technocamps

UNDEB EWROPEAIDD
EUROPEAN UNION

Llywodraeth Cymru
Welsh Government

Cronfa Gymdeithasol Ewrop
European Social Fund

Prifysgol Abertawe
Swansea University

CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

PRIFYSGOL BANGOR UNIVERSITY

Cardiff Metropolitan University
Prifysgol Metropolitan Caerdydd

i.t.wales

PRIFYSGOL ABERYSTWYTH UNIVERSITY

Prifysgol Glyndŵr Wrecsam | Wrexham Glyndŵr University

University of South Wales
Prifysgol De Cymru

technocamps

Computer Architecture & Assembly Language

technocamps

Surface Pro 5 vs. MacBook Pro 2017

You are going to watch the marketing videos for each of these laptops.

You need to decide which you would rather and why?

technocamps

Surface Pro 5

https://www.youtube.com/watch?v=sHp7f00JY8Q

MacBook Pro 2017

https://www.youtube.com/watch?v=1yVF-N__JKk

technocamps

Activity: Surface Pro 5 vs. MacBook Pro 2017

Marketing Nonsense

The videos were full of marketing buzzwords that sell items but don't really mean anything!

"This is the best speaker system ever implemented on a surface pro" doesn't mean much if the old surface pro didn't even have one.

To make informed decisions we should look at the tech specs of comparable systems. Then we should decide on what is best for the use we have in mind.

Components of Computer Systems

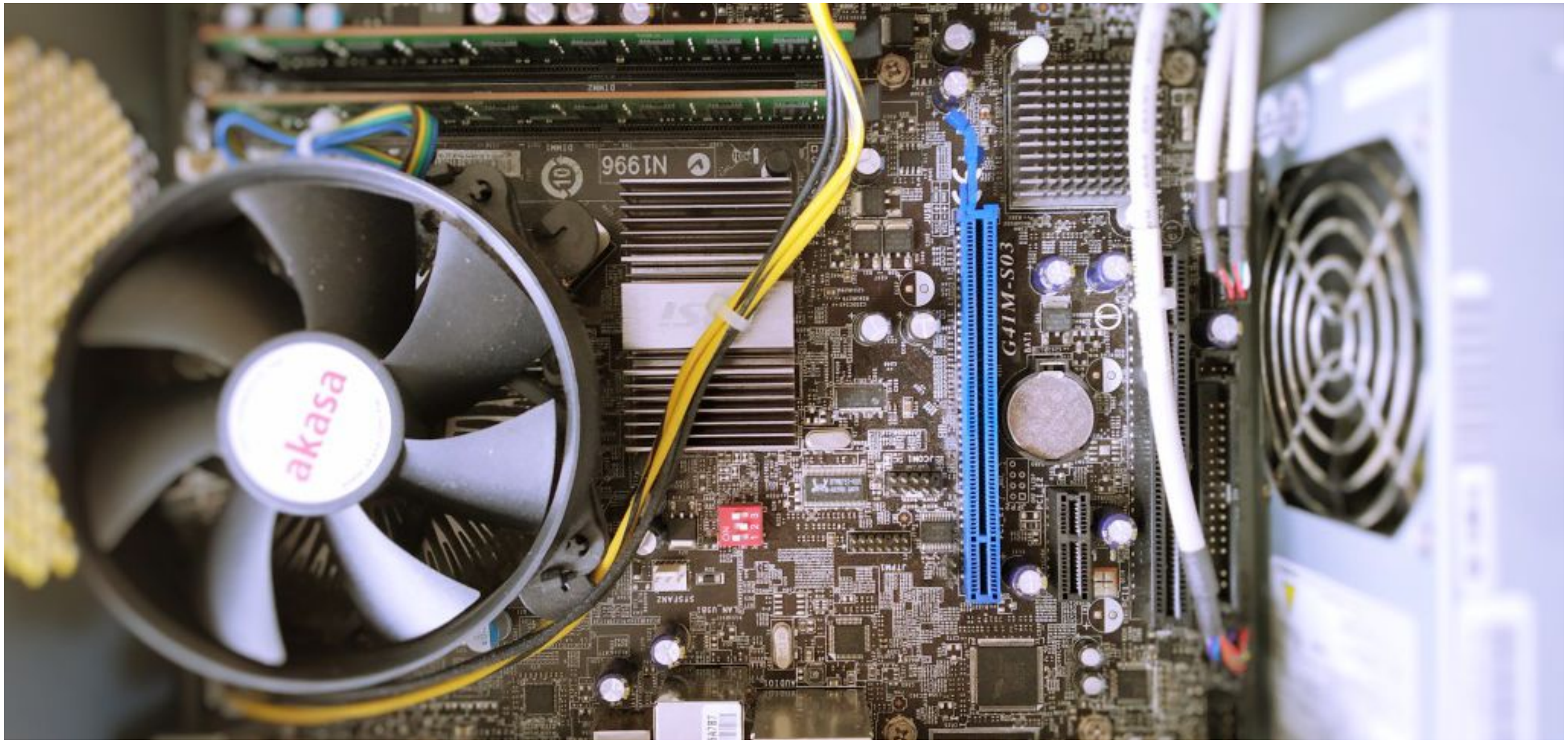
Internal Components

Let’s look at what’s inside a laptop or desktop.

We’ll look at what each component does and why they are important.

When we have worked out what is important and why, we can make informed decisions about which laptop, phone, or gaming console is better.

Computer Components



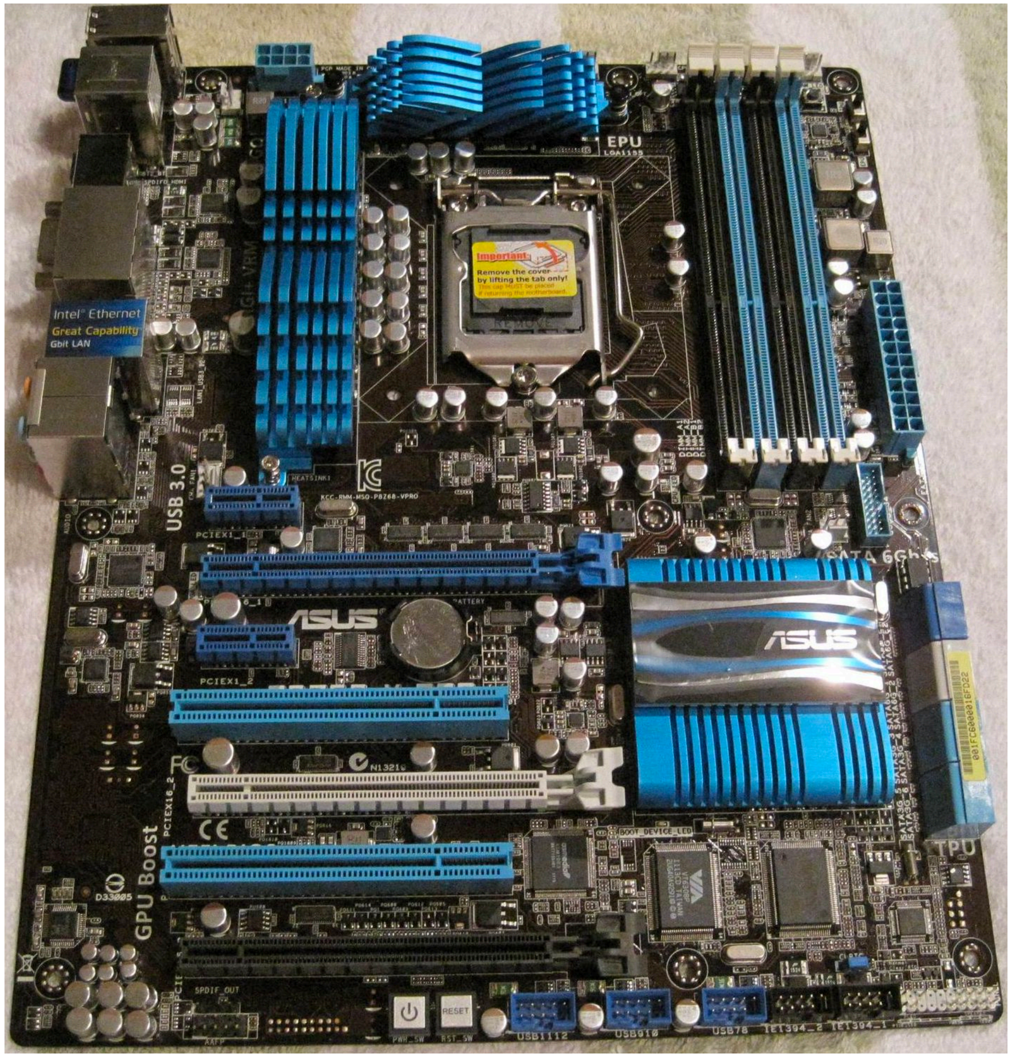
Motherboard

The motherboard is maybe the most important component within a computer.

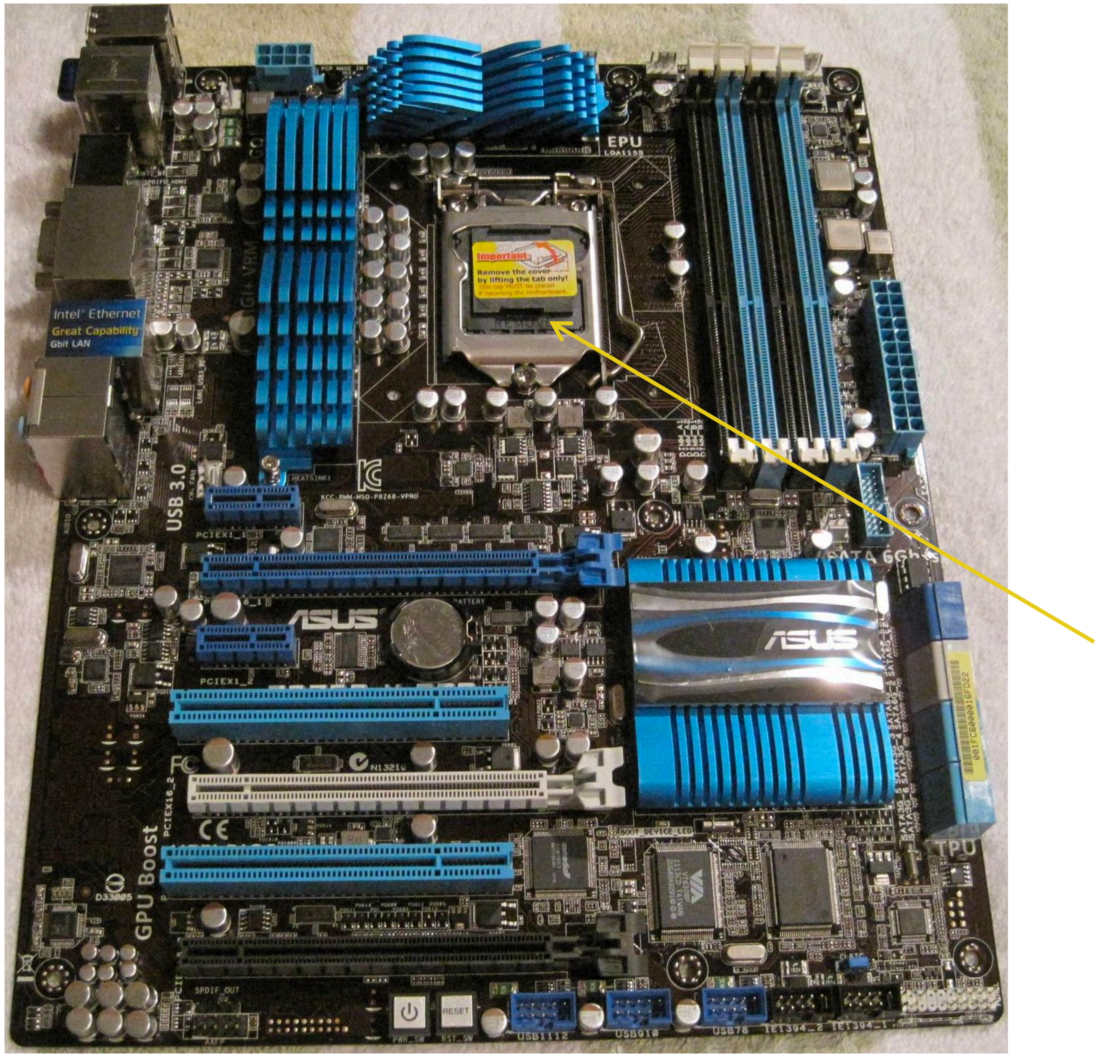
It is a PCB (printed circuit board) that connects all the components of a computer system together. It is also where most external devices (mouse, keyboard) connect to the computer.

technocamps

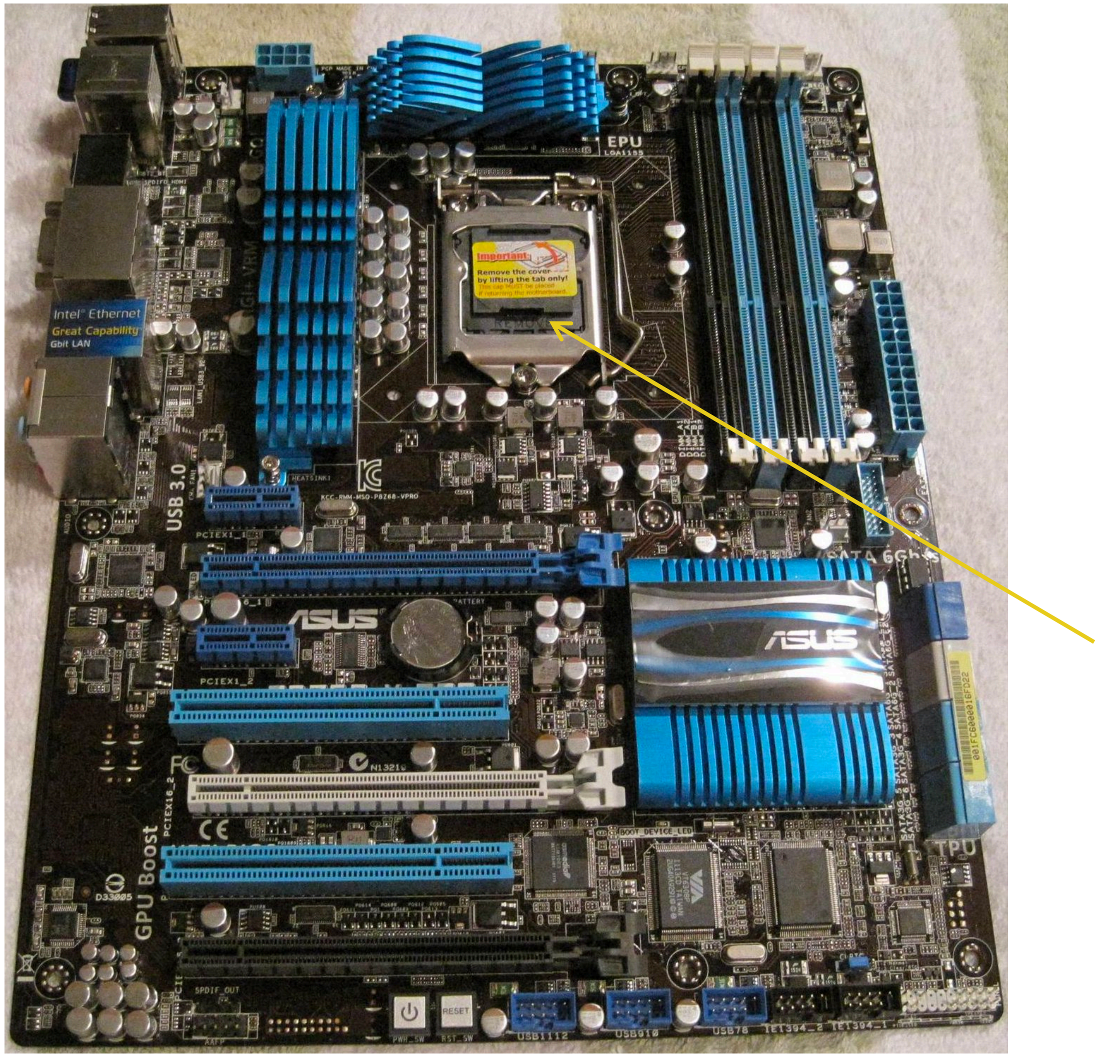
Motherboard



Computer Components



Computer Components



technocamps

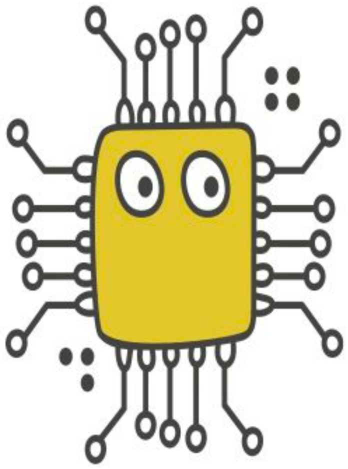
Extension Activity: Seating a CPU

CPU

The CPU (*Central Processing Unit)* is the '**brain**' of the computer.

On personal computers and small workstations, the CPU is housed in a single chip called a *microprocessor*.

It contains the circuitry which processes the instructions when running any computer program.



Checkout

Imagine we are at a supermarket at the checkout. We get to the checkout and Slowpoke Rodriguez, scans 1 item every 5 seconds. That's not very quick so we are sad.

But Rodriguez goes on break and in comes Speedy Gonzales who scans 1 items a second. That's rapid so we are happy.

Clock Speed

- The speed the cashier scans at is an analogy for **clock speed**.
- The clock speed of the CPU is measured in Hertz and refers to the number of clock cycles per second that the CPU runs at.
- A CPU with a speed of 5 Gigahertz (5 GHz) can process 5 billion instructions per second.
- if Speedy Gonzales was a CPU he would have a higher clock speed than Slowpoke Rodriguez and could execute more instructions per second.

technocamps

Checkout

We are back at the supermarket and there is a massive line of people in the queue.

But then a second checkout assistant comes along and opens a second checkout. This is great!

The queue of people splits between the two checkouts and this results in us getting our shopping checked out faster.

Core Count

The number of cores in a CPU is similar to the number of checkout stations in a supermarket. The more the merrier.

A core is the part of a CPU that receives instructions and performs calculations, or actions, based on those instructions.

CPUs can have a single core or multiple cores.

In theory, the more cores a processor has, the more sets of instructions the processor can receive and process at the same time, which makes the computer faster.

