Computer Systems
Lecture 11

Overview

- Output in inline assembly
- Input in inline assembly
- More about `printf`
- More about `scanf`
- Controlling program flow
- Jumps
- Unconditional jumps
- Conditional jumps

Output in inline assembly

```
…
char format[] = "Hello World\n"; // declare variables in C
…
…
lea eax,format      // load address of the string ‘format’ into eax
push eax            // address of string, stack parameter
call printf         // use library code subroutine
add esp,4           // clean 4 byte parameter off stack
```

- We have seen the above fragment implementing the call to `printf` function. Equivalent C code is:

```
printf(“Hello World\n”);
```

Printing numbers

- To print the value of the integer variable '`myint`' one can use the following call to the standard C library routine:

```
printf("%d",myint);
```

- Qualifier "`%d`" means the content of '`myint`' will be printed as a decimal integer.

Printing numbers (cont.)

- To implement such a call in assembly code one can proceed as follows:
  - Push the second parameter (integer variable to the stack).
  - Push the first parameter (actually, address of the string) to the stack.
  - Call `printf` routine.
  - Clean up top two positions in the stack.

Printing numbers (cont.)

```
/* demonstration of the use of asm instructions within C prog  */
#include <stdio.h>
#include <stdlib.h>
int main (void)
{
   char format[] = "%d\n";  // declaration of the format string to
                            // be used in printf function as the first parameter
   int myint = 157;  // declaration of an integer variable
   _asm
   {
     push myint          // push the value of the variable onto the stack
     lea eax,format      // address of the format string is saved in eax
     push eax            // push the address of the string to the stack
     call printf  // call printf, it will take two parameters from the stack
     add esp,8           //clean up top two positions in the stack
   }
   return 0;
}
```

Input in inline assembly

- To input the value into the integer variable '`input`' one can use the following call to the standard C library routine:

`scanf("%d",&input);`

- Qualifier **"%d"** means the input will be read as a decimal integer.
- **`&input`** presents the address of the variable '`input`'.

Implementation in the assembly

```
…
char format[] = “%d”;
int input;    //declaration of an integer variable for user's input
_asm
{
  lea eax,input
  push eax
  lea eax, format  //address of the format string is saved in eax
  push eax         // push the address of the string to the stack
  call scanf       // call scanf, it will take two parameters from
                   // the stack scanf(%d,&input);
                   // user's input will be put in the 'input' variable
  add esp,8        // clean top two positions in the stack
}
…
```

Read a number and print it out

```
int main(void)
{
    char message[] = "Your number is %i\n"; // declaration of the message
    char format[] = "%d";                   // declaration of the format string to
                                            // be used in scanf function as the first parameter
    int input;                              // declaration of an integer variable for user's input
    _asm{
     lea eax,input
     push eax
     lea eax, format                        // address of the format string is saved in ea
     push eax                               // push the address of the string to the stack
     call scanf     // call scanf, it will take two parameters from the stack; scanf(%d,&input);
                                            // user's input will be put in the 'input' variable
     add esp,8                              // clean top two positions in the stack
     push input                             // push the value of the input onto the stack
     lea eax,message                        // address of the message string is saved in eax
     push eax                               // the value of eax is pushed onto the stack
     call printf   //call printf, it will take two parameters from the stack; printf("Your number is %\n", input);
     add esp,8                  // clean top two positions in the stack
    }
   return 0;   // Ex: What is the equivalent C program code (for _asm)?
}
```

More about `printf`

- There are more qualifiers (types) which can be used in `printf`:
  - **`%c`** print a character
  - **`%d`**, or **`%i`** print a signed decimal number
  - **`%s`** print a string of characters

More about `scanf`

- As for `printf` there are more qualifiers one can use in `scanf`:
  - **`%c`** read a single character
  - **`%d`** read a signed decimal integer
  - **`%s`** read a string of characters until a white space or terminator (blank, new line, tab) is found

Controlling program flow

- Very few programs execute all instructions sequentially, from the first till the last one.
- Usually, one needs to control the **flow of the program**:
  - Jump from one point to another, often depending on some conditions.
  - Repeat some actions while some condition is maintained , or until some condition is reached.
  - Passing control to and from procedures.

