Name: Thanh Tam Vo
StudentID: 103487596

COS10004 - Computer Systems – Assignment 2

Introduction

This assignment 2 of the Unit COS10004 – Computer Systems aims to program assembly language and use raspberry pi to flash the LED

Content

Stage 1A: ………………………………………………..………………………page 2
Stage 1B: ………………………………………………...…………………...page 3
Stage 1C: ………………………………………………..…………………page 4
Stage 2: …………………………………………………….………...……page 6
Stage 3: ……………………………………….……….……………………page 8
Stage 4: ……………………………………….…………………….....…page 11
Demonstration: ………………………………………………..…………….page 11

Stage 1A

The requirement is writing a function that returns the minimum value among three values.
Before writing the function in assembly language, I would like to give my idea based on the following pseudocode:

```
MinValue(int value1, int value2, int value3):
     int result = value1
     If (result >= value2):
          result = value2
     If (result >= value2):
          result = value3
     return value
```

Now, this is my final implementation for Stage 1A

```
stage1a_min:
        ; implement your function here
        ; remember to push any registers you use to the stack before you use them
        ; ( and pop them off at the very end)

        push { lr }
        ;compare r0 and r1, if r0 is larger or equal to r1, assign r1 to r0
            cmp r0, r1
            movge r0, r1

        ;compare r0 and r2, if r0 is larger or equal to r2, assign r2 to r0
            cmp r0, r2
            movge r0, r2
        pop { lr }

         bx lr
```

Figure 1. Stage 1A in assembly language
(Screen captured from my Mac on November 27th, 2022)

Explanation:

r0, r1, and r2 are considered as parameters of the function *stage1a_min.* Because of the requirements, there are not any registers to store the result value but *r0*, so after comparing *r0* and *r1*, if *r0* is larger or equal to *r1*, then *r0* will store the value of *r1*. Doing so for the comparison between *r0* and *r2*, we finally have the minimum value that is stored at *r0*.

Stage 1B

The requirement is writing a function that returns the maximum value among three values.
Before writing the function in assembly language, I would like to give my idea based on the following pseudocode:

```
MaxValue(int value1, int value2, int value3):
     int result = value1
     If (result <= value2):
          result = value2
     If (result <= value2):
          result = value3
     return value
```

Now, this is my final implementation for Stage 1B

```
stage1b_max:
        ;compare r0 and r1, if r0 is less or equal to r1, assign r1 to r0
        cmp r0, r1
        movle r0, r1

        ;compare r0 and r2, if r0 is less or equal to r2, assign r1 to r0
        cmp r0, r2
        movle r0, r2

        bx lr
```

Figure 1b. Stage 1B in assembly language
(Screen captured from my Mac on November 27th, 2022)

Explanation:

r0, r1, and r2 are considered as parameters of the function *stage1b_max.* Because of the requirements, there are not any registers to store the result value but *r0*, so after comparing *r0* and *r1*, if *r0* is less or equal to *r1*, then *r0* will store the value of *r1*. Doing so for the comparison between *r0* and *r2*, we finally have the maximum value that is stored at *r0*.

Stage 1C

The requirement is writing a function that returns the difference between the max and the min value out of three values.
Before writing the function in assembly language, I would like to give my idea based on the following pseudocode:

```
DiffMinMax(int value1, int value2, int value3):
      int minValue = MinValue(int value1, int value2, int value3)
      int maxValue = MaxValue(int value1, int value2, int value3)
      int result = maxValue - minValue
      return result
```

Now, this is my final implementation for Stage 1C

```
stage1c_diff:
        ;get the minimum value
        push { r3, r4 }
        push { r0, lr }
        bl stage1a_min

        ;r3 is the register storing the minimum value
        mov r3, r0
        pop { r0, lr}


        push { lr }
        bl stage1b_max
          pop { lr }
        ;r4 is the register storing the max value
        mov r4, r0

        ;storing different value between min and max to r0
        sub r4, r3
        mov r0, r4
        pop { r3, r4 }

bx lr
```

Figure 1c. Stage 1C in assembly language
(Screen captured from my Mac on November 27th, 2022)

Explanation:

r0, *r1*, and *r2* are considered as parameters of the function ***stage1c_diff***. I decided to use *r3*, and *r4* for storing the minimum value and maximum value, respectively. However, due to the rule of assembly language, we can not change the value of *r3* and *r4* after calling a function so I pushed *r3* and *r4* in the stack and popped them out of the stack in order to keep their value after calling the function ***stage1c_diff***.

