2021 Fall CPSC 240-1 Answers
Final Concepts Test
December 9, 2021 2:30pm-6:00pm

The policies are the same as in the previous tests. No need to repeat them here.

Write the word "Blank" in the answer space to obtain 20% of credit.

Send only one of these 3 formats: doc, docx, or odt. **Do not send PDF.**

The total points for this test is 100.

Put your name and email address on this test on two different pages. You pick the pages.

Send to holliday@fullerton.edu PDF files will be deleted without warning.

End time = 6:00pm

1. Convert $79\frac{2}{11}$ to IEEE754-32 bits hex. [12]

[Show enough intermediate steps to convince the grader that you really know 32-bit float numbers.]

Solution: 79 = 1001111 (from a calculator)

(2/11) x 2 = 0 + (4/11)
(4/11) x 2 = 0 + (8/11)
(8/11) x 2 = 1 + (5/11)
(5/11) x 2 = 0 + (10/11)
(10/11) x 2 = 1 + (9/11)
(9/11) x 2 = 1 + (7/11)
(7/11) x 2 = 1 + (3/11)
(3/11) x 2 = 0 + (6/11)
(6/11) x 2 = 1 + (1/11)
(1/11) x 2 = 0 + (2/11)
(2/11) x 2 = 0 + (4/11)

A repeating pattern is clearly seen.

Hence, our number is 1001111 . 0010111010 0010111010 0010111010 ...x 2^0

= 1 . 0011110010111010 0010111010 0010111010 …. x 2^6

The true exponent is clearly 6. Now add bias + true exp = stored exp.

Stored exp = 0x7F + 0x6 = 0x86 = 1000 0110.

Put the components together to form our IEEE number first in binary:

0 1000 0110 00111100101110100010111 010 00101110

= 0100 0011 0001 1110 0101 1101 0001 0111

= 0x429E 5D17

2. Convert 0x39AA 9000 to a decimal float number. [12]

[Show enough intermediate steps to convince the grader that you really know 32-bit float numbers.]

Solution: Express the number in binary: 0$\boxed{011\ \ 1001\ \ 1}$010 1010 1001 0000 0000 0000

The stored exponent is 01110011. Subtract the bias number:

```
  0111 0011
- 0111 1111
```

It is easier to subtract in the opposite order:

```
   0111 1111
-  0111 0011
   0000 1100  =  12 (decimal).  However, the true exponent is -12 (decimal)
```

Now use the significand to write our number:

$1\ .\ 010\ \ 1010\ \ 1001\ \ 0000\ \ 0000\ \ 0000 \quad x \quad 2^{-12}$

$= \quad 1.01010101001 \quad x \quad 2^{-12}$

$= \quad 101010101001. \quad x \quad 2^{-23}$

$= \quad 2729 \quad x \quad 2^{-23}$

$= \quad 2729/8388608 \quad = \quad$ 0.000325322151184

//A hand calculator was used in a few of the steps above.

3. What are the names of all the preserved registers? [7]

Solution: The answer is on pages 173 and 174 of the class ebook. Here are the names:

rbx, rbp, r12, r13, r14, r15

4. Consider these integers in C++/C. [7]

```
long a = 401;
long b = 23;
long c = a%b;
```

Translate that to modern assembly. Use comments to explain which registers represent a, b and c.

Solution:

```
mov rbx, 401        ;rbx = a
mov rcx, 23         ;rcx = b
mov raxx, rbx
cqo
idiv rcx
mov r12, rdx        ;r12 = c
```

5. Directly below there are two IEEE754-64bit numbers. Add the two numbers. Show the steps used by a computer to do the addition. Show the sum. [12]

0x43F3 8000 0000 0000
0x400F C000 0000 0000

Second part. During the addition was there any loss of precision (lost of data). If yes, then how much was lost?

Solution: We'll give names X and Y to the two numbers.

X = 0x43F3 B000 0000 0000
Y = 0x400F C000 0000 0000

//I used the MATE calculator for the next steps. MATE is part of Tuffix (I think).

For X the true exponent is 0x43F – 0x3FF = 0x40 = 48 (decimal)

For Y the true exponent is 0x400- 0x3FF = 0x1 = 1 (decimal)

X = 1.00111 x 2^{48}
Y = 1.1111111 x 2^{1}

We have to move the dot in Y in order that its exponent match the exponent in X. Move the dot 47 places to the left.

