ScienceDirect

Hexadecimal

The binary or hexadecimal code for each digit will be included in the program in the form of a program data table.

From: PIC Microcontrollers (Third Edition), 2011

Related terms:

Semiconductor, Amplifier, Resistor, Transistor, Impedance, Oscillators, Magnetic Fields, Amplitudes, Binary Digit, Electric Potential

Logic and numbering systems

Theresa Schousek, in The Art of Assembly Language Programming Using PIC© Technology, 2018

Hexadecimal System

Hexadecimal is the name of the numbering system that is base 16. This system, therefore, has numerals 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, and 15. That means that two-digit decimal numbers 10, 11, 12, 13, 14, and 15 must be represented by a single numeral to exist in this numbering system. To address the two-digit decimal values, the alphabetic characters A, B, C, D, E, and F are used to represent these values in hexadecimal and are treated as valid numerals. Row 1 is, again, the decimal (base 10) equivalent value.

268,435,456	**16,777,216**	**1,048,576**
16 × 16 × 16 × 16 × 16 × 16 × 16	16 × 16 × 16 × 16 × 16 × 16	16 × 16 × 16 × 16 × 16
16^7	16^6	16^5
0	**0**	**0**

Using an example value of 538, what is the base 16 equivalent value. First place zeros in all columns that are known to be too big. 256 is the start of values below 538, so put a 0 in all columns to the left of 256.

268,435,456	**16,777,216**	**1,048,576**
16 × 16 × 16 × 16 × 16 × 16 × 16	16 × 16 × 16 × 16 × 16 × 16	16 × 16 × 16 × 16 × 16
16^7	16^6	16^5
0	**0**	**0**

538 – (2 × 256) = 26.

268,435,456	**16,777,216**	**1,048,576**
16 × 16 × 16 × 16 × 16 × 16 × 16	16 × 16 × 16 × 16 × 16 × 16	16 × 16 × 16 × 16 × 16
16^7	16^6	16^5

268,435,456	16,777,216	1,048,576
0	0	0

26 – (1 × 16) = 10

268,435,456	16,777,216	1,048,576
16 × 16 × 16 × 16 × 16 × 16 × 16	16 × 16 × 16 × 16 × 16 × 16	16 × 16 × 16 × 16 × 16
16^7	16^6	16^5
0	0	0

10 – (10 * 1) = 0. Recall, 10 decimal is represented by Ah hexadecimal. The complete value is then 21 Ah

268,435,456	16,777,216	1,048,576
16 × 16 × 16 × 16 × 16 × 16 × 16	16 × 16 × 16 × 16 × 16 × 16	16 × 16 × 16 × 16 × 16
16^7	16^6	16^5
0	0	0

This will come to you with practice and memorization of typical values. Rarely do we work with numbers greater than 4096 in hexadecimal.

Read full chapter

URL: https://www.sciencedirect.com/science/article/pii/B9780128126172000092

Data Processing

Martin Bates, in Interfacing PIC Microcontrollers (Second Edition), 2014

5.2.3 Binary to Hex

Binary to hex conversion is simple–that is why hex is used. Each group of 4 bits is converted to the corresponding hex digit, starting with the least significant four, and padding with leading zeros if necessary:

1001	**1111**	**0011**	**1101**	**=9F3D$_{16}$**
9	F	3	D	

The reverse process is just as trivial, where each hex digit is converted to a group of 4 bits, in order. The result can be checked by converting both to decimal. First binary to decimal:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**1**

$= 2^{15} + 2^{12} + 2^{11} + 2^{10} + 2^9 + 2^8 + 2^5 + 2^4 + 2^3 + 2^2 + 2^0$

$= 32{,}768 + 4096 + 2048 + 1024 + 512 + 256 + 32 + 16 + 8 + 4 + 1$

$= 40{,}765_{10}$

Now hex to decimal:

$$9F3D_{16} = (9\times16^3) + (15\times16^2) + (3\times16^1) + (13\times16^0)$$
$$= 36{,}864 + 3840 + 48 + 13$$
$$= 40{,}765_{10}$$

Read full chapter

URL: https://www.sciencedirect.com/science/article/pii/B9780080993638000054

Think Digitally

Adrian Fernandez, Dung Dang, in Getting Started with the MSP430 Launchpad, 2013

6.2 Hexadecimal—Binary's Big Brother

Hexadecimal is another way of representing numbers. Hexadecimal is an interesting numbering scheme, which counts from 0 through F. Let's take a closer look at what we mean.

