AQA Computer Science A-Level
4.6.2 Classification of programming languages

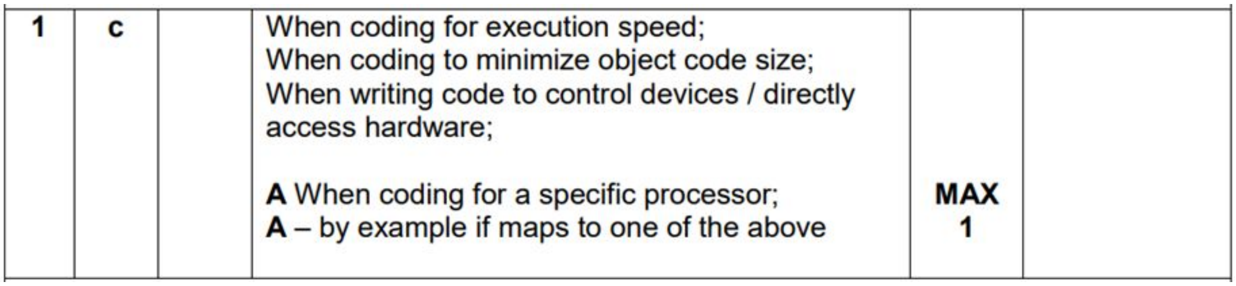
Past Paper Mark Schemes

January 2009 Comp 2

3	(a)		Load **B**; **Add** #5; **A** *absolute addresses instead of A and B* Store **A**;	**3**

