Tikrit University
Electrical Engineering Department

EE-307
Computer Engineering
2024-2025

Instructions:
Translating and Starting a Program

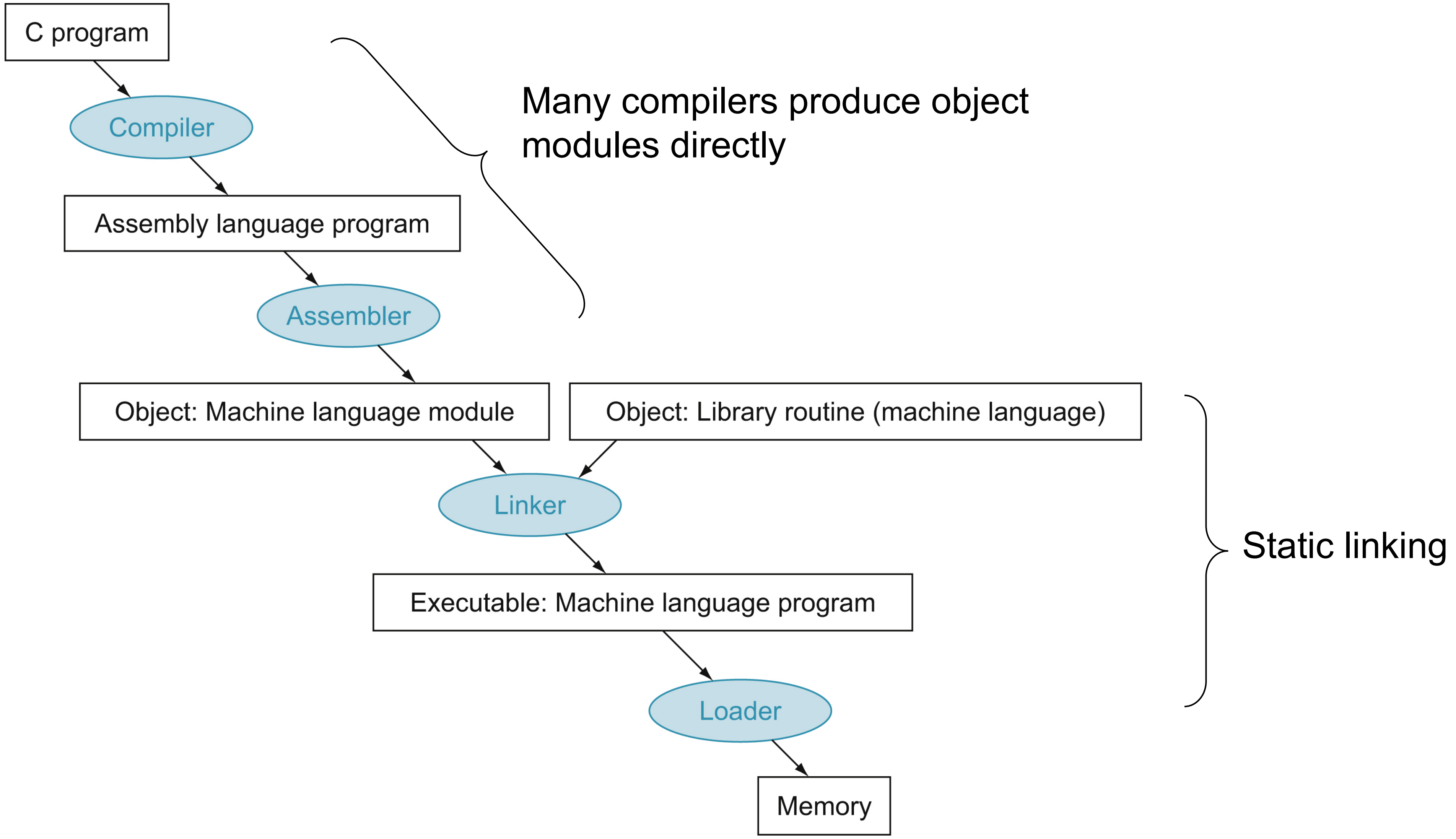
Jalal Nazar Abdulbaqi, Ph.D.

jalal.abdulbaqi@tu.edu.iq

Outline

- Translation and Startup
- Files Extension
- Producing an Object Module
- Linking Object Modules

Translation and Startup


C program
Compiler
Assembly language program
Assembler
Object: Machine language module
Object: Library routine (machine language)
Linker
Executable: Machine language program
Loader
Memory

Many compilers produce object modules directly

Static linking

Translation and Startup

Compiler Transfer the high-level language (HLL) program (e.g., C/C++) into assembly language program.

Assembler Transfer the assembly language program into the machine language (Object File).

Linker Combine multiple object files (the program and its libraries) into one executable file.

Loader Place the executable file into the memory for execution by the processor.

Files Extension

Files Types	UNIX	MS-DOS
C code	.c	.C
Assembly	.s	.ASM
Object file	.o	.OBJ
Statically linked library	.a	.LIB
Dynamically linked library	.so	.DLL
Executable file	.out	.EXE

Producing an Object Module

Provides information for building a complete program from the pieces

- **Header**: described contents of object module
- **Text segment**: translated instructions
- **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

Linking Object Modules

- Produces an executable image
  1. Merges segments
  2. Resolve labels (determine their addresses)
  3. Patch location-dependent and external references
- Could leave location dependencies for fixing by a relocating loader
  - But with virtual memory, no need to do this
  - Program can be loaded into absolute location in virtual memory space

Loading a Program

- Load from image file on disk into memory

1. Read header to determine segment sizes
2. Create virtual address space
3. Copy text and initialized data into memory
    - Or set page table entries so they can be faulted in
4. Set up arguments on stack
5. Initialize registers (including sp, fp, gp)
6. Jump to startup routine
    - Copies arguments to x10, … and calls main
    - When main returns, do exit syscall

Concluding Remarks

- Two stored-program computer principles:
  - the use of instructions that are indistinguishable from numbers
  - the use of alterable memory for programs

- number have no inherent type:
  - A given **bit pattern** can represent an **integer number** or a **string** or a **color** or even **an instruction**.
  - It is the program that determines the type of data.

Concluding Remarks

Three design principles:

1. **Simplicity favors regularity**

   always requiring three register operands in arithmetic instructions

   keeping the register fields in the same place in all instruction formats

2. **Smaller is faster**

   RISC-V has 32 registers rather than many more

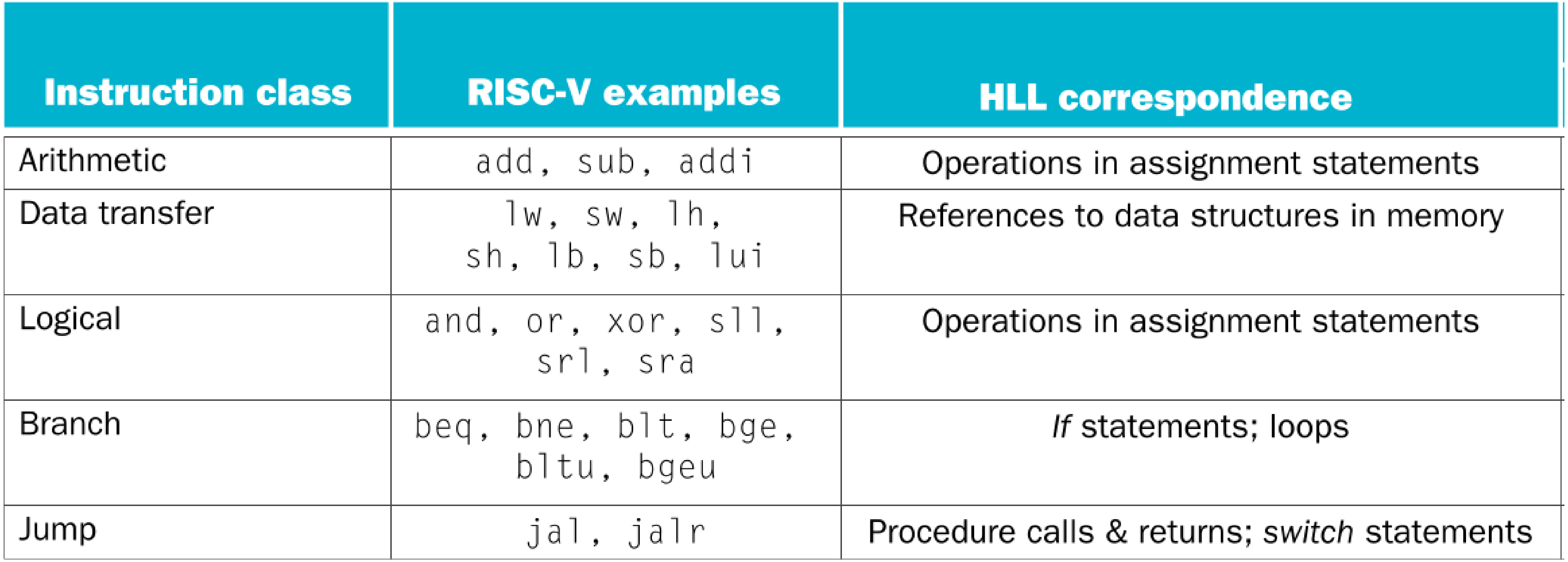
3. **Good design demands good compromises**

   keeping all instructions the same length

Concluding Remarks

- RISC-V instructions categories are associated with constructs that appear in HLL programming languages:
  - **Arithmetic** instructions correspond to the operations found in **assignment statements**.
  - **Transfer** instructions are most likely to occur when dealing with data structures like **arrays** or **structures**.
  - **Conditional branches** are used in **if statements** and in **loops**.
  - **Unconditional branches** are used in **procedure calls** and returns and for **case/switch statements**.

Concluding Remarks



Instruction class	RISC-V examples	HLL correspondence
Arithmetic	add, sub, addi	Operations in assignment statements
Data transfer	lw, sw, lh, sh, lb, sb, lui	References to data structures in memory
Logical	and, or, xor, sll, srl, sra	Operations in assignment statements
Branch	beq, bne, blt, bge, bltu, bgeu	*If* statements; loops
Jump	jal, jalr	Procedure calls & returns; *switch* statements