CSC3210 – Project A1

Developing Soft and Parallel Programming Skills Using Project-Based Learning

Fall-2019, Flower Power,

Nafis Hoque, Eric Oh, Marjon Santiago, Muhammad Ali Raza, Asha Vassel

Planning & Scheduling

Work Breakdown Structure

Assignee Name	Email	Task	Duration(Hours)	Dependency	Due Date	Note
Nafis Hoque (coordinator)	nhoque2@student.gsu.edu	Create slack and coordinate team members (Task 1 & 2)	2-3 hours	Slack, GroupMe	09/20/18	100%
Eric Oh	eoh13@student.gsu.edu	Report Written Communication (Task 5)	1-2 hours	Slack, GroupMe	09/20/18	100%
Marjon Santiago	msantiago9@student.gsu.edu	Set up Raspberry Pi and create Github (Task 4)	3-4 hours	Slack, Github, GroupMe	09/20/18	100%
Muhammad Ali Raza	mraza2@student.gsu.edu	Film and edit video (Task 6)	4-5 hours	Slack, GroupMe	09/20/18	100%
Asha Vassell	avassel2@student.gsu.edu	Upload video to YouTube (Task 6)	1-2 hours	Slack, GroupMe	09/20/18	100%

Teamwork Basics

- (5p) What to do to get the task accomplished and the team members' satisfaction high?
  - Identify the problem or task and everyone's strengths and weaknesses and design our roles around them to in order to accomplish the goal.
- (18p) Answer all the questions in the Work Norms, Facilitator Norms, Communication Norms using your own words and your own context.
  - **Work norms:** Work is distributed evenly and based on how much one member can handle at a time. Deadlines are also decided as a group and if someone doesn't follow through on their role we'll have the rest of the members do the job together. The project is reviewed by members showing and explaining their work to others to be proofread for quality. If members have different opinions on the quality they collaborate with each other to discuss what's wrong with it in order for revisions to be made. Members who procrastinate on their work are encouraged and talked to at least get their roles accomplished in a timely fashion so that other members don't worry of case that said member does not meet the deadline.
  - **Facilitator norms:** A Facilitator is a member that directs and explains the roles and information to the group for a project. This position is determined if a members displays this ethic more than any other member. Rotating the facilitator is recommended in order for that member to be released from the stress of the portion or in a worst case scenario, absent for a period of time causing confusion in duty for members.
  - **Communication norms:** Communication should take place through email, text, or social chat applications so that members can be immediately be notified of new information. Talking on the phone is good for better clarity however some members are probably in class or at a job not having that luxury. Members who want to talk on the phone should keep that fact in mind and work with others' schedules.
  - **Meeting norms:** Everyone's schedule could vary, so all members should coordinate meetings based on when they have free time from other responsibilities. Meetings should be held at quiet areas with little distractions such as a library. Most meetings are an exchange of information and guiding others with can be usually be done through media, group members missing a meeting is only urgent if their physical presence is required like recording a video or if that member has a history of not doing work and not communicating with the group through media. In that case the group would have to talk to that person to identify the problem and resolve this. If the behavior persists then the group would assume the worst case that the member is not fulfilling their role and continue without them then report to the instructor in the end.
- (4p) As a team, select two cases out of the four mentioned in Handling Difficult Behavior. (use your own words and your own context)
  - Too quiet: Is either well informed of the topic but is not confident in applying said knowledge or lacks knowledge of the topic and does not want to reveal this out of embarrassment. This type of person should experience more interaction with

members in order for that person to open up to create more contribution to the project
    - Overly Talkative: Depends mostly if what the person constantly talks about is relevant to the project. A person who constantly discusses information that supports the groups goals is greatly appreciated, but should also be reminded that it's a group effort, so that other members can have a voice on the situation and contribute. This behavior in the context of topics unrelated to the project could be distracting for group members. Remind them that discussions that are repeatable and non-academic is valued for spreading enthusiasm and creating bonds among members, but should only be done in short duration, so that focus on the project isn't lost.
- (4p) When making decisions, If the team is having trouble reaching consensus, what should you do? (use your own words and your own context)
    - Prioritize for the benefit of the group as a whole and not for select. This doesn't mean we should ignore the needs or complaints of a person or two, so a compromise should be made for disagreements so that a person or party does not feel excluded if possible.
- (3p)What should you do if person may reach a decision more quickly than others and pressure people to move on before it is a good idea to do so?
    - Have all members exchange the information and progress they have so far in a meeting and explain how their current amount of knowledge and work will make completing the later tasks of the assignments tedious without a sound decision.
- (3p)What happens if most people on the team want to get an “A” on the assignment, but another person decides that a “B” will be acceptable?
    - The rest of the group should still work for the best possible grade. Even if a person is okay with a B he or she should still work to meet the required expectations for the assignment, so the rest aren't held back by insufficient work.

