CprE 381 Homework 9

[Note: This homework is meant to help you form insights regarding the principles behind caches.]

1. Principle of Locality
    a. Write a valid MIPS assembly program that executes at least 20 instructions and demonstrates spatial locality in instruction fetching, but not data accesses. Explain this locality in the assembly comments.

```
#instructions are back to back no jumping so there is very high instruction locality
lui $t0, 0x1001
ori $t0, $t0, 0
lw $t1, 0($t0)
#This program is taking large jumps in data accesses so it does not have high data accesss locality
#A program that would would be a loop that keeps getting the next element in an array
lui $t0, 0x1005
ori $t0, $t0, 0
lw $t2, 0($t0)
addu $t3, $t1, $t2

lui $t0, 0x1008
ori $t0, $t0, 0
lw $t3, 0($t0)
lui $t0, 0x1010
ori $t0, $t0, 0
lw $t4, 0($t0)
addu $t5, $t3, $t4

addu $t6, $t3, $t5
sll $t6, $t6, 5
srl $t6, $t6, 6
lw $t7, 0x10010000($zero)
addu $t8, $t6, $t7

halt
```

b. Write a valid MIPS assembly program that executes at least 20 instructions and demonstrates temporal locality in data accesses, but not instruction fetching. Explain this locality in the assembly comments.

```
#Low instruction fetching locality means that there is a lot of jumps and branches
#High Data access locality means that there are back to back data access like an array
#Idea: Just jump around like crazy but keep accessing array +1
# MIPS assembly program demonstrating spatial locality in instruction fetching but not data accesses
   .data
array:  .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
   .text
   .globl main
main:
addiu $t0, $zero, 0
lui $t1, 0x1001
ori $t1, $t1, 0
lw $t2, 0($t1)
addiu $t3, $zero, 0
addu $t3, $t3, $t2
j jump1
nop
nop
nop

jump1:
lw $t2, 4($t1)
addu $t3, $t3, $t2
j jump2
nop
nop
nop

jump2:
lw $t2, 8($t1)
addu $t3, $t3, $t2
j jump3
nop
nop
nop
```

```
jump3:
lw $t2, 16($t1)
addu $t3, $t3, $t2
halt
```

c. Spend some time looking at open-source programs on Github.com. Find a piece of a C or C++ program on github that appears to display a significant amount of data locality. Provide the html browsable file URL and line numbers of the example. Justify why these lines demonstrate data locality. *[Note that since this is real code, you may need to reference multiple files to demonstrate locality even in a single example.]*

```
int G_CmdChecksum (ticcmd_t* cmd)
{
  int                 i;
  int                 sum = 0;

  for (i=0 ; i< sizeof(*cmd)/4 - 1 ; i++)
    sum += ((int *)cmd)[i];

  return sum;
}
```

