



**Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών**

Languages, Levels and Virtual Machines

Γ. Γαροφαλάκης, Σ. Σιούτας

Languages, Levels and Virtual Machines

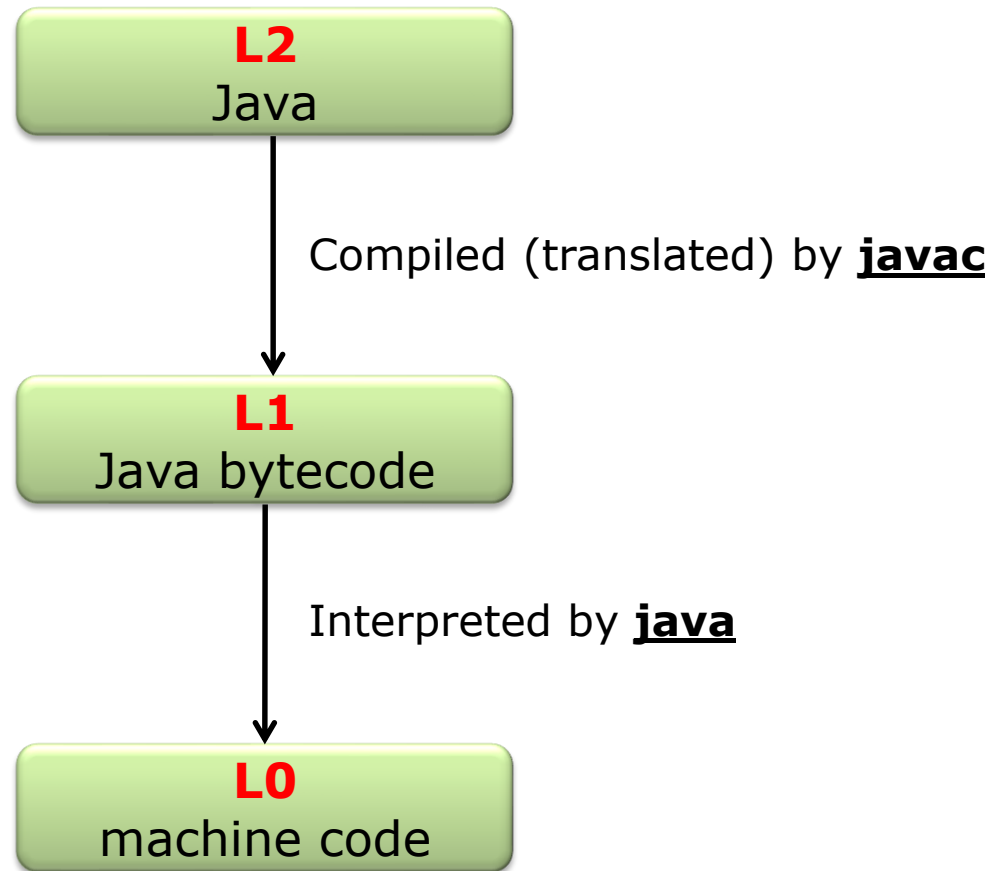
Language translation

- Τι συμβαίνει όταν τρέχεις αυτό;

```
javac Foo.java
```

- Το πρόγραμμα **Foo.java** (γραμμένο σε Java) μεταγλωττίζεται (**compiled** ή **translated**) στο πρόγραμμα **Foo.class** (που είναι πρόγραμμα σε Java Bytecode)
 - Πρόκειται για δύο διαφορετικές γλώσσες
 - **Java**: υψηλότερου επιπέδου (L1) → καλή για να λύνεις προβλήματα
 - **Java Bytecode** χαμηλότερου επιπέδου (L0) → πιο λιτή, απλή, με λιγότερα abstractions, πιο κοντά στην λογική των transistors.
 - Κάθε εντολή L1 αντικαθίσταται από μία ή περισσότερες εντολές L0
 - Το πρόγραμμα τελικά εκτελείται στη γλώσσα L0
- Μια γλώσσα μπορεί επίσης να διερμηνευτεί (**interpreted**)
 - Διαβάζεις μία-μία τις εντολές της L1
 - Για κάθε μία εντολή L1 εκτελείς τις αντίστοιχες εντολές σε L0
 - Αυτό κάνει άλλωστε και το JVM (Java Virtual Machine), για την εκτέλεση Java Bytecode

