0.1 User Inputs

In an era before Microsoft harnessed all inputs under DirectInput API with Windows 95, developers had to write drivers for each input type they wanted to support. This involved talking directly to the hardware in the vendor's protocol on a physical port. The keyboard is plugged into a PS/2 or AT port, the mouse to a serial port (DE9), and the joystick to a game port (DA-15).

0.1.1 Keyboard

As the keyboard is the standard and oldest input medium, it is fairly easy to access. When a key is pressed, the interrupt is routed to an ISR in the Vector Interrupt Table. The engine installs its own ISR there.

```
#define KeyInt    9 // The keyboard ISR number

static void INL_StartKbd(void) {

  IN_ClearKeysDown();

  OldKeyVect = getvect(KeyInt);
  setvect(KeyInt,INL_KeyService);

  INL_KeyHook = 0;  // Clear key hook
}

static void interrupt INL_KeyService(void) {
  byte  k;
  k = inportb(0x60);  // Get the scan code

  // Tell the XT keyboard controller to clear the key
  outportb(0x61,(temp = inportb(0x61)) | 0x80);
  outportb(0x61,temp);

  [...] // Process scan code.
  Keyboard[k] = XXX;

  outportb(0x20,0x20); // ACK interrupt to interrupt system
    .
}
```

The state of the keyboard is maintained in a global array `Keyboard`, available for the entire engine to lookup.

```
#define NumCodes  128
boolean   Keyboard[NumCodes];
```

0.1.2 Mouse

A driver has to be loaded at startup for the mouse to be accessible. DOS did not come with one. It was usually on a vendor provided floppy disk. `MOUSE.COM` (or `MOUSE.SYS`) had to be added to `config.sys` so it would reside in RAM. It was usually stored in `DOS` folder.

```
C:\DOS\MOUSE.COM
```

The driver takes almost 5KiB of RAM. With the driver loaded all interactions happen with software interrupt `0x33`. The interface works with requests issued in register AX[1] and responses issued in registers CX, BX and DX. With Borland compiler syntactic sugar it is easy to write with almost no boilerplate (notice direct access to registers thanks to _AX and co special keywords).

```
#define MouseInt  0x33
#define Mouse(x)  _AX = x,geninterrupt(MouseInt)

static void INL_GetMouseDelta(int *x,int *y) {
  Mouse(MDelta);
  *x = _CX;
  *y = _DX;
}
```

[1] For a full overview of all mouse interrupt function, see https://www.stanislavs.org/helppc/int_33.html.

Request	**Type**	**Response**
AX=0	Get Status	AX = FFFFh : available. AX Value = 0 : not available
AX=1	Show Pointer	
AX=2	Hide Pointer	
AX=3	Mouse Position	CX = X Coordinate, DX = Y Coordinate
AX=3	Mouse Buttons	BX = 1 Left Pressed, BX = 2 Right Pressed, BX = 3 Center Button Pressed
AX=7	Set Horizontal Limit	CX=MaxX1 DX=MaxX2
AX=8	Set Vertical Limit	CX=MaxY1 DX=MaxY2
AX=11	Read Mouse Motion Counters	CX = horizontal mickey count[2], DX = vertical mickey count

Figure 1: Mouse request/response.

0.1.3 Joystick

All interactions with the joystick happen over I/O port `0x201`. Two joysticks can be chained together and the state of both of them fits in a byte.

```
word INL_GetJoyButtons(word joy){
  register word result;

  result = inportb(0x201); // Get all the joystick buttons
  result >>= joy? 6 : 4;   // Shift into bits 0-1
  result &= 3;             // Mask off the useless bits
  result ^= 3;
  return(result);
}
```

[2]values are 1/200 inch intervals (1 mickey = 1/200 in.).

Bit Number	Meaning
0	Joystick A, X Axis
1	Joystick A, Y Axis
2	Joystick B, X Axis
3	Joystick B, Y Axis
4	Joystick A, Button 1
5	Joystick A, Button 2
6	Joystick B, Button 1
7	Joystick B, Button 2

Figure 2: Joystick sampling bits and their meaning.

The API looks clean at first, with each button associated with a bit indicating whether it is pressed or not. But if you take a closer look you will notice there is only one bit of information per axis, which is not enough to encode the position of a stick. This bit is actually a flag allowing an analog input to be converted into a digital value. To better understand, let's dive into details.

On the joystick side, each axis is connected to a 100kΩ potentiometer. An applied 5V voltage generates a variable current based on the stick position (from Ohm's law where $I = \frac{V}{R}$).

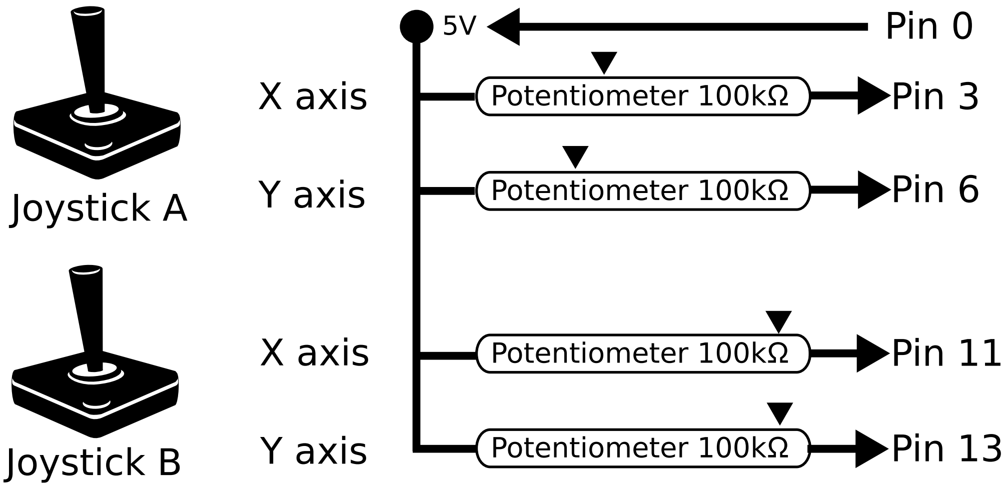


Figure 3: Two joysticks and the four potentiometers connected to the game port pins.

On the joystick side each pin carrying the current is connected to monostable multivibrators (which is a complicated name for a capacitor able to output 1 when it is charged and

0 when it is charging). The idea is to infer the position of the stick by measuring how long the vibrator takes to charge (a strong current will charge the capacitor faster than a weak current).

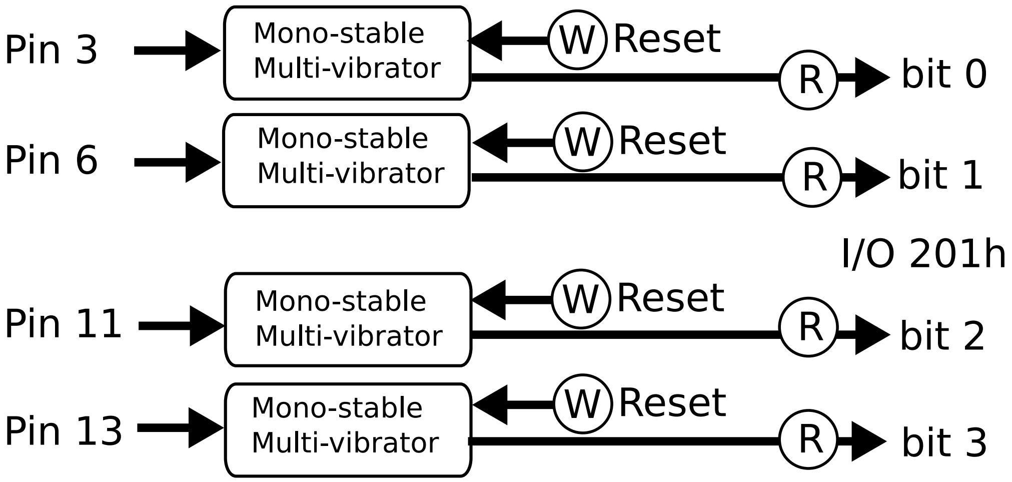

Pin 3
Mono-stable Multi-vibrator
W Reset
R bit 0
Pin 6
Mono-stable Multi-vibrator
W Reset
R bit 1
I/O 201h
Pin 11
Mono-stable Multi-vibrator
W Reset
R bit 2
Pin 13
Mono-stable Multi-vibrator
W Reset
R bit 3

Figure 4: Each potentiometer is connected to a capacitor able to output either 0 or 1 depending on its charging state.

On the CPU side, retrieving the stick position is a three-step process:

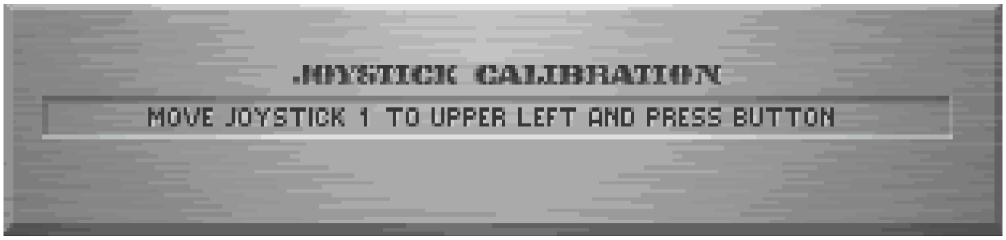
1. Write Ⓦ any value to I/O port `201h`. This will discharge all capacitors.

2. Initialize a counter to zero and read Ⓡ from `201h`. At first all bits 0-4 will be equal to zero.

3. Loop forever (or until counter == 0xFFFF as a safety measure) increase counter on each iteration. Save the counter value for each bit when it is flipped to 1.

On a 286 CPU the counter value can range from 7 to 900 depending on the stick/capacitor position. On a 386 CPU, which will run loops faster, these values would be higher. Hence the values measured can only be translated to a stick position if they are compared to a min and a max.

This explains why joysticks have to be calibrated. For the flight simulators of the 90s where accurate position was needed, the player would be asked to put the joystick in upper-left position (to set the potentiometers on both axis to minimum resistance) and press a button to read the "loop count". The player would then repeat the operation at the lower right position so that the system would know the min and max "loop count" for this joystick/CPU combination.[3]


JOYSTICK CALIBRATION

MOVE JOYSTICK 1 TO UPPER LEFT AND PRESS BUTTON

Figure 5: Strike Commander startup screen makes you calibrate your joystick.

There is no calibration process in Wolfenstein 3D because when the engine starts up it samples the loop count and assumes the joystick is in neutral position. When the game runs and joystick position is needed, the engine samples loop count and compares the count to what was measured with neutral. It is not enough to calculate the exact stick position on each axis but it is enough to determine up/down and left/right using >, == (with epsilon) and < comparison operations.

[3]The Mark-1 FCS by Thrustmaster and Flightstick Pro by CH were the best flight controllers of the 90s. They used all bits for one controller, offering a device with four buttons with the extra two axes serving as a four-way view hat.