ASSIGNMENT 2 REPORT

Anthony Tang, 103965555, COS10004, Tuesday 4:30PM - Tutor: Jai Cornes

---

1.1) Operating the Timer

Pressing 'P': Stops or starts the timer.
Pressing 'R': Resets the timer.
Pressing 'S' with timer running: Saves split times
Pressing 'S' with timer paused: Displays split times when split times are present.

2.1) Coding

The solution attempts to follows ABI (Application Binary Interface) conventions by following:
R0 - R3 used for functions argument and return values
R4 - R12 are not altered by functions (stacks are used to PUSH and POP when registers are used)
LR AND SP are not altered and only used for stack management

Two forms of interrupts: A clock interrupt and a keyboard interrupt. (Lines 13 - 23)

```
11 ;_____________________________________________INTERRUPT ASSIGNMENTS_____________________________________________
12 ;---------------------TIMER INTERRUPT-------------------------; ;every "interrupt frequency line 14" runs PrintCheck function
13       MOV R0, #PrintCheck ;stores function to be called
14       STR R0, .ClockISR ;stores interrupt function
15       MOV R0, #1000     ;sets interrupt frequency in ms
16       STR R0, .ClockInterruptFrequency ;stores interrupt frequency in ms
17 ;
18 ;----------------------KEYBOARD INTERRUPT----------------------; ;runs KeyInputs when key is pressed - enables interrupts
19       MOV R0, #KeyInputs
20       STR R0, .KeyboardISR
21       MOV R0, #1
22       STR R0, .KeyboardMask
23       STR R0, .InterruptRegister
```

General purpose register: R0 is used as a general purpose registers, that functions to store strings or integers print, used for addressing or storing data in memory.

Three global variables: To store the minutes, seconds and split index. (Lines 8 - 10)

```
 7 ;VARIABLES
 8       MOV R1, #0        ;Seconds
 9       MOV R2, #0        ;Minutes
10       MOV R3, #0        ;Split Number
```

Constants: Containing .ASCIZ strings (Lines 326 - 343)

```
326 ;CONSTANTS
327 center: .ASCIZ "                  "
328 center1: .ASCIZ "                "
329 center2: .ASCIZ "                  "
330 colon: .ASCIZ ":"
331 nextline: .ASCIZ "\n"
332 reset: .ASCIZ "TIMER RESET\n"
333 splittime: .ASCIZ "SPLIT " ;used to confirm split was saved
334 ;
335 displaysplittime1: .ASCIZ "SPLIT 1\n"
336 displaysplittime2: .ASCIZ "SPLIT 2\n"
337 displaysplittime3: .ASCIZ "SPLIT 3\n"
338 displaysplittime4: .ASCIZ "SPLIT 4\n"
339 displaysplittime5: .ASCIZ "SPLIT 5\n"
340 ;
341 splitIndex: .WORD 0     ;used to increment to next split save
342 SavedSplitIndex: .WORD 0 ;used to increment to the next split display
343 NoMoreSplitIndex: .WORD 0 ;used to check if there are anymore saved splits
```

Main loop: The program is constantly looping in the main function Stopwatch, functions are only called when interrupts occur (Line 26 - 27)

```
25 ;_____________________________________________________________MAIN_____________________________
26 Stopwatch:              ;keeps iterating through this loop until an interrupt is called
27       B Stopwatch
```

2.2) Stopwatch Beginnings

To incorporate a display of 'Seconds' (and 'Minutes' in my timer) in the text output, we utilise the Timer Interrupt mechanism. This interrupt is responsible for invoking functions at regular intervals, allowing us to increment the 'Seconds' variable, check if the 'Minutes' variable should be incremented, and subsequently print the updated values of 'Seconds' and 'Minutes'.

Implement the Timer Interrupt:

```
12 ;--------------------TIMER INTERRUPT-------------------------; ;every "
13       MOV R0, #PrintCheck ;stores function to be called
14       STR R0, .ClockISR ;stores interrupt function
15       MOV R0, #1000     ;sets interrupt frequency in ms
16       STR R0, .ClockInterruptFrequency ;stores interrupt frequency in ms
```

Upon program initialization, the Timer interrupt is configured and stored in memory as "ClockISR." The desired frequency of the interrupt is set to 1000ms or 1s, and this value is stored in the memory location labelled as "ClockInterruptFrequency." The purpose of the Timer interrupt is to invoke the "PrintCheck" function every 1s.

