

Exam 1  
Monday, Feb. 20

- You will be able to use all of the Motorola data manuals on the exam.
- No calculators will be allowed for the exam.
- Numbers
  - Decimal to Hex (signed and unsigned)
  - Hex to Decimal (signed and unsigned)
  - Binary to Hex
  - Hex to Binary
  - Addition and subtraction of fixed-length hex numbers
  - Overflow, Carry, Zero, Negative bits of CCR
- Programming Model
  - Internal registers – A, B, (D = AB), X, Y, SP, PC, CCR
- Addressing Modes and Effective Addresses
  - INH, IMM, DIR, EXT, REL, IDX (Not Indexed Indirect)
  - How to determine effective address
- Instructions
  - What they do - Core Users Guide
  - What machine code is generated
  - How many cycles to execute
  - Effect on CCR
  - Branch instructions – which to use with signed and which with unsigned
- Machine Code
  - Reverse Assembly
- Stack and Stack Pointer
  - What happens to stack and SP for instructions (e.g., PSHX, JSR)
  - How the SP is used in getting to and leaving subroutines

- Assembly Language
  - Be able to read and write simple assembly language program
  - Know basic assembler directives – e.g., equ, dc.b, ds.w
  - Flow charts

Timer inside the 68HC12:

When you enable timer (by writing a 1 to bit 7 of TCSR),  
you connect an 24-MHz oscillator to a 16-bit counter.

You can read the counter at address TCNT.

The counter will start at 0, will count to 0xFFFF, then  
roll over to 0x0000. It will take 2.7307 ms for this to happen.



To enable timer on HC12, set Bit 7 of register TCSR:

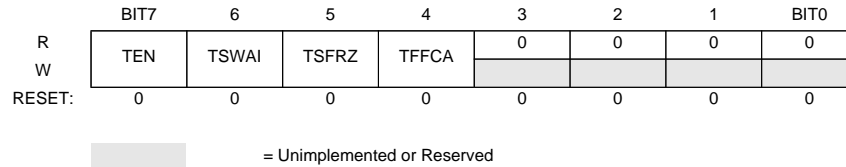
```
bset TCSR1, #80
```

```
TCSR1 = TCSR1 | 0x80;
```

ECT\_16B8C Block User Guide V01.03

**3.3.6 TSCR1 — Timer System Control Register 1**

Register offset: \$\_06

**Figure 3-6 Timer System Control Register 1 (TSCR1)**

Read or write anytime.

**TEN** — Timer Enable

- 0 = Disables the main timer, including the counter. Can be used for reducing power consumption.
- 1 = Allows the timer to function normally.

If for any reason the timer is not active, there is no  $\div 64$  clock for the pulse accumulator since the  $\div 64$  is generated by the timer prescaler.

**TSWAI** — Timer Module Stops While in Wait

- 0 = Allows the timer module to continue running during wait.
- 1 = Disables the timer module when the MCU is in the wait mode. Timer interrupts cannot be used to get the MCU out of wait.

TSWAI also affects pulse accumulators and modulus down counters.

**TSFRZ** — Timer and Modulus Counter Stop While in Freeze Mode

- 0 = Allows the timer and modulus counter to continue running while in freeze mode.
- 1 = Disables the timer and modulus counter whenever the MCU is in freeze mode. This is useful for emulation.

TSFRZ does not stop the pulse accumulator.

**TFFCA** — Timer Fast Flag Clear All

- 0 = Allows the timer flag clearing to function normally.
- 1 = For TFLG1(\$0E), a read from an input capture or a write to the output compare channel (\$10–\$1F) causes the corresponding channel flag, CnF, to be cleared. For TFLG2(\$0F), any access to the TCNT register (\$04, \$05) clears the TOF flag. Any access to the PACN3 and PACN2 registers (\$22, \$23) clears the PAOVF and PAIF flags in the PAFLG register (\$21). Any access to the PACN1 and PACN0 registers (\$24, \$25) clears the PBOVF flag in the PBFLG register (\$31). This has the advantage of eliminating software overhead in a separate clear sequence. Extra care is required to avoid accidental flag clearing due to unintended accesses.

### 3.3.4 OC7D — Output Compare 7 Data Register

Register offset: `$_03`

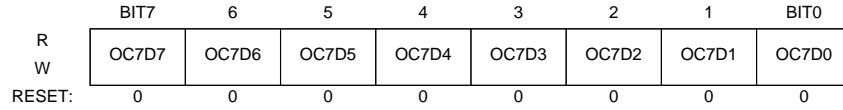


Figure 3-4 Output Compare 7 Data Register (OC7D)

Read or write anytime.

