Processor Fundamentals – 2021 AS 9618

1. Nov/2021/Paper_11/No.6

(a) There are **two** errors in the following register transfer notation for the fetch-execute cycle.

```
1   MAR ←[PC]

2   PC ← [PC] – 1

3   MDR ← [MAR]

4   CIR ← [MDR]
```

Complete the following table by:

- identifying the line number of each error
- describing the error
- writing the correct statement.

Line number	Description of the error	Correct statement

[4]

PapaCambridge

(b) The following table shows part of the instruction set for a processor. The processor has one general purpose register, 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
STO	<address>	Store the contents of ACC at the given address
INC	<register>	Add 1 to the contents of the register (ACC or IX)
CMP	<address>	Compare the contents of ACC with the contents of <address>
JPN	<address>	Following a compare instruction, jump to <address> if the compare was False
JMP	<address>	Jump to the given address
IN		Key in a character and store its ASCII value in ACC
OUT		Output to the screen the character whose ASCII value is stored in ACC
END		Return control to the operating system
XOR	#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>
AND	#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>
OR	#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
<address> can be an absolute or symbolic address # denotes a denary number, e.g. #123 B denotes a binary number, e.g. B01001101		

PapaCambridge

The current contents of main memory are shown:

Address	**Data**
100	00001111
101	11110000
102	01010101
103	11111111
104	00000000

Each row of the following table shows the current contents of ACC in binary and the instruction that will be performed on those contents.

Complete the table by writing the new contents of the ACC after the execution of each instruction.

Current contents of the ACC	Instruction	New contents of the ACC
`11111111`	`OR 101`	
`00000000`	`XOR #15`	
`10101010`	`LSR #2`	
`01010101`	`AND 104`	

[4]

PapaCambridge

2. Nov/2021/Paper_12/No.8

The Von Neumann model for a computer system uses registers.

(a) Describe the role of the following special purpose registers in the fetch-execute (F-E) cycle.

(i) Memory Address Register (MAR) ....

....

....

....

Memory Data Register (MDR) ....

....

....

.... [4]

(ii) Another special purpose register is the Index Register.

Identify **one other** special purpose register used in the Von Neumann model for a computer system.

....

.... [1]

PapaCambridge

(b) The following table shows part of the instruction set for a processor. The processor has one general purpose register, 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
`STO`	`<address>`	Store the contents of ACC at the given address
`INC`	`<register>`	Add 1 to the contents of the register (ACC or IX)
`CMP`	`<address>`	Compare the contents of ACC with the contents of <address>
`JPN`	`<address>`	Following a compare instruction, jump to <address> if the compare was False
`JMP`	`<address>`	Jump to the given address
`IN`		Key in a character and store its ASCII value in ACC
`OUT`		Output to the screen the character whose ASCII value is stored in ACC
`END`		Return control to the operating system
`XOR`	`#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`	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>
`AND`	`#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>
`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
<address> can be an absolute or symbolic address # denotes a denary number, e.g. #123		

PapaCambridge

The current contents of main memory are shown:

Address	**Data**
100	01010101
101	11110000
102	00001111
103	00000000
104	11111111

(i) In the following table, each row shows the current contents of the ACC in binary and the instruction that will be performed on those contents.

Complete the table by writing the new contents of the ACC after the execution of each instruction.

Current contents of the ACC	**Instruction**	**New contents of the ACC**
`01010101`	`XOR 101`	
`11110000`	`AND 104`	
`00001111`	`LSL #4`	
`11111111`	`OR 102`	

[4]

(ii) The following table contains five assembly language instruction groups.

Write an appropriate assembly language instruction for each instruction group, using the given instruction set. The first one has been completed for you.

Instruction Group	**Instruction**
Data movement	`LDM #2`
Input and output of data	
Arithmetic operations	
Unconditional and conditional instructions	
Compare instructions	

[4]

PapaCambridge

(iii) The opcode `LDM` uses immediate addressing. The opcode `LDD` uses direct addressing.

Identify **and** describe **one additional** mode of addressing.

Mode of addressing ....

Description ....

....

.... [2]

PapaCambridge

3. Nov/2021/Paper_13/No.6

(a) There are **two** errors in the following register transfer notation for the fetch-execute cycle.

```
1   MAR ←[PC]

2   PC ← [PC] – 1

3   MDR ← [MAR]

4   CIR ← [MDR]
```

Complete the following table by:

- identifying the line number of each error
- describing the error
- writing the correct statement.

Line number	Description of the error	Correct statement

[4]

PapaCambridge

(b) The following table shows part of the instruction set for a processor. The processor has one general purpose register, 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
`STO`	`<address>`	Store the contents of ACC at the given address
`INC`	`<register>`	Add 1 to the contents of the register (ACC or IX)
`CMP`	`<address>`	Compare the contents of ACC with the contents of <address>
`JPN`	`<address>`	Following a compare instruction, jump to <address> if the compare was False
`JMP`	`<address>`	Jump to the given address
`IN`		Key in a character and store its ASCII value in ACC
`OUT`		Output to the screen the character whose ASCII value is stored in ACC
`END`		Return control to the operating system
`XOR`	`#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>
`AND`	`#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>
`OR`	`#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
<address> can be an absolute or symbolic address # denotes a denary number, e.g. #123 B denotes a binary number, e.g. B01001101		

PapaCambridge

The current contents of main memory are shown:

Address	**Data**
100	00001111
101	11110000
102	01010101
103	11111111
104	00000000

Each row of the following table shows the current contents of ACC in binary and the instruction that will be performed on those contents.

