Lec11 Assembly Language Programming

High Level Languages

- More programmer friendly()
- More ISA(instruction set architecture) independent
- Each high-level statement translates to several instructions in the ISA of the computer

Assembly Languages

- Lowerlevel, closer to ISA
- Very ISA-dependent
- One assembly language instruction is translated into one machine instruction by the assembler
- Make slow level programming more user friendly
- More efficient code

Assembly Language Programming

Each lines has 4 fields:

```
[label]  mnemonic  operand  list  comment
```

1. [label] (Oprional):
   i. A label is an optional field used to identify a specific memory location or instruction.
   ii. Labels are often used as references for jumps, loops, or data locations.
   iii. Labels typically end with a colon (:).
2. Memoric:
   i. The mnemonic is the core part of the instruction, representing the operation to be performed by the CPU.
   ii. Mnemonics are short, human-readable codes that correspond to machine instructions.
3. Operand List
   i. The operand list specifies the data or addresses that the instruction operates on.
   ii. Operands can include registers, memory addresses, constants, or labels.
   iii. Some instructions may have no operands, one operand, or multiple operands.
   iv. There are many types of rules:
      a. 立即数 (Immediate Value)

        a. Ususlly use #+value to express immediate value.

```
MOV AX, #5       # 将立即数 5 加载到寄存器 AX 中
```

        b. In some assembly language, just need to use value

```
ADD CX, 10       # 将立即数 10 加到寄存器 CX 中
```

    b. 寄存器 (Register)

    c. 内存地址 (Memory Address)

        a. 直接寻址 (Direct Addressing)

```
MOV AX, [1234]   # 将内存地址 1234 处的值加载到寄存器 AX 中
```

        b. 间接寻址 (Indirect Addressing)

```
MOV AX, [BX]     ; 将寄存器 BX 中存储的地址指向的内存值加载到 AX 中
```

        c. 基址+偏移量寻址 (Base + Offset Addressing)

```
MOV AX, [BX + SI] ; 将寄存器 BX 和 SI 的值相加作为内存地址，加载该地址的值到 AX 中
```

    d. 段地址 (Segment Address)

    在某些架构（如 x86）中，内存地址分为段地址和偏移地址。

```
MOV AX, [DS:1234] ; DS 是段寄存器，1234 是偏移地址
```

    e. 不同的汇编语言可能对操作数有特殊的符号或约定。以下是一些常见规则：

        a. #：表示立即数（如 #5 表示立即数 5）。

        b. []：表示内存地址（如 [1234] 表示内存地址 1234）。

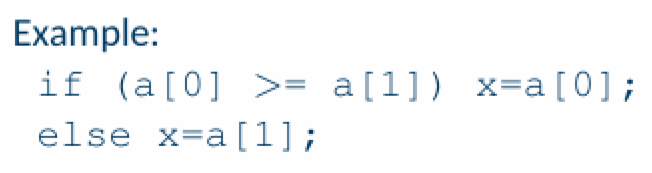
        c. ()：表示偏移量（如 4(R1) 表示寄存器 R1 的值加上 4）。

        d. ： 在某些汇编语言中， 表示立即数（如 $5 表示立即数 5）。

        e. %：在某些汇编语言中，% 表示寄存器（如 %eax 表示寄存器 EAX）。

4. Assembler Directives

Directives	Description
.data	Tells the assembler to add all subsequent data to the
.text	Tells the assembler to add subsequent code to the text
.globl name	Makes name external to other files, for multiple files
.space expression	Reserves space, amount specified by the value of expre
.word value1[, value2]	Puts the values in successive memory locations.

> examples
>
> 
>
> Example:
>
> ```
> if (a[0] >= a[1]) x=a[0];
> else x=a[1];
> ```

```
      .data    # data segment
 a:   .word 1 # create storage containing a[0]=1
      .word 3 # create storage containing a[1]=3
 x:   .word 4 # create storage containing 4, x=4
      .text    # program segment
main:
      ld #a, r8       # r8 = address of a (#a)
      ld 0(r8),r9     # r9 = a[0]=1
      ld 4(r8),r10    # r10= a[1]=3
      bgt r9,r10,f1 # branch if r9>r10, goto f1
      st r10,x        # x=r10
      br f2           # goto f2
f1:   st r9,x         # x=r9
f2:   ret             # return to OS (same as return in C++)
```

Example:

```
a=0;
for (i=0; i<10; i++) a+=i;
```

```
        .data
 a:     .word 0
        .text
 main:
        sub r8,r8,r8        # r8=0, or xor r8,r8,r8
        ld #0xa,r9          # r9=Oxa=10, no.  of iterations
        sub r10,r10,r10     # r10=0, loop counter
        ld #1,r11           # r11=1
 f1:    add r8,r10,r8       # r8+=r10
        add r10,r11,r10     # r10++, increment counter
        bgt r9,r10,f1       # if (r9>r10) goto f1
        st r8,a             # a=r8
        ret                 # return
```

Example:

```
temp=0;
a=1;
while (temp < 100){
  temp+=a;
  a++;
}
```

```
     # fill in the .data, .text part as in
     # previous examples
     sub r8,r8,r8    # r8 is temp, r8=0
     ld #1,r9        # r9 is a, r9=1
     mv r9,r10       # r10=r9
     ld #0x64,r11    # r11=100
f1:  add r8,r9,r8    # temp+=a
     add r9,r10,r9   # a++
     blt r8,r11,f1   # if (r8<r11) goto f1
```

Convert all characters into upper case letters.

```
              .data
      a:      .asciiz "This is a test"
              # zero-terminated string

              .text
      main:   sub r9,r9,r9      # r9=0
      loop:   lb a(r9),r10      # load byte
              beq r10,#0,exit   # r10==0?  end of string
              call capitalize   # call capitalize
              sb r10, a(r9)     # store result back
              add r9, #1,r9     # incr r9, next char
              br loop           # goto loop
      exit:   ret               # return
```

Suppose that characters a to z are represented by bytes 0x61 to 0x7a, and capital characters A to Z are represented by 0x41 to 0x7a.

```
        Capitalize:
        #input is r10 ,output is r10, if r10 is lower
        #case letter, change to upper case
               push r8
               push r9
               ld #0x61,r8         #r8='a'
               ld #0x7a,r9         #r9='z'
               blt r10,r8,ret1
               bgt r10,r9,ret1
               sub r10,#0x20,r10   #0x20='a'-'A'
        ret1:  pop r9
               pop r8
               ret                 # return
```