**TPS (Think-Pair-Share) activity 1 Paired with the classmate sitting next to you and do the following tasks (50 minutes):**

**1. Download “MIPS\_Reference\_Sheet” from CatCourses. We will need to refer to this sheet to complete all the exercises in this lab.**

**2. Load proc1.s in MARS and study the code. This is similar to compare.s from Lab #6.**

**3. After assembling the program, study the Text Segment window and see how your source code is translated into True Assembly Language (Basic) as well as machine code (Code).**

**4. In true assembly language, every single instruction can be translated into a machine instruction. How many bits does a machine instruction contain?**

A Machine instruction contains 32 bits or 4 bytes. For example, our first machine code reads as 0x20100019 which translates to

0010(0x2) 0000(0) 0001(1) 0000(0) 0000(0) 0000(0) 0001(1) 1001(9)

**5. To utilize the limited number of bits efficiently, all machine instructions are categorized into different types (or formats). How many types are there? What are they? Give 2 operations for each type as examples.**

There are three formats:

J - Format which is used for **j**  and **jal**:

**j 0x340990D0** # Jumps to that address

**jal 0x340990D0** # Jumps and stores the address in $ra

I - Format used for instructions with immediates like **lw** and **sw** or branches**:**

**lw $v0, 8($sp)**  #This is used to get a value from the stack utilizing immediate 8

**beq $t0, $zero, 0x340990D0**  **#**When $t0 equals to zero it will branch to an address

R - Format used for all the other instructions:

**add $v0, $a0, $t3**  # Instruction that adds two registers and stores into Return register

**slt $t0, $s1, $s0** #checks is $s1 is less than $s0 and stores a 0 or 1 in $t0

**6. Now, locate the instruction in line #7 of proc1.s. Let’s translate this instruction into machine code.**

**a. What instruction type is this? How many fields does this type of instruction have? What are the names of these fields?**

This instruction is of I-format. I format has 4 fields:

|  |  |  |  |
| --- | --- | --- | --- |
| 31 opcode 26 | 25 rs 21 | 20  rt 16 | 15 immediate  0 |

The name of field one is called **Operation Code**. The name of the second is called **Source Register(first operand)**. The third field is called **Target Register(second register)**  and the last is the **immediate**  which is just a number.

**b. Refer to the MIPS sheet, what is the value of the opcode of this instruction in Hex? What register is rs? What is the value of this register in Hex? What register is rt? What is the value of this register in Hex? What is the value of immediate in Hex?**

0x 20100019 is the value but when when we get the hex value form the machine code. We extract the 6 leftmost bits we will get 0x08. rs will be 0x00, rt is 0x10. The immediate will have hex value 0x0019

**c. Construct the machine code of line #7 using the values obtained from part 7. Write your answer in both binary and Hex formats. You can verify your answer with the Code column in Text Segment window.**

|  |  |  |  |
| --- | --- | --- | --- |
| **0x20100019** | | | |
| 0x08 | 0x00 | 0x10 | 0x0019 |
| 31 0010 00 26 | 25 00 000 21 | 20  1 0000 16 | 15 0000 0000 0001 1001  0 |
| 0001 0000 0001 0000 0000 0000 0001 1001 | | | |

**7. Now, let’s convert a machine code to a MIPS instruction. Locate address 0x00400010 from the Text Segment window.**

**a. What is the machine code at this address in Hex? Convert this code into binary.**

The machine code at address 0x00400010 is 0x0230402a. Converting this into binary would result in:

0000 0010 0011 0000 0100 0000 0010 1010

**b. From the binary version of this machine code. What is the instruction type? How can you tell? How many fields are there in this instruction type? What are the names of these fields?**

The instruction type for this is a Set Less Than we can tell because it has the binary value 0000 00. This is a R-format instruction meaning it will have 6 fields:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 Opcode 26 | 25 rs 21 | 20 rt 16 | 15 rd 11 | 10 shamt  6 | 5 funct 0 |
| 6bitsOperation Code | 5bitsSource Register | 5bitsTarget Register | 5bitsDestination Reg | 5 bits Shift Amount | 6 bitsFunction |