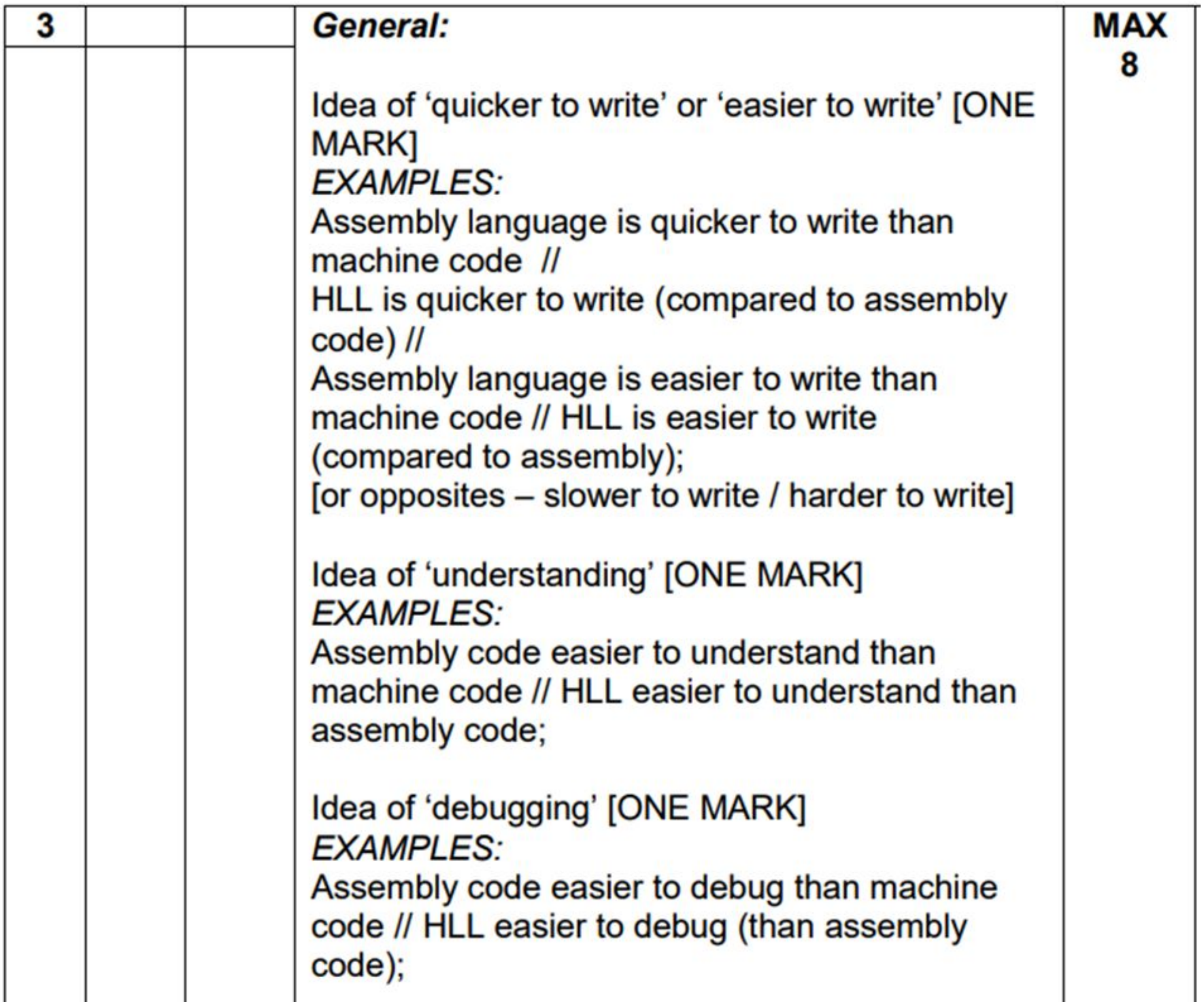
January 2012 Comp 2

1	**a**		Third (generation) // 3; **R** High Level Language	**1**	Do not reject high level language if answer also contains ‘3rd generation’ – refer upwards for anything else.
1	**b**	**i**	Hexadecimal // base 16; **A** Hex	**1**	Hex used in textbook
1	**b**	**ii**	Take up less space when printing/viewing; **NE** takes up less space Less likely to make errors; Op-codes are easier to recognize; Easier to understand; Less time taken when coding as more concise // quicker to program;		
			NE – easier to read **NE** – quick to write	**MAX 1**	
1	**b**	**iii**	Lowest address : 00 Highest address : FF BOTH correct to gain one mark; Accept 0 for lowest address Accept 255 for highest address	**1**	Accept notation in front of hex &, $



1	**c**		When coding for execution speed; When coding to minimize object code size; When writing code to control devices / directly access hardware; **A** When coding for a specific processor; **A** – by example if maps to one of the above	**MAX 1**	

January 2013 Comp 2



3			***General:*** Idea of 'quicker to write' or 'easier to write' [ONE MARK] *EXAMPLES:* Assembly language is quicker to write than machine code // HLL is quicker to write (compared to assembly code) // Assembly language is easier to write than machine code // HLL is easier to write (compared to assembly); [or opposites – slower to write / harder to write] Idea of 'understanding' [ONE MARK] *EXAMPLES:* Assembly code easier to understand than machine code // HLL easier to understand than assembly code; Idea of 'debugging' [ONE MARK] *EXAMPLES:* Assembly code easier to debug than machine code // HLL easier to debug (than assembly code);	**MAX 8**

			Assembly language: Solution expressed in terms of mnemonics; **A.** an example of a full instruction (operand and opcode) Easier to make mistakes in assembly language; Instruction composed of op-code and operand; Solution translated by using an assembler; Code is hard to port to other types of computer // machine-oriented languages; One assembly language instruction relates to one machine code instruction; ***Situation*** – working on embedded hardware // need for small object code size // need for fast execution // need to access hardware/registers directly; ***Imperative language:*** Imperative is where the programmer gives the computer a sequence of instructions to perform; Selection/Sequence/Iteration constructs available; **A.** a full example of a selection/iteration construct Library of pre-written functions available; Solution translated by using a compiler / interpreter; A compiler might not be available for a specific processor (disadvantage); ***Situation*** – anything sensible that would need a		

