COMSM1302
Overview of Computer Architecture

Lecture 1
Representation of data

Kerstin Eder

Department of Computer Science

Credits: The slides for this unit are based on course materials originally prepared by Dr Steve Kerrison.

In this lecture

Foundations	• **Data representation**, logic, Boolean algebra.
Building blocks	• Transistors, transistor based logic, simple devices, storage.
Modules	• Memory, simple controllers, FSMs, processors and execution.
Programming	• Machine code, assembly, high-level languages, compilers.
Wrap-up	• Operating systems, energy aware computing.

What’s in a number? What’s in a bit?

DATA REPRESENTATION

“There are 10 kinds of people in the world: those who understand binary numerals, and those who don’t.”

What is a number?

Without numbers, a large portion of mathematics is meaningless.

1	2	3
一	二	三
١	٢	٣

Representation – Base-10

- Sometimes we think about counting in terms of "units, tens, hundreds, …"

100s	10s	1s	Result
0	0	0	**0**
0	0	1	**1**
0	0	3	**3**
0	1	3	**13**
1	0	0	**100**
1	2	8	**128**

Representation – Base-10

- Sometimes we think about counting in terms of “units, tens, hundreds, …”, i.e. powers of 10.

10^2	10^1	10^0	Result
0	0	0	**0**
0	0	1	**1**
0	0	3	**3**
0	1	3	**13**
1	0	0	**100**
1	2	8	**128**

Representation – Base-10

- A formula for this is:

$$Y = \sum_{i=0}^{N-1} x_i \cdot 10^i \quad \text{where } X = x_{N\text{-}1\ldots}x_0$$

- Example:

$$X = x_2x_1x_0 \text{ where } x_2=1,\ x_1=0 \text{ and } x_0=4$$
$$Y = 1 \cdot 10^2 + 0 \cdot 10^1 + 4 \cdot 10^0$$
$$Y = 104$$

- **Base-10** numerical representation.

Representation – Base-10

- A formula for this is:

$$Y = \sum_{i=0}^{N-1} x_i \cdot 10^i \quad \text{where } X = x_{N\text{-}1}\ldots x_0$$

Sequence of N digits, x_i, with positions, i, from 0 to $N\text{-}1$

- **Base-10** numerical representation.

Representation – Base-10

- A formula for this is:

$$Y = \sum_{i=0}^{N-1} x_i \cdot 10^i \text{ where } X = x_{N-1} \ldots x_0$$

Sequence of N digits, x_i, with positions, i, from 0 to $N-1$

- Example:

$$X = x_2 x_1 x_0 \text{ where } x_2 = 1,\ x_1 = 0 \text{ and } x_0 = 4$$

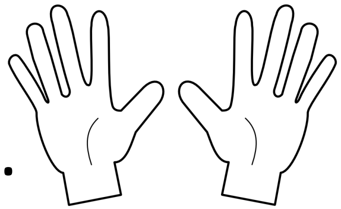
$$Y = 1 \cdot 10^2 + 0 \cdot 10^1 + 4 \cdot 10^0$$

$$Y = 104$$

- **Base-10** numerical representation.

Representation – Bases

- Base-10 is the most obvious, because we use it widely.
  - We (typically) have ten digits on our hands.



- But the base **does not have to be 10**.

$$Y = \sum_{i=0}^{N-1} x_i \cdot B^i$$

- Constraint: $x_i < B$ each digit must be smaller than the base, e.g. for B=10 we have digits 0..9.