This table summarizes hexadecimal, binary, and decimal format.

Decimal (Base 10)	Binary (Base 2)	Hexadecimal (Base 16)
0	00000011	0x0
1	00000001	0x1
2	00000010	0x2
3	00000011	0x3
4	00000100	0x4
5	00000101	0x5
6	00000110	0x6
7	00000111	0x7
8	00001000	0x8
9	00001001	0x9
10	00001010	0xA
11	00001011	0xB
12	00001100	0xC
13	00001101	0xD
14	00001110	0xE
15	00001111	0xF

Note

While binary numbers usually stand out having only 0s and 1s, hexadecimal numbers sometimes might look like decimal numbers when A–F are not used. To identify a number as a hexadecimal value instead of a decimal value, a 0x-prefix can be added as shown in the table above.

Hexadecimal format is used in lots of microcontroller code because of the way many microcontrollers handle and manipulate data. Most microcontrollers work with data 8-bits at a time. This is what we call a *byte.* We typically represent a byte of data in either a string of eight 1s or 0s, or as a 2-digit hexadecimal value. To convert from binary to hex, we simply split an 8-bit binary number into two sets of 4-bits. Once we've split it into two 4-bit values, we can determine the appropriate hex value.

Example

$1001\ 1110_2 \rightarrow 9E_{16}$

Challenges:

Convert the following binary numbers into decimal! To check if you're right, flip to that page in this book!

- 01010111_2
- 00101111_2

Convert the following hexadecimal numbers into decimal! To check if you're right, flip to that page in this book!

- 0x74
- 0x63

Read full chapter

URL: https://www.sciencedirect.com/science/article/pii/B9780124115880000066

Microcomputer Systems

Dogan Ibrahim, in Designing Embedded Systems with 32-Bit PIC Microcontrollers and MikroC, 2014

1.11 Converting Hexadecimal Numbers into Decimal

To convert a hexadecimal number into decimal, we have to calculate the sum of the powers of 16 of the number.

Example 1.11

Convert hexadecimal number $2AC_{16}$ into decimal.

Solution 1.11

Calculating the sum of the powers of 16 of the number:

$$\begin{aligned} 2AC_{16} &= 2 \times 16^2 + 10 \times 16^1 + 12 \times 16^0 \\ &= 512 + 160 + 12 \\ &= 684 \end{aligned}$$

The required decimal number is 684_{10}.

Example 1.12

Convert hexadecimal number EE_{16} into decimal.

Solution 1.12

Calculating the sum of the powers of 16 of the number:

$$\begin{aligned} EE_{16} &= 14 \times 16^1 + 14 \times 16^0 \\ &= 224 + 14 \\ &= 238 \end{aligned}$$

The required decimal number is 238_{10}.

