Nguyễn Đình Dương – 20225966 – Midterm

A- 6: Input a positive number N from the keyboard, print the representation of N in the binary format.

```
# Program to read a number and print its binary representation
.data
    prompt:     .string "Enter a positive number: "
    error_msg:  .string "Please enter a positive number!\n"
    result_msg: .string "Binary representation: "
    newline:    .string "\n"

.text
.globl main

main:
    # Print prompt
    la a0, prompt
    li a7, 4
    ecall

    # Read integer
    li a7, 5
    ecall
    mv s0, a0    # Save input number in s0

    # Check if number is positive
    blez s0, error

    # Print result message
    la a0, result_msg
    li a7, 4
    ecall

    # Initialize registers
    mv t0, s0    # Copy number to t0
    li t1, 32    # Counter for 32 bits
    li t3, 1     # For bit masking
    li t4, 31    # For shifting
    slli t3, t3, 31  # Create mask with leftmost bit set

print_binary:
    # Check if we've printed all bits
```

```
    beqz t1, done

    # Test current bit
    and t2, t0, t3
    beqz t2, print_zero
    li a0, 49    # ASCII '1'
    j do_print
print_zero:
    li a0, 48    # ASCII '0'
do_print:
    li a7, 11
    ecall

    # Shift bit mask right by 1
    srli t3, t3, 1

    # Decrement counter
    addi t1, t1, -1

    j print_binary

error:
    # Print error message
    la a0, error_msg
    li a7, 4
    ecall
    j exit

done:
    # Print newline
    la a0, newline
    li a7, 4
    ecall

exit:
    # Exit program
    li a7, 10
    ecall
```

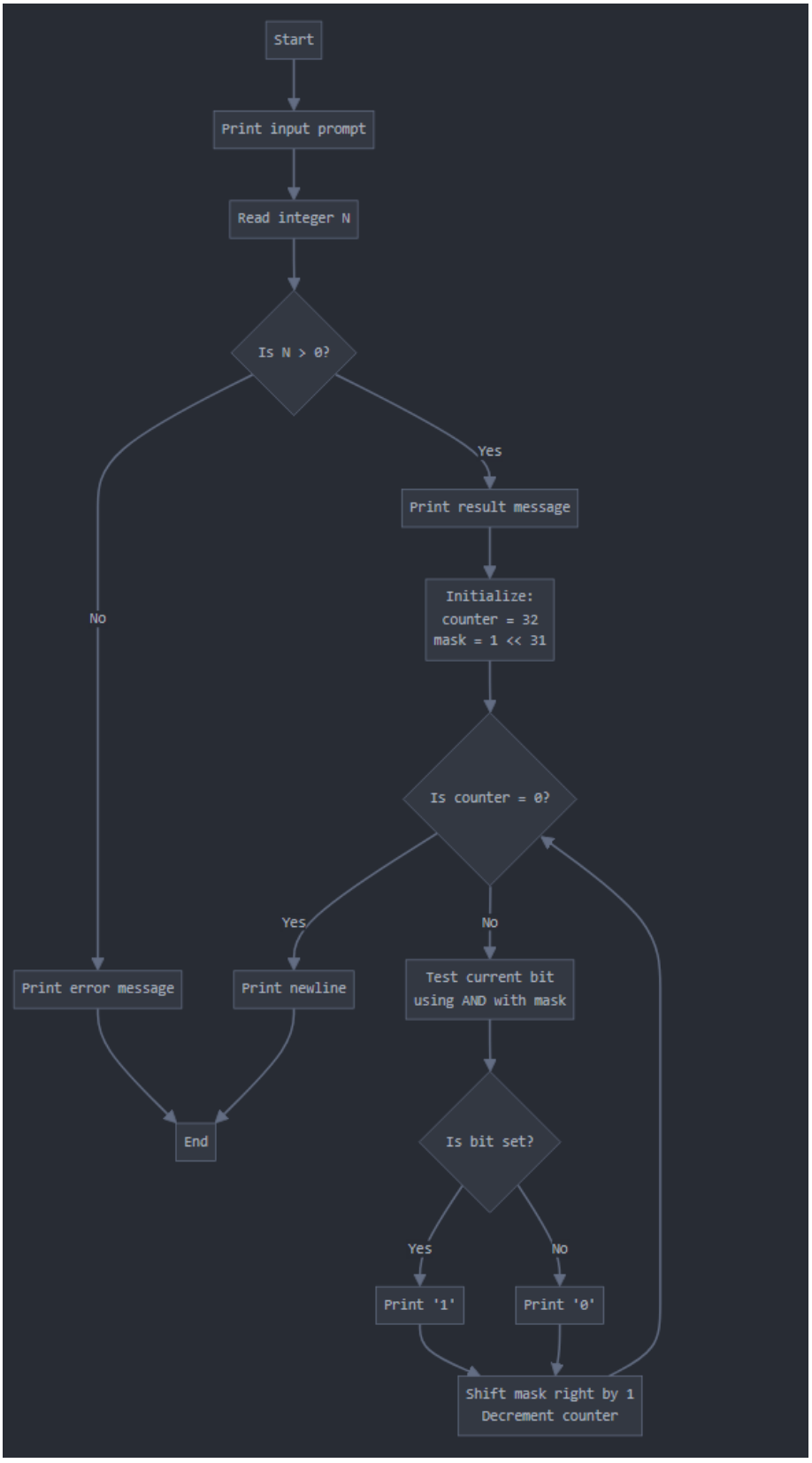
Detailed Program Flow:

1. Input Phase:

- Program starts by displaying prompt for input
- Uses syscall 4 to print string
- Uses syscall 5 to read integer
- Stores input in register s0

2. Validation Phase:
    - Checks if input number (s0) is positive
    - Uses blez (branch if less than or equal to zero)
    - If not positive, jumps to error handling
3. Initialization Phase:
    - Sets up registers:
        - t0: copy of input number
        - t1: bit counter (32)
        - t3: bit mask (starts with leftmost bit)
        - t4: for shift operations
4. Binary Conversion Phase:
    - Uses a loop to process all 32 bits
    - For each iteration:
        - Tests current bit using AND operation
        - Prints '1' or '0' based on result
        - Shifts mask right by 1
        - Decrements counter
5. Output Phase:
    - Uses syscall 11 to print each digit
    - Prints newline at end
    - Uses syscall 10 to exit program

Flow:


Start
Print input prompt
Read integer N
Is N > 0?
Yes
No
Print result message
Initialize:
counter = 32
mask = 1 << 31
Is counter = 0?
Yes
No
Print error message
Print newline
Test current bit
using AND with mask
End
Is bit set?
Yes
No
Print '1'
Print '0'
Shift mask right by 1
Decrement counter

Output:

Case	Output
15	Enter a positive number: 15 Binary representation: 00000000000000000000000000001111 -- program is finished running (0) --
16	Enter a positive number: 16 Binary representation: 00000000000000000000000000010000
-1	Enter a positive number: -1 Please enter a positive number! -- program is finished running (0) --
0	Enter a positive number: 0 Please enter a positive number!

B-5: Input an integer array from the keyboard, print the sum of positive elements and the sum of negative elements.

```
.data
    prompt_size: .string "Enter the size of array: "
    prompt_elem: .string "Enter element "
    colon:      .string ": "
    pos_msg:    .string "\nSum of positive elements: "
    neg_msg:    .string "\nSum of negative elements: "
    newline:    .string "\n"
    error_msg:   .string "The size of array must be positive"
    array:      .word 100    # Space for up to 100 integers

.text
.globl main

main:
    # Print prompt for array size
    la a0, prompt_size
    li a7, 4
```

```
    ecall

    # Read array size
    li a7, 5
    ecall
    ble a0,zero, end
    mv s0, a0        # s0 = array size

    # Initialize array index and address
    la s1, array     # s1 = array base address
    li t0, 0         # t0 = current index
    li s2, 0         # s2 = sum of positive numbers
    li s3, 0         # s3 = sum of negative numbers

input_loop:
    # Check if we've read all elements
    beq t0, s0, calc_done

    # Print "Enter element i: "
    la a0, prompt_elem
    li a7, 4
    ecall

    mv a0, t0
    li a7, 1
    ecall

    la a0, colon
    li a7, 4
    ecall

    # Read element
    li a7, 5
    ecall

    # Store element in array
    slli t1, t0, 2       # t1 = t0 * 4 (offset)
    add t1, t1, s1       # t1 = base + offset
    sw a0, 0(t1)         # store element

    # Increment index
    addi t0, t0, 1
    j input_loop
```

```
calc_done:
    # Reset index for summing
    li t0, 0         # t0 = current index

sum_loop:
    # Check if we've processed all elements
    beq t0, s0, print_results

    # Load current element
    slli t1, t0, 2       # t1 = t0 * 4
    add t1, t1, s1       # t1 = base + offset
    lw t2, 0(t1)         # t2 = current element

    # Check if positive or negative
    bgez t2, is_positive # if t2 >= 0, branch to is_positive

    # Negative number
    add s3, s3, t2
    j next_elem

is_positive:
    beqz t2, next_elem   # if t2 = 0, skip
    add s2, s2, t2

next_elem:
    addi t0, t0, 1
    j sum_loop

print_results:
    # Print sum of positive elements
    la a0, pos_msg
    li a7, 4
    ecall

    mv a0, s2
    li a7, 1
    ecall

    # Print sum of negative elements
    la a0, neg_msg
    li a7, 4
    ecall
```

