What you will learn.
In this chapter you will learn:

1. How to create a subprogram.
2. The `jal` (call subprogram) and `jr $ra` (return from subprogram) operators.
3. The `.include` MARS assembly directive.
4. How to pass parameters to and retrieve return values from a subprogram.
5. How the Program Counter register (`$pc`) the control program execution sequence.
6. How to structure and comment a file of subprograms.
7. Why the subprograms in this chapter cannot call themselves or other subprograms.

Chapter 5 Simple MIPS subprograms[1]

The programs that were implemented in Chapter 3 were very long and required a lot of code to implement simple operations. Many of these operations were common to more than one of the programs and having to handle the details of these operations each time was distracting and a possible source of errors. For example, every program needs to use syscall service 10 to exit a program. It would be nice to abstract the method of exiting the program once and then use this abstraction in every program. One way to abstract data is to use subprograms. Something like subprograms exist in every language and go by the names: functions, procedures, or methods.[2]

The concept of a subprogram is to create a group of MIPS assembly language statements which can be called (or dispatched) to perform a task. A good example is an abstraction we will call "Exit" which will automatically exit the program for the programmer.

These subprograms are called simple subprograms because they will be allowed to call any other subprograms and they will not be allowed to modify any save registers. These limits on subprograms simplify the implementation of subprograms tremendously. These limits will be taken away Chapter 8, which will allow the creation of much more powerful, but also much more complex, subprograms.

Chapter 5. 1 Exit Subprogram

The first subprogram to be introduced will be called `Exit` and will call `syscall` service 10 to exit the program. Because there will be no need to return to the calling program, this subprogram can be created by creating a label `Exit` and putting code after that label which sets up the call to `syscall` and executes it. This is shown in the following program that prompts a user for an integer, prints it out, and exits the program.

[1] These subprograms are simple because they are *non-reentrant*. In simple terms, this means that these subprograms cannot call other subprograms and cannot call themselves. The reason they are non-reentrant will be explained in the chapter. How to make reentrant subprograms requires a concept called a program stack, and will be covered in chapter 8.

[2] All subprograms in this chapter could be implemented using Mars macros. However there is real value in introducing the concept of subprograms at this point in the text, and so macros will not be covered.

```
# File:     Program5-1.asm
# Author:   Charles Kann
# Purpose:  Illustrates a subprogram named Exit

.text
main:
    # Read an input value from the user
    li $v0, 4
    la $a0, prompt
    syscall
    li $v0, 5
    syscall
    move $s0, $v0

    # Print the value back to the user
    li $v0, 4
    la $a0, result
    syscall
    li $v0, 1
    move $a0, $s0
    syscall

    # Call the Exit subprogram to exit
    jal Exit

.data
    prompt: .asciiz "Please enter an integer: "
    result: .asciiz "\nYou entered: "

# Subprogram:     Exit
# Author:         Charles Kann
# Purpose:        to use syscall service 10 to exit a program
# Input:          None
# Output:         None
# Side effects:   The program is exited

.text
Exit:
    li $v0, 10
    syscall
```

Program 5-1: Implementing and calling Exit subprogram

Chapter 5.1. 1 Commentary on Exit subprogram

1. This program has two `.text` sections. This is because the `Exit` subprogram is not part of the main and so exists in a different part of the file. Eventually the subprogram will be put into another file and included when needed. There can be as many `.text` and `.data` statements in assembly language as are needed and they can be interspersed as they are here. They inform the assembler what type of information follows. When in a `.text` segment, there should be only program text, in a `.data` segment, only data. Later these will also be associated with segments of memory used.

2. The `jal` instruction transfers control of the computer to the address at the label `Exit`. For the present the reader can think of this as *calling* or *executing* a method or function in a HLL. However, the wording used here of transferring control is much more accurate and this will become clear later in the chapter.

3. The `Exit` subprogram halts the program so there is no return statement when it is complete. This is a special case and all other subprograms in this text will execute a return using the `jr $ra` instruction. This means that in this case a jump (`j`) operator could have been used to transfer control to the `Exit` subprogram. To be consistent with all other subprogram, this text will always use `jal`.

4. Programmers should always attempt to be consistent with naming conventions. This text will use the conventions of camel casing with the first word beginning with a lower case for variables and labels in a program (e.g., inputLabel) and camel casing with the first word beginning with a capital letter for subprogram (e.g., InputInteger). There is no industry-wide standard for naming conventions as with Java or C#, so the reader is free to choose their own standard. But once chosen, it should be followed.

5. All subprograms should, at a minimum, contain the commented information provided in this example. The programmer should state the program name, author, purpose, inputs, outputs, and side effects of running the program. The input and output variables are especially important as they will be registers and not be named as in a HLL. It is very hard after the program is written to remember what these values represent and not documenting them makes the subprogram very difficult to use and maintain.

Chapter 5. 2 PrintNewLine subprogram

The next subprogram to be implemented will print a new line character. This subprogram will allow the program to stop inserting the control character "\n" into their programs. It will be used here to illustrate how to return from a subprogram.

```
# File:     Program5-2.asm
# Author:   Charles Kann
# Purpose:  Illustrates calling a subprogram named PrintNewLine.

.text
main:
    # Read an input value from the user
    li $v0, 4
    la $a0, prompt
    syscall
    li $v0, 5
    syscall
    move $s0, $v0
```

```
        # Print the value back to the user
        jal PrintNewLine
        li $v0, 4
        la $a0, result
        syscall
        li $v0, 1
        move $a0, $s0
        syscall

        # Call the Exit subprogram to exit
        jal Exit

    .data
        prompt: .asciiz "Please enter an integer: "
        result: .asciiz "You entered: "

    # Subprogram:     PrintNewLine
    # Author:         Charles Kann
    # Purpose:        to output a new line to the user console
    # Input:          None
    # Output:         None
    # Side effects:   A new line character is printed to the user's console

    .text
    PrintNewLine:
        li $v0, 4
        la $a0, __PNL_newline
        syscall
        jr $ra
    .data
       __PNL_newline:   .asciiz "\n"

    # Subprogram:     Exit
    # Author:         Charles Kann
    # Purpose:        to use syscall service 10 to exit a program
    # Input:          None
    # Output:         None
    # Side effects:   The program is exited

    .text
    Exit:
        li $v0, 10
        syscall
```

Program 5-2: Implementing and calling the PrintNewLine subprogram

Chapter 5.2. 1 Commentary on PrintNewLine subprogram

1. This program now has three `.text` segments. The `PrintNewLine` subprogram requires a value to be stored in the `.data` segment, the newline character. The `.text` segments are needed to inform the assembler that program instructions are again contained in the code. This is emphasized here as it points out a good rule to follow when writing subprograms. Always begin the subprogram with a `.text` statement. If the assembler already thinks it is in a `.text` segment, there is no effect, but the subprogram is protected from the case where the assembler thinks it is assembling a `.data` segment when reaching the subprogram. In real life

files can change often and omitting a simple `.text` or `.data` segment when it should be present can lead to myriads of unnecessary problems.

