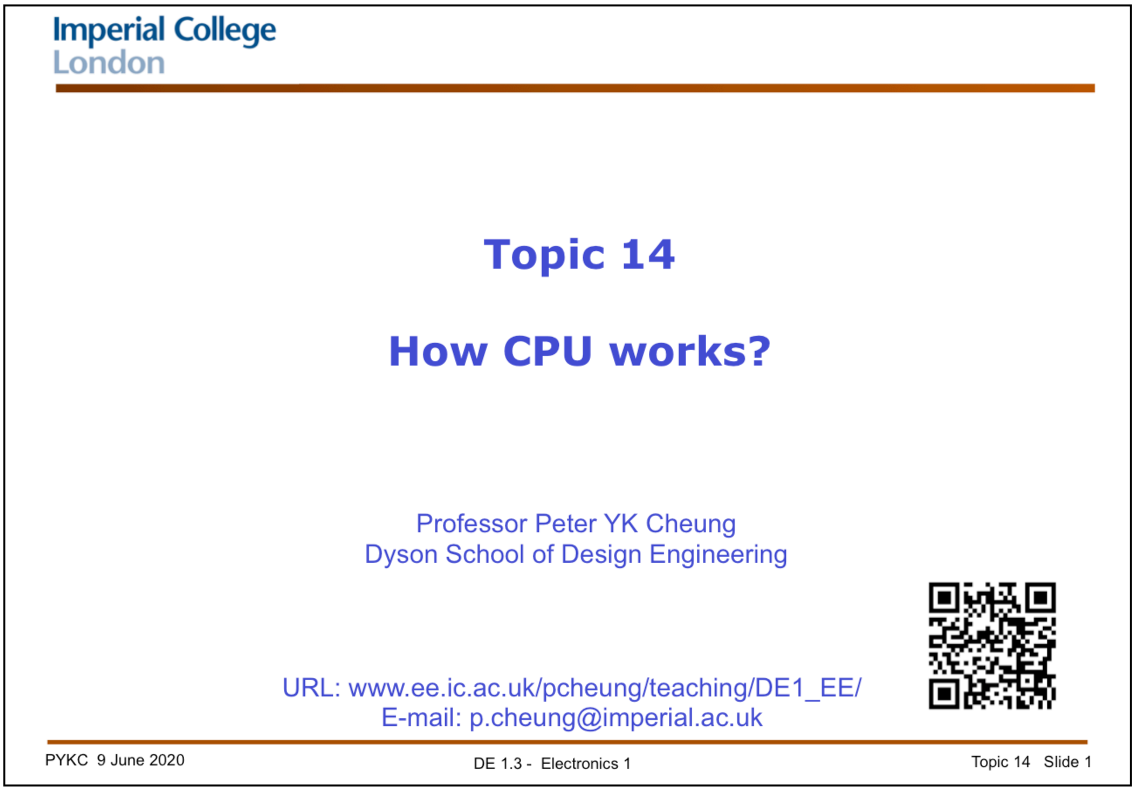


Imperial College London

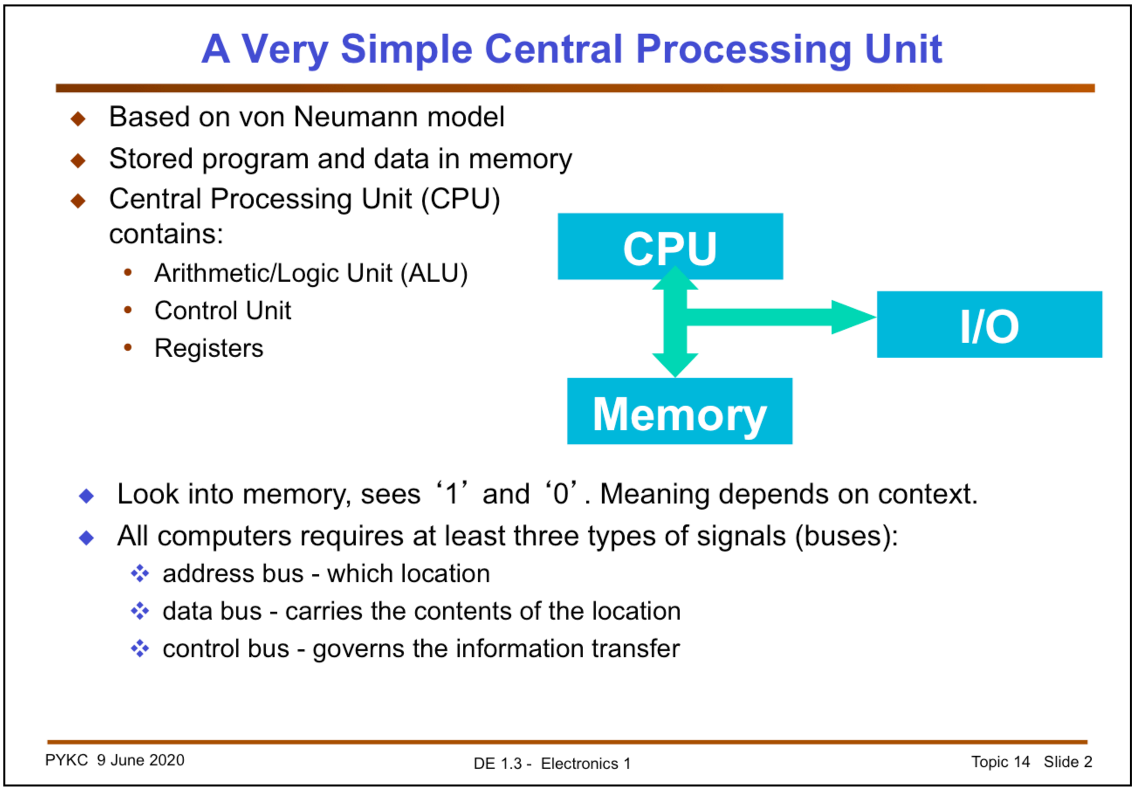
Topic 14

How CPU works?

