Lab Assignment: Sum the File

CMPE 310

UMBC

Murali Indukuri

April 3, 2025

muralii1@umbc.edu

0.1 Approach

The objective of this project was to read in integers from a file and compute their sum. Many of the functionalities from the first project were reused, like converting binary numbers to the ASCII format to print them and convenience functions for printing text from buffers. The major new code was just to convert ASCII text on each line to binary numbers that arithmetic can be performed on. To do this, a new function called read_ints was created. This function works by getting the starting memory address of a line in the buffer storing the file text. It reads each character from the starting address until it finds a newline character. For each character, it will compare with a newline character, and if the current character is not a newline, it will subtract 48(to convert to an integer), and add it to a variable storing the integer that is being read so far. Before this addition however, the integer variable is multiplied by ten to respect the place value of the preceding character/digit. Once a newline is found, the line-read loop terminates and returns the integer.

To go along with this, another function called fill_ints was declared. This function stores the maximum number of integers in the file(read using read_ints), the offset from the start address of the buffer storing the file text, the current line number, and the sum of the integers in the file. The function runs a loop until the line number matches the maximum number of integers in the file. In each iteration, it calls the read_ints function be passing in a pointer to the buffer storing the file text(with the offset added to get to the current line). The read_ints function then returns the integer on the current line and how many characters were read. The fill_ints function adds the integer to the sum and the number of characters read to the offset variable. This is repeated until the loop ends and the sum is then returned.

As mentioned, all other functionality was reused from the last project. The fill_ints function is called to get the sum, and the int_print function from the last project is used to print the sum. There is one limitation to the code. Since the code to convert the binary integer to ascii uses one register to store the ascii text, it can only store four characters. So the maximum sum that the program can output is 9999.

0.2 Code

```
1    section .data
2    input_msg : db "Filename: ", 0  ; path to read
3    input_msg_len: equ $-input_msg
4    output_msg : db "The sum is: ", 0 ; output msg
5    output_msg_len : equ $-output_msg
6
7  section .bss
8    dynamic_path: resb 1024
9    buffer : resb 1024
10
11 section .text
12   global _start
13
14 _start:
15   push input_msg
16   push input_msg_len
17   call print
18
19   mov eax, 3
20   mov ebx, 0
21   mov ecx, dynamic_path
22   mov edx, 1024
23   int 0x80
24
25   xor edi, edi
26 null_terminate:
27   mov al, byte [dynamic_path + edi]
28   cmp al, 10
29   je terminate
```

```
30      add edi, 1
31      jmp null_terminate
32    terminate:
33      mov [dynamic_path+edi], byte 0
34
35      ; puts file descriptor in eax
36      mov eax, 5
37      mov ebx, dynamic_path
38      xor ecx, ecx
39      int 0x80
40
41      mov ebx, eax
42      mov eax, 3
43      mov ecx, buffer
44      mov edx, 1024
45      int 0x80
46
47      ; print output message
48      push output_msg
49      push output_msg_len
50      call print
51      ; calculate sum and print
52      push buffer
53      call fill_ints
54      push eax
55      call int_print
56      ; print \n
57      push 10
58      push esp
59      push 1
60      call print
61      sub esp, 4
62
63      mov eax, 1
64      int 0x80
65
66    fill_ints:
67      ; Variable documentation
68      ; ebp:
69      ;   +8, buffer pointer
70      ; esp:
71      ;   +0 max num ints
72      ;   +4 line counter
73      ;   +8 buffer counter
74      ;   +12 sum
75      push ebp
76      mov ebp, esp
77      xor eax, eax
78      xor ebx, ebx
79      xor ecx, ecx
80      xor edx, edx
81
82      sub esp, 16
83
84      mov [esp + 12], dword 0
85      mov eax, dword [ebp + 8]
86      mov [esp + 8], dword eax
87      mov [esp + 4], dword 0
88      mov [esp + 0], dword 0
```

```
89
90      push dword [esp + 8]
91      call read_ints
92      ; set max ints
93      mov [esp + 0], eax
94      ; set cursor to first int line
95      add ebx, 1
96      mov ecx, [esp + 8]
97      add ebx, ecx
98      mov [esp + 8], ebx
99
100   sum_loop:
101     push dword [esp + 8]
102     call read_ints
103     ; update sum
104     mov ecx, [esp + 12]
105     add eax, ecx
106     mov [esp + 12], eax
107    ; update buffer counter
108     mov ecx, [esp + 8]
109     add ebx, ecx
110     add ebx, 1
111     mov [esp + 8], ebx
112     ; update line counter
113     mov ecx, [esp + 4]
114     add ecx, 1
115     mov [esp + 4], ecx
116
117     ; jump logic
118     mov ebx, [esp + 4]
119     mov eax, [esp + 0]
120     cmp ebx, eax
121     jl sum_loop
122   return_sum:
123     mov eax, [esp + 12]
124     add esp, 16
125     pop ebp
126     ret 4
127
128   read_ints:
129     push ebp
130     mov ebp, esp
131     xor eax, eax
132     xor ebx, ebx
133     xor ecx, ecx
134     xor edx, edx
135
136     mov eax, [ebp+8]  ; Pointer to the line to read
137     xor edi, edi
138     push dword 0  ; counter variable
139   read_loop:
140     mov bl, byte [eax + edi]
141     cmp bl, 10  ; compare with newline
142     je return_result
143     push eax
144     xor eax, eax
145     mov al, byte [esp+4]
146     mov ecx, 10
147     mul ecx
```

```
148     sub bl, 48
149     add eax, ebx
150     mov [esp+4], eax
151     pop eax
152 
153     add edi, 1
154     jmp read_loop
155 
156   return_result:
157     pop eax
158     mov ebx, edi
159     pop ebp
160     ret 4
161 
162   print:
163     ; create call frame
164     push ebp ; Remember that this is 32bit(4 bytes)
165     mov ebp, esp
166     ; Function body
167     mov eax, 4
168     mov ebx, 1
169     mov ecx, [ebp+12]
170     mov edx, [ebp+8]
171     int 0x80
172     ; dump stack frame
173     pop ebp
174     ret 8
175 
176   int_print:
177     push ebp
178     mov ebp, esp
179 
180     mov eax, [ebp+8] ; get int
181     cmp eax, 0
182     jne checked_zero
183       test:
184       mov eax, 0x30303030 ; 0000
185       push eax
186       push esp
187       push 4
188       call print
189       pop eax
190       pop ebp
191       ret 4
192     checked_zero:
193 
194     xor edi, edi
195     xor ecx, ecx
196     loop_divide:
197       ; Divide
198       xor edx, edx
199       mov ebx, 10
200       div ebx
201       add edx, 48
202       shrd ecx, edx, 8
203       add edi, 1
204       cmp edi, 4
205       jl loop_divide
206       ; Output
```

```
207       ; flip register
208       mov ebx, ecx
209       xor ecx, ecx
210       or ch, bl
211       or cl, bh
212       ror ecx, 16
213       ror ebx, 16
214       or ch, bl
215       or cl, bh
216       ; Finally print
217       push ecx
218       push esp
219       push 4
220       call print
221       pop eax
222     pop ebp
223     ret 4
```

0.3 Output

This section shows the output of the code based on the file provided and a custom text file.

```
murali@murali-Inspiron-16-Plus-7630:~/Documents/Code/ASM/sum_the_file$ ./build-asm.sh sum_the_file
Filename: randomInt100.txt
The sum is: 4579
```

Figure 1: Shows the program working with the provided text file of 100 random integers

```
murali@murali-Inspiron-16-Plus-7630:~/Documents/Code/ASM/sum_the_file$ cat test10Ints.txt
10
825
387
337
44
67
98
339
98
50
66
murali@murali-Inspiron-16-Plus-7630:~/Documents/Code/ASM/sum_the_file$ ./build-asm.sh sum_the_file
Filename: test10Ints.txt
The sum is: 2311
murali@murali-Inspiron-16-Plus-7630:~/Documents/Code/ASM/sum_the_file$ qalc 825+387+337+44+67+98+339+98+50+66
825 + 387 + 337 + 44 + 67 + 98 + 339 + 98 + 50 + 66 = 2311
```

Figure 2: Shows the program working with a custom file of 0 integers. The integers in the file are shown as well as the actual sum calculated by the command line calculator program "qalc"