

2001 Midterm

This is the midterm that was used in 2001. Only assembly language was tested in this midterm; it may be different for you!!

1) (30 Points) Assume that memory is initialized as follows:

	org	\$20000	Addresses:
	dc.l	\$20004	\$20000
	dc.l	\$20008	\$20004
	dc.l	\$2000C	\$20008
BLAH	dc.l	\$20010	\$2000C
	dc.l	\$20014	\$20010
	dc.l	\$20018	\$20014

Show the values of d0 and a0 after each of the following code fragments executes:

a) (5 Points)

```
move.l    #$20000, d0
movea.l   $20000, a0
```

d0 = \$20000 a0 = \$20004

b) (5 Points)

```
movea.l   $20004, a0
move.l    4(a0), d0
```

d0 = \$20010 a0 = \$20008

c) (5 Points)

```
movea.l   BLAH, a0
move.l    (a0)+, d0
```

d0 = \$20014 a0 = \$20014

d) (5 Points)

```
movea.l   BLAH, a0
move.l    -(a0), d0
```

d0 = \$20010 a0 = \$2000C

e) (5 Points) What direction will the “beq ISZERO” follow when the following piece of code executes? Circle the correct answer. Note that dbf and dbra are the same instruction.

```
org $30000
loop    dbf    d1, loop
        cmp.l  #0, d1
        beq   ISZERO
FALLTHRU ...
        ...
ISZERO  ...
```

- i.** ISZERO
- ii.** FALLTHRU ← - Correct Answer

iii. Can't tell unless we know what value `d1` has before executing the code.

Note that when `dbf` exits, the lowest order 16 bits are set to equal `-1`. (See Appendix C!)

2) (30 Points) Consider the following subroutine:

```

FOOB00 link a6, #-4
        move.l d1, -(a7)

        move.l 12(a6), d0
        move.l d0, -4(a6)
        move.l 16(a6), d1
        add.l d1, d0
        move.l d0, 8(a6)

AAAA   move.l (a7)+, d1
BBBB   unlk a6
CCCC   rts

```

a) (15 Points) The stack of the subroutine just before the first subroutine instruction executes is shown bellow. Fill in the contents of the stack just before instruction AAAA executes (you can use register names to denote memory contents).

BEFORE EXECUTION: JUST
BEFORE INSTRUCTION AAAA

Address	Data
\$1FFFC	
\$20000	
\$20004	
\$20008	
\$2000C	a7 → \$300010
\$20010	\$abba
\$20014	\$2000
\$20018	\$1011
\$2001C	a6 → \$20040

Address	Data
\$1FFFC	
\$20000	a7 → d1
\$20004	\$2000
\$20008	a6 → \$2001C
\$2000C	\$300010
\$20010	\$3011
\$20014	\$2000
\$20018	\$1011
\$2001C	\$20040

Note that the *d1*

on the stack is the contents of the register before the

i
nstr
ucti
on
FO

OB
OO
exe
cut
es!

b) (5 Points) What is the value of a7 and a6 just **after** the instruction at address AAAA executes?

a6 = \$20008 a7 = \$20004

c) (5 Points) Same as (b). but **after** BBBB.

a6 = \$2001C a7 = \$2000C

d) (5 Points) Same as (c) but **after** CCCC.

a6 = \$2001C a7 = \$20010

- 3) (25 Points) The following routine saves a message at location MSG in memory. Execution of course starts at \$30000 and stops at the trap #15 instruction.

```

                                org    $20000

MASK      dc.b  %10001101, %10001100, %10001100,
           dc.b  %10000101, $85, $00001111
           dc.b  %10000101, $FF, %00000000
HIDDEN    dc.b  "ROWS", "ESPY", "ARMS", "PEAR", "BEAD"
MASK      ds.b  100

                                org    $30000
cryptic   movea.l  #MASK, a0
           movea.l  #HIDDEN, a1
           movea.l  #MSG, a3
NEXTW     move.b   (a0)+, d1
           move.b   d1, d2
           and      #$80, d2
           beq     DONE

           move.l   #4, d3
NEXTL     move.b   (a1)+, d2
           btst.b   #3, d1
           beq     SKIPL

SAVEL     move.b   d2, (a3)+

SKIPL     lsl.b    #1, d1
           subq.l   #1, d3
           bne     NEXTL
           bra     NEXTW

DONE      trap    #15
```

