EXAM PAPERS PRACTICE

6.3 Types of program translator
Mark Scheme

Q1.

All marks AO1 (understanding)

Level	Description	Mark Range
4	A line of reasoning has been followed to produce a coherent, relevant, substantiated and logically structured response. The response covers all three areas indicated in the guidance below and in at least two of these areas there is sufficient detail to show that the student has a good level of understanding. To reach the top of this mark range, a good level of understanding must be shown of all three areas.	10-12
3	A line of reasoning has been followed to produce a coherent, relevant, substantiated and logically structured response which shows a good level of understanding of at least two areas indicated in the guidance below.	7-9
2	A limited attempt has been made to follow a line of reasoning and the response has a mostly logical structure. At least four points have been made. Either a good level of understanding of one area from the guidance has been shown or a limited understanding of two areas.	4-6
1	A few relevant points have been made but there is no evidence that a line of reasoning has been followed. The points may only relate to one or two of the areas from the guidance or may be made in a superficial way with little substantiation.	1-3

<u>Guidance – Indicative Response</u>

For each guidance point, if the student expands on the point to explain in what way the measure will improve performance then this can be considered to be a second point. For example:

- “Using a processor with more cores” is one point.
- “Using a processor with more cores which will be able to execute multiple instructions simultaneously” is two points.

Note that just “faster” is not enough to count as an expansion point without an explanation of why.

1. Server Hardware

Replace the processor with one which has more cores

Replace the processor with one which has more cache memory // increase the amount of cache memory

Replace the processor with one which runs at a faster clock speed **NE.** faster processor

Use a parallel processor architecture // use more processors <u>which can work in parallel</u>

Use a processor with a bigger word size

Use a processor that makes (better) use of pipelining

Install more RAM // main memory // primary memory

Use RAM // main memory // primary memory with a faster access time

Replace HDDs with SSDs // Replace HDDS with HDDs that can read data at a faster rate

Defragment the HDD

Replace the motherboard with one which has buses which run at a faster clock speed

Replace the motherboard with one which has more lines in the data bus

Use the Harvard architecture

Distribute the processing across multiple servers

2. Network

Replace the network cable with cable that has a higher bandwidth // replace copper cable with fibre-optic cable **A.** Ethernet cable for fibre-optic NE. higher bandwidth network

Replace any wireless / WiFi connections with wired ones

Replace the network cards with ones that can transmit data at a higher bitrate

Consider the overall network design eg how the network is divided into subnets **A.** split the network into subnets

Use a star topology (instead of a bus)

Consider using a more efficient protocol for the data across the network

Add additional wireless access points

3. Database and Software

Use a more efficient technique for controlling concurrent access to the database // replace record/table locks with serialisation/timestamp ordering/commitment ordering

Replace the database software with software that uses more efficient algorithms for tasks **A.** examples eg replace linear search with binary search

Use the index feature of the database to speed up searching on fields that are commonly used for this purpose

Rewrite the database software in a language that is suitable for concurrent execution // use a functional programming language for the database software

Ensure the software is compiled rather than executed by an interpreter // rewrite the software in assembly language/machine code

Review the conceptual model of the database to see if it contains any inefficiencies such as data redundancy that could be eliminated **A.** normalise the database design

Consider if it would be appropriate to sacrifice normalisation of the conceptual model to improve performance

Use a non-relational database system **A.** examples eg NoSQL

Distribute the data across multiple servers

Try to reduce the amount of other (unrelated) software that might be running on the database server at the same time

Try to reduce the number of database accesses that need to be made simultaneously // run some tasks at quiet times / overnight

Purge / archive data that is no longer necessary / in use

[12]

Q2.

(a) A language that is close to the hardware;

Language that interacts with basic hardware / tasks of the computer;

Commands map directly / very closely to processor instruction set;

One instruction maps to one processor instruction;

A processor / architecture dependent language // language that is not portable;

NE machine code or assembly language
R directly executable by the processor

MAX 1

(b) HLL allows several machine code statements to be replaced by one high level statement // HLL program shorter that its low level equivalent;

HLL program expressed in language that is human-oriented / uses English-like keywords;
A structured English
NE written in English / closer to English

Allow programmers to:
use meaningful identifier names;
use procedures / functions / subroutines / libraries;
use programming structures such as IF THEN ELSE / REPEAT UNTIL;
use data structures such as arrays / lists;

Easier to see logic / structure of program / what is to be executed;
A easier to spot / check errors // easier to debug;

Can maintain one codebase for use across multiple architectures;

MAX 2

(c)
- The role of a translator is to take program code / source code and to translate it into a low-level / machine code
- A compiler takes the whole source code and translates it (into machine code / object code)
- Compiled code will execute more quickly
- Produces an executable file // no need for compiler to be distributed with program // no need to distribute source code to execute program
- An interpreter works through / translates / recognises program source code line-by-line
- Interpreters call routines built into the interpreter to execute commands
- Interpreting code is slower than running compiled code
- Can run (parts of) a program using an interpreter even if it contains syntax errors

- Source code is required for the program to be interpreted // when running interpreted code the interpreter is always required

Situations (MAX 1 each for compiler and interpreter)
Compiler:

- So that source code cannot be accessed by users
- When creating an executable file for distribution
- Where speed of execution is important
- Where targeting a device with a small amount of memory

Interpreter:

- To allow execution on a wide range of processors
- When prototyping and testing / debugging code
- When no compiler yet exists for the processor

A (example of) building a web-application
How to award marks:

Mark Bands and Description

To achieve a mark in this band, candidates must meet the subject criterion (SUB) and all 5 of the quality of written communication criteria (QWCx).
SUB Candidate has made at least five mark-worthy points and covers both interpreter and compiler with a valid situation for at least one.
QWC1 Text is legible.
QWC2 There are few, if any, errors of spelling, punctuation and grammar. Meaning is clear.
QWC3 The candidate has selected and used a form and style of writing appropriate to the purpose and has expressed ideas clearly and fluently.
QWC4 Sentences (and paragraphs) follow on from one another clearly and coherently.
QWC5 Appropriate specialist vocabulary has been used.

