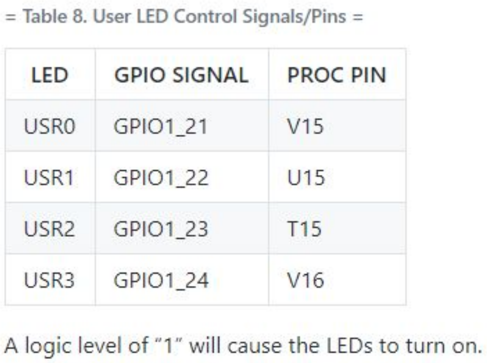
February 21, 2020

Part I:

Goal: learn how to control GPIO pins and how to turn the 4 Beaglebone Black USR LEDS on and off in a specified pattern with delay loop timing.

Task 1: The Manual

Go through the BeagleBone Black System Reference Manual. There is a section that shows which GPIO pins are needed and the logic levels to turn on the LEDs.



= Table 8. User LED Control Signals/Pins =

LED	GPIO SIGNAL	PROC PIN
USR0	GPIO1_21	V15
USR1	GPIO1_22	U15
USR2	GPIO1_23	T15
USR3	GPIO1_24	V16

A logic level of "1" will cause the LEDs to turn on.

Task 2: High Level Algorithm

```
Initialize all LEDs to OFF

REPEAT:
          For 1 second: Turn on LED3 and LED1. Turn off LED2 and LED0
          For 1 second: Turn off LED3 and LED1. Turn on LED2 and LED0
```

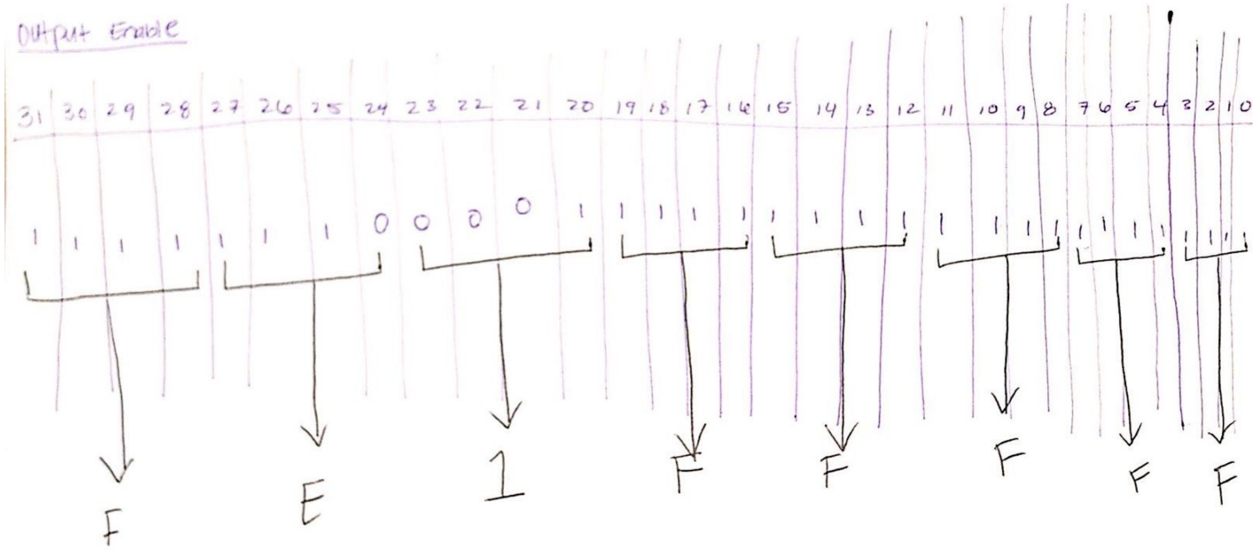
Task 3: GPIO Pins

```
GPIO1_21, GPIO1_22, GPIO1_23, GPIO1_24
```

Determine the registers that control the GPIO pins for the LEDS

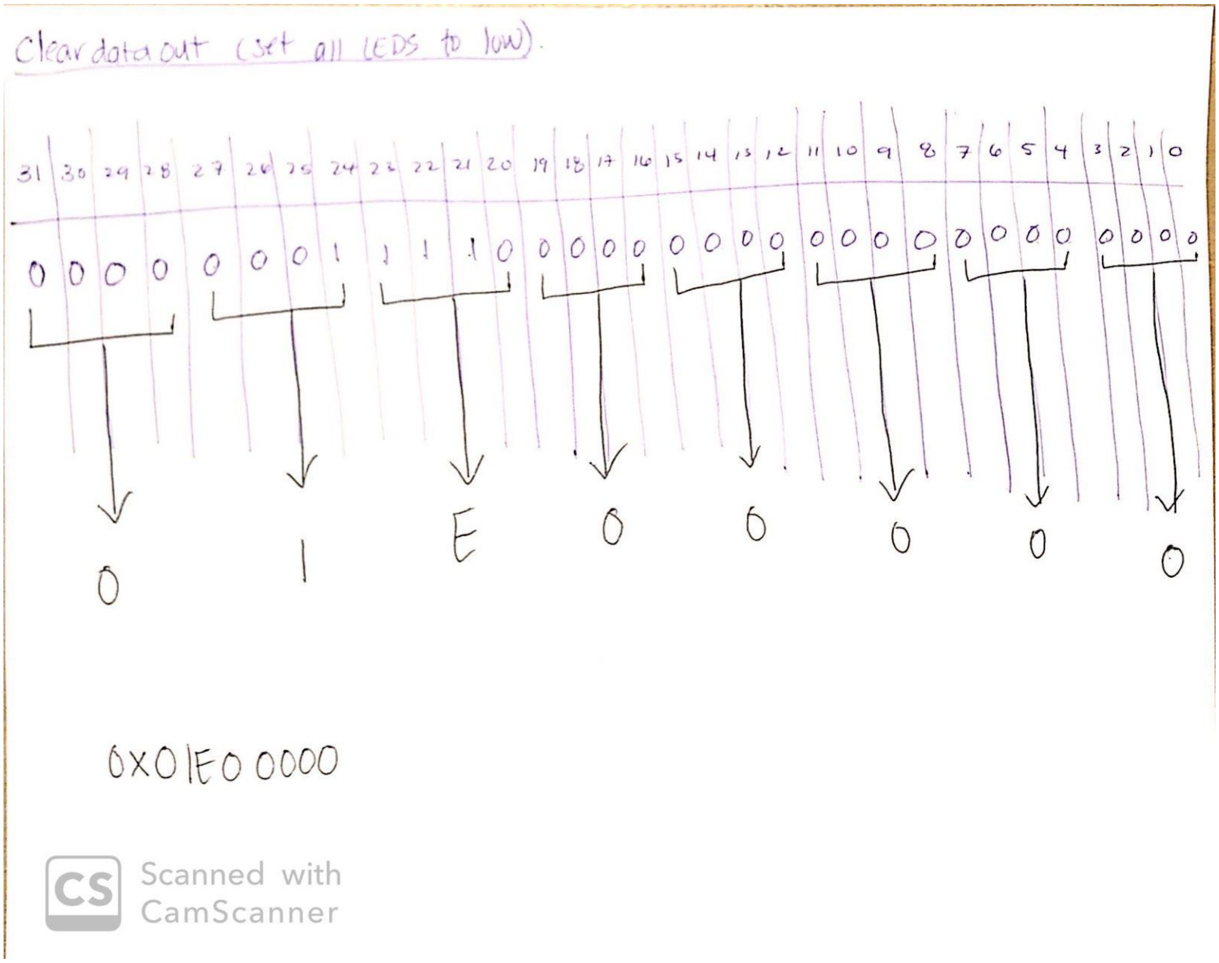
```
GPIO1 base address - [ 0x4804C000 + (offsets below) ]

Offsets for control registers
GPIO_SETDATAOUT 0x194
GPIO_CLEARDATAOUT 0x190
GPIO_OE 0x134
```

Task 4: Masks- OE



0XFE1F FFFF

Task 4: Masks- CLEARDATAOUT and SETDATAOUT Masks

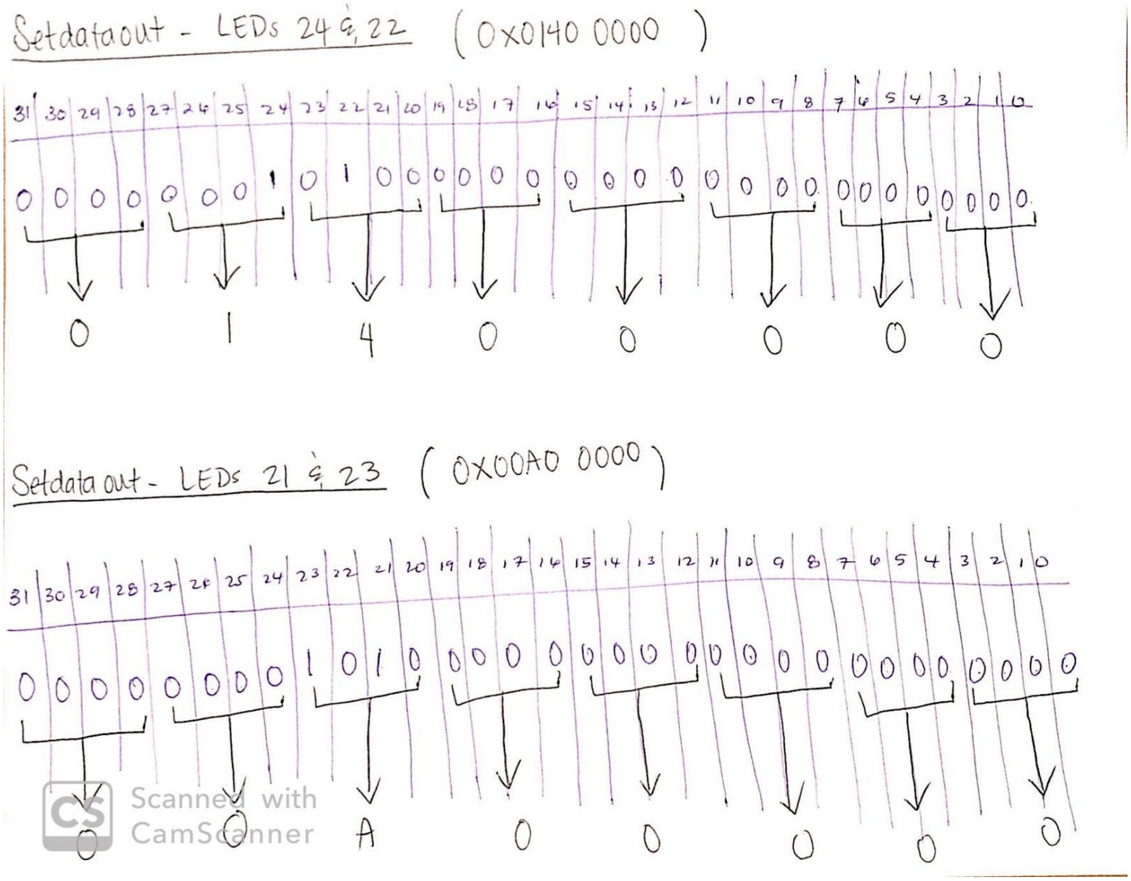


Clear data out (set all LEDS to low).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0 1 E 0 0 0 0 0

0X01E0 0000



Setdataout - LEDs 24 & 22 (0x0140 0000)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0 1 4 0 0 0 0 0

Setdata out - LEDs 21 & 23 (0x00A0 0000)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0 0 A 0 0 0 0 0

Task 5: Values and Addresses

GPIO1 base address - [0x4804C000 + (offsets)]

GPIO1 To set / clear with:

Pins 24 and 22: 0x0140 0000
Pins 23 and 21: 0x00A0 0000

To set as output mask: 0xFE1F FFFF

GPIO_SETDATAOUT 0x194
GPIO_CLEARDATAOUT 0x190
GPIO_OE 0x134

February 24, 2020

Task 6: RMW

GPIO1_21-24

READ- Configure GPIO1 using 0x4804C000 + 0x134 (OE Register Offset)
MODIFY- AND configured GPIO1 with pin21-24 mask
WRITE- Write the ANDED value back to OE Register

To initialize as low---> ADD (GPIO1 + CLEARDATAOUT offset) with 0x01E0 0000

February 25, 2020

Task 7: Low Level Algorithm

Note: base address for GPIO1 control registers and GPIO2 control registers are 0x4804C000 and 0x481AC000 respectively

Set GPIO1 bit 21 to low by writing 0x0020 0000 to GPIO1_CLEARDATAOUT at 0x4804C190 (**init LED21 as off**)

Set GPIO2 bits 22-24 to low by writing 0x01C0 0000 to GPIO2_CLEARDATAOUT at 0x481AC190 (**init LED22-24 as off**)

Set GPIO1 bit 21 to (**output)** by RMW 0xFFDF FFFF using OE at 0x4804C134 (+ 0x134)

Set GPIO2 bits 22-24 to (**outputs)** by RMW 0xFE3F FFFF using using OE at 0x481AC134 (+ 0x134)

REPEAT:

Wait 1 second with delay loop

- Load GPIO2_24 **(LED3**) and GPIO2_22 **(LED1)** using 0x01C0 0000 (**to GPIO2_SETDATAOUT)** and send.

