“There is no reason anyone would want a computer in their home.”
Ken Olson, president, chairman and founder of Digital Equipment Corporation, 1977

Exam 1 review
closed book, no calculator
Lectures 1-10 (no TCNT, indexed, arrays, pointers)
HW1-3 (some C programming)
Labs 1, 2, and 3
Problems on old tests/HW you are not responsible for

1) Definitions (matching or multiple choice)
volatile, nonvolatile, RAM, ROM, port
static efficiency, dynamic efficiency
structured program, call graph, data flow graph
basis, nibble, precision, decimal digits (see table below)
fixed point, overflow, ceiling and floor, drop out,
bus, address bus, data bus,
memory-mapped, I/O mapped
bus cycle, read cycle, write cycle,
IR, EAR, BIU, CU, ALU, registers,
device driver, reset vector
friendly, mask, toggle,

| $2^2 = 4$ | $2^8 = 256$ | $2^{14} = 16384$ |
| $2^3 = 8$ | $2^9 = 512$  | $2^{15} = 32768$ |
| $2^4 = 16$ | $2^{10} = 1024 \approx 10^3$ | $2^{16} = 65536$ |
| $2^5 = 32$ | $2^{11} = 2048$ | $16^3 = 256$ |
| $2^6 = 64$ | $2^{12} = 4096$ | $16^4 = 4096$ |
| $2^7 = 128$ | $2^{13} = 8192$ | $16^5 = 65536$ |

<table>
<thead>
<tr>
<th>decimal digits</th>
<th>exact range</th>
<th>exact alternatives</th>
<th>ADC bits needed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0 to 999</td>
<td>1,000</td>
<td>10</td>
</tr>
<tr>
<td>$3 \frac{1}{2}$</td>
<td>0 to 1999</td>
<td>2,000</td>
<td>11</td>
</tr>
<tr>
<td>$3 \frac{3}{4}$</td>
<td>0 to 3999</td>
<td>4,000</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>0 to 9999</td>
<td>10,000</td>
<td>14</td>
</tr>
<tr>
<td>$4 \frac{1}{2}$</td>
<td>0 to 19,999</td>
<td>20,000</td>
<td>15</td>
</tr>
<tr>
<td>$4 \frac{3}{4}$</td>
<td>0 to 39,999</td>
<td>40,000</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>0 to 99,999</td>
<td>100,000</td>
<td>17</td>
</tr>
<tr>
<td>$5 \frac{1}{2}$</td>
<td>0 to 199,999</td>
<td>200,000</td>
<td>18</td>
</tr>
<tr>
<td>$5 \frac{3}{4}$</td>
<td>0 to 399,999</td>
<td>400,000</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>0 to 999,999</td>
<td>1,000,000</td>
<td>20</td>
</tr>
<tr>
<td>$6 \frac{1}{2}$</td>
<td>0 to 199999</td>
<td>2,000,000</td>
<td>21</td>
</tr>
<tr>
<td>$6 \frac{3}{4}$</td>
<td>0 to 3,999,999</td>
<td>4,000,000</td>
<td>22</td>
</tr>
<tr>
<td>N</td>
<td>0 to $10^{N-1}$</td>
<td>$10^N$</td>
<td></td>
</tr>
<tr>
<td>$N \frac{1}{2}$</td>
<td>0 to $2\times10^{N-1}$</td>
<td>$2\times10^N$</td>
<td></td>
</tr>
<tr>
<td>$N \frac{3}{4}$</td>
<td>0 to $4\times10^{N-1}$</td>
<td>$4\times10^N$</td>
<td></td>
</tr>
</tbody>
</table>

Standard definition of decimal digits.

