Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Fall Term 2018

SYSTEMS PROGRAMMING AND COMPUTER ARCHITECTURE
Assignment 5: Assembly

Assigned on: **24th Oct 2018**
Due by: **30th Oct 23:59 2018**

NOTE: Unless otherwise stated, the assembly code in this assignment is x86-64 assembly.

1 Assembly basics

1.1 Array basics

For each of the arrays declared below provide:

a) the size of one element in bytes,

b) the total size of the array in bytes,

c) the byte address of element i if the array starts at address $x_{<arrayindentifier>}$,

d) two different C expressions for accessing element i of the array.

e) a C expression that dereferences an actual char, short, or int, at index 2 (index (2,0), index (2,0,0) respectively) of the array (i.e. char value = A[2]).

```
char    A[5];
char  **C[8];
short  *E[9];
int    *G[7];
char   *B[3];
short   D[4];
int     F[4];
```

1.2 Addressing modes

Assume the following values are stored at the indicated memory addresses and registers:

Address	Value
0x204	0xFF
0x208	0xCD
0x20C	0x21
0x210	0x11

Register	Value
%rax	0x2
%rcx	0x204
%rdx	0x3

Fill in the following table showing the types (i.e., immediate, register, memory) and the values of the indicated operands:

Operand	Type	Memory address	Value
%rax			
0x210			
$0x210			
(%rcx)			
4(%rcx)			
5(%rcx, %rdx)			
519(%rdx, %rax)			
0x204(, %rax, 4)			
(%rcx, %rax, 2)			

1.3 Arithmetic operations

Use the values of the memory addresses and registers from Question 1. Handle the different instructions independently. The result of one of the instructions does not affect the others. Fill in the following table showing the effects of the following instructions, both in terms of the register or memory location that will be updated and the resulting value:

Instruction	Destination	Computation and result
`addl %eax, (%rcx)`		
`subl %edx, 4(%rcx)`		
`imull (%rcx, %rax, 4), %eax`		
`incl 8(%rcx)`		
`decl %eax`		
`subl %edx, %ecx`		

1.4 leal and movl

Assume the following values are stored at the indicated memory addresses and registers:

Address	Value
0x108	0xFF
0x10C	0xCD
0x110	0x21

Register	Value
%rax	0x100
%rcx	0x4
%rdx	0x1

What is the difference between the two instructions? What value ends up in %ecx? Write the formula!

```
movl 8(%rax, %rdx, 4), %ecx
```

```
leal 8(%rax, %rdx, 4), %ecx
```

1.5 Condition codes

Consider the instruction `addl` %rax, %rbx. As a side-effect, it sets the condition flags (OF, SF, ZF, CF) according to the result.

Assuming a 4-bit machine, convert the given decimal pairs `(a, b)` to their binary representation and perform the addition. Give both the arithmetical value and the interpreted value (2's complement) of the result. List the condition flags that are set.

(-1, -1), (+4, TMin), (TMax, TMax), (TMax, -TMax),
(TMin, TMax), (TMin, TMin), (-1, TMax), (2, 3).

1.6 Reading Condition Codes with C

In this exercise you will obtain and print the processor's condition codes for different assembly instructions using a C program. To facilitate your task we have prepared a program skeleton that already does most of the work (`ccodes.c`). You can download the skeleton (`ccodes.c`) from the course web page.

On Intel processors the condition codes are stored in the 64-bit wide RFLAGS register. The program skeleton first executes an assembly instruction and stores the resulting contents of the RFLAGS register to a variable. **Your task is to complete the function getccodes()** (no other part of the program needs to be modified). This function extracts the four condition codes of interest (sign flag, carry flag, zero flag, overflow flag) from the RFLAGS register and stores their values to a C struct of type `struct ccodes`.

The layout of the RFLAGS register is described in the Intel Architecture Software Developer's Manual (Volume 1, Section 3.4.3); the link to the Intel manuals is indicated on the course web page.

When the program is complete, compile and run it to test if it functions properly. For example, the command may look like this: `gcc -Wall -Wextra ccodes.c -o ccodes`. You can also add new test cases to the `main()` function.

ADVICE: All required tools to compile the programs should be installed on the lab machines. If you want to build it on your own machine, make sure to install gcc and gcc-multilib (e.g. on Ubuntu:
`sudo apt-get install build-essential gcc-multilib`).

