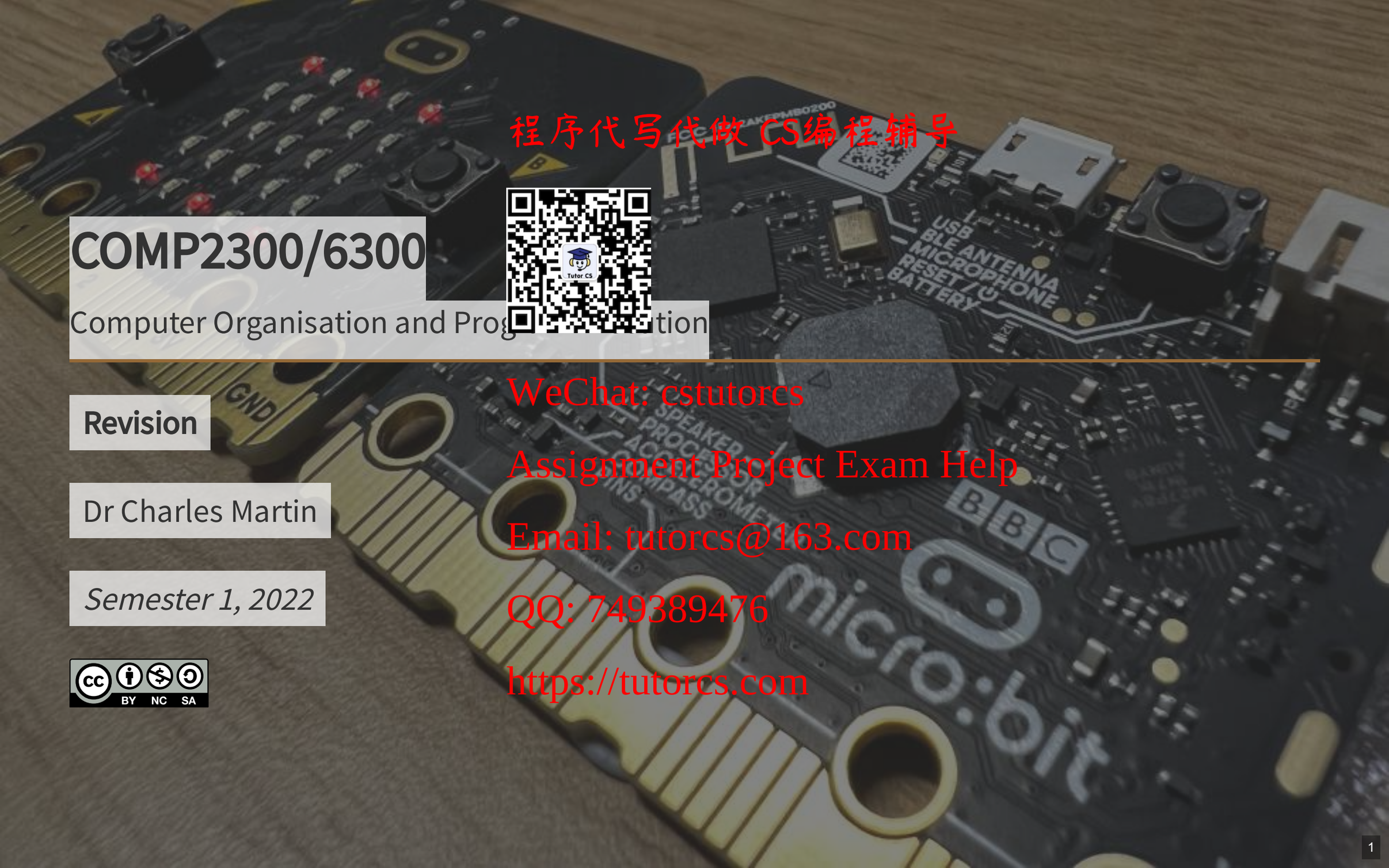


COMP2300/6300

Computer Organisation and Prog[illegible]tion

Revision

Dr Charles Martin

Semester 1, 2022

Admin Time

Assignment 2 done!

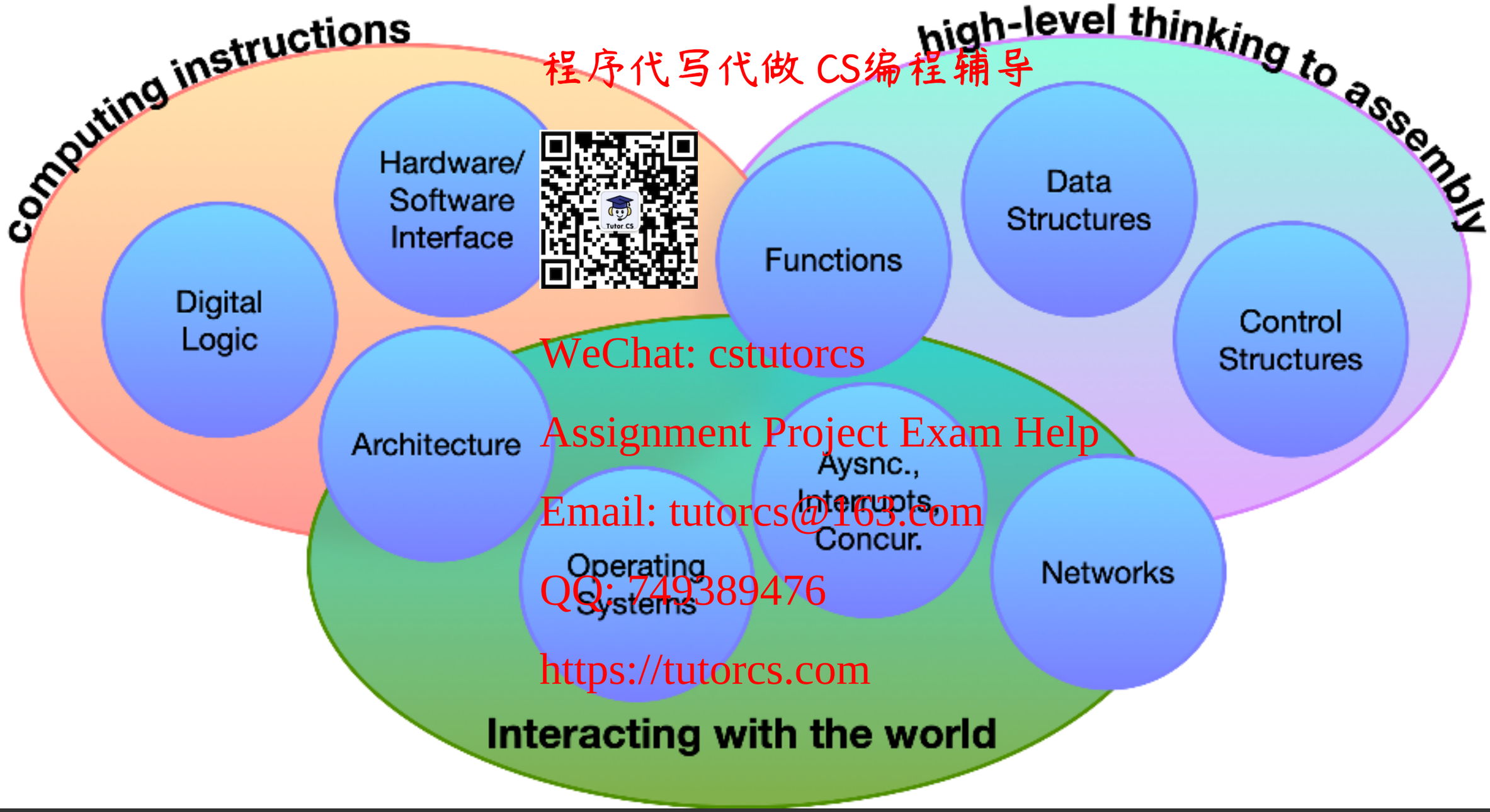
Lab 12 this week! Make your own OS!

Lab Pack 4 due: Wednesday 1/6/2022 23:59 AEST (day before exam period)

Quiz 2 still open!

Prepare for your final exam!

What was this course about again?



Digital Logic

Digital Logic Topics (in short):

- Boolean Algebra
- Combinatorial Logic Functions

- Digital Electronics - Logic Gates
- Binary Encoding, and 2’s-Complement
- Adders: Half, Full, Ripple Carry
- Arithmetic Logic Unit
- Simple CPU Architecture

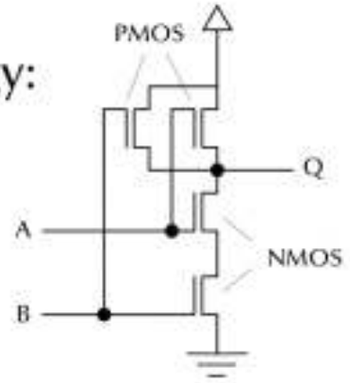
Logic gates

Symbolic: $Q = \overline{A \wedge B}$

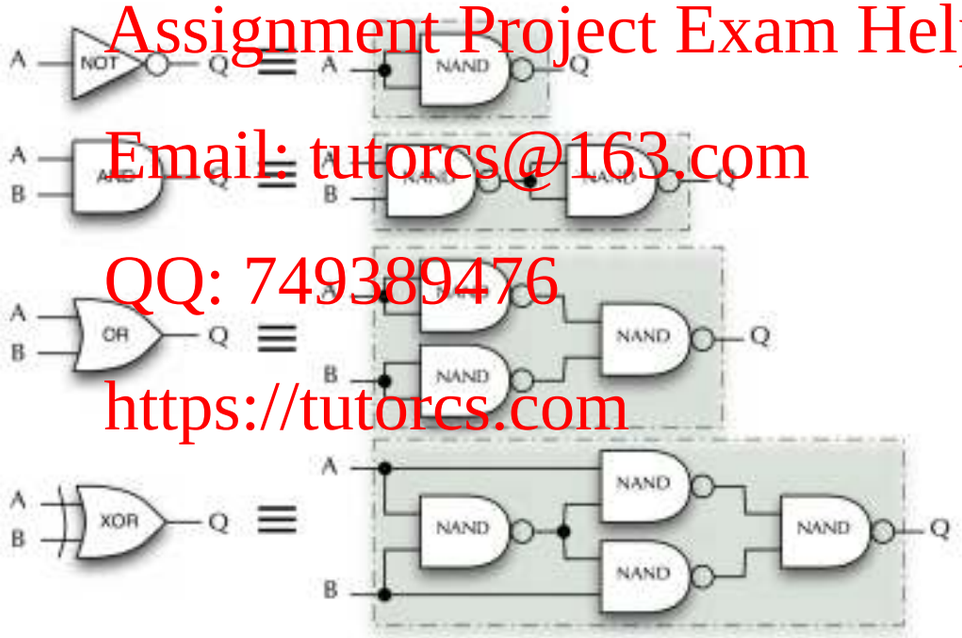
Diagram:

≡

Technology:



Elementary logic gate symbols:



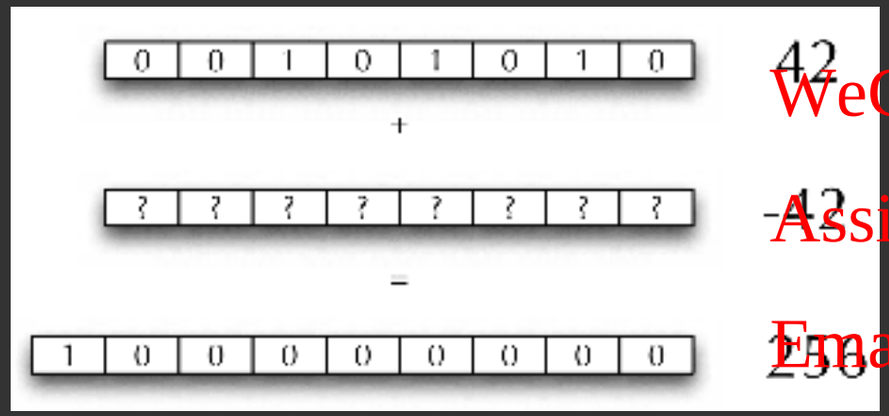
talk

1. how many bits can be added t
2. how long does it take?

3. where does the final carry bit go?

