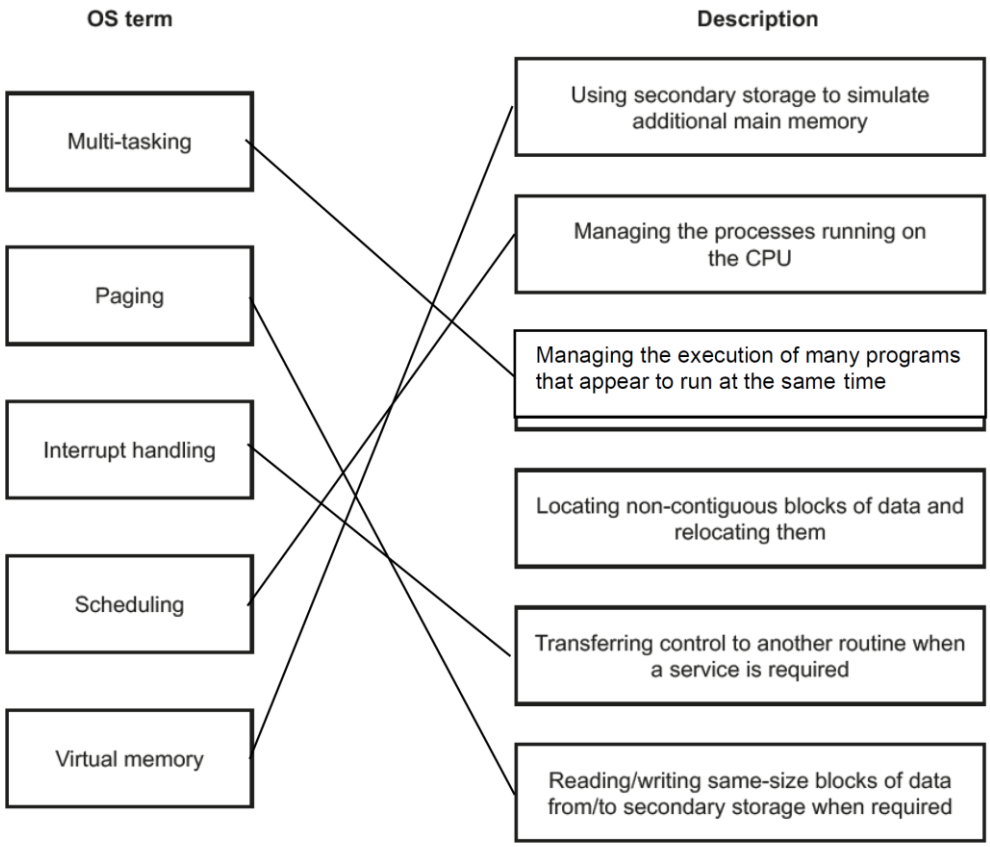
Question No. 1

a)	**One** mark for each correct line from Operating System Term to Description  	**5**
b)	**One** mark for each correct statement **(Max 4)** • An interpreter examines source code one statement at a time • Check each statement for errors • …If no error is found the statement is executed • …If an error is found this is reported and the interpreter halts • Interpretation is repeated for every iteration in repeated sections of code/in loops • Interpretation has to be repeated every time the program is run	**4**

Question No. 2

a)	*An interrupt* a signal/message from some device to indicate that some event has occurred //the device is seeking the attention of the processor [2]	**2**
b)	Identify the source of the interrupt Disable all interrupts of a lower priority Save the contents of the PC Save the contents of the other registers ... Onto the stack Load and run the appropriate ISR code Restore the registers From the stack (stack mentioned 1 mark only ...) Enable all interrupts Continue execution of the interrupted process	**6**
c)	- Partitioning - Memory is divided into partitions - One or more programs loaded into each partition - Different partitions used for different types of job - Partitions can be of fixed size or dynamic - Programs are scheduled when partition has space for whole program OR ... - Paging / Virtual memory - The program is divided into a number of pages // The main memory is divided into a number of page frames (of the same size) - Not all pages of the program need to be initially loaded - Pages swapped in/out of memory as required - use of page table OR - segmentation - Programs are divided into segments by the programmer - Not all segments are initially loaded // segments are loaded as and when required during execution - segments can be of varying size	**6**

Question No. 3

<table>
<tr><td>a)</td><td>The 245th page frame from the start of memory
// the 245th page frame from some base address</td><td>1</td></tr>
<tr><td>b)</td><td>Flash memory // magnetic disk // hard drive</td><td>1</td></tr>
<tr><td>c) i)</td><td>Time of entry (NOT time in memory)</td><td>1</td></tr>
<tr><td>c) ii)</td><td><table><tr><th>Page</th><th>Presence Flag</th><th>Page frame address</th><th>Additional data</th></tr><tr><td>4</td><td>1</td><td>542</td><td>12:07:34:49</td></tr></table>[1 +1 + 1]</td><td>3</td></tr>
<tr><td>c) iii)</td><td>Number of times the page has been accessed</td><td>1</td></tr>
<tr><td>c) iv)</td><td><table><tr><th>Page</th><th>Presence Flag</th><th>Page frame address</th><th>Additional data</th></tr><tr><td>3</td><td>1</td><td>132</td><td>0</td></tr></table>[1 +1 + 1]