Professor Peter YK Cheung
Dyson School of Design Engineering

URL: www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/
E-mail: p.cheung@imperial.ac.uk

In this lecture, we will examine how the CPU or microprocessor actually execute programs – the internal working of a computer.



A Very Simple Central Processing Unit

- Based on von Neumann model
- Stored program and data in memory
- Central Processing Unit (CPU) contains:
  - Arithmetic/Logic Unit (ALU)
  - Control Unit
  - Registers

CPU
I/O
Memory

- Look into memory, sees ‘1’ and ‘0’. Meaning depends on context.
- All computers requires at least three types of signals (buses):
  - address bus - which location
  - data bus - carries the contents of the location
  - control bus - governs the information transfer

In the last lecture we have looked at the overall structure of a computer, which has three main components: CPU, memory and IO, all connected together via three buses: address bus, data bus and control bus.

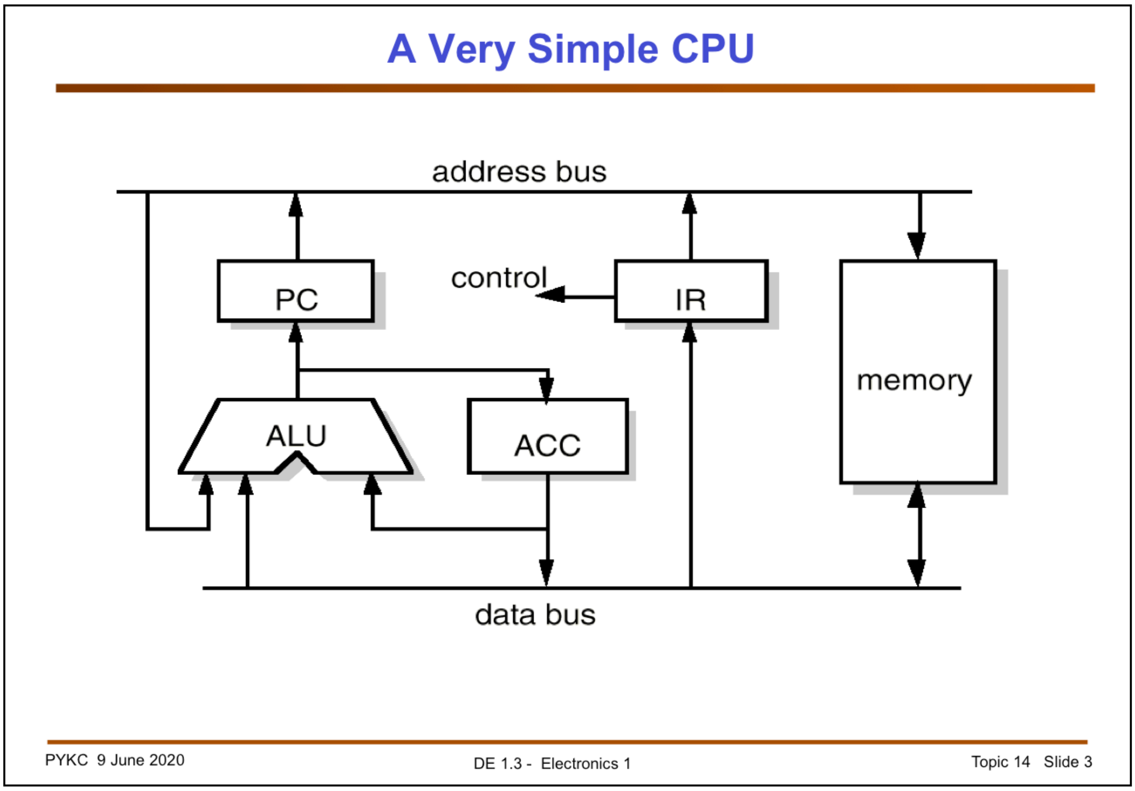
In this lecture we will look at the CPU in some detail.

A CPU has a number of modules inside. These include a arithmetic and logic unit, a control unit and at least one, but usually many more, registers.

Remember registers are just a bunch of D-FFs.

Modern computers and microprocessors are based on the von Neumann model which uses memory to store both the instruction codes (i.e. the program) and the data.

It is important to remember and understand that if you could open the top of a memory chip and read the values stored inside the chip, you will see ‘1’s and ‘0’s. What they mean depends on where they are stored and the context. You cannot tell by just looking at memory which word is an instruction word, and which word is data. For that, we need some other information.