Before calling the function ***stage1a_min***, I pushed *r0* and *linked register* to the stack (line 12) to keep the original value of r0 (r0 will be updated when the function *stage1a_min* was called)

When the function ***stage1a_min*** has been completely performed, *r0* now is holding the minimum value among three values. Then, with the mov instruction, *r3* will hold the minimum. I popped *r0* and *linked register* in line 17 so *r0* will hold the original value before it was updated in the **stage1a_min** function.

Now, three original values stored at *r0, r1, and r2* are ready to be found out the maximum value.

Doing so (push and pop linked register) when calling **stage1b_max** and storing the maximum value to r4.

Then the code performed the *sub-instruction* by subtracting *r4* by *r3* to get the difference. Then we store the value of *r4* to *r0* to meet the requirements of the assignment.

Stage 2

The requirement is writing a function that flash the given array

Before writing the function in assembly language, I would like to give my idea based on the following pseudocode:

```
FlashArray(address,size,array):
      //array parameter is the address of the array to flash
      for (int i = 0; i < size; i++):
            //get the address of the current element
            temp = array + 4
            r1 = temp
            r2 = delay time between flashes
            Call FLASH
            r1 = delay time before moving to the next element of the
array
            Call PAUSE
```

Now, this is my final implementation for Stage 2

```
stage2_flash_array:
        ;i used r3 for the index keep tracking ;;
        ;i used r4 for the value keep tracking ;
        ;i used r5 for the time delay between flashes
        push { r0, r3, r2, r4 , r5}
        mov r5, $50000
        mov r3, #1
        loop:
                    ldr r4, [r2] , #4
                push { lr , r1, r2 }

                mov r1, r4
                mov r2, r5
                bl FLASH
                ;pause before increase the index
                    mov r1,$150000 ; pause time
                    bl PAUSE

                pop { lr , r1, r2 }
                add r3, r3, #1
                cmp r3, r1
        ble loop
        pop { r0, r3, r2, r4, r5 }
        bx lr
```

Figure 1c. Stage 2 in assembly language
(Screen captured from my Mac on November 27th, 2022)

Explanation:

r3: index keep tracking (can be considered as `i`)
r4: value keep tracking (can be viewed as the value at `Array[ i ])`
r5 : delaying time between flashes

We need to get the value of the current element before passing it to *r1* and call the **FLASH** function.

Note: **FLASH** function takes *r1* as the number of flashes

Explaining line 17:

Because the data type of every element in the array is integer, so it takes 4 bytes for each element.

Therefore, the address of current element is *address of the array + 4*
To get that value, we use “[ ]” and use the ***ldr*** command to load that value to *r4*
Now, r4 is holding the number of flashes.
From line 22, the implementation of **FLASH** and **PAUSE** will not be explained here since it was covered in the previous section

Stage 3

The requirement is writing a function that sort the given array using bubble sort method and show the sorted result via Raspberry Pi
Before writing the function in assembly language, I would like to give my idea based on the following pseudocode:

```
BubbleSort(Array):
      sizeArray is the size of Array
      for (i = 0; i < sizeArray; i++):
            for (j = i; j < sizeArray - i - 1; i++):
                  if Array[ j ] >= Array[ j + 1]:
                        swap Array[ j ] and Array[ j + 1]
```