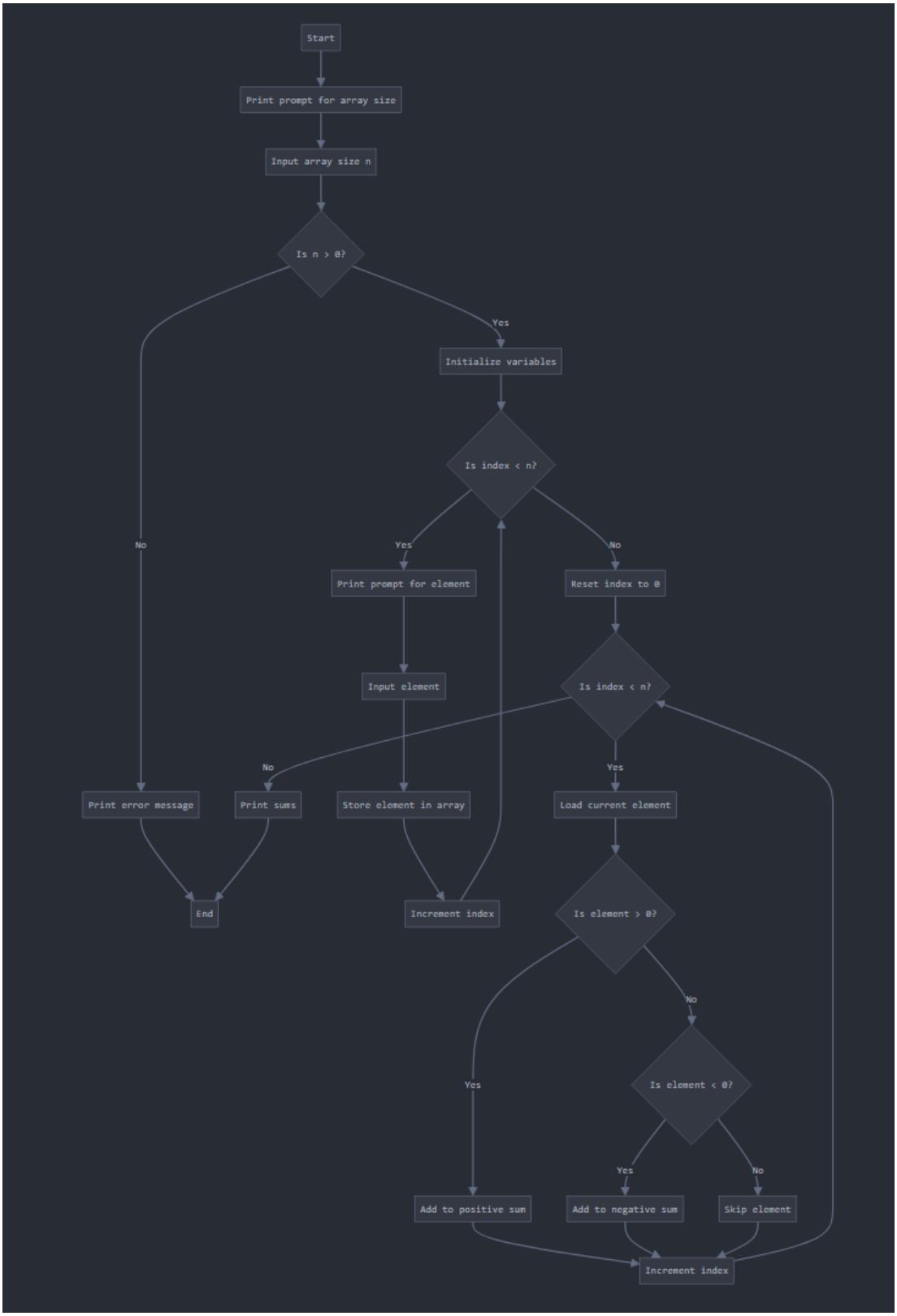
```
    mv a0, s3
    li a7, 1
    ecall

    # Print newline
    la a0, newline
    li a7, 4
    ecall

    # Exit program
    li a7, 10
    ecall
end:
    la a0, error_msg
    li a7, 4
    ecall
```

