Computer Organization

Register Transfer Logic

Number System

Department of Computer Science
Missouri University of Science & Technology
hurson@mst.edu

Decimal Numbers: Base 10

- Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- Example:

  3271 =

- $(3 \mathrm{x} 10^3) + (2 \mathrm{x} 10^2) + (7 \mathrm{x} 10^1) + (1 \mathrm{x} 10^0)$

Numbers: positional notation

- Number Base B => B symbols per digit:
  - Base 10 (Decimal): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
    Base 2 (Binary): 0, 1
- Number representation:
  - $d_{31}d_{30} \ldots d_2d_1d_0$ is a 32-digit number
  - value = $d_{31} \times B^{31} + d_{30} \times B^{30} + \ldots + d_2 \times B^2 + d_1 \times B^1 + d_0 \times B^0$
- Binary: 0,1
  - $1011010 = 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
    $= 64 + 16 + 8 + 2 = 90$
  - Notice that a 7-digit binary number converts into a 2-digit decimal number
  - Which base(s) convert(s) to binary easily?

Hexadecimal Numbers: Base 16

- Example (convert hex to decimal):
  - B28F0DD = $(B \times 16^6) + (2 \times 16^5) + (8 \times 16^4) + (F \times 16^3) + (0 \times 16^2) + (D \times 16^1) + (D \times 16^0)$

    $= (11 \times 16^6) + (2 \times 16^5) + (8 \times 16^4) + (15 \times 16^3) + (0 \times 16^2) + (13 \times 16^1) + (13 \times 16^0)$

    = 187232477 decimal
- Notice that a 7-digit hex number is converted to a 9-digit decimal number

Decimal vs. Hexadecimal vs. Binary

- Examples:
- 1010 1100 0101 (binary)
  = AC5 (hex)
- 10111 (binary)
  = 0001 0111 (binary)
  = 17 (hex)
- 3F9(hex)
  = 0011 1111 1001 (binary)

00	0	0000
01	1	0001
02	2	0010
03	3	0011
04	4	0100
05	5	0101
06	6	0110
07	7	0111
08	8	1000
09	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Hex to Binary Conversion

- HEX is a more compact representation of binary.
- Each hex digit represents 16 decimal values.
- Four binary digits represent 16 decimal values.
- Therefore, each hex digit can replace four binary digits.
- Example:

0011	1011	1001	1010	1100	1010	0000	0000	binary
3	b	9	a	c	a	0	0	hex

Which Base Should We Use?

- Decimal: Great for humans; most arithmetic is done with this base.
- Binary: This is what computers use, so get used to them. Become familiar with how to do basic arithmetic with them (+,-,*,/).
- Hex: Terrible for arithmetic; but if we are looking at long strings of binary numbers, it is much easier to convert them to hex and look at four bits at a time.

representations of numbers?

- Everything we can do with decimal numbers.
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Comparison
- Example: 10 + 7 = 17

```
     1  1
     1  0  1  0
+    0  1  1  1
-------------------------
  1  0  0  0  1
```

  - so simple to add in binary that we can build circuits to do it
  - subtraction also just as in decimal

Binary	Decimal	Octal	3-Bit String	Hexadecimal	4-Bit String
0	0	0	000	0	0000
1	1	1	001	1	0001
10	2	2	010	2	0010
11	3	3	011	3	0011
100	4	4	100	4	0100
101	5	5	101	5	0101
110	6	6	110	6	0110
111	7	7	111	7	0111
1000	8	10	—	8	1000
1001	9	11	—	9	1001
1010	10	12	—	A	1010
1011	11	13	—	B	1011
1100	12	14	—	C	1100
1101	13	15	—	D	1101
1110	14	16	—	E	1110
1111	15	17	—	F	1111

Table 2-1

Binary, decimal, octal, and hexadecimal numbers.

