COS10004 2023

Assignment 2 Report

10026
Lee Chi Kit Aiken

I’ve completed all the stages except for Stage 4, Stopwatch with Buffering. And follow the ABI conventions.

Instructions

A is for stop, S is for split, R is for Reset, You can't split or reset when you are in the stop state.
The running time is in the middle, while the split time is below it, indicated with an S on the side.

Stage 1

For stage 1, I’m using .WriteUnsignedNum to display the second increment, The second is represented by r4-r7, which will always represent the clock and nothing else. Originally, I use .Time to do a dumb wait timer, checking the time nonstop, and adding 1 to the second when it’s already 1 second. This is later changed to interrupt, sending a clock interrupt every time a second passes.

```
28|      beq addSecondDigit
29|timeOutput:              /output clock and draw the clock
30|      str r7, .WriteUnsignedNum
31|      str r6, .WriteUnsignedNum
32|      str r5, .WriteUnsignedNum
33|      str r4, .WriteUnsignedNum
```

Stage 2

For stage 2, my original idea is the pause the timer with an .ReadString, you can see it on earlier versions. So it will loop and check the Last key you pressed with a .LastKeyAndReset, stoping the clock with .ReadString after you pressed A(A for pause), and resume if you type on the input box. But I changed it to going into another loop and disabling the clock interrupt instead. On this version, it’s checking the input with .LastKeyAndReset, when the last key that the user presses is A, the stopwatch will resume back to where it stoped. Also, the LastKey in earlier version is actually looping nonstop, but I changed it into a Keyboard interrupt. It will only check what your input is after you pressed a button on your keyboard. For the reset button, I’m also putting it into the check input and do a second CMP to see if that button is R(R for reset). And setting everything to 0 once it’s R.

```
checkinput:
      ldr r12, .LastKeyAndReset
      cmp r12, #83
      beq splitTime
      cmp r12, #65      /keyboard A for stopping
      beq stop
      cmp r12, #82      /keyboard R for resetting
      beq resetTime
```

```
resetTime:
      mov r1, #reseted
      str r1, .WriteString
      b start
```

```
reset:                  /this is the four digit of the clock
      mov r4, #0
      mov r5, #0
      mov r6, #0
      mov r7, #0
```

```
stop:
      mov r1, #stopped
      str r1, .WriteString
      MOV R8,#loopwait
      STR R8,.ClockISR
loopwait:               /this is a loop waiting for stop
      ldr r12, .LastKeyAndReset
      cmp r12, #65
      beq returnfrominput
      b loopwait
```

Stage 3

For stage 3, I'm adding an array to store the information of split time. And I'm also checking the input for the key S. When the user presses S on the keyboard, the stopwatch will store the current time, which is r4-r7, to the array. When reset is pressed, it will also reset the split time to all 0.

```
splitTime:
      PUSH {r0}
      mov r0, #0
      str r7, [r10+r0]
      add r0, r0, #4
      str r6, [r10+r0]
      add r0, r0, #4
      str r5, [r10+r0]
      add r0, r0, #4
      str r4, [r10+r0]
      POP {r0}
      mov r3, #2264     /offset for drawing pixel for running clock
      bl drawTime
      b loop
```

Stage 4

For stage 4, I do have the whole idea on how to build it, but I do not have enough time to figure it out, so I leave it. My idea is to spread each digit to an array, each being an array of 5, and having total of 4 arrays. Each time you press the split button, all of the old array item will pass down to its current address plus #4, which is the address of the next item in the array, and the array number 5 will be discarded. And then the new split time will be store at array number 1. And displaying it to the output using the time interrupt, so everytime the time interrupt is here, the stopwatch not only showing the new current time, but also show the next split time in the array, tracking by a counter, everytime the interrupt comes, the counter will increase by #4 (indirect addressing), and then once it hits #16, it will reset back to #0, so it loops through the split time on the display. Finally, when the reset is hit, reset all of the array back to 0, including the counter too.

