Lab 3

Procedures

Learning objective of this Lab

* Procedures (‘functions’ in C)
* Load word (lw) and store word (sw) instructions
* Saving data to and later retrieving it from Stack

A procedure or function is one tool, C or Java programmers use to structure programs, both to make them easier to understand, and to allow the code to be reused.

* Pre-Lab

1. Procedures
   1. Need for Procedures

Procedures are used for two reasons

* Making the code more understandable by grouping together multiple instructions performing a particular task
* Removing repetition of certain code which is used multiple times in a program.

For example, the code given below just takes input from user and displays output= input number +2. This code is without procedures. Compare this code with the code given later on and observe how neat the code looks with the use of procedures.

# Name Reg No. Date

# **without\_**procedures.asm-- A "out=in+2" program.

# Registers used:

# $v0 - syscall parameter and return value.

# $a0 - syscall parameter-- the string to print.

.text

main:

la $a0, input\_msg # load the addr of hello\_msg into $a0.

li $v0, 4 # 4 is the print\_string calling code.

Syscall

li $v0, 1 # 1 is the get integer input code.

Syscall

addi $v0,$v0,2 # output= input+2

mov $t2,$v0 #Backup

la $a0, output\_msg # load the addr of hello\_msg into $a0.

li $v0, 4 # 4 is the print\_string calling code.

Syscall

mov $a0,$t2 #Retrieving output value back

li $v0, 5 # 5 is the code for displaying integer.

Syscall

li $v0, 10 # 10 is the code for exiting.

Syscall

# Data for the program:

.data

input\_msg: .asciiz "Enter Number:\n"

output\_msg: .asciiz "Answer is :\n"

# end withoutprocedures.asm

* 1. Instructions and example
     1. Instructions

Instruction pair for calling and returning from a procedure is

|  |  |
| --- | --- |
| **jal** Label | Copies the address of the next instruction into the register $ra (register 31) and then jumps to the address label |
| **jr** $Register | Jumps to the address in $register. Most common use jr $ra |

Table 3.1

For Example

main:

jal label #jumping to label address of next #instruction is saved in $ra

**move $t0,$a0**

li $v0,10 #Exiting

syscall

label:

li $v0,5

syscall

jr $ra #jumping back to instruction **move $t0,$a0**

Previous code, now after using procedures is given below

# Name Reg No. Date

# with\_procedures.asm-- A "out=in+2" program.

# Registers used:

# $v0 - syscall parameter and return value.

# $a0 - syscall parameter-- the string to print.

.text

main:

la $a0, input\_msg # load the addr of input\_msg into $a0.

jal displaymsg #Display String ‘input\_msg’

jal getinput #getting integer input

addi $v0,$v0,2 # output= input+2

mov $t2,$v0 #Backup

la $a0, output\_msg # load the addr of output\_msg into $a0.

jal displaymsg

mov $a0,$t2 #Retrieving output value back

jal display\_output

li $v0, 10 # 10 is the code for exiting.

Syscall # do the syscall.

Displaymsg:

li $v0, 4 # 4 is the print\_string calling code.

Syscall

jr $ra

getinput:

li $v0, 1 # 1 is the get integer input code.

Syscall

jr $ra

Display\_output

li $v0, 5 # 5 is the code to display integer.

Syscall

jr $ra

# Data for the program:

.data

input\_msg: .asciiz "Enter Number:\n"

output\_msg: .asciiz "Answer is :\n"

# end withoutprocedures.asm

Note how understandable the code becomes after using procedures.

* 1. Procedure structure for Excessive register usage and when function calls another function
     1. Register Usage Convention

Register convention given below is not compulsory but it is important. As we all write English from left to right. Few people may decide to write from right to left for communicating between themselves, but they will face problem communicating with the other people.

