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

28/10/2019

School of Computing

(University of Plymouth)

Outline

- Von Neumann archit
- What is the CPU?
- How CPU works?
  - Arithmetic Logic Unit (ALU)
  - Control Unit (CU)
  - Bus
  - Memory
  - Registers
  - Clock
- Memory Hierarchy
- Secondary Memory
- Instruction Pipeline

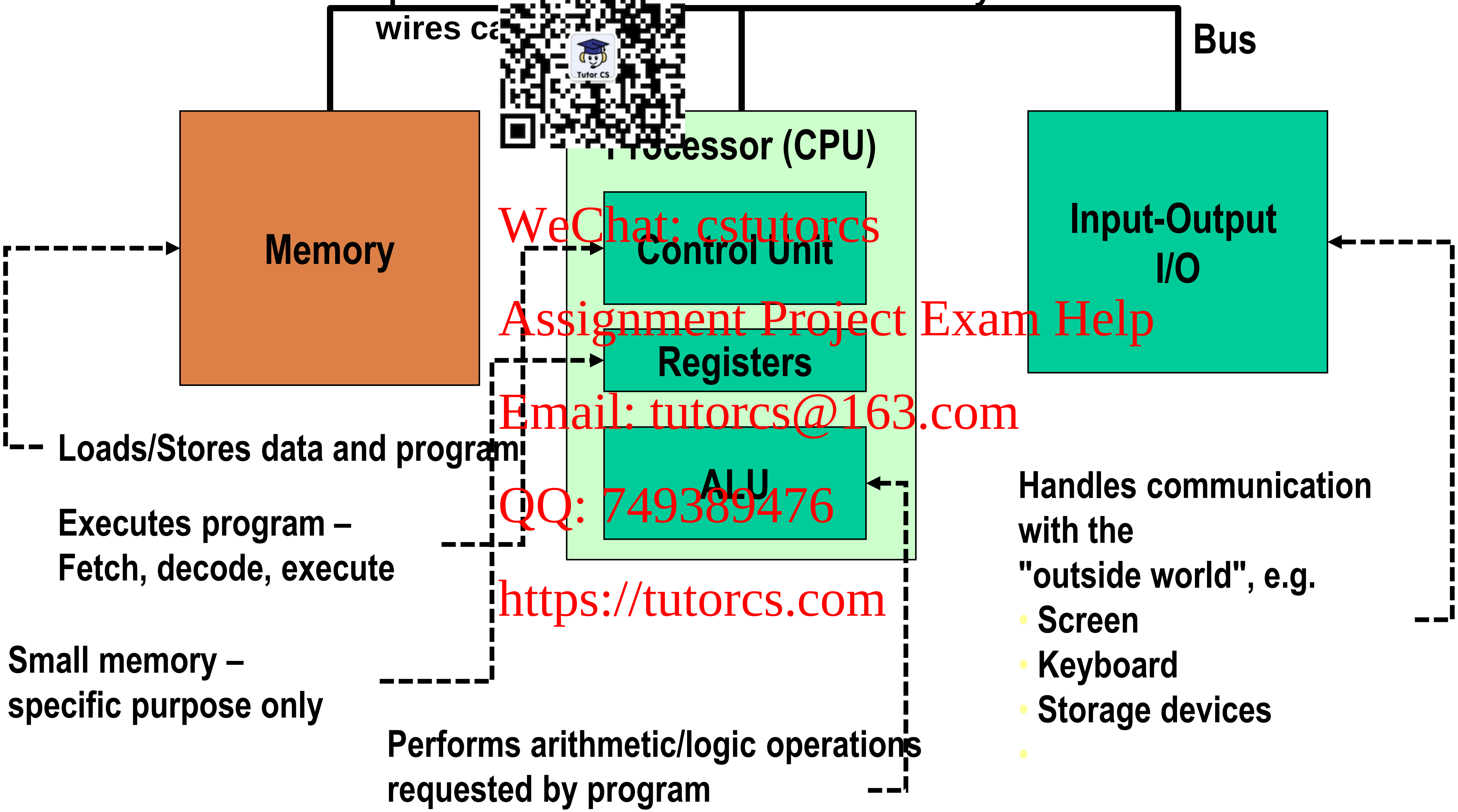
The Von Neumann Architecture

- All computers more or le[illegible] the same basic design, the **Von Neumann architecture**
- It is a Model for designi[illegible]ding computers, based on the following three characteristics:
  1. The computer consists of four main sub-systems:
     - Memory
     - ALU (Arithmetic/Logic Unit)
     - Control Unit
     - Input/Output System (I/O)
  2. Program is stored in memory during execution
  3. Program instructions are executed sequentially

- ✓ The architecture is named after the mathematician, John Von Neumann
- ✓ A variation of this architecture is the **Harvard** architecture which separates data and instructions into two pathways

The Von Neumann Architecture

The parts a d to one another by a collection of wires ca

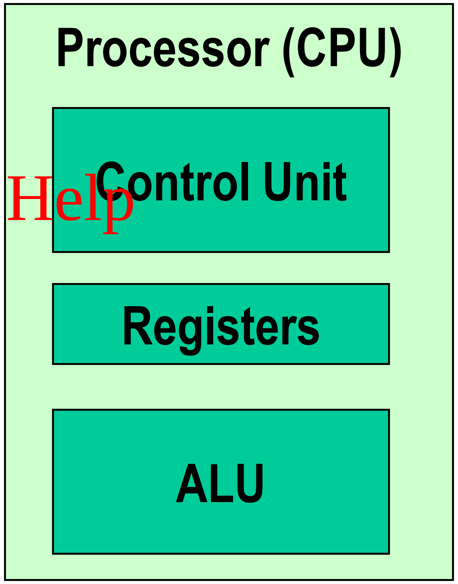


Central Processing Unit (CPU)

- Carries out the program [illegible]s!
- Operates on data it finc[illegible]puter's memory
  - Includes all binary [illegible] carry out arithmetic & logic operations- reduced to a single Integrated Circuit
- **CPU has four key parts that we will examine: Control Unit, Arithmetic & Logic Unit, Registers, Clock**
- CPUs support a set of very simple instructions that typically fall into the following categories:
  - Data movement (load, store, copy…)
  - Arithmetic/logical (add, subtract, compare..)
  - Program control (branch, jump…)
- Very primitive commands (operations) executed by the CPU
- These commands are implemented as electronic binary circuits which can transform the 0s and 1s.

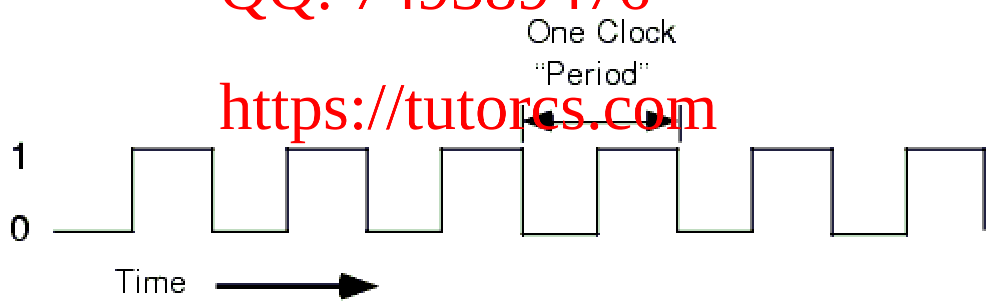
Central Processing Unit (CPU)

- CPU is responsible for [illegible] program instructions, **decoding** each instruction that is fetch[illegible]**xecuting** the indicated sequence of operations **on the cor**[illegible]
- **The key parts of the CPU are**
  1. **Arithmetic Logic Unit (ALU)**
  2. **Control Unit (CU)**
  3. **Registers**
  4. **Clock**



CPU Clock (1)

- **Every computer contains [illegible]l clock that regulates the rate at which instructions are execute[illegible]hronizes all the various computer components**
- All CPU and bus operations are synchronized to the clock
- Clock speeds are expressed in megahertz (MHz) or gigahertz ((GHz)
  - the beginning of each cycle is when the clock signal goes from "0" to "1"
  - e.g. CPU frequency 2 GHz -> clock cycle 0.5 ns
  - In the computer, all timings are measured in terms of clock cycles, e.g., an addition needs 2 cycles