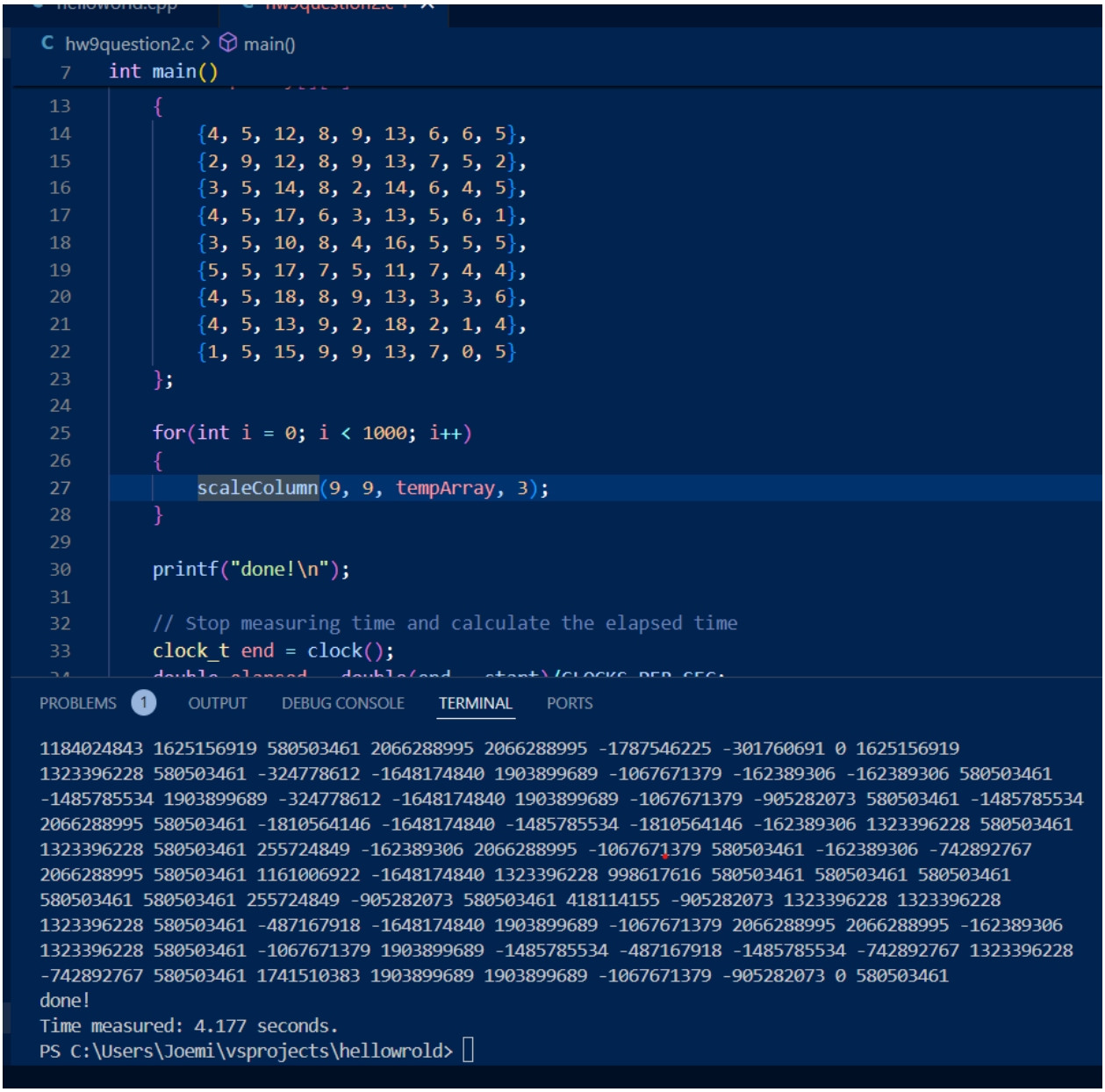
g_game.c line 219

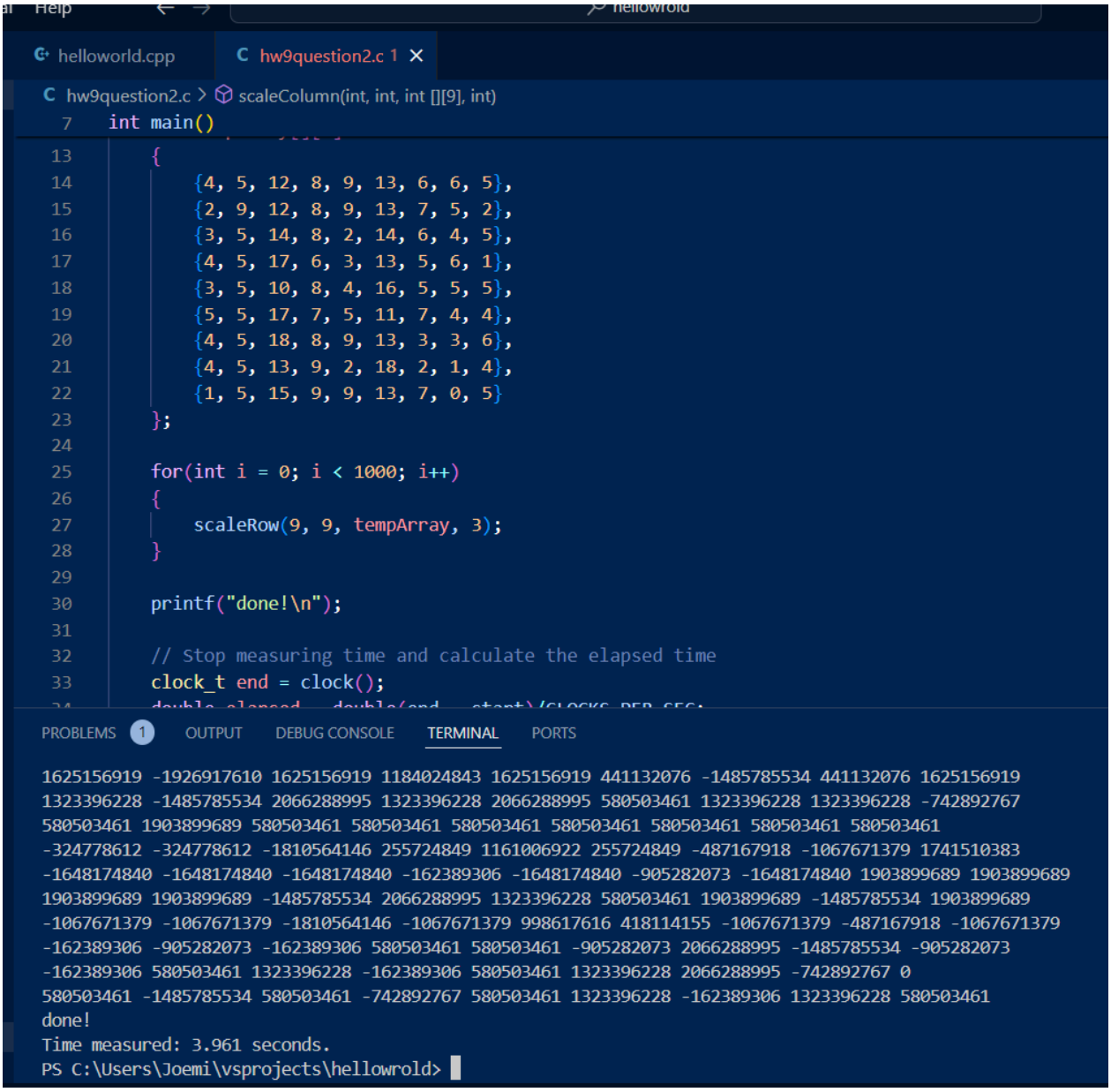
This is from the source code of the original Doom. I am not sure what this is doing in the code but I choose it because it looks like it is going through and incrementing i by 1 and then accessing an array with i. This would be data locality as it is accessing data in close proximity over and over again.

2. Breaking Locality

Complete the following C function that scales each element of a 2D array by a scalar. First, complete it in row-major ordering (https://en.wikipedia.org/wiki/Row-_and_column-major_order). Second, complete it in column-major ordering. Which is faster on your computer (report the time each takes to execute 1000 calls to scale and what processor model you have)? Why is it faster? Please use the C template provided with this homework.

```
void scale(int n, int m, int array[n][m], int scale);
```


C hw9question2.c > main()
int main()
{
{4, 5, 12, 8, 9, 13, 6, 6, 5},
{2, 9, 12, 8, 9, 13, 7, 5, 2},
{3, 5, 14, 8, 2, 14, 6, 4, 5},
{4, 5, 17, 6, 3, 13, 5, 6, 1},
{3, 5, 10, 8, 4, 16, 5, 5, 5},
{5, 5, 17, 7, 5, 11, 7, 4, 4},
{4, 5, 18, 8, 9, 13, 3, 3, 6},
{4, 5, 13, 9, 2, 18, 2, 1, 4},
{1, 5, 15, 9, 9, 13, 7, 0, 5}
};
