EE447 Pre Liminary # 2

Q1. (20%) Write a subroutine, DELAY100, that causes approximately 100 msec delay upon calling.

Ans. We were tasked to create a delay that lasts around 100ms approximately, below the code is shown that runs for approximately 100ms according to the clock frequency of the simulation panel which runs at 12 MHz, since we are using 5 cycles the subtraction should be executed from R0 this many times:

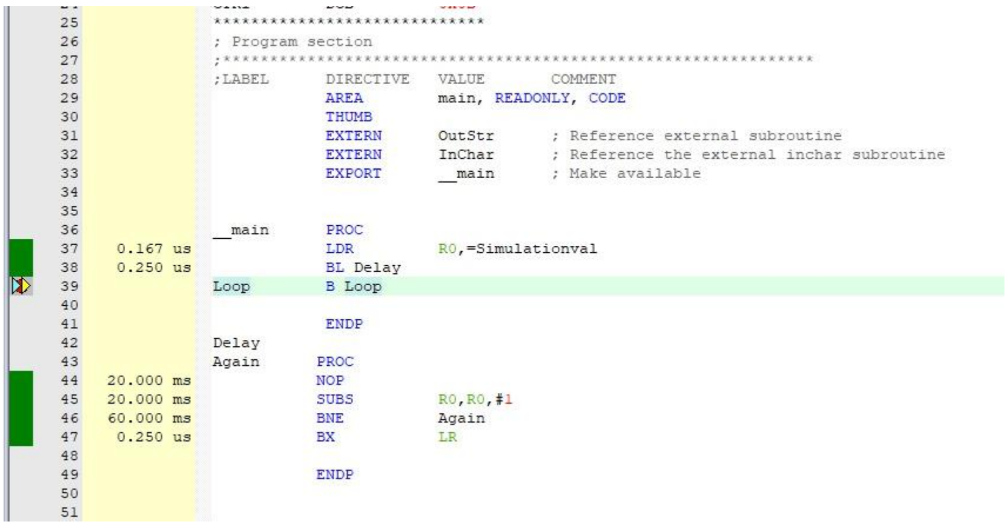
$$5\ cycles\ @\ 12\ MHz \rightarrow 5 \times \frac{1}{12 \times 10^6} \approx 0.4167\ \mu s$$

$$we\ need\ 100\ ms\ so\ Num = \frac{100 \times 10^{-3}}{0.416667 \times 10^{-6}} = 240{,}000\ (Dec) = 3A980\ (Hex)$$

However, the clocked frequency1 for the board is 16 MHz therefore the equivalent number value for the board will be:

$$Ratio \approx \frac{16MHz}{12MHz} \approx 1.33333,\ \ now\ the\ new\ Num = 240{,}000 \times 1.333 = 320{,}000\ (Dec) = 4E200\ (Hex)$$

The below pictures show the simulation results @ 16 MHz and @ 12 MHz:



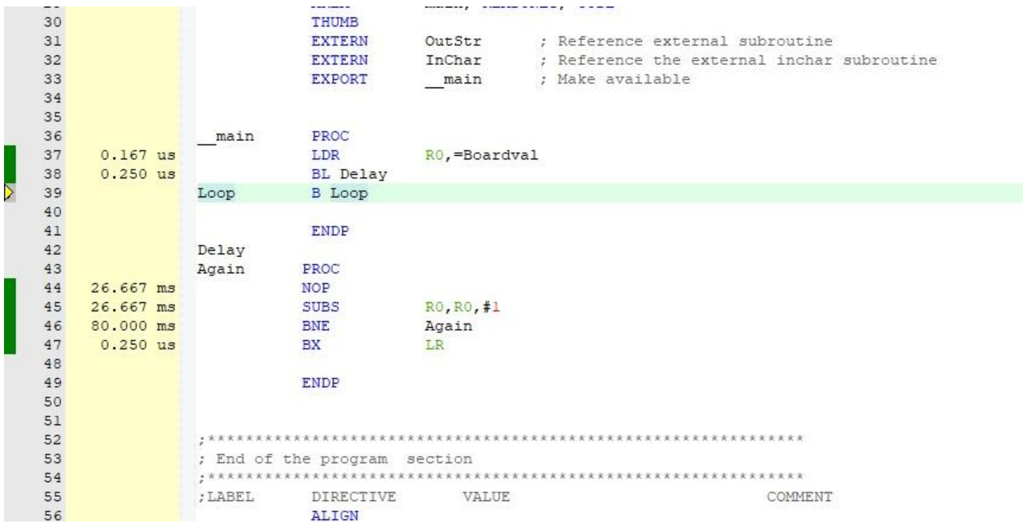
```
                 ****************************
                 ; Program section
                 ;*****************************************************************
                 ;LABEL      DIRECTIVE   VALUE       COMMENT
                             AREA        main, READONLY, CODE
                             THUMB
                             EXTERN      OutStr      ; Reference external subroutine
                             EXTERN      InChar      ; Reference the external inchar subroutine
                             EXPORT      __main      ; Make available


                 __main      PROC
    0.167 us                 LDR         R0,=Simulationval
    0.250 us                 BL Delay
                 Loop        B Loop

                             ENDP
                 Delay
                 Again      PROC
   20.000 ms                NOP
   20.000 ms                SUBS         R0,R0,#1
   60.000 ms                BNE          Again
    0.250 us                BX           LR

                            ENDP
```

