Assembly Language and Computer Architecture Lab

Nguyen Thi Thanh Nga
Dept. Computer engineering
School of Information and Communication Technology

Week 5: Subprogram

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

How to create a subprogram

- Many operations were common to more than one of the programs.

- It is better to abstract the method of the program once, and then use this abstraction in every program.

- One way to abstract data is to use subprograms.

- Similar to subprograms exist in every language, and go by the names functions, procedures, or methods.

How to create a subprogram

- The concept of a subprogram to create a group of MIPS assembly language statements which can be called (or dispatched) to perform a task.

- These subprograms are called simple subprogram 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.

How to create a subprogram

- There can be multiple **.text** and **.data** alternating directive declarations
- Follow the naming convention
- Comment the required information before the subprogram.

```
1  # File: Program5-1.asm
2  # Author: Charles Kann
3  # Purpose: To illustrate implementing and calling a
4  # subprogram named Exit
5  .text
6  main:
7      # read an input value from the user
8      li $v0, 4
9      la $a0, prompt
10     syscall
11     li $v0, 5
12     syscall
13     move $s0, $v0
14
15     # print the value back to the user
16     li $v0, 4
17     la $a0, result
18     syscall
19     li $v0, 1
20     move $a0, $s0
21     syscall
22
23      # call the Exit subprogram to exit
24     jal Exit
25 .data
26      prompt: .asciiz "Please enter an integer: "
27      result: .asciiz "\nYou entered: "
28
29 # subprogram: Exit
30 # author: Charles Kann
31 # purpose: to use syscall service 10 to exit a program
32 # input: None
33 # output: None
34 # side effects: The program is exited
35 .text
36 Exit:
37     li $v0, 10
38     syscall
```

Program 5.1 Exit Subprogram

- The following program prompts a user for an integer, prints it out, and exits the program.
- This program has two **.text** sections. This is because the **Exit** subprogram is not a part of the main, and so exists in a different part of the file.
- There can be as many **.text** and **.data** statements in assembly language as are needed, and they can be interspersed as they are here.
- When in a **.text** segment, there should be only program text, in a **.data** segment, only data.

```
1  # File: Program5-1.asm
2  # Author: Charles Kann
3  # Purpose: To illustrate implementing and calling a
4  # subprogram named Exit
5  .text
6  main:
7      # read an input value from the user
8      li $v0, 4
9      la $a0, prompt
10     syscall
11     li $v0, 5
12     syscall
13     move $s0, $v0
14
15     # print the value back to the user
16     li $v0, 4
17     la $a0, result
18     syscall
19     li $v0, 1
20     move $a0, $s0
21     syscall
22
23      # call the Exit subprogram to exit
24     jal Exit
25 .data
26      prompt: .asciiz "Please enter an integer: "
27      result: .asciiz "\nYou entered: "
28
29 # subprogram: Exit
30 # author: Charles Kann
31 # purpose: to use syscall service 10 to exit a program
32 # input: None
33 # output: None
34 # side effects: The program is exited
35 .text
36 Exit:
37     li $v0, 10
38     syscall
```

Program 5.1 Exit Subprogram

- 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.

- The **jal** instruction transfers control of the computer to the address at the label **Exit**.

- The **Exit** subprogram halts the program so there is no return statement when it is complete.

```
1  # File: Program5-1.asm
2  # Author: Charles Kann
3  # Purpose: To illustrate implementing and calling a
4  # subprogram named Exit
5  .text
6  main:
7      # read an input value from the user
8      li $v0, 4
9      la $a0, prompt
10     syscall
11     li $v0, 5
12     syscall
13     move $s0, $v0
14
15     # print the value back to the user
16     li $v0, 4
17     la $a0, result
18     syscall
19     li $v0, 1
20     move $a0, $s0
21     syscall
22
23      # call the Exit subprogram to exit
24     jal Exit
25 .data
26      prompt: .asciiz "Please enter an integer: "
27      result: .asciiz "\nYou entered: "
28
29 # subprogram: Exit
30 # author: Charles Kann
31 # purpose: to use syscall service 10 to exit a program
32 # input: None
33 # output: None
34 # side effects: The program is exited
35 .text
36 Exit:
37     li $v0, 10
38     syscall
```

Program 5.1 Exit Subprogram

- Programmers should always attempt to be consistent with naming conventions.
- All subprograms should at a minimum contain the information provided in this example (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.

