Unit 5: Computer architecture

1. CPU & Fetch-execute cycle

Learning objectives

- 1.3.2 - show understanding of the basic Von Neumann model for a computer system and the stored program concept (program instructions and data are stored in main memory and instructions are fetched and executed one after another)
- 1.3.2 - describe the stages of the fetch-execute cycle, including the use of registers and buses

Introduction

"A computer processor does moronically simple things — it moves a byte from memory to register, adds a byte to another byte, moves the result back to memory. The only reason anything substantial gets completed is that these operations occur very quickly. To quote Robert Noyce, 'After you become reconciled to the nanosecond, computer operations are conceptually fairly simple.'" - Giorgos Petkakis

- Read: Sections 4.4 and 4.5 of iGCSE Computer Science by Watson & Williams

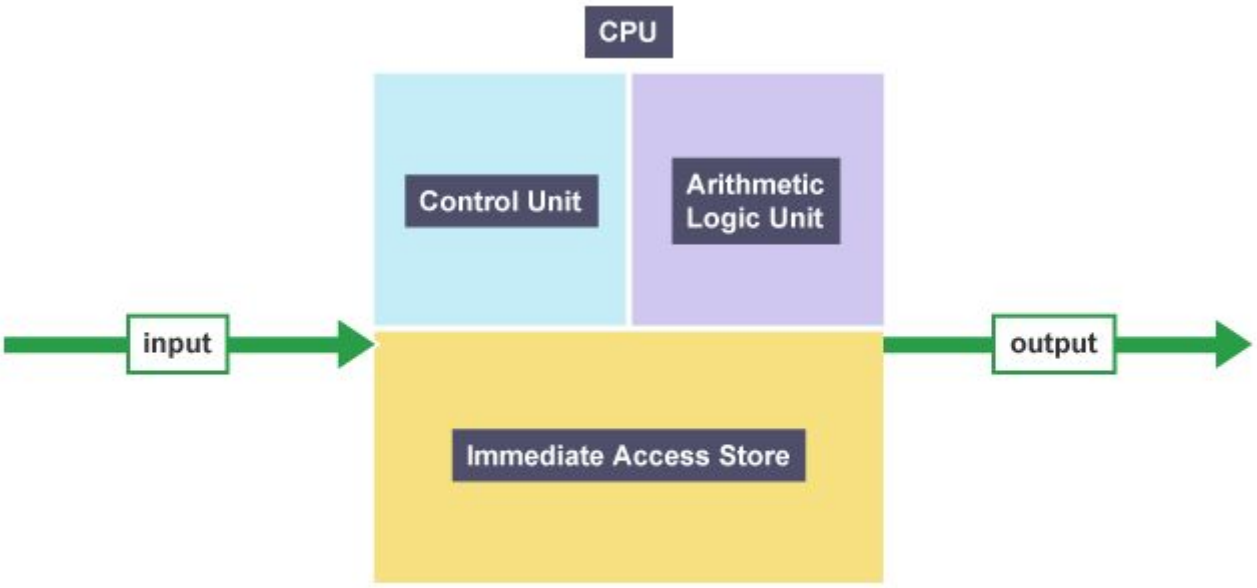
Discussion & learning items

Take notes on these two videos to introduce this topic:

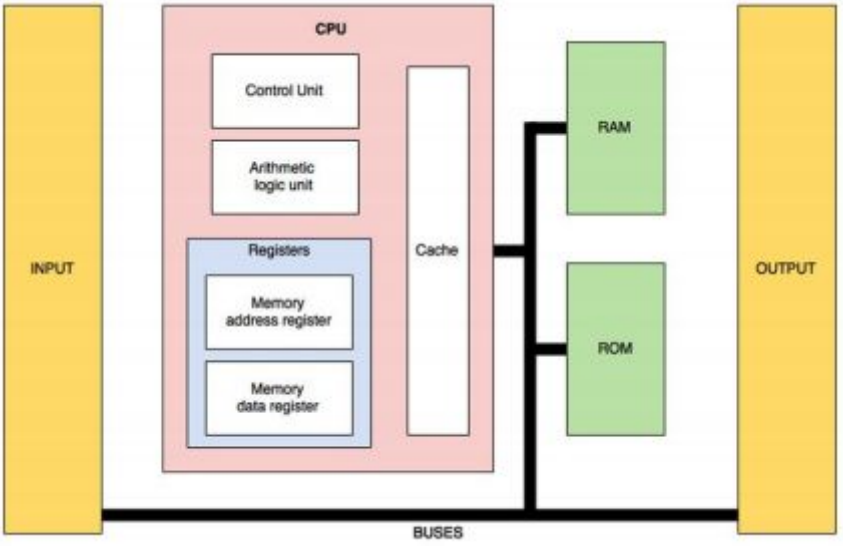
- Watch Tom Scott's excellent explainer into the Fetch-decode-execute cycle (9:03m).
- Watch Fetch decode execute cycle in more detail (7:54)

There are a couple of ways the Von Newmann architecture of a CPU might be illustrated.

