TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN ĐIỆN TỬ-VIỄN THÔNG

BÁO CÁO THÍ NGHIỆM

KIẾN TRÚC MÁY TÍNH

Họ tên sinh viên: Nguyễn Minh Hiếu

Mã lớp TN: 688522

MSSV: 20151336

Lớp: Điện tử 3 k60

Hà Nội, 2019

Example 1:

```
# A demonstration of some simple MIPS instructions
# used to test QtSPIM

        # Declare main as a global function
        .globl main

        # All program code is placed after the
        # .text assembler directive
        .text

# The label 'main' represents the starting point
main:
        li $t2, 25              # Load immediate value (25)
        lw $t3, value           # Load the word stored in value (see bottom)
        add $t4, $t2, $t3       # Add
        sub $t5, $t2, $t3       # Subtract
        sw $t5, Z               #Store the answer in Z (declared at the bottom)

        # Exit the program by means of a syscall.
        # There are many syscalls - pick the desired one
        # by placing its code in $v0. The code for exit is "10"
        li $v0, 10 # Sets $v0 to "10" to select exit syscall
        syscall # Exit

        # All memory structures are placed after the
        # .data assembler directive
        .data
```

The .word assembler directive reserves space

in memory for a single 4-byte word (or multiple 4-byte words)

and assigns that memory location an initial value

(or a comma separated list of initial values)

value: .word 12

Z: .word 0

Result:

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000019
$t3	11	0x0000000c
$t4	12	0x00000025
$t5	13	0x0000000d
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	temporary (not preserved across call)	00000
$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		0x00400024
hi		0x00000000
lo		0x00000000

Example 2:

```
# "Hello World" in MIPS assembly
# From: http://labs.cs.upt.ro/labs/so2/html/resources/nachos-doc/mipsf.html

        # All program code is placed after the
        # .text assembler directive
        .text

        # Declare main as a global function
        .globl  main

# The label 'main' represents the starting point
main:
        # Run the print_string syscall which has code 4
        li      $v0,4           # Code for syscall: print_string
        la      $a0, msg        # Pointer to string (load the address of msg)
        syscall
        li      $v0,10          # Code for syscall: exit
        syscall

        # All memory structures are placed after the
        # .data assembler directive
        .data

        # The .asciiz assembler directive creates
        # an ASCII string in memory terminated by
        # the null character. Note that strings are
        # surrounded by double-quotes
msg:    .asciiz "Hello World!\n"
```

Output:

Hello World!

Example 3:

Simple input/output in MIIPS assembly

From: http://labs.cs.upt.ro/labs/so2/html/resources/nachos-doc/mipsf.html

```
        # Start .text segment (program code)
        .text

        .globl  main
main:
        # Print string msg1
        li      $v0,4           # print_string syscall code = 4
        la      $a0, msg1       # load the address of msg
        syscall

        # Get input A from user and save
        li      $v0,5           # read_int syscall code = 5
        syscall
        move    $t0,$v0         # syscall results returned in $v0

        # Print string msg2
        li      $v0,4           # print_string syscall code = 4
        la      $a0, msg2       # load the address of msg2
        syscall

        # Get input B from user and save
```

```
        li      $v0,5           # read_int syscall code = 5
        syscall
        move    $t1,$v0         # syscall results returned in $v0

        # Math!
        add     $t0, $t0, $t1   # A = A + B

        # Print string msg3
        li      $v0, 4
        la      $a0, msg3
        syscall

        # Print sum
        li      $v0,1           # print_int syscall code = 1
        move    $a0, $t0        # int to print must be loaded into $a0
        syscall

        # Print \n
        li      $v0,4           # print_string syscall code = 4
        la      $a0, newline
        syscall

        li      $v0,10          # exit
        syscall

        # Start .data segment (data!)
        .data
msg1:   .asciiz "Enter A:   "
msg2:   .asciiz "Enter B:   "
```

```
msg3:	.asciiz	"A + B = "
newline:	.asciiz	"\n"
```

Output:

Enter A: 2

Enter B: 3

A + B = 5

Example 4:

```
# Simple routine to demo a loop
# Compute the sum of N integers: 1 + 2 + 3 + ... + N
# From: http://labs.cs.upt.ro/labs/so2/html/resources/nachos-doc/mipsf.html

	.text

	.globl	main
main:
	# Print msg1
	li	$v0,4		# print_string syscall code = 4
	la	$a0, msg1
	syscall

	# Get N from user and save
	li	$v0,5		# read_int syscall code = 5
	syscall
	move	$t0,$v0		# syscall results returned in $v0

	# Initialize registers
```

```
        li      $t1, 0          # initialize counter (i)
        li      $t2, 0          # initialize sum

        # Main loop body
loop:   addi    $t1, $t1, 1     # i = i + 1
        add     $t2, $t2, $t1   # sum = sum + i
        beq     $t0, $t1, exit  # if i = N, continue
        j       loop

        # Exit routine - print msg2
exit:   li      $v0, 4          # print_string syscall code = 4
        la      $a0, msg2
        syscall

        # Print sum
        li      $v0,1           # print_string syscall code = 4
        move    $a0, $t2
        syscall

        # Print newline
        li      $v0,4           # print_string syscall code = 4
        la      $a0, lf
        syscall
        li      $v0,10          # exit
        syscall

        # Start .data segment (data!)
        .data
msg1:   .asciiz "Number of integers (N)?  "
```

```
msg2:   .asciiz   "Sum = "
lf:     .asciiz   "\n"
```

Output:

Number of integers (N)? 5

Sum = 15

Example 5 with stack

```
# Simple routine to demo functions
# USING a stack in this example to preserve
# values of calling function

# ---------------------------------------------------------------

        .text

        .globl  main
main:
        # Register assignments
        # $s0 = x
        # $s1 = y

        # Initialize registers
        lw      $s0, x          # Reg $s0 = x
        lw      $s1, y          # Reg $s1 = y

        # Call function
        move    $a0, $s0        # Argument 1: x ($s0)
```

```
	jal	fun		# Save current PC in $ra, and jump to fun
	move	$s1,$v0		# Return value saved in $v0. This is y ($s1)

	# Print msg1
	li	$v0, 4		# print_string syscall code = 4
	la	$a0, msg1
	syscall

	# Print result (y)
	li	$v0,1		# print_int syscall code = 1
	move	$a0, $s1	# Load integer to print in $a0
	syscall

	# Print newline
	li	$v0,4		# print_string syscall code = 4
	la	$a0, lf
	syscall

	# Exit
	li	$v0,10		# exit
	syscall

# --------------------------------------------------------------

	# FUNCTION: int fun(int a)
	# Arguments are stored in $a0
	# Return value is stored in $v0
	# Return address is stored in $ra (put there by jal instruction)
	# Typical function operation is:
```

```
fun:    # This function overwrites $s0 and $s1
        # We should save those on the stack
        # This is PUSH'ing onto the stack
        addi $sp,$sp,-4         # Adjust stack pointer
        sw $s0,0($sp)           # Save $s0
        addi $sp,$sp,-4         # Adjust stack pointer
        sw $s1,0($sp)           # Save $s1

        # Do the function math
        li $s0, 3
        mul $s1,$s0,$a0                 # s1 = 3*$a0  (i.e. 3*a)
        addi $s1,$s1,5          # 3*a+5

        # Save the return value in $v0
        move $v0,$s1

        # Restore saved register values from stack in opposite order
        # This is POP'ing from the stack
        lw $s1,0($sp)           # Restore $s1
        addi $sp,$sp,4          # Adjust stack pointer
        lw $s0,0($sp)           # Restore $s0
        addi $sp,$sp,4          # Adjust stack pointer

        # Return from function
        jr $ra                  # Jump to addr stored in $ra

# ----------------------------------------------------------------
```

```
        # Start .data segment (data!)
        .data
x:      .word 5
y:      .word 0
msg1:   .asciiz  "y="
lf:   .asciiz     "\n"
```

