3.2.1 - CLASSIFICATION OF SOFTWARE

- Types of software
- The function of system software
- Types of application software
- How to select appropriate software for a purpose
- The development of programming languages
- Types of program translator and their use.

Learning Objectives

- Be able to **distinguish** between **system** and **application** software
- **Describe** and give examples of the FOUR **generations of programming languages**
- **Define** each of the THREE types of program translator

Software

- What is software?
  - Software is the general term used to describe all the programs that run on a computer.

Categories of software

- System software
  - OS
  - Library Programs
  - Utility programs
  - Programming language translators
- Application software
  - General purpose applications (generic software)
  - Special purpose application software
  - Bespoke software

System Software

- *Operating systems*
- *Utility programs* – designed to make life easier for computer users (e.g. compressing SW)
- *Library programs* – carry out common tasks required by everyone (e.g. search for a file)
- Programming language *translators* (compilers, interpreters and assemblers)
  - Why are they called "translators"?

System Software

- **Operating Systems**
  - Every computer needs an O/S to act as an interface between the user and the hardware
    - Name 3 O/S's
  - Provides a software platform on which other programs can run
  - An O/S is a set of programs that allow the user to perform tasks without having to know how they are done
    - E.g. Save a file on to a disk

System Software

- **Utility Programs**
  - Designed to make life easier for computer users
  - Perform common useful tasks
    - search for lost files
    - sort files of data into a particular sequence
    - copy disk files to magnetic tape for backup purposes
  - Can you think of others?

System Software

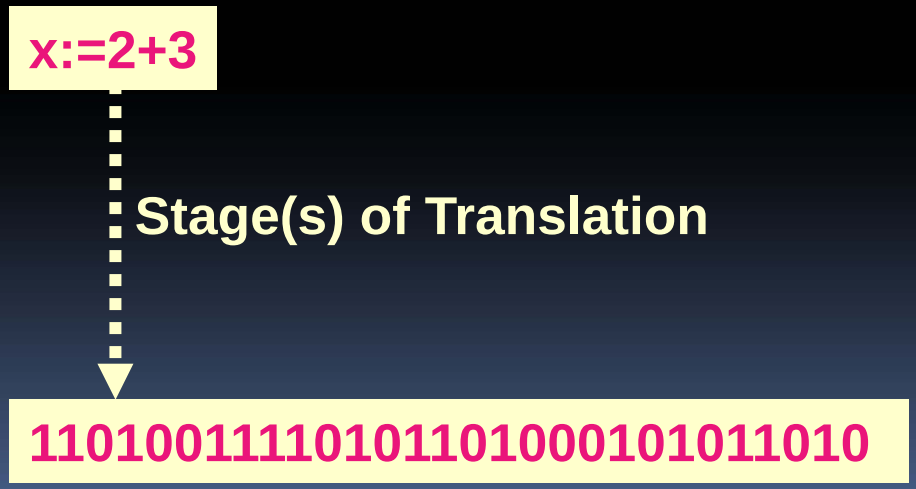
- **Library programs**
  - Like utility programs but…
  - Available to all users of a multi-user computer system to carry out common tasks required by everyone
    - example: any programming language (such as VB.Net) has a variety of library routines, eg runtime files required in order to run VB applications.

System software

- **Programming language translators**:
  - Assembler = assembly language → machine code
  - Compiler = high-level language code → object code
  - Interpreter = analyses and executes high-level language one line at a time
- translate the statements in a programming language (such as Pascal, Visual Basic or C) into a form that the computer can understand (i.e. assembly language, machine code)
  - Why is this useful for a user?

Program translators

- translate code from:
  - The format it is written in
  - TO a format that the computer can understand and execute



Assembler

- assembly code → machine code
  - What is assembly code?
  - Ready for the computer to execute it

- Each CPU has its own assembly language and an appropriate assembler
  - Usually defined by the hardware manufacturer

Compiler

- high level language source code → object code
  - this can then be executed without the need for a compiler to be present
  - Locked for editing to the end user
  - Portable

- takes the source code and scans through it
  - Each time performing different checks and building up tables of information needed to produce the final object code

Compiler

SOURCE Code

Written by programmer

OBJECT Code
(reusable + portable)

