

Example BLITZ Assembly Program

```
! Example.s -- Serial I/O Interface Routines
!  
! Harry Porter - 08/01/01
!  
! This program serves as an example of BLITZ assembly code and
! of the recommended style for indenting and commenting assembly code.
!  
! This program provides a "main" function which reads input characters
! from the terminal and echoes them back. It can be used to explore
! the differences between "raw" and "cooked" serial input modes.
!  
! In addition to the "main" function, this program also provides the
! following interface for the serial I/O device; these routines might
! provide the starting point for some other program.
!  
    .export getChar
    .export putChar
    .export putString
    .export flush
    .export initSerial
    .export checkSerialDevice
!  
! Program entry point
!  
    .text
_entry:
  
  
!  
! Here is the interrupt vector, which will be loaded at address 0x00000000.
! Each entry is 4 bytes. They are located at fixed, pre-defined addresses.
! This program will only handle SERIAL_INTERRUPTS. The asynchronous,
! hardware interrupts (i.e., TIMER and DISK) will be ignored by returning
! immediately. None of the other interrupts should occur; if they do, this
! program will get stuck in an infinite loop.
!  
PowerOnReset:
    jmp     main
TimerInterrupt:
    reti
DiskInterrupt:
    reti
SerialInterrupt:
    jmp     SerialInterruptHandler
HardwareFault:
    jmp     HardwareFault
IllegalInstruction:
    jmp     IllegalInstruction
ArithmeticException:
    jmp     ArithmeticException
AddressException:
    jmp     AddressException
PageInvalidException:
    jmp     PageInvalidException
PageReadOnlyException:
    jmp     PageReadOnlyException
PrivilegedInstruction:
    jmp     PrivilegedInstruction
AlignmentException:
    jmp     AlignmentException
ExceptionDuringInterrupt:
```

Example BLITZ Assembly Program

```
        jmp      ExceptionDuringInterrupt
SyscallTrap:
        jmp      SyscallTrap

!
! Interrupt Service routines
!
SerialInterruptHandler:
        call    checkSerialDevice
        reti

!
! main
!
! The main program repeatedly prints a prompt, then gets and echoes characters
! until a NEWLINE is entered.  It then repeats the prompt.
!
main:
        set     STACK_START,r15      ! initialize the stack pointer
        call    initSerial           ! initialize the serial I/O device
        seti    ! enable interrupts
loop1:  ! loop
        set     prompt,r1            ! putString ("Enter something: ")
        call    putString            ! .
loop2:  ! loop
        call    getChar              ! r1 := getChar
        cmp     r1,'\n'              ! if (r1 == '\n' or '\r') then
        be     then                  ! .
        cmp     r1,'\r'              ! .
        bne    else                  ! .
then:   ! .
        mov     '\r',r1              ! putChar ('\r')
        call    putChar              ! .
        mov     '\n',r1              ! putChar ('\n')
        call    putChar              ! .
        jmp     exit                ! break
else:   ! else
        cmp     r1,'q'              ! if (r1 == 'q') then
        bne    else2                ! .
        set     bye,r1               ! print "Good bye"
        call    putString            ! .
        call    flush                ! wait until I/O completes
        debug   ! Invoke DEBUG instruction
        jmp     cont                ! else
else2:  ! .
        call    putChar              ! putChar (r1)
cont:   ! end
        jmp     loop2               ! end
exit:   ! .
        jmp     loop1               ! end
prompt: .ascii "Enter something (or 'q' to terminate): \n\r\0"
bye:    .ascii "\n\rAbout to execute DEBUG instruction (type 'go' to resume)...
\n\r\0"
        .align

!
! getChar
!
! This routine reads one character from the terminal and returns it in r1.
! It does not echo the character or process special characters in any way.
! It checks the input buffer and gets a character from there if one is
! available.  Otherwise, it waits for a key to be typed.
!
```

