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 `0xFFFE` 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

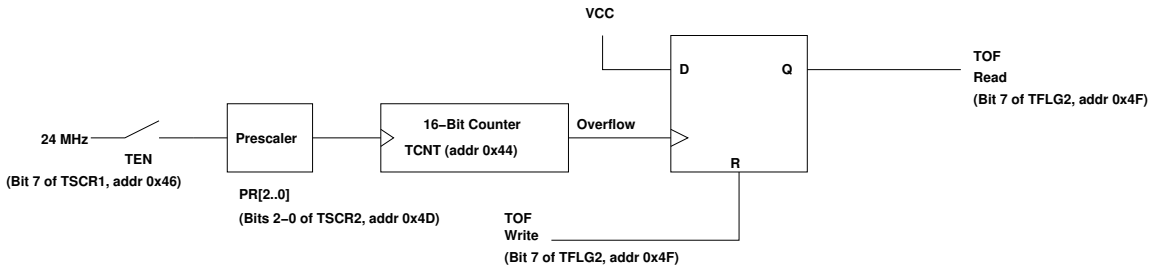
- On reset or power-up, the first instruction your HCS12 will execute is the one located at address `0xF000`.

The 9S12 Timer

- The 9S12 has a 16-bit free-running counter (timer).
- 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
```

TIMER OVERFLOW INTERRUPT



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 μs	2.73 ms
001	2	12 MHz	0.083 μs	5.46 ms
010	4	6 MHz	0.167 μs	10.92 ms
011	8	3 MHz	0.333 μs	21.85 ms
100	16	1.5 MHz	0.667 μs	43.69 ms
101	32	750 kHz	1.333 μs	87.38 ms
110	64	375 kHz	2.667 μs	174.76 ms
111	128	187.5 kHz	5.333 μs	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:

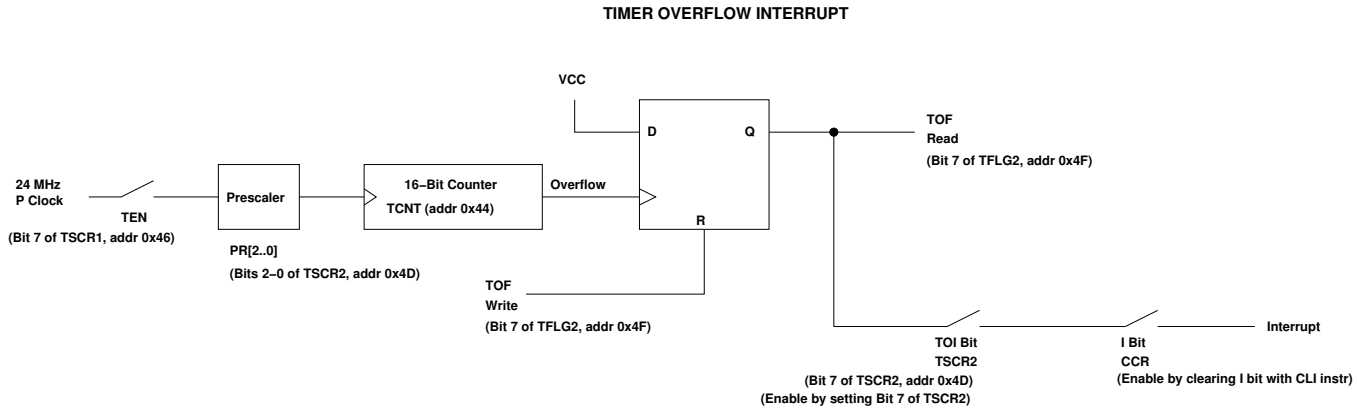
```
TFLG2 = 0x80;
```

- 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.

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 in 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*2)
#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)
    {
        asm(" wai");    /* 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"
prog:  equ   $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
    rti
```

