This is a closed book exam. You must put your answers on this piece of paper only. You have 50 minutes, so allocate your time accordingly. *Please read the entire quiz before starting.*

**Question 1.** Answer A, B, C, D, E

**Question 2.** Answer $00$ to $FF$

**Part 3a.** Specify RegB

**Part 3b.** Specify 0 or 1

**Part 3c.** Specify 0 or 1

**Part 3d.** Specify 0 or 1

**Part 3e.** Specify 0 or 1

**Question 4.** Specify Δ

**Question 5.** Show the machine code

**Question 6.** What value is pushed

**Question 7.** Simplified memory cycles (you may or may not need all 5 entries)

<table>
<thead>
<tr>
<th>R/W</th>
<th>Addr</th>
<th>Data</th>
<th>changes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Jonathan W. Valvano   September 26, 2003   11:00am-11:50am
(40) **Question 8.** Write the assembly language program that implements a thermostat.

```
PORTA equ $0000
PORTB equ $0001
DDRA equ $0002
DDRB equ $0003
```
These two tables interpret indexed-mode machine codes

<table>
<thead>
<tr>
<th>rr</th>
<th>register</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>X</td>
</tr>
<tr>
<td>01</td>
<td>Y</td>
</tr>
<tr>
<td>10</td>
<td>SP</td>
</tr>
<tr>
<td>11</td>
<td>PC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>postbyte, xb</th>
<th>syntax</th>
<th>mode</th>
<th>explanations</th>
</tr>
</thead>
<tbody>
<tr>
<td>rr000000</td>
<td>,r</td>
<td>IDX</td>
<td>5-bit constant, n=0</td>
</tr>
<tr>
<td>rr00nnnn</td>
<td>n,r</td>
<td>IDX</td>
<td>5-bit constant, n=0 to +15</td>
</tr>
<tr>
<td>rr01nnnn</td>
<td>-n,r</td>
<td>IDX</td>
<td>5-bit constant, n=-16 to -1</td>
</tr>
<tr>
<td>rr100nnn</td>
<td>n,+r</td>
<td>IDX</td>
<td>pre-increment, n=1 to 8</td>
</tr>
<tr>
<td>rr101nnn</td>
<td>n,-r</td>
<td>IDX</td>
<td>pre-decrement, n=1 to 8</td>
</tr>
<tr>
<td>rr110nnn</td>
<td>n,r+</td>
<td>IDX</td>
<td>post-increment, n=1 to 8</td>
</tr>
<tr>
<td>rr111nnn</td>
<td>n,r-</td>
<td>IDX</td>
<td>post-decrement, n=1 to 8</td>
</tr>
<tr>
<td>111rr100</td>
<td>A,r</td>
<td>IDX</td>
<td>Reg A accumulator offset</td>
</tr>
<tr>
<td>111rr101</td>
<td>B,r</td>
<td>IDX</td>
<td>Reg B accumulator offset</td>
</tr>
<tr>
<td>111rr110</td>
<td>D,r</td>
<td>IDX</td>
<td>Reg D accumulator offset</td>
</tr>
<tr>
<td>111rr000 ff</td>
<td>n,r</td>
<td>IDX1</td>
<td>9-bit cons, n 16 to 255</td>
</tr>
<tr>
<td>111rr001 ff</td>
<td>n,-r</td>
<td>IDX1</td>
<td>9-bit const, n -256 to -16</td>
</tr>
<tr>
<td>111rr010 eff</td>
<td>n,r</td>
<td>IDX2</td>
<td>16-bit const, any 16-bit n</td>
</tr>
<tr>
<td>111rr111</td>
<td>[D,r]</td>
<td>[D,IDX]</td>
<td>Reg D offset, indirect</td>
</tr>
<tr>
<td>111rr011 eff</td>
<td>[n,r]</td>
<td>[IDX2]</td>
<td>16-bit constant, indirect</td>
</tr>
</tbody>
</table>

**ORAB**

Operation: \((B) + (M) = B\)

Description: Performs bitwise logical inclusive OR between the content of accumulator B and the content of memory location M. The result is placed in B. Each bit of B after the operation is the logical inclusive OR of the corresponding bits of M and of B before the operation.

