

Microprocessor and Assembly Programming Laboratory

B.Tech. III Semester



Name : DEEPAK R

Roll Number : 18ETCS002041

Department : Computer Science and Engineering

**Faculty of Engineering & Technology
Ramaiah University of Applied Sciences**

Name: **DEEPAK**

Ramaiah University of Applied Sciences

Private University Established in Karnataka State by Act No. 15 of 2013



Faculty	Engineering & Technology
Programme	B. Tech. in Computer Science and Engineering
Year/Semester	2018/3 rd Semester
Name of the Laboratory	Microprocessor and Assembly Programming Laboratory
Laboratory Code	

List of Experiments

1. Data transfer operations	
2. Arithmetic operations	
3. Logical operations	
4. Controlling execution flow using conditional instructions	
5. String manipulation	
6. Searching an element in an array	
7. Sorting an array	
8. Interfacing	
9. Interfacing	

No.	Lab Experiment	Viva (6)	Results (7)	Documentation (7)	Total Marks (20)
1	Data transfer operations				
2	Arithmetic operations				
3	Logical operations				
4	Controlling execution flow using conditional instructions				
5	String manipulation				
6	Searching an element in an array				
7	Sorting an array				
8	Interfacing				
9	Interfacing				
10	Lab Internal Test conducted along the lines of SEE and valued for 50 Marks and reduced for 20 Marks				
	Total Marks				

Laboratory 1

Title of the Laboratory Exercise: **Data transfer operations**

1. Introduction and Purpose of Experiment

Students will be able to define data of different data types and perform data transfer operations on the data

2. Aim and Objectives

Aim

To develop assembly language program to perform data transfer operations on different data.

Objectives

At the end of this lab, the student will be able to

- Define data of different data types
- Perform data transfer operations
- Create a simple assembly language program
- Use GAS assembler
- Understand GNU debugger

3. Experimental Procedure

1. Write algorithm to solve the given problem
2. Translate the algorithm to assembly language code
3. Run the assembly code in GNU assembler
4. Create a laboratory report documenting the work

