COMS12200 lecture: week #1

- **COMS12200** begins the **computer architecture** theme:
  1. how do computers work,
  2. how do we design and build computers, and
  3. how do we use computers effectively.
- The term "computer architecture" can be explained via comparison, i.e.,
  - if a civil engineer fills a similar to an electrical engineer, then
  - an architect fills a similar role to a computer architect,

  but has somewhat bizarre history:

Quote

> *... we had asked for "computer architects"; the Royal Institute of British Architects pointed out that the word "architect" was controlled by an Act of Parliament passed in* 1933*, which specified that architects could design houses, landscapes or ships, but not computers.*
>
> *– Barron*

http://xkcd.com/676/

- Organisational points to keep in mind:
  1. *Everything* is driven via the **unit web-site**

     http://www.cs.bris.ac.uk/Teaching/Resources/COMS12200/

     except for the **unit forum** at

     https://www.cs.bris.ac.uk/forum/index.jsp?title=COMS12200

     noting that we *don't* use Blackboard.
  2. The timetabled content for this unit includes
     - 2 lectures $\times$ 1 hour, and
     - 1 lab. session $\times$ $1\frac{1}{2}$ hours (from week 2 onward only).

     per week.
  3. The unit assessment places
     - 50% weight on a viva in TB1 exam period, and
     - 50% weight on a viva in TB2 exam period,

     that are *both* supported by a portfolio of work completed within the lab. sessions.
  4. There are *plenty* of resources and sources of help available
     - sets of lecture notes (in slide and often also extended formats) including pointers to additional reading,
     - sets of exam-style practice questions,
     - offline, personal contact (e.g., lecturer office hours, lab. demonstrators), and
     - online, remote contact (e.g., unit forum)

     but the emphasis is on *you* to make use of them effectively.

- Note that

$$\begin{aligned} 20 \text{ credit points} &\simeq 200 \text{ hours work} \\ &\simeq (24 \text{ weeks} \times 2 \text{ lectures} \times 1 \text{ hour}) + X \text{ hours} \\ &\simeq (48 + X) \text{ hours} \end{aligned}$$

  meaning $X \simeq 152$ hours work *outside* lectures.
- So at $152/24 \simeq 6$ hours per week, an example week might also include *at least*
  - 1 lab. session $\times 1\frac{1}{2}$ hours $= 1\frac{1}{2}$ hours of hands-on practial work, plus
  - $4\frac{1}{2}$ hours of lab. catch-up, additional reading, revision etc.

Some motivation for COMS12200 (1)

- Question: ranging from the late 1970*s* to the late 1990*s*, spot the difference(s).

Some motivation for COMS12200 (1)

- Question: ranging from the late 1970*s* to the late 1990*s*, spot the difference(s).

Some motivation for COMS12200 (1)

- Question: ranging from the late 1970*s* to the late 1990*s*, spot the difference(s).

Some motivation for COMS12200 (1)

- Question: ranging from the late 1970*s* to the late 1990*s*, spot the difference(s).

Some motivation for COMS12200 (1)

- Question: ranging from the late 1970*s* to the late 1990*s*, spot the difference(s).

Some motivation for COMS12200 (1)

- Question: ranging from the late 1970*s* to the late 1990*s*, spot the difference(s).





Some motivation for COMS12200 (1)

- Question: ranging from the late 1970*s* to the late 1990*s*, spot the difference(s).





- Answer: advances in *at least* two fields, namely
  1. improved 2D and 3D computer graphics techniques, and
  2. improved general-purpose processors and display technologies (including a lineage of special-purpose GPUs)

  where it's hard to tell which one leads which: more often than not, computer architecture evolves symbiotically to enable other fields.