A channel 7 output compare can cause bits in the output compare 7 data register to transfer to the timer port data register depending on the output compare 7 mask register.

### 3.3.5 TCNT — Timer Count Register

Register offset: `$_04-$_05`

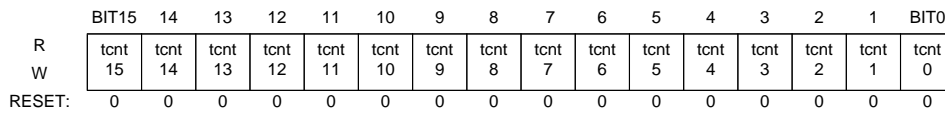


Figure 3-5 Timer Count Register (TCNT)

The 16-bit main timer is an up counter.

A full access for the counter register should take place in one clock cycle. A separate read/write for high byte and low byte will give a different result than accessing them as a word.

Read anytime.

Write has no meaning or effect in the normal mode; only writable in special modes (`test_mode = 1`).

The period of the first count after a write to the TCNT registers may be a different size because the write is not synchronized with the prescaler clock.

- To put in a delay of 2.7307 ms, you could wait from one reading of 0x0000 to the next reading of 0x0000.
- **Problem:** You cannot read the TCNT register quickly enough to make sure you will see the 0x0000.

To put in a delay for 2.7307 ms, could watch timer until

TCNT = 0x0000:

```

    bset  TSCR1, #80          TSCR1 = TSCR1 | 0x80;
l1:  ldd  TCNT              while (TCNT != 0x0000) ;
    bne  l1

```

**Problem:** You might see 0xFFFF and 0x0001, and miss 0x0000



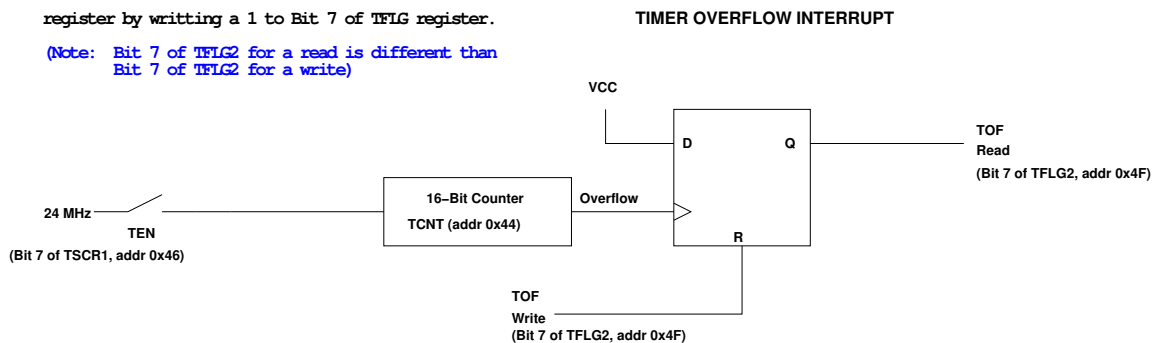
- **Solution:** The 9S12 has built-in hardware with will set a flip-flop every time the counter rolls over from 0xFFFF to 0x0000.
- To wait for 2.7307 ms, just wait until the flip-flop is set, then clear the flip-flop, and wait until the next time the flip-flop is set.
- You can find the state of the flip-flop by looking at bit 7 (the Timer Overflow Flag (TOF) bit) of the Timer Flag Register 2 (TFLG2) register at address 0x004F.
- You can clear the flip-flop by writing a 1 to the TOF bit of TFLG2.

Solution: When timer overflows, latch a 1 into a flip-flop.

Now when timer overflows (goes from 0xFFFF to 0x0000),

Bit 7 of TFLG2 register is set to one. Can clear register by writing a 1 to Bit 7 of TFLG2 register.

(Note: Bit 7 of TFLG2 for a read is different than Bit 7 of TFLG2 for a write)



```

bset   TSCR1, #80      ; Enable timer
11:   brclr  TFLG2, #80, 11 ; Wait until Bit 7 of TFLG2 is set
      ldaa  #80
      staa  TFLG2      ; Clear TOF flag


      TSCR1 = TSCR1 | 0x80;           //Enable timer
      while ((TFLG2 & 0x80) == 0) ; // Wait for TOF
      TFLG2 = 0x80;                 // Clear TOF

```

### 3.3.13 TFLG2 — Main Timer Interrupt Flag 2

Register offset: `$_OF`

	BIT7	6	5	4	3	2	1	BIT0
R	TOF	0	0	0	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

 = Unimplemented or Reserved

**Figure 3-13 Main Timer Interrupt Flag 2 (TFLG2)**

TFLG2 indicates when interrupt conditions have occurred. To clear a bit in the flag register, write the bit to one.

