

Chapter 7

汇编语言

汇编语言:具有可读性的机器语言

计算机的代码是由 ‘0’ , ‘1’ 组成 ...

0001110010000110



程序员使用机器语言编程非常困难, 采用助记符号的形式来表示每条指令将会更好..., 和机器码的转换也比较容易

ADD R6,R2,R6 ; *increment index reg.*

汇编器 (Assembler) 是将符号语言翻译成机器指令的程序。

- 汇编器是和机器的**ISA**是相关的: 符号与指令集定义的指令保持相应的一致性
 - 用容易记忆的符号表示操作码
 - 内存的位置用标号表示
- 为存储分配和数据的初始化提供额外的操作支持

一个简单的LC-3汇编语言程序

```
伪操作指令  
;  
; Program to multiply a number by the constant 6  
;  
指令  
    .ORIG    x3050  
    LD      R1, SIX  
    LD      R2, NUMBER  
    AND     R3, R3, #0  
; Clear R3.  It will  
; contain the product.  
  
; The inner loop  
;  
AGAIN  ADD   R3, R3, R2  
        ADD   R1, R1, #-1  
        BRp   AGAIN  
; R1 keeps track of  
; the iteration.  
;  
        HALT  
;  
NUMBER .BLKW 1  
SIX     .FILL x0006  
;  
        .END
```

LC-3汇编语言的语法

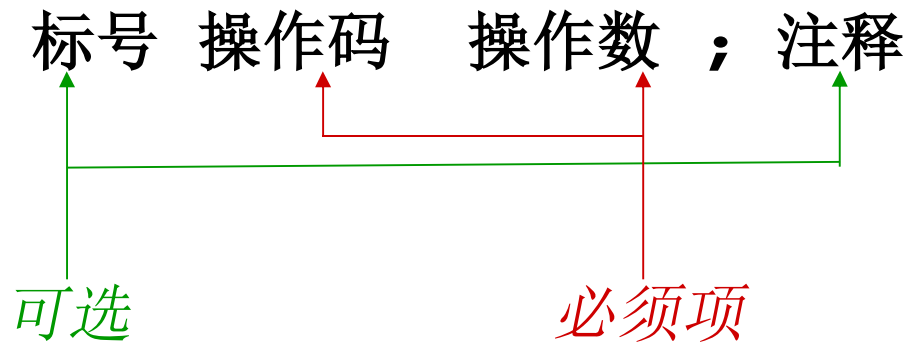
程序代码的组成:

- 可执行的机器指令
- 伪操作指令（传递给编译器指导汇编工作，不可执行）
- 注释

符号之间加入空格，不区分字母的大小写

注释 (“;”开始)，汇编器将忽略所有的注释

指令格式:



操作码、操作数

操作码

- 和**LC-3**指令集定义的操作码对应的助记符号，不能再用作标号，系统专用并保留
- 具体参见**Appendix A**

Øex: **ADD, AND, LD, LDR, ...**

操作数

- 寄存器数– 寄存器 **Rn**, **n**是寄存器编号
- 立即数 – 数字用 **# (十进制) or x (十六进制)**表示
- 内存数 – 标号，用符号表示的内存地址
- 用 ‘,’ 分割操作数
- 数字, 操作数的顺序以及类型和遵守机器码指令的定义

Øex:

```
ADD R1,R1,R3
ADD R1,R1,#3
LD  R6,NUMBER
BRz LOOP
```

数据类型

LC-3 只有两种基本数据类型

定点整数: **Integer** **16位补码**

字符: **Character** **16位**

都占用 **16 bits wide (a word)**

但实际字符只占用**8 bit**, 怎么存储??---高位零扩展

标号

标号: Label

- 放在每行代码的开始地址符号, 表示该行代码或数据的地址,
- 符号化的内存地址. 一般两种类型:

Ø 分支和跳转语句的目标地址

Ø 数据的存放地址

Øex:

```
LOOP  ADD  R1,R1,#-1  
      BRp  LOOP
```

ØEx:

```
LD    R2, NUMBER
```

...