Some motivation for COMS12200 (1)

Quote

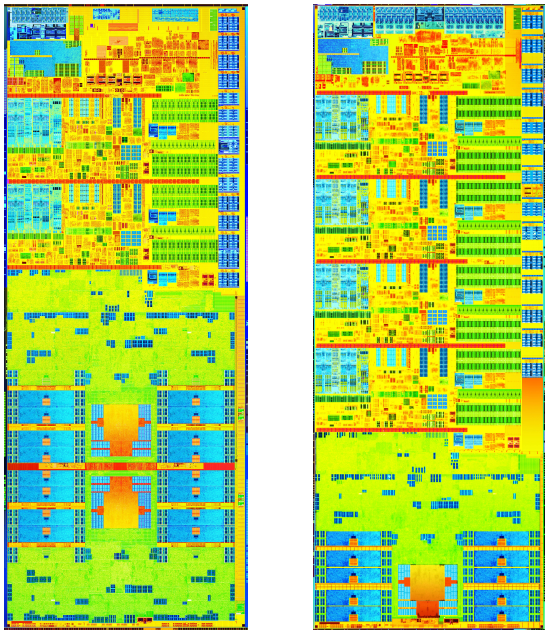
Where a calculator on the ENIAC is equipped with 18,000 *vacuum tubes and weighs* 30 *tons, computers in the future may have only* 1,000 *vacuum tubes and perhaps weigh* 1.5 *tons.*

– Popular Mechanics, 1949

Component	Then	Now
Processor	thousands of instructions/sec	billions of instructions/sec
Memory	hundreds of bits	gigabytes
Storage	thousands of bytes	terabytes
Input/Output	paper tape	anything you can imagine
Software	hand wired	high-level languages

Some motivation for COMS12200 (2)

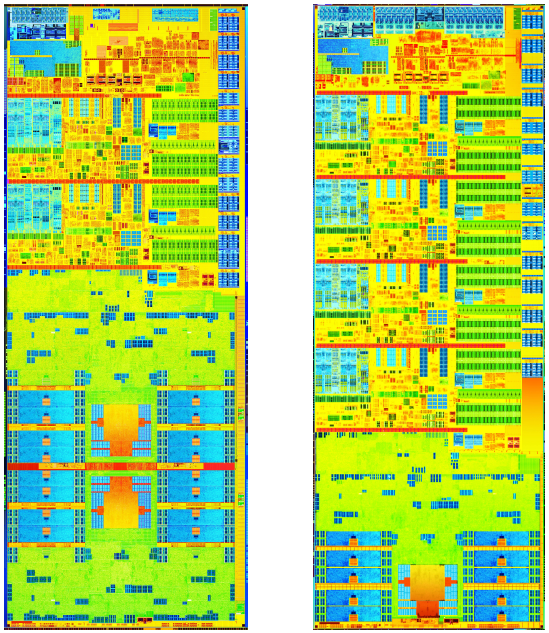
- Question: identify these objects, and spot the difference(s).



http://download.intel.com/newsroom/kits/core/4thgen/gallery/images/

Some motivation for COMS12200 (2)

- Question: identify these objects, and spot the difference(s).



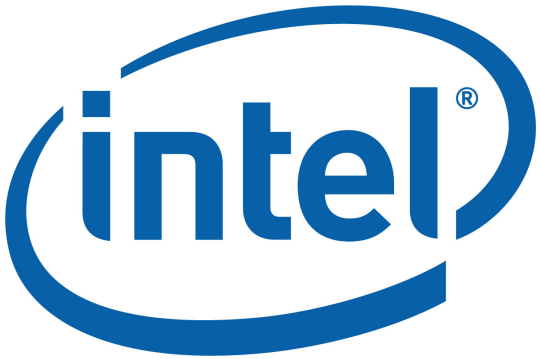
- Answer: the left-hand picture is a dual-core Haswell model Intel processor, while the right-hand picture is a quad-core Haswell model Intel processor.
- Effective design *and* utilisation of increasingly complex, next-generation computing platforms is vital for progress of most fields in CS.

---

http://download.intel.com/newsroom/kits/core/4thgen/gallery/images/

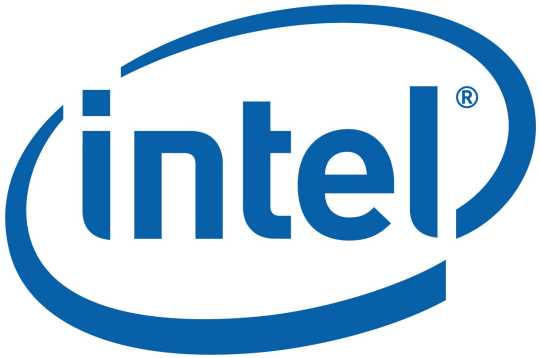
Some motivation for COMS12200 (3)

- Question: which processor designs have a larger deployment (i.e., which sell more units)?