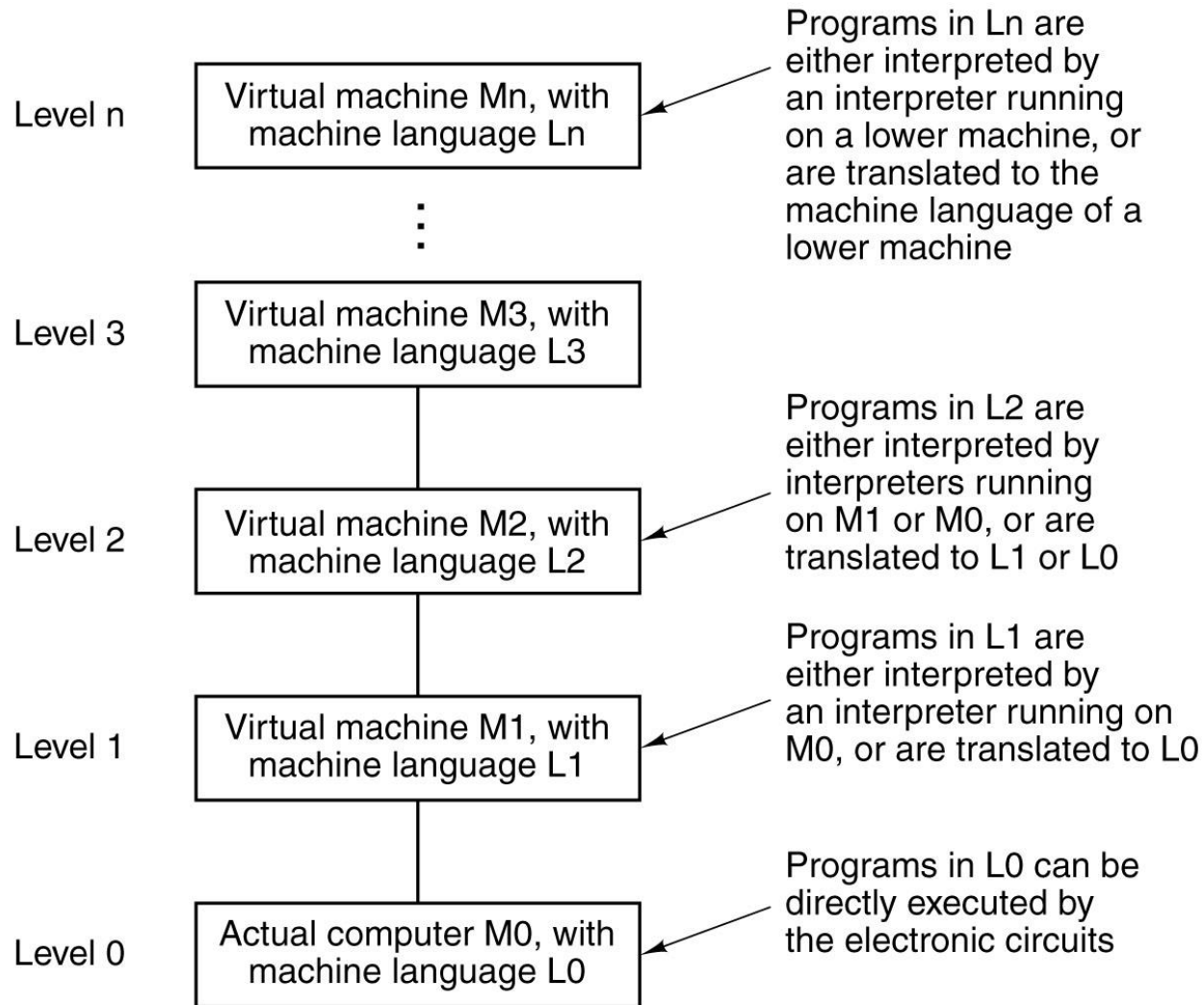
Compiling & Interpreting



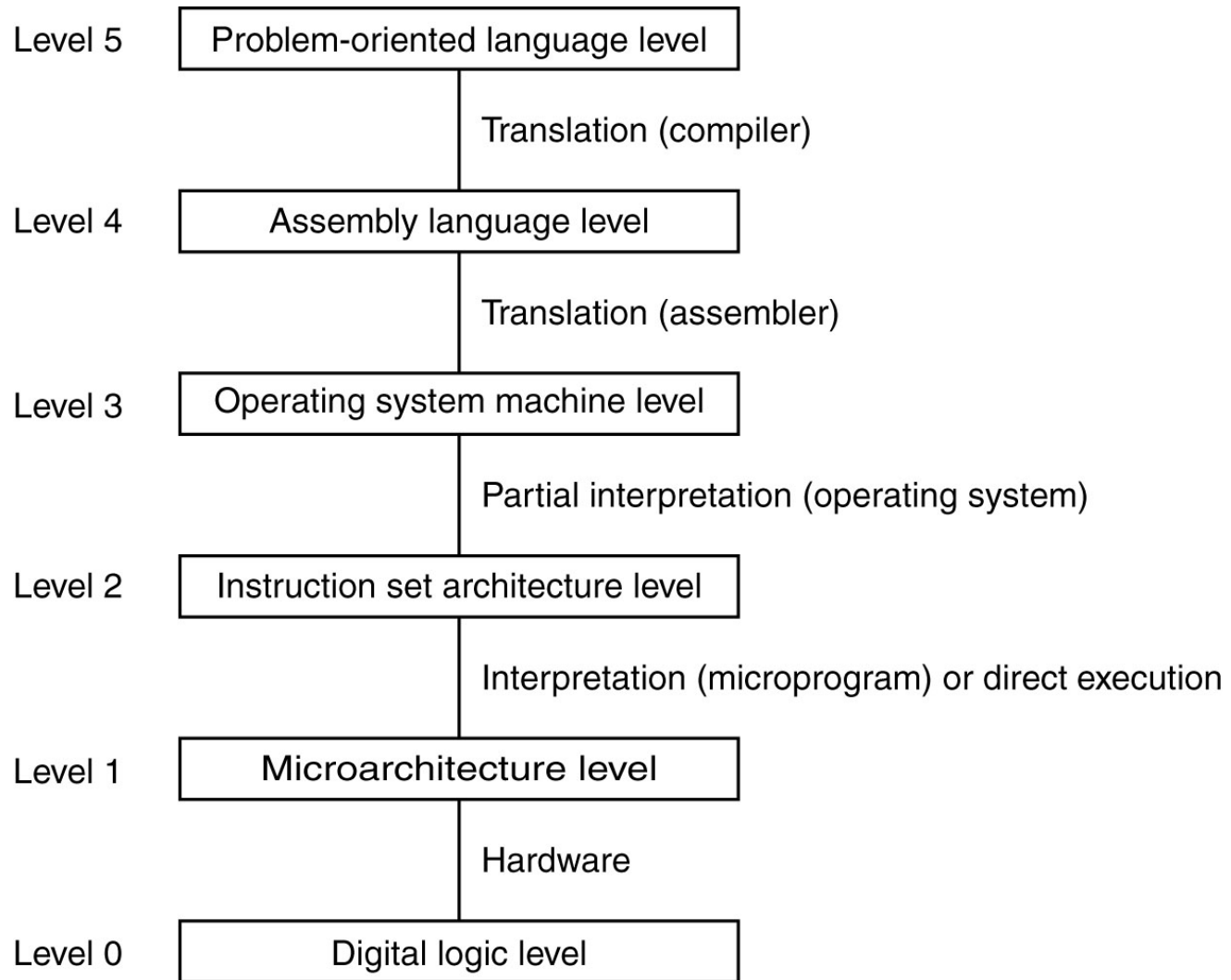
Virtual Machines

- Σκεφτείτε έναν υπολογιστή που εκτελεί προγράμματα σε γλώσσα L1
- Μπορεί να είναι κατασκευασμένος σε **hardware**
 - Αν η L1 είναι υψηλού επιπέδου → πολύ περίπλοκος ο σχεδιασμός του
 - Όποτε η L1 αλλάζει → Πρέπει να ξανασχεδιάζουμε το μηχάνημα από την αρχή
 - Όλα τα προγράμματα πρέπει να γράφονται σε L1
- Εναλλακτικά, θα μπορούσαμε να υλοποιήσουμε σε **software** ένα “virtual machine” που εκτελεί προγράμματα σε L1
 - Το L1 virtual machine τρέχει στο μηχάνημα L0
 - και το L0 μπορεί επίσης να είναι virtual!
 - Όταν αλλάζει η L1 → αλλάζουμε μόνο το virtual machine software
 - Για να υποστηρίξουμε τη γλώσσα L2, φτιάχνουμε απλά ένα νέο VM
 - Σε L0 ή ακόμα και σε L1
- Οι σημερινοί υπολογιστές έχουν πολλά επίπεδα “virtual machines”
 - Συνήθως περίπου 6

Layered view of computer organization



Layered view of computer organization



Από τη Java στα Transistors

Running a Java program

```
public class Foo
{
    public static void main(String[] args)
    {
        int i = 18;
        int j = 21;
        int k = i + j;
        System.out.println("18+21=" + k);
    }
}
```

```
$ javac Foo
$
```

```
$ java Foo
18+21=39
$
```

Magic!!!

