## Binary Digits (bits)

- data within a computer system are stored in one of 2 physical states (hence the use of binary digits)
  - 0V and 5V
  - charge / NO charge on a transistor gate
  - ferrite core magnetised clockwise or counter clockwise
  - …
- binary digits (bits) and are represented by the values 0 and 1
- binary digits are normally grouped together so they are easier to work with
  - 4 bits = nibble or nybble — 1110
  - 8 bits = byte (or 2 nibbles) — 01101001
  - 16 bits = halfword (or 2 bytes) — 01111000 01001011
  - 32 bits = word (or 4 bytes) — 11101101 01111101 01100111 01100101

intel
16 bits = WORD
32 bits = DWORD (double word)

# BINARY NUMBERS

## Unsigned Binary Integers

- unsigned == positive integers ONLY
- converting binary to decimal

$100101_2$ base 2 (assume base 10 if not specified)

$2^5=32$ $2^4=16$ $2^3=8$ $2^2=4$ $2^1=2$ $2^0=1$

$100101_2 = 1\times2^5 + 0\times2^4 + 0\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$

$100101_2 = 32 + 0 + 0 + 4 + 0 + 1$

$100101_2 = 37$

- similar decimal calculation (base 10)

$37 = 3\times10^1 + 7\times10^0$

$403 = 4\times10^2 + 0\times10^1 + 3\times10^0$

## Converting a positive decimal integer to binary

- keep dividing by 2 until 0 and remember remainders
- convert 37 to binary

| | | |
|---|---|---|
| 2 | 37 | |
| 2 | 18 | 1 |
| 2 | 9 | 0 |
| 2 | 4 | 1 |
| 2 | 2 | 0 |
| 2 | 1 | 0 |
| | 0 | 1 |

37 ÷ 2 = 18 remainder 1

read from bottom

$37 = 0010\ 0101_2$

- what is 42 in binary? $0010\ 1010_2$
- what is 16 in binary? $0001\ 0000_2$

# BINARY NUMBERS

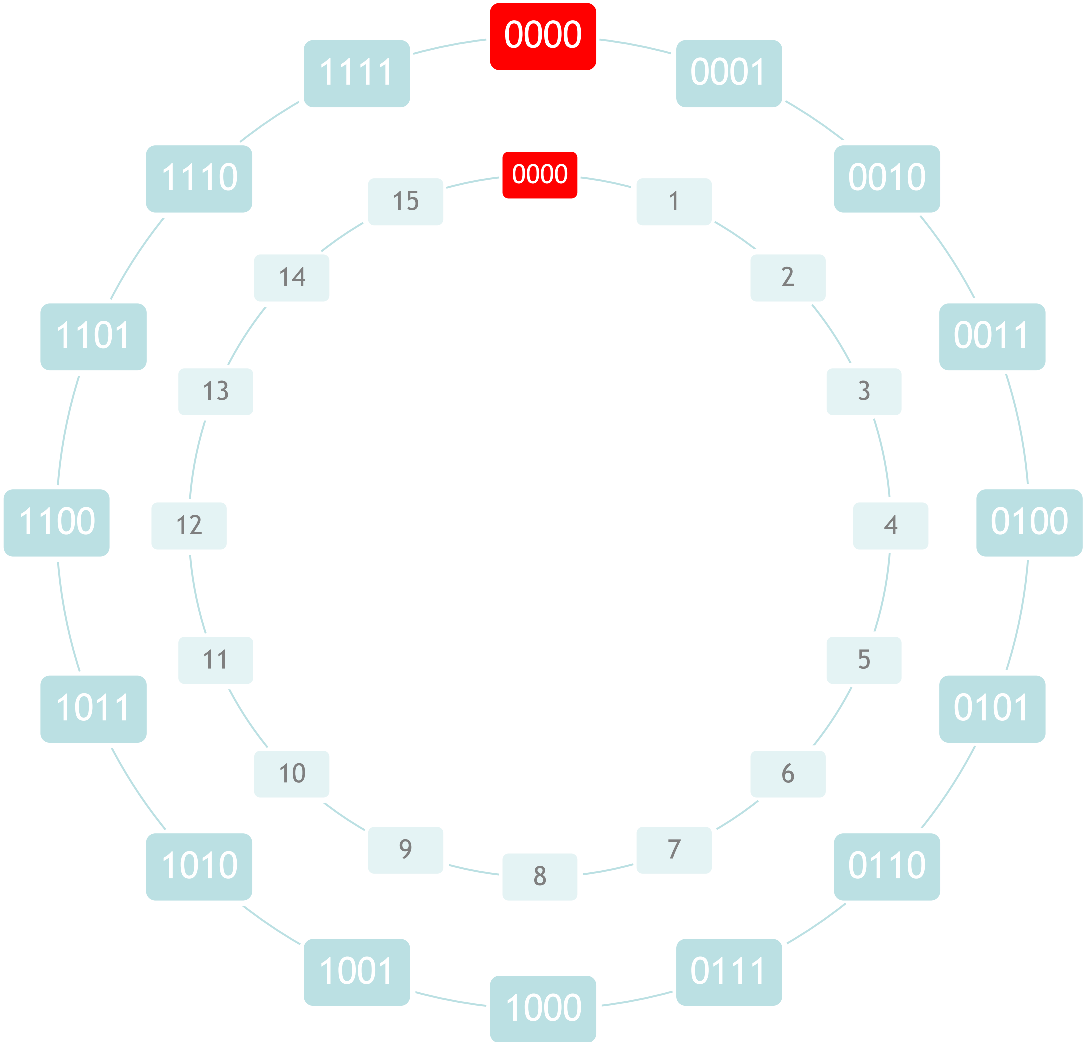
- pictorial view of a 4 bit unsigned binary integer

