

Microprocessor and Assembly Programming Laboratory

B.Tech. III Semester



Name : SUBHENDU MAJI

Roll Number : 18ETCS002121

Department : Computer Science and Engineering

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

Name: SUBHENI

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

- ✓ Data transfer instructions move data from one place in the computer to another without changing the data.
- ✓ Typical transfers are between memory and processor registers, between processor registers and input and output registers, and among the processor registers themselves.
- ✓ Where there are two operands, the rightmost one is the destination. The leftmost one is the source.
- ✓ For example, `movl %edx, %eax` means moves the contents of the edx register into the eax register.

ALGORITHM

STEP 1: Start

STEP 2: declare data section

STEP 3: declare variable name, type and value

STEP 4: declare data section, mark it as `_start` section

STEP 5: move 32bit value 1 into register `eax`

STEP 6: move 32bit value of register `eax` to register `ebx`

STEP 7: move 32bit value of register `eax` to memory (variable `a`)

STEP 8: move 32bit value stored in memory to the register `ecx`

STEP 9: move 32bit value 2 to memory

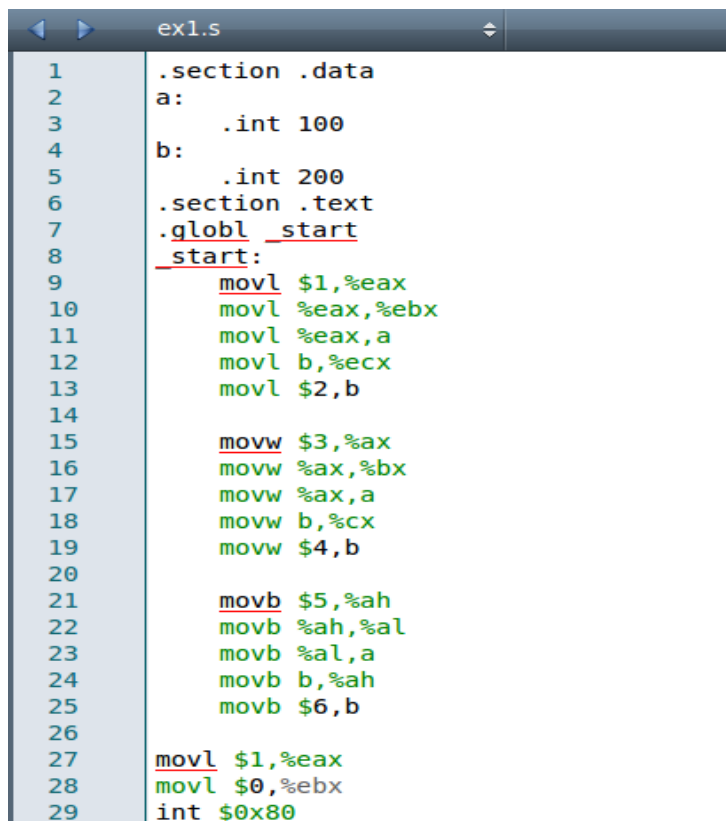
STEP 10: repeat same steps for 16bit data and 8bit data

STEP 11: Stop

6. Presentation of Results

```
mplab@msruas-cse-vbox-ubt:~$ mkdir smaji
mplab@msruas-cse-vbox-ubt:~$ cd smaji
mplab@msruas-cse-vbox-ubt:~/smaji$ gedit ex1.s
```

Figure 1 Make file



```
1  .section .data
2  a:
3      .int 100
4  b:
5      .int 200
6  .section .text
7  .globl _start
8  _start:
9      movl $1,%eax
10     movl %eax,%ebx
11     movl %eax,a
12     movl b,%ecx
13     movl $2,b
14
15     movw $3,%ax
16     movw %ax,%bx
17     movw %ax,a
18     movw b,%cx
19     movw $4,b
20
21     movb $5,%ah
22     movb %ah,%al
23     movb %al,a
24     movb b,%ah
25     movb $6,b
26
27     movl $1,%eax
28     movl $0,%ebx
29     int $0x80
```