Some motivation for COMS12200 (3)

- Question: which processor designs have a larger deployment (i.e., which sell more units)?



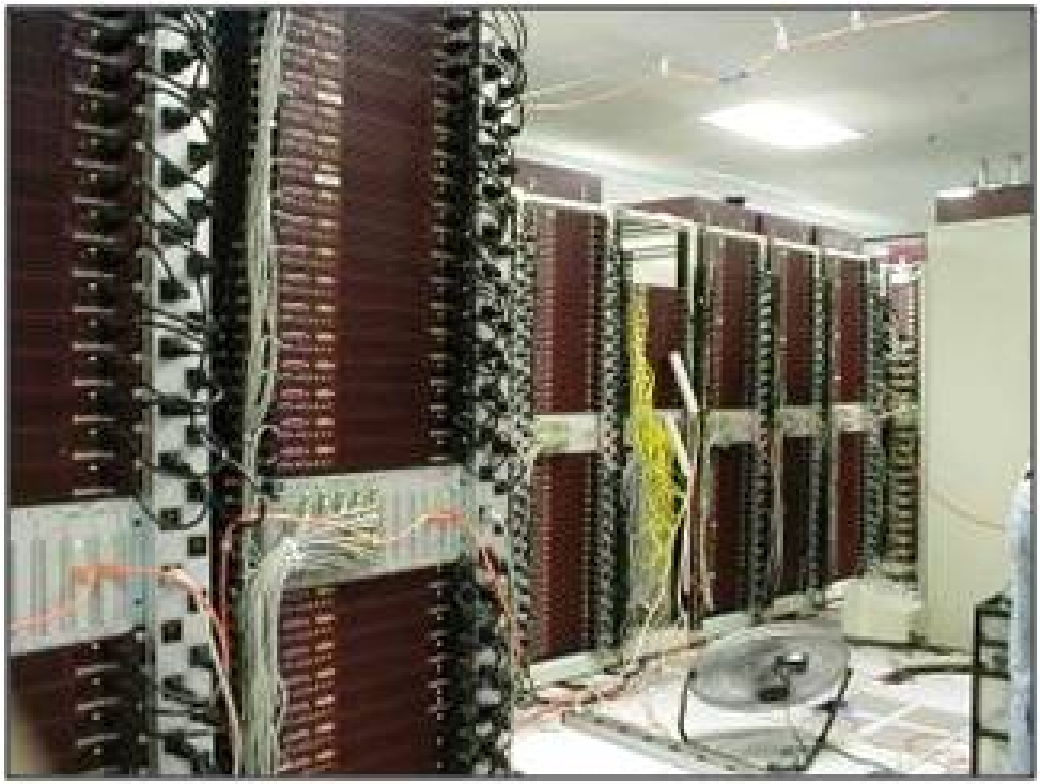
- Answer: hard to say, but arguably ARM: the embedded processor market is *huge* compared to the desktop/server market.
- Two points are important:
  1. effective use of embedded computing technologies (versus treating all computers as a desktop/server) is an increasingly important field, and
  2. the UK specifically is entering a phase of *knowledge driven* industry, so you'll need to "know" not just "do".

Some motivation for COMS12200 (4)

- Question: ranging from the late 1990*s* to the early 2000*s*, spot the difference(s).

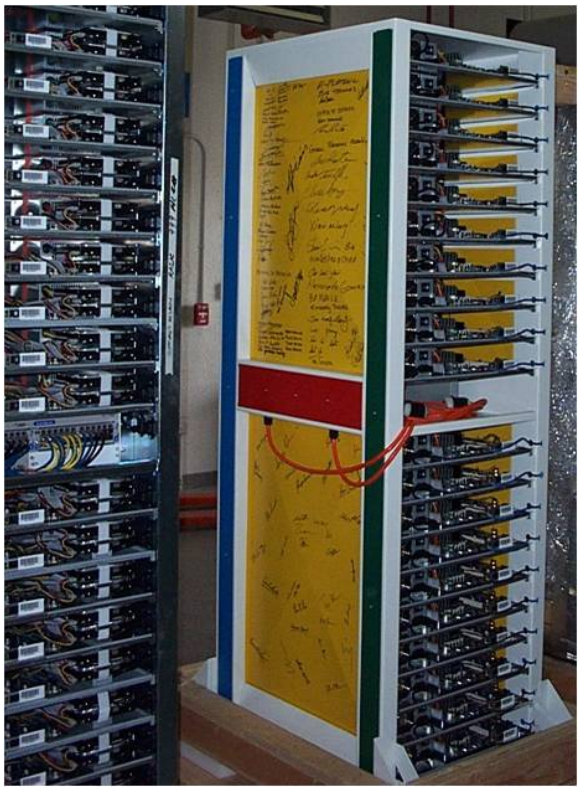
Some motivation for COMS12200 (4)

