

## Topic 4

### Programming Languages

Abstracting further from the Nuts and Bolts

### Assembly Languages

- CPU (the core of the computer) responds to binary instructions only.
- The very first computer programs were written directly as binary instructions – very complex and error prone
- To make things easier and to reduce errors
  - Symbolic names assigned to each of the binary codes that make up the instruction set of the CPU – *Assembly Languages*
  - Programs could then be written in terms of these operation names
  - Once the program is written, then it could be converted to the binary machine instructions.

### Assembly Languages

- Example assembly language program

Hex Instruction	Hex machine instructions	Assembler machine instructions
0: 55		pushl %ebp
1: 89 e5		movl %esp,%ebp
3: 83 ec 0c		subl \$0xc,%esp
6: c7 45 fc 02 00		movl \$0x2,0xffffffff(%ebp)
b: 00 00		
d: c7 45 f8 03 00		movl \$0x3,0xffffffff(%ebp)
12: 00 00		
14: 8b 55 fc		movl 0xffffffff(%ebp),%edx
17: 03 55 f8		addl 0xffffffff8(%ebp),%edx
1a: 89 55 f4		movl %edx,0xffffffff4(%ebp)
1d: 8b 45 f4		movl 0xffffffff4(%ebp),%eax
20: eb 02		jmp 24 <main+0x24>
22: 89 f6		movl %esi,%esi
24: c9		leave
25: c3		ret

### High-level Languages

- Today, we write programs in *high-level languages*.
  - Provide a degree of abstraction in expressing the solution to a programming problem without (or with minimal) regard to the physical hardware that will execute the program.
- However, all programs must ultimately end up as binary instructions for a CPU to execute.
- The program that translates the code written in the high-level language into binary machine instructions is called either a *compiler* or *interpreter*

## Compiler

- A **compiler** translates high-level language programs (*source* code) into a form the processor can run (*object* code, or “binaries”).
- Once in the executable form, the source code (and the compiler) are no longer required to run the program.
- The entire program is transformed, and in this process optimization can be performed to produce a highly efficient program.
- The compilation must be performed every time a change is made to the source code, and this can be a time consuming phase.

## Interpreter

- An **interpreter** also translates source code into machine code, but only one instruction at a time and then executes it.
- Because of this translation phase for every program statement, interpreted languages typically take longer during execution.
- Interpreted languages can only run with the interpreter program.
- Changes to source code can be immediately tested - no recompilation! Since code translation is on the fly, interpreters have flexibility lacking in compilers.

## High-level Languages

- A programming language must be:
  - Unambiguous
    - It must have a precise syntax (grammar) so that any syntactically correct program has unique meaning
  - Expressive
    - Allow easy modeling of a wide variety of tasks
  - Practical
    - Compilers must be able to convert the high level language into machine instructions efficiently
  - Simple to use
    - Minimise coding errors caused by complexity

## High-level Languages

- Here is a very simple program written in the C programming language
 

```

/*
A minimal C program that declares three integers, initialises two of
the integers to fixed values and assigns the sum to the third integer.
*/

int main() {
    int a;          /* Declare an integer variable called a */
    int b;
    int c;

    a = 2;
    b = 3;
    c = a + b;

    return(c);     /* The value stored in variable c is returned */
}
            
```

## High-level Languages

- The result of compiling the previous C program

Hex Instruction	Hex machine instructions	Assembler machine instructions
	0: 55	pushl %ebp
	1: 89 e5	movl %esp,%ebp
	3: 83 ec 0c	subl \$0xc,%esp
	6: c7 45 fc 02 00	movl \$0x2,0xffffffff(%ebp)
	b: 00 00	
	d: c7 45 f8 03 00	movl \$0x3,0xffffffff(%ebp)
	12: 00 00	
	14: 8b 55 fc	movl 0xffffffff(%ebp),%edx
	17: 03 55 f8	addl 0xffffffff(%ebp),%edx
	1a: 89 55 f4	movl %edx,0xffffffff4(%ebp)
	1d: 8b 45 f4	movl 0xffffffff4(%ebp),%eax
	20: eb 02	jmp 24 <main+0x24>
	22: 89 f6	movl %esi,%esi
	24: c9	leave
	25: c3	ret

## High-level Languages

A rough decoding...

