

Chapter 24 - Memory Model and MMIO Map

Part IV of this guide is the bridge between BASIC and machine language. From here on, every chapter assumes you want to read and write the Intuition Engine's hardware directly: from BASIC with PEEK and POKE, from the machine monitor, or from a program written in one of the six processor languages described in Chapters 26 through 30.

Everything you do at that level happens through one shared physical bus. This chapter is the map of the low fixed regions of that bus, and the rules for CPUs that see narrower address spaces.

24.1 The address space

Intuition Engine has one 64-bit physical bus. It carries 64-bit physical addresses and accepts 8, 16, 32, and 64-bit transfers. The classic chips and helper devices are not separate computers; they are devices and adapters on that bus.

The low 4 gigabytes, \$00000000 to \$FFFFFFFF, form the legacy CPU and device window used by BASIC examples, MMIO registers, video apertures, and the compatibility processors. A `MOVE.L D0, ($F0700).L` from M68K, a `STORE.L (R5), R6` from IE64 with `R5 = $F0700`, and a BASIC `POKE &H000F0700, n` all arrive at the same byte in that low window.

IE64 can address the full 64-bit physical range. Its MMU uses 64-bit virtual addresses and page-table entries with a 52-bit physical page number field, giving 64-bit physical addresses after the 4 KB page offset is added. Above the legacy low-memory slice, RAM is supplied by the machine's guest-RAM backing and discovered through SysInfo and `CR_RAM_SIZE_BYTES`.

This is not how every processor sees the world. IE32, M68K, and x86 have 32-bit address registers and see the low 4 GB window directly. The 6502 and Z80 can address only 64 kilobytes directly. Each of those CPUs has a small set of apertures into the larger bus; the apertures are described in the per-CPU chapters.

24.1.1 Three zones

The fixed low window divides into three zones:

Range	Size	Contents
\$00000000 - \$0009EFFF	636K	Main RAM (low memory, code and data)
\$0009F000 - \$000FFFFF	388K	Stack, VRAM apertures and MMIO
\$00100000 - \$005FFFFF	5M	Main VRAM
\$00600000 - \$FFFEFFFF	Dynamic	Extended RAM (sized at boot)
\$FFFF0000 - \$FFFFFFFF	64K	Sign-extended alias of \$00000000-\$0000FFFF

The 5-megabyte VRAM block from \$100000 to \$5FFFFFF is wired directly to the VideoChip framebuffer (Chapter 4). Writes here appear on screen the next time the compositor refreshes.

Memory beyond \$600000 is plain RAM when it is backed by the current guest-RAM allocation. Its upper bound depends on the amount of RAM the Intuition Engine was started with and on the active CPU profile. The two words at \$F2400/\$F2404 (`SYSDINFO_TOTAL_RAM_L0/HI`) report the byte size of total guest RAM as a 64-bit value; \$F2408/\$F240C

report the size visible to the currently executing CPU profile. \$F2414/\$F2418 report the dense low RAM window: the contiguous low-memory span that 32-bit clients, flat images, and low-window device buffers can address directly.

IE64 BASIC keeps its own programme text, variables, stacks, file bridges, line/input scratch, and compiler scratch areas in dynamic reservations inside the same address space. In the normal low32 fallback layout the line/input scratch begins at \$01000000, and the programme, variable, string, and file-bridge arena begins after that scratch reservation. The current pointer and capacity are BASIC state, not a fixed public line-buffer address.

Reader programmes should not depend on those private reservations. When BASIC needs a public buffer for a copper list, coprocessor request, file staging area, or DMA-style hardware block, use `MEMALLOC(size[, align])`. It allocates from public low32 ranges that are meant to be shared with devices and other CPUs.

The low memory slice and the backed RAM above it are both active RAM, but the boundary between them matters for scalar accesses. Byte reads and writes work on either side. Word and long reads and writes must fit wholly inside the low slice or wholly inside backed RAM. If a word or long starts before the boundary and ends after it, the access is unmapped: a read returns zero and a write does not partly update either side.

Byte-copy devices can still cross the same boundary one byte at a time. The File I/O block in Chapter 35 uses that rule when it copies a file into a destination buffer. Device MMIO is still MMIO, not RAM, even when its address lies inside the low 32-bit window.