2. The PrintNewLine subprogram shows how to return from a subprogram using the instruction "`jr $ra`". The next section of this chapter will explain how this works. For now, this can be thought of as a *return* statement.

3. A label was needed in the PrintNewLine subprogram to contain the address of the newline variable. In MIPS assembly, you cannot use labels with the same name[3]. This subprogram will eventually become part of a utility package which will be used by several programs. Care must be taken to make sure that any label used will not conflict with a label in a program. So, the convention of putting a double underscore (__) before the variable, a string representing the subprogram it is in (PNL), and another underscore before the variable name (newline) is used. It is hoped that this will solve nearly all name conflicts.

4. The `$a0` and `$v0` registers have been changed, but this is not listed as a side effect of the program. Part of the definition of the registers says that the save registers (`$s0…$s7`) must contain the same value when the subprogram returns as they did when the subprogram was called. For these simple subprograms, that means that these registers cannot be used. All other registers have no guarantee of the value after the subprogram is called. A programmer cannot rely on the values of `$a0` or `$v0` once a program is called. So, while changing them is a side effect of calling the function, it does not have to be listed as it is implied in the execution of the method.

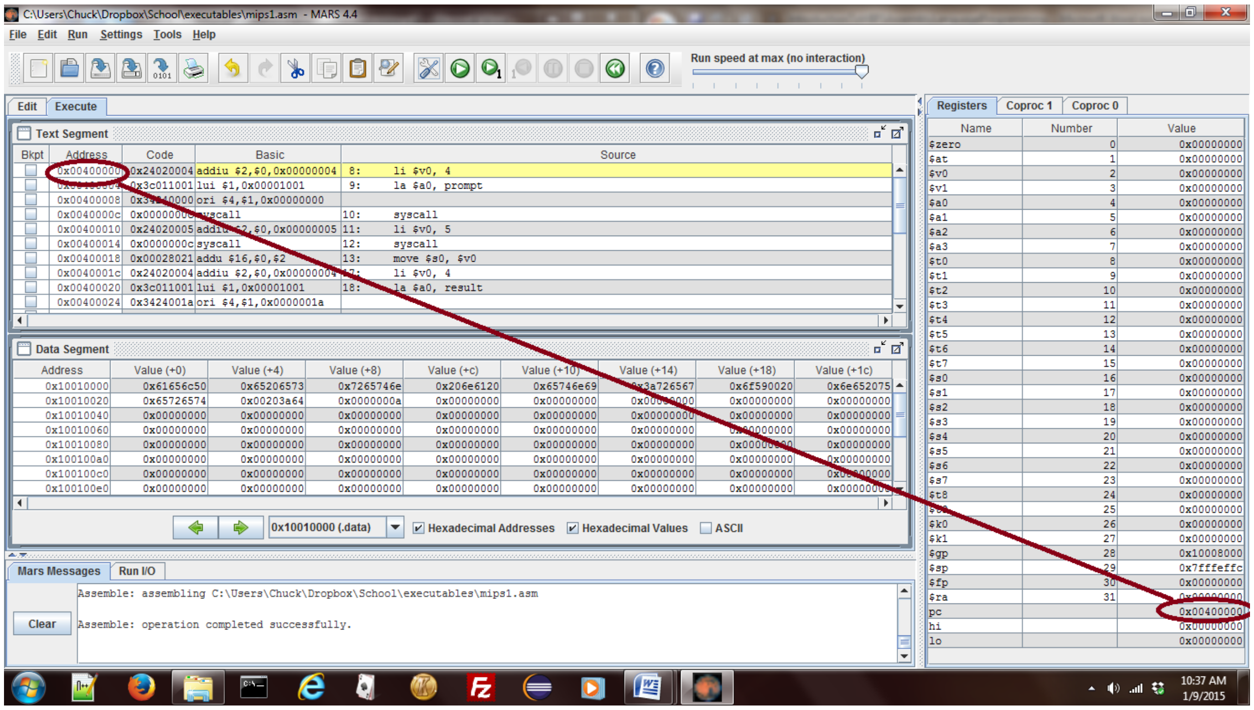
Chapter 5. 3 The Program Counter ($pc) register and calling a subprogram

One of the most important registers in the CPU is the `$pc` register. Note that it is not one of the 32 general purpose registers that the programmer can access directly. It is a special register that keeps track of the address in memory of the next instruction to be executed. For example, consider Program 5-2, PrintNewLine. Compile this program and bring up the MARS execution screen, as shown in the Figure 5.1. The address column shows the memory address where each instruction is stored. Note that the first line of the program, which is highlighted in yellow because it is the next instruction to execute, is at 0x00400000, and the `$pc` register contains the value 0x00400000. Step to the next instruction, as in Figure 5.2, and you will see that the `$pc` contains the value 00x00400004, which is the address of the next instruction to execute. Thus the `$pc` specifies the next instruction to be executed.

The screen shot in Figure 5.3 is at the `jal PrintNewLine` instruction. Now the PC points to 0x0040001c and the next statement would be 0x00400020. However, note the translation of the `jal PrintNewLine` instruction in the basic column has the address 0x00400040 in it. This address is the first line in the `PrintNewLine` subprogram. When the `jal` instruction is executed, the `$pc` register is set to 0x00400040 and the program continues by executing in the

[3] More accurately, in MIPS you cannot use labels with the same name unless the labels are resolved to addresses so that they do not conflict. If separate assembly and linking are performed, any none global symbols can be used in any file, so long as it is in that file once. This text does not handle separate compilation and linking, so the rule that label names cannot be repeated is correct for all programs in this text.

`PrintNewLine` subprogram, as shown in Figure 5.4. This is how the `jal` operator transfers the control of the computer to run the subprogram.


