**Take Home Test 1 CSC 342**

Danish Faruqi

April 5, 2021

Start Time and Date: April 2, 2021 1:53pm

End Time and Date: asa

***“I will neither give nor receive unauthorizes assistance on this TEST. I will use only one computing device to perform this TEST. I will not use cell while performing this TEST” - Danish Faruqi***

**Table of Contents**

1. **[Title Page](#Title_Page)**
2. [**Table of Contents**](#Table_of_Contents)
3. [**Objective**](#Objective)
4. [**Part 1 - MIPS**](#MIPS)
   1. [2-2\_1.asm](#MIPS_2_2_1)
   2. [2-2\_2.asm](#MIPS_2_2_2)
   3. [2-3\_1.asm](#MIPS_2_3_1)
   4. [2-3\_2.asm](#MIPS_2_3_2)
   5. [2-5\_2.asm](#MIPS_2_5_2)
   6. 2-6\_1.asm
   7. 2-7\_1.asm
   8. 2-7\_2.asm
   9. 2-8\_1.asm
   10. natural\_generator.asm
   11. main\_myadd.asm
5. **Part 2 - x86 Intel on Windows 32-bit**
   1. 2-2\_1.c
   2. 2-2\_2.c
   3. 2-3\_1.c
   4. 2-3\_2.c
   5. 2-5\_1.c
   6. 2-6\_1.c
   7. 2-7\_1.c
   8. 2-7\_2.c
   9. 2-8\_1.c
   10. natural\_generator.c
   11. main\_myadd.c
6. **Part 3 – LINUX, gcc, gdb 64bit on Intel x86-64 ISA**
   1. 2-2\_1.c
   2. 2-2\_2.c
   3. 2-3\_1.c
   4. 2-3\_2.c
   5. 2-5\_1.c
   6. 2-6\_1.c
   7. 2-7\_1.c
   8. 2-7\_2.c
   9. 2-8\_1.c
   10. natural\_generator.c
   11. main\_myadd.c
7. **Conclusion**

**Objective**

The objective of this test is to demonstrate the knowledge and compare MIPS, Windows 32bit Intel x86 ISA, and Linux 64bit Intel x86-x64 ISA architectures. Examples from Chapter 2 will help show the different concepts of Mips and C language. Other architectures and features will be present that are compared and highlighted throughout this test.

**Part 1 – MIPS**

**2-2\_1.asm**

2-2\_1.asm focuses on showing the relation of registers on basic level. **Figure 1** shows the code written out where a is being assigned the value b + c and d is assigned the value a – e. Each static variable is assigned a register and the values are present in the corresponding register. In the end the registers are updated again to hold the new values and the values in memory are also updated since they are all static variables.

**Figure 2** and **Figure 3** show that for each variable a through e have a corresponding register $s0 - $s4 respectively. Before each register can be assigned a value, the address of each variable is loaded from memory to be stored in the $at register, from which the value is retrieved from memory and stored into the correct register.

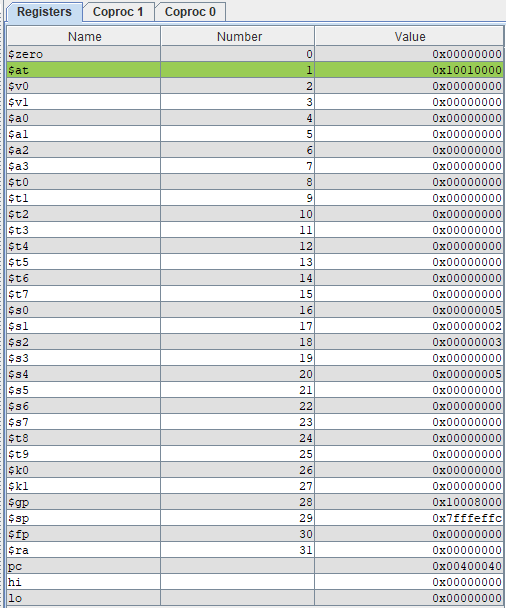
****For example, the address of static variable a, which is 0x10010000 is first stored in the register $at. Since a is the first value stored in memory location 0x10010000 at offset 0 (+0), the value is retrieved from memory and pushed onto register $s0. The value of a is 0x00000001 in big endian. This is done for the rest of the variables, all pulled from memory since they are static variables in big endian. To perform the operation a = b + c, register $s1 is added to $s2 and stored in $s0. The word is then stored back into memory at offset 0 (+0) in big endian as the value 0x00000005. To perform the operation d = a - e, register $s0 is subtracted from $s4 and stored in $s3. The word is then stored back into memory at offset c (+c) in big endian as the value 0x00000000.