...

```
NUMBER  .BLKW      1
```

注释

注释: **Comment**

- ‘;’ 之后的所有字符都是注释
- 汇编器将忽略所有的注释
- 注释用于帮助程序员理解程序和存档的需求
- 注释的技巧
 - Ø 不要滥用注释, 比如 “R1加1”, 没有提供比指令更多的信息
 - Ø 提供更深的洞察力, 比如 “指针加1指向下一个访问的数据”
 - Ø 分隔代码片段

编译器的伪操作

伪操作

- 不对程序产生效果，不是执行指令
- 仅供汇编器使用
- 区别于指令，以 ‘.’ 开始

<i>Opcode</i>	<i>Operand</i>	<i>Meaning</i>
.ORIG	address	指示程序起始地址
.END		指示程序在此结束，注意并不停止程序
.BLKW	n	分配 n 个字的内存单元空间
.FILL	n	分配一个字的内存单元空间并初始化为 n
.STRINGZ	n-character string	定义一个大小为 n 的字符串，占用 n+1 个内存单元。第 n+1 个字符为‘\0’。

伪操作: **.ORIG**

告诉编译器代码在内存中的起始地址

一个程序只允许一个

.ORIG伪操作

PC 在程序载入时初始化为**.ORIG**指向的地址

LC-3:内存分配

x0000 – x00FF : TRAP向量表

x0100 – x01FF 中断向量表

x0200 – x2FFF 系统STACK

x3000 – xFDFF 用户程序区域

xFE00 – xFFFF 设备寄存器

Example:

.orig x3000

LC-3的用户程序的起始地址一般设置为 **x3000**

伪操作: **.FILL**

在内存中定义并初始化程序变量，可读写
一个程序行只允许一个定义
定义变量的大小总是**16**位的字

Examples:

flag .FILL x0001

counter .FILL x0002

letter .FILL x0041

faradr .FILL x4241

访问:

LD R1,flag ;编译器会帮助产生PC相对寻址的偏移量

LDI R1,faradr ;编译器会帮助产生PC相对寻址的偏移量

伪操作: **.BLKW**

用于内存单元的分配,适用操作数
一开始不确定的场合

;保留3个未命名的内存单元

.BLKW 3

;保留1个命名的内存单元

Bob .BLKW 1

;保留7个命名的内存单元并全部
; 初始化为4

Num .BLKW 7 #4

unamed	?
	?
	?
Bob	?
num	4
	4
	4
	4

伪操作: **.STRINGZ**

在内存中定义1串字符串

在内存中连续存放

自动以 ‘\0’ 结束

哨兵: “Null-terminated”

每个字符高位0扩展, 占用16位

Example:

hello: .STRINGZ “Hello!”

访问:

LEA R0,hello

PUTS

hello	‘H’ : x0048
	‘e’ : x0065
	‘l’ : x006c
	‘l’ : x006c
	‘o’ : x006f
	‘!’ : x002c
	‘\0’

伪操作: **.END**

告诉编译器程序结束的地点

一个程序只允许一个**.END**

编译器在此停止编译，但不真正停止程序

C vs ASM

```
int a;           // simple variable (uninitialized)
int b = 2014;    // simple initialized variable
int c[10];       // array of 10 (uninitialized)
```

```
a  .BLKW 1       ; simple variable (or .FILL 0)
b  .FILL #2014    ; simple initialized variable
c  .BLKW 10 #0    ; array of ten ints (initialized to 0)
```

```
b = a;
```

```
LD R0, a ; load from memory to a register
ST R0, b ; store from register to memory
```

```
b = a + 1;
```

```
LD R0, a ; load from memory to a register
ADD R0, R0, #1 ; increment value
ST R R0, b ; store from register to memory
```

Trap 指令

LC-3 汇编器提供trap“伪指令”，方便程序员使用，无需记忆系统调用号

<i>Code</i>	<i>Equivalent</i>	<i>Description</i>
HALT	TRAP x25	Halt execution and print message to console.
IN	TRAP x23	Print prompt on console, read (and echo) one character from keybd. Character stored in R0[7:0].
OUT	TRAP x21	Write one character (in R0[7:0]) to console.
GETC	TRAP x20	Read one character from keyboard. Character stored in R0[7:0].
PUTS	TRAP x22	Write null-terminated string to console. Address of string is in R0.

