Akram *Quiz 3 , November 8, 2021* *Ismail*

Computer Science
C.Sc. 342

Quiz No.3 To be performed

12:00-1:40PM AND 5:00-6:15 PM ***on*** November 8, 2021

Submit by 6:15 PM 11/08/2021 on Slack to Instructor

Please write your Last Name on every page:

NO CORRECTIONS ARE ALLOWED IN ANSWER CELLS!!!!!

You may use the back page for computations.

Please answer all questions. **Not all questions are of equal difficulty.**

Please review the entire quiz first and then budget your time carefully.

Please hand write and sign statements affirming that you will not cheat:

"I will neither give nor receive unauthorized assistance on this exam.
I will use only one computing device to perform this test"

Please hand write and sign here:

I will neither give nor receive unauthorized assistance on this exam. I will use only one computing device to perform this test.

This quiz has 8 pages.

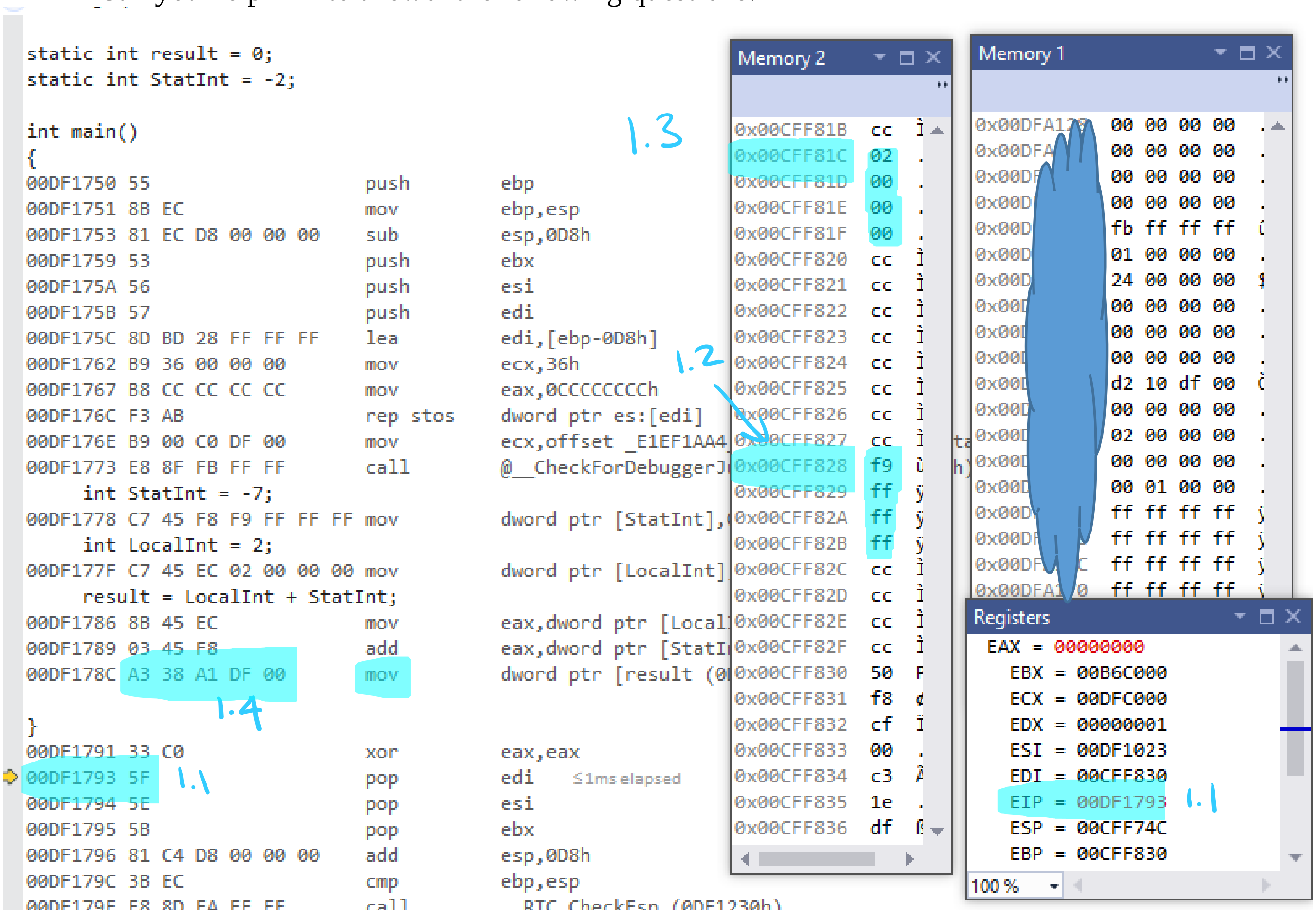
Question	Your Grade	Max Grade
1.1	5	5
1.2	10	10
1.3	10	10
1.4	10	10
2.1	5	5
2.2	5	5
2.3	10	10
2.4	10	10
3.1.1	5	5
3.1.2	5	5
3.1.3	5	5
3.2.1	5	5
3.2.2	5	5
3.2.3	5	5
3.3	5	5