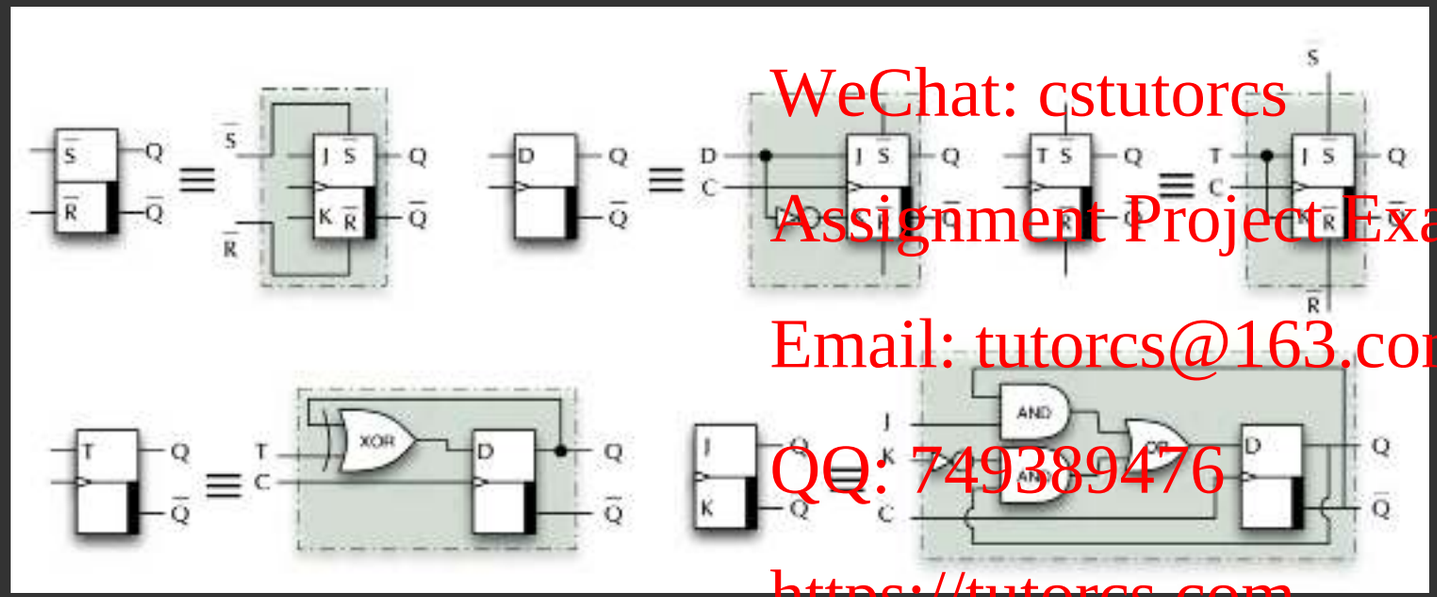
Twos complement representation

The basic idea: define (binary) ne[illegible]mbers so the adder works.



How do we make a number negative? Invert bits and add one! Why does that work?

Flip-flops



talk

What's the difference between B  ebra, combinational logic, and sequential logic?

Why do we need all three of thes describe computers?

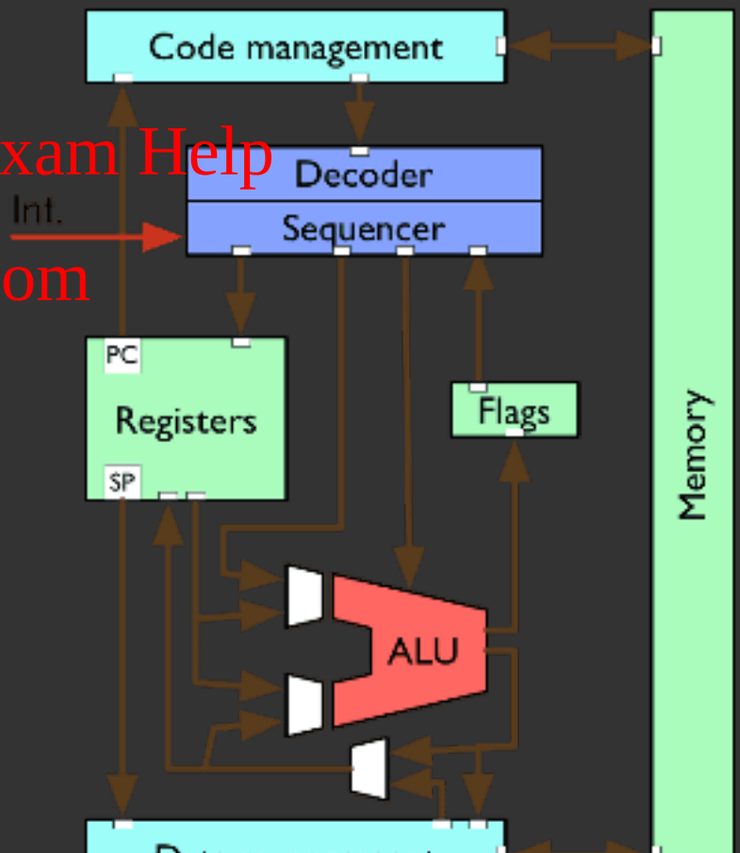
Are there parts of the story missing from COMP2300?

CPU Architecture

What are the **main components** o

Can you **explain** what each of the nents do?

We come back to this later…



Hardware/Software Interface

Hardware/Software Interface Topics (in short)

- Structure of an instruction
- Assembly to CPU instructions

- CPU Status Flags (NCZV)
- ARM v7 instructions (adding, subtracting, moving, rotate/shift, bit-wise ops)
- loading and storing from memory
- branch instructions
- Contents of Quick Ref. Card

NCZV Flags

- **N**egative
- **Z**ero

- **C**arry
- O**v**erflow

Negative

This status flag is set when the result of the ALU operation is negative *if interpreted as a twos complement signed integer*

```
movs r0, 5
movs r1, 6
subs r2, r0, r1
```

don't forget the `s` suffix

Zero

This status flag is set when the result of the ALU operation is zero

```
movs r5, 5
movs r6, -5
adds r4, r5, r6
```

Carry

This status flag is set when the re ALU operation requires a “carry out” *if interpreted as an unsigned 32-bi e.* it requires 33 or more bits to represent)

```
movs r2, 0xFF000000
movs r3, 0xFF000000
adds r5, r2, r3
```

Overflow

This status flag is set when the re ALU operation would overflow the min/max value *if interpreted as a twos complem d integer*

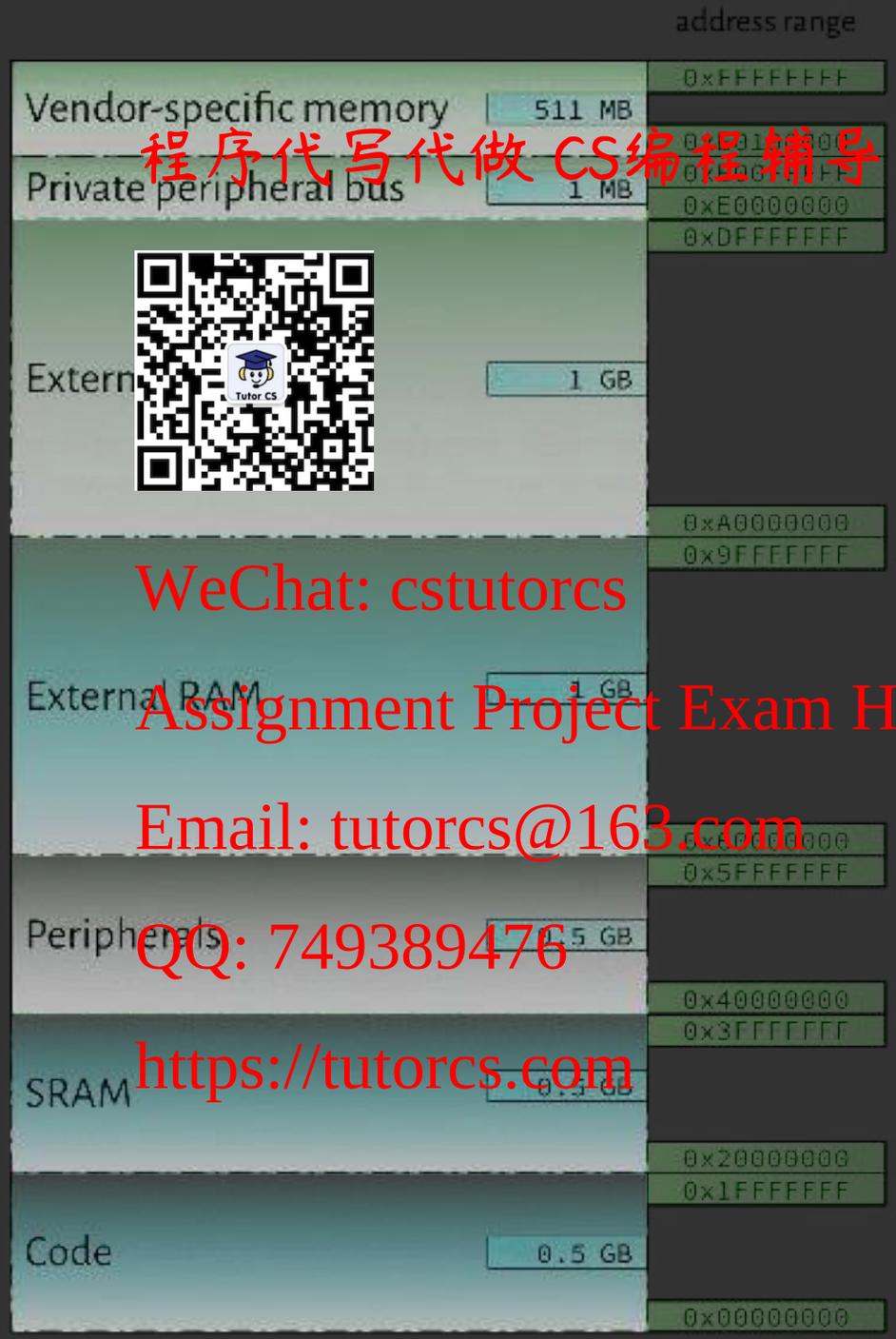
```
movs r0, 0x7FFFFFFF @ largest signed integer
adds r0, 1
```

```
movs r0, 0x80000000 @ smallest signed integer
subs r0, 1
```

talk


```
movs r0, 5
movs r1, 6
subs r2, r0, r1
```

What flags will be set after the `subs` instruction is executed?



ARM is a load-store architecture

Instructions are *either*:

- **ALU operations** which take inp[illegible]ve results to registers, or,
- **memory access** operations which *just* save and load from memory

What does this mean for the programmer?

Loading and Storing

Load:

```
mov r1, 0x20000000 @ put the address in r1
ldr r0, [r1]       @ load the data into r0
```


Store:

```
mov r0, 42
mov r1, 0x20000000
str r0, [r1]
```

Extra Operations

Load less than 32 bits

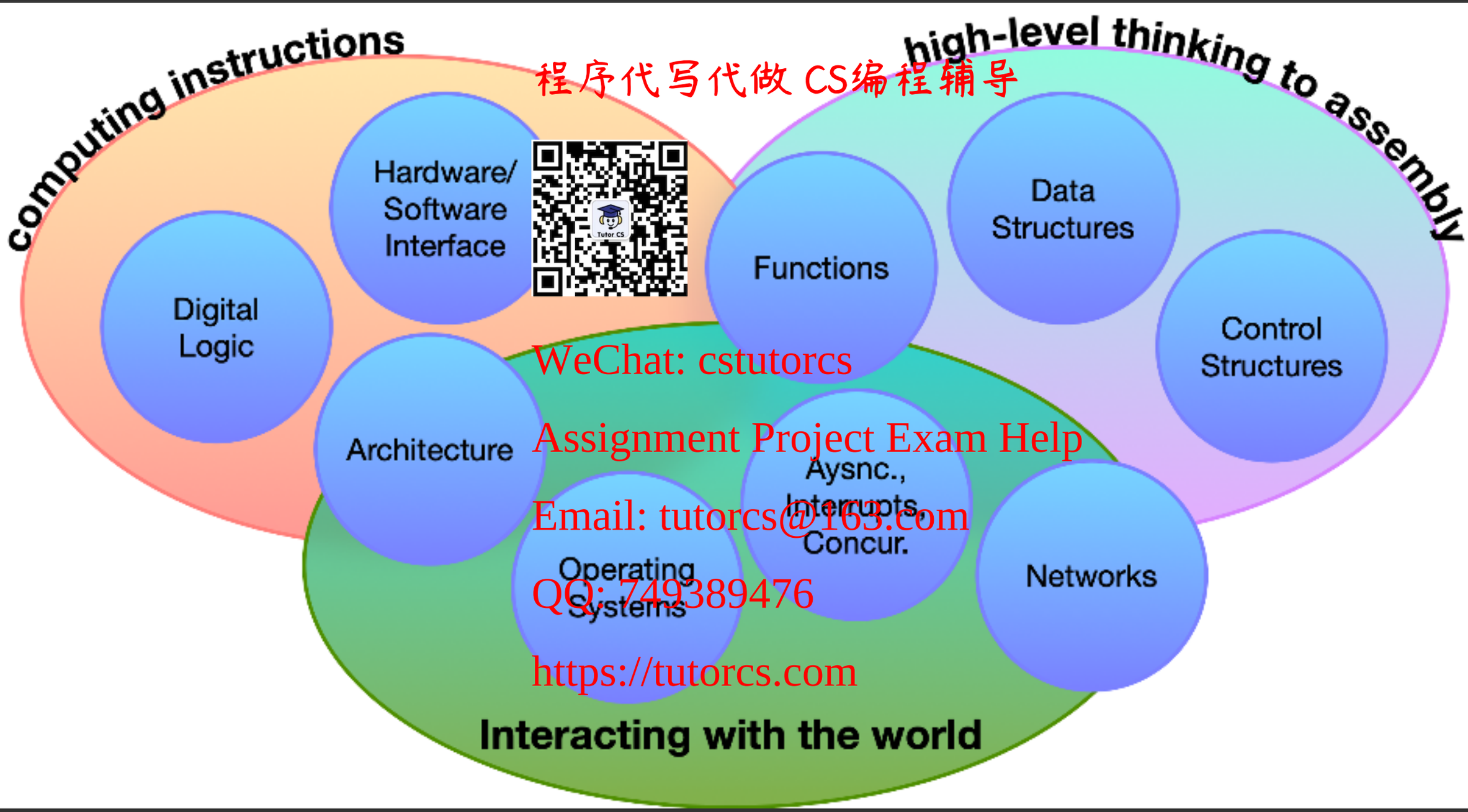
```
ldrb @ load byte from register
ldrh @ load halfword from register
strb @ store byte to register
strh @ store halfword to register
```