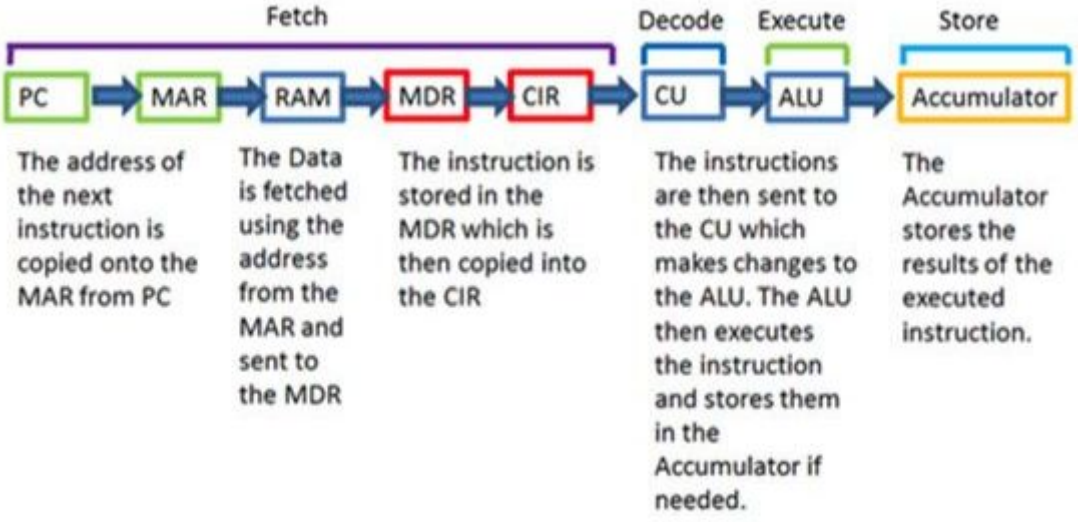
- Simplified representation



- More detailed representation



The fetch/decode/execute cycle can be shown as:



The **Control Unit (CU)**

- Coordinates the timing of the different components and the flow of data in the CPU
- Is responsible for fetching and decoding instructions and managing their execution on the processor
- Controls and monitors the hardware attached to the computer
- Tells the ALU, the computer's memory and the hardware devices how to respond to an instruction

The **Arithmetic and Logic Unit (ALU)**

- is where the CPU performs the arithmetic and logic operations. Data is passed to the ALU to allow the required calculations to be carried out. The results of any calculations are then sent to be stored in a register.
- There are two parts to the ALU:
    - The arithmetic part, which deals with calculations, e.g. 1 + 2 = 3
    - The logic part, which deals with any logical comparisons, such as the comparison of two numbers, e.g. 2 > 1
- Typically, the ALU includes a register that it uses to store intermediate results of calculations. This register is known as the **accumulator** (because it is accumulating the results).

The **Immediate Access Store (IAS)**

- This is where the CPU holds all the data and programs it is currently using
- The IAS is often referred to as the **registers**, which are a very small amount of storage. In a 64-bit processor, each register will store just 64 bits

Fetching an instruction:

- The **Program Counter (PC)** copies the address of the next instruction it contains into the **Memory Address Register (MAR)**.
- The **MAR** places the address to be used on to the address bus.
- The instruction on the data bus is loaded into the **Memory Data Register (MDR)**.
- The MDR copies the instruction into the **Instruction Register (IR)** sometimes known as the **Current Instruction Register (CIR)**.

Executing an instruction:

- The instruction in the IR is carried out (executed) by the CPU. This will generally involve the ALU carrying out a calculation. Data is moved in and out of the ALU via the MAR/MDR.
- Now that the CPU is executing an instruction, the **Program Counter** can now be reset to point to the next instruction.