Total: 100 100

Akram Quiz 3 , November 8, 2021 Ismail

Question 1.

A student, while debugging his program, unintentionally displayed partially corrupted DISSASSEMBLY windows in MS Visual Studio Debug environment.
He was able to display correctly Register window, and two Memory windows.
His task was to determine addresses of variables in the expression
result = LocalInt + StatInt in Memory at the instance of the snapshot.
He is not allowed to restart the debug session.
Can you help him to answer the following questions:



1.1 [5 points] What is the address of the instruction that will be executed next instance?

The address of the instruction that will be executed in the next instance is the address of the register EIP. This is **0x00DF1793**. The yellow arrow marker (as shown above) also tells us the address of the next executed instruction. 5 points

1.2 [10 points] Can you determine the address of variable **StatInt** in the expression? **YES** **or NO.**

Please circle around your answer. **IF** *No is your answer, then go to the next question*

ELSE *Please compute the address of variable* ***StatInt*** *in memory ,*

and determine the value of variable StatInt you can read from memory:

Address of ***StatInt*** *is …….* **0x00CFF828**

Value of ***StatInt*** *in memory is* **0x FF FF FF F9** = -7

Please justify your answers.

After the first three hex values, **FF FF FF F9** is seen being stored in memory window 2. This tells us that the address next to the F9 is the address where the variable StatInt is stored. That address is **0x00CFF828.** This is for local statint. Global statint: no.

(EBP) 0x00CFF830 + (offset) F8 = **0x00CFF828**

Value: FF FF FF F9 = 1111 1111 1111 1111 1111 1111 1111 1001

2's complement 0000 0000 0000 0000 0000 0000 0000 0110 +1 = **-7 (dec)** **10 points**

1.3 [10points] Can you determine the address of variable **LocalInt** in the expression? **YES** **or NO.**

Please circle around your answer. **IF** *No is your answer, then go to the next question*

ELSE *Please compute the address of variable* ***LocalInt*** *in memory ,*

and determine the value of variable ***LocalInt*** *you can read from memory:*

Address of ***LocalInt*** *is …….* **0x00CFF81C**

Value of ***LocalInt*** *in memory is….***0x 00 00 00 02 = 2**

Please justify your answers.

The machine code for LocalInt variable is C7 45 EC 00 00 00 02 (from picture). The hex values **00 00 00 02** are stored at address **0x00CFF81C** (seen in memory 2 window [second line]).

(EBP) 0x00CFF830 + (offset) EC = **0x00CFF81C**

Value: 00 00 00 02 = 0010 0000 0000 0000 0000 0000 0000 0000

2's complement 1101 1111 1111 1111 1111 1111 1111 1111 +1 = **2 (dec)** **10 points**

1.4 [10 points] Can you determine the address of variable **result** in the expression? **YES** **or NO.**

Please circle around your answer. **IF** *No is your answer, then go to the next question*

ELSE *Please compute the address of variable* **result** *in memory ,*

and determine the value of variable **result** *you can read from memory:*

Address of **result** *is …….*

Value of **result** *in memory is*

Please justify your answers.

Adding -7 and 2, we get -5. **-5** in hex is **FF FF FF FB**. We see this value in the code after the mov instruction (highlighted) being stored in little endian at the address **0x00DFA138.**

10 points

Question 2.

