MixEmul Documentation

Version 0.3.4

Table of Contents

1 What, why and how

1.1 What is MixEmul?

MixEmul is an emulator for the MIX computer that is described in The Art of Computer Programming (TAOCP) series of books from D.E. Knuth. MIX is a mythical, non-existent computer with features similar to those of real computers of the 1960s. It – or more accurately, its assembly language MIXAL – is used as the foundation for the text of aforementioned books.

MixEmul completely emulates the entire MIX instruction set. With that I mean that MixEmul not only performs the actions that an instruction should – like multiplying when it encounters a MUL instruction –, but it does it in the way MIX would. For example, the time unit ("tick") counts that are included in TAOCP Volume 1 are implemented as well. Also, as TAOCP specifies, I/O operations are performed in the background.

Speaking of I/O, all 21 MIX I/O devices (tapes, disks, card reader, card punch, printer, teletype and papertape) are implemented in MixEmul.

MixEmul lets you edit the contents of MIX's registers and memory before, during and after program execution using a number of editor types and review device status in a single glance. It includes a breakpoint feature and allows programs to be run in the background. It includes the GO button functionality, supports interrupts, comes with the floating point module, performs execution profiling and it's even mildly configurable.

Oh, I'd almost forget, it incorporates a MIXAL assembler too. This means that you can write MIXAL programs using any text editor you like and then load them into MixEmul to debug and run them.

In a few words: MixEmul contains all features that I think I need to be able to better absorb the contents of the TAOCP books.

1.2 But why write a MIX emulator (and in .NET even)?

Well, I actually wrote MixEmul for a number of reasons, being:

1. I believed a good MIX emulator would allow me to get more joy and knowledge from reading TAOCP;
2. I didn't think there was a sufficiently usable MIX emulator available for the Windows platform until I completed MixEmul;
3. I had a long vacation and not much to do;
4. It was a lot of fun to write.

So, the fact you might be able to use it wasn't actually one of the reasons. However, if you do I won't hold it against you either.

I wrote it in .NET because I like the characteristics of the .NET platform and the C# language. I used to program in Java which is another great platform and language. However, I think the creators of C# managed to take some of the quirks out of Java and include new bits and pieces that make using it just that much more pleasant.

1.3 Why not write MMixEmul?

There is a number of reasons for that as well:

1. The prints of the books I recently purchased are still based on MIX, and only mention MMIX (a more modern replacement for MIX) as a future development;
2. MMIX is a lot more powerful and complex than MIX and therefore its emulator would need to be also. Writing MMixEmul would therefore be a major task that might start to feel like work. Not that I mind programming for work, but I already have a job;
3. My vacation wasn't that long.

Having said that, I did attempt to write the code for MixEmul in such a way that it might be used as the foundation for an MMIX emulator. Of course, I do realize that I have probably made all the wrong decisions while designing MixEmul's structure, which means that that will prove impossible if and when I actually try to do it. But hey, I've tried, haven't I?

1.4 What do I need to run MixEmul?

This:

1. A computer running Windows;
2. The Microsoft .NET Framework version 4.6.1 (or higher, I guess);
3. A directory to unpack the contents of the ZIP archive you found this document in;
4. Enough disk space for the device files of those devices you choose to use. The size of the device files for all devices except the disks depend on yourself and/or your programs. The device files for the disks are 2,457,600 bytes each.

Regarding requirements 1 and 2 I should add that it might be possible to run MixEmul on other platforms than Windows using Mono. However, I haven't tried that with the latest version and am not planning to either. If you do try it please let me know what your findings are.

1.5 Will it actually work?

To be honest, I don't know, because I haven't yet used all instructions in the MIX instruction set. Nevertheless I decided it was time to (have other people) start using MixEmul. Of course, I did do some tests and I have been able to run a number of MIXAL programs on it. Amongst those are the Primes (TAOCP section 1.3.2), Easter (TAOCP 1.3.2 exercise 14) and Permutations (TAOCP section 1.3.3) programs. These programs have been included in the ZIP file, by the way.
I've also successfully performed some I/O operations on all MIX devices.

In short, everything I've used myself so far seems to work.

If you do find a bug and would like me to fix it please let me know and I'll attempt to set it straight.

1.6 What about the legal mumbo-jumbo?

There isn't any.

MixEmul is completely free and you may use it in any way you like. If you manage to build a 6 billion dollar corporation on top of it without sending me as much as a thank-you e-mail then that's fine by me. (However, in that case I would be more than happy to add some improvements to the program for just 10% of your company's shares.)

For the price you pay for MixEmul you get an appropriate amount of warranty: none. In other words, MixEmul is provided as is and if you do choose to use it then you do so at your own risk. If it doesn't do anything useful, erases your hard disk or eats your dog you're on your own. However, in the first case you could drop me a note in which case I'll see what I can do.

1.7 Can I have the source code?

Yes. It's actually out there on the big bad Interweb, on github to be precise. To be even more specific, you can find it here: https://github.com/rbergen/MixEmul.

Now, before you get your hopes up and expect to see some software engineering brilliance, I have to warn you that the source code will show that MixEmul has been under development for 10 years by now, and incorporates its fair share of technical debt due to that fact. Furthermore, the source code is almost completely liberated of any documentation.
All this may very well be bad programming practice, but I did this largely for the fun of it.

1.8 Who are you anyway?

I'm Rutger van Bergen, a 38 year old computer science undergraduate born and living in The Netherlands.

I have an e-mail address that you can reach me on, it's rbergen@xs4all.nl. Because I also have a job and a life I can't guarantee any response times but I do guarantee that any requests to do your school assignments for you will be happily ignored. So don't even think about sending them. Don't do it. Really.

If you contact me to report a bug, please check that you are using the latest version that is available. The full version number is shown in the About window, which can be opened from the Help menu. The current version of MixEmul is available from the MixEmul web page, which is located at http://rbergen.home.xs4all.nl/mixemul.html.

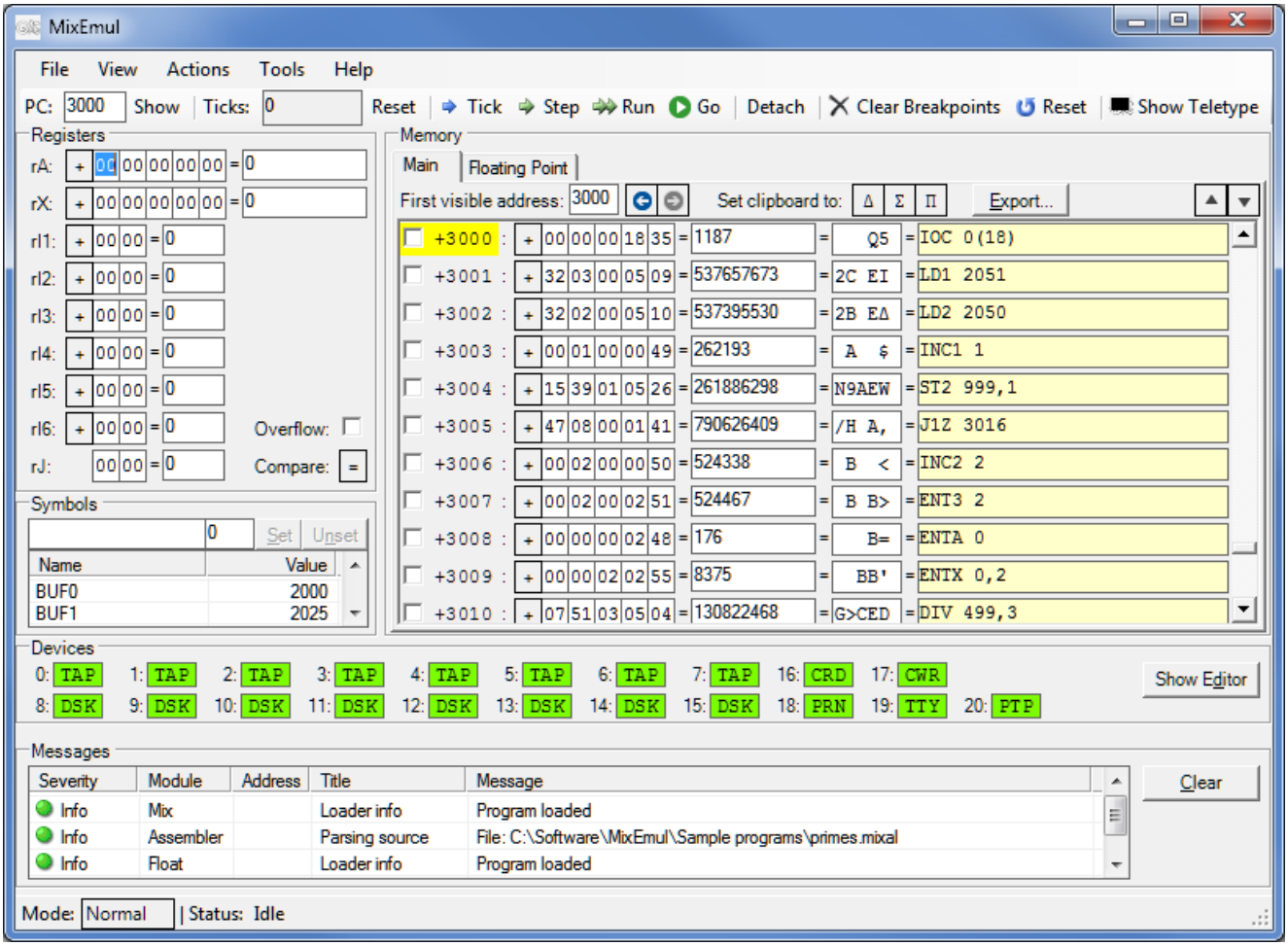
2 Running and using MixEmul

2.1 Starting MixEmul

MixEmul can be started by double-clicking on MixEmul.exe, after unpacking the ZIP file you got this document from.

