

CPE 325: Intro to Embedded Computer Systems

Lab11

Software Reverse Engineering

Submitted by: Michael Agnew

Date of Experiment: 4/4/2024

Report Deadline: 4/9/2024

Demonstration Deadline: 4/15/2024

Introduction:

The main aim of this lab was delving into the world of software reverse engineering. There were various tasks that were laid out to be completed for this lab in terms of finding particular data throughout provided materials. The first objective consisted of searching through a provided .out file and connecting it to MobaXTerm with UART connection where the user was to crack the password that was prompted. Then, “magic numbers” as well as other header information had to be found by using ELF commands in the command prompt, and finally, the assignment also included converting a hex file to assembly and analyzing what it does.

Theory:

Elf File Components: One of the most important parts of this lab was understanding ELF files, what they are, what parts they have and how to access their information. ELF stands for “Executable and linkable file” and is used for object files, core dumps, shared libraries and of course executable files. This format is the one that is produced and used by CCS, which is why it is important to learn and understand. The ELF file is composed of the following parts: file header, program header table, section header table, segments and sections. The ELF file header simply provides information about what the file is and what it contains. The Program header table holds information regarding memory segments that are required for execution of the program. This is the header that informs the loader in what way to process an image that is located in memory. The Section header table contains data that is accessed by the program header table and section header table. The segments simply hold information that is important for runtime execution and the sections hold necessary information required for relocation and linking processes.

Naken Utility: Naken_util is a disassembler tool that was developed by Joe Davisson and Michael Kohn. This was mostly designed for the purpose of disassembling machine code that can be found in different assembly architectures. In the instance of this lab, the architecture being disassembled is the MSP430 architecture. This tool takes in a hex file as input and operates by providing the user with the choice of removing memory locations that are erased. This is then followed by simply converting the hex machine code to its assembly language counterpart. This can all be done in the command line of windows command prompt with the following command:

```
naken_util -msp430 -disasm RetrievedHEX_Stripped.txt > ReverseMe.txt
```

Figure 1. Naken_util Used in Command Prompt.

This command utilizes naken_util with the MSP430 architecture and informs it to disassemble the hex file provided and store the result in ReverseMe.txt.

MSP430 Flasher: The MSP430 flasher utility has several useful functions. The first of which is it allows the user to extract things such as machine code from a targeted file or platform and save it to a designated output file in either text or hex form. The output file that it makes, in this case RetrievedHEX.txt, holds hex data from the flash memory. A lot of this memory however is not useful because it simply contains “0xFF” which are considered erased memory locations and are therefore removed from the final output file. These removed memory locations can be found in the other file named RetrievedHEX_Stripped.txt. This tool is very useful for one wanting to extract machine code from various microcontrollers, in this case the MSP430.

Problem 01 (Questions from Tutorial):

Can you find what symbol is associated with #0x3272?

The symbol associated with this address is the “memset” function. This is responsible for clearing the .bss section, with its prototype being “int memset(void *ptr, int fill, size_t nbytes).

It is called with the following parameters: “memset(__bssstart, 0, __bsssize). The following assembly code is what initializes the memset function:

```
00003272 <memset>:  
3272: 0f 4c mov r12, r15.
```

What insights can you glean from your analysis of elf commands?

ReadElf: The readelf command allows one to display a variety of useful information about the ELF format that is contained in the binary file. There is a lot of information regarding the sections, headers and the amount of data it takes to contain them.

Elf-NM: The elf-nm commands allow one to list the symbols that are located in object files. It can show the symbol tables of an object file which has the names of the symbols as well as their types and values. This can certainly help one understand which symbols are referenced and/or defined in the code.

