Sara Kazemi
CSIS 212
10/06/2019

Assignment #4

Chapter 9.4 Exercises 1-5

1. Enter the program in Listing 9.1.3 and use `gdb` to make sure it works. Next, change the program so that it returns a non-zero integer. Run it with `gdb`. What number base does `gdb` use to display the exit code?

```
@ doNothingProg2.s
@ Minimum components of a C program, in assembly language.
@ 2017-09-29: Bob Plantz

@ Define my Raspberry Pi
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Program code
        .text
        .align  2
        .global main
        .type   main, %function
main:
        str     fp, [sp, -4]!   @ save caller frame pointer
        add     fp, sp, 0       @ establish our frame pointer

        mov     r3, 0           @ return 0;
        mov     r0, r3          @ return values go in r0

        sub     sp, fp, 0       @ restore stack pointer
        ldr     fp, [sp], 4     @ restore caller's frame pointe
        bx      lr              @ back to caller
```

Listing 9.1.3. A "null" program (prog asm).

in-context

```
Reading symbols from ./doNothingProg2...done.
(gdb) break 19
Breakpoint 1 at 0x103f4: file doNothingProg2.s, line 19.
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x000103f4 doNothingProg2.s:19
(gdb) run
Starting program: /home/pi/ch9/donothing/doNothingProg2

Breakpoint 1, main () at doNothingProg2.s:19
19          mov r0, r3    @ return values go in r0
(gdb) i r
r0             0x1      1
r1             0x7efff384       2130703236
r2             0x7efff38c       2130703244
r3             0xa      10
r4             0x0      0
r5             0x0      0
r6             0x102c0  66240
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x76fff000       1996484608
r11            0x7efff22c       2130702892
r12            0x76fa3000       1996107776
sp             0x7efff22c       0x7efff22c
lr             0x76e7e294       1994908308
pc             0x103f4  0x103f4 <main+12>
cpsr           0x60000010       1610612752
(gdb) cont
Continuing.
[Inferior 1 (process 2440) exited with code 012]
```

The value that should be returned is 10, which is represented by gdb as 012, which is 10 in **octal**.

2. Write the C function:

```
/* f.c */
int f(void) {
return 0;
}
```

in assembly language. Make sure that it assembles with no errors. Use the `-S` option to compile `f.c` and compare `gcc` 's assembly language with yours.

(prog asm)

```
pi@kazemi:~/ch9/f $ more f.s
@ f.s
@ Does nothing -- returns 0
@ Sara Kazemi
@ CSIS 212

@ Define my Raspberry pi
    .cpu cortex-a53
    .fpu neon-fp-armv8
    .syntax unified

@ Program code
    .text
    .align 2
    .global f
    .type f, %function

f:
    str fp, [sp, -4]!   @ save caller frame pointer
    add fp, sp, 0       @ establish our frame ptr
    mov r3, 0           @ return 0
    mov r0, r3          @ return values go in r0
    sub sp, fp, 0       @ restore stack pointer
    ldr fp, [sp], 4     @ restore callers frame pointer
    bx lr               @ back to caller
pi@kazemi:~/ch9/f $ as --gstabs -o f.o f.s
pi@kazemi:~/ch9/f $ ls
f.o  f.s
```

(gcc asm)

```
pi@kazemi:~/ch9/f/gcc $ gcc -S -O0 f.c
pi@kazemi:~/ch9/f/gcc $ ls
f.c  f.s
pi@kazemi:~/ch9/f/gcc $ more f.s
        .arch armv6
        .eabi_attribute 27, 3
        .eabi_attribute 28, 1
        .fpu vfp
        .eabi_attribute 20, 1
        .eabi_attribute 21, 1
        .eabi_attribute 23, 3
        .eabi_attribute 24, 1
        .eabi_attribute 25, 1
        .eabi_attribute 26, 2
        .eabi_attribute 30, 6
        .eabi_attribute 34, 1
        .eabi_attribute 18, 4
        .file   "f.c"
        .text
        .align  2
        .global f
        .type   f, %function
f:
        @ args = 0, pretend = 0, frame = 0
        @ frame_needed = 1, uses_anonymous_args = 0
        @ link register save eliminated.
        str     fp, [sp, #-4]!
        add     fp, sp, #0
        mov     r3, #0
        mov     r0, r3
        sub     sp, fp, #0
        @ sp needed
        ldr     fp, [sp], #4
        bx      lr
        .size   f, .-f
        .ident  "GCC: (Raspbian 4.9.2-10) 4.9.2"
        .section        .note.GNU-stack,"",%progbits
```

The two versions are basically the same. The GCC-created assembly code contains some things my program does not: a slightly different architecture version, .eabi_attributes, .size, .file, .ident, and .section attributes, and the # prepended before numerical values.

