

INTRODUCTION TO RISC-V

# RISC-V入门教程

ISA | 汇编指令 | 系统编程 | 组成原理 | 嵌入式应用

汇编语言

主讲 邢建国



中国开放指令生态 (RISC-V) 联盟 | 浙江中心  
China RICS-V Alliance | Zhejiang Center



浙江图灵算力研究院  
ZHEJIANG TURING INSTITUTE





## 汇编语言



- 汇编程序语法
- 注释
- 标号
- 指令
- 常用编译命令 (directive)

## ■ 汇编程序语法

汇编程序被编码为纯文本文件，包含四个主要元素：

Comment: 注释

Label: 标号

Assembly Instruction: 汇编指令

Assembly directives: 汇编命令

## ■ 汇编程序语法

注释：注释是通常用于记录代码信息的文本注释，然而，它们对代码生成没有影响，汇编器会丢弃它们；

注释以#开始，一直到行的结束

标号：标号是表示程序位置的“标记”。它们通常由以“：“结尾的名称定义，并且可以插入到汇编程序中以“标记”程序位置，可以被汇编指令引用；

汇编指令：汇编指令是由汇编程序转换为机器指令的指令。

通常被编码为一个字符串，其中包含一个助记词和一系列参数，称为操作数。

如：“addi a0、a1、1”包括助记符addi和三个操作数：a0、a1和常数1：

汇编指令：用于协调汇编过程的命令。它们由汇编器解释。

汇编指令通常被编码为包含指令名称的字符串，指令名称有一个点（‘.’）前缀及其参数。

如”.word 10”指令指示汇编器将一个32位值（10）组装到程序中。

## ■ 汇编程序语法

PROGRAM → LINES

LINES → LINE [‘\n’ LINES]

LINE → [<label>] [<instruction>] |  
[<label>] [<directive>]

## ■ 汇编程序语法

PROGRAM → LINES

LINES → LINE [‘\n’ LINES]

LINE → [<label>] [<instruction>] |  
[<label>] [<directive>]

- 一行可以是空行；
- 一行可能包含单个标号；
- 一行可能包含一个标号，后跟一个汇编指令；
- 一行可能包含一个汇编指令；
- 一行可能包含一个标签，后跟一个汇编命令；
- 一行可能包含一个汇编命令；

## ■ 汇编程序例子

```
1 x:  
2  
3 sum: addi a0, a1, 1  
4     ret  
5 .section .data  
6 y:    .word 10
```

✓

```
1 x: z:  
2 addi a0, a1, 1 sum:  
3 li a0, 2      li a1, 1  
4 .word 10 .word 20  
5 .word 10 y:  
6 addi a0, a1, 1 .word 12  
7 .sdfoiywer 1
```

✗

```
1 addi  
2 a0, a1, 1
```

✗

## ■ 注释

```
1 x: .word 10      # This is a comment
2 foo:              # My special function
3 addi a0, a1, 1   # Adds one to a1 and store on a0 #
4 # This is # another # comment ## #
```



```
1 x: .word 10
2 foo:
3 addi a0, a1, 1
```

```
1 sum1:
2 /* This
3  is
4   a
5     multi-line
6       comment.
7 */
8   addi a0, a1, 1
9   ret
```



```
1 sum1: addi a0, a1, 1
2         ret
```

GNU汇编器支持C语言风格注释

## ■ 汇编指令

汇编指令是由汇编器转换为机器指令的指令。它们通常被编码为包含助记词和参数序列的字符串，称为操作数。

如汇编指令：“add x10, x11, x12”，使用17个字节编码为纯文本，由汇编器转换为其相应的4字节机器指令：0x00c58533

## ■ 汇编指令

汇编指令是由汇编器转换为机器指令的指令。它们通常被编码为包含助记词和参数序列的字符串，称为操作数。

如汇编指令：“add x10, x11, x12”，使用17个字节编码为纯文本，由汇编器转换为其相应的4字节机器指令：0x00c58533

伪指令也是一种汇编指令，它在ISA上没有对应的机器指令，但可以由汇编器自动翻译成一个或多个替代机器指令以实现相同的效果。

如，指令nop是一个RV32I伪指令，由汇编器转换为机器指令“addi x0, x0, 0”。

伪指令mv t1, t2，由汇编器转换为机器指令“addi t1,t2,0”

