2021 Fall CPSC 240-5
Final Program Test
December 13, 2021 12:00m-4:00pm

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

To receive "blank credit" (20%) send me the "Hello World" or the equivalent "Happy Birthday" program.

Send the completed set of program files to holliday@fullerton.edu

Do include all source files and a bash file.   Pause to be sure your are sending all the needed files. Forgetting to include a file makes the whole program inoperable.

There is no second chance to send a missing file later. I will be not be responding to email of any kind from test day until January 5.   You have to submit all files on December 13.

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.

In January the final class curve for section 5 will be posted in Discord by an accomplist.

Send to holliday@fullerton.edu

Requirements

Create a program with driver file in either C or C++ and operation file in X86 assembly. That is all: two source files and one bash file.

You should remember the distance formula from high school math class, namely: d = r * t. We call it distance equal rate time time. We are going to use that equation to find measure how fast your cpu performs multiplication.

We are not measuring input time or output time. We are measuring only time of execution of instructions in the cpu – not I/O time.

Freedom to use software. You may use any software products you can find on the web.

Sample dialog

This is Final exam by Lisa Finkelstein.

Welcome to SSE bench marking program.

Please enter the rate (m/sec): 15.77

Please enter the time duration (seconds): 22.3

[Invisible: Read cpu clock and save it]

[Invisible: Compute the distance and save it]

[Invisible: Read the cpu clock one more time and save it.

The distance traveled was 351.671 meters.

The computation required 7502 tics or 1266.4 nanosec.

The main program received 126.44 and will just keep it.

Have a nice day.

Color codes
Green: These actions are performed in the CPU, but there is no visible output.

Yellow: Output from the assembly source file.

Pink: Output from the driver.

Be sure to replace Lisa's name with your own name.

Multiplatform programming

Some cpu's retrun zero as the clock frequency. That cannot physically be right. When you call the get-frequency function if it tells you the frequency is 0.0 then ask the user for the frequency. In this case the dialog looks like the following.

This is Final exam by Lisa Finkelstein.

Welcome to SSE bench marking program.

Please enter the rate (m/sec): 15.77

Please enter the time duration (seconds): 22.3

[Invisible: Read cpu clock and save it]

[Invisible: Compute the distance and save it]

[Invisible: Read the cpu clock one more time and save it.

The distance traveled was 351.671 meters.

Please enter the cpu frequency (GHz):   2.4   <== Extra statement

The computation required 7502 tics or 1266.4 nanosec.

The main program received 126.44 and will just keep it.

Have a nice day.

This dialog and the dialog on the previous page came from the same program executed on different platforms. In the second dialog the AMD microprocessor returns 0.0 as it frequency. That cause the program to ask the user to manually input the CPU frequency.

Footnote: What if the user does not know the cpu of his or her own computer? Try this:
Instate the program inxi by entering the next line in the shell:

        sudo apt install inxi

When that is finished enter the bash command: inxi -C
with an upper case C. Just maybe the stars will shine and you'll see the frequency.

The semester has ended

I am glad you were all here this semester.   Now you have to move on to other academic studies.

I hope you saw I am not here simply to teach one more class and collect a pay check. That went away a long time ago. I was sharing my passion with you all semester. Thanks for coming along on the journey.

Go in peace.

F. Holliday

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   01010100000000000000000

The stored exponent is 10010001 = 0x91

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

Now we see our number = $1.01010100000000000000000 \times 2^{18}$

= 1.010101×2^{18}

= 1010101×2^{12}

= 85 x 4096   = <mark>348160</mark>

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 stored 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 continue 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.

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

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.

$X = 1.1011\ 1001\ 10000000000\ldots.000 \times 2^{27}$
$Y = 0.00000000\ \ldots\ldots\ 01\ 0111\ 1011\ 11 \times 2^{27}$

X+Y =
$1.1011\ \ 1001\ \ 1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 0111\ \underline{1011\ \ 11} \times 2^{27}$

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 $0.0000\ \ldots.\ 0000\ \ 1011\ 11 \times 2^{27}$
To simplify move the point 58 places to the right as follows

Lost = 101111 x 2^{-31} =   47 x 2^{-31} [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 the 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 om 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 that 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.

```
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   000000007ffffff2
  +0   00007fff01d17c28   ffffffffffffffff
```

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.