Read anytime. Write used in clearing mechanism (set bits cause corresponding bits to be cleared).

Any access to TCNT will clear TFLG2 register if the TFFCA bit in TSCR register is set.

TOF — Timer Overflow Flag

Set when 16-bit free-running timer overflows from \$FFFF to \$0000. This bit is cleared automatically by a write to the TFLG2 register with bit 7 set. (See also TCRE control bit explanation.)



- Another problem: Sometimes you may want to delay longer than 2.7307 ms, or time an event which takes longer than 2.7307 ms. This is hard to do if the counter rolls over every 2.7307 ms.
- Solution: The 9S12 allows you to slow down the clock which drives the counter.
- You can slow down the clock by dividing the 24 MHz clock by 2, 4, 8, 16, 32, 64 or 128.
- You do this by writing to the prescaler bits (PR2:0) of the Timer System Control Register 2 (TSCR2) Register at address 0x004D.

2.7307 ms will be too short if you want to see lights flash.

You can slow down clock by dividing it before you send it to the 16-bit counter. By setting prescaler bits PR2,PR1,PR0 of TSCR2 you can slow down the clock:

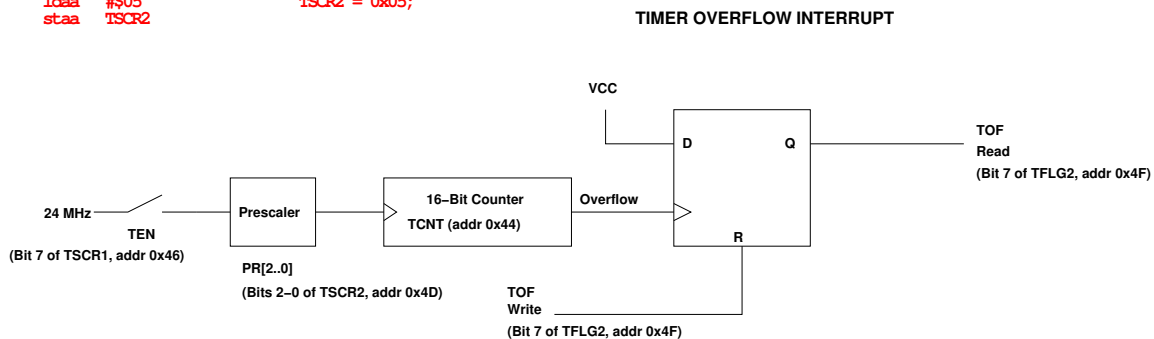
PR2:0 Divide	Freq	Overflow Rate
000	1 24 MHz	2.7307 ms
001	2 12 MHz	5.4613 ms
010	4 6 MHz	10.9227 ms
011	8 3 MHz	21.8453 ms
100	16 1.5 MHz	43.6907 ms
101	32 0.75 MHz	87.3813 ms
110	64 0.375 MHz	174.7627 ms
111	128 0.1875 MHz	349.5253 ms

To set up timer so it will overflow every 87.3813 ms:

```

bset  TSCR1, #80      TSCR1 = TSCR1 | 0x80;
ldaa  #05            TSCR2 = 0x05;
staa  TSCR2

```



### 3.3.10 TIE — Timer Interrupt Enable Register

Register offset: \$\_0C

	BIT7	6	5	4	3	2	1	BIT0
R	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I
W								
RESET:	0	0	0	0	0	0	0	0

Figure 3-10 Timer Interrupt Enable Register (TIE)

Read or write anytime.

The bits in TIE correspond bit-for-bit with the bits in the TFLG1 status register. If cleared, the corresponding flag is disabled from causing a hardware interrupt. If set, the corresponding flag is enabled to cause an interrupt.

C7I–C0I — Input Capture/Output Compare “x” Interrupt Enable

### 3.3.11 TSCR2 — Timer System Control Register 2

Register offset: \$\_0D

	BIT7	6	5	4	3	2	1	BIT0
R	TOI	0	0	0	TCRE	PR2	PR1	PR0
W								
RESET:	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 3-11 Timer System Control Register 2 (TSCR2)

Read or write anytime.

TOI — Timer Overflow Interrupt Enable

0 = Interrupt inhibited

1 = Hardware interrupt requested when TOF flag set

TCRE — Timer Counter Reset Enable

This bit allows the timer counter to be reset by a successful output compare 7 event. This mode of operation is similar to an up-counting modulus counter.

