

CPE 325: Intro to Embedded Computer System

Lab 11

Software Reverse Engineering

Submitted by: Anshika Sinha

Date of Experiment: 03/26/2025

Report Deadline: 04/01/2025

Demonstration Deadline: 04/07/2025

Theory

Topic 1: ELF File components

- a) The Executable and Linkable File (ELF) format is a standard used for executable files, object files, shared libraries, and core dumps. It is independent of the Instruction Set Architecture (ISA) or operating system.
- b) The ELF File Header defines the overall structure of the file. It contains metadata like file type, architecture, and entry point.
- c) The Program Header Table describes memory segments required for runtime execution. It guides the loader on how to create a process image in memory.
- d) Section Header Table describes sections containing data for linking and relocation. The sections include symbol tables, string tables, and other essential information.
- e) Segments contain executable code and data needed at runtime. They are loaded into memory by the program loader with attributes such as type, memory location, permissions (R, W, X), and size.
- f) Sections contain specific types of information such as code, data, or metadata. They are used for linking and relocation rather than direct execution.
- g) The linker combines object files into an executable file or library. The linker script defines the memory layout and section placement. Developers can modify linker scripts to customize memory usage.
- h) Loader is a utility that loads an ELF executable into memory and starts execution.

Topic 2: Naken Utility

- a) The Naken utility is an assembler used in embedded systems development to simplify the process of generating machine-readable code from assembly language instructions.
- b) `naken_asm` translates assembly language code into machine code (binary) that can be executed by a processor or microcontroller.
- c) The assembler produces output in NakenELF, a minimal version of the standard ELF (Executable and Linkable Format).
- d) As an assembler, `naken_asm` is highly optimized for efficiency, producing small and fast binaries, which is important when both memory and processing power are limited.
- e) It provides features like macro processing, conditional assembly, and handling of various architecture-specific instructions. It can also generate object files for use in larger embedded applications.

Topic 3: MSP430 Flasher

- a) The MSP430 Flasher is a command-line utility provided by Texas Instruments for programming and flashing of MSP430 microcontrollers.
- b) It can flash firmware or compiled binary files (like .hex or .elf files) onto the device. It supports multiple file formats, including Intel Hex and Motorola S-record, for flashing firmware onto the device.
- c) It integrates with other TI development tools, such as Code Composer Studio (CCS), for development and debugging.
- d) The Flasher tool provides error checking during the flashing process and can generate detailed logs, helping developers diagnose issues with the programming process.

Report Questions

Question 1:

- a. **Next the subroutine at #0x3272 is called. Can you find what symbol is associated with that address?**

The function at value 0x3273 in the Symbol table is memset.

- b. **Repeat the analysis of the executable using utilities discussed above: msp430-elf-readelf, msp430-elf-nm, msp430-elf-symbols, and msp430-elf-objdump. What insights can you glean from your analysis?**

The commands produce similar results for both the TI and GNU compiler. However, the ELF commands generated by the TI compiler are different from those produced by the GNU compiler, which result in variations in the .out files. While both compilers produce similar outputs, there are differences in the initial section headers, flags, count of section headers, section header string table index, and the magic number.

Question 2:

Use the “.out” file from your Lab 6 Q.2 C program. Create the HEX file using the out file and show the result (HEX file content) in your report. [15 Pts]

@4400

21 83 B2 40 80 5A 5C 01 D2 D3 04 02 F2 D0 80 00

25 02 E2 C3 05 02 E2 D3 07 02 E2 D3 03 02 E2 C3

04 02 E2 D3 06 02 E2 D3 02 02 D2 D3 02 02 F2 F0

7F 00 23 02 3C 40 80 84 3D 40 1E 00 B0 12 C0 44
E2 B3 00 02 0F 20 3C 40 80 96 3D 40 98 00 B0 12
C0 44 B2 40 80 96 00 24 B2 40 98 00 02 24 E2 B3
00 02 FD 27 E2 B3 01 02 17 20 B2 90 0F 00 02 24
10 28 04 20 B2 90 41 42 00 24 0B 28 12 C3 12 10
02 24 12 10 00 24 1C 42 00 24 1D 42 02 24 B0 12
C0 44 E2 B3 01 02 FD 27 D2 E3 02 02 F2 E0 80 00
23 02 81 43 00 00 B1 40 50 C3 00 00 81 93 00 00
C7 27 91 83 00 00 81 93 00 00 FB 23 C1 3F 03 43
21 82 81 4C 00 00 81 4D 02 00 2A 3C B2 40 50 00
62 01 B2 40 2C 11 64 01 4F 3C B2 40 40 00 62 01
B2 40 96 10 64 01 48 3C B2 40 30 00 62 01 B2 40
4B 10 64 01 41 3C B2 40 30 00 62 01 B2 40 3C 10
64 01 3A 3C B2 40 20 00 62 01 B2 40 25 10 64 01
33 3C B2 40 20 00 62 01 B2 40 1E 10 64 01 2C 3C
2F 41 3F 80 40 42 3D 70 0F 00 02 20 0F 93 F1 27
3F 80 90 D0 3D 70 03 00 02 20 0F 93 E3 27 3F 80
B0 71 3D 70 0B 00 02 20 0F 93 D5 27 3F 80 20 A1
3D 70 07 00 02 20 0F 93 C7 27 3F 80 A0 25 3D 70
26 00 02 20 0F 93 B9 27 3F 80 40 4B 3D 70 4C 00
03 20 0F 93 AB 27 00 3C 21 52 30 41 0A 12 09 12
08 12 0A 4C 78 4A 09 43 11 3C 0E 4D 0E 8B 1E 83
1D 53 FD 4E FF FF 1F 83 FB 23 03 3C 1D 53 FD 4A
FF FF 12 C3 08 10 19 53 39 92 EC 37 18 B3 F6 23
7B 4A 7F 4A 0C 4B B0 12 BE 46 0B 4C 0C 4F B0 12
8A 46 3C F0 0F 00 0B DC 3F F0 0F 00 3F 50 03 00
3F 90 12 00 0C 20 7E 4A 3E B0 80 00 07 24 7C 4A

