第十一章 汇编语言

汇编语言

- 目的
  - 程序设计的用户友好性比机器语言强
  - 精确控制计算机能够执行的指令
- 便于记忆的符号
  - 操作码，例如ADD和AND
  - 存储单元，例如SUM和LOOP
    - 符号地址
    - 标记

汇编语言

- 在汇编语言程序执行之前，必须被翻译成机器语言
  - 翻译程序，汇编器
  - 翻译过程，汇编

对10个整数求和的程序

```
01  ;
02  ;对10个整数求和的程序。
03  ;
04  ;10个整数及累加和
05                      .data       x0000600A
06                      .align      2
07   numbers:           .word       #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
08   sum:               .space      4
09  ;
0A  ; 初始化
0B                      .text       x40000000
0C                      .global     main
0D   main:              addi        r1, r0, numbers
0E                      addi        r3, r0, #0              ;R3清零，它将包含和
0F                      addi        r2, r0, #10             ;R2包含整数个数
10   ;
11  ;循环计算
12  again:              beqz        r2, exit
13                      lw          r4, 0(r1)
14                      add         r3, r3, r4
15                      addi        r1, r1, #4              ;R1跟踪下一个整数地址
16                      subi        r2, r2, #1
17                      j           again
18  exit:               sw          sum(r0), r3
19                      trap        #0
1A  ; 程序结束
```

汇编语言指令

```
                .....
0D   main:      addi    r1, r0, numbers
0E              addi    r3, r0, #0   ;R3清零，它将包含和
0F              addi    r2, r0, #10  ;R2包含整数个数
                .....
12   again:     beqz    r2, exit
13              lw      r4, 0(r1)
14              add     r3, r3, r4
15              addi    r1, r1, #4   ;R1跟踪下一个整数地址
16              subi    r2, r2, #1
17              j       again
18   exit:      sw      sum(r0), r3
19              trap    #0
```

- 被翻译成机器语言指令

标记（LABEL） 操作码（OPCODE） 操作数（OPERANDS） ；注释（COMMENTS）

- 标记和注释可选
- 不区分大小写
- 自由格式

标记

```
      ......
07    numbers:    .word    #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
08    sum:        .space   4
      ......
0D    main:       addi     r1, r0, numbers
0E                addi     r3, r0, #0   ;R3清零，它将包含和
0F                addi     r2, r0, #10  ;R2包含整数个数
      ......
12    again:      beqz     r2, exit
13                lw       r4, 0(r1)
14                add      r3, r3, r4
15                addi     r1, r1, #4   ;R1跟踪下一个整数地址
16                subi     r2, r2, #1
17                j        again
18    exit:       sw       sum(r0), r3
19                trap     #0
```

- 标识存储单元
- 命名
  - 由字母、数字及下划线组成
  - 以字母、下划线或$开头
  - 以冒号结尾
  - 指令操作码属于保留字，不能用做标记
  - 例如，NOW:，_21:，R2D:和$3PO:

操作码和操作数

```
.....
0D  main:   addi    r1, r0, numbers
0E          addi    r3, r0, #0   ;R3清零，它将包含和
0F          addi    r2, r0, #10  ;R2包含整数个数
.....
12  again:  beqz    r2, exit
13          lw      r4, 0(r1)
14          add     r3, r3, r4
15          addi    r1, r1, #4   ;R1跟踪下一个整数地址
16          subi    r2, r2, #1
17          j       again
18  exit:   sw      sum(r0), r3
19          trap    #0
```

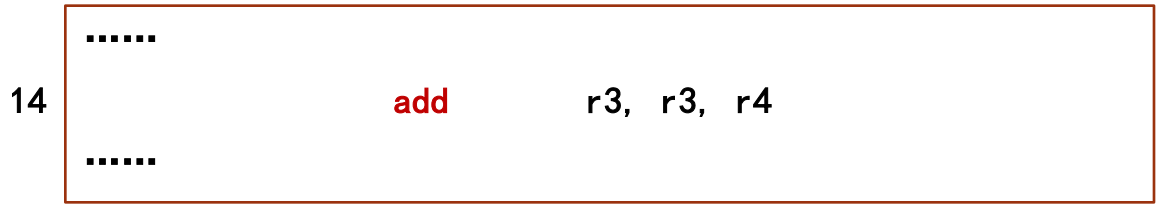
- **操作码**：指令操作码的符号名
- 操作数
  - 寄存器
    - R0，R1，…，R31
  - 立即数
    - 包含一个表明该数的基的符号
      - “#”，十进制
      - “x”，十六进制
      - “b”，二进制
    - 标记，代表一个数据的地址

算术/逻辑运算指令-1

```
      ......
0D    main:       addi    r1, r0, numbers     ;将numbers代表的地址值赋给R1
0E                addi    r3, r0, #0          ;R3清零，它将包含和
0F                addi    r2, r0, #10         ;R2包含整数个数
      ......
15                addi    r1, r1, #4          ;R1跟踪下一个整数地址
16                subi    r2, r2, #1
      ......
```

- 操作数数目为3个(LHI指令除外)
- I-类型汇编指令格式
  - OPCODE　DR, SR1, Imm16
    - 立即数可以使用标记
    - 立即数是16位补码整数

算术/逻辑运算指令-2



- R-类型汇编指令格式
  - OPCODE DR, SR1, SR2
- LHI指令格式
  - LHI DR, Imm16
    - 立即数可以使用标记，如 LHI R1, A
      - 将地址A的高16位值赋给R1
      - 如A代表地址x3000 01A0，R1=x3000 0000

数据传送指令

```
      ......
13                lw        r4, 0(r1)
      ......
18    exit:       sw        sum(r0), r3     ;将R3的值存储到sum标识的地址中
      ......
```

- 加载指令汇编格式
  - LW/LB　　DR, Imm16(SR1)
- 存储指令汇编格式
  - SW/SB　　Imm16(SR1)，DR
- 立即数可以使用标记

边界对齐

- LW和SW指令
  - “基址寄存器+偏移量”必须是4的倍数
  - 起始地址

控制指令-1

```
      ......
12    again:        beqz      r2, exit
      ......
17                  j         again
18    exit:         sw        sum(r0), r3
19                  trap      #0
```

- 条件分支指令格式
  - OPCODE　SR1，LABEL
    - 标记，条件分支指令的目标地址

控制指令-2

```
      ......
12    again:        beqz      r2, exit
      ......
17                  j         again
18    exit:         sw        sum(r0), r3
19                  trap      #0
```

- J指令格式
  - OPCODE　　LABEL
- JR指令格式
  - OPCODE　　SR1

控制指令-3

```
      ......
12    again:        beqz      r2, exit
      ......
17                  j         again
18    exit:         sw        sum(r0), r3
19                  trap      #0
```

- TRAP指令格式
  - TRAP　　Imm

注释

```
01  ;
02  ;对10个整数求和的程序。
03  ;
04  ;10个整数及累加和
.....
09  ;
0A  ; 初始化
.....
0E                      addi        r3, r0, #0          ;R3清零，它将包含和
0F                      addi        r2, r0, #10         ;R2包含整数个数
10  ;
11  ;循环计算
.....
15                      addi        r1, r1, #4          ;R1跟踪下一个整数地址
.....
1A  ; 程序结束
```

- 分号后面的部分
  - 某行的第一个非空字符
  - 一条指令之后
- 目的：提高可读性，不是重申显而易见的表象

提高可读性

```
01  ;
02  ;对10个整数求和的程序。
03  ;
04  ;10个整数及累加和
.....
09  ;
0A  ;  初始化
.....
0E                      addi        r3, r0, #0              ;R3清零，它将包含和
0F                      addi        r2, r0, #10             ;R2包含整数个数
10  ;
11  ;循环计算
.....
15                      addi        r1, r1, #4              ;R1跟踪下一个整数地址
.....
1A  ;  程序结束
```

- 注释为空行
- 程序对齐

伪操作

```
.....
05                      .data       x0000600A
06                      .align      2
07    numbers:          .word       #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
08    sum:              .space      4
.....
0B                      .text       x40000000
0C                      .global     main
.....
```

- Directive
  - 有助于汇编器实现翻译过程
- 以“点”作为第一个字符

数据区/代码区

```
.....
05                      .data       x0000600A
06                      .align      2
07    numbers:          .word       #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
08    sum:              .space      4
.....
0B                      .text       x40000000
0C                      .global     main
.....
```

- 汇编语言程序：指令和数据
- 数据和指令被加载到存储器中的不同区域
  - 数据区：.data
  - 代码区：.text

数据区

```
.....
05                      .data       x0000600A
06                      .align      2
07    numbers:          .word       #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
08    sum:              .space      4
.....
0B                      .text       x40000000
0C                      .global     main
.....
```

.data address

- 将数据放在数据区的某个地方
- 注意:
  - x0000 600A不是4的倍数，不能作为字的起始地址

边界对齐

```
.....
05                      .data       x0000600A
06                      .align      2                   ;被加载的数据将从地址x0000 600C开始
07  numbers:            .word       #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
08  sum:                .space      4
.....
0B                      .text       x40000000
0C                      .global     main
.....
```

.align n

- 将下面的数据或代码加载到以n个0结尾的地址中

数据区的数据

- 32位的字、8位的字节或字符串
- .word、.space、.ascii、.asciiz、.byte

字（32位）

```
.....
05                  .data       x0000600A
06                  .align      2                ;被加载的数据将从地址x0000 600C开始
07  numbers:        .word       #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
08  sum:            .space      4
.....
0B                  .text       x40000000
0C                  .global     main
0D  main:           addi        r1, r0, numbers
.....
```

.word word1, word2, ...

- 将字1、字2、……存储在连续的存储单元中

字节和字符串

.byte byte1, byte2,...

- 将字节1、字节2、……存储在连续的单元之中

.ascii "string1","..."

