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

6. (**Breakpoint: Line 9)** At the beginning of `main`, what is the value stored in the register `sp`? (*You can either print the register or check the register panel*)
   Fill your answer in the cell of Table 1 (on the next page) where shows the label `P6`.

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)

   Answer this question with 1) the absolute memory addresses and also 2) the relative addresses with regard to `sp` (e.g., `sp + 4`).

   For `a`, the absolute address is [ ] and the relative address is `sp +` [ ].

   For `b`, the absolute address is [ ] and the relative address is `sp +` [ ].

8. (**Breakpoint: Line 4)** At the beginning of `do_nothing`, what is the value stored in the register `sp`?

   Fill your answer in the cell of Table 1 (on the next page) where shows the label `P8`.

9. (**Breakpoint: Line 6)** At the end of `do_nothing`, check 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 `do_nothing` are stored in memory. (Check the source code to see the values of the variables and find those numbers on the stack)

   Answer this question with 1) the absolute memory addresses and also 2) the relative addresses with regard to `sp` (e.g., `sp + 4`).

   For `a`, the absolute address is [ ] and the relative address is `sp +` [ ].

   For `b`, the absolute address is [ ] and the relative address is `sp +` [ ].

10. (**Breakpoint: Line 12)** At the end of `main`, what is the value stored in the register `sp`?

    Find out this value in GEF and check with Table 1, what is the current value `sp` equivalent to? Circle the right answer below.

    `sp` *main* `sp` *do_nothing*

11. Visualize the stack space: Based on your answers above, complete the empty cells in the table below (*Note: Each cell is 4 bytes).* For the relative addresses, use your answers in Problem 7 and 9.

Absolute Addr	Relative Addr	Data	Variable Name in C	
0x				
0x P6				← sp *main*
0x				
0x P8				← sp *do_nothing*

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

12. Based on the observations above, you should have some rough ideas of how local variables are handled in a program. Here is a conclusion:
Each function has its own private space to store its local variables. This space is called *Stack Frame.* Namely, each function has its own *stack frame.*

    The register `sp` is *stack pointer.* To find out how it works, we need to dive into the Assembly code. Now open `do_nothing.s` (type `make do_nothing.s` to generate it), carefully check `do_nothing` function and find the instructions that modify `sp`.

    - What value is subtracted from `sp` at the beginning of `do_nothing` function? ____
    - Why do you think it is this particular value?

    - Overall, what is the purpose of `sp`?

      It points to (i.e., stores the address of) the stack frame of ____ function.

Ask your instructor to verify your answers and sign off before moving to the next part.

Verified:_____ Date/Time_____

Part 3: Stack/Procedure Call - Function Input Arguments

13. Right before calling the `add` function, there are the two lines of `ldr` instructions before the `bl` instruction (which will branch to the `add` function). What are these two instructions trying to do?

    (Check the Assembly code window in GDB)

    Answer this question by describing "which variable is loaded into memory as which register".

14. At the beginning of `add` before running `int c = 8`,

    - What are the two numbers stored in the memory space starting with the address `sp`?
      (Check the *stack* window in GDB)
      Answer with specific numbers.

    - What variables do you think they are in the `add` function?
      (Check the *stack* window in GDB)
      Answer with variable names.

    - What are values of register `r0` and `r1` at this moment?

15. At the end of `add`, what are the numbers stored in the memory space starting with the address `sp` that you think are local variables of the `add` function?

    (Check the *stack* window in GDB and also refer to the values in C code)

    Answer this question with variables names (not values).

16. At the end of `add`, which variable is loaded to `r0`? What is the purpose of doing this?

17. Based on the observations above, explain in your own language that 1) how input arguments are passed to a function and 2) how to return a value from the function.

Ask your instructor to verify your answers and sign off before moving to the next part.

Verified:_____ Date/Time_____

Part 4: Stack/Procedure Call - Function Return

18. Right before calling the `add` function in `main`,

    - In the Assembly code window in GDB, you can see an instruction `bl <some address>` will be called soon. What is the address of the instruction after the `bl` instruction? (*Hint: It should be a* `str` *instruction.*)
      (*Note that the question asks the address of the instruction STORED after the* **bl** *instruction in the memeory, NOT the instruction that will be RUN after the* **bl** *instruction*)

    - What is the value stored in the register `$lr` at this moment?

19. At the beginning of the `add` function, What is the value stored in the register `$lr` at this moment?

20. Right before calling the `do_nothing` function in `add`,

    - In the Assembly code window in GDB, what is the address of the instruction after the `bl` instruction? (*Hint: It should be a* `ldr` *instruction.*)
      (*Note that the question asks the address of the instruction STORED after the* **bl** *instruction in the memeory, NOT the instruction that will be RUN after the* **bl** *instruction*)

    - What is the value stored in the register `$lr` at this moment?

21. At the beginning of the `do_nothing` function, What is the value stored in the register `$lr` at this moment?

22. At the end of the `do_nothing` function, in the Assembly code window in GDB, you can see a `bx lr` instruction will be run soon. Given that `bx` is almost equivalent to `b`, what do you think will happen after running this instruction?

23. Based on the observations above, what do you think the instruction `bl` does?
(*Hint: This is related to the register* `$lr`.)

24. At the end of the `add` function,

    - As you can see in the Assembly code window in GDB, it does not call `bx lr`. Why is this different from what the `do_nothing` function does?

    - In this case, take a guess at how the `add` function knows what specific place in the `main` function it should go back to.
    (*Hint: Check the complete Assembly code of add function in* `simple.s`.)

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

Verified:_____ Date/Time_____