- 4 bit unsigned binary integer range

  $0000_2$ *to* $1111_2$
  0 *to* 15

- n bit unsigned binary integer range

  0 *to* $2^n$-1



# BINARY NUMBERS

*There are 10 types of people in the world: those who understand binary and those who don't.*

## Signed Binary Integers

- SIGNED = positive and negative integers
- 2's complement notation
- convert +5 to -5 by taking the 2's complement (invert bits and add 1)

| | |
|---|---|
| 5 | $0101_2$ |
| invert bits | $1010_2$ |
| add 1 | $0001_2$ |
| -5 | $1011_2$ |
| invert bits | $0100_2$ |
| add 1 | $0001_2$ |
| 5 | $0101_2$ |

## Signed Binary Integers

- same effect achieved by subtracting from 0 (modulo 16 in this case)

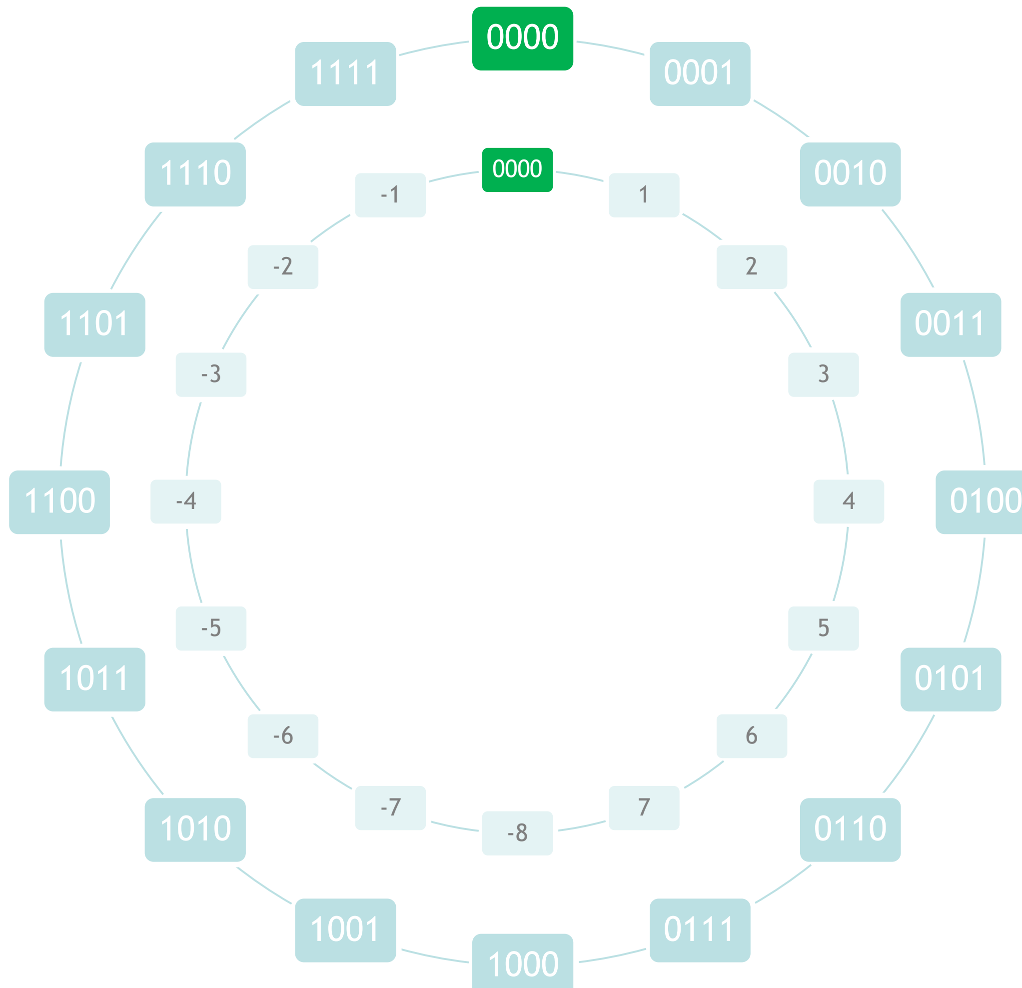
| | |
|---|---|
| zero | $0000_2$ |
| subtract 5 | $0101_2$ |
| -5 | $1011_2$ |

ignore bits beyond the first 4

- 4 bit signed binary integer
  - positive range $0000_2$ *to* $0111_2$ 0 *to* 7
  - negative range $1111_2$ *to* $1000_2$ -1 *to* -8
- **n** bit signed binary integer range: $-2^{n-1}$ *to* $2^{n-1} - 1$
- most significant bit (MSB) indicates sign (0 - positive, 1 - negative)
- note asymmetrical range – only one zero (do have a +0 and a -0)

# Binary Numbers

- pictorial view of a 4 bit signed binary integer
- if unsigned, inner ring would have values 0 to 15
- value depends whether the binary numbers are interpreted as unsigned or signed (the programmer should know!)
- 2's complement notation used because the same CPU hardware can perform unsigned and signed binary arithmetic simultaneously



# Try these

- what is -42 in binary?

| | |
|---|---|
| +42 | $0010\ 1010_2$ |
| invert bits and add 1 | $1101\ 0110_2$ |

- what is -16 in binary?

| | |
|---|---|
| +16 | $0001\ 0000_2$ |
| invert bits and add 1 | $1111\ 0000_2$ |

## Hexadecimal Notation

- base 16
- easier to handle large binary numbers by grouping 4 binary bits into a hexadecimal digit (starting at the least significant bit)
- consider the following 16 bit unsigned binary integer

$1111\ 1010\ 1100\ 1110_2 = FACE_{16}$

$F\times16^3 + A\times16^2 + C\times16^1 + E\times16^0$
$15\times16^3 + 10\times16^2 + 12\times16^1 + 14\times16^0$
$15\times4096 + 10\times256 + 12\times16 + 14\times1$
64,206

- what about?

$0000\ 1011\ 1010\ 1101_2 = 0BAD_{16}$

$0\times16^3 + 11\times16^2 + 10\times16^1 + 13\times16^0$
2,989

| BINARY | DEC | HEX |
|---|---|---|
| 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 | 10 | A |
| 1011 | 11 | B |
| 1100 | 12 | C |
| 1101 | 13 | D |
| 1110 | 14 | E |
| 1111 | 15 | F |

## Try this

- what decimal value is $FACE_{16}$ if interpreted as a 16 bit <u>signed</u> integer?
- MSB is 1, hence negative so take 2's complement by inverting bits and adding 1

| | $FACE_{16}$ |
|---|---|
| invert bits | $0531_{16}$ |
| add 1 | $0532_{16}$ |

- convert $0532_{16}$ to decimal

  $0{\times}16^3 + 5{\times}16^2 + 3{\times}16^1 + 2{\times}16^0$
  1,330

- $FACE_{16}$ when interpreted as a 16 bit <u>signed</u> integer = -1,330

| BINARY | DEC | HEX |
|---|---|---|
| 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 | 10 | A |
| 1011 | 11 | B |
| 1100 | 12 | C |
| 1101 | 13 | D |
| 1110 | 14 | E |
| 1111 | 15 | F |

## Decimal to hexadecimal conversion

- convert 20,085 to hexadecimal
- keep dividing by 16 until 0 and remember remainders

| | | |
|---|---|---|
| 16 | 20085 | |
| 16 | 1255 | 5 |
| 16 | 78 | 7 |
| 16 | 4 | E |
| | 0 | 4 |

$20085 \div 16 = 1255$ remainder 5

read from bottom

$20{,}085 = 4E75_{16}$

- convert -20,085 to hexadecimal (assume 16 bit <u>signed</u> integer)

| | |
|---|---|
| 20,085 | $4E75_{16}$ |
| invert bits | $B18A_{16}$ |
| add 1 | $0001_{16}$ |
| -20,085 | $B18B_{16}$ |

| BINARY | DEC | HEX |
|---|---|---|
| 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 | 10 | A |
| 1011 | 11 | B |
| 1100 | 12 | C |
| 1101 | 13 | D |
| 1110 | 14 | E |
| 1111 | 15 | F |

## Alternative notation

- when writing ARM Assembly Language, can use the following notion  for decimal, hexadecimal and binary integers

| | |
|---|---|
| 1000 | no prefix usually means decimal |
| 0x1000 | hexadecimal (also used by C/C++ and Java) |
| &1000 | alternative hexadecimal notation |
| 2_1000 | binary |
| n_1000 | base n eg. 8_777 is octal (base 8) |

