

Chapter 41 - Building, Loading, and Laying Out Programmes

A small BASIC listing can live entirely at the prompt. A larger programme needs a layout: code, data, stack, screen buffers, sound buffers, file buffers, and shared request buffers all have to fit on the same bus.

This chapter is about choosing that layout from inside Intuition Engine. It does not require an external assembler. BASIC can compile or assemble IE64 code from files on the disk volume, and IE Mon can enter bytes or short IE64 instructions directly.

41.1 File Types

Suffix	CPU or data	Usual reader path
.BAS	IE64 BASIC text	LOAD, SAVE, RUN.
.IE64	IE64 flat image	COMPILE, ASSEMBLE, or IE Mon byte/assemble entry.
.IE32 or .IEX	IE32 flat image	Loaded by RUN or coprocessor start.
.IE65	6502 image	Loaded by RUN or coprocessor start.
.IE80	Z80 image	Loaded by RUN or coprocessor start.
.IE68	M68K flat image	Loaded by RUN or coprocessor start.
.IE86	x86 flat image	Loaded by RUN or coprocessor start.
.IES	IE Script automation	RUN "name.IES" or IE Mon script.

The file suffix selects the loader path. It does not change the machine. All of these images still run on the same Intuition Engine bus.

41.2 Native IE64 From BASIC

For IE64, the machine can make a native image without leaving BASIC:

```
10 REM MAKE A SMALL NATIVE PROGRAMME
20 PRINT "MADE INSIDE IE"
30 COMPILE "MADE"
40 RUN "MADE.IE64"
```

Expected result: COMPILE writes MADE.ie64 and a generated assembly listing on the disk volume, then RUN starts the native IE64 image.

TRANSPILE "MADE" writes only the assembly source. ASSEMBLE "MADE" reads that source and writes MADE.ie64 again. Both commands are direct-mode commands. They are not stored-programme statements.

If there is no stored programme to compile, RUN AOT, COMPILE, and TRANSPILE print ?NO CODE TO COMPILE.

41.3 IE Mon Entry

For small routines, IE Mon is often the cleanest path. The old route is byte entry:

```
(ie64)> w 1000 01 17 00 00 00 20 00 00
(ie64)> d 1000 #1
00001000: 01 17 00 00 00 20 00 00  move.q r2, #2000
```

For IE64 only, the monitor can also assemble one instruction at a time:

```
(ie64)> A 1000
asm $0000000000001000> move.q r2,#2000
$0000000000001000: 01 17 00 00 00 20 00 00  move.q r2, #2000
asm $0000000000001008>
Exited IE64 assemble mode
```

The monitor assembler is still IE-native. It is not a source-file assembler. It accepts one IE64 instruction, writes its bytes, and moves on. Use `d` after entry to prove what the bytes mean.

41.4 Load Addresses and Stacks

CPU	Normal image entry	Stack rule
IE64	\$001000 for flat images	R31 starts at the IE64 stack area; use MEMALLOC for shared buffers.
IE32	\$001000 for ordinary programme bytes	Stack grows down from \$09F000.
6502	\$0600 for the normal runner; \$0000 for coprocessor worker images	Stack page is \$0100 to \$01FF.
Z80	\$0000 for the normal runner and coprocessor worker images	Use SP explicitly in machine code.
M68K	M68K entry point in flat RAM	A7 starts at the M68K stack top; the flat loader leaves the stack guard hole untouched.
x86	.ie86 starts with EIP = 0	Segments are flat and zero-based.

These are image-loading rules, not MMIO rules. Once a CPU is running, ordinary loads and stores reach the bus according to that CPU's memory model.

41.5 Where To Put Data

Use this checklist for larger programmes:

Data	Good home
Small BASIC scratch buffers	Ordinary variables and strings.
Shared device buffers	MEMALLOC(size[,align]).
VideoChip front framebuffer	\$00100000 by convention in examples.
VideoChip back buffers	Other backed RAM below the active low-window limit.
Voodoo texture RAM	\$000D0000 to \$000DFFFF.

Data	Good home
File I/O buffers	MEMALLOC with FILE_DATA_PTR, or IE64 FILE_DATA_PTR64 for high buffers.
Coprocessor request and response buffers	MEMALLOC; COCALL rejects raw spans that are not public buffers.
Monitor-entered code	\$1000 unless the CPU chapter says otherwise.

Do not place private data in an MMIO aperture. Reads and writes there go to devices.

41.6 A Layout Probe

This BASIC listing allocates three public buffers, writes a marker into each, and then reads back the markers.

```

10 REM LAYOUT PROBE
20 CODE=MEMALLOC(256,16)
30 REQ=MEMALLOC(16,4)
40 RESP=MEMALLOC(16,4)
50 POKE32 CODE,&H11111111
60 POKE32 REQ,&H22222222
70 POKE32 RESP,&H33333333
80 PRINT HEX$(CODE),HEX$(REQ),HEX$(RESP)
90 PRINT HEX$(PEEK32(CODE))
100 PRINT HEX$(PEEK32(REQ))
110 PRINT HEX$(PEEK32(RESP))

```

Expected result: line 80 prints three different low-memory addresses. Lines 90 to 110 print 11111111, 22222222, and 33333333.

This is not a memory allocator tutorial. It is the habit to keep: allocate shared buffers explicitly, write known markers, then inspect them before connecting the buffer to a device or worker.

41.7 Symbol Families

The per-CPU symbol families define the same public device names for machine-code authors:

CPU	Symbol family
IE64	IE64 symbols and IE64 assembler names.
IE32	IE32 symbols.
6502	6502 symbols and small-CPU aliases.
Z80	Z80 symbols and port aliases.
M68K	M68K symbols.
x86	x86 symbols.

In this guide, those names are lookup aids. The reader path remains: type BASIC, enter bytes in IE Mon, or assemble IE64 inside the machine.

41.8 Side Effects and Limits

- COMPILE, TRANSPILE, and ASSEMBLE write files on the disk volume. Check Chapter 35 for file errors.

- MEMALLOC returns public low-memory buffers. It is the right tool when a device, coprocessor, or other CPU must see the data.
- A loaded flat image can overwrite ordinary RAM. It must not be used as a casual storage format for private BASIC data.
- A 32-bit CPU image can only name the low 4 GB bus window.
- The 6502 and Z80 need their banked ranges or per-chip aliases for regions beyond their 16-bit address space.

Chapter 42 uses these layout rules to start a worker CPU and pass a request through shared memory.