Jordan Wu

U0900517

Homework Assignment 10

Assume the critical loop in some embedded system (with 16-bit addresses and 32-bit instructions) accesses memory (as instructions are fetched) in the following address order:

```
 16
 20
 24
 28
 32
 36
 60
 64
 56
 60
 64
 68
 56
 60
 64
 72
 76
 92
 96
100
104
108
112
120
124
128
144
148
... Pattern repeats...
```

The addresses above are in decimal, memory is byte-addressed with 16-bit addresses, and each memory access reads one 32-bit word from memory. As memory is accessed, the data read from these locations can be cached. (The data is not shown because it is not needed for this problem.) This problem is concerned with the performance of caches that cache the data from these memory locations *in the specified order*.

To perform the needed analysis, you will design and simulate three basic cache types: fully associative, direct mapped, and set associative. You will choose the block size for each row, as well as the number of rows, and then you will simulate each cache using the memory access pattern above. By keeping track of hits and misses, you will be able to compute the cache performance.

For the analysis, consider the cache behavior for the above memory access pattern *after the first few repetitions of the loop*. In other words, since this memory access pattern happens many times, you should not base your performance estimates on the first repetition of these accesses. Instead, base your performance estimates on a subsequent repetition (since that would be the *common case*).

Cache constraints

Assume that each cache you design will have the following timing characteristics:

- **1 cycle access time for cache hits,**
- **20 cycles plus 1 cycle per byte access time for cache misses, and**
- **A cache miss will stall the processor until all the memory for the relevant cache block has been entirely read (a simplifying assumption).**

(For example, if a sequence of reads are cache hits, the reads would complete one per cycle. On the other hand, if a sequence of reads are all cache misses, and the cache has eight byte blocks, one read would complete every 28 cycles.)

Additionally, your cache organization must allow for essentially instant computations of row indices, tags, and offsets from binary memory addresses. (The number of rows and the number of bytes per block must be powers of 2.)

Finally, assume the number of storage bits for the cache is limited to 900 bits. This means that the valid bits, tags, and data storage for the entire cache must fit within 900 bits. In addition:

> **For set associative caches, you are also required to keep a least recently used (LRU) counter for *each* data block. For an N-way set associative cache there must be *ceil(lg n)* bits in *each* LRU counter.**
>
> **For fully associative caches, you are also required to keep a least recently used (LRU) counter for *each* data block. For a fully associative cache with r rows, there must be *ceil(lg r)* bits in *each* LRU counter.**

This small size constraint means that there will be cache misses no matter what cache is chosen, and the ideal cache is not obvious.

Fully Associative Cache

A cache structure in which a block can be placed in any location in the cache. All the entries in the cache must be searched because a block can be placed in any one. I will be using MATLAB for this

homework assignment. Looking at the simulated cache example program output provided on the class website.