- Write LED values to **(GPIO2_SETDATAOUT)** register at 0x481AC194. **(TURN ON)**

- Load GPIO2_23 **(LED2)**using 0x01C0 0000 **(to GPIO2_CLEARDATAOUT)** and load and GPIO1_21 **(LED0)** using 0x0020 0000 **(to GPIO1_CLEARDATAOUT).**

- Write LED Values to **(GPIO2_CLEARDATAOUT)** register at 0x481AC190 for GPIO2_23 **(LED2)** and **(GPIO1_CLEARDATAOUT)** at 0x4804C190 for GPIO1_21 **(LED0). (TURN OFF)**

Wait 1 second with delay loop

- Load GPIO2_24 **(LED3)** and GPIO2_22 **(LED1)**using 0x01C0 0000 **(to GPIO2_CLEARDATAOUT).**

- Write LED Values to (**GPIO2_CLEARDATAOUT)** register at 0x481AC190. **(TURN OFF)**

- Load GPIO2_23 **(LED2**) using 0x01C0 0000 (**to GPIO2_SETDATAOUT)**and GPIO1_21 **(LED0)** using 0x0020 0000 **(to GPIO1_SETDATAOUT**) and send.

- Write LED values to **(GPIO2_SETDATAOUT)** register at 0x481AC194 and **(GPIO1_SETDATAOUT)** register at 0x4804C194 . **(TURN ON)**

February 26, 2020

Modified High Level Algorithm (Suggested Changes by Professor)

Initialize LEDs to OFF (could be an optional step)

Set LED3 and LED1 (GPIO2_SETDATAOUT)

Set LED0, LED1, LED2, LED3 as outputs (GPIO1_OE and GPIO2_OE)

Wait 1 second

Clear LED3 and LED1 (GPIO2_CLEARDATAOUT)

Set LED2 and LED0 (GPIO2_SETDATAOUT and GPIO1_SETDATAOUT)

Wait 1 second

Clear LED2 and LED 0 (GPIO2_CLEARDATAOUT and GPIO1_CLEARDATAOUT)

Set LED3 and LED1 (GPIO2_SETDATAOUT)

Modified Low Level Algorithm

Note: base address for GPIO1 control registers and GPIO2 control registers are 0x4804C000 and 0x481AC000 respectively

Set GPIO2_24 **(LED3)** and GPIO2_22 **(LED1)** to high by writing 0x0100 0000 and 0x0040 0000 to GPIO2_SETDATAOUT at 0x4804C194 **(Turn on LED3 and LED1**)

Set GPIO1 bit 21 to (**output)** by RMW 0xFFDF FFFF using OE at 0x4804C134 (+ 0x134)

Set GPIO2 bits 22-24 to (**outputs)** by RMW 0xFE3F FFFF using using OE at 0x481AC134 (+ 0x134)

Wait 1 second

Clear GPIO2_24 **(LED3)** and GPIO2_22 **(LED1)** by writing 0x0100 0000 and 0x0040 0000 to the GPIO2_CLEARDATAOUT at 0x4804C190 **(Turn off LED3 and LED1)**

Set GPIO2_23 **(LED2)** and GPIO1_21 **(LED0)** to high by writing

0x0080 0000 to GPIO2_SETDATAOUT at 0x4804C194 and 0x0020 0000 to GPIO1_SETDATAOUT at 0x4804C194 **(Turn on LED2 and LED0)**

Wait 1 second

Clear GPIO2_23 **(LED2)** and GPIO1_21 **(LED0)** to low by writing 0x0080 0000 to GPIO2_CLEARDATAOUT at 0x4804C190 and 0x0020 0000 to GPIO1_CLEARDATAOUT at 0x4804C190 **(Turn off LED2 and LED0)**

Last Modification of Low Level Algorithm (March 3, 2020)

Note: base address for GPIO1 control registers is 0x4804C000

Set all of the pins to low by writing 0x 01E0 0000 to GPIO1_CLEARDATAOUT at 0x4804C190 **(TURN OFF ALL LEDS)**

Set GPIO1_24 **(LED3)** and GPIO1_22 **(LED1)** to high by writing 0X0140 0000 to GPIO1_SETDATAOUT at 0x4804C194 **(Turn on LED3 and LED1**)

Set GPIO1 bits 21-24 to (**output)** by RMW 0xFE1F FFFF using OE at 0x4804C134 (+ 0x134)
Wait 1 second

Clear GPIO1_24 **(LED3)** and GPIO1_22 **(LED1)** by writing 0X0140 0000 to the GPIO1_CLEARDATAOUT at 0x4804C190 **(Turn off LED3 and LED1)**

Set GPIO1_23 **(LED2)** and GPIO1_21 **(LED0)** to high by writing 0X00A0 0000 to GPIO1_SETDATAOUT at 0x4804C194 **(Turn on LED2 and LED0)**

Wait 1 second

Clear GPIO1_23 **(LED2)** and GPIO1_21 **(LED0)** to low by writing 0X00A0 0000 to GPIO1_CLEARDATAOUT at 0x4804C190 **(Turn off LED2 and LED0)**

March 3, 2020

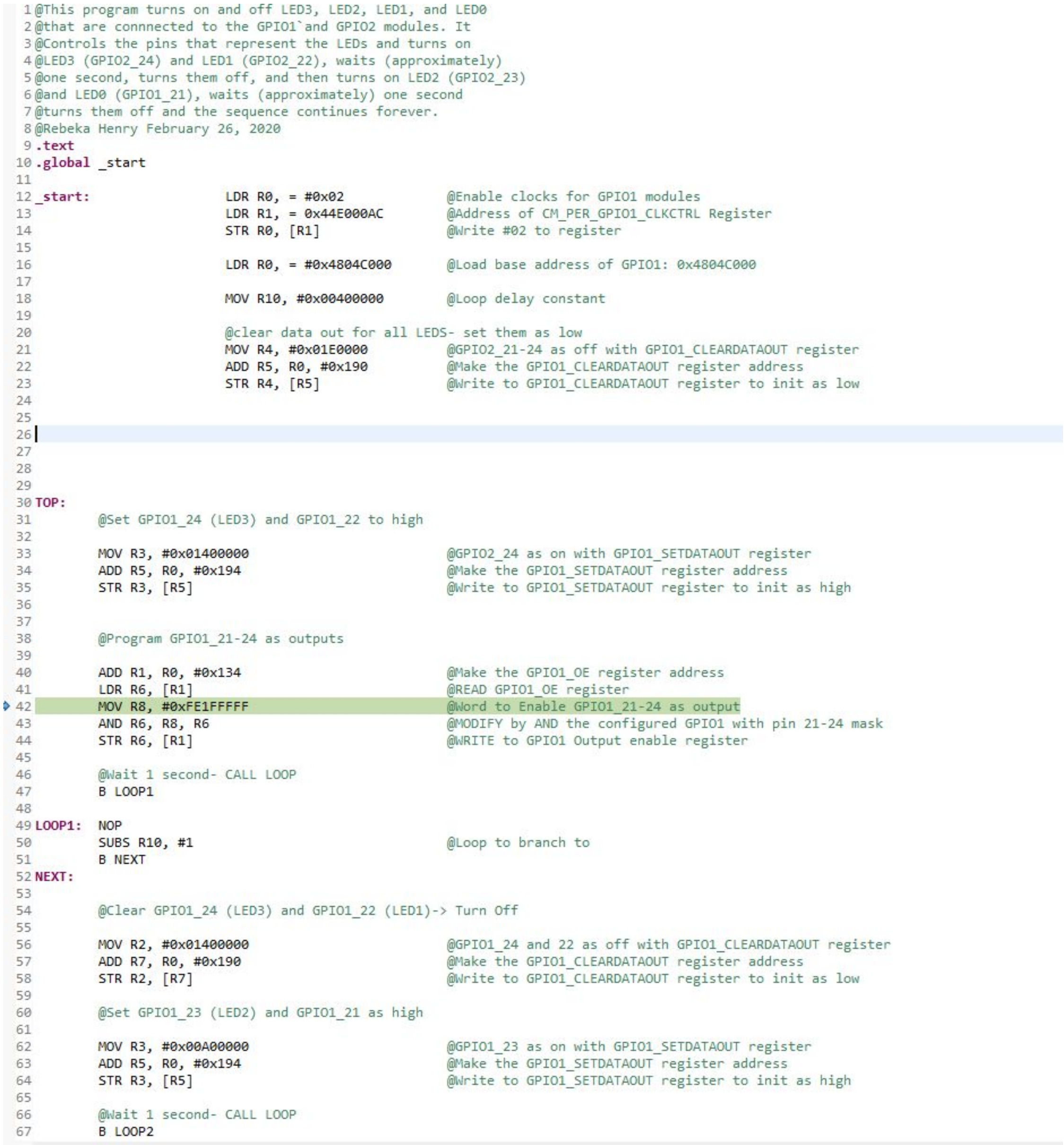
Task 8: Assembly Language Program

https://github.com/beagleboard/beaglebone-black/wiki/System-Reference-Manual

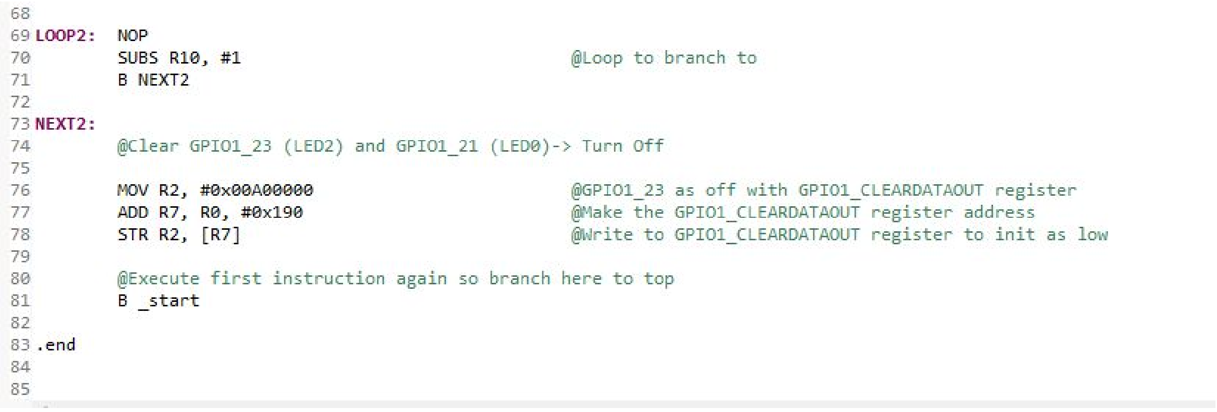
I was using the incorrect manual version and so I had to change the addresses and GPIO pins

I only need GPIO1 and do not need GPIO2. The changes to the code are below:

I decided to rewrite to set and clear data out at once for each set of LED pins so the addresses are going to be different to control the pins.



