WAVE File Player

WAV Player

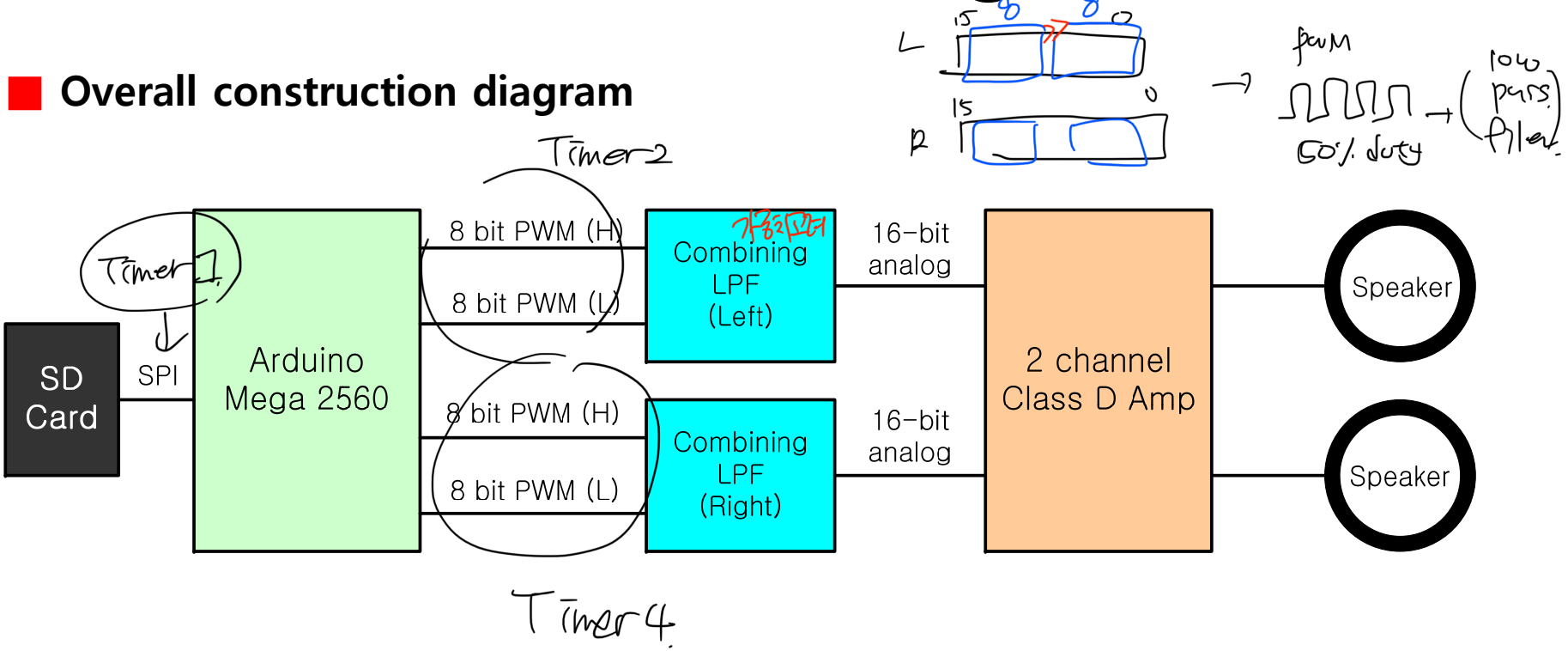
- Purpose : To make a fancy music player using the Arduino Mega 2560
- File type to be played : WAV file
  - MP3 file is more common file type for musics than WAV.
  - However, we need a MP3 decoder IC to play MP3 files because MP3 contains compressed information.
  - WAV file is a container format for uncompressed audio files

MP3 decoder IC

- Common sampling frequency in digital audio : 44.1 KHz.
  - It means that we need to update audio data 44100 times per second.
- Approximate size of WAV file for a 4 minute long mucic?
  - At each sample time, 16-bit data (2 Bytes) for each channel (left, right)
  - 44.1 KHz sampling frequency
  - 4 minute = 4 x 60 = 240 seconds
  - File size = 240 (sec) x 44100 x 2 Byte x 2 channel = 42336000 Bytes
  - 4 minute long music will require approximately 42 MB. Quite huge !
- For storing muisc files, the SD card will be used.
  - SD card : 16GB, 32GB, etc → No worry for file size.

Construction Diagram

Overall construction diagram

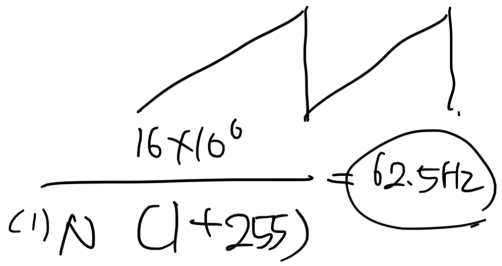


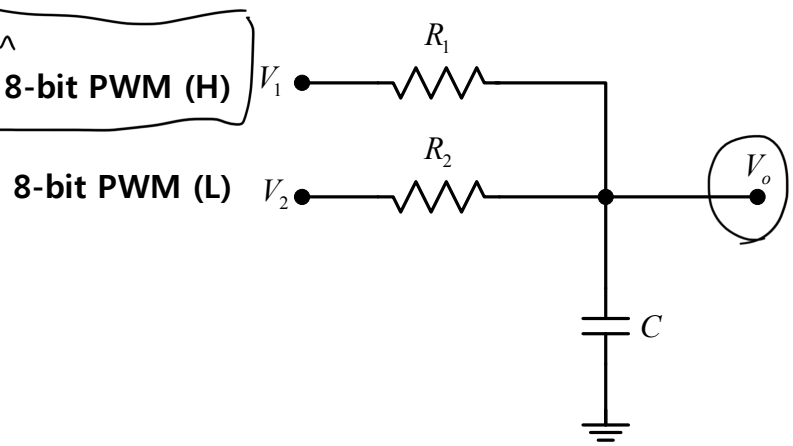
The question is how we can combine two voltage signals, which has different weighting.