Jumps

- Jump is the most straightforward way to change program control from one location to another.

- Jump instructions fall into two categories:
  - Unconditional.
  - Conditional.

Unconditional jumps

- JMP instruction transfers control **unconditionally** to another instruction.
- It has a syntax:

  JMP <address of the target instruction>

- The address of the target instruction is given by its label.

Typical use of JMP instruction

```
label1:    …
           …
           …
           jmp label1

label2:    …
           …
           …
           jmp label2
           …
keep-going:
```

- ‘**label1**’, ‘**label2**’ and ‘**keep-going**’ are all labels.
- Unconditional jumps skip over code that should not be executed.

Conditional jumps

- Conditional jumps work as follows:
  - First test the condition.
  - Then jump if the condition is true or continue if it is false.
- There are more than 30 jump instructions:
  - Two of them, JCXZ and JECXZ, test whether the counter register CX, or ECX is zero.
  - Remaining jump instructions test the status flags.

Example

```
        …
        …
        …
        jecxz finish
        mov eax,inp
        …
        …
        …
finish: add esp,4
```

- When '**`jecxz finish`**' instruction is executed: if **`ecx`** register contains `0`, then the next instruction to execute is

  `add esp,4`

- Otherwise,

  `mov eax,inp`

  is executed.

Jumping based on status flags

Instruction	Jump if
JC/JB	Carry flag is set (=1)
JNC/JNB	Carry flag is clear (=0)
JE/JZ	Zero flag is set (=1)
JNE/JNZ	Zero flag is clear (=0)
JS	Sign flag is set (=1)
JNS	Sign flag is clear (=0)
JO	Overflow flag is set (=1)
JNO	Overflow flag is clear (=0)

Jumps Based on Comparison of Two Values

- The **CMP** instruction is the most common way to test for conditional jumps.
- It compares two values without changing them, while it changes the status flags according to the results of the comparison.
- For example, if the value of `eax` and `ebx` are the same then the execution

```
cmp eax,ebx
```

will set zero flag Z=1.

Jumps Based on Comparison

Instruction	Jumps if (assuming execution just after CMP)
JE	The first operand (in CMP) is **e**qual to the second operand.
JNE	The first and second operands are **n**ot **e**qual.
JG	First operand is **g**reater.
JLE	**F**irst operand is **l**ess or **e**qual.
JL	**F**irst operand is **l**ess.
JGE	**F**irst operand is **g**reater, or **e**qual.

An example

```
cmp ax, bx      ; Compare AX and BX
jg  label1      ; Equivalent to: If ( AX > BX )
                ; go to label1
jl  label2      ; Equivalent to: If ( AX < BX )
                ; go to label2
```

Another example

```
add ax,input  ; Add  input to AX
cmp ax,0      ; Compare AX with 0
jge label1    ; Equivalent to:
              ; If ( AX >= 0 ) go to label1
jl  label2    ; Equivalent to:
              ; If ( AX < 0 ) go to label2
```

- Q. To pass two parameters to *printf* in inline assembly code, the first parameter should be pushed onto the stack first. (True or false)

- Q. When calling *printf* in inline assembly, the parameters passed to it will be popped off stack by *printf*. (True or false)

- Q. When calling *scanf* in inline assembly, the address of the variable to receive input needs to be pushed onto the stack. (True or false)

- Q. What is the conversion specifier to be used when printing a string under *printf*?

- Q. If three integer parameters were pushed onto stack when calling ‘scanf’ in inline assembly, how would you adjust the value of register ‘esp’ when returning from ‘scanf’?

- Q. The execution of

```
cmp eax,ebx
```

will check upon the setting of zero flag.
(T or F)

Readings

- [Wil06] Sections 8.4, 8.5.
- http://www.cplusplus.com/ref/cstdio/printf
- http://www.cplusplus.com/ref/cstdio/scanf