5-6

To achieve a mark in this band, candidates must meet the subject criterion (SUB) and 4 of the 5 quality of written communication criteria (QWCx).
SUB Candidate has made at least three mark-worthy points and covers both compiler and interpreter.
QWC1 Text is legible.
QWC2 There may be occasional errors of spelling, punctuation and grammar. Meaning is clear.
QWC3 The candidate has, in the main, used a form and style of writing appropriate to the purpose, with occasional lapses. The candidate has expressed ideas clearly and reasonably fluently.
QWC4 The candidate has used well-linked sentences (and paragraphs).
QWC5 Appropriate specialist vocabulary has been used.

3-4

To achieve a mark in this band, candidates must meet the subject criterion (SUB) and 3 of the 5 quality of written communication criteria (QWCx).
SUB Candidate has made a small number of relevant points.
QWC1 Most of the text is legible.
QWC2 There may be some errors of spelling, punctuation and grammar but it should still be possible to understand most of the response.
QWC3 The candidate has used a form and style of writing which has many deficiencies. Ideas are not always clearly expressed.
QWC4 Sentences (and paragraphs) may not always be well-connected.
QWC5 Specialist vocabulary has been used inappropriately or not at all.

1-2

Candidate has made no relevant points.

0

Note: Even if English is perfect, candidates can only get marks for the points made at the top of the mark scheme for this question.

If a candidate meets the subject criterion in a band but does not meet the quality of written communication criteria then drop mark by one band, providing that at least 4 of the quality of language criteria are met in the lower band. If 4 criteria are not met then drop by two bands.

MAX 6

[9]

Q3.

(a) 1 special purpose (application software);
A specific purpose
R special (software) / specialist (software)

2 word processor / / spreadsheet / /
presentation software / program / / database;
A any other sensible answer
R (web) browser
R text editor

3 translator (software / program);
A translating / translation

4 utility (software / program);

R just trade name of a specific piece of software unless used as an example (ie MS Word)

4

(b) (i) assembly (language);
A assembly code
R assembler

1

(ii) has to be translated into <u>machine code</u> / / each assembly language instruction will be translated into <u>machine code</u> equivalent;

by an assembler;
A converted for translated
A first generation for machine code

2

(iii) Because it does not have the same processor (type) / / the instruction set is different / / different architecture / platform;

(Assembled / linked for a) different operating system;
NE operating software

The program refers to a memory address that does not exist on this computer / / relocatable code used but addressing system on new machine different;

not enough memory space;
required peripherals are not available;

required library (code / program) missing;

MAX 1

[8]

Q4.

(a) A set of rules / regulations (to allow communication between devices) // set of agreed signals / codes for data exchange;
NE a rule // a regulation // a signal // a code
NE instruction(s)

1

(b) Analyses statement by statement each line of source code
A runs / translates / executes line by line
R compiles (line by line)

Calls routines to carry out each instruction / statement

Max 2

(c) Instructions / programs stored (with data) in main memory; **A** memory // RAM

Program run by fetching, (decoding and executing) instructions (from main memory)* in sequence;

Program can be replaced by loading another program into (main) memory

Contents of a (main) memory location can be interpreted as either an instruction or data;

* = This mark can be awarded without the explicit reference to main memory if main memory has already been mentioned elsewhere in the response.

Otherwise, the answer must make clear that the instructions are coming from the main memory to get this mark.

3

(d) LOAD 21
STORE 23

LOAD 22
STORE 21

LOAD 23
STORE 22

1 mark for value from 21 stored into 23;
1 mark for value from 22 being moved to 21;
1 mark for value from 23 being moved to 22;

Alternative :

LOAD 22
STORE 23

LOAD 21
STORE 22

LOAD 23

STORE 21

1 mark for value from 22 stored into 23;
1 mark for value from 21 being moved to 22;
1 mark for value from 23 being moved to 21;
DPT if a different temporary storage area is used
I end of statement separators
Max 2 if the program does not fully work

3

(e) Robots find it hard to adapt to changes in environment // Robots are unable to adapt to changes easily;

Robots find it hard to work with 3D vision;

Robots find it hard to detect edges between similar objects // robots find it hard to perform shape detection;

Robots find it hard to get feedback when gripping items;

Robots find it hard to pick up balls // ball difficult shape to grip // balls can roll away;

Robots have limited processing power // too many variables to deal with;

Programming for vision/grip is a complex problem;
A child builds up experience of using touch / vision;

A Robot cannot recognise when it makes mistakes;
A Robot can't think for themselves // can't perform lateral thinking

Max 3

(f) (i) (Lens focuses) light / photons onto image sensor;
R if uses 'reflection';

Image sensor is a CMOS / CCD / photoelectric device;
CCD used ADC to convert measurement of light intensity into binary;
CMOS uses transistors to generate binary value;
Image sensor converts light into discrete / electrical signals / binary;

Image is captured when the shutter is pressed;
Large pixels collect more electrons than small pixels and so produce better quality images;
Firmware performs data processing to "tidy up" image;
(Colour) filter used to generate data separately for Red, Green, Blue colour components;
Aperture / shutter speed can be adjusted to cope with varying lighting conditions;
Image is recorded as group / array of pixels // Image sensor consists of array of pixel (sensors)//etched into the image sensor's silicon are pixels;

Image data transferred to robot;
Image data usually stored on solid-state disk;

Max 3

(ii) Robot has a low powered microprocessor;

Too much image data for the robot to process quickly // smaller resolution can be processed quicker;

A high resolution image has too much image data for the robot to store // low resolution uses less storage space;

Do not need high resolution to determine colour of balls;

NE allows more images to be stored

Max 1

[16]

Q5.

(a) Third (generation) // 3;
R High Level Language

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

Hex used in textbook

1

(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

(iii) Lowest address : 00
Highest address : FF

BOTH correct to gain one mark;

A 0 for lowest address
A 255 for highest address
A 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

(d) A compiler produces object code/machine code;
whilst an interpreter does not produce any object code;
Interpreted code will execute slower;
than executing the object code produced by a compiler;
You always need the interpreter to interpret source code;
but you do not need the compiler to execute a compiled program;

Once compiled source code is no longer required to run the program;
An interpreter always needs source code at runtime;
Compiled code can only be executed on a machine with the same processor type / instruction set;
Interpreted code is more portable;
A compiler translates the whole source code (at once);
An interpreter analyses the code line by line;
NE reads

Max 4

[9]

Q6.

