OREGON STATE UNIVERSITY

CS 472 - Computer Architecture Spring 2014

Lab 2 - The ARM ISA

Author: Drake Bridgewater Ryan Phillips

Professor:
Kevin McGrath

April 18, 2014

1 & 2) Write a simple program to perform: $Z = A + B + B - (D \times E)$. The instructions you may use are ADD, SUB, and MUL. Assume that the data is in registers r0 to r4 (representing A to E) and the result is put in r5. Enter your program into the Keil simulator and run it. You can use move instruction to load data into registers. Do you get the expected answer?

```
lab2_part1/lab2_1.asm
     AREA lab2_1, CODE, READONLY
    ENTRY
4 PART1
    MOV r0, #6
                       ; a
    MOV r1, #3
                       ; b
     MOV r2, #4
                       ;c; MOV r3, #3
                                              ; d
    MOV r4, #2
                       ; e
8
10
     ADD r1, r1, r1
                          ;r1=r1+r1 || b=b+b
12
     ADD r1, r1, r0
                          ;r1=r1+r0 \mid | b=a+(b+b)
     ; for muls you have to have diff destination register
14
     MULS r5, r3, r4
                        ;r5=r3+r4 || z=d*e
    SUB r5, r1, r5
                         ;r5=r1-r3 \mid | z=b-(d*e)
16
18 PART2
     ADR r6, A
     ADR r7, C
20
     ADR r8, D
22
     ADR r9, E
     ADR r10, F
     ADR r11, Z
24
       DCD
  Α
       DCD
26 C
            -2
       DCD
  D
             3
28 E
       DCD
            -12
  F
       DCD
             5
30 Z
       DCD
             0
32
     ADD r7, r7, r7
                          ; b=b+b
     ADD r7, r7, r6
                          ; b=a+(b+b)
34
    MULS r11, r8, r9
                            ; z=d*e
36
     SUB r11, r7, r11
                            ; z=b-(d*e)
38
```

While watching the register reflect each line of code I was able to view each commands effects on the bits. This allowed me to realize that the add operation and subtraction operations are performed exactly as expected, but once the multiplication command came into play I noticed that when the numbers are smaller minus larger the result is a negative.

END

40

3) Write a program that includes deliberate syntax errors. Enter it in the development system, assemble (build) it and then debug it.

	, ,	0	
	$ m lab2_part1/lab2_3_errors.asm$	lab	$2_{ m part}1/{ m lab}2_{ m 2}$ -fixed.asm
	AREA lab2_3_errors, CODE, READONLY		AREA lab2_3_fixed, CODE, READONLY
2	ENTRY	2	ENTRY
4	ADR rO, A	4	ADR rO, A
	ADR r2, C		ADR r2, C
6 .	ADR r4, D	6	ADR r4, D
	ADR r6, E		ADR r6, E
8	ADR r8	8	ADR r8, F
	ADR r10, Z		ADR r10, Z
10 .	A DCD 4	10 A	DCD 4
	C DCD	C	DCD -2
12	D DCD 3	12 D	DCD 3
]	E DCD -12	E	DCD -12
14	F DCD 5	14 F	DCD 5
:	Z DCD O	Z	DCD 0
16		16	
18	ADD r2,r2	18	ADD r2,r2,r2
	ADD r2,r2,r0		ADD r2,r2,r0
20		20	
	MULS r10,r6,r8		MULS r10,r6,r8
22	SUB r10,r2,r10	22	SUB r10,r2,r10
		24	END

```
assembling lab2_3_errors.asm...
lab2_3_errors.asm(6): error: A1163E: Unknown opcode r4, expecting opcode or Macro
lab2_3_errors.asm(8): error: A1106E: Missing comma
lab2_3_errors.asm(11): error: A1110E: Expected constant expression
lab2_3_errors.asm(24): warning: A1313W: Missing END directive at end of file
assembling lab2_3_fixed.asm...
".\lab2.axf" - 3 Error(s), 1 Warning(s).
```

Error on line 6: can be fixed by adding tabbing out the command out. The compiler thinks that ADR is a label and that r4 is the opcode; that is why it is saying unknown opcode r4. Error on line 8: can be fixed by adding comma but will still complain if you only add a comma. Therefore you will have to add an additional operand. Error on line 11: can be fixed by adding the expected expression, either numeric or PC-relative, as the Directive allocates one or more words of memory aligned on four-byte boundaries. Error on line 24: can be fixed by telling the compiler where the end of the program is or placing 'END' on the last line. Error on line 18: this error is not seen as an error but as a bug; the attempt is to add the two register but you must specify the Rd or destination, Rn or the first operand and operand2 or the second operand in order to sum two values.