A student compiled his C code using compiler:

"GCC: (GNU) 4.8.5 20150623 (Red Hat 4.8.5-11)"
Target processor: x64, i7

Figure 1. Dump of assembly code in GDB:

```
(gdb) disassemble
Dump of assembler code for function main:
   0x00000000004004ed <+0>:     push   %rbp
   0x00000000004004ee <+1>:     mov    %rsp,%rbp
=> 0x00000000004004f1 <+4>:     movl   $0xffffffff,-0x4(%rbp)
   0x00000000004004f8 <+11>:    movl   $0x7ffffff,-0x8(%rbp)
   0x00000000004004ff <+18>:    movl   $0x8000000,-0xc(%rbp)
   0x0000000000400506 <+25>:    movl   $0x0,-0x10(%rbp)
   0x000000000040050d <+32>:    mov    -0x8(%rbp),%eax
   0x0000000000400510 <+35>:    mov    -0x4(%rbp),%edx
   0x0000000000400513 <+38>:    add    %edx,%eax
   0x0000000000400515 <+40>:    mov    %eax,-0x10(%rbp)
   0x0000000000400518 <+43>:    mov    0x200b0e(%rip),%eax
   0x000000000040051e <+49>:    mov    -0x8(%rbp),%edx
   0x0000000000400521 <+52>:    sub    %eax,%edx
   0x0000000000400523 <+54>:    mov    %edx,%eax
   0x0000000000400525 <+56>:    mov    %eax,-0x14(%rbp)
   0x0000000000400528 <+59>:    mov    $0x0,%eax
   0x000000000040052d <+64>:    pop    %rbp
   0x000000000040052e <+65>:    retq
End of assembler dump.
```

Question 2.1 [5 points] Do you have enough information to determine the content of register %eax after executing instruction at offset +40 in the dump of assembly code shown in Figure 1.?

Yes. Looking at the disassembly window, we see the first two values; 0xffffffff and 0x7ffffff, are stored into the stack with the offsets from the base pointer. Then the value is copied from the stack into registers; %eax and %edx. Adding these values together, we get **0x7fffffe**, stored into register %eax. 5 points

Question 2.2 [5 points] Please compute the address of the static variable referenced in this dump of assembly code show in Figure 1.?

We must add the offset, which is 0x200b0e, to the base address to register %rip, which is 0x000000000040051e.

0x000000000040051e + 0x200b0e = **0x000000000060102C**…address of the static variable. 5 points

Question 2.3 [10 points] In GDB environment you typed the following commands:

(gdb) x $rbp - 4
0x7fffffffdcac: 0xffffffff
(gdb) x $rbp - 8
0x7fffffffdca8: 0x07ffffff

Can you determine the content of register %rbp. ***YES or NO***?

If *No* go to next question **ELSE** Please determine the content of register *%rbp.*

Yes, adding the offset of 4 to 0x7fffffffdcac is the contents of %rbp:

0x07ffffffdcac + 4 = **0x07ffffffdcb0** **10 points**

Question 2.4 [10 points] Shown below partial stack memory for dump of assembly code shown in Figure 1?

0x7fffffffdca4:	0x00	0x00	0x00	0x08	0xff	0xff	0xff	0x07
0x7fffffffdcac:	0xff	0xff	0xff	0xff	0x00	0x00	0x00	0x00
0x7fffffffdcb4:	0x00	0x00	0x00	0x00	0x35	0xcb	0xa3	0xf7

Please determine the value of variable on stack at offset -12 decimal from base pointer %rbp. Use the value for Register %rbp you obtained in question 2.3.

We see that the value of the variable on the stack at offset -12 is **0x08000000** (little endian) since the value 0x07ffffff is offset by -8 from the base pointer (from question 2.3). When we add 4, we get an offset of -12 from the base pointer.

So, the first 4 values of the first line (highlighted in red) starting with address 0x7fffffffdca4 is an offset of -12 from the base pointer.

0x07ffffffdcb0 – 12 = 0x07ffffffdca4
Value: **0x08000000 = 2^27 (dec)** **10 points**

Question 3.

A student wrote MIPS assembly program and executed it in MARS simulator.

3.2.2

```
 .data
array1:  .word -1,0x7fffffff,0x10000080,0x80000010
.text
    main:
                la $t1,array1
# create Frame pointer
                    add $fp,$zero,$sp
#Store the address of the first element on stack
using frame pointer
                    sw $t1,0($fp)
#allocate memory on Stack for 6 integers
            addi $sp,$sp,-24          3.1.3
#load FIRST element from array1[0] to register $s0
            lw  $s0,0($t1)
#push $s0 (NO PUSH!)i.e. store register $s0
on #top of the stack
            sw  $s0,0($sp)          3.1.2
#load SECOND element from array1[1] to register $s0
            lw  $s0,4($t1)
#create new top of the stack
            addi $sp,$sp,-4
            sw  $s0,0($sp)
            #
#load third element from array1[2] to register
$s0
            lw $s0,8($t1)
#create new top of the stack
            addi $sp,$sp,-4
            sw  $s0,0(sp)          3.2.2
#load forth element from array1[3] to register
$s0
            lw  $s0,12($t1)
#create new top of the stack
            addi $sp,$sp,-4
            sw  $s0,0($sp)
```