Trap 指令使用

1 输出一个字符

; the char must be in R0

TRAP x21

or

OUT

2 To read in a character

; will go into R0[7:0], no echo.

TRAP x20

or

GETC

3 To end the program

TRAP x25

or

HALT

4显示字符串

LEA R0,hello

PUTS

...

hello .STRINGZ "Hello!"

汇编语言程序格式

.ORIG x3000

...

your code goes here

...

HALT

...

your memory variable definition

...

.END

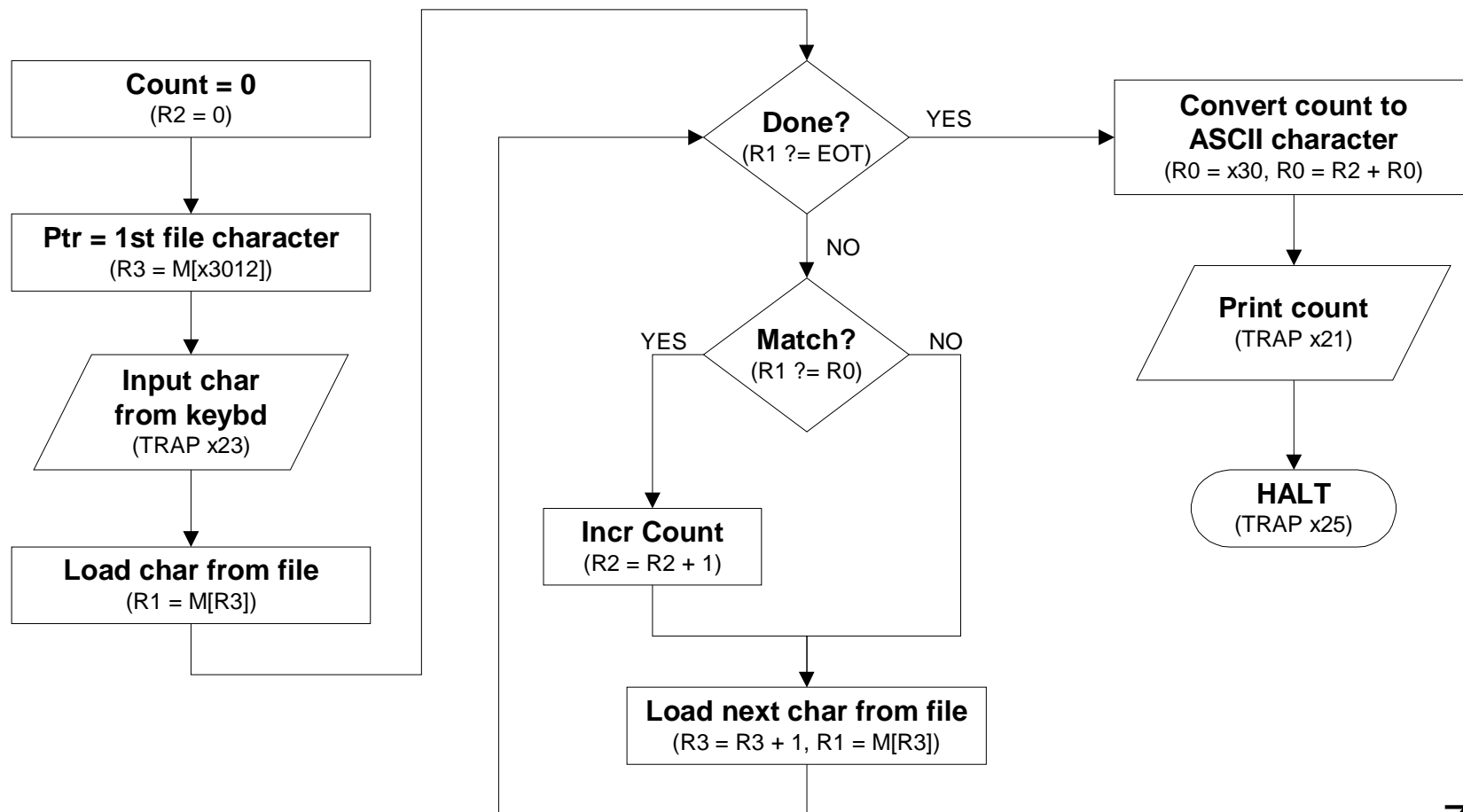
编程风格

养成良好的编程风格，增强程序的可读性和可理解性。

- 1.养成写程序头的习惯，加入作者的名字，日期，以及程序的功能等.
- 2.标号，操作码，操作数，注释都要列对齐. (除非整行都是注释.)
- 3.寄存器需要注释其用途.
- 4.为程序加入注释.
- 5.符号的名字要有意义.
 - 加入适当的大写和小写字母.
 - ASCIItoBinary, InputRoutine, SaveR1
- 6.不同的程序段加入注释分割.
- 7.尽量一行一条指令.
 - 较长的描述要整齐的分段.

示例程序

统计文件中特定字符的出现次数？还记得吗，第5章的例子



程序头

汇编语言的实现 (1/3)

```
;  
; Program to count occurrences of a character in a file.  
; Character to be input from the keyboard.  
; Result to be displayed on the monitor.  
; Program only works if no more than 9 occurrences are found.  
;  
;  
; Initialization  
;  
    .ORIG    x3000  
    AND      R2, R2, #0      ; R2 is counter, initially 0  
    LD       R3, PTR        ; R3 is pointer to characters  
    GETC     ; R0 gets character input  
    LDR      R1, R3, #0      ; R1 gets first character  
;  
; Test character for end of file  
;  