Sourced from: https://www.bbc.co.uk/bitesize/guides/ztyd3k7/revision/1

Key terms you should be comfortable with:

- Von Neumann model for a computer system
- Control Unit (CU),
- Arithmetic logic unit (ALU)
- Registers
- Accumulator (ACC)
- Program counter (PC)
- Current instruction register (CIR)
- Memory address register (MAR) & Memory data register (MDR)
- Random access memory (RAM)
- Bus

Questions

Complete the sentences, using the list given. Not all items in the list need to be used.

- accumulator (ACC)
- address bus
- arithmetic logic unit (ALU)
- control unit (CU)
- data bus
- executed
- fetches
- immediate access store (IAS)
- memory address register (MAR)
- memory data register (MDR)
- program counter (PC)
- saved
- transmits

The central processing unit (CPU) .....
the data and instructions needed and stores them in the
..... to wait to be processed.
The ..... holds the address of the next
instruction. This address is sent to the ..... .
The data from this address is sent to the ..... .
The instruction can then be decoded and ..... .
Any calculations that are carried out on the data are done by the
..... . During calculations, the data is temporarily
held in a register called the ..... .
[8] (June, 2018, p12, q6)

The fetch-execute cycle make use of registers.
(a) Describe the role of the Program Counter (PC).
.....
.....
.....
.....[2]
(b) Describe the role of the Memory Data Register (MDR).
.....
.....
.....
.....[2]
(Winter 2018, p13, q11)

Additional practice:

- https://quizlet.com/9917997/test
- https://www.bbc.co.uk/bitesize/guides/zbfny4j/test

2. Memory & storage media

Learning objectives

1.3.5 Memory, storage devices and media

- show understanding of the difference between: primary, secondary and off-line storage and provide examples of each, such as: primary: Read Only Memory (ROM) and Random Access Memory (RAM) secondary: hard disk drive (HDD) and Solid State Drive (SSD); off-line: Digital Versatile Disc (DVD), Compact Disc (CD), Blu-ray disc, USB flash memory and removable HDD
- describe the principles of operation of a range of types of storage device and media including magnetic, optical and solid state
- describe how these principles are applied to currently available storage solutions, such as SSDs, HDDs, USB flash memory, DVDs, CDs and Blu-ray discs
- ~~calculate the storage requirement of a file~~

Introduction

Read: Sections 6.4 of iGCSE Computer Science by Watson & Williams

Discussion & learning items

Random access memory (RAM)

- What does it mean for memory to be volatile?
- The textbook states "In reality the RAM never runs out of memory". This is actually an incorrect statement. Modern operating systems are able to give the appearance of not running out of memory. The following sentences "sort of" explain this but not very clearly.
    - Class discussion: explain the concept of virtual memory.

DRAM vs SRAM

- Remember back to when we looked at logic circuits. The "d-latch" logic circuit we looked at is an example of how a "flip-flop" can be created.
- For an online demo of a d-latch in action, visit https://logic.ly/demo

Read only memory (ROM)

- What is EEPROM? How is it still considered "ROM"?
- Role of ROM vs RAM?

Hard disk drives (HDD)

Solid state drives (SSD)

- Be aware of the difference between FLASH and EEPROM based SSDs. Which type do you have?

Optical drives (CD, DVD, DVD-RAM, Blue-ray)

- Distinguish between these different disk types
- Have your own copy of table 6.1, but add to it the DVD-RAM disk type.

USB flash memory

Questions

The supermarket uses secondary storage and off-line storage to store data about its stock.
Explain what is meant by (a) secondary storage and (b) off-line storage. [4] (June 2018, p11, q8d)

Identify three similarities between CDs and DVDs. [3] (June 2018, p13, q11)

Five storage devices or media are listed. Show whether each storage device or media is an example of primary, secondary or off-line storage.

- External HDD
- RAM
- Internal SSD
- ROM
- DVD

[5] (Winter 2018, p11, q5a)

Explain how data is written to optical storage media. [4] (Winter 2018, p11, q5b)

A sports events company uses a digital camera attached to a drone (small flying system), to video their events from the sky. The video is stored as it is captured, on a device that is attached to the drone.

- Identify the most suitable type of storage to store the video
  Optical OR Magnetic OR Solid state [1]
- Explain the reasons for your choice .