Representation – Base-10 vs Base-2

Base-10

10^1	10^0	Result decimal
0	0	0
0	1	1
0	2	2
0	3	3
0	4	4
0	5	5
0	6	6
0	7	7
0	8	8
0	9	9
1	0	10

Base-2

2^3	2^2	2^1	2^0	Result binary
0	0	0	0	0
0	0	0	1	1
0	0	1	0	10
0	0	1	1	11
0	1	0	0	100
0	1	0	1	101
0	1	1	0	110
0	1	1	1	111
1	0	0	0	1000
1	0	0	1	1001
1	0	1	0	1010

Representation – Base-10 vs Base-2

Base-10

10s	1s	Result decimal
0	0	0
0	1	1
0	2	2
0	3	3
0	4	4
0	5	5
0	6	6
0	7	7
0	8	8
0	9	9
1	0	10

Base-2: binary

8s	4s	2s	1s	Result binary
0	0	0	0	0
0	0	0	1	1
0	0	1	0	10
0	0	1	1	11
0	1	0	0	100
0	1	0	1	101
0	1	1	0	110
0	1	1	1	111
1	0	0	0	1000
1	0	0	1	1001
1	0	1	0	1010

Representation – Base-10 vs Base-2

Base-10

10s	1s	Result decimal
0	0	**0**
0	1	**1**
0	2	**2**
0	3	**3**
0	4	**4**
0	5	**5**
0	6	**6**
0	7	**7**
0	8	**8**
0	9	**9**
1	0	**10**

Base-2: binary

8s	4s	2s	1s	Result binary
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**10**
0	0	1	1	**11**
0	1	0	0	**100**
0	1	0	1	**101**
0	1	1	0	**110**
0	1	1	1	**111**
1	0	0	0	**1000**
1	0	0	1	**1001**
1	0	1	0	**1010**

Representation – Bases 8 and 16

Base-8

8s	1s	Result octal
0	0	0
0	1	1
0	2	2
0	3	3
0	4	4
0	5	5
0	6	6
0	7	7
1	0	10
1	1	11
1	2	12

Base-16

16s	1s	Result hexadecimal
0	0	0
0	1	1
0	2	2
0	3	3
0	4	4
0	5	5
0	6	6
0	7	7
0	8	8
0	9	9
0	a	a

Representation – Bases 8 and 16

Base-8

8s	1s	
0	0	
0	1	
0	2	
0	3	
0	4	4
0	5	5
0	6	6
0	7	7
1	0	10
1	1	11
1	2	12

Remember…

1, 2, 3
a, b, c

They're just symbols, digits!

Base-16

6s	1s	Result hexadecimal
	0	0
	1	1
	2	2
	3	3
0	4	4
0	5	5
0	6	6
0	7	7
0	8	8
0	9	9
0	a	a

Final notes on bases

- Base-16 needs 16 symbols to provide 16 digits:

  `0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f`

Final notes on bases

- Base-16 needs 16 symbols to provide 16 digits:

  `0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f`

Final notes on bases

- Base-16 needs 16 symbols to provide 16 digits:

  0,1,2,3,4,5,6,7,8,9,**a,b,c,d,e,f**

- And it doesn't stop there … Base-64

  A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,a,b,c,d,e,f,g,h,I,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,0,1,2,3,4,5,6,7,8,9,+,/

- What does 1011 mean?

Final notes on bases

- Base-16 needs 16 symbols to provide 16 digits:

  `0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f`

- And it doesn't stop there … Base-64

  `A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,a,b,c,d,e,f,g,h,I,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,0,1,2,3,4,5,6,7,8,9,+,/`

- What does 1011 mean?
- Sometimes we need to give hints with prefixes:
  - 0b1011 (Base-2, binary)
  - 0o1011 (Base-8, octal)
  - 0x1011 (Base-16, hexadecimal)
  - Because it's **not always obvious** what base a number is!

In comp-arch, base-2 is king

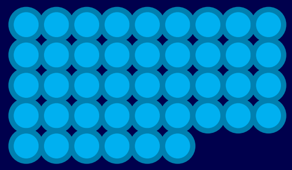
- Computers tend to represent data internally in base-2 (binary).
  - We will see why in our next lecture!
- Binary is not very easy to read as a human.
- Base-16 (hexadecimal) is easier, more compact.

0x	8				b			
0b	1	0	0	0	1	0	1	1

What's in a number?

