

## Buffalo Function Calls and Interrupt Vectors

### Buffalo Function Calls

Buffalo supports several function calls that perform various housekeeping functions, such as displaying characters on the serial-port terminal. These calls are listed below.

To use them, take two steps:

1. Use an EQU statement to set up the address correctly. For example, if you want to use OUTA, use `OUTA EQU $FFB8`.
2. To call the function, JSR to the label defined in the EQU statement. For example, you can do `JSR OUTA`.

<code>\$FFA0</code>	<code>JMP</code>	<code>UPCASE</code>	Convert character to uppercase
<code>\$FFA3</code>	<code>JMP</code>	<code>WCHEK</code>	Test character for whitespace
<code>\$FFA6</code>	<code>JMP</code>	<code>DCHEK</code>	Check character for delimiter
<code>\$FFA9</code>	<code>JMP</code>	<code>INIT</code>	Initialize I/O device
<code>\$FFAC</code>	<code>JMP</code>	<code>INPUT</code>	Read I/O device
<code>\$FFAF</code>	<code>JMP</code>	<code>OUTPUT</code>	Write I/O device
<code>\$FFB2</code>	<code>JMP</code>	<code>OUTLHLF</code>	Convert left nibble to ASCII and output
<code>\$FFB5</code>	<code>JMP</code>	<code>OUTRHLF</code>	Convert right nibble to ASCII and output
<code>\$FFB8</code>	<code>JMP</code>	<code>OUTA</code>	Output ASCII character
<code>\$FFBB</code>	<code>JMP</code>	<code>OUT1BYT</code>	Convert binary byte to 2 ASCII characters and output
<code>\$FFBE</code>	<code>JMP</code>	<code>OUT1BSP</code>	Convert binary byte to 2 ASCII characters and output followed by space
<code>\$FFC1</code>	<code>JMP</code>	<code>OUT2BSP</code>	Convert 2 consecutive binary bytes to 4 ASCII characters and output followed by space.
<code>\$FFC4</code>	<code>JMP</code>	<code>OUTCRLF</code>	Output ASCII carriage return followed by line feed
<code>\$FFC7</code>	<code>JMP</code>	<code>OUTSTRG</code>	Output ASCII string until end of transmission (\$04)
<code>\$FFCA</code>	<code>JMP</code>	<code>OUTSTRGO</code>	Same as OUTSTRG except leading carriage return and line feed is skipped
<code>\$FFCD</code>	<code>JMP</code>	<code>INCHAR</code>	Input ASCII character and echo back
<code>\$FFD0</code>	<code>JMP</code>	<code>VECINIT</code>	Initialize indirect vectors in RAM

A more detailed description of each function:

UPCASE	If character in accumulator A is lower case alpha, convert to upper case.
WCHEK	Test character in accumulator A and return with Z bit set if character is white space (space, comma, tab).
DCHEK	Test character in accumulator A and return with Z bit set if character is delimiter (carriage return or white space).
INIT	Initialize I/O device.
INPUT	Read I/O device.
OUTPUT	Write I/O device.
OUTLHLF	Convert left nibble of accumulator A contents to ASCII and output to terminal port.
OUTRHLF	Convert right nibble of accumulator A contents to ASCII and output to terminal port.
OUTA	Output accumulator A ASCII character.
OUT1BYT	Convert binary byte at address in index register X to two ASCII characters and output. Returns address in index register X pointing to next byte.
OUT1BSP	Convert binary byte at address in index register X to two ASCII characters and output followed by a space. Returns address in index register
OUT2BSP	Convert two consecutive binary bytes starting at address in index register X to four ASCII characters and output followed by a space. Returns address in index register X pointing to next byte.
OUTCCRLF	Output ASCII carriage return followed by a line feed.
OUTSTRG	Output string of ASCII bytes pointed to by address in index register X until character is an end of transmission (\$04).
OUTSTRGO	Same as OUTSTRG except leading carriage return and line feed is skipped.
INCHAR	Input ASCII character to accumulator A and echo back. This routine loops until character is actually received.

Utility jump subroutines for performing I/O tasks are shown below. These subroutines are in ROM and are programmed as jumps. To use the jump subroutine, execute a JSR to the applicable address shown below.

\$FFA0	JMP UPCASE	Convert character to uppercase
\$FFA3	JMP WCHEK	Test character for white space
\$FFA6	JMP DCHEK	Check character for delimiter
\$FFA9	JMP INIT	Initialize I/O device
\$FFAC	JMP INPUT	Read I/O device
\$FFAF	JMP OUTPUT	Write I/O device
\$FFB2	JMP OUTLHLF	Convert left nibble to ASCII and output
\$FFB5	JMP OUTRHLF	Convert right nibble to ASCII and output
\$FFB8	JMP OUTA	Output ASCII character
\$FFBB	JMP OUT1BYT	Convert binary byte to 2 ASCII characters and output
\$FFBE	JMP OUT1BSP	Convert binary byte to 2 ASCII characters and output followed by space
\$FFC1	JMP OUT2BSP	Convert 2 consecutive binary bytes to 4 ASCII characters and output followed by space.
\$FFC4	JMP OUTCRLF	Output ASCII carriage return followed by line feed
\$FFC7	JMP OUTSTRG	Output ASCII string until end of transmission (\$04)
\$FFCA	JMP OUTSTRGO	Same as OUTSTRG except leading carriage return and line fees is skipped
\$FFCD	JMP INCHAR	Input ASCII character and echo back

## Buffalo Interrupt Vectors

Buffalo maps each of the interrupts listed below to the location shown in the table. When the interrupt occurs, Buffalo moves execution to that location. To install your own interrupt handler, go to the indicated location (with an ORG statement) and put in a jump instruction at that location to your handler.

For example, if you have an interrupt handler named HANDL and you want to use it to handle a Timer Overflow interrupt, you can do the following.

```
ORG $00D0
JMP HANDL

ORG $8000
HANDL <start of interrupt handler>
```

**Table 3-1. Interrupt Vector Jump Table**

INTERRUPT VECTOR	FIELD
Serial Communications Interface (SCI)	\$00C4-\$00C6
Serial Peripheral Interface (SPI)	\$00C7-\$00C9
Pulse Accumulator Input Edge	\$00CA-\$00CC
Pulse Accumulator Overflow	\$00CD-\$00CF
Timer Overflow	\$00D0-\$00D2
Timer Output Compare 5	\$00D3-\$00D5
Timer Output Compare 4	\$00D6-\$00D8
Timer Output Compare 3	\$00D9-\$00DB
Timer Output Compare 2	\$00DC-\$00DE
Timer Output Compare 1	\$00DF-\$00E1
Timer Input Capture 3	\$00E2-\$00E4
Timer Input Capture 2	\$00E5-\$00E7
Timer Input Capture 1	\$00E8-\$00EA
Real Time Interrupt	\$00EB-\$00ED
IRQ	\$00EE-\$00F0
XIRQ	\$00F1-\$00F3
Software Interrupt (SWI)	\$00F4-\$00F6
Illegal Opcode	\$00F7-\$00F9
Computer Operating Properly (COP)	\$00FA-\$00FC
Clock Monitor	\$00FD-\$00FF