			HLL (for example games programming) ***Declarative language:*** (Certain languages) define what is to be computed rather than how the computation is to be done; (Certain languages) lack side effects; (Certain languages) have a clear link to mathematical logic; (Certain languages) express solutions in terms of facts and rules // rule-based; (Certain languages) will use an inference engine to work out the answer; The user asks a question of the system rather than provide an algorithm of the solution; Uses back-chaining/backtracking; (Certain languages) express solutions using markup languages (such as HTML); (Certain languages) express solutions as CSS / regular expressions / (subset of) SQL; **A.** example code from part of a declarative program (ie an SQL statement) ***Situation*** – medical diagnosis // expert systems // database query //creating a web page/website ; ***Imperative and Declarative language:*** Solution expressed in terms of statements written using English-like keywords; Code easier than assembly language to port to other types of computer; One language statement maps to many (more than one) machine code instruction; NOTE: accept any sensible situation for each area.		

June 2010 Comp 2

7	**(d)**	**(ii)**	HLLs are problem oriented; HLL programs are portable // machine / platform independent ; English like keywords/commands/syntax/code; **R** closer to English Less code required // less tedious to program // one to many mapping of HLL statements to machine code commands; Quicker/easier to understand/write/debug/learn/maintain code; **R** just quicker/easier HLLs offer extra features e.g. data types/structures // structured statements // local variables // parameters // named variables/constants; **R** procedures/modular **A** example of a data structure **NE** "extra features" without example Speed of execution not crucial for most tasks so faster execution of assembly language not required; Most computer systems have a lot of (main) memory/RAM so compact object code not essential; **Accept converse points for Assembly Language** **MAX 3**	3

June 2011 Comp 2

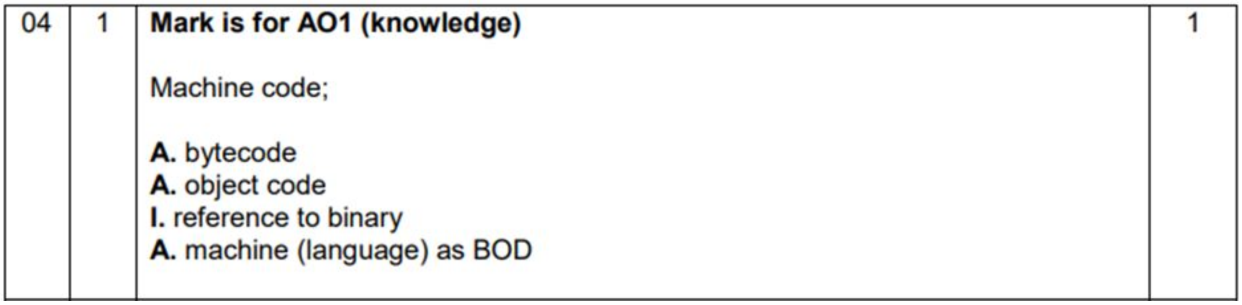
4	**a**		Second (generation); **A** 2 **R** assembly code / language Note: Adding "assembly" / "assembler" does not talk out a valid mark for second / 2	**1**
4	**b**		(memory) Address / location / offset; **A** line number **R** instruction number	**1**
4	**c**		(y) Opcode / operation code; **A** op-code **NE** operation (z) Operand;	**2**
4	**d**		**Individual Instructions:** One to one / each assembly language instruction translates to one machine code instruction; **Programs:** Figure 2 assembly language equivalent of figure 3 // figure 3 machine code version of figure 2 // figure 3 is assembled version of figure 2; **NE** figure 3 "binary version" of figure 2 **NE** different generations of language	**1**

