Chapter 16 Student Book Answers

16.1 What you should already know

1 Key OS management tasks

- memory management
- file management
- security management
- hardware management
- processor management

2 a) OS provides an environment in which programs can be run and gives an interface to human operators.

b) The interface is a simple set of dials/switches and the device carries out repeated (relatively simple) tasks not requiring an OS.

3 OS utility software

- hard disk formatter
- virus checker
- defragmentation software
- disk contents analysis/disk repair software
- file compression
- back-up software

4 Command line interface (CLI):

- It requires a user to type in instructions to choose options from menus, open software etc.
- There are often a number of commands that need to be typed in, for example, to save or load a file.
- The user has to therefore learn a number of commands (which must be typed exactly with no errors) just to carry out basic operations.
- It is also slow having to key in these commands every time an operation has to be carried out.
- However, the advantage of CLI is that the user is in direct communication with the computer and is not restricted to a number of pre-determined options.

Graphical user interface (GUI)

- It allows the user to interact with a computer (or MP3 player, gaming device, mobile phone, etc.) using pictures or symbols (icons) rather than having to type in a number of commands.

5 Interface software

- Provides communication with all input and output devices using device drivers.
- A device driver takes data from a file (defined by the operating system) and translates it into a format that the input/output device can understand.
- It ensures each hardware resource has a priority so that they can be used and released as required.

6 a) Program libraries are used for

- software under development using already-written code in program libraries
- benefits to a developer writing software constructed of library files such as dynamic link library (DLL) files
  - when software routines are written (e.g. a sort routine), they are frequently saved in a program library for future use by other programmers …
  - … therefore a program stored in a program library would be known as a library program
- we also have the term library routines to describe subroutines which could be used in another piece of software under development.

b)

- **Static libraries** – where software being developed is linked to executable code in the library at the time of compilation. The library routines would be embedded directly into the new program code.
- **Dynamic** – where software being developed is not linked to the library routines until actual run time (these are known as dynamic link library files or DLL). These library routines would be stand-alone files only being accessed as required by the new program. The routines will be available to several applications at the same time).

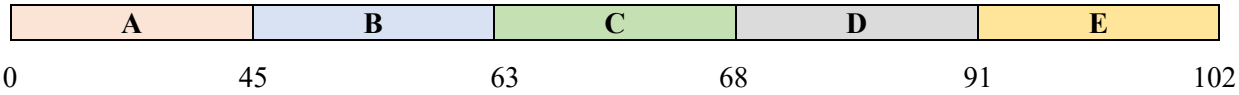
Activity 16A

1 A
2 B
3 B
4 C
5 E
6 D
7 E
8 C
9 B
10 A
11 E
12 C
13 E
14 A
15 E

Extension Activity 16B

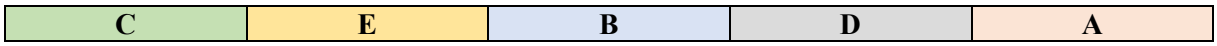
a)

FCFS



0 45 63 68 91 102

SJF

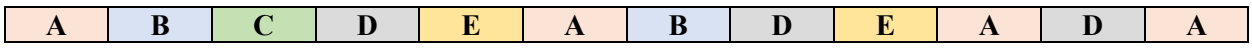


0 5 16 34 57 102

SRTF

0 8 10 15 20 31 42 65 102

Round Robin



0 10 20 25 35 45 55 63 73 74 84 87 102

b)

FCFS = 61.75 ms

SJF = 28.0 ms

SRTF = 12.75 ms

Round Robin = 62.25 ms

16.3 What you should already know

1 Assembler, compiler, interpreter.

2 **Assembler** – only type of language translator available for low level languages, makes writing programs easier as mnemonics can be used instead of hexadecimal.
Compiler – once translated the object program can be executed without the compiler present, this means that generally compiled programs are smaller in size so take up less storage space in primary and secondary memory, and the execution time taken by a task is usually shorter.

Interpreter – during program development, errors can be corrected as encountered during translation and/or execution leading to a reduced development time.

3 An integrated development environment often contains these features:

- an integrated editor to use instead of a separate program
- error diagnostics including auto-completion and auto-correction to prevent errors in spelling and syntax
- a runtime environment is used to test a program, this can include inspection of variables and executing statements one at a time
- prettyprinting to print out and easily readable copy of a program with the use of indenting and colour coding keywords etc.