Accept only zero for ‘additional data’</td><td>3</td></tr>
<tr><td>d)</td><td>For example:
Longest resident: page in for lengthy period of time may be being accessed often … so not a good candidate for being removed

Least used: a page just entered has a low least used value …
so likely to be a candidate for immediately being swapped out</td><td>4</td></tr>
</table>

Question No. 4

<table>
<tr><td>a)</td><td>The page is present in memory
Loaded at / stored /present in page frame 542 // its memory address is 542</td><td>2</td></tr>
<tr><td>b) i)</td><td>Next instruction is first instruction in Page 6
Page 6 is not present in memory
Instruction can only be executed if present in memory
Program cannot continue until Page 6 is loaded</td><td>2</td></tr>
<tr><td>b) ii)</td><td>When there is an attempt to load an instruction for a page not in memory
A page fault occurs // Page 5 finishes …
this generates an interrupt
ISR code is executed
Causes the OS to load page 6 into memory</td><td>3</td></tr>
<tr><td>c) i)</td><td>Time of entry (NOT time in memory)</td><td>1</td></tr>
<tr><td>c) ii)</td><td><table><tr><th>Page</th><th>Presence Flag</th><th>Page frame address</th><th>Additional data</th></tr><tr><td>6</td><td>1</td><td>221</td><td>12:07:34:49</td></tr></table>[1 + 1 + 1]</td><td>3</td></tr>
<tr><td>c) iii)</td><td>When the procedure call is made – Page 1 is swapped out and Page 3 is swapped in
At the end of the procedure call – Page 3 is swapped out and Page 1 is swapped in
Page 1/3 is always in memory shortest amount of time
The entire sequence is repeated for every iteration</td><td>3</td></tr>
<tr><td>c) iv)</td><td>Thrashing // continually swapping pages</td><td>1</td></tr>
</table>

Question No. 5

a)	**1 mark** per bullet point **1 mark** for identifying the state, **max 2** for description M**ax 3** marks for each state □ Ready □ The process is not being executed □ The process is in the queue… □ … waiting for the processor's attention / time slice □ Running □ The process is being executed by the processor □ The process is currently using its allocated processor time / time slice □ Blocked □ The process is waiting for an event … □ … so it cannot be executed at the moment □ …e.g. input/output	**6**
b)	For up to 2 maximisation techniques for each of memory and disk Max 2 for Memory, Max 2 for disk if no descriptions are given **1 mark** for identification of maximisation technique**, 1 mark** for description, **1 mark** for further description or information about improvement to **max 4 for memory** **Memory** □ Moving frequently accessed instructions to cache (1) for faster recall (1) as SRAM is used rather than DRAM for cache (1) □ Making use of virtual memory (1) with paging or segmentation (1) to swap memory to and from a disk (1) □ Partitioning memory (1) dividing main memory into static/dynamic partitions (1) to allow for more than one program/task to be available //multiprogramming (1) □ Removing unused items/tasks from RAM (1) by marking a partition as available (1) as soon as the process using it has terminated (1) **1 mark** for identification of maximisation technique, **1 mark** for description, **1 mark** for further description or information about improvement to **max 4 for disk** **Disk** □ Disk caching (1) a disk cache holds data that is frequently transferred to/from the disk (1) the cache can be held on disk or in RAM (1) □ Compression utility (1) decreasing the size of a file stored on disk (1) in order fit more / larger files on the disk (1) □ Defragmentation utility (1) files are rearranged to occupy contiguous disk space (1) this reduces the time taken to access files// decreases latency (1)	**6**

Question No. 6

a)	**1 mark** per bullet point Page: • Virtual Memory is divided into blocks of a fixed size Page frame: • the main memory is divided into page frames of the same size as a page Page table: • the Page (Map) table shows the mapping of pages to page frames	**3**
b)	**1 mark** per bullet point to **max 3** • To allow multiprogramming / multitasking to take place • To ensure fair usage of the processor • To ensure fair usage of peripherals • To ensure fair usage of memory • To ensure higher priority tasks are executed sooner • To ensure all processes have the opportunity to finish	**3**
c)	A **signal** from a software **source** or hardware device seeking the **attention of the processer**	**1**
d)	**1 mark** per bullet point in the order given • JOB32 • JOB42 • JOB42	**3**

Question No. 7

a)	1 mark per bullet point to **max 2** • Disk / secondary storage is used to extend the RAM / memory available • … so CPU can access more memory space than available RAM • Only part of program / data in use needs to be in RAM • Data is swapped between RAM and disk	**2**
b) i)	1 mark per bullet point to **max 4** • Divide memory / RAM into frames • Divide virtual memory into blocks of same size called pages • Frames / pages are a fixed size • Set up a page table to translate logical to physical addresses • Keep track of all free frames • Swap pages in memory with new pages from disk when needed	**4**
b) ii)	First-in-first-out // least-recently-used page // least-used-page	**1**
b) iii)	1 mark per bullet point to **max 2** • Pages are required back in RAM as soon as they are moved to disk • There is **continuous** swapping (of the same pages) • No useful processing happens // deadlock • … (because) pages that are in RAM and on disk are inter-dependent • … (nearly) all processing time is used for swapping pages	**2**