CPU Clock (2)

- The faster the clock, the r[illegible]tions the CPU can execute per second
- But, to think that clock an[illegible]nce is the same thing is the most common misconception about proc[illegible]

- CPU frequency is not necessarily indicative of the execution speed; e.g. complex operations, cache misses etc.
  - FLOPS: floating operations per second, a useful measure for (super)computers dedicated to extensive computations

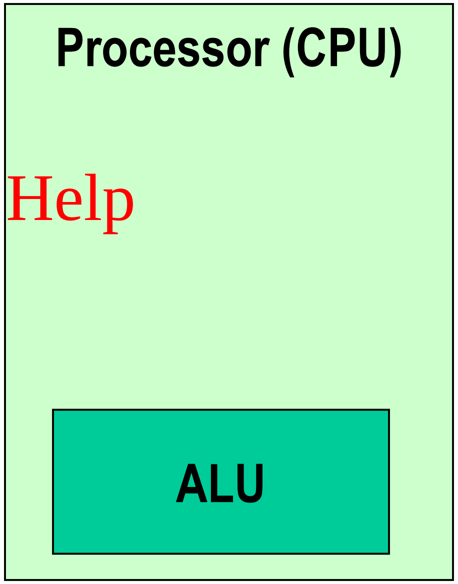
- A typical modern PC now has either four or five different clocks, running at different (but related) speeds, e.g., system (memory) bus, L2 cache bus, PCI bus
- The entire system is tied to the speed of the system clock. This is why increasing the system clock speed is usually more important than increasing the CPU clock
  - Normally, the processor spends a significant amount of time waiting on data and signals from much slower devices, e.g., memory

Registers

- In a computer, a register ... st memory
- Registers are fast stand- ... ge locations that hold data temporarily
- Multiple registers are needed to facilitate the operation of the CPU
- The main registers are:
  - The Instruction Register (IR)
  - The Program Counter (PC) or Instruction Pointer (IP)
  - Data registers

Central Processing Unit (CPU) (1)

- The arithmetic logic unit [illegible]s out the **logic operations** (such as comparisons) and **arithm**[illegible]**ions** (add, shift, multiply) required during the program execution
- **The ALU knows**
  1. which operation to perform
  2. Where are the input data
  3. Where to store the output data
  - All the above are provided by the CU
- **The ALU performs**
  1. **Integer** arithmetic operations
     - Add, subtract, increment, decrement
  2. Bitwise logical operations
     - AND, OR, XOR, NOT, Arithmetic shift, logical shift, rotate



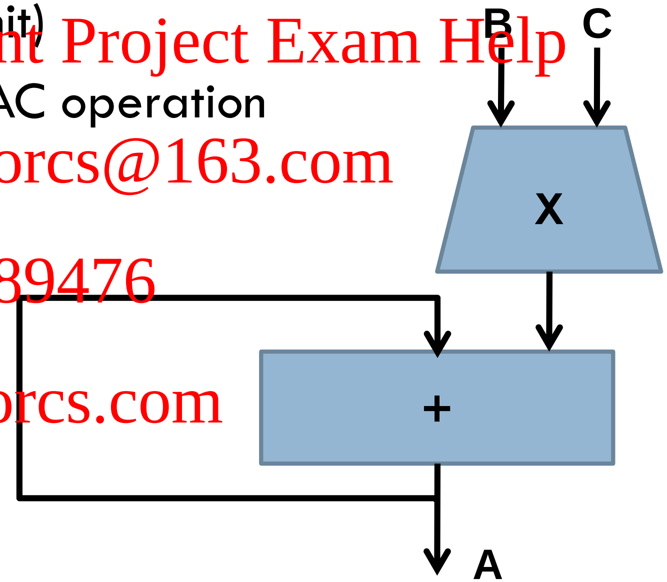
Central Processing Unit (CPU) (2)

- ALUs often handle the m[illegible] of two integers, since the result is also an integer
- ALUs typically **do not perform division operations,** since the result may be a fraction, or a "floating point" number
- **The floating-point unit (FPU)** performs operations on floating point numbers
- Modern CPUs have separate unit to perform multiplication faster
- **Processors include a coprocessor hardware unit which is used to perform much more complex mathematical operations such as arcsine, cosine, floating-point division, etc**

Central Processing Unit (CPU) (3)

- **The multiply-accumulat … n is a common step that computes the product of two number … that product to an accumulator**
- It speeds up many comp … t involve the accumulation of products, e.g., Matrix-matrix Multiplication
  - The hardware unit that performs the operation is known as a Multiplier Accumulator unit (MAC, or MAC unit)
  - the operation is called MAC or MAC operation
  - The MAC operation performs

$$A = A + B \times C$$



Central Processing Unit (CPU) (4)

- **Arithmetic shift:** when shifting to the right, the leftmost bit (the vacant MSB) is filled with the value of the original MSB (sign)
  - Ideal for signed two's complement binary numbers

- **Logical shift** : The vacant bits are filled with zero
- the logical and arithmetic **left-shifts are exactly the same**
- Ideal for unsigned binary numbers

- **Circular shift or bit rotation** : In this operation, the bits are "rotated" as if the left and right ends of the register were joined. The value that is shifted in on the right during a left-shift is whatever value was shifted out on the left, and vice versa

Think Pair Share

- Apply logical and circ… o the following register:

1	0	1	1	0	0	1	0

Control Unit (CU) (1)

- A Control Unit is the unit ...es the central work of the computer
- **There are two registers ...rol unit**
  - The **instruction regi**... ...ntains the instruction that is being executed
  - The **program counter** **(PC)** contains the address of the instruction being executed

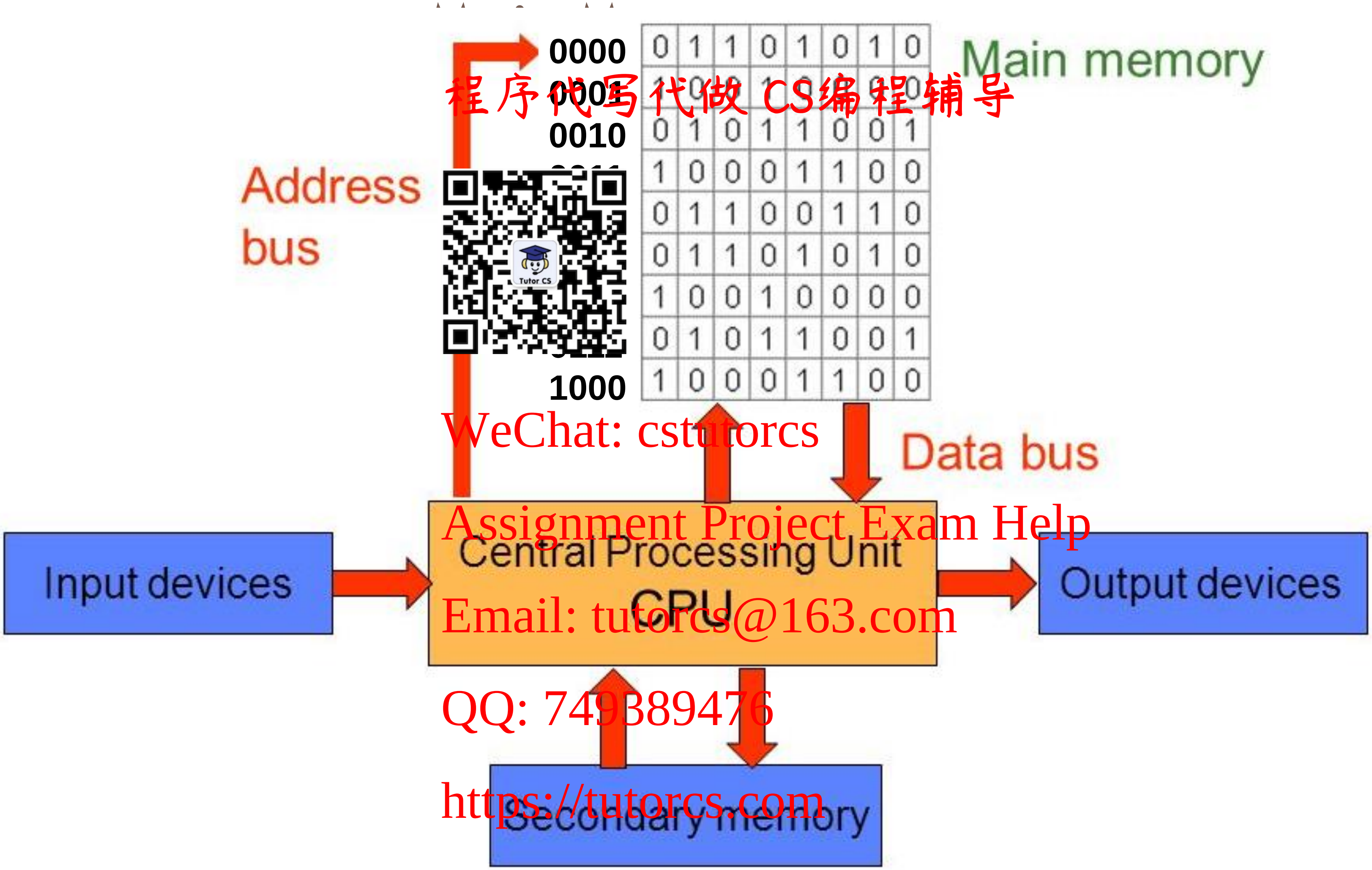
- **The CU is responsible of executing the right instruction and organizes the other function units appropriately**