Complete the table by writing the new contents of the ACC after the execution of each instruction.

Current contents of the ACC	Instruction	New contents of the ACC
`11111111`	`OR 101`	
`00000000`	`XOR #15`	
`10101010`	`LSR #2`	
`01010101`	`AND 104`	

[4]

PapaCambridge

4. Nov/2021/Paper_11/No.3

A processor has one general purpose register, the Accumulator (ACC), and several special purpose registers.

(a) Complete the following description of the role of the registers in the fetch-execute cycle by writing the missing registers.

The .... holds the address of the next instruction

to be loaded. This address is sent to the ....

The .... holds the data fetched from this address.

This data is sent to the .... and the Control Unit

decodes the instruction's opcode.

The .... is incremented.

[5]

PapaCambridge

(b) The following table shows part of the instruction set for a processor. The processor has one general purpose register, 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
`LDR`	`#n`	Immediate addressing. Load the number n to IX
`MOV`	`<register>`	Move the contents of the accumulator to the given register (IX)
`STO`	`<address>`	Store contents of ACC at the given address
`ADD`	`<address>`	Add the contents of the given address to the ACC
`INC`	`<register>`	Add 1 to the contents of the register (ACC or IX)
`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
`JPN`	`<address>`	Following a compare instruction, jump to <address> if the compare was False
`JMP`	`<address>`	Jump to the given address
`OUT`		Output to the screen the character whose ASCII value is stored in ACC
`END`		Return control to the operating system
`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
<address> can be an absolute address or a symbolic address # denotes a denary number, e.g. #123		

PapaCambridge

The current contents of the main memory and selected values from the ASCII character set are shown.

Address	Instruction
200	LDD 365
201	CMP 366
202	JPE 209
203	INC ACC
204	STO 365
205	MOV IX
206	LDX 365
207	OUT
208	JMP 200
209	END
…	
365	1
366	3
367	65
368	66
IX	0

ASCII code table (selected codes only)

ASCII code	Character
65	A
66	B
67	C
68	D

PapaCambridge

Complete the trace table for the program currently in main memory.

Instruction address	ACC	Memory address				IX	Output
		365	366	367	368		
		1	3	65	66	0	

[6]

PapaCambridge

(c) **(i)** The Accumulator currently contains the binary number:

0	0	1	1	0	1	0	1

Write the contents of the Accumulator after the processor has executed the following instruction:

```
LSL #2
```


[1]

(ii) The Accumulator currently contains the binary number:

0	0	1	1	0	1	0	1

Identify the mathematical operation that the following instruction will perform on the contents of the accumulator.

```
LSR #3
```

....

.... [1]

PapaCambridge

5. June/2021/Paper_12/No.4

The table shows part of the instruction set for a processor. The processor has one general purpose register, 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
`STO`	`<address>`	Store contents of ACC at the given address
`ADD`	`<address>`	Add the contents of the given address to 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)
`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
`JPN`	`<address>`	Following a compare instruction, jump to `<address>` if the compare was False
`JMP`	`<address>`	Jump to the given address
`IN`		Key in a character and store its ASCII value in ACC
`OUT`		Output to the screen the character whose ASCII value is stored in ACC
`END`		Return control to the operating system

denotes a denary number, e.g. #123

The current contents of the main memory and selected values from the ASCII character set are:

Address	Instruction
70	`IN`
71	`CMP 100`
72	`JPE 80`
73	`CMP 101`
74	`JPE 76`
75	`JMP 80`
76	`LDD 102`
77	`INC ACC`
78	`STO 102`
79	`JMP 70`
80	`LDD 102`
81	`DEC ACC`
82	`STO 102`
83	`JMP 70`
…	
100	`68`
101	`65`
102	`100`

ASCII code table (selected codes only)

ASCII code	Character
65	A
66	B
67	C
68	D

PapaCambridge

(a) Complete the trace table for the program currently in main memory when the following characters are input:

```
A  D
```

Do not trace the program any further when the third input is required.

Instruction address	**ACC**	**Memory address**		
		100	**101**	**102**
		68	65	100

[4]

PapaCambridge

(b) Some bit manipulation instructions are shown in the table:

Instruction		**Explanation**
Opcode	**Operand**	
`AND`	`#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`	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`	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 address or a symbolic address # denotes a denary number, e.g. #123		

The contents of the memory address 300 are shown:

Bit Number	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
300	0	1	1	0	0	1	1	0

(i) The contents of memory address 300 represent an unsigned binary integer.

Write the denary value of the unsigned binary integer in memory address 300.

.... [1]

(ii) An assembly language program needs to test if bit number 2 in memory address 300 is a 1.

Complete the assembly language instruction to perform this test.

.... #4

[1]

(iii) An assembly language program needs to set bit numbers 4, 5, 6 and 7 to 0, but keep bits 0 to 3 with their existing values.

Write the assembly language instruction to perform this action.

....

.... [2]

PapaCambridge

6. June/2021/Paper_12/No.5a,b

Seth uses a computer for work.

(a) Complete the following descriptions of internal components of a computer by writing the missing terms.

The .... transmits the signals to coordinate events based

on the electronic pulses of the ....

The .... carries data to the components, while the

.... carries the address where data needs to be written to

or read from.

The .... performs mathematical operations and

logical comparisons.

[5]

(b) Describe the ways in which the following factors can affect the performance of his laptop computer.

Number of cores

....

....

....

....

Clock speed

....

....

....

....

[4]

PapaCambridge