Raspberry PI Installation and ARM Assembly

Preface/Setup: After having received the Raspberry PI from our coordinator, I noticed that the Raspberry PI would power on, but not boot. This either meant that I had a faulty HDMI cable or there was no operating system on the micro sd card. I assumed the latter and downloaded NOOBS from the raspberry pi website and reformatted the micro sd card to FAT32 (I think by default it was formatted to FAT16 but was completely empty). NOOBS itself was around 2.1 gigabytes. After unzipping to micro sd card, I attempted to boot up the Raspberry PI again and this time it worked. Choosing Raspbian (FULL) as the OS, the installation took about 40 minutes, probably due to slow reading and writing capabilities of the micro sd card. Regardless, the installation itself was easy.

(Appendix1) NOOBS was enough to get Raspbian installed. I didn't need to use Sleezy or any extra steps just download the latest version of Noobs, extract to FAT32-formatted sd card, and mount it into the PI. Done.
My workstation was suboptimal, but got the job done.

Part 1: Raspbian is a pretty smooth operating system. Having used resource-heavy, feature-rich desktop OSes for a very long time (My personal computer runs Windows 10 Enterprise LTSC 2019 and a Debian virtual machine), It was really interesting to see how bare-bones Raspbian was. From the first log-in, there really isn't much to set-up besides the password and Wi-Fi, whereas Windows requires a plenitude of settings-tweaking and permission-granting. This operating system really feels like it was designed with minimalism and power-saving in mind.

Now with that out of the way, working with terminal was pretty straight-forward (I have created BASH scripts before so this terminal wasn't exactly foreign--ls, cd, mv, cp, grep, sed, awk, and the compilers work fine. I am completely new to Assembly however, so I had to run the following code without too much knowledge of what the headers meant:

```
@first program
.section .data
.section .text
.global _start
 _start:
mov r1,#5
sub r1,r1,#1
add r1,r1,#4

mov r7,#1
svc #0
.end
```

The function of the header of this code is still unknown but the assembly instructions under the _start: line are parse-able:

mov r1,#5 basically assigns the value 5 to register r1. The syntax would be mov destination,source. sub r1,r1,#1 subtracts 1 from the register r1 and stores it into r1. The syntax would be sub destination,operand1,operand2, where operand1 minus operand2 is stored into the destination.

add r1,r1,#4 is the same thing, except instead of subtract 1 from r1, 4 is added to r1 and stored into r1 as well.

mov r7,#1 is just like mov r1,#5, this time using the immediate operand 1 and assigning it to the register r7.

svc #0 -- this one was new so I had to perform a quick search on what the svc function does. It apparently stands for (Supervisor Calls), and is usually followed with a number called the SVC number (in this case that would be zero). It's mainly used to "interrupt"the operating system to make a request. For all intents and purposes, this seems to act like a termination/break point or pause for our debugging program so that we can see what happens in the previous lines.

I wrote the code using the nano command, but I'm sure other ways of writing it were possible (like the cat command or vi editor). Then I compiled and linked the program as requested. Then I was asked to run the following command:

```
./first
```

Normally this should execute some type of compiled program (I used this command a lot when writing in C to execute the program in terminal), and it didn't cause any errors, but it also didn't yield any output.

Did you see any output, why or why not?
No. The best explanation would be that none of the assembly instructions explicitly say to print what they are doing to the registers.
For example, "mov r1,#5" does modify the value inside of r1 but it doesn't invoke a command to print anything like "Hello human, I did what you asked me to" to the terminal.

So the only way we can really see what is going on in the program is to run it step by step in debugging mode, which was the what the next steps were.

The debugger of choice was GDB (GNU Debugger) and it has some very useful features (in particular, setting break points where the program should pause, and also listing the current values of the registers). Using it for the first time, a dialogue popped up about the GNU license, maybe as a sign of precaution. typing in the "list" command was useful, but displaying only the first 10 lines and every 10 lines afterwards was not the most efficient way of doing things. I added two operands after. making it "list 1,15" as in showing the first 15 lines.

The break point allows us to "freeze" the program in the middle of execution. We want to see the result of our mov, sub and add instructions so the break point was set after them on the 11th line using "(gdb) b 11."

Now I could safely run the program up until line 11 using the "run" command, and during this time we can assess the final values of the registers using the "info registers" command.

(Appendix2)As seen in this snippet of the output, the value stored in r1 was 8, and r7 was 1. This makes sense because we first set the value of r1 to 5, subtract one to make it 4, and then add 4 to make it 8. Then we simply assigned the immediate value 1 to r7. What surprises me, though, is unlike in Visual Studio, the rest of the registers are assigned the value 0 as the default. In Visual Studio the values stored in the registers are initialized as random hexadecimals. This brings up a question that I hope to one day answer--why do some debuggers assign random values to registers, and why do some debuggers assign the value zero?

Part 2: Our next task was to implement A = (A+B)-(C*D) in ARM Assembly, using registers as variable names. To make things more assembly-friendly, I first assigned A, B, C, and D registers and rewrote the problem.
r1 = (r1+r2)-(r3*r4)
where r1 = A, r2 = B, r3 = C, r4 = D.

