Computer Systems Architecture

http://cs.nott.ac.uk/~txa/g51csa/

Thorsten Altenkirch

School of Computer Science and IT
University of Nottingham

Lecture 01: Bits, bytes and numbers

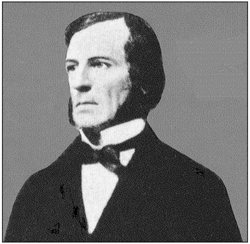
Bits

- Bit = **Bi**nary Digi**t**
- Binary means two (states):
  - 1 or 0
  - True or False
  - On or Off
  - Yes or No
  - High or Low
  - $\mathtt{tt}$ or $\mathtt{ff}$
  - $\top$ or $\bot$
- Contrast with decimal digits:
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

The University of Nottingham

What can we do with truth values?

George Boole (1815–1864)



- Indicate one of two choices
- Boolean logic
- e.g. Java: `bool`, Haskell: Bool
- *NOT*, *AND*, *OR* and so on. . .

Boolean operators

NOT

A	$\neg A$
0	1
1	0

OR

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

AND

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

- ... *XOR*, *NAND*, *NOR* and more! But note e.g.

$$\neg(A \vee B) = \neg A \wedge \neg B$$

- *NOT* and *AND* are sufficient.

More Boolean operators

- With only *NOT*, *AND* and *OR*, fill in the blanks

NAND

A	B	$\neg(A \wedge B)$
0	0	1
0	1	1
1	0	1
1	1	0

XOR

A	B	$(\neg A \wedge B) \vee (A \wedge \neg B)$
0	0	0
0	1	1
1	0	1
1	1	0

The University of Nottingham

Addition in binary?

Addition

A	B	$A+B$
0	0	0
0	1	1
1	0	1
1	1	???

- Decimal: $9+1=10$
- Binary: $1+1=10$

Where necessary, use subscript to indicate base: e.g. $10_2 \equiv 2_{10}$.

Adding, one at a time (or... counting)

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101

Decimal	Binary
6	110
7	111
8	1000
9	1001
10	1010
11	1011

Decimal	Binary
12	1100
13	1101
14	1110
15	1111
16	10000
17	10001

- Note the 1 in binary behaves like 9 in decimal
- Result of $1 + 1$ is 0, carrying a 1 to the next digit

Binary to decimal

- Each binary digit corresponds to a power of 2:

Place	7^{th}	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Weight	2^7 $= 128$	2^6 $= 64$	2^5 $= 32$	2^4 $= 16$	2^3 $= 8$	2^2 $= 4$	2^1 $= 2$	2^0 $= 1$

- Where the digit is 1, we add the corresponding weight.
- For example, 1100 1010 binary is 202 decimal, since

$$\begin{aligned} 1100\,1010_2 &= 1 \times 128 + 1 \times 64 + 0 \times 32 + 0 \times 16 \\ &\quad + 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1 \\ &= 128 + 64 + 8 + 2 = 202_{10} \end{aligned}$$

The University of Nottingham

Decimal to binary

- Repeatedly divide by 2, until we reach 0
- The right/left-most binary digit is the first/last remainder

101	Remainder
50	1
25	0
12	1
6	0
3	0
1	1
0	1

- Check: $1100101_2 = 101_{10}$
- Why does this work?

The University of Nottingham

Hexadecimal notation

- Hexadecimal: numbers in base 16

Decimal	0	1	2	3	4	5	6	7
Hexadecimal	0	1	2	3	4	5	6	7

Decimal	8	9	10	11	12	13	14	15
Hexadecimal	8	9	A	B	C	D	E	F

- Decimal conversion? As for binary, but 16 instead of 2!

Hexadecimal and binary

- Binary numbers often too long to write
- Each hexadecimal digit $\equiv$ 4 binary digits; a *nibble*

Hex	Bin	Hex	Bin
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

- e.g. $1100\,1010\,1111\,1110_2 = CAFE_{16}$

The University of Nottingham

Quick quiz

- Complete the following table:

Decimal	Binary	Hexadecimal
123	1111011	$7B$
229	11100101	$E5$
101	1100101	65

- Can you think of coding systems in other bases?
  - Base 3: Red Amber Green
  - Base 4: $\uparrow \rightarrow \downarrow \leftarrow$; A C G T

Bytes

- Computers work with binary numbers of *fixed size*
- Basic unit is a *byte* = 8 bits, e.g.

$$0010\,1010_2$$

- Include leading zeros to indicate size
- Each byte represents one of 2^8 distinct values
- Count from 0 to 255:

Decimal	Binary	Hexadecimal
0	0000 0000	00
1	0000 0001	01
2	0000 0010	02
255	1111 1111	*FF*

Characters

- What can we do with a byte? Store a character!
- American Standard Code for Information Interchange
- ASCII (from 1963) defines 128 codes; requires 7 bits
- Includes codes for mechanical teletypes, e.g. CR/LF
- Being US-centric, does not include £, ß, é…
- Extensions and variations:
  - ISO 8859-1: Latin, Western European
  - MacOS Cyrillic: Russian, Slavic languages
  - Windows-1256: Arabic (limited; missing glyphs)

ASCII Table

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

- Top: first hex digit
- Left: second hex digit
- '`@`' has the code 40_{16}
- '`i`' has the code 69_{16}
- Codes 00_{16} to $1F_{16}$ are *control characters*

Unicode

- Lots of *incompatible* ASCII extensions and variations
- Insufficient codes for complex CJK scripts
  - GB: Simplified Chinese (PRC)
  - Big5: Traditional Chinese (Taiwan)
  - Shift JIS: Japanese
- Unicode attempts to incorporate all languages
- UTF-8: each character requires 1 to 4 bytes
- Currently over 100,000 characters defined
- Java, Windows, XML... all use Unicode

Strings

- Sequences of characters are termed *strings*
- Stored in consecutive memory locations
- By convention, *NUL* (00_{16}) byte marks end of string
- The ASCII string "`Hello`", starting at location 0100_{16}:

Address	Byte	Character
0100	48	'H'
0101	65	'e'
0102	$6C$	'l'
0103	$6C$	'l'
0104	$6F$	'o'
0105	00	*NUL*

Words

- Amount of data computer can process in one step
- On the 32-bit MIPS, a word is 4 bytes long
- Also, must be aligned to 4 byte boundaries
- Our MIPS processor is *little endian* (c.f. *big endian*)
    - least significant byte stored first

Address	Bytes	Characters	Word
0100	48 65 6*C* 6*C*	'H' 'e' 'l' 'l'	6*C*6*C*6548
0104	6*F* 00 00 00	'o' '*NUL*' '*NUL*' '*NUL*'	0000006*F*
0108	...	...	...