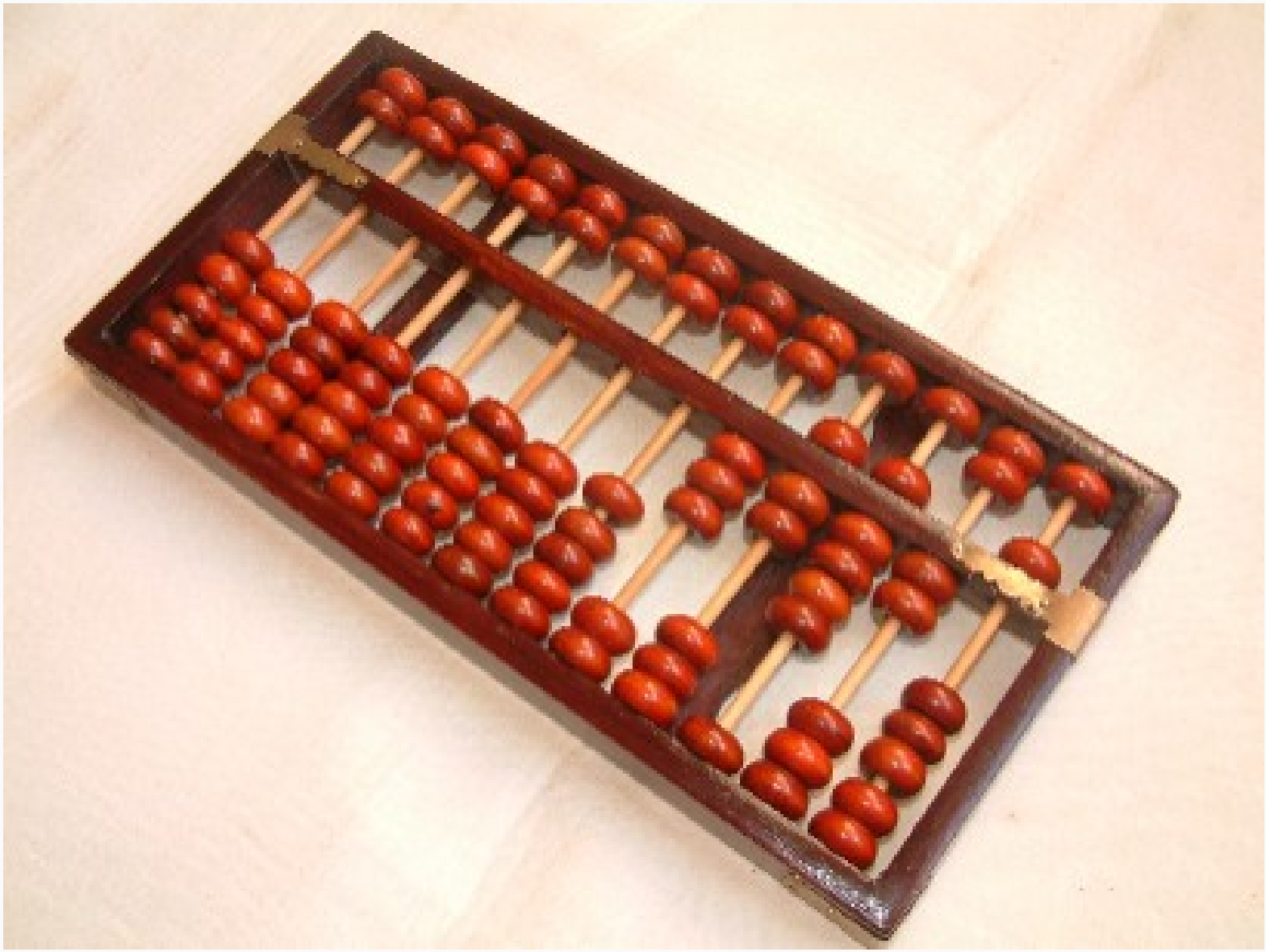
Computer Hardware 2500 BC - wood

Abacus Sumeria c. 2500 B

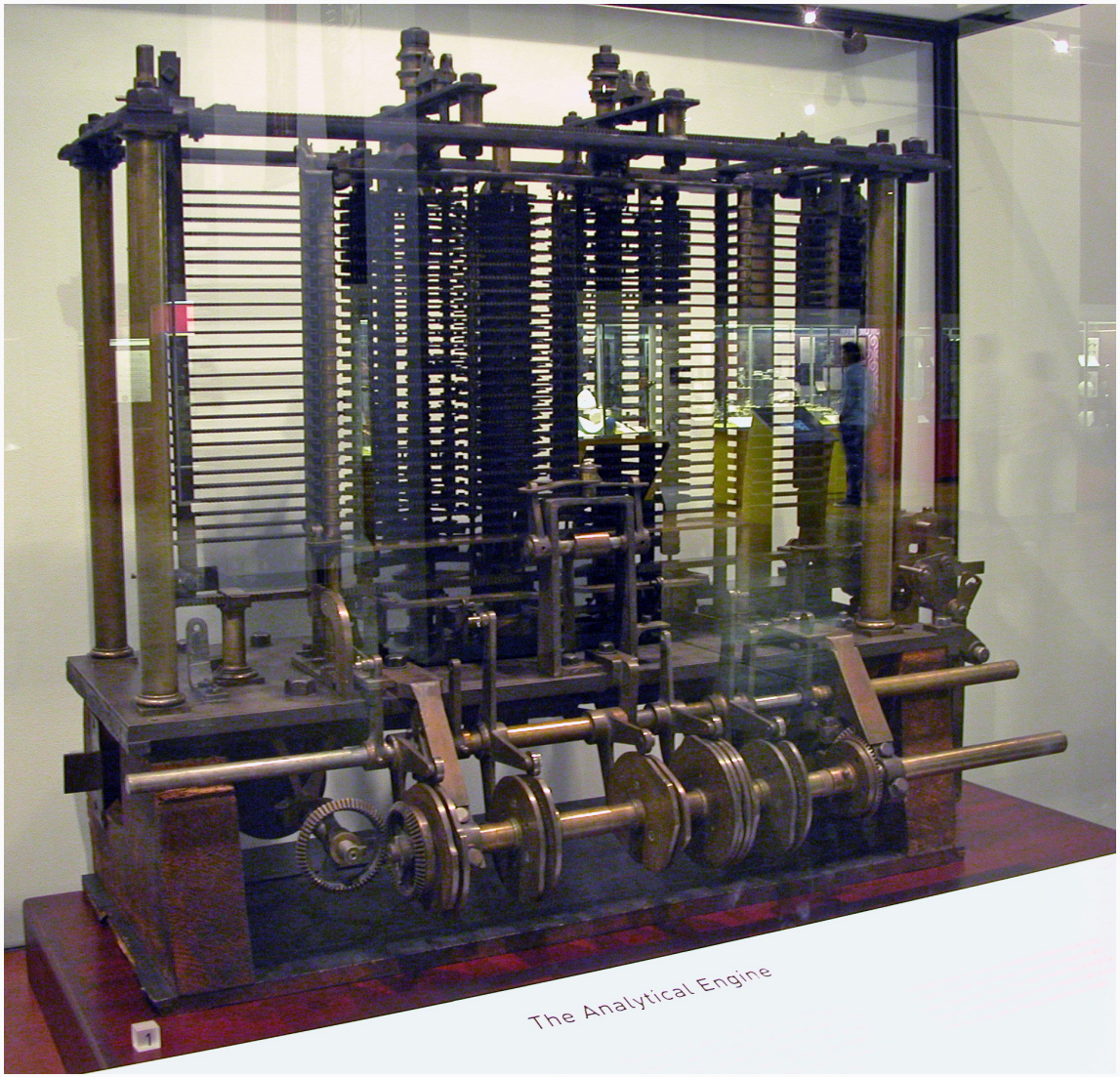


Computer Hardware 100 BC - brass

- Antikythera mechanism - special purpose analog computer
- used to predict astronomical positions and eclipses

Computer Hardware 1835 - brass & steam

- analytical Engine designed by Charles Babbage 1835 - never built.
- general purpose programmable computer using punch cards and steam power



The first Coder 1835

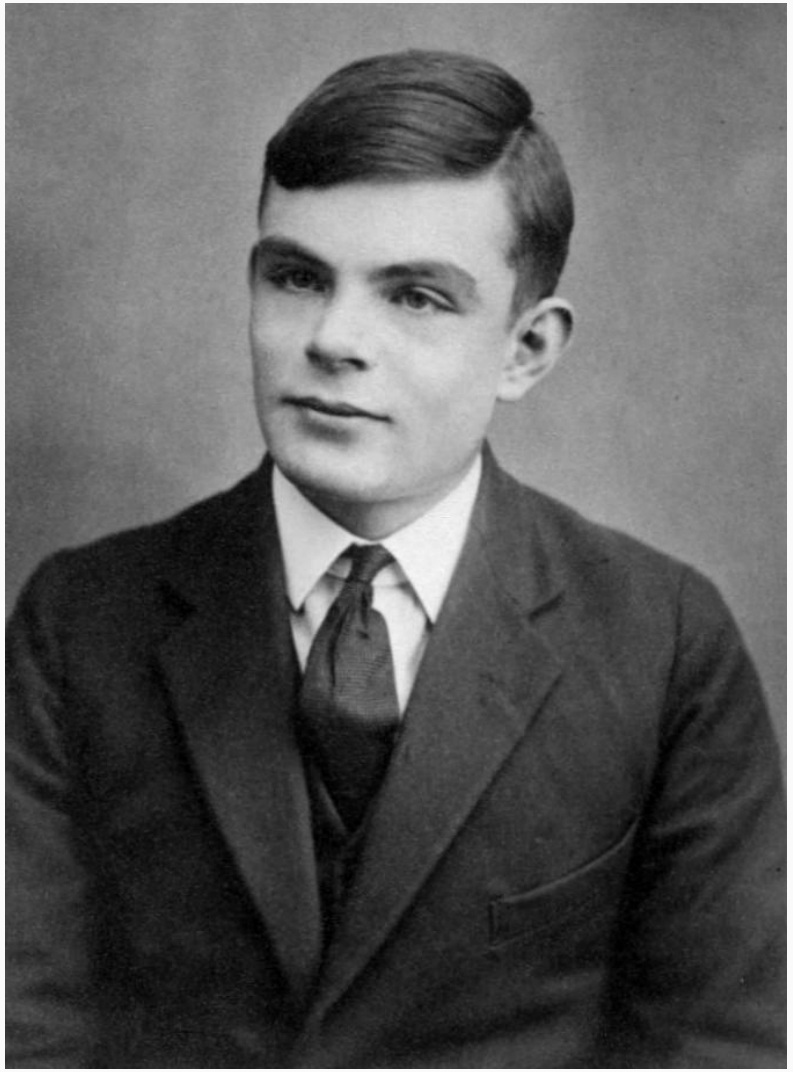
Ada Lovelace - mathematician who wrote the first programs.

Computer Hardware: 1890 - electromechanical

- hollerith tabulating machine used for US census
- company eventually becomes IBM