24.1.2 The sign-extended alias

The top 64 kilobytes (\$FFFF0000 - \$FFFFFFFF) are not real memory of their own. They are a mirror of the bottom 64 kilobytes: address \$FFFFFF00 reads the same byte as \$0000FF00.

This alias exists for the 68K. A pre-decrement from a register holding zero (`-(SP)` with `SP = 0`) wraps to \$FFFFFFFC, which must still hit the stack. The alias makes that work.

24.2 Main RAM

Main RAM begins at \$00000000 and runs up to the bottom of the I/O region.

\$00000000	Vector table	(interrupt vectors and reserved slots)
\$00001000	Program code	(PROG_START)
\$00002000	Stack region	(grows downward from STACK_START)
\$0009F000	Top of stack	(STACK_START - initial SP for IE32)
\$0009FFFF	End of low RAM	

The vector table layout depends on which CPU is active. The 68K follows the standard 68000 vector table (Chapter 29). The 6502 keeps its NMI/RESET/IRQ vectors in the top six bytes of the 16-bit page that maps to \$0000FFFA-\$0000FFFF. IE64 and IE32 use their own vector formats (Chapters 25 and 26).

PROG_START at \$1000 is the conventional low-RAM entry point for monitor-entered and loaded machine-code programs. Programs are free to extend past \$1000 upward and to use the stack region below it for data, but the floor at \$2000 is the conventional ceiling for the stack and the floor for static data. Programs that need more stack than 637 kilobytes can simply place SP higher in extended RAM.

24.3 The MMIO region

Everything between \$F0000 and \$FFFFF, together with the two VGA legacy VRAM apertures at \$A0000 and \$B8000 and the ULA VRAM aperture at \$FA000, is memory-mapped I/O: the registers of the six picture chips, the ten audio engines,

the file system bridge, the coprocessor, and various small control surfaces. A read or write to an MMIO address does not touch RAM; it asks the corresponding device to do something.

The full map:

Range	Size	Device
\$A0000-\$AFFFF	64K	VGA graphics VRAM (mode \$12/\$13/\$14)
\$B8000-\$BFFFF	32K	VGA text VRAM
\$F0000-\$F049B	1180B	VideoChip + palette + extended blitter
\$F0700-\$F07FF	256B	Terminal, mouse, keyboard, RTC
\$F0800-\$F0B7F	896B	SoundChip (core six engines)
\$F0BA0-\$F0BBF	32B	MIDI/MUS player
\$F0BC0-\$F0BD7	24B	MOD player
\$F0BD8-\$F0BF3	28B	WAV sample player
\$F0C00-\$F0C20	33B	PSG (AY-3-8910/YM2149)
\$F0C40-\$F0CFF	192B	SoundChip flex channels 4-6
\$F0D00-\$F0D20	33B	POKEY
\$F0D40-\$F0DFF	192B	SoundChip flex channels 7-9
\$F0E00-\$F0E2D	46B	SID (6581/8580)
\$F0E80-\$F0EFF	128B	SFX trigger legacy aliases, channels 0-3
\$D0000-\$DFFFF	64K	Voodoo texture memory
\$F0F00-\$F0F1F	32B	TED audio
\$F0F20-\$F0F6B	76B	TED video
\$F1000-\$F13FF	1K	VGA registers
\$F1400-\$F140F	16B	System helper MMIO
\$F2000-\$F2017	24B	ULA registers
\$F2100-\$F213F	64B	ANTIC
\$F2140-\$F21FB	188B	GTIA
\$F2200-\$F221F	32B	File I/O
\$F2260-\$F22AF	80B	Paula DMA (audio)
\$F2300-\$F231F	32B	Media loader
\$F2320-\$F233F	32B	RUN loader block
\$F2340-\$F238F	80B	Coprocessor control
\$F2390-\$F23AF	32B	Clipboard bridge
\$F23B0-\$F23BF	16B	Coprocessor extended monitor
\$F23C0-\$F23DF	32B	IRQ diagnostics
\$F23E0-\$F23FF	32B	Bootstrap file bridge

Range	Size	Device
\$F2400-\$F24FF	256B	SysInfo (RAM-size discovery)
\$F2600-\$F29FF	1K	SFX trigger extended window, channels 0-31
\$F8000-\$F87FF	2K	Voodoo 3D registers
\$FA000-\$FBFFF	6912B	ULA VRAM (bitmap + attrs)

A few small ranges between these blocks read as zero on read and ignore on write. They are reserved for future devices.