A Very Simple CPU

PYKC 9 June 2020 DE 1.3 - Electronics 1 Topic 14 Slide 3

If you look inside a simple CPU, you are going to find these modules:

PC = **P**rogram **C**ounter – it stores the **address** of the NEXT instruction to be executed

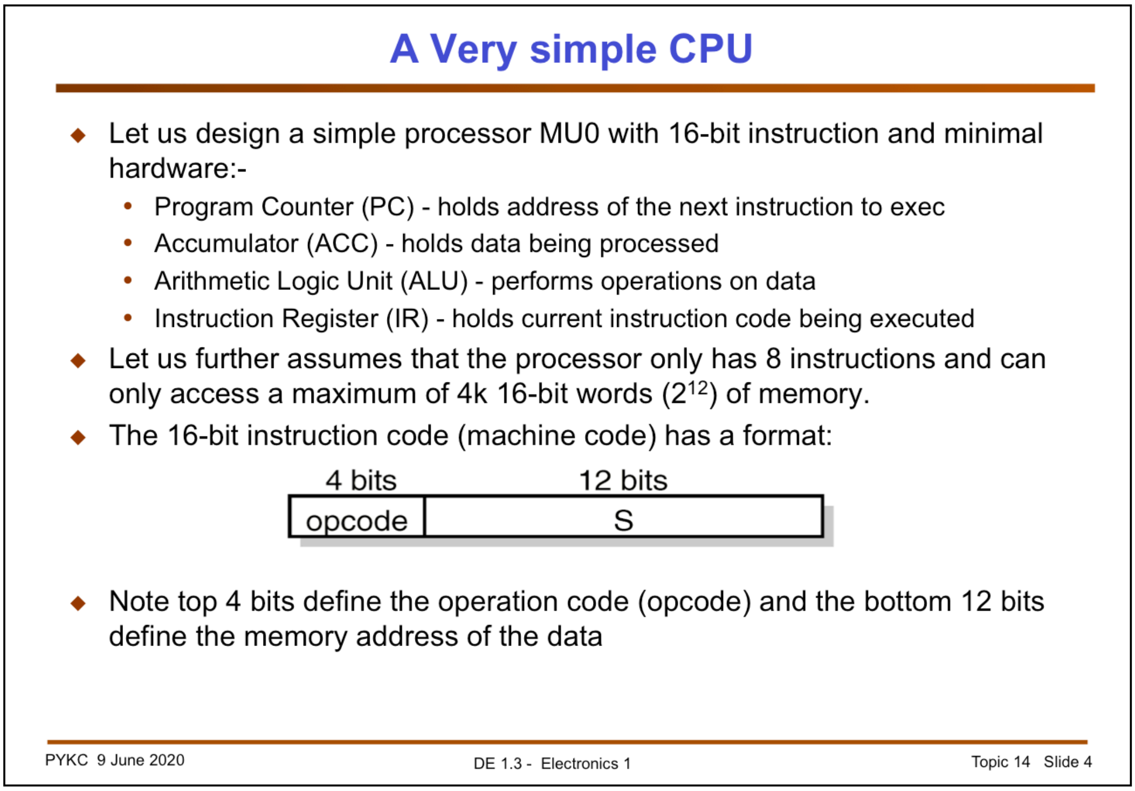
IR = **I**nstruction **R**egister – It stores the current instruction binary code (called machine code) to be executed

ALU = **A**rithmetic/**L**ogic **U**nit – It performs the actual arithmetic or logical operations

ACC = **Acc**umulator (or result register) – It stores the temporary data or result

In modern CPUs, there are many other modules. For example, instead of having only one ACC, there could be many more temporary registers. For example, the ARM processor in the Pyboard has 16 registers (including one for the PC). In addition to the ALU, there is also a floating point unit (FPU) and even an additional computational engine known as Adaptive Real-Time Accelerator (ART).

In order for you to understand how a CPU does its job, let us now create one that is very simple and only having the few modules shown above.



A Very simple CPU

- Let us design a simple processor MU0 with 16-bit instruction and minimal hardware:-
  - Program Counter (PC) - holds address of the next instruction to exec
  - Accumulator (ACC) - holds data being processed
  - Arithmetic Logic Unit (ALU) - performs operations on data
  - Instruction Register (IR) - holds current instruction code being executed
- Let us further assumes that the processor only has 8 instructions and can only access a maximum of 4k 16-bit words (2^{12}) of memory.
- The 16-bit instruction code (machine code) has a format:

4 bits	12 bits
opcode	S

- Note top 4 bits define the operation code (opcode) and the bottom 12 bits define the memory address of the data

