Mzfdisas.

Jeroen F. J. Laros

September 6, 2003

Contents

1 Introduction. **3**

2 Why interactive? **3**

3 The console. **3**

4 Dump mode. **4**

5 Usage. **4**

6 Nice features. **4**

7 Release. **4**

8 License. **4**

9 References and credits. **4**

1 Introduction.

This utility has the sole purpose to retrieve information from an MZF file. It is just a viewer. No editing or assembling can be done with this program because a simmilar application already exists, it is named AS2MZF and can be downloaded at: http://www.sharpmz.org/mz-700/as2mzf.htm
I wrote Mzfdisas beacause I was searching for loaders in MZF files and I found running a disassembler on an emulator too much work, especially when looking at many files.

2 Why interactive?

The first version I made simply disassembled a raw file, which was what I wanted, however, there was one problem: the allignment. Let's look at an example:

```
0000: c3 04 00                jp 0004h    ; jump to address 0004
0003: 06                                  ; just a byte with value 06
0004: 00                      nop         ; no operation
```

When you disassemble this code you'll end up with this:

```
0000: c3 04 00                jp 0004h    ; jump to address 0004
0003: 06 00                   ld b,00     ; put 00 in register b
```

That's the problem. We don't know what is data and what is code. I've taken a look at how DOS debug handles this and the answer is: it doesn't. Moreover; it can be proven that disassembling an arbitrary file correctly is imposible, although the proof is beyond the scope of this document. What you can do is disassemble about 99% of the program, for the remaining 1% some inteligence of the user is required. To disassemble the example correctly the user should disassemble from address `0004` in order to retrieve the `nop` instruction.

3 The console.

In order to give the user the chance to alter the offset from which will be disassembled I implemented a console. It recognises the following commands:
jxxxx : Set the address pointer to xxxx.
e : Go to the execution address found in the header.
l : Go to the load address found in the header.
d : Toggle disassemble / dump mode.
u : Disassemble or dump one page.
h : Print a help message.
q : Quit.

4 Dump mode.

I found that just disassembling wasn't enough to retrieve all needed information easilly. The header for example has relevant infomation we might want to see and data chunks don't look very meaningful when disassembled. To see this data in a meaningful way we need a hexadecimal dump, the problem with this is that the MZ uses an ASCII table that differs from the one the PC uses. That's why the dump mode is implemented.

5 Usage.

When you start the disassembler you will see a dump of the header and a prompt (`>`). The disassembler is now in disassemble mode and you can see the disassembled code by typing `u`. If you want to see data you type `d` to go to dump mode and `u` do dump a page. All subsequent `u`'s will dump another page.

6 Nice features.

All undocumented opcodes are implemented, as well as all duplicates. The inactive opcodes will decode as a duplicate undocumented nop instruction.

7 Release.

This release will include the source, a linux executable and a w32 exacutable.

8 License.

9 References and credits.

Most of the undocumented opcodes I found at:
http://www.geocities.com/SiliconValley/Peaks/3938/z80undoc.htm About eleven were missing, I found those at:
http://www.mdfsnet.f9.co.uk/Docs/Comp/Z80/UnDocOps See:
http://www.sharpmz.org/index.html for almost everything about the Sharp series.