Figure 2 - Faruqi\_2-2\_1 Registers

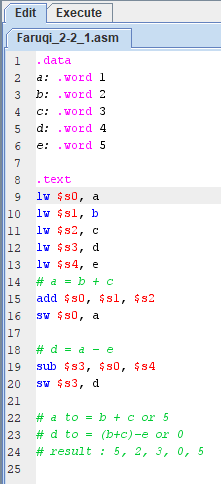


Figure - Faruqi\_2-2\_1 Code

$at stores memory address

$s0 stores value of a

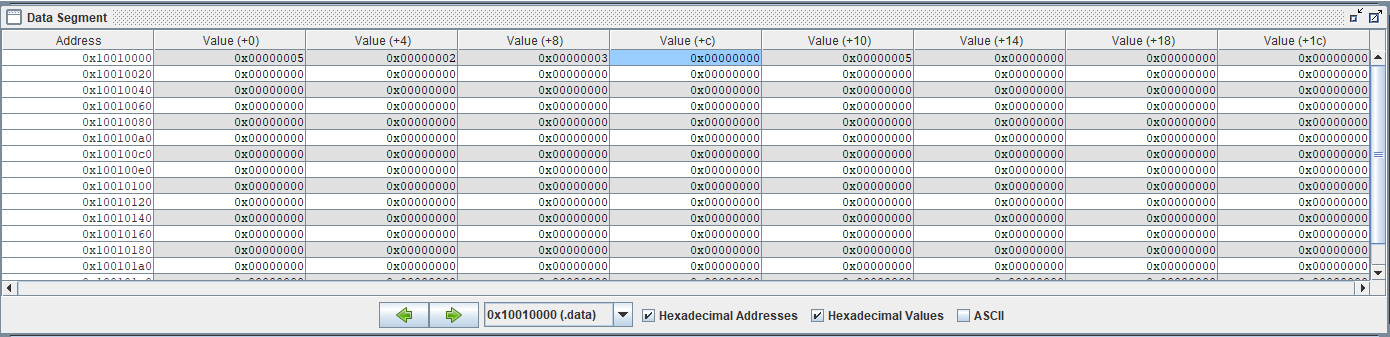
****Variable values are stored in memory and reflect all changes.