.....
.....
.....
..... [2]
(Winter 2018, p11, q5c)

A computer uses RAM and ROM to store data. Indicate which statements apply to each type of memory:

	RAM	ROM
Stores the programs and data that are currently in use		
Used to boot up the computer when power is turned on		
Contents are retained when power is turned off		

[3] (Winter 2018, p12, q2a)

Which storage category includes both RAM and ROM.

- Primary
- Secondary
- Off-line

[1] (Winter 2018, p12, q2b)

Explain what is meant by off-line storage.
[2] (Winter 2018, p12, q2b)

Sarah stores data electronically. Describe three methods that she could use to avoid loss of stored data.
Method 1 .....
.....
.....
Method 2 .....
.....
.....
Method 3 .....
.....
.....
[6] (Winter 2018, p13, q6)

Describe two differences between Read Only Memory (ROM) and Random Access Memory (RAM).
Difference 1 .....
.....
.....
.....
Difference 2 .....
.....
.....
.....
[4] (Winter 2018, p13, q9)

State which types of storage device or media would be most suitable for these scenarios.
For each device or media, justify your choice.
(a) Creating a backup of 150 GB of data.
.....
Justification .....
..... [2]
(b) Storing applications on a tablet device.
.....
Justification .....
..... [2]
(c) Storing a 1200 MB high-definition promotional movie about a new car. The movie is to be given to people who are interested in buying a new car.
.....
Justification .....
..... [2]
(Winter 2018, p13, q13)

3. Memory & storage file formats

Learning objectives

- 1.1.3 - show understanding that sound (music), pictures, video, text and numbers are stored in different formats
- 1.1.3 - show understanding of the concept of Musical Instrument Digital Interface (MIDI) files, JPEG files, MP3 and MP4 files
- 1.3.5 - calculate the storage requirement of a file

Introduction

Read: Sections 6.1, 6.2, 6.5 of iGCSE Computer Science by Watson & Williams

Discussion & learning items

MIDI file format - Read 6.2.1 then answer these questions

- In what way is MIDI different to an audio recording?
- In what way might MIDI be considered similar to a programming language?
- One MIDI interface can control up to how many musical instruments?
- One example of when you might use MIDI instead of MP3 audio formats?

MP3 & MP4 file format - Read 6.2.2 then answer these questions

- What is the file compression algorithm used by MP3? What sounds does it remove?
- What is the consequence of using a LOSSY format?
- If an MP3 has a bit rate of 200 kilobits/second, how many megabytes would a 3 minute (180 second) recording be?
- What is the key difference between MP3 and MP4?
- Complete activity 6.1

JPEG file format - Read 6.2.3 then answer these questions

- If a RAW Bitmap photo is taken on an 8 megapixel camera, how many Megabytes of storage would this consume?
- What factors does JPEG compression rely on to maintain the quality of an image?
- Complete activity 6.2

TEXT file format

- Why is formatting of a text file important for programs to correctly interpret the content of text files?
- What is different about the compression used with text files, and why does it matter?
- What is a common program on your computer that can be used to read/write text files?

Exercises

Create a simple Python "file storage calculator" that will prompt the user for pixel dimensions for a photo, plus the bit-colour depth. It should output the resulting file size in kilobytes and megabytes (to one decimal place). Remember 8 bits = 1 byte, 1024 bytes = 1 KB, 1024 KB = 1 MB.

For example:

```
File size calculator!
Pixels wide? 640
Pixels high? 480
Bits of color depth: 8
Working...
The resulting bitmap will be 921.6 KB or 0.9 MB
```

Submit via the provided Google Classroom assignment

Questions

An image is to be stored electronically.
The image is 256 pixels high by 200 pixels wide with a 16-bit colour depth.
Calculate the file size of the image. You must show all of your working.

File size ..... kB [3]
(June 2018, p13, q12)

Jamelia wants to store an image file. The image has an 8-bit resolution and is 150 pixels by 100 pixels in size. Calculate the file size of the image. Give your answer in kilobytes (kB). Show all of your working.
File size .....kB [3]
(Winter 2018, p11, q1b)

Explain the difference between a Musical Instrument Digital Interface (MIDI) file and a MP3 file.
.....
.....
[4] (Winter 2018, p13, q12)

4. Compression

Learning objectives

1.1.3

- show understanding of the principles of data compression (lossless and lossy) applied to music/video, photos and text files