- Question: ranging from the late 1990*s* to the early 2000*s*, spot the difference(s).



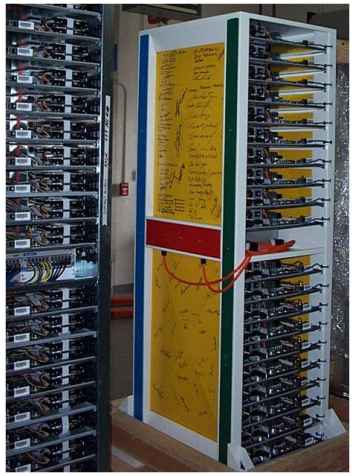
Some motivation for COMS12200 (4)

- Question: ranging from the late 1990*s* to the early 2000*s*, spot the difference(s).



Some motivation for COMS12200 (4)

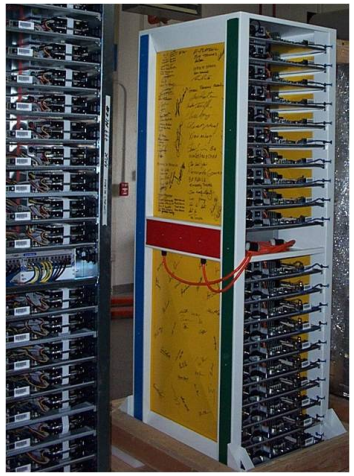
- Question: ranging from the late 1990*s* to the early 2000*s*, spot the difference(s).



http://research.google.com/people/jeff/

Some motivation for COMS12200 (4)

- Question: ranging from the late 1990*s* to the early 2000*s*, spot the difference(s).



- Answer: the left-hand picture is *the* Google data center circa 1997, the right-hand picture is *a* Google data center circa 2007.
- Even if you "just" build web-sites, high-volume (e.g., Google, Amazon, BBC etc.) demands care wrt. computer architecture: a holistic approach to CS is unavoidable.

---

http://research.google.com/people/jeff/

Some motivation for COMS12200 (5)

- Question: you want to print n integers held in an array A to the terminal, tab or new line delimitation is possible; which is better and why?

Listing (C)

```
1 void write( FILE* F, int* A, int n ) {
2   for( int i = 0; i < n; i++ ) {
3     fprintf( F, "%d\t", A[ i ] );
4   }
5 }
```

Listing (C)

```
1 void write( FILE* F, int* A, int n ) {
2   for( int i = 0; i < n; i++ ) {
3     fprintf( F, "%d\n", A[ i ] );
4   }
5 }
```

Some motivation for COMS12200 (5)

- Question: you want to print n integers held in an array A to the terminal, tab or new line delimitation is possible; which is better and why?

Listing (C)

```
1 void write( FILE* F, int* A, int n ) {
2   for( int i = 0; i < n; i++ ) {
3     fprintf( F, "%d\t", A[ i ] );
4   }
5 }
```

Listing (C)

```
1 void write( FILE* F, int* A, int n ) {
2   for( int i = 0; i < n; i++ ) {
3     fprintf( F, "%d\n", A[ i ] );
4   }
5 }
```

- Answer: if better means faster, the left-hand tab delimited one.
  - The C standard library is more complex than the interface suggests; for example, `printf` employs a user-space buffer when `stdout` is connected to a terminal ...
  - ... the purpose is to reduce the number of system calls, i.e., calls into the OS kernel.
- This means understanding
  - how your program interfaces with the the OS, and
  - how the OS interfaces with physical hardware

  is an important part of writing better programs.