Looking at the right side of the equation, the task requires two sub problems to be resolved first, r1+r2 and r3*r4. Then, taking the resultant of the first sub problem and subtracting it with the resultant of the second would give us our final value of r1.

r1+r2 in assembly is as simple as add r1,r1,r2 -- I prefer using three operands to be absolutely sure that the destination is r1, and that we are using r1 and r2 as the arithmetic inputs.

r3*r4 is the same format, but I used the "MUL" instruction to multiply, and the operands are destination,multiplicative1,multiplicative2. MUL r3,r4,r3 -- MUL is used here instead of IMUL because we are given the information later that we are multiplying in the positive domain, so unsigned integers will work fine. MUL doesn’t like operand1 being the destination, so I just swapped the multiplicatives.

The next problem I had to think about was what the order of the operands would be: sub destination,operand1,operand2 performs the operation operand1 minus operand2 and stores it into destination, so it goes like this: sub r1,r1,r3 -- This makes sense because r1 and r3 are the resultants of the sub problems, and r1 is the destination because in the original substitution, A = r1.

At this point the logic of the arithmetic is implemented but the values themselves are not initialized. According to the task, we had to assign 10 to A, 11 to B, 7 to C, and 2 to D. All we need to do is use the mov instruction at the beginning of our assembly program with immediates:

```
mov r1,#10
mov r2,#11
mov r3,#7
mov r4,#2
```

Thankfully, due to these values, our initial worry about having to keep up with the sign flag is gone, because if we compare the resultant of the two subproblems:
(10+11) and (7*2) = 21 and 14
We see that the former is greater than the latter, so the resulting value of r1 is positive, we don't have to worry about the sign flag. (Appendix3)

r1 = 21 - 14 = 7. **Our predicted value of r1 is 7**. For safe measures, I also added the svc instruction to the end, to set the break point there. **r2 should remain 11** because we did not perform any arithmetic on it. **r3 should have stored the value of 14** because it is the resultant of 7*2 (r3*r4). **r4 should be 2** because we also didn't perform any arithmetic on it.

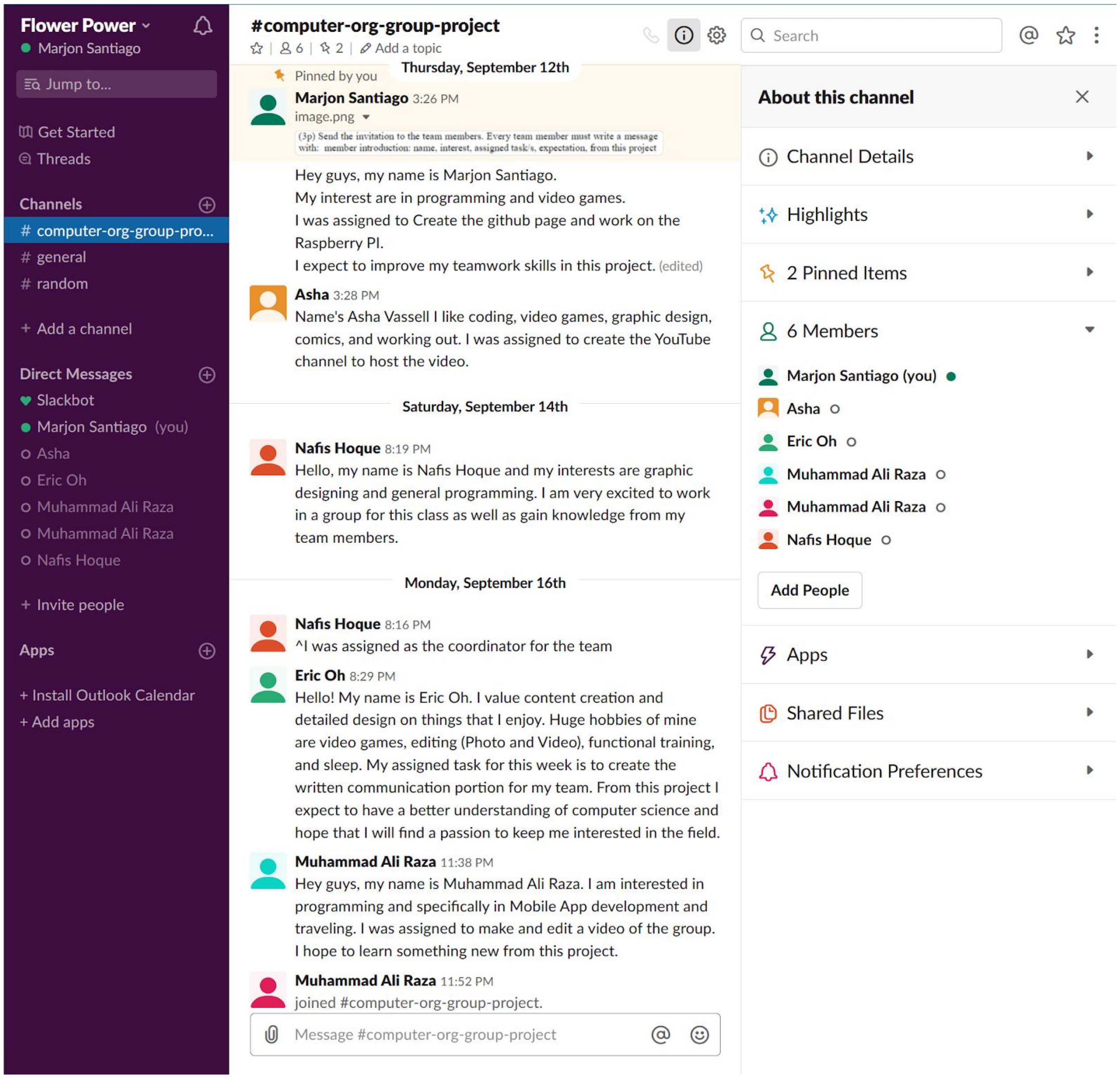
The next few steps are essentially the same commands as before--assembling the code, linking it make the executable, and then debugging it using GDB. However, I forgot to mention that used -g because apparently it makes debugging more consistent.