Read full chapter

URL: https://www.sciencedirect.com/science/article/pii/B9780080977867000014

Hardware Prototyping

Martin Bates, in PIC Microcontrollers (Third Edition), 2011

10.4.4 DIZI Application Outlines

A further eight applications are specified below, and the source code for each is listed in Programs 10.4. They can be downloaded from www.picmicros.org.uk, and tested in simulation mode in MPSIM or ISIS (if available). If the DIZI hardware is constructed, they can be programmed into a 16F84A chip using an out-of-circuit programmer.

```
;***************************
;  BEL1.ASM   MPB 2-12-10
;...........................
;  Program to output a tone
;  Sequence (random) of 8
;  RB0 = Output Buzzer
;  RA0 = Input Button
;  .........................

       PROCESSOR 16F84

PCL     EQU     02
PortB   EQU     06
PortA   EQU     05
Notnum  EQU     0C
Tabnum  EQU     0D
Cycnum  EQU     0E
Count   EQU     0F

;Initialise ...............

       MOVLW   B'11111110'
       TRIS    PortB
Wait   BTFSC   PortA,4
       GOTO    Wait

; Get note ................

Start  MOVLW   08
       MOVWF   Notnum
Nexnot MOVF    Notnum,W
       CALL    Table
       MOVWF   Tabnum

; 256 Cycles of note ......

       CLRW
       MOVWF   Cycnum
Cycle  BSF     PortB,0
       CALL    Half
       BCF     PortB,0
       CALL    Half
       DECFSZ  Cycnum
       GOTO    Cycle

; Next note of 8 .........

       DECFSZ  Notnum
       GOTO    Nexnot
       GOTO    Wait

;Half cycle delay ........

Half   MOVF    Tabnum,W
       MOVWF   Count
Down   DECFSZ  Count
       GOTO    Down
       RETURN

;Table of delay values....

Table  ADDWF   PCL
       NOP
       RETLW   D'124'
       RETLW   D'82'
       RETLW   D'117'
       RETLW   D'156'
       RETLW   D'77'
       RETLW   D'156'
       RETLW   D'92'
       RETLW   D'104'

       END
```

```
;*******************************
;  GEN1.ASM  MPB 2-12-10
;  Audio generator 200Hz-20kHz
;  RB0 = Output to buzzer
;  RA0 = Decrease frequency
;*******************************

       PROCESSOR 16F84A

PORTA   EQU     05
PORTB   EQU     06
Multi   EQU     0C
Count1  EQU     0D
Count2  EQU     0E

; Initialise ...............

       MOVLW   B'11111110'
       TRIS    PORTB
       MOVLW   02
       MOVWF   Multi

; Output one cycle .........

Cycle  BSF     PORTB,0
       CALL    Half
       BCF     PORTB,0
       CALL    Half
       BTFSC   PORTA,4
       GOTO    Cycle

; Reduce frequency.........

       INCF    Multi
       CLRF    Count2
Down2  DECFSZ  Count2
       GOTO    Down2
Wait   BTFSS   PORTA,4
       GOTO    Wait
       GOTO    Cycle

; Delay one half cycle...

Half   MOVF    Multi,W
       MOVWF   Count1
Down1  NOP
       NOP
       NOP
       NOP
       NOP
       DECFSZ  Count1
       GOTO    Down1
       RETURN
```

```
;************************************
;  GIT1.ASM   MPB  2-12-10
;  Guitar Tuner
;  Outputs standard frequencies
;  330,245,196,147,110,82Hz
;  3030,4081,5102,6802,9090,12195us
;  Count = 30,41,51,68,91,122 x50us
;  Measured accurate to about 1%
;  RB0 = buzzer(string tone)
;  RA4 = button(next string)
;
;************************************

PortA     EQU       05
PortB     EQU       06
String    EQU       0C
Count1    EQU       0D
Count2    EQU       0E
PCL       EQU       02

          PROCESSOR 16F84A

; Initialise ..................

          MOVLW     B'11111110'
          TRIS      PortB
          MOVLW     06
          MOVWF     String

; Output one cycle .............

Next      BSF       PortB,0
          CALL      Cycle
          BCF       PortB,0
          CALL      Cycle
          GOTO      Next

; Delay and check inputs ......

Cycle     MOVF      String,W
          CALL      Table
          CALL      Tone
          BTFSS     PortA,4
