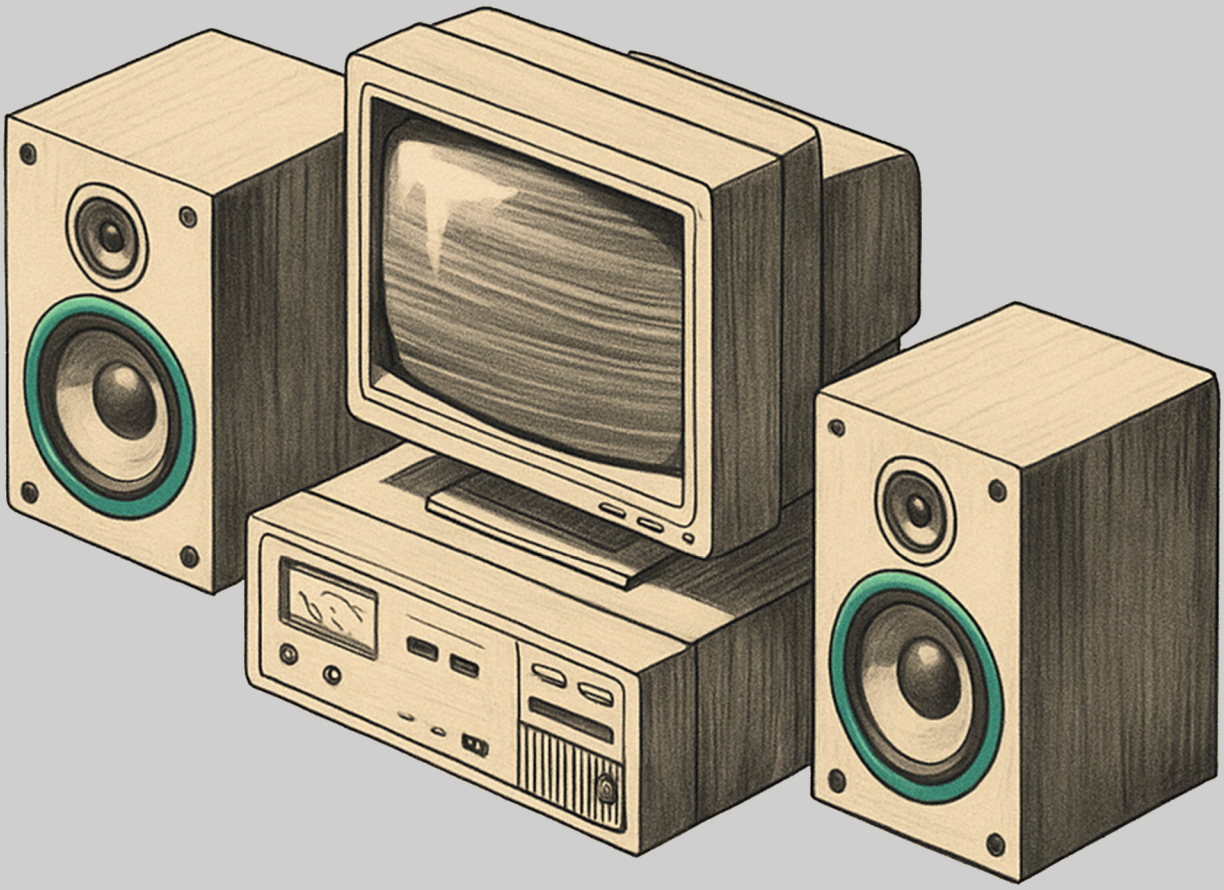
# GETTING STARTED WITH ASSEMBLY LANGUAGE PROGRAMMING

## FOR YOUR BRADSONIC 69000



Jim Lardmeadow

# Getting Started with Assembly Language Programming for your BRADSONIC 69000

## By Jim Lardmeadow

### Contents

# Getting Started with Assembly Language Programming for your BRADSONIC 69000

By Jim Lardmeadow

## 1. The Core Paradigm: Data Triage and Relocation

Assembly language programming on 8-bit architectures is fundamentally a task of **data triage**. It is the process of precisely taking a single 8-bit datum (or byte) and relocating it to a specific, critical memory address or hardware register.

