AQA Computer Science A-Level

4.7.3 Structure and role of the processor and its components

Advanced Notes

Specification:

4.7.3.1 The processor and its components:

Explain the role and operation of a processor and its major components:

- arithmetic logic unit
- control unit
- clock
- general-purpose registers
- dedicated registers, including:
    - program counter
    - current instruction register
    - memory address register
    - memory buffer register
    - status register

4.7.3.2 The Fetch-Execute cycle and the role of registers within it:

Explain how the Fetch-Execute cycle is used to execute machine code programs including the stages in the cycle (fetch, decode, execute) and details of registers used.

4.7.3.3 The processor instruction set:

Understand the term ‘processor instruction set’ and know that an instruction set is processor specific.

Know that instructions consist of an opcode and one or more operands (value, memory address or register).

4.7.3.4 Addressing modes:

Understand and apply immediate and direct addressing modes.

4.7.3.5 Machine-code/assembly language operations:

Understand and apply the basic machine-code operations of:

- load
- add
- subtract
- store
- branching (conditional and unconditional)
- compare
- logical bitwise operators (AND, OR, NOT, XOR)
- logical
  - shift right
  - shift left
- halt

Use the basic machine-code operations above when machine-code instructions are expressed in mnemonic form- assembly language, using immediate and direct addressing.

4.7.3.6 Interrupts:

Describe the role of interrupts and interrupt service routines (ISRs); their effect on the Fetch-Execute cycle; and the need to save the volatile environment while the interrupt is being serviced.

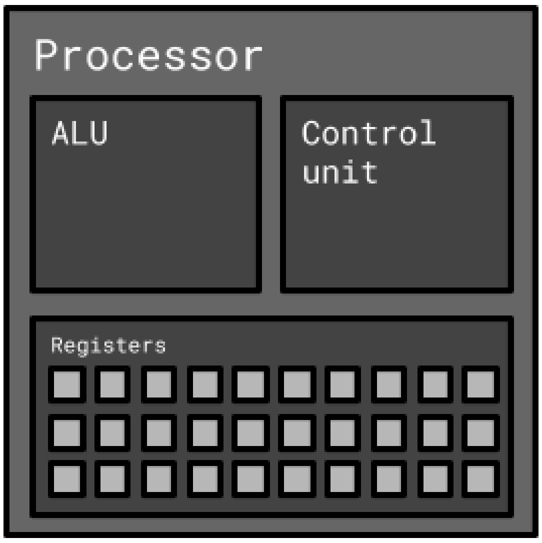
4.7.3.7 Factors affecting processor performance:

Explain the effect on processor performance of:

- multiple cores
- cache memory
- clock speed
- word length
- address bus width
- data bus width

The processor and its components

At the heart of every computer is a processor which executes instructions in order to run programs. Processors contain an arithmetic logic unit (or ALU), a control unit and numerous registers.



Arithmetic logic unit (ALU)
The role of the ALU is to perform arithmetic and logic operations. Arithmetic operations are mathematical operations like addition, logic operations include AND, OR and XOR.

Control unit
A processor's control unit is responsible for controlling the various components of the processor. It is responsible for controlling the fetch-execute cycle which is covered later in these notes.

Registers
Registers are small storage locations used to hold data temporarily. They have high read and write speeds.

General purpose registers are registers that can be used as storage for any data that is required by instructions during execution. In contrast, special purpose registers are assigned to the storage of specific information. These are listed in the table below.

Register	Purpose
Program counter (PC)	Used to hold the memory address of the next instruction to be executed in the fetch-execute cycle.
Current instruction register (CIR)	Holds the instruction that is currently being executed by the processor.
Memory address register (MAR)	Stores the memory address of a memory location that is to be read from or written to.
Memory buffer register (MBR)	Also called the memory data register (MDR). Holds the contents of a memory location that has been read from or data that is to be stored.
Status register (SR)	Contains a number of bits, the values of which can change to indicate the occurrence of an interrupt.

Clock

Inside a computer's processor is the system clock, a device that generates a timing signal which changes at a regular frequency. This signal is used to synchronise communication between the components of the processor and the rest of the computer system.

The Fetch-Execute cycle

The fetch-execute cycle is a continuous cycle performed by the processor. It consists of three stages: fetch, decode and execute.

Fetch

In the fetch stage of the cycle, the next instruction to execute is retrieved from main memory.