4C 4C B0 12 B8 46 3E F0 7F 00 0E DC 0F 5E 3B 90
FF 0F CB 23 30 40 10 47 0A 12 09 12 08 12 19 42
5C 01 B2 40 80 5A 5C 01 3F 40 34 47 3F 90 38 47
16 24 3F 40 38 47 3F 90 3C 47 11 24 3A 40 3C 47
3A 80 38 47 0A 11 0A 11 38 40 38 47 3C 48 7F 4C
0F 5F 1F 4F 34 47 3D 48 8F 12 1A 83 F7 23 79 C2
39 D0 08 5A 82 49 5C 01 B0 12 22 47 30 40 10 47
3D F0 0F 00 3D E0 0F 00 0D 5D 0D 5D 00 5D 12 C3
0C 10 12 C3 0C 10 12 C3 0C 10 12 C3 0C 10 12 C3
0C 10 12 C3 0C 10 12 C3 0C 10 12 C3 0C 10 12 C3
0C 10 12 C3 0C 10 12 C3 0C 10 12 C3 0C 10 12 C3
0C 10 12 C3 0C 10 12 C3 0C 10 30 41 3D F0 0F 00
3D E0 0F 00 0D 5D 00 5D 0C 5C 0C 5C 0C 5C 0C 5C
0C 5C 0C 5C 0C 5C 0C 5C 0C 5C 0C 5C 0C 5C 0C 5C
0C 5C 0C 5C 0C 5C 30 41 31 40 00 44 B0 12 1E 47
0C 93 02 24 B0 12 F8 45 0C 43 B0 12 00 44 1C 43
B0 12 18 47 0F 4C 0C 4D 3D 40 03 00 0D 5F 1E 4F
01 00 30 40 F6 46 0E 93 06 24 0F 4C 1F 53 FF 4D
FF FF 1E 83 FB 23 30 41 34 41 35 41 36 41 37 41
38 41 39 41 3A 41 30 41 03 43 FF 3F 03 43 1C 43
30 41 30 41 32 D0 10 00 FD 3F 03 43 01 00 04 00
80 84 1E 00 7C 45 E4 46 2C 47 00 24

@ffd2

24 47 24 47 24 47 24 47 24 47 24 47 24 47
24 47 24 47 24 47 24 47 24 47 24 47 24 47
24 47 24 47 24 47 24 47 24 47 24 47 C8 46

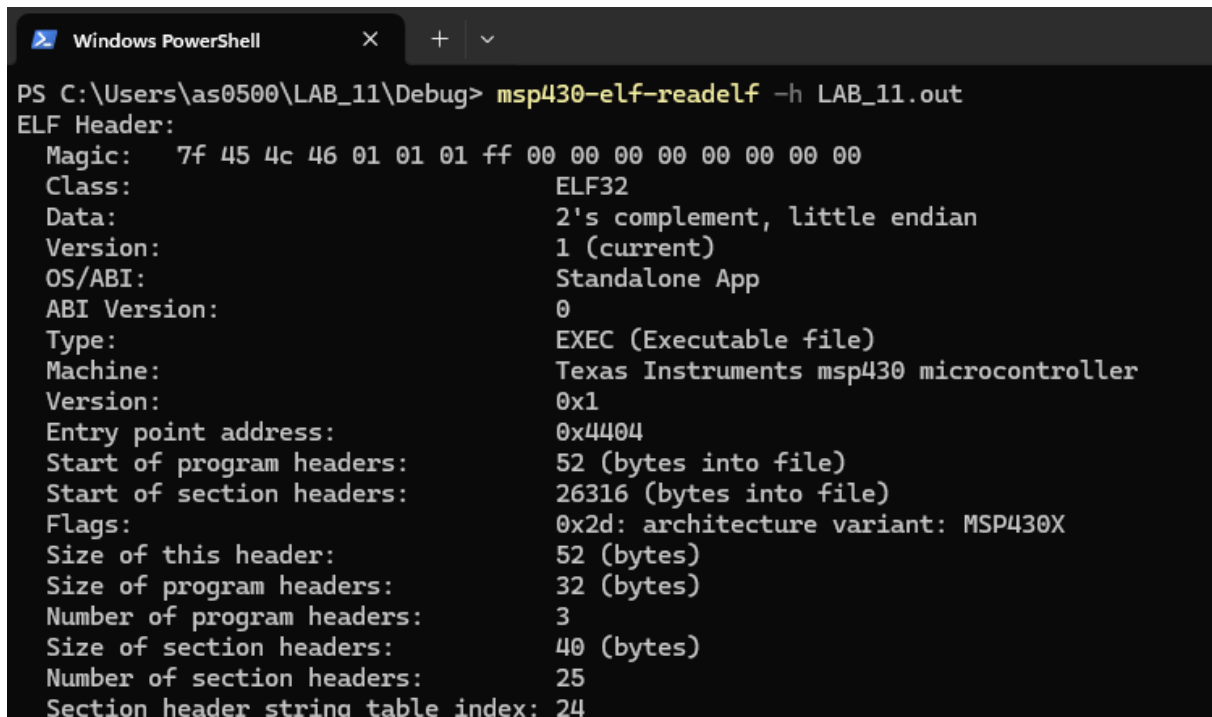
q

Question 3:

From the same .out file from Q.2, find the following relevant information. What tool did you use? Take a screenshot and put it in your report.

The tool I used is elf-reader in MSP430 GCC. The command I executed is:

```
msp430-elf-readelf -h LAB_11.out
```



```
Windows PowerShell
PS C:\Users\as0500\LAB_11\Debug> msp430-elf-readelf -h LAB_11.out
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 ff 00 00 00 00 00 00 00
  Class:                               ELF32
  Data:                                   2's complement, little endian
  Version:                               1 (current)
  OS/ABI:                                Standalone App
  ABI Version:                           0
  Type:                                   EXEC (Executable file)
  Machine:                                Texas Instruments msp430 microcontroller
  Version:                                0x1
  Entry point address:                    0x4404
  Start of program headers:                52 (bytes into file)
  Start of section headers:                26316 (bytes into file)
  Flags:                                   0x2d: architecture variant: MSP430X
  Size of this header:                     52 (bytes)
  Size of program headers:                 32 (bytes)
  Number of program headers:                3
  Size of section headers:                 40 (bytes)
  Number of section headers:                25
  Section header string table index:       24
```

Figure 01: ELF file header information in MSP430 GCC

a. What is the magic number used? [2 pts]

7f 45 4c 46 01 01 01 ff 00 00 00 00 00 00 00

b. What is the class of this .out file? [2 pts]

The class of the .out file is ELF32.

c. What machine was this file built for? [2 pts]

This file was built for Texas Instruments msp430 microcontroller.

d. What is the size of the header? [2pts]

The size of the header is 52 bytes.

e. How many section headers are there? Please verify. You may need to run the command again. [7 pts]