Ideas for implementation

- WAV 44.1 KHz : timer interrupt
- 16-bit resolution : Combination of two 8-bit PWMs
- Frequency of the fastest 8-bit PWM : 16*10^6/256 = 62.5 KHz

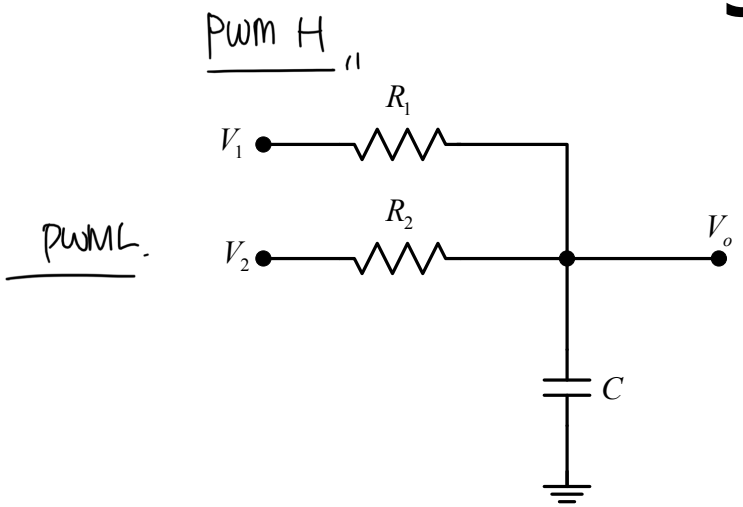
  [Note] 62.5 KHz is faster than 44.1 KHz, which is the sampling rate of the WAV file.





Circuit for the weighted addition of voltage signal with a LPF

Weighted Sum



$$V_o = \frac{R_2 V_1 + R_1 V_2}{R_1 R_2 C s + R_1 + R_2}$$

We can use the superposition rule to get the above result.

At steady state

$$V_o = \frac{R_2 V_1 + R_1 V_2}{R_1 + R_2} = \left(\frac{R_2}{R_1 + R_2}\right) V_1 + \left(\frac{R_1}{R_1 + R_2}\right) V_2$$

$$R_1 = 4.3K,\ R_2 = 1.1M \longrightarrow \frac{R_2}{R_1} = 255.8139... \approx 256 \longrightarrow V_o = \frac{256 \times V_1 + V_2}{257}$$

Resistance tolerance F : $\pm 1\%$

Weighted Sum

- **Weighted sum with LPF : What is fc (Cutoff frequency)**

$$V_o = \frac{R_2 V_1 + R_1 V_2}{R_1 R_2 Cs + R_1 + R_2}$$

$$V_o = \frac{R_2 V_1 + R_1 V_2}{R_1 R_2 Cs + R_1 + R_2} = \frac{\frac{R_2 V_1 + R_1 V_2}{R_2}}{R_1 Cs + \frac{R_1 + R_2}{R_2}} \approx \frac{V_1}{R_1 Cs + 1} = \frac{\frac{V_1}{R_1 C}}{s + \frac{1}{R_1 C}}$$

$$w_c = 2\pi f_c = \frac{1}{R_1 C} \rightarrow f_c = \frac{1}{2\pi R_1 C}$$

With C=66nF, we get fc = 560.8 Hz

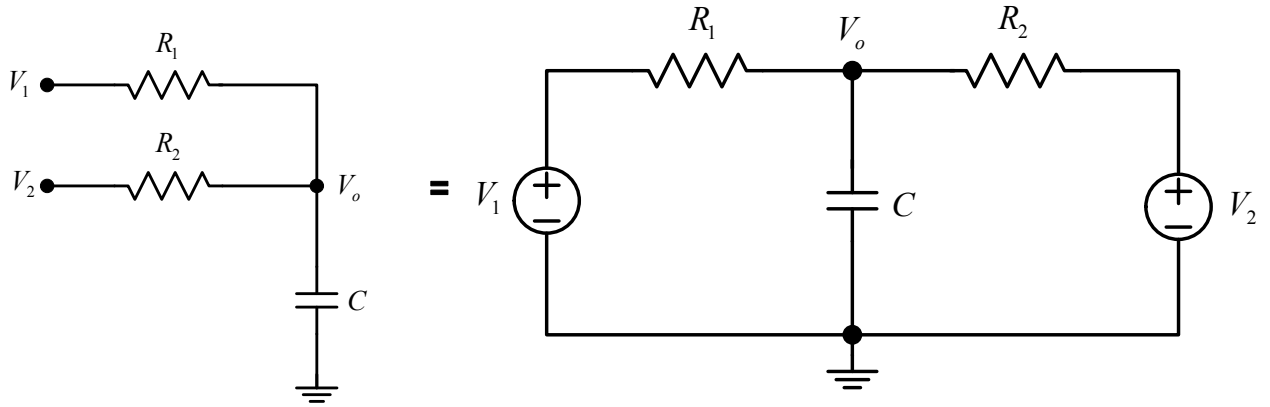
[Note] Why fc=560.8? The frequency of PWM is 62.5 KHz.
We can choose fc such that it is 62.5KHz/100.

Weighted Sum

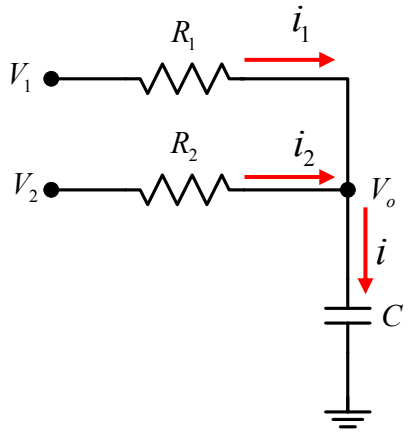
$$V_o = \frac{R_2 V_1 + R_1 V_2}{R_1 R_2 C s + R_1 + R_2}$$

How can we get this?

First method : Superposition rule



Second method



$$i_1 + i_2 = i_3$$

$$i_1 = \frac{V_1 - V_o}{R_1}$$

$$i_2 = ?$$

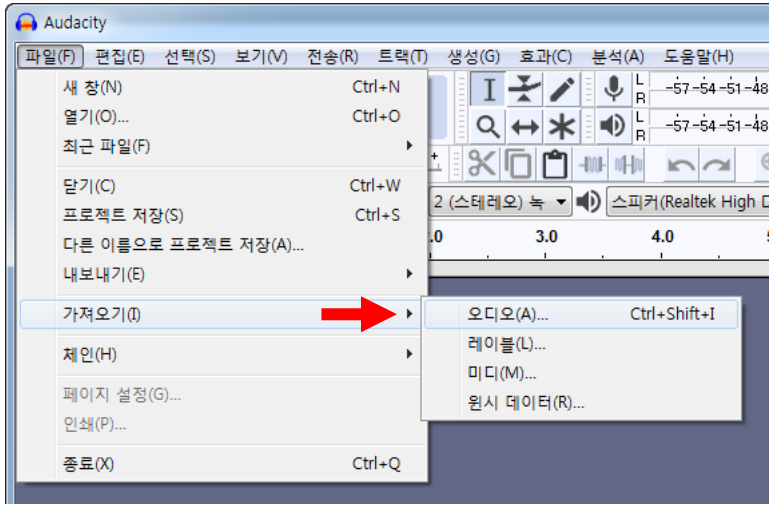
$$i_3 = ?$$

MP3 to WAV Conversion

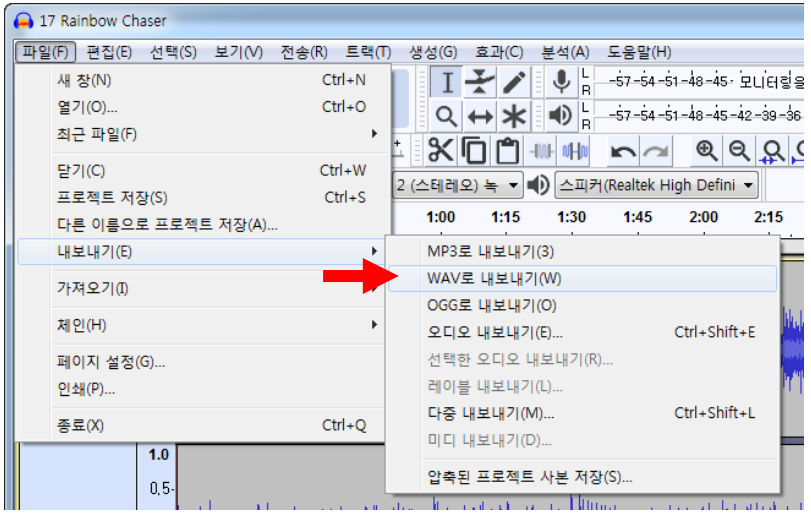
- **Audacity : free and open-source digital audio editor and recording application software**

