

# Chapter 31 - Processor Timing, Traps, and Exceptions

The six processors of Part IV each have their own trap and exception machinery, but they all share one bus, one clock, and one shared set of interrupt sources. This chapter describes that common ground: how time is counted, how each CPU receives an interrupt, how exceptions are raised, and how the IRQ diagnostics block at \$F23C0 makes all of it visible.

## 31.1 The shared clock

Intuition Engine has a single master clock. Each processor draws from it at its own rate; instructions on the smaller CPUs cost a few cycles each, while IE64 retires one instruction per decoded step in steady state. The compositor, audio mixer, and device timing paths derive their own clocks from the machine, independently of which CPU is active. The IE64 control-register timer in section 31.2 counts decoded IE64 instruction steps.

For BASIC programs there is no need to count cycles directly: the `WAIT` statement, the audio engines' own clocks, and the video chip's VBlank flag are usually enough. For machine code that wants precise control, each per-CPU chapter lists the published cycle counts for its instruction set. Use those chapter tables and Appendix G as the Intuition Engine reference.

## 31.2 Timer

IE64 has a decoded-instruction-step timer in its control-register bank:

CR index	Name	Purpose
9	TIMER_PERIOD	Reload value in timer steps
10	TIMER_COUNT	Current timer-step countdown
11	TIMER_CTRL	bit 0 enable, bit 1 IRQ enable

When the timer is enabled, `TIMER_COUNT` is decremented once before each decoded IE64 instruction step. When the count reaches zero, it reloads from `TIMER_PERIOD`. If `TIMER_CTRL` has both enable bits set, the reload raises the IE64 timer interrupt through `CR_INTR_VEC`.

The legacy `TIMER_*` MMIO names are reserved compatibility symbols, not the timer interface. The \$F0800 block belongs to the SoundChip. Programs on the other CPUs should use `WAIT`, VBlank/status polling, device interrupts, or a chip-specific clock source instead of \$F0804 and \$F0808.

### 31.2.1 BASIC polling example

`WAIT` is the native BASIC timing primitive for memory-visible events. The test is `((PEEK(addr) EOR xor) AND mask) <> 0`; omit `xor` when the comparison value is zero.

Type this listing:

```

10 REM CHOOSE A WORD OF SHARED RAM
20 A=&H00050000
30 REM CLEAR IT, THEN SHOW THE STARTING VALUE
40 POKE A,0
50 PRINT "BEFORE ";PEEK(A)
60 REM SET BIT 2 AND WAIT FOR THAT BIT
70 POKE A,4
80 WAIT A,4
90 PRINT "AFTER ";PEEK(A)

```

Line 20 chooses an ordinary shared-memory word. Lines 40 and 50 prove the word starts clear. Line 70 sets bit 2, and line 80 waits for that same bit. The program prints BEFORE 0 and AFTER 4. Replace line 70 with a device action when you want to wait for a VBlank flag, a blitter done bit, or another hardware status bit.

Do not use one-argument WAIT as a delay. BASIC WAIT always needs an address and a mask because it polls a memory-visible event. For a plain delay between two sound or video writes, use the device's own status bit, VSYNC where it is available, or a small FOR . . . NEXT loop.

## 31.3 Interrupt sources

---

There are four canonical hardware interrupt sources:

- **VBlank** - raised by VideoChip at the start of each vertical blanking interval. Mapped to M68K level 5.
- **Copper / raster** - raised by the copper-list engine when a WAIT matches the current raster position. Mapped to M68K level 5.
- **Audio** - raised by any audio engine that has reached the end of a loop or sample. Mapped to M68K level 4.
- **Timer** - raised when the IE64 timer-step counter reloads.

For the heritage CPUs the system collapses these into one or two lines:

- 6502: IRQ (maskable) and NMI (non-maskable). Audio uses IRQ; VBlank uses NMI on programs that need it.
- Z80: INT (maskable) and NMI. The interrupt-mode setting (IM 0/IM 1/IM 2) chooses how the vector is formed.
- M68K: seven prioritised levels. IE drives L4 for audio and L5 for video, both as auto-vectored interrupts. Level 7 is non-maskable; IE does not raise it.
- x86: a single INTR line plus NMI (vector 2). The vector number is supplied by the interrupting device.
- IE32: a single interrupt vector at memory \$0004.
- IE64: the trap vector in CR\_TRAP\_VEC, or the timer-specific CR\_INTR\_VEC.

## 31.4 Exceptions versus interrupts

---

Across all six CPUs the distinction is the same:

- **Interrupt** - caused by an external event (a device finishing a frame, a timer expiring). Maskable in general, except for NMI and M68K level 7.
- **Exception** - caused by the executing instruction itself (divide by zero, address error, illegal opcode, privilege violation, breakpoint, syscall).

Both end the current instruction and jump to a vector through the same mechanism. The CPU's view of "what just happened" is the distinction.

