Computer Science @NCP

- The learning curve from Year 11 to Year 12 is a big one.
- These resources aim to help you bridge the gap and improve some of the independent study skills you will need to develop for A Level.
- Please ensure the completed workbook is available at your first Yr12 lesson.
- Read each slide carefully and enjoy - this is also a great opportunity for you to demonstrate your skills and interests!

1. Who Are You?

Compulsory – must do!

Maths grade	
English grade	
Computer Science grade	
Average GCSE point score (if known)	

In this task you get to tell me a little bit about yourself. Who are you and what do you enjoy about Computer Science?

Name:

1. What are your main interests in school? What subjects and clubs do you enjoy and why?

2. What are your interests outside of school?

3. Why did you choose Computer Science?

4. What are your plans beyond New College? What would like to get out of studying Computer Science?

2. The Course

Optional – strongly recommended!

Exercise: Research the following

How is the OCR A level Computer Science qualification structured?

1a. What is the name of the first paper?

1b. How long is the exam?

1c. How much of the course is it worth?

2a. What is the name of the second paper?

2b. How long is the exam?

2c. How much of the course is it worth?

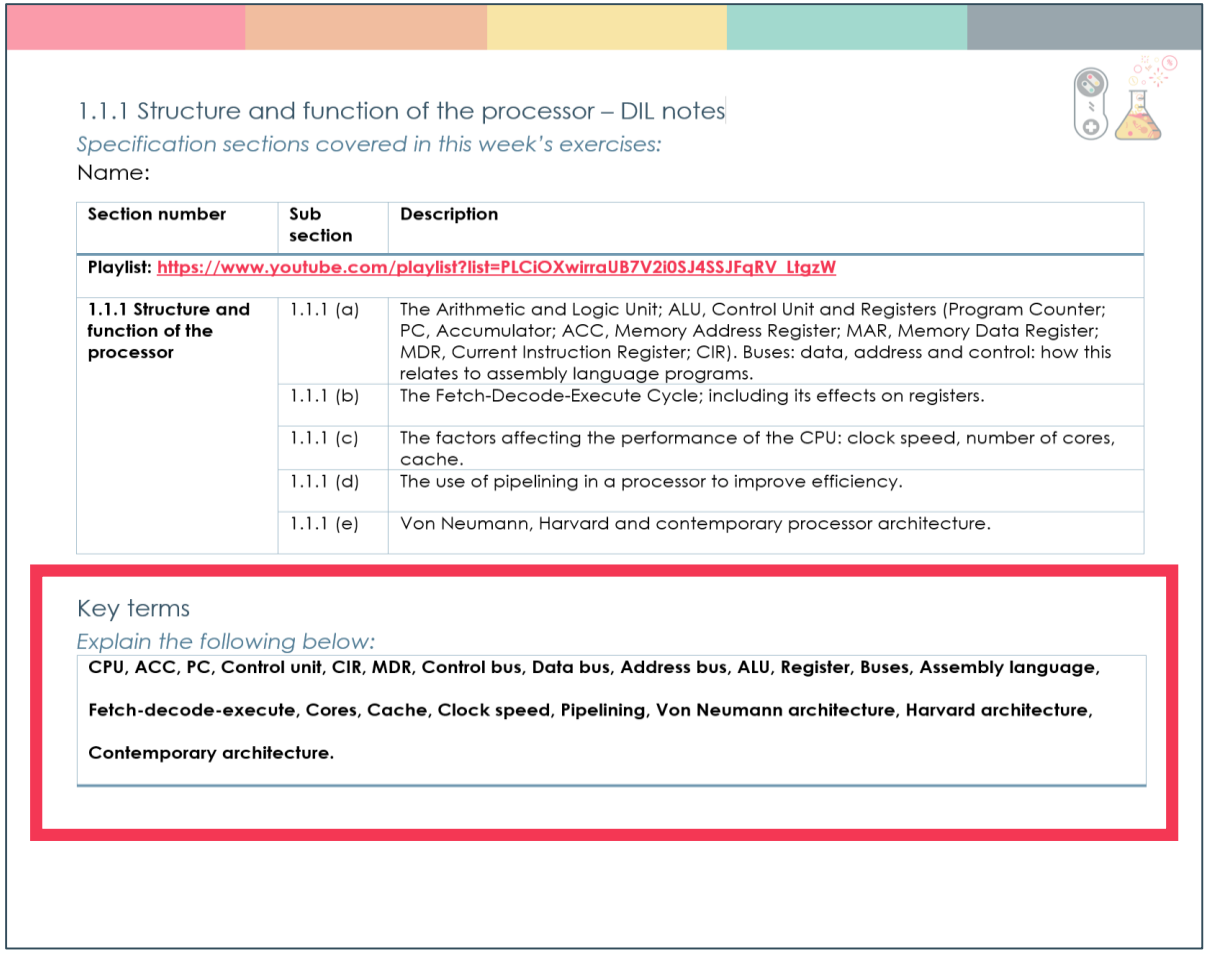
3a. What is the final part of the course?

c. How much of the course is it worth?

3. Flipped Learning

Compulsory – must do!

At New College you are expected to preview the following weeks learning. We do this using a method of note taking called Cornell notes, we will look at them on the following pages. On the first page of each Cornell note booklet we begin by listing key terms for a topic, and your task is to find a definition for each key term, as shown below.



1.1.1 Structure and function of the processor – DIL notes

Specification sections covered in this week's exercises:

Name:

Section number	Sub section	Description
Playlist: https://www.youtube.com/playlist?list=PLCiOXwirraUB7V2i0SJ4SSJFqRV_LtgzW		
1.1.1 Structure and function of the processor	1.1.1 (a)	The Arithmetic and Logic Unit; ALU, Control Unit and Registers (Program Counter; PC, Accumulator; ACC, Memory Address Register; MAR, Memory Data Register; MDR, Current Instruction Register; CIR). Buses: data, address and control: how this relates to assembly language programs.
	1.1.1 (b)	The Fetch-Decode-Execute Cycle; including its effects on registers.
	1.1.1 (c)	The factors affecting the performance of the CPU: clock speed, number of cores, cache.
	1.1.1 (d)	The use of pipelining in a processor to improve efficiency.
	1.1.1 (e)	Von Neumann, Harvard and contemporary processor architecture.