Question No. 8

a)	For each task: **One** mark for correct state **One** mark for suitable reason • Temperature: ready • Reason: waiting for the 10 seconds to be finished • Windspeed: running • Reason: it is currently recording the windspeed • Sending: blocked • Reason: it is waiting for the internet connection	**6**
b)	Any **four** from: • Uses a timer // uses two timers • Each timer is continually checked to see if 10 seconds has passed • …if it has, an interrupt is sent to the OS • …OS checks interrupt status • …and may pass control to the interrupt handling routine • (If 10 seconds has passed) then the ISR switches process state to running/ready • When finished it passes control back to OS • The timer is restarted	**4**

Question No. 9

a) i)	1 mark for each bullet point to **max 2** Keyword table: • The reserved words used • The operators used • Their matching tokens	**2**
a) ii)	1 mark for each bullet point to **max 2** Symbol table: • Identifier name used • … the (data) type • … role, e.g. variable, constant, array, procedure / scope • Location (marker) // value of constant	**2**
a) iii)	1 mark per bullet point to **max 2** • Keywords / operators are looked up (in the keyword table) • Keywords / operators are represented by tokens • Identifiers are looked up in (the symbol table) • Identifiers are converted to locations / addresses • Used to create a sequence of tokens (for the program)	**2**
a) iv)	The white space removed // redundant characters are removed // removal of comments // identification of errors	**1**
b)	1 mark per bullet point to **max 2** • Redundant code removed // fewer instructions required • Program requires less memory / storage space • Code reorganised to make it more efficient • Program will complete task in a shorter time	**2**

Question No. 10

a) — 3

1 mark for each correct row

Symbol	Token	
	Value	Type
Start	60	Variable
1	61	Constant
Number	62	Variable
Counter	63	Variable
12	64	Constant

b) — 2

1 mark for each circled section

60	01	61	51	62	4E	63	01	60	50	64	52	62	02	63	53

c) — 2

1 mark per bullet point to max 2:

- constructing parse tree // parsing
- checking the table of tokens to ensure that the rules/syntax/grammar of the language are/is obeyed
- producing an error report

d) i) — 1

shortens execution time of program// time taken to execute whole program decreases

Question No. 11

a) — 3

1 mark per row

Symbol	Token	
	Value	Type
Number1	60	Variable
Number2	61	Variable
Answer	62	Variable
10	63	Constant//Literal
0	64	Constant//Literal

b) — 2

1 mark for each circled section

51	60	51	61	51	62	4A	62	03	60	02	61	4B	52	63	4D	52	64	4C

c) i) — 1

(Code) Optimisation

c) ii) — 3

1 mark per bullet point:

- `LDD 236`
  `ADD 237`
  `ADD 238`
  `SUB 239` Copy the instructions
  `STO 235`

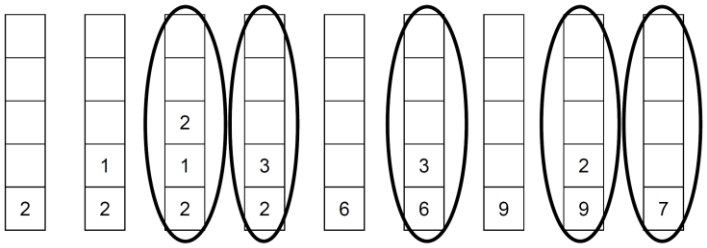
- Remove line 4 `STO 540` correct lines 3 and 6 in original code
- Remove line 5 `LDD 540` correct lines 3 and 6 in original code

c) iii) — 2

1 mark per bullet point:

- Code has fewer instructions/occupies less space in memory
- shortens execution time of program // time taken to execute whole program decreases

d) — 4



1 mark ← 1 mark → 1 mark

1 mark no operators on the stack anywhere

Question No. 12

a) — **3**

1 mark for **2** correct rows, **2 marks** for **3** correct rows, **3 marks** for **4** correct rows

Symbol	Token	
	Value	Type
Counter	60	Variable
0	61	Constant
Password	62	Variable
"Cambridge"	63	Constant
1	64	Constant

b) — **2**

60	01

First two cells given in question.

1 mark for next 3 cells

61	51	62

1 mark for the remainder

4E	4A	62	04	63	4B	51	62	4C	60	....

.... | 01 | 60 | 02 | 64 | 4F | 62 | 03 | 63 | 52 | 60 |

c) i) — **3**

1 mark per bullet point

- Removing the fourth line (LDD 238) …
- Changing operand for second ADD from 236 to 238 …
- … First three lines and last line unchanged

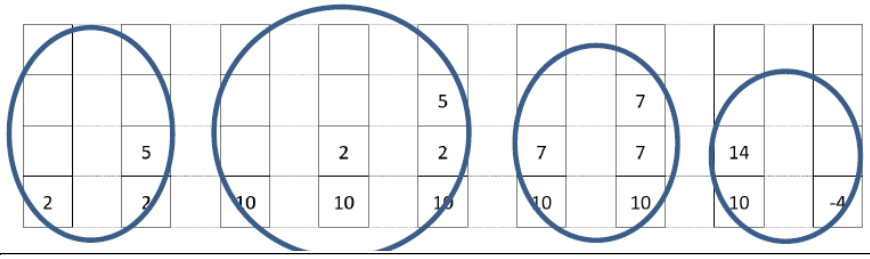
```
LDD 236
ADD 237
STO 236
ADD 238
STO 238
```

c) ii) — **2**

1 mark per bullet point (**max 2**)