0 = Counter reset inhibited and counter free runs

1 = Counter reset by a successful output compare 7

If TC7 = \$0000 and TCRE = 1, TCNT will stay at \$0000 continuously. If TC7 = \$FFFF and TCRE = 1, TOF will never be set when TCNT is reset from \$FFFF to \$0000.

PR2, PR1, PR0 — Timer Prescaler Select

## ECT\_16B8C Block User Guide V01.03

These three bits specify the number of  $\div 2$  stages that are to be inserted between the bus clock and the main timer counter.

Table 3-4 Prescaler Selection

PR2	PR1	PR0	Prescale Factor
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

The newly selected prescale factor will not take effect until the next synchronized edge where all prescale counter stages equal zero.

## 3.3.12 TFLG1 — Main Timer Interrupt Flag 1

Register offset:  $\$_{0E}$

	BIT7	6	5	4	3	2	1	BIT0
R								
W								
RESET:	0	0	0	0	0	0	0	0

Figure 3-12 Main Timer Interrupt Flag 1 (TFLG1)

TFLG1 indicates when interrupt conditions have occurred. To clear a bit in the flag register, write a one to the bit.

Use of the TFMOD bit in the ICSYS register ( $\$2B$ ) in conjunction with the use of the ICOVW register ( $\$2A$ ) allows a timer interrupt to be generated after capturing two values in the capture and holding registers instead of generating an interrupt for every capture.

Read anytime. Write used in the clearing mechanism (set bits cause corresponding bits to be cleared). Writing a zero will not affect current status of the bit.

When TFFCA bit in TSCR register is set, a read from an input capture or a write into an output compare channel ( $\$10$ – $\$1F$ ) will cause the corresponding channel flag CnF to be cleared.

C7F–C0F — Input Capture/Output Compare Channel “n” Flag.

C0F can also be set by 16 - bit Pulse Accumulator B (PACB). C3F - C0F can also be set by 8 - bit pulse accumulators PAC3 - PAC0.

## What Happens When You Reset the HCS12?

- What happens to the HCS12 when you turn on power or push the reset button?
- How does the HCS12 know which instruction to execute first?
- On reset the HCS12 loads the PC with the address located at address 0xFFFFE and 0xFFFF.
- Here is what is in the memory of our HCS12:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
FFF0	F6	EC	F6	F0	F6	F4	F6	F8	F6	FC	F7	00	F7	04	F0	00

- On reset or power-up, the first instruction your HCS12 will execute is the one located at address 0xF000.

## Using the Timer Overflow Flag to implement a delay

- The HCS12 timer counts at a rate set by the prescaler:

PR2:0	Divide	Clock Freq	Clock Period	Overflow Period
000	1	24 MHz	0.042 $\mu$ s	2.73 ms
001	2	12 MHz	0.083 $\mu$ s	5.46 ms
010	4	6 MHz	0.167 $\mu$ s	10.92 ms
011	8	3 MHz	0.333 $\mu$ s	21.85 ms
100	16	1.5 MHz	0.667 $\mu$ s	43.69 ms
101	32	750 kHz	1.333 $\mu$ s	87.38 ms
110	64	375 kHz	2.667 $\mu$ s	174.76 ms
111	128	187.5 kHz	5.333 $\mu$ s	349.53 ms

- When the timer overflows it sets the TOF flag (bit 7 of the TFLG2 register).
- To clear the TOF flag write a 1 to bit 7 of the TFLG2 register, and 0 to all other bits of TFLG2:

```
TFLG2 = 0x80;
```

- You can implement a delay using the TOF flag by waiting for the TOF flag to be set, then clearing it:

```
void delay(void)
{
    while ((TFLG2 & 0x80) == 0) ;    /* Wait for TOF */
    TFLG2 = 0x80;                    /* Clear flag */
}
```

- If the prescaler is set to 010, you will exit the delay subroutine after 10.92 ms have passed.

## Introduction to Interrupts

Can implement a delay by waiting for the TOF flag to become set:

```
void delay(void)
{
    while ((TFLG2 & 0x80) == 0) ;
    TFLG2 = 0x80;
}
```

**Problem:** Can't do anything else while waiting. Wastes resources of HCS12.

**Solution:** Use an interrupt to tell you when the timer overflow has occurred.

**Interrupt:** Allow the HCS12 to do other things while waiting for an event to happen. When the event happens, tell HCS12 to take care of event, then go back to what it was doing.

**What happens when HCS12 gets an interrupt:** HCS12 automatically jumps to part of the program which tells it what to do when it receives the interrupt (Interrupt Service Routine).