(a) Very hard/difficult to understand;
Very easy to make mistakes;
Hard to find any errors/mistakes in the code;
Time consuming to develop software in assembly language;
Lack of portability;
Lack of in-built functions/procedures;
NE harder to learn

Max 2

(b) Compiler produces object code to distribute that is difficult to reverse engineer/ no need to distribute the source code;
Compiler optimises the code // The object code /program runs faster (as it does not need translating);
NE "Runs faster", if not clear whether this applies to the program or the compiler.
The target computer has no need to have the original compiler;
Object code can be installed on target computer;
No interpreter available for target machine;

2

[4]

Q7.

(a)

Number	**Component Name**
1	Memory Address Register
2	Address Bus
3	Memory Data / Buffer Register
4	Data Bus

4

(b) The <u>instruction</u> is held in the CIR;
A IR
The <u>control unit / instruction decoder</u> decodes the instruction;
The opcode identifies the type of instruction it is;
Relevant part of CPU / processor executes instruction;
A ALU
Further memory fetches / saves carried out if required;
Result of computation stored in accumulator / register / written to main

memory;
Status register updated;
If jump / branch instruction, PC is updated;
A SCR

Max 3

(c) Can be <u>displayed</u> in less space;
R takes up less space **NE**
Easier to remember / learn / read / understand;
Less error prone;

Max 1

(d) (i) Assembler;

1

(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;
A converse points for Assembly Language

3

[12]

Q8.

(a) So that source code cannot be accessed by users;
Users do not need to have an interpreter / compiler / translator // users do not need programming environment;
Users do not need knowledge of the programming environment;
So that the program will execute more quickly;
NE it’s faster
NE does not need to be compiled each time executed/run
R saves disk space

Max 2

(b) Can’t know what type of processor will be in user’s computer // Internet users have range of computers / devices with different processors;
A compiled program will only execute on a processor of specific type / family / with same instruction set//A program run using an interpreter can execute on a computer with any type of processor;
A References to just different types of computer / device rather than specifically processors
NB Virtual Machine
R No compiler exists
R computers may have different web browsers/software

2

[4]

Q9.

(a) Compiler
R Interpreter
A Misspellings where meaning remains clear e.g. complier
R More than one answer e.g. compiler or interpreter

1

(b) Assembler
A Misspellings where meaning remains clear
R More than one answer

1

[2]

Q10.

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

3

(b) (i) Assembler;

1

(ii) Compiler;
R Interpreter

1

[5]

Q11.

(a) Text / scene / code / object editor;
A Word processor
A form/screen for text/code entry/input/write

1

(b) Translates (**A** converts/changes) program code/source program; from high level language; into machine code/object code;
R binary
Checks the program code/statements for errors;
If no errors found the executable file is generated;
T/O –1 Mark if the explanation includes 'executing the machine code'

Max 2

error report/list; **A** error message(s) **R** error (only)
an intermediate (object) file;
A a copy of the source code;

1

(c) Interpreter software translates/checks/reads the program code one statement at a time (**A** line by line);
Checks the statement for the correct syntax;
If no errors found, interpreter recognises the statement;
Interpreter calls a procedure to execute the statement;

A if no error found, that statement is executed;
If an error is found, program execution is halted (and the error reported);
The program runs until an error is found;

Max 2

(d) (i) Interpreter should allow for faster program development / faster error correction / errors easier to identify ;
A easier to debug

1

(ii) Compiler/source code will not be needed in order to distribute the final executable code / the exe code (alone) can be distributed to others;
the exe code (**A** the program) will execute (run) faster;
the exe code/ the program cannot be changed (by others);

Max 1

[8]

Q12.

(a) (memory) address / location;
R Line number

1

(b) second (generation) //assembly language/code/program //
2 / 2nd;

1

(c) (i) assemb<u>ler</u>;
R. assemb<u>ly</u>

1

(ii) error list / error report / error count / **A** error message / highlight statement(s) illegally formed / instruction count // symbol table;
R error

1

(d) program (instructions are) transferred from backing store to main memory;
program consists of a sequence of instructions;
stored in a (continuous area of) <u>main memory</u>;
an <u>instruction</u> is fetched (and decoded);
and then <u>instruction</u> executed (by the processor);
program can be replaced by another program at any time;
program instructions are treated as data;

Max 4

[8]

Q13.

(a) 3rd (generation);

1

(b) (i) (Program) 2; (Program) 3;

1

(ii) (Program) 1;

1

(c)

	Assembler	Compiler	None
Program 1		✓	
Program 2	✓		
Program 3			✓

If more than one entry per row – look for a single tick
If mixture of X and ticks used – mark (as long as only one tick per row)

3

(d) The interpreter software is resident in memory at the same time as the application program is run;
The interpreter recognises/translates/reads/converts each statement **A** instruction/line;
T/O if added "converts to machine code"
(Syntax) checks the program; line-by-line; if 'error free' the statement /line is executed;
The interpreter calls a procedure/code to execute the statement;
When (first) error encountered program execution is halted;

Max 2

(e) The processor/architecture/(hardware) platform is different;
instructions are not the same;
Assembler software is processor/architecture specific;
A Assembler software is 'machine specific'

The computers use a different operating system;
R the 'computer' might be …
R possible bugs in the software

Max 1

[9]

Q14.

(a) (i) Functions always return some value when called;
Procedures may return a value;
Functions appear in expressions;
Procedures do not appear in expressions;
Procedures name alone makes up the statement / call <name>

Max 2

(ii) Anything named which is plausible;
Examples could include: computation / formatting / string handling;
R software features / button events / DLL
A Dynamic Linked Library

2

(b) (i) True/Yes/ 1;

1

(ii) False/No/0;

1

(iii) Error;

1

(c) Program / constant / function / procedure / module / unit / user defined type / record / label / object /class;

Max 2

(d) **Advantage of an Interpreter:**
- Should allow faster/easier program development // faster/easier testing / debugging / finding errors;
- Correcting mistakes is less time consuming;

Max 1

Advantage of a compiler:
- The executable code/object code/program will run faster;
- Once the executable file has been produced no further action;
- Software distribution requires no further software to be available to the user;
- Prevents tampering of the code by users other than the developer;

Max 1

2

[11]

Q15.

