Imperial College
London

TUTORIAL

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

C113 Architecture

Lecturer:
Dr. Jana Giceva
j.giceva@imperial.ac.uk

Head Teaching Assistant:
Izaak Coleman
ic711@imperial.ac.uk

Date: February 17, 2018

1 Assembly Programming

1.1 Arithmetic Operations

Assume the following values are stored at the given 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 given 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. Please handle the different instructions independently. The result of one of the instructions does not affect the others.

Instruction	Dest.	Computation and Result
`addl %eax, (%rcx)`	0x204	0x2 + 0xFF = 0x101
`subl %edx, 4(%rcx)`	0x208	0xCD - 0x3 = 0xCA
`imull (%rcx, %rax, 4),%eax`	%eax	0x2 * 0x21 = 0x42
`incl 8(%rcx)`	0x20C	0x21 + 0x1 = 0x22
`decl %eax`	%eax	0x2 - 0x1 = 0x1
`subl %edx, %ecx`	%ecx	0x204 - 0x3 = 0x201

1.2 `leal` and `movl` instructions

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 following two instructions? What value is stored in %ecx in the end? Write the formulas!

`movl 8(%rax, %rdx, 4), %ecx`
`movl` loads into %rdx the value residing in the address computed by the expression %rax + 4*%rdx+8. Therefore, the value in %ecx is 0xCD.

`leal 8(%rax, %rdx, 4), %ecx`
`leal` loads into %ecx the computed value of the expression %rax+4*%rdx+8. The value in %ecx is 0x10C.

1.3 Logical operations

The portion of the generated assembly code implementing these expressions is:

```
1: # x at %ebp+8, y at %ebp+12, z at %ebp+16
2:
3: movl 12(%ebp), %eax
4: xorl 8(%ebp), %eax
5: sarl $3, %eax
6: notl %eax
7: subl 16(%ebp), %eax
```

Based on this assembly code, fill in the missing expressions in the Java/C code.

Answer:

```
int logic_exp (int x, int y, int z)
{
    int t1 = y^x ;
    int t2 = t1 >> 3 ;
    int t3 = ~t2 ;
    int t4 = t3 - z ;
    return t4;
}
```

1.4 Assembly Code Fragments

Consider the following pairs of procedures (in Java/C) and assembly code. Fill in the missing instructions in the assembly code fragments (one instruction per blank).

Answer:

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

      return a - b;

  }
```

```
f1:  pushq %rbp
     movq  %rsp, %rbp
     movl  %edi, %eax
     subl  %esi, %eax
     movq  %rbp, %rsp
     popq  %rbp
     ret
```

```
b int f2(int a) {




      return a+5;
  }
```

```
f2:  pushq %rbp
     movq  %rsp, %rbp
     leal  5(%rdi), %eax
     movq  %rbp, %rsp
     pop   %rbp
     ret
```

```
c int f3(int a) {

      if (a <= 0)
          return -a;

      else
          return a;
```

```
 f3:  pushq %rbp
      movq  %rsp, %rbp
      cmpl  $0, %edi
      movl  %edi, %eax
      jle   .L11
.L8:  movq  %rbp, %rsp
      popq  %rbp
      ret
.L11: negl  %eax
      jmp   .L8
```

1.5 For loop

This problem tests your understanding of how `for` loops 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 the corresponding Java/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.).

Answer:

```
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 = 1;
    for (i = 0; x < y; x=+2)
    {
        result = result * x;
    }
    return result;
}
```

1.6 Switch statement

Which of the following code fragments matches the Java/C function shown?
Answer: Assembly code `sw_3` correctly implements the given `sw` function.

```
int sw(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;
}
```

```
sw_1:   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 .L8
        .long .L5
        .text
 .L3:   movl  $4, %eax
        jmp   .L2
 .L5:   movl  $7, %eax
        jmp   .L2
 .L7:   movl  $11, %eax
 .L2:   ret
```

```
sw_2:   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
```

```
sw_3:   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
```