Parallelisation

technocamps

Activity: Installing a Fan

Cooling

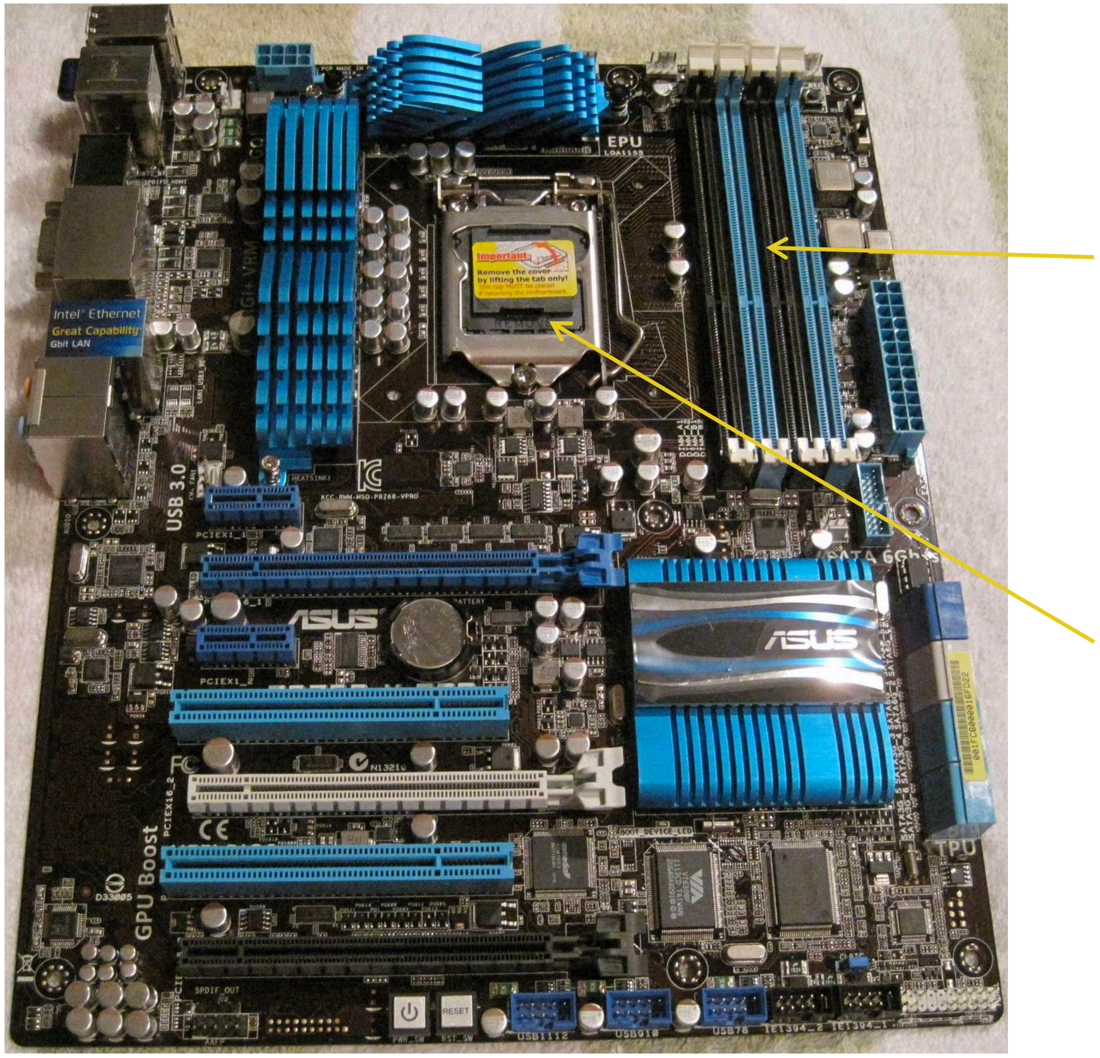
Like a car engine has a massive radiator to keep it cool, components of a computer also need to be kept cool so they can work effectively.

Graphics cards and CPUs generate a lot of heat. If they get too hot their performance gets worse and can damage the chips inside them.

As a result, we need to cool our system. This is normally done using fans but can be done by using liquid (water normally).

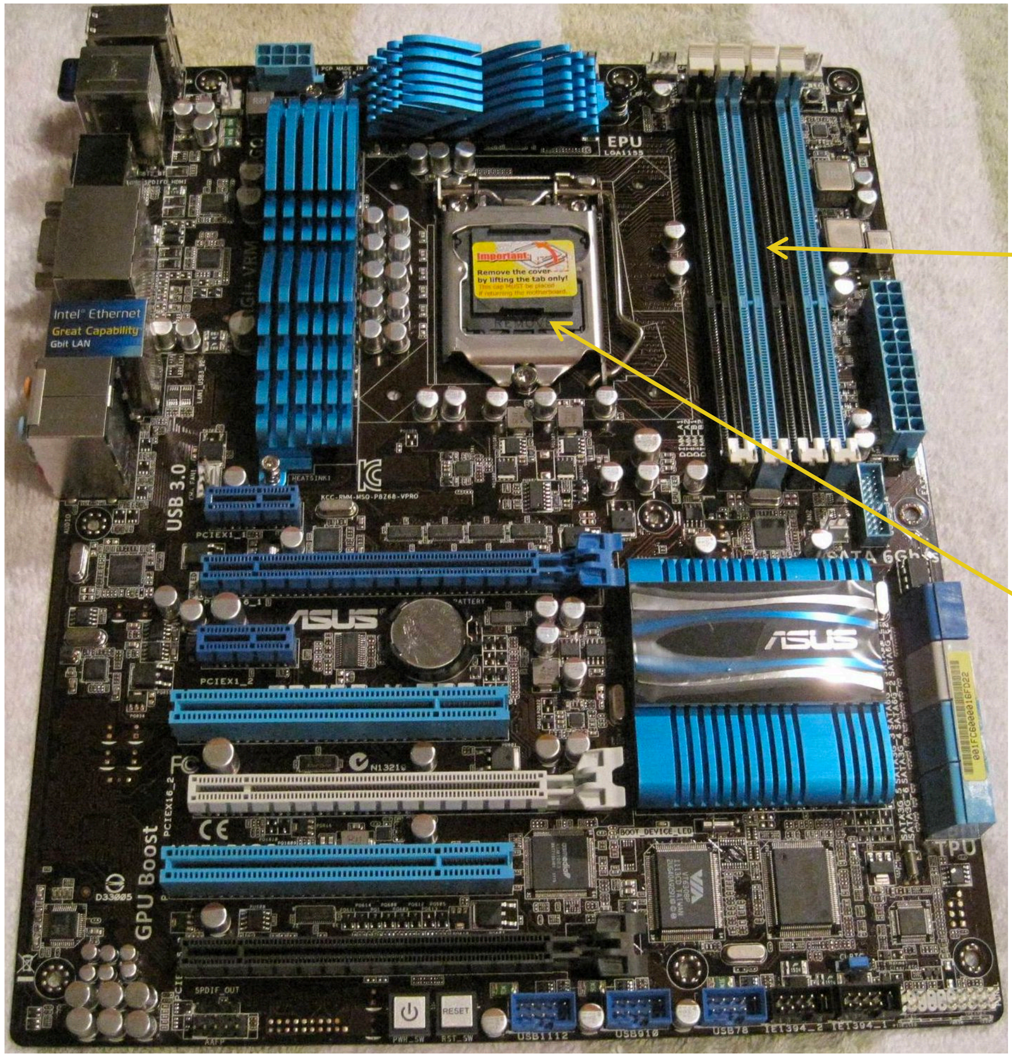
technocamps

Computer Components



technocamps

Computer Components



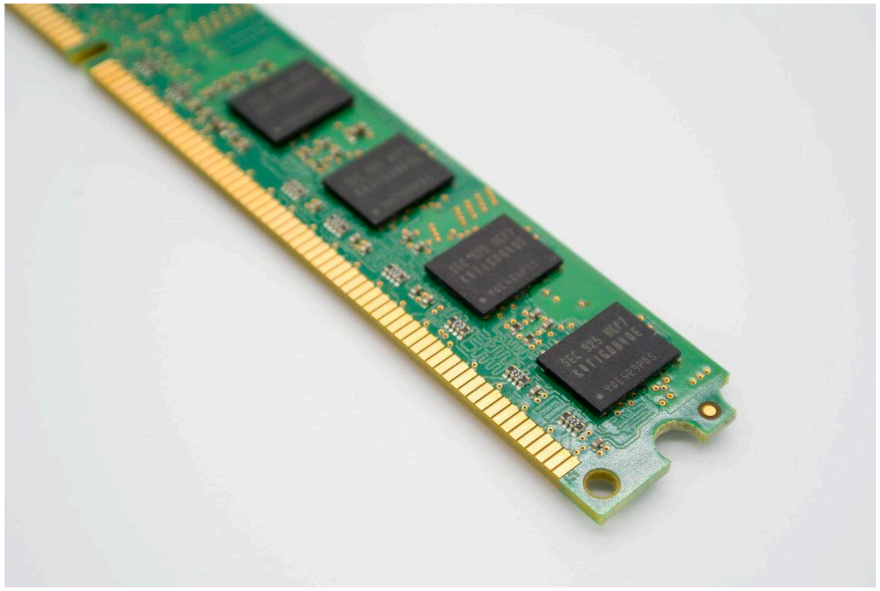
Activity: Installing RAM

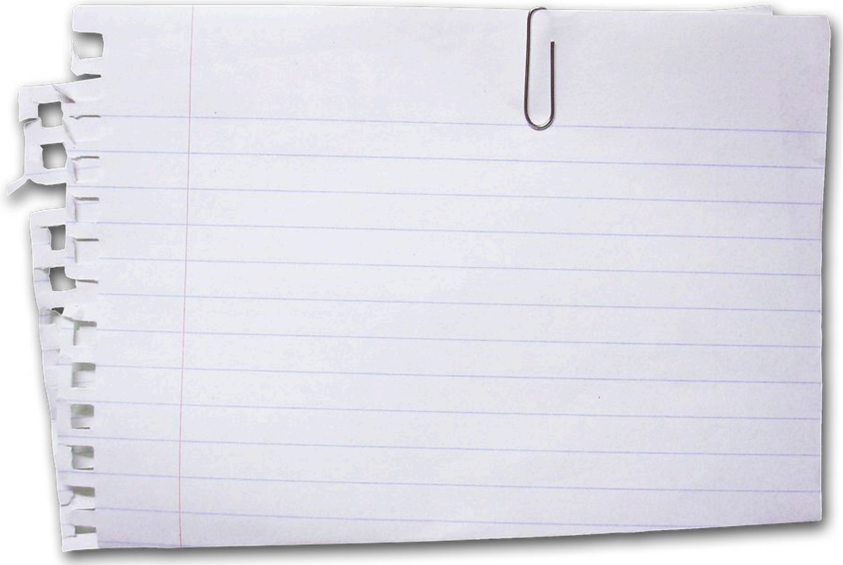
Cache Memory

Cache memory is like the brain of a goldfish. Its super fast to access but you can't store much in there unfortunately.

RAM
Random Access Memory

RAM is like sheets of paper on your desk. You can store more information on them than in a goldfish brain but it takes longer to access the information and, if not filed away properly after use it will be lost.

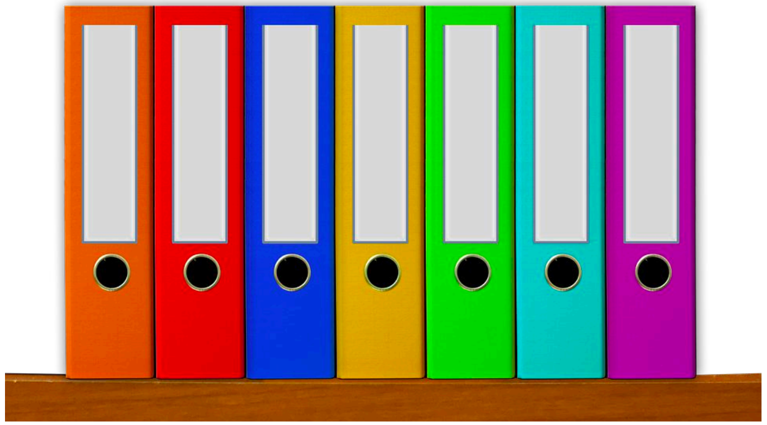




technocamps

Mass Storage Devices

Hard Disk Drive (HDD) , **Solid State Drive** are mass storage devices and are like a file or folder. It can store even more information than the loose pieces of paper but takes even longer to access the information stored inside. Most importantly you don't lose the sheets in the file when you stop using them.



Storage Measurements

Internal memory in a computer is used for two main purposes:

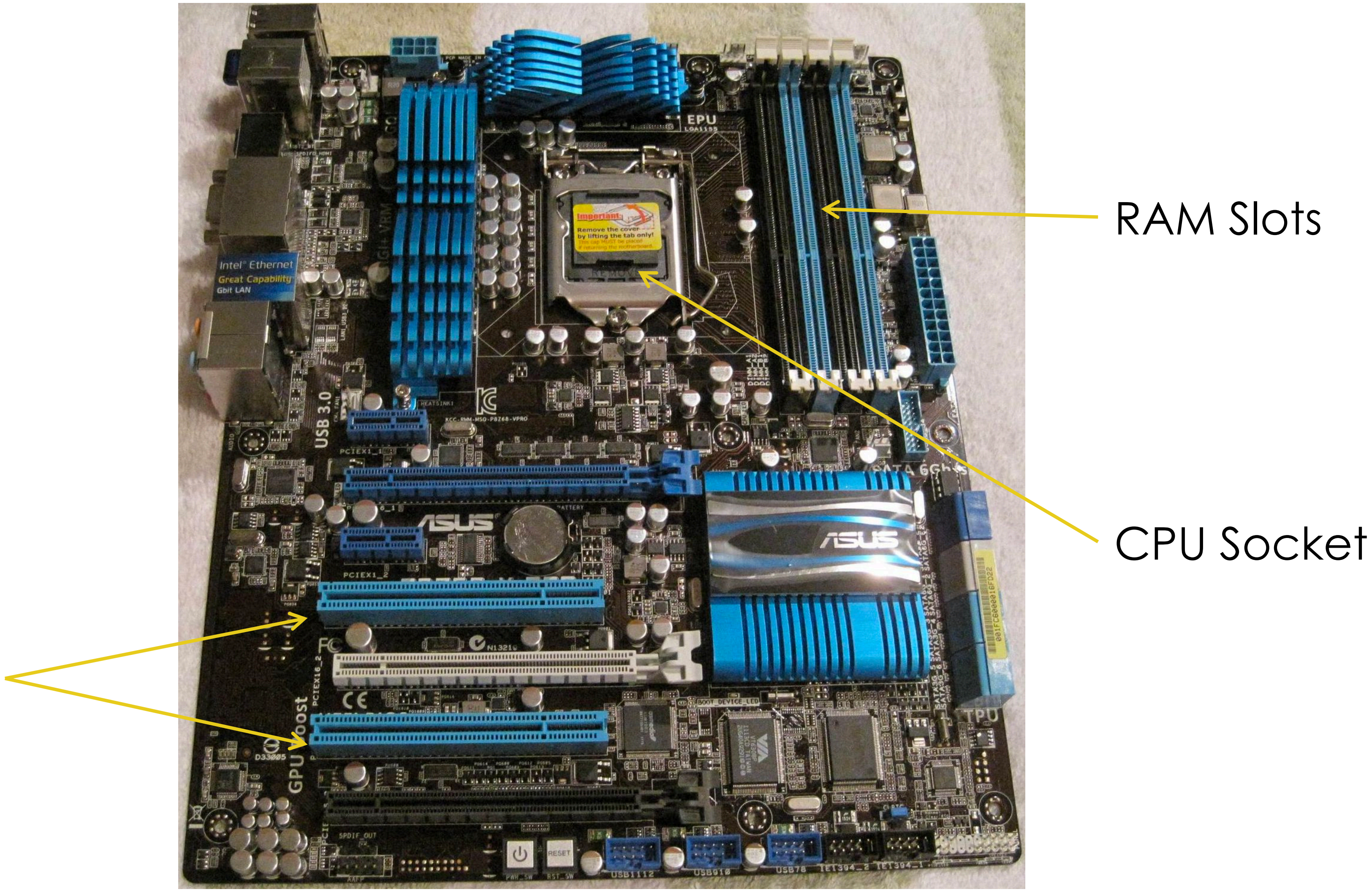
- to store programs that are being run and
- to store the data that the program works on

Memory is measured in bytes, like KB, GB etc.

Having a greater amount of RAM results in there being more space for more instructions that can be held close to the processor at any one time, and this reduces the amount of time spent swapping data into and out of RAM.

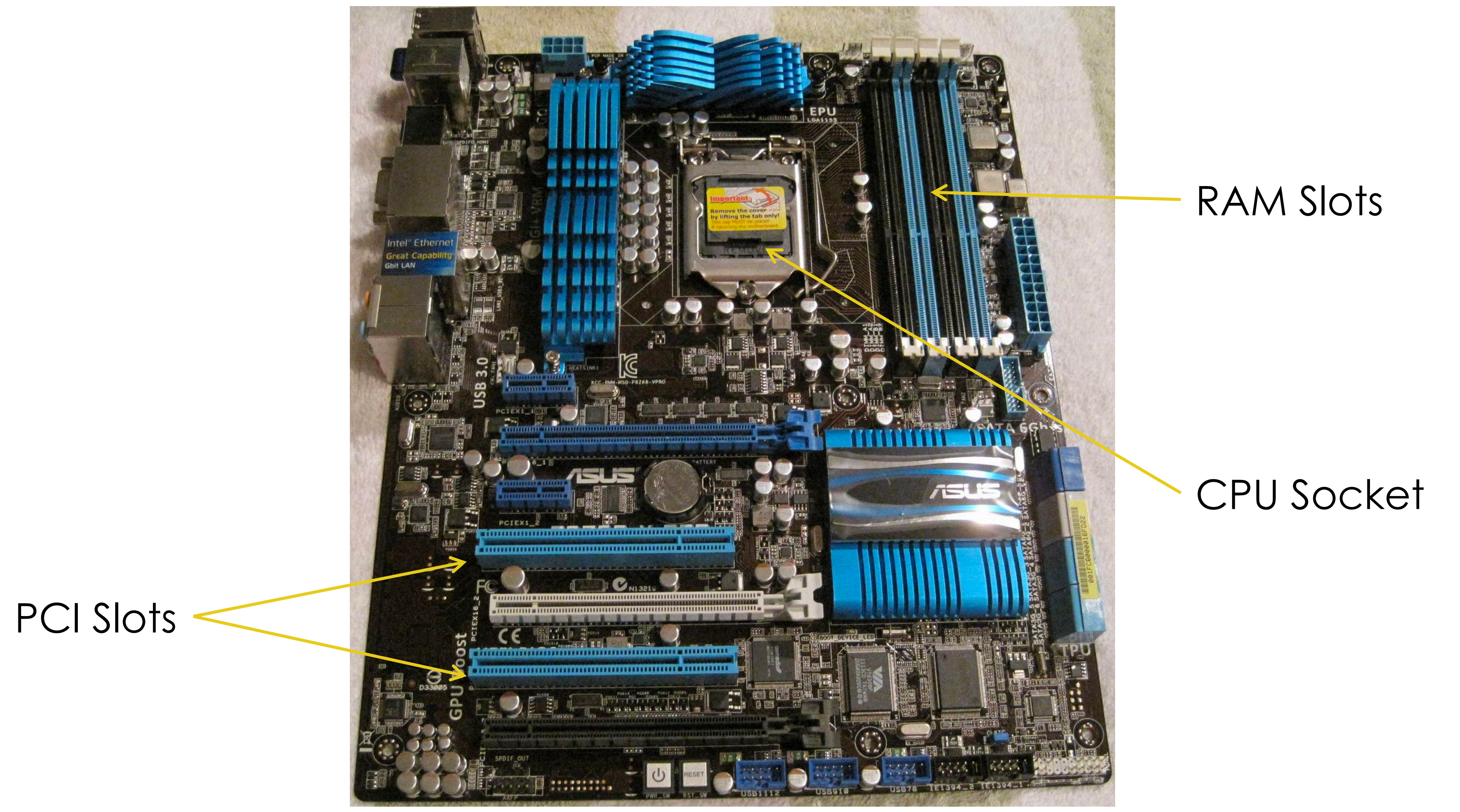
Activity: Memory

Computer Components



technocamps

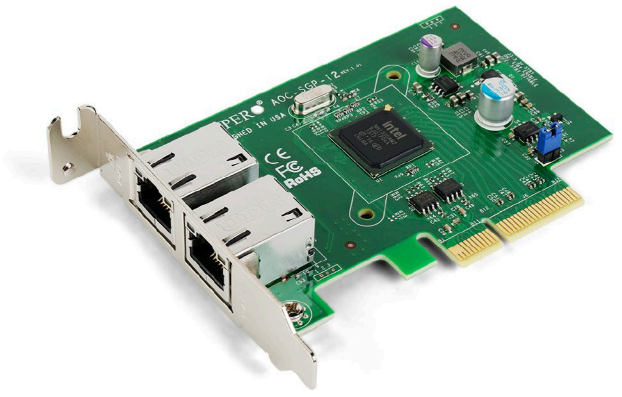
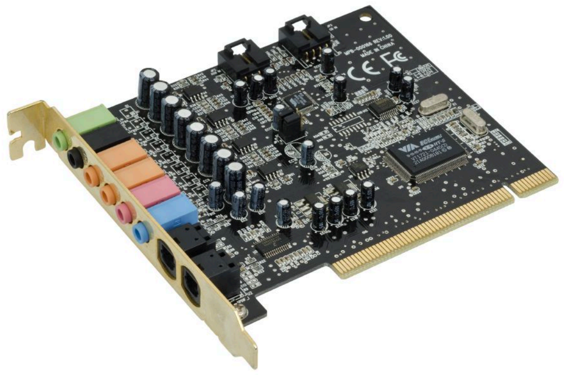
Computer Components