Y = 0.0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0011 1111 1 x 2^{48}

The underline shows the 47 bits where the dot was moved.

Now we align the two numbers and add.

X = 1.0011 1000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 x 2^{48}
Y = 0.0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0011 1111 1 x 2^{48}

The sum is
X+Y = 1.0011 1000 0000 0000 0000 0000 0000 0000 0000 0000 0011 1111 x 2^{48}
X+Y = 0x43F3 8000 0000 003F

One bit was not included in the sum. That represents lost data.

How much is the lost data? None. Fix the math above.

6. The following dialog is part of the output of a larger program. [13]

Please enter your name: Johnny Junior
How much candy is in your Christmas sock? 4TT77
Please try again: 4...6
Please try again: ++8
Please try again: 10
We are glad you have that much candy Johnny Junior.

Show a portion of X86 that makes this dialog. This is about validating an integer input. You may use any publicly available function including functions in C/C++ libraries.

Don't make an callable function. Focus only on the block that validates an integer. You decide if the integer is 64 bits or 32 bits.

What should you do if you run out of time? Scale it down. Perhaps you omit the iteration. Perhaps you omit the dialog with the 4^{th} grader Johnny Junior.

Solution:

This question should remind you of Assignment 2 (string I/O) and Assignment 4 (numeric input validation). The answer to this question combines parts from each of those two assignments.

The solution is long, but here it is.

```
null equ 0              ;null is the name of any sequence of zero bits
max_string_length equ 64

extern isfloat
extern scanf
extern printf
global get_candy

segment .data
prompt_for_name db "Please enter your name: ", 0
sock_question db "How much candy is in your Christmas sock? ", 0
invalid_message db "Please try again: ",0
departure db "We are glad you have that much candy %s",10,0

segment .bss
namestring resb 64

segment .text
get_candy:

;Save time on a test: 15 pushes were omitted.
```

```
;Ask user for first and last names.
mov rax,0
mov rdi, prompt_for_name
call printf
```

```
;Obtain the user's names and store them in namestring
mov rax,0
mov rdi, namestring
mov rsi, max_str_length
mov rdx, [stdin]
call fgets

;Remove the newline character from the inputted string
mov rdi, namestring
call strlen
;The length is now in rax.  Save a copy in r15, and then remove the newline
mov r15,rax
mov al,null
mov [namestring+r15-1], al     ;al is the name of the lowest 1 byte of rax

;Prompt for the number of pieces of candy
mov rdi, sock_question
call printf

;Make space for the incoming candy number
sub rsp,64  ;Create 64 bytes of space for the incoming candies.

input_a_quantity:
mov rdi, string_form
mov rsi, rsp
call scanf

;Is it a real integer?
mov rdi, rsp
call isinteger

;The response is in rax: rax == 0 means false; rax /= 0 means true
cmp rax,0
jne valid_input
mov rdi, invalid_message
call printf
jmp input_a_quantity

valid_input:
mov rdi,rsp
call atol
;The candy number is now in rax.  Save a backup copy in r15.
mov r15,rax

;Say good-bye
mov rax,0
mov rdi,departure
mov rsi,namestring
call printf
```

```
;Set up the return value
mov rax, x15

add rsp,64   ;Reverse an earlier subtraction from rsp
;Fifteen pop were omitted.
ret
```

