Past-paper walk-through

Bit manipulation ▪ 4.3

eXercise

Questions in this set

- 9618/11 J25 Q8 ▪ 7 marks
- 9618/13 J25 Q7 ▪ 5 marks
- 9618/13 N24 Q7 ▪ 6 marks
- 9618/12 J24 Q5 ▪ 7 marks

Reading the mark scheme

- **B1** a mark that stands on its own – the point is either made or not.
- **M1** a *method* mark: the right approach, even if the number is wrong.
- **A1** an *answer* mark, and it usually needs its M mark first.
- **C1** a working mark on the way to the answer.
- // separates answers the examiner accepts equally.
- **ecf** = error carried forward: a later part is marked on your own earlier answer.
- **owtte** = "or words to that effect" – the wording need not match.

Question 8

9618/11 J25 ▪ Q8 ▪ 7 marks

The following table shows part of the instruction set for a processor. The processor has two registers: the Accumulator (ACC) and an Index Register (IX).

(a)

Instruction		Explanation
Opcode	**Operand**	
LDM	#n	Immediate addressing. Load the number n to ACC
LDD	<address>	Direct addressing. Load the contents of the location at the given address to ACC
LDI	<address>	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC
INC	<register>	Add 1 to the contents of the register (ACC or IX)
STO	<address>	Store the contents of ACC at the given address
ADD	#n/Bn/&n	Add the number n to the ACC
DEC	<register>	Subtract 1 from the contents of the register (ACC or IX)
JMP	<address>	Jump to the given address
CMP	<address>	Compare the contents of ACC with the contents of <address>
JPE	<address>	Following a compare instruction, jump to <address> if the compare was True
END		Return control to the operating system

Question 8

- ACC denotes Accumulator
- `<address>` can be an absolute or a symbolic address
- # denotes a denary number, e.g. #123
- B denotes a binary number, e.g. B01001010
- & denotes a hexadecimal number, e.g. &4A

[4]

Question 8

Instruction		Explanation
Opcode	**Operand**	
LDM	#n	Immediate addressing. Load the number n to ACC
LDD	<address>	Direct addressing. Load the contents of the location at the given address to ACC
LDI	<address>	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC
INC	<register>	Add 1 to the contents of the register (ACC or IX)
STO	<address>	Store the contents of ACC at the given address
ADD	#n/Bn/&n	Add the number n to the ACC
DEC	<register>	Subtract 1 from the contents of the register (ACC or IX)
JMP	<address>	Jump to the given address
CMP	<address>	Compare the contents of ACC with the contents of <address>
JPE	<address>	Following a compare instruction, jump to <address> if the compare was True
END		Return control to the operating system

Question 8

END		Return control to the operating system

ACC denotes Accumulator
<address> can be an absolute or a symbolic address
# denotes a denary number, e.g. #123
B denotes a binary number, e.g. B01001010
& denotes a hexadecimal number, e.g. &4A

Question 8

Instruction		Explanation
Opcode	**Operand**	
AND	#n/Bn/&n	Bitwise AND operation of the contents of ACC with the operand
AND	<address>	Bitwise AND operation of the contents of ACC with the contents of <address>
XOR	#n/Bn/&n	Bitwise XOR operation of the contents of ACC with the operand
XOR	<address>	Bitwise XOR operation of the contents of ACC with the contents of <address>
OR	#n/Bn/&n	Bitwise OR operation of the contents of ACC with the operand
OR	<address>	Bitwise OR operation of the contents of ACC with the contents of <address>
LSL	#n	Bits in ACC are shifted logically n places to the left. Zeros are introduced on the right-hand end
LSR	#n	Bits in ACC are shifted logically n places to the right. Zeros are introduced on the left-hand end

Question 8

		Zeros are introduced on the left-hand end

<address> can be an absolute or symbolic address
# denotes a denary number, e.g. #123
B denotes a binary number, e.g. B01001010
& denotes a hexadecimal number, e.g. &4A

Question 8

Instruction address	ACC	Memory address			
		80	81	82	83
		10	8	80	81

[4]

Question 8

9618/11 J25 ▪ Q8 ▪ 7 marks

The following table shows part of the instruction set for a processor. The processor has two registers: the Accumulator (ACC) and an Index Register (IX).

(b)(i) Write the bit manipulation instruction that can be used to set the least significant bit to 1 in an 8-bit register. All other bits must remain unchanged.

The instruction needs to work on a register that contains any 8-bit binary number. **[1]**

(b)(ii) The ACC currently contains the following binary value.

