Nội dung

- Điều khiển luồng: Condition codes
- Rẽ nhánh có điều kiện
- **Vòng lặp**

Vòng lặp – Ví dụ

Code C

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

```
int i = 0, sum = 0;
while (i < 10)
{
    sum += i;
    i++;
}
```

Code assembly

```
1.        movl $0, -4(%ebp)    # i
2.        movl $0, -8(%ebp)    # result
3.        jmp .test
4.   .Loop:
5.        movl -4(%ebp), %eax
6.        addl %eax, -8(%ebp)
7.        incl -4(%ebp)
8.   .test:
9.        cmpl $10, -4(%ebp)
10.       jl .Loop
11.   // outside of loop
```

Vòng lặp (loops)

- **Vòng lặp trong C**
  - do-while
  - while
  - for

- **Vòng lặp ở mức máy tính**
  - Không có instruction hỗ trợ trực tiếp
  - Là tổ hợp các phép **kiểm tra** và **jump có điều kiện**
  - Dựa trên dạng vòng lặp **do-while**
    - Các dạng vòng lặp khác trong C sẽ được chuyển sang dạng này sau đó biên dịch thành mã máy

Vòng lặp Do-While

C Code

```
int pcount_do(unsigned int x)
{
  int result = 0;
  do {
    result += x & 0x1;
    x >>= 1;
  } while (x);
  return result;
}
```

Goto Version

```
int pcount_goto(unsigned int x)
{
    long result = 0;
 loop:
    result += x & 0x1;
    x >>= 1;
    if(x) goto loop;
    return result;
}
```

- Đếm số bit 1 có trong tham số x (“popcount”)
- Sử dụng rẽ nhánh có điều kiện để tiếp tục hoặc thoát khỏi vòng lặp

Biên dịch vòng lặp Do-While

Goto Version

```
int pcount_goto(unsigned int x)
{
        int result = 0;
 loop:
        result += x & 0x1;
        x >>= 1;
        if(x) goto loop;
        return result;
}
```

- Registers:

%edx	x
%ecx	result

```
   movl   $0, %ecx      #   result = 0
.L2:      # loop:
   movl   %edx, %eax
   andl   $1, %eax      #   t = x & 1
   addl   %eax, %ecx    #   result += t
   shrl   %edx          #   x >>= 1
   jne    .L2           #   If !0, goto loop
```

Chuyển mã vòng lặp **Do-while**: **Tổng quát**

C Code

```
do
   Body
while (Test);
```

Goto Version

```
loop:
   Body
   if (Test)
      goto loop
```

- **Body:**

```
{
   Statement1;
   Statement2;
      …
   Statementn;
}
```

Chuyển mã vòng lặp – Từ C sang assembly

Ví dụ

C Code

```
int func1(int a)
{
  int sum = 0, n = 0;
  do{
     sum += a;
     n++;
  } while (n<10)
  return sum;
}
```

```
int sum = 0, n = 0
loop:
   sum += a;
   n++;
if (n<10) goto loop;
return sum
```

Code assembly

```
// a ở ô nhớ %ebp+8
1.    ...
2.    movl $0, -4(%ebp) # sum
3.    movl $0, -8(%ebp) # n
      Loop:
        movl 8(%ebp), %eax   //a
        addl %eax, -4(%ebp)  //sum+=a
        addl $1, -8(%ebp)  //n++
        cmpl $10, -8(%ebp)
        jl .Loop
      //return sum
```

Chuyển mã vòng lặp **While**

- Khác biệt giữa **do-while** và **while**?
  - **Do-while**: thực hiện body ít nhất 1 lần
  - **While**: có thể không thực hiện
- **Chuyển While sang Do-while**
  - Cần đảm bảo thực hiện kiểm tra điều kiện trước tiên!

Chuyển mã vòng lặp **While – Dạng 1**

- Chuyển mã dạng "**nhảy vào giữa**" → kiểm tra điều kiện trước
- Sử dụng với option **-Og**

While version

```
while (Test)
  Body
```

Goto Version

```
  goto test;
loop:
  Body
test:
  if (Test)
    goto loop;
done:
```

Chuyển mã vòng lặp **While – Dạng 1 – Ví dụ**

C Code

```
int pcount_while
(unsigned int x)
{
  int result = 0;
  while (x) {
    result += x & 0x1;
    x >>= 1;
  }
  return result;
}
```

Dạng "Nhảy vào giữa"

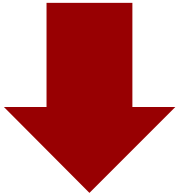
```
int pcount_goto_jtm
  (unsigned int x)
{
    int result = 0;
    goto test;
 loop:
    result += x & 0x1;
    x >>= 1;
 test:
    if(x) goto loop;
    return result;
}
```

- Goto đầu tiên bắt đầu vòng lặp tại test để kiểm tra điều kiện trước

Chuyển mã vòng lặp **While – Dạng 2**

While version

```
while (Test)
  Body
```



Do-While Version

```
  if (!Test)
    goto done;
  do
    Body
  while(Test);
done:
```

Goto Version

```
  if (!Test)
    goto done;
loop:
  Body
  if (Test)
    goto loop;
done:
```

- Chuyển sang dạng "Do-while"
- Sử dụng với option **–O1**

Chuyển mã vòng lặp **While – Dạng 2 – Ví dụ**

C Code

```
int pcount_while
(unsigned int x)
{
  int result = 0;
  while (x) {
    result += x & 0x1;
    x >>= 1;
  }
  return result;
}
```

Dạng Do-While

```
int pcount_goto_dw
  (unsigned int x)
{
    int result = 0;
    if (!x) goto done;
 loop:
    result += x & 0x1;
    x >>= 1;
    if(x) goto loop;
 done:
    return result;
}
```

- Điều kiện ban đầu được kiểm tra trước khi vào vòng lặp

Dạng vòng lặp For

for (*Init*; *Test* ; *Update*)

Body

```
#define WSIZE 8*sizeof(int)
int pcount_for(unsigned int x)
{
  size_t i;
  int result = 0;
  for (i = 0; i < WSIZE; i++)
  {
    unsigned bit =
      (x >> i) & 0x1;
    result += bit;
  }
  return result;
}
```

Khởi tạo

```
i = 0
```

Kiểm tra

```
i < WSIZE
```

Cập nhật

```
i++
```

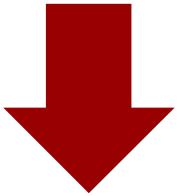
Body

```
{
  unsigned bit =
     (x >> i) & 0x1;
  result += bit;
}
```

Vòng lặp For → Vòng lặp While

For Version

```
for (Init; Test; Update)
        Body
```



While Version

```
Init;
while (Test) {
        Body
        Update;
}
```

Chuyển vòng lặp For sang While

Khởi tạo

```
i = 0
```

Kiểm tra

```
i < WSIZE
```

Cập nhật

```
i++
```

Body

```
{
  unsigned bit =
     (x >> i) & 0x1;
  result += bit;
}
```

```
int pcount_for_while(unsigned int x)
{
  size_t i;
  long result = 0;
  i = 0;
  while (i < WSIZE)
  {
    unsigned bit =
      (x >> i) & 0x1;
    result += bit;
    i++;
  }
  return result;
}
```

Chuyển vòng lặp For sang Do-While

C Code

```
int pcount_for(unsigned int x)
{
  size_t i;
  int result = 0;
  for (i = 0; i < WSIZE; i++)
  {
    unsigned bit =
      (x >> i) & 0x1;
    result += bit;
  }
  return result;
}
```

Goto Version

```
int pcount_for_goto_dw
  (unsigned int x) {
  size_t i;
  int result = 0;
  i = 0;                          Init
  if (!(i < WSIZE))               ! Test
    goto done;
 loop:
  {
    unsigned bit =
      (x >> i) & 0x1;             Body
    result += bit;
  }
  i++;                            Update
  if (i < WSIZE)                  Test
    goto loop;
 done:
  return result;
}
```

Chuyển mã vòng lặp – Từ C sang assembly

Ví dụ

C Code

```
int func1(int a)
{
  int sum = 0;
  for (int i = 0; i < a; i+=2)
     sum += (a - i);
  return sum;
}
```

goto Test:
Loop:
  s+= (a-i)
  i+=2
Test:
  if (i<a)
  goto Loop
return sum

Code assembly

```
// a ở ô nhớ %ebp+8
1.    ...
2.    movl $0, -4(%ebp) # sum
3.    movl $0, -8(%ebp) # i
      movl 8(%ebp), %eax //a
      jmp .Test
      Loop:
        subl -8(%ebp),%eax //a=a-i
        addl %eax, -4(%ebp) //sum=sum+a=sum+(a-i)
        addl $1, -8(%ebp)

      Test:
        cmpl %eax, -8(%ebp)  //a > i
        jl .Loop
```

Chuyển mã vòng lặp – Từ assembly sang C

Ví dụ 1

```
// x at %ebp+8
1.  func:
2.          pushl %ebp
3.          movl %esp,%ebp
4.          subl $4,%esp
5.          movl $0,-4(%ebp)      # count
6.  .L1:
7.          addl $2,8(%ebp)
8.          incl -4(%ebp)
9.          cmpl $9,8(%ebp)
10.         jle .L1
11.         movl -4(%ebp),%eax
12.         leave
13.         ret
```

- Khởi tạo?

  count =0

- Điều kiện duy trì vòng lặp?

- Body?

Code C?

```
count =0;
do{
x+=2;
count++;
}
while (x<=9)
```

Chuyển mã vòng lặp – Từ assembly sang C
Ví dụ 2

```
func:
1.          …
2.          movl $0,-8(%ebp)      # count
3.          movl $0,-4(%ebp)      # i
4.  .L2:
5.          cmpl $19,-4(%ebp)
6.          jg .L3
7.          movl -4(%ebp),%eax
8.          addl %eax,-8(%ebp)
9.          incl -4(%ebp)
10.         jmp .L2
11. .L3:
12.         leave
13.         ret
```

- Khởi tạo?
- Điều kiện duy trì vòng lặp?
- Cập nhật?
- Body?

```
for (i=0,i<20, i++)
{
count += i
}
```

Code C?

Chuyển mã vòng lặp – Từ assembly sang C
Ví dụ 3

```
1.          movl $0,-8(%ebp)      # count
2.          movl $0,-4(%ebp)      # i
3.  .L1:
4.          cmpl $25,-4(%ebp)
5.          jge  .L3
6.          movl -4(%ebp),%eax
7.          cmpl -8(%ebp), %eax
8.          jg   .L2
9.          addl %eax,-8(%ebp)
10..L2:
11.         subl %eax, -8(%ebp)
12.         incl -4(%ebp)
13.         jmp .L1
14..L3:
15.         leave
16.         ret
```

- Khởi tạo?
- Điều kiện duy trì vòng lặp?
- Body?

Code C?

```
count =0;
i =0;
while (i<25)
{
if (i<=count)
   count -=i;
count --;
i++;
}
```

Chuyển mã vòng lặp – Từ assembly sang C
Ví dụ 4

Cho mảng ký tự **char* a** có độ dài **len**

```
//  &a[0] at %ebp+8, len at %ebp+12
1.  array_func:
2.          movl    $0, -8(%ebp) # result
3.          movl    $0, -4(%ebp) # i
4.          jmp     .L2
5.  .L3:
6.          movl    -4(%ebp), %edx
7.          movl    8(%ebp), %eax
8.          addl    %edx, %eax
9.          mov     (%eax), %al
10.         subl    $48, %eax
11.         addl    %eax, -8(%ebp)
12.         addl    $1, -4(%ebp)
13. .L2:
14.         movl    -4(%ebp), %eax
15.         cmpl    12(%ebp), %eax
16.         jl      .L3
17.         movl    -8(%ebp), %eax #return
```

- Khởi tạo?
- Điều kiện dừng?
- Cập nhật?
- Body?

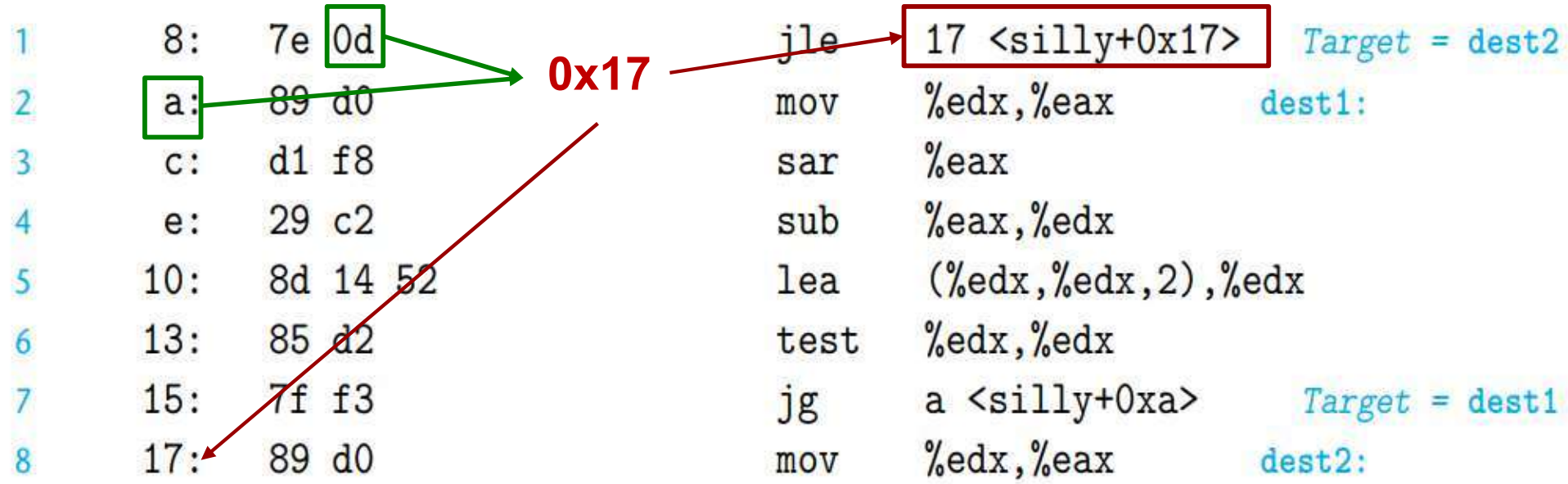
Code C?

Extra 1: Các câu lệnh jump - Label

- Vị trí sẽ nhảy đến của các lệnh jump trong mã assembly được biểu diễn dưới dạng các *label.*
- **Assembler** và **Linker** có thể lựa chọn 1 trong 2 cách để xác định vị trí nhảy đến:
  - *Địa chỉ tuyệt đối*: 4 (hoặc 8) bytes địa chỉ chính xác của instruction đích muốn nhảy đến.
  - *PC relative – địa chỉ tương đối*: khoảng cách tương đối giữa instruction đích và vị trí instruction liền sau lệnh jump (giá trị thanh ghi PC).

Extra 1: Các câu lệnh jump - Label

- ***PC relative – địa chỉ tương đối***



Extra 2: Sử dụng Condition Codes
Gán giá trị dựa trên điều kiện

- Các instruction SetX
  - **`setx`** *`dest`*
  - Gán ***byte thấp nhất (low-order byte)*** của destination thành 1 hoặc 0 dựa trên 1 nhóm các condition codes.
  - Không thay đổi 7 bytes còn lại

SetX	Condition	Description
`sete`	`ZF`	Equal / Zero
`setne`	`~ZF`	Not Equal / Not Zero
`sets`	`SF`	Negative
`setns`	`~SF`	Nonnegative
`setg`	`~(SF^OF)&~ZF`	Greater (Signed)
`setge`	`~(SF^OF)`	Greater or Equal (Signed)
`setl`	`(SF^OF)`	Less (Signed)
`setle`	`(SF^OF)|ZF`	Less or Equal (Signed)
`seta`	`~CF&~ZF`	Above (unsigned)
`setb`	`CF`	Below (unsigned)

Các thanh ghi x86-64: low-order byte?

%rax	%al	%r8	%r8b
%rbx	%bl	%r9	%r9b
%rcx	%cl	%r10	%r10b
%rdx	%dl	%r11	%r11b
%rsi	%sil	%r12	%r12b
%rdi	%dil	%r13	%r13b
%rsp	%spl	%r14	%r14b
%rbp	%bpl	%r15	%r15b

- Có thể tham chiếu đến các byte thấp này

Extra 2: Sử dụng Condition Codes
Gán giá trị dựa trên điều kiện (tt)

- Các instruction SetX:
  - Gán giá trị cho 1 byte dựa trên 1 nhóm các condition codes
- Thay đổi 1 byte trong các thanh ghi
  - Không thay đổi các bytes còn lại
  - Thường dùng **`movzbl`**
    - Instruction 32-bit cũng gán 32 bits cao thành 0

```
int gt (long x, long y)
{
  return x > y;
}
```

Thanh ghi	Tác dụng
`%rdi`	Tham số **`x`**
`%rsi`	Tham số **`y`**
`%rax`	Giá trị trả về

```
cmpq    %rsi, %rdi    # Compare x:y
setg    %al           # Set when >
movzbl  %al, %eax     # Zero rest of %rax
ret
```

Extra 3: Sử dụng Condition Codes
Chuyển giá trị có điều kiện (conditional move)

- Các instruction move có điều kiện
  - Hỗ trợ thực hiện:

    if (Test) Dest ← Src
  - Hỗ trợ trong các bộ xử lý x86 từ 1995 trở về sau
  - GCC tries to use them
    - But, only when known to be safe
- Why?
  - Branches are very disruptive to instruction flow through pipelines
  - Conditional moves không cần chuyển luồng

C Code

```
val = Test
    ? Then_Expr
    : Else_Expr;
```

Goto Version

```
result = Then_Expr;
eval = Else_Expr;
nt = !Test;
if (nt) result = eval;
return result;
```

Chuyển giá trị có điều kiện (conditional move)
Ví dụ

```
long absdiff
   (long x, long y)
{
    long result;
    if (x > y)
        result = x-y;
    else
        result = y-x;
    return result;
}
```

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rax	Return value

```
absdiff:
    movq    %rdi, %rax   # x
    subq    %rsi, %rax   # result = x-y
    movq    %rsi, %rdx
    subq    %rdi, %rdx   # eval = y-x
    cmpq    %rsi, %rdi   # x:y
    cmovle  %rdx, %rax   # if <=, result = eval
    ret
```

Chuyển giá trị có điều kiện (conditional move)
Bad cases

Tính toán phức tạp

```
val = Test(x) ? Hard1(x) : Hard2(x);
```

- Cả 2 giá trị đều được tính toán
- Chỉ hữu ích khi các phép tính toán đều đơn giản

Tính toán có rủi ro

```
val = p ? *p : 0;
```

- Cả 2 giá trị đều được tính toán
- Có thể có những ảnh hưởng không mong muốn (p null?)

Tính toán có tác động phụ

```
val = x > 0 ? x*=7 : x+=3;
```

- Cả 2 giá trị đều được tính toán
- Cần loại bỏ tác động phụ

Nội dung

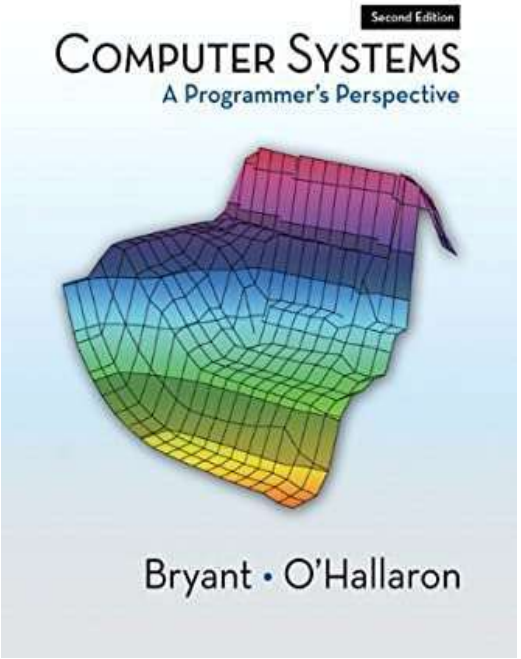
- **Các chủ đề chính:**
  1) Biểu diễn các kiểu dữ liệu và các phép tính toán bit
  2) Ngôn ngữ assembly cơ bản
  3) Điều khiển luồng trong C với assembly
  4) Các thủ tục/hàm (procedure) trong C ở mức assembly
  5) Biểu diễn mảng, cấu trúc dữ liệu trong C
  6) Một số topic ATTT: reverse engineering, bufferoverflow
  7) Phân cấp bộ nhớ, cache
  8) Linking trong biên dịch file thực thi
- **Lab liên quan**
  - Lab 1: Nội dung **1**
  - Lab 2: Nội dung 1, **2, 3**
  - Lab 3: Nội dung 1, **2, 3, 4, 5, 6**
  - Lab 4: Nội dung 1, **2,** 3**,** **4**, 5, **6**
  - Lab 5: Nội dung 1, **2,** 3**,** **4**, 5, **6**
  - Lab 6: Nội dung **1, 2, 3, 4, 5, 6**

Giáo trình

- **Giáo trình chính**

Computer Systems: A Programmer's Perspective

  - Second Edition (CS:APP2e), Pearson, 2010
  - Randal E. Bryant, David R. O'Hallaron
  - http://csapp.cs.cmu.edu



- **Tài liệu khác**

  - *The C Programming Language*, Second Edition, Prentice Hall, 1988
    - Brian Kernighan and Dennis Ritchie
  - *The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler*, 1st Edition, 2008
    - Chris Eagle
  - *Reversing: Secrets of Reverse Engineering*, 1st Edition, 2011
    - Eldad Eilam


KEEP
CALM
AND
ENJOY YOUR
SEMESTER :)