Partners: Jean Herbst - solution

Laboratory 2
More on MIPS

Computer Science 240

MIPS Review

Debugging

Exercise 1 – 1: Download the **lab2-1.asm** file from the Lab Google Group. It contains a program that should output:

> **The code is: 8**
> **Now it is: 9**

If you have not selected the checkbox to **Show Line Numbers in the Edit window**, you will want to select that option to assist you in debugging.

Try to assemble, and examine the error messages listed.

Once you have corrected the syntax errors using the error messages, and can assemble the program, try to run it.

2. Describe and explain the run-time errors that occur.
Runtime exception: fetch address not aligned on word boundary 0x00000001

This error occurred because of the instruction 'lw $t0,1"

You cannot load a word from the address 1 because it is not on a word boundary. This is also an illegal address because it is in the reserved area of memory, which the programmer is not allowed to use.

The instruction should be "li $t0,1"

3. Modify the program to get the expected output. Add comments to the corrected program. Copy the code from MARS and paste it here:

```
# lab2-1.asm
# Debugging and Review Exercise
        .text
        .globl main

main:   li      $v0,4           # print the first string
        la      $a0,prompt1
        syscall

        li      $v0,1           # print the 8 (from memory)
        lb      $a0,val
        syscall

        move    $s0,$a0         # save the value of 8 in $s0

        li      $t0,1
        add     $s0,$t0,$s0     # add 1 to 8 to get 9

        li      $v0,4           # print the second string
        la      $a0,prompt2
        syscall

        li      $v0,1           # print the 9
        move    $a0,$s0
        syscall

        li      $v0,10
        syscall

        .data
prompt1:        .asciiz "The code is:  "  # had to change .ascii to .asciiz so strings didn't run together

prompt2:        .asciiz  "\nNow it is: "

val:            .byte    08
```

Storage allocation and program execution

Exercise 2 - 1. Predict the address and data contents of the following data segment from lecture .

```
        .data
str:    .byte       1,2,3,4
        .half       5,6,7,8
        .word       9,10,11,12
        .space      5
        .word       9,10,11,12
letters: .asciiz    "ABCD"
        .ascii      "ABCD"
        .byte       -1
```

Address	Label	Data			
0x1001003C		00	00	FF	44
0x10010038		43	42	41	00
0x10010034	letters	44	43	42	41/5A
0x10010030		00	00	00	0C
0x1001002C		00	00	00	0B
0x10010028		00	00	00	0A
0x10010024		00	00	00	09
0x10010020		00	00	00	00
0x1001001C		00	00	00	00
0x10010018		00	00	00	0C
0x10010014		00	00	00	0B
0x10010010		00	00	00	0A
0x1001000C		00	00	00	09
0x10010008		00	08	00	07
0x10010004		00	06	00	05
0x10010000	str	04	03	02	01

- Use **little-endian** byte order
- Show **one word (4 bytes)** per row.
- Lowest address (0x10010000) should be at **bottom** of stack.
- Label the addresses corresponding to **str** and **letters**
- Use hexadecimal notation

2. Predict how the values of **$a0** and **$t0** change as each instruction is executed, and answer the stated questions:

		$t0	**$a0**
		0x00000000	**0x00000000**
main:	*li $v0,11*		
	li $t0,2	**0x00000002**	
	lb $a0,letters($t0)		**0x00000043**
	syscall		

What is the result of the syscall? Prints a "C"

	addi $a0,$a0,-1	**0x00000042**
	syscall	

What is the result of the syscall? Prints a "B"

	addi $t0,$t0,1	**0x00000003**
	lb $a0,letters($t0)	**0x00000044**
	syscall	

What is the result of the syscall? Prints a "D"

	li $t0,'Z'	**0x0000005A**
	sb $t0,letters	
	lb $a0,letters	**0x0000005A**
	syscall	

What is the result of the syscall? Prints a "Z"

Update the memory diagram above to show what happens after the sb $t0,letters is executed.
The byte 41 becomes 5A at address represented by *letters*

3. Download **lab2-2.asm** from the Lab Google Group and single-step the program to verify your results. Examine the **Data Segment** to check that your storage allocation diagram is correct.

