816 BEAT

By Mark Fellows

SUPERCPU INTERRUPTS

Handling interrupts on the SuperCPU is a unique challenge. Both the 65C816 and SuperCPU have a number of new registers and modes which must be considered when writing interrupt service routines. The aim of this article is to provide programmers with a solid foundation upon which to build their own SuperCPU-aware interrupt routines that take full advantage of the enhanced instruction set and extended addressing of the 65C816.

New 65C816 Interrupts

The 65C816 includes two new interrupts that may not be familiar to the C64/128 programmer: ABORT and COP.

ABORT is a hardware interrupt invoked by a negative pulse (or level) on the ABORTB pin of the 65C816. ABORT will inhibit modification of any internal register during the current instruction, and upon completion of the instruction, will initiate an interrupt sequence. The location of the aborted opcode is stored as the return address in stack memory. Besides RESET, ABORT has the highest interrupt priority. ABORT is not currently implemented in the SuperCPU, and is reserved for future use.

COP is a software interrupt initiated by executing the 65C816 COP instruction (opcode $02). The byte following the COP opcode is called the COP signature. COP signatures $00-$3F may be user defined. Signatures $40-$7F are reserved by the SuperCPU operating system, and signatures $80-$FF are reserved for use on future WDC microprocessors. A COP interrupt handler can examine the stack to find the location of the COP signature, and then examine the signature in order to determine the appropriate course of action.

Except for having a different vector location, COP works exactly like BRK—the other 65C816 software interrupt. Upon return from a COP or BRK interrupt the 65C816 resumes execution at a point two bytes ahead of the COP or BRK opcode (the processor does not execute the signature byte). Therefore, a BRK interrupt handler can also find and examine the signature byte in order to determine what action should be taken. All BRK signatures ($00-$FF) may be user defined.

Emulation Mode vs. Native Mode

There are important differences in the way that the 65C816 handles interrupts in the Emulation and Native modes. The programmer must be aware of these factors when designing new, larger programs for the SuperCPU which are located in banks above 00.

In emulation mode, the PBR (Program Bank Register) is automatically cleared to 00 whenever an interrupt occurs. The previous contents of the PBR is not saved on the stack. If an emulation mode program is running in a bank other than 00 when an interrupt occurs, the RTI instruction at the end of the interrupt service routine will not restore the PBR to its former value. Because of this, it is impractical to implement interrupt routines in emulation mode programs that are located in banks other than 00.

Fortunately, the 65C816 Native Mode does not have this shortcoming. When an interrupt occurs in Native Mode, the PBR is automatically saved on the stack. When an RTI is executed in Native Mode, the 65C816 automatically pulls the PBR off the stack so that execution of the interrupted routine will resume in the correct bank.

The Decimal Flag

On 6502/6510/8500/8502 processors, the Decimal (D) flag in the processor status register (Bit 3) is unknown after reset and remains unchanged after an interrupt occurs. On the 65C816, however, the D flag is cleared whenever a reset or interrupt occurs. In practice, this difference causes few problems with existing software written for the Commodore processors. There are two reasons: 1) On all processors, the former value of the status register (including the D flag) is restored upon execution of an RTI instruction and, 2) Commodore programs that use decimal mode usually include interrupt handlers that begin by clearing the decimal flag with the CLD instruction.

Interrupt Sequence

The 65C816 responds to an interrupt by initiating a sequence of machine cycles that first save the program counter and status register on the stack. Next, the address stored at the corresponding interrupt vector is moved into the program counter and the processor begins execution of the interrupt routine at that address. In the case of hardware interrupts (IRQ, NMI, ABORT, RESET), the 65C816 will wait until the currently executing instruction is completed before beginning the interrupt response sequence. An IRQ will not invoke the interrupt sequence if the interrupt disable flag has been set via the SEI instruction. All other interrupts (BRK, COP, NMI, ABORT, RESET) are non-maskable and will initiate a response regardless of the interrupt disable flag.

