01- Number systems & more

Ahmad Abdullah, PhD
abdullah@uvic.ca
https://web.uvic.ca/~abdullah/csc230

Lectures: MR 10:00 – 11:20 am
Location: ECS 125

Numbers systems (and more)

- Review of basic concepts (i.e., positional number)
- Conversion between bases (positive numbers)
- Some terminology involving binary numbers
- Representation of negative numbers in binary
- Canonical set of bit operations

Consider...

- A typical base 10 number...

7409.01

- We've seen this kind of number in the past
- Positional notation is very familiar to us.

The “meaning” of 7409.01

```
7    4    0    9 . 0    1
```

the sum of...

7 x 1000	4 x 100	0 x 10	9 x 1	0 x .1	1 x .01
7×10^3	4×10^2	0×10^1	9×10^0	0×10^{-1}	1×10^{-2}

Abstracting a little bit...

$B = 10$

$d_3=7$ $d_2=4$ $d_1=0$ $d_0=9$ $d_{-1}=0$ $d_{-2}=1$

7 4 0 9 . 0 1

the sum of...

$d_3 \times B^3$ $d_2 \times B^2$ $d_1 \times B^1$ $d_0 \times B^0$ $d_{-1} \times B^{-1}$ $d_{-2} \times B^{-2}$

Weighted positional representation

- We can denote like this:

the sum of...

$d_3 \times B^3$	$d_2 \times B^2$	$d_1 \times B^1$	$d_0 \times B^0$	$d_{-1} \times B^{-1}$	$d_{-2} \times B^{-2}$

- And can also instead write:

$$\sum_{-m}^{n-1} d_i B^i$$

  - *where* ***n*** is the number of digits to **left** of the **radix point**
  - and ***m*** is the number of digits to **right** of the **radix point**

Beyond base 10

- Binary (base 2)
  - Digit values are 0 and 1
- Octal (base 8)
  - Digit values are 0, 1, 2, 3, 4, 5, 6 and 7
- Hexadecimal (base 16)
  - Digit values are 0 through 9, then A, B, C, D, E and F
- Weighted positional representation formula...
  - ... suggests an approach to converting from these three bases into decimal
- In what follows, we'll ignore fractional numbers
  - This is mean to simplify explanations
  - We'll look at fractional & floating point numbers later in the course

Binary

B = 2

$d_6=1$	$d_5=1$	$d_4=0$	$d_3=1$	$d_2=0$	$d_1=1$	$d_0=1$
1	**1**	**0**	**1**	**0**	**1**	**1**

the sum of...

$d_6 \times B^6$	$d_5 \times B^5$	$d_4 \times B^4$	$d_3 \times B^3$	$d_2 \times B^2$	$d_1 \times B^1$	$d_0 \times B^0$
1 x 64	1 x 32	0 x 16	1 x 8	0 x 4	1 x 2	1 x 1

== 107

Octal

B = 8

d_4=7 d_3=2 d_2=1 d_1=5 d_0=5

7 2 1 5 5

the sum of...

$d_4 \times B^4$ $d_3 \times B^3$ $d_2 \times B^2$ $d_1 \times B^1$ $d_0 \times B^0$

7 x 4096 2 x 512 1 x 64 5 x 8 5 x 1

== 29805

Hexadecimal

- Base 16 numbers must use additional symbols in addition to 0 through 9
  - Hex digits can be either upper or lower case
  - In practical terms, we try to avoid mixing cases (i.e., either use all lower-case, or all upper-case)

decimal	hexadecimal (hex)
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

Hexadecimal

B = 16

d_3=D d_2=0 d_1=A d_0=7

D 0 A 7

the sum of...

$d_3 \times B^3$ $d_2 \times B^2$ $d_1 \times B^1$ $d_0 \times B^0$

13 x 4096 0 x 256 10 x 16 7 x 1

== 53415

Notational conventions

- Binary numbers normally represented:
  - with a prefix: **0b** or **%**
  - or with a suffix: **b**
- Hex numbers normally represented:
  - with a prefix: **0x** or **$**
  - or with a suffix: **h**
- Octal numbers normally represented:
  - with a prefix: **0** or **0o** (“zero-oh”)
  - or with a suffix: **o**
- Our previous three examples: 0b1101011, 072155, 0xD0A7 (or 0xd0a7)
- Decimal numbers (i.e., base 10) normally have no special prefix or suffix

Conversions

- In our work we will commonly need to convert numbers between bases
- From binary, octal, hexadecimal to decimal:
  - As we have seen, this is a straightforward application of the weighted-positional representation formula
- Between binary, octal, hexadecimal (i.e., from binary to hexadecimal)
  - use standard shortcuts
- From decimal to binary, octal or hexadecimal
  - use repeated division

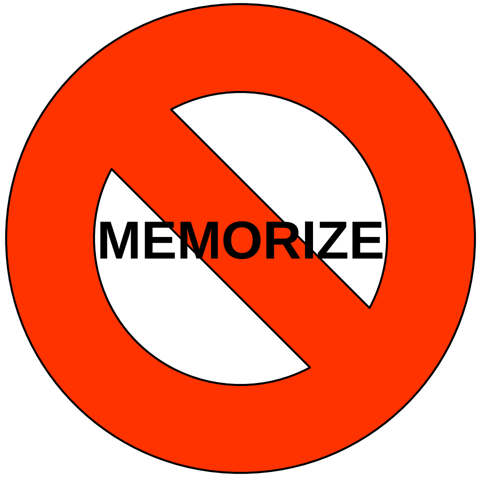
Between binary, octal & hex

- From binary to octal and hexadecimal
- Take the binary sequence...
- ... and starting from the right ...
- ... arrange into groups of three (for octal) or four (for hexadecimal)
- Finally convert each group to its corresponding digit (octal or hexadecimal)

Between binary, octal & hex (cont.)