https://www.audacityteam.org/

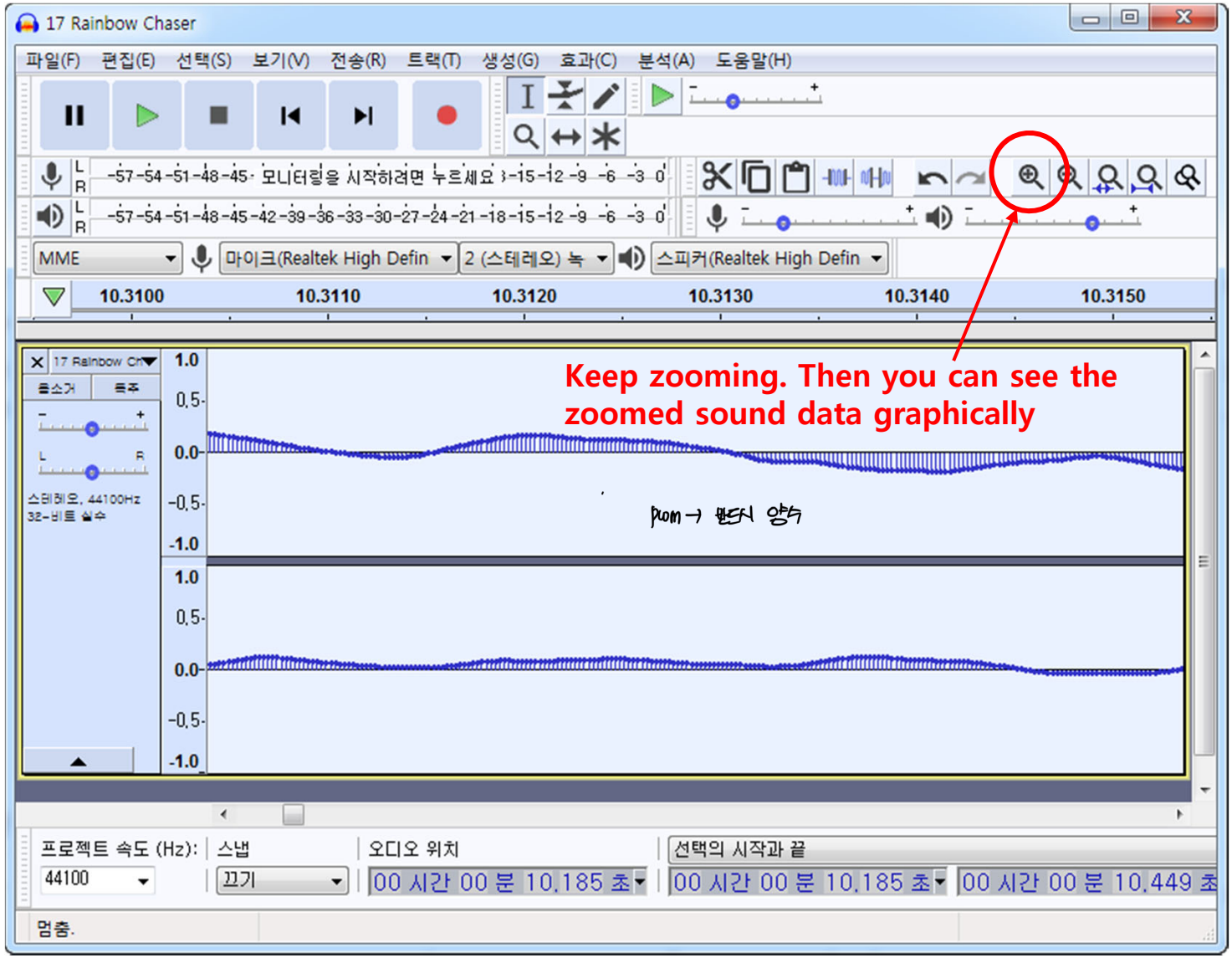
File > Import



File > Export > Choose WAV



Sound Display in Audacity



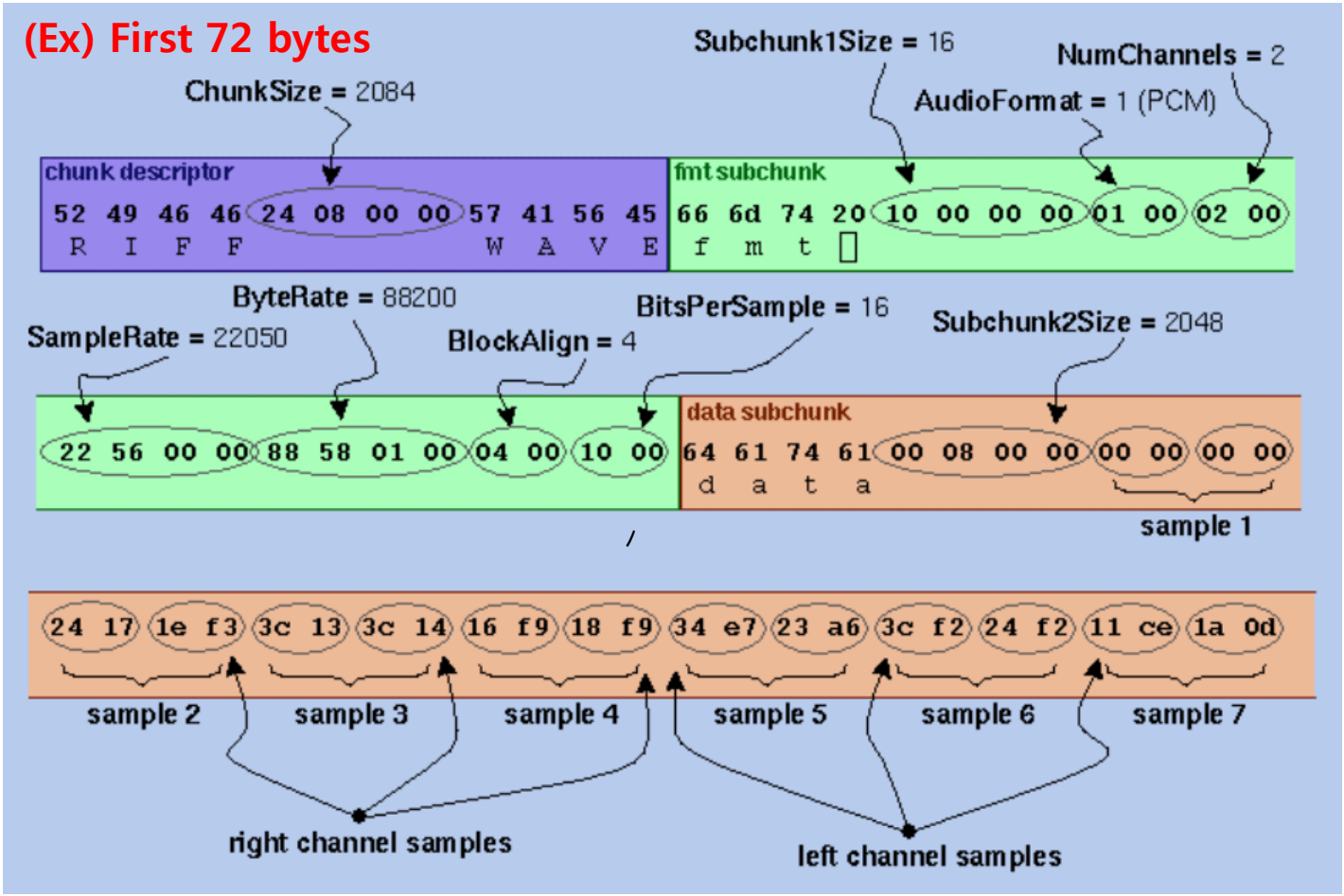
pwm → 이용

Signed 16-bit PCM → Unsigned 16-bit

- WAVE file supports signed 16-bit PCM data (data A)
- However, for PWM to DAC, we need unsigned 16-bit data (data B)
- Conversion : B = A + 0x8000