Key terms

Explain the following below:

CPU, ACC, PC, Control unit, CIR, MDR, Control bus, Data bus, Address bus, ALU, Register, Buses, Assembly language, Fetch-decode-execute, Cores, Cache, Clock speed, Pipelining, Von Neumann architecture, Harvard architecture, Contemporary architecture.

Pick 10 of the key terms highlighted in the red box on the left (apart from CPU) and find a definition for each. Complete the table on the next slide…

If the word is an acronym like CPU, do not simply write what the letters stand for, also write what a CPU does.

For example:

Chosen key term	Definition
e.g. CPU	This stands for Central Processing Unit. This is the main electronic circuitry that executes the instructions from a computer program. It is often referred to as the brain of a computer.

3. Flipped Learning

Compulsory – must do!

Exercise: Write your chosen ten key terms and their definition below.

Chosen key term	Definition
e.g. CPU	This stands for Central Processing Unit. This is the main electronic circuitry that executes the instructions from a computer program. It is often referred to as the brain of a computer.
1.	
2.	
3.	
4.	
5.	
6.	
7.	
8.	
9.	
10.	

4. Flipped Learning

Compulsory – must do!

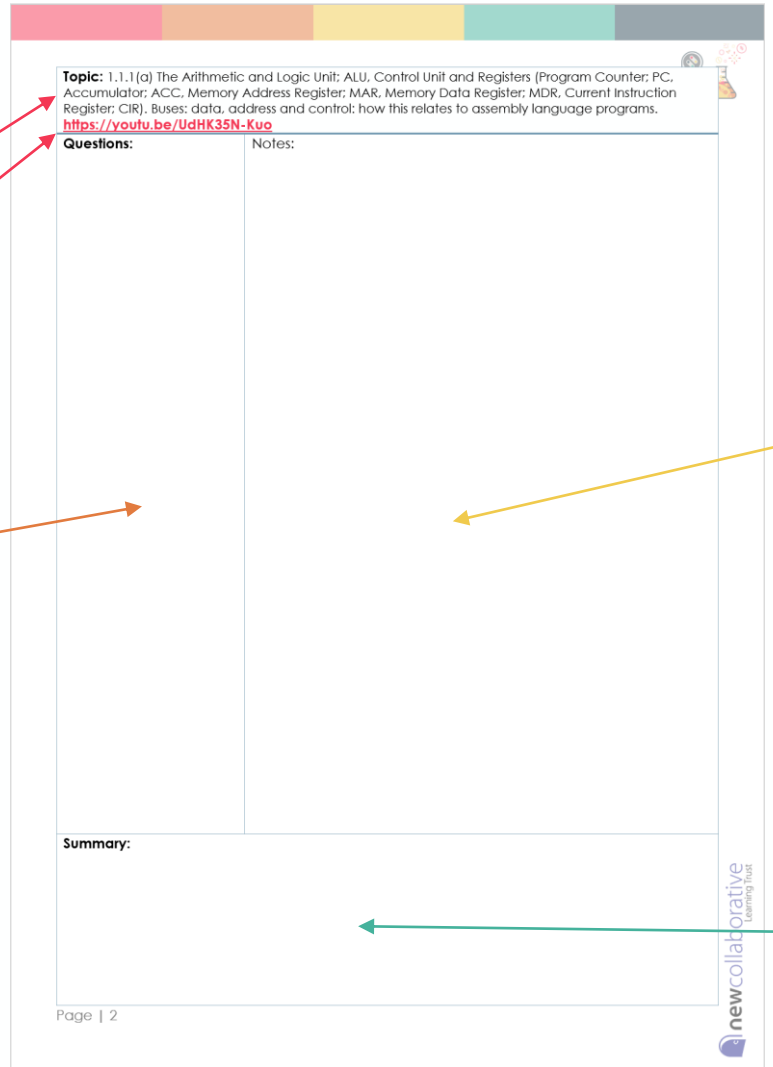
Cornell Note Taking

Once you have completed the key terms, you will have to complete the Cornell notes for each specification point using online videos. The structure of a Cornell notes page is shown below:

1. The point from the specification document that this page links to.
2. The link to the YouTube video that you need to make notes about.

6. The questions section should include questions that you think you could be asked about in the exam.

7. It should also include questions that you have for your teacher to clarify.



3. The main notes section should include information about the theory discussed in the video. They should be **hand written in pen** or **hand written electronically**, ***never typed***. The only exception is if you would normally use a word processor in an examination.

4. Any diagrams shown should also be included, to refer back to when revising.

5. *If you need more space you should duplicate the page, or make the notes section larger so you go onto two pages, don't think you have to stick to the page given.*

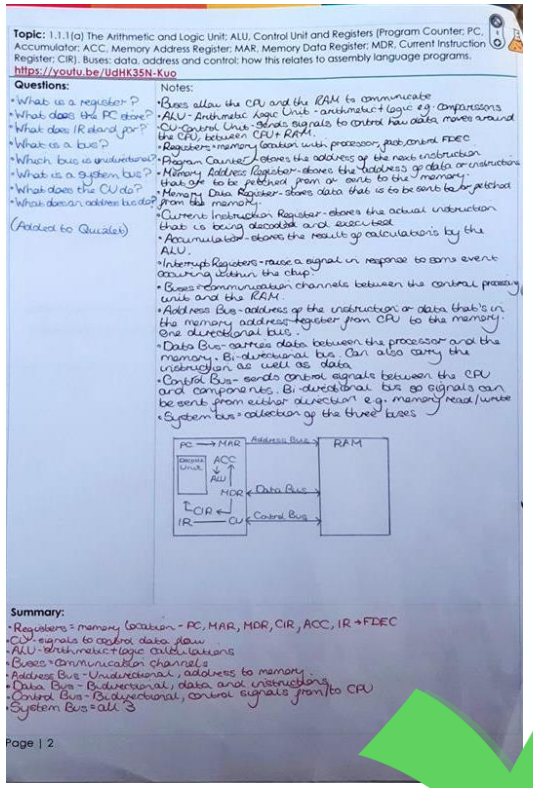
9. The summary should be a short explanation of what this page is about. It is usually 5 – 6 sentences long. Writing short, to the point explanations is a key skill for the exam.

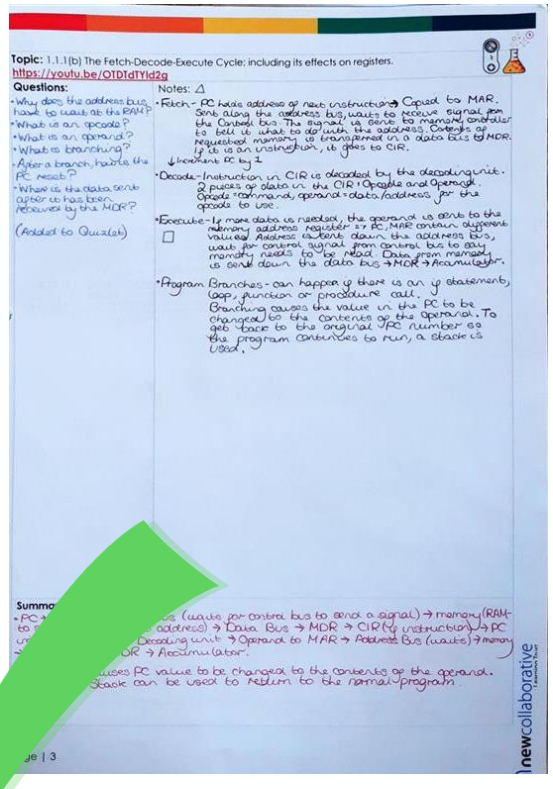
4. Flipped Learning

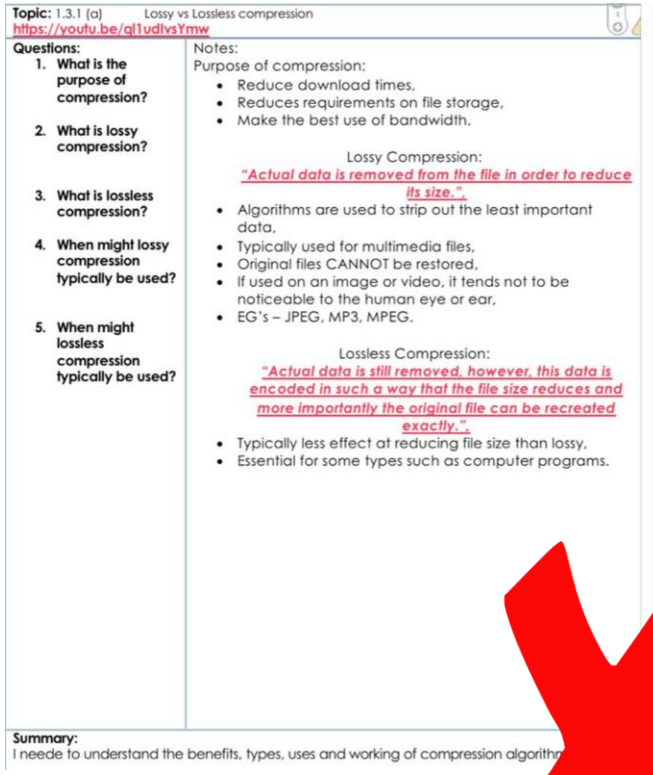
Compulsory – must do!

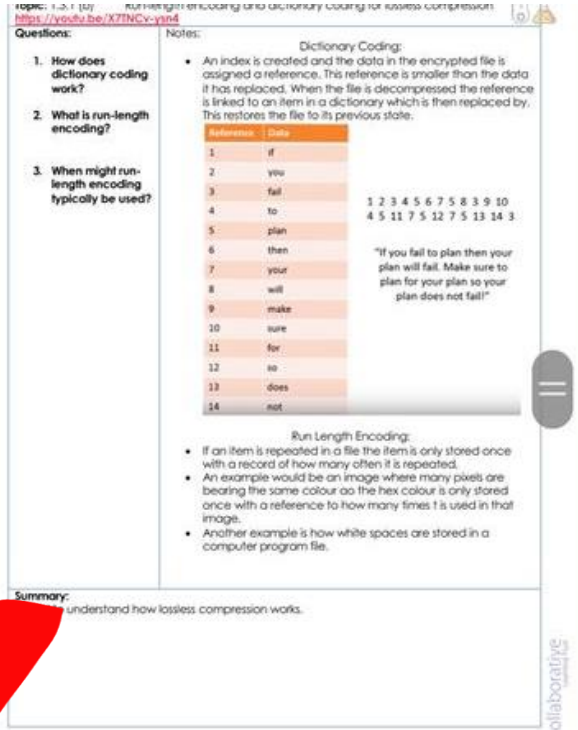
Cornell Note Taking

We ask you to handwrite your Cornell Notes for two reasons; Firstly, your exams will be hand written, so you need to get used to writing quickly, but clearly and to the point. Secondly, copying and pasting doesn't involve any mental processing, so you're less likely to remember what you've included!







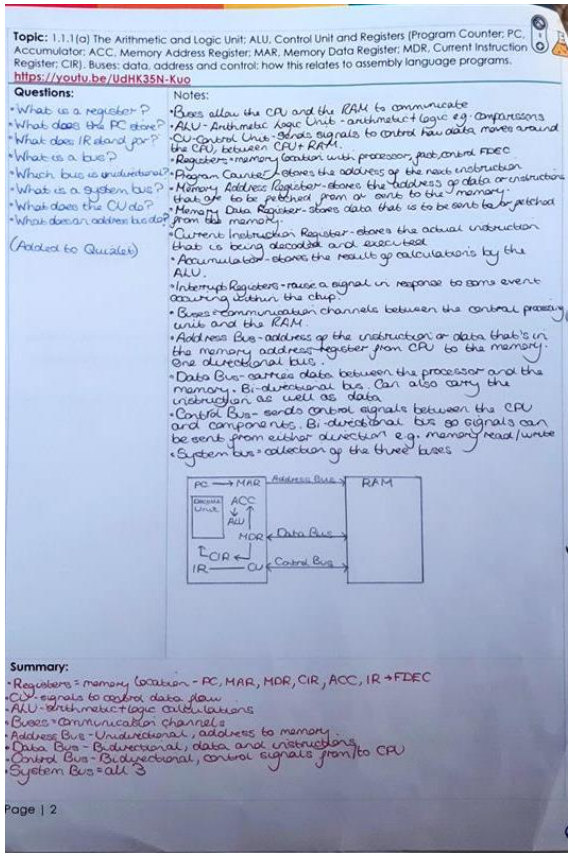


4. Flipped Learning

Compulsory – must do!

Cornell Note Taking

Here are some examples – you can see how different students interpret the videos differently and how they used diagrams, colour, bullets and written explanations to aid their explanation.



Topic: 1.1.1(a) The Arithmetic and Logic Unit; ALU, Control Unit and Registers (Program Counter; PC, Accumulator; ACC, Memory Address Register; MAR, Memory Data Register; MDR, Current Instruction Register; CIR). Buses: data, address and control: how this relates to assembly language programs.
https://youtu.be/UdHK35N-Kuo

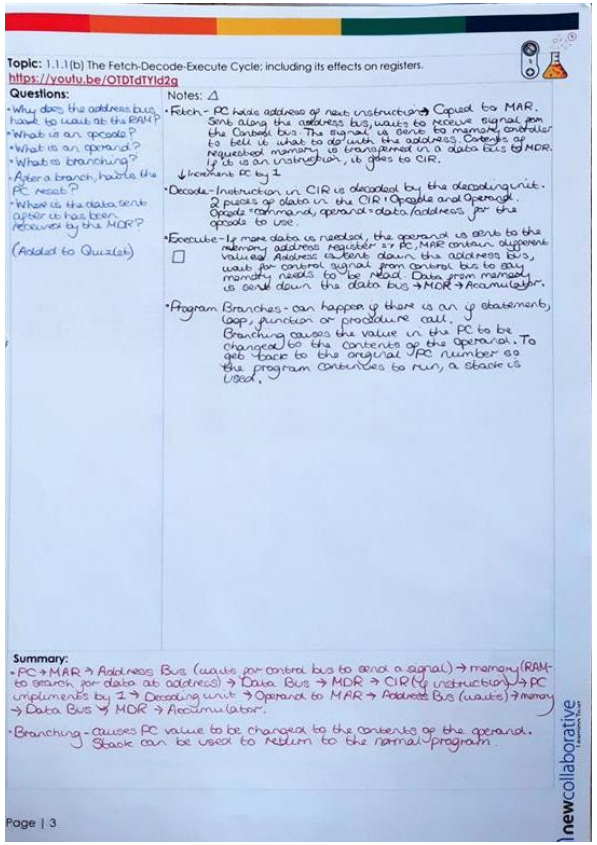
Questions:
- What is a register?
- What does the PC store?
- What does IR stand for?
- What is a bus?
- Which bus is unidirectional?
- What is a system bus?
- What does the CU do?
- What does an address bus do?

(Added to Quizlet)

Notes:
- Buses allow the CPU and the RAM to communicate
- ALU - Arithmetic Logic Unit - arithmetic + logic eg. comparisons
- CU - Control Unit - sends signals to control how data moves around the CPU, between CPU + RAM
- Registers = memory location with processor, fast, costly FDEC
- Program Counter - stores the address of the next instruction
- Memory Address Register - stores the address of data or instructions that are to be fetched from or sent to the memory
- Memory Data Register - stores data that is to be sent to be fetched from the memory
- Current Instruction Register - stores the actual instruction that is being decoded and executed
- Accumulator - stores the results of calculations by the ALU
- Interrupt Registers - raise a signal in response to some event occurring within the chip
- Buses = communication channels between the control processing unit and the RAM
- Address Bus - address of the instruction or data that's in the memory address register from CPU to the memory. One directional bus.
- Data Bus - carries data between the processor and the memory. Bi-directional bus. Can also carry the instruction as well as data.
- Control Bus - sends control signals between the CPU and components. Bi-directional bus so signals can be sent from either direction e.g. memory read/write
- System bus = collection of the three buses

Summary:
- Registers = memory location - PC, MAR, MDR, CIR, ACC, IR → FDEC
- CU - signals to control data flow
- ALU - arithmetic + logic calculations
- Buses = communication channels
- Address Bus - Unidirectional, addresses to memory
- Data Bus - Bi-directional, data and instructions
- Control Bus - Bi-directional, control signals from/to CPU
- System Bus = all 3

Page | 2



Topic: 1.1.1(b) The Fetch-Decode-Execute Cycle; including its effects on registers.
https://youtu.be/OTDTdTYId2g

Questions:
- Why does the address bus have to wait at the RAM?
- What is an opcode?
- What is an operand?
- What is branching?
- After a branch, has the PC reset?
- What is the data sent after it has been received by the MDR?

(Added to Quizlet)

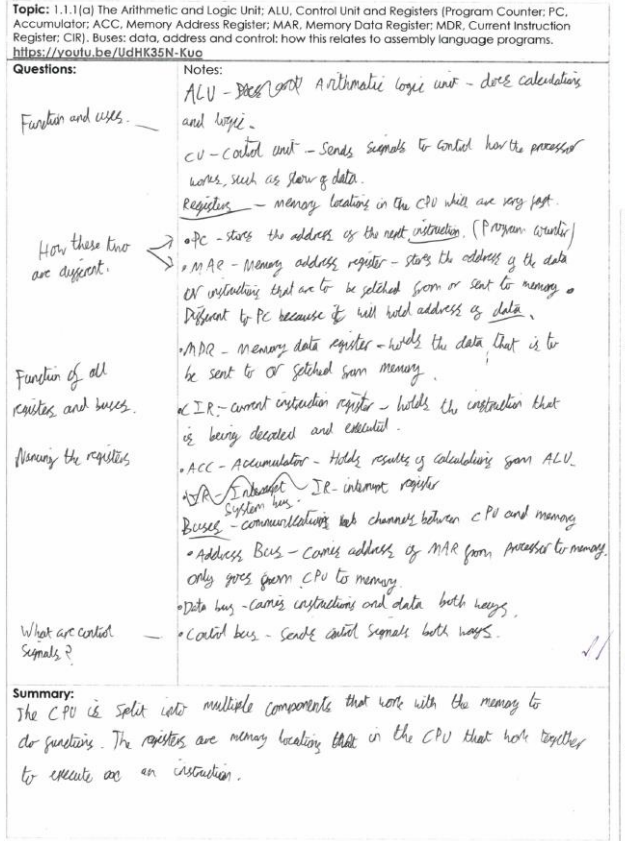
Notes: Δ
- Fetch - PC holds address of next instruction → Copied to MAR. Sent along the address bus, waits to receive signal on the Control bus. The signal is sent to memory controller to tell it what to do with the address. Contents of requested memory is transferred in a data bus to MDR. If it is an instruction, it goes to CIR.
  ↓ increases PC by 1
- Decode - Instruction in CIR is decoded by the decoding unit. 2 pieces of data in the CIR: Opcode and Operand. Opcode = command, operand = data/address for the opcode to use.
- Execute - If more data is needed, the operand is sent to the memory address register → PC, MAR contain different values. Address is sent down the address bus, wait for control signal from control bus to say memory needs to be read. Data from memory is sent down the data bus → MDR → Accumulator.
- Program Branches - can happen if there is an if statement, loop, function or procedure call. Branching causes the value in the PC to be changed to the contents of the operand. To get back to the original PC number so the program continues to run, a stack is used.

Summary:
- PC → MAR → Address Bus (waits for control bus to send a signal) → memory (RAM to search for data at address) → Data Bus → MDR → CIR (if instruction) → PC increments by 1 → Decoding unit → Operand to MAR → Address Bus (waits) → memory → Data Bus → MDR → Accumulator.
- Branching - causes PC value to be changed to the contents of the operand. Stack can be used to return to the normal program.

Page | 3

newcollaborative

Student 1



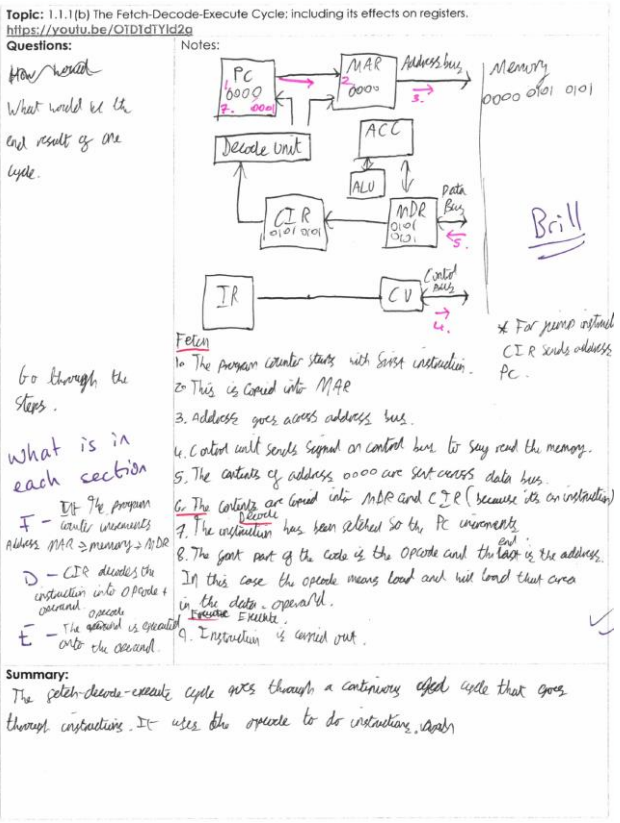
Topic: 1.1.1(a) The Arithmetic and Logic Unit; ALU, Control Unit and Registers (Program Counter; PC, Accumulator; ACC, Memory Address Register; MAR, Memory Data Register; MDR, Current Instruction Register; CIR). Buses: data, address and control: how this relates to assembly language programs.
https://youtu.be/UdHK35N-Kuo

Questions:
- Function and uses
- How these two are different.
- Function of all registers and buses.
- Naming the registers
- What are control signals?

Notes:
- ALU - Arithmetic logic unit - does calculations and logic.
- CU - Control unit - Sends signals to control how the processor works, such as flow of data.
- Registers - memory locations in the CPU that are very fast.
- PC - stores the address of the next instruction. (Program counter)
- MAR - Memory address register - stores the address of the data or instructions that are to be fetched from or sent to memory. Different to PC because it will hold address of data.
- MDR - memory data register - holds the data that is to be sent to or fetched from memory.
- CIR - current instruction register - holds the instruction that is being decoded and executed.
- ACC - Accumulator - Holds results of calculations from ALU.
- IR - Interrupt register
- Buses - communication link channels between CPU and memory
- Address Bus - Carries address of MAR from processor to memory only goes from CPU to memory
- Data bus - carries instructions and data both ways.
- Control bus - Sends control signals both ways.

Summary:
The CPU is split into multiple components that work with the memory to do functions. The registers are memory locations that in the CPU that work together to execute an instruction.

Student 2



Topic: 1.1.1(b) The Fetch-Decode-Execute Cycle; including its effects on registers.
https://youtu.be/OTDTdTYId2g

Questions:
- How would
- What would be the end result of one cycle.
- Go through the steps.
- What is in each section
  - F - The program counter increments Address MAR → memory → MDR
  - D - CIR decodes the instruction into OPcode + operand. Opcode
  - E - The operand is executed onto the accumulator

Notes:

* For jump instruction CIR sends address to PC.

Fetch
1. The program counter starts with first instruction.
2. This is copied into MAR
3. Address goes across address bus.
4. Control unit sends signal on control bus to say read the memory.
5. The contents of address 0000 are sent across data bus.
6. The contents are copied into MDR and CIR (because it's an instruction)
7. The instruction has been fetched so the PC increments.
8. The first part of the code is the Opcode and the last is the address. In this case the opcode means load and will load that area in the data - operand.
9. Instruction is carried out.

Summary:
The fetch-decode-execute cycle goes through a continuous fetch cycle that goes through instructions. It uses the opcode to do instructions, again

4. Flipped Learning

Optional – strongly recommended!

Cornell Note Taking

Exercise: From the example notes on the previous page, look at what they wrote, and then given them a score using the criteria below.

Cornell Notes Marking Criteria

A* 7 points	A 6 points	B 5 points	C 4 points	D 3 points	E 2 points	U 1 point
Extremely detailed notes with diagrams, linked clearly to spec points, with questions and summary complete.	Detailed notes with diagrams, linked clearly to spec points, with questions and summary complete.	Notes, questions and summaries all completed in detail, but isn't obvious how the notes link to the spec points.	Detailed enough to revise from in parts, but not consistently to a high standard. Questions and summaries attempted.	Medium detailed notes and diagrams. No questions or summaries.	Low detail notes, not enough to revise from, some sections incomplete.	Nothing handed in or extremely minimal notes.

Student 1:	Student 2:
Your score:	Your score:
Reasons for your score:	Reasons for your score:

4. Flipped Learning

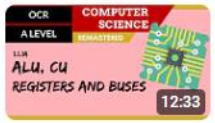
Compulsory – must do!

Cornell Note Taking

Exercise:

Download the Cornell Notes templates for the five 1.1.1 topics from here:
https://learn-cs.com/wp-content/uploads/2020/05/1.1.1-Cornell-notes-DIL.pdf

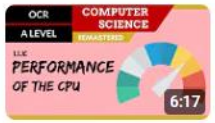
Follow the links to watch the videos, then make detailed Cornell Notes as described on the previous slides.

1. 

1. OCR A Level (H046-H446) SLR1 - 1.1 ALU, CU, registers and buses
Craig'n'Dave • 179K views • 3 years ago

2. 

2. OCR A Level (H406-H466) SLR1 - 1.1 Fetch, decode, execute cycle
Craig'n'Dave • 86K views • 3 years ago

3. 

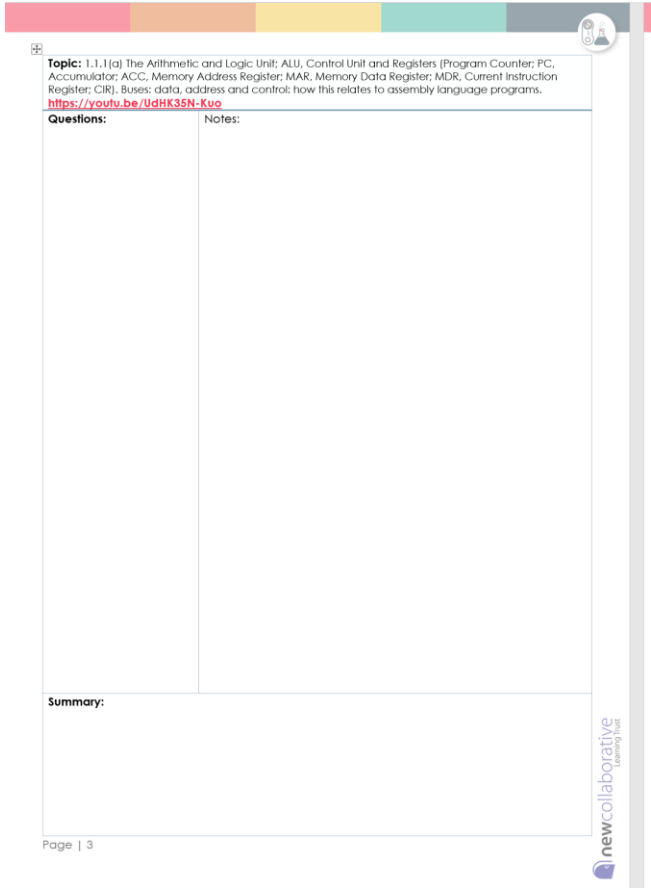
3. OCR A Level (H406-H466) SLR1 - 1.1 Performance of the CPU
Craig'n'Dave • 40K views • 3 years ago

4. 

4. OCR A Level (H466) SLR1 - 1.1 Pipelining
Craig'n'Dave • 25K views • 1 year ago

5. 

5. OCR A Level (H046-H466) SLR1 - 1.1 Von Neumann and Harvard
Craig'n'Dave • 57K views • 3 years ago

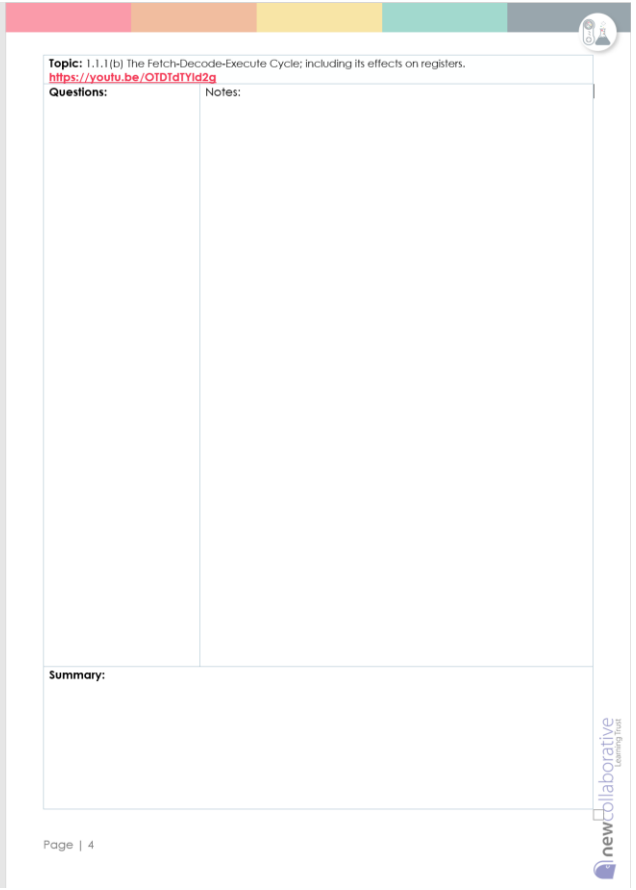


Topic: 1.1.1(a) The Arithmetic and Logic Unit; ALU, Control Unit and Registers (Program Counter; PC, Accumulator; ACC, Memory Address Register; MAR, Memory Data Register; MDR, Current Instruction Register; CIR). Buses: data, address and control: how this relates to assembly language programs.
https://youtu.be/UdHK35N-Kuo

Questions:	Notes:

Summary:

Page | 3



Topic: 1.1.1(b) The Fetch-Decode-Execute Cycle; including its effects on registers.
https://youtu.be/OTDTdTYId2g

Questions:	Notes:

Summary:

Page | 4

5. Programming

Optional – strongly recommended!

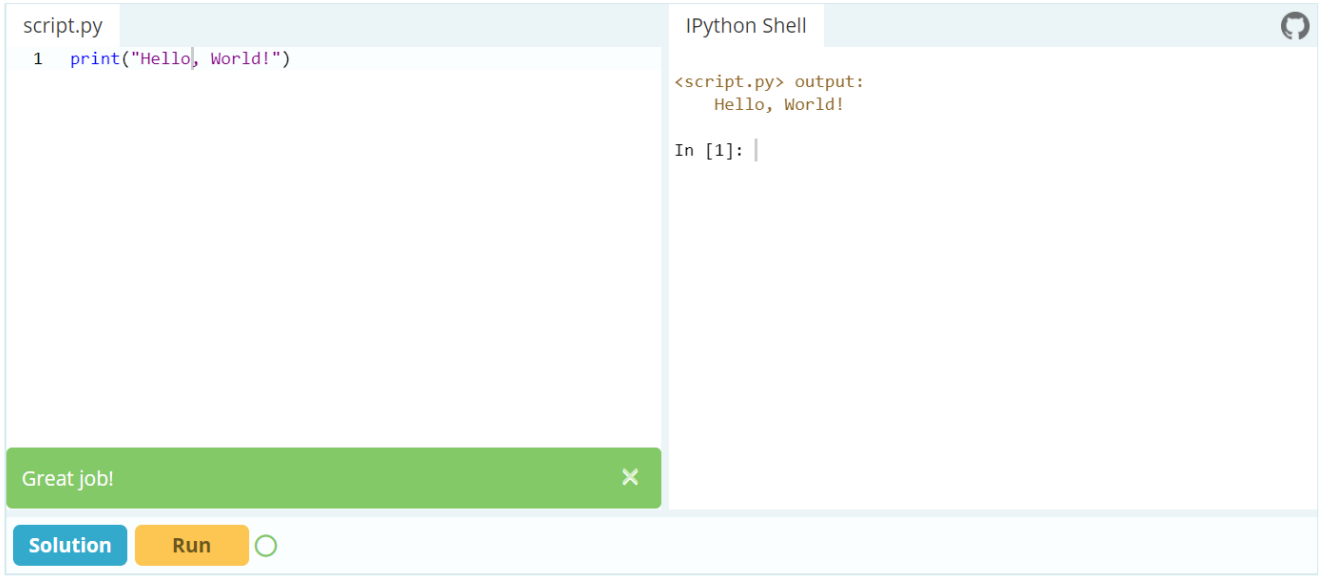
Exercise:

1. Have a go at each of the warm up tasks on the next few slides, ensuring you read them carefully
2. Paste your code for the final exercise below:

a) Hello, World! (5-10mins)

https://www.learnpython.org/en/Hello%2C_World%21

Use the "print" command to print the line "Hello, World!".



Add screenshot of code here

5. Programming

Optional – strongly recommended!

b) Variables and Types (10-20mins)

https://www.learnpython.org/en/Variables_and_Types

Add screenshot of code here

5. Programming

Optional – strongly recommended!

c) Lists (10-20mins)
https://www.learnpython.org/en/Lists

Add screenshot of code here

5. Programming

Optional – strongly recommended!

d) String formatting (10-20mins)

https://www.learnpython.org/en/String_Formatting

Add screenshot of code here

5. Programming

Optional – strongly recommended!

e) String operations (10-20mins)

https://www.learnpython.org/en/Basic_String_Operations

Add screenshot of code here

5. Programming

Optional – strongly recommended!

f) Conditions - branching (15-25mins)
https://www.learnpython.org/en/Conditions

Add screenshot of code here

5. Programming

Optional – strongly recommended!

g) Loops - iteration (15-25mins)
https://www.learnpython.org/en/Loops

Add screenshot of code here

5. Programming

Optional – strongly recommended!

h) Functions – reusable code (20-30mins)
https://www.learnpython.org/en/Functions

Add screenshot of code here

Feel free to continue with the online tutorials, but you should now have enough knowledge to be able to attempt the 'challenge' exercises on the next slides…

6. Programming

Compulsory – must do!

Preparation for Challenge 1

Heads or tails

Create a program that asks the user how many times they would like to simulate tossing a coin. It should then generate either a 'head' or 'tails' randomly, for the specified number of times. It should print out at the end:

1. How many coin tosses it simulated
2. How many heads in total
3. How many tails in total
4. The best head streak – best run of heads in a row without any tails
5. The best tails streak – best run of tails in a row without any heads

To the right are example outputs to help you work out how to program the solution:

```
How many coin tosses would you like to simulate?
-> 10
Result 1 : tails
Result 2 : heads
Result 3 : heads
Result 4 : tails
Result 5 : tails
Result 6 : heads
Result 7 : tails
Result 8 : heads
Result 9 : tails
Result 10 : tails
The total count of heads:  4
The best streak for heads:  2
The total count of tails:  6
The best streak for tails:  2
Would you like to simulate again?(y/n)
->
```

```
Would you like to simulate again?(y/n)
-> y
How many coin tosses would you like to simulate?
-> 20
Result 1 : tails
Result 2 : tails
Result 3 : heads
Result 4 : tails
Result 5 : heads
Result 6 : heads
Result 7 : heads
Result 8 : tails
Result 9 : heads
Result 10 : heads
Result 11 : tails
Result 12 : heads
Result 13 : tails
Result 14 : heads
Result 15 : heads
Result 16 : tails
Result 17 : heads
Result 18 : heads
Result 19 : heads
Result 20 : tails
The total count of heads:  12
The best streak for heads:  3
The total count of tails:  8
The best streak for tails:  2
Would you like to simulate again?(y/n)
->
```

6. Programming

Compulsory – must do!

Code for Challenge 1

Heads or tails

Exercise: Paste your solution and proof of working here:

Add screenshots of code and output here

6. Programming

Compulsory – must do!

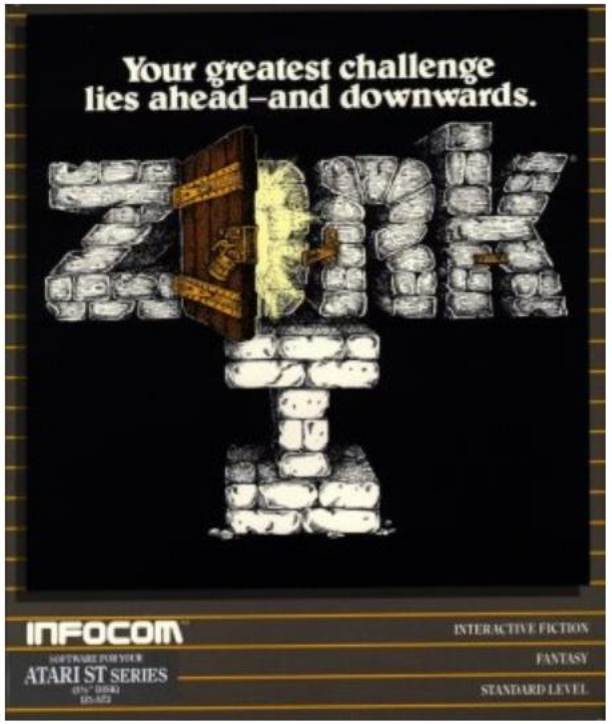
Preparation for Challenge 2

Exercise: Play the text adventure game Zork

That's right, I am asking you to play a game, and that forms part of your Summer Independent Learning!

At the time of writing this link worked to a web version of Zork http://textadventures.co.uk/games/view/5zyoqrsugeopel3ffhz_vq/zork

If it doesn't, then Google "Text Adventure Zork Online" and you should be able to find a link.



There is method to my madness, playing a cutting edge game like this (for 1977ish) allows you to think about all the skills a programmer needs:

- Use of variables
- Inputs
- Outputs
- Lists
- Operators
- Formatting strings
- Conditions and branching
- Loops or iteration
- Functions

As you play, think about what must be happening under the hood, how do you collect items, have battles, have choices, get random responses, move around?

If you get stuck, try the help guide here: http://www.eristic.net/games/infocom/zork1.html

Lets play some games! Expected completion time: I have no idea! I got lost inside this dungeon for what seemed like an eternity, maybe you will have more luck!

6. Programming

Compulsory – must do!

Making your version of the game

Exercise: Outline the basic idea behind your own text adventure game here.

To think about:

1. Who are your characters?
2. Are they playable?
3. Who are the non playable characters?
4. What skills or weapons can you pick up along the way?
5. What items can you carry and how many?
6. How do you win?
7. Will you have health or lives?
8. How do you manoeuvre around?

My ideas:

My ideas continued:

6. Programming

Compulsory – must do!

Making your version of the game

Exercise: Paste your code here – if you need more space just duplicate the slide or put it on a word document and bring it with you to your first lesson in September. Feel free to delete the help boxes.

Help 1: Tech with Tim
https://youtu.be/DEcFCn2ubSg

This is a basic introduction to a text adventure game on YouTube – a good place to get you started.

Help 2: Invent with Python
https://inventwithpython.com/invent4thed/chapter5.html

This game was has a lot of similar features to Zork and will help if you read the explanation and code.

Copy and paste your code here, so I can have a go!