Each three binary bits can be represented by one Octal digit

BIN	OCT	DEC
000	0	0
001	1	1
010	2	2
011	3	3
100	4	4
101	5	5
110	6	6
111	7	7



Each four binary bits can be represented by one Hexadecimal digit

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

Example

- Convert 0b1010111000100010
  A. to octal
  B. to hex

A. To octal: groups of (up to) three binary digits **starting from the right**
   - that is: 1 010 111 000 100 010
   - for which we use one octal digit per group: 1 2 7 0 4 2
   - and finally: 0127042

B. To hex: groups of four binary digits **starting from the right**
   - that is: 1010 1110 0010 0010
   - for which we use on hex digit per group: A E 2 2
   - and finally: 0xAE22

Between binary, octal & hex

- From octal to binary: trivial
- From hex to binary: trivial
- From octal to hex, or hex to octal:
  - convert to binary
  - count out binary digits from right as appropriate (in groups of three or four as needed)
  - rewrite groups into appropriate digits

Conversions from base 10 to other bases

- This is a bit more involved...
- ... but still relatively straightforward
- To convert from a **positive** base 10 number to base B, we use the **repeated division algorithm**
  - Perform repeated divisions of base B into number to be converted
  - Each division yields a **quotient** (passed on to the next step)...
  - ... and a **remainder** (which is a number from 0 up to but not including B) which is the next digit in the base B representation
  - Digits are produced from right (**least significant**) to the left (**most significant**)
  - Keep dividing until quotient is zero
  - Leading zeros may be added to final result as needed

Example

- Convert 232 (base 10) to octal (base 8)

pass 1	232 / 8 = 29	232 % 8 = 0	d_0 = 0
pass 2	29 / 8 = 3	29 % 8 = 5	d_1 = 5
pass 3	3 / 8 = 0	3 % 8 = 3	d_2 = 3
stop			

result is 350

Example

- Convert 35 (base 10) to binary (base 2)

pass 1	35 / 2 = 17	35 % 2 = 1	$d_0 = 1$
pass 2	17 / 2 = 8	17 % 2 = 1	$d_1 = 1$
pass 3	8 / 2 = 4	8 % 2 = 0	$d_2 = 0$
pass 3	4 / 2 = 2	4 % 2 = 0	$d_3 = 0$
pass 4	2 / 2 = 1	2 % 2 = 0	$d_4 = 0$
pass 5	1 / 2 = 0	1 % 2 = 1	$d_5 = 1$
stop		result is 0b100011	

Binary-number terminology (MIPS)

- bit: a binary digit
- special names for fixed-length bit sequences
  - four bits: nibble (or nybble)
  - eight bits: byte
  - (for AVR) 16 bits: word
  - (for AVR) 32 bits: double word
- least-significant bit: right-most bit
- most-significant bit: left-most bit

An aside...

- Converting binary / octal / hex into decimal can be expensive
  - equation involves to repeated use of exponentiation
- Example:

$$
\begin{aligned}
07536 = \ & 7(8^3) + 5(8^2) + 3(8^1) + 6(8^0) \\
&= 7 * 8 * 8 * 8 \ + 5 * 8 * 8 + 3 * 8 + 6 \\
&= 3934
\end{aligned}
$$

six multiplications
three additions

An aside...

- We could just pre-compute the exponentiated terms
  - and keep them in a table...
  - ... but this is a possibly error-prone approach
- Alternative: ***Horner’s Algorithm***
  - Also called “*Horner’s Method*”
  - Iterative algorithm used to reduce number of arithmetic operations
  - uses a factored form of the positional polynomial
- Insight: equation is re-written so that higher-order terms are “deeper inside the onion”

Using polynomial evaluation...

$B = 8$

$d_3=7$ $d_2=5$ $d_1=3$ $d_0=6$

7 5 3 6

the sum of...

$d_3 \times B^3$ $d_2 \times B^2$ $d_1 \times B^1$ $d_0 \times B^0$

7×512 5×64 3×8 6×1

== 3934

Using Horner's algorithm...

$B = 8$

$d_3=7$ $d_2=5$ $d_1=3$ $d_0=6$

7 5 3 6

$b_3 = d_3$	b_3	**7**
$b_2 = d_2 + b_3 \times B$	b_2	**61**
$b_1 = d_1 + b_2 \times B$	b_1	**491**
$b_0 = d_0 + b_1 \times B$	b_0	**3934**

Answer is equal to $\mathbf{b_0}$

three multiplications
three additions

For more: http://bit.ly/2eGu7M0

Conversion summary

- Decimal ➔ binary / octal / hex:
  - use repeated division algorithm
- Binary / octal / hex ➔ decimal:
  - use weighted position representation equation
  - also referred to as the polynomial evaluation algorithm
  - Could also instead use Horner’s algorithm
- Binary / octal / hex ➔ binary / octal hex
  - Either write out octal / hex digits into binary (trivial), or...
  - ... write binary version and group bits into groups of three or four bits
- Positive integer ➔ negative integer
  - First find binary representation of positive integer
  - Use change-sign rule to convert into two’s complement
  - Change-sign rule also works from negative to positive

Benefits of octal and hex notation

- Although ultimately bits are used at the most fundamental organizational level of a computer...
- ... they can be unwieldly as just literals 1s and 0s.
- Therefore we use hexadecimal more often to represent numbers made of bits
- Example:
  - value of a byte (8 bits) can be expressed as two hex digits (e.g., 0x81)
  - value of a word can be expressed as four hex digits (e.g., 0xFACE)
  - value of a double word can be expressed as eight hex digits (e.g., 0x55f6077c)

Any Questions?