Conversion	Method	Example
Binary to		
Octal	Substitution	$10111011001_2 = 10\ 111\ 011\ 001_2 = 2731_8$
Hexadecimal	Substitution	$10111011001_2 = 101\ 1101\ 1001_2 = 5D9_{16}$
Decimal	Summation	$10111011001_2 = 1 \cdot 1024 + 0 \cdot 512 + 1 \cdot 256 + 1 \cdot 128 + 1 \cdot 64$ $+ 0 \cdot 32 + 1 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 1497_{10}$
Octal to		
Binary	Substitution	$1234_8 = 001\ 010\ 011\ 100_2$
Hexadecimal	Substitution	$1234_8 = 001\ 010\ 011\ 100_2 = 0010\ 1001\ 1100_2 = 29C_{16}$
Decimal	Summation	$1234_8 = 1 \cdot 512 + 2 \cdot 64 + 3 \cdot 8 + 4 \cdot 1 = 668_{10}$
Hexadecimal to		
Binary	Substitution	$C0DE_{16} = 1100\ 0000\ 1101\ 1110_2$
Octal	Substitution	$C0DE_{16} = 1100\ 0000\ 1101\ 1110_2 = 1\ 100\ 000\ 011\ 011\ 110_2 = 140336_8$
Decimal	Summation	$C0DE_{16} = 12 \cdot 4096 + 0 \cdot 256 + 13 \cdot 16 + 14 \cdot 1 = 49374_{10}$
Decimal to		
Binary	Division	$108_{10} \div 2 = 54$ remainder 0 (LSB) $\div 2 = 27$ remainder 0 $\div 2 = 13$ remainder 1 $\div 2 = 6$ remainder 1 $\div 2 = 3$ remainder 0 $\div 2 = 1$ remainder 1 $\div 2 = 0$ remainder 1 (MSB) $108_{10} = 1101100_2$
Octal	Division	$108_{10} \div 8 = 13$ remainder 4 (least significant digit) $\div 8 = 1$ remainder 5 $\div 8 = 0$ remainder 1 (most significant digit) $108_{10} = 154_8$
Hexadecimal	Division	$108_{10} \div 16 = 6$ remainder 12 (least significant digit) $\div 16 = 0$ remainder 6 (most significant digit) $108_{10} = 6C_{16}$

Table 2-2

Conversion methods for common radices.

Signed-magnitude representation

- In decimal: +98, -10, +0, -0
- In binary, the Most Significant Bit (MSB) (leftmost bit) is dedicated as the sign bit.
  - MSB = 0 for positive numbers
  - MSB = 1 for negative numbers
  - 8-bit examples: 01010101 = + $85_{(10)}$
    11010101 = - $85_{(10)}$
  - Range: $-(2^{n-1} – 1)$ through $+(2^{n-1} – 1)$
    - With 8 bits: -127 through +127
  - Two representations of zero

Complement number systems

- Assumptions are the following:
- Fixed number of digits: n
- Radix is r
- Integers of form: $D = d_{n-1}\, d_{n-2} \ldots d_1\, d_0$
- If the result of an operation produces a number that needs more than n digits, we discard the higher-order digits
- If D is complemented twice, the result is D
  - $-(-D) = D$

Radix-complement notation

- Complement of n-digit D is $-D = r^n - D$
- If $1 \le D \le 2^n - 1$, then $1 \le -D \le 2^n - 1$
- $0' = 2^n$, which is $n+1$ bits long: 100000000
  - Per convention, we discard the MSB
  - Results in only one representation of 0

2's-complement notation

- $-D = 2^n - D = ((2^n-1)-D)+1$
- 2^n-1 has the form 11111111
  - 1-1 = 0; 1-0 = 1 toggles each bit
- Toggle every bit to get $((2^n-1)-D)$
- Add 1 to result to get 2's complement

2's-complement notation

- A number is negative iff its MSB is 1
- When converting to decimal, everything is the same, except weight of MSB for a negative number is $-(2^{n-1})$ instead of $+(2^{n-1})$

- Range: $-(2^{n-1})$ through $+(2^{n-1} - 1)$
  - For 8 bits: -128 through 127

Examples

- $85_{(10)}$ = 01010101; toggle bits:

  10101010
  $\underline{\qquad\quad +1}$ add 1;
  10101011 = - $85_{(10)}$

- Check: -128 + 32 + 8 + 2 + 1 = -85
- $-99_{(10)}$ = 10011101; toggle bits:

  01100010
  $\underline{\qquad\quad +1}$; add 1
  01100011 = $99_{(10)}$

- Check: 64 + 32 + 2 + 1 = 99

2's complement addition

- Just like decimal, but per convention, ignore carry out of MSB
- Result will be correct unless range is exceeded (overflow)
- Overflow only happens when two numbers being added have the same sign

2's complement addition

- Recall that range for 8 bits: -128 through 127

$$
\begin{array}{l}
01111111 = 127_{(10)} \\
\underline{+00000001\ } = \\
10000000 = -128_{(10)}
\end{array}
$$
incorrect result

We expected 128, which cannot be represented with 8 digits (out of range)