Algorithm: Sum of Positive and Negative Numbers

1. Start
2. Print prompt for array size
3. Input array size (n)
4. If n ≤ 0, print error message and end program
5. Initialize:
    - array base address
    - current index = 0
    - positive sum = 0
    - negative sum = 0
6. Input Loop:
    - While current index < n:
        - Print "Enter element [index]: "
        - Input element
        - Store element in array
        - Increment index
7. Reset index to 0

8. Sum Loop:
    - While current index < n:
        - Load current element
        - If element > 0:
            - Add to positive sum
        - If element < 0:
            - Add to negative sum
        - Increment index
9. Print "Sum of positive elements: " and positive sum
10. Print "Sum of negative elements: " and negative sum
11. End

Flow chart:


Start
Print prompt for array size
Input array size n
Is n > 0?
Yes
No
Initialize variables
Is index < n?
Yes
No
Print prompt for element
Reset index to 0
Input element
Is index < n?
No
Yes
Print error message
Print sums
Store element in array
Load current element
End
Increment index
Is element > 0?
No
Yes
Is element < 0?
Yes
No
Add to positive sum
Add to negative sum
Skip element
Increment index

Case	Output
Enter the size of array: 4 Enter element 0: 0 Enter element 1: 0 Enter element 2: 0 Enter element 3: 0	Sum of positive elements: 0 Sum of negative elements: 0
Enter the size of array: -1	The size of array must be positive
Enter the size of array: 0	The size of array must be positive
Enter the size of array: 5 Enter element 0: 1 Enter element 1: 1 Enter element 2: -3 Enter element 3: 4 Enter element 4: 5	Sum of positive elements: 11 Sum of negative elements: -3
Enter the size of array: 3 Enter element 0: -1 Enter element 1: -5 Enter element 2: 8	Sum of positive elements: 8 Sum of negative elements: -6