My MATLAB code:

```
% Fully Associative Cache
clc        % clear command window
clear all  % clear all variables

% Initialize cache setup
sequence = [16 20 24 28 32 36 60 64 72 76 92 96 100 104 108 112 ...
            120 124 128 144 148];
wordBits = 32;
valid = 1;
rows = 8;
blockSizeWords = 2;
blockSizeBytes = (blockSizeWords*wordBits)/8;
blockSizeBits = blockSizeWords*wordBits;
offset = ceil(log2(blockSizeBytes));
LRU = ceil(log2(rows));
tag = 16-offset;
usedBits = rows*(valid+tag+blockSizeBits+LRU);
totalStorage = 900;
unusedBits = totalStorage-usedBits;
hitTime = 1;
missTime = 20+blockSizeBytes;

% Print out cache setup
fprintf('Fully associative cache with %d rows and %d bytes per row:\n',...
        rows, blockSizeBytes);
fprintf('Offset address bits: %d\n', offset);
fprintf('Bits in the valid bit: %d\n', valid);
fprintf('Bits in the data block: %d\n', blockSizeBits);
fprintf('Bits in the tag: %d\n', tag);
fprintf('Bits in the LRU: %d\n', LRU);
fprintf('Total bits used: %d, bits remaining: %d\n', usedBits,unusedBits);
fprintf('Hit time: %d\n', hitTime);
fprintf('Miss time: %d\n\n', missTime);

% Initialize cache
sequenceLength = length(sequence);
cacheValidBit = zeros(rows,1);
cacheTag = zeros(rows,1);
cacheDataBlock = zeros(rows,blockSizeWords);
cachedRow = zeros(rows,1);
cacheHit = 0;
cacheMiss = 0;
needLRU = 0;
loop = 2;

% Accessing cache
for i=1:loop
    cacheHit = 0;
    cacheMiss = 0;
    for i=1:sequenceLength
        isHit = false;
```

```
        currentAddress = sequence(i);
        currentTag = floor(sequence(i)/blockSizeBytes);
        for j=1:rows
            if(cacheValidBit(j,1)==1 && currentTag==cacheTag(j,1))
                for k=1:blockSizeWords
                    if(currentAddress==cacheDataBlock(j,k))
                        isHit = true;
                        cacheHit = cacheHit+1;
                    fprintf('Accessing %d (tag %d): hit from row %d\n',...
                       currentAddress,currentTag,j-1);
                    end
                end
            end
        end
        if(isHit==0)
            cacheMiss = cacheMiss+1;
            for a=1:rows
                if(cachedRow(a,1)==0)
                    if(a == rows)
                        needLRU = 1;
                    end
                cacheValidBit(a,1)=1;
                cacheTag(a,1)=currentTag;
                cachedRow(a,1)=1;
                index = i;
                index = currentAddress;
                for b=1:blockSizeWords
                    data(b)= index;
                    index = index+4;
                end
               cacheDataBlock(a,:)=data;
               fprintf('Accessing %d (tag %d): miss - cached to row %d\n',...
                   currentAddress,currentTag,a-1)
               break
           end
        end
        if(needLRU == 1)
            cachedRow = zeros(rows,1);
            needLRU =0;
        end
    end
end
cost = cacheHit*hitTime+cacheMiss*missTime;
fprintf('Cost in cycles for this repetition: %d\n',cost);
fprintf('Average CPI: %f\n',cost/sequenceLength);
end
```

With the following OUTPUT:

```
Fully associative cache with 8 rows and 8 bytes per row:
Offset address bits: 3
```

```
Bits in the valid bit: 1
Bits in the data block: 64
Bits in the tag: 13
Bits in the LRU: 3
Total bits used: 648, bits remaining: 252
Hit time: 1
Miss time: 28

Accessing 16 (tag 2): miss - cached to row 0
Accessing 20 (tag 2): hit from row 0
Accessing 24 (tag 3): miss - cached to row 1
Accessing 28 (tag 3): hit from row 1
Accessing 32 (tag 4): miss - cached to row 2
Accessing 36 (tag 4): hit from row 2
Accessing 60 (tag 7): miss - cached to row 3
Accessing 64 (tag 8): miss - cached to row 4
Accessing 72 (tag 9): miss - cached to row 5
Accessing 76 (tag 9): hit from row 5
Accessing 92 (tag 11): miss - cached to row 6
Accessing 96 (tag 12): miss - cached to row 7
Accessing 100 (tag 12): hit from row 7
Accessing 104 (tag 13): miss - cached to row 0
Accessing 108 (tag 13): hit from row 0
Accessing 112 (tag 14): miss - cached to row 1
Accessing 120 (tag 15): miss - cached to row 2
Accessing 124 (tag 15): hit from row 2
Accessing 128 (tag 16): miss - cached to row 3
Accessing 144 (tag 18): miss - cached to row 4
Accessing 148 (tag 18): hit from row 4
Cost in cycles for this repetition: 372
Average CPI: 17.714286
Accessing 16 (tag 2): miss - cached to row 5
Accessing 20 (tag 2): hit from row 5
Accessing 24 (tag 3): miss - cached to row 6
Accessing 28 (tag 3): hit from row 6
Accessing 32 (tag 4): miss - cached to row 7
Accessing 36 (tag 4): hit from row 7
Accessing 60 (tag 7): miss - cached to row 0
Accessing 64 (tag 8): miss - cached to row 1
Accessing 72 (tag 9): miss - cached to row 2
Accessing 76 (tag 9): hit from row 2
Accessing 92 (tag 11): miss - cached to row 3
Accessing 96 (tag 12): miss - cached to row 4
Accessing 100 (tag 12): hit from row 4
```

```
Accessing 104 (tag 13): miss - cached to row 5
Accessing 108 (tag 13): hit from row 5
Accessing 112 (tag 14): miss - cached to row 6
Accessing 120 (tag 15): miss - cached to row 7
Accessing 124 (tag 15): hit from row 7
Accessing 128 (tag 16): miss - cached to row 0
Accessing 144 (tag 18): miss - cached to row 1
Accessing 148 (tag 18): hit from row 1
Cost in cycles for this repetition: 372
Average CPI: 17.714286
```