C:\Users\Chuck\Dropbox\School\executables\mips1.asm - MARS 4.4
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Text Segment
Data Segment
Registers Coproc 1 Coproc 0
Mars Messages Run I/O
Clear
0x10010000 (.data)
☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII
10:37 AM 1/9/2015

Bkpt	Address	Code	Basic	Source
☐	0x00400000	0x24020004	addiu $2,$0,0x00000004	8: li $v0, 4
☐	0x00400004	0x3c011001	lui $1,0x00001001	9: la $a0, prompt
☐	0x00400008	0x34240000	ori $4,$1,0x00000000	
☐	0x0040000c	0x0000000c	syscall	10: syscall
☐	0x00400010	0x24020005	addiu $2,$0,0x00000005	11: li $v0, 5
☐	0x00400014	0x0000000c	syscall	12: syscall
☐	0x00400018	0x00028021	addu $16,$0,$2	13: move $s0, $v0
☐	0x0040001c	0x24020004	addiu $2,$0,0x00000004	17: li $v0, 4
☐	0x00400020	0x3c011001	lui $1,0x00001001	18: la $a0, result
☐	0x00400024	0x3424001a	ori $4,$1,0x0000001a	

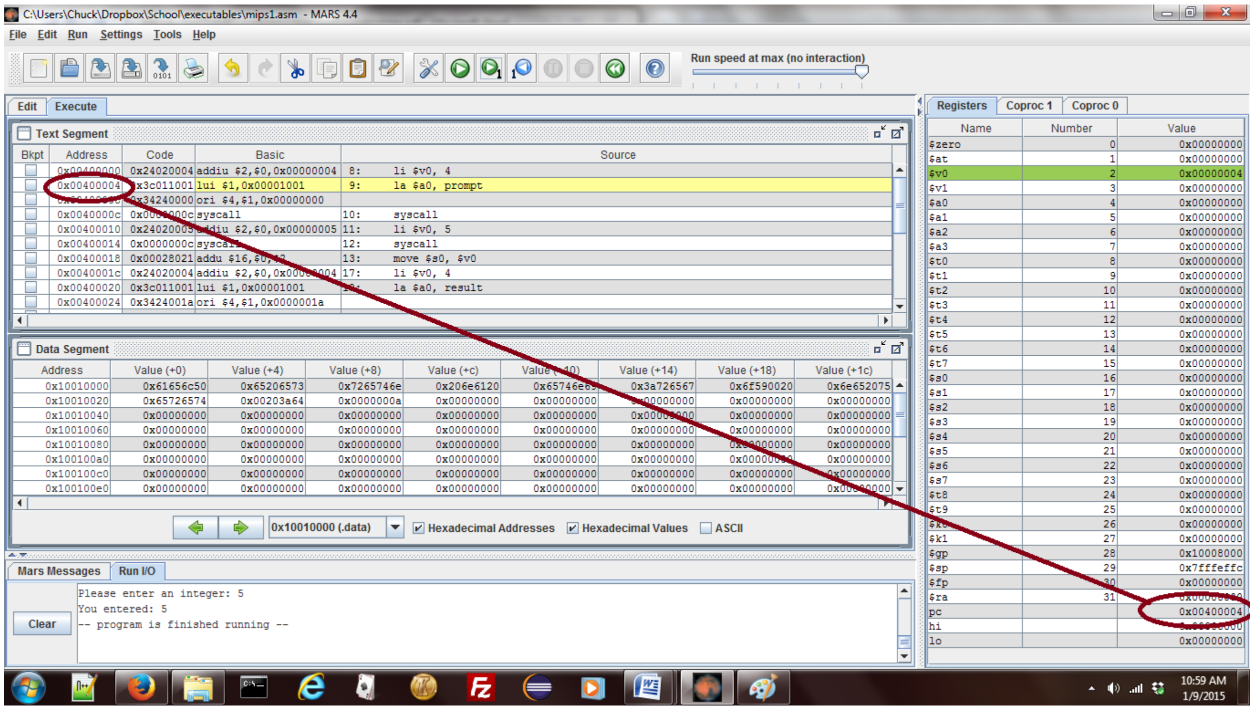
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x61656c50	0x65206573	0x7265746e	0x206e6120	0x65746e69	0x3a726567	0x6f590020	0x6e652075
0x10010020	0x65726574	0x00203a64	0x0000000a	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

```
Assemble: assembling C:\Users\Chuck\Dropbox\School\executables\mips1.asm

Assemble: operation completed successfully.
```

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Figure 5-1: `$pc` when program execution starts


C:\Users\Chuck\Dropbox\School\executables\mips1.asm - MARS 4.4
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Text Segment
Data Segment
Registers Coproc 1 Coproc 0
Mars Messages Run I/O
Clear
0x10010000 (.data)
☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII
10:59 AM 1/9/2015