Introduction

Read: Sections 6.3 of iGCSE Computer Science by Watson & Williams

Exercises

Python has a built in compression library you can easily experiment with.

```
# Exercise experimenting with lossless compression
import zlib                     # Python's lossless compression module

test = """Programmers are in a race with the Universe to create bigger and better
idiot-proof programs, while the Universe is trying to create bigger and better
idiots.  So far the Universe is winning. (Rich Cook)"""
test = test.encode('utf-8')     # Convert to utf-8 bytes string

print('=== Original ===')
print('Text:   ' , test)
print('Hex:    ' , test.hex() )
print('Length: ' , len(test))

compressed = zlib.compress( test )

print('=== Compressed ===')
print('Text:   ' , compressed)
print('Hex:    ' , compressed.hex() )
print('Length: ' , len(compressed))

decompressed = zlib.decompress( compressed )

print('=== Decompressed ===')
print('Text:   ' , decompressed)
print('Hex:    ' , decompressed.hex() )
print('Length: ' , len(decompressed))

ratio = int(100 * len(compressed) / len(test))

print('=== Summary ===')
print(f'Compressed to {ratio}% of original')
```

- Test the lossless compression with several different text samples
- For larger text samples (for example 2000 words or more), does compression become better or worse?

```
# Exercise experimenting with lossy compression
from PIL import Image           # Python image library... pip install pillow
import requests                 # Requests is a commonly used library to download
items from the web

# Download a photo
url = "https://apod.nasa.gov/apod/image/1908/5D4_5485seeley_1067.jpg"
response = requests.get(url)

# Check the download was successful, error message if not
if response.status_code != 200:
    print('Problem with download. Status code: ', response.status_code)
    exit()

# Save the file
with open('original.jpg', 'wb') as f:
    f.write(response.content)

# Open the image via PIL
image = Image.open("original.jpg")

# Create several versions of decreasing compression quality
for n in range(100,0,-10):    # From 100 until 0, increments of -10
    image.save(f'quality-{quality}.jpg', optimize=True, quality=n)
```

- What are the parts of the image that are the first to deteriorate?
- At what minimum level of quality was the image "acceptable" to you?
- How significant was the difference in file size for each level of quality? What would make the best "happy median" in your view?
- What would be the features of a photo that might determine if you selected a higher or lower quality?

Questions

Michele wants to email a file to Elsa. The file is too large so it must be compressed.
(a) Name two types of compression that Michele could use.
Compression type 1 .....
Compression type 2 ..... [2]

(b) The file Michele is sending contains the source code for a large computer program. Identify which type of compression would be most suitable for Michele to use.
Explain your choice.
Compression type .....
Explanation .....
.....
.....

.....
.....
..... [4]
(June 2018, p11, q4)

Large files can be compressed to reduce their file size. Two types of compression that can be used are lossy and lossless. Explain how a file is compressed using lossless compression.
.....
.....
.....
..... [3]
(Winter 2018, p11, q1c)

The following are four different file formats that use compression.
State whether each file format uses lossy or lossless compression.

- JPEG
- MP3
- MP4
- ZIP

[4] (Winter 2018, p11, q1d)

5, 6. High & low level languages; compilers; assembler

Learning objectives

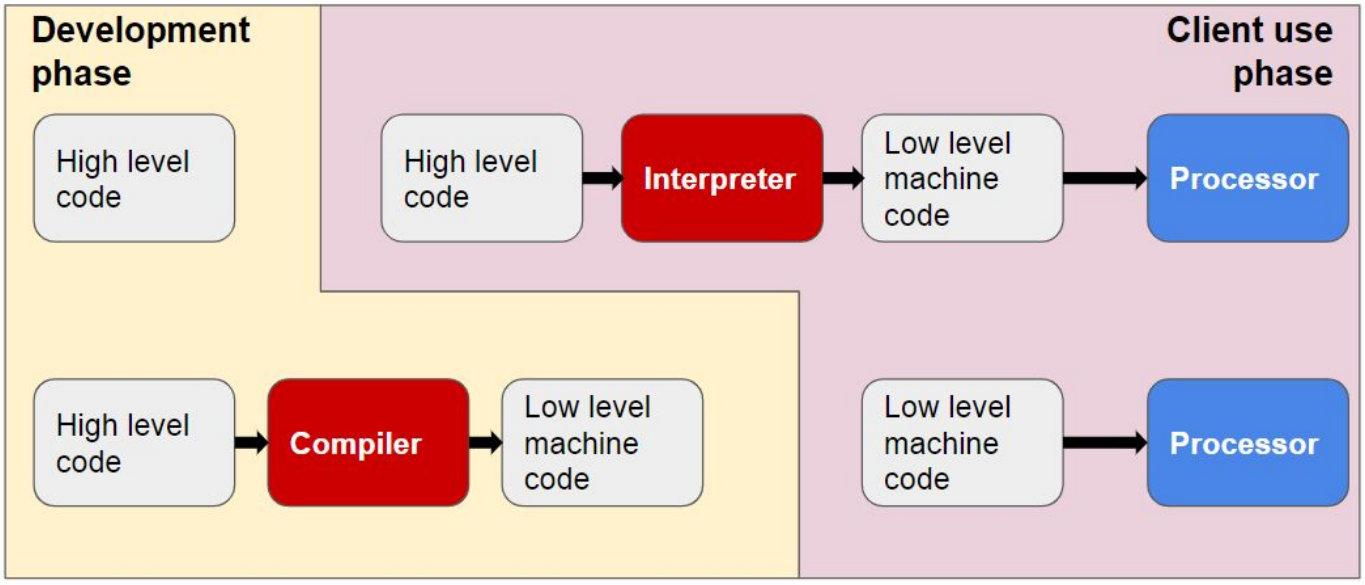
1.3.7 High- and low-level languages and their translators

