What Happens When You Reset the HC12?

- What happens to the HC12 when you turn on power or push the reset button?
- How does the HC12 know which instruction to execute first?
- On reset the HC12 loads the PC with the address located at address `0xFFFE` and `0xFFFF`.
- Here is what is in the memory of our HC12:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
FFF0	FF	56	FF	5A	FF	5E	FF	62	FF	66	FF	6A	FF	6E	FF	80

- On reset or power-up, the first instruction your HC12 will execute is the one located at address `0xFF80`.
- Here is the code your HC12 initially executes:

```
FF80:     LDS   #$0C00
FF83:     LDAB  $6F
FF85:     ANDB  #$03
FF87:     CMPB  #$03
FF89:     BEQ   $FF8B
FF8C:     BRA   $FF95
FF8F:     CMPB  #$01
```

- The HC12 reads the input from address `$006F`, `AND`s it with `$03` to keep only the two lowest bits, and decides what to do based on those two bits.
- This is how the HC12 tells whether to run the DBug12 monitor program or run directly from EEPROM.

Using the Timer Overflow Flag to implement a delay

- The HC12 timer counts at a rate set by the prescaler:

PR2:0	Divide	Clock Freq	Clock Period	Overflow Period
000	1	8 MHz	0.125 μs	8.192 ms
001	2	4 MHz	0.250 μs	16.384 ms
010	4	2 MHz	0.500 μs	32.768 ms
011	8	1 MHz	1.000 μs	65.536 ms
100	16	500 kHz	2.000 μs	131.072 ms
101	32	250 kHz	4.000 μs	262.144 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 32.768 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 HC12.

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

Interrupt: Allow the HC12 to do other things while waiting for an event to happen. When the event happens, tell HC12 to take care of event, then go back to what it was doing.

What happens when HC12 gets an interrupt: HC12 automatically jumps to part of the program which tells it what to do when it receives the interrupt (Interrupt Service Routine).

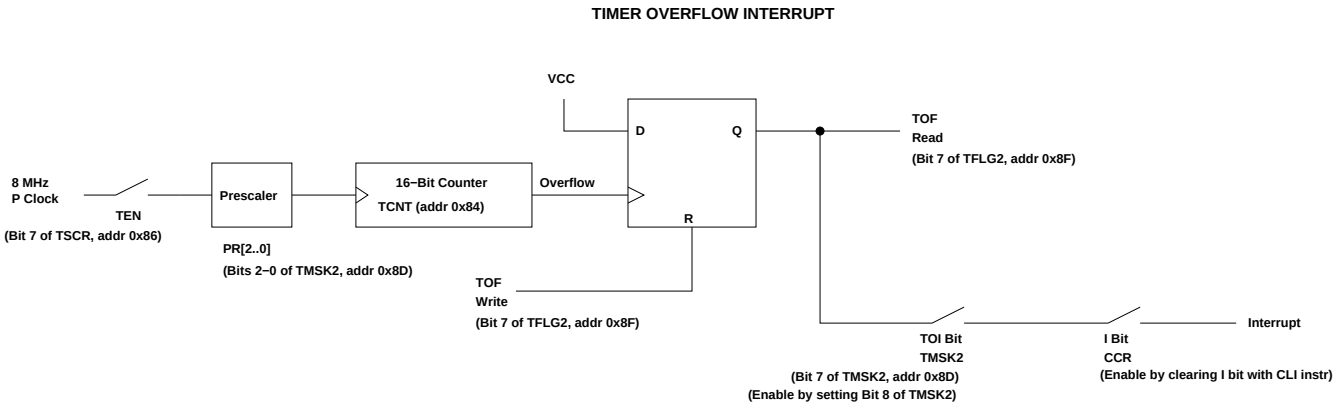
How does HC12 know where the ISR is located: A set of memory locations called Interrupt Vectors tell the HC12 the address of the ISR for each type of interrupt.

How does HC12 know where to return to: Return address pushed onto stack before HC12 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 HC12 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 TSCR)
Set prescaler         (Bits 2:0 of TMSK2)
Enable TOF interrupt  (set Bit 7 of TMSK2)
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 "hc12b32.h"

main()
{
    DDRA = 0xff;         /* Make Port A output */
    TSCR = 0x80;         /* Turn on timer */
    TMSK2 = 0x84;        /* Enable timer overflow interrupt, set prescaler */
    TFLG2 = 0x80;        /* Clear timer interrupt flag */
    enable();            /* Enable interrupts (clear I bit) */
    while (1)
    {
                         /* Do nothing  */
    }
}