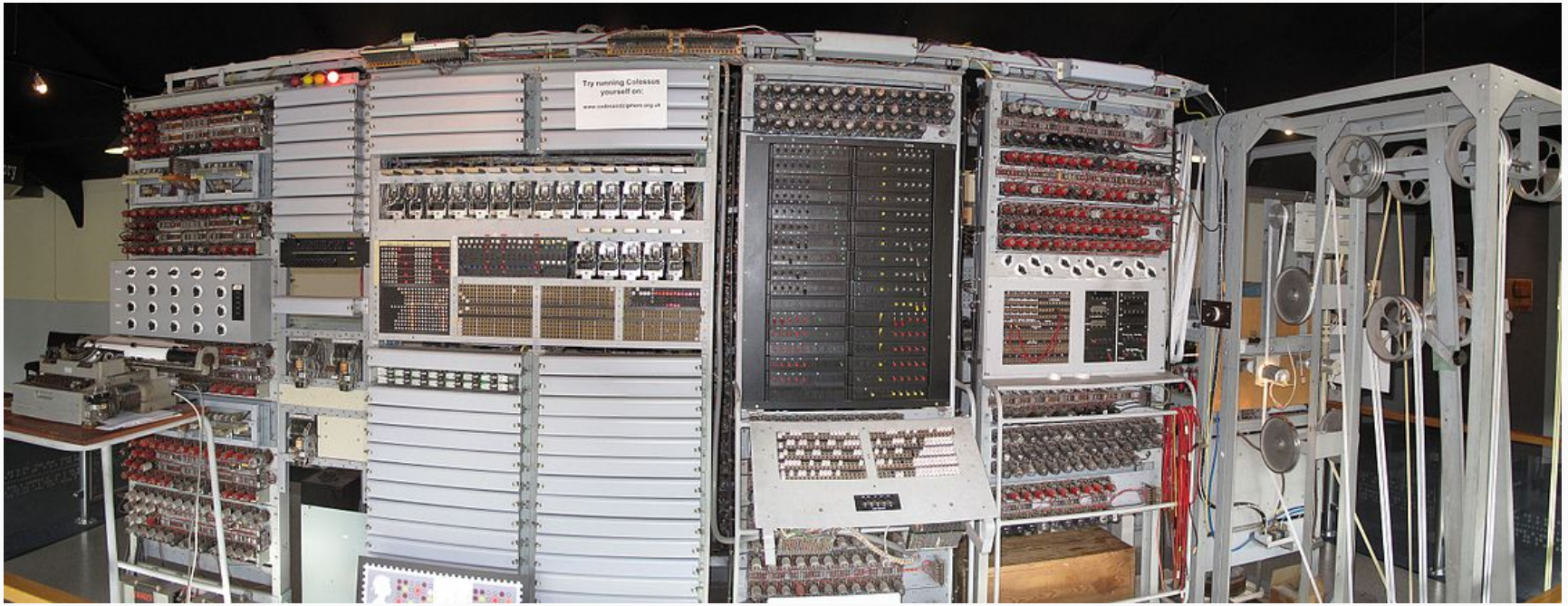
Alan Turing aged 16

- developed model of computation
  - the Turing Machine
- showed limits of computation
  - the halting problem
  - with Godel & Church, changed our understanding of the universe
- WWII codebreaker at Bletchley Park
  - watch the movie *The Imitation Game*, historically inaccurate but has Benedict Cumberatch
- invented the Turing Test
  - famous benchmark for machine intelligence
- prosecuted in 1952 for homosexual acts
- apology by British PM in 2009
- pardon from Queen in 2017
- portrait on Bank of England £50 note