- show understanding of the need for both high-level and low-level languages
- show understanding of the need for compilers when translating programs written in a high-level language
- show understanding of the use of interpreters with high-level language programs
- show understanding of the need for assemblers when translating programs written in assembly language

Introduction

Read: Chapter 7 of iGCSE Computer Science by Watson & Williams

Discussion & learning items



As we know, a computer program is a list of instructions that enable a computer to perform a specific task. The languages that we use to write our computer programs in are generally split into two categories, depending on the task and the hardware being used. These are high level languages and low level languages.

High Level Languages

When we think about computer programmers, we are probably thinking about people who write in high-level languages.

High level languages are written in a form that is close to our human language, enabling the programmer to just focus on the problem being solved.

No particular knowledge of the hardware is needed as high level languages create programs that are portable and not tied to a particular computer or microchip.

These programmer friendly languages are called ‘high level’ as they are far removed from the machine code instructions understood by the computer.

Examples include: C++, Java, Pascal, Python, Visual Basic.

Advantages

- Easier to modify as it uses English like statements
- Easier/faster to write code as it uses English like statements
- Easier to debug during development due to English like statements
- Portable code – not designed to run on just one type of machine

Modern high level languages would come with features such as:

- Variables & constants
- Types - integers, floating point numbers, characters, strings, booleans
- Operators - add, subtract, multiply, divide, modulus, concatenate etc
- Loops - for, while, repeat until
- Branching - if, if else, else
- Collections - saving and retrieving multiple items to one variable identifier
- Arrays/lists - a subtype of collection
- Sub routines, functions
- Error or exception handling

Low Level Languages

Low level languages are used to write programs that relate to the specific architecture and hardware of a particular type of computer. They are *closer* to the native language of a computer (binary), making them harder for programmers to understand. Low level languages include Assembly Language and Machine Code.

Assembly Language

- Few programmers write programs in low level assembly language, but it is still used for developing code for specialist hardware, such as device drivers.
- It is easily distinguishable from a high level language as it contains few recognisable human words but plenty of mnemonic code.
- Typical assembly language opcodes include: add, subtract, load, compare, branch, store

Advantages

- Can make use of special hardware or special machine-dependent instructions (e.g. on the specific chip)
- Translated program requires less memory
- Write code that can be executed faster
- Total control over the code
- Can work directly on memory locations

Machine Code

- Programmers rarely write in machine code (binary) as it is difficult to understand.

Hello world examples

Hello world in C

```c
#include <stdio.h>
int main() {
   printf("Hello, World!");
   return 0;
}
```

Hello world in x86 assembler

```
; ------------------------------------------------------------------------
; Writes "Hello, World" to the console. Runs on 64-bit Linux only.
; To assemble and run:
;     nasm -felf64 hello.asm && ld hello.o && ./a.out
; ------------------------------------------------------------------------

          global    _start

          section   .text
_start:   mov       rax, 1                  ; system call for write
          mov       rdi, 1                  ; file handle 1 is stdout
          mov       rsi, message            ; address of string to output
          mov       rdx, 13                 ; number of bytes
          syscall                           ; invoke operating system to write
          mov       rax, 60                 ; system call for exit
          xor       rdi, rdi                ; exit code 0
          syscall                           ; invoke operating system to exit

          section   .data
message:  db        "Hello, World", 10      ; note the newline at the end
```