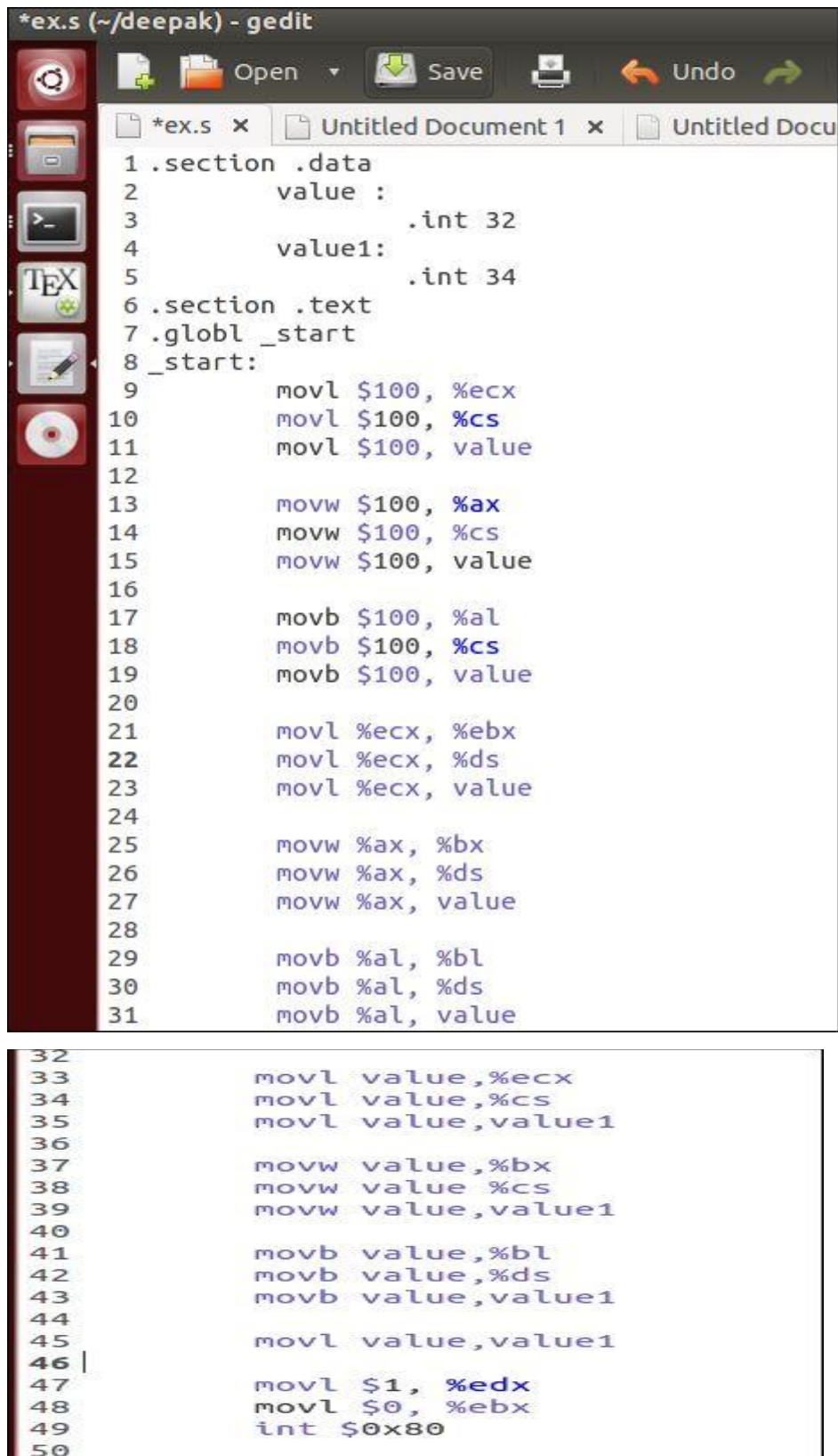
4. Questions

1. Perform the following data transfer operations

1. 32 bit integer data to a	General Purpose register Segment Register Memory
2. 16 bit integer data to a	General Purpose register Segment Register Memory

3. 8 bit integer data to a	General Purpose register Segment Register Memory
4. 32 bit integer data from a General purpose register to a <i>(Repeat the same for 16 bit integer data and 8 bit integer data)</i>	General Purpose register Segment Register Memory
5. 32 bit integer data from memory to a <i>(Repeat the same for 16 bit integer data and 8 bit integer data)</i>	General Purpose register Segment Register Memory
6. 32 bit integer data from memory to	Memory region

5. Calculations/Computations/Algorithms



```

*ex.s (~/deepak) - gedit
1 .section .data
2     value :
3         .int 32
4     value1:
5         .int 34
6 .section .text
7 .globl _start
8 _start:
9     movl $100, %ecx
10    movl $100, %cs
11    movl $100, value
12
13    movw $100, %ax
14    movw $100, %cs
15    movw $100, value
16
17    movb $100, %al
18    movb $100, %cs
19    movb $100, value
20
21    movl %ecx, %ebx
22    movl %ecx, %ds
23    movl %ecx, value
24
25    movw %ax, %bx
26    movw %ax, %ds
27    movw %ax, value
28
29    movb %al, %bl
30    movb %al, %ds
31    movb %al, value

32
33    movl value,%ecx
34    movl value,%cs
35    movl value,value1
36
37    movw value,%bx
38    movw value %cs
39    movw value,value1
40
41    movb value,%bl
42    movb value,%ds
43    movb value,value1
44
45    movl value,value1
46
47    movl $1, %edx
48    movl $0, %ebx
49    int $0x80
50

```

FIG 1 SHOWING CODES FOR DATA TRANSFER

```
exam@msruas-cse-vbox-ubt:~/deepak$ as -gstabs ex.s -o ex.o
exam@msruas-cse-vbox-ubt:~/deepak$ ld ex.o -o ex
exam@msruas-cse-vbox-ubt:~/deepak$ gdb ex
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ex...done.
```

FIG2 SHOWING WRITTEN INPUT IN TERMINAL

```
(gdb) break 9
Breakpoint 1 at 0x8048074: file ex.s, line 9.
(gdb) break 10
Breakpoint 2 at 0x8048079: file ex.s, line 10.
(gdb) break 11
Breakpoint 3 at 0x804807b: file ex.s, line 11.
(gdb) run
Starting program: /home/exam/deepak/ex

Breakpoint 1, _start () at ex.s:9
9          movl $100,%ecx
```

FIG 3 SHOWING GIVEN COMMANDS TO BREAKPOINT EACH LINE

6. PRESENTATION OF RESULTS

```
(gdb) info registers
eax            0x0            0
ecx            0x0            0
edx            0x0            0
ebx            0x0            0
esp            0xbffff050      0xbffff050
ebp            0x0            0x0
esi            0x0            0
edi            0x0            0
eip            0x8048074        0x8048074 <_start>
eflags         0x202          [ IF ]
cs             0x73            115
ss             0x7b            123
ds             0x7b            123
es             0x7b            123
fs             0x0            0
gs             0x0            0
(gdb) c
Continuing.
```

FIG 4 SHOWING INFO OF REGISTERS BEFORE DATA TRANSFER

```
Breakpoint 2, _start () at ex.s:10
10      movl %eax,%ebx
(gdb) info registers
eax            0x0            0
ecx            0x64            100
edx            0x0            0
ebx            0x0            0
esp            0xbffff050      0xbffff050
ebp            0x0            0x0
esi            0x0            0
edi            0x0            0
eip            0x8048079        0x8048079 <_start+5>
eflags         0x202          [ IF ]
cs             0x73            115
ss             0x7b            123
ds             0x7b            123
es             0x7b            123
fs             0x0            0
gs             0x0            0
```

FIG 5 SHOWING INFO OF REGISTERS AFTER DATA TRANSFER

7. Analysis and Discussions

Assembly language is a low level language and dependent on the system architecture but in case of C, it can be ported to various platforms

8. Conclusions

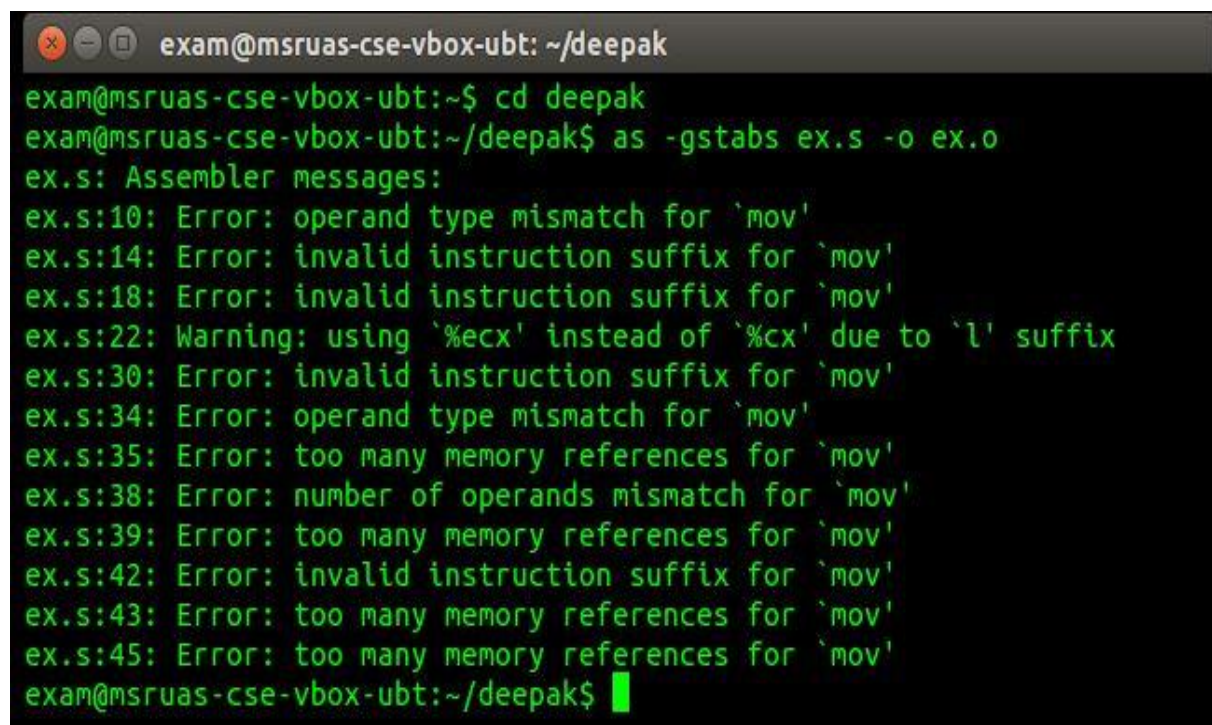
The assembly code for one architecture can not port to another architecture with out radical rewrite of the code

9. Comments

1. Limitations of Experiments

The certain programs cannot be performed using above method certain example are shown in the below picture .

2. Limitations of Results

A screenshot of a terminal window with a dark background and green text. The window title is 'exam@msruas-cse-vbox-ubt: ~/deepak'. The user has entered the command 'cd deepak' and then 'as -gstabs ex.s -o ex.o'. The assembler has produced several error and warning messages. The errors include operand type mismatches, invalid instruction suffixes for 'mov', too many memory references, and number of operands mismatches. One warning indicates the use of '%ecx' instead of '%cx' due to a 'l' suffix. The terminal ends with a prompt 'exam@msruas-cse-vbox-ubt:~/deepak\$' and a green cursor.

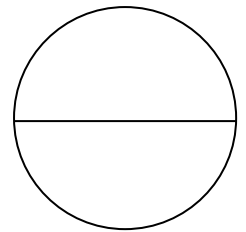
```
exam@msruas-cse-vbox-ubt:~/deepak$ cd deepak
exam@msruas-cse-vbox-ubt:~/deepak$ as -gstabs ex.s -o ex.o
ex.s: Assembler messages:
ex.s:10: Error: operand type mismatch for `mov'
ex.s:14: Error: invalid instruction suffix for `mov'
ex.s:18: Error: invalid instruction suffix for `mov'
ex.s:22: Warning: using `%ecx' instead of `%cx' due to `l' suffix
ex.s:30: Error: invalid instruction suffix for `mov'
ex.s:34: Error: operand type mismatch for `mov'
ex.s:35: Error: too many memory references for `mov'
ex.s:38: Error: number of operands mismatch for `mov'
ex.s:39: Error: too many memory references for `mov'
ex.s:42: Error: invalid instruction suffix for `mov'
ex.s:43: Error: too many memory references for `mov'
ex.s:45: Error: too many memory references for `mov'
exam@msruas-cse-vbox-ubt:~/deepak$
```

3. Learning happened

To develop assembly language program to perform data transfer operations on different data. Performing data transfer operation using Immediate, Register and Direct addressing modes

4. Recommendations

Assembly language is a human readable translation of the machine code. There is a direct one to one translation from assembly instruction to processor instruction. Assembly language is used when you need to guarantee certain things happen in a certain order, or a certain number of cycles. Assembly programming is used a lot when writing operating systems or drivers where there is close interaction with hardware needed. Often it is the only way to access specialised instruction that a processor may provide. So it can be used in many language compilers and interpreters.

Signature and date**Marks**

Name: **DEEPAK R**

Registration Number: **18ETCS002041**