RISC V Architecture

Dr. Santhameena.S

Department of Electronics and Communication Engineering

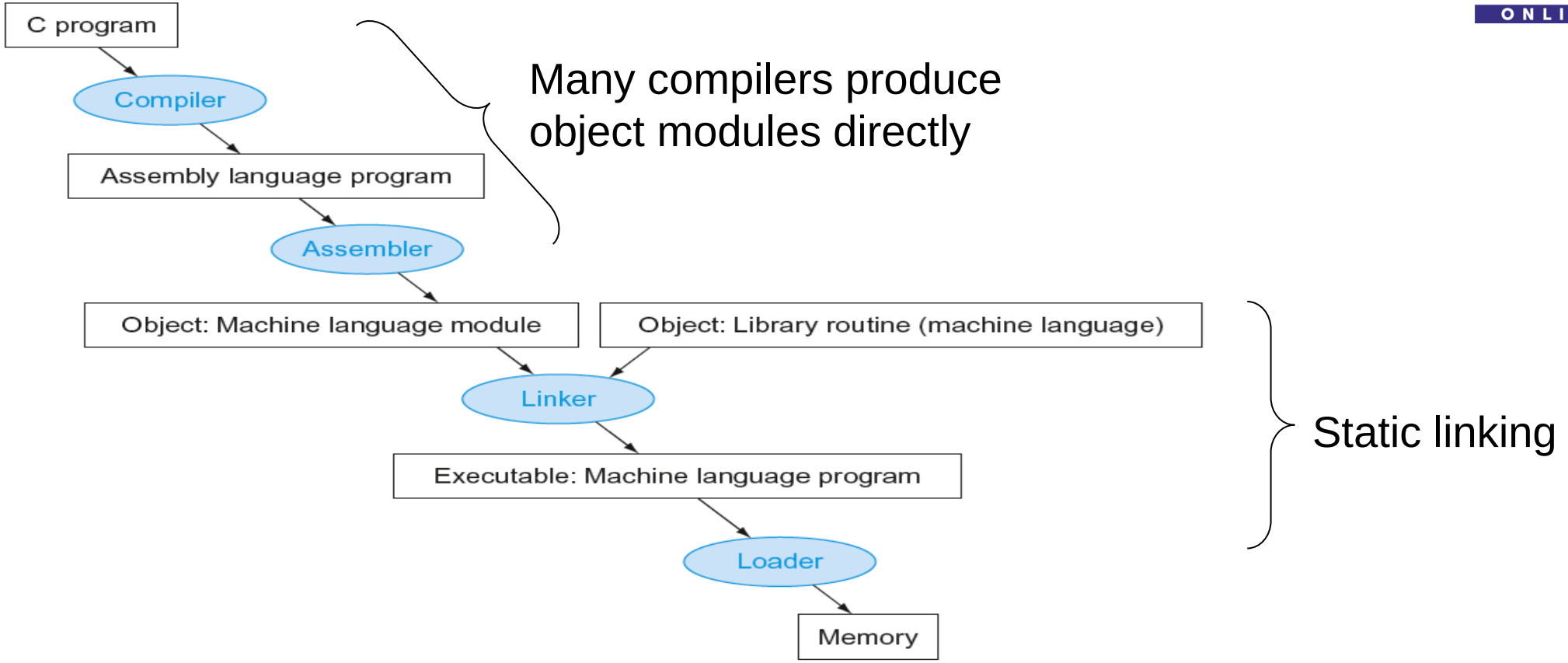
RISC V Architecture

UNIT 3: Instructions: The Language of Computer

Dr. Santhameena.S
Department of Electronics and Communication Engineering

Instructions – Language of Computer

Translation and Startup



Instructions – Language of Computer

Assembler- pseudoinstructions

```
li x9, 123 // load immediate value 123 into register x9
        addi x9, x0, 123 // register x9 gets register x0 + 123

li x9,0x8123
        lui x9 0x8
        addi x9 x9 291

la x10,num
        auipc x10 0x10000
        addi x10 x10 0
```

Instructions – Language of Computer

Assembler- pseudoinstructions

mv x10, x11 // register x10 gets register x11

addi x10, x11, 0 // register x10 gets register x11 + 0

and x9, x10, 15 // register x9 gets x10 AND 15

andi x9, x10, 15 // register x9 gets x10 AND 15

j Label //unconditionally branch to a label

jal x0, Label //unconditionally branch to a label

Instructions – Language of Computer

Producing an Object Module

- Assembler (or compiler) translates program into machine instructions
- Provides information for building a complete program. The object file for UNIX systems typically contains six distinct pieces:
  - **Header**: described contents of object module
  - **Text segment**: translated instructions- contains the machine language code.
  - **Static data segment**: data allocated for the life of the program
  - **Relocation info**: for contents that depend on absolute location of loaded program
  - **Symbol table**: global definitions and external refs
  - **Debug info**: for associating with source code- contains a concise description of how the modules were compiled so that a debugger can associate machine instructions with C source files and make data structures readable.