Part 2: Examination of compiler output

In the homework, you were asked to write an assembly routine that checks for palindromes in odd length strings. For this portion of the lab, please write a simple C program that does the same thing, use the web based compiler and compile your C code to ARM assembly (use arm-linux-gnueabi-g++-4.6 with option -O0). Compare this assembly with what you wrote. Modify the optimization level from 0 to one of the values in the set 1,2,3,s. What changes? Why do you think these changes were made?

For this lab, you will need to create a write-up discussing the differences you see in Part 2. Do you see any interesting features being used? Is the hand assembly you wrote significantly different than what the compiler produced? Which uses fewer instructions? In other words, provide an analysis of Part 2.

palindrome.c

```
/*
      Author: Drake Bridgewater and Ryan Phillips
      Created: 04/16/2014
4
      Filename: palindrome.c
      Description:
     Checks an odd length string input to see if it is a palindrome.
8
10
   * Input: second command line param is an odd length string
      Return: 1 if it's a palindrome, 0 if it isn't
12 */
14 #include <stdio.h>
  #include <math.h>
16 #include <string.h>
18 void noop_message(int argc, char *argv[]){
    printf("**********palindrome.c*********\n");
    printf("****By Drake Bridgewater and Ryan Phillips****\n");
20
    printf("Usage: %s INPUT_STRING_HERE \n", argv[0]);
    printf("Note: Only accepts odd length input strings \n");
22
24 }
26 int main(int argc, char *argv[]){
    char *in_string;
    in_string = argv[1]; //
    int in_length = strlen(in_string);
    // should have just one additional arg, and it should be an odd length
32
       string
    if (argc == 1 || argc > 2 || ((in_length%2) != 1)) {
     noop_message(argc, argv);
34
    } else {
```

```
36
      int ret = 1;
      int i;
38
      for (i = 0; i < (in_length-1)/2; i++){</pre>
40
         if (in_string[i] != in_string[in_length-i-1])
          ret = 0;
      }
42
      //for testing
44
      //printf("%s %d",in_string, in_length);
      //printf("%d\n", ret);
46
    return ret;
48
    }
50 }
```

palindrome_0.s palindrome_3.s 2 # compiled from palindrome.c 2 # compiled from palindrome.c # using: http://gcc.godbolt.org/ # using: http://gcc.godbolt.org/ 4 # compiler: arm-linux-gnueabi-g 4 # compiler: arm-linux-gnueabi-g ++-4.6 ++-4.6 # optimization level: 0 # optimization level: 3 6 # 6 # .LC0: noop_message(int, char**): .ascii "********* push {r4, lr} movw r0, #:lower16:.LC0 palindrome.c movt r0, #:upper16:.LC0 *********\000" 10 .LC1: mov r4, r1 bl puts "****By Drake 10 .ascii 12 movw r0, #:lower16:.LC1 Bridgewater and Ryan Phillips****\000" movt r0, #:upper16:.LC1 14 .LC2: bl puts "Usage: %s .ascii movs r0, #1 16 INPUT_STRING_HERE \012\000" ldr r2, [r4, #0] .LC3: movw r1, #:lower16:.LC2 .ascii "Note: Only accepts odd movt r1, #:upper16:.LC2 14 length input strings \000" __printf_chk noop_message(int, char**): movw r0, #:lower16:.LC3 movt r0, #:upper16:.LC3 push $\{r7, lr\}$ sub sp, sp, #8 pop {r4, lr} add r7, sp, #0 b puts str r0, [r7, #4] main: str r1, [r7, #0] 20 push {r4, r5, r6, lr} movw r0, #:lower16:.LC0 mov r5, r0 movt r0, #:upper16:.LC0 ldr r4, [r1, #4] 22 28 mov r6, r1 bl puts 24 movw r0, #:lower16:.LC1 30 mov r0, r4 movt r0, #:upper16:.LC1 bl strlen bl puts sub r2, r5, #1 26 32 ldr r3, [r7, #0] rsbs r3, r2, #0 ldr r3, [r3, #0] adc r3, r3, r2 28 34 movw r0, #:lower16:.LC2 cmp r5, #2 30 movt r0, #:upper16:.LC2 36 it gt mov r1, r3 orrgt r3, r3, #1 cbnz r3, .L3 bl printf 32 38 movw r0, #:lower16:.LC3 lsrs r2, r0, #31 movt r0, #:upper16:.LC3 adds r1, r0, r2 34 40 bl puts and r1, r1, #1 add r7, r7, #8 subs r1, r1, r2 36 42 mov sp, r7 cmp r1, #1 pop {r7, pc} beq .L11 38 44 .L3: main: 40 push $\{r7, lr\}$ 46 mov r0, r5 sub sp, sp, #24 mov r1, r6 add r7, sp, #0 noop_message(int, char**) 42 48 bl str r0, [r7, #4] movs r0, #0 44 str r1, [r7, #0] 50 pop {r4, r5, r6, pc} ldr r3, [r7, #0]