MESSAGE IS: ROSESARERED

Note: There was no need to brute force your way and trace through the entire code. All you had to do was realize that the constants at the location MASK were being used to choose the letters from each word that would make it to the output. The most significant bit determined whether to continue or stop. The *and #\$80, d2* instruction looks at that highest bit, and the program goes to DONE if it is zero. The lower four bits (from bit #3 down to bit #0) are checked - if the bit is one (ie, *btst.b* sets the Z-flag to false because the bit was NOT zero) then the SAVEL instruction will be executed, putting the corresponding letter into the MSG location. The *lsl.b* instruction only serves to move the next bit in the bitmask to

the location of bit #3 so that the *btst.b #3, d1* instruction can be used again.

- 4) (10 Points) The following code is meant to compare the corresponding bytes of two lists of bytes placing the **smaller (numerically)** byte in a third list. The two input lists are pointed to by a0 and a1 and the output list is pointed to by a2. The two input lists are of equal size and that size is given in d0. There are several problems with this code. Write a bug-free version of the code in the space provided. Underline all corrections.

ORIGINAL CODE
CORRECTED CODE

	org \$30000		org \$30000
loop	move.b (a0)+, d2 move.b d1, (a1)+ cmp.w d1, d2 beq FOO move.l d1, -(a2) bra loop	loop	move.b move.b cmp.b blt move.b bra
FOO	move.w d1, (a2)+	FOO	move.b
BOO	subq.l #1, d1 bne loop trap #15	BOO	subq.l bne trap

Note: Also accepted for full marks were *ble* instead of *blt*, or *bgt* or *bge* if you saved the appropriate register to memory. Marks were not deducted if you didn't save either byte when they were of equal value.

5) (10 Points) **EXTRA CREDIT: ATTEMPT THIS ONLY IF YOU HAVE SOME TIME LEFT AT THE END.**

Consider a hypothetical processor called SIC (Single Instruction Computer). As its name implies, this processor has only one instruction: subtract and branch if negative, or SBN for short. The SBN instruction has three operands, each being the address of a long word in memory:

```
sbn  a, b, c
```

Its function is:

```
MEM[a] = MEM[a] - MEM[b]
if  MEM[a] < 0 then PC = C
otherwise PC points to the next in sequence
instruction
```

- a) (5 Points) If every address requires 4 bytes, how many bytes do we need to completely encode the SBN instruction?

Since there is only 1 operation, there is no need for an opcode. Each operation requires 3 operands (addresses), each taking 4 bytes. Therefore the instruction requires $3 \times 4 = 12$ bytes to encode.

- b) (5 Points) The following code moves the value from memory location **a** to memory location **b**.

```

                                org  $20000
a                                dc.l  $20
b                                dc.l  $20
temp                             dc.l  $00

                                org  $30000
                                code is doing:
ZERO                             SBN  temp, temp, ONE           temp = 0
ONE                              SBN  temp, a, TWO             temp = -a
TWO                              SBN  b, b, THREE              b = 0
THREE                            SBN  b, temp, FOUR             b = -temp
= a
FOUR
```

Write a program that adds the value from **b** to the value of **a** and leaves the result in **a**. The value in memory location **b** should be left unchanged.

```

                                org  $30000
ZERO                             SBN  temp, temp, ONE           ; temp = 0
ONE                              SBN  temp, b, TWO              ; temp = -b
TWO                              SBN  a, temp, THREE             ; a = a -
temp = a + b
THREE
```