as -o -g arithmetic1.o arithmetic1.s (create an object file called "arithmetic1.o from arithmetic1.s")
ld -o arithmetic1 arithmetic1.o (create an executable called "arithmetic1" from "arithmetic1.o")
gdb arithmetic1 (opens up GNU Debugger so we can see what happens to the registers during execution)
(gdb) list 1,17 (shows us the first 17 lines of my code)
(gdb) b 17 (adds a break point to the last line)
(gdb) run (runs the executable up until the break point)
(gdb) info registers (shows the current values of the registers at the break point)

(Appendix4) The resulting screenshot indicates that the implementation followed through successfully. The values of r1 thru r4 are as expected, which is a good sign that the instructions worked. Working with the Raspberry PI has been quite a fun task, but there is still so much more to learn.

Appendix

Slack:
https://flowerpowerco.slack.com



Thursday, September 12th

Pinned by you

Marjon Santiago 3:26 PM
image.png

(3p) Send the invitation to the team members. Every team member must write a message with: member introduction: name, interest, assigned task/s, expectation, from this project

Hey guys, my name is Marjon Santiago.
My interest are in programming and video games.
I was assigned to Create the github page and work on the Raspberry PI.
I expect to improve my teamwork skills in this project. (edited)

Asha 3:28 PM
Name's Asha Vassell I like coding, video games, graphic design, comics, and working out. I was assigned to create the YouTube channel to host the video.

Saturday, September 14th

Nafis Hoque 8:19 PM
Hello, my name is Nafis Hoque and my interests are graphic designing and general programming. I am very excited to work in a group for this class as well as gain knowledge from my team members.

Monday, September 16th

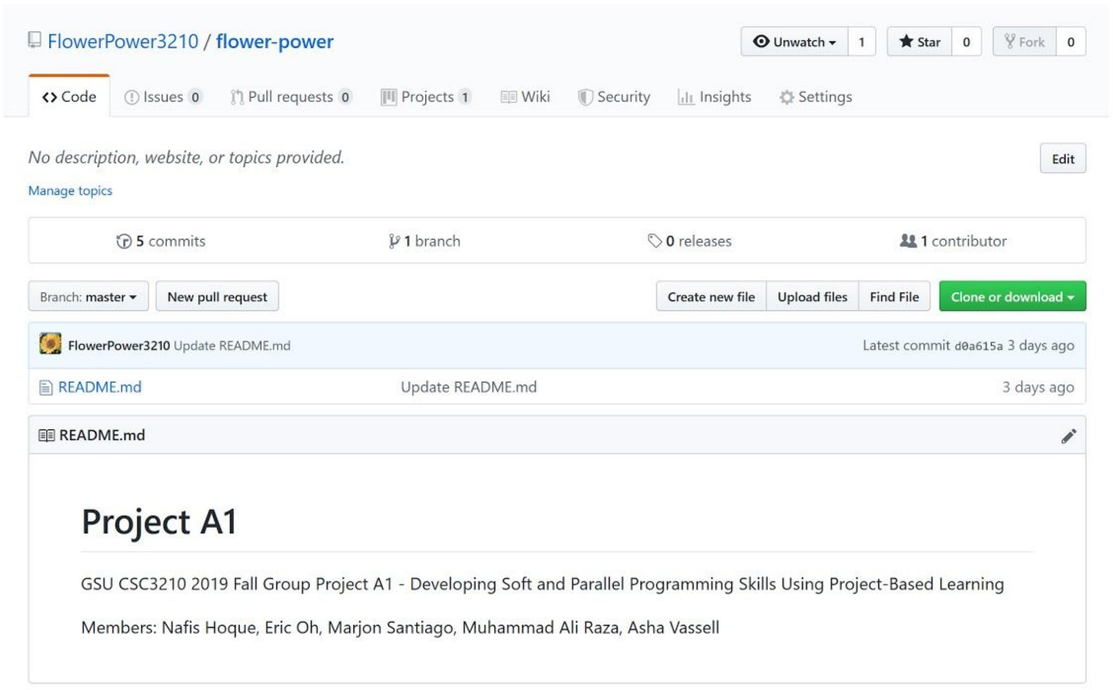
Nafis Hoque 8:16 PM
^I was assigned as the coordinator for the team

Eric Oh 8:29 PM
Hello! My name is Eric Oh. I value content creation and detailed design on things that I enjoy. Huge hobbies of mine are video games, editing (Photo and Video), functional training, and sleep. My assigned task for this week is to create the written communication portion for my team. From this project I expect to have a better understanding of computer science and hope that I will find a passion to keep me interested in the field.