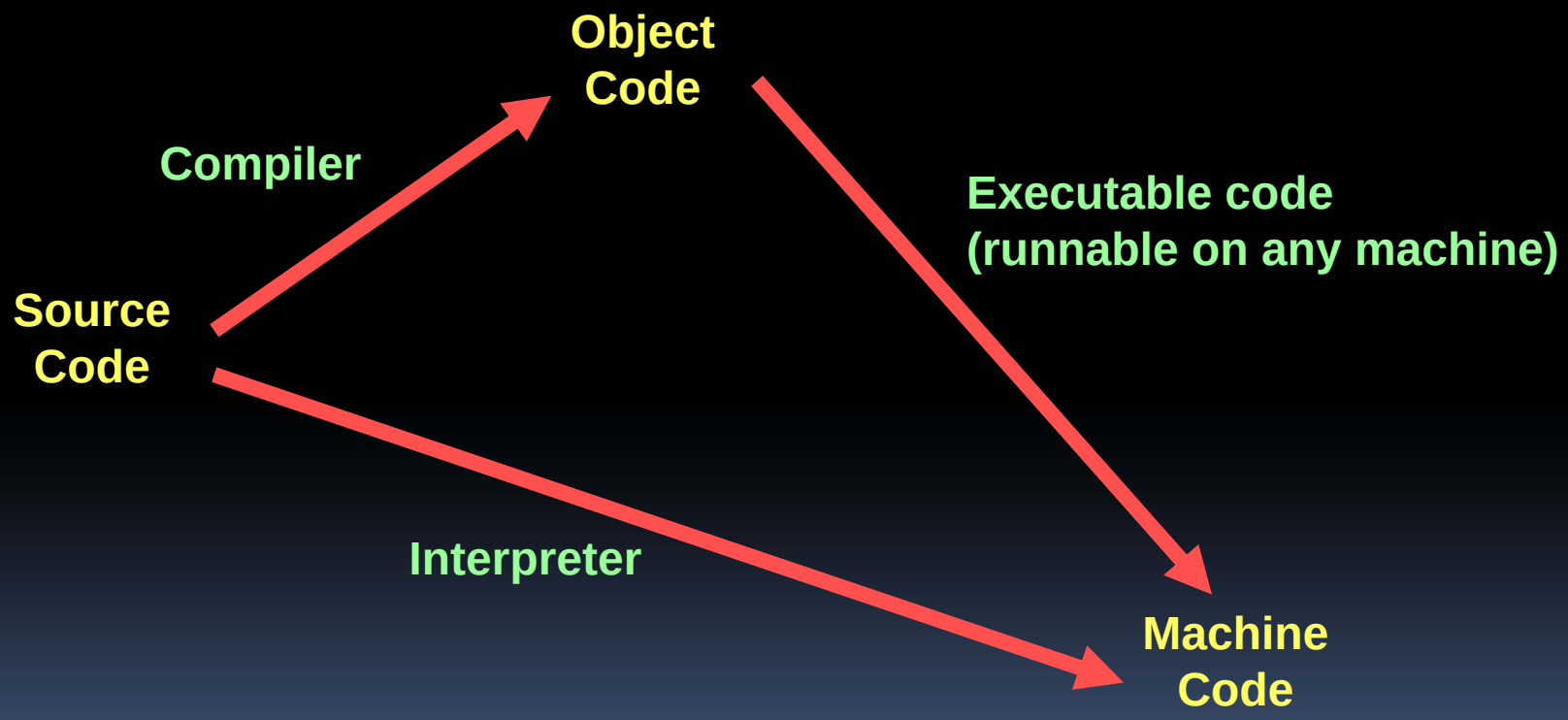
The compiled code

Why do companies (e.g. EA Games) distribute object code and not source code?

Interpreter

- Defn: Analyses the source code
  - statement by statement as execution proceeds
  - decoding each statement
  - and calling routines to carry out each instruction
- No object code is produced
  - Therefore program has to be interpreted each time it is to be run
- So when is it suitable?

Summary


Object
Code
Compiler
Executable code
(runnable on any machine)
Source
Code
Interpreter
Machine
Code

Compiler vs Interpreter

- Object code can be saved on disk and run whenever required without need to recompile
  - Limitation?
- Object code executes faster than interpreted code
- Object code can be distributed without the need for the compiler to be present
- Object code is more secure
  - Why?