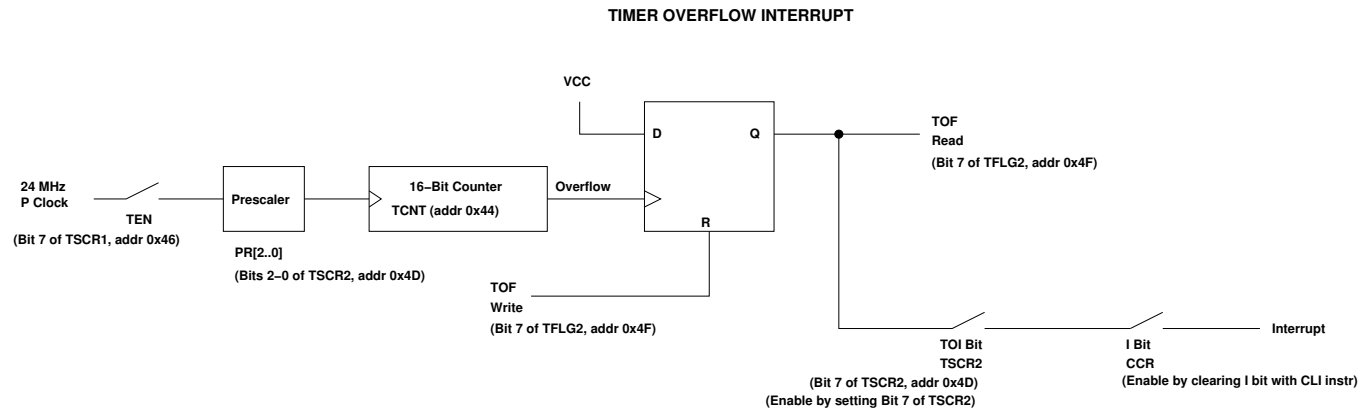
**How does HCS12 know where the ISR is located:** A set of memory locations called Interrupt Vectors tell the HCS12 the address of the ISR for each type of interrupt.

**How does HCS12 know where to return to:** Return address pushed onto stack before HCS12 jumps to ISR. You use the RTI (Return from Interrupt) instruction to pull the return address off of the stack when you exit the ISR.

**What happens if ISR changes registers:** All registers are pushed onto stack before jumping to ISR, and pulled off the stack before returning to program. When you execute the RTI instruction at the end of the ISR, the registers are pulled off of the stack.

**To Return from the ISR** You must return from the ISR using the RTI instruction. The RTI instruction tells the HCS12 to pull all the registers off of the stack and return to the address where it was processing when the interrupt occurred.

## How to generate an interrupt when the timer overflows



To generate a TOF interrupt:

Enable timer (set Bit 7 of TSCR1)  
 Set prescaler (Bits 2:0 of TSCR2)  
 Enable TOF interrupt (set Bit 7 of TSCR2)  
 Enable interrupts (clear I bit of CCR)

Inside TOF ISR:

Take care of event  
 Clear TOF flag (Write 1 to Bit 7 of TFLG2)  
 Return with RTI

```
#include "hcs12.h"
```

```
main()
{
```

```
    DDRA = 0xff; /* Make Port A output */
    TSCR1 = 0x80; /* Turn on timer */
    TSCR2 = 0x85; /* Enable timer overflow interrupt, set prescaler */
    TFLG2 = 0x80; /* Clear timer interrupt flag */
    enable(); /* Enable interrupts (clear I bit) */
    while (1)
    {
        /* Do nothing */
    }
}
```

```
void INTERRUPT toi_isr(void)
```

```
{
    PORTA = PORTA + 1; /* Increment Port A */
    TFLG2 = 0x80; /* Clear timer interrupt flag */
}
```

## How to tell the HCS12 where the Interrupt Service Routine is located

- You need to tell the HCS12 where to go when it receives a TOF interrupt
- You do this by setting the TOF Interrupt Vector
- The TOF interrupt vector is located at `0xFFDE`. This is in flash EPROM, and is very difficult to change — you would have to modify and reload DBug-12 to change it.
- DBug-12 redirects the interrupts to a set of vectors in RAM, from `0x3E00` to `0x3E7F`. The TOF interrupt is redirected to `0x3E5E`. When you get a TOF interrupt, the HCS12 initially executes code starting at `0xFFDE`. This code tells the HCS12 to load the program counter with the address in `0x3E5E`. Because this address is in RAM, you can change it without having to modify and reload DBug-12.
- Because the redirected interrupt vectors are in RAM, you can change them in your program.



