Arduino

Tutorial 5
Binary and Memory

2021 by TeakSoon Ding
for STEMKRAF
https://github.com/teaksoon/stemkraf

https://gi

Memory and Numbers

Numbering Systems (Decimal, Binary, Hexadecimal, Octal)

Memory BITS and BYTES

Number Conversion – Decimal / Binary Number Conversion for whole numbers part

C-Language DataType – unsigned whole number (int / short / long)

C-Language DataType – signed whole number (int / short / long)

C-Language DataType – char

Number Conversion – Decimal / Binary Number Conversion for fraction part

IEEE 754 Floating Point Standard

C-Language DataType – float / double

Decimal Numbering System (10 Symbols)

0 1 2 3 4 5 6 7 8 9

This numbering system is used in our daily activities.

Binary Numbering System (2 Symbols)

0 1

This numbering system is used in micro-controllers Memory

Hexadecimal Numbering System (16 Symbols)

0 1 2 3 4 5 6 7 8 9 A B C D E F

This numbering system is used to simplify big binary numbers, to reduce the number of digits.

Octal Numbering System (8 Symbols) – rarely used

0 1 2 3 4 5 6 7 8

This numbering system were used in older Computer Systems to simplify big binary numbers where multiplies of 3-bits were commonly used.

The following represents the same number but in different numbering system. Hex, Decimal, Octal and Binary

```
FF       ( 2 digits in Hex )
255      ( 3 digits in Decimal )
377      ( 3 digits in Octal )
11111111 ( 8 digits in Binary )
```

By looking a the number above we can guess the number FF is from the Hex numbering system. However, for number 255, 377 and 11111111 we cannot tell which numbering system they are using. In C-Language programming, we use a Prefix to indicate the numbering system used by a number

```
0xFF        "0x" as Prefix for Hex
255         No Prefix for Decimal
0o377       "0o" as Prefix for Octal ( 0 is Zero, o is letter 'o')
0b11111111  "0b" as Prefix for Binary
```

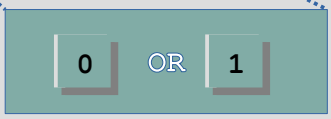
Regardless of what numbering system we use, the number will be stored as binary in the micro-controllers memory.

```
0xFF       will be stored as 11111111 in the micro-controllers Memory
255        will be stored as 11111111 in the micro-controllers Memory
0o377      will be stored as 11111111 in the micro-controllers Memory
0b11111111 will be stored as 11111111 in the micro-controllers Memory
```

Micro-controllers Memory is consist of many individual physical memory device arranged in a logical sequence.

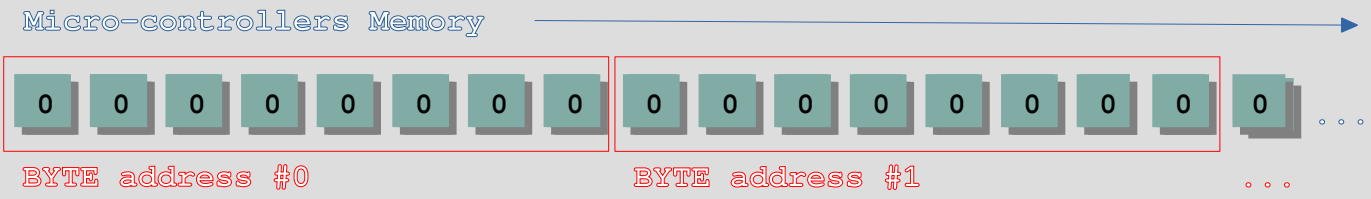
Each physical memory device can have TWO(2) different states where only one state can be active at a time. The current active state in each memory device is represented by either 0 or 1.

Each memory device must be represented by either 0 or 1. This single memory device is known as a **BIT**



The C-Language does not allow us to access to a single BIT from the micro-controller Memory. We can only access/address micro-controllers Working Memory in the smallest **groups of 8-BIT**, also known as a **BYTE**.

