! This class has been made inactive. No posts will be allowed until an instructor reactivates the class.

note @2126 220 views

Actions

[Past Exams] 2019

You can find the past exams here: https://cs61c.org/sp22/resources/exams/

When posting questions, please reference the semester, exam, and question in this format so it's easier for students and staff to search for similar questions:

Semester-Exam-Question Number

For example: **SP19-Final-Q1**, or **FA19-MT2-Q3**

- Here's a video walkthrough by Daniel for the SP19 Final: https://www.youtube.com/watch?v=8DiN5Hu9x24&list=PLDoI-XvXO0apuEacxuUrUaBq2YDuKYPtV&index=2 (handout and timestamps in comments)
- Here's a video walkthrough for the SU19 Final made by Sunay.
  - Q1 Potpourri: https://youtu.be/FY5dAMrXvxo
  - Q2 FSM: https://youtu.be/gmHbw6LSeSw
  - Q3 C Coding: https://youtu.be/v4B1WTs5UNU
  - Q4 RISC-V: https://youtu.be/2VHjG-gy9Dk
  - Q5 Data-Level Parallelism: https://youtu.be/oG9Rrzmi0M4
  - Q6 RAID and ECC: https://youtu.be/rfCNTIzNZ2M
  - Q7 Caches: https://youtu.be/xojc8YZaO3Q
  - Q8 Spark: https://youtu.be/A37BFXRXmm0
  - Q9 Datapath: https://youtu.be/q-T4N3hBhUM
  - Q10 Digital Logic: https://youtu.be/3RI36lsDSg4
  - Q11 Virtual Memory: https://youtu.be/5_2fKsK4I34

exam exam/final

good note | 0

Updated 5 months ago by Jerry Xu and Peyrin Kao

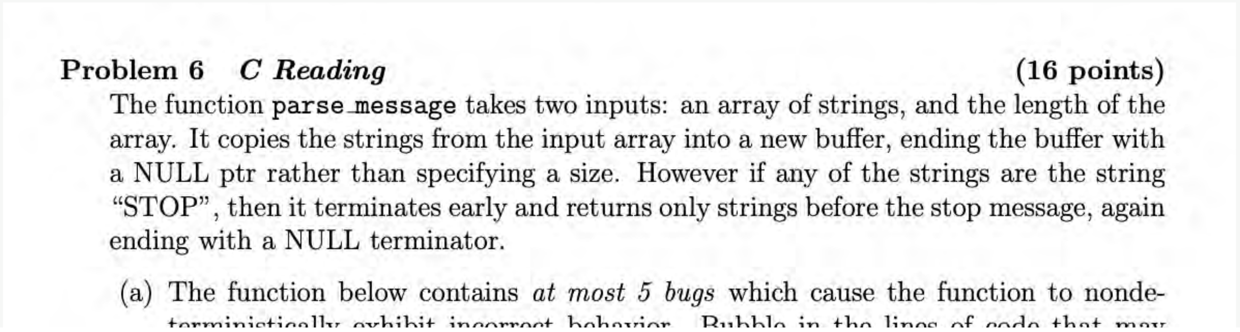
followup discussions, *for lingering questions and comments*

[✓ Resolved] Unresolved @2126_f1

This was marked a duplicate to the question/note above by Peyrin Kao 5 months ago

Anonymous Mouse 5 months ago

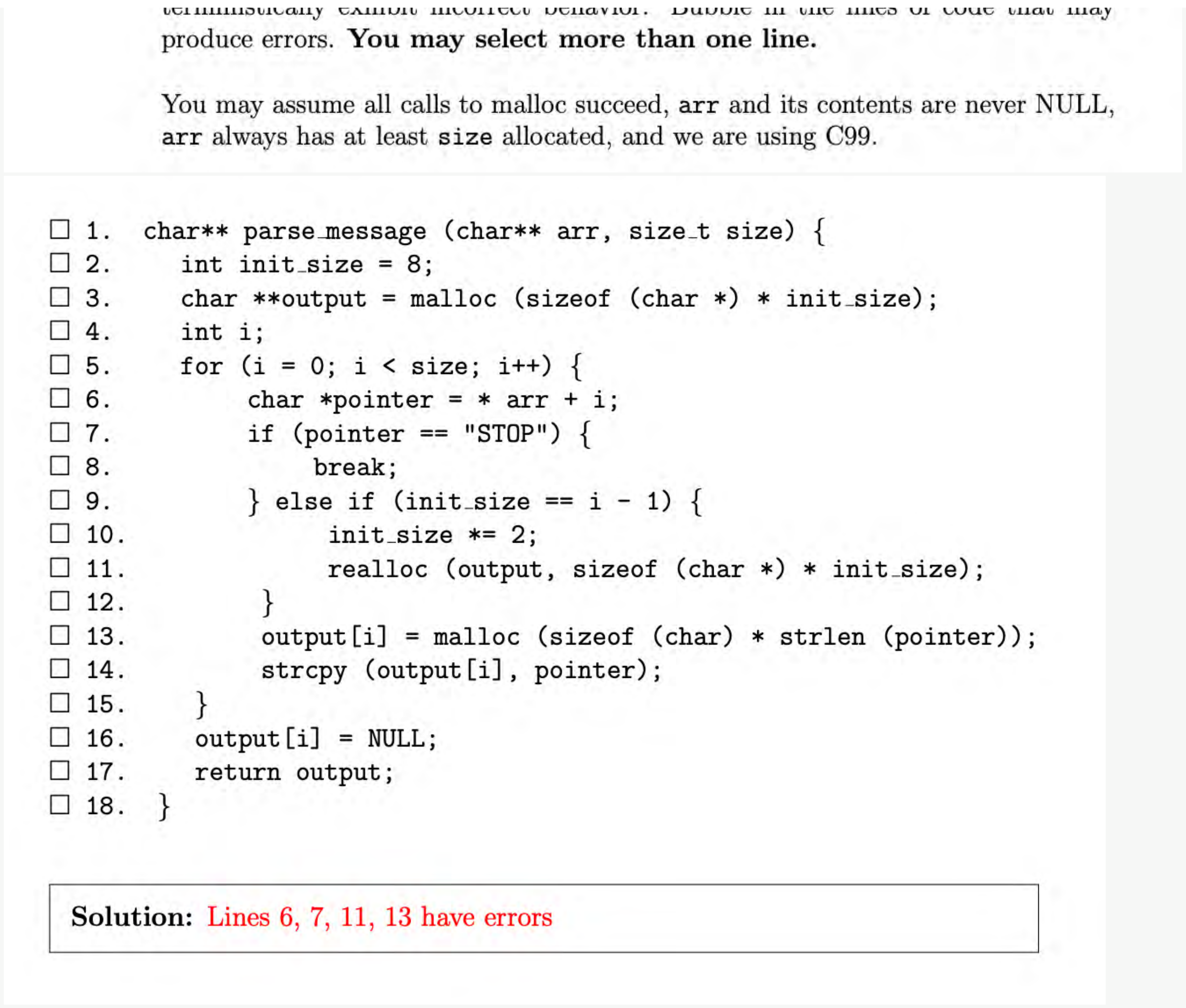
Spring 2019 final Q6



Problem 6 ***C Reading*** **(16 points)**

The function `parse_message` takes two inputs: an array of strings, and the length of the array. It copies the strings from the input array into a new buffer, ending the buffer with a NULL ptr rather than specifying a size. However if any of the strings are the string "STOP", then it terminates early and returns only strings before the stop message, again ending with a NULL terminator.

(a) The function below contains *at most 5 bugs* which cause the function to nondeterministically exhibit incorrect behavior. Bubble in the lines of code that may


terministically exhibit incorrect behavior. Bubble in the lines of code that may produce errors. **You may select more than one line.**

You may assume all calls to malloc succeed, **arr** and its contents are never NULL, **arr** always has at least **size** allocated, and we are using C99.

```
☐ 1.  char** parse_message (char** arr, size_t size) {
☐ 2.    int init_size = 8;
☐ 3.    char **output = malloc (sizeof (char *) * init_size);
☐ 4.    int i;
☐ 5.    for (i = 0; i < size; i++) {
☐ 6.        char *pointer = * arr + i;
☐ 7.        if (pointer == "STOP") {
☐ 8.            break;
☐ 9.        } else if (init_size == i - 1) {
☐ 10.            init_size *= 2;
☐ 11.            realloc (output, sizeof (char *) * init_size);
☐ 12.        }
☐ 13.        output[i] = malloc (sizeof (char) * strlen (pointer));
☐ 14.        strcpy (output[i], pointer);
☐ 15.    }
☐ 16.    output[i] = NULL;
☐ 17.    return output;
☐ 18. }
```

Solution: Lines 6, 7, 11, 13 have errors

I don't see anything wrong with 11. Yet, I do see something wrong with line 9. If we check to double the size of the string array when (i - 1 == init_size), then first time that the size of the array will be doubled is when i = 9, which is a problem since we first initialized our string array to be of size 8, which means that the previous iteration at i = 8 would have accessed memory that was not malloced. Is there something that I'm missing?

helpful! | 0

Peyrin Kao 5 months ago

Here's what I answered last time I was asked this, though I'm also not 100% sure about this one...

Line 6: `*arr + i` should be `*(arr+i)` . The dereference operator takes precedence over addition in C.

Line 7: Left-hand side is a pointer, right-hand side is a string. Probably need to use `strcmp` or something here.

Line 11: I think you have to actually assign the return value of `realloc` back into the pointer, i.e.

`c (output, sizeof (char *) * initsize);`

Line 13: Need to allocate 1 extra byte for the null terminator.

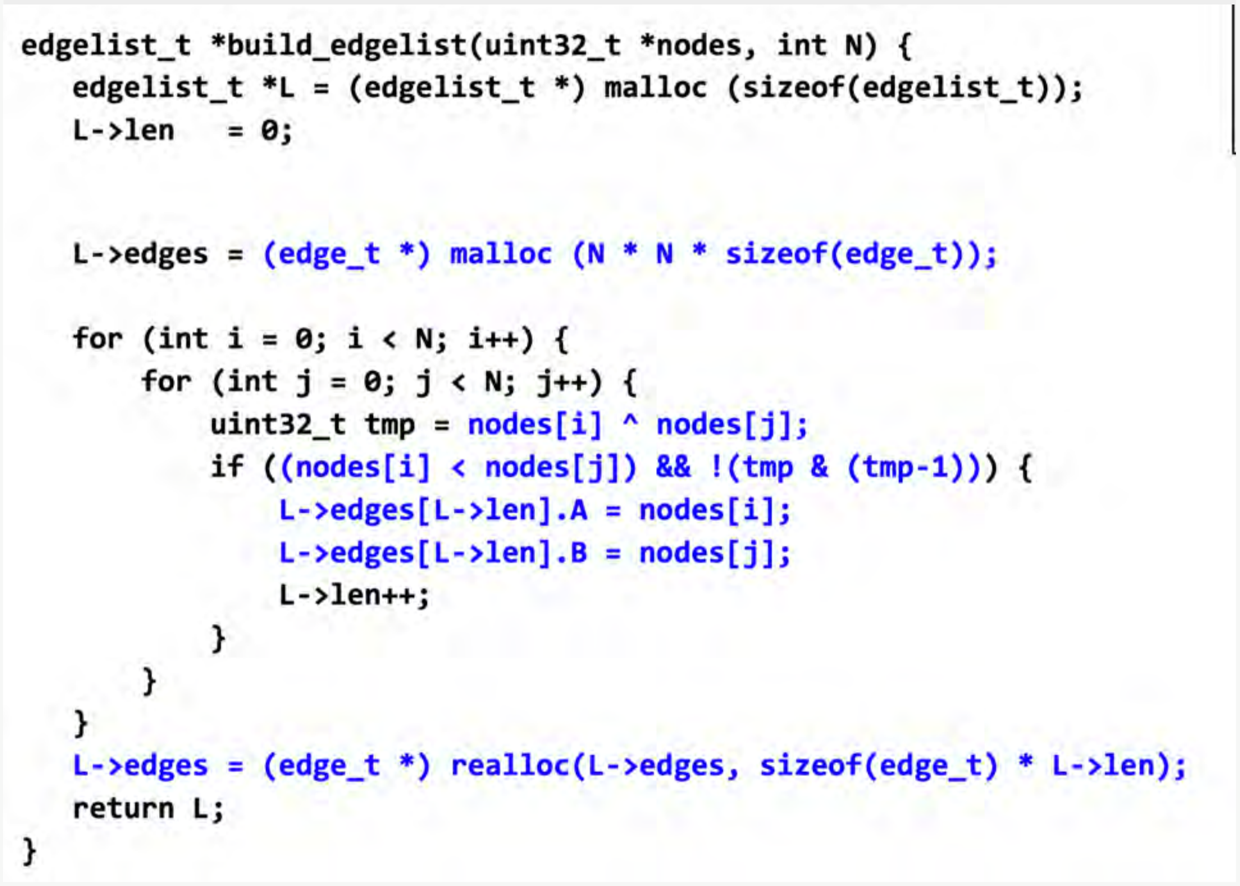
good comment | 0

[✓ Resolved] Unresolved @2126_f2

This was marked a duplicate to the question/note above by Peyrin Kao 5 months ago