palindrome_0.s $palindrome_3.s$.L11: ldr r3, [r3, #4] 46 subs r6, r0, #1 str r3, [r7, #16] ldr r0, [r7, #16] add r6, r6, r6, lsr #31 48 asrs r6, r6, #1 bl strlen 50 mov r3, r0 cmp r6, #0 str r3, [r7, #20] ble .L12 ldr r3, [r7, #4] adds r2, r4, r0 52 cmp r3, #1 mov r0, r1 .L8: beq .L3 54 ldr r3, [r7, #4] ldrb r5, [r4, r3] cmp r3, #2 zero_extendqisi2 56 bgt .L3 adds r3, r3, #1 ldr r2, [r7, #20] ldrb r1, [r2, #-1]! 58 62 asr r3, r2, #31 zero_extendqisi2 lsr r3, r3, #31 cmp r5, r1 60 adds r2, r2, r3 64 it ne and r2, r2, #1 movne r0, #0 62 subs r3, r2, r3 cmp r3, r6 66 bne .L8 cmp r3, #1 beq .L4 68 pop {r4, r5, r6, pc} 66 .L3: .L12: ldr r0, [r7, #4] 70 mov r0, r1 ldr r1, [r7, #0] pop {r4, r5, r6, pc} 72 .LCO: bl noop_message(int, char**) "********* mov r3, #0 .ascii b .L5 palindrome.c 72 .L4: *********\000" 74 .LC1: mov r3, #1 str r3, [r7, #8] "****By Drake 74 .ascii mov r3, #0 Bridgewater and Ryan str r3, [r7, #12] Phillips****\000" 76 b .L6 76 .LC2: 78 .L8: .ascii "Usage: %s ldr r3, [r7, #12] INPUT_STRING_HERE \012\000" 78 .LC3: ldr r2, [r7, #16] 80 "Note: Only accepts odd adds r3, r2, r3 ldrb r2, [r3, #0] 82 length input strings \000" zero_extendqisi2 ldr r1, [r7, #20] ldr r3, [r7, #12] subs r3, r1, r3 add r3, r3, #-1 86 ldr r1, [r7, #16] adds r3, r1, r3 88 ldrb r3, [r3, #0] zero_extendqisi2 90 cmp r2, r3

```
palindrome\_0.s
     cmp r2, r3
     beq .L7
     mov r3, #0
92
     str r3, [r7, #8]
94 . L7:
     ldr r3, [r7, #12]
     add r3, r3, #1
     str r3, [r7, #12]
98 .L6:
     ldr r3, [r7, #20]
     add r3, r3, #-1
100
     lsr r2, r3, #31
     adds r3, r2, r3
102
     asr r3, r3, #1
     mov r2, r3
104
     ldr r3, [r7, #12]
106
     cmp r2, r3
     ite le
     movle r3, #0
108
     movgt r3, #1
     uxtb r3, r3
110
     cmp r3, #0
     bne .L8
112
     ldr r3, [r7, #8]
114 . L5:
     mov r0, r3
116
     add r7, r7, \#24
```

mov sp, r7

pop {r7, pc}

118