Negative Stack

```
stmdb <Rs>!, {Rgstrs} @ store multiple decrement before
ldmia <Rs>!, {Rgstrs} @ load multiple increment after
push {Rgstrs}
pop {Rgstrs}
```

Conditional branch examples

```
beq <label> @ branch i
bne <label> @ branch i
bcs <label> @ branch i

bcc <label> @ branch if C = 1
bmi <label> @ branch if N = 1
bpl <label> @ branch if N = 0
bvs <label> @ branch if V = 1
bvc <label> @ branch if V = 0
```



Functions

Functions Main Topics (in short)

- There and back again, `bl`, `bx`
- The stack

- Calling conventions
- Functions calling functions
- Functions calling themselves! (a.k.a. recursive functions)
- Local variables, and the stack frame (incl. `sp` and `fp`)
- Relative addressing
- Passing values: by copy and by reference

there, and back again

AAPCS

The ARMv7 Architecture Procedu[illegible]ndard is the convention we'll (try to) adhere to in programming our microbits

The full standard is quite detailed, but the general summary is:

- `r0`-`r3` are the parameter and scratch registers
- `r0`-`r1` are also the result registers
- `r4`-`r11` are callee-save registers
- `r12`-`r15` are special registers (`ip`, `sp`, `lr`, `pc`)

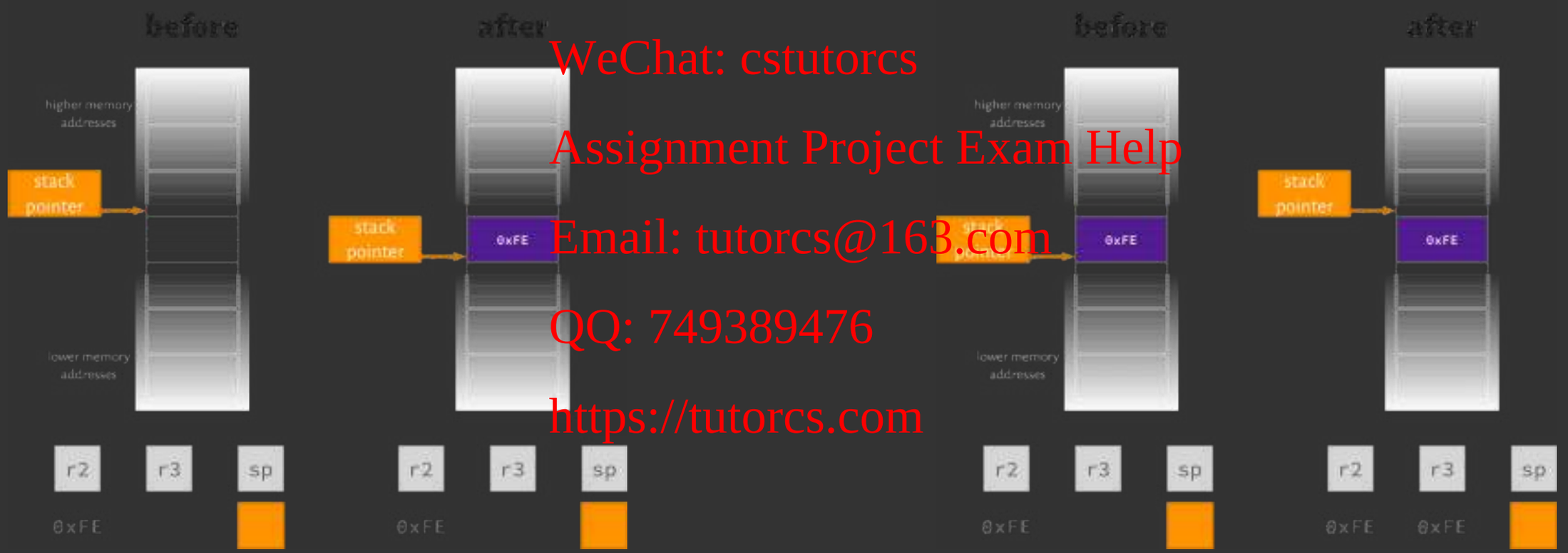
Store and Load to the stack


```
@ Push the value in r2 onto
str r2, [sp, -4]
sub sp, sp, 4

@ Different one-liners for Push
str r2, [sp, -4]!
push {r2}
stmdb sp!, {r2}

@ Pop the value from the "top" of the stack into r3
ldr r3, [sp]
add sp, sp, 4

@ One-liners for Pop
ldr r3, [sp], 4
pop {r3}
ldmia sp!, {r2}
```

Push and Pop; illustrated

<– Push

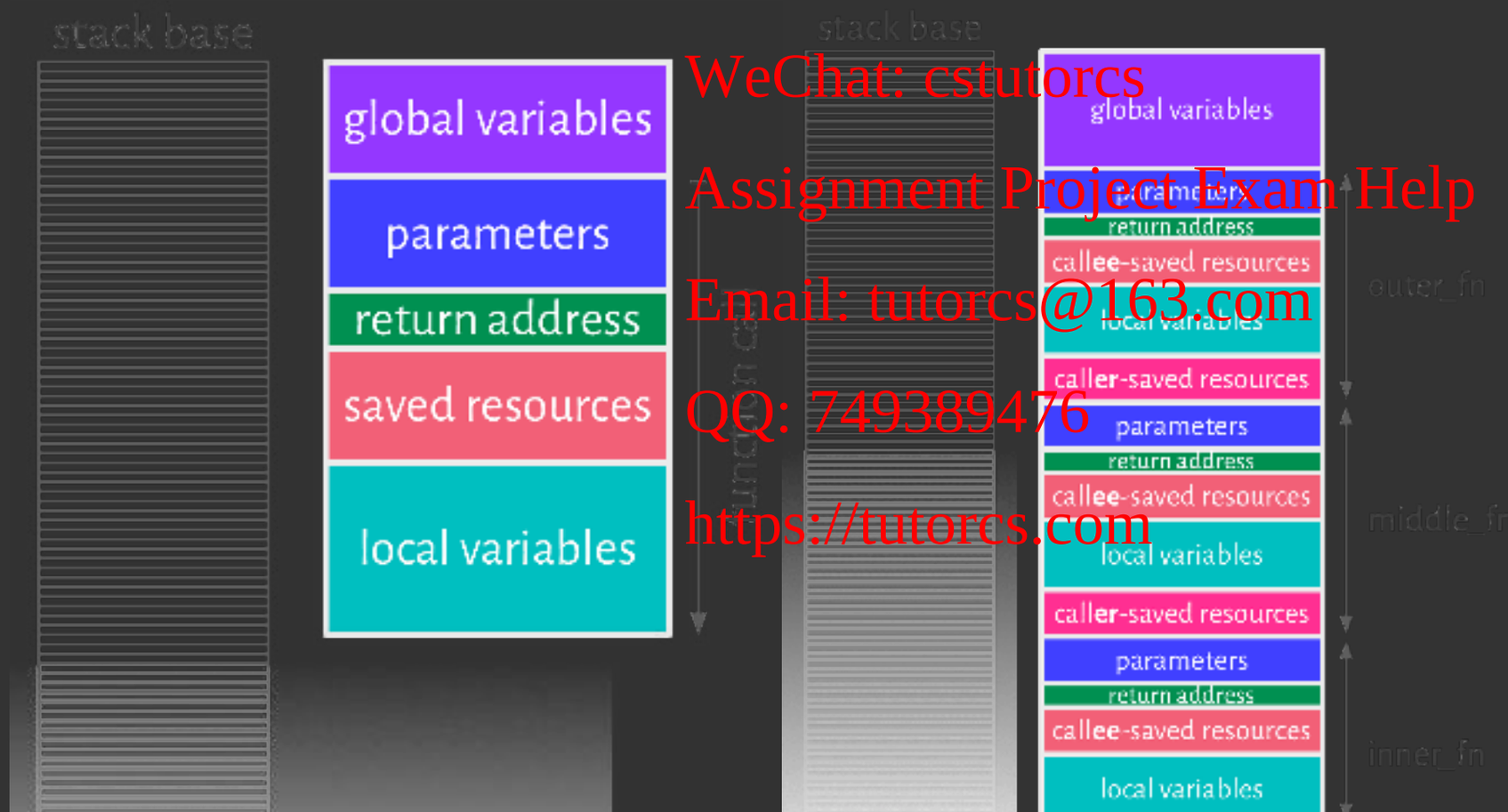
Pop –>



Passing by Copy or Reference

		Access	
		by copy	by reference
Information flow	in	**by value** Parameter becomes a constant inside the function or is copied into a local variable.	**by reference (immutable)** No write access is allowed while the function runs (also from outside the function).
	out	**by result** Calling function expects the parameter value to appear in a specific space at return.	**by reference (mutable, no read)** No read access from inside the function, write access on return.
	in & out	**by value result** Parameter is copied to a local variable and copied back at return.	**by reference (mutable)** Function can read and write at any time. Outside code shall not write.

Function stack frame



Recursive Functions: Factorial


```
fact: @ assume input is in r
  push {lr}
  cmp r1, #1
  beq base_case

  @ recursive case
  push {r1}
  sub r1, #1
  bl fact @ get fact(n-1)
  pop {r1}
  mul r0, r0, r1 @ calc fact(n-1) * n
  b continue_code

  base_case:
  mov r0, #1
  continue_code:
  pop {lr}
  bx lr
```

Control Structures

Control Structures Main Topics (in short)

- Conditional branching
- Control Structures in Machine

- if
- while
- for

CPSR table

<c>	meaning	flags
eq	equal	Z=1
ne	not equal	Z=0
cs	carry set	C=1
cc	carry clear	C=0
mi	minus/negative	N=1
pl	plus/positive	N=0
vs	overflow set	V=1
vc	overflow clear	V=0
hi	unsigned higher	C=1 ∧ Z=0
ls	unsigned lower or same	C=0 ∨ Z=1
ge	signed greater or equal	N=V
lt	signed less	N≠V
gt	signed greater	Z=0 ∧ N=V
le	signed less or equal	Z=1 ∨ N≠V

while loop components


```
while Register_1 < 100 loop
   Register_1 := Register_1 ** 2;
end loop;
```

```
while (register1 < 100) {
   register1 = register1 * register1;
}
```

```
while register1 < 100:
   register1 = register1 ** 2
```

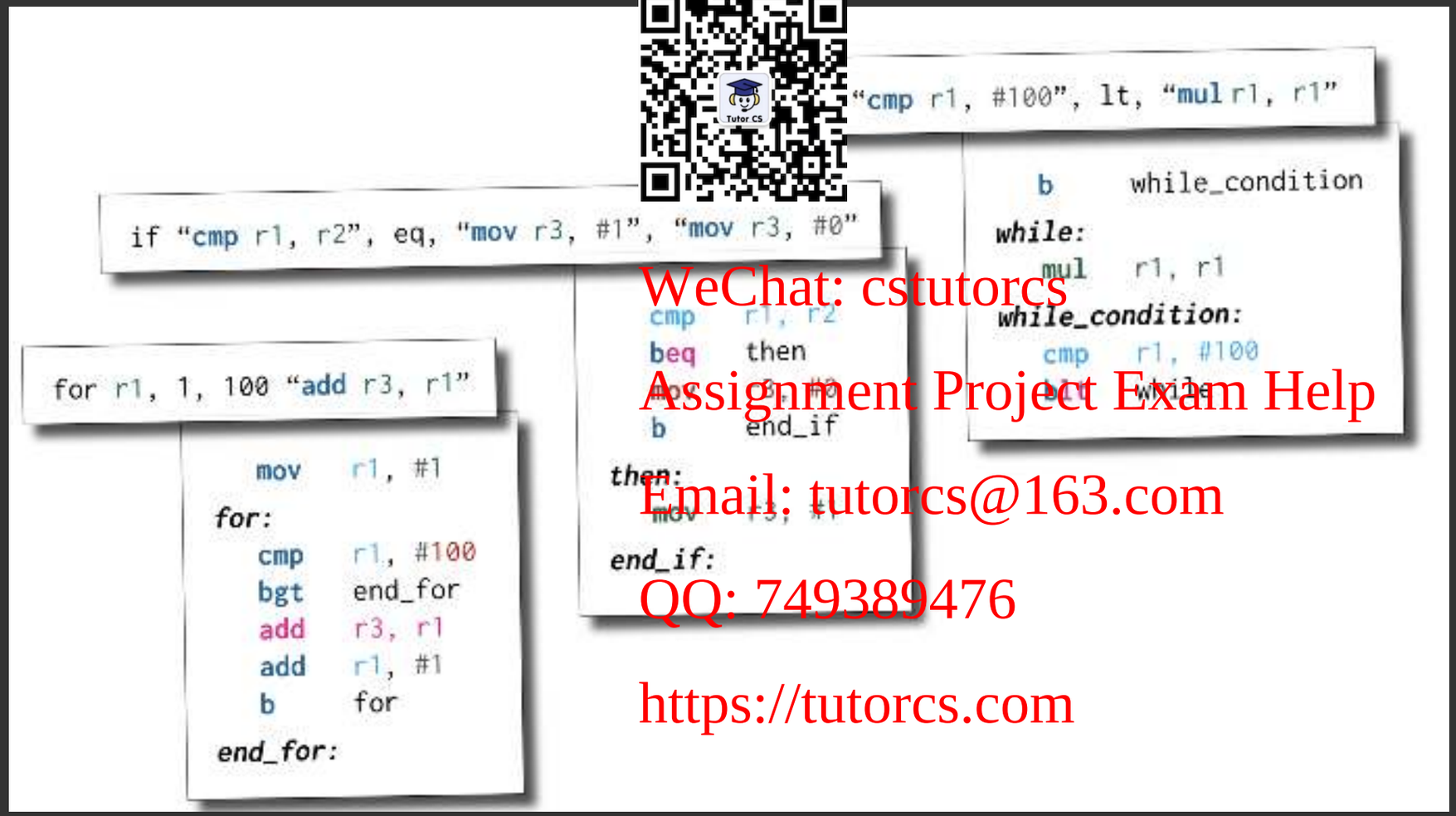
```
while Register_1 < 100 do
   Register_1 := Register_1 ** 2;
```

```
while Register_1 < 100 do
   Register_1 = Register_1 ** 2
enddo
```

```
while (Register_1 < 100) {
   Register_1 = Register_1 ** 2;
}
```

1. an expression (if)
2. a Boolean condition (if)
3. code inside the loop

control structures gallery - practice these!



talk

Which control structures were us  ur assignments?