June 2012 Comp 2

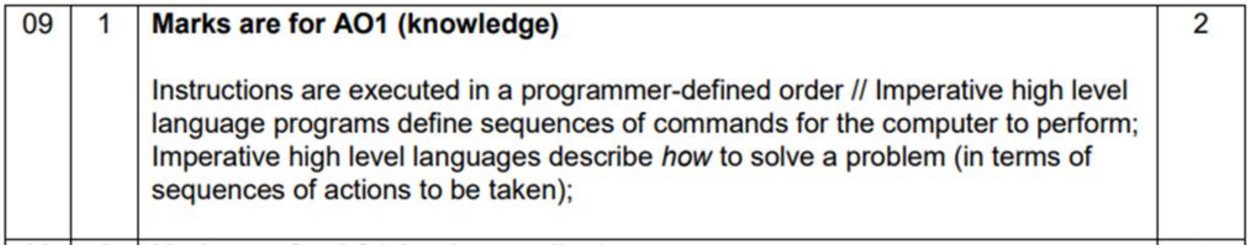
2	b		Easier to understand; Takes less time to code (as using mnemonic opcodes and hex operands); Fewer mistakes made in coding; Ability to add comments to code; Use of symbolic names for operands // easier to remember opcodes/mnemonics; Use of labels; Easier to maintain/debug; **NE** easier to read/code/write **NE** quicker **A** converse points if clearly discussing machine code	MAX 2

4	b		Languages used for a specific problem type/domain; **A** different uses/purposes/tasks Access to specific data types; Providing different function libraries; Languages developed for specific hardware / devices ; Languages developed for visual applications/GUIs; Competition between different companies who develop languages;	MAX 1

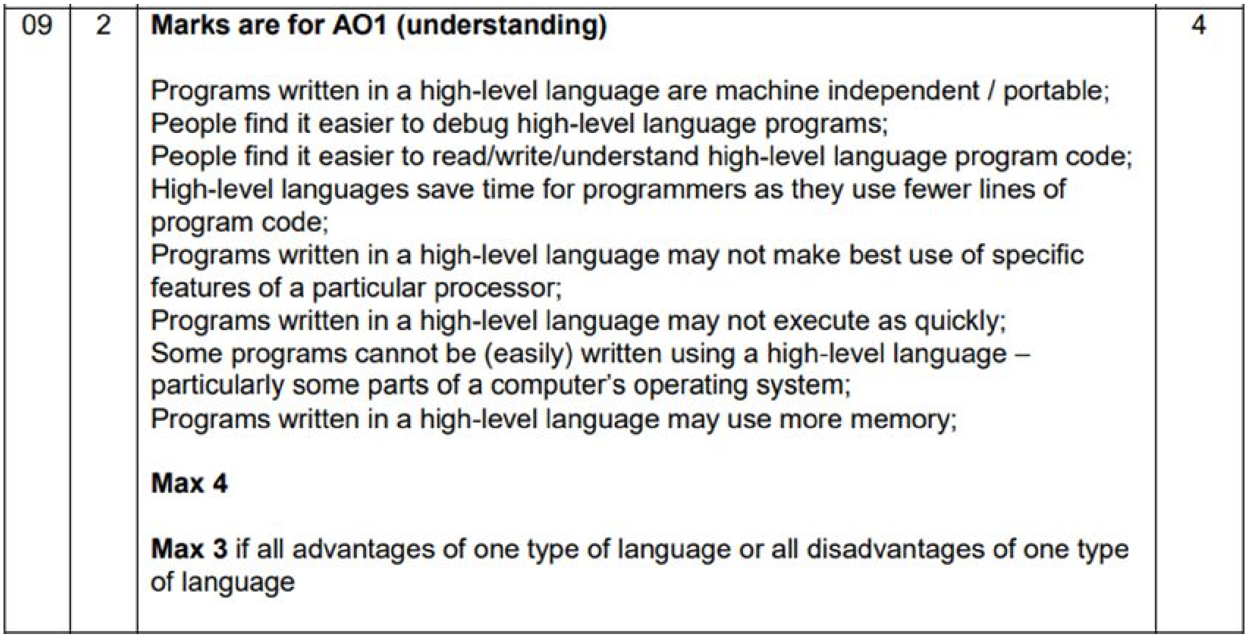
June 2016 AS Paper 2



04	1	**Mark is for AO1 (knowledge)** Machine code; **A.** bytecode **A.** object code **I.** reference to binary **A.** machine (language) as BOD	1