Some motivation for COMS12200 (6)

- Question: these are both valid C programs for x86-32 processors; what do they do, and why write them like this?

Listing (C)

```
1  int weight( uint32_t x ) {
2    int t = 0;
3
4    for( int i = 0; i < 32; i++ ) {
5      if( ( x >> i ) & 1 ) {
6        t += 1;
7      }
8    }
9
10   return t;
11 }
```

Listing (C)

```
1  int weight( uint32_t x ) {
2    int t;
3
4    asm ( "   movl $0,   %0 ; movl $32,  %%ecx ; "
5          "0: decl %%ecx    ; bt   %%ecx,%1    ; "
6          "   adcl $0,   %0 ; test %%ecx,%%ecx ; "
7          "   jnz  0b       ;                    "
8
9          : "=&r" (t) : "r" (x) : "%ecx", "cc" );
10
11   return t;
12 }
```

Some motivation for COMS12200 (6)

- Question: these are both valid C programs for x86-32 processors; what do they do, and why write them like this?

Listing (C)

```
1  int weight( uint32_t x ) {
2    int t = 0;
3
4    for( int i = 0; i < 32; i++ ) {
5      if( ( x >> i ) & 1 ) {
6        t += 1;
7      }
8    }
9
10   return t;
11 }
```

Listing (C)

```
1  int weight( uint32_t x ) {
2    int t;
3
4    asm ( "   movl $0,   %0 ; movl $32,  %%ecx ; "
5          "0: decl %%ecx    ; bt   %%ecx,%1    ; "
6          "   adcl $0,   %0 ; test %%ecx,%%ecx ; "
7          "   jnz  0b       ;                    "
8
9          : "=&r" (t) : "r" (x) : "%ecx", "cc" );
10
11   return t;
12 }
```

- Answer: both compute the Hamming weight of a 32-bit integer `x`.
  - C doesn't provide you with (direct) access to some things a processor can do.
  - For x86-32, two examples used here are the `adcl` (or "add-with-carry") and `bt` (or "bit test") instructions.
- This means understanding
  - what a processor can do irrespective of the language, and
  - use of any non-standard language features

  is an important part of writing better programs.

Some motivation for COMS12200 (7)

- Question: these two programs both compute the sum of elements in an $(n \times m)$-element matrix A; which one is faster, and why?

Listing (C)

```
1  int sum( int n, int m, int A[ n ][ m ] ) {
2    int t = 0;
3
4    for( int i = 0; i < n; i++ ) {
5      for( int j = 0; j < m; j++ ) {
6        t += A[ i ][ j ];
7      }
8    }
9
10   return t;
11 }
```

Listing (C)

```
1  int sum( int n, int m, int A[ n ][ m ] ) {
2    int t = 0;
3
4    for( int j = 0; j < m; j++ ) {
5      for( int i = 0; i < n; i++ ) {
6        t += A[ i ][ j ];
7      }
8    }
9
10   return t;
11 }
```

Some motivation for COMS12200 (7)

- Question: these two programs both compute the sum of elements in an $(n \times m)$-element matrix A; which one is faster, and why?

Listing (C)

```
1  int sum( int n, int m, int A[ n ][ m ] ) {
2    int t = 0;
3
4    for( int i = 0; i < n; i++ ) {
5      for( int j = 0; j < m; j++ ) {
6        t += A[ i ][ j ];
7      }
8    }
9
10   return t;
11 }
```

Listing (C)

```
1  int sum( int n, int m, int A[ n ][ m ] ) {
2    int t = 0;
3
4    for( int j = 0; j < m; j++ ) {
5      for( int i = 0; i < n; i++ ) {
6        t += A[ i ][ j ];
7      }
8    }
9
10   return t;
11 }
```

- Answer: the left-hand one.
  - C uses a row-major layout for 2-dimensional arrays.
  - Row-major access to `x` results in sequential access in memory, whereas column-major access doesn't; the latter is less efficient where a cache memory is present.
- This means understanding
  - how compilers work to produce low-level programs, and
  - how the memory hierarchy works

  is an important part of writing better programs.

