

Thực hành Kiến trúc máy tính

Giảng viên: Nguyễn Thị Thanh Nga
Khoa Kỹ thuật máy tính
Trường CNTT&TT

Tuần 6

- Các lệnh điều khiển trong chương trình hợp ngữ, bao gồm lệnh rẽ nhánh và lệnh nhảy.
- Các cấu trúc điều khiển đơn giản trong lập trình có cấu trúc và cách dịch chúng sang mã hợp ngữ:
 - a) Câu lệnh if
 - b) Câu lệnh if-else
 - c) Câu lệnh if-elseif-else
 - d) Vòng lặp điều kiện
 - e) Vòng lặp biến đếm
 - f) Các khối lệnh lồng nhau

Các lệnh điều khiển

- Các lệnh điều khiển được sử dụng nếu không thực thi giá trị lưu trong thanh ghi **\$pc** tiếp theo mà chuyển sang một không gian lệnh khác.
- Có hai nhóm lệnh:
 - Rẽ nhánh
 - Nhảy

Các lệnh điều khiển

- Rẽ nhánh
 - Là lệnh nhảy có điều kiện
 - Địa chỉ đích là địa chỉ gần với giá trị trong \$pc hiện tại, khoảng cách từ \$pc hiện tại đến địa chỉ đích tương ứng với giá trị tức thì (16 bit).
 - Ví dụ: vòng lặp, các câu lệnh if

Các lệnh điều khiển

- Nhảy
 - Là lệnh nhảy không có điều kiện
 - Địa chỉ đích là địa chỉ xa với giá trị trong **\$pc** hiện tại.
 - Ví dụ: lời gọi các chương trình con

Lệnh rẽ nhánh trong MIPS

Điều khiển rẽ nhánh có điều kiện, dựa trên:

- So sánh:

- Giá trị bằng hoặc không bằng của hai thanh ghi

Toán tử R_s , R_t , đích

R_s , R_t : hai thanh ghi để so sánh

đích: đích rẽ nhánh

- Giá trị $>$, $<$, \geq , \leq của một thanh ghi với 0

Toán tử R_s , đích

R_s : thanh ghi để so sánh

đích: đích rẽ nhánh

- Rẽ nhánh: đến một đích chính là dịch chuyển một giá trị chênh lệch không dấu (biểu thị bằng số lệnh) từ lệnh sau lệnh rẽ nhánh

Lệnh rẽ nhánh trong MIPS

- Ví dụ:

beq \$t0, \$t1, Target	# branch to Target if \$t0 == \$t1
bgez \$t0, Target	# branch to Target if \$t0 ≥ 0

Lệnh rẽ nhánh trong MIPS

- Các toán tử rẽ nhánh có điều kiện:
 - beq (branch if equal)
 - bne (branch if not equal)
 - bgtz (branch if greater than zero)
 - bltz (branch if less than zero)
 - bgez (branch if greater than or equal to zero)
 - blez (branch if less than or equal to zero)
- Sử dụng slt cho các so sánh $>$, $<$, \geq , \leq giữa hai thanh ghi

slt R_d, R_s, R_t #if $R_s < R_t$, $R_d=1$; else $R_d=0$

Lệnh rẽ nhánh trong MIPS

- Ví dụ:

slt \$t0, \$t1, \$t2 # \$t0=1 if \$t1,\$t2 else \$t0=0

bne \$t0, \$0, L1 # branch to L1 if \$t0=1

Khoảng cách rẽ nhánh

Mở rộng dịch chuyển của địa chỉ đích rẽ nhánh:

- Là một giá trị offset có dấu 16 bit, biểu diễn số lệnh, không phải số bytes
- Được cộng thêm vào giá trị của \$pc
- Địa chỉ đích là một địa chỉ dưới dạng word, không phải là địa chỉ dưới dạng byte, hai bit cuối cùng là 0
- Trong hợp ngữ, sử dụng một địa chỉ đích tượng trưng

Khoảng cách rẽ nhánh

- Khoảng cách rẽ nhánh có kích thước vừa phải:
 - Giá trị offset 16 bit
 - Được cộng vào giá trị của \$pc
 - Biểu diễn bằng một địa chỉ dạng word
- Tuy nhiên, nếu khoảng cách rẽ nhánh quá nhỏ thì:
 - Trình biên dịch chèn vào một lệnh nhảy không có điều kiện
 - Rẽ nhánh điều kiện chuyển tới nhánh ứng với điều kiện sai hoặc chuyển sang lệnh nhảy luôn.

Khoảng cách rẽ nhánh

- Ví dụ:

beq \$s0, \$s1, L1

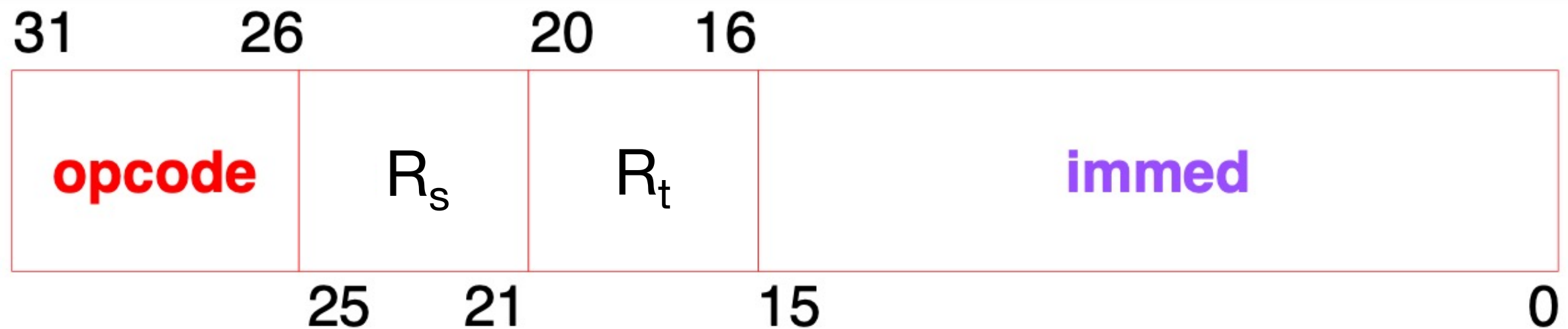
Chuyển thành:

bne \$s0, \$s1, L2

j L1

#L2: nhánh ứng với điều kiện sai

Định dạng rẽ nhánh với lệnh I



- **opcode** = Lệnh điều khiển
- R_s, R_t = Toán hạng nguồn
- **immed** = Địa chỉ offset 16-bit, bằng word, $\pm 2^{15}$
 - Địa chỉ đích = địa chỉ trong \$pc + **immed** * 4

Còn gọi là **địa chỉ PC-tương đối**

bne \$s0, \$s1, Exit

5	16	17	(Exit - PC+4) / 4
---	----	----	-------------------

Lệnh nhảy trong MIPS

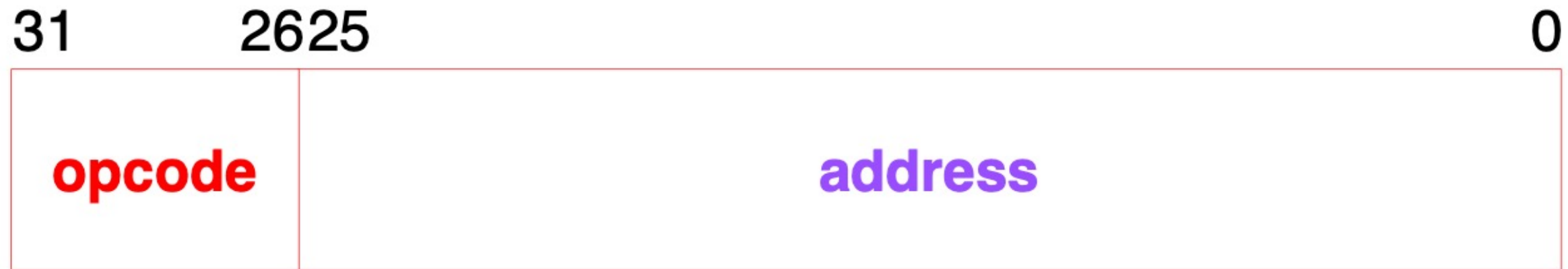
Lệnh nhảy: chuyển điều khiển không điều kiện

j target # Nhảy tới một địa chỉ đích xác định

jr rs # Nhảy tới thanh ghi
Đến địa chỉ lưu trong R_s
(còn gọi là nhảy gián tiếp)

jal target # jump and link
Nhảy tới địa chỉ đích $\$pc+4$ lưu trong $\$ra$

Định dạng lệnh nhảy J



- **opcode** = Lệnh chuyển dữ liệu
- **address** = Phần địa chỉ ở định dạng word

j 10000

2	10000
---	-------

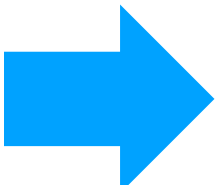
Các cấu trúc điều khiển trong lập trình có cấu trúc

Có 3 cấu trúc điều khiển, bao gồm:

- Tuần tự
- Rẽ nhánh
- Vòng lặp

Các cấu trúc điều khiển trong 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



Cần tuân theo nguyên tắc lập trình có cấu trúc và không cố gắng viết chương trình trực tiếp trong hợp ngữ.

Nguyên lý 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 thô được sử dụng để tạo ra phác thảo của một chương trình.
- Bản thân ngôn ngữ chỉ bao gồm đủ chi tiết để cho phép một lập trình viên hiểu những gì cần phải làm.
- Có thể dễ dàng thay đổi khi nhu cầu của lập trình viên thay đổi.

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);
}
```

Mã giả là gì?

- Ví dụ: hãy xem xét chương trình mã giả sau để đọc vào hai số, thực hiện phép cộng và in kết quả trả lại cho người dùng.

```
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);
}
```

- Tạo một chương trình sẽ chứa 3 giá trị nguyên.
- Việc sử dụng khai báo register báo cho người lập trình biết rằng họ nên sử dụng thanh ghi lưu (**\$s0 ... \$s7**) nếu có thể để chứa các giá trị này.

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
- Các khối mã lồng nhau

Câu lệnh goto

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

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")
}
```

- Từ chương trình đơn giản này, có rất nhiều vấn đề phức tạp ẩn chứa trong đó:
 - Biến **num** sẽ không thể được sử dụng trực tiếp trong chương trình và sẽ phải được nạp vào một thanh ghi.
 - Chương trình con để in phải được tạo ra.

Câu lệnh if đơn giản – Ví dụ 1

- Một số điều kiện quan trọng để hiểu câu lệnh if:
 - Câu lệnh (num>0) là một câu lệnh trong đó toán tử ">" trả về một giá trị logic (boolean).
 - Giá trị của biến boolean sẽ phải được tính toán trước khi nó được sử dụng trong hợp ngữ.
 - Tất cả các giá trị boolean sẽ bằng 0 (nếu sai) và 1 (nếu đúng).
- Bất kỳ mã nào trong dấu ngoặc "{" và "}" được coi là một phần của đoạn mã.

Câu lệnh if đơn giản – Ví dụ 1

- Dịch sang hợp ngữ:

```
1  .data
2      num: .word 5
3      PositiveNumber: .asciiz "Number is positive"
4  .text
5      # if (num > 0 )
6      lw $t0, num
7      sgt $t1, $t0, $zero      # $t1 is the boolean (num > 0)
8      beqz $t1, end_if        # note: the code block is entered if
9                              # if logical is true, skipped if false.
10     #{
11     # print ("Number is positive")
12     la $a0, PositiveNumber
13     jal PrintString
14     #}
15     end_if:
16     jal Exit
17
18  .include "utils.asm"
```

sgt: set greater than
if \$t0>0 then set \$t1=1 or 0

- Để biên dịch lệnh kiểm tra (**num > 0**) cần 2 lệnh hợp ngữ.

- Đầu tiên tải num vào \$t0 để các giá trị có thể được sử dụng.
- Lệnh **sgt \$t1, \$t0, \$zero** tải giá trị boolean vào \$t1 để nó có thể được so sánh trong phép thử if.

Câu lệnh if đơn giản – Ví dụ 1

- Dịch sang hợp ngữ:

```
1  .data
2      num: .word 5
3      PositiveNumber: .asciiz "Number is positive"
4  .text
5      # if (num > 0 )
6      lw $t0, num
7      sgt $t1, $t0, $zero      # $t1 is the boolean (num > 0)
8      beqz $t1, end_if        # note: the code block is entered if
9                              # if logical is true, skipped if false.
10     #{
11     # print ("Number is positive")
12     la $a0, PositiveNumber
13     jal PrintString
14     #}
15     end_if:
16     jal Exit
17
18 .include "utils.asm"
```

beqz: branch if equal zero
if true (\$t1=1), enter code block
If false (\$t1=0), jump to end_if

- Lệnh if hoạt động bằng cách kiểm tra giá trị boolean (nằm trong **\$t1**).

- Nếu giá trị boolean là đúng, thì sẽ tiếp tục với khối lệnh sau beqz (không rẽ nhánh).

- Nếu sai, rẽ nhánh đến cuối khối lệnh, và do đó sẽ không đi vào khối lệnh sau beqz.

Câu lệnh if đơn giản – Ví dụ 1

- Dịch sang hợp ngữ:

```
1  .data
2      num: .word 5
3      PositiveNumber: .asciiz "Number
4  .text
5      # if (num > 0 )
6      lw $t0, num
7      sgt $t1, $t0, $zero      # $t1 i
8      beqz $t1, end_if        # note:
9                              # if lo
10
11      #{
12      # print ("Number is positive
13      la $a0, PositiveNumber
14      jal PrintString
15      #}
16      end_if:
17      jal Exit
18  .include "utils.asm"
```

- Khi thực thi một khối lệnh, ngoặc nhọn **}** cuối cùng của khối lệnh được chuyển thành một nhãn (**end_if**). Do đó, câu lệnh **if** đơn giản ở trên được thực hiện như sau:

- Tính toán giá trị boolean để xem có vào khối lệnh **if** hay không.
- Vào khối lệnh **if** nếu giá trị boolean đúng hoặc rẽ nhánh nếu giá trị boolean sai.

Câu lệnh if đơn giản – Ví dụ 2

Với các điều kiện logic phức tạp

- Xét điều kiện sau:

```
if ( (x > 0 && (x%2) == 0) )
```

is x > 0 and even?

- Trong một ngôn ngữ lập trình bậc cao, trình biên dịch sẽ chuyển các điều kiện logic phức tạp thành phương trình duy nhất. Vì vậy, câu lệnh **if** phức tạp ở trên sẽ được chuyển thành tương đương với đoạn mã sau:

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

Câu lệnh if đơn giản – Ví dụ 2

Với các điều kiện logic phức tạp

- Đoạn mã:

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

- Được biên dịch thành mã hợp ngữ như sau:

```
lw  $t0,x  
sgt  $t1,$t0,$zero  
rem  $t2,$t0,2  
not  $t2,$t2  
and  $t1,$t1,$t2  
beqz $t1,end_if
```

Câu lệnh if đơn giản – Ví dụ 3

Với các điều kiện logic phức tạp hơn

- Xét điều kiện sau:

```
if ( (x > 0) && ( (x%2) == 0) && (x < 10) )
```

```
# is 0 < x < 10 and even?
```

- Sức mạnh thực sự của phương pháp xử lý các điều kiện logic này trở nên rõ ràng khi các điều kiện logic trở nên phức tạp hơn.

```
boolean flag = ( (x > 0) && ( (x%2) == 0) && (x < 10) )  
if (flag) ...
```


Câu lệnh if đơn giản – Ví dụ 3

Với các điều kiện logic phức tạp hơn

- Đoạn mã:

```
boolean flag = ((x > 0) && ((x%2) == 0) && (x < 10))  
if (flag)...
```

- Được biên dịch thành mã hợp ngữ như sau:

```
lw $t0,x  
sgt $t1,$t0,$zero  
li $t5,10  
slt $t2,$t0,$t5  
rem $t3,$t0,2  
not $t3,$t3  
and $t1,$t1,$t2  
and $t1,$t1,$t3  
beqz $t1,end_if
```

slt: set less than
if \$t0<\$t5 then set \$t2=1 or 0

Câu lệnh if-else

- Một dạng khác của câu lệnh **if** cho phép điều kiện sai là câu lệnh **if-else**. Nếu điều kiện đúng, khối lệnh thứ nhất được thực thi, nếu sai khối lệnh thứ hai được thực thi.
- Xét chương trình sau:

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

Câu lệnh if-else

1. Thực thi phần điều kiện của câu lệnh.

2. Thêm 2 nhãn vào chương trình:

- Một nhãn cho **else**
- Một nhãn để kết thúc lệnh **if**

3. Lệnh **beqz** được thêm vào để đánh giá điều kiện rẽ nhánh tới nhãn **else**.

4. Ở cuối khối **if**, rẽ nhánh vòng qua khối **else** bằng cách sử dụng câu lệnh rẽ nhánh không điều kiện tới **end_if**.

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

Câu lệnh if-else

5. Ngay khi cấu trúc câu lệnh **if-else** được thiết lập đúng, đặt khối lệnh vào trong cấu trúc chương trình để hoàn thành khối lệnh **if-else**.

```
5  .data
6      num: .word -5
7      PositiveNumber: .asciiz "Number is positive"
8      NegativeNumber: .asciiz "Number is negative"
9  .include "utils.asm"
10
11  .text
12      lw $t0, num
13      sgt $t1, $t0, $zero
14      beqz $t1, else
15          #if block
16          la $a0, PositiveNumber
17          li $v0, 4
18          syscall
19          b end_if
20          #else block
21      else:
22          la $a0, NegativeNumber
23          jal PrintString
24      end_if:
25      jal Exit
```

Câu lệnh if-elseif-else

- Xét chương trình sau:

```
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 bằng 1 chú thích cho toàn bộ khối lệnh và đặt 1 nhãn cho:

- Mỗi điều kiện **elseif**
- **Else** cuối cùng
- Các điều kiện **end_if**

Ở cuối mỗi khối lệnh, đặt một rẽ nhánh tới nhãn **end_if**. Ngay khi bất kỳ một khối lệnh nào được thực thi thì sẽ thoát ra khỏi toàn bộ khối 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. Tiếp theo đặt các điều kiện logic vào đầu mỗi khối **if** và **elseif**. Các lệnh **if** và **elseif** 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
```

slti: set less than immediate
if $\$s0 < 0$ then set $\$t1 = 1$ or 0

sge: set greater or equal
if $\$s0 \geq 90$ then set $\$t1 = 1$ or 0

```
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 mỗi khối lệnh các lệnh logic thích hợp.

```
4  .data
5      num: .word 70
6      InvalidInput: .asciiz "Number must be > 0 and < 100"
7      OutputA: .asciiz "Grade is A"
8      OutputB: .asciiz "Grade is B"
9      OutputC: .asciiz "Grade is C"
10     OutputD: .asciiz "Grade is D"
11     OutputF: .asciiz "Grade is F"
12  .include "utils.asm"
14  .text
15      #if block
16          lw $s0, num
17          slti $t1, $s0, 0
18          sgt $t2, $s0, 100
19          or $t1, $t1, $t2
20          beqz $t1, grade_A
21      #invalid input block
22          la $a0, InvalidInput
23          jal PrintString
24          b end_if
25
26      grade_A:
27          sge $t1, $s0, 90
28          beqz $t1, grade_B
29          la $a0, OutputA
30          jal PrintString
31          b end_if
32
33      grade_B:
34          sge $t1, $s0, 80
35          beqz $t1, grade_C
36          la $a0, OutputB
37          jal PrintString
38          b end_if
39
40      grade_C:
41          sge $t1, $s0, 70
42          beqz $t1, grade_D
43          la $a0, OutputC
44          jal PrintString
45          b end_if
46
47      grade_D:
48          sge $t1, $s0, 60
49          beqz $t1, else
50          la $a0, OutputD
51          jal PrintString
52          b end_if
53
54      else:
55          la $a0, OutputF
56          jal PrintString
57          b end_if
58
59      end_if:
60      jal Exit
```

Vòng lặp điều kiện

- Khái niệm của một vòng lặp điều kiện chính là một vòng lặp có câu lệnh bảo vệ kiểm soát việc vòng lặp có được thực thi hay không.
- Vấn đề chính của vòng lặp điều kiện là xử lý đầu vào cho đến khi đáp ứng một số điều kiện (giá trị lặp).
- Ví dụ, một vòng lặp điều kiện có thể được sử dụng để kiểm tra giá trị người dùng nhập vào cho đến khi thoả mãn một giá trị cụ thể nào đó.

Vòng lặp điều kiện

- Đoạn mã sau sử dụng câu lệnh while để thực hiện một vòng lặp điều kiện nhắc người dùng nhập vào một số nguyên và in ra số nguyên đó cho đến khi người dùng nhập vào giá trị “-1”.

```
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

- Đoạn mã như sau:

```
2  .data
3      prompt: .asciiz "Enter an integer, -1 to stop: "
4
5  .text
6      #set sentinel value (prompt the user for input)
7      la $a0, prompt
8      jal PromptInt
9      move $s0, $v0
10     start_loop:
11         sne $t1, $s0, -1
12         beqz $t1, end_loop
13         # code block
14         la $a0, prompt
15         jal PromptInt
16         move $s0, $v0
17         b start_loop
18     end_loop:
```

sne: set not equal
if $\$s0 \neq -1$ then set $\$t1 = 1$ or 0

Vòng lặp điều kiện

6. Cấu trúc ở trên chính là cấu trúc cho vòng lặp điều kiện.

Có thể thêm các logic và lệnh cần thiết để kết thúc chương trình trong các khối lệnh.

Vòng lặp điều kiện

- Kết quả cuối cùng như sau:

```
2  .data
3      prompt: .asciiz "Enter an integer, -1 to stop: "
4      output: .asciiz "\nYou entered: "
5  .include "utils.asm"
6
7  .text
8      #set sentinel value (prompt the user for input)
9      la $a0, prompt
10     jal PromptInt
11     move $s0, $v0
12     start_loop:
13         sne $t1, $s0, -1
14         beqz $t1, end_loop
15
16         # code block
17         la $a0, output
18         move $a1, $s0
19         jal PrintInt
20
21         la $a0, prompt
22         jal PromptInt
23         move $s0, $v0
24         b start_loop
25     end_loop:
26     jal Exit
```

Vòng lặp biến đếm

- Một vòng lặp biến đếm là một vòng lặp có dự định được thực thi một số lần nhất định.
- Định dạng chung như sau:
 - Giá trị khởi tạo biến đếm
 - Điều kiện kết thúc (thường khi bộ đếm đạt được một giá trị đã xác định trước)
 - Bước tăng của bộ đếm

Vòng lặp biến đếm

- Xét ví dụ sau: tính tổng từ 0 đến $n-1$.

```
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);
```

- Cấu trúc vòng lặp **for** có 3 phần:
 - Đầu tiên là khởi tạo, xảy ra trước khi vòng lặp được thực thi ("i=0").
 - Thứ 2 là điều kiện để tiếp tục vào vòng lặp ("i < size").
 - Điều kiện cuối cùng chỉ ra cách tăng biến đếm ("i++", hay $i=i+1$).

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 điều kiện vào vòng lặp, hoặc kết thúc vòng lặp nếu điều kiện thoả mãn.
4. Thực hiện tăng biến đếm, và rẽ nhánh quay lại nơi bắt đầu vòng lặp.

Vòng lặp biến đếm

- Khi những bước trên hoàn thành, cấu trúc cơ bản của bộ đếm vòng lặp điều khiển 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:
```

sle: set less or equal
if $\$s0 \leq \$s1$ then set $\$t1=1$ or 0

Vòng lặp biến đếm

5. Thực hiện khối lệnh cho vòng lặp for. Bổ sung thêm các lệnh khác nếu cần. Chương trình cuối cùng như sau:

```
1  .data
2      prompt: .asciiz "Enter the value to calculate the sum up to: "
3      output: .asciiz "The final result is: "
4
5  .include "utils.asm"
6
7  .text
8      la $a0, prompt
9      jal PromptInt
10     move $s1, $v0
11     li $s0, 0
12     #Initialize the total
13     li $s2, 0
14
15     start_loop:
16         sle $t1, $s0, $s1
17         beqz $t1, end_loop
18
19         # code block
20         add $s2, $s2, $s0
21
22         addi $s0, $s0, 1
23         b start_loop
24     end_loop:
25
26     la $a0, output
27     move $a1, $s2
28     jal PrintInt
29
30     jal Exit
```

Các khối lệnh lồng nhau

- Khối lệnh lồng nhau thường xuất hiện trong các thuật toán.
- Xét ví dụ sau:

```
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

- Chương trình này bao gồm:
 - Một vòng lặp điều kiện để lấy giá trị người dùng nhập vào.
 - Một câu lệnh **if**, để kiểm tra đầu vào có > 0 hay không.
 - Một vòng lặp biến đếm.
- Câu lệnh **if** được lồng vào trong khối vòng lặp điều kiện, và vòng lặp biến đếm lại được lồng trong câu lệnh **if-else**.

Các khối lệnh lồng nhau

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

```
#Sentinel Control Loop
```

```
.data
```

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

```
.text
```

```
la $a0, prompt
```

```
jal PromptInt
```

```
move $s0, $v0
```

```
start_outer_loop:
```

```
sne $t1, $s0, -1
```

```
beqz $t1, end_outer_loop
```

```
#code block
```

```
la $a0, prompt
```

```
jal PromptInt
```

```
move $s0, $v0
```

```
b start_outer_loop
```

```
end_outer_loop:
```


Các khối lệnh lồng nhau

- Khối lệnh trong vòng lặp điều kiện ở trên được thay thế bằng câu lệnh **if-else** để kiểm tra giá trị đầu vào.
- Sau khi hoàn thành, khối lệnh trông như sau:

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


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.

```
#if block
la $a0,error
jal PrintInt
b end_if

else:
#else block
#summation loop
li $s1,0
li $s2,0 #initialize loop

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 $s0,output
move $a1,$s2
jal PrintInt

end_if:
```

Các khối lệnh lồng nhau

- Chương trình sau khi hoàn thành:

```
1  #Sentinel Control Loop
2  .data
3      prompt: .asciiz "\nEnter an integer, -1 to stop: "
4      error: .asciiz "\nValues for n must be > 0"
5      output: .asciiz "\nThe total is: "
6  .include "utils.asm"
7
8  .text
9      la $a0,prompt
10     jal PromptInt
11     move $s0,$v0
12     start_outer_loop:
13         sne $t1,$s0,-1
14         beqz $t1,end_outer_loop
15
16         #code block
17         #If test for valid input
18         slti $t1,$s0,-1
19         beqz $t1,else
20             #if block
21             la $a0,error
22             jal PrintInt
23             b end_if
24         else:
25             #else block
26             #summation loop
27             li $s1,0
28             li $s2,0 #initialize loop
29
30             start_inner_loop:
31                 sle $t1,$s1,$s0
32                 beqz $t1,end_inner_loop
33
34                 add $s2,$s2,$s1
35
36                 addi $s1,$s1,1
37                 b start_inner_loop
38             end_inner_loop:
39                 la $s0,output
40                 move $a1,$s2
41                 jal PrintInt
42             end_if:
43
44                 la $a0,prompt
45                 jal PromptInt
46                 move $s0,$v0
47                 b start_outer_loop
48     end_outer_loop:
```

Câu hỏi

- Sự khác nhau giữa lệnh **jump** và **branch** là gì?

Chương trình đầy đủ bằng hợp ngữ

Chương trình tính điểm trung bình

- Viết chương trình đọc vào điểm số từ người dùng và tính điểm trung bình.
- Điểm trung bình và điểm chữ tương ứng sẽ được in ra màn hình.

Chương trình đầy đủ bằng hợp ngữ

1. Trước khi bắt đầu, nên viết chương trình dưới dạng mã giả vì:
 - Cho phép suy luận ở mức độ cao hơn
 - Lập trình dễ dàng hơn vì có thể dịch thẳng từ mã giả sang hợp ngữ.
 - Mã giả có thể coi là tài liệu mô tả cách thức chương trình làm việc, và nên được thêm vào phần chú thích ở đầu mỗi chương trình.

Chương trình đầy đủ bằng hợp ngữ

2. Chú thích mở đầu chứa các thông tin:

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

```

# 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")
# }
# }
```


Assignment 6.1

- Bài tập này thực hiện câu lệnh “if-then-else” bằng cách sử dụng một số lệnh cơ bản như **slt**, **addi**, **jump** và **branch**.

if ($i \leq j$)
 $x = x + 1;$
 $z = 1;$
else
 $y = y - 1;$
 *$z = 2 * z;$*
- Mã hợp ngữ như sau:

```
start:
    slt    $t0,$s2,$s1      # j<i
    bne    $t0,$zero,else   # branch to else if j<i
    addi    $t1,$t1,1        # then part: x=x+1
    addi    $t3,$zero,1      # z=1
    j      endif            # skip "else" part
else:
    addi    $t2,$t2,-1       # begin else part: y=y-1
    add     $t3,$t3,$t3      # z=2*z
endif:
```

Assignment 6.1

- Tạo ra một chương trình mới sử dụng đoạn mã trên.
 - Khởi tạo giá trị cho biến i và j .
 - Biên dịch chương trình.
 - Chạy chương trình từng bước một, quan sát sự thay đổi của bộ nhớ và nội dung của thanh ghi từng bước một.

Assignment 6.2

Sửa Assignment 6.1, sao cho điều kiện kiểm tra trở thành:

- $i < j$
- $i \geq j$
- $i + j \leq 0$
- $i + j > m + n$

Assignment 6.3

- Đoạn mã sau mô tả cách thực hiện câu lệnh vòng lặp. Chương trình này tính tổng các phần tử trong mảng A. *loop: i=i+step;*
Sum=sum+A[i];
If(i !=n) goto loop;
- Giả thiết chỉ số i, địa chỉ bắt đầu của A, hằng số so sánh n, số bước và tổng được lưu lần lượt trong các thanh ghi \$s1, \$s2, \$s3, \$s4 và \$s5, respectively.

```
.text
loop: add    $s1,$s1,$s4      #i=i+step
      add    $t1,$s1,$s1     #t1=2*s1
      add    $t1,$t1,$t1     #t1=4*s1
      add    $t1,$t1,$s2     #t1 store the address of A[i]
      lw     $t0,0($t1)      #load value of A[i] in $t0
      add    $s5,$s5,$t0     #sum=sum+A[i]
      bne    $s1,$s3,loop    #if i != n, goto loop
```

Assignment 6.3

Tạo ra một chương trình mới sử dụng đoạn mã trên.

- Khởi tạo giá trị cho các biến i , n , số bước, tổng và mảng A .
- Biên dịch chương trình.
- Chạy chương trình từng bước một, quan sát sự thay đổi của bộ nhớ và nội dung của thanh ghi từng bước một.
- Thay đổi giá trị biến vài lần và kiểm tra lại các giá trị.

Assignment 6.4

Sửa Assignment 6.3, sao cho điều kiện kiểm tra ở cuối mỗi vòng lặp là:

- $i < n$
- $i \leq n$
- $\text{sum} \geq 0$
- $A[i] == 0$

Assignment 6.5

- Một câu lệnh switch/case cho phép rẽ nhiều nhánh dựa trên các giá trị số nguyên.
- Trong ví dụ sau, biến test trong câu lệnh switch có thể là một trong 3 giá trị [0, 2] và mỗi trường hợp sẽ thực thi các xử lý khác nhau.

```
switch(test) {  
    case 0:  
        a=a+1; break;  
    case 1:  
        a=a-1; break;  
    case 2:  
        b=2*b; break;  
}
```

Assignment 6.5

- Giả thiết **a** và **b** được lưu trong các thanh ghi \$s2 và \$s3.

```
.data
test: .word 1
.text
    la    $s0, test    #load the address of test variable
    lw    $s1, 0($s0)  #load the value of test to register $t1
    li    $t0, 0        #load value for test case
    li    $t1, 1
    li    $t2, 2
    beq   $s1, $t0, case_0
    beq   $s1, $t1, case_1
    beq   $s1, $t2, case_2
    j     default
case_0:   addi  $s2, $s2, 1      #a=a+1
    j     continue
case_1:   sub   $s2, $s2, $t1    #a=a-1
    j     continue
case_2:   add   $s3, $s3, $s3    #b=2*b
    j     continue
default:
continue:
```


Assignment 6.5

Tạo ra một chương trình thực hiện đoạn mã trên.

- Biên dịch chương trình.
- Chạy chương trình từng bước một, quan sát sự thay đổi của bộ nhớ và nội dung của thanh ghi từng bước một.
- Thay đổi giá trị biến test và chạy chương trình vài lần để kiểm tra tất cả các trường hợp.

Assignment 6.6

- Để phát hiện tràn trong phép cộng, có thể sử dụng quy tắc: khi cộng hai toán hạng cùng dấu, tràn số sẽ xảy ra nếu tổng không cùng dấu với cả hai toán hạng.
- Hãy sử dụng quy tắc trên để viết một chương trình phát hiện tràn số.

Assignment 6.7

- Hãy viết một chương trình yêu cầu người dùng nhập vào một số và kiểm tra xem số đó có phải là số nguyên tố hay không.
 - Chương trình in ra dòng thông báo “Number n is prime” nếu là số nguyên tố, và “Number n is not prime” nếu không phải là số nguyên tố.

Kết thúc tuần 6