TEST      ADD      R4, R1, #-4      ; Test for EOT (ASCII x04)  
          BRz      OUTPUT          ; If done, prepare the output
```

程序片段的注释
分割

程序注释, 标注
寄存器的功能

程序注释, 标注
指令功能

汇编语言的实现 (2/3)

```
;
; Test character for match.  If a match, increment count.
;
        NOT      R1, R1
        ADD      R1, R1, R0 ; If match, R1 = xFFFF
        NOT      R1, R1    ; If match, R1 = x0000
        BRnp     GETCHAR   ; If no match, do not increment
        ADD      R2, R2, #1

;
; Get next character from file.
;
GETCHAR  ADD      R3, R3, #1 ; Point to next character.
        LDR      R1, R3, #0 ; R1 gets next char to test
        BRnzp    TEST

;
; Output the count.
;
OUTPUT   LD       R0, ASCII ; Load the ASCII template
        ADD      R0, R0, R2 ; Covert binary count to ASCII
        OUT      ; ASCII code in R0 is displayed.
        HALT     ; Halt machine
```

汇编语言的实现（2/3）

```
;
; Storage for pointer and ASCII template
;
ASCII    .FILL    x0030
PTR      .FILL    x4000
        .END
```

Discussion: LC-3汇编程序分析

```
.orig x3000
LD R2, Zero
LD R0, M0
LD R1, M1
Loop BRz Done
    ADD R2, R2, R0
    ADD R1, R1, #-1
    BRnzp Loop
Done ST R2, Res
    HALT

Res .FILL x0000
Zero .FILL x0000
M0 .FILL x0007
M1 .FILL x0003
.END
```

程序分析:

- 1 程序功能是什么?
- 2 最终**RES**的值是什么?