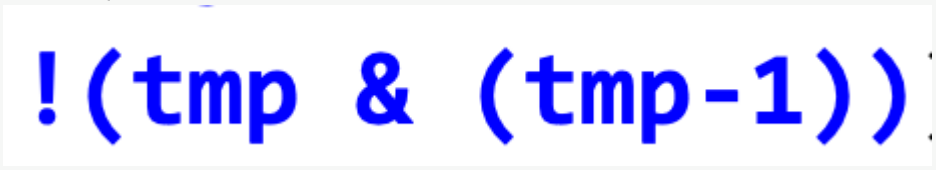
Kendrick Sharpe 5 months ago

[Fall '19] Final exam question



```
edgelist_t *build_edgelist(uint32_t *nodes, int N) {
    edgelist_t *L = (edgelist_t *) malloc (sizeof(edgelist_t));
    L->len   = 0;


    L->edges = (edge_t *) malloc (N * N * sizeof(edge_t));

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            uint32_t tmp = nodes[i] ^ nodes[j];
            if ((nodes[i] < nodes[j]) && !(tmp & (tmp-1))) {
                L->edges[L->len].A = nodes[i];
                L->edges[L->len].B = nodes[j];
                L->len++;
            }
        }
    }
    L->edges = (edge_t *) realloc(L->edges, sizeof(edge_t) * L->len);
    return L;
}
```

For this question, what does



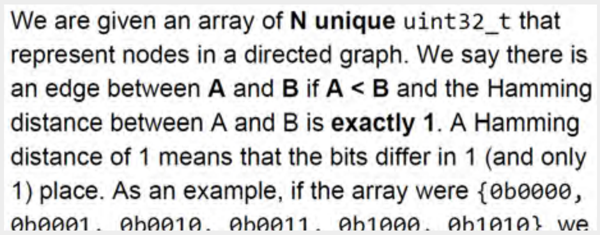
```
!(tmp & (tmp-1))
```

do?

helpful! | 0

Adelson Chua 5 months ago

That checks for the hamming distance of 1.



We are given an array of **N unique** `uint32_t` that represent nodes in a directed graph. We say there is an edge between **A** and **B** if **A < B** and the Hamming distance between A and B is **exactly 1**. A Hamming distance of 1 means that the bits differ in 1 (and only 1) place. As an example, if the array were `{0b0000, 0b0001, 0b0010, 0b0011, 0b1000, 0b1010}` we

The nodes[i] ^ nodes[j]; line before the if statement compares the nodes array bit per bit through XOR. If there is only 1 bit of difference, the XOR result will always be a power of 2 (only 1 bit is 1).
!(tmp & (tmp-1)) only becomes true if tmp is a power of 2. Try it out in binary: 16 = 010000. 16-1 = 15 = 001111. 010000 & 001111 = 00000.

To be honest, this is a weird, hacky (somewhat abstract) way of checking if something is a power of 2.

Addendum: This verifies my claim: https://stackoverflow.com/questions/600293/how-to-check-if-a-number-is-a-power-of-2

good comment | 0

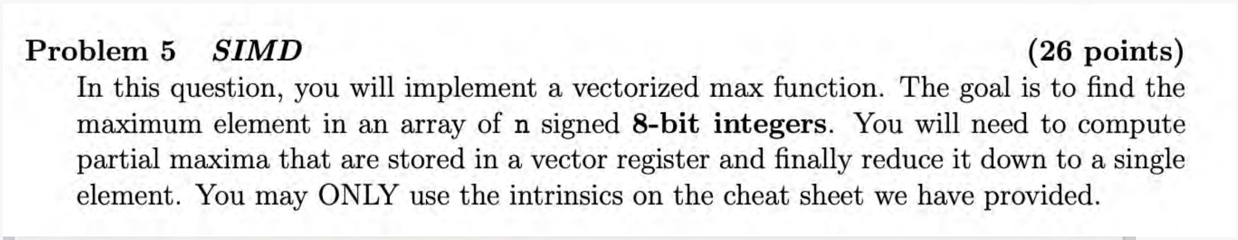
Reply to this followup discussion

[✓ Resolved] Unresolved @2126_f3

This was marked a duplicate to the question/note above by Jerry Xu 5 months ago

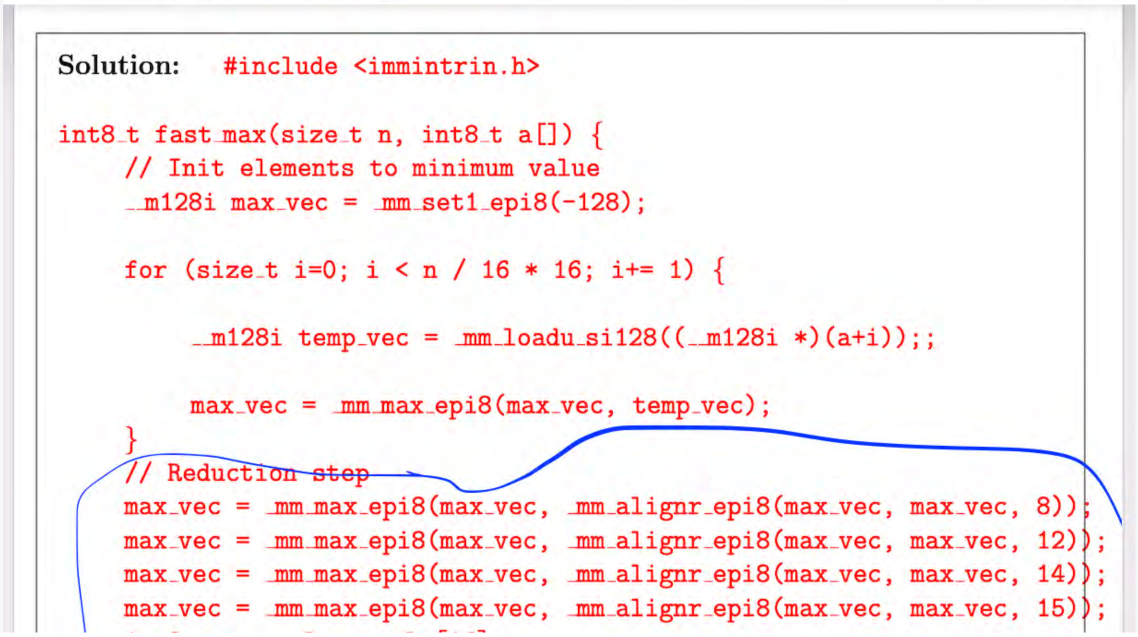
Anonymous Mouse 5 months ago

Sp 2019 Final q5

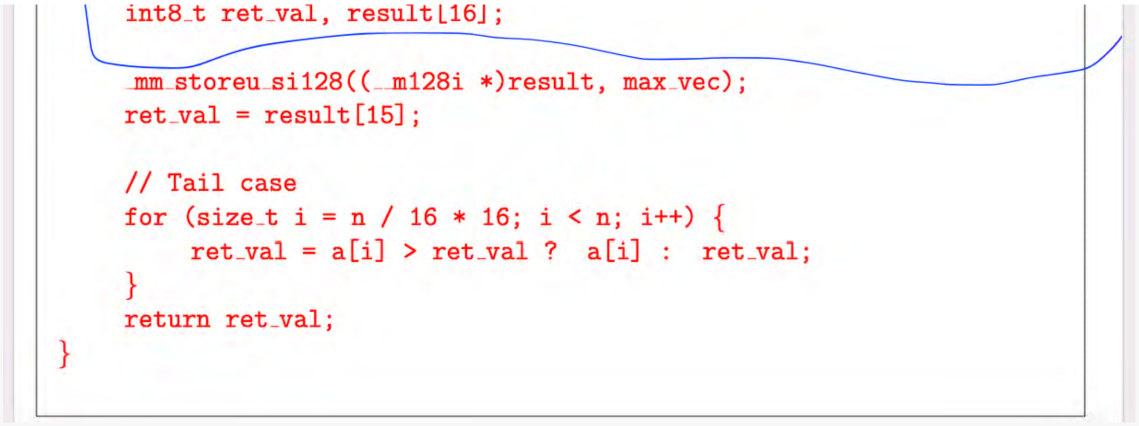


Problem 5 ***SIMD*** **(26 points)**

In this question, you will implement a vectorized max function. The goal is to find the maximum element in an array of n signed **8-bit integers**. You will need to compute partial maxima that are stored in a vector register and finally reduce it down to a single element. You may ONLY use the intrinsics on the cheat sheet we have provided.



Solution:
```
#include <immintrin.h>

int8_t fast_max(size_t n, int8_t a[]) {
    // Init elements to minimum value
    __m128i max_vec = _mm_set1_epi8(-128);

    for (size_t i=0; i < n / 16 * 16; i+= 1) {

        __m128i temp_vec = _mm_loadu_si128((__m128i *)(a+i));;

        max_vec = _mm_max_epi8(max_vec, temp_vec);
    }
    // Reduction step
    max_vec = _mm_max_epi8(max_vec, _mm_alignr_epi8(max_vec, max_vec, 8));
    max_vec = _mm_max_epi8(max_vec, _mm_alignr_epi8(max_vec, max_vec, 12));
    max_vec = _mm_max_epi8(max_vec, _mm_alignr_epi8(max_vec, max_vec, 14));
    max_vec = _mm_max_epi8(max_vec, _mm_alignr_epi8(max_vec, max_vec, 15));
```



```
    int8_t ret_val, result[16];
    _mm_storeu_si128((__m128i *)result, max_vec);
    ret_val = result[15];

    // Tail case
    for (size_t i = n / 16 * 16; i < n; i++) {
        ret_val = a[i] > ret_val ?  a[i] :  ret_val;
    }
    return ret_val;
}
```

I'm having trouble understanding the part I circled in blue. I know what alignr does but I'm not sure how this gets us the max value. Could someone explain?

helpful! | 0

Adelson Chua 5 months ago

After the initial loop, you get max_vec which contains 16, 8-bit ints which were found to be the maximum so far.

So you have a vector with 16 elements. You want to reduce those down to 1 element which will be the maximum for the entire array.

_mm_alignr_epi8(max_vec, max_vec, 8) effectively shifts the 16-element vector by 8 elements to the right. This allows you to compare the upper 8 vectors with the lower 8 vectors. Now you only have 8 effective vectors to get the max from.