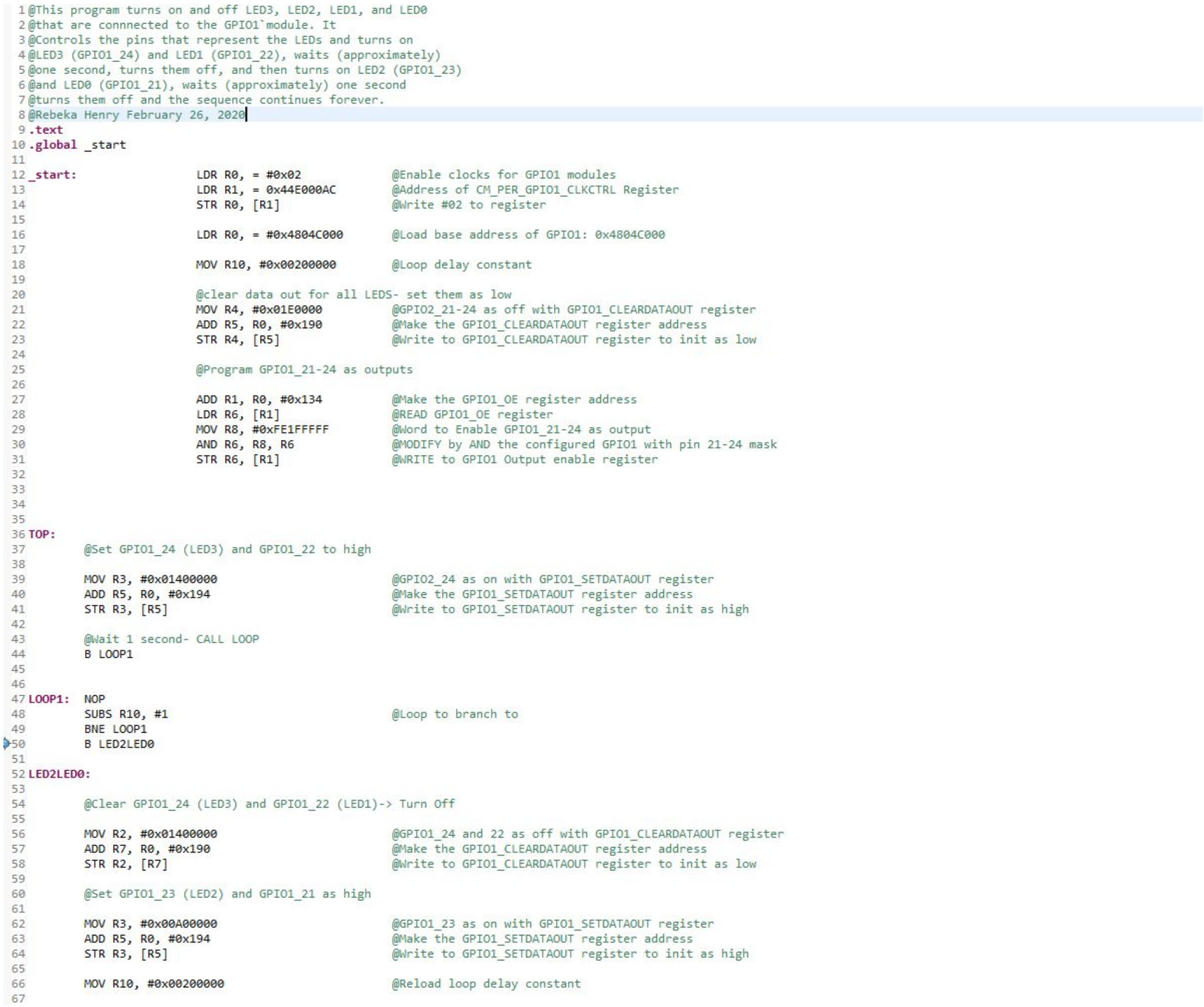
```
@This program turns on and off LED3, LED2, LED1, and LED0
@that are connnected to the GPIO1`and GPIO2 modules. It
@Controls the pins that represent the LEDs and turns on
@LED3 (GPIO2_24) and LED1 (GPIO2_22), waits (approximately)
@one second, turns them off, and then turns on LED2 (GPIO2_23)
@and LED0 (GPIO1_21), waits (approximately) one second
@turns them off and the sequence continues forever.
@Rebeka Henry February 26, 2020
.text
.global _start

_start:                 LDR R0, = #0x02             @Enable clocks for GPIO1 modules
                        LDR R1, = 0x44E000AC        @Address of CM_PER_GPIO1_CLKCTRL Register
                        STR R0, [R1]                @Write #02 to register

                        LDR R0, = #0x4804C000       @Load base address of GPIO1: 0x4804C000

                        MOV R10, #0x00400000        @Loop delay constant

                        @clear data out for all LEDS- set them as low
                        MOV R4, #0x01E0000          @GPIO2_21-24 as off with GPIO1_CLEARDATAOUT register
                        ADD R5, R0, #0x190          @Make the GPIO1_CLEARDATAOUT register address
                        STR R4, [R5]                @Write to GPIO1_CLEARDATAOUT register to init as low






TOP:
        @Set GPIO1_24 (LED3) and GPIO1_22 to high

        MOV R3, #0x01400000                         @GPIO2_24 as on with GPIO1_SETDATAOUT register
        ADD R5, R0, #0x194                          @Make the GPIO1_SETDATAOUT register address
        STR R3, [R5]                                @Write to GPIO1_SETDATAOUT register to init as high


        @Program GPIO1_21-24 as outputs

        ADD R1, R0, #0x134                          @Make the GPIO1_OE register address
        LDR R6, [R1]                                @READ GPIO1_OE register
        MOV R8, #0xFE1FFFFF                         @Word to Enable GPIO1_21-24 as output
        AND R6, R8, R6                              @MODIFY by AND the configured GPIO1 with pin 21-24 mask
        STR R6, [R1]                                @WRITE to GPIO1 Output enable register

        @Wait 1 second- CALL LOOP
        B LOOP1

LOOP1:  NOP
        SUBS R10, #1                                @Loop to branch to
        B NEXT
NEXT:

        @Clear GPIO1_24 (LED3) and GPIO1_22 (LED1)-> Turn Off

        MOV R2, #0x01400000                         @GPIO1_24 and 22 as off with GPIO1_CLEARDATAOUT register
        ADD R7, R0, #0x190                          @Make the GPIO1_CLEARDATAOUT register address
        STR R2, [R7]                                @Write to GPIO1_CLEARDATAOUT register to init as low

        @Set GPIO1_23 (LED2) and GPIO1_21 as high

        MOV R3, #0x00A00000                         @GPIO1_23 as on with GPIO1_SETDATAOUT register
        ADD R5, R0, #0x194                          @Make the GPIO1_SETDATAOUT register address
        STR R3, [R5]                                @Write to GPIO1_SETDATAOUT register to init as high

        @Wait 1 second- CALL LOOP
        B LOOP2
```



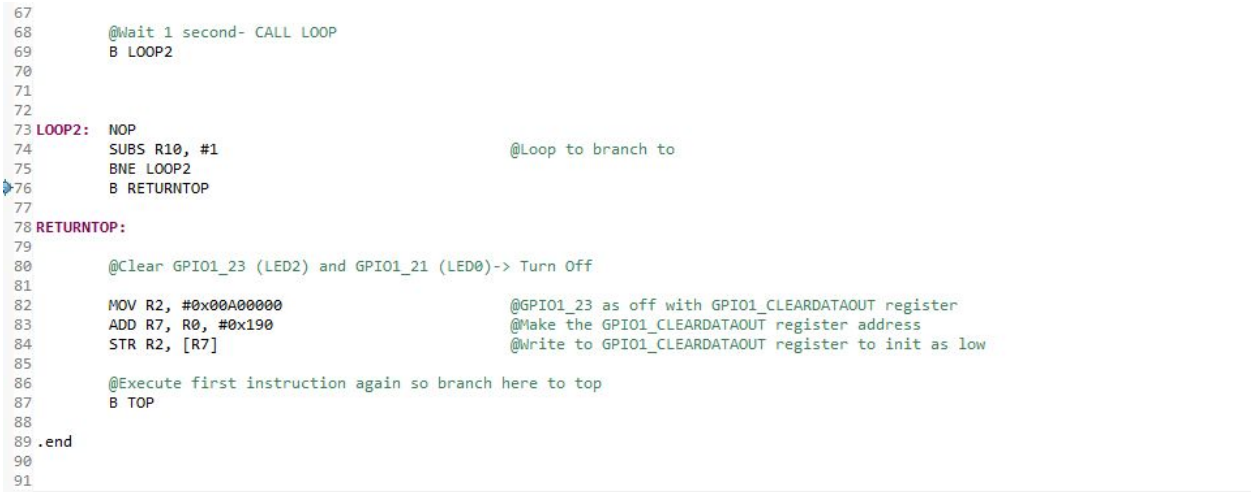
```

LOOP2:  NOP
        SUBS R10, #1                                @Loop to branch to
        B NEXT2

NEXT2:
        @Clear GPIO1_23 (LED2) and GPIO1_21 (LED0)-> Turn Off

        MOV R2, #0x00A00000                         @GPIO1_23 as off with GPIO1_CLEARDATAOUT register
        ADD R7, R0, #0x190                          @Make the GPIO1_CLEARDATAOUT register address
        STR R2, [R7]                                @Write to GPIO1_CLEARDATAOUT register to init as low

        @Execute first instruction again so branch here to top
        B _start

.end


```

The code successfully works as expected. The next step is to decrease the loop delay constant and to make sure that the end code goes to the top instead of the start of the program. I may also want to correctly word the labels that I am jumping to instead of using NEXT and NEXT2 or TOP. Another thing to do is to make sure I am BNE instead of B NEXT. Install breakpoints instead of Branching to fix the code. And finally, change the description at the top of the code.

I encountered an error with the loop delay constant (the second loop was not running properly and therefore not turning off the second sequence of LEDs) and was given the suggestion to reload it again so that the same delay constant is there. Here is the updated code that works:



```
@This program turns on and off LED3, LED2, LED1, and LED0
@that are connnected to the GPIO1`module. It
@Controls the pins that represent the LEDs and turns on
@LED3 (GPIO1_24) and LED1 (GPIO1_22), waits (approximately)
@one second, turns them off, and then turns on LED2 (GPIO1_23)
@and LED0 (GPIO1_21), waits (approximately) one second
@turns them off and the sequence continues forever.
@Rebeka Henry February 26, 2020
.text
.global _start

_start:                 LDR R0, = #0x02             @Enable clocks for GPIO1 modules
                        LDR R1, = 0x44E000AC        @Address of CM_PER_GPIO1_CLKCTRL Register
                        STR R0, [R1]                @Write #02 to register

                        LDR R0, = #0x4804C000       @Load base address of GPIO1: 0x4804C000

                        MOV R10, #0x00200000        @Loop delay constant

                        @clear data out for all LEDS- set them as low
                        MOV R4, #0x01E0000          @GPIO2_21-24 as off with GPIO1_CLEARDATAOUT register
                        ADD R5, R0, #0x190          @Make the GPIO1_CLEARDATAOUT register address
                        STR R4, [R5]                @Write to GPIO1_CLEARDATAOUT register to init as low

                        @Program GPIO1_21-24 as outputs

                        ADD R1, R0, #0x134          @Make the GPIO1_OE register address
                        LDR R6, [R1]                @READ GPIO1_OE register
                        MOV R8, #0xFE1FFFFF         @Word to Enable GPIO1_21-24 as output
                        AND R6, R8, R6              @MODIFY by AND the configured GPIO1 with pin 21-24 mask
                        STR R6, [R1]                @WRITE to GPIO1 Output enable register




TOP:
        @Set GPIO1_24 (LED3) and GPIO1_22 to high

        MOV R3, #0x01400000                         @GPIO2_24 as on with GPIO1_SETDATAOUT register
        ADD R5, R0, #0x194                          @Make the GPIO1_SETDATAOUT register address
        STR R3, [R5]                                @Write to GPIO1_SETDATAOUT register to init as high

        @Wait 1 second- CALL LOOP
        B LOOP1


LOOP1:  NOP
        SUBS R10, #1                                @Loop to branch to
        BNE LOOP1
        B LED2LED0

LED2LED0:

        @Clear GPIO1_24 (LED3) and GPIO1_22 (LED1)-> Turn Off

        MOV R2, #0x01400000                         @GPIO1_24 and 22 as off with GPIO1_CLEARDATAOUT register
        ADD R7, R0, #0x190                          @Make the GPIO1_CLEARDATAOUT register address
        STR R2, [R7]                                @Write to GPIO1_CLEARDATAOUT register to init as low

        @Set GPIO1_23 (LED2) and GPIO1_21 as high

        MOV R3, #0x00A00000                         @GPIO1_23 as on with GPIO1_SETDATAOUT register
        ADD R5, R0, #0x194                          @Make the GPIO1_SETDATAOUT register address
        STR R3, [R5]                                @Write to GPIO1_SETDATAOUT register to init as high

        MOV R10, #0x00200000                        @Reload loop delay constant