7. Here is a stack dump of the kind we all love. [10]

Offset	Address	Value
+608	00007fff01d17e88	00007fff01d19271
+600	00007fff01d17e80	0000000000000000
+592	00007fff01d17e78	00007fff01d19269
+584	00007fff01d17e70	0000000000000001
+576	00007fff01d17e68	000000000000001c
+568	00007fff01d17e60	00007fff01d17e68
+560	00007fff01d17e58	00000000004004b9
+552	00007fff01d17e50	0000000000000000
+544	00007fff01d17e48	00007fff01d17e70
+536	00007fff01d17e40	0000000000000000
+528	00007fff01d17e38	00000008FF000000
+520	00007fff01d17e30	0000000000066678
+512	00007fff01d17e28	0000000000000001
+504	00007fff01d17e20	00007fff01d17e78
+496	00007fff01d17e18	0000000000401dd0
+488	00007fff01d17e10	00007fff01d17e40
+480	00007fff01d17e08	0000000000000000
+472	00007fff01d17e00	0000000000000000
+464	00007fff01d17df8	dda1284af90c3abc
+456	00007fff01d17df0	dc5cd74251bc3abc
+448	00007fff01d17de8	0000000000000000
+440	00007fff01d17de0	0000000000000000
+432	00007fff01d17dd8	00007fff01d17e70
+424	00007fff01d17dd0	0000000000400490
+416	00007fff01d17dc8	23a2d460915c3abc
+408	00007fff01d17dc0	0000000000000000
+400	00007fff01d17db8	0000000000400586
+392	00007fff01d17db0	0000000100000000
+384	00007fff01d17da8	00007fff01d17e78
+376	00007fff01d17da0	00007fff01d17e78
+368	00007fff01d17d98	00007f01fe552a40
+360	00007fff01d17d90	0000000000000000
+352	00007fff01d17d88	0000000000000020
+344	00007fff01d17d80	0000000000000006
+336	00007fff01d17d78	0000000000000005
+328	00007fff01d17d70	0000000000000004
+320	00007fff01d17d68	0000000000000003
+312	00007fff01d17d60	0000000000000002
+304	00007fff01d17d58	0000000000000001
+296	00007fff01d17d50	0000000000000001
+288	00007fff01d17d48	0000000100000000

```
+280  00007fff01d17d40  00007fff01d17e78
+272  00007fff01d17d38  00000000ffffffff
+264  00007fff01d17d30  0000000000400613

+256  00007fff01d17d28  00007fff01d17e10
+248  00007fff01d17d20  0000000000000008
+240  00007fff01d17d18  0000000000000007
+232  00007fff01d17d10  0000000000000006
+224  00007fff01d17d08  0000000000000002
+216  00007fff01d17d00  0000000000000002
+200  00007fff01d17cf0  00007fff01d17d30
+192  00007fff01d17ce8  0000000000000007
+184  00007fff01d17ce0  000000000000000a
+176  00007fff01d17cd8  0000000000000009
+168  00007fff01d17cd0  0000000000000000
+160  00007fff01d17cc8  0000000000000007
+152  00007fff01d17cc0  00007fff01d17e70
+136  00007fff01d17cb0  00007fff01d17cf0
+128  00007fff01d17ca8  0000000000000016
+120  00007fff01d17ca0  000000000000000b
+112  00007fff01d17c98  0000000000409ffc

+104  00007fff01d17c90  00007fff01d17d28
 +96  00007fff01d17c88  00000000004007fc
 +88  00007fff01d17c80  00007fff01d17cb0
 +80  00007fff01d17c78  0000000000000012
 +72  00007fff01d17c70  0000000000000000
 +64  00007fff01d17c68  0000000000000012
 +56  00007fff01d17c60  0000000000000000
 +48  00007fff01d17c58  0000000000400876

 +40  00007fff01d17c50  00007fff01d17c90
 +32  00007fff01d17c48  000000000000001c
 +24  00007fff01d17c40  00000000004020a7
 +16  00007fff01d17c38  00007f01fe8f8970
  +8  00007fff01d17c30  000000007ffffff2
  +0  00007fff01d17c28  ffffffffffffffff
```

Use your editor to place a blank line between the activation records.

8. Suppose you have written a C++ function to which you will need to pass 9 long integers.

The prototype looks like this:

long sunshine(long a, long b, long c, long d, long e, long f, long g, long h, long i);

An assembly program wants to call sunshine. How should the programmer make the setup before calling sunshine? [9]

Solution: The assembly program has 9 integers to pass to the function. Let's make some simple assumptions regarding the current location of the 9 integers

Suppose
a is in rbx
b is in r8
c is in r9
d is in r10
e is in r11
f is in r12
g is in r13
h is in r14
i is in r15

```
;Begin set up for calling function sunshine.
mov  rdi, rbx
mov  rsi, r8
mov  rdx, r9
mov  rcx, r10
mov  r8, r11
mov r9, r12
push  r15
push  r14
push  r13
call sunshine

;The return value will be in rax
;Done.
```

9. Suppose rbx contains an address. What is the gdb command that will show in IEEE754-64 hex 100 consecutive quadwords of memory beginning that the address stored in rbx? [9]

Solution: x/100xg $rbx

10. What is a gdb command that will show the contents of the upper half of xmm9 in IEEE754-64 hex format? [9]

Solution: p/x $xmm9.v2_int64[1]

End of test. Exactly 10 questions.

Put your name and email address on this test on two different pages.

Come and say hi in the CS building in the new semester.

Have a wonderful Christmas vacation.

F. H.