The Average CPI for this cache can be improve by adjusting the number of rows and the bytes per rows. Looking at the sequence of address, it would be wise to use spatial locality. To implement the spatial locality the bytes per rows should be increased to say eight words per block. If we increase the words per block, the rows must be decreased and must be a power of 2. Ideal block size for a fully associative cache has a row of 2 and 32 bytes per row which gives the ideal block size of 256 bits.

Results:

```
Fully associative cache with 2 rows and 32 bytes per row:
Offset address bits: 5
Bits in the valid bit: 1
Bits in the data block: 256
Bits in the tag: 11
Bits in the LRU: 1
Total bits used: 538, bits remaining: 362
Hit time: 1
Miss time: 52

Accessing 16 (tag 0): miss - cached to row 0
Accessing 20 (tag 0): hit from row 0
Accessing 24 (tag 0): hit from row 0
Accessing 28 (tag 0): hit from row 0
Accessing 32 (tag 1): miss - cached to row 1
Accessing 36 (tag 1): hit from row 1
Accessing 60 (tag 1): hit from row 1
Accessing 64 (tag 2): miss - cached to row 0
Accessing 72 (tag 2): hit from row 0
Accessing 76 (tag 2): hit from row 0
Accessing 92 (tag 2): hit from row 0
Accessing 96 (tag 3): miss - cached to row 1
Accessing 100 (tag 3): hit from row 1
```

```
Accessing 104 (tag 3): hit from row 1
Accessing 108 (tag 3): hit from row 1
Accessing 112 (tag 3): hit from row 1
Accessing 120 (tag 3): hit from row 1
Accessing 124 (tag 3): hit from row 1
Accessing 128 (tag 4): miss - cached to row 0
Accessing 144 (tag 4): hit from row 0
Accessing 148 (tag 4): hit from row 0
Cost in cycles for this repetition: 276
Average CPI: 13.142857
Accessing 16 (tag 0): miss - cached to row 1
Accessing 20 (tag 0): hit from row 1
Accessing 24 (tag 0): hit from row 1
Accessing 28 (tag 0): hit from row 1
Accessing 32 (tag 1): miss - cached to row 0
Accessing 36 (tag 1): hit from row 0
Accessing 60 (tag 1): hit from row 0
Accessing 64 (tag 2): miss - cached to row 1
Accessing 72 (tag 2): hit from row 1
Accessing 76 (tag 2): hit from row 1
Accessing 92 (tag 2): hit from row 1
Accessing 96 (tag 3): miss - cached to row 0
Accessing 100 (tag 3): hit from row 0
Accessing 104 (tag 3): hit from row 0
Accessing 108 (tag 3): hit from row 0
Accessing 112 (tag 3): hit from row 0
Accessing 120 (tag 3): hit from row 0
Accessing 124 (tag 3): hit from row 0
Accessing 128 (tag 4): miss - cached to row 1
Accessing 144 (tag 4): hit from row 1
Accessing 148 (tag 4): hit from row 1
Cost in cycles for this repetition: 276
Average CPI: 13.142857
```

Assuming that my code is correct and that changing rows to 2 and blockSizeWords to 8. This will give a better Average CPI and will decrease the number of bits needed in the cache. Here is the content of my cache tag and the valid bits. Note that this is a fully associate cache which uses all the rows, and after all the rows are filled and a miss occurs the LRU is used. Lastly the last two misses in my simulation should store the tags into my cacheTag which when I type cacheTag and cacheValidBit into the command window I get the following.

```
cacheTag =
```

```
     3
     4


cacheValidBit =

     1
     1
```

