CS2002 Logic and Architecture
Lecture 1
Introduction to Architecture

Ian Gent

Credits: Logic and Architecture

These slides have developed over several years

Thanks for contributions to slides to Juliana Bowles, Ian Gent, Ruth Hoffmann, Chris Jefferson, Steve Linton, Mark-Jan Nederhof, Alex Voss

Of course any mistakes in the slides are my fault!

Aims: Architecture

- overview of computer architecture from programmer's (rather than hardware designer's) perspective
- logic on which computer architecture is based
- highlight aspects that impact on performance of program execution
- so you can write code that is not just correct but efficient and makes good use of modern hardware

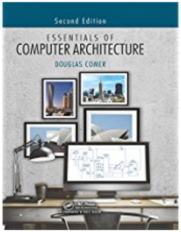
Topics: Architecture

- von Neumann architecture (ALU, control, registers, memory, I/O)
- assembler programming, instruction set architecture (ISA), RISC & CISC architectures, addressing modes
- memory management, stacks and subroutines
- memory hierarchy, caches and paging
- instruction-level parallelism, speculative execution & branch prediction

Books

Most material is covered in this book but not in same order:

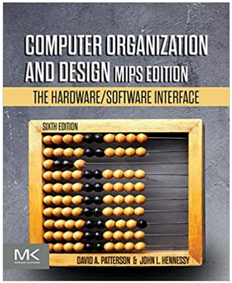
- D. Comer, **Essentials of Computer Architecture**, Pearson, 2nd ed, 2017 (available as e-book via SAULCAT)



Books (cont.)

Optional:

- D. Patterson & J. Hennessy, **Computer Organization and Design – The Hardware/Software Interface**, 6th ed, 2020 (other editions exist with emphasis on ARM or RISC-V rather than MIPS)


COMPUTER ORGANIZATION AND DESIGN MIPS EDITION
THE HARDWARE/SOFTWARE INTERFACE
SIXTH EDITION
MK
DAVID A. PATTERSON & JOHN L. HENNESSY

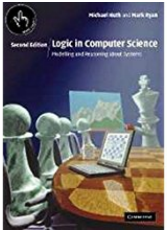
(Much) further reading (for CS4202):

- J. Hennessy & D. Patterson, **Computer Architecture – A Qualitative Approach**, 6th ed, 2017

Books (cont.)

For lectures on logic, helpful may also be Chapter 1 of:

- M. Huth & M. Ryan, **Logic in Computer Science – Modelling and Reasoning about Systems**, 2nd ed, 2004



Ways to interpret sequences of bits

CS1003:

- as logical trues and falses
- as integers, signed or unsigned
- as floating point numbers
- as characters and strings

CS2002:

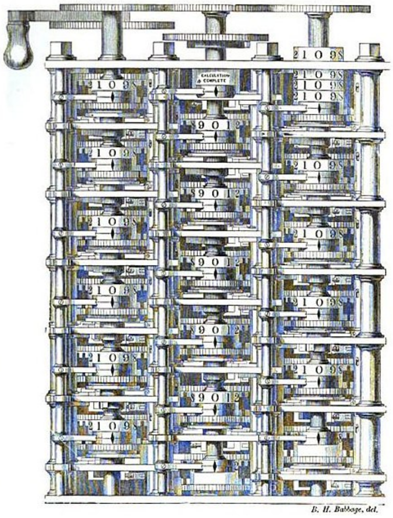
- as instructions telling the computer what to do
- as “addresses” identifying particular piece of memory

Also recall:

- binary, octal and hexadecimal representations of sequences of bits

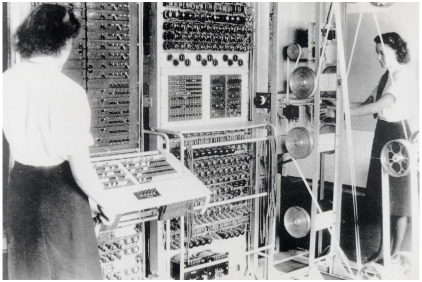
Some computer history

- **Difference Engine** (Charles Babbage)
- proposed 1820s, partially built 1830s
- full Difference Engine No. 2 only realised in 1991
- intended to compute log and trig tables
- “program” by setting mechanical wheels in position



Some computer history (cont.)