2.2 Main window overview

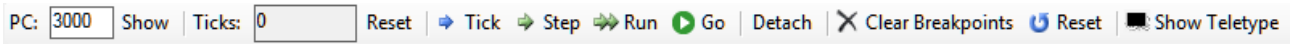
When MixEmul starts its main window is shown. It looks like this:



The main window consists of a number of regions which are discussed in the following text.

2.3 Control strip

The control strip is located at the top of the main window. It looks as follows:



The control strip contains the following items:

- An editable field (PC) that shows the current value of the program counter, i.e. the memory address from which the instruction to execute (the current instruction) will be loaded if the Tick, Step or Run buttons are pressed. You can change the program counter value by editing the field's contents and pressing Enter. The program counter memory address can be made visible in the memory region (see section 2.6) by clicking the Show button next to it.
  Please note that the PC field always shows the value of the *main* program counter, even if MixEmul is in Module mode and the Floating Point tab is selected in the memory region.

- A read-only field that shows the current value of the tick counter, i.e. the number of time units that have passed since program startup or the last system or tick counter reset. The tick counter can be reset to 0 by clicking the Reset button right next to it.

- A Tick button. When this is pressed in Normal or Control mode, MixEmul runs for precisely one time unit. Depending on the current instruction and the status of Mix this may have the same effect as clicking the Step button, or have no visible effect at all. If MixEmul is in Module mode, one module instruction is executed (irrespective of the actual tick count of the instruction in question) without the tick counter being increased. Please refer to chapter 5 for more information about Normal and Control modes, and chapter 6 for more information about Module mode.

- A Step button. Pressing this will run MixEmul until the current step is finished. A step is finished when the program counter changes from the value it has at the time of pressing Step to another value. Note that the current instruction might execute multiple times within a single step. An example of this is when the current instruction is JBUS *(18) and the printer is busy.

- A Run button. Pressing the Run button will run MixEmul until one of the following conditions occurs:
  - The HLT instruction is executed. In this case MixEmul, or rather the tick counter, continues running until all busy devices are ready.
  - Teletype input is required but not available (see the discussion on the teletype window in section 4.3).
  - A breakpoint is reached.
  - A runtime error occurs.
  - The Stop button is pressed.

  While MixEmul is running after clicking Step or Run, the Run button changes into a Stop button. Clicking this will stop execution after completing the then current tick.

- A Go button. This button will activate the MIX loader. Please refer to chapter 8 for more information about using the MIX loader and this button.

- A Detach/Attach button. This button can be used to change MixEmul's execution mode. MixEmul can run in either of the following modes:
  - In Attached mode the main window is regularly updated when MixEmul is running. The controls, registers, memory and devices regions are updated to show the current status of the respective parts of the emulator. Using this mode allows you to monitor the effects of your programs as they are running. However, because the updating of the main window takes time, programs run slower than in Detached mode.
  - In Detached mode MixEmul runs the programs it executes in the background. In this mode the main window is not updated until execution stops, but programs will run faster.

  The execution mode can be changed when MixEmul is running.

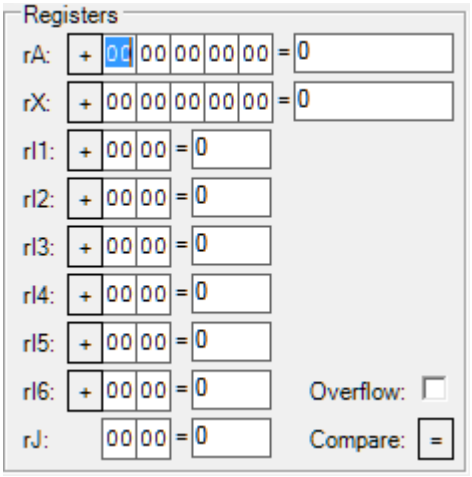
- A Clear Breakpoints button. Clicking this button will clear any breakpoints that are currently set.

- A Reset button. Clicking this will Reset MixEmul. All register and memory values are set to 0, as are the program and tick counters. All devices are reset as well.

- A Show/Hide Teletype button, which does exactly what its name implies. More information about the teletype can be found in section 4.3.

2.4 Registers region

The registers region looks like this:

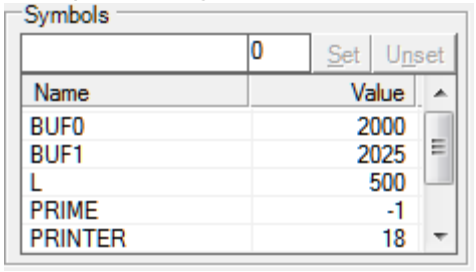


It contains the following items:

- Value editors for the registers. Values can be edited per byte (left of the equal sign) or as a composed value (right of the equal sign). The editors for rA, rX and the index registers support negative values. In the byte editor the sign can be changed by clicking it.
  A byte or composed value that has been edited but not yet applied is shown in blue.
  Changes are applied by pressing Enter or leaving the field in question and can be aborted by pressing Escape.
- An overflow indicator/editor. The box is checked when the overflow flag is set.
- A comparison indicator/editor. The current value of the comparison register is shown, it can be changed by clicking the indicator.

2.5 Symbols region

The symbols region looks like this:



Upon loading a MIXAL program into MixEmul, the symbols region shows the named symbols that were included in said program.
It is possible to perform the following actions using the symbols region:

- Adding a new symbol, by entering a unique, valid, symbol name and the value it should have, and pressing the Set button.
- Modifying an existing symbol, by entering the symbol's name or selecting it in the list, entering the symbol's new value, and pressing the Set button.
- Removing an existing symbol, by entering the symbol's name or selecting it in the list, and pressing the Unset button.

The symbols that are included in the symbols region, regardless if they are loaded from a program or entered manually, can be used in instructions that are entered into the instruction editors in the memory region (see the next section).

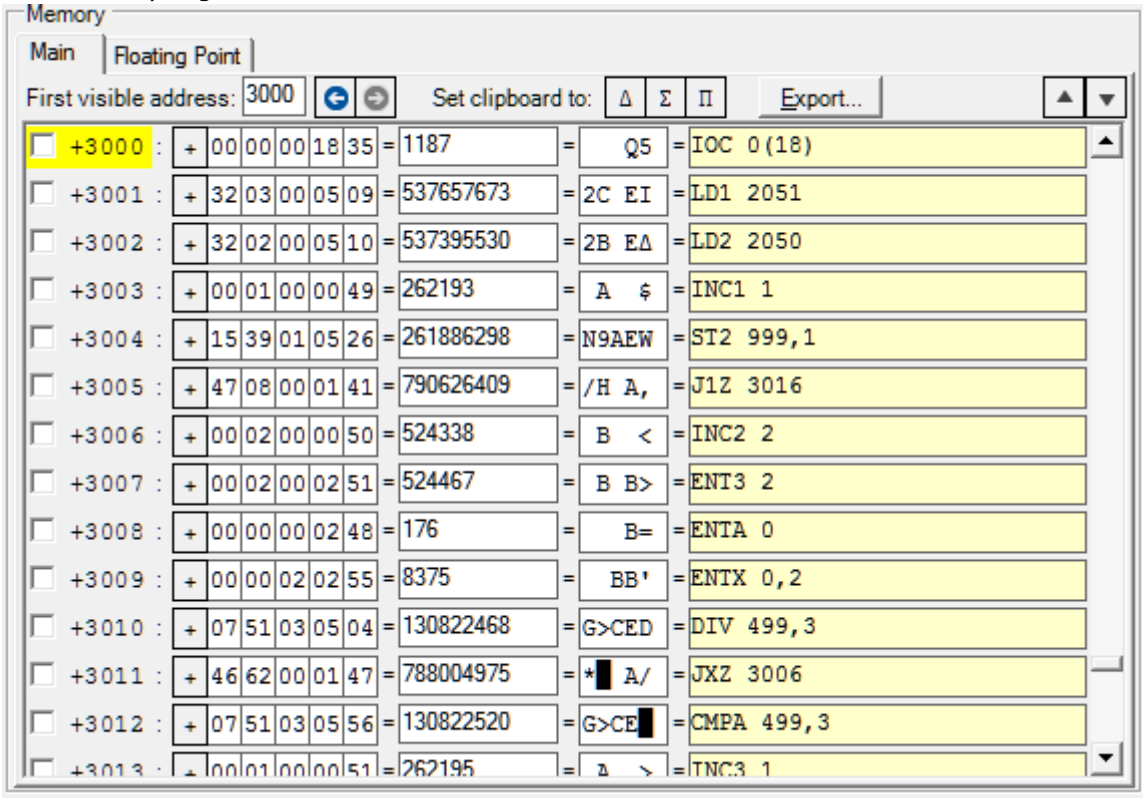
For those symbols of which the value is a valid memory address, the address in question can be made visible in the memory region either by double-clicking the symbol or by selecting it and pressing Enter.

Please note that the symbols region shows the symbols for the memory tab that is selected in the memory region. That means that if the Floating Point memory region tab is selected, the symbols region shows the symbols for the program that is loaded into the floating point module.
For more information about the floating point module, please refer to chapter 6.

2.6 Memory region

2.6.1 Description

The memory region looks like this:



It contains two tabs, one for MixEmul's main memory and one for the memory of the floating point module.