Fig 1. Delay loop for 100 ms @ 12 MHz



```
                             THUMB
                             EXTERN      OutStr      ; Reference external subroutine
                             EXTERN      InChar      ; Reference the external inchar subroutine
                             EXPORT      __main      ; Make available


                 __main      PROC
    0.167 us                 LDR         R0,=Boardval
    0.250 us                 BL Delay
                 Loop        B Loop

                             ENDP
                 Delay
                 Again      PROC
   26.667 ms                NOP
   26.667 ms                SUBS         R0,R0,#1
   80.000 ms                BNE          Again
    0.250 us                BX           LR

                            ENDP


                 ;*****************************************************************
                 ; End of the program  section
                 ;*****************************************************************
                 ;LABEL      DIRECTIVE       VALUE                           COMMENT
                             ALIGN
```

Fig 2. Delay loop for 100 ms @ 16 MHz

Syed Muhammad Farrukh Aijaz-2417152 Mubeen Ahmad Fayyaz-2415842

EE447 Pre Liminary # 2

Q2. (20%) Write a program for a simple data transfer scheme. You are required to take inputs from ………. low nibble of Port B (B3 to B0) for inputs, and high nibble of Port B (B7 to B4) for outputs.

Ans. In this part of code debouncing and masking were equally considered, and the equivalent hardcoded masking of the LED's are in the code just for reference so they can be shown at the memory address,

LEDONE	*EQU*	*0xE0*
LEDTWO	*EQU*	*0xD0*
LEDTHREE	*EQU*	*0xB0*
LEDFOUR	*EQU*	*0x70*

The delay subroutine is also inside the algorithm which is called almost **40 times** for a 4 second delay until the next key is pressed and the output LED is shown in the pictures below:

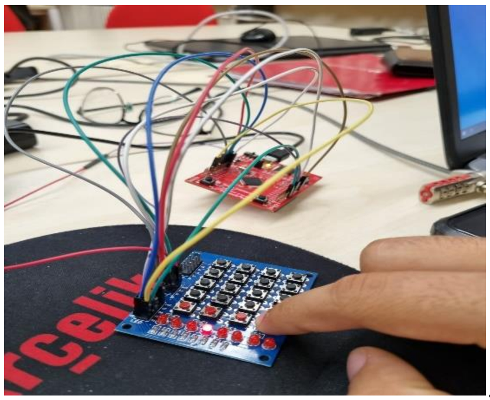
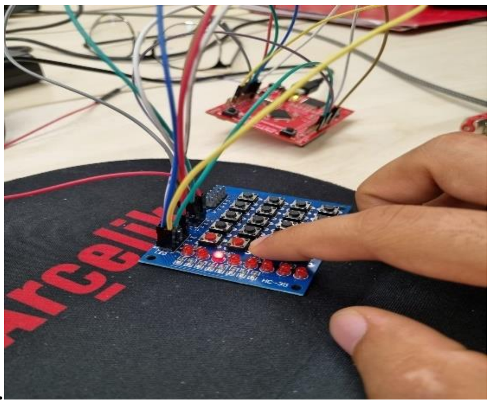
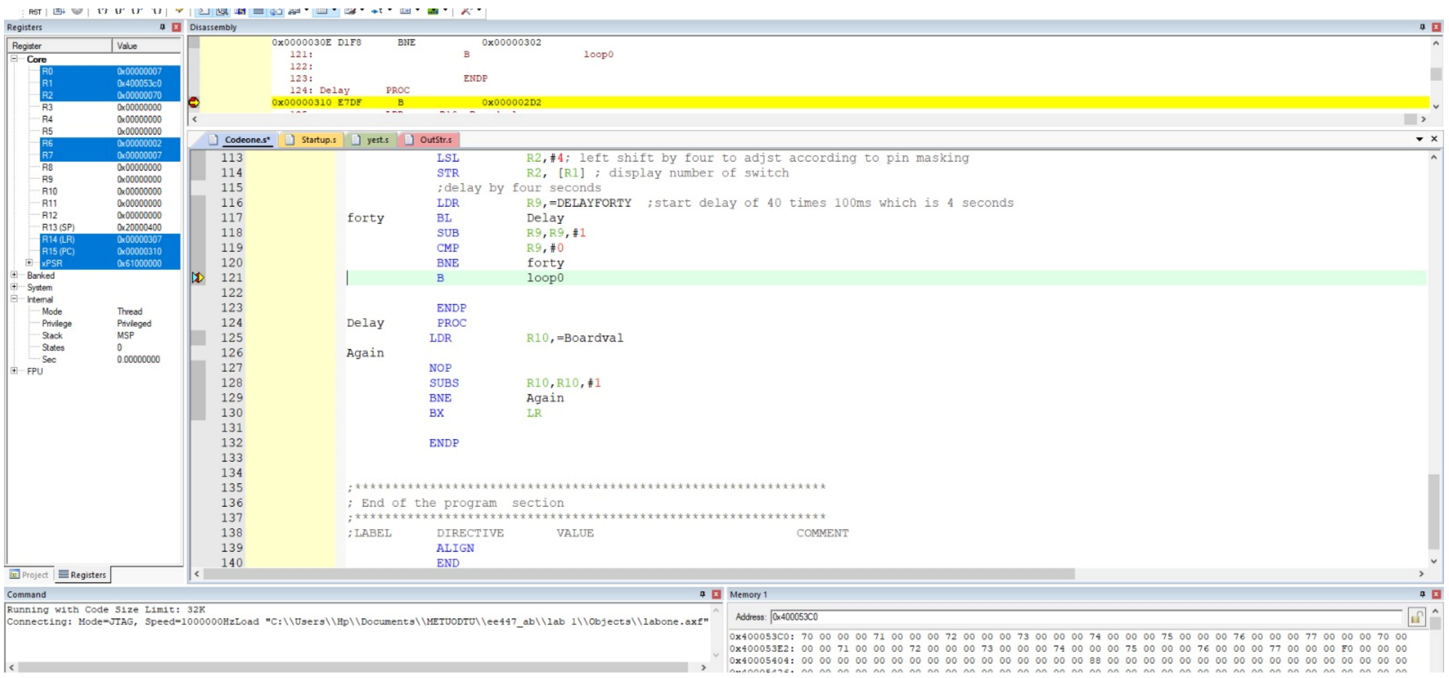

….1s….2s….3s….


Fig 3. Display of LED's 3 & 4 , after a delay of 4 seconds



```
Registers
Register        Value
Core
  R0            0x00000007
  R1            0x400053c0
  R2            0x00000070
  R3            0x00000000
  R4            0x00000000
  R5            0x00000000
  R6            0x00000002
  R7            0x00000007
  R8            0x00000000
  R9            0x00000000
  R10           0x00000000
  R11           0x00000000
  R12           0x00000000
  R13 (SP)      0x20000400
  R14 (LR)      0x00000307
  R15 (PC)      0x00000310
  xPSR          0x61000000
Banked
System
Internal
  Mode          Thread
  Privilege     Privileged
  Stack         MSP
  States        0
  Sec           0.00000000
FPU
```

```
Disassembly
0x0000030E D1F8      BNE          0x00000302
   121:                      B                  loop0
   122:
   123:                      ENDP
   124: Delay        PROC
0x00000310 E7DF      B            0x000002D2
```

```
                LSL         R2,#4; left shift by four to adjst according to pin masking
                STR         R2, [R1] ; display number of switch
                ;delay by four seconds
                LDR         R9,=DELAYFORTY  ;start delay of 40 times 100ms which is 4 seconds
forty           BL          Delay
                SUB         R9,R9,#1
                CMP         R9,#0
                BNE         forty
                B           loop0

                ENDP
Delay           PROC
               LDR          R10,=Boardval
Again
               NOP
               SUBS         R10,R10,#1
               BNE          Again
               BX           LR

               ENDP


;****************************************************************
; End of the program  section
;****************************************************************
;LABEL      DIRECTIVE       VALUE                           COMMENT
           ALIGN
           END
```

