KOM206 Laboratory (Dept. of Computer Science, IPB) Module 05 - Mar 11, 2015

Looping instructions

Lecturers: ARD, SWJ *Lab tutors: VDE, WUL*

Objectives

1. Able to explain the loop structure in the assembly language
2. Able to utilize the register CX as a counter
3. Able to create a program with some looping structures

1 Theoretical Review

There are two categories of software loop instructions: an unconditional loop and con ditional loops. Both categories use the (E)CX register as a loop control counter; the counter determines the number of times that the loop will be executed. All loop instructions decrement the count in the (E)CX register by one each time the loop instruction is decoded. If the count is nonzero, then the loop operation is executed; if the count is zero then the loop operation is not executed and program control is transferred to the instruction that immediately follows the loop instruction. If the count in the (E)CX register is zero when the loop instruction is initially decoded, then the counter is decremented to a value of 216 = FFFFH if register CX is used or to a value of 232 = FFFFFFFFH if register ECX is used. In order to avoid this situation, the jump if CX register is 0 (JCXZ) or the jump if ECX register is 0 (JECXZ) should be used.

The loop instructions do not change the state of the flags in the EFLAGS register. If the loop instruction is executed, then the destination address is relative to the contents of the (E)IP register and is characterized as a short jump; that is, within -128 bytes to +127 bytes of the current value in the (E)IP register.

1.1 Unconditional loops

An unconditional loop (LOOP) instruction transfers control to another instruction in the specified range as indicated by a label. The label at the destination address is terminated by a colon, which indicates an instruction within the current code segment. The label name in the LOOP instruction, however, does not have a colon.

If the LOOP instruction does not generate a transfer, then the instruction immediately following the LOOP instruction is executed more quickly than if a transfer occurred. This is because fewer clock cycles are required, since there is no address calculation to determine the destination address; the (E)IP register is simply incremented.

Figure 1 illustrates an unconditional LOOP instruction to transfer control to an instruction with a label specified as NXT_NUM. The program in the loop performs a calculation on numbers in the loop. The body of the loop is executed 20 times.

1.2 Conditional loops

The conditional loop instructions are loop while equal/zero (LOOPE/LOOPZ) and loop while not equal/not zero (LOOPNE/LOOPNZ). The LOOPE and LOOPZ instructions are different mnemonics that refer to the same instruction and they repeat the loop (a short jump) if the (E)CX register is nonzero and the ZF flag is equal to 1. Otherwise, the instruction immediately following the loop instruction is executed.

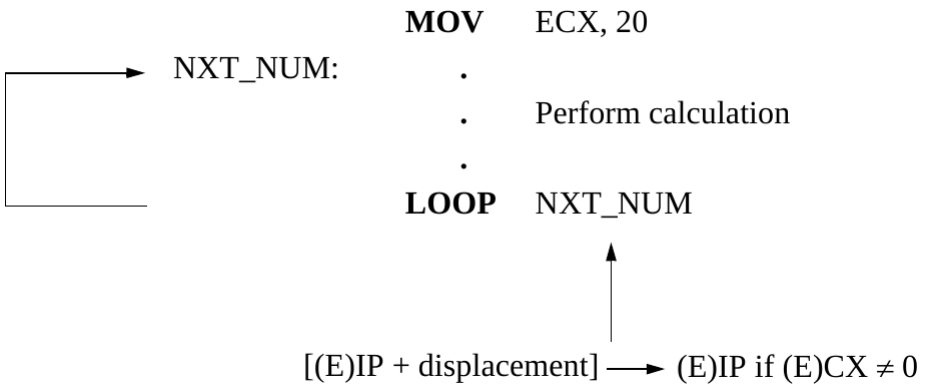

Figure 1: An unconditional LOOP instruction to transfer control to an instruction with a label specified as NXT_NUM.

```
        MOV     ECX, 20
        MOV     EAX, 10
        MOV     EBX, 1

LP1:    ADD     EBX, 1      //add 1 to EBX
        CMP     EAX, EBX    //ZF = 0 for 9 iterations
        LOOPNE  LP1         //loop if ECX ≠ 0 and ZF = 0,
        MOV     rslt, ECX   //otherwise print result
```

Figure 2: A code snippet to show the use of the LOOPNE instruction.

The ZF flag is set by a previous instruction. The LOOPNE and LOOPNZ are different mnemonics that refer to the same instruction and repeat the loop (a short jump) if (E)CX is nonzero and the ZF flag is equal to 0. Otherwise, the instruction immediately following the loop instruction is executed. The ZF is reset by a previous instruction.