## ■ 汇编指令

每条汇编指令可以包含1-3个操作数，操作数可以是：

**寄存器**：RV32I ISA寄存器编号从0到31，并命名为x0、x1、…x31。RV32I的寄存器也可以通过其别名来标识，例如zero、sp、ra、gp、a0、t1等。

- 立即数**：立即数是一个常量，作为比特序列直接编码到机器指令中。
- 符号名称**：符号表里的符号，在汇编和链接过程中被它们各自的值替换。

<b>基础整数指令集: RV32I 和 RV64I</b>				<b>RV 特权指令</b>				
<b>类别</b>	<b>名称</b>	<b>类型</b>	<b>基础RV32I</b>	<b>+RV64I</b>	<b>类别</b>	<b>名称</b>	<b>类型</b>	<b>RV 助记符</b>
<b>移位</b>	逻辑左移	R	SLL rd,rs1,rs2	SLLW rd,rs1,rs2	<b>自陷</b>	M模式异常返回	R	MRET
	逻辑左移立即数	I	SLLI rd,rs1,shamt	SLLIW rd,rs1,shamt		S模式异常返回	R	SRET
	逻辑右移	R	SRL rd,rs1,rs2	SRLW rd,rs1,rs2		<b>中断</b>		WFI
	逻辑右移立即数	I	SRLI rd,rs1,shamt	SRLIW rd,rs1,shamt		<b>MMU</b>		SFENCE.VMA rs1,rs2
	算术右移	R	SRA rd,rs1,rs2	SRAW rd,rs1,rs2		<b>60条RV伪指令举例</b>		
	算术右移立即数	I	SRAI rd,rs1,shamt	SRAIW rd,rs1,shamt		等于0时分支(即 BEQ rs,x0,imm)	B	BEQZ rs,imm
<b>算术</b>	加	R	ADD rd,rs1,rs2	ADDDW rd,rs1,rs2		跳转(即 JAL x0,imm)	J	J imm
	加立即数	I	ADDI rd,rs1,imm	ADDIW rd,rs1,imm		传送(即 ADDI rd,rs,0)	R	MV rd,rs
	减	R	SUB rd,rs1,rs2	SUBW rd,rs1,rs2		返回(即 JALR x0,0(ra))	I	RET
	装入高位立即数	U	LUI rd,imm					
	PC加高位立即数	U	AUIPC rd,imm					
<b>逻辑</b>	异或	R	XOR rd,rs1,rs2					
	异或立即数	I	XORI rd,rs1,imm					
	或	R	OR rd,rs1,rs2					
	或立即数	I	ORI rd,rs1,imm					
	与	R	AND rd,rs1,rs2					
	与立即数	I	ANDI rd,rs1,imm					
<b>比较-置位</b>	小于则置位	R	SLT rd,rs1,rs2					
	小于立即数则置位	I	SLTI rd,rs1,imm					
	无符号小于则置位	R	SLTU rd,rs1,rs2					
	无符号小于立即数则置位	I	SLTIU rd,rs1,imm					
<b>分支</b>	相等时分支	B	BEQ rs1,rs2,imm					
	不等时分支	B	BNE rs1,rs2,imm					
	小于时分支	B	BLT rs1,rs2,imm					
	大于等于时分支	B	BGE rs1,rs2,imm					
	无符号小于时分支	B	BLTU rs1,rs2,imm					
	无符号大于等于时分支	B	BGEU rs1,rs2,imm					
<b>跳转并链接</b>	跳转并链接	J	JAL rd,imm					
	寄存器跳转并链接	I	JALR rd,imm(rs1)					
<b>同步</b>	同步线程	I	FENCE					
	同步指令和数据	I	FENCE.I					
<b>环境</b>	环境调用	I	ECALL					
	环境断点	I	EBREAK					
<b>控制状态寄存器(CSR)</b>								
			读后写	I	CSRRW rd,csr,rs1			
			读后置位	I	CSRRS rd,csr,rs1			
			读后清位	I	CSRRC rd,csr,rs1			
			读后写立即数	I	CSRRWI rd,csr,imm			
			读后置位立即数	I	CSRRSI rd,csr,imm			
			读后清位立即数	I	CSRRCI rd,csr,imm			
<b>取数</b>								
			取字节	I	LB rd,imm(rs1)			
			取半字	I	LH rd,imm(rs1)			
			取无符号字节	I	LBU rd,imm(rs1)			
			取无符号半字	I	LHU rd,imm(rs1)			
			取字	I	LW rd,imm(rs1)			
<b>存数</b>								
			存字节	S	SB rs2,imm(rs1)			
			存半字	S	SH rs2,imm(rs1)			
			存字	S	SW rs2,imm(rs1)			
<b>RV 特权指令</b>								
<b>类别</b>	<b>名称</b>	<b>类型</b>	<b>RV 助记符</b>					
<b>自陷</b>	M模式异常返回	R	MRET					
	S模式异常返回	R	SRET					
<b>中断</b>	等待中断	R	WFI					
	虚拟存储屏障	R	SFENCE.VMA rs1,rs2					

## ■ 立即数

在汇编语言中，立即数（常数）由字母数字字符序列表示。

---

```
1 li a0, 10      # loads value ten into register a0
2 li a1, 0xa     # loads value ten into register a1
3 li a2, 0b1010  # loads value ten into register a2
4 li a3, 012     # loads value ten into register a3
5 li a4, '0'      # loads value forty eight into register a4
6 li a5, 'a'      # loads value ninety seven into register a5
```

---

## ■ 立即数

---

```
1 li a0, 10      # loads value ten into register a0
2 li a1, 0xa     # loads value ten into register a1
3 li a2, 0b1010 # loads value ten into register a2
4 li a3, 012     # loads value ten into register a3
5 li a4, '0'      # loads value forty eight into register a4
6 li a5, 'a'      # loads value ninety seven into register a5
```

---

---

```
1 li a0, -12     # loads value minus twelve into register a0
2 li a1, -0xc     # loads value minus twelve into register a1
3 li a2, -0b1100  # loads value minus twelve into register a2
4 li a3, -014     # loads value minus twelve into register a3
5 li a4, -'0'      # loads value minus forty eight into register a4
6 li a5, -'a'      # loads value minus ninety seven into register a5
```

