Lab 5 : DÙNG HỢP NGỮ TRONG C

1. **Chuẩn đầu ra :** Sau bài này, người học có thể :
   - ✓ Khai báo và sử dụng được hợp ngữ từ ngôn ngữ cấp cao C
2. **Chuẩn bị :** Đọc trước phần lý thuyết về C.
3. **Phương tiện :**
   - ✓ Máy vi tính.
   - ✓ Chương trình nasm
4. **Thời lượng : 4 tiết**
5. **Tóm tắt lý thuyết**

Actually we have 3 ways to use it together:

- Call assembly routines from C code
- Call c routines from assembly code
- Use inline assembly in C code
-

6. **Nội dung thực hành**
   - **Cài đặt gcc**
     - **Vào terminal**
     - **Ra lệnh : sudo apt install gcc**

   Passing parameters on the stack, returning a value in EAX — a simple example

   6.1. Mô tả

   This program is built from main(), written in C to do I/O, and a function add2(), written in assembly, to do a calculation. The main() function passes two arguments to add2() on the stack, and receives a return value in the EAX register, according to standard C protocol.

   - Assembly function
   - C main() program
   - commands for building the program

   6.2. Assembly Function

```
add2() for use with main()
  ;-----------------------------------------------------------
  ; add2.asm - C-style subroutine
  ; Receive 2 arguments on the stack, add them,
  ; return the result in EAX.
  ; Satisfies
  ; int add2(int, int);
  ;-----------------------------------------------------------
      section .text
      global add2
```

```
  add2:
      ; preamble
      push rbp    ; save the prior EBP value
      mov rbp, rsp
      mov rax, [ rbp + 8 ]        ; based-indexed addressing
      add rax, [ rbp + 12 ]
      ; postamble
      pop rbp     ; minimal cleanup
      ret
  ;---------------------------------------------------------------

add3()showing more arguments, and a local variable on the stack
  ;---------------------------------------------------------------
  ;  add3.asm - find sum of 3 values
  ;  C-compatible assembly routine
  ;  This is very inefficient code, but it demonstrates
  ;  the use of a local variable on the stack.
  ;  in C:
  ;  int add3( int i1, int i2, int i3)
  ;       {
  ;               int sum = i1;
  ;               sum += i2;
  ;               sum += i3;
  ;               return sum;
  ;       }
  ;
  ; 2007-11-14 -bob,mon.
  ;---------------------------------------------------------------
      global add3

      section .text
  add3:
      push rbp
      mov rbp, rsp
      sub rsp, 8  ; leave space on stack for local var "sum"
      mov rax, [ rbp+8 ]
      mov [rbp-8 ], rax
      mov rax, [ rbp+12 ]
      add [rbp-4 ], rax
      mov rax, [ rbp+16 ]
      add [rbp-4 ], rax
```

```
    mov rax, [ rbp-4 ]
    mov rsp, rbp
    pop rbp     ; or use _leave_
    ret
;-------------------------------------------------------------
```

6.3. Main() function

add2-main This trivial C program calls a function written in Assembly language.It demonstrates the C protocol for passing arguments and returning a value.

```
 #include <stdio.h>
 int add2(int, int);     /*  a  function,  written  in  Nasm
assembly lang. */

  int main(int argc, char *argv[ ])
{
      int a, b, answer;
      printf("Enter a, b: ");
      scanf(" %i %i", &a, &b);
      answer = add2( a, b );
      printf("%i  %u  %x\n", answer, answer, answer);
      return 0;
  }
```

6.4. Building A Program From A C File And An ASM File

Assume that you have a C program in a file called c-code.c, and one or more assembly-language functions in a file called asm-code.asm. The DOS commands to build a program from these files are:

```
P:\> gcc -Wall -c c-code.c
        --> produces "c-code.o" by default
P:\> nasm -f elf64  asm-code.asm
        --> produces "asm-code.o" by default
P:\> gcc -o program.exe c-code.o asm-code.o
        --> produces "program.exe" program
```

- The first two steps (producing .o files) can be done in any order, and the .o files may be listed in any order in the final step.
- The "gcc -c" C-compiling step can be combined with the final "gcc" linking step if desired.

- More than two files can be included; apply the appropriate command for each, and include all the ".o" files in the final linking step.
- There must be exactly one main() function; it can be either C or assembly.

7. Call assembly form C (Example 2)

- Program print hello, world on creen using C function call assembly function
- First of all let's write simple C program like this

```
#include <string.h>

int main() {
  char* str = "Hello World\n";
  int len = strlen(str);
  printHelloWorld(str, len);
  return 0;
}
```

- Here we can see C code which defines two variables: our Hello world string which we will write to stdout and length of this string. Next we call printHelloWorld assembly function with this 2 variables as parameters. As we use x86_64 Linux, we must know x86_64 linux calling convetions, so we will know how to write printHelloWorld function, how to get incoming parameters and etc… When we call function first six parameters passes through rdi, rsi, rdx, rcx, r8 and r9 general purpose registers, all another through the stack. So we can get first and second parameter from rdi and rsi registers and call write syscall and than return from function with ret instruction:

```
global printHelloWorld

section .text
printHelloWorld:
        ;; 1 arg
        mov r10, rdi
        ;; 2 arg
        mov r11, rsi
        ;; call write syscall
        mov rax, 1
        mov rdi, 1
        mov rsi, r10
        mov rdx, r11
        syscall
        ret
```

- Now we can build it with:
  - `nasm -f elf64 -o casm.o casm.asm`
  - `gcc casm.o casm.c -o casm`
  - `./casm`

- `Result : hello world`

-

8. Call C from assembly

- And the last method is to call C function from assembly code. For example we have following simple C code with one function which just prints Hello world:

```c
#include <stdio.h>

extern int print();

int print() {
 printf("Hello World\n");
 return 0;
}
```

- **Save with name casm.c**
- Now we can define this function as extern in our assembly code and call it with call instruction as we do it much times in previous posts:

```
global _start

extern print

section .text

_start:
        call print

        mov rax, 60
        mov rdi, 0
        syscall
```

- **Save with name casm.asm**
- **Build :**
  - `gcc  -c casm.c -o c.o`
  - `nasm -f elf64 casm.asm -o casm.o`
  - `ld     -dynamic-linker  /lib64/ld-linux-x86-64.so.2  -lc casm.o c.o -o casm`
- **run ./casm**
- **result : hello world**

9. Example

printf1.asm basic calling printf
The nasm source code is printf1.asm
The result of the assembly is printf1.lst
The equivalent "C" program is printf1.c

Running the program produces output printf1.out

This program demonstrates basic use of "C" library function printf.
The equivalent "C" code is shown as comments in the assembly language.

```
; printf1.asm   print an integer from storage and from a register
; Assemble:  nasm -f elf -l printf.lst  printf1.asm
; Link:      gcc -o printf1  printf1.o
; Run:       printf1
; Output:    a=5, eax=7

; Equivalent C code
; /* printf1.c  print an int and an expression */
; #include
; int main()
; {
;   int a=5;
;   printf("a=%d, eax=%d\n", a, a+2);
;   return 0;
; }

; Declare some external functions
;
        extern     printf       ; the C function, to be called

       SECTION .data     ; Data section, initialized variables

      a:    dd    5            ; int a=5;
fmt:    db "a=%d, eax=%d", 10, 0 ; The printf format, "\n",'0'


        SECTION .text                   ; Code section.

        global main       ; the standard gcc entry point
main:               ; the program label for the entry point
        push    ebp       ; set up stack frame
        mov     ebp,esp

       mov   eax, [a]     ; put a from store into register
       add   eax, 2       ; a+2
       push  eax          ; value of a+2
      push    dword [a]  ; value of variable a
      push    dword fmt  ; address of ctrl string
      call    printf           ; Call C function
```

```
        add     esp, 12     ; pop stack 3 push times 4 bytes
        mov     esp, ebp    ; takedown stack frame
        pop     ebp         ; same as "leave" op
       mov    eax,0         ;  normal, no error, return value
       ret                  ; return
```

10. Example

printf2.asm more types with printf
The nasm source code is printf2.asm
The result of the assembly is printf2.lst
The equivalent "C" program is printf2.c
Running the program produces output printf2.out

This program demonstrates basic use of "C" library function printf.
The equivalent "C" code is shown as comments in the assembly language.

```
; printf2.asm  use "C" printf on char, string, int, double
;
; Assemble:  nasm -f elf -l printf2.lst  printf2.asm
; Link:      gcc -o printf2  printf2.o
; Run:       printf2
; Output:
;Hello world: a string of length 7 1234567 6789ABCD 5.327000e-30 -1.234568E+302
;
; A similar "C" program
; #include
; int main()
; {
;   char   char1='a';          /* sample character */
;   char   str1[]="string";    /* sample string */
;   int    int1=1234567;       /* sample integer */
;   int    hex1=0x6789ABCD;    /* sample hexadecimal */
;   float  flt1=5.327e-30;     /* sample float */
;   double flt2=-123.4e300;    /* sample double */
;
;   printf("Hello world: %c %s %d %X %e %E \n", /* format string for printf */
;          char1, str1, int1, hex1, flt1, flt2);
;   return 0;
; }


       extern printf              ; the C function to be called
```

```
       SECTION .data                    ; Data section

msg:    db "Hello world: %c %s of length %d %d %X %e %E",10,0
                                 ; format string for printf
char1: db     'a'                ; a character
str1:  db     "string",0         ; a C string, "string" needs 0
len:   equ    $-str1             ; len has value, not an address
inta1: dd     1234567            ; integer 1234567
hex1:  dd     0x6789ABCD         ; hex constant
flt1:  dd     5.327e-30          ; 32-bit floating point
flt2:  dq     -123.456789e300    ; 64-bit floating point

SECTION .bss

flttmp:      resq 1       ;64-bit temporary for printing flt1

      SECTION .text                   ; Code section.

     global  main                 ; "C" main program
main:                            ; label, start of main program

       fld   dword [flt1]    ; need to convert 32-bit to 64-bit
       fstp  qword [flttmp]  ; floating load makes 80-bit,
                             ; store as 64-bit
                             ; push last argument first
       push  dword [flt2+4]  ; 64 bit floating point (bottom)
       push  dword [flt2]    ; 64 bit floating point (top)
       push  dword [flttmp+4]; 64 bit floating point (bottom)
       push  dword [flttmp]  ; 64 bit floating point (top)
       push  dword [hex1]         ; hex constant
       push  dword [inta1]       ; integer data pass by value
       push  dword len            ; constant pass by value
       push  dword str1          ; "string" pass by reference
      push    dword [char1]             ; 'a'
      push    dword msg          ; address of format string
      call    printf                    ; Call C function
      add     esp, 40                   ; pop stack 10*4 bytes

      mov     eax, 0                    ; exit code, 0=normal
      ret                  ; main returns to operating system
```

11. Example

12.