Example BLITZ Assembly Program

```
! r1 = the character
! r2 = addr of inBufferCount
! r3 = inBufferCount
! r4 = addr of inBufferOut
! r5 = inBufferOut
!
! Registers modified: r1
!
getChar:
    push    r2                ! save registers
    push    r3                ! .
    push    r4                ! .
    push    r5                ! .
    set     inBufferCount,r2  ! initialize address registers
    set     inBufferOut,r4    ! .
getChLoop:
    ! loop
    !   loop
    cleari                !   disable interrupts
    load    [r2],r3         !   if (inBufferCount != 0)
    cmp     r3,0            !   .
    bne     getChExit       !   then break
    seti                    !   enable interrupts
    jmp     getChLoop        !   end
getChExit:
    !   .
    sub     r3,1,r3         !   inBufferCount --
    store   r3,[r2]         !   .
    load    [r4],r5         !   r1 := *inBufferOut
    loadb   [r5],r1         !   .
    add     r5,1,r5         !   inBufferOut ++
    cmp     r5,inBufferEnd  !   if (inBufferOut == inBufferEnd)
    bne     getChElse       !   .
    set     inBuffer,r5     !   inBufferOut := &inBuffer
getChElse:
    !   end
    store   r5,[r4]         !   save inBufferOut
    seti                    !   enable interrupts
    cmp     r1,'\0'         ! until (r1 != '\0')
    be      getChLoop       ! .
    pop     r5                ! restore regs
    pop     r4                ! .
    pop     r3                ! .
    pop     r2                ! .
    ret                                ! return

!
! putChar
!
! This routine is passed a character in r1.  It writes it to the terminal
! exactly as it is.  Normally, the output character is added to a buffer
! and will be written as soon as the device is ready.  If the buffer is
! full, this routine will busy-wait for the buffer to become not-full.
!
! r1 = the character
! r2 = addr of outBufferCount
! r3 = outBufferCount
! r4 = addr of outBufferIn
! r5 = outBufferIn
!
! Registers modified: none
!
putChar:
    push    r2                ! save registers
    push    r3                ! .
    push    r4                ! .
    push    r5                ! .
    set     outBufferCount,r2 ! initialize address registers
    set     outBufferIn,r4    ! .
putChLoop:
    ! loop
```

Example BLITZ Assembly Program

```

cleari                ! disable interrupts
load   [r2],r3        ! if (outBufferCount < BUFFER_SIZE)
cmp    r3,BUFFER_SIZE ! .
bl     putChExit      ! then break
seti   putChExit      ! enable interrupts
jmp    putChLoop      ! end
putChExit:            ! .
add    r3,1,r3        ! outBufferCount ++
store  r3,[r2]        ! .
load   [r4],r5        ! *outBufferIn := r1
storeb r1,[r5]        ! .
add    r5,1,r5        ! outBufferIn ++
cmp    r5,outBufferEnd ! if (outBufferIn == outBufferEnd) then
bne   putChElse      ! .
set    outBuffer,r5   ! outBufferIn := &outBuffer
putChElse:           ! end
store  r5,[r4]        ! save outBufferIn
call   checkSerialDevice ! start output if necessary
seti   ! enable interrupts
pop    r5              ! restore regs
pop    r4              ! .
pop    r3              ! .
pop    r2              ! .
ret    ! return

```

```

!
! putString
!
! This routine is passed a pointer to a string of characters, terminated
! by '\0'. It sends all of them except the final '\0' to the terminal by
! calling 'putChar' repeatedly.
!
! Registers modified: none
!

```

```

putString:
push   r1              ! save registers
push   r2              ! .
mov    r1,r2          ! r2 := ptr into string
putStLoop:
loadb  [r2],r1        ! r1 := next char
add    r2,1,r2        ! incr ptr
cmp    r1,0           ! if (r1 == '\0')
be     putStExit      ! then break
call   putChar        ! putChar (r1)
jmp    putStLoop      ! end
putStExit:
pop    r2              ! restore regs
pop    r1              ! .
ret    ! return

```

```

!
! flush
!
! This routine waits until the output buffer has been emptied, then returns.
! It busy-waits until the buffer has been emptied.
!
! Registers modified: none
!

```

```

flush:
push   r1              ! save registers
push   r2              ! .
flushLoop:
cleari                ! disable interrupts
set    outBufferCount,r1 ! r2 = outBufferCount
load   [r1],r2        ! .

