

CSSE 132 – Introduction to Systems Programming
Rose-Hulman Institute of Technology

Lab 3 – Question Sheet

For each of these questions, clearly explain your answers and write out any commands the questions request.

Name (Print): _____ RHIT Username: _____

Part 1: GDB

1. What is the difference between the commands `next` and `nexti` (or `n` and `ni`).

`n` is for next line of but `ni` is for next .

2. What is the difference between the commands `next` and `step` (or `n` and `s`).

`n` is for stepping functions but `s` stepping functions if any encountered.

3. What is the command to set a break point at line `X`?

4. What is the command to print out the value of a register named `X`?

5. What is the command to run the program to the next break point?

Part 2: Stack/Procedure Call: Local Variables

Read the web instructions for this part before continuing.

6. (**Breakpoint: Line 9**) At this line in `main`, what is the absolute address recorded in the register `sp`? (You can either print the register or check the register panel)
Write your answer in the cell marked **C0** of Table 1 below.

7. (**Breakpoint: Line 11**) After assigning of the values of `a` and `b` in `main`, check out the *stack* panel in GEF (scroll up in GEF to find the *stack* panel). Where do you think the local variables `a` and `b` in `main` are stored in memory? (Check the source code to see the values of the variables and find those numbers on the stack.) Remember that `ints` take up 4 bytes, while GDB shows you 8 bytes in each line of the stack.

In Table 1, each row corresponds to 4 bytes of stack memory (. Write the addresses where `a` and `b` are stored in Table 1: put the absolute address of `a` in **C1**, the relative address (i.e., $\$sp_{main} + N$) in **C2**, and the value in **C3**.

Put the absolute address of `b` in **C4**, the relative address in **C5**, and the value in **C6**.

8. (**Breakpoint: Line 4**) At this line in `do_nothing`, what is the absolute address recorded in the register `sp`? Write your answer in the **C7** cell of Table 1.

9. (**Breakpoint: Line 6**) At the end of `do_nothing`, where do you think the local variables `a` and `b` in `do_nothing` are stored in memory?

Write the (starting) addresses where `a` and `b` are stored in Table 1: put the absolute address of `a` in **C8**, the relative address (i.e., $\$sp_{do_nothing} + N$) in **C9**, and the value in **C10**.

Put the absolute address of `b` in **C11**, the relative address in **C12**, and the value in **C13**.

Absolute Addr	Relative Addr	Data	Variable	
C1	C2	C3	<code>a</code>	
C4	C5	C6	<code>b</code>	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	(several words)
C0	$\$sp_{main} + 0$			$\leftarrow sp_{main}$
C8	C9	C10	<code>a</code>	
C11	C12	C13	<code>b</code>	
C7	$\$sp_{do_nothing} + 0$			$\leftarrow sp_{do_nothing}$

Table 1: Stack for Part 2 `do_nothing.c`

10. (**Breakpoint: Line 12**) At the end of `main`, what is the value in the register `sp` equivalent to? Circle the right answer below.

`spmain`

`spdo_nothing`

11. We have noticed that `sp` changes from a value `spmain` in `main`, to a different value `spdo_nothing` in `do_nothing`, and then back to `spmain` after returning to `main`.

How does the code carry this out? Open `do_nothing.s` (type `make do_nothing.s` to generate it), carefully check the `do_nothing` function, and find the instructions that modify `sp`.

- Write the line of Assembly code that modifies `sp` at the start of the `do_nothing` function:

- Write the line of Assembly code that modifies `sp` at the end of the `do_nothing` function:

12. Based on the observations above, you should have a good idea of how local variables are handled in a program. Here is a summary:

- Each function has its own private space to store its local variables, called the *stack frame*.
- The register `sp` is the *stack pointer*. Its movement allocates and deallocates stack frames. This is called *automatic* memory allocation (because it is handled by the compiler).

Overall, what is the purpose of `sp`?

It points to (i.e., stores the address of) the stack frame of the _____ function. (Circle the correct answer below to fill in the blank.)

☐ first

☐ previous

☐ current

☐ last

Ask your instructor to verify your answers and sign off.

Verified: _____

Date/Time _____

Part 3: Stack/Procedure Call: Function Input Arguments

Read the web instructions for this part before continuing.

13. (**Breakpoint: Line 12**) Right before calling function `add`, check GEF's ARM panel: there are two `ldr` instructions before the `bl` instruction (which will branch to the `add` function).

```
1 ldr w1, [sp, #24]
2 ldr w0, [sp, #28]
```

Step over these two instructions (using `ni`). What is their effect?

