

Chapter 27 - The 6502 Processor

The 6502 is the small 8-bit processor that powered the home computers of the late 1970s and early 1980s: the Commodore 64, the Apple II, the BBC Micro, the Atari 8-bit family, and many others. Intuition Engine runs it natively, with cycle-accurate timing and the full documented and undocumented instruction set.

This chapter is the architectural reference for writing 6502 code on Intuition Engine: registers, addressing modes, instruction groups, vectors, and the bank-and-MMIO mapping that gives the small CPU a path to the low 32-bit hardware window on the 64-bit Intuition Engine bus.

27.1 Architecture

The 6502 has three 8-bit data registers, a 16-bit program counter, and an 8-bit stack pointer:

Register	Width	Purpose
A	8	Accumulator. The destination for arithmetic and logic.
X	8	Index register. Used for indexed addressing and counters.
Y	8	Index register. Same uses as X but with different addressing modes.
PC	16	Program counter.
SP	8	Stack pointer. The actual stack lives at $\$0100 + SP$, growing downward.
P	8	Status register. Flags N, V, -, B, D, I, Z, C (high bit first).

The seven P flags:

Bit	Name	Meaning
7	N	Negative - copy of bit 7 of last result
6	V	Overflow - set on signed overflow from arithmetic
5	-	Always reads 1
4	B	Break - set on BRK, used to distinguish BRK from IRQ
3	D	Decimal - when set, ADC/SBC operate in BCD
2	I	Interrupt disable - when set, the CPU ignores IRQ
1	Z	Zero - set when last result was zero
0	C	Carry - set on unsigned overflow from arithmetic and from shifts

27.2 Memory model

The 6502 sees a 64-kilobyte address space, $\$0000-\$FFFF$. Two pages have a special role:

- **Zero page** ($\$0000-\$00FF$) - addressed with a single byte; cheaper and faster to access than ordinary memory. Most variables in classic 6502 programs live here.
- **Stack** ($\$0100-\$01FF$) - the SP register names a byte in this page. Push decrements SP, pop increments SP.

The top six bytes of the address space hold the interrupt vectors:

Address	Vector	Triggered by
\$FFFA-\$FFFB	NMI	Non-maskable interrupt
\$FFFC-\$FFFD	RESET	Power-on or reset
\$FFFE-\$FFFF	IRQ/BRK	Maskable interrupt or BRK instr

Each vector is a 16-bit little-endian pointer to the handler.

27.3 The Intuition Engine memory map for the 6502

On a real 1980 machine the 64-kilobyte map was hard-coded by the manufacturer. Intuition Engine gives the 6502 the same 64-K view, but provides a small set of fixed apertures that map back into the larger low-window bus space:

Range	Purpose
\$0000-\$00FF	Zero page (CPU RAM)
\$0100-\$01FF	Stack page (CPU RAM)
\$0200-\$1FFF	Free RAM
\$2000-\$3FFF	Bank 1 window (8 KiB, sprite/general data)
\$4000-\$5FFF	Bank 2 window (8 KiB, font/general data)
\$6000-\$7FFF	Bank 3 window (8 KiB, general data)
\$8000-\$BFFF	VRAM bank window (16 KiB)
\$C000-\$CFFF	Free RAM
\$D200-\$D20A	POKEY registers
\$D400-\$D40F	PSG registers
\$D500-\$D55F	SID registers
\$D600-\$D605	TED audio registers
\$D620-\$D632	TED video registers
\$D700-\$D70D	VGA registers
\$D800-\$D817	ULA registers (with paged VRAM port)
\$E000-\$EFFF	Voodoo banked window
\$F000-\$FFF9	MMIO window (maps to \$F0000-\$F0FF9)
\$FFFA-\$FFFF	Vector table (not translated)

The CPU adapter additionally intercepts the bank-control bytes inside the MMIO window:

Address	Register
\$F700/\$F701	BANK1_REG_LO / BANK1_REG_HI
\$F702/\$F703	BANK2_REG_LO / BANK2_REG_HI
\$F704/\$F705	BANK3_REG_LO / BANK3_REG_HI

Address	Register
\$F7F0	VRAM_BANK_REG
\$F7F2	Voodoo window bank low byte
\$F7F3	Voodoo window bank page

Writes to these bytes change the backing region of the matching banked range, instead of falling through to the bus. All other addresses inside \$F000-\$FFF9 translate transparently; for example, \$FC30 reaches \$F0C30 which is SN_PORT_WRITE, and \$F0C0 reaches \$F00C0 which is VIDEO_PAL_BANK_4.