Program 5.2
PrintNewLine Subprogram

```
1  # File: Program5-2.asm
2  # Author: Charles Kann
3  # Purpose: To illustrate implementing and calling a
4  # subprogram named PrintNewLine.
5  .text
6  main:
7          # read an input value from the user
8          li $v0, 4
9          la $a0, prompt
10         syscall
11         li $v0, 5
12         syscall
13         move $s0, $v0
14
15         # print the value back to the user
16         jal PrintNewLine
17         li $v0, 4
18         la $a0, result
19         syscall
20         li $v0, 1
21         move $a0, $s0
22         syscall
23
24         # call the Exit subprogram to exit
25         jal Exit
26 .data
27         prompt: .asciiz "Please enter an integer: "
28         result: .asciiz "You entered: "
29
30 # subprogram: PrintNewLine
31 # author: Charles Kann
32 # purpose: to output a new line to the user console
33 # input: None
34 # output: None
35 # side effects: A new line character is printed to the
36 # user's console
37 .text
38 PrintNewLine:
39         li $v0, 4
40         la $a0, __PNL_newline
41         syscall
42         jr $ra
43 .data
44         __PNL_newline: .asciiz "\n"
45
46 # subprogram: Exit
47 # author: Charles Kann
48 # purpose: to use syscall service 10 to exit a program
49 # input: None
50 # output: None
51 # side effects: The program is exited
52 .text
53 Exit:
54         li $v0, 10
55         syscall
```

- This subprogram implements to print a new line character, allowing the program to stop inserting the control character "**\n**" into their programs.
- There are three **.text** segments.
- The newline character is required to be stored in the **.data** segment.

Program 5.2
PrintNewLine Subprogram

```
1  # File: Program5-2.asm
2  # Author: Charles Kann
3  # Purpose: To illustrate implementing and calling a
4  # subprogram named PrintNewLine.
5  .text
6  main:
7          # read an input value from the user
8          li $v0, 4
9          la $a0, prompt
10         syscall
11         li $v0, 5
12         syscall
13         move $s0, $v0
14
15         # print the value back to the user
16         jal PrintNewLine
17         li $v0, 4
18         la $a0, result
19         syscall
20         li $v0, 1
21         move $a0, $s0
22         syscall
23
24         # call the Exit subprogram to exit
25         jal Exit
26 .data
27       prompt: .asciiz "Please enter an integer: "
28       result: .asciiz "You entered: "
29
30 # subprogram: PrintNewLine
31 # author: Charles Kann
32 # purpose: to output a new line to the user console
33 # input: None
34 # output: None
35 # side effects: A new line character is printed to the
36 # user's console
37 .text
38 PrintNewLine:
39         li $v0, 4
40         la $a0, __PNL_newline
41         syscall
42         jr $ra
43 .data
44         __PNL_newline: .asciiz "\n"
45
46 # subprogram: Exit
47 # author: Charles Kann
48 # purpose: to use syscall service 10 to exit a program
49 # input: None
50 # output: None
51 # side effects: The program is exited
52 .text
53 Exit:
54         li $v0, 10
55         syscall
```

- The **.text** segments are needed to inform the assembler that program instructions are again contained in the code.

- **Rule**: 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** of **.data** segment when it should be present can lead to unnecessary problems.

