Interrupts
Resets
Low Power Modes

Drop everything and get your priorities straight!

Alan Claghorn
Chris Golder
Raja Shah

Outline

- Interrupts
  - Why use interrupts?
  - Types of interrupts
  - Interrupt Flow
  - Priorities
- Resets
- Low Power Modes
  - Wait
  - Stop
- Real World Applications
History

- Univac 1103/1103A (1953/1956) - Typically cited as first CPU with interrupts. Current instruction stored in memory location and program counter loaded with a fixed address.

But earlier examples include:
- UNIVAC (1951) – first exception handling
- IBM (1954) – first to use interrupt masking
- NBS DYSEAC (1954) – first with I/O interrupts

Introduction

Interrupts – What are they good for?
- To modify or do additional instructions based on internal requests and/or external devices
- Provide a means for suspending current instructions for more important tasks
- Hardware interrupts (e.g. power on reset, keyboard, and printer)
- Software interrupts (e.g. timer resets, timer interrupts, and traps)
Methods of informing CPU

- Polling – constantly checks external devices for data by iteration
  - similar to picking up a phone when waiting for a call because the ringer doesn’t work
  - Why is this so bad...?
    - Interrupts – method of performing standard operations until the computer is informed to do something else

Interrupts

- Communication between CPU and I/O devices is established with the issue of an interrupt request
  - NOTE: Request can be issued at any time

- Suspends normal execution and completes instruction in the Interrupt Service Routine (ISR)

- Returns to normal program as if no change has been made
Types of Interrupts

15 Maskable interrupts:
1. IRQ
2. Real-Time Interrupt
3. Timer Input Capture1 (TIC1)
4. TIC2
5. TIC3
6. Timer Output Compare 1 (TOC1)
7. TOC2
8. TOC3
9. TOC4
10. TIC 4/OC 5
11. Timer Overflow
12. Pulse Accumulator Overflow
13. Pulse Accumulator Input Edge
14. SPI transfer Complete
15. SCI system

6 non-maskable interrupts:
1. POR of RESET pin
2. Clock monitor reset
3. COP watchdog reset
4. XIRQ interrupt
5. Illegal opcode interrupt
6. Software interrupt (SWI)

Types of Interrupts

The 15 Maskable Interrupts:
- Two types of Masking
  - Local - Interrupt enable bit
  - Global - 1-bit in CCR
- Follows a default priority arrangement. Any one interrupt can be promoted for higher priority using HPRIO (see previous slide)
- Sets I bit in CCR when serviced

The 6 Non-maskable Interrupts:
- Follows the default priorities
- Not subject to global masking
- Set I and X bit in CCR when serviced
Maskable Interrupts: IRQ Input

- IRQ pin provides additional external interrupting source
- Other additional MCU pins can be used as interrupt inputs
  - Example:
    - XIRQ input
    - Main-Timer Capture Pins
    - Pulse Accumulator Pin
- IRQE Bit in the OPTION control used to specify IRQ pin configuration
  - IRQE = 0 Low level Sensitive
  - IRQE = 1 low-going edge sensitive (single source only)

Maskable Interrupts: Peripheral Subsystems

- Interrupts from Internal Peripheral Subsystems
  - Global masking using I-bit in CCR
  - Flag bit, which requests service
  - Interrupt enable bit, which enables flag to generate interrupt service
  - Programmer determines whether to use polling/interrupts for each source
Maskable Interrupt: I-Bit

- Global Masking by the I-Bit
- Bit 4 in the CCR
- When I-bit is 0, allows interrupt servicing when called
- When I-bit is 1, inhibits interrupt service (pending)
- Set by reset to allow minimum system initialization
- Set upon entry in interrupt service routine (ISR)
- Can be set by software to inhibit further maskable interrupts
  - SEI (Set Interrupt Mask)
- Automatically cleared by the RTI instruction
- Can be cleared by software
  - CLI (Clear Interrupt Mask)

Non-Maskable Interrupts: Illegal Opcode Trap

- Generates interrupt request to the illegal opcode vector
- Reinitialize stack pointer once interrupt service is completed
- Left uninitialized, illegal opcode vector can cause an infinite loop causing stack underflow
Non-Maskable Interrupt: Software Interrupt