Exercise 3: Write a MIPS program which does the same thing as the following Java statements.

```
//initialize only these two strings in memory
String phrase = "Change: inevitable";
String addon = " except from vending machines";

//should output the string 'Change: inevitable'
System.out.println(phrase);

//should output 'Change: inevitable except from vending machines' with a single syscall
phrase = phrase.concat(addon);
System.out.println(phrase);

//should output 'Charge!'
phrase = phrase.replace(':', '!');
phrase = phrase.substring(0,7)
phrase = phrase.replace('n', 'r');
System.out.println(phrase);
```

Copy the code from MARS and paste it here:

```
# Written by:  Jean Herbst
# CS 240 lab 2  exercise 3 this program does some character replacements to change how the strings are printed
        .text
        .globl main

main:  # print the initial  phrase
                li $v0,4
                la $a0,phrase
                syscall

        # print a line feed character
                li $v0,11
                li $a0,'\n'
                syscall

        # replace the null byte at the end of the first phrase (which is offset 18 from the beginning of the string)
        # with a space so the two strings prompt and addon will become one longer string
                la $s0,phrase
                li $t0,' '
                sb $t0,18($s0)

        # print the concatenated phrase + addon
                li $v0,4
                la $a0,phrase
                syscall

        # print a line feed character
                li $v0,11
                li $a0,'\n'
                syscall
```

```
                li $t0,'r'
                sb $t0,3($s0) #replace the 'n' with an 'r' in 'Change' to make it 'Charge'

                li $t0,'!'
                sb $t0,6($s0) #replace the ':' with a "!"

                li $t0,0
                sb $t0,7($s0) #put a null after 'Charge!' to terminate the  string

        # print 'Charge!'
                li $v0,4
                la $a0,phrase
                syscall

                #halt program
                li $v0,10                                   #sys code for exit must be in $v0
                syscall

                .data
# initialize only these two strings in memory
phrase: .asciiz  "Change: inevitable"
addon: .asciiz  "except from vending machines"
```

Numeric representation

Exercise 4: On paper, perform addition on the following binary and hexadecimal numbers (assume **two's complement** format!). Indicate whether there is a carry-out or an overflow for each addition.

Hex	Binary	Hex	Binary
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

1. For the first 2 calculations, assume **16- bit representation.** Do the calculation using the binary values.

Then, convert the numbers for the operands and result to hexadecimal notation (to convert, divide the digits into groups of 4, and translate each group to the corresponding hexadecimal value).

	Carry-Out?	Overflow?	Hexadecimal Value
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1			0xFFFF
+1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1			0xFFFF
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0	yes	no	0xFFFE

	Carry-Out?	Overflow?	Hexadecimal Value
0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0			0xFFFF
+0 1 1 1 1 1 1 1 1 0 0 1 1 0 0 1			0xFFFF
	yes	no	0xFFFE

2. Now, assume **32-bit representation**. The numbers are given are in hexadecimal notion.

	Carry-Out?	Overflow?
0x A A F F 9 0 1 4		
+0x A A E 3 C D 1 2		
1 5 5 E 3 5 D 2 6	yes	yes

	Carry-Out?	Overflow?
0x 7 F A A 3 2 7 8		
+0x 6 0 2 4 C D 1 2		
D F C E F F 8 A	no	yes

3. Use MARS to write a very simple program that adds the contents of $t0 and $t1, and puts the result in $t2:

```
        .text
        .globl main
main:   add $t2,$t0,$t1         # add contents of $t0 and $t1, and store result in $t2

        li $v0,10               # terminate execution
        syscall
```

After assembling the program (but **before** executing), enter the values for $t0 and $t1 directly into the **Registers** panel. Use the 32-bit hexadecimal values from the previous exercise:

```
    0x A A F F 9 0 1 4
+   0x A A E 3 C D 1 2
```

Execute the program. Describe what happens:

There is a run-time error because of arithmetic overflow displayed in the console, and the rightmost pane in MARS displays the status register showing that the overflow bit is set.