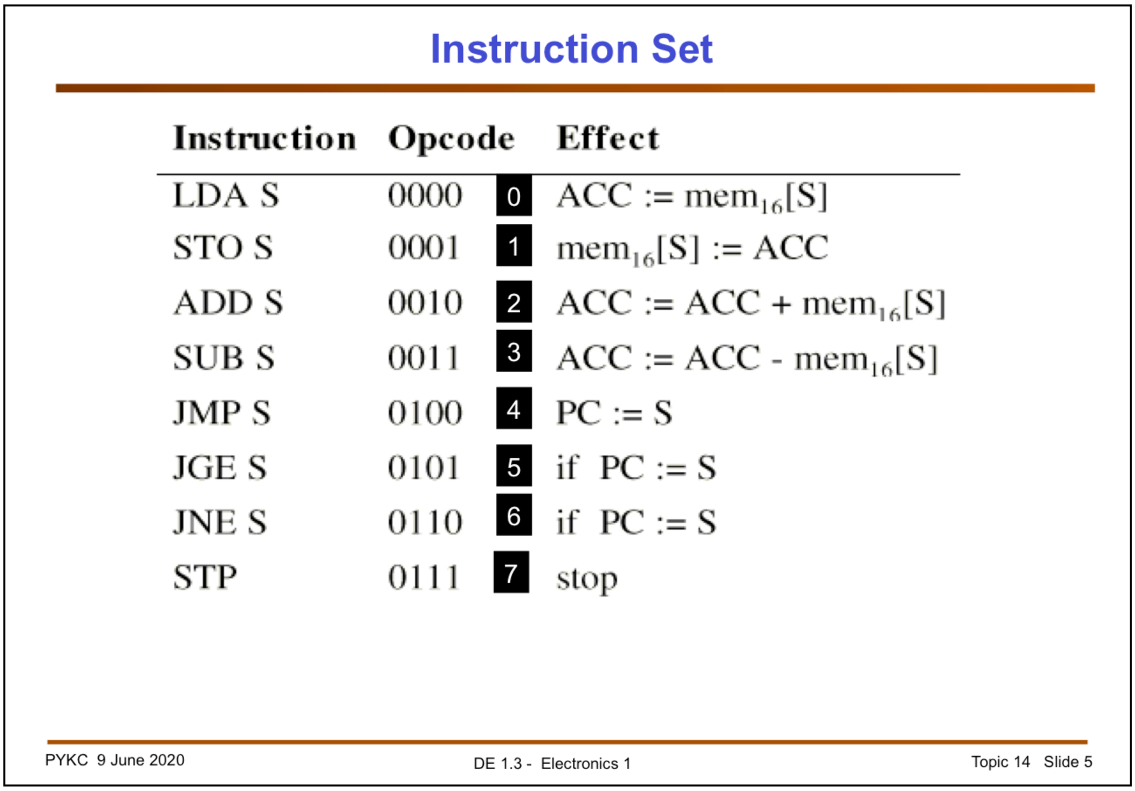
This simple CPU uses instruction code that has 16-bits.

We assume that there is only 8k byte of memory and the processor is a 16-bit processor meaning that it process data in 16-bit words. (ESP32 normally process data in 32-bits.) Therefore the processor memory is organised as 4k x 16 bits. 4k words require 12 address bits.

Shown here is the format of the instruction. The top 4 bits are used to specify **the operation** to be performed. This is known as the **opcode**.

With 4-bit opcode, we could specify 16 operations, but we will only create 8 operations with opcode going from 0 to 7.

The remaining lower significant 12 bits in the instruction word are used to specify the **memory address** for the operation if required. We will see what that means in the next few slides.



Instruction Set

Instruction	Opcode		Effect
LDA S	0000	0	ACC := mem$_{16}$[S]
STO S	0001	1	mem$_{16}$[S] := ACC
ADD S	0010	2	ACC := ACC + mem$_{16}$[S]
SUB S	0011	3	ACC := ACC - mem$_{16}$[S]
JMP S	0100	4	PC := S
JGE S	0101	5	if PC := S
JNE S	0110	6	if PC := S
STP	0111	7	stop

Here is our simple instruction set with only 8 instructions. Here are the explanations for all 8 instructions. These instructions are shown as three letter code (known as mnemonics) to indicate what each instruction is suppose to do. This is known as **Assembly Language** program. However, the CPU can only understand binary code. Therefore shown here are the opcodes for each instructions forming the 4 most-significant bits (MSBs). Then each instruction is followed by a 12-bit memory address S. The binary form of the instruction is known as **Machine Code** program.

LDA S - Load Accumulator with the content at memory location S

STO S - Store Accumulator value to memory location S

ADD S - Add the data in memory location S to ACC and store the result back to ACC

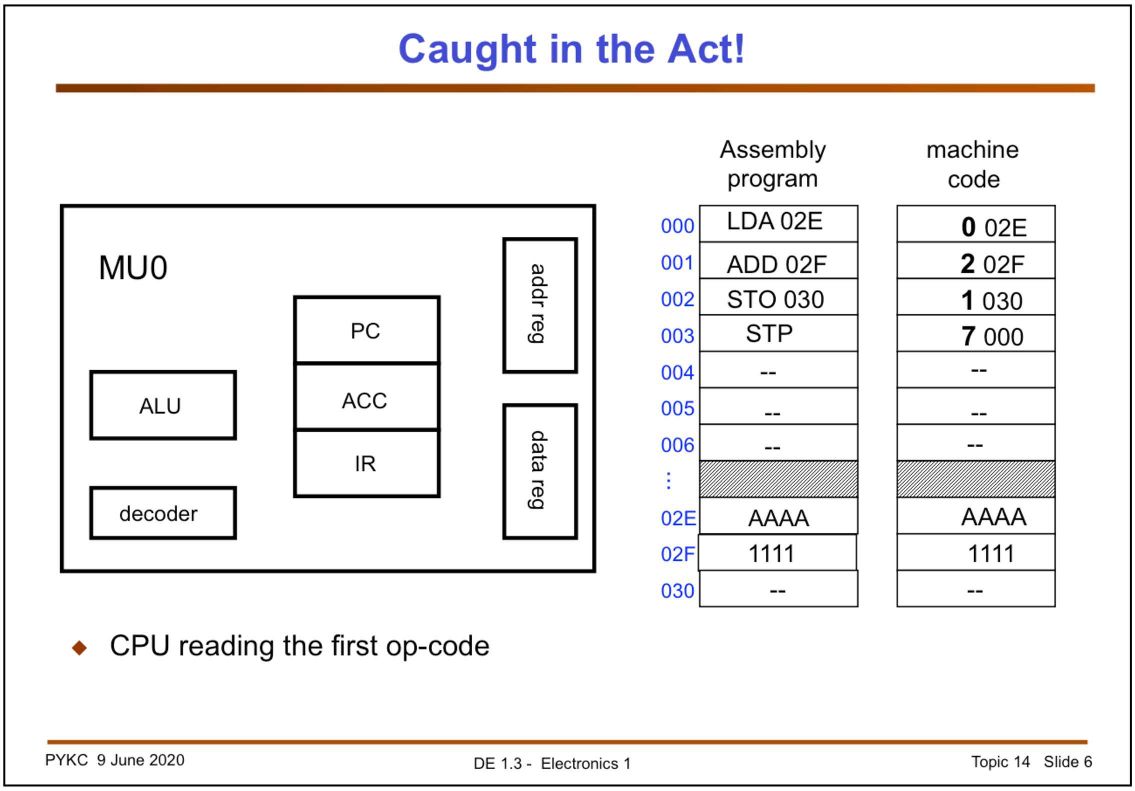
SUB S - Subtract the data in memory location S from ACC and store the result back to ACC

