Computer Architecture

Daniel Page

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB. UK.
⟨csdsp@bristol.ac.uk⟩

September 5, 2025

Keep in mind there are *two* PDFs available (of which this is the latter):

1. a PDF of examinable material used as lecture slides, and
2. a PDF of non-examinable, extra material:
   - the associated notes page may be pre-populated with extra, written explaination of material covered in lecture(s), plus
   - anything with a "grey'ed out" header/footer represents extra material which is useful and/or interesting but out of scope (and hence not covered).

Notes:

Notes:

- Agenda: a non-technical introduction to
  1. unit objectives,
  2. unit organisation, and
  3. some motivation (i.e., *why* the unit exists).

Notes:

Part 1: unit objectives, i.e., the "what" (1)

- The term **computer architecture** can be explained via analogy, e.g., if

  building a house = architect + civil engineer

  then

  building a computer = *computer* architect + electrical engineer.

Notes:

https://www.bris.ac.uk/unit-programme-catalogue/UnitDetails.jsa?unitCode=COMS10015

Part 1: unit objectives, i.e., the "what" (1)

Objectives

Put simply, after completing this unit you *should* be able to understand *and* apply concepts relating to

1. how computers are designed and manufactured, e.g., how logic gates are organised to perform computation
2. how computers work, e.g., how instructions and so programs are executed
3. how computers can be used more effectively, e.g., behaviour of high-level programs wrt. low-level resources

reading

computer = computer processor + supporting infrastructure (i.e., the wider computer *system*).

https://www.bris.ac.uk/unit-programme-catalogue/UnitDetails.jsa?unitCode=COMS10015

Notes:

Part 1: unit objectives, i.e., the "what" (2)

Quote

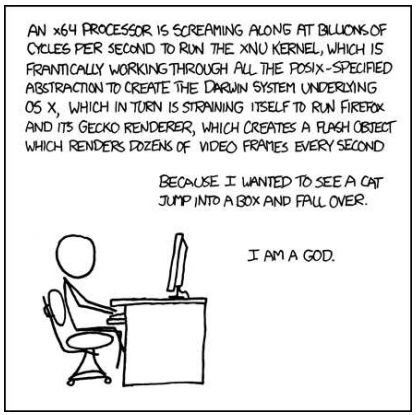
... we had asked for "computer architects"; the Royal Institute of British Architects pointed out that the word "architect" was controlled by an Act of Parliament passed in 1933, which specified that architects could design houses, landscapes or ships, but not computers.

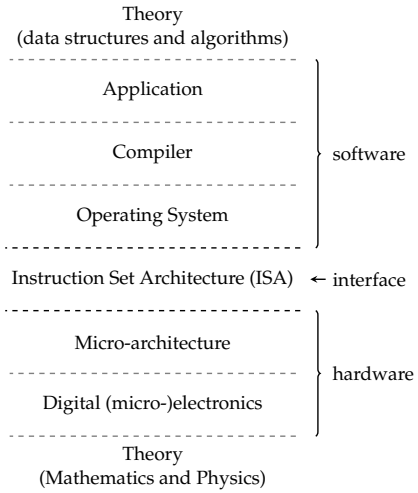
– Barron

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

Notes:

Part 1: unit objectives, i.e., the "what" (3)





https://xkcd.com/676

Notes:

- There are (at least) two approaches to teaching this content: bottom-to-top, or top-to-bottom i.e., starting at the lowest (resp. highest) layer and then, at each step, considering the next higher (resp. lower) layer. We opt for the former, and thus start with the lowest layer then work upwards toward the highest. The rationale for this choice is that at each step, you understand *everything* up to that point: each step extends your knowledge, vs. the alternative which means filling in the previously unexplained.
- The focus and content of each layer naturally differs, and it is important to keep this in mind:
  - It could be the case that a given layer is more or less *interesting* to you, depending on your background and goals. For example, some students might be more familiar with and pursue a career in developing application software: the higher layers are naturally might aligned with this.
  - It could be the case that a given layer is more or less *difficult* for you, depending on your background and goals. For example, unlike some units where material in earlier lectures is introductory with respect to later lectures, it isn't true that the lower layers are "easier" per se: they simply support the higher layers.