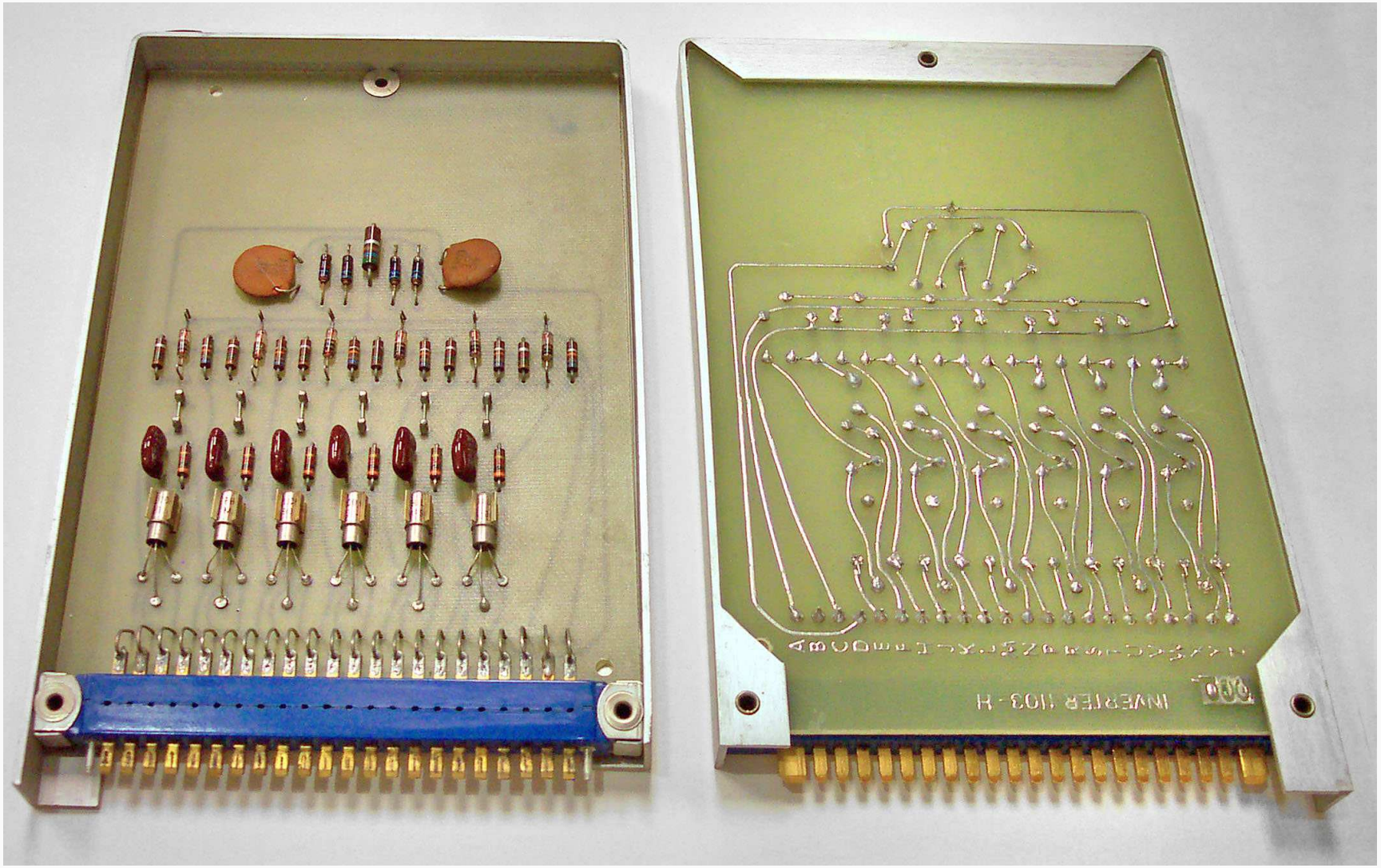
Computer Hardware: 1944 - vacuum tubes

- Colossus: arguably first programmable, electronic, digital computer.
- hardware designed by Tommy Flowers for WWII codebreaking.



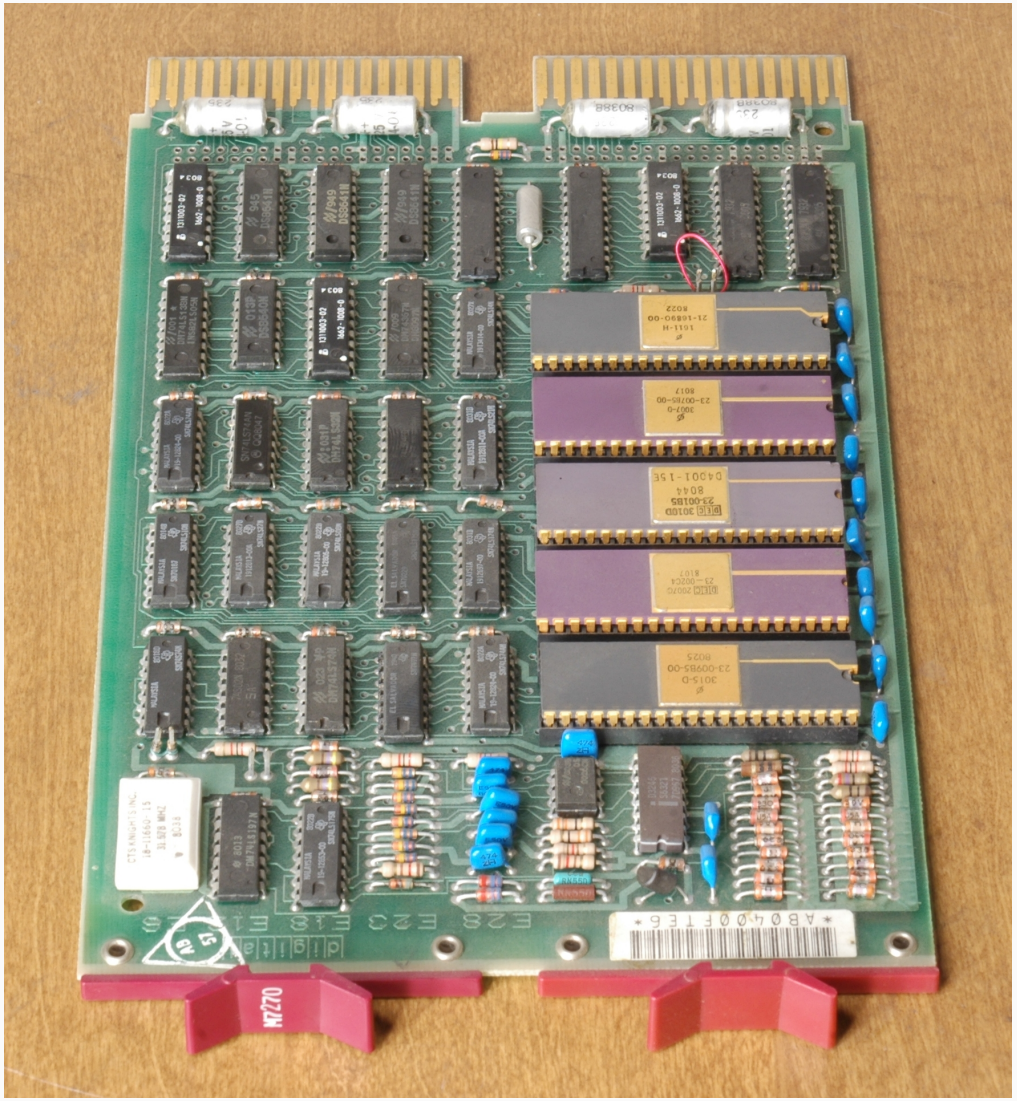
Computer Hardware: 1959 - transistors

- PDP-1 first computer in Digital Equipment Corporation's successful line.
- Successors were first machines C and Unix used on.