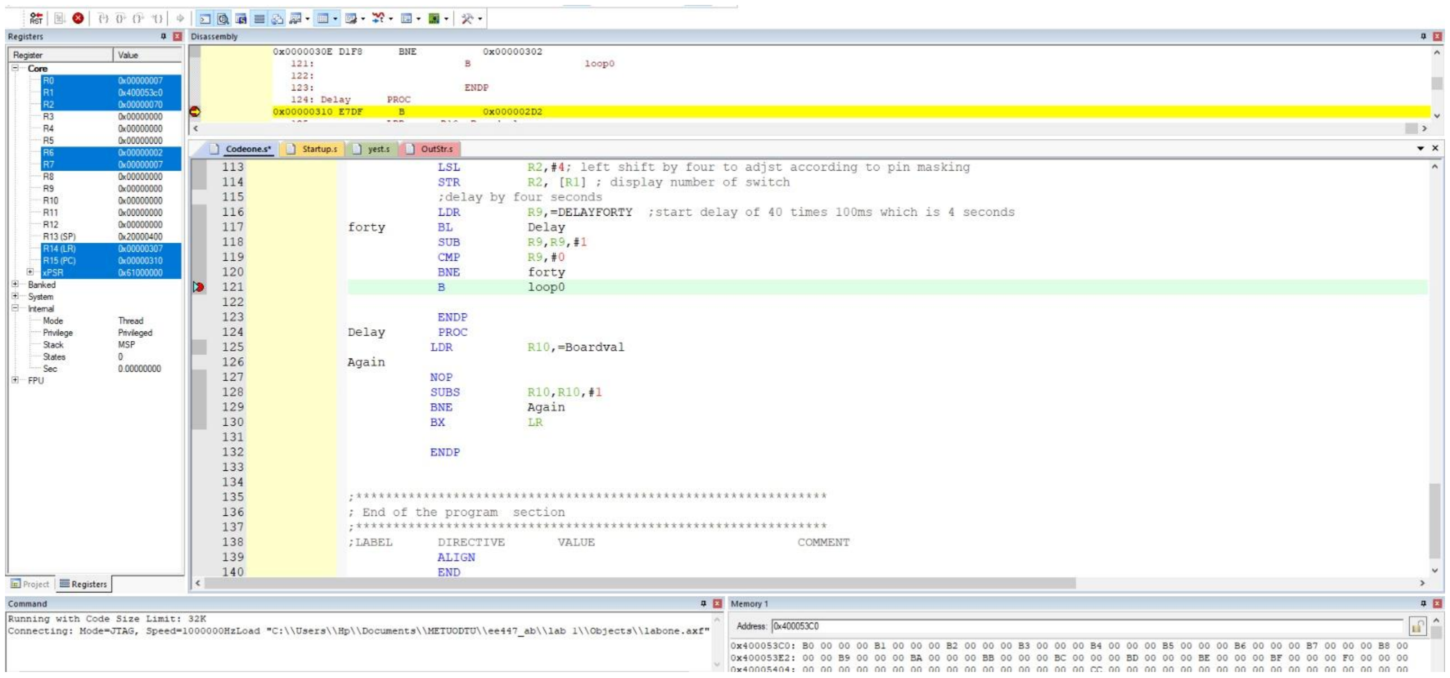
```
Command
Running with Code Size Limit: 32K
Connecting: Mode=JTAG, Speed=1000000HzLoad "C:\\Users\\Hp\\Documents\\METUODTU\\ee447_ab\\lab 1\\Objects\\labone.axf"
```

```
Memory 1
Address: 0x400053C0
0x400053C0: 70 00 00 00 71 00 00 00 72 00 00 00 73 00 00 00 74 00 00 00 75 00 00 00 76 00 00 00 77 00 00 00 70 00
0x400053E2: 00 00 71 00 00 00 72 00 00 00 73 00 00 00 74 00 00 00 75 00 00 00 76 00 00 00 77 00 00 00 F0 00 00 00
0x40005404: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 88 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Fig 4. Showing masked LEDFOUR hex code in the memory address 0x400053C0

Syed Muhammad Farrukh Aijaz-2417152 Mubeen Ahmad Fayyaz-2415842

EE447 Pre Liminary # 2