- Software instruction, thus cannot be interrupted until completed
- Uninhibited by global mask bits in the CCR
- Similar to other interrupts, sets the I-bit upon servicing

Non-Maskable Interrupts: XIRQ

- Enabled by TAP instruction by clearing X-bit upon system initialization
- After being cleared, software cannot set the X-bit, thus XIRQ is non-maskable
- Higher priority than any source maskable by the I-bit
- Both the X and I bits are automatically set by hardware after stacking the CCR
- RTI restores X and I bit to pre-interrupt states
Interrupt Flow

Interrupt condition is met

Global Masking

no

Local Masking

yes

Check flag if cleared

no

Complete current instruction

yes

Store all registers on the stack

Continue Program

A

Analyze Priority

Set the (I) or (X) To prohibit another Interrupt

Standard interrupt table

Load address in Program counter

ISR instruction

RTI

B

Clear I or X bit in CCR

Restore registers w/ org. values

Note: Flag must be cleared prior to performing RTI

Interrupt Stack Pointer

- The stack pointer register holds the location of the top of the stack at all times.
- When the CPU detects an interrupt, the contents of the registers are pushed on the stack.
- After completion of the interrupt the saved registers are retrieved form the stack in a first in last out order.
## Stacking Order

<table>
<thead>
<tr>
<th>Memory Location</th>
<th>CPU Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
<td>PCL</td>
</tr>
<tr>
<td>SP-1</td>
<td>PCH</td>
</tr>
<tr>
<td>SP-2</td>
<td>IYL</td>
</tr>
<tr>
<td>SP-3</td>
<td>IYH</td>
</tr>
<tr>
<td>SP-4</td>
<td>IXL</td>
</tr>
<tr>
<td>SP-5</td>
<td>IXH</td>
</tr>
<tr>
<td>SP-6</td>
<td>ACCA</td>
</tr>
<tr>
<td>SP-7</td>
<td>ACCB</td>
</tr>
<tr>
<td>SP-8</td>
<td>CRR</td>
</tr>
</tbody>
</table>

First Pushed In Last Pulled Out
Last Pushed In First Pulled Out

## Interrupt Types

- **Non-Maskable Interrupts**
  - 6 Non-Maskable Interrupts.
  - Always interrupts program execution.
  - Has priority over maskable interrupts.

- **Maskable Interrupts**
  - 15 Maskable Interrupts.
  - Can be disabled by setting I bit of the CCR
  - The Priority Level of Maskable Interrupts can be changed.
## Non-Maskable Interrupts

### Priority Level

1. RESET
2. Clock Monitor
3. COP Watchdog
4. Illegal Opcode
5. XIRQ
6. SWI

## Maskable Interrupts

### (Default Priority)

7. IRQ
8. Periodic Interrupt (real Time Interrupt)
9. Timer Input Capture 1
10. Timer Input Capture 2
11. Timer Input Capture 3
12. Timer Output compare 1
13. Timer Output compare 2
14. Timer Output compare 3
15. Timer Output compare 4
16. Timer Output compare 5
17. Timer Overflow
18. Pulse Accumulator Overflow
19. Pulse Accumulator Input Edge
20. SPI Transfer Complete
21. SCI Serial System
Maskable Interrupts HPRIO

- Located at $103C
- Can be changed at any time during a program as long as I bit is set.
- Bits 0-3 are used to set high priority to one of the maskable interrupts.
- Default is IRQ

HPRIO $103C

<table>
<thead>
<tr>
<th>PSEL3</th>
<th>PSEL2</th>
<th>PSEL1</th>
<th>PSEL0</th>
<th>Interrupt Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Timer Overflow</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Pulse Accum. Overflow</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Pulse Accum Input Edge</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>SPI serial xfer complete</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>SCI serial system</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Reserved (default to IRQ)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>IRQ</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Real Time Interrupt</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Timer Input Capture 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Timer Input Capture 2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Timer Input Capture 3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Timer Output Compare 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Timer Output Compare 2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Timer Output Compare 3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Timer Output Compare 4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Timer Output Compare 5</td>
</tr>
</tbody>
</table>
Interrupts

- When a Non-Maskable interrupts come in:
  - X bit is set to 1, blocking further non-maskable interrupts.
  - I bit is set to 1, blocking maskable interrupts.

