So, you basically didn't study and now you're in the final and you hope to pass this test and save your miserable grade... And you expect this cheat sheet to save you!? Well, I sincerely hope it does.

Slacker. ;)

Registers:

A - One byte register
B - One byte register
D - Two byte register (A and B combined to make A:B)
X - Two byte register
Y - Two byte register

Immediate vs. Extended mode:

Immediate values are marked with a # symbol. They also are different instructions when assembled.

EX)

```
 LDAA #$00   Loads $00 into A
LDAA $2000  Loads the data at $2000 into A
```

Variable declarations:

EX)

```
array  DB  $00,$01,$02,$03
```

Remember: Variable names are replaced by their addresses in memory when the program is assembled.

Constants:

EX)

```
ZERO  EQU  $00
```

Remember: Every instance of "ZERO" in the program will be replaced with a "$00". So use #s when necessary.

Stacks:

Stacks start from a memory location and move up.

EX) Instruction	Stack pointer	Notes
LDS #$2000	$2000	Set SP to $2000
PSHA	$1FFF	Pushed one byte...
PSHD	$1FFD	Pushed two bytes...

Remember: When pushing a two byte accumulator on to the stack, the low order byte is pushed first. (ie: The data in B is pushed, then the data in A is pushed when pushing D.)

The CCR:

The bits look like:

SXHINZVC

S - Dunno
X - Dunno
H - Dunno
I - Dunno
N - The negative bit
Z - The zero bit
V - The overflow bit
C - The carry bit

Notes:

The N bit is set when an operation results in a negative number.

The Z bit is set when an operation results in a zero.

The V bit is set when an operation results in an overflow.

The C bit is set when an operation results in a carry.

Comparison and Branching:

EX)

```
LDAA #$33
CMPA #$0F

BGE - Is $33 >= $0F? - Yes, branch
BLE - Is $33 <= $0F? - No
BGT - Is $33 > $0F?  - Yes, branch
BLT - Is $33 < $0F?  - No
BEQ - Is $33 == $0F? - No
BNE - Is $33 != $0F? - Yes, branch
```

Unit conversion:

Binary to Hex and Hex to Binary:

Simply, every hex "digit" equates to four binary "digits":

- 0 - 0000
- 1 - 0001
- 2 - 0010
- 3 - 0011
- 4 - 0100
- 5 - 0101
- 6 - 0110
- 7 - 0111
- 8 - 1000
- 9 - 1001
- A - 1010
- B - 1011
- C - 1100
- D - 1101
- E - 1110
- F - 1111

EX)

AF = 1010 1111

Remember: The most significant bit (128) in a signed value is actually -128.

Decimal to Binary:

The principle is to cram as many high value bits in as possible.

(The example is long and thus, on the next page.)

EX)
94 = 0 (128 does not fit in 94)
     1 (64 does fit in 94) (94-64=30)
     0 (32 does not fit in 30)
     1 (16 does fit in 30) (30-16=14)
     1 (8 does fit in 14) (14-8=6)
     1 (4 does fit in 6) (6-4=2)
     1 (2 does fit in 2) (2-2=0)
(LSB)--> 0 (1 does not fit in 0)
Thus, 94 = 01011110

Remember: The most significant bit (128) in a signed value is actually -128.

Decimal to Binary:
Each bit is a power of two, add up the bits, multiplying them by the powers of two.

EX)
01011110 = 128(0) + 64(1) + 32(0) + 16(1) + 8(1) +
           4(1) + 2(1) + 1(0)
         = 94

Remember: The most significant bit (128) in a signed value is actually -128.

Decimal to Hex/Hex to Decimal:
I recommend you first convert into binary, then to the desired type.

EX)
Hex -> Binary -> Decimal
Decimal -> Binary -> Hex

Remember: The most significant bit (128) in a signed value is actually -128.

2's compliment:

Known as "flip and add one". You can also subtract from $FF (or all 1s in binary) to flip.

EX)

```
  00111111 (63)
 (Flip 'em)
  11000000 (-64) (-128+64)

  11000000
+ 00000001
----------
  11000001 (-63)
```

Binary Addition and Subtraction:

Addition:

It's easy, just add each column and carry if necessary.

EX)

```
       1 <---Carry out
  11001011
+ 10010010
----------
1 01011101
```

Subtraction:

This is a bit more difficult, you must subtract each column, and barrow from a higher order bit if necessary, When you do barrow, remember that you are now subtracting 10 - 1, (2 - 1 in decimal) which results in 1.

EX)

```
    1 <---Carry in
  11001011
- 10100010
----------
  00101001
```

Bit shifting:

Logical shift left:

All bits are shifted to the left, with a zero being introduced in the lowest order bit, and the highest order bit spills over into the carry bit.

[C]<--[7][6][5][4][3][2][1][0]<--[Zero]

It can also be thought of as “multiply by 2” for both signed and unsigned values. As such, it is also the same as arithmetic shift left.

Logical shift right:

All bits are shifted to the right, with a zero being introduced in the highest order bit, and the lowest order bit spills over into the carry bit.

[Zero]-->[7][6][5][4][3][2][1][0]-->[C]

It can also be thought of as “divide by 2”, but only for only unsigned values. As such, it is not the same as arithmetic shift right.

Arithmetic shift left:

This is the same as logical shift left.

Arithmetic shift right:

All bits are shifted to the right, with the highest order bit pushing it’s own value back in. The lowest order bit overflows into the carry bit.

[7]--->[7][6][5][4][3][2][1][0]-->[C]

Pre and Post Increment:

It’s just best to give examples...

EX)

```
LDAA 0,X ;Load A with the data at the memory address in X
LDAA 1,X ;Load A with the data at the memory address in X+1
LDAA 1,X+ ;Load A with data pointed to by X, then inc. X
LDAA 1,X- ;Load A with data pointed to by X, then dec. X
LDAA +1,X ;Increment X, load A with data pointed to by X
LDAA -1,X ;Decrement X, load A with data pointed to by X
```

Ports:

Ports are the way the microcontroller interacts with real world devices using it's digital I/O pins. In this class, we only use port B and port B. Be sure to include hcs12.inc in your program if you are using ports.

EX)

```
#include hcs12.inc
```

Port B:

In the Dragon-12 demo board that we use, port B is hooked up to both the LEDs and the seven segment displays. The LEDs displays the byte currently in port B. This port has a register that determines whether this port will be used for input or output. It's called the data direction register, in this case, ddrb.

EX)

```
LDAA #$FF  ;$FF is used to set for output mode
STAA ddrb  ;Set port B to output

LDAA #%10101010
STAA portb  ;Light the LEDs with 10101010
```

Port H:

In the Dragon-12 demo board that we use, port H is hooked up to those really tiny, annoying to use switches. Each switch corresponds with each bit in a byte. It also has it's own data direction register which is often set to input mode.

EX)

```
LDAA #$00  ;$00 is used to set for input mode
STAA ddrh  ;Set port H to input

LDAA pth  ;Load the switch states from port H
```

Important: No, that's not a typo, port H is really called "pth" even though port B is called "portb".

Interrupts:

Honestly, this makes zero sense, use the attached interrupt worksheet and lab code.

Relative branching/jumping:

The offset is calculated by subtracting the destination address with the next program counter value. (The one after the jump instruction.)

EX) (Note the offset goes in the blank next to "26")

```
  PC  Machine Code Instruction
$3006    8B 03       ADDA #3    <---Jumping to here
$3008    73 2000     DEC count
$300B    26 __       BNE AGAIN  <---Jumping from here
$300D    <---The address of the next instruction
```

The calculate the offset:
$3006 - $300D = -7 = F9

General oddities:

-When doing any operation, the CCR may change depending on the result of that operation. Shift lefts on a value that goes negative to positive, the overflow bit (V bit) is set.

```
EX)                                    CCR Bits
LDAA #%10000000  ;Before LSLA:         N=1, V=0
LSLA             ;A is now 00000000, N=0, V=1
```

