

1) [10 marks] Code analysis

Given the following data, what is the result of the operations on it? If an instruction is illegal or the result cannot be determined with the information given, clearly state the reason. All instructions are independent of each other.

d0 = \$98237654      d1 = \$A68B3237      d2 = \$000300FF  
d5 = \$0D358B45      d7 = \$FF00F2D0      a0 = \$00009802  
a7 = \$00001000

*[one mark per answer; condition codes all correct for one mark]*

a) and.w      d2,d0

*Result: d0 = \$0030054*

b) or.b      (a0),d1

*Result: d1 = can't tell (don't know what is in memory at location \$1000)*

c) moveq #\$A0,d5

*Result: d5 = \$000000A0*

d) addi.l d7,#1

*Result: d7 = illegal (destination can't be #1)*

e) swap d7

*Result: d7 = \$F2D0FF00*

f) ext.l d2

*Result: d2 = \$000000FF*

g) cmp.l #1,d5

*Result: d5 = \$0D358B45      , N = 0      , Z = 0      , V = 0      , C = 0*

h) pea.l 8(a0)

*Result: a7 = \$0000FFC      (a7) = \$0000980A*

- 2) [3 marks] What will be the contents of register d0 when the nop instruction is executed?  
[See pg 767 in the textbook for a description of dbra if you don't know what it is]

```
                move.l    #0,d0
                move.l    #2,d4
LOOP            addq.l    #3,d0
                dbra     d4,LOOP
                nop
```

*d0 = 9*

- 3) [5 marks] The LINK and UNLK instructions are useful for subroutine linkages, but the same operations can be performed using multiple other instructions. Show how link.l a6,#-4 can be done using multiple other instructions. Code efficiency will be part of your mark.

```
move.l a6,-(a7)
move.l a7,a6
subql #4,a7
```

4) [15 marks] Answer questions about the following code segment. Show the derivation of your results.

COUNT	equ	10				
TUTOR	equ	228				
PROGRAM	equ	\$1000				
	org	PROGRAM	<i>Address</i>	<i>cycles</i>	<i>cycles'</i>	<i>Address'</i>
	move.w	#0,d0	<b>\$1000</b>	<b>2</b>	<b>2</b>	<b>\$1000</b>
	move.w	#COUNT,d1	<b>\$1004</b>	<b>2</b>	<b>2</b>	<b>\$1004</b>
LOOP	subq	#1,d1	<b>\$1008</b>	<b>1*10</b>	<b>1*10</b>	<b>\$1008</b>
	addq	#1,d0	<b>\$100A</b>	<b>1*10</b>	<b>1*10</b>	<b>\$100A</b>
	tst	d1	<b>\$100C</b>	<b>1*10</b>	<b>1*10</b>	<b>\$100C</b>
	bne	LOOP	<b>\$100E</b>	<b>1*10</b>	<b>1*10</b>	<b>\$100E</b>
	move.w	d0,RESULT	<b>\$1010</b>	<b>4</b>	<b>3 [see (c)]</b>	<b>\$1010</b>
	move.w	#TUTOR,d7	<b>\$1016</b>	<b>2</b>	<b>2</b>	<b>\$1014</b>
	trap	#15	<b>\$101A</b>	<b>4</b>	<b>4</b>	<b>\$1018</b>
RESULT	ds.w	1	<b>\$101C</b>	<b>Total 54</b>	<b>Total 53</b>	<b>\$101A</b>
		end				

- a) [5] Specify addresses for each instruction & for RESULT in the boxes above.
- b) [2] What is the signed displacement (or offset) to the destination in the “bne LOOP” instruction? (express this as a decimal number) **-8**
- c) [6] How many memory cycles are needed for execution of the program? **54 (53 if you used absolute short addressing for RESULT; this also changes the addresses to those in the rightmost column above)**
- d) [2] What are the contents of memory location RESULT after execution of this program segment? **\$0A (or 10)**

5) [10 marks] Depending on an index sent by an outside device we want to be able to call one of ten subroutines in our 68000 program. For 0 sent we call Sub0, for 1 sent we call Sub1, ..., for 9 sent we call Sub9.

The subroutine addresses are in a table of longwords, SubAdrs. The index value is passed to our code in d0. Write the code that uses table SubAdrs to make the call to the correct subroutine using only one JSR instruction. State any other assumptions you need. Hint: Look up JSR in Appendix C, in particular the variety of addressing modes available. Code efficiency will be part of your mark.

FYI: This is part of the code that might result from the compilation of a CASE statement in C or C++.

```

movea.l    #SubAdrs,a0    ;a0 points to the table
asl.l     #2,d0           ;change index to a longword offset
movea.l    0(a0,d0),a0    ;get tabled value to a0
jsr       (a0)           ;go to routine [note that jsr uses the effective address]

```





- 7) [20 marks] Every rotation of a motor causes a switch to be pushed. The switch, like most mechanical switches, 'clatters' or bounces when the contacts come together, causing an interval when a single press causes multiple closures. This bouncing can continue for just under 3 milliseconds.

The switch is linked into the memory map of a 68000 processor at bit 0 of location \$FFFFFF00. It is connected so that the bit will read a '1' when the switch is open, and a '0' when closed and not bouncing. The bouncing will cause alternating zeros and ones.

**For ease of any analysis you do, assume that every instruction takes 1 usec., regardless of addressing mode.**

- a) [14] Write a *commented* program that continuously counts the number of times the switch is pushed. It must ignore the bouncing of the switch. This is all your program needs to do; don't worry about other functions that the 68000 may have to perform at this point. Code efficiency will be part of your mark.

*Assumptions: The switch is like a switch on a calculator, where there is no latched 'on' position. At one point in the turn of the motor the switch is pressed and bounces. Later on in the turn the switch is released. This repeats every revolution of the motor. There is no bounce when the switch is released.*

*Note that no marks were deducted for different reasonable interpretations of the mechanics of the problem.*

*SwitchPlace EQU \$FFFFFF00*

*SwitchBit EQU \$00*

*DelayCount EQU 1500 ;1500 \* 2 instr \* 1 usec = 3 msec.*

```
org $10000
Start moveq #0,d0 ;d0 is the counter
; The main counting loop starts here
Open btst.b #SwitchBit,SwitchPlace ;see if switch open
bne Open ;back if so
; When we get here, the switch has just been pressed
addq.l #1,d0 ;advance the count
; Delay over 3 msec to ignore bounce
movei #DelayCount,d1 ;loop this many times
Loop subq #1,d0 ;d0 will be zero by 3 msec.
bne Loop
; Now must wait for switch to be released
Closed btst.b #SwitchBit,SwitchPlace ;see if switch closed
bne Open ;back to start if open (released)
bra Closed ;keep looping here if not
```

- b) **[2]** What is the approximate limit to the rate at which the motor can turn and your program from (a) still correctly count the pushes of the switch? Show your analysis and calculations. ***300 Hz to 330 Hz was accepted. [1/(3 msec)]***
- c) If the switch were connected to a falling-edge-sensitive interrupt request line ...
- i) **[2]** ... what problem could result during the bouncing?  
***Multiple interrupts, one per bounce***
- ii) **[2]** ... what is one way the problem might be solved? Only the first answer will be marked.  
***Disable interrupts in ISR during bounce time***  
***Put debounce as in (a) in the ISR***  
***Use hardware debounce circuit***  
***Use flag, set flag while bouncing and don't count***