for(int i = 0; i < 1000; i++)
{
scaleColumn(9, 9, tempArray, 3);
}
printf("done!\n");
// Stop measuring time and calculate the elapsed time
clock_t end = clock();
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
1184024843 1625156919 580503461 2066288995 2066288995 -1787546225 -301760691 0 1625156919
1323396228 580503461 -324778612 -1648174840 1903899689 -1067671379 -162389306 -162389306 580503461
-1485785534 1903899689 -324778612 -1648174840 1903899689 -1067671379 -905282073 580503461 -1485785534
2066288995 580503461 -1810564146 -1648174840 -1485785534 -1810564146 -162389306 1323396228 580503461
1323396228 580503461 255724849 -162389306 2066288995 -1067671379 580503461 -162389306 -742892767
2066288995 580503461 1161006922 -1648174840 1323396228 998617616 580503461 580503461 580503461
580503461 580503461 255724849 -905282073 580503461 418114155 -905282073 1323396228 1323396228
1323396228 580503461 -487167918 -1648174840 1903899689 -1067671379 2066288995 2066288995 -162389306
1323396228 580503461 -1067671379 1903899689 -1485785534 -487167918 -1485785534 -742892767 1323396228
-742892767 580503461 1741510383 1903899689 1903899689 -1067671379 -905282073 0 580503461
done!
Time measured: 4.177 seconds.
PS C:\Users\Joemi\vsprojects\hellowrold>