```

Example BLITZ Assembly Program

```

    cmp     r2,0           ! if (r2 == 0)
    be     flushLoopEx    ! break
    seti   flushLoop     ! re-enable interrupts
    jmp    flushLoop     ! end
flushLoopEx:
    seti   .              ! .
    seti   .              ! re-enable interrupts
    pop    r2             ! restore regs
    pop    r1             ! .
    ret    .              ! return

!
! initSerial
!
! This routine initializes the serial input and output buffers.
!
! Registers modified: r1, r2
!
initSerial:
    set    inBuffer,r1    ! inBuferIn = &inBuffer
    set    inBufferIn,r2  ! .
    store  r1,[r2]        ! .
    set    inBufferOut,r2 ! inBufferOut = &inBuffer
    store  r1,[r2]        ! .
    set    outBuffer,r1   ! outBufferIn = &outBuffer
    set    outBufferIn,r2 ! .
    store  r1,[r2]        ! .
    set    outBufferOut,r2 ! outBufferOut = &outBuffer
    store  r1,[r2]        ! .
    clr    r1              ! inBufferCount = 0
    set    inBufferCount,r2 ! .
    store  r1,[r2]        ! .
    set    outBufferCount,r2 ! outBufferCount = 0
    store  r1,[r2]        ! .
    ret    .              ! return

!
! checkSerialDevice
!
! This routine is called whenever there is a SerialInterrupt. If a character
! is ready on the input, it is moved into the inBuffer. If there is no
! more room in the buffer, the character is simply dropped, with no
! error indication. If the output device is ready to for another character
! and there are any characters in the outBuffer, then the next character is
! transmitted to the output device.
!
! No arguments, no result.
! This routine must be called with interrupts disabled!
! Registers modified: none
!
! r8 = addr of SERIAL_STATUS_WORD
! r1 = SERIAL_STATUS_WORD
! r11 = addr of SERIAL_DATA_WORD
! r2 = the character
! r9 = addr of inBufferCount
! r5 = inBufferCount
! r10 = addr of inBufferIn
! r3 = inBufferIn
! r7 = addr of outBufferOut
! r4 = outBufferOut
! r6 = addr of outBufferCount
! r5 = outBufferCount
!
checkSerialDevice:
    push   r1             ! save all registers we use
    push   r2             ! .
    push   r3             ! .
```

Example BLITZ Assembly Program

```

push    r4                ! .
push    r5                ! .
push    r6                ! .
push    r7                ! .
push    r8                ! .
push    r9                ! .
push    r10               ! .
push    r11               ! .
set     SERIAL_DATA,r11   ! r11 = addr of SERIAL_DATA_WORD
set     SERIAL_STAT,r8    ! r1 := serial status word
load    [r8],r1           ! .
btst   0x00000001,r1     ! if status[charAvail] == 1 then
be     end1               ! .
load    [r11],r2         ! r2 := input char
set     inBufferCount,r9 ! if inBufferCount < bufSize then
load    [r9],r5          ! .
cmp     r5,BUFFER_SIZE   ! .
bge    end1              ! .
set     inBufferIn,r10    ! *inBufferIn := char
load    [r10],r3         ! .
storeb r2,[r3]           ! .
add     r5,1,r5           ! inBufferCount++
store  r5,[r9]           ! .
add     r3,1,r3           ! inBufferIn++
cmp     r3,inBufferEnd   ! if inBufferIn == inBufferEnd then
bne    end2              ! .
set     inBuffer,r3      ! inBufferIn = &inBuffer
end2:   ! end
store  r3,[r10]         ! store inBufferIn
end1:   ! end
! end
btst   0x00000002,r1     ! if status[outputReady] == 1 then
be     end4               ! .
set     outBufferCount,r6 ! if outBufferCount>0 then
load    [r6],r5          ! .
cmp     r5,0              ! .
ble    end4              ! .
set     outBufferOut,r7  ! r2 := *outBufferOut
load    [r7],r4          ! .
loadb  [r4],r2           ! .
store  r2,[r11]         ! send char in r2 to serial output
sub     r5,1,r5           ! outBufferCount--
store  r5,[r6]           ! .
add     r4,1,r4           ! outBufferOut++
cmp     r4,outBufferEnd  ! if outBufferOut==outBufferEnd then
bne    end3              ! .
set     outBuffer,r4     ! outBufferOut = &outBuffer
end3:   ! end
store  r4,[r7]         ! store outBufferOut
end4:   ! end
! end
pop     r11              ! restore all registers
pop     r10              ! .
pop     r9               ! .
pop     r8               ! .
pop     r7               ! .
pop     r6               ! .
pop     r5               ! .
pop     r4               ! .
pop     r3               ! .
pop     r2               ! .
pop     r1               ! .
ret     ! return

```

```

.data
BUFFER_SIZE = 128
inBuffer:   .skip BUFFER_SIZE ! Serial Input buffer area
inBufferEnd:

```

Example BLITZ Assembly Program

```
inBufferIn:    .word    0           ! Addr of next place to add to
inBufferOut:   .word    0           ! Addr of next place to remove from
inBufferCount: .word    0           ! Number of characters in inBuffer

outBuffer:     .skip    BUFFER_SIZE ! Serial Output buffer area
outBufferEnd:  .word    0
outBufferIn:   .word    0           ! Addr of next place to add to
outBufferOut:  .word    0           ! Addr of next place to remove from
outBufferCount: .word    0         ! Number of characters in outBuffer

STACK_START   =         0x00ffff00
SERIAL_STAT   =         0x00ffff00 ! Addr of SERIAL_STATUS_WORD
SERIAL_DATA   =         0x00ffff04 ! Addr of SERIAL_DATA_WORD
```