EECS 2021 lab A Report

Name: Rayhaan Yaser Mohammed
Student ID: 219739887
Date : 25/09/23
Intro: this lab is an introduction on assembly code and doing simple operations like addition, subtraction and multiplication. And Assembly machine instructions.

Explanation:

A1: Integer arithmetic (add, addi, sub)

1.1. a1a05.asm

This code stores values of 2 and 3 in registers and performs simple addition to add the values and stores it and then outputs it in register as 5.

1.2. a1b05.asm

This code performs the same as a1a05.asm but uses (addi) which is add immediate which does the same commands but in a shorter version

1.3. a1c05.asm

This code does the same addition but for a positive value and a negative value which can be seen as subtraction.

1.4. a1d05.asm

This code uses the (sub) which is subtract to subtract values stored in the registers x6 and x7 and outputs the result stored in register x5 but the result in hex value is seen an a very big hex code which is 0xffffffffffffff which is basically -1.

A2. Binary Shifts(slli, srli, srai)

2.1 a2a05.asm

This code uses addi x6, x0, 17 to store the decimal 17 in the register x6 but adding immediate 0 + 17 and then storing the output 17 in x6. The by writing the command slli x5, x6, 2 : this code function by taking the value stored in register x6 and then shifting it by 2 which is by multiplying it by 4 because of the command slli(shift left logic immediate) as any shift to the left multiplies the number by another factor of 2 and then store the result in x5.

2.2 a2b05.asm

This code performs by shifting the logic immediately to the right which by a similar method divides the value by a factor of 2. In the code we first store the value 88 in register x6 by typing addi x6, x0, 88 and srli x5, x6, 3 we can divide the value 88 stored in x6 by 3x 2 which is 8 and then storing the value in register x5 which would be 88/8 = 11.

2.3 a2c05.asm

This code is the same but the values entered now are changed from positive to negative value and we can see that the result is a very big number this is due to 0 bits fed into the MSB part of the number during the shift

2.4 a2d05.asm

The a2b05.asm code calculates the values perfectly for positive values but when any negative integers are supplied it produces a very big value so this code solves this issue but feeding 1 bit into the register during the shift instead of 0 bits when doing it for positive

values. The srai(shift right arithmetic immediate) divides the values properly for negative values. And outputs the correct value which is -11 being stored in register x5.

2.5 a2e05.asm

This code stores the 0x123 in register x6, then multiplies the value in x6 by 2 to power of 56 then stores the value in register x7 , then the srli function divides the value by 2 to power of 60 then stores the result in register x5. Which is x5 = 2.

A3 logical operations (andi, or, xori)

3.1 a3a05.asm

This code stores the value of 0x123 in register x6 and then by using the andi operator the code will extract bits [7:4] value using the and operator and logically shifts the value of it to the right by 4 bits .

3.2 a3b05.asm

In this source code we store 0x123 in a register and then use the andi operator to combine 0x123 and 0x0f0 and then store it in x6 and does the same for register x7 which is combining 0x456 and 0xf0f and then storing it in x7. At last it uses the or operator to combine the values in x6 and x7 by combining the value 0x456 and obtaining the result 0x426 and store it in register x5.

3.3 a3c05.asm

This code basically uses the xor function to invert the value stored in the register and then store the inverted value in an another register which in this source does is -291stored in x5 this is inverted and stored in 291 in x6

A4 Loading larger Values (lui, EQU)

4.1 a4a05.asm

This source code uses lui and addi. Which in this case lui stores the most significant 20 bits and the addi then stores the least 12 bits then by addi 2 is added to the least 12 bits and then the result is displayed as the whole 32 bits with the 20 bits stored previously, this is achieved by the lui function. Here 4096 is broken into binary and the from this 4095 is stored and x6 is saved as just one and then using addi we do the ad immediate and then output the whole value of 4095 + 1 +2 = 4098 stored in the register x6.

4.2 a4b05.asm

This code performs a similar action as the above code but in a more long method.
First the most significant 20 bits of 4098 is stored and the register saves value of 1 in x6 then by addi we add immediate 2 and the x6 is now of value 3 then by another addi we assign x7 a value of 3 then by using the add function we add x6 and x7 then output everything in x5 which is a total value of 4095+1+2+3 = 4101.

4.3 a4c05.asm

This code performs the same function but by using only 2 registers instead of 3 which reduces memory usage but the value should be the same as previous code which is 4101.

A5 Assembly language and machine instruction and formats

5.1 a5a05.asm

add x5,x6,x7

This source code outputs the binary

0000000 00111 00110 000 00101 0110011

By using the assembly instruction and format we can look at the binary output above and In this specific example, the rd field [11:7] has a binary value of 00101, which corresponds to the decimal value 5, indicating that the destination register for this instruction is x5.

5.2 a5b05.asm

The source code

addi x6, x0, 2 compiles the given binary code.

The compiled machine code is essentially a sequence of 32 bits organised in five groups corresponding to the fields in the I instruction format row in the RISC-V CORE INSTRUCTION FORMATS table. In the computer memory, however, such sequences of bits are continuous so our example bit sequence could also be denoted as follows:

0000 0000 0010 0000 0000 0011 0001 0011

This can be re written as hex code : 0x00200313

And by dividing the code by risc v standard

000000000010 00000 000 00110 0010011

We can see that the highlighted part the value is stored in x6 as instructed from the code.