CS210 : mini Project Report

Instructor – Prof. Jimson Mathew

Submitted by – Harshvardhan Singh(2201CS92)

Date – 10/03/24

Simple Instructions Assembler and Emulator

Introduction- This is a project comprised of making a simple assembler of instruction set given in question statement and an emulator with some of the supported instructions.

Assumptions-

1. Assembler gives all error if found on first pass(duplicate label/bogus labelname). If no error found then proceed to second pass. If an error is found on second pass ;it is a potential one. So the Assembling process is halted and the only the error on that line is printed on logfile(not all error exceeding that line). Object code and memdump(listing file) is also written upto that line.
2. If program use data segment, then HALT is necessary to avoid the pc getting into memory section.

All other standards are according to the project question description.

Methodology –

For assembler –

It is a two pass assembler. First we input a asm file in asm.cpp (scripts for running is given in makefile.txt) . It filters out all the comments, inline comments, blank spaces(before and after instructions),blank lines . Then it stores the filtered instructions in a separate array/table. Errors also has a array.

After it we use first and second pass on that array/table

FIRST PASS- It scans the labels and pushes into a label array along with pc(or <value> if SET <value> is there). Identify common error like duplicate label or bogus label and pushes into errors array. If it is a instructions then push into BufferInstructions array.

SECOND PASS – Iterates BufferInstructions array, scans for validity of opcode/mnemonic using a lookup table(instructions table defined in global area) . If opcode is correct then proceed for address/operand. Using suitable logic for error finding in both(opcode and address/operand) it writes the error on errors array and halts the assembling program. Now if the instruction is correct ; it writes into listing file and object file.

After that it writes into logfile.(errors if any)

FOR Emulator-

For emulator it was easy . We take four registers as int variables in global area with Memory of size almost 8000 words and an instruction array.

We defined function for each operation add,ldc etc. First we take .obj file then read and write it instruction array and memory. Then Run the emulator operation with suitable loop conditions.

Then we print the memdump file and close it.

Submission –

- Assembler source file is name asm.cpp.
- Claims and self attestation is given in claims.txt

- Running scripts for assembler is given in Makefile.txt
- Sum of array program in test04.asm
- Max of array program in test03.asm
- Bubblesort program is stored in test06.asm

Since my assembler halts assembling when a potential error occurs(ie in 2nd pass) ,assembly code and output screenshot for each possible error is stored in pdf .

Issues Faced

- Most problematic was convert the instruction and opcode as whole to 4 byte instruction and push in object file. Many a times while conversion from string to int , negative number were giving overflow. So I have to make suitable logic for postive and negative numbers separately.

Evidences for a working assembler-

One can verify the output of test program by running the assembler with the scripts provided.

Along with that I am also attaching some screenshots of outputs-

My tests-

Test01.asm

mytests > test01 > test01.asm

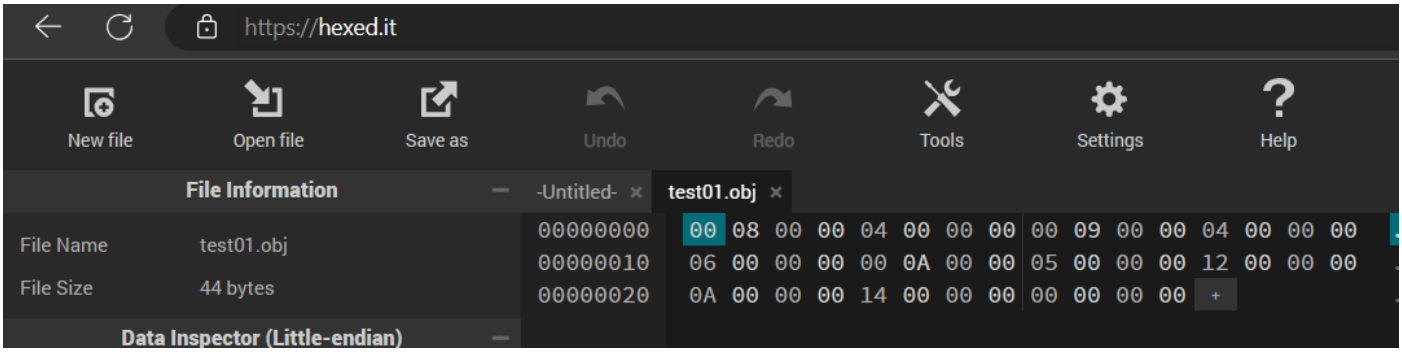
```
1   ; asm program to sum two integers
2
3
4
5   ldc  a      ; acc=(address of a) ,base=0(previous acc)
6   ldnl 0      ; acc= 10
7   ldc  b      ; acc=(address of b) , base=10(previous acc)
8   ldnl 0      ; acc= 20
9   add         ; acc= acc+base (10 +20)
10  ldc  result ; acc=(address of result),base=30(prev accumulator)
11  stnl 0      ; mem[acc+0]= base
12  HALT        ; program to halt before memory block
13
14  a:  data 10
15  b:  data 20
16  result: data 0
```