- When a Maskable Interrupt comes in:
  - I bit is set to 1, blocking further maskable interrupts.

Interrupt Procedures

When executing instructions and a non-maskable interrupt comes in:
1. Finishes current instruction.
2. Set the X and I bit.
3. Executes the interrupt.
4. Return to instructions.
Interrupt Procedures

A maskable interrupt is currently running and a non-maskable interrupt comes in:

1. Finishes current instruction of the maskable interrupt.
2. Sets the X bit to 1, I bit is already set due to the maskable interrupt.
3. Executes the non-maskable interrupt.
4. Executes the next interrupt of highest priority.

Interrupt Vectors

- When an interrupt takes place, the CPU is directed by the interrupt vector.
- The address for each interrupt source depends on the condition of ROMON.
- Each vector is itself an address when ROMON is enabled.
Interrupt Vectors

- Each interrupt has its own vector, If:
- ROMON Disabled
  - Vector addresses are not occupied by Buffalo.
  - The starting line address of the vector must be programmed into Buffalo.
- ROMON Enabled
  - The jump command ($7E) to the first byte of the vector address must be in the program.
  - In the remaining two bytes the starting location of the ISR is written.

### Interrupt Vector Table: ROMON Disabled

<table>
<thead>
<tr>
<th>Vector Address</th>
<th>Interrupt Source</th>
<th>CCR Mask Bit</th>
<th>Local Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFC0, C1 – FFDA, D5</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FFD6, D7</td>
<td>SCI receive data register full</td>
<td>RIE</td>
<td>RIE</td>
</tr>
<tr>
<td></td>
<td>SCI receiver overrun</td>
<td>RIE</td>
<td>RIE</td>
</tr>
<tr>
<td></td>
<td>SCI transmit data register empty</td>
<td>TIE</td>
<td>TIE</td>
</tr>
<tr>
<td></td>
<td>SCI transmit complete</td>
<td>TCIE</td>
<td>TCIE</td>
</tr>
<tr>
<td></td>
<td>SCI idle line detect</td>
<td>ILIE</td>
<td>ILIE</td>
</tr>
<tr>
<td>FFD8, D9</td>
<td>SPI serial transfer complete</td>
<td>SPIE</td>
<td>SPIE</td>
</tr>
<tr>
<td>FFDA, DB</td>
<td>Pulse accumulator input edge</td>
<td>PAIH</td>
<td>PAIH</td>
</tr>
<tr>
<td>FFDC, DD</td>
<td>Pulse accumulator overflow</td>
<td>PAOV1</td>
<td>PAOV1</td>
</tr>
<tr>
<td>FFDE, DF</td>
<td>Timer overflow</td>
<td>TOI</td>
<td>TOI</td>
</tr>
<tr>
<td>FFED, E1</td>
<td>Timer IC4/OC5</td>
<td>IO3I</td>
<td>IO3I</td>
</tr>
<tr>
<td>FFED, E2</td>
<td>Timer output compare 4</td>
<td>OC4I</td>
<td>OC4I</td>
</tr>
<tr>
<td>FFEE, E3</td>
<td>Timer output compare 3</td>
<td>OC3I</td>
<td>OC3I</td>
</tr>
<tr>
<td>FFEE, E4</td>
<td>Timer output compare 2</td>
<td>OC2I</td>
<td>OC2I</td>
</tr>
<tr>
<td>FFEE, E5</td>
<td>Timer output compare 1</td>
<td>OC1I</td>
<td>OC1I</td>
</tr>
<tr>
<td>FFEE, E6</td>
<td>Timer input capture 5</td>
<td>IC5I</td>
<td>IC5I</td>
</tr>
<tr>
<td>FFEE, E7</td>
<td>Timer input capture 4</td>
<td>IC4I</td>
<td>IC4I</td>
</tr>
<tr>
<td>FFED, E8</td>
<td>Timer input capture 3</td>
<td>IC3I</td>
<td>IC3I</td>
</tr>
<tr>
<td>FFED, E9</td>
<td>Timer input capture 2</td>
<td>IC2I</td>
<td>IC2I</td>
</tr>
<tr>
<td>FFEE, EF</td>
<td>Timer input capture 1</td>
<td>IC1I</td>
<td>IC1I</td>
</tr>
<tr>
<td>FFFE, F0</td>
<td>Real-time interrupt</td>
<td>RTII</td>
<td>RTII</td>
</tr>
<tr>
<td>FFFE, F1</td>
<td>IRQ (external pin)</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>FFFE, F2</td>
<td>XIRQ pin</td>
<td>X</td>
<td>None</td>
</tr>
<tr>
<td>FFFE, F3</td>
<td>Software interrupt</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>FFFE, F4</td>
<td>Illegal opcode trap</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>FFFE, F5</td>
<td>COP failure</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>FFFE, F6</td>
<td>Clock monitor fail</td>
<td>None</td>
<td>CME</td>
</tr>
<tr>
<td>FFFE, F7</td>
<td>RESET</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
### Interrupt Vector Table: ROMON Enabled