- If you need unsigned 8-bit data, keep the higher byte then add 0x80 to convert bipolar (i.e., signed) to unipolar (i.e., unsigned).

WAVE File Format

```
52 49 46 46 24 08 00 00 57 41 56 45 66 6d 74 20 10 00 00 00 01 00 02 00
22 56 00 00 88 58 01 00 04 00 10 00 64 61 74 61 00 08 00 00 00 00 00 00
24 17 1e f3 3c 13 3c 14 16 f9 18 f9 34 e7 23 a6 3c f2 24 f2 11 ce 1a 0d
```

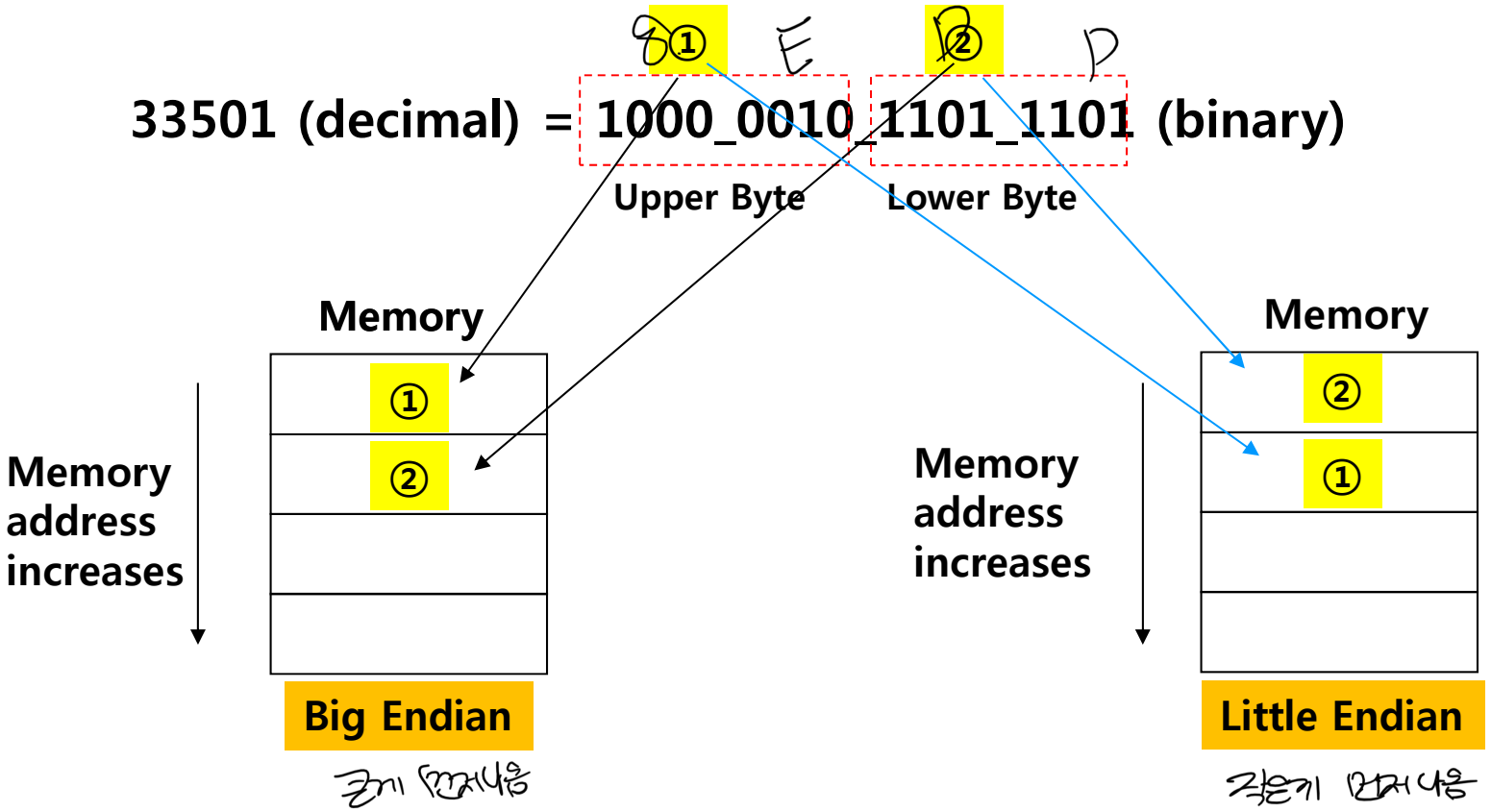


ASCII Table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Little Endian/Big Endian