Condition Codes and Boolean Formulas:

\[
\begin{array}{cccccccc}
S & X & H & I & N & Z & V & C \\
- & - & - & - & A & A & 0 & -
\end{array}
\]

- **N**: Set if MSB of result is set; cleared otherwise.
- **Z**: Set if result is $0000\ldots$; cleared otherwise.
- **V**: 0; Cleared.

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
<th>Cycles</th>
<th>Access Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORAB #oprB</td>
<td>IMM</td>
<td>CA ii</td>
<td>1</td>
<td>P</td>
</tr>
<tr>
<td>ORAB opr3a</td>
<td>DIR</td>
<td>DA dd</td>
<td>3</td>
<td>rFP</td>
</tr>
<tr>
<td>ORAB opr16a</td>
<td>EXT</td>
<td>PA hh ll</td>
<td>3</td>
<td>rOP</td>
</tr>
<tr>
<td>ORAB oprx0,xy</td>
<td>IDX</td>
<td>EA xb</td>
<td>3</td>
<td>rFP</td>
</tr>
<tr>
<td>ORAB oprx16,xy</td>
<td>IDX1</td>
<td>EA xb ff</td>
<td>3</td>
<td>rPO</td>
</tr>
<tr>
<td>ORAB oprx16,xy</td>
<td>IDX2</td>
<td>EA xb ff</td>
<td>4</td>
<td>fIPrFP</td>
</tr>
<tr>
<td>ORAB [D,xy]</td>
<td>[D,IDX]</td>
<td>EA xb</td>
<td>6</td>
<td>fIPrFP</td>
</tr>
<tr>
<td>ORAB [opr16,xy]</td>
<td>[IDX2]</td>
<td>EA xb ff</td>
<td>6</td>
<td>fIPrFP</td>
</tr>
</tbody>
</table>
beq branch if result is zero (Z=1)
addd 16-bit add to RegD
addb 8-bit add to RegB
addc 8-bit add to RegD
addcl 16-bit add to RegD
anda 8-bit logical and to RegA
andb 8-bit logical and to RegB
andcc 8-bit logical and to RegCC
bge branch if signed =
bcs branch if carry set
bclr clear bits in memory
asrb 8-bit arith right shift to RegB
asra 8-bit arith right shift
blick branch if signed =
bhi branch if unsigned =
bgnd enter background debug mode
bgt branch if unsigned >
bcc branch if carry clear
asr 8-bit arith right shift Memory
asld/lsld 16-bit left shift RegD
asla/lsla 8-bit left shift RegA
asl/lsl 8-bit left shift Memory
coma 8-bit logical complement to RegA
com 8-bit logical complement to Memory
cmpb 8-bit compare RegB with memory
cmpa 8-bit compare RegA with memory
bitb 8-bit and with RegA, sets CCR
bita 8-bit and with RegB, sets CCR
bhs branch if unsigned =
bsr branch to subroutine
brset branch if result is zero (Z=0)
blt branch if signed <
ble branch if unsigned =
bls branch if unsigned =
bllo branch if unsigned <
blr branch if signed <
bmi branch if result is negative (N=1)
bne branch if result is nonzero (Z=0)
bpl branch if result is positive (N=0)
bra branch always
brclr branch if bits are clear,
brn branch never
brset branch if bits are set
bset branch if result is zero (Z=1)
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bitb 8-bit and with RegA, sets CCR
bita 8-bit and with RegB, sets CCR
bhs branch if unsigned =
bsr branch to subroutine
brset branch if result is zero (Z=0)
blt branch if signed <
ble branch if unsigned =
bls branch if unsigned =
bllo branch if unsigned <
blr branch if signed <
bmi branch if result is negative (N=1)
bne branch if result is nonzero (Z=0)
bpl branch if result is positive (N=0)
bra branch always
brclr branch if bits are clear,
brn branch never
brset branch if bits are set
bset branch if result is zero (Z=1)
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bitb 8-bit and with RegA, sets CCR
bita 8-bit and with RegB, sets CCR
bhs branch if unsigned =
bsr branch to subroutine
brset branch if result is zero (Z=0)
blt branch if signed <
ble branch if unsigned =
bls branch if unsigned =
bllo branch if unsigned <
blr branch if signed <
bmi branch if result is negative (N=1)
bne branch if result is nonzero (Z=0)
bpl branch if result is positive (N=0)
bra branch always
brclr branch if bits are clear,
brn branch never
brset branch if bits are set
bset branch if result is zero (Z=1)
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bitb 8-bit and with RegA, sets CCR
bita 8-bit and with RegB, sets CCR
bhs branch if unsigned =
bsr branch to subroutine
brset branch if result is zero (Z=0)
blt branch if signed <
ble branch if unsigned =
bls branch if unsigned =
bllo branch if unsigned <
blr branch if signed <
bmi branch if result is negative (N=1)
bne branch if result is nonzero (Z=0)
bpl branch if result is positive (N=0)
bra branch always
brclr branch if bits are clear,
brn branch never
brset branch if bits are set