Activity 16B

```
34 85 04 83
```

Activity 16C

1 Valid variables are:

`A1` letter (must be A, B or C) followed by digit (must be 1, 2 or 3)

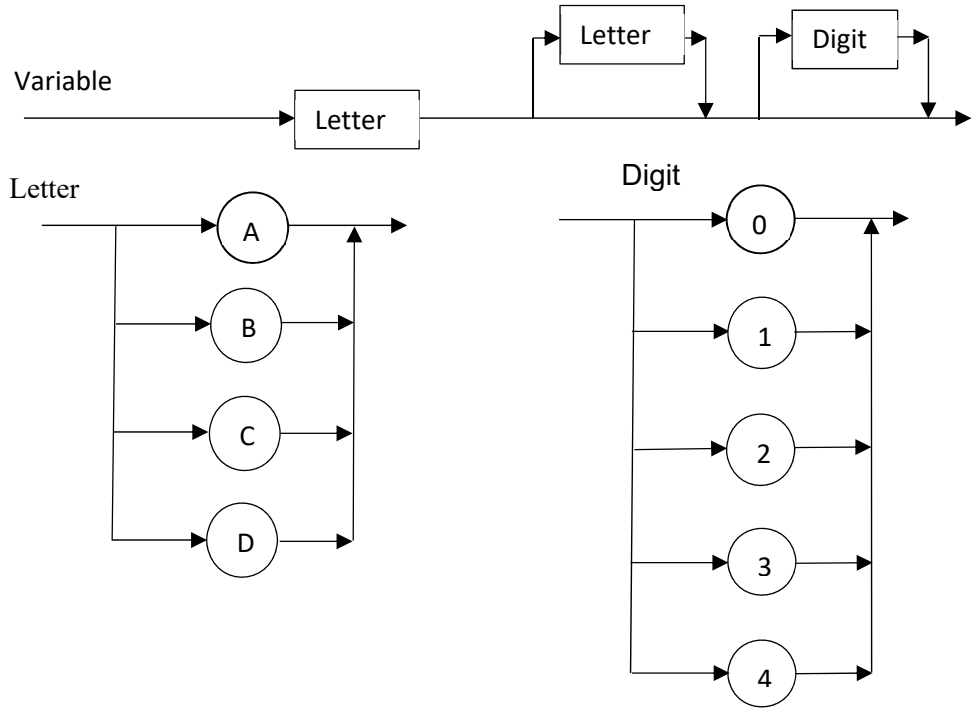
`C3` letter (must be A, B or C) followed by digit (must be 1, 2 or 3).

2 Invalid statements are:

`A1 = A1 + B1 + C1` can only have 2 variables on the right-hand side of the assignment statement

`A1 := C1 - C2` `:=` is not included in the syntax diagram for an assignment statement.