- 将字符串1、字符串2、……存储于存储器中

字符串

```
……
        .data           x30000000
        .asciiz         "Hello, World!"
……
```

地址	字节
x3000 0000	x48
x3000 0001	x65
x3000 0002	x6C
x3000 0003	x6C
x3000 0004	x6F
x3000 0005	x2C
x3000 0006	x20
x3000 0007	x57
x3000 0008	x6F
x3000 0009	x72
x3000 000A	x6C
x3000 000B	x64
x3000 000C	x21
x3000 000D	x00

.asciiz "string1", "..."

- 在每一个字符串末尾，存储一个字节0

预留空间

```
.....
05                  .data       x0000600A
06                  .align      2                ;被加载的数据将从地址x0000 600C开始
07  numbers:        .word       #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
08  sum:            .space      4                ;x0000 6034-x0000 6037,保存计算出来的累加和
.....
0B                  .text       x40000000
0C                  .global     main
.....
18  exit:           sw          sum(r0), r3      ;将R3的值存储到x0000 6034-x0000 6037中
.....
```

.space size

- 在数据区中留出一定数目的连续的存储单元
  - 数目：size个字节
- 操作数的实际值未知

代码区

```
.....
05                  .data       x0000600A
06                  .align      2                ;被加载的数据将从地址x0000 600C开始
07  numbers:        .word       #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
08  sum:            .space      4                ;x0000 6034-x0000 6037,保存计算出来的累加和
.....
0B                  .text       x40000000
0C                  .global     main
.....
```

.text address

- 将指令放在存储器的某个地方
- 指令的起始地址必须是4的倍数
  - .align

全局标记

- 多个文件组成的汇编语言程序

.global label

  - 全局标记
  - 其他文件可以使用

.global main

```
.....
05                  .data       x0000600A
06                  .align      2                ;被加载的数据将从地址x0000 600C开始
07  numbers:        .word       #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
08  sum:            .space      4                ;x0000 6034-x0000 6037,保存计算出来的累加和
.....
0B                  .text       x40000000
0C                  .global     main
0D  main:           addi        r1, r0, numbers
.....
```

- 先执行哪一个文件？
  - 从标记为main:的指令开始
  - main是全局标记
- 单文件程序
  - 从main:开始执行

示例：文档加密

```
01  ;
02  ; 用于对文档进行加密的程序。注：数据区在另一个汇编语言程序文件中
03  ; 判断是否需要加密的字符由键盘输入。
04  ; 是否加密的结果显示在显示器上。
05  ;
06                  .text     x04000000
07                  .global   main
08  main:           trap      x06                 ; R4获取输入的字符
09  ;初始化
0A                  subi      r4, r4, x30         ; R4为数值n
0B                  lhi       r3, x1000           ; R3是字符的指针
0C                  lb        r1, 0(r3)           ; R1取得下一个字符
0D  ;
```

```
0E          ; 检验字符，否到达文件的末尾
0F          ;
10          TEST:     seqi      r2, r1, #4          ; 检验EOT
11                    bnez      r2, OUTPUT          ; 如果完成，准备输出
12          ;
13          ; 根据字符值进行加密
14          ;
15                    addi      r5, r0, #127
16                    sub       r5, r5, r4
17                    slt       r2, r1, r5          ; 检验是否小于127-n
18                    bnez      r2, ADDN            ; 如果小于127-n，执行加n
19                    subi      r5, r5, #33
1A                    sub       r1, r1, r5          ; 如果不小于127-n，减94-n
1B                     j        GETCHAR
1C          ADDN:     add       r1, r1, r4
```

```
1D      ;
1E      ;从文档中取得下一个字符
20      ;
21      GETCHAR:        sb      0(r3), r1           ; 存储R1
22                      addi    r3, r3, #1          ; 指针加1
23                      lb      r1, 0(r3)    ; R1取得下一个检验的字符
24                      j       TEST
25      ;
26      ; 输出结果
27      ;
28      OUTPUT:         addi    r4, r0, x59
29                      trap    x07                 ; 显示R4中的“Y”
2A                      trap    x00                 ; 停止机器
```

汇编过程

- 汇编器
  - 将汇编语言程序，翻译成机器语言程序
  - 汇编语言指令和机器语言指令：“一一对应”

对10个整数求和的程序

```
01  ;
02  ;对10个整数求和的程序。
03  ;
04  ;10个整数及累加和
05                      .data       x0000600A
06                      .align      2
07   numbers:           .word       #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
08   sum:               .space      4
09  ;
0A  ; 初始化
0B                      .text       x40000000
0C                      .global     main
0D   main:              addi        r1, r0, numbers
0E                      addi        r3, r0, #0              ;R3清零，它将包含和
0F                      addi        r2, r0, #10             ;R2包含整数个数
10   ;
11  ;循环计算
12  again:              beqz        r2, exit
13                      lw          r4, 0(r1)
14                      add         r3, r3, r4
15                      addi        r1, r1, #4              ;R1跟踪下一个整数地址
16                      subi        r2, r2, #1
17                      j           again
18  exit:               sw          sum(r0), r3
19                      trap        #0
1A  ; 程序结束
```

扫描-1

```
01  ;
02  ;对10个整数求和的程序。
03  ;
04  ;10个整数及累加和
05                      .data       x0000600A
06                      .align      2
……
```

- 从顶部开始
- 01到04行
  - 抛弃——注释
- 05行
  - 该程序的数据起始于地址x0000 600A
- 06行
  - 下面的数据起始于地址x0000 600C

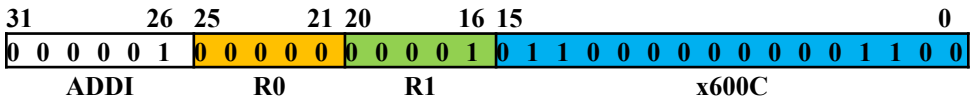
扫描-2

```
……
07   numbers:          .word      #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
08   sum:              .space     4
09   ;
0A   ; 初始化
0B                     .text      x40000000
0C                     .global    main
0D   main:             addi       r1, r0, numbers
……
```

- 07行
  - 将十个十进制整数依次翻译为二进制补码整数
- 08行
  - 保留4个单元
- 09和0A行
  - 抛弃
- 0B行
  - 该程序的指令起始于地址x4000 0000
- 0C行
  - main是一个全局标记
- 0D行
  - 不知道符号地址numbers的意思，无法翻译。汇编过程失败。

“两趟” 扫描

- “第一趟”扫描：
  - 标识出符号地址（标记）对应的实际的二进制地址
  - 建立符号表
  - 如：numbers ——— x0000 600C
- “第二趟”扫描：
  - 把汇编语言指令翻译成机器语言指令
  - 如：addi r1, r0, numbers



符号表

- 符号名和存储地址对应的关系
- 用分配的地址标识标记

第一趟-1

```
01  ;
02  ;对10个整数求和的程序。
03  ;
04  ;10个整数及累加和
05                          .data       x0000600A
……
```

- 从顶部开始
- 01到04行
  - 抛弃——注释
- 05行
  - 该程序的数据起始于地址x0000 600A
  - 地址计数器LC（Location Counter）←x0000 600A

第一趟-2

```
.....
06                      .align      2
07    numbers:          .word       #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
.....
```

- 06行
  - 下面的数据起始于地址x0000 600C
  - LC←x0000 600C
- 07行
  - 标记numbers:，在符号表中增加一条纪录

符号	地址
numbers	x0000 600C

  - LC←x0000 6034

第一趟-3

```
.....
08   sum:                .space      4
09  ;
0A  ; 初始化
.....
```

- 08行
  - 标记sum:

符号	地址
numbers	x0000 600C
sum	x0000 6034

  - LC←x0000 6038
- 09和0A行
  - 抛弃

第一趟-4

```
.....
0B                      .text       x40000000
0C                      .global     main
0D   main:              addi        r1, r0, numbers
.....
```

- 0B行
  - 代码被分配到以地址x4000 0000开头的空间中
  - LC←x4000 0000
- 0C行
  - main是全局标记
  - LC不变
- 0D行
  - 标记main:
  - LC←x4000 0004

符号	地址
numbers	x0000 600C
sum	x0000 6034
main	x4000 0000

第一趟-5

```
.....
12  again:          beqz        r2, exit
13                  lw          r4, 0(r1)
14                  add         r3, r3, r4
15                  addi        r1, r1, #4          ;R1跟踪下一个整数地址
16                  subi        r2, r2, #1
17                  j           again
18  exit:           sw          sum(r0), r3
.....
```

- ……
- 12行
  - again:
- ……
- 18行
  - exit:

符号	地址
numbers	x0000 600C
sum	x0000 6034
main	x4000 0000
again	x4000 000C
exit	x4000 0024

第二趟

- 在符号表的帮助下，再次遍历汇编语言程序
  - 汇编语言指令被翻译成机器语言指令

第二趟-1

```
01  ;
02  ;对10个整数求和的程序。
03  ;
04  ;10个整数及累加和
05                      .data       x0000600A
06                      .align      2
07   numbers:           .word       #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
08   sum:               .space      4
....
```

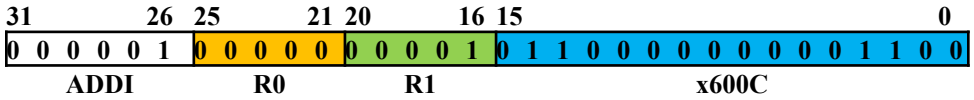
- 从顶部开始
- 01到04行
  - 抛弃——注释
- 05行
  - LC←x0000 600A
- 06行
  - LC←x0000 600C
- 07行
  - 将10个整数依次翻译为二进制补码整数
  - LC←x0000 6034
- 08行
  - 留下4个单元
  - LC←x0000 6038

第二趟-2

```
.....
09  ;
0A  ; 初始化
0B                      .text       x40000000
0C                      .global     main
0D   main:              addi        r1, r0, numbers
.....
```

符号	地址
numbers	x0000 600C
sum	x0000 6034
main	x4000 0000
again	x4000 000C
exit	x4000 0024