The 6502 program never sees the full 32-bit address; it sees a 16-K (VRAM) or 8-K (banks) piece of it at a time. There is no 16-bit alias that reaches TERM_OUT (\$F0700) directly, because the \$F700 page is captured by the bank-register intercept. Code that needs the system terminal can select TERM_IO_BANK into bank 1 with SET_TERMINAL_BANK, then use the 6502 terminal aliases at \$2700-\$27FF; TERM_OUT is \$2700 in that window and resolves to bus \$F0700.

27.4 Addressing modes

The 6502 has 13 addressing modes. The mnemonic conventions used throughout this chapter are:

Mode	Notation	Meaning
Implied	CLC	No operand
Accumulator	ASL A	The accumulator is the operand
Immediate	LDA #\$42	The literal byte
Zero page	LDA \$80	Byte at zero-page address
Zero page,X	LDA \$80,X	Byte at $(\$80 + X) \bmod 256$
Zero page,Y	LDX \$80,Y	Byte at $(\$80 + Y) \bmod 256$
Absolute	LDA \$1234	Byte at \$1234
Absolute,X	LDA \$1234,X	Byte at $\$1234 + X$
Absolute,Y	LDA \$1234,Y	Byte at $\$1234 + Y$
Indirect	JMP (\$1234)	Jump to address held at \$1234/\$1235 (JMP only)
Indexed indirect	LDA (\$80,X)	Byte at the address held at $(\$80 + X)/(\$80 + X + 1)$
Indirect indexed	LDA (\$80),Y	Byte at $(\text{address held at } \$80/\$81) + Y$
Relative	BNE label	Signed 8-bit branch offset

The two indirect modes accept only zero-page operands. The indirect-jump bug of the original 6502 is preserved: a JMP (\$XXFF) fetches its high byte from \$XX00, not $(XX+1)00$.

27.5 Instruction set summary

The 6502 has 56 mnemonics. They divide into the following groups. Appendix G has the full opcode table; this section lists each mnemonic and what it does.

27.5.1 Load and store

Mnemonic	Operation
LDA	Load accumulator (sets N, Z)
LDX	Load X (sets N, Z)
LDY	Load Y (sets N, Z)
STA	Store accumulator
STX	Store X
STY	Store Y

27.5.2 Register transfers and stack

Mnemonic	Operation
TAX/TAY/TSX/TXA/TXS/TYA	Copy between registers
PHA/PHP	Push A / push P
PLA/PLP	Pop into A / pop into P

27.5.3 Arithmetic and logic

Mnemonic	Operation
ADC	Add with carry ($A = A + M + C$)
SBC	Subtract with carry ($A = A - M - (1 - C)$)
AND/ORR/EOR	Bitwise AND / OR / XOR
INC/DEC	Increment / decrement memory
INX/INY/DEX/DEY	Increment / decrement index registers
ASL/LSR	Arithmetic shift left / logical shift right
ROL/ROR	Rotate left / right through carry
CMP/CPX/CPY	Compare against A/X/Y (sets N,Z,C)
BIT	Test memory bits against A (sets N,V,Z)

In decimal mode (D flag set), ADC and SBC operate on packed BCD digits.

