

2021 Fall CPSC 240-5 Answers
Final Concepts Test
December 8, 2021 1:00pm-2:50pm

The policy is the same as in the previous tests. No need to repeat it here.

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

Send the completed test file to holliday@fullerton.edu

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.

1. Convert $117\frac{5}{12}$ to IEEE754-32 bits hex.

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

Solution:

Use a calculator to find $117 = 1110101$

For the fractional part use successive multiplication by 2.

$$(5/12) \times 2 = 0 + (5/6)$$

$$(5/6) \times 2 = 1 + (2/3)$$

$$(2/3) \times 2 = 1 + (1/3)$$

$$(1/3) \times 2 = 0 + (2/3)$$

$$(2/3) \times 2 = 1 + (1/3)$$

A pattern is starting to appear: $5/12 = 0.011010101010101 \dots$

Therefore, the original number $117\frac{5}{12}$ equals

$$1110101 . 01101010101010101 \dots \times 2^0$$

$$= 1 . 11010101101010101010101 \dots \times 2^6$$

True exponent = 6.

Compute the stored exponent = $0x7F + 0x6 = 0x85 = 10000101$

Combine the sign bit + stored exponent + significand minus hidden bit to obtain

This number = 0 10000101 110101011010101010101010101010

Next organize that binary number into groups of size 4 bits:

Number = 0100 0010 1110 1010 1101 0101 0101 0101

$$= \text{0x42E5 D555}$$

2. Convert 0x48AA 0000 to a decimal float number.

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

Solution: Write the number in binary and separate the 3 components: sign, stored exponent, and significand.

0100 1000 1010 1010 0000 0000 0000 0000

= 0 10010001 010101000000000000000000

The stored exponent is 10010001 = 0x91

Then true exponent = 0x91 – 0x7F = 0x12 = 18 (decimal)

Now we see our number = 1.010101000000000000000000 x 2¹⁸

= 1.010101 x 2¹⁸

= 1010101 x 2¹²

= 85 x 4096 = **348160**

3. What is the definition (defining property) of dynamic data in a computer program?

Solution:

Dynamic data are those whose storage requirements cannot be determined at compile-time.

In C++ and C dynamic variables will store their data in the heap.

In X86 all data are stored in the heap both static and dynamic.

4. What did Richard Stallman do to promote or advance computer technology?

Solution: He is recognized as the founder of the open source software movement.

To the current day he continues to travel the world advocating for FOSS (free open source software).

5. The next two numbers seem to be far apart. But are they really far apart? Show manually how a computer performs the process of adding the numbers. Show the sum.

Have any data been lost in the process, and, if so, how much data were lost?

All numbers are IEEE754-64 bits.

0x41AB 9800 0000 0000
0x3EA7 BC00 0000 0000

Solution: To save space we will give names to the two numbers

X = First number = 0x41AB 9800 0000 0000

Y = Second numb = 0x3EA7 BC00 0000 0000

For X the true exp is 41A – 3FF = 0x1B = 27 decimal.

[I used the MATE calculator set to hex mode to do the math in this solution. I believe MATE comes installed with Tuffix.]]


Therefore, $X = 1.1011\ 1001\ 1 \times 2^{27}$

For Y the true exp is 3EA – 3FF = - 0x15 = -21 decimal.

Therefore, $Y = 1.0111\ 1011\ 11 \times 2^{-21}$

The strategy is to express Y with the same exponential factor as X. To do this we shift the point in Y 48 places to the left.

$Y = 0.000000\ \dots\ 01\ 0111\ 1011\ 11 \times 2^{27}$



The underline shows the shift of the point 48 places to the left.


Now the two numbers have matching exponential terms and we can add them.

$X = 1.1011\ 1001\ 10000000000\ \dots\ 000 \times 2^{27}$

$Y = 0.00000000\ \dots\ 01\ 0111\ 1011\ 11 \times 2^{27}$

X+Y =

1.1011 1001 1000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001 0111 1011 11 $\times 2^{27}$



The underline shows the 52 bits of mantissa (significand). Bits on the right of the underline will be lost.