          CALL      Wait1
          RETURN

; Select next tone ...........

Wait1     BTFSS     PortA,4
          GOTO      Wait1
          DECFSZ    String
          RETURN
          MOVLW     06
          MOVWF     String
          RETURN

;Table of tone values..........

Table     ADDWF     02
          NOP
          RETLW     D'122'
          RETLW     D'91'
          RETLW     D'68'
          RETLW     D'51'
          RETLW     D'41'
          RETLW     D'30'

; Subroutine to generate Tone..

Tone      MOVWF     Count1
Loop1     CALL      Fifty
          DECFSZ    Count1
          GOTO      Loop1
          RETURN
```

```
; Subroutine 50us   delay .....

Fifty     NOP
          NOP
          MOVLW     08
          MOVWF     Count2
Loop2     NOP
          NOP
          DECFSZ    Count2
          GOTO      Loop2
          RETURN

          END
```

```
;**********************************
;       HEX1.ASM  MPB  2-12-10
;       Program to convert binary
;       input to 7 segment output
; ********************************

        PROCESSOR 16F84A

PortA   EQU    05
PortB   EQU    06
PCL     EQU    02

        MOVLW  B'0000000'
        TRIS   PortB

Start   MOVF   PortA,W
        ANDLW  B'00001111'
        CALL   Table
        MOVWF  PortB
        GOTO   Start

Table   ADDWF  PCL
        RETLW  07E
        RETLW  00C
        RETLW  0B6
        RETLW  09E
        RETLW  0CC
        RETLW  0DA
        RETLW  0FA
        RETLW  00E
        RETLW  0FE
        RETLW  0CE
        RETLW  0EE
        RETLW  0F8
        RETLW  072
        RETLW  0BC
        RETLW  0F2
        RETLW  0E2

        END
```

```
;***************************
;       MESS1.ASM
;       MPB 2-12-10
;       Message display
; *************************

        PROCESSOR 16F84A

PCL     EQU     02
PortA   EQU     05
PortB   EQU     06
Timer1  EQU     0C
Timer2  EQU     0D
Timer3  EQU     0E
count   EQU     0F

; Initialise...............

        CLRW
        TRIS    PortB

; Output loop..............

repeat  MOVLW   D'12'
        MOVWF   count

next    MOVF    count,w
        CALL    table
        MOVWF   PortB
        CALL    delay
        DECFSZ  count
        GOTO    next
        GOTO    repeat

; Meassage delays..........

delay   MOVLW   05
        MOVWF   Timer3

loop3   MOVLW   0FF
        MOVWF   Timer2
loop2   MOVLW   0FF
        MOVWF   Timer1
loop1   DECFSZ  Timer1
        GOTO    loop1

        DECFSZ  Timer2
        GOTO    loop2
        DECFSZ  Timer3
        GOTO    loop3
        RETURN

; Message characters.....

table   ADDWF   PCL
        NOP
        RETLW   B'00000000'
        RETLW   B'00000000'
        RETLW   B'01111110'
        RETLW   B'00000000'
        RETLW   B'01110000'
        RETLW   B'00000000'
        RETLW   B'01110000'
        RETLW   B'00000000'
        RETLW   B'11110010'
        RETLW   B'00000000'
        RETLW   B'11101100'
        RETLW   B'00000000'

        END
```

```
;*******************************
;       MET1.ASM  MPB  2-12-10
;       Program to output beeps
;       between 0.1-10Hz
;       RB0 = Output Buzzer
;       RA0 = Input Button Up
;       RA1 = Input Button Down
; ..............................

        PROCESSOR 16F84A

PortB   EQU     06
PortA   EQU     05
Count1  EQU     0C
Count2  EQU     0D
Count3  EQU     0E
Wait1   EQU     0F
Count0  EQU     10

; Initialise...................

        MOVLW   B'11111110'
        TRIS    PortB
        MOVLW   D'10'
        MOVWF   Wait1

; Main loop..................

start   MOVLW   020
        MOVWF   Count0
beep    BSF     PortB,0
        CALL    delay1
        BCF     PortB,0
        CALL    delay1
        DECFSZ  Count0
        GOTO    beep

; Read buttons................

fup     BTFSS   PortA,0
        DECFSZ  Wait1
        GOTO    fdown