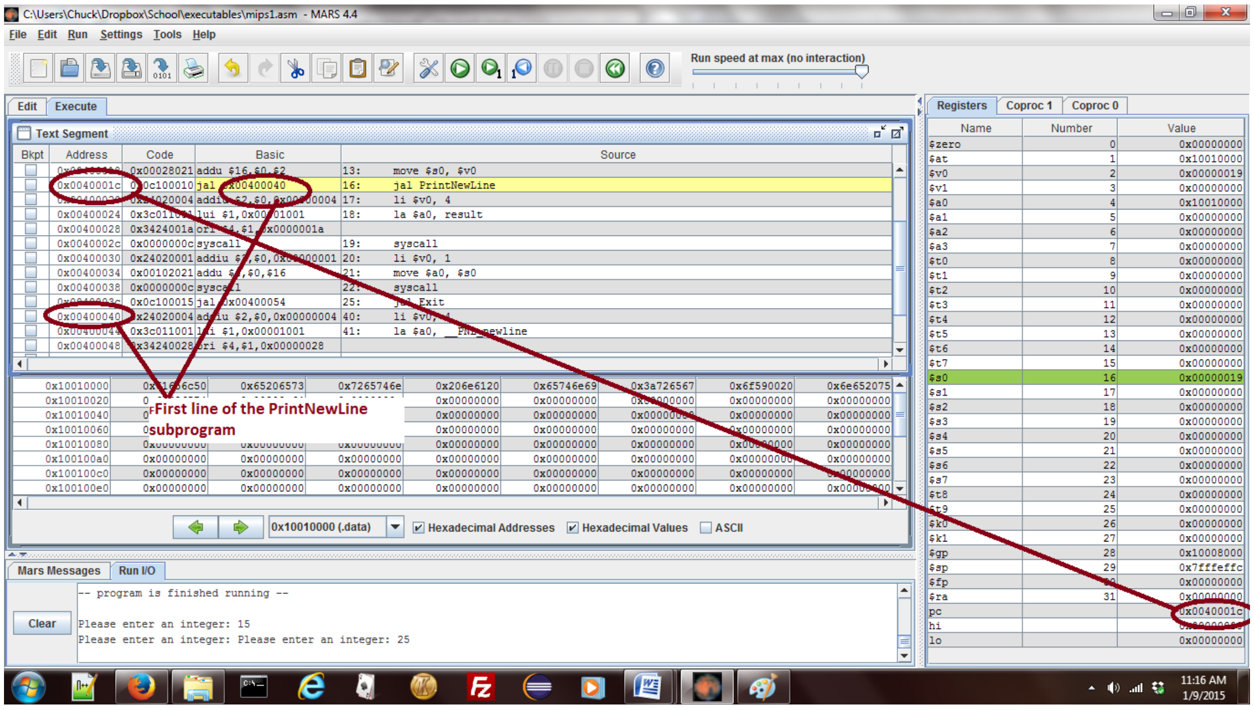
Bkpt	Address	Code	Basic	Source
☐	0x00400000	0x24020004	addiu $2,$0,0x00000004	8: li $v0, 4
☐	0x00400004	0x3c011001	lui $1,0x00001001	9: la $a0, prompt
☐	0x00400008	0x34240000	ori $4,$1,0x00000000	
☐	0x0040000c	0x0000000c	syscall	10: syscall
☐	0x00400010	0x24020005	addiu $2,$0,0x00000005	11: li $v0, 5
☐	0x00400014	0x0000000c	syscall	12: syscall
☐	0x00400018	0x00028021	addu $16,$0,$2	13: move $s0, $v0
☐	0x0040001c	0x24020004	addiu $2,$0,0x00000004	17: li $v0, 4
☐	0x00400020	0x3c011001	lui $1,0x00001001	18: la $a0, result
☐	0x00400024	0x3424001a	ori $4,$1,0x0000001a	

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x61656c50	0x65206573	0x7265746e	0x206e6120	0x65746e69	0x3a726567	0x6f590020	0x6e652075
0x10010020	0x65726574	0x00203a64	0x0000000a	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

```
Please enter an integer: 5
You entered: 5
-- program is finished running --
```

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000004
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400004
hi		0x00000000
lo		0x00000000

Figure 5-2: `$pc` after the execution of the first instruction


C:\Users\Chuck\Dropbox\School\executables\mips1.asm - MARS 4.4
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Text Segment
First line of the PrintNewLine subprogram
0x10010000 (.data)
☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII
Mars Messages Run I/O
Clear
Registers Coproc 1 Coproc 0
11:16 AM 1/9/2015

```
Bkpt Address    Code       Basic                       Source
     0x00400018 0x00028021 addu $16,$0,$2              13:     move $s0, $v0
     0x0040001c 0x0c100010 jal 0x00400040              16:     jal PrintNewLine
     0x00400020 0x24020004 addiu $2,$0,0x00000004      17:     li $v0, 4
     0x00400024 0x3c011001 lui $1,0x00001001           18:     la $a0, result
     0x00400028 0x3424001a ori $4,$1,0x0000001a
     0x0040002c 0x0000000c syscall                     19:     syscall
     0x00400030 0x24020001 addiu $2,$0,0x00000001      20:     li $v0, 1
     0x00400034 0x00102021 addu $4,$0,$16              21:     move $a0, $s0
     0x00400038 0x0000000c syscall                     22:     syscall
     0x0040003c 0x0c100015 jal 0x00400054              25:     jal Exit
     0x00400040 0x24020004 addiu $2,$0,0x00000004      40:     li $v0, 4
     0x00400044 0x3c011001 lui $1,0x00001001           41:     la $a0, __PNL_newline
     0x00400048 0x34240028 ori $4,$1,0x00000028
```

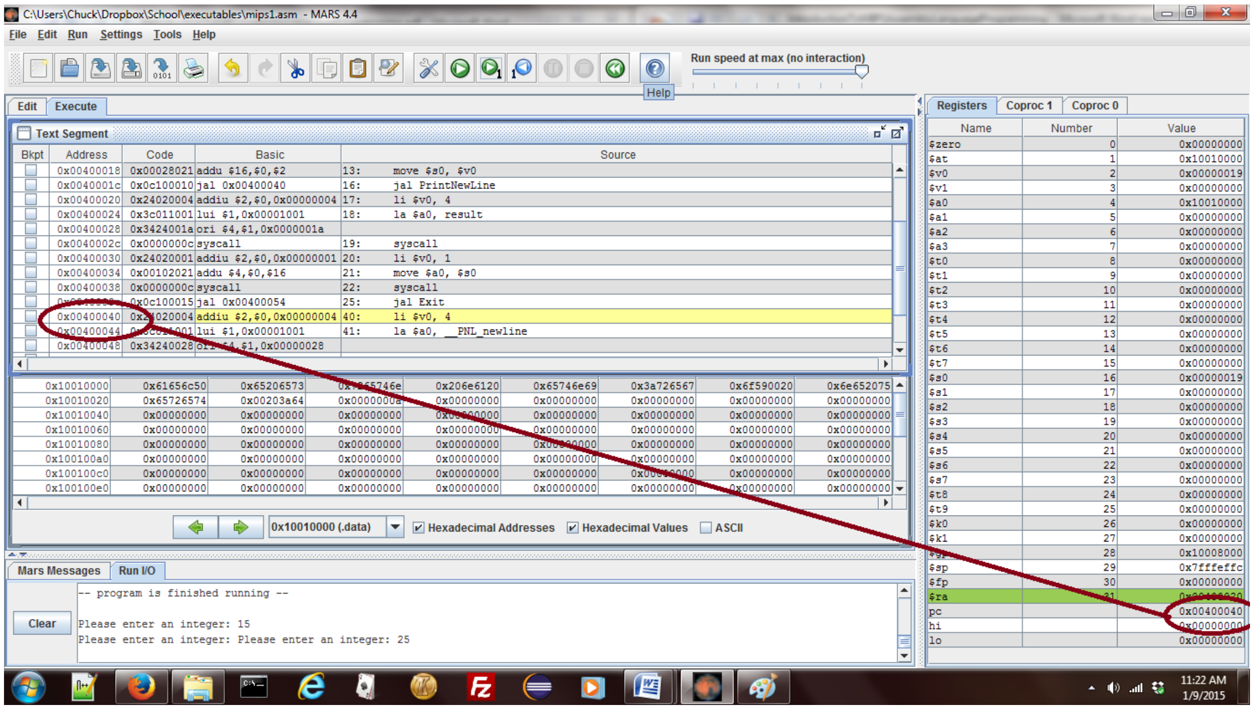
```
0x10010000  0x61656c50  0x65206573  0x7265746e  0x206e6120  0x65746e69  0x3a726567  0x6f590020  0x6e652075
0x10010020  [illegible]                         0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x10010040  [illegible]                         0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x10010060  [illegible]                         0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x10010080  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x100100a0  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x100100c0  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x100100e0  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
```

```
-- program is finished running --

Please enter an integer: 15
Please enter an integer: Please enter an integer: 25
```

