Thực hành
Kiến trúc Máy tính

TS. Lê Xuân Thành

Bộ môn KTMT

Viện CNTT&TT

Tuần 3

- Mô hình lập trình có cấu trúc
  - Tuần tự
  - Rẽ nhánh
  - Vòng lặp

Mô hình lập trình có cấu trúc

- Tuần tự: cho phép chương trình thực thi các câu lệnh theo thứ tự từng câu lệnh
- Rẽ nhánh: cho phép chương trình nhảy đến các điểm khác nhau trong một chương trình
- Vòng lặp: cho phép một chương trình thực thi một đoạn mã lệnh nhiều lần

Tuân theo nguyên tắc lập trình có cấu trúc để không tạo ra "spagetti code"

Mô hình lập trình có cấu trúc

Mô tả thuật toán

Xây dựng mã giả

Dịch sang hợp ngữ

Mã giả là gì?

- Không phải là một ngôn ngữ chính thức
- Là một tập hợp các khái niệm linh hoạt dùng để phác thảo một chương trình

- Ví dụ:

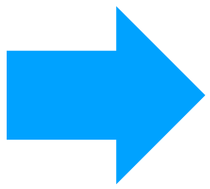
```
main
{
    register int i = input("Please enter the first value to add: ");
    register int j = input("Please enter the second value to add: ");
    register int k = i + j;
    print("The result is " + k);
}
```

Cấu trúc điều khiển

- Câu lệnh goto
- Câu lệnh if đơn giản
- Câu lệnh if-else
- Câu lệnh if-elseif-else
- Vòng lặp điều kiện
- Vòng lặp biến đếm

Sử dụng lệnh goto

- Lệnh rẽ nhánh đơn giản nhất
- Cho phép rẽ nhánh không giới hạn đến bất kỳ điểm nào trong chương trình
- Dẫn đến chương trình bị xáo trộn



Dùng lệnh rẽ nhánh goto là không tốt!

- Câu lệnh goto không phải là vấn đề
- Cơ chế cho phép các lập trình viên tạo ra vấn đề

Cấu trúc điều khiển

- Câu lệnh goto
- Câu lệnh if đơn giản
- Câu lệnh if-else
- Câu lệnh if-elseif-else
- Vòng lặp điều kiện
- Vòng lặp biến đếm

Câu lệnh if đơn giản

- Là câu lệnh if không có bất kỳ điều kiện nào khác
- Xét 3 ví dụ:
  - Ví dụ 1: Điều kiện logic đơn
  - Ví dụ 2: Điều kiện logic phức tạp
  - Ví dụ 3: Điều kiện logic phức tạp hơn

Câu lệnh if đơn giản – Ví dụ 1

- Xét chương trình:

```
if (num > 0)
{
        print("Number is positive")
}
```

- Trong đó:
  - `(num > 0)` là một lệnh trả về giá trị logic

    ```
    boolean flag = num > 0;
    if (flag) ...
    ```

  - Khối lệnh nằm trong dấu { }

Câu lệnh if đơn giản – Ví dụ 1

- Dịch sang mã hợp ngữ:

```
.data
        num: .word 5
        PositiveNumber: .asciiz "Number is positive"
.text
        # if (num > 0 )
        lw $t0, num
        sgt $t1, $t0, $zero     # $t1 is the boolean (num > 0)
        beqz $t1, end_if        # note: the code block is entered if
                                # if logical is true, skipped if false
        #{
        # print ("Number is positive")
        la $a0, PositiveNumber
        li $v0, 4
        syscall
        #}
        end_if:

        li $v0, 10
        syscall
```

Câu lệnh if đơn giản – Ví dụ 2

- Xét điều kiện sau:

```
if ((x > 0 && ((x%2) == 0))  # is x > 0 and even?
```

- Tương đương với:

```
boolean flag = ((x > 0) && ((x%2) == 0))
if (flag)...
```

- Giả ngữ tương đương:

```
lw $t0, x
sgt $t1, $t0, $zero
rem $t2, $t0, 2
and $t1, $t1, $t2
beqz $t1, end_if
```

Câu lệnh if đơn giản – Ví dụ 3

- Xét điều kiện sau:

```
if ((x > 0) && ((x%2) == 0) && (x < 10))  # is 0 < x < 10 and even?
```

- Giả ngữ tương đương với:

```
lw $t0, x
sgt $t1, $t0, $zero
li $t5, 10
slt $t2, $t0, $t5
rem $t3, $t0, 2
and $t1, $t1, $t2
and $t1, $t1, $t3
beqz $t1, end_if
```

Cấu trúc điều khiển

- Câu lệnh goto
- Câu lệnh if đơn giản
- Câu lệnh if-else
- Câu lệnh if-elseif-else
- Vòng lặp điều kiện
- Vòng lặp biến đếm

Câu lệnh if-else

- Xét chương trình:

```
if (($s0 > 0) == 0)
{
    print("Number is positive")
}
else
{
    print("Number is negative")

}
```

Câu lệnh if-else

1. Thực hiện phần điều kiện của câu lệnh
2. Thêm vào 2 nhãn cho chương trình:
   - 1 nhãn cho **else**
   - 1 nhãn cho kết thúc lệnh **if**
3. Lệnh **beqz** nên được thêm vào sau khi đánh giá điều kiện rẽ nhánh tới nhãn **else**.

Câu lệnh if-else

3. Ở cuối của khối **if**, rẽ nhánh qua khối **else** bằng cách sử dụng câu lệnh không điều kiện tới **endif**.

```
lw $t0, num
sgt $t1, $t0, $zero
beqz $t1, else
#if block
b end_if
#else block
else:
end_if:
```

Câu lệnh if-else

4. Ngay khi cấu trúc câu lệnh **if-else** được thiết lập đúng, đặt khối lệnh vào trong cấu trúc chương trình:

```
.data
        num: .word -5
        PositiveNumber: .asciiz "Number is positive"
        NegativeNumber: .asciiz "Number is negative"
.text
        lw $t0, num
        sgt $t1, $t0, $zero
        beqz $t1, else
                #if block
                la $a0, PositiveNumber
                li $v0, 4
                syscall
                b end_if
                #else block
        else:
                la $a0, NegativeNumber
                li $v0, 4
                syscall
end_if:

li $v0, 10
syscall
```

Cấu trúc điều khiển

- Câu lệnh goto
- Câu lệnh if đơn giản
- Câu lệnh if-else
- Câu lệnh if-elseif-else
- Vòng lặp điều kiện
- Vòng lặp biến đếm

Câu lệnh if-elseif-else

- Xét chương trình:

```
if (grade > 100) || grade < 0)
{
     print("Grade must be between 0..100")
}
elseif (grade >= 90)
{
     print("Grade is A")
}
elseif (grade >= 80)
{
     print("Grade is B")
}
elseif (grade >= 70)
{
     print("Grade is C")
}
elseif (grade >= 60)
{
     print("Grade is D")
}
else{
     print("Grade is F")
}
```

Câu lệnh if-elseif-else

1. Bắt đầu thực hiện lệnh bằng 1 chú thích và đặt 1 nhãn trong khối mã cho mỗi điều kiện:

   - elseif
   - lệnh else cuối cùng
   - các điều kiện else_if

Ở cuối mỗi khối lệnh, đặt một nhãn rẽ nhánh tới lệnh else_if (ngay khi bất kỳ một khối lệnh nào được thực thi thì sẽ thoát ra khỏi toàn bộ câu lệnh if-elseif-else).

Câu lệnh if-elseif-else

Khối lệnh được thực hiện sẽ như sau:

```
#if block
    # first if check, invalid input block
    b end_if
grade_A:
    b end_if
grade_B:
    b end_if
grade_C:
    b end_if
grade_D:
    b end_if
else:
    b end_if
end_if:
```

Câu lệnh if-elseif-else

2. Đặt các điều kiện logic vào đầu mỗi khối **if** và **elseif**. Lệnh sẽ được rẽ nhánh tới nhãn tiếp theo.

```
#if block
    lw $s0, num
    slti $t1, $s0, 0
    sgt $t2, $s0, 100
    or $t1, $t1, $t2
    beqz $t1, grade_A
    #invalid input block
    b end_if
grade_A:
    sge $t1, $s0, 90
    beqz $t1, grade_B
    b end_if
grade_B:
    sge $t1, $s0, 80
    beqz $t1, grade_C
    b end_if
grade_C:
    sge $t1, $s0, 70
    beqz $t1, grade_D
    b end_if
grade_D:
    sge $t1, $s0, 60
    beqz $t1, else
    b end_if
else:
    b end_if
end_if:
```

Câu lệnh if-elseif-else

3. Bước cuối cùng là điền vào khối lệnh logic thích hợp.

```
.data
        num: .word 70
        InvalidInput: .asciiz "Number must be > 0 and < 100"
        OutputA: .asciiz "Grade is A"
        OutputB: .asciiz "Grade is B"
        OutputC: .asciiz "Grade is C"
        OutputD: .asciiz "Grade is D"
        OutputF: .asciiz "Grade is F"

.text
    #if block
        lw $s0, num
        slti $t1, $s0, 0
        sgt $t2, $s0, 100
        or $t1, $t1, $t2
        beqz $t1, grade_A
        #invalid input block
        la $a0, InvalidInput
        li $v0, 4
        syscall
        b end_if
    grade_A:
        sge $t1, $s0, 90
        beqz $t1, grade_B
        la $a0, OutputA
        li $v0, 4
        syscall
        b end_if
    grade_B:
        sge $t1, $s0, 80
        beqz $t1, grade_C
        la $a0, OutputB
        li $v0, 4
        syscall
        b end_if
    grade_C:
        sge $t1, $s0, 70
        beqz $t1, grade_D
        la $a0, OutputC
        li $v0, 4
        syscall
        b end_if
    grade_D:
        sge $t1, $s0, 60
        beqz $t1, else
        la $a0, OutputD
        li $v0, 4
        syscall
        b end_if
    else:
        la $a0, OutputF
        li $v0, 4
        syscall
        b end_if
    end_if:
li $v0, 10
syscall
```

Cấu trúc điều khiển

- Câu lệnh goto
- Câu lệnh if đơn giản
- Câu lệnh if-else
- Câu lệnh if-elseif-else
- Vòng lặp điều kiện
- Vòng lặp biến đếm

Vòng lặp điều kiện

- Xét chương trình:

```
int i = prompt("Enter an integer, or -1 to exit")
while (i != -1)
{
    print("You entered " + i);
    i = prompt("Enter an integer, or -1 to exit");
}
```

Vòng lặp điều kiện

1. Kiểm tra điều kiện trước khi vào vòng lặp

2. Tạo một nhãn bắt đầu vòng lặp để chương trình có thể quay về đầu vòng lặp khi đến cuối vòng lặp.

3. Tạo một nhãn kết thúc vòng lặp để vòng lặp có thể nhảy ra ngoài khi điều kiện không thoả mãn.

4. Đặt lệnh kiểm tra điều kiện thoả mãn. Nếu đúng, nhảy về nhãn kết thúc vòng lặp.

5. Đặt điều kiện kiểm tra ở dòng lệnh cuối cùng trong khối lệnh cho vòng lặp, và lệnh rẽ nhánh không điều kiện quay trở về khởi đầu vòng lặp để kết thúc.

Vòng lặp điều kiện

Kết thúc có đoạn lệnh như sau:

```
.data
        prompt: .asciiz "Enter an integer, -1 to stop: "

.text
        #set sentinel value (prompt the user for input)
        la $a0, prompt
        li $v0, 4
        syscall
        move $a0, $a1
        li $v0, 5
        syscall
        #return
        #jr $ra
        move $s0, $v0
        start_loop:
                sne $t1, $s0, -1
                beqz $t1, end_loop
                # code block
                la $a0, prompt
                li $v0, 4
                syscall
                move $a0, $a1
                li $v0, 5
                syscall
                move $s0, $v0
                b start_loop
        end_loop:
```

Vòng lặp điều kiện

6. Cấu trúc cần thiết cho vòng lặp điều kiện đã sẵn sàng. Có thể thêm các phép logic và bất kỳ lệnh cần thiết để kết thúc chương trình.

Vòng lặp điều kiện

6. Chương trình:

```
.data
        prompt: .asciiz "Enter an integer, -1 to stop: "
        output: .asciiz "\nYou entered: "

.text
        #set sentinel value (prompt the user for input)
        la $a0, prompt
        li $v0, 4
        syscall
        move $a0, $a1
        li $v0, 5
        syscall
        #return
        #jr $ra
        move $s0, $v0
        start_loop:
                sne $t1, $s0, -1
                beqz $t1, end_loop
                # code block
                la $a0, output
                move $a1, $s0
                li $v0, 4
                syscall
                # Print integer
                move $a0, $a1
                li $v0, 1
                syscall

                la $a0, prompt
                li $v0, 4
                syscall
                move $a0, $a1
                li $v0, 5
                syscall
                move $s0, $v0
                b start_loop
        end_loop:
```

Cấu trúc điều khiển

- Câu lệnh goto
- Câu lệnh if đơn giản
- Câu lệnh if-else
- Câu lệnh if-elseif-else
- Vòng lặp điều kiện
- Vòng lặp biến đếm

Vòng lặp biến đếm

- Xét chương trình:

```
n = prompt("enter the value to calculate the sum up to: ")
total = 0; # Initial the total variable for sum
for (i = 0; i < n; i++)
{
    total = total + i
}
print("Total = " + total);
```

Vòng lặp biến đếm

1. Thực hiện bước khởi tạo để khởi tạo bộ đếm và các biến điều kiện kết thúc.

2. Tạo ra các nhãn để bắt đầu và kết thúc vòng lặp.

3. Thực hiện kiểm tra để vào khối lặp, hoặc kết thúc khối lặp nếu điều kiện thoả mãn.

4. Thực hiện tăng bộ đếm, và rẽ nhánh ngược trở lại nơi bắt đầu vòng lặp.

Vòng lặp biến đếm

- Được khối lệnh như sau:

```
.data
        n: .word 5

.text
li $s0, 0
lw $s1, n
start_loop:
        sle $t1, $s0, $s1
        beqz $t1, end_loop
        # code block
        addi $s0, $s0, 1
        b start_loop
end_loop:
```

Vòng lặp biến đếm

5. Thực hiện khối lệnh cho câu lệnh for.

```
.data
        prompt: .asciiz "enter the value to calculate the sum up to: "
        output: .asciiz "The final result is: "

.text
        la $a0, prompt
        li $v0, 4
        syscall
        move $a0, $a1
        li $v0, 5
        syscall
        move $s1, $v0
        li $s0, 0
        li $s2, 0 # Initialize the total
        start_loop:
              sle $t1, $s0, $s1
              beqz $t1, end_loop
              # code block
              add $s2, $s2, $s0
              addi $s0, $s0, 1
              b start_loop
        end_loop:
        la $a0, output
        move $a1, $s2
        li $v0, 4
        syscall
        move $a0, $a1
        li $v0, 1
        syscall

li $v0,10
syscall
```

Các khối lệnh lồng nhau

- Xét chương trình:

```
int n = prompt("Enter a value for the summation n, -1 to stop");
while (n != -1)
{
    if (n < -1)
    {
        print("Negative input is invalid");
    }
    else
    {
        int total = 0
        for (int i = 0; i < n; i++)
        {
            total = total + i;
        }
        print("The summation is " + total);
    }
}
```

Các khối lệnh lồng nhau

1. Bắt đầu bằng việc thực hiện khối ngoài cùng - khối lặp điều kiện.

```
# Sentinel Control Loop
.data
        prompt: .asciiz "Enter an integer, -1 to stop: "

.text
        la $a0, prompt
li $v0, 4
syscall
move $a0, $a1
li $v0, 5
syscall
move $s0, $v0
start_outer_loop:
        sne $t1, $s0, -1
        beqz $t1, end_outer_loop

        # code block

        la $a0, prompt
        li $v0, 4
        syscall
        move $a0, $a1
        li $v0, 5
        syscall
        move $s0, $v0
        b start_outer_loop
end_outer_loop:
```

Các khối lệnh lồng nhau

2. Khối mã trong vòng lặp điều kiện ở trên được thay bằng câu lệnh điều kiện **if-else** để kiểm tra giá trị đầu vào.

```
# Sentinel Control Loop
.data
        prompt: .asciiz "Enter an integer, -1 to stop: "

.text
        la $a0, prompt
        li $v0, 4
        syscall
        move $a0, $a1
        li $v0, 5
        syscall
        move $s0, $v0
        start_outer_loop:
        sne $t1, $s0, -1
        beqz $t1, end_outer_loop

        # If test for valid input
        slti $t1, $s0, -1
        beqz $t1, else
             #if block
             b end_if
        else:
             #else block
        end_if:

        la $a0, prompt
        li $v0, 4
        syscall
        move $a0, $a1
        li $v0, 5
        syscall
        move $s0, $v0
        b start_outer_loop
end_outer_loop:
```

Các khối lệnh lồng nhau

3. Khối **if** trong đoạn mã ở trên được thay thế bởi thông báo lỗi, và khối **else** được thay thế bởi vòng lặp điều kiện.

Các khối lệnh lồng nhau

```
# Sentinel Control Loop
.data
        prompt: .asciiz "\nEnter an integer, -1 to stop: "
        error: .asciiz  "\nValues for n must be > 0"
        output: .asciiz "\nThe total is: "

.text
        la $a0, prompt
        li $v0, 4
        syscall
        move $a0, $a1
        li $v0, 5
        syscall
        move $s0, $v0
        start_outer_loop:
        sne $t1, $s0, -1
        beqz $t1, end_outer_loop
                # If test for valid input
                slti $t1, $s0, -1
                beqz $t1, else
                la $a0, error
                li $v0,4
                syscall
                b end_if
         else:
                # summation loop
                li $s1, 0
                li $s2, 0 # initialize total
                start_inner_loop:
                sle $t1, $s1, $s0
                beqz $t1, end_inner_loop

                add $s2, $s2, $s1

                addi $s1, $s1, 1
                b start_inner_loop
                end_inner_loop:
                la $a0, output
                move $a1, $s2
                li $v0, 4
                syscall
                move $a0, $a1
                li $v0, 1
                syscall
        end_if:
        la $a0, prompt
        li $v0, 4
        syscall
        move $a0, $a1
        li $v0, 5
        syscall
        move $s0, $v0
        b start_outer_loop
        end_outer_loop:
        li $v0,10
        syscall
```

Chương trình đầy đủ

1. Viết ra mã hợp ngữ:

   - Cho phép suy luận ở mức độ cao hơn
   - Triển khai mã dễ dàng hơn vì nó có thể dịch thẳng từ mã giả sang hợp ngữ.
   - Là tài liệu mô tả cách thức chương trình làm việc, nên được thêm vào phần chú thích ở đầu mỗi chương trình.

Chương trình đầy đủ

2. Chú thích mở đầu chứa các thông tin:

- Tên tệp tin
- Tác giả
- Ngày tạo
- Mục đích
- Lịch sử sửa chữa
- Mã giả

```
# Filename: AverageGrade.asm
# Author:   Charles Kann
# Date:           12/29/2013
# Purpose:  Illustration of program to calculate a student grade
# Modification Log:
#     12/29/2014 - Initial release
#
# Pseudo Code
#global main()
#{
#    // The following variables are to be stored in data segment, and
#    // not simply used from a register.  They must be read each time
#    // they are used, and saved when they are changed.
#    static volatile int numberOfEntries = 0
#    static volatile int total = 0
#
#    // The following variable can be kept in a save register.
#    register int inputGrade  # input grade from the user
#    register int average
#
#    // Sentinel loop to get grades, calculate total.
#    inputGrade = prompt("Enter grade, or -1 when done")
#    while (inputGrade != -1)
#    {
#        numberOfEntries = numberOfEntries + 1
#        total = total + inputGrade
#        inputGrade = prompt("Enter grade, or -1 when done")
#    }
#
#    # Calculate average
#    average = total / numberOfEntries
#
#    // Print average
#    print("Average = " + average)
#
#    //Print grade if average is between 0 and 100, otherwise an error
#    if ((grade >= 0) & (grade <= 100))
#    {
```

```
#            if (grade >= 90)
#            {
#                print("Grade is A")
#            }
#            if (grade >= 80)
#            {
#                print("Grade is B")
#            }
#            if (grade >= 70)
#            {
#                print("Grade is C")
#            }
#            else
#            {
#                print("Grade is F")
#            }
#        }
#        else
#        {
#            print("The average is invalid")
#        }
#}
```

```
.data
    numberOfEntries: .word 0
    total:           .word 0
    average:         .word
    prompt:          .asciiz "Enter grade, or -1 when done: "
    avgOutput:       .asciiz "The average is "
    gradeA:          .asciiz "The grades is an A"
    gradeB:          .asciiz "The grade is a B"
    gradeC:          .asciiz "The grade is a C"
    gradeF:          .asciiz "The grade is a F"
    invalidAvg:       .asciiz "The average is invalid"


  .text
  .globl main
  main:
      # Register Conventions:
      #    $s0 - current inputGrade
      #    $s1 - average
      la $a0, prompt
      jal PromptInt
      move $s0, $v0

      BeginInputLoop:
addi $t0, $zero, -1       # set condition $s0 != -1
seq  $t0, $t0, $s0
xor  $t0, $t0, 0x00000001
beqz $t0, EndInputLoop    # check condition to end loop

  la $t0, numberOfEntries # increment # of entries
  lw $t1, 0($t0)
  addi $t1, $t1, 1
  sw $t1, 0($t0)

  la $t0, total           # accumulate total
  lw $t1, 0($t0)
  add $t1, $t1, $s0
  sw $t1, 0($t0)

  la $a0, prompt          # prompt for next input
  jal PromptInt
  move $s0, $v0
  b BeginInputLoop
EndInputLoop:

la $t0, numberOfEntries   #Calculate Average
lw $t1, 0($t0)
```

```
la $t0, total
lw $t2, 0($t0)
div $s1, $t2, $t1

la $a0, avgOutput          # Print the average
move $a1, $s1
jal PrintInt
jal NewLine


sge $t0, $s1, 0            # Set the condition
                           #(average > 0) & (average < 100)
addi $t1, $zero, 100
sle $t1, $s1, $t1
and $t0, $t0, $t1
beqz $t0, AverageError     # if Not AverageError
    sge $t0, $s1, 90       # PrintGrades
    beqz $t0, NotA
        la $a0, gradeA
        jal PrintString
        b EndPrintGrades
    NotA:
        sge $t0, $s1, 80
        beqz $t0, NotB
        la $a0, gradeB
        jal PrintString
        b EndPrintGrades
    NotB:
        seq $t0, $s1, 70
        beqz $t0, NotC
        la $a0, NotC
        la $a0, gradeC
        jal PrintString
        b EndPrintGrades
    NotC:
        la $a0, gradeF
        jal PrintString
    EndPrintGrades:
    b EndAverageError
AverageError:                  #else AverageError
    la $a0, invalidAvg
    jal PrintString
EndAverageError:

jal Exit
```