There are 25 section headers.

```

Windows PowerShell
PS C:\Users\as0500\LAB_11\Debug> msp430-elf-readelf --section-headers LAB_11.out
There are 25 section headers, starting at offset 0x66cc:

Section Headers:
[Nr] Name                Type           Addr      Off      Size    ES Flg Lk Inf Al
[ 0]                      NULL          00000000  000000  000000  00  0  0  0  0
[ 1] __reset_vector         PROGBITS      0000ffff  0002d2  000002  00  A  0  0  1
[ 2] .lower.rodata          PROGBITS      00004400  0002d4  000000  00  W  0  0  1
[ 3] .rodata                PROGBITS      00004400  0002d4  000000  00  WA 0  0  1
[ 4] .rodata2               PROGBITS      00004400  0002d4  000000  00  W  0  0  1
[ 5] .data                  PROGBITS      00002400  000094  000004  00  WA 0  0  2
[ 6] .bss                   NOBITS        00002404  000000  000000  00  WA 0  0  1
[ 7] .noinit                PROGBITS      00002404  0002d4  000000  00  W  0  0  1
[ 8] .heap                  NOBITS        00002404  000098  000004  00  WA 0  0  1
[ 9] .lowtext               PROGBITS      00004404  000098  00001e  00  AX 0  0  1
[10] .lower.text            PROGBITS      00004422  0002d4  000000  00  W  0  0  1
[11] .text                  PROGBITS      00004422  0000b6  00021a  00  AX 0  0  2
[12] .upper.text            PROGBITS      00010000  0002d4  000000  00  W  0  0  1
[13] .MSP430.attribute     MSP430_ATTRIB 00000000  0002d4  000026  00  0  0  0  1
[14] .comment               PROGBITS      00000000  0002fa  000039  01  MS 0  0  1
[15] .debug_aranges         PROGBITS      00000000  000333  000020  00  0  0  0  1
[16] .debug_info            PROGBITS      00000000  000353  002b39  00  0  0  0  1
[17] .debug_abbrev           PROGBITS      00000000  002e8c  0000dc  00  0  0  0  1
[18] .debug_line            PROGBITS      00000000  002f68  0003b8  00  0  0  0  1
[19] .debug_frame           PROGBITS      00000000  003320  000040  00  0  0  0  4
[20] .debug_str             PROGBITS      00000000  003360  001c2c  01  MS 0  0  1
[21] .debug_loc             PROGBITS      00000000  004f8c  000013  00  0  0  0  1
[22] .symtab                SYMTAB        00000000  004fa0  001030  10  23 223 4
[23] .strtab                STRTAB        00000000  005fd0  00060b  00  0  0  0  1
[24] .shstrtab              STRTAB        00000000  0065db  0000ef  00  0  0  0  1

```

Figure 02: Verified section headers in MSP430 GCC

Question 4:

Use the HEX file you generated in Q.2 to do the followings:

- Program the given hex file to your microcontroller using the MSP430Flasher tool and paste the output in your report. [10 pts]

```

Windows PowerShell
PS C:\Users\as0500\Lab11_2\Debug> MSP430Flasher.exe -n MSP430F5529 -w Lab11_2.txt -v -z [Vcc]

*-----*
* /      \ MSP Flasher v1.3.20 *
* \      / *
*-----*

* Evaluating triggers...done
* Checking for available FET debuggers:
* Found USB FET @ COM25 <- Selected
* Initializing interface @ COM25...done
* Checking firmware compatibility:
* FET firmware is up to date.
* Reading FW version...
* Debugger does not support target voltages other than 3000 mV!
* Setting VCC to 3000 mV...done
* Accessing device...done
* Reading device information...done
* Loading file into device...done
* Verifying memory (Lab11_2.txt)...done

*-----*
* Arguments : -n MSP430F5529 -w Lab11_2.txt -v -z [Vcc]
*-----*
* Driver      : Loaded
* DLL Version : 31400000
* FWVersion   : 31200000
* Interface   : TIUSB
* HWVersion   : E 3.0
* JTAG Mode   : AUTO
* Device      : MSP430F5529
* EEM         : Level 7, ClockCtrl 2
* Erase Mode   : ERASE_ALL
* Prog.File   : Lab11_2.txt
* Verified    : TRUE
* BSL Unlock  : FALSE
* InfoA Access: FALSE
* VCC ON      : 3000 mV

*-----*
* Starting target code execution...done
* Disconnecting from device...done
*-----*
* Driver      : closed (No error)
*-----*

```

Figure 03: Output from MSP430 Flasher

b. Show that using the Flasher you can change the blinking frequency as you could in the CCS environment. [15 pts]

<https://drive.google.com/file/d/1FtK7Ohou-yT2m6vLN1KF3CqBfKr5hib2/view?usp=sharing>

c. Using the naked utility and the steps shown in Section 5.2 of the tutorial, reverse engineer the hex file to assembly code. [5 pts]

naken_util - by Michael Kohn

Joe Davisson

Web: <http://www.mikekohn.net/>

Email: mike@mikekohn.net

Version:

Loaded ti_txt RetrievedHEX.txt from 0x4400 to 0x473b

Type help for a list of commands.

Addr	Opcode Instruction	Cycles
-----	-----	-----
0x4400:	0x8321 sub.w #2, SP	1
0x4402:	0x40b2 mov.w #0x5a80, &0x015c	5

... (see Appendix)

d. Comment on each line of the assembly code generated from Q4c above to describe what each line is doing. Try to identify delay loops, switch inputs, LED outputs. [10 pts]

Appendix [Table 01]

e. Describe what the program is doing in a neat flowchart. You can also write a paragraph to describe in addition to the flowchart. [10 pts]

Program Description:

This program interfaces Switch 1, Switch 2, and the two LEDs. Initially, LED1 is on and LED2 is off. The clock frequency is set to 2 MHz. LED1 and LED2 blink using a 50,000 interaction loop delay. Every time SW2 is pressed, the clock frequency is set to 10 MHz. Everytime SW1 is pressed, the clock frequency is halved, and it cannot go under 1 MHz.

