Feedback on Quiz # 1 (Summative)

The **Quiz # 1** was composed of **10** questions, which were randomly selected from a Question Bank of 30 questions. Answers and feedback comments for all of the questions are given below:

Q1.1	The result of a bitwise XOR operation between 0x5D and __________ gives 0xEB. Find the unknown hexadecimal number?
(a)	**0xB6**
(b)	0xA7
(c)	0x5A
(d)	0xFF

Feedback:

In order to find out the unknown hexadecimal number, we can take XOR between 0x5D and 0xEB:

```
0x5D = 0101 1101
0xEB = 1110 1011
----------------
0xB6 = 1011 0110
```

The correct answer is **0xB6**

Q1.2	The result of a bitwise XOR operation between 0xCA and __________ gives 0x6D. Find the unknown hexadecimal number?
(a)	**0xA7**
(b)	0xB6
(c)	0x5A
(d)	0xEF

Feedback:

In order to find out the unknown hexadecimal number, we can take XOR between 0xCA and 0x6D:

```
0xCA = 1100 1010
0x6D = 0110 1101
----------------
0xA7 = 1010 0111
```

The correct answer is **0xA7**

Q1.3	The result of a bitwise XOR operation between 0xED and __________ gives 0xB7. Find the unknown hexadecimal number?
(a)	**0x5A**
(b)	0xA7
(c)	0xB6
(d)	0xFF

Feedback:

In order to find out the unknown hexadecimal number, we can take XOR between 0xED and 0xB7:

```
0xED = 1110 1101
0xB7 = 1011 0111
----------------
0x5A = 0101 1010
```

The correct answer is **0x5A**

Q2.1	Which one of the following MIPS instructions can be used to add the value $(12)_{10}$ to the value in register $1 and place the result in register $2?
(a)	add $2, $1, 12
(b)	addi $2, $1, 0xC
(c)	addu $2, $1, $2
(d)	addi $2, 0xC, $1

Feedback:

The MIPS instruction for adding an immediate value to a register is “addi”.

$(12)_{10} = (0xC)_{16}$, hence the correct answer is: **`addi $2, $1, 0xC`**

Q2.2	Consider that register $4 contains the value $(160)_{10}$ and we execute the following MIPS instruction: **`ori $6, $4, 100`** What will be the result stored in register $6 after the execution of the above instruction?
(a)	$(160)_{10}$
(b)	$(196)_{10}$
(c)	$(228)_{10}$
(d)	$(100)_{10}$

Feedback:

Explanation:

$(160)_{10} = (10100000)_2$

$(100)_{10} = (01100100)_2$

$(10100000)_2$ OR $(01100100)_2 = (11100100)_2 = (228)_{10}$

Q2.3	Consider that register \$4 contains the value $(180)_{10}$ and we execute the following MIPS instruction: **`andi $5, $4, 125`** What will be the value stored in register \$5 after the execution of the above instruction?
(a)	$(253)_{10}$
(b)	$(176)_{10}$
(c)	$(52)_{10}$
(d)	$(256)_{10}$
Feedback: Explanation: $(180)_{10} = (1011\ 0100)_2$ $(125)_{10} = (0111\ 1101)_2$ $(1011\ 0100)_2$ AND $(0111\ 1101)_2 = (0011\ 0100)_2 = (52)_{10}$ **Note:** The question (on canvas) had a typo which referred to register \$3 rather than register \$4. Appropriate marks have been awarded to those students who were affected.	

Q3.1	What are the advantage(s) of a processor having a set of registers in addition to just a large main memory?
(a)	Registers provide much faster access to data values compared to RAM.
(b)	Registers can be addressed with a small number of bits.
(c)	The registers are cheaper than RAM.
(d)	The CPU needs to use the RAM for holding intermediate results.
(e)	The registers can store instructions permanently as compared to RAM.
Feedback: Registers are used by the CPU to store small amounts of data as well as intermediate results that are needed during processing. Registers are much faster as compared to RAM. There are a small number of registers and so only a few bits are needed to specify a register.	

Q3.2	What are the main differences between the von Neumann and Harvard architecture?
(a)	Harvard architecture separates memory space for data and instructions.
(b)	In the Harvard architecture there are separate buses for data and instructions
(c)	Both models are similar, except for the way they access the ALU.
(d)	In the von Neuman architecture there are separate data and address buses.
(e)	To access data and instructions, the Harvard architecture employs a single bus.
Feedback:	

In a system with a Harvard architecture, the memory is organised in two separate blocks, one for data and one for instructions. In a von Neumann architecture, data and instructions cannot be accessed at the same time; hence the CPU cannot simultaneously read an instruction and write data from/to the memory.