This was with printf statements in each function call. When I removed them it went down to 0.001 seconds. It seems that the row function is faster. It had the lowest time out of all of the runs I did. And the column had the longest time I recorded. I think this is due to locality. For a 2D array it is an array of arrays. When you go across a row you are grabbing elements from the same array and then when you increment to go to the next row you start grabbing elements in the next array. For the column way you are switching between different arrays and grabbing their corresponding element.

3. Direct-Mapped Cache Configuration and Simulation

a. ZyBooks 5.19.2.a

0x03, 0xb4, 0x2b, 0x02, 0xbf, 0x58, 0xbe, 0x0e, 0xb5, 0x2c,0xba, 0xfd

(a)

For each of these references, identify the binary address, the tag, and the index given a direct-mapped cache with 16 one-word blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

Address will be the least significant hex value, the tag will be the most sig hex value

	Tag	Address	Hit/miss
0x03	0x0	0x3	miss
0xb4	0xb	0x4	miss
0x2b	0x2	0xb	miss
0x02	0x0	0x2	miss
0xbf	0xb	0xf	miss
0x58	0x5	0x8	miss
0xbe	0xb	0xe	miss
0x0e	0x0	0xe	miss
0xb5	0xb	0x5	miss
0x2c	0x2	0xc	miss
0xba	0xb	0xa	miss
0xfd	0xf	0xd	miss

b. ZyBooks 5.19.2.b

(b)

For each of these references, identify the binary address, the tag, and the index given a direct-mapped cache with two-word blocks and a total size of 8 blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

32 mod 8 for address

	Tag	Address	Hit/miss
0x03	0x00	011	miss
0xb4	0x16	100	miss
0x2b	0x05	011	miss
0x02	0x00	010	miss
0xbf	0x17	111	miss
0x58	0x0b	000	miss

0xbe	0x17	110	miss
0x0e	0x01	110	miss
0xb5	0x16	101	miss
0x2c	0x05	100	miss
0xba	0x17	010	miss
0xfd	0x1f	101	miss

c. ZyBooks 5.19.2.c

(c)
You are asked to optimize a cache design for the given references. There are three direct-mapped cache designs possible, all with a total of 8 words of data:

C1 has 1-word blocks,
C2 has 2-word blocks, and
C3 has 4-word blocks.
In terms of miss rate, which cache design is the best? If the miss stall time is 25 cycles, and C1 has an access time of 2 cycles, C2 takes 3 cycles, and C3 takes 5 cycles, which is the best cache design?

C3 will be the best because it will have the least miss rate.

AMAT = Time for a hit + Miss rate x Miss penalty

AMATC1 = 2 cycles + MissrateC1 x 25 cycles
AMATC2 = 3 cycles + MissrateC2 x 25 cycles
AMATC3 = 5 cycles + MissrateC3 x 25 cycles

Depends on the miss rate for each of them

d. (From P&H 5th Edition) There are many different design parameters that are important to a cache's overall performance. Below are listed parameters for different direct-mapped cache designs.
Cache Data Size: 32 KiB
Cache Block Size: 2 words
Cache Access Time: 1 cycle

The formula shown in Section 5.3 shows the typical method to index a direct mapped cache, specifically (Block address) modulo (Number of blocks in the cache). Assuming a 32-bit address and 1024 blocks in the cache, consider a different indexing function, specifically (Block address[31 :27] XOR Block address[26:22]). Is it possible to use this to index a direct-mapped cache? If so,

explain why and discuss any changes that might need to be made to the cache. If it is not possible, explain why.

You are not going to be using all of the blocks like you could be using with mod

log2(1024) = 10 bit index bits for addressing

31:27 XOR 26:22 yields 5 bits for index addressing

It would be possible though. To implement it you would need to add multi gate xors in order to calculate the xor value.