X86 assembly quick tutorial

Memory model
- Real mode (address up to $2^{20}$ bytes)
- Memory access is done by
  \[
  \text{Segment:offset}
  \]
  \[
  \begin{array}{c}
  \text{Segment 16 bits} \\
  + \hspace{1cm} \text{Offset 16 bits}
  \end{array}
  \]
  \[
  \text{Real address = 20 bits}
  \]

Memory access
- Remember the default segment register
- Data access: (general registers)
  \[
  \text{Movw} \hspace{0.2cm} (\%\text{si}), \hspace{0.2cm} \%\text{ax}
  \]
  \[
  = \hspace{0.2cm} \text{movw} \hspace{0.2cm} \%\text{ds}: (\%\text{si}), \hspace{0.2cm} \%\text{ax}
  \]
- Stack access: (\%bp, \%sp)
  \[
  \text{Movw} \hspace{0.2cm} 4(\%\text{bp}), \hspace{0.2cm} \%\text{ax}
  \]
  \[
  = \hspace{0.2cm} \text{movw} \hspace{0.2cm} \%\text{ss} : 4(\%\text{bp}), \hspace{0.2cm} \%\text{ax}
  \]

Memory access (cont)
- Code access:
  Normally you do not explicitly change \%IP.
  Use jmp, jz, call etc instead.
- Short jump: \text{jmp label}
  actually this is to jmp \%cs, offset of label
- Long jump: \text{ljmp NEW_CS, offset}
Calling convention (Gcc style)

```c
int foo2(int n) {
    return n + 2;
}
int foo( int n ) {
    return foo2(n - 1);
}
int main (void) {
    return foo( 5);
}
```

C -> Assembly (passing parameter)

```assembly
foo2:
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %eax
addl $2, %eax
leave
ret
```

C -> assembly: (local variable)

```c
int foo3(int n)
{
    int i;
    i = n + 2;
    return i;
}
```

Stack view

```
pushl %ebp
movl %esp, %ebp
subl $4, %esp
movl 8(%ebp), %eax
addl $2, %eax
movl %eax, -4(%ebp)
movl %eax, -4(%ebp)
leave
ret
```
Calling a function

pushw %ax
call foo2

(%ax can be accessed by 8(%bp) within function)

-4(%ebp)

%ebp

%ss: 0

Note for project 1

- In our project, the bootloader is working in real mode (16 bits).
- The gcc example given earlier is compiled in 32 bits mode.
- So beware of the difference of accessing the calling parameter:
  - 32 bits -> 8(%ebp)
  - 16 bits -> 4(%bp)

More notes for bootloader

- Bootloader code is loaded by BIOS, so it did not have %ds, %ss, %sp setup properly when it is loaded.
- You shall put strings and extra instructions after “over:” so that BIOS will not run into those code.
- In bootloader, all the code and data share the same 512 bytes. So data will have the same segment as code.