Program Flowchart:

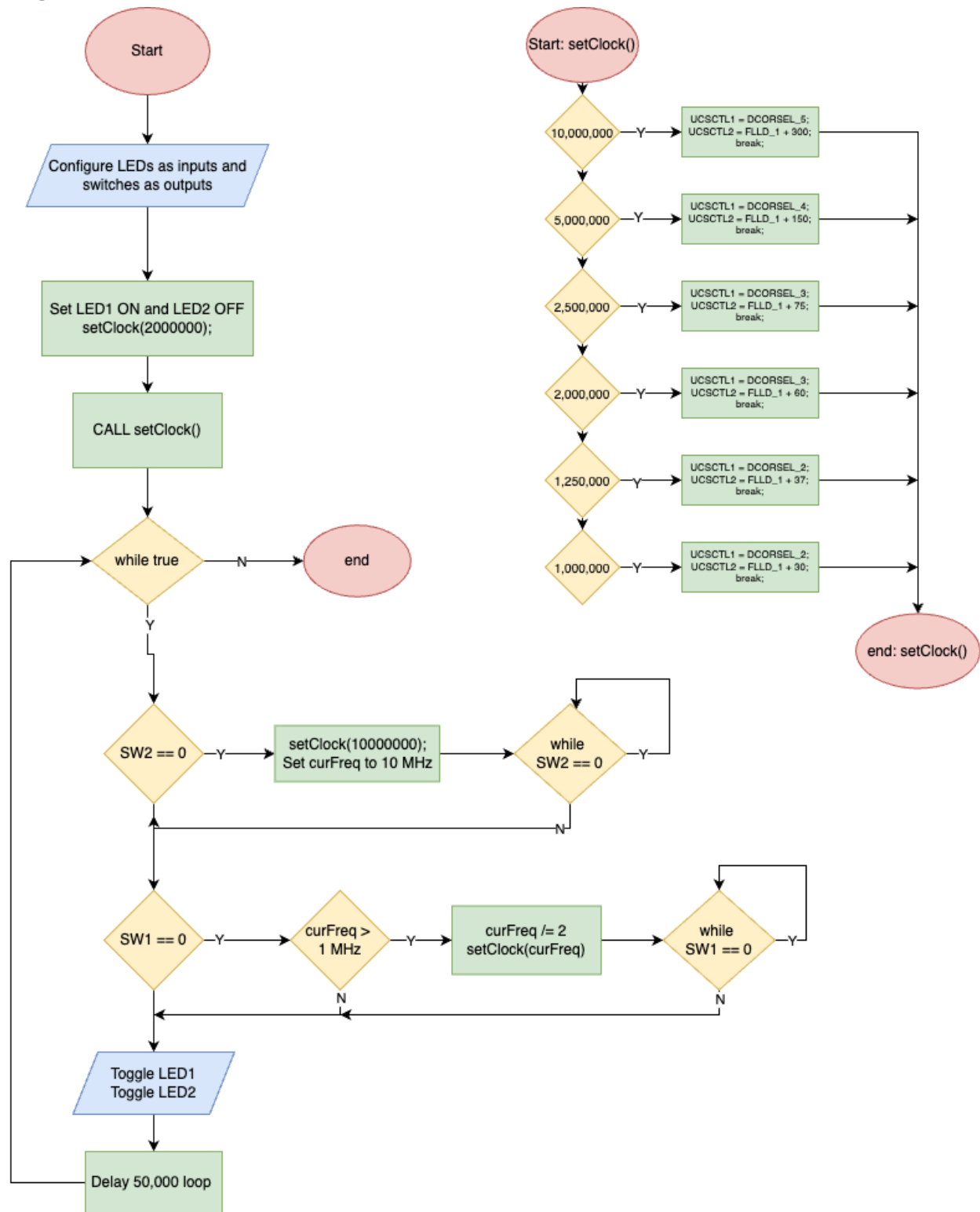


Figure 04: Program Flowchart

Appendix

Table 01: Reverse Engineering Disassembled Code

naken_util - by Michael Kohn Joe Davisson Web: http://www.mikekohn.net/ Email: mike@mikekohn.net			
Version:			
Loaded ti_txt RetrievedHEX.txt from 0x4400 to 0x473b Type help for a list of commands.			
Addr	Opcode	Instruction	Cycles
0x4400:	0x8321	sub.w #2, SP	1 ; Allocate space on stack
0x4402:	0x40b2	mov.w #0x5a80, &0x015c	5 ; STOP WDT,
0x0120:		WDCTL	
0x4404:	0x5a80		
0x4406:	0x015c		
0x4408:	0xd3d2	bis.b #1, &0x0204	4 ; Set P2.0 to output
0x440a:	0x0204		
0x440c:	0xd0f2	bis.b #0x80, &0x0225	5 ; Set P4.7 to output
0x440e:	0x0080		
0x4410:	0x0225		
0x4412:	0xc3e2	bic.b #2, &0x0205	4 ; Clear BIT1,
		set P2.1 as input for Switch 1	
0x4414:	0x0205		
0x4416:	0xd3e2	bis.b #2, &0x0207	4 ; Enable pull-up register at P2.1
0x4418:	0x0207		
0x441a:	0xd3e2	bis.b #2, &0x0203	4 ; Set P2OUT
0x441c:	0x0203		
0x441e:	0xc3e2	bic.b #2, &0x0204	4 ; Clear BIT1,
		set P1.1 as input for Switch 2	
0x4420:	0x0204		
0x4422:	0xd3e2	bis.b #2, &0x0206	4 ; Enable pull-up register at P1.1
0x4424:	0x0206		
0x4426:	0xd3e2	bis.b #2, &0x0202	4 ; Set P1OUT
0x4428:	0x0202		
0x442a:	0xd3d2	bis.b #1, &0x0202	4 ; Set P1OUT to 0x01 (LED1 ON)
0x442c:	0x0202		
0x442e:	0xf0f2	and.b #0x7f, &0x0223	5 ; Clear bit to Turn LED2 OFF
0x4430:	0x007f		
0x4432:	0x0223		
0x4434:	0x403c	mov.w #0x8480, r12	2 ; Move value to register for subroutine
0x4436:	0x8480		

