

## CS 110 Practice Final Examination 2

---

### Notes:

This exam, like the 2<sup>nd</sup> practice midterm, is one of Tim Chou's old exams, which tend to be on the difficult side (also, Tim emphasized different things than I did). Then again, this exam may be a little short as I removed a few questions that I didn't think were appropriate.

The final exam will be cumulative, though the second half of the quarter will be emphasized (post-midterm material). Important things to review:

- Lecture notes
- Programs
- The Motorola manual:
  - 1-11: Status Register format
  - 2-2: Brief extension word format (for encoding indexed addressing modes)
  - 3-19: Conditional tests
  - Sec 4: User Mode instructions
  - Sec 6: Privileged instructions
  - B-2: Exception vector table.

### 1

You are given the following Pascal-style subroutine PSUB:

```
PSUB    MOVE.L    A6, -(A7)
        MOVE.L    A7, A6

        ; <<Subroutine body goes in here>>

        MOVE.L    A6, A7
        MOVE.L    (A7)+, A6
        MOVE.L    (A7)+, A1
        ADD.L     #8, A7
        JMP      (A1)

; Main Program
        MOVE.L    VAR(A5), -A7
        PEA      RVAR(A5)
        JSR      PSUB
```



3

The state of memory is as follows with (SSP) = 0000 2004.

Address	Contents
0000 2000	1278
0000 2002	0660
0000 2004	8800
0000 2006	1212
0000 2008	CCCC
0000 200A	0000

What is the address of the next instruction to execute and the corresponding stack pointer in the following cases?

	Instruction Addr.	Stack Pointer
RTD #2		
RTR		
RTS		
RTE		

4

The following procedure is part of the Mac's ToolBox:

```
procedure SetHandleSize (theHandle:Handle; newSize: Size);
```

All pointers (including handles) are 4 bytes long in the Mac. Size is a long integer representing the size of a heap block in bytes. Long integers are 4 bytes long. The toolbox routines conform to **Pascal-style** calling conventions.

Address	Contents
0000 2000	4E70
0000 2002	3456
0000 2004	22EA
0000 2006	0004
0000 2008	2000
0000 200A	4356
0000 200C	A352
0000 200E	1344
0000 2010	1554

The preceding table lists the contents of a portion of memory during the execution of a call to SetHandleSize. Assume the stack pointer was at \$2008 *just prior* to the instruction JSR SetHandleSize which invoked the procedure (in other words, assume traps aren't used). Fill in the following table with the values for the quantities assuming the above conditions.

theHandle	
newSize	
Old Frame Pointer	
Return Address	

5.

Here is some code:

```
Start    LEA        MyCode(PC), A0
         MOVE.W   (a0), $0
         MOVEA.L  #0, a0
         JMP     $0
MyCode   MOVE.W   (a0)+, (a0)
```

A) What is the preceding piece of code supposed to do (in general, what is its “goal in life”)?

B) Under what conditions does this code cease execution? Why?

C) If we were running on a 1 megabyte Mac Plus (with a 68000 cpu) and the above code had been running for a while (say 1000 lines of code) but then a level 5 interrupt occurred, what would be the address of the next instruction to be executed?

D) What assembly programming no-no does this code do?

## 6

Consider the following piece of code

```
MOVE.W    #$FF00, D0
MOVE.W    #$9, D1
CMP.W     D0, D1
LEA       $12(A3, D3.L), A1
JMP       (A1)
```

The program is running in user mode at interrupt level 0 and is loaded at address \$8800. An autovector 2 interrupt occurs while the program is executing the LEA instruction.

What is the address of the next instruction to execute?

PC =

What is the value of the SR when the next instruction executes?

SR =

## 7

Give brief answers to the following questions.

Is there any reason why you couldn't write a Mac (Palm) program that uses a stack (such as Quicksort), and uses A7 (SSP) as a stack pointer, but grows from low addresses to high addresses rather than the usual downward-growing stack? (i.e. the program would use post-increment to push things on the stack and pre-decrement to pop them off.) If it's possible, explain what assumptions you would have to make in order to guarantee error-free execution. If it's not possible, explain why not.

What are two major features that MacTasker (PalmTasker) lacks which a true multi-tasking OS should have?

Give an argument as to why recursive code is generally slower than iterative code.

What is a lock? How are locks used to enforce mutual exclusion in a multi-tasking environment? How can the TAS instruction be used to help ensure mutual exclusion during critical sections. Note (Spring '99): we just barely covered this in class—don't stress too much about it.

8. Compare the instruction sequence

```
LEA    $2000, A3
LEA    $1FF0, SP
```

with the instruction LINK A3, -#\$10 if SP = \$2000 when LINK is executed.

Is the LINK instruction equivalent?    Yes                    No

9. Assume the following

- A multitasking system supporting only 4 concurrent tasks.
- Each of the 4 task stack pointers are kept in the TASKTBL.
- Each task runs in **supervisor** state.
- Tasks are scheduled round robin.
- D0 contains the current task ID number (0,1,2 or 3)
- Don't worry about saving or restoring data or non-A7 address registers
- Tasks give up the processor by TRAP #0, which traps to the SWITCH trap handler below.

Fill in the appropriate instructions. Note (Spring '99): this question is a little odd because of the differences between MacTasker and PalmTasker. The main differences are the way functions are called (JSR vs TRAP) and the max number of tasks (4 vs. infinite).

```
TASKTBL DS.L          16

SWITCH
  _____
  MOVE          A7, TASKTBL(A5, D0)
  _____
  ADD           #1, D0
  CMP           #4, D0
  BLS          NEXT
  MOVE          #0, D0

NEXT
  _____
  _____
  _____
  _____
```