- **In every clock cycle the CU:**
  - Loads from memory the next instruction to be executed – PC register contains that address
  - The instruction is stored into IR and is decoded
  - The CU sends the appropriate signals to the ALU, memory, I/O devices in order to execute the instruction
  - The CU increments PC to show next instrution

Control Unit (CU) (2)

- Program is stored in m
  - machine language i are in binary format

- The task of the control unit is to execute programs by repeatedly:
  - Fetch from memory the next instruction to be executed
  - Decode it, that is, determine what is to be done
  - Execute it by issuing the appropriate signals to the ALU, memory, and I/O subsystems
  - Continues until the program terminates (HALT instruction)


Main memory
0000
0010
1000
Address bus
Data bus
Input devices
Central Processing Unit
CPU
Output devices
Secondary memory

Main Memory

- **Computer memory con inear array of addressable storage cells that are similar to regis**
- Both program and data into memory
- Load/store operations are performed on both instructions and data

- Consists of many memory cells (storage units) of a fixed size
  - **Each cell has an address associated with it**
- All accesses to memory are to a specified address
- The time it takes to fetch/store a word is the same for all words

Main Memory - Address Space

- To access a word in memory requires an identifier. Although programmers use a name to identify a word (or a collection of words), at the hardware level each word is identified by an address
- The total number of uniquely identifiable locations in memory is called the **address space.** For example, a memory with 64 kilobytes and a word size of 1 byte has an address space that ranges from 0 to 65,535

- **If memory contains N words, then $\log_2 N$ bits are needed to address all words in memory**
- **If the memory address space needs N bits, then there are 2^N words in memory**

Think Pair Share

1. A computer has 32 M… es) of memory. How many bits are needed to address any single … ory?

The memory address space is 32 MB, or 2^{25} ($2^5 \times 2^{20}$). This means that we need $\log_2 2^{25}$, or 25 bits, to address each byte

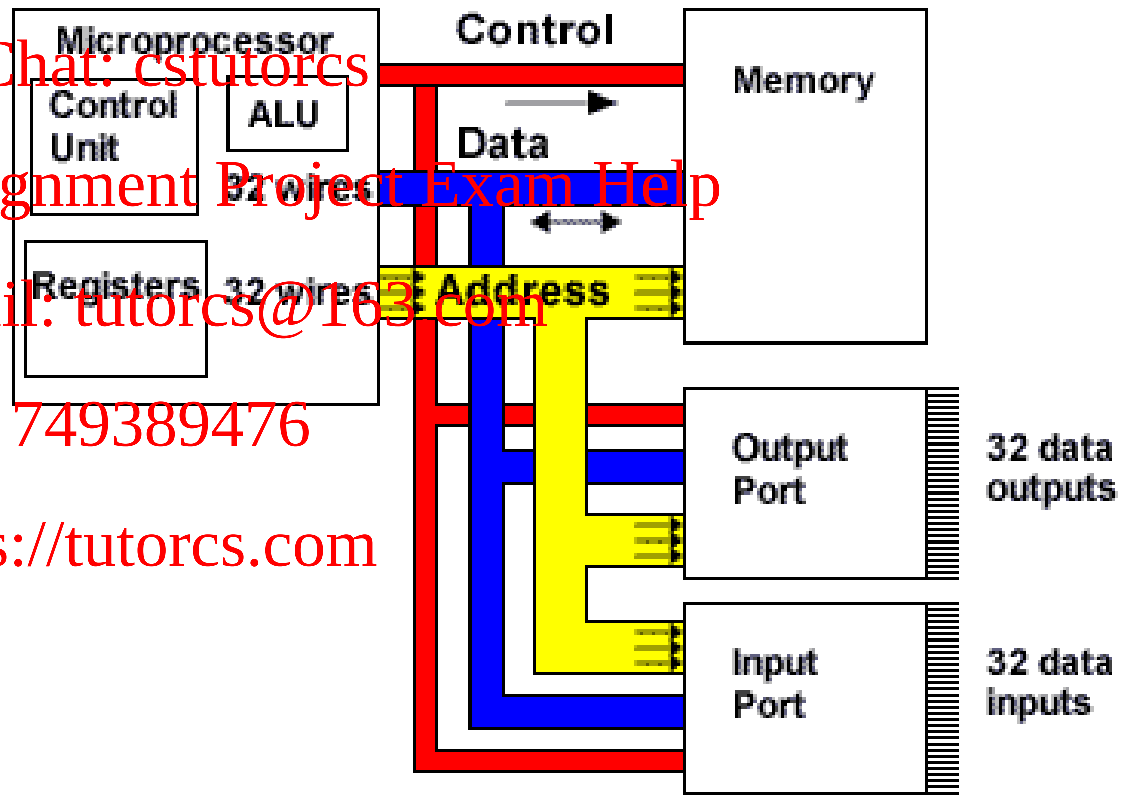
2. If main memory is of 64Mbyte and every word is of 2 bytes how many bits do we need to address any single word in memory?

The memory address space is 64 MB, which means 2^{26}. However, each word is two (2^1) bytes, which means that we have 2^{25} words. Note that (*Mem.size=number.words x word.size*)

This means that we need $\log_2 2^{25}$, or 25 bits, to address each word

How Main Memory and CPU are connected?