Program 5.2
PrintNewLine Subprogram

```
1  # File: Program5-2.asm
2  # Author: Charles Kann
3  # Purpose: To illustrate implementing and calling a
4  # subprogram named PrintNewLine.
5  .text
6  main:
7          # read an input value from the user
8          li $v0, 4
9          la $a0, prompt
10         syscall
11         li $v0, 5
12         syscall
13         move $s0, $v0
14
15         # print the value back to the user
16         jal PrintNewLine
17         li $v0, 4
18         la $a0, result
19         syscall
20         li $v0, 1
21         move $a0, $s0
22         syscall
23
24         # call the Exit subprogram to exit
25         jal Exit
26 .data
27       prompt: .asciiz "Please enter an integer: "
28       result: .asciiz "You entered: "
29
30 # subprogram: PrintNewLine
31 # author: Charles Kann
32 # purpose: to output a new line to the user console
33 # input: None
34 # output: None
35 # side effects: A new line character is printed to the
36 # user's console
37 .text
38 PrintNewLine:
39         li $v0, 4
40         la $a0, __PNL_newline
41         syscall
42         jr $ra
43 .data
44         __PNL_newline: .asciiz "\n"
45
46 # subprogram: Exit
47 # author: Charles Kann
48 # purpose: to use syscall service 10 to exit a program
49 # input: None
50 # output: None
51 # side effects: The program is exited
52 .text
53 Exit:
54         li $v0, 10
55         syscall
```

- The PrintNewLine subprogram shows how to return from a subprogram using the instruction "**jr $ra**".