Figure 2 - Faruqi\_2-2\_1 MIPS Register

Figure - Faruqi\_2-2\_1 Code

Figure 3 - Faruqi\_2-2\_1 Data Segment

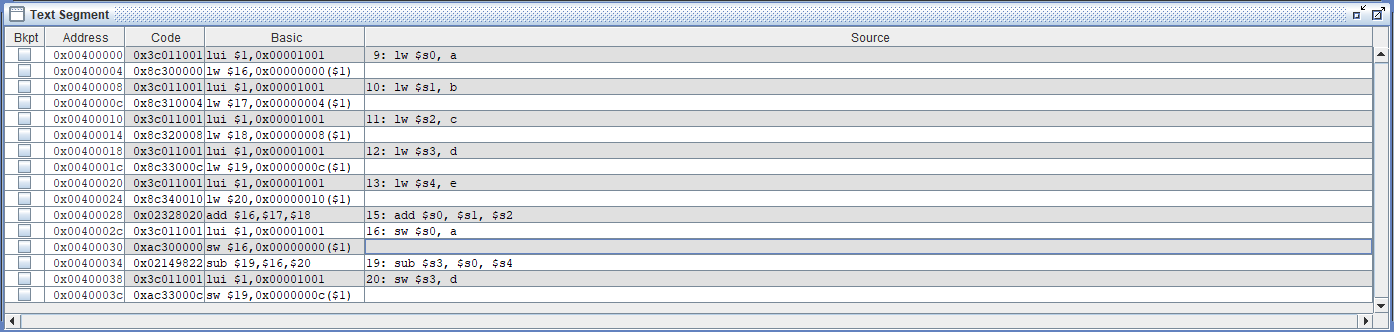
****

Figure 4 - Faruqi\_2-2\_1 Instructions

**2-2\_2.asm**

2-2\_2.asm also focuses on showing the relation of registers on basic level. **Figure 5** shows the code written out where f is being assigned the value (g + h) – (I + j). Each static variable is assigned a register and the values are present in the corresponding register. In the end the registers are updated again to hold the new values and the values in memory are also updated since they are all static variables. There is also a use of two other registers to store the temporary values of (g + h) and (I + j).

**Figure 6** and **Figure 8** show that for each variable f through j have a corresponding register $s0 - $s4 respectively. And to store the temporary values of (g + h) and (I + j) registers $t0 and $t1 are used respectively. Before each register can be assigned a value, the address of each value is loaded from memory to be stored in the $at register, from which the value is retrieved from memory and stored into the correct register as big endian.

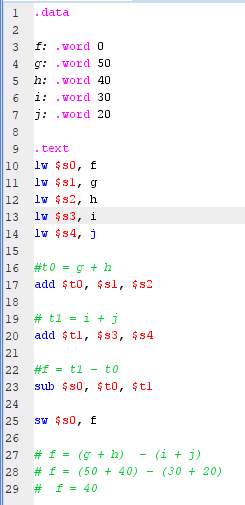
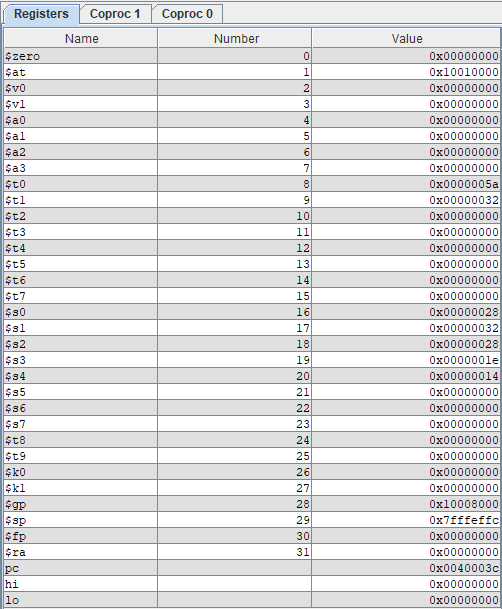
For example, to get the value of g, the address of g is stored into register $at, which is 0x10010000. Then the word is loaded into $s1 by adding 4 (+4) to the memory address since it is the second value. That value of g, 0x00000032 is then stored in $s1. After this is done for every value, the value of registers $s1 + $s2 is stored in a new register called $t0. Register $t1 stores the value of $s3 + $s4. From here, register $s0 stores the value of $t0 - $t1 which is 0x00000028. And finally, the value of $s0 is stored back in memory since it is a static variable in big endian. It’s stored at the address 0x10010000 by adding 0 (+ 0) since f is the first address on memory.