* $a0-$a3: four argument registers in which to pass parameters.
* $v0-$v1: two value registers in which to return values
* $ra: one return address register to return to the point of origin
* $t0-$t9: 10 temporary registers (caller saved) that are not preserved by the callee (called procedure) on a procedure call
* $s0-$s7: 8 saved registers(callee saved) that must be preserved on a procedure call (if used, the callee saves and restores them)

**Caller :** One which is calling

**Callee:** One which is being called

**Caller Saved Registers :** Backup to stack before calling another function, use in that function freely; and after returning retrieve them back

**Callee Saved Registers:** If it is required to use them in the called function, first make a backup in the stack, use them and then restore them to original values before executing jr $ra

* + 1. Instructions for Memory Read/Write

|  |  |
| --- | --- |
| **lw** $1, Offset($2) | Load Word (32 bits) from memory to a Register $1. Address of memory location is (offset + Register $2). Offset is any constant value and should be a multiple of 4 since one word (4-bytes) are read at a time.  $1 = Memory[$2 + Offset] |
| **sw** $1, Offset($2) | Store Word (32 bits) to memory from Register $1. Remaining details as above. |

Table 3.2

Offset must be a multiple of 4 because 1word = 4bytes

For example, if 0 is the base address of an array Memory

li $t1,0

lw $t2,4($t1)

will result in

$t2 = Memory[1]

Because Memory is byte addressable so successive elements(words) are separated by 4 bytes

And,

li $t2,86

li $t1,8

sw $t2,12($t1)

will result in

Memory[3+2] = 86

* + 1. Stack

Stack is an area of memory to store variables temporarily. If there is a shortage of registers, valuable data from a register can be moved to stack before using the register. After the register has been used, data can be restored from stack. Stack also has a pointer ($sp for MIPS) which points to the last memory location allocated. Stack grows from top to bottom (Higher memory address to lower memory address).

The registers in an ALU, if compared with C language, are like global variables. If you change them in a function, the same value is then available for all the code in all the functions. So, it is required sometime, to save their values in stack to restore them afterwards. In order to store data on stack, you should first reserve the place by just decreasing the value of $sp and then save the values using sw. After the registers are loaded back you must restore the stack pointer to its previous value. For example, we need to save three registers: $s0, $t0, and $t1 before using them for some operation where the important values inside these registers will be lost. We “push” the old values onto the stack by creating space for three words on the stack and then store them. Stack grows from top to bottoms

.text

#Storing current values of $s0, $t0, and $t1

addi $sp,$sp,-12 # adjust stack to make room for 3 items

# 4bytes for each means 12 bytes)

sw $t1, 8($sp) # save register $t1 for use afterwards

sw $t0, 4($sp) # save register $t0 for use afterwards

sw $s0, 0($sp) # save register $s0 for use afterwards

#Code where $s0, $t0, and $t1 are filled with new values …………………

…………………

#Now finally restoring the values of $s0, $t0, and $t1

lw $t1, 8($sp) # restore value of $t1

lw $t0, 4($sp) # restore value of $t0

lw $s0, 0($sp) # restore value of $s0

addi $sp,$sp,+12 # restore stack pointer to previous value



Figure 3.1 The values of the stack pointer and the stack (a) before, (b) during, and (c) after the procedure call

* In-Lab

Task (Code should contain maximum use of procedures)

* Rewrite the first program of previous lab (Total Marks without loop) using procedures. It should contain procedures display\_string and get\_input.
* Rewrite the Total Marks code. The code in the main label should call two procedure get\_marks and calculate\_sum. The get\_marks procedure should further call display\_string and get\_input procedures.
* Post-Lab

Write a program that first inputs total no. of student. It then gets student roll no. and marks secured by that student, and stores both in memory. After that it waits in a loop asking to enter roll no. On getting roll no. it displays the marks secured by that roll no. If user enters roll no 9999, it terminates.

* **Your lab report, a .doc file, should contain properly commented Post-Lab task code, MARS screenshot for each code which clearly show the register contents after execution and I/O panel, and critical analysis.**
* **The report must have a title page.**
* **Name the .doc file RegNo.docx; eg SP14-BCE-99.docx**
* **Upload it on portal**
* **Deadline for Lab Report submission is before start of the next Lab.**