---

## ■ 符号 Symbol

符号是与数值相关联的“名称”，“符号表”是将每个程序符号映射到其值的数据结构。

符号名称由一系列字母数字字符和下划线'\_'组成。第一个字符不是数字。

有效符号名称示例: x、var1、z12345、\_x、\_、\_1、\_a12b

无效符号名称示例: 1、1varz、@12345、x-y、-var、a+b

## ■ 符号Symbol

---

```
1 .set max_temp, 100          # Set the max_temp limit
2
3 check_temp:                 # check_temp routine
4   li  t1, max_temp          # Loads the max_temp limit into t1
5   ble a0, t1, temp_ok       # If a0 <= max_temp, then ok
6   jal alarm                 # Else, invokes the alarm routine
7 temp_ok:
8   ret                      # Returns from the routine
```

---

## ■ 标号Label

标签是代表程序位置的“标记”。它们可以由指令和汇编指令引用，并在汇编和链接过程中转换为地址

GNU汇编器通常接受两种标号：符号label和数字label。符号label作为符号存储在符号表中，通常用于标识全局变量和例程。它们由标识符后跟冒号（:）定义。标识符遵循符号名称的相同语法。

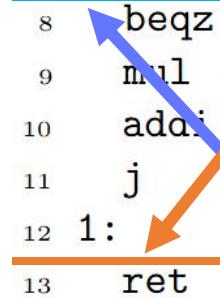
```
1 age: .word 42
2
3 get_age:
4   la t1, age
5   lw a0, (t1)
6   ret
```

## ■ 标号Label

数字label由一个十进制数字后跟一个冒号 (:) 定义。可用于本地引用，不包含在可执行文件的符号表中。它们可以在同一个汇编程序中重复定义。

对数字label的引用包含一个后缀b或f，它指示引用是位于引用之前 ('b') 还是之后 ('f') 的数字label。

```
1 # Pow function -- computes a^b
2 # Inputs: a0=a, a1=b
3 # Output: a0=a^b
4 pow:
5     mv    a2, a0      # Saves a0 in a2
6     li    a0, 1       # Sets a0 to 1
7     1:
8     beqz a1, 1f      # If a1 = 0 then done
9     mul   a0, a0, a2 # Else, multiply
10    addi  a1, a1, -1 # Decrements the counter
11    j     1b          # Repeat
12    1:
13    ret
```