  Fundamentally, it is important to realise that (modulo special cases, e.g., where an operating system is not required) fully understanding a modern computer system demands understanding of *all* the layers to some extent.

Part 2: unit organisation, i.e., the "how" (1)

- Important:

1. The unit is delivered by the following members of (academic) staff

   Tom Deakin ⇒ Lecturer and Unit Director
   Daniel Page ⇒ Lecturer

   supplemented by, e.g., a wider team who act in/as Teaching Support Roles (TSRs).

Notes:

Part 2: unit organisation, i.e., the "how" (1)

- Important:

2. At a high(er) level, the unit is delivered as a set of themes:

   Theme #1 ⇒ "from Mathematics and Physics to digital logic"
   Theme #2 ⇒ "from digital logic to computer processors"
   Theme #3 ⇒ "from computer processors to software applications"

3. At a low(er) level, the unit involves the following activities:

   lecture slot ⇒ synchronous, i.e., timetabled
   ⇒ in-person

   lab. slot ⇒ synchronous, i.e., timetabled
   ⇒ in-person

   drop-in slot ⇒ synchronous, i.e., timetabled
   ⇒ in-person and online, i.e., hybrid

https://www.bristol.ac.uk/timetables/TimetablePDF.pdf?unit=COMS10015

Notes:

Part 2: unit organisation, i.e., the "how" (1)

- Important:

3. The *summative* assessment for this unit includes

   summative coursework assignment ⇝ TB1, week 10
   ↦ 30% weight = 6CP
   summative exam ⇝ TB2, assessment period
   ↦ 70% weight = 14CP

4. The *formative* assessment for this unit includes

   formative exam #1 ⇝ TB1, week 6
   formative exam #2 ⇝ TB1, week 12
   formative exam #3 ⇝ TB2, week 18
   formative exam #4 ⇝ TB2, week 24

   none of which is credit bearing, i.e., it has 0% weight = 0CP.

© Daniel Page ⟨...⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

Notes:

- Important:

5. *Everything* related to the unit is accessible via *either*
   - the *internal*-facing Blackboard-based unit web-site

     `https://www.ole.bris.ac.uk`

   or
   - the *external*-facing GitHub-based unit web-site

     `https://cs-uob.github.io/COMS10015`

   or, more specifically,

unit-wide communication, e.g., announcements	⇒	Blackboard	} *internal*-facing
assessment submission, marks, and feedback	⇒	Blackboard	
discussion forum	⇒	Teams	
teaching material	⇒	GitHub	} *external*-facing

Notes:

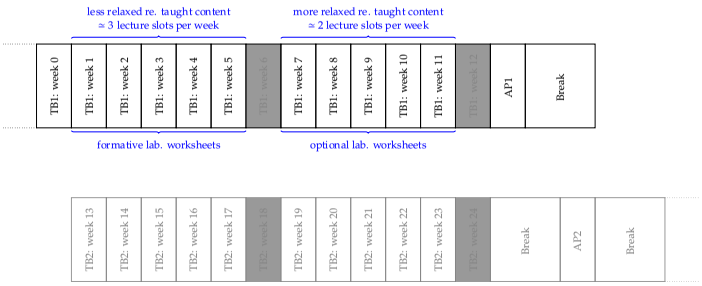
TB1	TB2
week 0	week 13
week 1	week 14
week 2	week 15
week 3	week 16
week 4	week 17
week 5	week 18
week 6	week 19
week 7	week 20
week 8	week 21
week 9	week 22
week 10	week 23
week 11	week 24
week 12	
AP1	Break
Break	AP2
	Break

Notes:

© Daniel Page ⟨cdp@cs.bris.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

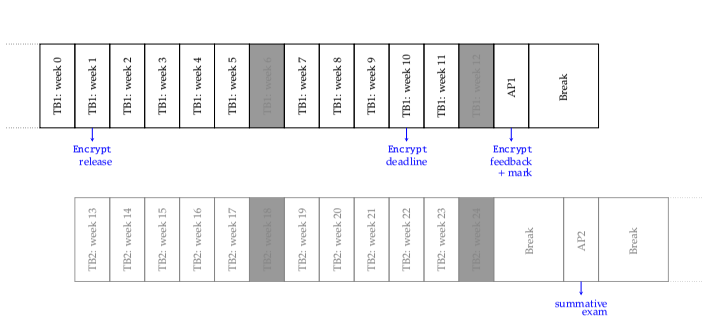
Notes:

Part 2: unit organisation, i.e., the “how” (2)

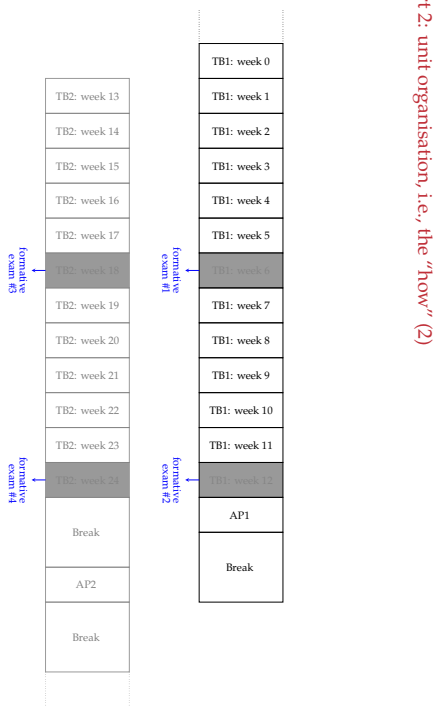


Notes:

Part 2: unit organisation, i.e., the “how” (2)



Part 2: unit organisation, i.e., the "how" (2)



Notes:

Part 3: unit motivation, i.e., the "why" (1)

Computer architecture as hardware design

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

Notes:

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

- Answer: probably ARM, because the embedded processor market is so large;
  1. keep in mind that

     $$\text{computer} = \{\text{desktop computer}, \text{laptop computer}, \text{embedded computer}, \ldots\},$$

     i.e., there's a *lot* more to the topic than traditional models of computing, and
  2. *knowledge driven* industries demand you "understand" not just "do".

Notes:

- Question:
  1. what *is* this device, and
  2. who is the designer and/or vendor?

Notes:

Part 3: unit motivation, i.e., the "why" (2)
Computer architecture as hardware design

- Question:
  1. what *is* this device, and
  2. who is the designer and/or vendor?

- Answer: this is an *open*-hardware based laptop called Novena [7];
  - designed by Andrew "bunnie" Huang and Sean "xobs" Cross,
  - houses an ARM-based processor plus Xilinx-based FPGA,
  - raised ~ $780,000 via CrowdSupply campaign: you can buy one,
  - if equipped with the right set of skills, you can design and manufacture something similar.

https://spectrum.ieee.org/consumer-electronics/portable-devices/novena-a-laptop-with-no-secrets

Notes:

Part 3: unit motivation, i.e., the "why" (3)
High-level applications of computer architecture

> **Quote**
>
> *People who are really serious about software should make their own hardware.*
>
> *– Kay (https://en.m.wikiquote.org/wiki/Alan_Kay)*

Notes:

Part 3: unit motivation, i.e., the "why" (4)
High-level applications of computer architecture

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

Notes:

- The games are (from left-to-right):
  - Night Driver (Atari 2600, 1978),
  - Pole Position (Atari 2600, 1983),
  - Out Run (Commodore Amiga, 1989),
  - Indianapolis 500 (Commodore Amiga, 1990), and
  - Gran Turismo (PlayStation 1998).

Part 3: unit motivation, i.e., the "why" (4)
High-level applications of computer architecture

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

Notes:

- The games are (from left-to-right):
  - Night Driver (Atari 2600, 1978),
  - Pole Position (Atari 2600, 1983),
  - Out Run (Commodore Amiga, 1989),
  - Indianapolis 500 (Commodore Amiga, 1990), and
  - Gran Turismo (PlayStation 1998).