(a) Assembly language/code/program // second (generation);

1

(b) Machine code // first (generation);

1

(c) (Memory) address / location;
R Line number

1

(d) Assembler;

1

(e) 1-to-I (mapping between instructions/op code/numbers written in assembler and their machine code equivalent) / each assembly instruction translates into one machine code instruction

1

(f) <u>Error</u> / error list / error report / error count / highlight statement(s) illegally formed // / instruction count // symbol table;

1

[6]

Q16.

(a) (i) Machine code;

1

(ii) Assembly code/language;

1

(iii) Pascal / Visual Basic / Basic / Java or any other 3GL;

1

(b) Problem Oriented;

Portable/Platform independent;
One-to-many mapping of HLL statement to machine code statement;
Data types / structured statements / local variables / parameters / data structures / named variables / named constants / English-like keywords / commands / syntax;
Quick /easy to understand / write / debug / learn / maintain;
R easy to read

Any 2 from 5

2

(c) (i) Syntax checking / Translate the (whole) source program;
Generate executable code;

2

(ii) Syntax checking / Translate the source program line by line;
Execute the program;

2

(d) (i) Where tested software is to be shipped to a user / or any situation where the program needs to respond rapidly / can be run as a stand alone program / create an executable;
Reasons may be software unable to be changed / speed of execution;

2

(ii) Software under development / run in a sandbox;
Reasons may be software debugging tools / controlled environment;

2

[13]

Q17.

(a) (i) A compiler translates the whole source code;
Into object code; **A** machine code (instructions); **A** executable file;~

If implied that compiler executes then talked out

Max 1

(ii) Interpreter translates line by line; as it executes/runs;

If object code created then talked out

Max 1

(b) (i) Use compiler when execution should run as fast as possible;
When development is finished; when giving program to end user to run;
to protect source code from end user interference;
To turn program without translator/compiler on computer;

Max 1

(ii) Use interpreter during development time/testing/debugging/finding/correcting mistakes;
To support platform independence;
(e.g. Java $\rightarrow$ bytecode, bytecode interpreted)

Max 1

(c) Assembler translates assembly/low level program;
Compiler translates high level program;
Assembler maps 1:1; compiler maps 1:many;

Max 1

[5]

Q18.

(a) (i) Assembler;

1

(ii) Interpreter / compiler;

Max 1

(b) Problem oriented;
Portable; machine independent;
One-to-many mapping of HLL statement to machine code statement;
Datatypes; structured statements; local variables; parameters; data structures;
A *example of a datastructure;*
Named variables; named constants; English-like keywords/commands;
Quick/easy to understand / write / debug / learn / maintain;
R quick/ easy to use
R modular
R procedures
R takes longer to translate
R closer to English

Max 2

(c) Easier to understand / more transparent; less error prone; easier to maintain / change (if the value changes);

Allow by example
eg if VAT rate changes only need to change value in declaration

Max 1

(d) (i) Accept any imperative HLL such as Pascal/VB/C/C++/PL1/Cobol;

SEE TABLE FOR DIFFERENT LANGUAGE EXAMPLES FOR (ii) & (iii)
Ignore line breaks in statements

1

(ii) *1 mark for correct key words in correct order (shown in bold in table overleaf); 1 mark for correct Boolean expression / loop control expression ;*

2

(iii) *1 mark for correct key words in correct order (shown in bold in table overleaf); 1 mark for correct boolean expression;*

2

If (i) does not match (ii) and (iii) do not give marks for (ii) and (iii)
If candidate names a language you are not familiar with, contact your team leader

Language	Bool expr	Iteration *....are possible statements(ignore)*	Selection *....are possible statements(ignore)*

Pascal
Delphi
Kylix
	a>b
	a=b
	a< >b
	a<b
	a>=b
	a<=b
		FOR <variable>:= <value1> TO <value2> DO ......
		REPEAT ......
		UNTIL <bool expr>
		WHILE <bool expr>
		DO .....
			IF <bool expr> THEN ......
			else part optional
			CASE <variable> OF
			<value1>.....
			<value2>......
			ENDCASE
			else part optional.
			No of values can vary

Visual Basic VSScript	a>b a=b a< >b a<b a>=b a<=b	FOR <variable> = <value 1> TO <value2> ..... NEXT DO WHILE/UNTIL <bool expr> .....LOOP DO ..... LOOP UNTIL/WHILE <bool expr> WHILE <bool expr> ..... WEND	IF <bool expr> THEN ..... END IF *Else part optional* SELECT CASE <variable> CASE <value1> ..... CASE <value2> ..... End Selct *Else part optional* *No of CASE values can vary*

C/C++
Java
Javascript
	a>b
	a==b
	a!=b
	a<b
	a>=b
	a<=b
		FOR (<initialisation>; <condition>; <increment>).....
		WHILE (bool expr) DO ....
			IF (bool expr) { }
			Else part optional
			SWITCH ( ) {CASE <value>: BREAK}
			No of CASE values can vary

COBOL	a>b a=b a< >b a<b a>=b a<=b	PERFORM .....<number> TIMES PERFORM VARYING <variable> FROM <value>BY <value> UNTIL <bool expr using variable>	IF <bool expr> PERFORM ...... *Else part optional*

Fortran	a.LT.b
	a.GE.b
	a.LE.b
	a.GT.b
	a.NE.b
	a.EQ.b
		DO <number> < variable>= <init value> <final value>
		step value optional
			IF <bool expr> .....
			IF (<arithmetic expr>) label1, label2, label3

Basic	a>b a=b a< >b a<b a>=b a<=b	FOR <variable> = <start value> TO <stop value> .... NEXT <variable> *step value optional* REPEAT .... UNTIL <bool expr>	IF <bool expr> THEN ...... GOTO label1, label2, label3 DEPENDING ON <variable>

2

[10]

Q19.