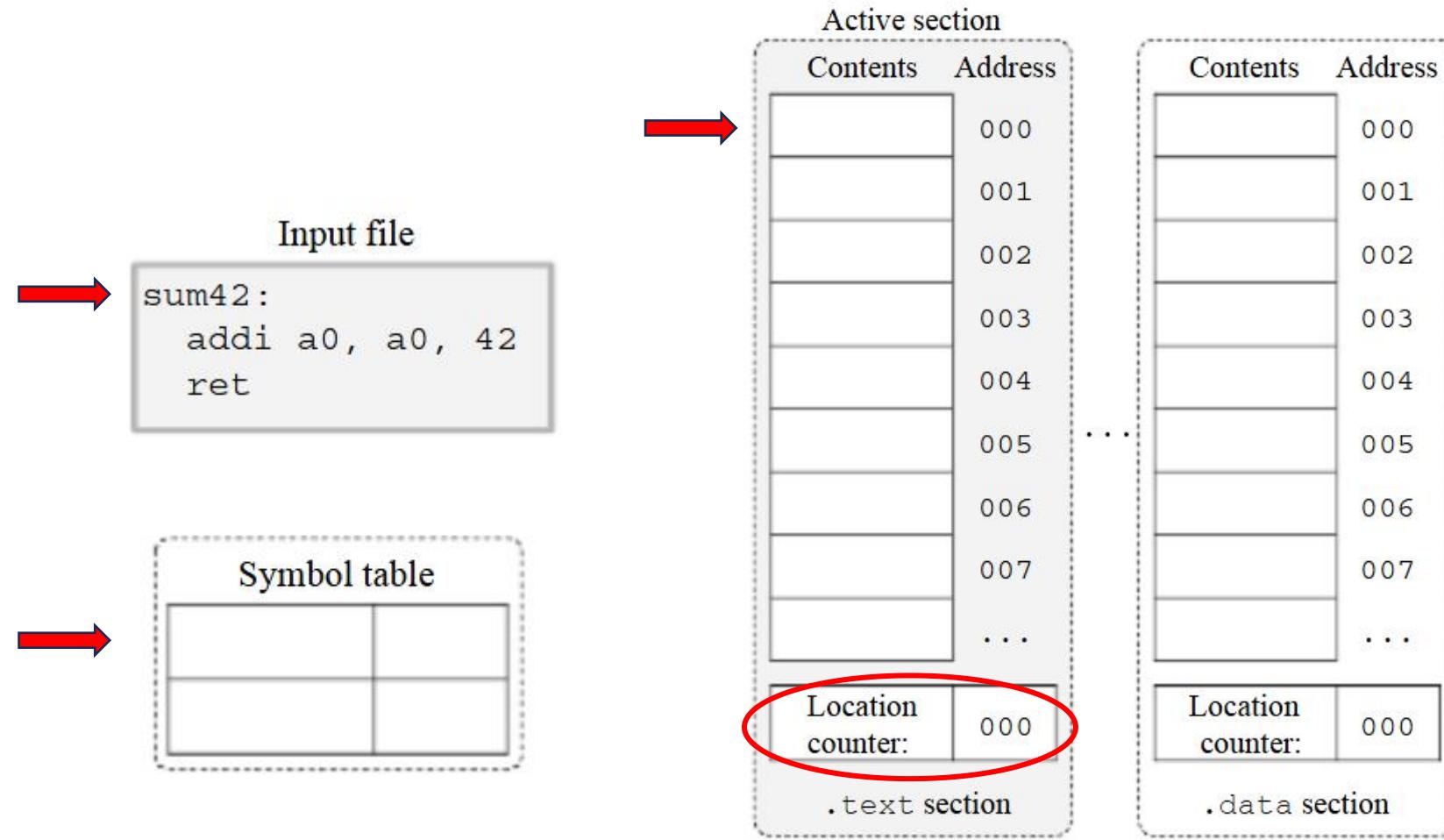
## ■ 位置计数器和汇编过程

位置计数器是一个内部计数器，它在程序汇编时跟踪地址。它记录下一个可用内存位置的地址。

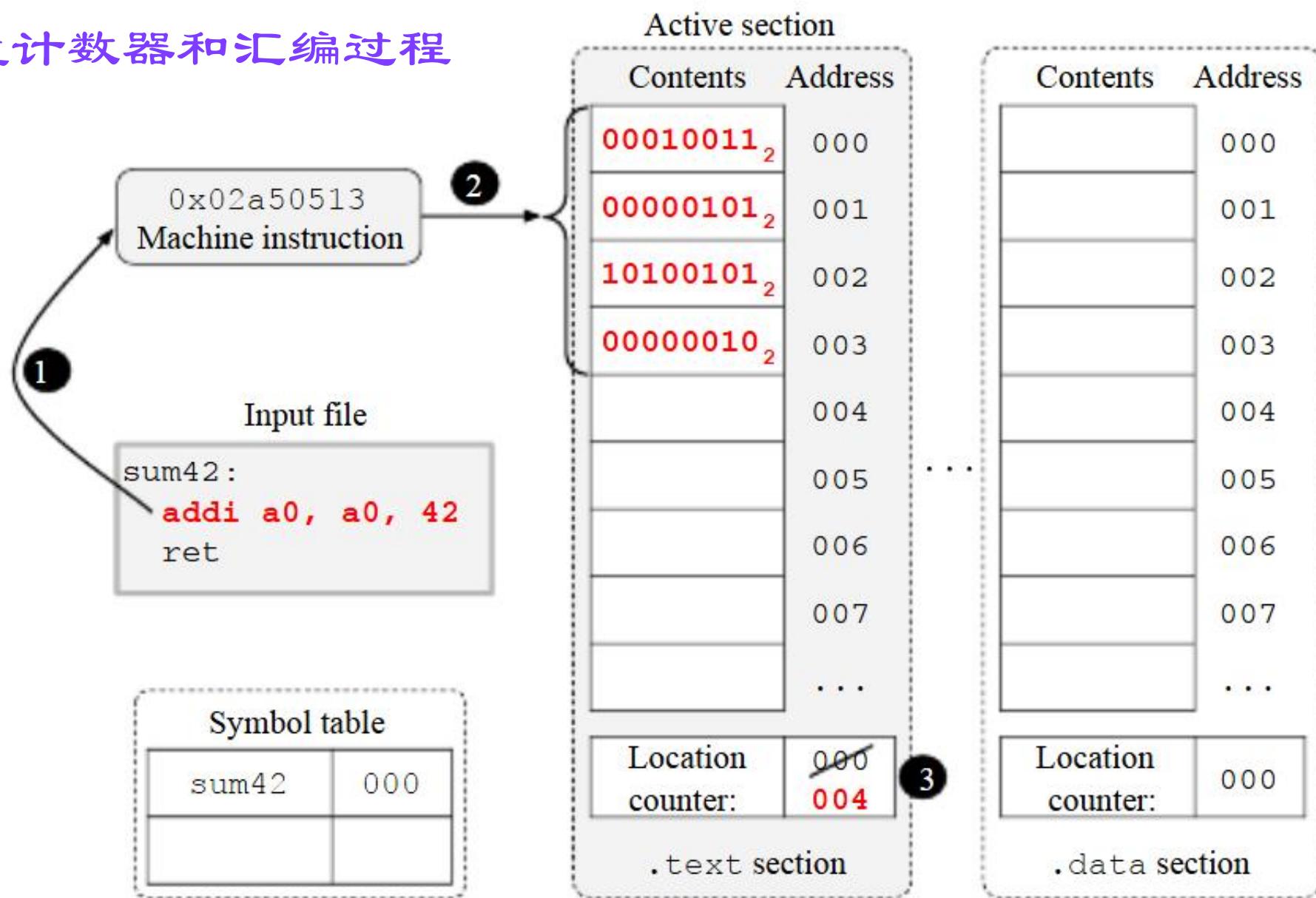
每个section都有自己的位置计数器，活动位置计数器是活动section的位置计数器。

