|  |  |
| --- | --- |
| Arm assembly programming | Fall 2019  Group Name: Quinary  Members: Ivana Lanzani, Dominick DiLeo, Meet Patel, Akiva Ochoa, Chowdhury Mobin. |

Raspberry pi is a low cost mini computer which was developed in the United Kingdom. It is a single board Linux based computer which can perform programs written in many languages such as C, C++, Python, JAVA etc.

Properly installing the pi was essential for this project. First, we downloaded the zipped file Raspbian Stretch with Desktop and image flasher app called Etcher. After which we used Etcher to flash the Raspbian image to our SD card. We used the power cable and HDMI cable provided within the box to connect the Raspberry pi to a monitor. A keyboard and a mouse which was connected via USB A cable to the Raspberry was used to complete our setup.

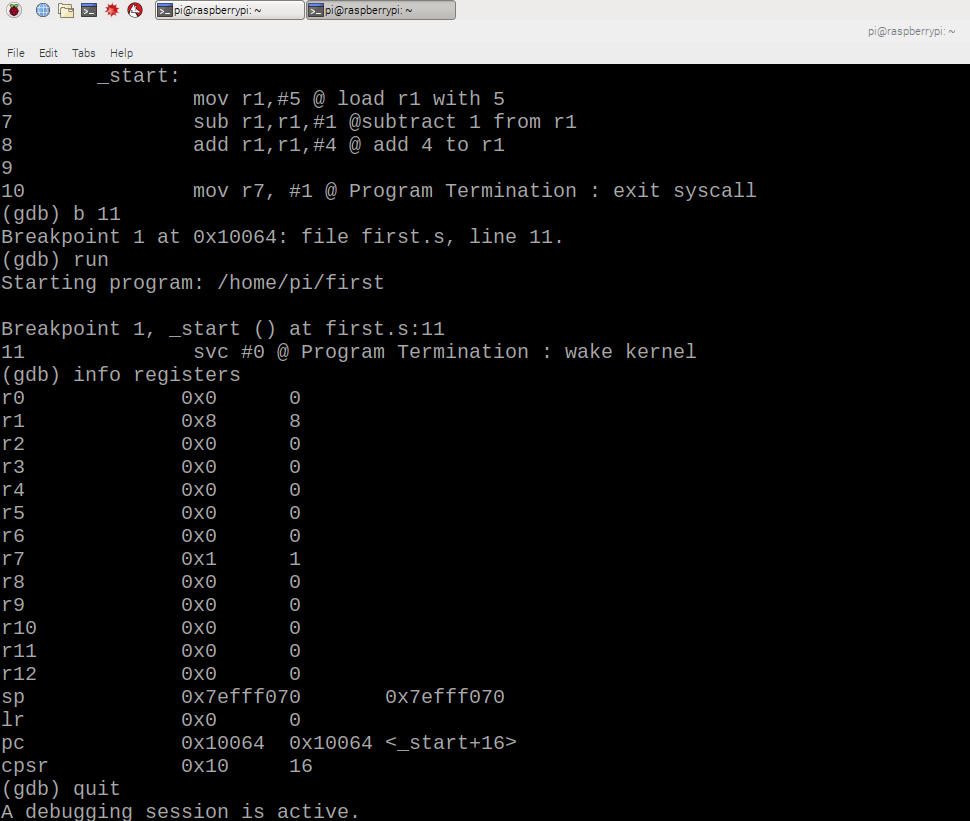
Launching the Raspberry pi for the first time, six icons were present on the top bar. The 4th icon is the terminal, which will be used to write and run our program on Raspberry Pi.

To open the text editor and write a program, we typed the command “nano first.s”. Which opened a new file named first.s using nano editor. The following commands “.selection .data”, “.selection .text” and “.globl \_start” are written for the linker. The command “\_start:” , tells the linker that this is the entry point of the program.

When we write “ mov r1, #5”, it loads the register r1 with the value of 5. The next line of command “sub r1, r1, #1”, subtracts 1 from the value of register r1, which was 5 and saves the value after the subtraction in register r1. “add r1, r1, #4”, this adds 4 to the value of the register r1 and stores the value to r1. “mov r7, #1” loads the register r7 with the value of 1 and the next line of command “svc #0”, which directs the processor to pass control of the computer to the supervisor program of the operating system. The command, “. end” marks the end of the program. Then we save the file by pressing “control w” on our keyboard.