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x00000019
$v1	3	0x00000000
$a0	4	0x10010000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000019
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x0040001c
hi		0x00000000
lo		0x00000000

Figure 5-3: Just before calling PrintNewLine subprogram


C:\Users\Chuck\Dropbox\School\executables\mips1.asm - MARS 4.4
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Help
Edit Execute
Text Segment
0x10010000 (.data)
☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII
Mars Messages Run I/O
Clear
Registers Coproc 1 Coproc 0
11:22 AM 1/9/2015

```
Bkpt Address    Code       Basic                       Source
     0x00400018 0x00028021 addu $16,$0,$2              13:     move $s0, $v0
     0x0040001c 0x0c100010 jal 0x00400040              16:     jal PrintNewLine
     0x00400020 0x24020004 addiu $2,$0,0x00000004      17:     li $v0, 4
     0x00400024 0x3c011001 lui $1,0x00001001           18:     la $a0, result
     0x00400028 0x3424001a ori $4,$1,0x0000001a
     0x0040002c 0x0000000c syscall                     19:     syscall
     0x00400030 0x24020001 addiu $2,$0,0x00000001      20:     li $v0, 1
     0x00400034 0x00102021 addu $4,$0,$16              21:     move $a0, $s0
     0x00400038 0x0000000c syscall                     22:     syscall
     0x0040003c 0x0c100015 jal 0x00400054              25:     jal Exit
     0x00400040 0x24020004 addiu $2,$0,0x00000004      40:     li $v0, 4
     0x00400044 0x3c011001 lui $1,0x00001001           41:     la $a0, __PNL_newline
     0x00400048 0x34240028 ori $4,$1,0x00000028
```

```
0x10010000  0x61656c50  0x65206573  0x7265746e  0x206e6120  0x65746e69  0x3a726567  0x6f590020  0x6e652075
0x10010020  0x65726574  0x00203a64  0x0000000a  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x10010040  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x10010060  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x10010080  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x100100a0  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x100100c0  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x100100e0  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
```

```
-- program is finished running --

Please enter an integer: 15
Please enter an integer: Please enter an integer: 25
```

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x00000019
$v1	3	0x00000000
$a0	4	0x10010000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000019
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00400020
pc		0x00400040
hi		0x00000000
lo		0x00000000

Figure 5-4: Program has transferred control into the PrintNewLine subprogram

Chapter 5. 4 Returning from a subprogram and the `$ra` register

As the previous section has shown, the `$pc` register controls the next statement to be executed in a program. When calling a subprogram, the `$pc` is set to the first line in the subprogram, so to return from the subprogram it is simply a matter of putting into the `$pc` the address of the instruction to execute when returning from the subprogram. To find this return address, the mechanics of a subprogram call must be defined.

No matter what HLL the reader has used, the semantics of a subprogram call is always the same. When the call is executed the program transfers control to the subprogram. When the subprogram is finished, control is transferred back to the next statement immediately following the subprogram call in the calling subprogram. This is illustrated in Figure 5.5 below.

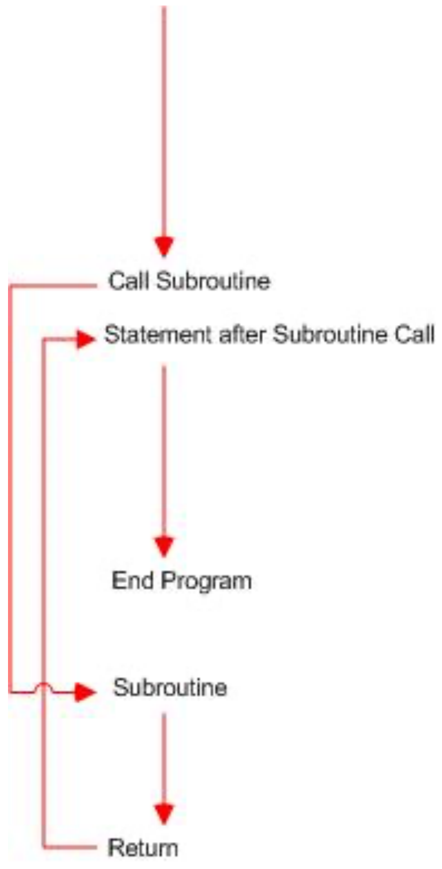
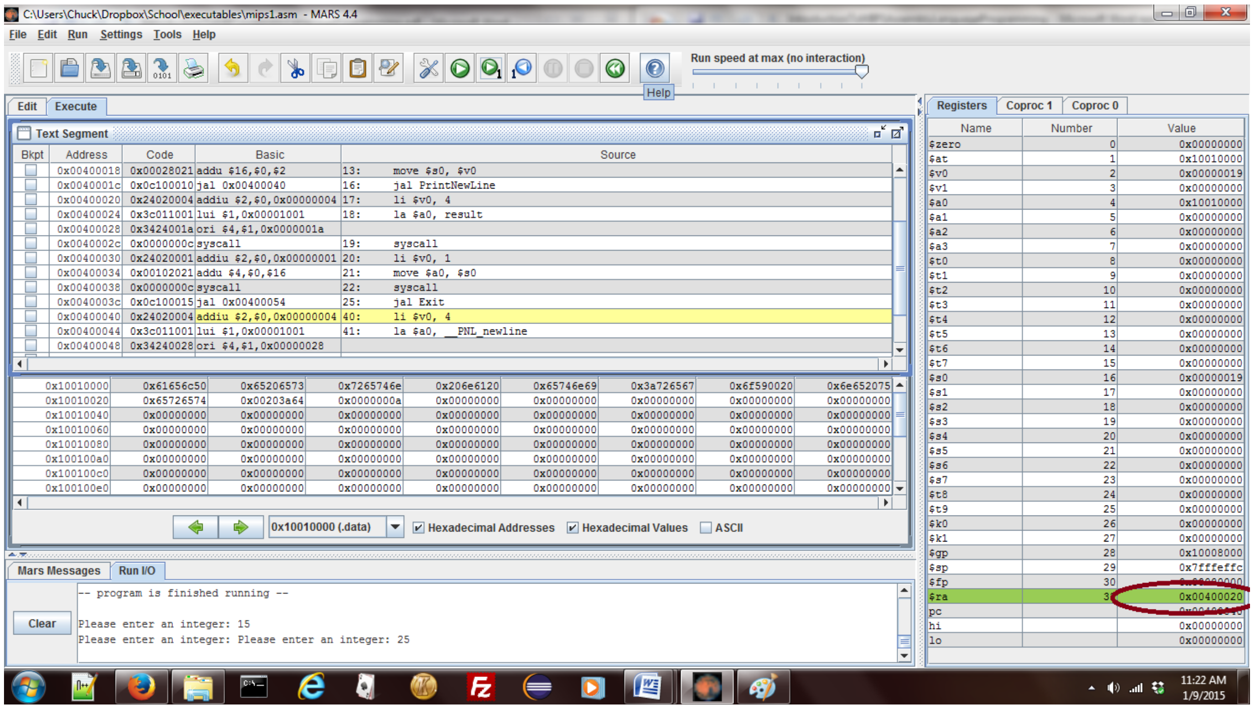