## ■ 位置计数器和汇编过程

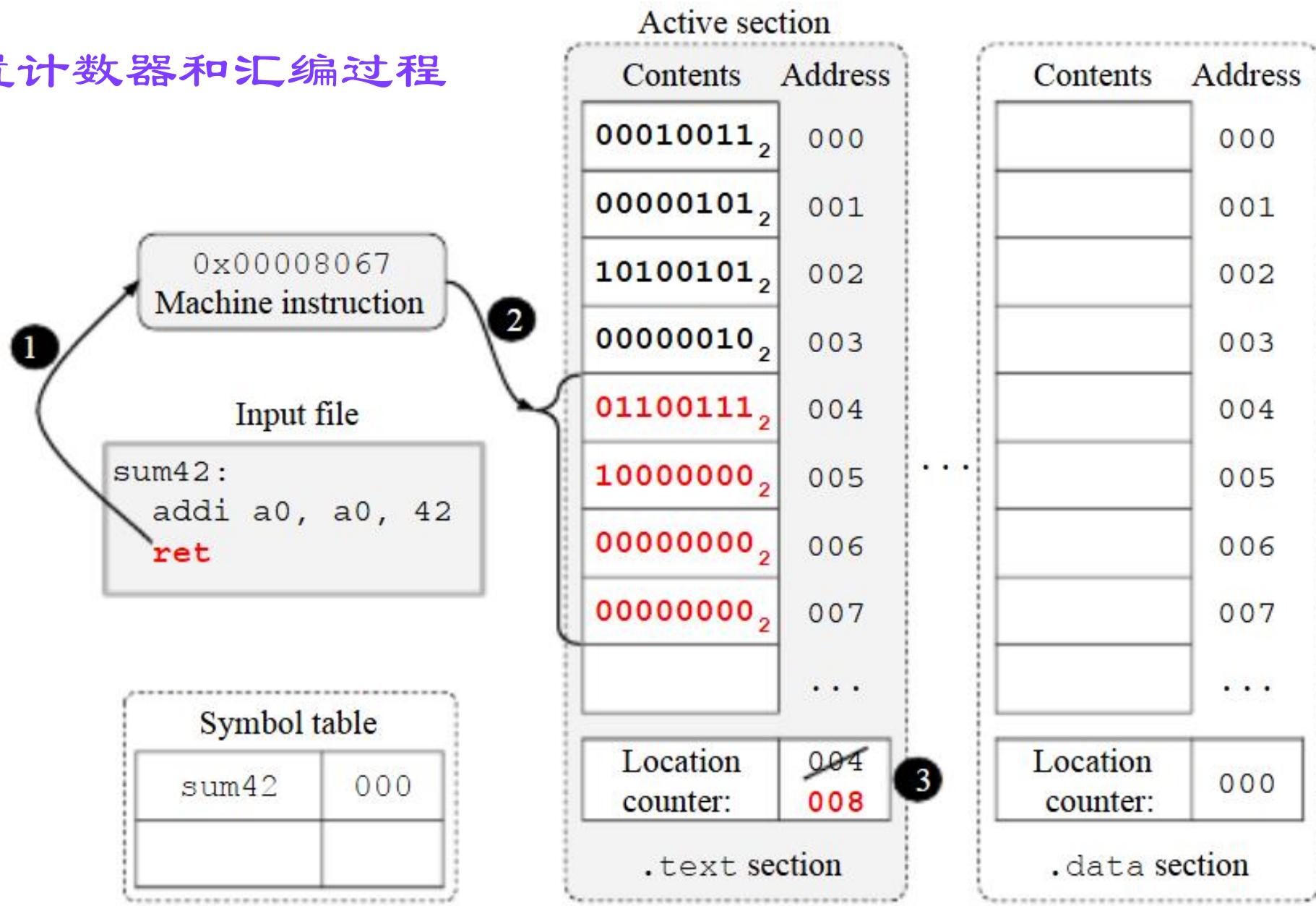
为了讨论位置计数器在整个汇编过程中是如何使用和更新的，我们将逐步汇编以下汇编程序



## ■ 位置计数器和汇编过程



## ■ 位置计数器和汇编过程



## ■ 汇编命令 (directive)

汇编命令用于控制汇编器的行为。例如，“.section .data”指示汇编器将section data转换为活动section, '.word 10'指示汇编器将一个值为10的32位字添加到活动section中。

汇编指令通常编码为包含指令名称及其参数的字符串。在GNU汇编器上，指令名称包含一个点（‘.’）前缀。

## ■ 添加值定义的命令

Directive	Arguments	Description
.byte	expression [, expression]*	Emit one or more 8-bit comma separated words
.half	expression [, expression]*	Emit one or more 16-bit comma separated words
.word	expression [, expression]*	Emit one or more 32-bit comma separated words
.dword	expression [, expression]*	Emit one or more 64-bit comma separated words
.string	string	Emit NULL terminated string
.asciz	string	Emit NULL terminated string (alias for .string)
.ascii	string	Emit string without NULL character