PCI
Peripheral Component Interconnect

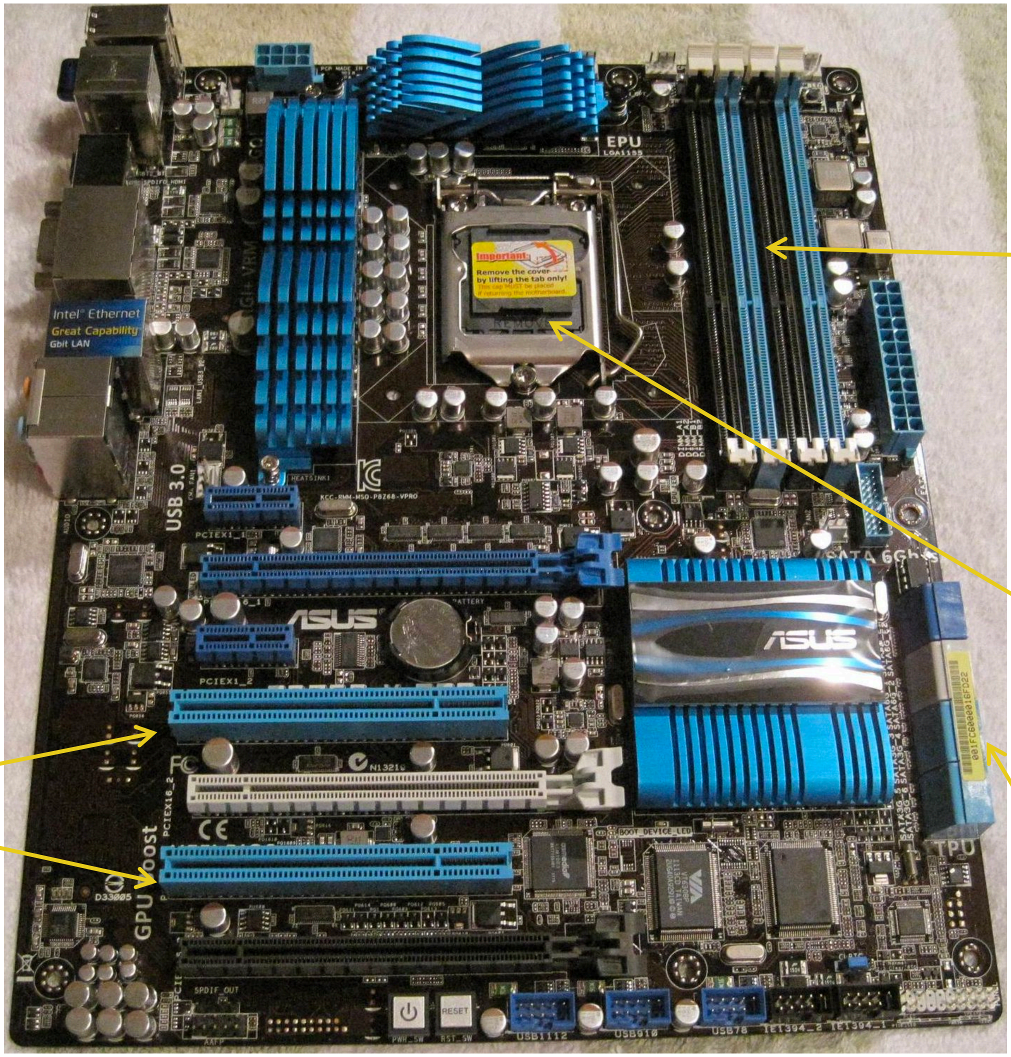
PCI ports are used inside personal computers for connecting peripherals such as dedicated sound cards, graphics cards or wireless internet cards.

Expansion slots allow the life of a computer to be extended, since new technology can be added as it becomes available like better graphics cards for games or dedicated ethernet cards for faster internet access.

technocamps

Computer Components



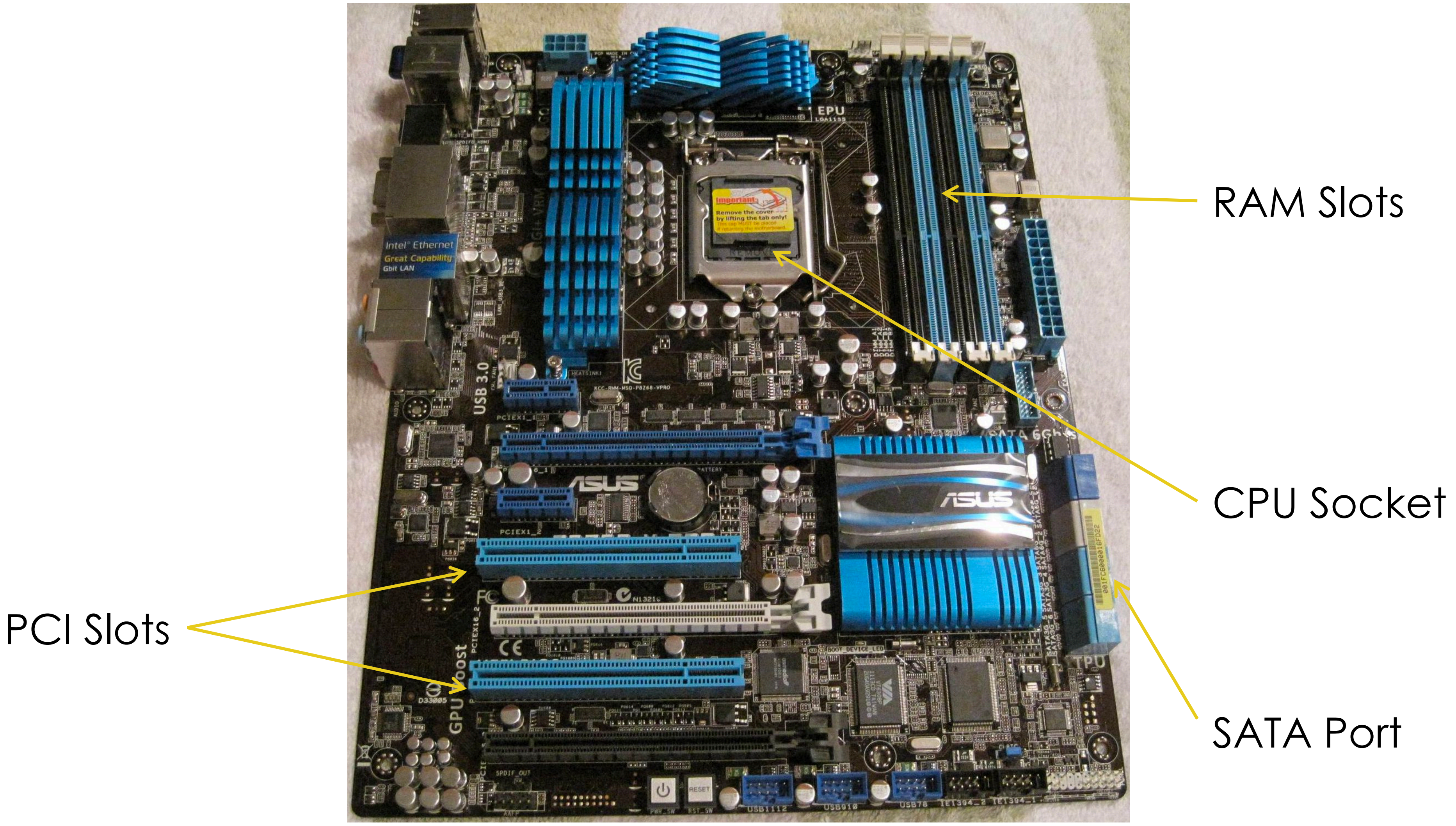
RAM Slots

CPU Socket

PCI Slots

technocamps

Computer Components



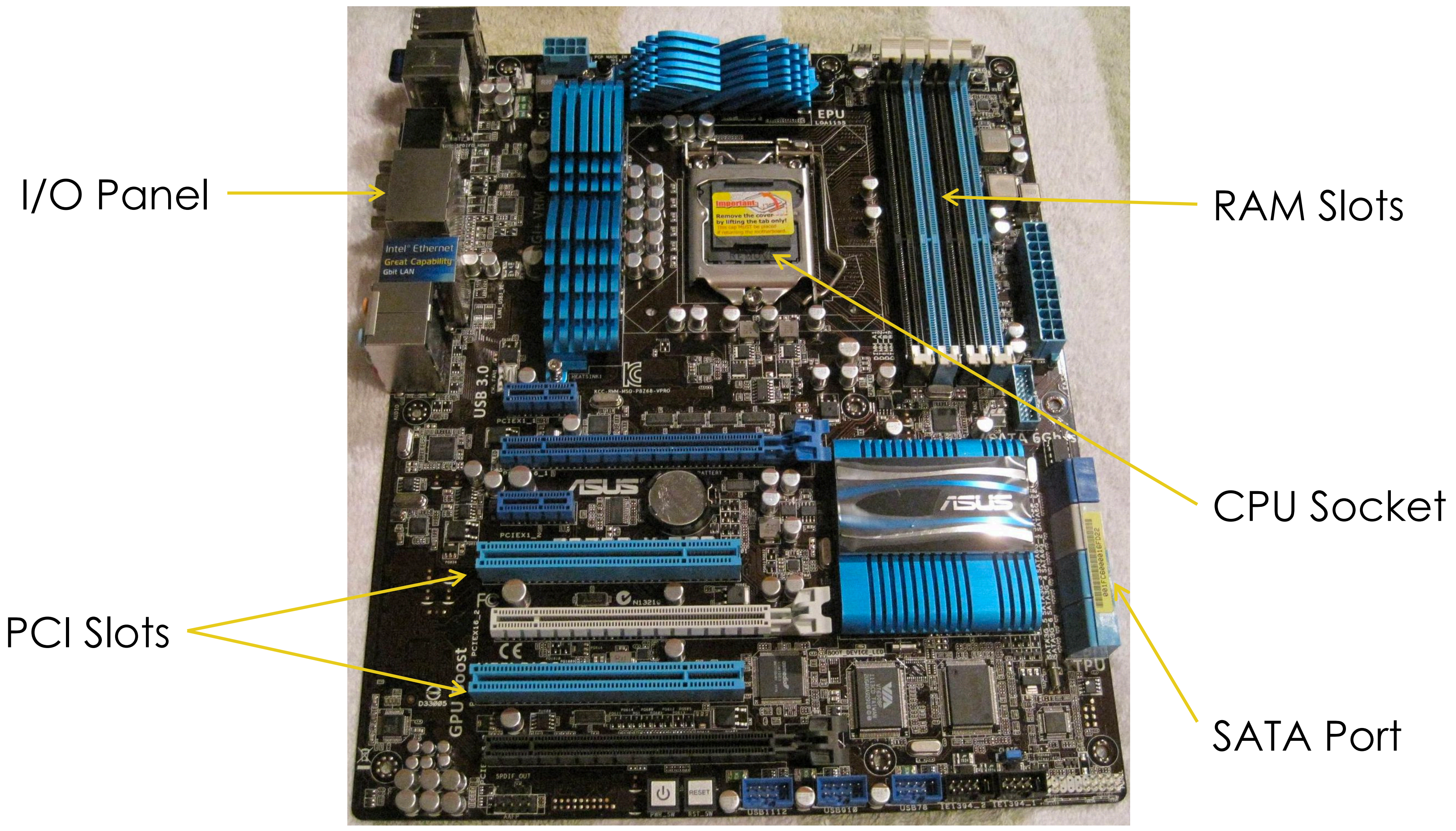
technocamps

Activity: Installing a Hard Drive

SATA

SATA (serial advanced technology attachment) is a computer bus interface for connecting host bus adapters to mass storage devices such as hard disk drives and optical drives.

Computer Components



technocamps

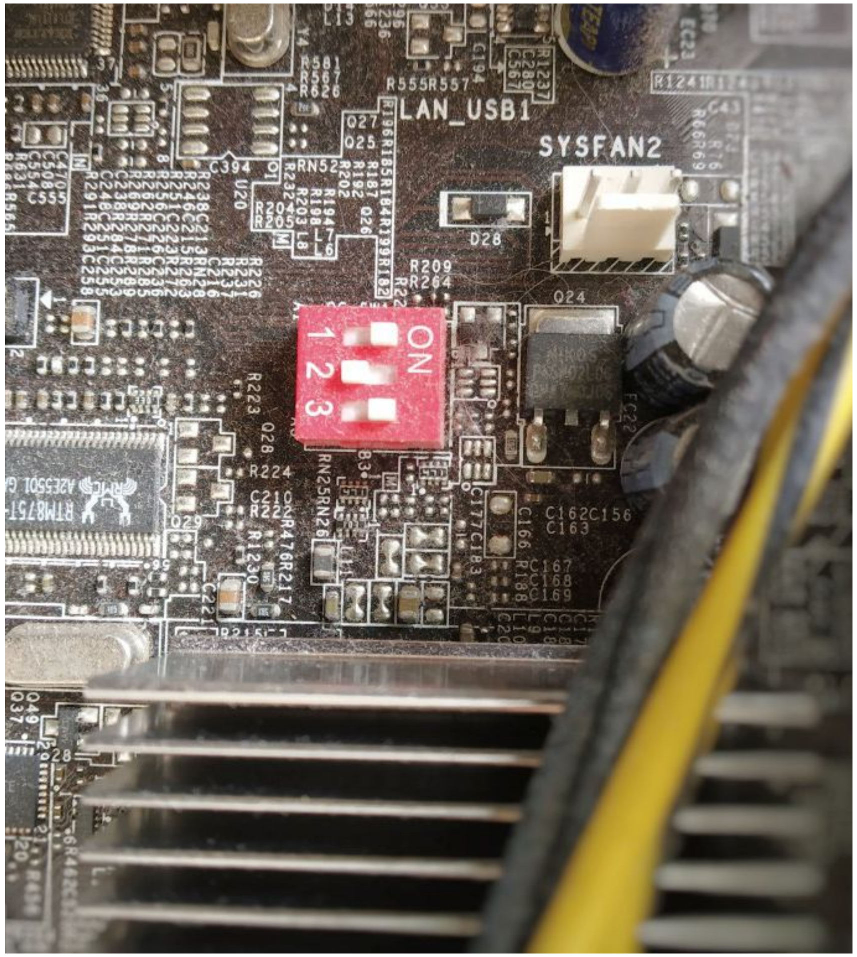
I/O Device Ports

Input devices such as a keyboard, mouse or scanner provide a way for the user to put data into the processor and to give commands (e.g. a pointing device like a mouse to click on the application you want to open, or to select an option from a menu).

Output devices such as a monitor or printer present the results of any processing to the user.

technocamps

Extension: Overclocking



Overclocking increases the operating speed of a given component.

The target of overclocking is increasing the performance of a major chip or subsystem, such as the main processor or graphics controller, but other components, such as system memory (RAM) or system buses (generally on the motherboard), are commonly involved.

technocamps

Hooray!!!

technocamps

eMac(2005) vs. iMac Pro

Processor and RAM

<u>eMac</u>		<u>iMac Pro</u>	
Clock speed:	1.42 GHz	Clock speed:	3.2GHz
Number of Cores:	1	Number of Cores:	8
Cache size:	512KB	Cache size:	11MB
Standard RAM size:	256MB	Standard RAM size:	32GB
Max RAM size:	2GB	Max RAM size:	256GB
Min RAM speed:	333MHz	Min RAM speed:	2666MHz

Storage and Graphics

eMac		iMac Pro	
Standard VRAM:	64MB	Standard VRAM:	8GB
Max VRAM:	64MB	Max VRAM:	16GB
Display support:	17”	Display support:	27”
Resolution support:	1024x768	Native resolution:	5120x2880
Storage:	80 GB HDD	Storage:	1TB SSD

technocamps

Which Would You Rather Have?

Activity: Compare Technology

Compare either the Xbox One X and the Playstation 4 Pro or the HP Envy 13 (2018) and HP Pavilion 15-cs1006na.

Try and find out the following specifications

Processor:

Core/Clock speed:

Weight:

Memory (Internal and External):

Audio/Video capabilities:

Extras:

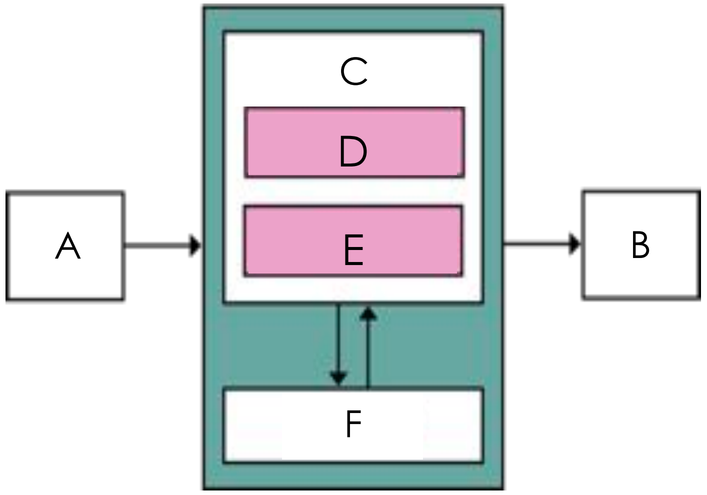
Compare their performance and identify the one you think is better for you and write down why.