- A label was needed in the PrintNewLine subprogram to contain the address of the newline variable.
- In MIPS assembly, cannot use labels with the same name.
- Make sure that any label used will 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.

```asm
# File: Program5-2.asm
# Author: Charles Kann
# Purpose: To illustrate implementing and 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 PrintNewLine Subprogram

- The **$a0** and **$v0** registers have been changed, but this is not listed as a side effect of the program.
- The save registers (**$s0**...**$s9**) cannot be used because they must contain the same value when the subprogram returns.
- All other registers have no guarantee of the value after the subprogram 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.

The Program Counter ($pc) register

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

Text Segment

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	0x0c100010	jal 0x00400040	16: jal PrintNewLine
	0x00400020	0x24020004	addiu $2,$0,0x00000004	17: li $v0, 4

Labels

Label	Address
mips5-2.asm	
main	0x00400000
PrintNewLine	0x00400040
Exit	0x00400054
prompt	0x10010000
result	0x1001001a
__PNL_newline	0x10010028

Data Text

Data Segment

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

0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

Registers Coproc 1 Coproc 0

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

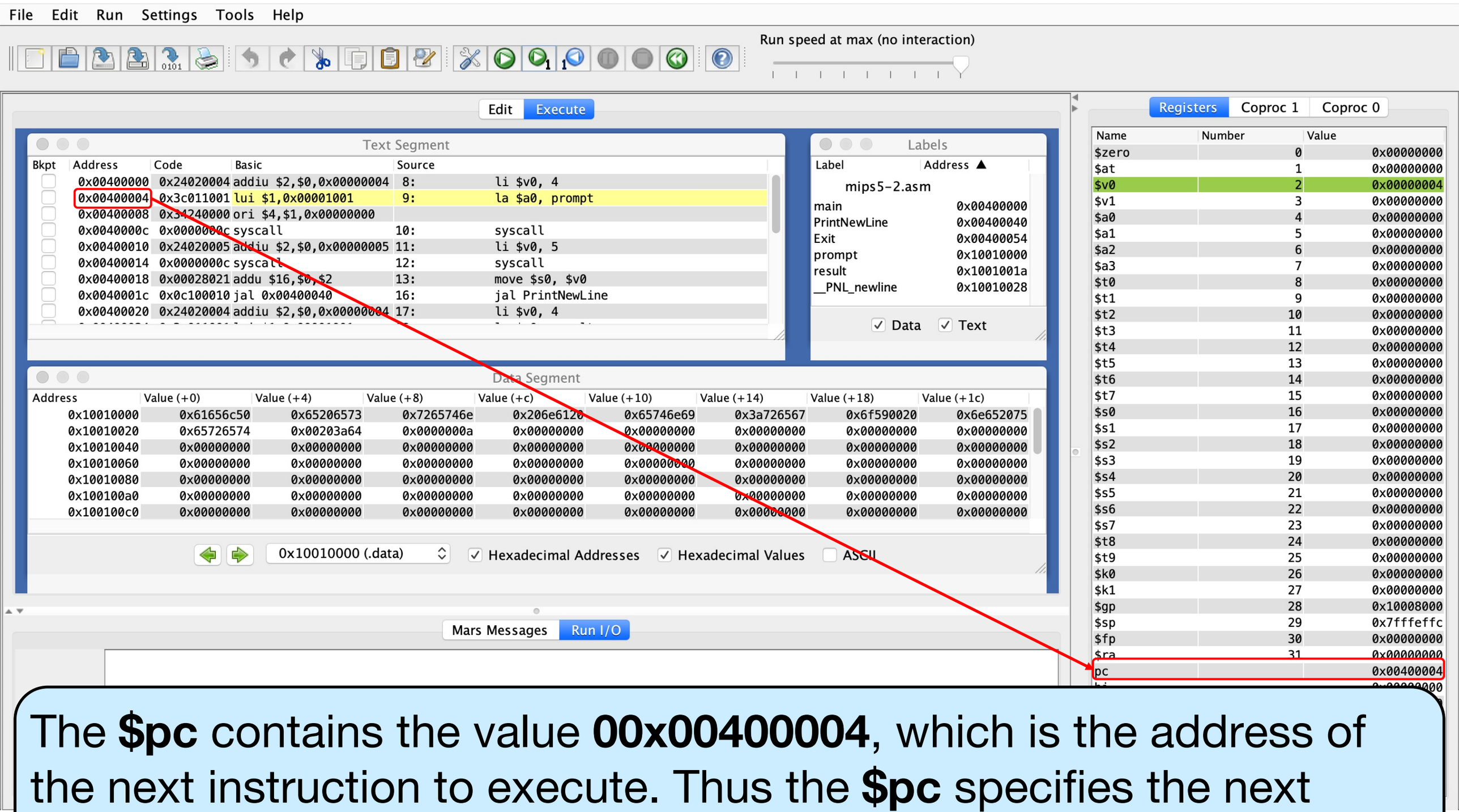
Mars Messages Run I/O