Figure 6 - Faruqi\_2-2\_2.asm Register

Figure 5 - Faruqi\_2-2\_2.asm Code

$s0 is initially 0 but then stores f = (g + h) - (i + j) or f = 0x00000028

$t0 stores g + h and $t1 stores i + j

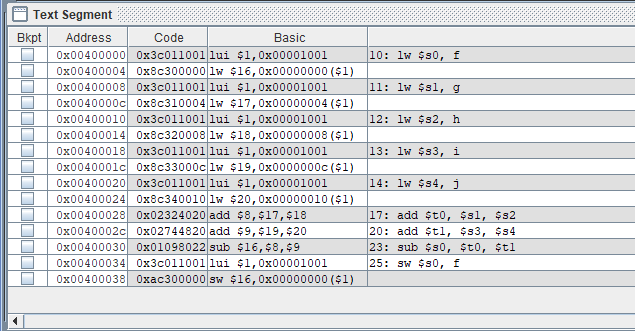
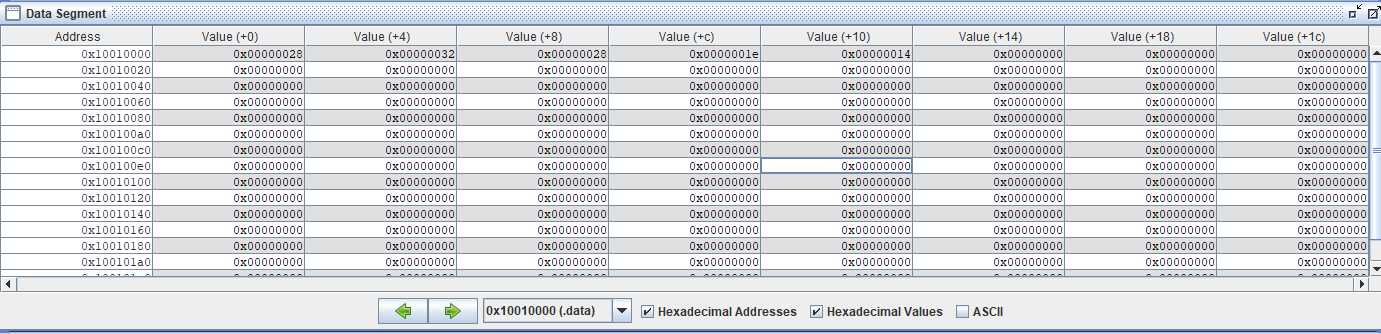


Figure 7 - Faruqi\_2-2\_2.asm Instructions



F is stored at + 0 and so when its value is changes, the value is changed in memory as well.

No other value changes on value since no other value was changed.

Figure 8 - Faruqi\_2-2\_2.asm Data Segment

**2-3\_1.asm**

2-3\_1.asm shows memory operands along with normal operands. It focuses on showing the relationship of memory with operands and how to access memory. **Figure 12** shows the code written out where static variables g and h are declared along with the static array A. A[8] is set to a value and then used to compute g = h + A[8].

At first, the register $t1 stores 0x00000037 or 55 in decimal as shown in **Figure 10**. Then array A’s address is loaded int register $at and pushed into register $s3, which is 0x10010008 in big endian. Then the value of $t1, which is 0x00000037 is stored in memory at location 0x10010008 + 32. 0x10010008 is the address of A[0] is memory, to get to A[8], 8 is multiplied by 4 to get to the correct address. From here the value of g is loaded from memory into register $s1 since g is a static variable in big endian. Next, in the same way, h’s value of 0x00000016 is loaded from memory since it’s a static variable in big endian in $s2. Now 0x00000037 is loaded from A[8] into $t0, again to access it from memory you need to access 0x10010008 + 32. $s1 stores the addition of 0x00000016 + 0x00000037, which is $s2 + $t0. In the end since g is a static variable it’s new value needs to be saved into memory. At memory address 0x10010000 + 0 the value 0x0000004d is stored. In the end we have

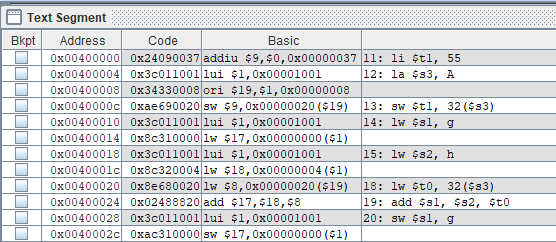
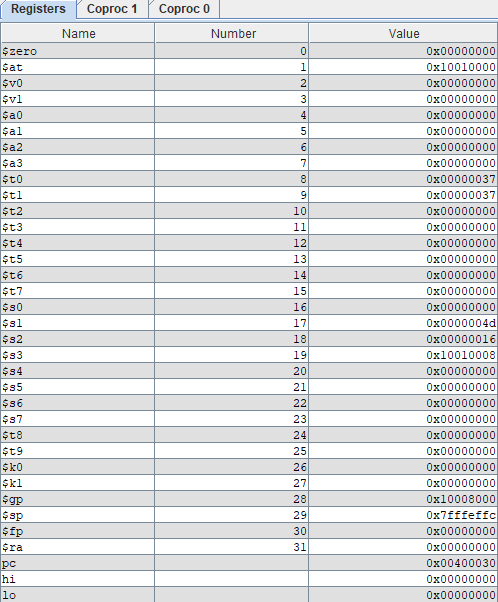
g = h + A[8] 🡺 g = 0x00000016 + 0x00000037 🡺 g = 0x0000004d or g = 77