Muhammad Ali Raza 11:38 PM
Hey guys, my name is Muhammad Ali Raza. I am interested in programming and specifically in Mobile App development and traveling. I was assigned to make and edit a video of the group. I hope to learn something new from this project.

Muhammad Ali Raza 11:52 PM
joined #computer-org-group-project.

Github:
https://github.com/FlowerPower3210/flower-power


FlowerPower3210 / flower-power

No description, website, or topics provided.

5 commits | 1 branch | 0 releases | 1 contributor

FlowerPower3210 Update README.md — Latest commit d0a615a 3 days ago

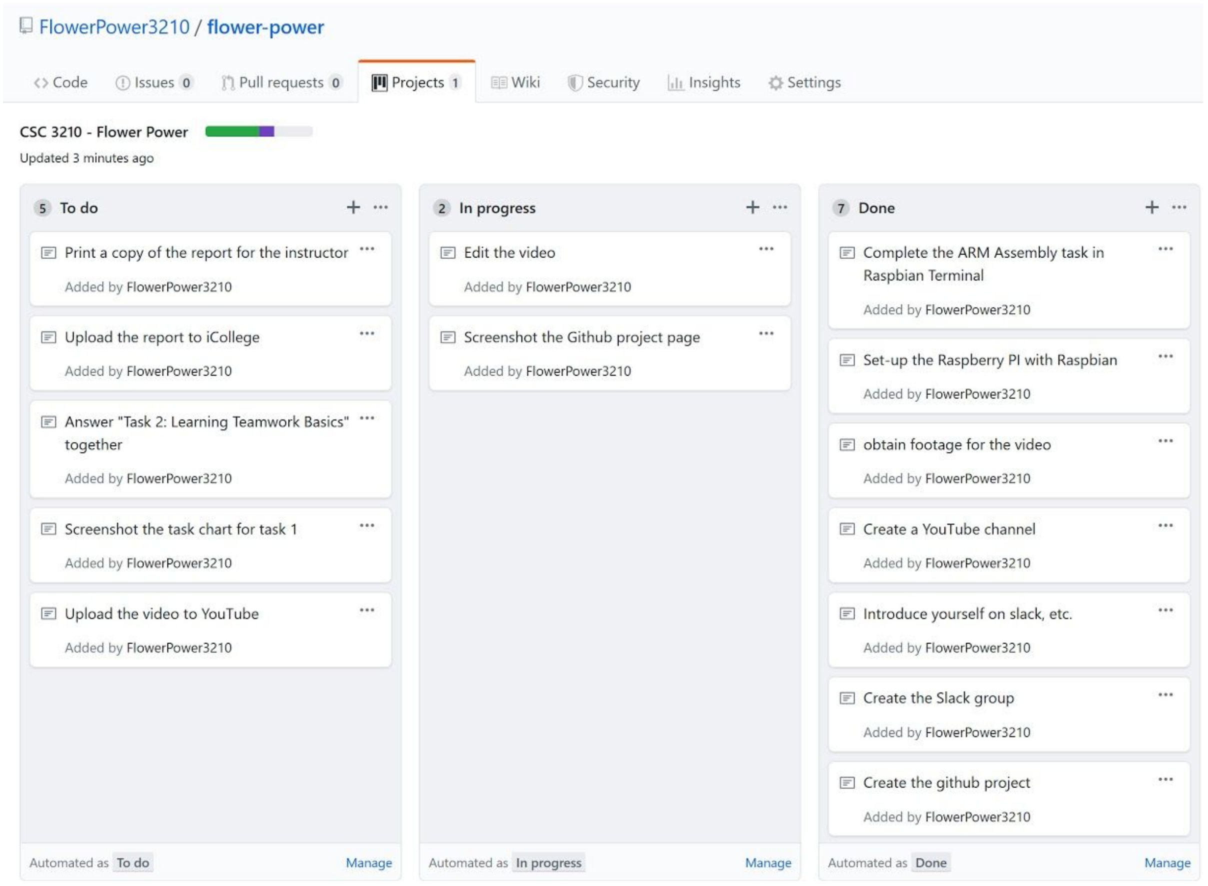
README.md — Update README.md — 3 days ago

README.md

Project A1

GSU CSC3210 2019 Fall Group Project A1 - Developing Soft and Parallel Programming Skills Using Project-Based Learning

Members: Nafis Hoque, Eric Oh, Marjon Santiago, Muhammad Ali Raza, Asha Vassell


FlowerPower3210 / flower-power

CSC 3210 - Flower Power

Updated 3 minutes ago

5 To do

- Print a copy of the report for the instructor — Added by FlowerPower3210
- Upload the report to iCollege — Added by FlowerPower3210
- Answer "Task 2: Learning Teamwork Basics" together — Added by FlowerPower3210
- Screenshot the task chart for task 1 — Added by FlowerPower3210
- Upload the video to YouTube — Added by FlowerPower3210