Q3.3	Which of the following statements are true?
(a)	Access to cache memory is faster than register access
(b)	With 32 bits, the maximum memory address is 2,147,483,647
(c)	Jump instructions reduce the performance benefit gained from pipelining
(d)	An ‘immediate' instruction includes one of the operand values within the instruction
(e)	The actual address of a label in assembly code (referenced in a jump instruction) must be determined before a program is run.
Feedback: Registers are the fastest memory in a computer system. With 32 bits we can address memory in the range 0 → (2^32) – 1 = 0 → 4294967295 Jump instructions transfer control. The instructions in the pipeline might no longer be executed. An instruction such as ADDI $dest, $oper, 42 uses the value 42 as the second operand A label is a symbolic reference. A jump instruction needs to know the actual address to which control should be transferred.	

Q4.1	Consider the following expression: `f = g + h + B[4];` Assume that the variables f, g and h are assigned to registers $1, $2 and $3, respectively. Assume that the base address of array B is held in register $4, and each element in the array is 4 bytes long. To access the i-th element of an array, we need to use (base address of that array + offset). What would be the equivalent representation of the above expression in MIPS assembly language?

(a)	`lw $5, 16($4)`
	`add $1, $2, $3`
	`add $1, $1, $5`

(b)	`lw $5, 12($4)`
	`add $1, $2, $3`
	`add $1, $1, $5`

(c)	`lw $5, 8($4)`
	`add $1, $2, $3`
	`add $1, $1, $5`

(d)	`lw $5, 4($4)`
	`add $1, $2, $3`
	`add $1, $1, $5`

Feedback:

The register ($4) is called the “base register” and holds the “base address” of the array B.
“0($4)” contains one variable ($4) and one constant, which is the “offset”.
The constant is a multiple of 4 because there are four bytes in each element.
So, to access the element at index 4, (4*4=16) à get the offset of the value in terms of byte from the beginning of the array. So: B[4] = 16($4).
The correct sequence of MIPS instructions is:

```
lw $5, 16($4)      # $5 = B[4]
add $1, $2, $3     # f = g + h
add $1, $1, $5     # f = f + B[4]
```

Q4.2	Consider the following expression: `h = j + k – A[3];` Assume that the variables h, j and k are assigned to registers $2, $3 and $4, respectively. Assume that the base address of array A is held in register $5 and each element in the array is 4 bytes long. To access the i-th element of an array, we need to use (base address of that array + offset). What would be the equivalent representation of the above expression in MIPS assembly language?

(a)	`lw $1, 12($5)`
	 `add $2, $3, $4`
	 `sub $2, $2, $1`

(b)	`lw $1, 8($5)`
	 `add $2, $3, $4`
	 `sub $2, $2, $1`

(c)	`lw $1, 4($5)`
	 `add $2, $3, $4`
	 `sub $2, $2, $1`

(d)	`lw $1, 0($5)`
	 `add $2, $3, $4`
	 `sub $2, $2, $1`

Feedback:

The register ($5) is called the “base register” and holds the “base address” of the array A.
“0($5)” contains one variable ($5) and one constant, which is the “offset”.
The constant is a multiple of 4 because there are four bytes in an element.
So, to access the element at index 3, (3*4=12) à get the offset of the value in terms of byte from the beginning of the array. So: A[3] = 12($5).
The correct sequence of MIPS instructions is:

```
lw $1, 12($5)      # $1 = A[3]
add $2, $3, $4     # h = j + k
sub $2, $2, $1     # h = h – A[3]
```

Q4.3	Consider the following expression: `a = b * d / C[2];` Assume that the variables a, b and d are assigned to registers `$6`, `$7` and `$8`, respectively. Assume that the base address of array C is held in register `$9` and each element in the array is 4 bytes long. To access the i-th element of an array, we need to use (base address of that array + offset). What would be the equivalent representation of the above expression in MIPS assembly language?