Hello world in machine code

```
b8    21 0a 00 00   #moving "!\n" into eax
a3    0c 10 00 06   #moving eax into first memory location
b8    6f 72 6c 64   #moving "orld" into eax
a3    08 10 00 06   #moving eax into next memory location
b8    6f 2c 20 57   #moving "o, W" into eax
a3    04 10 00 06   #moving eax into next memory location
b8    48 65 6c 6c   #moving "Hell" into eax
a3    00 10 00 06   #moving eax into next memory location
b9    00 10 00 06   #moving pointer to start of memory location into ecx
ba    10 00 00 00   #moving string size into edx
bb    01 00 00 00   #moving "stdout" number to ebx
b8    04 00 00 00   #moving "print out" syscall number to eax
```

```
cd    80            #calling the linux kernel to execute our print to stdout
b8    01 00 00 00   #moving "sys_exit" call number to eax
cd    80            #executing it via linux sys_call
```

Exercises

Every CPU has its own type of assembler, it is not universally portable like a high language.

There is an online simulator that uses a slightly simplified assembler we can experiment with. It is based on an 8-bit cpu and has 256 bytes of memory.

- http://schweigi.github.io/assembler-simulator/

All instructions (code) and variables (data) need to fit inside the memory. For simplicity every instruction (and operand) is 1 byte. Therefore a MOV instruction will use 3 bytes of memory. The simulator provides a console output which is memory mapped from 0xE8 to 0xFF. Memory mapped means that every value written to this memory block is visible on the console.

A detailed look at the instructions available in the simulator are available here:
http://schweigi.github.io/assembler-simulator/instruction-set.html

Play the walk through of the Hello World example.

What else can you make it do?

As an example, the following is my implementation of a simple counter, that outputs 0 through 9. Can you modify it? A couple of suggestions:

- Turn it into a countdown
- Make it work for values greater than 10 (more complicated than it might initially think)

```
; Counts from 0 to 9

      JMP start      ; Jump to `start`
                     ; (so we don't "execute" our variables as if instructions)
i:    DB 0           ; Declare a variable
until: DB 10         ; Declare a variable

start:
loop:
      ; PART 1 - PRINT VALUE OF i
      MOV D, 232     ; Address of output buffer
      MOV A, i       ; Look up the address of var `i`
      MOV C, [A]     ; Look up value at the address in A
      ADD C, 48      ; Convert from integer to character by adding 48
      MOV [D], C     ; Copy value in C to address of output buffer

      ; PART 2 - INCREMENT i
      MOV A, i       ; Look up the address of var `i`
      MOV B, [A]     ; Look up value at the address in A
      INC B          ; Increment the value
      MOV [A], B     ; Copy value back to the address referred to by A

      ; PART 3 - LOOP CONTINUE?
      MOV A, i       ; Look up the address of var `i`
      MOV B, until   ; Look up the address of var `until`
      MOV C, [A]     ; Look up value at the address in A
      MOV D, [B]     ; Look up value at the address in B
      CMP C, D       ; Do the two values compare?
      JNZ loop       ; If the values do not compare, jump to `loop`
      HLT            ; Halt the program
```

Question: Why does adding 48 convert the integer to a character? (make sure I discuss this in the lesson, don't let me forget)

(don't worry if you don't get anywhere with this… it is waaaaaay beyond the course and just included for some nerdy fun. It took me a solid hour just to make this above code sample)

Questions

Translators, such as a compiler and an interpreter, are used when writing and running computer programs. Describe how a compiler and an interpreter translates a computer program.

Compiler .....

.....

.....

.....

Interpreter .....

.....

.....

..... [6]

(June 2018, p11, q7)

Dimitri is writing a computer program in a high-level language. He needs to send just the machine code for the program to his friend, electronically. It is important that the program is executed as quickly as possible. Identify which translator will be most suitable for Dimitri to use. Explain your choice.

Type of translator .....

Explanation .....

.....

.....

.....

[4] (June 2018, p12, q8)

Three types of translators are assemblers, compilers and interpreters.

Tick (✓) the appropriate boxes to show which statements apply to each type of translator.

Statement	Assembler	Compiler	Interpreter
Translates high-level language into machine code			
Provides error diagnostics			
Translates whole program to object code in one operation			
Translates and executes one line of code at a time			

[3] (June 2018, p13, q9)

Explain two reasons why a computer programmer may choose to write a program in a high level language, rather than a low-level language.

