Student Last Name : Zainudin
Student Given Name : Anis Faqiehah binti
Student ID : 200382845

Q1-a) Represented in hex

loop	$t0	$t1	$t3
	?	?	?
1		0x0000001b	
	0x00000000		
			0x00000000
2		0x00000016	
	0x00000001		
			0x00000000
3		0x00000011	
	0x00000002		
			0x00000000
4		0x0000000c	
	0x00000003		
			0x00000000
5		0x00000007	
	0x00000004		
			0x00000000
6		0x00000002	
	0x00000005		
			0x00000001

Explanation : The register $t1 has the value of 0x0000001b from the instruction add $t1, $zero, $v0 where $t1 gets the value of $v0. $t2 gets the value of 0x00000005. $t0 gets the value of 0x00000000 from the instruction add $t0, $zero, $zero when $t0=0. Entering the loop, the instruction slt puts the value 0x00000000 if false and 0x00000001 if true in $t3 when $t1 is less

than $t2. In this case $t3 is 0x00000000 because 0x0000001b is more than the value of $t2 which is 0x00000005. Branch if not equal when $t3 is not equal to zero, branching to DONE. Since $t3 is zero, so it will not branch to DONE, instead it will execute the next line which will subtract $t1-$t2 storing it in $t1. According to the table, 0x0000001b - 0x00000005 and we will get the new value of $t1, 0x00000016. Next, increment $t0 by 1 which makes $t0 0x00000001. The loop will continue until it reaches $t1 > $t2 which will make $t3 equal to 0x00000001. Then it will branch to label DONE. DONE will print the message Result and give the last value obtained in $t0 from the trace table. Then exit the program.

Q1-b) The result obtained will be 0. This is because $t1 will get the value 0xfffffffe5. It will not go through the loop at all as $t1 is more than the value of $t2 which will equate $t3 to 0x00000000. When checked with a branch if not equal instruction it will branch to DONE. Therefore, no operation in the loop will be executed. So $t3 will not be incremented at all.

Q1-c)

```
.data
msg1 : .asciiz "Enter the first integer: "
msg2 : .asciiz "Enter the second integer: "
msg3 : .asciiz "Quotient : "
msg4 : .asciiz "\nRemainder: "

.text
li $v0 , 4
la $a0 , msg1
syscall

li $v0 , 5
syscall
add $t1 , $zero , $v0

li $v0 , 4
la $a0 , msg2
syscall

li $v0 , 5
syscall
add $t2 , $zero , $v0

add $t0 , $zero , $zero

div $t1, $t2

mflo $s0 # Quotient - loads quotient into low
mfhi $s1 # Reminder - loads quotient into high
```

```
li $v0 , 4 #syscall to print string
la $a0 , msg3
syscall

li $v0 , 1 #to print integer
add $a0 , $s0 , $zero
syscall

li $v0 , 4 # syscall to print string
la $a0 , msg4
syscall

li $v0 , 1 #syscall to print integer
add $a0 , $s1 , $zero
syscall

li $v0 , 10 #syscall to exit
syscall
```

Q2-a)

```
.text
   la  $t1, A              # t1: to hold the "address" of the next
                           # array element, initialised to the
                           # address of the first byte of the array
   lw $t2, 0 ($t1)         # t2 <- the current array element
   la  $t0, A_LENGTH
   lw  $t0, 0($t0)         # t0 <- A_LENGTH

   addi $t3, $zero, 0      # the index i that is initialised to 0

   NEXT_ARRAY_ELEMENT:
   add $t3, $t3, 1              # increment index by 1
   beq $t3, $t0, DONE           # if increment = 8 (length)
                                # means all element is examined, so DONE

   add $t4, $t3, $t3                # stores 2i in $t4
   add $t4, $t4, $t4                # stores 4i in $t4
   add $t4, $t4, $t1                # computes address  A[i] in $t4
   lw $s0,0($t4)                    # load value of A[i] into $s0

   slt $t6, $t2, $s0                  # is maximum < A[i]?
   beq $t6, $zero, NEXT_ARRAY_ELEMENT   # if equal to 9, then branch from the start of the loop
   addi $t2, $s0, 0                 #if $t6 is 1, then the current element is the new max


   j NEXT_ARRAY_ELEMENT       # jump to NEXT_ARRAY_ELEMENT (for loop)

   DONE:
   la $a0,msg
   li $v0,4
   syscall                  # print message "Maximum:"

   add $v0, $zero, 1  #set v0 to "1" to select
   add $a0,$zero, $t2 # "print integer" syscall
                            # a0 <-- t2 (max) to be printed
   syscall                  # invoking syscall to print the integer

   li $v0,10
   syscall                  # end



.data
   A:                   # our integer array
      .word   -1
      .word   4
      .word   -16
```

```
    .word   0
    .word   -2
    .word   5
    .word   13
    .word   2
A_LENGTH:   .word   8         # the length of the array
msg: .asciiz "\nMaximum :"
```