Every functional command executed by the CPU involves a transaction with its primary resource: **The Accumulator**.

### The Accumulator (Register A): The CPU's Single Repository

The Accumulator is the CPU's sole **8-bit working register** for arithmetic and logical operations.

- It must be conceptualized as the CPU's only available **Hand**.
- All data must pass through this Accumulator, be it data destined for storage or data retrieved for inspection, as it holds the singular byte upon which all immediate operations are conducted

---

## 2. The Mandate of Data Transfer: LDA and STA

The programming of the BRADSONIC is dominated by two mnemonic instructions governing the ingress and egress of data relative to the Accumulator.

### A. LDA (Load Accumulator)

This instruction commands the CPU to retrieve a byte of data and deposit it into the Accumulator.

| Instruction | Meaning | Analogy |
|---|---|---|
| LDA #$01 | Load the **immediate** value 01 into the Accumulator. | This code tells the CPU to create the datum 01 and place it directly into its Hand. |
| LDA $C800 | Load the byte **stored at** the absolute memory address $C800 into the Accumulator. | This code tells the CPU to get or retrieve the current reading from the register designated at the memory address $C800 and place it into its Hand. |

### B. STA (STore Accumulator)

This instruction is the inverse of LDA. It commands the CPU to **deposit (**don't be fooled by the word **store**) the byte currently held within the Accumulator is sent to a designated memory location or hardware register.

| Instruction | Meaning | Analogy |
|---|---|---|
| STA $C400 | Store whatever we have, the byte currently held in the Accumulator, into the absolute memory address $C400. | The CPU writes the datum from its Hand to the control register designated $C400. |

# Getting Started with Assembly Language Programming for your BRADSONIC 69000

## By Jim Lardmeadow

---

## 3. The Crucial Distinction: Addressing Modes

The diligent programmer must observe the semantic difference between the two primary addressing modes, which dictate the source of the data for the LDA instruction.

| Type | Syntax | Name | Source of the Data |
|---|---|---|---|
| **Immediate** | #$XX | Literal Constant | The constant value is supplied directly within the instruction code itself. This is data created in the code. |
| **Absolute** | $XXXX | Memory Address | The data value is extracted from a specified 16-bit physical location in the system map. *Data that is already sitting there.* |

Any hex number with a # in front of it is a number created in the code, it hasn't come from anywhere else, the code the programmer writes has these values and will then store them into certain memory addresses within the system in order to do different things. One of the most obvious could be to change the colour of the screen. #00 is white on most machines and the video chip's colour ram could be set at $d400 so setting the screen to white could look like this:

```
CHANGE_SCREEN_COL

        LDA #00
        STA $D400
```

This sequence is the **fastest possible** way to perform this byte-level I/O.

Audio example where we would switch on the audio capabilities of a sound card:

| Example | Effect on the Accumulator... | Programming Use Case... |
|---|---|---|
| LDA #$01 | Loads the number 01. | Used exclusively to set an explicit configuration byte (e.g., enable power, set a volume level). |
| LDA $C800 | Loads the current reading of the $C800 register. | Used to read variable system data, such as the next audio packet or a status flag. |

---

## 4. Simple Program: Initialization of a Peripheral

Here, we present the fundamental two-step sequence for initial device activation, using the Radland Sound Card's Master Power Register ($C400) as the target.

| Step | Code | Procedural Description |
|---|---|---|
| **1.** | LDA #$01 | **Data Creation:** The activation byte (01) is loaded into the Accumulator. |
| **2.** | STA $C400 | **Data Delivery:** The activation byte is stored from the Accumulator into the Master Power Register. |

**Result:** The $C400 hardware register receives the required 01 signal, enabling the peripheral circuitry.

# Getting Started with Assembly Language Programming for your BRADSONIC 69000

By Jim Lardmeadow

---

## 5. Branching and Conditional Logic: BEQ and BNE

Up to this point, we have focused on moving data into and out of the Accumulator. However, a functional program must be able to react to data. This introduces the essential mechanism of comparison and conditional branching. These instructions form the backbone of all “if statements” in BRADSONIC 69000 Assembly Language.

