Bryan Martinez

CSE 3666

10 February 2024

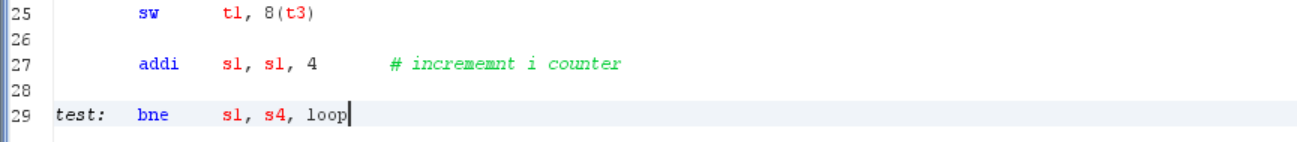
A screenshot of a computer

Description automatically generated1)

The change that needed to be made was the addition of A[i] with 4. This was done on line 11, where the contents at the address of A[i] are added by 4 and then saved to B[i]. As there are 8 instructions that are ran with every iteration, and there will be 100 iterations, that will be 800 instructions, plus the first two instructions that run in the beginning of the program. In total, there will be 802 instructions executed in total.

1. A screenshot of a computer

   Description automatically generated



To load and store the contents at A[i] and B[i] are the same as 1a, but when doing the next index of A[i+1], an offset of 4 is used when loading words and saving words. So, for B[i+1] and A[i+1], an offset of 4 is used. Then, A[i+2] and B[i+2] use an offset of 8. Thus, to find the offset is done by multiplying the number being added to i by 4. As such, 17 instructions are executed for each iteration, and there will be 25 iterations. Thus, it will be 425 plus the two instructions that run at the beginning of the program. In total, 427 instructions are executed.

A close-up of a test

Description automatically generatedA screenshot of a computer code

Description automatically generated2)

In main, all the counters were created for the nested for loops. The maximum number of iterations was also created for each for loop. For the first for loop, a greater than or equal branch is used to exit the loop when the conditions are met. Then, it moves on directly to another branch that checks to see if j is under 8. If it is, then it moves on to performing calculations to find the memory address of the 2-d array and save information to said address. If the nested for loop reaches past 8 iterations, then it moves on to resetting j back to 0, incrementing i by 1, and returning to the beginning of the for loop. When inside the nested for loop, a left bit shift of 8 bits is performed on I in order to multiply it by 256. Then this number is added with j and saved to t0 register, which becomes the information that will be stored at the memory address that is to be calculated. Two-bit shifts to the left are performed. The first is 5 bits to the left, thus multiplying i by 32 as i represents the row of element. Thus, an entire row takes up 32 bytes as each element in the row is 4 bytes. Then the second bit shifts to the left is 2 bits, thus it multiplies j by 4. Then, both calculations are then added together and saved to register t3 where it is added to s9 to move s9 to that specific memory address that was calculated. Then, we save t0 to s9 by using sw t0, 0(s9). J is then incremented by 1 and a equal branch is used to move back to the beginning of the for loop.

3)

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A white screen with green text

Description automatically generated

In order to do addition of digits stored in memory as characters, I had to do conversions between ASCII and decimals. This was done by subtracting the loaded byte by the ascii value of ‘0’. After converting into a decimal from both str1 and str2, they are added together, taking into account any remainders from previous iterations. If the sum is greater than 10, then the remainder is dealt with by subtracting 10 and then setting s4 in the program to 1 for the next iteration. If the sum is less than 10, then s4 is added in case there is a 1, and then it is reset back to 0. Then, the total sum of the two digits is converted back to an ASCII by adding the ascii value of ‘0’ to the sum and then loaded to s3 after it was offset by t0.

4)

1. Instruction: or s1, s2, s3

Register Values: or x9, x18, x19

R-type:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Funct7 | rs2 | rs1 | funct3 | rd | opcode |
| 0000000 | 10011 | 10010 | 110 | 01001 | 0110011 |

Machine code (Binary): 0000 0001 0011 1001 0110 0100 1011 0011

Machine code (Hex): 0x013964b3

1. Instruction: slli t1, t2, 16

Register Values: slli x6, x7, 16

I-type

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Funct7 | Imm[0:4] | rs1 | funct3 | rd | opcode |
| 0000000 | 10000 | 00111 | 001 | 00110 | 0010011 |

Machine code (Binary): 0000 0001 0000 0011 1001 0011 0001 0011

Machine code (Hex): 0x01039313

1. Instruction: xori x1, x1, -1

Register values: xori x1, x1, -1

I-type

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Imm[11:0] | rs1 | funct3 | rd | opcode |
| 111111111111 | 00001 | 100 | 00001 | 0010011 |

Machine code (Binary): 1111 1111 1111 0000 1100 0000 1001 0011

Machine code (Hex): 0xfff0c093

1. Instruction: lw x2, -100(x3)

Register Values: lw x2, -100(x3)

I-type

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Imm[11:0] | rs1 | funct3 | rd | opcode |
| 111110011100 | 00011 | 010 | 00010 | 0000011 |

Machine code (Binary): 1111 1001 1100 0001 1010 0001 0000 0011

Machine code (Hex): 0xf9c1a103

5)

1. S-type

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Imm[11:5] | rs2 | rs1 | funct3 | Imm[4:0] | opcode |
| 1111111 | 01010 | 11001 | 010 | 10000 | 0100011 |

Hex: 0xfeaca823

Binary: 1111 1110 1010 1100 1010 1000 0010 0011

Instruction: sw x10, -16(x25)

1. I-type

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Imm[11:0] | rs1 | funct3 | rd | opcode |
| 000001000000 | 00100 | 000 | 01110 | 0010011 |

Hex: 0x04020713

Binary: 0000 0100 0000 0010 0000 0111 0001 0011

Instruction: addi x14, x4, 64

1. R-type

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Funct7 | rs2 | rs1 | funct3 | rd | opcode |
| 0000000 | 00101 | 01010 | 111 | 10111 | 0110011 |

Hex: 0x00557bb3

Binary: 0000 0000 0101 0101 0111 1011 1011 0011

Instructions: and x23, x10, x5

1. I-type

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Funct7 | Imm[0:4] | rs1 | funct3 | rd | opcode |
| 0100000 | 10100 | 11111 | 101 | 11110 | 0010011 |

Hex: 0x414fdf13

Binary: 0100 0001 0100 1111 1101 1111 0001 0011

Instructions: srai x30 x31, 20