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

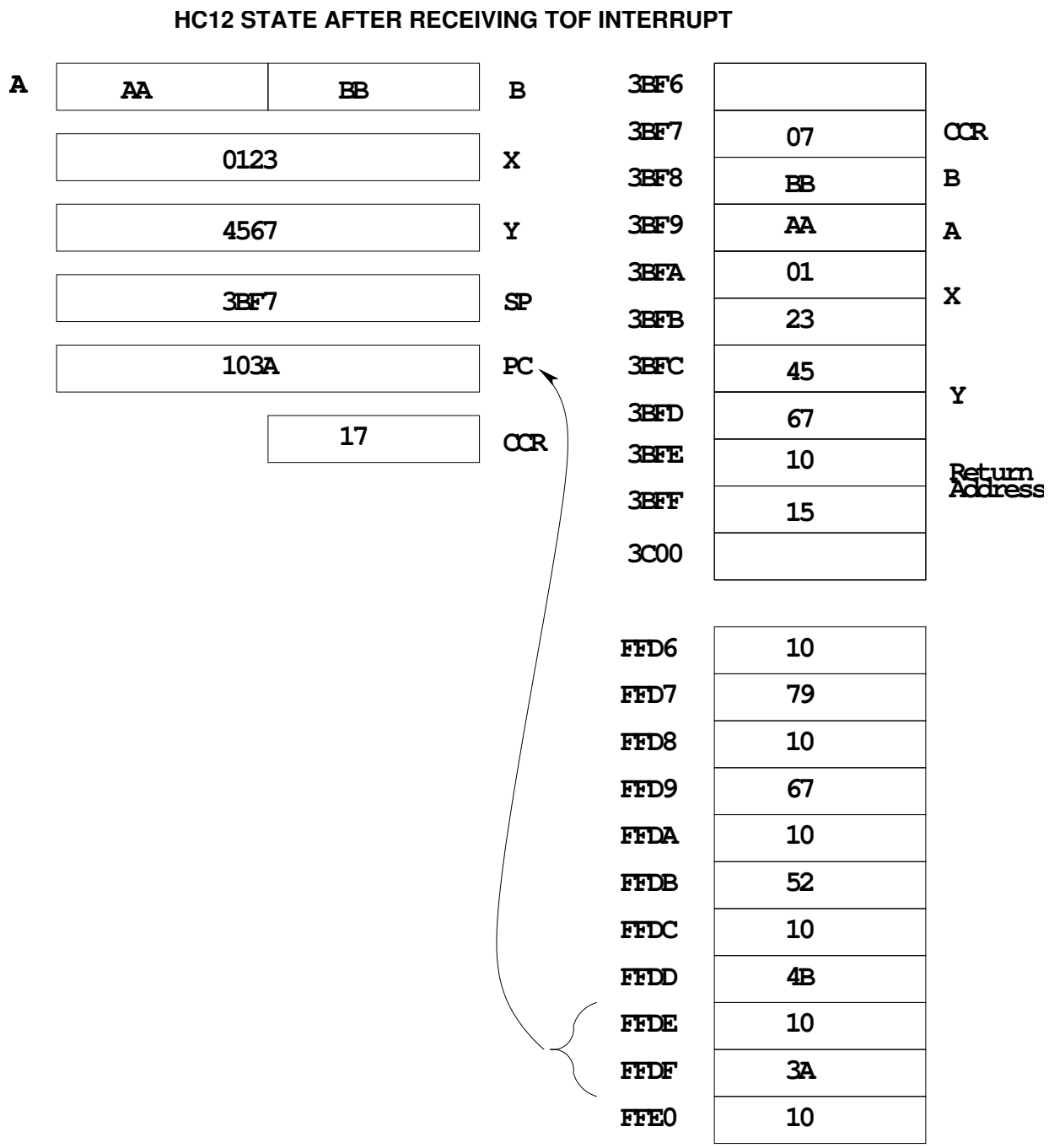
A	AA	BB	B
	0123		X
	4567		Y
	3C00		SP
	1015		PC
		07	CCR

3BF6	
3BF7	
3BF8	
3BF9	
3BFA	
3BFB	
3BFC	
3BFD	
3BFE	
3BFF	
3C00	

FFD6	10
FFD7	79
FFD8	10
FFD9	67
FFDA	10
FFDB	52
FFDC	10
FFDD	4B
FFDE	10
FFDF	3A
FFE0	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



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	ATD0CTL2 (ASCIE)	I	FFD2, FFD3	3E52, 3E53
SCI1	SC1CR2 (TIE, TCIE, RIE, ILIE)	I	FFD4, FFD5	3E54, 3E55
SCI0	SC0CR2 (TIE, TCIE, RIE, ILIE)	I	FFD6, FFD7	3E56, 3E57
SPI0	SP0CR1 (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

EXCEPTIONS ON THE HCS12

- Exceptions are the way a processor responds to things other than the normal sequence of instructions in memory.
- Exceptions consist of such things as Reset and Interrupts.
- Interrupts allow a processor to respond to an event without constantly polling to see whether the event has occurred.
- On the HCS12 some interrupts cannot be masked — these are the Unimplemented Instruction Trap and the Software Interrupt (`SWI` instruction).
- XIRQ interrupt is masked with the X bit of the Condition Code Register. Once the X bit is cleared to enable the XIRQ interrupt, it cannot be set to disable it.
  - The XIRQ interrupt is for external events such as power fail which must be responed to.
- The rest of the HCS12 interrupts are masked with the I bit of the CCR.
  - All these other interrupts are also masked with a specific interrupt mask. For example, the Timer Overflow Interrupt is masked with the TOI bit of the TMSK2 register.
  - This allows you to enable any of these other interrupts you want.
  - The I bit can be set to 1 to disable all of these interrupts if needed.

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 `crts.s`
2. Write Interrupt Service Routine
   - Do whatever needs to be done to service interrupt. Keep it short — do not do things which take a long time, such as a `printf()`, or wait for some external event.
   - Clear interrupt flag
   - Exit with `RTI`
     - Use the `@interrupt` function of the Cosmic 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.