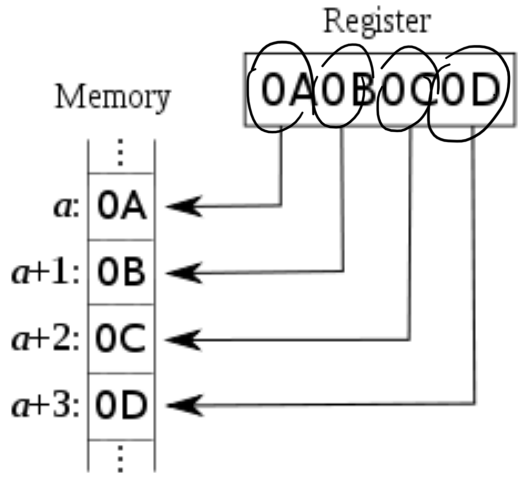
- Endian : sequential order in which bytes are arranged into longer numberical values when stored in memory or when transmitted over digital links.



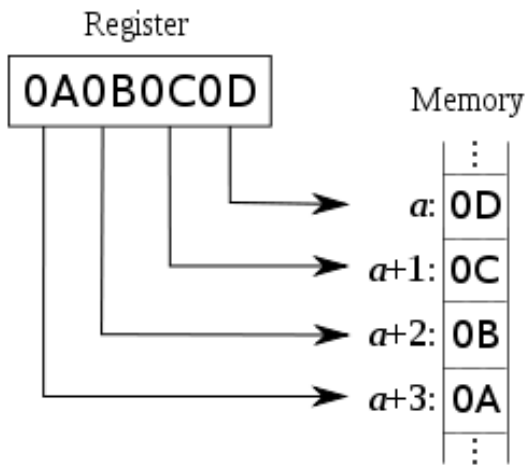
Little Endian/Big Endian

- **Hexadecimal notation : simpler than binary notation**

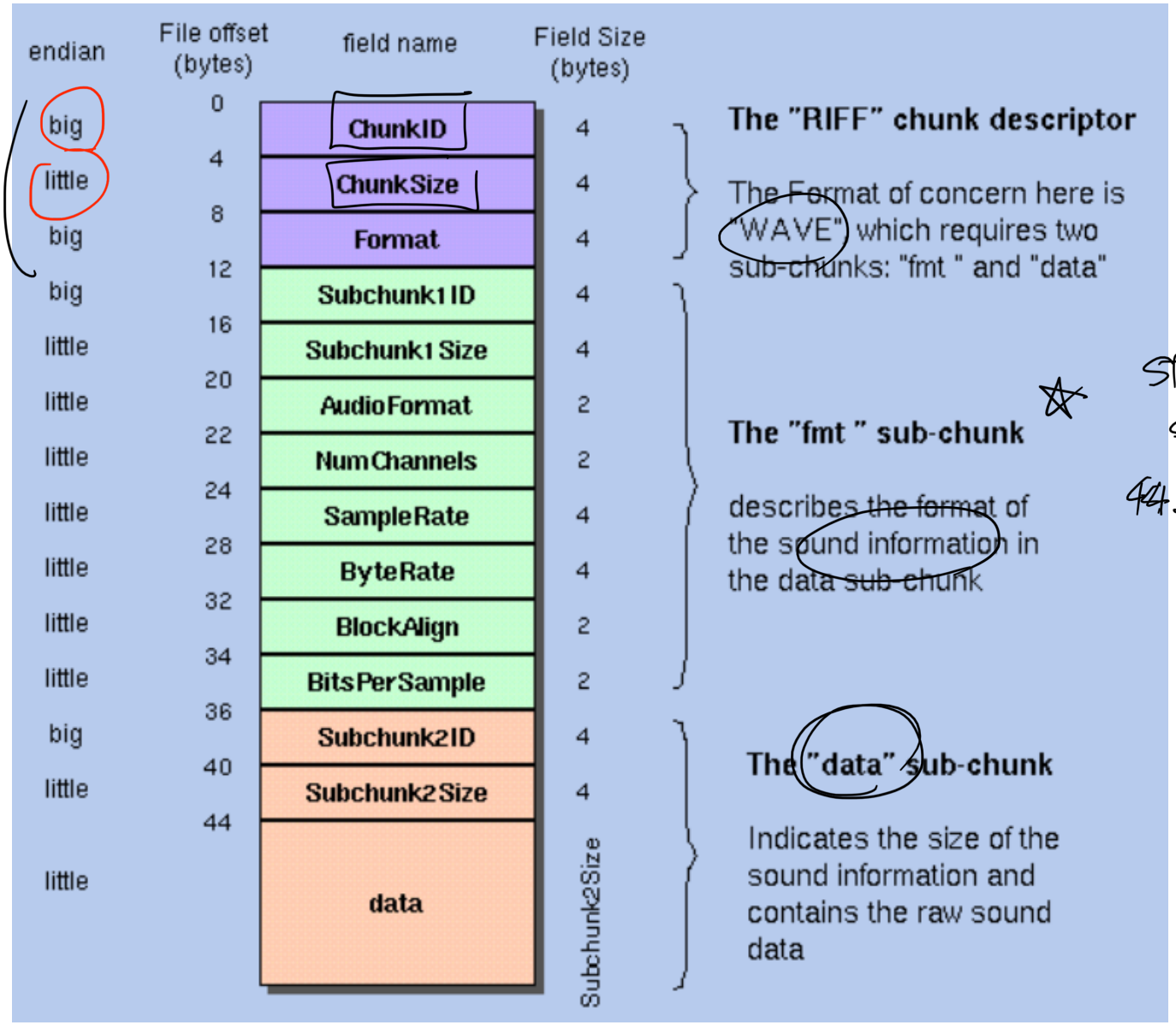
Big Endian



Little Endian



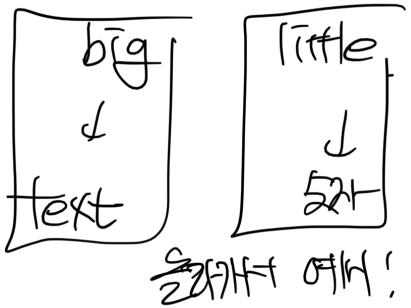
WAVE File Format

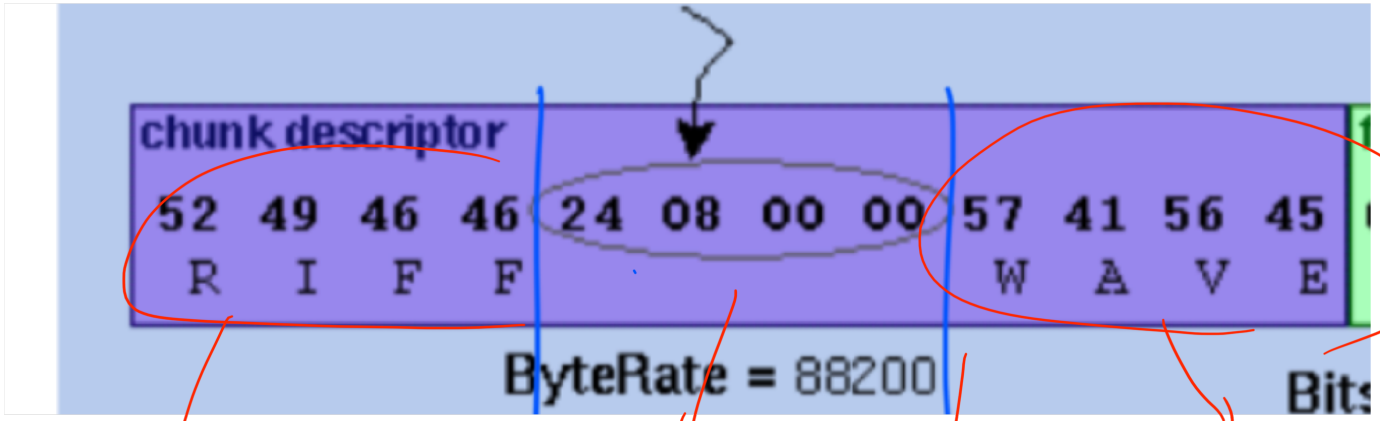


Stereo or Mono ?
8bit or 16bit !
44.1Hz or 기타 ?

WAVE File Format

Offset	Size	Name	Description
The canonical WAVE format starts with the RIFF header:			
0	4	**ChunkID**	Contains the letters "RIFF" in ASCII form (0x52494646 big-endian form).
4	4	**ChunkSize**	36 + SubChunk2Size, or more precisely: 4 + (8 + SubChunk1Size) + (8 + SubChunk2Size) This is the size of the rest of the chunk following this number. This is the size of the entire file in bytes minus 8 bytes for the two fields not included in this count: ChunkID and ChunkSize.
8	4	**Format**	Contains the letters "WAVE" (0x57415645 big-endian form).





RIFF : Big

그대로

Small

↓

Little.

↓

예시

| 00 | 00 | 08 | 24 |

=

DEC = 2084

=

Big

그대로.