Stage 5

For stage 5, it's hard to showcase everything on it. But I will talk about some important features. First of all, I use 10 function to draw the number on the screen, which is draw0 - draw 9. It is draw through indirect addressing, so even though we have 8 number in total, we don't need 80 function to draw it in the pixel output. We just need to pass the offset for the first bit in that number, and it will draw that number with offset. Also, in the start of the clock, I also draw the base of the clock, which includes the S(indicator for split time) and the :, which makes it easier to read. The draw number only activate when it receives the interrupt from the clock. Preventing the stopwatch from drawing every single loop(which cost a lot of computer power to do so.). Also, I've used a lot of PUSH and POP to have "more register" to work with, but I only do it to r0-r3 and lr(for redirecting to different places). Every time the stop watch draws something in the pixel screen, it will first erase the pixel, so the next number would not overlap and make it unreadable.

```
464|       ret
465|erasePixel:              /this is for erase the pixel drawn
466|       str r1, [r2+r3]
467|       add r3, r3, #4
468|       str r1, [r2+r3]
469|       add r3, r3, #4
470|       str r1, [r2+r3]
471|       add r3, r3, #120
472|       str r1, [r2+r3]
473|       add r3, r3, #4
474|       str r1, [r2+r3]
475|       add r3, r3, #4
476|       str r1, [r2+r3]
477|       add r3, r3, #120
478|       str r1, [r2+r3]
479|       add r3, r3, #4
480|       str r1, [r2+r3]
481|       add r3, r3, #4
482|       str r1, [r2+r3]
483|       add r3, r3, #120
484|       str r1, [r2+r3]
485|       add r3, r3, #4
486|       str r1, [r2+r3]
487|       add r3, r3, #4
488|       str r1, [r2+r3]
489|       add r3, r3, #120
490|       str r1, [r2+r3]
491|       add r3, r3, #4
492|       str r1, [r2+r3]
493|       add r3, r3, #4
494|       str r1, [r2+r3]
495|       ret
```

```
407|draw8:
408|      PUSH {lr, r3}     /save lr and the offset of the pixel first
409|      bl erasePixel
410|      POP {lr, r3}      /change the lr and offset back so that the draw won't be effected
411|      PUSH {r3}
412|      str r0, [r2+r3]
413|      add r3, r3, #4
414|      str r0, [r2+r3]
415|      add r3, r3, #4
416|      str r0, [r2+r3]
417|      add r3, r3, #120
418|      str r0, [r2+r3]
419|      add r3, r3, #8
420|      str r0, [r2+r3]
421|      add r3, r3, #120
422|      str r0, [r2+r3]
423|      add r3, r3, #4
424|      str r0, [r2+r3]
425|      add r3, r3, #4
426|      str r0, [r2+r3]
427|      add r3, r3, #120
428|      str r0, [r2+r3]
429|      add r3, r3, #8
430|      str r0, [r2+r3]
431|      add r3, r3, #120
432|      str r0, [r2+r3]
433|      add r3, r3, #4
434|      str r0, [r2+r3]
435|      add r3, r3, #4
436|      str r0, [r2+r3]
437|      POP {r3}
438|      ret
```

```
 46|drawBase:               /this draws the base of the first second(00:00)
 47|       PUSH {lr}
 48|       mov r0, #.black
 49|       str r0, .Pixel336
 50|       str r0, .Pixel400
 51|       str r0, .Pixel514
 52|       str r0, .Pixel545
 53|       str r0, .Pixel547
 54|       str r0, .Pixel577
 55|       str r0, .Pixel610
 56|       str r0, .Pixel643
 57|       str r0, .Pixel673
 58|       str r0, .Pixel675
 59|       str r0, .Pixel706
 60|       str r0, .Pixel592
 61|       str r0, .Pixel656
 62|       mov r0, #.black
 63|       mov r1, #.white
 64|       mov r2, #.Pixel0
 65|       mov r3, #1240
 66|       bl draw0
 67|       mov r3, #1224
 68|       bl draw0
 69|       mov r3, #1200
 70|       bl draw0
 71|       mov r3, #1184
 72|       bl draw0
 73|       mov r3, #2264
 74|       bl draw0
 75|       mov r3, #2248
 76|       bl draw0
 77|       mov r3, #2224
 78|       bl draw0
 79|       mov r3, #2208
 80|       bl draw0
```

Highlights and difficult part

The highlight of this stopwatch is definitely on the main loop, since both keyboard input and clock are in the interrupt. I can manage to create the main loop with nothing on it.

```
loop:                   /loop the timer if it isn't 1 second yet
      b loop
```

The difficult part of this clock is definitely with pixel output, it is very difficult to update the stopwatch feature without messing up the pixel output, one of the mistake that I've make is building the pixel output in early stages, making me very difficult to improve the stopwatch with new feature later on.

This assignment is much harder than the assignment 1, but by building the assignment 1 first. It helps me to build the basic structure of the stopwatch.