They assign the value of (answer with a variable name in C), which is (answer with a scalar number), to register , and the value of (answer with a variable name in C), which is (answer with a scalar number), to register .

14. (**Breakpoint: Line 4**) At the beginning of `add` before running `int c = 8`, there are the following two `str` instructions:

```
1 str w0, [sp, #12]
2 str w1, [sp, #8]
```

What is their effect?

They assign the value of register , which is (scalar number), to the input argument (variable name in C), and the value of register , which is (scalar number), to the input argument (variable name in C).

15. (**Breakpoint: Line 7**) Where are `add`'s variables `a`, `b`, `c`, `d` stored in `add`'s stack frame? Write the values and names of these four variables in the table below. (*Note: Each data cell is 4 bytes. Each 8-byte word has been split into half-words in each line. You don't need to fill all the boxes, just the ones for `a`, `b`, `c`, `d`.*)

Relative Addr	Data	Var. name in C	Relative Addr	Data	Var. name in C
<code>sp + 28</code>			<code>sp + 24</code>		
<code>sp + 20</code>			<code>sp + 16</code>		
<code>sp + 12</code>			<code>sp + 8</code>		
<code>sp + 4</code>			<code>sp + 0</code>		

Table 2: Stack frame of `add` function in Part 3

16. (**Breakpoint: Line 7**) At this moment, the code is about to deallocate `add`'s stack frame (`add sp, sp, #0x20`) and then return with the instruction `ret` which will branch back to the calling function `main`. Immediately before these instructions, which variable (answer with a variable name in C) is loaded to `w0`? What is the purpose of doing this?
17. Based on the observations above, explain (a) how a caller function passes input arguments to a callee function and (b) how the callee returns a value to the caller.
- (a) To pass input arguments to a function, the caller loads values of input arguments into (find the answer in Problem 13). Then after starting that callee, those values are stored to the of the callee.
- (b) Before returning, the callee saves its return value to register where the calling function can find it.
18. What do *local variables* and *input arguments* have in common?
- They are both stored on .

Ask your instructor to verify your answers and sign off.

Verified: _____ Date/Time _____

Part 4: Stack/Procedure Call: Function Return

Read the web instructions for this part before continuing.

19. (**Breakpoint: Line 17**) Right before calling the `add` function in `main`, in the ARM panel in GEF, you can see a `b1` instruction will be called soon. On my computer, it shows as this (your address might appear differently)

```
1  b1 0xaaaaaaaa076c <add>
```

This is the instruction that will call the `add` function. I want to you do something seemingly odd: Write down the **address** of the next instruction in memory after this `b1` instruction.

Hint: 1) It should be a `str` instruction. 2) The address of an instruction is in the leftmost column in GEF (Check Part1: ARM Panel for reference)

20. (**Breakpoint: Line 8**) The register `x30` is used for a special purpose and is also known as `lr` for the *link register*. At the beginning of the `add` function,

- What is the value stored in the register `lr(x30)` at this moment?
- Compare your answer above to your answer to Problem 19 (*Hint: They should be the same*). What is the significance of this address? Why is it important that we stored it?

21. (**Breakpoint: Line 10**) Right before calling the `do_nothing` function in `add`, in the ARM panel in GEF, you can see a `bl` instruction will be called soon. This is the instruction that will call `do_nothing` function. Similar to Problem 19, write down the **address** of the next instruction in memory after this `bl` instruction.

Hint: It should be a `ldr` instruction.

22. (**Breakpoint: Line 4**) At the beginning of the `do_nothing` function, what is the value stored in the register `lr(x30)` at this moment? (*It will be different from the answer to Problem 20*)

23. (**Breakpoint: Line 5**) At the end of the `do_nothing` function, in the ARM panel in GEF, you can see this instruction `ret` will be run soon. The ARM documentation states that `ret` “branches unconditionally to an address in a register, with a hint that this is a subroutine return.” If no register is provided, the default is `x30(lr)`. What do you think will happen after running the `ret` instruction?

It will go back to the function, specifically in line (write a line number in C).

24. Based on the observations above, what do you think the instruction `bl` does? Assume the specific instruction is the one in Problem 19

1 `bl 0xaaaaaaaa076c <add>`

(*Hint: It changes the value in register `lr`.*)

The instruction will do two things: 1) Branch to the beginning of function ,

2) Store the address of the instruction to `$lr`. (Fill in a word that describes where this instruction is with regard to the `bl` instruction)

Ask your instructor to verify your answers and sign off before turning in this sheet.

Verified: _____ Date/Time _____