Computer Architecture

Computer Architecture

Just like architecture of a building, computer architecture is the way that a computer is designed to function in terms of hardware.

The most common architecture is known as von Neumann architecture. This architecture is made up of:

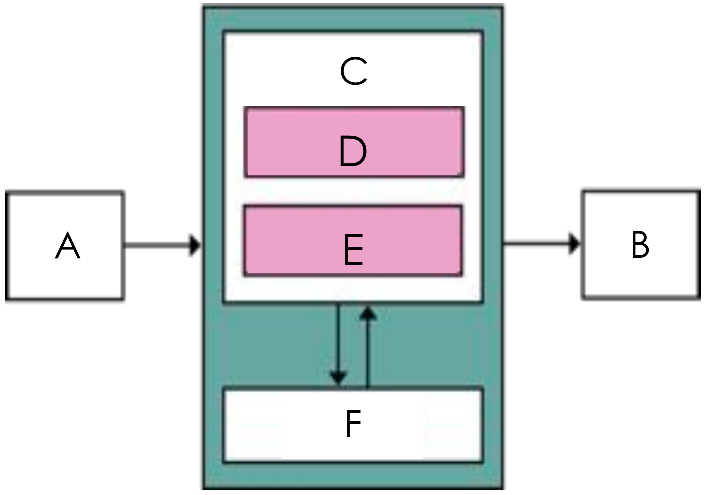


Computer Architecture

Just like architecture of a building, computer architecture is the way that a computer is designed to function in terms of hardware.

The most common architecture is known as von Neumann architecture. This architecture is made up of:

- CPU – Control unit, Arithmetic and Logic unit and registers

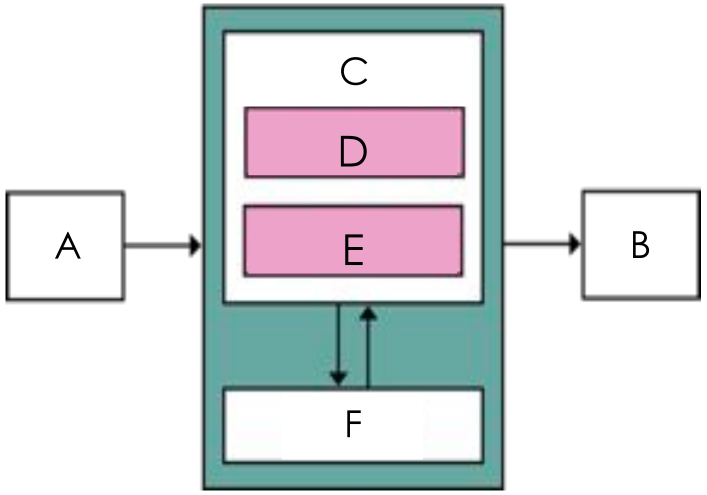


Computer Architecture

Just like architecture of a building, computer architecture is the way that a computer is designed to function in terms of hardware.

The most common architecture is known as von Neumann architecture. This architecture is made up of:

- CPU – Control unit, Arithmetic and Logic unit and registers
- Memory unit – RAM



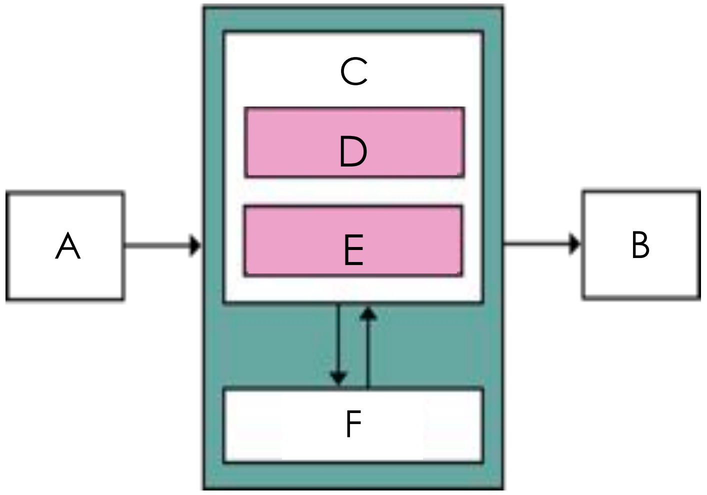
technocamps

Computer Architecture

Just like architecture of a building, computer architecture is the way that a computer is designed to function in terms of hardware.

The most common architecture is known as von Neumann architecture. This architecture is made up of:

- CPU – Control unit, Arithmetic and Logic unit and registers
- Memory unit – RAM
- Buses – Data/address/control



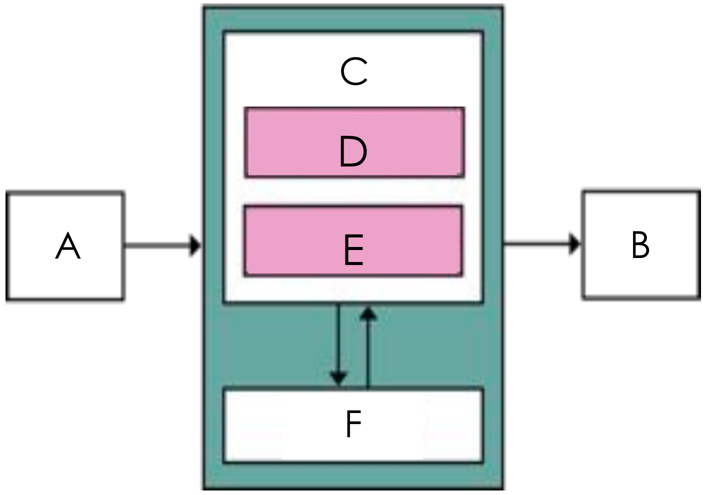
technocamps

Computer Architecture

Just like architecture of a building, computer architecture is the way that a computer is designed to function in terms of hardware.

The most common architecture is known as von Neumann architecture. This architecture is made up of:

- CPU – Control unit, Arithmetic and Logic unit and registers
- Memory unit – RAM
- Buses – Data/address/control
- Input device – Keyboard, mouse

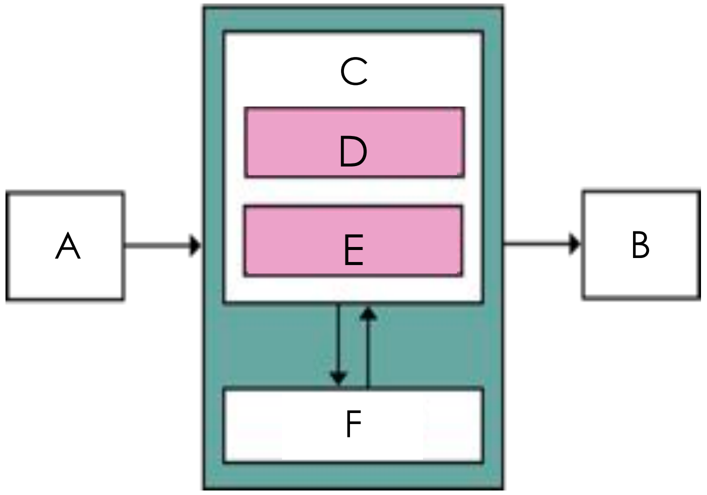


Computer Architecture

Just like architecture of a building, computer architecture is the way that a computer is designed to function in terms of hardware.

The most common architecture is known as von Neumann architecture. This architecture is made up of:

- CPU – Control unit, Arithmetic and Logic unit and registers
- Memory unit – RAM
- Buses – Data/address/control
- Input device – Keyboard, mouse
- Output device – Monitor, speakers



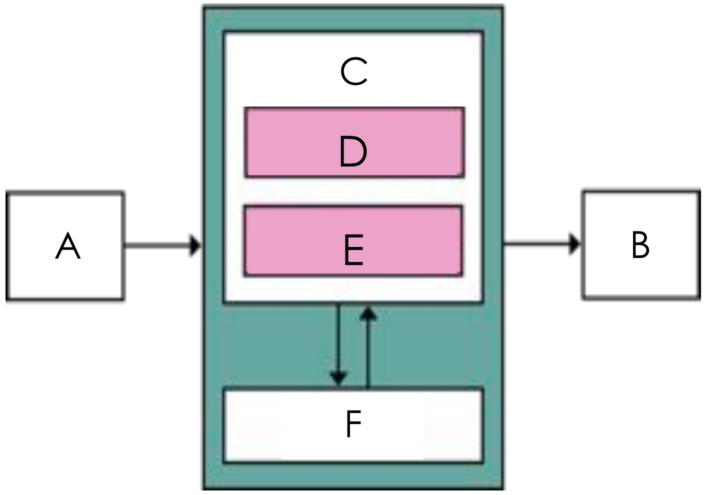
Computer Architecture

Just like architecture of a building, computer architecture is the way that a computer is designed to function in terms of hardware.

The most common architecture is known as von Neumann architecture. This architecture is made up of:

- CPU – Control unit, Arithmetic and Logic unit and registers
- Memory unit – RAM
- Buses – Data/address/control
- Input device – Keyboard, mouse
- Output device – Monitor, speakers

Activity: von Neumann Architecture



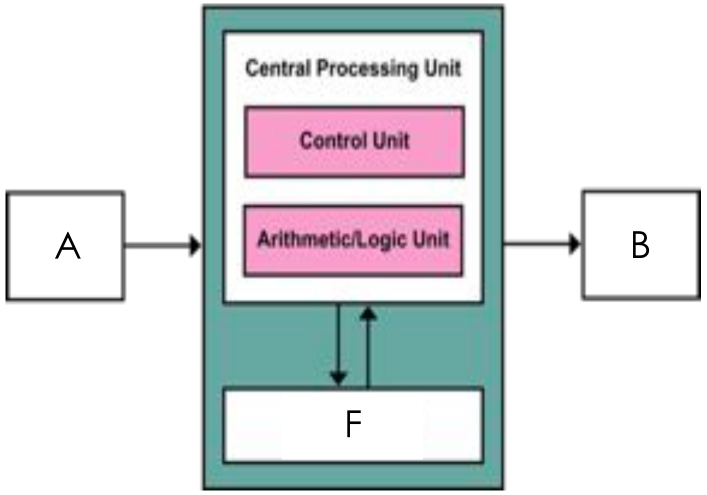
Computer Architecture

Just like architecture of a building, computer architecture is the way that a computer is designed to function in terms of hardware.

The most common architecture is known as von Neumann architecture. This architecture is made up of:

- CPU – Control unit, Arithmetic and Logic unit and registers
- Memory unit – RAM
- Buses – Data/address/control
- Input device – Keyboard, mouse
- Output device – Monitor, speakers

Activity: von Neumann Architecture



technocamps

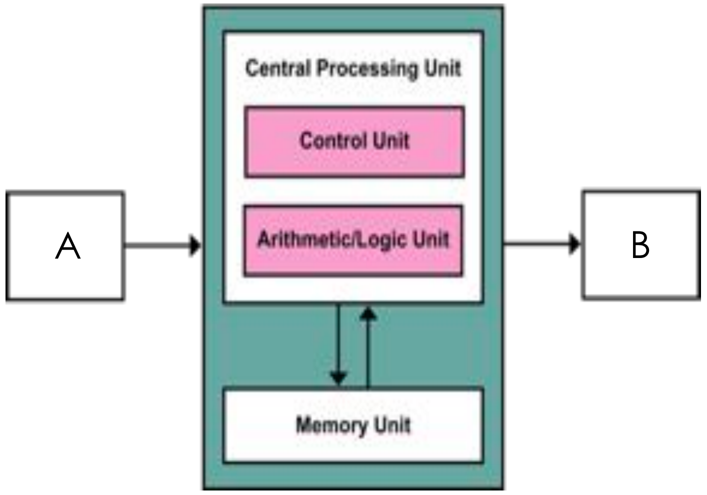
Computer Architecture

Just like architecture of a building, computer architecture is the way that a computer is designed to function in terms of hardware.

The most common architecture is known as von Neumann architecture. This architecture is made up of:

- CPU – Control unit, Arithmetic and Logic unit and registers
- Memory unit – RAM
- Buses – Data/address/control
- Input device – Keyboard, mouse
- Output device – Monitor, speakers

Activity: von Neumann Architecture



technocamps

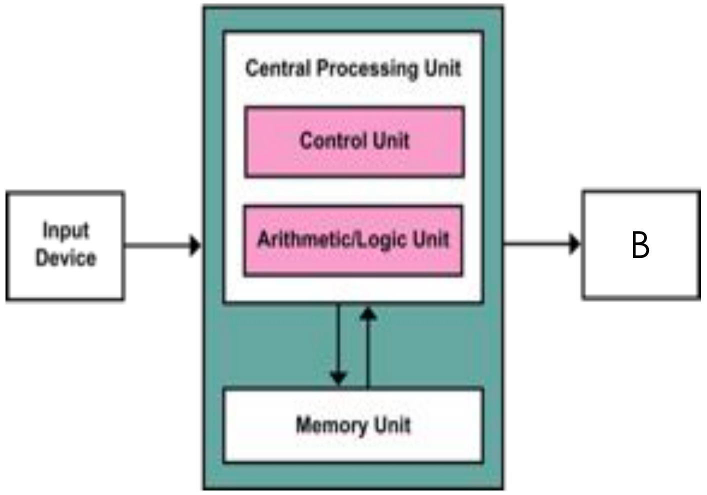
Computer Architecture

Just like architecture of a building, computer architecture is the way that a computer is designed to function in terms of hardware.

The most common architecture is known as von Neumann architecture. This architecture is made up of:

- CPU – Control unit, Arithmetic and Logic unit and registers
- Memory unit – RAM
- Buses – Data/address/control
- Input device – Keyboard, mouse
- Output device – Monitor, speakers

Activity: von Neumann Architecture



technocamps

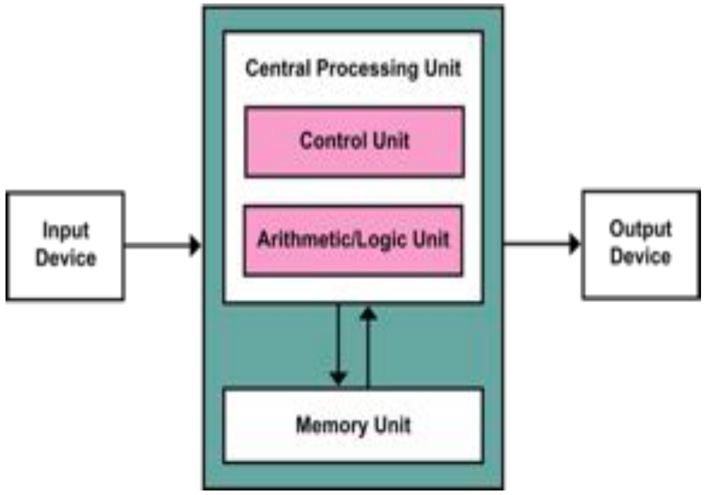
Computer Architecture

Activity: von Neumann Architecture

Just like architecture of a building, computer architecture is the way that a computer is designed to function in terms of hardware.

The most common architecture is known as von Neumann architecture. This architecture is made up of:

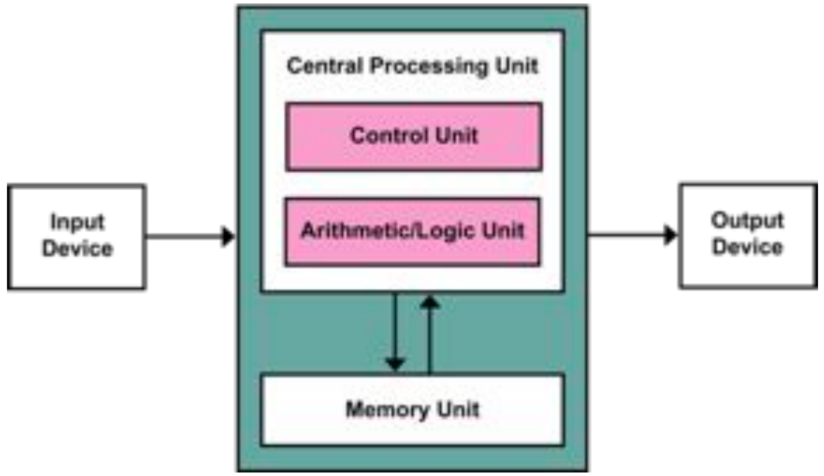
- CPU – Control unit, Arithmetic and Logic unit and registers
- Memory unit – RAM
- Buses – Data/address/control
- Input device – Keyboard, mouse
- Output device – Monitor, speakers



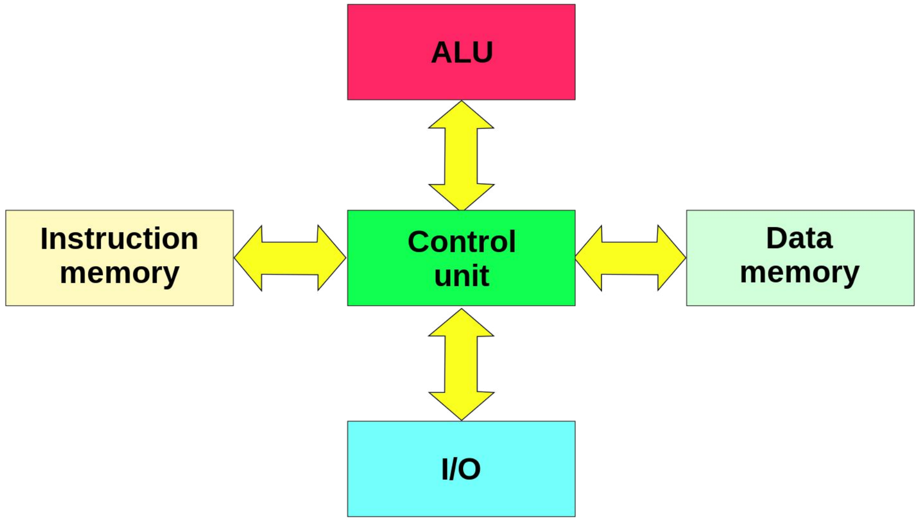
technocamps

von Neumann vs. Harvard

von Neumann Architecture



Harvard Architecture



This stores both instructions and data within the same memory addresses and uses the same bus for both.

