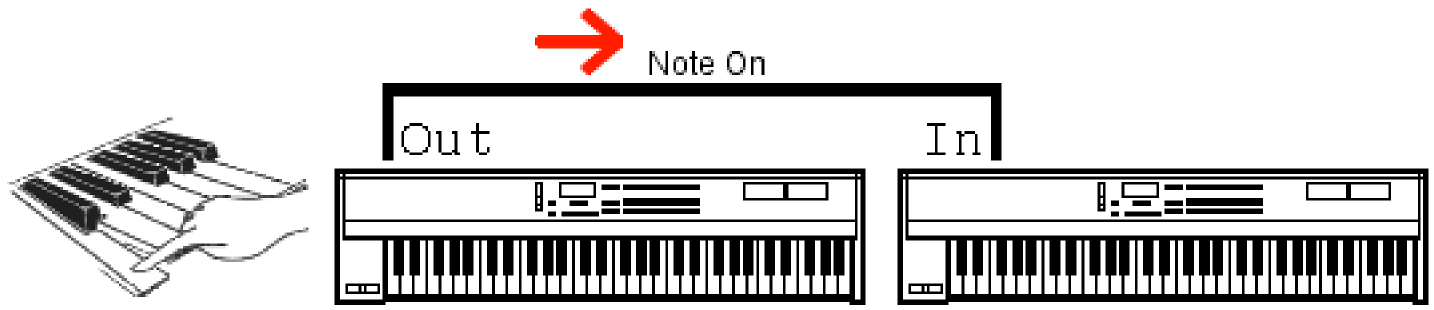
The MIDI Protocol

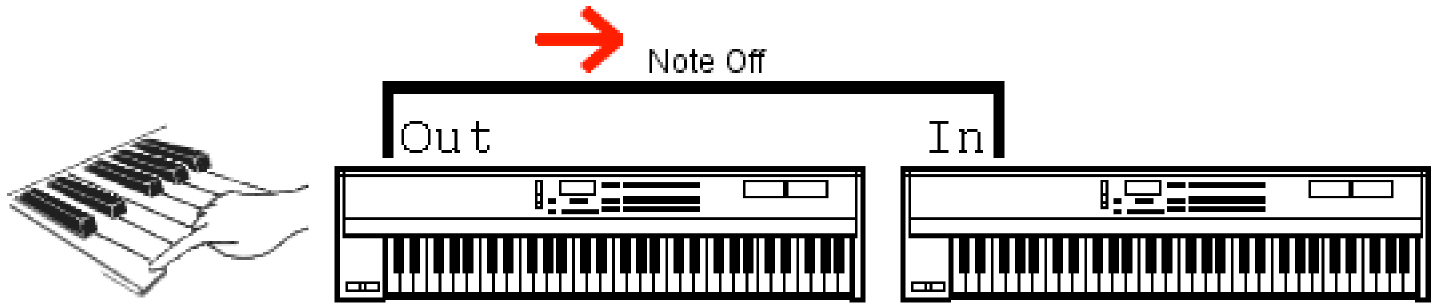
MIDI (Musical Instrument Digital Interface) defines what data is sent from one musical device to another in order to communicate a musical event. It consists of both a simple **hardware interface** and a **software protocol**. *MIDI does not contain any audio information*. Rather, MIDI communicates performance information like triggering a note, using the sustain pedal, etc.

For example, when a key is pressed on a MIDI keyboard, a **Note On** message is sent out the keyboard's **MIDI Out** jack. If a MIDI cable is connected from that **MIDI Out** to the **MIDI In** of another keyboard or sound module, the **Note On** message will pass through the cable to the other instrument. The **Note On** message describes which note was triggered (the **MIDI note #**), how fast the key was pressed (key **velocity**), and what **MIDI channel** it is on.

The receiving device interprets the event and plays the note with the sound assigned to that particular MIDI channel.



The note continues until the finger is lifted off the key, which sends a **Note Off** message.



MIDI software protocol

There are 8 types of MIDI Messages:

Note Off	(stops playing a note)
Note On	(starts playing a note)
Aftertouch	(Play a key, then press down harder. Use this 0-127 range like a continuous controller.)
Continuous controller	(sliders, knobs, pedals, modulation wheels, etc., with a 0-127 range)
Patch change	(sends a number to a sound module to change it to a specific sound or "patch")
Channel Pressure	(like aftertouch but across the entire MIDI channel instead of key specific)
Pitch bend	(like a continuos controller but with 1000's of values for fine pitch bending control)
System Exclusive	(for proprietary things like storing banks of sounds, etc.)