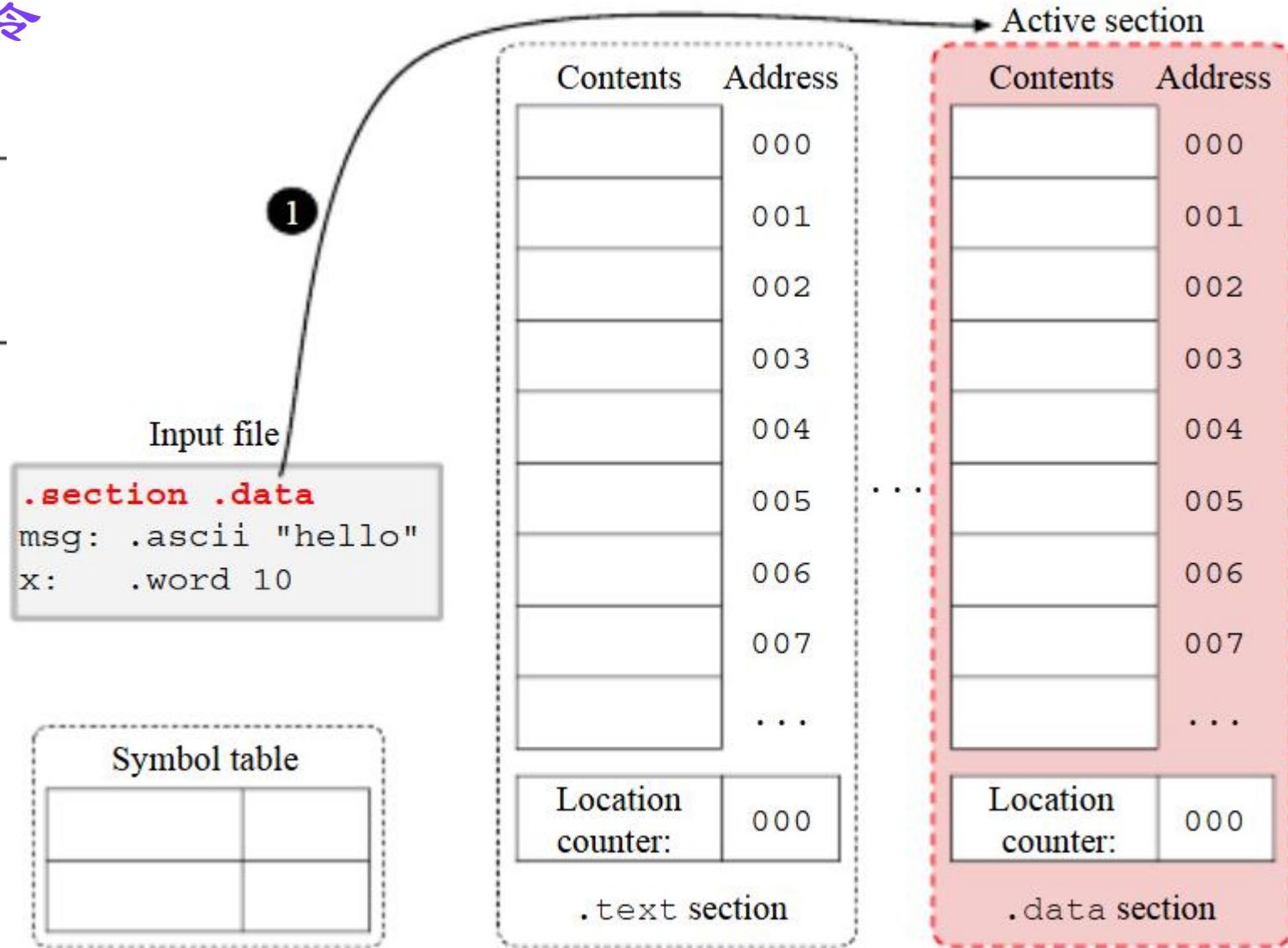
## ■ 添加值定义的命令

```
1           x: .byte 10, 12, 'A', 5+5  
2 y: .word x  
3 z: .word y+4  
4 i: .word 0  
5 j: .word 1
```

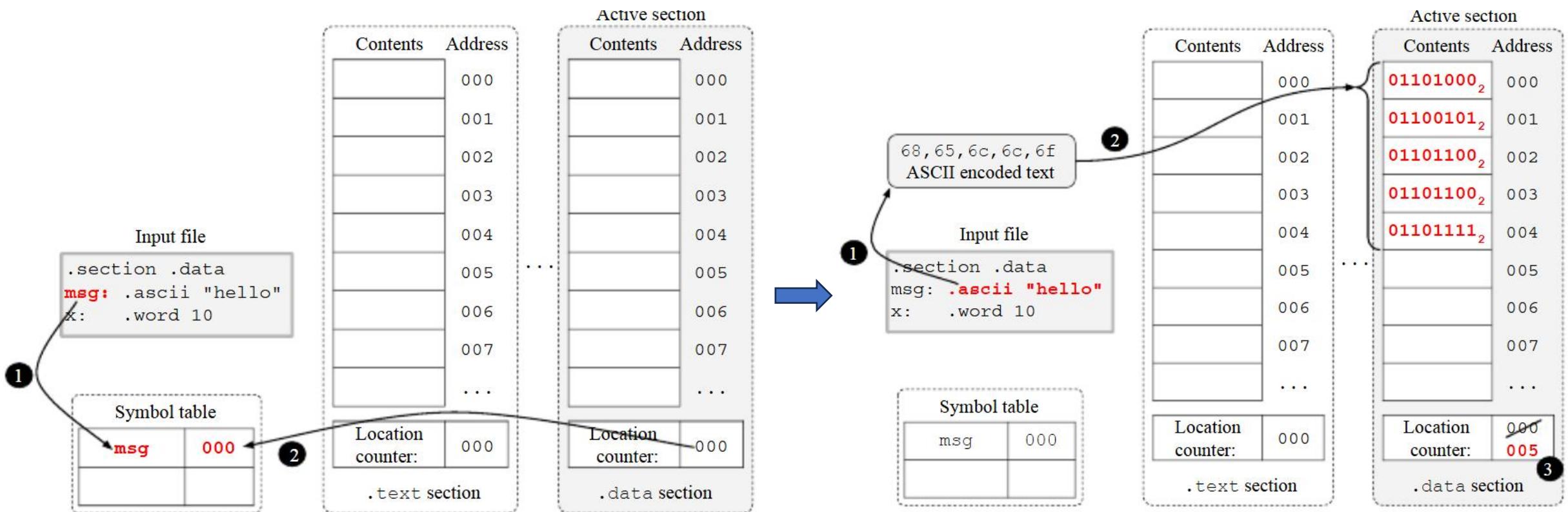
---

## ■ 添加值定义的命令

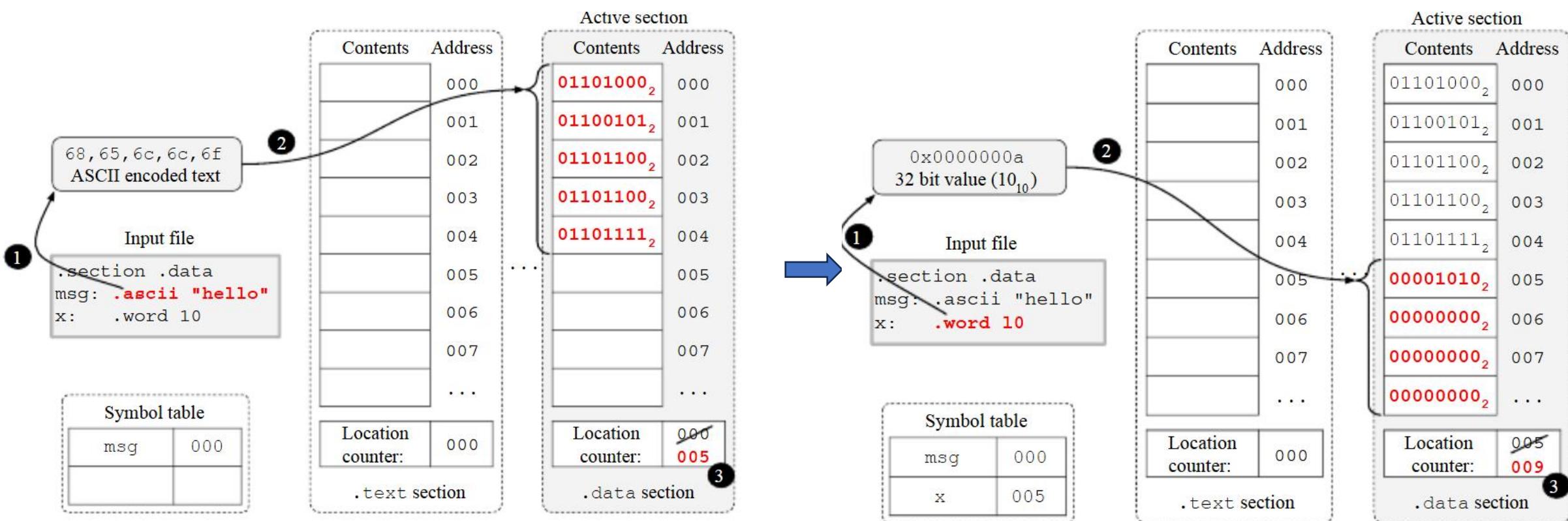
```
1 .section .data  
2 msg: .ascii "hello"  
3 x: .word 10
```



## ■ 添加值定义的命令



## ■ 添加值定义的命令



## ■ .section命令

汇编程序、目标文件和可执行文件按section来组织的。

为了指示汇编器将汇编好的信息添加到其他section中，程序员（或编译器）可以使用命令.section secname, 将活动section改为secname。在将此命令之后，汇编器将生成的所有信息都应放置在该section中。

代码通常放在.text, 只读数据放在.rodata, 全局变量必须放置在.data, 未初始化的全局变量应该放在.bss

## ■ .section命令

代码通常放在.text, 只读数据放在.rodata, , 全局变量必须放置在.data, 未初始化的全局变量应该放在.bss

```
1 .section .text
2 set_x:
3   la t1, x
4   sw a0, (t1)
5   ret
6 get_msg:
7   la a0, msg
8   ret
9 .section .data
10 x: .word 10
11 .section .rodata
12 msg: .string "Assembly rocks!"
```

**NOTE:** The RV32I GNU assembler also contains the “.text”, “.data”, and “.bss” directives, which are aliases to “.section .text”, “.section .data”, and the “.section .bss” directives, respectively.