Automated as To do — Manage

2 In progress

- Edit the video — Added by FlowerPower3210
- Screenshot the Github project page — Added by FlowerPower3210

Automated as In progress — Manage

7 Done

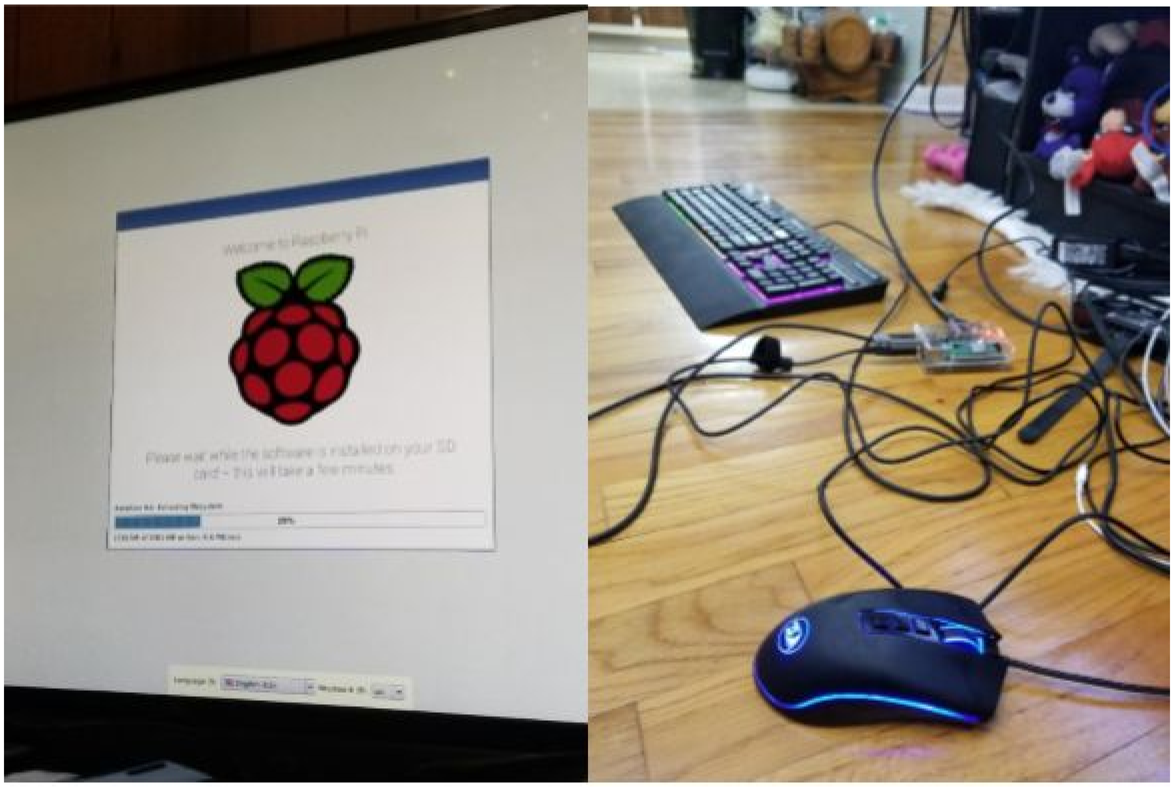
- Complete the ARM Assembly task in Raspbian Terminal — Added by FlowerPower3210
- Set-up the Raspberry PI with Raspbian — Added by FlowerPower3210
- obtain footage for the video — Added by FlowerPower3210
- Create a YouTube channel — Added by FlowerPower3210
- Introduce yourself on slack, etc. — Added by FlowerPower3210
- Create the Slack group — Added by FlowerPower3210
- Create the github project — Added by FlowerPower3210