This has separate memory addresses for instructions and data meaning it can run a program and access data simultaneously.

von Neumann vs. Harvard

	von Neumann	Harvard
Flexibility		
Speed		
Cost		
Examples		

von Neumann vs. Harvard

	von Neumann	Harvard
Flexibility	High level of flexibility as the memory is shared between instructions and data so the amount assigned to each can fluctuate depending on the task.	
Speed		
Cost		
Examples		

technocamps

von Neumann vs. Harvard

	von Neumann	Harvard
Flexibility	High level of flexibility as the memory is shared between instructions and data so the amount assigned to each can fluctuate depending on the task.	Limited flexibility as there is only a certain amount of memory that can be used for data and a certain amount for instructions.
Speed		
Cost		
Examples		

technocamps

von Neumann vs. Harvard

	von Neumann	Harvard
Flexibility	High level of flexibility as the memory is shared between instructions and data so the amount assigned to each can fluctuate depending on the task.	Limited flexibility as there is only a certain amount of memory that can be used for data and a certain amount for instructions.
Speed	Speed is limited when compared to Harvard due to only having one memory location and one set of buses.	
Cost		
Examples		

technocamps

von Neumann vs. Harvard

	von Neumann	Harvard
Flexibility	High level of flexibility as the memory is shared between instructions and data so the amount assigned to each can fluctuate depending on the task.	Limited flexibility as there is only a certain amount of memory that can be used for data and a certain amount for instructions.
Speed	Speed is limited when compared to Harvard due to only having one memory location and one set of buses.	Two sets of memory and buses mean data can be handled more quickly which would result in decreased execution time.
Cost		
Examples		

von Neumann vs. Harvard

	von Neumann	Harvard
Flexibility	High level of flexibility as the memory is shared between instructions and data so the amount assigned to each can fluctuate depending on the task.	Limited flexibility as there is only a certain amount of memory that can be used for data and a certain amount for instructions.
Speed	Speed is limited when compared to Harvard due to only having one memory location and one set of buses.	Two sets of memory and buses mean data can be handled more quickly which would result in decreased execution time.
Cost	Simpler control unit design, and development of one bus is cheaper and faster.	
Examples		

technocamps

von Neumann vs. Harvard

	von Neumann	Harvard
Flexibility	High level of flexibility as the memory is shared between instructions and data so the amount assigned to each can fluctuate depending on the task.	Limited flexibility as there is only a certain amount of memory that can be used for data and a certain amount for instructions.
Speed	Speed is limited when compared to Harvard due to only having one memory location and one set of buses.	Two sets of memory and buses mean data can be handled more quickly which would result in decreased execution time.
Cost	Simpler control unit design, and development of one bus is cheaper and faster.	Control unit for two buses is more complicated which adds to the development cost.
Examples		

technocamps

von Neumann vs. Harvard

	von Neumann	Harvard
Flexibility	High level of flexibility as the memory is shared between instructions and data so the amount assigned to each can fluctuate depending on the task.	Limited flexibility as there is only a certain amount of memory that can be used for data and a certain amount for instructions.
Speed	Speed is limited when compared to Harvard due to only having one memory location and one set of buses.	Two sets of memory and buses mean data can be handled more quickly which would result in decreased execution time.
Cost	Simpler control unit design, and development of one bus is cheaper and faster.	Control unit for two buses is more complicated which adds to the development cost.
Examples	Typically used in general purpose computers that will be used for many different purposes.	

von Neumann vs. Harvard

	von Neumann	Harvard
Flexibility	High level of flexibility as the memory is shared between instructions and data so the amount assigned to each can fluctuate depending on the task.	Limited flexibility as there is only a certain amount of memory that can be used for data and a certain amount for instructions.
Speed	Speed is limited when compared to Harvard due to only having one memory location and one set of buses.	Two sets of memory and buses mean data can be handled more quickly which would result in decreased execution time.
Cost	Simpler control unit design, and development of one bus is cheaper and faster.	Control unit for two buses is more complicated which adds to the development cost.
Examples	Typically used in general purpose computers that will be used for many different purposes.	Typically used in embedded systems that only perform few functions like washing machines, burglar alarms etc.

Activity: Flexibility

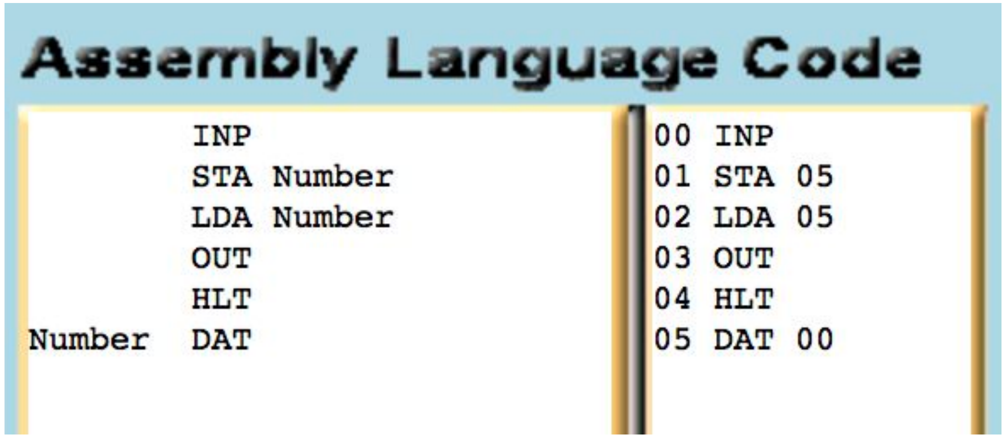
Assembly Language

technocamps

Assembly Language

Assembly language is a low-level programming language which uses an assembler to convert a program into machine code which can be run by the computer.

Assembly languages usually use short mnemonics as instructions and each one is specific to the computer architecture and operating system.



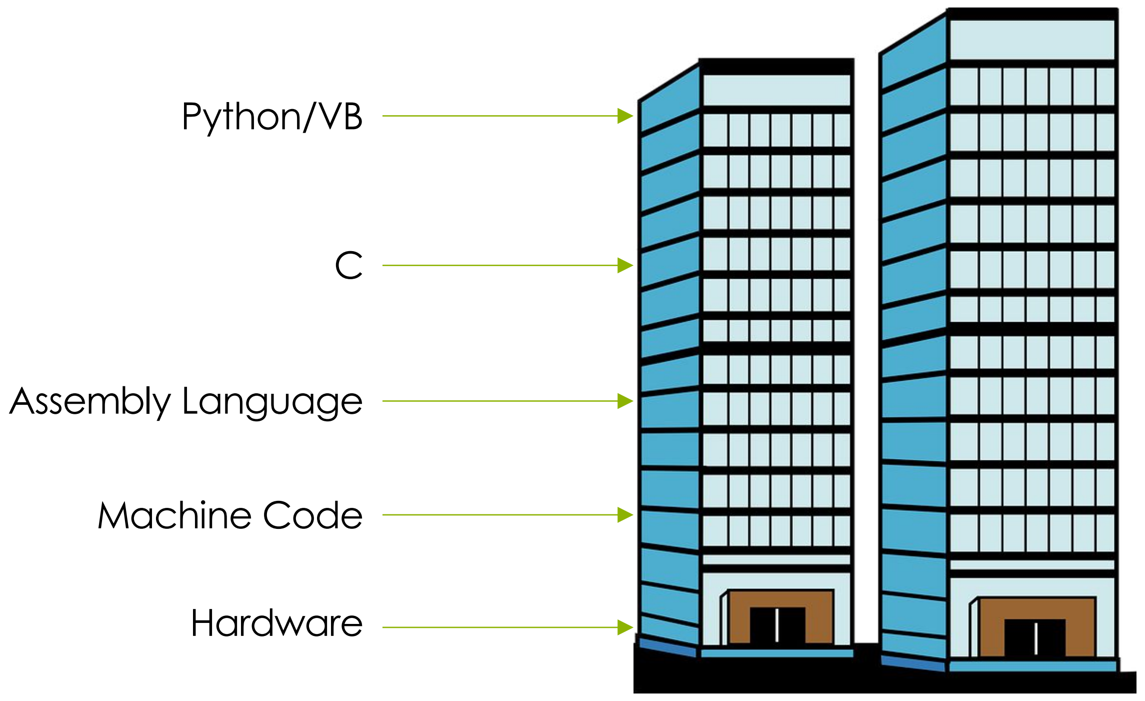
```
        INP
        STA Number
        LDA Number
        OUT
        HLT
Number  DAT
```

```
00 INP
01 STA 05
02 LDA 05
03 OUT
04 HLT
05 DAT 00
```

Little Man Computer

Assembly Language

Assembly languages are considered to be low-level because they are very close to machine languages. They are only one step removed from a computer's machine language.



Why Assembly?

Relationship between Assembly Language and Machine Code

A CPU cannot directly read source code. Different CPUs may have different architecture and each different architecture has its own machine language.

This prevents direct source code to machine code translation – we need to use an assembly language to assemble the code which bridges the gap.

For example, a piece of Python code assembled to run on a 64 bit Windows machine will not have the same instruction set as the Python code assembled to run on a 32 bit Linux machine.

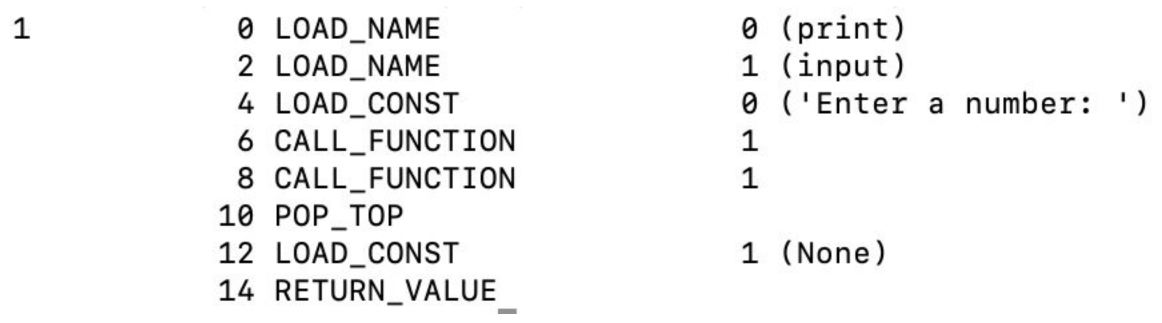
technocamps

Python → Assembly

Simple one line program in Python:

```
print(input("Enter a number: " ))
```

Internally, it is converted to an Assembly language which looks like:



```
  1           0 LOAD_NAME                0 (print)
              2 LOAD_NAME                1 (input)
              4 LOAD_CONST               0 ('Enter a number: ')
              6 CALL_FUNCTION            1
              8 CALL_FUNCTION            1
             10 POP_TOP
             12 LOAD_CONST               1 (None)
             14 RETURN_VALUE
```

technocamps

Assembly Language

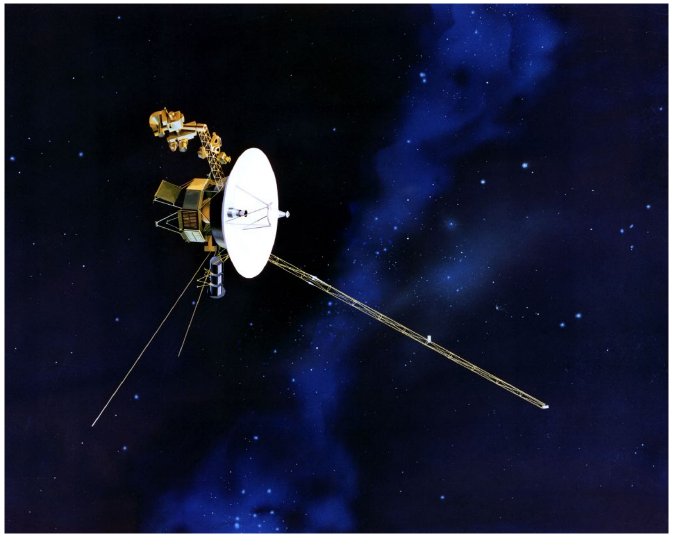
If we wrote the same code in Assembly language, it would look like this.

```
INP
OUT
HLT
```

We have three lines of Assembly language code instead of eight which was converted from Python to Assembly and yet doing the same function as the one in Python.

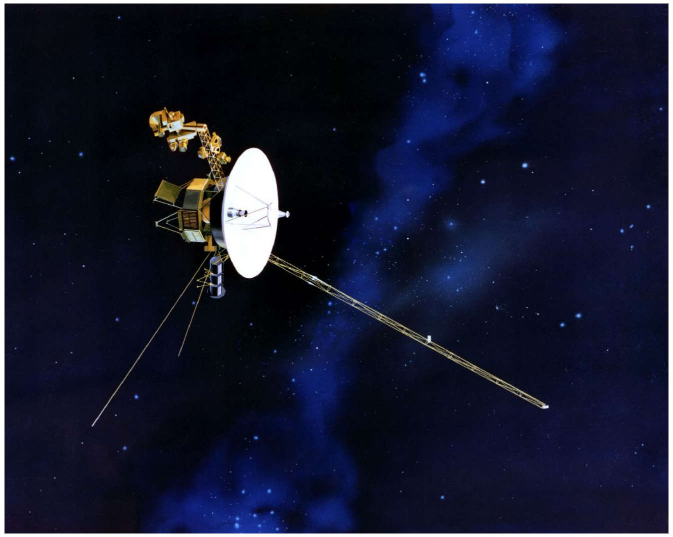
Why Are Assembly Languages Used?

Low-level languages are especially useful when speed of execution is critical or when writing software which interfaces directly with the hardware, e.g. device drivers.



Why Are Assembly Languages Used?

Example: The Voyager space probe launched in 1977 (now outside our solar system) is programmed using an old assembly language. NASA are struggling to find anyone who still has a working knowledge of the language to keep it going!

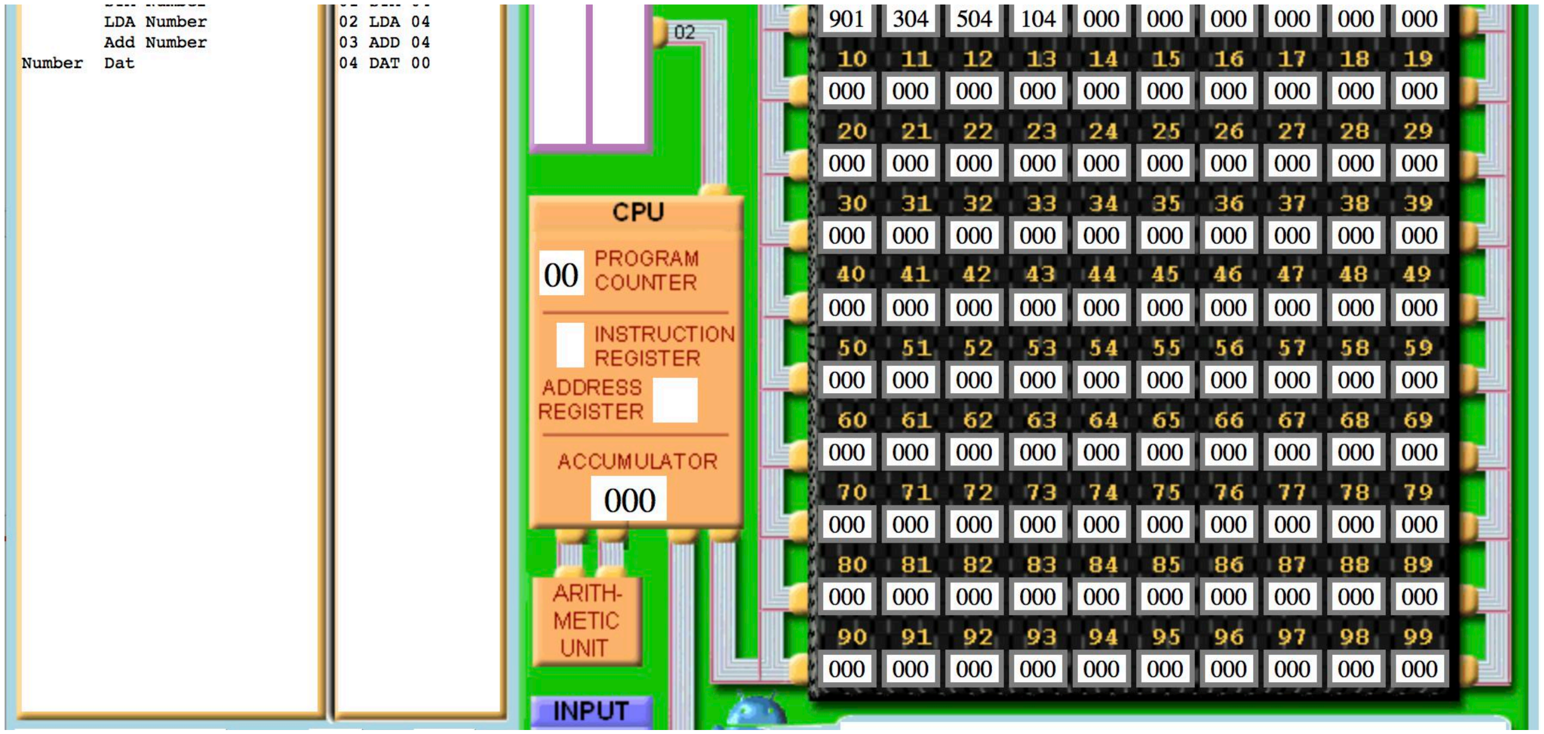


Activity:
What is an Assembly Language?

Little Man Computer (LMC)

Little Man Computer (LMC)

Little Man Computer (LMC) is a simulator that mimics von Neumann architecture.



```
        LDA  Number
        Add  Number
Number  Dat
```

```
02 LDA 04
03 ADD 04
04 DAT 00
```

901	304	504	104	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

LMC Lifecycle

Everything in a computer's memory is data.

Although programs may seem different from data, they are treated in exactly the same way: the computer executes a program, instruction by instruction.