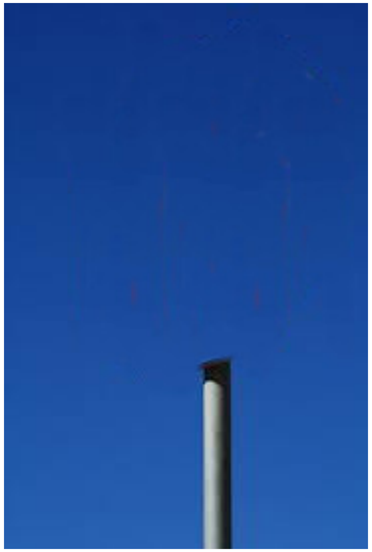
- We now know how to represent numbers.
- But what do those numbers represent?

42, 0x2a, 0b101010, "Forty two"

A quantity	An intensity of colour	A character	An angle
		*	

Binary number representations

- Size
  - measured in Bits
- Limited range
  - Unsigned
  - Signed
  - Fixed point
- Dynamic range
  - Floating point



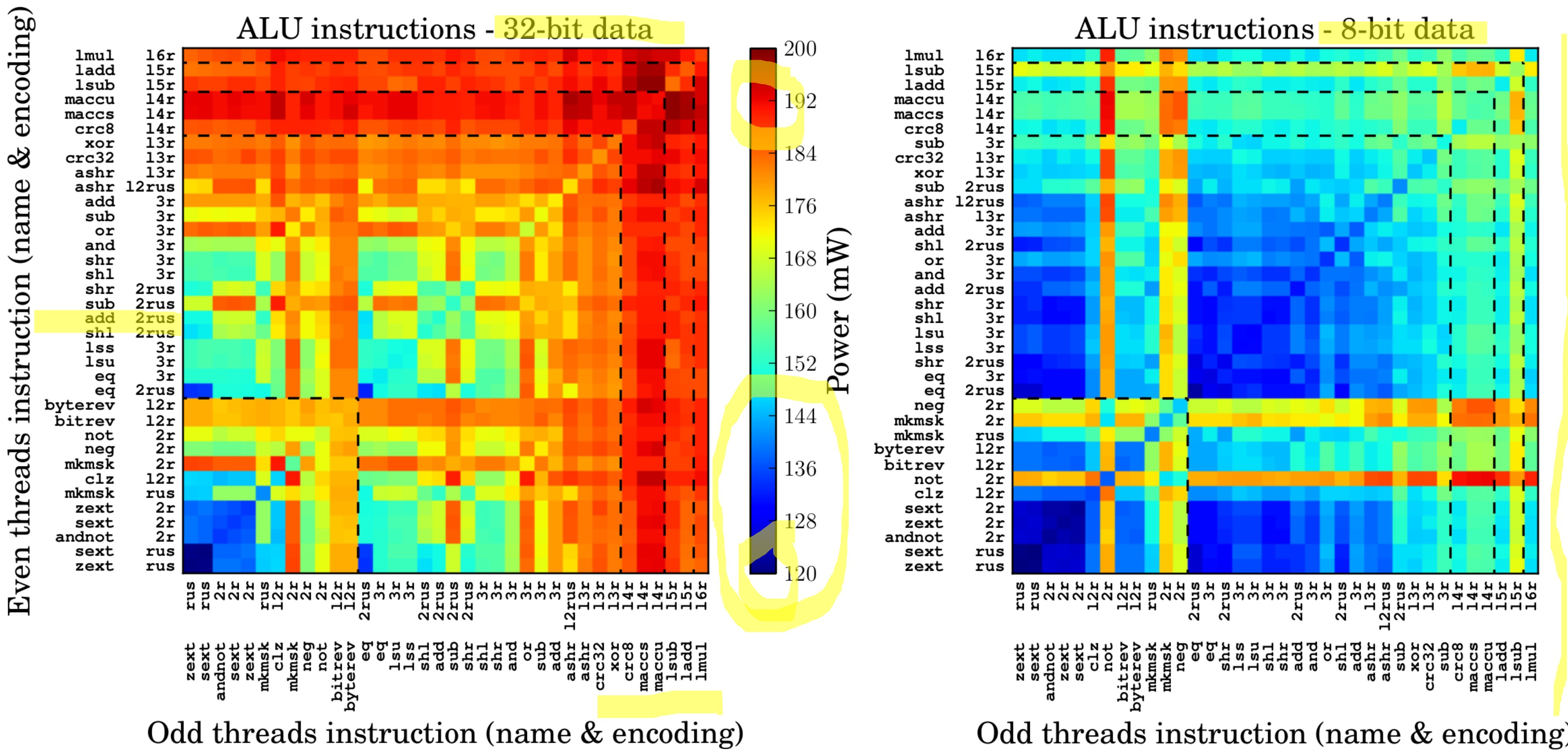
The smallest unit of information:

The smallest unit of information: Bit

The smallest unit of information: Bit

- In binary, each digit contains **one bit** of information.
  - Analogous to an on/off switch
- In computer architecture, most of what we do is governed by **how many bits** we use to represent something.
  - 8-bit, 16-bit, 32-bit, …
- Professional programmers should also care about how many bits are needed.

Why does this matter?



S. Kerrison and K. Eder. 2015. “Energy Modeling of Software for a Hardware Multithreaded Embedded Microprocessor”. ACM Trans. Embed. Comput. Syst. 14, 3, Article 56 (April 2015), 25 pages. DOI=10.1145/2700104 http://doi.acm.org/10.1145/2700104

Unsigned numbers

- Can be calculated based on the formula we defined earlier
- $B = 2$
- $N = ?$

$$Y = \sum_{i=0}^{N-1} x_i \cdot B^i$$

Binary

0b10000000

Hex

0x10

C

```
uint32_t a = 128;
uint32_t b = 0x10;

if (a == b) {
    return 1;
} else {
    return 0;
}
```

Unsigned numbers in 8 bits

- If we have limited storage space, we have a limited range.
- For N bits: $0 <= Y <= 2^N - 1$
  - e.g. for N=8 bits we can represent unsigned numbers from ___ to ___, giving a total of ___ numbers

Unsigned numbers in 8 bits

- If we have limited storage space, we have a limited range.
- For N bits: $$0 <= Y <= 2^N - 1$$
  - e.g. for N=8 bits we can represent unsigned numbers from 0 to 255, giving a total of 256 numbers

Unsigned numbers in 8 bits

- If we have limited storage space, we have a limited range.
- For N bits: $$0 <= Y <= 2^N - 1$$
  - e.g. for N=8 bits we can represent unsigned numbers from 0 to 255, giving a total of 256 numbers

Binary

0b11111111 + 0b00000001

=

0b00000000

Hex

0xff + 0x01

=

0x00

C

```
#include <stdio.h>
#include <stdint.h>

int main() {

uint8_t a = 0xff;
uint8_t b = a + 1;

printf("a = %u\n",a);
printf("a = %x\n",a);

printf("b = %u\n",b);

}
```

To represent the result of this addition we need one extra bit, which we don't have. This creates an **overflow.**

Different integers https://os.mbed.com/handbook/C-Data-Types

Integer Data Types

C type	stdint.h type	Bits	Sign	Range
char	uint8_t	8	Unsigned	0 .. 255
signed char	int8_t	8	Signed	-128 .. 127
unsigned short	uint16_t	16	Unsigned	0 .. 65,535
short	int16_t	16	Signed	-32,768 .. 32,767
unsigned int	uint32_t	32	Unsigned	0 .. 4,294,967,295
int	int32_t	32	Signed	-2,147,483,648 .. 2,147,483,647
unsigned long long	uint64_t	64	Unsigned	0 .. 18,446,744,073,709,551,615
long long	int64_t	64	Signed	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807

Signed numbers

- We typically represent numbers with an implicit "+" and an explicit "-" when we write them.
- Computer architecture requires some space to store that information.
- Simplest example: sign-magnitude representation

Sign bit	64	32	16	8	4	2	1	Base 10
0	0	0	0	1	0	1	0	10
1	0	0	0	1	0	1	0	-10

What's wrong with sign-magnitude?

- In 8 bit sign-magnitude representation:
  - What range of numbers can we represent?
  - How many numbers can we represent?