Most MIDI Messages contain 3 bytes of data. The first byte is called a **Status Byte**, and the following bytes are called **Data Bytes**.

- The **Status Byte** specifies which type of MIDI message is being sent and on what MIDI channel.
- The **Data Bytes** contain performance information according to what type of MIDI message it is.

For example, when you play a note on a keyboard, a **Note On** command gets sent in 3 bytes:

STATUS BYTE	DATA BYTE	DATA BYTE
1 0 0 1 0 0 0 0	**0 0 1 1 1 1 0 0**	**0 0 1 1 1 1 1 1**

Let's look at it bit by bit. What could be more fun?! **:-)**

The **status byte** always starts with a **1**. It says, *"Hi, I am a status byte! A MIDI message is coming!"*

1 _ _ _ _ _ _ _   _ _ _ _ _ _ _ _   _ _ _ _ _ _ _ _

The next 3 bits tell *which* **MIDI message** is being sent . **001** stands for **Note On**. "*Hello, I'm a note!"*

1 **0 0 1** _ _ _ _   _ _ _ _ _ _ _ _   _ _ _ _ _ _ _ _

The next 4 bits tell what **MIDI channel** to play the note on. The sound assigned to this particular MIDI channel will be the one to play. **0000** stands for **MIDI channel 1**. *"I'm on MIDI channel 1!"*

1 0 0 1 **0 0 0 0**   _ _ _ _ _ _ _ _   _ _ _ _ _ _ _ _

(Why does **0000** stand for **MIDI channel 1**?! Here's why! **0000-1111** in binary is **0-15** in decimal. We call them **MIDI channel 1-16** only because it would sound strange to say "channel zero".)

The **data bytes** start with a **0**. *"Stand by for data bytes!"*

1 0 0 1 0 0 0 0   **0** _ _ _ _ _ _ _   **0** _ _ _ _ _ _ _

The remaining 7 bits in the first **data byte** specifies which **MIDI note number** is being played. **0111100** stands for **Middle C**, or **MIDI note #60**. *"Don't hate me because I'm middle C!"*

1 0 0 1 0 0 0 0   0 **0 1 1 1 1 0 0**   0 _ _ _ _ _ _ _

The remaining 7 bits in the second **data byte** specifies the **velocity** the note was played.
1111111 is the highest **velocity** value possible (playing the note as hard as you can). *"Whack!!"*

1 0 0 1 0 0 0 0   0 0 1 1 1 1 0 0   0 **1 1 1 1 1 1 1**

That's a **note on** in just three tiny little bytes! We could never have this much fun looking at audio data since we'd have to look at **millions** of bytes instead of just three!

A **Note Off** message looks exactly the same as a **Note On** except the 3 bits in the first byte would be **000** instead of **001**. The Note Off velocity (how fast you lift your finger up) usually gets ignored.

Look at this chart to see what information all of the 8 types of MIDI messages contain:

STATUS BYTE 1 _ _ _ . . . . _ _ _ _ — **DATA BYTE** 0 _ _ _ _ _ _ _ — **DATA BYTE** 0 _ _ _ _ _ _ _

These 3 bits say which MIDI message it is.

These 4 bits specify the MIDI channel .

The 7 bits in each data byte define specific data depending on what type of MIDI message it is.

000	**Note Off**	MIDI Channel (1-16)	note number (0-127)	velocity (0-127) <-- *ignored*
001	**Note On**	MIDI Channel (1-16)	note number (0-127)	velocity (0-127)
010	**Aftertouch**	MIDI Channel (1-16)	note number (0-127)	value (0-127)
011	**Continuous controllers**	MIDI Channel (1-16)	controller # (0-127)	value (0-127)
100	**Patch change**	MIDI Channel (1-16)	instrument # (0-127)	-
101	**Channel Pressure**	MIDI Channel (1-16)	pressure (0-127)	-
110	**Pitch bend**	MIDI Channel (1-16)	value (LSB)	value (MSB) (0-16384)
111	**System Exclusive**	(These contain more than 3 bytes and are exclusive to the manufacturer and model of the synthesizer being used. System Exclusive commands are beyond the scope of this lesson.)		

