

altairsim — MUSIC3: a four-voice music player on a Cromemco D+7A

2026-10-11 · 9862263

MUSIC3 plays "The Star Spangled Banner" in four voices. It sends the sound to the analog output of a **Cromemco D+7A**, which drives the speaker of joystick 1. This folder has the source, a CP/M machine that builds it, and a machine that plays it.

Machine file	What it is
<code>music3.toml</code>	A bare Z80 machine with a D+7A. It loads <code>MUSIC3.HEX</code> and plays it.
<code>music3-build.toml</code>	CP/M 2.2b with the Microsoft <code>M80</code> and <code>L80</code> . It builds <code>MUSIC3.HEX</code> from <code>MUSIC3.ASM</code> .

Play it

```
cd examples/d7a
altairsim music3.toml
```

The machine loads `MUSIC3.HEX` and runs it at `0100`. The song takes about 60 seconds of emulated time. After the song, the program stops at an `HLT`. Press `Ctrl-E` to get the monitor prompt.

You hear the sound only on a build of altairsim that has a sound device. On another build, the program runs the same and the speaker is silent. To see that the speaker is in use, type `SHOW d7a0` at the monitor prompt. The line `speaker 1` says `playing`.

The processor is a Z80 at 4 MHz. The program uses the undocumented `IYH` and `IYL` instructions, and its frequency table is tuned for 4 MHz. Do not change `clock_hz` in `music3.toml`. With `clock_hz = 0` the D+7A plays nothing.

Build it

The build runs in the guest. It needs no host tool.

1. Start the build machine from this folder:

```
cd examples/d7a
altairsim music3-build.toml
```

2. Copy the source and the build commands from the folder to the CP/M disk:

```
A>R MUSIC3.ASM
A>R MUSIC3.SUB
```

3. Run the build:

```
A>SUBMIT MUSIC3
```

`MUSIC3.SUB` runs two commands. `M80` assembles the source and writes `MUSIC3.REL` and the listing `MUSIC3.PRN`. `L80` links the program and writes `MUSIC3.HEX`. A correct build prints `No Fatal error(s)` and then the link map:

```
Data    0100    0610    < 1296>
```

4. Copy the results back to the folder. Use `T` for a text file:

```
A>W MUSIC3.HEX MUSIC3.HEX T
A>W MUSIC3.PRN MUSIC3.PRN T
```

5. Press `Ctrl-E` to get the monitor prompt, and leave `altairsim`. Play the new file with `altairsim music3.toml`.

The `R` and `W` programs use the folder that you started `altairsim` from. Without `T`, `W` writes up to 127 `Ctrl-Z` bytes (1A) after the last line of a text file. `LOAD` ignores them, but the file is not clean.

`MUSIC3.DSK` has about 24K free before a build. A build writes over the `MUSIC3.REL`, `MUSIC3.PRN` and `MUSIC3.HEX` from an earlier build. The listing alone is about 65K. The disk is mounted read/write, so copy the image first if you want the original.

The program

- Four voices play at the same time. Each voice has a 16-bit counter that steps through a wave table with a note frequency from a frequency table.
- The four wave table values add together. The program subtracts `80h` from the sum, and sends the result to the DAC with `OUT (19h), A`.
- The play loop takes 244 T-states, so the frequency table is right for a 4 MHz Z80 only.
- The score is the song table from *BYTE* magazine, volume 2, number 9 (September 1977), pages 77 and 78. A duration of 64 is a quarter note.
- The program puts its stack at the top of its 2K of working memory. It clears that memory before it plays.

The source

`MUSIC3.ASM` is `MUSIC3.z80` by Dave Roberts (daver2), version 0.30. He posted it as `MUSIC3 20240817.zip` on 17 August 2024, in post 356 of the VCFed thread [Cromemco JS-1 joystick replica](#). He wrote it for the ASM80 web assembler and emulator (asm80.com). The `.emu` file in his archive is the ASM80 emulator setup.

The source uses ASM80 syntax, which `M80` does not accept. `MUSIC3.ASM` has these changes and no others. The code, the tables and the comments are his.

In <code>MUSIC3.z80</code>	In <code>MUSIC3.ASM</code>
<code>.engine music</code>	removed
<code>.if</code> , <code>.else</code> , <code>.endif</code>	<code>IF</code> , <code>ELSE</code> , <code>ENDIF</code>
<code>.macro NAME</code> and <code>.endm</code>	<code>NAME MACRO</code> and <code>ENDM</code>
<code>.org</code>	<code>ORG</code>
<code>0x20</code> and <code>\$F0</code>	<code>20H</code> and <code>0F0H</code>
<code>msb(FREQTBL)</code>	<code>HIGH FREQTBL</code>
<code>.align 256</code>	<code>ORG (\$+255) AND 0FF00H</code>
<code>V1FC EQU RAM + 0</code> and the other <code>EQU</code> lines that use <code>RAM</code>	moved to the end of the file
(none)	<code>.Z80</code> and <code>ASEG</code> at the start
<code>DEBUG EQU 1</code>	<code>DEBUG EQU 0</code>

`M80` does not accept an `EQU` that uses a label defined later in the file. For this reason, the `EQU` lines that use `RAM` are at the end.

DEBUG selects the build. `DEBUG EQU 0` is the build in this folder. The program is at `0100`, the tempo is `99h`, and the sound goes to the DAC. `DEBUG EQU 1` is the original debug build. The program is at `0000`, the tempo is `06h`, and the program prints hex values to the console at port 2 and makes no sound. With `DEBUG EQU 1`, `M80` makes the same 1050 bytes as ASM80 makes from `MUSIC3.z80`. A `.COM` file needs `DEBUG EQU 0`.

Files

File	What it is
music3.toml	The machine file that plays MUSIC3.HEX .
music3-build.toml	The machine file that builds MUSIC3.HEX under CP/M.
MUSIC3.ASM	The source, for M80 , with the original comments.
MUSIC3.SUB	The two build commands: M80 and L80 .
MUSIC3.HEX	The program as Intel HEX, from 0100 to 060F .
MUSIC3.PRN	The M80 listing.
MUSIC3.DSK	The CP/M 2.2b disk image that has M80 , L80 , SUBMIT , R , W and HDIR .