COMSM1302
Overview of Computer Architecture

Lecture 1 – Introduction & representation of data

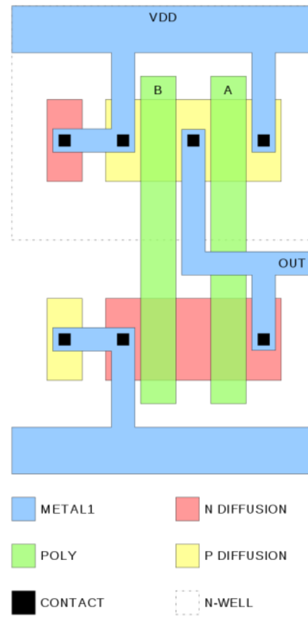
What to expect

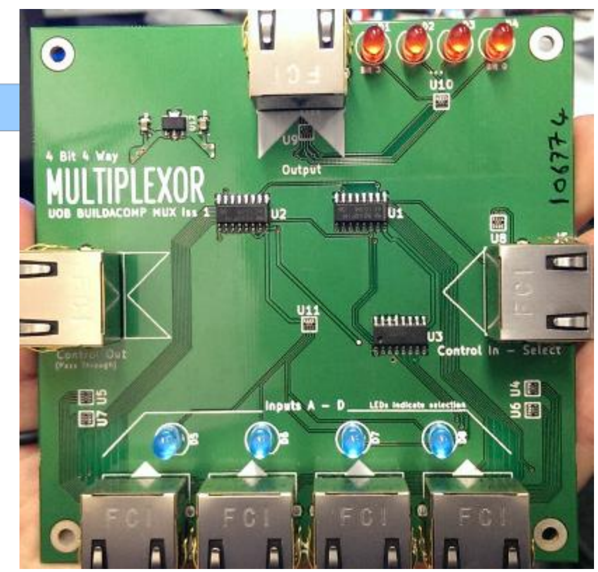
INTRODUCTION

Where are you?

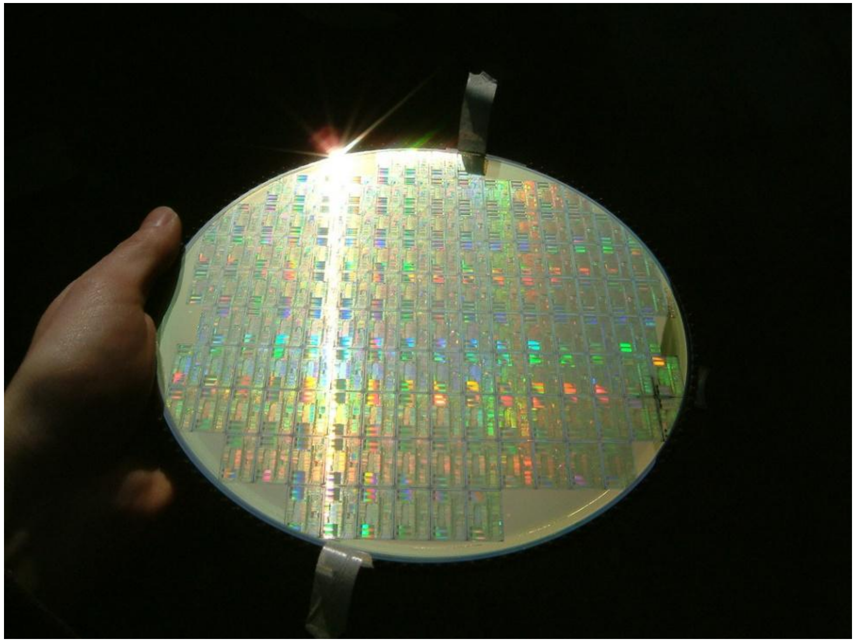
- This is a unit on **computer architecture**.
- But *what is that*?

- Modern computer systems are extremely complex.
- Computer architecture is about how we design and construct them.
  - And make sure they're **useful**!