课堂练习

- 1 设计汇编语言程序实现 $c=2a+b$
- 2 提示用户输入一个数字，并回显。
please input a number(0~9):9

```
.ORIG x3000
AND  R2, R2, #0    ; store result
AND  R3, R3, #0    ; a is stored in R3
LD   R3, a
AND  R4, R4, #0    ; b is stored in R4
LD   R4, b
```

```
MAIN ADD R2,R3,R3
      ADD R2,R2,R4
```

```
a    .FILL  x003a
b    .FILL  x4000
```

```
.END
```

```
.ORIG x3000
LEA R0,MESSAGE ;Print Prompt onto screen (TRAP x22)
PUTS

TRAP x23 ;Input ASCII of the number and echo (IN)
TRAP x21 ;Output ASCII of the number stored in R0 (OUT)

MESSAGE STRINGZ "please input a number(0~9)"
.END
```

设计实例

设计汇编语言程序实现 **$c = a \text{ xor } b$**

设计实例

设计汇编语言程序实现 $c = a \text{ xor } b$

```
                .ORIG      x3000
                ANDR2, R2, #0      ; R2 is counter, initially 16
                ADDR2, R2, #16
                ANDR3, R3, #0      ; a is stored in R3
                LD   R3, a
                AND  R4, R4, #0      ; b is stored in R4
                LD   R4, b
                AND  R5, R5, #0      ; R5 stores mid result
                AND  R6, R6, #0      ; R6 stores the XOR result

TEST            ADD  R5, R6, R6      ; Prepare R6 to store result (left shift)
                ADD  R6, R5, #0

                NOT  R5, R3
                AND  R5, R5, R4      ;
                BRp  WR_1
                NOT  R5, R4
                AND  R5, R5, R3      ;
                BRp  WR_1

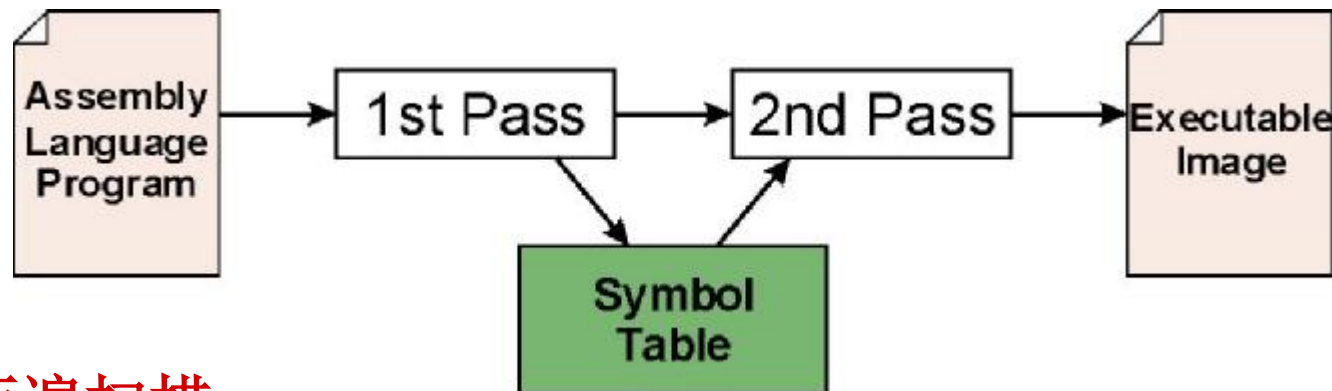
                ADD  R6, R6, #0      ; Determine the XOR result of MSB
                BRnpz NEXT
WR_1            ADD  R6, R6, #1

NEXT            ADD  R5, R3, R3      ; Prepare the next XOR calculation (left shift R3 and R4)
                ADD  R3, R5, #0
                ADD  R5, R4, R4
                ADD  R4, R5, #0
                ADD  R2, #-1          ; Check residual Loop times
                BRp  TEST

A               .FILL  x0030
B               .FILL  x4000
                .END
```

LC-3汇编过程

把汇编语言源程序(.asm)转换为可执行机器代码的过程 (.obj)



过程：两遍扫描

第一遍（ **First Pass** ）： 创建符号表

- 扫描源程序文件
- 找到所有的标号，并计算对应的地址
产生所谓的符号表（*symbol table*）

第二遍（ **Second Pass** ） 生成机器语言程序代码

- 利用符号表信息将指令转化为机器语言代码

第一遍（ **First Pass** ）： 创建符号表

1. 找到 **.ORIG** 伪操作

确定第一条指令的起始汇编地址

- 初始化地址跟踪计数器 (**LC: location counter**), 用于记录每条当前指令的地址

2. 依次扫描源程序的每一行代码

a) 如果代码行存在标号, 将标号和指令对应的**LC**添加到符号表中.