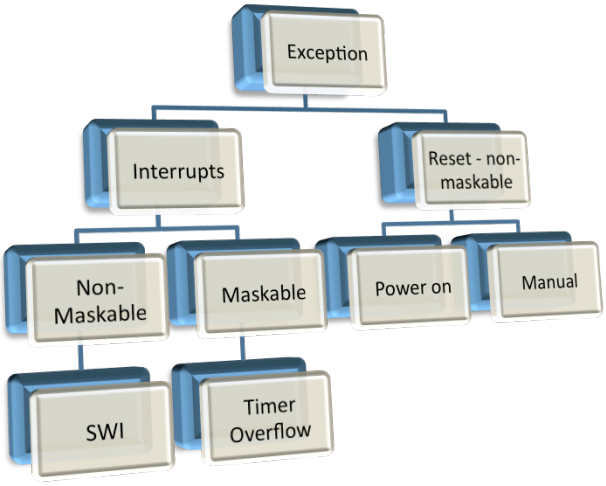
-When jumping to a subroutine and using stacks, remember that the jumpback address is pushed to the stack when you JSR.

CSC 202
Interrupts

1. What is an interrupt?

2. Uses/Functions

3. Types



4. Interrupt Process
   - Disable Interrupts
   - Save the Program Counter
   - Save CPU registers
   - Determine cause of Interrupt
   - Get the address (interrupt vector) of the ISR from the Interrupt Vector Table
   - Execute the ISR
   - RTI – Return from Interrupt instruction
     - Restore CPU status and PC
     - Enable future interrupts
   - Restart the interrupted program

5. Priority

Interrupt Program

- What does this program do?
    - Demonstrates the TCNT and timer overflow
    - Allows us to modify the speed of the TCNT
    - Suppose we need to count longer than $000-$fff?
    - Use a second loop.
    - Use the switches to input the number of times the second loop will execute
    - Each timer overflow causes an interrupt
    - The LEDs will stay on for a certain number of overflows
    - Repeat the process with the LEDs off.
    - Look at the speed of the ON and OFF of the LEDs with the change of the DIPsw.

- Program Logic

```
for tofcnt = Y downto 1                    ; Y set by DIPsw
        for TCNT = 0 to 65,535             ;TCNT overflow
        end
end
```

- Program Set up
    - Write the code for the ISR
    - Set the location of the ISR
    - Set up ports
    - Set register bits

Let's look at the Flow Chart and then the code.

Timer System

Uses

- Count events
- Measure a signal
- Can be used to cause an interrupt signaling something to occur

TCNT

- 16 bit counter ($0044)
- $0000-$ffff – rolls over
- Programmer can cause an interrupt on rollover
- Can extend the range of an event by using another counter

Registers and flags used by the Timer system

- TSCR1 – Timer System Control Register 1 ($0046)
    - **TEN** – Timer Enable, bit 7
    - Enables the Timer System
    - bset TSCR1, $80
- TSCR2 - Timer System Control Register 2 ($004d)
    - **TOI** - Timer Overflow Interrupt Enable, bit 7
    - Enables Timer System Interrupts
    - bset TSCR2, $80
    - bits 0,1,2 used to set a prescaler
- TFLG2 – Timer Interrupt Flag 2 Register ($004f)
    - **TOF** - Timer Overflow Flag, bit 7
    - Signals that an overflow has occurred
    - Programmer clears the flag by writing a 1 to it
    - bset TFLG2, $80 – flag = 0
    - When a timer overflow occurs, flag is set to 1 by the system
    - Programmer must clear the flag after the interrupt

Questions:

- What is the purpose of the Timer Module?
- What are the 3 registers used by the Timer Module?
- How do we configure these registers to allow for interrupts?

.

