CS 224 – Spring 2019 – Lab #1

Creating and Running Simple MIPS Assembly Language Programs

Preliminary Design Report

Berrak Taşkınsu / 21602054

Section No.: 2

CS 224
Section No.: 2
Spring 2019
Berrak Taşkınsu / 21602054

Part 1. Preliminary Work / Preliminary Design Report

1. Write a MIPS program that
Creates an array of maximum size of 20 elements that asks the user first the number of elements and then enters the elements one by one.
Displays array contents
Reverses the array contents and display the array (for example 1, 2, 3 becomes 3, 2, 1).

```
           .text
# Intitialize array
     la $t1, array
     lw $t2, index
     lw $t4, arraySize
reject:
# Check array size
     li $v0, 4
     la $a0, str5
     syscall
     li $v0, 5
     syscall
     move $t3, $v0
# If number of elements does not exceed array size,
     ble $t3, $t4, accept
# Else, error message
     li $v0, 4
     la $a0, error
     syscall
     b reject
accept:
     move $t5, $t3
next:
# Input and store elements one by one
     li $v0, 4
     la $a0, str1
     syscall
     addi $t2, $t2, 1
     li $v0, 1
     move $a0, $t2
     syscall
     li $v0,4
     la $a0, str2
     syscall
     li $v0, 5
     syscall
     sw $v0, 0($t1)
     addi $t1, $t1, 4
     addi $t5, $t5, -1
     bgt $t5, $zero, next
# Print the array
     la $t1, array # $t1 points to first element of array
```

```
      li $v0, 4
      la $a0, str3
      syscall
next2:
      li $v0, 4
      la $a0, str4
      syscall
      li $v0, 1
      lw $a0, 0($t1)
      syscall
      addi $t1, $t1, 4
      addi $t2, $t2, -1
      bgt $t2, $zero, next2
      move $s0, $t1
      la $t1, array
      addi $t1, $t1, -4
next3:
# Reverse the contents of the array
      addi $t1, $t1, 4
      addi $s0, $s0, -4
      lw $s1, 0($t1)
      lw $s2, 0($s0)
      sw $s1, 0($s0)
      sw $s2, 0($t1)
      sub $t0, $t1, $s0
      bgt $t0, 8, next3
# Print the reversed array
      la $t1, array
      li $v0, 4
      la $a0, str6
      syscall
next4:
      li $v0, 4
      la $a0, str4
      syscall
      li $v0, 1
      lw $a0, 0($t1)
      syscall
      addi $t1, $t1, 4
      addi $t3, $t3, -1
      bgt $t3, $zero, next4

      .data
array:     .space  80
arraySize: .word   20
index:     .word   0
str1:      .asciiz "Element "
str2:      .asciiz ": "
str3:      .asciiz "Array:"
str4:      .asciiz " "
str5:      .asciiz "Number of elements: "
error:     .asciiz "Number of elements exceeds array size.\n"
str6:      .asciiz "\nReversed array: "
```

CS 224
Section No.: 2
Spring 2019
Berrak Taşkınsu / 21602054

2. Write a MIPS program that
Gets a input string and checks if it is a palindrome. (Study load byte, store byte and some other instructions if need). Study the necessary syscall to read a string

```
      .text
      li $v0, 8
      la $a0, buffer
      li $a1, 20
      syscall
      move $t1,$a0
      # Count is stored in $t0
      # String is stored in $t1
next:
      lb $t3, 0($t1)
      beqz $t3, end
      addi $t1, $t1, 1
      addi $t0, $t0, 1
      b next
end:
      la $t1, buffer
      addi $t1, $t1, -1
      add $t2, $t1, $t0
next2:
      addi $t1, $t1, 1
      addi $t2, $t2, -1
      sub $t5, $t1, $t2
      lb $t3, 0($t1)
      lb $t4, 0($t2)
      bne $t3, $t4, notPalin
      bgt $t5, 2, next2
      li $v0, 4
      la $a0, buffer
      syscall
      la $a0, palindrome
      syscall
      b end2
notPalin:
      li $v0, 4
      la $a0, buffer
      syscall
      la $a0, notPalindrome
      syscall
end2:

      .data
buffer:         .space 20
palindrome:     .asciiz " is palindrome!"
notPalindrome:  .asciiz " is not palindrome."
line:           .asciiz "\n"
```

