# 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

1. 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

1. 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

1. 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  *(Repeat the same for 16 bit integer data and 8 bit integer data)* | General Purpose register  Segment Register  Memory |
| 5. 32 bit integer data from memory to a  *(Repeat the same for 16 bit integer data and 8 bit integer data)* | General Purpose register  Segment Register  Memory |
| 6. 32 bit integer data from memory to | Memory region |

1. 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.
* We will look over the above example for 32-bit, 16-bit and 8-bits along with two different kinds of register, one the General-Purpose Register and the other Segment Register.

General Purpose Registers :

AX is the primary accumulator; it is used in input/output and most arithmetic instructions. For example, in multiplication operation, one operand is stored in EAX or AX or AL register according to the size of the operand.

BX is known as the base register, as it could be used in indexed addressing.

CX is known as the count register, as the ECX, CX registers store the loop count in iterative operations.

DX is known as the data register. It is also used in input/output operations. It is also used with AX register along with DX for multiply and divide operations involving large values.

Segment Registers:

Segments are specific areas defined in a program for containing data, code and stack. There are three main segments −

Code Segment − It contains all the instructions to be executed. A 16-bit Code Segment register or CS register stores the starting address of the code segment.

Data Segment − It contains data, constants and work areas. A 16-bit Data Segment register or DS register stores the starting address of the data segment.

Stack Segment − It contains data and return addresses of procedures or subroutines. It is implemented as a 'stack' data structure. The Stack Segment register or SS register stores the starting address of the stack.

Algorithm

STEP 1: Start

STEP 2: declare data section

STEP 3: declare bss section

STEP 3: declare variable name, type and value

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

STEP 5: move value 10 into register cl

STEP 6: move value 100 into register bx

STEP 7: move value 10000000 into register eax

STEP 8: move value in register eax to the register ecx

STEP 9: move value value to eax

STEP 10: move value eax x

STEP 11: syscall and exit

STEP 12: Stop

1. Presentation of Results



Figure 1 ASM Source Code

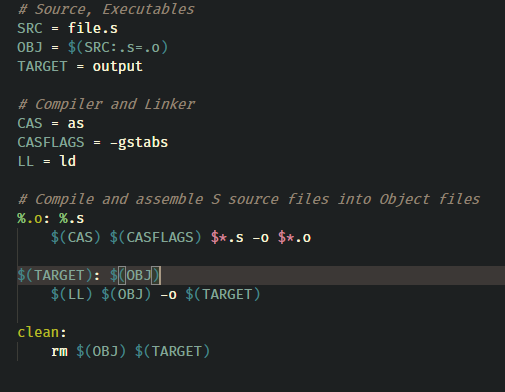


Figure 2 Makefile

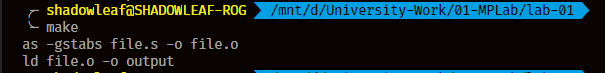


Figure 3 compile and link

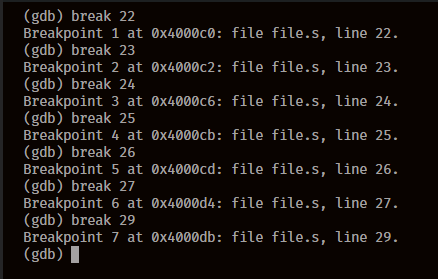


Figure 4 Set Breakpoints

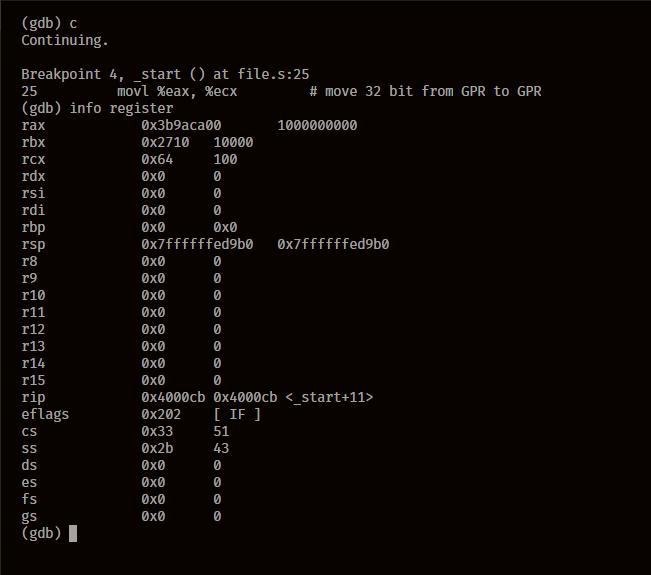


Figure 5 Register Status 1

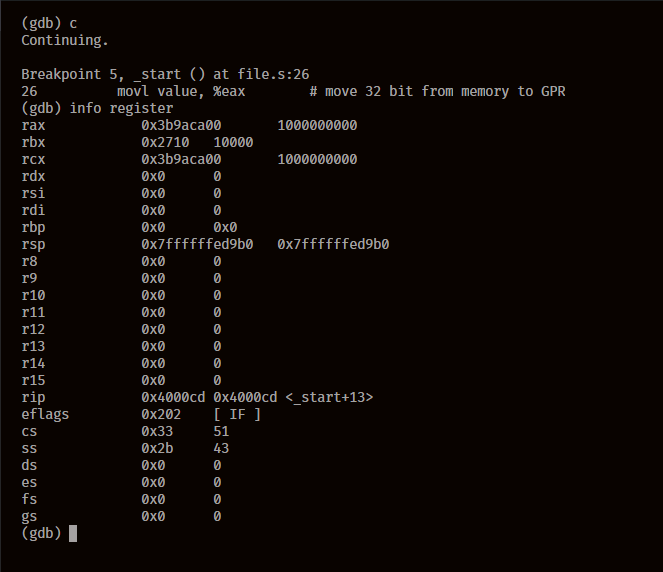


Figure 6 Register Status 2

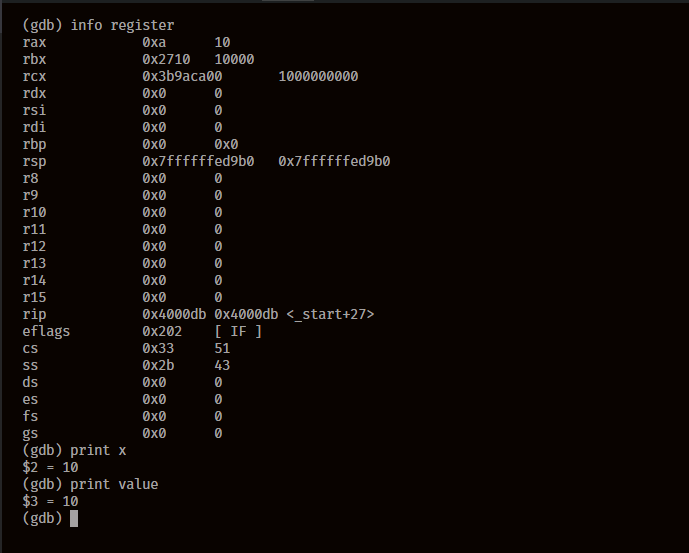


Figure 7 Value Status 1

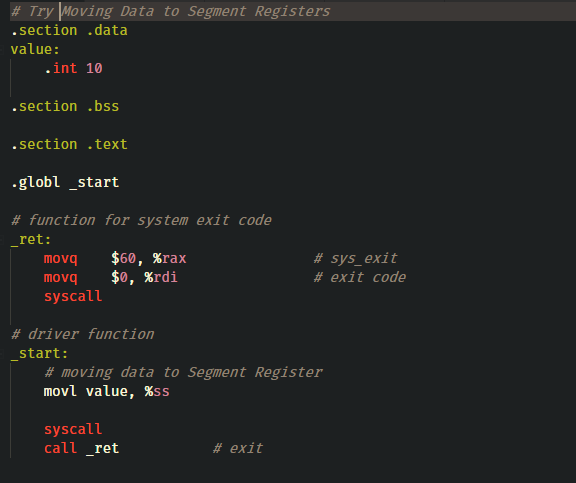


Figure 8 Source Code for Section Register

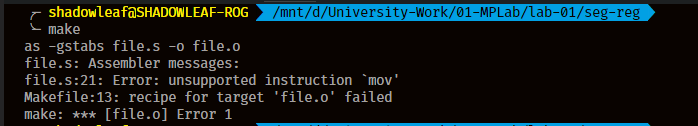


Figure 9 Compile and Link

1. 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. 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.

The mov instruction copies the data item referred to by its second operand (i.e. register contents, memory contents, or a constant value) into the location referred to by its first operand (i.e. a register or memory). While register-to-register moves are possible, direct memory-to-memory moves are not. 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.

The data can be moved from registers and in between register, although memory to memory movement of data is forbidden, As we can see in Figure 8, as complains about mov why ? The most likely reason is simply one of simplifying the design: it takes extra wiring and gates to do that, and it's an uncommon enough operation (this is the 70's) that it's not worth the real estate in the chip. This is not surprising; the 8086 already went overboard allowing any of the normal registers to be connected to the ALU (arithmetic logic unit) which allows any register to be used as an accumulator. Most processors at the time only allowed one register (the accumulator) to be used for that purpose, thus the GAS compiler gives an error of mov is not a valid operation for memory to memory data movement.

It's also possible that allowing a segment register to be written from a memory read resulted in several weird edge cases that were hard to get right in the circuitry. After all, the segment registers to be written might be used to address the source operand.

1. Conclusions

Data can be moved from and between registers and memory using mov, which can be used with l, b, w 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.

1. Comments

1. Limitations of Experiments

All the aims of the data transfers cannot be performed due to limitations of assembly and the instruction set that it follows, there are workarounds for this but that’s not the standard method of doing so.

2. Limitations of Results

The segment registers are not the same (on hardware level) as the general purpose registers. Texact reason why you can't move directly immediate value into the segment register is known only by the Intel developers. But I suppose, it is because the design is simple this way. Note that this choice does not affects the processor performance, because the segment register operations are very rare. So, one instruction more, one less is not important at all.

3. Learning happened

8E /r mov Sreg, r/m16, and allows a register or memory source (but not immediate).

mov ds, [5000h] runs just fine, assuming the 16 bytes at address 5000h hold a useful segment value for the mode you're in. (Real mode where they're used directly as numbers vs. protected where Sreg values are selectors that index the LDT / GDT).

4. Recommendations

The exit code for different machines on different architectures are different which needs to be taken care of first, otherwise this gives a segmentation fault.

Memory to memory and immediate value to segment registers aren’t allowed implicitly, although there are workarounds to make this work, even though has not practical purposes of doing so.

Signature and date Marks