C-10: Input a string from the keyboard, print the uppercase character that has the largest ASCII code in the string

```
# Program to find the largest uppercase character in a string with exception handling
.data
    buffer:     .space 100     # Buffer to store input string
    prompt:     .string "Enter a string: "
    result:     .string "Largest uppercase character: "
    newline:    .string "\n"
    no_upper:   .string "Error: No uppercase characters found in the string!\n"
    empty_str:  .string "Error: Empty string or whitespace only!\n"
    same_chars: .string "Note: Multiple characters found with ASCII value "
    error_io:   .string "Error: Input/Output error occurred!\n"
    count_msg:  .string "Number of occurrences: "

# Text Section
.text
.globl main
```

```
main:
    # Print prompt
    li a7, 4
    la a0, prompt
    ecall

    # Read string
    li a7, 8
    la a0, buffer
    li a1, 100
    ecall

    # Check for I/O errors (simplified - checking if first byte is 0)
    lb t0, (a0)
    beqz t0, io_error

    # Initialize variables
    la t0, buffer        # t0 points to current character
    li t1, 0             # t1 holds the largest uppercase char
    li t2, 1             # t2 is whitespace flag (1 = only whitespace seen so far)
    li t3, 0             # t3 counts occurrences of largest char

process_loop:
    lb t4, (t0)          # load current character
    beqz t4, end_loop    # if null terminator, end loop

    # Check if character is not whitespace
    li t5, 32            # Space character
    bne t4, t5, not_whitespace
    j check_next

not_whitespace:
    li t2, 0             # Clear whitespace flag

check_next:
    # Check if character is uppercase (ASCII 65-90)
    li t5, 65            # 'A'
    blt t4, t5, next_char
    li t5, 90            # 'Z'
    bgt t4, t5, next_char

    # Compare with current largest
    blt t4, t1, next_char    # if current < largest, skip
```

```
    bne t4, t1, update_largest
    # If equal, increment counter
    addi t3, t3, 1
    j next_char

update_largest:
    # If new largest found, reset counter and update
    li t3, 1
    mv t1, t4

next_char:
    addi t0, t0, 1       # move to next character
    j process_loop

end_loop:
    # Check if string was empty or only whitespace
    bnez t2, empty_string

    # Check if we found any uppercase characters
    beqz t1, no_uppercase

    # Print result message
    li a7, 4
    la a0, result
    ecall

    # Print the character
    li a7, 11
    mv a0, t1
    ecall

    # Print newline
    li a7, 4
    la a0, newline
    ecall

    # If multiple occurrences, print count
    li t4, 1
    ble t3, t4, exit

    # Print multiple occurrence message
    la a0, same_chars
    li a7, 4
```

```
    ecall

    # Print ASCII value
    mv a0, t1
    li a7, 1
    ecall

    # Print newline
    la a0, newline
    li a7, 4
    ecall

    # Print count message
    la a0, count_msg
    li a7, 4
    ecall

    # Print count
    mv a0, t3
    li a7, 1
    ecall

    # Print newline
    la a0, newline
    li a7, 4
    ecall

    j exit

no_uppercase:
    # Print no uppercase found message
    li a7, 4
    la a0, no_upper
    ecall
    j exit

empty_string:
    # Print empty string message
    li a7, 4
    la a0, empty_str
    ecall
    j exit
```

```
io_error:
    # Print I/O error message
    li a7, 4
    la a0, error_io
    ecall
    j exit

exit:
    # Exit program
    li a7, 10
    ecall
```

Flow chart:

Exception Handling Details

1. **Empty String Detection**

IF buffer is empty OR contains only whitespace THEN

Handle empty string error

2. **I/O Error Handling**

   IF input operation fails THEN

   Display I/O error message

   Terminate program

3. **No Uppercase Characters**

   IF no uppercase characters found THEN

   Display appropriate message

   Terminate program

4. **Multiple Occurrences Handling**

   IF occurrenceCount > 1 THEN

   Display count and ASCII value

Complexity Analysis

- Time Complexity: O(n) where n is the length of input string
- Space Complexity: O(1) as we use fixed buffer size

Input/Output Specifications

Input Requirements

1. String length must not exceed 99 characters (100 including null terminator)
2. Input can contain any ASCII characters
3. Input can be empty or whitespace-only (will be handled as error cases)

Output Format

1. **Success Case:**

Largest uppercase character: X

[Optional] Number of occurrences: N

2. **Error Cases:**

Error: Empty string or whitespace only!

Error: No uppercase characters found!

Error: Input/Output error occurred!

Validation Rules

1. Character is uppercase if ASCII value is between 65 ('A') and 90 ('Z')
2. Whitespace characters include: space (32), tab (9), newline (10)
3. Valid string must contain at least one non-whitespace character
4. Valid result requires at least one uppercase character

Output:

Case	Output
Enter a string: ZZzzzzZZZZYYYYYYYYYYYY	Largest uppercase character: Z Note: Multiple characters found with ASCII value 90 Number of occurrences: 6
Enter a string: sdmYYYYYYYyyzz	Largest uppercase character: Y Note: Multiple characters found with ASCII value 89 Number of occurrences: 7
Enter a string: -238323	Error: No uppercase characters found in the string!
Enter a string: duongdeptrai	Error: No uppercase characters found in the string!
Enter a string: thayvuideptrai	Error: No uppercase characters found in the string!
Enter a string:	Error: No uppercase characters found in the string!