2) Number conversions, 8-bit (fill in the blank)
convert one format to another without a calculator
signed decimal e.g., –56
unsigned decimal e.g., 200
binary e.g., $\text{11001000}$
hexadecimal e.g., $\text{C8}$

Jonathan W. Valvano
I won’t ask you to convert signed binary or signed hex:

signed binary e.g., \(-00101111\)
signed hexadecimal e.g., \(-$2F\)

**fixed-point representations**
- given resolution convert between value and integer
- given precision and range choose the fixed-point format

3) Details of executing single instructions
- 8-bit addition, subtraction yielding result, N, Z, V, C (like HW)
- simplified cycle by cycle execution
- assembly listing to execution cycles (aLec04)

for indexed mode addresses, for example
- \(\text{lda } 4, x\)
- \(\text{lda } 40, x\)
- \(\text{lda } -4, x\)
- \(\text{lda } -40, x\)
- \(\text{lda } \$400, x\)
- \(\text{lda } 4, x\)
- \(\text{lda } 4, -x\)
- \(\text{lda } 4, x+\)
- \(\text{lda } 4, x-\)
- calculate effective address
- go from assembly to machine code \(\text{xb}\)
- go from machine code \(\text{xb}\) to assembly
- simple multiply and divide \((\text{mul, idiv, fdiv})\)
- stack functions for \(\text{bsr}\) and \(\text{rts}\)

4) Simple programs (either C or assembly)
- initialize stack (this automatically happens in C)
- create global variables
- set reset vector (this automatically happens in C)
- specify an I/O pin is an input
- specify an I/O pin is an output
- clear an I/O output pin to zero
- set an I/O output pin to one
- toggle an I/O output pin
- check if an I/O input pin is high or low
  e.g., \(\text{if PT4 is low then make PM2 high}\)

****study question******

8-bit operations
- add, sub, shift left, shift right, and, or, eor
- \(\text{if-then}\) like examples in Chapter 5
- \(\text{if-then-else}\)
- \(\text{if}((uG1>5)&&(uG2<100))\)
- \(\text{while-loop}\) like examples in Chapter 5
- \(\text{for-loop}\) like those in this lecture
- simple subroutines, parameters passed in registers
  four lines of comments for client
  * purpose
  * inputs: registers, format, units
  * outputs: registers, format, units
  * error possibilities
  called with \(\text{bsr}\), returns using \(\text{rts}\)
5) Switch and LED interfaces (Labs 2, 3, and the book)

6) C programming

How to create a C program, and functions without parameters

```c
void TogglePTT(void) {
    PTT = PTT^0x01;
}

void main(void) {
    DDRT = DDRT|0x01;
    while(1) {
        TogglePTT();
    }
}
```

How to define global variables (with and without `unsigned`)

```c
char Data;    // 8-bit variable  
short D1, D2; // 16-bit variables 
long L3;      // 32-bit variable  
```

At this point we are not distinguishing between local and global variables, but soon we will make a big deal out of whether the variable is global or local. So far we have only taught you how to make global variables in assembly.

An integer variable has size that depends on the machine
(with and without `unsigned`)

```c
int Z1;       // variable  
```

How to read from and write to C variables

```c
D1 = 100;  
D2 = D1 + 100; 
```

Simple calculations

- Arithmetic operations: `+`, `-`, `*`, `/`, `%`, `++`, `--`
- Logical operations: `|`, `&`, `^`, `~`
- Shift operations: `>>`, `<<`

Conditional structures

- Compare operators: `==`, `!=`, `<`, `<=`, `>`, `>=`
- Boolean operators: `&&`, `||`

If-then

```c
if(D2 < D1){
    D2isLess();
}
```

If-then-else

```c
if(D2 < D1){
    D2isLess();
} else{
    D1isLessOrEqual();
}
```

Looping structures

- `while`-loop (test before each execution of the body)

```c
while(D2 < 100){
    OverAndOver(); // repeat while D2<100
}
```
do-while-loop (test after each execution of the body)
\[
\text{do}
\]
\[
\text{OverAndOver(); // repeat while D2 < 100}
\]
\[
\text{while(}D2 < 100)\text{;}
\]

for-loop (test before each execution of the body)
\[
\text{for}(D2=0; D2 < 100; D2=D2+1)\{
\]
\[
\text{OverAndOver(); // repeat 100 times}
\]
\[
\}
\]

Look at previous exams to see the types of information given to you. Notice also the format of the exam and the expected answers. You will get information a list of instructions and addressing modes You will also get the CPU12 page(s) for any instruction(s) for which you need to find bus cycles.

It is important to know
- precision (e.g., 8-bit, 16-bit)
- format (e.g., unsigned, signed)
- unsigned, \textbf{bhi bhs and bls}
- signed, \textbf{bgt bie bge and ble}
- either signed or unsigned, \textbf{beq and bne}

It takes three steps
1. read the first value into a register
2. compare the first value with the second value
3. conditional branch

\textbf{Compare the four possible inequalities}
Assume \textbf{PTT} is a unsigned 8-bit input port, and let \textbf{Threshold} be an unsigned 8-bit global variable

<table>
<thead>
<tr>
<th>C code</th>
<th>assembly code</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{if}(PTT &gt; Threshold){</td>
<td>\texttt{ldab PTT} \texttt{cmpb Threshold} \texttt{bhs next next}</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>\textit{if}(PTT &gt;= Threshold){</td>
<td>\texttt{ldab PTT} \texttt{cmpb Threshold} \texttt{blo next next}</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>\textit{if}(PTT &lt; Threshold){</td>
<td>\texttt{ldab PTT} \texttt{cmpb Threshold} \texttt{bhs next next}</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>\textit{if}(PTT &lt;= Threshold){</td>
<td>\texttt{ldab PTT} \texttt{cmpb Threshold} \texttt{bhi next next}</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>
Compare signed versus unsigned conditionals
Assume uG is an unsigned 8-bit global variable
Assume sG is a signed 8-bit global variable

<table>
<thead>
<tr>
<th>C code</th>
<th>assembly code</th>
</tr>
</thead>
<tbody>
<tr>
<td>if(uG &gt;= 5){</td>
<td>ldauG</td>
</tr>
<tr>
<td></td>
<td>cmpa #5</td>
</tr>
<tr>
<td></td>
<td>blo next</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td>if(sG &gt;= 5){</td>
<td>ldasG</td>
</tr>
<tr>
<td></td>
<td>cmpa #5</td>
</tr>
<tr>
<td></td>
<td>blt next</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

Compare 8-bit versus 16-bit conditionals
Assume uG1 and uG2 are unsigned 8-bit variables
Assume uH1 and uH2 are unsigned 16-bit variables

<table>
<thead>
<tr>
<th>C code</th>
<th>assembly code</th>
</tr>
</thead>
<tbody>
<tr>
<td>if(uG2 &gt;= uG1){</td>
<td>ldauG2</td>
</tr>
<tr>
<td></td>
<td>cmpauG1</td>
</tr>
<tr>
<td></td>
<td>blo next</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td>if(uH2 &gt;= uH1){</td>
<td>lduH2</td>
</tr>
<tr>
<td></td>
<td>cpduH1</td>
</tr>
<tr>
<td></td>
<td>blo next</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

for(uG=0;uG<5;uG++){  
  PTT = uG;  
}  
clr uG  
loop ldaa uG  
cmpa #5  
bhs next ; stop when uG>=5  
ldauG  
staa PTT  
inc uG  
bra loop  
next  

PTT = PTT & ~0x02;  
for(i=5;i>0;i--){ // something 5 times  
  PTT = PTT^2;  
}  
bclr PTT,$$02 ;PT1=0  
ldauG #5 ; loop 5 down to 0  
loop ldab PTT ; body of for loop  
etrb $$02 ;toggle PT2  
stab PTT  
dbne A,loop
do{
    ♪♫♪♫♫
}
while(uG < 5);

ldaa uG
cmpa #5
blo  loop  ; stop when uG>=5

Problem: write code that waits for a switch to be pressed. Assume PP3 is an input with a switch attached.
1) how are we going to test it?
2) flow chart
3) pseudocode
4) assembly
4) testing

The bottom line
Study previous exam1s
no indexed mode, no arrays,
no fixed-point, no pointers
Study homework 1,2,3,
Review Labs 1, 2, and 3
Review lecture notes 1-10