        INCF    Wait1
fdown   BTFSS   PortA,1
        INCFSZ  Wait1
        GOTO    Wait
        DECF    Wait1

; Wait 0.1 - 2.5s..........

Wait    MOVF    Wait1,w
        MOVWF   Count3
loop3   CALL    del100
        DECFSZ  Count3
        GOTO    loop3
        GOTO    start

; Wait 100ms...............

del100  MOVLW   D'100'
        MOVWF   Count2
loop2   CALL    delay1
        DECFSZ  Count2
        GOTO    loop2
        RETURN

; 1ms Delay...............

delay1  MOVLW   D'250'
        MOVWF   Count1
loop1   NOP
        DECFSZ  Count1
        GOTO    loop1
        RETURN
        END
```

```
;******************************
;  REACT1.ASM  MPB 30-11-10
;  Reaction time program
;  RB0 = Buzzer
;  RA4 = Test Input
;  RB1-RR7 = Display
;******************************

       PROCESSOR 16F84A

PortA  EQU    05
PortB  EQU    06
Random EQU    0C
Rtime  EQU    0D
Count3 EQU    0E
Count2 EQU    0F
Count1 EQU    10

; Initialise..................

       MOVLW  B'00000000'
       TRIS   PortB
       MOVLW  0FF
       MOVWF  PortB

; Generate random count 0-100..

wait   BTFSC  PortA,4
       GOTO   wait
       CALL   onehun
       CLRW
       MOVWF  PortB
reload MOVLW  D'100'
       MOVWF  Random

down   BTFSC  PortA,4
       GOTO   randel
       DECFSZ Random
       GOTO   down
       GOTO   reload

; Delay for random time(0-10s)..

randel CALL   onehun
       DECFSZ Random
       GOTO   randel

; Beep and start timer(512ms)..

       CLRF   Rtime
beep   BSF    PortB,0
       CALL   onems
       BCF    PortB,0
       CALL   onems
       BTFSS  PortA,4
       GOTO   stop
       INCFSZ Rtime
       GOTO   beep

; Divide Reaction time by 32..

stop   MOVLW  4
       MOVWF  Count3
loop3  BCF    3,0
       RRF    Rtime
       DECFSZ Count3
       GOTO   loop3
```

```
; Display reaction time..

       MOVF   Rtime,W
       CALL   table
       MOVWF  PortB
done   CALL   onehun
       BTFSS  PortA,4
       GOTO   done
       GOTO   wait

;100ms delay............

onehun MOVLW  D'100'
       MOVWF  Count2
loop2  CALL   onems
       DECFSZ Count2
       GOTO   loop2
       RETURN

; 1ms delay..................

onems  MOVLW  D'249'
       MOVWF  Count1
loop1  NOP
       DECFSZ Count1
       GOTO   loop1
       RETURN

; Display codes 0-9...........

table  ADDWF  002
       RETLW  0EC    ; H
       RETLW  00C    ; 1
       RETLW  0B7    ; 2
       RETLW  09E    ; 3
       RETLW  0CC    ; 4
       RETLW  0DA    ; 5
       RETLW  0EA    ; 6
       RETLW  00E    ; 7
       RETLW  0FE    ; 8
       RETLW  0CE    ; 9
       RETLW  0EC    ; H
       RETLW  0EC    ; H
       RETLW  0EC    ; H
       RETLW  0EC    ; H
       RETLW  0EC    ; H

       END
```

```
;*****************************
;       SEC1.ASM
;       MPB 30-11-10
;       One second counter
;*****************************

        PROCESSOR 16F84A

PCL     EQU     02
PortA   EQU     05
PortB   EQU     06
count   EQU     0C
Timer0  EQU     0D
Timer1  EQU     0E
Timer2  EQU     0F

        CLRW
        TRIS    PortB

repeat  MOVLW   D'10'
        MOVWF   count

next    MOVF    count,w
        CALL    table
        MOVWF   PortB
        CALL    delay
        DECFSZ  count
        GOTO    next
        GOTO    repeat

delay   MOVLW   D'25'
        MOVWF   Timer0
loop0   MOVLW   D'100'
        MOVWF   Timer1
loop1   MOVLW   D'99'
        MOVWF    Timer2
loop2   NOP
        DECFSZ   Timer2
        GOTO     loop2
        DECFSZ  Timer1
        GOTO    loop1
        DECFSZ  Timer0
        GOTO    loop0
        RETURN

Table   ADDWF   PCL
        NOP
        RETLW    07E
        RETLW    00C
        RETLW    0B6
        RETLW    09E
        RETLW    0CC
        RETLW    0DA
        RETLW    0FA
        RETLW    00E
        RETLW    0FE
        RETLW    0CE

        END
```