To assemble the file the following command should be typed “as -o first.o first.s”. Here it will create an object file for our file “first.s”. Now linked our file to an executable by typing “ld -o first first.o”. After we tried to run the program by typing the “./first” command, we couldn’t see any output. To see the contents of the registers and memory we must GDB for debugging our program. To do so we added “-g” to the assembler command which now stands “as -g -o first.o first.s”. The next line for linker command will remain the same. To launch the debugger, we typed gdb followed by our executable file name which is “gdb first”. Typing “gdb list” gives us the first 10 lines of the source code. To see the next ten lines, we just must press Enter key on the keyboard.

To examine the registers, we must stop the program in its execution by setting a breakpoint. We can set a breakpoint at line 11 by typing “b 11” then pressing the Enter key. To hit the breakpoint, we must start the program execution from the beginning by typing “run”. CPU registers can be examined to see the result by typing the command “info registers”, which will show us the register name, hexadecimal contents and the contents in decimal.



To take the screenshot we opened another terminal window and typed “scrot”. Here we can see that the value of r1 is 8 which has changed as we subtracted 1 from 5, the initial value, then added 4 to it. Register r7 has stored value 1 on it from the command “mov r1, #1”.

For part 2, we have to calculate the expression , and the values given are A=10, B=11, C=7 and D=2. To start the new program, we start the terminal and type the following command “nano arithmetic1.s”. It will start the nano editor and open our file, named arithmetic1.s. The following commands are written for the linker, “ .selection .data”, “.selection .text”, “.globl \_start”, all in separate lines. To specify an entry point of the program we typed the command “\_start”.

The next step of our program is to load values to the registers. To do so, we type the following commands:

mov r1, #10 @r1 is A

mov r2, #11 @r2 is B

mov r3, #7 @r3 is C

mov r4, #2 @r4 id D

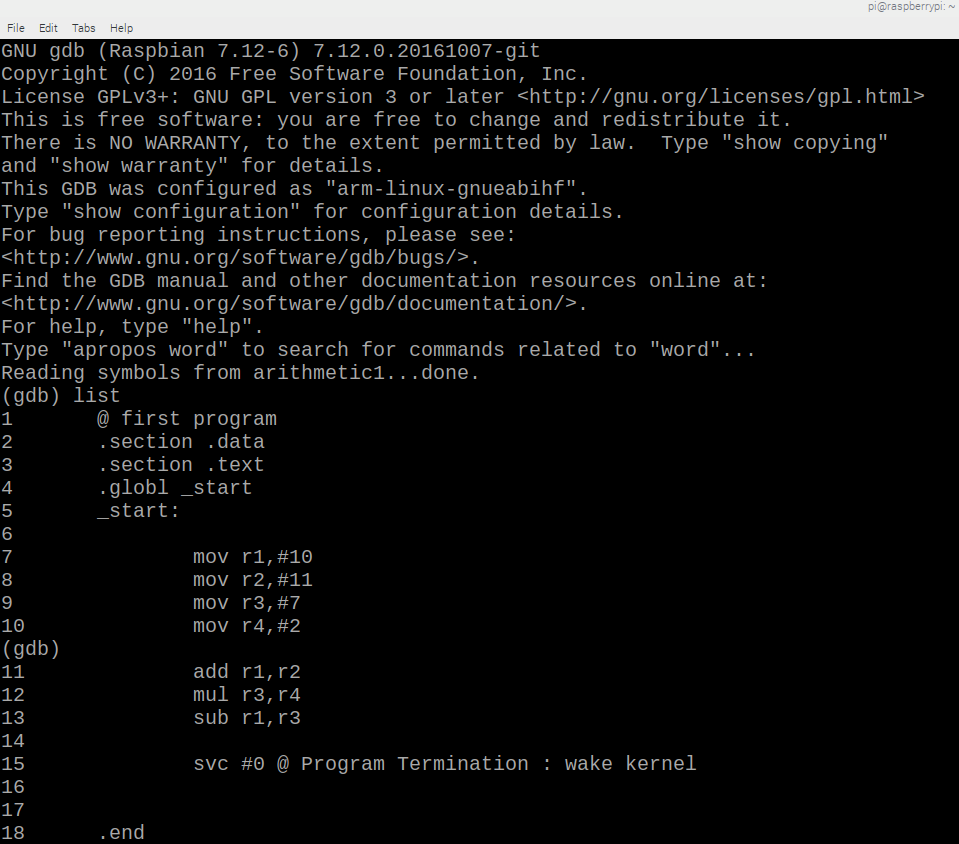
These commands just loaded the values 10, 11, 7, 2 to r1, r2, r3 and r4 registers. After loading the value, we start adding, subtracting and multiplying by typing the following commands on terminal:

add r1, r2

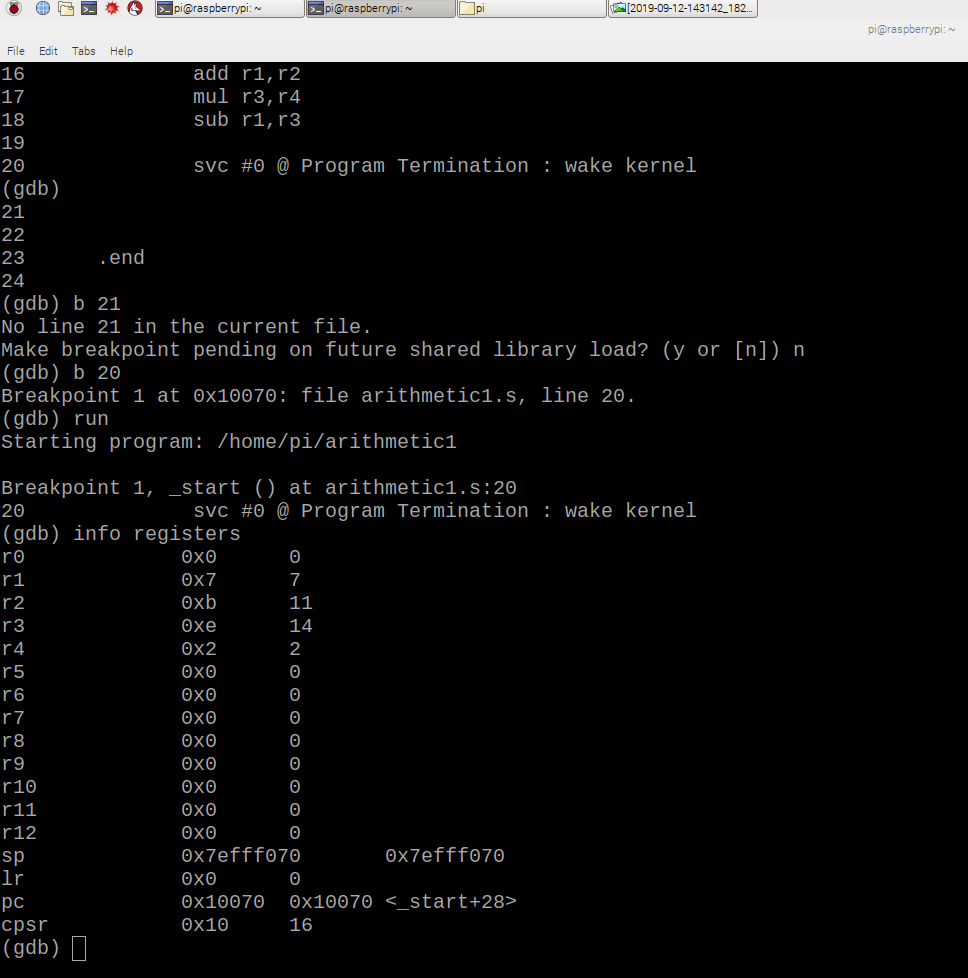
mul r3, r4

sub r1, r3

The first command adds the value loaded on r1 and r2, then stores the result on r1 register. The second line of command multiplies the values loaded on r3 and r4 and loads the result on register r3. Third line of command subtracts the value loaded on r3 from r1, this command will use will the values, register r1 and r3 had after the addition and multiplication. Then we write the supervisor call command “svc #0” and end our program with “.end” command on the terminal.



To run the program, we assemble it by typing “as -g -o arithmetic1.o arithmetic1.s”, which creates an object file. To get an executable file we type the linker command “ld -o arithmetic1 arithmetic1.o”. The command “gdb arithmetic1” will launch the GNU debugger. After we set a breakpoint “b (line number)” the program is ready to run by using the command “run”. As the program has been executed, we can now see the register values by writing “info registers” command on the terminal.



The register r1 which was our A holds the value 7, r2 or B is 11, r3 or C is 14 and r4 or D is 2. Looking at registers we can confirm that we have successfully calculated the expression with the given values. , which is the value of register r1.