```
Register        Value
Core
  R0            0x00000007
  R1            0x400053c0
  R2            0x00000070
  R3            0x00000000
  R4            0x00000000
  R5            0x00000000
  R6            0x00000002
  R7            0x00000007
  R8            0x00000000
  R9            0x00000000
  R10           0x00000000
  R11           0x00000000
  R12           0x00000000
  R13 (SP)      0x20000400
  R14 (LR)      0x00000307
  R15 (PC)      0x00000310
  xPSR          0x61000000
Banked
System
Internal
  Mode          Thread
  Privilege     Privileged
  Stack         MSP
  States        0
  Sec           0.00000000
FPU
```

```
0x0000030E D1F8      BNE          0x00000302
   121:                      B                  loop0
   122:
   123:                      ENDP
   124: Delay      PROC
0x00000310 E7DF      B            0x000002D2
```

```
                LSL         R2,#4; left shift by four to adjst according to pin masking
                STR         R2, [R1] ; display number of switch
                ;delay by four seconds
                LDR         R9,=DELAYFORTY  ;start delay of 40 times 100ms which is 4 seconds
forty           BL          Delay
                SUB         R9,R9,#1
                CMP         R9,#0
                BNE         forty
                B           loop0

                ENDP
Delay           PROC
               LDR          R10,=Boardval
Again
               NOP
               SUBS         R10,R10,#1
               BNE          Again
               BX           LR

               ENDP


;***************************************************************
; End of the program  section
;***************************************************************
;LABEL          DIRECTIVE       VALUE                           COMMENT
                ALIGN
                END
```

```
Running with Code Size Limit: 32K
Connecting: Mode=JTAG, Speed=1000000HzLoad "C:\\Users\\Hp\\Documents\\METUODTU\\ee447_ab\\lab 1\\Objects\\labone.axf"
```

```
Address: 0x400053C0
0x400053C0: B0 00 00 00 B1 00 00 00 B2 00 00 00 B3 00 00 00 B4 00 00 00 B5 00 00 00 B6 00 00 00 B7 00 00 00 B8 00
0x400053E2: 00 00 B9 00 00 00 BA 00 00 00 BB 00 00 00 BC 00 00 00 BD 00 00 00 BE 00 00 00 BF 00 00 00 F0 00 00 00
0x40005404: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 CC 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Fig 5. Showing masked LEDTHREE hex code in the memory address 0x400053C0

Q3. Consider the interface of the 4x4 keypad …….to possible bouncing effect during both pressing and releasing. Please attempt this problem by answering the following items:

a) How can you detect whether any key is pressed?

Ans.

The hex keypad has 8 communication lines namely R1, R2, R3, R4, L1, L2, L3 and L4. R1 to R4 represents the four columns and L1 to L4 represents the four rows. When a particular key is pressed the corresponding row and column to which the terminals of the key are connected gets shorted. For example if key 1 is pressed row L1 and column R1 gets shorted and so on.

The program identifies which key is pressed by a method known as column scanning. In this method a particular row is kept low (other rows are kept high) and the columns are checked for low. If a particular column is found low then that means that the key connected between that column and the corresponding row (the row that is kept low) is been pressed. For example if row L1 is initially kept low and column R1 is found low during scanning, that means key 1 is pressed.

b) How can you detect whether a pressed key is released?

Ans.

A simple algorithm watches the key's status over time to determine when a pressed key is released. Initially, a timestamp and the condition of the key that was identified as pushed are logged. The stability of the pressed state is then continuously verified, taking into consideration bouncing effects that might result in abrupt changes between the pressed and released states. A key is deemed pushed if it is held down for a predetermined steady duration. A timestamp is recorded if the key changes into a released state within this period.

The system then determines if the key stayed steady during the debounce period, signifying a legitimate key release. If this is the case, the system can carry out the intended key release operations. By using this method, bouncing-induced transient states are filtered out and precise key release detection in embedded systems is guaranteed. It is important to adjust the debounce time parameter; depending on the features of the particular keypad and the anticipated bouncing behavior, this may need to be fine-tuned.

c) **Assuming that you have detected that a key is pressed. Explain your algorithm to determine which one is pressed?**

Ans.