JMP S - Jump to instruction at memory address S

JGE S - Jump to instruction at memory address S if the previous ALU operation is greater than or equal to zero

JNE S - Same as before but for the case where the previous ALU operation does not equal to zero

STP - Stop the processor. Since this instruction does not require any memory address, it only uses the top 4-bits of the instruction word



Caught in the Act!

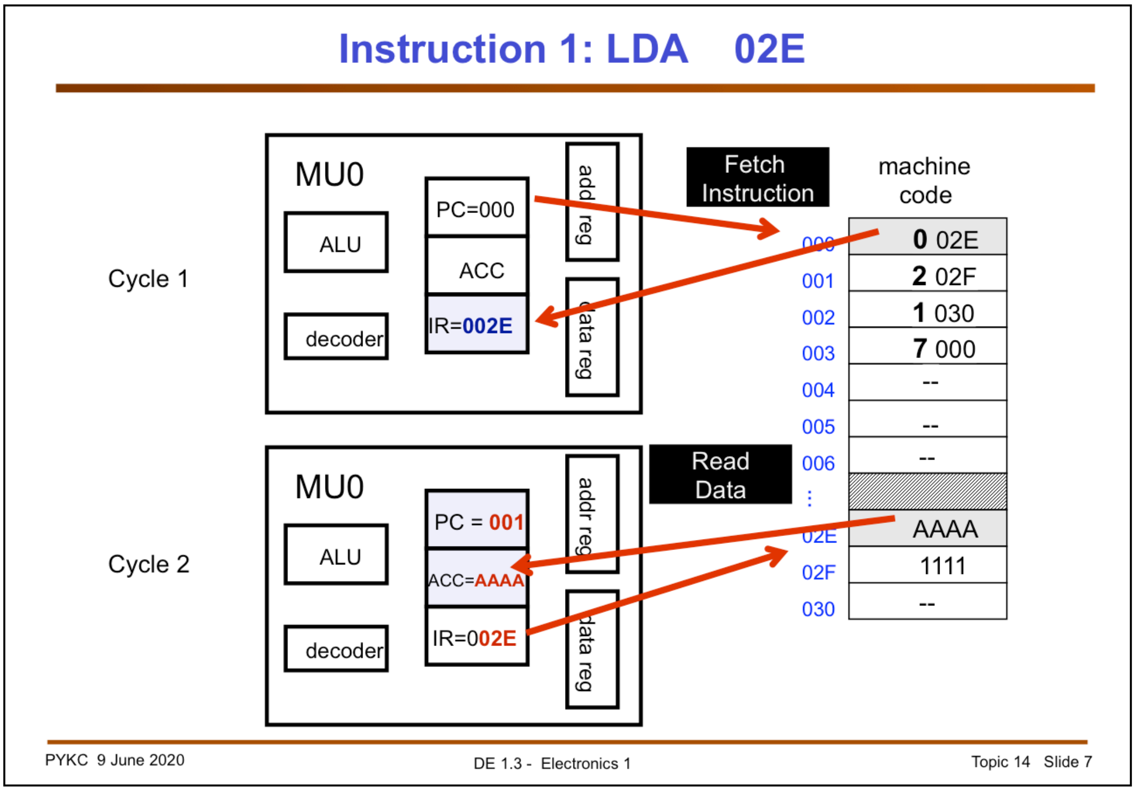
	Assembly program	machine code
000	LDA 02E	**0** 02E
001	ADD 02F	**2** 02F
002	STO 030	**1** 030
003	STP	**7** 000
004	--	--
005	--	--
006	--	--
⋮	▨	▨
02E	AAAA	AAAA
02F	1111	1111
030	--	--

- CPU reading the first op-code

PYKC 9 June 2020 DE 1.3 - Electronics 1 Topic 14 Slide 6

Now let us assume that at memory location 000 (hex) to 003, there are already stored four instructions. Furthermore, we also assume that in memory locations 02E and 02F (in hex again) stores two numbers AAAA and 1111 to be added together. The result of this addition, which is BBBB, will be stored back to memory location 030.

Let us now see how these instructions are executed by the CPU, one instruction at a time.



Instruction 1: LDA 02E

	machine code
000	**0** 02E
001	**2** 02F
002	**1** 030
003	**7** 000
004	--
005	--
006	--
⋮	▨
02E	AAAA
02F	1111
030	--

PYKC 9 June 2020 DE 1.3 - Electronics 1 Topic 14 Slide 7

Firstly, we assume that we start from PC = 000, i.e. on reset, the program counter is zeroed.