bset branch if result is zero (Z=1)
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bitb 8-bit and with RegA, sets CCR
bita 8-bit and with RegB, sets CCR
bhs branch if unsigned =
bsr branch to subroutine
brset branch if result is zero (Z=0)
blt branch if signed <
ble branch if unsigned =
bls branch if unsigned =
bllo branch if unsigned <
blr branch if signed <
bmi branch if result is negative (N=1)
bne branch if result is nonzero (Z=0)
bpl branch if result is positive (N=0)
bra branch always
brclr branch if bits are clear,
brn branch never
brset branch if bits are set
bset branch if result is zero (Z=1)
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bitb 8-bit and with RegA, sets CCR
bita 8-bit and with RegB, sets CCR
bhs branch if unsigned =
bsr branch to subroutine
brset branch if result is zero (Z=0)
blt branch if signed <
ble branch if unsigned =
bls branch if unsigned =
bllo branch if unsigned <
blr branch if signed <
bmi branch if result is negative (N=1)
bne branch if result is nonzero (Z=0)
bpl branch if result is positive (N=0)
bra branch always
brclr branch if bits are clear,
brn branch never
brset branch if bits are set
bset branch if result is zero (Z=1)
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bitb 8-bit and with RegA, sets CCR
bita 8-bit and with RegB, sets CCR
bhs branch if unsigned =
bsr branch to subroutine
brset branch if result is zero (Z=0)
blt branch if signed <
ble branch if unsigned =
bls branch if unsigned =
bllo branch if unsigned <
blr branch if signed <
bmi branch if result is negative (N=1)
bne branch if result is nonzero (Z=0)
bpl branch if result is positive (N=0)
bra branch always
brclr branch if bits are clear,
brn branch never
brset branch if bits are set
bset branch if result is zero (Z=1)
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bitb 8-bit and with RegA, sets CCR
bita 8-bit and with RegB, sets CCR
bhs branch if unsigned =
bsr branch to subroutine
brset branch if result is zero (Z=0)
blt branch if signed <
ble branch if unsigned =
bls branch if unsigned =
bllo branch if unsigned <
blr branch if signed <
bmi branch if result is negative (N=1)
bne branch if result is nonzero (Z=0)
bpl branch if result is positive (N=0)
bra branch always
brclr branch if bits are clear,
brn branch never
brset branch if bits are set
bset branch if result is zero (Z=1)
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bitb 8-bit and with RegA, sets CCR
bita 8-bit and with RegB, sets CCR
bhs branch if unsigned =
bsr branch to subroutine
brset branch if result is zero (Z=0)
blt branch if signed <
ble branch if unsigned =
bls branch if unsigned =
bllo branch if unsigned <
blr branch if signed <
bmi branch if result is negative (N=1)
bne branch if result is nonzero (Z=0)
bpl branch if result is positive (N=0)
bra branch always
brclr branch if bits are clear,
brn branch never
brset branch if bits are set
bset branch if result is zero (Z=1)
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bitb 8-bit and with RegA, sets CCR
bita 8-bit and with RegB, sets CCR
bhs branch if unsigned =
bsr branch to subroutine
brset branch if result is zero (Z=0)
blt branch if signed <
ble branch if unsigned =
bls branch if unsigned =
bllo branch if unsigned <
blr branch if signed <
bmi branch if result is negative (N=1)
bne branch if result is nonzero (Z=0)
bpl branch if result is positive (N=0)
bra branch always
brclr branch if bits are clear,
brn branch never
brset branch if bits are set
bset branch if result is zero (Z=1)
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bge branch if signed =
bgte branch if signed >
bhi branch if unsigned =
bitb 8-bit and with RegA, sets CCR
bita 8-bit and with RegB, sets CCR
bhs branch if unsigned =
bsr branch to subroutine
brset branch if result is zero (Z=0)
blt branch if signed <
ble branch if unsigned =
bls branch if unsigned =
bllo branch if unsigned <
blr branch if signed <
bmi branch if result is negative (N=1)
bne branch if result is nonzero (Z=0)
bpl branch if result is positive (N=0)
bra branch always
brclr branch if bits are clear,
Motorola 6812 addressing modes