Programs 10.4. 8 DIZI applications.

HEX1 Hex Converter

The hexadecimal number corresponding to the binary setting of the DIP switch inputs is displayed. The input switches select from a table of 16 seven-segment codes which drive the display in the required pattern for each hex digit: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, C, d and E. Note that numbers B and D are displayed in lower case so that they can be distinguished from 8 and 0, respectively.

MESS1 Message Display

A sequence of characters is displayed for about 0.5 s each. Most letters of the alphabet can be obtained on the seven-segment display in either upper or lower case, for instance 'HELLO'. The number of characters must be set in a counter, or a termination character used.

SEC1 Second Counter

An output is displayed which counts down exactly once per second, from 0 to 9, and then repeats. A table of display codes is required as in the Hex Converter application. A 1 s time delay can be achieved using the hardware timer (Chapter 6) and spare register. A tick could be produced at the audio output by pulsing the speaker at each step.

REACT1 Reaction Timer

The user's reaction time is tested by generating a random delay of between 1 and 10 s, outputting a beep, and timing the delay before the input button is pressed. A number representing the time between the sound and the input, in multiples of 100 ms, should be displayed as a number 0–9, giving a maximum reaction time of 900 ms.

GEN1 AF Generator

An audio frequency generator outputs frequencies in the range 20 Hz to 20 kHz. The sounder output is toggled with a delay between each operation determined by the frequency required, as in the BUZZ1 program. For example, for a frequency of 1 kHz, a delay of 1 ms is required, which is 1000 instruction cycles at a cycle time of 1 μs. The information on program timing must be studied in Chapter 6. The delay time, and hence the frequency, can then be incremented using the input button, and range selection with the input switches might be incorporated, as there are only 255 steps available when using an 8-bit register as the period counter.

MET1 Metronome

An audible pulse is output at a rate set by the DIP switches or input buttons. The output tick can be adjustable from, say, 1 up to 4 beats per second, using the interrupt button to step the speed up and down, and the input button to select up or down. A software loop or the TMR0 register can be used to provide the necessary time delays.

BELL1 Doorbell

A tune is played when the input button is pressed, using a program look-up table for the tone frequency and duration. Each tone must be played for a suitable time, or number of cycles, as required by the tune. The program can be elaborated by selecting a tune using the DIP switches, and displaying the number of the tune selected.

GIT1 Guitar Tuner

The program will allow the user to step through the frequencies for tuning the strings of a guitar, or another musical instrument using the input button, or selecting the tone at the DIP switches. The program could be enhanced by displaying the string number to be tuned. The tone frequencies will be generated as for the doorbell application. The digit display codes would also be required in a table.

Questions 10

1. State one advantage and one disadvantage of: (a) breadboard; (b) stripboard; (c) simulation for testing prototype designs. (6)
2. State an output binary code for: (a) all segments off and (b) displaying a ‘2’ in a common cathode seven-segment LED display, assuming the connections shown in Figure 10.15. (4)
3. Outline an algorithm for generating a fixed frequency output of approximately 1 kHz from the DIZI board using the hardware timer. (5)
4. Draw a flowchart representing the process of generating a ‘random’ delay between a button being pressed and an output LED being switched on. (5)

Answers on page 423. (Total 20 marks)

Activities 10

1. Build the DIZI circuit on breadboard, stripboard or PCB and test the programs BUZZ1, DICE1 and SCALE1.
2. Confirm by calculation or simulation that the values used in the program data table in SCALE1.ASM will give the required delays.
3. Devise a breadboard layout for the BIN circuit in Figure 3.3. Build the circuit and test the BINx programs.
4. Design and implement one of the programs outlined for the DIZI hardware, and compare your solution with the model programs provided for HEX1, MESS1, SEC1, REACT1, GEN1, MET1, BELL1 or GIT1.
5. (a) Investigate how input from a numeric keypad can be detected. Refer to Chapter 1, Section 1.4.1. The typical keypad, shown in Figure 10.17, has 12 keys in four rows of three: 1, 2, 3; 4, 5, 6; 7, 8, 9; *, 0, #. These are connected to seven terminals, and can bescanned in rows and columns. A key press is detected as a connection between a row and column. The pull-up resistors ensure that all lines default to logic ‘1’. If a ‘0’ is applied to one of the column terminals (C1, C2, C3), and a key is pressed, this ‘0’ can be detected at the row terminal (R1, R2, R3, R4). If

the keypad terminals are connected to a PIC port, and a '0' output in rotation to the three columns, a key can be detected as a combination of the column selected and the row detected. Column terminals can be set as outputs, and rows as inputs. Draw a flowchart to represent the process for converting each decimal key into the corresponding BCD number.

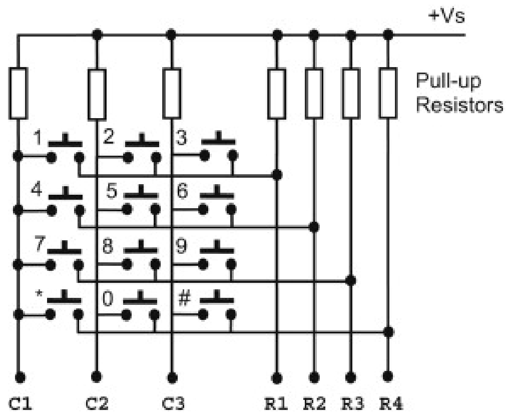


Figure 10.17. Keypad connections

(b) A lock function may be implemented by matching an input sequence with a stored sequence of, say, four digits, and switching on an output to a door solenoid if a match is detected. Specify the hardware and outline the program for the lock application.

(c) Design, build and test an electronic lock system using the keypad shown, a suitable PIC and an LED to indicate the state of the lock (ON = unlocked). Research the design for the interface to a solenoid operated door lock.

Note: Keypad scanning is used in Program 13.1, and a lock application outlined in Appendix D.

Read full chapter

URL: https://www.sciencedirect.com/science/article/pii/B9780080969114100102

Microcontroller Systems

Dogan Ibrahim, in SD Card Projects Using the PIC Microcontroller, 2010

1.9 Converting Binary Numbers into Hexadecimal

To convert a binary number into hexadecimal, arrange the number in groups of four and find the hexadecimal equivalent of each group. If the number cannot be divided exactly into groups of four, insert zeroes to the left-hand side of the number.

Example 1.6

Convert binary number 10011111_2 into hexadecimal.

Solution

First, divide the number into groups of four and then find the hexadecimal equivalent of each group:

$$10011111 = \underbrace{1001}_{9}\;\underbrace{1111}_{F}$$

The required hexadecimal number is $9F_{16}$.

Example 1.7

Convert binary number 1110111100001110_2 into hexadecimal.

Solution

First, divide the number into groups of four and then find the equivalent of each group:

$$1110111100001110 = \underset{E}{1110}\ \underset{F}{1111}\ \underset{0}{0000}\ \underset{E}{1110}$$

The required hexadecimal number is $EF0E_{16}$.

Example 1.8

Convert binary number 111110_2 into hexadecimal.

Solution

Because the number cannot be divided exactly into groups of four, we have to insert zeroes to the left of the number:

$$111110 = \underset{3}{0011}\ \underset{E}{1110}$$

The required hexadecimal number is $3E_{16}$.

Table 1.2 shows the hexadecimal equivalent of decimal numbers 0 to 31.

Table 1.2. Hexadecimal Equivalent of Decimal Numbers

Decimal	Hexadecimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10
17	11
18	12
19	13
20	14
21	15
22	16
23	17

Decimal	Hexadecimal
24	18
25	19
26	1A
27	1B
28	1C
29	1D
30	1E
31	1F

Read full chapter

URL: https://www.sciencedirect.com/science/article/pii/B9781856177191000051

Number Systems

MICHAEL L. SCHMIT, in Pentium™ Processor, 1995

Publisher Summary

This chapter reviews binary, hexadecimal, and decimal number systems. Decimal numbers are used for money, time, measurements, and even television channels. Everything is based on decimal except the internals of computers and other electronic devices. The binary number system is used internally in every computer. Binary or base two has two digits, 0 and 1. Decimal or base 10 has 10 digits, 0 through 9. Computers use binary because the electronic circuits can have only two states, on or off. Different devices may use different physical properties; a magnetic disk may store binary digits as magnetized or not magnetized or as north or south but the effect is the same—on or off. Decimal numbers are formed by combining a number of digits.

Read full chapter

URL: https://www.sciencedirect.com/science/article/pii/B9780126272307500069

Programming with the MPLAB C18 Compiler

Dogan Ibrahim, in SD Card Projects Using the PIC Microcontroller, 2010

Integer Constants

Integer constants can be decimal, hexadecimal, octal, or binary. The data type of a constant is derived by the compiler from its value. However, suffixes can be used to change the type of a constant.

From Table 4.1, we can see that integer variables can be 8, 16, 24, or 32 bit wide. In C18 language, Constants are declared using the key word **const rom**, and they are stored in the flash *program memory* of the PIC microcontroller, thus not wasting any valuable RAM space (it is important to note that in most C languages constants are declared using the key word **const** only). In the following example, constant integer **MAX** is declared as 100 and is stored in the flash program memory of the PIC microcontroller:

```
const rom int MAX = 100;
```

Hexadecimal constants start with the characters 0x and may contain numeric data 0–9 and hexadecimal characters A–F. In the following example, constant **TOTAL** is given the hexadecimal value FF:

```
const rom int TOTAL = 0xFF;
```

Octal constants have a zero at the beginning of the number and may contain numeric data 0–7. In the following example, constant **CNT** is given an octal value 17:

```
const rom int CNT = 017;
```

Binary constant numbers start with 0b and may contain only 0 or 1. In the following example, a constant named **Min** is declared having the binary value "11110000":

```
const rom int Min = 0b11110000
```

Read full chapter

URL: https://www.sciencedirect.com/science/article/pii/B9781856177191000087

Non-integral mathematics

Larry D. Pyeatt, William Ughetta, in ARM 64-Bit Assembly Language, 2020

8.1.2.1 Bases that are powers-of-two

Converting fractional values between binary, hexadecimal, and octal can be accomplished in the same manner as with integer values. However, care must be taken to align the radix point properly. As with integers, converting from hexadecimal or octal to binary is accomplished by replacing each hex or octal digit with the corresponding binary digits from the appropriate table shown in Fig. 1.3.

For example, to convert $\mathbf{5AC.43B_{16}}$ to binary, we just replace "5" with "0101," replace "A" with "1010," replace "C" with "1100," replace "4" with "0100," replace "3" with "0011," replace "B" with "1011," So, using the table, we can immediately see that $\mathbf{5AC.43B_{16} = 010110101100.010000111011_2}$. This method works exactly the same for converting from octal to binary, except that it uses the table on the right side of Fig. 1.3.

Converting fractional numbers from binary to hexadecimal or octal is also very easy using the tables. The procedure is to split the binary string into groups of bits, working outwards from the radix point, then replace each group with its hexadecimal or octal equivalent. For example, to convert 01110010.1010111_2 to hexadecimal, just divide the number into groups of four bits, starting at the radix point and working outwards in both directions. It may be necessary to pad with zeroes to make a complete group on the left or right, or both. Our example is grouped as follows: $\mathbf{|0111|0010.1010|1110|_2}$. Now each group of four bits is converted to hexadecimal by looking up the corresponding hex digit in the table on the left side of Fig. 1.3. This yields $\mathbf{72.AE_{16}}$. For octal, the binary number would be grouped as follows: $\mathbf{|001|110|010.101|011|100|_2}$. Now each group of three bits is converted to octal by looking up the corresponding digit in the table on the right side of Fig. 1.3. This yields $\mathbf{162.534_8}$. Note that the conversion to octal required the addition of leading and trailing zeroes.

Read full chapter

URL: https://www.sciencedirect.com/science/article/pii/B9780128192214000158

Arithmetic and digital electronics

John Crowe, Barrie Hayes-Gill, in Introduction to Digital Electronics, 1998

2.2.3 Binary and hexadecimal

The bases of most importance in digital electronics are binary and hexadecimal (base-2 and base-16). So, it is worth looking at conversion between these as a special case.[4] The reason hexadecimal is commonplace is because the wires in digitial circuits are often grouped into *buses* of 16 for addressing solid state memory and other devices.

The first four least significant digits of a binary number encode the numbers, in base-10, from 0 to 15. This is the range covered by the least significant digit in hexadecimal. The next four binary digits allow this range to be extended to cover up to 255_{10} (by using the numbers 16_{10}, 32_{10}, 64_{10} and 128_{10}, i.e. the numbers represented by these binary digits, as appropriate). Correspondingly, the second hexadecimal digit enables numbers requiring up to $F_H = 15_{10}$ multiples of 16_{10} to be encoded. Hence, conversion from base-2 to hexadecimal can be performed by operating on blocks of four binary digits to produce the equivalent single hexadecimal digit.[5]

Example 2.7

What is $A4E2_H$ in binary?

Solution

Since A= 1010, 4=0100, E = 1110 and 2 = 0010 then $A4E2_H = 1010010011100010_2$.

Example 2.8

What is 100111110011_2 in hexadecimal?

Solution

Since 1001 = 9, 1111 = F and 0011 = 3 then $100111110011_2 = 9F3_H$.

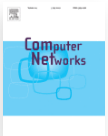
In concluding this section it is important to realise that to fully understand arithmetic operation in digital circuits, and the addressing of memory locations in computer systems, it is necessary to be able to readily convert numbers between bases-2, 10 and 16.

Read full chapter

URL: https://www.sciencedirect.com/science/article/pii/B9780340645703500046

Recommended publications

Microprocessors and Microsystems
Journal



Computer Networks
Journal

Measurement
Journal

Information Sciences
Journal