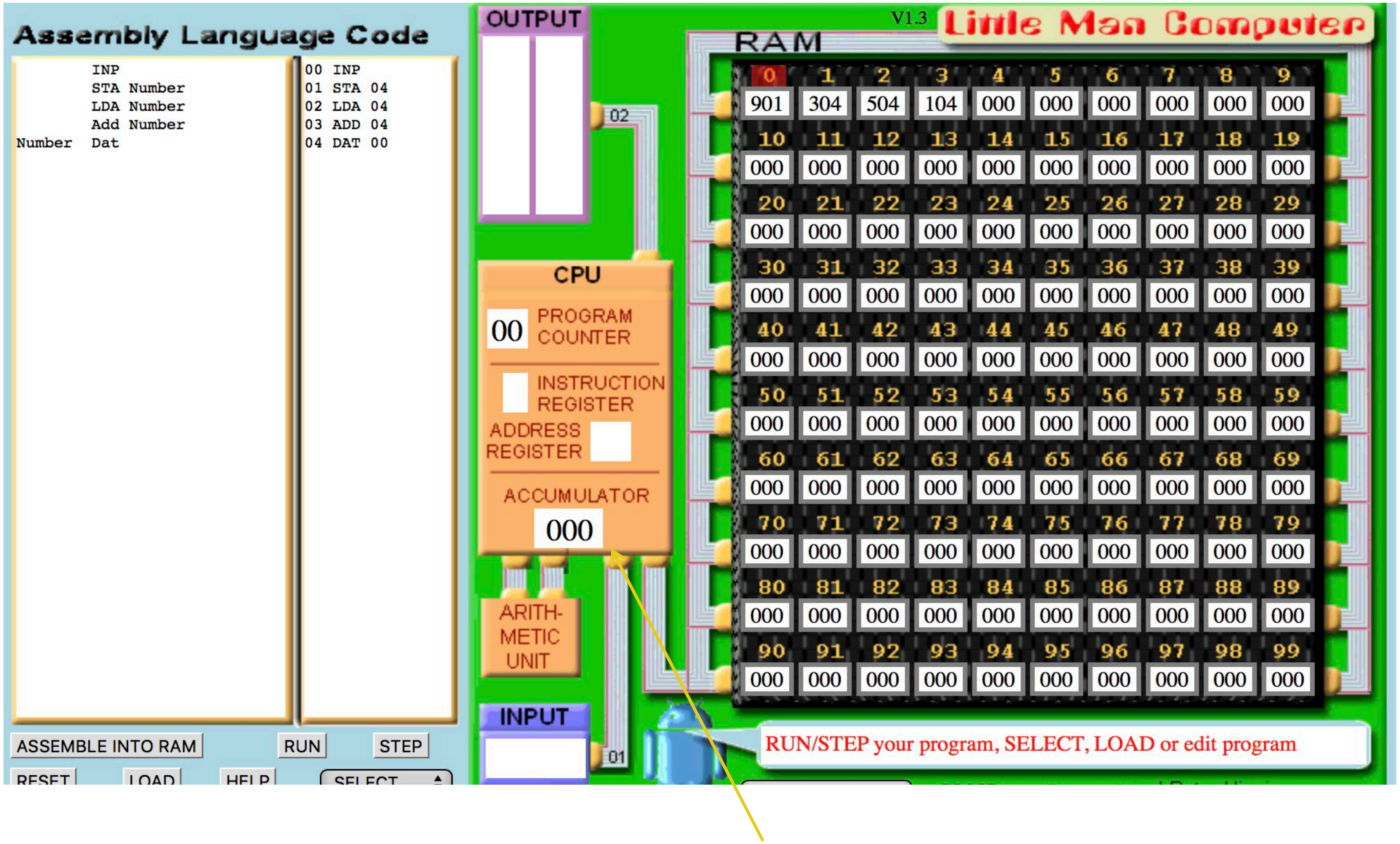
These instructions are the 'data' of the fundamental program cycle:

1. **fetch** the next instruction
2. **decode** it
3. **execute** it

Then the next program cycle starts which will process the next instruction. Even the location of the next instruction is just data.

The LMC Environment

- **Accumulator** – This is like the active memory of the simulator. The majority of our instructions will modify the contents of the Accumulator.



Assembly Language Code

```
        INP
        STA Number
        LDA Number
        Add Number
Number  Dat
```

```
00 INP
01 STA 04
02 LDA 04
03 ADD 04
04 DAT 00
```

PROGRAM COUNTER 00

ACCUMULATOR 000

RAM

0	1	2	3	4	5	6	7	8	9
901	304	504	104	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

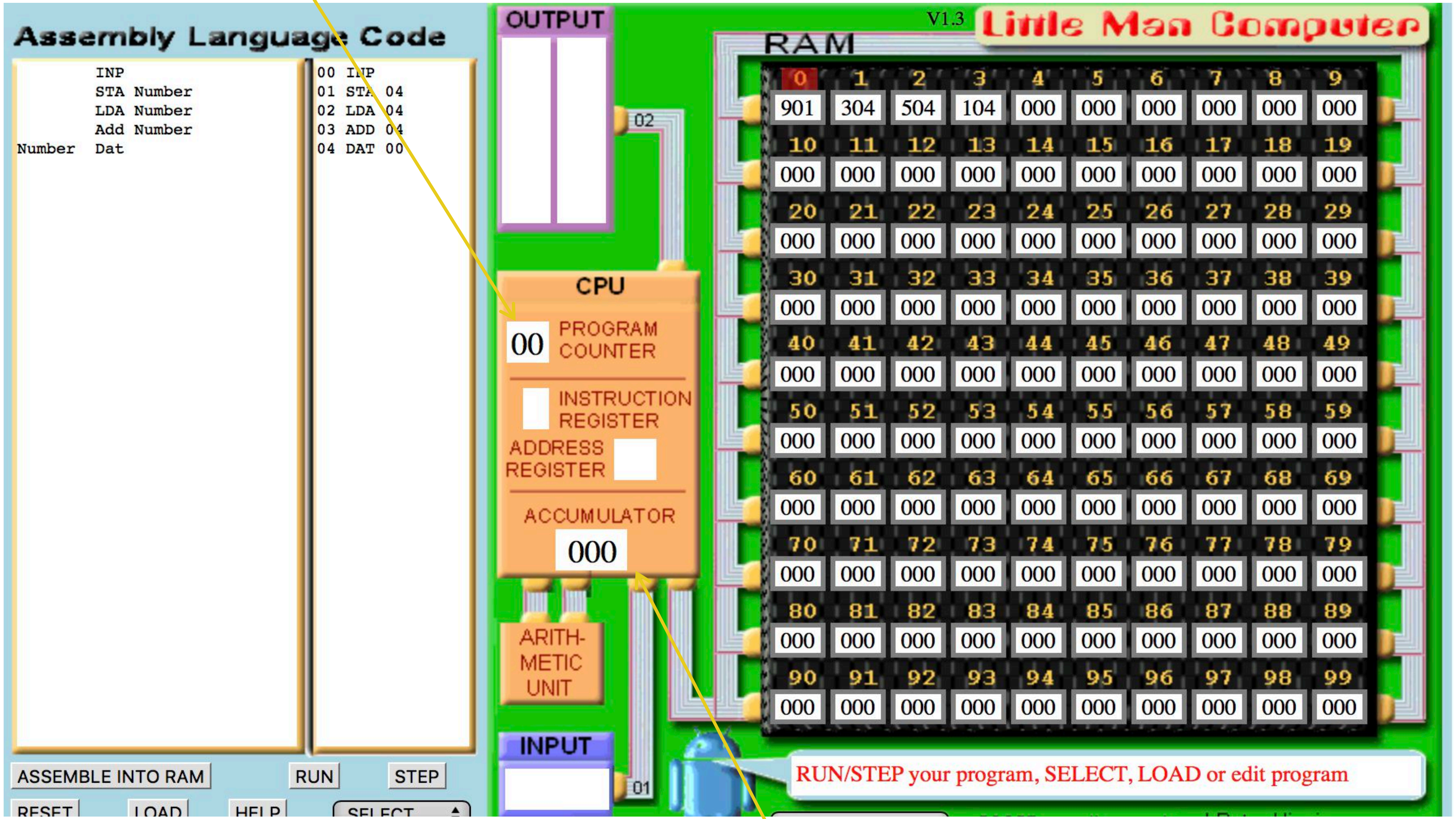
RUN/STEP your program, SELECT, LOAD or edit program

Accumulator

The LMC Environment

- **Accumulator** – This is like the active memory of the simulator. The majority of our instructions will modify the contents of the Accumulator.
- **Program Counter** – This shows the current memory location that the processor is running.

Program Counter



```
        INP
        STA Number
        LDA Number
        Add Number
Number  Dat
```

```
00 INP
01 STA 04
02 LDA 04
03 ADD 04
04 DAT 00
```

0	1	2	3	4	5	6	7	8	9
901	304	504	104	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

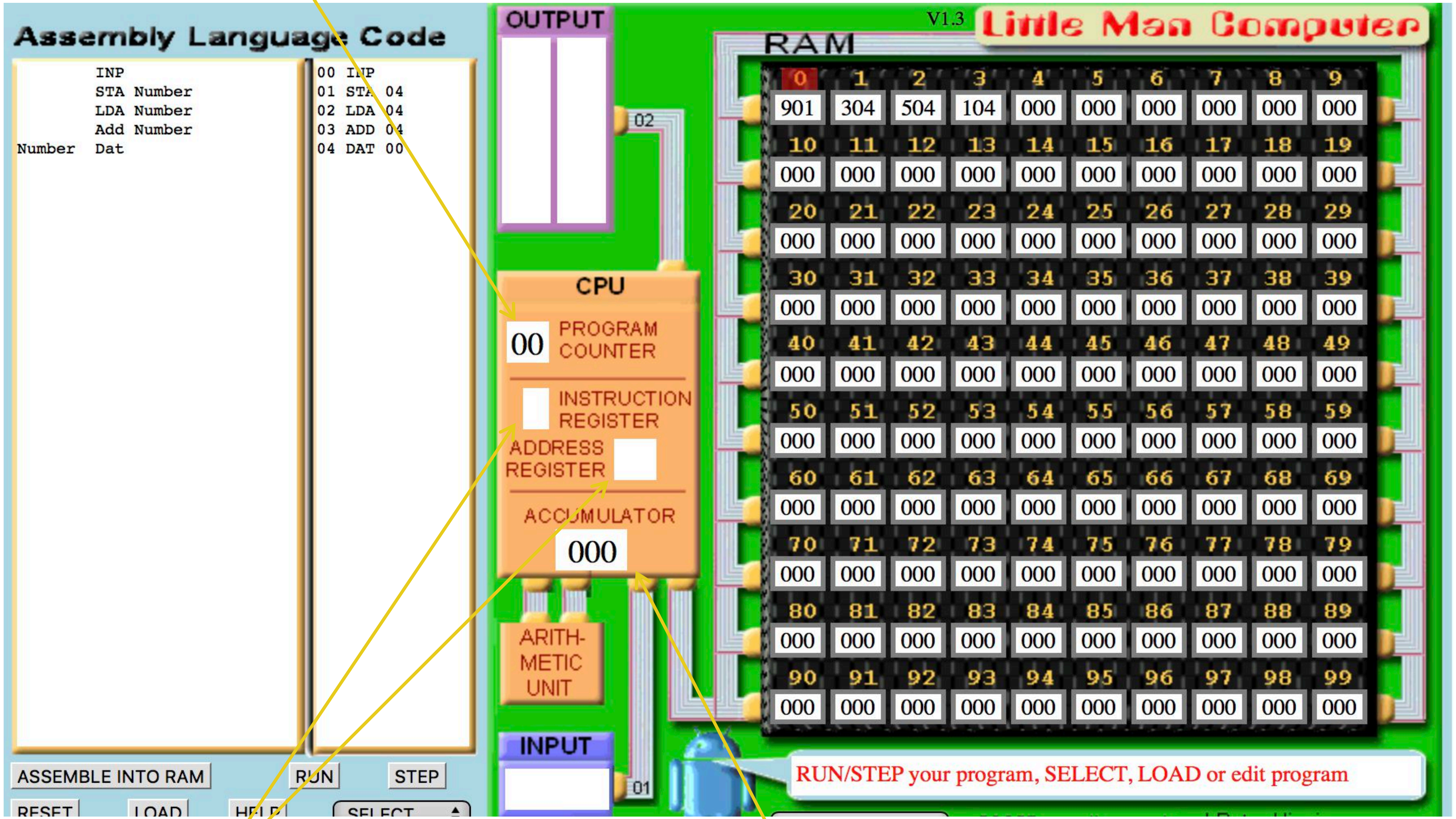
RUN/STEP your program, SELECT, LOAD or edit program

Accumulator

The LMC Environment

- **Accumulator** – This is like the active memory of the simulator. The majority of our instructions will modify the contents of the Accumulator.
- **Program Counter** – This shows the current memory location that the processor is running.
- **Instruction and Address register** – This shows which type of instruction is being used and which memory address it is being used on.

Program Counter



```
        INP
        STA Number
        LDA Number
        Add Number
Number  Dat
```

```
00 INP
01 STA 04
02 LDA 04
03 ADD 04
04 DAT 00
```

0	1	2	3	4	5	6	7	8	9
901	304	504	104	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

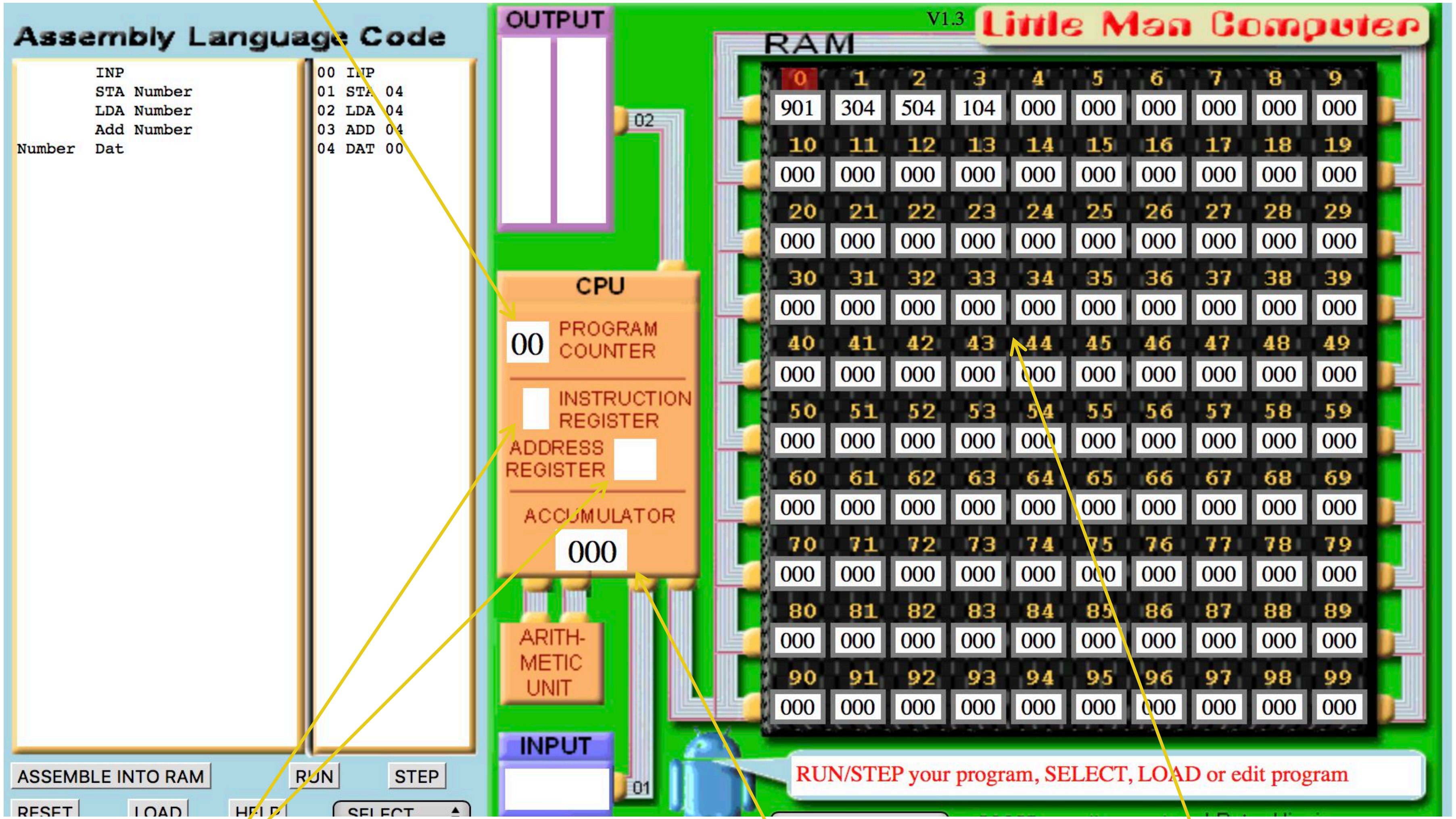
RUN/STEP your program, SELECT, LOAD or edit program

Instruction and Address register

Accumulator

The LMC Environment

- **Accumulator** – This is like the active memory of the simulator. The majority of our instructions will modify the contents of the Accumulator.
- **Program Counter** – This shows the current memory location that the processor is running.
- **Instruction and Address register** – This shows which type of instruction is being used and which memory address it is being used on.
- **Memory Addresses** – These are the memory addresses which are used to store instructions and data.



Assembly Language Code

```
        INP
        STA Number
        LDA Number
        Add Number
Number  Dat
```

```
00 INP
01 STA 04
02 LDA 04
03 ADD 04
04 DAT 00
```

RAM

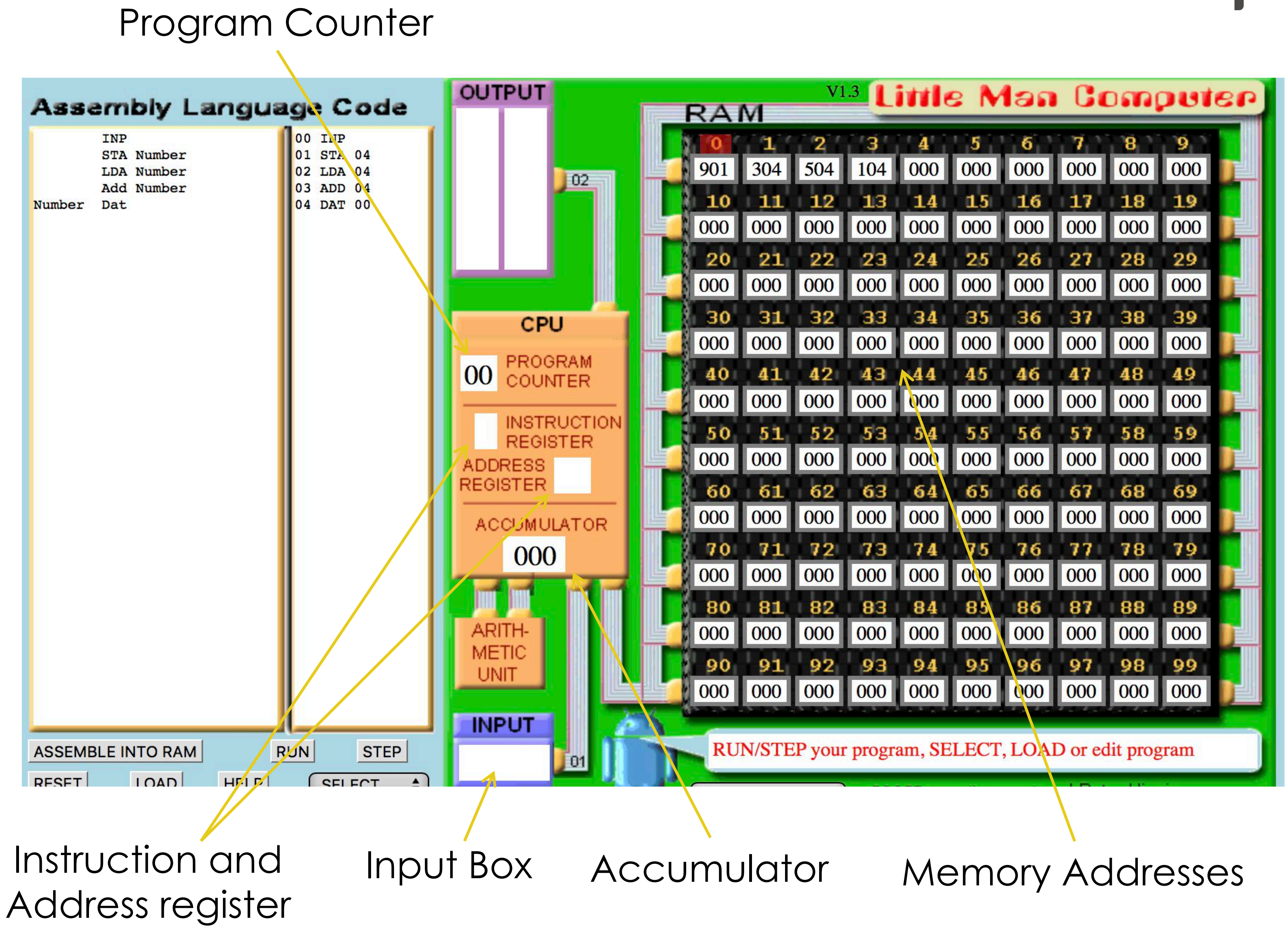
0	1	2	3	4	5	6	7	8	9
901	304	504	104	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