## How to Use Interrupts in C Programs

- For our C compiler, you can set the interrupt vector by including the file `vectors12.h`. In this file, pointers to the locations of all of the 9212 interrupt vectors are defined.
- For example, the pointer to the Timer Overflow Interrupt vector is called `UserTimerOvf`:

```
#define VECTOR_BASE 0x3E00
#define _VEC16(off)      *(volatile unsigned short *) (VECTOR_BASE + off)
#define UserTimerOvf _VEC16(47)
```

You can set the interrupt vector to point to the interrupt service routine `toi_isr()` with the C statement:

```
UserTimerOvf = (unsigned short) &toi_isr;
```

- Here is a program where the interrupt vector is set in the program:

```
#include <hcs12.h>
#include <vectors12.h>
#include "DBug12.h"

#define enable() _asm(" cli")
#define disable() _asm(" sei")

void INTERRUPT toi_isr(void);

main()
{
    DDRA = 0xff;          /* Make Port A output */
    TSCR1 = 0x80;        /* Turn on timer */
    TSCR2 = 0x86;        /* Enable timer overflow interrupt, set prescaler
                          so interrupt period is 175 ms */
    TFLG2 = 0x80;        /* Clear timer interrupt flag */

    UserTimerOvf = (unsigned short) &toi_isr;

    enable();            /* Enable interrupts (clear I bit) */
    while (1)
    {
        /* Do nothing - go into low power mode */
    }
}

void INTERRUPT toi_isr(void)
{
    PORTA = PORTA+1;
    TFLG2 = 0x80;        /* Clear timer interrupt flag */
}
```

## How to Use Interrupts in Assembly Programs

- For our assembler, you can set the interrupt vector by including the file `hcs12.inc`. In this file, the addresses of all of the 9212 interrupt vectors are defined.
- For example, the pointer to the Timer Overflow Interrupt vector is called `UserTimerOvf`:

```
UserTimerOvf equ $3E5E
```

You can set the interrupt vector to point to the interrupt service routine `toi_isr` with the Assembly statement:

```
movw    #toi_isr,UserTimerOvf
```

- Here is a program where the interrupt vector is set in the program:

```
#include "hcs12.h"
#define prog $1000

    movw    #toi_isr,UserTimerOvf  ; Set interrupt vector
    movb    #$ff,DDRA
    movb    #$80,TSCR1             ; Turn on timer
    movb    #$86,TSCR2             ; Enable timer overflow interrupt, set prescaler
                                        ; so interrupt period is 175 ms
    movb    #$80,TFLG2             ; Clear timer interrupt flag
    cli                                           ; Enable interrupts

l1: wai                                           ; Do nothing - go into low power mode */
    bra    l1

toi_isr:
    inc    PORTA
    movb    #$80,TFLG2             ; Clear timer overflow interrupt flag
    rts
```

## USING INTERRUPTS ON THE HCS12

What happens when the HCS12 receives an unmasked interrupt?

1. Finish current instruction
2. Push all registers onto the stack
3. Set I bit of CCR
4. Load Program Counter from interrupt vector for particular interrupt

Most interrupts have both a specific mask and a general mask. For most interrupts the general mask is the I bit of the CCR. For the TOF interrupt the specific mask is the TOI bit of the TSCR2 register.

Before using interrupts, make sure to:

1. Load stack pointer
  - Done for you in C by the C startup code
2. Write Interrupt Service Routine
  - Do whatever needs to be done to service interrupt
  - Clear interrupt flag
  - Exit with RTI
    - Use the `INTERRUPT` definition in the Gnu C compiler
3. Load address of interrupt service routine into interrupt vector
4. Do any setup needed for interrupt
  - For example, for the TOF interrupt, turn on timer and set prescaler
5. Enable specific interrupt
6. Enable interrupts in general (clear I bit of CCR with `cli` instruction or `enable()` function)

Can disable all (maskable) interrupts with the `sei` instruction or `disable()` function.

