CS 382 Fall 2023 — Computer Architecture and Organization — Lab 3

1 Task 1: Calculating Dot Product

In this task, you'll need to write a complete program to calculate the dot product of two vectors in assembly. You should write this program in a `.s` file, and you should be able to assemble/link/execute it using the QEMU emulator without any warning or error. For now we haven't learned how to print numbers out to the terminal, so you don't need to print out anything, and the CAs will check your code manually.

The `.data` segment must be declared as follows:

```
.data
vec1:   .quad   10, 20, 30
vec2:   .quad   1,  2,  3
dot:    .quad   0
```

where `vec1` and `vec2` are two vectors, and `dot` is where we store the dot product result. You should store the dot product into a variable. There's no need to use loops; you can just hard code the offsets for now. You can always assume the vector length is 3.

Requirements

- **Note** your code is a **complete** assembly program (not just a sequence of instructions). It should be able to assemble, link, and execute without error and warnings. When executed, the program should finish without problems (also **without any outputs**);
- If your code cannot assemble, you get no credit – this is the same as C programs that cannot be compiled;
- `MUL` instruction can be used for multiplications;
- Avoid using registers `X29` and `X30`;
- You have to put comments on each line of instruction;
- Put your name and honor code pledge at the top of your code in comments.

2 Task 2: Debugging Assembly Using `gdb`

To check if our programs are correct, we would have to rely on `gdb` (sorry, still not `printf()` yet!). A very comprehensive tutorial of using `gdb` to debug assembly programs is in Appendix B.3 of the textbook. Read through the section before you start this task.

In this task, you'd need to write a report on using `gdb` to debug task 1. You need to provide sufficient screenshots of `gdb` to show that your program is correct. Step into `gdb` and use commands to show that the result is correct.

Requirements

- Simply one screenshot of showing the final result is not sufficient. For each step you took and command you typed on `gdb`, you need a screenshot, and explain what you're trying to accomplish at that step. For example, setting a break point needs one; stepping into an instruction needs one, and so on;
- You must use the correct command to show directly that the dot-product calculation is correct;

- The screenshots must not be pictures taken from your phone or camera;
- The report needs to be in PDF format, with your name and honor code pledge at the top.

3 Grading

The lab will be graded based on a total of 10 points, 5 for task 1 and 5 for task 2. The following lists deductibles, and the lowest score is 0 – no negative scores:

- Task 1:
  - **-5:** the code does not assemble, or the program terminates abnormally/unsuccessfully;
  - **-5:** the code is generated by compiler;
  - **-3:** the calculation result of dot-product is wrong;
  - **-1:** one or more instructions is missing comments;
  - **-1:** the program has any type of output on terminal when executing;
  - **-1:** no pledge and/or name.
- Task 2:
  - **-5:** the code in task 1 is generated by compiler;
  - **-5:** the report is not in PDF format;
  - **-2:** the screenshots are not taken directly from the laptop;
  - **-2:** missing screenshot and/or explanation of one or more steps in debugging;
  - **-2:** not showing the final value of dot-product calculation in `gdb`;
  - **-1:** no pledge and/or name in the report.

Earlybird Extra Credit: 2% of extra credit will be given if the lab is finished by Wednesday 11:59PM EST (1 day before the lab deadline). For specific policy, see syllabus.

Attendance: check off at the end of the lab to get attendance credit.

Deliverable

(1) Assembly code for task 1;
(2) A **PDF** lab report for task 2.
No need to zip all files; just submit all files separately on Canvas.