Reason 1 .....

.....

.....

.....

Reason 2 .....

.....

.....

.....

[4] (Winter 2018, p12, q6b)

Three examples of computer code are given in the table. Indicate whether the computer code is a high level language, assembly language or machine code.

10110111 11001100 01011100	FOR X = 1 TO 10 PRINT X NEXT X	INP X STA X LDA Y
Category of code:	Category of code:	Category of code:

[3] (Winter 2018, p12, q6c)

David is writing a program using a high-level language. The program will be published and sold for profit.

(a) David uses an interpreter when creating the computer program. State three features of an interpreter.

Feature 1 .....

.....

Feature 2 .....

.....

Feature 3 .....

..... [3]

(b) David compiles the program when he has completed it. Explain two benefits of compiling the program.

Benefit 1 .....

.....

.....

.....

Benefit 2 .....

.....

.....

.....[4]

(Winter 2018, p13, q7)

Further practice & resources

There is a more x86-like assembler emulator that works online here (inspired by the one we use above but has more x86 registers).

- https://kobzol.github.io/davis/

If you want to see what full x86 assembler looks like, try this emulator

- https://carlosrafaelgn.com.br/asm86/

7. Operating systems & interrupts

Learning objectives

1.3.6 Operating systems

- describe the purpose of an operating system (Candidates will be required to understand the purpose and function of an operating system and why it is needed. They will not be required to understand how operating systems work.)
- show understanding of the need for interrupts

Introduction

Read: Sections 4.1, 4.2, 4.3 of iGCSE Computer Science by Watson & Williams

Discussion & learning items

The operating system's overall role is to manage the resources within the system (CPU time, RAM, HDD access, networking etc) amongst the competing demands of the applications. Includes prioritisation and security. The operating system is the link between system hardware and application software

In more detail, various roles of the operating system include (be able to provide a definition or basic description of each)

- Human computer interface (HCI)
- Multi tasking
- Multi programming
- Batch processing
- Error handling
- Loading and running of application software
- Management of user accounts
- Providing file utilities (copy, save, sort, delete files etc)
- Processor management
- Memory management
- Real-time processing
- Interrupt handling
- Security
- Input and output devices

Common operating systems and some key differences

- Windows, OSX, Android, iOS

Interrupts

- An interrupt is a signal sent to the processor that interrupts the current process. It may be generated by a hardware device or a software program. A hardware interrupt is often created by an input device such as a mouse or keyboard. - *From* *https://techterms.com/definition/interrupt*

Buffers: While in the textbook, it is not in the syllabus

Exercises

An example of an interrupt in a program is when an exception event occurs. It will interrupt whatever Python is doing and trigger your exception handling code. We can actually use this to have some fun modifying default behaviour. For instance, typically in Python pressing CTRL C on the keyboard will abort your program.

We can stop that from happening by handling the keyboardinterrupt exception!

```
import sys
from datetime import datetime
import time

interrupts = 0
while interrupts < 3:
    try:
        while True: # Infinite loop we can't escape from
            print('The time is ', datetime.now().strftime("%H:%M:%S"))
            time.sleep(0.2)
    except KeyboardInterrupt:
        print("You interrupted me. That wasn't very nice. You can't stop me tho.")
        interrupts += 1
print("Fine, you win, I'll stop")
```

Questions

One of the roles of an operating system is to deal with interrupts.
(a) Explain the term interrupt.
.....
.....
.....
..... [2]
(b) Identify three devices that make use of interrupts.
Device 1 .....
Device 2 .....
Device 3 .....
[3]
(June 2018, p13, q6)

Personal computers (PCs) use an operating system.
Explain why this type of computer needs an operating system. [4]
(Winter 2018, p11, q10)

Describe the purpose of an interrupt in a computer system. [4] (Winter 2018, p12, q8)

The diagram shows **five** operating system functions and **five** descriptions.

Draw a line between each operating system function and its description.

Function	Description
Interrupt	Many processes appear to run simultaneously
Utility	Data are temporarily held in a buffer waiting for an output device to access it
Memory management	A signal that causes the operating system to take a specified action
Spooling	A program that performs a specific task required for the operation of a computer system
Multitasking	A process of assigning blocks of memory to programs running in a computer

[4]

(May 2017, p12, q6)

8. Quiz