University of Maryland College Park
Dept of Computer Science
CMSC216 Summer 2014
Midterm III

Last Name (PRINT): _____

First Name (PRINT): _____

University Directory ID (e.g., umcpturtle)_____

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

Your signature: _____

Instructions

- **Assembly and process cheat sheets can be found at the end of the exam.**
- This exam is a closed-book and closed-notes exam.
- Total point value is 200 points.
- The exam is a 50 minutes exam.
- Please use a pencil to complete the exam.
- WRITE NEATLY.
- You don't need to use meaningful variable names; however, we expect good indentation.

Grader Use Only

#1	Problem 1 (Miscellaneous)	(35)	
#2	Problem 2 (Process Control)	(55)	
#3	Problem 3 (Assembly Programming)	(110)	
Total	Total (200)	(200)	

Problem #1, Miscellaneous (35 pts)

For the following Assembly questions assume the stack frame organization discussed in class, and that the stack pointer has been initialized correctly.

1. (4 pts) Which of the following are INVALID instructions in Assembly? You can assume **data** is a label. Circle only the invalid ones.

```
a. rrmovl data, %ebp
b. irmovl %eax, 10
c. mrmovl data, %ecx
d. wrint 10
```

2. (4 pts) Complete the following irmovl statement so we can have **3** local variables.

```
irmovl       ,%eax
subl %eax ,%esp
```

3. (4 pts) Complete the following statement so we can copy the value of **%eax** to the second local variable (assuming we have two or more local variables).

```
rmmovl %eax,
```

4. (7 pts) Provide assembly code that implements the functionality associated with **popl %eax**.

5. (4 pts) Name two components of a parent process that are copied to the child when a fork() system call takes place.

6. (4 pts) A parent process has the following statements. Assuming that fork, execvp, and wait will not fail, will the parent process be able to reap the child process? Yes/No answer without explanation will receive no credit.

```
fork();
execvp(argv[0], argv);
wait(NULL);
```

7. (4 pts) Which process has process id one?

8. (4 pts) Which of the following are true for a **single** call of the wait() system call (executed by the parent process) after the parent process has created three processes via fork()? Circle the true ones.

   a. It blocks the parent process if no child has finished.
   b. It will reap first the first process the parent created, even if other child processes have finished.
   c. It will reap all the processes that have finished.
   d. None of the above.

Problem #2, Process Control (55 pts)

For this problem you will write a program where the parent process creates two child processes. For this problem:

- The parent will create a child process that executes the Unix command "cal 8 2014" using execlp. The "cal" command takes two command line arguments. The first one represents the month, and the second the year. This child will be printing the calendar for August 2014.
- If the child process terminated normally (use WIFEXITED to verify the status) the parent will create a second child that will execute the unix command "date". No second child will be created if the child process executing the "cal" command did not terminate normally.
- You can assume the fork(), execlp, and wait() system calls do not fail (you do not need to check for errors nor print any error messages).

PAGE FOR YOUR CODE

Problem #3, Assembly Programming (110 pts)

On the next page, complete the assembly code that corresponds to the **to_positive** function below. The function changes to positive, negative values in an array. The second parameter represents the number of elements to examine/check. For this problem:

- You only need to provide assembly code for the **to_positive** function.
- Do not provide assembly code for main(). We have provided the main function to illustrate how to use the function.
- On the next page you will see assembly representing the array.
- Although the example below relies on the provided array, your assembly code must work for any array of any length. You can assume we will take your code and change the data associated with array **d.**
- You can assume the **check** parameter will not exceed the length of the array.
- Provide comments with your assembly code. You will receive at least 10 pts for providing comments.
- **Your code must be efficient.**
- The **to_positive** function should push and pop the base pointer (%ebp).
- Parameters must be passed on the stack. You can assume the stack pointer has been set correctly.
- You may not define helper functions.
- You may not modify the provided function and implement the modified version.
- Do not use jmp to call a function.
- Notice that in the code provided on the next page, we are already establishing the registers you will use for the parameters. Use %edx (for the array) and %ecx (for the check parameter).
- **Use a register to represent the *changed* local variable (you do not need to use the stack to store the variable).**
- **Your solution must be recursive otherwise you will receive no credit (-110 pts).**

Example:

```
#include <stdio.h>

int to_positive(int *d, int check) {
   int changed = 0;

   if (check != 0) {
      if (*d < 0) {
         *d *= -1;
         changed = 1;
      }

      return changed + to_positive(++d, --check);
   }

   return 0;
}

int main() {
   int d[] = {4, 11, -8, 17, -87, 9}, check = 6, i;

   printf("Changed to positive: %d\n", to_positive(d, check));
   for(i = 0; i < check; i++) {
      printf("%d ", d[i]);
   }
   printf("\n");

   return 0;
}
```

```
to_positive:
   # PROLOGUE
   subu $sp, $sp, 8
   sw $ra, 8($sp)
   sw $fp, 4($sp)
   addu $fp, $sp, 8

   # BODY
   la $t0, 8($fp)    # $t0 = d
   lw $t1, 4($sp)    # $t1 = check

   beqz $t1, ret_0 # if check ==0, exec ret_0

   move $t2, 0($t0)
   bgez $t2, ret_func # if *d >= 0, exec ret_func
   mul $t2, $t2, -1
```

Example Output:

```
Changed to positive: 2
4 11 8 17 87 9
```

PAGE FOR YOUR CODE

```
to_positive:    pushl %ebp              # On entry saving old frame ptr
                rrmovl %esp, %ebp       # Setting new frame ptr

                mrmovl 8(%ebp),  %edx   # Retrieving array ptr (USE THIS REGISTER)
                mrmovl 12(%ebp), %ecx   # Retrieving number of elements (USE THIS REGISTER)
```

```
end:            rrmovl %ebp, %esp       # Reset stack ptr
                popl %ebp               # Restore callers frame ptr
                ret                     # Return
.align 4
d: .long 4
   .long 11
   .long -8
   .long 17
   .long -87
   .long 9
```

PAGE FOR YOUR CODE

PAGE FOR YOUR CODE

Cheat Sheets

Assembly Cheat Sheet

- Registers: %eax, %ecx, %edx, %ebx, %esi, %edi, %esp, %ebp
- Assembler Directives: .align, .long, .pos
- Data movement: irmovl, rrmovl, rmmovl, mrmovl
- Integer instructions: addl, subl, multl, divl, modl
- Branch instructions: jmp, jle, jl, je, jne, jge, jg
- Reading/Writing instructions: rdch, rdint, wrch, wrint
- Other: pushl, popl, call, ret, halt
- Ascii code for newline character: 0x0a
- Ascii code for space: 0x20

End of Assembly Cheat Sheet

Process Cheat Sheet

- pid_t fork(void);
- pid_t wait(int *status);
- pid_t waitpid(pid_t pid, int *status, int options);
- int execvp(const char *file, char *const argv[]);
- int execlp(const char *file, const char *arg, ...);

End of Process Cheat Sheet