(a) (i) (Data/address/control/internal/system) bus;
R just a description of a bus
R names of buses which don't exist e.g. memory bus

1

(ii) Store programs and/or data/files when not in use/
When computer is off permanent/long term storage
Of programs and/or data; save programs/data;
R offline/backup **R** ROM **R** temporary storage
A save on magnetic disk/ tape storage;
A information instead of data

1

(iii) (Machine code) instruction/data is fetched from main memory;
A what is fetched or from where
Instruction is decoded;
Instruction is executed (by the processor); **R** data executed

Max 2

(b) (i) Assembly language; mnemonic code; mnemonics; assembly code;
R low level language **A** assembler;

1

(ii) Translated/assembled/converted/decoded; into machine code (instructions);
R compiled **R** interpreted **A** object/target code;
A binary instructions;

2

(iii) Computer executes instructions in <u>programmer</u> defined sequence;
A the programmer tells the computer how to do it;
R user *instead of* programmer

1

(iv) Pascal /Visual Basic/Basic/C/C++/Cobol/Fortran/Ada/Delphi/Lylix/Modula /or any other imperative HLL
R Prolog
R Lisp
R Pop11

1

(v) One statement/instruction/command in a high level language translates into several machine code instructions; 1 to many;

1

(vi) Laborious/time-consuming to write; hard to debug; harder to program; easier to make mistakes; more difficult to understand/ learn; difficult to maintain; different assembler/instruction set for different type of computer; machine dependent; low level programs not portable;

Max 2

[12]

Q20.

(a) Easier maintenance/upgradeable; can get an overview of system;
Quicker to write/Easier development; can break problem down into sub tasks/ can re–use modules/ distribute among team;
R easier to read so can get a team to write program;
More systematic testing; can test a module at a time;
Fewer mistakes made; clear organisation of code;
Quicker/easier to debug; easier to see where errors are;
Fewer lines of code using subroutines; code not duplicated;

(1 mark for reason, 1 mark for explanation
OR 2 marks for good explanation per point)

Max 4

(b) Structured statements such as iteration / selection;
Use of Procedures / functions/subroutines/modules/libraries;
User defined data types; built–in data types; Data structures;
Can choose sensible names for identifiers;
English–like keywords / constructs; Indentation; Comments;
Use of <u>local</u> variables; parameters; named constants;
R closer to natural language/ almost written in English
R machine independent / problem oriented / top–down

Max 3

(c) Compiler translates <u>whole</u> source code into object/machine/runnable code / exe file;

Compiled program runs faster than interpreted program;
Interpreter translates line by line as it executes/is running;
Interpreter must be in memory to execute program; compiler only needed during translation stage, not during execution of program.;
interpreter runs one line at a time;

Max 2

[9]

Q21.

(a) 2 each for any feature of a LLL, contrasted with the corresponding feature of a HLL, eg processor dependence / processor independence machine-oriented / task oriented 1-to-1 / 1 -to-many correspondence source / object code *translation is by assembler / compiler respectively* should be at least two aspects contrasted

4

(b) Where either program size or execution speed is critical, or precise timing needed, eg o/s kernel, device driver, real-time control, comms systems, or if direct control of the way a program functions is a requirement, etc

Can award 1 if situation is simply named

2

[6]

Q22.

(a) Can run just a part of the program so easier to find problems.
Do not have to keep re–compiling the whole program when correcting errors

Any 1 × 1

1

(b) Produces complete object code / executable version
No need to install language to run the program / needs less memory
Runs faster as no time wasted on translation.
User cannot change source code

Any 2 × 1

2

[3]

Q23.

Assembler:
To translate an assembly (low level) language routine or program;
One to one translation;
Assembler is m/c specific;

Compiler:
To translate a HLL program;
One to many translation;
Has many options / stages;
Language specific;

For each, two statements to distinguish 1 mark each
Max 4

[4]

EXAM PAPERS PRACTICE

Q1.

A very good range of responses was received to this question, with approximately half of students achieving five or more marks. Most students addressed all three aspects of the question (hardware, network, database and software). Students tended to make more points about how the hardware could be improved than about the other two areas. This was acceptable but students needed to have covered all three areas to achieve a mark of ten or above.

Some students wrote too vaguely to achieve marks, for example by writing that a “faster processor” would improve performance, without referencing a factor such as the clock speed that would make the processor faster. Other mistakes included believing that the question required students to contrast thin-client and thick-client and that the system was web based.

A small number of students wrote about issues which might be causing the system to perform poorly instead of explaining how the performance of the system could be improved. Such responses were not worthy of a mark.

Q2.

This question was looking at generations of programming languages and the concept of a translator. Whilst a lot of students could describe low-level languages as ‘machine code and assembly code’ this was not enough to secure a mark. It was pleasing to see answers pointing out that they are specific to a processor type.

Describing the benefits of higher level languages has been asked before and the majority of students picked up a mark. Good answers pointed out some of the features of HLL such as procedures, functions and libraries. It was pleasing to see that students could point out that you could have one program and then use it on different architectures.

Some students are still providing answers for HLL programs in the form ‘written in English’ which is not creditworthy but those that could identify that the syntax, commands or keywords would be in English did get rewarded with a mark.

Part (c) was based around the topic of translators and asked students to describe the differences between compilers and interpreters. Students who had a sound grasp of this area scored highly and could point out the key features of compilers and interpreters alongside being able to provide a situation where they would be used.

It was evident from looking at students’ answers that there is also some confusion over this topic area. A group of students thought that it was the length of the code that would determine whether you should use a compiler or an interpreter. There was also confusion over the idea of error checking with a group of students assuming that an interpreter was better as it checked line by line or was a better translator as it was slower. A group of students presented the idea that a compiler would translate to a low-level language but an interpreter would translate to a high-level language.

The idea of one-to-one for low-level languages and one-to-many for high-level languages was presented by students in answering these question parts but was only rewarded with a mark if a student explained a bit further. For example, an answer of the form ‘a high-level language is one to many’ did not secure a mark unless the student explained what this actually means.

Q3.

(a) In this part , candidates were expected to complete a figure representing the classifications of various types of software. Over half of the candidates achieved three or more marks for this part. Candidates should be reminded that answers such as 'Microsoft Word document' will not be accepted as generic terms such as word processor are required. The most challenging item for candidates to identify was translator software.

(b) This part was well tackled with a good number of candidates securing all of the marks. For part (i) a few candidates provided the answer 'assembler' which was not accepted as the name for the second generation of programming language. The majority of candidates either answered with assembly code or assembly language which both secured the mark. Part (ii) was also generally well known, but there was confusion over what would be used to translate the assembly code into machine code. Most candidates secured at least one mark, but if they then discussed compiler or interpreters they did not secure the second mark which was for mentioning assembler. Part (iii) was found to be slightly more challenging and around 40% of candidates secured the mark. Good answers talked about differences between processors or the architecture of machines with a few including detailed examples. Weaker responses demonstrated some confusion over what the meaning of executable and discussed problems in the actual program itself, for example a bug in the code.

