First Name (PRINT): ________________________________________________

Last Name (PRINT): ________________________________________________

University ID: ______________________________________________________

I pledge on my honor that I have not given or received any unauthorized assistance on this examination.

Your signature: ______________________________________________________

Instructions

- Write your name now (we will not wait for you to write your name at the end).
- You must stop writing once time is over.
- This exam is a closed-book and closed-notes exam.
- Total point value is 100 points.
- The exam is a 50 minutes exam.
- Please use a pencil to complete the exam.
- WRITE NEATLY.

Grader Use Only

<table>
<thead>
<tr>
<th>#1</th>
<th>Problem 1 (Process Control)</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>#2</td>
<td>Problem 2 (Assembly I)</td>
<td>15</td>
</tr>
<tr>
<td>#3</td>
<td>Problem 3 (Assembly II)</td>
<td>65</td>
</tr>
<tr>
<td>Total</td>
<td>Total (100)</td>
<td>100</td>
</tr>
</tbody>
</table>
**Problem 1 (Process Control) (20 pts)**

Define a main function that forks a child process and passes to it the string “mary”. The child process will take the string and return it to the parent after it converts it to uppercase using the `toUpperCase()` function provided below. The parent will print the string sends. You can assume system calls will not fail (no need to check for errors). You must use pipes for the exchange of data between the processes.

```c
void toUpper(char *data) {
    while (*data != '\0')
        *data++ -= 32;
}

Answer:

```c
int main() {  
    pid_t child_pid;  
    int par_pipe[2], chi_pipe[2], size = 5, status;  
    char message[4];  
    pipe(par_pipe);  
    pipe(chi_pipe);  
    child_pid = fork();  
    if (child_pid != 0) { /* parent */  
        close(par_pipe[0]); /* read end */  
        close(chi_pipe[1]); /* write end */  
        write(par_pipe[1], "mary", size);  
        read(chi_pipe[0], message, size);  
        printf("%s", message);  
        wait(&status);  
    } else { /* child */  
        close(par_pipe[1]);  
        read(par_pipe[0], message, size);  
        toUpper(message);  
        close(chi_pipe[0]);  
        write(chi_pipe[1], message, size);  
    }  
}
```

**Problem 2 (Assembly I) (15 pts)**

For all the questions below, you can assume the stack pointer has already being initialized.

1. (2 pts) Can we have a recursive function that does not require us to save/set the %ebp register?
   a. Yes, it is possible.
   b. No, we always need to save/set the %ebp.
   c. Only if the base case is handled by using %eax.
   d. Only if the result of the function is placed in %eax.
   e. c. and d. above.
   f. None of the above.

   **Answer:** a

---

2
2. (5 pts) Add at most **two** assembly code instructions to the code below, so we can have for **4** local variables in the function named **my_function**.

```assembly
my_function:                pushl %ebp
                      rrmovl %esp, %ebp
Answer:
    irmovl $16, %eax
    subl %eax, %esp
```

3. (4 pts) Without using the %ebp register, add a **single** assembly code instruction to the code below that will place the value of the first parameter into the register %eax.

```assembly
my_second_function:        pushl %ebp
                      rrmovl %esp, %ebp
                      irmovl $1, %ecx
                      pushl %ecx
                      pushl %ecx

Answer:
    mrmovl 16(%esp), %eax
```

4. (4 pts) Add at most **four** assembly code instructions before the “ret” instruction, so we can correctly return from the function assuming **30** pushl’s, **10** popl’s and **5** pushl’s instructions are executed before the **ret** instruction. You can assume we don’t care about the data associated with the pushl’s and popl’s.

```assembly
my_third_function:        pushl %ebp
                      rrmovl %esp, %ebp
                      irmovl $1, %ecx
                      # 30 pushl %ecx instructions executed
                      # 10 popl %ecx instructions executed
                      # 5 pushl %ecx instructions executed
                      # code you should provide goes here
                      ret

Answer:
    rrmovl %esp, %ebp
    popl %ebp
```


Problem 3 (Assembly II) (65 pts)

On the next page complete the assembly program that has the functionality associated with C program below. Even though we are using the string "HELLO", your program should work for any string we provide (do not write a program that only counts characters for the string “HELLO”). Your solution must be recursive, otherwise you will not receive credit. You can assume strings are null terminated. Your instances function must take two parameters and must not use local variables.

```c
#include <stdio.h>

char data[6] = "HELLO";

int instances(char *word, char target) {
    if (word[0] == '\0') {
        return 0;
    } else {
        if (word[0] == target) {
            return 1 + instances(word + 1, target);
        } else {
            return instances(word + 1, target);
        }
    }
}

int main() {
    char c;
    scanf("%c", &c);
    printf("%d", instances(data, c));
    return 0;
}
```

Answer:

main: irmovl $0x1000, %esp # init stack ptr

```assembly
    rdch %ecx              # reading character
    pushl %ecx             # pushing character
    irmovl data, %ebx      # pushing word
    pushl %ebx             # pushing word
    call instances
    wrint %eax
    halt
```

instances: pushl %ebp   # setting frame

```assembly
    rrmovl %esp, %ebp
    mrmovl 8(%ebp), %ebx # %ebx is the word
    mrmovl 12(%ebp), %ecx # %ecx is char
    irmovl $0, %eax      # base case
    mrmovl (%ebx), %edx
    subl %edx, %eax
    je instances_done
    subl %ecx, %edx      # recursive step (comparing character)
    je Equal
    irmovl $0, %esi
    jmp EndIf
    Equal:
    irmovl $1, %esi
    EndIf:
    pushl %esi            # saving value in stack
    pushl %ecx            # pushing character
    irmovl $4, %edx
    addl %edx, %ebx
```


pushl %ebx          # pushing word ptr
call instances
po1 %ebx           # cleaning stack (word ptr)
po1 %ebx           # cleaning stack (character)
po1 %esi           # restoring %esi
addl %esi, %eax    # final result in %eax

instances_done: rrmovl %ebp, %esp  # exiting
po1 %ebp
ret

.align 4
# HELLO string below
data:
.long 72
.long 69
.long 76
.long 76
.long 79
.long 0