LAST NAME: *FIRST NAME :*

Take home TEST 1 CSC 342

Due date: by 23:59 PM on March 19, , 2017

Please post to me on slack completed TEST by11:59 PM on 3/19/2017

Please circle major: Computer Science or Engineering

You should allocate no more than 6 hours for this TEST.
Please write:
Start Time and date:
End TIME and date:

You can use any resources.
You are not allowed to communicate with fellow students or other professionals.

THIS IS NOT A TEAM PROJECT!!

Objective

The objective of this TEST is to demonstrate your understanding and ability to compare MIPS instruction set architecture, Intel x86 using Windows 32-bit compiler and debugger, and a Intel X86_64 bit processor running Linux, 64 bit gcc and gdb.

- **In order to compare the instruction set architectures, You will be testing and analyzing programs described in Sections 2.1, 2.2, 2.3,2.4, 2.5, 2.6, 2.7, 2.8 of the textbook on all 3 platforms.**
- For MIPS, YOU will be running the programs on the MARS MIPS simulator.
- For the Windows 32-bit platform, You will be running the programs on Visual Studio and using its debugger.
- For the Linux 64-bit platform, You will be running the programs using GCC and debugging it using GDB.

YOU must explain how local, static variables are handled in each case. Explain little endian, and big endian where appropriate.

For each example, you have to display disassembly, registers, memory and explain.
There are 3 parts in this test. You may use examples shown below, or you can create your own examples.
YOU MUST COVER SECTIONS 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8 from the textbook.

LAST NAME: *FIRST NAME :*

Take home TEST 1 CSC 342

Due date: by 23:59 PM on March 19, , 2017

In your comparison you should describe how local and static variables are handled on each platform.

It is important to demonstrate all uses of static variable as described below:

Static (C++)

http://msdn.microsoft.com/en-us/library/s1sb61xd.aspx
Visual Studio 2013

25 out of 27 rated this helpful - Rate this topic

The **static** keyword can be used to declare variables, functions, class data members and class functions.
By default, an object or variable that is defined outside all blocks has STATIC DURATION and EXTERNAL LINKAGE.

- **Static duration** *means that the object or variable is allocated when the program starts and is deallocated when the program ends*.
- **External linkage** *means that the name of the variable is visible from outside the file in which the variable is declared. Conversely, internal linkage means that the name is not visible outside the file in which the variable is declared*.

The **static** keyword can be used in the following situations.

- When you declare a variable or function at file scope (global and/or namespace scope), the **static** keyword specifies that the variable or function has internal linkage. When you declare a variable, the variable has static duration and the compiler initializes it to 0 unless you specify another value.
- When you declare a variable in a function, the **static** keyword specifies that the variable retains its state between calls to that function.
- When you declare a data member in a class declaration, the **static** keyword specifies that one copy of the member is shared by all instances of the class. A static data member must be defined at file scope. An integral data member that you declare as **const static** can have an initializer.
- When you declare a member function in a class declaration, the **static** keyword specifies that the function is shared by all instances of the class. A static member function cannot access an instance member because the function does not have an implicit **this** pointer. To access an instance member, declare the function with a parameter that is an instance pointer or reference.
- You cannot declare the members of a union as static. However, a globally declared anonymous union must be explicitly declared **static**.

LAST NAME: *FIRST NAME :*

Take home TEST 1 CSC 342

Due date: by 23:59 PM on March 19, , 2017

PART I

Examples in MIPS you may use

In order to run these programs on the MARS simulator, you have to write the programs all in MIPS assembly and then run the programs.

The first example is in Section 2.2 and is a simple program that is equivalent to running a = b+c and d = a-e in C.

2-2_1.asm

```
.data
a: .word 1
b: .word 2
c: .word 3
d: .word 4
e: .word 5
.text
lw $s0, a
lw $s1, b
lw $s2, c
lw $s3, d
lw $s4, e
# a = b + c
add $s0, $s1, $s2
sw $s0, a
# d = a - e
sub $s3, $s0, $s4
sw $s3, d
```

2-2_2.asm

```
.data
f: .word 0
g: .word 50
h: .word 40
i: .word 30
j: .word 20
.text
lw $s0, f
lw $s1, g
lw $s2, h
lw $s3, i
lw $s4, j
# t0 = g+h
```

LAST NAME: *FIRST NAME :*

Take home TEST 1 CSC 342

Due date: by 23:59 PM on March 19, , 2017

```
add $t0, $s1, $s2
# t1 = i+j
add $t1, $s3, $s4
# f = t0 - t1
sub $s0, $t0, $t1
sw $s0, f
```