This means that the PC is pointing to address 000 in memory where the first instruction code is stored. It is 002E, which is LDA 02E instruction, meaning that it should read the 16-bit data from memory location 02E and stores this in the accumulator.

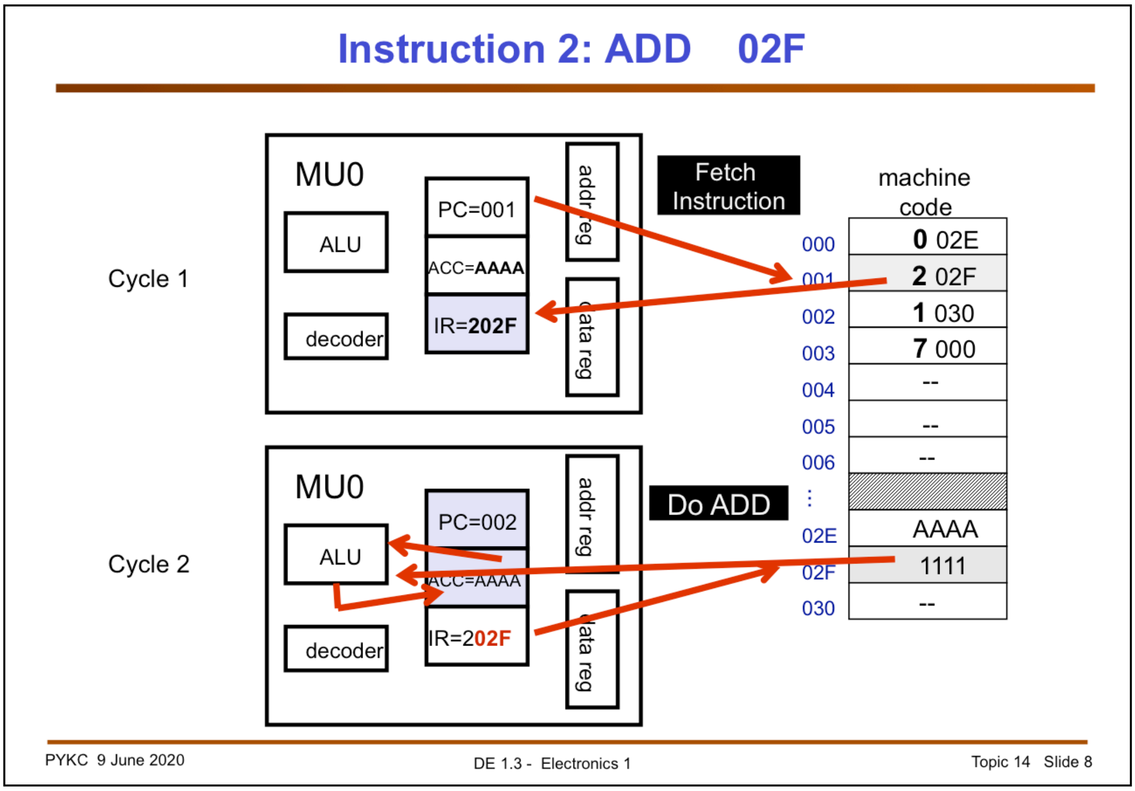
So in the first clock cycle, the PC is pushed out via the address register onto the address bus. A memory read operation is performed and the instruction code 002E is fetched and put into the instruction register IR. This is the instruction fetch cycle.

The opcode part (top 4-bits) of the instruction is passed through the instruction decoder unit and the CPU is ready to “execute” the opcode by performing a data-read-from-memory operation.

This operation is done in the second clock cycle.

The bottom 12-bit of the instruction stored in IR, 02E, is now pushed to the address register then onto the address bus. The data stored at location 02E (which is AAAA) is read from this memory location and stored in ACC.

Now something else ALWAYS happens. After the PC is used to fetch an instruction, its value is ALWAYS automatically incremented. In other words, the PC always counts up once it is used (hence the name program counter). This is because we want it to always point to the **next** instruction.