## 在.bss中分配变量

未初始化的全局变量应该放在.bss

```
1 .section .bss
2 x: .word 10 X
3 .section .text
4 set_x:
5 la t1, x
6 sw a0, (t1)
7 ret
```



**NOTE:** Some systems initialize the memory words dedicated to the .bss section with zeros when loading the program into the main memory for execution. Nonetheless, the programmer should not assume variables on the .bss section will be initialized with zeros.

```
1 .section .bss
2 x: .skip 4 ✓
3 V: .skip 80
4 y: .skip 4
```

---

```
1 $ riscv64-unknown-elf-as -march=rv32im data-on-bss.s -o data-on-bss.o
2 data-on-bss.s: Assembler messages:
3 data-on-bss.s:2: Error: attempt to store non-zero value in section '.bss'
```

---

## ■ .set、.equ命令

.set命令将符号添加到符号表中。它将表达式名、和表达式作为参数，将表达式计算为值，并将名称和值存储到符号表中。

.equ命令和.set命令类似

```
1 .set max_value, 42
2
3 truncates_value_to_max:
4     li    t1, max_value
5     ble   a0, t1, ok
6     mv    a0, t1
7 ok:
8     ret
```

## ■ .global命令

汇编器默认标号或使用.set或.equ创建的符号作为局部符号存储在符号表中。

.global命令用于将局部符号转换为全局符号。

---

```
1 .globl max_value
2 .globl start
3
4 .set max_value, 42
5
6 start:
7     li  a0, max_value
8     jal process_temp
9     ret
```

---

## ■ .align(对齐)命令

一些指令集体体系结构要求指令或多字节数据存储在给定数字的倍数地址上。例如，RV32I ISA要求指令存储在4的倍数地址上。

GNU汇编器不会自动验证RV32I指令是否分配给4的倍数地址。程序员（或编译器）负责保持RV32I指令与4字节边界对齐，地址为4的倍数。在下面的例子中，可以在将8位值添加到程序后立即将位置计数器前进三个单位（第4行）。

```
1 .text
2 foo:
3 j next
4 .byte 0xa
5 next:
6 ret
```

```
1 .text
2 foo:
3 j next
4 .byte 0xa
5 .skip 3    # Advancing the location counter by 3 units.
6           # This is a very poor way of keeping the
7           # location counter aligned to a 4 byte boundary.
8 next:
9 ret
```

注意：避免手工对齐

编译不会报错，但运行时会触发异常



## ■ .align(对齐)命令

.align N命令检查位置计数器是否是 $2^N$ 的倍数，如果是，则对程序没有影响，否则，它将位置计数器推进到 $2^N$ 倍数的下一个值。

RV32I ISA允许程序在未对齐的内存地址上加载和存储数据，但是，出于性能原因，RISC-V指令集手册建议将16位、32位和64位值分别存储在为2、4和8倍数的地址上。

---

```
1 .data
2 .align 1
3 i: .half 1 # 16-bit variable initialized with value 1
4 .align 2
5 x: .word 9 # 32-bit variable initialized with value 9
6 .align 3
7 y: .dword 11 # 64-bit variable initialized with value 11
8 .bss
9 .align 3
0 z: .skip 8 # 64-bit variable (uninitialized)
```

---

## 编译命令汇总

指示符	描述
<code>.text</code>	后续内容存放在代码节（机器代码）。
<code>.data</code>	后续内容存放在数据节（全局变量）。
<code>.bss</code>	后续内容存放在 bss 节（初始化为 0 的全局变量）。
<code>.section .foo</code>	后续内容存放在名为 <code>.foo</code> 的节。
<code>.align n</code>	后续数据按 $2^n$ 字节对齐。如 <code>.align 2</code> 指示后续数据按字对齐
<code>.balign n</code>	后续数据按 $n$ 字节对齐。如 <code>.balign 4</code> 指示后续数据按字对齐
<code>.globl sym</code>	声明 sym 为全局符号，可从其他文件引用。
<code>.string "str"</code>	将字符串 str 存放在内存，以空字符结尾。
<code>.byte b1,..., bn</code>	在内存中连续存放 n 个 8 位数据。
<code>.half w1,..., wn</code>	在内存中连续存放 n 个 16 位数据。
<code>.word w1,..., wn</code>	在内存中连续存放 n 个 32 位数据。
<code>.dword w1,..., wn</code>	在内存中连续存放 n 个 64 位数据。
<code>.float f1,..., fn</code>	在内存中连续存放 n 个单精度浮点数。
<code>.double d1,..., dn</code>	在内存中连续存放 n 个双精度浮点数。
<code>.option rvc</code>	压缩后续指令（见第 7 章）。
<code>.option norvc</code>	不压缩后续指令。
<code>.option relax</code>	允许链接器松弛后续指令。
<code>.option norelax</code>	禁止链接器松弛后续指令。
<code>.option pic</code>	后续指令为位置无关代码。
<code>.option nopic</code>	后续指令为位置相关代码。
<code>.option push</code>	将当前所有 <code>.option</code> 选项压栈，后续 <code>.option pop</code> 可恢复。
<code>.option pop</code>	将选项弹栈，将所有 <code>.option</code> 恢复为上次 <code>.option push</code> 的配置。



## 汇编语言



- 汇编程序语法
- 注释
- 标号
- 指令
- 常用编译命令 (directive)