L6: Problem-oriented language

L6

L5

L4

L3

L2

L1

L0

□ Java

```
public class Foo
{
    public static void main(String[] args)
    {
        int i = 18;
        int j = 21;
        int k = i + j;
        System.out.println("18+21=" + k);
    }
}
```

L5: Java Bytecode

- Open the Java Bytecode (class file) in a text editor:

L6
L5
L4
L3
L2
L1
L0

```

Ép³¼^@^@^@2^@+
^@^K^@^T   ^@^U^@^V^G^@^W
^@^C^@^T^H^@^X
^@^C^@^Y
^@^C^@^Z
^@^C^@^[
^@^\\^@^]^G^@^^^G^@^_ ^A^@^F<init>^A^@^C()V^A^@^DCode^A^@^OLineNumberTable^A^@^Dmain^A^@^V([L java/lang/Str
ing;)V^A^@
SourceFile^A^@^HFoo.java^L^@^L^@^M^G^@ ^L^@!^@" ^A^@^Wjava/lang/StringBuilder^A^@^F17+21=^L^@#^@#^L^@#^@%2
^L^@%&^@ ^G^@(^L^@)^@*^A^@^C^F^@^P java/lang/Object^A^@^P java/lang/System^A^@^C^@^UL java/io/PrintS2
tream:^A^@^Fappend^A^@-(L java/lang/String;)L java/lang/StringBuilder:^A^@^(I)L java/lang/StringBuilder:^A2
^@^HtoString^A^@^T(L java/lang/String:^A^@^S java/io/PrintStream^A^@^GprintLn^A^@^U(L java/lang/String;)V^2
@!^@
^@^K^@^@^@^@^@B^@^A^@^L^@^M^@^A^@^N^@^@^@^@] ^@^A^@^A^@^@^@^E* ^@^A±^@^@^@^@^@^@^@^@F^@^A^@^@^@^@^@
^@^P^@^Q^@^A^@^N^@^@^@L^@^C^@^@^@^@#^@P^@Q<^@P^@U=^[^@\\]^@^B>^@^@CY ^@^D^@R^@E^@^@F^@] ^@^@G^@^@H^@^@ ±^@^@^@^@2
A^@^D^@^@^@^@V^@^E^@^@^@^@C^@^C^@^D^@^@F^@^E^@
^@^F^@#^@^G^@^A^@^R^@^@^@^@B^@^S
  
```

L5: Java Bytecode

- Open it in binary mode:

L6
L5
L4
L3
L2
L1
L0

```

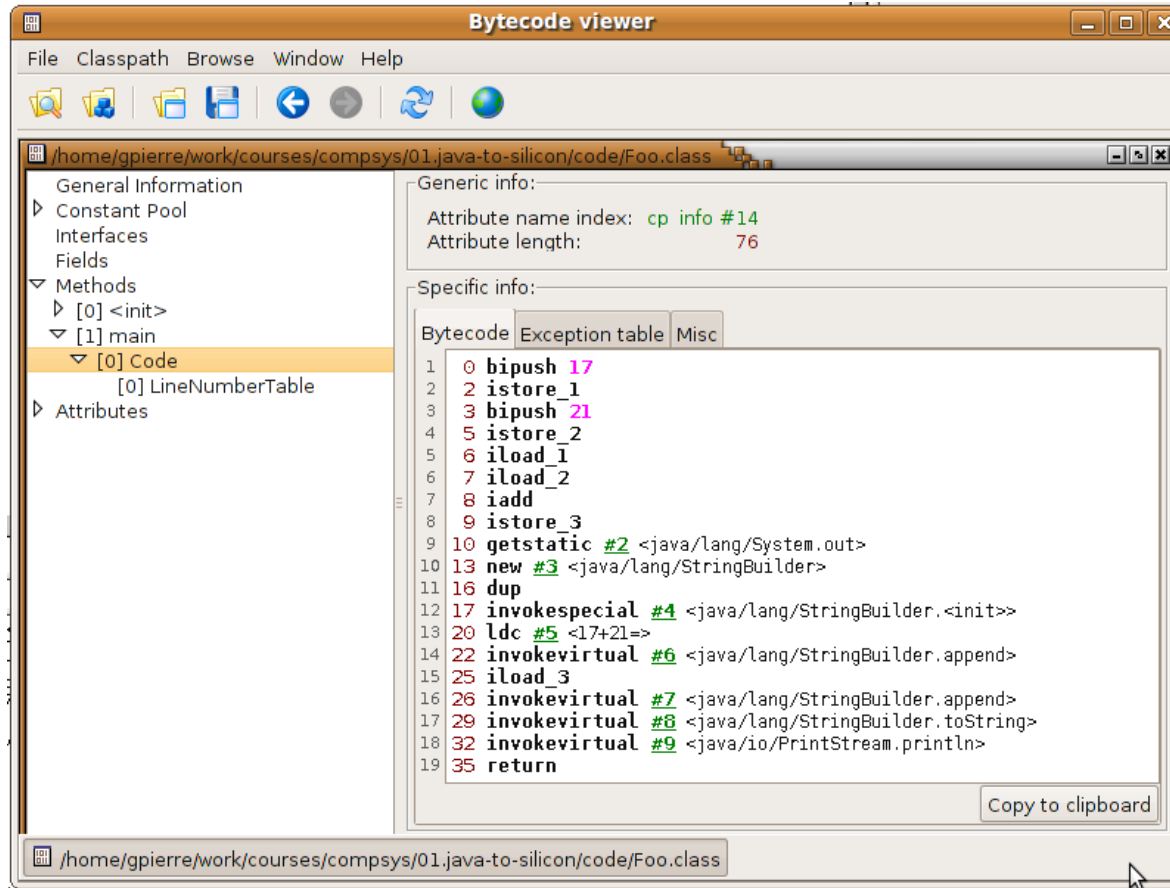
emacs: Foo.class
File Edit View Cmds Tools Options Buffers Help
00000000: cafe babe 0000 0032 002b 0a00 0b00 1409 .....2.+.....
00000010: 0015 0016 0700 170a 0003 0014 0800 180a .....
00000020: 0003 0019 0a00 0300 1a0a 0003 001b 0a00 .....
00000030: 1c00 1d07 001e 0700 1f01 0006 3c69 6e69 .....<ini
00000040: 743e 0100 0328 2956 0100 0443 6f64 6501 t>...()V...Code.
00000050: 000f 4c69 6e65 4e75 6d62 6572 5461 626c ..LineNumberTabl
00000060: 6501 0004 6d61 696e 0100 1628 5b4c 6a61 e...main...<[Ja
00000070: 7661 2f6c 616e 672f 5374 7269 6e67 3b29 va/lang/String;)
00000080: 5601 000a 536f 7572 6365 4669 6c65 0100 V...SourceFile..
00000090: 0846 6f6f 2e6a 6176 610c 000c 000d 0700 .Foo.java.....
000000a0: 200c 0021 0022 0100 176a 6176 612f 6c61 ..!"...java/la
000000b0: 6e67 2f53 7472 696e 6742 7569 6c64 6572 ng/StringBuilder
000000c0: 0100 0631 372b 3231 3d0c 0023 0024 0c00 ...17+21=...#,$..
000000d0: 2300 250c 0026 0027 0700 280c 0029 002a #.%..&..'(..)*
000000e0: 0100 0346 6f6f 0100 106a 6176 612f 6c61 ...Foo...java/la
000000f0: 6e67 2f4f 626a 6563 7401 0010 6a61 7661 ng/Object....java
00000100: 2f6c 616e 672f 5379 7374 656d 0100 036f /lang/System...o
00000110: 7574 0100 154c 6a61 7661 2f69 6f2f 5072 ut...L.java/io/Pr
00000120: 696e 7453 7472 6561 6d3b 0100 0661 7070 intStream:...app
00000130: 656e 6401 002d 284c 6a61 7661 2f6c 616e end..-<L.java/lan
00000140: 672f 5374 7269 6e67 3b29 4c6a 6176 612f g/String;)L.java/
00000150: 6c61 6e67 2f53 7472 696e 6742 7569 6c64 lang/StringBuild
00000160: 6572 3b01 001c 2849 294c 6a61 7661 2f6c er:...<I>L.java/l
00000170: 616e 672f 5374 7269 6e67 4275 696c 6465 ang/StringBuilde
00000180: 723b 0100 0874 6f53 7472 696e 6701 0014 r:...toString...
00000190: 2829 4c6a 6176 612f 6c61 6e67 2f53 7472 (<L.java/lang/Str
000001a0: 696e 673b 0100 136a 6176 612f 696f 2f50 ing:...java/io/P
000001b0: 7269 6e74 5374 7265 616d 0100 0770 7269 rintStream...pri
000001c0: 6e74 6c6e 0100 1528 4c6a 6176 612f 6c61 ntln...<L.java/la
000001d0: 6e67 2f53 7472 696e 673b 2956 0021 000a ng/String;)V!..
000001e0: 000b 0000 0000 0002 0001 000c 000d 0001 .....
000001f0: 000e 0000 001d 0001 0001 0000 0005 2ab7 .....*
00000200: 0001 b100 0000 0100 0f00 0000 0600 0100 .....
00000210: 0000 0100 0900 1000 1100 0100 0e00 0000 .....
00000220: 4c00 0300 0400 0000 2410 113c 1015 3d1b L.....$...<..=
--0x0=0--Raw---XEmacs: Foo.class [(Hex) Fill(adapt)]---L1--Top-----