Enable interrupts to occur:

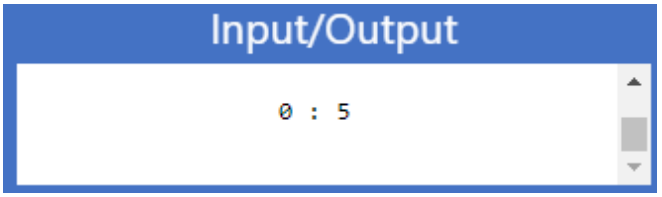
```
21       MOV R0, #1
22       STR R0, .KeyboardMask
23       STR R0, .InterruptRegister
```

1 is stored in the .InterruptRegister and acts to enable interrupts to occur in the program (Line 23)

Implement the functions:

```
30 ;-----------------PRINT FUNCTION-------------------; ;centers and prints the current time
31 PrintCheck:
32       MOV R0, #center
33       STR R0, .WriteString
34       MOV R0, #colon
35       STR R2, .WriteSignedNum
36       STR R0, .WriteString
37       STR R1, .WriteSignedNum
38       MOV R0, #nextline
39       STR R0, .WriteString
40       STR R0, .WriteString
41 SecondsCheck:
42 ;--------------CHECK SECONDS FUNCTION---------------; ;increments the second variable and checks if minutes needs to be incremented
43       ADD R1, R1, #1
44       CMP R1, #59
45       BEQ MinutesCheck
46       RFE
47 ;--------------CHECK MINUTES FUNCTION---------------;   ;increments the minutes variable and reset the seconds variable
48 MinutesCheck:
49       ADD R2, R2, #1
50       MOV R1, #0
51       RFE
```

The PrintCheck function plays a crucial role in displaying the string constants, as well as the global variables representing seconds and minutes. It achieves this by utilising the MOV instruction to move the desired values into register R0. The .WriteString memory location is then employed to print the integers or strings onto the Output display.



Furthermore, after executing the PrintCheck function, the program executes the SecondsCheck function. This function serves the purpose of incrementing the Seconds variable and comparing its current value with 59 using the CMP instruction. In the event that the value equals 59, as determined by the BEQ instruction, the program proceeds to execute the MinutesCheck function. This function increments the Minutes variable by 1 and resets the Seconds variable to 0. However, if the comparison yields a different result, the program returns to the Stopwatch loop using the RFE instruction. This cycle continues indefinitely, even in the presence of Keyboard interrupts, effectively functioning as a persistent timer.

2.3) Stopwatch Basics

A pause feature is accomplished by incorporating keyboard input reading into the program, which triggers specific functions to temporarily halt interrupts and place the timer into a dedicated pause loop. Despite being paused and interrupts being turned off, the program continues to actively monitor user keyboard inputs, enabling the user to resume the timer whenever desired.

By integrating this functionality, we provide users with the ability to control the timer's behaviour, granting them the flexibility to temporarily halt its progress without losing the ongoing monitoring of keyboard inputs. This design ensures that the timer can be easily paused and resumed as needed.

Implement the Keyboard Interrupt:

```
;----------------------KEYBOARD INTERRUPT----------------------;
      MOV R0, #KeyInputs
      STR R0, .KeyboardISR
      MOV R0, #1
      STR R0, .KeyboardMask
      STR R0, .InterruptRegister
```

Similar to the timer interrupt, we store the function KeyInputs into the memory location of .KeyboardISR using our R0 register and MOV and STR instructions. Additionally, we store 1 in the .KeyboardMask and .InterruptRegister memory location which enables the interrupt to function.

By storing the KeyInputs function in memory and appropriately setting the necessary memory locations, we establish the foundation for a responsive and effective keyboard interrupt mechanism. This design allows the program to promptly detect and handle keyboard inputs, and run the KeyInputs function which reads and runs their respective functions based on the key pressed (‘S’,’ R’ or ‘P’).

Read the user's input:

```
KeyInputs:
      LDR R0, .LastKeyAndReset
      CMP R0, #80       ;If "p" is pressed the pause function is called
      BEQ Pause
      CMP R0, #82       ;if "r" is pressed the reset function is called
      BEQ Reset
      CMP R0, #83       ;if "s" is pressed the split function is called
      BEQ SplitDisplay
;
      RFE               ;returns to loop if invalid keys are pressed
```

The KeyInput function retrieves and stores the value of the last key pressed in register R0 using the LDR instruction and the memory location ".LastKeyAndReset". It then compares the retrieved value to the ASCII code for 'P', ‘R’ or ‘S’ (80 in decimal). If the comparison indicates a match, indicating that the last key pressed was 'P' or ‘R’, the program branches to the Pause or Reset function. If invalid keys are pressed the program returns to the Stopwatch loop with the RFE instruction.