Answer this question

before moving to the next slide ;)

What’s wrong with sign-magnitude?

- In 8 bits, what range of numbers can we represent?
  - We use the first bit as the sign bit, 0 for “+” and 1 for “–”.
  - The remaining 7 bits can represent numbers from 0 to 127.
  - This gives a range from -127 to +127; a total of 255 numbers.

sign bit	64	32	16	8	4	2	1	decimal
0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	127
1	1	1	1	1	1	1	1	-127

What’s wrong with sign-magnitude?

- In 8 bits, what range of numbers can we represent?
  - We use the first bit as the sign bit, 0 for “+” and 1 for “–”.
  - The remaining 7 bits can represent numbers from 0 to 127.
  - This gives a range from -127 to +127, a total of 255 numbers.

There are two ways to represent zero: 00000000 (+0) and 10000000 (-0)!

sign bit	64	32	16	8	4	2	1	decimal
0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	127
1	1	1	1	1	1	1	1	-127
1	0	0	0	0	0	0	0	-0

2s complement

- Let's change what the **most significant bit** (MSB) represents.

$$Y = -x_{N-1} \cdot 2^{N-1} + \sum_{i=0}^{N-2} x_i \cdot 2^i$$

- 128	64	32	16	8	4	2	1	Base 10
0	0	0	0	1	0	1	0	**10**
1	0	0	0	1	0	1	0	**-118**

- In 2s complement, the circuitry for addition and subtraction can be unified.

2s complement

- Let's change what the **most significant bit** (MSB) represents.

$$Y = -x_{N-1} \cdot 2^{N-1} + \sum_{i=0}^{N-2} x_i \cdot 2^i$$

-128	64	32	16	8	4	2	1	Base 10
0	0	0	0	1	0	1	0	10
1	0	0	0	1	0	1	0	-118

- In 2s complement, the circuitry for addition and subtraction can be unified.

2s complement range

- In 8 bits 2s complement representation:
  – What range of numbers can we represent?
  – How many numbers can we represent?

Answer this question

before moving to the next slide ;)

2s complement range

- In 8 bits 2s complement we can represent 256

-128	64	32	16	8	4	2	1	decimal
0	1	1	1	1	1	1	1	127
0	1	1	1	1	1	1	0	126
0	0	0	0	0	0	1	0	2
0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	-1
1	1	1	1	1	1	1	0	-2
1	0	0	0	0	0	1	0	-126
1	0	0	0	0	0	0	1	-127
1	0	0	0	0	0	0	0	-128

Unsigned vs 2s complement

bit pattern	unsigned decimal value	2s complement decimal value
011	3	**3**
010	2	2
001	1	1
000	**0**	0
111	**7**	-1
110	6	-2
101	5	-3
100	4	**-4**

Calculating the 2s complement

- To calculate the 2's complement of an integer, invert the binary equivalent of the number by changing all the ones to zeroes and all the zeroes to ones (also called 1's complement), then add one.
  - How do we represent -5?

Calculating the 2s complement

- To calculate the 2's complement of an integer, invert the binary equivalent of the number by changing all the ones to zeroes and all the zeroes to ones (also called 1's complement), then add one.
  - How do we represent -5?
    - How many bits do we need?