Overflow

$$
\begin{array}{r l}
10000000 & = -128_{(10} \\
\underline{+\ 11111111\ } & = -1_{(10)} \\
1\ 01111111 & = 127_{(10)}
\end{array}
$$

incorrect result

We expected -129, which cannot be represented with 8 digits (out of range)

Overflow

- Check for overflow
  - Do both addends have the same sign?
  - If no, overflow is impossible.
  - If yes, does the sum have the same sign as them? If it does not, then overflow.
- Other method:
  - If carry into MSB $\neq$ carry out of MSB; then overflow

2's complement subtraction

- Turn it into an addition by negating the subtrahend

  (+4) – (+3) = (+4) + (-3) = +1

```
   0100            1
+  1101             0100
 1 0001          +  1100
                  1 0001
```

  (+3) – (+4) = (+3) + (-4) = -1

```
   0011            1
+ 1100             0011
  1111          +  1011
                   1111
```

2's complement subtraction

- Shortcut: To negate the second number, we toggle the bits and add 1 to the result. Since we will eventually be adding two numbers, we can combine this addition with the final one.
- Toggle bits of the second number (minuend), and add to the first, with a carry-in of 1.

Overflow detection

- For overflow detection, check the signs of the two numbers being added, and the sign of the result. This is exactly the same as before.
- Or: If carry into MSB $\neq$ carry out of MSB; then overflow
- (-8)-(+1) = -9 overflow is expected

$$\begin{array}{r} 1000 \\ \underline{+\ 1111} \\ 1\ 0111 \end{array}$$

2's complement of a non-integer

- Definition is the same as for integers:
  - Complement of n-digit D is $-D = r^n - D$
  - Here, n refers to the number of digits to the left of the decimal point (integer digits)
- Example: $D = 010.11$
  - Number of integer digits = $n = 3$
  - $-D = 2^n - D = 2^3 - D = 1000 - 010.11$

$$\begin{array}{r} 1000.00 \\ +\ \ 010.11 \\ \hline 101.01 \end{array}$$

Decimal codes

- Binary numbers are most appropriate for internal operations of a computer.
- External interfaces (I/O) may read or display decimal, for the benefit of humans.
- Logical conclusion is that we need an easy way of representing decimal numbers with bits.
- A coded representation of the 10 digits of the decimal number system (0-9) is known as a binary-coded decimal (BCD) representation.

Some definitions

- **Code**: a set of *n*-bit strings, where each string represents a different number, letter, or other thing.
- **Code word**: one such *n*-bit string.
- A legal, or valid code word, is one that is actually used to represent something.
  - With n bits, we can have 2^n code words, but not all of these are necessarily used to represent something. Some of them may be unused.
  - Example: A BCD code needs to represent 10 digits (0-9)
    - At least 4 bits are needed to represent 10 things
    - 4 bits give us 16 possible code words
    - 10 of these 16 are legal code words
    - 6 are unused

Binary coded decimal (BCD)

- Most natural representation is to use 4-bit strings, where each decimal digit is represented its binary representation
  - 0000 through 1001 is used to represent the decimal digits 0 through 9, respectively.
  - This is the 8421 BCD scheme, which is a weighted code.
- To convert from decimal to BCD, replace each decimal digit with its BCD 4-bit string.

Binary coded decimal (BCD)

- Keep in mind that this BCD number is NOT the same as you would get if converting decimal to binary the usual way.
- Example: BCD string for 16 is 0001 0110.
  Binary equivalent of 16 is 0001 0000.
- 2 BCD digits (one byte) can represent 0 through 99.
- A normal byte can represent 0 to 255 (unsigned), or -128 to 127 (signed).
- We will not discuss BCD representation of signed numbers.
- We may come back to BCD arithmetic later in the course.

Unit-distance codes

- Useful for when an analog quantity needs to be converted to digital.
- Only one bit can change as successive integers are coded.
- Gray code is a common example.

4-bit Gray code

Decimal number	Gray code
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

Why is it useful?

- Assume that the position of a shaft, which is an analog quantity, needs to be digitally represented.
- A positional encoder wheel is attached to the shaft.
- Accuracy provided by 4 binary digits is sufficient.

Alphanumeric codes

- Alphabetic information also needs to be handled by digital systems.
- Need to represent letters of the alphabet in upper and lowercase, numbers, punctuation marks, symbols such as $ and @, and control operations such as Backspace and Carriage Return.
- The best known alphanumeric code is the 7-bit American Standard Code for Information Exchange (ASCII).
- A more recent code, the Unicode Standard, uses 16-bit strings and codes characters from foreign languages as well. Also includes codes for math symbols, etc.