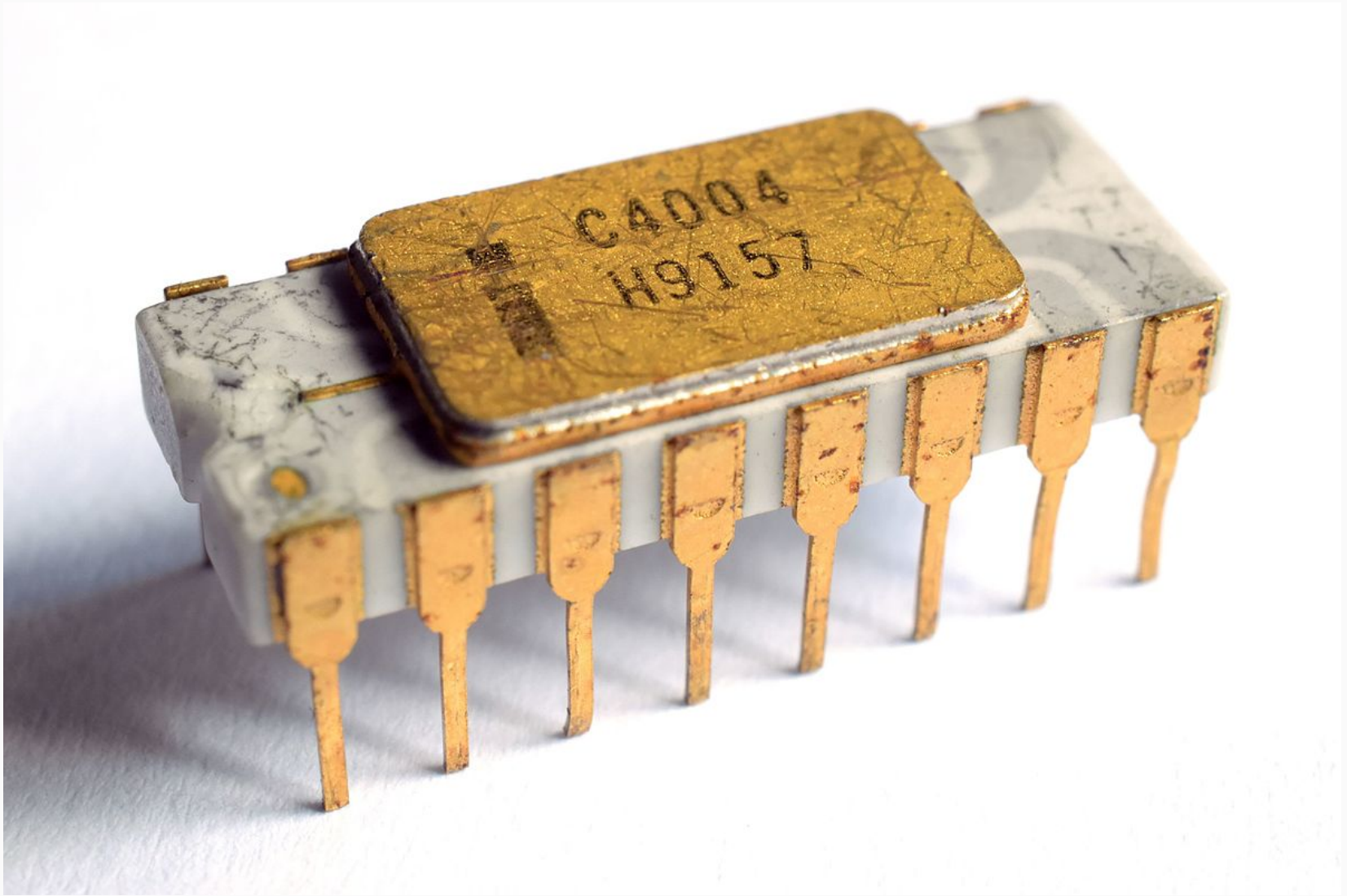
Computer Hardware: 1975 - Integrated Circuits

PDP-11 computer using large-scale integrated circuits containing thousands of transistors.



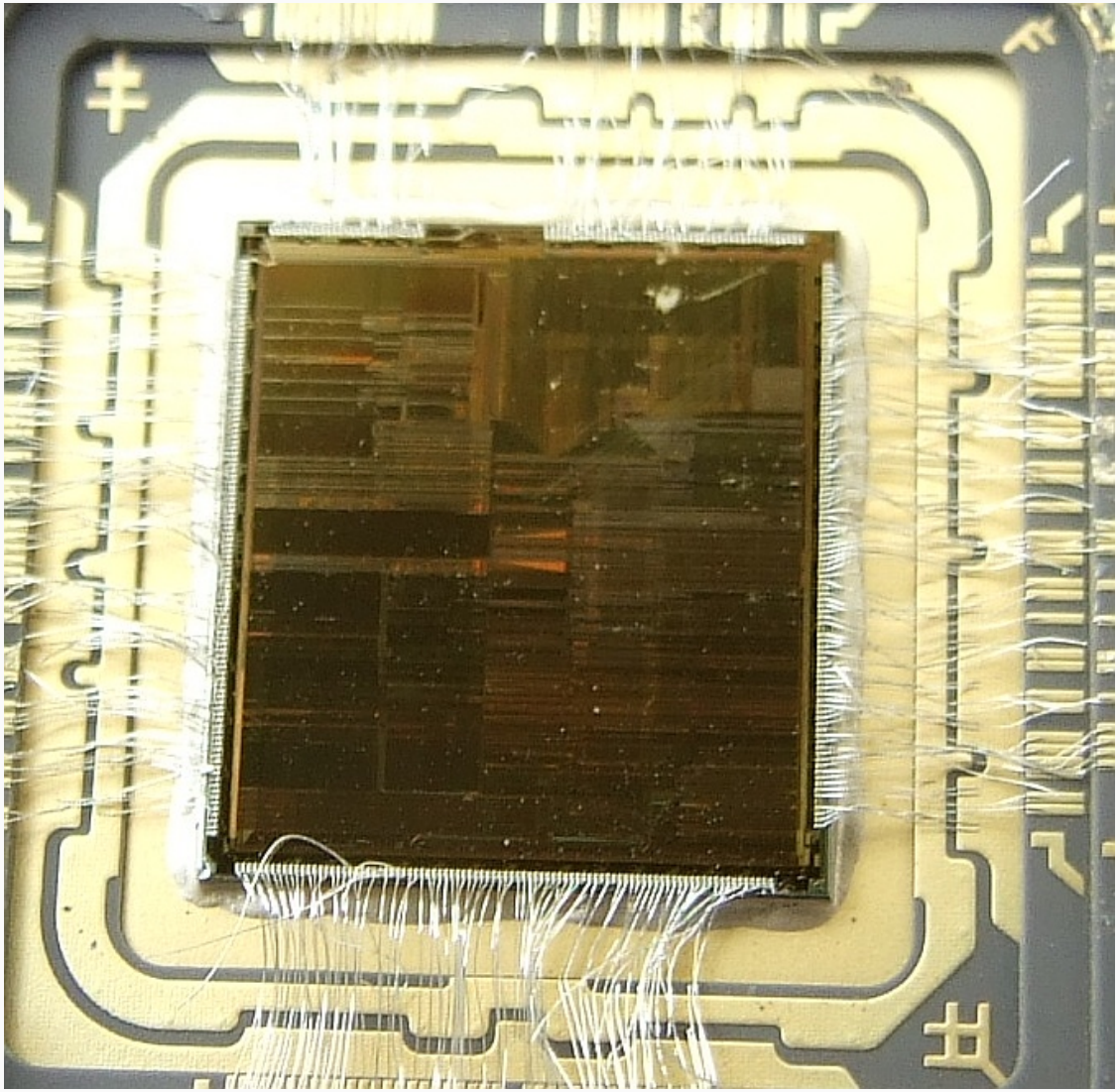
Computer Hardware: 1972 - Integrated Circuits

Intel 4004 4-bit microprocessor - computer on single chip - 2300 transistors.



Computer Hardware: 1993 - Integrated Circuits

Intel "Pentium" 32-bit microprocessor - computer on single chip - 1000000+ transistors.



Modern Computing

We now have much more processing capability than in the past

So what can we do with it?

- realtime solutions to very difficult problems
- global simulations for climate and weather patterns
- connecting individuals globally in communication networks
- simulating highly complex virtual environments
- deciding what you want to watch next on Netflix . . .
- and maybe you will come up with something new in the future . . .

Using CSE's Computing Resources

- our labs are running **Linux** with the basic tools necessary to get started
- you will definitely want to get your own computer ready to code with:
- vLAB allows you to remotely use CSE's resources
- or - You can set up a programming environment on your own computer
  - check the course website for links to guides
- for COMP1511 we need:
  - a text editor start with **gedit**
  - a compiler - we use **dcc** - you might use other C compilers on your own computer

Working in Linux

- first thing we need to get setup with a simple programming environment
- at CSE we use the Linux operating system
- what is an Operating System?
  - software sitting between our code and the computer hardware
- what is Linux?
  - widely used free operating system
  - many other similar (Unix-like) operating systems
  - Unix first widely used multi-user and multi-tasking OS
  - long history; many innovations come from Unix
  - Linux, Android, OSX, FreeBSD - descendants of Unix

Using a Terminal

- our main interface to Linux is a terminal
- this means all our interaction is in text
- some commands (programs we can run)
- **ls**
  - Lists all the files in the current directory
- **mkdir** *directory_name*
  - Makes a new directory called *directory_name*
- **cd**
  - Changes the current directory
- **pwd**
  - Tells you where you are in the directory structure at the moment

Two important commands

- **gedit** - a easy-to-learn text editor
  - helps out a little by highlighting C in different colours
  - many other text editors available
  - coders often have strong personal preferences
- **dcc** - C compiler
  - A translator that takes our formal human readable C and turns it into a actual machine readable program
  - its output is a program we can run ("machine code")
  - dcc includes extra checking to help novice coders
  - there are other C compilers (gcc, clang)
  - dcc tricky to get working on your homem achine
- you can use VLAB to access `gedit` and `dcc` at CSE from anywhere
  - other compilers and editors also available at CSE
  - many you can install for free on your own computer

Programming in C

Programming is like talking to your computer

- we need a shared language to be able to have this conversation
- we'll be looking at one particular language, **C** and learning how to write it

C is:

- a clear language with defined rules so that nothing we write in it is ambiguous
- many modern programming languages are based on C
- a good starting point for learning how to control a computer from its roots

Let's see some C

```c
// Example Program showing output in C
// Andrew Taylor, June 2020
#include <stdio.h>
int main(void) {
    printf("Hello Andrew!\n");
    return 0;
}
```

source code for hello.c

Comments

```
// Example Program showing output in C
// Andrew Taylor, June 2020
```

Words for humans

- half our code is for the machine, the other half is for humans! (roughly)
- we put **comments** in to describe to other people or our future selves what we intended for this code
- `//` in front of a line makes it a comment
- if we use `/\*` and `\*/` everything between them will be comments
- the compiler ignore comments, so they don't have to be proper code

#include

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

- **`#include`** is a special tag for our compiler
- it asks the compiler to grab another file of code and add it to ours
- in this case, it's the Standard Input Output Library
  - allowing us to make text appear on the screen
  - as well as other things
- almost every C program you write in COMP1511 will have this line
- `stdio.h` is the only include file we'll need in the next few weeks

The “main” Function

```
int main(void) {
    printf("Hello Andrew!\n");
    return 0;
}
```

- a **function** is a block of code that is a set of instructions
- our computer will run this code line by line, executing our instructions
- the first line has details that we’ll cover in later lectures
- `int` is the output - this stands for integer, which is a whole number
- `main` is the name of the function
- `(void)` means that this function doesn’t take any input

The Body of the Function

```
int main(void) {
    printf("Hello Andrew!\n");
    return 0;
}
```

- Between the `{` and `}` are a set of program instructions
- `printf()` makes text appear on the screen
  - it is actually a function from `stdio.h` which we included.
- `return` is a C keyword
  - it says we are now delivering the output of the function.
- `main` returns 0 to signify a correct outcome of the program
  - important when programs run other programs
  - we won't do that in COMP1511 - you will in COMP1521 & COMP2041

Editing and Compilation

- we can open a terminal now and try the code we've just looked at
- in the linux terminal we will open the file to edit by typing:

```
$ gedit hello.c &
```

- once we're happy with the code we've written, we'll compile it by typing:

```
$ dcc hello.c -o hello
```

- the `-o` part tells our compiler to write out a file called **`hello`** that we can then run by typing:

```
$ ./hello
```

- the `./` lets us run the program **`hello`** that is in our current directory

One working Program!

That's one program working!

What to do next?

- try this yourself!
- try it using VLAB via your own computer
- try setting up a programming environment on your own computer
    - differing levels of difficulty depending on your operating

What did we learn today?

- COMP1511 and administration
- where to find resources (course webpage and forum!)
- what your responsibilities are and how to succeed in programming
- a brief look at the history of computers
- an overview of how they work
- your very first C program
- using the basics of Linux