CS 367 - Recitation #8 **Data Structures**

Name Adilet Kuroda G#

Group Member Name: Kidist Nida

Group Member Name:

Today's Goals: We want to get comfortable with data structures in assembly and pointer arithmetic.

Work in groups of 2-3 students. Every group will turn in what they've got to Blackboard.

Arrays

A) Given an array with this below declaration, answer the following questions.

int gmu[5]; *// The starting address for gmu is 0x100*

1) What is the total size of the array gmu in bytes?

20

2) Give a numeric (mathematical) expression to access the address of gmu[i]. *(Remember, you are working with bytes)*

0x100 + i*4

B) Given an array with this below declaration, answer the following questions.

char *cs[5][10]; *// The starting address for cs is 0x200*

1) What is the total size of the array cs in bytes?

5*10*8 = 400

2) Give numeric (mathematical) expression to access the address of cs[i]. *(Remember, you are working with bytes)*

0x200 + (i*5*8)

3) Give numeric (mathematical) expression to access the address of cs[i][j]. *(Remember, you are working with bytes)*

0x200 + (i*5*8) + j*8

Struct and Alignment

Consider the following datatype definitions on an x86-64 machine.

```
typedef struct {
      char c;
      long *p;
      int i;
      int f;
      short s;
} struct1;
```

Alignment Rules:

1) All elements begin on an offset that is an even multiple of their type size. (Largest type for Structs)

2) Struct Size is a multiple of its largest type size.

A. Using the template below (allowing a maximum of 32 bytes), indicate the allocation of data for a structure of type **struct1.** Mark off and label the areas for each individual element using field names. Use dashes for the padding bytes that are allocated, but not used (to satisfy alignment). Assume the alignment rules discussed in lecture: data types of size must be aligned on -byte boundaries. **Leave the unused bytes blank**.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
c								p	p	p	p	p	p	p	p	i	i	i	i	f	f	f	f	s	s						

B. How many bytes are allocated for an object of type **struct1**?

32

C. What alignment is required for an object of type **struct1**?

(If an object must be aligned on an x-byte boundary, then your answer should be x.)

8

D. If we define the fields of **struct1** in a different order, we can reduce the number of bytes wasted by each variable of type **struct1**. How many bytes are **unused** in the best case? How many bytes will be **allocated** in this case?

Unused: 5 bytes
Allocated: 24 bytes

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
p	p	p	p	p	p	p	p	i	i	i	i	f	f	f	f	s	s	c													

(use this as scratch)

Struct and Assembly

In the following problem, you are given the task of reconstructing C code based on some declarations of C structures, and the x86-64 assembly code generated when compiling the C code.

Use the same size and alignment assumptions as question 1. Below are the data structure declarations. (Note that this is a single declaration which includes several data structures; they are shown horizontally rather than vertically simply so that they fit on one page.)

```
struct s1 {                   struct s2 {
      char a[3];                    struct s1 *d;
      struct s2 *b;                 char e;
      int c;                        int f[4];
      };                            struct s2 *g;
                              };
```

You may find it helpful to determine offsets for each struct type before considering the code below. For each x86-64 assembly code sequence below on the left, fill in the missing portion of corresponding C code on the right.

```
proc1:
      movl 16(%rdi), %eax
      ret
```

```
int proc1(struct s2 *x) {
            F[1]
   return x->____________
}
```

```
proc2:
      movq (%rdi), %rax
      movl 16(%rax), %eax
      ret
```

```
int proc2(struct s2 *x) {
            d -> c
   return x->_____________
}
```

```
proc3:
      movq 8(%rdi), %rax
      movq 32(%rax), %rax
      movl 12(%rax,%rsi,4), %eax}
ret
```

```
int proc3(struct s1 *x,long i) {
            b -> g -> f[i]
   return x->______________
}
```