Essentials of the MIDI protocol

The MIDI protocol which transfers musical information between keyboards and synthesizers, etc. is described here.

Hexadecimal numbers

You should become familiar with hexadecimal numbers to understand better the organization of the MIDI protocol. Here is a basic review:

The symbol 165 is a shorthand representation of a particular number. More specifically, 165 can be represented by:

$$\mathbf{165} = \mathbf{1} \times 10^2 + \mathbf{6} \times 10^1 + \mathbf{5} \times 10^0$$

I will write 165_{10} to mean the number 165 in base-10 representation.

Using other number bases other than base-10 is an equally valid way to represent the number 165_{10}. Computers work well with binary numbers, so let's convert 165 to binary form:

$$165_{10} = \mathbf{1} \times 2^7 + \mathbf{0} \times 2^6 + \mathbf{1} \times 2^5 + \mathbf{0} \times 2^4 + \mathbf{0} \times 2^3 + \mathbf{1} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{1} \times 2^0 = \mathbf{10100101_2}$$

Hexadecimal numbers are a base-16 representation of numbers and are useful for humans when dealing with binary numbers. Every 4 binary digits can be repesented by one hexadecimal digit. Below is a table of equivalence between decimal, hexadecimal, and binary number representations for the numbers between 0 and 15:

number base equivalences

dec	hex	bin	dec	hex	bin
0	0	0	8	8	1000
1	1	1	9	9	1001
2	2	10	10	A	1010
3	3	11	11	B	1011
4	4	100	12	C	1100
5	5	101	13	D	1101
6	6	110	14	E	1110
7	7	111	15	F	1111

Using the above table, you should quickly be able to convert the number 10100101_2 to hexadecimal by grouping every four binary digits into a group and looking up the hex equivalent in the table.

$$10100101_2 \Longrightarrow 1010\ 0101 \Longrightarrow A\ 5 \Longrightarrow A5_{16}$$

Since hexadecimal numbers contain digits for 10 through 15, it is standard practice to use the letters of the alphabet to reresent those digits, starting at $A = 10$.

In base-36, you would use all letters of the alphabet to represent all of the possible digits. Try to convert base-36 words to base-10, e.g.:

$$
\begin{aligned}
the_{36} &= t \times 36^2 + h \times 36^1 + e \times 36^0 \\
&= 29 \times 36^2 + 17 \times 36^1 + 14 \times 36^0 \\
&= 37584 + 612 + 14 \\
&= 38210_{10}
\end{aligned}
$$

Hexadecimal numbers are often written with an `h` following them to distinguish them from decimal numbers, *e.g.*: A5h indicates that A5 is a hexadecimal number. There is no confustion in this case, but you would need to know if 42 is in base-10 or in base-16 since it is an entirely different number in each base and you can't tell which base it is by looking at it out of context.

In the C programming language, hexadecimal numbers are indicated by prepending the string "0x" in front of the number, *e.g.*: `0xA5` which would be the same as `A5h`. I will use the `0x` notation later on this page.

Organization MIDI commands and data in a byte of information

MIDI bytes range between 0 and 255, or shown below in various representations:

Numeric range of MIDI bytes

decimal	hexadecimal	binary
0	0	0
255	FF	11111111

Note that a byte is a binary number that contains 8 digits. Looking at the binary range above you should be able to see that the range from 00000000 to 11111111 will cover all possible combinations of digits in a byte.

MIDI commands and data are distinguished according to the most significant bit of the byte. If there is a zero in the top bit, then the byte is a data byte, and if there is a one in the top bit, then the byte is a command byte. Here is how they are separated:

Division of data and commands by values

decimal	hexadecimal	binary
DATA bytes:		
0	0	00000000
...	...	...
127	7F	01111111
COMMAND bytes:		
128	80	10000000
...	...	...
255	FF	11111111

Furthermore, command bytes are split into half. The most significant half contains the actual MIDI command, and the second half contains the MIDI channel for which the command is for. For

example, `0x91` is the note-on command for the second MIDI channel. the `9` digit is the actual command for note-on and the digit `1` specifies the second channel (the first channel being `0`). The `0xF0` set of commands do not follow this convention.

Here is a table of the MIDI commands:

MIDI commands

0x**8**0	Note Off
0x**9**0	Note On
0x**A**0	Aftertouch
0x**B**0	Continuous controller
0x**C**0	Patch change
0x**D**0	Channel Pressure
0x**E**0	Pitch bend
0x**F**0	(non-musical commands)

The messages from 0x80 to 0xEF are called *Channel Messages* because the second four bits of the command specify which channel the message affects. The messages from 0xF0 to 0xFF are called *System Messages*; they do not affect any particular channel.

MIDI messages

A MIDI command plus its MIDI data parameters to be called a *MIDI message* . The minimum size of a MIDI message is 1 byte (one command byte and no parameter bytes). The maximum size of a MIDI message (note considering 0xF0 commands) is three bytes. A MIDI message always starts with a command byte. Here is a table of the MIDI messages that are possible in the MIDI protocol:

Command	Meaning	# parameters	param 1	param 2
0x**8**0	Note-off	2	key	velocity
0x**9**0	Note-on	2	key	veolcity
0x**A**0	Aftertouch	2	key	touch
0x**B**0	Continuous controller	2	controller #	controller value
0x**C**0	Patch change	2	instrument #	
0x**D**0	Channel Pressure	1	pressure	
0x**E**0	Pitch bend	2	lsb (7 bits)	msb (7 bits)
0x**F**0	(non-musical commands)			

I won't discuss the 0xF0 set of commands (System messages) here very much, but here is a basic table of them:

command	meaning	# param
0xF0	start of system exclusive message	variable
0xF1	MIDI Time Code Quarter Frame (Sys Common)	
0xF2	Song Position Pointer (Sys Common)	
0xF3	Song Select (Sys Common)	
0xF4	???	

0xF5	???	
0xF6	Tune Request (Sys Common)	
0xF7	end of system exclusive message	0
0xF8	Timing Clock (Sys Realtime)	
0xFA	Start (Sys Realtime)	
0xFB	Continue (Sys Realtime)	
0xFC	Stop (Sys Realtime)	
0xFD	???	
0xFE	Active Sensing (Sys Realtime)	
0xFF	System Reset (Sys Realtime)	

Running status should be mentioned around here...

---

craig@ccrma.stanford.edu