TO: Professor Robert Antonetti

FROM: Arjun Aravind

DATE: February 12th, 2025

SUBJECT: Program 3 Observations

The purpose of Program 3 is to implement an Input/Output program that takes in a number of up to 3 digits and converts that number to binary. In addition to this, the program must also validate that the user did enter a numerical value (not a character) and prompt them to re-enter if they didn’t.

To begin, the assembly file “extend\_uio.s” is created and populated with code utilizing the nano text editor (Code attached below in Reference). Each function in the program has a unique purpose:

* main – sets up the registers, displays prompts to user, and starts input\_loop
* input\_loop – reads the characters entered by user one at a time and validates that it is a number. If it is a number, proceed to process\_number. If it isn’t a number, proceed to invalid\_input to prompt the user to re-enter the values.
* invalid\_input – displays error message, clears registers and input buffer, and goes back up to input\_loop.
* process\_number – stores input number for processing and starts binary\_loop
* binary\_loop – converts decimal number to binary, converts to ASCII characters, and starts reverse\_loop (since values are currently in reverse order because they were read in reverse order)
* reverse\_loop – reverses the binary string of characters to the correct order
* print\_binary – displays the correctly formatted binary string
* exit\_program – exits program when successfully completed

The next step from here is to compile the Assembly code using in gcc and with the debugger using the command “gcc -g -o extend\_uio extend\_uio.s”, which creates the executable file “extend\_uio” (Figure 1). The “-g” segment of the command is there to ensure the debugger is permitted during compilation.



Figure 1

The next step is to run the Assembly code and test sample inputs. When the value 100 is input, the program properly converts it to 1100100 (Figure 2). When the non-numerical character “a” is input, the program appropriately displays the error message and prompts the user to re-enter a numerical value (Figure 3). It also proceeds to process the following numerical value correctly.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 2

A screen shot of a computer

AI-generated content may be incorrect.\

Figure 3

The next step is to open the debugger window to test the program utilizing a breakpoint. The debugger window can be opened through the command “gdb extend\_uio” and then run the command “r” to ensure that the code runs properly (Figure 4). From this window, we can also create and test a breakpoint to ensure that the code is executing properly in between steps. To do this, the command “b 15” is entered to create a breakpoint at line 9 and then the command “r” is used to run until the breakpoint (Figure 5). This shows that the code runs smoothly until it reaches the breakpoint.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4

A screen shot of a computer

AI-generated content may be incorrect.

Figure 5

The last step was to analyze the object dump files as a reference to visualize the Assembly codes alongside memory addresses and Opcodes (Figures 6-16). To do this, the command “objdump extend\_uio -d” was utilized.

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 6

A computer screen shot of white text

AI-generated content may be incorrect.

Figure 7

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 8

A computer screen shot of a computer program

AI-generated content may be incorrect.

Figure 9

A computer screen shot of white text

AI-generated content may be incorrect.

Figure 10

A computer screen shot of a program code

AI-generated content may be incorrect.

Figure 11

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 12

A computer screen shot of a computer code

AI-generated content may be incorrect.

Figure 13

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 14

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 15

A black screen with white text

AI-generated content may be incorrect.

Figure 16

In conclusion, this program works well to achieve the intended goal of being able to take user input, validate that it is a number, convert it to a binary value, and display the results back to the user.

Reference:

@\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@ name: extend\_uio.s

@

@ description: program to read 3 digits from stdin,

@ validates it, and render to stdout.

@

@\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.global main @entry point

.text @code area

main:

ldr r1, =prompt @prompt character

mov r2, #30 @Length of prompt message

mov r0, #1 @Stdio

mov r7, #4 @write system call

svc 0 @execute system call

@Prepare registers for number storage

mov r4, #0 @clear the register

mov r5, #0 @digit counter

input\_loop:

ldr r1, =inbuff @set pointer to input buffer

mov r2, #1 @read one char

mov r0, #0 @read from stdio

mov r7, #3 @read system call

svc 0 @execute system call

@Validate if input is a number (0-9)

ldrb r3, [r1] @load input character

cmp r3, #10 @check for newline

beq process\_number @if newline, process input

cmp r3, #'0' @check if char is 0

blt invalid\_input @if char is less than 0, it is invalid

cmp r3, #'9' @check if char is greater than 9

bgt invalid\_input @if char is greater than 9, it is invalid

@Convert valid character to integer to process it

sub r3, r3, #'0' @convert char to integer

mov r6, #10 @create multiplier (x10)

mul r4, r4, r6 @multiply current digit by multiplier (Shift left in decimal)

add r4, r4, r3 @append digit to final number

add r5, r5, #1 @increment digit counter

@limit input to 3 digits

cmp r5, #3

bge process\_number @if 3 digits entered, process input

@Continue receiving input

b input\_loop

invalid\_input:

@Output error for invalid input

ldr r1, =error\_msg @error message for invalid input

mov r2, #32 @length of error message

mov r0, #1 @stdio

mov r7, #4 @write system call

svc 0 @execute system call

@Reset Registers

mov r4, #0 @reset stored number

mov r5, #0 @reset digit counter

@Flush input buffer

ldr r1, =inbuff @point to buffer

mov r2, #4 @read remaining characters

mov r0, #0 @stdin

mov r7, #3 @read system call

svc 0 @discard input

@Prompt for re-entry

b input\_loop

process\_number:

ldr r1, =bin\_output @binary output message

mov r2, #24 @length of output message

mov r0, #1 @stdout

mov r7, #4 @write system call

svc 0 @execute system call

@Convert the decimal to binary

mov r6, r4 @copy decimal value

ldr r1, =binbuff @buffer for binary string

mov r2, #0 @string index

binary\_loop:

and r3, r6, #1 @least significant bit

add r3, r3, #'0' @convert to char

strb r3, [r1, r2] @store bit in buffer

lsr r6, r6, #1 @shift right

add r2, r2, #1 @increment the index

cmp r6, #0 @check if there is more to convert

bne binary\_loop @if yes, continue loop

@terminate safely

mov r3, #0

strb r3, [r1,r2]

@Reverse binary string for correct display

mov r6, #0

sub r2, r2, #1 @adjust index

reverse\_loop:

cmp r6, r2

bge print\_binary @if done, print output

ldrb r3, [r1, r6] @load first char

ldrb r7, [r1, r2] @load last char

strb r7, [r1, r6] @swap char

strb r3, [r1, r2]

add r6, r6, #1

sub r2, r2, #1

b reverse\_loop

print\_binary:

ldr r1,=binbuff @load binary buffer

mov r2, #10 @set max length

mov r0, #1 @stdout

mov r7, #4 @write system call

svc 0 @execute system call

exit\_program:

mov r7, #1 @exit system call, return 0

mov r0, #0 @successful execution

svc 0

.data @data follows

inbuff: .space 4 @buffer for input

binbuff: .space 12 @buffer for output

prompt: .ascii "\nEnter a base 10 digit (0-9): " @prompt message

error\_msg: .ascii "\nInvaild Input. Try Again (0-9):" @error message

bin\_output: .ascii "\nBinary represenation: " @binary output message

@ fix executable stack warning

.section .note.GNU-stack ,"", %progbits