Origins of Programming

Relevant for: COMP5711M

Brandon Bennett

School of Computing
University of Leeds

B.Bennett@leeds.ac.uk

Last Updated: August 2019

Learning Goals

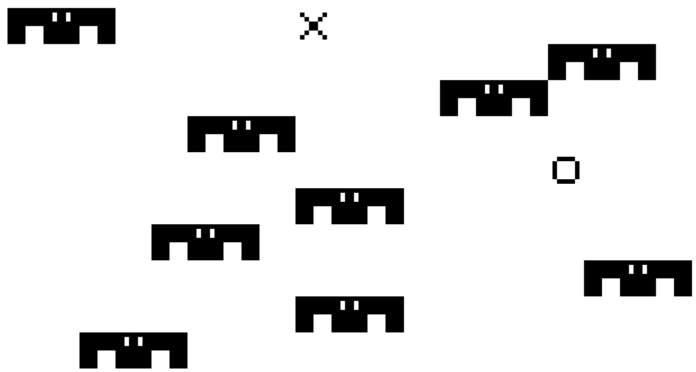
- To appreciate that modern computers developed from a variety of much older devices.
- To recognise that the essential characteristic of a computer is that it can be *programmed* to perform different tasks.
- To understand some basic aspects of the nature of programming languages.

Overview

In today's lecture we shall cover the following:

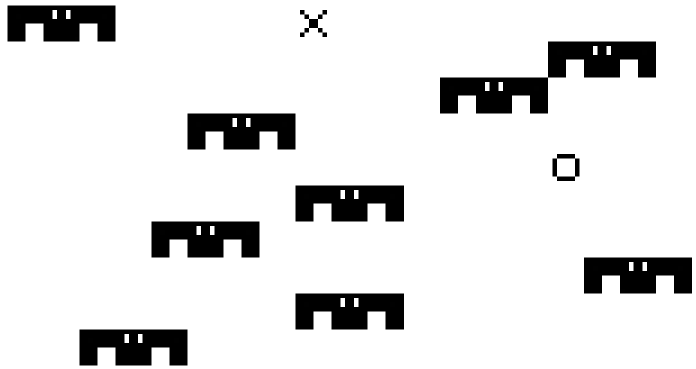
- The origins and early development of computing devices.
- Basic theoretical ideas of computation and programming.
- Essentials of machine code and assembly language.
- The inception of more sophisticated languages.
- The idea that different languages are equivalent in their essential power but differ in style and usability.

How did we get from here ...



slide not in handout

How did we get from here ...



to here?

slide not in handout

Man-Eating Budgies to GuildWars

This is the entire code for *Man-Eating Budgies* (released 1981).

```
   10 LET A=15
   20 LET E=0
   30 LET P=1+PEEK 16396+PEEK 163
97*256
   40 LET A$="▛▀▀▜"
   50 IF RND>.9 THEN LET A$="O"
   60 PRINT AT 10,INT (RND*29);A$
   70 SCROLL
   80 PRINT AT 0,31;" "
   90 IF PEEK (P+A)=52 THEN GOTO
140
  100 IF PEEK (P+A)<>0 THEN GOTO
160
  110 PRINT AT 0,A;"X"
  120 LET A=A+(INKEY$="8")-(INKEY
$="5")
  130 GOTO 40
  140 LET E=E+1
  150 GOTO 110
  160 PRINT "YOU HAVE BEEN EATEN"
,"YOU SCORED ";E;" POINTS"
  170 INPUT A$
  180 CLS
  190 RUN
```

Man-Eating Budgies to GuildWars

This is the entire code for *Man-Eating Budgies* (released 1981).

```
  10 LET A=15
  20 LET E=0
  30 LET P=1+PEEK 16396+PEEK 163
97*256
  40 LET A$="▛█▜"
  50 IF RND>.9 THEN LET A$="O"
  60 PRINT AT 10,INT (RND*29);A$
  70 SCROLL
  80 PRINT AT 0,31;" "
  90 IF PEEK (P+A)=52 THEN GOTO
140
 100 IF PEEK (P+A)<>0 THEN GOTO
160
 110 PRINT AT 0,A;"X"
 120 LET A=A+(INKEY$="8")-(INKEY
$="5")
 130 GOTO 40
 140 LET E=E+1
 150 GOTO 110
 160 PRINT "YOU HAVE BEEN EATEN"
,"YOU SCORED ";E;" POINTS"
 170 INPUT A$
 180 CLS
 190 RUN
```

GuildWars takes around 500MB of disk space. Although a lot of this is taken up by graphics, it includes millions of lines of actual program code (C++ I think?).

