

Assignment for Lab 9
Data Structure Representations
Computer Science 240

In lab this week, you will write some assembly language programs to study how data structures are stored in memory. To investigate this concept, it is useful to write some X86 assembly code directly (rather than producing it by compiling C code, as we have been doing up to now).

Assembly Directives

When you create X86 code directly, you will include *assembly directives*, which begin with a dot and indicate structural information useful to the assembler, linker, or debugger, but are not in and of themselves assembly instructions. For example, we use:

```
.globl main
```

to indicate that the label *main* is a global symbol that can be accessed by other code modules.

We state what part of memory to store code or data, and also declare and initialize all variables and strings, using the following directives:

```
.text .data, .quad, and .string
```

To see a list of possible directives, visit: <http://tigcc.ticalc.org/doc/gnuasm.html#SEC67>

We can also use variable names directly in X86 to reference memory locations.

On the next page is an example of a simple C program, and on the right is an X86 program that performs the equivalent task. Read carefully to correlate the C code to the X86.

simple.c: (C code)

```
#include <stdio.h>
```

```
long total = 0;
```

```
I  
int sum(int x,int y) {  
    int t = x + y;  
    total +=t;  
    return t;  
}
```

```
int main() {  
    int x = 2;  
    int y = 3;  
    printf("Sum = %d\n",sum(x,y));  
    printf("Total = %d\n",total);  
    return 0;  
}
```

simple.s: (X86 code)

```
.data //use the data segment of memory
```

```
total: .quad 0 //8 bytes with initial value 0  
fstr1: .string "Sum = %d\n"  
fstr2: .string "Total = %d\n"
```

```
.text //use the text segment of memory
```

(code)

```
.globl main  
sum:  
    lea (%rsi,%rdi,1),%eax  
    add %eax,total //variable to reference  
memory  
    ret
```

```
main:  
    mov $0x3,%esi  
    mov $0x2,%edi  
    call sum  
  
    mov %eax,%esi  
    mov $fstr1,%edi  
    mov $0x0,%eax  
    call printf  
  
    mov $total,%esi  
    mov $fstr2,%edi  
    mov $0x0,%eax  
    call printf  
  
    mov $0x0,%eax  
    ret
```

1. Using the previous program as a guide, write an X86 program which implements the following C program (do NOT use the computer to compile the C program and produce the X86 code: write it from scratch).

```
#include <stdio.h>
```

```
int z;
```

```
int square(int n) {  
    return n*n;  
}
```

```
int main() {  
    int x = square(3);  
    int y = square(4);  
    z = x + y;  
    printf("Calculation produces %d\n",z);  
}
```

```
return 0;
```

```
}
```