2 Assembly control flow

2.1 Assembly Code Fragments

Consider the following pairs of C functions and assembly code. Fill in the missing instructions in the assembly code fragments (one instruction per blank). Your answers should be correct x86_64 assembly code.

a)
```
int f1(int a, int b) {

  return a - b;

}
```
```
f1:  pushq   %rbp
     movq    %rsp, %rbp
     __________________
     __________________
     movq    %rbp, %rsp
     popq    %rbp
     ret
```

b)
```
int f2(int a) {



  return a*5;

}
```
```
f2:  pushq   %rbp
     movq    %rsp, %rbp
     leal    ______________
     movq    %rbp, %rsp
     popq    %rbp
     ret
```

c)
```
int f3(int a) {


  if (a <= 0)
    return -a;
  else
    return a;

}
```
```
f3:   pushq   %rbp
      movq    %rsp, %rbp
      ____________________
      movl    %edi, %eax
      jle     .L11
.L8:  movq    %rbp, %rsp
      popq    %rbp
      ret
.L11: negl    %eax
      jmp     .L8
```

2.2 Conditional branches

What is the value of %eax, when the last label (respectively `.L3` and `.L17`) is reached? First, annotate the assembly code and then, write the corresponding C-statements!

i) Assume %eax := `a`, %edx := `d`.

```
        ...
        cmpl    %eax, %edx
        jle     .L2
        subl    %eax, %edx
        movl    %edx, %eax
        jmp     .L3
.L2:
        subl    %edx, %eax
.L3:
        ...
```

ii) Assume %eax := `1`, %ecx := `N`.

```
        ...
        testl   %ecx, %ecx
        jle     .L17
        xorl    %edx, %edx
.L18:
        incl    %edx
        addl    %eax, %eax
        cmpl    %edx, %ecx
        jne     .L18
.L17:
        ...
```

2.3 For Loop

This problem tests your understanding of how for loops in C relate to machine code. Consider the following x86_64 assembly code for a procedure `dog()`.

Based on the assembly code, fill in the blanks in its corresponding C source code. (Note: you may only use symbolic variables x, y, i, and result from the source code in your expressions below. Do not use register names.)

```
dog:
        movl    $1, %eax
        cmpl    %esi, %edi
        jge     .L2
.L1:
        imull   %edi, %eax
        addl    $2, %edi
        cmpl    %esi, %edi
        jl      .L1
.L2:
        retq
```

```
int dog(int x, int y)
{
  int i, result;
  result = _____;
  for (i = _____; __________; _____)
  {
    result = _________________;
  }
  return result;
}
```

2.4 Switch Statement

Consider the following C function and three assembly code fragments. Which of the assembly code fragments matches the C function shown?

C Code

```
int woohoo(int a)
{
  int ret = 0;
  switch(a) {
  case 11:
    ret = 4;
    break;
  case 22:
  case 55:
    ret = 7;
    break;
  case 33:
  case 44:
    ret = 11;
    break;
  default:
    ret = 1;
  }
  return ret;
}
```

Fragment 1

```
woohoo: movl $0, %ecx
        cmpl $11, %edi
        jne .L2
        movl $4, %ecx
        jmp .L3
.L2:    cmpl $22, %edi
        jne .L3
        movl $7, %ecx
.L3:    cmpl $55, %edi
        jne .L5
        movl $7, %ecx
.L5:    cmpl $33, %edi
        sete %al
        cmpl $44, %edi
        sete %dl
        orl %edi, %eax
        testb $1, %al
        je .L6
        movl $11, %ecx
.L6:    movl %ecx, %eax
        ret
```

Fragment 2

```
woohoo: subl $1, %edi
        movl $1, %eax
        cmpl $4, %edi
        ja .L2
        jmp *.L9(,%edi,4)
        .section .rodata
        .align 4
.L9:    .long .L3
        .long .L5
        .long .L7
        .long .L7
        .long .L5
        .text
.L3:    movl $4, %eax
        jmp .L2
.L5:    movl $7, %eax
        jmp .L2
.L7:    movl $11, %eax
.L2:    ret
```

Fragment 3

```
woohoo: subl $11, %edi
        je .L6
        subl $11, %edi
        je .L7
        subl $11, %edi
        je .L8
        subl $11, %edi
        je .L8
        subl $11, %edi
        je .L7
        jmp .L9
.L6:    movl $4, %eax
        jmp .L4
.L7:    movl $7, %eax
        jmp .L4
.L8:    movl $11, %eax
        jmp .L4
.L9:    movl $1, %eax
.L4:    ret
```

Hand In Instructions

Except Question 1.6, this is a paper exercise. If you want your solution to be revised please hand it in during your exercise class on the due date. Upload your `ccodes.c` (Question 1.6) to a subfolder **assignment5** of your SVN folder. Refer to Assignment 1 for instructions on using SVN.