Is there anything you can do in a at goes beyond “typical control structures”?

Data Structures

Data Structures Main Topics (in short)

- **Arrays**
- Structure

- Alignment
- Addressing
- Iterators
- Copying

talk

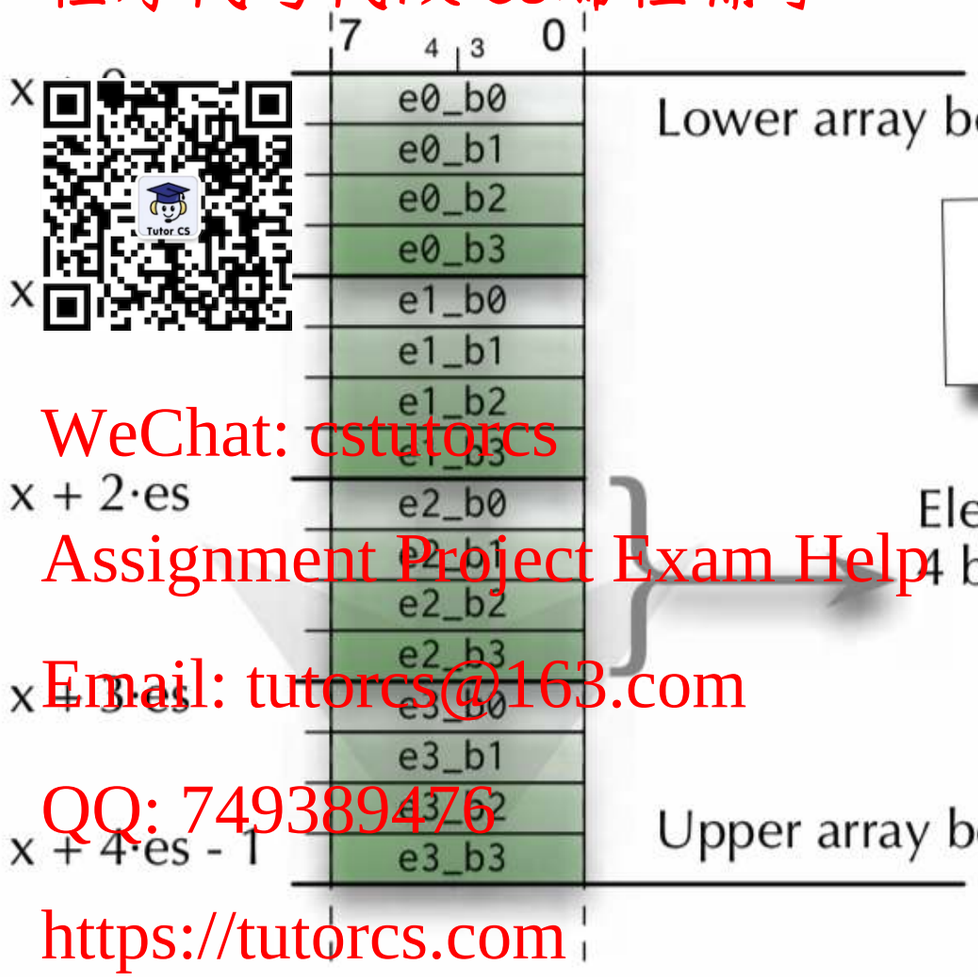
How do we know how big an arra mory?

Is it possible to write outside the the array?

Can you make an array where the size can be changed? (mutable array?)

How do we address a particular element in an array?

Array addressing



$x + i \cdot es$

Can arrays always be stored "packed"?

Maybe be good to have $es = 2^n$?

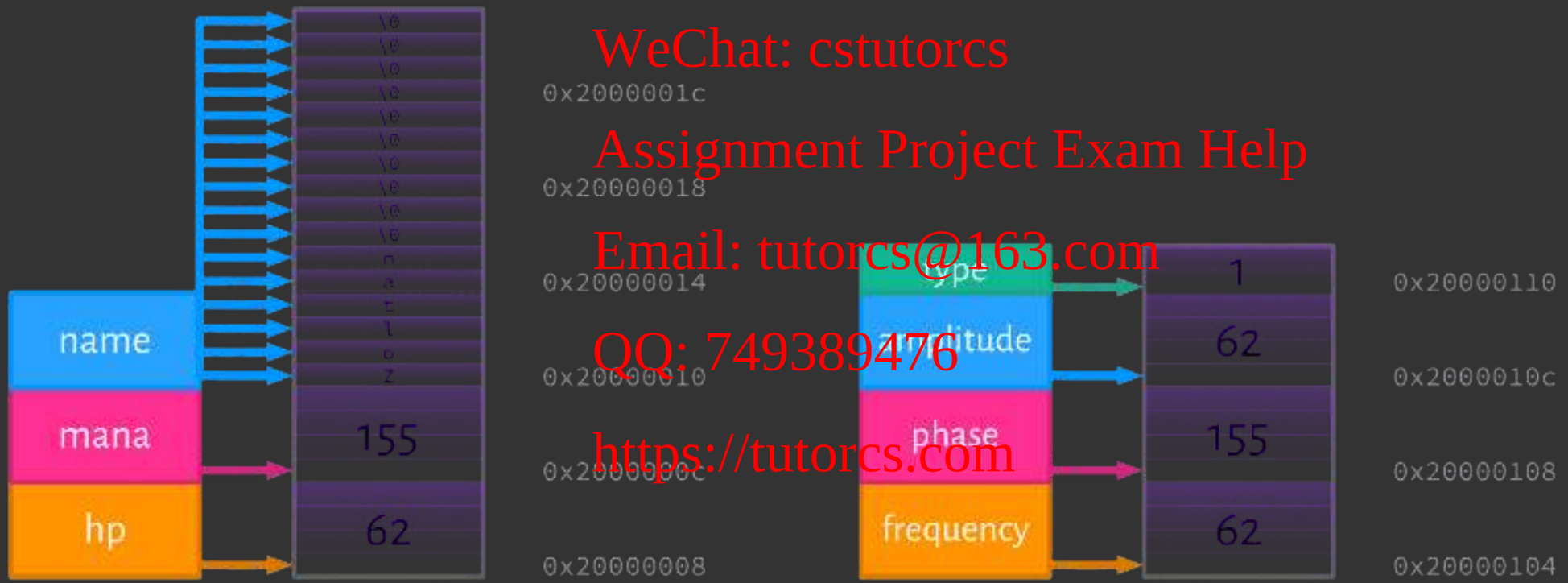
What happens if array bounds are violated?

☞ And who is checking that?

Add up the numbers in an array

```
  ldr r0, =array @ base addr
  mov r1, 0      @ from_inde
  mov r2, 7      @ to_index
@ setup

  mov r3, 0 @ "accumulator" register
  mov r4, 4 @ element size

array_sum:
  mul r5, r1, r4    @ calculate offset
  ldr r6, [r0, r5] @ load from offset
  add r3, r6        @ update accumulator
  add r1, 1         @ increment index
  cmp r1, r2        @ keep looping?
  ble array_sum
```

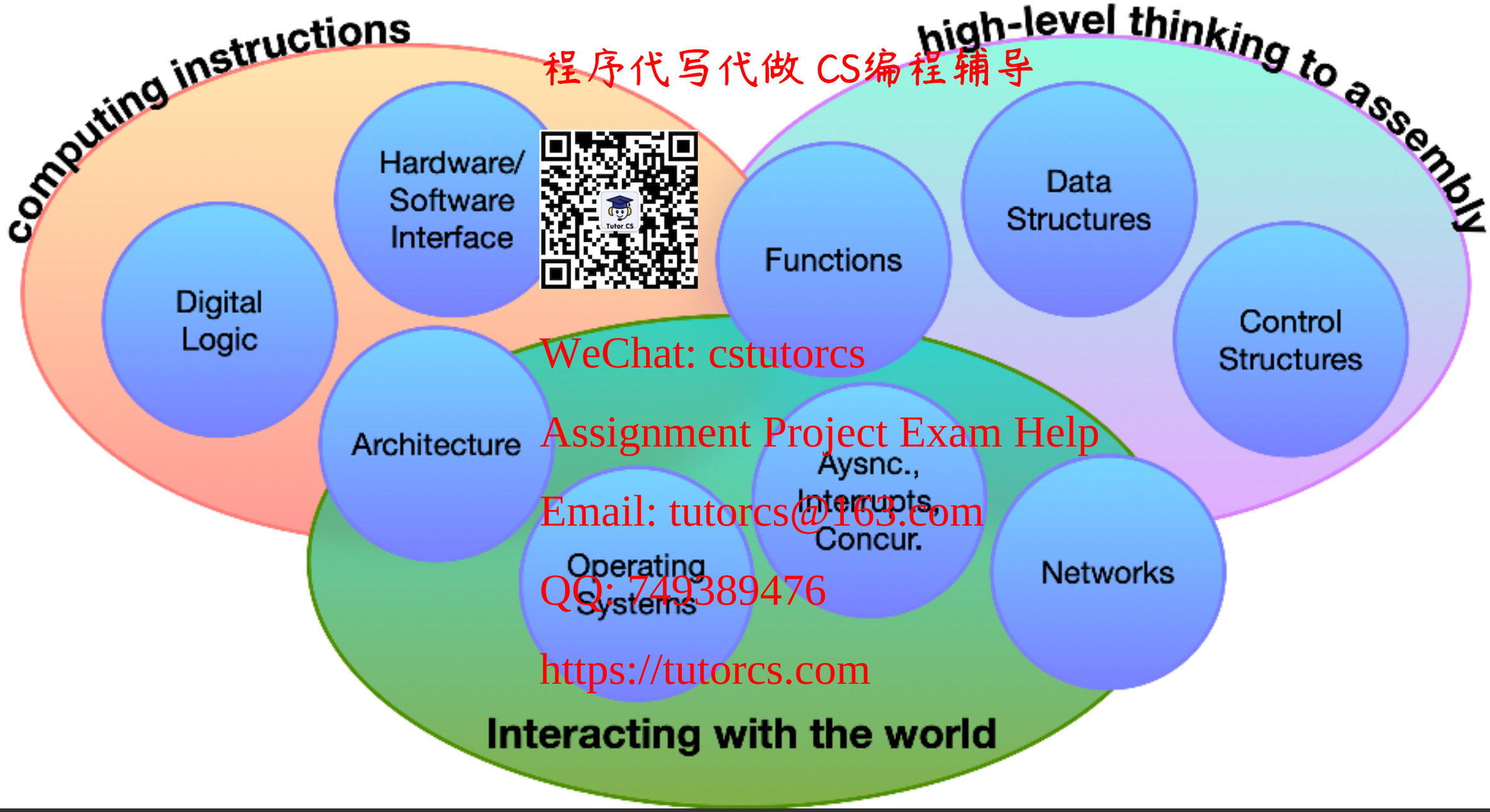
Records



talk

What's the difference between a  a **record**?

Imagine you are creating a Point-of-Sale system (cash register) using a microbit. What data structures might be required?