Direct Mapped Cache

My MATLAB code:

```
% Direct Mapped Cache
clc        % clear command window
clear all  % clear all variables

% Initialize cache setup
sequence = [16 20 24 28 32 36 60 64 72 76 92 96 100 104 108 112 ...
            120 124 128 144 148];
wordBits = 32;
valid = 1;
rows = 4;
index = log2(rows);
blockSizeWords = 4;
blockSizeBytes = (blockSizeWords*wordBits)/8;
blockSizeBits = blockSizeWords*wordBits;
offset = log2(blockSizeBytes);
tag = 16-offset-index;
usedBits = rows*(valid+tag+blockSizeBits);
totalStorage = 900;
unusedBits = totalStorage-usedBits;
hitTime = 1;
missTime = 20+blockSizeBytes;

% Print out cache setup
fprintf('Direct mapped cache with %d rows and %d bytes per row:\n',...
        rows, blockSizeBytes);
fprintf('Offset address bits: %d\n', offset);
fprintf('Bits in the index bit: %d\n', index);
fprintf('Bits in the valid bit: %d\n', valid);
fprintf('Bits in the data block: %d\n', blockSizeBits);
fprintf('Bits in the tag: %d\n', tag);
fprintf('Total bits used: %d, bits remaining: %d\n', usedBits,unusedBits);
fprintf('Hit time: %d\n', hitTime);
fprintf('Miss time: %d\n\n', missTime);

% Initialize cache
sequenceLength = length(sequence);
cacheValidBit = zeros(rows,1);
cacheTag = zeros(rows,1);
cacheDataBlock = zeros(rows,blockSizeWords);
```

```
cacheHit = 0;
cacheMiss = 0;
loop = 2;

% Accessing cache
for i=1:loop
    cacheHit = 0;
    cacheMiss = 0;
    for i=1:sequenceLength
        isHit = false;
        currentAddress = sequence(i);
        currentRow = mod(floor(currentAddress/(2^offset)),rows);
        currentTag = floor(sequence(i)/(blockSizeBytes*rows));
        if(cacheValidBit(currentRow+1,1)==1 &&
      currentTag==cacheTag(currentRow+1,1))
            for j=1:blockSizeWords
               if(currentAddress==cacheDataBlock(currentRow+1,j))
                   isHit=true;
                   cacheHit = cacheHit+1;
                   fprintf('Accessing %d (index %d) (tag %d): hit from row
%d\n',...
                   currentAddress,currentRow,currentTag,currentRow);
               end
            end
        end
        if(isHit==false)
            cacheMiss = cacheMiss+1;
            cacheValidBit(currentRow+1,1)=1;
            cacheTag(currentRow+1,1)=currentTag;
            count = currentAddress;
                for b=1:blockSizeWords
                    data(b)= count;
                    count = count+4;
                end
             cacheDataBlock(currentRow+1,:)=data;
             fprintf('Accessing %d (index %d) (tag %d): miss - cached to row
%d\n',...
                   currentAddress,currentRow,currentTag,currentRow)
        end

    end
cost = cacheHit*hitTime+cacheMiss*missTime;
fprintf('Cost in cycles for this repetition: %d\n',cost);
fprintf('Average CPI: %f\n',cost/sequenceLength);
end
```

