Machine Programming with Assemble

COMP201 Lab Session
Fall 2020

KOÇ UNIVERSITY

Assembly Language

- Low-level programming language
- Designed for a specific type of processor
- It may be produced by compiling source code from a high-level programming language (such as C/C++)
- It can also be written from scratch.
- Assembly code can be converted to machine code using an assembler.

Assembly Language

- Assembly languages differ between processor architectures
- They often include similar instructions and operators
- Below are some examples of instructions supported by x86 processors:
  - MOV - move data from one location to another
  - ADD - add two values
  - SUB - subtract a value from another value
  - PUSH - push data onto a stack (will be covered in this week's lectures
  - POP - pop data from a stack (will be covered in this week's lectures)
  - JMP - jump to another location
  - INT - interrupt a process

Registers

- Registers are data storage locations directly on the CPU
- Usually, the size, or width, of a CPU's registers define its architecture
- In a 64-bit CPU, the registers will be 64 bits wide
- The same is true of 32-bit CPUs (32-bit registers), 16-bit CPUs, and so on.
- Registers are very fast to access and are often the operands for arithmetic and logic operations.
- rbp and rsp are special purpose registers
  - rbp is the base pointer, which points to the base of the current stack frame
  - rsp is the stack pointer, which points to the top of the current stack frame
  - rbp always has a higher value than %rsp because the stack starts at a high memory address and grows downwards.

Understanding Assembly

Consider the following Assembly code:

```
push   rbp
mov    rbp, rsp
mov    DWORD PTR [rbp-4], edi
mov    eax, DWORD PTR [rbp-4]
imul   eax, eax
pop    rbp
ret
```

Understanding Assembly

Normally these are the first 2 instructions of all Assembly codes:

```
push   rbp
mov    rsp, rbp
```

- The first two instructions are called the function prologue or preamble.
- First we push the old base pointer onto the stack to save it for later.
- Then we copy the value of the stack pointer to the base pointer.
- After this, %rbp points to the base of main‘s stack frame.

Understanding Assembly

```
mov     DWORD PTR [rbp - 4], edi
```

- The first integer argument is passed in the rdi/edi register.
- So this line copies the argument to a local (offset -4 bytes from the frame pointer value stored in rbp).

```
mov eax, DWORD PTR [rbp-4]
```

- This copies the value in the local to the eax register.

KOÇ UNIVERSITY

Understanding Assembly

imul   eax, eax

- Multiply the contents of eax register with eax register

pop   rbp

- pop original register out of stack

ret

- return

Let’s Revisit

```
push    rbp
mov     rbp, rsp
mov     DWORD PTR [rbp-4], edi
mov     eax, DWORD PTR [rbp-4]
imul    eax, eax
pop     rbp
Ret
```

Yes, it is just simple squaring function:

```
int square(int num) {
    return num * num;
}
```

KOÇ UNIVERSITY

Try to understand this!

```
weirdProduct(int, int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        mov     DWORD PTR [rbp-8], esi
        add     DWORD PTR [rbp-4], 1
        sub     DWORD PTR [rbp-8], 1
        mov     eax, DWORD PTR [rbp-4]
        imul    eax, DWORD PTR [rbp-8]
        pop     rbp
        ret
```

References

[1] “Assembly Language,” *Assembly Language Definition*. [Online]. Available: https://techterms.com/definition/assembly_language . [Accessed: 21-Nov-2020].

[2] "Understanding C by learning assembly - Blog - Recurse Center", Recurse Center, 2020. [Online]. Available: https://www.recurse.com/blog/7-understanding-c-by-learning-assembly#:~:text=%25rbp%20is%20the%20base%20pointer,memory%20address%20and%20grows%20downwards . [Accessed: 21- Nov- 2020].