

1 Converting between different numbering systems

The following questions will walk you through converting between binary (or base-2), decimal (base-10), and hexadecimal (base-16) numbers. In order to keep track of the three numbering systems, a number with a subscript 2, e.g., 101101_2 , refers to a number in binary, a subscript 10, like 55_{10} , is a number written in base-10, and a subscript 16, as in $7F_{16}$, refers to a number in hexadecimal (or hex for short).

The following recipe will convert from base-10 to binary:

1. Divide the number, n , by 2, keeping track of the quotient and the remainder
2. Repeat step 1, using the quotient in step 1) as your new number, n , until the quotient is zero
3. To get the binary number, simply write the remainders in reverse order

Example: Convert 55_{10} to binary.

Divide n by 2	Quotient	Remainder
$55/2$	27	1 (this is the last bit written)
$27/2$	13	1
$13/2$	6	1
$6/2$	3	0
$3/2$	1	1
$1/2$	0	1 (this is the first bit written)
$0/2$	0	0

Hence, 55_{10} in base-10 is 110111_2 in base-2 (we can ignore leading zeros).

Questions

- (a) What is 79_{10} in binary?
- (b) What is 18_{10} in binary?

The conversion from binary to decimal is a little simpler, but first we introduce a commonly used convention. A generic n -bit binary number is often written as: $b_{n-1}b_{n-2}...b_i...b_2b_1b_0$, where i is the position (or index) of the bit and b_i is its value (this is the same convention used with arrays in C). We refer to b_{n-1} as the most-significant bit (MSB)¹ and b_0 as the least-significant bit (LSB). To convert from binary to decimal, you compute the following sum:

$$d = \sum_{i=0}^{n-1} 2^i \times b_i = (b_{n-1} \times 2^{n-1}) + \dots + (b_1 \times 2^1) + (b_0 \times 2^0)$$

where d is the final number in base-10 and b_i is the value of the $(i + 1)$ -st bit in the binary number.

Example: Convert 101101_2 to base-10.

$$101101_2 = (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 45_{10}$$

¹In certain contexts, MSB can also refer to the most significant byte. It should always be clear from context, whether bit or byte is assumed.

Questions

(a) What is 10110101_2 in decimal?

As the previous two examples have shown, converting between binary and decimal numbers isn't always the quickest operation and dealing with long binary numbers can get cumbersome. This is where hexadecimal or base-16 numbers come in. The following table shows the relationship between the three bases.

decimal	binary	hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Similar rules can be developed to convert from binary to hexadecimal, but here is a simple and quick shortcut.

1. Starting with the LSB partition the binary number into groups of 4 (remember leading zeros do not affect the number, so every bit should be in a group of 4)
2. Replace each group of 4 with its corresponding hexadecimal number

Example: Convert 101101_2 to hexadecimal.

$$101101_2 = (0010)(1101)_2 = 2D_{16}$$

Questions

(a) What is 10110101_2 in hexadecimal?

(b) What is 80_{10} in hexadecimal?

2 The C bitwise operators

Questions

After reading Essential C p. 4 int Constants and pp. 9-10 Bitwise Operators,

- (a) What is 80_{10} in hexadecimal using C syntax?
- (b) What are the four bitwise operators?
- (c) Write the truth table for each bitwise operator

After reading section 1.11, Binary Constants in MPLAB C Compiler For PIC32 MCUs Users Guide

- (a) What is the syntax for assigning 80_{10} in binary to an int variable x?
- (b) Is the syntax for writing binary numbers portable (e.g., can it be used outside the Microchip C compiler) ?

The C programming language does not have a bit data type, but, with its bitwise operators, we can modify the bits within an int (and its qualifiers like short, long, signed, unsigned, etc.) or char data type. The bitwise operators are applied to each individual bit and the result is stored in a data type of the same size.

Example: In the lines below, what is the final value of z?

```
char x = 0xC4;
char y = 0xAA;
char z = x | y;
```

variable	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
x	1	1	0	0	0	1	0	0
y	1	0	1	0	1	0	1	0
z	1	1	1	0	1	1	1	0

Hence, $z = 0xEE$.

Questions

- (a) What is $0x55 \& 0x17$?
- (b) What is $0x55 \wedge 0x17$?
- (c) What is $\sim 0x55 | 0x17$?
- (d) Given $a = 0x783F$, $c = 0x15E$, what is b in $c = a \& b$?

3 The PIC32 Hardware

The following questions are based on the wiki page, NU32v2: Introduction to the PIC32. You may also have to consult the PIC32 data sheet (DS) and reference manual(RM).

Questions

- (a) What does “peripheral” mean for the PIC?