Each tab contains the following items:

- An editable field that contains the address of the first visible memory word.
- Two navigation buttons, with which it is possible to navigate backwards and forwards in the history of memory addresses that have been shown. The address history includes those addresses that have been selected by user actions like setting the first visible address, editing the program counter or double-clicking a symbol in the symbol region.
- Three buttons with the delta (Δ), sigma (Σ) and pi (Π) characters, which I could not find on my keyboard. Clicking any of these will put the character that is shown on the button on the clipboard.
- An Export button with which (part of) the memory's contents can be exported in a format that can be used in conjunction with the MIX loader. Please refer to chapter 8 for more information about using the MIX loader and this button.
- At the far right, just above the scroll bar, two buttons with which it is possible to browse through the areas of memory that are not empty. In this context, a memory word is "not empty" if it either contains a value different than +0, or has an assembled line of source code associated with it. If either the first or last memory words that aren't empty are currently being shown, the corresponding browse button will be disabled.

- A value editor for each of the memory words. Each of these consists of the following:
  - A checkbox that can be used to mark the address as a breakpoint;
  - The memory word's address. If the address is equal to the program counter, the address and checkbox are marked in yellow;
  - Byte and composed value editors that are identical to those for rA and rX (see the discussion of the Registers region in section 2.4);
  - A character editor. Each character corresponds to one byte of the memory word. Byte values for which no character is available are shown as a vertical block (▯);
  - An instruction editor.

2.6.2 The instruction editor

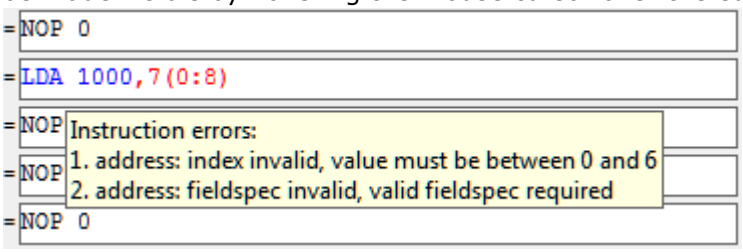
The instruction editor merits a slightly more detailed discussion. The instruction editor for a memory word shows, if possible, which instruction the word's value represents. Values that do not represent an instruction are indicated with the text *Not an instruction*.

Being an editor, it is also possible to enter the instruction you want the memory word to contain. Instructions entered in the editor must comply with MixEmul's MIXAL format (see the discussion of MixEmul's assembler in section 3.2), with the following restrictions:

- The location field is not supported and must not be specified;
- Only MIX instructions are supported. That means that the following instructions cannot be entered: EQU, ORIG, CON, ALF and END;
- In the address field, the only symbols that are supported are those included in the symbols region (see the previous section), and *.

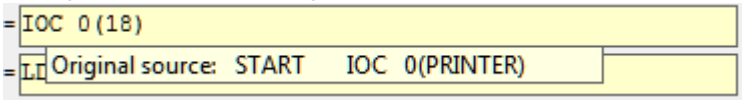
As with the other editors, any changes made can be applied by pressing Enter or leaving the editor.

Any invalid parts of the instruction (both in case of a word value being shown or an instruction being parsed) are marked in red. The actual errors are shown in the editor's tooltip, which can be made visible by hovering the mouse cursor over the editor:



Please note that an invalid instruction will not be applied. This means that if the editor is left while it contains an (edited) invalid instruction, the changes that were made are lost.

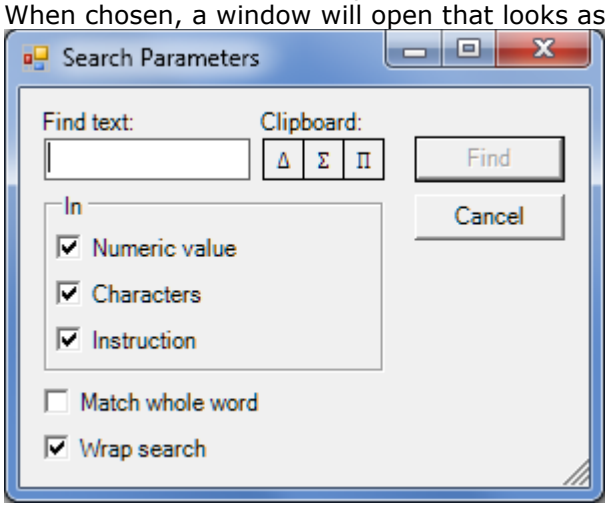
If a program has been loaded into memory, the memory addresses that contain instructions or values (CON and ALF) that were assembled and loaded from the source file are marked by a different background color. The actual instruction that was loaded is then shown in the editor's tooltip, in addition to any instruction errors that were found:



Right-clicking an instruction editor shows a menu through which the memory address that is referred to in the instruction's address field can be made visible in the memory region. Through another option in the same menu this can be done with indexing applied. Either option is only available if the address (and index, if applicable) is in fact valid.

2.6.3 Searching

The contents of the memory can be searched via the "Find in memory" option in the View menu. When chosen, a window will open that looks as follows:



The various controls have the following functions:

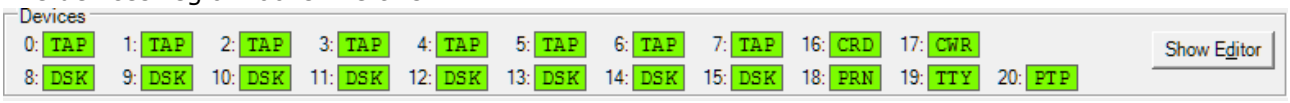
- In the "Find text" editor (you'll never guess this!) the text can be entered that needs to be searched for. Only characters that are part of the MIX character set can be used; other characters won't be found by definition.
- The buttons under "Clipboard" perform the same function as the ones that are part of the memory region itself (see above).
- Using the options in the "In" box, it is possible to indicate which editors need to be searched. The available options are the composed value (numerical) editor, character editor and/or instruction editor, each as discussed in earlier text in this section. At least one editor must be chosen.
- An option to indicate if the search should be performed using "whole word" matching. This means that the search text will only be considered to be found if it is not part of a longer "word" (to be precise, a bigger consecutive set of letters or numbers). For instance, if whole word matching is selected and a search is performed for "15" (excluding quotes), then "(15)" will be considered a match, but "150" or "A15B" will not.
- An option to indicate if the search should wrap around the end of the memory. That is to say, if the search should continue at the beginning of the memory if no match is found before the end of it.

After pressing Find, the search will be performed in the current (selected) tab of the memory region, starting at the current location of the cursor.

It is possible to perform the last search again from the now current location, using the "Find next" option in the View menu.

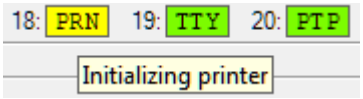
2.7 Devices region

The devices region looks like this:



It contains indicators for each of MixEmul's devices.
Each of these shows the following:

- The device number, as used in the F-field of the I/O instructions.
- An abbreviated name of the device. The following abbreviations are used:
  - TAP: Tape
  - DSK: Disk
  - CRD: Card reader
  - CWR: Card punch
  - PRN: Printer
  - TTY: Teletype
  - PTP: Paper tape
- The device's status. For idle devices the abbreviated name is shown on a green background, busy devices on a yellow one. More information on the device's status is shown in the indicator's tooltip, which is made visible by hovering the mouse cursor over the indicator:



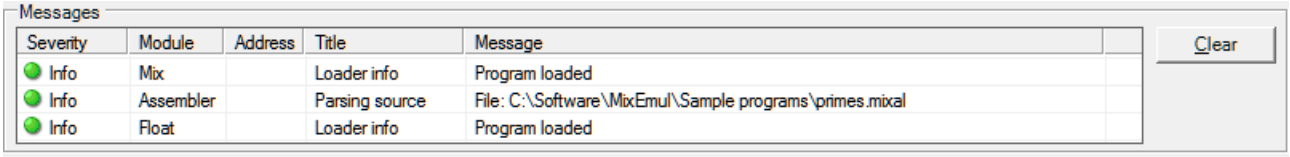
Right-clicking an indicator opens a menu that shows the following:

- If the device supports input, output or both;
- An option to reset the device to its initial state.

Right next to the indicators is a button with which the device editor window can be opened or closed. More information about the device editor can be found in section 4.2.

2.8 Messages region

The messages region looks like this:

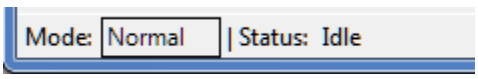


Messages that are generated by MixEmul or its devices are shown in this region, newest message first.
If the message includes an address then that address can be made visible in the memory region by double-clicking the message, or by selecting the message and pressing Enter.

The messages region can be emptied with the Clear button and resized with the splitter that is located directly above it.

2.9 Status bar

The status bar looks as follows:



The mode can either be Normal, Control or Module, and can be alternated between Normal and Control by clicking the mode box (button). If the button is pressed when the mode is Module, it will switch to Normal.
Please refer to chapter 5 for more information about Normal and Control modes, and chapter 6 for more information about Module mode.

3 Using the assembler

3.1 Overview

MixEmul's built-in assembler can be used to parse MIXAL programs written using an external editor and then load them into MixEmul.

At the moment MixEmul doesn't allow you to assemble your programs into binary executables. The reasons for this are that (a) I don't think that there would be much use for them besides loading them into MixEmul, and (b) the assembly of typical MIXAL programs doesn't take enough time to merit avoiding it. That said, it is possible to export assembled programs as card decks that can be used with the MIX loader. Please refer to chapter 8 for more information about using the MIX loader and creating and using exports of assembled programs.

The assembler is a two-pass assembler, which performs the syntactic checks during the first pass and the semantic checks during the second. If your program is correct you won't notice anything of this, but you will if it contains both syntactic and semantic errors. In that case only the syntactic errors are reported.

3.2 MIXAL input format

The MIXAL assembler accepts the terminal input format that is discussed in TAOCP section 1.3.2 with the following exceptions:

- Any characters that are included in an ALF address field but are not known to MIX are parsed as a byte with value 63.

- The address field of the ALF instruction starts with the first non-blank character after the blank(s) that follow the op field. To allow the ALF address field to start with a blank, double quotes (") can be used around the address field.