(a)	`lw $10, 8($9)`
	`mult $6, $7, $8`
	`div $6, $6, $10`

(b)	`lw $10, 0($9)`
	`mult $6, $7, $8`
	`div $6, $6, $10`

(c)	`lw $10, 4($9)`
	`mult $6, $7, $8`
	`div $6, $6, $10`

(d)	`lw $10, 12($9)`
	`mult $6, $7, $8`
	`div $6, $6, $10`

Feedback:

The register ($9) is called the "base register" and holds the "base address" of the array C.
"0($9)" contains one variable ($9) and one constant, which is the "offset".
The constant is a multiple of 4 because there are four bytes in an element.
So, to access the element at index 2, (2*4=8) à get the offset of the value in terms of byte from the beginning of the array. So: C[2] = 8($9).
The correct sequence of MIPS instructions is:

```
lw $10, 8($9)      # $10 = C[2]
mult $6, $7, $8    # a = b * d
div $6, $6, $10    # a = a / C[2]
```

Q5.1	Which of the following statements are true?
(a)	A compiled program will normally execute faster than an interpreted program
(b)	A pure interpreter (a software implementation of a virtual machine) still needs to understand the architecture of the computer on which the program will run.
(c)	A just-in-time compiler is independent of the architecture of the computer on which it is running
(d)	A pure interpreter is portable across different architectures
(e)	Cross compilation occurs when a compiler running on one machine generates code for a different computer architecture or operating system.

Feedback:
A compiled program will execute native machine code whereas an interpreter will need to decode and execute instructions in software at run time. So, it will be slower.
A pure interpreter is a software implementation of a machine. Therefore, it can be independent of the architecture and instruction set of the actual machine on which it runs and portable across machines.
A just-in-time compiler generates machine code at run time. Therefore, it needs to understand the instruction set and architecture of the machine.
Cross compilation is when a compiler runs on one machine and generates code to be run on another.

Q5.2	Which of the following statements are false?
(a)	A compiled program will normally execute faster than an interpreted program
(b)	A pure interpreter (a software implementation of a virtual machine) still needs to understand the architecture of the computer on which the program will run.
(c)	A just-in-time compiler is independent of the architecture of the computer on which it is running
(d)	A pure interpreter is portable across different architectures
(e)	Cross compilation occurs when a compiler running on one machine generates code for a different computer architecture or operating system.

Feedback:
A compiled program will execute native machine code whereas an interpreter will need to decode and execute instructions in software at run time. So, it will be slower.
A pure interpreter is a software implementation of a machine. Therefore, it can be independent of the architecture and instruction set of the actual machine on which it runs and portable across machines.
A just-in-time compiler generates machine code at run time. Therefore, it needs to understand the instruction set and architecture of the machine.
Cross compilation is when a compiler runs on one machine and generates code to be run on another.

Q5.3	Which of the following statements are true?
(a)	Every instruction is of the same length (number of bits) in Java bytecode
(b)	Machine code instructions can only address registers
(c)	Compilation translates source code in one language into an equivalent program in another language
(d)	A pure interpreter is usually portable across different architectures and operating systems
(e)	A compiler always generates machine code for a particular computer

Feedback:
Usually (and in Java byte code) instructions are of variable length.
Some machine instructions need to access RAM (e.g. load and store instructions).
Compilation takes code in one language and generates equivalent code in another – usually the source code is a high-level language, and the object code is in a lower-level language (this is often

machine code, but it could be another high-level language or an intermediate language as well.)
Pure interpreters are software implementations of a machine. Therefore, they can be independent of the actual hardware and operating system on which they run.

Q6.1 Consider the following algorithm:

```
while (there are one or more characters left to read)
{
    read a character
    push the character onto the stack
}

while (the stack is not empty)
{
    x = pop a character off the stack
    write the character x to the screen
}
```

For the input “computer science”, which of the following statement(s) is/are true?

(a)	We need at least two stacks to implement the given algorithm
(b)	The output of this algorithm is ecneics retupmoc
(c)	The output of this algorithm is retupmoc ecneics
(d)	We need only one stack to implement this algorithm
(e)	The output of this algorithm is ecneicsretupmoc

Feedback:
First, the input “computer science” is pushed one by one into the stack. When it is popped, the output will be as "ecneics retupmoc".
Only 1 stack is required for implementation of this algorithm.

Q6.2 Consider the following algorithm:

```
while (there are one or more characters left to read)
{
    read a character
    push the character onto the stack
}

while (the stack is not empty)
{
    x = pop a character off the stack
    write the character x to the screen
}
```

	For the input “software engineering”, which of the following statement(s) is/are true?
(a)	We need at least two stacks to implement the given algorithm
(b)	The output of this algorithm is gnireenigne erawtfos
(c)	The output of this algorithm is erawtfos gnireenigne
(d)	We need only one stack to implement this algorithm
(e)	The output of this algorithm is gnireenigneerawtfos

Feedback:

First, the input “software engineering” is pushed one by one into the stack. When it is popped, the output will be as "gnireenigne erawtfos".
Only 1 stack is required for implementation of this algorithm.

Q6.3 Consider the following algorithm:

```
while (there are one or more characters left to read)
{
    read a character
    push the character onto the stack
}

while (the stack is not empty)
{
    x = pop a character off the stack
    write the character x to the screen
}
```

For the input “operating system”, which of the following statement(s) is/are false?

(a)	We need at least two stacks to implement the given algorithm
(b)	The output of this algorithm is metsys gnitarepo
(c)	The output of this algorithm is gnitarepo metsys
(d)	We need only one stack to implement this algorithm
(e)	The output of this algorithm is metsysgnitarepo

Feedback:

First, the input “operating system” is pushed one by one into the stack. When it is popped, the output will be as "metsys gnitarepo".
Only 1 stack is required for implementation of this algorithm.

Q7.1	Evaluate the following reverse polish notation expression: **2 5 8 2 0 + + * *** What is the result?
(a)	100
(b)	7
(c)	82
(d)	0
(e)	This is an invalid expression

Feedback:

A simplified algorithm to evaluate RPN (assuming only binary operators and without error checking) is:

```
WHILE there are tokens to be read
   IF the token is an operator
      Pop the first operand from the stack
      Pop the second operator from the stack
      Perform the operation
      Push the result onto the stack
   ELSE
      Push the value onto the stack
```

Let's evaluate the following expression with the help of a stack.

2 5 8 2 0 + + * *

We can evaluate the RPN using the following steps:

Step#	INPUT	STACK
1	2	2
2	5	2 5
3	8	2 5 8
4	2	2 5 8 2
5	0	2 5 8 2 0
6	+	2 5 8 2
7	+	2 5 10
8	*	2 50
9	*	100

So, the correct answer is **100**, as shown in step#9 above.

Q7.2	Evaluate the following reverse polish notation expression: **7 5 * 2 – 6 2 * *** What is the result?
(a)	**396**
(b)	128
(c)	77
(d)	252
(e)	This is an invalid expression

Feedback:

A simplified algorithm to evaluate RPN (assuming only binary operators and without error checking) is:

```
WHILE there are tokens to be read
  IF the token is an operator
    Pop the first operand from the stack
    Pop the second operator from the stack
    Perform the operation
    Push the result onto the stack
  ELSE
    Push the value onto the stack
```

Let's evaluate the following expression with the help of a stack.

7 5 * 2 – 6 2 * *

We can evaluate the RPN using the following steps:

Step#	INPUT	STACK
1	7	7
2	5	7 5
3	*	35
4	2	35 2
5	-	33
6	6	33 6
7	2	33 6 2
8	*	33 12
9	*	396

So, the correct answer is **396**, as shown in step#9 above.

Q7.3	Evaluate the following reverse polish notation expression: **4 3 7 * + 20 -** What is the result?
(a)	**5**
(b)	-3
(c)	3
(d)	160
(e)	This is an invalid expression

Feedback:

A simplified algorithm to evaluate RPN (assuming only binary operators and without error checking) is:

```
WHILE there are tokens to be read
  IF the token is an operator
    Pop the first operand from the stack
    Pop the second operator from the stack
    Perform the operation
    Push the result onto the stack
  ELSE
    Push the value onto the stack
```

Let's evaluate the following expression with the help of a stack.

4 3 7 * + 20 -

We can evaluate the RPN using the following steps:

Step#	INPUT	STACK
1	4	4
2	3	4 3
3	7	4 3 7
4	*	4 21
5	+	25
6	20	25 20
7	-	5

So, the correct answer is **5**, as shown in step#7 above.

Q8.1 On a MIPS machine, assume that x, y, z and q are mapped to registers $1, $2, $3 and $4, respectively. Given the following MIPS instructions:

```
sub $5, $3, $4
add $1, $1, $2
add $1, $1, $5
```

Which of the expressions below is equivalent to the above sequence of instructions?