0	1	0	1	0	1	0	1

Question 8

Write the result after the instruction `XOR &FE` is run.

[1]

(b)(iii) The ACC currently contains the following binary value.

0	1	1	0	1	0	1	1

Write the result after the instruction `LSR #5` is run.

[1]

Solution 8

(a) Mark scheme

- 1 mark for each shaded section
- Instruction address
- ACC
- Memory address
- 80
- 81
- 82
- 83

Solution 8

- 10
- 8
- 80
- 81
- 200
- 8
- 201
- 9
- 202
- 9

Solution 8

- 203
- 10
- 204
- 206
- 9
- 207
- 19
- 208
- 210
- 19

Solution 8

(b)(i)

Mark scheme

- 1 mark for:
- OR B0000 0001 // OR #1 // OR &1

(b)(ii)

Mark scheme

- 1 mark for:
- 1010 1011

Solution 8

(b)(iii)

Mark scheme

- 1 mark for:
- 0000 0011

Question 7

9618/13 J25 ▪ Q7 ▪ 5 marks

(a)(i) Show the result of a logical left shift of 2 places on the two's complement binary integer 11001010 **[1]**

(a)(ii) Show the result of an arithmetic right shift of 3 places on the two's complement binary integer 10011110 **[1]**

(b) Complete the following binary addition. Show your working. Include any overflow bit(s).

$$\begin{array}{rcccccccc} & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ + & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ \hline \end{array}$$

[1]

(c) Subtract the binary number 00011110 from the binary number 01100100 using binary subtraction. Show your working. **[2]**

Solution 7

(a)(i)

Mark scheme

- 1 mark for
- 0010 1000

(a)(ii)

Mark scheme

- 1 mark for
- 1111 0011

Solution 7

(b) Worked steps

Add the columns from the right, carrying into the next column — the same procedure as denary, except that the carry happens at 2 rather than at 10.
Write the carries in a row of their own above the sum, and keep the columns aligned. In binary each column can only be 0, 1, 2 or 3 once the carry is included, so:

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 1 = 0$, carry 1
- $1 + 1 + 1 = 1$, carry 1

Solution 7

The eight-bit result here is `1010 0110`, with a **carry out of the top column**.
The question asks explicitly to *include any overflow bits*, so write that ninth bit down and label it. Two numbers of eight bits can sum to nine, and the ninth has nowhere to go in an eight-bit register — reporting it is the difference between a wrong answer and a right answer with a flag.

Solution 7

Show the carries. The mark is for the working as much as the total, and a bare answer cannot be given partial credit.

Mark scheme

- 1 mark for
- 1 1 1 1 0 1 0 1
- +1 11 01 11 1 0 0 01 1
- (1) 1 0 1 0 0 1 1 0
- Answer: (1) 1010 0110

Solution 7

(c) Worked steps

Two methods are accepted, and both earn the working mark.

Direct subtraction — borrow from the next column left, where borrowing turns a 0 into a 10 in binary (that is, 2):

```
  0 1 1 0 0 1 0 0
- 0 0 0 1 1 1 1 0
  ---------------
  0 1 0 0 0 1 1 0
```

Solution 7

Two's complement addition — negate the subtrahend and add, which is what a processor actually does:

- 30 is `0001 1110`; invert to `1110 0001` and add one $\rightarrow$ `1110 0010`.
- `0110 0100` + `1110 0010` = `(1) 0100 0110`, and the carry out is **discarded**.

Either way the answer is `0100 0110`.
Check in denary: $100 - 30 = 70$, and $64 + 4 + 2 = 70$. ✓

Solution 7

The carry out of the two's complement addition is **not** an overflow. Adding numbers of opposite signs can never overflow, so a carry out of the top bit there is normal and is thrown away — unlike the ninth bit in part (b), which was a genuine overflow. **Mark scheme**

- 1 mark for showing binary subtraction (any method)
- Direct subtraction:
- 0 1 1 10 10 11 10 0
- -0 0 01 11 11 11 1 0
- 0 1 0 0 0 1 1 0
- Adding the two's complement:
- 0 1 1 0 0 1 0 0
- +1 11 11 1 0 0 0 1 0
- (1) 0 1 0 0 0 1 1 0
- 1 mark for answer 0100 0110

Question 7

9618/13 N24 ▪ Q7 ▪ 6 marks

7 The following table shows part of the instruction set for a processor. The processor has two registers, the Accumulator (ACC) and the Index Register (IX).