- Control Bus – sends app gnal whether store or load
- Address Bus – sends the ddress
- Data Bus – sends the data

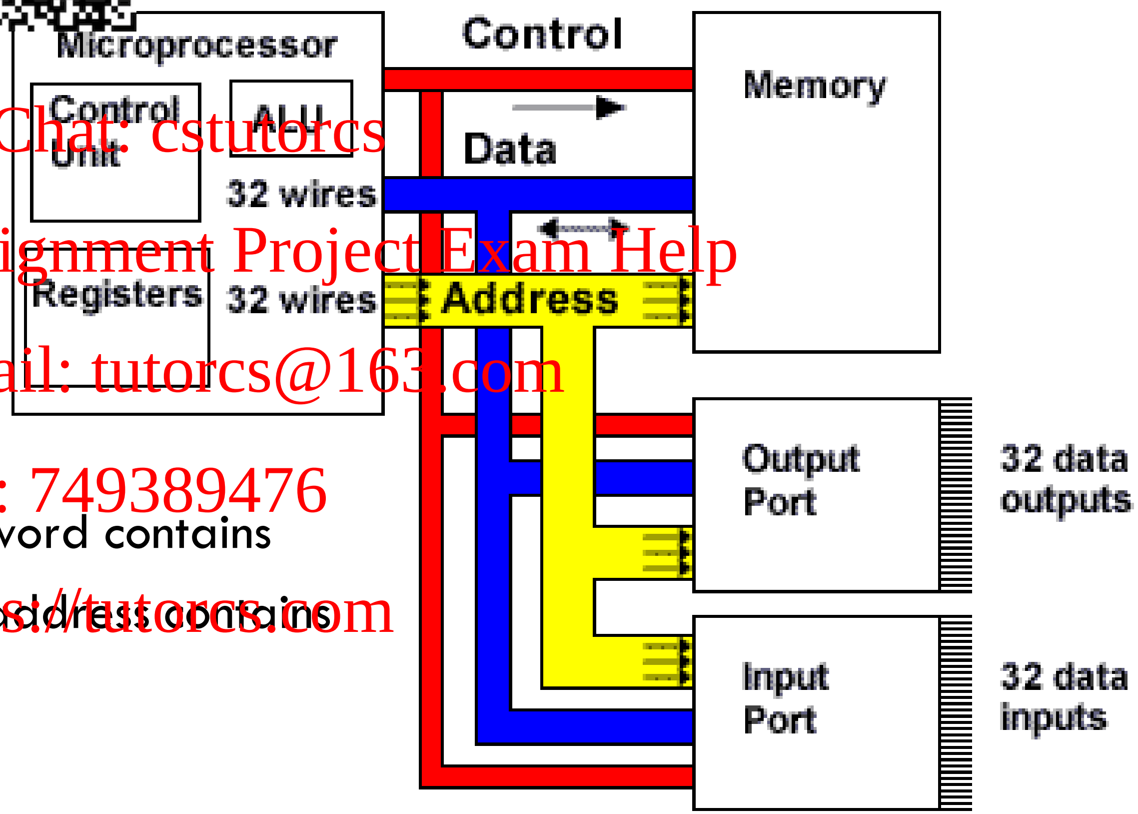


Buses (1)

- **Bus:** a group of wires tha[illegible]ata from one part to another (data, address, control)
  - ✓ **Data bus:**
    - bi-directional (read/write)
    - 8, 16, 32, 64-bit wide (same as 'word size')
  - ✓ **Address bus:**
    - specifies memory location in RAM/ROM/interface device to be accessed; monodirectional
    - address space: 16-bit wide -> 2^{16} words= 64×2^{10} = 64KB
    - 32-bit wide -> 2^{32} = 4GB – this is why in 32-bit PCs we cannot use more than 4Gbyte of RAM
  - ✓ **Control bus:**
    - carries commands from the CPU and returns status signals from the devices

Buses (2)

- How many wires nee[illegible]e?
  - A. Data bus?
  - B. Address bus?
  - C. Control Bus?



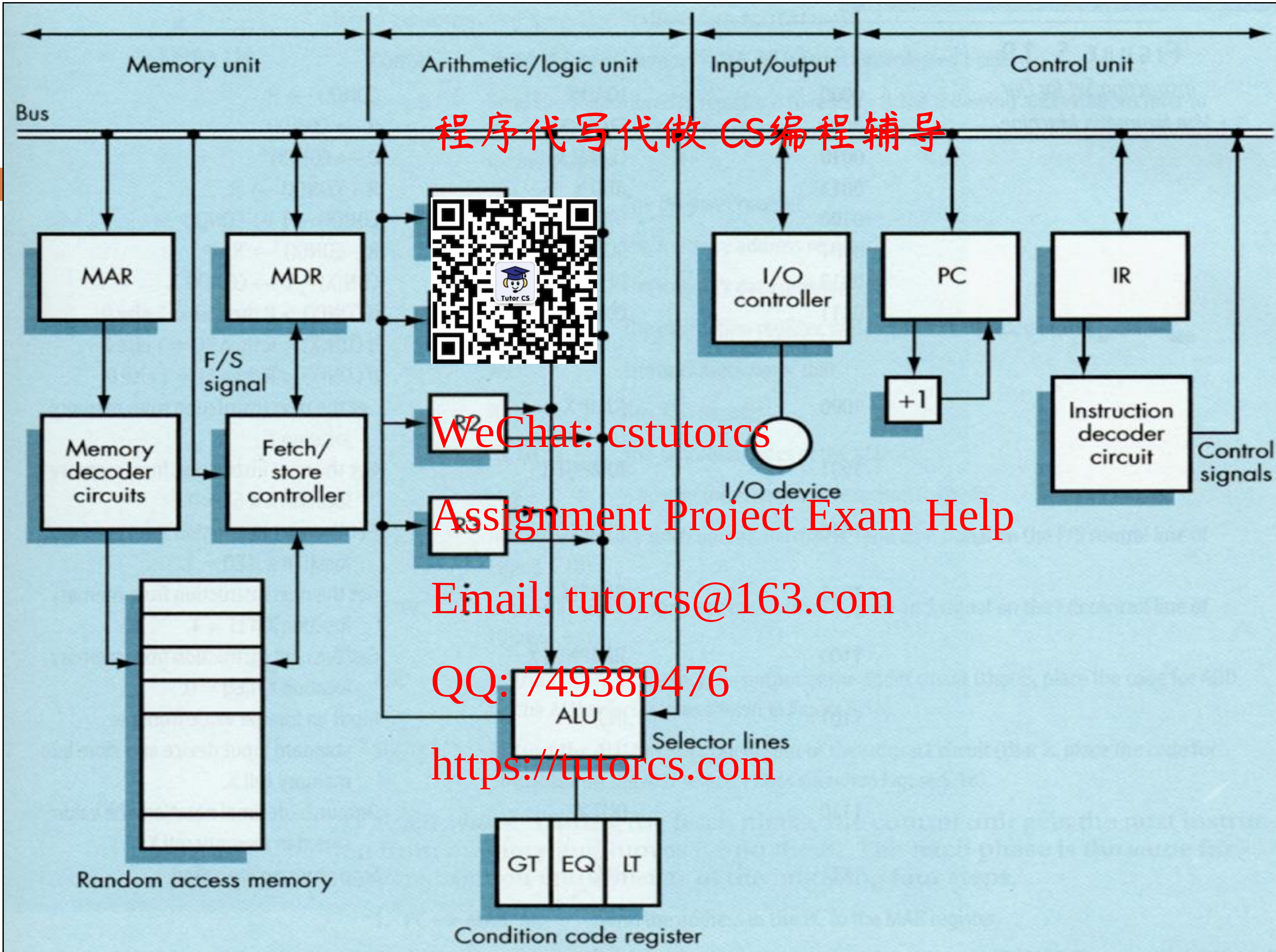
A. As many bits as the memory word contains

B. As many bits as the memory address contains

C. 1 bit is enough

Think Pair Share

Q: If main memory is of 64K very word is of 8 bytes how many wires do we need for the address


Memory unit
Arithmetic/logic unit
Input/output
Control unit
Bus
MAR
MDR
F/S signal
Memory decoder circuits
Fetch/store controller
Random access memory
R2
I/O controller
I/O device
PC
+1
IR
Instruction decoder circuit
Control signals
ALU
Selector lines
GT
EQ
LT
Condition code register

Machine Language or Machine Code (1)

- A *program* consists of ce of instructions (in binary)
- EACH instruction spec
  - The operation to p
  - The address of the data

- Instructions are stored and processed in *machine language*--also called ***microcode***
- Like everything else (e.g. like ASCII characters) machine language consists solely of **bit patterns**

Machine Language or Machine Code (2)

- **Machine languages [illegible]tirely of binary numbers and are almost impossible fo[illegible] to read and write**
- **Assembly language**[illegible] same structure and set of commands as machine languages, but they enable a programmer to use names instead of numbers

- Each type of CPU has its own machine language and assembly language

  - an assembly language program written for one type of CPU won't run on another

- In the early days of programming, all programs were written in assembly language