An example of the HCS12 registers and stack when a TOF interrupt is received

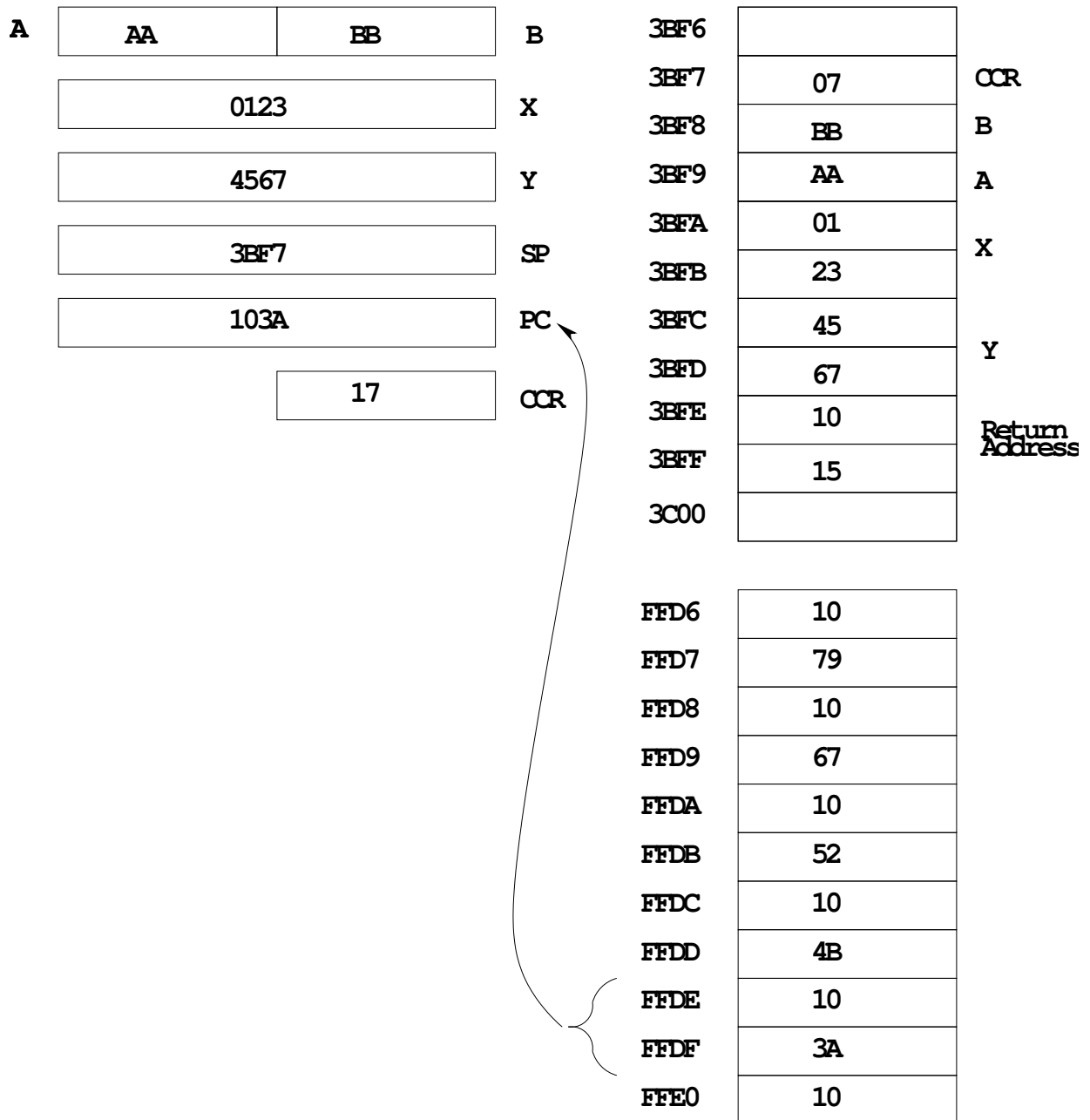
HC12 STATE BEFORE RECEIVING TOF INTERRUPT

<b>A</b>	<b>AA</b>	<b>BB</b>	<b>B</b>	<b>3BF6</b>	
	0123		<b>X</b>	<b>3BF7</b>	
	4567		<b>Y</b>	<b>3BF8</b>	
	3C00		<b>SP</b>	<b>3BF9</b>	
	1015		<b>PC</b>	<b>3BFA</b>	
	07		<b>CCR</b>	<b>3BFB</b>	
				<b>3BFC</b>	
				<b>3BFD</b>	
				<b>3BFE</b>	
				<b>3BFF</b>	
			<b>3C00</b>		
			<b>FFD6</b>	10	
			<b>FFD7</b>	79	
			<b>FFD8</b>	10	
			<b>FFD9</b>	67	
			<b>FFDA</b>	10	
			<b>FFDB</b>	52	
			<b>FFDC</b>	10	
			<b>FFDD</b>	4B	
			<b>FFDE</b>	10	
			<b>FFDF</b>	3A	
			<b>FFE0</b>	10	

### An example of the HCS12 registers and stack just after a TOF interrupt is received

- All of the HCS12 registers are pushed onto the stack, the PC is loaded with the contents of the Interrupt Vector, and the I bit of the CCR is set