65C816 Interrupt Sequence (IRQ, NMI, ABORT, RESET)

Cycle	Operation
1	(internal operation)
2	(internal operation)
(3)	Push Program Bank Register (Native Mode only)
4	Push Program Counter (High Byte)*
5	Push Program Counter (Low Byte)*
6	Push Processor Status Register
7	Fetch Interrupt Vector Address (Low Byte)
8	Fetch Interrupt Vector Address (High Byte)
1	Fetch first opcode of Interrupt Routine

* Address of next instruction to be executed upon RTI

65C816 Interrupt Sequence (BRK, COP)

Cycle	Operation
1	Fetch BRK or COP opcode
2	Read BRK/COP Signature
(3)	Push Program Bank Register (Native Mode only)
4	Push Program Counter (High Byte)*
5	Push Program Counter (Low Byte)*
6	Push Processor Status Register
7	Fetch Interrupt Vector Address (Low Byte)
8	Fetch Interrupt Vector Address (High Byte)
1	Fetch first opcode of Interrupt Routine

* Address of BRK/COP Signature + 1

Vector Locations

The 6510/8500/8502 processors in the C64/128 have three interrupt vector locations: NMI at $FFFA,B; RESET at $FFFC,D; and the shared IRQ/BRK vector at $FFFE,F. In Emulation mode, the 65C816 uses these same locations, as well as the new ABORT vector at $FFF8,9 and the COP vector at $FFF4,5. Vector location $FFF6,7 is reserved for future use.

When the 65C816 is in Native mode, a new set of vector locations is employed. At first glance, the new vectors may seem to add unnecessary complexity, but in practice prove to be quite helpful because special interrupt handlers are needed in the Native mode to save and restore the various 65C816 registers and register states. The Native mode also provides the programmer with the luxury of a separate BRK vector location. This is both a convenience and a necessity because the BRK flag (Bit 4) in the processor status register becomes the Index Register Select bit in Native mode, thereby making it impossible for a combined IRQ/BRK interrupt handler to test Bit 4 in order to determine whether an IRQ or BRK has occurred. You may note that there is no RESET vector defined for the Native mode. A Native mode RESET vector is not necessary because the 65C816 switches itself to Emulation mode whenever a RESET occurs and as a result always fetches the RESET vector from the Emulation mode location of $FFFC,D. This leaves the Native mode vector location of $FFEC,D reserved for future implementation.

65C816 Interrupt Vector Locations

Emulation Mode		Native Mode	
Interrupt Vector	Location	Interrupt Vector	Location
IRQ/BRK	$FFFE,F	IRQ	$FFEE,F
RESET	$FFFC,D	(reserved)	$FFEC,D
NMI	$FFFA,B	NMI	$FFEA,B
ABORT	$FFF8,9	ABORT	$FFE8,9
(reserved)	$FFF6,7	BRK	$FFE6,7
COP	$FFF4,5	COP	$FFE4,5

Vector Access on the SuperCPU

Implementing the additional 65C816 interrupt vector locations was a challenge during the design of the SuperCPU. Upon examination, you will find that all the Native mode vectors (locations $FFE4—$FFEF) and the Emulation mode COP and ABORT vectors are located within the Kernal Jump Table! At first glance this seems to preclude the use of Native interrupts, ABORT or COP while the Kernal ROM is in context. Luckily, the 65C816's designers may have forseen such a circumstance and have provided the VPB signal which, when asserted, indicates when a vector fetch is taking place. The SuperCPU PLD decodes VPB and can remap memory so that the vectors are fetched from a table in another location.

Although the problem of accessing the additional vectors was solved, there were still some issues regarding compatibility with the C64/128 operating system that had to be addressed. In order to guarantee 100% compatibility with existing Commodore programs, the existing IRQ/BRK and NMI vectors and interrupt routines had to be duplicated exactly. This was no problem because the 65C816 in Emulation mode will execute the existing Kernal interrupt routines properly.

