User Id : weizhe.goh@digipen.edu | Started : 2019.11.28 11:20:59 | Score :

DigiPen CS Vadim Surov CS100 Presentation #18

Assembler - Arrays

This presentation guides you through working with null-terminated strings using assembler.

1. Strings

- The following code sets "ABC" to a string

Run

```
.macro PRINT fmt, v
    mov   \fmt, %edi
    mov   \v, %esi
    xor   %eax, %eax # Clear AL
    call  printf
.endm

    .data
str: .skip  10
fmt: .asciz "%s"
    .text
    .global main
main:
    push %rbx # For alignment

    mov  $str, %rax
    movb $65,  0(%rax)
    movb $66,  1(%rax)
    movb $67,  2(%rax)
    movb $0,   3(%rax)

    PRINT $fmt, $str

    xor  %eax, %eax # return 0;
    pop  %rbx
    ret
```

2. Comments

- mov copies the first operand to the second.
- Second operand is an example of **inderect addressing** using base address given in register %rax plus n displacement bytes.
- For strings or arrays of bytes n(reg) in asm is equivalent to reg[n] or *(reg+n) in c.
  - 0 in 0(reg) can be omitted: (reg)
- There are up to 4 parameters of an address operand that are presented in the syntax

displacement(**base register**, index **register**, scale factor)

- Example: movl -8(%ebp, %edx, 4), %eax
  load a value by address (-8 + ebp + edx*4) into %eax
- Todo: Print out ASCII table

3. Copy

- The following assembly code copies string src to string dst character by character:

Run

```
    .data
src: .asciz "abc"
dst: .skip  4
fmt: .asciz "%s"
    .text
    .global main
main:
    push  %rbx # For alignment

    mov $src, %rax
    mov $dst, %rbx
    movb 0(%rax), %cl
    movb %cl, 0(%rbx)
    movb 1(%rax), %cl
    movb %cl, 1(%rbx)
    movb 2(%rax), %cl
    movb %cl, 2(%rbx)
    #movb $0, 3(%rbx)

    PRINT   $fmt, $dst

    xor   %eax, %eax # return 0;
    pop   %rbx
    ret
```

```
jdoodle.s: Assembler messages:
jdoodle.s:20: Error: no such instruction:
```

4. Comments

- movb can copy one byte from
    - memory to register, or
    - register to memory.
- That's why to copy from memory to memory we use temp register %cl.
- Todo: make copy using loop (or loopnz)

5. Copy

- The following code copies 4 characters of string src to dst:

Run

```
.macro PRINT fmt, v
    mov   \fmt, %edi
    mov   \v, %esi
    xor   %eax, %eax # Clear AL
    call    printf
.endm

    .data
src: .asciz "abcdefg"
dst: .skip  10,0
fmt: .asciz "%s"
    .text
    .global main
main:
    push %rbx # For alignment

    movq src, %rcx
    movq %rcx, dst

    PRINT $fmt, $dst

    xor  %eax, %eax # return 0;
    pop  %rbx
    ret
```

```
abcdefg
```

5. Comments

- Using movq copies 8 bytes in a single instruction.

6. Compare

- The following code compares two strings and returns 1 when strings are equal:

Run

```
.macro PRINT fmt, v
    mov   \fmt, %edi
    mov   \v, %esi
    xor   %eax, %eax # Clear AL
    call    printf
.endm

    .data
s1: .asciz "abc"
s2: .asciz  "abc"
fmt: .asciz "%d"
    .text
    .global main
main:
    push %rbx # For alignment

    mov $s1, %rsi
    mov $s2, %rdi

    cld
    cmpsl
    je equal
    PRINT   $fmt, $0
    jmp end
equal:
    PRINT $fmt, $1

end:
    xor  %eax, %eax # return 0;
    pop  %rbx
    ret
```

```
1
```

7. Comments

- cmps - family of instructions is used to compare string values. The locations of the implied source and destination operands are stored in the %rsi and %rdi registers.
- Each time the instruction is executed, the %rsi and %rdi registers are incremented or decremented by the amount of the data size compared.
- The cld / std instruction clear / set the DF (Direction Flag), that defines what to do with %rsi and %rdi incremet or decrement.