© Daniel Page ⟨cdsapp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

Part 3: unit motivation, i.e., the "why" (4)
High-level applications of computer architecture

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

Notes:

- The games are (from left-to-right):
  - Night Driver (Atari 2600, 1978),
  - Pole Position (Atari 2600, 1983),
  - Out Run (Commodore Amiga, 1989),
  - Indianapolis 500 (Commodore Amiga, 1990), and
  - Gran Turismo (PlayStation 1998).

Part 3: unit motivation, i.e., the "why" (4)
High-level applications of computer architecture

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

Notes:

- The games are (from left-to-right):
  - Night Driver (Atari 2600, 1978),
  - Pole Position (Atari 2600, 1983),
  - Out Run (Commodore Amiga, 1989),
  - Indianapolis 500 (Commodore Amiga, 1990), and
  - Gran Turismo (PlayStation 1998).

© Daniel Page (cdsp@cs.bris.ac.uk)
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

Part 3: unit motivation, i.e., the "why" (4)
High-level applications of computer architecture

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

Notes:

- The games are (from left-to-right):
  - Night Driver (Atari 2600, 1978),
  - Pole Position (Atari 2600, 1983),
  - Out Run (Commodore Amiga, 1989),
  - Indianapolis 500 (Commodore Amiga, 1990), and
  - Gran Turismo (PlayStation 1998).

Part 3: unit motivation, i.e., the "why" (4)
High-level applications of computer architecture

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

Notes:

- The games are (from left-to-right):
  - Night Driver (Atari 2600, 1978),
  - Pole Position (Atari 2600, 1983),
  - Out Run (Commodore Amiga, 1989),
  - Indianapolis 500 (Commodore Amiga, 1990), and
  - Gran Turismo (PlayStation 1998).

© Daniel Page ⟨cspage@cs.bris.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

Part 3: unit motivation, i.e., the "why" (4)
High-level applications of computer architecture

- 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)

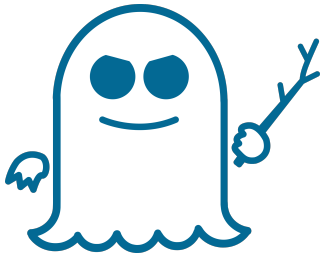
  suggesting computer architecture evolves symbiotically with other fields.

Notes:

- The games are (from left-to-right):
  - Night Driver (Atari 2600, 1978),
  - Pole Position (Atari 2600, 1983),
  - Out Run (Commodore Amiga, 1989),
  - Indianapolis 500 (Commodore Amiga, 1990), and
  - Gran Turismo (PlayStation 1998).

Part 3: unit motivation, i.e., the "why" (5)
High-level applications of computer architecture

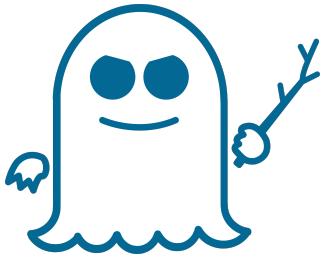
- Question: what do these logos relate to?

Notes:

Part 3: unit motivation, i.e., the "why" (5)
High-level applications of computer architecture

- Question: what do these logos relate to?



- Answer: the Meltdown [9] and Spectre [8] security vulnerabilities, which
  1. can be broadly classified as micro-architectural side-channel attacks, and, as such,
  2. demand deep understanding of processor (micro-)architecture to either
     - mount (i.e., use), *or*
     - prevent.

https://www.meltdownattack.com/meltdown.png
https://www.spectreattack.com/spectre.png

Notes:

Part 3: unit motivation, i.e., the "why" (6)
High-level applications of computer architecture

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

Notes:

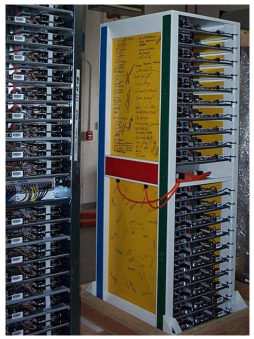
- Question: ranging from the late 1990*s* to the early 2000*s*, spot the difference(s).