3. Write the C function:

```
/* g.c */
int g(void) {
return 123;
}
```

in assembly language. Make sure that it assembles with no errors. Use the `-S` option to compile `g.c` and compare `gcc` 's assembly language with yours.

(prog asm)

```
@ g.s
@ Does nothing -- returns 123
@ Sara Kazemi
@ CSIS 212

@ Define my Raspberry pi
    .cpu cortex-a53
    .fpu neon-fp-armv8
    .syntax unified

@ Program code
    .text
    .align 2
    .global f
    .type f, %function


g:
    str fp, [sp, -4]!   @ save caller frame pointer
    add fp, sp, 0       @ establish our frame ptr
    mov r0, 123         @ return value 123 goes in r0
    sub sp, fp, 0       @ restore stack pointer
    ldr fp, [sp], 4     @ restore callers frame pointer
    bx lr               @ back to caller
```

(gcc asm)

```
[pi@kazemi:~/ch9/f/gcc $ more g.s
        .arch armv6
        .eabi_attribute 27, 3
        .eabi_attribute 28, 1
        .fpu vfp
        .eabi_attribute 20, 1
        .eabi_attribute 21, 1
        .eabi_attribute 23, 3
        .eabi_attribute 24, 1
        .eabi_attribute 25, 1
        .eabi_attribute 26, 2
        .eabi_attribute 30, 6
        .eabi_attribute 34, 1
        .eabi_attribute 18, 4
        .file   "g.c"
        .text
        .align  2
        .global g
        .type   g, %function
g:
        @ args = 0, pretend = 0, frame = 0
        @ frame_needed = 1, uses_anonymous_args = 0
        @ link register save eliminated.
        str     fp, [sp, #-4]!
        add     fp, sp, #0
        mov     r3, #123
        mov     r0, r3
        sub     sp, fp, #0
        @ sp needed
        ldr     fp, [sp], #4
        bx      lr
        .size   g, .-g
        .ident  "GCC: (Raspbian 4.9.2-10) 4.9.2"
        .section        .note.GNU-stack,"",%progbits
```

The two versions are basically the same. The GCC-created assembly code contains some things my program does not: a slightly different architecture version, .eabi_attributes, .size, .file, .ident, and .section attributes, and the # prepended before numerical values. It also uses r3 to store 123.

4. Write three assembly language functions that do nothing but return an integer. They should each return different, non-zero, integers. Write a C `main` function to test your assembly language functions. The `main` function should capture each of the return values and display them using `printf`.

```
[pi@kazemi:~/ch9/checker $ more checkReturned.c
/* checkReturned.c
 * Checks assembly functions
 * that returns a non-zero integer
 * by printing them out
 *
 * Sara Kazemi
 * CSIS 212
 */

#include <stdio.h>
int n1(void);
int n2(void);
int n3(void);

int main()
{
  int number;

  number = n1();
  printf("Result of n1(): %i, ", number);

  number = n2();
  printf("Result of n2(): %i, ", number);

  number = n3();
  printf("Result of n3(): %i.\n", number);

  return 0;
}
```

```
[pi@kazemi:~/ch9/checker $ more n1.s
@ n1.s
@ Does nothing -- returns -50
@ Sara Kazemi
@ CSIS 212

@ Define my Raspberry pi
    .cpu cortex-a53
    .fpu neon-fp-armv8
    .syntax unified

@ Program code
    .text
    .align 2
    .global n1
    .type n1, %function

n1:
    str fp, [sp, -4]!   @ save caller frame pointer
    add fp, sp, 0       @ establish our frame ptr
    mov r3, -50           @ return -50
    mov r0, r3          @ return values go in r0
    sub sp, fp, 0       @ restore stack pointer
    ldr fp, [sp], 4     @ restore callers frame pointer
    bx lr               @ back to caller
```

```
[pi@kazemi:~/ch9/checker $ more n2.s
@ n2.s
@ Does nothing -- returns 50
@ Sara Kazemi
@ CSIS 212

@ Define my Raspberry pi
    .cpu cortex-a53
    .fpu neon-fp-armv8
    .syntax unified

@ Program code
    .text
    .align 2
    .global n2
    .type n2, %function

n2:
    str fp, [sp, -4]!   @ save caller frame pointer
    add fp, sp, 0       @ establish our frame ptr
    mov r3, 50           @ return 50
    mov r0, r3          @ return values go in r0
    sub sp, fp, 0       @ restore stack pointer
    ldr fp, [sp], 4     @ restore callers frame pointer
    bx lr               @ back to caller
```

```
[pi@kazemi:~/ch9/checker $ more n3.s
@ n3.s
@ Does nothing -- returns 255
@ Sara Kazemi
@ CSIS 212

@ Define my Raspberry pi
    .cpu cortex-a53
    .fpu neon-fp-armv8
    .syntax unified

@ Program code
    .text
    .align 2
    .global n3
    .type n3, %function

n3:
    str fp, [sp, -4]!   @ save caller frame pointer
    add fp, sp, 0       @ establish our frame ptr
    mov r3, 255           @ return 255
    mov r0, r3          @ return values go in r0
    sub sp, fp, 0       @ restore stack pointer
    ldr fp, [sp], 4     @ restore callers frame pointer
    bx lr               @ back to caller
```

```
[pi@kazemi:~/ch9/checker $ ./checkReturned
Result of n1(): -50, Result of n2(): 50, Result of n3(): 255.
```