Figure 2 ASM code

```

mplab@msruas-cse-vbox-ubt:~/smaji$ as -gstabs ex1.s -o ex1.o
mplab@msruas-cse-vbox-ubt:~/smaji$ ld ex1.o -o ex1
mplab@msruas-cse-vbox-ubt:~/smaji$ gdb ex1
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
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copyi
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 ex1...done.
(gdb) break 9
Breakpoint 1 at 0x8048074: file ex1.s, line 9.
(gdb)

```

Figure 3 compile and link

```

(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/mplab/smaj/ex1

Breakpoint 1, _start () at ex1.s:9
9      movl $1,%eax
(gdb) info register
eax                0x0            0
ecx                0x0            0
edx                0x0            0
ebx                0x0            0
esp                0xbffff5c0      0xbffff5c0
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)

```

Figure 4 running and register status

```

(gdb) c
Continuing.

Breakpoint 3, _start () at ex1.s:11
11     movl %eax,a
(gdb) print a
$1 = 100
(gdb)

```

Figure 5 value status

```

mplab@msruas-cse-vbox-ubt:~/smaji$ as -gstabs ex1.s -o ex1.o
ex1.s: Assembler messages:
ex1.s:15: Error: too many memory references for `mov'
ex1.s:16: Error: operand type mismatch for `mov'
mplab@msruas-cse-vbox-ubt:~/smaji$

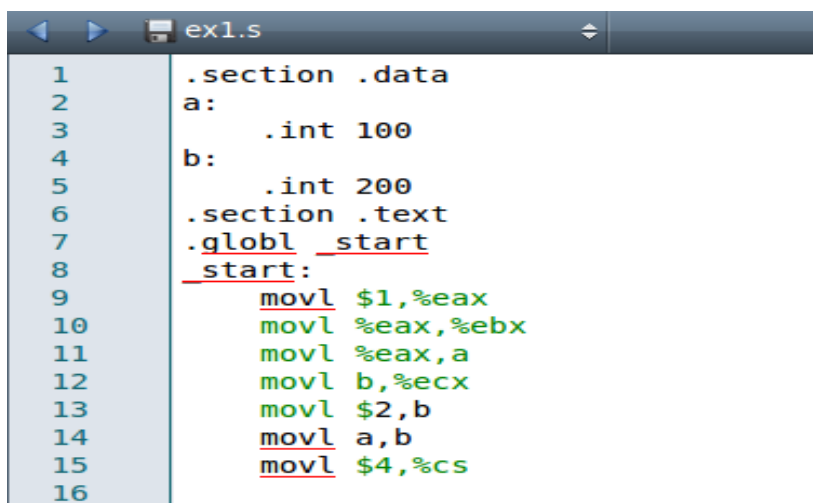
```

Figure 6 Error Screenshot for trying to move data from memory to memory

7. Analysis and Discussions

The `mov` instruction copies the data item referred to by its first operand (i.e. register contents, memory contents, or a constant value) into the location referred to by its second operand (i.e. a register or memory). While register-to-register moves are possible, direct memory-to-memory moves are not. The assembler gives error if we try to move data direct from memory-to-memory (Fig. 6). In cases where memory transfers are desired, the source memory contents must first be loaded into a register, then can be stored to the destination memory address.

We cannot move data into segment registers. As we can see in the following code, assembler gives error while linking.



```

1  .section .data
2  a:
3      .int 100
4  b:
5      .int 200
6  .section .text
7  .globl start
8  start:
9      movl $1,%eax
10     movl %eax,%ebx
11     movl %eax,a
12     movl b,%ecx
13     movl $2,b
14     movl a,b
15     movl $4,%cs
16

```

Figure 7 Code snippet

```

lab1.s: Assembler messages:
lab1.s:8: Error: operand type mismatch for `mov'
lab1.s:12: Error: invalid instruction suffix for `mov'
lab1.s:16: Error: invalid instruction suffix for `mov'

```

Figure 8 Error screenshot

8. Conclusions

Data can be moved from and between registers and memory using `MOV`, which can be used with `b`, `w`, `d` i.e. 8bit, 16bit, and 32bit data movement.

We successfully executed the doable parts of the question of moving data from registers, memory and segments of the registers.

Signature and date

Marks

