CS2002 Practical 2 - x86 Assembler

Matriculation Number: 160001362

27/02/2018

Contents

1 Overview

This practical specified the commenting of AT&T x86 64-bit assembly code, the implementation of a print function to print three stack frames during the execution of a sort algorithm (and to write an analysis of the output), and an analysis of the functions of sort0.s (and how sort1.s, sort2.s, and sort3.s differ from this file). The practical also specified that as an extension the sort.c source file should be compiled for other architectures such as MIPS and ARM, and that the results should be analysed and compared with x86-64 assembly.

The first stage of the practical has been achieved with the assembly in the sort-commented.s file having been commented for the sortsub function.

The functionality requested by the second stage of the practical has been implemented in the stack.c file in the print_stack and print_frame functions which can be run in terminal by using the command ./main-plus which will print the address, offset and value of each value in each stack frame. Analysis of the results of this function can be found in the analysis section further into this report.

The third section of this practical, analysis of the functions of sort0.s and a comparison between this file and the other sort files, can also be found in the analysis section of this report.

The extension comparing MIPS and ARM assembly with x86-64 assembly is discussed under the 'extensions' heading of this report.

2 Design and Implementation

In the second section of the practical, the stack_frame function runs an in-line assembly command to move the value stored in the base pointer register (rbp) into a local variable. This variable is then dereferenced to retrieve the value of the base pointer from the previous stack frame. My own function print_frame is then called with the two base pointers and the number of stack frames to print as arguments.

The print_frame function then iterates over the addresses between the previous base pointer and the current base pointer in increments of 8 bytes and prints out the address of each value, as well as its offset from the base pointer in bytes, and the value stored at the current address. The print_frame function is then recursively called until there are no more frames to print.

```
ADDRESS,        OFFSET,  VALUE OF QUADWORD
7ffe03b0fbb0, -0(%rbp), 140728960351232
7ffe03b0fba8, -8(%rbp), 140728960351456
7ffe03b0fba0, -16(%rbp), 8
7ffe03b0fb98, -24(%rbp), 2
7ffe03b0fb90, -32(%rbp), 5
7ffe03b0fb88, -40(%rbp), 4196048
7ffe03b0fb80, -48(%rbp), 3
7ffe03b0fb78, -56(%rbp), 2
7ffe03b0fb70, -64(%rbp), 0
7ffe03b0fb68, -72(%rbp), 4196393

ADDRESS,        OFFSET,  VALUE OF QUADWORD
7ffe03b0fc00, -0(%rbp), 140728960351312
7ffe03b0fbf8, -8(%rbp), 140728960351456
7ffe03b0fbf0, -16(%rbp), 8
7ffe03b0fbe8, -24(%rbp), 0
7ffe03b0fbe0, -32(%rbp), 5
7ffe03b0fbd8, -40(%rbp), 4196048
7ffe03b0fbd0, -48(%rbp), 2
7ffe03b0fbc8, -56(%rbp), 0
7ffe03b0fbc0, -64(%rbp), 0
7ffe03b0fbb8, -72(%rbp), 4196317

ADDRESS,        OFFSET,  VALUE OF QUADWORD
7ffe03b0fc50, -0(%rbp), 140728960351376
7ffe03b0fc48, -8(%rbp), 140728960351456
7ffe03b0fc40, -16(%rbp), 8
7ffe03b0fc38, -24(%rbp), 0
7ffe03b0fc30, -32(%rbp), 10
7ffe03b0fc28, -40(%rbp), 4196048
7ffe03b0fc20, -48(%rbp), 5
7ffe03b0fc18, -56(%rbp), 0
7ffe03b0fc10, -64(%rbp), 57680252435528
7ffe03b0fc08, -72(%rbp), 4196292
```

Figure 1: Output of print_stack Function

3 Analysis

3.1 Section 2 - Findings

The above figure shows the terminal output of running the main-plus executable with the specified implementation of the print_stack function. The output has been displayed in the suggested format of address, offset and value. For the most part the values output seem to correspond with values in the assembly and C code.

The first value in each stack frame is a decimal address which makes sense as it has an offset of 0 from the base pointer and therefore is the value at that current base pointer. The value in the first stack frame printed is decimal 140728960351232 which is the address of the middle stack frame in hex 7ffe03b0fc00. The second value (offset -8) is also a decimal address which corresponds to the void* array as the names of arrays in C are pointers to the first element of the array. The third, fourth and fifth values (offsets -16, -24 and -32) correspond to the size, left and right variables of type long. For the sixth value (offset -40) it appears to be the address pointing to the function comp_long used to initialise the comp variable. However this is not certain. The seventh value output (offset -48) seems to be the value of mid which is being passed into subsort, subsort and merge respectively and is used to split the list into two smallers halves recursively. The eighth value (offset -56) is the same value as offset -24 which contains the value of left, this is because this location is where the value of left is stored before it is copied into register rcx for use in the calculation during the assignment of mid where:

mid = left + (right - left) / 2.

The last two results are harder to explain. The second last value (offset -64) varies between the value 0 and a large integer 57680252435528. This value is not referenced in the assembly code but may be additional space allocated for the stack frame. It appears to be empty in the first two stack frames but may have been allocated an address or similar value by the operating system or other process. The final value (offset -72) appears to be a similar value for each stack frame and is likely the return address of the previous frame on the stack.

3.1.1 Comparison of sort0.s and sort-plus.s

```
 92   .LBB1_2:
 93       movq    -32(%rbp), %rax
 94       subq    -24(%rbp), %rax
 95       cmpq    $3, %rax
 96       jne .LBB1_6
 97       cmpq    $0, -24(%rbp)
 98       jle .LBB1_6
 99       cmpq    $10, -32(%rbp)
100       jge .LBB1_6
101       movb    $0, %al
102       callq   print_stack
```

Aside from minor differences between assembler directives in the sort0.s and sort-plus.s files, the above seems to be the only major difference between the two files. This, as suggested by the specification, shows that global changes to the assembly implementing the sort-plus.c file have occurred with the addition of a print_stack call. This would change the layout of the stack frames in the program with an additional frame for each call of print_stack at the end of each sort_sub function run. The code appears to be allocating space for the call of print_stack with the subq command and is assessing the conditional statement at the end of the sortsub function to determine whether print_stack should be called.

3.2 Section 3

3.2.1 Word Size Analysis

```
132  .LBB2_1:
133      movb    $1, %al
134      movq    -72(%rbp), %rcx
135      cmpq    -32(%rbp), %rcx
136      movb    %al, -89(%rbp)
137      jl  .LBB2_3
138      movq    -80(%rbp), %rax
139      cmpq    -40(%rbp), %rax
140      setl    %cl
141      movb    %cl, -89(%rbp)
142  .LBB2_3:
143      movb    -89(%rbp), %al
144      testb   $1, %al
```

The above shows operations involving (single) words in sort0.s (operations include movb and testb). These operations act on values related to the conditional statement in the while loop of the merge function. Lines 133-136 compare the size of p1 to mid (and store 1 in offset -89 of rbp if true) and lines 138-141 compare the size of p2 and right (and store the result in offset -89 of rbp). Line 137 skips the second comparison if the first is true and moves to .LBB2_3 which moves the overall comparison result to the register al and then line 144 does a bitwise AND comparison between the value 1 (true) and the value returned from the loop condition. The code jumps to one location if the result is true (to continue the loop) and another if the result is false. Words are used instead of quadwords as only the values 0 and 1 need to be stored for false and true and bytes are the smallest addressable unit of memory.

3.2.2 Assembler Code for Comp Function Call

3.2.3 Implementation of a While-Loop in Assembler

3.2.4 Comparison of Sort Files

4 Evaluation

5 Conclusion

6 Section 4 - Extension