上机作业实验报告

1820211062 洪子翔

作业目的及要求

大数相乘。要求实现两个十进制大整数的相乘（100 位以上），输出乘法运算的结果。

实验环境

使用 x86 架构的汇编语言，在 Visual Studio 2019 中进行开发。

作业步骤以及代码解析

实现大数相乘作业的程序需要考虑输入输出以及进行大数相乘的代码。首先导入需要使用到的库,输入部分要求接收两个超过100位的十进制大数，因此需要使用 `scanf` 函数进行输入；而输出部分则使用 `printf` 函数进行输出。

```
.model flat, stdcall
option casemap:none

includelib msvcrt.lib
printf PROTO C :ptr sbyte, :VARARG
scanf PROTO C :ptr sbyte, :VARARG
strlen PROTO C :ptr sbyte, :VARARG
```

接下来是对数据的定义。`input_s byte` 用作后续 `scanf` 函数的输入格式。定义的 `input_str1` 和 `input_str2` 用于存储接收到的输入字符串。为了进行运算，需要将输入的字符串转换成数组，因此定义了双字的 `input_num1` 和 `input_num2`。为了存储结果，设置了 `output_num` 和 `output_str` 用于输出。最后，对每个字符串的长度进行初始化。`radix` 变量用于指定进制。

```
.data
input_s byte "%s", 0
input_str1 byte 200 dup(0)
input_str2 byte 200 dup(0)
output_str byte 200 dup(0)
input_num1 dword 200 dup(0)
input_num2 dword 200 dup(0)
output_num dword 200 dup(0)

len_input1 dword 0
len_input2 dword 0
len_output dword 0

radix dword 10
```

然后进入函数部分，大数相乘经需要三个函数来进行完成，他们分别是字符串转数组函数、数组转字符串函数和大数相乘函数。

str_to_num

首先我们先看字符串到整数转换函数（`str_to_num` 过程）。这个函数通过遍历输入字符串，将每个字符转换为对应的数字，并通过栈的方式将这些数字存储在整数数组中。

通过 `mov ecx, len` 将传入的字符串长度保存在 `ecx` 寄存器中，并通过 `mov esi, str_in` 将字符串的起始地址保存在 `esi` 寄存器中。然后，使用标签 `L1` 进入循环，通过 `movzx eax, byte ptr [esi]` 获取字符串中当前字符的 ASCII 值，减去 '0' 的 ASCII 值（`sub eax, 30h`）得到对应的数字。将得到的数字通过 `push eax` 压入栈中。通过 `inc esi` 指向下一个字符，使用 `loop L1` 判断是否还需要继续循环。接着，转到 `L2` 标签，通过 `pop eax` 弹出栈中的数字，然后通过 `mov dword ptr [esi], eax` 将数字存储在输出的整数数组中。通过 `add esi, 4` 移动到下一个位置，使用 `loop L2` 判断是否还需要继续循环。最终，通过 `ret` 返回。该函数的参数包括输入字符串的地址 `str_in`、输出整数数组的地址 `num_out`，以及字符串的长度 `len`。

```
str_to_num proc stdcall str_in :ptr byte, num_out :ptr dword, len :dword

  mov ecx, len
  mov esi, str_in

  L1:
    movzx eax, byte ptr [esi]
    sub eax, 30h
    push eax
    inc esi
    loop L1

  mov ecx, len
  mov esi, num_out
  L2:
    pop eax
    mov dword ptr [esi], eax
    add esi, 4
    loop L2

  ret
str_to_num endp
```

num_to_str

接着是整数到字符串转换函数（`num_to_str` 过程）。这个函数将整数数组中的每个数字加上 '0' 的 ASCII 值，并通过栈的方式将这些字符存储在输出字符串中。

通过 `mov ecx, len` 将传入的整数数组的长度保存在 `ecx` 寄存器中，并通过 `mov esi, num_in` 将整数数组的起始地址保存在 `esi` 寄存器中。然后，使用标签 `L1` 进入循环，通过 `mov eax, dword ptr [esi]` 获取整数数组中当前数字。将得到的数字通过 `add eax, 30h` 加上 '0' 的 ASCII 值，然后通过 `push eax` 压入栈中。通过 `add esi, 4` 移动到下一个位置，使用 `loop L1` 判断是否还需要继续循环。接着，转到 `L2` 标签，通过 `pop eax` 弹出栈中的字符，然后通过 `mov byte ptr [esi], al` 将字符存储在输出的字符串数组中。通过 `inc esi` 移动到下一个位置，使用 `loop L2` 判断是否还需要继续循环。最终，通过 `ret` 返回。该函数的参数包括输入整数数组的地址 `num_in`、输出字符串的地址 `str_out`，以及整数数组的长度 `len`。

```
num_to_str proc stdcall str_out :ptr byte, num_in :ptr dword, len :dword

  mov ecx, len
```

```
  mov esi, num_in
  L1:
    mov eax, dword ptr [esi]
    add esi, 4
    push eax
    loop L1

  mov ecx, len
  mov esi, str_out
  L2:
    pop eax
    add eax, 30h
    mov byte ptr [esi], al
    inc esi
    loop L2

  ret
num_to_str endp
```

big_num_mul

最后是大整数乘法函数（`big_num_mul` 过程）。这个函数通过两个嵌套循环遍历两个输入整数数组，在每一步乘法后，将得到的结果存储在输出整数数组中，同时考虑进位。

通过 `mov ecx, len_input1` 将第一个输入整数数组的长度保存在 `ecx` 寄存器中，并使用 `xor edi, edi` 清零 `edi` 寄存器，用于循环计数。然后，通过标签 `L1` 进入外层循环，表示对第一个输入整数数组的每个数字进行遍历。使用 `xor esi, esi` 清零 `esi` 寄存器，用于内层循环计数。通过标签 `L2` 进入内层循环，表示对第二个输入整数数组的每个数字进行遍历。使用 `mov eax, dword ptr input_num1[edi*4]` 获取第一个输入整数数组中当前数字。使用 `mul input_num2[esi*4]` 进行乘法运算，结果存放在 `eax:edx` 中。使用 `mov ebx, esi` 和 `add ebx, edi` 计算当前位置在输出整数数组中的索引。使用 `add eax, output_num[ebx*4]` 将当前位置的值加上乘法的结果。使用 `mov ebx, 10` 和 `div ebx` 进行除法运算，得到进位和当前位置的值。使用 `mov ebx, esi` 添加到 `edi` 和 `esi` 上，判断是否还需要继续内层循环。最后，通过 `mov eax, len_input1` 和 `add eax, len_input2` 计算结果数组的长度。通过 `cmp output_num[eax*4-4], 0` 判断是否需要进位。若需要进位，则通过 `sub eax, 1` 减去一个长度。最终通过 `jmp END_FUNC` 跳转到结束标签。结束标签 `END_FUNC` 通过 `mov len_output, eax` 存储结果数组的长度，然后通过 `ret` 返回。若不需要进位，则直接通过 `END_FUNC` 跳转到结束标签。

```
big_num_mul proc far C

  mov ecx, len_input1
  xor edi, edi

  L1:
    xor esi, esi
    L2:
      mov eax, dword ptr input_num1[edi*4]
      mul input_num2[esi*4]

      mov ebx, esi
      add ebx, edi
      add eax, output_num[ebx*4]

      mov ebx, 10
      div ebx
```

```
      mov ebx, esi
      add ebx, edi

      mov output_num[ebx*4], edx
      add output_num[ebx*4+4], eax

      inc esi
      mov ebx, len_input2
      cmp esi, ebx
      jb L2

    inc edi
    loop L1

  mov eax, len_input1
  add eax, len_input2

  cmp output_num[eax*4-4], 0
  jz DLZ
END_FUNC:
  mov len_output, eax
  ret
DLZ:
  sub eax, 1
  jmp END_FUNC
big_num_mul endp
```

主函数

`main` 调用上述三个函数完成大整数相乘的任务。首先，通过 `invoke scanf` 接收用户输入的两个大数，并通过 `invoke strlen` 获取输入字符串的长度。接着，通过 `invoke str_to_num` 将输入字符串转换为整数数组，分别存储在 `input_num1` 和 `input_num2` 中。然后，再次使用 `invoke scanf` 接收用户输入的两个大数，并通过 `invoke strlen` 获取输入字符串的长度。再次通过 `invoke str_to_num` 将输入字符串转换为整数数组，分别存储在 `input_num1` 和 `input_num2` 中。接下来，通过 `invoke big_num_mul` 调用大整数相乘的函数，得到结果存储在 `output_num` 中。最后，通过 `invoke num_to_str` 将结果数组转换为字符串，并通过 `invoke printf` 输出结果。整个过程的参数包括输入字符串的地址 `input_str1` 和 `input_str2`、输入整数数组的地址 `input_num1` 和 `input_num2`、输出整数数组的地址 `output_num`，以及字符串的长度 `len_input1` 和 `len_input2`。

```
main proc

  invoke scanf, offset input_s, offset input_str1
  invoke strlen, offset input_str1
  mov len_input1, eax
  invoke str_to_num, offset input_str1, offset input_num1, eax

  invoke scanf, offset input_s, offset input_str2
  invoke strlen, offset input_str2
  mov len_input2, eax
  invoke str_to_num, offset input_str2, offset input_num2, eax

  invoke big_num_mul
  mov eax, len_output
  invoke num_to_str, offset output_str, offset output_num, eax
```

```
    invoke printf, offset input_s, offset output_str

    ret
main endp
```

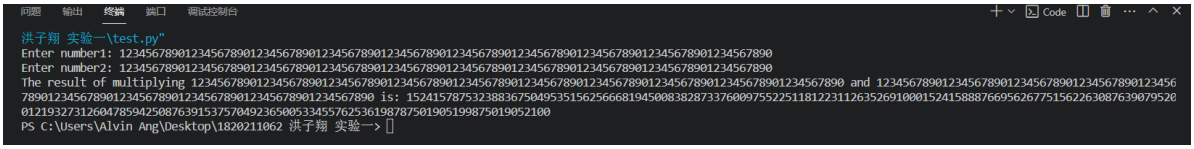
实验结果

以下为测试用例并已经得到求证结果属于正确。

python代码求证：

```
def main():
    input_str1 = input("Enter number1: ")
    input_str2 = input("Enter number2: ")

    num1 = int(input_str1)
    num2 = int(input_str2)

    result_num = num1 * num2

    print(f"The result of multiplying {input_str1} and {input_str2} is:
{result_num}")

if __name__ == "__main__":
    main()
```

结果：



总结

这次的汇编上机作业让我学到了很多新的代码知识， 从大数相乘开始对masm32的汇编语言有了初步发了解以及对指令的使用有了一定的认识，深有感悟。