Q4.

This question asked students about a variety of topics all linked back to the idea of robotics. Over half of all students correctly provided a definition for protocol and the clearer answers linked this to the idea of an agreed set of rules to allow communication between devices. Some students who failed to secure the mark answered along the lines of instructions and programs rather than the idea of communication.

Part (b) asked students to identify how a HLL interpreter works. It was perhaps surprising that only half of the students managed to secure at least one mark on this question. It is clear that students got confused with the differences between a compiler and an interpreter with, some students answering this question by stating that it would 'compile'. Answers that just stated that 'it would interpret code….' also failed to secure marks. How an interpreter works beyond just translating code line by line is clearly not well understood and perhaps is an area centres could be encouraged to look at further.

Part (c) asked about the stored program concept. As a topic included in the name of the examination unit it was surprising to see that less than half of the students secured a mark on this question part. Of the credit worthy points made, it was common to see the idea that instructions are stored in the main memory of a device. A few students then went on to correctly identify that instructions are then fetched and executed by the processor. It was pleasing to see some students then discuss that the stored program concept allows different programs to be switched in and out of memory providing the ability to run different programs.

Unit 2 looks at only three machine code instructions and these were all given on the question paper in part (d) as a reminder to students. The correct answer only needed use of the LOAD and STORE instructions and over half of all students secured all three marks for this question part. A common mistake was to just see an answer of the form 'LOAD 21 ADD 22 STORE 23' showing that perhaps a student did not understand what the question was really asking them to perform which was swapping two stored values around.

Part (e) was a question looking at the differences between robotics and how we cope with situations. It was pleasing to see students identify aspects such as robots finding it hard to get feedback when gripping an item, and the problems in separating two similar coloured balls when they are obscuring each other. An answer that was rarely seen was the idea of a child building up experience over time and learning, compared to a robot just being programmed.

Students continue to struggle with identifying the major principles of how hardware devices work and it was common that no marks were achieved when discussing the digital camera and nearly 10% left this question part blank. Better answers considered items such as the shutter opening and closing to capture the image and correctly identifying the use of a sensor with more able students stating that this would be a CMOS or CCD device. It was surprising to see a few candidates talk about the use of film to capture the image.

The second part to the digital camera question was answered well and it was clear that students could link the idea of low resolution images to the needs of a robot. The simplest answer was to talk about the lower storage space required but the more able students linked this to the robot being able to process this amount of data faster or even that to identify colour differences would not require high resolution. Students should be encouraged to express their answers with direct reference to a question context, when appropriate, as this does allow them to demonstrate their understanding at a higher level.

Q5.

The majority of students correctly identified that it was a third generation programming language for question part (a). Incorrect answers included students just writing ‘high level languages’ alongside the wrong answers of fourth and second generation.

The majority of students could also identify that the machine code was written in hexadecimal format. Students did struggle to provide a reason for using this format with ‘easier to understand’ being a common accepted answer. It does seem to be a common misunderstanding that hexadecimal uses less memory than binary / machine code.

Whilst hexadecimal memory addressing is included in the specification students struggled to identify the lowest and highest memory addresses that would be available. This was a question part that required the student to identify the op-code and operand part of the instruction and to know that hexadecimal characters range from 0 to F. Responses written in hexadecimal were appropriate; there is no requirement to convert from hexadecimal to denary in COMP2.

Asking students to provide a situation where it would be appropriate to use a low level language gave rise to lots of varied answers. Strong students identified programs that needed to be optimised for speed of execution or object code memory size. Whilst parts of an operating system might be written in a low level language, this was not awarded a mark on its own. However, students who identified device drivers or code used to directly manipulate hardware were awarded a mark.

The idea of compilers translating all in one go and interpreters translating and executing line by line is well known. Students also often secured a mark by stating that a compiler produces object code.
Marks were not awarded when students simply repeated parts of the question for example: ‘a compiler compiles...’ or, ‘an interpreter interprets....’. It was quite common to see, ‘an interpreter compile...’

Across the papers it was evident that a group of students were confused over the terms machine code, object code and source code. It was also evident that some students are confused over machine code, assembly code and high-level languages.

It should be noted that students need to be careful when writing about execution speed. It was common to see students write about compilers executing code fast. Students need to be aware that it is the machine code of the compiled program that executes faster compared with interpreting the same program.

Q6.

Part (a) about the limitations of assembly language had many candidates scoring at least one mark. A misconception amongst some candidates was that assembly language could not be used to code complex programs when in reality any program could be created in assembly language. Candidates who stated only that it would take a long time to program in assembly language did not secure the mark unless it was clear that it would take longer than using a HLL. Stating that assembly language is hard to read is not enough to secure a mark, but candidates were rewarded if they identified that assembly language is harder to understand.

Even though part (b) has been asked in various forms before, many candidates failed to gain any credit. Too often candidates gave weak and vague answers that did not have enough detail to secure a mark. It is clear that some candidates are not clear over the differences between the terms ‘program’, ‘source code’, ‘object code’ and ‘executable’. Candidates also need to be careful when they merely state that something will be ‘quicker’ or ‘faster’. This was not enough for this question; candidates needed to make clear that it was the program that would execute more quickly.

A few candidates stated that, ‘it makes it hard to copy,’ which is not true as you can copy object code.
The point that these candidates were probably trying to make is that it is harder to reverse engineer the object code. In the same way, many candidates wrote about a compiler ‘protecting’ the source code. It is not really clear what ‘protecting’ means in this sense.

Q7.

This question covered another aspect of computer hardware, the fetch–execute cycle, and why programs are written in high level languages. The table was correctly completed in the majority of cases with the labelled parts of the processor. However some answers simply gave the acronyms rather than the full names of the registers, which the question had clearly asked for. Questions about the ‘decode and execute’ parts of the cycle have not been asked before this showed in the answers with many candidates describing the fetch part of the cycle and not what was asked. The part of the question concerning why programmers prefer instructions in hexadecimal compared to binary was often answered by saying it takes less storage space which clearly it does not. The answer to the question about the program translator was almost universally well known. When answering the final part, worth three marks, about why programs re written in a high level language, candidates often gave only two reasons and automatically failed to gain one mark. Answers stating that it is, ‘like English,’ were simply not enough and were marked accordingly; candidates needed to add to this to gain a mark.