- **Colossus** (1943-1945)
- at Bletchley Park
- used valves (vacuum tubes)
- programmed by plugboard and paper tape

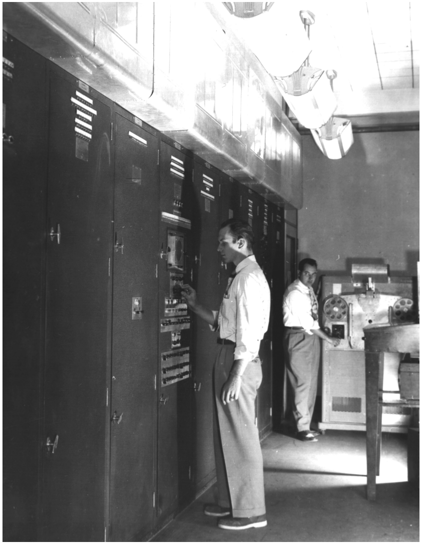


Some computer history (cont.)

- **IBM 407** Accounting Machine (1949)
- punched cards
- program by manual wiring of control panel

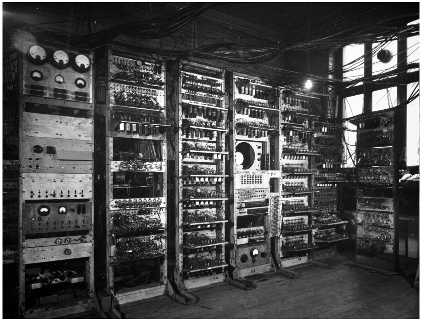
Stored program computing

- John von Neumann, **First Draft of a Report on the EDVAC** (1945)
- EDVAC was delivered in 1949
- same storage could be used for numbers and instructions saying what computations to do with these numbers
- invented software!
- long debate to be had about who was first. . .



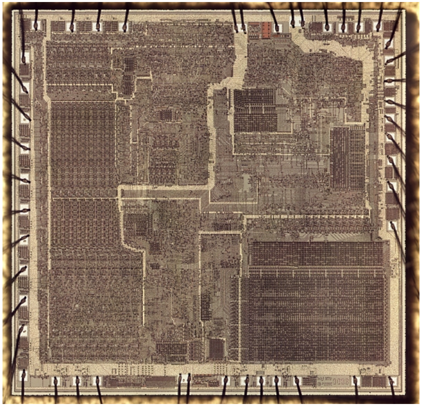
Stored program computing (cont.)

- Long debate to be had about who was first. . .
- Strong contender is the **Manchester Baby**
- Ran a stored program on 21 June 1948
- Designed by F.C. Williams, Tom Kilburn and Geoff Tootill
- The circular screen you can see was actually storing memory, not for output



Transistors and Moore's Law

- First transistor patent on 26 June 1948
- Later came integrated circuit's (IC) and **Moore's Law**
- Transistors on an IC approximately double every two years
- By 1978 Intel could produce the 8086 processor with 29,000 transistors
- Transistor counts have gone up $\times 1,000,000$



Turing on universal computing devices

“It is intended that the setting up of the machine for new problems shall be virtually only a matter of paper work. . . There will be positively no internal alterations to be made even if we wish suddenly from calculating the energy levels of the neon atom to the enumeration of groups of order 720.

Instruction tables will have to be made up by mathematicians with computing experience and perhaps a certain puzzle-solving ability. There will probably be a great deal of work of this kind to be done, for every known process has got to be translated into instruction table form at some stage.

Turing on universal computing devices (cont.)