Asynchronism, Interrupts, and Concurrency

Async Main Topics (in short)

- Interrupts & Exceptions: When
- What happens during an inter

- How is this related to **parallel computing**?
- Concurrency and Synchronisation
- Race Conditions
- Mutual Exclusion
- Synchronisation (Locks and Semaphores)

talk

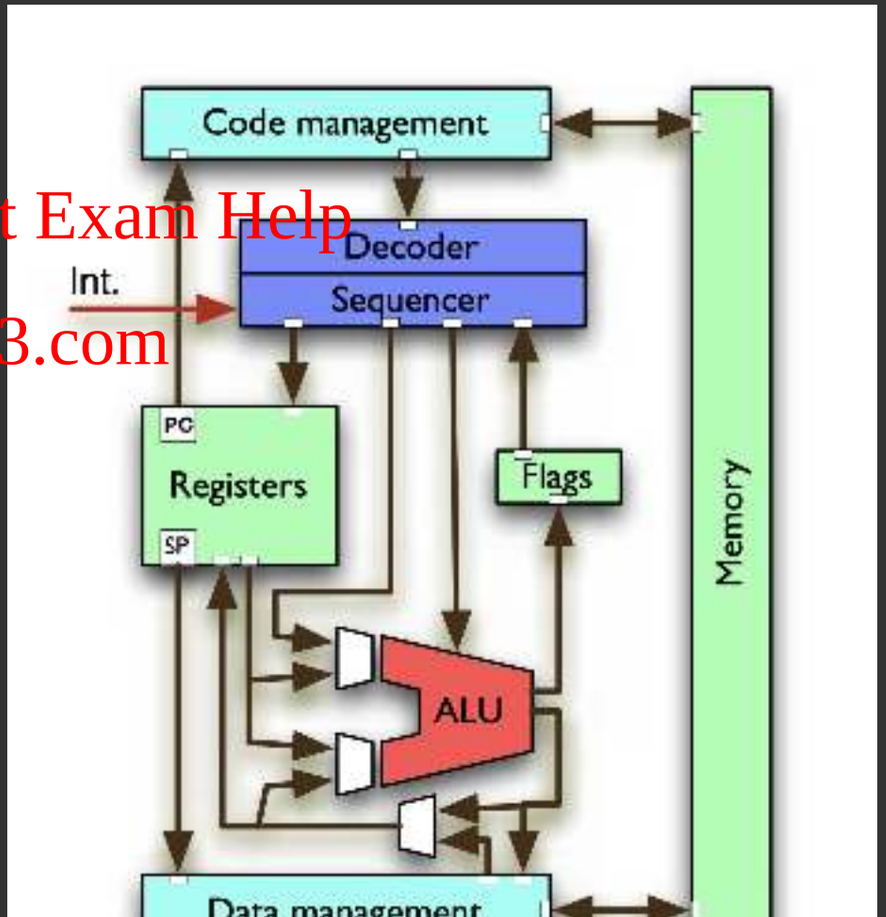
What's an interrupt? Why are they?

Interrupts

One or multiple lines **wired direc**[QR]**equencer**.

Interrupts: Enables pre-emptive scheduling, timer driven actions, transient hardware interactions, etc…

BUT: A little bit more work to set up, requires external hardware ("interrupt controller") to encode external requests.

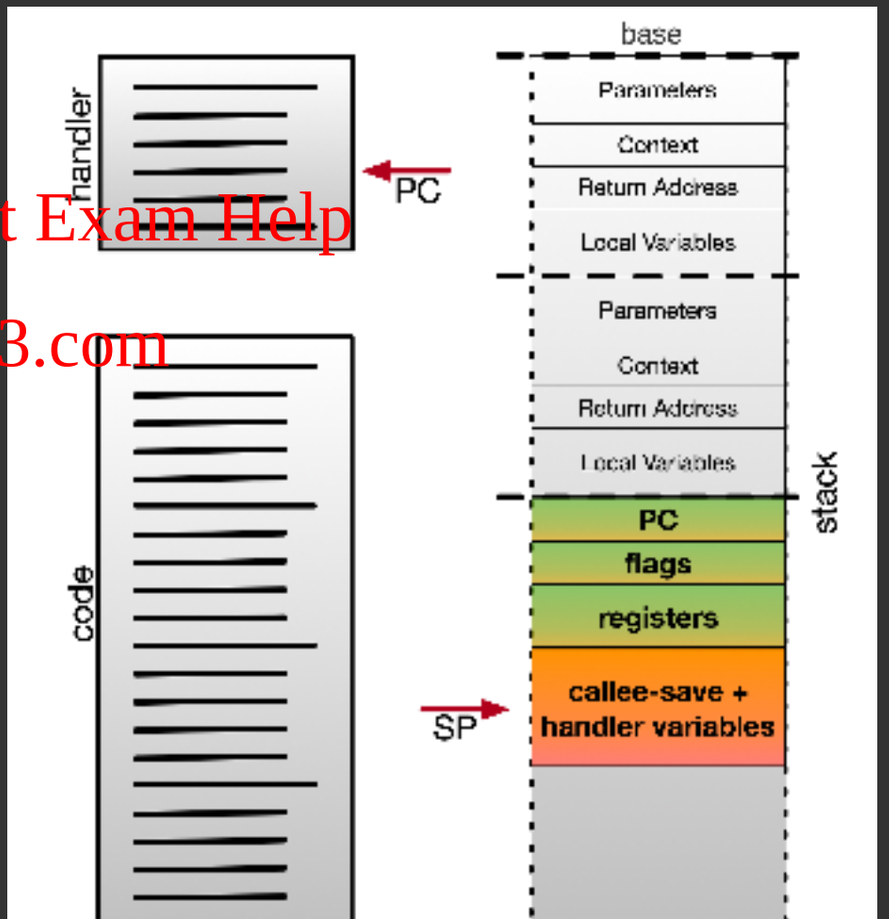


Interrupt Handler

A “normal function”, called when interrupt is triggered.

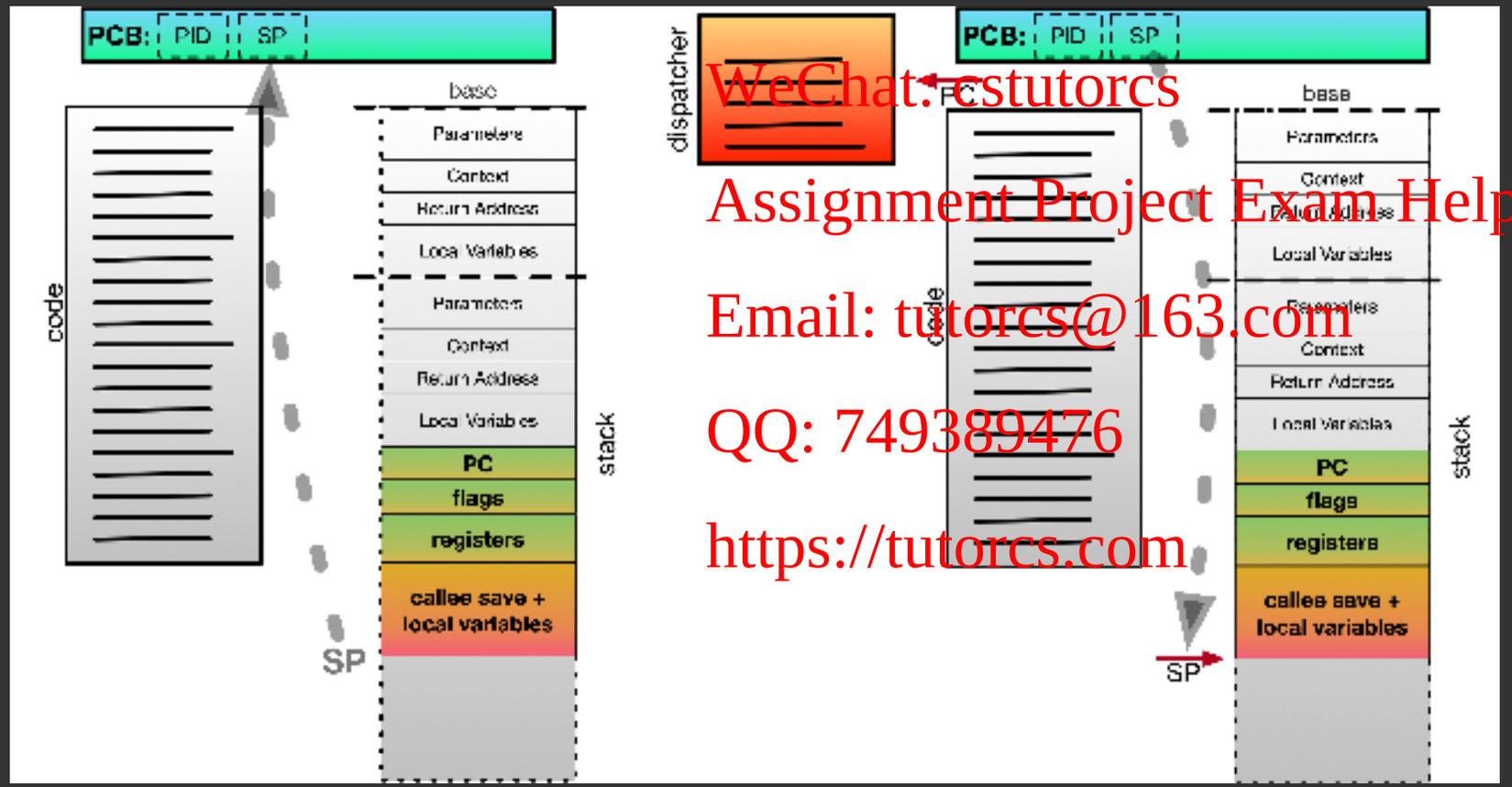
The CPU saves the “caller-save” context to the stack, loads `lr` with special value.

Handler saves “callee-save” context, then runs I/O or time-critical code.



Context Switch

Switch out a program (during an  without it even noticing!



talk

What's problems can occur when concurrent programs? What are the possible solutions?

How does the "too much milk" problem help us understand this issue?

Race Conditions and Mutual Exclusion

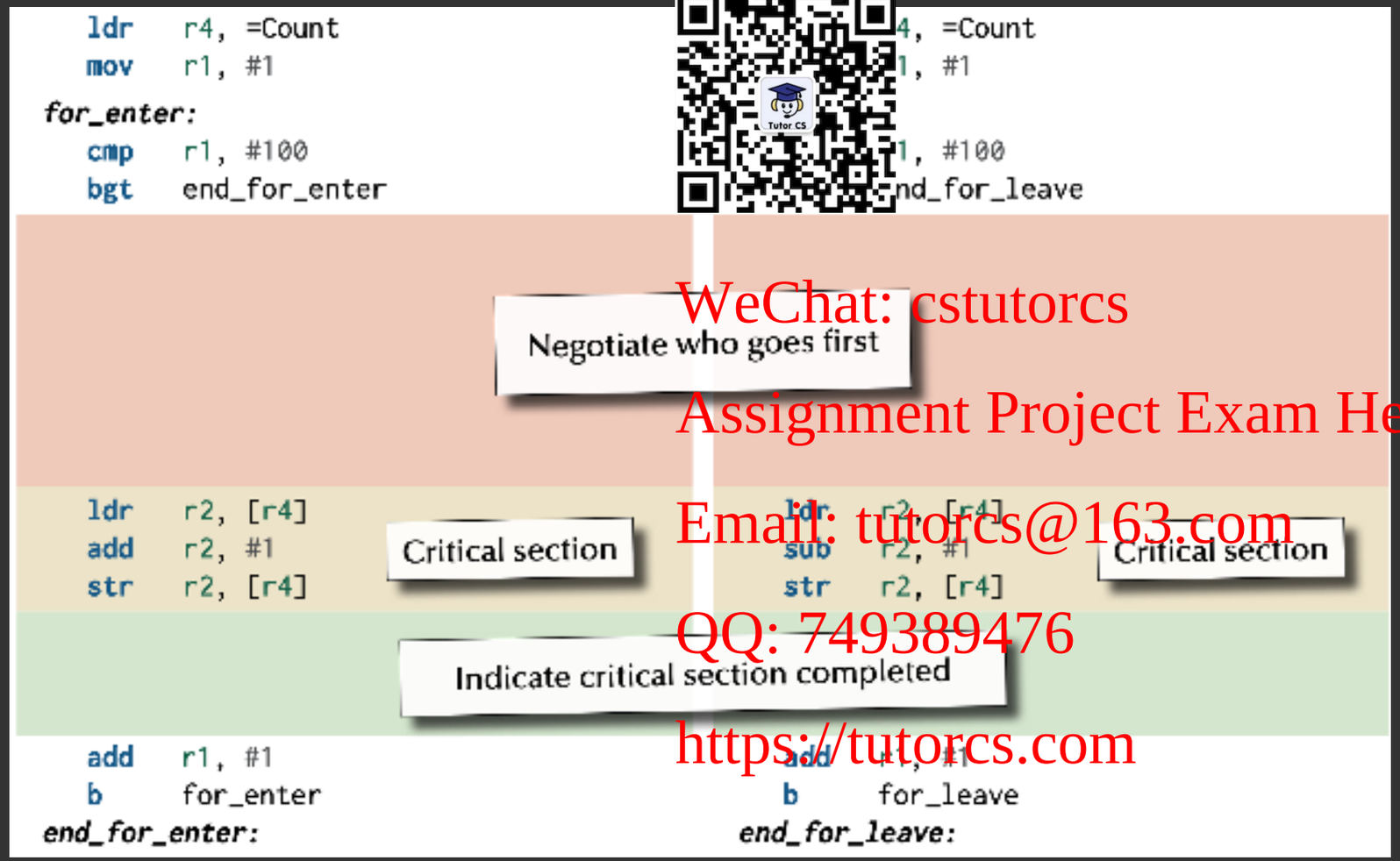
When the sequence or timing of *execution* has an effect on the outcome.