Q8.

Many candidates answered the question that they would like to have been asked about compilers and interpreters, rather than the question that was actually asked. Responses were often no more than repetition of bookwork about how compilers and interpreters work and showed little understanding.

Responses to part (a) usually described the process of compilation, not why the compiler had been used by the software company. Answers often compared compilers and interpreters regarding their role at finding program errors – this is not what the question was asking. Nevertheless, some candidates tackled the question well, gaining full marks in this part.

Part (b) was also answered with reference to what an interpreter does regarding how it is easier to spot and correct errors, missing the point of the question entirely. A small but significant portion of the answers were left entirely blank suggesting that the topic was not well known by the candidates. A common incorrect answer was that "the interpreter allowed a script to download line by line so that it could run line by line immediately rather than having to wait for a full compilation to download and then execute". Correct responses identified that interpreted code could be run on any type of processor which is important when programs are run in a browser as the program authors could not know what type of processor the end user would have.

Q9.

The responses to this question were similar to those made on the same topic in January 2009.
Most candidates were aware that an assembler is required to translate assembly code into machine code. Fewer candidates knew that a compiler translates a high level language statement into machine code. An interpreter will not produce machine code; it merely interprets and executes high level language statements.

Q10.

This question clearly showed which candidates had studied this topic. Answers mostly either gained full marks or hardly any. Candidates are not expected to be able to write lengthy low level code. They are expected to understand how a fairly simple assignment statement in a high level language would be represented in assembly code. Most candidates knew that an assembler is required to translate assembly code statements into machine code, but rather fewer candidates knew that only a compiler translates a high level language statement into machine code. An interpreter will not produce machine code; it merely interprets and executes high level language statements.

Q11.

(a) This was the first time candidates had been asked for an essential feature of a program development environment for the creation of program code and a correct answer was given only by the more able candidates.

(b)(c)(d)

These questions should have been straightforward bookwork-type questions as candidates were asked to describe how a compiler and an interpreter process a source program and there were many ways in which the candidates could score the marks. Candidates often let themselves down by their poor communication skills and answers such as, 'read a whole program and produce an .exe out of this,' said nothing other than a re-wording of the stem of the question. The word 'it' – commented on in previous Reports on the Examination - was again in evidence for part (d) and the answer was then unclear as to whether the candidate was referring to the compilation process or the running of the executable code.

Many answers stated the compiler output as being the executable code, despite this being stated in the question.

Q12.

Parts (a) and (b) were generally well answered but less so (c) and (d). Many candidates gave a vague answer for the assembler output as 'errors' and this was considered insufficient.

Very few candidates were able to store the maximum 4 marks in part (d). There were very few convincing answers, and common misconceptions were that the program is executed from the hard disk, with no mention of program instructions. Often candidates generally described the need for the program to be loaded into main memory, without any further explanation.

Q13.

(a)(b)

Candidates generally scored well on these parts which was encouraging. At this level is it likely that candidates will have had little or no practical experience of low level programming. Therefore it was sufficient that the candidate was able to recognise the various generations and appreciate likely applications for which each would be used to score the marks.

(c) Well answered.

(d) This question was a variation from those in previous papers where candidates had been asked to compare compiler and interpreter software. There were many different points which the candidates could have made to secure the two marks. Most common correct answers suggested 'the interpreter translates each statement' – although answers which went on to suggest 'into machine code' were considered to have talked themselves out of this mark - the concept that it does the translation 'line by line'. The mark scheme was considered generous but, despite this, very few candidates scored the maximum 2 marks.

Candidates must read the question. There were often statements such as 'each HHL statement translates to several LLL statements' which suggests knowledge but did not answer this question.

(e) This question was asked for the first time, but something more than 'because they are not compatible' was looked for. The assembler software is specific to a particular processor (architecture) is what was wanted.

Q14.

(a) Many candidates were able to explain that functions always return a value but few candidates were able to distinguish this from the way a procedure behaves.

For candidates who had covered this theory in a practical context this was an easy two marks. Candidates should have been exposed to a subset of the functions available in their programming language. The final part of the question stem "…or when using a generic software package" was intended to help the weaker candidates in triggering some of the functions they would have used; unfortunately, candidates often gave answers describing features of a generic software package.

(b) This question was generally well answered, although it was noticeable that the standard of answers varied between centres. Candidates who found the question easy were undoubtedly those who had practical experience of using functions which required none, one or two parameters when used.

(c) The most popular answer was to use identifier names for constants, followed by procedures and functions.

(d) This was well answered with most candidates able to score marks. The key word in the question stem was “advantage” and so answers required more than just a description of a compiler and an interpreter.

Q15.

The majority of candidates were able to recognise (a) and (b) as assembly language and machine code respectively.

(a) Very few candidates were able to suggest the numbers represented memory addresses, despite previous CPT1 papers asking candidates to explain the stored program concept where the basic concept of a program resident in addressable memory is fundamental to any computer system.

(b) Candidates usually failed to get the mark by being too vague with explanations such as ‘Fig 1 was the same as Fig 2’, or ‘did the same thing’.

The required answer was the fundamental relationship between assembly language and machine code; namely a one-to-one correspondence between the instructions.

(c) This was poorly answered.

Q16.

(a) The generations of programming languages were well understood but some candidates only gave high level language as an answer to part (iii) without giving an example.

(b) Candidates found it difficult to find two distinct points. There were many answers that received only one mark. There were also many answers that simply said easy to read. It is expected that candidates will be able to produce more explicit answers at this level.

(c) Although most candidates were able to provide some idea of the difference between a compiler and an interpreter few were able to explain the process of either compilation or interpretation clearly. Many candidates were under the impression that a compiler executed the program.

(d) Many candidates associated the use of a compiler with a completed program or its distribution. Fewer were able to give a valid reason. Some seemed to be under the impression that a compiler compiles code faster. They did not make it clear that the compiled code executes faster. Many candidates correctly identified that an interpreter may be used when debugging code. Far fewer were able to indicate why. There seemed to be a common misunderstanding that only inexperienced programmers use interpreters.

Q17.

This is not a new question. However, candidates provided very confused answers.

(a) A compiler translates a high level language program into object code. It does NOT execute it. An interpreter translates a line of code as it executes. It does NOT produce object code. Many candidates wrongly seemed to be under the impression that binary code is synonymous with machine code.

(b) Some very weak answers were given such as ‘a compiler is used for large programs, an interpreter for small programs’ – sometimes the other way round!

Creditworthy answers suggested the use of a compiler to protect source code or for distribution after development is completed and the use of an interpreter during program development while debugging.

(c) A significant number of candidates had no idea that an assembler translates assembly code programs. Many wrongly assigned it the job of linker/loader or even for reverse engineering.

Q18.

(a) Candidates often gave two high-level languages, which were inappropriate responses. The question asked for translators and the only correct responses were assembler and compiler/interpreter respectively.

(b) Most candidates gained one mark for the obvious answer that it is easier to write programs in a high level language, without being entirely convincing that they knew what they were talking about. Fewer candidates could list a second characteristic, e.g. that such languages are problem oriented rather than machine oriented, or that they support data structures and structured statements. The response 'can use English words' was not enough to gain credit.

(c) Many candidates gained credit for stating that the use of named constants makes a program easier to maintain or understand.

(d) Those who had, clearly, studied a high level language had little difficulty with this part. Nearly all candidates gained the mark for part (i) since they had heard of a programming language. The responses to (ii) and (iii) however showed that far too many candidates do not get enough exposure to practical programming in a high level language. HTML is not appropriate here.

Q19.

(a) (i) Most candidates correctly named one or more buses. The term 'bus' was enough to gain the mark, but some candidates still referred to a 'memory bus' which did not gain credit.

(ii) Very few candidates seem to appreciate that secondary storage is used to save programs and data when they are not in use. Most referred to backup copies, which may well be saved on secondary storage, but is not the primary purpose of such storage.

(ii) The fetch-execute cycle was very well explained by a few candidates, though in too much detail by some others (e.g. by those who had presumably just studied machine architecture for CPT4). The majority of answers involved fetching data from memory and then executing data which gained no credit. Correct responses stated that an instruction is fetched from main memory, decoded and executed by the processor. At this machine level of operation, the term 'information' is not appropriate.

(b) (i) Assembly languages are second generation programming languages. The term 'assembler' was accepted this time, but candidates should be able to distinguish between the two terms and appreciate that the assembler is the translator, which converts the assembly language program into machine code.

(ii) Many candidates lost a mark by wrongly stating that a compiler or interpreter converts an assembly language program into machine code. The terms 'source code' and 'object code' belong to the translation of high level language programs by compilers and should not be used in the context of second generation languages.

(iii) Very few candidates could explain what the term ‘imperative’ meant in the context of high level languages. Most thought it meant important or problem oriented. Of those who were on the right track, some then confused the definitions of imperative and declarative languages. A correct response explained that the computer executes instructions in programmer-defined sequence. It was not acceptable to equate a programmer with a user.

(iv) A great many different languages quoted here gained credit. However, Prolog or HTML were not acceptable examples.

(v) Few candidates could state that one high level language statement would translate into one or more machine code instructions. Some candidates denied that there was any relationship.

(vi) This was a well answered question even for middle-scoring candidates, though the answers were sometimes a little vague. ‘Hard to learn’ and ‘debug’ were probably the most common answers which gained credit.

Q20.

This question was generally done poorly. Most candidates read into the question that a comparison with machine code or assembly code programming was required. This was clearly not appropriate.

(a) Candidates often quoted the detail of the question again as an answer, but many gained some credit for answers including that it was easier to debug a structured program because it would be easier to find errors in procedures. The misconception that structured code makes it easier for the compiler was rather widespread.

(b) Candidates do not know what features are, with many carrying on answering as in part (a). Often no distinction between the features of an editor used to type a program and the features of the actual programming language itself were made. Most candidates thought that programming in a high level language was almost like writing in English, for which no credit was given. Good answers included use of procedures/functions, English-like keywords, data types, data structures, local variables, parameters, being able to choose sensible names for identifiers and the existence of constructs such as IF..THEN..ELSE, REPEAT..UNTIL etc.

(c) The distinction between a compiler and an interpreter was clearly taught well in some centres and not in others. The fact that a compiler will translate the whole source code into object code, which can then be executed independently of the compiler seemed to evade many. That an interpreter translates one line at a time as it executes without producing object code was also only fully understood by few.

Q21.

This question was badly done - one suspects that few, if any, candidates have actually seen any low- level code or discussed the various programming languages in any depth. The instruction to “contrast” was widely ignored, many candidates simply brain-dumping a few phrases about high-level languages followed by a (usually inaccurate) sentence or two on low-level languages. A large number wrote, “high-level is close to English and low-level close to machine code”, although why candidates think that, for example, **while (*p++*(object..>name++));**
(a perfectly valid C++ instruction) is more English than **ADD EAX, [total]**

(ditto Intel 80386 Assembler) is not immediately obvious.

A fair number of candidates could give one correct contrast, such as machine specific / portable or many- one/one-one instruction translation, although few managed both and the examiners had to struggle to find what was being contrasted with what. Many believe, wrongly, that a low-level program occupies less memory than a high-level one, not necessarily true since in neither case is the translator present at run-time. For the second part, programs such as device drivers were the most popular, although few candidates appeared to understand why - the reason, of course, being that in such code precise control of timing is necessary to achieve synchronisation between devices. Some gave operating systems or BIOS as examples, on the grounds that they must be written in a low-level language because until they have executed no high-level translator can run - although in fact such systems are invariably written and compiled on other systems, usually in C.

Q22.

Candidates mostly knew the difference between an interpreter and a compiler but some found it hard to relate their knowledge to the question. In part (a) good candidates made the advantages of the interpreter during development very clear. Correct answers referred to the time saved during debugging as there is no need to re-compile after a change to the source code or the idea of being able to execute just part of the source code to help find errors. Weaker candidates tended to say “it is easier” and scored no marks.

Part (b) produced good answers from the best candidates dealing with the idea that an executable file is produced so there is no need for translation software at run-time. Many also realised that object code is less likely to be tampered with by users than source code. A minority of candidates had very little knowledge of the function of the translation software.

Q23.

Even candidates who scored well overall confused assemblers with interpreters.