Some motivation for COMS12200 (8)

- Question: a string data structure is important; Pascal and C take different approaches, i.e.,

$$
\begin{array}{rcccccccccc}
i & = & & \ldots, & 3, & 4, & 5, & 6, & 7, & 8, & \ldots \\
MEM & = & \langle & \ldots, & 5, & 104, & 101, & 108, & 108, & 111, & \ldots & \rangle \\
\textsc{Chr}(MEM[i]) & = & & \ldots, & ENQ, & \text{‘h’}, & \text{‘e’}, & \text{‘l’}, & \text{‘l’}, & \text{‘o’}, & \ldots
\end{array}
$$

and

$$
\begin{array}{rcccccccccc}
i & = & & \ldots, & 3, & 4, & 5, & 6, & 7, & 8, & \ldots \\
MEM & = & \langle & \ldots, & 104, & 101, & 108, & 108, & 111, & 0, & \ldots & \rangle \\
\textsc{Chr}(MEM[i]) & = & & \ldots, & \text{‘h’}, & \text{‘e’}, & \text{‘l’}, & \text{‘l’}, & \text{‘o’}, & NUL, & \ldots
\end{array}
$$

but how was the choice made during development of C?

Some motivation for COMS12200 (8)

- Question: a string data structure is important; Pascal and C take different approaches, i.e.,

$$\begin{array}{rcccccccccc} i & = & & \ldots, & 3, & 4, & 5, & 6, & 7, & 8, & \ldots \\ MEM & = & \langle & \ldots, & 5, & 104, & 101, & 108, & 108, & 111, & \ldots & \rangle \\ \textsc{Chr}(MEM[i]) & = & & \ldots, & ENQ, & \text{'h'}, & \text{'e'}, & \text{'l'}, & \text{'l'}, & \text{'o'}, & \ldots \end{array}$$

  and

$$\begin{array}{rcccccccccc} i & = & & \ldots, & 3, & 4, & 5, & 6, & 7, & 8, & \ldots \\ MEM & = & \langle & \ldots, & 104, & 101, & 108, & 108, & 111, & 0, & \ldots & \rangle \\ \textsc{Chr}(MEM[i]) & = & & \ldots, & \text{'h'}, & \text{'e'}, & \text{'l'}, & \text{'l'}, & \text{'o'}, & NUL, & \ldots \end{array}$$

  but how was the choice made during development of C?

- Answer: *many* reasons, e.g.,
  1. memory footprint was a real issue, so allowing larger strings via a larger P-string length was unattractive,
  2. the PDP range of computers already had an ASCIIZ string "type" so the C-string data structure was partly dictated by hardware.

Some motivation for COMS12200 (9)

- Question have a look at this 1956 advert for the UNIVAC computer

  http://www.youtube.com/watch?v=Pd63MHGQygQ

  and compare it with modern adverts of the same type.

Some motivation for COMS12200 (9)

- Question have a look at this 1956 advert for the UNIVAC computer

  http://www.youtube.com/watch?v=Pd63MHGQygQ

  and compare it with modern adverts of the same type.
- Observation: users and terminology have changed a *lot*

UNIVAC	Intel Core2 Duo
... takes business statistics from magnetic tape ... processing them at phenomenal speeds ... computes payrolls electronically and produce printed cheques, over 8000 cheques an hour.	... combines two independent processor cores in one physical package ... processors run at the same frequency and share up to 6 MB of L2 cache and up to 1333 MHz Front Side Bus for truly parallel computing.

so effective communication via correct notation and terminology is important.

Conclusions

- Take away points: keep in mind that COMS12200
  1. *is* difficult and *does* demand hard work, but the pay-off is a level of knowledge that will set you apart from others, and let you do things they cannot,
  2. underpins most *other* fields in CS by delivering concrete computational platforms that can realise otherwise theoretical concepts, and
  3. *isn't* just about designing hardware, but also using it effectively through better understanding of software.

References and Further Reading

[1] A.S. Tanenbaum.
Section 1.1: Structured computer organisation.
In *Structured Computer Organisation* [4].

[2] A.S. Tanenbaum.
Section 1.2: Milestones in computer architecture.
In *Structured Computer Organisation* [4].

[3] A.S. Tanenbaum.
Section 1.3: The computer zoo.
In *Structured Computer Organisation* [4].

[4] A.S. Tanenbaum.
Structured Computer Organisation.
Prentice-Hall, 6th edition, 2012.