b) **LC+1**

- 说明: 如果碰到伪指令 **.BLKW** 或 **.STRINGZ**, 则**LC**的增量为对应分配的字数。

空行不处理

3. 碰到 **.END**伪操作则停止汇编过程。

NOTE: 空行: 只有一个 ‘;’ 号起始的行

例子

生成图7-2程序的符号表

Symbol	Address

课堂练习：创建程序的符号表

```
;
; Program to multiply a number by
the constant 6
;
        .ORIG      x3050
LD       R1, SIX
LD       R2, NUMBER
AND      R3, R3, #0
; Clear R3.  It will
        ; contain the product.
; The inner loop
;
AGAIN    ADD      R3, R3, R2
        ADD      R1, R1, #-1
        ; R1 keeps track of
BRp      AGAIN
        ; the iteration.
;
        HALT
;
NUMBER   .BLKW     1
SIX       .FILL    x0006
MESSAGE  .STRINGZ  "CSI IS COOL"
        .BLKW     #20
BOTTOM   .BLKW     #1
;
        .END
```

Symbol	Address

第二遍（ **Second Pass** ） 生成机器语言程序代码

对每条可执行的指令，将其转换为机器语言代码

- 如果操作数是一个标号
从符号表中查找其地址并计算

可能存在的问题：

- 操作数个数或类型不对
Øex: NOT R1,#7
ADD R1,R2
ADD R3,R3,NUMBER
- 立即数的范围太大
Øex: ADD R1,R2,#1023
- 标号对应的地址相聚指令太远
Ø超过了PC相对寻址的范围（-256~+255）

练习： 利用符号表的信息将可执行指令转化为机器语言代码

```

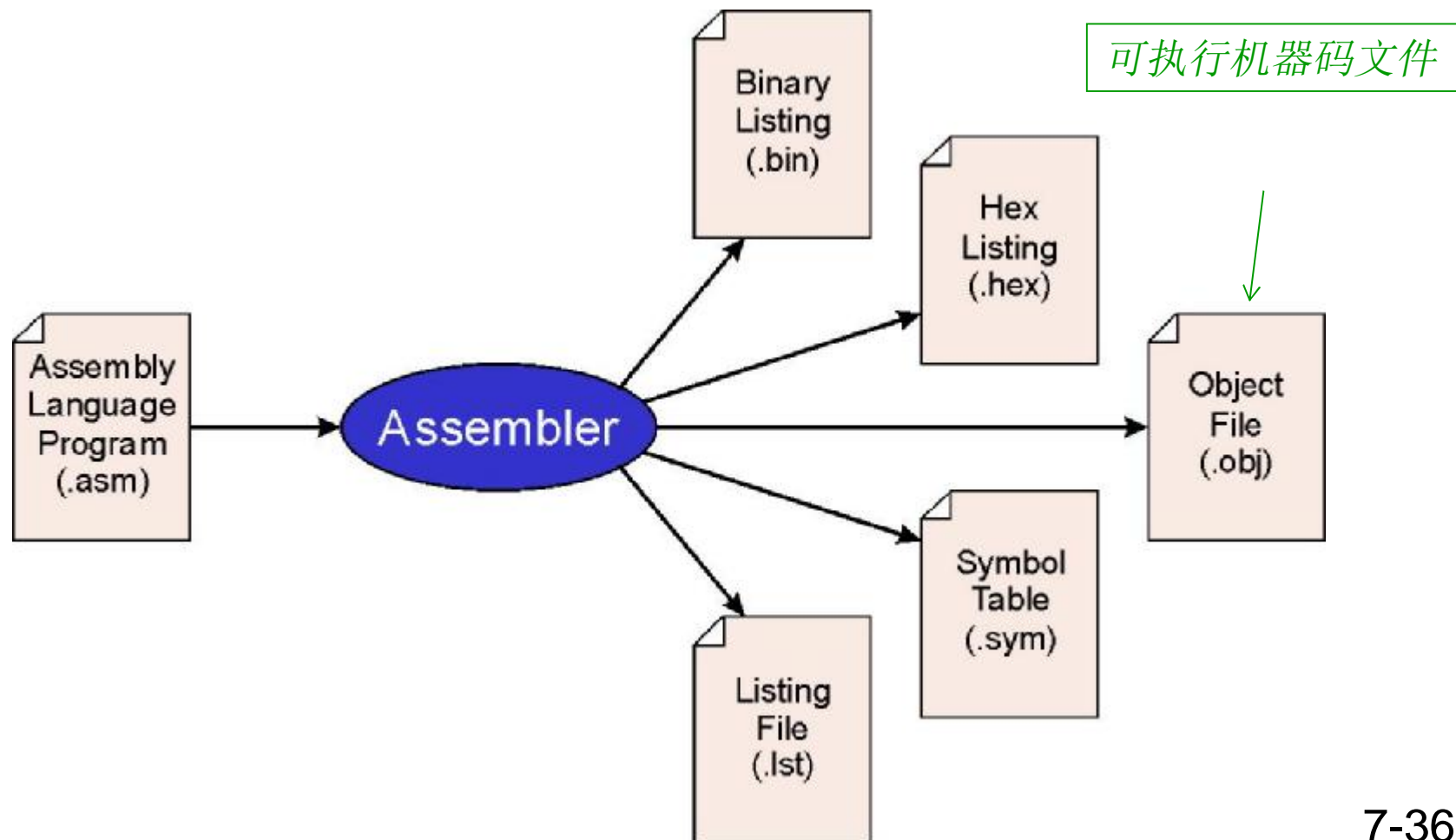
;
; Program to multiply a number by
the constant 6
;
        .ORIG      x3050
        LD         R1, SIX
        LD         R2, NUMBER
        AND        R3, R3, #0
        ; Clear R3.  It will
                ; contain the product.
; The inner loop
;
AGAIN    ADD        R3, R3, R2
        ADD        R1, R1, #-1
        ; R1 keeps track of
        BRp       AGAIN
        ; the iteration.
;
        HALT
;
NUMBER   .BLKW      1
SIX      .FILL      x0006
;
        .END

```

Statement	Machine Language
LD R1, SIX	
LD R2, NUMBER	
AND R3, R3, #0	
BRp AGAIN	

LC-3 汇编器

利用“**assemble**” (Unix) or **LC3Edit** (Windows),
产生不同的输出文件



目标文件格式

LC-3 目标文件包含以下内容

- 起始装载地址
随后是...

机器语言代码

例子:

- Beginning of “count character” object file looks like this:

001100000000000000	← .ORIG x3000
0101010010100000	← AND R2, R2, #0
0010011000010001	← LD R3, PTR
1111000000100011	← TRAP x23
•	
•	
•	

多个目标文件

一个目标文件可能不包含所有程序执行所需要的机器代码

- 系统提供的库调用
- 共享别人的成果,事先写好的子程序等

LC-3 simulator, 可以加载多个目标文件到内存, 然后从指定地址执行

- 系统调用, 比如键盘输入或是显示输出等是自动装载的
 - Ø 装入到 “**system memory**,” 位于 **x3000** 以下
 - Ø 用户例程应该加载到 **x3000 and xFDFF**
- 每个目标文件都有起始地址
- 但装载时要注意不能重合覆盖

链接与加载

Loading（加载） 将可执行文件拷贝到内存中.

- 大多数的加载器能够将可执行文件重新加载到可获取的内存空间
- 需要重新调整加载和存储跳转程序的地址

Linking（链接） 将解决不同目标文件中的符号问题.

- 假定我们在一个模块中定义的符号需要在其他的模块中使用
- 如 `.EXTERNAL` 用于告诉汇编器该符号在其他的模块中已经定义了
- 链接器将在多个模块的符号表中查找外部定义符号并在载入前生成机器代码

作业

Ex 7.1 to 7.16, 7.18 to 7.25 (yes, all of them except 7.17)