AURA STUDIO

8085 Microprocessor Simulator & Assembly IDE

User & Reference Manual

Product Version 1.0

STABLE RELEASE

Developed by

Dev Gondaliya · Dev Letwala · Jigar Ghoghari

Table of Contents

1. Introduction

1.1 About Aura Studio

Aura Studio is an integrated development environment (IDE) for the Intel 8085 microprocessor, built to help students, instructors, and self-learners understand microprocessor architecture through direct, hands-on simulation. The application combines a full 8085 assembly-language editor, a two-pass assembler, a cycle-accurate simulation engine, and a set of interactive peripheral emulators inside a single, modern desktop interface.

Rather than requiring physical trainer-kit hardware, Aura Studio reproduces the internal behaviour of the 8085 microprocessor entirely in software — registers, flags, memory, interrupts, and I/O ports all update in real time as a program executes, giving users immediate, visual feedback on how each instruction changes the state of the machine.

1.2 Purpose and Audience

Understanding the Intel 8085 remains fundamental to learning the Von Neumann model of computer architecture that underpins modern processor design. Although newer architectures have since taken its place in industry, the 8085 continues to be taught widely in engineering and technical institutes because its instruction set and register model are compact enough to be grasped fully within a single course, while still illustrating every core concept of instruction fetch, decode, and execution.

Aura Studio is designed with this academic purpose in mind. It allows a learner to write 8085 assembly code, assemble it, and step through execution instruction by instruction, observing exactly how the accumulator, general-purpose registers, flags, stack, and memory change — without needing access to physical trainer-kit hardware.

1.3 Key Capabilities

- A syntax-highlighted assembly editor with autocorrect, assembler directives, and error checking.
- A two-pass assembler and disassembler supporting Intel HEX and raw binary formats.
- A full simulation engine with run, step, step-back, pause, stop, and reset controls.
- Live Register, Flag, Pointer, Execution, and Interrupt panels.
- A dedicated Memory Editor and I/O Port Editor for direct inspection and modification of system state.
- An animated CPU Architecture diagram that visualises data and address bus activity.
- A Memory Heatmap & Visualizer for tracking code execution, reads, and writes across the address space.
- Built-in peripheral simulators: 7-Segment LED display, 8-bit ADC/DAC oscilloscope, a 4-way traffic-light controller, a stepper-motor motion simulator, and a 16×2 character LCD (HD44780) display.
- Wizard-driven tools for generating delay subroutines and configuring interrupt service routines.

- A library of ready-to-load sample programs covering classic 8085 programming exercises.
- Session recovery, printing, and documentation import/export, including conversion of assembly code to C.

1.4 System Requirements and Installation

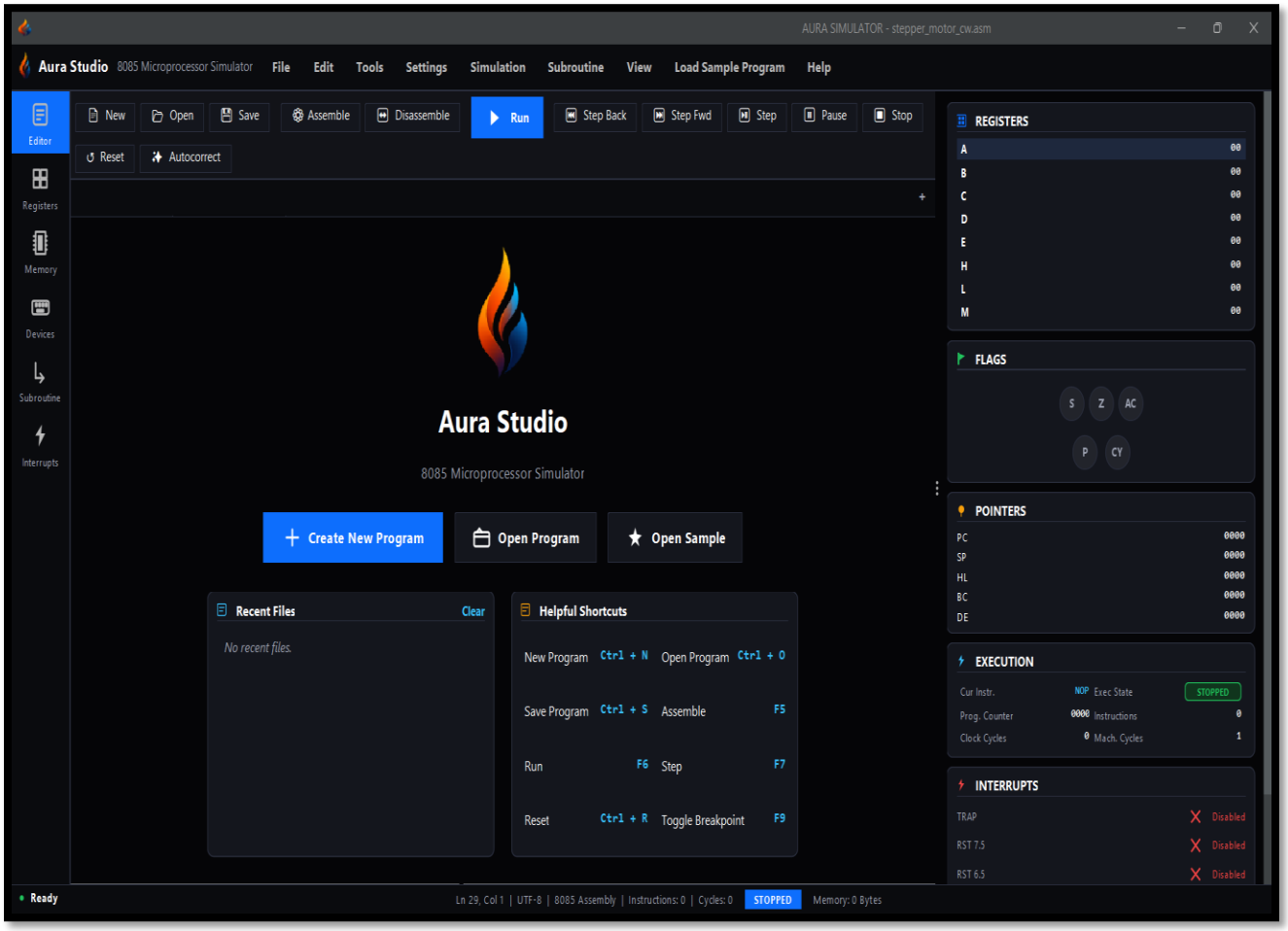
Aura Studio runs as a native desktop application on Windows. Installation follows the standard setup procedure for the distributed package: run the installer (or launch the executable directly for portable builds) and follow the on-screen instructions. No separate 8085 hardware or trainer kit is required to use any feature described in this manual.

1.5 About This Manual

This manual documents the menus, panels, editors, tools, and peripheral simulators that make up Aura Studio, in the order a new user is likely to encounter them: starting from the main window, through the menu bar, the live register and memory views, the built-in tools, and finally the simulation and debugging workflow.

2. The Main Window

When Aura Studio launches, it opens directly into the Editor workspace with an untitled program. The window is divided into four regions: the menu bar and toolbar along the top, a navigation sidebar on the left, the assembly editor in the centre, and the live simulation panels (Registers, Flags, Pointers, Execution, and Interrupts) on the right. A status bar along the bottom reports cursor position, encoding, line/instruction counts, and the current run state.



2.1 Toolbar

The main toolbar exposes the most frequently used actions as single-click buttons:

Control	Function
New	Creates a new, empty assembly program.
Open	Opens an existing program from disk.
Save	Saves the current program.
Assemble	Runs the two-pass assembler on the current source and generates the hex/opcode workspace.

Control	Function
Disassemble	Reverse-translates loaded hex/machine code back into assembly mnemonics.
Run	Executes the assembled program from the current start address.
Step Back	Reverts the simulation state to the previous instruction (backward traversal).
Step Fwd	Advances the simulation by a single instruction (forward traversal).
Step	Executes the program one line at a time under manual control.
Pause	Suspends execution, preserving the current register and memory state.
Stop	Terminates the running simulation.
Reset	Restores registers, flags, and pointers to their initial state.
Autocorrect	Automatically corrects common syntax and formatting issues in the source code.

2.2 Navigation Sidebar

The left-hand sidebar switches the working view between the core areas of the IDE:

- Editor — the assembly source editor.
- Registers — the detailed, tabbed register and flag inspector.
- Memory — the Memory Editor.
- Devices — the I/O Port Editor / Interfacing Devices panel.
- Subroutine — quick access to the Delay Subroutine and Interrupt Service Subroutine wizards.
- Interrupts — direct control of TRAP, RST 7.5, RST 6.5, and RST 5.5 interrupt lines.

2.3 Start Screen

The centre pane of a new session presents three quick-start actions — Create New Program, Open Program, and Open Sample — alongside a Recent Files list and a Helpful Shortcuts reference summarising the most common keyboard shortcuts (for example, Ctrl+N to start a new program, F5 to assemble, F6 to run, and F7 to step).

2.4 Status Bar

The status bar at the bottom of the window continuously reports the cursor's line and column position, file encoding, source language, instruction and cycle counts, the current run state (Ready, Running, Paused, or Stopped), and the amount of memory currently in use by the loaded program.

3. Menu Reference

This chapter documents every command available from the main menu bar.

3.1 File Menu

Command	Shortcut	Description
Load Assembly Language code	Ctrl+O	Opens an existing .asm source file into the editor.
Load Intel HEX File (.hex)	Alt+O	Loads a program that has already been assembled into Intel HEX format.
Load Raw Binary File (.bin)	—	Loads a program stored as raw machine-code bytes.
Import Program Documentation	—	Imports accompanying notes for a program from .docx, .doc, or .txt files.
Save Assembly Language code	Ctrl+S	Saves the current source code to disk.
Save Intel HEX File (.hex)	Alt+S	Exports the assembled program in Intel HEX format.
Save Raw Binary File (.bin)	—	Exports the assembled program as raw binary.
Export Program Documentation	—	Exports program notes to .docx, .doc, or .txt.
Export to C Code (.c)	—	Converts the assembled 8085 program into equivalent C source.
Code to C Converter…	—	Opens the dedicated assembly-to-C conversion utility.
Print Editor Content	Ctrl+P	Prints the source code currently in the editor.
Print Workspace Content	Alt+P	Prints the assembled workspace (address, mnemonic, hex-code, cycles).
Recover last session	—	Restores the editor and workspace state from before an unexpected shutdown.
Create Practical	Ctrl+Shift+P	Packages the current program and documentation as a formatted lab practical.
Exit	Alt+F4	Closes Aura Studio.