  The last double quote of the instruction line that is followed by a blank is considered to be the closing quote for the address field. This allows for double quotes to be included in the address field without the need for "escaping" them. (Not that there's much use in doing so, because the double quote isn't part of the MIX character set. But anyway…)

  Consider the following examples:
  - ALF FIRST and ALF "FIRST" both yield a word containing the characters FIRST (06|09|19|22|23).
  - ALF FIVE and ALF " FIVE" yield different words. The first word contains FIVE followed by a space (06|09|25|05|00), the second a space followed by FIVE (00|06|09|25|05).
  - ALF "QUO"TE yields a word containing 18|24|16|63|23
  - ALF "QUO" TE yields a word containing 18|24|16|00|00, and a comment TE.
  - ALF "QUO" TE" yields a word containing 18|24|16|63|00

- The rule that the END instruction is the last instruction of a program is enforced. Any instructions following an END instruction yield a warning and are treated as comment lines.

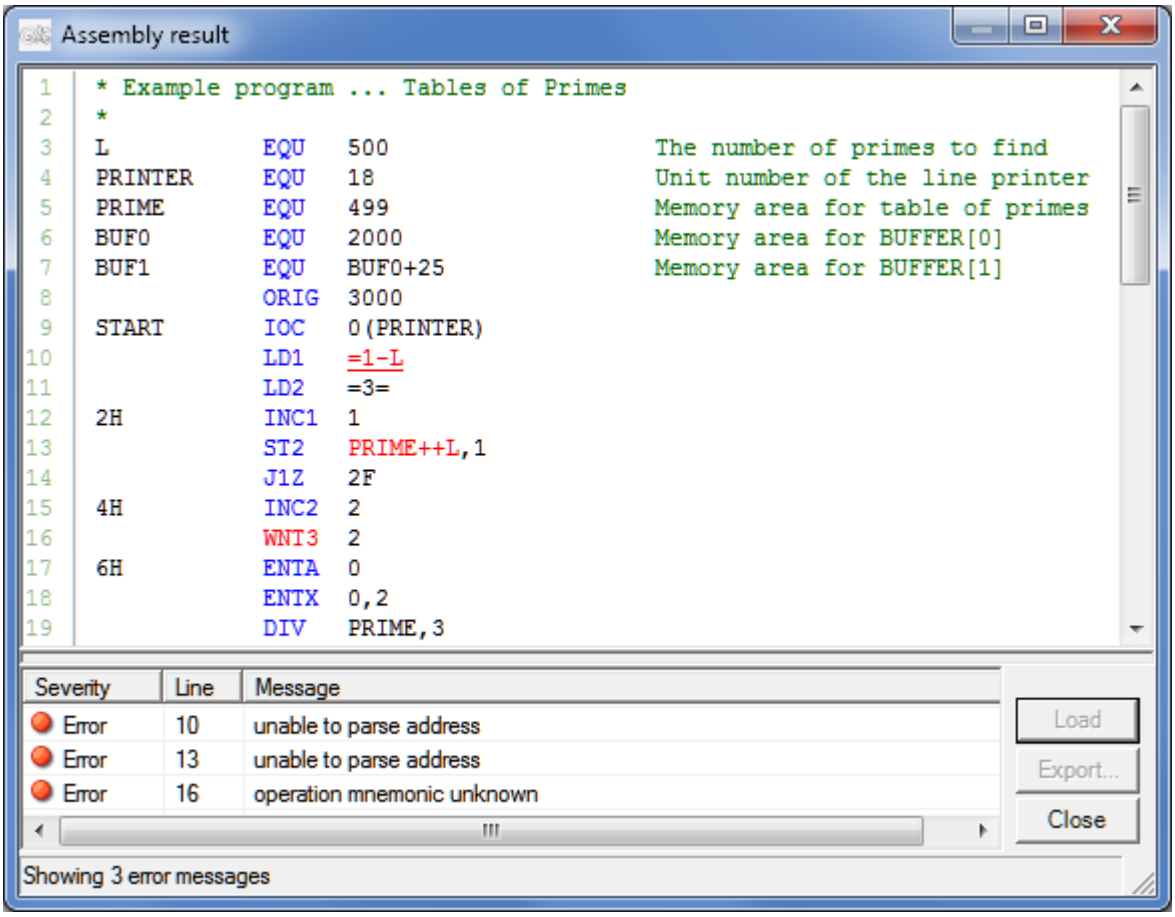
3.3 Invoking the assembler

MixEmul's assembler is activated by opening a MIXAL program. This can be done using the "Open program" option in the File menu.

The assembler is invoked automatically when the MIXAL program has been read from disk. After assembly completes the assembly result window is shown.

3.4 Assembly result window

This window shows the result of the assembly of an opened program. It looks as follows:



The lower part of the screen shows the messages that were generated during assembly. The upper part of the screen shows an evenly indented version of the MIXAL program that was assembled, with syntax coloring applied. Warnings and errors are also color-marked in the program source code.

In this case three syntactical errors were found. If a message is selected in the message list, the appropriate part of the source code is made visible and underlined.

If no errors were encountered during the assembly then the assembled program can be loaded into MixEmul's memory by pressing the Load button.
Furthermore, upon successful assembly, the Export button can be used to export the assembled program in a format that can be used in conjunction with the MIX loader. Please refer to chapter 8 for more information about using the MIX loader and this button.

4 Using devices

4.1 Device implementation

A few words about the way the devices are implemented are in order. The I/O operations (IN, OUT and IOC) are implemented as a series of steps. The general structure of these is as follows:

- IN: Initialization, Open file (if applicable), Read from file, Close file (if applicable), Copy read words to memory
- OUT: Initialization, Copy words to write from memory, Open file (if applicable), Write to file, Close file (if applicable)
- IOC: Initialization, Seek.

The number of ticks spent on the Initialization step is configurable per device (see the discussion of the preferences window in chapter 9), as is the seek time per record/sector for those devices that support seeking. Reading from and writing to memory is done one word per tick.

On binary devices (tapes and disks) words are stored as six bytes. The first byte contains the word's numeric sign (+ or -) and is followed by the five bytes of the word.

Text-based devices (card reader, card punch, printer, teletype and paper tape) are read from and/or written to one line at a time. For example, if the instruction IN 1000(16) is executed then one line is read from the card reader device file (default: crdin16.mixdev). If a line from an input file is longer than the device's record byte size (80 in case of the card reader) then the rest of the line is discarded. If a line is shorter, then the remaining bytes in the record are set to 0.
On input any characters that are not known to MIX are parsed as a byte with value 63. On output trailing spaces are dropped before a line is written.

Text devices are read from, and written to, using the ASCII character set. This means that of the MIX character set, the delta, sigma and pi characters cannot be read or written.

On startup or after a (device) reset the location pointer is at the following position:

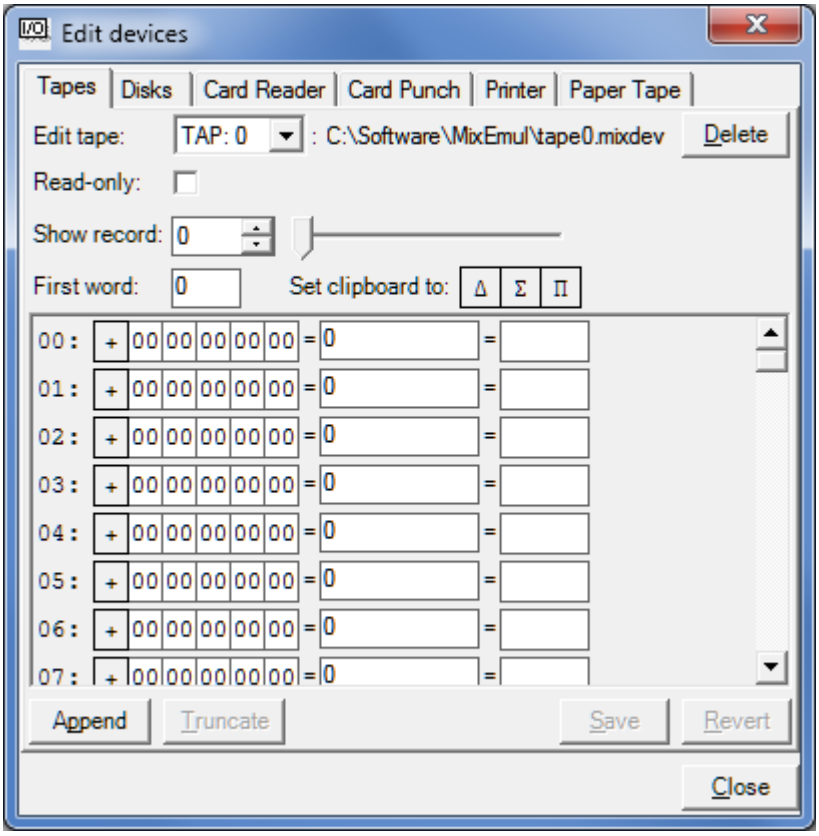
- For tapes, at the end of the device file;
- For disks, at the beginning of the device file (sector 0);
- For the card reader, at the beginning of the device file;
- For the card punch, at the end of the device file;
- For the printer, at the end of the device file;
- For the paper tape, at the beginning of the device file.

4.2 The device editor

The device editor can be opened and closed by clicking the Show/Hide Editor button in the Devices region of the MixEmul main window, or the Show/Hide Device Editor option in the View menu.

Through the device editor, the contents of all of MixEmul's devices except the teletype can be viewed and, if applicable, edited. The teletype device has its own window, which is discussed in the next section.

The device editor looks like this when opened:



The device editor contains 6 tabs, one for each type of device.

4.2.1 Binary devices

For the tapes and disks, a binary device editor is used. Through this editor, one record/sector can be edited at a time.
It is made up of the following parts:

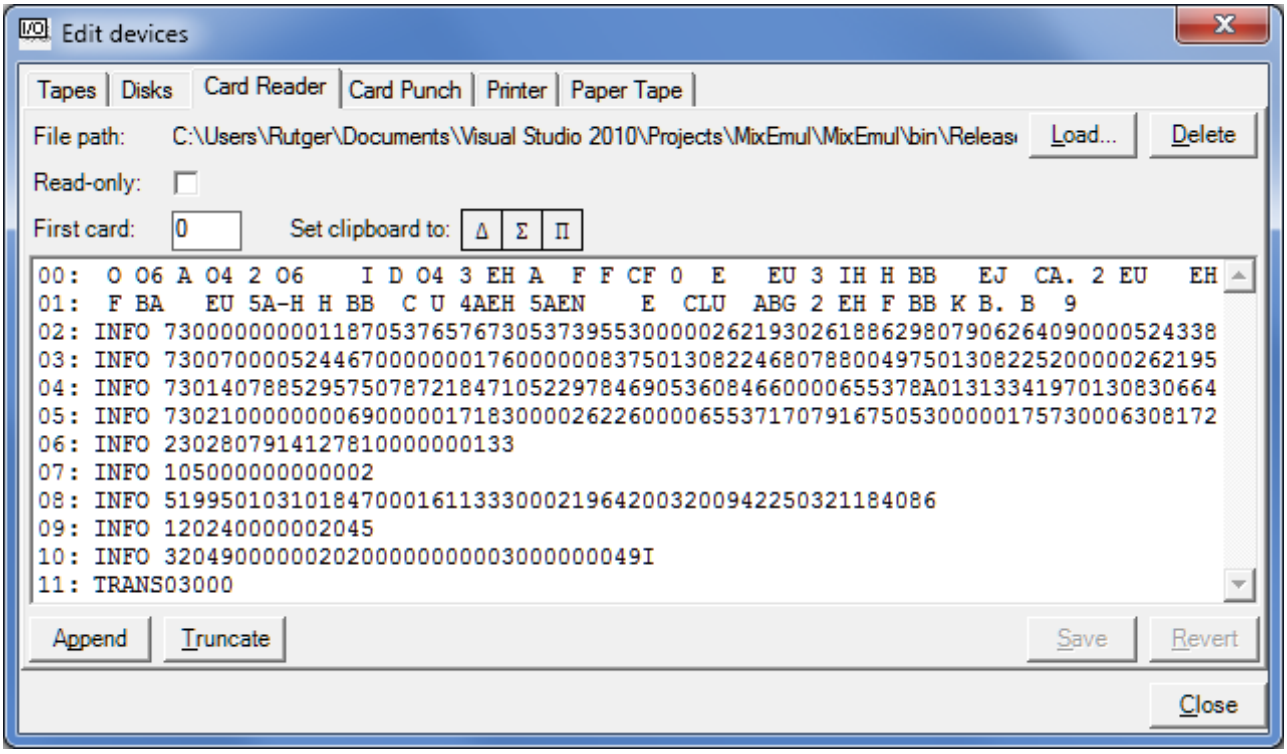
- A selection control to choose which tape/disk to edit. Next to it, the selected device's file path is shown.
- A button to delete the device file of the selected tape/disk, effectively resetting the device in question.
- A checkbox to switch the device editor to read-only (to prevent unintended modifications of the device's contents).
- A text field and slider to select the record/sector to edit.
- A text field that shows/sets which word is shown at the top of the record/sector editor.
- Buttons to load Mix-specific characters onto the clipboard. These work the same way the buttons in MixEmul's main window's memory region do.
- A list of editors for the words in the record/sector. These are identical to the editors in MixEmul's main window's memory region, except that there is no instruction text field in the device editor.
- For tape devices, buttons to append or truncate records to/from the end of the tape.
- Save and revert buttons to save or undo the changes to the current record.

Note that upon changing to another record/sector or selecting another device (through the aforementioned selection control) or device type (by clicking the appropriate tab), any changes to the current record/sector will be automatically saved.

4.2.2 Text devices

For the card reader and paper tape, a text device editor is used.
Each of these editors allows all of the device's contents to be edited at once.

It looks like this:



The editor is made up of the following parts:

- An indication of the selected device's file path.
- A button to delete the device's file, effectively resetting the device in question.
- A checkbox to switch the device editor to read-only (to prevent unintended modifications of the device's contents).
- A text field that shows/sets which card or record is shown at the top of the device content editor.
- Buttons to load Mix-specific characters onto the clipboard. These work the same way the buttons in MixEmul's main window's memory region do.
- A list of editors for the device's contents. These are identical to the character editors in MixEmul's main window's memory region, except that the maximum number of characters per editor is equal to the respective device's record length.
- Buttons to append or truncate cards/records to/from the end of the device.
- Save and revert buttons to save or undo the changes to the current device.

In addition to this, the card reader (which is shown above) includes a Load button, with which the current contents of the card reader device can be replaced with that of a file. After the file is loaded, the device pointer is reset to the beginning of the device file.
Although the button can be used to load any (text) file into the card reader, it was primarily added to load exports from memory (see section 2.6) or the assembler (see section 3.4), and thus be used with the MIX loader. Please refer to chapter 8 for more information about using the MIX loader and this button.

Note that upon selecting another device type (by clicking the appropriate tab), any changes to the current device will be automatically saved.

For the printer and card punch, a read-only viewer is used. These are identical to the text editors for the card reader and paper tape, except the following controls are not shown:

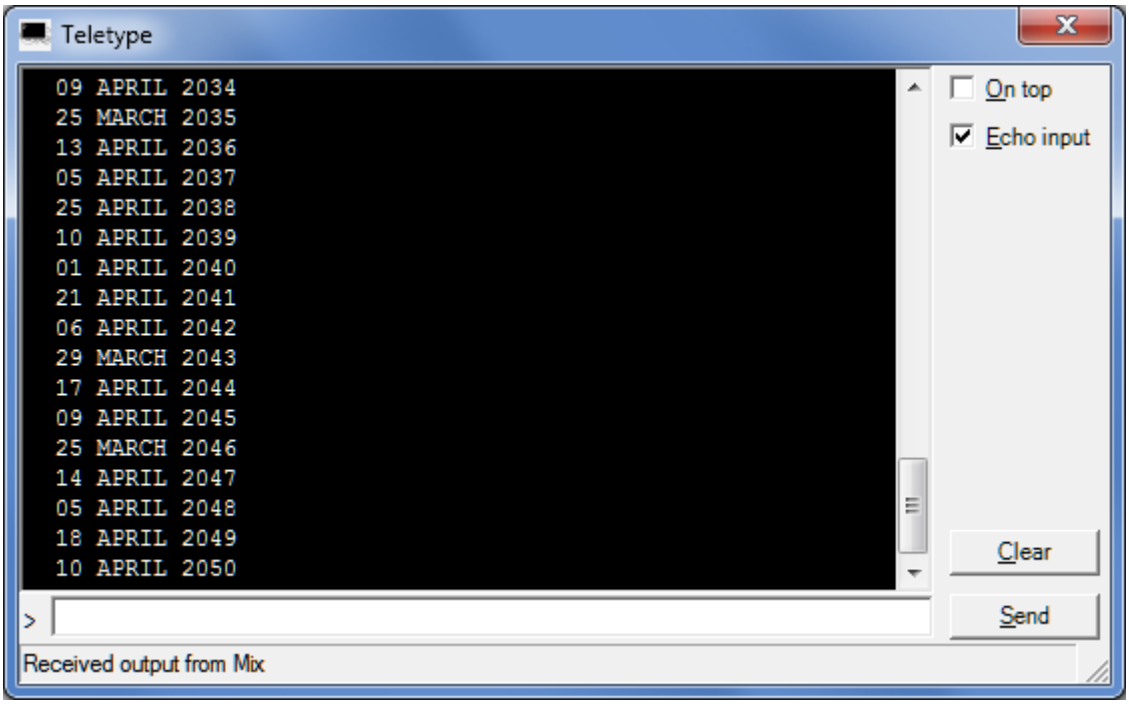
- The read-only check box
- The Mix character clipboard buttons
- The Append, Truncate, Save and Revert buttons

Besides the aforementioned Hide Editor button and menu option, the editor can also be hidden using the Close button at the bottom right of the editor window.

Whenever a device editor is visible – the device editor window is visible and the device editor tab in question is selected – the respective device's file is monitored for changes. When a change is detected, because the device was written to by MixEmul or manual modifications were made, the device editor's contents will be automatically reloaded. The minimum interval between these reloads is configurable using MixEmul's preferences window (see chapter 9).

4.3 Teletype window

The teletype window can be shown or hidden using the Show/Hide Teletype button on MixEmul's main window or the Show/Hide Teletype option in the View menu. The teletype window looks like this:



The upper part of the screen shows data that the teletype has received from MIX (i.e. the data that has been transferred using OUT …(19) instructions).
The output area can be cleared with the Clear button.

Input can be entered in the field at the bottom of the screen. If the "Echo input" option is checked then input that has been sent (by pressing Enter or clicking the Send button) is echoed in the output section, preceded by the prompt character (>).
Input is stored in an input buffer, where it is kept until it is retrieved using the IN …(19) instruction.
If an IN instruction is executed while the input buffer is empty then program execution is halted so that teletype input can be sent.

If the "On top" option is checked then the teletype window is kept in front of all other windows.

5 Interrupts

5.1 Introduction

Exercise 18 of section 1.4.4 of TAOCP describes the proposed extension of MIX with an interrupt feature. This extension consists of the following:

- An additional 3999 words of memory, which are in *negative address space*. This means that the total address space of MIX spans from -3999 to +3999.
- Introduction of a *Control mode*, next to the Normal mode that MIX usually runs in. The additional memory with negative addresses just mentioned is only accessible in control mode. That is to say, in Normal mode the accessible address space (still) spans from 0 to 3999.
- Introduction of a number of interrupt types that switch MIX into Control mode, and transfer execution to an interrupt-dependent address.

MixEmul includes support for this interrupt extension, conforming to the specifications in said TAOCP section exercise.

5.2 Interrupt types

5.2.1 Forced interrupts

A forced interrupt can be triggered by using the INT operation in Normal mode. After the contents of the registers have been stored in addresses -9 to -1 and the switch to Control mode has been made, execution transfers to address -12.
When the execution mode is switched back to Normal (by executing the INT operation in Control mode), execution is resumed at the address that follows the one the original INT instruction was loaded from.

5.2.2 Timer interrupts

Every 1000 ticks, the value at address -10 is decreased by 1. A timer interrupt occurs when this value reaches 0. After storing registers and switching to Control mode, execution transfers to address -11.

5.2.3 Device interrupts

A device interrupt occurs when a device finishes an I/O operation (IOC, IN or OUT) *that was started in Control mode.* A device interrupt transfers execution to address -(20 + device ID) – being -20 for the first tape unit, -38 for the printer and -40 for the paper tape, to name some examples – after storing registers and switching to Control mode, as usual.

5.3 MixEmul implementation specifics

The following remarks apply to the interrupt implementation of MixEmul:

- Switching to Control mode does not change the direction of the program counter. That is to say, the program counter *increases* on execution of instructions in Control mode, just as it does in Normal mode. For instance, after executing ENTA 0 on address -80 the program counter will change to -79, not -81;
- As specified in TAOCP, interrupts never actually occur in Control mode; they are queued instead. The first queued interrupt, if any, occurs after execution mode has returned to Normal *and one "Normal mode instruction" has been executed*. In the case of MixEmul, any unforced (thus timer or device) interrupt occurs when the then current Normal mode instruction has completed, and the program counter has been changed accordingly.
- The last remark in the previous point means that if, for instance, the instruction executed just before an unforced interrupt is a successful jump instruction, the program counter will have been set to the jump instruction's target address when the interrupt occurs. Because of this, when execution mode is switched back to Normal after an unforced interrupt has been handled, the program counter is restored to the same value it had when the interrupt occurred.

- The value at address -10 is only decreased if it is greater than 0. That means that the timer interrupt will not occur if said address contains a value equal to or less than 0.
- The INT instruction is only supported in MIX itself, which is to say it is not supported in the floating point module, more information of which can be found in chapter 6.
- It is possible to *load* MIXAL programs into negative memory space when the execution mode is Normal, as it is possible to edit memory contents using the memory region of MixEmul's main window (see section 2.6). As mentioned, *executing* such programs is only possible in Control mode.

5.4 Manual mode switching

MixEmul allows the manual switching between Normal and Control modes by use of the mode button in the main window status bar (see section 2.9). When a mode switch has been made, MixEmul's behavior is exactly as it would have been if the mode switch had occurred because of interrupt handling. Amongst others, this means that:

- Addresses -3999 to -1 are only accessible to MIX instructions in Control mode. This also means that if the program counter is a negative value on a switch to Normal mode, it will be set to 0.
- After a manual switch to Normal mode, the first of any queued interrupts – being any unforced interrupts that were triggered while in Control mode – will occur after execution of one instruction has completed.

5.5 Control program

MixEmul includes a feature to automatically load a control program upon startup and after a system reset. If a file "control.mixal" exists in the directory MixEmul is started from, it will be loaded into memory. If the file does not exists, loading it is silently skipped.
Although this feature is intended to provide implementations for the various interrupt handlers in negative address space, it can include any instructions in any part of the address space.

The default control program supplied with MixEmul immediately transfers control back to Normal mode on every interrupt, and ensures the timer interrupt is disabled.

6 The floating point module

6.1 Introduction

Section 4.2.1 of TAOCP describes an optional, single-precision floating point hardware module that can be added to MIX "at extra cost". The good news for today is that MixEmul includes this floating point module for free.

The floating point module provides the FADD, FSUB, FMUL, FDIV, FLOT, FIX and FCMP operations that are discussed under "Floating point hardware" in said section of TAOCP.

6.2 Module description

The MixEmul floating point module consists of a separate range of memory addresses (200 by default), in which a floating point MIXAL program is loaded when MixEmul starts. The program in question is "floatingpoint.mixal", which must be available in the directory from which MixEmul is started.

When a floating point instruction is encountered by MixEmul, control is transferred to the floating point module, which executes a specific part of the floating point program (see the "meaningful symbols" section, below). After that part of the program has finished executing, either successfully or with error, control is transferred back to the main MixEmul program.
Although the collection of MIX instructions executed for a floating point operation would typically far exceed the tick counts mentioned in section 4.2.1 of TAOCP, the tick counter will increase in accordance with said TAOCP section. In fact, the floating point module will execute all the instructions for a particular floating point operation during the last tick of that operation.

As mentioned, the floating point module comes with its own set of memory addresses, which is accessible via the Floating Point tab of the main window's memory region (see section 2.6). It does not include a separate set of registers. Instead, while the floating point module operates, the MIX registers are stored in and subsequently reloaded from 9 memory words that are automatically added to the end of the floating point module memory range. Although these memory words are visible on the memory region's Floating Point tab, they cannot be reached by the floating point program itself.
In case of the FADD, FSUB, FMUL, FDIV, FLOT and FIX operations, the contents of the rA register at the end of the floating point operation are kept and loaded into rA after the registers have been restored. In case of the FCMP operation all registers are restored to the values they carried before execution of the floating point operation started; in this case the comparison indicator is carried over from the floating point module to regular MIX operation.
In all cases, any over- or underflow, including division by zero, is signaled by setting the overflow indicator after reloading the registers.

The default implementation of the floating point program is based on the code for the floating point routines as it is included in section 4.2.1 of TAOCP, and for the FIX and FCMP operations, in exercises in later sections. The code has been modified slightly, mainly to work with the way that the MixEmul floating point module's interaction with the program has been implemented. This is described in more detail in the following section.
The default floating point program sets EPSILON (ϵ) to be $\frac{1}{2}b^{1-p}$.

Floating point instructions can only be issued from the "main" MIX module; it is not possible to include a floating point instruction in the floating point program itself.

6.3 Meaningful symbols

The link between the MixEmul floating point module and the MIXAL program it executes is based on the presence of a number of symbols, representing addresses, in the latter. They are discussed here, with the note that they are only really relevant to those who want to replace the default

floating point program with their own. Symbols that are marked as mandatory must be present in any program loaded into the floating point module; it will not work without them.

- **PRENORM**: Transfer is controlled to this address to "prenormalize" the value of rA and the value of the parameter for the FADD, FSUB, FMUL and FDIV operations. In that, "the parameter" is loaded from the instruction's indexed address operand, and the prenorm routine is executed once for each of the values mentioned. This symbol is mandatory.
- **ACC**: This address is used to store the value of rA at the start of a binary floating point operation (FADD, FSUB, FMUL, FDIV or FCMP), after which the operand value will be loaded into rA. This symbol is mandatory.
- **FADD**, **FSUB**, **FMUL**, **FDIV**, **FLOT**, **FIX**, **FCMP**: The implementation of any floating point operation that is actually used must start at the address marked by the corresponding symbol. Although it is not strictly necessary to have all symbols/addresses in place, any floating point operation for which one is not present will fail when its execution is attempted.
- **EXIT**: This address signals the end of a floating point operation or the prenormalization that occurs before it. When the floating point module program counter reaches this address, control is transferred back to MIX after restoring registers and setting rA/comparison indicator as indicated in the previous section. This symbol is mandatory.
- **OFLO**: If the program counter reaches this address, an error is raised indicating that the overflow indicator was unexpectedly set. This condition causes the execution of the current floating point operation to be aborted.
- **EXPUN**: The program counter reaching this address signals exponent underflow. Execution of the floating point program does continue when this occurs.
- **EXPOV**: The program counter reaching this address signals exponent overflow. Execution of the floating point program does continue when this occurs.
- **FIXOV**: The program counter reaching this address signals overflow during float to fix conversion (i.e. the FIX operation). Execution of the floating point program does continue when this occurs.
- **DIVZRO**: The program counter reaching this address signals division by zero. Execution of the floating point program does continue when this occurs.

6.4 Floating point debugging

As indicated before, execution of floating point operations is basically considered to be atomic, which is to say that all instructions that the floating point module needs to execute to perform a floating point operation, which includes any prenormalization, are executed in one tick. When MixEmul performs a step or is running, both according to the definitions in section 2.3, this translates into the tick counter being increased in accordance with the specifications of section 4.2.1 of TAOCP. However, it is possible to trace the execution of a floating point instruction using the Tick button.
If the Tick button is pressed when the floating point module is about to execute a floating point operation, MixEmul switches to *Module mode*. In Module mode, each time the Tick button is pressed, the floating point module executes one instruction of the floating point program. As with the main MIX module, execution can be followed using the memory region (see section 2.6), although in this case on the Floating Point tab. Using either the Step or Run buttons will resume execution in the way just described.
When the floating point module finishes executing a floating point operation, either successfully or due to an error condition, MixEmul switches execution mode back to what it was before execution of the floating point operation started. After that, regular operation continues.

Besides tracing the execution of floating point operations step-by-step using the Tick button, it is also possible to set breakpoints in the floating point module. When a floating point module breakpoint is hit then program execution is interrupted as usual, with the exception that MixEmul will be in Module mode. The Tick, Step and Run buttons operate accordingly.

7 Profiling

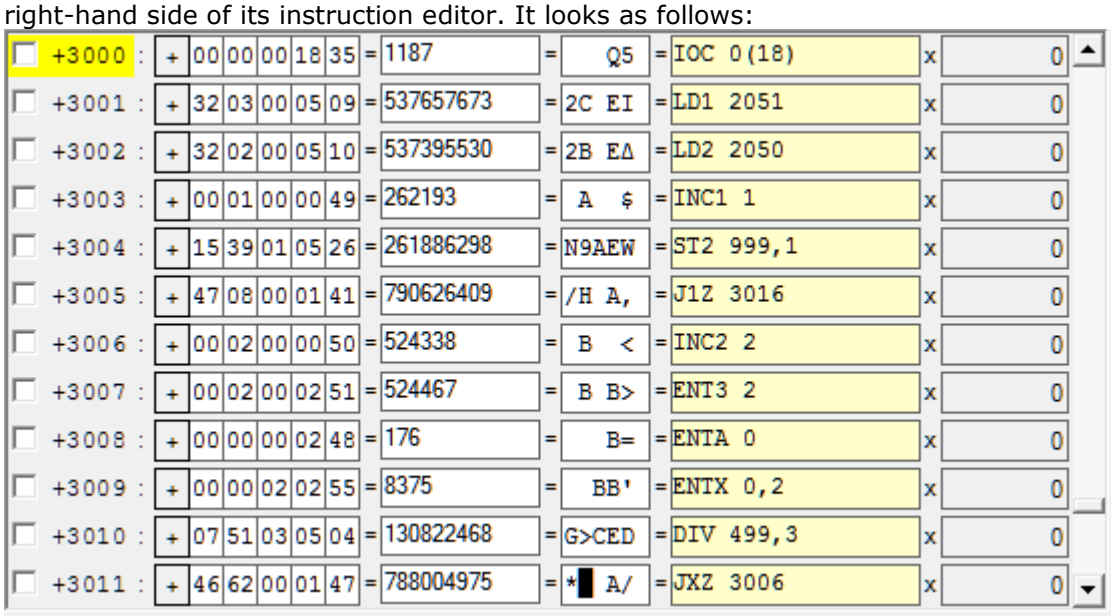
7.1 The concept

Often, it is not only relevant to see if a program works, but also what the performance characteristics of the program are. This is true for many if not most programs, and therefore also for MIXAL programs running in MixEmul.
One way to measure the performance characteristics of a program is counting how many times its instructions are executed in the course of a program run, and/or how much time is spent on that execution.
The measurement of these figures, and others in more complex scenario's, is called *profiling*.

7.2 The MixEmul profiler

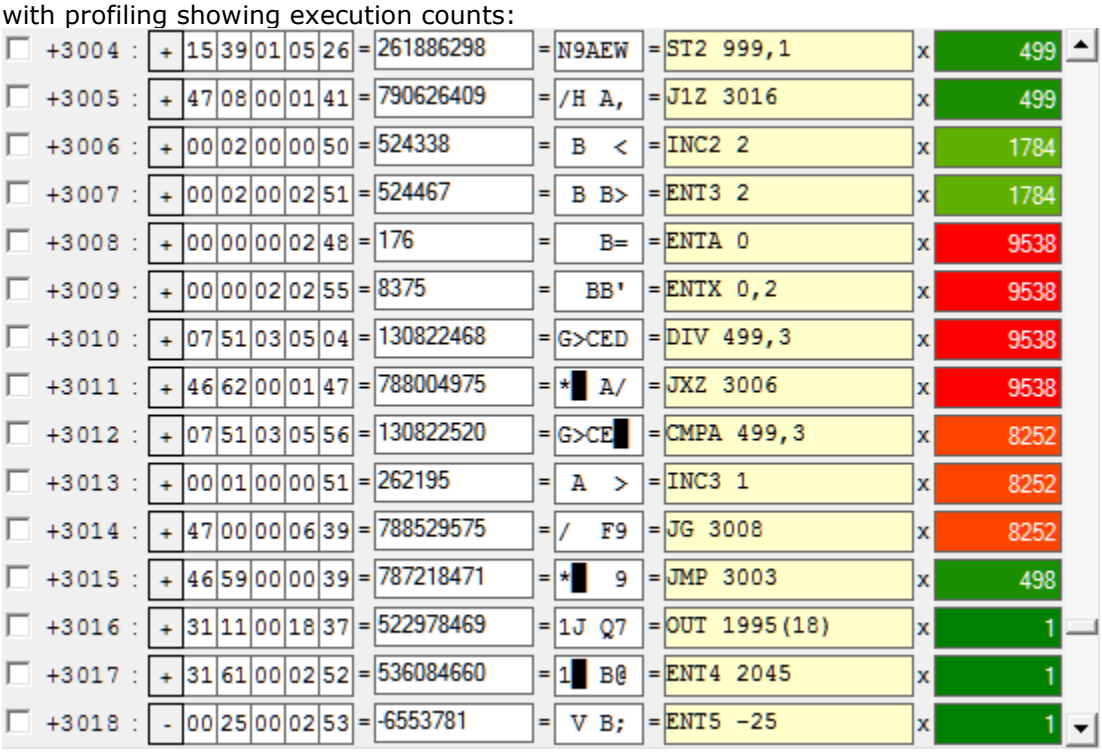
7.2.1 Usage

The MixEmul profiler can be enabled using the "Enable profiling" option in the Tools menu. When selected, a counter field will be shown for each word in the memory editor (see section 2.6), at the right-hand side of its instruction editor. It looks as follows:

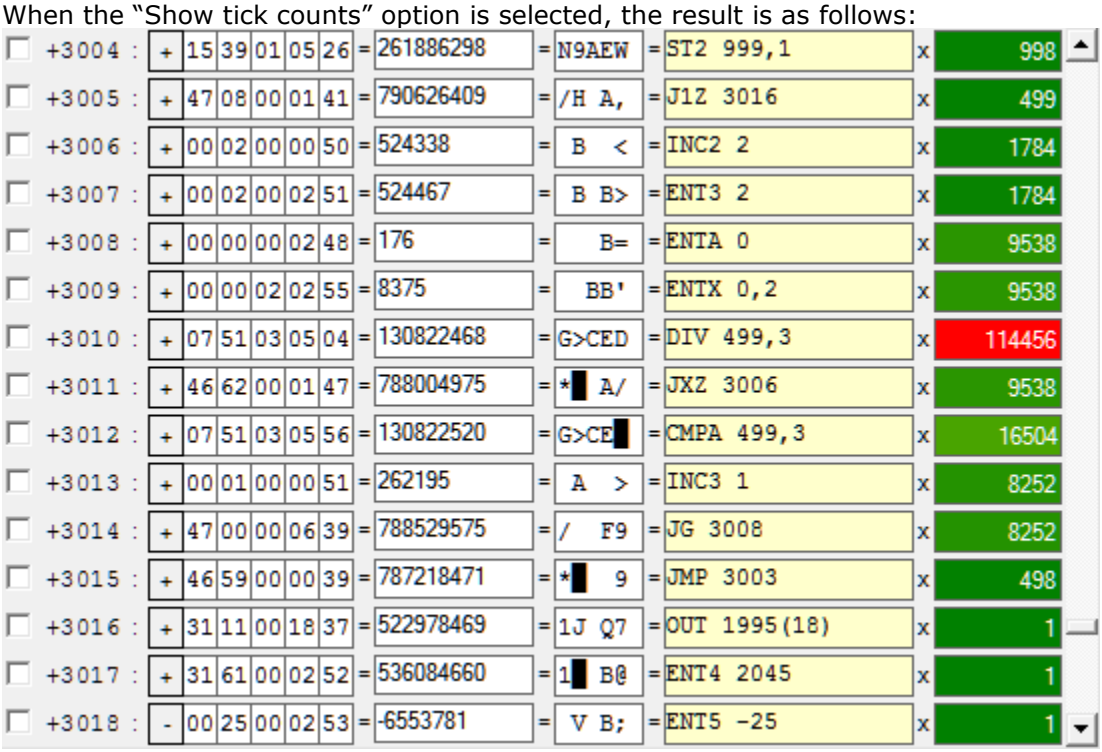


By default, upon execution of a Tick, Step or Run command (see section 2.3), the counters will be updated to reflect how many times each instruction is executed. This can be changed with the "Show tick counts" option in the Tools menu; then the counters will show how many ticks are spent on the execution of each instruction.

As an example, the following is what is displayed after the Primes example program has been run, with profiling showing execution counts:



When the "Show tick counts" option is selected, the result is as follows:



As these screenshots show, by default the profiling counters are color coded from dark green (lowest execution/tick counts) to bright red (highest execution/tick counts). This can be switched off using the preferences window (see chapter 9).

7.2.2 Additional notes

The following remarks apply to the MixEmul profiler:

- The profiling counters will be reset when MixEmul is reset in its entirety, or when a program is loaded using the assembler (see chapter 3). They can also be manually reset independently from other parts of MixEmul using the "Reset counts" option in the Tools menu.
- When enabled, profiling is not only performed for the main memory region, but also for the instructions in the floating point module.
- Disabling profiling, using the "Enable profiling" option in the Tools menu, does in itself not *reset* the profiling counters, even though they are hidden. However, *updates* to the counters will no longer occur until profiling is switched back on.

8 Using the MIX loader

8.1 Introduction

Exercise 26 of section 1.3.1 of TAOCP describes that MIX includes a GO button, which performs the following actions:

1. One card, containing (the first part of) the loader program is read into memory locations 0 to 15.
2. When reading has finished, the program counter is set to 0, as is rJ.
3. Execution is started.

MixEmul provides an implementation of this feature, an example loader program, and the ability to create and use export files that can be used in conjunction with it. Each of these is described in the following section.

8.2 The Go button

The MixEmul implementation of the MIX loader is activated by pressing the Go button on the main window's control strip (see section 2.3). When this button is pressed, the steps discussed in the previous section are performed. In other words, the following MIX instructions are effectively executed without loading them into memory:

```
IN   0(16)
JBUS *(16)
JMP  0
```

As indicated above, after/with the JMP instruction, rJ is also set to 0, which is an action for which no MIX instruction exists.