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

Notes:

Part 3: unit motivation, i.e., the "why" (6)
High-level applications of computer architecture

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



© Daniel Page ⟨cdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

Notes:

Part 3: unit motivation, i.e., the "why" (6)
High-level applications of computer architecture

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



Notes:

Part 3: unit motivation, i.e., the "why" (6)
High-level applications of computer architecture

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

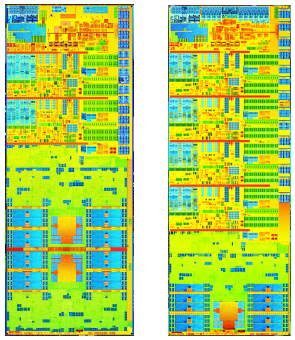


- Answer:
  - left-hand picture is *the* Google data center circa 1997,
  - right-hand picture is *a* Google data center circa 2007

  suggesting efficient software stacks demand care with respect to computer architecture: a holistic approach to CS is unavoidable.

Notes:

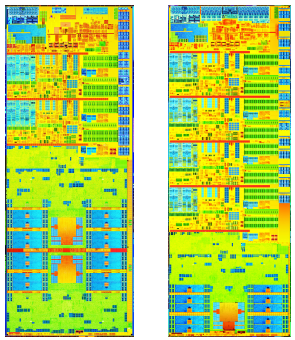
- Question: identify these objects, and spot the difference(s).



Notes:

Part 3: unit motivation, i.e., the "why" (7)
Low-level applications of computer architecture

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



- Answer:
  - the left-hand picture is a dual-core Haswell model Intel processor, whereas
  - the right-hand picture is a quad-core Haswell model Intel processor,

  which highlights a trend: parallelism and concurrency are inherent, suggesting that understanding, coping with, and exploiting them are all vital.

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

Notes:

- Question: given the list of operations (circa 2010)

  L1 cache access
  Branch misprediction
  L2 cache access
  Memory access
  Read 1 MB sequentially from memory
  IP packet round-trip (local area network)
  Disk seek
  Read 1 MB sequentially from disk
  IP packet round-trip (internet)

  estimate the associated latencies (i.e., how long they take)?

Notes:

Part 3: unit motivation, i.e., the "why" (8)
Low-level applications of computer architecture

- Question: given the list of operations (circa 2010)

L1 cache access	0.5 ns
Branch misprediction	5.0 ns
L2 cache access	7.0 ns
Memory access	100.0 ns
Read 1 MB sequentially from memory	250000.0 ns
IP packet round-trip (local area network)	500000.0 ns
Disk seek	10000000.0 ns
Read 1 MB sequentially from disk	20000000.0 ns
IP packet round-trip (internet)	150000000.0 ns

  estimate the associated latencies (i.e., how long they take)?

https://research.google.com/people/jeff/Stanford-DL-Nov-2010.pdf

© Daniel Page ⟨cs-ga@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

Notes:

- Question: given the list of operations (circa 2010)

L1 cache access	0.5 ns	=	$5.0 \cdot 10^{-10}$ s
Branch misprediction	5.0 ns	=	$5.0 \cdot 10^{-9}$ s
L2 cache access	7.0 ns	=	$7.0 \cdot 10^{-9}$ s
Memory access	100.0 ns	=	$1.0 \cdot 10^{-7}$ s
Read 1 MB sequentially from memory	250000.0 ns	=	$2.5 \cdot 10^{-4}$ s
IP packet round-trip (local area network)	500000.0 ns	=	$5.0 \cdot 10^{-4}$ s
Disk seek	10000000.0 ns	=	$1.0 \cdot 10^{-2}$ s
Read 1 MB sequentially from disk	20000000.0 ns	=	$2.0 \cdot 10^{-2}$ s
IP packet round-trip (internet)	150000000.0 ns	=	$1.5 \cdot 10^{-1}$ s

estimate the associated latencies (i.e., how long they take)?

https://research.google.com/people/jeff/Stanford-DL-Nov-2010.pdf

Notes:

Part 3: unit motivation, i.e., the "why" (8)
Low-level applications of computer architecture

- Question: given the list of operations (circa 2010)

L1 cache access	0.5 ns	=	$5.0 \cdot 10^{-10}$ s	≃	1 seconds
Branch misprediction	5.0 ns	=	$5.0 \cdot 10^{-9}$ s	≃	10 seconds
L2 cache access	7.0 ns	=	$7.0 \cdot 10^{-9}$ s	≃	10 seconds
Memory access	100.0 ns	=	$1.0 \cdot 10^{-7}$ s	≃	3 minutes
Read 1 MB sequentially from memory	250000.0 ns	=	$2.5 \cdot 10^{-4}$ s	≃	5 days
IP packet round-trip (local area network)	500000.0 ns	=	$5.0 \cdot 10^{-4}$ s	≃	11 days
Disk seek	10000000.0 ns	=	$1.0 \cdot 10^{-2}$ s	≃	231 days
Read 1 MB sequentially from disk	20000000.0 ns	=	$2.0 \cdot 10^{-2}$ s	≃	462 days
IP packet round-trip (internet)	150000000.0 ns	=	$1.5 \cdot 10^{-1}$ s	≃	3472 days

estimate the associated latencies (i.e., how long they take)?

- Answer: these operations stem from standard components;
  - knowing what they *are*, plus
  - understanding how they *work*, i.e., why the absolute *and* relative latencies are as listed,

is an important step toward using them effectively, or improving their design or implementation.

https://research.google.com/people/jeff/Stanford-DL-Nov-2010.pdf

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

Notes:

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

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

Notes:

- Question: these are 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 }
```

Notes:

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

Listing (C)

```c
int weight( uint32_t x ) {
  int t = 0;

  for( int i = 0; i < 32; i++ ) {
    if( ( x >> i ) & 1 ) {
      t += 1;
    }
  }

  return t;
}
```

Listing (C)

```c
int weight( uint32_t x ) {
  int t;

  asm ( "   movl $0,   %0 ; movl $32,  %%ecx ; "
        "0: decl %%ecx    ; bt   %%ecx,%1    ; "
        "   adcl $0,   %0 ; test %%ecx,%%ecx ; "
        "   jnz  0b       ;                    "

        : "=&r" (t) : "r" (x) : "%ecx", "cc" );

  return t;
}
```

- 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,
  - two examples here are `adcl` (or "add-with-carry") and `bt` (or "bit test") instructions

  which means understanding
  - what a processor can do irrespective of the language, and
  - use of any non-standard language features

  are important aspects of writing better programs.

Notes:

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

Listing (C)

```c
int sum( int n, int m, int A[ n ][ m ] ) {
  int t = 0;

  for( int i = 0; i < n; i++ ) {
    for( int j = 0; j < m; j++ ) {
      t += A[ i ][ j ];
    }
  }

  return t;
}
```

Listing (C)

```c
int sum( int n, int m, int A[ n ][ m ] ) {
  int t = 0;

  for( int j = 0; j < m; j++ ) {
    for( int i = 0; i < n; i++ ) {
      t += A[ i ][ j ];
    }
  }

  return t;
}
```


Notes:

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

Listing (C)

```c
int sum( int n, int m, int A[ n ][ m ] ) {
  int t = 0;

  for( int i = 0; i < n; i++ ) {
    for( int j = 0; j < m; j++ ) {
      t += A[ i ][ j ];
    }
  }

  return t;
}
```

Listing (C)