(a)	**x = x + y + (z – q)**
(b)	x = x + y + (q – z)
(c)	y = (x + y) + z – q
(d)	z = x + y + (q – z)

Feedback:

The following instructions show their mapping to the corresponding parts of the expression:

```
sub $5, $3, $4   // z - q
add $1, $1, $2   // x = x + y
add $1, $1, $5   // x = x + y + (z - q)
```

Q8.2 On a MIPS machine, assume that x, y, z and q are mapped to registers $1, $2, $3 and $4, respectively. Given the following MIPS instructions:

```
sub $5, $4, $3
add $1, $1, $2
add $1, $1, $5
```

Which of the expressions below is equivalent to the above sequence of instructions?

(a)	x = x + y + (z – q)
(b)	**x = x + y + (q – z)**
(c)	y = (x + y) + z – q
(d)	z = x + y + (q – z)

Feedback:

The following instructions show their mapping to the corresponding parts of the expression:

```
sub $5, $4, $3 // q-z
add $1, $1, $2 // x = x + y
add $1, $1, $5 // x = x + y + (q - z)
```

Q8.3	On a MIPS machine, assume that x, y, z and q are mapped to registers \$1, \$2, \$3 and \$4, respectively. Given the following MIPS instructions:
	
	`sub $5, $4, $3`
	`add $1, $1, $2`
	`add $2, $1, $5`
	
	Which of the expressions below is equivalent to the above sequence of instructions?

(a)	x = x + y + (z – q)
(b)	x = x + y + (q – z)
(c)	**y = x + y + (q – z)**
(d)	z = (x + y) + z – q

Feedback:

The following instructions show their mapping to the corresponding parts of the expression:

```
sub $5, $4, $3 // q - z
add $1, $1, $2 // x = x + y
add $2, $1, $5 // y = x + y + (q - z)
```

Q9.1	Given that a floating-point number is represented in 2 bytes as follows: Sign of mantissa (1 bit) | Exponent represented in 2's complement signed binary (3 bits) | Magnitude of mantissa (12 bits) The conversion of $(3A.BC)_{16}$ to this representation gives us: (You can assume that we don't need to add offset to the exponent part)
(a)	0011 1101 0101 1111
(b)	0101 1101 0101 1110
(c)	0101 1101 0101 1111
(d)	0011 1101 0101 1110
(e)	The floating-point number cannot be represented in the given format

Feedback:

1. $(3A.BC)_{16}$ = **00**11 1010 . 1011 11**00** = 11 1010 . 1011 11
2. 11 1010 . 1011 11 = 1.11010101111 * 2^5.
3. Sign of mantissa (+) = 0
4. Exponent 5 = 101 and it cannot be represented in 3 bits using 2's complement notation, as we need 1 more bit to represent this value.
5. Magnitude of mantissa = 1101 0101 111 (11 bits so add 0 to the right to make the 12 bits) = 1101 0101 111**0**

The correct answer should be: The floating-point number cannot be represented in the given format.

Note: The correct option was missing in the canvas quiz, therefore, appropriate marks have been awarded to those students affected.

Q9.2	Given that a floating-point number is represented in 2 bytes as follows:

Sign of mantissa (1 bit)	Exponent represented in 2's complement signed binary (3 bits)	Magnitude of mantissa (12 bits)

The conversion of $(-3A.BC)_{16}$ to this representation gives us:
(You can assume that we don't need to add offset to the exponent part)

(a)	1011 1101 0101 1111
(b)	1101 1101 0101 1110
(c)	1101 1101 0101 1111
(d)	1011 1101 0101 1110
(e)	The floating-point number cannot be represented in the given format

Feedback:

1. $(3A.BC)_{16}$ = **00**11 1010 . 1011 11**00** = 11 1010.1011 11
2. 11 1010.1011 11 = 1.11010101111 * 2^5.
3. Sign of mantissa (-) = 1
4. Exponent 5 = 101 and it cannot be represented in 3 bits using 2's complement notation, as we need 1 more bit to represent this value.
5. Magnitude of mantissa = 1101 0101 111 (11 bits so add 0 to the right to make the 12 bits) = 1101 0101 111**0**

The correct answer should be: The floating-point number cannot be represented in the given format.

Note: The correct option was missing in the canvas quiz, therefore, appropriate marks have been awarded to those students affected.

