

## INTERRUPTS

The 68HC11 has many separate interrupt vectors. In order not to compromise their user availability (since the ROM inside the chip can not be user modifiable) these vectors were assigned addresses outside the ROM. The addresses chosen for these vectors to point at were the high locations of the EEPROM from \$B7BF to \$B7F8. This collection of 3 byte locations will typically consist of a jump table. At each location a user machine coded jump instruction will direct the interrupt on to the location of its own machine code handler. Of course, if EEPROM has been disabled, external memory can provide this second redirection.

To use a particular interrupt, a machine code handler must first be written. Normally it will end with an RTI to return to the interrupted program. The JMP op-code and the address of the code definition's parameter field should be installed in the jump table. The particular interrupt can then be enabled.

```
CODE IRQRTN ( HANDLE EXT PIN IRQ'S )
```

```
( ... )
```

```
( ... )
```

```
( ... )
```

```
3B C, ( RTI )
```

```
END-CODE
```

```
CODE-SUB CLI
```

```
0E C, ( CLI )
```

```
39 C, ( RTS )
```

```
END-CODE
```

```
: ENABLE-IRQ'S
```

```
7E B7E9 EEC! ( JMP OP-CODE )
```

```
[ ' IRQRTN @ >< FF AND ] LITERAL B7EA EEC! ( HI BYTE )
```

```
[ ' IRQRTN @ FF AND ] LITERAL B7EB EEC! ( LO BYTE )
```

```
CLI
```

```
;
```

In this example, ENABLE-IRQ's puts the JMP op-code in the address (\$B7E9) pointed to by the External IRQ Pin Vector (\$FFF2). It gets the address of the interrupt routine that will handle the IRQ Pin's requests, IRQRTN, and puts it in after the JMP op-code. This is accomplished one byte at a time as follows. First the compilation mode is left. The interrupt routine's name is ticked. This returns its PFAPTR. The PFAPTR is converted to a PFA by the @. The appropriate high byte or low byte of the result is masked out. The compilation mode is then reentered. The value on the stack is compiled as a literal to be programmed into EEPROM at run time. A routine is then called that clears the Interrupt Disable bit in the Condition Codes Register.

The EEPROM has a limited life cycle, usually rated at 10,000 writes. Therefore, a better method of initializing the vectors would only write them if they needed to be changed. This will be shown in the next example.

High level interrupts can easily be handled in Max-FORTH. Low level interrupt handlers can be written that call high level interrupts with only 3 or 4 lines of machine code. This is possible because of the inclusion of a subroutine called ATO4 which can start a high level word from machine code.

To run a high level word, all that is necessary is to first load the D Register with the Code Field Address of the word to be run, for versions earlier than V3.5, load the Y Register with a pointer to free memory area that can be used as data stack and to then jump subroutine to ATO4 . This routine catches the system after the completion of the called high level word and does an RTS to return to the machine code that called it. Of course, the high level word will probably use the Data and Return Stacks, but since Max-FORTH has been written to be totally preemptable, this should pose no problem, other than the additional depth needed during interrupts. The SP register should not need adjustment. (In many instances, the Y Register would not need to be modified either. Only the FIND routine in Max-FORTH versions earlier than V3.5 uses the Y Register for anything except a very carefully maintained stack pointer. If the foreground program is not using FIND or in other words the outer interpreter, but is instead a dedicated program itself, and the user has not entered any machine coded definitions that tamper with Y's purpose as a data stack pointer, no concern need be paid to Y's contents during interrupt.)

There is a built-in constant that returns the address of this special subroutine. To acquire this address simply enter ATO4 .

```
: HI-LEVEL-IRQRTN ( HANDLE EXT PIN IRQ'S )  
  (... )  
  (... )  
  (... )  
;
```

```
CODE LO-LEVEL-IRQRTN  
  CC C, ' HI-LEVEL-IRQRTN CFA , ( LDD # CFA-OF-HI-LEVEL-IRQRTN )  
  ( FOLLOWING LINE MAY BE NECESSARY FOR VERSIONS PRIOR TO V3.5  
  ( 18 C, CE C, XXXX , ( LDY # XXXX, WHERE XXXX = FREE D.S. AREA )  
  BD C, ATO4 , ( JSR ATO4 )  
  3B C, ( RTI )  
END-CODE
```

```
CODE-SUB CLI  
  0E C, ( CLI )  
  39 C, ( RTS )  
END-CODE
```

CODE-SUB SEI  
0F C, ( SEI )  
39 C, ( RTS )  
END-CODE

VARIABLE VEC-TABLE -2 ALLOT

7E C, FFFE @ , ( B7BF SCI SER SYS )  
7E C, FFFE @ , ( B7C2 SPI SER )  
7E C, FFFE @ , ( B7C5 PLS ACC OVFL )  
7E C, FFFE @ , ( B7C8 PLS ACC EDGE )  
7E C, FFFE @ , ( B7CB TMR OVERFLOW )  
7E C, FFFE @ , ( B7CE TMR OUT CMP 5 )  
7E C, FFFE @ , ( B7D1 TMR OUT CMP 4 )  
7E C, FFFE @ , ( B7D4 TMR OUT CMP 3 )  
7E C, FFFE @ , ( B7D7 TMR OUT CMP 2 )  
7E C, FFFE @ , ( B7DA TMR OUT CMP 1 )  
7E C, FFFE @ , ( B7DD TMR IN CAP 3 )  
7E C, FFFE @ , ( B7E0 TMR IN CAP 2 )  
7E C, FFFE @ , ( B7E3 TMR IN CAP 1 )  
7E C, FFFE @ , ( B7E6 REAL TIME )  
7E C, ' LO-LEVEL-IRQRTN @ , ( B7E9 IRQ )  
7E C, FFFE @ , ( B7EC XIRQ )  
7E C, FFFE @ , ( B7EF SWI )  
7E C, FFFE @ , ( B7F2 OP-CODE TRAP )  
7E C, FFFE @ , ( B7F5 COP FAILURE )  
7E C, FFFE @ , ( B7F8 CLK MON )

HERE CONSTANT VEC-TABLE-END

: VEC-INIT  
( CHECK AND MOVE VECTORS IF NECESSARY )  
VEC-TABLE-END VEC-TABLE - 0 ( RANGE )  
DO  
  B7BF I + C@ VEC-TABLE I + C@ = NOT  
  IF VEC-TABLE I + C@ B7BF I + ." ." EEC! THEN  
  LOOP  
;

: ENABLE-IRQ'S  
VEC-INIT  
CLI  
;