

# Chapter 37 - Keyboard, Mouse, and Time of Day

---

Input arrives through the terminal register block at \$F0700-\$F07FF. BASIC usually reads characters with GET and INPUT; games, editors, and music tools can also poll the raw registers directly.

All registers in this chapter are 32-bit on the bus. Most values use only the low byte. Mouse X and Y use the low 16 bits. Relative mouse movement uses signed 32-bit values.

## 37.1 Register Map

Address	Name	R/W	Meaning
\$F0728	TERM_KEY_IN	R	Read one raw key byte and advance the key queue.
\$F072C	TERM_KEY_STATUS	R	Bit 0 set when a raw key byte is queued.
\$F0740	SCAN_CODE	R	Read one physical-key scancode and advance the scancode queue.
\$F0744	SCAN_STATUS	R	Bit 0 set when a scancode is queued.
\$F0748	SCAN_MODIFIERS	R	Bit 0 shift, bit 1 ctrl, bit 2 alt, bit 3 capslock.
\$F0730	MOUSE_X	R	Absolute mouse X, low 16 bits.
\$F0734	MOUSE_Y	R	Absolute mouse Y, low 16 bits.
\$F0738	MOUSE_BUTTONS	R	Bit 0 left, bit 1 right, bit 2 middle.
\$F073C	MOUSE_STATUS	R	Bit 0 set after a mouse change; reading clears it.
\$F074C	MOUSE_CTRL	R/W	Bit 0 requests relative captured mouse mode.
\$F0750	RTC_EPOCH	R	Seconds since 1970-01-01 00:00:00 UTC.
\$F0754	MOUSE_DX	R	Signed accumulated relative X movement; reading clears it.
\$F0758	MOUSE_DY	R	Signed accumulated relative Y movement; reading clears it.
\$F075C	RTC_MONO_USEC_LO	R	Low 32 bits of monotonic microseconds since engine start.
\$F0760	RTC_MONO_USEC_HI	R	High 32 bits of monotonic microseconds since engine start.

## 37.2 Raw Key Queue

The raw key queue contains one byte per terminal key after the graphical input path has delivered it to the MMIO block. It is not line-buffered and it is not echoed. Use it when a program wants immediate key presses but still wants the usual character values.

```
10 REM WAIT FOR ONE RAW KEY BYTE
20 PRINT "PRESS A KEY"
30 IF PEEK32(&H000F072C)=0 THEN GOTO 30
40 K=PEEK32(&H000F0728)
50 PRINT "KEY ";K
```

If you press A, the final line prints KEY 65. Reading \$F0728 when the queue is empty returns 0, so check \$F072C first when zero is a meaningful key value for your program. Line 30 is the guard. Line 40 consumes the byte, so reading it again would move on to the next queued key.

## 37.3 Physical Scancodes

The scancode queue reports physical key events. A press and release both appear in the queue. The high bit marks release, so 30 and 158 are the press and release forms of the same key code.

```
10 REM SHOW PRESS AND RELEASE SCANCODES
20 PRINT "PRESS AND RELEASE A KEY"
30 IF PEEK32(&H000F0744)=0 THEN GOTO 30
40 C=PEEK32(&H000F0740)
50 M=PEEK32(&H000F0748)
60 PRINT "SCAN ";C;" MOD ";M
70 IF (C AND 128)=0 THEN GOTO 30
```

The modifier byte is not queued. It reports the modifier state at the moment it is read. Line 70 keeps the loop alive until the release form of the scancode appears. That release form is the press code plus 128.

## 37.4 Absolute Mouse

Absolute mouse mode reports the current pointer position and button state. MOUSE\_STATUS is a one-shot changed bit: it reads 1 after a change and clears to 0 when read.

```
10 REM READ STATUS FIRST, BECAUSE IT CLEARS ON READ
20 S=PEEK32(&H000F073C)
30 X=PEEK32(&H000F0730)
40 Y=PEEK32(&H000F0734)
50 B=PEEK32(&H000F0738)
60 PRINT "MOUSE ";X;Y;B;" CHANGED ";S
```

The button value is a bit field. Left is 1, right is 2, middle is 4, and combined buttons add those values. Read \$F073C once per sample. A second read before the next mouse event returns 0.

## 37.5 Relative Mouse

Relative mode is for games and editors that care about movement rather than pointer position. Write 1 to \$F074C to request captured relative mode. Write 0 to return to normal absolute mode.

```
10 REM CAPTURE RELATIVE MOVEMENT UNTIL A KEY IS PRESSED
20 POKE32 &H000F074C,1
30 PRINT "MOVE MOUSE, THEN PRESS A KEY"
40 IF PEEK32(&H000F072C)=0 THEN GOTO 40
50 K=PEEK32(&H000F0728)
60 DX=PEEK32(&H000F0754):DY=PEEK32(&H000F0758)
70 PRINT "DELTA ";DX;DY;" KEY ";K
80 POKE32 &H000F074C,0
```

MOUSE\_DX and MOUSE\_DY clear independently when read. Poll once per frame if you want frame-by-frame movement. Negative movement is reported as a signed 32-bit value. If BASIC prints a value greater than 2147483647, subtract

4294967296 to view it as a negative number.

## 37.6 Time Of Day

---

RTC\_EPOCH reads whole seconds since 1970-01-01 00:00:00 UTC. This is wall-clock time. It can jump if the clock is changed outside the machine, so use it for dates, not for measuring short intervals.

```
10 T=PEEK32(&H000F0750)
20 PRINT "SECONDS ";T
```

Two reads about a second apart normally differ by one. For shorter elapsed-time measurements, use the monotonic microsecond registers in the next section, a CPU timer, or a device status bit.

The value is a signed 32-bit seconds counter. In 2038 it crosses from positive to negative, then keeps counting in signed arithmetic.

## 37.7 Monotonic Elapsed Time

---

RTC\_MON0\_USEC\_L0 and RTC\_MON0\_USEC\_HI form a 64-bit microsecond counter since Intuition Engine started. It is monotonic: it is for intervals and timeouts, not for calendar time.

Read the high word, then the low word, then the high word again. If the two high reads differ, the low word crossed \$FFFFFFFF while you were reading and you should try again.

```
10 REM READ MONOTONIC MICROSECOND TIMER
20 H1=PEEK32(&H000F0760)
30 L=PEEK32(&H000F075C)
40 H2=PEEK32(&H000F0760)
50 IF H1<>H2 THEN GOTO 20
60 PRINT "USEC ";H2;L
```

On a short run the high word is usually 0, and the low word is the elapsed microsecond count. For long-running programs, keep the value as two words. Even though BASIC uses double-precision numbers, not every 64-bit integer can be represented as one exact decimal value.

## 37.8 Small-CPU Access

---

The 6502 and Z80 reach these registers through their terminal and MMIO apertures described in Chapters 27 and 28. A 32-bit register appears as four byte lanes. For most registers the useful bits are in the low byte. For MOUSE\_X and MOUSE\_Y, read the low two bytes for the 16-bit coordinate.

The cooked-key register at \$F0728 shares its queue with terminal character input. A program should choose one consumer for that queue: BASIC GET, BASIC INPUT, terminal reads, or direct reads from \$F0728.