5. Write three assembly language functions that do nothing but return a character. They should each return different characters. Write a C `main` function to test your assembly language functions. The `main` function should capture each of the return values and display them using `printf`.

```
pi@kazemi:~/ch9/checker $ more checkReturned2.c
/* checkReturned2.c
 * Checks assembly functions
 * that returns a character
 * by printing them out
 *
 * Sara Kazemi
 * CSIS 212
 */

#include <stdio.h>
char c1(void);
char c2(void);
char c3(void);

int main()
{
  char c;

  c = c1();
  printf("Result of c1(): %c, ", c);

  c = c2();
  printf("Result of c2(): %c, ", c);

  c = c3();
  printf("Result of c3(): %c.\n", c);

  return 0;
}
```

```
pi@kazemi:~/ch9/checker $ more c1.s
@ c1.s
@ Does nothing -- returns s
@ Sara Kazemi
@ CSIS 212

@ Define my Raspberry pi
    .cpu cortex-a53
    .fpu neon-fp-armv8
    .syntax unified

@ Program code
    .text
    .align 2
    .global c1
    .type c1, %function

c1:
    str fp, [sp, -4]!   @ save caller frame pointer
    add fp, sp, 0       @ establish our frame ptr
    mov r3, 's          @ return char s
    mov r0, r3          @ return values go in r0
    sub sp, fp, 0       @ restore stack pointer
    ldr fp, [sp], 4     @ restore callers frame pointer
    bx lr               @ back to caller
```

```
pi@kazemi:~/ch9/checker $ more c2.s
@ c2.s
@ Does nothing -- returns K
@ Sara Kazemi
@ CSIS 212

@ Define my Raspberry pi
    .cpu cortex-a53
    .fpu neon-fp-armv8
    .syntax unified

@ Program code
    .text
    .align 2
    .global c2
    .type c2, %function

c2:
    str fp, [sp, -4]!   @ save caller frame pointer
    add fp, sp, 0       @ establish our frame ptr
    mov r3, 'K          @ return K
    mov r0, r3          @ return values go in r0
    sub sp, fp, 0       @ restore stack pointer
    ldr fp, [sp], 4     @ restore callers frame pointer
    bx lr               @ back to caller
```

```
pi@kazemi:~/ch9/checker $ more c3.s
@ c3.s
@ Does nothing -- returns %
@ Sara Kazemi
@ CSIS 212

@ Define my Raspberry pi
    .cpu cortex-a53
    .fpu neon-fp-armv8
    .syntax unified

@ Program code
    .text
    .align 2
    .global c3
    .type c3, %function

c3:
    str fp, [sp, -4]!   @ save caller frame pointer
    add fp, sp, 0       @ establish our frame ptr
    mov r3, '%          @ return %
    mov r0, r3          @ return values go in r0
    sub sp, fp, 0       @ restore stack pointer
    ldr fp, [sp], 4     @ restore callers frame pointer
    bx lr               @ back to caller
pi@kazemi:~/ch9/checker $
```

-
```
pi@kazemi:~/ch9/checker $ ./checkReturned2
Result of c1(): s, Result of c2(): K, Result of c3(): %.
```