Enter pause loop but continue to read user inputs:

```
Pause:
      MOV R0, #0
      STR R0,.InterruptRegister ;Stops the interrupt register
      PUSH {R4, R5, R6}
PauseLoop:
      MOV R7, #0
      MOV R4, #80
      LDR R5, .LastKeyAndReset
      CMP R5, #82       ;Calls reset function if "r" is pressed
      BEQ Reset
      CMP R5, #83       ;if "s" is pressed the savedSplitArray function is called
      BEQ DisplaySplitArrayCheck
      CMP R5, R4        ;Resumes if "p" is pressed
      BNE PauseLoop
Resume:
      MOV R0, #1
      STR R0,.InterruptRegister
      MOV R3, #0        ;Reset the split index to display earliest split time after resuming and pausing again
      POP {R4, R5, R6}
      B Stopwatch
```

In accordance with the ABI conventions, we adhere to the practice of preserving register values by pushing R4-R6 onto the stack.

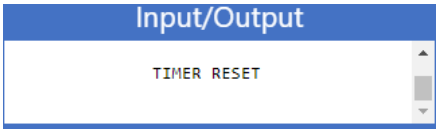
To initiate the pause functionality, it is essential to prevent the occurrence of timer interrupts. This is achieved by storing a value of 0 into the memory location ".InterruptRegister," effectively pausing all interrupts, including the keyboard interrupt. However, while the timer is paused, we still want to monitor and respond to user keyboard inputs. For this purpose, we enter a loop using the PauseLoop function in conjunction with the ".LastKeyandReset" memory location. Within the loop, the program continuously compares the user's keyboard inputs until a valid key is pressed (‘R’, ‘P’ or ‘S’). This is accomplished by using the BNE instruction, which instructs the program to execute the PauseLoop indefinitely until a condition is met.

If the user presses the 'R' key, the program executes the Reset function within the program. If the user presses 'P' again, it breaks out of the PauseLoop, re-enables both interrupts by storing a value of 0 into the ".InterruptRegister," and restores the previously preserved register values by popping R4-R6 from the stack. Finally, the program branches back to the Stopwatch loop to resume normal execution

Implement the Reset function:

```
87 Reset:
88       MOV R1, #0
89       MOV R2, #0
90       MOV R3, #0
91       MOV R0, #nextline
92       STR R0, .WriteString
93       MOV R0, #center1
94       STR R0, .WriteString
95       MOV R0, #reset
96       STR R0, .WriteString
97       MOV R0, #nextline
98       STR R0, .WriteString
99       MOV R0, #0         ;used to reset splits
100 ResetSplits:
101      MOV R4, #splitTimesArr ;move the location of split times array into r4
102      STR R0, [R4]       ;store 0 into minutes
103      ADD R4, R4, #4     ;increments to seconds
104      STR R0, [R4]       ;store 0 into seconds
105      ADD R6, R6, #1     ;each loop will increment r6 by 1
106      CMP R6, #5
107      BNE ResetSplits    ;reset all 5 splits by looping 5 times
108      MOV R6, #0         ;stops ResetSplits function from looping after second calling of reset by resetting R6 back to 0
109      MOV R0, #0
110      STR R0, NoMoreSplitIndex
111      B Pause
```

If the Reset function is invoked during the Keyboard interrupt or within the PauseLoop function, the program takes appropriate actions to reset the timer. This is achieved by utilising the MOV instruction to store the value 0 in the seconds, minutes, and split variables, effectively resetting the timer to its initial state (lines 88 - 90).



To provide feedback to the user, the program proceeds to print the message "TIMER RESET" in the program's output. This is accomplished by storing the necessary string values and utilising the memory location ".WriteString", R0, MOV and STR instructions to print the lines onto the output display.

Furthermore, the program executes the ResetSplits function, which is responsible for resetting the split variables. Which loop through 5 of the split locations in the splits array and NoMoreSplitIndex and sets the values to 0, we discuss this more in depth further in the report. Once the reset operations are complete, the program is directed back to the Pause function, allowing the user to resume or interact with the timer as needed.