- Optimisation means that the code will have fewer instructions
- Optimised code occupies less space in memory
- Fewer instructions reduces the execution time of the program

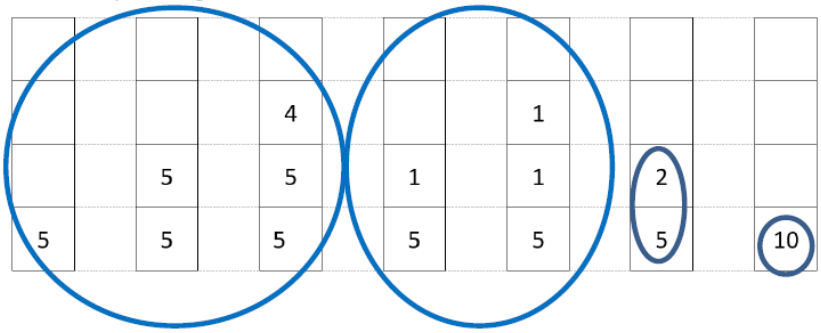
Question No. 13

a) i)	`a b * a b + c + -` **One mark** for `a b *` **One mark** for `a b + c + -`	**2**
a) ii)	**One** mark per ring max 4 	**4**

b)	**Two** marks all 3 elements of the expression are seen
	One mark if 2 elements of the expression are seen
	`(d * b) // d * b`
	`+(b - (c + d)) // + b - (c + d)`
	`/ a`
	
	One mark for fully correct expression
	`((d * b) + (b - (c + d))) / a`
	`// (d * b + b - (c + d)) / a`
		3

c)	Any **three** from • Evaluation does not need to use rules of precedence for operators • No need for brackets // infix may require the use of brackets • Enables evaluation in the sequence read / left to right • … no need to backtrack	**3**

Question No. 14

a) i)	`p p q - r + *` **One mark** for `p p q -` **One mark** for `r + *`	**2**
a) ii)	**One mark** per ring to **max four** 	**4**

b)	**Two marks** all 3 elements of the expression are seen
	One mark if any 2 elements of the expression are seen
	`(p * q) // p * q`
	`+(p - (q + r)) // p - (q + r)`
	`/ p`
	One mark for fully correct expression
	`((p * q) + (p - (q + r))) / p`
	`(p * q + p - (q + r)) / p`
		3

c)	Any **two** stages, **one** mark name, **one** mark matching description • Lexical Analysis (1) converts source code into tokens (1) • Code Generation (1) produces the object code (1) • (Code) Optimisation (1) improving efficiency of object code (1)	**4**

Question No. 15

a) i)	**One** mark for each correct marking point **(Max 2)** • Reverse Polish Notation provides an unambiguous method of representing an expression • … reading from left to right • …without the need to use brackets • …with no need for rules of precedence / BODMAS	**2**
a) ii)	**One mark** for identification of the data structure, **One mark** for a sensible reason **Either:** Structure: stack The operands are popped from the stack in the reverse order to how they were pushed **Or:** Structure: Binary tree A (binary) tree allows both infix and postfix to be evaluated (tree traversal)	**2**
b)	`a b - a c + * 7 /`	**1**
c)	`a / b * 4 - (a + b)`	**1**

d)	**1 mark** for correct structure
	1 mark for correct substitution
	
	`(a + b) / (c / d)`
	`(17 + 3) / (48 / 12)`
		2

Question No. 16

a) i)	35 is not a variable	**1**
a) ii)	:= is not an operator	**1**
a) iii)	9 is not a digit	**1**
b)	**1 mark** for each bullet point <operator>::= • == | > | < <number>::= • <digit><digit> <variable>::= • <letter> • |<letter><variable> <condition>::= • <variable><operator><number> • |<variable><operator><variable>	**6**

Question No. 17

a) i)	';' missing	**1**
a) ii)	'2' is not a variable	**1**
a) iii)	'e' is not a valid letter	**1**
b)	<assignment statement> ::= <variable> = <variable><operator><variable>; <variable> ::= <letter>|<letter><letter> |<letter><letter><letter> <letter> ::= a|b|c|d <operator> :: =+|-|*|÷	**6**
c)	<letter> | <letter><variable> // <letter> | <variable><letter>	**2**

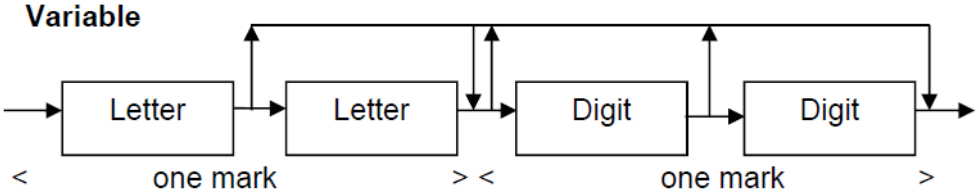
d) i)	debugging is faster / easier
	// can debug incomplete code
	// better diagnostics
		1

d) ii)	compiler produces executable version – not readable / no need for source code // difficult to reverse-engineer	**1**