Instruction 2: ADD 02F

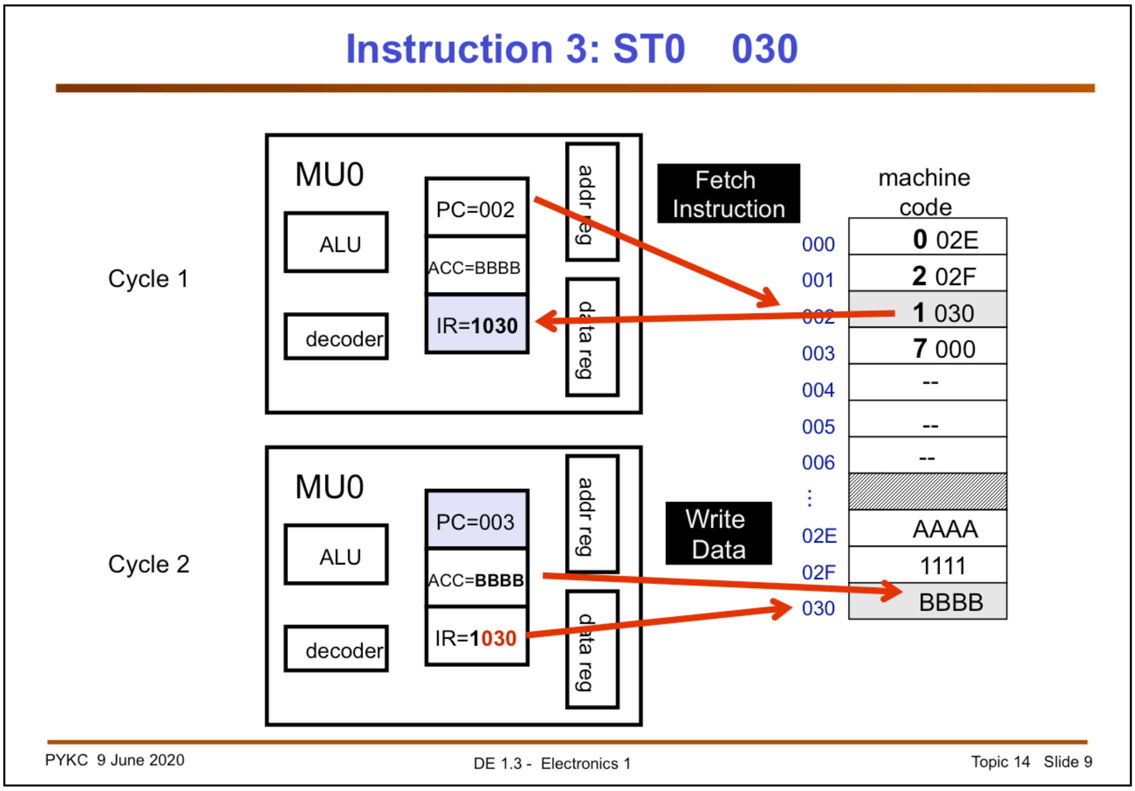
	machine code
000	**0** 02E
001	**2** 02F
002	**1** 030
003	**7** 000
004	--
005	--
006	--
⋮	▨
02E	AAAA
02F	1111
030	--

PYKC 9 June 2020 DE 1.3 - Electronics 1 Topic 14 Slide 8

The second instruction is to add the previously fetch data to the data stored in memory location 02F.

Again in the first clock cycle, the PC value is send to the address bus, and the instruction code 202F is fetched and stored in IR as shown.

On the next cycle, the opcode (2) is decoded by the CPU and force an add operation to be performed. This involves the CPU sending the lower 12-bit of the instruction code (02F) to the address bus and performs a memory read. The data from memory 02F (i.e. 1111) is send, together with the value stored in ACC, to the ALU and the opcode tells the ALU to do an addition. The result BBBB is stored back to ACC.



Instruction 3: ST0 030

	machine code
000	**0** 02E
001	**2** 02F
002	**1** 030
003	**7** 000
004	--
005	--
006	--
⋮	▨
02E	AAAA
02F	1111
030	BBBB

PYKC 9 June 2020 DE 1.3 - Electronics 1 Topic 14 Slide 9

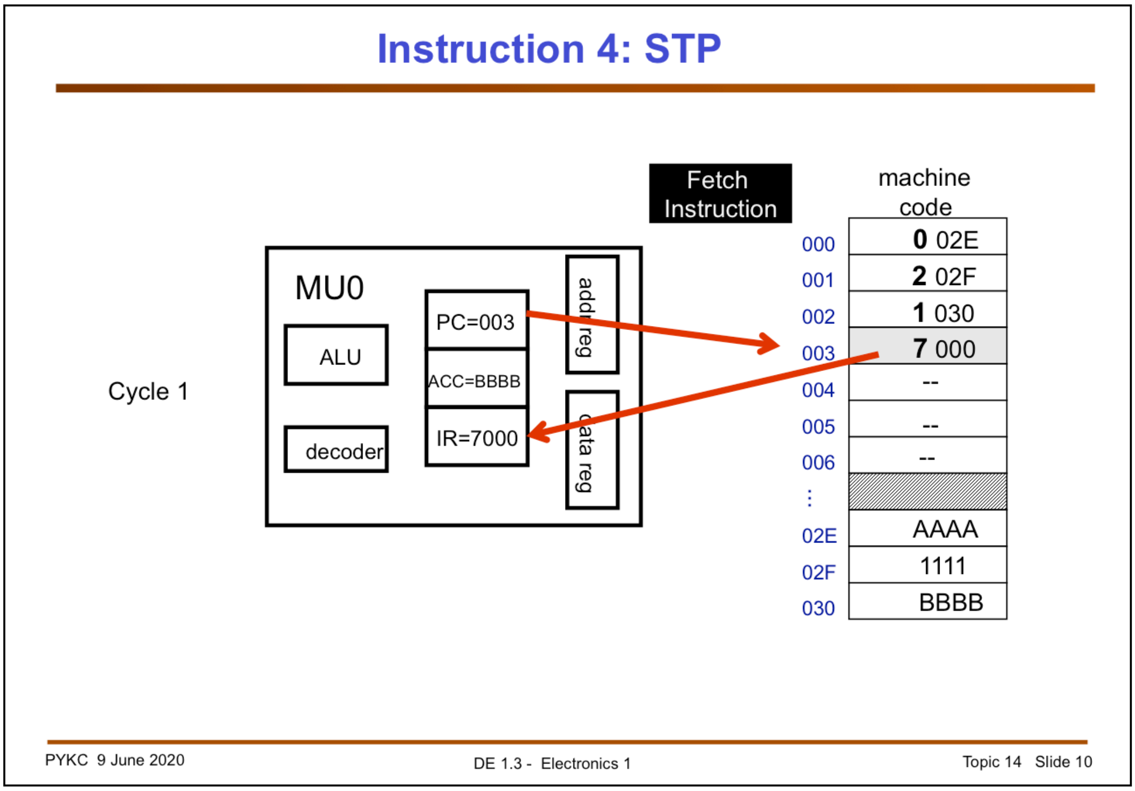
In this instruction, the contents of ACC is stored to memory location 030.