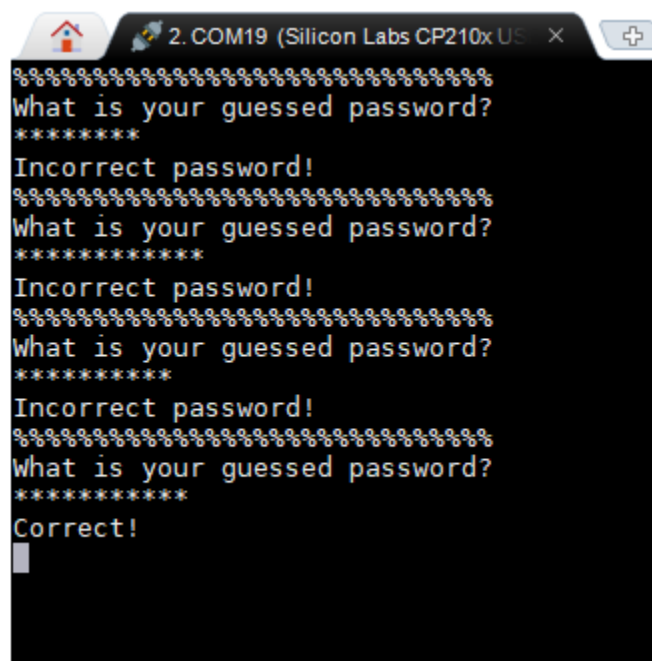
Elf-Symbols: This is a very similar command to elf-nm but simply provides a different format for displaying the symbol table.

Elf-Objdump: This is a command that displays various information that exists in the object files. It can also show the disassembly of various sections in the object file which may include sections of data and code which might help one understand the assembly code behind it. With all

of this information, it is clear that these available commands are instrumental in gaining insight and knowledge into the structure of binary files.

Problem 02:

In problem two of this lab was to find and guess the correct password in MobaXTerm using UART serial connection. There was a list of passwords that were available in the .out file that was provided for this lab. This file was opened by means of CCS and the passwords were located. The password that worked for my file was: “qua5drupe4d”. After this password was entered, the console on UART sent a message confirming that the password was, in fact, correct.

A screenshot of the MobaXTerm UART console window. The window title is "2. COM19 (Silicon Labs CP210x USB to UART Bridge)". The console output shows a series of prompts and responses. It starts with a separator line of equals signs, followed by "What is your guessed password?", a line of asterisks for input, and "Incorrect password!". This sequence repeats three times. Finally, after another input line of asterisks, it displays "Correct!". A cursor is visible on the line following "Correct!".

```
=====
What is your guessed password?
*****
Incorrect password!
=====
What is your guessed password?
*****
Incorrect password!
=====
What is your guessed password?
*****
Incorrect password!
=====
What is your guessed password?
*****
Correct!
█
```

Figure 2. Screenshot of MobaXTerm UART Console Showing Correct Password Guess.

Problem 02 Bonus:

An attempt was made to find the master password for this crack_me file. I found a tool that displays the binary values in hex and text format which the file was loaded into where I could look at some of the strings that were being passed back and forth in the file. I was able to find

this section of code where the master password is confirmed but I could not find the password itself:

```

seg000:0000000000000362      db  43h ; C
seg000:0000000000000363      db  6Fh ; o
seg000:0000000000000364  aRrectUnlockedU db  'rrect. Unlocked using master password.',0Ah
seg000:000000000000038B      db  0Dh,0
seg000:000000000000038D      align 2
seg000:000000000000038E      db  0Ah
seg000:000000000000038F      db  0Dh
seg000:0000000000000390      db  43h ; C
seg000:0000000000000391      db  6Fh ; o
seg000:0000000000000392  aRrect      db  'rrect!',0Ah
seg000:0000000000000399      db  0Dh,0
seg000:000000000000039B      align 4
seg000:000000000000039C      db  0Ah
seg000:000000000000039D      db  0Dh
seg000:000000000000039E      db  49h ; I
seg000:000000000000039F      db  6Eh ; n
seg000:00000000000003A0  aCorrectPasswor db  'correct password!',0Ah
seg000:00000000000003B2      db  0Dh,0
seg000:00000000000003B4      db  25h ; %
seg000:00000000000003B5      db  25h ; %
seg000:00000000000003B6      db  25h ; %
seg000:00000000000003B7      db  25h ; %
seg000:00000000000003B8      db  25h ; %
seg000:00000000000003B9      db  '%%%%%%%%%',0Dh,0Ah,0
seg000:00000000000003D5      align 2
seg000:00000000000003D6  aWhatIsYourGues db  'What is your guessed password?',0Ah
seg000:00000000000003F5      db  0Dh,0

```

Figure 3. Screenshot of IDA Program Assembling .out File.