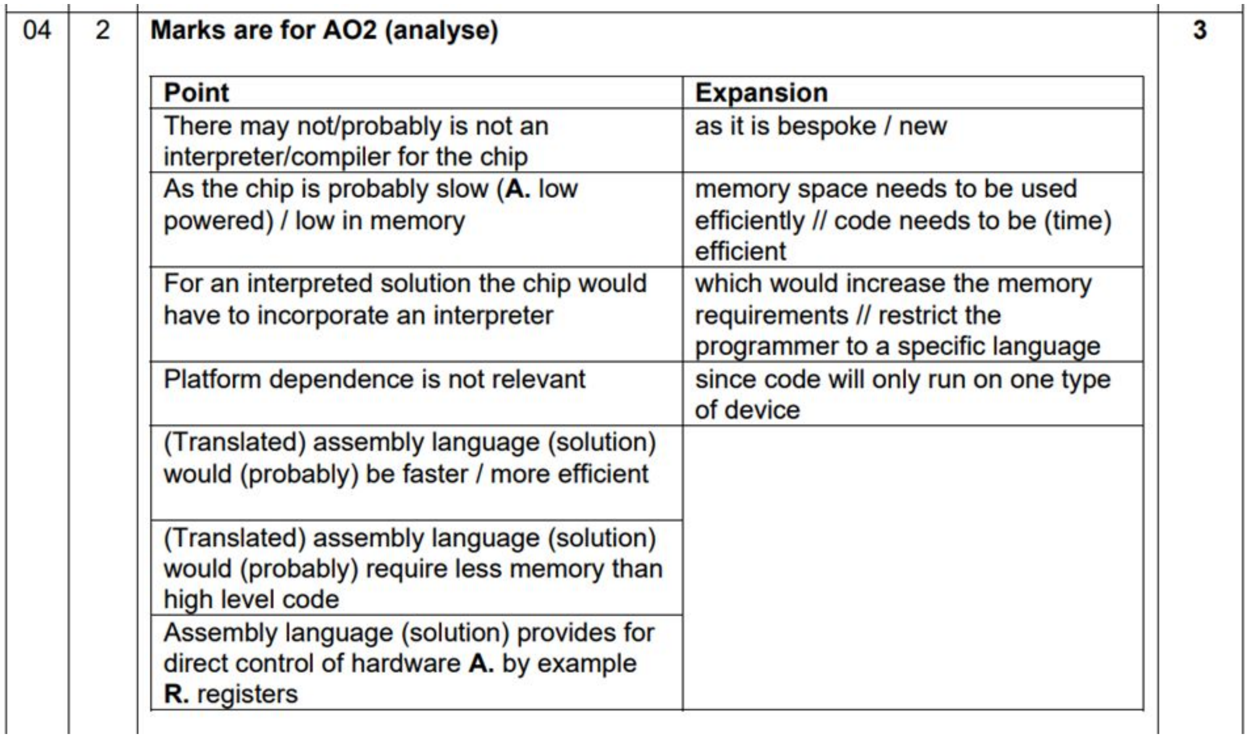


09	1	**Marks are for AO1 (knowledge)** Instructions are executed in a programmer-defined order // Imperative high level language programs define sequences of commands for the computer to perform; Imperative high level languages describe *how* to solve a problem (in terms of sequences of actions to be taken);	2



09	2	**Marks are for AO1 (understanding)** Programs written in a high-level language are machine independent / portable; People find it easier to debug high-level language programs; People find it easier to read/write/understand high-level language program code; High-level languages save time for programmers as they use fewer lines of program code; Programs written in a high-level language may not make best use of specific features of a particular processor; Programs written in a high-level language may not execute as quickly; Some programs cannot be (easily) written using a high-level language – particularly some parts of a computer's operating system; Programs written in a high-level language may use more memory; **Max 4** **Max 3** if all advantages of one type of language or all disadvantages of one type of language	4