CS 224
Section No.: 2
Spring 2019
Berrak Taşkınsu / 21602054

3. Write a MIPS program that
Implements the following expressions without using div. If necessary use instructions other that we saw in the class.

```
	.text
	li $v0, 4
	la $a0, cstr
	syscall
	li $v0, 5
	syscall
	sw $v0, c
	li $v0, 4
	la $a0, dstr
	syscall
	li $v0, 5
	syscall
	sw $v0, d

	# Calculation
	lw $t1, c
	lw $t2, d
	sub $t0, $t1, $t2
	rem $t0, $t0, 16
	bge $t0, $zero, pos
	addi $t0, $t0, 16
pos:
	sw $t0, x
	li $v0, 4
	la $a0, str1
	syscall
	li $v0, 1
	move $a0, $t1
	syscall
	li $v0, 4
	la $a0, str2
	syscall
	li $v0, 1
	move $a0, $t2
	syscall
	li $v0, 4
	la $a0, str3
	syscall
	li $v0, 1
	move $a0, $t0
	syscall

	.data
x:	.space 4
c:	.space 4
d:	.space 4
cstr: .asciiz "c value: "
dstr: .asciiz "d value: "
str1: .asciiz "x = ( "
str2: .asciiz " - "
str3: .asciiz " ) % 16 = "
```

CS 224
Section No.: 2
Spring 2019
Berrak Taşkınsu / 21602054

4. Generate the object code (in hex) for the following la and lw instructions. Show your work for the intermediate steps (both la and lw -in this form- are pseudo instructions and are implemented by two instructions).

```
(1)   la      $t1, a
(2)   la      $t2, b
(3)   lw      $t2, b
      lw      $t2, b
      .....
      .data
      .space  20
a:    .word   1, 2, 3
b:    .word   1
```

Suppose that ".data" stands for the address 0x10010000
Then, since the first 20 bytes are allocated , the initial addresses of a and b are 0x10010014 and 0x10010020, respectively.

Both la and lw are pseudo instructions and the given instructions are implemented by two instructions, lui and ori, and lui and lw, respectively, as follows:

Both lui, ori and lw are I-type instructions and their opcodes are f(hex), d(hex) and 23(hex), respectively, in hexadecimal. Thus, their opcodes in binary are 001111, 001101 and 100011, respectively.

(1)
pseudo instruction: la $t1, a implementation: lui $at, 0x1001
ori $t1, 0x0014($at)

lui $at, 0x1001: 001111 | 00000 | 00001 | 0001000000000001
The object code in hexadecimal form is 0x3C011001

ori $t1, 0x0014($at): 001101 | 00001 | 01001 | 0000000000010100
The object code in hexadecimal form is 0x342A0014

(2)
pseudo instruction: la $t2, b implementation: lui $at, 0x1001
ori $t2, 0x0020($at)

lui $at, 0x1001: 001111 | 00000 | 00001 | 0001000000000001
The object code in hexadecimal form is 0x3C011001

ori $t1, 0x0020($at): 001101 | 00001 | 01010 | 0000000000100000
The object code in hexadecimal form is 0x342A0020

(3)
pseudo instruction: lw $t2, b implementation: lui $at, 0x1001
lw $t2, 0x0020($at)

lui $at, 0x1001: 001111 | 00000 | 00001 | 0001000000000001
The object code in hexadecimal form is 0x3C011001

lw $t2, 0x0020($at): 100011 | 00001 | 01010 | 0000000000100000
The object code in hexadecimal form is 0x8C2A0020

CS 224
Section No.: 2
Spring 2019
Berrak Taşkınsu / 21602054

5. Define the following terms and provide an answer etc as described.
a. Symbolic machine instruction: give two examples
b. Machine instruction: give two examples and write their symbolic equivalents
c. Assembler directive: give two examples.
d. Pseudo instruction: give two examples and provide its implementation using real instructions.
Note in your answer only use symbolic machine instructions.

a- A symbolic machine intruction is a human-readable version of a machine instruction.

ex: addi $t1, $t1, 4
ex: sub $t0, $t2, $t1

b- A machine instruction is an instruction in machine code, written for a specific machine to read and execute.

ex: 0x3C091001
The corresponding symbolic machine instruction is "lui $t1, 0x1001".
ex: 0x21080017
The correcponding symbolic machine instruction is "addi $t0, $t0, 23".

c- An assembler directive is a message to the assembler which contains information required for the assembler to carry out the assembling process.

ex: .space 4
This assembler directive tells the assembler to allocate 4 bytes of space in the current data segment.

ex: .asciiz "Hello"
This assembler directive tells the assembler to store the string "Hello" in memory and to null-terminate it.

d- A pseudo instruction is a set of machine instructions (it corresponds to multiple machine instructions), allowing the user to do more complicated tasks in less number of lines.

ex:

pseudo instruction:	move $a0, $t3
implementation:	addi $a0, $t3, 0

ex:

pseudo instruction:	rem $t1, $t2, $t3
implementation:	div $t2, $t3
	mfhi $t1