```c
int sum( int n, int m, int A[ n ][ m ] ) {
  int t = 0;

  for( int j = 0; j < m; j++ ) {
    for( int i = 0; i < n; i++ ) {
      t += A[ i ][ j ];
    }
  }

  return t;
}
```

- 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, which is more efficient if a cache memory is used

  which means understanding
  - how compilers work to produce low-level programs, and
  - how the memory hierarchy works

  are important aspects of writing better programs.

Notes:

- 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)

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

Listing (C)

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


Notes:

- 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,
  - `printf` has a user-space buffer when `stdout` is connected to a terminal, which reduces the number of system calls (i.e., interaction with the kernel)

  which means understanding
  - how your program interfaces with the the kernel, and
  - how the kernel interfaces with physical hardware

  are important aspects of writing better programs.

Notes:

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

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

and

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

but how was the choice made during development of C?

Notes:

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

i	=		...,	3,	4,	5,	6,	7,	8,	...	
MEM	=	⟨	...,	5,	104,	101,	108,	108,	111,	...	⟩
Chr(MEM[i])	=		...,	*ENQ*,	'h',	'e',	'l',	'l',	'o',	...	

and

i	=		...,	3,	4,	5,	6,	7,	8,	...	
MEM	=	⟨	...,	104,	101,	108,	108,	111,	0,	...	⟩
Chr(MEM[i])	=		...,	'h',	'e',	'l',	'l',	'o',	*NUL*,	...	

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, and
  2. the PDP-11 had native support for ASCIIZ strings, meaning the choice of C-string was partly dictated by hardware,

  suggesting that understanding *historical* design decisions can be relevant and useful in *modern* contexts!

Notes:

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

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

  and compare it with modern adverts of the same type.

Notes:

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

  https://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 6MB of L2 cache and up to 1333MHz Front Side Bus for truly parallel computing.

suggesting that effective communication via correct notation and terminology is important.

Notes:

Conclusions

- Take away points: this unit
  1. *is* challenging and *does* demand hard work,
  2. *isn't* exclusively about hardware, and
  3. will equip you with the knowledge and skills required to
     - design better hardware,
     - design better software,
     - design better systems (e.g., combinations of hardware and software),
     - make better trade-offs,
     - understand complex behaviour,
     - debug complex behaviour,
     - think critically,
     - think "in parallel",
     - ...

  and, ultimately, solve important and interesting problems (in the short- and longer-term).

Notes:

Additional Reading

- *Wikipedia: Computer architecture*. URL: https://en.wikipedia.org/wiki/Computer_architecture.
- A.S. Tanenbaum and T. Austin. "Section 1.1: Structured computer organisation". In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012.
- A.S. Tanenbaum and T. Austin. "Section 1.2: Milestones in computer architecture". In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012.
- A.S. Tanenbaum and T. Austin. "Section 1.3: The computer zoo". In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012.
- W. Stallings. "Chapter 2: Computer evolution and performance". In: *Computer Organisation and Architecture*. 9th ed. Prentice Hall, 2013.
- W. Stallings. "Chapter 3: A top-level view of computer function and interconnection". In: *Computer Organisation and Architecture*. 9th ed. Prentice Hall, 2013.

Notes:

References

[1] *Wikipedia: Computer architecture*. URL: https://en.wikipedia.org/wiki/Computer_architecture (see p. 105).

[2] W. Stallings. "Chapter 2: Computer evolution and performance". In: *Computer Organisation and Architecture*. 9th ed. Prentice Hall, 2013 (see p. 105).

[3] W. Stallings. "Chapter 3: A top-level view of computer function and interconnection". In: *Computer Organisation and Architecture*. 9th ed. Prentice Hall, 2013 (see p. 105).

[4] A.S. Tanenbaum and T. Austin. "Section 1.1: Structured computer organisation". In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012 (see p. 105).

[5] A.S. Tanenbaum and T. Austin. "Section 1.2: Milestones in computer architecture". In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012 (see p. 105).

[6] A.S. Tanenbaum and T. Austin. "Section 1.3: The computer zoo". In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012 (see p. 105).

[7] A. Huang and S. Cross. "Novena: a laptop with no secrets". In: *IEEE Spectrum* 52.11 (2015), pp. 40–56 (see pp. 35, 37).

[8] P. Kocher et al. "Spectre Attacks: Exploiting Speculative Execution". In: *IEEE Symposium on Security and Privacy*. 2019, pp. 19–37 (see pp. 55, 57).

[9] M. Lipp et al. "Meltdown: Reading Kernel Memory from User Space". In: *USENIX Security Symposium*. 2018, pp. 973–990 (see pp. 55, 57).

Notes: