ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 212E
MICROPROCESSOR SYSTEMS

HOMEWORK NO : 2

SUBMITION DATE : 31.12.2024

GROUP MEMBERS:

150210100 : Selin Yılmaz

FALL 2024

Contents

1 SYSTEM TIMER

The *system tick timer* is a simple 24-bit down counter to produce a small fixed time quantum. The timer counts down from N-1 to 0, and the processor generates a SysTick interrupt once the counter reaches 0. After reaching zero, the SysTick counter loads the value held in a special register named the SysTick Reload register and counts down again.

Its registers look like Fig. 1.

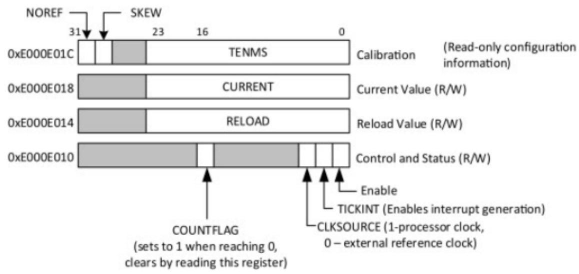

Figure 1: SysTick Registers

At the beginning of the file, some initializations were made for later use.

```
        AREA  SysTick_Definitions, DATA, READONLY
SysTick_CTRL  EQU 0xE000E010    ; SysTick Control and STATES Register.
SysTick_LOAD  EQU 0xE000E014    ; SysTick Reload Value Register
SysTick_VAL   EQU 0xE000E018    ; SysTick Current Value Register
```

To implement the system timer, I used 3 functions.

1.1 Systick_Start

First, it resets *ticks*.

```
        LDR    R0, =ticks
        MOVS   R1, #0
        STR    R1, [R0]
```

The following formula is used to calculate the *reload value*.

*SysTick_LOAD = SysTick Interrupt Period * SysTick Counter Clock Frequency* - 1

The period is chosen 1µs and Clock Frequency is (*System Core Clock* / 100000) MHz. So the initializations for calculating the reload value is as follows.

```
        LDR    R0, =SysTick_LOAD
        LDR    R1, =SystemCoreClock
        LDR    R2, [R1]
        LDR    R1, =100000

        BL     Division
```

A *Division* function is implemented to find quotient (value).

```
Division FUNCTION
  ; Input: R2 = dividend, R1 = divisor
  ; Output: R3 = quotient
        MOVS  R3, #0          ; Initialize quotient to 0
        MOVS  R0, R2          ; R0 = dividend
Loop
        CMP   R0, R1          ; Compare dividend and divisor
        BLT   end_divide        ; End when r2 < r1
        SUBS  R0, R0, R1        ; R2 = R2 - R1
        ADDS  R3, R3, #1        ; Increment quotient
        B     Loop          ; Repeat the loop
end_divide
        BX    LR            ; Return
        ENDFUNC
```

The clock frequency is in R3 after division. Then, I decremented it to find the reload value. Then, the reload value is loaded to the LOAD register.

```
        LDR   R0, =SysTick_LOAD
        SUBS  R3, R3, #1
        STR   R3, [R0]
```

The current value in the value register is reseted.

```
        LDR   R0, =SysTick_VAL
        MOVS  R1, #0
        STR   R1, [R0]
```

Finally, the control and status register is configured.

```
        LDR   R0, =SysTick_CTRL
        LDR   R1, =0x00000007
        STR   R1, [R0]
```

1.2 Systick_Stop

First, it stops the timer by clearing the enable flag. It does it by creating a mask and it clears only the lowest bit (enable).

```
        LDR   R0, =SysTick_CTRL
        LDR   R1, [R0]
        ; ENABLE_Mask
        LDR   R2, =0xFFFFFFFE
        ANDS  R1, R1, R2
        STR   R1, [R0]
```

Then, it reads the *ticks*, resets it, and returns them (in R0 it is the return register).

```
        ; Read ticks
        LDR   R0, =ticks
        LDR   R1, [R0]

        ; Reset ticks
        MOVS  R2, #0
        STR   R2, [R0]

        ; Return must be in R0
        MOV   R0, R1
```

1.3 Systick_Handler

This function only counts the ticks.

```
        LDR   R0, =ticks
        LDR   R1, [R0]
        ADDS  R1, R1, #1
        STR   R1, [R0]
```

After writing this part, I uncommented the *EXPORT* instruction given above and commented the *SysTick_Handler* function in *timing.c* file as expected.

2 SORTING ALGORITHM

A bubble sort algorithm with a linked list structure is asked to be implemented in assembly. The pseudocode for it is as below.

Algorithm 1 ft_lstsort_asm

```
Input: Pointer to list (head), Comparison function (ft_cmp)
Output: Sorted list
Initialize swap_flag to 1
while swap_flag = 1 do
    Set swap_flag to 0
    prev ← NULL
    current ← head
    while current ≠ NULL AND current->next ≠ NULL do
        Compare current->value and current->next->value using ft_cmp
        if ft_cmp(current->value, current->next->value) = 0 then
            Swap current and current->next
            Update links to maintain the list structure
            Set swap_flag to 1
            if current is head then
                Update head to point to current->next
            else
                Update prev to point to current->next
            end if
        end if
        Move to the next pair of nodes
        prev ← current
        current ← current->next
    end while
end while
Return sorted list
```

After planning this pseudocode, I implemented it in assembly language.

3 BIG O ANALYSIS

Number of Sorted Elements	`time`	`time_asm`
5	5	5
10	14	19
15	25	48
20	35	86
25	47	146
30	59	255
35	71	361
40	85	440
45	99	549
50	114	695
55	126	772
60	141	898
65	155	1189
70	169	1329
75	185	1433
80	198	1810
85	216	2077
90	231	2218
95	248	2374
100	263	2528

Table 1: Time Measurements

The theoretical time complexity of merge sort is

$$T(n) = O(nlogn)$$

To verify the complexity, we calculate $\frac{\text{time}}{n \log n}$ for the given data. The results are as follows:

From the table 2, $\frac{\text{time}}{n \log n}$ is nearly constant, confirming the $O(n \log n)$ growth rate.

Graphic for `time` measurements according to number of elements can be observed from Fig. 2.

n	**time**	$\log n$ (base 2)	$n \log n$	$\frac{\text{time}}{n \log n}$
5	5	2.32	11.6	0.43
10	14	3.32	33.2	0.42
20	35	4.32	86.4	0.41
100	263	6.64	664.0	0.40

Table 2: Analysis of Merge Sort Data

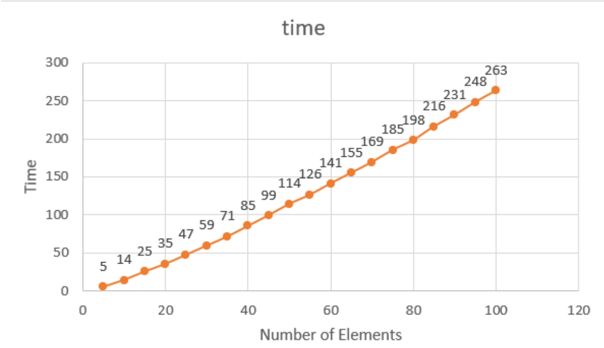

Figure 2: Time x Number of Elements

The theoretical time complexity of bubble sort is

$$T(n) = O(n^2)$$

To verify the complexity, we calculate $\frac{\text{time\_asm}}{n^2}$ for the given data. The results are as follows:

n	**time_asm**	n^2	$\frac{\text{time\_asm}}{n^2}$
5	5	25	0.20
10	19	100	0.19
20	86	400	0.22
100	2528	10000	0.25

Table 3: Analysis of Bubble Sort Data

From the table 3, $\frac{\text{time\_asm}}{n^2}$ is roughly constant, confirming the $O(n^2)$ growth rate.

Graphic for `time_asm` measurements according to the number of elements can be observed from Fig. 3.

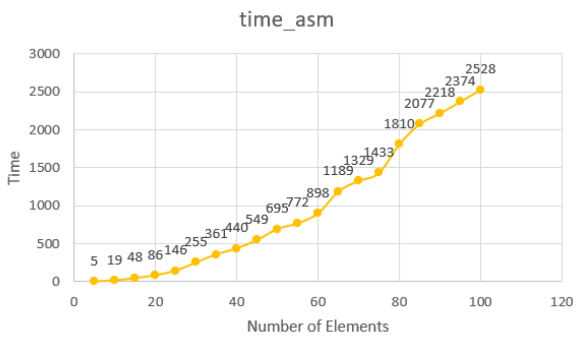

Figure 3: Time_asm x Number of Elements

Together, their graph looks like Fig. 4. The yellow line represents time measurement of bubble sort, the orange line represents time measurement of merge sort.

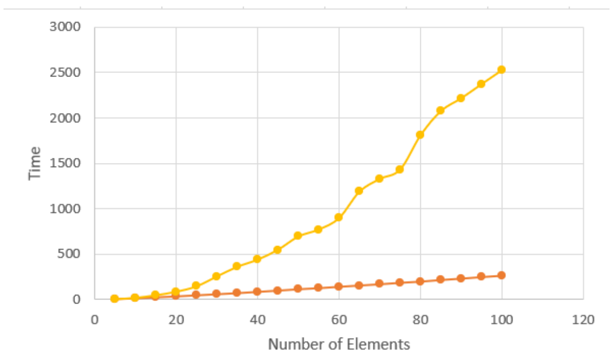

Figure 4: Time x Number of Elements

REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 4 edition, 2022.

[2] Yifeng Zhu. *Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C*. E-Man Press LLC, 3rd edition, June 2018.