```
Assemble: assembling /Users/ngantt/Dropbox/Baigiang/IT3280/mips5-2.asm

Assemble: operation completed successfully.
```

The next instruction to execute is at **0x0040000**, and the $pc register contains the value **0x0040000**

The Program Counter ($pc) register



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 Program Counter ($pc) register

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit | Execute

Text Segment

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	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	39: li $v0, 4
	0x00400044	0x3c011001	lui $1,0x00001001	40: la $a0, __PNL_newline
	0x00400048	0x34240028	ori $4,$1,0x00000028	
	0x0040004c	0x0000000c	syscall	41: syscall
	0x00400050	0x03e00008	jr $31	42: jr $ra
	0x00400054	0x2402000a	addiu $2,$0,0x0000000a	54: li $v0, 10
	0x00400058	0x0000000c	syscall	55: syscall

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x00000006
$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	0x00000006
$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

Mars Messages | Run I/O

```
You entered: 5
-- program is finished running --
```

At **jal PrintNewLine**, the **$pc** points to **0x0040001c**, the next statement would be **0x00400020**. However, translation of the **jal PrintNewLine** has the address **0x00400040** in it which is the first line in the **PrintNewLine** subprogram.

The Program Counter ($pc) register

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

Text Segment

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	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	39: li $v0, 4
	0x00400044	0x3c011001	lui $1,0x00001001	40: la $a0, __PNL_newline
	0x00400048	0x34240028	ori $4,$1,0x00000028	
	0x0040004c	0x0000000c	syscall	41: syscall
	0x00400050	0x03e00008	jr $31	42: jr $ra
	0x00400054	0x2402000a	addiu $2,$0,0x0000000a	54: li $v0, 10
	0x00400058	0x0000000c	syscall	55: syscall

Registers Coproc 1 Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x00000006
$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	0x00000006
$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

Mars Messages Run I/O

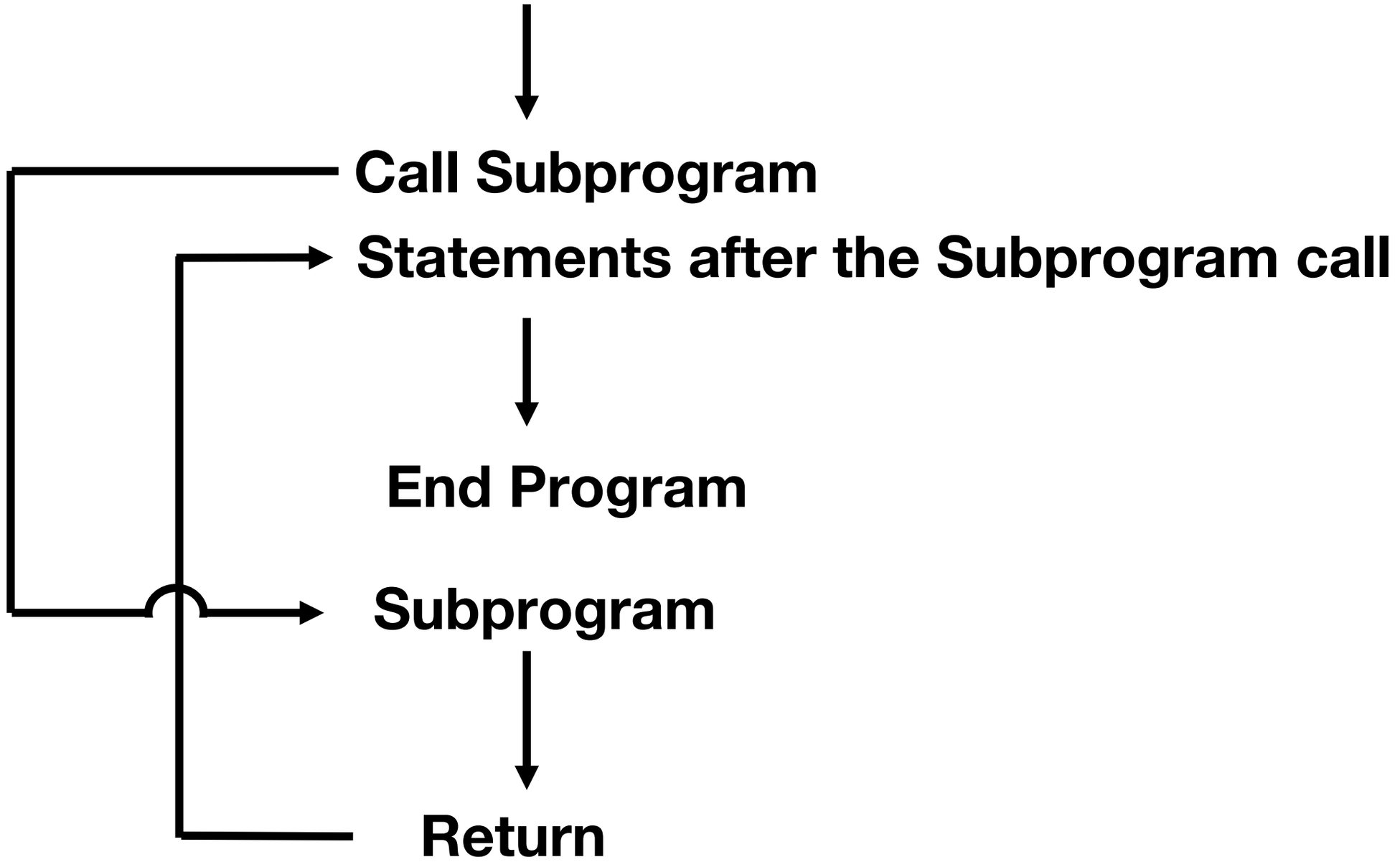
```
You entered: 5
-- program is finished running --