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

How to tell the HC12 where the Interrupt Service Routine is located

- You need to tell the HC12 where to go when it receives a TOF interrupt
- You do this by setting the TOF Interrupt Vector
- On your HC12 this is located at address `0x0B1E`
- In C, you include a file which has a holding place for each possible interrupt
- You put the name of the ISRs you need to use in the appropriate slot

```
/*      INTERRUPT VECTORS TABLE 68HC12
 */
void toi_isr();         /* character receive handler */

/* Vectors at 0xFFD0 on standard HC12; remapped to 0x0B10 with D-Bug 12 */
void (* const _vectab[])() = {
        0,              /* BDLC               */
        0,              /* ATD                */
        0,              /* reserved           */
        0,              /* SCI0               */
        0,              /* SPI                */
        0,              /* Pulse acc input    */
        0,              /* Pulse acc overflow */
        toi_isr,        /* Timer overflow     */
        0,              /* Timer channel 7    */
        0,              /* Timer channel 6    */
        0,              /* Timer channel 5    */
        0,              /* Timer channel 4    */
        0,              /* Timer channel 3    */
        0,              /* Timer channel 2    */
        0,              /* Timer channel 1    */
        0,              /* Timer channel 0    */
        0,              /* Real time          */
        0,              /* IRQ                */
        0,              /* XIRQ               */
        0,              /* SWI                */
        0,              /* illegal            */
        0,              /* cop fail           */
        0,              /* cop clock fail     */
        (void *)0xff80, /* RESET              */
        };
```

USING INTERRUPTS ON THE HC12

What happens when the HC12 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 TMSK2 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
   - 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.

An example of the HC12 registers and stack when a TOF interrupt is received

HC12 STATE BEFORE RECEIVING TOF INTERRUPT

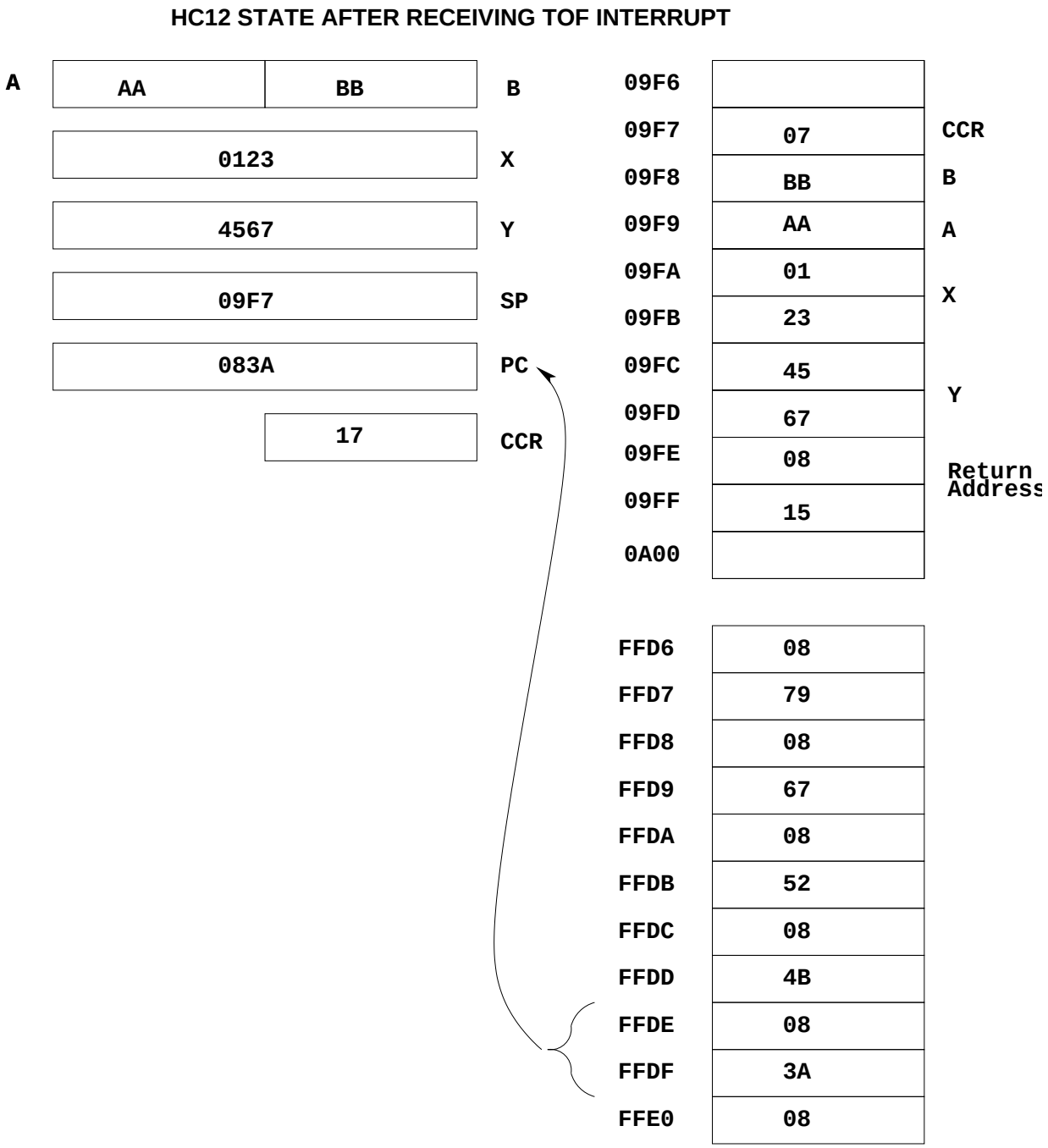
A	AA	BB	B
	0123		X
	4567		Y
	0A00		SP
	0815		PC
		07	CCR