RUN/STEP your program, SELECT, LOAD or edit program

The LMC Environment

- **Accumulator** – This is like the active memory of the simulator. The majority of our instructions will modify the contents of the Accumulator.
- **Program Counter** – This shows the current memory location that the processor is running.
- **Instruction and Address register** – This shows which type of instruction is being used and which memory address it is being used on.
- **Memory Addresses** – These are the memory addresses which are used to store instructions and data.
- **Input Box** – This is where user inputs are stored initially before being copied to the Accumulator.

technocamps



```
         INP
         STA Number
         LDA Number
         Add Number
Number   Dat
```

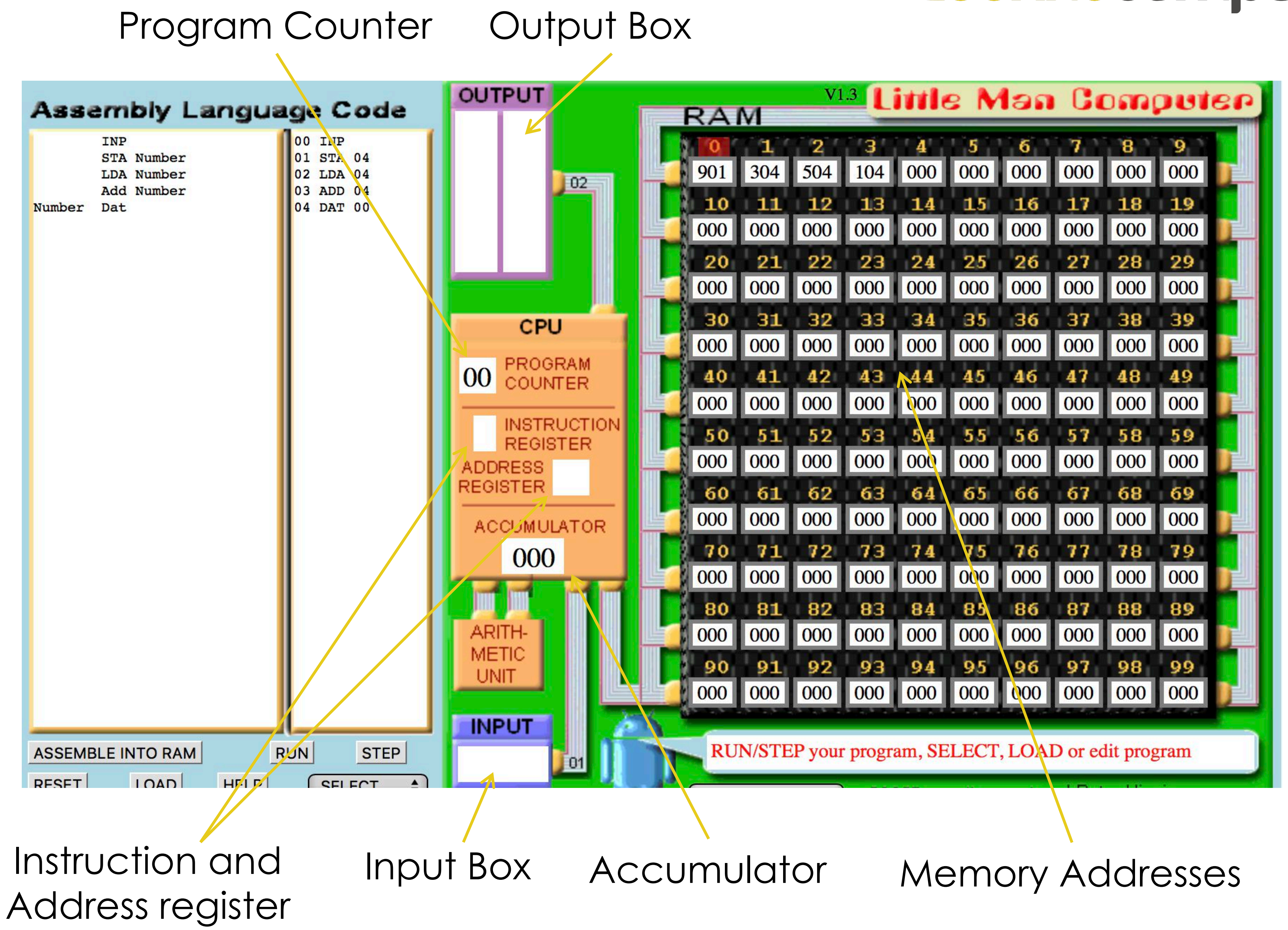
```
00 INP
01 STA 04
02 LDA 04
03 ADD 04
04 DAT 00
```

0	1	2	3	4	5	6	7	8	9
901	304	504	104	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

RUN/STEP your program, SELECT, LOAD or edit program

The LMC Environment

- **Accumulator** – This is like the active memory of the simulator. The majority of our instructions will modify the contents of the Accumulator.
- **Program Counter** – This shows the current memory location that the processor is running.
- **Instruction and Address register** – This shows which type of instruction is being used and which memory address it is being used on.
- **Memory Addresses** – These are the memory addresses which are used to store instructions and data.
- **Input Box** – This is where user inputs are stored initially before being copied to the Accumulator.
- **Output Box** – This is where a value is copied to from the Accumulator to display to the user.



Assembly Language Code

```
         INP
         STA Number
         LDA Number
         Add Number
Number   Dat
```

```
00 INP
01 STA 04
02 LDA 04
03 ADD 04
04 DAT 00
```

0	1	2	3	4	5	6	7	8	9
901	304	504	104	000	000	000	000	000	000
10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
000	000	000	000	000	000	000	000	000	000
20	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**
000	000	000	000	000	000	000	000	000	000
30	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**
000	000	000	000	000	000	000	000	000	000
40	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**
000	000	000	000	000	000	000	000	000	000
50	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**
000	000	000	000	000	000	000	000	000	000
60	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**
000	000	000	000	000	000	000	000	000	000
70	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	**79**
000	000	000	000	000	000	000	000	000	000
80	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**
000	000	000	000	000	000	000	000	000	000
90	**91**	**92**	**93**	**94**	**95**	**96**	**97**	**98**	**99**
000	000	000	000	000	000	000	000	000	000

RUN/STEP your program, SELECT, LOAD or edit program

Activity: Fill in the Blanks

LMC Instruction Set

Taking Input

Name:	Input
Mnemonic:	INP
Code:	901

Description:
The Input instruction copies the value input by the user into the Accumulator.

Next Action:
After the value has been copied, the Program Counter will move onto the next (sequential) memory location.

technocamps

Providing Output

Name:	Output
Mnemonic:	OUT
Code:	902

Description:
The Output instruction copies the value in the Accumulator into the Output Box.

Next Action:
After the value has been copied, the Program Counter will move onto the next (sequential) memory location.

Stopping the Programming

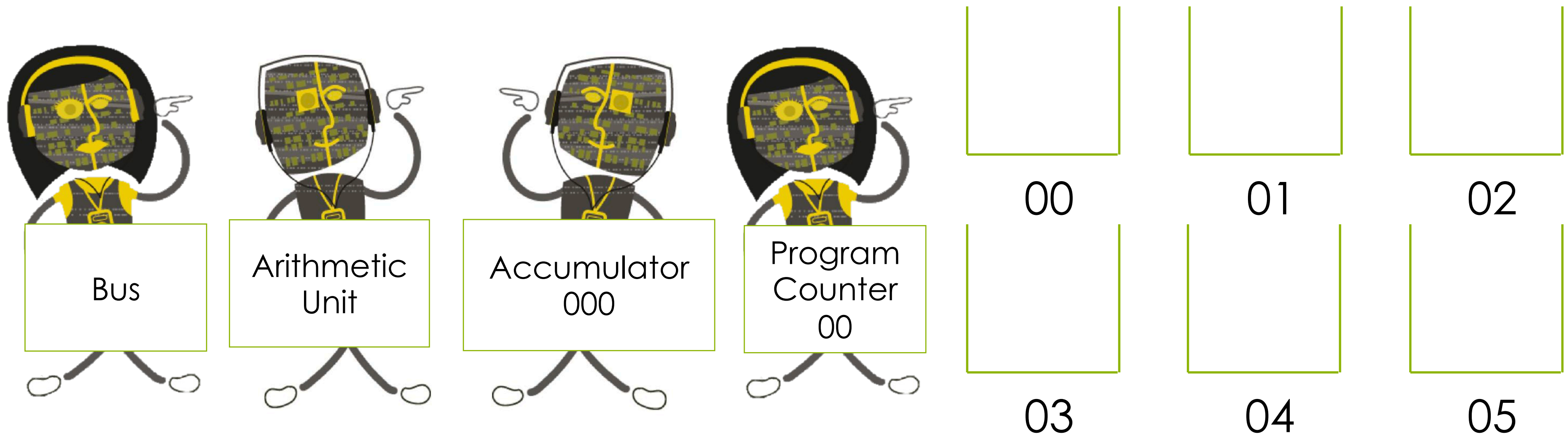
Name:	Halt
Mnemonic:	HLT
Code:	000

Description:
The Halt instruction does not affect any of the memory locations and stops the program.

Next Action:
The execution of the program will stop.

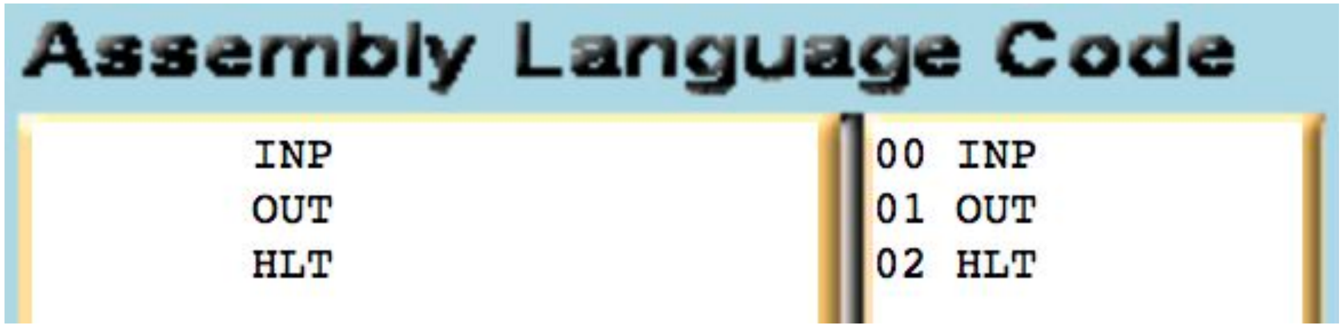
Activity: Visualising a Program Running

Using a few boxes and a few volunteers we can simulate running an assembly language program in the classroom. The rest of you will need to turn to the list of instructions in your workbooks.



The rest of the class are acting as the control unit, deciding what to do with the instructions and data.

Here's the Program



Assembly Language Code

```
INP
OUT
HLT
```

```
00 INP
01 OUT
02 HLT
```

LMC Code Summary

____	INP	____	- Input
____	OUT	____	- Output
____	HLT	____	- Halt

Storing Data

Name:	Store
Mnemonic:	STA variable
Code:	3 _ _

Description:
The Store instruction will copy the value from the Accumulator and place it in an allocated memory location referred to by the variable name given.

Next Action:
After the value has been copied, the Program Counter will move onto the next (sequential) memory location.

Retrieving Data

Name:	Load
Mnemonic:	LDA variable
Code:	5 _ _

Description:
The Load instruction will copy the value stored at the memory location, given by the variable, into the Accumulator.

Next Action:
After the value has been loaded into the Accumulator, the Program Counter will move onto the next (sequential) memory location.

Data Memory Locations

Name:	Data
Mnemonic:	variable DAT xxx
Code:	(the data)

Description:
The Data instruction will reserve a memory location to store data. This location can be referred to by the given variable name.
If you want to give the variable an initial value, replace the xxx with a value, the default is 0.

Next Action:
After the memory location has been reserved, the Program Counter will move onto the next (sequential) memory location.

Input & Print a Number

Python Code:

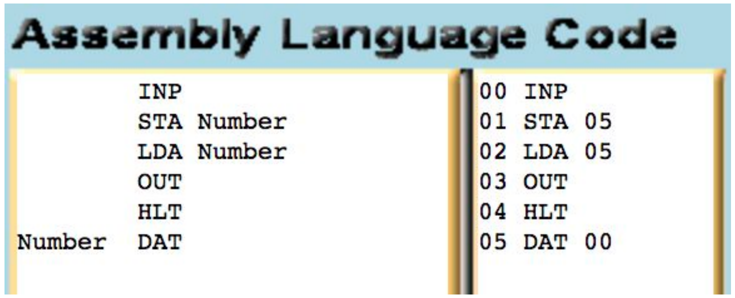
```
num = int (input("Enter a
                 number: " ))
print (num)
```

Assembly Language:

```
        INP
        STA     num
        LDA     num
        OUT
        HLT
num     DAT
```

technocamps

Activity: Running a Program


Assembly Language Code

```
        INP
        STA Number
        LDA Number
        OUT
        HLT
Number  DAT
```

```
00 INP
01 STA 05
02 LDA 05
03 OUT
04 HLT
05 DAT 00
```

Program counter	
Instruction register	
Address register	
Accumulator	

00	01	02	03	04
05	06	07	08	09
10	11	12	13	14
15	16	17	18	19

How to Write Assembly Programs

Create a program which takes in two inputs and outputs them in reverse order.

How to Write Assembly Programs

Create a program which takes in two inputs and outputs them in reverse order.

We have to output the second input before the first. So we know we are going to have to store the first number at some point.

How to Write Assembly Programs

Create a program which takes in two inputs and outputs them in reverse order.

We have to output the second input before the first. So we know we are going to have to store the first number at some point.

INP	$\longrightarrow$ get the first number
STA number1	$\longrightarrow$ store it away for later

technocamps

How to Write Assembly Programs

Create a program which takes in two inputs and outputs them in reverse order.

Next up we need to get the second number.

INP	$\longrightarrow$	get the first number
STA number1	$\longrightarrow$	store it away for later
INP	$\longrightarrow$	get the second number

How to Write Assembly Programs

Create a program which takes in two inputs and outputs them in reverse order.

We need to output the second number before the first. We could store the second number but we don’t have to.

INP	→	get the first number
STA number1	→	store it away for later
INP	→	get the second number
OUT	→	output the second number

How to Write Assembly Programs

Create a program which takes in two inputs and outputs them in reverse order.

Now we need to output the first number. But we can only output the number in the accumulator. So we need to load it first.

INP	→	get the first number
STA number1	→	store it away for later
INP	→	get the second number
OUT	→	output the second number
LDA number1	→	load the first number from the registry
OUT	→	output the first number

Activity: Storing and Loading

1. Create a program which takes and **stores** two inputs from the user and outputs the first input followed by the second input.
2. Create a program which takes and **stores** four inputs from the user and always outputs the third input.
3. Create a program which takes in three inputs and outputs them in reverse order.

LMC Code Summary

___	INP	___	- Input
___	OUT	___	- Output
___	HLT	___	- Halt
___	STA	var	- Store
___	LDA	var	- Load
var	DAT	xxx	- Data (Default for xxx is 0)

Addition

Name:	Addition
Mnemonic:	ADD variable
Code:	1 _ _

Description:
The Add instruction adds the value stored in the given memory location to the Accumulator.

Next Action:
After the value has been loaded into the Accumulator, the Program Counter will move onto the next (sequential) memory location.

Subtraction

Name:	Subtraction
Mnemonic:	SUB variable
Code:	2 _ _

Description:
The Subtraction instruction subtracts the value stored in the given memory location away from the Accumulator.

Next Action:
After the value has been loaded into the Accumulator, the Program Counter will move onto the next (sequential) memory location.

Activity: Addition and Subtraction (1)

Using a pen and paper write LMC programs to solve the problems.

1. Create a program which takes in and stores two inputs from the user and outputs the sum of them.

2. Create a program which takes in three numbers and stores them and then outputs the sum of the first two numbers with the third subtracted.

When you think you are finished, go to your computer and test it.

Activity: Addition and Subtraction (2)

In small groups, try and solve the following problems.

1. Create a program which takes in a number, doubles it and outputs the result.

2. Create a program which takes a number and multiplies it by eight.

Challenge - Create a program which takes in a number and multiplies it by forty.

LMC Code Summary

___	INP	___	- Input
___	OUT	___	- Output
___	HLT	___	- Halt
___	STA	var	- Store
___	LDA	var	- Load
var	DAT	xxx	- Data (Default for xxx is 0)
___	ADD	var	- Addition
___	SUB	var	- Subtraction

Go To (Branch Always)

Name:	Branch Always
Mnemonic:	BRA variable
Code:	6 _ _

Description:
Updates the Program Counter to the memory location referred to by the variable given.

Next Action:
After the memory location has been loaded into the program counter, that memory location will be executed.

Activity: Looping

1. Create a program which allows the user to input numbers indefinitely and outputs each number.

2. Create a program which allows the user to input numbers indefinitely and outputs the running total after each entry.

Go To (Branch If Zero)

Name:	Branch If Zero
Mnemonic:	BRZ variable
Code:	7 _ _

Description:
Updates the Program Counter to the memory location referred to by the variable given if the value in the Accumulator is **equal** to zero.

Next Action:
After the memory location has been loaded into the program counter, that memory location will be executed.

Go To (Branch If Zero or Positive)

Name:	Branch If Zero or Positive
Mnemonic:	BRP variable
Code:	8 _ _

Description:
Updates the Program Counter to the memory location referred to by the variable given if the value in the Accumulator is **zero or positive**.

Next Action:
After the memory location has been loaded into the program counter, that memory location will be executed.

Comparing Values in LMC

In Little Man Computer we do not have “if statements” like we have in Python for comparisons. The only way to branch based on a condition is to do a subtraction and then branch based on the result.

Comparing Values in LMC

In Little Man Computer we do not have “if statements” like we have in Python for comparisons. The only way to branch based on a condition is to do a subtraction and then branch based on the result.

```
          LDA two
          SUB five
          BRP outputTwo
          LDA five
          OUT
          HLT
outputTwo LDA two
          OUT
          HLT
two       DAT 2
five      DAT 5
```

Comparing Values in LMC

In Little Man Computer we do not have "if statements" like we have in Python for comparisons. The only way to branch based on a condition is to do a subtraction and then branch based on the result.

```
          LDA two
          SUB five
          BRP outputTwo

          LDA five
          OUT
          HLT

outputTwo LDA two
          OUT
          HLT

two       DAT 2
five      DAT 5
```

Split up into 4 sections of code

Comparing Values in LMC

In Little Man Computer we do not have "if statements" like we have in Python for comparisons. The only way to branch based on a condition is to do a subtraction and then branch based on the result.

```
          LDA two
          SUB five
          BRP outputTwo

          LDA five
          OUT
          HLT

outputTwo LDA two
          OUT
          HLT

two       DAT 2
five      DAT 5
```

Load 2, subtract 5 and check the result. If it is positive, jump to instruction "outputTwo"

Comparing Values in LMC

In Little Man Computer we do not have "if statements" like we have in Python for comparisons. The only way to branch based on a condition is to do a subtraction and then branch based on the result.

```
         LDA two
         SUB five
         BRP outputTwo

         LDA five
         OUT
         HLT

outputTwo LDA two
         OUT
         HLT

two      DAT 2
five     DAT 5
```

Load 2, subtract 5 and check the result. If it is positive, jump to instruction "outputTwo"

Otherwise, carry on, load 5 and output it before stopping the program.