Activity 16D



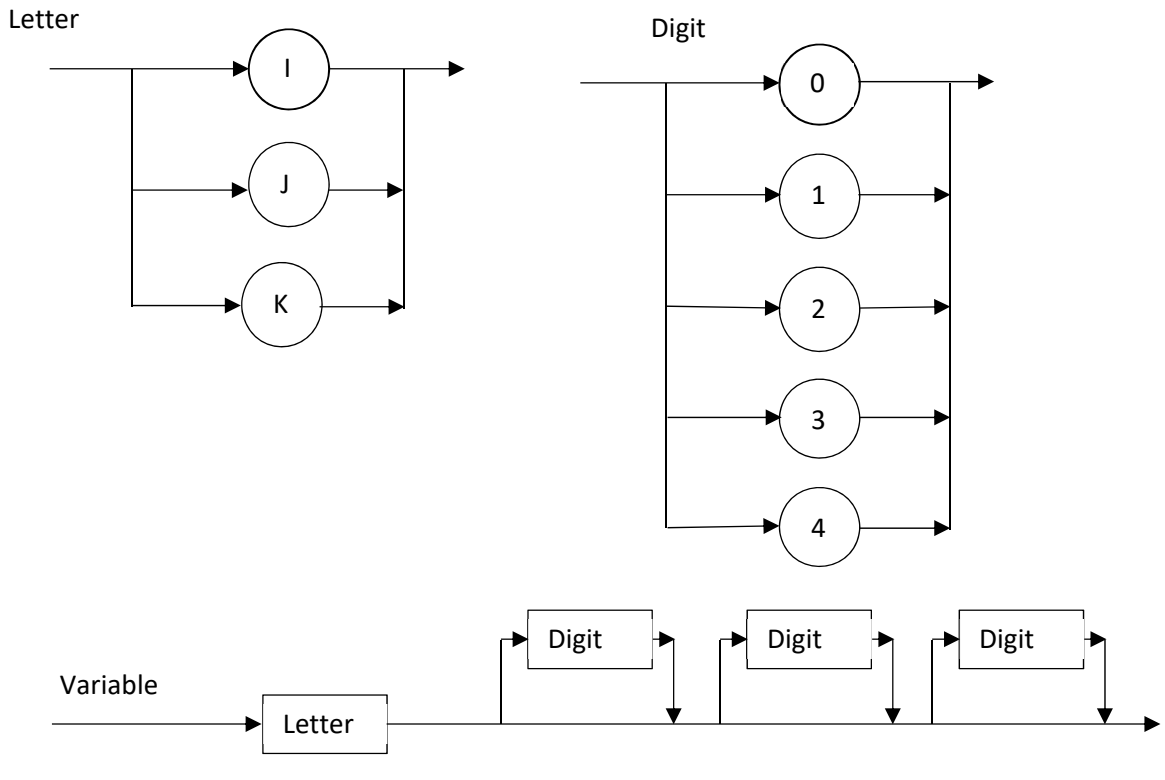
Activity 16E

```
<integer> ::= <digit> | <integer> <digit>
```

Activity 16F

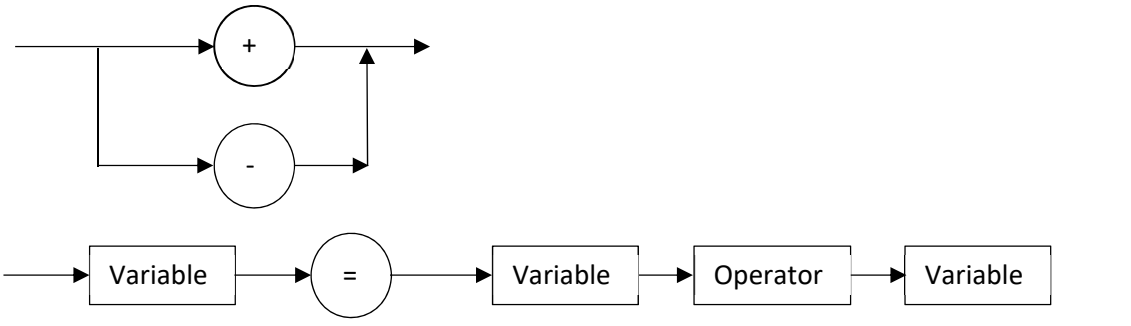
1 Lexical analysis, Syntax analysis, Code generation and Optimisation – see section 16.3.2

2 **a)**



b)
```
<variable> ::= <letter> | <letter> <digit> | <letter> <digit>
<digit>| <letter> <digit> <digit> <digit>
```

c) operator




```
<assignment> ::= <variable> = <variable> <operator> <variable>
```

3 a) i) `A B C D * + +`

ii) `A B C + + D *`

iii) `A B + D * A B - C * +`

b) i)

2			
5	10		
3	3	13	
7	7	7	20

ii)

5			
3	8	2	
7	7	15	30

iii)

			3		5		
3	2		7	4	4	20	
7	10	20	20	20	20	20	40

End of chapter questions

1 a) FCFS = 17.5 ms

b) SJF = 6.25 ms

c) SRTF = 6.0 ms

d) round robin = 12.75 ms

2 a)

- The page is present in **memory**.
- Loaded at/stored/present in page frame 542//its memory address is 542.

b) i)

- The next instruction is first instruction in Page 6.
- Page 6 is not present in memory.
- The instruction can only be executed if present in memory.
- The program cannot continue until Page 6 is loaded.

ii) When there is an attempt to load an instruction for a Page not in memory

- a page default occurs/Page 5 finishes …
- … this generates an interrupt
- ISR code is executed
- causes the OS to load Page 6 into memory.

c) i) time of entry

ii)

page	presence flag	page frame address	additional data
6	1	221	12:07:34:49

iii)

- 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 the shortest amount of time.
- The entire sequence is repeated for every iteration.

iv) Thrashing/continuously swapping pages

3 **a)** quantum

b) pre-emptive

c) virtual memory

d) low level scheduler

e) context switching

f) paging

g) non-preemptive

h) burst

i) segmentation

j) starvation

4 **a)** **i)** **From blocked to ready**

- Process is waiting for resource/IO operation to complete (blocked state).
- When IO operation completed, process goes into ready queue (ready state).

ii) **From running to ready**

- When process is executing, it is allocated a time slice (running state).
- The process is allocated time in processor.
- When time slice completed, interrupt occurs …
- … process can no longer use processor even though it is capable of further processing (ready state).

b) **A process cannot move directly from ready state to blocked state because**

- to be in blocked state, process must initiate some IO operation
- to initiate operation, process must be executing
- if process is in ready state, it cannot be executing/must be in running state.

c) **i)** exit/termination/completion

ii) when process has finished execution

d) **A low-level scheduler**

- decides which of processes is in ready state
- should get use of processor/be put in running state
- is based on position/priority
- is invoked after interrupt/OS call.

5 **a)** **Programs can access data from memory when using virtual memory because**

- program executes load process with a virtual address
- computer translates virtual address to give a physical address in memory
- if physical address not in memory, the OS loads it from the HDD
- computer then reads RAM using physical address and returns the data to program.

b) i) FIFO:

- When using first in first out (FIFO), the OS keeps track of all pages in memory using a queue structure
- The oldest page is at the front of the queue and is the first to be removed when a new page needs to be added.
- FIFO algorithms don't consider page usage when replacing pages; a page may be replaced simply because it arrived earlier than another page.
- It suffers from, what is known as, Belady's Anomaly where it is possible to have more page faults when increasing the number of page frames.

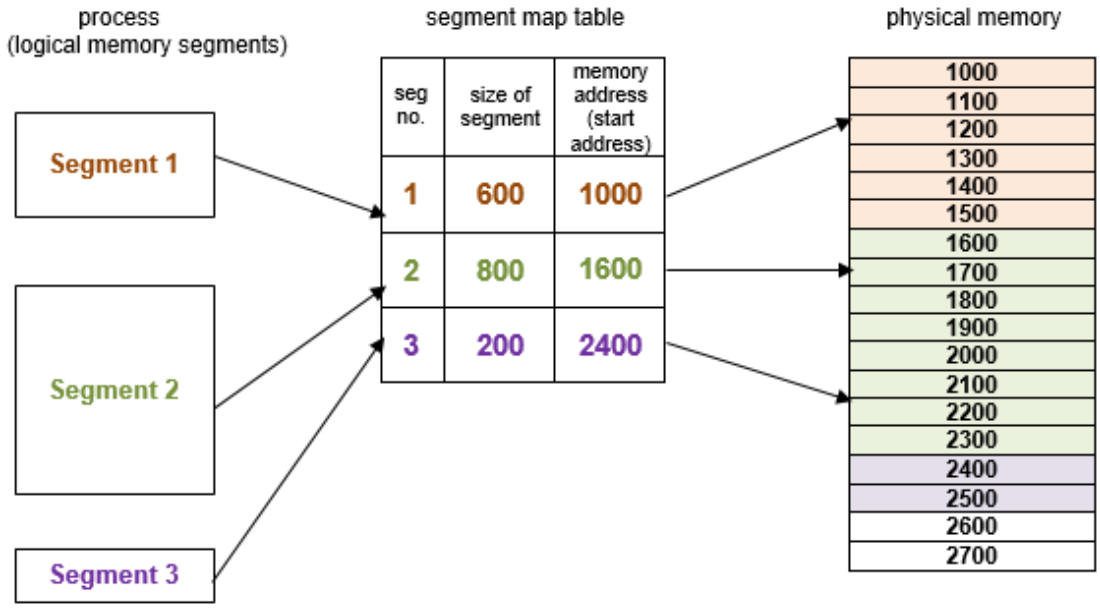
ii) OPR:

- Optimal page replacement looks forward in time to see which frame it can replace in the event of a page fault.
- The algorithm is actually impossible to implement; at the time of a page fault, the OS has no way of knowing when each of the pages will be replaced next.
- It tends to get used for comparison studies but has the advantage that it is free of Belady's Anomaly and also has the fewest page faults.

iii) LRU:

- With least recently used page replacement (LRU), the page which has not been used for the longest time is replaced.
- To implement this method, it is necessary to maintain a linked list of all pages in memory with the most recently used page at the front and the least recently used page at the rear.