**c. According to the binary machine code, what is the value of each field in Hex?**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **0x0230402a** | | | | | |
| 0x00 | 0x11 | 0x10 | 0x08 | 0x00 | 0x2a |
| 31 0000 00 26 | 25 10 001 21 | 20 1 0000 16 | 15 01 000 11 | 10 000 00 6 | 5 10 1010 0 |
| 0000 0010 0011 0000 0100 0000 0010 1010 | | | | | |

**d. Refer to the MIPS sheet, what operation is this instruction? How can you tell? What is the mapping of the registers being used in this instruction?**

The operation of this instruction is a **slt**. You can tell because of the binary encoding.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 Opcode 26 | 25 rs 21 | 20 rt 16 | 15 rd 11 | 10 shamt  6 | 5 funct 0 |
| **slt** | **$s1** | **$s0** | **$t0** | **No Shift(0)** | **42dec(2ahex)** |

**e. What is the final MIPS instruction? Is it the same as the Source Column in the Text Segment window?**

The final MIPS instruction will be:

**Slt $t0, $s1, $s0**

**8. Now, let’s take a look at line #17 of proc1.s.**

**a. What format is this instruction?**

I-format because this format encompases **bne** and **beq**

**b. What are the values of opcode, rs, and rt of this instruction in hex?**

|  |  |  |  |
| --- | --- | --- | --- |
| **0001 0101 0000 0000 0000 0000 0000 0001** | | | |
| 31 opcode 26 | 25 rs 21 | 20  rt 16 | 15 immediate  0 |
| 0001 01 | 01 000 | 0 0000 | 0000 0000 0000 0001 |
| 0x05 | 0x08 | 0x00 | 0x0001 |

**c. What is the name of the target label if it takes the branch? What is the address of this label in hex? (Hint: you can find it in the Text Segment window.)**

The name of the target label is called LESS. The address of this label is 0x00000001

**d. So, do we put this address as the value of the immediate field of the instruction? Why?**

No, we actually just put the offset, in this case it would be just our hexadecimal value.

**e. How do we find the value of the immediate field? What is this value?**

This value can be found in the code column in the MIPS software, you divide by four. The value of this is 4 bytes.

**f. What is the machine code of this instruction in binary and hex formats? Does your answer match the Code column in the Text Segment window?**

|  |  |  |  |
| --- | --- | --- | --- |
| **0x15000001** | | | |
| 0x05 | 0x10 | 0x00 | 0x0001 |
| 31 0001 01 26 | 25 01 000 21 | 20  0 0000 16 | 15 0000 0000 0000 0001  0 |
| 0001 0101 0000 0000 0000 0000 0000 0001 | | | |

**9. Finally, let’s convert the j instruction in line #20.**

**a. What format is this instruction? How many fields are there in this format?**

This instruction is in j-format, this format only has two fields:

|  |  |
| --- | --- |
| 31 opcode 26 | 25 address 0 |

**b. What is the opcode of this instruction in hex?**

The opcode of this instruction in hex has the value of:

0x02

**c. What label and address does this instruction jump to?**

The label being jumped to by this instruction is **GREQ**, and the address it jumps to has the hexadecimal value of **0x00400030**

**d. How many bits can you use in the address field of the instruction? How can we “squeeze” the address into this field? What are the reasons behind this approach? What is the value of the address field in binary?**

The address field of J-format has a total of 26 bits available for storage. The way we “squeeze” the address in this field is always assuming that the first six bits will never change because in our program our lines are in close proximity. We then disregard the last two bits since we always shift in multiples of four, finally, in between these two changes we insert our 26 bit value from the address field.

**e. What is the machine code of this instruction in binary and hex? Is it the same as what’s in the Code Column of the Text Segment window?**

|  |  |
| --- | --- |
| **0x0810000c** | |
| 31 0000 10 26 | 25 00 0001 0000 0000 0000 0000 1100 0 |
| 0x02 | 0x010000c |
| 0010 0000 0100 0000 0000 0000 0011 0000 | |

**(Assignment Individual)**

**Convert the following line in proc2.s to machine code and then back to MIPS instructions:**

**Line #7**

**addi $s0, $zero, -15**

This is I-format so we are going to have a field:

0x2010fff1

|  |  |  |  |
| --- | --- | --- | --- |
| 31 opcode 26 | 25 rs 21 | 20  rt 16 | 15 immediate  0 |

addi has a binary value of 00100, this will be our opcode:

|  |  |  |  |
| --- | --- | --- | --- |
| 0010 00 | 25 rs 21 | 20  rt 16 | 15 immediate  0 |

In order to represent -15 we first have to turn our value into binary: 0000 1111 then we need to obtain our 1’s complement: 1111 0000 and finally we apply 2’s complement: 1111 0001

|  |  |  |  |
| --- | --- | --- | --- |
| 0010 00 | 25 rs 21 | 20  rt 16 | 1111 1111 1111 0001 |

We also know that our source register is going to have the value of zero since it utilizes $zero:

|  |  |  |  |
| --- | --- | --- | --- |
| 0010 00 | 00 000 | 20  0001 16 | 1111 1111 1111 0001 |

**Line #14**

**slt $t0, $s0, $s1**

This is R-format so we are going to have a field:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 Opcode 26 | 25 rs 21 | 20 rt 16 | 15 rd 11 | 10 shamt  6 | 5 funct 0 |

We know that slt has an opcode of 0000 00 and function of 2a(0010 1010)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0000 00 | 25 rs 21 | 20 rt 16 | 15 rd 11 | 10 shamt 8 00 | 10 1010 |

We also know that register $s0 and $s1 have binary encoding 10000 and 10001:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0000 00 | 10 000 | 1 0001 | 15 rd 11 | 10 shamt 8 00 | 10 1010 |

Since we are not shifting by any amount we will have 0000:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0000 00 | 10 000 | 1 0001 | 15 rd 120 | 000 00 | 10 1010 |

Lastly, our destination register will be $t0:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0000 00 | 10 000 | 1 0001 | 0100 0 | 000 00 | 10 1010 |

**Line #17**

**beq $t0, $zero, LEEQ**

This is I-format so we are going to have a field:

|  |  |  |  |
| --- | --- | --- | --- |
| 31 opcode 26 | 25 rs 21 | 20  rt 16 | 15 immediate  0 |

We know the value of our source register $t0(01000)and the value of our target register $zero(0000):

|  |  |  |  |
| --- | --- | --- | --- |
| 31 opcode 26 | 01 000 | 0 0000 | 15 immediate  0 |

beq has an opcode of 00 0100:

|  |  |  |  |
| --- | --- | --- | --- |
| 0001 00 | 01 000 | 0 0000 | 15 immediate  0 |

Finally our branch label LEEQ has the hex value of 0x00000006:

|  |  |  |  |
| --- | --- | --- | --- |
| 0001 00 | 01 000 | 0 0000 | 0000 0000 0000 0110 |

**Line #20**

**j GRT**

This is J-format so we are going to have a field:

|  |  |
| --- | --- |
| 31 opcode 26 | 25 address 0 |

j has an opcode of 00 0010:

|  |  |
| --- | --- |
| 00 0010 | 25 address 0 |

Finally GRT has a hex value of 0x0040001c which is

0000 0000 0100 0000 0000 0000 0001 1100 in binary;

we first get rid of the last two digits in our 32 bits to obtain

0000 0000 0100 0000 0000 0000 0001 11 since this is always in multiples of four leaving us with 30 bits.

We then make the assumption that the last 4 bits are always going to be the same so we rid of

0000 0100 0000 0000 0000 0001 11 leaving us with 26 bits total, now we can “squeeze” into address:

|  |  |
| --- | --- |
| 0000 10 | 00 0001 0000 0000 0000 0000 0111 |