Q2-b)

```
.text
   la  $t1, A               # t1: to hold the "address" of the next
                            # array element, initialised to the
                             # address of the first byte of the array
   lw $t2, 0 ($t1)                  # t2 <- the current array element
   la  $t0, A_LENGTH
   lw  $t0, 0($t0)           # t0 <- A_LENGTH

   addi $t3, $zero, 0         # the index i that is initialised to 0


   NEXT_ARRAY_ELEMENT:
   add $t3, $t3, 1                  # increment index by 1
   beq $t3, $t0, DONE                # if increment = 8 (length)
                             # means all element is examined, so DONE

   add $t4, $t3, $t3                # stores 2i in $t4
   add $t4, $t4, $t4                # stores 4i in $t4
   add $t4, $t4, $t1                # computes address  A[i] in $t4
   lw $t2, 0 ($t1)                  # t2 <- the current array element

   slt $t5, $zero, $t2       # is current element less than zero?
   beq $t5, $zero, ABS                    # if yes, then proceed to ABS

   lw $s0,0($t4)                           # load value of A[i] into $s0

   slt $t6, $t2, $s0                     # is maximum < A[i]?
   beq $t6, $zero, NEXT_ARRAY_ELEMENT   # if equal to 9, then branch from the start of the
loop
   addi $t2, $s0, 0                      #if $t6 is 1, then the current element is the new max

   j NEXT_ARRAY_ELEMENT       # jump to NEXT_ARRAY_ELEMENT (for loop)

   ABS :
   sub $t2, $zero, $t2           # $t2 = 0 - (-ve number) so we will get positive
   slt $s3, $zero, $t2           # is it still less than zero (negative)
   bne $s3, $zero, NEXT_ARRAY_ELEMENT # if no, then it will return into the loop as a
positive number

   DONE:
   la $a0,msg
   li $v0,4
   syscall                  # print message "Divisible by 8:"
```

```
    add $v0, $zero, 1  #set v0 to "1" to select
    add $a0,$zero, $t2 # "print integer" syscall
                       # a0 <-- t2 (max) to be printed
    syscall            # invoking syscall to print the integer

    li $v0,10
    syscall            # end



.data
    A:                 # our integer array
        .word   -1
        .word   4
        .word   -16
        .word   0
        .word   -2
        .word   5
        .word   13
        .word   2
    A_LENGTH:   .word   8       # the length of the array
    msg: .asciiz "\nMaximum :"
```

Q2-c-i) As the instruction suggest, we can get the result to determine whether the number is divisible by 8 or not because the last part includes an immediate to be compared with and the instruction andi ending with 0, in binary indicates that it will produce an output ending with also zero while instruction andi ending with 1 will produce the original bit where the masking technique comes in and masks out the rest of the string while ensuring $t0 holds the first bitstring and the rest holds the other bitstrings.

Q2-C-ii)

```
.text
   la  $t1, A              # t1: to hold the "address" of the next
                           # array element, initialised to the
                            # address of the first byte of the array
   la  $t0, A_LENGTH
   lw  $t0, 0($t0)          # t0 <- A_LENGTH

   addi $t3, $zero, 0       # the index i that is initialised to 0

   NEXT_ARRAY_ELEMENT:
   add $t3, $t3, 1              # increment index by 1
   beq $t3, $t0, DONE               # if increment = 8 (length)
                            # means all element is examined, so DONE

   lw $t2, 0 ($t1)              # t2 <- the current array element

   # check if s0 & 1
   andi $t0, $t2, 0x0007
   bne $t0, $zero, NEXT_ARRAY_ELEMENT   # if not, branch to NEXT_ARRAY_ELEMENT

   addi $t2, $s0, 0


   j NEXT_ARRAY_ELEMENT       # jump to NEXT_ARRAY_ELEMENT (for loop)



   DONE:
   la $a0,msg2
   li $v0,4
   syscall              # print message "Divisible by 8:"

   add $v0, $zero, 1  #set v0 to "1" to select
   add $a0,$zero, $t2 # "print integer" syscall
                        # a0 <-- t2 (integer) to be printed
```

```
    syscall                 # invoking syscall to print the integer

    li $v0,10
    syscall                 # end


.data
    A:                      # our integer array
        .word   -1
        .word   4
        .word   -16
        .word   0
        .word   -2
        .word   5
        .word   13
        .word   2
    A_LENGTH:   .word   8       # the length of the array
    msg2: .asciiz "\nInteger divisible by 8? :"
```

Q3-a) h holds a value that will multiply 2^i. In this case h is the exponent, i. For example, $rt has an integer value and we want to multiply the value to 4. Just change h to 2 since 2^2 = 4. Therefore, we will get the answer 4 x $rt. Next, the answer will be stored in $rd.

Q3-b) 0000 0000 0000 0000 0000 0000 0000 0000
The opcode has 6 bits.
It has 3 registers($rd, $rs, $rt respectively) and they all have 5 bits each.
The shamt or the shift amount has 5 bits as well.
The function has 6 bits.
The bits have shifted to the left making 2 extra spaces and 00 are added.

Q3-c-iii) sll $t1, $t1, 3

Q3-c-iv) sll $t0, $t2, 4
sll $t3, $t4, 3
add $t1, $t0, $t2

Q3-c-v) sll $t1, $t1, 5
subi $t1, $t1,4

Q3-c-iii) sll $t1, $t1, 6
subi $t1, $t1, 4