Figure 9 - Faruqi\_2-3\_1 Instructions

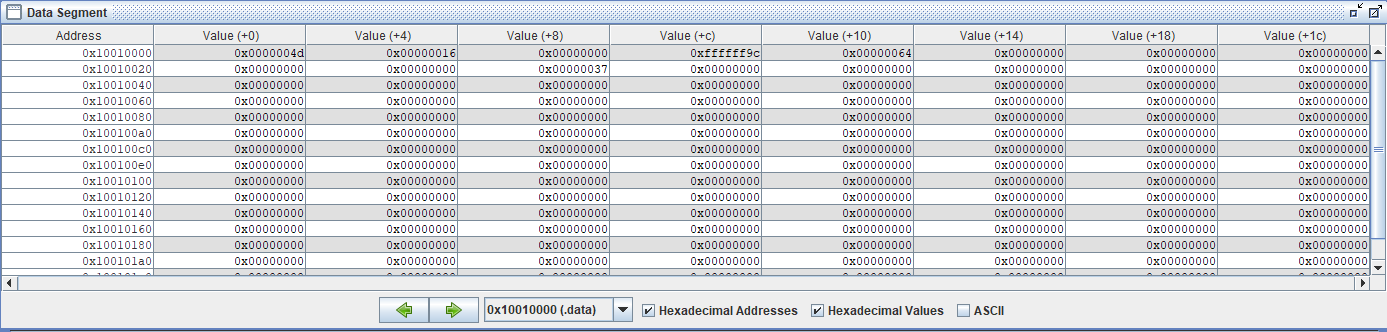


The Array start at 0x10010008 but A[8] is at 010010028

$s1 stores the value of g

$s3 stores the address of the array

Figure 10 - Faruqi\_2-3\_1 Register



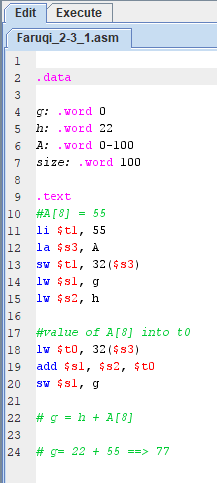
Figure 11 - Faruqi\_2-3\_1 Data Segment

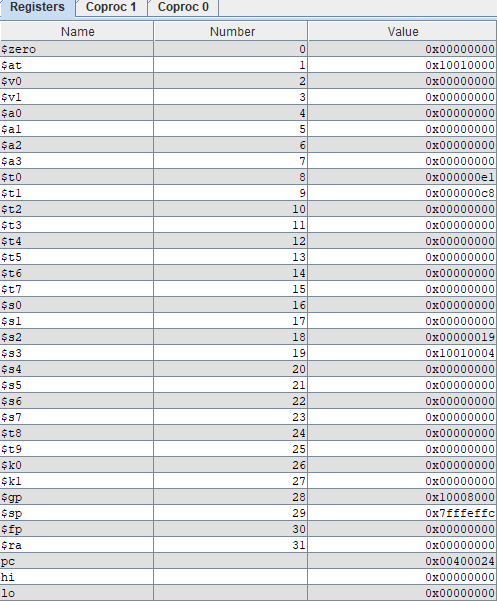
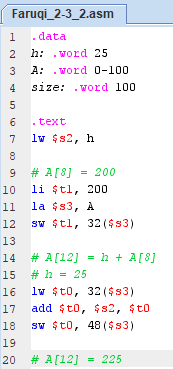
Figure 12 - Faruqi\_2-3\_1 Code

**2-3\_2.asm**

2-3\_2.asm also shows memory operands along with normal operands. It focuses on showing the relationship of memory with operands and how to access memory. **Figure 13** shows the code written out where static variables h = 25 is declared along with the static array A. A[8] is set to 200 a value and then used to compute A[12] = h + A[8].

This works the same way as the previous example worked as shown in **Figure 14**. Starting with h, the value of h is retrieved from memory at address 0x10010000 + 0 since it’s the first value stored at that address. It’s a static variable so it’s value is stored in memory in little endian. The value of h is 0x00000019 or 25 in decimal. Similar to before 200 or 0x000000c8 is stored in the register $t1. Address of A is loaded into $s3, which is 0x10010004 and will be sued from here on to edit the values in memory. First 200 which is stored at $t1 is loaded into memory address 0x10010004 + 0x00000020 which is A[8]. 8\*4 = 32, so we need to add 32 or 0x00000020 to base pointer to get to the 8th index. This is all stores in memory since the array is a static variable and being MIPS its in big endian. 200 or 0x000000c8 is then loaded from address 0x10010004 + 0x00000020 or A[8] into $t0. $t0 then stores the addition of itself and $s2 which is 25 + 200 = 225 or 0x00000019 + 0x000000c8 = 0x000000e1. Now the only thing left is to store this new value into A[12] which is 0x10010004 + 0x00000030 or + 32 from base pointer. The new value of A[12] is 0x000000e1.

A[12] = h + A[8] 🡺 A[12] = 25 + 200 = 225 🡺 A[12] = 0x00000019 + 0x000000c8 = 0x000000e1



Value of h

Value of A[12]

Value of A[8]

Figure 14 - Faruqi\_2-3\_2 Register

Figure 13 - Faruqi\_2-3\_2 Code

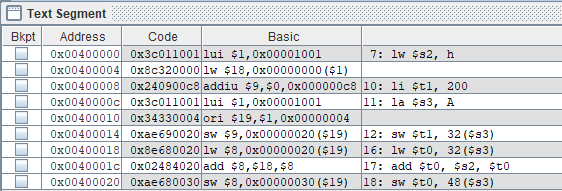


Figure 15 - Faruqi\_2-3\_2 Instructions

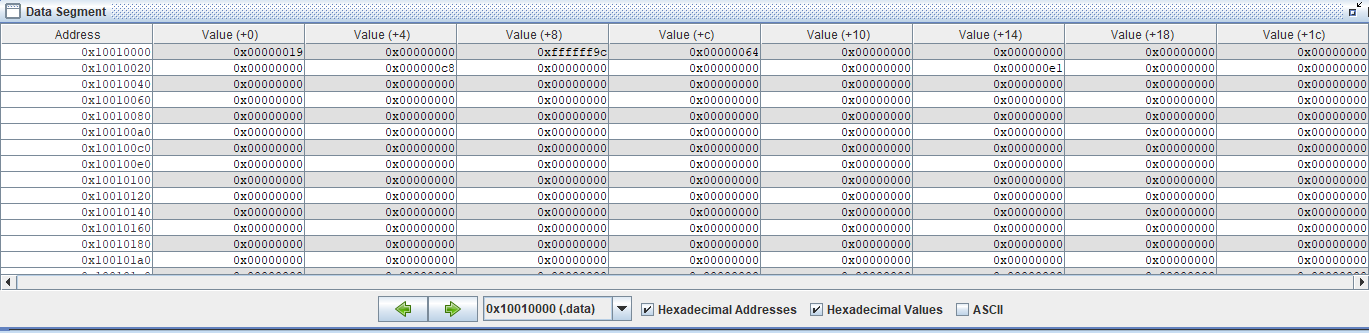


Figure 16 - Faruqi\_2-3\_2 Data Segment

The final value of A[12] gets store base pointer + 48 away which is A[0] + 12 indices

**2-5\_2.asm**