## Adding Hexadecimal Numbers

- compute 0xA89F + 0x09A1

| | | unsigned | signed |
|---|---|---|---|
| | $A89F_{16}$ | 43,167 | -22,369 |
| + | $09A1_{16}$ | 2,465 | 2,465 |
| | $B240_{16}$ | 45,632 | -19,904 |

- remember hexadecimal/binary numbers can be interpreted as being unsigned or signed

| BINARY | DEC | HEX |
|---|---|---|
| 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 | 10 | A |
| 1011 | 11 | B |
| 1100 | 12 | C |
| 1101 | 13 | D |
| 1110 | 14 | E |
| 1111 | 15 | F |

## Subtracting Hexadecimal Numbers

- compute 0xA89F - 0x09A1

| | | unsigned | signed |
|---|---|---|---|
| | $A89F_{16}$ | 43,167 | -22,369 |
| - | $09A1_{16}$ | 2,465 | 2,465 |
| | $9EFE_{16}$ | 40,702 | -24,834 |

- remember hexadecimal/binary numbers can be interpreted as being unsigned or signed

| BINARY | DEC | HEX |
|---|---|---|
| 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 | 10 | A |
| 1011 | 11 | B |
| 1100 | 12 | C |
| 1101 | 13 | D |
| 1110 | 14 | E |
| 1111 | 15 | F |

# BINARY NUMBERS

## Real Binary Numbers

- binary point (rather than a decimal point)

- what is the value of the following binary number?

$$11.010101_2$$

$2^1=2$ $2^0=1$ $2^{-1}=0.5$ $2^{-2}=0.25$ $2^{-3}=0.125$ $2^{-4}=0.0625$ …

- 2 + 1 + 0.25 + 0.0625 + 0.015625 = 3.328125

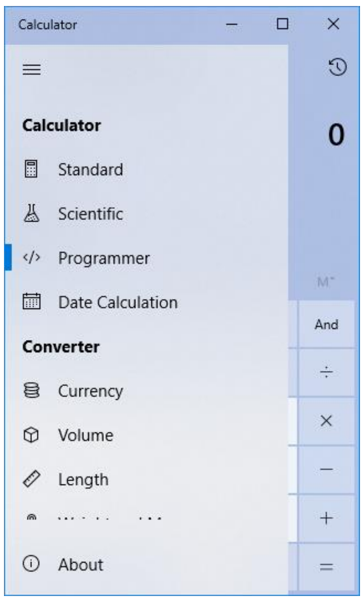
- shows how real numbers can be represented as *floating point binary numbers* inside a computer, but further detail is beyond the scope of CS1021

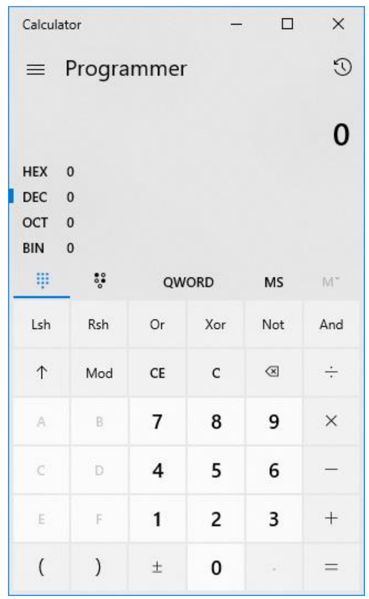
## Larger units

- larger units of information
  - 1 kilobyte (KB) = $2^{10}$ bytes = 1,024 bytes
  - 1 megabyte (MB) = 1024 x 1024 bytes = 1,024 KB = $2^{20}$ bytes = 1,048,576 bytes
  - 1 gigabyte (GB) = 1,024 MB = $2^{30}$ bytes = 1,073,741,824 bytes
  - 1 terabyte (TB) = 1024 GB = $2^{40}$ bytes = 1,099,511,627,776 bytes
- the following units are used when expressing data rates (eg. Mb/s – note the lowercase b)
  - 1 kilobit (Kb/s) = 1,000 bits per second
  - 1 megabit (Mb/s) = 1,000 kilobits = 1,000,000 bits per second
- IEC prefixes KiB, MiB, GiB, … used to differentiate between 1000 and 1024
  - technically 1KB = 1000 bytes and 1KiB = 1024bytes (although KB is often used to mean 1024)

# Programmer Calculator

- many calculators have a programmer mode (eg. Windows 10 calculator) for performing binary and hexadecimal arithmetic




- don’t use one until you know how to do the calculations “by hand”
- calculators NOT allowed in the CS1021 mid-term test or exams