Question No. 18

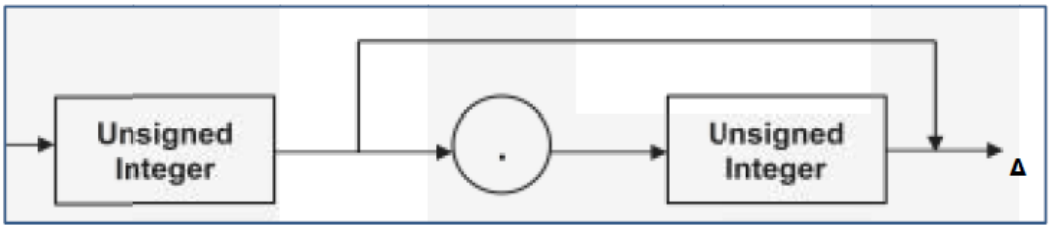
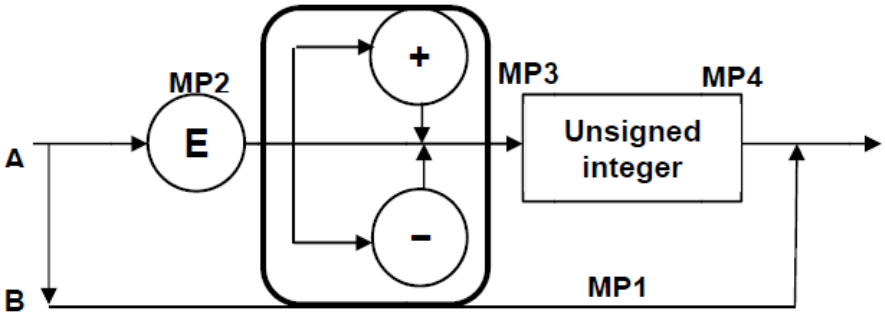
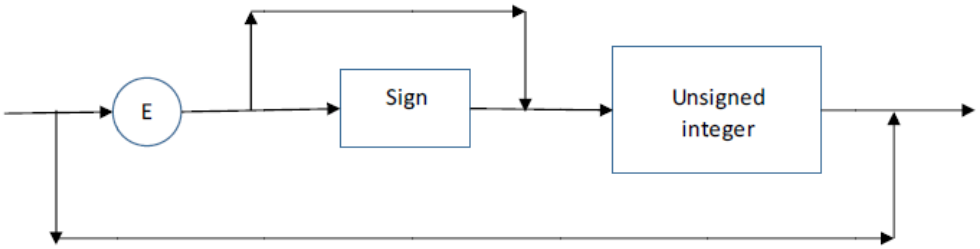
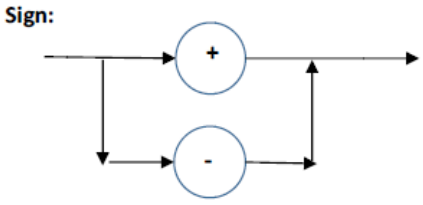
a) i)	Wrong assignment operator (should be ':=' not '=')	**1**
a) ii)	0 is not a digit	**1**
a) iii)	'B' is not a number	**1**
b)	<assignmentstatement> ::= <variable> := <variable><operator><number> <variable> ::= <letter><number> <number> ::= <digit>| <digit><number> <letter> ::= A|B|C <digit> ::= 1|2|3|4|5 <operator> ::= +|-|*|/	**6**

Question No. 19

a) i)	There should be a colon before the '=' sign	**1**
a) ii)	The second operand should be an unsigned integer and not a variable	**1**
a) iii)	A32 is not a variable, as a variable should be a letter followed by a single digit	**1**
b)	<assignment_statement> ::= <variable> := 1 <variable> <operator> <unsigned_integer> 1 <variable> ::= <letter> <digit> 1 <unsigned_integer> ::= <digit> | 1 <digit> <unsigned_integer> 1 <letter> ::= A | B | C <operator> ::= + | - | * | ^ 1	**6**
c)	**Variable**  Syntax diagram shows one or two letters 1 Syntax diagram shows zero, one or two digits 1	**2**
d)	<assignment_statement> ::= <variable> := <variable> <operator> <real> 1 <real> ::= <unsigned_integer> . <unsigned_integer> 1	**2**