Can result in bugs! (e.g., in Assig What is the value at `Count` in this code?

```
Count: .word 0x00000000

    ldr   r4, =Count
    mov   r1, #1
for_enter:
    cmp   r1, #100
    bgt   end_for_enter
    ldr   r2, [r4]
    add   r2, #1
    str   r2, [r4]
    add   r1, #1
    b     for_enter
end_for_enter:
```

```
    ldr   r4, =Count
    mov   r1, #1
for_leave:
    cmp   r1, #100
    bgt   end_for_leave
    ldr   r2, [r4]
    sub   r2, #1
    str   r2, [r4]
    add   r1, #1
    b     for_leave
end_for_leave:
```

Critical Section

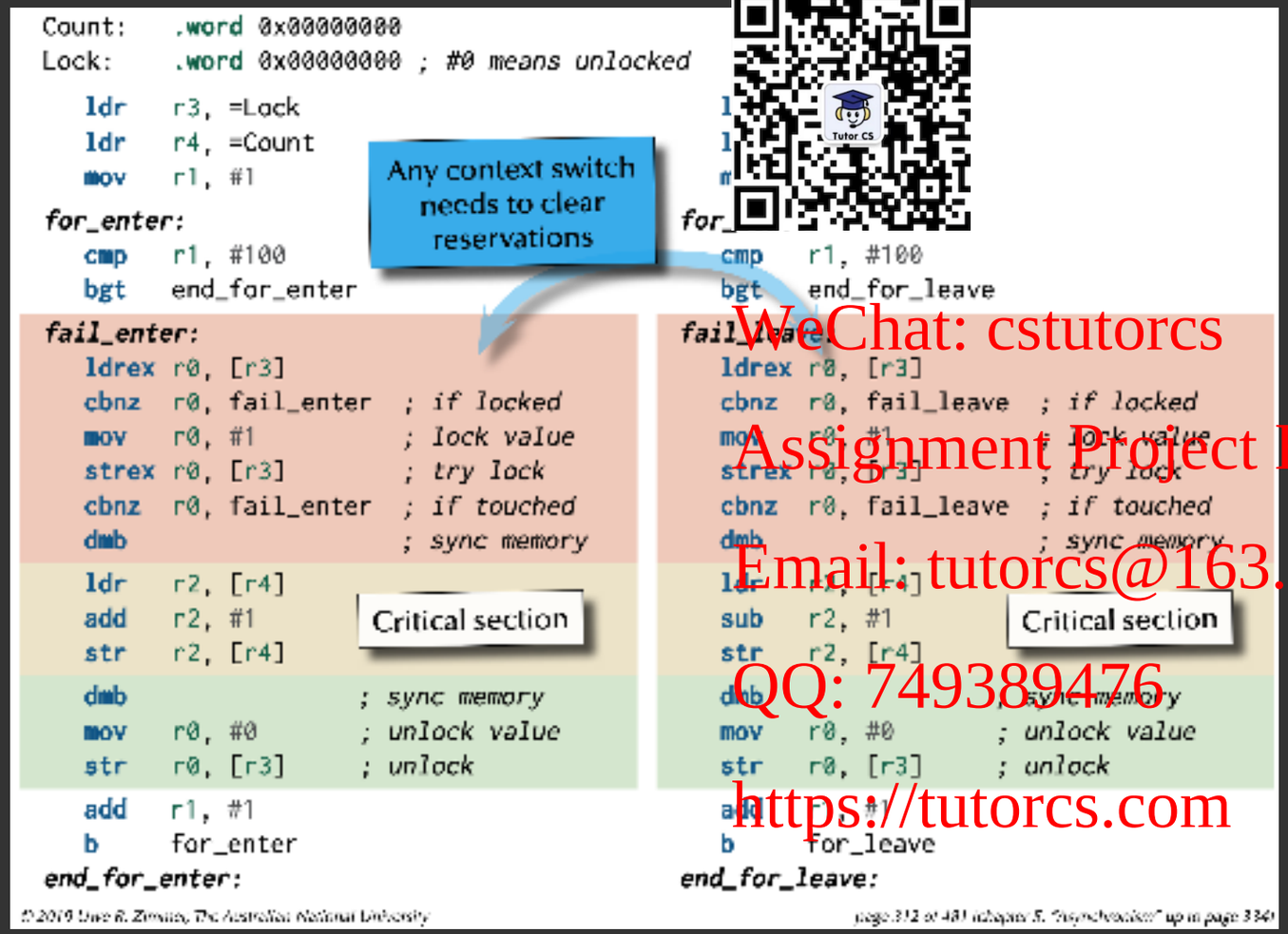


What does ARMv7-M give us?

```
Count: .word 0x00000000

   ldr   r4, =Count
   mov   r1, #1
for_enter:
   cmp   r1, #100
   bgt   end_for_enter
enter_strex_fail:
   ldrex r2, [r4] ; tag [r4] as exclusive
   add   r2, #1
   strex r2, [r4] ; only if untouched
   cbnz  r2, enter_strex_fail
   add   r1, #1
   b     for_enter
end_for_enter:
```

```
   ldr   r4, =Count
   mov   r1, #1
for_leave:
   cmp   r1, #100
   bgt   end_for_leave
leave_strex_fail:
   ldrex r2, [r4] ; tag [r4] as exclusive
   sub   r2, #1
   strex r2, [r4] ; only if untouched
   cbnz  r2, leave_strex_fail
   add   r1, #1
   b     for_leave
end_for_leave:
```

Using a “lock” variable



talk

What's mutual exclusion?

Can this be achieved on a microl

How would you do it?

Networks

Networks Main Topics (in short)

- Transmission mediums
- Communications protocols

- Packet switched/circuit switched
- Simplex/duplex
- Topology
- Parallel/Serial
- Timing and Synchronisation
- OSI reference model (7-layers!)

talk

How many transmission mediu name?

If you were stuck on a desert island what transmission media could you use to send a message for help?

What is a communications protocol?

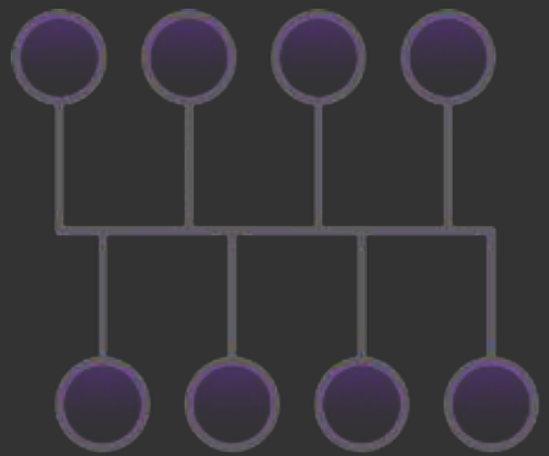
Why would it be needed?

Explain your answer.

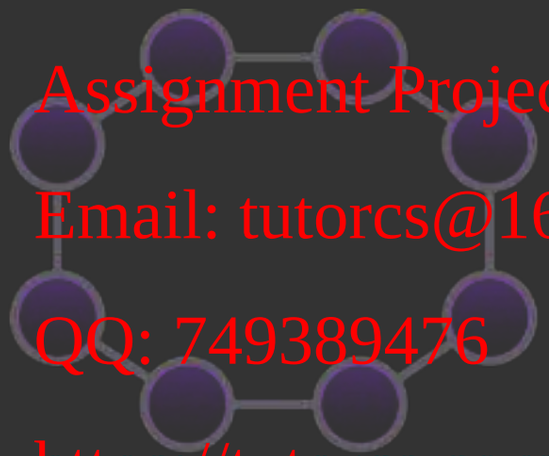
Topology

topology is the way that the nodes are connected to one another (both physically and logically)

there are several different ways to connect the nodes together, each with pros and cons



bus



ring

star

Serial vs parallel

serial

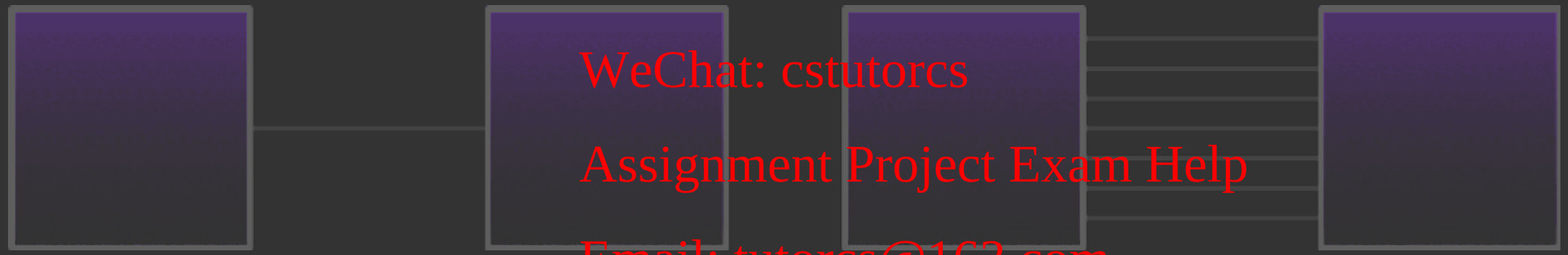
data is sent one-bit-at-a-time

fewer bits sent per signal, but si

llel

iple bits sent simultaneously (e.g. multiple wires)