In hex the sum is 0x41AB 9800 0000 0017

The lost value is $0.0000\ \dots\ 0000\ 1011\ 11 \times 2^{27}$

To simplify move the point 58 places to the right as follows

$$\text{Lost} = 101111 \times 2^{-31} = 47 \times 2^{-31} \quad [\text{That is sufficient simplification.}]$$

End of solution question #5.

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

```
Please enter your legal name: Santa Claus
Good evening Santa Claus
How many presents will you deliver tonight? 64H7
Invalid. Please try again: 657.85
Invalid. Please try again: ++695
Invalid. Please try again: 8440
Thank you Santa Claus. Have a great trip.
```

Show a portion of an X86 function that provides the dialog above. This is about validating an inputted long integer. You may use known functions from the C/C++ libraries. You may use publicly available functions. Do declare those functions as externs. Declare strings in the .data segment.

What if you run out of time? Scale it down. Perhaps omit the iteration. Maybe omit Santa's name. You are an experienced programmer. Solving part of the problem is worth more points than solving none of it.

Solution:

This question asks for a "fragment" of an assembly program.

To show the answer here I decided to simply write the entire program and paste the key fragment on the next page of this document.

You may obtain the entire executable solution at the class website at this location.

<https://sites.google.com/a/fullerton.edu/activeprofessor/4-subjects/x86-programming/x86-examples/long-integer-validation>

Solution of Question 6:

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

segment .text
validate_integer:

;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

;Show greeting to the user.
mov rax,0
mov rdi, greeting
mov rsi, namestring
call printf

;Prompt for the number of presents
mov rdi, present_question
call printf

;Make space for the incoming number of presents.
sub rsp,64 ;Create 64 bytes of space for the incoming integer number.

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

;Is it an 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_an_integer

valid_input:
mov rdi,rsp
```



```
call atol
;The twos complement long integer is in rax.  Save a backup copy.
mov r14,rax

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

;Set up the return value
mov rax, r14

add rsp,64    ;Reverse an earlier subtraction from rsp

ret
```

The solution for question 6 has been posted as a complete program. Look for the sample program with the name: "Long Integer Validation".

7. Here is a stack dump.

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	0000000000400490
+528	00007fff01d17e38	0000000000000000
+520	00007fff01d17e30	0000000000000000
+512	00007fff01d17e28	0000000000000001
+504	00007fff01d17e20	00007fff01d17e78
+496	00007fff01d17e18	0000000000401dd0
+488	00007fff01d17e10	0000000000000000
+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	0000000000400613
+264	00007fff01d17d30	00007fff01d17d90
+256	00007fff01d17d28	0000000000000002
+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	0000000000000016
+104	00007fff01d17c90	0000000000000000
+96	00007fff01d17c88	00000000004007fc
+88	00007fff01d17c80	00007fff01d17cb0
+80	00007fff01d17c78	0000000000000012
+72	00007fff01d17c70	0000000000000000
+64	00007fff01d17c68	0000000000000012
+56	00007fff01d17c60	0000000000000000
+48	00007fff01d17c58	0000000000400876
+40	00007fff01d17c50	00007fff01d17c80
+32	00007fff01d17c48	000000000000001c
+24	00007fff01d17c40	00000000004020a7
+16	00007fff01d17c38	00007f01fe8f8970
+8	00007fff01d17c30	000000007fffffff2
+0	00007fff01d17c28	ffffffffffffffffffff

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

FYI: The back end of each AR is shown in yellow.

8. Show how to swap the contents of xmm6 and xmm9 without the usual third register or other storage.

Answer: `xorpd xmm6, xmm9`
`xorpd xmm9, xmm6`
`xorpd xmm6, xmm9`

9 This array is declared in the .data area.

```
bumble dq 3.4, 6.77, -1.96, 800.0, 499.99, 0.7
```

What is the gdb command that will show all the values of the array in IEEE754-64-bit hex format?

Answer: `x/6fg &bumble`

10. This array is declared in C++: `double w[5] = {-1.0, 6.2, 8.77, 3.49, 1.99};`

What is the gdb command that will show all the values of the array in floating point decimal format?

Answer: `p/f w`

That's all: 10 questions. Put your name and email address on this test on two different pages.

Grades are due sometime in early January. I will be going into isolation status until all your grades have been submitted.

I enjoyed having you in class during the months August to December 2021.

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

Have a wonderful Christmas vacation 2021. – FH.