With the following OUTPUT:

```
Direct mapped cache with 4 rows and 16 bytes per row:
Offset address bits: 4
Bits in the index bit: 2
Bits in the valid bit: 1
Bits in the data block: 128
Bits in the tag: 10
Total bits used: 556, bits remaining: 344
```

```
Hit time: 1
Miss time: 36

Accessing 16 (index 1) (tag 0): miss - cached to row 1
Accessing 20 (index 1) (tag 0): hit from row 1
Accessing 24 (index 1) (tag 0): hit from row 1
Accessing 28 (index 1) (tag 0): hit from row 1
Accessing 32 (index 2) (tag 0): miss - cached to row 2
Accessing 36 (index 2) (tag 0): hit from row 2
Accessing 60 (index 3) (tag 0): miss - cached to row 3
Accessing 64 (index 0) (tag 1): miss - cached to row 0
Accessing 72 (index 0) (tag 1): hit from row 0
Accessing 76 (index 0) (tag 1): hit from row 0
Accessing 92 (index 1) (tag 1): miss - cached to row 1
Accessing 96 (index 2) (tag 1): miss - cached to row 2
Accessing 100 (index 2) (tag 1): hit from row 2
Accessing 104 (index 2) (tag 1): hit from row 2
Accessing 108 (index 2) (tag 1): hit from row 2
Accessing 112 (index 3) (tag 1): miss - cached to row 3
Accessing 120 (index 3) (tag 1): hit from row 3
Accessing 124 (index 3) (tag 1): hit from row 3
Accessing 128 (index 0) (tag 2): miss - cached to row 0
Accessing 144 (index 1) (tag 2): miss - cached to row 1
Accessing 148 (index 1) (tag 2): hit from row 1
Cost in cycles for this repetition: 336
Average CPI: 16.000000
Accessing 16 (index 1) (tag 0): miss - cached to row 1
Accessing 20 (index 1) (tag 0): hit from row 1
Accessing 24 (index 1) (tag 0): hit from row 1
Accessing 28 (index 1) (tag 0): hit from row 1
Accessing 32 (index 2) (tag 0): miss - cached to row 2
Accessing 36 (index 2) (tag 0): hit from row 2
Accessing 60 (index 3) (tag 0): miss - cached to row 3
Accessing 64 (index 0) (tag 1): miss - cached to row 0
Accessing 72 (index 0) (tag 1): hit from row 0
Accessing 76 (index 0) (tag 1): hit from row 0
Accessing 92 (index 1) (tag 1): miss - cached to row 1
Accessing 96 (index 2) (tag 1): miss - cached to row 2
Accessing 100 (index 2) (tag 1): hit from row 2
Accessing 104 (index 2) (tag 1): hit from row 2
Accessing 108 (index 2) (tag 1): hit from row 2
Accessing 112 (index 3) (tag 1): miss - cached to row 3
Accessing 120 (index 3) (tag 1): hit from row 3
Accessing 124 (index 3) (tag 1): hit from row 3
Accessing 128 (index 0) (tag 2): miss - cached to row 0
Accessing 144 (index 1) (tag 2): miss - cached to row 1
Accessing 148 (index 1) (tag 2): hit from row 1
```

```
Cost in cycles for this repetition: 336
Average CPI: 16.000000
EDU>> cacheTag

cacheTag =

     2
     2
     1
     1

EDU>> cacheValidBit

cacheValidBit =

     1
     1
     1
     1
```

Seem like the ideal block size is 128 bits for direct mapped cache, it consist of four words per data block. This will use the spatial locality of four words which seem to be the best solution. So in this simulation I had four rows and sixteen bytes per row. This give me the best Average CPI and used the least amount of bits for storage. Also it uses all the rows in the cache.

