Programming a Given Computer System

Ghassan Arnouk

SYSC 3006A
Summer 2020
Lab 5 Report
Group 1

Instructor: Michel Sayde
TA: Khalid Almahrog
Submitted: 2020/06/11

1 Fragment 1

1.1 What is the high-level objective (purpose) of the code fragment? Explain the objective in terms of the net effect of the fragment on the variables it modifies.

The high-level objective of the code fragment is to:

- Create an array of size three
- Fill the array with the integers 20, -4, and 0
- Check that the array size is correct
- And finally, add the number 10 to each integer in the array

1.2 Write C-like pseudocode that accomplishes the same objective (see lab statement for more details).

```
int Arr_Size = 3;
int Arr[Arr_Size] = {20, -4, 0};
int R2, R3, R5;

R2 = *Arr[0];
R3 = Arr_Size;

if (R3 == 0){
  end;
}

R3 = R3-1;

while (R3 >= 0){
  R5 = Arr[R3];
  R5 = R5+10;
  Arr[R3] = R5;
  R3 = R3-1;
}
```

1.3 Starting after the SkipOverVariables declaration, add comments to the instructions that document what is being done ... the comments should be at the level of the pseudocode objective, not at the RTL level.

```
1  B SkipOverVariables
2
3    ; Arr is an array of 3 words
4  Arr_Size DCD #3
5  Arr
6    DCD #20 ; first (0 - th)element of Arr = 20
7    DCD #-4 ; second (1 - th)element of Arr = -4
8    DCD #0  ; third (2 - th)element of Arr = 0
9
10 SkipOverVariables
11 MOV R2, Arr            ; // R2 is initialized to the address head of the array
12 LDR R3, [ Arr_Size ]   ; // Loading the arr_size (3) and storing it in R3
13 CMP R3, #0             ; // check if the content of R3 is equal to zero
14 BEQ Done               ; // if the content of R3 is zero, branch is done
15 SUB R3, R3, #1         ; // subtract 1 from the content of R3 and then store it in R3
16
17 Loop                   ; // label
18 LDR R5, [R2, R3]       ; // adding the content of R2 and R3 which makes an address which
                              will the content of R5 be stored in
19 ADD R5, R5, #10        ; // adding 10 to the content of R5
20 STR R5, [R2, R3]       ; // storing the content of R5 in the address of the sum of
                              addresses of R2 and R3 which is R3
21 SUB R3, R3, #1         ; // subtract 1 from the content of R3 and then store it in R3
22 BPL Loop               ; // if the negative is flagged in the ALU, the loop will be broken
23
24 Done
25 DCD #0xFFFFFFFF        ; breakpoint instruction
```

1.4 When the fragment is executed, how many instructions will be executed (including the breakpoint instruction)?

Branch to SkipOverVariables: 1
SkipOverVariables: 5
Loop: $5 + 5 + 5 = 15$
Done: 0 instructions for breakpoint (fetched, but never executed)
Total $= 1 + 5 + 15 + 0 = 21$ instructions

1.5 When assembled, how many words of memory will the fragment occupy?

Total = 16 words of main memory will be needed for fragment1

1.6 Assemble and run Fragment 1. To validate running the fragment in your lab report, submit the contents of Main Memory RAM before and after executing the fragment. (Hint: right-click on RAM Save Image ...).

Before execution:

```
v2.0 raw
80F00004 00000003 00000014 FFFFFFFC 00000000 23200002
333FFFFA 57300000 80100006 22330001 32523000 2155000A
36523000 22330001 806FFFFB FFFFFFFF
```

After execution:

```
v2.0 raw
80f00004 3 1e 6 a 23200002
333ffffa 57300000 80100006 22330001 32523000 2155000a
36523000 22330001 806ffffb fffffff
```

2 Fragment 2

2.1 complete the code by replacing all occurrences of "***" with the necessary details and execute the processing for the data values in the template. Do not add additional instructions. Submit your completed (working) SRC fragment. This part of the lab will be easier to complete in the lab if some options for the "***" entries have been considered prior to arriving for the lab.

(Edit your final completed Fragment2 SRC code here)

```
    B     SkipOverVariables

Arr_Size DCD   #5    ; Arr is an array of 5 words
Arr
  DCD  #3           ; first (0-th) element of Arr
  DCD  #-4
  DCD  #0
  DCD  #-8
  DCD  #6

SkipOverVariables

    ; for ( R11 = 0; R11 < Arr_size; R11++ )
    ; R10 = Arr_Size
  LDR R10, [ Arr_Size ]
  MOV  R11, #0        ; R11 is index into array, start with index = 0

for_test               ; test whether to enter loop
  CMP  R11, R10
  BEQ  end_for        ; if fail test, then finished for loop

    ; {    ; start of for loop body
    ; if ( Arr[ R11 ] < 0 )
    ; for access to Arr: R9 = address of Arr
  MOV  R9, Arr
  LDR  R5, [ R9 , R11 ]      ; R5 = Arr[ R11 ]
  CMP  R5, #0
  BEQ  end_if

    ;    {  Arr[ R11 ] = abs( Arr[ R11 ] )      ; abs() is absolute value
    ; need value 0 for calculating abs
  MOV  R6, #0         ; R6 = 0
  SUB  R5, R6, R5     ; initial value in R5 is negative: R5 = 0 - R5 = abs( R5 )
  STR  R5, [R9,R11]   ; store Arr[ R11 ]
    ;    }
end_if

    ; }     ; end of for loop body
    ; adjust Arr index
  ADD   R11, R11, #1
  BPL    for_test

end_for
  DCD  #0xFFFFFFFF    ; breakpoint instruction
```

2.2 Assemble and run Fragment 2. To validate running the fragment in your lab report, submit the contents of Main Memory RAM before and after executing the fragment. (Hint: right-click on RAM → Save Image ...).

Before execution:

```
v2.0 raw
80F00006 00000005 00000003 FFFFFFFC 00000000 FFFFFFF8
00000006 33AFFFF9 23B00000 47BA0000 80100009 23900002
3259B000 57500000 80100003 23600000 02565000 3659B000
21BB0001 806FFFF5 FFFFFFFF
```

After execution:

```
v2.0 raw
80f00006 5 fffffffd 4 0 8
fffffffa 33afffff9 23b00000 47ba0000 80100009 23900002
3259b000 57500000 80100003 23600000
2565000 3659b000 21bb0001 806ffff5 ffffffff
```