Instruction		Explanation
Opcode	**Operand**	
LDM	#n	Immediate addressing. Load the number n to ACC
LDD	<address>	Direct addressing. Load the contents of the location at the given address to ACC
LDI	<address>	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC
LDX	<address>	Indexed addressing. Form the address from <address> + the contents of the index register. Copy the contents of this calculated address to ACC

Question 7

LDR	#n	Immediate addressing. Load the number n to IX
ADD	#n/Bn/&n	Add the number n to the ACC
ADD	<address>	Add the contents of the given address to the ACC
SUB	#n/Bn/&n	Subtract the number n from the ACC
SUB	<address>	Subtract the contents of the given address from the ACC
INC	<register>	Add 1 to the contents of the register (ACC or IX)
DEC	<register>	Subtract 1 from the contents of the register (ACC or IX)
<address> can be an absolute or a symbolic address # denotes a denary number, e.g. #123 B denotes a binary number, e.g. B01001010 & denotes a hexadecimal number, e.g. &4A		

(a) The current contents of memory are shown:

Address Data

Question 7

Address	Data
50	54
51	55
52	50
53	52
54	100
55	25
56	50

The current contents of the ACC and IX are shown:

ACC	50
IX	45

Complete the table by writing the content of the ACC after each program has run

Question 7

Complete the table by writing the content of the ACC after each program has run.

	Instructions	ACC content

1	`LDD 50`
	`ADD #4`
	`ADD 54`	

2	`LDI 53`
	`DEC ACC`
	`ADD 56`	

3	`LDM #55` `SUB #5`	

[3]

(b) The instruction set also includes these bit manipulation instructions:

Question 7

Instruction		

Question 7

Instruction		Explanation
Opcode	**Operand**	
AND	#n/Bn/&n	Bitwise AND operation of the contents of ACC with the operand
AND	<address>	Bitwise AND operation of the contents of ACC with the contents of <address>
XOR	#n/Bn/&n	Bitwise XOR operation of the contents of ACC with the operand
XOR	<address>	Bitwise XOR operation of the contents of ACC with the contents of <address>
OR	#n/Bn/&n	Bitwise OR operation of the contents of ACC with the operand
OR	<address>	Bitwise OR operation of the contents of ACC with the contents of <address>

<address> can be an absolute or a symbolic address
# denotes a denary number, e.g. #123
B denotes a binary number, e.g. B01001010
& denotes a hexadecimal number, e.g. &4A

Question 7

& denotes a hexadecimal number, e.g. &4A

Question 7

Explain how bit manipulation can be used to clear the data in an 8-bit register.

Write the bit manipulation instruction that will be used.

Question 7

[3]

Solution 7

(a) Mark scheme

- 1 mark each:
- Instructions
- ACC content
- 1
- LDD 50
- ADD #4
- ADD 54
- 158

Solution 7

- 2
- LDI 53
- DEC ACC
- ADD 56/99
- 3
- LDM #55
- SUB #5
- 50
- 3

Solution 7

(b) Worked steps

Three marks: two for the explanation, one for the instruction itself.

- A **bit-manipulation operation** is needed to set **all the bits to zero**.
- **Compare the result with 0** …
- … and the comparison is **true if the register has been cleared**.
- The instruction is `AND B00000000` (equivalently `AND #0` or `AND &00`).

The reason `AND` with zero works is the property of the operator: AND gives 1 only where **both** bits are 1, so a mask of all zeros forces every bit to 0 whatever the register held.

Contrast it with the other two, which is the understanding being tested:

- `OR` with a mask **sets** bits — `OR B11111111` would fill the register with ones.

Solution 7

- `XOR` with a mask **flips** bits — and `XOR` with the register's *own* value is the other standard way to clear it, since anything XORed with itself is zero.

So the general rule: **AND masks bits off, OR sets bits on, XOR flips them.** Every question in this topic is answered by choosing the operator whose behaviour matches the required change. **Mark scheme**

- 1 mark for each bullet point for the explanation
- 1 mark for correct instruction
- ▪
- A bit manipulation operation is required to set all the bits to zero
- ▪

Solution 7

- Compare the result of the masking with 0
- ▪
- ... the result of comparison will be true if the register is cleared
- ▪
- AND B00000000 / #00 / &00
- 3

Question 5

9618/12 J24 ▪ Q5 ▪ 7 marks

5 The following table shows part of the instruction set for a processor. The processor has two registers: the Accumulator (ACC) and an Index Register (IX).

Instruction		Explanation
Opcode	**Operand**	
LDM	#n	Immediate addressing. Load the number n to ACC
LDD	<address>	Direct addressing. Load the contents of the location at the given address to ACC
LDI	<address>	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC
LDX	<address>	Indexed addressing. Form the address from <address> + the contents of the index register. Copy the contents of this calculated address to ACC

Question 5

LDR	#n	Immediate addressing. Load the number n to IX
ADD	#n/Bn/&n	Add the number n to the ACC
ADD	<address>	Add the contents of the given address to the ACC
SUB	#n/Bn/&n	Subtract the number n from the ACC
SUB	<address>	Subtract the contents of the given address from the ACC
INC	<register>	Add 1 to the contents of the register (ACC or IX)

<address> can be an absolute or a symbolic address
# denotes a denary number, e.g. #123
B denotes a binary number, e.g. B01001010
& denotes a hexadecimal number, e.g. &4A

(a) The current contents of memory are shown:

Question 5

Address	Data
10	1
11	3
12	5
13	11
14	10
15	16
16	12

The current contents of the ACC and IX are shown:

Question 5

ACC	10
IX	0

Complete the table by writing the content of the ACC after each program has run.

Program number	Code	ACC content
1	`LDI 15` `SUB #1`	
2	`LDD 14` `ADD 11`	
3	`LDM #11` `ADD #3`	

Question 5

Question 5

	SUB 16	
4	LDR #2 LDX 14 ADD #2	

[4]

(b) The processor includes these bit manipulation instructions:

Instruction		Explanation
Opcode	**Operand**	
AND	#n/Bn/&n	Bitwise AND operation of the contents of ACC with the operand
AND	<address>	Bitwise AND operation of the contents of ACC with the contents of <address>
XOR	#n/Bn/&n	Bitwise XOR operation of the contents of ACC with the operand

Question 5

		Bitwise XOR operation of the contents of ACC with the contents of

Question 5

XOR	<address>	Bitwise XOR operation of the contents of ACC with the contents of <address>
OR	#n/Bn/&n	Bitwise OR operation of the contents of ACC with the operand
OR	<address>	Bitwise OR operation of the contents of ACC with the contents of <address>

<address> can be an absolute or a symbolic address
# denotes a denary number, e.g. #123
B denotes a binary number, e.g. B01001010
& denotes a hexadecimal number, e.g. &4A

The current contents of memory are shown:

Question 5

Address	Data
25	11000110
26	11100001
27	10000001
28	11001101
29	00001111

The current content of the ACC is shown:

0	1	0	0	0	1	1	0

Complete the table by writing the content of the ACC after each program has run.

Question 5

The binary number 01000110 is reloaded into the ACC before each program is run.

Program number	Code	ACC content
1	`XOR 29`	
2	`AND #29`	
3	`OR B11111111`	

[3]

Solution 5

(a) Mark scheme

- 1 mark for each correct answer:
- Program
- Number
- Code
- ACC Content
- 1
- LDI 15
- SUB #1

Solution 5

- 11
- 2
- LDD 14
- ADD 11/13
- 3
- LDM #11
- ADD #3
- SUB 16/2
- 4
- LDR #2

Solution 5

- LDX 14
- ADD #2
- 14
- 4

Solution 5

(b) Worked steps

Work each program from the **stated starting value of the ACC**, not from the previous program's result — the question reloads it each time, and that is easy to miss. Then apply the operator column by column. Write the two operands one above the other:

- `AND` gives 1 only where **both** bits are 1 — it **masks bits off**, since a 0 in the mask clears that position.
- `OR` gives 1 where **either** bit is 1 — it **sets bits on**.
- `XOR` gives 1 where the bits **differ** — it **flips** the bits the mask marks with 1.
- `LSL #n` shifts left `n` places, filling with zeros and discarding what falls off the top — each place **multiplies by 2**.

Solution 5

- `LSR #n` shifts right, filling with zeros — each place **divides by 2**, discarding the remainder.

One mark per correct final ACC content.

The detail that costs marks quietly: the operands are given in **different bases** in the same question — #29 is denary, `B00011101` is binary, `&1D` is hexadecimal. Convert every operand to eight binary digits before combining, and write the conversion down. A useful check after each line: an AND can only ever **reduce** the number of 1 bits, and an OR can only **increase** it. A result that gains bits after an AND has a slipped column somewhere. **Mark scheme**

- 1 mark for each correct answer:

Solution 5

- Program
- Number
- Code
- ACC Content
- 1
- XOR 29
- 0100 1001
- 2
- AND #29
- 0000 0100

Solution 5

- 3
- OR B11111111
- 1111 1111
- 3
- 9618/12
- May/June 2024