Finally, the last instruction is to tell the CPU to stop. This instruction does not involve any memory read AFTER fetching the instruction. It therefore only use the top four bit of the instruction for opcode, and it only takes one clock cycle. This is shown on the next slide.

To summarise

The operation of most processors are governed by a clock signal. For this simple CPU, we assume that:

1. The number of clock cycles taken by an instruction is the same as the number of memory access it makes.
2. LDA, STO, ADD, SUB therefore takes 2 clock cycles each: one to fetch (and decode) the instruction, a second to fetch (and operate on) the data
3. JMP, JGE, JNE, STP only need one memory read and therefore can be executed in one clock cycle.
4. Program counter (PC) - its content is incremented every time it is used (i.e. it also points to the next instruction).
5. The processor must start from a known state. Therefore, there is always a reset signal to initialise the processor on power-up.
6. Assume MU0 will always reset to start execution from address 000_{16}.

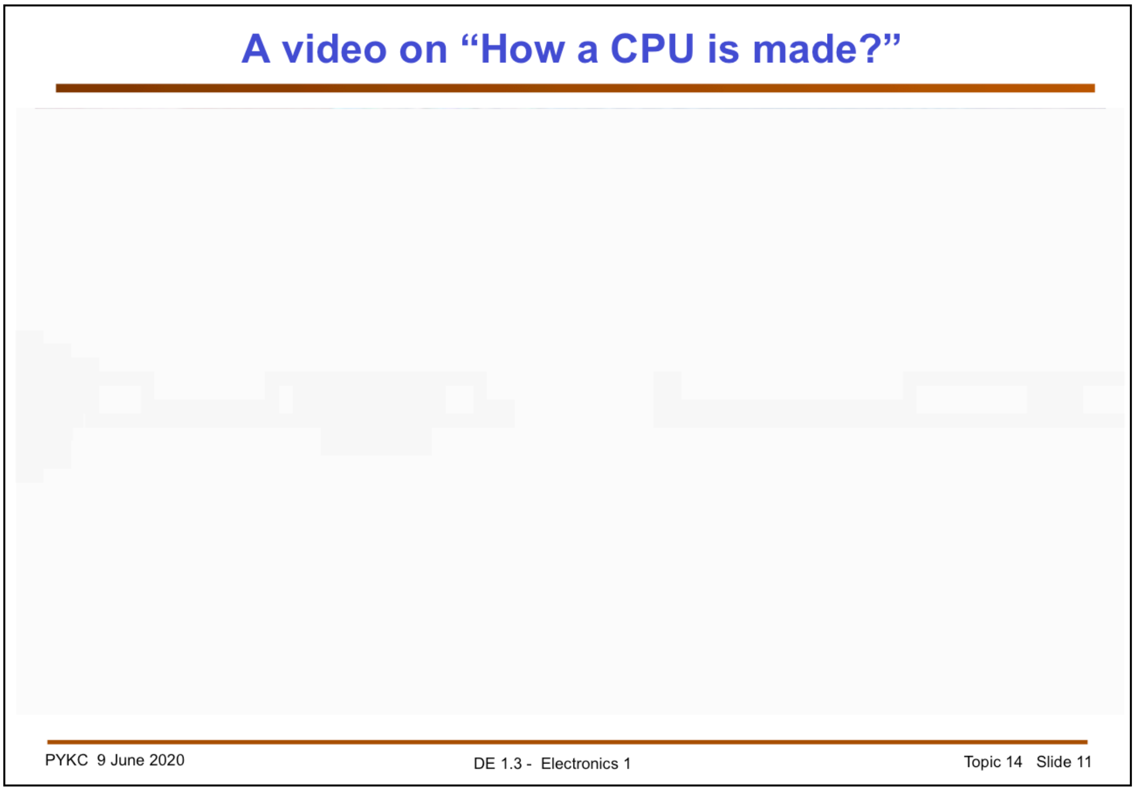


Instruction 4: STP

	machine code
000	**0** 02E
001	**2** 02F
002	**1** 030
003	**7** 000
004	--
005	--
006	--
⋮	▨
02E	AAAA
02F	1111
030	BBBB

PYKC 9 June 2020 DE 1.3 - Electronics 1 Topic 14 Slide 10

8. Microprocessors performs operations depending on instruction codes stored in memory
9. Instruction usually has two parts:
   - Opcode - determines what is to be done
   - Operand - specifies where/what is the data
10. Memory contains both program and data. A peek into memory will tell you very little except a bunch of ‘1’s and ‘0’s
11. Program area and data area in memory are usually well separated
12. ALU is responsible for arithmetic and logic operations
13. There is always at least one register known as **accumulator** where the result from ALU is stored
14. There is usually one or more general purpose register for storing results or memory addresses temporarily
15. Fetching data from inside the CPU is much faster than from external memory



A video on “How a CPU is made?”

This video explains how a CPU or a microprocessor is made. You can find this video on youtube:

https://www.youtube.com/watch?v=qm67wbB5GmI