Figure 5-5: Subprogram calling semantics

When the subprogram is called the next sequential address of an instruction must be stored and when the subprogram is complete, the control must branch back to that instruction. In the example in the last section, Figure 5.3 showed the subprogram call was at instruction 0x0040001c, and the next sequential instruction would have been 0x00400020. So, the value 0x00400020 must be stored somewhere and, when the end of the subroutine is reached, the program must transfer control back to that statement.

Figure 5.6 is a screen shot when the program is executing at the first line in the PrintNewLine subroutine. This is the same as Figure 5.4, but now the register circled is the `$ra` register. Note that in the `$ra` register is stored the address 0x00400020. This tells us that the `jal` operator not only sets the `$pc` register to the value of the label representing the first line of the subprogram, it also sets the `$ra` register to point back to the next sequential line, which is the return address used after the subprogram completes.



C:\Users\Chuck\Dropbox\School\executables\mips1.asm - MARS 4.4

Run speed at max (no interaction)

Text Segment

```
Bkpt Address    Code       Basic                        Source
     0x00400018 0x00028021 addu $16,$0,$2               13:     move $s0, $v0
     0x0040001c 0x0c100010 jal 0x00400040               16:     jal PrintNewLine
     0x00400020 0x24020004 addiu $2,$0,0x00000004       17:     li $v0, 4
     0x00400024 0x3c011001 lui $1,0x00001001            18:     la $a0, result
     0x00400028 0x3424001a ori $4,$1,0x0000001a
     0x0040002c 0x0000000c syscall                      19:     syscall
     0x00400030 0x24020001 addiu $2,$0,0x00000001       20:     li $v0, 1
     0x00400034 0x00102021 addu $4,$0,$16               21:     move $a0, $s0
     0x00400038 0x0000000c syscall                      22:     syscall
     0x0040003c 0x0c100015 jal 0x00400054               25:     jal Exit
     0x00400040 0x24020004 addiu $2,$0,0x00000004       40:     li $v0, 4
     0x00400044 0x3c011001 lui $1,0x00001001            41:     la $a0, __PNL_newline
     0x00400048 0x34240028 ori $4,$1,0x00000028
```

```
0x10010000  0x61656c50  0x65206573  0x7265746e  0x206e6120  0x65746e69  0x3a726567  0x6f590020  0x6e652075
0x10010020  0x65726574  0x00203a64  0x0000000a  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x10010040  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x10010060  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x10010080  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x100100a0  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x100100c0  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
0x100100e0  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000
```

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII

Run I/O

```
-- program is finished running --

Please enter an integer: 15
Please enter an integer: Please enter an integer: 25
```

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x00000019
$v1	3	0x00000000
$a0	4	0x10010000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000019
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00400020
pc		0x00400040
hi		0x00000000
lo		0x00000000

11:22 AM 1/9/2015

Figure 5-6: Saving the `$ra` register

To summarize: When a subprogram is called, the `jal` operator updates `$pc` to point to the address of the label for the subprogram. The `jal` operator also sets `$ra` to the next address after the subprogram call. When the "`jr $ra`" instruction is called, the subprogram *returns* by transferring control to the address in the `$ra` register.

This also explains why a subprogram in this chapter cannot call another subprogram. When the first subprogram is called, the value for the return is saved in `$ra`. When the second subprogram is called, the value in `$ra` is changed and the original value is lost. This problem will be explored more in the exercises at the end of the chapter.

Chapter 5. 5 Input parameter with PrintString subprogram

The next subprogram to be implemented is `PrintString`. It abstracts the ability to print a string. This subprogram will require an input parameter to be passed into the program, the address of the string to print. Remembering the register conventions from chapter 2, the registers `$a0…$a3` are used to pass input parameters into a program, so the register `$a0` will be used for the input parameter to this subprogram. The subprogram will then load a 4 into `$v0`, invoke `syscall`, and return.