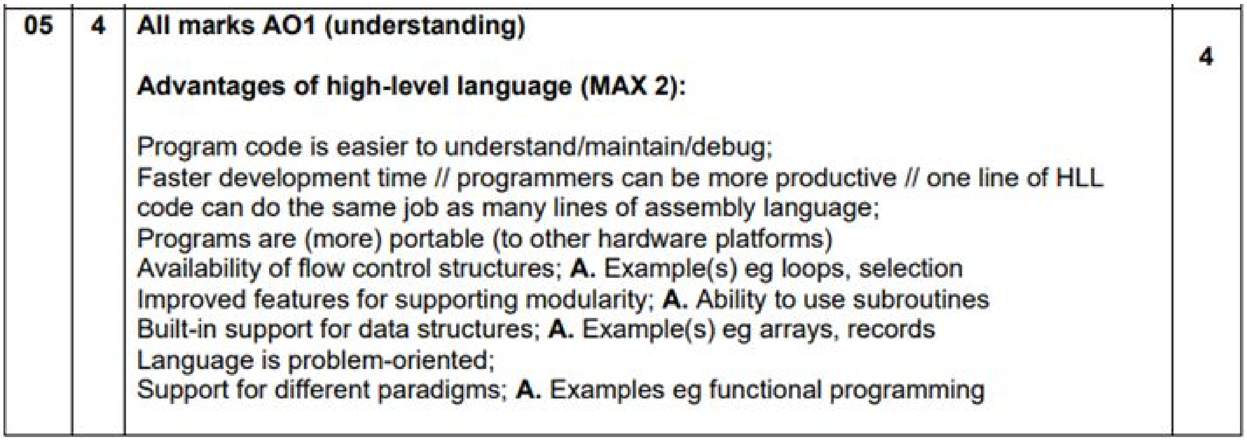
June 2017 AS Paper 2



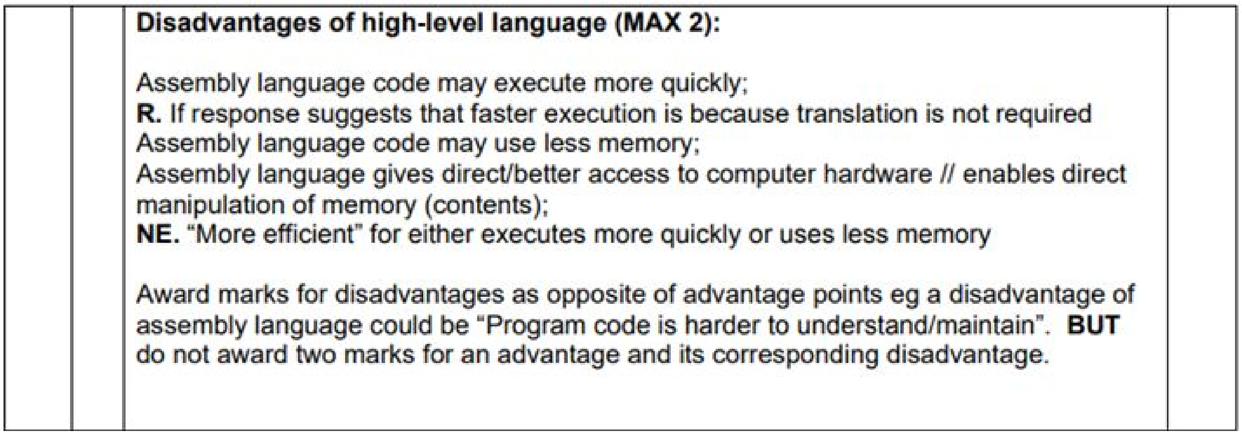
04	2	**Marks are for AO2 (analyse)**	**3**

