```
ASM    = asm
AFLAGS = -iinclude:
CC     = lc
CFLAGS = -cfistE
LN     = blink

LIBS = LIB:lc.lib LIB:amiga.lib

.c.o:
            $(CC) $(CFLAGS) $*.c

.asm.o:
            $(ASM) $(AFLAGS) $*.asm

4play: 4play.o read34.o
            $(LN) FROM LIB:c.o,4play.o,read34.o TO 4play LIBRARY $(LIBS)
```

```
/*
 * 4play.c
 */

#include    <exec/types.h>
#include    <libraries/dos.h>

#include    <stdlib.h>
#include    <stdio.h>

#include    <clib/exec_protos.h>

UBYTE portdata;
UBYTE *portptr = &portdata;

UBYTE firedata;
UBYTE *fireptr = &firedata;

extern int getport(void);
extern void read34(void);
extern void freeport(void);


/*
 * Lattice control-c stop...
 */
int CXBRK(void) { return(0); }      /* Disable Lattice CTRL/C handling */
int chkabort(void) { return(0); }   /* really */


void Quit(char whytext[], LONG return_code)
{
    if(return_code==0) freeport();  /* Assembly routine to
                                       de-allocate parallel port */

    printf("%s\n",whytext);

    exit(return_code);              /* returning non-zero
                                       terminates the program */
}


void main(void)
{
    BOOL done=FALSE;
    UBYTE error;

    /* getport() is an assembly routine that allocates the parallel port
     * and makes all the lines we're interested in "read" lines.
     */
    if(error=getport()) Quit("Parallel port in use",25);

    /* WARNING:
     * This example continuously reads the ports and checks for CTRL_C,
     * thereby eating a lot of CPU time.  Actual applications that expect
     * to be even more system friendly might want to set up some interrupts
     * on the fire button lines, such that the game can read the ports less
     * often, but never miss a "fire" press.
     */

    while(!done)
    {
        read34();       /* read34() is the assembly routine that copies the
                         * relavent data from the port into our variables.
                         */

        /* We'll just print the raw bytes from the read, and leave it as an
         * exercise for the reader to mask out the relevant bits.
         * (Check the pinouts to find which bits the switches appear at.)
         */
        printf("portdata = %u, firedata = %u\n",portdata,firedata);

        /* Check CTRL_C */
        if(SetSignal(0L,0L) & SIGBREAKF_CTRL_C) /* Hit since last check? */
        {
            SetSignal(0L,SIGBREAKF_CTRL_C); /* Clear old status */
            done=TRUE;
        }
    }
    Quit("Ctrl-C was pressed.",0);
}
```

```
;
; read34.asm
;
; interface code for the "2 more players" parallel port hack.
;
      csect text         ; this here's the meat

      xdef  Name         ; Name of our application, so that
Name  dc.b  '4play',0    ; other applications will know
                         ; who's tying up the port. ;-)


      xdef  _read34      ; function names for linker
      xdef  _getport
      xdef  _freeport

      xref  _portptr     ; c pointer for port data
      xref  _fireptr     ; c pointer for fire buttons

      xref  _SysBase     ; exec system base (from c.o)


      INCLUDE      "resources/misc.i"


      xdef  MiscName
MiscName    MISCNAME     ; macro from resources/misc.i

      xdef  _MiscResource
_MiscResource      dc.l  0; place to store misc.resource base

;parallel port hardware addresses (from amiga.lib)

      xref  _ciaaprb     ; the actual port address
      xref  _ciaaddrb    ; data direction register

      xref  _ciabpra     ; control lines are here
      xref  _ciabddra    ; data direction register
;from amiga.lib
      xref  _LVOOpenResource
      xref  _LVOAllocMiscResource
      xref  _LVOFreeMiscResource
```

```
_getport
;This routine simply allocates the parallel port in a system friendly
;way, and sets up the lines we want to use as input lines.
;


;save registers on the stack

        movem.l     a2-a6/d2-d7,-(sp); push regs


;open the misc.resource

        lea     MiscName,a1  ; put name of misc.resource in a1
        movea.l     _SysBase,a6; put SysBase in a6
        jsr     _LVOOpenResource(a6)
        move.l  d0,_MiscResource; store address of misc.resource
        bne.s   grabit

;Oops, couldn't open misc.resource.  Sounds like big trouble to me.

        moveq   #20,d0      ; error code
        bra     done


;This is where we grab the hardware.  If some other task has allocated
;the parallel data port or the parallel control bits, this routine will
;return non-zero.

;This part grabs the port itself

grabit  lea     Name,a1     ; The name of our app
        moveq   #MR_PARALLELPORT,d0; what we want
        movea.l     _MiscResource,a6; MiscResource Base is in A6
        jsr     _LVOAllocMiscResource(a6)
        move.l  d0,d1
        beq.s   grab2

;well, somebody else must've got the port first.

        moveq   #30,d0      ; error code
        bra     done

;This part grabs the control bits (busy, pout, and sel.)
;We really don't need pout, but it comes free with PARALLELBITS,
;so we'll take it anyway.

grab2   lea     Name,a1     ; The name of our app
        moveq   #MR_PARALLELBITS,d0; what we want
        jsr     _LVOAllocMiscResource(a6)
        move.l  d0,d1
        beq.s   setread

;well, somebody else must've got the bits first.

        moveq   #40,d2
        bra     freepar


;set up parallel port for reading

setread     move.b  #0,_ciaaddrb; all lines read

        andi.b  #$FF,_ciabddra; busy, pout, and sel. to read

;Well, we made it this far, so we've got exclusive access to
;the parallel port, and all the lines we want to use are
;set up.   From here we can just put back the regs and return to
;the caller.

        bra     done

;If something happened AFTER we got exclusive access to the parallel port,
;we'll need to let go of the port before we return the error.

freepar     moveq   #MR_PARALLELPORT,d0
        movea.l     _MiscResource,a6
        jsr     _LVOFreeMiscResource(a6)

        move.l  d2,d0       ; put error code into d0


;Restore registers and return
;(error code is in d0)

done    movem.l (sp)+,a2-a6/d2-d7; pop regs
        rts


_freeport
;This routine just makes sure that we let go of the parallel port and
;control lines, so somebody else can use 'em, now that we're all done.
;
;PS - Don't call this one if you got an error from _getport, as some
;of the resources might not have been opened, etc.
;

;save registers on the stack

        movem.l     a2-a6/d2-d7,-(sp); push regs


;free control lines

        moveq   #MR_PARALLELBITS,d0
        movea.l     _MiscResource,a6
        jsr     _LVOFreeMiscResource(a6)


;free parallel port

        moveq   #MR_PARALLELPORT,d0
        movea.l     _MiscResource,a6
        jsr     _LVOFreeMiscResource(a6)


;Clean up, restore registers, and return

        movem.l (sp)+,a2-a6/d2-d7; pop regs
        rts


_read34
;All this routine does is copy the data from the ports to other addresses.
;
;In this case the destinations happens to be whatever C variables are
;pointed at by _portptr and _fireptr.
;

        movea.l     _portptr,a1; a1 now holds the destination
        move.b  _ciaaprb,(a1); move byte from port to dest

        movea.l     _fireptr,a1; a1 now holds the destination
        move.b  _ciabpra,(a1); move byte from port to dest

        rts

        end
```