Output:

y=20

Example 5 without stack

```
# Simple routine to demo functions
# NOT using a stack in this example.
# Thus, the function does not preserve values
# of calling function!

# ---------------------------------------------------------------

        .text

        .globl   main
main:
        # Register assignments
        # $s0 = x
        # $s1 = y
```

```
	# Initialize registers
	lw	$s0, x		# Reg $s0 = x
	lw	$s1, y		# Reg $s1 = y

	# Call function
	move	$a0, $s0	# Argument 1: x ($s0)
	jal	fun		# Save current PC in $ra, and jump to fun
	move	$s1,$v0		# Return value saved in $v0. This is y ($s1)

	# Print msg1
	li	$v0, 4		# print_string syscall code = 4
	la	$a0, msg1
	syscall

	# Print result (y)
	li	$v0,1		# print_int syscall code = 1
	move	$a0, $s1	# Load integer to print in $a0
	syscall

	# Print newline
	li	$v0,4		# print_string syscall code = 4
	la	$a0, lf
	syscall

	# Exit
	li	$v0,10		# exit
	syscall

# ------------------------------------------------------------------
```

```
        # FUNCTION: int fun(int a)
        # Arguments are stored in $a0
        # Return value is stored in $v0
        # Return address is stored in $ra (put there by jal instruction)
        # Typical function operation is:

fun:    # Do the function math
        li $s0, 3
        mul $s1,$s0,$a0             # s1 = 3*$a0  (i.e. 3*a)
        addi $s1,$s1,5          # 3*a+5

        # Save the return value in $v0
        move $v0,$s1

        # Return from function
        jr $ra                  # Jump to addr stored in $ra

# ----------------------------------------------------------------

        # Start .data segment (data!)
        .data
x:      .word 5
y:      .word 0
msg1:   .asciiz   "y="
lf:   .asciiz     "\n"
```

Output:

y=20

Example 6:

```
# Simple routine to demo a loop

# Compute the sum of N integers: 1 + 2 + 3 + ... + N

# Same result as example4, but here a function performs the

# addition operation:  int add(int num1, int num2)


# -----------------------------------------------------------------


        .text


        .globl  main

main:

        # Register assignments

        # $s0 = N

        # $s1 = counter (i)

        # $s2 = sum


        # Print msg1

        li      $v0,4           # print_string syscall code = 4

        la      $a0, msg1

        syscall


        # Get N from user and save

        li      $v0,5           # read_int syscall code = 5

        syscall

        move    $s0,$v0         # syscall results returned in $v0


        # Initialize registers
```

```
        li      $s1, 0          # Reg $s1 = counter (i)
        li      $s2, 0          # Reg $s2 = sum

        # Main loop body
loop:   addi    $s1, $s1, 1     # i = i + 1

        # Call add function
        move    $a0, $s2        # Argument 1: sum ($s2)
        move    $a1, $s1        # Argument 2: i ($s1)
        jal     add2            # Save current PC in $ra, and jump to add2
        move    $s2,$v0         # Return value saved in $v0. This is sum ($s2)
        beq     $s0, $s1, exit  # if i = N, continue
        j       loop

        # Exit routine - print msg2
exit:   li      $v0, 4          # print_string syscall code = 4
        la      $a0, msg2
        syscall

        # Print sum
        li      $v0,1           # print_string syscall code = 4
        move    $a0, $s2
        syscall

        # Print newline
        li      $v0,4           # print_string syscall code = 4
        la      $a0, lf
        syscall
        li      $v0,10          # exit
```

```
        syscall

# -----------------------------------------------------------------


        # FUNCTION: int add(int num1, int num2)
        # Arguments are stored in $a0 and $a1
        # Return value is stored in $v0
        # Return address is stored in $ra (put there by jal instruction)
        # Typical function operation is:
        #  1.) Store registers on the stack that we will overwrite
        #  2.) Run the function
        #  3.) Save the return value
        #  4.) Restore registers from the stack
        #  5.) Return (jump) to previous location
        # Note: This function is longer than it needs to be,
        # in order to demonstrate the usual 5 step function process...

add2:   # Store registers on the stack that we will overwrite (just $s0)
        addi $sp,$sp, -4 # Adjust stack pointer
        sw $s0,0($sp)           # Save $s0 on the stack

        # Run the function
        add $s0,$a0,$a1                 # Sum = sum + i

        # Save the return value in $v0
        move $v0,$s0

        # Restore overwritten registers from the stack
        lw $s0,0($sp)
```

```
        addi $sp,$sp,4          # Adjust stack pointer

        # Return from function
        jr $ra                  # Jump to addr stored in $ra

# --------------------------------------------------------------

        # Start .data segment (data!)
        .data
msg1:   .asciiz   "Number of integers (N)?  "
msg2:   .asciiz   "Sum = "
lf:     .asciiz   "\n"
```

Output:

Number of integers (N)? 10

Sum = 55