3.2 Edit Menu

Command	Shortcut	Description
Undo	Ctrl+Z	Reverts the most recent editing action.
Redo	Ctrl+Y	Re-applies an undone editing action.
Cut	Ctrl+X	Removes the selected text and copies it to the clipboard.
Copy	Ctrl+C	Copies the selected text to the clipboard.

Command	Shortcut	Description
Paste	Ctrl+V	Inserts clipboard content at the cursor.
Find	Ctrl+F	Searches the source code for matching text.
Replace	Ctrl+R	Finds and replaces text within the source code.

3.3 Tools Menu

Command	Shortcut	Description
Assemble	F7	Runs the two-pass assembler over the current source.
Disassemble	F8	Converts loaded hex/machine code back to assembly mnemonics.
Autocorrect	F12	Automatically fixes common syntax issues in the source.
CPU Architecture Diagram	—	Opens the animated internal-architecture visualisation.
Format Code	Ctrl+Shift+F	Re-indents and formats the source code for readability.
Memory Heatmap & Visualizer	Ctrl+Shift+M	Opens the address-space activity heatmap.
Multi-Format Register Inspector	Ctrl+Shift+R	Opens the detailed register/flag/pointer inspector window.
7-Segment LED Display Unit	Ctrl+Shift+7	Opens the 7-segment display peripheral simulator.
8-Bit ADC & DAC Waveform Oscilloscope	Ctrl+Shift+A	Opens the ADC/DAC waveform peripheral simulator.
4-Way Traffic Light Controller	Ctrl+Shift+T	Opens the traffic-light controller peripheral simulator.
Stepper Motor Motion Simulator	Ctrl+Shift+P	Opens the stepper-motor peripheral simulator.
16x2 Character LCD Display (HD44780)	Ctrl+Shift+L	Opens the HD44780-compatible LCD peripheral simulator.

3.4 Settings Menu

Command	Shortcut	Description
Set Memory Range	Ctrl+Shift+M	Defines the address range shown and editable in the Memory Editor.
Simulation Speed	(submenu)	Selects the execution speed used by the simulation engine.
Clear Memory	Ctrl+Shift+Delete	Resets all memory locations to their default value.
Show/Hide Interface Editor	Ctrl+Shift+I	Toggles visibility of the I/O interface editor.

Command	Shortcut	Description
Show/Hide Editor	Ctrl+Shift+E	Toggles visibility of the source editor pane.
Show/Hide Workspace	Ctrl+Shift+W	Toggles visibility of the assembled workspace pane.
Stop simulation at mnemonic	Ctrl+Shift+T	Defines a mnemonic at which execution should automatically halt.

3.5 Simulation Menu

Command	Shortcut	Description
Run all at time	Alt+Up	Executes the entire program in a single continuous run.
Run step by step	(submenu)	Executes the program one instruction at a time, under manual control.
Stop	Alt+Down	Halts the currently running simulation.

3.6 Subroutine Menu

Command	Shortcut	Description
Interrupt Service Subroutine	—	Opens the wizard for configuring interrupt vector code.
Insert Delay Subroutine	—	Opens the wizard for generating a timed delay subroutine.

These tools are described in full in Chapter 8, “Subroutine Tools.”

3.7 View Menu

Command	Shortcut	Description
8085 microprocessor trainer kit	F9	Opens the graphical trainer-kit emulator.
Code Wizard	F6	Opens a guided, form-based tool for building assembly programs without typing mnemonics directly.

3.8 Load Sample Program Menu

Aura Studio ships with a library of ready-made teaching programs, accessible directly from the menu bar and searchable from the same menu:

- 1's complement of a 16-bit number
- 1's complement of an 8-bit number
- 2's complement of a 16-bit number

- 2's complement of an 8-bit number
- 8-bit division
- 8-bit multiplication
- 8-bit decimal subtraction
- Addition of two 16-bit numbers having a 16-bit or larger sum
- Addition of two 8-bit numbers having a 16-bit sum
- Addition of two 8-bit numbers
- Bubble sorting
- Decimal addition of two 8-bit numbers having a 16-bit sum
- Fibonacci series
- Masking off the least significant 4 bits of an 8-bit number
- Masking off the most significant 4 bits of an 8-bit number
- Sum of a series of multi-byte decimal numbers
- Shifting a 16-bit number left by 1 bit
- Shifting a 16-bit number left by 2 bits
- Shifting an 8-bit number left by 1 bit
- Shifting an 8-bit number left by 2 bits
- Subtraction of two 8-bit numbers
- Finding a square from a look-up table

A search field at the top of this menu allows a specific sample to be located by name.

3.9 Help Menu

Command	Shortcut	Description
Mnemonic Help	Ctrl+H	Opens reference documentation for every 8085 instruction mnemonic.
Documentation	F1	Opens this user manual.
About	F4	Displays application, version, and creator information.

4. Registers, Flags, Pointers & Execution Panel

Aura Studio keeps the full internal state of the simulated 8085 visible at all times. This state is presented in two forms: a compact live panel docked to the right of the editor, and a detailed Multi-Format Register Inspector window that can be opened at any time from the Tools menu, the sidebar, or Ctrl+Shift+R.

4.1 Live Panel

- **Registers —** shows the current value of the accumulator (A) and the general-purpose registers B, C, D, E, H, and L, plus the memory register (M) addressed by the H-L pair.
- **Flags —** shows the state of the Sign (S), Zero (Z), Auxiliary Carry (AC), Parity (P), and Carry (CY) flags.
- **Pointers —** shows the Program Counter (PC), Stack Pointer (SP), and the register-pair values HL, BC, and DE.
- **Execution —** reports the current instruction, execution state (Stopped/Running/Paused), Program Counter, instruction count, clock-cycle count, and machine-cycle count.
- **Interrupts —** shows the enabled/disabled status of TRAP, RST 7.5, and RST 6.5.

4.2 Multi-Format Register Inspector

This window presents every register alongside its binary bit pattern (bits 7 through 0), the Flag register with each individual flag bit labelled, and a consolidated status block covering the Stack Pointer, Memory Pointer (HL), Program Status Word (PSW), Program Counter, Clock Cycle Counter, and Instruction Counter.

Below this, two additional rows expose the interrupt-mask lines used by the RIM and SIM instructions: SOD, SID, INTR, TRAP, R7.5, R6.5, and R5.5 for RIM; and SOD, SDE, R7.5, MSE, M7.5, M6.5, and M5.5 for SIM. This lets a learner directly verify the effect of RIM/SIM instructions on the processor's interrupt-control logic during a step-through session.

5. Memory Editor

The Memory Editor gives direct, address-level access to the simulated 8085's memory space. It is opened from the Memory icon in the sidebar or via Settings → Set Memory Range.

5.1 Setting a Memory Range

A Memory Range field at the top of the editor accepts a start and end address (for example 3000 to FFFF), restricting the table below to that window of the address space. This keeps the view focused when working with a program loaded at a specific origin address.

5.2 Display Modes

The Memory Editor offers three interchangeable viewing modes, selectable from radio buttons at the bottom of the window:

- **Show entire memory content —** lists every address within the selected range, whether or not it currently holds meaningful data.
- **Show only loaded memory location —** restricts the table to addresses that have actually been written to by an assembled program, keeping the view compact.
- **Store directly to specified memory location —** allows a value to be typed directly against a chosen address, writing it straight into memory without going through the assembler.

Each row of the table shows a Memory Address and its corresponding Value; values can be edited in place, which is particularly useful for pre-loading input data before running a program, or for inspecting output data after execution.

6. I/O Port Editor (Interfacing Devices)

The I/O Port Editor, labelled Interfacing Devices in the application, provides direct access to all 256 possible 8085 I/O ports (00H–FFH), arranged in a 16×16 grid identical in layout to a hexadecimal memory map. It is opened from the Devices icon in the sidebar.

Each cell in the grid represents a single I/O port address and its current byte value. Because the 8085 communicates with external hardware through IN and OUT instructions directed at these ports, this editor is essential when working with peripheral interfacing exercises: values written by an OUT instruction during simulation appear immediately in the corresponding cell, and values can also be set manually here to simulate external input before running an IN instruction.

7. Tools

The Tools menu gathers Aura Studio's assembler utilities, diagnostic visualisations, and peripheral simulators.

7.1 Assembler and Disassembler

Aura Studio uses a two-pass assembler. On the first pass, it scans the source and builds a symbol table recording the memory location of every label used in the program. On the second pass, it resolves each partially defined mnemonic against that symbol table and converts every mnemonic into its corresponding opcode, producing the assembled hex workspace shown alongside the editor (address, label, mnemonic, hex-code, byte count, machine cycles, and T-states).

The Disassembler performs the reverse operation: it accepts machine code — either typed directly in Intel HEX format or loaded from a .hex file — and reconstructs the equivalent assembly-language mnemonics. Because raw machine code does not distinguish between opcode bytes and data bytes, the disassembler decodes every byte as an opcode by default, and it cannot automatically determine the intended execution start address; both should be set manually where the original program is not already known.

Assembler Directives

Directives are preprocessor instructions rather than executable mnemonics; in the editor they are written with a leading '#' or '.' and are highlighted accordingly. The supported directives are:

Directive	Example	Description
ORG	# ORG C000H	Sets the memory location where the next block of code will be stored.
BEGIN	# BEGIN 2000H	Sets the address at which simulation should start.
END	# END	Marks the end of the assembly; equivalent to the mnemonic set under Settings → Stop simulation at mnemonic.
EQU	# OUTBUF EQU 3945H	Assigns a fixed value to a label, which may then be used as a memory reference.
DB	# DATA: DB F5H,12H	Defines one or more bytes at successive memory locations, starting at the given label.
DW	# LABEL: DW 2050H	Defines a value two bytes at a time.
DS	# STACK: DS 4	Reserves a specified number of memory locations starting at the given label.