- **Now, most programs are written in a high-level language** such as Java, Python, C/C++
- Programmers still use assembly language when speed is essential

Machine Language or Machine Code (2)

- **A machine language ins**[illegible]**nsists of:**
  - **Operation code,** spe[illegible]ch operation to perform
  - **Address field(s),** specifying the memory addresses of the values on which the operation works

Opcode (8 bits)	Address 1 (16 bits)	Address 2 (16 bits)
00001001	0000000001100011	0000000001100100

- **The above could be (ADD 99, 100)** , assuming that the opcode for ADD instruction is 9
  - Add content of memory locations 99 and 100, and store back in memory location 99)

- Instructions are given to the processor in the form of a program … so it knows what circuits to use, in what order; and from where the data should be read or to where it should be stored

Assembly basic instructions

Depending on the target CPU[illegible]assembly instructions exist

<u>Instr:</u>	<u>Me[illegible]</u>
Store 0xA2B , R1	St[illegible]2B$_{16}$ memory location
ADD R1, R2	Add R1 and R2 and store the result in R1
SUB R1, R2, R3	R1 = R2- R3
MUL R1, R2	R1= R1 * R2
DIV R1, R2	R1= R1 / R2
INC R4	R4=R4+1
HALT	Stops program execution
CMP R1, 10	If R1==10, then set the register EQ=1, else EQ=0
JMP EQ S1	Load next instruction from memory location S1, if EQ==1

- **Assemblers** translate instructions that are comprehensible to humans into the machine language that is comprehensible to computers

High level language VS Assembly language

- Easily understandable
- Portable - **run on just any co**
- Debugging of the code is eas
- High level languages like java, C++, etc. have **one to many relationship with assembly**, i.e., one statement of java expands into many assembly language commands
- **High level language is always converted into assembly language**
- The high level language programmer doesn't need to know the HW details

- Hard to understand
- **Runs only on the target CPU only**
- Debugging is very hard
- **One to one or one to a few relationship**
- The assembly language programmer must know about the hardware such as registers, etc.
- Assembly language can control the machine code better
- **Assembly is much faster**

High level language VS Assembly language

High Level Language

a = x + y - z

Assembly Language (AL):

load x into r1

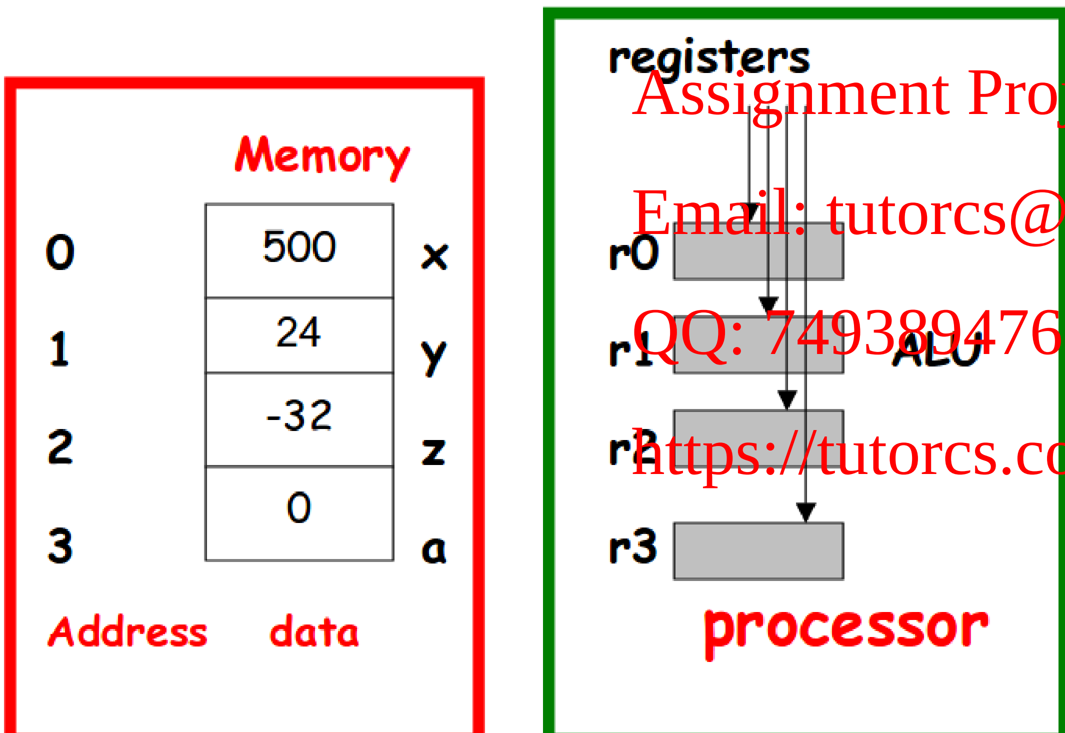
load y into r2

load z into r0

r3 ← r1 + r2

r0 ← r3 − r0

store r0 into a



- In HLL we write a=5 and we assume that it is stored somewhere – we don't care
- But the computer does not work like that – it has to store every variable in an exact memory location

- **In assembly we must specify the Hardware registers as well as the memory locations**

Program Execution

1. PC is set to the addres[illegible]e first program instruction is stored in memory

2. Repeat until HALT ins[illegible] fatal error

2a. Fetch instruction

- Fetches instruction (from memory) at address given by PC; copies it into storage register

2b. Decode instruction

- Copies op code into IR and operands into address registers
- Interprets instruction
- ALU is invoked

2c. Execute instruction

- Execution cycles vary, depending on the op code (instruction), e.g., Load copies data from memory to ALU register; ADD adds values inside the ALU

2d. Increment PC by one, now contains the memory address of the *next* instruction that will be fetched