Instructions 3:5      subl      \$0xc,%esp

Subtract 12 (0xc) from the value stored in register %esp. The %esp register maintains the current stack pointer. This makes space for the 3 integers a, b, and c - each integer requiring 4 bytes of space. Note that the registers are simply special memory locations located within the CPU.

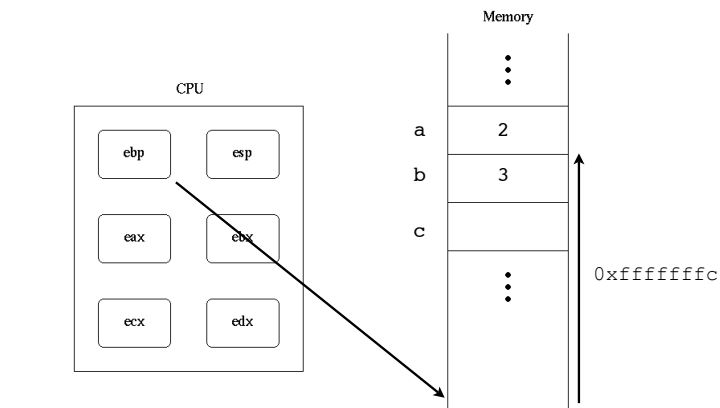
Instructions 6:a      movl      \$0x2,0xffffffff(%ebp)

Move the value 2 (0x2) to memory location 0xffffffff relative to the value in register %ebp. The %ebp register maintains the base pointer - the address in memory where the program's local data area begins. Note that at this stage we do not know where the operating system will load the program into memory when it is run, so all memory locations have to be specified in relative terms.

Instructions d:11      movl      \$0x3,0xffffffff8(%ebp)

Move the value 3 (0x3) to memory location 0xffffffff8 relative to the value in register %ebp.

## High-level Languages



## High-level Languages

Instructions 14:16      movl      0xffffffffc(%ebp),%edx

Move the value sitting at memory location 0xffffffffc relative to %ebp (this is the value 2 stored here earlier) to the %edx register.

Instructions 17:19      addl      0xffffffff8(%ebp),%edx

Add the value sitting at memory location 0xffffffff8 relative to %ebp (this is the value 3 stored here earlier) to the value in the %edx register (which holds the value 2). Store the result in the %edx register.

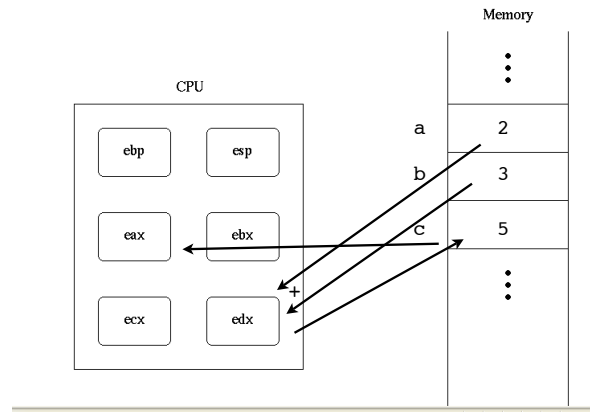
Instructions 1a:1c      movl      %edx,0xfffffffff4(%ebp)

Move the value in the %edx register to memory location 0xfffffffff4(%ebp) - the memory location of variable c.

Instructions 1d:1f      movl      0xfffffffff4(%ebp),%eax

Move the value of variable c to the %eax register for return to the operating system.

## High-level Languages



## Interpreted Languages

- Some languages are designed to be **interpreted**.
  - The code is read by another program (which has been compiled into machine instructions)
  - The code is interpreted and executed by the program running on the CPU. eg:
    - MATLAB
    - Java
    - Prolog
    - Lisp
    - Haskell

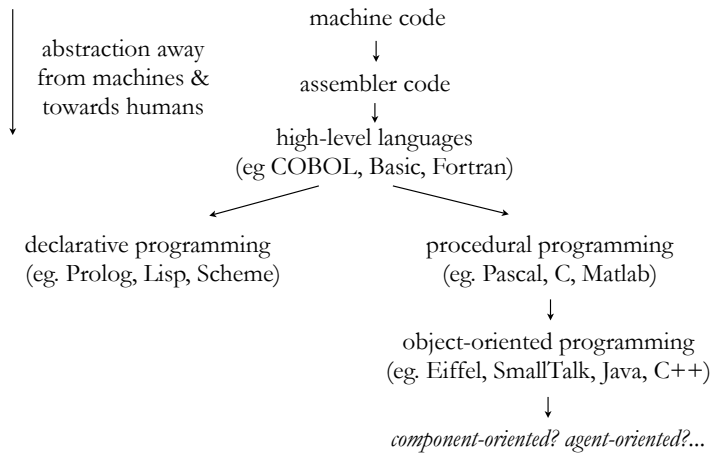
## Interpreted Languages

- In some cases, the code is compiled into an intermediate language that can be interpreted more efficiently.
  - MATLAB – before execution, MATLAB is compiled into **p-code** without being noticed.
  - Java – compiled into **'byte code'** instructions to be interpreted by the Java Virtual Machine.
- Interpreter – simulation of a virtual CPU running on the computer.
- This virtual CPU executes (interprets) the instructions sent to it.

## Interpreted Languages

- The advantage of interpreted languages are
  - They are very interactive – code can be tried out immediately (especially so with MATLAB)
  - They are portable across different machines and operating systems.
    - compiled code only works for the processor and operating system for which it was compiled.
- Providing an interpreter is available on the machine you want to run on, your program written in an interpreted language will run without needing any changes.
  - The reason for much of the buzz surrounding Java.
  - MATLAB also runs on a wide variety of platforms – Windows 95/98/NT/XP, MacOS, Linux + other Unix variants.

## Evolution of Programming



17

## Specialisation of Languages

- No one “best” programming language - different languages excel at different tasks. Eg
  - Scripting languages - running computer processes, string processing, web backends, ...
    - Eg. Shells (sh, csh, zsh, bash..), Pearl, Ruby, Python, php,...
  - Graphical languages
    - Tcl/Tk, OpenGL, VRML (virtual reality modelling language)
  - General application development languages - “capable” at many things (maths, GUIs, string processing, etc)
    - Basic, Pascal, C, Java,...

18

## Specialisation of Languages

- Scientific/engineering languages - mathematical modeling, solving systems of equations, fast floating point operations, maths libraries...
  - In the beginning, there was Fortran!
- Add interpreted language, fast prototyping, built-in graphical display of results, gui-based model building, more specialist libraries, ...
  - ➔ Matlab!

19

## Historical Note



<http://www.sdsu.edu/ScienceWomen/hopper.html>

- The first compiler was developed by Grace Hopper in 1949.
- Hopper was trained as a mathematician with a PhD from Yale.
- During WW II, she joined the Naval Reserve and worked with the Bureau of Ordnance Computation Project at Harvard University.
- After the war she joined the Eckert-Mauchly Computer Corporation and worked on UNIVAC I, the large scale electronic digital computer used for commercial applications.
- At the time, the only way to program computers was via direct machine instructions written in binary.

## Historical Note

- She believed that computer programs could be written in English
  - No one believed this was possible!
- She developed an elementary language called A-O in 1949, this was followed by a compiler for the UNIVAC called B-O.
- It was three years before her ideas were finally accepted – she published her first compiler paper in 1952.
- The B-O language was later renamed FLOW-MATIC which influenced the development of COBOL.
- She returned to the Navy in 1967, served with the Naval Data Automation Command, and retired in 1986 with the rank of Rear Admiral.
- She died in 1992.