LAST Name:____

FIRST Name:____

Student ID:____

Score: /84

I do swear that all work done on this Midterm is my own. I swear that I did not copy any answers from anyone or attempt to cheat an any way. I swear this by the honor of my name and my true and trustworthy word .

Signature: ____

Name of Student On Left: ____

Name of Student On Right: ____

1. (2) If you saw a label in the data section you could assume that in the C code there was either a

   global or a **static** variable.

2. (4) If a function is **NOT** recursive could its local variables be turned into global variables? Assume that all local variables in all functions have unique names. **Justify** your answer.

   Yes it would be possible as there would only ever be a single instance of each variable.

3. (4) If a function **IS** recursive could its local variables be turned into global variables? Assume that all local variables in all functions have unique names. **Justify** your answer.
   No, it would not be possible as each function needs its own instance of the local variable.

4. (2) What is considered the stack?
   Wherever esp points and all addresses higher than it.

5. (2) What is considered the current stack frame?
   Everything between ESP and EBP

6. (2) What does the prologue establish for each function?
   That function's stack frame

7. (4) Explain what the call instruction does. Make sure to explain **what it does** and not what it is normally used to do.
   It pushes the address of the next instruction onto the stack and then jumps to the function.

8. (4) Explain what the ret instruction does. Make sure to explain **what it does** and not what it is normally used to do.
   It pops the element on top of the stack into EIP (The instruction pointer)

9. (4) According to the gcc C calling conventions, how are arguments passed to a function? In what order are they passed?
   They are passed on the stack in reverse order of appearance (right to left).

10. (8) According to the gcc C calling conventions, which registers will not have live values in them when a function call is made? How does this affect the code contained in the calling function? How does this affect the code in the function being called?

EAX, ECX, and EDX. The calling function must make sure to save the contents of those registers if they do contain live values before calling a function. The called function can overwrite those registers but if it wants to use any of the other ones it must make sure to save the values first and then restore before it returns.

11. (2) According to the gcc C calling conventions, how are values to be returned?
Through EAX.

12. (3 each) Given we have the following struct declaration in C

```
1. struct Employee{
2. unsigned short age;
3. char last_name[15];
4. unsigned int ID;
5. char first_name[20];
6. };
```

and a pointer to an instance of the Employee struct call emp, translate each of the following lines of C to assembly. For the following questions Assume that emp is stored in EAX, index is stored in ECX, and b is stored in EBX. Assume also that there is no padding or word aligning done on the struct. Only the **first** 2 lines of your answer will be considered for grading.

```
1. emp -> age = 25;
```
`movw $25, (%eax)`

```
2. emp - > ID = 47859;
```
`movl $47859, 17(%eax)`

```
3. emp - > last_name[5] = 65;
```
`movb $65, 7(%eax)`

```
4. emp - > first_name[3] = 66;
```
`movb $66, 24(%eax)`

```
5. emp - > last_name[i] = 66;
```
`movb $66, 2(%eax, %ecx)`

```
6. b = & (emp - > first_name[i]);
```
leal 21(%eax, %ecx), %ebx

13. (2 each) Fill in the blanks to complete the assembly code that emulates the following C code.

```
int fib(int n){
  int fib1;
  if(n == 0 || n == 1){
    return 1;
  } else{
    fib1 = fib(n-1);
    return fib1 + fib(n-2);
  }
```

```
.global fib

.equ wordsize, 4

.text
base_case:
  movl ________$1______________, %eax
  jmp epilogue

    fib:

      #prologue
      pushl %ebp
      movl %esp, %ebp
      subl $wordsize, %esp


      .equ n, 2*wordsize
      .equ fib1, -1*wordsize


      movl n(%ebp), %eax #put n into eax
      #if(n == 0 || n == 1)
      cmpl $0, %eax
      jz base_case
      cmpl $1, %eax
      jz base_case
      #recursive case
      #fib1 =  fib(n-1)
      push %eax
```

```
decl (%esp) #n - 1 on the stack
call fib
movl %eax, fib1(%ebp)
#fib(n-2)
decl (%ebp) #n - 2
call fib
addl $4, %esp #clear both args
addl fib1(%ebp), %eax
epilogue:
  movl %ebp, %esp
  pop %ebp
  ret
```