Please enter an integer: 6
```

Clear

When the **jal** instruction is executed, the **$pc** register is set to **0x00400040**, and the program continues by executing in the **PrintNewLine** subprogram.

Returning from a subprogram and the $ra register



When the subprogram is called, the next sequential address of an instruction must be stored. When the subprogram is completed, the control must branch back to that instruction.

Returning from a subprogram and the $ra register

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

Text Segment

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	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	39: li $v0, 4
	0x00400044	0x3c011001	lui $1,0x00001001	40: la $a0, __PNL_newline
	0x00400048	0x34240028	ori $4,$1,0x00000028	
	0x0040004c	0x0000000c	syscall	41: syscall
	0x00400050	0x03e00008	jr $31	42: jr $ra
	0x00400054	0x2402000a	addiu $2,$0,0x0000000a	54: li $v0, 10
	0x00400058	0x0000000c	syscall	55: syscall

Registers Coproc 1 Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x00000006
$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	0x00000006
$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

Mars Messages Run I/O

You entered: 5

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.

Returning from a subprogram and the $ra register

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

Text Segment

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	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	39: li $v0, 4
	0x00400044	0x3c011001	lui $1,0x00001001	40: la $a0, __PNL_newline
	0x00400048	0x34240028	ori $4,$1,0x00000028	
	0x0040004c	0x0000000c	syscall	41: syscall
	0x00400050	0x03e00008	jr $31	42: jr $ra
	0x00400054	0x2402000a	addiu $2,$0,0x0000000a	54: li $v0, 10
	0x00400058	0x0000000c	syscall	55: syscall

Registers Coproc 1 Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x00000004
$v1	3	0x00000000
$a0	4	0x10010028
$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	0x00000006
$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		0x00400050

Mars Messages Run I/O

-- program is finished running --

The **$ra** register stored the address **0x00400020**. It shows that the **jal** operator not only sets the **$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.

Returning from a subprogram and the $ra register

- When a subprogram is called, the **jal** operator updates the **$pc** register to point to the address of the label for the subprogram.

- The **jal** operator also sets the **$ra** register to the next sequential 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.

Program 5.3 PrintString subprogram How to input parameter

```
1  # File: Program5-3.asm
2  # Author: Charles Kann
3  # Purpose: To illustrate implementing and calling a
4  # subprogram named PrintNewLine
5  .text
6  main:
7      # read an input value from the user
8      la $a0, prompt
9      jal PrintString
10     li $v0, 5
11     syscall
12     move $s0, $v0
13     # print the value back to the user
14     jal PrintNewLine
15     la $a0, result
16     jal PrintString
17     li $v0, 1
18     move $a0, $s0
19     syscall
20     # call the Exit subprogram to exit
21     jal Exit
22 .data
23         prompt: .asciiz "Please enter an integer: "
24         result: .asciiz "You entered: "
25
26 # subprogram: PrintNewLine
27 # author: Charles Kann
28 # purpose: to output a new line to the user console
29 # input: None
30 # output: None
31 # side effects: A new line character is printed to the
32 # user's console
33 .text
34 PrintNewLine:
35         li $v0, 4
36         la $a0, __PNL_newline
37         syscall
38         jr $ra
39 .data
40         __PNL_newline: .asciiz "\n"
41
```

- The **PrintString** subprogram abstracts the ability to print a string.
- This subprogram requires **an input parameter** to be passed into the program, the address of the string to print.

```
42  # subprogram: PrintString
43  # author: Charles W. Kann
44  # purpose:  To print a string to the console
45  # input: $a0 - The address of the string to print.
46  # returns:  None
47  # side effects: The String is printed to the console.
48  .text
49  PrintString:
50          addi $v0, $zero, 4
51          syscall
52          jr $ra
53
54 # subprogram: Exit
55 # author: Charles Kann
56 # purpose: to use syscall service 10 to exit a program # input: None
57 # output: None
58 # side effects: The program is exited
59 .text
60 Exit:
61          li $v0, 10
62          syscall
```

Program 5.3 PrintString subprogram How to input parameter

```
1  # File: Program5-3.asm
2  # Author: Charles Kann
3  # Purpose: To illustrate implementing and calling a
4  # subprogram named PrintNewLine
5  .text
6  main:
7      # read an input value from the user
8      la $a0, prompt
9      jal PrintString
10     li $v0, 5
11     syscall
12     move $s0, $v0
13     # print the value back to the user
14     jal PrintNewLine
15     la $a0, result
16     jal PrintString
17     li $v0, 1
18     move $a0, $s0
19     syscall
20     # call the Exit subprogram to exit
21     jal Exit
22 .data
23         prompt: .asciiz "Please enter an integer: "
24         result: .asciiz "You entered: "
25
26 # subprogram: PrintNewLine
27 # author: Charles Kann
28 # purpose: to output a new line to the user console
29 # input: None
30 # output: None
31 # side effects: A new line character is printed to the
32 # user's console
33 .text
34 PrintNewLine:
35         li $v0, 4
36         la $a0, __PNL_newline
37         syscall
38         jr $ra
39 .data
40         __PNL_newline: .asciiz "\n"
41
```

- The registers **$a0**...**$a4** are used to pass input parameters into a program
- So the register **$a0** is used for the input parameter to this subprogram.
- The subprogram then loads a **4** into **$v0**, invoke **syscall**, and return.