mytests > test01 > test01.lst

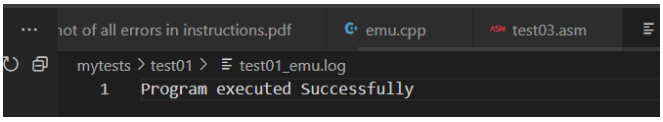
```
1   00000000  00000800  ldc  a
2   00000001  00000004  ldnl 0
3   00000002  00000900  ldc  b
4   00000003  00000004  ldnl 0
5   00000004  00000006  add
6   00000005  00000a00  ldc  result
7   00000006  00000005  stnl 0
8   00000007  00000012  HALT
9   00000008            a:
10  00000008  0000000a    data 10
11  00000009            b:
12  00000009  00000014    data 20
13  0000000a            result:
14  0000000a  00000000    data 0
15
```

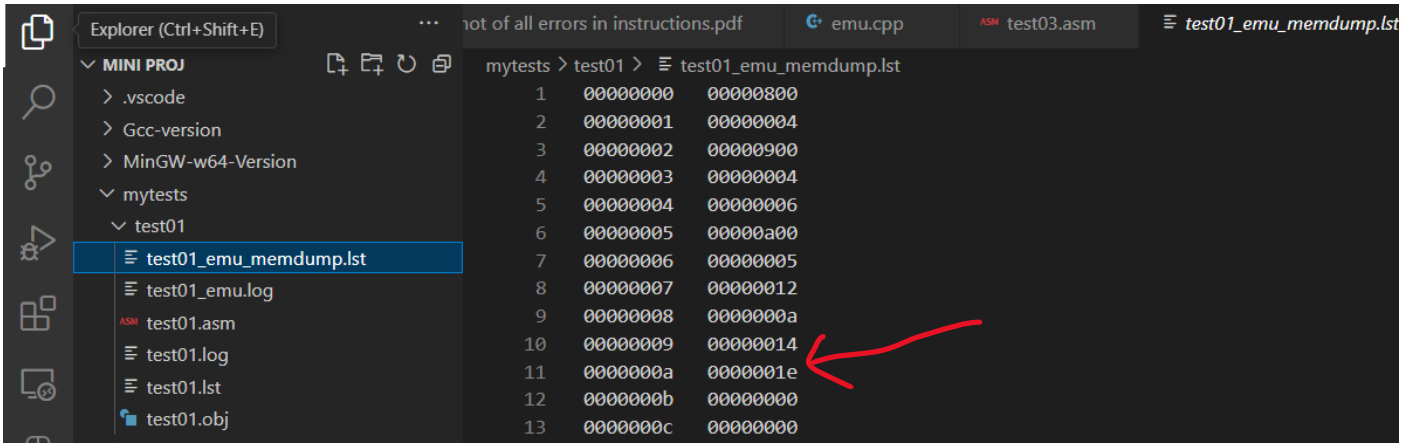
mytests > test01 > test01.log

```
1   Assembled Successfully
```

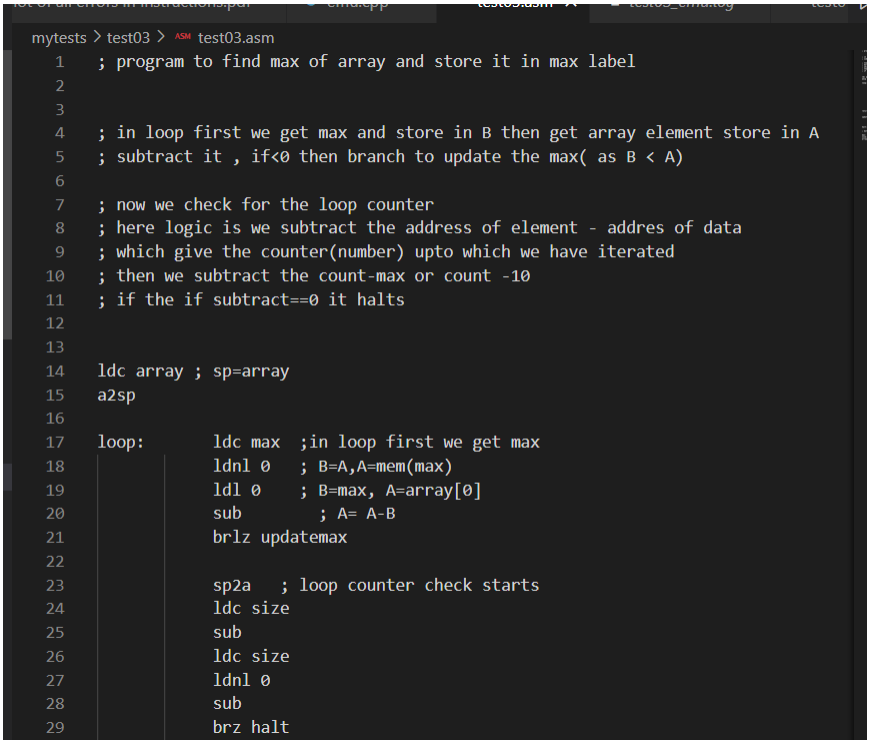


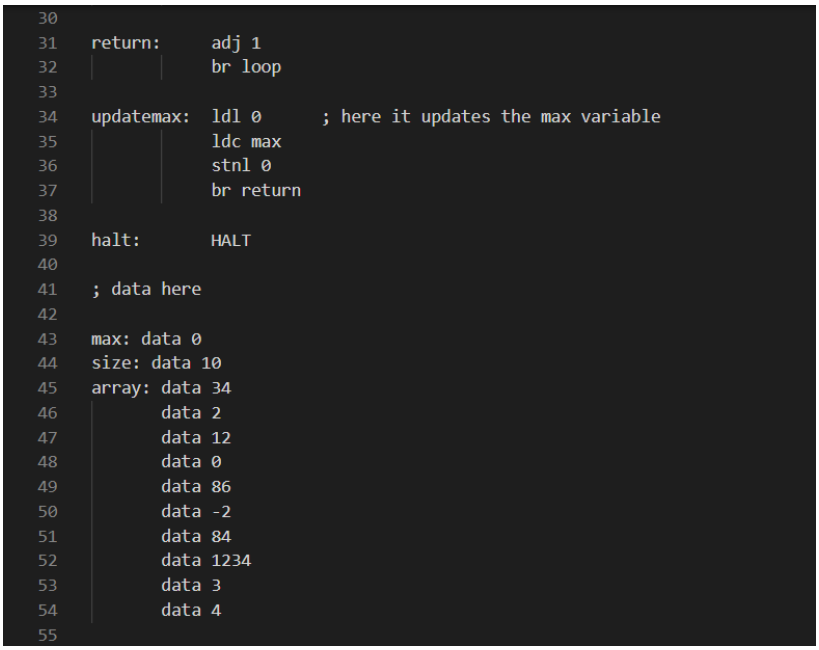
Test01 Emu outputs




Explorer (Ctrl+Shift+E)
MINI PROJ
.vscode
Gcc-version
MinGW-w64-Version
mytests
test01
test01_emu_memdump.lst
test01_emu.log
test01.asm
test01.log
test01.lst
test01.obj
not of all errors in instructions.pdf
emu.cpp
test03.asm
test01_emu_memdump.lst
mytests > test01 > test01_emu_memdump.lst
1 00000000 00000800
2 00000001 00000004
3 00000002 00000900
4 00000003 00000004
5 00000004 00000006
6 00000005 00000a00
7 00000006 00000005
8 00000007 00000012
9 00000008 0000000a
10 00000009 00000014
11 0000000a 0000001e
12 0000000b 00000000
13 0000000c 00000000

Test03.asm


mytests > test03 > test03.asm
; program to find max of array and store it in max label
; in loop first we get max and store in B then get array element store in A
; subtract it , if<0 then branch to update the max(as B < A)
; now we check for the loop counter
; here logic is we subtract the address of element - addres of data
; which give the counter(number) upto which we have iterated
; then we subtract the count-max or count -10
; if the if subtract==0 it halts
ldc array ; sp=array
a2sp
loop: ldc max ;in loop first we get max
ldnl 0 ; B=A,A=mem(max)
ldl 0 ; B=max, A=array[0]
sub ; A= A-B
brlz updatemax
sp2a ; loop counter check starts
ldc size
sub
ldc size
ldnl 0
sub
brz halt


return: adj 1
br loop
updatemax: ldl 0 ; here it updates the max variable
ldc max
stnl 0
br return
halt: HALT
; data here
max: data 0
size: data 10
array: data 34
data 2
data 12
data 0
data 86
data -2
data 84
data 1234
data 3
data 4

mytests > test03 > test03.log

```
1   Assembled Successfully
```

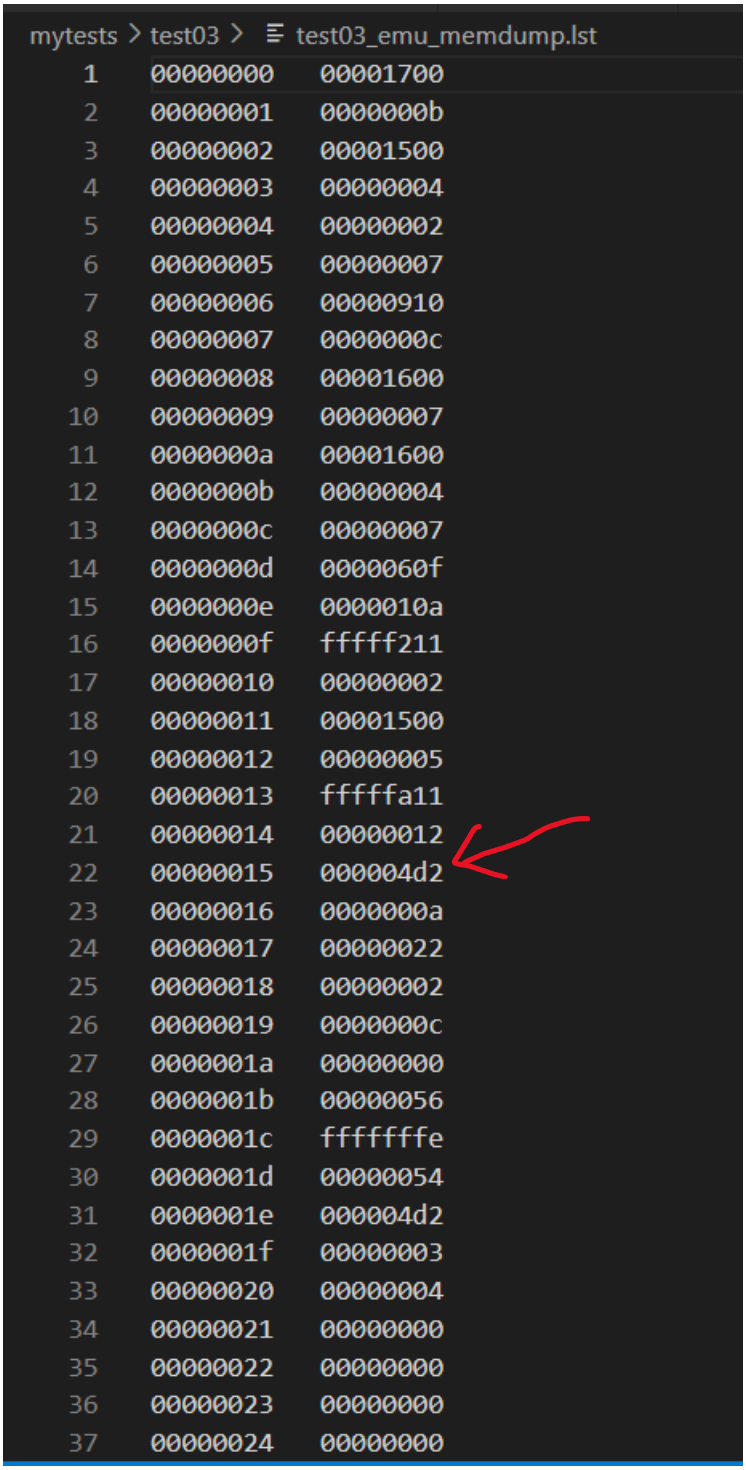
mytests > test03 > test03.lst

```
1   00000000  00001700  ldc array
2   00000001  0000000b  a2sp
3   00000002            loop:
4   00000002  00001500  ldc max
5   00000003  00000004  ldnl 0
6   00000004  00000002  ldl 0
7   00000005  00000007  sub
8   00000006  00000910  brlz updatemax
9   00000007  0000000c  sp2a
10  00000008  00001600  ldc size
11  00000009  00000007  sub
12  0000000a  00001600  ldc size
13  0000000b  00000004  ldnl 0
14  0000000c  00000007  sub
15  0000000d  0000060f  brz halt
16  0000000e            return:
17  0000000e  0000010a  adj 1
18  0000000f  fffff211  br loop
19  00000010            updatemax:
20  00000010  00000002  ldl 0
21  00000011  00001500  ldc max
22  00000012  00000005  stnl 0
23  00000013  fffffa11  br return
24  00000014            halt:
25  00000014  00000012  HALT
26  00000015            max:
27  00000015  00000000    data 0
28  00000016            size:
29  00000016  0000000a    data 10
30  00000017            array:
31  00000017  00000022    data 34
32  00000018  00000002    data 2
33  00000019  0000000c    data 12
34  0000001a  00000000    data 0
35  0000001b  00000056    data 86
36  0000001c  fffffffe    data -2
37  0000001d  00000054    data 84
38  0000001e  000004d2    data 1234
39  0000001f  00000003    data 3
40  00000020  00000004    data 4
41
```

Now emu outputs-

mytests > test03 > test03_emu.log

```
1   Program executed Successfully
```



More test files are given