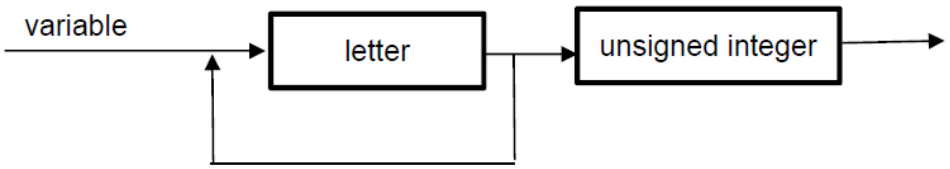
Question No. 20

a) i)	Because a valid unsigned integer can be two digits / one or more digits (1) Both 3 and 2 are digits (1)	**2**
a) ii)	Because a valid unsigned number can be an unsigned integer followed by a decimal point followed by an unsigned integer (1) 32 is an unsigned integer and 5 is an unsigned integer (because it is a digit) and there is a point in between (1) Alternative response for 2 marks, combination of order and validity: 32 is a (valid) unsigned integer, followed by a decimal point, and 5 which is another (valid) unsigned integer Validity mark must refer to 32 and 5	**2**
b)	<unsigned number> ::= **<unsigned_integer>** | (1) **<unsigned_integer>.<unsigned_integer>** (1) Accept order reversed: <unsigned_integer> ::= **<digit>** | (1) **<digit> <unsigned_integer>** (1) Accept **<digit> |<unsigned_integer> <digit>** If order reversed mark as above <digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 (1) Accept the list in any order, as long as all 10 digits included	**5**

c) i)	  **MP1**: Line to indicate exponent is optional (B line) (1) **MP2**: 'E' present at start of exponent (1) **MP3**: Optional '+' or '-' (1) **MP4**: Unsigned integer (1) Alternative correct answer: MP3 needs both the sign 'box' and the sign diagram for the mark  	4

c) ii)	<unsigned number> ::= <unsigned_integer > | <unsigned integer>.<unsigned_integer> (1) Accept any order | <unsigned_integer > <exponent> | <unsigned integer>.<unsigned_integer> <exponent> (1) Accept any order <exponent> ::= E <sign> <unsigned_integer> | E <unsigned integer> (1) <sign> ::= + | - (1)	4

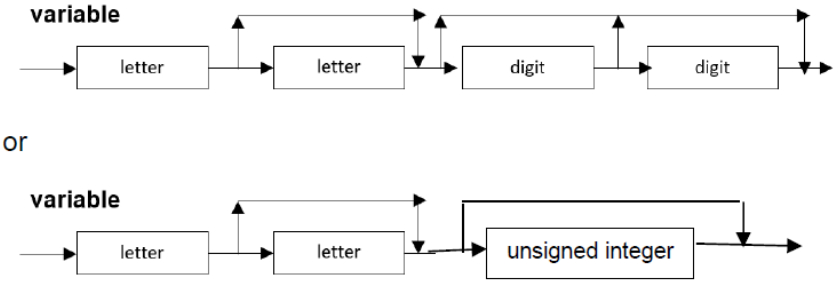
Question No. 21

a) i)	5 is not a variable	**1**
a) ii)	D is not a valid letter	**1**
a) iii)	There are two operators (only one is allowed) // three variables on the right hand side but only two allowed	**1**
b)	**1** mark for each bullet assignment: • <variable> = <variable><operator><variable>; variable: • <letter>| • <letter><unsigned integer> unsigned integer: • <digit>| • <digit><unsigned integer> operator: • + | - | * | / <assignment statement> ::= <variable> = <variable><operator><variable>; <variable> ::= <letter> | <letter><unsigned integer> <unsigned integer> ::= <digit> | <digit><unsigned integer> <operator> ::= + | - | * | /	**6**
c)	**1** mark per bullet • variable with arrow • followed by repeated letter • followed by unsigned integer and arrow 	**3**

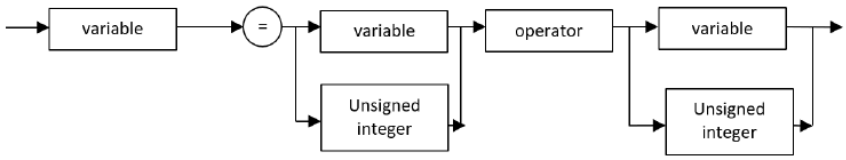
Question No. 22

a) i)	c4 is not a signed integer	**1**
a) ii)	10 is not a valid signed integer // 0 is not a valid digit/signed integer // only one digit allowed	**1**
a) iii)	wrong assignment operator // should be = not := // 6 is not a valid digit/signed integer	**1**
b)	1 mark per bullet assignment • <variable>=<variable><operator><signed integer> variable • <letter><letter> signed integer • +<digit> | -<digit> operator • ^ | \ <assignment statement> ::= <variable> = <variable><operator><signed integer> <variable> ::= <letter><letter> <signed integer> ::= +<digit> | -<digit> <operator> ::= ^ | \	**4**
c)	1 mark per bullet • <letter>| • <letter><variable> For example: <letter>|<letter><variable> <letter>|<variable><letter>	**2**

Question No. 23

a)	• X is not a variable • := should be = for an assignment statement • 5 is not a valid digit	**3**
b)	<assignment_statement> ::= <variable> = <variable><operator><variable> 1 |<variable> = <variable><operator><unsigned_integer> 1 <variable> ::= <letter><letter> 1 <unsigned_integer> ::= <digit>|<digit><digit> 1 <digit>::= 1 | 2 | 3 <operator> ::= + | - | * 1	**5**
c) i)	 or • two letters **and** two digits / one unsigned integer **and** arrows in and out seen • allows for one or two letters at start • zero, one or two digits // zero or one unsigned integer at end	**3**
c) ii)	**Three** marks for completely correct **Two** marks for four alternatives correct **One** mark for three alternatives correct <variable> ::= <letter> | <letter><digit> | <letter><digit><digit> | <letter><letter> | <letter><letter><digit> | <letter><letter><digit><digit> **Or** **Three** marks for completely correct **Two** marks for three alternatives correct **One** mark for two alternatives correct <variable> ::= <letter> | <letter><unsigned integer> | <letter><letter> | <letter><letter><unsigned integer>	**3**