27.5.4 Branches

Each branch is a signed 8-bit PC-relative offset, range -128 to +127:

Mnemonic	Branches if
BCC	$C = 0$
BCS	$C = 1$
BEQ	$Z = 1$
BNE	$Z = 0$

Mnemonic	Branches if
BMI	N = 1
BPL	N = 0
BVC	V = 0
BVS	V = 1

27.5.5 Jumps and subroutines

Mnemonic	Operation
JMP	Unconditional jump (absolute or indirect)
JSR	Jump to subroutine (push return - 1)
RTS	Return from subroutine
BRK	Software interrupt (push PC, push P, set B, jump via \$FFFE/\$FFFF)
RTI	Return from interrupt (pop P, pop PC)

27.5.6 Flag operations

Mnemonic	Operation
CLC/SEC	Clear / set carry
CLI/SEI	Clear / set interrupt disable
CLV	Clear overflow
CLD/SED	Clear / set decimal mode
NOP	No operation

27.6 Undocumented opcodes

The original NMOS 6502 has 105 unintended opcodes that the silicon nevertheless decodes into observable behaviour. Intuition Engine's 6502 implements every one of them so that classic software using them continues to work. Their mnemonics in common use include SLO, RLA, SRE, RRA, SAX, LAX, DCP, ISC, and a few others. The full list is in Appendix G. These are best avoided in fresh code; they exist for compatibility with software that already relies on them.

27.7 Interrupts on Intuition Engine

When the IRQ line is asserted and the I flag is clear:

1. The current PC is pushed (high byte first, then low).
2. P is pushed with the B flag clear.
3. I is set.
4. PC is loaded from \$FFFE/\$FFFF.

NMI is identical but uses \$FFFA/\$FFFB and ignores the I flag. RESET is identical but uses \$FFFC/\$FFFD, sets I, and does not push anything (it uses the stack page as scratch).

BRK works like an IRQ but pushes P with the B flag set, so the handler can distinguish a software trap from a hardware interrupt.

27.8 A small example

This 6502 byte-entry program plays a three-channel Atari-style POKEY chord. It uses \$F800, the 6502 alias for AUDIO_CTRL, and \$D200-\$D208, the 6502 POKEY register window:

```
(6502)> w 1000 A9 01 8D 00 F8 A9 00 8D 08 D2
(6502)> w 100A A9 79 8D 00 D2 A9 AF 8D 01 D2
(6502)> w 1014 A9 5F 8D 02 D2 A9 AC 8D 03 D2
(6502)> w 101E A9 3F 8D 04 D2 A9 A8 8D 05 D2
(6502)> w 1028 4C 28 10
(6502)> d 1000 #17
 1000: A9 01          LDA #$01
 1002: 8D 00 F8      STA $F800
 1005: A9 00          LDA #$00
 1007: 8D 08 D2      STA $D208
 100A: A9 79          LDA #$79
 100C: 8D 00 D2      STA $D200
 100F: A9 AF          LDA #$AF
 1011: 8D 01 D2      STA $D201
 1014: A9 5F          LDA #$5F
 1016: 8D 02 D2      STA $D202
 1019: A9 AC          LDA #$AC
 101B: 8D 03 D2      STA $D203
 101E: A9 3F          LDA #$3F
 1020: 8D 04 D2      STA $D204
 1023: A9 A8          LDA #$A8
 1025: 8D 05 D2      STA $D205
T 1028: 4C 28 10     JMP $1028
(6502)> r pc 1000
(6502)> b 1028
(6502)> g
(6502)> m D200 1
D200: 79 AF 5F AC 3F A8 00 00 00 00 C0 00 00 00 00 00  y._.?.....
(6502)> bc 1028
```

The program uses only two 6502 instruction forms. LDA #byte loads a value into the accumulator, and STA absolute stores that value into a hardware register. The 6502 is little-endian, so the address operand in each STA is entered low byte first: STA \$D204 is entered as 8D 04 D2.

Each instruction is variable length, so the bytes are explained by instruction rather than by fixed slots:

Address	Bytes	Meaning
\$1000	A9 01	LDA #\$01. Opcode \$A9 is immediate load into A; \$01 is the value that enables audio.
\$1002	8D 00 F8	STA \$F800. Opcode \$8D is absolute store; the operand is low byte then high byte, so \$F800 is entered as 00 F8.
\$1005	A9 00	Loads \$00 for AUDCTL, selecting ordinary channel clocking.
\$1007	8D 08 D2	Stores to \$D208, the 6502 POKEY alias for AUDCTL.
\$100A	A9 79	Loads divider \$79 for channel 1.

Address	Bytes	Meaning
\$100C	8D 00 D2	Stores to \$D200, AUDF1.
\$100F	A9 AF	Loads control \$AF, pure-tone distortion with volume \$0F.
\$1011	8D 01 D2	Stores to \$D201, AUDC1, which starts channel 1.
\$1014	A9 5F	Loads divider \$5F for channel 2.
\$1016	8D 02 D2	Stores to \$D202, AUDF2.
\$1019	A9 AC	Loads pure-tone control with volume \$0C.
\$101B	8D 03 D2	Stores to \$D203, AUDC2, which starts channel 2.
\$101E	A9 3F	Loads divider \$3F for channel 3.
\$1020	8D 04 D2	Stores to \$D204, AUDF3.
\$1023	A9 A8	Loads pure-tone control with volume \$08.
\$1025	8D 05 D2	Stores to \$D205, AUDC3, which starts channel 3.
\$1028	4C 28 10	JMP \$1028, a self-loop that leaves the chord active.

\$D208 is AUDCTL; writing 0 selects the ordinary channel clocking. The even registers \$D200, \$D202, and \$D204 are the frequency dividers for channels 1, 2, and 3. Lower divider values give higher pitches, so \$79, \$5F, and \$3F form a spread chord instead of three copies of the same note. The odd registers \$D201, \$D203, and \$D205 are the channel controls. The high nibble \$A0 selects pure-tone distortion, and the low nibble is the volume, here \$0F, \$0C, and \$08.

The `m D200 1` command proves that the six sound registers contain the intended values: the first six bytes of the dump are exactly the bytes the program wrote. The `m` command counts *rows of sixteen bytes*, not bytes, so a single row is the minimum unit it will show (see chapter 33 §33.4.1 for the full format reference). The `d` count uses the `#` prefix to force decimal. Without it, the count is parsed as hexadecimal and `d 1000 17` would emit `0x17 = 23` lines instead of seventeen. To alter the balance, change only the low nibble of the AUDC values. For example, changing \$A8 to \$AF makes the third voice as loud as the first without changing its pitch.

27.9 ULA graphics example

The next program uses the same two 6502 instruction forms, but it talks to the ULA instead of POKEY. The ULA is a byte display device, which makes it a natural partner for the 6502. The 6502 does not write ULA VRAM through a large flat address. It writes a small address latch, then sends each byte through ULA_DATA.