_mm_alignr_epi8(max_vec, max_vec, 12) will allow you to compare the upper 4 vectors to the lower 4 vectors (remember, you only have an effective 8 element vector at this point). Now you only have 4 effective vectors to get the max from.

_mm_alignr_epi8(max_vec, max_vec, 14) will allow you to compare the upper 2 vectors to the lower 2 vectors (remember, you only have an effective 4 element vector at this point). Now you only have 2 effective vectors to get the max from.

_mm_alignr_epi8(max_vec, max_vec, 15) will allow you to compare the upper 1 vector to the lower 1 vector (remember, you only have an effective 2 element vector at this point). Now you only have 1 effective vector to get the max from.

The _mm_storeu_si128 basically just extracts that 1 vector, which is now the max of the entire array.

This probably needs some drawing to fully understand what's happening. I don't have time for that. Hopefully, the text version of my explanation leads you to the right train of thought.

You could try drawing it on your own: draw the 16 array elements and see how they get 'reduced' after doing the alignr and max functions.

good comment | 0

Anonymous Mouse 5 months ago

I see thank you for the explanation!

helpful! | 0

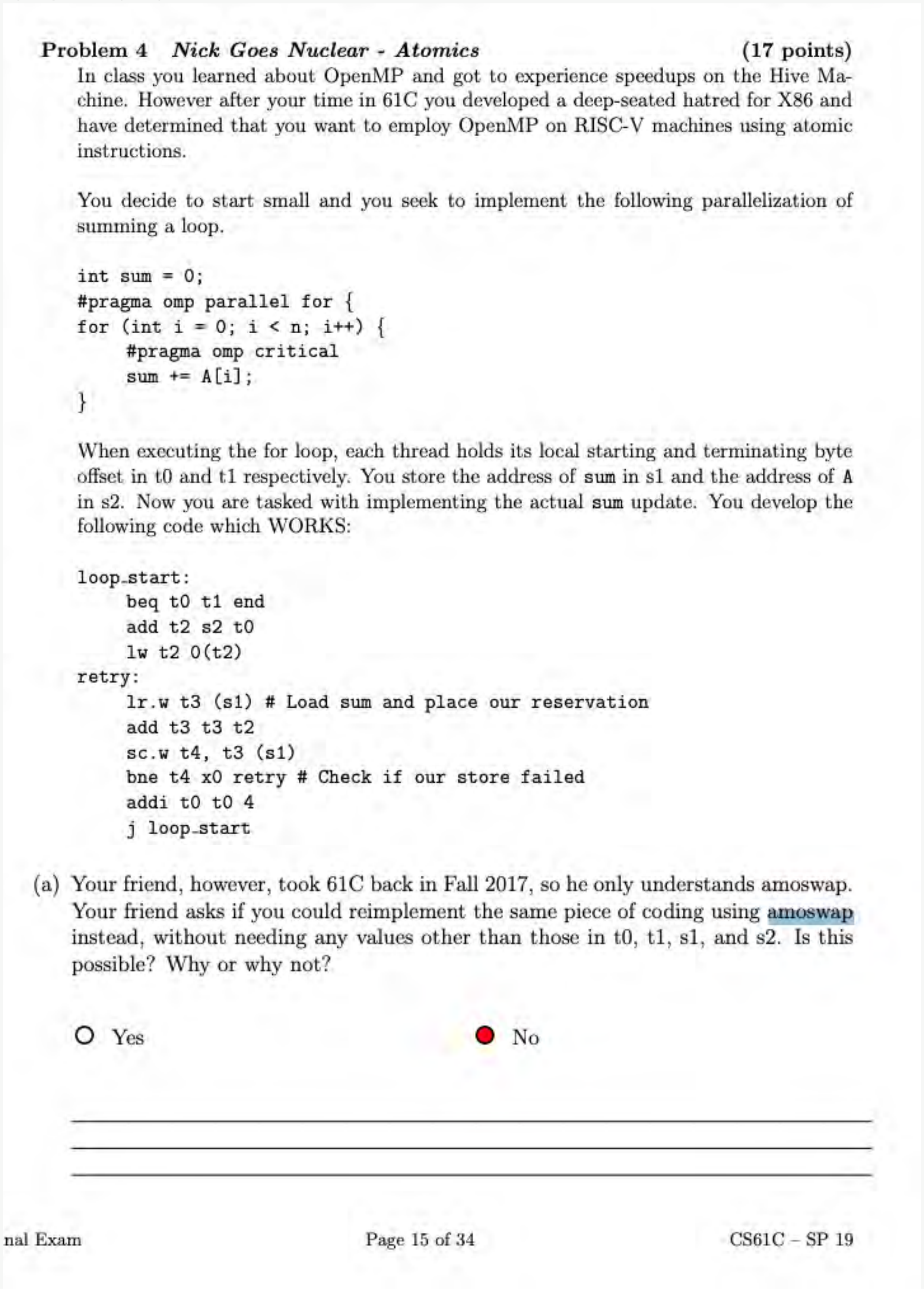
Reply to this followup discussion

Resolved Unresolved @2126_f4

This was marked a duplicate to the question/note above by Jerry Xu 5 months ago

Jerry Xu 5 months ago

Spring 2019 q4a question



Problem 4 ***Nick Goes Nuclear - Atomics*** **(17 points)**

In class you learned about OpenMP and got to experience speedups on the Hive Machine. However after your time in 61C you developed a deep-seated hatred for X86 and have determined that you want to employ OpenMP on RISC-V machines using atomic instructions.

You decide to start small and you seek to implement the following parallelization of summing a loop.

```
int sum = 0;
#pragma omp parallel for {
for (int i = 0; i < n; i++) {
     #pragma omp critical
     sum += A[i];
}
```

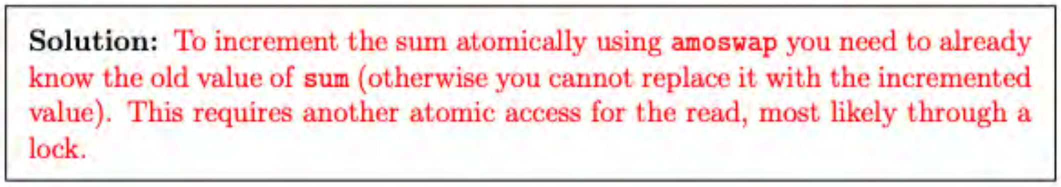
When executing the for loop, each thread holds its local starting and terminating byte offset in t0 and t1 respectively. You store the address of `sum` in s1 and the address of `A` in s2. Now you are tasked with implementing the actual `sum` update. You develop the following code which WORKS:

```
loop_start:
     beq t0 t1 end
     add t2 s2 t0
     lw t2 0(t2)
retry:
     lr.w t3 (s1) # Load sum and place our reservation
     add t3 t3 t2
     sc.w t4, t3 (s1)
     bne t4 x0 retry # Check if our store failed
     addi t0 t0 4
     j loop_start
```

(a) Your friend, however, took 61C back in Fall 2017, so he only understands amoswap. Your friend asks if you could reimplement the same piece of coding using amoswap instead, without needing any values other than those in t0, t1, s1, and s2. Is this possible? Why or why not?

Yes [✓ No]

nal Exam Page 15 of 34 CS61C – SP 19



Solution: To increment the sum atomically using **amoswap** you need to already know the old value of **sum** (otherwise you cannot replace it with the incremented value). This requires another atomic access for the read, most likely through a lock.

If this were amoadd instead of amoswap, we could do it right?

I just want to make sure my understand is sound. In order to do amoswap to swap the old sum with the updated incremented sum, we would have to know the old value of the sum in the first place. In order to do that we need a lock to get the value of sum and make sure that nothing changes it before we amoswap it with the new value?

good comment | 0

Adelson Chua 5 months ago
amoadd works... I think? I didn't think much about where the amoadd instruction will go in though.

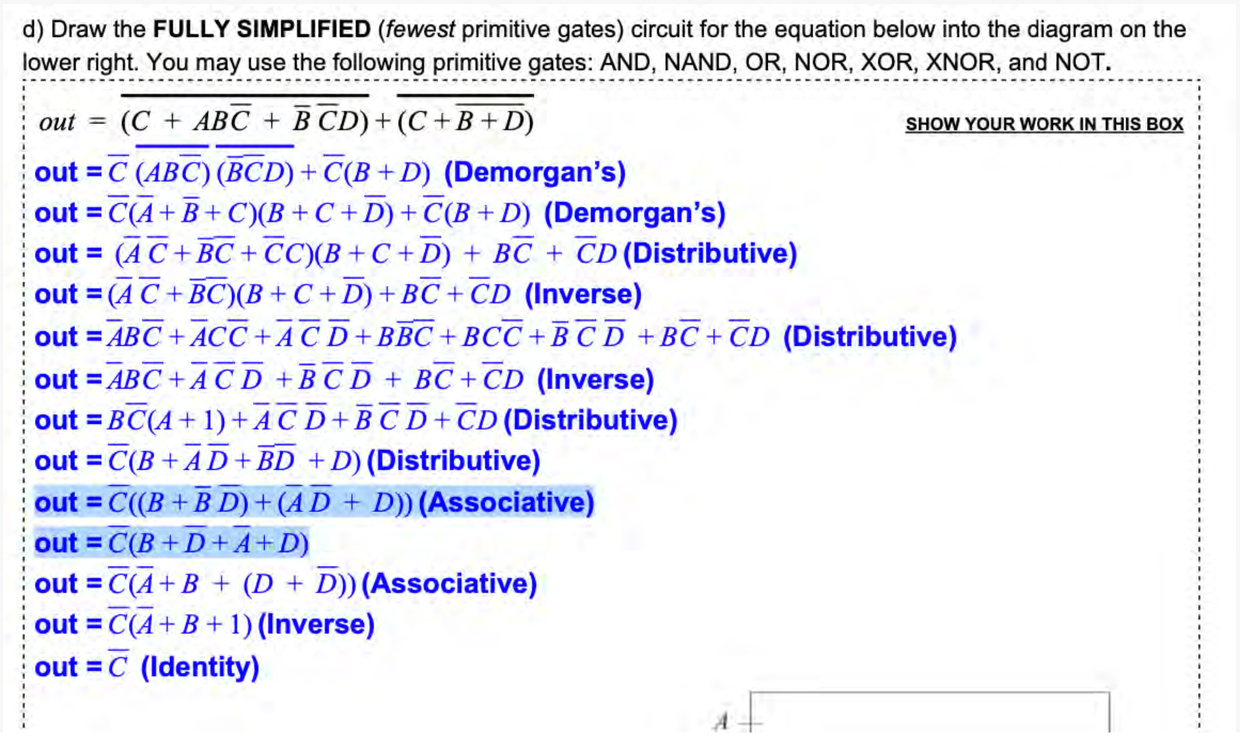
Your reasoning is also correct.

good comment | 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2126_f5

Anonymous Gear 5 months ago
[fa19-final-q5d] How to go from the first highlight to the second?



d) Draw the **FULLY SIMPLIFIED** (*fewest* primitive gates) circuit for the equation below into the diagram on the lower right. You may use the following primitive gates: AND, NAND, OR, NOR, XOR, XNOR, and NOT.

$$out = \overline{(C + AB\overline{C} + \overline{B}\,\overline{C}D)} + \overline{(C + \overline{B+D})}$$