24.3.1 Width and alignment

MMIO width depends on the device.

Width	BASIC form	Typical use
8 bits	POKE8 / PEEK8	SID, TED audio, POKEY, PSG, SN76489 ports, palette bytes, character memory.
16 bits	POKE16 / PEEK16	Some CPU-friendly aliases and split register writes.
32 bits	POKE32 / PEEK32	Pointers, lengths, control words, status words, framebuffer bases, blitter registers.
64 bits	POKE64 / PEEK64 or CPU doubleword stores where documented	Native 64-bit memory blocks and split high/low register pairs.

Do not widen a byte register just because the address is in the MMIO region. POKEY, TED audio, SID, PSG, and many VGA registers are byte-oriented. Use the owning chapter's table.

PEEK16/POKE16, PEEK32/POKE32, and PEEK64/POKE64 require 2-, 4-, and 8-byte aligned addresses. PEEK, POKE, PEEK8, and POKE8 are byte-width forms and accept any byte address.

Multi-byte player registers use little-endian byte order when you write their individual bytes. For example, a pointer value \$00100000 staged through byte writes goes low byte first:

```

10 REM STAGE MOD POINTER $00100000
20 POKE8 &H000F0BC0,0
30 POKE8 &H000F0BC1,0
40 POKE8 &H000F0BC2,16
50 POKE8 &H000F0BC3,0
60 PRINT PEEK8(&H000F0BC0),PEEK8(&H000F0BC1)
70 PRINT PEEK8(&H000F0BC2),PEEK8(&H000F0BC3)

```

A 32-bit POKE32 &H000F0BC0,&H00100000 writes the same staged pointer in one operation.

Lines 20 to 50 write the four bytes of the pointer from least significant byte to most significant byte. Lines 60 and 70 read those bytes back in two print zones, so the expected values are 0, 0, then 16, 0.

64-bit accesses to a legacy MMIO region are not the normal reader workflow. Use the documented 32-bit high/low registers, such as SYSINFO_TOTAL_RAM_LO and SYSINFO_TOTAL_RAM_HI, unless a chip chapter explicitly documents a 64-bit register.

24.4 The terminal region

\$F0700-\$F07FF holds the basic human-facing peripherals: the text terminal, mouse, keyboard scancodes, and a real-time clock.

Address	Name	Description
\$F0700	TERM_OUT	Write a byte: emit to terminal
\$F0704	TERM_STATUS	b0 input ready, b1 output ready
\$F0708	TERM_IN	Read next input byte (dequeues)
\$F070C	TERM_LINE_STATUS	b0 complete line available
\$F0710	TERM_ECHO	b0 local echo enable (default 1)
\$F0724	TERM_CTRL	b0 line input mode
\$F0728	TERM_KEY_IN	Read next raw key byte
\$F072C	TERM_KEY_STATUS	b0 raw key available
\$F0730	MOUSE_X	Absolute X position
\$F0734	MOUSE_Y	Absolute Y position
\$F0738	MOUSE_BUTTONS	b0 left, b1 right, b2 middle
\$F073C	MOUSE_STATUS	b0 changed-since-last-read
\$F074C	MOUSE_CTRL	b0 request relative/captured mode
\$F0740	SCAN_CODE	Dequeue raw keyboard scancode
\$F0744	SCAN_STATUS	b0 scancode available
\$F0748	SCAN_MODIFIERS	b0 shift, b1 ctrl, b2 alt, b3 caps
\$F0750	RTC_EPOCH	UTC seconds since 1970-01-01
\$F0754	MOUSE_DX	Signed relative X delta (clears)
\$F0758	MOUSE_DY	Signed relative Y delta (clears)
\$F075C	RTC_MONO_USEC_LO	Low 32 bits of monotonic microseconds since engine start
\$F0760	RTC_MONO_USEC_HI	High 32 bits of monotonic microseconds since engine start
\$F07F0	TERM_SENTINEL	Write \$DEAD to halt the CPU

The M68K reaches TERM_OUT through a second alias as well:

- TERM_OUT_SIGNEXT = \$FFFF700: the sign-extended .W form that an M68K MOVE.B D0, (\$F700).W resolves to. The bus folds this back onto the same \$F0700 register through the sign-extended mirror described in section 24.1.2.