6 a)



b) paging is fixed size; segmentation is variable size; pages are smaller than segments

c) Types of interrupts

- device interrupt (e.g. printer out of paper)
- exception (e.g. division by zero)
- trap/software interrupt (e.g. software needs to access/use a resource).

7 a)

Symbol	Token	
	Value	Type
Start	60	Variable
0.1	61	Constant
Counter	62	Variable
10	63	Constant

b)

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

c) i) Syntax analysis

ii) Checking the grammar of the program and producing an error report.

d) i) Minimise the time taken to execute the program.

ii) Replace 2 * 6 with the value 12

iv)

```
LDD 436
ADD 437
STO 612
ADD 438
STO 613
```

8 a) i) There should be a colon before the equals sign.

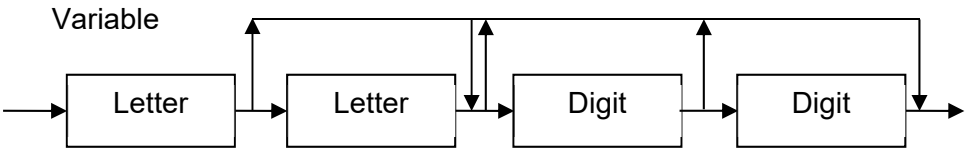
ii) The second operand should be an unsigned integer and not a variable.

iii) `A32` is not a variable, as a variable should be a letter followed by a single digit.

b)

```
<assignment_statement> ::= <variable> := <variable> <operator>
<unsigned_integer>
<variable> ::= <letter> <digit>
<unsigned_integer> ::= <digit> | <digit> <unsigned_integer>
<letter> ::= A | B | C
<operator> ::= + | - | * | ^
```

c)

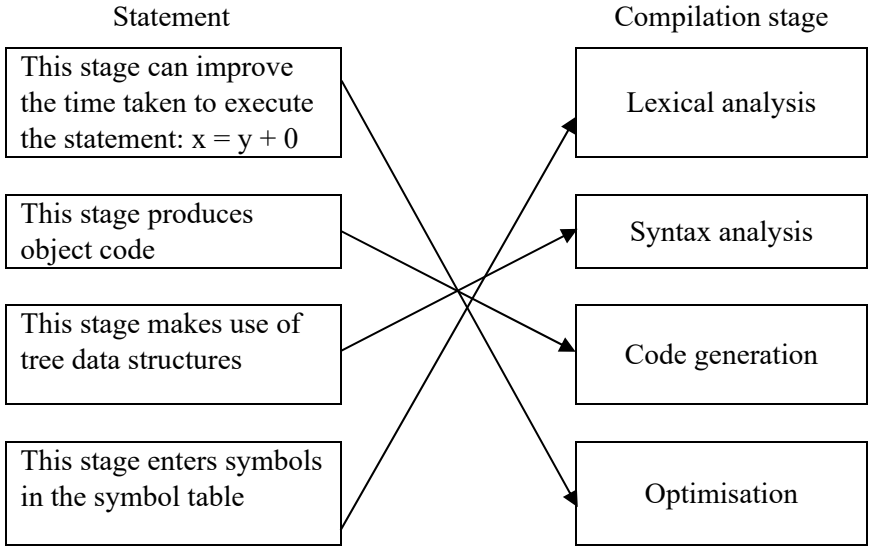


d)

```
<assignment_statement> ::= <variable> := <variable> <operator> <real>

<real> ::= <unsigned_integer> . <unsigned_integer>
```

9 a)



b) `P Q + R S / -`

c) i)

					2			
				3	3	5		
	2		1	1	1	1	6	
2	2	4	4	4	4	4	4	-2

ii) `b * a - (c + d + a)`

iii) In RPN evaluation of operators is left to right so there is no need for brackets to establish precedence.