- (b) Go to the Microchip homepage, check out the Parametric Table of PIC32s, and find a PIC with the following specs: 80 MHz max clock speed, 512 K flash (program memory), 32 K RAM (data memory), 4 channels of DMA, 16 A/D converters with 10-bit resolution, USB capabilities, 2 comparators, 5 16-bit timers and 1 32-bit timer, 5 PWM channels and 5 input compare channels, 2 UARTs, 2 SPI, and 2 I2C pins, no CAN modules, no Ethernet, and 100 pins. What is the part number? What types of packages does it come in (e.g., DIP, or different kinds of surface mount packages, etc.)? How much does it cost in quantity 1? What is the difference in price and features from our PIC, the PIC32MX795F512L?
- (c) Refer to the “Block diagram of the PIC32MX7XX” under “PIC32 Hardware Overview” or p. 31 “Device Overview” in the DS. In one sentence each, without going into detail, explain the basic function of the following items shown in the block diagram: Timing Generation, MCLR, SYSCLK, PBCLK, USBCLK, PORTA...G (and indicate which of these can be used for analog input on our PIC), Timer 1-5, 10-bit ADC, Comparators, Ethernet, UART 1-6, I2C 1-5, SPI 1-4, IC 1-5, PWM OC 1-5, CN1-22, Data RAM, Program Flash Memory, and Pre-Fetch Module.
- (d) Describe the four functions that pin 23 of our PIC32 can have. Is it 5V tolerant?
- (e) How many pins from the PIC32 are available on the NU32v2 board?
- (f) What is the absolute maximum amount of current that the NU32v2 board can provide from the 9V wall plug? What are some limiting factors in drawing the full amount of current from the plug?

4 Putting it all together

You’ve been introduced to different number systems, new C syntax, and the PIC32 hardware. In the following programming exercise, you will learn how to interact with the LATG register in C to modify the blinking pattern of the LEDs on the NU32v2 board. Deeply buried in the Microchip header files (which you indirectly include through `<plib.h>`), LATG is defined as an unsigned int. Unfortunately, the problem with using a C data type is that we can’t directly manipulate the bits. We could access the bits one by one using the predefined structs in `<plib.h>`, but what if we wanted to change multiple bits at the same time? This is where the bitwise operators come in, especially bitwise-and (`&`) and bitwise-or (`|`). The following are useful rules to remember:

- To set the i -th bit(s) to 1 (or “on”) use bitwise-or (`|`), to leave the bit(s) unchanged use 0
- To clear the i -th bit(s) to 0 (or “off”) use bitwise-and (`&`), to leave the bit(s) unchanged use 1

For example, say that the current state of LATG = 0x9F35, if we wanted to only change the output of pin G12 to low while preserving the state of the other pins, then we would turn the PORTG bit position 12 to “off” using bitwise-and. To pose the problem differently, given, $a = 0x9F35$, $c = 0x8F35$, what is b in $c = a \& b$?

Bit	G15	G14	G13	G12	G11	G10	G9	G8
$a = \text{LATG} \langle 15 : 8 \rangle$ (initially)	1	0	0	1	1	1	1	1
$b = \text{MASK} \langle 15 : 8 \rangle$?	?	?	?	?	?	?	?
$c = \text{LATG} \langle 15 : 8 \rangle$ (desired)	1	0	0	0	1	1	1	1

Bit	G7	G6	G5	G4	G3	G2	G1	G0
a = LATG< 7 : 0 > (initially)	0	0	1	1	0	1	0	1
b = MASK< 7 : 0 >	?	?	?	?	?	?	?	?
c = LATG< 7 : 0 > (desired)	0	0	1	1	0	1	0	1

The value for b (aka MASK) has to be 0xEFFF, you can verify this for yourself. This will clear bit G12 and preserve the previous state of the other pins. The corresponding C code is:

```
LATG = LATG & 0xEFFF;
```

We would actually define 0xEFFF as a constant. This type of constant is known as a bit mask or mask for short. If later on we change our minds and we want to set G12 back to high, then we would do:

```
LATG = LATG | 0x1000;
```

You can work this out to verify that G12 is now set to high and that we've left all other pins unmodified.

Programming Assignment

Starting with the HelloWorld Code modify only the while loop in main, so that the LEDs blink in the following sequence:

1. Both LED_1 and LED_2 are off for WAIT_LOOPS amount of time
2. Only LED_1 is on for WAIT_LOOPS amount of time
3. Only LED_2 is on for WAIT_LOOPS amount of time
4. Both LED_1 and LED_2 are on for WAIT_LOOPS amount of time

You must use the LATG register and the bitwise operators in your solution. There should be no reference to the constants LED_1 or LED_2 in your code.