Q9.3	Given that a floating-point number is represented in 2 bytes as follows: | Sign of mantissa (1 bit) | Exponent represented in 2's complement signed binary (3 bits) | Magnitude of mantissa (12 bits) | The conversion of $(0.3AB)_{16}$ to this representation gives us: (You can assume that we don't need to add offset to the exponent part)
(a)	0011 1101 0101 1001
(b)	0011 1101 0101 1000
(c)	**0101 1101 0101 1000**
(d)	0101 1101 0101 1001

Feedback:

1. $(0.3AB)_{16}$ = 0000 . 0011 1010 1011 = 0.0011 1010 1011
2. 0.0011 1010 1011 = 1.110101011 * 2^(-3)
3. Sign of mantissa (+) = 0
4. Exponent -3 = -011 and it is negative, so we need to apply 2's complement => -3 = -011 = 101
5. Magnitude of mantissa = 1101 0101 1 (9 bits so add 0s to the right to make the 12 bits) = 1101 0101 1**000**

The correct answer is: **0101 1101 0101 1000**

Q10.1 Consider the following Toy CPU program:

```
Address      Mnemonics       Machine Code
=======      =========       ============
10           ld b 3A         0E, 3A
12           ld c 06         16, 06
14           call 50         DE, 50
...          ...             ...
50           ld a 10         06, 10
52           add b 02        2E, 02
54           add a [b]       24
55           sub c 02        56, 02
57           jnz c 52        B6, 52
59           add b 02        2E, 02
5B           st a [b]        64
5C           ret             F8
```

and the following initial state of the memory:

```
--------------------------------------------------
| 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
--------------------------------------------------
  3A   3B   3C   3D   3E   3F   40   41   42   43
```

Once the above program has been executed, what will be the state memory?

Note: All the values / addresses given above are in hexadecimal.

(a)

```
--------------------------------------------------
| 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 2B | 0E |
--------------------------------------------------
  3A   3B   3C   3D   3E   3F   40   41   42   43
```

(b)

```
--------------------------------------------------
| 05 | 06 | 07 | 08 | 09 | 0A | 2B | 0C | 0D | 0E |
--------------------------------------------------
  3A   3B   3C   3D   3E   3F   40   41   42   43
```

(c)

```
--------------------------------------------------
| 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 37 | 0E |
--------------------------------------------------
  3A   3B   3C   3D   3E   3F   40   41   42   43
```

(d)

```
--------------------------------------------------
| 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 3C | 0E |
--------------------------------------------------
  3A   3B   3C   3D   3E   3F   40   41   42   43
```

Feedback:

The main routine starts at address 0x10, loads the register b with 0x3A (the start of memory address), loads the register c with 06 and calls the subroutine at 0x50.

The subroutine loads the register a with 0x10 (initial value), adds 0x02 to the memory address in register b i.e. 0x3A + 0x02, and then adds the memory contents from the new address to register a. The register c is decreased by 0x02, and the next instruction checks if the register c has become zero. Effectively, this loop executes three times, and adds the values in memory locations 0x3C, 0x3E and 0x40 to the initial value of register a.

Once the loop terminates, the register b is incremented by 0x02 again, to compute the address for storing the result in memory i.e. 0x42 in this case, where the result of hexadecimal addition of 0x10 + 0x07 + 0x09 + 0x0B = 0x2B will be stored.

Hence the correct answer is:

```
--------------------------------------------------
| 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 2B | 0E |
--------------------------------------------------
  3A   3B   3C   3D   3E   3F   40   41   42   43
```

Q.10.2

Consider the following Toy CPU program:

```
Address      Mnemonics        Machine Code
=======      =========        ============
10           ld b 3B          0E, 3B
12           ld c 06          16, 06
14           call 50          DE, 50
...          ...              ...
50           ld a 10          06, 10
52           add b 02         2E, 02
54           add a [b]        24
55           sub c 02         56, 02
57           jnz c 52         B6, 52
59           add b 02         2E, 02
5B           st a [b]         64
5C           ret              F8
```

and the following initial state of the memory:

```
---------------------------------------------------
| 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
---------------------------------------------------
  3A   3B   3C   3D   3E   3F   40   41   42   43
```

Once the above program has been executed, what will be the state memory?
Note: All the values / addresses given above are in hexadecimal.