```
42  # subprogram: PrintString
43  # author: Charles W. Kann
44  # purpose:  To print a string to the console
45  # input: $a0 - The address of the string to print.
46  # returns:  None
47  # side effects: The String is printed to the console.
48  .text
49  PrintString:
50        addi $v0, $zero, 4
51        syscall
52        jr $ra
53
54 # subprogram: Exit
55 # author: Charles Kann
56 # purpose: to use syscall service 10 to exit a program # input: None
57 # output: None
58 # side effects: The program is exited
59 .text
60 Exit:
61          li $v0, 10
62          syscall
```

Program 5.4 PrintInt subprogram How to input multiple parameters

```
1  # File: Program5-4.asm
2  # Author: Charles Kann
3  # Purpose: To illustrate implementing and calling a
4  # subprogram named PrintNewLine
5  .text
6  main:
7          # read an input value from the user
8          la $a0, prompt
9          jal PrintString
10         li $v0, 5
11         syscall
12         move $s0, $v0
13         # print the value back to the user
14         jal PrintNewLine
15         la $a0, result
16         move $a1, $s0
17         jal PrintInt
18         # call the Exit subprogram to exit
19         jal Exit
20 .data
21         prompt: .asciiz "Please enter an integer: "
22         result: .asciiz "You entered: "
23
24 # subprogram: PrintNewLine
25 # author: Charles Kann
26 # purpose: to output a new line to the user console
27 # input: None
28 # output: None
29 # side effects: A new line character is printed to the
30 # user's console
31 .text
32 PrintNewLine:
33         li $v0, 4
34         la $a0, __PNL_newline
35         syscall
36         jr $ra
37 .data
38         __PNL_newline:  .asciiz "\n"
```

- The **PrintInt** subprogram abstracts the ability to print a string first to tell the user what was being printed, and then the integer was printed.

- 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**.

```
40  # subprogram:     PrintInt
41  # author: Charles W. Kann
42  # purpose:  To print a string to the console
43  # input:    $a0 - The address of the string to print.
44  #           $a1 - The value of the int to print
45  # returns:  None
46  # side effects: The String is printed followed by the integer value
47 .text
48 PrintInt:
49         # Print string. The string address is already in $a0
50         li $v0, 4
51         syscall
52         # Print integer. The integer value is in $a1, and must
53         # be first moved to $a0.
54
55         li $v0, 1
56         syscall
57         #return
58         jr $ra
```

Program 5.5 PromptInt subprogram
How to return a value from a subprogram

```
1  # File: Program5-5.asm
2  # Author: Charles Kann
3  # Purpose: To illustrate implementing and calling a
4  # subprogram named PrintNewLine
5  .text
6  main:
7          # read an input value from the user
8          la $a0, prompt
9          jal PromptInt
10         move $s0, $v0
11         # print the value back to the user
12         jal PrintNewLine
13         la $a0, result
14         move $a1, $s0
15         jal PrintInt
16         # call the Exit subprogram to exit
17         jal Exit
18 .data
19         prompt: .asciiz "Please enter an integer: "
20         result: .asciiz "You entered: "
21
22 # subprogram: PrintNewLine
23 # author: Charles Kann
24 # purpose: to output a new line to the user console
25 # input: None
26 # output: None
27 # side effects: A new line character is printed to the
28 # user's console
29 .text
30 PrintNewLine:
31         li $v0, 4
32         la $a0, __PNL_newline
33         syscall
34         jr $ra
35 .data
36         __PNL_newline:  .asciiz "\n"
37
```

- The PromptInt subprogram print the user for an integer input, and to return that input value to the caller.
- The registers **$a0**...**$a3** are used to pass values into a subprogram
- The registers **$v0**...**$v1** are used to return values.

