CS-UY 2214 — Recitation 14

Introduction

Complete the following exercises. Unless otherwise specified, put your answers in a plain text file named `recitation14.txt`. Number your solution to each question. When you finish, submit your file on Gradescope. Then, in order to receive credit, you must ask your TA to check your work. Your work should be completed and checked during the recitation session.

The slides, available on Brightspace, cover many useful Intel opcodes. You may consult the online Intel assembly language reference to help you understand any unknown opcodes. You can also use the official Intel developer's manual.

Problems

1. What is endianism? Describe with an example.
2. Consider the following fragment of an Intel assembly language program:

```
numbers:
    dd 0x34, 0xc99, 0x45, 0xaa52
```

   (a) The instruction `mov eax, [numbers]` is executed. What value will be stored in `eax`? Give your answer as a hexadecimal value.

   (b) The instruction `mov ax, [numbers]` is executed. What value will be stored in `ax`? Give your answer as a hexadecimal value.

   (c) The instruction `mov eax, [numbers+4]` is executed. What value will be stored in `eax`? Give your answer as a hexadecimal value.

   (d) The instruction `mov eax, [numbers+4]` is executed. What value will be stored in `ah`? Give your answer as a hexadecimal value.

   (e) The instruction `mov ax, [numbers+1]` is executed. What value will be stored in `ax`? Give your answer as a hexadecimal value.

   (f) The instruction `mov eax, [numbers+1]` is executed. What value will be stored in `eax`? Give your answer as a hexadecimal value.

   (g) The instruction `mov bh, [numbers+5]` is executed. What value will be stored in `bh`? Give your answer as a hexadecimal value.

   (h) The instruction `mov eax, numbers` is executed. What value will be stored in `eax`?

3. The following Intel assembly language code fragment examines the value of `eax` and leaves a value in `ebx`. What function does this code calculate, i.e. what is the final value of `ebx` relative to the initial value of `eax`?

```
    xor ebx, ebx
loop_top:
    cmp eax, 0
    je end_of_program
    mov ecx, eax
    shr eax, 1
    and ecx, 1
    jz loop_top
    inc ebx
    jmp loop_top
end_of_program:
```

Hint: the `and` opcode, like most arithmetic opcodes, sets the zero flag to 1 if its result is zero.

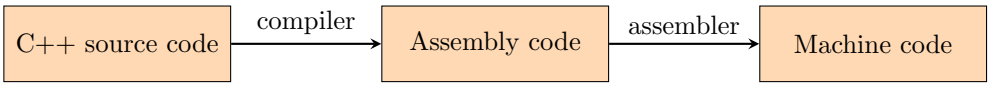
4. You are given a data structure consisting of an array of unsigned dword values, starting at a memory location identified by the label `num_array`. After the last element in array, the end of the array is marked with a dword of value zero. You may assume that no element within the array is zero. That is:

```
num_array:
    dd 45
    dd 99
    dd 8190
    ; ....      some unknown number of array elements
    dd 0      ; end of array
```

Write an Intel assembly language program fragment that will (a) store the sum of the array in `eax` and (b) store the largest array value and `ebx`. Your solution must use a loop.

Hint: remember that in order to compare unsigned integers, you should use `cmp` followed by `jb` (jump if below), `ja` (jump if above), `jbe` (jump if below or equal), or `jae` (jump if above or equal).

5. As you know, a *compiler* is a piece of software that acts as a translator: it translates source code into assembly language code, which can then be converted into machine code, which can subsequently be executed by the processor:



When you've used a C++ compiler in the past, the translation process was largely automated. However, we can "open the lid" and see how the translation works. In particular, we want to see the exact relationship between the input (C++ source code) and output (x86 assembly language code).

Consider the following simple C++ program. Enter this program into your computer in a file named `test.cpp`.

```
int i;
int arr[10];

int main() {
    for (i=0; i<10; i++)
        arr[i]=i*100;
    return 0;
}
```

Compile this program on Linux using the `g++` compiler and the command shown below. Note that the whole command should be entered on a single line; the arrows indicate that the command continues.

```
g++ -S -march=x86-64 -m32 -O0 -masm=intel -fno-asynchronous-unwind-tables
    ↪ -fno-dwarf2-cfi-asm -g0 -fno-pie test.cpp
```

Please note that you must use exactly the above command or else you may get unexpected results. Note that the command must be entered all together on one line. If this command doesn't work for you, make sure that you've entered it correctly and make sure you are using the Linux `g++` compiler (as found on Anubis and Vital), and not some other compiler.

The above command should produce a file named `text.s`. You can open this file with your text editor. The file contains the assembly language produced from the C++ source code.

Examine the assembly code and try to understand how it works. Note that the dialect of assembly differs somewhat from what we've learned in class. For example, you may see `DWORD PTR`: this is a type specifier that expresses the size of the operation, and is usually optional. There will be many directives (beginning with a period) that you won't recognize. That's okay: even so, the code should be understandable. Most unknown directives can be ignored.

Answer the following questions about the produced assembly code in `test.s`.

(a) What pair of assembly language instructions are used to test if the loop has reached its last iteration? Explain your answer.

(b) The program has a dword-sized global variable named `i`. However, `g++` does not use the conventional `dd` directive to allocate memory for it. What directive allocates memory for the variable `i` in the produced assembly language?

Explain how you determined your answer.

(c) The program has a global array variable named `arr`. What directive allocates memory for the variable `arr` in the produced assembly language?

Explain how you determined your answer.

(d) The program multiplies the value of `i` by 100. Which instruction performs this multiplication in the produced assembly language?

Explain how you determined your answer.

(e) After multiplying `i` by 100, the program stores the product at the ith element of the array `arr`. Which instruction performs this storage in the produced assembly language?

Explain how you determined your answer.

(f) The program contains a loop, which terminates when the counter reaches 10. Which instruction or instructions break out of the loop?

Explain how you determined your answer.

(g) When the program ends, it doesn't invoke the `sys_exit` syscall. Instead, the `main` function simply returns. Which instruction corresponds to the function return?

Explain how you determined your answer.