- 09和0A行
  - 抛弃
- 0B行
  - LC←x4000 0000
- 0C行
  - LC不变
- 0D行
  - 翻译，地址x4000 0000~ x4000 0003：



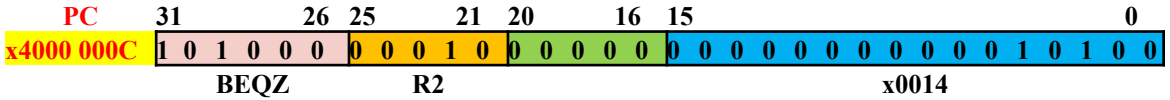
  - LC←x4000 0004

第二趟-3

```
0E                      addi        r3, r0, #0          ;R3清零，它将包含和
0F                      addi        r2, r0, #10         ;R2包含整数个数
10   ;
11  ;循环计算
12  again:              beqz        r2, exit
.....
```

符号	地址
numbers	x0000 600C
sum	x0000 6034
main	x4000 0000
again	x4000 000C
exit	x4000 0024

- 0E和0F行
  - 直接翻译
  - LC←x4000 000C
- 12行
  - 翻译
    - 增加了4的PC:LC+4，即x4000 0010
    - 偏移量:x0014（x4000 0024 - x4000 0010）



    - LC← x4000 0010

第二趟-4

```
13                  lw      r4, 0(r1)
14                  add     r3, r3, r4
15                  addi    r1, r1, #4          ;R1跟踪下一个整数地址
16                  subi    r2, r2, #1
17                  j       again
.....
```

符号	地址
numbers	x0000 600C
sum	x0000 6034
main	x4000 0000
again	x4000 000C
exit	x4000 0024

- 13~16
  - LC← x4000 0020
- 17行
  - 翻译
    - 增加了4的PC:LC+4，即x4000 0024
    - 偏移量:xFFE8（x4000 000C - x4000 0024）

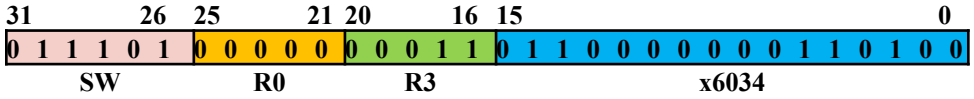
  - LC← x4000 0024

第二趟-5

```
18  exit:               sw          sum(r0), r3
19                      trap        #0
.....
```

符号	地址
numbers	x0000 600C
sum	x0000 6034
main	x4000 0000
again	x4000 000C
exit	x4000 0024

- 18行
  - 翻译



  - LC← x4000 0028
- 19行
  - 直接翻译
  - LC← x4000 002C

机器语言程序

地址	二进制
x0000 600C~x0000 600F	0000 0000 0000 0000 0000 0000 0000 1010
x0000 6010~x0000 6013	0000 0000 0000 0000 0000 0000 0000 0011
x0000 6014~x0000 6017	0000 0000 0000 0000 0000 0000 0000 0100
x0000 6018~x0000 601B	0000 0000 0000 0000 0000 0000 0000 0110
x0000 601C~x0000 601F	0000 0000 0000 0000 0000 0000 0000 1000
x0000 6020~x0000 6023	1111 1111 1111 1111 1111 1111 1111 1110
x0000 6024~x0000 6027	0000 0000 0000 0000 0000 0000 0010 1101
x0000 6028~x0000 602B	0000 0000 0000 0000 0000 0000 0000 0101
x0000 602C~x0000 602F	0000 0000 0000 0000 0000 0000 0000 1000
x0000 6030~x0000 6033	0000 0000 0000 0000 0000 0000 0000 1001
x0000 6034~x0000 6037	……
……	……
x4000 0000~x4000 0003	000001 00000 00001 0110 0000 0000 1100
x4000 0004~x4000 0007	000001 00000 00011 0000 0000 0000 0000
x4000 0008~x4000 000B	000001 00000 00010 0000 0000 0000 1010
x4000 000C~x4000 000F	101000 00010 00000 0000 0000 0001 0100
x4000 0010~x4000 0013	011100 00001 00100 0000 0000 0000 0000
x4000 0014~x4000 0017	000000 00011 00100 00011 00000 000001
x4000 0018~x4000 001B	000001 00001 00001 0000 0000 0000 0100
x4000 001C~x4000 001F	000011 00010 00010 0000 0000 0000 0001
x4000 0020~x4000 0023	101100 111111 1111 1111 1111 1110 1000
x4000 0024~x4000 0027	011101 00000 00011 0110 0000 0011 0100
x4000 0028~x4000 002B	110000 00000000000000000000000000

立即数是标记——ADDI指令

```
.....
05                  .data       x3000000A
06                  .align      2
07   numbers:       .word       #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
.....
0D   main:          addi        r1, r0, numbers
.....
```

- 问题：
  - 符号表：numbers —— x3000 000C
  - 无法用16位的立即数表示，如何解决？
    - 如何将R1的值设为x3000 000C？
- 解决方案：
  - 翻译为两条指令

```
LHI         R1, x3000          ; R1=x3000 0000
ADDI        R1, R1, x000C      ; R1=x3000 000C
```

  - LC ← LC+8

立即数是标记——SW指令

```
.....
05                      .data       x3000000A
06                      .align      2
07   numbers:           .word       #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
08   sum:               .space      4
.....
18  exit:               sw          sum(r0), r3
19                      trap        #0
.....
```

- 问题：
  - 符号表：sum —— x3000 0034
  - 无法用16位的立即数表示，如何解决？
    - 如何将“基址寄存器+偏移量”的值设为x3000 0034？
- 解决方案：
  - 翻译为两条指令

```
LHI         R5, x3000              ;使用临时寄存器R5=x3000 0000
SW          x0034 (R5), R3         ;基址寄存器+偏移量=x3000 0034
```

  - LC ← LC+8

符号表

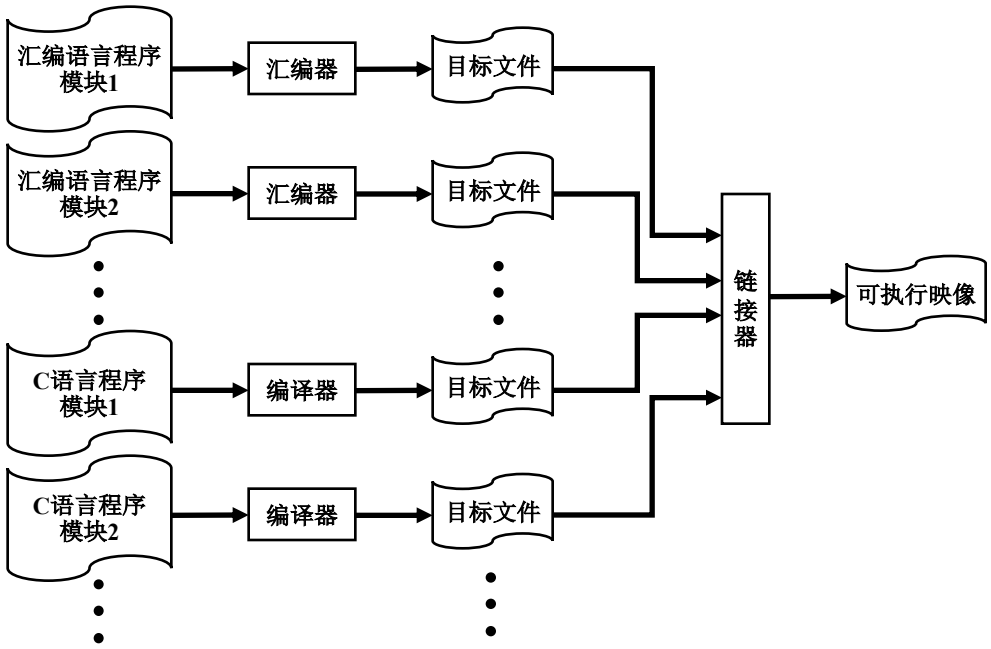
```
.....
07   numbers:        .word     #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
08   sum:            .space    4
.....
0D   main:           addi      r1, r0, numbers        ;翻译为2条指令
.....
12  again:           beqz      r2, exit
.....
18  exit:            sw        sum(r0), r3            ;翻译为2条指令
.....
```

- 立即数是标记
  - 根据需要，翻译为多条指令
  - 符号表相应调整

符号	原地址	新地址
numbers	x0000 600C	x3000 000C
sum	x0000 6034	x3000 0034
main	x4000 0000	x4000 0000
again	x4000 000C	x4000 0010
exit	x4000 0024	x4000 0028

可执行映像

- 被执行的程序实体
- 由不同模块组成（C模块、汇编模块）
  - 翻译为目标文件
  - 链接目标文件，形成可执行映像



问题

- 模块A
  - 计算整数和，不包括计算用的数据

```
……
ADDI        R1, R0, numbers
……
```

- 模块B
  - 数据输入

```
……
numbers:    .space   #40        ;用于存储输入的10个整数
sum:        .space   #4         ;用于存储整数和
……
```

.global

- 模块B

```
……
                .global  numbers
numbers:        .space   #40
……
```

- B的符号表

符号	地址	属性
numbers	x3000 000C	global
……		
……		

.extern

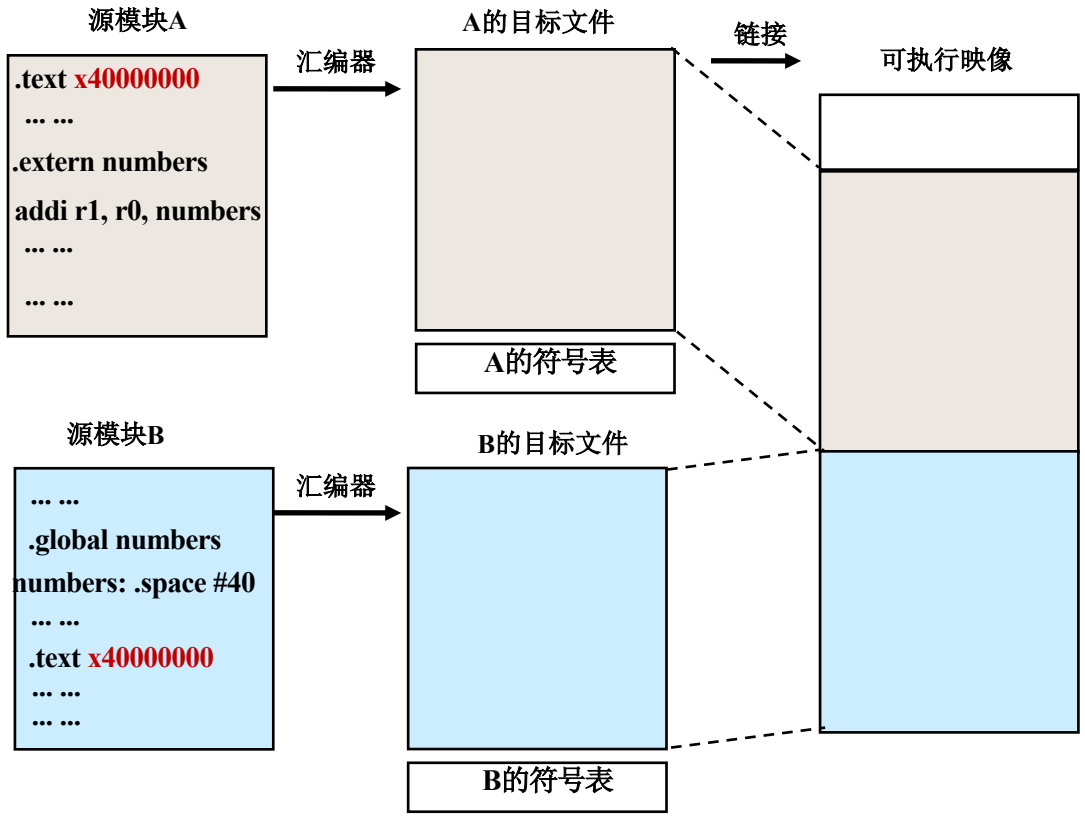
- 模块A

```
……
                .extern   numbers
                ADDI      R1, R0, numbers
……
```

- 汇编器无法实现完全汇编
  - 不报错，标记合法

链接器

- 管理“结合”过程的程序
  - 完成翻译
  - 为每个模块重新分配存储空间



address可选

- 伪操作“.text/.data address”
  - address可选
- “.text”/“.data”
  - 不必给出具体地址

DLX模拟器

- 附IV
- DLX套件.ppt
- 视频

习题

上机作业

- 10. 3
- 10. 6
  - 2)
- 10. 7
- 11. 1
- 11. 2
- 11. 3
- 11. 6

书面作业

- 10. 8
- 11. 8
- 11. 10
- 11. 12
- 11. 14

测试和调试

- 可以计算所有整数乘法的程序？
- 使用-6和3，4和-12，-5和-7做初始值
  - 使用这些测试集存在一个问题：忽略了最重要的初始值——0
- 要点：对于一个可以运行的程序，它必须能对所有的值运行
  - 好的测试：用那些不寻常的值进行初始化，那是程序员可能没有考虑的值。这些值通常被称为“极端状况”

例1：判断一段连续的存储单元内是否包含一个5

- 检查从地址x3000 0000 开始存储的10个整数中是否包含5，只要有1个5，就把R1设置为1，如果一个5都没有则使R1为0。

地址	31 26	25 21	20 16	15 11 10 6 5 0	
x4000 0000	000001	00000	00001	0000 0000 0000 0001	R1←1
x4000 0004	000001	00000	00010	0000 0000 0000 1010	R3←10
x4000 0008	001100	00000	00100	0011 0000 0000 0000	R4←首地址
x4000 000C	011100	00100	00010	0000 0000 0000 0000	R2←M[R4]
x4000 0010	010100	00010	00101	0000 0000 0000 0101	R2?=5
x4000 0014	101001	00101	00000	0000 0000 0001 0100	BNEZ
x4000 0018	000001	00100	00100	0000 0000 0000 0100	R4← R4+4
x4000 001C	000011	00011	00011	0000 0000 0000 0001	R3← R3-1
x4000 0020	011100	00100	00010	0000 0000 0000 0000	R2←M[R4]
x4000 0024	101001	00010	00000	1111 1111 1110 1000	BNEZ
x4000 0028	000001	00000	00001	0000 0000 0000 0000	R1←0
x4000 002C	110000	000000 0000 0000 0000 0000 0000			HALT

- 使用如下样本数据：1，2，3，0，5，6，7，8，9，10，运行这个程序，程序结束时R1等于0
- 如何调试？

例2：找到一个字中的第一个“1”

- 检查一个存储于x3000 0000~x3000 0003中的整数，找出被设为1的第一位（从左到右），而且把那一位的位置存储到R1中，如果没有任何一位被设为1，这段程序就把-1存储到R1中。

地址	31 26	25 21	20 16	15 11 10 6 5 0	
x4000 0000	000001	00000	00001	0000 0000 0001 1111	R1←31
x4000 0004	001100	00000	00100	0011 0000 0000 0000	R4←首地址
x4000 0008	011100	00100	00010	0000 0000 0000 0000	R2←M[R4]
x4000 000C	010000	00010	00011	0000 0000 0000 0000	R2<0?
x4000 0010	101001	00011	00000	0000 0000 0001 0100	BNEZ
x4000 0014	000011	00001	00001	0000 0000 0000 0001	R1← R1-1
x4000 0018	001101	00010	00010	0000 0000 0000 0001	R2左移1位
x4000 001C	010000	00010	00011	0000 0000 0000 0000	R2<0?
x4000 0020	101001	00011	00000	0000 0000 0000 0100	BNEZ
x4000 0024	101100	111111 1111 1111 1111 1110 1100			J x4000 0014
x4000 0028	110000	000000 0000 0000 0000 0000 0000			HALT

- 使用数据x0000 0000来运行这段程序时，无法停止
- 如何调试？

调试

- 与C语言程序类似
  - 模块化程序设计，从大任务到小任务
- 不同：跟踪的是指令序列的执行，以及每条指令执行后得到的结果

调试操作

- 与C语言的源水平调试器不同
  - 从机器指令集结构水平上，完成一些基本的交互式调试
- 与C语言的源水平调试器类似
  - 模拟器也提供了断点、观察点、单步和显示值等功能，包括：
    - 在存储器和寄存器中设置值
    - 顺序地执行一个程序中的指令
    - 能够按照期望停止执行
    - 在程序中的任何地方，检查存储器和寄存器中的内容

调试程序片段：使用加法指令实现乘法运算

- 将分别来自于R4和R5的两个正数做乘法运算

地址	31 26	25 21	20 16	15 11	10 6	5 0	
x4000 0000	000001	00000	00010	0000 0000 0000 0000			R2←0
x4000 0004	000000	00010	00100	00010	00000	000001	R2←R2+R4
x4000 0008	000011	00101	00101	0000 0000 0000 0001			R5← R5-1
x4000 000C	010000	00101	00011	0000 0000 0000 0000			R5<0?
x4000 0010	101000	00011	00000	1111 1111 1111 0000			BEQZ R3
x4000 0014	110000	000000 0000 0000 0000 0000 0000					HALT

1、使用“设置值”命令，在R4中设置10，R5中为3；运行程序，结果R2=40？

2、通过单步调试，跟踪程序；

3、使用断点跟踪程序。

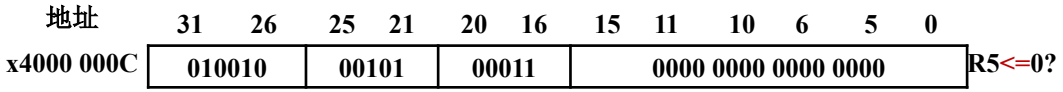
跟踪结果

PC	R2	R3	R4	R5
x4000 0000	0		10	3
x4000 0004	10		10	3
x4000 0008	10		10	2
x4000 000C	10	0	10	2
x4000 0010	10	0	10	2
x4000 0004	20	0	10	2
x4000 0008	20	0	10	1
x4000 000C	20	0	10	1
x4000 0010	20	0	10	1
x4000 0004	30	0	10	1
x4000 0008	30	0	10	0
x4000 000C	30	0	10	0
x4000 0010	30	0	10	0
x4000 0004	40	0	10	0
x4000 0008	40	0	10	-1
x4000 000C	40	1	10	-1
x4000 0010	40	1	10	-1

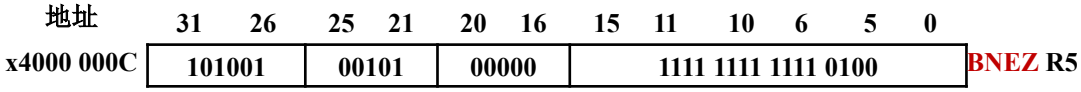
PC	R2	R3	R4	R5
x4000 0010	10	0	10	2
x4000 0010	20	0	10	1
x4000 0010	30	0	10	0
x4000 0010	40	1	10	-1

更正

- 把x4000 000C ~x4000 000F存储的指令替换为



- 或删除x4000 0010 ~x4000 0013行的指令，将x4000 000C ~x4000 000F行的指令改为：



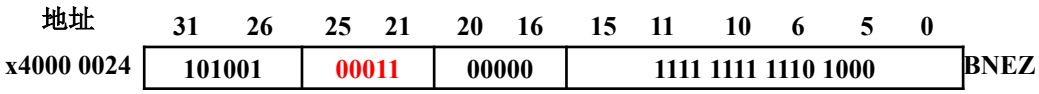
例1调试

- 从地址x3000 0000开始存储的10个整数
  - 有5，R1设置为1
  - 没有5，R1为0
- 使用断点跟踪程序
  - 在x4000 0024处设断点，检查每次循环的结果

PC	R1	R2	R3	R4
x4000 0024	1	2	9	x3000 0004
x4000 0024	1	3	8	x3000 0008
x4000 0024	1	0	7	x3000 000C

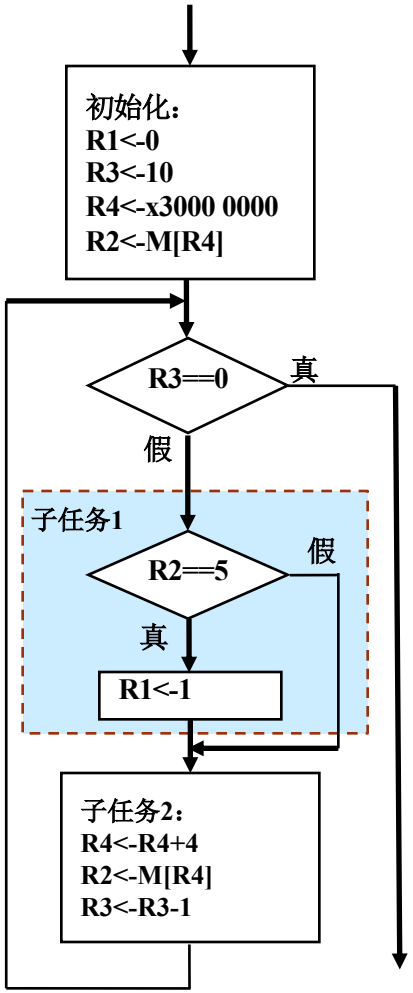
例1调试

- 更正：
  - 把x4000 0024 ~x4000 0027存储的指令中的R2替换为R3

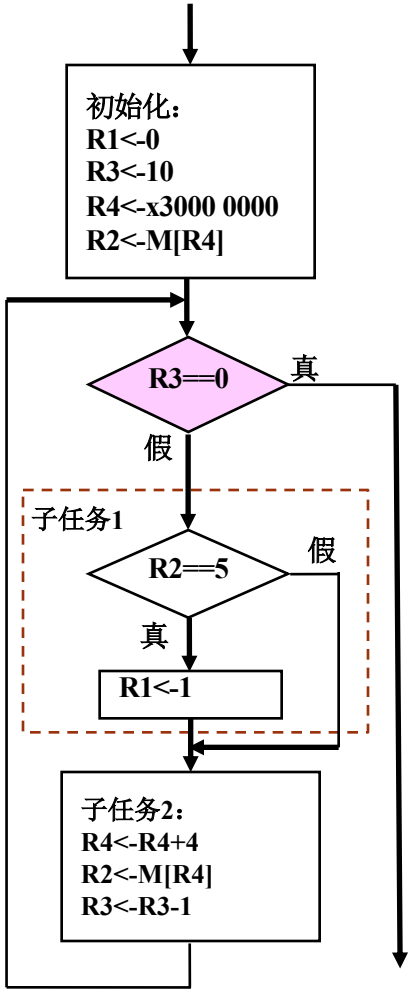


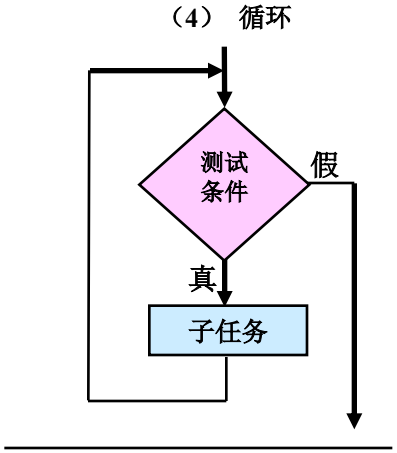
例1分析

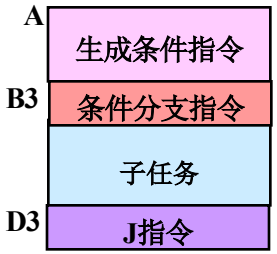
- 计数器控制的循环
  - R3，计数器
- 子任务1
  - 选择结构



测试条件 R3==0

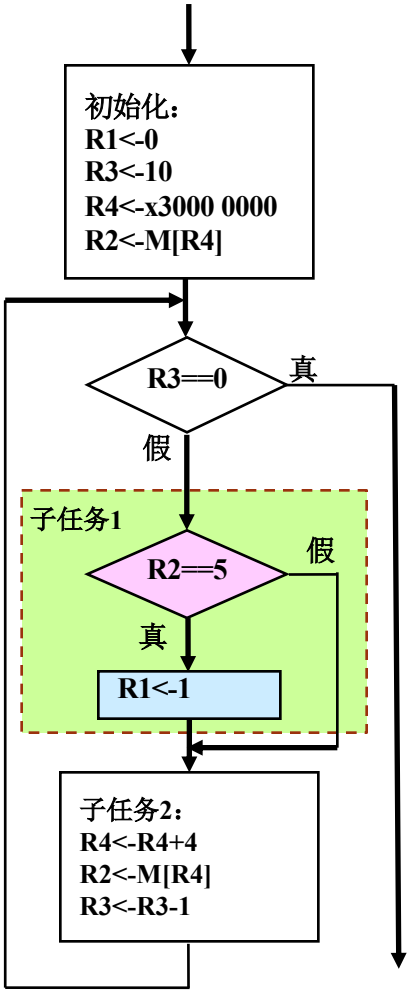


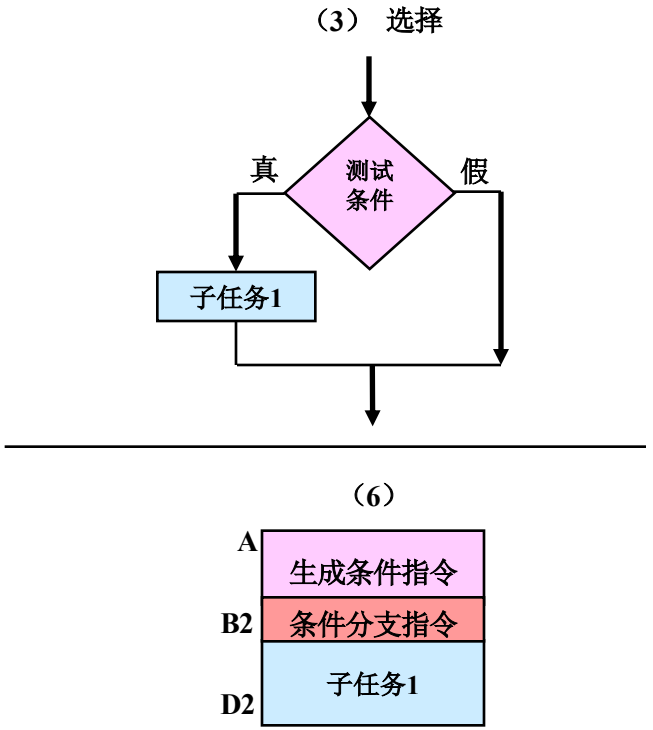




- 不需要生成条件指令
- 条件分支指令
  - BEQZ R3, D3+4

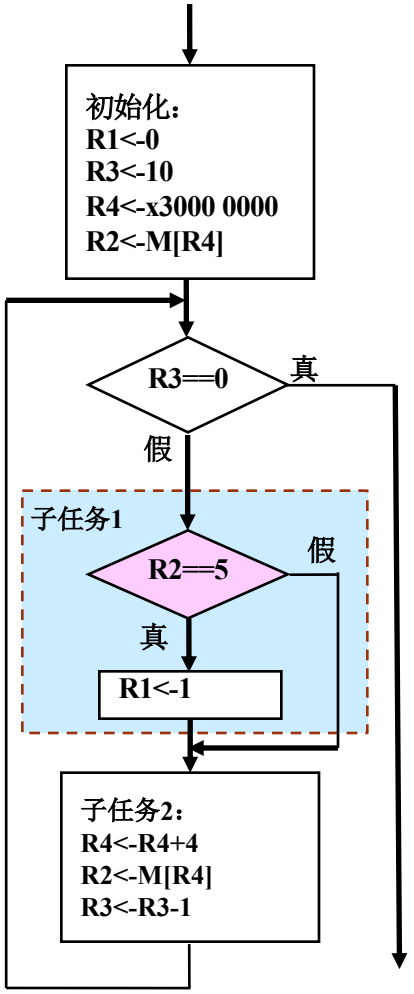
测试条件 R2==5





- 生成条件指令
  - SEQI Rx, R2, #5
- 条件分支指令
  - BEQZ Rx, D2+4

汇编语言程序




```
;
;判断10个整数中是否有5
;
;10个整数
                .data           x30000000
 numbers:       .word           #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
 ;
; 初始化
                .text           x40000000
                .global         main
 main:          addi            r4, r0, numbers   ;R4，整数地址
                addi            r1, r0, #0                        ;R1，结果
                addi            r3, r0, #10       ;R3，整数个数
                lw              r2, 0(r4)                         ;R2，整数值
 ;
;循环计算
again:          beqz            r3, exit
                seqi            r5, r2, #5
                beqz            r5, next
                addi            r1, r0, #1
                j               exit
next:           addi            r4, r4, #4        ;下一个整数地址
                subi            r3, r3, #1
                lw              r2, 0(r4)
                j               again
exit:           trap            #0
; 程序结束
```

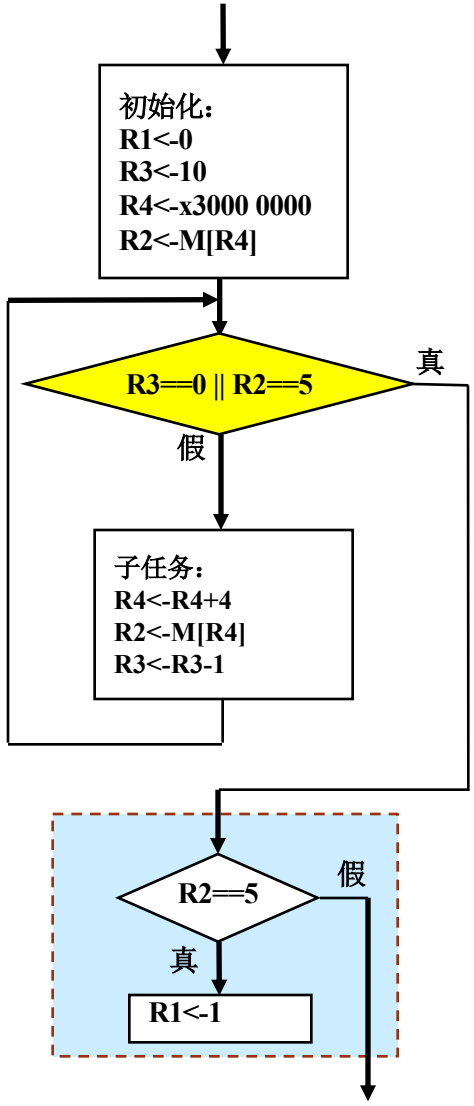
- 选择结构

  if (R2==5)

  {R1=1; break;}

- 使用J指令跳出循环

循环结束条件 R3==0 || R2==5




```
;
;判断10个整数中是否有5
;
;10个整数
                .data           x30000000
 numbers:       .word           #10, #3, #4, #6, #8, #-2, #45, #5, #8, #9
 ;
; 初始化
                .text           x40000000
                .global         main
 main:          addi            r4, r0, numbers   ;R4，整数地址
                addi            r1, r0, #0                        ;R1，结果
                addi            r3, r0, #10       ;R3，整数个数
                lw              r2, 0(r4)                         ;R2，整数值
 ;
;循环计算
again:          beqz            r3, exit
                seqi            r5, r2, #5
                bnez            r5, setR1
                addi            r4, r4, #4        ;下一个整数地址
                subi            r3, r3, #1
                lw              r2, 0(r4)
                j               again
setR1:          addi            r1, r0, #1
exit:           trap            #0
; 程序结束
```

- 汇编语言：
  - 非结构化！

多个汇编语言文件

- 数据文件(find5data.dlx)
- 单独汇编
- 方便测试

```
        ;10个整数
      .DATA
      .GLOBAL NUMBERS
NUMBERS :  .WORD  #10,  #3,  #4,  #6,  #8,  #-2,  #45,  #0,  #8,  #9
    ;代码区
      .TEXT
```

- 程序文件(find5.dlx)
- 单独汇编

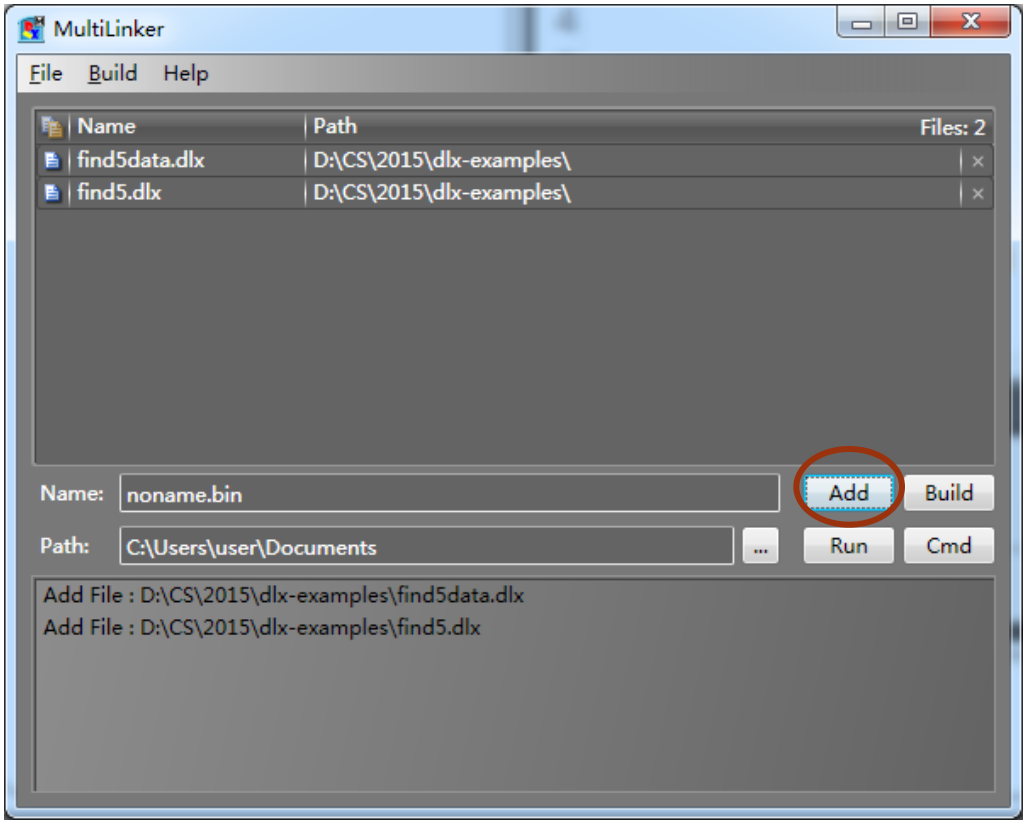
```
        ;
    ;判断10个整数中是否有5

      .DATA
    ;代码区
    ;  初始化
      .TEXT
      .GLOBAL MAIN
      .EXTERN NUMBERS
MAIN :    ADDI   R4, R0,  NUMBERS      ;R4，整数地址
......
```

链接

- Multi Link或命令行

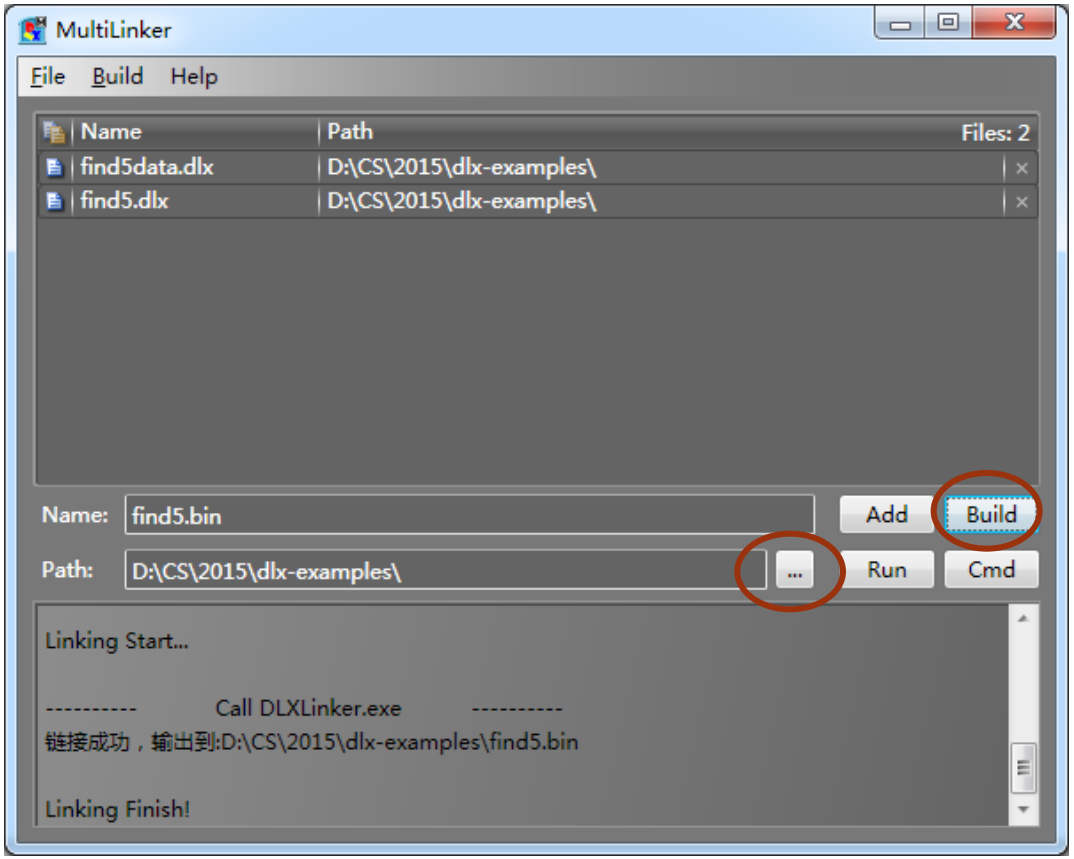
选择dlx文件

- Add
  - 选择dlx文件
  - 执行2次



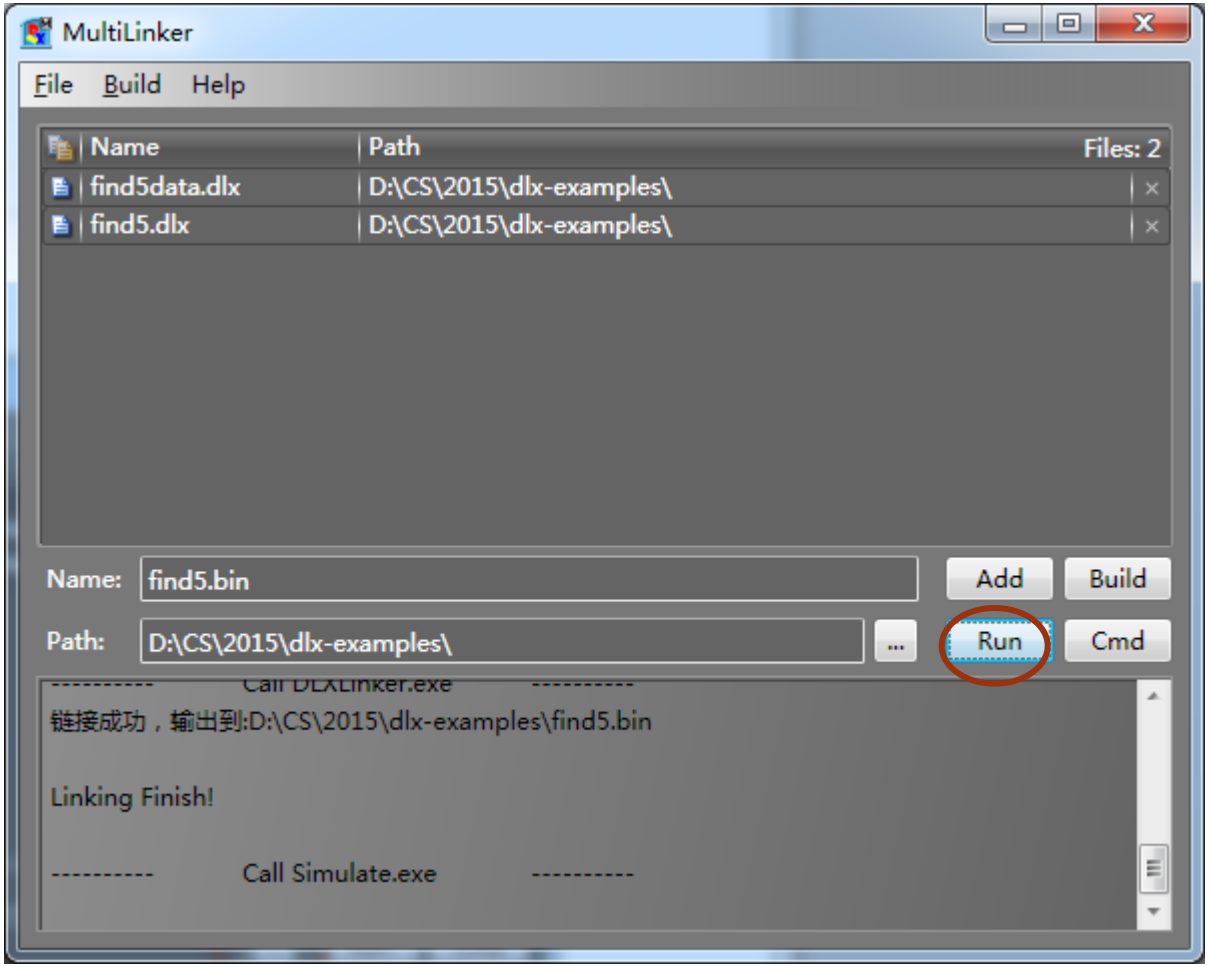
链接

- 1、选择Path和bin文件Name
- 2、Build



运行

- Run


MultiLinker
File Build Help
Name Path Files: 2
find5data.dlx D:\CS\2015\dlx-examples\
find5.dlx D:\CS\2015\dlx-examples\
Name: find5.bin
Add Build
Path: D:\CS\2015\dlx-examples\
...
Run Cmd
链接成功，输出到:D:\CS\2015\dlx-examples\find5.bin
Linking Finish!
Call Simulate.exe

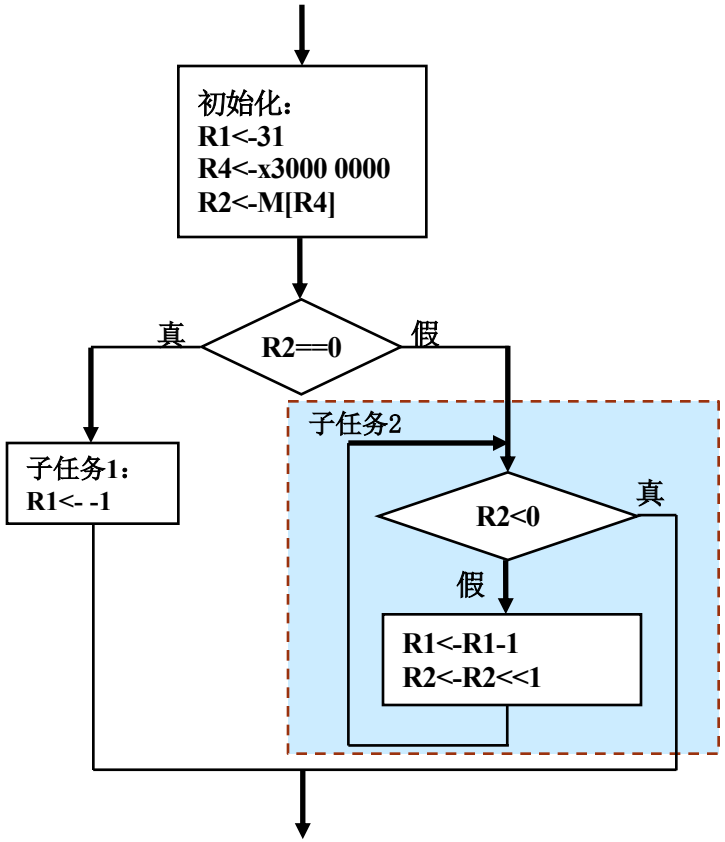
例2调试

- x3000 0000~x3000 0003中的字
- 找出第一个“1”（从左到右）
  - 存储到R1中
    - 如果没有1
      - R1 <- -1
- 使用断点跟踪程序
  - 在x4000 0024处设断点，检查每次循环的结果
  - 31, 30, ……, 0, –1, –2, …

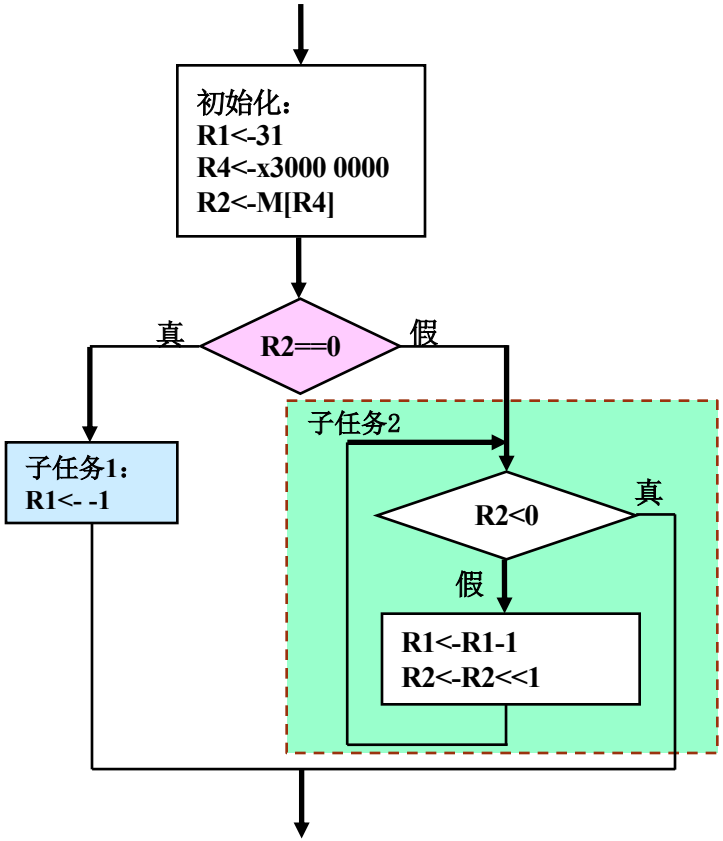
PC	R1
x4000 0024	31
x4000 0024	30
x4000 0024	29
… …	… …
x4000 0024	3
x4000 0024	2
x4000 0024	1
x4000 0024	0
x4000 0024	-1
x4000 0024	-2
x4000 0024	-3
x4000 0024	-4
… …	… …

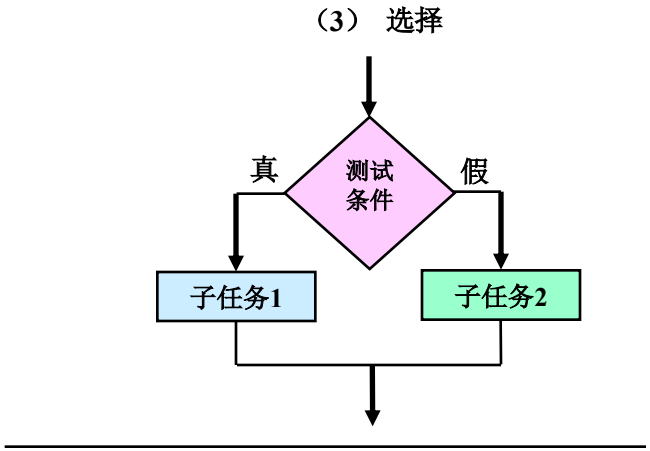
例2分析

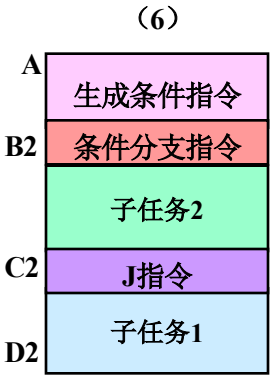
- 选择结构
- 子任务2
  - 标志控制的循环
  - 标志
    - R2<0：R2[31]=1
  - 循环子任务
    - R2=R2<<1
      - R2[30], R2[29]…==1？



测试条件 R2==0

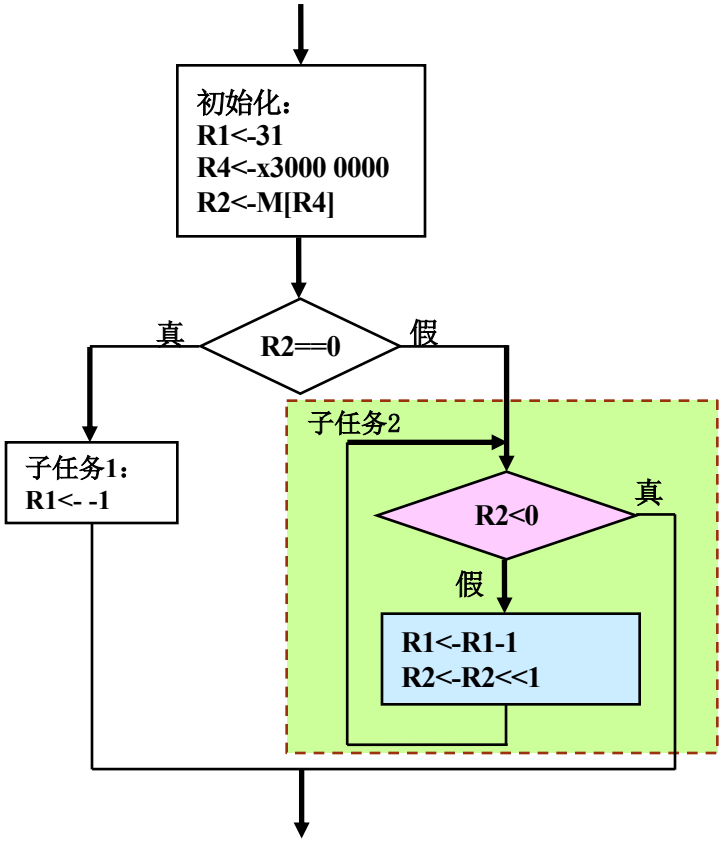


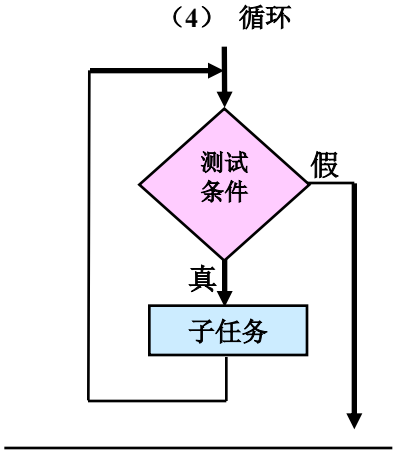


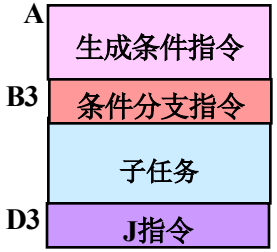


- 不需要生成条件指令
- 条件分支指令
  - BEQZ R2, C2+4

测试条件 R2<=0

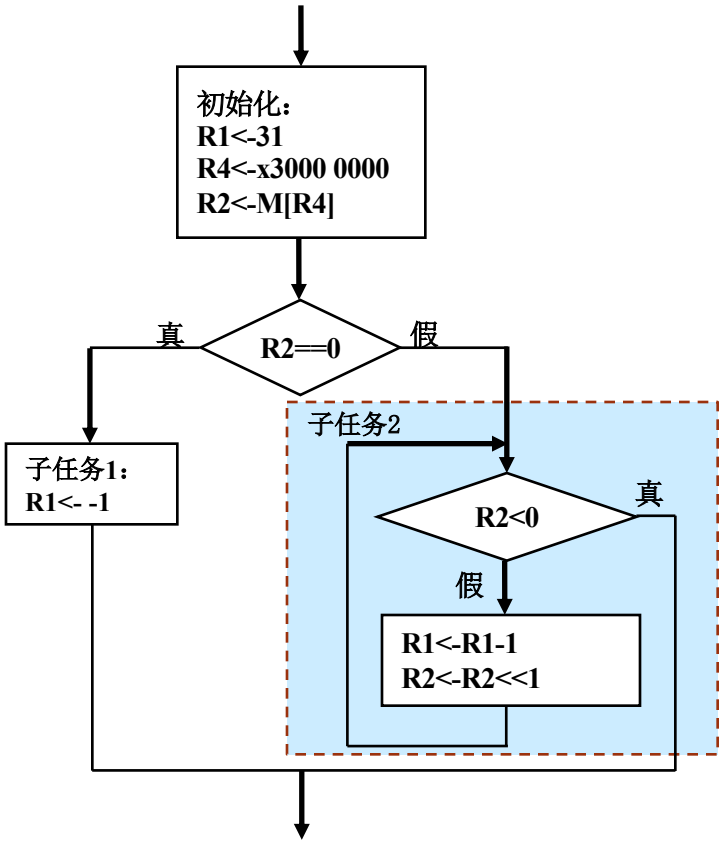






- 生成条件指令
  - SLTI Rx, R2, #0
- 条件分支指令
  - BNEZ Rx, D3+4

汇编语言程序




```
;
;判断1个数中的第一个1
;
;数据
            .data       x30000000
 numbers:   .word       x30000000
 ;
; 初始化
            .text       x40000000
            .global     main
 main:      addi        r4, r0, numbers         ;R4，数据地址
            addi        r1, r0, #31             ;R31，结果
            lw          r2, 0(r4)               ;R2，整数值
 ;
;选择
again:      beqz        r2, noone
            slti        r5, r2, #0
            bnez        r5, exit
            subi        r1, r1, #1
            slli        r2, r2, #1
            j           again
noone:      addi        r1, r0,#-1
exit:       trap        #0
; 程序结束
```

C语言程序到DLX汇编语言

- 编译器
  - 接受C程序代码，并将它们转化成能被底层硬件执行的机器代码
- C-DLX
  - 编译器必须把程序可能包含的任何运算翻译成DLX指令集中的指令——假如DLX只有很少的运算指令，这显然不是一件容易的工作

变量——寄存器

- 寄存器的访问比存储器快得多，而且DLX算术/逻辑运算指令也是对寄存器进行运算
  - 在计算机中执行时，应尽量多的使用寄存器

寄存器分配规则

- 在本章的示例中
  - 将变量分配给寄存器R16~R23
  - 将寄存器R8~R15和R24、R25用于存储临时产生的值

R0	0
R1	汇编器保留
R2、R3	返回值
R4~R7	参数
R8~R15	临时值
R16~R23	局部变量
R24、R25	临时值
R26、R27	操作系统保留
R28	全局指针
R29	栈指针
R30	帧指针
R31	返回地址

z = x * y;

- x、y和z是局部变量，且均为无符号整数
- R16：x，R17：y，R18：z
  - 本问题：R18←R16*R17
    - 注:R16最后值为0，这是有问题的

```
   addi   r18, r0, #0       ; z = 0
   slt    r8, r16, r17      ; x < y?, R8为临时变量
   bnez   r8, loop
   xor    r16, r16, r17     ; 交换x和y的值
   xor    r17, r16, r17
   xor    r16, r16, r17
loop:    add     r18, r18, r17     ; z = z + y
   subi   r16, r16, #1      ; x = x - 1
   bnez   r16, loop         ; x > 0
```

if语句

```
if (x == 1)
    y = 5;
```

- 假设x和y被声明为局部整数变量
- R16：x，R17：y
- 本问题：if (R16==1) R17←5

```
        seqi    r8, r16, #1     ; x == 1? , R8为临时变量
        beqz    r8, not_true    ; 如果条件不为真,
                                ; 那么跳过赋值
        addi    r17, r0, #5     ; y = 5
not_true:       ... ...         ; 程序的其余部分
        ... ...
```

if-else语句

- 假设x、y和z被声明为局部整数变量
- R16：x，R17：y，R18：z

```
if (x) {
   y++;
   z--;
}
else {
   y--;
   z++;
}
```

```
        beqz    r16, else       ; 如果x等于0
                                ;执行else部分
        addi    r17,r17,#1      ; y++;
        subi    r18, r18, #1    ; z--;
        j       done
else:   subi    r17, r17, #1    ; y--;
        addi    r18, r18, #1    ; z++;
done:   ... ...                 ; 程序
                                ;其余部分
        ... ...
```

while语句

- R16：i

```
int i = 0;
while (i < 10) {
    printf ("%d", i);
    i = i + 1;
}
```

```
            addi        r16, r0, #0             ; i = 0;
; while (i < 10)
loop:                   slti        r8, r16, #10            ; 进行测试，R8临时变量
            beqz        r8, done                ; i不小于10
; 循环体
... ...
; 调用函数printf的代码
... ...
            addi        r16, r16, #1            ; i = i + 1
            j           loop                    ; 再重复一次
done :      ... ...                                         ; 程序的其余部分
```

for语句

- R16：i

```
for (i = 0; i < 10; i++)
      printf ("%d", i);
```

```
          addi      r16, r0, #0          ; i = 0;
; i < 10
loop:               slti      r8, r16, #10          ; 进行测试，R8临时变量
          beqz      r8, done               ; i不小于10
; 循环体
... ...
; 调用函数printf的代码
... ...
          addi      r16, r16, #1         ; i = i + 1
          j         loop                 ; 再重复一次
done :    ... ...                                   ; 程序的其余部分
```

- 与While语句相同！

for语句

- R16：x，R17：sum

```
int x;
int sum = 0;
for (x = 0; x < 10; x++)
        sum = sum + x;
```

```
    addi    r17, r0, #0             ; sum = 0;
; 初始化
    addi    r16, r0, #0             ; 初始化(x = 0)
; 测试
loop:       slti        r8, r16, #10            ; 进行测试，R8为临时变量
    beqz    r8, done                ; x不小于10
; 循环体
    add     r17, r17, r16           ; sum = sum + x
; 重新初始化
    addi    r16, r16, #1            ; x++
    j       loop
done :      ... ...                             ; 程序的其余部分
```

总结（1）

- 高级语言支持选择结构和循环结构，但是实现这些结构的计算机底层指令都是条件分支指令

总结（2）

- 变量个数多于寄存器数目时，需要使用存储器
  - 编译器应尽量将最常用的变量保存在寄存器中，而将不常用的变量放到存储器中
  - 为了实现寄存器和存储器之间的变量交换，需要使用一种被称为“栈”的存储结构，将变量的值存储于存储器之中
    - →第十四章

习题

书面作业

- 11.16
- 11.17（考虑编译优化）