

## Chapter 26 - IE32

---

IE32 is the 32-bit predecessor of IE64. It uses the same memory bus, sees the same MMIO map, and has the same fixed-length encoding. The differences are entirely about size and complexity: IE32 has half-width registers, fewer of them, no MMU, no FPU, and a simpler instruction set.

For programs that fit in 4 gigabytes of address space and do not need floating-point or virtual memory, IE32 is faster to write for and easier to read back.

### 26.1 Register file

---

IE32 names its 16 general-purpose registers with letters rather than numbers:

Register	Role
A	Accumulator. Default destination for most arithmetic.
X, Y, Z	Index registers. Used for addressing and counters.
B, C, D, E, F, G, H, S, T, U, V, W	General purpose.
PC	Program counter (32 bits, not addressable as a GPR).
SP	Stack pointer (32 bits, not addressable as a GPR).

Each register holds one 32-bit unsigned value. Signed operations interpret the same bits as two's-complement.

There is no zero register on IE32. Code that needs the constant zero loads it into a free register or uses the addressing mode's immediate form.

### 26.2 Instruction encoding

---

Every IE32 instruction is 8 bytes long:

```
byte 0 : opcode           (8 bits)
byte 1 : register field   (8 bits - low 4 bits = register index)
byte 2 : addressing mode  (8 bits)
byte 3 : reserved        (8 bits)
bytes 4-7 : operand       (32 bits)
```

The operand interpretation depends on the addressing mode. Word alignment is required for jump and call targets.

### 26.3 Addressing modes

---

Code	Mode	Operand meaning
\$00	Immediate	Operand is the literal value
\$01	Register direct	Operand is a register index
\$02	Register indirect	Operand is a register holding the address

Code	Mode	Operand meaning
\$03	Memory indirect	Operand is the address of an address (double indirection)
\$04	Direct	Operand is the memory address

For example, LDA #\$42 (immediate) loads \$42 into A; LDA \$F0700 (direct) loads from address \$F0700; LDA (X) (register indirect) loads from the address in X.

## 26.4 Instruction set

### 26.4.1 Data movement

LOAD / STORE are the generic forms. The letter-suffixed forms target specific registers:

Mnemonic	Operation
LDA/LDX/LDY/LDZ	Load A/X/Y/Z from operand
LDB-LDW, LDH/LDS/LDT	Load extended register
STA/STX/STY/STZ	Store A/X/Y/Z to operand
STB-STW, STH/STS/STT	Store extended register
INC reg	Increment a register (or memory in operand)
DEC reg	Decrement a register (or memory in operand)

### 26.4.2 Arithmetic and logic

ADD, SUB, MUL, DIV, MOD, AND, OR, XOR, NOT, SHL, SHR. Each takes a register operand and an addressing-mode operand; the result lands in the named register.

Example: ADD A, X adds the value in X to A. ADD A, #5 adds an immediate 5 to A.

### 26.4.3 Control flow

IE32 has no flags. Conditional jumps test the named register directly:

Mnemonic	Branches if
JMP	unconditional
JZ reg, addr	reg == 0
JNZ reg, addr	reg != 0
JGT reg, addr	reg > 0 (signed)
JGE reg, addr	reg >= 0 (signed)
JLT reg, addr	reg < 0 (signed)
JLE reg, addr	reg <= 0 (signed)

The signed sense of JGT/JGE/JLT/JLE matters: a value of \$FFFFFFFF is treated as -1 by these instructions, so JLT A, addr branches when A has its high bit set.

A typical comparison sequence is:

```

SUB A, X      ; A = A - X
JGT A, label  ; branch if A > X

```

## 26.4.4 Subroutines

Mnemonic	Operation
JSR addr	Push PC + 8 to the stack, jump to addr
RTS	Pop a return address from the stack, jump there
PUSH reg	SP -= 4; [SP] = reg
POP reg	reg = [SP]; SP += 4

The stack grows downward. The reset SP is `STACK_START = $9F000`.

## 26.4.5 System

Mnemonic	Operation
NOP	No operation
HALT	Halt the CPU
SEI	Set the interrupt-enable flag
CLI	Clear the interrupt-enable flag
RTI	Return from interrupt
WAIT n	Wait approximately n microseconds

## 26.5 Interrupts and timing

IE32 has a single interrupt vector at memory `$0004`. When an interrupt fires:

1. PC is pushed to the stack.
2. The interrupt-enable flag is cleared.
3. PC is loaded from `$0004`.

RTI reverses the sequence: pops PC and re-enables interrupts.

IE32 publishes no timer MMIO block. Use `WAIT n` for short microsecond delays during normal execution, and use device status registers or interrupts for frame, audio, and I/O timing. In IE Mon single-step mode, `WAIT` advances as one instruction and does not sleep; resumed execution with `g` uses the real-time delay.

## 26.6 Reset and boot

On reset, `PC = $0000` and `SP = STACK_START = $9F000`. The conventional first instruction at `$0000` is a `JMP` to `PROG_START = $1000`, the usual start address for byte-entered IE32 programs and loaded IE32 images.

## 26.7 A small example

This IE32 byte-entry program plays a three-channel chord through the SN76489. IE32 stores 32-bit words, and the SN76489 write port uses the low byte of each word as the chip's byte-stream input. That makes the example a good first bus exercise: every STA is a word store, but the sound chip hears a single command byte.

```
(ie32)> w 1000 20 00 00 00 01 00 00 00 24 00 04 00 00 08 0F 00
(ie32)> w 1010 20 00 00 00 00 8B 00 00 00 24 00 04 00 30 0C 0F 00
(ie32)> w 1020 20 00 00 00 00 1A 00 00 00 24 00 04 00 30 0C 0F 00
(ie32)> w 1030 20 00 00 00 00 90 00 00 00 24 00 04 00 30 0C 0F 00
(ie32)> w 1040 20 00 00 00 00 A3 00 00 00 24 00 04 00 30 0C 0F 00
(ie32)> w 1050 20 00 00 00 00 15 00 00 00 24 00 04 00 30 0C 0F 00
(ie32)> w 1060 20 00 00 00 00 B2 00 00 00 24 00 04 00 30 0C 0F 00
(ie32)> w 1070 20 00 00 00 00 CD 00 00 00 24 00 04 00 30 0C 0F 00
(ie32)> w 1080 20 00 00 00 00 11 00 00 00 24 00 04 00 30 0C 0F 00
(ie32)> w 1090 20 00 00 00 00 D3 00 00 00 24 00 04 00 30 0C 0F 00
(ie32)> w 10A0 06 00 00 00 00 A0 10 00 00
(ie32)> r pc 1000
(ie32)> d 1000 #21
> 001000: 20 00 00 00 01 00 00 00 LDA A, #00000001
   001008: 24 00 04 00 00 08 0F 00 STA A, M:$000F0800
   001010: 20 00 00 00 8B 00 00 00 LDA A, #0000008B
   001018: 24 00 04 00 30 0C 0F 00 STA A, M:$000F0C30
   001020: 20 00 00 00 1A 00 00 00 LDA A, #0000001A
   001028: 24 00 04 00 30 0C 0F 00 STA A, M:$000F0C30
   001030: 20 00 00 00 90 00 00 00 LDA A, #00000090
   001038: 24 00 04 00 30 0C 0F 00 STA A, M:$000F0C30
   001040: 20 00 00 00 A3 00 00 00 LDA A, #000000A3
   001048: 24 00 04 00 30 0C 0F 00 STA A, M:$000F0C30
   001050: 20 00 00 00 15 00 00 00 LDA A, #00000015
   001058: 24 00 04 00 30 0C 0F 00 STA A, M:$000F0C30
   001060: 20 00 00 00 B2 00 00 00 LDA A, #000000B2
   001068: 24 00 04 00 30 0C 0F 00 STA A, M:$000F0C30
   001070: 20 00 00 00 CD 00 00 00 LDA A, #000000CD
   001078: 24 00 04 00 30 0C 0F 00 STA A, M:$000F0C30
   001080: 20 00 00 00 11 00 00 00 LDA A, #00000011
   001088: 24 00 04 00 30 0C 0F 00 STA A, M:$000F0C30
   001090: 20 00 00 00 D3 00 00 00 LDA A, #000000D3
   001098: 24 00 04 00 30 0C 0F 00 STA A, M:$000F0C30
T 0010A0: 06 00 00 00 A0 10 00 00 JMP $000010A0
(ie32)> b 10A0
(ie32)> g
(ie32)> m F0C31 1
000F0C31: 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
(ie32)> bc 10A0
```

The instruction format makes each 8-byte group readable once the mode byte is known:

Address	Bytes	Meaning
\$1000	20 00 00 00 01 00 00 00	LDA A, #1. Opcode \$20 is LDA; register byte \$00 selects A; mode \$00 is immediate; operand is little-endian \$00000001.
\$1008	24 00 04 00 00 08 0F 00	STA A, M:\$000F0800. Opcode \$24 is STA; mode \$04 is direct memory. This writes AUDIO_CTRL.

Address	Bytes	Meaning
\$1010	20 00 00 00 8B 00 00 00	Loads the SN76489 channel 0 tone latch byte \$8B into A.
\$1018	24 00 04 00 30 0C 0F 00	Stores the low byte of A to \$F0C30, the SN76489 byte-stream port.
\$1020	20 00 00 00 1A 00 00 00	Loads the channel 0 tone high-byte payload \$1A.
\$1028	24 00 04 00 30 0C 0F 00	Sends that payload through the same port.
\$1030	20 00 00 00 90 00 00 00	Loads attenuation latch \$90, channel 0 at full volume.
\$1038	24 00 04 00 30 0C 0F 00	Sends the attenuation latch.
\$1040-\$1068	Six instructions	Sends channel 1: latch \$A3, high byte \$15, attenuation \$B2.
\$1070-\$1098	Six instructions	Sends channel 2: latch \$CD, high byte \$11, attenuation \$D3.
\$10A0	06 00 00 00 A0 10 00 00	JMP \$000010A0, a self-loop that leaves the chord active.

The first two instructions enable audio by writing 1 to AUDIO\_CTRL. The final dump reads SN\_PORT\_READY, which returns bit 0 set when the byte-stream port can accept another byte. The SN76489 divider is  $\text{clock} / (32 * \text{Hz})$ . At the NTSC clock, useful musical dividers are about \$1AB for C, \$153 for E, and \$11D for G. The latch byte carries the channel number and the low four divider bits; the following data byte carries the upper six bits. \$90, \$B2, and \$D3 are attenuation latches for channels 0, 1, and 2; lower attenuation is louder.

Try changing \$B2 to \$B0 and run again. That makes the middle voice as loud as the root without changing the pitch bytes.

## 26.8 VGA text example

This second IE32 byte-entry program uses VGA text mode. It switches the VGA card to mode \$03, enables it, and writes the first four text cells at \$B8000. Each cell is two bytes: character, then attribute. The attribute high nibble is the background colour and the low nibble is the foreground colour.

```

(ie32)> w 1100 20 00 00 00 03 00 00 00 24 00 04 00 00 10 0F 00
(ie32)> w 1110 20 00 00 00 01 00 00 00 24 00 04 00 08 10 0F 00
(ie32)> w 1120 20 00 00 00 49 00 00 00 24 00 04 00 00 80 0B 00
(ie32)> w 1130 20 00 00 00 1E 00 00 00 24 00 04 00 01 80 0B 00
(ie32)> w 1140 20 00 00 00 45 00 00 00 24 00 04 00 02 80 0B 00
(ie32)> w 1150 20 00 00 00 2F 00 00 00 24 00 04 00 03 80 0B 00
(ie32)> w 1160 20 00 00 00 33 00 00 00 24 00 04 00 04 80 0B 00
(ie32)> w 1170 20 00 00 00 4E 00 00 00 24 00 04 00 05 80 0B 00
(ie32)> w 1180 20 00 00 00 32 00 00 00 24 00 04 00 06 80 0B 00
(ie32)> w 1190 20 00 00 00 5F 00 00 00 24 00 04 00 07 80 0B 00
(ie32)> w 11A0 06 00 00 00 A0 11 00 00
(ie32)> r pc 1100
(ie32)> d 1100 #21
> 001100: 20 00 00 00 03 00 00 00 LDA A, #$00000003
001108: 24 00 04 00 00 10 0F 00 STA A, M:$000F1000
001110: 20 00 00 00 01 00 00 00 LDA A, #$00000001
001118: 24 00 04 00 08 10 0F 00 STA A, M:$000F1008
001120: 20 00 00 00 49 00 00 00 LDA A, #$00000049
001128: 24 00 04 00 00 80 0B 00 STA A, M:$000B8000
001130: 20 00 00 00 1E 00 00 00 LDA A, #$0000001E
001138: 24 00 04 00 01 80 0B 00 STA A, M:$000B8001
001140: 20 00 00 00 45 00 00 00 LDA A, #$00000045
001148: 24 00 04 00 02 80 0B 00 STA A, M:$000B8002
001150: 20 00 00 00 2F 00 00 00 LDA A, #$0000002F
001158: 24 00 04 00 03 80 0B 00 STA A, M:$000B8003
001160: 20 00 00 00 33 00 00 00 LDA A, #$00000033
001168: 24 00 04 00 04 80 0B 00 STA A, M:$000B8004
001170: 20 00 00 00 4E 00 00 00 LDA A, #$0000004E
001178: 24 00 04 00 05 80 0B 00 STA A, M:$000B8005
001180: 20 00 00 00 32 00 00 00 LDA A, #$00000032
001188: 24 00 04 00 06 80 0B 00 STA A, M:$000B8006
001190: 20 00 00 00 5F 00 00 00 LDA A, #$0000005F
001198: 24 00 04 00 07 80 0B 00 STA A, M:$000B8007
T 0011A0: 06 00 00 00 A0 11 00 00 JMP $000011A0
(ie32)> b 11A0
(ie32)> g
(ie32)> m B8000 1
000B8000: 49 1E 45 2F 33 4E 32 5F 00 00 00 00 00 00 00 00 I.E/3N2_.....
(ie32)> bc 11A0

```

The first four instructions set up the card: VGA\_MODE at \$F1000 gets mode \$03, and VGA\_CTRL at \$F1008 gets bit 0 set. After that, each pair of instructions writes one byte to the text buffer. IE32 stores a word, but the VGA text window uses the low byte at the address being written, so the program writes the character and attribute bytes separately.

The dump at \$B8000 is the proof. It reads as:

Bytes	Meaning
49 1E	Character I, yellow on blue.
45 2F	Character E, bright white on green.
33 4E	Character 3, yellow on red.
32 5F	Character 2, bright white on magenta.

On the VGA layer, the top-left of the screen should show IE32 in four different attributes. Try changing the byte \$49 at \$1120 to \$56; the first character becomes V.

## 26.9 What comes next

---

Chapter 27 leaves the IE-native processors behind and begins the heritage section: the 6502, the small 8-bit CPU that powered the home computers of the late 1970s and early 1980s. The 6502 has only three registers and a 64-kilobyte address space, but much of the software written for it is still beautiful to read, and Intuition Engine runs it natively.