(https://www.edx.org) Conversion | 11.3 Conversions | UT.6.01x Co... https://courses.edx.org/courses/UTAustinX/UT...

UTAustinX: UT.6.01x Embedded Systems - Shape the World KarenWest (/dashboard) ▼

Courseware (/courses/UTAustinX/UT.6.01x/1T2014/courseware) Course Info (/courses/UTAustinX/UT.6.01x/1T2014/info)

Discussion (/courses/UTAustinX/UT.6.01x/1T2014/discussion/forum) Progress (/courses/UTAustinX/UT.6.01x/1T2014/progress)

Questions (/courses/UTAustinX/UT.6.01x/1T2014/a3da417940af4ec49a9c02b3eae3460b/)

Syllabus (/courses/UTAustinX/UT.6.01x/1T2014/a827a8b3cc204927b6efaa49580170d1/)

Help

In this section we will develop methods to convert between ASCII strings and binary numbers. Let's begin with a simple example. Let **Data** be a fixed length string of three ASCII characters. Each entry of **Data** is an ASCII character 0 to 9. Let **Data[0]** be the ASCII code for the hundred's digit, **Data[1]** be the ten's digit and **Data[2]** be the one's digit. Let **n** be an unsigned 32-bit integer. We will also need an index, **i**. The decimal digits 0 to 9 are encoded in ASCII as 0x30 to 0x39. So, to convert a single ASCII digit to binary, we simply subtract 0x30. To convert this string of 3 decimal digits into binary we can simply calculate

```
n = 100*(Data[0]-0x30) + 10*(Data[1]-0x30) + (Data[2]-0x30);
```

Adding parentheses, we can convert this 3-digit ASCII string as

```
n = (Data[2]-0x30) + 10*((Data[1]-0x30) + 10*(Data[0]-0x30));
```

This second method could be used for converting any string of known and fixed length. If **Data** were a string of 9 decimal digits we could put the above function into a loop

```
n = 0;
for (i=0; i<9 ;i++){
  n = 10*n + (Data[i]-0x30);
}
```

If the length were variable, we can replace the for-loop with a while-loop. If **Data** were a variable length string of ASCII characters terminated with a null character (0), we could convert it to binary using a while loop, as shown in Program 11.2. A pointer to the string is passed using call by reference. In the assembly version, the pointer R0 is incremented as the string is parsed. R1 contains the local variable **n**, R2 contains the data from the string, and R3 contains the constant 10.

```
// Convert ASCII string to
//    unsigned 32-bit decimal
// string is null-terminated
unsigned long Str2UDec(unsigned char string[]){
  unsigned long i = 0;  // index
  unsigned long n = 0;  // number
  while(string[i] != 0){
    n = 10*n +(string[i]-0x30);
    i++;
  }
  return n;
}
```

Program 11.2. Unsigned ASCII string to decimal conversion.

The example, shown in Program 11.3, uses an I/O device capable of sending and receiving ASCII characters. When using a development board, we can send serial data to/from the PC using the UART. The function **`UART_InChar()`**returns an ASCII character from the I/O device. The function **`UART_OutChar()`**sends an ASCII character to the I/O device. The function **`UART_InUDec()`**will accept number characters (0x30 to 0x39) from the device until any non-number is typed. All input characters are echoed.

```
#define CR 0x0D
// Accept ASCII input in unsigned decimal format, up to 4294967295
// If n> 4294967295, it will truncate without reporting the error
unsigned long UART_InUDec(void){
unsigned long n=0;            // this will be the return value
unsigned char character;      // this is the input ASCII typed
  while(1){
   character = UART_InChar(); // accepts input
   UART_OutChar(character);   // echo this character
   if((character < '0') || (character > '9')){  // check for non-number
     return n;                // quit if not a number
   }
   n = 10*n+(character-0x30); // overflows if above 4294967295
  }
}
```

Program 11.3. Input an unsigned decimal number.

If the ASCII characters were to contain optional "+" and "-" signs, we could look for the presence of the sign character in the first position. If there is a minus sign, then set a flag. Next use our unsigned conversion routine to process the rest of the ASCII characters and generate the unsigned number, **n**. If the flag was previously set, we can negate the value **n**. Be careful to guarantee the + and – are only processed as the first character.

To convert an unsigned integer into a fixed length string of ASCII characters, we could use the integer divide. Assume **n** is an unsigned integer less than or equal to 999. In Program 11.4, the number 0 is converted to the string "000". The first program stores the conversion in an array and the second function outputs the conversion to the UART.

```
unsigned char Data[4]; // 4-byte empty buffer
// n is the input 0 to 999
void UDec2Str(unsigned short n){
  Data[0] = n/100 + 0x30; // hundreds digit
  n = n%100;              // n is now between 0 and 99
  Data[1] = n/10 + 0x30;  // tens digit
  n = n%10;               // n is now between 0 and 9
  Data[2] = n + 0x30;     // ones digit
  Data[3] = 0;            // null termination
}
// n is the input 0 to 999
void UART_OutUDec3(unsigned short n){
  UART_OutChar(0x30+n/100); // hundreds digit
  n = n%100;                // 0 to 99
  UART_OutChar(0x30+n/10);  // tens digit
  n = n%10;                 // 0 to 9
  UART_OutChar(0x30+n);     // ones digit
}
```

Program 11.4. Unsigned decimal to ASCII string conversion.

Sometimes we represent noninteger values using integers. For example the system could store the value 1.23 as the integer 123. In this example, the voltage ranges from 0.00 to 9.99V, but the values in the computer are stored as integers 0 to 999. If the system wishes to display those values in volts, we simply add a decimal point to the output while converting, as shown in Program 11.5. For example calling **`OutVolt(123)`** will output the string “1.23V”. Instead of creating an output string, this function outputs each character to display device by calling **`UART_OutChar`**.

```
//-----------------------UART_OutVolt-----------------------
// Output a voltage to the UART
// Input: n is an integer from 0 to 999 meaning 0.00 to 9.99V
// Output: none
// Fixed format: for example n=8 displayed as "0.08V"
void OutVolt(unsigned long n){  // each integer means 0.01V
  UART_OutChar(0x30+n/100);      // digit to the left of the decimal
  n = n%100;                     // 0 to 99
  UART_OutChar('.');             // decimal point
  UART_OutChar(0x30+n/10);       // tenths digit
  n = n%10;                      // 0 to 9
  UART_OutChar(0x30+n);          // hundredths digit
  UART_OutChar('V');}            // units
```

Program 11.5. Print the voltage value to an output device (0≤n≤999).

To convert an unsigned integer into a variable length string of ASCII characters, we convert the digits in reverse order, and then switch them.

```
//-----------------------UART_OutUDec-----------------------
// Output a 32-bit number in unsigned decimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1-10 digits with no space before or after
void UART_OutUDec(unsigned long n){
  if(n >= 10){
    UART_OutUDec(n/10);
    n = n%10;
  }
  UART_OutChar(n+'0'); /* n is between 0 and 9 */
}
```

Program 11.6. Print unsigned 32-bit decimal number to an output device.