Set Associative Cache

My MATLAB code:

```
% N-way Set Associative Cache
clc        % clear command window
clear all  % clear all variables

% Initialize cache setup
sequence = [16 20 24 28 32 36 60 64 72 76 92 96 100 104 108 112 ...
            120 124 128 144 148];
wordBits = 32;
valid = 1;
N = 2;
rows = 2;
index = log2(rows);
blockSizeWords = 4;
blockSizeBytes = (blockSizeWords*wordBits)/8;
blockSizeBits = blockSizeWords*wordBits;
offset = log2(blockSizeBytes);
LRU = ceil(log2(blockSizeWords));
tag = 16-offset-index;
usedBits = rows*N*(valid+tag+blockSizeBits+LRU);
```

```
totalStorage = 900;
unusedBits = totalStorage-usedBits;
hitTime = 1;
missTime = 20+blockSizeBytes;

% Print out cache setup
fprintf('%d-way set associative cache with %d rows and %d bytes per
row:\n',...
        N,rows, blockSizeBytes);
fprintf('Offset address bits: %d\n', offset);
fprintf('Bits in the index bit: %d\n', index);
fprintf('Bits in the valid bit: %d\n', valid);
fprintf('Bits in the data block: %d\n', blockSizeBits);
fprintf('Bits in the tag: %d\n', tag);
fprintf('Bits in the LRU: %d\n', LRU);
fprintf('Total bits used: %d, bits remaining: %d\n', usedBits,unusedBits);
fprintf('Hit time: %d\n', hitTime);
fprintf('Miss time: %d\n\n', missTime);

% Initialize cache
sequenceLength = length(sequence);
cacheRowValidBit = zeros(rows*N,1);
cacheTag = zeros(rows*N,1);
cacheDataBlock = zeros(rows*N,blockSizeWords);
cachedRow = zeros(N*rows,1);
cacheHit = 0;
cacheMiss = 0;
needLRU = zeros(N,1);
loop = 4;

% Accessing cache
for i=1:loop
    cacheHit = 0;
    cacheMiss = 0;
    for i=1:sequenceLength
        isHit = false;
        currentAddress = sequence(i);
        currentSet = mod(floor(currentAddress/(2^offset)),rows);
        currentSet = currentSet+1;
        currentTag = floor(sequence(i)/(blockSizeBytes*rows));
        for k=1:N
           if(cacheRowValidBit(currentSet+k,1)==1 &&
currentTag==cacheTag(currentSet+k,1))
               for a=1:blockSizeWords
                    if(currentAddress==cacheDataBlock(currentSet+k,a))
                        isHit=true;
                        cacheHit = cacheHit+1;
                        fprintf('Accessing %d (set %d) (tag %d): hit from row
%d\n',...
                    currentAddress,currentSet-1,currentTag,k)
                     end
               end
           end
        end
        if(isHit==false)
            cacheMiss = cacheMiss+1;
```

```
            for a=1:N
                if(cachedRow(currentSet+a,1)==0)
                    if(a == rows)
                        needLRU(currentSet) = true;
                    end
                cacheRowValidBit(currentSet+a,1)=1;
                cacheTag(currentSet+a,1)=currentTag;
                cachedRow(currentSet+a,1)=1;
                index = currentAddress;
                for b=1:blockSizeWords
                    data(b)= index;
                    index = index+4;
                end
               cacheDataBlock(currentSet+a,:)=data;
               fprintf('Accessing %d (set %d)(tag %d): miss - cached to row
%d\n',...
                   currentAddress,currentSet-1,currentTag,a)
              break
           end
            end
            if(needLRU(currentSet) == 1)
                for b=1:N
               cachedRow(currentSet+b) = 0;
                end
                needLRU(currentSet) = 0;
            end
        end
    end
    cost = cacheHit*hitTime+cacheMiss*missTime;
fprintf('Cost in cycles for this repetition: %d\n',cost);
fprintf('Average CPI: %f\n',cost/sequenceLength);
end
```

