Assembler Test

1. This performs mathematical operations (e.g. add / subtract) and logical operations (e.g. AND, OR). The ALU ~~tells~~ loads in data from input registers, performs the operations on the data (~~for~~ based on instructions from the CPU) then stores the result in an output register

2. This decodes program instructions and handles logistics for the execution of decoded instructions. i.e. it instructs the memory, ALU and input & output devices how to respond to the instructions that have been sent to the processor.

3. The Fetch-Execute cycle is the sequence of operations that are completed in order to execute an instruction. This cycle takes place over several CPU clock cycles as the components involved are constructed using sequential / combinational logic circuits.

   ① FETCH: the next instruction is to execute is retrieved from main memory

   - Address from PC is copied to MAR
   - PC is incremented to point to next instruction in memory
   - Instruction held at that address is copied to MDR by the data bus and then copied to the CIR

   ② DECODE: the fetched instruction is decoded (to form recognisable operations)

   - Retrieved instruction is sent to the CU and decoded

   ③ EXECUTE: the instruction is executed

   - CU signals functional CPU components using a sequence of control signals
   - May result in changes to data registers / PC / ALU / I/O etc

* Between each execute and fetch stage, the content of the status register is checked for changes that could mean there is an interrupt → leads to the appearance of a the process 'multitasking' and running different programs.

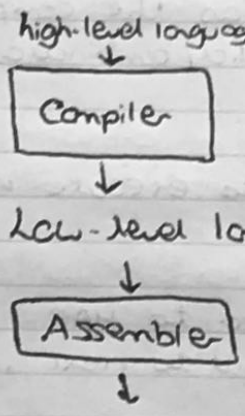
4. PC: holds the memory address of the next instruction to be ~~incremented~~ executed. This is incremented once the address has been copied to the MAR as the instructions are held sequentially in memory.

IR: holds the most recently fetched instruction (divided into operand & opcode)

5. This is a collection of status flags which are set upon certain conditions arising in the ALU. This information is used to decide if branching should be allowed.

E.g. Zero flag (Z) will be set if an arithmetic/logical operation was zero

6.



Compiles translate programs written in high-level languages into machine code. This is outputted as a file that can be run independently.

7. General form: LABEL: OPCODE OPERAND(S) | COMMENT

E.g. START: move.b #5, D0 | load D0 register with the constant 5 (a byte)

Here move.b is the opcode (mnemonic) and #5 and D0 are the operands. move.b can move 1 byte (as specified by .b)

8. E.g. ~~move.b~~ add.b #$A, D0 | add 10 to D0

This adds the value of 10 to the value currently stored in the D0 register.

9. This is used to refer to data that is located in an address field of an instruction

E.g. move.b #$42, D5 | puts the hex value 42 into register D5

As the operand forms part of the instruction it remains constant throughout execution of a program

10. With absolute addressing the <u>address</u> of the data (i.e. the address of the register) is specified in the operand, so no further processing is required

E.g. move.b $0 0F (address is F$, not the value)

Disadvantage: it doesn't allow position independent code as a program will consistently use the same memory address.