09F6	
09F7	
09F8	
09F9	
09FA	
09FB	
09FC	
09FD	
09FE	
09FF	
0A00	

FFD6	08
FFD7	79
FFD8	08
FFD9	67
FFDA	08
FFDB	52
FFDC	08
FFDD	4B
FFDE	08
FFDF	3A
FFE0	08

An example of the HC12 registers and stack just after a TOF interrupt is received

- All of the HC12 registers are pushed onto the stack, and the PC is loaded with the contents of the Interrupt Vector



Interrupt vectors for the 68HC912B32

- The interrupt vectors for the 68HC912B32 are located in memory from `0xFFD0` 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 HC12 gets a TOF interrupt:
  - It loads the PC with the contents of `0xFFDE` and `0xFFDF`.
  - The program at that address tells the HC12 to look at address `0x0B1E` and `0x0B1F`.
  - 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 `0x0B1E` and `0x0B1F`.

Interrupt	Specific Mask	General Mask	Normal Vector	D-Bug 12 Vector
BDLC	BCR1 (IE)	I	FFD0, FFD1	0B10, 0B11
ATD	ATDCTL2 (ASCIE)	I	FFD2, FFD3	0B12, 0B13
Reserved		I	FFD4, FFD5	0B14, 0B15
SCI	SC0CR2 (TIE, TCIE, RIE, ILIE)	I	FFD6, FFD7	0B16, 0B17
SPI	SP0CR1 (SPIE)	I	FFD8, FFD9	0B18, 0B19
Pulse Acc Edge	PACTL (PAI)	I	FFDA, FFDB	0B1A, 0B1B
Pulse Acc Overflow	PACTL (PAOVI)	I	FFDC, FFDD	0B1C, 0B1D
Timer Overflow	TMSK2 (TOI)	I	FFDE, FFDF	0B1E, 0B1F
Timer Channel 7	TMSK1 (C7I)	I	FFE0, FFE1	0B20, 0B21
Timer Channel 6	TMSK1 (C6I)	I	FFE2, FFE3	0B22, 0B23
Timer Channel 5	TMSK1 (C5I)	I	FFE4, FFE5	0B24, 0B25
Timer Channel 4	TMSK1 (C4I)	I	FFE6, FFE7	0B26, 0B27
Timer Channel 3	TMSK1 (C3I)	I	FFE8, FFE9	0B28, 0B29
Timer Channel 2	TMSK1 (C2I)	I	FFEA, FFEB	0B2A, 0B2B
Timer Channel 1	TMSK1 (C1I)	I	FFEC, FFED	0B2C, 0B2D
Timer Channel 0	TMSK1 (C0I)	I	FFEE, FFEF	0B2E, 0B2F
Real Time	RTICTL (RTIE)	I	FFF0, FFF1	0B30, 0B31
IRQ	INTCR (IRQEN)	I	FFF2, FFF3	0B32, 0B33
XIRQ	(None)	X	FFFF, FFFF	0B34, 0B35
SWI	(None)	(None)	FFF6, FFF7	0B36, 0B37
Unimplemented Instruction	(None)	(None)	FFF8, FFF9	0B38, 0B39
COP failure	COP rate select	(None)	FFFA, FFFB	0B3A, 0B3B
COP clock moniotr fail	COPCTL (CME, FCME)	(None)	FFFC, FFFD	0B3C, 0B3D
Reset	(None)	(None)	FFFE, FFFF	0B3E, 0B3F