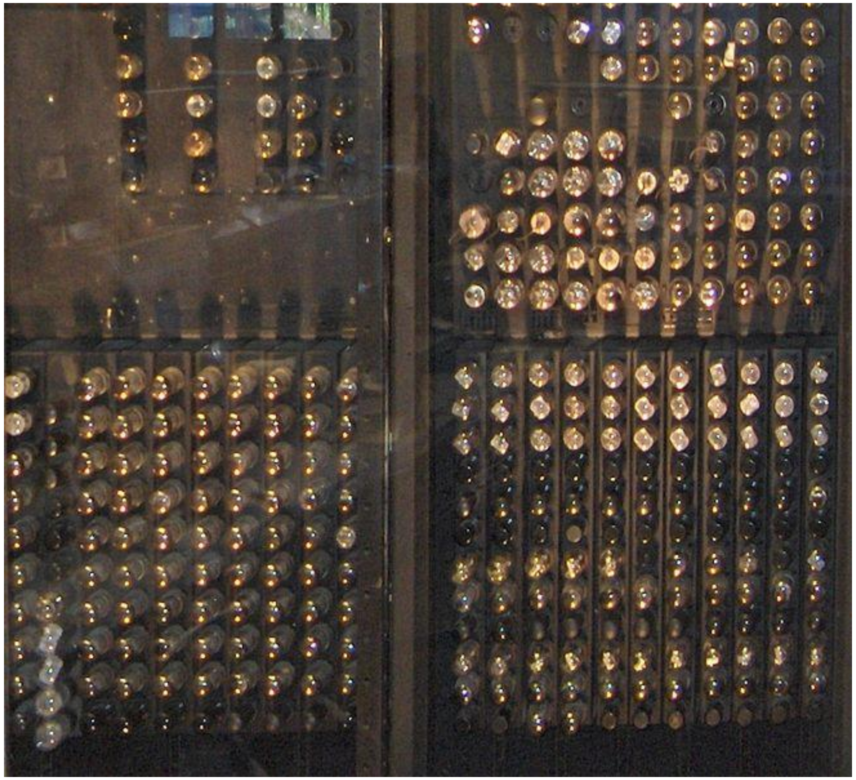
Unit structure



- Teaching
  - Two lectures per week, 1 hour each.
  - Lab sessions, every week, 2 hours each.
  - Worksheets (self assessed).
- **Assessment**
  - **Lab-based** assignments.
  - End of unit **viva** (short interview).

Getting help

- Questions in lectures
- Supervision and demonstration in labs
- [Unit web page]()
- [Forum]()
  - Register for e-mail alerts
  - Help each other
- Don't be shy ☺



In this unit

Foundations	• Data representation, logic.
Building blocks	• Transistors, transistor based logic, simple devices, storage.
Modules	• Hex modules, memory, simple controller and processor.
Programming	• Assembly, assembler, language, compilation phases, boot-strapping.
Bigger systems	• ARM & Thumb, I/O, protecting shared systems, memory hierarchy, multi-processors, networks.
Wrap-up	• More examples, historical computers, contemporary systems.

What’s in a number? What’s in a bit?

DATA REPRESENTATION

In this unit

Foundations	• **Data representation**, logic.
Building blocks	• Transistors, transistor based logic, simple devices, storage.
Modules	• Hex modules, memory, simple controller and processor.
Programming	• Assembly, assembler, language, compilation phases, boot-strapping.
Bigger systems	• ARM & Thumb, I/O, protecting shared systems, memory hierarchy, multi-processors, networks.
Wrap-up	• More examples, historical computers, contemporary systems.

What is a number?

- Counting

1	2	3
一	二	三
١	٢	٣

- Without which a large portion of mathematics is meaningless.

Representation - Bases

- 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 - Bases

- An expression for this:

$$Y = \sum_{i=0}^{N-1} x_i \cdot 10^i$$

- Example:

$$x = \{ 1, \quad 0, \quad 4 \}$$

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

$$Y = 104$$

- **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:

$$x_i < B$$

Representation – Base-2

Base-10

10s	1s	Result
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

8s	4s	2s	1s	Result
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	
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!

Base-16

16s	1s	Result
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**

Final notes on bases

- Base-16 needs 16 symbols per digit:
  - `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,+,/`
- Sometimes we give hints with prefixes:
  - 0b1011 (Base-2, binary)
  - 0o1011 (Base-8, octal)
  - 0x1011 (Base-16, hexadecimal)
  - Because it's **not always obvious** what base something is!

In comp-arch, base-2 is king

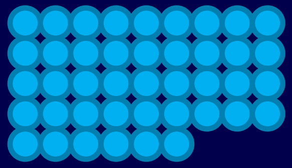
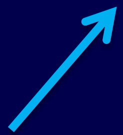
- Computers tend to represent data internally in base-2 (binary).
  - We will see why next lecture!
- Binary isn't very easy to read as a human.
- Base-16 (hexadecimal) is easier, more compact.

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

What’s in a number?

- We now know about 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
  - Bits
- Limited range
  - Unsigned
  - Signed
  - Fixed point
- Dynamic range
  - Floating point

Bits

- Information theory
  - A **bit** is a unit of information gained from, or uncertainty (entropy) prior to, observing an outcome.
    - Flipping a coin – one bit of information
    - Rolling a 6-sided die – $\log_2(6)$ – 2.58 bits
  - Claude Shannon, 1948, A Mathematical Theory of Communication - http://www3.alcatel-lucent.com/bstj/vol27-1948/articles/bstj27-3-379.pdf

Bits

- 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, …

Unsigned numbers

- Follows the expression we defined
- B = 2
- N = ?

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

Binary

0b10000000

Hex

0x10

C

```
uint32_t foo = 128;
uint32_t bar = 0x10;

if (foo == bar) {
    return 1;
} else {
    return 0;
}
```

Unsigned numbers in 8 bits

- If we have limited storage space, we have limited range.
- For N bits:

$$0 <= Y <= 2^N - 1$$

Binary

```
0b11111111 + 0b00000001

=

0b00000000
```

Hex

```
0xff + 0x01

=

0x0
```

C

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

printf(“%u\n”,b);
```

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

Neg	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 bits, what range of numbers can we represent?

Example on the board

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**

2s complement range

- In 8 bits, what range of numbers can we represent?

Example on the board

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

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

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.
- What **precision** do you need?
- What **range** do you need?

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			Mantissa				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
  - Ease of understanding
  - Ease of implementation
- What a number represents
  - Anything!
- Representing numbers
  - Signed / unsigned
  - Integer / fixed- or floating-point
  - Space, bits, range and precision

In this unit

Foundations	• Data representation, **logic**.
Building blocks	• Transistors, transistor based logic, simple devices, storage.
Modules	• Hex modules, memory, simple controller and processor.
Programming	• Assembly, assembler, language, compilation phases, boot-strapping.
Bigger systems	• ARM & Thumb, I/O, protecting shared systems, memory hierarchy, multi-processors, networks.
Wrap-up	• More examples, historical computers, contemporary systems.