0x4438: 0x403d mov.w #0x001e, r13	2	; Move value to
register for subroutine		
0x443a: 0x001e		
0x443c: 0x12b0 call #0x44c0	5	; CALL
setClock() subroutine		
0x443e: 0x44c0		
0x4440: 0xb3e2 bit.b #2, &0x0200	4	; Test BIT1 to
see if SW1 is pressed		
0x4442: 0x0200		
0x4444: 0x200f jne 0x4464 (offset: 30)	2	; Jump if button
is not pressed		
0x4446: 0x403c mov.w #0x9680, r12	2	; Load 0x9680 to
r12		
0x4448: 0x9680		
0x444a: 0x403d mov.w #0x0098, r13	2	; Load 0x0098 to
r13		
0x444c: 0x0098		
0x444e: 0x12b0 call #0x44c0	5	; CALL
setClock()		
0x4450: 0x44c0		
0x4452: 0x40b2 mov.w #0x9680, &0x2400	5	; Store
frequency to memory		
0x4454: 0x9680		
0x4456: 0x2400		
0x4458: 0x40b2 mov.w #0x0098, &0x2402	5	; Store duration
to memory		
0x445a: 0x0098		
0x445c: 0x2402		
0x445e: 0xb3e2 bit.b #2, &0x0200	4	; Check if
switch is still pressed		
0x4460: 0x0200		
0x4462: 0x27fd jeq 0x445e (offset: -6)	2	; Wait until
switch is released		
0x4464: 0xb3e2 bit.b #2, &0x0201	4	; Test Switch 2
at P1.1		
0x4466: 0x0201		
0x4468: 0x2017 jne 0x4498 (offset: 46)	2	; Jump if not
pressed		
0x446a: 0x90b2 cmp.w #0x000f, &0x2402	5	; Compare
duration with lower limit(1 Mhz)		
0x446c: 0x000f		
0x446e: 0x2402		
0x4470: 0x2810 jlo 0x4492 (offset: 32)	2	; Jump to check
switches if lower		
0x4472: 0x2004 jne 0x447c (offset: 8)	2	; Jump if not
equal		
0x4474: 0x90b2 cmp.w #0x4241, &0x2400	5	; Compare
frequency with limit		
0x4476: 0x4241		
0x4478: 0x2400		
0x447a: 0x280b jlo 0x4492 (offset: 22)	2	; Jump if lower
0x447c: 0xc312 clrc -- bic.w #1, SR	1	; Clear carry
flag		
0x447e: 0x1012 rrc.w &0x2402	4	; Rotate stored
duration right		
0x4480: 0x2402		

0x4482: 0x1012 rrc.w &0x2400 frequency right	4	; Rotate stored
0x4484: 0x2400		
0x4486: 0x421c mov.w &0x2400, r12 frequency to R12	3	; Load modified
0x4488: 0x2400		
0x448a: 0x421d mov.w &0x2402, r13 duration to R12	3	; Load modified
0x448c: 0x2402		
0x448e: 0x12b0 call #0x44c0 setClock()	5	; CALL
0x4490: 0x44c0		
0x4492: 0xb3e2 bit.b #2, &0x0201 still pressed	4	; Test if SW2 is
0x4494: 0x0201		
0x4496: 0x27fd jeq 0x4492 (offset: -6) switch is released	2	; Wait until
0x4498: 0xe3d2 xor.b #1, &0x0202	4	; Toggle LED1
0x449a: 0x0202		
0x449c: 0xe0f2 xor.b #0x80, &0x0223	5	; Toggle LED2
0x449e: 0x0080		
0x44a0: 0x0223		
0x44a2: 0x4381 mov.w #0, 0(SP) stack pointer to 0	4	; Initalize
0x44a4: 0x0000		
0x44a6: 0x40b1 mov.w #0xc350, 0(SP) stack pointer	5	; Load 0xc350 to
0x44a8: 0xc350		
0x44aa: 0x0000		
0x44ac: 0x9381 cmp.w #0, 0(SP)	4	; Test SP
0x44ae: 0x0000		
0x44b0: 0x27c7 jeq 0x4440 (offset: -114) to 0	2	; Jump if equal
0x44b2: 0x8391 sub.w #1, 0(SP) stack pointer	4	; Decrement
0x44b4: 0x0000		
0x44b6: 0x9381 cmp.w #0, 0(SP)	4	; Test SP
0x44b8: 0x0000		
0x44ba: 0x23fb jne 0x44b2 (offset: -10) equal	2	; Jump if not
0x44bc: 0x3fc1 jmp 0x4440 (offset: -126) unconditionally	2	; Jump
0x44be: 0x4303 nop -- mov.w #0, CG	1	; no operation
0x44c0: 0x8221 sub.w #4, SP	1	; Move SP
0x44c2: 0x4c81 mov.w r12, 0(SP) stack	4	; Save r12 to
0x44c4: 0x0000		
0x44c6: 0x4d81 mov.w r13, 2(SP) stack	4	; Save r13 to
0x44c8: 0x0002		
0x44ca: 0x3c2a jmp 0x4520 (offset: 84)	2	; Jump to 0x4520
0x44cc: 0x40b2 mov.w #0x0050, &0x0162	5	; Store values
0x44ce: 0x0050		
0x44d0: 0x0162		
0x44d2: 0x40b2 mov.w #0x112c, &0x0164	5	
0x44d4: 0x112c		
0x44d6: 0x0164		

0x44d8: 0x3c4f jmp 0x4578 (offset: 158)	2	; Jump to 0x4578
0x44da: 0x40b2 mov.w #0x0040, &0x0162	5	; Store values
0x44dc: 0x0040		
0x44de: 0x0162		
0x44e0: 0x40b2 mov.w #0x1096, &0x0164	5	
0x44e2: 0x1096		
0x44e4: 0x0164		
0x44e6: 0x3c48 jmp 0x4578 (offset: 144)	2	; Jump to 0x4578
0x44e8: 0x40b2 mov.w #0x0030, &0x0162	5	; Store values
0x44ea: 0x0030		
0x44ec: 0x0162		
0x44ee: 0x40b2 mov.w #0x104b, &0x0164	5	
0x44f0: 0x104b		
0x44f2: 0x0164		
0x44f4: 0x3c41 jmp 0x4578 (offset: 130)	2	; Jump to 0x4578
0x44f6: 0x40b2 mov.w #0x0030, &0x0162	5	; Store values
0x44f8: 0x0030		
0x44fa: 0x0162		
0x44fc: 0x40b2 mov.w #0x103c, &0x0164	5	
0x44fe: 0x103c		
0x4500: 0x0164		
0x4502: 0x3c3a jmp 0x4578 (offset: 116)	2	; Jump to 0x4578
0x4504: 0x40b2 mov.w #0x0020, &0x0162	5	; Store values
0x4506: 0x0020		
0x4508: 0x0162		
0x450a: 0x40b2 mov.w #0x1025, &0x0164	5	
0x450c: 0x1025		
0x450e: 0x0164		
0x4510: 0x3c33 jmp 0x4578 (offset: 102)	2	; Jump to 0x4578
0x4512: 0x40b2 mov.w #0x0020, &0x0162	5	; Store values
0x4514: 0x0020		
0x4516: 0x0162		
0x4518: 0x40b2 mov.w #0x101e, &0x0164	5	
0x451a: 0x101e		
0x451c: 0x0164		
0x451e: 0x3c2c jmp 0x4578 (offset: 88)	2	; Jump to 0x4578
0x4520: 0x412f mov.w @SP, r15	2	; Load stack
pointer into register r15		
0x4522: 0x803f sub.w #0x4240, r15	2	; Subtract
immediate value		
0x4524: 0x4240		
0x4526: 0x703d subc.w #0x000f, r13	2	; Subtract with
carry from r13		
0x4528: 0x000f		
0x452a: 0x2002 jne 0x4530 (offset: 4)	2	; Jump if not
equal		
0x452c: 0x930f cmp.w #0, r15	1	; Test r15
0x452e: 0x27f1 jeq 0x4512 (offset: -30)	2	; Jump if equal
0x4530: 0x803f sub.w #0xd090, r15	2	; Subtract value
from r15		
0x4532: 0xd090		
0x4534: 0x703d subc.w #0x0003, r13	2	; Subtract value
from r13		
0x4536: 0x0003		
0x4538: 0x2002 jne 0x453e (offset: 4)	2	; Jump if not
equal		
0x453a: 0x930f cmp.w #0, r15	1	; Test again