Let's look at **Continuous Controllers** next. They are things like modulation, volume, panning - most anything that can be controlled by a knob, wheel or slider. There are 128 (0-127) continuous controllers, but we only need to memorize a few of them: **volume** = **controller 7**, **panning** = **controller 10**, **modulation** = **controller 1**. Each of them has a value range of **0-127**. For instance, 000000 means volume is down, 111111 is all the way up and 010011 is somewhere in between.

Let's dissect a **volume** command! Remember, **volume** = **controller 7**. Seven is **0000111** in binary.

"Hello, I am a status byte!"

1 _ _ _ _ _ _ _   _ _ _ _ _ _ _ _   _ _ _ _ _ _ _ _

"*Hello, I'm a continuous controller!" (why 011? see chart above!)*

1 **0 1 1** _ _ _ _   _ _ _ _ _ _ _ _   _ _ _ _ _ _ _ _

"I'm on MIDI channel 1!"

1 0 0 1 **0 0 0 0**   _ _ _ _ _ _ _ _   _ _ _ _ _ _ _ _

"I'm volume!"

1 0 0 1 0 0 0 0   0 **0 0 0 0 1 1 1**   0 _ _ _ _ _ _ _

"I've just moved the volume slider up a tiny bit."

1 0 0 1 0 0 0 0   0 0 1 1 1 1 0 0   0 **0 0 0 0 0 0 1**

Continuous controllers usually go in streams. If we were to move the volume slider all the way up, 127 3-bit chunks would fly by, with the last 7 bits incrementing by 1 each time until **1111111** (127) is reached. It would go **0000001**, **0000010**, **0000011**, **0000100**, **0000101**, **0000110**, **0000111**, **0001000**, **0001001**, **0001010**, etc. *In case you haven't noticed, counting up in binary is fun!* :-D

We'll look at one last example - **pitch bend**. Pitch bend is a bit different than continuous controllers, which only range from **0-127.** Pitch bend ranges from **0-16383**! Wow! This bigger range allows for smooth, fine control over pitch. To get this big number, **pitch bend uses both of the data bytes together for the value**. But how can we get such a big number with 14 bits, when 7 bits can only add up to 127?

It's time for a lesson in binary.

The binary system (zeros and ones)

To understand binary, it may help to review the normal decimal system first.

The decimal system is base 10, meaning there are 10 values per digit (0-9).
Each digit is worth powers of 10 with the lowest value on the right.

n	**n**	**n**	**n**
x1000	x100	x10	x1
(10^3)	(10^2)	(10^1)	(10^0)

We automatically know that a number like 8,206 is "8 **thousand,** 2 **hundred**, and 6".

8	**2**	**0**	**6**
x1000	x100	x10	x1

We automatically know what 8,206 means because we learned the decimal system in school since we were kids. We read decimal as easily as we read our first language. We "speak decimal"! Most of us don't "speak binary" just as there are lots of languages we don't speak. So, we have to add it up each time to figure out what the corresponding value would be in decimal, similar to how one might use a Spanish to English translator. The good news is that it is easy to translate binary into decimal!

Binary is base 2, meaning there are 2 values per digit (0-1).
Each digit is worth powers of 2 with the lowest value on the right.

n	**n**	**n**	**n**
x8	x4	x2	x1
(2^3)	(2^2)	(2^1)	(2^0)

Compare this to the decimal system above. You may have to stare at it a while if you don't immediately get it. Go ahead! Stare!

We can't have a number like 8,206 in binary because in binary there are only 0's and 1's. 2-9 don't exist in binary. So let's look at another four digit binary number and translate it into decimal.

A binary number like 1011 would be: (**1** x 8) +(**0** x 4) + (**1** x 2) + (**1** x 1) = 11 in decimal.

1111111 in binary = 127 in decimal, because if we count what the 1's are worth from right to left and add them up, we get:
1 + 2 + 4 + 8 + 16 + 32 + 64 = 127. (127 shows up quite often in MIDI. We've already seen it several times in this lesson!)

1111111 in decimal is simply 1,111,111 (writing it with commas like we're used to). If we add up what each digit is worth, we get:
1 + 10 + 100 + 1,000 + 10,000 + 100,000 + 1,000,000 = 1,111,111.

It seems silly to add it up like that since decimal is our language! When we see **1,111,111** we automatically know it's **"one**

million, one hundred eleven thousand, one hundred eleven”. Don’t you wish you spoke binary too? You can if you practice! Try memorizing that 0010 is “two”, 1000 is “eight” and 1010 is “ten” and 1111 is “fifteen”. Add them up to check!