<table>
<thead>
<tr>
<th>Vector Address</th>
<th>Interrupt Source</th>
<th>CCR Mask Bit</th>
<th>Local Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>00C4-00C6</td>
<td>SCI receive data register full</td>
<td></td>
<td>RIE</td>
</tr>
<tr>
<td></td>
<td>SCI receiver overrun</td>
<td></td>
<td>RIE</td>
</tr>
<tr>
<td></td>
<td>SCI transmit data register empty</td>
<td>1</td>
<td>TIE</td>
</tr>
<tr>
<td></td>
<td>SCI idle line detect</td>
<td></td>
<td>TCIE</td>
</tr>
<tr>
<td>00C7-00C9</td>
<td>SPI serial transfer complete</td>
<td>I</td>
<td>SPIE</td>
</tr>
<tr>
<td>00CA-00CC</td>
<td>Pulse accumulator input edge</td>
<td>1</td>
<td>PAI</td>
</tr>
<tr>
<td>00CD-00CF</td>
<td>Pulse accumulator overflow</td>
<td>1</td>
<td>PAOV1</td>
</tr>
<tr>
<td>00D0-00D1</td>
<td>Timer overflow</td>
<td>I</td>
<td>TOI</td>
</tr>
<tr>
<td>00D3-00D5</td>
<td>Timer IC4/OC5</td>
<td>I</td>
<td>I4/O5I</td>
</tr>
<tr>
<td>00D6-00D8</td>
<td>Timer output compare 4</td>
<td>1</td>
<td>OC4I</td>
</tr>
<tr>
<td>00D9-00DB</td>
<td>Timer output compare 3</td>
<td>1</td>
<td>OC3I</td>
</tr>
<tr>
<td>00DC-00DE</td>
<td>Timer output compare 2</td>
<td>I</td>
<td>OC2I</td>
</tr>
<tr>
<td>00DF-00E1</td>
<td>Timer output compare 1</td>
<td>I</td>
<td>OC1I</td>
</tr>
<tr>
<td>00E2-00E4</td>
<td>Timer input capture 3</td>
<td>I</td>
<td>I3C</td>
</tr>
<tr>
<td>00E5-00E7</td>
<td>Timer input capture 2</td>
<td>I</td>
<td>I2C</td>
</tr>
<tr>
<td>00E8-00ED</td>
<td>Timer input capture 1</td>
<td>I</td>
<td>I1C</td>
</tr>
<tr>
<td>00EB-00ED</td>
<td>Real-time interrupt</td>
<td>1</td>
<td>RTII</td>
</tr>
<tr>
<td>00EE-00EF</td>
<td>IRQ (external pin)</td>
<td>I</td>
<td>None</td>
</tr>
<tr>
<td>00F1-00F3</td>
<td>XIRQ pin</td>
<td>X</td>
<td>None</td>
</tr>
<tr>
<td>00FA-00FF</td>
<td>Software interrupt</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>00FF-00F9</td>
<td>Illegal opcode trap</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>00FA-00FC</td>
<td>COP failure</td>
<td>None</td>
<td>NOCOP</td>
</tr>
<tr>
<td>00FB-00FF</td>
<td>Clock monitor fail</td>
<td>None</td>
<td>CME</td>
</tr>
</tbody>
</table>

---

### Resets

- **Why do we need them?**
  - Lost track of what bits are set where
- **Compare and Contrast to interrupts**
- **Start with a clean slate**
  - **Initial Conditions**
- **Sources of Resets**
- **Internal vs. External**
Resets – What do They Do?