0x453c: 0x27e3	jeq 0x4504 (offset: -58)	2	
0x453e: 0x803f	sub.w #0x71b0, r15	2	
0x4540: 0x71b0			
0x4542: 0x703d	subc.w #0x000b, r13	2	
0x4544: 0x000b			
0x4546: 0x2002	jne 0x454c (offset: 4)	2	
0x4548: 0x930f	cmp.w #0, r15	1	; Test again
0x454a: 0x27d5	jeq 0x44f6 (offset: -86)	2	
0x454c: 0x803f	sub.w #0xa120, r15	2	
0x454e: 0xa120			
0x4550: 0x703d	subc.w #0x0007, r13	2	
0x4552: 0x0007			
0x4554: 0x2002	jne 0x455a (offset: 4)	2	
0x4556: 0x930f	cmp.w #0, r15	1	; Test again
0x4558: 0x27c7	jeq 0x44e8 (offset: -114)	2	
0x455a: 0x803f	sub.w #0x25a0, r15	2	
0x455c: 0x25a0			
0x455e: 0x703d	subc.w #0x0026, r13	2	
0x4560: 0x0026			
0x4562: 0x2002	jne 0x4568 (offset: 4)	2	
0x4564: 0x930f	cmp.w #0, r15	1	; Test again
0x4566: 0x27b9	jeq 0x44da (offset: -142)	2	
0x4568: 0x803f	sub.w #0x4b40, r15	2	
0x456a: 0x4b40			
0x456c: 0x703d	subc.w #0x004c, r13	2	
0x456e: 0x004c			
0x4570: 0x2003	jne 0x4578 (offset: 6)	2	
0x4572: 0x930f	cmp.w #0, r15	1	; Test again
0x4574: 0x27ab	jeq 0x44cc (offset: -170)	2	
0x4576: 0x3c00	jmp 0x4578 (offset: 0)	2	; Jump
unconditionally			
0x4578: 0x5221	add.w #4, SP	1	; Allocate space
on stack			
0x457a: 0x4130	ret -- mov.w @SP+, PC	3	; Return from
subroutine			
0x457c: 0x120a	push.w r10	3	; Save registers
onto stack			
0x457e: 0x1209	push.w r9	3	
0x4580: 0x1208	push.w r8	3	
0x4582: 0x4c0a	mov.w r12, r10	1	; Copy r12 to
r10			
0x4584: 0x4a78	mov.b @r10+, r8	2	; Load byte into
r8 and autoincrement r10			
0x4586: 0x4309	mov.w #0, r9	1	; Clear r9
0x4588: 0x3c11	jmp 0x45ac (offset: 34)	2	; Jump to 0x45ac
0x458a: 0x4d0e	mov.w r13, r14	1	; Copy register
value			
0x458c: 0x8b0e	sub.w r11, r14	1	; Subtract r11
from r14			
0x458e: 0x831e	sub.w #1, r14	1	; Decrement r14
0x4590: 0x531d	add.w #1, r13	1	; Increment r13
0x4592: 0x4efd	mov.b @r14+, -1(r13)	5	; Store value
from r14 at address r13 - 1			
0x4594: 0xffff			
0x4596: 0x831f	sub.w #1, r15	1	; Subtract 1
from r15			
0x4598: 0x23fb	jne 0x4590 (offset: -10)	2	; Loop

0x459a: 0x3c03 jmp 0x45a2 (offset: 6)	2	; Loop
0x459c: 0x531d add.w #1, r13	1	; Increment r13
0x459e: 0x4afd mov.b @r10+, -1(r13)	5	; Repeat
0x45a0: 0xffff		
0x45a2: 0xc312 clrc -- bic.w #1, SR	1	; Clear carry
0x45a4: 0x1008 rrc.w r8	1	; Rotate r8 and carry bit
0x45a6: 0x5319 add.w #1, r9	1	; Increment r9
0x45a8: 0x9239 cmp.w #8, r9	1	; Compare r9 with 8
0x45aa: 0x37ec jge 0x4584 (offset: -40)	2	; Loop if greater than or equal
0x45ac: 0xb318 bit.w #1, r8	1	; Test r8
0x45ae: 0x23f6 jne 0x459c (offset: -20)	2	; Loop if not equal
0x45b0: 0x4a7b mov.b @r10+, r11	2	; Load value at r10 to r11 and autoincrement
0x45b2: 0x4a7f mov.b @r10+, r15	2	; Load value at r10 to r15 and autoincrement
0x45b4: 0x4b0c mov.w r11, r12	1	; Move r11 to r12
0x45b6: 0x12b0 call #0x46be	5	; CALL subroutine at 0x46be
0x45b8: 0x46be		
0x45ba: 0x4c0b mov.w r12, r11	1	; Copy result back to r11
0x45bc: 0x4f0c mov.w r15, r12	1	; Move r15 to r12
0x45be: 0x12b0 call #0x468a	5	; CALL subroutine at 0x468a
0x45c0: 0x468a		
0x45c2: 0xf03c and.w #0x000f, r12	2	; Set lower 4 bits
0x45c4: 0x000f		
0x45c6: 0xdc0b bis.w r12, r11	1	; Set r11 with OR result
0x45c8: 0xf03f and.w #0x000f, r15	2	; Set lower 4 bits
0x45ca: 0x000f		
0x45cc: 0x503f add.w #0x0003, r15	2	; Add 3 to r15
0x45ce: 0x0003		
0x45d0: 0x903f cmp.w #0x0012, r15	2	; Compare r15 with 0x12
0x45d2: 0x0012		
0x45d4: 0x200c jne 0x45ee (offset: 24)	2	; Jump if not equal to 0x12
0x45d6: 0x4a7e mov.b @r10+, r14	2	; Load value at r10 to r14 and autoincrement
0x45d8: 0xb03e bit.w #0x0080, r14	2	; Check BIT7
0x45da: 0x0080		
0x45dc: 0x2407 jeq 0x45ec (offset: 14)	2	; Jump if BIT7 == 0
0x45de: 0x4a7c mov.b @r10+, r12	2	; Load value at r10 to r12 and autoincrement
0x45e0: 0x4c4c mov.b r12, r12	1	
0x45e2: 0x12b0 call #0x46b8	5	; CALL subroutine at 0x46b8