ONE(1) BYTE = EIGHT(8) BIT



Convert to Decimal from Binary

```
#REFERENCE Convert to Decimal from Decimal (multiplies of 10)
Decimal,( multiplies of 10 ) ...,1000000,100000,10000,1000,100,10,1

Original Number = 111 in Decimal, to be converted to Decimal
111 position 1, multiply it with 1   (1*1)   = 1   (in Decimal)
111 position 2, multiply it with 10  (1*10)  = 10  (in Decimal)
111 position 3, multiply it with 100 (1*100) = 100 (in Decimal)
Add all the results,( 1 + 10 + 100 ) = 111

111 in Decimal, converted to Decimal is 111
```

```
Convert to Decimal from Binary (based on #REFERENCE, but multiplies of 2)
Binary,( multiplies of 2 ) ...,64,32,16,8,4,2,1

Original number = 111 in Binary, to be converted to Decimal
111 position 1, multiply it with 1 (1*1) = 1 (in Decimal)
111 position 2, multiply it with 2 (1*2) = 2 (in Decimal)
111 position 3, multiply it with 4 (1*4) = 4 (in Decimal)
Add all the results, ( 1 + 2 + 4 ) = 7

111 in Binary, converted to Decimal is 7
```

Convert to Binary from Decimal

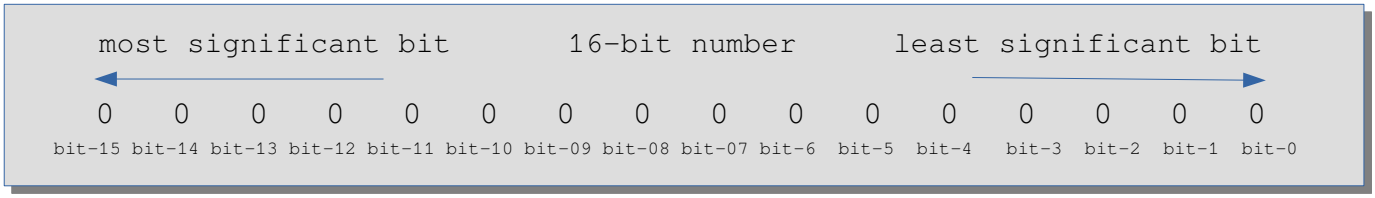
```
#REFERENCE Convert to Decimal from Decimal (divide by 10)

Original number = 342 in Decimal, to be converted to Decimal
342 / 10 = 34,remainder=2 (whole part for next row),partial value is 2
34  / 10 =  3,remainder=4 (whole part for next row),partial value is 42
3   / 10 =  0,remainder=3 (whole part for next row),partial value is 342
whole part=0 DONE, final value = 342

342 in Decimal, converted to Decimal is 342
```

```
Convert to Decimal from Decimal (based on #REFERENCE, but divide by 2)

Original number = 123 in Decimal, to be converted to Binary
123 / 2 = 61,remainder=1,(whole part for next row),partial value=1
61  / 2 = 30,remainder=1,(whole part for next row),partial value=11
30  / 2 = 15,remainder=0,(whole part for next row),partial value=011
15  / 2 =  7,remainder=1,(whole part for next row),partial value=1011
7   / 2 =  3,remainder=1,(whole part for next row),partial value=11011
3   / 2 =  1,remainder=1,(whole part for next row),partial value=111011
1   / 2 =  0,remainder=1,(whole part for next row),partial value=1111011
whole part=0 DONE, final value = 1111011

123 in Decimal, converted to Binary is 1111011
```

```
Manual conversion can be very time consuming.

There are many online nummber conversion tools available on the internet to
help make this process simple. Link below is one of them.
https://www.rapidtables.com/convert/number/decimal-to-binary.html
```

unsigned numbers are positive only whole numbers (no fractions)

Convert the number into binary and store the binary directly into the Memory.
This example is for "unsigned int" DataType storage (16-bit Memory Block)



unsigned int number 16 Bits **(using direct binary conversion)**

```
Binary 0000000000000000 is 0
Binary 1111111111111111 is 655352
```