After execution of the program in MARS simulator, he displayed the following memory windows and register file:

3.2.1

Data Segment

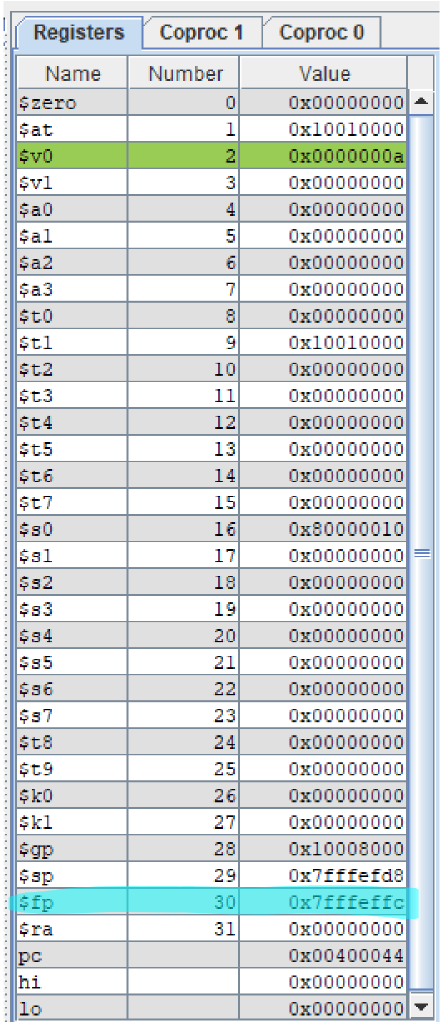
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefc0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x80000010	0x10000080
0x7fffefe0	0x7fffffff	0xffffffff	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x10010000
0x7ffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

current $sp ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Va
0x10010000	0xffffffff	0x7fffffff	0x10000080	0x80000010	
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	

0x10010000 (.data) ☑ Hexadecimal Addresses



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x10010000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x80000010
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffefd8
$fp	30	0x7fffeffc
$ra	31	0x00000000
pc		0x00400044
hi		0x00000000
lo		0x00000000

Figure 2. Register file and memory windows in MARS simulator.

Based on the information displayed in **Figure 2.** memory windows and register file above, please answer the following questions

3.1.1 [5 points] What is the address of an integer that was **first** pushed on to stack?

Address of the first integer that is pushed on to stack is 0x7fffefe0 + 0x4 (offset) = **0x7fffefe4**. When looking at the stack, the last value at the bottom is the first value pushed onto the stack. This is because stack's follow a LIFO (last-in-first-out) structure. 5 points

3.1.2 [5 points] What is the value in Hex and signed decimal of an integer that was **first** pushed on to stack?

Hex value: 0xffffffff
Signed dec: -1
Last value seen on the current stack value. 5 points

3.1.3 [5 points] What is the offset from FRAME POINTER to an integer that was **first** pushed on to stack?

(Frame pointer) 0x7fffeffc – (address of the integer first pushed on to stack) 0x7fffefe4 = **-24** (offset). 5 points

3.2.1 [5 points] What is the address of an integer that was **Last** pushed on to stack?

Address of the integer that last pushed as seen in the current $sp in the window: 0x7fffefd8. Stack follows the LIFO structure, so the last value pushed onto the stack is on top. 0x7fffefc0 + 0x18 (-24 offset in hex) = **0x7fffefd8**. 5 points

3.2.2 [5 points] What is the value in Hex and signed decimal of an integer that was **Last** pushed on to stack?

Hex: **0x80000010**
Signed dec: -8*16^7 + 1*16^1 = **-2147483632** 5 points

3.2.3 [5 points] What is the offset from FRAME POINTER to an integer that was **Last** pushed on to stack?

(Frame pointer) 0x7fffeffc – (address of the integer last pushed on to stack) 0x7fffefd8 = **-36.** You can also do this by looking at the code and noticing -24 – 4 - 4 – 4 = -36. 5 points

3.3 [5 points] Based on the data shown Figure 2. Can you determine if Frame pointer points to an **address** *or a* **value?** Please circle around your answer.
Please explain.

0x7fffeffc is an address. Looking at figure 2, $fp stores the address 0x7fffeffc. So the frame pointer points to an address at the bottom of the stack. The value stored at this address is the address of array1; 0x10010000. 5 points