1. The content of the PC is copied to the MAR
2. The content of the MAR is transferred to main memory by the address bus
3. The instruction is sent from main memory to the MBR by the data bus
4. The PC is incremented by one
5. The content of the MBR is copied to the CIR

Decode

In the decode stage of the cycle, the fetched instruction is decoded.

1. The content of the CIR is decoded by the control unit
2. The decoded instruction is split into two parts: opcode and operands

Note

Opcode specifies the operation to carry out. **Operands** are data on which the operation is performed.

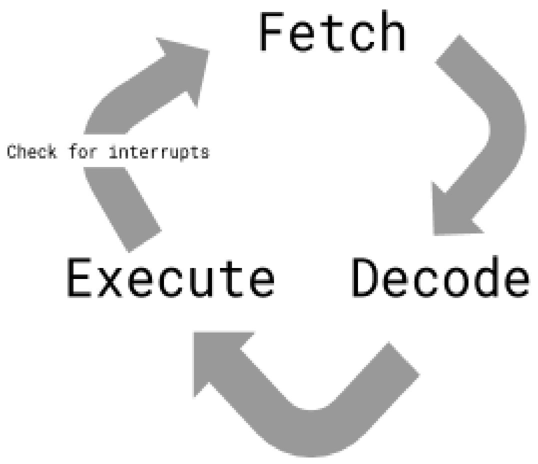
Execute

The instruction is carried out in the execute stage of the cycle.

1. Any data required by the instruction that isn't present in registers is fetched
2. The instruction is carried out
3. Results of any calculations are stored in general purpose registers or main memory

Check for interrupts

Between each execute stage and fetch stage of the cycle, the content of the status register is checked for changes that could signify the occurrence of an interrupt.



The processor instruction set

A processor's instruction set is the group of instructions that it can carry out. Each type of processor has its own instruction set, so instructions for one processor may not be compatible with other processors.

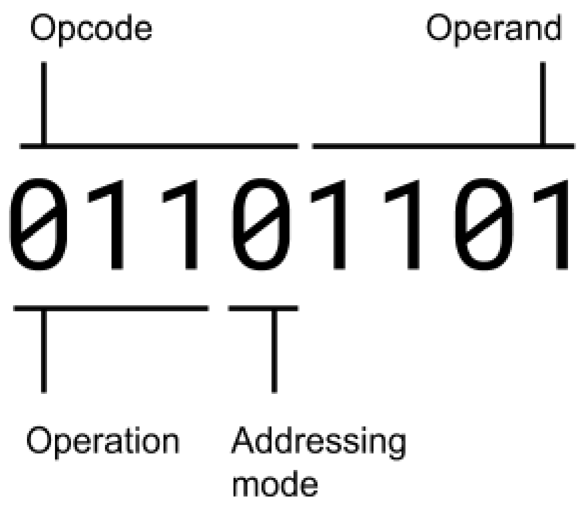
Synoptic Link

Machine code instructions are formed entirely from the binary digits 0 and 1.

Machine code is covered in **classification of programming languages** under **fundamentals of computer systems**.

Instructions are usually stored in machine code and consist of two primary parts: opcode and one or more operands.

Opcode specifies the type of operation that is to be carried out, for example: addition, subtraction or logical shifting. Operands are the pieces of data on which the operation is performed.



<u>Addressing modes</u>
One bit in a machine code instruction is usually assigned to the addressing mode in use. There are two addressing modes: immediate and direct.

In immediate addressing, the value specified in the operand is to be treated as the actual value. For example, if the operand were 18, the value to be used by the operation would be 18.

Direct addressing differs from immediate addressing in that the value specified by an operand signifies a memory address. For example, if the operand were again 18, the value to be used by the operation would be whatever the content of memory location 18 is.

Machine-code/assembly language operations

The table below lists the basic machine-code operations and their equivalent in AQA's assembly language. This is a type of assembly language that will be used in exam questions, you need to learn to both interpret and write AQA assembly language.

Operation	AQA Assembly Language	Description
Load	`LDR Rx, <memory reference>`	Load the value stored in the memory location specified by `<memory reference>` into register x.
Store	`STR Rx, <memory reference>`	Store the value that is in register x into the memory location specified by `<memory reference>`.
Add	`ADD Rx, Ry, <operand>`	Add the value specified by `<operand>` to register y and store the result in register x
Subtract	`SUB Rx, Ry, <operand>`	Subtract the value specified by `<operand>` from the value in register y and store the result in register x.
Move	`MOV Rx, <operand>`	Copy the value specified by `<operand>` into register x.
Compare	`CMP Rx, <operand>`	Compare the value stored in register x with the value specified by `<operand>`.
Branch (unconditional)	`B <label>`	Always branch to the instruction at the position specified by `<label>` in the program.