Interpreter vs Compiler

- Easier to partially test and debug programs
  - Useful for program development as no need for lengthy recompilation each time an error is discovered

Compiler or Interpreter?

- Use: Program development
  - Interpreter
- Use: Program that needs distributing
  - Create portable object code using Compiler

Summary

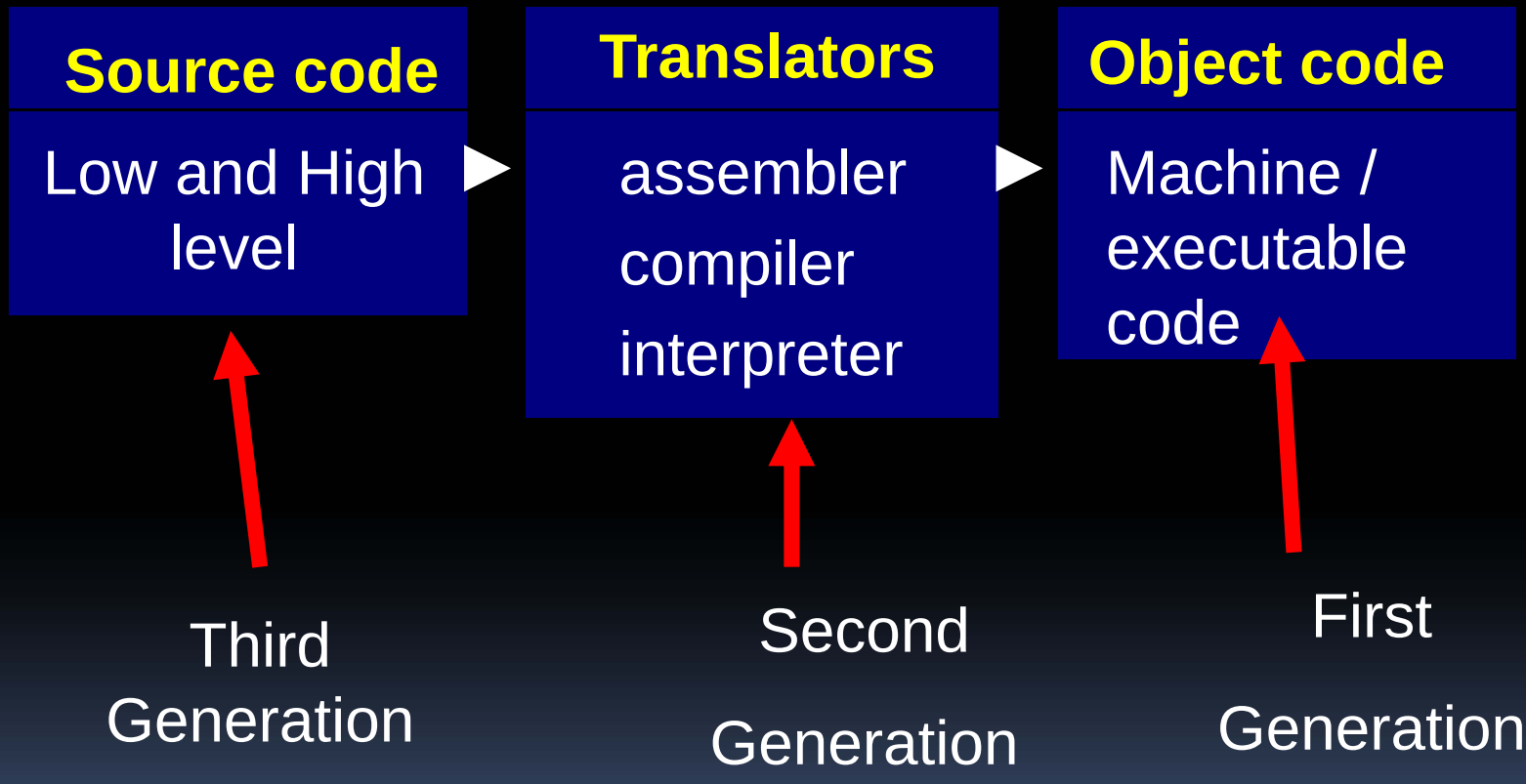
• if you wrote a program in a low-level language called Assembly then you would translate the source-code into object-code with an assembler

• if you wrote a program with certain high level languages such as Pascal or C, you would translate the source code into object code using a compiler

• if you wrote a program with certain high-level languages such as Basic or Perl, you would translate the source code into object code using an interpreter

• If you wrote a program with certain high level languages such as VB or Java you would test the source code using an interpreter .

The Translation process



Application Software

- Software designed to carry out specific tasks for the user
- They are primarily independent of computers, such as writing a letter or processing orders and invoices
- Can be designed especially for a company
  - Bespoke
- Bought off the shelf

Bespoke software

- Written for a customer's specific needs.
  - Customer pays development costs
  - Can be very expensive
  - Eg
    - Air traffic control
    - BBC TV licensing service
    - London congestion charging

General Purpose Software

- Software that can be made to do many tasks
- Includes all common application packages such as:
  - word processing
  - desktop publishing
  - spreadsheet
  - database
  - computer-aided design (CAD)
  - presentation graphics
- Most general purpose software is sold as a package, including a CD containing the software and manuals to help you get started and to be used as a reference.

Special Purpose Software

- Examples
  - a payroll
  - stock control system
  - software package to help fill in an income tax return.

- Designed to do one particular task

Bespoke Or Off-the-shelf?

- Advantages of buying an off-the-shelf package
  - generally less expensive
  - it may be possible to speak to other users of the package for their evaluation before buying
  - can be bought and installed straight away
  - software is tried and tested and likely to contain fewer bugs than newly written software
  - usually well documented
  - training may be available in common packages

- Advantages in buying tailor-made software (bespoke)
  - designed to do exactly what the user wants
  - can be written to run on specified hardware
  - can be integrated with existing software
  - there may not be a suitable package available

Generations of programming languages

- *1st generation*
  - Machine code – the only program code that a computer "understands" and can execute; it is entirely binary code.
  - Each family of processors has its own machine code – the instruction set
- *2nd generation*
  - Assembly language – same structure & instruction set as machine code, but using mnemonics instead of numbers.
- *3rd generation*
  - Imperative languages – FORTRAN, COBOL, C
- *4th generation*
  - Declarative languages – PROLOG, SQL

Machine code (1G)

- Programmers code in 1's and 0's (BINARY numbers)
  - As used in computer's memory
- Executes directly without translation
  - Makes it very fast to execute programs
- Disadvantages
  - Laborious
  - Time-consuming
  - Error-prone

Assembly code (2G)

- Low level language
  - Very close to machine code and the detail of the computer architecture

- Programmers use **mnemonics** and denary numbers instead of binary
  - Mnemonics are abbreviations that represent the instructions in a more memorable way

Assembly code (2G)

- low – level languages are much closer to the working of the computer

- often control programs that **require very fast execution speeds** are written in assembly

- this is because when they are converted into machine-code (using an assembler), they produce less machine code than if the equivalent program was written in a high-level language

Assembly code (2G)

- Programs written in assembly code have to be translated into machine code using an assembler

- Assembly code statements mostly have a 1:1 correspondence between them and their equivalent machine code statement
  - This allows programs to be written in the most efficient way so that they:
    - Take up less space
    - Execute faster

Assembly code (2G)

- Example usage
  - Used to program device drivers that control a printer, disk drive etc.
    - Why not just use machine code?
  - Embedded systems like satellite decoders
  - Any application where the same routine is called frequently and the routine doesn't require change

- Typical instructions

```
MOVE   R1, R2      ; Move contents of register R2 to R1
LDA    #32         ; Load number 32 into accumulator
STA    X           ; Store contents of accumulator into
    memory location X
```

Imperative high level languages (3G)

- Developed to help programmers write code easily
- Instructions are executed in a programmer-defined sequence

- Examples
  - Algol (ALGOrithmic Language)
  - Fortran (FORmula TRANslation)
  - COBOL (COmmon Business Oriented Language)
  - BASIC
  - Pascal
    - Used to teach structured programming to students

Declarative high level languages (4G)

- Define *what* is to be computed, rather than *how* the computation is to be done.
- Eg:
  - Prolog
  - SQL

SUMMARY of software.

Task

DO the worksheet sent to you by email.

To be finished as HOMEWORK.