Number Entry Formats

Numeric literals may be entered in binary, decimal, or hexadecimal, distinguished by a trailing suffix:

- Binary — digits limited to 0 and 1, suffixed with 'B' or 'b' (e.g. 01111B).

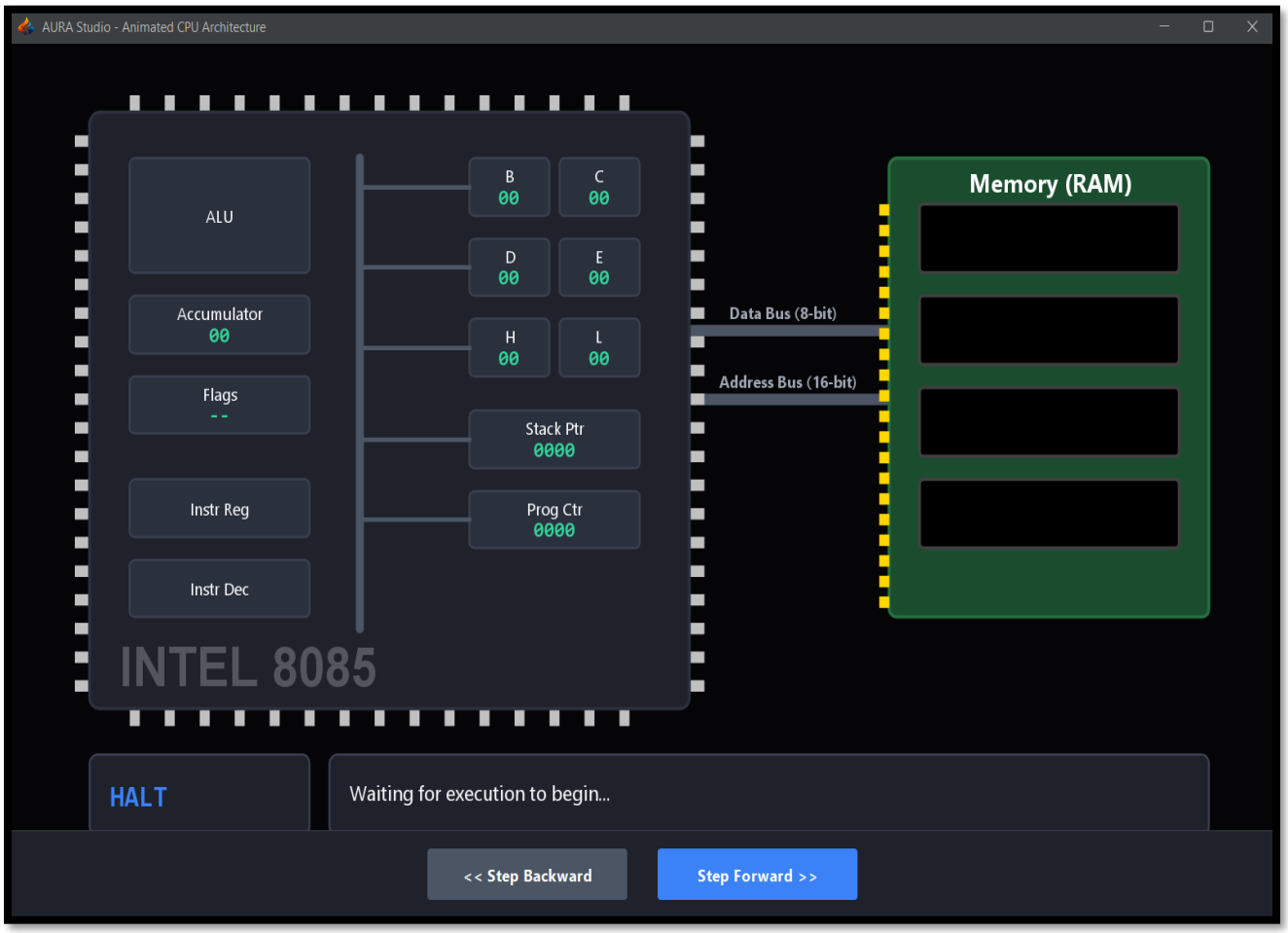
- Decimal — digits 0–9, suffixed with 'D' or 'd' (e.g. 0015D).
- Hexadecimal — digits 0–9 and A–F, optionally suffixed with 'H' or 'h' (e.g. 0FH); this is the default format when no suffix is present.

7.2 Autocorrect

Autocorrect automatically fixes common formatting mistakes in the source — such as inconsistent spacing, casing, or label punctuation — without altering the logic of the program, reducing the number of trivial assembler errors a learner has to diagnose manually.

7.3 CPU Architecture Diagram

This tool opens an animated block diagram of the Intel 8085, showing the ALU, Accumulator, Flags register, Instruction Register, Instruction Decoder, the B-C, D-E, and H-L register pairs, the Stack Pointer, and the Program Counter, connected via an internal bus to an 8-bit Data Bus and a 16-bit Address Bus leading to main memory (RAM). Step Backward and Step Forward controls beneath the diagram animate the movement of data and addresses across these buses as the program executes, and a status line reports the current HALT state and a description of the operation in progress. This makes the diagram a valuable teaching aid for visualising how an instruction actually moves through the processor during fetch, decode, and execute.



7.4 Format Code

Re-indents and re-aligns the source editor's content into a consistent column layout for labels, mnemonics, operands, and comments, improving readability without changing program behaviour.

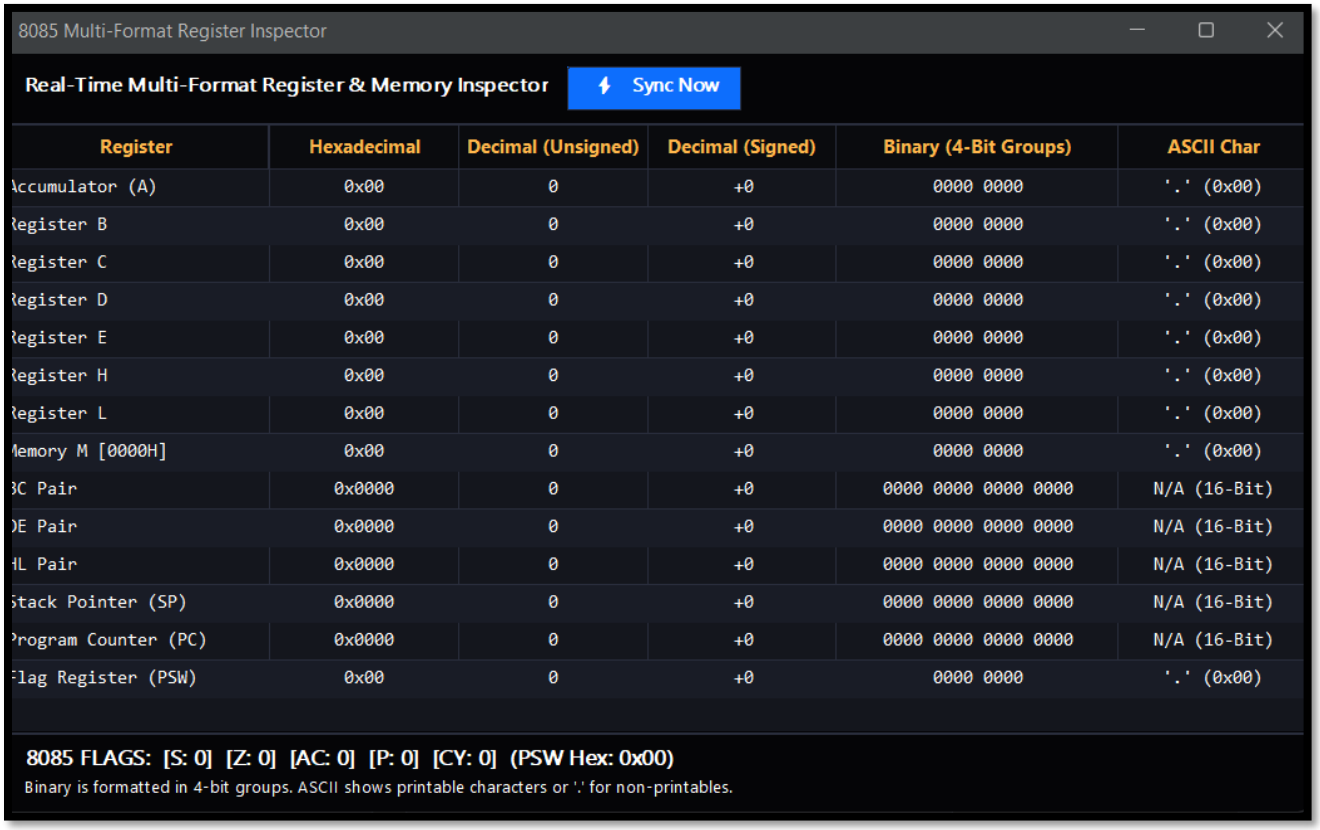
7.5 Memory Heatmap & Visualizer

The Memory Heatmap & Visualizer renders the full 64 KB address space (or a single 256-byte page at a time) as a grid of cells, each showing an address and its current hexadecimal value. Three independent tracking layers can be toggled on or off: Code Exec, Memory Read, and Memory Write, allowing a user to isolate exactly how a program touches memory during execution.

Selecting any cell reveals its full detail in the status bar beneath the grid: address, hexadecimal value, decimal value, binary value, the time of last access, and running counters for the number of Execute (E), Read (R), and Write (W) accesses that address has received. A colour-coded legend distinguishes Execution Byte, Read Access, Write Access, and Unaccessed cells, and a Clear Activity button resets all counters. A Switch to 64KB Full Map control expands the view from a single page to the entire address space.

7.6 Multi-Format Register Inspector

See Section 4.2 for a full description of this window.



7.7 Peripheral Simulators

Aura Studio includes a set of interactive, port-driven peripheral simulators that mirror common 8085 interfacing exercises. Each simulator reads from and writes to the same I/O ports exposed in the Interfacing Devices editor (Chapter 6), so any program that performs standard IN/OUT operations on the configured ports will drive the peripheral directly.

7-Segment LED Display Unit

Simulates a bank of 7-segment LED digits (configurable digit count) driven by a data port and a digit-select port. Configuration options include the Data Port and Select Port addresses, Decode Mode (for example, Direct BCD/Hex Decoder), digit count, LED colour, and polarity (common cathode or common anode). A Sample Code selector and Load Sample / Load & Run Code buttons let a ready-made demonstration program — such as a BCD 00–99 counter — be loaded and executed against the display directly.

8-Bit ADC & DAC Waveform Oscilloscope

Provides a simulated 8-bit analog-to-digital and digital-to-analog conversion path with an oscilloscope-style waveform display, allowing programs that generate or sample analog-style waveforms through I/O ports to be visualised in real time.

4-Way Traffic Light Controller

Simulates a four-direction traffic-light intersection, with each set of lights mapped to specific I/O port bits — a classic 8085 interfacing exercise for practising timed, sequential port output.

Stepper Motor Motion Simulator

Simulates a stepper motor whose coils are driven by an I/O port, visualising rotor movement in response to the step sequences written by a program — commonly used to teach motor-control interfacing.

16×2 Character LCD Display (HD44780)

Simulates a standard 16×2 character LCD module compatible with the HD44780 controller, driven through data and control ports, for practising character-display interfacing routines.

8. Subroutine Tools

Accessible from the Subroutine menu or the sidebar, these wizards generate ready-to-use blocks of assembly code for two common tasks: timed delays and interrupt vector configuration.

8.1 Insert Delay Subroutine

This wizard generates a complete delay subroutine, sized to a user-specified delay time, for a chosen operating frequency of the 8085. To use it, a Delay Subroutine Label is entered, the required Delay (in milliseconds) is specified, and the Time for 1 T-state (in nanoseconds) is set to match the target clock frequency. A Format of Subroutine selector then determines how many registers the generated loop should use:

- Delay Subroutine using One Register
- Delay Subroutine using Two Registers
- Delay Subroutine using Three Registers
- Delay Subroutine using Register Pair

Once a format is chosen, the specific registers to use are selected from dropdown lists — the left-most register carries the highest loop priority. The dialog automatically calculates and displays the minimum and maximum delay achievable with the chosen configuration, so the requested delay can be validated before code is generated. Pressing OK inserts the finished subroutine directly into the editor at the given label, ready to be called from the main program.

8.2 Interrupt Service Subroutine

This wizard provides a direct way to populate the fixed interrupt vector locations of the 8085 with a branch instruction pointing at a user-defined interrupt service routine, without needing to hand-calculate vector addresses.

The dialog lists every interrupt and its corresponding call location:

Interrupt	Call Location
TRAP	0024H
RST 7.5	003CH
RST 6.5	0034H
RST 5.5	002CH
RST 0	0000H
RST 1	0008H
RST 2	0010H
RST 3	0018H

Interrupt	Call Location
RST 4	0020H
RST 5	0028H
RST 6	0030H
RST 7	0038H

Against any interrupt, a branch instruction — typically a JMP to a label already defined in the assembled program — can be typed directly into the corresponding field. On pressing Enter, the label is resolved to its numeric address, and once the dialog is closed, the required opcode bytes are written straight into memory at that interrupt's call location, ready to be triggered from the Interrupts panel or by the executing program.

9. Simulation & Debugging

9.1 Simulation Modes

Aura Studio's simulation engine can be driven in two ways, both available from the Simulation menu and the main toolbar:

- **Run all at a time —** executes the assembled program from start to finish (or until a breakpoint or stop-mnemonic is reached) in a single continuous run.
- **Run step by step —** executes the program one instruction at a time under manual control, updating every register, flag, and memory panel after each step so its effect can be examined individually.

A simulation in progress can be paused, stopped, or reset at any time from the toolbar.

9.2 Breakpoints and Debug Mode

Any line in the assembled workspace can be marked as a breakpoint, shown by a marker beside that line. When the simulation reaches a marked line during a full run, it automatically halts and switches into manual, step-wise control — at which point Backward and Forward controls become available, letting execution be stepped in either direction from that point. A Continue control resumes the run until the next breakpoint (or the end of the program) is reached.

This combination of forward and backward traversal is particularly useful for debugging: because Aura Studio preserves the full memory and register state at each step, an unexpected result can be traced back instruction by instruction to find the point at which the fault was introduced, rather than only being able to move forward through the program.

9.3 Simulation Speed

Under Settings → Simulation Speed, the rate of a full run can be adjusted between a fully manual step-by-step pace, a normal speed that reflects intermediate states periodically, and an ultimate speed that jumps directly to the final state of the program.

9.4 Interrupts

TRAP, RST 7.5, RST 6.5, and RST 5.5 can be triggered directly from the Interrupts panel in the sidebar, in addition to being generated by program logic, allowing interrupt-handling code to be tested interactively during a simulation run.

10. Trainer Kit Emulator & Code Wizard

10.1 8085 Microprocessor Trainer Kit

Opened from the View menu (or F9), the Trainer Kit Emulator reproduces the keypad-and-display workflow of a physical 8085 trainer board, using the same simulation engine as the main editor — a user can freely switch between the Editor view and the Trainer Kit view at any point in a session without losing state.

The emulated keypad provides 16 hexadecimal digit keys (0–F) plus dedicated function keys:

Key	Function
RESET	Terminates the current execution without clearing register or memory contents.
HALT	Pauses execution at the current instruction, allowing forward or backward stepping from that point.
DCR	Decrements the currently displayed memory address and shows its new contents.

Key	Function
INR	Increments the currently displayed memory address and shows its new contents.
SET/MEM	Enters or edits the contents of a specific memory address.
REG	Displays the current contents of a chosen register.
GO	Sets the starting address for execution.
EXEC	Begins execution from the address set by GO, at the simulation speed configured in the editor.

A program is entered by pressing SET/MEM, typing the target address, and then using INR together with the hexadecimal keys to enter each successive opcode byte. Execution begins by pressing GO, entering the start address, and pressing EXEC.

10.2 Code Wizard

The Code Wizard (View menu, or F6) offers a guided, form-based way to build 8085 assembly programs without typing raw mnemonics directly, making it a useful on-ramp for users who are still becoming familiar with the instruction set.

11. About Aura Studio

The About dialog (Help → About, or F4) identifies the application and its development team:

Field	Value
Application	AURA STUDIO
Description	8085 Microprocessor Simulator & Assembly IDE
Creators	Dev Gondaliya, Dev Letwala, Jigar Ghoghari

11.1 Development Team

Aura Studio is designed and developed by:

- Dev Gondaliya
- Dev Letwala
- Jigar Ghoghari

12. Intel 8085 Microprocessor Architecture

This chapter is drawn from the AURA STUDIO Technical Reference & Programming Manual and documents the underlying Intel 8085 architecture that the simulation engine reproduces: the register set, the flag layout, and the processor's pin-level signal classification.

12.1 Microprocessor Architecture Overview

The Intel 8085 is an 8-bit NMOS microprocessor with a 16-bit address bus capable of directly addressing 65,536 bytes (64 KB) of memory space (0000H to FFFFH) and 256 independent Input/Output (I/O) ports (00H to FFH). The CPU engine comprises the Accumulator (A) and Flags (PSW), the general-purpose register pairs B-C, D-E, and H-L, a 16-bit Stack Pointer (SP), and a 16-bit Program Counter (PC), all connected to memory and the I/O port bus through the internal 8085 system bus.

12.2 CPU Register Specifications

Register	Bit Width	Primary Function & Hardware Description
A	8-Bit	Accumulator: primary register for arithmetic, logic, load/store operations, and all direct I/O transfers via IN and OUT instructions.
B, C	8-Bit / 16-Bit	BC Register Pair: general-purpose 8-bit registers, paired as BC for 16-bit data operations and loop counters.
D, E	8-Bit / 16-Bit	DE Register Pair: general-purpose 8-bit registers, paired as DE for 16-bit memory addressing and data manipulation.

Register	Bit Width	Primary Function & Hardware Description
H, L	8-Bit / 16-Bit	HL Register Pair / Pointer M: primary memory pointer register pair. Operands referencing memory location M use the 16-bit address held in HL.
SP	16-Bit	Stack Pointer: holds the 16-bit RAM address pointing to the current top of the system call stack (default FFFFH).
PC	16-Bit	Program Counter: holds the 16-bit address of the next instruction opcode byte to be fetched from memory.
PSW	8-Bit	Program Status Word (Flags): contains five hardware condition flags updated after arithmetic and logic operations.

12.3 Flag Register (PSW) Bit Layout

The Program Status Word occupies bits D7 through D0, laid out as S, Z, X, AC, X, P, X, CY (where X marks an unused bit):

- S (Sign Flag, bit 7) — set to 1 if the most significant bit of the result is 1 (negative in two's-complement form).
- Z (Zero Flag, bit 6) — set to 1 if the result equals zero (00H).
- AC (Auxiliary Carry Flag, bit 4) — set to 1 if a carry is generated out of bit D3 into bit D4 during addition; used by the DAA instruction.
- P (Parity Flag, bit 2) — set to 1 if the result contains an even number of 1 bits.
- CY (Carry Flag, bit 0) — set to 1 if an arithmetic operation generates a carry out of bit D7.

12.4 8085 Pin Architecture & Signal Classification

The 8085 is packaged in a 40-pin Dual In-line Package (DIP). Its pins fall into the following functional groups:

- Address Bus (A8–A15) — pins 21 to 28, output, carrying the high-order 8 bits of the memory address.
- Multiplexed Address/Data Bus (AD0–AD7) — pins 12 to 19, input/output, carrying the low-order 8-bit address followed by the 8-bit data byte.
- ALE (Address Latch Enable, pin 30) — a positive pulse indicating that AD0–AD7 currently holds the low-order address.
- RD (Read Control, pin 32) and WR (Write Control, pin 31) — active-low signals identifying a memory/IO read or write cycle.
- IO/M (pin 34) — high indicates an I/O cycle; low indicates a memory cycle.
- S0, S1 (Status Signals, pins 29 and 33) — together identify the current cycle: S1=1,S0=1 is an opcode fetch; S1=1,S0=0 is a read; S1=0,S0=1 is a write.
- Interrupt Signals — TRAP (pin 4, non-maskable), RST 7.5 (pin 5), RST 6.5 (pin 6), RST 5.5 (pin 7), INTR (pin 10), and INTA (pin 11).

13. Complete 8085 Instruction Set Reference

The 8085 instruction set consists of 74 distinct instructions yielding 246 machine opcodes, classified into five functional groups documented below. Bytes and T-states are shown for each mnemonic as implemented by the simulation engine.

13.1 Data Transfer Instruction Group

Mnemonic	Bytes	T-States	Flags	Description
MOV r1, r2	1	4	None	Move data from register r2 to r1. Example: MOV A,B copies B into A.
MOV r, M	1	7	None	Move byte from memory location HL to register r.
MOV M, r	1	7	None	Move byte from register r to memory location HL.
MVI r, data	2	7	None	Move 8-bit immediate data into register r.
MVI M, data	2	10	None	Move 8-bit immediate data into memory location HL.
LXI rp, data16	3	10	None	Load 16-bit immediate data into register pair rp (BC, DE, HL, SP).
LDA addr	3	13	None	Load Accumulator directly from a 16-bit memory address.
STA addr	3	13	None	Store Accumulator directly into a 16-bit memory address.
LHLD addr	3	16	None	Load register L from addr and register H from addr+1.
SHLD addr	3	16	None	Store register L into addr and register H into addr+1.
LDAX rp	1	7	None	Load Accumulator indirect from the address in register pair BC or DE.
STAX rp	1	7	None	Store Accumulator indirect into the address in register pair BC or DE.
XCHG	1	4	None	Exchange the 16-bit contents of register pairs DE and HL.

13.2 Arithmetic Instruction Group

Mnemonic	Bytes	T-States	Flags	Description
ADD r	1	4	All	Add contents of register r to Accumulator A.

Mnemonic	Bytes	T-States	Flags	Description
ADD M	1	7	All	Add the byte at memory location HL to Accumulator A.
ADI data	2	7	All	Add 8-bit immediate data to Accumulator A.
ADC r	1	4	All	Add register r plus the Carry flag to Accumulator A.
SUB r	1	4	All	Subtract register r from Accumulator A.
SUI data	2	7	All	Subtract 8-bit immediate data from Accumulator A.
SBB r	1	4	All	Subtract register r and the Carry flag from Accumulator A.
INR r	1	4	S, Z, AC, P	Increment register r by 1 (Carry flag unaffected).
DCR r	1	4	S, Z, AC, P	Decrement register r by 1 (Carry flag unaffected).
INX rp	1	6	None	Increment 16-bit register pair rp by 1.
DCX rp	1	6	None	Decrement 16-bit register pair rp by 1.
DAD rp	1	10	CY	Add 16-bit register pair rp to the HL register pair.
DAA	1	4	All	Decimal-adjust the Accumulator after addition to format packed BCD.

13.3 Logical Instruction Group

Mnemonic	Bytes	T-States	Flags	Description
ANA r	1	4	S,Z,P,AC=1,CY=0	Logical AND register r with Accumulator A.
ANI data	2	7	S,Z,P,AC=1,CY=0	Logical AND 8-bit immediate data with Accumulator A.
ORA r	1	4	S,Z,P,AC=0,CY=0	Logical OR register r with Accumulator A.
ORI data	2	7	S,Z,P,AC=0,CY=0	Logical OR 8-bit immediate data with Accumulator A.
XRA r	1	4	S,Z,P,AC=0,CY=0	Logical XOR register r with Accumulator A. XRA A clears A.
CMP r	1	4	All	Compare register r with A, setting flags as if A minus r were computed.

Mnemonic	Bytes	T-States	Flags	Description
CPI data	2	7	All	Compare 8-bit immediate data with Accumulator A.
RLC	1	4	CY	Rotate Accumulator left by 1 bit; bit D7 moves into D0 and into Carry.
RRC	1	4	CY	Rotate Accumulator right by 1 bit; bit D0 moves into D7 and into Carry.
RAL	1	4	CY	Rotate Accumulator left through the Carry flag.
RAR	1	4	CY	Rotate Accumulator right through the Carry flag.
CMA	1	4	None	Complement the Accumulator (one's complement of A).
CMC	1	4	CY	Complement the Carry flag.
STC	1	4	CY=1	Set the Carry flag.

13.4 Branching & Program Control Group

Mnemonic	Bytes	T-States	Flags	Description
JMP addr	3	10	None	Unconditional jump to a 16-bit memory address.
JNZ addr	3	10/7	None	Jump to addr if the Zero flag is 0 (result not zero).
JZ addr	3	10/7	None	Jump to addr if the Zero flag is 1 (result is zero).
JNC addr	3	10/7	None	Jump to addr if the Carry flag is 0.
JC addr	3	10/7	None	Jump to addr if the Carry flag is 1.
CALL addr	3	18	None	Call a subroutine at addr, pushing the return address onto the stack.
RET	1	10	None	Return from a subroutine, popping a 16-bit address from the stack into PC.

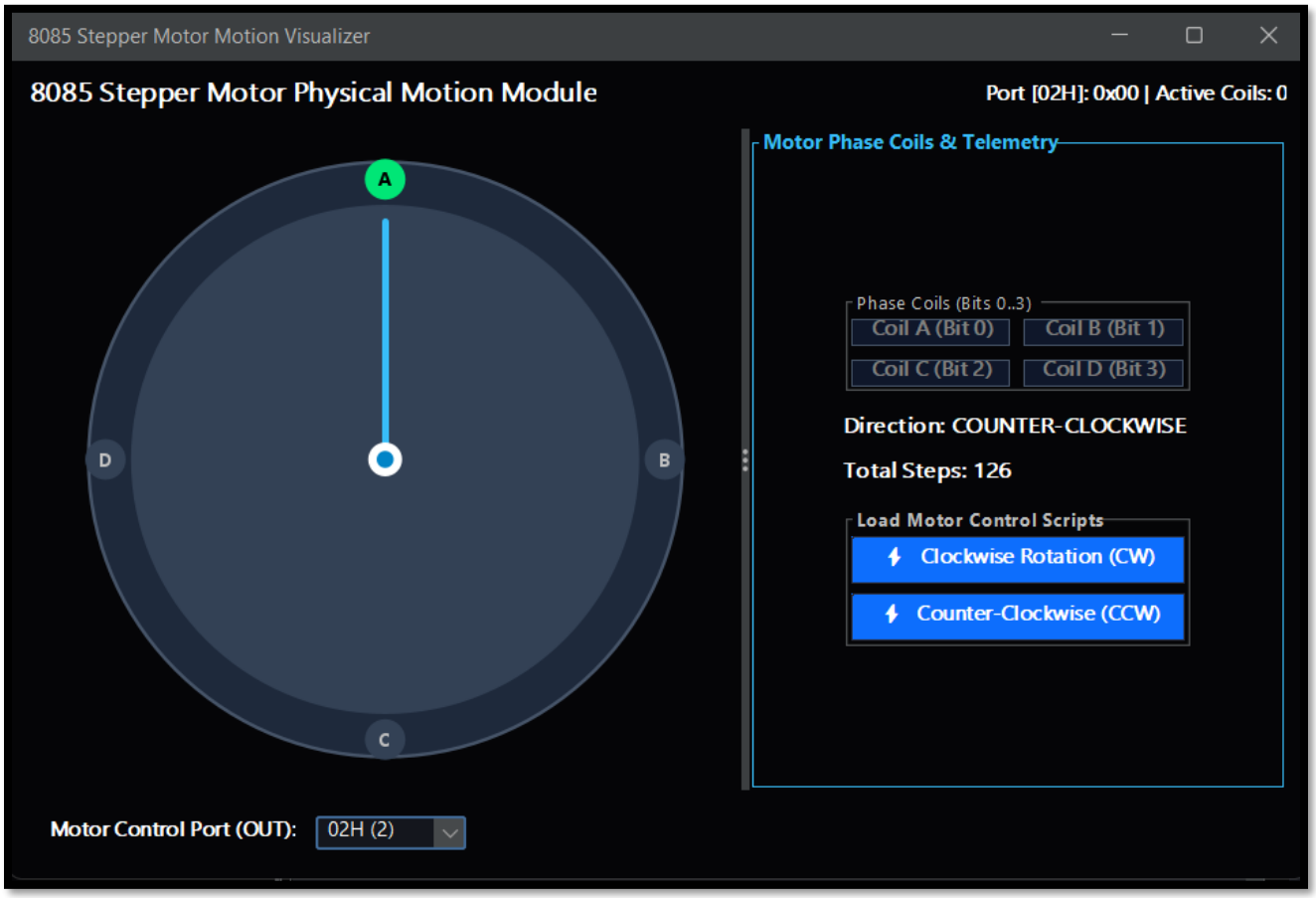
13.5 Stack, I/O & Machine Control Group

Mnemonic	Bytes	T-States	Flags	Description
PUSH rp	1	12	None	Push a 16-bit register pair rp onto the system stack.

Mnemonic	Bytes	T-States	Flags	Description
POP rp	1	10	None (all for PSW)	Pop 16-bit data from the system stack into register pair rp.
IN port	2	10	None	Read an 8-bit digital value from an I/O port into Accumulator A.
OUT port	2	10	None	Write the Accumulator A contents to a peripheral I/O port.
NOP	1	4	None	No operation; consumes 4 T-states without altering processor state.
HLT	1	5	None	Halt CPU execution; the bus enters a stop state.

14. Stepper Motor Motion Visualizer

The Stepper Motor Visualizer simulates a 4-phase unipolar stepper motor. The stator consists of four field coils (Phases A, B, C, D) positioned at 90-degree intervals around a permanent-magnet rotor, driven entirely through a single output port.



14.1 Port Mapping & Bit Assignments

Target Data Port: 00H (configurable from 00H to FFH). Bit allocation:

- Bit 0 (01H) — Phase A stator coil.
- Bit 1 (02H) — Phase B stator coil.
- Bit 2 (04H) — Phase C stator coil.
- Bit 3 (08H) — Phase D stator coil.
- Bits 4–7 — unused / reserved.

14.2 Excitation Sequence Reference

Three drive modes are supported: Full-Step Clockwise, Full-Step Counter-Clockwise, and Half-Stepping (which interleaves single- and dual-coil states for finer rotor resolution).

Drive Mode	Step	Port Value	Active Coil(s)	Rotor Angle
Full-Step CW	1	01H	Phase A	0°
Full-Step CW	2	02H	Phase B	90°
Full-Step CW	3	04H	Phase C	180°
Full-Step CW	4	08H	Phase D	270°
Full-Step CCW	1–4	08H → 04H → 02H → 01H	D → C → B → A	270° → 0°
Half-Stepping	1	01H	Phase A	0°
Half-Stepping	2	03H	A + B	45°
Half-Stepping	3	02H	Phase B	90°
Half-Stepping	4	06H	B + C	135°
Half-Stepping	5	04H	Phase C	180°
Half-Stepping	6	0CH	C + D	225°
Half-Stepping	7	08H	Phase D	270°
Half-Stepping	8	09H	D + A	315°

14.3 Sample Assembly Source

The listing below drives continuous full-step clockwise rotation followed by an 8-stage half-step microstepping sequence, both against Port 00H.

```
; STEPPER MOTOR CONTROLLER — Target Port: 00H
ROTATE_CW:
    MVI A, 01H
    OUT 00H
    CALL STEP_DELAY
    MVI A, 02H
    OUT 00H
    CALL STEP_DELAY
    MVI A, 04H
    OUT 00H
    CALL STEP_DELAY
    MVI A, 08H
    OUT 00H
    CALL STEP_DELAY
    JMP ROTATE_CW

HALF_STEP_MICRO:
    MVI A, 01H \ OUT 00H \ CALL STEP_DELAY
    MVI A, 03H \ OUT 00H \ CALL STEP_DELAY
    MVI A, 02H \ OUT 00H \ CALL STEP_DELAY
    MVI A, 06H \ OUT 00H \ CALL STEP_DELAY
    MVI A, 04H \ OUT 00H \ CALL STEP_DELAY
```

```
    MVI A, 0CH \ OUT 00H \ CALL STEP_DELAY
    MVI A, 08H \ OUT 00H \ CALL STEP_DELAY
    MVI A, 09H \ OUT 00H \ CALL STEP_DELAY
    JMP HALF_STEP_MICRO

STEP_DELAY:
    MVI C, 01H
D_LOOP:
    DCR C
    JNZ D_LOOP
    RET
```

15. 7-Segment LED Display Unit

The 7-segment display consists of seven individually addressable LED segments, labelled a through g, plus an optional decimal point (dp), all driven from a single output port.

15.1 Segment Bit Mapping

- Bit 0 (01H) — Segment a (top horizontal).
- Bit 1 (02H) — Segment b (top-right vertical).
- Bit 2 (04H) — Segment c (bottom-right vertical).
- Bit 3 (08H) — Segment d (bottom horizontal).
- Bit 4 (10H) — Segment e (bottom-left vertical).
- Bit 5 (20H) — Segment f (top-left vertical).
- Bit 6 (40H) — Segment g (middle horizontal).
- Bit 7 (80H) — Segment dp (decimal point).

15.2 Segment Bitmask Lookup Table

Glyph	Active Segments	Hex Code	Binary
0	a,b,c,d,e,f	3FH	0011 1111

Glyph	Active Segments	Hex Code	Binary
1	b,c	06H	0000 0110
2	a,b,d,e,g	5BH	0101 1011
3	a,b,c,d,g	4FH	0100 1111
4	b,c,f,g	66H	0110 0110
5	a,c,d,f,g	6DH	0110 1101
6	a,c,d,e,f,g	7DH	0111 1101
7	a,b,c	07H	0000 0111
8	a,b,c,d,e,f,g	7FH	0111 1111
9	a,b,c,d,f,g	6FH	0110 1111
A	a,b,c,e,f,g	77H	0111 0111
b	c,d,e,f,g	7CH	0111 1100
C	a,d,e,f	39H	0011 1001
d	b,c,d,e,g	5EH	0101 1110
E	a,d,e,f,g	79H	0111 1001
F	a,e,f,g	71H	0111 0001
H	b,c,e,f,g	76H	0111 0110
L	d,e,f	38H	0011 1000
P	a,b,e,f,g	73H	0111 0011
–	g	40H	0100 0000

15.3 Sample Assembly Source

This multiplexed 4-digit driver writes the value “1234” across four digit positions, using Port 00H for segment data and Port 01H for digit select.

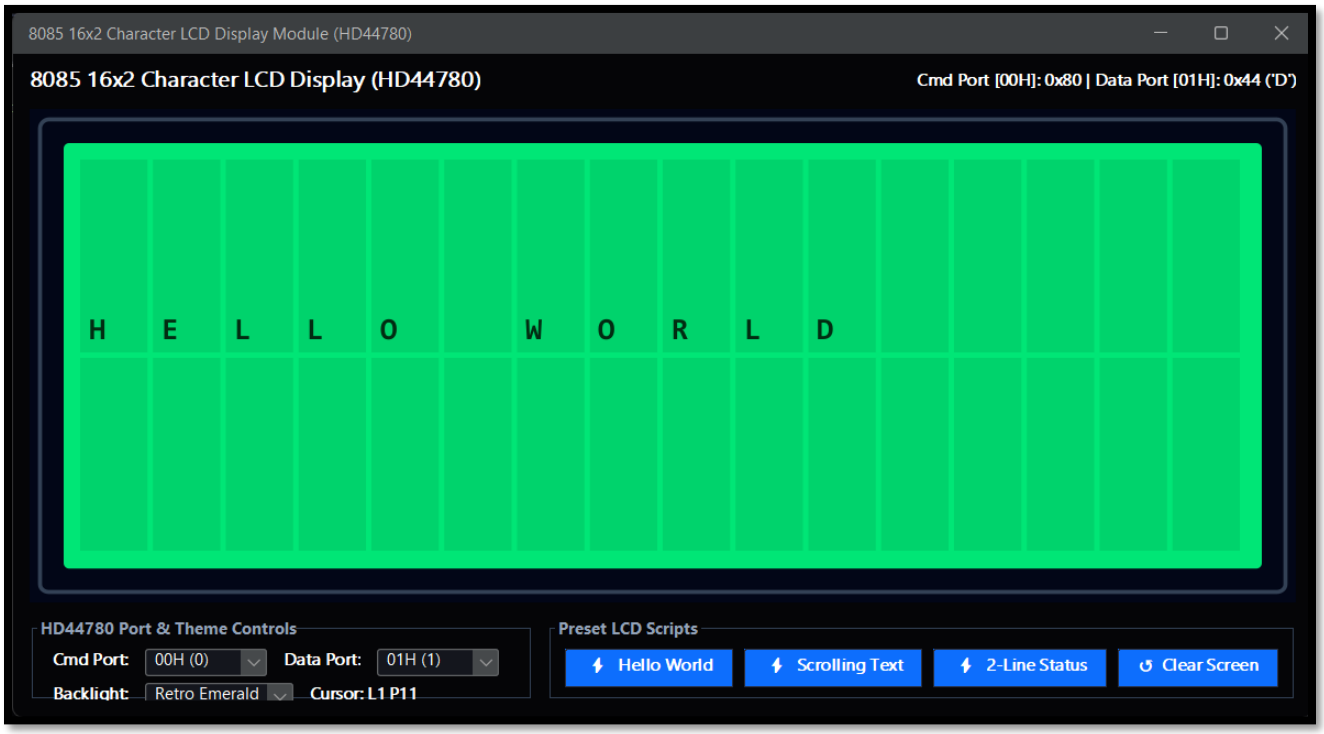
```
; 7-SEGMENT DISPLAY — MULTIPLEXED 4-DIGIT DRIVER ('1234')
; Port 00H: Segment Data | Port 01H: Digit Select
DISPLAY_LOOP:
    MVI A, 04H \ OUT 00H     ; BCD value 4
    MVI A, 01H \ OUT 01H     ; Select Digit 0 -> shows '4'
    CALL SHORT_DELAY
    MVI A, 03H \ OUT 00H
    MVI A, 02H \ OUT 01H     ; Select Digit 1 -> shows '3'
    CALL SHORT_DELAY
    MVI A, 02H \ OUT 00H
    MVI A, 04H \ OUT 01H     ; Select Digit 2 -> shows '2'
    CALL SHORT_DELAY
    MVI A, 01H \ OUT 00H
    MVI A, 08H \ OUT 01H     ; Select Digit 3 -> shows '1'
```

```
    CALL SHORT_DELAY
    JMP DISPLAY_LOOP

SHORT_DELAY:
    MVI E, 01H
D1: DCR E \ JNZ D1
    RET
```

16. 16×2 Character LCD Module (HD44780)

This simulator reproduces a 16-character by 2-line Hitachi HD44780-compatible LCD module, addressed through a data/command port and a control port.



16.1 System Architecture & Pins

- Data/Command Port (00H) — sends an 8-bit command byte or an ASCII character data byte.
- Control Port (01H), Bit 0 (RS — Register Select) — 0 selects the Instruction Register (command), 1 selects the Data Register (RAM).
- Control Port (01H), Bit 1 (R/W — Read/Write) — 0 for write, 1 for read.
- Control Port (01H), Bit 2 (E — Enable Pulse) — triggers the latch.

16.2 Command Set & DDRAM Mapping

Command Code	Description	Visual Screen Reaction
01H	Clear Display	Clears all 32 character cells and resets the DDRAM address to 00H.
02H	Return Home	Moves the cursor to Line 1, Column 0 (00H).
06H	Entry Mode Set	Configures the cursor to auto-increment right after each write.

Command Code	Description	Visual Screen Reaction
0CH	Display ON, Cursor OFF	Turns the screen on without a visible cursor underline.
0EH	Display ON, Cursor ON	Turns the screen on with a visible cursor underline.
38H	Function Set (8-bit, 2 lines)	Configures an 8-bit bus width, two display lines, and a 5x8 font.
80H	Force Cursor to Line 1	Sets the DDRAM address to Line 1, Column 0 (80H + 00H).
C0H	Force Cursor to Line 2	Sets the DDRAM address to Line 2, Column 0 (80H + 40H).

16.3 Sample Assembly Source

The following listing initialises the module and writes “8085 CPU” to Line 1, using Port 00H for data/command and Port 01H (bit 0 = RS) for control.

```
; 16x2 LCD DRIVER – WRITE "8085 CPU" TO LINE 1
; Port 00H: Data/Command | Port 01H: Control (Bit 0 = RS)

; Initialise (RS = 0, command mode)
MVI A, 38H \ OUT 00H \ MVI A, 00H \ OUT 01H  ; Function Set
MVI A, 0EH \ OUT 00H \ MVI A, 00H \ OUT 01H  ; Display ON
MVI A, 01H \ OUT 00H \ MVI A, 00H \ OUT 01H  ; Clear Screen

; Write characters (RS = 1, data mode)
MVI A, 38H \ OUT 00H \ MVI A, 01H \ OUT 01H  ; '8'
MVI A, 30H \ OUT 00H \ MVI A, 01H \ OUT 01H  ; '0'
MVI A, 38H \ OUT 00H \ MVI A, 01H \ OUT 01H  ; '8'
MVI A, 35H \ OUT 00H \ MVI A, 01H \ OUT 01H  ; '5'
MVI A, 20H \ OUT 00H \ MVI A, 01H \ OUT 01H  ; ' '
MVI A, 43H \ OUT 00H \ MVI A, 01H \ OUT 01H  ; 'C'
MVI A, 50H \ OUT 00H \ MVI A, 01H \ OUT 01H  ; 'P'
MVI A, 55H \ OUT 00H \ MVI A, 01H \ OUT 01H  ; 'U'
HLT
```

17. 4-Way Traffic Light Controller Simulator

Simulates a four-way urban road intersection with independent North-South and East-West signal controllers, each mapped to its own output port.

17.1 Intersection Topology & Bit Mapping

North-South Port: 00H. East-West Port: 01H.

- Bit 0 (01H) — RED light (stop traffic).
- Bit 1 (02H) — AMBER light (slow down / prepare).
- Bit 2 (04H) — GREEN light (proceed traffic).
- Bit 3 (08H) — pedestrian WALK signal.

17.2 4-Phase Traffic Sequence Matrix

Phase	N-S Port (00H)	E-W Port (01H)	N-S Light	E-W Light
1	04H	01H	GREEN	RED
2	02H	01H	AMBER	RED
3	01H	04H	RED	GREEN
4	01H	02H	RED	AMBER

17.3 Sample Assembly Source

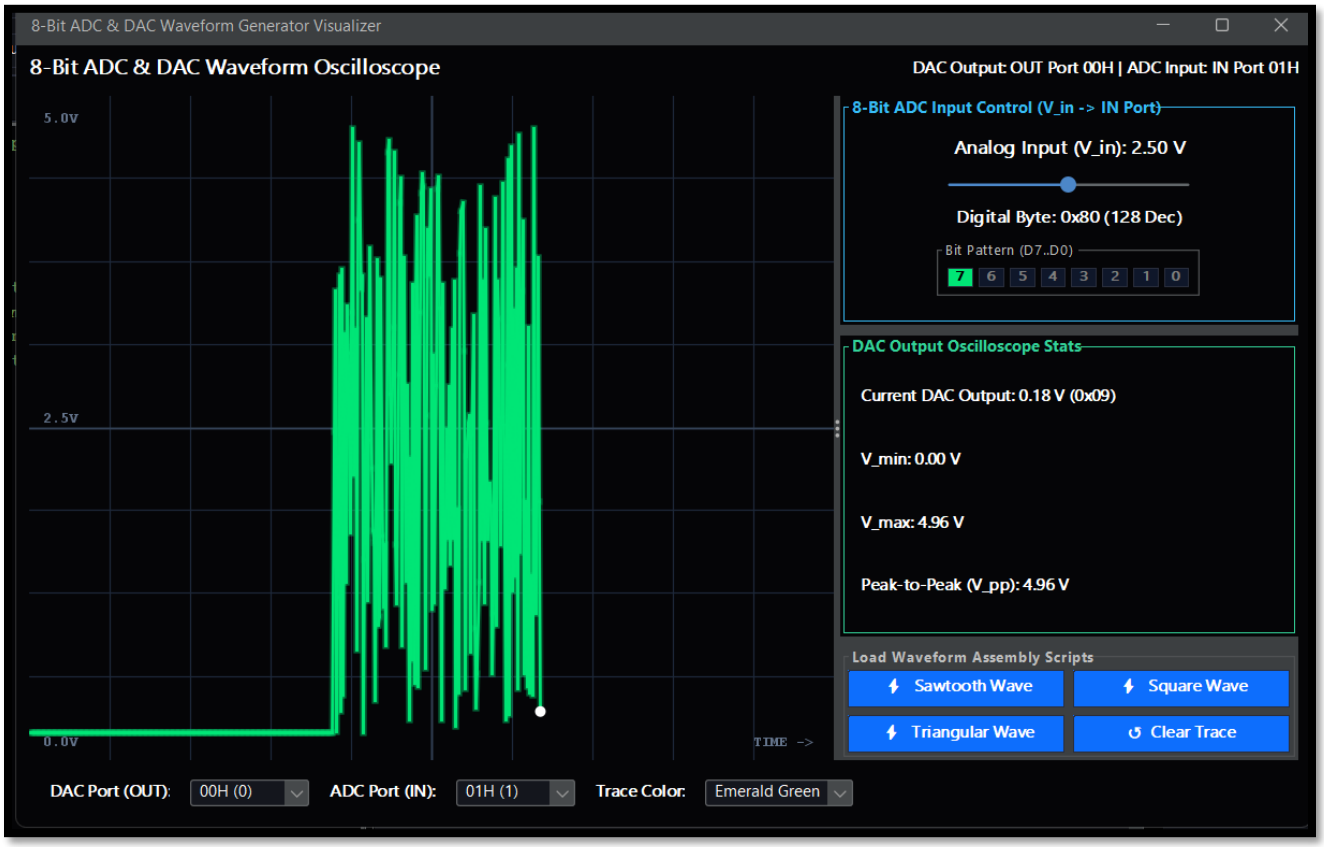
```
; 4-WAY TRAFFIC LIGHT CONTROLLER
; Port 00H: North-South | Port 01H: East-West
TRAFFIC_LOOP:
    ; Phase 1: N-S GREEN (04H), E-W RED (01H)
    MVI A, 04H \ OUT 00H
    MVI A, 01H \ OUT 01H
    CALL LONG_DELAY
    ; Phase 2: N-S AMBER (02H), E-W RED (01H)
    MVI A, 02H \ OUT 00H
    MVI A, 01H \ OUT 01H
    CALL SHORT_DELAY
    ; Phase 3: N-S RED (01H), E-W GREEN (04H)
    MVI A, 01H \ OUT 00H
    MVI A, 04H \ OUT 01H
    CALL LONG_DELAY
    ; Phase 4: N-S RED (01H), E-W AMBER (02H)
    MVI A, 01H \ OUT 00H
    MVI A, 02H \ OUT 01H
    CALL SHORT_DELAY
    JMP TRAFFIC_LOOP

LONG_DELAY:
    MVI B, 02H
L1: DCR B \ JNZ L1
    RET

SHORT_DELAY:
    MVI B, 01H
S1: DCR B \ JNZ S1
    RET
```

18. 8-Bit ADC & DAC Waveform Oscilloscope

Simulates an 8-bit DAC connected to a real-time digital storage oscilloscope, with a companion 8-bit ADC input path. DAC Port: 00H. ADC Port: 01H. Reference voltage (Vref): 5.00 V, giving an LSB voltage resolution of 5.00 V ÷ 255 ≈ 0.019607 V (19.6 mV per LSB).



18.1 Input Voltage Conversion Reference

Port Value	Decimal	Output Voltage	Oscilloscope Reaction
00H	0	0.00 V	Graph line rests at the bottom zero line.
40H	64	1.25 V	Graph line rises to 25% height.
80H	128	2.50 V	Graph line sits at the 50% centreline.
C0H	192	3.75 V	Graph line rises to 75% height.
FFH	255	5.00 V	Graph line reaches the top peak line.

18.2 Waveform Generation Assembly Scripts

```
; 8-BIT DAC — SAWTOOTH & TRIANGULAR WAVEFORM GENERATORS
; DAC Port: 00H

GEN_SAWTOOTH:
    MVI A, 00H
RAMP_SAW:
    OUT 00H       ; Output voltage to DAC
    INR A         ; Step voltage up by 1 LSB (+19.6 mV)
    JNZ RAMP_SAW
    JMP GEN_SAWTOOTH

GEN_TRIANGLE:
    MVI A, 00H
TRI_UP:
    OUT 00H \ INR A \ CPI FFH \ JNZ TRI_UP
TRI_DOWN:
    OUT 00H \ DCR A \ JNZ TRI_DOWN
    JMP GEN_TRIANGLE
```

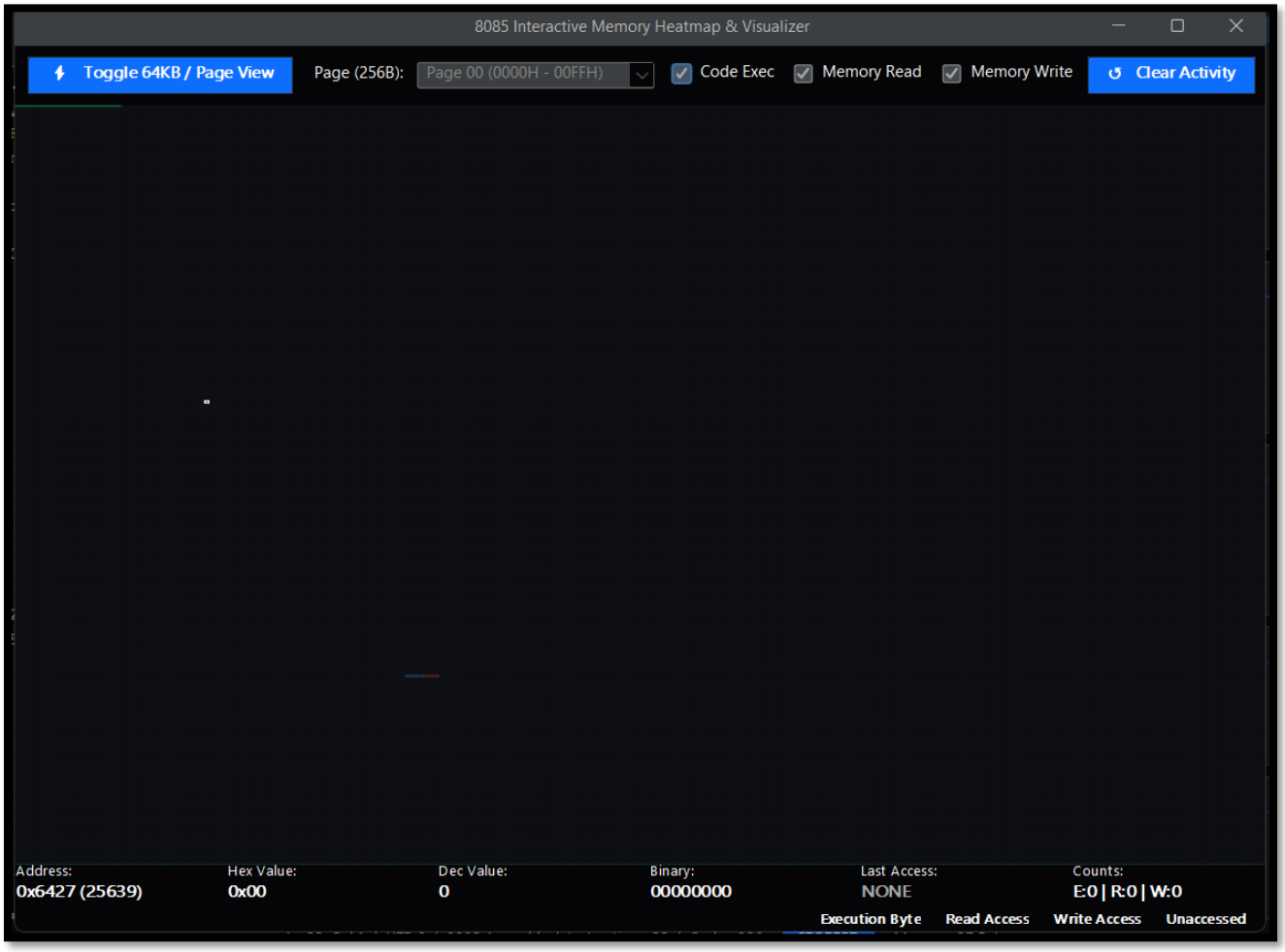
19. Interactive Memory Heatmap & Register Inspector

This chapter documents the colour-coding conventions and multi-format display used by the Memory Heatmap & Visualizer and the Multi-Format Register Inspector, introduced in Sections 7.5 and 4.2 respectively.

19.1 Memory Heatmap Colour Classification

The heatmap monitors the full 64 KB memory space (0000H–FFFFH) in real time, using the following colour classification for each cell:

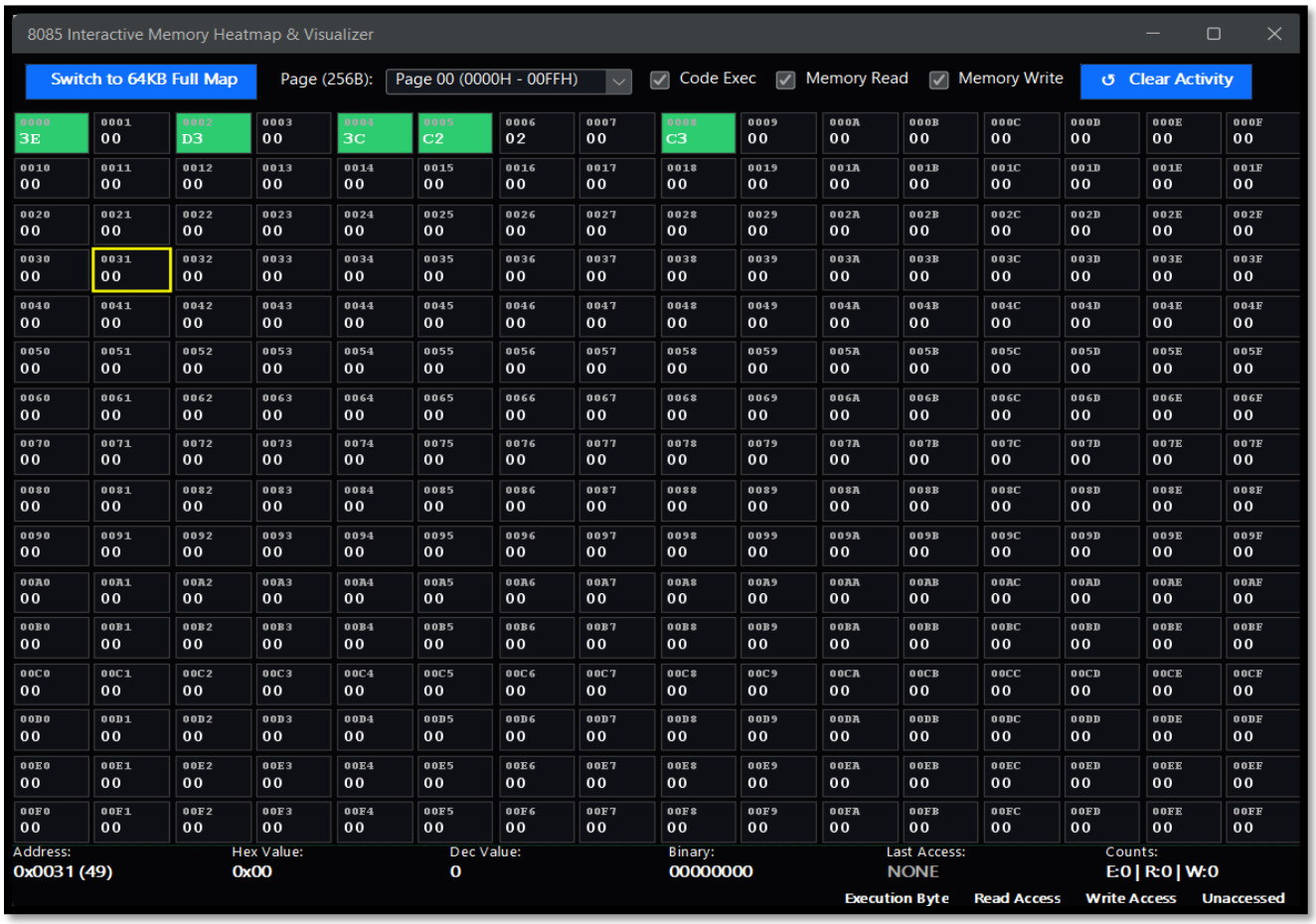
- Green cells — a memory read operation (e.g. MOV A,M or LDA 2000H).
- Red cells — a memory write operation (e.g. MOV M,A or STA 4000H).
- Blue cells — an instruction opcode fetch (PC execution).
- Gold cells — a high-density execution hotspot, where a loop has been repeatedly executed.



19.2 Multi-Format Register Inspector Matrix

Each register byte selected in the inspector is shown simultaneously in five formats:

Byte (Hex)	Binary	Unsigned Decimal	Signed Decimal	ASCII
41H	0100 0001	65	+65	'A'
00H	0000 0000	0	0	NUL
FFH	1111 1111	255	-1	ÿ
30H	0011 0000	48	+48	'0'



20. Delay Subroutine Generator & Timing Analysis

This chapter documents the timing mathematics underlying the Insert Delay Subroutine wizard described in Section 8.1. Generated loops calculate exact T-state counts for a target delay based on the configured CPU clock frequency.

20.1 Timing Formula

For a target clock frequency of 3.072 MHz, the resulting delay is: T-delay = N-T-States × (1 / f-clock). The wizard sums the T-state cost of every instruction in the generated loop (load, decrement, conditional jump, and return) across the requested number of iterations to compute the minimum and maximum achievable delay before code is generated.

20.2 Reference Single-Register Delay Loop

```
; 1-REGISTER DELAY LOOP
DELAY:
    MVI B, 32H    ; 7 T-States (load iteration count)
D_LOOP:
    DCR B         ; 4 T-States
    JNZ D_LOOP    ; 10 T-States (jump), 7 T-States (last exit)
    RET           ; 10 T-States
```

21. Disassembler & Code Wizard Libraries

Aura Studio's Disassembler decodes RAM contents byte-by-byte, converting raw machine code back into formatted 8085 assembly source (see Section 7.1 for its behaviour and limitations when opcode and data bytes are intermixed).

The Code Wizard (Section 10.2) additionally ships with a library of pre-built template subroutines covering common teaching exercises, including bubble sorting, array searching, string manipulation, and BCD arithmetic — the same categories of sample program listed in Section 3.8.

This manual is provided as the official reference documentation for Aura Studio and is intended to accompany the application for classroom, laboratory, and self-study use.