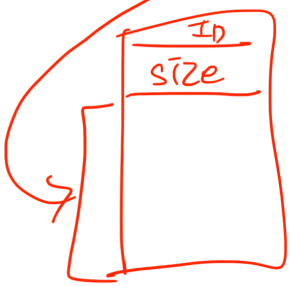
→

보통 2084 사이즈

WAVE File Format

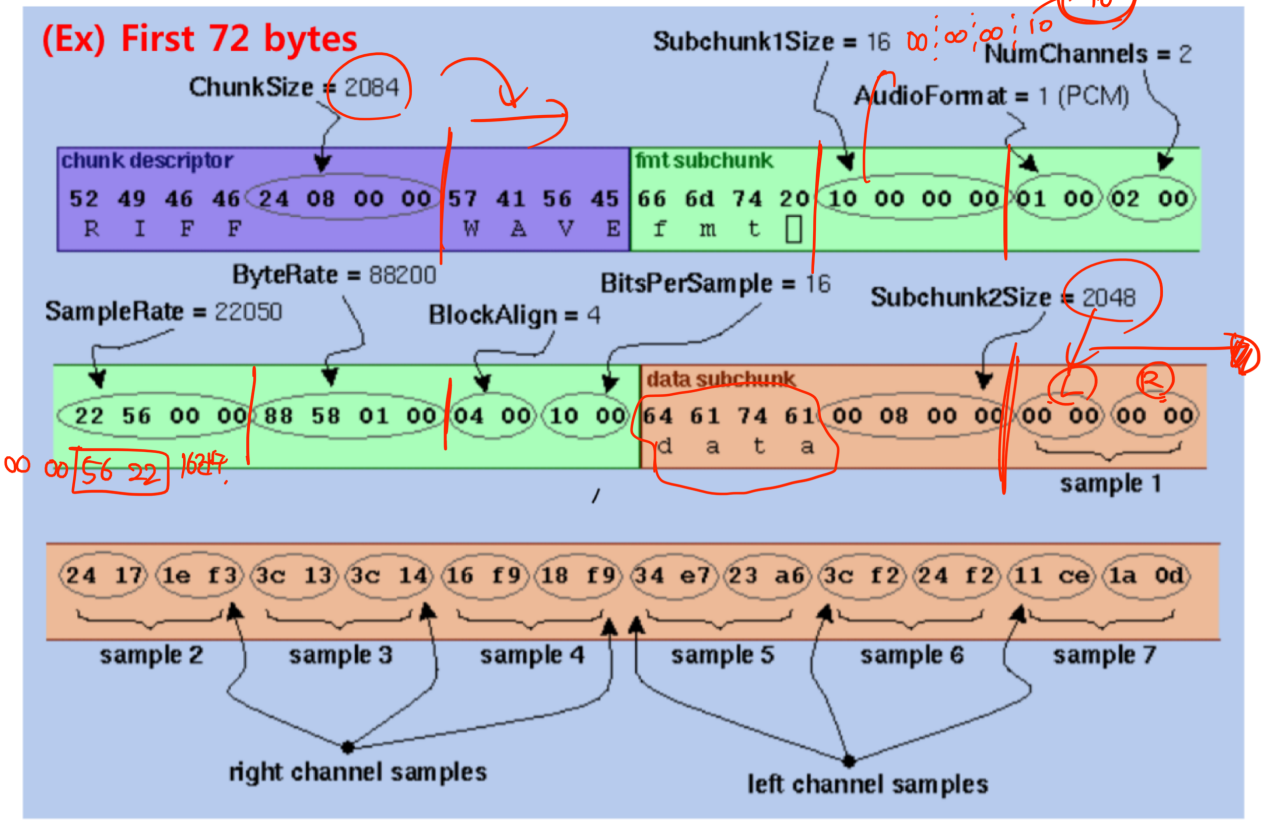
The "WAVE" format consists of two subchunks: "fmt " and "data":
The "fmt " subchunk describes the sound data's format:

12	4	**Subchunk1ID**	Contains the letters "fmt " (0x666d7420 big-endian form).
16	4	**Subchunk1Size**	16 for PCM. This is the size of the rest of the Subchunk which follows this number.
20	2	**AudioFormat**	PCM = 1 (i.e. Linear quantization) Values other than 1 indicate some form of compression.
22	2	**NumChannels**	Mono = 1, Stereo = 2, etc.
24	4	**SampleRate**	8000, 44100, etc.
28	4	**ByteRate**	== SampleRate * NumChannels * BitsPerSample/8
32	2	**BlockAlign**	== NumChannels * BitsPerSample/8 The number of bytes for one sample including all channels. I wonder what happens when this number isn't an integer?
34	2	**BitsPerSample**	8 bits = 8, 16 bits = 16, etc.



WAVE File Format

```
52 49 46 46 24 08 00 00 57 41 56 45 66 6d 74 20 10 00 00 00 01 00 02 00
22 56 00 00 88 58 01 00 04 00 10 00 64 61 74 61 00 08 00 00 00 00 00 00
24 17 1e f3 3c 13 3c 14 16 f9 18 f9 34 e7 23 a6 3c f2 24 f2 11 ce 1a 0d
```



WAVE File Format

The "data" subchunk contains the size of the data and the actual sound:

36	4	**Subchunk2ID**	Contains the letters "data" (0x64617461 big-endian form).
40	4	**Subchunk2Size**	== NumSamples * NumChannels * BitsPerSample/8 This is the number of bytes in the data. You can also think of this as the size of the read of the subchunk following this number.
44	*	**Data**	The actual sound data.

WAVE File Format (Summary)

Endian	Offset	Field Name	Length	Contents
Big	0	ChunkID	4	'RIFF'
Little	4	ChunkSize	4	<File length-8>
Big	8	Format	4	'WAVE'
Big	12	Subchunk1ID	4	'fmt '
Little	16	Subchunk1Size	4	0x00000010 → 0x10000000
Little	20	AudioFormat	4	0x0001 → 0x0100
Little	22	NumChannels	2	<channels>
Little	24	SampleRate	4	<sample rate>
Little	28	ByteRate	4	<bytes/second>
Little	32	BlockAlign	2	Number of bytes for one sample
Little	34	BitsPerSample	2	<bits/sample>
BIG	36	Subchunk2ID	4	'data'
Little	40	Subchunk2Size	4	<length of the data block>
Little	44	data	Subchunk2Size	<sample data>

WAVE File Format

```
52 49 46 46 24 08 00 00 57 41 56 45 66 6d 74 20 10 00 00 00 01 00 02 00
22 56 00 00 88 58 01 00 04 00 10 00 64 61 74 61 00 08 00 00 00 00 00 00
24 17 1e f3 3c 13 3c 14 16 f9 18 f9 34 e7 23 a6 3c f2 24 f2 11 ce 1a 0d
```