A comparison always follows the same pattern:

1. Load a value into the Accumulator.
2. Compare that value with another value.
3. Branch depending on whether the result is equal or not equal.

CMP (CoMPare) updates the CPU's status flags. The most important for branching is the Zero Flag.

– If the values are equal, the Zero Flag is set.
– If they are not equal, it is cleared.

Two instructions read this flag:

BEQ — Branch if Equal
BNE — Branch if Not Equal

Below is an example from a program designed to observe a flashing screen colour and detect when it reaches 255:

```
COLOR_CHECK:
    LDA $F408
    CMP #$FF
    BNE COLOR_CHECK
    JMP OUTPUT
```

The CPU loops at COLOR_CHECK until the value becomes 255, at which point it proceeds to OUTPUT.

A simple BEQ example:

```
CHECK_FLAG:
    LDA $D010
    CMP #$01
    BEQ FLAG_FOUND
    JMP CHECK_FLAG
```

FLAG_FOUND is executed only when the compared value matches.

In summary, BEQ and BNE allow the program to respond to the state of memory, forming the foundation of all conditional behaviour in 69000 Assembly Language.

## 6. Elegance Through Simplicity

The decision to use a single-accumulator architecture for the BRADSONIC 69000 is a design choice that offers significant benefits, particularly for I/O-intensive tasks like peripheral control and real-time audio.

### Predictable Timing for Real-Time Audio

The most significant benefit of a single-accumulator design in this context is **predictable cycle counting**.

- **Fixed Execution Time:** Instructions involving the Accumulator (like LDA and STA) always take a known, minimal number of clock cycles to execute. There are no variable overheads from complex register-to-register transfers.
- **Audio Integrity:** For real-time audio processing (where samples must be delivered at a constant rate, e.g., 11kHz or 22kHz), this **fixed timing is crucial**. It ensures the driver's busy-wait loop executes with minimum *jitter*, preventing pops, clicks, or gaps in the stream caused by missed timing deadlines. This is especially vital for dealing with modem-based, incoming data which demands immediate service - like streaming radio services from the **The American Pacifica Isles, Government Approved** bulletin boards.

# Getting Started with Assembly Language Programming for your BRADSONIC 69000

## By Jim Lardmeadow

### Efficiency in Control Systems

The function of the Radland LAPC-1 is primarily a **control system**, setting an output (volume) or reading a status (Data Ready Flag).

- **Minimizing Overhead:** Complex math is not required. The driver's job is simply to move a single control byte (#$01 for ON, #$FF for max volume, or the audio sample byte) from point A to point B.
- **Optimal Cycle Count:** The **LDA → STA pattern** is the fastest possible way to perform this byte-level I/O. Adding more registers would introduce extra instructions just to move data between registers, slowing down the core I/O loop unnecessarily.

### Reduced Hardware Complexity

From an engineering perspective, fewer registers mean:

- **Fewer Transistors:** Leading to a smaller chip size and lower manufacturing cost.
- **Simpler Bus Structure:** Making it easier for system integrators (like the BRADSONIC team) to map the device onto the shared system bus.

---

## 6. In Summary

In summary, every required sequence, be it a test routine, an initialisation phase, or a continuous data stream, is constructed from this basic mechanical transaction. The core mechanism is **always LDA and STA**, load data into A, store data from A. The single-Accumulator architecture, while seemingly simplistic, is an elegant choice that provides **predictable, fast I/O timing** essential for the BRADSONIC 69000's real-time peripheral control.