## 31.5 Trap and exception causes

The five canonical trap causes that every CPU can raise:

Cause	6502	Z80	M68K vector	x86 vector	IE32 / IE64
Reset	\$FFFC/\$FFFD	RESET pin	0/1	EIP=0	\$0000
Divide by zero	(none)	(none)	5	0	IE32 stops with a division-by-zero error; IE64 integer divide/modulo by zero writes 0 and does not trap
Illegal opcode	(none)	(none)	4	6	IE64 FAULT_ILLEGAL_INSTRUCTION
Privilege violation	(none)	(none)	8	13	IE64 FAULT_PRIV
Software trap	BRK	RST n	TRAP #n	INT n	IE64 SYSCALL
Breakpoint	(none)	(none)	(debugger)	INT 3	(debugger)
Bus / address error	(none)	(none)	2 / 3	(none)	(none)
Trace / single-step	(none)	(none)	9	TF flag	IE64 FAULT_TRACE (not implemented)

The smaller CPUs simply do not have the silicon to detect some of these; an illegal opcode on the 6502, for example, just executes the corresponding undocumented operation. See Chapter 27, section 26.6.

IE64 floating-point divide-by-zero is an FPU status condition. Read the FPU status register described in Chapter 25 when a program needs to distinguish it from an ordinary finite result.

## 31.6 The IRQ diagnostics block

The 32-byte block at \$F23C0 reports interrupt activity in real time. Programs do not normally need it; it is here for debugging stuck handlers.

Address	Name	Reports
\$F23C0	IRQ_DIAG_ISR	Interrupt-in-service bitmask
\$F23C4	IRQ_DIAG_FLAGS	b0 stopped, b1 in-exception, b2 INTENA, b3 running
\$F23C8	IRQ_DIAG_PENDING	Pending-interrupt bitmask
\$F23CC	IRQ_DIAG_COUNTERS	L5 delivered (lo16) + L4 delivered (hi16)
\$F23D0	IRQ_DIAG_BLOCKED	L5 blocked (lo16) + L4 blocked (hi16)
\$F23D4	IRQ_DIAG_RTE	RTE count
\$F23D8	IRQ_DIAG_STOP_SPINS	Consecutive STOP iterations without wake
\$F23DC	IRQ_DIAG_WATCHDOG	Latched watchdog event count

The counters are most useful when the M68K is the active CPU. A non-zero IRQ\_DIAG\_BLOCKED value plus a stable STOP\_SPINS counter is a classic sign of an interrupt-mask leak: the device raised the IRQ, the CPU was running with the mask too high, and the IRQ was deferred indefinitely.

## 31.7 Reset

---

Reset is the universal exception: every CPU has one, and it always runs to completion before any other code executes.

- **IE64** - PC = \$0000, all GPRs zeroed, MMU disabled, mode = supervisor. The low-RAM entry convention is JMP PROG\_START.
- **IE32** - PC = \$0000, SP = STACK\_START (\$9F000), interrupts disabled. Convention is JMP \$1000.
- **6502** - PC is loaded from \$FFFC/\$FFFD, I flag set.
- **Z80** - PC = \$0000, interrupt mode set to 0, IFF1 and IFF2 cleared.
- **M68K** - SSP loaded from vector 0, PC loaded from vector 1. Supervisor mode entered. I2:I1:I0 = 7 (all interrupts masked).
- **x86** - EIP = \$00000000, all segments cleared to 0, IF = 1.

On Intuition Engine the selected CPU is reset after memory has been prepared for the chosen profile. The first instruction seen by the CPU is therefore the profile's reset entry: low RAM for IE64 and IE32 examples, the 6502 reset vector, vectors 0 and 1 on M68K, or EIP = 0 for x86. Chapter 32 covers cross-CPU cooperation.

## 31.8 Timing on the heritage CPUs

---

The heritage chips have published cycle counts for every instruction in this guide. Their schedulers account for those CPU cycles against the shared master clock, so an instruction listed as 4 cycles consumes four heritage-CPU cycle units. This is separate from the IE64 control-register timer in section 31.2, which counts decoded IE64 instruction steps.

The smaller CPUs run at the speed of the master clock divided by a per-CPU scaling factor: the 6502 and Z80 each get roughly the same instruction throughput as a real 4-MHz part on a busy period, but the master clock continues at the full IE rate so a 6502 instruction does not stretch the VBlank interval. Programs that need very precise timing should use a device clock, VBlank, or the copper-list rather than counting instructions.

IE64's timer-step countdown is reported in CR10 (the timer count) and the CPU decrements it once per decoded instruction step. The published latencies in Chapter 25 are minimum bounds; modern hardware may retire instructions faster in straight-line code.

## 31.9 What comes next

---

Chapter 32 covers how programs running on different CPUs cooperate through the coprocessor and cross-CPU calling mechanism, plus the COSTART/COSTOP/COWAIT BASIC verbs that drive it. After that, Chapters 33 and 34 cover the two interactive debugging surfaces: the Machine Monitor and IE Script.