How this Device Changed my Life

The Sinclair ZX-80 (give wistful monologue)

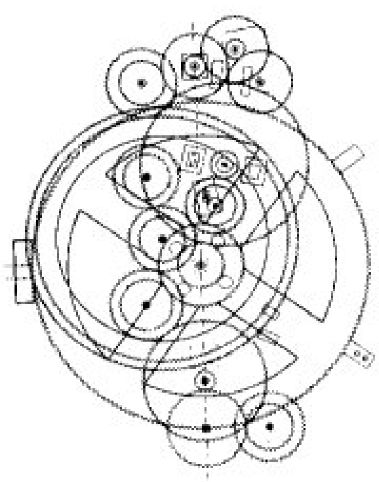
slide not in handout

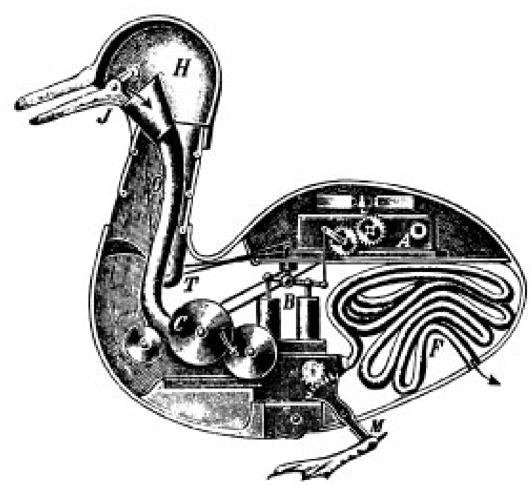
Early Computational Machines and Automata

There are many earlier precursors of modern computers.

These include mathematical computing devices, such as the Abacus and Slide Rule.

Complex devices were constructed, such as the ancient Greek Antikythera mechanism, an astronomical calculator, and robotic automata such as Vaucanson's Digesting Duck.





Programmable Machines

Many machines carry out clever complex tasks; however, most of them are not comparable to computers because they can only perform a fixed series of operations.

Computers by contrast can be configured to perform a wide range of functions — they are *programmable*.

Programmable Machines

Many machines carry out clever complex tasks; however, most of them are not comparable to computers because they can only perform a fixed series of operations.

Computers by contrast can be configured to perform a wide range of functions — they are *programmable*.

(But note that certain ‘embedded systems’ are generally considered to be a kind of computer although they only run a single fixed program.)

Hero's Ancient Ropebot

One of the most remarkable ancient machines is an automaton built by Hero (aka Heron) of Alexandra around 70 AD.

This is a moving, wheeled robot designed for use in theatrical entertainment. It is powered by a heavy weight that slowly descends pulling a string that is attached to its axels.

The movement of the robot is truly programmable by an ingenious system whereby the string is wound round the axels.

See: `http://www.youtube.com/watch?v=xyQIo9iS_z0`

The Jacquard Programmable Loom

Based on simpler machines produced by earlier French weavers, Joseph Marie Jacquard in 1801 constructed a loom which could be *programmed* to produce different patterns of woven cloth.

The loom was programmed using boards into which holes were punched to specify the pattern and configure the machine to produce it.

Early Computers

Charles Babbage was the first (1837) to design a fully programmable mechanical computer, which he called *The Analytical Engine*.

Due to limited finance, and continual revisions to the design Babbage never actually built a working computer.

Ada Lovelace is credited with writing the first computer program, which was a specification of how to compute Bernoulli numbers using the Analytical Engine.

(See Wikipedia for lots of interesting information about Babbage and Lovelace.)

Theories of Computation

The theory of computation and computability was a subject of intense interest during the 30's 40's and 50's. Several mathematicians (e.g. Turing, Church, Kleene, Post) proposed different mathematical models of the process of automated computation. The best know of these is called the *Turing Machine*.

Theories of Computation

The theory of computation and computability was a subject of intense interest during the 30's 40's and 50's. Several mathematicians (e.g. Turing, Church, Kleene, Post) proposed different mathematical models of the process of automated computation. The best know of these is called the *Turing Machine*.

Subsequently all these models were shown to be essentially equivalent.

Theories of Computation

The theory of computation and computability was a subject of intense interest during the 30's 40's and 50's. Several mathematicians (e.g. Turing, Church, Kleene, Post) proposed different mathematical models of the process of automated computation. The best know of these is called the *Turing Machine*.

Subsequently all these models were shown to be essentially equivalent.

The *Church-Turing Thesis* proposed in 1936, is the conjecture that all computational mechanisms are in fact either more limited than or equivalent in their computing capabilities to a Turing machine.

Theories of Computation

The theory of computation and computability was a subject of intense interest during the 30's 40's and 50's. Several mathematicians (e.g. Turing, Church, Kleene, Post) proposed different mathematical models of the process of automated computation. The best know of these is called the *Turing Machine*.

Subsequently all these models were shown to be essentially equivalent.

The *Church-Turing Thesis* proposed in 1936, is the conjecture that all computational mechanisms are in fact either more limited than or equivalent in their computing capabilities to a Turing machine.

(CS Students will study this in more detail later in their corse).

Machine Code

(This should be covered in Computer Systems (COMP 1440), so here is only a brief explanation of machine code.)

Machine Code

(This should be covered in Computer Systems (COMP 1440), so here is only a brief explanation of machine code.)

Machine Code is a form of digital (binary) information that is processed by and determines the actions performed by a computer's CPU.

Machine Code

(This should be covered in Computer Systems (COMP 1440), so here is only a brief explanation of machine code.)

Machine Code is a form of digital (binary) information that is processed by and determines the actions performed by a computer's CPU.

This code is normally stored as a sequence of bytes in memory, and the fundamental functionality of a CPU is to *execute* this code as a sequence of instructions.

Machine Code

(This should be covered in Computer Systems (COMP 1440), so here is only a brief explanation of machine code.)

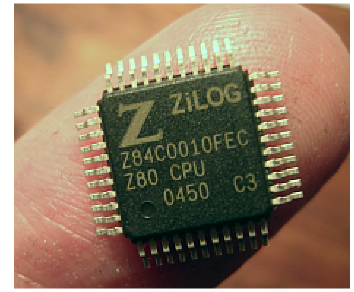
Machine Code is a form of digital (binary) information that is processed by and determines the actions performed by a computer's CPU.

This code is normally stored as a sequence of bytes in memory, and the fundamental functionality of a CPU is to *execute* this code as a sequence of instructions.

The actions resulting from these instructions include: logical and arithmetic manipulation of digital data; transfer of data from one memory location to another (or to/from some peripheral device); making the sequence of execution jump to a new memory location.

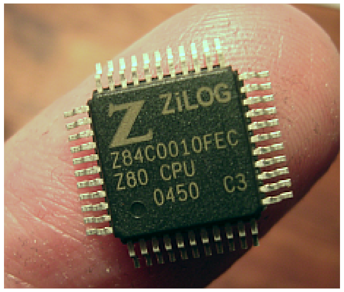
Z80 Machine Code

To get a flavour of what machine code is and how it works, we consider the code used on the old Z80 chip, which was used in the Sinclair ZX80 (one of the first home computers).



Z80 Machine Code

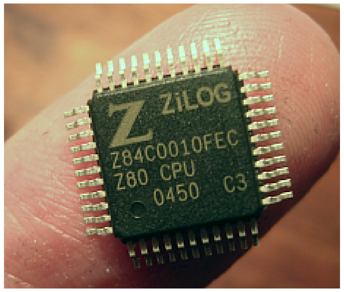
To get a flavour of what machine code is and how it works, we consider the code used on the old Z80 chip, which was used in the Sinclair ZX80 (one of the first home computers).



The Z80 CPU has 8 main *registers* designated A, B, C, D, E, F, H, L (and some other special registers that we won't consider). Each of these can store a *Byte* — i.e. 8 binary digits.

Z80 Machine Code

To get a flavour of what machine code is and how it works, we consider the code used on the old Z80 chip, which was used in the Sinclair ZX80 (one of the first home computers).



The Z80 CPU has 8 main *registers* designated A, B, C, D, E, F, H, L (and some other special registers that we won't consider). Each of these can store a *Byte* — i.e. 8 binary digits.

The CPU performs the following kinds of operation:

- load a byte into a register
- copy bytes from one register to another,
- copy bytes from a register to a memory location or vice

versa,

- perform arithmetic operations on register contents.

Example Z80 Instructions

Each machine instruction executed by the chip is also encoded in one byte. Hence there are 256 different operation codes.
Here are some of the more straightforward operations:

Hex Code	Mnemonic	Operation
`3E`	`LD A, 6`	load A with number 6
`78`	`LD A, B`	copy contents of B into A
`67`	`LD H, A`	copy contents of A into H
`77`	`LD (HL) A`	copy contents of A to mem loc HL
`7E`	`LD A (HL)`	load A with the contents of mem loc HL
`3C`	`INC A`	add 1 to contents of A
`80`	`ADD A, B`	add A to B and put result in A

The registers H and L store the high and low bytes of a two-byte memory address. Hence the chip can address 64K of memory.

Machine Code Execution

In running a machine code programme the CPU processes a sequence of bytes stored in memory, interpreting each byte as an instruction according the operation coding.

Machine Code Execution

In running a machine code programme the CPU processes a sequence of bytes stored in memory, interpreting each byte as an instruction according the operation coding.

(Actually, some bytes in machine code will be interpreted as numerical data rather than instructions. E.g. in executing the code sequence 3E, 06, the 06 is the number to be loaded into the A register — see last slide.)

Machine Code Execution

In running a machine code programme the CPU processes a sequence of bytes stored in memory, interpreting each byte as an instruction according the operation coding.

(Actually, some bytes in machine code will be interpreted as numerical data rather than instructions. E.g. in executing the code sequence 3E, 06, the 06 is the number to be loaded into the A register — see last slide.)

The effect of running the program arises from its reading and modification of memory locations. Some locations may hold values resulting from keyboard input or may be used to determine the monitor screen display.

Languages on top of Languages

We have seen how machine code is a language which directly controls the operation of a computer's CPU. Such a language is called *low-level*.

The evolution of programming languages is primarily driven by a desire to rise above the nitty-gritty details of machine operations to a higher-level of specification.

Languages on top of Languages

We have seen how machine code is a language which directly controls the operation of a computer's CPU. Such a language is called *low-level*.

The evolution of programming languages is primarily driven by a desire to rise above the nitty-gritty details of machine operations to a higher-level of specification.

High-level languages are intended to provide natural modes of expressing computational functionality, which are geared towards conceptualisation of problems and articulation of how they should be solved.

We shall later see how the processes of *interpretation* and *compilation* allow high-level languages to be implemented by translation into low-level machine code.

Machines within Machines

How can we describe the nature of programming?

Machines within Machines

How can we describe the nature of programming?

A program is like a machine operating within another machine.

Machines within Machines

How can we describe the nature of programming?

A program is like a machine operating within another machine.

Programming is like building a machine.

Machines within Machines

How can we describe the nature of programming?

A program is like a machine operating within another machine.

Programming is like building a machine.

The machine (i.e. program) is constructed from a basic components which can be combined in specific ways (according to the syntax of a programming language).

Machines within Machines

How can we describe the nature of programming?

A program is like a machine operating within another machine.

Programming is like building a machine.

The machine (i.e. program) is constructed from a basic components which can be combined in specific ways (according to the syntax of a programming language).

The program itself is just a specification of how the machine works. It's implementation is achieved by a compiler or interpreter translating the program into a form that is executable by the CPU.

All Languages are the Same (?)

According to the *Church-Turing Thesis*, all programming languages capable of certain fundamental processing capabilities are equivalent to Turing Machines, and so have essentially the same computing power (they can perform the same calculations).

All Languages are the Same (?)

According to the *Church-Turing Thesis*, all programming languages capable of certain fundamental processing capabilities are equivalent to Turing Machines, and so have essentially the same computing power (they can perform the same calculations).

However, languages differ greatly in the way computations are specified and executed.

All Languages are the Same (?)

According to the *Church-Turing Thesis*, all programming languages capable of certain fundamental processing capabilities are equivalent to Turing Machines, and so have essentially the same computing power (they can perform the same calculations).

However, languages differ greatly in the way computations are specified and executed.

Thus, the difference between programming languages is one of style, not of content.

Or, in other words, the difference is in *how* things can be done, not in *what* can be done.

Conclusions

- Computing has a long history.

Conclusions

- Computing has a long history.
- At the level of CPU function, programs are reduced to the execution of simple machine code instructions.

Conclusions

- Computing has a long history.
- At the level of CPU function, programs are reduced to the execution of simple machine code instructions.
- Programs are like *machines within machines*, and programming languages provide toolkits to build them.

Conclusions

- Computing has a long history.
- At the level of CPU function, programs are reduced to the execution of simple machine code instructions.
- Programs are like *machines within machines*, and programming languages provide toolkits to build them.
- Machine code provides a foundation upon which more sophisticated programming languages can be built (you will learn more in subsequent lectures).

Conclusions

- Computing has a long history.
- At the level of CPU function, programs are reduced to the execution of simple machine code instructions.
- Programs are like *machines within machines*, and programming languages provide toolkits to build them.
- Machine code provides a foundation upon which more sophisticated programming languages can be built (you will learn more in subsequent lectures).
- Different language are equivalent in their essential power, but vary greatly in style and usability (again, you will hear more).

Follow-Up Work

- Check out details of early computing machines and of Allan Turing's life and work on Wikipedia.

- Look at the material on the module web site at:

`https://teaching.bb-ai.net/PythonCoding/PracticalProgramming.html`