```



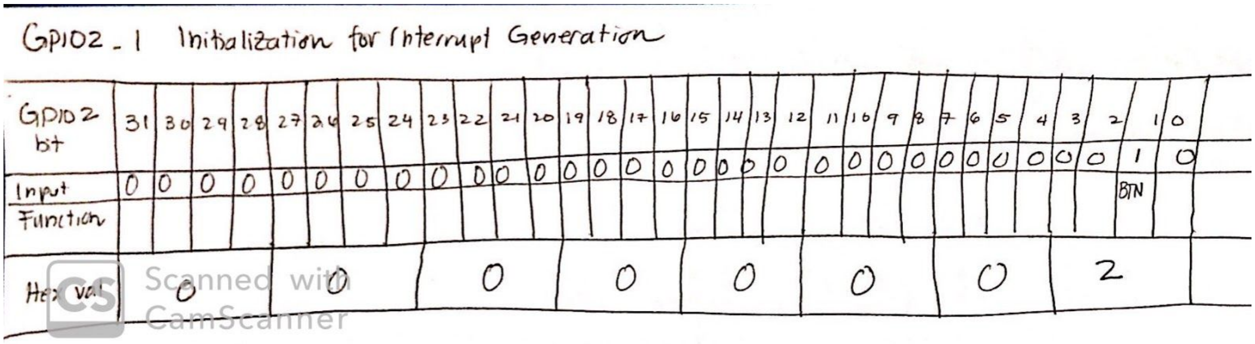
```

        @Wait 1 second- CALL LOOP
        B LOOP2



LOOP2:  NOP
        SUBS R10, #1                                @Loop to branch to
        BNE LOOP2
        B RETURNTOP

RETURNTOP:

        @Clear GPIO1_23 (LED2) and GPIO1_21 (LED0)-> Turn Off

        MOV R2, #0x00A00000                         @GPIO1_23 as off with GPIO1_CLEARDATAOUT register
        ADD R7, R0, #0x190                          @Make the GPIO1_CLEARDATAOUT register address
        STR R2, [R7]                                @Write to GPIO1_CLEARDATAOUT register to init as low

        @Execute first instruction again so branch here to top
        B TOP

.end


```

March 5, 2020

PART II:

Task 1:

GPIO2_1 Initialization for Interrupt Generation



GPIO2_1 Initialization for Interrupt Generation

GPIO2 bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Input Function	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1 BTN	0
Hex val	0				0				0				0				0				0				0				2			

```
GPIO2 base address: 0x481A C000
GPIO2_FALLINGDETECT: 0x14C
GPIO2_1: (to detect falling edge) 0x0000 0002
GPIO2_IRQSTATUS_SET_0 (POINTRPEND1: 0x34)
```

Note: OE for pins 21-24 is separate because these pins are on GPIO1 module. RMW is already done in Part I of the project.

INTC Initialization

Unmask the interrupt coming from GPIO2 so that INTC can generate an IRQ.

Since using GPIOINT2A (for GPIO2 Module and POINTRPEND1- INT: 32)

Unmask the an interrupt request using the INTC_MIR register

INTC base register address: 0x4820 0000

<u>MIR Registers</u>

MIR0 → 0-31
MIR1 → 32-63
MIR2 → 64-95
MIR3 → 96-127

It looks like INT 32 is going to fit in MIR1 register at bit 0. Can do this by writing 0x0000 0001 to INTC_MIR_SET1 register at address 0x4820 0000 + offset (0xAC)

To Mask I can just write 0x0000 0001 to INTC_MIR_CLEAR1 register at 0x4820 0000 + offset (0xA8)

The highlighted method will not be used

Hooking and Chaining

Since using the IRQ Exception, use the PC = [0x4030 CE38]. Use the SYS_IRQ register.

Modify Startup_ARMCA8.s File

See page 234. What is notable here is that .extern INT_DIRECTOR is at the top of the page and then the INT_DIRECTOR is somewhere in between ldr pc, [pc, #-8] **(0x14 Not used)** and ldr pc, [pc, #-8] **(0x1C FIQ Interrupt).** Another important point is that the INT_DIRECTOR is going to be at .global at the top of the assembly program

Enable Processor IRQ Input

As mentioned in the text, “reading the CPSR into R3 with special MRS R3, CPSR instruction, clearing bit 7, the IRQ bit, in the word read in to enable IRQ, and then writing the resultant word back to the CPRSR with MSR CPSR_c, R3 instruction”

INT_DIRECTOR (Modified High and Low Algorithms)

INTC_PENDING_IRQ1 Register (offset = B8h) [reset = 0h]

High Level:

```
Check INT_PENDING_IRQ1 register bit 0 to see IRQ from GPIOINT2A
IF no, then restore registers and return
ELSE,
     Check GPIO2_IRQSTATUS_0 register bit 1 to see if Button pressed
     If No, then restore registers and return to Wait Loop.
     Else go to Button Service
```

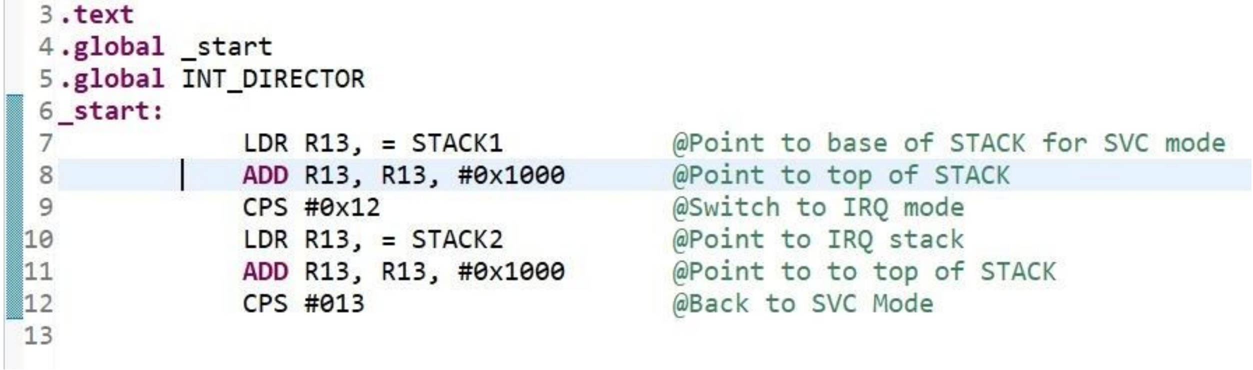
Low Level:

```
Restore registers

Read INT_PENDING_IRQ1 register at 0x4820 00B8 (INTC base + offset 0xB8)

Test bit 0, to see if GPIO2 POINTRPEND1 from GPIO2A
If bit 0 = 0, restore registers and return from interrupt
Else read GPIO2_IRQSTATUS_0 register at  0x481A C02C (GPIO2 base + offset 0x2C)

Test bit 1 with 0x0000 0002 to see if GPIO2_1 button pressed
If bit 1 = 0 restore registers and return from interrupt
Else go to BUTTON_SVC
```

Task 2: Modifications to Figure 5-14

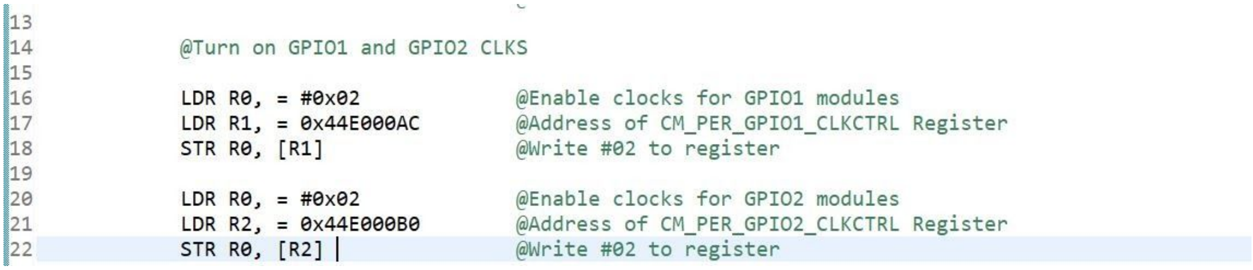
Visualization provided below as to how this program is modified to fit the Part 2 but to also fit the example.

What can be kept is the 2 stacks for now and the switch to IRQ mode and then back to SVC mode



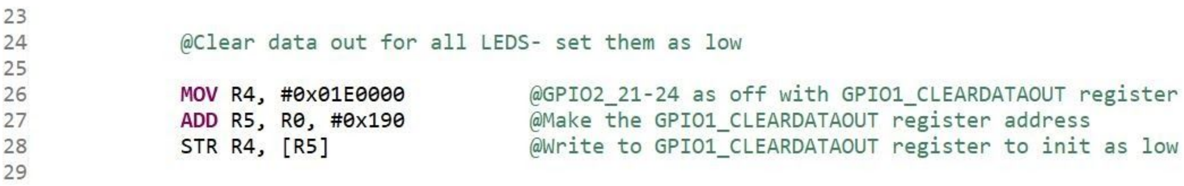
```
.text
.global _start
.global INT_DIRECTOR
_start:
        LDR R13, = STACK1           @Point to base of STACK for SVC mode
        ADD R13, R13, #0x1000       @Point to top of STACK
        CPS #0x12                   @Switch to IRQ mode
        LDR R13, = STACK2           @Point to IRQ stack
        ADD R13, R13, #0x1000       @Point to to top of STACK
        CPS #013                    @Back to SVC Mode
```

In this program, since we are using two GPIO modules, both have to be initialized at the start of the program. GPIO1 module is for the LEDS, GPIO2 module is for the button



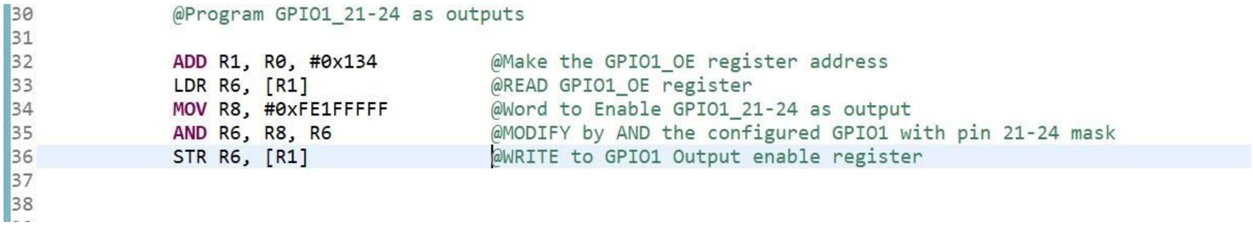
```
        @Turn on GPIO1 and GPIO2 CLKS

        LDR R0, = #0x02             @Enable clocks for GPIO1 modules
        LDR R1, = 0x44E000AC        @Address of CM_PER_GPIO1_CLKCTRL Register
        STR R0, [R1]                @Write #02 to register

        LDR R0, = #0x02             @Enable clocks for GPIO2 modules
        LDR R2, = 0x44E000B0        @Address of CM_PER_GPIO2_CLKCTRL Register
        STR R0, [R2]                @Write #02 to register
```

Making sure the LEDS are all off is the next step. This is a change since we are using GPIO1_21-24



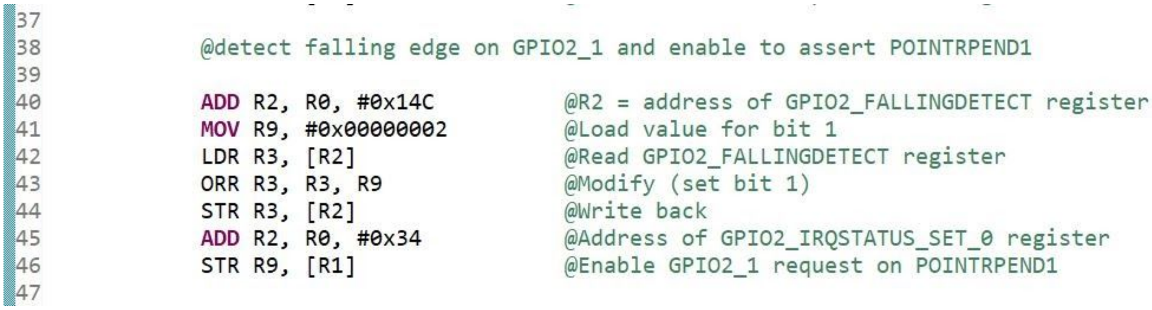
```
        @Clear data out for all LEDS- set them as low

        MOV R4, #0x01E0000          @GPIO2_21-24 as off with GPIO1_CLEARDATAOUT register
        ADD R5, R0, #0x190          @Make the GPIO1_CLEARDATAOUT register address
        STR R4, [R5]                @Write to GPIO1_CLEARDATAOUT register to init as low
```

The next change is for all of the LEDS to be enabled as outputs as shown in figure 5-14



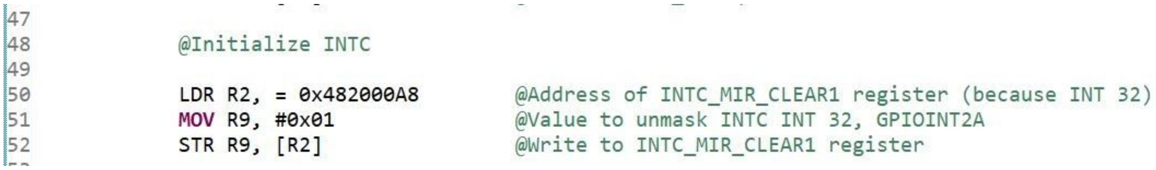
```
        @Program GPIO1_21-24 as outputs

        ADD R1, R0, #0x134          @Make the GPIO1_OE register address
        LDR R6, [R1]                @READ GPIO1_OE register
        MOV R8, #0xFE1FFFFF         @Word to Enable GPIO1_21-24 as output
        AND R6, R8, R6              @MODIFY by AND the configured GPIO1 with pin 21-24 mask
        STR R6, [R1]                @WRITE to GPIO1 Output enable register
```

Falling Edge detection modifications (since using GPIO2 instead of GPIO1)



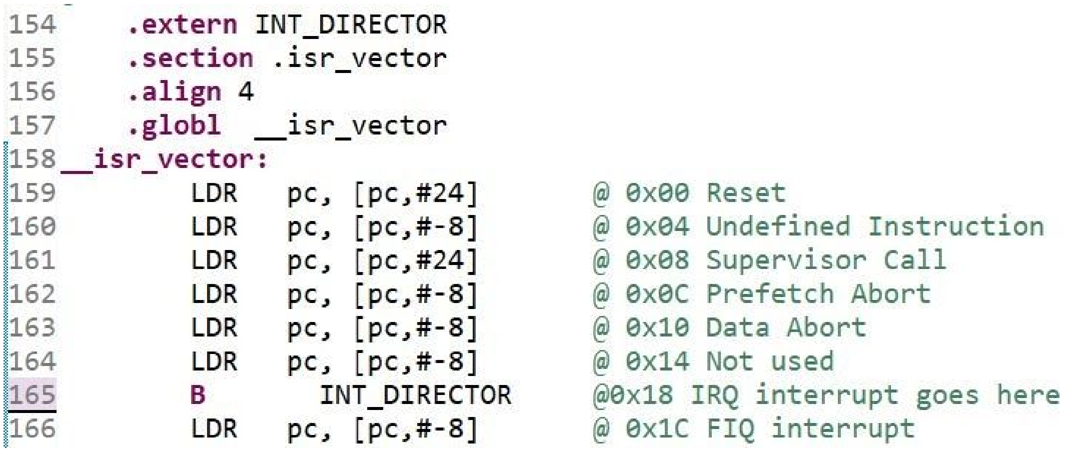
```
        @detect falling edge on GPIO2_1 and enable to assert POINTRPEND1

        ADD R2, R0, #0x14C          @R2 = address of GPIO2_FALLINGDETECT register
        MOV R9, #0x00000002         @Load value for bit 1
        LDR R3, [R2]                @Read GPIO2_FALLINGDETECT register
        ORR R3, R3, R9              @Modify (set bit 1)
        STR R3, [R2]                @Write back
        ADD R2, R0, #0x34           @Address of GPIO2_IRQSTATUS_SET_0 register
        STR R9, [R1]                @Enable GPIO2_1 request on POINTRPEND1
```

Initialize INTC Modification



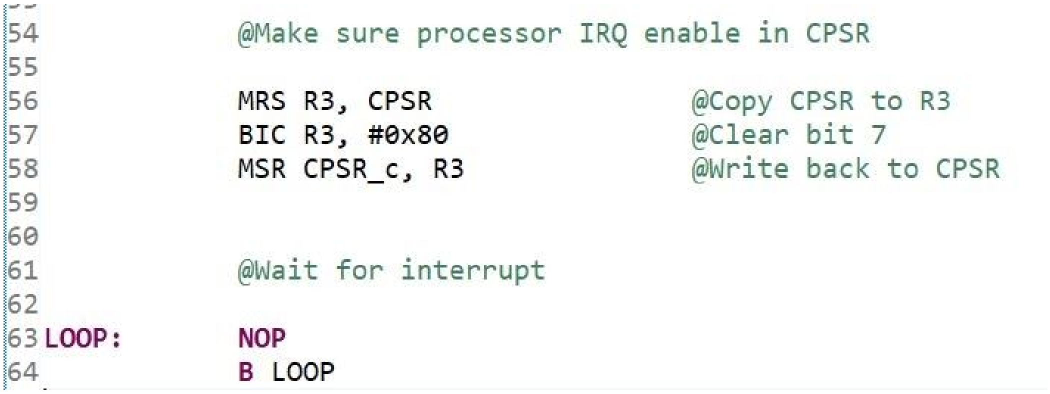
```
        @Initialize INTC

        LDR R2, = 0x482000A8        @Address of INTC_MIR_CLEAR1 register (because INT 32)
        MOV R9, #0x01               @Value to unmask INTC INT 32, GPIOINT2A
        STR R9, [R2]                @Write to INTC_MIR_CLEAR1 register
```

Instead of hooking and chaining, make changes to the `startup_ARMCA8.s` file



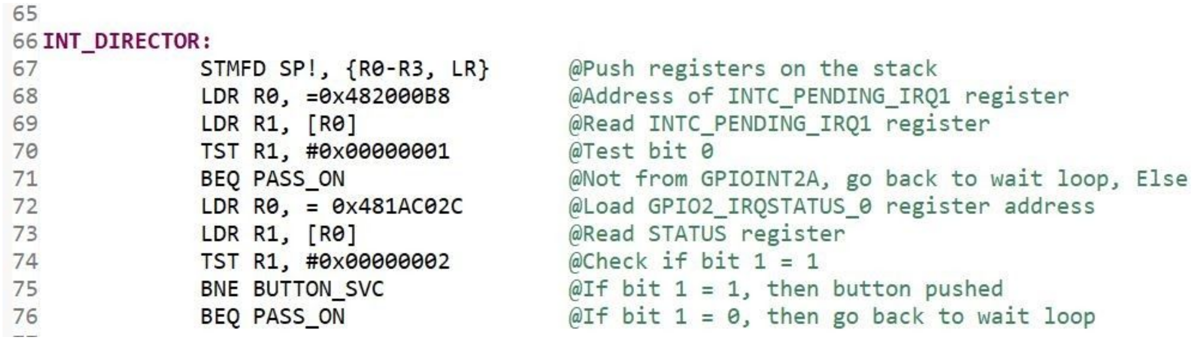
```
   .extern INT_DIRECTOR
   .section .isr_vector
   .align 4
   .globl  __isr_vector
__isr_vector:
        LDR   pc, [pc,#24]       @ 0x00 Reset
        LDR   pc, [pc,#-8]       @ 0x04 Undefined Instruction
        LDR   pc, [pc,#24]       @ 0x08 Supervisor Call
        LDR   pc, [pc,#-8]       @ 0x0C Prefetch Abort
        LDR   pc, [pc,#-8]       @ 0x10 Data Abort
        LDR   pc, [pc,#-8]       @ 0x14 Not used
        B       INT_DIRECTOR     @0x18 IRQ interrupt goes here
        LDR   pc, [pc,#-8]       @ 0x1C FIQ interrupt
```

No changes to enable IRQ in CPSR or loop



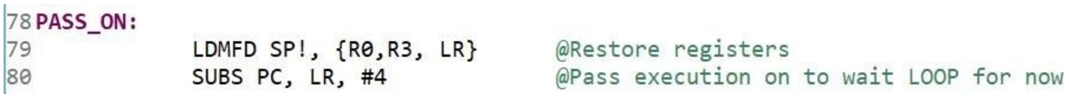
```
            @Make sure processor IRQ enable in CPSR

            MRS R3, CPSR                @Copy CPSR to R3
            BIC R3, #0x80               @Clear bit 7
            MSR CPSR_c, R3              @Write back to CPSR


            @Wait for interrupt

LOOP:       NOP
            B LOOP
```

Modifications to INT_DIRECTOR



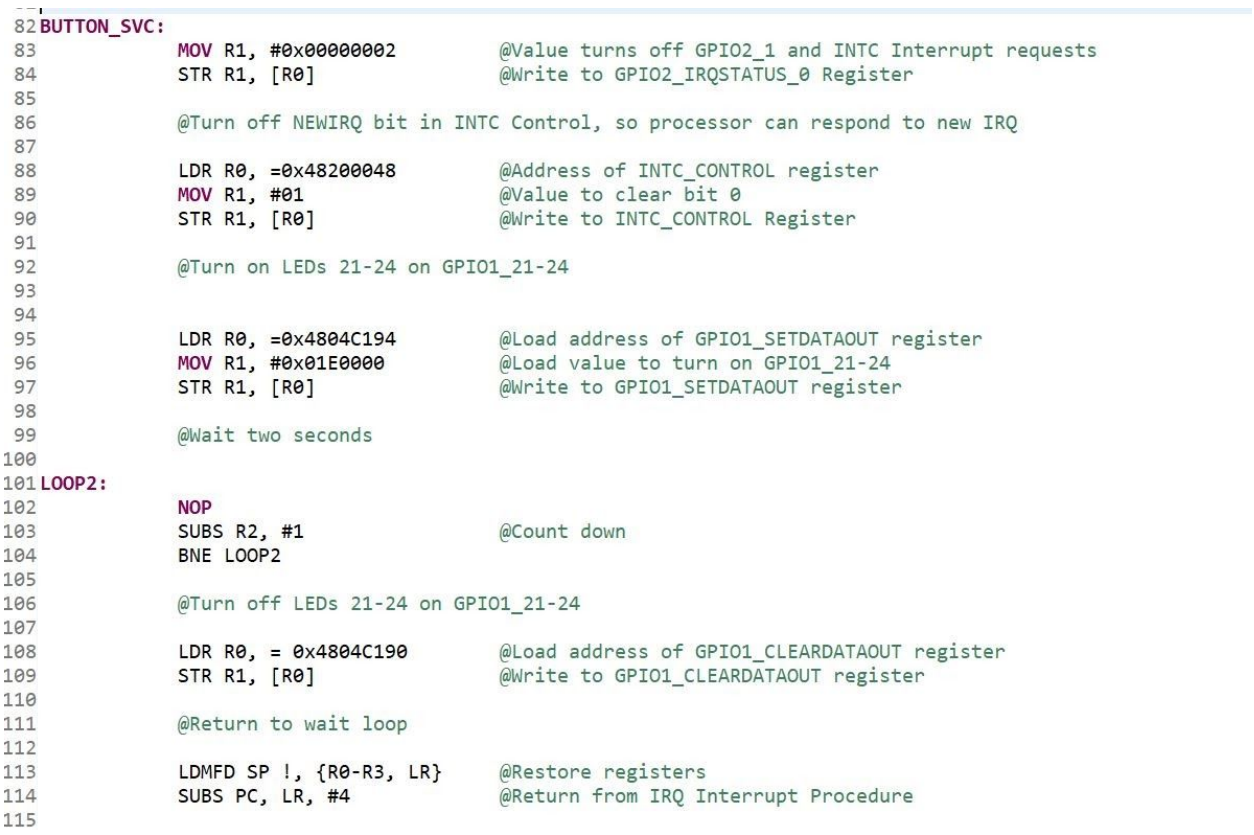
```
INT_DIRECTOR:
            STMFD SP!, {R0-R3, LR}      @Push registers on the stack
            LDR R0, =0x482000B8         @Address of INTC_PENDING_IRQ1 register
            LDR R1, [R0]                @Read INTC_PENDING_IRQ1 register
            TST R1, #0x00000001         @Test bit 0
            BEQ PASS_ON                 @Not from GPIOINT2A, go back to wait loop, Else
            LDR R0, = 0x481AC02C        @Load GPIO2_IRQSTATUS_0 register address
            LDR R1, [R0]                @Read STATUS register
            TST R1, #0x00000002         @Check if bit 1 = 1
            BNE BUTTON_SVC              @If bit 1 = 1, then button pushed
            BEQ PASS_ON                 @If bit 1 = 0, then go back to wait loop
```

No changes to PASS_ON



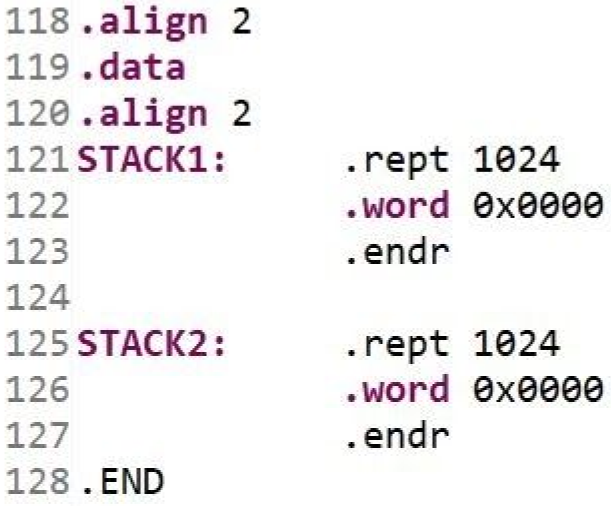
```
PASS_ON:
            LDMFD SP!, {R0,R3, LR}      @Restore registers
            SUBS PC, LR, #4             @Pass execution on to wait LOOP for now
```