The program below is from Chapter 5.1 and is modified to use the `PrintString` program. The changes to the program are highlighted in yellow.

```
# File:     Program5-3.asm
# Author:   Charles Kann
# Purpose:  Illustrates implementing a subprogram named PrintNewLine.

.text
main:
    # Read an input value from the user
    la $a0, prompt
    jal PrintString
    li $v0, 5
    syscall
    move $s0, $v0

    # Print the value back to the user
    jal PrintNewLine
    la $a0, result
    jal PrintString
    li $v0, 1
    move $a0, $s0
    syscall

    # Call the Exit subprogram to exit
    jal Exit

.data
    prompt: .asciiz "Please enter an integer: "
    result: .asciiz "You entered: "

# Subprogram:     PrintNewLine
# Author:         Charles Kann
# Purpose:        to output a new line to the user console
# Input/Output:   None
# Side effects:   A new line character is printed to the user's console

.text
PrintNewLine:
    li $v0, 4
    la $a0, __PNL_newline
    syscall
    jr $ra
.data
   __PNL_newline:   .asciiz "\n"

# Subprogram:     PrintString
# Author:         Charles W. Kann
# Purpose:        To print a string to the console
# Input:          $a0 - The address of the string to print.
# Returns:        None
# Side effects:   The String is printed to the console.

.text
PrintString:
    addi $v0, $zero, 4
    syscall
    jr $ra
```

```
# Subprogram:     Exit
# Author:         Charles Kann
# Purpose:        to use syscall service 10 to exit a program
# Input/Output:   None
# Side effects:   The program is exited

.text
Exit:
    li $v0, 10
    syscall
```

Program 5-3: Input parameter with the PrintString subprogram

Chapter 5. 6 Multiple input parameters with PrintInt subprogram

When printing an integer value, the program should always print a string first to tell the user what is being printed and then the integer is printed. This behavior will be abstracted in the following subprogram, `PrintInt`. In this subprogram there are two input parameters: the address of the string to print, which is stored in `$a0`; and the integer value to print, which is stored in `$a1`. Other than passing in two parameters and creating a slightly larger subprogram, this program does not add any new material. The changes are highlighted in yellow.

```
# File:     Program5-4.asm
# Author:   Charles Kann
# Purpose:  Illustrates implementing a subprogram named PrintNewLine.

.text
main:
    # Read an input value from the user
    la $a0, prompt
    jal PrintString
    li $v0, 5
    syscall
    move $s0, $v0

    # Print the value back to the user
    jal PrintNewLine
    la $a0, result
    move $a1, $s0
    jal PrintInt

    # Call the Exit subprogram to exit
    jal Exit

.data
    prompt: .asciiz "Please enter an integer: "
    result: .asciiz "You entered: "

# Subprogram:     PrintNewLine
# Author:         Charles Kann
# Purpose:        to output a new line to the user console
# Input/Output:   None
# Side effects:   A new line character is printed to the user's console
```

```
.text
PrintNewLine:
    li $v0, 4
    la $a0, __PNL_newline
    syscall
    jr $ra
.data
   __PNL_newline:   .asciiz "\n"

# Subprogram:     PrintInt
# Author:   Charles W. Kann
# Purpose:  To print a string to the console
# Input:    $a0 - The address of the string to print.
#           $a1 - The value of the int to print
# Returns:  None
# Side effects:   The String is printed followed by the integer value.

.text
PrintInt:
    # Print string.  The string address is already in $a0
    li $v0, 4
    syscall

    # Print integer.   The integer value is in $a1, and must
    # be first moved to $a0.
    move $a0, $a1
    li $v0, 1
    syscall

    #Return
    jr $ra

# Subprogram:     PrintString
# Author:   Charles W. Kann
# Purpose:  To print a string to the console
# Input:    $a0 - The address of the string to print.
# Returns:  None
# Side effects:   The String is printed to the console.

.text
PrintString:
    addi $v0, $zero, 4
    syscall
    jr $ra

# Subprogram:     Exit
# Author:         Charles Kann
# Purpose:        to use syscall service 10 to exit a program
# Input/Output:   None
# Side effects:   The program is exited

.text
Exit:
    li $v0, 10
    syscall
```

Program 5-4: Multiple input parameters with PrintInt subprogram

Chapter 5. 7 Return values with PromptInt subprogram

The next subprogram in this chapter, PromptInt, shows how to return a value from a subprogram. The registers `$a0…$a3` are used to pass values into a subprogram, and the registers `$v0` & `$v1` are used to return values. The program from Chapter 5.6 has been changed to include the PromptInt subprogram and the changes are highlighted in yellow.

```
# File:     Program5-5.asm
# Author:   Charles Kann
# Purpose:  Illustrates implementing a subprogram named PrintNewLine.

.text
main:
    # Read an input value from the user
    la $a0, prompt
    jal PromptInt
    move $s0, $v0

    # Print the value back to the user
    jal PrintNewLine
    la $a0, result
    move $a1, $s0
    jal PrintInt

    # Call the Exit subprogram to exit
    jal Exit

.data
    prompt: .asciiz "Please enter an integer: "
    result: .asciiz "You entered: "

# Subprogram:     PrintNewLine
# Author:         Charles Kann
# Purpose:        to output a new line to the user console
# Input/Output:   None
# Side effects:   A new line character is printed to the user's console

.text
PrintNewLine:
    li $v0, 4
    la $a0, __PNL_newline
    syscall
    jr $ra

.data
   __PNL_newline:   .asciiz "\n"

# Subprogram:     PrintInt
# Author:         Charles W. Kann
# Purpose:        To print a string to the console
# Input:          $a0 - The address of the string to print.
#                 $a1 - The value of the int to print
# Output:         None
# Side effects:   The String is printed followed by the integer value.
```

```
.text
PrintInt:
    # Print string. The string address is already in $a0
    li $v0, 4
    syscall

    # Print integer. The value is in $a1 & must be first moved to $a0.
    move $a0, $a1
    li $v0, 1
    syscall

    #Return
    jr $ra

# Subprogram:     PromptInt
# Author:         Charles W. Kann
# Purpose:        Prompt for integer & return that value to the caller.
# Input:          $a0 - The address of the string to print.
# Output:         $v0 - The value the user entered
# Side effects:   The String is printed followed by the integer value.

.text
PromptInt:
    # Print the prompt, which is already in $a0
    li $v0, 4
    syscall

    # Read the integer.  Note: at the end of the syscall the value is
    #  already in $v0, so there is no need to move it anywhere.
    li $v0, 5
    syscall

    #Return
    jr $ra

# Subprogram:     PrintString
# Author:         Charles W. Kann
# Purpose:        To print a string to the console
# Input:          $a0 - The address of the string to print.
# Output:         None
# Side effects:   The String is printed to the console.

.text
PrintString:
    addi $v0, $zero, 4
    syscall
    jr $ra

# Subprogram:     Exit
# Author:         Charles Kann
# Purpose:        to use syscall service 10 to exit a program
# Input/Output:   None
# Side effects:   The program is exited

.text
Exit:
    li $v0, 10
    syscall
```

Chapter 5. 8 Create a utils.asm file

The final step is to add the subprograms developed in this chapter in a separate file so that any program you write can use them. To do this, create a file called *utils.asm*, and copy all the subprograms (including comments) into this file. This utilities file needs to be correctly documented. The preamble comment (the comment at the start of the file) should, at a minimum, contain the following information:

- The name of the file, so that it can be found.
- The purpose of the file, in this case what type of subprograms are defined in it.
- The initial author.
- A copyright statement.
- An index so that a user can quickly find the subprograms in the file and determine if they are of use to the programmer. Ideally this index should be in alphabetic order.
- A modification history. Every change to this file should be documented here. Even innocuous changes like new comments should be recorded.