```
57
58 # subprogram:     PromptInt
59 # author: Charles W. Kann
60 # purpose: To print the user for an integer input, and
61 # to return that input value to the caller.
62 # input:        $a0 - The address of the string to print.
63 # returns:      $v0 - The value the user entered
64 # side effects: The String is printed followed by the integer value.
65 .text
66 PromptInt:
67         # Print the prompt, which is already in $a0
68         li $v0, 4
69         syscall
70         # Read the integer value. Note that at the end of the
71         # syscall the value is already in $v0, so there is no
72         # need to move it anywhere.
73
74         li $v0, 5
75         syscall
76         #return
77         jr $ra
```

Program 5.6
Create a utils.asm file

- The final step in adding the subprograms developed to put them 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 of the subprograms (including comments) into this file.
- The preamble comment (the comment at the start of the file) in the utility 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 of the file.
  - A copyright statement
  - A subprogram 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.

Program 5.6
Create a utils.asm file

```
# File:      utils.asm
# Purpose:   To define utilities which will be used in MIPS programs.
# Author:    Charles Kann
#
# Instructors are granted permission to make copies of this file
# for use by # students in their courses. Title to and ownership
# of all intellectual property rights
# in this file are the exclusive property of
# Charles W. Kann, Gettysburg, Pa.
#
# Subprograms Index:
#     Exit -       Call syscall with a server 10 to exit the program
#     NewLine -    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 the user to enter an integer, and return
#                 it to the calling program.
#
# Modification History
#     12/27/2014 - Initial release
```

Program 5.7 Final program to prompt, read, and print an integer

```
3   # Purpose: To illustrate implementing and calling a
4   # subprogram named PrintNewLine.
5
6   .text
7   main:
8           # read an input value from the user
9           la $a0, prompt
10          jal PromptInt
11          move $s0, $v0
12
13          # print the value back to the user
14          jal PrintNewLine
15          la $a0, result
16          move $a1, $s0
17          jal PrintInt
18
19          # call the Exit subprogram to exit
20          jal Exit
21
22  .data
23          prompt: .asciiz "Please enter an integer: "
24          result: .asciiz "You entered: "
25
26  .include "utils.asm"
```

Assigment 5.1

- 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
# 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

    # Print a new line character
    jal PrintNewLine

    #return
    jr $ra
```

Assigment 5.1

- 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.

  - Explain what is happening in the program

  - Come up with a mechanism which shows that this program is indeed in an infinite loop.

  - Come up with a solution which fixes this problem.

Assigment 5.2

- 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.

Assigment 5.2

```
# 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"

PrintTab:
     li $v0, 4
     la $a0, tab
     syscall
     jr $ra
.data
     tab: .asciiz "\t"
```

Assigment 5.3

1. There are many algorithms presented in this text that would lend themselves to be included as subprograms in the utils.asm file. Implement some or all of the following into the `utils.asm` file, properly documenting them, and include programs to test them.
    a. `NOR` subprogram - take two input parameters, and return the NOR operation on those two parameter.
    b. `NAND`- take two input parameters, and return the NAND operation on those two parameter.
    c. `NOT`- take one input parameters, and return the NOT operation on that parameter.
    d. `Mult4` - take an input parameter, and return that parameter multiplied by 4 using only shift and add operations.
    e. `Mult10` - take an input parameter, and return that parameter multiplied by 10 using only shift and add operations.
    f. `Swap`- take two input parameters, swap them using only the XOR operation.
    g. `RightCircularShift` - take an input parameter, and return two values. The first is the value that has been shifted 1 bit using a right circular shift, and the second is the value of the bit which has been shifted.
    h. `LeftCircularShift` - take an input parameter, and return two values. The first is the value that has been shifted 1 bit using a left circular shift, and the second is the value of the bit which has been shifted.
    i. `ToUpper` - take a 32 bit input which is 3 characters and a null, or a 3 character string. Convert the 3 characters to upper case if they are lower case, or do nothing if they are already upper case.
    j. `ToLower` - take a 32 bit input which is 3 characters and a null, or a 3 character string. Convert the 3 characters to lower case if they are upper case, or do nothing if they are already lower case.

End of week 5