Changes to BUTTON_SVC and LEDs Turn on and off. **Note: Button program will be rewritten so that it follows an algorithm that does the pattern from part 1**



```
BUTTON_SVC:
            MOV R1, #0x00000002         @Value turns off GPIO2_1 and INTC Interrupt requests
            STR R1, [R0]                @Write to GPIO2_IRQSTATUS_0 Register

            @Turn off NEWIRQ bit in INTC Control, so processor can respond to new IRQ

            LDR R0, =0x48200048         @Address of INTC_CONTROL register
            MOV R1, #01                 @Value to clear bit 0
            STR R1, [R0]                @Write to INTC_CONTROL Register

            @Turn on LEDs 21-24 on GPIO1_21-24


            LDR R0, =0x4804C194         @Load address of GPIO1_SETDATAOUT register
            MOV R1, #0x01E0000          @Load value to turn on GPIO1_21-24
            STR R1, [R0]                @Write to GPIO1_SETDATAOUT register

            @Wait two seconds

LOOP2:
            NOP
            SUBS R2, #1                 @Count down
            BNE LOOP2

            @Turn off LEDs 21-24 on GPIO1_21-24

            LDR R0, = 0x4804C190        @Load address of GPIO1_CLEARDATAOUT register
            STR R1, [R0]                @Write to GPIO1_CLEARDATAOUT register

            @Return to wait loop

            LDMFD SP !, {R0-R3, LR}     @Restore registers
            SUBS PC, LR, #4             @Return from IRQ Interrupt Procedure

```

End of program changes (don’t need to have the SYS_IRQ since no hooking or chaining)



```
.align 2
.data
.align 2
STACK1:     .rept 1024
            .word 0x0000
            .endr

STACK2:     .rept 1024
            .word 0x0000
            .endr
.END
```

Task 3: Algorithm for Button

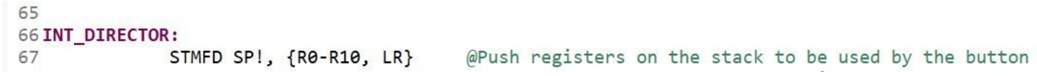
Brainstorm: To determine whether the LEDs are pulsing or not, use TOGGLE memory location (store a word) that changes every time. Store a 1 when pulsing and store a zero when not pulsing. Go to the toggle. Depending on this, I can either run the pattern or turn off the LEDS, and then exit the interrupt. Basically going to need a label that changes the value in toggle each time that the button is pressed.

Algorithm:

```
Load TOMEM from memory (it is already set to 0x0000 0000)

Compare TOMEM register with a temporary register
BEQ to T_ONE

T_ONE:    TURN OFF ALL LEDS

          Change TOMEM = 1 (0x0000 0001)

          Restore registers and return from IRQ

Compare TOMEM register with a temporary register
BEQ to T_ZERO

T_ZERO:   STROBE LEDS

          Change TOMEM = 0 (0x0000 0000)

          Restore registers and return from IRQ
```

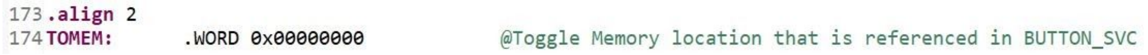
Task 4: The Code

Made changes to INT_DIRECTOR so that there are more registers available on the stack to do what I want it to. I want to separately write to CLEAR and SET data out for the pins when the strobe is happening or when I turn off all of the LEDS



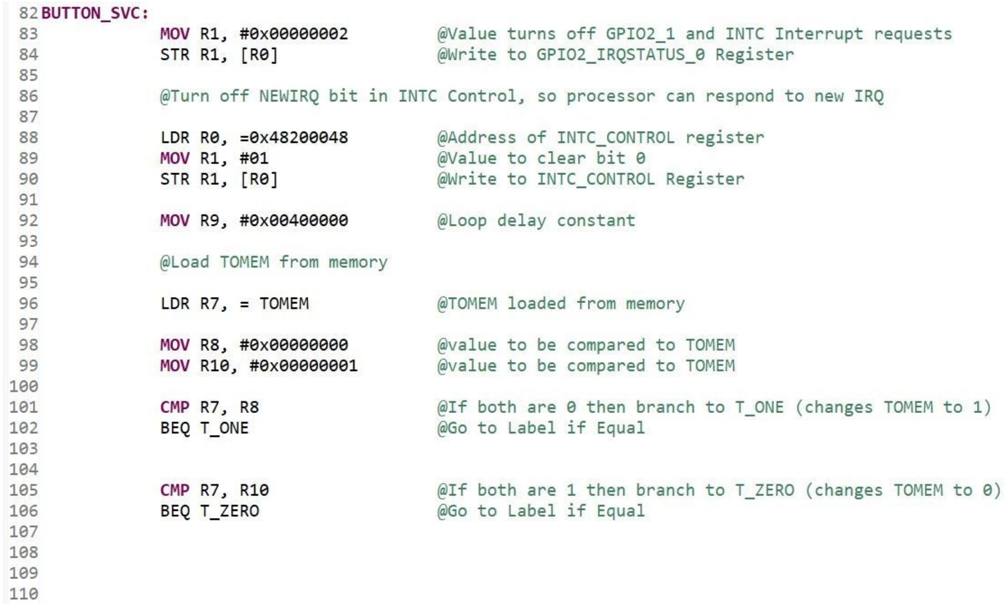
```

INT_DIRECTOR:
            STMFD SP!, {R0-R10, LR}     @Push registers on the stack to be used by the button
```

Another change is to the data points. TOMEM is added to be referenced by the button

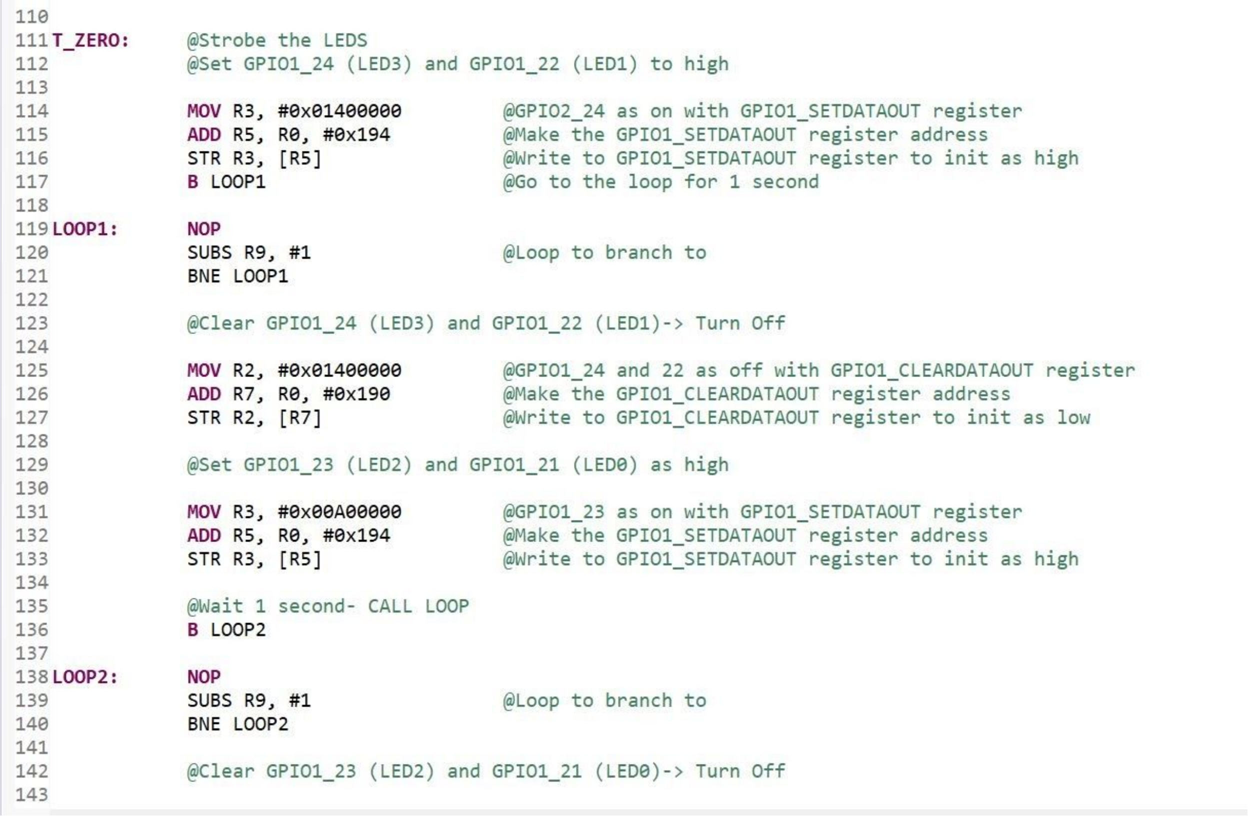


```
.align 2
TOMEM:      .WORD 0x00000000            @Toggle Memory location that is referenced in BUTTON_SVC
```

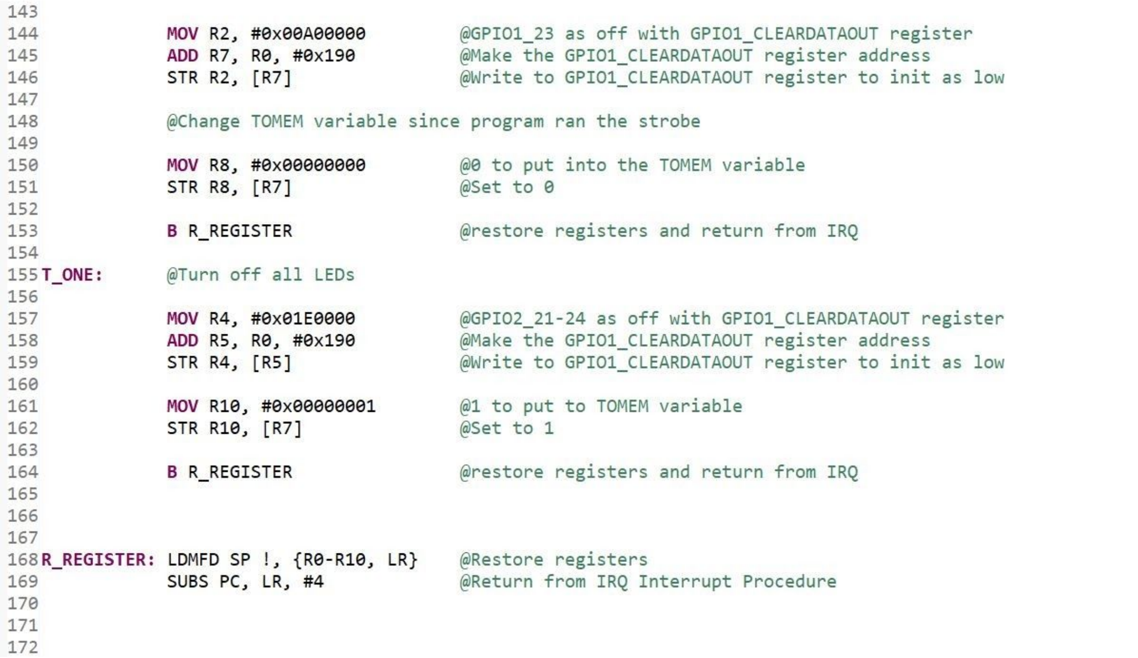
BUTTON_SVC Code:



```
BUTTON_SVC:
            MOV R1, #0x00000002         @Value turns off GPIO2_1 and INTC Interrupt requests
            STR R1, [R0]                @Write to GPIO2_IRQSTATUS_0 Register

            @Turn off NEWIRQ bit in INTC Control, so processor can respond to new IRQ

            LDR R0, =0x48200048         @Address of INTC_CONTROL register
            MOV R1, #01                 @Value to clear bit 0
            STR R1, [R0]                @Write to INTC_CONTROL Register

            MOV R9, #0x00400000         @Loop delay constant

            @Load TOMEM from memory

            LDR R7, = TOMEM             @TOMEM loaded from memory

            MOV R8, #0x00000000         @value to be compared to TOMEM
            MOV R10, #0x00000001        @value to be compared to TOMEM

            CMP R7, R8                  @If both are 0 then branch to T_ONE (changes TOMEM to 1)
            BEQ T_ONE                   @Go to Label if Equal


            CMP R7, R10                 @If both are 1 then branch to T_ZERO (changes TOMEM to 0)
            BEQ T_ZERO                  @Go to Label if Equal