This example draws an 8 by 8 pixel motif in the top-left ULA cell. The first eight bytes go to bitmap offset \$0000. The attribute byte goes to \$1800, the start of the ULA attribute area, and selects bright yellow ink on black paper. The cyan border tells you immediately that the program reached the ULA register block.

```

(6502)> w 1100 A9 05 8D 00 D8 A9 05 8D 04 D8
(6502)> w 110A A9 00 8D 0C D8 A9 00 8D 10 D8
(6502)> w 1114 A9 FF 8D 14 D8 A9 81 8D 14 D8
(6502)> w 111E A9 BD 8D 14 D8 A9 A5 8D 14 D8
(6502)> w 1128 A9 A5 8D 14 D8 A9 BD 8D 14 D8
(6502)> w 1132 A9 81 8D 14 D8 A9 FF 8D 14 D8
(6502)> w 113C A9 00 8D 0C D8 A9 18 8D 10 D8
(6502)> w 1146 A9 46 8D 14 D8 4C 4B 11
(6502)> d 1100 #31
1100: A9 05          LDA #$05
1102: 8D 00 D8      STA $D800
1105: A9 05          LDA #$05
1107: 8D 04 D8      STA $D804
110A: A9 00          LDA #$00
110C: 8D 0C D8      STA $D80C
110F: A9 00          LDA #$00
1111: 8D 10 D8      STA $D810
1114: A9 FF          LDA #$FF
1116: 8D 14 D8      STA $D814
1119: A9 81          LDA #$81
111B: 8D 14 D8      STA $D814
111E: A9 BD          LDA #$BD
1120: 8D 14 D8      STA $D814
1123: A9 A5          LDA #$A5
1125: 8D 14 D8      STA $D814
1128: A9 A5          LDA #$A5
112A: 8D 14 D8      STA $D814
112D: A9 BD          LDA #$BD
112F: 8D 14 D8      STA $D814
1132: A9 81          LDA #$81
1134: 8D 14 D8      STA $D814
1137: A9 FF          LDA #$FF
1139: 8D 14 D8      STA $D814
113C: A9 00          LDA #$00
113E: 8D 0C D8      STA $D80C
1141: A9 18          LDA #$18
1143: 8D 10 D8      STA $D810
1146: A9 46          LDA #$46
1148: 8D 14 D8      STA $D814
T 114B: 4C 4B 11    JMP $114B
(6502)> r pc 1100
(6502)> b 114B
(6502)> g
(6502)> m D800 1
D800: 05 00 00 00 05 00 00 00 01 00 00 00 02 00 00 00 .....
(6502)> bc 114B

```

After g, the top-left display cell contains a bright yellow motif on black paper, with a cyan border around the ULA picture. The single m D800 1 dump shows the row that contains the two visible state changes: \$D800 is the border register (05 in the first column) and \$D804 is ULA_CTRL (05 in the fifth column). The value \$05 means bit 0 is enabled and bit 2, auto-increment, is enabled.

The bitmap bytes the program wrote through \$D814 are not recoverable by reading \$D814 back: that address is a write-only auto-incrementing latch port that does not echo the bytes the program latched. The verification of the bitmap is the visible motif on screen, not a follow-on m D814 read. If a later monitor view gives you a readable ULA VRAM window, use that window for memory inspection. \$D814 itself remains a write-only data port.

Here is the program by instruction:

Address	Bytes	Meaning
\$1100	A9 05	Loads \$05, the ULA border colour number.
\$1102	8D 00 D8	Stores it to \$D800, the 6502 alias for ULA_BORDER.
\$1105	A9 05	Loads ENABLE + AUTO_INC.
\$1107	8D 04 D8	Stores it to \$D804, ULA_CTRL.
\$110A	A9 00	Loads the low latch byte for bitmap offset \$0000.
\$110C	8D 0C D8	Stores it to \$D80C, ULA_ADDR_LO.
\$110F	A9 00	Loads the high latch byte for bitmap offset \$0000.
\$1111	8D 10 D8	Stores it to \$D810, ULA_ADDR_HI.
\$1114-\$1139	A9 byte, 8D 14 D8 repeated	Sends eight bitmap bytes through \$D814, ULA_DATA. Auto-increment advances the latch after each write.
\$113C	A9 00	Loads the low latch byte for attribute offset \$1800.
\$113E	8D 0C D8	Stores it to ULA_ADDR_LO.
\$1141	A9 18	Loads the high latch byte for attribute offset \$1800.
\$1143	8D 10 D8	Stores it to ULA_ADDR_HI.
\$1146	A9 46	Loads the attribute byte: bright ink 6, paper 0.
\$1148	8D 14 D8	Stores it through ULA_DATA at \$1800.
\$114B	4C 4B 11	Loops at \$114B, leaving the picture on screen.

This is a compact example of the 6502's role in Intuition Engine: small byte writes can still control a modern shared bus device, provided the device offers a byte-sized access path.

27.10 What comes next

Chapter 28 covers the Z80, the other dominant 8-bit processor of the era. The Z80 has many more registers than the 6502 and a more powerful instruction set, but it shares the 64-kilobyte address space and similar Intuition Engine bank-and-MMIO mapping.