A simplified example

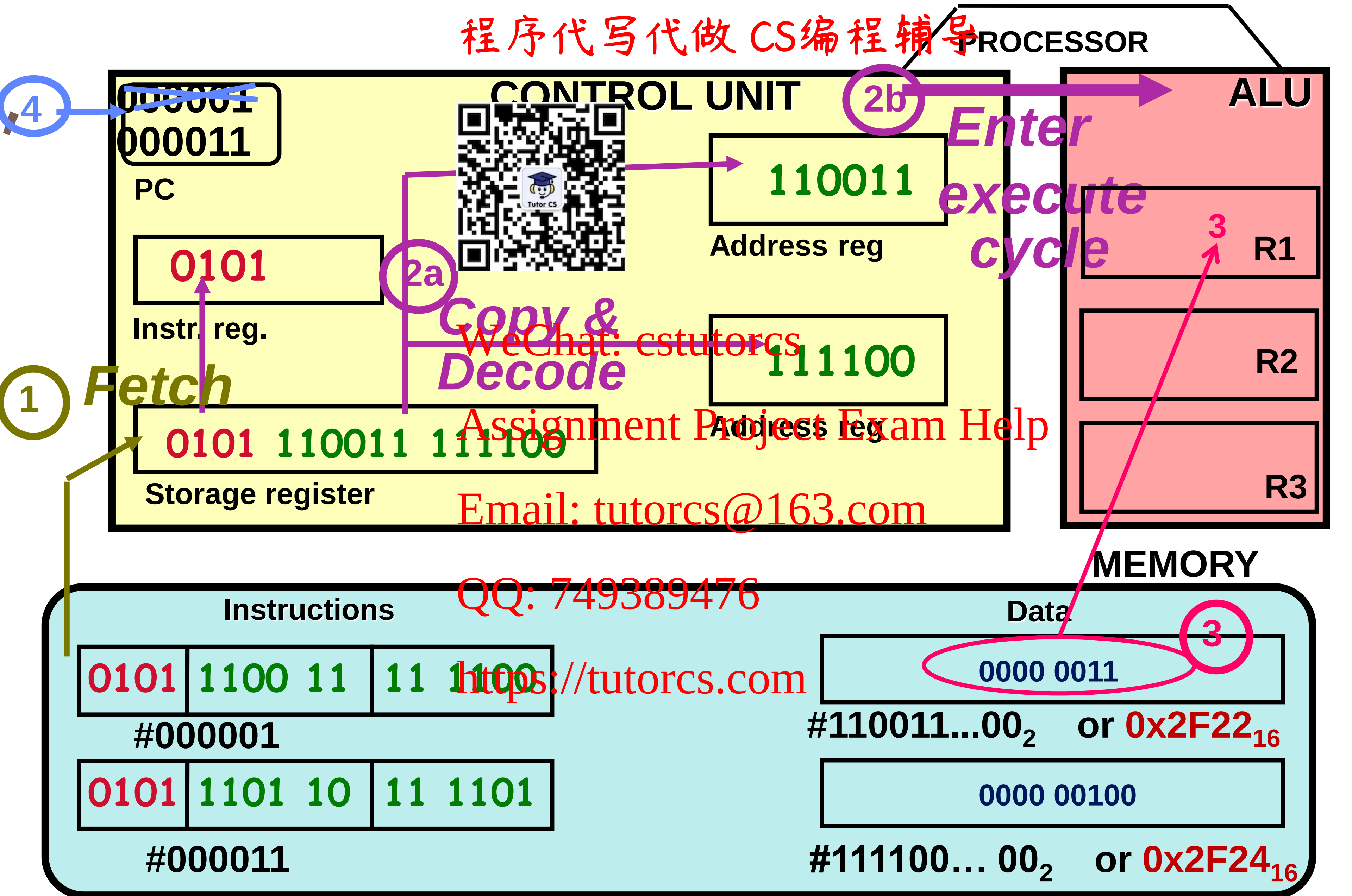
R1, 0x2F22

R2, 0x2F24

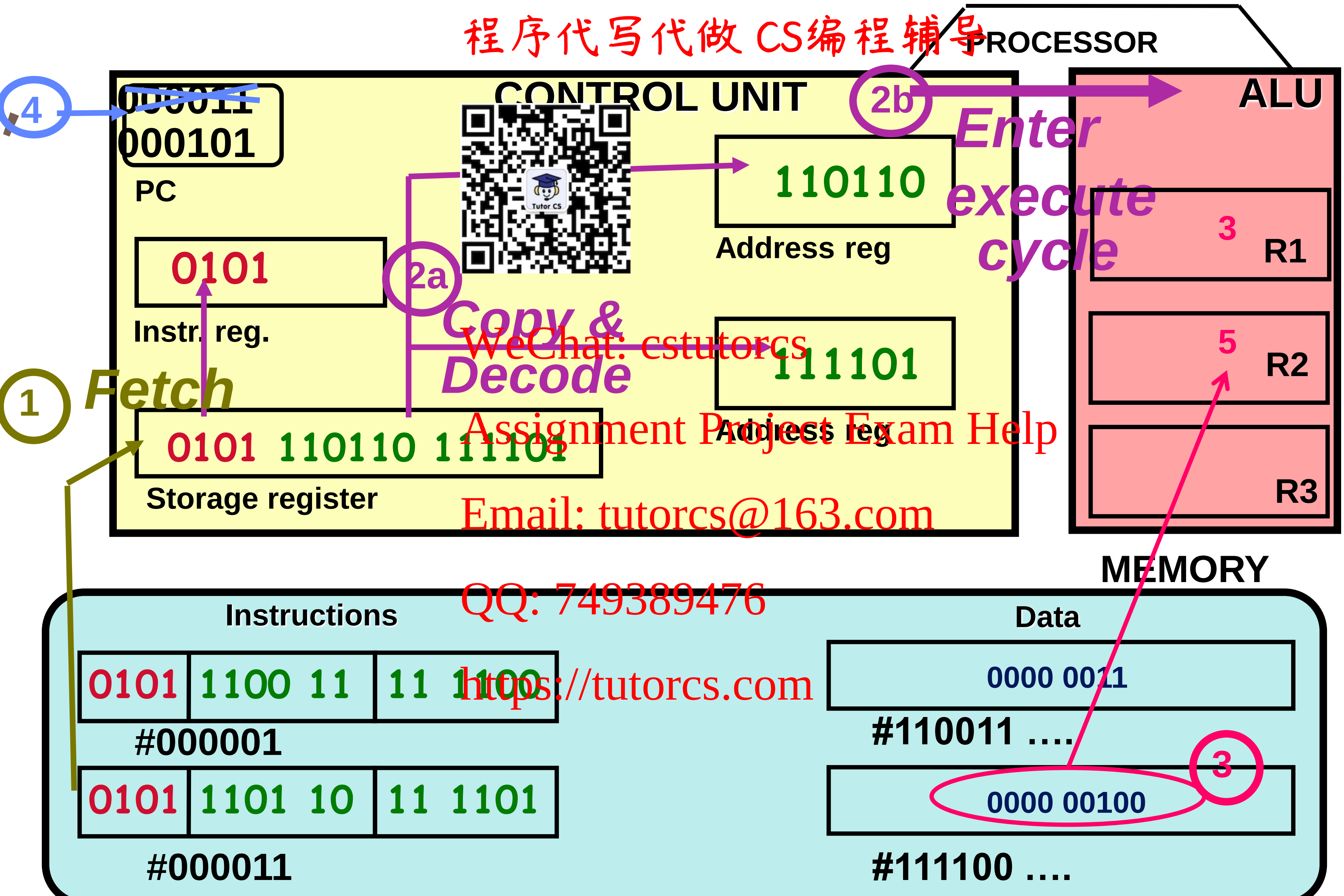
ADD R3, R1, R2

Store 0x2F24, R3

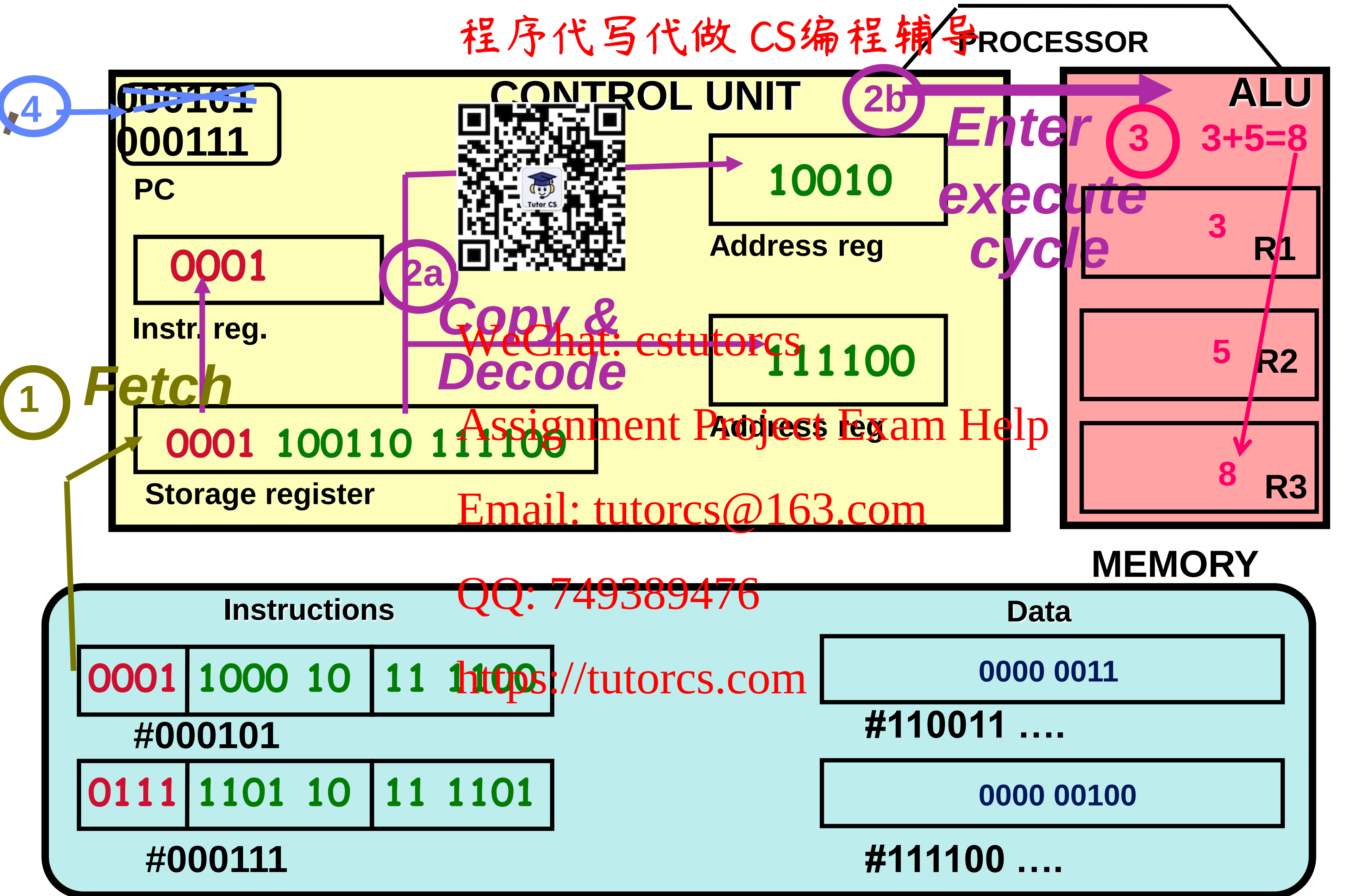
A simplified example



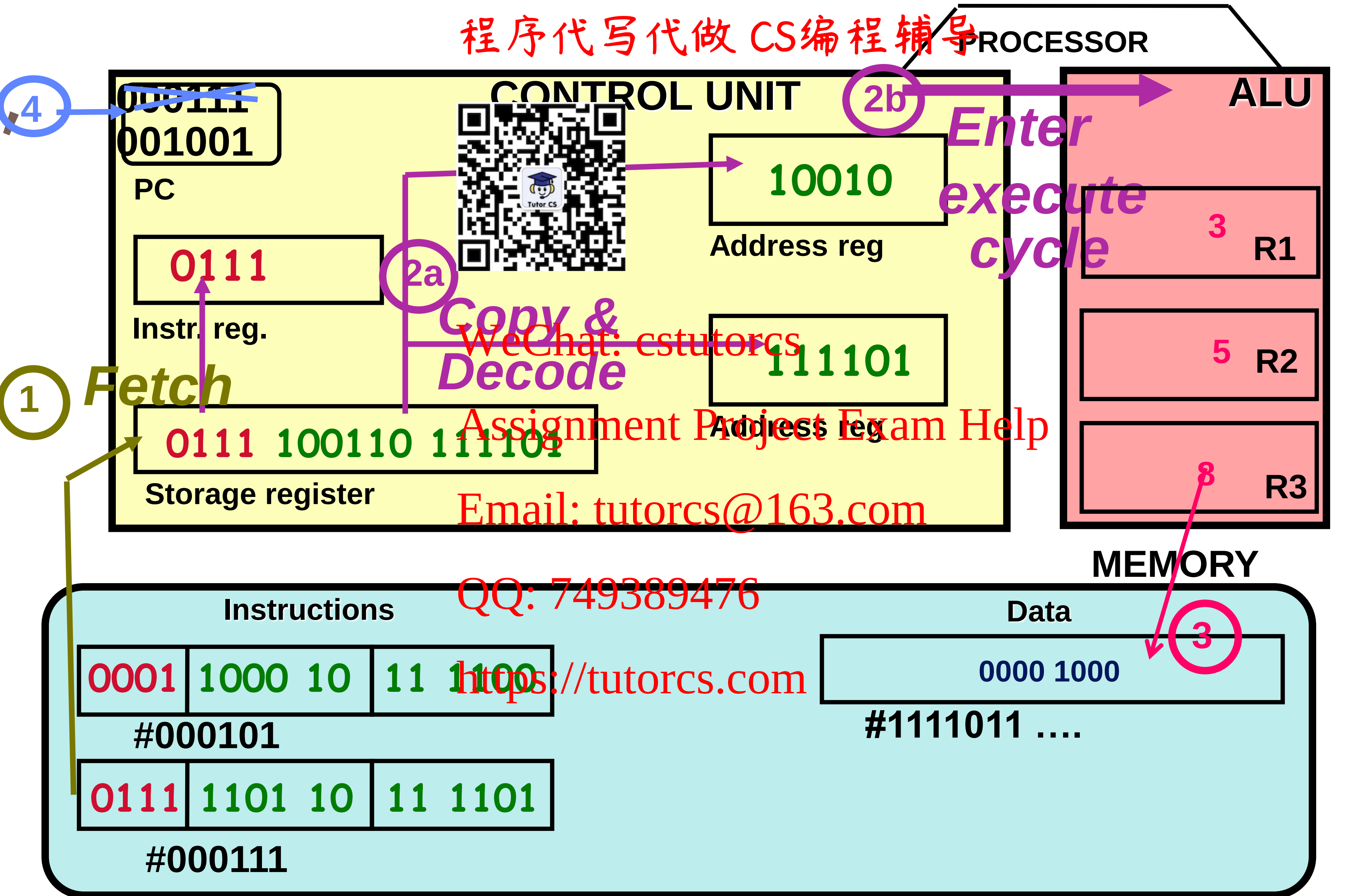
A simplified example



A simplified example



A simplified example

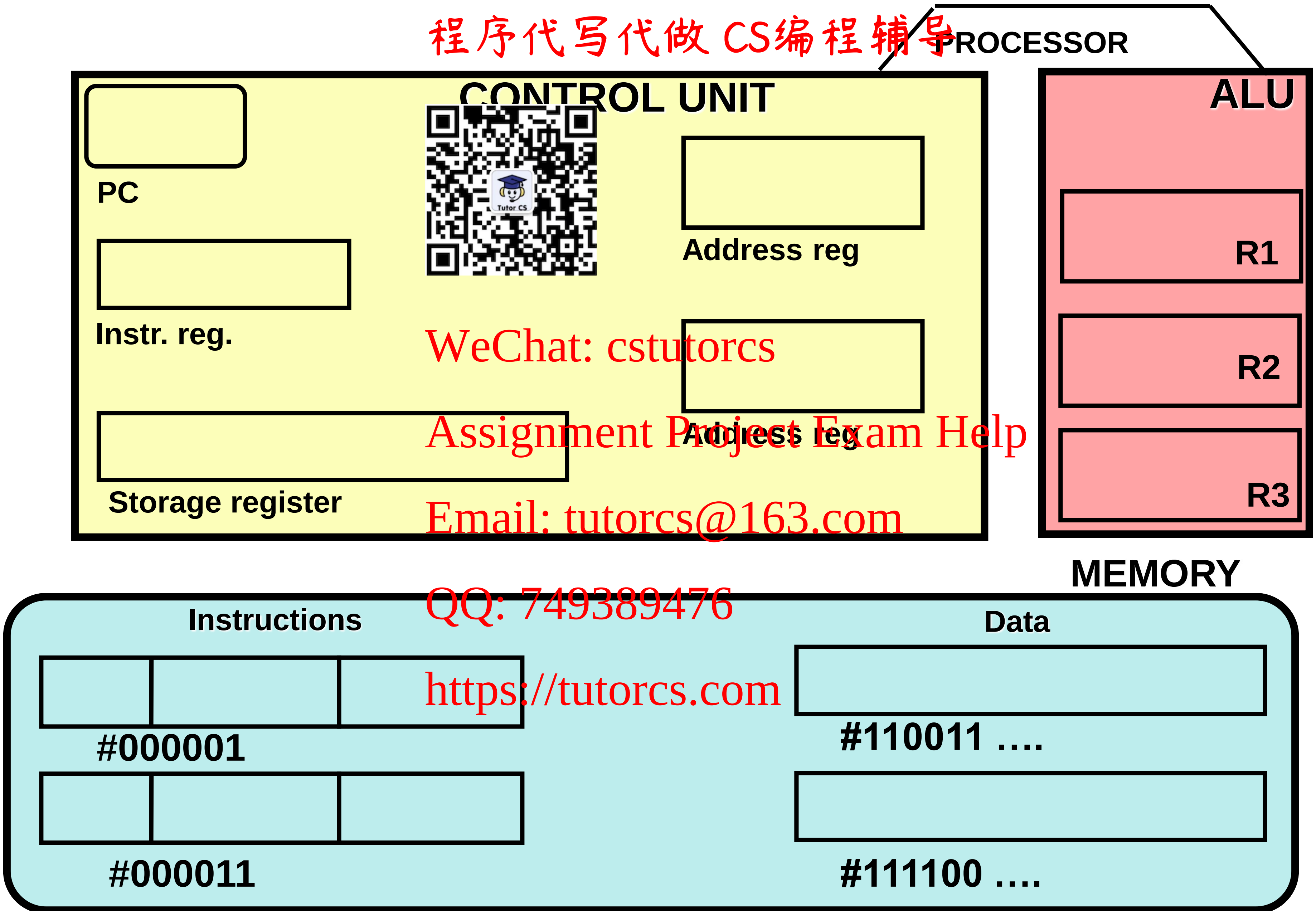


Think Pair Share

Given the following high e, first translate it into assembly pseudo code and second exp PU steps as before. Consider that a=0, x=2, y=3, z=4.

a = x + y – z;

Think Pair Share



Memory Types (1)

- Random Access Memo[illegible] Alternatively **referred to as main memory**
  - Static RAM (SRAM)
  - Dynamic RAM (DRAM)

- Read Only Memory (ROM)

  - Programmable read-only memory (PROM)

  - Erasable programmable read-only memory (EPROM)

  - Electrically erasable programmable read-only memory (EEPROM)

- **RAM loses any information it is holding when the power is turned off**
- **ROM is meant for permanent storage, while RAM is for temporary storage**

Memory Types (2)

- **ROM is used primaril[illegible]art up process of a computer, whereas a RAM chip is used i[illegible]al operations of a computer once the operating system has[illegible]ded**
