

# BÀI TẬP ÔN TẬP LẬP TRÌNH HỆ THỐNG

**Bài tập 1.** Thực hiện các phép chuyển đổi và tính toán sau:

- a. Chuyển các số hexan sang hệ nhị phân:
- 0x39A7F8<sub>16</sub>** = .....<sub>2</sub>
- 0xD5E4C<sub>16</sub>** = .....<sub>2</sub>
- b. Chuyển số nhị phân sang hệ hexan (16):
- 110010010111011<sub>2</sub>** = .....<sub>16</sub>
- 100110111001110110101<sub>2</sub>** = .....<sub>16</sub>
- c. Thực hiện tính toán:
- 0x506 + 0x12** = .....
- 0x503C – 0x42** = .....
- 0x6653 + 98** = .....


**Bài tập 2.** Byte ordering

a) Cho đoạn chương trình:

```
/* Biến val gồm 4 byte đánh thứ tự từ 1 đến 4 từ địa chỉ bắt đầu */
int val = 0x87654321;
/* pointer trỏ đến ô nhớ lưu trữ biến val */
byte_pointer valp = (byte_pointer) &val;
/* A. hàm trả về byte thứ 1 kể từ địa chỉ bắt đầu */
show_bytes(valp, 1);
/* B. hàm trả về byte thứ 2 kể từ địa chỉ bắt đầu */
show_bytes(valp, 2);
```

Kết quả trả về của 2 hàm **show\_bytes()** sẽ khác nhau như thế nào trong trường hợp chạy trên hệ thống sử dụng little-endian và big-endian?

| Hệ thống      | show_bytes(valp,1) | show_bytes(valp,2) |
|---------------|--------------------|--------------------|
| little-endian |                    |                    |
| big-endian    |                    |                    |

 đang nằm trong một số ô nhớ có địa chỉ tương ứng như bên dưới  
g với 1 byte trong bộ nhớ. Biết hệ thống đang xét là Linux 32 bit.

| Giá trị | 0x1   | 0x2   | 0x56  | 0xAB  | 0x0   | 0x12  | 0x23  | 0x16  | 0x10  |
|---------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Địa chỉ | 0x102 | 0x103 | 0x104 | 0x105 | 0x106 | 0x107 | 0x108 | 0x109 | 0x10A |

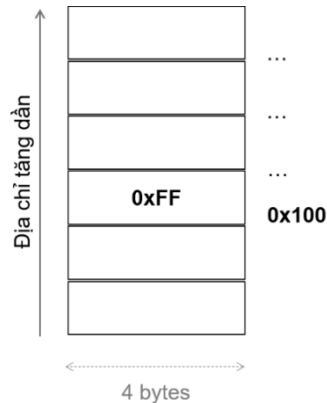
Cho trước địa chỉ lưu trữ của một số biến sau, xác định giá trị cụ thể của chúng?

| Biến     | Địa chỉ | Giá trị |
|----------|---------|---------|
| char a   | 0x10A   | 0x10    |
| int b    | 0x104   |         |
| short c  | 0x106   |         |
| double d | 0x102   |         |
| char * e | 0x106   |         |

**Bài tập 3.** Giả sử có các giá trị sau đang được lưu trong các ô nhớ và các thanh ghi:

| Địa chỉ | Giá trị | Thanh ghi | Giá trị |
|---------|---------|-----------|---------|
| 0x100   | 0xFF    | %eax      | 0x100   |
| 0x104   | 0xAB    | %ecx      | 0x1     |
| 0x108   | 0x13    | %edx      | 0x3     |
| 0x10C   | 0x11    |           |         |

a) Hãy điền vào hình bên dưới các địa chỉ và giá trị tương ứng của các ô nhớ.



b) Xác định giá trị lấy được ở lệnh mov

Giả sử ta có câu lệnh **movl [toán hạng x], %ebx** để lấy giá trị dựa trên toán hạng x và đưa vào thanh ghi %ebx. Dựa vào các giá trị trong ô nhớ và thanh ghi ở trên, điền các giá trị sẽ lấy được nếu sử dụng các toán hạng sau:

| Toán hạng x     | Giá trị lấy được                              |
|-----------------|---|
| %eax            | Giá trị lưu trong thanh ghi eax: <b>0x100</b> |
| 0x104           |   |
| \$0x108         |   |
| (%eax)          |   |
| 4(%eax)         |   |
| 9(%eax, %edx)   |   |
| 0xFC(,%ecx,4)   |   |
| (%eax, %edx, 4) |   |

c) Tác động của một số lệnh

Điền vào chỗ trống ảnh hưởng của những câu lệnh dưới đây, bao gồm thanh ghi/ô nhớ nào bị thay đổi giá trị và giá trị đó là bao nhiêu?

*Lưu ý: Giá trị các thanh ghi/ô nhớ ở mỗi câu lệnh vẫn lấy từ bảng trên, các lệnh thực thi riêng biệt không liên quan đến nhau.*

| Câu lệnh                           | Thanh ghi/ô nhớ bị thay đổi   | Giá trị                   |
|------------------------------------|-------------------------------|---------------------------|
| <b>addl</b> %ecx, (%eax)           | Ô nhớ có địa chỉ <b>0x100</b> | 0xFF + 0x1 = <b>0x100</b> |
| <b>imull</b> \$16, (%eax, %edx, 4) | Ô nhớ có địa chỉ ...          |                           |
| <b>subl</b> %edx, %eax             |                               |                           |
| <b>movl</b> (%eax, %edx, 4), %eax  |                               |                           |
| <b>leal</b> (%eax, %edx, 4), %eax  |                               |                           |

#### d) Viết một số lệnh hiện thực các ý tưởng sau

*Lưu ý: Giá trị các thanh ghi/ô nhớ ở mỗi câu lệnh vẫn lấy từ bảng trên, các lệnh thực thi riêng biệt không liên quan đến nhau. Hệ thống đang xét là 32bit*

| Tác vụ   | Lệnh assembly              |
|--|----------------------------|
| Tăng giá trị thanh ghi %eax lên 1 đơn vị   | incl %eax / addl \$1, %eax |
| Nhân giá trị đang lưu trong %ecx với 4   |                            |
| Tính toán địa chỉ ở ô nhớ nằm trên địa chỉ đang lưu trong %eax 12 byte. Kết quả lưu trong %ebx                     |                            |
| Trừ giá trị đang lưu trong ô nhớ địa chỉ 0x104 cho %edx, lưu kết quả vào chính ô nhớ đó.                           |                            |
| Giữ nguyên 4 bit thấp nhất của giá trị thanh ghi %ecx, các bit còn lại gán về 0.                                   |                            |
| Chỉ lấy 2 byte từ ô nhớ nằm trên địa chỉ đang lưu trong %eax 12 byte, kết quả lưu trong thanh ghi 2 byte của %ecx. |                            |

#### Bài tập 4. Giả sử 1 lập trình viên muốn viết 1 đoạn mã thực hiện chức năng sau:

```

1  int req_fun(int a, int b){
2      int val = 0;
3      int x = a & 0xFFFF0000;
4      int y = b & 0xFFFF;
5      val = x | y;
6      return val;
7  }
```

**a.** Hoàn thành đoạn mã assembly bên dưới để có chức năng tương tự, kết quả val lưu trong %eax.

```

1  movl    8(%ebp), %ebx
2  movl    12(%ebp), %ecx
3  andl    $0xFFFF0000, .....
4  andl    ....., .....
5  .....
6  .....
```

**b.** Một lập trình viên khác có ý tưởng khác với đoạn code tối ưu hơn, hãy hoàn thành đoạn code assembly bên dưới?

```

1  movl    8(%ebp), %ebx
2  movl    12(%ebp), %ecx
3  movl    ....., %eax
4  movw    ....., %ax
```

**Bài tập 5.** Cho đoạn mã assembly như bên dưới:

*x, y, z là các tham số thứ nhất, thứ 2 và thứ ba của hàm arith*

```

1  movl 8(%ebp), %ecx
2  movl 12(%ebp), %eax
3  imull %ecx, %eax
4  subl %ecx, %eax
5  leal (%eax,%eax,4), %eax
6  addl 16(%ebp), %eax
7  sarl $2, %eax

```

Dựa vào mã assembly, điền vào những phần còn trống trong các hàm C tương ứng:

**a. Hàm arith() phiên bản 1**

```

1  int arith(int x, int y, int z)
2  {
3      int t1 = .....;
4      int t2 = .....;
5      int t3 = .....;
6      int t4 = .....;
7      int t5 = .....;
8      return t5;
9  }

```

**b. Hàm arith() phiên bản 2 (rút gọn)**

```

1  int arith(int x, int y, int z)
2  {
3      int t1 = .....;
4      return t1;
5  }

```

**Bài tập 6.** Cho đoạn mã assembly dưới đây được tạo bởi GCC:

*x, y lần lượt là các tham số thứ nhất và thứ hai của hàm test*

```

1  movl 8(%ebp), %eax
2  movl 12(%ebp), %edx
3  cmpl $-3, %eax
4  jge .L2
5  cmpl %edx, %eax
6  jle .L3
7  imull %edx, %eax
8  jmp .L4
9  .L3:
10 leal (%edx,%eax), %eax
11 jmp .L4
12 .L2:
13 cmpl $2, %eax
14 jg .L5
15 xorl %edx, %eax
16 jmp .L4
17 .L5:
18 subl %edx, %eax
19 .L4:
20 // return val

```

- a. Hoàn thành đoạn mã C tương ứng với đoạn mã assembly trên, trong đó giá trị cuối cùng của **val** được lưu trong **%eax** để trả về tại **.L4**.

(Bài tập có nhiều đáp án, SV chỉ cần ánh xạ đúng các điều kiện – đoạn code tương ứng)

```

1  int test(int x, int y) {
2      int val = .....;
3      if ( ..... ) {
4          if ( ..... )
5              val = .....;
6          else
7              val = .....;
8      } else if ( ..... )
9          val = .....;
10     return val;
11 }
```

- b. Giả sử với tham số **x = 4, y = 2**. Khi đó **val = .....**

- c. Giả sử với tham số **x = 1, y = 9**. Khi đó **val = .....**

**Bài tập 7.** Cho đoạn mã assembly như bên dưới: *x* là tham số thứ nhất của hàm **fun\_b**

```

1  movl 8(%ebp), %ebx
2  movl $0, %eax
3  movl $0, %ecx
4  jmp  .L2
5  .L1:
6  leal (%eax,%eax), %edx
7  movl %ebx, %eax
8  andl $1, %eax
9  orl  %edx, %eax
10 shrl %ebx
11 addl $1, %ecx
12 .L2:
13 cmpl $5, %ecx
14 jle .L1
```

- a. Hoàn thành đoạn mã C tương ứng với đoạn mã assembly trên. Biết giá trị cuối cùng của **val** lưu trong **%eax** để trả về sau khi thoát vòng lặp **for**.

```

1  int fun_b(unsigned x) {
2      int val = 0;
3      int i;
4      for (.....) {
5          .....
6          .....
7      }
8      return val;
9  }
```

b. Với **x = 32**, xác định giá trị của **val**? Giải thích?

.....

.....

.....

**Bài tập 8.** Cho hàm C như sau:

```

1  int my_function()
2  {
3      int first_var = 0;
4      int second_var = 0xdeadbeef;
5      char str[2] = ?;
6
7      char buf[10];
8      gets(buf);
9      return len(buf);
10 }
```

GCC tạo ra mã assembly tương ứng như sau:

```

.section .data
.LC0:
    .byte 0x68,0x69,0x74,0x68,0x75,0x0 # mảng các byte liên tục nhau

.section .text
1  my_function:
2      pushl   %ebp
3      movl    %esp, %ebp
4      subl    $24, %esp
5      movl    $0, -4(%ebp)
6      movl    $0xdeadbeef, -8(%ebp)
7      movw    (.LC0), %dx
8      movw    %dx, -12(%ebp)
9      leal    -24(%ebp), %eax
10     pushl    %eax
11     call     gets
12     leal    -24(%ebp), %eax
13     pushl    %eax
14     call     len
15     leave
16     ret
```

Giả sử hàm **my\_function** bắt đầu thực thi với những giá trị thanh ghi như sau:

| Thanh ghi | Giá trị  |
|-----------|----------|
| %esp      | 0x800168 |
| %ebp      | 0x800180 |

Biết **.LC0** là label của 1 vùng nhớ.







**b.** Giả sử với 1 trường hợp của **T A[N]** ở trên, ta có đoạn mã C và assembly tương ứng truy xuất các phần tử của mảng như bên dưới. Hãy điền vào chỗ còn trống ở 2 đoạn code?

**Code C**

```
1 ..... A[.....]; // khai báo mảng A
2 A[0] = .....;
3 for (int i = .....; i < .....; i++)
4 {
5     .....
6     .....
7     .....
8 }
```

**Code assembly**

```
1    movl    $A, %eax    // address of A
2    movw    $0, (%eax)
3    movl    $1, %ecx    // chỉ số i
4    .L1:
5    xorl     %ebx, %ebx
6    movw    -2(%eax, %ecx, 2), %bx
7    addl    %ecx, %ebx
8    movw    %bx, (%eax, %ecx, 2)
9    incl    %ecx
10   cmpl     ....., %ecx
11   jl      .L1
```

**c.** Với đoạn mã và **T A[N]** ở câu b, cho biết sau khi thực hiện đoạn code trên, ta thu được mảng A với các giá trị phần tử như thế nào?

.....

.....

.....

**d.** Vẫn với mảng **T A[N]** ở câu b, cho địa chỉ của **A[0]** là **0x1010**. Cho biết địa chỉ của thành phần cuối của mảng A?

.....

.....

**Bài tập 10.** Cho các định nghĩa sau trong code C, với giá trị N chưa biết.

```
1  # define N ?
2  void matrix_set_val(int A[N][N], int val)
3  {
4      int i;
5      for (i = 0 ; i < N; i++)
6          A[i][i] = val;
7  }
```

Và đoạn code assembly tương ứng được tạo bởi GCC:

*Địa chỉ mảng A là tham số thứ nhất, giá trị val là tham số thứ 2*

```
1    movl    8(%ebp), %ecx
2    movl    12(%ebp), %edx
3    movl    $0, %eax
4    .L14:
5    movl    %edx, (%ecx,%eax)
6    addl    $68, %eax
7    cmpl    $1088, %eax
8    jne     .L14
```



c. Tổng kích thước của A thay đổi tùy vào hệ thống 32 bit hoặc 64 bit, trong đó chênh lệch kích thước là 96 bytes và tồn tại phần tử A[4][3].

.....

.....

.....

.....

.....

.....

**Bài tập 12.** Cho định nghĩa struct như bên dưới trong Windows 32 bit, có alignment.

```
1 typedef struct {
2     int a[2];
3     char b;
4     long c;
5     double d;
6 } str_ex1;
```

a. Vẽ hình minh họa việc cấp phát struct **str\_ex1** trên trong bộ nhớ?

.....

.....

.....

b. Có bao nhiêu byte trống được chèn thêm vào struct khi thực hiện căn chỉnh?

.....

.....

c. Tổng kích thước của struct trên là bao nhiêu?

.....

.....

d. Giả sử có code assembly của 1 hàm **func** với 2 tham số lần lượt là i và val, thực hiện xử lý các thành phần của 1 struct **str\_ex1** có tên là **s** như sau.

```
1 func:
2     movl    $str1, %eax                # address of struct s
3     movl    $1, (%eax)
4     movl    12(%ebp), %ebx
5     leal    1(%ebx), %ebx
6     movl    %ebx, 4(%eax)
7     movl    12(%ebp), %ebx
8     andl    $0xF, %ebx
9     addb    $0x30, %bl
10    movb    %bl, 8(%eax)
11    movl    8(%ebp), %ebx
12    movl    (%eax, %ebx, 4), %ecx
13    movl    %ecx, 12(%eax)
14    movl    4(%eax), %eax              # return
```

Hoàn thành đoạn code C có chức năng tương ứng với đoạn mã assembly trên?

```

1 int func(int i, int val)
2 {
3     str_ex1 s;
4     .....
5     .....
6     .....
7     .....
8     .....
9     return .....
10}

```

**Bài tập 13.** Cho struct như bên dưới trong Linux 32-bit, có yêu cầu alignment.

```

1 typedef struct {
2     short a[4];
3     char b;
4     int c;
5 } str1;

```

Một hàm func1 được dùng để gán giá trị cho thành phần a[i] và c của struct, kết quả trả về là giá trị của thành phần c như bên dưới.

```

1 int func1(int i, int val)
2 {
3     str1 s;
4     s.c = 1;
5     s.a[i] = val;
6     return s.c;
7 }

```

a. Vẽ hình minh họa việc cấp phát struct **str1** trên trong bộ nhớ?

.....  
 .....  
 .....

b. Tổng kích thước của struct trên là bao nhiêu?

.....  
 .....

c. Tìm giá trị trả về của hàm **func** với các tham số sau? Giải thích các thay đổi có trong vùng nhớ của struct?

Giả định compiler chỉ warning khi có truy xuất ngoài mảng, vẫn cho chương trình chạy bình thường.

- **func1(2, 2)**

.....  
 .....  
 .....  
 .....  
 .....

- **func1(4, 2)**

- **func1(6, 2)**

**Bài tập 14.** Cho 2 định nghĩa struct với 2 giá trị A và B chưa biết.

```

1  typedef struct {
2      short x[A][B];      /* Hằng số A và B chưa biết */
3      int y;
4  } str1;
5
6  typedef struct {
7      char array[B];      /* Hằng số B chưa biết */
8      int t;
9      short s[B];
10     int u;
11 } str2;
```

Cho đoạn code C cùng đoạn mã assembly tương ứng như bên dưới:

```

1 void setVal(str1 *r1, str2 *r2)
2 {
3     int v1 = r2->t;
4     int v2 = r2->u;
5     r1->y = v1+v2;
6 }
```

```

1 setVal:
2     movl 12(%ebp), %eax
3     movl 36(%eax), %edx
4     addl 12(%eax), %edx
5     movl 8(%ebp), %eax
6     movl %edx, 92(%eax)
```

Dựa vào tương quan giữa 2 đoạn mã C và assembly, xác định 2 giá trị **A** và **B**, biết hệ thống 32 bit và có yêu cầu alignment.

**Bài tập 15.** Cho 2 file **main.c** và **fib.c** như sau.

**/\* main.c \*/**

```
1. void fib (int n);
2. int main (int argc, char** argv)
3. {
4.     int n = 0;
5.     sscanf(argv[1], "%d", &n);
6.     fib(n);
7. }
```

**/\* fib.c \*/**

```
1. #define N 16
2. static unsigned int ring[3][N];
3. void print_bignat(unsigned int* a)
4. {
5.     int i;
6.     ...
7. }
8. void fib (int n) {
9.     int i;
10.    static int carry;
11.    ...
12. }
```

Hoàn thành bảng sau về các symbol có trong symbol table có trong 2 mô-đun main.o và fib.o, xác định các symbol là **local/global** hay **external**, **strong** hay **weak**.

- Ghi '-' ở cả 2 cột nếu tên không có trong symbol table của mô-đun tương ứng.
- Ghi N/A ở cột **Strong hay weak** nếu loại symbol là local.

**Symbol table của main.o**

| Tên symbol | Loại symbol | Strong hay weak |
|------------|-------------|-----------------|
| main       |             |                 |
| fib        |             |                 |
| n          |             |                 |

**Symbol table của fib.o**

| Tên symbol   | Loại symbol | Strong hay weak |
|--------------|-------------|-----------------|
| ring         |             |                 |
| print_bignat |             |                 |
| fib          |             |                 |
| canary       |             |                 |