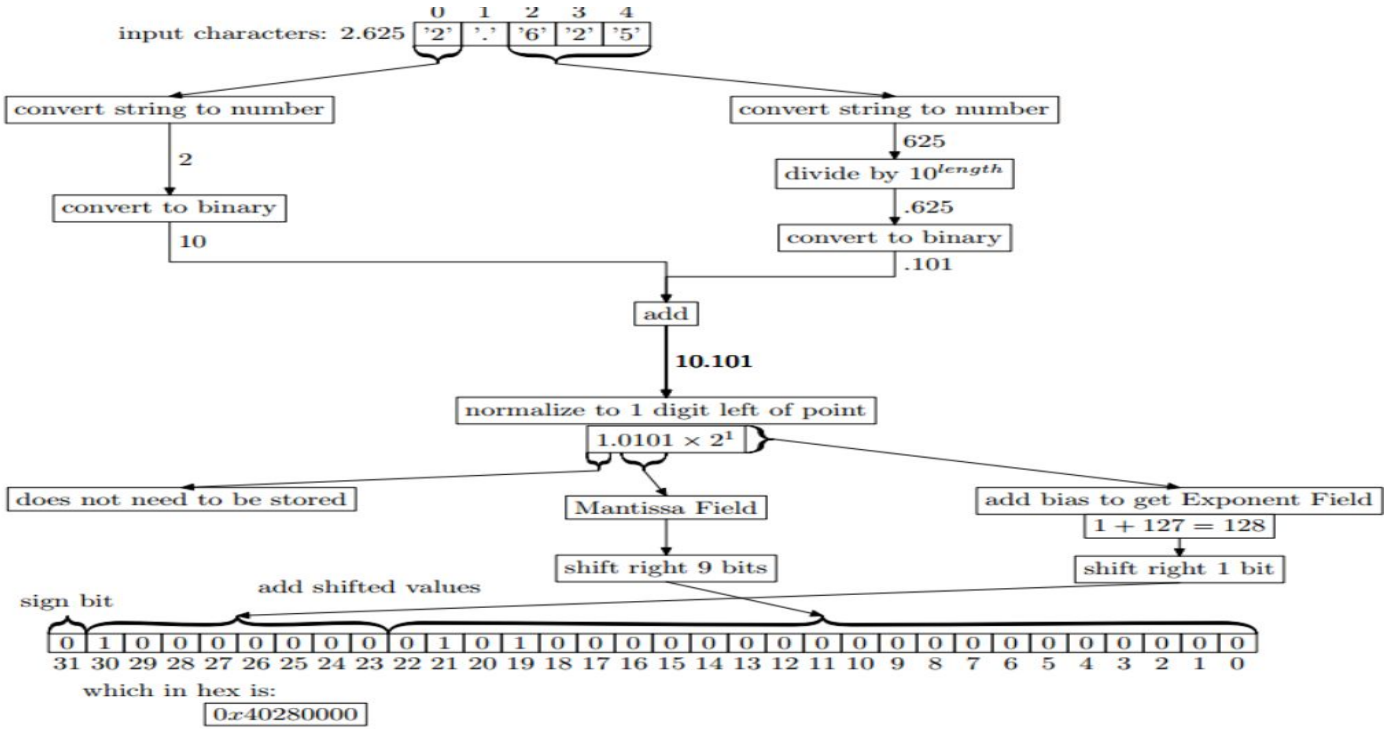
Walking through the number conversion process

Pratik Singh

Specification

This lab is an intermediate iteration to break down our c code into small files, to make each part of the float to hex translation more readable, and modular for our final goal to translate into asm. As we go through, I will be highlighting the portion of the diagram that each function is doing (hopefully in order)



Lab.h-the pieces of the puzzle

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
const int MAX = 10 ;
```

These are only a directory of c files. Binary.s and Inputnumbers.s are also called.

```
void PrintHexvalue( unsigned ) ;
void getNumber( char * ) ;
```

```
int getwhole( char* , char*  ) ;          //return length of the whole
int getfraction char* , char*  , int ) ; //return length of fractional part
```

```
int stringto2int( char* , int ) ;
unsigned convert( float ) ;
```

Overview of implementation

```
#include "lab.h"
int main(){
        const int MAX = 10;
        char s[MAX]; // = "0.0"

        char inputarray[MAX] ;
        inputFloat(inputarray) ;
        char w[MAX] ; //array of max value (to be passed)
        char f[MAX] ;

        int lenw = getwhole( w , s ) ;
        int lenf = getfraction( f , s , lenw ) ;

        int wp = casteString2Int( w , lenw ) ;
        int fn = casteString2Int( f , lenf ) ;

        float fp = fn/pow( 10 , lenf ) ;
        float nf = wp + fp ;
        unsigned num = convertUnsigned( nf ) ;
        PrintHexvalue( num ) ;
}
```

stringtoint.c

#include <math.h>

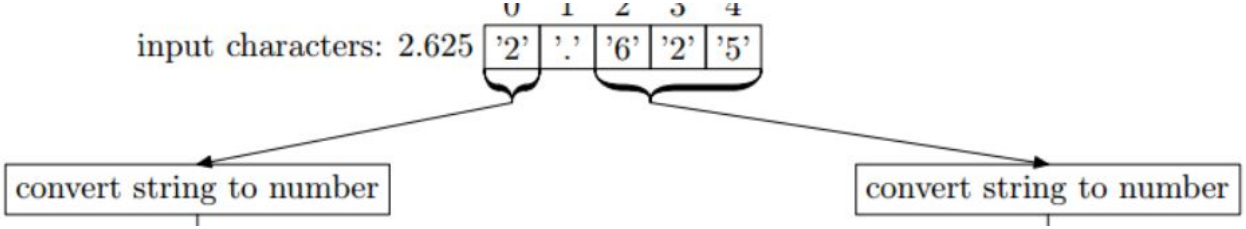
```
#include <stdio.h>
int strintoint( char * w , int lenwhole ){
        int wn = 0 ;
        int i ;
        for( i = 0 ; i < lenwhole ; ++i ){

                wn += ( w[i] - '0' )*pow(10 , lenwhole-i-1) ;
        }


        printf( "Number (int wn): %i\n" , wn ) ;

        return wn ;


}
```

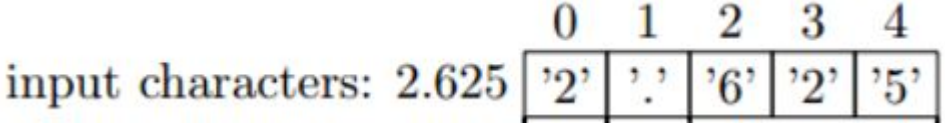


To convert from character type (array) to int, we subtract for each value in s “0”.

Inputfloat.s

We use a system call along ($0x03) and we store into the global variable length which is a length of the float value entered.

So, float value 2.625 is collected now.




```
.data
.globl len
len: .int 0

.text
.globl getNumber
getNumber:
    push %ebp
    mov %esp, %ebp
    mov 8(%ebp), %eax

    mov %eax, %ecx
    mov $9, %edx
    mov $0x03, %eax
    mov $0, %ebx

    int $0x80
    mov %eax, %ebx
    dec %ebx
    mov %ebx, len
    mov 8(%ebp), %eax
    movb $0, (%eax,%ebx)

    mov %ebp, %esp
    pop %ebp
    ret
```

Getfraction.c & Getwhole.c

```
#include <stdio.h>


int getfraction( char* f , char* s , int i ){
        int j = 0 ; i++ ;
        while( s[i] ){
                f[j] = s[i] ;
                i++ ; j++ ;
        }
        int lenf = j ;

        printf( "Length of decimal number: %i\n" , j ) ;
        f[ j ] = '\0' ;
        printf("Decimal number: %s\n",f);
        return lenf ;
}
```

Both functions have counters running as they append numbers to an array.
Then, we print the length, (which is the counter) and is returned.

Notice we have to increment i to skip over the decimal in getfraction.