(a)

```
---------------------------------------------------
| 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 2E |
---------------------------------------------------
  3A   3B   3C   3D   3E   3F   40   41   42   43
```

(b)

```
---------------------------------------------------
| 05 | 06 | 07 | 08 | 09 | 0A | 0B | 2E | 0D | 0E |
---------------------------------------------------
  3A   3B   3C   3D   3E   3F   40   41   42   43
```

(c)

```
---------------------------------------------------
| 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 40 |
---------------------------------------------------
  3A   3B   3C   3D   3E   3F   40   41   42   43
```

(d)

```
---------------------------------------------------
| 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 34 |
---------------------------------------------------
  3A   3B   3C   3D   3E   3F   40   41   42   43
```

Feedback:

The main routine starts at address 0x10, loads the register b with 0x3B (the start of memory address), loads the register c with 06 and calls the subroutine at 0x50.

The subroutine loads the register a with 0x10 (initial value), adds 0x02 to the memory address in register b i.e. 0x3B + 0x02, and then adds the memory contents from the new address to register a. The register c is decreased by 0x02, and the next instruction checks if the register c has become zero. Effectively, this loop executes three times, and adds the values in memory locations 0x3D, 0x3F and 0x41 to the initial value of register a.

Once the loop terminates, the register b is incremented by 0x02 again, to compute the address for storing the result in memory i.e. 0x43 in this case, where the result of hexadecimal addition of 0x10 + 0x08 + 0x0A + 0x0C = 0x2E will be stored.

Hence the correct answer is:

```
---------------------------------------------------
| 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 2E |
---------------------------------------------------
  3A   3B   3C   3D   3E   3F   40   41   42   43
```

Q10.3 Consider the following Toy CPU program:

```
Address     Mnemonics       Machine Code
=======     =========       ============
10          ld b 48         0E, 48
12          ld c 06         16, 06
14          call 50         DE, 50
...         ...             ...
50          ld a 10         06, 10
52          add b 02        2E, 02
54          add a [b]       24
55          sub c 02        56, 02
57          jnz c 52        B6, 52
59          add b 02        2E, 02
5B          st a [b]        64
5C          ret             F8
```

and the following initial state of the memory:

```
---------------------------------------------------
| 0A | 0B | 0C | 0D | 04 | 05 | 06 | 07 | 0E | 0F |
---------------------------------------------------
  48   49   4A   4B   4C   4D   4E   4F   50   51
```

Once the above program has been executed, what will be the state memory?
Note: All the values / addresses given above are in hexadecimal.

(a)

```
---------------------------------------------------
| 0A | 0B | 0C | 0D | 04 | 05 | 06 | 07 | 26 | 0F |
---------------------------------------------------
  48   49   4A   4B   4C   4D   4E   4F   50   51
```

(b)

```
---------------------------------------------------
| 0A | 0B | 0C | 0D | 04 | 05 | 26 | 07 | 0E | 0F |
---------------------------------------------------
  48   49   4A   4B   4C   4D   4E   4F   50   51
```

(c)

```
---------------------------------------------------
| 0A | 0B | 0C | 0D | 04 | 05 | 06 | 07 | 32 | 0F |
---------------------------------------------------
  48   49   4A   4B   4C   4D   4E   4F   50   51
```

(d)

```
---------------------------------------------------
| 0A | 0B | 0C | 0D | 04 | 05 | 06 | 07 | 30 | 0F |
---------------------------------------------------
  48   49   4A   4B   4C   4D   4E   4F   50   51
```

Feedback:
The main routine starts at address 0x10, loads the register b with 0x48 (the start of memory address), loads the register c with 06 and calls the subroutine at 0x50.

The subroutine loads the register a with 0x10 (initial value), adds 0x02 to the memory address in register b i.e. 0x48 + 0x02, and then adds the memory contents from the new address to register a. The register c is decreased by 0x02, and the next instruction checks if the register c has become zero. Effectively, this loop executes three times, and adds the values in memory locations 0x4A, 0x4C and 0x4E to the initial value of register a.

Once the loop terminates, the register b is incremented by 0x02 again, to compute the address for storing the result in memory i.e. 0x50 in this case, where the result of hexadecimal addition of 0x10 + 0x0C + 0x04 + 0x06 = 0x26 will be stored.

Hence the correct answer is:

```
---------------------------------------------------
| 0A | 0B | 0C | 0D | 04 | 05 | 06 | 07 | 26 | 0F |
---------------------------------------------------
  48   49   4A   4B   4C   4D   4E   4F   50   51
```