Prescaler

- 68hcs12 clock speed – 24 MHZ = 24,000,000 cycles/second
- TCNT – 16 bit counter - $0000-$ffff so 65,536 cycles to roll over. User can cause an interrupt to occur on rollover
- Counter speed, the amount of time it takes for one rollover.
- Speed of the counter can be modified by using a Prescaler

Prescaler – bits 0,1,2 0f TSCR2 used to modify counter speed - TCNT	Rollover Frequency How often does the TCNT rollover?	Number of Rollovers of the TCNT per second
Prescaler factor	(Prescaler*#of cycles for 1 rollover)/ (#cycles/second)	(#cycles/second)/ #of cycles for 1 rollover
1 2^X=1 X = 0 000	1*65,536/24,000,000 65536/24,000,000 .0027306 **one roll over every 2.73ms**	24,000,000/65,536 **366.2 rollovers/sec**
32 2^X=32 X = 5 101	32*65,536/24,000,000 2,097,152/24,000,000 **87.381ms**	24,000,000/2,097,152 **11.44**
64 ???	64*65,536/24,000,000 4,194,304/24,000,000 **174.76ms**	24,000,000/4,194,304 **5.72**
128 ???		
255 (not a true prescaler) DIP sw = 11111111 or $ff		

Questions

- What is the speed of the HCS12 processor?
- Fill in the values in the table for the prescaler 128 and the pseudo prescaler 255
- How will the different prescalers affect the behavior of the LEDs?
- How would you set up the Timer System with a prescaler factor of 8?

```
; Ex1.asm  ---- Example program 1 for the Ep9S12DP256 board (c)2002, EVBplus.com
;               Written by Wayne Chu. Modified by Barry Walker and Lorraine D'Ortona
;
; Function:     Demonstrates HCS12 interupts, ports, and Timer Module
;
#include        hcs12.inc

        ORG     $2000

tofcnt  DS      1               ;counts the timer overflows, initialized by DIPsw

        ORG     $3e5e           ;address of the ISR will be stored at $3e5e
        DW      TOFISR          ;address of the ISR


        ORG     $2100           ;program code starts here

        ldaa    #$ff
        staa    ddrb            ;make port B an output port
        ldaa    #$00
        staa    ddrh            ;make port H an input port


        bset    TSCR1,$80       ;enable Timer Module
        bset    TSCR2,$80       ;enable Timer Interrupts
        bset    TFLG2,$80       ;clear Timer Overflow Flag
        cli                     ;clear interrupt bit (I) in CCR



REPEAT  ldaa    pth             ;read from DIPsw (port H).
        staa    tofcnt          ;initialize the timer overflow counter from DIPsw
        swi
        bset    TSCR1,$80

************************
        ldaa    #$FF            ;turn on all LEDs (port B)
        staa    portb
BACK    wai                     ;wait for an interrupt - TCNT rollover
        dec     tofcnt          ;after ISR executes; decrement timer overflow counter
        bne     BACK            ;if DIPsw value not done
************************

        ldaa    pth             ;read from DIPsw (port H)
        staa    tofcnt          ;initialize the timer overflow counter from DIPsw

************************
        ldaa    #$00            ;turn off all LEDs (port B)
        staa    portb
BACK2   wai                     ;wait for an interrupt
        dec     tofcnt          ;after ISR executes; decrement timer overflow counter
        bne     BACK2           ;if DIPsw value not done
************************

        bra     REPEAT          ;start over, get port H input value



******Interrupt Service Routine**************************

TOFISR  NOP
        swi
        bset    TFLG2,$80       ;clear timer overflow flag
```

```
bset    TFLG2,$80       ;clear Timer Overflow Flag (bit 7)
RTI                     ;return to statement following interrupt
                        ;return the state of the CPU
END
```

8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

0	0	0	0	0	
0	0	0	1	1	
0	0	1	0	2	
0	0	1	1	3	
0	1	0	0	4	
0	1	0	1	5	
0	1	1	0	6	
0	1	1	1	7	
1	0	0	0	8	
1	0	0	1	9	
1	0	1	0	A	10
1	0	1	1	B	11
1	1	0	0	C	12
1	1	0	1	D	13
1	1	1	0	E	14
1	1	1	1	F	15