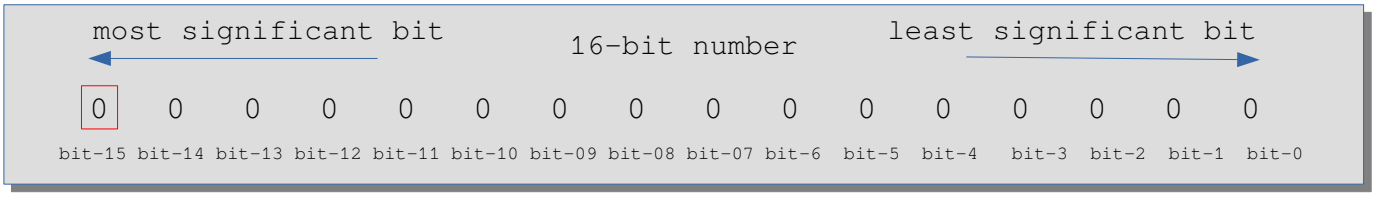
```
// program name = datatype unsigned number
void setup() {
unsigned char  myVar0 = 12;
unsigned int   myVar1 = 12;
unsigned short myVar2 = 12;
unsigned long  myVar3 = 12;

  Serial.begin(9600);

  Serial.print("\n");
  Serial.print("\nunsigned char = 12, bits=");
  Serial.print(sizeof(myVar0)*8);
  Serial.print("\nSRAM Memory = ");
  for (int i=8; i>=0; i--) {
    Serial.print((myVar0 >> i) & 0x1);Serial.print(" ");
  }
  Serial.print("\n");
  Serial.print("\nunsigned int = 12, bits=");
  Serial.print(sizeof(myVar1)*8);
  Serial.print("\nSRAM Memory = ");
  for (int i=15; i>=0; i--) {
    Serial.print((myVar1 >> i) & 0x1);Serial.print(" ");
  }
  Serial.print("\n");
  Serial.print("\nunsigned short = 12, bits=");
  Serial.print(sizeof(myVar2)*8);
  Serial.print("\nSRAM Memory = ");
  for (int i=15; i>=0; i--) {
    Serial.print((myVar2 >> i) & 0x1);Serial.print(" ");
  }
  Serial.print("\n");
  Serial.print("\nlong Decimal = 12, bits=");
  Serial.print(sizeof(myVar3)*8);
  Serial.print("\nSRAM Memory = ");
  for (int i=32; i>=0; i--) {
    Serial.print((myVar3 >> i) & 0x1);Serial.print(" ");
  }
}
void loop(){}
```

Use the Serial Monitor to see output

We take out raw bits from the SRAM Memory and display them on the screen.
char, int, short and long works the same (the difference is in size).
char = 1-byte
int   = 2-bytes or 16-bits (sometimes is 4-bytes or 32-bits for some systems)
short = 2-bytes or 16-bits
long = 4-bytes or 32-bits

signed numbers are positive and negative whole numbers (no fractions)

We use one bit from our Block of Memory to represent the plus (+) or minus(-) symbol. This example is for "signed int" DataType storage (16-bit Memory Block)



Bit-15 = 0, Positive Number (0 represent + sign)
We work on the remaining 15-bits (using direct binary conversion)

Binary 000000000000000 is 0, +sign, 0000000000000000 is +0
Binary 111111111111111 is 32767, +sign, 0111111111111111 is +32767

Bit-15 = 1, Negative Number (1 represent – sign)
We work on the remaining 15-bits (using direct binary conversion)

binary 000000000000000 is 0, -sign, 1000000000000000 is -0
binary 111111111111111 is 32767, -sign, 1111111111111111 is -32767
We have a problem, 0 is represented by 2 binary numbers. **Previously 0 was represented by 0000000000000000, now 0 is represented by 1000000000000000**

SOLUTION = Two's Complement for negative whole number

Two's Complement = Invert all the bits ("0" to "1" or "1" to "0") and then plus 1 to the result

Bit-15 = 1, Negative Number(1 represent – sign)
We work on the remaining 15-bits (with two's complement conversion)

000000000000000 = 111111111111111 is 32767(+1)=32768, -sign, it is -32768
111111111111111 = 000000000000000 is 0(+1) =1, -sign, it is -1
Problem Solved, NO MORE 0 **represented by 2 binary numbers**

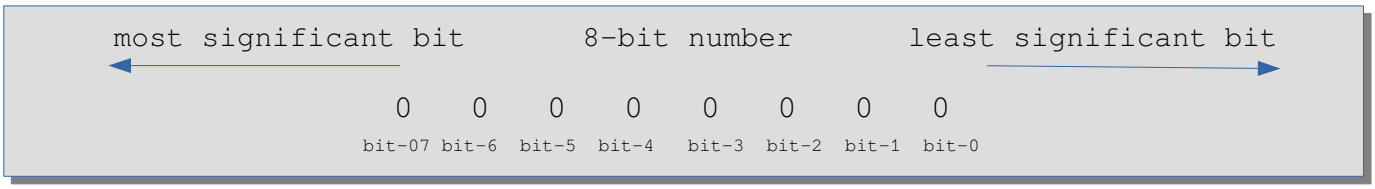
```
// program name = datatype signed number
void setup() {
signed int myVar1 = 1;
signed int myVar2 = -1;
  Serial.begin(9600);

  Serial.print("\n\nint Decimal = 1    int Size =");
  Serial.print(sizeof(myVar1)*8);
  Serial.print("\nSRAM Memory = ");
  for (int i=15; i>=0; i--) {
    Serial.print((myVar1 >> i) & 0x1);Serial.print(" ");
  }

  Serial.print("\n\nint Decimal = -1  int Size =");
  Serial.print(sizeof(myVar2)*8);
  Serial.print("\nSRAM Memory = ");
  for (int i=15; i>=0; i--) {
    Serial.print((myVar2 >> i) & 0x1);Serial.print(" ");
  }
}
void loop(){}
```

Use the Serial Monitor to see output

We take each raw bit from SRAM Memory and display them on the Serial Monitor int, short and long works the same (difference is in size). The keyword "signed" is optional ("signed" is the default, we can leave it empty)

char is same like int/short/long, we also do conversion to binary for Memory storage, except that char is 8-bit and it has an ASCII Table representation

Convert the number into binary and store the binary directly into the Memory.
This example is for "char" DataType storage (8-bit Memory Block)



ASCII Chart (character codes 0 - 127)

000	(nul)	016 ►	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001 ☺	(soh)	017 ◄	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002 ☻	(stx)	018 ↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003 ♥	(etx)	019 ‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004 ♦	(eot)	020 ¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005 ♣	(enq)	021 §	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006 ♠	(ack)	022 ▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007 •	(bel)	023 ↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008 ◘	(bs)	024 ↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009	(tab)	025 ↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010	(lf)	026	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011 ♂	(vt)	027 ←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012 ♀	(np)	028 ∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013	(cr)	029 ↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014 ♫	(so)	030 ▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015 ☼	(si)	031 ▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

Use the Arduino IDE Software
1.Save as "datatype char"
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = datatype char
void setup() {
char myChar1 = 'A';
char myChar2 = 65;

  Serial.begin(9600);

  Serial.print("\n\nmyChar1 = 'A'");
  Serial.print("\nSRAM Memory = ");
  for (int i=8; i>=0; i--) {
    Serial.print((myChar1 >> i) & 0x1);Serial.print(" ");
  }
  Serial.print("\n\nmyChar2 = 65");
  Serial.print("\nSRAM Memory = ");
  for (int i=8; i>=0; i--) {
    Serial.print((myChar2 >> i) & 0x1);Serial.print(" ");
  }

}
void loop(){}
```

Use the Serial Monitor to see output

We take out raw bits from the SRAM Memory Block and display them on the screen

In addition to assigning a number to a char datatype, for example char myChar2 = 65; we can also assign a char DatatType symbols from the ASCII Chart. for example char myChar1 = 'A';

Convert to Decimal from Binary (fraction part)

```
#REFERENCE Convert to Decimal from Decimal (multiplies of 10)
Decimal,( multiplies of 10 ) ...,1000000,100000,10000,1000,100,10,1

Original Number = 0.111 in Decimal, to be converted to Decimal
0.111 position 1, divide it with 10   (1/10),   = 0.1 (in base-10 number)
0.111 position 2, divide it with 100  (1/100),  = 0.01 (in base-10 number)
0.111 position 3, divide it with 1000 (1/1000), = 0.001 (in base-10 number)
Add all the results,( 0.1 + 0.01 + 0.001 ) = 0.111
0.111 in Decimal, converted to Decimal is 0.111
```

```
Convert to Decimal from Binary (based on #REFERENCE, but multiplies of 2)
Binary,( multiplies of 2 ) ...,64,32,16,8,4,2,1

Original number = 0.111 in Binary, to be converted to Decimal
0.111 position 1, divide it with 2 ( 1/2 ) = 0.5 (in base-10 number)
0.111 position 2, divide it with 4 ( 1/4 ) = 0.25 (in base-10 number)
0.111 position 3, divide it with 8 ( 1/8 ) = 0.125 (in base-10 number)
Add all the results, ( 0.5 + 0.25 + 0.125 ) = 0.875
0.111 in Binary, converted to Decimal is 0.875
```

Convert to Binary from Decimal (fraction part)

```
#REFERENCE Convert to Decimal from Decimal (multiplies of 10)

Original number = 0.342 in Decimal, to be converted to Decimal
0.342 x 10 = 3.42 (whole=3,frac part for next row),partial value is 0.3
0.42 x 10  = 4.2  (whole=4,frac part for next row),partial value is 0.34
0.2 x 10   = 2.0  (whole=2,frac part for next row),partial value is 0.342
frac part=0 DONE, final value 0.342

0.342 in Decimal, converted to Decimal is 0.342
```

```
Convert to Binary from Decimal (based on #REFERENCE, but multiples of 2)

Case 1: ( we get lucky )
Original number = 0.25 in Decimal, to be converted to Binary
0.25 x 2 = 0.5 (whole=0,fraction for next row),partial value 0.0
0.5  x 2 = 1.0 (whole=1,fraction for next row),partial value 0.01
frac part=0 DONE, final value is 0.01

0.25 in Decimal, converted to Binary is 0.01

Case 2:( not so lucky ) *** THIS IS THE TROUBLESOME PART ***
Original number = 0.2 in base-10, to be converted to Binary
0.2 x 2 = 0.4 (whole=0,frac part for next row),partial value=0.0
0.4 x 2 = 0.8 (whole=0,frac part for next row),partial value=0.00
0.8 x 2 = 1.6 (whole=1,frac part for next row),partial value=0.001
0.6 x 2 = 1.2 (whole=1,frac part for next row),partial value=0.0011
0.2 x 2 = 0.4 (whole=0,frac part for next row),partial value=0.00110
0.4 x 2 = 0.8 (whole=0,frac part for next row),partial value=0.001100
0.8 x 2 = 1.6 (whole=1,frac part for next row),partial value=0.0011001
... and so on
we can't get frac part=0, we just stop here or because it can go on forever.
We just "accept" the partial value 0.00110011 (more digits, more accurate)

0.2 in Decimal, converted to base-2 is 0.0011001
```

```
Manual conversion can be very time consuming.

There are many online nummber conversion tools available on the internet to
help make this process simple. Link below is one of them.
https://www.rapidtables.com/convert/number/decimal-to-binary.html
```

Floating point numbers

Floating point numbers are numbers with two parts, whole and fraction, separated by a dot “.” symbol can be both positive and negative. If we have a number 123.456 (whole part=123, fraction part=456). We cannot do a direct binary storage because it has the dot (.) symbol as well as the plus (+) or minus (-) symbol used in float number. The float uses a special format for storage, the IEEE 754 Floating Point Standard.

Step 1: Convert our number into Scientific Notation format

What is Scientific Notation format ? This is how we convert a regular number into Scientific Notation number.

Move the dot . to the position after the first non-zero digit.

Example:Original number = 12345.0 (move dot 4 position to the left)
Scientific Notation will be 1.2345 e +4 (exponent is +4)

Example:Original number = 0.012345 (move dot 2 position to the right)
Scientific Notation will be 1.2345 e -2 (Exponent is -2)

Example:Original number = 1.2345 (no need to move dot, already in position)
Scientific Notation will be 1.2345 e +0,(Exponent is +0)

We get 3 parts,
1. a single digit number before the dot
2. a fraction part after the dot
3. an Exponent part

Step 2: Convert the Exponent part in decimal into binary.

Although the Exponent has the + and - sign, they are just movement direction of the dot . to the left or to the right in the float number in the Scientific Notation.

We use the “bais method” to represent them for our conversion of Exponent number to binary. We use **127** as bais point (mid-point of an 8-bit number).

Since the + and - symbol in Exponent are simply the movement of the dot, to the left or to the right. This is how we do conversion for the Exponent Value and storing them in 8-bits Memory

if Exponent is -2, we take 127-2 = 125, 125 is 01111101 in binary,
Binary representation for e -2 = 01111101

if Exponent is -1, we take 127-1 = 126, 126 is 01111110 in binary,
Binary representation for e -1 = 01111110

if Exponent is +1, we take 127+1 = 128, 128 is 10000000 in binary,
Binary representation for e + = 10000000

if Exponent is +2, we take 127+2 = 129, 129 is 10000001 in binary,
Binary representation fo e +2 = 10000001

Step 3: Preparing the IEEE 754 data for Memory storage

We will be using 2 parts from our Scientific Notation Binary Number to for our IEEE 754 Floating Point Standard.

1.Mantissa - From Scientific Notation (digits after the dot .)
2.Exponent - From Scientific Notation (Binary representation of the exponent based on 127 bais point)

```
IEEE 754 Floating Point Standard for Memory Storage and Retrieval

4-bytes ( 32 bits ). The 32-bits are broken into 3 parts
32-bits from left to right
S = 1-bit ( Sign Bit, 0 or 1 )
E = Next 8-bit ( Exponent Value )
M = Next 23-bit ( Mantissa Value )
SEEEEEEEEMMMMMMMMMMMMMMMMMMMMMMM
```

***S-Sign Bit (Stores either 0 or 1)**
0 = Posistive Number
1 = Negative Number

***E-Exponent (Exponent in binary)**
Exponent value comes from our number in Scientific Notation format. The Exponent Value needs to be converted into binary using the bais method, 127 as mid point. It will be updated into the 8-bits slot

***M-Mantissa (all the digits after the dot . From a Scientific Notation)**
We get this when we convert our regular binary float number into Scientific Notation binary number. First digit will surely be 1 before the dot (because it is in binary, we do not need to store this in our memory). What we need is all those “1”s and” “0”s after the dot . , the Mantissa and store all of them into the maximum 23-bits slot.

```
IEEE 754 Float Point Standard for Memory Storage is ready

1.Fill 1-bit from Left with Sign Bit
2.Next 8-bits with our Exponent in binary
3.Next 23-bits with our Mantissa in binary

We are done !!!
This was for storing the float number into Memory.
```

```
For retrieval, we work backwards.

1.Construct the Scientific Notation number back from Memory Storage ( still in
binary form )
2.Convert back to decimal from binary ( still in Scientific Notation )
3.Convert Scientific Notation number back to our regular number by getting rid
of the exponent part ( just move the dot . accordingly )
```

double DataType storage is exactly same like the float except that it is larger+higher precision, in most computers double has 8-bytes(64-bits) vs float 4-bytes(32-bits).

double DataType storage in the IEEE 754 Floating Point Standard
Sign Bit = 1-bit
Exponent = 11-bits
Mantissa = 52-bits

NOTE: For Atmega328 micro-controller, both double and float are exactly the same, 32-bits

Floating point numbers are positive and negative numbers with decimal point fraction.

The float uses the **IEEE 754 Floating Point Standard** (also in binary) for Memory storage and retrieval.

This example is for "float" DataType storage (32-bit Memory Block)

Use the Arduino IDE Software
1.Save as "datatype float"
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// Program Name = datatype float
void setup (void) {
float myVar;
union {
  float f;
  uint32_t u;
} memory;

  myVar = 0.078125; // this is the orignal float number in decimal
  memory.f = myVar;

  // 0.078125 = 0.000101 in binary
  // Scientific notation for 0.000101 = 1.01 e-4
  //
  // Sign Bit (1-bit)  = 0
  //
  // Exponent (8-bits) = 01111011
  // e-4, Using 127 baised, 127-4 = 125 ( 01111011 in binary )
  //
  // Mantissa (23-bits) = from scientific number 1.01 e-4, Mantissa = 01
  //
  // Based on the parts, Sign Bit(1), Exponent(8) and Mantissa(23)
  // 0.078125 will be 00111101101000000000000000000000 IEEE 754 standard

  Serial.begin(9600);
  Serial.print("\n\nOriginal float value in decimal = ");
  Serial.print(myVar,6);

  Serial.print("\nThe raw 32-bits stored in SRAM Memory,myVar = ");
  for (int i=31; i>=0; i--) {
    Serial.print((memory.u >> i) & 0x1);Serial.print(" ");
  }
}
void loop (void) {}
```

Use the Serial Monitor to see output

We take out raw bits from the 32-bit Memory Block and display them on the screen