



Assembly Language Fundamentals



Topics (Chapters 2 and 4)

1. **Assembler**
2. **Assembly File/Program**
3. **Object File/Program**
4. **OP codes and Mnemonics**
5. **Labels and Assembler Directives**
6. **Assembly Process**
7. **Compiler**
8. **Linker**
9. **Loader**
10. **Debugger**



What is an Assembler?

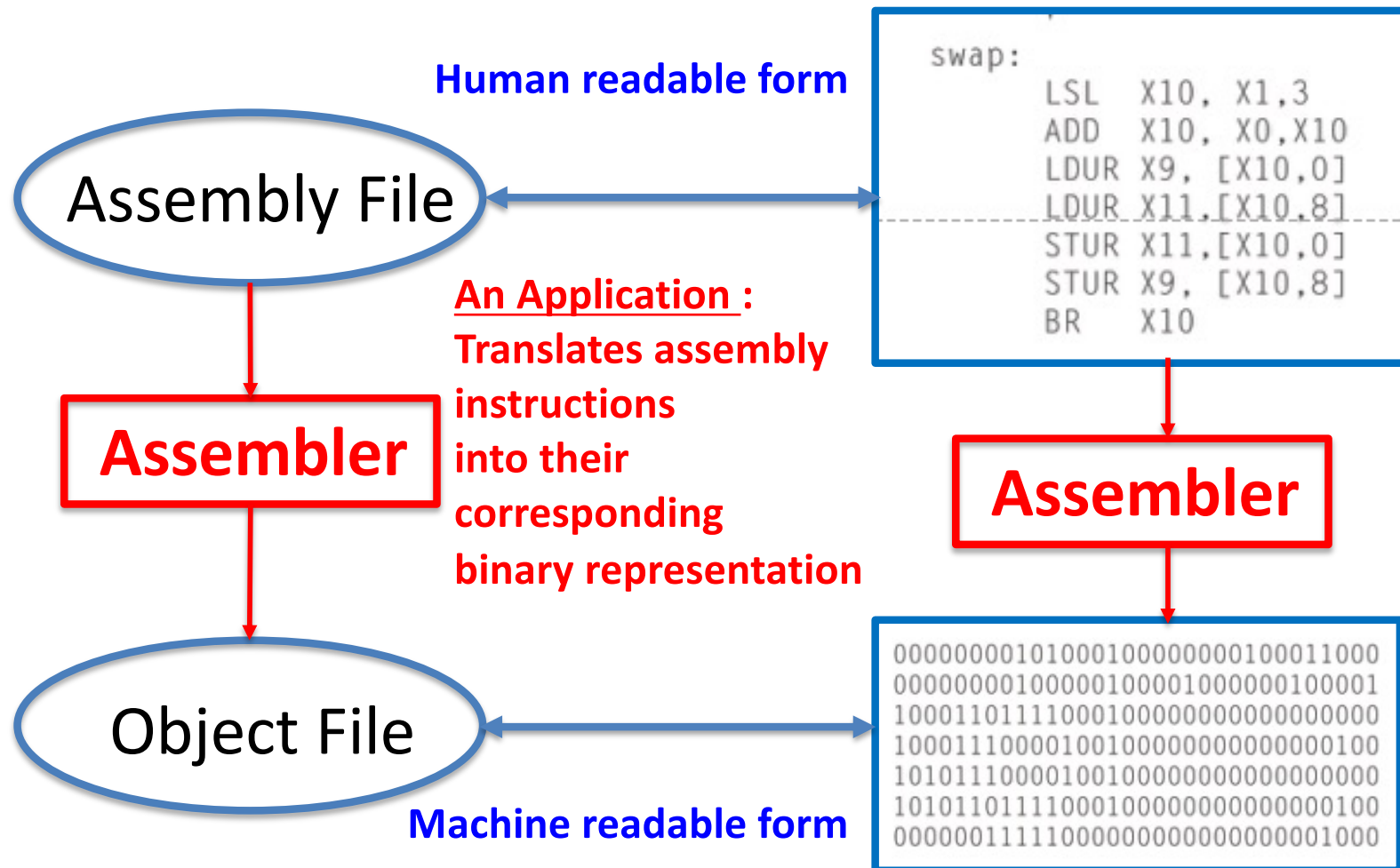
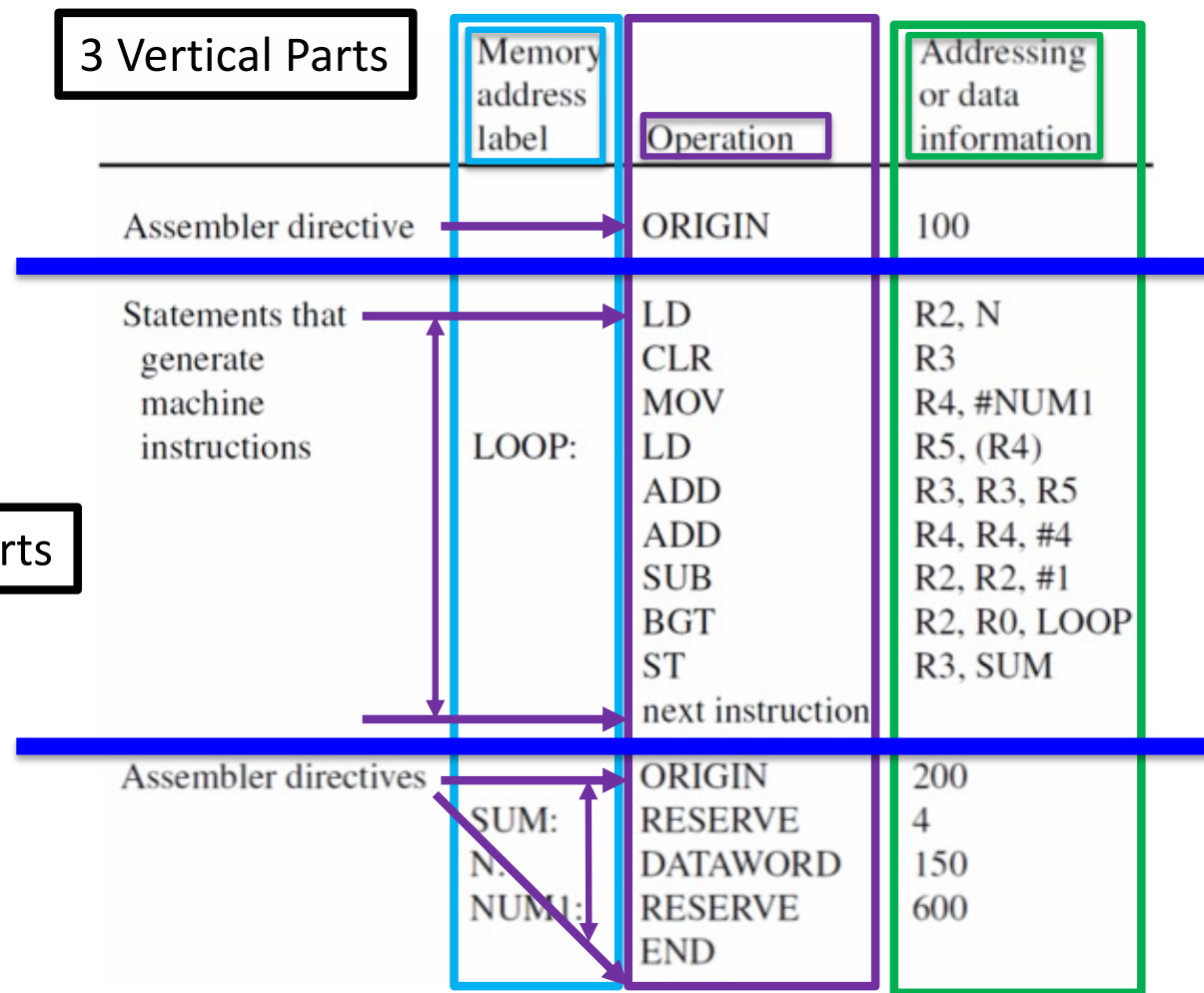




Fig. 2.13 A Sample Assembly Program





2.5 Assembly Instructions



- OP codes (operation codes):

operation	to	from
0110	0010	1100

- Binary pattern suitable for computer processing
- But difficult to communicate and interpret by human

- Mnemonic: e.g., Store, Load, Add, Sub

Load R2, LOC

- Human recognizable alphanumeric format
- Processor specific for manufacturer/model/version

Note: Syntax vs Semantics

- Store/Add (on Processor X) same as ST/AD (on Processor Y)



Assembly Program: Mnemonics & Labels



	Memory address label	Operation	Addressing or data information
Assembler directive		ORIGIN	100
Statements that generate machine instructions		LD	R2, N
		CLR	R3
		MOV	R4, #NUM1
	LOOP:	LD	R5, (R4)
		ADD	R3, R3, R5
		ADD	R4, R4, #4
		SUB	R2, R2, #1
		BGT	R2, R0, LOOP
		ST	R3, SUM
		next instruction	
Assembler directives		ORIGIN	200
		RESERVE	4
		DATAWORD	150
		RESERVE	600
		END	

**Mnemonics: human
recognizable format**

Labels



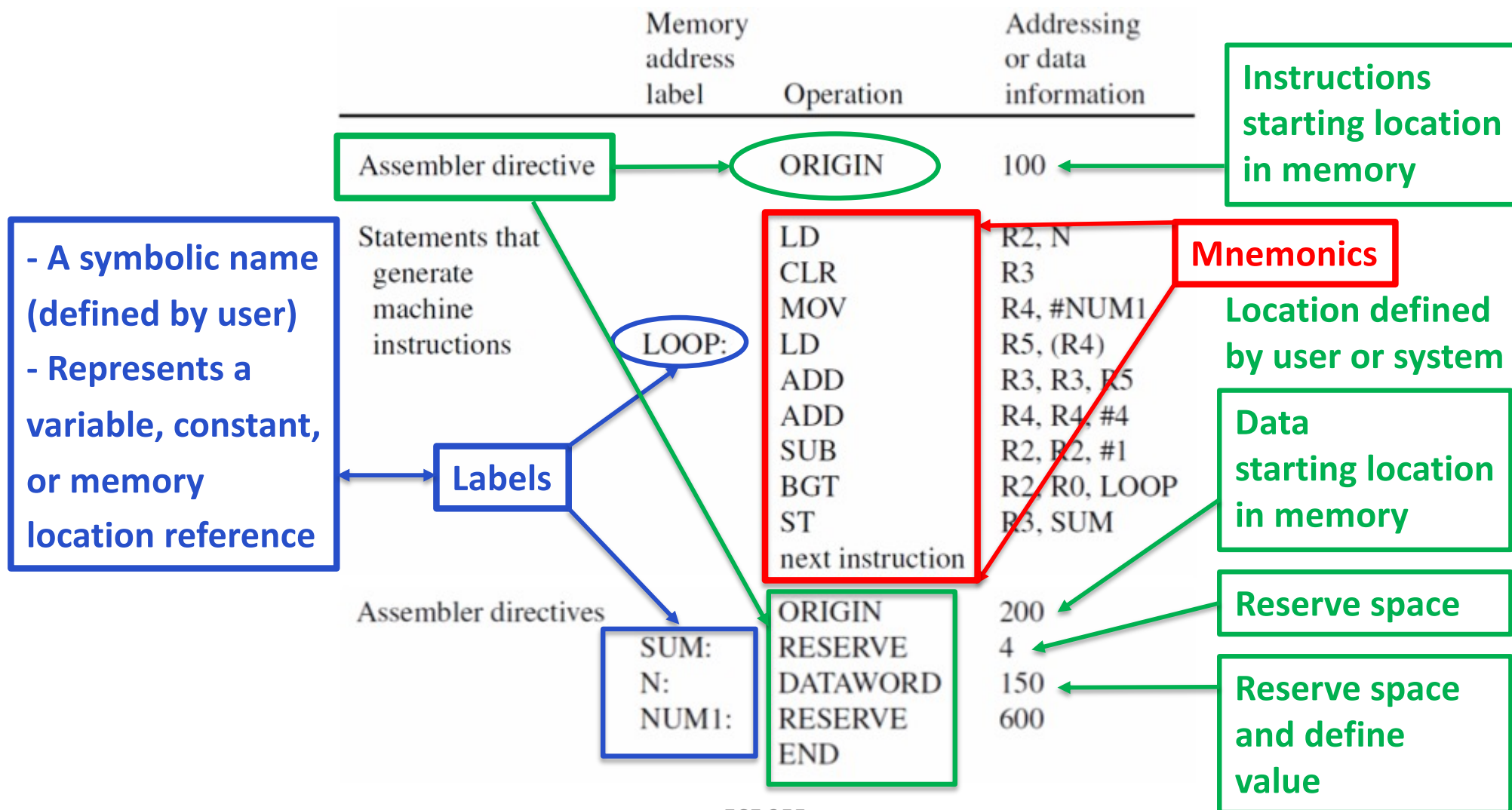
2.5.1 Labels and Directives



- Label:
 - A symbolic (often meaningful) name defined by the user
 - Represents a variable, or constant, or a memory location reference
- Assembler directives: instructions for the **assembler** (**not processor**)
 - ORIGIN (or ORI etc.): instruction/data starting location in memory
 - Defined by user or development system
 - RESERVE (or RES etc.) and DATAWORD (or Data etc.) define data storage
 - RESERVE spaces in memory (no value assigned)
 - DATAWORD is a value to be put in the memory space indicated



Assembly Program: Data Locations





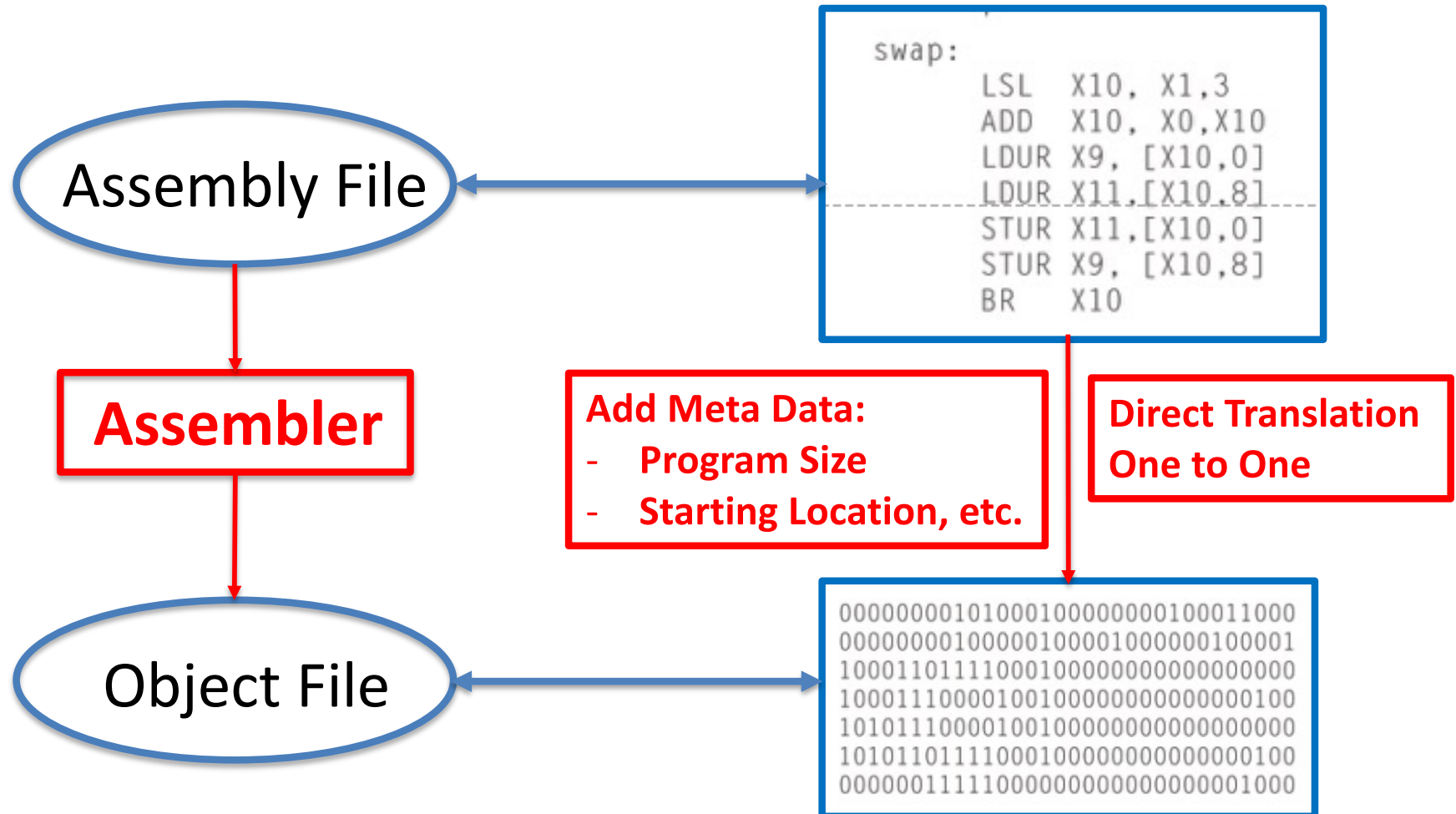
4.1 The Assembly Process



- Assembly Program:
 - Written as text (Mnemonics and Operands)
 - Created using a text editor, or
 - Created by compiling a high-level language HLL (C, C++, etc.) program
 - Example: using the gcc compiler
 - `gcc -S`: switch – S gives assembly file as output



What is an Assembler?





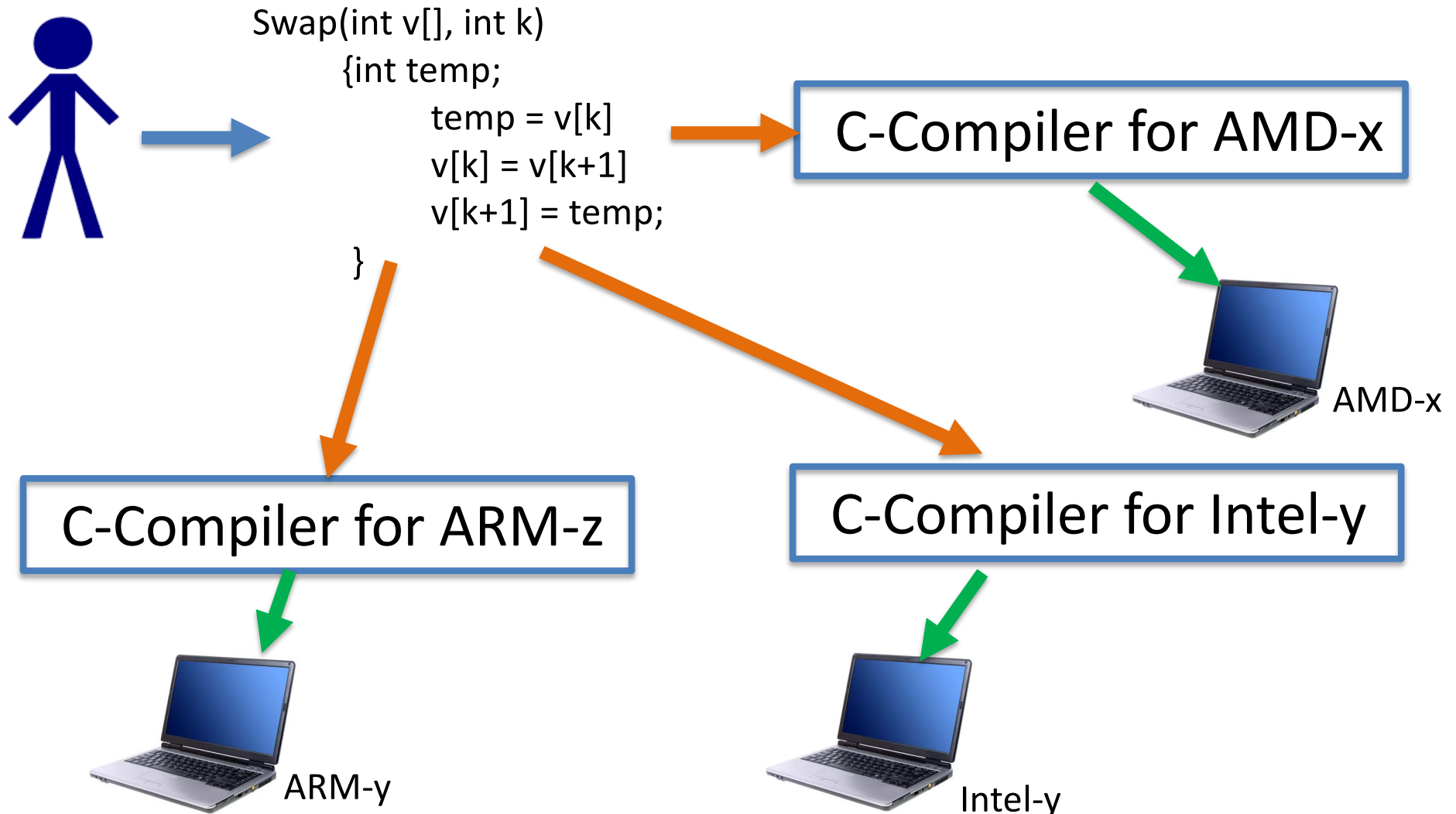
4.5 The Compiler



- Assembly programming requires machine-specific knowledge such as,
 - Bus system, memory size, ALU capability, etc.
- High-level language (HLL) programming reduces this need
 - Same HLL program
 - Just need a compiler for each machine/processor



Why High-Level Language?





Compiler vs Assembler



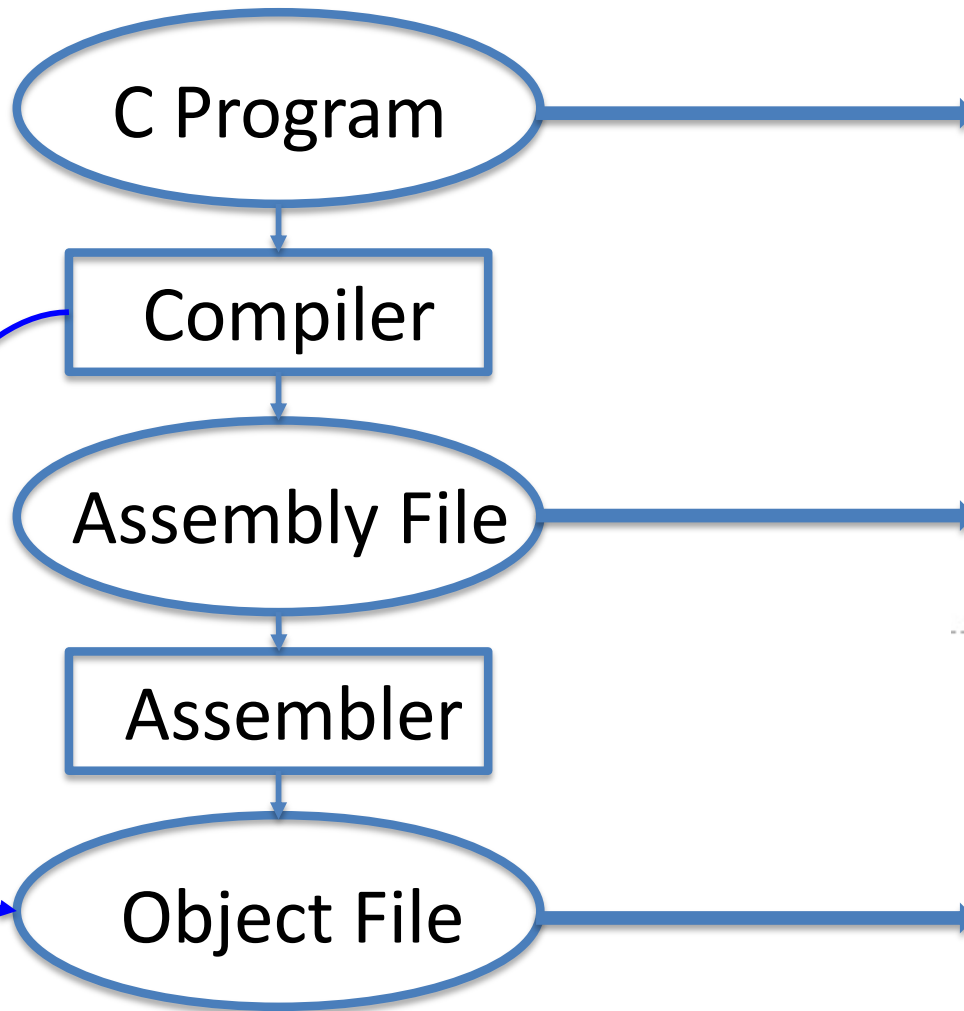
- Compiler
 - From HLL program to assembly, then to object code
 - Performs translation/optimization
- Assembler
 - From assembly to object
 - Converts assembly to object code



From C to Assembly to Object



Option:
Compile
to
Object
directly



```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

One-to-Many mapping

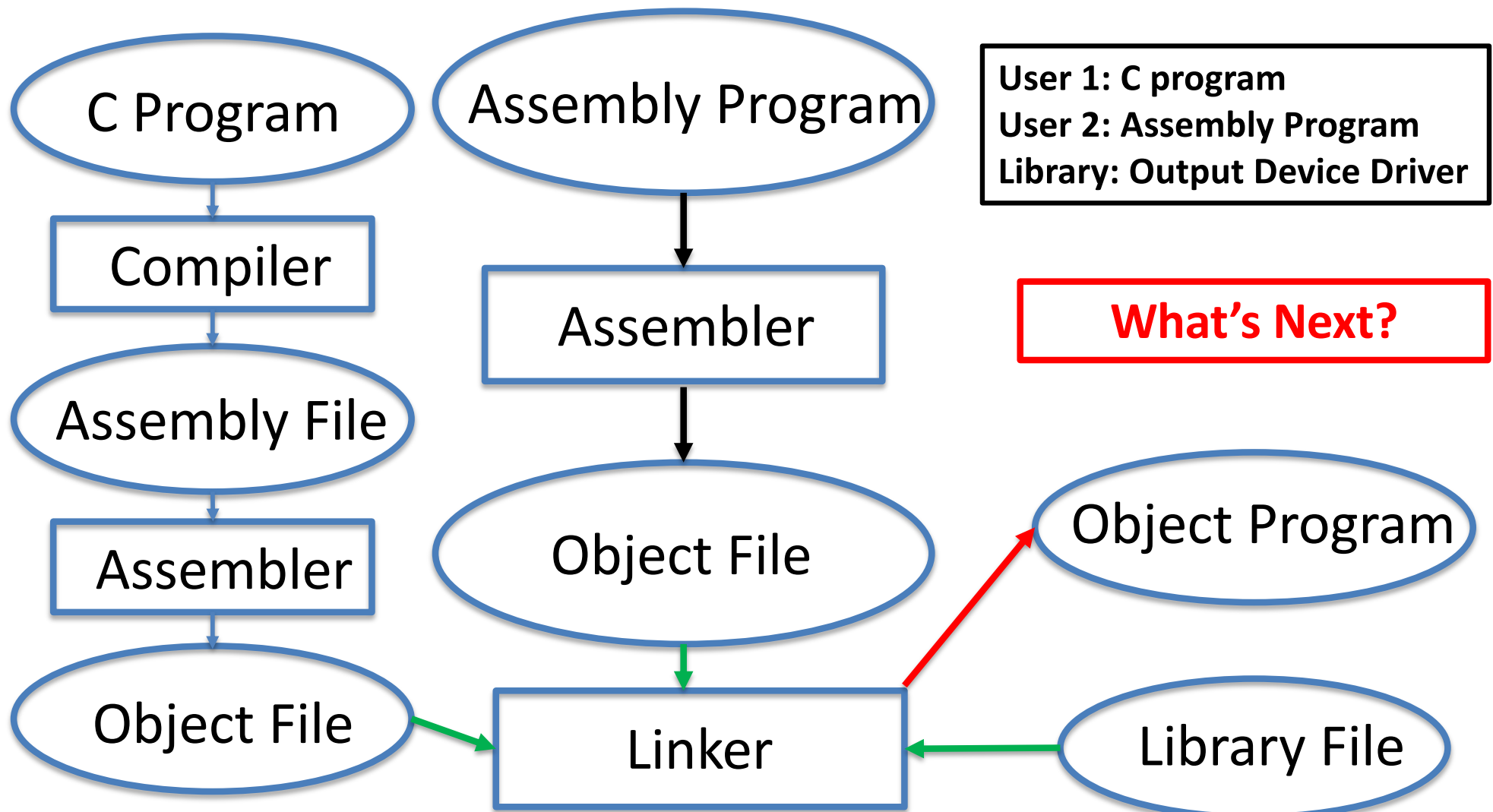
```
swap:
  LSL  X10, X1,3
  ADD  X10, X0,X10
  LDUR X9, [X10,0]
  LDUR X11,[X10,8]
  STUR X11,[X10,0]
  STUR X9, [X10,8]
  BR   X10
```

One-to-One mapping

```
000000001010001000000000100011000
000000000100000100001000000100001
100011011110001000000000000000000
100011100001001000000000000000100
101011100001001000000000000000000
101011011110001000000000000000100
00000011111000000000000000001000
```



What is a Linker?





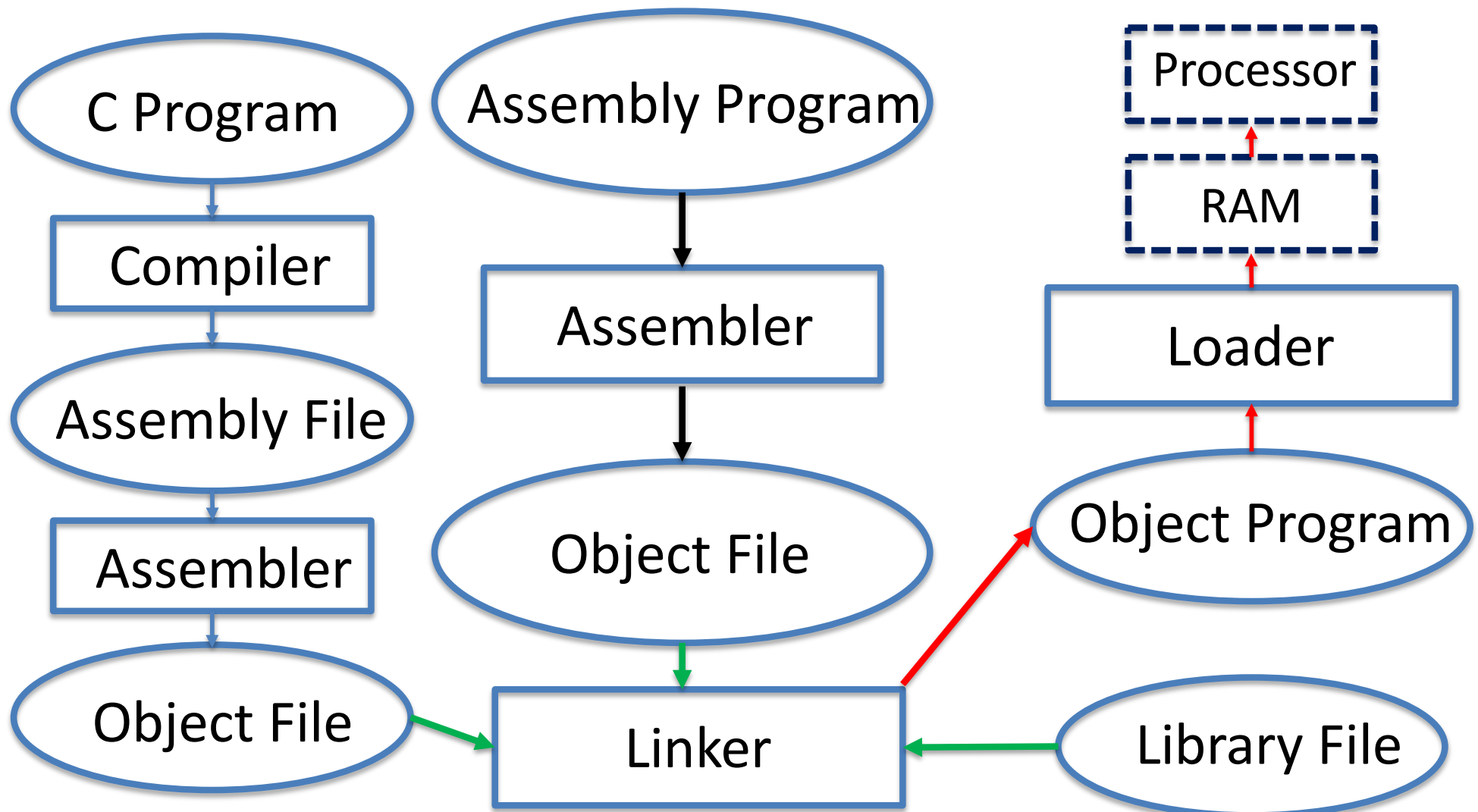
4.2 Loading/Executing Object Programs



- Loader : program/data/meta-data transfer
 1. Locates object program on disk (secondary memory)
 2. Identifies starting location + length of program (meta-data)
 3. Transfers object program from disk to RAM (primary memory)
- Processor starts execution at starting location (program in primary memory) until termination



Loader and Execution





4.6 The Debugger



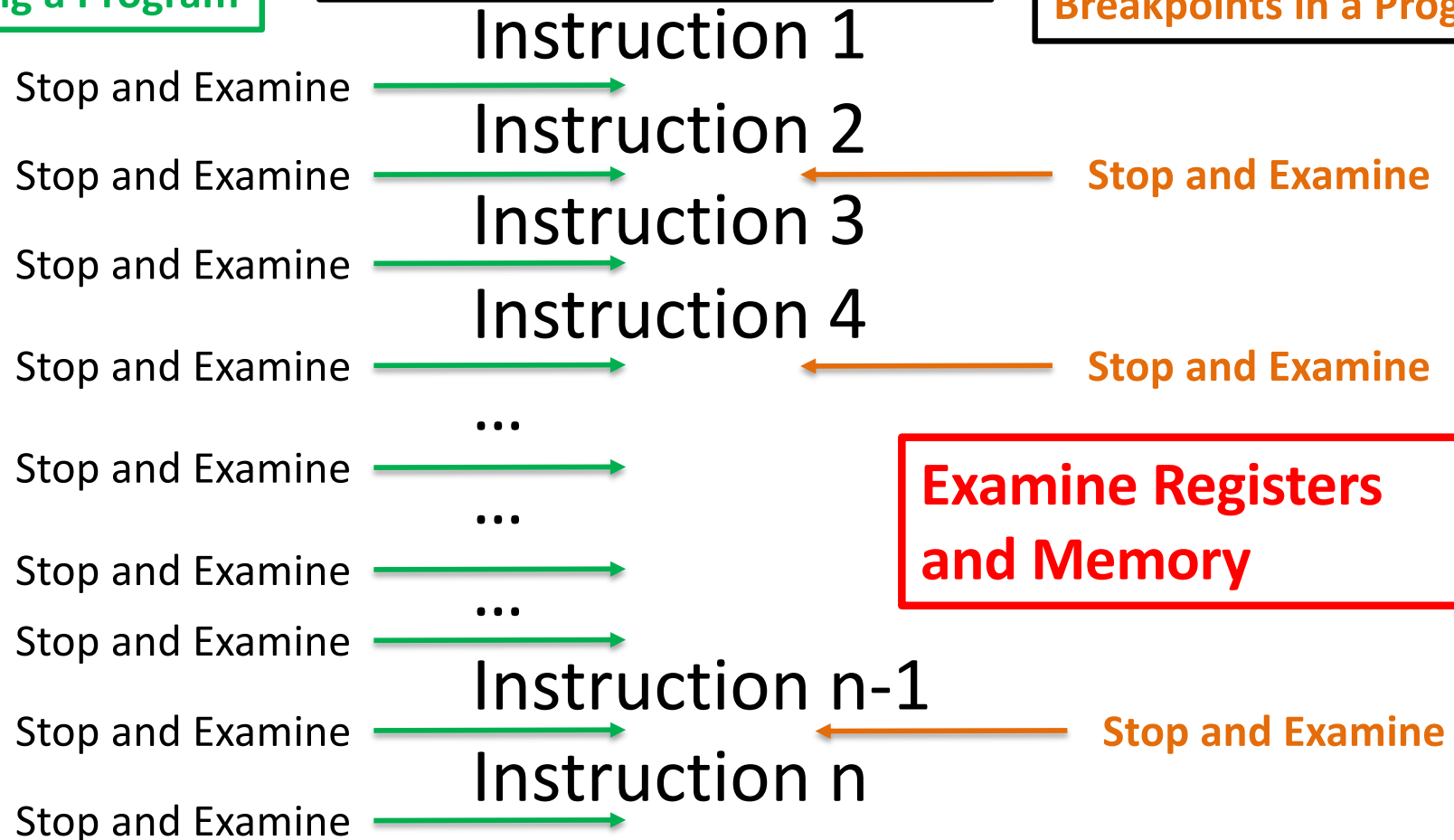
- Programming errors give incorrect results
- How to find the errors?
- [Debugger](#) to identify errors at execution time
 - Stop execution at points of interest
 - Display contents of registers or memory
 - Resume execution until next point of interest
- 1. [Trace](#) each instruction (stop and examine)
- 2. [Breakpoint](#) placed in program (stop and go)



Tracing a Program

Instruction Executed One by One

Breakpoints in a Program



Examine Registers and Memory



2.5.3 Number Notation



- Decimal numbers used as operand values:
`ADDI R2, R3, 93; R3 + 93 => R2`
 - Assembler translates to binary representation
- Programmer may also specify binary numbers:
`ADDI R2, R3, %01011101 or $01011101`
- Hexadecimal specification is an alternative:
`ADDI R2, R3, 0x5D`
- Note that $93 = 1011101_2 = 5D_{16}$



Tasks of the Assembler



- Assembler: convert assembly program (text) to object code (0s and 1s) in multiple passes
 1. Preprocesses text (Eliminate comments, space, etc.)
 2. Identifies labels
 3. Allocates space in memory for data
 4. Generates symbol table for label and address mapping
 5. Maps mnemonics to OP codes
 6. Converts operands (data information) into binary



Tasks of the Assembler 1



	Memory address label	Operation	Addressing or data information
Assembler directive		ORIGIN	100
Statements that generate machine instructions	LOOP:	LD	R2, N
		CLR	R3
		MOV	R4, #NUM1
		LD	R5, (R4)
		ADD	R3, R3, R5
		ADD	R4, R4, #4
		SUB	R2, R2, #1
		BGT	R2, R0, LOOP
		ST	R3, SUM
		next instruction	
Assembler directives		ORIGIN	200
	SUM:	RESERVE	4
	N:	DATAWORD	150
	NUM1:	RESERVE	600
		END	

1. Preprocesses text (Eliminate comments, space, etc.)



Tasks of the Assembler 2



	Memory address label	Operation	Addressing or data information
Assembler directive		ORIGIN	100
Statements that generate machine instructions	LOOP:	LD	R2, N
		CLR	R3
		MOV	R4, #NUM1
		LD	R5, (R4)
		ADD	R3, R3, R5
		ADD	R4, R4, #4
		SUB	R2, R2, #1
		BGT	R2, R0, LOOP
		ST	R3, SUM
		next instruction	
Assembler directives		ORIGIN	200
	SUM:	RESERVE	4
	N:	DATAWORD	150
	NUM1:	RESERVE	600
		END	

1. Preprocesses text (Eliminate comments, space, etc.)

2. Identifies labels:

- LOOP
- SUM
- N
- NUM1



Tasks of the Assembler 3



	Memory address label	Operation	Addressing or data information
Assembler directive		ORIGIN	100
Statements that generate machine instructions	LOOP:	LD	R2, N
		CLR	R3
		MOV	R4, #NUM1
		LD	R5, (R4)
		ADD	R3, R3, R5
		ADD	R4, R4, #4
		SUB	R2, R2, #1
		BGT	R2, R0, LOOP
		ST	R3, SUM
		next instruction	
Assembler directives		ORIGIN	200
	SUM:	RESERVE	4
	N:	DATAWORD	150
	NUM1:	RESERVE	600
		END	

1. Preprocesses text (Eliminate comments, space, etc.)

2. Identifies labels:

- LOOP
- SUM
- N
- NUM1

3. Allocates space in memory for data

- SUM = M[200..203]
- N = 150 ➔ M[204..207]
- NUM1 = M[208..804]



Tasks of the Assembler 4



	Memory address label	Operation	Addressing or data information
Assembler directive		ORIGIN	100
Statements that generate machine instructions		LD	R2, N
		CLR	R3
		MOV	R4, #NUM1
	LOOP:	LD	R5, (R4)
		ADD	R3, R3, R5
		ADD	R4, R4, #4
		SUB	R2, R2, #1
		BGT	R2, R0, LOOP
		ST	R3, SUM
		next instruction	
Assembler directives		ORIGIN	200
	SUM:	RESERVE	4
	N:	DATAWORD	150
	NUM1:	RESERVE	600
		END	

4. Generates symbol table for label and address mapping

- LOOP = 112
- SUM = 200
- N = 204
- NUM1 = 208



Tasks of the Assembler 5



	Memory address label	Operation	Addressing or data information
Assembler directive		ORIGIN	100
Statements that generate machine instructions	LOOP:	LD	R2, N
		CLR	R3
		MOV	R4, #NUM1
		LD	R5, (R4)
		ADD	R3, R3, R5
		ADD	R4, R4, #4
		SUB	R2, R2, #1
		BGT	R2, R0, LOOP
		ST	R3, SUM
		next instruction	
Assembler directives		ORIGIN	200
	SUM:	RESERVE	4
	N:	DATAWORD	150
	NUM1:	RESERVE	600
		END	

4. Generates symbol table for label and address mapping

- LOOP = 112
- SUM = 200
- N = 204
- NUM1 = 208

5. Maps mnemonics to OP codes

- LD → op(LD)
- CLR → op(CLR)
- ST → op(ST)
- ...



Tasks of the Assembler 6



	Memory address label	Operation	Addressing or data information
Assembler directive		ORIGIN	100
Statements that generate machine instructions	LOOP:	LD	R2, N
		CLR	R3
		MOV	R4, #NUM1
		LD	R5, (R4)
		ADD	R3, R3, R5
		ADD	R4, R4, #4
		SUB	R2, R2, #1
		BGT	R2, R0, LOOP
		ST	R3, SUM
		next instruction	
Assembler directives		ORIGIN	200
	SUM:	RESERVE	4
	N:	DATAWORD	150
	NUM1:	RESERVE	600
		END	

4. Generates symbol table for label and address mapping

- LOOP = 112
- SUM = 200
- N = 204
- NUM1 = 208

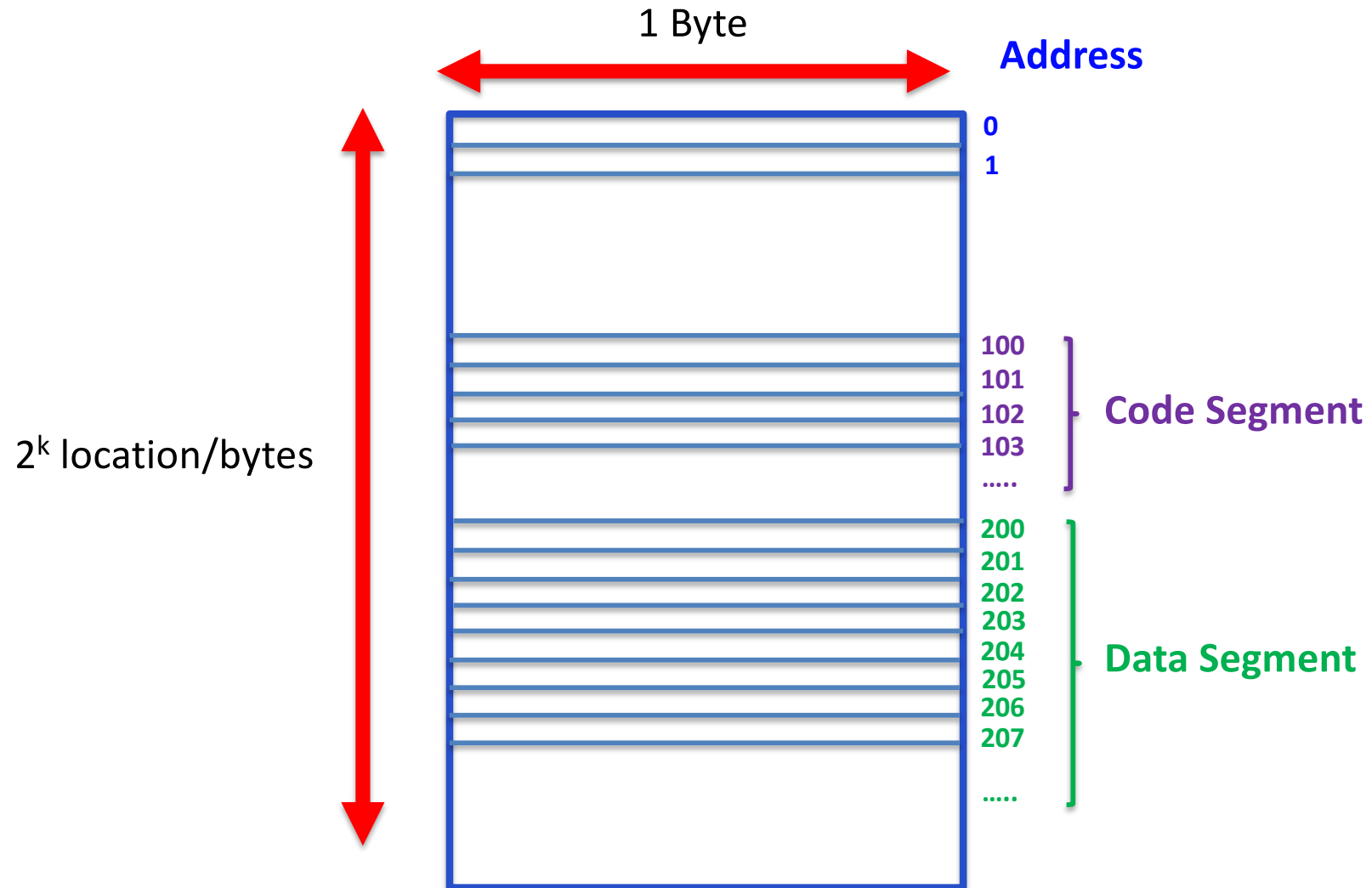
5. Maps mnemonics to OP codes

- LD → op(LD)
- CLR → op(CLR)
- ST → op(ST)
- ...

6. Convert operands into binary:
N, #NUM1, #4, #1, LOOP, SUM



Memory Locations and Contents





Tracing: Sample Assembly Program



	Memory address label	Operation	Addressing or data information
Assembler directive		ORIGIN	100
Statements that generate machine instructions		100 LD	R2, N $R2 \leftarrow M[204]$
		104 CLR	R3 $R3 \leftarrow 0$
		108 MOV	R4, #NUM1 $R4 \leftarrow 208$
	LOOP:	112 LD	R5, (R4) $R5 \leftarrow M[208]$
		116 ADD	R3, R3, R5 $R3 \leftarrow R3 + R5$
		120 ADD	R4, R4, #4 $R4 \leftarrow R4 + 4$
		124 SUB	R2, R2, #1 $R2 \leftarrow R2 - 1$
		128 BGT	R2, R0, LOOP If $R2 > R0$, PC=112
		132 ST	R3, SUM $R3 \rightarrow M[200]$
		next instruction	
Assembler directives		ORIGIN	200
	SUM:	200 RESERVE	4
	N:	204 DATAWORD	150
	NUM1	208 RESERVE	600
		212 END	
		xxx	
		
		804	

Given the following, find out what this program does:

Many Processors have R0 set to 0



Fig. 2.13 A Sample Assembly Program



	Memory address label	Operation	Addressing or data information
Assembler directive		ORIGIN	100
Statements that generate machine instructions		100 LD	R2, N $R2 \leftarrow M[204]$
		104 CLR	R3 $R3 \leftarrow 0$
		108 MOV	R4, #NUM1 $R4 \leftarrow 208$
	LOOP:	112 LD	R5, (R4) $R5 \leftarrow M[208]$
		116 ADD	R3, R3, R5 $R3 \leftarrow R3 + R5$
		120 ADD	R4, R4, #4 $R4 \leftarrow R4 + 4$
		124 SUB	R2, R2, #1 $R2 \leftarrow R2 - 1$
		128 BGT	R2, R0, LOOP If $R2 > R0$, $PC=112$
		132 ST	R3, SUM $R3 \rightarrow M[200]$
		next instruction	
Assembler directives		ORIGIN	200
	SUM:	200 RESERVE	4
	N:	204 DATAWORD	150
	NUM1	208 RESERVE	600
		212 END	
		xxx	
		
		804	

R2	R3	R4	R5



show the values of each register after the execution of an instruction



Address	R2	R3	R4	R5
100	150	X	X	X
104				
108				
112				
116				
120				
124				
128				
132				

After the execution of the first instruction, we have (X means don't know or don't care)



Compiler Output



- https://docs.oracle.com/cd/E19957-01/806-3567/cc_options.html
 - CC with option/flag -S to produce an assembly file
- https://www.delorie.com/djgpp/v2faq/faq8_20.html
 - GCC to generate assembly code
- <https://godbolt.org/>
 - Square of an integer: C to assembly (M68K, ARM-32, 13.2)
- <https://www.codeconvert.ai/c-to-assembly-converter>
 - C to assembly converter