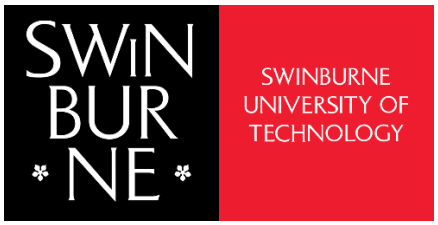
Swinburne University of Technology



COS10004 – COMPUTER SYSTEM

Semester: SEP 2022

Assignment 2:

ARM Assembly Programming

Huynh Thanh Thy 103805499

Class: COS10004

Lecturer: Dr. Minh Quang Nguyen

Link video: [Assignment 2 - YouTube]

Stage 1A

First, comparting the first two values (register r0 and r1)

```
cmp r0, r1; compare values in register r0 and r1
bgt set_min1; if r0 > r1, go to set_min1
      ; if r0 < r1 then continue to compare r0 and r2
```

Comparting the smaller value in the above comparation which will store in r0 and the third value (r2). If r0 is smaller than r2, , then r0 is the smallest, go to end function. Otherwise, branch to set_min2 in order to store the smallest value (r2) to r0.

```
compare1:
        cmp r0,r2; compare values in register r0 and r2
        bgt set_min2; if r0 > r2, go to set_min2
                    ; if r0 < r2; continue go to final
```

Moving smaller value r1 into r0, then branch to compare1 to continue compare with the third value

```
set_min1:
        mov r0, r1   ; mov value in r1 into r0
        b compare1   ;then continue to compare r0 (r1) and r2
```

In compare1 function, if r2 is smaller r0, move value in r2 to r0 then branch to final

```
set_min2:
        mov r0, r2    ; mov value in r2 into r0
        b final
```

Stage 1B

First, comparting the first two values (register r0 and r1)

```
cmp r0, r1; compare values in register r0 and r1
blt set_max1; if r0 < r1, go to set_max1
              ; if r0 > r1; continue go to compare2
```

Comparting the larger value in the above comparation (store in r0) and the third value (r2). If r0 is larger than r2, then r0 is the largest, go to end function. Otherwise, branch to set_max2 in order to store the largest value (r2) to r0.

```
compare2:
        cmp r0,r2; compare values in register r0 and r2
        blt set_max2; if r0 < r2, go to set_max2
                    ; if r0 > r2; continue go to final2
```

Moving larger value r1 into r0, then branch to compare2 to continue compare with the third value

```
set_max1:
        mov r0, r1; mov value in r1 into r0
        b compare2; continue compare2
```

In compare1 function, if r2 is larger than r0, move value in r2 to r0 then branch to final

```
set_max2:
        mov r0, r2  ; mov value in r2 into r0
        b final2    ; go to final2
```

Stage 1C

```
stage1c_diff:
        push{lr}
        bl stage1a_min     ;load min value from stage1a_min
        mov r1,r0

        push{r1}
        mov r0,r5
        mov r1,r6
        mov r2,r7

        bl stage1b_max     ;load max value from stage1a_min
        pop{r1}

        sub r0,r0,r1     ; subtract min from max to find difference
        pop{lr}

        bx lr
```

Load min value from stage1a_min which store in r0, then move it to r1

Load max value from stage1b_max which store in r0

Subtract min value (r1) from max value (r0) then store in r0

Stage 2

First, define counter (r3), the byte of array (r4), the address of first element of the array (r6). Loop through the array, load each element into r5. If counter r3 equals the size of array (r4), end loop. Otherwise, continue looping loop_stage2.

```
mov r3, #0            ;    r3 = loop counter (index)
lsl r4,r1,#2   ; r4=32
mov r6, r2
```

```
ldr   r5, [r6, r3]       ;r5 = element at address r2 + r3   ( numarray1[i])
```

```
add r3, r3, #4          ;i++
cmp r3, r4              ;compare counter r3 to size of array in r1
beq end_stage2
b loop_stage2     ;jump to loop_stage2
```

Function to flash array. First moving the value of array in r1, then call function flash to flash that value. After that call function pause to pause between the values in array

```
 ; prepare to flash answer
mov r1,r5        ; value to flash
mov r2,$30000  ; pause time between flashes r10
 push{lr}  ; store current lr before it is overwritten during function call
 bl FLASH   ; call the FLASH function
 pop{lr} ; restore old lr value

mov r1,$200000 ; pause time
push {lr}
bl PAUSE
pop {lr}
```

Stage 3

Define the counter for 2 loops

```
next:
        mov r7, #0   ;r7=i
        mov r6, #0   ;r6=j
```

Compare counter and the size of array

```
loop:
        add r3,r7,#1     ; r3= i+1
        cmp r3,r0        ; compare counter and length of array
        bge check_loop   ;if   r3>r0, go to check_loop

        ;Calculate address of element by adding offset to base address
        ;load first value to r4
        ldr r4, [r1, r7, lsl #2]   ;r4 = element at address r1 + r7 + #4   ( numarray1[i])
        ;load second value to r5
        ldr r5, [r1, r3, lsl #2]   ;r5 = element at address r1 + r3 + #4   ( numarray1[i+1])

        cmp r4,r5 ; compare first value (r4) and second value (r5)
        bgt swap  ; if r4>r5, branch to swap

        mov r7,r3  ; move value in r3 (i) to r7 (i+1)     ;i++
        b loop ;branch to loop
```

Load current value to r4

Load next value to r5

Compare the values in r4 and r5. If r5 < r4, branch to swap, otherwise continue the loop

Increase the counter by moving the r3 (i+1) into r7 (i)

Swap the value then go back to loop function. Increase the counter by moving the r3 (i+1) into r7 (i)

```
swap: ;swap the value in r5 and r4
        str r5, [r1, r7, lsl #2]
        str r4, [r1, r3, lsl #2]

        add r6,r6,#1 ; j++

        mov r7,r3 ; move value in r3 (i) to r7 (i+1)     ;i++
        b loop  ; branch to loop
```

Check outer loop:

```
check_loop:
        cmp r6,#0     ; compare counter and 0
        subgt r0,r0,#1   ;if j counter < length (r6<r0) then r0=r0-1 (size array - 1)
        bgt next   ;if r6>0, branch to next
```

Call file stage2_flash_array in order to flash the sorted array after load value which are needed into it.

```
; load value into stage 2
mov r0,r2
adr r2,numarray1
mov r1,r8

push{lr}
bl stage2_flash_array
pop{lr}
```