SHOW YOUR WORK IN THIS BOX

out = $\overline{C}\,(\overline{AB\overline{C}})\,(\overline{\overline{B}\,\overline{C}D}) + \overline{C}(B+D)$ **(Demorgan's)**

out = $\overline{C}(\overline{A}+\overline{B}+C)(B+C+\overline{D}) + \overline{C}(B+D)$ **(Demorgan's)**

out = $(\overline{A}\,\overline{C} + \overline{B}\,\overline{C} + \overline{C}C)(B+C+\overline{D}) + B\overline{C} + \overline{C}D$ **(Distributive)**

out = $(\overline{A}\,\overline{C} + \overline{B}\,\overline{C})(B+C+\overline{D}) + B\overline{C} + \overline{C}D$ **(Inverse)**

out = $\overline{A}B\overline{C} + \overline{A}C\overline{C} + \overline{A}\,\overline{C}\,\overline{D} + B\overline{B}\,\overline{C} + BC\overline{C} + \overline{B}\,\overline{C}\,\overline{D} + B\overline{C} + \overline{C}D$ **(Distributive)**

out = $\overline{A}B\overline{C} + \overline{A}\,\overline{C}\,\overline{D} + \overline{B}\,\overline{C}\,\overline{D} + B\overline{C} + \overline{C}D$ **(Inverse)**

out = $B\overline{C}(A+1) + \overline{A}\,\overline{C}\,\overline{D} + \overline{B}\,\overline{C}\,\overline{D} + \overline{C}D$ **(Distributive)**

out = $\overline{C}(B + \overline{A}\,\overline{D} + \overline{B}\,\overline{D} + D)$ **(Distributive)**

out = $\overline{C}((B + \overline{B}\,\overline{D}) + (\overline{A}\,\overline{D} + D))$ **(Associative)**

out = $\overline{C}(B + \overline{D} + \overline{A} + D)$

out = $\overline{C}(\overline{A} + B + (D + \overline{D}))$ **(Associative)**

out = $\overline{C}(\overline{A} + B + 1)$ **(Inverse)**

out = $\overline{C}$ **(Identity)**

A

helpful! | 0

Adelson Chua 5 months ago
B + B'D' => B + D' (uniting theorem, Lecture 10, slide 33)

A'D' + D => A' + D (same thing)

good comment | 0

Reply to this followup discussion

[● Resolved] ○ Unresolved @2126_f6

Anonymous Helix 5 months ago
FA9-Final-Q9d

For a physically indexed and tagged cache, we don't flush the cache, but do we flush the TLB? And what about for a virtually indexed and tagged cache? Virtually indexed, physically tagged cache? Also, I don't understand why we only clear certain bits. Thank you!

helpful! | 0

Adelson Chua 5 months ago
Clearing valid bits basically flushes it. The dirty bit will not matter when it is flushed, which is why it is 'No'.

For the following explanations, refer to lecture 23, starting slide 52.

For PIPT, no need to flush the cache since flushing the TLB will force cache replacements (since the new process will point to a different physical address, which is not in the cache).

For VIVT, you need to flush the cache since a different process might use the same virtual addresses.

For VIPT, no need to flush the cache since the tag comparison will fail anyway due to TLB flush (new set of physical addresses = new tags = forced cache replacements like in PIPT).

In all cases, TLB always gets flushed when context switching.

good comment | 3

Anonymous Atom 5 months ago
for this question, what does update page table address register mean?

helpful! | 0

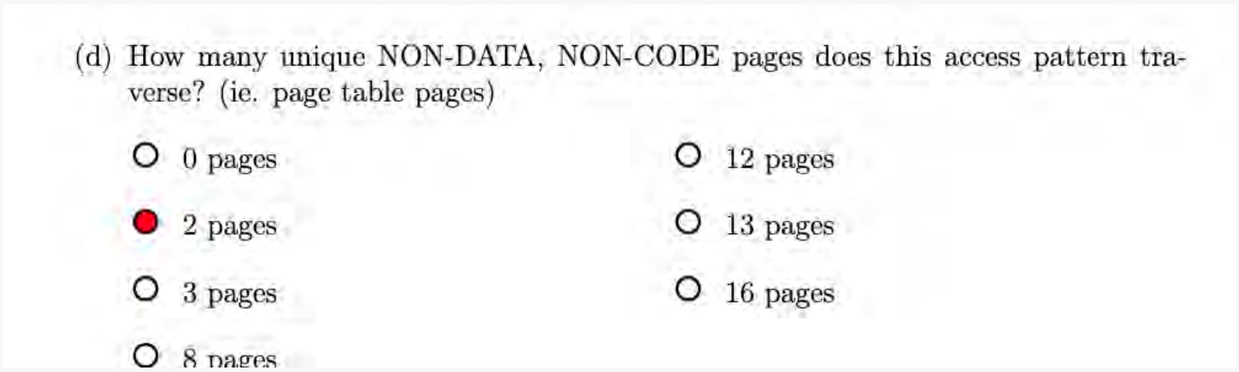
Adelson Chua 5 months ago
The page table address register is something that is very briefly covered in the lecture. It is basically the pointer to the page table. Just like how the L1 page table entries are pointers to the L2 page table, and L2 page table entries are pointers to the data page.

good comment | 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2126_f7

Anonymous Gear 5 months ago
[sp19-final-q3d] I still don't understand why it's 2 pages after watching the video walkthrough. Can someone explain why this is the case? Thanks!


(d) How many unique NON-DATA, NON-CODE pages does this access pattern traverse? (ie. page table pages)

○ 0 pages ○ 12 pages
● 2 pages ○ 13 pages
○ 3 pages ○ 16 pages
○ 8 pages

helpful! | 0

Adelson Chua 5 months ago
Can you tell me how they explained it in the walthrough?
good comment | 0

Anonymous Comp 5 months ago
It's because the virtual addresses index the page table, and 32 entries in the page table would be a single page of memory. Then, if we assume that the entries corresponding to A and B are each contained in a single page of memory (this is possible because we need 8 pages for each of them, i.e., 8 contiguous entries from the page table each), we would need to access one page of memory containing the page table for each A and B, i.e., 2 pages total

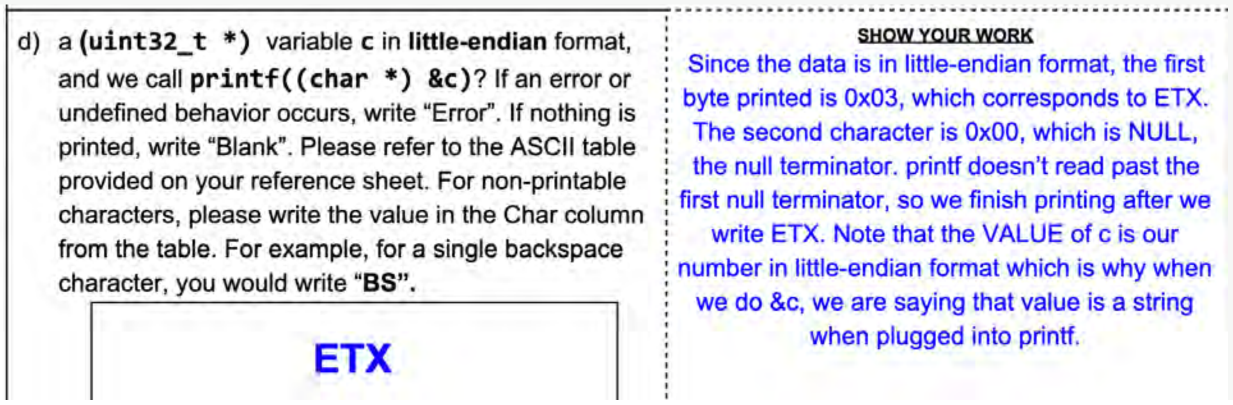
helpful! | 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2126_f8

Anonymous Comp 5 months ago
FA19-Final-Q2D.

It asks you to consider 0xFF000003 as follows


d) a **(uint32_t *)** variable **c** in **little-endian** format, and we call **printf((char *) &c)**? If an error or undefined behavior occurs, write "Error". If nothing is printed, write "Blank". Please refer to the ASCII table provided on your reference sheet. For non-printable characters, please write the value in the Char column from the table. For example, for a single backspace character, you would write "**BS**".

ETX

SHOW YOUR WORK
Since the data is in little-endian format, the first byte printed is 0x03, which corresponds to ETX. The second character is 0x00, which is NULL, the null terminator. printf doesn't read past the first null terminator, so we finish printing after we write ETX. Note that the VALUE of c is our number in little-endian format which is why when we do &c, we are saying that value is a string when plugged into printf.

Why is &c equal to c? c is a pointer to an integer, so &c is a pointer to a pointer to an integer. Furthermore, c is not an array, so it should not necessarily be true that c = &c in general (in fact, it seems false in almost every case)

helpful! | 0

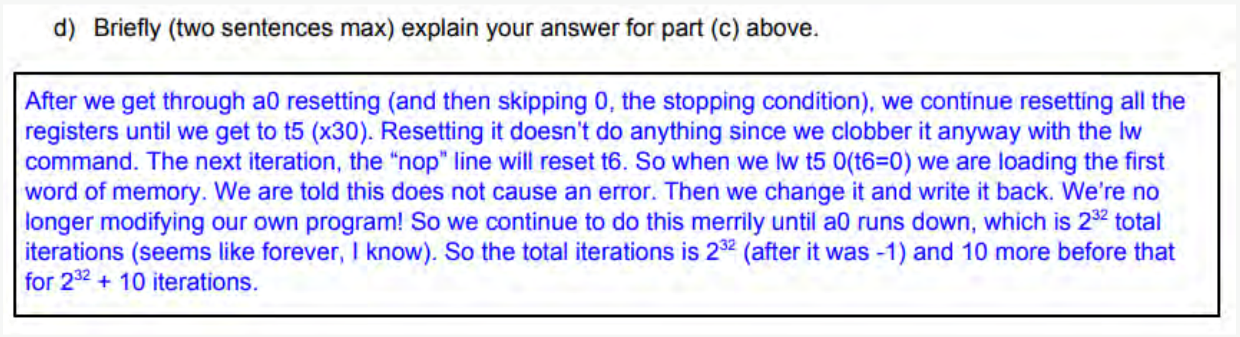
Justin Yokota 5 months ago
&c != c in this case, as you mentioned. The key here is the printf; unlike Python or Java's print functions (which can receive any object type input due to function overloading), C's printf function only receives as input a char pointer, with format strings allowing for other output types. In this case, the printf will read the pointer to c and dereference it, treating the data it finds as a character array.

good comment | 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2126_f9

Anonymous Calc 5 months ago
FA19-final-q4d



d) Briefly (two sentences max) explain your answer for part (c) above.

After we get through a0 resetting (and then skipping 0, the stopping condition), we continue resetting all the registers until we get to t5 (x30). Resetting it doesn't do anything since we clobber it anyway with the lw command. The next iteration, the "nop" line will reset t6. So when we lw t5 0(t6=0) we are loading the first word of memory. We are told this does not cause an error. Then we change it and write it back. We're no longer modifying our own program! So we continue to do this merrily until a0 runs down, which is 2^{32} total iterations (seems like forever, I know). So the total iterations is 2^{32} (after it was -1) and 10 more before that for 2^{32} + 10 iterations.

I'm not understanding the solutions very well. What does it mean for a0 to "run down," and why does it take 2^32 iterations?

helpful! | 0

Peyrin Kao 5 months ago
The loop sets each register to 0 in order. Part of that loop's logic is `lw t5, 0(t6)` , which requires t6 to contain the address of the loop code. Eventually, t6 is set to 0, which means that lw instruction no longer loads code to modify, and instead loads from address 0 in memory.

At this point, a0 has the value -1 = 2^{32} - 1 (see the solution in the previous part for why this happens; it's also because a0 eventually gets reset to 0 as part of the loop). Since the code is no longer modifying itself, the loop will run repeatedly until a0 becomes 0, giving us the 2^{32} iterations in the solution.

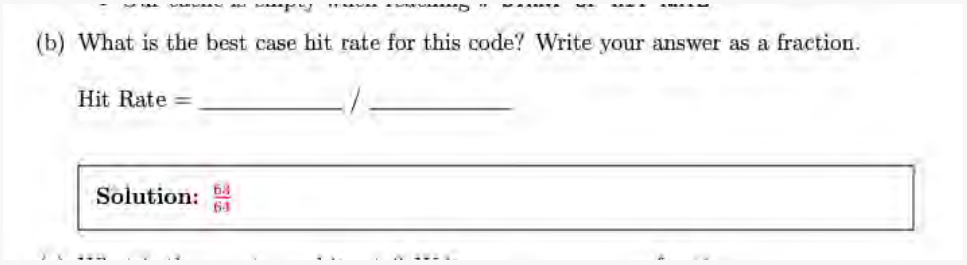
good comment | 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2126_f10

Anonymous Poet 5 months ago

Spring 19, P8



(b) What is the best case hit rate for this code? Write your answer as a fraction.

Hit Rate = ____________ / ____________

Solution: $\frac{63}{64}$

I am still a bit confused after watching walk thorough video. I got the part where array and C have same index where A and B have same index. However, I don't know why we only have 1 missed in total and since we loop 256 times, why we only have 64 accesses

helpful! | 0

Anonymous Comp 5 months ago

If I did it correctly, you do access more than 64 times, but the important pieces of information are that you have 64 bytes of data in each block and there is enough associativity and memory alignment to avoid conflict misses between A, B, C, and arr. This tells you that you'll only have compulsory misses because we'll never need to access anything evicted again. Accesses are symmetric in A, B, C, and arr, so we then only have to look at one of these; furthermore, because 256 is a power of two greater than 64, we only need to look at the hit rate of a single line in the cache. This is $\frac{63}{64}$, the desired answer. The hit rate is just the percentage of memory accesses that are cache hits, so there's no requirement that it is in the form # hits / total # of accesses

helpful! | 1

Adelson Chua 5 months ago

This is correct.

good comment | 1

Reply to this followup discussion

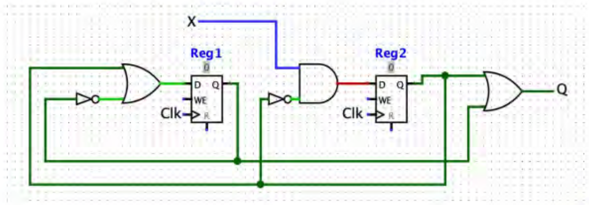
[✓ Resolved] Unresolved @2126_f11

Anonymous Calc 5 months ago

Fa19-final-q5a

Q5) Watch the clock and don't delay! **(30 pts = 2*5 + 10 + 10)**

Consider the following circuit:



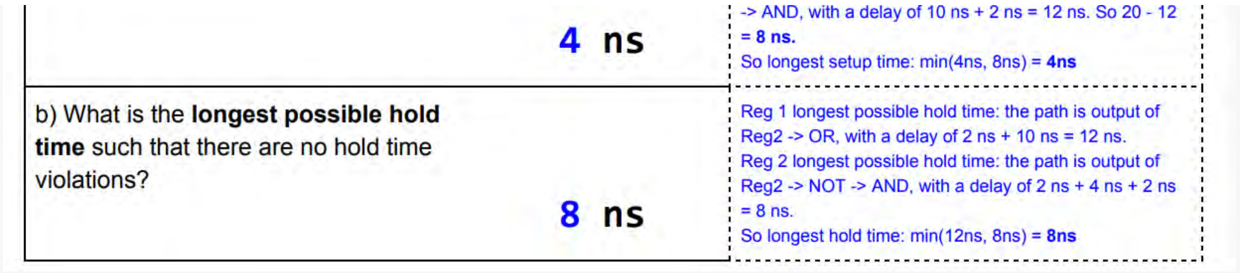
You are given the following information:

- Clk has a frequency of 50 MHz
- AND gates have a propagation delay of 2 ns
- NOT gates have a propagation delay of 4 ns
- OR gates have a propagation delay of 10 ns
- X changes 10ns after the rising edge of Clk
- Reg1 and Reg2 have a clock-to-Q delay of 2 ns

The clock period is 1/(50 * 10^6) s = 20 ns. This means that if X changes, it changes 10 ns after the clock positive edge.

SHOW YOUR WORK BELOW

a) What is the **longest possible setup time** such that there are no setup time violations?	Reg 1 longest possible setup time: the path is output of Reg1 -> NOT -> OR, with a delay of 2 ns + 4 ns + 10 ns = 16 ns. So 20 - 16 = **4 ns.** Reg 2 longest possible setup time: the path is X changes



	4 ns	-> AND, with a delay of 10 ns + 2 ns = 12 ns. So 20 - 12 = **8 ns.** So longest setup time: min(4ns, 8ns) = **4ns**
b) What is the **longest possible hold time** such that there are no hold time violations?	**8 ns**	Reg 1 longest possible hold time: the path is output of Reg2 -> OR, with a delay of 2 ns + 10 ns = 12 ns. Reg 2 longest possible hold time: the path is output of Reg2 -> NOT -> AND, with a delay of 2 ns + 4 ns + 2 ns = 8 ns. So longest hold time: min(12ns, 8ns) = **8ns**

where are we getting the equation for longest possible setup/hold times from?

helpful! 0

Peyrin Kao 5 months ago

Setup time is the time before the next rising edge when the signal must arrive at the register input. Intuitively, we're looking for the longest possible paths in the circuit to see how much time we have left for setup time before the next rising edge. After the rising edge, it takes a maximum of 16ns for values to propagate from reg1 to reg2, and it takes a maximum of 12ns for values to propagate from reg2 to reg1. That means that it takes a maximum 16ns before the inputs to both registers get a stable signal, which leaves us 4ns for setup time.

Hold time is the time after the rising edge when the input signal to the register must stay stable. Intuitively, we're looking for the shortest possible paths in the circuit to see what's the earliest time when the input to the register is going to change. After the rising edge, it takes a minimum of 12ns for values to propagate from reg1 to reg2, and it takes a minimum of 8ns for values to propagate from reg2 to reg1. The soonest a register input will change is 8ns, which means that we have 8ns of hold time when the register inputs are guaranteed to stay stable.

I think formulas for these exist, but at least for me, thinking through the definitions of hold time and setup time intuitively is helpful for when the question doesn't neatly fit into a formula.

good comment 1

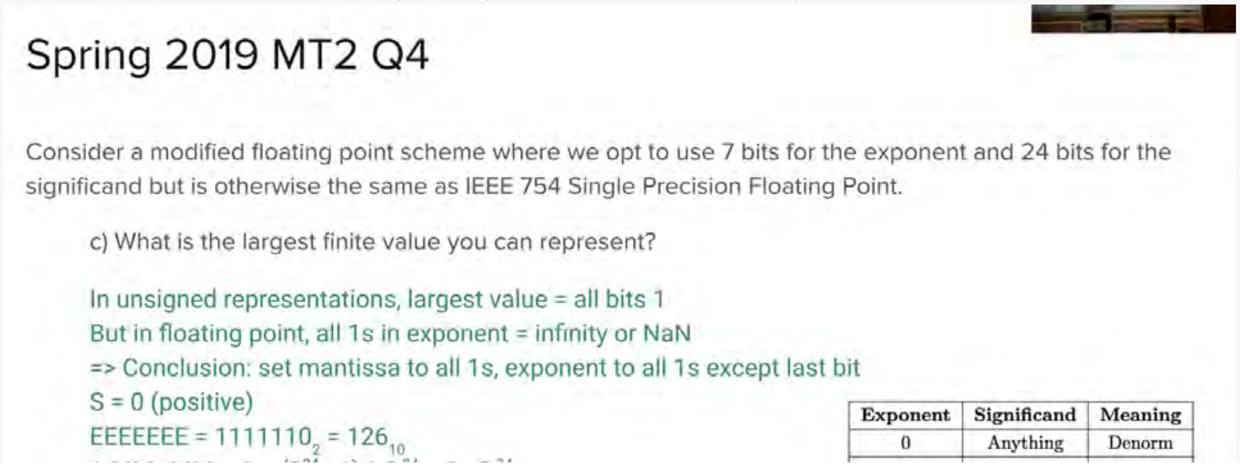
Reply to this followup discussion

[✓ Resolved] Unresolved **@2126_f12**

Anonymous Beaker 5 months ago

SP-29-MT2-Q4

I'm a bit confused as to how exactly was the mantissa calculated in the exam problem from the exam review today. I know why the mantissa is all 1s, but I'm not sure how you get to a final value of 2-2^-24. I know that this is derived from 1 + (2^24 -1) * 2^-24, but I'm not exactly sure where those values come from.



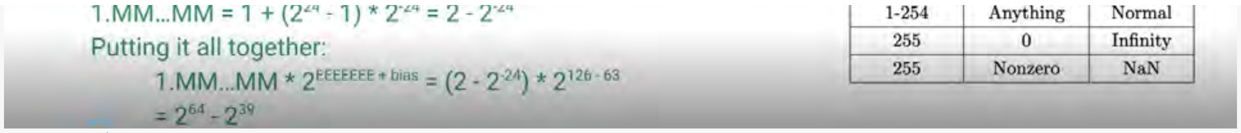
Spring 2019 MT2 Q4

Consider a modified floating point scheme where we opt to use 7 bits for the exponent and 24 bits for the significand but is otherwise the same as IEEE 754 Single Precision Floating Point.

c) What is the largest finite value you can represent?

In unsigned representations, largest value = all bits 1
But in floating point, all 1s in exponent = infinity or NaN
=> Conclusion: set mantissa to all 1s, exponent to all 1s except last bit
S = 0 (positive)
EEEEEEE = $1111110_2 = 126_{10}$

Exponent	Significand	Meaning
0	Anything	Denorm



1.MM...MM = $1 + (2^{24} - 1) * 2^{-24} = 2 - 2^{-24}$

Putting it all together:

1.MM...MM * $2^{EEEEEEE + bias} = (2 - 2^{-24}) * 2^{126 - 63}$

$= 2^{64} - 2^{39}$

1-254	Anything	Normal
255	0	Infinity
255	Nonzero	NaN

helpful! | 0

Adelson Chua 5 months ago

You can follow the steps here.

@2125_f16

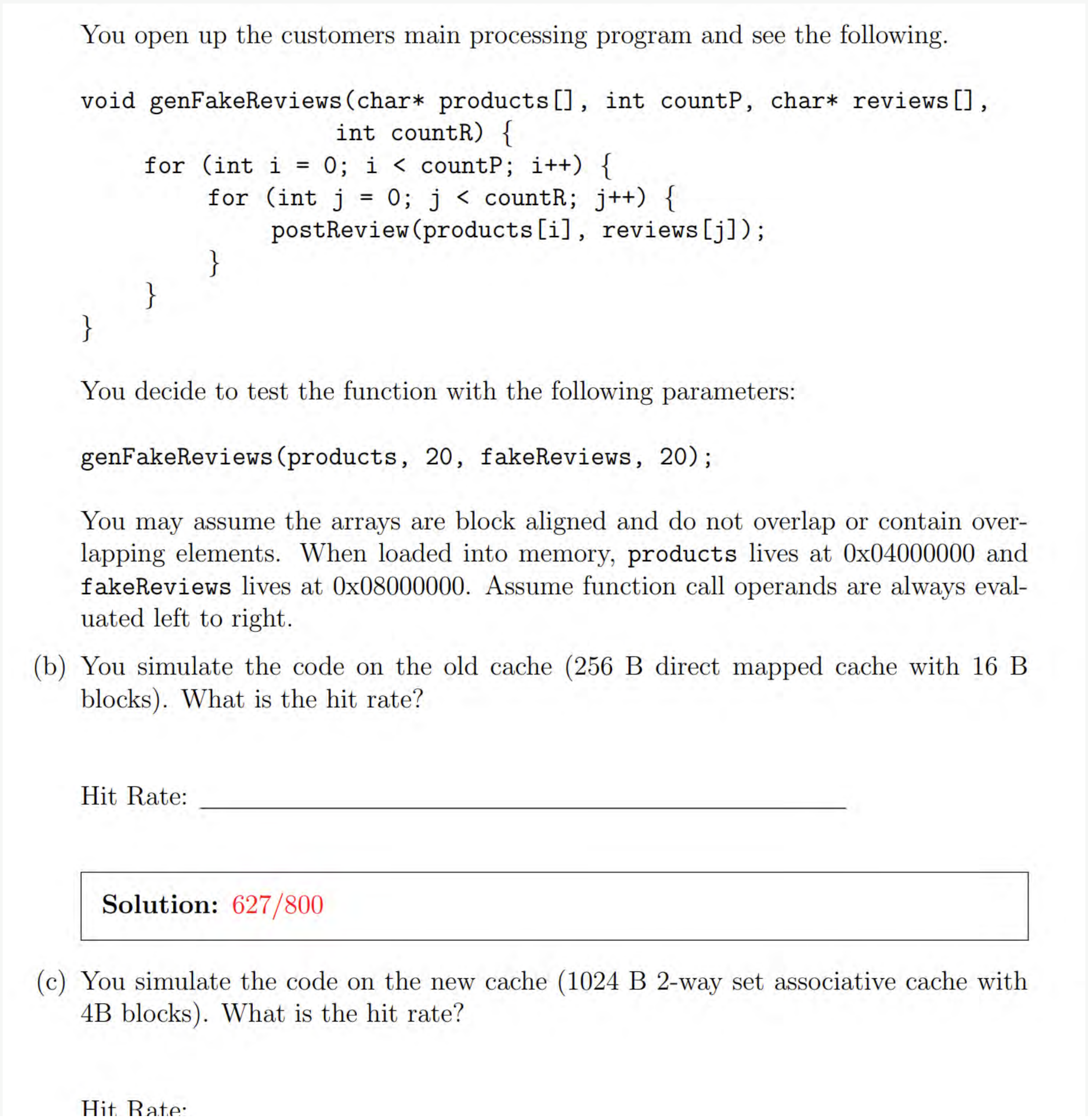
good comment | 0

Reply to this followup discussion

[✓ Resolved] Unresolved **@2126_f13**

Anonymous Scale 5 months ago

I'm not sure how the cache hit rates have been arrived at in this solution:



You open up the customers main processing program and see the following.

```
void genFakeReviews(char* products[], int countP, char* reviews[],
                    int countR) {
     for (int i = 0; i < countP; i++) {
          for (int j = 0; j < countR; j++) {
               postReview(products[i], reviews[j]);
          }
     }
}
```

You decide to test the function with the following parameters:

```
genFakeReviews(products, 20, fakeReviews, 20);
```

You may assume the arrays are block aligned and do not overlap or contain overlapping elements. When loaded into memory, `products` lives at 0x04000000 and `fakeReviews` lives at 0x08000000. Assume function call operands are always evaluated left to right.

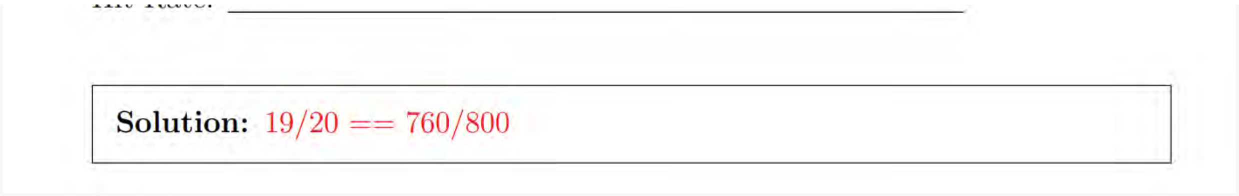
(b) You simulate the code on the old cache (256 B direct mapped cache with 16 B blocks). What is the hit rate?

Hit Rate: ______________________________

Solution: 627/800

(c) You simulate the code on the new cache (1024 B 2-way set associative cache with 4B blocks). What is the hit rate?

Hit Rate:


Solution: 19/20 == 760/800

helpful! | 0

Anonymous Scale 5 months ago
From spring 19 midterm #2.

helpful! | 0

Peyrin Kao 5 months ago
Here's an explanation from an older Piazza:

Here's my totally not copy-pasted response from my semester's piazza thread lol.

Here's Nate Armstrong's and my answer from the [Midterm 2] Grades Released! thread. Lemme know if it doesn't make sense. Would highly encourage you to work out both of them on paper if you're not convinced or don't understand why some misses are occurring. This is a great question to do it on because there is a relatively low number of loops and the pattern is evident pretty quickly.

Nate Armstrong
For 6c, the cache size of 1024B can store the entire character array. Because it is 2-way associative, it never overwrites anything. Thus, the only misses are compulsory misses, of which there are 40. There are 800 total accesses (2*20*20), so the hit rate is (800 - 40)/(800) = 19/20.

my answer
For 6b, the pattern is as follows...
On i = 0 and j = 0 -> 19,
Product[i] will miss a total of 5 times: 1 compulsory miss and 4 conflict misses as review[j] maps to the same index
Review[j] will miss a total of 8 times: 4 conflict misses as it maps to product[i]'s index and 4 more compulsory misses every 4 consecutive integers (the block size fits 4 integers so it will miss, then 3x hit, amounting to 16 remaining iterations / 4 integers = 4 misses)

On i = 1 and j = 0 -> 19,
Product[i] will miss a total of 4 times: only 4 conflict misses as it already has the block from i = 0
Review[j] will miss a total of 4 times: only 4 conflict misses as it has already loaded every other block into memory

This repeats for i = 2, 3

On i = 4 and j = 0 -> 19,
Product [i] will miss a total of 5 times: 4 conflict misses plus 1 compulsory miss to bring in the never before seen block

Review[j] will miss a total of 5 times: 4 conflict misses plus 1 additional conflict miss to bring in the block product[0->3] was squatting on

Both will then only have 4 misses apiece for the next 3 iterations as product[i] doesn't have the compulsory miss and review[j] has already brought the block it was missing on i = 4

This second pattern then continues to repeat every 4 iterations (for the remaining 12 iterations) due to the reasons I listed above.

You can sum the product misses as (5 + 4 + 4 + 4) * 5 = 85 total misses
Summing the review misses gets (8 + 4 + 4 + 4) + (5 + 4 + 4 + 4) * 4 = 88 total misses
Hit rate is (total accesses - misses) / total accesses.
Total accesses = 20 * 20 * 2 due to nested loops and 2 accesses per inner iteration.

Final answer is (800 - (88 + 85)) / 800 = 627 / 800 as stated in the answer.

You may notice that I mistakenly treated the arrays as type int when in reality they are of type string pointer which fortunately does not change the computation. Thanks to the anon who pointed that out on the original thread.

good comment | 1

Reply to this followup discussion

[✓ Resolved] Unresolved **@2126_f14**

Anonymous Poet 5 months ago
is summer 2019 final Q8 in scope?

helpful! | 0

Peyrin Kao 5 months ago
Map-reduce and Spark is not in scope this semester.

good comment | 0

Reply to this followup discussion

[✓ Resolved] Unresolved **@2126_f15**

Anonymous Mouse 2 5 months ago
sp19 q1 part h.1

could we have used a denormalized representation for this and if so would it have been

0b1000000000100100?

helpful! | 0

Adelson Chua 5 months ago
I don't want to think too much.
Can you show the process on how you arrived at your solution and I can verify?

good comment | 0

Anonymous Mouse 2 5 months ago

significand = 1 because -3/64 is negative

3/64 is represented as a 1 at the 1/32 and 1/64 bit so the mantissa is 00001100. That way we can get 0.000011 or 3/64. to get the denorm form, we set all exponent bits to 0 so that would be 0000000.

thanks!

helpful! | 0

Adelson Chua 5 months ago

But then, what is the exponent field for the denormal number? That would be too small.

What you are showing is 3/64 * 2^(-63) (If I'm calculating the exponent of the denormal correctly).

good comment | 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2126_f16

Anonymous Gear 2 5 months ago

Is Problem #4 on Sp19 Final in-scope this semester? If so, where can I get more information on amoswap, lr.w, and sc.w?

helpful! | 0

Adelson Chua 5 months ago

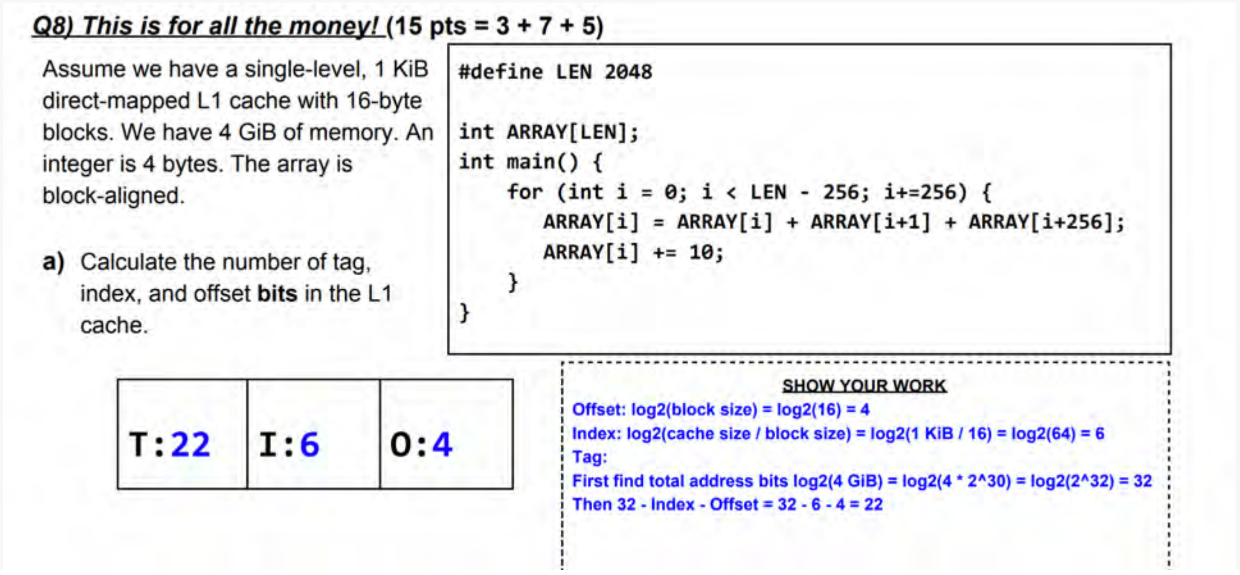
This is covered in the lecture. If that is not enough, there might be resources on the internet (the RISC-V instruction manual, for example) that can give more information on how these instructions work.

good comment | 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2126_f17

Anonymous Helix 2 5 months ago



Q8) This is for all the money! **(15 pts = 3 + 7 + 5)**

Assume we have a single-level, 1 KiB direct-mapped L1 cache with 16-byte blocks. We have 4 GiB of memory. An integer is 4 bytes. The array is block-aligned.

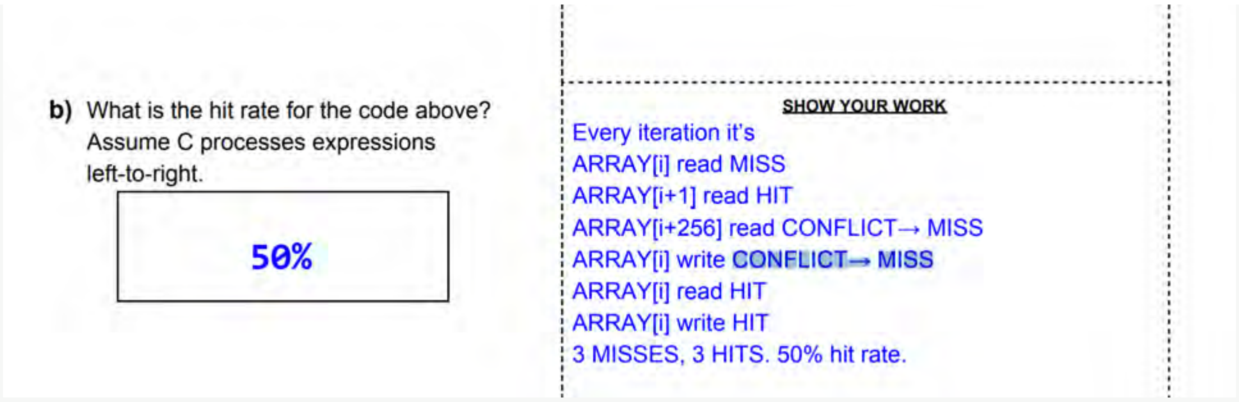
```
#define LEN 2048

int ARRAY[LEN];
int main() {
    for (int i = 0; i < LEN - 256; i+=256) {
        ARRAY[i] = ARRAY[i] + ARRAY[i+1] + ARRAY[i+256];
        ARRAY[i] += 10;
    }
}
```

a) Calculate the number of tag, index, and offset **bits** in the L1 cache.

T:22	I:6	O:4

SHOW YOUR WORK

Offset: log2(block size) = log2(16) = 4
Index: log2(cache size / block size) = log2(1 KiB / 16) = log2(64) = 6
Tag:
First find total address bits log2(4 GiB) = log2(4 * 2^30) = log2(2^32) = 32
Then 32 - Index - Offset = 32 - 6 - 4 = 22



b) What is the hit rate for the code above? Assume C processes expressions left-to-right.

50%

SHOW YOUR WORK

Every iteration it's
ARRAY[i] read MISS
ARRAY[i+1] read HIT
ARRAY[i+256] read CONFLICT→ MISS
ARRAY[i] write CONFLICT→ MISS
ARRAY[i] read HIT
ARRAY[i] write HIT
3 MISSES, 3 HITS. 50% hit rate.

For this question, how can we assume that array[i+256] will map into the same index of the direct mapped cache as array[i]. Thanks!

helpful! 0

Adelson Chua 5 months ago

Write it out in binary. The T/I/O bits are already there.

Address 0: 0000 0000 0000

Address 256 (times 4 since 4 bytes per int): 0100 0000 0000

Regrouping so that it is obvious:

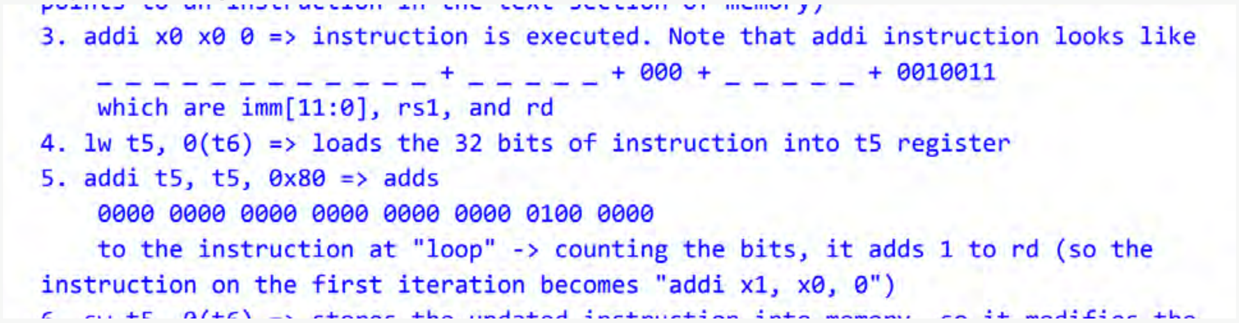
Address 0	00	000000	0000
Address 256	01	000000	0000

good comment 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2126_f18

Anonymous Comp 2 5 months ago



```
3. addi x0 x0 0 => instruction is executed. Note that addi instruction looks like
    _ _ _ _ _ _ _ _ _ _ _ _ + _ _ _ _ _ + 000 + _ _ _ _ _ + 0010011
    which are imm[11:0], rs1, and rd
4. lw t5, 0(t6) => loads the 32 bits of instruction into t5 register
5. addi t5, t5, 0x80 => adds
    0000 0000 0000 0000 0000 0000 0100 0000
    to the instruction at "loop" -> counting the bits, it adds 1 to rd (so the
instruction on the first iteration becomes "addi x1, x0, 0")
```

For fa19, isn't when we do step 5 aren't we adding 1 to the opcode and not the register, because:

0010011

1000000

helpful! 0

Peyrin Kao 5 months ago

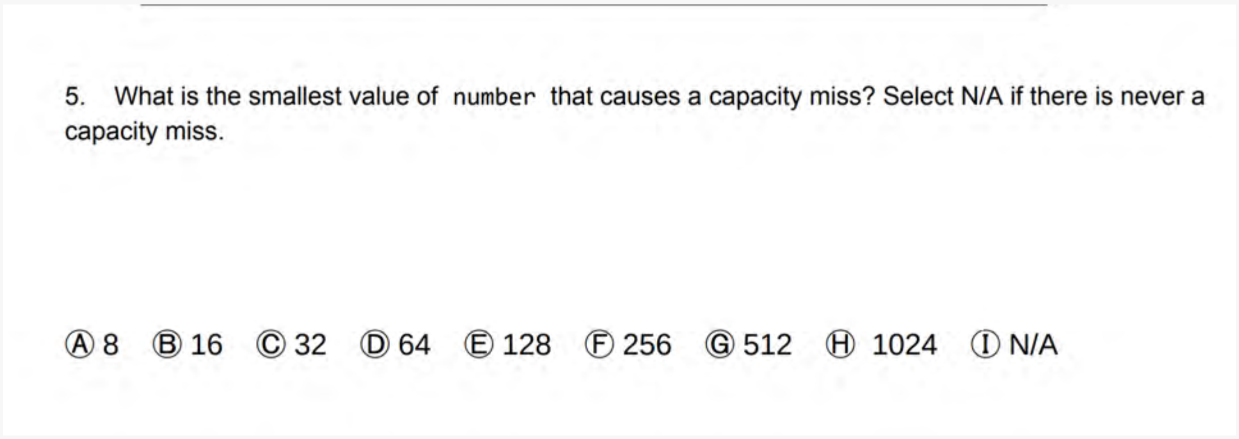
0x80 = 0b1000 0000. The bottom 7 bits are the opcode, and the 8th bit is the lowest bit of rd.

good comment 0

Reply to this followup discussion

Resolved [✓] Unresolved @2126_f19

Anonymous Comp 2 5 months ago


5. What is the smallest value of `number` that causes a capacity miss? Select N/A if there is never a capacity miss.

Ⓐ 8 Ⓑ 16 Ⓒ 32 Ⓓ 64 Ⓔ 128 Ⓕ 256 Ⓖ 512 Ⓗ 1024 Ⓘ N/A

For Su19 question 7.5 I fail to see how there isn't a capacity miss, because at most it can hold 32 longs, and at fib(64) We are accessing more than 32 longs and capacity misses occur when the caches are entirety full, hence it should occur after 32?

helpful! 0

Anonymous Comp 2 5 months ago
Same thing for 7.6 for conflict miss, because the set runs out of space?

helpful! 0

Adelson Chua 5 months ago
Capacity miss only happens when you access the memory that was evicted before.
This problem does not revisit any previous memory accesses, so all addresses are 'new'. These are compulsory misses.

@1270_f6
If the address was in the cache, but isn't anymore (got evicted) and the cache is still not full: Conflict miss
If the address was in the cache, but isn't anymore (got evicted) and the cache is full: Capacity miss
If the address was never in the cache (cold start): Compulsory miss

good comment 1

Reply to this followup discussion

Resolved [✓] Unresolved @2126_f20

Anonymous Atom 2 5 months ago
For SP19 Q5:
Just for clarification, will the SIMD intrinsics be provided in the test?
It seems like it's not provided in this test.

helpful! 0

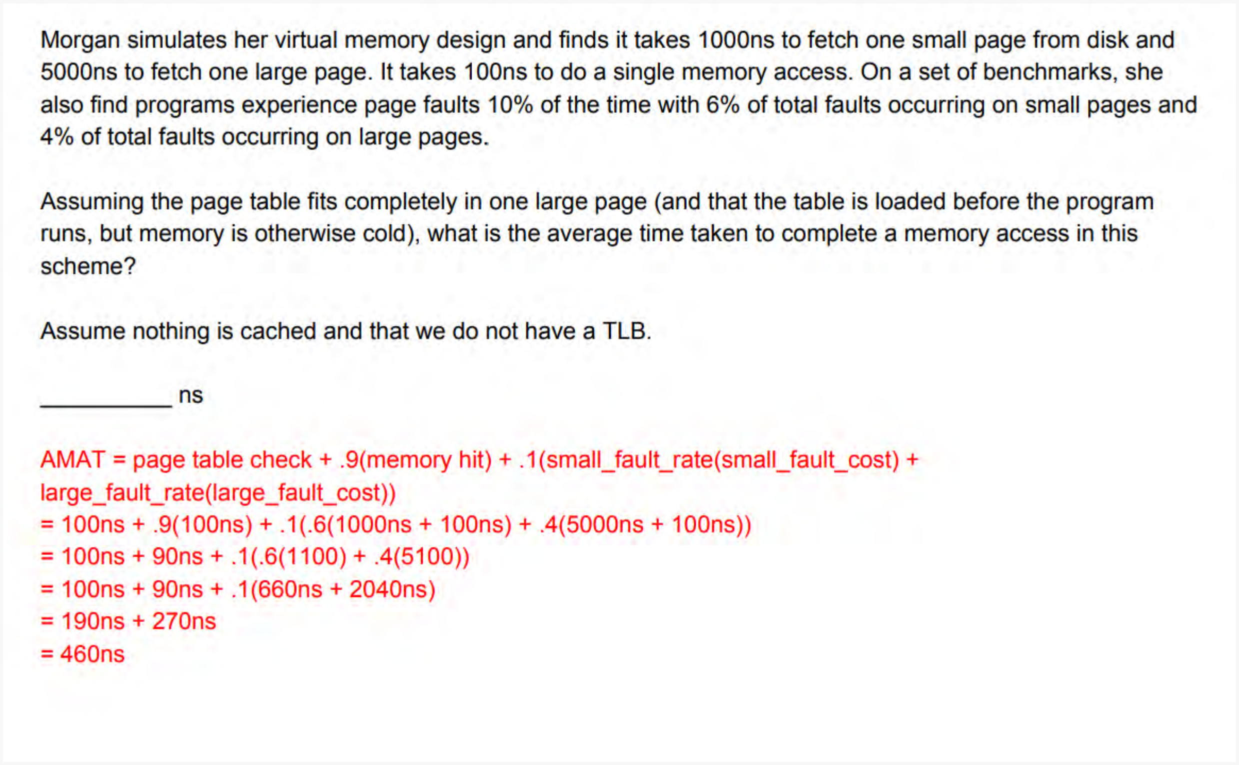
Adelson Chua 5 months ago
@1936_f7

good comment | 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2126_f21

Anonymous Comp 2 5 months ago



Morgan simulates her virtual memory design and finds it takes 1000ns to fetch one small page from disk and 5000ns to fetch one large page. It takes 100ns to do a single memory access. On a set of benchmarks, she also find programs experience page faults 10% of the time with 6% of total faults occurring on small pages and 4% of total faults occurring on large pages.

Assuming the page table fits completely in one large page (and that the table is loaded before the program runs, but memory is otherwise cold), what is the average time taken to complete a memory access in this scheme?

Assume nothing is cached and that we do not have a TLB.

__________ ns

AMAT = page table check + .9(memory hit) + .1(small_fault_rate(small_fault_cost) + large_fault_rate(large_fault_cost))
= 100ns + .9(100ns) + .1(.6(1000ns + 100ns) + .4(5000ns + 100ns))
= 100ns + 90ns + .1(.6(1100) + .4(5100))
= 100ns + 90ns + .1(660ns + 2040ns)
= 190ns + 270ns
= 460ns

What is the first 100 ns + .9 (100 ns) for? I thought the 100 ns is time to check page table and in that same time memory would either be hit or not?

helpful! | 0

Adelson Chua 5 months ago
The first 100ns is the page table check in the main memory.
The second is the 100ns is the access to the main memory to get the data that you want (after getting the translation from the page table).

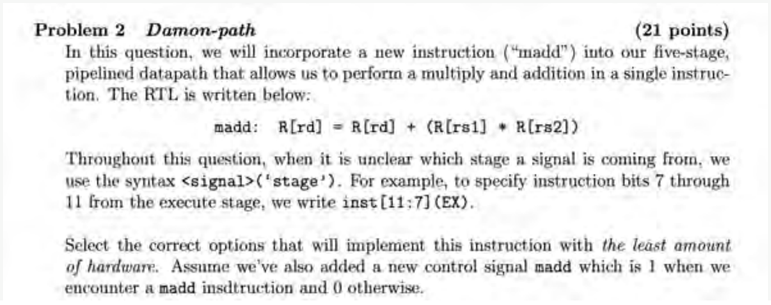
good comment | 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2126_f22

Anonymous Mouse 5 months ago

SP19-Final-Q1c



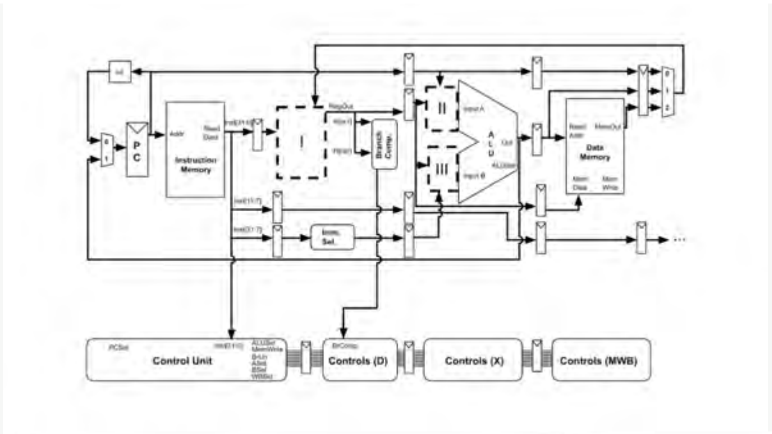
Problem 2 ***Damon-path*** **(21 points)**

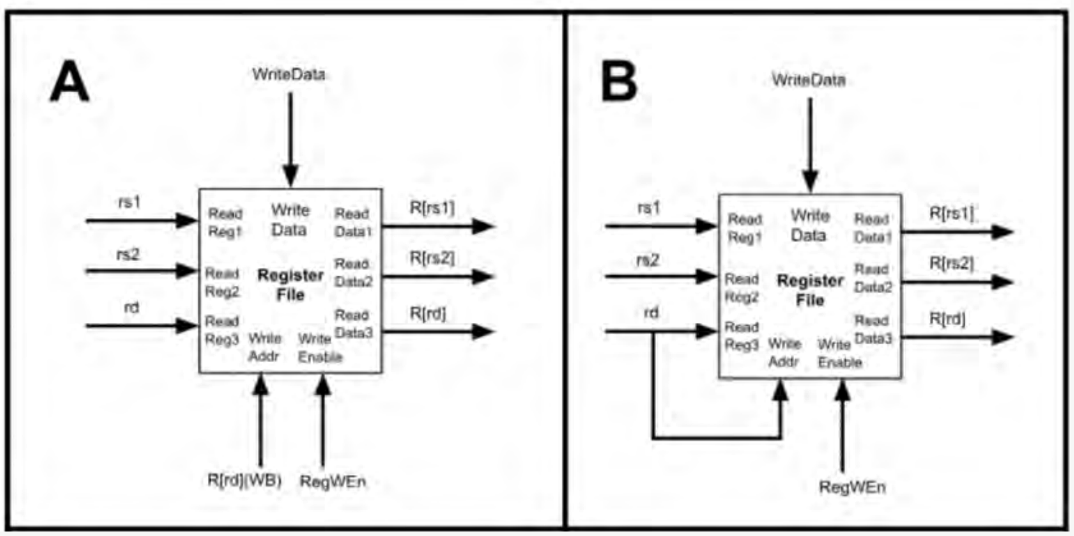
In this question, we will incorporate a new instruction ("madd") into our five-stage, pipelined datapath that allows us to perform a multiply and addition in a single instruction. The RTL is written below:

```
madd:  R[rd] = R[rd] + (R[rs1] * R[rs2])
```

Throughout this question, when it is unclear which stage a signal is coming from, we use the syntax `<signal>('stage')`. For example, to specify instruction bits 7 through 11 from the execute stage, we write `inst[11:7](EX)`.

Select the correct options that will implement this instruction with *the least amount of hardware*. Assume we've also added a new control signal **madd** which is 1 when we encounter a **madd** insdtruction and 0 otherwise.





This question is asking which should be in the dotted box I. The solution has A as the answer selected. However, shouldn't it be B? The WriteAddr should be 5 bits to indicate which of the 32 registers to write back to right? And this fits rd being input as Write Addr. In answer A, we have R[rd] connected up to Write Addr, which means we are feeding the actual data of R[rd] into the Write Addr input, which doesn't feel right to me. Am I missing something?

helpful! | 0

Anonymous Mouse 5 months ago

SP19-Final-Q2***

helpful! | 0

Adelson Chua 5 months ago

The processor is pipelined. The R[rd](WB) refers to rd that is on the WB stage of the pipeline which is synchronized with the actual data to be written to rd itself.

good comment | 0

Anonymous Mouse 5 months ago

I still am a bit foggy on this. So you are saying that somehow R[rd](WB) refers to the 5 bits that represent the rd register?

How is this information transferred to the the Regfile? I don't see any arrows from the control logic being sent up to block I.

helpful! | 0

Adelson Chua 5 months ago

So you are saying that somehow R[rd](WB) refers to the 5 bits that represent the rd register?

Yes, their naming scheme is weird.

How is this information transferred to the the Regfile? I don't see any arrows from the control logic being sent up to block I.

Yeah, they're also not showing it. Either they want the student to figure it out on their own, or it is a weirdly made problem.

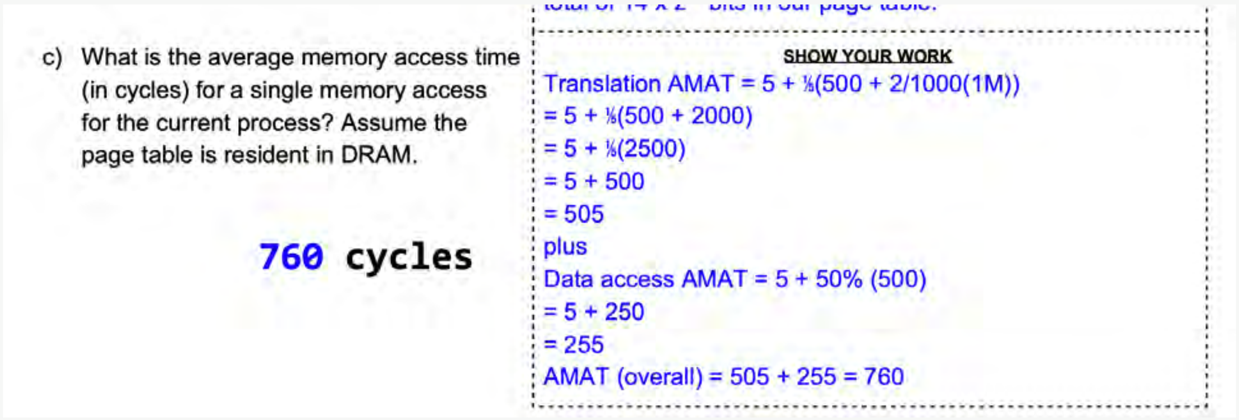
good comment | 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2126_f23

Anonymous Calc 2 5 months ago

FA19-Final-Q9, for this question what is translation and data access AMAT and how do you differentiate the two?



total of 14 x 2 bits in our page table.

c) What is the average memory access time (in cycles) for a single memory access for the current process? Assume the page table is resident in DRAM.

760 cycles

SHOW YOUR WORK

Translation AMAT = 5 + ⅕(500 + 2/1000(1M))
= 5 + ⅕(500 + 2000)
= 5 + ⅕(2500)
= 5 + 500
= 505
plus
Data access AMAT = 5 + 50% (500)
= 5 + 250
= 255
AMAT (overall) = 505 + 255 = 760

helpful! | 0

Adelson Chua 5 months ago

Remember that the CPU initially has the virtual address? That has to be translated first which can be acquired through the TLB or (if that misses) to the main memory or (if that also misses, page fault) to the secondary storage.

After getting the translation, the CPU now has the physical address which is used to access the cache or (if that misses) to the main memory.

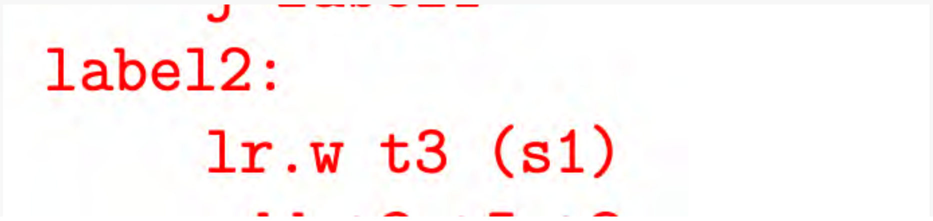
good comment | 0

Reply to this followup discussion

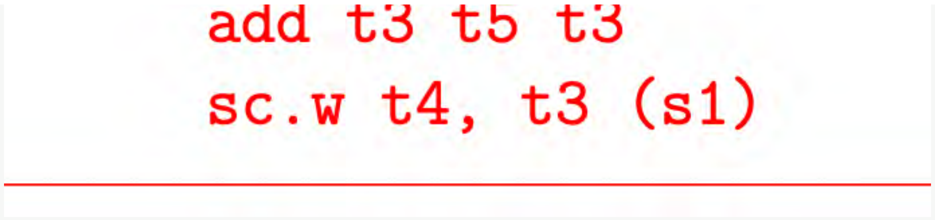
[✓ Resolved] Unresolved @2126_f24

Anonymous Mouse 5 months ago

SP19-Final-Q4.b.



```
label2:
        lr.w t3 (s1)
```



```
add t3 t5 t3
sc.w t4, t3 (s1)
```

this set of instructions can always be done with an amoadd instead right? I can't think of any scenario where amoadd isn't simpler.

helpful! | 0

Adelson Chua 5 months ago

Probably. Just check the actual functionality of amoadd:

Format

amoadd.w rd,rs2,(rs1)

Description

atomically load a 32-bit signed data value from the address in rs1, place the value into register rd, apply add the loaded value and the original 32-bit signed value in rs2, then store the result back to the address in rs1.

Not sure if this is the intended operation. I don't want to think too much. :)

good comment | 1

Reply to this followup discussion

Start a new followup discussion

Compose a new followup discussion