For example if key 1 is pressed row L1 and column R1 gets shorted and so on. The program identifies which key is pressed by a method known as column scanning. In this method a particular row is kept low (other rows are kept high) and the columns are checked for low.

If a particular column is found low then that means that the key connected between that column and the corresponding row (the row that is kept low) is been pressed. For example, if row L1 is initially kept low and column R1 is found low during scanning, that means key 1 is pressed.

Starting from the top row, the microcontroller will ground it by providing a low to row L1 only. Now read the columns, if the data read is all 1s, no key in that row is pressed and the process continues for the next row. So, now ground the next row, L2.

Read the columns, check for any zero and this process continues until the row is identified. E.g. In the above case we will get row 2 in which column is not equal to 1111. So, after identification of the row in which the key has been pressed, we can easily find out the key by row and column value.

d) **Discuss what can happen due to bouncing. How can you avoid bouncing effects?**

Ans.

Bouncing in switch signals can lead to multiple state transitions, causing issues such as unintended actions, erroneous input, and increased interrupt latency. To mitigate these effects, various methods are employed. Hardware debouncing utilizes components like resistors and capacitors to create low-pass filters that smooth out switch signals. Software debouncing involves implementing algorithms in firmware to filter out transient changes, often using delay loops, state machines, or timers. Mechanical solutions include choosing high-quality switches with low bounce characteristics or those with built-in hardware debouncing circuits.

Adjusting interrupt handling routines to accommodate bouncing and using timers or delays is another effective approach. Requiring a stable switch state for a predefined duration helps filter out brief bouncing events. Additionally, filtering techniques like low-pass filters or median filtering can be applied in software. Some microcontroller platforms offer debouncing libraries or functions to simplify implementation. By employing a combination of hardware and software solutions, these methods contribute to a more reliable switch interface, minimizing the impact of bouncing in embedded systems.

e) **Now, develop your overall end-to-end algorithm that outputs ID of the pressed key to the terminal window and draw its flow chart.**

Ans.

Initialization: The algorithm begins with the initialization phase, where the required GPIO pins for the keypad's rows and columns are configured. Additionally, crucial variables are set up, including a 16 msg key to map the combinations of rows and columns to specific key IDs. A debounce duration is defined to confirm a stable state after a key press, addressing potential bouncing effects.

Key Detection Loop: The algorithm enters a continuous loop responsible for scanning through each row of the keypad. For each row, the corresponding columns are activated and read. The algorithm checks if a key press is detected by examining whether any column in the activated row is low. If a key press is identified, the corresponding key ID is recorded, and the system transitions to the debounce phase.

Debouncing: Upon detecting a key press, the algorithm initiates a debounce phase to mitigate the bouncing effects. It waits for a predefined debounce duration, continually checking for stability during this period. If the key remains stable, the algorithm proceeds to the key identification phase; otherwise, it continues debouncing until stability is achieved.

Key Identification: In the key identification phase, the algorithm determines the specific key that was pressed based on the row and column information. The corresponding key ID is then output to the terminal window.

Continue Scanning: The algorithm concludes each iteration by returning to the key detection loop, ensuring the continuous monitoring of the keypad for new key presses. This cyclic process repeats until the program is terminated.

Flowchart:

Start → Initialization → Key Detection Loop →

?

