CS 241 Winter 2017

Foundations of Sequential Programs

Kevin Lanctot

Much of this material comes from, or is based on, lecture notes by Brad Lushman and lectures slides by Troy Vasiga.

Topic 1 – Representing Data

Key Ideas

- Understand Binary, Decimal and *Two's Complement* and *Hexadecimal* representations of integers
- Data representation: bit, nibble, byte and word
- Representing Characters: *ASCII, Unicode*

References

- CO&D sections *2.4* and 2.9

Binary Number System - Review

The Decimal Number System

- *Humans* often represent numbers using combinations of 10 different symbols {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}.
- Called *base 10*, *radix 10* or the *decimal system.*

The Binary Number System

- *Computers* represent numbers using combinations of 2 different symbols {0, 1}.
- Called *base 2*, *radix 2* or the *binary system.*

The Hexadecimal Number System

- *Compromise* easier to use than binary but harder than decimal
- Represent numbers using combinations of 16 different symbols {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f}.

Binary Number System - Review

Why Do Computers Use Binary?

- Originally used base 10.
- Led to *complicated designs* in the age of vacuum tubes.
- Have to be able to detect 10 different states.
- Konrad Zuse's mechanical computer Z1 (developed 1935 – 1938) was the first to use binary numbers.
- It led to a *much simpler design.*
- Bonus: also a *more reliable* way to ...
  - store information over time, e.g. hard drive
  - transmit information over distance, e.g. network

Binary Number System - Review

Radix Representation – Base 10

$50320_{\mathbf{dec}} = 5 \cdot 10000 + 0 \cdot 1000 + 3 \cdot 100 + 2 \cdot 10 + 0 \cdot 1$

$50320_{\mathbf{dec}} = 5 \cdot 10^4 + 0 \cdot 10^3 + 3 \cdot 10^2 + 2 \cdot 10^1 + 0 \cdot 10^0$

Radix Representation – Base 2

$10110_{\mathbf{bin}} = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$

$10110_{\mathbf{bin}} = 1 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1$

$10110_{\mathbf{bin}} = 22_{\mathbf{dec}}$

Binary Addition - Review

- similar to addition of decimals
- add digits from right to left
- include carry
- with these basic rules…

$$\begin{array}{r} 0 \\ +\,0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ +\,1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array}$$

can calculate any sum

$$\begin{array}{r} {\scriptstyle 1\ 1\ 1\ \ } \\ 101_2 \\ +\ \ 011_2 \\ \hline 1000_2 \end{array} \qquad \begin{array}{r} 5_{10} \\ +3_{10} \\ \hline 8_{10} \end{array}$$

Two Issues

1. fixed width (*i.e. n*-bit representation) means the possibility of *overflow*: answer can take more than *n* bits to represent
2. how to represent negative numbers?

Negative Numbers: Attempt 1

Issues with Sign Extension

- fixed width n-bit representation
  - *most significant bit (MSB)*: left-most (highest value)
  - *least significant bit (LSB)*: right-most (lowest value)
- Attempt 1: *sign extension*
  - i.e. treat the MSB as the sign
  - 0 means positive, 1 means negative
  - e.g. 0001_2 is $+1_{10}$, 1001_2 is -1_{10} (in four bit case)
- Problem

  two ways to represent zero: 0000 and 1000

Negative Numbers: Attempt 2

Two's Complement

- *goal:* get rid of this pesky two 0's issue
- *to represent a negative number: invert the bits and add 1*

		invert		add 1		
0_{10}:	0000	$\rightarrow$	1111	$\rightarrow$	0000	0_{10}
1_{10}:	0001	$\rightarrow$	1110	$\rightarrow$	1111	-1_{10}
4_{10}:	0100	$\rightarrow$	1011	$\rightarrow$	1100	-4_{10}
7_{10}:	0111	$\rightarrow$	1000	$\rightarrow$	1001	-7_{10}

- now have a single zero 0000 !
- bonus: easier to implement in hardware

Negative Numbers: Attempt 2

	2's Comp
4_{10}	0100
3_{10}	0011
2_{10}	0010
1	0001
0	0000
-1	1111
-2_{10}	1110
-3_{10}	1101
-4_{10}	1100

Why Does Twos Complement Work?

- it is *modular arithmetic*

  $-1 \equiv 15 \bmod 16$
- comp(0001) + 0001

  = 1110 + 0001

  = 1111 = 15_{10}
- (comp(0001) + 1) + 0001

  = 1111 + 0001

  = 0000 (overflow ignored)

  = 0

Negative Numbers: Attempt 2

Example 1

`0101` 5

`1010` 5 comp

`1011` - 5 in 2's comp

~~1~~ 1 1 0

0110	6
+1011	+(-5)
~~1~~0001	1

ignore last carry bit

Example 2

`0111` 7

`1000` 7 comp

`1001` - 7 in 2's comp

~~0~~ 0 0 0

0110	6
+1001	+(-7)
~~0~~1111	-1

ignore last carry bit

Negative Numbers: Overflow

Example 1a

(5 + 3) -1 = 8 - 1 = 7

```
~~0~~ 1 1 1
  0101        5
+0011        +3
-----        --
  1001       -7

~~1~~ 1 1 1
  1001       -7
+1111     +(-1)
-----     -----
  1000       -8
```

Example 1b

5 + (3 -1) = 5 + 2 = 7

```
~~1~~ 1 1 1
  0011        3
+1111     +(-1)
-----     -----
  0010        2

~~0~~ 0 0 0
  0101        5
+0010        +2
-----        --
  0111        7
```

When adding 5 + 3, there is *overflow* in Example 1a.