Automated as Done — Manage

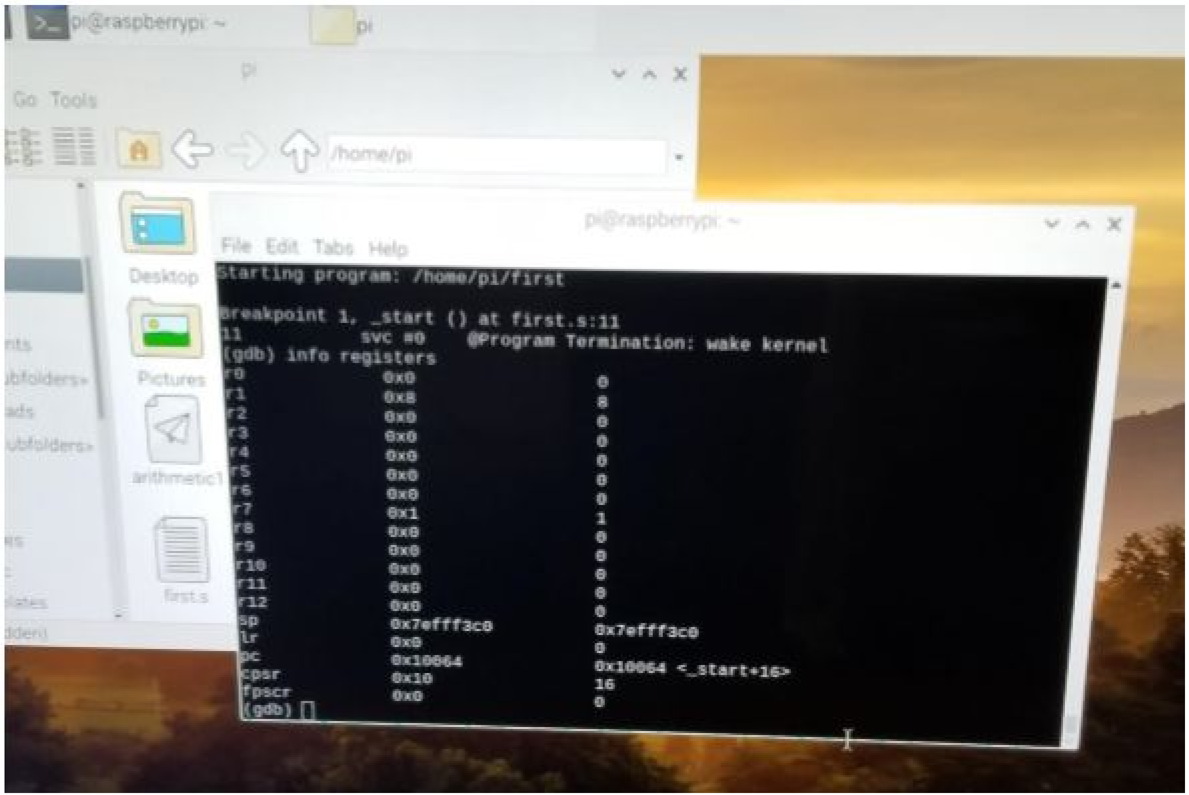
Youtube:
https://youtu.be/yKyKg6zDIuU
https://www.youtube.com/channel/UChuA0E52xD9KTHEAExKOcEw

RaspberryPi and ARM Programming:

Apendix1:

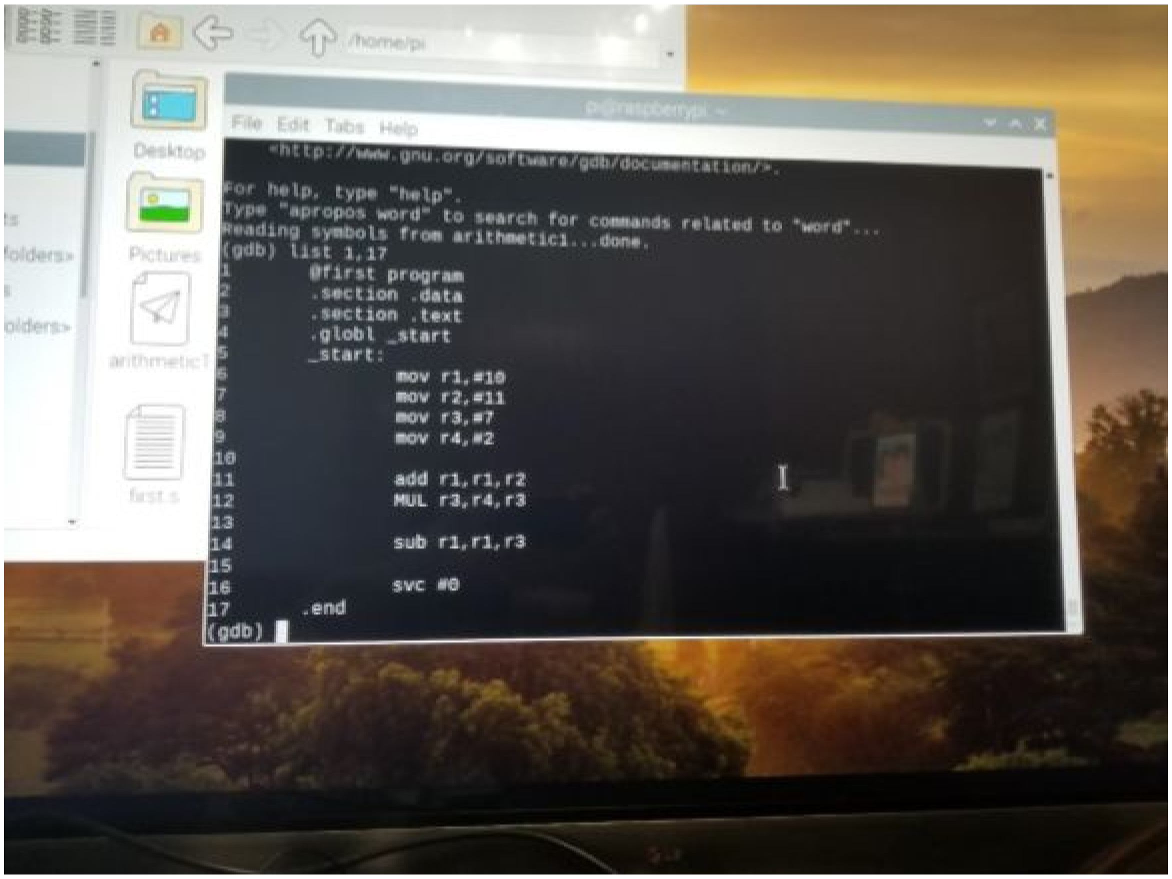


Appendix2:



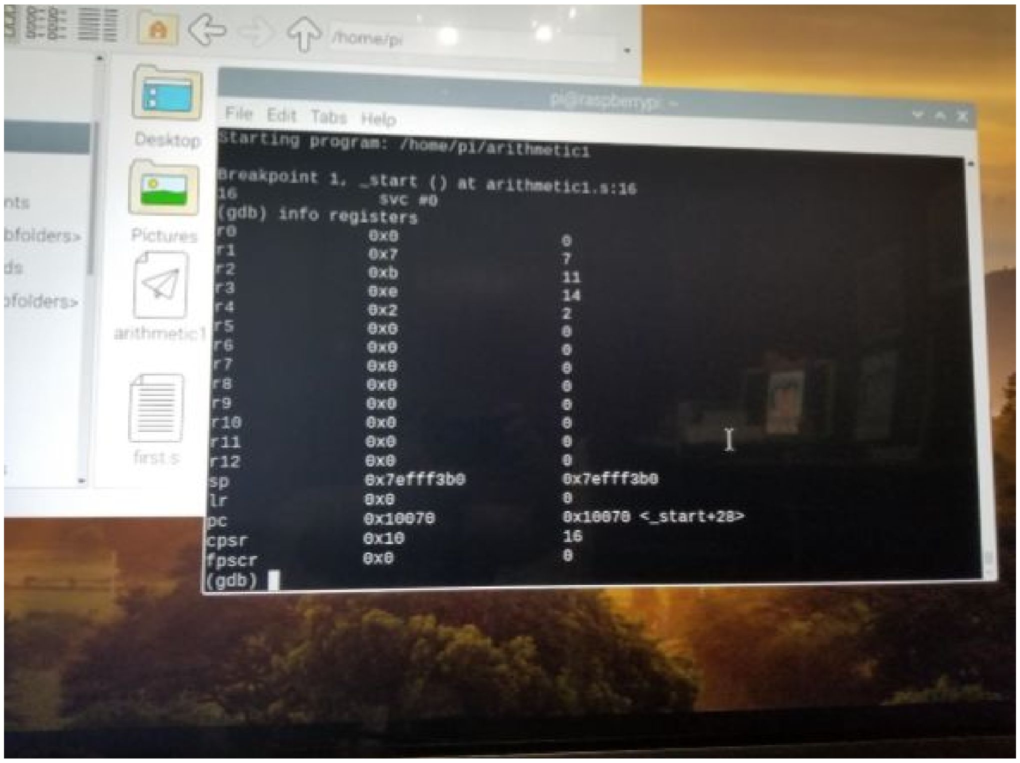
```
Starting program: /home/pi/first

Breakpoint 1, _start () at first.s:11
11              svc #0      @Program Termination: wake kernel
(gdb) info registers
r0             0x0                 0
r1             0x8                 8
r2             0x0                 0
r3             0x0                 0
r4             0x0                 0
r5             0x0                 0
r6             0x0                 0
r7             0x1                 1
r8             0x0                 0
r9             0x0                 0
r10            0x0                 0
r11            0x0                 0
r12            0x0                 0
sp             0x7efff3c0          0x7efff3c0
lr             0x0                 0
pc             0x10064             0x10064 <_start+16>
cpsr           0x10                16
fpscr          0x0                 0
(gdb) 
```

Appendix3:



```
    <http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from arithmetic1...done.
(gdb) list 1,17
1          @first program
2          .section .data
3          .section .text
4          .globl _start
5          _start:
6                   mov r1,#10
7                   mov r2,#11
8                   mov r3,#7
9                   mov r4,#2
10
11                  add r1,r1,r2
12                  MUL r3,r4,r3
13
14                  sub r1,r1,r3
15
16                  svc #0
17         .end
(gdb)
```

Appendix4:



```
Starting program: /home/pi/arithmetic1

Breakpoint 1, _start () at arithmetic1.s:16
16                 svc #0
(gdb) info registers
r0             0x0                 0
r1             0x7                 7
r2             0xb                 11
r3             0xe                 14
r4             0x2                 2
r5             0x0                 0
r6             0x0                 0
r7             0x0                 0
r8             0x0                 0
r9             0x0                 0
r10            0x0                 0
r11            0x0                 0
r12            0x0                 0
sp             0x7efff3b0          0x7efff3b0
lr             0x0                 0
pc             0x10070             0x10070 <_start+28>
cpsr           0x10                16
fpscr          0x0                 0
(gdb)
```