Branch (conditional)	`B<condition> <label>`	Branch to the instruction at position `<label>` if the last comparison met the criterion specified by `<condition>`. Possible values for `<condition>` are: ● EQ: equal to ● NE: not equal to ● GT: greater than ● LT: less than
Logical AND	`AND Rx, Ry, <operand>`	Perform a bitwise logical AND operation between the value in register y and the value specified by `<operand>` and store the result in register x.
Logical OR	`ORR Rx, Ry, <operand>`	Perform a bitwise logical OR operation between the value in register y and the value specified by `<operand>` and store the result in register x.
Logical XOR	`EOR Rx, Ry, <operand>`	Perform a bitwise logical XOR (exclusive or) operation between the value in register y and the value specified by `<operand>` and store the result in register x.
Logical NOT	`MVN Rx, <operand>`	Perform a bitwise logical NOT operation on the value specified by `<operand>` and store the result in register x.
Logical shift left	`LSL Rx, Ry, <operand>`	Logically shift left the value stored in register y by the number of bits specified by `<operand>` and store the result in register x.
Logical shift right	`LSR Rx, Ry, <operand>`	Logically shift right the value stored in register y by the number of bits specified by `<operand>` and store the result in register x.
Halt	`HALT`	Stops the execution of the program

Logical shifting

A logical shift is an operation that can be performed on binary numbers which involves shifting all of the bits in a number (doubling or halving the number) a specified number of positions to the right or to the left.

Example: Perform a logical shift left by three places on the binary number `011011010`

All we need to do is move all of the digits three places to the left. In doing so, we add three zeros to the end of the number.

```
011011010000
```

Interrupts

An interrupt is a signal sent to the processor by another part of the computer requesting the attention of the processor. Examples of hardware interrupts could be the computer's I/O controller informing the processor that the mouse has been moved or that a keyboard key has been pressed.

Software can also send interrupts, which could include unexpected errors like division by zero or a stack overflow.

When an interrupt occurs, it is detected as a change in the content of the status register between the execute and fetch stages of the fetch-execute cycle.

Interrupts can be handled using the vectored interrupt method. When an interrupt occurs, the processor stops executing the current program and places the content of its registers onto the system stack. This is referred to as saving the "volatile environment".

Now that the processor has saved its progress on the system stack, it loads the appropriate interrupt service routine: a series of instructions for handling the interrupt that is specific to the type of interrupt.

Once the processor finishes executing the interrupt service routine, it restores the volatile environment from the system stack and resumes execution of any programs that were running before the interrupt.

Factors affecting processor performance

A processor's performance is key to the speed of a computer, and there are numerous factors that can affect this.

Number of cores

The number of cores a processor has directly affects the performance of the processor. Each core can perform its own fetch-execute cycle independently of others, so different applications can be allocated different cores. Quad-core (four-core) and even octa-core (eight-core) processors are becoming common.

Cache memory

A processor's cache is a small portion of incredibly fast memory. It has read and write speeds far higher than hard disk drives and even SSDs. Cache is used to store frequently used information and reduces time wasted in fetching the same information from main memory time and time again. The more cache a processor has, the more information it can store and the more time it can save in fetching information from main memory.

Clock speed

A processor's clock speed relates to the frequency of the pulses generated by the system clock. The higher the frequency, the more cycles of the fetch-execute cycle can be completed in the same period of time. However, errors can occur when a computer's clock speed is increased too far.

Word length

A word is a group of bits that is treated as a single unit by a processor. Words can be used for representing both instructions and data. The length of a word is the number of bits that are assigned to it, with higher word lengths allowing for more bits to be transferred and manipulated as a single unit.

Address bus width

The width of a bus relates to the number of parallel wires that make up the bus. Increasing the width of the address bus increases the range of addresses that it can specify, hence increasing the computer's amount of addressable memory. Adding a single wire doubles the number of addressable memory locations.

Synoptic Link

The busses in the processor are covered in detail in the notes for **Internal hardware components of a computer.**

Data bus width

Increasing the width of the data bus increases the volume of data that can be transferred over the bus at any one time. A wider data bus allows the processor to fetch more data from main memory in one cycle of the fetch-execute cycle, reducing the number of cycles required to fetch large volumes of data.