0x45e4: 0x46b8		
0x45e6: 0xf03e and.w #0x007f, r14	2	; Set lower 7 bits of r14
0x45e8: 0x007f		
0x45ea: 0xdc0e bis.w r12, r14	1	; Bitwise OR and set r14
0x45ec: 0x5e0f add.w r14, r15	1	; Add r14 to r15
0x45ee: 0x903b cmp.w #0xffff, r11	2	; Compare r11 with 0xffff
0x45f0: 0xffff		
0x45f2: 0x23cb jne 0x458a (offset: -106)	2	; Loop if not equal
0x45f4: 0x4030 mov.w #0x4710, PC	3	; Jump to 0x4710
0x45f6: 0x4710		
0x45f8: 0x120a push.w r10	3	; Save registers onto stack
0x45fa: 0x1209 push.w r9	3	
0x45fc: 0x1208 push.w r8	3	
0x45fe: 0x4219 mov.w &0x015c, r9	3	; Move address 0x015c to r9
0x4600: 0x015c		
0x4602: 0x40b2 mov.w #0x5a80, &0x015c	5	; Store 0x5a80 at address 0x015c
0x4604: 0x5a80		
0x4606: 0x015c		
0x4608: 0x403f mov.w #0x4734, r15	2	; Move value 0x4734 to r15
0x460a: 0x4734		
0x460c: 0x903f cmp.w #0x4738, r15	2	; Compare with r15
0x460e: 0x4738		
0x4610: 0x2416 jeq 0x463e (offset: 44)	2	; Jump if equal
0x4612: 0x403f mov.w #0x4738, r15	2	; Move value into r15
0x4614: 0x4738		
0x4616: 0x903f cmp.w #0x473c, r15	2	; Compare r15 with 0x473c
0x4618: 0x473c		
0x461a: 0x2411 jeq 0x463e (offset: 34)	2	; Jump if equal
0x461c: 0x403a mov.w #0x473c, r10	2	; Move 0x473c to r10
0x461e: 0x473c		
0x4620: 0x803a sub.w #0x4738, r10	2	; Subtract 0x4738 from r10
0x4622: 0x4738		
0x4624: 0x110a rra.w r10	1	; Rotate r10 right
0x4626: 0x110a rra.w r10	1	; Rotate r10 right
0x4628: 0x4038 mov.w #0x4738, r8	2	; Move 0x4738 to r8
0x462a: 0x4738		
0x462c: 0x483c mov.w @r8+, r12	2	
0x462e: 0x4c7f mov.b @r12+, r15	2	
0x4630: 0x5f0f add.w r15, r15	1	; Double r15
0x4632: 0x4f1f mov.w 18228(r15), r15	3	
0x4634: 0x4734		

0x4636: 0x483d mov.w @r8+, r13	2	
0x4638: 0x128f call r15	4	
0x463a: 0x831a sub.w #1, r10	1	; Decrement r10
0x463c: 0x23f7 jne 0x462c (offset: -18)	2	; Jump if not equal
0x463e: 0xc279 bic.b #8, r9	1	; Bit clear r9
0x4640: 0xd039 bis.w #0x5a08, r9	2	; Set r9 to 0x5a08
0x4642: 0x5a08		
0x4644: 0x4982 mov.w r9, &0x015c	4	
0x4646: 0x015c		
0x4648: 0x12b0 call #0x4722	5	; CALL subroutine
0x464a: 0x4722		
0x464c: 0x4030 mov.w #0x4710, PC	3	
0x464e: 0x4710		
0x4650: 0xf03d and.w #0x000f, r13	2	; Bitwise AND r13
0x4652: 0x000f		
0x4654: 0xe03d xor.w #0x000f, r13	2	; Toggle r13
0x4656: 0x000f		
0x4658: 0x5d0d add.w r13, r13	1	; Double r13
0x465a: 0x5d0d add.w r13, r13	1	; Double r13
0x465c: 0x5d00 add.w r13, PC	2	; Add r13 to PC
0x465e: 0xc312 clrc -- bic.w #1, SR	1	; Clear carry
0x4660: 0x100c rrc.w r12	1	; Rotate r12 right and clear carry
0x4662: 0xc312 clrc -- bic.w #1, SR	1	; Repeat
0x4664: 0x100c rrc.w r12	1	
0x4666: 0xc312 clrc -- bic.w #1, SR	1	
0x4668: 0x100c rrc.w r12	1	
0x466a: 0xc312 clrc -- bic.w #1, SR	1	
0x466c: 0x100c rrc.w r12	1	
0x466e: 0xc312 clrc -- bic.w #1, SR	1	
0x4670: 0x100c rrc.w r12	1	
0x4672: 0xc312 clrc -- bic.w #1, SR	1	
0x4674: 0x100c rrc.w r12	1	
0x4676: 0xc312 clrc -- bic.w #1, SR	1	
0x4678: 0x100c rrc.w r12	1	
0x467a: 0xc312 clrc -- bic.w #1, SR	1	
0x467c: 0x100c rrc.w r12	1	
0x467e: 0xc312 clrc -- bic.w #1, SR	1	
0x4680: 0x100c rrc.w r12	1	
0x4682: 0xc312 clrc -- bic.w #1, SR	1	
0x4684: 0x100c rrc.w r12	1	
0x4686: 0xc312 clrc -- bic.w #1, SR	1	
0x4688: 0x100c rrc.w r12	1	
0x468a: 0xc312 clrc -- bic.w #1, SR	1	
0x468c: 0x100c rrc.w r12	1	
0x468e: 0xc312 clrc -- bic.w #1, SR	1	
0x4690: 0x100c rrc.w r12	1	
0x4692: 0xc312 clrc -- bic.w #1, SR	1	
0x4694: 0x100c rrc.w r12	1	
0x4696: 0xc312 clrc -- bic.w #1, SR	1	
0x4698: 0x100c rrc.w r12	1	
0x469a: 0x4130 ret -- mov.w @SP+, PC	3	; Return from subroutine