(5) Question 1. Which term best describes a memory that retains its information when power is removed and restored?

A) nonvolatile  B) volatile  C) memory mapped  D) friendly  E) none of these

(5) Question 2. What is the 8-bit unsigned hexadecimal representation of the number 171?

Question 3. Consider the result of executing the following two 6812 assembly instructions.

```
ldab #101
subb #110
```

(4) Part a) What is the value in Register B after these two instructions are executed? Give your answer in unsigned decimal (0 to 255).

(2) Part b) What will be the value of the carry (C) bit?

(2) Part c) What will be the value of the overflow (V) bit?

(2) Part d) What will be the value of the zero (Z) bit?

(2) Part e) What will be the value of the negative (N) bit?
(10) **Question 4.** The range of values spans 0 to 0.255 (2.55e-1), and you will use an 8-bit unsigned decimal fixed-point format. What resolution $\Delta$ would be best?

(10) **Question 5.** Show the machine code generated by the instruction
\[
\text{orab } -20, y
\]

(10) **Question 6.** When the `bsr sub` instruction is executed, what value is pushed on the stack?

```
SF120 CF0C00 [2](0){OP}main lds #$C000
SF123 87 [1](2){O }cira
SF124 0766 [4](3){PPPS}loop bsr sub
SF126 20FC [3](7){PPP}bra loop

; Purpose: increment a number
; Input: RegA, range 0 to 255
; Output: RegA=Input+1
; Errors: overflow if 255
```

(10) **Question 7.** Give the simplified memory cycles produced when the following one instruction is executed. Assume the PC contains $F129$, Reg Y equals $0980$, Reg B is 0, and each memory location from $0900$ to $09FF$ contains a value equal to the least significant byte of its address. I.e., $0900$ contains $00$, $0901$ contains $01$, $0902$ contains $02$ etc. Show changes during the cycle they occur to the memory, PC, Y, B, IR, and EAR. Ignore changes to the CCR.

```
F129 EA4A [3](3){rfP} orab 10,y
```

(40) **Question 8.** Write the assembly language program that implements a thermostat. PORTB is an 8-bit output that controls the heater to the room. If you set PORTB to $00$, the heater will go off. If you set PORTB to $FF$, then the heater will turn on. PORTA is an 8-bit input containing the current temperature in unsigned binary fixed-point format with a resolution of $2^{-1}$ºF. The range is 0ºF to 127.5ºF. For example, if the room temperature is 70ºF, then reading PORTA will return 140 (decimal). If the temperature goes below 68ºF, turn the heater on. If it goes above 72ºF, turn the heater off.

![Thermostat Diagram](image)
Table A-3 Indexed Addressing Mode Postbyte Encoding (xb)

| A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | C0 | C1 | C2 | C3 | C4 | C5 | C6 | C7 | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | E0 | E1 | E2 | E3 | E4 | E5 | E6 | E7 | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0  | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1  | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2  | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3  | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4  | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5  | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6  | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7  | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8  | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9  | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | X | 16 | & | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Jonathan W. Valvano     September 26, 2003       11:00am-11:50am