Comparing Values in LMC

In Little Man Computer we do not have "if statements" like we have in Python for comparisons. The only way to branch based on a condition is to do a subtraction and then branch based on the result.

```
          LDA two
          SUB five
          BRP outputTwo

          LDA five
          OUT
          HLT

outputTwo LDA two
          OUT
          HLT

two       DAT 2
five      DAT 5
```

Load 2, subtract 5 and check the result. If it is positive, jump to instruction "outputTwo"

Otherwise, carry on, load 5 and output it before stopping the program.

If it was positive, load 2 and output it before stopping the program.

Activity: Conditional Branching

1. Create a program which allows the user to input two numbers and outputs the smallest number. Hint: if you do a - b and the number is positive, then a is bigger than b.
2. Create a program which allows the user to repeatedly input two numbers and checks if they're equal. Only output the number if they are equal.
3. Create a program that repeatedly takes in inputs and only outputs them if they are zero.
4. Similar to 3, create a program which outputs everything except zeroes.

Challenge - Create a program which allows the user to input two numbers and outputs the multiplication of the two numbers.

LMC Code Summary

___	INP	___	- Input
___	OUT	___	- Output
___	HLT	___	- Halt
___	STA	var	- Store
___	LDA	var	- Load
var	DAT	xxx	- Data (Default for xxx is 0)
___	ADD	var	- Addition
___	SUB	var	- Subtraction
___	BRA	var	- Branch Always
___	BRZ	var	- Branch If Zero
___	BRP	var	- Branch If Positive

Sequences (Mathematics GCSE)

In order to calculate the equation for a given sequence of numbers we must first look at the difference between them e.g.

Index term:	1 2 3 4 5 …
Number:	3 , 5 , 7 , 9 , 11 …

Sequences (Mathematics GCSE)

In order to calculate the equation for a given sequence of numbers we must first look at the difference between them e.g.

Index term: 1 2 3 4 5 …

+2

Number: 3 , 5 , 7 , 9 , 11 …

Sequences (Mathematics GCSE)

In order to calculate the equation for a given sequence of numbers we must first look at the difference between them e.g.

Index term: 1 2 3 4 5 …

Number:

$$\begin{tikzcd}3 \arrow[r, bend left, "+2"] & 5 \arrow[r, bend left, "+2"] & 7 & 9 & 11 \ldots\end{tikzcd}$$

3 , 5 , 7 , 9 , 11 …

Sequences (Mathematics GCSE)

In order to calculate the equation for a given sequence of numbers we must first look at the difference between them e.g.

Index term: 1 2 3 4 5 …

$$\begin{tikzcd} 3 \arrow[r, bend left, "+2"] & 5 \arrow[r, bend left, "+2"] & 7 \arrow[r, bend left, "+2"] & 9 & 11 \end{tikzcd}$$

Number: 3 , 5 , 7 , 9 , 11 …

Sequences (Mathematics GCSE)

In order to calculate the equation for a given sequence of numbers we must first look at the difference between them e.g.

Index term: 1 2 3 4 5 …

$$\begin{tikzcd} \arrow[r, bend left, "+2"] & \arrow[r, bend left, "+2"] & \arrow[r, bend left, "+2"] & \arrow[r, bend left, "+2"] & \end{tikzcd}$$

Number: 3 , 5 , 7 , 9 , 11 …

Sequences (Mathematics GCSE)

In order to calculate the equation for a given sequence of numbers we must first look at the difference between them e.g.

Index term: 1 2 3 4 5 …

$$\begin{tikzcd} 3 \arrow[r, bend left, "+2"] & 5 \arrow[r, bend left, "+2"] & 7 \arrow[r, bend left, "+2"] & 9 \arrow[r, bend left, "+2"] & 11 \end{tikzcd}$$

Number: 3 , 5 , 7 , 9 , 11 …

The difference between each number is +2.

So the number in front of the nth term in our equation must be 2 i.e. **2n**

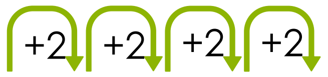
The final step is to check if we need to add or subtract from **2n**.

technocamps

Sequences (Mathematics GCSE)

In order to calculate the equation for a given sequence of numbers we must first look at the difference between them e.g.

Index term: 1 2 3 4 5 …



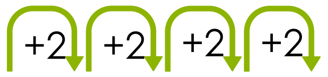
Number: 3 , 5 , 7 , 9 , 11 …

If we try inserting the index term into our nth term equation **2n** does the answer match up correctly?

Sequences (Mathematics GCSE)

In order to calculate the equation for a given sequence of numbers we must first look at the difference between them e.g.

Index term: 1 2 3 4 5 …



Number: 3 , 5 , 7 , 9 , 11 …

If we try inserting the index term into our nth term equation **2n** does the answer match up correctly? 2 x 1 = 2

Sequences (Mathematics GCSE)

In order to calculate the equation for a given sequence of numbers we must first look at the difference between them e.g.

Index term: 1 2 3 4 5 …

$$\begin{tikzcd}
3 \arrow[r, bend left, "+2"] & 5 \arrow[r, bend left, "+2"] & 7 \arrow[r, bend left, "+2"] & 9 \arrow[r, bend left, "+2"] & 11
\end{tikzcd}$$

Number: 3 , 5 , 7 , 9 , 11 …

If we try inserting the index term into our nth term equation **2n** does the answer match up correctly? 2 x 1 = 2

What should we add to correct this?

Sequences (Mathematics GCSE)

In order to calculate the equation for a given sequence of numbers we must first look at the difference between them e.g.

Index term: 1 2 3 4 5 …

$$\begin{tikzcd} 3 \arrow[r, bend left, "+2"] & 5 \arrow[r, bend left, "+2"] & 7 \arrow[r, bend left, "+2"] & 9 \arrow[r, bend left, "+2"] & 11 \end{tikzcd}$$

Number: 3 , 5 , 7 , 9 , 11 …

If we try inserting the index term into our nth term equation **2n** does the answer match up correctly? 2 x 1 = 2

What should we add to correct this? **+1**

Sequences (Mathematics GCSE)

In order to calculate the equation for a given sequence of numbers we must first look at the difference between them e.g.

$$\begin{array}{ll}
\text{Index term:} & 1 \quad 2 \quad 3 \quad 4 \quad 5 \ \ldots \\
 & \quad +2 \quad +2 \quad +2 \quad +2 \\
\text{Number:} & 3 , 5 , 7 , 9 , 11 \ \ldots
\end{array}$$

If we try inserting the index term into our nth term equation **2n** does the answer match up correctly? 2 x 1 = 2

What should we add to correct this? **+1**

Therefore our equation is: **2n + 1**

Another Example

The first 5 terms of a sequence are:

-3, -1, 1, 3, 5

What is the difference between each term?

What is the equation so far?

Do we need to add/subtract something to get the right values?

What is the correct equation?

Another Example

The first 5 terms of a sequence are:

-3, -1, 1, 3, 5

What is the difference between each term? +2

What is the equation so far?

Do we need to add/subtract something to get the right values?

What is the correct equation?

technocamps

Another Example

The first 5 terms of a sequence are:

-3, -1, 1, 3, 5

What is the difference between each term? +2

What is the equation so far? 2n

Do we need to add/subtract something to get the right values?

What is the correct equation?

Another Example

The first 5 terms of a sequence are:

-3, -1, 1, 3, 5

What is the difference between each term? +2

What is the equation so far? 2n

Do we need to add/subtract something to get the right values? -5

What is the correct equation?

Another Example

The first 5 terms of a sequence are:

-3, -1, 1, 3, 5

What is the difference between each term? +2

What is the equation so far? 2n

Do we need to add/subtract something to get the right values? -5

What is the correct equation? 2n - 5

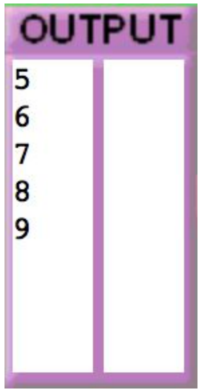
Activity: Sequences

For the following sequences:

a. Write out the nth term equation.
b. Calculate the 20th term in the sequence

1. 7, 8, 9, 10, 11 …
2. 3, 6, 9, 12, 15 …
3. 12, 17, 22, 27, 32 …
4. -6, -2, 2, 6, 10 …
5. 3, -3, -9, -15, -21 …

6. a. Write out the first 5 terms of the sequence given by 3n - 7.
   b. Calculate the 15th term of the sequence.



```
5
6
7
8
9
```

Activity: LMC Example

Now we are going to implement this nth term equation in LMC to produce the first 5 terms in the sequence: 5, 6, 7, 8, 9.

Using the space in your workbooks discuss with a partner and try to write down the steps you would need to implement it. Think about:

- What is the nth term equation?
- Would you need to use a loop?
- What other variables would you need?
- You will need to be adding or subtracting by 1, how could you implement this?

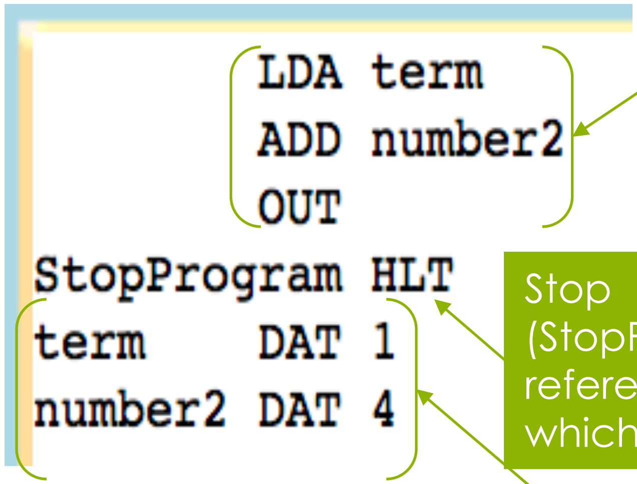
The First Value

To get the first result we need to load the first index term = 1 , add 4 to it and then output it.

We always add 4 in our nth term equation so should store 4 as a variable called number2.

We also need to define a variable so we know which index term we are inserting into our equation.

Load 1 and add 4 to it. Then Output the Value 5.



```
            LDA term
            ADD number2
            OUT
StopProgram HLT
term        DAT 1
number2     DAT 4
```

Stop the program. (StopProgram is a reference to the Halt which will be useful later).

Define Variables: term = 1
number2 = 4

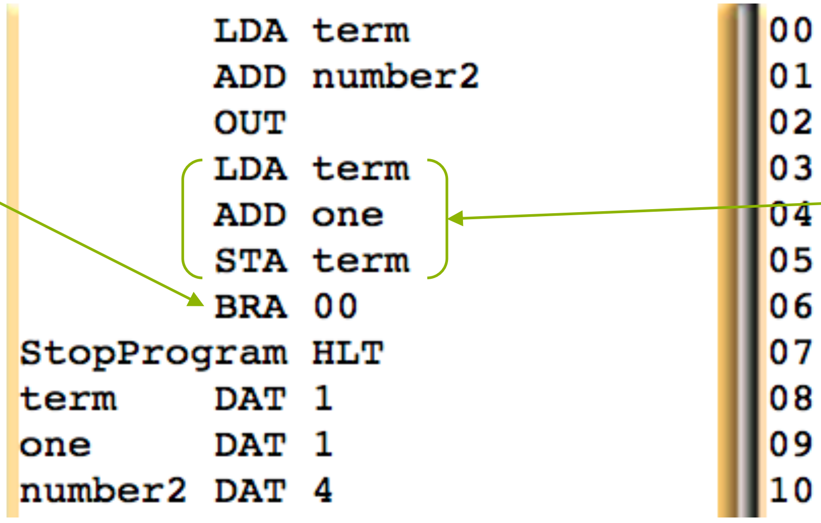
technocamps

Looping for More Values

Now we need to add one to the index term variable before we calculate the next number in our sequence.

To do this we define a variable called one which we add to the index term variable.

We use a loop to repeat the previous calculations and output each new number in the sequence.



```
            LDA  term          00
            ADD  number2       01
            OUT                02
            LDA  term          03
            ADD  one           04
            STA  term          05
            BRA  00            06
StopProgram HLT                07
term        DAT  1             08
one         DAT  1             09
number2     DAT  4             10
```

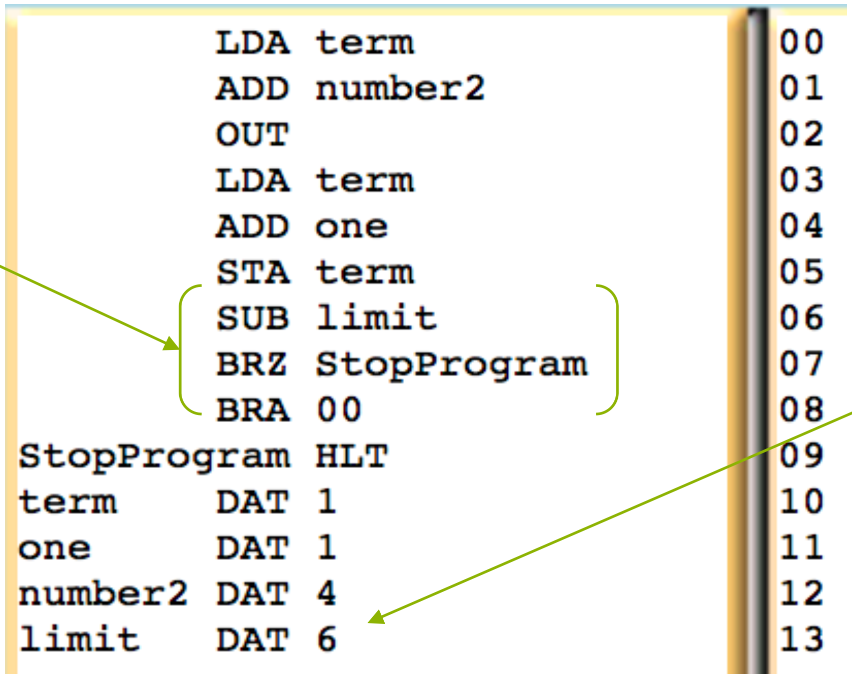
Loop back to the first line (00) Always.

Increase the current term by one and store it again.

technocamps

Only Outputting the First 5 Values

If we want to stop the loop after 5 values have been output we need to compare our term variable to a limit. Once our term reaches the same value as the limit, we halt the program.



```
            LDA term           00
            ADD number2        01
            OUT                02
            LDA term           03
            ADD one            04
            STA term           05
            SUB limit          06
            BRZ StopProgram    07
            BRA 00             08
StopProgram HLT                09
term        DAT 1              10
one         DAT 1              11
number2     DAT 4              12
limit       DAT 6              13
```

Check if the term and limit are the same, if so jump to the HLT instruction.

Note: The variable limit is set to 6. Is this correct?

Yes, we increment the loop counter before checking how many times we have looped.

technocamps

Activity: Creating Your Own Sequences

You can use this code as a starting point for creating your own sequences. What would we change to make the sequence n + 8 for example?

In your workbooks, answer the questions and try running the code in LMC to see if you're correct.

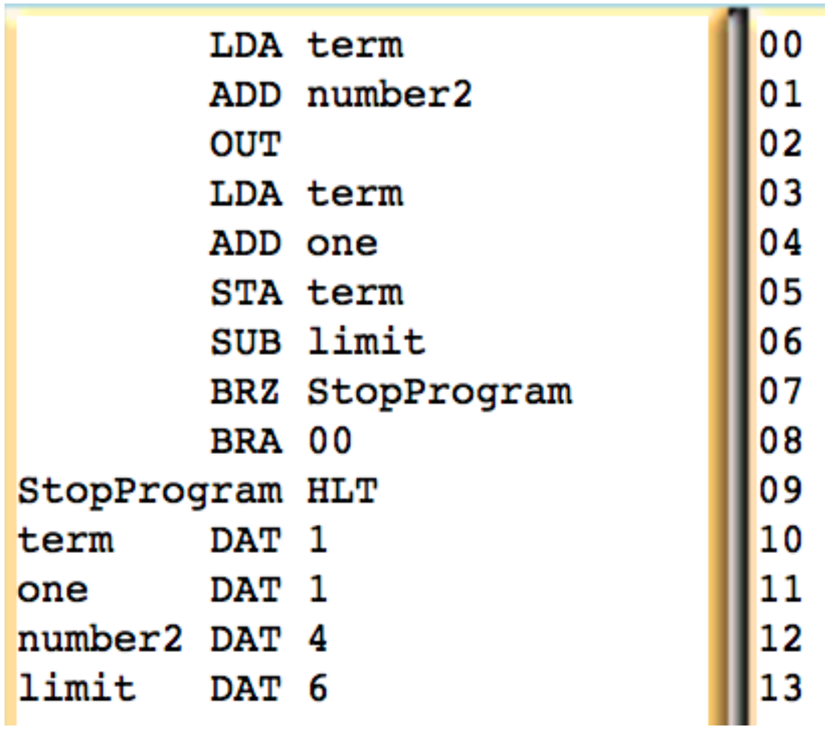
n – 7

2n + 4

2n – 6

3n + 8

8n - 3



```
            LDA term           00
            ADD number2        01
            OUT                02
            LDA term           03
            ADD one            04
            STA term           05
            SUB limit          06
            BRZ StopProgram    07
            BRA 00             08
StopProgram HLT                09
term        DAT 1              10
one         DAT 1              11
number2     DAT 4              12
limit       DAT 6              13
```

Activity: Advanced LMC

1. Create a program which takes in inputs and outputs the positive value, i.e. if it's negative, you output the positive, -3 would output 3.
2. Create a program which takes an input, outputs that value and then counts down and outputs every value until it reaches 0 (or counts up to 0 if value is negative).
3. Create a program which takes two inputs and checks if they have the same sign (both positive or both negative). If they have the same sign output a zero, otherwise output a 1.
4. Create a program which takes two inputs and returns the remainder if you divided the first by the second. (Don't worry about negative numbers, but dividing zero by a number and diving a number by zero should be considered.)

technocamps

Activity: Very Advanced LMC

Create a program which takes in an input and outputs all of the numbers in the Fibonacci sequence up to that input number.
The Fibonacci sequence is 1, 1, 2, 3, 5, 8, 13, 21
You can set one variable to 1 at the beginning. No cheating!

Note the Fibonacci sequence is made by adding the previous number to the current one, starting with 1:

1
0+1=1
1+1=2
2+1=3
3+2=5