LAST NAME: *FIRST NAME :*

Take home TEST 1 CSC 342

Due date: by 23:59 PM on March 19, , 2017

2-3_1.asm

```
.data
g: .word 0
h: .word 22
A: .word 0-100
size: .word 100
.text
#just to set A[8] to 55
li $t1, 55
la $s3, A
sw $t1, 32($s3)
lw $s1, g
lw $s2, h
#loading the value of A[8] into t0
lw $t0, 32($s3)
add $s1, $s2, $t0
sw $s1, g
```

2-3_2.asm

```
.data
h: .word 25
A: .word 0-100
size: .word 100
.text
lw $s2, h
#initializing A[8] to 200
li $t1, 200
la $s3, A
sw $t1, 32($s3)
#A[12] = h + A[8]
lw $t0, 32($s3)
add $t0, $s2, $t0
sw $t0, 48($s3)
```

2-5_2.asm

```
.data
h: .word 20
A: .word 0-400
size: .word 400
.text
la $t1, A
lw $s2, h
```

LAST NAME: *FIRST NAME :*

Take home TEST 1 CSC 342

Due date: by 23:59 PM on March 19, , 2017

```
#initialzing A[300] to 13
li $t2, 13
sw $t2, 1200($t1)
lw $t0, 1200($t1) add
$t0, $s2, $t0
sw $t0, 1200($t1)
```

This code essentially performs the C equivalent of A[300] = h + A[300].

2.6

The next example is in section 2.6 and covers several logical operations that can be done in MIPS. You can use the code

2-6_1.asm

```
# left shift
li $s0, 9
sll $t2, $s0, 4
# AND
li $t2, 0xdc0
li $t1, 0x3c00
and $t0, $t1, $t2
# OR
or $t0, $t1, $t2
# NOR li
$t3, 0
nor $t0, $t1, $t3
```

LAST NAME: *FIRST NAME :*

Take home TEST 1 CSC 342

Due date: by 23:59 PM on March 19, , 2017

This code performs the logical operations sll, AND, OR, and NOR. You have to explain each while executing the program.

For section 2.8
Please write a simple "MAIN" in MIPS assembly that calls "myadd" function and analyze.

PART II.

Examples in x86 Intel on Windows 32-bit

In order to properly run these examples on a Windows 32-bit OS and debug them with Visual Studio's Debugger, You can write all these examples covered in MIPS in C. The first example is once again the first example in Section 2.2. Recall that this just does a = b+c and d = a-e. Here is the code You can write and use

2-2_1.c 1

```
void main(){
          static int a = 1;
          static int b = 2;
          static int c = 3;
          static int d = 4;
          static int e = 5;
          a = b + c;
          d = a - e;
          }
```

2-2_2.c 1

```
void main(){
          static int f = 0;
          static int g = 50;
          static int h = 40;
          static int i = 30;
          static int j = 20; 7
                  f = (g + h) - (i
          + j);
  }
```

LAST NAME: *FIRST NAME :*

Take home TEST 1 CSC 342

Due date: by 23:59 PM on March 19, , 2017

2-3_1.c 1

```
void main(){
          static int g = 0;
          static int h = 22;
          static int A[100];
          A[8] = 55;
          g = h + A[8];
          }
```

2-3_2.c 1

```
void main(){
          static int h = 25;
          static int A[100];
          A[8] = 200;
          A[12] = h + A[8];
          }
```

2-5_1.c 1

```
void main(){
          static int a = 0;
          static int b = 0;
          static int c = 0;
          a = b + c;
          }
```

2-6_1.c 1

```
void main(){
          static int s0 = 9;
          static int t1 = 0x3c00;
          static int t2 = 0xdc0;
          static int t3 = 0;
t3 = s0 << 4;
static int t0 = 0;
t0 = t1 & t2;
t0 = t1 | t2;
t0 = ~t1;
}
```

LAST NAME: *FIRST NAME :*

Take home TEST CSC 342

March 21-28 , 2016

2-6_1.asm

```
    # left shift
    li $s0, 9
    sll $t2, $s0, 4
    # AND
    li $t2, 0xdc0
    li $t1, 0x3c00
and $t0, $t1, $t2
# OR
or $t0, $t1, $t2
# NOR li
$t3, 0
nor $t0, $t1, $t3
```

This code performs the logical operations sll, AND, OR, and NOR. I will explain each while executing the program.

2.8 Write and analyze in Debug mode example for “MAIN” calls simple “myadd” function.

PART III.

Repeat all the above for LINUX, gcc, gdb 64 bit.