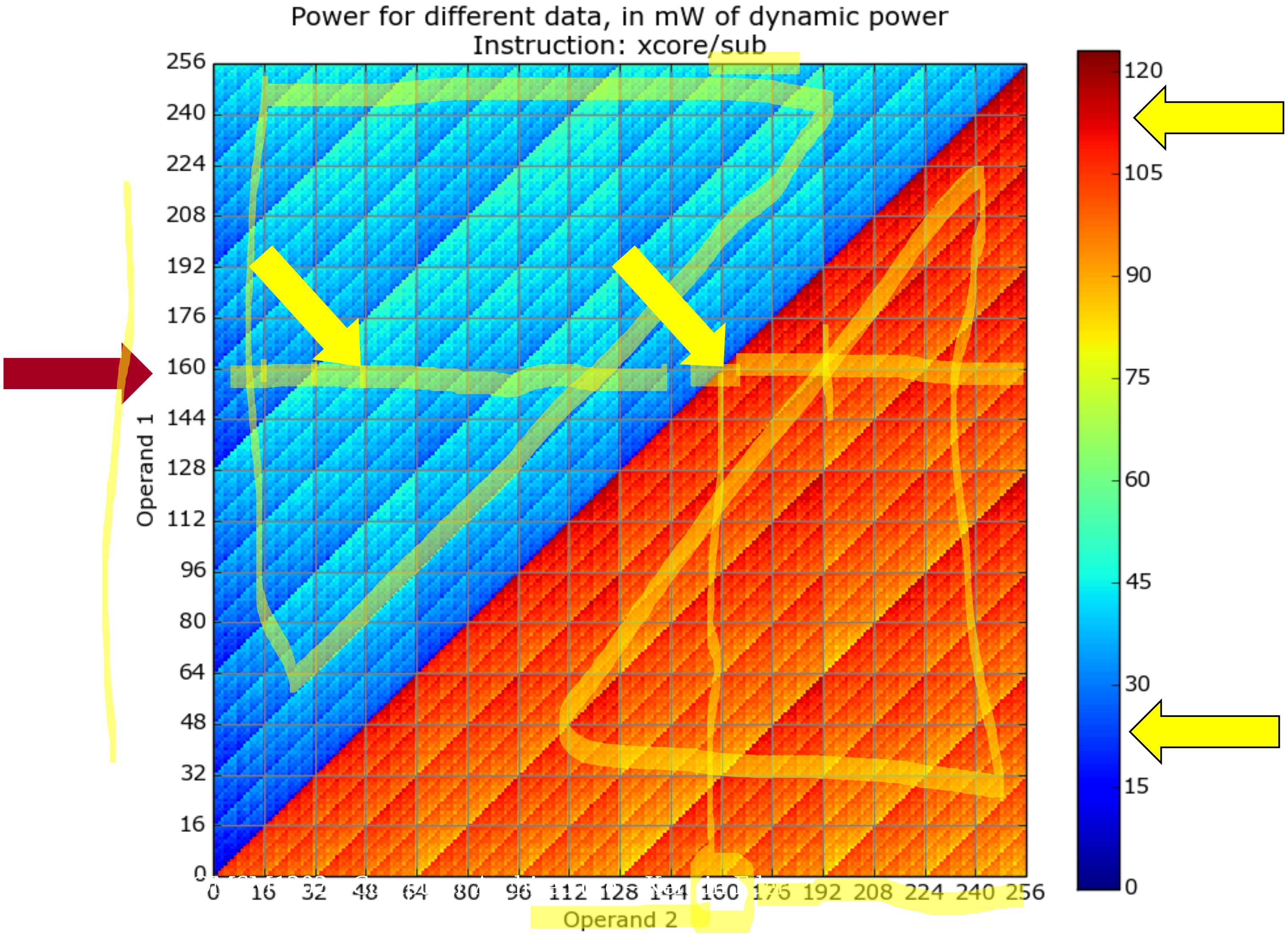
Calculating the 2s complement

- To calculate the 2's complement of an integer, invert the binary equivalent of the number by changing all the ones to zeroes and all the zeroes to ones (also called 1's complement), then add one.
  – How do we represent -5?
    - How many bits do we need? *3 bits for 5 but 4 bits for -5*

Calculating the 2s complement

- To calculate the 2's complement of an integer, invert the binary equivalent of the number by changing all the ones to zeroes and all the zeroes to ones (also called 1's complement), then add one.
  - How do we represent -5?
    - How many bits do we need? *3 bits for 5 but 4 bits for -5*
    - 5 is represented by binary 0101 (using 4 bits)
    - 0101 inverted is 1010
    - 1010 + 0001 = 1011
    - 1011 = 1x(-8) + 0x4 + 1x2 + 1x1 = -8 +2 +1 = -5

What happens here?



What's wrong with integers?