With the compatibility issue solved we moved on to the next problem—how to provide enhanced interrupt services in the Kernal for the 65C816 Native mode and for situations where an Emulation mode interrupt occurs when the SuperCPU hardware registers have been enabled. To solve these problems a second vector table was added that is accessed when one or more of the following conditions is true:

1) The 65C816 is in Native mode
2) The SuperCPU hardware registers are enabled ($D0B2 Bit 7 = 1)
3) The System 1MHz flag is true ($D0B2 Bit 6 = 1)
4) The DOS Extension mode is enabled ($D0BC Bit 7 = 1)
5) The RAMLink hardware registers are enabled ($D0BC Bit 6 = 1)

The enhanced vector table directs interrupts that occur under the above circumstances to a new set of routines in the SuperCPU Kernal.

Note: When the Kernal ROM is switched out, the vectors are accessed from the RAM 'under' the Kernal ROM. As on the C64, the programmer in this case must provide his own vector table and interrupt service routines.

Native Mode Interrupt Servicing

In order to provide compatibility with existing Commodore programs, the SuperCPU Kernal is by necessity an 8-bit operating system. This includes the Kernal interrupt service routines that scan the keyboard and control RS-232 communications. With new SuperCPU programs, a problem arises when the Kernal must service an interrupt that is generated while the 65C816 is in Native mode. To solve the problem, the Native mode interrupt service routines in the SuperCPU Kernal first save the 16-bit register values on the stack along with a re-entry address and then switch the 65C816 to Emulation mode. Control is then passed to the enhanced Emulation mode interrupt routines which perform the interrupt processing. When interrupt processing has finished, the re-entry routine switches the processor back to Native mode and restores the 16-bit registers.

Handling Native interrupts in this manner makes it easy for programmers to write new software for the SuperCPU that takes advantage of the 16-bit registers and other advanced features of the 65C816. Unless the Kernal ROM is switched out, programmers do not need to include any interrupt handlers to provide special service to interrupts that occur while the 65C816 is in Native mode. In addition, because the Kernal takes care of the switching from Native to Emulation mode, existing custom interrupt handlers do not have to be re-written and can be used as-is by linking into the standard Kernal redirection vectors at $0314—$0319.

Enhanced Emulation Mode Interrupt Servicing

Even in Emulation mode, the SuperCPU uses special registers that change the memory map. Because of this, additional interrupt routines had to be included in the Kernal to handle interrupts that occur when the memory map is not in its default configuration. The enhanced Emulation mode interrupt routines first save the status of the Hardware Register Enable Bit, the System 1MHz bit, the DOS Extension Mode bit, and the RAMLink Hardware Enable bit on the stack. These bits are then set to their default values (thereby restoring the default memory map) and a re-entry address is pushed on the stack. Control is then passed to the standard Commodore Kernal interrupt routines. After the interrupt routines have finished, the re-entry routine pulls the configuration bits off the stack, restores the memory map and then returns to the interrupted program.

Because the standard Kernal interrupt routines are utilized, programs can still redirect the Emulation mode interrupt routines by using the standard redirection vectors at $0314—$0319.

The Enhanced Interrupt Redirection Jump Table

In order to provide programmers with complete control over both Native mode and enhanced Emulation mode interrupt servicing, the SuperCPU Kernal includes the Interrupt Redirection Jump Table. If a programmer does not wish to let the Kernal take care of interrupt processing, the jump

table addresses can be changed so that interrupt servicing is redirected to custom interrupt handlers. For example, by changing the jump table a Native mode program can bypass the Kernal routines that switch the processor into emulation mode during interrupts; thus allowing the use of custom Native code interrupt handlers.

Each jump table location contains a JML instruction and a 3-byte long address operand. The JML is the first instruction executed during a Native mode or enhanced Emulation mode interrupt, thus giving the programmer the ability to redirect the entire interrupt processing routine.