to keep all the connections in sync



Synchronous vs Asynchronous

synchronous

transitions on a *clock line*

no clock skew issues, but require

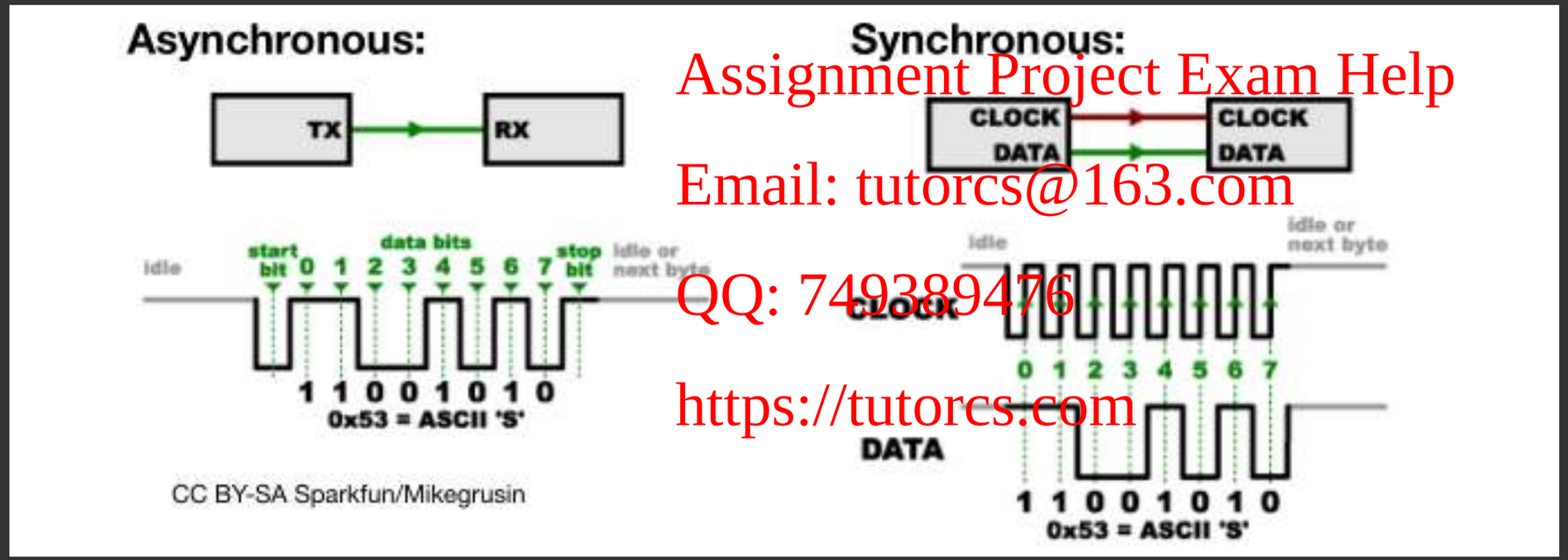
extra connection

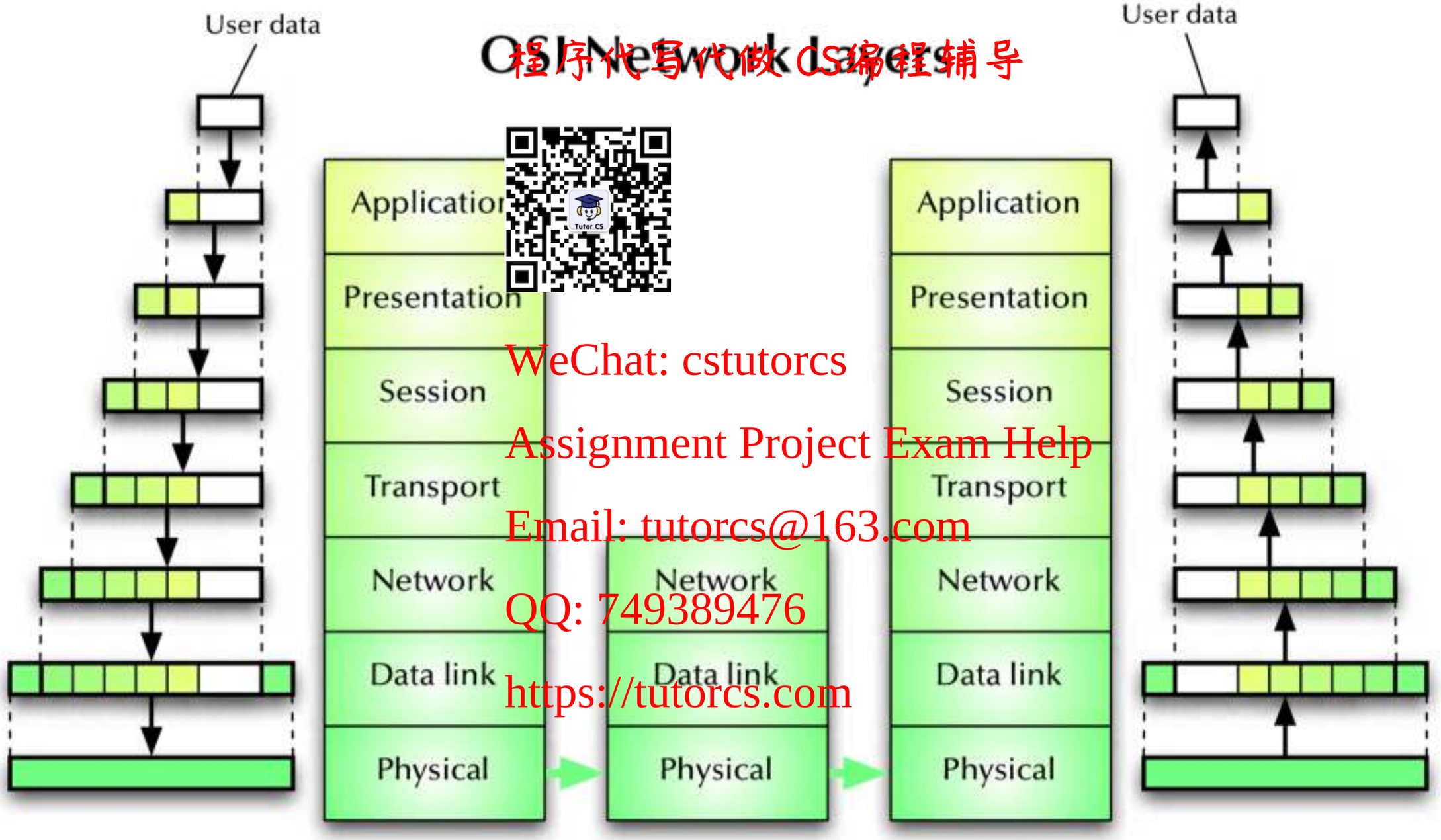
chronous

pendent) timers at each end

xtra connections required, but more vulnerable to

synchronisation issues





Operating Systems

OS Main Topics (in short)

- Operating Systems: Concept
- OS Categories

- OS Architectures
- Processes - what are processes anyway?
- How do OSs handle processes?
- Scheduling

Why do we need operating systems anyway?

What's an OS? …two main roles

virtual machine

provides friendly & safe environ

memory management

hardware abstraction

process management

inter-process comms (IPC)

esource manager

coordinates access to resources

processors

memory

mass storage

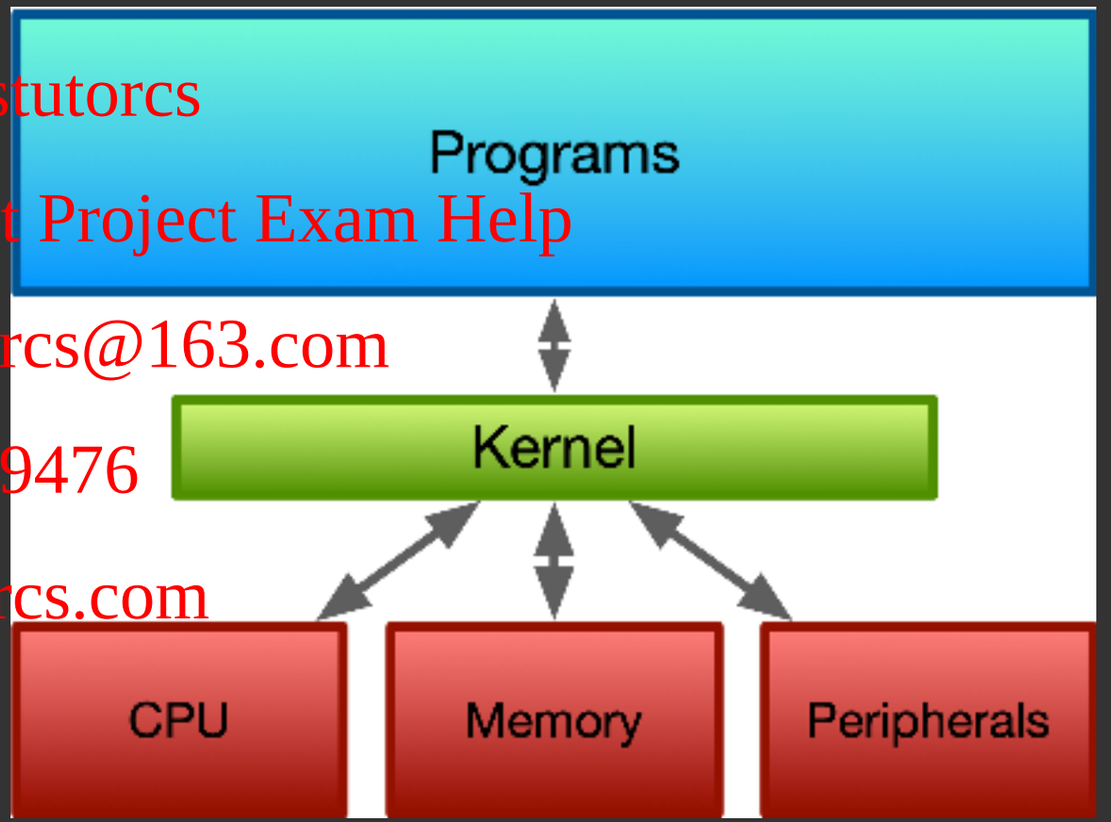
communications channels

devices (timers, GPUs, DSPs, peripherals…)

Kernel: definition

the **kernel** is the program (functi tructures in memory, etc.) which performs the *core* role(s) of the OS

access to the CPU, memory, peripherals all happens *through* the kernel through a **system call**



Monolithic OS - Modular - µKernels

talk

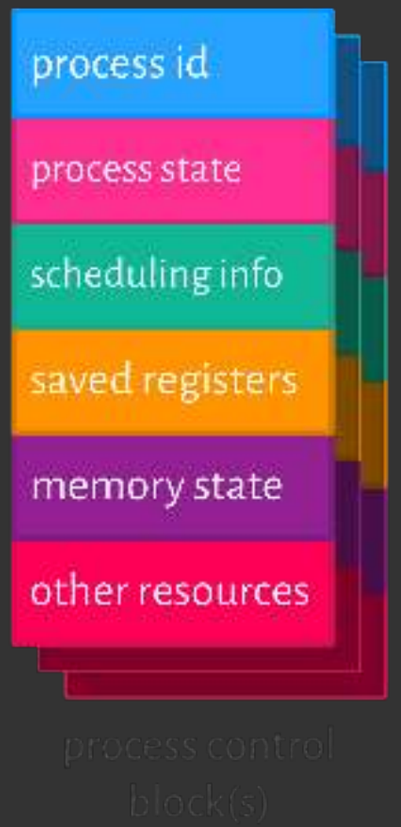
Why are kernels and (user) progr ate?

How are they different?

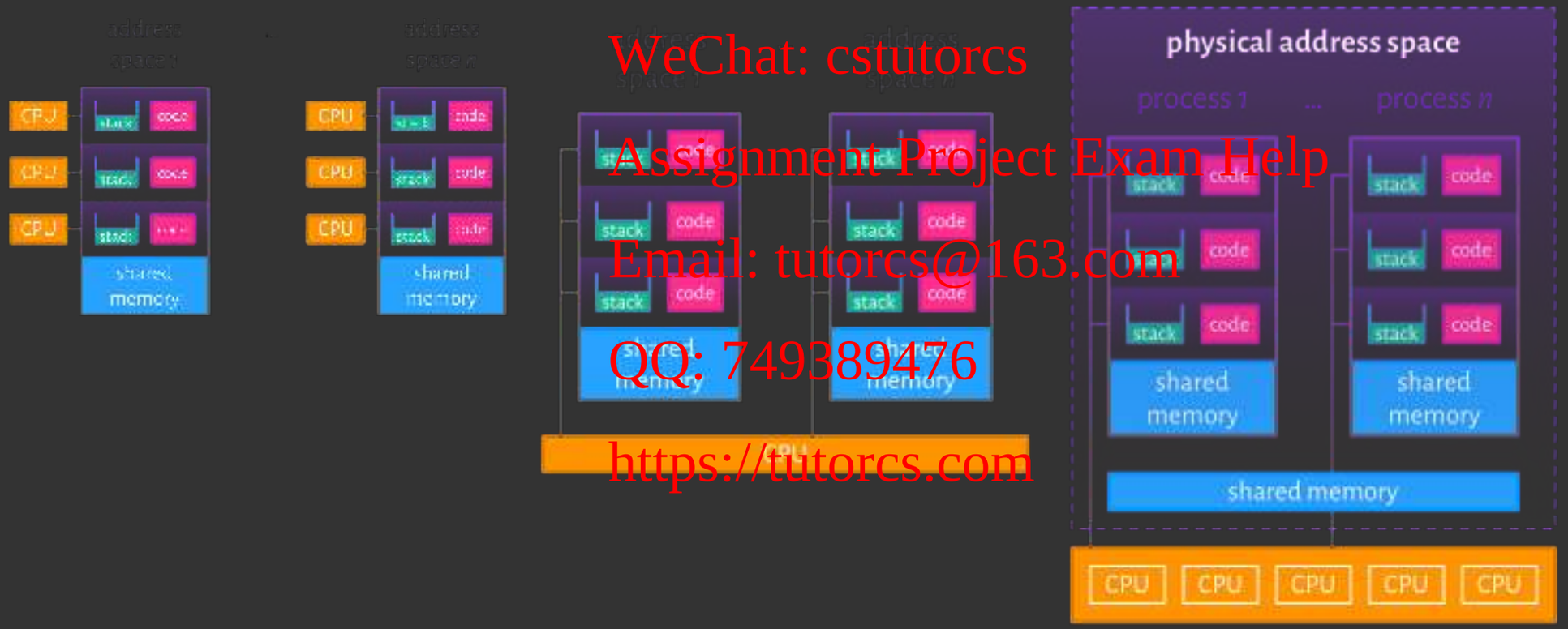
Aren’t they both programs?

Process: definition

- a *running* program
- includes the code (instructions) for the program, and the current state/context:
- registers/flags
- memory (stack and heap)
- permissions/privileges
- other resources (e.g. global variables, open files & network connections, address space mappings)



Mapping processes to CPUs



Architecture

Architecture Main Topics (in short)

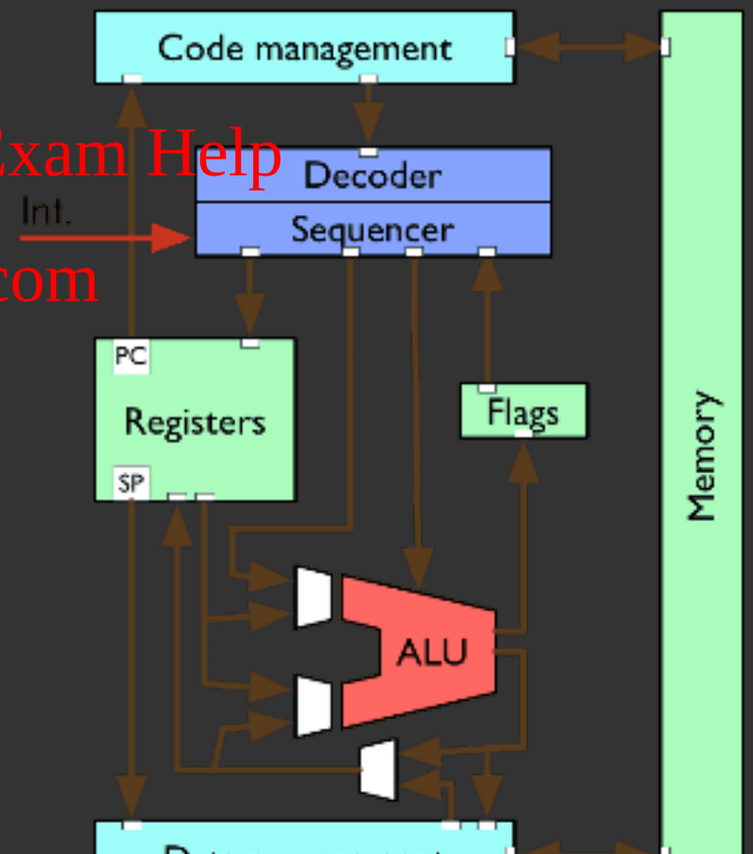
- History of computing architec
- Harvard vs von Neumann arch

- Pipelines
- Out-of-order execution
- Vector/SIMD instructions
- Hyper-threading
- Multi-core computing
- Virtual Memory
- Alternative architectures (Parallax Propeller)

