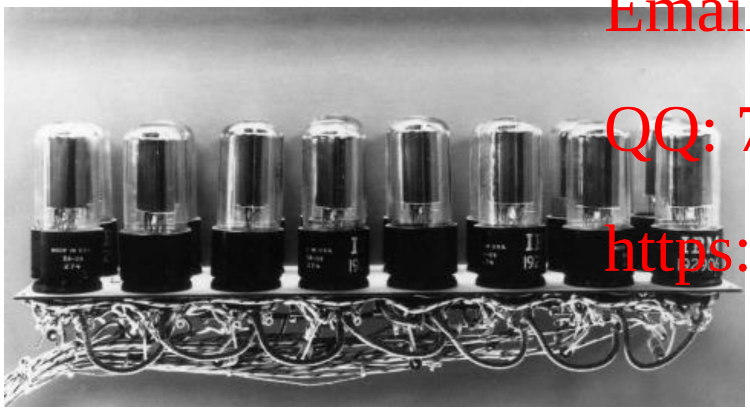


Fig.2.
A vacuum-tube circuit storing 1 byte

2^{nd} generation computers – Transistors (1955-1965)

- Transistors replaced vac
- Magnetic core memories introduced

- ✓ Smaller
- ✓ Faster
- ✓ Cheaper
- ✓ more energy-efficient
- ✓ more reliable
  - Various programming languages introduced (assembly, high-level)

Fig.3. The transistor

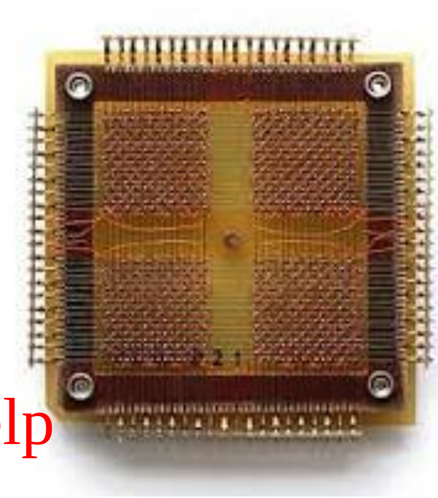


Fig.4. A 32x32 core memory plane storing 1024 bits of data.

3rd generation computers – Integrated circuits (1965-1980)

- Transistors were miniatu placed on silicon chips, called s tors
- ✓ Faster
- ✓ Increased memory capacity
- ✓ Lower cost – massive production

- Introduction of
  - Keyboards
  - Monitors
  - operating system

Fig.5. 3rd generation computer

4th generation computers – Very Large Scale Integration (VLSI) (1980-today)

- Thousands of integrate... ere built onto a single silico...
- What in the first gener... an entire room could now fit in the palm of the hand
- Development of the first microprocessor

- ✓ They are even smaller
- ✓ They are even faster

- Development of GUIs
- Introduction of Mouse pad

Fig.6. 4th generation computer

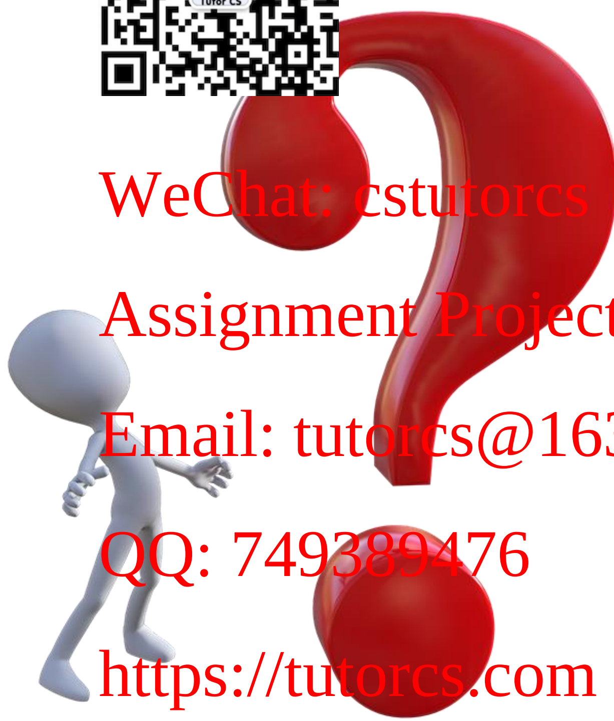
5^{th} generation computers – Low-power and invisible computers (present and beyond)

- Still in development
- Artificial intelligence
- Computers shrank
- Invisible computers are embedded into devices, e.g., watches
- Tablets, smart phones
- ULSI (Ultra Large Scale Integration) technology
- Microprocessor chips have ten million electronic components

✓ Smaller, faster, lower power consumption

Fig.7. 5^{th} generation computers - CPUs are embedded into devices

Any questions?



Reading List

Main Textbook

Linda Null, Julia Lobur. The E[illegible] Computer Organization and Architecture, 3rd Edition. Jo[illegible] Publishers, 2010

Computer Organization & Architecture: Designing for Performance. William Stallings, Seventh Edition, 2006

Structured Computer Organization. Sixth Edition, Andrew S. Tanenbaum, Todd Austin, PEARSON, 2012

Thank you