Now, this is my final implementation for Stage 3

```
stage3_bubblesort:
            push { lr, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10 }

            mov r7, #0    ;;i start from 0
            mov r8, #0      ;;j start from 0
            mov r9, r0    ;;r0 is the size of the array
            mov r2, #1
            loopBubble:
                  loopBubble1:
                        ;assigning r10
                        push { r7, r9 }
                                sub r9, r9, r7
                                sub r9, r9, r2
                                mov r10, r9
                        pop { r7, r9 }

                        ;;;load r5, r6
                        push { r7, r8, r9 }
                        mov r7, r8
                        mov r9, #4
                        mul r8, r7, r9  ;;;increase r8
                        push { r1 }
                        add r1, r1, r8
                        ldr r5, [r1]
                        pop { r1 }
                        mov r3, r8
                        add r8, r8, r9          ;;add 4 to r8
                        push { r1 }
                        add r1, r1, r8
                        ldr r6, [r1]
                        pop { r1 }

                        mov r4, r8
                        pop { r7, r8, r9 }
```

Figure 3.1. Stage 3 in assembly language
(Screen captured from my Mac on November 27th, 2022)

```
42
43                      ;;;;;;;swap value here
44                      cmp r5, r6
45                      strge r6, [r1, r3]  ;;storing r6 to the address of array[ i ] if r5 is greater or equal to r6
46                      strge r5, [r1, r4]  ;;storing r5 to the address of array[ i  + 1 ] if r5 is greater or equal to r6
47
48                      add r8, r8, #1
49          cmp r8, r10
50              blt loopBubble1                 ;if r8 is LESS than r10, then continue loop1
51
52              add r7, r7, #1
53              cmp r7, r9
54              movlt r8, #0
55      blt loopBubble                          ;if r7 is LESS than r9, then continue loop
56        mov r0, r1
57
58        ;flash sorted array
59        push { lr, r0, r1, r2}
60              BASE = $FE000000
61              mov r1, r9
62              mov r2, r0  ;;array to flash
63              mov r0, BASE
64              bl stage2_flash_array
65        pop { lr, r0, r1, r2}
66        pop { lr, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10 }
67      bx lr
```

Figure 3.2. Stage 3 in assembly language
(Screen captured from my Mac on November 27th, 2022)

Explanation:

r5 is the register that store the value at `Array[ j ]`
r3 is the register storing the needed value to go to `Array[ i ]`
r6 is the register that store the value at `Array[ j + 1 ]`
r4 is the register storing the value at `Array[ i + 1 ]`
r7 acts like `i`
r8 acts like `j`
r9 acts like `sizeArray`
r10 acts like "`sizeArray - i - 1`"

I will give the implementation of swapping the first element and second element

From the line 44,
r3 is holding the **needed number** to go to the **address** of `Array[ i ]`
r5 is holding the **value** of `Array[ i ]`
r4 is holding the **needed number** to go to the **address** of `Array[ i + 1 ]`
r6 is holding the **value** of `Array[ i + 1]`

```
;;;;;;;swap value here
cmp r5, r6
strge r6, [r1, r3]  ;;storing r6 to the address of array[ i ] if r5 is greater or equal to r6
strge r5, [r1, r4]  ;;storing r5 to the address of array[ i  + 1 ] if r5 is greater or equal to r6
```

Compare *r5* and *r6*, if *r5* is greater than *r6* then

`strge  r6, [r1, r3]` : go to the address of `Array[ i ]` and take the value of *r6* and put it in there.

`strge  r5, [r1, r4]` : go to the address of `Array[ i + 1]` and take the value of *r5* and put it in there.

Now we have the value at `Array[ i ]` is less than the value at `Array[ i + 1 ]`

Stage 4

I decided to ignore this part due to my time management skill

Demonstration

Video

Video link here

RPI version

The Raspberry Pi in the video is the RPI 4. If you want to apply my code to RP2 or RP3, then

Firstly, in file ***kernel.asm**,* please modify the BASE address at line 10 and 11

```
10    BASE=$FE000000 ; RPI 4 Peripherals address ;
11    ;BASE=$3F000000 for RP2 and RP3 Peripherals address
```

Secondly, in file ***stage3_bubblesort.asm***, please modify the BASE address at line 79 and 80

```
79                        BASE = $FE000000 ; RPI 4 Peripherals address ;
80                        ;BASE=$3F000000 for RP2 and RP3 Peripherals address
```