It should be noted that while the loader is running, the tick counter is increased as usual for the instructions just mentioned, and any other busy devices also continue to operate. Resetting rJ to 0 is considered to take 1 tick.

8.3 The loader program

MixEmul comes with an example loader program that can be used to process information and transfer cards that conform to the specifications discussed in exercise 26 of section 1.3.1 of TAOCP, with the following remarks:

- As there is no way to "overpunch" a minus (-) over a digit to indicate the corresponding word value is negative, a negative word value on an information card needs to be specified by replacing its least significant (last) digit with a character between space (equaling -0) and the letter I (equaling -9);
- This negative numbering scheme is also applied to addresses, to support the loader program loading information cards into the control memory area and setting the program counter to a negative value. Of course, MixEmul must be in Control mode when the loader program runs for negative memory addresses to be accessible and thus for this to work. Please refer to chapter 5 for more information about Control mode and negative addressing.
- Due to the support for negative addresses, the loader program spans three cards instead of the indicated limit of two.
- The program does not verify if information or transfer cards contain valid start/transfer addresses or, for information cards, valid word values. That is to say, the loader program will happily load information cards that overwrite the loader program as it is running, or contain word values that exceed the MixEmul 6-bit byte maximum word magnitude of 1,073,741,823. Of course, any chaos that ensues is the sole responsibility of whoever it is that loaded the cards in question into the card reader(…)

The loader program is included in the Sample programs folder in the MixEmul ZIP file.

8.4 Making and using exports

8.4.1 Export card deck description

From the assembler and the memory region it is possible to easily create export card decks (in the shape of text files) that can be used with the MixEmul loader implementation. These card decks start with the cards that contain the loader program, and are followed by the information and transfer cards that encode the assembled program/memory area that is exported.

By default, the loader program cards that are exported contain the example loader program that is supplied with MixEmul, which is described in the previous section of this chapter. It is possible to have another loader program be exported by specifying the desired loader instruction cards in the MixEmul preferences window (see chapter 9).
With this, the following should be noted:

- A maximum of three loader cards can be configured;
- When exports are made, any empty instruction cards are skipped. That means that it is possible to prevent any loader instruction cards being exported by configuring three empty cards.

It is pointed out that in case one wants to use another loading program, a good approach to use is the following:

1. Write/obtain the program in MIXAL, having it start at address 0,
2. Load it using the assembler, and
3. Output the memory addresses that contain the program to the card punch.

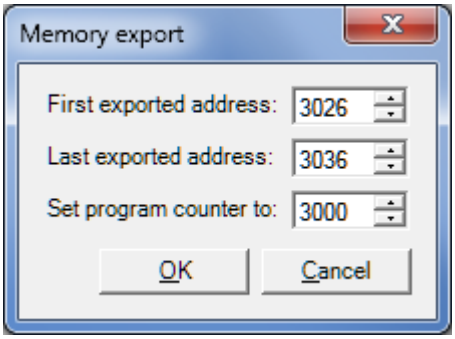
As the maximum number of loader program cards is three, the loader program cannot occupy more than 48 consecutive addresses.

8.4.2 Exporting assembled programs

After a MIXAL program has been successfully assembled, it can be exported using the Export button on the assembly result window (see section 3.4). When the button is pressed, a window is shown with which the file name and location of the card deck file can be specified.

8.4.3 Exporting memory areas

It is possible to export (part of) the MixEmul memory contents using the Export button on either tab in the memory region of the MixEmul main window (see section 2.6). When the button is pressed, first a window is shown with which it is possible to specify what memory area is to be exported (by means of specifying the first and last address), and what the program counter should be set to once the memory area has been loaded. It looks like this:



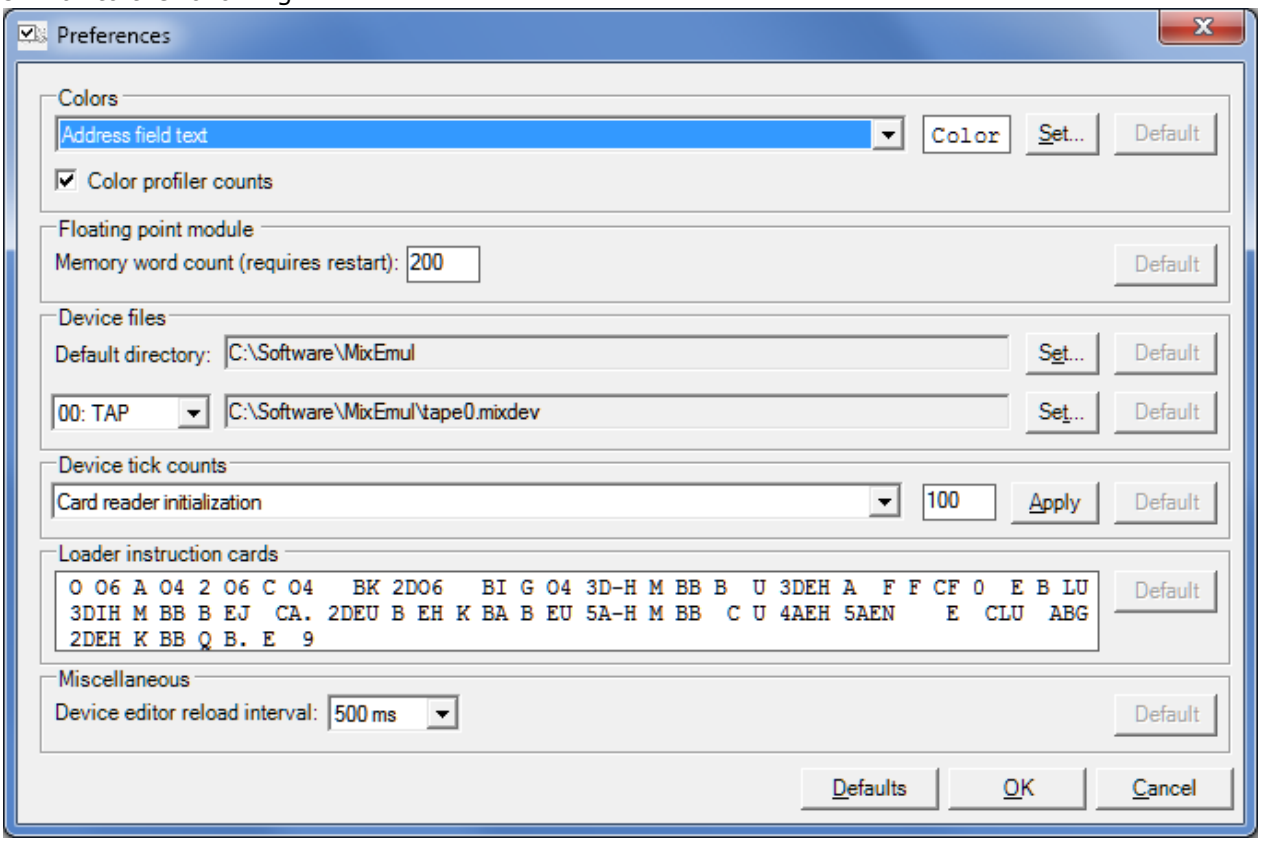
When these parameters have been specified and OK is pressed, another window is shown with which the file name and location of the card deck file can be specified.

8.4.4 Using exported card decks

Card decks that have been created using the methods described above can be loaded into the card reader with the Load button on the card reader tab of the device editor (see section 4.2.2). The card deck can then be executed by pressing the Go button discussed in section 8.2.

9 Preferences

The preferences window can be accessed via the Preferences option in the Tools menu. It looks similar to the following:



The following preferences can be changed:

- A selection of the colors that are used throughout MixEmul. Most of the color names speak for themselves, but the following might need some explanation:
  - Address text vs. Address field text. The Address text color is used for the address numbers in the Memory region of the main window. The Address field text color is used for the address fields of the MIXAL programs as they are shown in the assembly result window. Similarly, the Location field, Op field and Comment text colors are used for the respective fields of the assembly result window.
  - The Rendered text color is used for text that displays information (e.g. in editors) which has not yet been edited. The editing text color is used for text that has been edited but the changes to which have not yet been applied.
  - The loaded instruction text and background colors are used for instruction editors into which a MIXAL instruction has been loaded by the assembler (see also section 2.6).
- If color coding should be applied to the profiling counters when profiling has been enabled. Please refer to chapter 7 for more information about MixEmul's profiling feature.
- The number of memory words that is made available to the floating point module. Please ensure that a number of memory words is selected that is sufficient for loading and running the floating point program. It is noted that the number entered here is increased by 9, to allow the contents of the registers to be stored and loaded when execution respectively enters and leaves the floating point module.
  MixEmul must be restarted to apply any changes to this setting.
  Please refer to chapter 6 for more information about the floating point module.
- The default directory for device files. This directory is used to store all device files for which a specific file hasn't been selected using the preference that is discussed next. Note that any existing device files are not automatically moved if this directory is changed.
- The device files for individual devices. Note that an existing device file is not automatically moved or renamed if this setting is changed.
- Tick counts for a number of I/O operation steps (see also the discussion of the Devices region in section 0). This enables you to modify the speed at which the MixEmul devices operate.

- The contents of the cards that are written at the top of export files that are made from either the memory region (see section 2.6) or the assembler (see section 3.4). Please refer to chapter 8 for more information about the MIX loader and the aforementioned export features.
- The minimum interval between reloads of the contents of the visible device editor, upon modification of the device's file on disk.

For each preference it is possible to revert to the default setting using the respective Default button at the right. All preferences can be reset to defaults at once with the Defaults button at the bottom of the window.