Just to finish up our binary lesson, let’s count to 10 in both binary and decimal.

Decimal is easy since we know it so well! We count up to 9 and then “carrying over” to get 10.
1, 2, 3, 4, 5, 6, 7, 8, 9, 10 !

In binary we can only count up to 1, then we “carry over” to get 10 (which means “two”), then add one more to get 11 (“three”), and so on and so forth up to 1010 (“ten”).
0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010 !

Try counting in binary with four fingers. Just be careful of 2 or 4, depending on which way you hold your hand!

So, we’ve learned that binary works just like our normal decimal system but instead of base 10, it’s base 2. From now on, if a friend asks you, “What the heck is binary?” you can say, “Oh, it’s just another number system, but instead base 10 like we’re used to, it’s base 2.” Then you can show them how to count from 1 to 10 with four fingers!

To get back to our original question, how can a mere 14 bits (in binary) have a huge value like 16,383 (in decimal) for our pitch bend value? Using what we’ve already learned and extending it to 14 digits, we get:

n	**n**	**n**	**n**	**n**	**n**	**n**	**n**	**n**	**n**	**n**	**n**	**n**	**n**
x8102	x4096	x2048	x1024	x512	x256	x128	x64	x32	x16	x 8	x4	x2	x1
(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^{9})	(2^{8})	(2^{7})	(2^{6})	(2^{5})	(2^{4})	(2^{3})	(2^{2})	(2^{1})	(2^{0})

So **11111111111111** is **8102 + 4096 + 2048 + 1024 + 512 + 256 + 128 + 64 + 16 + 8 + 4 + 2 + 1** = **16,383** in decimal.

This concludes our lesson in binary!

Now let’s look at the bits and bytes of a **pitch bend** command.

“Hello, I am a status byte!“

1 _ _ _ _ _ _ _   _ _ _ _ _ _ _ _   _ _ _ _ _ _ _ _

“*Hello, I’m pitch bend!”*

1 **1 1 0** _ _ _ _   _ _ _ _ _ _ _ _   _ _ _ _ _ _ _ _

“I’m on MIDI channel 1!”

1 0 0 1 **0 0 0 0**   _ _ _ _ _ _ _ _   _ _ _ _ _ _ _ _

“I’ve just moved the pitch bend wheel all the way up.”