2.4) Stopwatch with Benefits and Stopwatch with Buffering

To implement a split function that saves the last 5 split times, displays them in chronological order, and allows for deleting the latest split time, We utilise arrays and three indexes: splitIndex, SavedSplitIndex, and NoMoreSplitIndex. These indexes are initialised in memory in lines 341-356, and play a crucial role in managing the data and controlling the split operations. Through various functions, the program stores the split times in the arrays and updates the indexes accordingly. Additionally, the program calls the relevant operations, and ensures the desired functionality is achieved.

Save Split times when the timer is running:

```
114 SplitDisplay:
115      PUSH {R4, R5, R6}
116      MOV R0, #center
117      STR R0, .WriteString
118      MOV R0, #splittime
119      STR R0, .WriteString
120      MOV R0, #nextline
121      STR R0, .WriteString
122      MOV R0, #center
123      STR R0, .WriteString
124      MOV R0, #colon
125      STR R2, .WriteSignedNum
126      STR R0, .WriteString
127      STR R1, .WriteSignedNum
128      MOV R0, #nextline
129      STR R0, .WriteString
130      STR R0, .WriteString
```

```
132 PushSplitDownArr:      ;push
133      MOV R6, #splitTimesArr
134      LDR R5, [R6, #28]
135      STR R5, [R6, #36]
136      LDR R5, [R6, #24]
137      STR R5, [R6, #32]
138      LDR R5, [R6, #20]
139      STR R5, [R6, #28]
140      LDR R5, [R6, #16]
141      STR R5, [R6, #24]
142      LDR R5, [R6, #12]
143      STR R5, [R6, #20]
144      LDR R5, [R6, #8]
145      STR R5, [R6, #16]
146      LDR R5, [R6, #4]
147      STR R5, [R6, #12]
148      LDR R5, [R6, #0]
149      STR R5, [R6, #8]
```

```
151 SaveSplit:
152      MOV R0, #splitTimesArr ;Lo
153      STR R2, [R0]       ; Store
154      STR R1, [R0, #4]   ; Store
155 UpdateIndexes:
156      LDR R0, NoMoreSplitIndex ;
157      ADD R0, R0, #1
158      STR R0, NoMoreSplitIndex
159      POP {R5, R6}
160      RFE
```

When the user presses 'S' while the timer is running, the Keyboard interrupt is triggered, which then executes the KeyInputs function. This function reads the last key pressed, stores it in R0, and compares it to the ASCII code for 'S' (83 in decimal) using the CMP instruction. If the comparison indicates a match, the SplitDisplay function is called (lines 113-129). Within the SplitDisplay function, {R4} is pushed onto the stack to preserve its value. By utilising the MOV, STR, and the ".WriteString" memory location, the program provides feedback to the user, indicating that a split time has been saved.

The PushSplitDownArr function calls and acts to push the earliest split time starting from the latest down the splitTimeArr which contains the saved times using the instructions LDR and STR. By moving the memory location of the array into R6, we then address each location of the array with the corresponding bit location. Each minute and seconds is 4 bits, meaning there is a gap of 8 bits in

between each save location we increment by. The SaveSplit function is then run which accesses the first two locations of the array and stores the earliest split

The UpdateIndexes function is then invoked, NoMoreSplitIndex is incremented using R0 and stored back into memory, the index is used to ensure only saved split times are displayed and ensures the display split functions loop back to the first one once all saved splits have been displayed.

Implement splits display:

```
DisplaySplitArrayCheck:
      LDR R0, NoMoreSplitIndex
;
      CMP R7, #3        ;used t
      BEQ PauseLoop
      CMP R0, #0
      BEQ ResetSavedSplitIndex
      CMP R3, #0
      BEQ DisplaySplitArray1
      CMP R0, #1
      BEQ ResetSavedSplitIndex
      CMP R3, #1
      BEQ DisplaySplitArray2
      CMP R0, #2
      BEQ ResetSavedSplitIndex
      CMP R3, #2
      BEQ DisplaySplitArray3
      CMP R0, #3
      BEQ ResetSavedSplitIndex
      CMP R3, #3
      BEQ DisplaySplitArray4
      CMP R0, #4
      BEQ ResetSavedSplitIndex
      CMP R3, #4
      BEQ DisplaySplitArray5
      RFE
ResetSavedSplitIndex:
      MOV R3, #0
      ADD R7, R7, #1
      B DisplaySplitArrayCheck
```

```
DisplaySplitArray1:
      PUSH {R4, R5, R6, R7} ;R4
      ADD R3, R3, #1    ;moves
      MOV R6, #8        ;set up
      MOV R0, #SavedSplitIndex
      STR R6, [R0]      ;save s
      MOV R4, #4        ;access
      MOV R5, #splitTimesArr
;
      MOV R0, #nextline
      STR R0, .WriteString
      MOV R0, #center2
      STR R0, .WriteString
      MOV R0, #displaysplittime
      STR R0, .WriteString
      MOV R0, #center
      STR R0, .WriteString
;
      LDR R0, [R5]      ;load s
      STR R0, .WriteSignedNum ;
      MOV R0, #colon
      STR R0, .WriteString
      LDR R0, [R5, R4]  ;load s
      STR R0, .WriteSignedNum ;
;
      MOV R0, #nextline
      STR R0, .WriteString
      STR R0, .WriteString
      POP {R4, R5, R6}
      B PauseLoop
```

```
DisplaySplitArray2:
      PUSH {R4, R5, R6, R7} ;R4 =
      ADD R3, R3, #1    ;moves on
      LDR R7, SavedSplitIndex ;Loa
      ADD R4, R7, #4    ;access th
      MOV R6, #16       ;display t
      MOV R0, #SavedSplitIndex
      STR R6, [R0]
      MOV R5, #splitTimesArr
;
      MOV R0, #center2
      STR R0, .WriteString
      MOV R0, #displaysplittime2
      STR R0, .WriteString
      MOV R0, #center
      STR R0, .WriteString
;
      LDR R0, [R5, R7]  ;load spli
      STR R0, .WriteSignedNum ;pri
      MOV R0, #colon
      STR R0, .WriteString
      LDR R0, [R5, R4]  ;load spli
      STR R0, .WriteSignedNum ;pri
      MOV R0, #nextline
      STR R0, .WriteString
      STR R0, .WriteString
      POP {R4, R5, R6, R7}
      B PauseLoop
```

When the 'S' key is pressed while the timer is paused, the KeyInput function triggers the DisplaySplitArrayCheck function, which is responsible for displaying the split times stored in the splitTimeArr array. This function utilises the split number (R3) and the NoMoreSplitIndex to determine whether there are splits to be displayed.

The DisplaySplitArrayCheck function effectively manages the split display process. By iterating through each split number, it calls the DisplaySplitArray function (1-5) to print the corresponding split time. Each DisplaySplitArray function operates similarly, using indexed memory addressing to access the appropriate memory locations in the split array. For example, DisplaySplitArray(1) displays the first split location, DisplaySplitArray(2) displays the second split location, and so on. Additionally, the function increments R3 to allow for the display of subsequent splits if the 'S' key is pressed again.

To ensure that only saved splits are displayed, the NoMoreSplitIndex is incremented by 1 every time the split button is pressed while the timer is running. This value is saved in memory and then loaded by the DisplaySplitArrayCheck function. The function compares the NoMoreSplitIndex value to the split number (R3) minus 1. If these values match, indicating that all the saved splits have been displayed, the program executes the ResetSavedSplitIndex function. This function resets the value of the split index (R3) back to 0, allowing for the looping display of the saved splits. For example, if there are 3 saved splits and the NoMoreSplitIndex holds a value of 3, the function compares the value at the fourth split display to 3 and triggers the reset to display the first split again. However, if the user saves 4 splits, the NoMoreSplitIndex will be 4, and the ResetSavedSplitIndex function will only run when attempting to display the fifth split.

3.1) Issues and Solutions

Upon pressing ‘S’ after resetting the DisplaySplitArrayCheck enters a loop as split number (R3) is equal to 0:
The **solution** I implemented was to have each loop iteration increment R7 and have the function compare the number to #3, if R7 is equal to 3 meaning it looped 3 times, the CMP instruction will call the PauseLoop putting the program back into pause and allowing the user to operate the timer again.

One function for all displays: There may have been a more efficient way to code the printing of the splits for the bonus 5% task (Stopwatch with Buffering, and that would be to only have one function which takes in different parameters each time the ‘S’ key is pressed. A possible solution may have been to increment each memory location value for the seconds and minutes by 4 and 8 each time the ‘S’ key was pressed and only having one string that printed out “SPLIT TIME” and another incrementing integer printing “1 - 5” according to the split time.

Saving a split right after the timer increments will cause it to increment again: I’ve concluded that this is an ARMlite simulator issue with the KeyboardInterrupt resetting the count for the clock frequency, which has the timer interrupt run the PrintCheck function as soon as the keyboard interrupt is finished which increments the timer.