The process of constructing instruction tables should be very fascinating. There need be no real danger of it ever becoming a drudge, for any processes that are quite mechanical may be turned over to the machine itself."

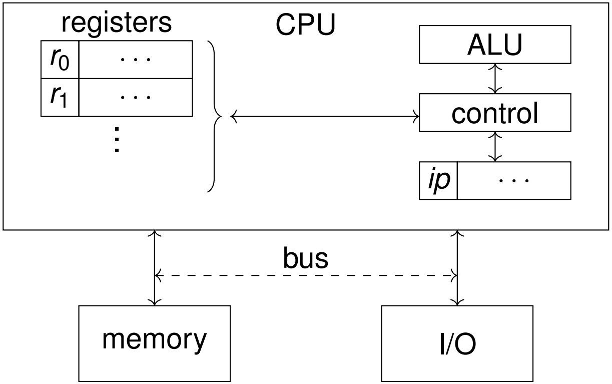
A. M. Turing, **Proposed Electronic Calculator**, report for National Physical Laboratory, Teddington, 1946

Turing on programming languages (and logic)

> *"Actually one could communicate with these machines in any language provided it was an exact language, i.e. in principle one should be able to communicate in any symbolic logic, provided that the machine were given instruction tables which would allow it to interpret that logical system. . . "*

A. M. Turing, **Lecture to the London Mathematical Society**, 20 February 1947

von Neumann architecture



Distinguishing feature is that instructions and data occupy same memory

Fetch-execute cycle

Done by CPU (**central processing unit**)

Repeat forever:

1. load data from memory address in *ip* (**instruction pointer**)
2. adjust *ip* to refer to next instruction
3. control unit decodes data as an instruction
4. instruction is carried out, e.g. by ALU (**arithmetic logic unit**)
5. (some instructions change *ip*)

Instruction set architecture (ISA)

. . . or just **instruction set**

Set of instructions understood by a CPU

- their encodings into bits
- their meanings

Single ISA can have many implementations

- e.g. AMD make x86 chips
- many manufacturers make ARM chips

ISA is an interface, not a hardware description

Example ISA: x86

ISAs live much longer than CPU hardware designs

- x86 ISA started with 8086 chip released in 1978
- changed many times since
- but with backwards compatibility

- Intel/AMD can (and do) completely change how instructions are implemented
- as long as the same interface is maintained

Example x86 instruction

Suppose *ip* holds the address of the first of the following four bytes (in hex), at the start of the fetch-execute cycle:

```
    48 6b 08 56
```

Rendered for humans as (**AT&T assembler syntax**):

```
    imulq $86, (%rax), %rcx
```

Or (**Intel syntax**):

```
    imul rcx, qword ptr [rax], 86
```

Example x86 instruction (cont.)

Let us dissect:

```
imulq $86, (%rax), %rcx
```

or

```
imul rcx, qword ptr [rax], 86
```

- `imul` = signed integer multiply
- `q` / `qword` = quadword = 4x16 bits = 64 bit calculation
- `$86` / `86` = constant 86 (decimal)
- `%rax` / `rax` = register *rax*, a 64 bit register (one of 16)
- `%rcx` / `rcx` = register *rcx*, another 64 bit register
- `(%X)` / `ptr [X]` = value stored at memory address in X (ptr for pointer)

Example x86 instruction (cont.)

So

```
imulq $86, (%rax), %rcx
```

or

```
imul rcx, qword ptr [rax], 86
```

means:

> *Get the signed integer 64 bit value stored at the memory address in register rax, multiply it by 86, and put the result in register rcx*

Recommended reading

Comer:

- Sections 2.1–2.2 (history of computing)
- Sections 3.1–3.8, 3.11, 3.13–3.18 (computer representations of numbers)
- Sections 4.1–4.3, 4.6–4.7, 4.9–4.11, 4.14–4.15 (history of computing)

Next time

- Assembly language, assemblers, compilers
- Issues in ISA design:
    - registers
    - operands