Because the SuperCPU incorporates a dual-layer Kernal, the redirection address must be changed in two locations. Both locations are given in the table below. For example, to redirect the Native mode IRQ interrupt (located at $01FCAC in the jump table) a programmer would take the following steps:

1) Write the low byte of the new address to $01FCAD & $017CAD
2) Write the high byte of the new address to locations $01FCAE & $017CAE
3) Write the bank# of the new address to locations $01FCAF & $017CAF

Note: We did not write to locations $01FCAC & $017CAC because the JML opcode is stored in these locations.

Enhanced Interrupt Redirection Jump Table			
Location 1	Location 2	Vector	Default
$01FC80	$017C80	JML EMUL. COP	XRTI
$01FC84	$017C84	JML (reserved)	XRTI
$01FC88	$017C88	JML EMUL. ABORT	XRTI
$01FC8C	$017C8C	JML EMUL. NMI	ENMI
$01FC90	$017C90	JML EMUL. RESET	CPURES
$01FC94	$017C94	JML EMUL. IRQ/BRK	EIRQ
$01FC98	$017C98	JML NATIVE COP	XRTI
$01FC9C	$017C9C	JML NATIVE BRK	NBRK
$01FCA0	$017CA0	JML NATIVE ABORT	XRTI
$01FCA4	$017CA4	JML NATIVE NMI	NNMI
$01FCA8	$017CA8	JML (reserved)	CPURES
$01FCAC	$017CAC	JML NATIVE IRQ	NIRQ

SuperCPU Native Mode Interrupt Handler

Listed below is the current SuperCPU 64 Native mode interrupt handler. Its function is to save the 16-bit register values, perform the necessary steps required to switch the processor into Emulation mode, and then jump to the Emulation mode interrupt handler. One of the more complex elements is the sequence required to save and switch the current Native mode stack which may reside anywhere within the full 64K range of Bank 00. Programmers wishing to include an interrupt handler that performs a switch from Native to Emulation mode may use the following routine as a guide.

```
;(COP & ABORT ROUTINE)-------------------------

XRTI    RTI             ;DON'T DO ANYTHING

;(NATIVE MODE NMI ROUTINE)----------------------

NNMI    REP %00110000   ;ALL REGISTERS 16-BIT
        PHY             ;SAVE .Y (16 BITS)
        LDY #$00A4      ;$A4 IS PSEUDO NMI STATUS
                        ;AFTER SWITCH TO EMUL. MODE
        BNE GNI         ;BRANCH ALWAYS

;(NATIVE MODE BRK ROUTINE)----------------------

NBRK    REP %00110000   ;ALL REGISTERS 16-BIT
        PHY             ;SAVE .Y (16 BITS)
        LDY #$0034      ;$34 PSEUDO BRK STATUS
        BNE GNI         ;BRANCH ALWAYS

;(NATIVE MODE IRQ ROUTINE)----------------------

NIRQ    REP %00110000   ;ALL REGISTERS 16-BIT
        PHY             ;SAVE .Y (16 BITS)
        LDY #$0024      ;$24 PSEUDO IRQ STATUS

GNI     PHX             ;SAVE .X (16 BITS)
        PHA             ;SAVE .A (16 BITS)
        TSX             ;GET STACK PTR (16 BITS)
        TXA             ;TO .A REG.
        AND #$FF00      ;MASK OFF LOWER BYTE
        CMP #$0100      ;WAS STACK WITHIN PAGE 1?
        BEQ *           ;YES - LEAVE STACK ALONE
        LDA #$01FB      ;NO - LOAD NEW STACK VALUE
        TCS             ;TRANSFER TO STACK POINTER
*       PHX             ;SAVE OLD STACK PTR VALUE
        SEC             ;MAKE THE SWITCH
        XCE             ;TO EMULATION MODE
                        ;(ALL REGISTERS NOW 8 BITS)
        PEA NRTIIN      ;PUSH RE-ENTRY ADDRESS
        TYA             ;GET/TEST PSEUDO-STATUS
        PHA             ;PUSH STATUS VALUE
        BPL EIRQ        ;BRANCH IF IRQ/BRK
                        ;NMI FALLS THRU TO 'ENMI'
```

SuperCPU Emulation Mode Interrupt Handler

The SuperCPU 64 enhanced Emulation mode interrupt handler performs a number of functions not present in the standard Commodore Kernal interrupt routines. These functions are necessary to save and restore additional 65C816 registers and the current SuperCPU hardware state. The additional registers saved are the .B register, the Direct Register and the Data Bank Register. The Direct Register and Data Bank register are then both cleared to 0.

The SuperCPU and RAMLink hardware states are saved as well, and then cleared to their default values (SuperCPU hardware out, DOS Extension mode off, System Speed = Turbo, and RAMLink hardware out). The handler then pushes the re-entry address ('RTIIN') on the stack so that the handler can regain control upon RTI and restore the SuperCPU and RAMLink hardware states and additional 65C816 registers.

```
;(EMULATION MODE NMI ROUTINE)-------------------

ENMI    SEC             ;FLAG FOR LATER
        BCS IENTR       ;BRANCH ALWAYS

;(EMULATION MODE IRQ/BRK ROUTINE)---------------

EIRQ    CLC             ;FLAG FOR LATER
IENTR   PHA             ;SAVE .A
        LDA #$00        ;USE LATER TO CLR DIRECT
        XBA             ;EXCHANGE .B & .A
        PHA             ;SAVE .B
```

```
        LDA #$00        ;READY TO CLEAR DIRECT REG.
        PHD             ;SAVE DIRECT REGISTER
        TCD             ;DIRECT REGISTER =$0000
        PHB             ;SAVE DATA BANK REGISTER
        PHA             ;PUSH $00 ON STACK
        PLB             ;DATA BANK REGISTER =$00
        LDA $D0B2       ;H/W REG & SYS SPEED STATUS
        STA $D073       ;SET SYS SPEED TO TURBO
        PHA             ;SAVE H/W & SYS SPD STATUS
        LDA $D0BC       ;GET DOS EXT. & RL STATUS
        PHA             ;SAVE IT
        AND #%01000000  ;TEST FOR RAMLINK H/W IN
        BEQ ++          ;BRANCH NO
        BCC +           ;BRANCH IF IRQ/BRK
        STA $DF20       ;IF NMI, RL WRT PROTECT OFF
+       STA $DF7F       ;SWITCH OUT RL H/W REG.
+       STA $D0BF       ;CPU DOS EXT. MODE OFF
        PEA RTIIN       ;SETUP RE-ENTRY ADDRESS
        LDA (10,S)      ;STATUS REGISTER FROM STACK
        PHA             ;PUSH IT
        STA $D07F       ;CPU HARDWARE OUT
        LDA (10,S)      ;.A REGISTER FROM STACK
        BCS +           ;BRANCH IF NMI
        JMP $FF48       ;IRQ/BRK TO STOCK ROUTINE
+       JMP ($0318)     ;NMI
```

SuperCPU Emulation Mode Re-entry Routine

The Emulation mode re-entry routine restores the SuperCPU and RAMLink hardware states and additional 65C816 registers. Control is passed to this routine when the stock interrupt routine executes an RTI. Re-entry was enabled by the interrupt handler above which pushed the re-entry address (PEA RTIIN) and the status register value on the stack.

```
RTIIN   STA $D07E       ;CPU HARDWARE IN
        PLA             ;GET DOS EXT. & RL STATUS
        STA $D0BC       ;RESTORE DOS/RL STATUS
        AND #$40        ;TEST FOR RAMLINK ENABLED
        BEQ +           ;NO - SKIP
        STA $DF7E       ;YES - RE-ENABLE RAMLINK
+       PLA             ;GET CPU REG. & SYS SPD
        STA $D0B2       ;RESTORE CPU REG. & SYS SPD
        PLB             ;RESTORE DATA BANK REGISTER
        PLD             ;RESTORE DIRECT REGISTER
        PLA             ;GET .B FROM STACK
        XBA             ;RESTORE .B
        PLA             ;RESTORE .A
        RTI
```

SuperCPU Native Mode Re-entry Routine

The Native mode re-entry routine re-enables Native mode, then restores the stack pointer and 16-bit register values. Control is passed to this routine when the Emulation mode re-entry routine executes its RTI. Re-entry was enabled by the Native interrupt handler which pushed the re-entry address (PEA NRTIIN) and the status register value on the stack.

```
NRTIIN CLC              ;SWITCH TO
       XCE              ;NATIVE MODE
       REP %00110000    ;ALL REGISTERS 16-BIT
       PLA              ;GET OLD STACK LOCATION
       TCS              ;AND RESTORE IT
       PLA              ;RESTORE .A
       PLX              ;RESTORE .X
       PLY              ;RESTORE .Y
       RTI
```

Sample Native-Mode-Only Interrupt Handler

The interrupt handler below is provided for programmers who wish to install their own Native-mode-only interrupt routine. All 65C816 registers are saved along with the current SuperCPU and RAMLink hardware states. This routine contains elements found in both the SuperCPU Native and Emulation mode interrupt handlers, but is much more efficient because the switch to Emulation mode is not performed.

The routine is installed by changing the appropriate entry in the Interrupt Redirection Jump Table. Note: This routine changes the Direct Register to $0000 and the Data Bank Register to $00. However, the programmer is free to substitute other values as required by the custom interrupt routine.

```
;(SAMPLE NATIVE-MODE-ONLY INTERRUPT ROUTINE)-----

IENTER REP %00110000    ;ALL REGISTERS 16-BIT
       PHA              ;SAVE .A,.B
       PHX              ;SAVE .X
       PHY              ;SAVE .Y
       PHD              ;SAVE DIRECT REG.
       LDA #$0000       ;ALL 16 BITS = 0
       TCD              ;DIRECT REGISTER = $0000
       SEP %00100000    ;ACCUMULATOR 8-BIT
       PHB              ;SAVE DATA BANK REGISTER
       PHA              ;PUSH $00 ON STACK
       PLB              ;DATA BANK REGISTER =$00
       LDA $D0B2        ;H/W REG & SYS SPEED STATUS
       STA $D073        ;SET SYS SPEED TO TURBO
       PHA              ;SAVE H/W & SYS SPD STATUS
       LDA $D0BC        ;GET DOS EXT. & RL STATUS
       PHA              ;SAVE IT
       AND #%01000000   ;TEST FOR RAMLINK H/W IN
       BEQ +            ;BRANCH NO
       STA $DF20        ;RL WRT PROTECT OFF
+      STA $DF7F        ;SWITCH OUT RL H/W REG.
+      STA $D0BF        ;CPU DOS EXT. MODE OFF
       STA $D07F        ;CPU HARDWARE OUT

       (CUSTOM INTERRUPT SERVICE ROUTINE HERE)

IEXIT  SEP %00100000    ;ACCUMULATOR 8-BIT
       STA $D07E        ;CPU HARDWARE IN
       PLA              ;GET DOS EXT. & RL STATUS
       STA $D0BC        ;RESTORE DOS/RL STATUS
       AND #$40         ;TEST FOR RAMLINK ENABLED
       BEQ +            ;NO - SKIP
       STA $DF7E        ;YES - RE-ENABLE RAMLINK
+      PLA              ;GET CPU REG. & SYS SPD
       STA $D0B2        ;RESTORE CPU REG. & SYS SPD
       PLB              ;RESTORE DATA BANK REGISTER
       REP %00110000    ;ALL REGISTERS 16-BIT
       PLD              ;RESTORE DIRECT REGISTER
       PLY              ;RESTORE .Y
       PLX              ;RESTORE .X
       PLA              ;RESTORE .A
       RTI
```