1 0 0 1 0 0 0 0   0 **1 1 1 1 1 1 1**   0 **1 1 1 1 1 1 1**

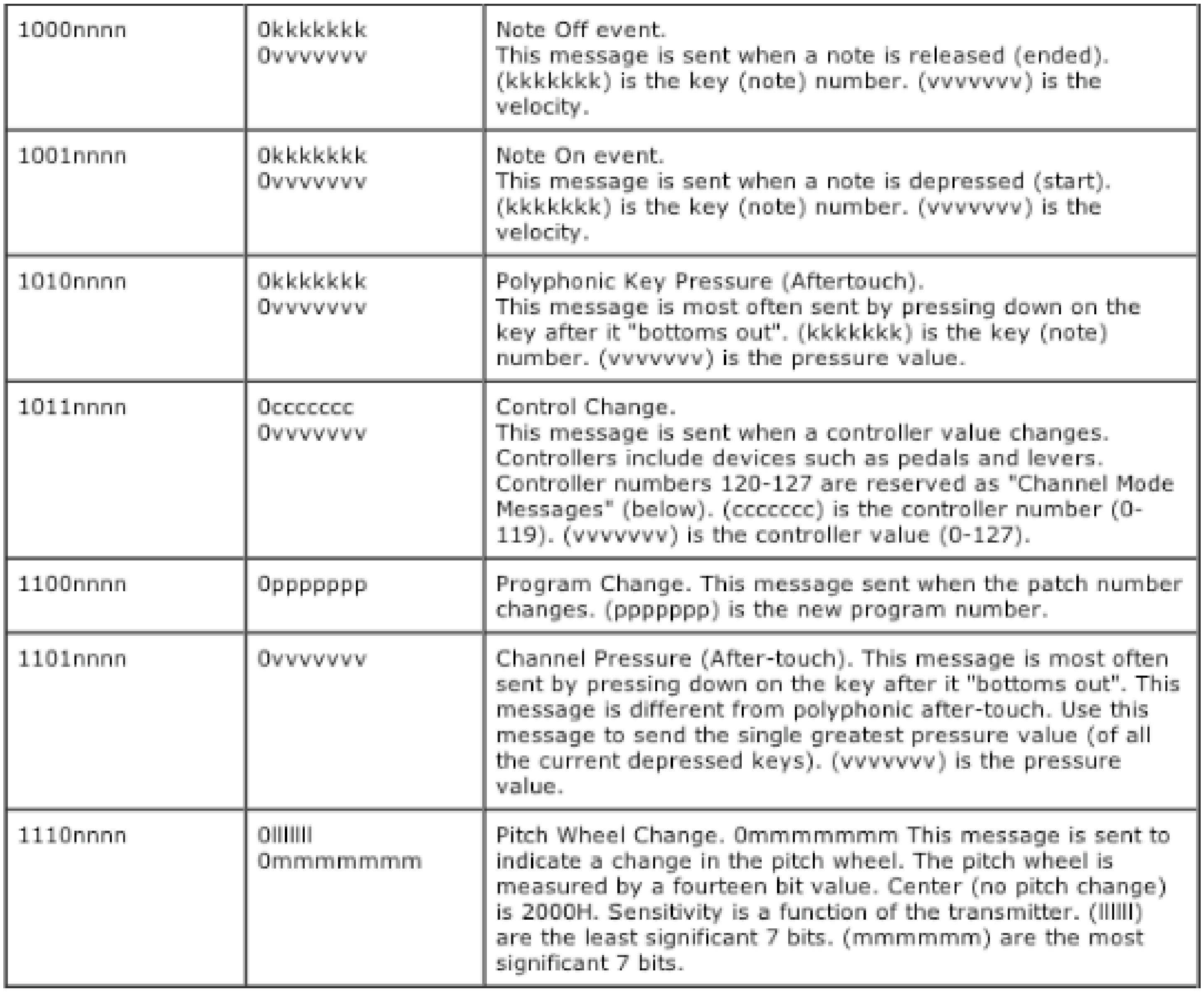
You know it. **11111111111111** = **16,383**! You’re a binary genius!

Many keyboards will have a chart like this in the back of their user manuals. You may also see this on websites for MIDI products in the “specs” section. The Status bytes (left column) and Data bytes (middle column) and list of MIDI messages (right column) should look familiar by now.

The nnnn in the status byte stands for the MIDI channel and the other letters, like kkkkkkkk and vvvvvvv, represent things like key (MIDI note number) and velocity. It is all explained in the right column.

It can’t hurt to stare at this chart for a while until it sinks in.

Ready? Go!



1000nnnn	0kkkkkkk 0vvvvvvv	Note Off event. This message is sent when a note is released (ended). (kkkkkkk) is the key (note) number. (vvvvvvv) is the velocity.
1001nnnn	0kkkkkkk 0vvvvvvv	Note On event. This message is sent when a note is depressed (start). (kkkkkkk) is the key (note) number. (vvvvvvv) is the velocity.
1010nnnn	0kkkkkkk 0vvvvvvv	Polyphonic Key Pressure (Aftertouch). This message is most often sent by pressing down on the key after it "bottoms out". (kkkkkkk) is the key (note) number. (vvvvvvv) is the pressure value.
1011nnnn	0ccccccc 0vvvvvvv	Control Change. This message is sent when a controller value changes. Controllers include devices such as pedals and levers. Controller numbers 120-127 are reserved as "Channel Mode Messages" (below). (ccccccc) is the controller number (0-119). (vvvvvvv) is the controller value (0-127).
1100nnnn	0ppppppp	Program Change. This message sent when the patch number changes. (ppppppp) is the new program number.
1101nnnn	0vvvvvvv	Channel Pressure (After-touch). This message is most often sent by pressing down on the key after it "bottoms out". This message is different from polyphonic after-touch. Use this message to send the single greatest pressure value (of all the current depressed keys). (vvvvvvv) is the pressure value.
1110nnnn	0lllllll 0mmmmmmm	Pitch Wheel Change. 0mmmmmmm This message is sent to indicate a change in the pitch wheel. The pitch wheel is measured by a fourteen bit value. Center (no pitch change) is 2000H. Sensitivity is a function of the transmitter. (llllll) are the least significant 7 bits. (mmmmmm) are the most significant 7 bits.

You are a MIDI genius.