```



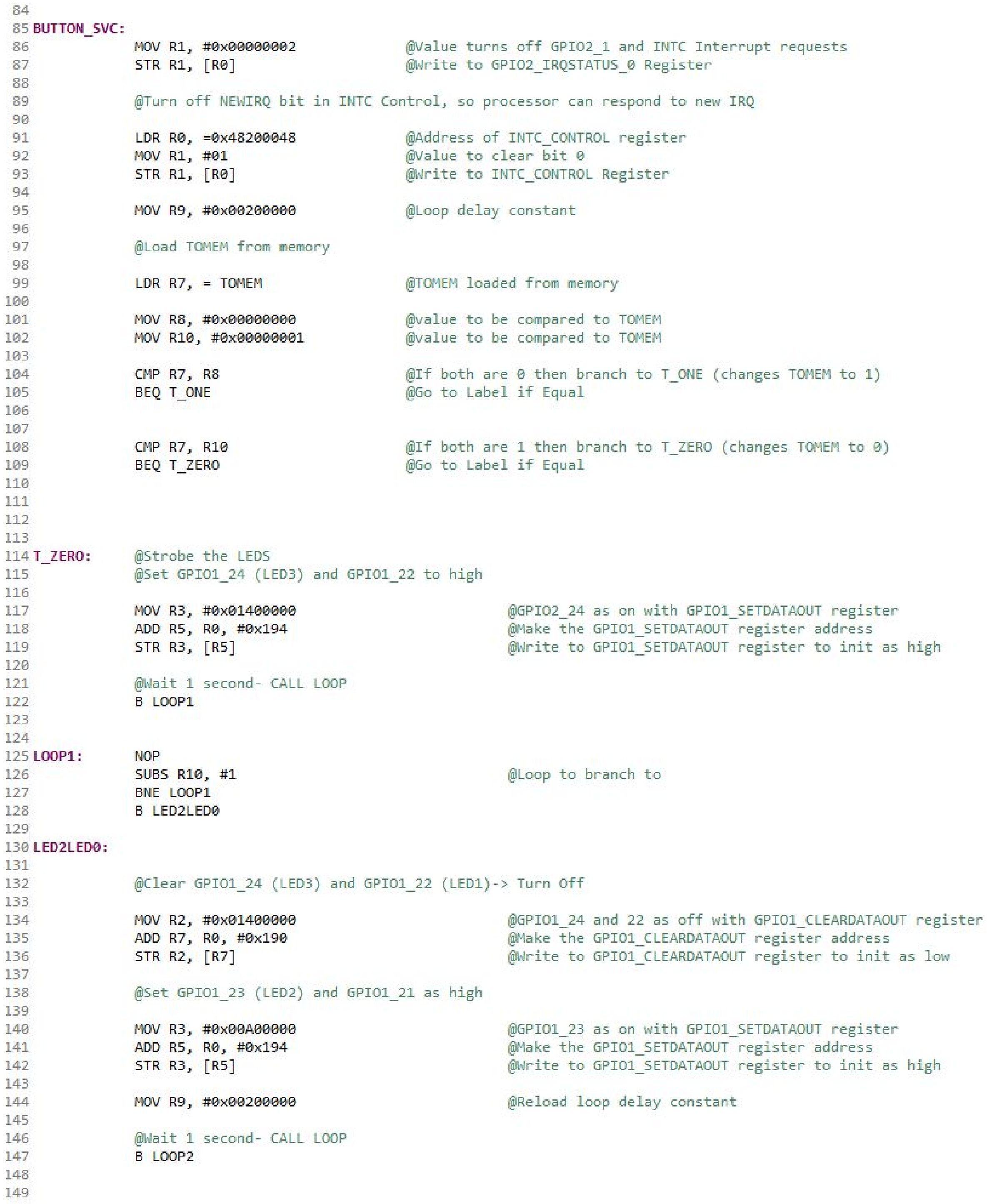
```

T_ZERO:     @Strobe the LEDS
            @Set GPIO1_24 (LED3) and GPIO1_22 (LED1) to high

            MOV R3, #0x01400000         @GPIO2_24 as on with GPIO1_SETDATAOUT register
            ADD R5, R0, #0x194          @Make the GPIO1_SETDATAOUT register address
            STR R3, [R5]                @Write to GPIO1_SETDATAOUT register to init as high
            B LOOP1                     @Go to the loop for 1 second

LOOP1:      NOP
            SUBS R9, #1                 @Loop to branch to
            BNE LOOP1

            @Clear GPIO1_24 (LED3) and GPIO1_22 (LED1)-> Turn Off

            MOV R2, #0x01400000         @GPIO1_24 and 22 as off with GPIO1_CLEARDATAOUT register
            ADD R7, R0, #0x190          @Make the GPIO1_CLEARDATAOUT register address
            STR R2, [R7]                @Write to GPIO1_CLEARDATAOUT register to init as low

            @Set GPIO1_23 (LED2) and GPIO1_21 (LED0) as high

            MOV R3, #0x00A00000         @GPIO1_23 as on with GPIO1_SETDATAOUT register
            ADD R5, R0, #0x194          @Make the GPIO1_SETDATAOUT register address
            STR R3, [R5]                @Write to GPIO1_SETDATAOUT register to init as high

            @Wait 1 second- CALL LOOP
            B LOOP2

LOOP2:      NOP
            SUBS R9, #1                 @Loop to branch to
            BNE LOOP2

            @Clear GPIO1_23 (LED2) and GPIO1_21 (LED0)-> Turn Off

```



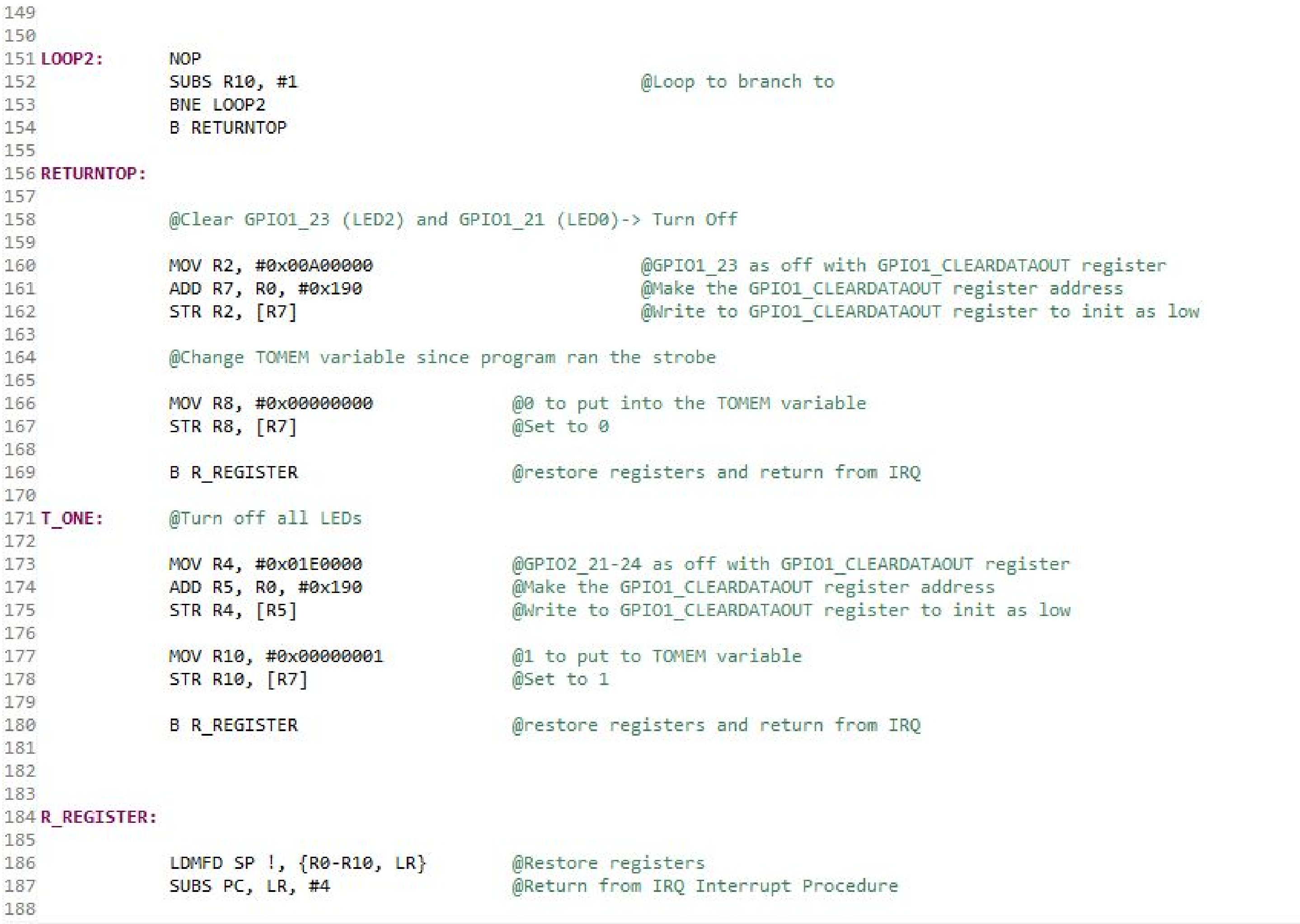
```

            MOV R2, #0x00A00000         @GPIO1_23 as off with GPIO1_CLEARDATAOUT register
            ADD R7, R0, #0x190          @Make the GPIO1_CLEARDATAOUT register address
            STR R2, [R7]                @Write to GPIO1_CLEARDATAOUT register to init as low

            @Change TOMEM variable since program ran the strobe

            MOV R8, #0x00000000         @0 to put into the TOMEM variable
            STR R8, [R7]                @Set to 0

            B R_REGISTER                @restore registers and return from IRQ

T_ONE:      @Turn off all LEDs

            MOV R4, #0x01E0000          @GPIO2_21-24 as off with GPIO1_CLEARDATAOUT register
            ADD R5, R0, #0x190          @Make the GPIO1_CLEARDATAOUT register address
            STR R4, [R5]                @Write to GPIO1_CLEARDATAOUT register to init as low

            MOV R10, #0x00000001        @1 to put to TOMEM variable
            STR R10, [R7]               @Set to 1

            B R_REGISTER                @restore registers and return from IRQ



R_REGISTER: LDMFD SP !, {R0-R10, LR}    @Restore registers
            SUBS PC, LR, #4             @Return from IRQ Interrupt Procedure



```

March 6, 2020

Changes made to the Part I of the program are going to be added to Part 2 to be debugged (this has to do with the loop delay constants and the labels)



```

BUTTON_SVC:
            MOV R1, #0x00000002            @Value turns off GPIO2_1 and INTC Interrupt requests
            STR R1, [R0]                   @Write to GPIO2_IRQSTATUS_0 Register

            @Turn off NEWIRQ bit in INTC Control, so processor can respond to new IRQ

            LDR R0, =0x48200048            @Address of INTC_CONTROL register
            MOV R1, #01                    @Value to clear bit 0
            STR R1, [R0]                   @Write to INTC_CONTROL Register

            MOV R9, #0x00200000            @Loop delay constant

            @Load TOMEM from memory

            LDR R7, = TOMEM                @TOMEM loaded from memory

            MOV R8, #0x00000000            @value to be compared to TOMEM
            MOV R10, #0x00000001           @value to be compared to TOMEM

            CMP R7, R8                     @If both are 0 then branch to T_ONE (changes TOMEM to 1)
            BEQ T_ONE                      @Go to Label if Equal


            CMP R7, R10                    @If both are 1 then branch to T_ZERO (changes TOMEM to 0)
            BEQ T_ZERO                     @Go to Label if Equal




T_ZERO:     @Strobe the LEDS
            @Set GPIO1_24 (LED3) and GPIO1_22 to high

            MOV R3, #0x01400000                        @GPIO2_24 as on with GPIO1_SETDATAOUT register
            ADD R5, R0, #0x194                         @Make the GPIO1_SETDATAOUT register address
            STR R3, [R5]                               @Write to GPIO1_SETDATAOUT register to init as high

            @Wait 1 second- CALL LOOP
            B LOOP1


LOOP1:      NOP
            SUBS R10, #1                               @Loop to branch to
            BNE LOOP1
            B LED2LED0

LED2LED0:

            @Clear GPIO1_24 (LED3) and GPIO1_22 (LED1)-> Turn Off

            MOV R2, #0x01400000                        @GPIO1_24 and 22 as off with GPIO1_CLEARDATAOUT register
            ADD R7, R0, #0x190                         @Make the GPIO1_CLEARDATAOUT register address
            STR R2, [R7]                               @Write to GPIO1_CLEARDATAOUT register to init as low

            @Set GPIO1_23 (LED2) and GPIO1_21 as high

            MOV R3, #0x00A00000                        @GPIO1_23 as on with GPIO1_SETDATAOUT register
            ADD R5, R0, #0x194                         @Make the GPIO1_SETDATAOUT register address
            STR R3, [R5]                               @Write to GPIO1_SETDATAOUT register to init as high

            MOV R9, #0x00200000                        @Reload loop delay constant

            @Wait 1 second- CALL LOOP
            B LOOP2


```



```
LOOP2:      NOP
            SUBS R10, #1                                @Loop to branch to
            BNE LOOP2
            B RETURNTOP

RETURNTOP:

            @Clear GPIO1_23 (LED2) and GPIO1_21 (LED0)-> Turn Off

            MOV R2, #0x00A00000                         @GPIO1_23 as off with GPIO1_CLEARDATAOUT register
            ADD R7, R0, #0x190                          @Make the GPIO1_CLEARDATAOUT register address
            STR R2, [R7]                                @Write to GPIO1_CLEARDATAOUT register to init as low

            @Change TOMEM variable since program ran the strobe

            MOV R8, #0x00000000             @0 to put into the TOMEM variable
            STR R8, [R7]                    @Set to 0

            B R_REGISTER                    @restore registers and return from IRQ

T_ONE:      @Turn off all LEDs

            MOV R4, #0x01E0000              @GPIO2_21-24 as off with GPIO1_CLEARDATAOUT register
            ADD R5, R0, #0x190              @Make the GPIO1_CLEARDATAOUT register address
            STR R4, [R5]                    @Write to GPIO1_CLEARDATAOUT register to init as low

            MOV R10, #0x00000001            @1 to put to TOMEM variable
            STR R10, [R7]                   @Set to 1

            B R_REGISTER                    @restore registers and return from IRQ



R_REGISTER:

            LDMFD SP !, {R0-R10, LR}        @Restore registers
            SUBS PC, LR, #4                 @Return from IRQ Interrupt Procedure
```

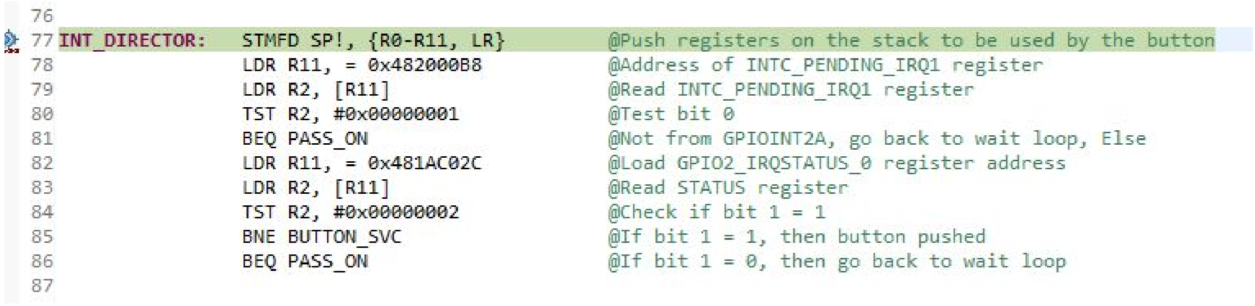
Also I forgot to load the base address of the GPIO1 module and I added it and did not experience issues:

```
            LDR R0, = #0x4804C000          @Load base address of GPIO1: 0x4804C000
```

March 7, 2020

Task 5: Breakpoint on INT_DIRECTOR

I was able to successfully go to INT_DIRECTOR after making a few minor changes to the initialization code (the final updated code will be provided later in the log)

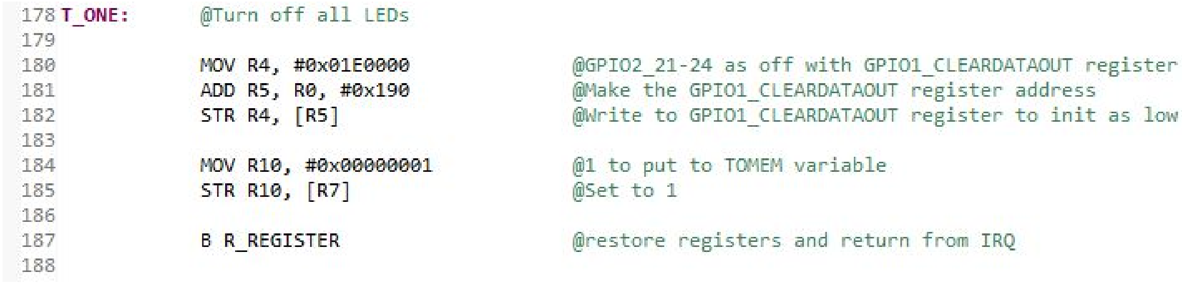


```

INT_DIRECTOR:   STMFD SP!, {R0-R11, LR}         @Push registers on the stack to be used by the button
                LDR R11, = 0x482000B8           @Address of INTC_PENDING_IRQ1 register
                LDR R2, [R11]                   @Read INTC_PENDING_IRQ1 register
                TST R2, #0x00000001             @Test bit 0
                BEQ PASS_ON                     @Not from GPIOINT2A, go back to wait loop, Else
                LDR R11, = 0x481AC02C           @Load GPIO2_IRQSTATUS_0 register address
                LDR R2, [R11]                   @Read STATUS register
                TST R2, #0x00000002             @Check if bit 1 = 1
                BNE BUTTON_SVC                  @If bit 1 = 1, then button pushed
                BEQ PASS_ON                     @If bit 1 = 0, then go back to wait loop

```

Task 6: BUTTON_SVC

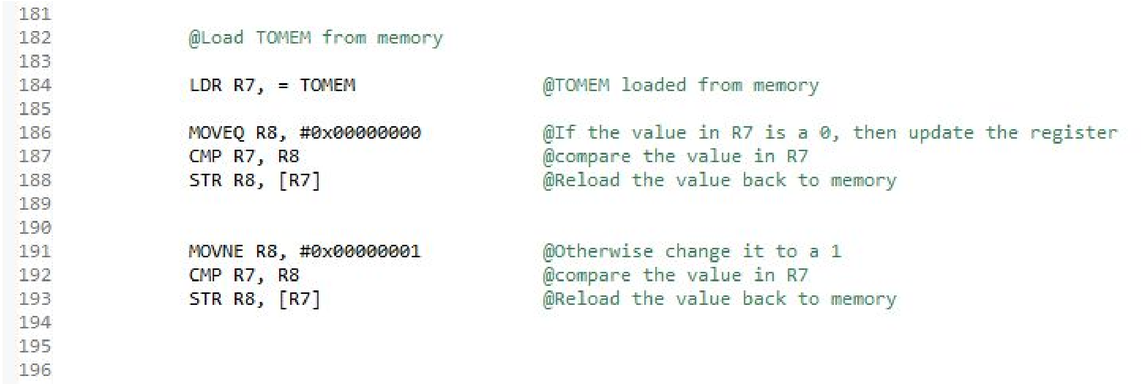
I was able to successfully go through the button service program but there are a few issues below:

Setting up the breakpoints to see what is happening with the button presses shows me that it does not go to T_ONE when I press the button another time.



```
T_ONE:      @Turn off all LEDs

            MOV R4, #0x01E0000              @GPIO2_21-24 as off with GPIO1_CLEARDATAOUT register
            ADD R5, R0, #0x190              @Make the GPIO1_CLEARDATAOUT register address
            STR R4, [R5]                    @Write to GPIO1_CLEARDATAOUT register to init as low

            MOV R10, #0x00000001            @1 to put to TOMEM variable
            STR R10, [R7]                   @Set to 1

            B R_REGISTER                    @restore registers and return from IRQ

```

Changes were made to this section as recommended by Tyler where the T_ONE and T_ZERO are going to happen outside of the button service. My issue for the past 8 hours has been to get the updated value.



```
        @Load TOMEM from memory

        LDR R7, = TOMEM                 @TOMEM loaded from memory

        MOVEQ R8, #0x00000000           @If the value in R7 is a 0, then update the register
        CMP R7, R8                      @compare the value in R7
        STR R8, [R7]                    @Reload the value back to memory


        MOVNE R8, #0x00000001           @Otherwise change it to a 1
        CMP R7, R8                      @compare the value in R7
        STR R8, [R7]                    @Reload the value back to memory
```

This will be the next task. The value in R7 does not get updated correctly as should be expected. What I am trying to do is check whether the value in R8 changed and then update the R7 register properly. This is not happening so there is something wrong with the way the code is written.

March 10, 2020

Here is what I want the algorithm for the comparison to do:

If same

```
Initial value = 0


          0 0            change to 1

Pseudocode

CMP       0 0
MOVEQ     R 1
STR       R 1
```

```
Initial value = 1

          1 1            change to 0
```

Pseudocode

```
CMP       1 1
MOVEQ     R 0
STR       R 0
```

If different

```
Initial value = 0
          0 1            store 1
```

Pseudocode

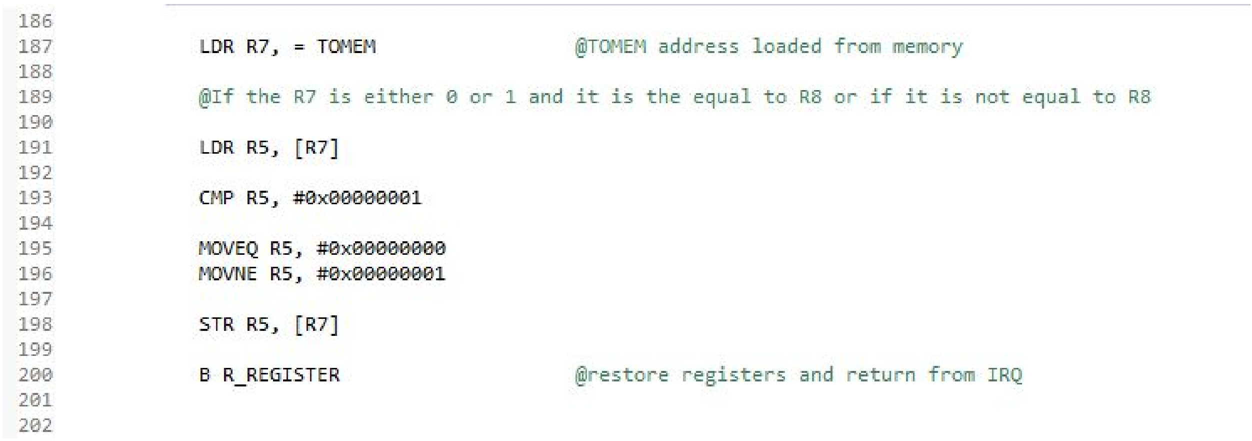
```
CMP       0 1
MOVNE     R 1
STR       R 1
```

```
Initial value = 1
          1 0            store 0
```

Pseudocode

```
CMP       1 0
MOVNE     R 0
STR       R 0
```

Since there are two zeros and two ones, I can consolidate these instructions to a few lines:

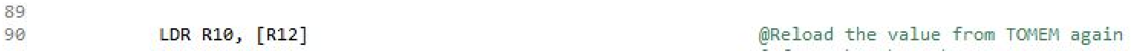


```
        LDR R7, = TOMEM                 @TOMEM address loaded from memory

        @If the R7 is either 0 or 1 and it is the equal to R8 or if it is not equal to R8

        LDR R5, [R7]

        CMP R5, #0x00000001

        MOVEQ R5, #0x00000000
        MOVNE R5, #0x00000001

        STR R5, [R7]

        B R_REGISTER                    @restore registers and return from IRQ
```

This solution worked because I am now correctly reading the value from memory instead of reading its address like I was before.

Next issue: The solution works and the entire program is able to work by turning on the LEDs in the sequence and then when the button press happens again, the lights are off. But when the button is pressed a third time, the LED strobe does not happen until the program is reloaded and run again from the button press.

The fix to this problem is to reload the memory location within the T_ONE and T_ZERO code



```
        LDR R10, [R12]                                  @Reload the value from TOMEM again
```

The program works without the breakpoints and the demo is completed

The signoff was received 3/10/2020