CS201
Homework 03

Le Duong Cong Duc
Student ID: 1651044

December 12, 2017

Contents

1 Exercises 1

Consider the following source code, where R, S, and T are constants declared with #*define*:

```
long A[R][S][T];
long store_ele(long i, long j, long k, long *dest)
{
        *dest = A[i][j][k];
        return sizeof(A);
}
```

Assembly code:

```
;long store_ele(long i, long j, long k, long *dest)
;i in %rdi, j in %rsi, k in %rdx, dest in %rcx
store_ele :
        leaq (%rsi,%rsi,2), %rax
        leaq (%rsi,%rax,4), %rax
        movq %rdi, %rsi
        salq $6, %rsi
        addq %rsi, %rdi
        addq %rax, %rdi
        addq %rdi, %rdx
        movq A(,%rdx,8), %rax
        movq %rax, (%rcx)
        movl $3640, %eax
        ret
```

Determine the values of R, S, and T.

Answer:

1. R = 7
2. S = 5
3. T = 13

2 Exercises 2

The code that follows shows an example of branching on an enumerated type value in a switch statement. Recall that enumerated types in C are simply a way to introduce a set of names having associated integer values. By default, the values assigned to the names count from zero upward. In our code, the action associated with the different case labels have been omitted.

```
//Fill the missing parts

/*Enumerated type creates set of constants numbered 0 and upward */
typedef enum {MODE_A, MODE_B, MODE_C, MODE_D, MODE_E} mode_t;

long switch3(long *p1, long *p2, mode_t action)
{
        long result = 0;
        switch(action) {
                case MODE_A:
                    result = *p2;
                    action = *p1;
                    *p2 = action;
                    break;
                case MODE_B:
                    result = *p1;
                    result = result + *p2;
                    *p1 = *p1 + *p2;
                    break;
                case MODE_C:
                    *p1 = 59;
                    result = *p2;
                    break;
                case MODE_D:
                    result = *p2;
                    *p1 = result;
```

```
            case MODE_E:
                result = 27;
                break;
            default:
                result = 12;
        }
        return result;
}
```

The part of the generated assembly code implementing the different actions is shown as the code below. The annotations indicate the argument locations, the register values, and the case labels for the different jump destinations.

```
        ;p1 in %rdi, p2 in %rsi, action in %edx
.L8:                                        ;MODE_E
        movq    $27, %eax
        ret
.L3:                                        ;MODE_A
        movq    (%rsi), %rax
        movq    (%rdi), %rdx
        movq    %rdx, (%rsi)
        ret
.L5:                                        ;MODE_B
        movq    (%rdi), %rax
        addq    (%rsi), %rax
        movq    %rax, (%rdi)
        ret
.L6:                                        ;MODE_C
        movq    $59, (%rdi)
        movq    (%rsi), %rax
        ret
.L7:                                        ;MODE_D
        movq    (%rsi), %rax
```

```
        movq    %rax, (%rdi)
        movq    $27, %eax
        ret
.L9:                            ;default
        movl    $12, %eax
        ret
```

Fill in the missing parts of the C code. It contained one case that fell through to another–try to reconstruct this.

3 Exercises 3

Consider the following source code, where NR and NC are macro expressions declared with *#define* that compute the dimensions of array A in terms of parameter n. This code computes the sum of the elements of column j of the array.

```
long sum_col(long n, long A[NR(n)][NC(n)], long j) {
        long i;
        long result = 0;
        for (i = 0; i < NR(n); i++)
                result += A[i][j];
        return result;
}
```

Assembly code for the body of setVal:

```
;long sum_col (long n, long A[NR(n)][NC(n)], long j)
;n in %rdi, A in %rsi, j in %rdx
sum_col:
        leaq 1(,%rdi,4), %r8
        leaq (%rdi,%rdi,2), %rax
        movq %rax, %rdi
        testq %rax, %rax
        jle .L4
        salq $3, %r8
```

```
        leaq (%rsi,%rdx,8), %rcx
        movl $0, %eax
        movl $0, %edx
.L3:
        addq (%rcx) , %rax
        addq $1, %rdx
        addq %r8, %rcx
        cmpq %rdi, %rdx
        jne .L3
        rep; ret
.L4:
        movl $0, %eax
        ret
```

Determine the definition of NR and NC.

Answer:

1. NR = 3n
2. NC = 4n + 1

4 Exercises 4

For each of the following structure declarations, determine the offset of each field, the total size of the structure, and its alignment requirement for x86-64.

1. `struct P1 { int i; char c; int j ; char d; } ;`
2. `struct P2 { int w[2]; char c[3]; long j[2]; } ;`
3. `struct P3 { short *w[3]; char c[3]; } ;`
4. `struct P4 { struct P1 a[2]; char c[5]; struct P2 t;} ;`

Answer:

1.

P1	i	c	j	d	Total	Alignment
	0	4	8	12	16	4

2.

P2	w	c	j	Total	Alignment
	0	8	16	32	8

3.

P3	w	c	Total	Alignment
	0	24	32	8

4.

P4	a	c	t	Total	Alignment
	0	32	40	72	8