#### HC12 STATE AFTER RECEIVING TOF INTERRUPT



## Interrupt vectors for the 68HC912B32

- The interrupt vectors for the MC9S12DP256 are located in memory from 0xFF80 to 0xFFFF.
- These vectors are programmed into Flash EEPROM and are very difficult to change
- DBug12 redirects the interrupts to a region of RAM where they are easy to change
- For example, when the HCS12 gets a TOF interrupt:
  - It loads the PC with the contents of 0xFFDE and 0xFFDF.
  - The program at that address tells the HCS12 to look at address 0x3E5E and 0x3E5F.
  - If there is a 0x0000 at these two addresses, DBug12 gives an error stating that the interrupt vector is uninitialized.
  - If there is anything else at these two addresses, DBug12 loads this data into the PC and executes the routine located there.
  - To use the TOF interrupt you need to put the address of your TOF ISR at addresses 0x3E5E and 0x3E5F.



## Commonly Used Interrupt Vectors for the MC9S12DP256

Interrupt	Specific Mask	General Mask	Normal Vector	DBug-12 Vector
SPI2	SP2CR1 (SPIE, SPTIE)	I	FFBC, FFBD	3E3C, 3E3D
SPI1	SP1CR1 (SPIE, SPTIE)	I	FFBE, FFBF	3E3E, 3E3F
IIC	IBCR (IBIR)	I	FFC0, FFC1	3E40, 3E41
BDLC	DLCBCR (IE)	I	FFC2, FFC3	3E42, 3E43
CRG Self Clock Mode	CRGINT (SCMIE)	I	FFC4, FFC5	3E44, 3E45
CRG Lock	CRGINT (LOCKIE)	I	FFC6, FFC7	3E46, 3E47
Pulse Acc B Overflow	PBCTL (PBOVI)	I	FFC8, FFC9	3E48, 3E49
Mod Down Ctr UnderFlow	MCCTL (MCZI)	I	FFCA, FFCB	3E4A, 3E4B
Port H	PTHIF (PTHIE)	I	FFCC, FFCD	3E4C, 3E4D
Port J	PTJIF (PTJIE)	I	FFCE, FFCF	3E4E, 3E4F
ATD1	ATD1CTL2 (ASCIE)	I	FFD0, FFD1	3E50, 3E51
ATD0	ATDOCTL2 (ASCIE)	I	FFD2, FFD3	3E52, 3E53
SCI1	SC1CR2 (TIE, TCIE, RIE, ILIE)	I	FFD4, FFD5	3E54, 3E55
SCIO	SCOCR2 (TIE, TCIE, RIE, ILIE)	I	FFD6, FFD7	3E56, 3E57
SPIO	SPOCR1 (SPIE)	I	FFD8, FFD9	3E58, 3E59
Pulse Acc A Edge	PACTL (PAI)	I	FFDA, FFDB	3E5A, 3E5B
Pulse Acc A Overflow	PACTL (PAOVI)	I	FFDC, FFDD	3E5C, 3E5D
Enh Capt Timer Overflow	TSCR2 (TOI)	I	FFDE, FFDF	3E5E, 3E5F
Enh Capt Timer Channel 7	TIE (C7I)	I	FFE0, FFE1	3E60, 3E61
Enh Capt Timer Channel 6	TIE (C6I)	I	FFE2, FFE3	3E62, 3E63
Enh Capt Timer Channel 5	TIE (C5I)	I	FFE4, FFE5	3E64, 3E65
Enh Capt Timer Channel 4	TIE (C4I)	I	FFE6, FFE7	3E66, 3E67
Enh Capt Timer Channel 3	TIE (C3I)	I	FFE8, FFE9	3E68, 3E69
Enh Capt Timer Channel 2	TIE (C2I)	I	FFEA, FFEB	3E6A, 3E6B
Enh Capt Timer Channel 1	TIE (C1I)	I	FFEC, FFED	3E6C, 3E6D
Enh Capt Timer Channel 0	TIE (C0I)	I	FFEE, FFEF	3E6E, 3E6F
Real Time	CRGINT (RTIE)	I	FFF0, FFF1	3E70, 3E71
IRQ	IRQCR (IRQEN)	I	FFF2, FFF3	3E72, 3E73
XIRQ	(None)	X	FFFF, FFFF	3E74, 3E75
SWI	(None)	(None)	FFF6, FFF7	3E76, 3E77
Unimplemented Instruction	(None)	(None)	FFF8, FFF9	3E78, 3E79
COP Failure	COPCTL (CR2-CR0 COP Rate Select)	(None)	FFFA, FFFB	3E7A, 3E7B
COP Clock Moniotr Fail	PLLCTL (CME, SCME)	(None)	FFFC, FFFD	3E7C, 3E7D
Reset	(None)	(None)	FFFE, FFFF	3E7E, 3E7F