Point	**Expansion**
There may not/probably is not an interpreter/compiler for the chip	as it is bespoke / new
As the chip is probably slow (**A.** low powered) / low in memory	memory space needs to be used efficiently // code needs to be (time) efficient
For an interpreted solution the chip would have to incorporate an interpreter	which would increase the memory requirements // restrict the programmer to a specific language
Platform dependence is not relevant	since code will only run on one type of device
(Translated) assembly language (solution) would (probably) be faster / more efficient	
(Translated) assembly language (solution) would (probably) require less memory than high level code	
Assembly language (solution) provides for direct control of hardware **A.** by example **R.** registers	

June 2017 Paper 2

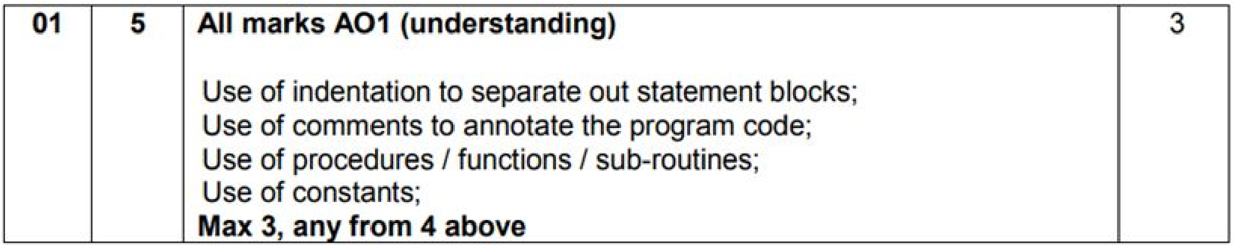


05	**4**	**All marks AO1 (understanding)** **Advantages of high-level language (MAX 2):** Program code is easier to understand/maintain/debug; Faster development time // programmers can be more productive // one line of HLL code can do the same job as many lines of assembly language; Programs are (more) portable (to other hardware platforms) Availability of flow control structures; **A.** Example(s) eg loops, selection Improved features for supporting modularity; **A.** Ability to use subroutines Built-in support for data structures; **A.** Example(s) eg arrays, records Language is problem-oriented; Support for different paradigms; **A.** Examples eg functional programming	**4**



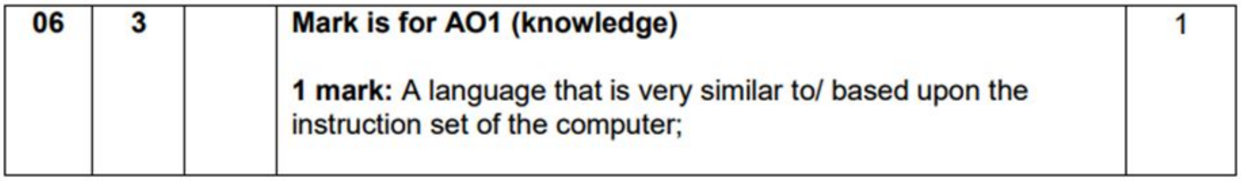
		Disadvantages of high-level language (MAX 2): Assembly language code may execute more quickly; **R.** If response suggests that faster execution is because translation is not required Assembly language code may use less memory; Assembly language gives direct/better access to computer hardware // enables direct manipulation of memory (contents); **NE.** "More efficient" for either executes more quickly or uses less memory Award marks for disadvantages as opposite of advantage points eg a disadvantage of assembly language could be "Program code is harder to understand/maintain". **BUT** do not award two marks for an advantage and its corresponding disadvantage.	

Specimen AS Paper 1



01	**5**	**All marks AO1 (understanding)** Use of indentation to separate out statement blocks; Use of comments to annotate the program code; Use of procedures / functions / sub-routines; Use of constants; **Max 3, any from 4 above**	3

Specimen AS Paper 2



06	**3**		**Mark is for AO1 (knowledge)** **1 mark:** A language that is very similar to/ based upon the instruction set of the computer;	1