```
#include <stdio.h>
int getwhole( char* w , char* s ){
        int i = 0 ;
        while( i < 10 - 1 &&  s[i] != '.'){
                w[i] = s[i] ;
                i++ ;
        }

        int lenw = i ;
        printf("Length of whole number: %i\n" , lenw ) ;
        w[i] = '\0';
        printf("Whole number: %s\n",w);
        return lenw ;
}
```

binary.c


```
binary(wn) ;
char binar [20];
i=7 ;
while (v[i] == 0 ) i-- ;
int whole [8] ;
{ whole[j] = v[i] ; j++ ;}
whole[j] = '\0' ; // binar[j] = '.' ; j++;
int stop = j ; //- 1 ;
```

Observe, that we have the binary digts seperate before we co-late them together into floating ieee.

Here, we are taking the fractional And whole parts and using powers Of 2 to convert into 2.

```
for(i=0; i<j; i++)

binary[i] =  whole[i] ;
```

```
printf("Whole Binary= ") ;
for (i=0; i< stop && (whole[i] == 0 || whole[i] == 1) ; i++)
printf("%i", whole[i]);   printf("\n") ;
int fract[10] ; //storing the binary conversion of fraction int digit;
float part=fp;
for (i=0; i <15 && part!=0; i++)
{
digit = part * 2 ;
part = part * 2;
if (part > 1) part = part-1 ;
fract[i]= binary[j+i]  = digit ;
} i++; fract[i] = binary[j+i] = '\0' ;
printf("Fraction Binary= ") ;
for (i=0; i<10 && (fract[i] == 0 || fract[i] == 1) ; i++) printf("%i",
fract[i]);  printf("\n") ;
printf("Binary= %s\n",binary);
```

binary.s


```
.data
.global a
r: .int 0,0,0,0,0,0,0,0 k: .int 0
x: .int 0

.text
.globl binary
binary:
push %ebp
mov %esp, %ebp

mov 8(%ebp), %eax

 mov %eax, x
while:
cmp $0, x
je end_while
```

```
mov x, %eax
mov $2, %ebx
mov $0, %edx
div %ebx
mov k, %edi
mov %edx, a(,%edi,4) mov %eax, x
incl k
jmp while
end_while:
mov r, %eax
mov %ebp, %esp
pop %ebp
ret
```

Division occurs and the binary numbers are put into the array by indirect addressing.

convert.c

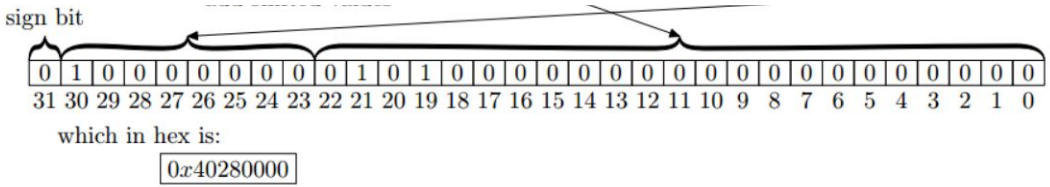
```
unsigned convert( float nf ){
        unsigned int * p = ( unsigned int * ) &nf ;
        unsigned k = *p ;
        return k ;
}
```

Here we are taking a float as input and returning

Are we are de-referencing a pointer to, a pointer?

PrintHexvalue.c

```
#include <stdio.h>
void PrintHexvalue( unsigned k ){
        char h[16] = "1234567890abcdef" ;
        putchar('0') ;
        putchar('x') ;
        unsigned int a ;
        for( int i = 0 ; i < 8 ; ++i ){
                a = k >> 28 ;
                k = k << 4 ;
                putchar(h[a]);
        }
        putchar('\n');
}
```



Putchar is a watered down version of print f. What is 28 and 4 supposed to be? Unsinged k is accepted for preservation of the information to keep

Attempted s code translation

Shift bits?

```
2.836
The length of the full length is :  1
The length of the fractional part is : 3
The whole part = 2
The fractional part = 836
Current, fp=0.836000
nf =2.836000
The whole binary = 10
The fraction binary = 1101011000
The number = 1077248256
The hexadecimal format is : 0x40358100
```

```
4.8376
The length of the full length is :  1
The length of the fractional part is : 4
The whole part = 4
The fractional part = 8376
Current, fp=0.837600
nf =4.837600
The whole binary = 100
The fraction binary = 1101011001
The number = 1083834526
The hexadecimal format is : 0x409A009E
```