- Forces HC11 to assume set of initial conditions
- Orderly software start-up from predetermined starting address
- Like interrupts, they share concept of vector fetching to force a new starting point for further CPU operations
- Unlike interrupts they don’t save any states

System Initial Conditions

**CPU**

- Upon reset, CPU fetches restart vector from $FFFE, $FFFF during the first 3 cycles and begins executing instructions
- Stack pointer and other CPU registers are indeterminate immediately after reset
- X and I interrupt mask bits in the CCR are set to mask any interrupt requests
  - $X = 1$  $I = 1$
- S bit is also set in CCR to disable stop mode
System Initial Conditions

Memory Map

- INIT register initialized to $01
  - Puts 256 bytes of RAM at $0000-$00FF
  - Puts control registers at $1000-$103F

Parallel Input/Output (I/O)

- Strobe A Flag (STAF), strobe A interrupt (STAI), and handshake (HNDS) control bits in parallel I/O control register are cleared
  - Prevents interrupt from being enabled
- All other A – E registers set back to block diagram in reference manual

System Initial Conditions

Timer

- Initialized to count of $0000
- All output compare (OC) registers are initialized to $FFFF
  - Use OC to program an action to occur at a specific time (when counter matches OC register, task is executed)
- Input capture registers are indeterminate
  - IC records the time that an external event takes
Effects of Resets

- Real Time Interrupt flag is cleared to allow for other RTI’s to follow
  - This interrupt exists to establish a heartbeat you can use to watch time pass
- Memory Map
- Pulse Accumulator
  - PAI pin defaults to a general purpose input
- COP Watchdog (more detail later)
  - Enabled if NOCOP control bit in CONFIG register is clear (continuously programmer’s responsibility)

Reset changes even more

- Serial Communications Interface (SCI)
  - Baud rate must be reestablished
  - Transmitter and receiver are disabled
- Serial Peripheral Interface (SPI)
  - Disabled by resetting
- Analog-to-Digital Converter (A/D)
  - Conversion complete flag is cleared by reset
  - ADPU bit is cleared, disabling A/D system
- Mode of operation
  - Determined by bits set in the HPRIO register
  - Can be found on page 167 of reference manual
Remember

- Don’t assume what is set where, check all necessary register control bits, so less errors are encountered.

Sources of Resets

- External RESET pin
- Power-on reset
- Computer Operating Properly (COP) watchdog timer reset
- Clock monitor reset
Reset pin

- When reset condition sensed, pin is driven low for 4 E-clock cycles, then released
- Pin is sampled 2 E-clock cycles later
  - If still low: system assumes external reset has occurred
  - Else: reset was initiated by COP watchdog or clock monitor

RESET pin usage

- Should be held low (by external circuit) while $V_{DD}$ is below minimum operating level to protect EEPROM from corruption
- This minimum level should exceed 4.6 V to prevent accidental overwriting of EEPROM
Power-on Reset (POR)

- Used only for power-up conditions to initialize MCU internal circuits
- Applying $V_{DD}$ to the MCU triggers the POR circuit, initiates a reset sequence, and starts an internal timing circuit
- A 4064 clock cycle delay after the oscillator becomes active, allows the clock generator to stabilize

COP Watchdog Timer Reset

- The COP is a constantly decreasing timer that is never supposed to reach zero
- Reset occurs when watchdog times out (counter gets to zero)
- Intended to detect software processing errors
- Programmer must periodically restart the watchdog timer
Servicing the COP Timer

1. Write $55 to COPRST register to arm the clearing mechanism
2. Write $AA to the COPRST register
- Any number of instructions can be performed between the above 2 steps
- Must be performed in the correct sequence before the timer times out
- Rates are set by bits CR1 and CR0 in COPRST register

Clock Monitor Reset (CMR)

- Detects a slow or stopped E clock
- An E-clock frequency below 10 kHz is detected as a clock monitor error
- Enabled by setting the CME bit in OPTION
- Useful as a backup for COP watchdog because CMR requires no clock while COP does
- CMR also provides additional level of protection by generating a system reset if the MCU clocks are accidentally stopped
Process Flow out of Resets When Triggered

- Vector fetch (program counter loaded with contents of specified address)
- S, X, and I bits set in the CCR
- MCU hardware reset
- Checks for interrupts

