Language and development environment:
MIOS and it's applications run on the PIC series of processors. Applications may be written in C or assembly. The MBMixer is written in assembly, as that is the language I am most comfortable with. During development of the MBMixer, a new development environment was announced for MIOS, that uses GCC and SDCC instead of the MPLAB package supplied by Microchip. I may move the code over to the new environment at some time, but for this release, I am still working inside MPLAB for development.

Code Structure:
The purpose of the code is to receive and organize MIDI control events, and convert them to gain settings for the channel boards. I designed the code to work between two memory tables. One records the MIDI control data for the current board, organized by channel. The second memory table holds the gain settings to be sent to the array of PGA chips that control volume. Between them, of course, are the calculations that describe how the board will behave. The table of MIDI control data is labeled MIDI1. Within the table are MIDI2, MIDI3 etc. All the way to MIDI16, but MIDI1 and occasionally MIDI9 are the only labels referenced by the program. Each of these channels has eight bytes of space reserved for use. Six of the eight bytes are currently used, described as follows:

MIDI1 + 0: Flags, each bit controls an on/off condition, as described below.
MIDI1 + 1: Volume. Data value 0-127.
MIDI1 + 2: Expression. Data value 0 – 127.
MIDI1 + 3: Balance/Pan. Data value 1-64-127.
MIDI1 + 4: Effects Level 1. Data value 0-127.
MIDI1 + 5: Effects Level 2. Data value 0-127.
MIDI1 + 6: Not Used.
MIDI1 + 7: Not Used.

MIDI1 + 8 is the start of the next channel, MIDI2.

The Flags are described as follows:
Bit 0: if SET Effects 1 is “Pre-Fader”, if cleared Effects 1 is “Post-Fader”.
Bit 1: if SET Effects 2 is “Pre-Fader”, if cleared Effects 2 is “Post-Fader”.
Bit 2: if SET Effects 1 muted.
Bit 3: if SET Effects 2 muted.
Bit 4: if SET this channel includes Effects Loops.
Bit 5: Not Used.
Bit 6: Not Used
Bit 7: Used internally to flag this channel as modified.

MIOS will call the USER_MPROC_NotifyReceivedEvent routine whenever a complete MIDI event is received. This routine must sort the received control into the MIDI table described above. If any change is made to a MIDI channel, bit 7 of that channels flags must also be set. After changes have been made, the Mid2Gain routine from setlevels.inc is called, which scans the MIDI data tables, and re-calculates all channels that have been modified. The results of Mid2Gain are stored in the Channel_Gain table, starting at Channel_Gain_0. The math used to calculate each gain setting depends on the current settings of Volume, Expression, Balance/Pan, Effects1, Effects2, and the master volume level, as well a all the flag settings for that channel. The last step for each gain setting is to call “LogByte” from logbyte.inc, which will apply a log curve to the result. Once all the channels have been recalculated, a call to PGA_SEND_GAIN from pga.inc will copy the data in the gain table out to the PGA chips. Then the DISPLAY_UPDATE_REQ bit is set, so that the display will be updated the next time around. Of course, there is more detail to it than that, but the overview should give you a fair start into understanding the process. I have been careful to over comment my code, so that

people that are new to assembly would have a fair shot at understanding it.