Sequence 1.1 – Introduction to Compilers

P. de Oliveira Castro S. Tardieu

How do we tell machines what to do ?

Figure 1: Jacquard Loom (1804)

How do we tell machines what to do ?

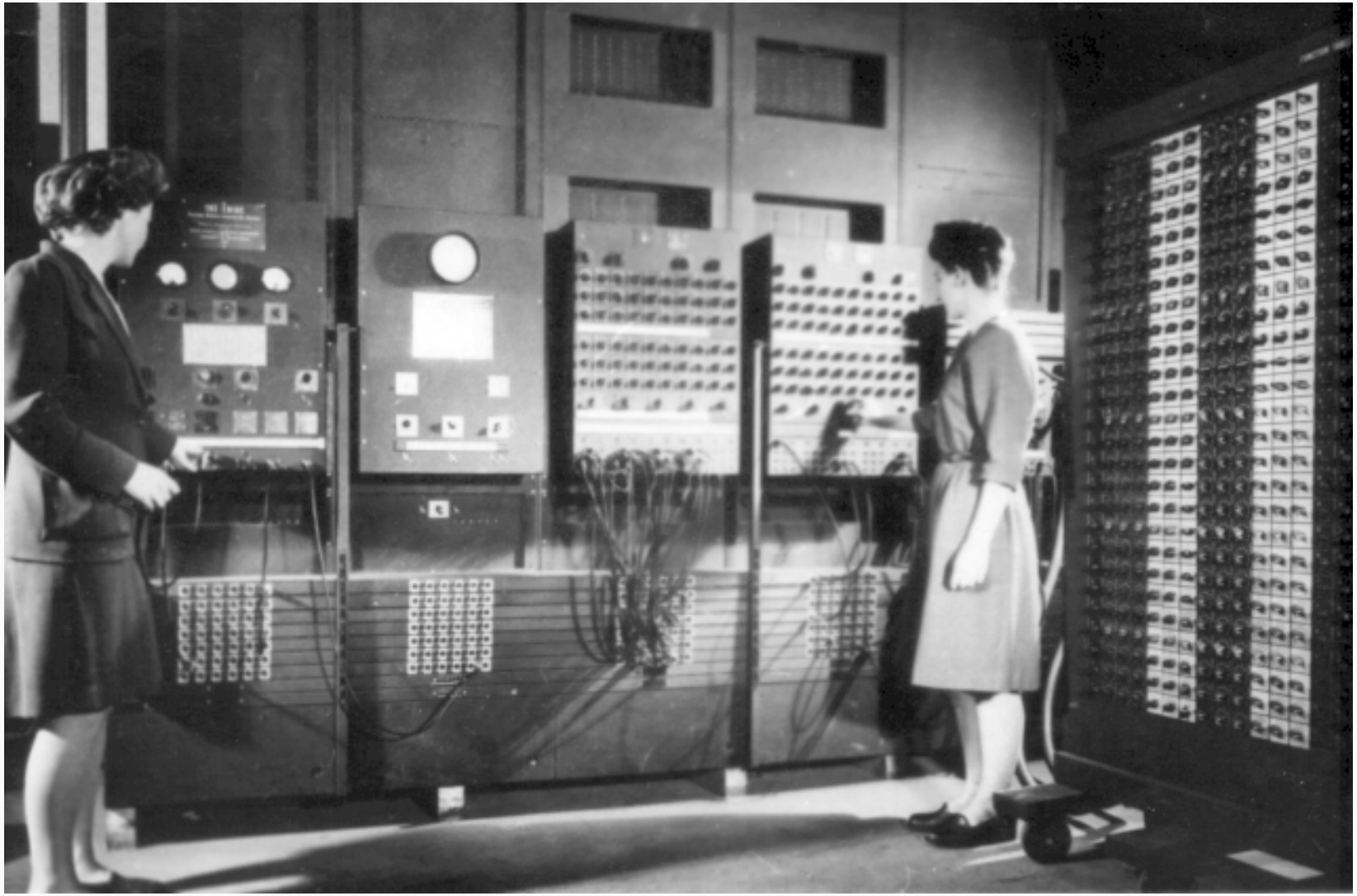

Figure 2: ENIAC (1943)

How do we tell machines what to do ?

Figure 3: Macintosh Plus (1986)

How do we tell machines what to do ?

- Switches or connection boards
- Perforated cards
- Keyboards
- Stored programs in memory (tapes, disks, electronic memory)

Each constructor has its own *instruction set*

What is an instruction set ?

- Each instruction available in the machine such as addition or memory moves is encoded as a binary number: *assembly* language
  - in the ARM 32 instruction set, 1110000110100000 0000 000000000001, moves the content of register `R1` into `R0`.
  - in the Intel x86 instruction set, 10001001 11 000 011, moves the content of register `eax` into `ebx`
- Remembering binary codes by heart is difficult; we prefer mnemonics:
  - `mov $r0, $r1`
  - `mov %eax, %ebx`

No single instruction set !

- Mnemonics are much easier to write and remember
- But each processor has unique features that require different mnemonics
  - x86 can combine arithmetic operations and memory operations
  - ARM 32 can predicate any instruction with a condition flag
- How can we write a portable program that works across all architectures ?
- Mnemonics are *low level* : easily understood by the machine
  - We want a *high level* language : easily understood by humans

Interpreters

- Define a common *high level* language, such as BASIC.
- For each architecture we write in assembly an *interpreter*
  - It reads a BASIC program instruction by instruction
  - For each instruction it calls an assembly procedure
  - Control-Flow instructions (`GOTO`, `IF`) move the reading cursor

Advantages

- Only write a single program (the interpreter) in assembly
- BASIC programs are portable across all architectures

Disadvantages

- Slow: instead of being directly executed each instruction must be read, decoded and simulated
- Hard to optimize: instructions are read on the fly, the interpreter cannot easily simplify or remove duplicate patterns

Interpreters and Virtual Machines

- Interpreters are still used today for many languages (Python, Ruby, PHP)
- Another option is to write a *Virtual Machine*, an efficient interpreter for a common assembly language (Java HotSpot VM, Dot Net).
- Modern virtual machines and Interpreters use some tricks to speed-up the execution and optimize the code. But they still add an additional software indirection layer.

Compilers

- An interpreter translates the common language on the fly
- A *Compiler* translates the whole program to assembly before the execution
- It reads a high level language (such as BASIC) and produces an *equivalent* assembly program.
- The notion of equivalence is tricky. A simple definition is that two programs are equivalent when they produce the same observable effects.

Compilers

Advantages

- Cost of translation is payed only once
- No need to ship an interpreter
- Many optimizations can be applied

Disadvantages

- Less flexible and dynamic than an interpreter for distributing code
- Some optimizations can only be applied at run time

Image Credits

- *Jacquard Loom*, photography by D. Monniaux, CC SA 3.0
- *Macintosh Plus*, photography by B. Patterson, CC BY 2.0