Instructions – Language of Computer

Linker

There are three steps for the linker:

1. Place code and data modules symbolically in memory.
2. Determine the addresses of data and instruction labels.
3. Patch both the internal and external references.

Instructions – Language of Computer

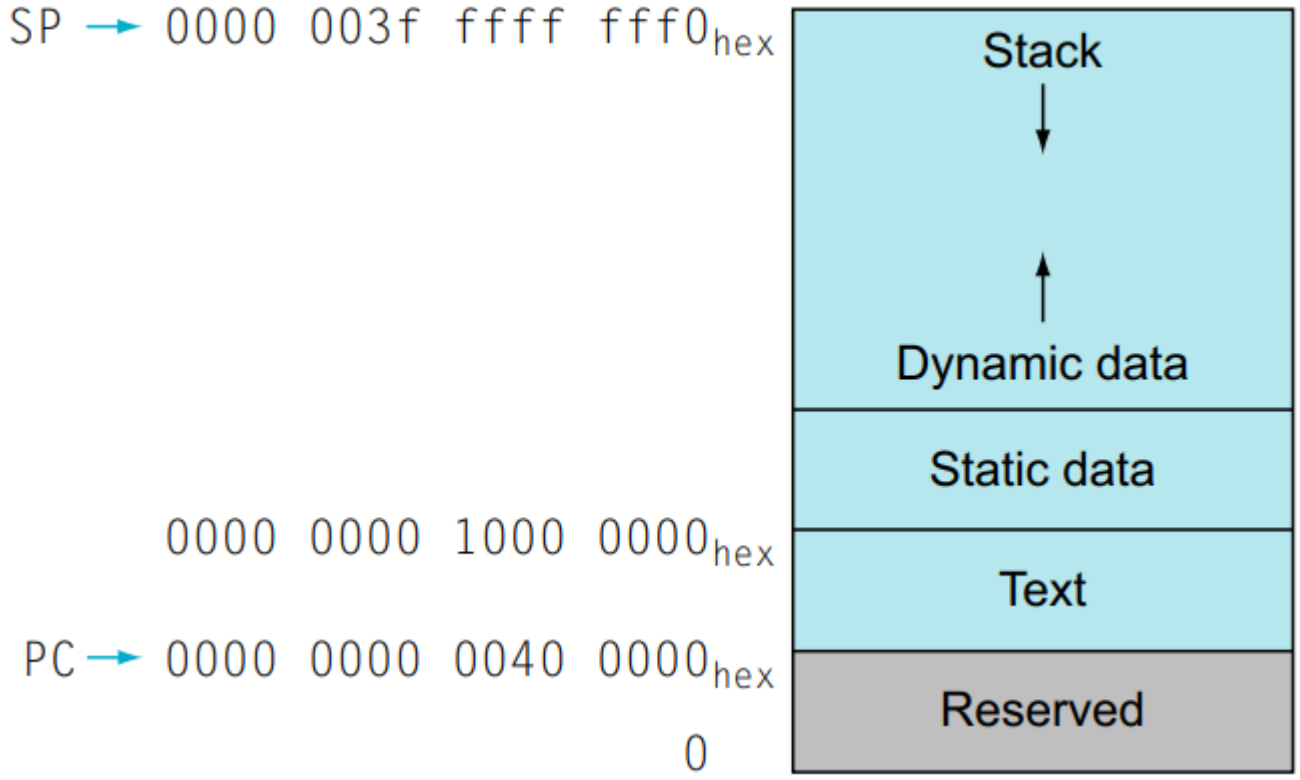
Linking Object Modules

- Produces an executable image
  1. Merges segments
  2. Resolve labels (determine their addresses)
  3. Patch location-dependent and external refs.

  (Typically, this file has the same format as an object file, except that it contains no unresolved references. It is possible to have partially linked files, such as library routines, that still have unresolved addresses and hence result in object files.)
- Could leave location dependencies for fixing by a relocating loader
  - When the linker places a module in memory, all absolute references, that is, memory addresses that are not relative to a register, must be relocated to reflect its true location

Instructions – Language of Computer

The RISC-V memory allocation for program and data



Instructions – Language of Computer
Linking Object Files

Link the two object files below.

Show updated addresses of the first few instructions of the completed executable file.

Object file header			
	Name	Procedure A	
	Text size	100_{hex}	
	Data size	20_{hex}	
Text segment	Address	Instruction	
	0	`lw x10, 0(x3)`	
	4	`jal x1, 0`	
	. . .	. . .	
Data segment	0	(X)	
	. . .	. . .	
Relocation information	Address	Instruction type	Dependency
	0	`lw`	X
	4	`jal`	B
Symbol table	Label	Address	
	X	–	
	B	–	
	Name	Procedure B	
	Text size	200_{hex}	
	Data size	30_{hex}	
Text segment	Address	Instruction	
	0	`sw x11, 0(x3)`	
	4	`jal x1, 0`	
	. . .	. . .	
Data segment	0	(Y)	
	. . .	. . .	
Relocation information	Address	Instruction type	Dependency
	0	`sw`	Y
	4	`jal`	A
Symbol table	Label	Address	
	Y	–	
	A	–	

Executable file header		
	Text size	300_{hex}
	Data size	50_{hex}
Text segment	Address	Instruction
	$0000\ 0000\ 0040\ 0000_{hex}$	`lw x10, 0(x3)`
	$0000\ 0000\ 0040\ 0004_{hex}$	`jal x1,` 252_{ten}
	. . .	. . .
	$0000\ 0000\ 0040\ 0100_{hex}$	`sw x11, 32(x3)`
	$0000\ 0000\ 0040\ 0104_{hex}$	`jal x1,` -260_{ten}
	. . .	. . .
Data segment	Address	
	$0000\ 0000\ 1000\ 0000_{hex}$	(X)
	. . .	. . .
	$0000\ 0000\ 1000\ 0020_{hex}$	(Y)
	. . .	. . .

Instructions – Language of Computer

Loading a Program

The loader follows these steps in UNIX systems:

1. Reads the executable file header to determine size of the text and data segments.
2. Creates an address space large enough for the text and data.
3. Copies the instructions and data from the executable file into memory.
4. Copies the parameters (if any) to the main program onto the stack.
5. Initializes the processor registers and sets the stack pointer to the first free location.
6. Branches to a start-up routine that copies the parameters into the argument registers and calls the main routine of the program. When the main routine returns, the start-up routine terminates the program with an exit system call

Instructions – Language of Computer

Dynamic Linking

- Only link/load library procedure when it is called
  - Requires procedure code to be relocatable
  - Avoids image bloat caused by static linking of all (transitively) referenced libraries
  - Automatically picks up new library versions

Instructions – Language of Computer

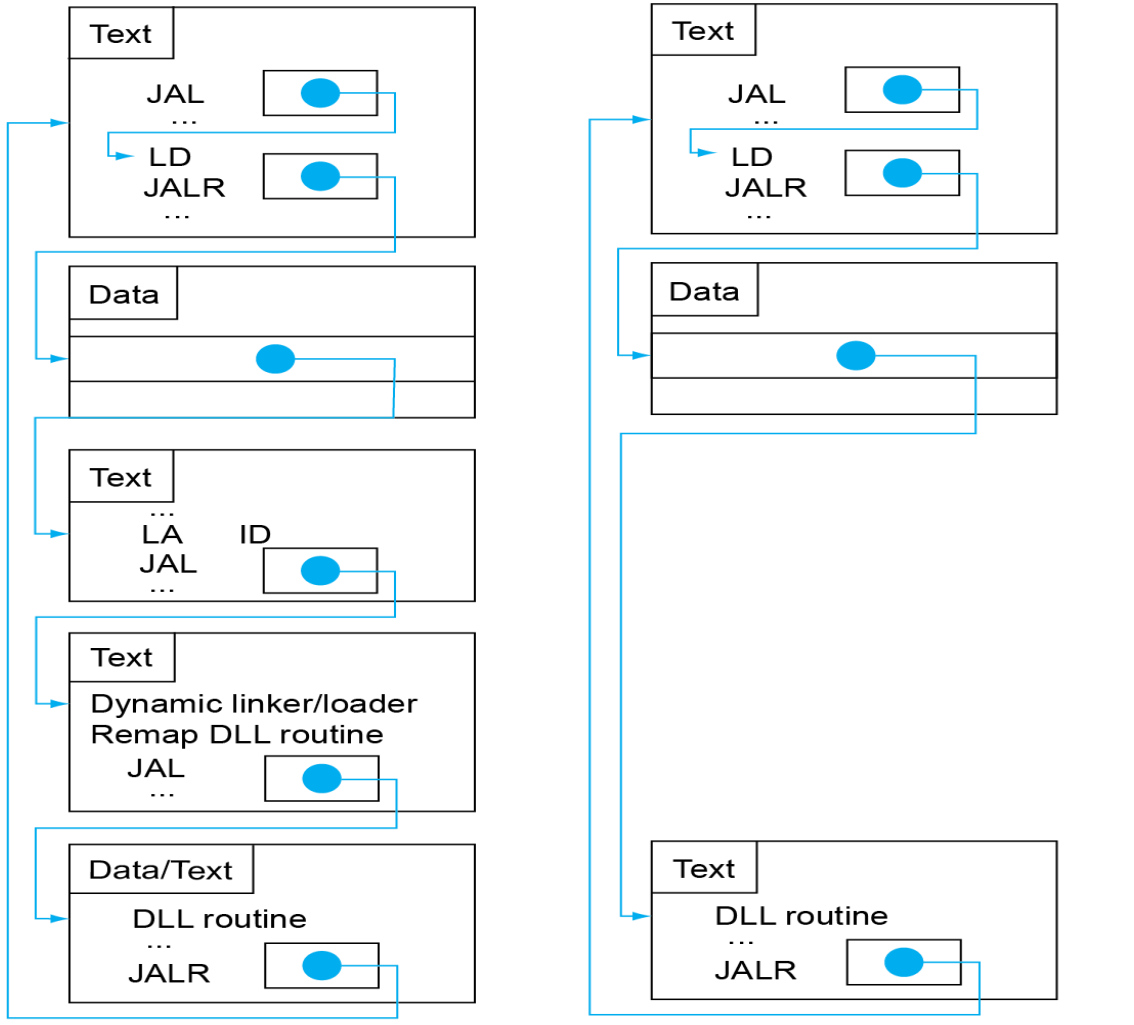
Lazy Linkage

Indirection table

Stub: Loads routine ID,
Jump to linker/loader

Linker/loader code

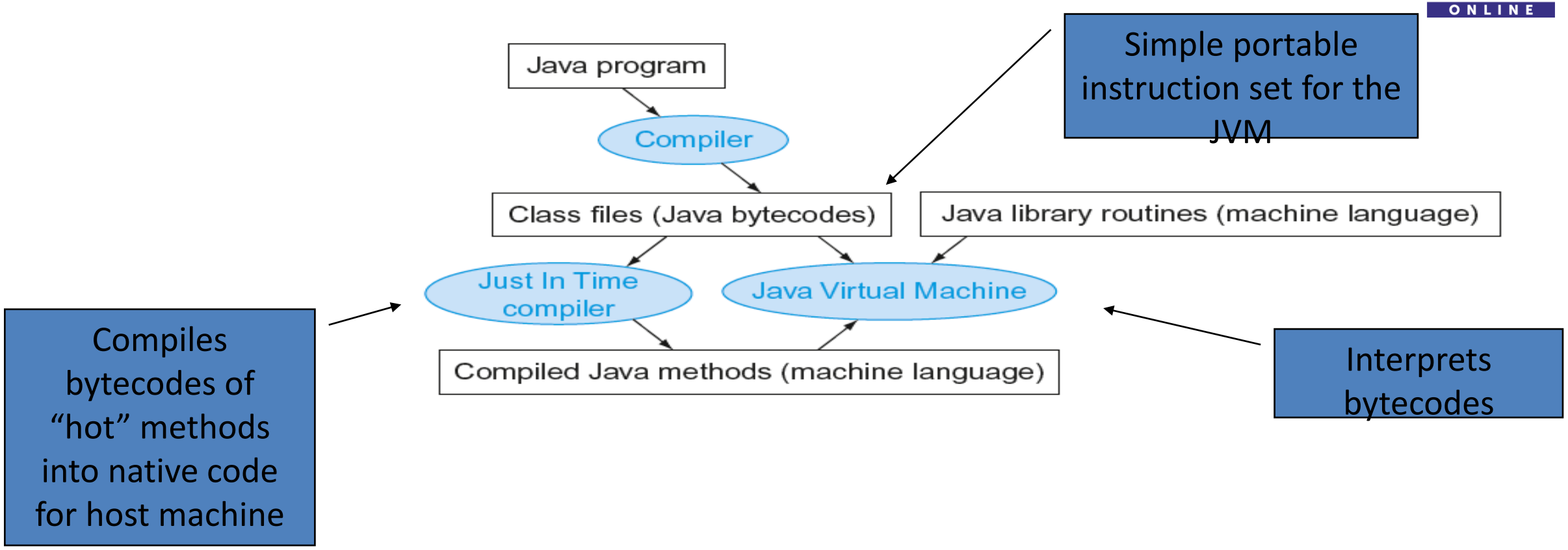
Dynamically
mapped code



(a) First call to DLL routine (b) Subsequent calls to DLL routine

Instructions – Language of Computer

Starting Java Applications



THANK YOU

Dr. Santhameena.S
Department of Electronics and Communication Engineering
santhameena.s@pes.edu