The entire file, utils.asm, is included here to show what it should look like when completed.

```
# File:     utils.asm
# Purpose:  To define utilities which will be used in MIPS programs.
# Author:   Charles Kann
#
# Title to and ownership of all intellectual property rights
# in this file are the exclusive property of Charles W. Kann.
#
# Subprograms Index:
#   Exit           Call syscall with a server 10 to exit the program
#   PrintNewLine   Print a new line character (\n) to the console
#   PrintInt       Print a string with an integer to the console
#   PrintString    Print a string to the console
#   PromptInt      Prompt for an int & return it to the calling program.
#
# Modification History
#     12/27/2014 - Initial release

# Subprogram:     Exit
# Author:         Charles Kann
# Purpose:        to use syscall service 10 to exit a program
# Input/Output:   None
# Side effects:   The program is exited

.text
Exit:
    li $v0, 10
    syscall

# Subprogram:     PrintNewLine
# Author:         Charles Kann
# Purpose:        to output a new line to the user console
# Input/Output:   None
# Side effects:   A new line character is printed to the user's console
```

```
.text
PrintNewLine:
    li $v0, 4
    la $a0, __PNL_newline
    syscall
    jr $ra

.data
   __PNL_newline:   .asciiz "\n"


# Subprogram:     PrintInt
# Author:         Charles W. Kann
# Purpose:        To print a string to the console
# Input:          $a0 - The address of the string to print.
#                 $a1 - The value of the int to print
# Output:         None
# Side effects:   The String is printed followed by the integer value.

.text
PrintInt:
    # Print string.  The string address is already in $a0
    li $v0, 4
    syscall

    # Print integer.   The integer value is in $a1, and must
    # be first moved to $a0.
    move $a0, $a1
    li $v0, 1
    syscall

    #Return
    jr $ra

# Subprogram:     PrintString
# Author:         Charles W. Kann
# Purpose:        To print a string to the console
# Input:          $a0 - The address of the string to print.
# Output:         None
# Side effects:   The String is printed to the console.

.text
PrintString:
    addi $v0, $zero, 4
    syscall
    jr $ra

# Subprogram:     PromptInt
# Author:         Charles W. Kann
# Purpose:        To prompt the user for an integer, and
#                 to return that input value to the caller.
# Input:          $a0 - The address of the string to print.
# Output:         $v0 - The value the user entered
# Side effects:   The String is printed followed by the integer value.
```

```
.text
PromptInt:
    # Print the prompt, which is already in $a0
    li $v0, 4
    syscall

    # Read the integer.  Note: at the end of the syscall the value is
    # already in $v0, so there is no need to move it anywhere.
    move $a0, $a1
    li $v0, 5
    syscall

    #Return
    jr $ra
```

Program 5-5: File utils.asm

Chapter 5. 9 Final program to prompt, read, and print an integer

The following program is the culmination of this chapter. Make sure that the file utils.asm is in the same directory as the Program5-9.asm. Bring Program5-9.asm into the edit window in MARS, assemble it, and run it.

```
# File:     Program5-9.asm
# Author:   Charles Kann
# Purpose:  Illustrates implementing a subprogram named PrintNewLine.

.text
main:
    # Read an input value from the user
    la $a0, prompt
    jal PromptInt
    move $s0, $v0

    # Print the value back to the user
    jal PrintNewLine
    la $a0, result
    move $a1, $s0
    jal PrintInt

    # Call the Exit subprogram to exit
    jal Exit

.data
    prompt: .asciiz "Please enter an integer: "
    result: .asciiz "You entered: "

.include "utils.asm"
```

Program 5-6: Final program to prompt for, read, and print an integer

Note that by using subprogram abstraction how much easier this program is to understand than the one at the start of the chapter. It is beginning to look like the HLL code most readers are already familiar with.

Chapter 5. 10 Summary

This chapter covered the basics of subprogram calls, how to implement and call subprograms, and the basics of how to pass basic parameters to the subprogram. It also explained the `$pc` and `$ra` registers and their role in controlling the execution sequence of a program. How to properly comment subprograms and files was also covered.

Chapter 5. 11 Exercises

1. A colleague decided that the PrintInt subprogram should print a new line after the integer has been printed and has modified the PrintInt to print a new line character as follows.

```
# Subprogram:     PrintInt
# Author:         Charles W. Kann
# Purpose:        To print a string to the console
# Input:          $a0 - The address of the string to print.
#                 $a1 - The value of the int to print
# Output:         None
# Side effects:   The string is printed followed by the integer

.text
PrintInt:
    # Print string.  The string address is already in $a0
    li $v0, 4
    syscall

    # Print integer.   The integer value is in $a1, and must
    # be first moved to $a0.
    move $a0, $a1
    li $v0, 1
    syscall

    # Print a new line character
    jal PrintNewLine

    #Return
    jr $ra
```

When the program is run, it never ends and acts like it is stuck in an infinite loop. Help this colleague figure out what is wrong with the program.

   a. Explain what is happening in the program
   b. Come up with a mechanism which shows that this program is indeed in an infinite loop.
   c. Come up with a solution which fixes this problem (we want to keep PrintNewLine as a subprogram in PrintInt for this example).

2. Someone has modified the utils.asm file to insert a PrintTab subprogram immediately after the PrintNewLine subprogram as shown below (changes are highlighted in yellow). The programmer complains that the PrintTab command cannot be called using the "`jal PrintTab`" instruction. What is wrong and how can this be fixed? Explain all the problems in the code this programmer has written.

```
# Subprogram:     PrintNewLine
# Author:         Charles Kann
# Purpose:        to output a new line to the user console
# Input/Output:   None
# Side effects:   A new line char is printed to the console

.text
PrintNewLine:
    li $v0, 4
    la $a0, __PNL_newline
    syscall
    jr $ra

.data
   __PNL_newline:   .asciiz "\n"

PrintTab:
    li $v0, 4
    la $a0, tab
    syscall
    jr $ra

.data
    tab: .asciiz "\t"
```