Question No. 24

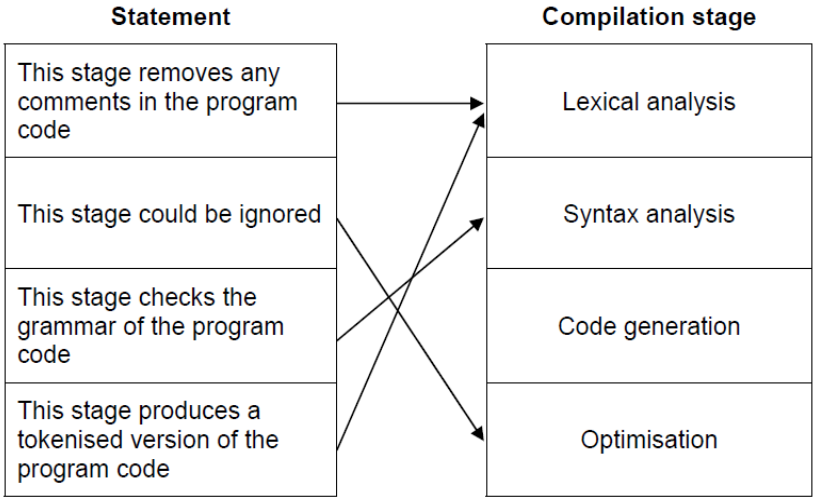
a)	• `Y` is not a variable • `:=` should be `=` for an assignment statement • `7` is not a valid digit	**3**
b) i)	`<assignment_statement> ::= <variable> = <variable>` 1 `<operator> <unsigned_integer>` 1 `<variable> ::= <letter> <digit>` 1 `<unsigned_integer> ::= <digit> | <digit><digit>` 1 `<operator> ::= + | - | *` `<digit>::= 1 | 2 | 3` 1	**5**
c) i)	assignment statement  Syntax diagram shows variable and unsigned integer in correct places 1 with **correct** arrows throughout 1	**2**

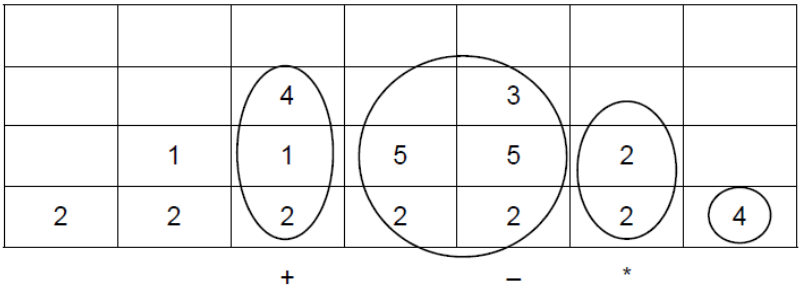
c) ii)	**Two** marks fully correct
	Or
	One mark `<assignment statement>::=` **and** any 2 correct alternatives
	Or
	One mark missing `<assignment statement> ::=` **and** rest correct
	
	`<assignment statement> ::=`
	`<variable> = <variable><operator><variable>`
	`|<variable> = <unsigned integer><operator><variable>`
	`|<variable> = <variable><operator><unsigned integer>`
	`|<variable> = <unsigned integer><operator>`
	`<unsigned integer>`
		2

Question No, 25

a)	**One** mark for each marking point (**Max 2**) • <character>::= • $|%|&|*|# **Complete answer** <character>::= $|%|&|*|#	**2**
b) i)	For example: $A9E3	**1**
b) ii)	**One** mark for each marking point (**Max 4**) • <password>::=<character> … • … <code> • <code>::= … • … <digit>|<capital_letter> • … |<digit><code>|<capital_letter><code> **Complete answer** <password>::=<character><code> <code>::=<digit>|<capital_letter>|<digit><code>|<capital_letter><code>	**4**