The 6502 and Z80 have no equivalent 16-bit alias for TERM_OUT: the only 16-bit address that would translate to \$F0700 is \$F700, which the CPU adapters intercept as BANK1_REG_LO before the bus translation runs. Code that needs the system terminal can select TERM_IO_BANK into bank 1 with SET_TERMINAL_BANK, then use the 6502/Z80 include-file terminal aliases at \$2700-\$27FF; TERM_OUT is \$2700 in that window and resolves to bus \$F0700. See Chapters 27 and 28 for the per-CPU details.

Chapter 37 covers the keyboard, mouse, and controller MMIO in detail. Chapter 38 covers the serial interface that overlays TERM_OUT/TERM_IN.

24.5 The system-information block

\$F2400-\$F24FF. Six read-only words let a program discover how much memory it has to play with:

Address	Name	Description
\$F2400	SYSINFO_TOTAL_RAM_LO	Low 32 bits of total RAM, bytes
\$F2404	SYSINFO_TOTAL_RAM_HI	High 32 bits of total RAM
\$F2408	SYSINFO_ACTIVE_RAM_LO	Low 32 bits of RAM visible to the active CPU
\$F240C	SYSINFO_ACTIVE_RAM_HI	High 32 bits of CPU-visible RAM
\$F2414	SYSINFO_LOW_WINDOW_LO	Low 32 bits of the dense low RAM window
\$F2418	SYSINFO_LOW_WINDOW_HI	High 32 bits of the dense low RAM window

The total and active values can differ when a 16-bit profile (6502 or Z80) is the active CPU: total reports the physical RAM, active reports the window the small CPU can currently see. The low-window value is different again. It tells software how much contiguous low RAM is backed by the ordinary low address path. Use it when a buffer must be reachable by a 32-bit image, a compatibility CPU, or a device register that stores a low-window pointer.

Type this to print both low words. On machines with more than 4 GB, the high words at lines 30 and 50 are non-zero.

```
10 REM SYSINFO RAM SIZE WORDS
20 TL=PEEK32(&H000F2400)
30 TH=PEEK32(&H000F2404)
40 AL=PEEK32(&H000F2408)
50 AH=PEEK32(&H000F240C)
60 PRINT "TOTAL LO/HI ";TL,TH
70 PRINT "ACTIVE LO/HI ";AL,AH
```

Writes to the SysInfo block are ignored. It is a discovery block, not a configuration block.

The low word is enough on ordinary small configurations. The high word is there so the same code can describe a larger memory configuration without changing the ABI.

24.6 The bootstrap file bridge

\$F23E0-\$F23FF. This eight-register block exposes the file system to bootstrap code that runs before BASIC is up.

Offset	Name	Direction	Description
+0	CMD	W	Command code, 0-7
+4	ARG1	W	First argument
+8	ARG2	W	Second argument
+12	ARG3	W	Third argument
+16	ARG4	W	Fourth argument
+20	RES1	R	First result
+24	RES2	R	Second result
+28	ERR	R	Error code (0 = success)

Command codes are 0 DISCOVER, 1 OPEN, 2 READ, 3 CLOSE, 4 STAT, 5 REaddir, 6 CREATE_WRITE, 7 WRITE. Chapter 35 has the full protocol.

Most programs never touch this region directly. BASIC's LOAD, SAVE, BLOAD, COMPILE, TRANSPILE, ASSEMBLE, DIR, and TYPE go through it, as does the program loader. It is here for the rare case where you are writing your own loader.

The File I/O block carries its data address through a 32-bit register. A read, write, or directory listing is refused if the staged buffer would cross the sign-extended alias guard at \$FFFF0000 or run past active RAM. In that case the file block reports FILE_ERR_RANGE instead of wrapping into low memory or copying only part of the transfer. Chapter 35 gives the register-level details.

24.7 The IRQ diagnostics block

\$F23C0-\$F23DF. Eight read-only words that report interrupt activity. These are diagnostic taps; ordinary programs do not need them.

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
\$F23DC	IRQ_DIAG_WATCHDOG	Latched watchdog event count

Chapter 31 describes the trap and exception model that backs these counters.

24.8 PEEK and POKE from BASIC

BASIC has byte-width historical primitives and explicit wider forms:

- PEEK(addr) and PEEK8(addr) read an 8-bit unsigned value. POKE addr, value and POKE8 addr, value write one byte. These accept any address and are the right choice for byte-oriented MMIO registers such as TED audio (\$F0F00-\$F0F05), the WAV volume bytes (\$F0BF1-\$F0BF2), the SID register file, and any single byte of the VGA palette.
- PEEK32(addr) reads a 32-bit unsigned value. POKE32 addr, value writes a 32-bit unsigned value. Both require addr to be a multiple of 4; an unaligned address raises ?FC ERROR.
- PEEK64(addr) reads a 64-bit value from an 8-byte aligned address. It returns an exact integer qword, so HEX\$(PEEK64(addr)), POKE64 dst, PEEK64(src), and other integer-compatible expressions preserve pointer and bitfield payloads.

```
10 REM READ THE VBLANK FLAG FROM VIDEOCHIP
20 V=PEEK32(&H000F0008)
30 IF (V AND 1)=0 THEN GOTO 20
40 REM DO SOMETHING AT THE START OF VBLANK
```

This is a 32-bit status register, so PEEK32 is the right access width. The loop waits until bit 0 becomes non-zero.

```
10 REM EMIT A BANNER ONE BYTE AT A TIME
20 B$="INTUITION ENGINE"
30 FOR I=1 TO LEN(B$)
40 POKE8 &H000F0700,ASC(MID$(B$,I,1))
50 NEXT I
60 POKE8 &H000F0700,13
```

TERM_OUT is a byte output register. The loop writes one character code at a time, then line 60 writes carriage return. Use POKE8 here; a 32-bit POKE would still deliver a low byte, but it hides the fact that this is a byte device.

The MMIO map decides what a read actually returns. \$F0700 is the TERM_OUT register: it is write-only and reads back zero. To poll terminal status, read TERM_STATUS at \$F0704 (bit 0 input available, bit 1 output ready). Tables in the following chapters mark which MMIO addresses are write-only and which return a stable value on read.

24.9 Addresses on the smaller CPUs

The 6502 and Z80 cannot see more than 64 kilobytes at a time. For them, the I/O region appears at the top of the 16-bit space:

- 6502: addresses \$F000-\$FFF9 map to \$F0000-\$F0FF9, except for the bank-register page \$F700-\$F705 and the VRAM_BANK_REG at \$F7F0, which the CPU adapter intercepts. \$FFFA-\$FFFF is the 6502 vector table and is not translated. The VGA is reached at \$D700-\$D70D; the ULA at \$D800-\$D817 with a paged VRAM port. Chapter 27 lists the full set of aliases.
- Z80: same \$F000-\$FFF9 MMIO window with the same \$F700 bank-register intercept. The Z80 also exposes the VGA through OUT (\$A0..\$AA) port I/O, and the ULA through ports \$FE/\$FD/\$BE/\$FA-\$FC with a paged VRAM port.

IE32, M68K, and x86 see the low 32-bit bus window directly. IE64 sees that same low window and can also reach RAM above it through its 64-bit physical path. x86 data accesses may use the native MMIO addresses at \$000F0000-\$000FFFFFF; the \$F000-\$FFFF compatibility mirror is a data-access mirror only. Instruction fetch at \$F000 reads program RAM at \$0000F000. A POKE at \$000F0700 from BASIC, a 68K MOVE.L D0, (\$F0700).L, and an x86 data store to \$000F0700 reach the same terminal register because all three names land inside the common low MMIO window.

24.10 What comes next

Chapter 25 begins the per-CPU section with IE64, the native processor of the Intuition Engine. It is the easiest CPU to write for because it has the most general instruction set and the most registers; everything else in Part IV is the same story told through a smaller instruction set.