0x469c: 0xf03d and.w #0x000f, r13	2	; Bitwise AND
0x000f with r13		
0x469e: 0x000f		
0x46a0: 0xe03d xor.w #0x000f, r13	2	; Toggle
0x46a2: 0x000f		
0x46a4: 0x5d0d add.w r13, r13	1	; Add r13 to
r13		
0x46a6: 0x5d00 add.w r13, PC	2	; Add r13 to
PC		
0x46a8: 0x5c0c add.w r12, r12	1	; Add r12 to
r12		
0x46aa: 0x5c0c add.w r12, r12	1	
0x46ac: 0x5c0c add.w r12, r12	1	
0x46ae: 0x5c0c add.w r12, r12	1	
0x46b0: 0x5c0c add.w r12, r12	1	
0x46b2: 0x5c0c add.w r12, r12	1	
0x46b4: 0x5c0c add.w r12, r12	1	
0x46b6: 0x5c0c add.w r12, r12	1	
0x46b8: 0x5c0c add.w r12, r12	1	
0x46ba: 0x5c0c add.w r12, r12	1	
0x46bc: 0x5c0c add.w r12, r12	1	
0x46be: 0x5c0c add.w r12, r12	1	
0x46c0: 0x5c0c add.w r12, r12	1	
0x46c2: 0x5c0c add.w r12, r12	1	
0x46c4: 0x5c0c add.w r12, r12	1	
0x46c6: 0x4130 ret -- mov.w @SP+, PC	3	; Return from
subroutine		
0x46c8: 0x4031 mov.w #0x4400, SP	2	; Move
starting address to Stack pointer		
0x46ca: 0x4400		
0x46cc: 0x12b0 call #0x471e	5	; Call
subroutine at 0x471e		
0x46ce: 0x471e		
0x46d0: 0x930c cmp.w #0, r12	1	; Test r12
0x46d2: 0x2402 jeq 0x46d8 (offset: 4)	2	; Jump if
equal		
0x46d4: 0x12b0 call #0x45f8	5	
0x46d6: 0x45f8		
0x46d8: 0x430c mov.w #0, r12	1	; Clear r12
0x46da: 0x12b0 call #0x4400	5	
0x46dc: 0x4400		
0x46de: 0x431c mov.w #1, r12	1	; Move 1 to
r12		
0x46e0: 0x12b0 call #0x4718	5	
0x46e2: 0x4718		
0x46e4: 0x4c0f mov.w r12, r15	1	; Move
contents of r12 to r15		
0x46e6: 0x4d0c mov.w r13, r12	1	; Move
contents of r13 to r12		
0x46e8: 0x403d mov.w #0x0003, r13	2	; Move 0x03 to
r13		
0x46ea: 0x0003		
0x46ec: 0x5f0d add.w r15, r13	1	; Add r15 to
r13		
0x46ee: 0x4f1e mov.w 1(r15), r14	3	; Move r15
shifted by 1 to r14		
0x46f0: 0x0001		

0x46f2: 0x4030 mov.w #0x46f6, PC	3	; Jump to
0x46f6		
0x46f4: 0x46f6		
0x46f6: 0x930e cmp.w #0, r14	1	; Test r14
0x46f8: 0x2406 jeq 0x4706 (offset: 12)	2	; Jump if 0
0x46fa: 0x4c0f mov.w r12, r15	1	; Move
contents of r12 to r15		
0x46fc: 0x531f add.w #1, r15	1	; Increment
r15		
0x46fe: 0x4dff mov.b @r13+, -1(r15)	5	; Move r13 to
r15 with an offset of -1, autoincrement r13		
0x4700: 0xffff		
0x4702: 0x831e sub.w #1, r14	1	; Subtract 1
from r14		
0x4704: 0x23fb jne 0x46fc (offset: -10)	2	; Jump if not
equal		
0x4706: 0x4130 ret -- mov.w @SP+, PC	3	; Return,
Increment stack pointer		
0x4708: 0x4134 pop.w r4 -- mov.w @SP+, r4	2	; Restore
registers		
0x470a: 0x4135 pop.w r5 -- mov.w @SP+, r5	2	
0x470c: 0x4136 pop.w r6 -- mov.w @SP+, r6	2	
0x470e: 0x4137 pop.w r7 -- mov.w @SP+, r7	2	
0x4710: 0x4138 pop.w r8 -- mov.w @SP+, r8	2	
0x4712: 0x4139 pop.w r9 -- mov.w @SP+, r9	2	
0x4714: 0x413a pop.w r10 -- mov.w @SP+, r10	2	
0x4716: 0x4130 ret -- mov.w @SP+, PC	3	; Return from
subroutine		
0x4718: 0x4303 nop -- mov.w #0, CG	1	; no operation
delay		
0x471a: 0x3fff jmp 0x471a (offset: -2)	2	; Infinite
loop		
0x471c: 0x4303 nop -- mov.w #0, CG	1	; delay
0x471e: 0x431c mov.w #1, r12	1	; Load r12
with value 1		
0x4720: 0x4130 ret -- mov.w @SP+, PC	3	; return from
subroutine		
0x4722: 0x4130 ret -- mov.w @SP+, PC	3	; return from
subroutine		
0x4724: 0xd032 bis.w #0x0010, SR	2	; Set status
register to 0x10		
0x4726: 0x0010		
0x4728: 0x3ffd jmp 0x4724 (offset: -6)	2	; Loop
0x472a: 0x4303 nop -- mov.w #0, CG	1	; delay
0x472c: 0x0001 mova @PC, SP	?	; ??
0x472e: 0x0004 mova @PC, r4	?	
0x4730: 0x8480 sub.w r4, 0x4750	2	; Subtract
value at r4 from 0x4750		
0x4732: 0x001e		
0x4734: 0x457c mov.b @r5+, r12	2	; Load value
at r5 to r12 and autoincrement		
0x4736: 0x46e4 mov.b @r6, 18220(r4)	5	; Load value
at r6 to r4 with offset		
0x4738: 0x472c		
0x473a: 0x2400 jeq 0x473c (offset: 0)	2	; Jump if
equal		

--

Make sure to submit a single pdf file only