I was also able to find the locations in the file of the other passwords but again, was not able to find the master password.

```

0000000000000320  30 41 75 66 67 61 74 6F 72 73 38 31 00 00 00 00 0Aufgators81....
0000000000000330  71 75 61 35 64 72 75 70 65 34 64 00 61 6E 63 68 qua5drupe4d.anch
0000000000000340  6F 72 31 35 21 00 73 61 6E 67 75 69 6E 65 31 3F or15!.sanguine1?
0000000000000350  00 00 35 4E 6F 6E 63 6F 6D 6D 69 74 74 61 6C 00 ..5Noncommittal.
0000000000000360  0A 0D 43 6F 72 72 65 63 74 2E 20 55 6E 6C 6F 63 ..Correct.·Unloc
0000000000000370  68 65 64 20 75 73 69 6E 67 20 6D 61 73 74 65 72 ked·using·master
0000000000000380  20 70 61 73 73 77 6F 72 64 2E 0A 0D 00 00 0A 0D ·password.....
0000000000000390  43 6F 72 72 65 63 74 21 0A 0D 00 00 0A 0D 49 6E Correct!.....In
00000000000003A0  63 6F 72 72 65 63 74 20 70 61 73 73 77 6F 72 64 correct·password
00000000000003B0  21 0A 0D 00 25 25 25 25 25 25 25 25 25 25 25 25 !...%%%%%%%%%
00000000000003C0  25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 %%%%%%%%%%%%%%
00000000000003D0  25 25 0D 0A 00 00 57 68 61 74 20 69 73 20 79 6F %%....What·is·yo
00000000000003E0  75 72 20 67 75 65 73 73 65 64 20 70 61 73 73 77 ur·guessed·passw
00000000000003F0  6F 72 64 3F 0A 0D 00 00 43 50 45 33 32 35 5F 75 ord?....CPE325_u
0000000000000400  73 65 72 00 1C 0A 14 38 0A 15 31 38 64 3B 00 00 ser....8..18d;..
0000000000000410  32 D0 10 00 FD 3F 03 43 FF FF 00 00 DC 34 DC 34 2....?.C.....

```

Figure 4. Screenshot of Other Passwords.

Problem 03:

Problem three consisted of finding several pieces of information by using a command in the windows command prompt. This command that was used and the information that was found is as follows:

```
C:\Users\ma0133\project11>msp430-elf-readelf -h Lab11_crack_me_michaelagnew.out
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                               ELF32
  Data:                                   2's complement, little endian
  Version:                             1 (current)
  OS/ABI:                               UNIX - System V
  ABI Version:                         0
  Type:                                 EXEC (Executable file)
  Machine:                             Texas Instruments msp430 microcontroller
  Version:                             0x1
  Entry point address:                 0x33aa
  Start of program headers:            39536 (bytes into file)
  Start of section headers:           39696 (bytes into file)
  Flags:                               0x0
  Size of this header:                 52 (bytes)
  Size of program headers:             32 (bytes)
  Number of program headers:           5
  Size of section headers:            40 (bytes)
  Number of section headers:          67
  Section header string table index: 66

C:\Users\ma0133\project11>_
```

Figure 5. ReadElf Command and its Output in Command Prompt.

a. What is the magic number used?

ANS: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00

b. What is the class of this .out file?

ANS: ELF32

c. What machine was this file built for?

ANS: Texas Instruments msp430 microcontroller.

d. What is the size of the header?