- A good example of ROM is the computer BIOS, a PROM chip that stores the programming needed to begin the initial computer start up process
- Writing data to a ROM chip is a much slower process than writing it to a RAM chip
- A RAM chip can store multiple gigabytes (GB) of data, ranging from 1 GB to 256 GB per chip. A ROM chip stores several megabytes (MB) of data, typically 4 MB or 8 MB per chip

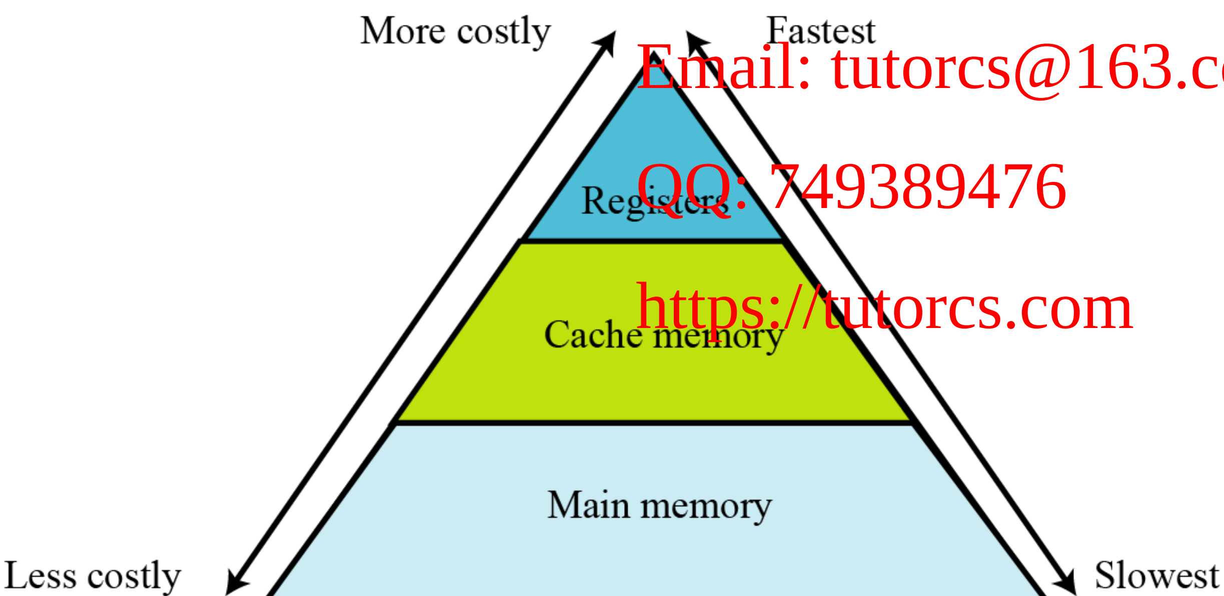
Cache Memory

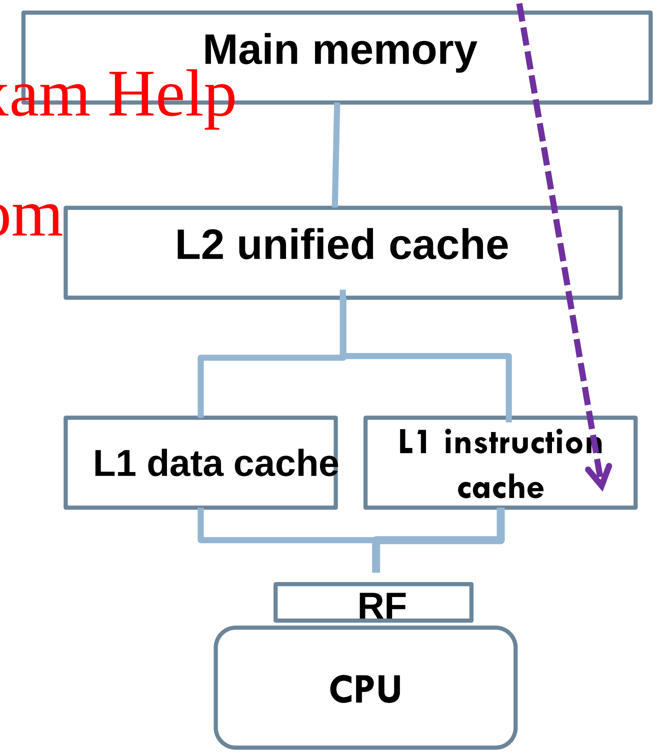
- Cache is a high-speed st[illegible]n access memory (SRAM) that a CPU can access more quickly thar[illegible]ess regular random access memory (RAM)
- This memory is typically [illegible] directly into the CPU chip
- Cache memory is faster than main memory, but slower than the CPU and its registers
- Cache memory, which is normally small in size, is placed between the CPU and main memory
- **The purpose of cache memory is to store program instructions and data that are used repeatedly -** The computer processor can access this information quickly from the cache rather than having to get it from computer's main memory
- Fast access to these instructions increases the overall speed of the program

Memory Hierarchy

- users want unlimited fast memory
- fast memory expensive, slow memory cheap
- cache: small, fast memory near CPU
- large, slow memory (main memory, disk, …)
- connected to faster memory (one level up)

```
for (i=0; i< 1000; i=i+1){
        S1
        S2
        …
        S10
}
```



Secondary memory (1)

- **Secondary memory is where programs and data are kept on a long-term basis**
  - Common secondary storage devices are the hard disk and optical disks

- The hard disk has enormous storage capacity compared to main memory

- **The hard disk is used for long-term storage of programs and data**

- Data and programs on the hard disk are organized into files
  - A file is a collection of data on the disk that has a name

Secondary memory (1)

- Running programs ar[illegible]located in main memory
- When creating a new [illegible]ype something it is stored into main memory; When you "save" your document, the characters are copied to a file on the hard disk

- A permanent copy will also be in secondary memory on the hard disk

Main Memory	Hard Disc
Fast	Slow
Expensive	Cheap
Low Capacity	High Capacity

Question: Do you think that data transfer from the network is slower or faster than from main memory?

Answer: Data transfers from the network are much slower than from main memory

Questions?