```

L5: Java Bytecode

- Use a Java Bytecode editor:

L6
L5
L4
L3
L2
L1
L0



<http://www.ej-technologies.com/products/jclasslib/overview.html>

L4: Assembly Language

L6
L5
L4
L3
L2
L1
L0

- Ποια η σχέση **machine code** και **assembly**;;;
 - Αν και είναι πολύ κοντά σε machine code, η assembly εξακολουθεί να είναι μια γλώσσα κατανοητή σε ανθρώπους (σχετικά 😊)
 - Ένα πρόγραμμα assembly πρέπει να γίνει compiled σε machine code από έναν assembler για να τρέξει

- Το JVM κάνει interpretation του προγράμματος “Foo.class” από **Java Bytecode** σε **machine code**
 - Το JVM είναι γραμμένο σε machine code (εκτελέσιμο)
 - Χαμηλότερου επιπέδου από το Java Bytecode
 - Πολύ χαμηλότερου επιπέδου απο την ίδια τη Java!

L4: Assembly Language

L6
L5
L4
L3
L2
L1
L0

```

spyros@chaos: ~
File Edit View Search Terminal Help
#include <stdio.h>
int main()
{
    int i = 17;
    int j = 21;
    int k = i+j;
    printf("i+j=%d\n", k);
    return 0;
}
1,1 Top

```

Note -- You can get the Assembly language of a C program by:

```

$ gcc -S foo.c
$

```

```

spyros@chaos: ~
File Edit View Search Terminal Help
.file "foo.c"
.section .rodata
.LC0:
.string "i+j=%d\n"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $16, %rsp
movl $17, -12(%rbp)
movl $21, -8(%rbp)
movl -8(%rbp), %eax
movl -12(%rbp), %edx
addl %edx, %eax
movl %eax, -4(%rbp)
movl $.LC0, %eax
movl -4(%rbp), %edx
movl %edx, %esi
movq %rax, %rdi
movl $0, %eax
call printf
movl $0, %eax
leave
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE0:
.size main, .-main
.ident "GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
.section .note.GNU-stack,"",@progbits
1,2-11 All

```

L3: Operating System

- “user” and “system” both take large percentages of CPU time

```

spyros@chaos: ~
File Edit View Search Terminal Help
top - 19:58:41 up 8:20, 7 users, load average: 1.49, 1.49, 1.22
Tasks: 173 total, 1 running, 172 sleeping, 0 stopped, 0 zombie
%Cpu(s): 57.8 us, 42.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.2 si, 0.0 st
KiB Mem: 5978908 total, 3783148 used, 195760 free, 281732 buffers
KiB Swap: 4124668 total, 1372 used, 4123296 free, 959780 cached

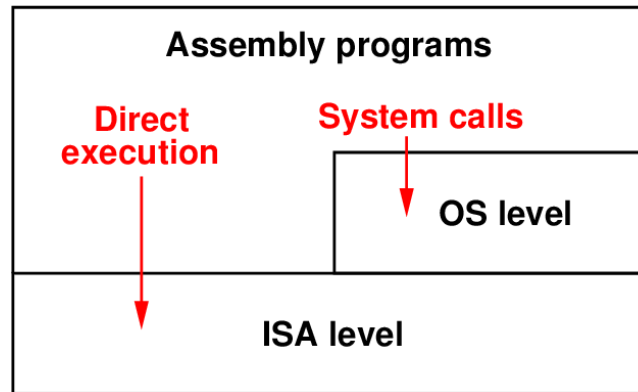
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
9903	spyros	20	0	2018m	15m	8996	S	196.5	0.4	0:40.13	java
2589	spyros	20	0	1824m	259m	35m	S	1.3	6.7	10:34.32	gnome-shell
1599	root	20	0	196m	34m	17m	S	0.3	0.9	1:18.44	Xorg
2889	spyros	20	0	749m	29m	13m	S	0.3	0.8	0:05.63	gnome-terminal
2959	spyros	20	0	123m	2412	1736	S	0.3	0.1	0:26.14	synergyc
9545	spyros	20	0	24792	1604	1148	S	0.3	0.0	0:00.85	top
9866	spyros	20	0	24800	1608	1148	R	0.3	0.0	0:00.25	top
1	root	20	0	24604	2592	1384	S	0.0	0.1	0:00.51	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:03.26	ksoftirqd/0
6	root	rt	0	0	0	0	S	0.0	0.0	0:00.02	migration/0
7	root	rt	0	0	0	0	S	0.0	0.0	0:00.14	watchdog/0
8	root	rt	0	0	0	0	S	0.0	0.0	0:00.11	migration/1
10	root	20	0	0	0	0	S	0.0	0.0	0:01.86	ksoftirqd/1
11	root	rt	0	0	0	0	S	0.0	0.0	0:00.16	watchdog/1
12	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	cpuset
13	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	khelper

L3: Operating System

- Τα εκτελέσιμα προγράμματα «πατάνε» σε δύο χαμηλότερα επίπεδα:
 - Στο Λειτουργικό Σύστημα
 - Και στο επίπεδο ISA (Instruction Set Architecture)

L6
L5
L4
L3
L2
L1
L0



L3: Operating System

Note – You can see which system calls a program makes by:

```
$ strace <prog>
```

```
spyros@chaos: ~/tmp
```

```
File Edit View Search Terminal Help
```

```
execve("./foo", [ "./foo" ], [ /* 52 vars */ ]) = 0
brk(0) = 0x112d000
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd830a5e000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=104790, ...}) = 0
mmap(NULL, 104790, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd830a44000
close(3) = 0
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\200\30\2\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1811160, ...}) = 0
mmap(NULL, 3925240, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd83047f000
mprotect(0x7fd830634000, 2093056, PROT_NONE) = 0
close(3) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd830a43000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd830a42000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd830a41000
arch_prctl(ARCH_SET_FS, 0x7fd830a42700) = 0
mprotect(0x7fd830833000, 16384, PROT_READ) = 0
mprotect(0x6000000, 4096, PROT_READ) = 0
mprotect(0x7fd830a60000, 4096, PROT_READ) = 0
munmap(0x7fd830a44000, 104790) = 0
fstat(1, {st_mode=S_IFREG|0664, st_size=1777, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd830a5d000
write(1, "Hello, World!", 13Hello, World!) = 13
exit_group(0) = ?
```

```
spyros@chaos: ~/tmp
```

```
File Edit View Search Terminal Help
```

```
#include <stdio.h>
int main()
{
    printf("Hello, World!");
    return 0;
}
```

```
6,2
```

```
Top
```

```
28,24
```

```
All
```

L6
L5
L4
L3
L2
L1
L0

L2: Instruction Set Architecture

L6
L5
L4
L3
L2
L1
L0

- Ο κάθε επεξεργαστής έχει ένα σετ εντολών μηχανής (machine code), που λέγεται Instruction Set Architecture.
 - Π.χ., x86, SPARC, ARM, ...
- Οι compilers μεταγλωτίζουν ένα πρόγραμμα υψηλότερου επιπέδου σε machine code του εκάστοτε επεξεργαστή

L1: Microarchitecture

L6
L5
L4
L3
L2
L1
L0

- Πως «ξέρει» ένας επεξεργαστής να εκτελέσει μια εντολή;
 - Οι εντολές μηχανής είναι υλοποιημένες σε ένα χαμηλότερο επίπεδο, αυτό του μικροπρογράμματος (**micro-program**)
 - Το micro-program επίπεδο ουσιαστικά υλοποιεί ένα **virtual machine** που επιτρέπει την εκτέλεση machine code (ISA level)
- Γιατί να μην υλοποιήσουμε τις εντολές machine code απευθείας σε hardware;
 - Γιατί είναι απλούστερο να έχουμε ένα ακόμα επίπεδο!
 - Επίσης βοηθάει στην υλοποίηση CPU pipelining
 - Αλλά γι' αυτά παρακολουθήστε ένα μάθημα αρχιτεκτονικής υπολογιστών 😊

L0: Digital gates

L6
L5
L4
L3
L2
L1
L0

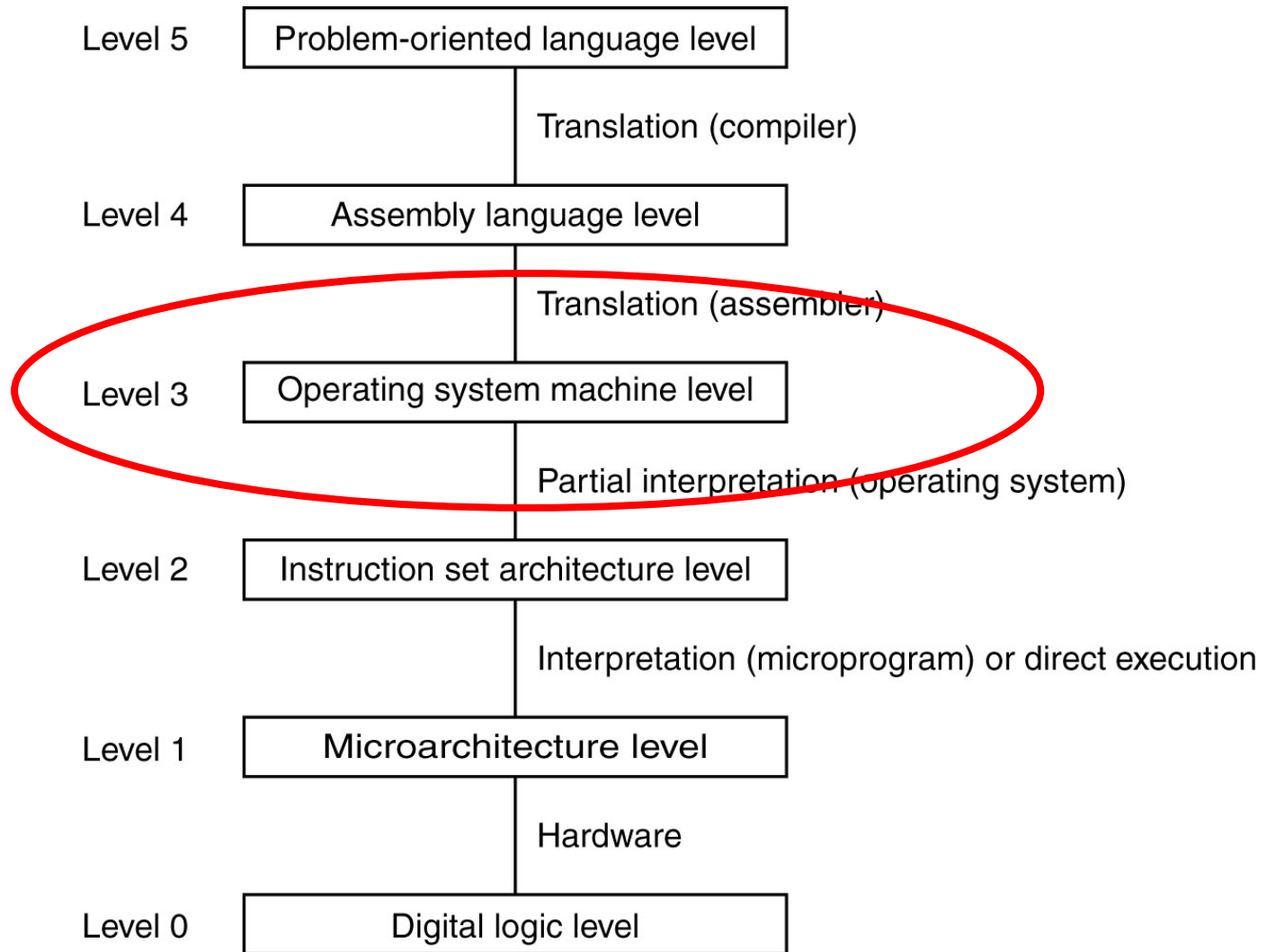
- Το hardware κάνει όλο το **hard work!**
 - Store data in memory, read/write memory, copy data
 - Arithmetic operations (add, subtract, multiply, ...)
 - Issue I/O operations

- Αυτά εκτελούνται με ψηφιακές πύλες (**digital gates**)
 - Digital gates manipulate only **boolean values** (either 0 or 1)
 - a **AND** b
 - a **OR** b
 - **NOT** a

- Οι ψηφιακές πύλες υλοποιούνται με **transistors**

- Transistors are magic! 😊
 - Ρωτήστε τον Ηλεκτρολόγο Μηχανικό της γειτονιάς σας για το πως δουλεύουν...!

Operating System



Λειτουργικό Σύστημα

- Το ΛΣ είναι ουσιαστικά ένας **διαχειριστής πόρων του συστήματος!**
 - Ίσως ο πιο «ασφαλής» ορισμός
 - Εξασφαλίζει ότι πολλά προγράμματα (διεργασίες) μπορούν να τρέχουν ταυτόχρονα
 - χωρίς να παρεμβαίνει το ένα στο άλλο
 - έχοντας το καθένα την «ψευδαίσθηση» πλήρους κυριότητας του μηχανήματος
 - Διαχειρίζεται τις περιφερειακές συσκευές
 - Επιτρέποντας τη χρήση τους στις διεργασίες που τις χρειάζονται
 - Παρέχοντας ένα ομοιόμορφο interface για την προσπέλασή τους

Processes (Διεργασίες – Προγράμματα - Υποπρογράμματα)

- Ένα process είναι το πλαίσιο στο οποίο εκτελείται ένα πρόγραμμα
 - Διαθέτει ιδιωτικό memory address space (ξεχωριστό από άλλες διεργασίες)
 - Διαθέτει ιδιωτικό stack, PC & SP registers, κ.τ.λ.

- Το ΛΣ αναλαμβάνει το process scheduling (χρονοπρογραμματισμό διεργασιών)
 - Ανά πάσα στιγμή μόνο ένα process μπορεί να τρέχει σε ένα CPU core
 - Το ΛΣ φροντίζει να εναλλάσσονται πολλά processes στα διαθέσιμα cores
 - Δίνει την ψευδαίσθηση ότι όλα τρέχουν ταυτόχρονα, ακόμα κι όταν είναι περισσότερα από τον αριθμό των cores (το οποία πρακτικά συμβαίνει πάντα)
 - Φροντίζει να κατανέμει τους πόρους και τον χρόνο εκτέλεσης δίκαια

- Οι διεργασίες είναι isolated
 - Κάθε διεργασία τρέχει «μόνη της»
 - Έχει την ψευδαίσθηση ότι έχει όλο το μηχάνημα στην αποκλειστική διάθεσή της
 - Προαιρετικά, οι διεργασίες μπορούν να επικοινωνήσουν μεταξύ τους
 - Αυτό γίνεται μέσω τεχνικών Inter-Process Communication (IPC) που προσφέρονται επίσης από το ΛΣ

Input/Output

- Οι διεργασίες θέλουν να αλληλεπιδράσουν με διάφορες συσκευές
 - Οθόνη, πληκτρολόγιο, **δίσκο**, εκτυπωτές, δίκτυο, κάρτα ήχου, USB, κ.τ.λ.

- Κάθε συσκευή είναι διαφορετική!
 - Όχι μόνο οι διαφορετικού τύπου (π.χ., εκτυπωτής και δίσκος)
 - Αλλά και οι συσκευές ίδιου τύπου
 - Π.χ., εκατοντάδες μοντέλα εκτυπωτών
 - Π.χ., εκατοντάδες μοντέλα δίσκων
 - κ.τ.λ.
 - Διαφορετικοί κατασκευαστές
 - Διαφορετικές ταχύτητες

- Το ΛΣ προσφέρει
 - **Ομοιογενές interface** σε όλα τα είδη διαφορετικών συσκευών
 - **Διαιτησία (arbitration)** ανάμεσα στις διεργασίες που θέλουν να χρησιμοποιήσουν την ίδια συσκευή ταυτόχρονα

Memory Management

- Οι διεργασίες χρειάζονται μνήμη
 - Αλλά θέλουμε η κάθε διεργασία να είναι απομονωμένη από τις άλλες
 - Καλύτερα: Δώσε σε κάθε διεργασία την ψευδαίσθηση ότι έχει όλη τη μνήμη του υπολογιστή στη διάθεσή της
 - Ακόμα καλύτερα: Δώσε σε κάθε διεργασία την ψευδαίσθηση ότι έχει στη διάθεσή της όλον τον χώρο διευθύνσεων (address space), άσχετα με το πόσο λιγότερη μνήμη έχει ο υπολογιστής στην πραγματικότητα

- Για να πετύχει τα παραπάνω, το ΛΣ πρέπει να παρέχει:
 - Memory isolation
 - Paging (δηλ., να μεταφέρει κομμάτια της μνήμης στον δίσκο όταν δεν χωράνε όλα στη μνήμη)
 - Virtual Memory (δηλ., να δίνει στην κάθε διεργασία την ψευδαίσθηση πλήρους κυριότητας σε όλο το address space)