- Whole numbers
- Limited range
  - 32-bit int (int32_t) range is -2^{31} to $2^{31}-1$

$$3 \div 2 = ?$$

Fixed point

- In decimal, we have the **decimal-point** (1.5)
- Let's introduce a point…
- Let p = 1

$$Y = \sum_{i=0-p}^{N-1-p} x_i \cdot 2^i$$

$$a^{-b} = \frac{1}{a^b}$$

2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	
64	32	16	8	4	2	1	0.5	**Base 10**
0	0	0	0	1	0	1	1	**5.5**
1	0	0	0	1	0	0	0	**68**

Fixed point

Choose the location of the point carefully, considering

- What **range** do you need?
  - from *<smallest number>* to *<largest number>*
- What **precision** do you need?
  - *What is the required distance between successive numbers?*

2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	
8	4	2	1	0.5	0.25	0.125	0.0625	**Base 10**
0	0	0	0	1	0	1	1	**0.6875**
1	0	0	0	1	0	0	0	**8.5**

Floating point

- Flexible representation by having a point that can be moved.

$$Y = (-1)^S \cdot M \cdot 2^E$$

Sign	Exponent *(E)*			Mantissa *(M)*				Base 10
S	-4	2	1	8	4	2	1	
0	0	0	1	1	0	0	1	18
1	1	0	1	1	0	0	1	?

Floating point

- Flexible representation by having a point that can be moved.

$$Y = (-1)^S \cdot M \cdot 2^E$$

Sign	Exponent (E)			Mantissa (M)				Base 10
S	-4	2	1	8	4	2	1	
0	0	0	1	1	0	0	1	18
1	1	0	1	1	0	0	1	-1.125

See the difference

```
printf(“%d\n”,3/2);          //Integer
printf(“%f\n”,3.0/2.0);      //Float
```

What's wrong with floating point?

- Its precision can be a problem
  - Divide a **very large** number by a **very small** number… get an inaccurate answer.
- How do we choose the number of bits in E and M?
- **IEEE 754** tells us!
  - Defines different types of floating point representation.
  - How special values like infinity and not-a-number (NaN) should be represented.

Summary

- Different bases
  - binary, octal, hexadecimal
- What a number represents
  - Anything!
- Representing numbers
  - Signed / unsigned
  - Integer / fixed- or floating-point
    - What Every Computer Scientist Should Know About Floating-Point Arithmetic (https://dl.acm.org/doi/10.1145/103162.103163)
  - Space, bits, range and precision

Now, read this sentence again

"There are 10 kinds of people in the world: those who understand binary numerals, and those who don't."

Well done!

In this lecture

Foundations	• **Data representation**, logic, Boolean algebra.
Building blocks	• Transistors, transistor based logic, simple devices, storage.
Modules	• Memory, simple controllers, FSMs, processors and execution.
Programming	• Machine code, assembly, high-level languages, compilers.
Wrap-up	• Operating systems, energy aware computing.

In the next lecture

Foundations	• Data representation, **logic, Boolean algebra.**
Building blocks	• Transistors, transistor based logic, simple devices, storage.
Modules	• Memory, simple controllers, FSMs, processors and execution.
Programming	• Machine code, assembly, high-level languages, compilers.
Wrap-up	• Operating systems, energy aware computing.

A bit of number fun ☺

- How does this work?
  - *Below is a set of 6 cards, each showing a set of numbers.*
  - *Show all the cards to a friend and ask your friend to select one number from any one card. Then show the other 5 cards to your friend asking her or him to tell you whether their chosen number appears on these cards.*
  - *Take all the cards on which your friend says their number appears, add together the top left hand corner number of each card. The total is the number your friend selected.*

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63	8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31 40 41 42 43 44 45 46 47 56 57 58 59 60 61 62 63	32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63	2 3 6 7 10 11 14 15 18 19 22 23 26 27 30 31 34 35 38 39 42 43 46 47 50 51 54 55 58 59 62 63	4 5 6 7 12 13 14 15 20 21 22 23 28 29 30 31 36 37 38 39 44 45 46 47 52 53 54 55 60 61 62 63