Question No. 26

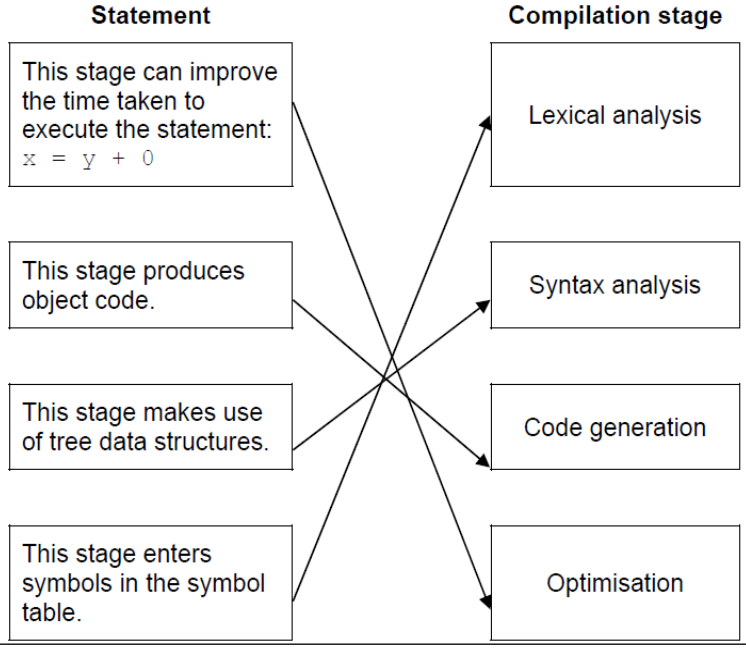
<table>
<tr><td>a)</td><td> 

1 mark for each correct line</td><td>4</td></tr>
<tr><td>b) i)</td><td>A B +
C D – *</td><td>2</td></tr>
<tr><td>b) ii)</td><td>A –
B / 4 *
C D – /</td><td>3</td></tr>
<tr><td>c) i)</td><td> 

1 mark per ring</td><td>4</td></tr>
<tr><td>c) ii)</td><td>x *
(w + z – y)
Order must be correct for both parts</td><td>2</td></tr>
<tr><td>c) iii)</td><td>No need for rules of precedence
No need for brackets
In RPN evaluation of operators is always left to right</td><td>2</td></tr>
</table>

Question No. 27

a)



1 mark for each correct line

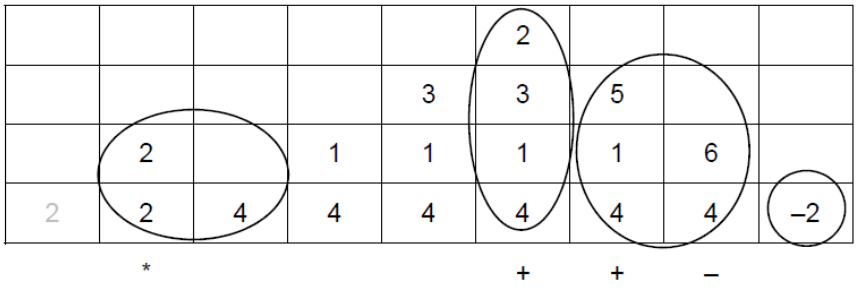
4

b)

P Q +
R S / –

2

c) i)



1 mark per ring

3

c) ii)

b * a
– (c + d + a)
Order must be correct for both parts

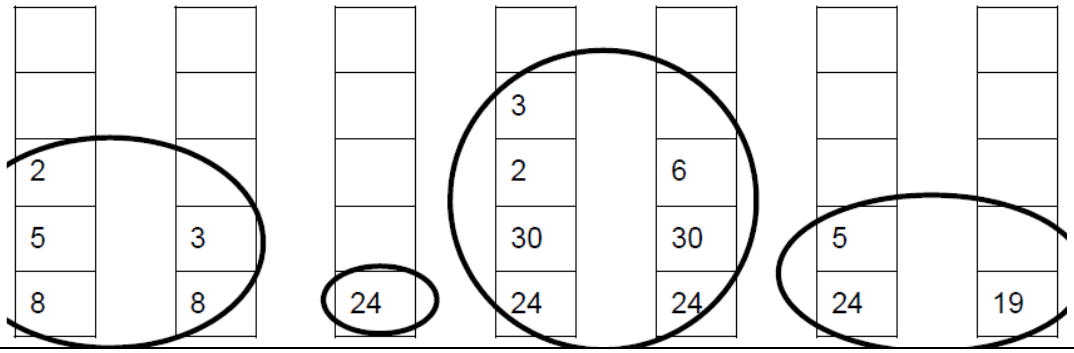
4

c) iii)

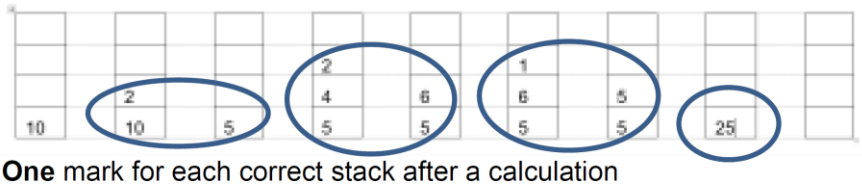
No need for rules of precedence
No need for brackets
In RPN evaluation of operators is always left to right

2

Question No. 28

a)	**1 mark** per bullet point (max 4) - Working from left to right in the expression - If element is a number PUSH that number onto the stack - If element is an operator then POP the first two numbers from stack … - … perform that operation on those numbers - PUSH result back onto stack - End once the last item in the expression has been dealt with	**4**
b)	**1 mark** per ring (not all stacks are shown) Do not allow operators in stacks Accept intermediate correct stack values 	**4**

Question No. 29

a)	`P Q + P Q - *` **One** mark for `P Q +` **One** mark for `P Q - *`	**2**
b) i)	 **One** mark for each correct stack after a calculation	**4**
b) ii)	`((P + Q) * M) – (R – P)` **One** mark for `((P + Q) * M)` **One** mark for `– (R – P)`	**2**
c)	Any **two** from: • Expressions are always evaluated left to right • Each operator uses the two previous values on the stack (except unary minus) • Description of pushing and popping on a stack	**2**