Results:

```
2-way set associative cache with 2 rows and 16 bytes per row:
Offset address bits: 4
Bits in the index bit: 1
Bits in the valid bit: 1
Bits in the data block: 128
Bits in the tag: 11
Bits in the LRU: 2
Total bits used: 568, bits remaining: 332
Hit time: 1
Miss time: 36

Accessing 16 (set 1)(tag 0): miss - cached to row 1
Accessing 20 (set 1) (tag 0): hit from row 1
Accessing 24 (set 1) (tag 0): hit from row 1
Accessing 28 (set 1) (tag 0): hit from row 1
Accessing 32 (set 0)(tag 1): miss - cached to row 1
```

```
Accessing 36 (set 0) (tag 1): hit from row 1
Accessing 60 (set 1)(tag 1): miss - cached to row 2
Accessing 64 (set 0)(tag 2): miss - cached to row 2
Accessing 72 (set 0) (tag 2): hit from row 2
Accessing 76 (set 0) (tag 2): hit from row 2
Accessing 92 (set 1)(tag 2): miss - cached to row 1
Accessing 96 (set 0)(tag 3): miss - cached to row 1
Accessing 100 (set 0) (tag 3): hit from row 1
Accessing 104 (set 0) (tag 3): hit from row 1
Accessing 108 (set 0) (tag 3): hit from row 1
Accessing 112 (set 1)(tag 3): miss - cached to row 2
Accessing 120 (set 1) (tag 3): hit from row 2
Accessing 124 (set 1) (tag 3): hit from row 2
Accessing 128 (set 0)(tag 4): miss - cached to row 2
Accessing 144 (set 1)(tag 4): miss - cached to row 1
Accessing 148 (set 1) (tag 4): hit from row 1
Cost in cycles for this repetition: 336
Average CPI: 16.000000
Accessing 16 (set 1)(tag 0): miss - cached to row 2
Accessing 20 (set 1) (tag 0): hit from row 2
Accessing 24 (set 1) (tag 0): hit from row 2
Accessing 28 (set 1) (tag 0): hit from row 2
Accessing 32 (set 0)(tag 1): miss - cached to row 1
Accessing 36 (set 0) (tag 1): hit from row 1
Accessing 60 (set 1)(tag 1): miss - cached to row 1
Accessing 92 (set 1)(tag 2): miss - cached to row 2
Accessing 96 (set 0)(tag 3): miss - cached to row 2
Accessing 100 (set 0) (tag 3): hit from row 2
Accessing 104 (set 0) (tag 3): hit from row 2
Accessing 108 (set 0) (tag 3): hit from row 2
Accessing 112 (set 1)(tag 3): miss - cached to row 1
Accessing 120 (set 1) (tag 3): hit from row 1
Accessing 124 (set 1) (tag 3): hit from row 1
Accessing 128 (set 0)(tag 4): miss - cached to row 1
Accessing 144 (set 1)(tag 4): miss - cached to row 2
Accessing 148 (set 1) (tag 4): hit from row 2
Cost in cycles for this repetition: 406
Average CPI: 19.333333
Accessing 16 (set 1)(tag 0): miss - cached to row 1
Accessing 20 (set 1) (tag 0): hit from row 1
Accessing 24 (set 1) (tag 0): hit from row 1
Accessing 28 (set 1) (tag 0): hit from row 1
Accessing 60 (set 1)(tag 1): miss - cached to row 2
Accessing 64 (set 0)(tag 2): miss - cached to row 2
```

```
Accessing 72 (set 0) (tag 2): hit from row 2
Accessing 76 (set 0) (tag 2): hit from row 2
Accessing 92 (set 1)(tag 2): miss - cached to row 1
Accessing 96 (set 0)(tag 3): miss - cached to row 1
Accessing 100 (set 0) (tag 3): hit from row 1
Accessing 104 (set 0) (tag 3): hit from row 1
Accessing 108 (set 0) (tag 3): hit from row 1
Accessing 112 (set 1)(tag 3): miss - cached to row 2
Accessing 120 (set 1) (tag 3): hit from row 2
Accessing 124 (set 1) (tag 3): hit from row 2
Accessing 128 (set 0)(tag 4): miss - cached to row 2
Accessing 144 (set 1)(tag 4): miss - cached to row 1
Accessing 148 (set 1) (tag 4): hit from row 1
Cost in cycles for this repetition: 371
Average CPI: 17.666667
Accessing 16 (set 1)(tag 0): miss - cached to row 2
Accessing 20 (set 1) (tag 0): hit from row 2
Accessing 24 (set 1) (tag 0): hit from row 2
Accessing 28 (set 1) (tag 0): hit from row 2
Accessing 32 (set 0)(tag 1): miss - cached to row 1
Accessing 36 (set 0) (tag 1): hit from row 1
Accessing 60 (set 1)(tag 1): miss - cached to row 1
Accessing 92 (set 1)(tag 2): miss - cached to row 2
Accessing 96 (set 0)(tag 3): miss - cached to row 2
Accessing 100 (set 0) (tag 3): hit from row 2
Accessing 104 (set 0) (tag 3): hit from row 2
Accessing 108 (set 0) (tag 3): hit from row 2
Accessing 112 (set 1)(tag 3): miss - cached to row 1
Accessing 120 (set 1) (tag 3): hit from row 1
Accessing 124 (set 1) (tag 3): hit from row 1
Accessing 128 (set 0)(tag 4): miss - cached to row 1
Accessing 144 (set 1)(tag 4): miss - cached to row 2
Accessing 148 (set 1) (tag 4): hit from row 2
Cost in cycles for this repetition: 406
Average CPI: 19.333333
```

The ideal block size is 128 bits with 2 sets. The reason why I pick a set size of 2 so that more words per block can be increased. More words per block will increase spatial locality and increase hit rate.

CONCULSION:
To improve cache performance for this sequence of address it is important to use spatial locality. Since instruction are byte addressed it is more likely that the address close to the reference address is going to be referenced soon. So increase the words per block will improve the spatial locality in the cache. This means that there is a higher probability to hit in the cache and will lower our average CPI.