The conditional loop instructions use the count in the (E)CX register to determine the number of times to execute the loop; the count in the (E)CX register is decremented by one for each iteration. None of the flags are affected by these conditional loop instructions.

Figure 2 shows the use of the LOOPNE instruction. The count in register ECX is initially set to a value of 20, register EAX is set to a value of 10, and register EBX is set to a value of 1. The conditional loop repeats while register $ECX \neq 0$ and the zero flag $ZF = 0$. A value of 1 is added to register EBX with each iteration of the loop. After nine iterations, the values in registers EAX and EBX are equal. Therefore, even though the count in register ECX is nonzero (11 10), the ZF flag is set to a value of 1, which results in the termination of the loop.

1.3 Yet another GDB tricks

If you need to execute occasional shell commands during your debugging session, there is no need to leave or suspend GDB; you can just use the shell command.

```
(gdb) make clean
rm -f *.o *.lst main *~ .*.gdb
(gdb) make
yasm main.asm -f elf32 -g dwarf2 -l main.lst
ld -o main -e main -m elf_i386 main.o
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
(gdb) where
```

Some logging settings:

```
(gdb) set logging [on|off]
Copying output to gdb.txt.
(gdb) set logging overwrite [on|off]
By default, gdb will append to the logfile.
```

gdb executes file .gdbinit after running. This means that you can add some common commands to .gdbinit file as follows.

```
# @file: .gdbinit (put in your home dir)
file main

set logging file gdb.txt
set logging on
set logging overwrite on

b main
b exit
```

2 Tasks

1. Write a program to double the value of an integer six times (at each iteration, the integer resulted from the previous iteration is doubled). Do not use any MUL/IMUL instructions. Use some looping instructions. Test cases: $value = 1$ yields $result = 64$; $value = 2$ yields $result = 128$; $value = 5$ yields $result = 320$; $value = 25$ yields $result = 1600$;

2. Write a program to reproduce sequence of iterations shown in figure 3.

3. Implement a while-loop. Write a program that takes a (harcoded) number i, where $i \in 1, 2, ..., 9$. The program does a number of loops until the value in EAX is of 11. In each loop the value of EAX register is incremented by one.

4. Implement a for-loop. Write a program to reproduce a sequence of iterations shown in figure 4.

5. Write an assembly program to perform a "find and replace" operation on a string in memory. Make your program replace every occurrence of "amazing" with "incredible".

6. Write an assembly program to sort an array of double words using bubble sort. Bubble sort is defined as in listing 1.

Iteration	EAX	EBX	ZF (CMP EAX, EBX)	ECX
0	10	1	0	20
1	10	2	0	19
2	10	3	0	18
3	10	4	0	17
4	10	5	0	16
5	10	6	0	15
6	10	7	0	14
7	10	8	0	13
8	10	9	0	12
9	10	10	1	11

Figure 3: A sequence of iterations produced by a looping program.

Iteration	ECX	EAX
0	3	0
1	4	1
2	5	2
3	6	3
4	7	4
5	8	5
6	9	6
7	10	7
8	11	8

Figure 4: A sequence of iterations produced by a for-loop.

7. Write an assembly program to compute Fibonacci numbers storing all the computed Fibonacci numbers in a quad-word array in memory. (What is the largest i for which you can compute $fib(i)$?) Fibonacci numbers are defined by

$$fib(O) = 0$$

$$fib(1) = 1$$

$$fib(i) = fib(i-1) + fib(i-2), \forall i > 1.$$

Listing 1: The Data Bubble sort

```
1 do {
2     swapped = false ;
3     for ( i = 0 ; i < n- 1 ; i++ ) {
4         if ( a [i] > a [i+1] } {
5             swap a [i] and a [i+1]
6             swapped = true ;
7         }
8     }
9 } while ( swapped ) ;
```

3 Homework

There is no homework. Yeeayy.

4 Miscellany

The content is mainly based on the previous lab module and the following resources: chapter 8 of [1], chapter 6 of [2] and chapter 6 of [3].
(compiled on 05/03/2015 at 12:56 Noon)

References

[1] R. Seyfarth, *Introduction to 64 Bit Intel Assembly Language Programming for Linux*. CreateSpace, 2012.

[2] K. Irvine, *Assembly Language for X86 Processors*. Pearson Education, Limited, 2011. [Online]. Available: http://books.google.co.id/books?id=0k20RAAACAAJ

[3] J. Cavanagh, *X86 Assembly Language and C Fundamentals*. CRC Press, 2013.