"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,

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

decimal digits	exact range	exact alternatives	ADC bits needed?
3	0 to 999	1,000	10
3½	0 to 1999	2,000	11
3¾	0 to 3999	4,000	12
4	0 to 9999	10,000	14
4½	0 to 19,999	20,000	15
4¾	0 to 39,999	40,000	16
5	0 to 99,999	100,000	17
5½	0 to 199,999	200,000	18
5¾	0 to 399,999	400,000	19
6	0 to 999,999	1,000,000	20
6½	0 to 199,999	2,000,000	21
6¾	0 to 3,999,999	4,000,000	22
N	0 to 10^N-1	10^N	
N½	0 to $2*10^N$-1	$2*10^N$	
N¾	0 to $4*10^N$-1	$4*10^N$	

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., **`%11001000`**
hexadecimal e.g., **`$C8`**

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~~
~~**ldaa 4,x**~~
~~**ldaa 40,x**~~
~~**ldaa -4,x**~~
~~**ldaa -40,x**~~
~~**ldaa $400,x**~~
~~**ldaa 4,+x**~~
~~**ldaa 4,-x**~~
~~**ldaa 4,x+**~~
~~**ldaa 4,x-**~~
~~calculate effective address~~
~~go from assembly to machine code **xb**~~
~~go from machine code **xb** to assembly~~
simple multiply and divide (**mul idiv** ~~**fdiv**~~)
stack functions for **bsr** and **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., **if PT4 is low then make PM2 high**
****study question******
8-bit operations
add, sub, shift left, shift right, and, or, eor
if-then like examples in Chapter 5
if-then-else
if((uG1>5)&&(uG2<100)){♪♫♪♫♬}
while-loop like examples in Chapter 5
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 **bsr,** returns using **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

```
void TogglePT0(void){
  PTT = PTT^0x01;
}
void main(void){
  DDRT = DDRT|0x01;
  while(1){
    TogglePT0();
  }
}
```

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

```
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**)

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

How to read from and write to C variables

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

Simple calculations

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

Conditional structures

Compare operators	`==  !=  <  <=  >  >=`
Boolean operators	`&&  ||`

If-then

```
 if(D2 < D1){
   D2isLess();
 }
 if(((PTT&0x08)==0x08)&&((PTH&0x03)==0)){
   PT3HighAndPH3210Low();
 }
```

If-then-else

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

Looping structures

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

```
 while(D2 < 100){
   OverAndOver(); // repeat while D2<100
 }
```

do-while-loop (test after each execution of the body)

```
do{
  OverAndOver(); // repeat while D2<100
} while(D2 < 100);
```

for-loop (test before each execution of the body)

```
for(D2=0; D2<100; D2=D2+1){
  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, **bhi blo bhs** and **bls**
  - signed, **bgt bls bge** and **ble**
  - either signed or unsigned, **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

Compare the four possible inequalities

Assume **PTT** is a unsigned 8-bit input port, and
let **Threshold** be an unsigned 8-bit global variable

C code	assembly code

`if(PTT > Threshold){`
`    ♪♫♪♫♫`
`}`
	`   ldab PTT`
	`   cmpb Threshold`
	`   bls  next`
	`   ♪♫♪♫♫`
	`next`

`if(PTT >= Threshold){`
`    ♪♫♪♫♫`
`}`
	`   ldab PTT`
	`   cmpb Threshold`
	`   blo  next`
	`   ♪♫♪♫♫`
	`next`

`if(PTT < Threshold){`
`    ♪♫♪♫♫`
`}`
	`   ldab PTT`
	`   cmpb Threshold`
	`   bhs  next`
	`   ♪♫♪♫♫`
	`next`

`if(PTT <= Threshold){`
`    ♪♫♪♫♫`
`}`
	`   ldab PTT`
	`   cmpb Threshold`
	`   bhi  next`
	`   ♪♫♪♫♫`
	`next`

Compare signed versus unsigned conditionals
Assume **uG** is an unsigned 8-bit global variable
Assume **sG** is a signed 8-bit global variable

C code	assembly code

`if(uG >= 5){`
`    ♪♫♪♫♫`
`}`
	`    ldaa uG`
	`    cmpa #5`
	`    blo  next`
	`     ♪♫♪♫♫`
	`next`

`if(sG >= 5){`
`    ♪♫♪♫♫`
`}`
	`    ldaa sG`
	`    cmpa #5`
	`    blt  next`
	`     ♪♫♪♫♫`
	`next`

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

C code	assembly code

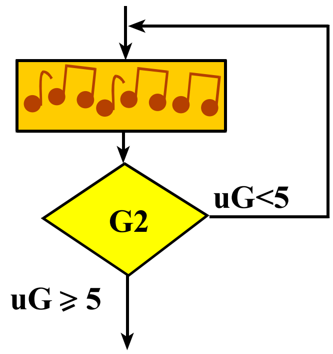
`if(uG2 >= uG1){`
`    ♪♫♪♫♫`
`}`
	`    ldaa uG2`
	`    cmpa uG1`
	`    blo  next`
	`     ♪♫♪♫♫`
	`next`

`if(uH2 >= uH1){`
`    ♪♫♪♫♫`
`}`
	`    ldd  uH2`
	`    cpd  uH1`
	`    blo  next`
	`     ♪♫♪♫♫`
	`next`

```
      for(uG=0;uG<5;uG++){
        PTT = uG;
      }
      clr  uG
loop  ldaa uG
      cmpa #5
      bhs  next   ; stop when uG>=5
      ldaa uG
      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
      ldaa #5     ; loop 5 down to 0
loop  ldab PTT  ; body of for loop
      eorb #$02 ;toggle PT2
      stab PTT
      dbne A,loop
```

```
do{
  ♪♫♪♫♫
}
while(uG < 5);
```




```
loop  ♪♫♪♫♫       ; body of while loop
      ldaa uG
      cmpa #5
      blo  loop   ; stop when uG>=5


loop
      ♪♫♪♫♫       ; body of while loop
      bra  loop
```

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