Oregon State University

CS 472 - Computer Architecture

Spring 2014

Lab 2 - The ARM ISA

Author:
Drake Bridgewater
Ryan Phillips

Professor:
Kevin McGrath

April 27, 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

PART1
  MOV r0, #6       ;a
  MOV r1, #3       ;b
  MOV r2, #4       ;c; MOV r3, #3       ;d
  MOV r4, #2       ;e


  ADD r1,r1,r1       ;r1=r1+r1 || b=b+b
  ADD r1,r1,r0       ;r1=r1+r0 || b=a+(b+b)

  ;for muls you have to have diff destination register
  MULS r5,r3,r4      ;r5=r3+r4 || z=d*e
  SUB r5,r1,r5       ;r5=r1-r3 || z=b-(d*e)

PART2
  ADR r6, A
  ADR r7, C
  ADR r8, D
  ADR r9, E
  ADR r10, F
  ADR r11, Z
A   DCD  4
C   DCD  -2
D   DCD  3
E   DCD  -12
F   DCD  5
Z   DCD  0


  ADD r7,r7,r7       ; b=b+b
  ADD r7,r7,r6       ; b=a+(b+b)

  MULS r11,r8,r9       ; z=d*e
  SUB r11,r7,r11       ; z=b-(d*e)


  END
```

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.

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

lab2_part1/lab2_3_errors.asm

```
    AREA lab2_3_errors, CODE,
       READONLY
    ENTRY

  ADR r0, A
  ADR r2, C
ADR r4, D
  ADR r6, E
  ADR r8
  ADR r10, Z
A   DCD  4
C   DCD
D   DCD  3
E   DCD  -12
F   DCD  5
Z   DCD  0


  ADD r2,r2
  ADD r2,r2,r0

  MULS r10,r6,r8
  SUB r10,r2,r10
```

lab2_part1/lab2_3_fixed.asm

```
    AREA lab2_3_fixed, CODE,
       READONLY
    ENTRY

    ADR r0, A
    ADR r2, C
    ADR r4, D
    ADR r6, E
    ADR r8, F
    ADR r10, Z
A   DCD  4
C   DCD  -2
D   DCD  3
E   DCD  -12
F   DCD  5
Z   DCD  0


    ADD r2,r2,r2
    ADD r2,r2,r0

    MULS r10,r6,r8
    SUB r10,r2,r10

    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.

lab2_part2/palindrome.c

```
/*
 *  Author:  Drake Bridgewater and Ryan Phillips
 *  Created:  04/16/2014
 *  Filename:  palindrome.c
 *
 *  Description:
 *
 *  Checks an odd length string input to see if it is a palindrome.
 *
 *  Input: second command line param is an odd length string
 *  Return: 1 if it's a palindrome, 0 if it isn't
*/

#include <stdio.h>
#include <math.h>
#include <string.h>

void noop_message(int argc, char *argv[]){
  printf("**************palindrome.c************\n");
  printf("****By Drake Bridgewater and Ryan Phillips****\n");
  printf("Usage: %s INPUT_STRING_HERE \n",argv[0]);
  printf("Note: Only accepts odd length input strings \n");

}

int main(int argc, char *argv[]){

  if (argc == 1 || argc > 2) { noop_message(argc,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
     string
  if ( (in_length%2) != 1){
```

```
36      noop_message(argc, argv);
    } else {
38
      int ret = 1;
40    int i;
      for (i = 0; i < (in_length-1)/2; i++){
42      if (in_string[i] != in_string[in_length-i-1])
          ret = 0;
44    }

46    //for testing
      printf("%s %d",in_string, in_length);
48    printf("%d\n", ret);
    return ret;
50
    }
52 }
```

lab2_part2/palindrome_0.asm

```
#
# compiled from palindrome.c
# using: http://gcc.godbolt.org/
# compiler: arm-linux-gnueabi-g++-4.6
# optimization level: 0
#
.LC0:
  .ascii  "**************palindrome.c************\000"
.LC1:
  .ascii  "****By Drake Bridgewater and Ryan Phillips****\000"
.LC2:
  .ascii  "Usage: %s INPUT_STRING_HERE \012\000"
.LC3:
  .ascii  "Note: Only accepts odd length input strings \000"
noop_message(int, char**):
  push  {r7, lr}
  sub sp, sp, #8
  add r7, sp, #0
  str r0, [r7, #4]
  str r1, [r7, #0]
  movw  r0, #:lower16:.LC0
  movt  r0, #:upper16:.LC0
  bl  puts
  movw  r0, #:lower16:.LC1
  movt  r0, #:upper16:.LC1
  bl  puts
  ldr r3, [r7, #0]
  ldr r3, [r3, #0]
  movw  r0, #:lower16:.LC2
  movt  r0, #:upper16:.LC2
  mov r1, r3
  bl  printf
  movw  r0, #:lower16:.LC3
  movt  r0, #:upper16:.LC3
  bl  puts
  add r7, r7, #8
  mov sp, r7
  pop {r7, pc}
main:
  push  {r7, lr}
  sub sp, sp, #24
  add r7, sp, #0
  str r0, [r7, #4]
  str r1, [r7, #0]
  ldr r3, [r7, #0]
```

lab2_part2/palindrome_3.asm

```
#
# compiled from palindrome.c
# using: http://gcc.godbolt.org/
# compiler: arm-linux-gnueabi-g++-4.6
# optimization level: 3
#
noop_message(int, char**):
  push  {r4, lr}
  movw  r0, #:lower16:.LC0
  movt  r0, #:upper16:.LC0
  mov r4, r1
  bl  puts
  movw  r0, #:lower16:.LC1
  movt  r0, #:upper16:.LC1
  bl  puts
  movs  r0, #1
  ldr r2, [r4, #0]
  movw  r1, #:lower16:.LC2
  movt  r1, #:upper16:.LC2
  bl  __printf_chk
  movw  r0, #:lower16:.LC3
  movt  r0, #:upper16:.LC3
  pop {r4, lr}
  b puts
main:
  push  {r4, r5, r6, lr}
  mov r5, r0
  ldr r4, [r1, #4]
  mov r6, r1
  mov r0, r4
  bl  strlen
  sub r2, r5, #1
  rsbs  r3, r2, #0
  adc r3, r3, r2
  cmp r5, #2
  it  gt
  orrgt r3, r3, #1
  cbnz  r3, .L3
  lsrs  r2, r0, #31
  adds  r1, r0, r2
  and r1, r1, #1
  subs  r1, r1, r2
  cmp r1, #1
  beq .L11
.L3:
  mov r0, r5
  mov r1, r6
  bl  noop_message(int, char**)
  movs  r0, #0
  pop {r4, r5, r6, pc}
```

lab2_part2/palindrome_0.asm

```
  ldr r3, [r3, #4]
  str r3, [r7, #16]
  ldr r0, [r7, #16]
  bl  strlen
  mov r3, r0
  str r3, [r7, #20]
  ldr r3, [r7, #4]
  cmp r3, #1
  beq .L3
  ldr r3, [r7, #4]
  cmp r3, #2
  bgt .L3
  ldr r2, [r7, #20]
  asr r3, r2, #31
  lsr r3, r3, #31
  adds  r2, r2, r3
  and r2, r2, #1
  subs  r3, r2, r3
  cmp r3, #1
  beq .L4
.L3:
  ldr r0, [r7, #4]
  ldr r1, [r7, #0]
  bl  noop_message(int, char**)
  mov r3, #0
  b .L5
.L4:
  mov r3, #1
  str r3, [r7, #8]
  mov r3, #0
  str r3, [r7, #12]
  b .L6
.L8:
  ldr r3, [r7, #12]
  ldr r2, [r7, #16]
  adds  r3, r2, r3
  ldrb  r2, [r3, #0]  @
     zero_extendqisi2
  ldr r1, [r7, #20]
  ldr r3, [r7, #12]
  subs  r3, r1, r3
  add r3, r3, #-1
  ldr r1, [r7, #16]
  adds  r3, r1, r3
  ldrb  r3, [r3, #0]  @
     zero_extendqisi2
  cmp r2, r3
```

lab2_part2/palindrome_3.asm

```
.L11:
  subs  r6, r0, #1
  add r6, r6, r6, lsr #31
  asrs  r6, r6, #1
  cmp r6, #0
  ble .L12
  adds  r2, r4, r0
  mov r0, r1
.L8:
  ldrb  r5, [r4, r3]  @
     zero_extendqisi2
  adds  r3, r3, #1
  ldrb  r1, [r2, #-1]!  @
     zero_extendqisi2
  cmp r5, r1
  it  ne
  movne r0, #0
  cmp r3, r6
  bne .L8
  pop {r4, r5, r6, pc}
.L12:
  mov r0, r1
  pop {r4, r5, r6, pc}
.LC0:
  .ascii  "**************
     palindrome.c
     ************\000"
.LC1:
  .ascii  "****By Drake
     Bridgewater and Ryan
     Phillips****\000"
.LC2:
  .ascii  "Usage: %s
     INPUT_STRING_HERE \012\000"
.LC3:
  .ascii  "Note: Only accepts odd
      length input strings \000"
```

lab2_part2/palindrome_0.asm

```
  cmp r2, r3
  beq .L7
  mov r3, #0
  str r3, [r7, #8]
.L7:
  ldr r3, [r7, #12]
  add r3, r3, #1
  str r3, [r7, #12]
.L6:
  ldr r3, [r7, #20]
  add r3, r3, #-1
  lsr r2, r3, #31
  adds  r3, r2, r3
  asr r3, r3, #1
  mov r2, r3
  ldr r3, [r7, #12]
  cmp r2, r3
  ite le
  movle r3, #0
  movgt r3, #1
  uxtb  r3, r3
  cmp r3, #0
  bne .L8
  ldr r3, [r7, #8]
.L5:
  mov r0, r3
  add r7, r7, #24
  mov sp, r7
  pop {r7, pc}
```

There are a few important differences between the hand-written assembly code and the assembly code generated from the .c source. The generated code is less readable, mainly because it uses default numbered labels which yield no information about the code within these sections. Also, the hand-written assembly is much shorter in length at 36 lines. The unoptimized, generated code is over 100 lines, and even the optimized, generated code is over twice as long at 80 lines.