I pledge my honor that I have abided by the Stevens Honor System

**Question 1**



**Question 2**

2.1)

2.2)

­

2.3)

Therefore, is the faster processor for this instruction set.

**Question 3**

3.1) The variables that are exhibiting temporal locality are the index variables i and j, the length variables n and m, the references for both the A array and Anew array, and finally the errs variable. This is because they are being referenced every iteration of the loop and being often by the program.

3.2) The variables that are exhibiting spatial locality are the values of both the A array and the Anew array that are referenced by row. More specifically this would include Anew[j][i], A[j][i+1], A[j][i-1], and A[j][i]. This is because when an array is stored into memory, the values of each row are stored next to each other. Therefore, when the loops iterate over each row and get new values, they are right next to each other and easy to access with caches.

3.3) The program would become slightly slower. This is because C is a row-major language, which means that in memory, values in each row of a matrix are stored next to each other. Therefore, if we switched the i and j variables, we would not be efficiently taking advantage of the spatial locality that is available when getting values from the arrays.

**Question 4**

4.1) C = 16 \* 4 = 64 bytes

S = 16 s = 4

B = 4 b = 2

E = 1 t = 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Hex Value** | **Binary Address** | **Tag** | **Set Index** | **Offset** | **Hit or Miss** |
| 0x43 | 01000011 | 01 | 0000 | 11 | miss |
| 0xc4 | 11000100 | 11 | 0001 | 00 | miss |
| 0x2b | 00101011 | 00 | 1010 | 11 | miss |
| 0x42 | 01000010 | 01 | 0000 | 10 | hit |
| 0xc5 | 11000101 | 11 | 0001 | 01 | hit |
| 0x28 | 00101000 | 00 | 1010 | 00 | hit |
| 0xbe | 10111110 | 10 | 1111 | 10 | miss |
| 0x05 | 00000101 | 00 | 0001 | 01 | miss |
| 0x92 | 10010010 | 10 | 0100 | 10 | miss |
| 0x2a | 00101010 | 00 | 1010 | 10 | hit |
| 0xba | 10111010 | 10 | 1110 | 10 | miss |
| 0xbd | 10111101 | 10 | 1111 | 01 | miss |

4.2) C = 8 \* 8 = 64 bytes

S = 8 s = 3

B = 8 b = 3

E = 1 t = 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Hex Value** | **Binary Address** | **Tag** | **Set Index** | **Offset** | **Hit or Miss** |
| 0x43 | 01000011 | 01 | 000 | 011 | miss |
| 0xc4 | 11000100 | 11 | 000 | 100 | miss |
| 0x2b | 00101011 | 00 | 101 | 011 | miss |
| 0x42 | 01000010 | 01 | 000 | 010 | miss |
| 0xc5 | 11000101 | 11 | 000 | 101 | miss |
| 0x28 | 00101000 | 00 | 101 | 000 | hit |
| 0xbe | 10111110 | 10 | 111 | 110 | miss |
| 0x05 | 00000101 | 00 | 000 | 101 | miss |
| 0x92 | 10010010 | 10 | 010 | 010 | miss |
| 0x2a | 00101010 | 00 | 101 | 010 | hit |
| 0xba | 10111010 | 10 | 111 | 010 | hit |
| 0xbd | 10111101 | 10 | 111 | 101 | hit |

**Question 5**

5.1) C = 512 \* 4 = 2048 bytes

S = 512 s = 9

B = 4 b = 2

E = 1 t = 53

The first 53 bits would be used for the tag. 9 bits would be used for the set index. Then 2 bits would be used for the byte offset.

5.2) C = 64 \* 32 = 2048 bytes

S = 64 s = 6

B = 32 b = 5

E = 1 t = 53

The first 53 bits would be used for the tag. 6 bits would be used for the set index. Then 5 bits would be used for the byte offset.

5.3) For 5.1:

Total bits in cache = (1 valid bit + 53 bits for tag + (8 \* 4) bits for data) \* 64 sets

= 19840 bits

Total bits storing data = 2048 \* 8 = 16384 bits

For 5.2:

Total bits in cache = (1 valid bit + 53 bits for tag + (8 \* 32) bits for data) \* 512 sets

= 44032 bits

Total bits storing data = 2048 \* 8 = 16384 bits

5.4) C = 512 \* 4 = 2048 bytes

S = 256 s = 8

B = 4 b = 2

E = 2 t = 54

The first 54 bits would be used for the tag. 8 bits would be used for the set index. Then 2 bits would be used for the byte offset.

**Question 6**

C = 16 \* 4 = 64 bytes

S = 4 s = 2

B = 4 b = 2

E = 4 t = 8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Hex Value** | **Binary Address** | **Tag** | **Set Index** | **Offset** |
| 0x143 | 000101000011 | 00010100 | 00 | 11 |
| 0xc4a | 110001001010 | 11000100 | 10 | 10 |
| 0x22b | 001000101011 | 00100010 | 10 | 11 |
| 0x42f | 010000101111 | 01000010 | 11 | 11 |
| 0x492 | 010010010010 | 01001001 | 00 | 10 |
| 0x2a2 | 001010100010 | 00101010 | 00 | 10 |
| 0x3ba | 001110111010 | 00111011 | 10 | 10 |
| 0xb2d | 101100101101 | 10110010 | 11 | 01 |

See cache below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Set/Line** | *Line 0* | *Line 1* | *Line 2* | *Line 3* |
| *Set 0* | Tag = 00010100  Data  0x140, 0x141, 0x142, **0x143** | Tag = 01001001  Data  0x490, 0x491, **0x492**, 0x493 | Tag = 00101010  Data  0x2a0, 0x2a1, **0x2a2**, 0x2a3 |  |
| *Set 1* |  |  |  |  |
| *Set 2* | Tag = 11000100  Data  0xc49, 0xc48, **0xc4a**, 0xc4b | Tag = 00100010  ­Data  0x228, 0x229, 0x22a, **0x22b** | Tag = 00111011  Data  0x3b8, 0x3b9, **0x3ba**, 0x3bb |  |
| *Set 3* | Tag = 01000010  Data  0x42c, 0x42d, 0x42e, **0x42f** | Tag = 10110010  Data  0xb2c, **0xb2d**, 0xb2e, 0xb2f |  |  |