Key Press Detected ? →

Yes →

Record Key ID →

Enter Debounce Phase →

Wait for Stable State →

Stable State Reached? →

Yes →

Identify Key →

Output Key ID →

Continue Scanning →

No →

Continue Debouncing →

Continue Scanning →

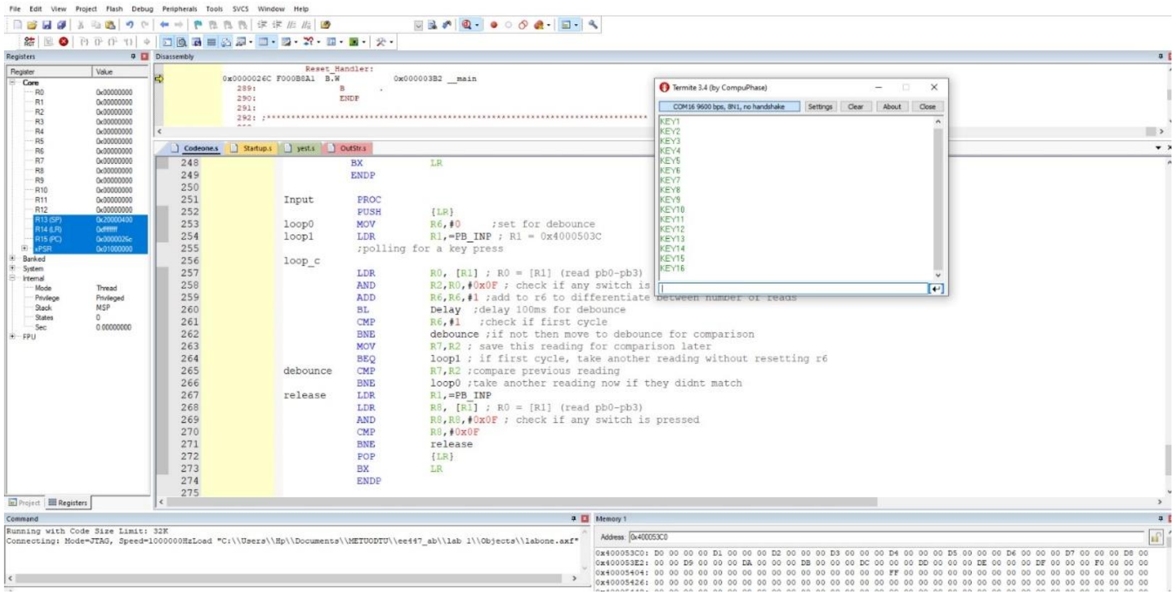
No →

Continue Scanning →

End

f) Implement the developed algorithm in part-e by using assembly language.

The code is given in the pre-liminary files and the output can be shown in termite as below:



```
                 Reset_Handler:
0x0000026C F000B8A1  B.W          0x000003B2 __main
   289:                B        .
   290:                ENDP
   291:
   292: ;*****************************************************************************
```

```
                BX          LR
                ENDP

Input           PROC
                PUSH        {LR}
loop0           MOV         R6,#0      ;set for debounce
loop1           LDR         R1,=PB_INP ; R1 = 0x4000503C
                ;polling for a key press
loop_c
                LDR         R0, [R1] ; R0 = [R1] (read pb0-pb3)
                AND         R2,R0,#0x0F ; check if any switch is
                ADD         R6,R6,#1 ;add to r6 to differentiate between number of reads
                BL          Delay  ;delay 100ms for debounce
                CMP         R6,#1    ;check if first cycle
                BNE         debounce ;if not then move to debounce for comparison
                MOV         R7,R2 ; save this reading for comparison later
                BEQ         loop1 ; if first cycle, take another reading without resetting r6
debounce        CMP         R7,R2 ;compare previous reading
                BNE         loop0 ;take another reading now if they didnt match
release         LDR         R1,=PB_INP
                LDR         R8, [R1] ; R0 = [R1] (read pb0-pb3)
                AND         R8,R8,#0x0F ; check if any switch is pressed
                CMP         R8,#0x0F
                BNE         release
                POP         {LR}
                BX          LR
                ENDP
```

```
KEY1
KEY2
KEY3
KEY4
KEY5
KEY6
KEY7
KEY8
KEY9
KEY10
KEY11
KEY12
KEY13
KEY14
KEY15
KEY16
```

```
Running with Code Size Limit: 32K
Connecting: Mode=JTAG, Speed=1000000HzLoad "C:\\Users\\Hp\\Documents\\METUODTU\\ee447_ab\\lab 1\\Objects\\labone.axf"
```

```
Address: 0x400053C0
0x400053C0: D0 00 00 00 D1 00 00 00 D2 00 00 00 D3 00 00 00 D4 00 00 00 D5 00 00 00 D6 00 00 00 D7 00 00 00 D8 00
0x400053E2: 00 00 D9 00 00 00 DA 00 00 00 DB 00 00 00 DC 00 00 00 DD 00 00 00 DE 00 00 00 DF 00 00 00 F0 00 00 00
0x40005404: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40005426: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Fig 6. Different key presses and their specific KEYID is displayed in TERMITE