Don't ignore 2^{nd} *last carry bit.*

Hexadecimal Numbers

The Problem with Humans using Binary Numbers

- e.g. 10110100011000010010111000111111
- long strings of binary digits are hard to read and remember
- easy to make a mistake reading or typing them
- *group them into groups of 4, convert each group*
- 1011 0100 0110 0001 0010 1110 0011 1111
- 11 4 6 1 2 14 3 15
- introduce 6 new symbols, {a, b, c, d, e, f}, to represent the two digit numbers {10, 11, 12, 13, 14, 15}

Hexadecimal Numbers

The Problem with Humans using Binary Numbers

- 1011 0100 0110 0001 0010 1110 0011 1111

  is now represented as ...

  b 4 6 1 2 e 3 f

So the binary number ...

10110100011000010010111000111111

can be written as ...

b4612e3f or b4612E3F or 0xb4612e3f

(*i.e. caps or small letters, sometimes with a leading 0x ...*)

Hexadecimal Numbers

Table to Convert between Binary and Hexadecimal

$0000_{bin} = 0_{hex}$	$1000_{bin} = 8_{hex}$
$0001_{bin} = 1_{hex}$	$1001_{bin} = 9_{hex}$
$0010_{bin} = 2_{hex}$	$1010_{bin} = a_{hex}$
$0011_{bin} = 3_{hex}$	$1011_{bin} = b_{hex}$
$0100_{bin} = 4_{hex}$	$1100_{bin} = c_{hex}$
$0101_{bin} = 5_{hex}$	$1101_{bin} = d_{hex}$
$0110_{bin} = 6_{hex}$	$1110_{bin} = e_{hex}$
$0111_{bin} = 7_{hex}$	$1111_{bin} = f_{hex}$

Who Uses What

- *Humans* use and represent numbers in decimal.
- *Computers* use and represent numbers in binary.
- People! Computers! Why can't we all just get along?
- Compromise position
  - When looking at the *low level workings* of a computer, programmers often use hexadecimal.
  - When talking about *memory locations* (pointers, references) programmers often use hexadecimal.
  - *It is easy to covert* between hexadecimal and binary representation.

Data Representation

How to Interpret Data

- *Interpretation is in the eye of the beholder.*
- What does the following represent?
  01110111011010000111100100111111
- It could be a number, a character, machine instruction, part of an audio clip, a picture or a video, etc.
- Storage devices just store 0's and 1's.
- Digital circuits just process 0's and 1's.
- We must (somehow) keep track of what the data means, i.e. *context*.

Data Representation

Bit

- a single 1 or 0 (voltage level, magnetic orientation)

Nibble

- 1 nibble = 1 hexadecimal digit = 4 bits

Byte

- 1 byte = 2 hexadecimal digits = 8 bits
- useful range to represent a English character

Data Representation

Word

- It depends on the processor:
  - for 32-bit *architecture*: 1 word = 4 bytes = 32 bits,
  - for 64-bit architecture: 1 word = 8 bytes = 64 bits.
- For CS 241, we'll used a 32-bit architecture
  - i.e. the processor can transfer 32 bits in parallel (at the same time).
- As more transistors can fit on a chip, it increases the circuit capacity.
- Individual bytes are still accessible from memory.

Representing Data: ASCII

ASCII

- American Standard Code for Information Interchange

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20		!	“	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Representing Data: ASCII

ASCII Cautions

- *ASCII inherited much from Baudot (meant for teletypes)* including control characters such as SOH (start of header) STX (start of text) ETX (end of text), EOT (end of transmission), LF (line feed), CR (carriage return)
- Difference OSs interpret some of them differently
- To end a line in ...
  - Linux / UNIX: "\n"
  - MS Windows text editors: "\r\n"
  - Macs up to OS-9 "\r"

Representing Data: Multilingual Codes

Unicode

- different countries had different codes
- *goal: create a standard for most languages*
- Unicode = *Uni*fication *Code*
- currently 110,000 characters from 100 scripts
  - English, French, Spanish, Italian, Portuguese, etc., use a Roman script.
  - Russian, Ukrainian, Serbian, etc., use a Cyrillic script
  - Arabic, Persian, Pashto, Kurdish, Urdu , etc., use an Arabic script.