Low Power Modes (LPM)

- To reduce power consumption of the controller
- Temporarily stops CPU operations until a reset or interrupt occurs
- 2 modes
  - WAIT
  - STOP
LPM: Wait Mode

- Command: WAI
- CPU always shut down during wait mode
- CPU registers are stacked
- Program is suspended until interrupted
- On-chip crystal oscillator remains active
- Power conservation depends on number of peripheral systems shut down
  - A/D, SPI, and SCI can be shut down by controlling respective bits

Stop Mode

- Command: STOP + S bit of CCR is clear
- Lowest possible power consumption
- All clocks are stopped (crystal oscillator too)
- Data in internal RAM is retained as long as $V_{DD}$ power is maintained
- Exit using RESET, XIRQ, or unmasked IRQ
- XIRQ has 2 recover methods
  - If X is set, returns to command following STOP
  - If X is clear, stacking sequence that leads to normal XIRQ request
Uses

- **Important events**
  - Prepare system for shutdown
  - Power supply switched to battery backup

- **Infrequent events**
  - Warning light in car
  - Activate switch when temperature gets too high

- **Slow events**
  - Can use end of conversion bit to signal reading of data because A/D converter is slow

Real World Applications

- Cell phones
- Pause button on VCR
- Input capture of optical encoder for DC motor lab
- Optical sensors
- Set a light to go on in a certain time
- Compiler recognizes an error (illegal opcode trap)
Example Program

Write a routine to interrupt the HC11 after 10 msec of elapsed time (Assume E=1Mhz, Prescaler = 1) using OC3.

```
ORG $C000
SEI /* Set I bit in CCR. Stops maskable interrupts from occurring */
LDAA #BIT5HI /* BIT5HI = %00100000 */
STAA TFLG1 /* Clear previously set OC3 Flag */
STAA TMSK1 /* Enable OC3 Interrupt */
LDAB #$30 /* $30 = %00110000 */
STAB TCTL1 /* OC3 (PA5) will be high for a successful compare */
LDAA #JUMP /* JUMP = $07E */
STAA FIRSTAD /* First AD = $00D9 */
LDX #OC3ISR /* OC3ISR = $D000, 2 bytes -- beginning address of interrupt service routine */
STX SECONDAD /* SECONDAD = $00DA. High byte (DO) stored in location $00DA and low byte (00) stored in $00DB */
LDD TCNT /* TCNT = $102E */
ADD #DLYIOMS /* DLYIOMS = $2710 = 10000 cycles or 10 msec */
STD TOC3 /* If not done elsewhere */
LDAA #$00 /* Clears CCR including I and X bits. Lets maskable and non-maskable interrupts occur. */
TAP /* TAP transfers Accumulator A to CCR */
```
### Ex. Program Technical Data (continued)

#### Timer Control Register 1 (TCTL1)

<table>
<thead>
<tr>
<th>Address: $1020</th>
<th>Timer Control Register 1 (TCTL1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT 7 6 5 4 3 2 1 0</td>
<td>OM2 OL2 OM3 OL3 OM4 OL4 OM5 OL5</td>
</tr>
<tr>
<td>Read:</td>
<td>Write:</td>
</tr>
<tr>
<td>Reset:</td>
<td>Program:</td>
</tr>
</tbody>
</table>

Code bit pairs are encoded to specify the action taken after a successful OCx compare.

#### Timer Output Compare Register Pair (T0C3)

<table>
<thead>
<tr>
<th>Address: $101A</th>
<th>Timer Output Compare 3 Register (high)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT 7 6 5 4 3 2 1 0</td>
<td>Bit 15 Bit 14 Bit 13 Bit 12 Bit 11 Bit 10 Bit 9 Bit 8</td>
</tr>
<tr>
<td>Read:</td>
<td>Write:</td>
</tr>
<tr>
<td>Reset:</td>
<td>Program:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address: $101B</th>
<th>Timer Output Compare 3 Register (low)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT 7 6 5 4 3 2 1 0</td>
<td>Bit 7 Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 Bit 0</td>
</tr>
<tr>
<td>Read:</td>
<td>Write:</td>
</tr>
<tr>
<td>Reset:</td>
<td>Program:</td>
</tr>
</tbody>
</table>

Timer Output Compare Register Pair (T0C3)