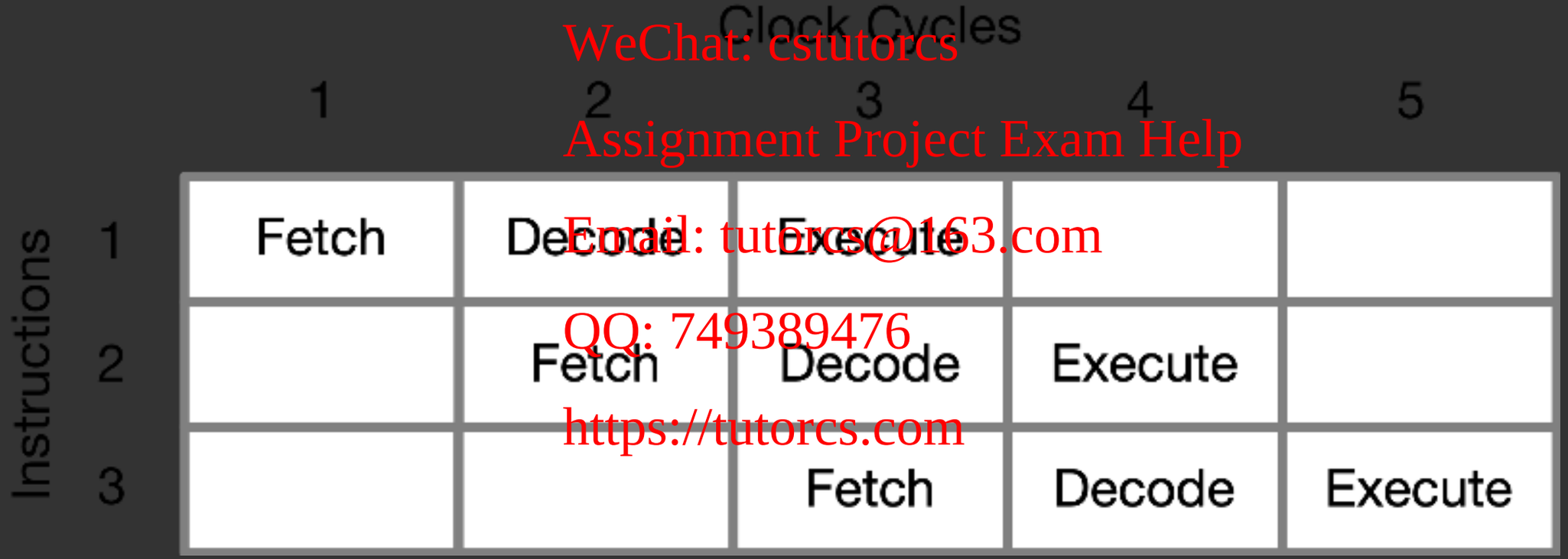
A simple CPU

- **decoder/sequencer** converts instruction[illegible]ntrol signals
- **arithmetic logic unit** (ALU) performs mat[illegible]erations
- **registers** provide small, fast storage to the CPU
- **flags** indicate the states of the latest calculations
- **code/data management** for loading/storing, caching
- **memory**

Are any of these components unneccessary?



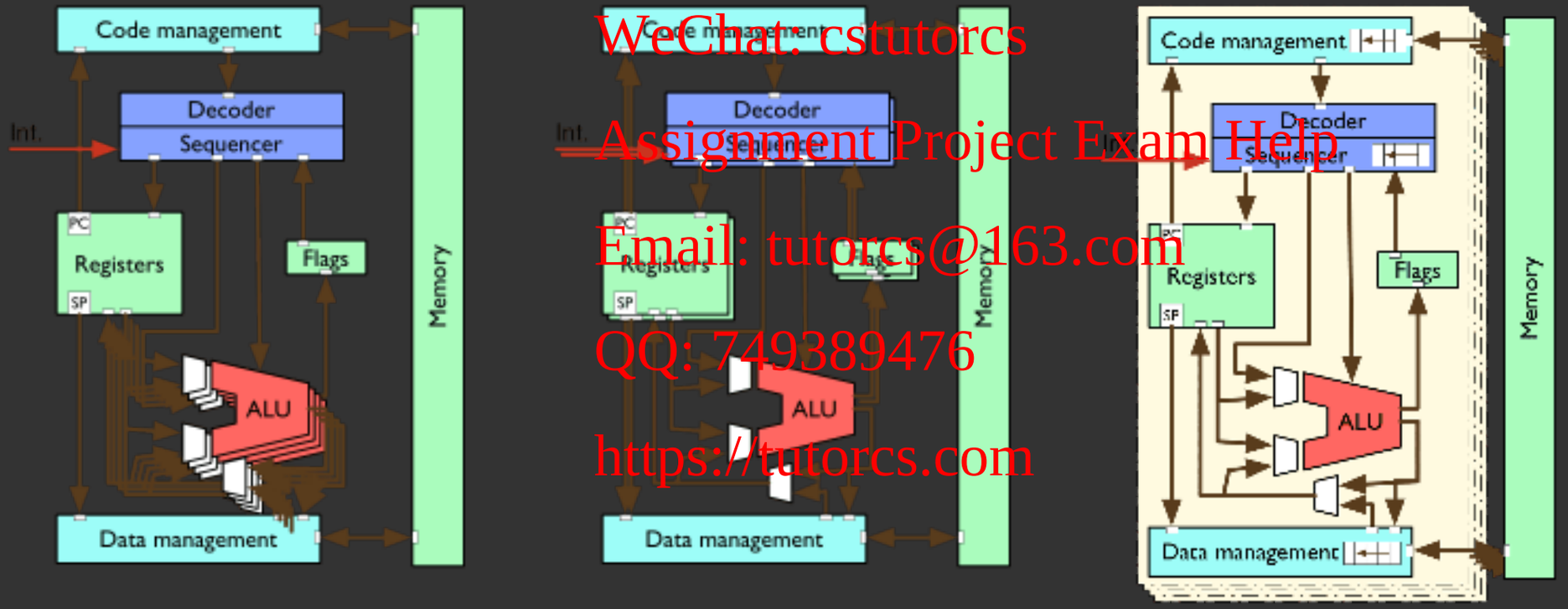
Simple Pipeline



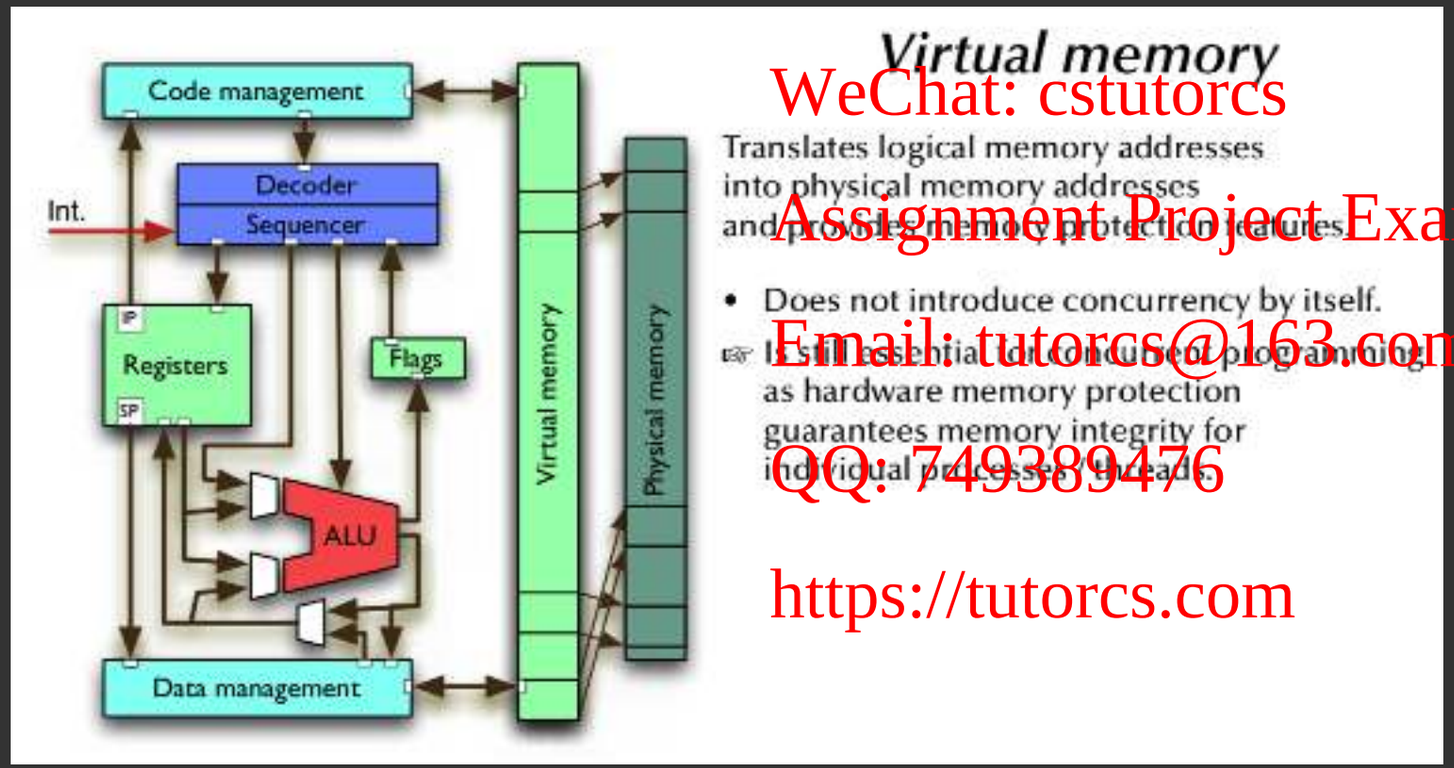
What can go wrong in a pipeline?

Vector/SIMD vs hyper-threading vs multi-core

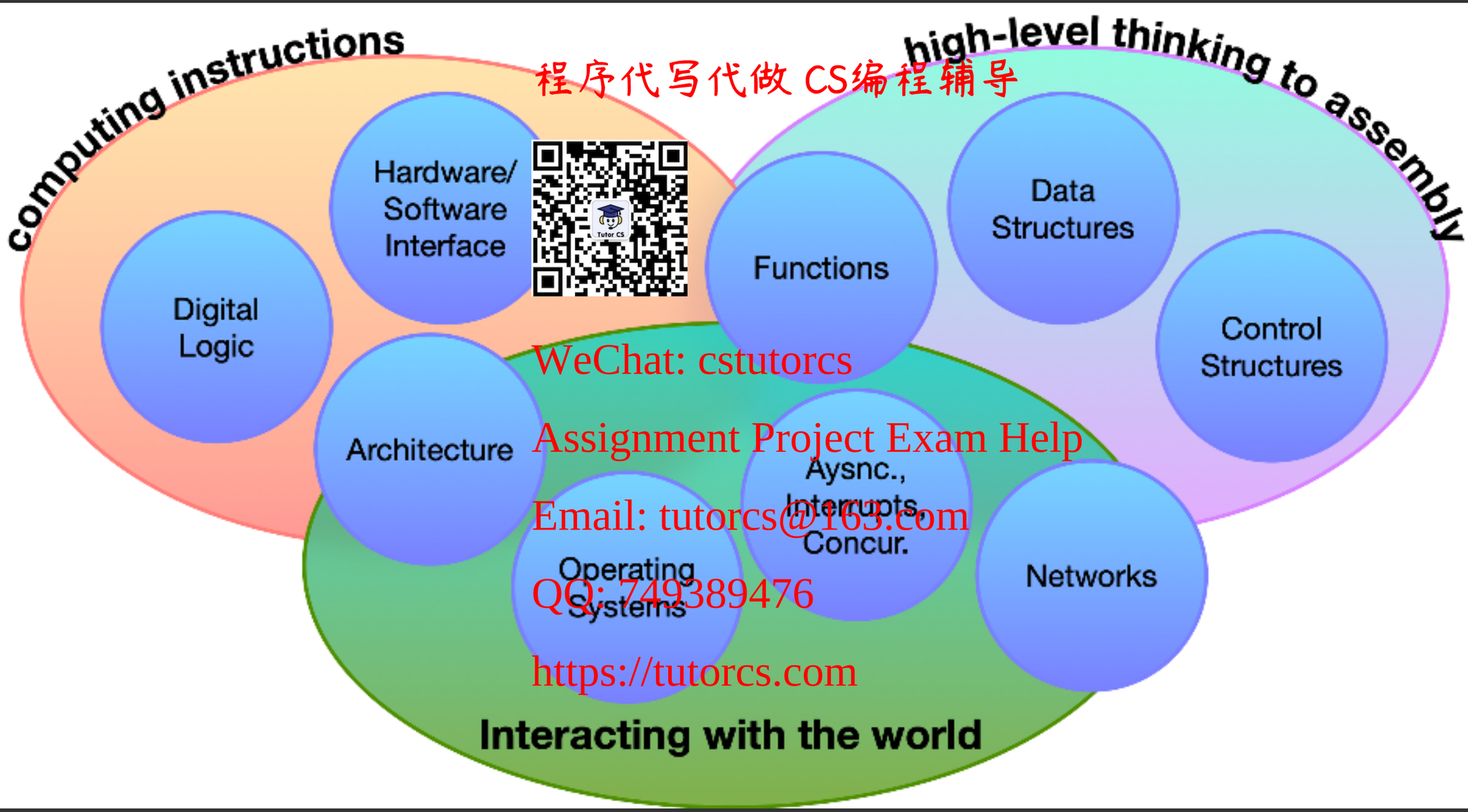
What are the differences?



Virtual Memory



Is virtual memory an architecture (hardware) topic or an OS (software) topic?



Finally done. That was epic, thanks for coming with me everybody!