ANS: 52 bytes.

e. How many section headers are there?

ANS: There are 67 section headers.

Problem 04:

- a) The first step of this problem was to program the MSP430 with the given code using the flasher utility. The following shows the output from this action:

```
C:\Users\ma0133\Desktop\MSPFlasher_1.3.20>MSP430Flasher.exe MSP430FG4618 -w Lab11_reverse_me.txt -v -g -z [VCC]
* -----/|-----*
* /|_ *
* /_|_ MSP Flasher v1.3.20 *
* |_/ *
* -----/|-----*
*
* Evaluating triggers...done
* Checking for available FET debuggers:
* Found USB FET @ COM29 <- Selected
* Initializing interface @ COM29...done
* Checking firmware compatibility:
* FET firmware is up to date.
* Reading FW version...done
* Setting VCC to 3000 mV...done
* Accessing device...done
* Reading device information...done
*
* -----
* Arguments : MSP430FG4618 -w Lab11_reverse_me.txt -v -g -z [VCC]
* -----
* Driver : loaded
* Dll Version : 31400000
* FwVersion : 31200000
* Interface : TIUSB
* HwVersion : U 3.0
* JTAG Mode : AUTO
* Device : MSP430FG4618
* EEM : Level 3, ClockCntrl 2
* VCC OFF
*
* -----
* Powering down...done
* Disconnecting from device...done
*
* -----
* Driver : closed (No error)
* -----
*/
C:\Users\ma0133\Desktop\MSPFlasher_1.3.20>
```

Figure 6. Output from MSP430Flasher.exe.

- b) The second step of this problem was to guess approximately what the program that was provided does. When the program is started initially, there is nothing that seems to be happening. However, when switch 2 is pressed, the green and yellow LED blink at a slightly different rate which allows one to be on for a small amount of time after the other goes off. The following image was captured while the board had both LEDs on during the program.

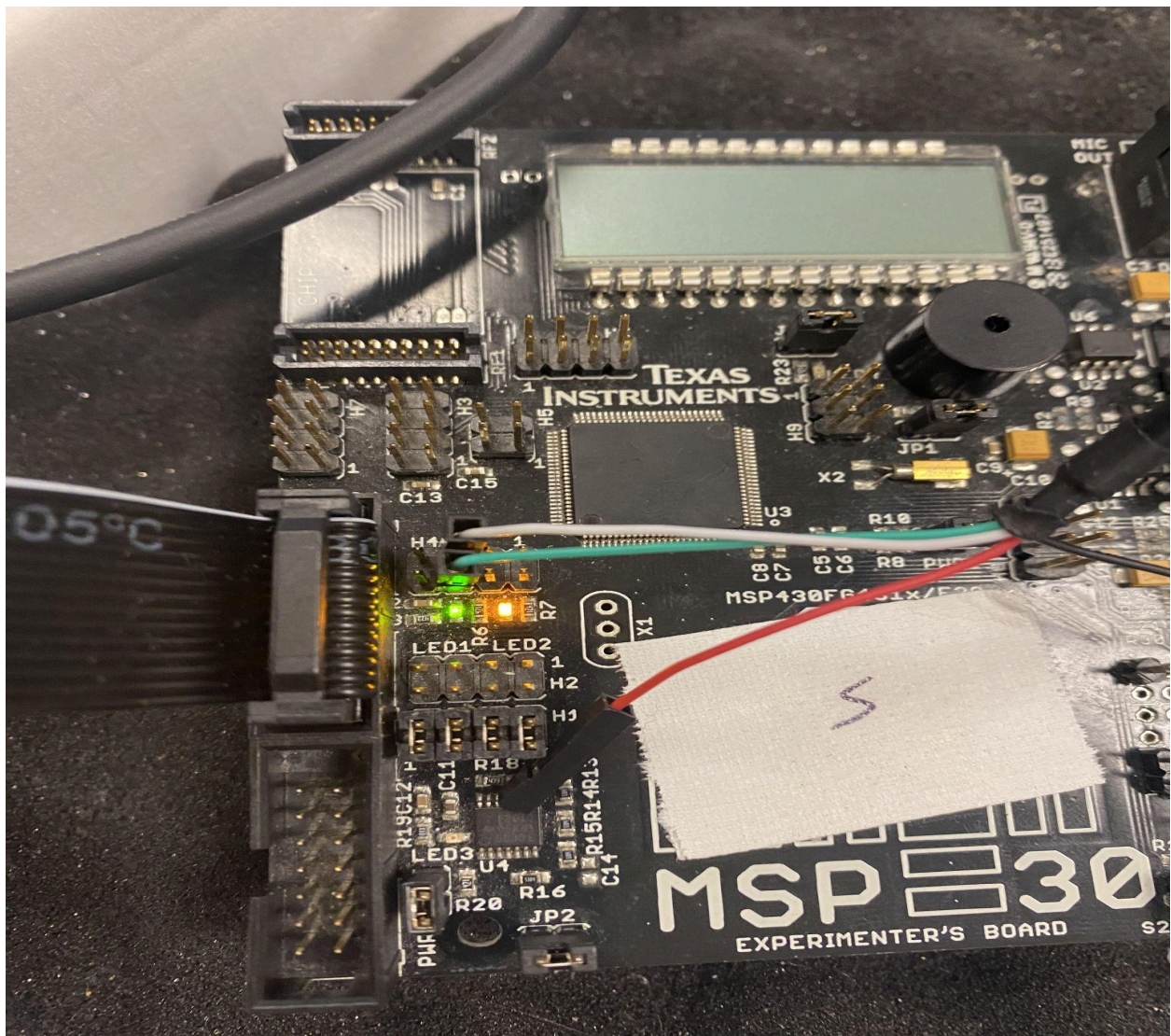


Figure 7. LED Output from Board.

- c) The third part of this problem was using the `naken_utility` tool to disassemble a hex file that was provided named `Lab11_reverse_me.txt`. The command run was:

`Naken_util -msp430 -disasm Lab11_reverse_me.txt > ReverseMe.txt`

This resulted in an output file named “ReverseMe.txt” which then contained the resulting assembly code.

```
C:\Users\ma0133\Desktop\yay\MSPFlasher_1.3.20>naken_util -msp430 -disasm Lab11_reverse_me.txt > ReverseMe.txt
C:\Users\ma0133\Desktop\yay\MSPFlasher_1.3.20>more ReverseMe.txt

naken_util - by Michael Kohn
              Joe Davisson
  Web: http://www.mikekohn.net/
  Email: mike@mikekohn.net

Version:

Loaded ti_txt Lab11_reverse_me.txt from 0x3100 to 0x1ffff
Type help for a list of commands.
```

Addr	Opcode	Instruction	Cycles
0x3100:	0x8321	sub.w #2, SP	1
0x3102:	0x40b2	mov.w #0x5a80, &0x0120	5
0x3104:	0x5a80		
0x3106:	0x0120		
0x3108:	0xd0f2	bis.b #0x06, &0x002a	5
0x310a:	0x0006		
0x310c:	0x002a		
0x310e:	0x43c2	mov.b #0, &0x0029	4
0x3110:	0x0029		
0x3112:	0xe0f2	xor.b #0x06, &0x0029	5
0x3114:	0x0006		
0x3116:	0x0029		
0x3118:	0x4381	mov.w #0, 0(SP)	4
0x311a:	0x0000		
0x311c:	0x90b1	cmp.w #0xc350, 0(SP)	5
0x311e:	0xc350		
0x3120:	0x0000		
0x3122:	0x2ff7	jhs 0x3112 (offset: -18)	2
0x3124:	0x5391	add.w #1, 0(SP)	4
0x3126:	0x0000		
0x3128:	0x90b1	cmp.w #0xc350, 0(SP)	5
0x312a:	0xc350		
0x312c:	0x0000		
0x312e:	0x2bfa	jlo 0x3124 (offset: -12)	2
0x3130:	0x3ff0	jmp 0x3112 (offset: -32)	2
0x3132:	0x4031	mov.w #0x3100, SP	2
0x3134:	0x3100		
0x3136:	0x12b0	call #0x314c	5
0x3138:	0x314c		
0x313a:	0x430c	mov.w #0, r12	1
0x313c:	0x12b0	call #0x3100	5
0x313e:	0x3100		
0x3140:	0x431c	mov.w #1, r12	1
0x3142:	0x12b0	call #0x3146	5
0x3144:	0x3146		
0x3146:	0x4303	nop -- mov.w #0, CG	1
0x3148:	0x3fff	jmp 0x3148 (offset: -2)	2
0x314a:	0x4303	nop -- mov.w #0, CG	1
0x314c:	0x431c	mov.w #1, r12	1
0x314e:	0x4130	ret -- mov.w @SP+, PC	3
0x3150:	0xd032	bis.w #0x0010, SR	2
0x3152:	0x0010		
0x3154:	0x3ffd	jmp 0x3150 (offset: -6)	2
0x3156:	0x4303	nop -- mov.w #0, CG	1

Figure 8. Resulting File from `naken_utility`.

- d) The fourth step of this problem was to comment the resulting assembly code and to describe what it does.

```
0x3100: 0x40b2 mov.w #0x5a80, &0x0120 ; This initializes the switch
; These lines all have to do with switch and LED setup
0x3106: 0xd2e2 bis.b #4, &0x002a ;
0x310a: 0xd3e2 bis.b #2, &0x002a ; P1.2 LED2
0x310e: 0xc2e2 bic.b #4, &0x0029 ; P1.4 LED1
0x3112: 0xc3e2 bic.b #2, &0x0029 ; P1.2 LED2
0x3116: 0xc3d2 bic.b #1, &0x0022 ; P1.1 Switch setup
0x311a: 0xc3e2 bic.b #2, &0x0022 ; P1.1 Switch setup
0x311e: 0xb3e2 bit.b #2, &0x0020 ; check if switch pressed
0x3122: 0x23fd jne 0x311e (offset: -6) ; if not pressed loop back
0x3124: 0x120d push.w r13 ; push R13 onto stack
0x3126: 0x403d mov.w #0x031d, r13 ; load R13 with a delay value
0x312a: 0x831d sub.w #1, r13 ; subtract R13 by 1
0x312c: 0x23fe jne 0x312a (offset: -4) ; if R13 is not 0, continue delay loop
0x312e: 0x413d pop.w r13 ; pop delay into R13
0x3130: 0x3c00 jmp 0x3132 (offset: 0) ; jump back to main loop

; Main part of code with Loop
0x3132: 0xb3e2 bit.b #2, &0x0020 ; check if the switch is pressed
0x3136: 0x23f3 jne 0x311e (offset: -26) ; if pressed, go back
0x3138: 0xb3e2 bit.b #2, &0x0020 ; check if switch is released
0x313c: 0x23fd jne 0x3138 (offset: -6) ; keep checking if not
0x313e: 0x120d push.w r13 ; push R13 onto stack
0x3140: 0x403d mov.w #0x031d, r13 ; load R13 with a delay value
0x3144: 0x831d sub.w #1, r13 ; decrement R13
0x3146: 0x23fe jne 0x3144 (offset: -4) ; if R13 is not 0, continue delay
0x3148: 0x413d pop.w r13 ; pop delay into R13
0x314a: 0x3c00 jmp 0x314c (offset: 0) ; jump to next toggle for LED

; This segment of code controls the first LED
0x314c: 0xb3e2 bit.b #2, &0x0020 ; check if switch is pressed
0x3150: 0x23f3 jne 0x3138 (offset: -26) ; if pressed, continue
0x3152: 0xb3d2 bit.b #1, &0x0020 ; check if it's released
0x3156: 0x23fd jne 0x3152 (offset: -6) ; if not released wait until it is
0x3158: 0x120d push.w r13 ; push R13 onto stack
0x315a: 0x403d mov.w #0x031d, r13 ; load R13 with a delay value
0x315e: 0x831d sub.w #1, r13 ; subtract 1 from R13
0x3160: 0x23fe jne 0x315e (offset: -4) ; continue loop if R13 is not 0
0x3162: 0x413d pop.w r13 ; pop delay into R13
0x3164: 0x3c00 jmp 0x3166 (offset: 0) ; jump to next LED
```

```

; This segment of code controls the second LED
0x3166: 0xb3d2 bit.b #1, &0x0020      ; check if switch pressed
0x316a: 0x23f3 jne 0x3152 (offset: -26) ; if pressed continue
0x316c: 0xe2e2 xor.b #4, &0x0029      ; turn on LED
0x3170: 0x120d push.w r13              ; push R13 onto stack
0x3172: 0x120e push.w r14              ; push R14 onto stack
0x3174: 0x403d mov.w #0x8b80, r13      ; load R13 with a delay value
0x3178: 0x430e mov.w #0, r14           ; set R14 to 0
0x317a: 0x831d sub.w #1, r13            ; decrement R13
0x317c: 0x730e subc.w #0, r14          ; decrement R14 with carry bit
0x317e: 0x23fd jne 0x317a (offset: -6) ; continue loop until R13 is 0
0x3180: 0x930d cmp.w #0, r13           ; compare R13 to 0
0x3182: 0x23fb jne 0x317a (offset: -10) ; if not 0, continue delay
0x31b2: 0x12b0 call #0x31c8            ; Subroutine call to different memory address
0x31b6: 0x430c mov.w #0, r12           ; Loading 0 into R12
0x31b8: 0x12b0 call #0x3100            ; another subroutine call to different memory
0x31bc: 0x431c mov.w #1, r12           ; setting R12 to 1
0x31be: 0x12b0 call #0x31c2            ; calling another subroutine

```

- e) The final step of this problem was to make a flowchart for the commented code. This is shown below:

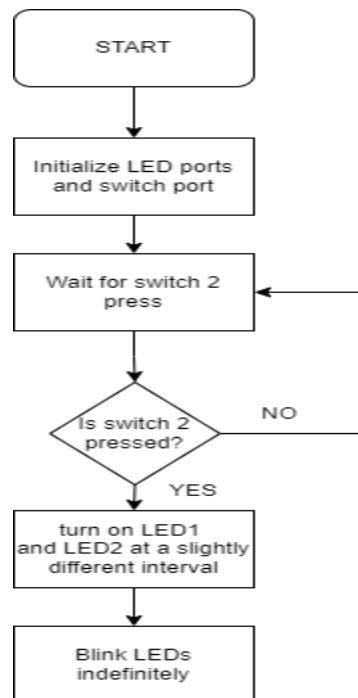


Figure 9. Flowchart for Disassembled Code

Conclusion:

Fortunately, no issues were encountered over the course of this lab. Everything that was required to be done was done so and in the proper way. The password was found as required, the reverse_me file was programmed to the board using the flasher, and the naken_utility tool was able to be used in order to convert the file into assembly. Progress was made in finding the master password but unfortunately it was not found.

Appendix:

```
0x3100: 0x40b2 mov.w #0x5a80, &0x0120 ; This initializes the switch
; These lines all have to do with switch and LED setup
0x3106: 0xd2e2 bis.b #4, &0x002a ;
0x310a: 0xd3e2 bis.b #2, &0x002a ; P1.2 LED2
0x310e: 0xc2e2 bic.b #4, &0x0029 ; P1.4 LED1
0x3112: 0xc3e2 bic.b #2, &0x0029 ; P1.2 LED2
0x3116: 0xc3d2 bic.b #1, &0x0022 ; P1.1 Switch setup
0x311a: 0xc3e2 bic.b #2, &0x0022 ; P1.1 Switch setup
0x311e: 0xb3e2 bit.b #2, &0x0020 ; check if switch pressed
0x3122: 0x23fd jne 0x311e (offset: -6) ; if not pressed loop back
0x3124: 0x120d push.w r13 ; push R13 onto stack
0x3126: 0x403d mov.w #0x031d, r13 ; load R13 with a delay value
0x312a: 0x831d sub.w #1, r13 ; subtract R13 by 1
0x312c: 0x23fe jne 0x312a (offset: -4) ; if R13 is not 0, continue delay loop
0x312e: 0x413d pop.w r13 ; pop delay into R13
0x3130: 0x3c00 jmp 0x3132 (offset: 0) ; jump back to main loop

; Main part of code with Loop
0x3132: 0xb3e2 bit.b #2, &0x0020 ; check if the switch is pressed
0x3136: 0x23f3 jne 0x311e (offset: -26) ; if pressed, go back
0x3138: 0xb3e2 bit.b #2, &0x0020 ; check if switch is released
0x313c: 0x23fd jne 0x3138 (offset: -6) ; keep checking if not
0x313e: 0x120d push.w r13 ; push R13 onto stack
0x3140: 0x403d mov.w #0x031d, r13 ; load R13 with a delay value
0x3144: 0x831d sub.w #1, r13 ; decrement R13
0x3146: 0x23fe jne 0x3144 (offset: -4) ; if R13 is not 0, continue delay
0x3148: 0x413d pop.w r13 ; pop delay into R13
0x314a: 0x3c00 jmp 0x314c (offset: 0) ; jump to next toggle for LED
```

; This segment of code controls the first LED

0x314c: 0xb3e2 bit.b #2, &0x0020	; check if switch is pressed
0x3150: 0x23f3 jne 0x3138 (offset: -26)	; if pressed, continue
0x3152: 0xb3d2 bit.b #1, &0x0020	; check if it's released
0x3156: 0x23fd jne 0x3152 (offset: -6)	; if not released wait until it is
0x3158: 0x120d push.w r13	; push R13 onto stack
0x315a: 0x403d mov.w #0x031d, r13	; load R13 with a delay value
0x315e: 0x831d sub.w #1, r13	; subtract 1 from R13
0x3160: 0x23fe jne 0x315e (offset: -4)	; continue loop if R13 is not 0
0x3162: 0x413d pop.w r13	; pop delay into R13
0x3164: 0x3c00 jmp 0x3166 (offset: 0)	; jump to next LED

; This segment of code controls the second LED

0x3166: 0xb3d2 bit.b #1, &0x0020	; check if switch pressed
0x316a: 0x23f3 jne 0x3152 (offset: -26)	; if pressed continue
0x316c: 0xe2e2 xor.b #4, &0x0029	; turn on LED
0x3170: 0x120d push.w r13	; push R13 onto stack
0x3172: 0x120e push.w r14	; push R14 onto stack
0x3174: 0x403d mov.w #0x8b80, r13	; load R13 with a delay value
0x3178: 0x430e mov.w #0, r14	; set R14 to 0
0x317a: 0x831d sub.w #1, r13	; decrement R13
0x317c: 0x730e subc.w #0, r14	; decrement R14 with carry bit
0x317e: 0x23fd jne 0x317a (offset: -6)	; continue loop until R13 is 0
0x3180: 0x930d cmp.w #0, r13	; compare R13 to 0
0x3182: 0x23fb jne 0x317a (offset: -10)	; if not 0, continue delay
0x31b2: 0x12b0 call #0x31c8	; Subroutine call to different memory address
0x31b6: 0x430c mov.w #0, r12	; Loading 0 into R12
0x31b8: 0x12b0 call #0x3100	; another subroutine call to different memory
0x31bc: 0x431c mov.w #1, r12	; setting R12 to 1
0x31be: 0x12b0 call #0x31c2	; calling another subroutine