8. Compare

- The following inline code compares first 12 characters of two strings:

Run

```
.macro PRINT fmt, v
    mov   \fmt, %edi
    mov   \v, %esi
    xor   %eax, %eax # Clear AL
    call    printf
.endm

    .data
s1: .asciz "Hello World!"
s2: .asciz  "Hello world!"
fmt: .asciz "%d"
    .text
    .global main
main:
    push %rbx # For alignment

    mov $s1, %rsi
    mov $s2, %rdi
    mov $12, %rcx

    cld
    repe cmpsb

    PRINT   $fmt, %ecx

end:
    xor  %eax, %eax # return 0;
    pop  %rbx
    ret
```

```
5
```

9. Comments

- The program loads the source and destination string locations into the %rsi and %rdi registers, as well as the string length in the %rcx register.
- The repe cmpsb instructions repeat the string compare byte by byte until either the %rcx register runs out or the zero flag is set, indicating a nonmatch.
- The %rcx register will contain the position of the mismatched character (counting back from the end of the string).
- This example also can demonstrate how sensitive the string comparisons are. The two strings differ only in the capitalization of one character, which will be detected by the comparison:

10. Scanning

- The following inline code scans for character W in the given string and returns 12-position of the character in the string, or nothing if the character is not found:

Run

```
.macro PRINT fmt, v
    mov   \fmt, %edi
    mov   \v, %esi
    xor   %eax, %eax # Clear AL
    call    printf
.endm

    .data
str: .asciz "Hello World!"
fmt: .asciz "%d"
    .text
    .global main
main:
    push %rbx # For alignment

    mov  $12, %rcx
    mov  $'W', %rax
    mov  $str, %rdi

    cld
    repne scasb
    jne   end

    PRINT $fmt, %ecx

end:
    xor  %eax, %eax # return 0;
    pop  %rbx
    ret
```

```
5
```

11. Comments

- The repne scasb instruction is used to scan the string (in %rdi) for the location of the search character (in %rax).
- If the character is found, its location (actually, 12 - location - 1) is now in %rcx.
- Todo: count occurrences of characters in string

12. Length

- To find length of a null-terminated string scanning for character 0 can be used.

Run

```
.macro PRINT fmt, v
    mov   \fmt, %edi
    mov   \v, %esi
    xor   %eax, %eax # Clear AL
    call    printf
.endm

    .data
str: .asciz "Hello World!"
fmt: .asciz "%d"
    .text
    .global main
main:
    push %rbx # For alignment

    mov $0, %rcx
    mov $str, %rdi
next:
    xor  %rax, %rax
    movb (%rdi), %al
    cmp  $0, %al
    je   end

    inc  %rdi
    inc  %rcx
    jmp  next

end:
    PRINT $fmt, %ecx

    xor  %eax, %eax # return 0;
    pop  %rbx
    ret
```

```
12
```

13. Comments

- To find length of a null-terminated string scanning for character 0 using repne scasb can be used.
- To do that set the counter register with a value bigger than real length of the string.
- Todo: try above

14. Inline assembler

- The following code removes spaces in the given string using inline assembler:

Run

```
/* Remove all spaces */
#include <stdio.h>
char str[] = " Hello World!";
int main(void) {
  __asm__ (
  "mov  %[str],    %%rdi   \n\t"
 "next:                    \n\t"
  "movb  (%%rdi), %%al     \n\t"
  "cmp   $0,       %%al    \n\t"
  "je    end               \n\t"
  "cmp   $32,      %%al    \n\t"
  "je    shiftleft         \n\t"
  "inc   %%rdi             \n\t"
  "jmp   next              \n\t"
 "shiftleft:               \n\t"
  "mov  %%rdi,     %%rsi   \n\t"
 "shiftleftagain:          \n\t"
  "inc   %%rsi             \n\t"
  "movb  (%%rsi), %%al     \n\t"
  "cmp   $0,      %%al     \n\t"
  "je    shiftend          \n\t"
  "movb  (%%rsi), %%al     \n\t"
  "movb  %%al, -1(%%rsi)   \n\t"
  "jmp   shiftleftagain    \n\t"
 "shiftend:                \n\t"
  "movb  $0, -1(%%rsi)     \n\t"
  "jmp   next              \n\t"
 "end:                     \n\t"
  :
  : [str]"r"(str)
  : "rdi", "rsi", "al"
  );
  printf("%s\n", str);
  return 0;
}
```

```
HelloWorld!
```

15. More To Do

- Clear. Erases the contents of the string, which becomes an empty string (with a length of 0 characters).
- Swap. Exchanges the content of two strings. Lengths may differ
- Substring. Returns a substring of a given string. The substring is the portion of the string that starts at character position pos and spans len characters (or until the end of the string, whichever comes first).
- Trim. Remove leading and trailing whitespace.
- Append. Extends the given string by appending at the end additional characters from the second string.
- Insert. Inserts additional characters into the string right before the indicated character.
- Sort characters in different orders

100 References

Manual – The GNU Assembler manual

By signing this document you fully agree that all information provided therein is complete and true in all respects.	Responder sign: