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

note @2125 — 157 views

Actions

[Past Exams] 2018 and older

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: **SP21-Final-Q1**, or **SU21-MT2-Q3**

exam exam/final

good note | 0 — Updated 5 months ago by Jerry Xu and Peyrin Kao

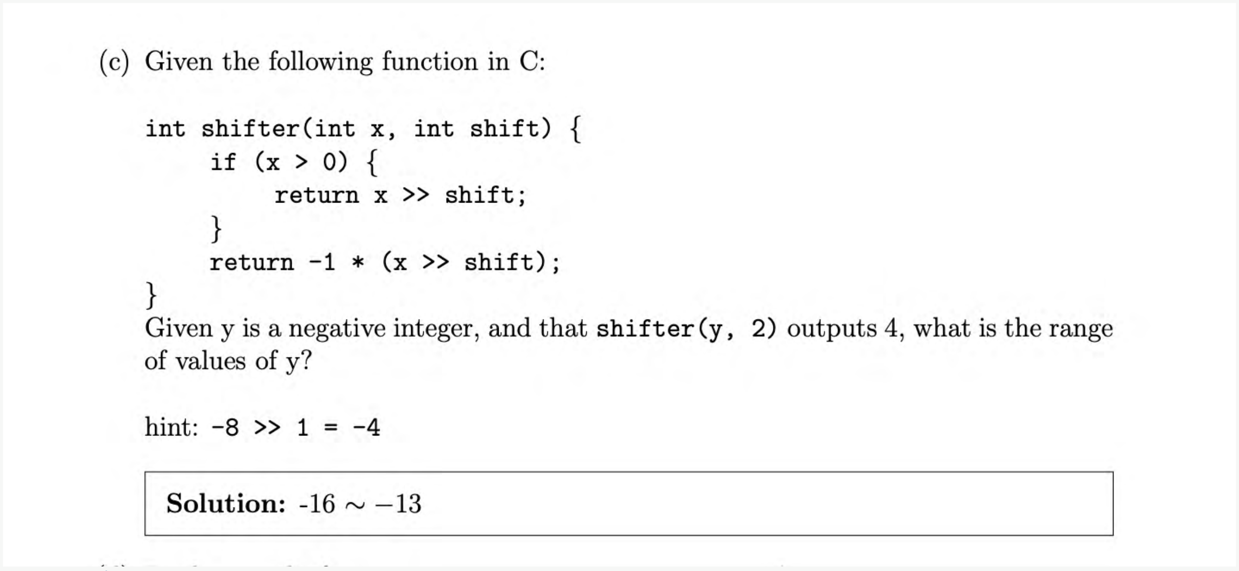
followup discussions, *for lingering questions and comments*

[✓ Resolved] Unresolved @2125_f1

Anonymous Gear 5 months ago

SP18-Final-Q1C

For this question, why would there be a range of available values? I was thinking about it could only be 16. The previous questions are about base 4, I am not sure is this could be done in base 4.



(c) Given the following function in C:

```
int shifter(int x, int shift) {
    if (x > 0) {
        return x >> shift;
    }
    return -1 * (x >> shift);
}
```

Given y is a negative integer, and that `shifter(y, 2)` outputs 4, what is the range of values of y?

hint: `-8 >> 1 = -4`

Solution: -16 ~ −13

helpful! | 0

Adelson Chua 5 months ago

This is in binary.

>> is an arithmetic right shift operation. -16 >> 2 can be translated to -16/4 = -4.

y has a range because shifting to the right by 2 truncates/removes the last two bits. You can also think about it as -13/4 = -4 (it gets rounded towards the negative side).

good comment 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f2

Anonymous Gear 5 months ago

SP18-Final-Q1C

For 1b, part ii, does 128 only include one 0? I was thinking about 127 because we double counted negative zero and positive zero.

helpful! 0

Adelson Chua 5 months ago

I think it does include the doubled 0 representation. It's only really asking for 'valid' representations.

good comment 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f3

Anonymous Poet 5 months ago

SP-18-4b

Why is jal printf considered a pseudoinstruction? jal isn't listed as a pseudoinstruction on the 61C reference card, it is it because of printf?

helpful! 0

Anonymous Poet 5 months ago

*SP-18-Final-Q4b

helpful! 0

Adelson Chua 5 months ago

I think it was considered as pseudoinstruction because it does not include the register where the return address will be saved.

The 'full' instruction should be: **jal ra printf**

good comment 1

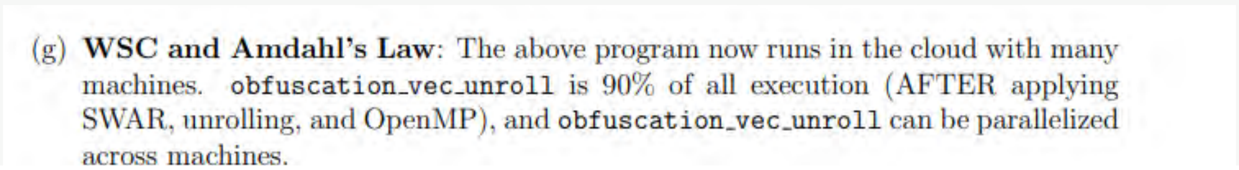
Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f4

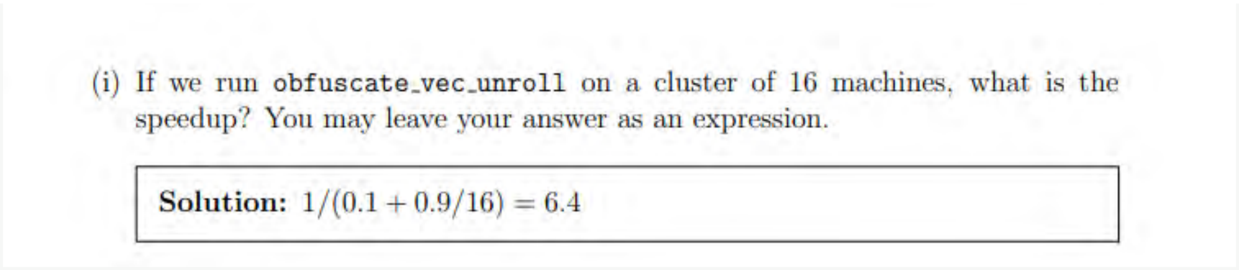
Anonymous Poet 5 months ago

SP-18-Final-Q10g

Sorry if this is obvious, but why is S=16 here? Can we generalize that if we run a program on n different machines, the speed up factor is n?



(g) **WSC and Amdahl's Law**: The above program now runs in the cloud with many machines. `obfuscation_vec_unroll` is 90% of all execution (AFTER applying SWAR, unrolling, and OpenMP), and `obfuscation_vec_unroll` can be parallelized across machines.



(i) If we run `obfuscate_vec_unroll` on a cluster of 16 machines, what is the speedup? You may leave your answer as an expression.

Solution: $1/(0.1 + 0.9/16) = 6.4$

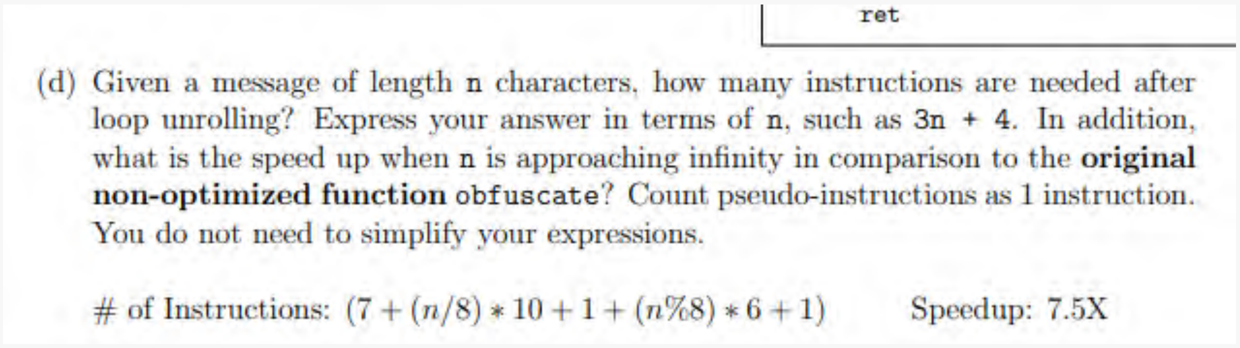
helpful! | 0

Peyrin Kao 5 months ago
I think it depends on the setup of the question. In this case, since there wasn't any extra information about the speedup, I guess it's reasonable to assume 16x speedup.
good comment | 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f5

Anonymous Poet 5 months ago
SP18-Final-Q10d
Is this question in scope? I don't recall doing anything like it. If it is, how can we derive the equation, I don't really understand anything other than we are breaking up the tail cases with n % 8



```
ret
```

(d) Given a message of length `n` characters, how many instructions are needed after loop unrolling? Express your answer in terms of `n`, such as `3n + 4`. In addition, what is the speed up when `n` is approaching infinity in comparison to the **original non-optimized function** `obfuscate`? Count pseudo-instructions as 1 instruction. You do not need to simplify your expressions.

# of Instructions: $(7 + (n/8) * 10 + 1 + (n\%8) * 6 + 1)$ Speedup: 7.5X

helpful! | 0

Peyrin Kao 5 months ago
There are 7 setup instructions between `obfuscate_vec_unrolled` and `LOOP_VEC`.

There are 10 instructions inside the `LOOP_VEC` loop. The loop runs `n/8` times (you can use the bounds of the for loop in the C code to see this).

There's 1 instruction in `TAIL`.

There's 6 instructions in `LOOP_TAIL`. This loop runs `n%8` times (again, the bounds of the for loop in C are helpful to calculate this).

There's one instruction in `end`.
good comment | 1

Anonymous Comp 5 months ago
How to calculate the 7.5x speedup?

helpful! | 1

Peyrin Kao 5 months ago
Non-optimized code: The loop has 6 instructions that works on 1 character. That's 6 instructions per character.

Optimized code: The loop has 10 instructions that works on 8 characters. That's 1.25 instructions per character.

6/1.25 = 4.8, so I think my answer here is 4.8x. Not sure how you'd get 7.5 here.
good comment | 0

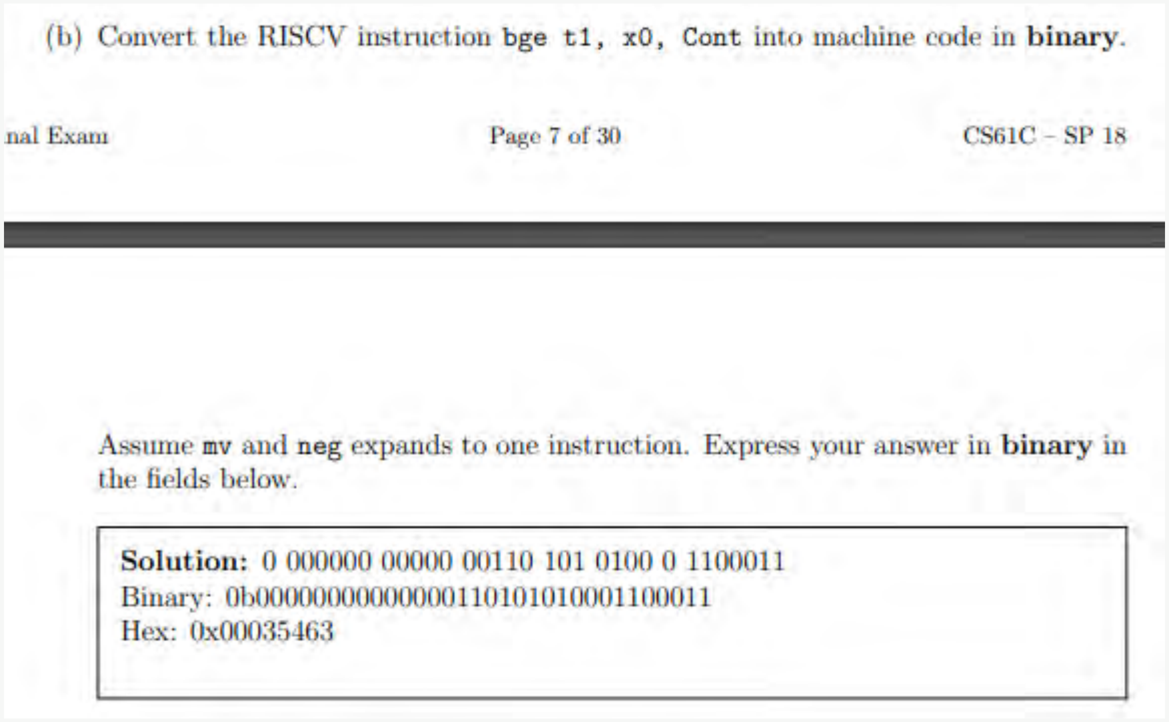
Anonymous Mouse 5 months ago
i think that's what answer gives? 7.5X
helpful! | 0

Peyrin Kao 5 months ago
Yeah, I'm not sure how they got that number. If I were to solve this, I'd answer 4.8x. I got the same answer trying it out last semester, so maybe it's safe to assume it's a bug in the solutions...
good comment | 1

Reply to this followup discussion

[● Resolved] Unresolved @2125_f6

Anonymous Poet 5 months ago
SP18-Final-Q3b
How did we get the immediate here? I'm confused because the code did not have line numbers on them .


(b) Convert the RISCV instruction `bge t1, x0, Cont` into machine code in **binary**.

nal Exam Page 7 of 30 CS61C – SP 18

Assume `mv` and `neg` expands to one instruction. Express your answer in **binary** in the fields below.

Solution: 0 000000 00000 00110 101 0100 0 1100011
Binary: 0b00000000000000110101010001100011
Hex: 0x00035463

helpful! | 0

Peyrin Kao 5 months ago

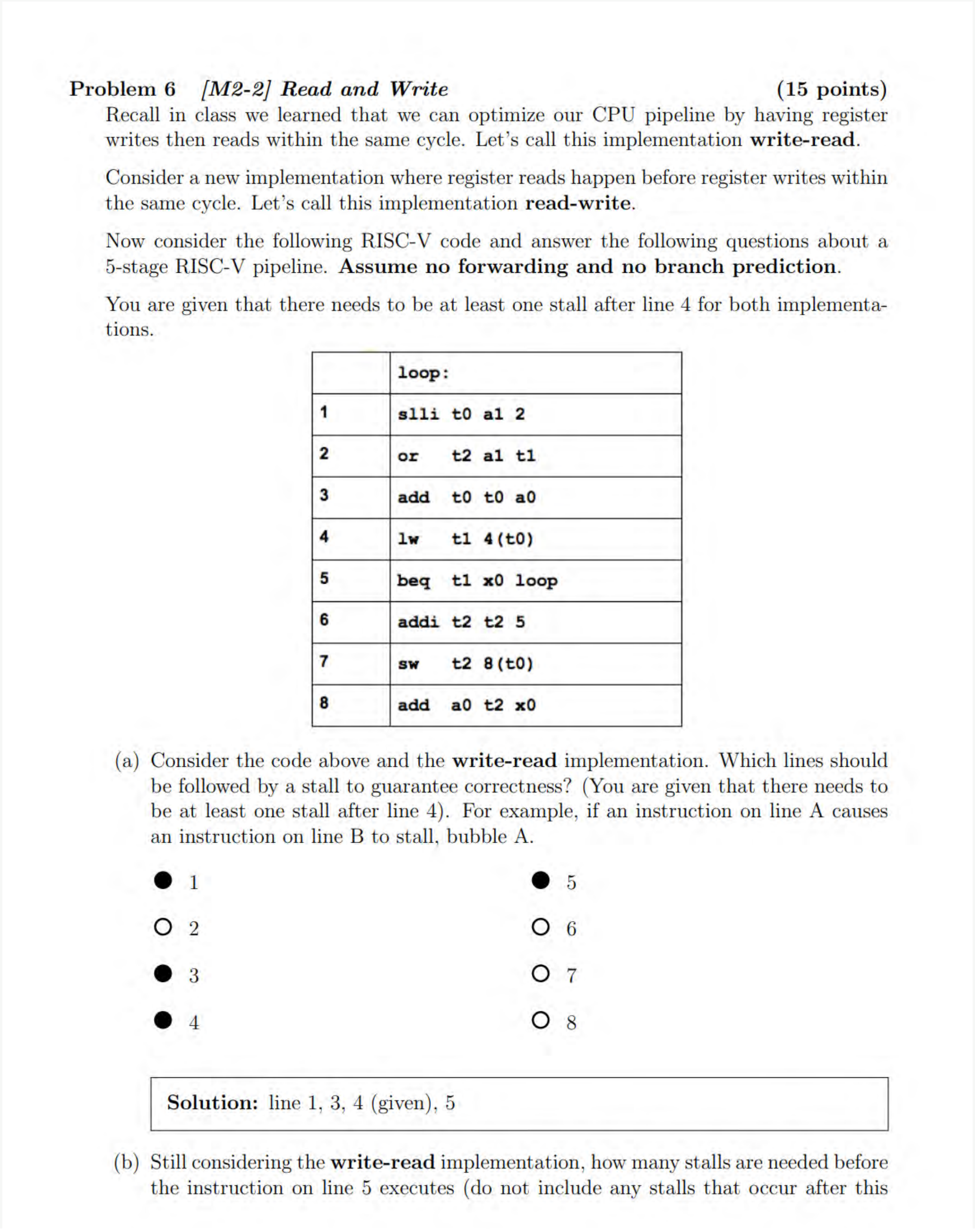
Branches use relative offsets. The `cont` label is two instructions = 8 bytes ahead of the bge instruction (using the assumption that the `neg` pseudoinstruction expands to one instruction).

good comment | 1

Reply to this followup discussion

Resolved Unresolved @2125_f7

Anonymous Helix 5 months ago



Problem 6 ***[M2-2] Read and Write*** **(15 points)**

Recall in class we learned that we can optimize our CPU pipeline by having register writes then reads within the same cycle. Let's call this implementation **write-read**.

Consider a new implementation where register reads happen before register writes within the same cycle. Let's call this implementation **read-write**.

Now consider the following RISC-V code and answer the following questions about a 5-stage RISC-V pipeline. **Assume no forwarding and no branch prediction**.

You are given that there needs to be at least one stall after line 4 for both implementations.

	`loop:`
1	`slli t0 a1 2`
2	`or   t2 a1 t1`
3	`add  t0 t0 a0`
4	`lw   t1 4(t0)`
5	`beq  t1 x0 loop`
6	`addi t2 t2 5`
7	`sw   t2 8(t0)`
8	`add  a0 t2 x0`

(a) Consider the code above and the **write-read** implementation. Which lines should be followed by a stall to guarantee correctness? (You are given that there needs to be at least one stall after line 4). For example, if an instruction on line A causes an instruction on line B to stall, bubble A.

● 1
○ 2
● 3
● 4
● 5
○ 6
○ 7
○ 8

Solution: line 1, 3, 4 (given), 5

(b) Still considering the **write-read** implementation, how many stalls are needed before the instruction on line 5 executes (do not include any stalls that occur after this

Why isn't 6 selected for this?

We are writing t2 = t2 + 5, so won't we need to wait for that lines writeback before we start the next line's decode phase?

Thanks!

helpful! | 1

Adelson Chua 5 months ago

Yeah, this might have been a typo. I need to verify.

I agree with your observation.

good comment | 0

Anonymous Comp 5 months ago

Why do we need to stall after 1?

helpful! | 0

Adelson Chua 5 months ago

There is data dependency from the 3rd instruction.

F	D	X	M	**W**		
	F	D	X	M	W	
		F	**D**	X	M	W

For instruction 3 to get the updated data at t0, the WB stage of the 1st instruction needs to align with the D stage of the 3rd instruction.

It is not aligned.

good comment | 0

Anonymous Calc 5 months ago

Any update on why 6 isn't selected?

helpful! | 0

Anonymous Calc 5 months ago

Is the reason we stall after 5 because it says no branch prediction? Basically everything stalls until the ALU step of the branch instruction finishes in the pipeline?

helpful! | 0

Adelson Chua 5 months ago

Stall at 5 because of control hazard (yes, no branch prediction).

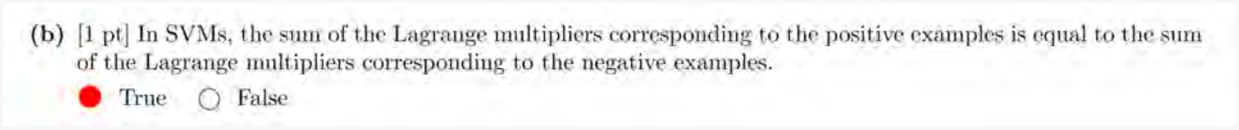
Accept that 6 should be included. The data dependency is pretty obvious.

good comment | 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f8

Anonymous Helix 5 months ago



(b) [1 pt] In SVMs, the sum of the Lagrange multipliers corresponding to the positive examples is equal to the sum of the Lagrange multipliers corresponding to the negative examples.

[✓ True] False

Are the Lagrange multipliers here the slack variables?

Thanks!

helpful! | 0

Anonymous Helix 5 months ago

Sorry wrong Piazza

helpful! | 0

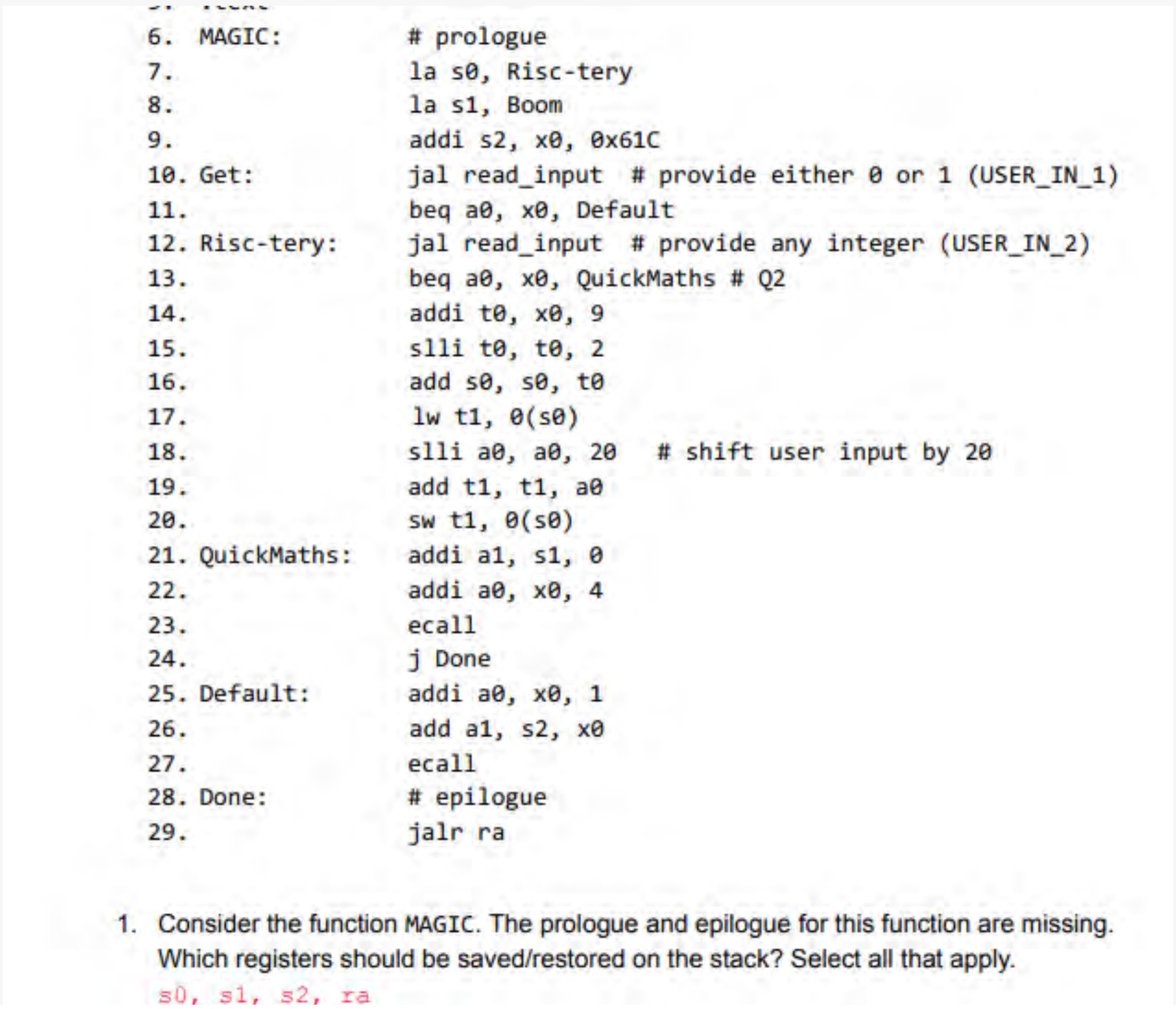
Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f9

Anonymous Beaker 5 months ago

FA 17, Q4 part 2:

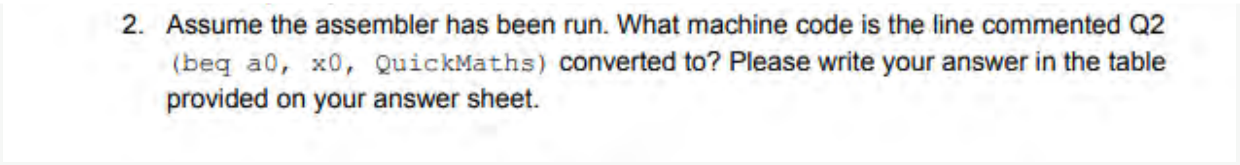
how do we ascertain what address A function is like quickmaths in the risc v code below?



```
.text
MAGIC:         # prologue
               la s0, Risc-tery
               la s1, Boom
               addi s2, x0, 0x61C
Get:           jal read_input  # provide either 0 or 1 (USER_IN_1)
               beq a0, x0, Default
Risc-tery:     jal read_input  # provide any integer (USER_IN_2)
               beq a0, x0, QuickMaths # Q2
               addi t0, x0, 9
               slli t0, t0, 2
               add s0, s0, t0
               lw t1, 0(s0)
               slli a0, a0, 20   # shift user input by 20
               add t1, t1, a0
               sw t1, 0(s0)
QuickMaths:    addi a1, s1, 0
               addi a0, x0, 4
               ecall
               j Done
Default:       addi a0, x0, 1
               add a1, s2, x0
               ecall
Done:          # epilogue
               jalr ra
```

1. Consider the function MAGIC. The prologue and epilogue for this function are missing. Which registers should be saved/restored on the stack? Select all that apply.

   s0, s1, s2, ra


2. Assume the assembler has been run. What machine code is the line commented Q2 (`beq a0, x0, QuickMaths`) converted to? Please write your answer in the table provided on your answer sheet.

helpful! | 0

Adelson Chua 5 months ago
branch instructions does not need to know the absolute address. You just calculate the amount of offset needed from the branch instruction to the target label.
good comment | 0

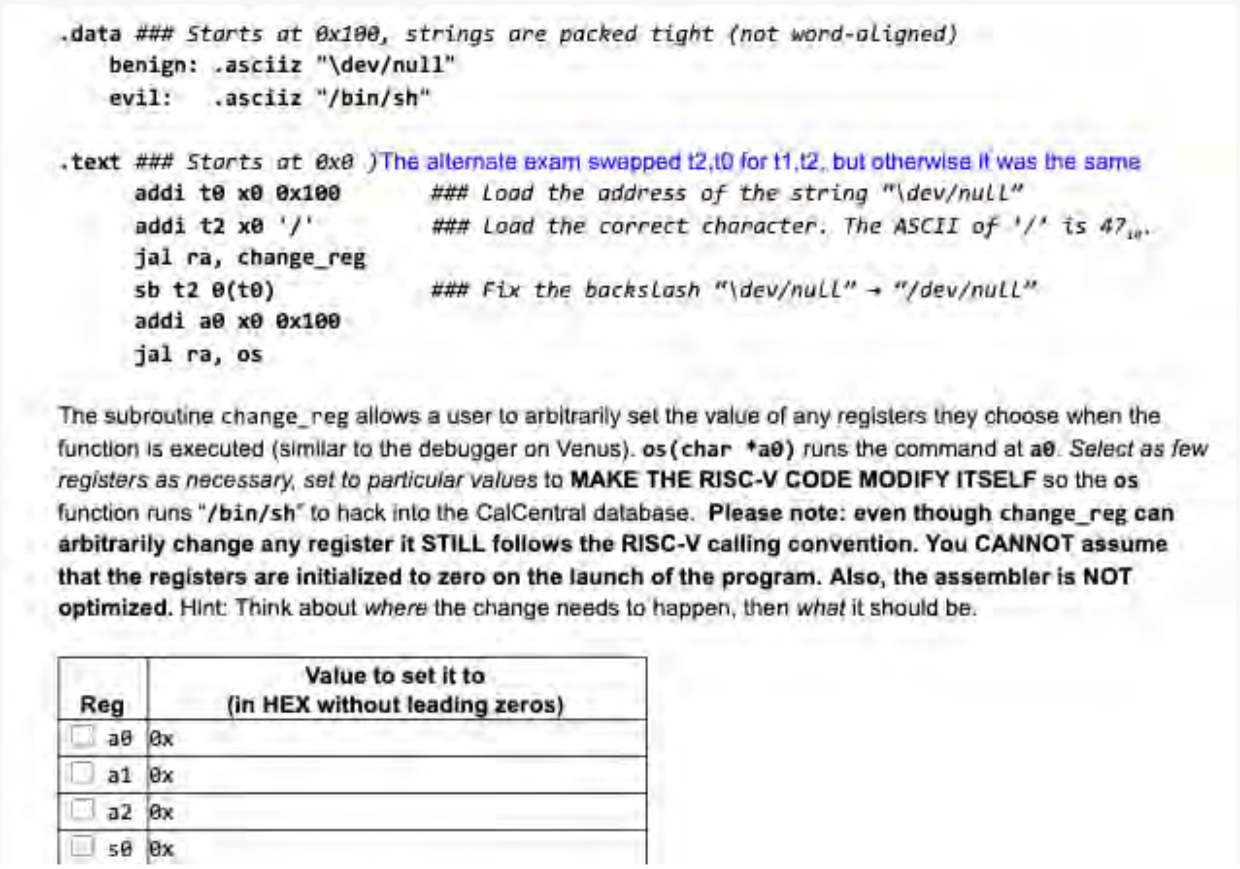
Anonymous Beaker 5 months ago
so if the branch was at line 13 and the label is at line 21 is our offset just 8?
helpful! | 0

Peyrin Kao 5 months ago
Offsets are measured in bytes, so an offset of 8 instructions corresponds to 32 bytes.
good comment | 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f10

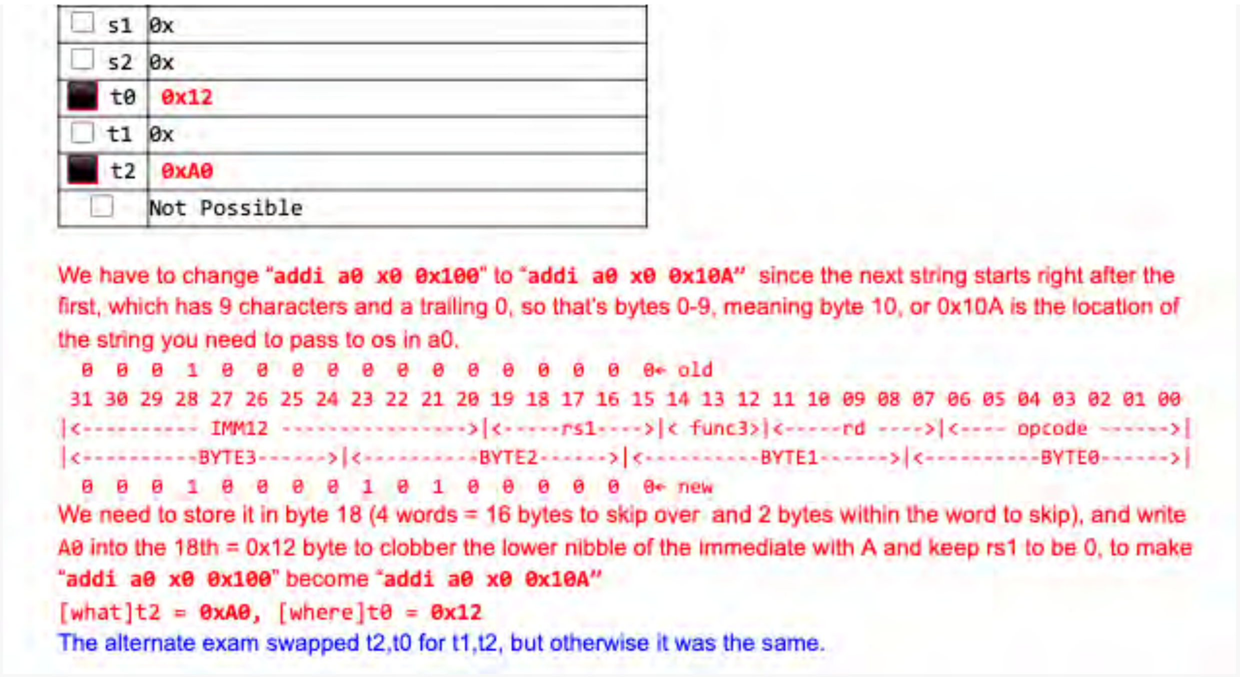
Anonymous Comp 5 months ago
[fa18-final-q3] I'm really confused about this question. I don't understand what it's trying to achieve and also the solution


```
.data ### Starts at 0x100, strings are packed tight (not word-aligned)
    benign: .asciiz "\dev/null"
    evil:   .asciiz "/bin/sh"

.text ### Starts at 0x0 )The alternate exam swapped t2,t0 for t1,t2, but otherwise it was the same
      addi t0 x0 0x100        ### Load the address of the string "\dev/null"
      addi t2 x0 '/'          ### Load the correct character. The ASCII of '/' is 47₁₀.
      jal ra, change_reg
      sb t2 0(t0)             ### Fix the backslash "\dev/null" → "/dev/null"
      addi a0 x0 0x100
      jal ra, os
```

The subroutine change_reg allows a user to arbitrarily set the value of any registers they choose when the function is executed (similar to the debugger on Venus). `os(char *a0)` runs the command at `a0`. *Select as few registers as necessary, set to particular values* to **MAKE THE RISC-V CODE MODIFY ITSELF** so the `os` function runs "`/bin/sh`" to hack into the CalCentral database. **Please note: even though change_reg can arbitrarily change any register it STILL follows the RISC-V calling convention. You CANNOT assume that the registers are initialized to zero on the launch of the program. Also, the assembler is NOT optimized.** Hint: Think about *where* the change needs to happen, then *what* it should be.

Reg	Value to set it to (in HEX without leading zeros)
☐ a0	0x
☐ a1	0x
☐ a2	0x
☐ s0	0x



☐	s1	0x
☐	s2	0x
■	t0	0x12
☐	t1	0x
■	t2	0xA0
☐		Not Possible

We have to change "**addi a0 x0 0x100**" to "**addi a0 x0 0x10A**" since the next string starts right after the first, which has 9 characters and a trailing 0, so that's bytes 0-9, meaning byte 10, or 0x10A is the location of the string you need to pass to os in a0.

```
  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0← old
 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
|<---------- IMM12 ---------------->|<-----rs1---->|< func3>|<-----rd ---->|<---- opcode ------>|
|<----------BYTE3------>|<----------BYTE2------>|<----------BYTE1------>|<----------BYTE0------>|
  0  0  0  1  0  0  0  0  1  0  1  0  0  0  0  0  0← new
```

We need to store it in byte 18 (4 words = 16 bytes to skip over and 2 bytes within the word to skip), and write A0 into the 18th = 0x12 byte to clobber the lower nibble of the immediate with A and keep rs1 to be 0, to make "**addi a0 x0 0x100**" become "**addi a0 x0 0x10A**"

`[what]t2 = 0xA0, [where]t0 = 0x12`

The alternate exam swapped t2,t0 for t1,t2, but otherwise it was the same.

helpful! | 0

Adelson Chua 5 months ago

Oh wow. This problem is very complicated.

The goal is to make the provided code overwrite itself such that instead of doing:

```
...
addi a0 x0 0x100
jal ra, os
```

We do instead:

```
...
addi a0 x0 0x10A
jal ra, os
```

This modification will force a0 to point to the string "/bin/sh", which is what the problem wants you to do.

The string "/bin/sh" is located in the following declaration:

```
.data ### Starts at 0x100, strings are packed tight (not word-aligned)
benign: .asciiz "\dev/null"
evil: .asciiz "/bin/sh"
```

run code snippet Visit Manage Class to disable runnable code snippets ×

Initially, the code just sets a0 to be 0x100, which is the starting address of string "\dev/null", but now we want to point to "/bin/sh" for the 'hack'.

The provided explanation says that the starting address of "/bin/sh" is 10 bytes away from the starting address of "\dev/null".

You can count the number of bytes, including the null terminator, of the string "\dev/null" to see that it is indeed 10 bytes long.

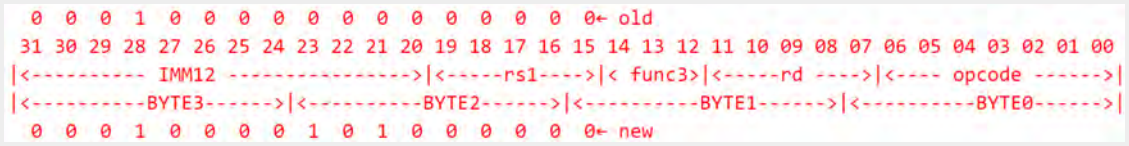
Thus, the next goal is to do the modifications by changing the **sb t2 0(t0)** (which is executed after the call to change_reg) so that it modifies the **addi a0 x0 0x100** instruction and change it to **addi a0 x0 0x10A**.

The next step would be to know what t2 and t0 should be (since those are the registers used in the sb instruction) to do the necessary modifications.

We should set t0 as the address of the addi a0 x0 0x100 instruction that we want to modify.

We should set t2 as the modification that we need to apply.

The explanation then proceeds to write out the instruction encoding for the addi instruction.



```
  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0← old
 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
|<---------- IMM12 ---------------->|<-----rs1---->|< func3>|<-----rd ---->|<---- opcode ------>|
|<----------BYTE3------>|<----------BYTE2------>|<----------BYTE1------>|<----------BYTE0------>|
  0  0  0  1  0  0  0  0  1  0  1  0  0  0  0  0  0← new
```

We primarily want to change the immediate field. We want to change the last 4 bits (last byte) of the immediate to be 0xA.

Looking at the encoding, we can do this by modifying BYTE2. The upper 4 bits of BYTE2 will be set to 0xA, and the lower 4 bits will be set to 0x0 (since rs1 for the addi instruction is x0).

Now, we need to find the location for BYTE2.

Looking at the .text section of the code:

```
.text ### Starts at 0x0
addi t0 x0 0x100 ### Load the address of the string “\dev/null”
addi t2 x0 '/' ### Load the correct character. The ASCII of ’/’ is 47 10 .
jal ra, change_reg
sb t2 0(t0) ### Fix the backslash “\dev/null” → “/dev/null”
addi a0 x0 0x100
```

run code snippet Visit Manage Class to disable runnable code snippets ×

The addi instruction that we want to modify is the 5th instruction. There are 4 instructions before that. Every instruction is 4 bytes. The very first instruction is at address 0x0 (as indicated in the comment).

4 instructions * 4 bytes = 16 bytes.

Then we add 2 more bytes so that we modify BYTE2 of the 5th instruction (the addi instruction that we are targetting).

16 + 2 = 18 = 0x12 in hex (this is shown in the explanation).

Thus, we have the following conclusion. To modify the **addi a0 x0 0x100** to be **addi a0 x0 0x10A** instead, we need to set it up such that the sb t2 0(t0) instruction will be writing 0xA0 to the address 0x12.

Thus, t2 = 0xA0 and t0 = 0x12.

good comment 1

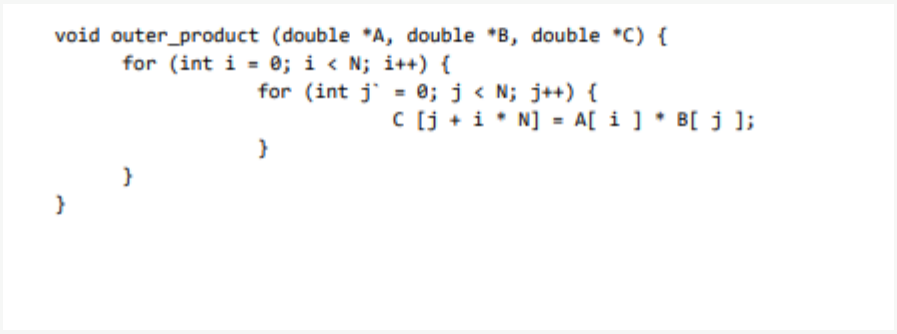
Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f11

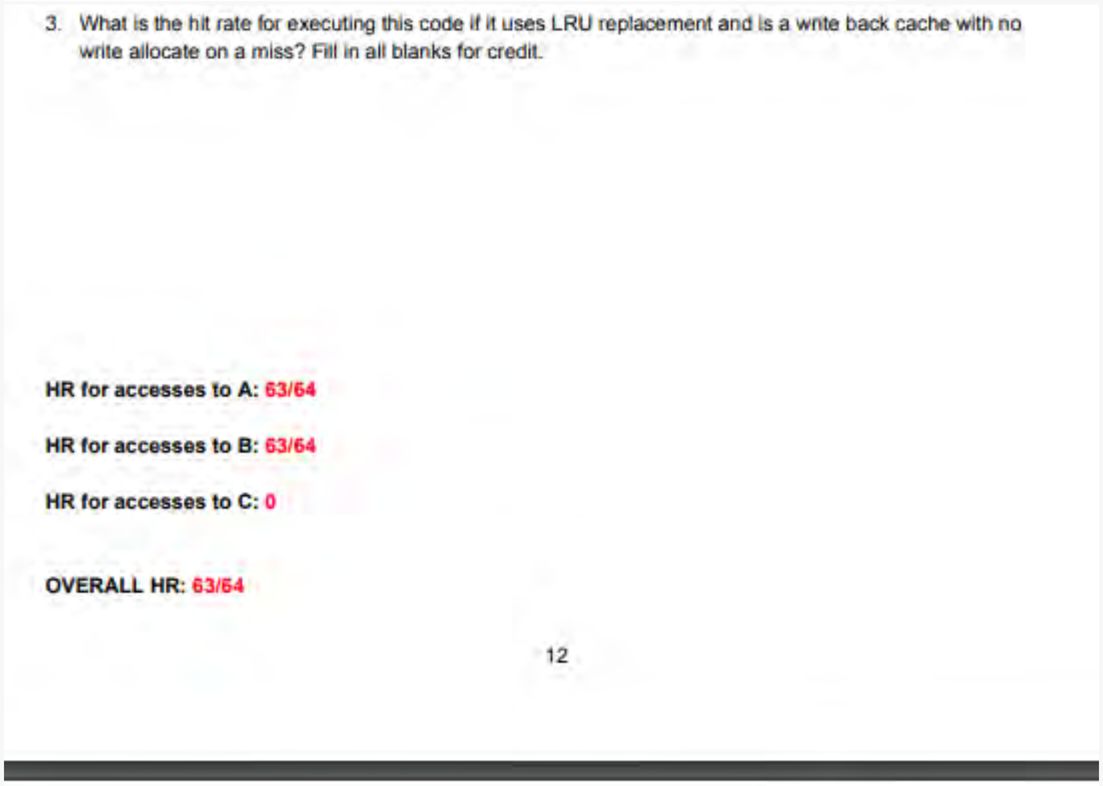
Anonymous Beaker 5 months ago

Su 18 Question 6 Final:

Why do we completely ignore C with this policy?



```
void outer_product (double *A, double *B, double *C) {
      for (int i = 0; i < N; i++) {
              for (int j` = 0; j < N; j++) {
                      C [j + i * N] = A[ i ] * B[ j ];
              }
      }
}
```



3. What is the hit rate for executing this code if it uses LRU replacement and is a write back cache with no write allocate on a miss? Fill in all blanks for credit.

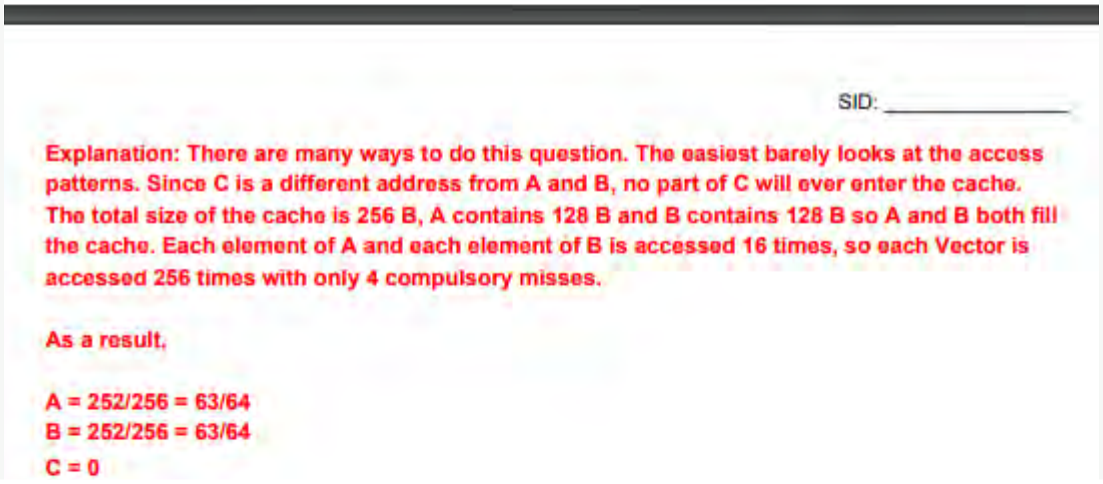
HR for accesses to A: 63/64

HR for accesses to B: 63/64

HR for accesses to C: 0

OVERALL HR: 63/64

12



SID: ________________

Explanation: There are many ways to do this question. The easiest barely looks at the access patterns. Since C is a different address from A and B, no part of C will ever enter the cache. The total size of the cache is 256 B, A contains 128 B and B contains 128 B so A and B both fill the cache. Each element of A and each element of B is accessed 16 times, so each Vector is accessed 256 times with only 4 compulsory misses.

As a result,

A = 252/256 = 63/64
B = 252/256 = 63/64
C = 0

helpful! | 0

Adelson Chua 5 months ago

The write policy is "no write allocate", meaning once we miss on a write, we don't pull the data into the cache. Therefore, we will miss again on the next successive writes.

good comment | 1

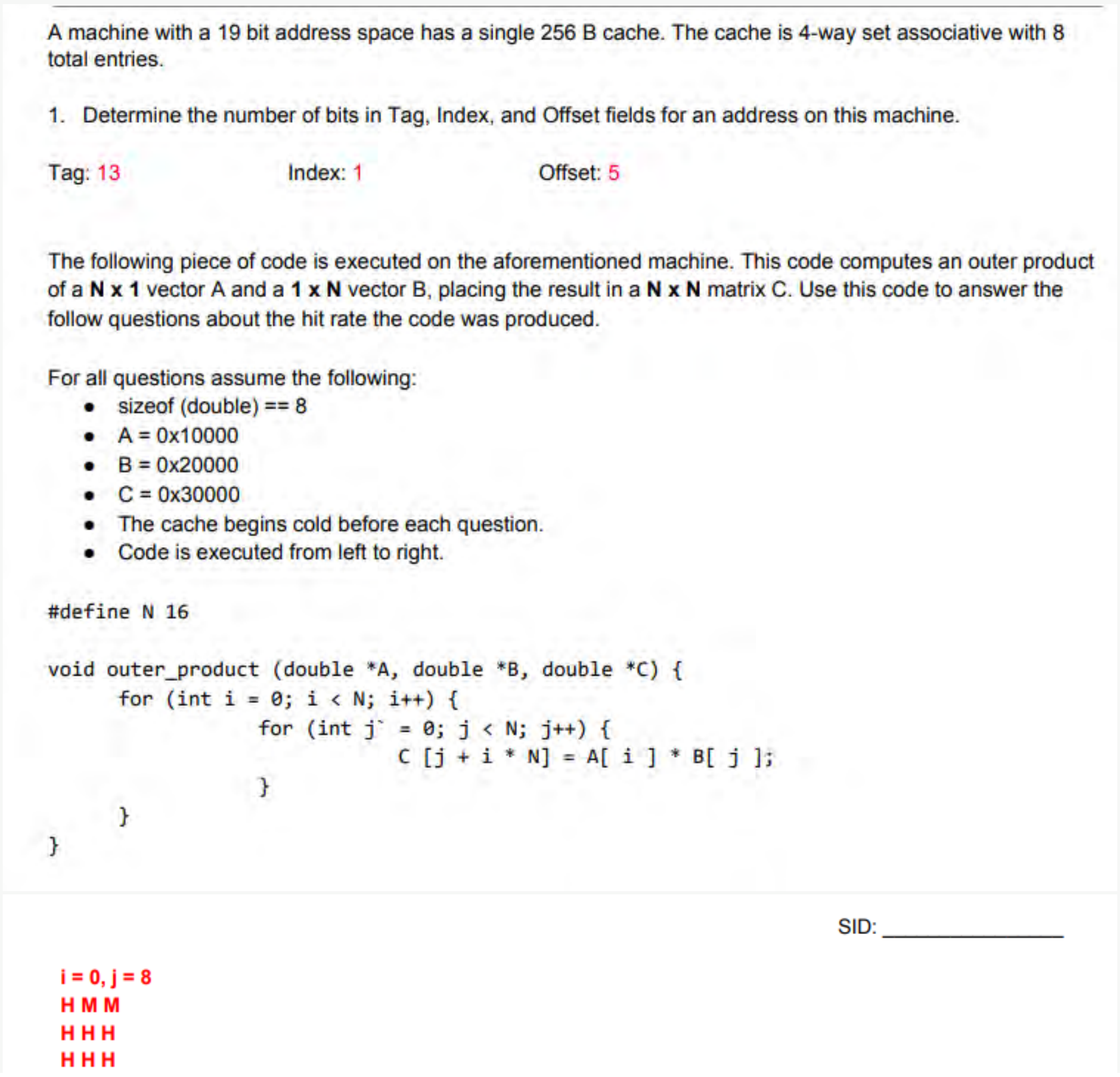
Reply to this followup discussion

[✓ Resolved] Unresolved **@2125_f12**

Anonymous Beaker 5 months ago

SU 18 Final Q6:

I am little confused on how the hit rate and evictions work for B in this loop. Right now I understand that C will need to load a new block every 4 iterations because our block size fits 4 doubles at a time, so in the inner loop we hit 12/16, everytime. But B is only missing twice after the first inner loop is what confuses me.



A machine with a 19 bit address space has a single 256 B cache. The cache is 4-way set associative with 8 total entries.

1. Determine the number of bits in Tag, Index, and Offset fields for an address on this machine.

Tag: 13　　　Index: 1　　　Offset: 5

The following piece of code is executed on the aforementioned machine. This code computes an outer product of a **N x 1** vector A and a **1 x N** vector B, placing the result in a **N x N** matrix C. Use this code to answer the follow questions about the hit rate the code was produced.

For all questions assume the following:

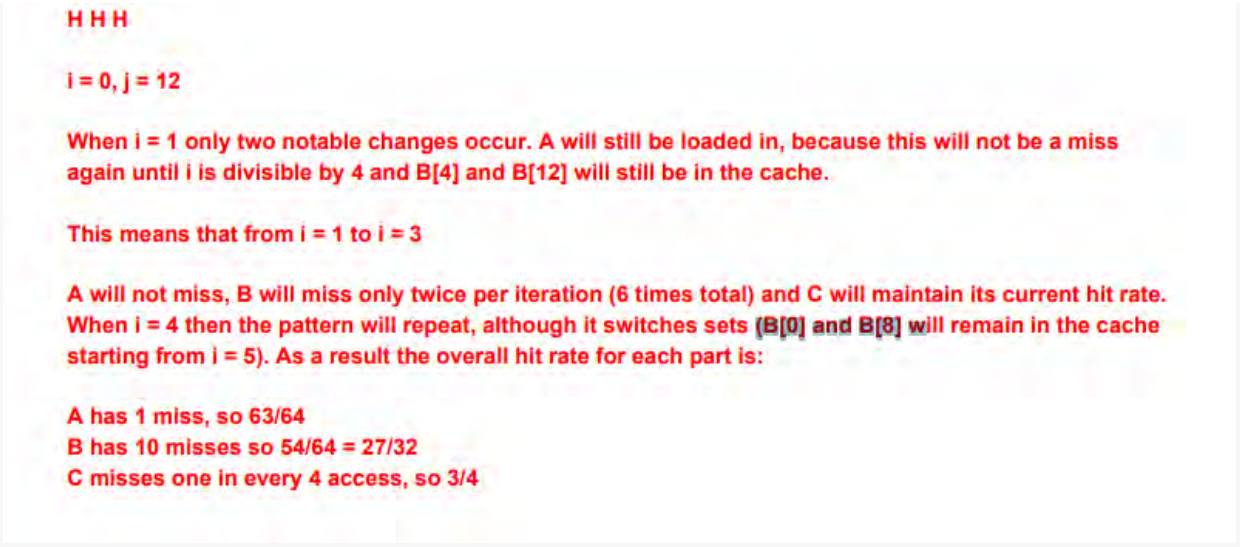
- sizeof (double) == 8
- A = 0x10000
- B = 0x20000
- C = 0x30000
- The cache begins cold before each question.
- Code is executed from left to right.

```
#define N 16

void outer_product (double *A, double *B, double *C) {
      for (int i = 0; i < N; i++) {
                  for (int j` = 0; j < N; j++) {
                              C [j + i * N] = A[ i ] * B[ j ];
                  }
      }
}
```

SID: ________________

i = 0, j = 8
H M M
H H H
H H H



H H H

i = 0, j = 12

When i = 1 only two notable changes occur. A will still be loaded in, because this will not be a miss again until i is divisible by 4 and B[4] and B[12] will still be in the cache.

This means that from i = 1 to i = 3

A will not miss, B will miss only twice per iteration (6 times total) and C will maintain its current hit rate. When i = 4 then the pattern will repeat, although it switches sets (B[0] and B[8] will remain in the cache starting from i = 5). As a result the overall hit rate for each part is:

A has 1 miss, so 63/64
B has 10 misses so 54/64 = 27/32
C misses one in every 4 access, so 3/4

helpful! | 0

Adelson Chua 5 months ago
At this point, we should be drawing what is happening in the cache...

i=0, j=0

0	**A[0]**	**B[0]**	**C[0]**	-
1	-	-	-	-

i=0, j=4

0	A[0]	B[0]	C[0]	-
1	**B[4]**	**C[4]**	-	-

i=0, j=8

0	A[0]	**C[8]**	C[0]	**B[8]**
1	B[4]	C[4]	-	-

i=0, j=12

0	A[0]	C[8]	C[0]	B[8]
1	B[4]	C[4]	**B[12]**	**C[12]**

i=1, j=0

0	A[0]	C[8]	**B[0]**	**C[0]**
1	B[4]	C[4]	B[12]	C[12]

i=1, j=4

0	A[0]	C[8]	B[0]	C[0]

1	B[4]	C[4]	B[12]	C[12]

You see how at this point, B[4] is actually already in the cache? B[12] also remains in the cache. So for j=4 and j=12, there are no misses on B.

Just extend this diagram for the other iterations. Always take note of the LRU eviction policy when overwriting entries.

good comment | 1

Reply to this followup discussion

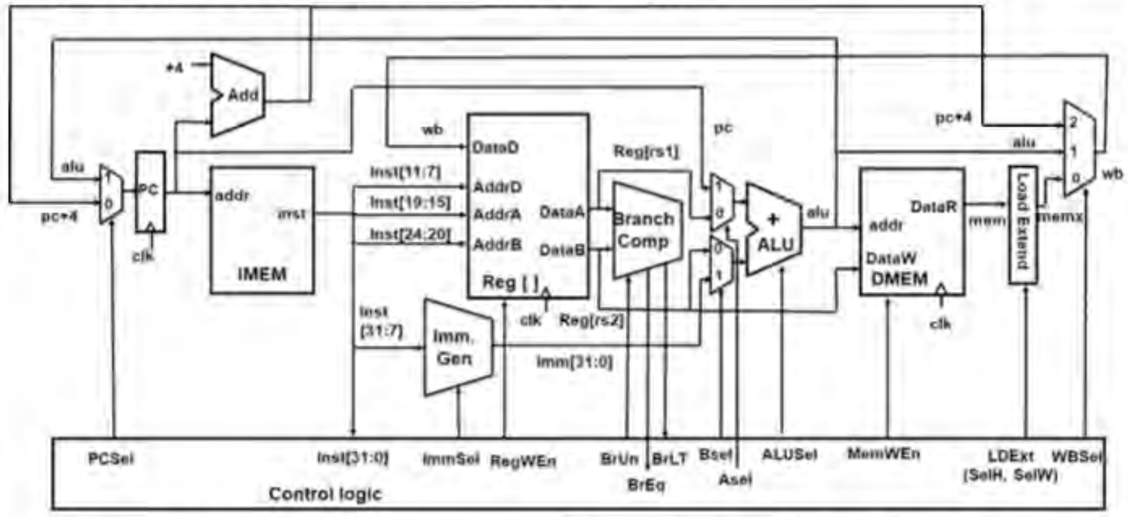
[✓ Resolved] Unresolved @2125_f13

Anonymous Comp 5 months ago

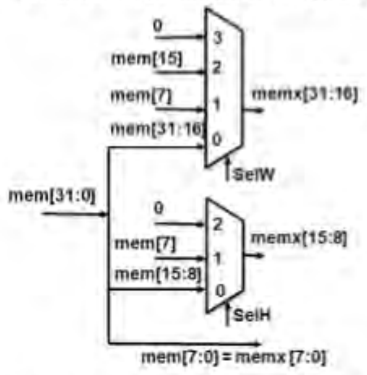
[fa18-final-f3a] I'm a bit confused about how this question works

F3) *Datapathology* [this is a 2-page question] (20 points = 4+10+6, 30 minutes)

The datapath below implements the RV32I instruction set. We'd like to implement sign extension for loaded data, but our loaded data can come in different sizes (recall: `lb`, `lh`, `lw`) and different intended signs (`lbu` vs. `lb` and `lhu` vs `lh`). Each load instruction will retrieve the data from the memory and "right-aligns" the LSB of the byte or the half-word with the LSB of the word to form `mem[31:0]`.



a) To correctly load the data into the registers, we've created two control signals **SelH** and **SelW** that perform sign extension of `mem[31:0]` to `memx[31:0]` (see below). **SelH** controls the half-word sign extension, while **SelW** controls sign extension in the two most significant bytes. What are the Boolean logic expressions for the four (0, 1, 2, 3) **SelW** cases in terms of Inst[14:12] bits to handle these five instructions (`lb`, `lh`, `lw`, `lbu` and `lhu`)? **SelH** has been done for you. In writing your answers, use the shorthands "I14" for Inst[14], "I13" for Inst[13] and "I12" for Inst[12]. You don't have to reduce the Boolean expressions to simplest form. (Hint: green card!) Answers (and simplified form)



SelW=3	**I14 • ~I13 • ~I12 + I14 • ~I13 • I12 = I14**
SelW=2	**~I14 • ~I13 • I12 = ~I14 • I12**
SelW=1	**~I14 • ~I13 • ~I12**
SelW=0	**~I14 • I13 • ~I12 = I13**
SelH=2	I14 • ~I13 • ~I12
SelH=1	~I14 • ~I13 • ~I12
SelH=0	I13 + I12

(Single-bit values `mem[7]` and `mem[15]` are wired to 8 or 16 outputs)

helpful! | 0

Adelson Chua 5 months ago

For every load instruction variant, identify the needed values for SelH and SelW. (I know SelH is already given, but I'm doing it for completeness sake).

lb (load byte, signed). The output for this one should be {mem[7], mem[7], mem[7:0]}. Basically sign-extension of the byte in mem[7:0]. **SelH = 1, SelW = 1.**

lbu (load byte, unsigned). The output for this one should be {0, 0, mem[7:0]}. Basically zero-extension of the byte in mem[7:0]. **SelH = 2, SelW = 3.**

lh (load half, signed). The output for this one should be {mem[15], mem[15:8], mem[7:0]}. Basically sign-extension of the half-word in mem[15:0]. **SelH = 0, SelW = 2.**

lhu (load half, unsigned). The output for this one should be {0, mem[15:8], mem[7:0]}. Basically zero-extension of the half-word in mem[15:0]. **SelH = 0, SelW = 3.**

lw (load word). The output for this one should be {mem[31:16], mem[15:8], mem[7:0]}. Basically the entire word in mem[31:0]. **SelH = 0, SelW = 0.**

Rearrange so that we are now checking for each SelH and SelW values. For every SelH and SelW values, identify the instructions that need them.

SelH=0, lh lhu lw

SelH=1, lb

SelH=2, lbu

SelW=0, lw

SelW=1, lb

SelW=2, lh

SelW=3, lbu lhu

Now let's focus on SelW, also write the corresponding funct3 for every instruction (check the reference card).

Note that the funct3 is the instruction bits 14, 13, 12. The problem refers to them as I14, I13, I12. Write the boolean expression accordingly.

SelW=0, lw (funct3 = **010**). **~I14 * I13 * ~I12**

SelW=1, lb (funct3 = **000**). **~I14 * ~I13 * ~I12**

SelW=2, lh (funct3 = **001**). **~I14 * ~I13 * I12**

SelW=3, lbu (funct3 = **100**), lhu (funct3 = **101**). **I14 * ~I13 * ~I12 + I14 * ~I13 * I12**

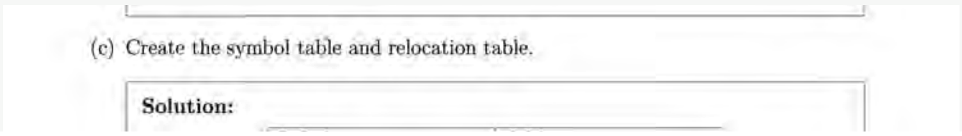
good comment | 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f14

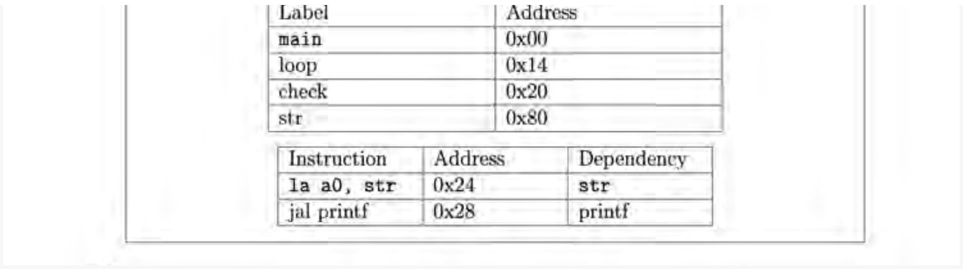
Anonymous Gear 5 months ago

[SP 18 Final Q4C]



(c) Create the symbol table and relocation table.

Solution:



Label	Address
main	0x00
loop	0x14
check	0x20
str	0x80

Instruction	Address	Dependency
la a0, str	0x24	str
jal printf	0x28	printf

Could anyone explain why the la and jal instructions are in the relocation table while the ret and j check are not?

helpful! | 0

Anonymous Comp 5 months ago
Because str and printf are not part of the code, which means that we have to find the absolute address of those two later

helpful! | 0

Anonymous Gear 5 months ago
I see. Is it because str is in the data section?

helpful! | 0

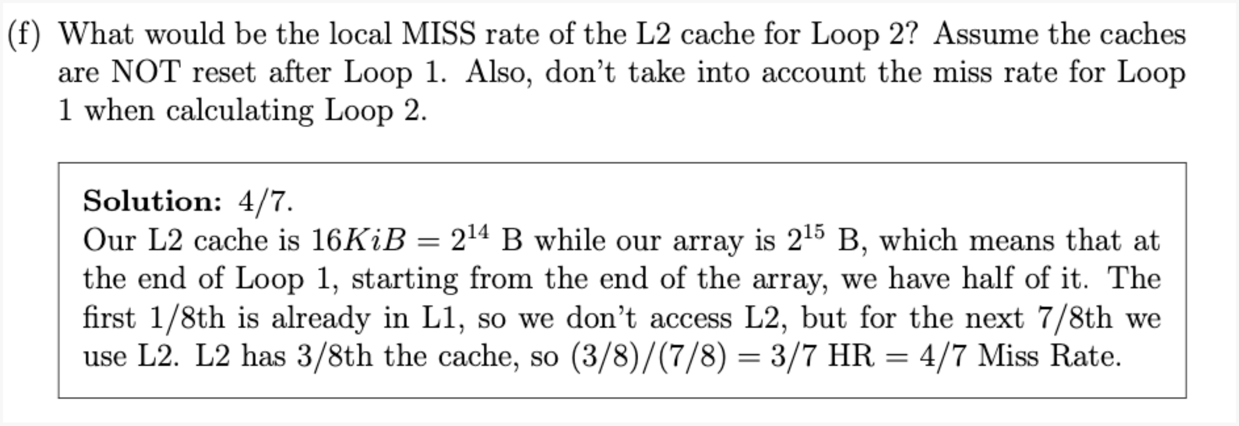
Adelson Chua 5 months ago
Correct.

good comment | 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f15

Anonymous Comp 5 months ago
[sp18-final-q7f] why L2 has 3/8 of the cache?



(f) What would be the local MISS rate of the L2 cache for Loop 2? Assume the caches are NOT reset after Loop 1. Also, don't take into account the miss rate for Loop 1 when calculating Loop 2.

Solution: 4/7.
Our L2 cache is $16KiB = 2^{14}$ B while our array is 2^{15} B, which means that at the end of Loop 1, starting from the end of the array, we have half of it. The first 1/8th is already in L1, so we don't access L2, but for the next 7/8th we use L2. L2 has 3/8th the cache, so $(3/8)/(7/8) = 3/7$ HR $= 4/7$ Miss Rate.

helpful! | 0

Adelson Chua 5 months ago
It was stated that since L2 size is half the array size, L2 cache will contain the second half of the array after loop 1. This means 4/8 of the array is in it.
However, 1/8 is already in the L1 cache and we are hitting it, so we don't access L2 during that time.
We only access L2 after the 1st 1/8 of the array was traversed. But since L2 has 4/8 of the array,

4/8 - 1/8 = 3/8 will be hits.

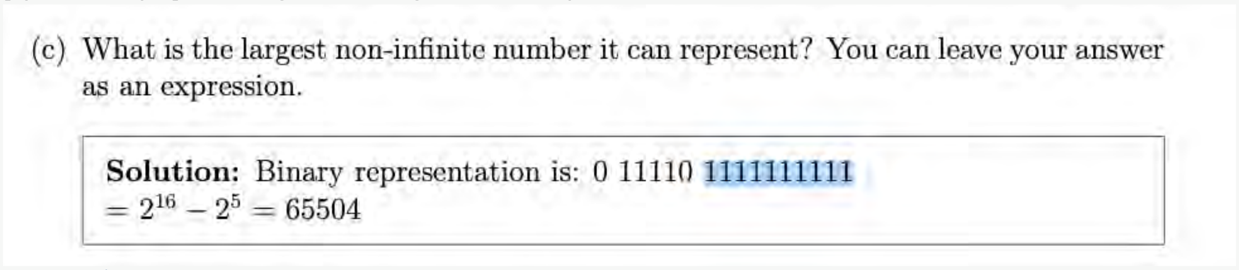
good comment | 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f16

Anonymous Comp 5 months ago

[sp18-final q9c] How to get 2^16? I got 2^5 for exponent, but not sure what the 2^16 means



(c) What is the largest non-infinite number it can represent? You can leave your answer as an expression.

Solution: Binary representation is: 0 11110 1111111111 $= 2^{16} - 2^{5} = 65504$

helpful! | 0

Adelson Chua 5 months ago

Given the binary representation 0 11110 1111111111

Rewriting into the 'scientific notation', we get 1.1111111111 * 2^(11110 -15) => 1.1111111111 * 2^(30-15) => 1.1111111111 * 2^(15) (Note that this is already probably a good enough solution)

Now, we can rewrite this in a simpler form using only powers of 2.

1.1111111111 is basically 10 - 0.0000000001 in binary.

Appending the exponent, we get:

10 * 2^15 - 0.0000000001 * 2^15

Shifting the radix point so that we are just left with the form 1 * 2^x:

10 * 2^15 => 1 * 2^16 (we moved the radix point to the left once, adding 1 exponent)

0.0000000001 * 2^15 => 2^5 (we moved the radix point to the right 10 times, subtracting 10 exponent)

Thus, we get the final answer:

2^16 - 2^5

good comment | 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f17

Anonymous Gear 5 months ago

[sp18-final-q6]

For part b , why is it 3 stall? I am thinking about we do not need stalling after 1 since we have write-read. We need 2 stalls after line 3 since we need to do WB for instruction 3 and ID for instruction 4 in the same stage, same for after line 4, we need 2 stalls. But the answer is 3 which I'm not sure how we get 3.

helpful! | 0

Anonymous Comp 5 months ago

1 stall after instruction #2 to align write and read in instruction 1 (ID for #3 align with WB for #1). 2

stalls after instruction 3 for the same reason. ID for #4 needs to align with WB for #3

helpful! | 0

Anonymous Comp 5 months ago

Oh nevermind I misunderstood the question

helpful! | 0

Anonymous Gear 5 months ago

So we don't need to account for between instruction #4 and #5?

helpful! | 0

Adelson Chua 5 months ago

Got it. I think the solution *does not* consider the needed stall due to #4 and #5.

For a write-read implementation, we get 3 total stalls. One for #1-#3 dependency, two for #3-#4 dependency.

slli **t0** a1 2	F	D	X	M	**W**					
or t2 a1 t1		F	D	X	M	W				
add **t0** **t0** a0			F	**D**	**D**	X	M	W		
lw t1 4(**t0**)				F	F	**D**	**D**	**D**	X	M

For read-write implementation, we need 1 more stall for each of the pairs since the D stage should come 1 cycle after the WB, giving us 5 total stalls.

good comment | 1

Anonymous Mouse 5 months ago

Why we don't count the stall for IF? for #4?

helpful! | 0

Adelson Chua 5 months ago

That stall was a consequence of the ID stall from #3, it happened at the same time. It is not an 'additional' stall.

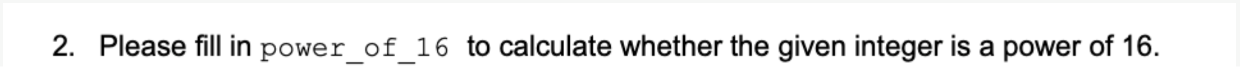
good comment | 0

Reply to this followup discussion

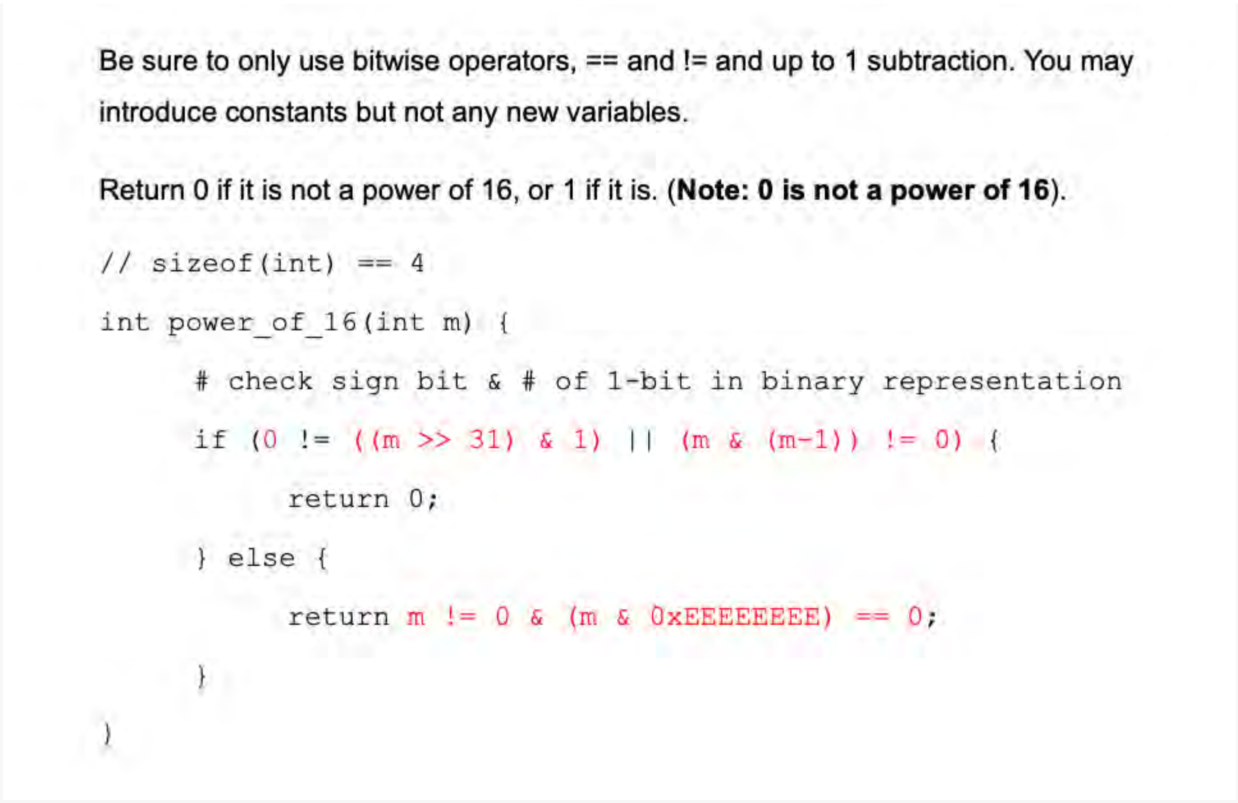
[✓ Resolved] Unresolved @2125_f18

Anonymous Comp 5 months ago

[fa17-final-q1.2] I'm not sure how to come up with this second return statement if I'm doing this question in exam. Can someone explain the thought process of coming up with this solution?



2. Please fill in `power_of_16` to calculate whether the given integer is a power of 16.


Be sure to only use bitwise operators, == and != and up to 1 subtraction. You may introduce constants but not any new variables.

Return 0 if it is not a power of 16, or 1 if it is. (**Note: 0 is not a power of 16**).

```
// sizeof(int) == 4
int power_of_16(int m) {
    # check sign bit & # of 1-bit in binary representation
    if (0 != ((m >> 31) & 1) || (m & (m-1)) != 0) {
        return 0;
    } else {
        return m != 0 & (m & 0xEEEEEEEE) == 0;
    }
}
```

helpful! | 0

Adelson Chua 5 months ago
In binary, a power of 16 will be will always follow the pattern:

16^0 = 0x00000001

16^1 = 0x00000010

16^2 = 0x00000100

...

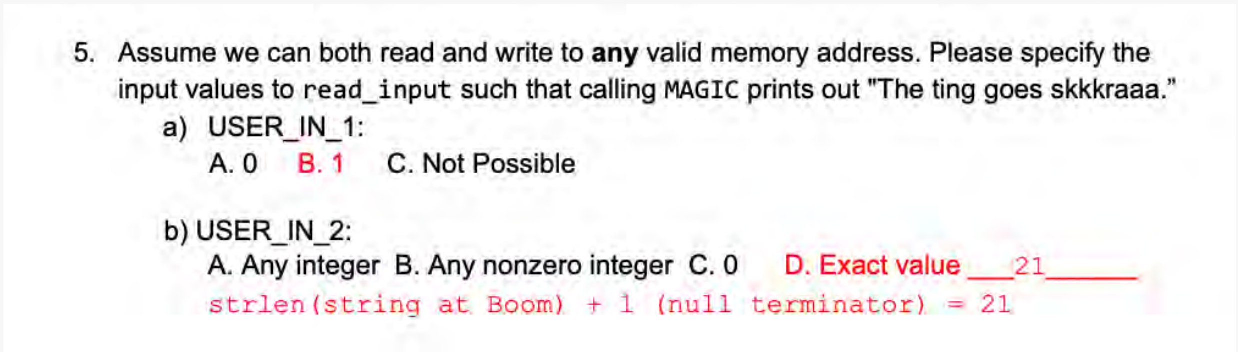
Are you seeing the pattern now?
ANDing with strings of 0xE... checks for the presence of that '1'.

good comment | 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f19

Anonymous Comp 5 months ago
[fa17-final-q4.5b] Can somebody explain how to get part b? thanks!


5. Assume we can both read and write to **any** valid memory address. Please specify the input values to `read_input` such that calling `MAGIC` prints out "The ting goes skkkraaa."

a) USER_IN_1:
   A. 0   B. 1   C. Not Possible

b) USER_IN_2:
   A. Any integer B. Any nonzero integer C. 0   D. Exact value ___21______

```
strlen(string at Boom) + 1 (null terminator) = 21
```

helpful! | 0

Peyrin Kao 5 months ago

```
12. Risc-tery: jal read_input
13.   beq a0, x0, QuickMaths    # branch not taken in Q4.5
14.   addi t0, x0, 9            # t0 = 9
15.   slli t0, t0, 2            # t0 = 36
16.   add s0, s0, t0            # s0 = address of line 21 of code (address
of RISC-tery + 9 instructions)
17.   lw t1, 0(s0)              # load the instruction at line 21 into t1
18.   slli a0, a0, 20           # shift user input by 20
19.   add t1, t1, a0            # add user input a0 (shifted by 20) to the
instruction (in t1)
20.   sw t1, 0(s0)              # store the modified instruction back in me
mory
21. QuickMaths: addi a1, s1, 0
22.   addi a0, x0, 4
23.   ecall
24.   j Done
```

run code snippet | Visit Manage Class to disable runnable code snippets

The instruction at line 21 is an `addi` instruction, so the immediate is stored at bits 31-20. This code will change that immediate to the number the user inputted and store the changed instruction back into memory.

At line 21, the instruction we want is `addi a1, s1, 21` because that's the address of string we want to print (the address of `Skraa` is 21 bytes after the address of `Boom`, which is in s1).

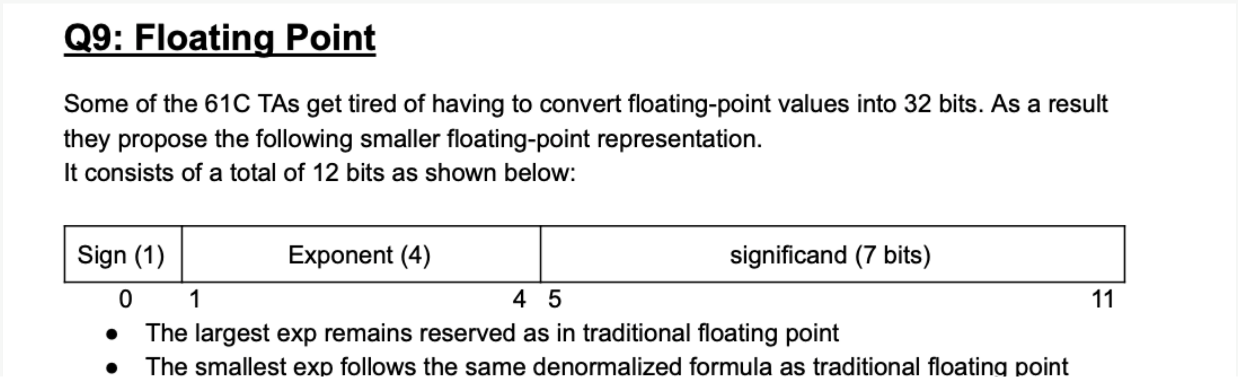
good comment | 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f20

Anonymous Comp 5 months ago

[fa17-final-q9] why is this not 2^-6?



Q9: Floating Point

Some of the 61C TAs get tired of having to convert floating-point values into 32 bits. As a result they propose the following smaller floating-point representation.
It consists of a total of 12 bits as shown below:

Sign (1)	Exponent (4)	significand (7 bits)
0	1 4	5 11

- The largest exp remains reserved as in traditional floating point
- The smallest exp follows the same denormalized formula as traditional floating point

- **Numbers are rounded to the closest representable value. Any numbers that have 2 equidistant representations are rounded down towards zero.**

All of the parts of this question refer to this floating-point scheme.

1. What is the smallest nonzero positive value that can be represented? Write your answer as a numerical expression in the answer packet.

2^{-13}

helpful! | 0

Peyrin Kao 5 months ago

2^{-6} is the exponent for denorm numbers. You could make the significand 0b0000001, which would create the denorm number $0b0.0000001 \times 2^{-6} = 2^{-7} \times 2^{-6} = 2^{-13}$.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @2125_f21

Anonymous Scale 5 months ago

Problem 5 *[MT2-1] Circuits and Timing* **(9 points)**

In this question, you will be working with a circuit that takes in three 8-bit inputs. For all parts, assume the delays below:

$t_{clk-to-q} = 3\text{ps}$, $t_{setup} = 4\text{ps}$, $t_{shifter} = 1\text{ps}$
$t_{adder} = 5\text{ps}$, $t_{multiplier} = 6\text{ps}$, $t_{subtracter} = 4\text{ps}$

Furthermore, assume that the inputs A, B, and C take on their new values exactly at the rising edge of every clock cycle and that all registers are initialized to zero.

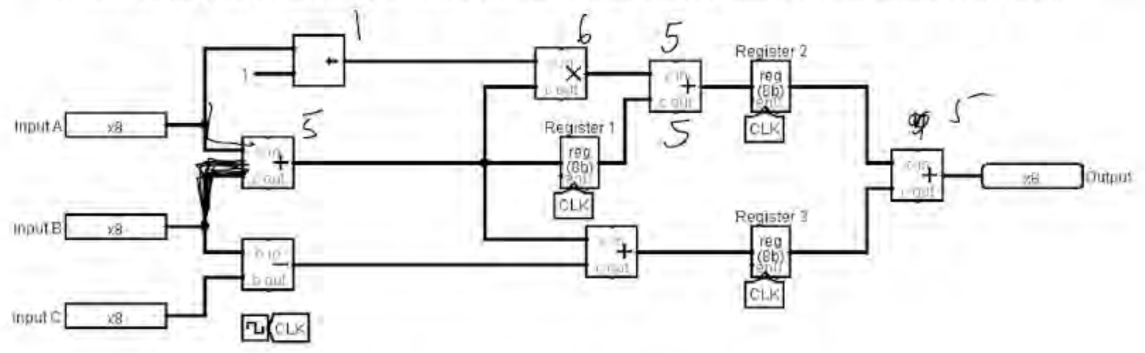


Figure 2: Non-pipelined circuit

(a) What is the maximum possible hold time that still ensures the correctness of the non-pipelined circuit in figure 1? (Select only one)

- 1ps
- 5ps
- 3ps
- [✓ 7ps]
- 4ps

How is the answer 5? Isn't the max hold time = time of shortest combinatorial delay + clock-to-q time?

helpful! | 0

Adelson Chua 5 months ago

The path we are interested in is the path from either input A or B through the multiplier going into the Register 1 input.

Inputs A and B are synchronized to the clock, but they don't have any clk-to-q delay.

good comment | 1

Anonymous Scale 5 months ago

Doesn't the instructions say that each of the inputs has a 3ps clock-to-q delay?

helpful! | 0

Adelson Chua 5 months ago

Those would refer to the registers in the circuit.

Inputs A and B are only synchronous to the clock without any delay, so it is like a register without any clk-to-q.

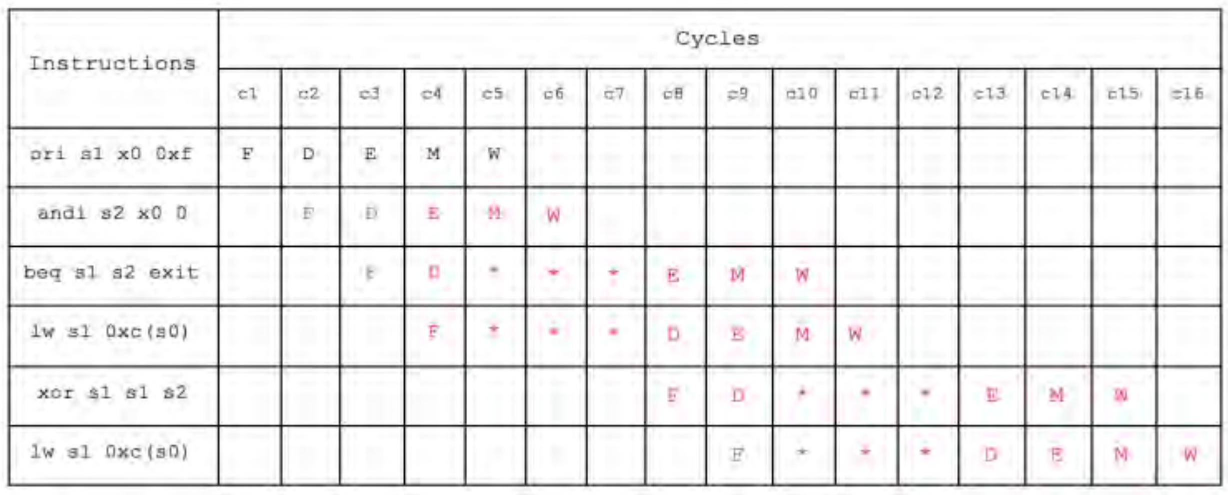
good comment | 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f22

Anonymous Atom 5 months ago

Sp18-mt2-4a



Instructions	Cycles															
	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16
ori s1 x0 0xf	F	D	E	M	W											
andi s2 x0 0		F	D	E	M	W										
beq s1 s2 exit			F	D	*	*	*	E	M	W						
lw s1 0xc(s0)				F	*	*	*	D	E	M	W					
xor s1 s1 s2								F	D	*	*	*	E	M	W	
lw s1 0xc(s0)									F	*	*	*	D	E	M	W

Why does there have to be an extra delay "*" after the W in the beq and xor lines? Why can't we insert the E right after the W finishes?

helpful! | 0

Peyrin Kao 5 months ago

The decode stage of the beq instruction can't happen until after the write-back stage of the andi instruction happens, so at time c7, the D stage of the beq instruction is being run. The same idea applies for the lw and xor instructions.

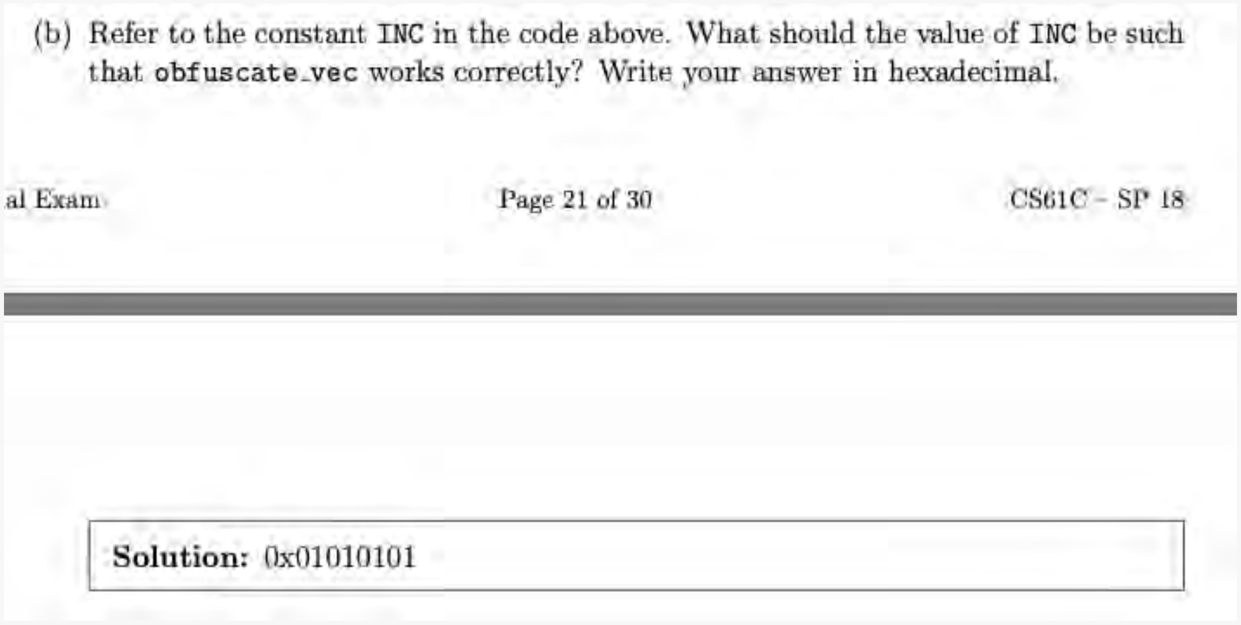
good comment | 0

Reply to this followup discussion

Resolved Unresolved @2125_f23

Anonymous Scale 5 months ago

Sp2018 Final



(b) Refer to the constant `INC` in the code above. What should the value of `INC` be such that `obfuscate_vec` works correctly? Write your answer in hexadecimal.

al Exam Page 21 of 30 CS61C – SP 18

Solution: 0x01010101

How do we get to the solution?

helpful! | 0

Peyrin Kao 5 months ago

In ASCII, adding 1 to an alphanumeric character gets you the next character. For example, `'a' + 1 = 'b'`.

We have four characters packed into a 4-byte integer, and adding `0x01010101` adds 1 to each character.

good comment | 0

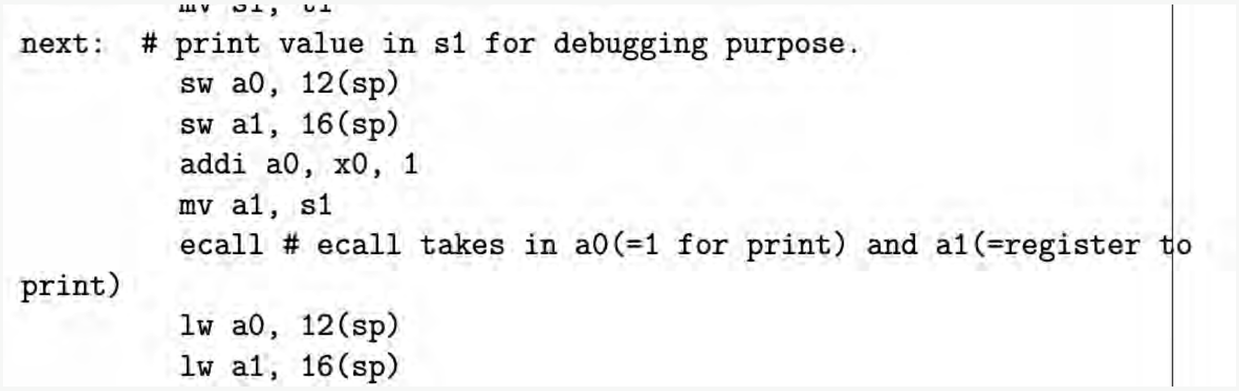
Reply to this followup discussion

Resolved Unresolved @2125_f24

Anonymous Mouse 5 months ago

SP 2018, FINAL P3

Why we need to save a0, a1 in to memory?



```
next:  # print value in s1 for debugging purpose.
         sw a0, 12(sp)
         sw a1, 16(sp)
         addi a0, x0, 1
         mv a1, s1
         ecall # ecall takes in a0(=1 for print) and a1(=register to
print)
         lw a0, 12(sp)
         lw a1, 16(sp)
```

helpful! | 0

Peyrin Kao 5 months ago
To follow calling convention, we have to save a0 and a1 on the stack before the ecall and load them back after the ecall.
good comment | 0

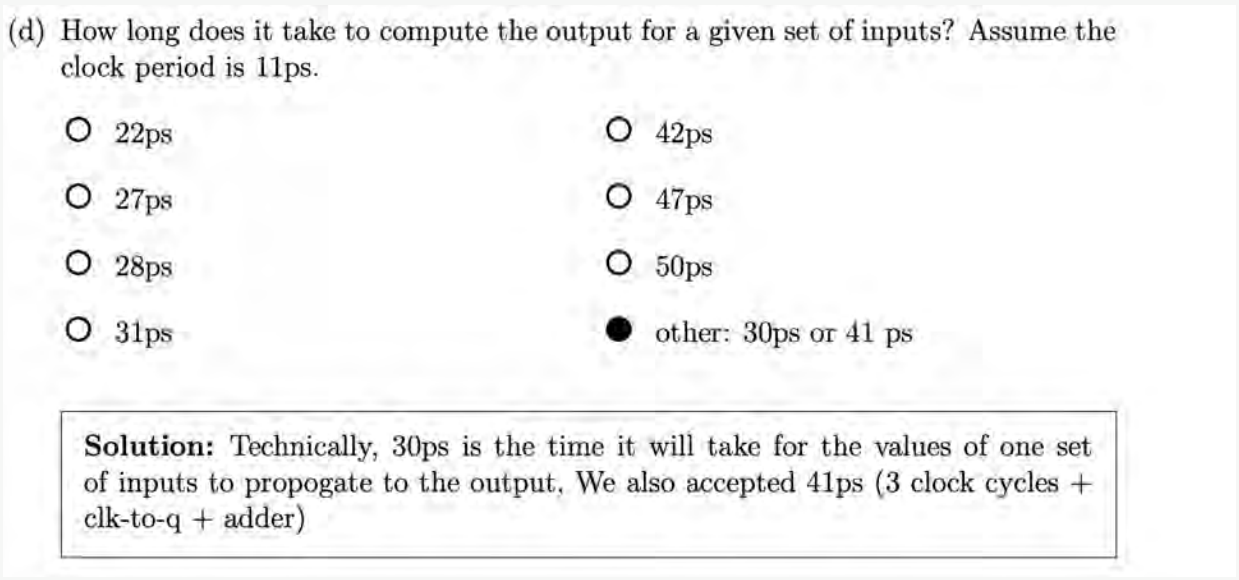
Anonymous Mouse 5 months ago
I thought we can only do it with save registers the ones start with s?
helpful! | 0

Peyrin Kao 5 months ago
Function calls (including ecall) are allowed to change any non-preserved registers, which are all the a registers and t registers.
good comment | 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f25

Anonymous Mouse 5 months ago
SP 18 FINAL P5
I don't understand here how 30 or 41 is calculated.


(d) How long does it take to compute the output for a given set of inputs? Assume the clock period is 11ps.

○ 22ps	○ 42ps
○ 27ps	○ 47ps
○ 28ps	○ 50ps
○ 31ps	[✓ other: 30ps or 41 ps]

Solution: Technically, 30ps is the time it will take for the values of one set of inputs to propogate to the output. We also accepted 41ps (3 clock cycles + clk-to-q + adder)

helpful! | 0

Peyrin Kao 5 months ago
In the non-pipelined circuit: At 0ps, the inputs change and the signal propagates to register 1. At 11ps, the clock ticks and the signal propagates to register 2. At 22ps, the clock ticks. At 25ps (+3ps for clk-to-q time), the signal appears at the output of register 2. At 30ps (+5ps for adder time), the signal appears at the output.

In the pipelined circuit: At 0ps, the inputs change and the signals propagate to the first set of registers (4, 5, 6). At 11ps, the clock ticks and the signals propagate to the next set of registers (1). At 22ps, the clock ticks and the signals propagate to the next set of registers (2, 3). At 33ps, the clock ticks. At 36ps (+3ps for clk-to-q time), the signals appear at the output of the last set of registers (2, 3). At 41ps (+5 for adder time), the signal appears at the output.

good comment | 1

Anonymous Mouse 5 months ago

Why at 0ps, the input changes immediately for register1(non-pipeline), 4,5,6(pipeline), shouldn't we count for one clock cycle for that as well? So they get updated at t = 11? Also, we only add the clk-to-q time for the last stage register instead of counting all registers? For example, clk-to-q time for register 4,5,6(pipeline)

helpful! | 0

Adelson Chua 5 months ago

At **0ps** there is a rising edge of the clock which changed the inputs.

Due to this input change, delays propagate through the combinational logic. However, the delay propagation stop at the input of the register (since it has to wait for the rising edge of the clock to propagate the data)

At **11ps** is the second rising edge of the clock (it was stated in the problem that the clock period is 11ps), so registers (4,5,6) update at this point.

Due to this update, delays propagate again but stop at the input of the next stage of registers (1)

At **22ps** is the third rising edge of the clock, register 1 update.

Delays propagate again stopping at the input next stage registers (2,3).

At **33ps** is the fourth rising edge of the clock, register 2 and 3 updates.

Delays propagate again until now they can reach the output (there are no more registers stopping them). Total delay = clk-to-q + adder delay = 8ps.

In total, you waited for 33ps + 8ps = **41ps** for the output to finally change due to the input provided at time=0ps.

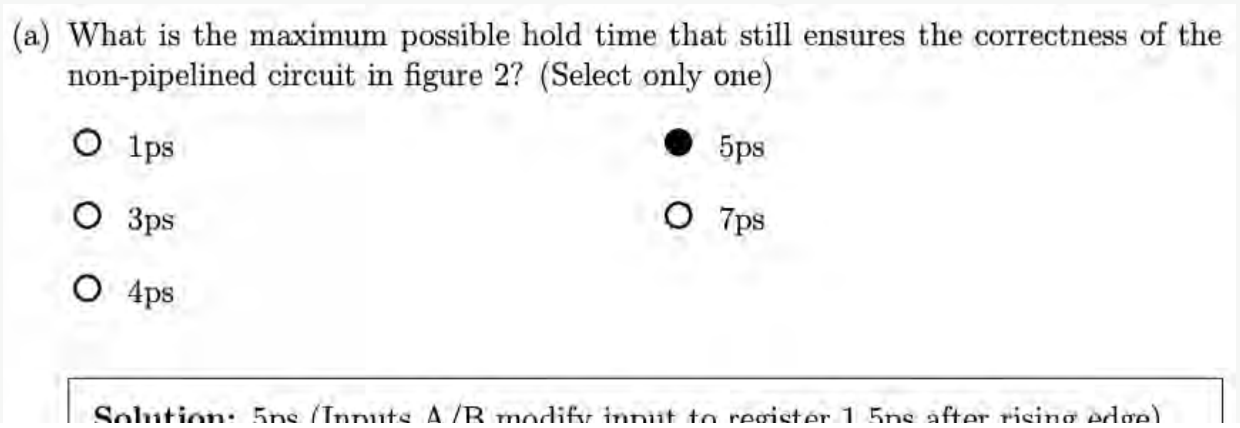
good comment | 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f26

Anonymous Atom 5 months ago

sp18-final-5a and 5b



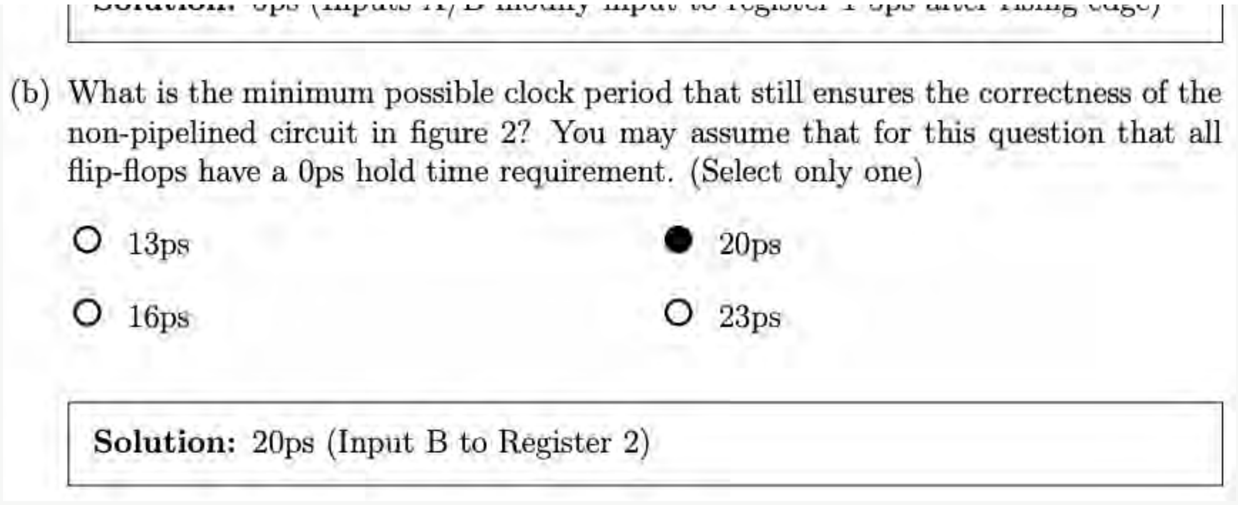
(a) What is the maximum possible hold time that still ensures the correctness of the non-pipelined circuit in figure 2? (Select only one)

○ 1ps [✓ 5ps]

○ 3ps ○ 7ps

○ 4ps

Solution: 5ps (Inputs A/B modify input to register 1 5ps after rising edge)



Solution: 0ps (inputs A/B modify input to register 1 0ps after rising edge)

(b) What is the minimum possible clock period that still ensures the correctness of the non-pipelined circuit in figure 2? You may assume that for this question that all flip-flops have a 0ps hold time requirement. (Select only one)

○ 13ps [✓ 20ps]

○ 16ps ○ 23ps

Solution: 20ps (Input B to Register 2)

Is the reason why we don't include the clock-to-queue time because inputs A and B are not registers that are connected to the clock?

helpful! 0

Peyrin Kao 5 months ago

The question says: "Furthermore, assume that the inputs A, B, and C take on their new values exactly at the rising edge of every clock cycle."

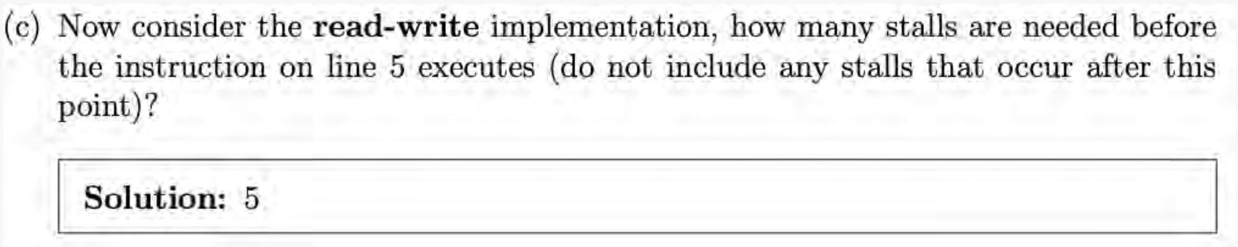
good comment 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f27

Anonymous Atom 5 months ago

sp18-final-6c



(c) Now consider the **read-write** implementation, how many stalls are needed before the instruction on line 5 executes (do not include any stalls that occur after this point)?

Solution: 5

why is the solution not 3? I thought the M and W doesn't matter for beq's since we're only changing the PC

helpful! 0

Adelson Chua 5 months ago

@2125_f17

good comment 0

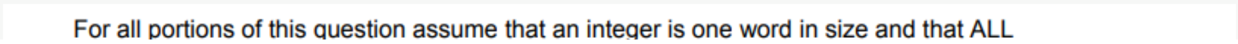
Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f28

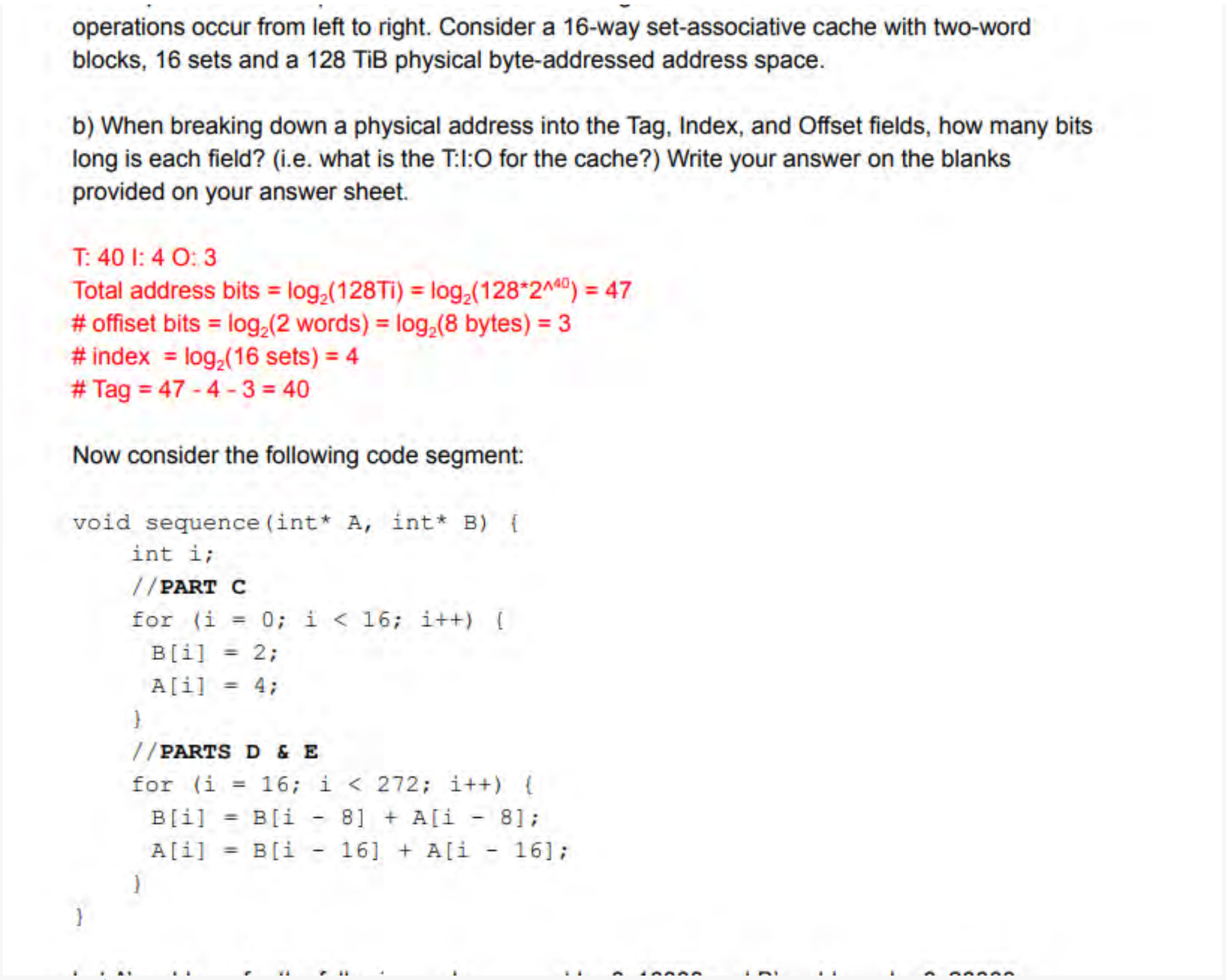
Anonymous Beaker 5 months ago

Fa 2017 Midterm question 5

I am confused why the miss pattern is not just MHHMHH for part D&E. the solution says HHMHHM then HHHHHH



For all portions of this question assume that an integer is one word in size and that ALL



operations occur from left to right. Consider a 16-way set-associative cache with two-word blocks, 16 sets and a 128 TiB physical byte-addressed address space.

b) When breaking down a physical address into the Tag, Index, and Offset fields, how many bits long is each field? (i.e. what is the T:I:O for the cache?) Write your answer on the blanks provided on your answer sheet.

T: 40 I: 4 O: 3
Total address bits = $\log_2$(128Ti) = $\log_2$(128*2^^{40}) = 47
# offiset bits = $\log_2$(2 words) = $\log_2$(8 bytes) = 3
# index = $\log_2$(16 sets) = 4
Tag = 47 - 4 - 3 = 40

Now consider the following code segment:

```
void sequence(int* A, int* B) {
    int i;
    //PART C
    for (i = 0; i < 16; i++) {
     B[i] = 2;
     A[i] = 4;
    }
    //PARTS D & E
    for (i = 16; i < 272; i++) {
     B[i] = B[i - 8] + A[i - 8];
     A[i] = B[i - 16] + A[i - 16];
    }
}
```

helpful! | 0

Peyrin Kao 5 months ago

Each cache block fits 2 words. The cache is 16-way set associative and there are two arrays that could put a block in a set, so we don't need to worry about conflict misses.

At the start of the second for loop, `A[0]-A[15]` and `B[0]-B[15]` are in the cache.

First iteration:
`B[i-8] = B[8]` hit
`A[i-8] = A[8]` hit
`B[i] = B[16]` miss
`B[i-16] = B[0]` hit
`A[i-16] = A[0]` hit
`A[i] = A[16]` miss

The two misses brought `A[16]-A[17]` and `B[16]-B[17]` into the cache.

Second iteration:
`B[i-8] = B[9]` hit
`A[i-8] = A[9]` hit
`B[i] = B[17]` hit
`B[i-16] = B[1]` hit
`A[i-16] = A[1]` hit

`A[i] = A[17]` hit

Everything hit, so nothing extra was brought into the cache.

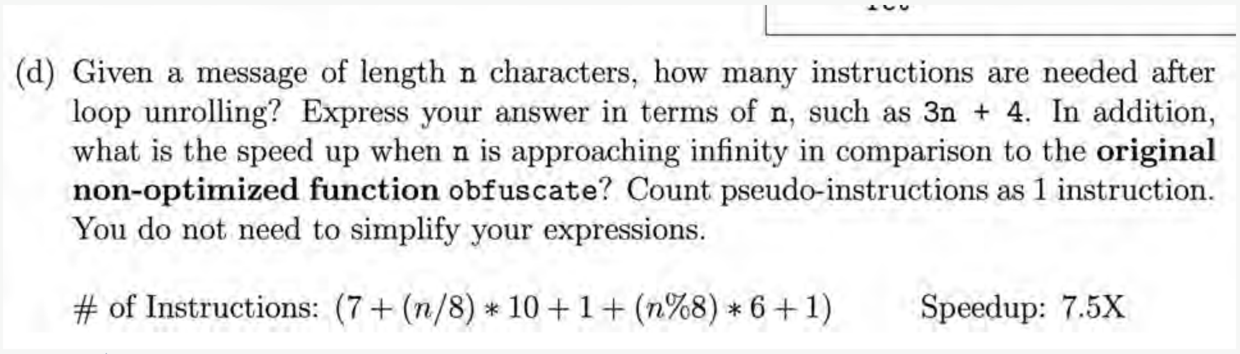
Third iteration:
`B[i-8] = B[10]` hit
`A[i-8] = A[10]` hit
`B[i] = B[18]` miss
`B[i-16] = B[2]` hit
`A[i-16] = A[2]` hit
`A[i] = A[18]` miss

From here you should be able to spot the pattern.
good comment | 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f29

Anonymous Mouse 5 months ago
fall 2018 P10
Can I have some explanation on how to approach this one?



(d) Given a message of length `n` characters, how many instructions are needed after loop unrolling? Express your answer in terms of `n`, such as `3n + 4`. In addition, what is the speed up when `n` is approaching infinity in comparison to the **original non-optimized function** `obfuscate`? Count pseudo-instructions as 1 instruction. You do not need to simplify your expressions.

# of Instructions: $(7 + (n/8) * 10 + 1 + (n\%8) * 6 + 1)$ Speedup: 7.5X

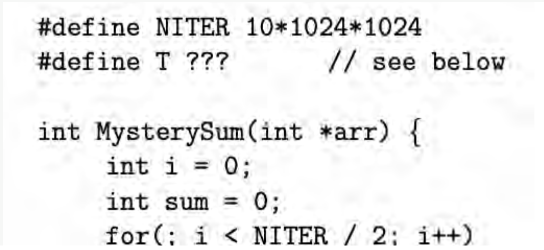
helpful! | 0

Peyrin Kao 5 months ago
@2125_f5
good comment | 0

Reply to this followup discussion

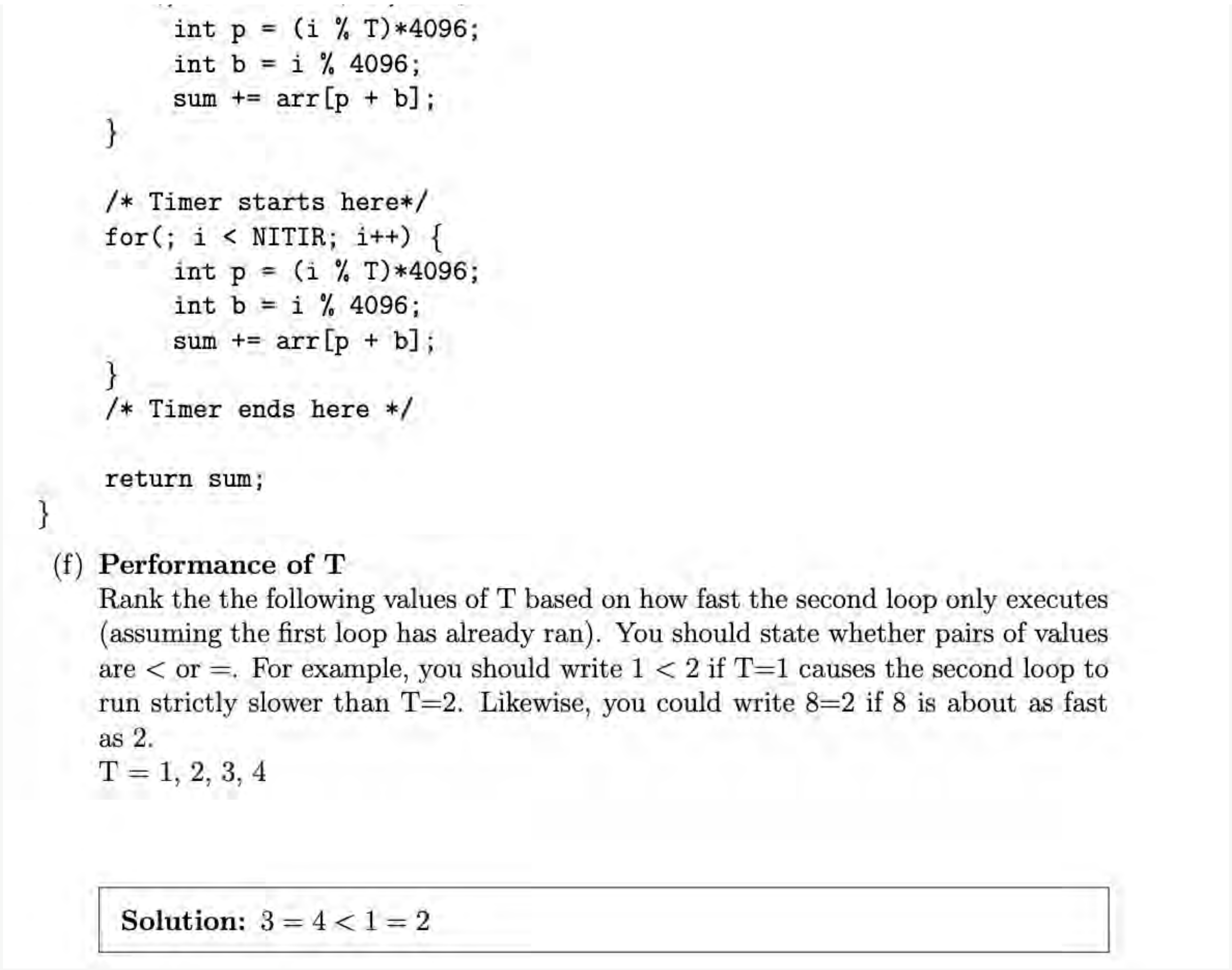
[✓ Resolved] Unresolved @2125_f30

Anonymous Mouse 5 months ago
sp 18 p12
i am so lost in this problem. can someone explain this and how t affects the performance? thank you!



```
#define NITER 10*1024*1024
#define T ???       // see below

int MysterySum(int *arr) {
     int i = 0;
     int sum = 0;
     for(; i < NITER / 2; i++)
```



```
        int p = (i % T)*4096;
        int b = i % 4096;
        sum += arr[p + b];
    }

    /* Timer starts here*/
    for(; i < NITIR; i++) {
        int p = (i % T)*4096;
        int b = i % 4096;
        sum += arr[p + b];
    }
    /* Timer ends here */

    return sum;
}
```

(f) **Performance of T**
Rank the the following values of T based on how fast the second loop only executes (assuming the first loop has already ran). You should state whether pairs of values are $<$ or $=$. For example, you should write $1 < 2$ if T=1 causes the second loop to run strictly slower than T=2. Likewise, you could write 8=2 if 8 is about as fast as 2.
T = 1, 2, 3, 4

Solution: $3 = 4 < 1 = 2$

helpful! | 0

Peyrin Kao 5 months ago
Answers from a past semester:

I believe what they're getting at here is...

Let's assume arr is a byte array (I don't think it's specified? I'll leave it as an exercise for the reader to see how things change if it's an int array.)

b is cycling over 4096 values. If arr is a byte array, then that means b just cycling through one page worth of bytes.

p changes depending on T. If T is 1, p is always 0. So then b is the only factor, and the memory access is all on a single page. This one page's virtual to physical mapping will get stored in the TLB. If T is 2, p toggles between 0 and 4096. So now we're talking about two pages. Since the TLB has two entries and is fully associative, this also fits in the TLB. So T=1 and T=2 take the same amount of time.

When T=3 or T=4, we now have to keep evicting TLB entries. If we assume LRU (which is specified), then when the third page is accessed, the first one's TLB entry is evicted. When we cycle back to the first one, we need space for it, so we evict the second one. When we move on to the second one, it's now gone. Etc. Even with T=3, we are always evicting the next one we need and never get to reuse one. This is basically the worst case already, so T=4 can't be any worse, so these two are equal. And since we keep needing to actually check the page table to refill the TLB entries, they're definitely worse than T=1 or T=2.

good comment | 0

Peyrin Kao 5 months ago

- Firstly, note how huge the cache are compared to how much memory we're accessing. From loop 1, all the contents of arr which we'll need are already in the cache, so no need to worry about main memory or disk. Everything is in the cache and the TLB.
- As you mentioned, 1024 consecutive ints are on each page.
- If T = 1, we're just incrementing the index of arr by 1 each iteration. A small number of times (every 1024 iterations), we'll be tasked with retrieving a new page and putting it in the TLB, but only approximately every 1024 iterations.
- If T = 2, the array access pattern looks something like this: 0, 4096, 1, 4097, 2, 4098, etc. Every single iteration, we're switching pages. However, the TLB contains 2 pages, so we're still good in this case. We still only have to fetch new pages from the page table every 1024 iterations.
- If T = 3, the access pattern becomes something like 0, 4096, 8192, 1, 4097, 8193, etc. **This is bad news.** When we try to access 8192, we kick 0 out of the TLB and retrieve the page table entry for 8192 from the cache (a more costly operation than just retrieving an entry from the TLB). Then, we go to access 1 but it's gone from the TLB; again we kick out the least recently used, the entry corresponding to 4096 in this case. We keep kicking out the entry we'll want on the next iteration!
- If T = 4, we run into essentially the same situation, in which we keep kicking the least recently used entry out of the TLB and then having to fetch it again from the cached page table. Just like T = 3, we end up retrieving a page table entry from the cache every single iteration.

good comment | 2

Anonymous Calc 5 months ago

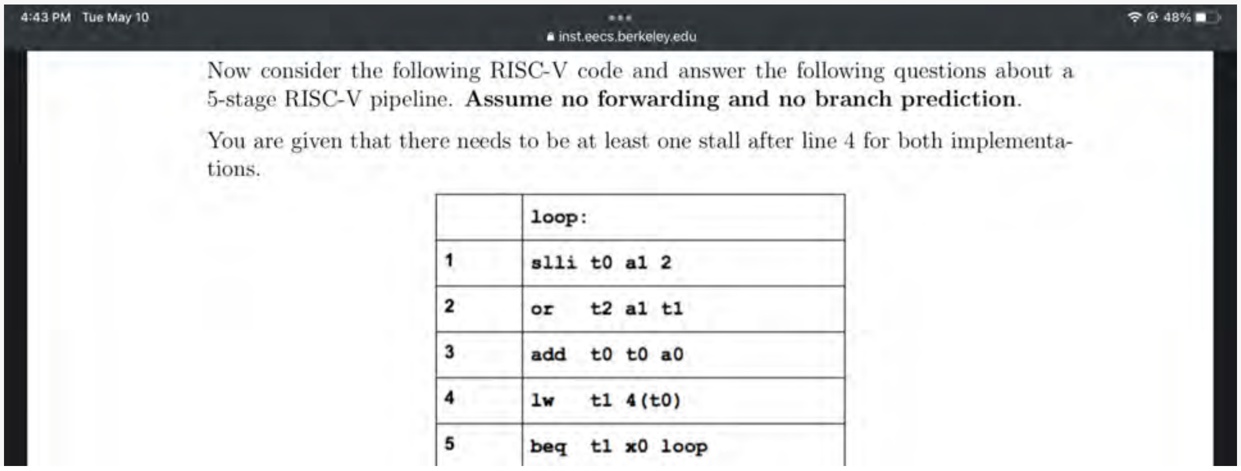
^ thank you for this extremely clear explanation

helpful! | 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f31

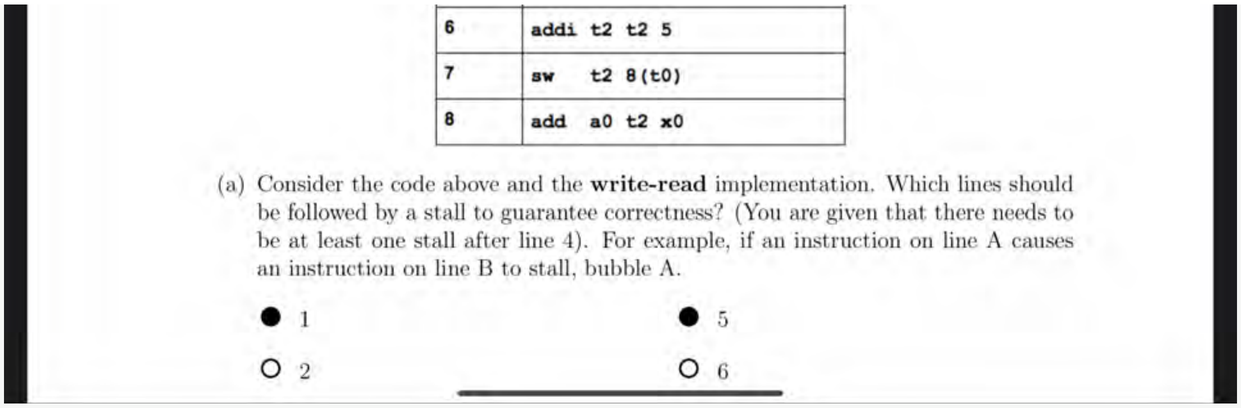
Anonymous Scale 5 months ago


4:43 PM Tue May 10 inst.eecs.berkeley.edu 48%

Now consider the following RISC-V code and answer the following questions about a 5-stage RISC-V pipeline. **Assume no forwarding and no branch prediction.**

You are given that there needs to be at least one stall after line 4 for both implementations.

	`loop:`
1	`slli t0 a1 2`
2	`or   t2 a1 t1`
3	`add  t0 t0 a0`
4	`lw   t1 4(t0)`
5	`beq  t1 x0 loop`



6	addi t2 t2 5
7	sw t2 8(t0)
8	add a0 t2 x0

(a) Consider the code above and the **write-read** implementation. Which lines should be followed by a stall to guarantee correctness? (You are given that there needs to be at least one stall after line 4). For example, if an instruction on line A causes an instruction on line B to stall, bubble A.

● 1 ● 5

○ 2 ○ 6

Why do we include 5? Is it because of control hazard

helpful! 1

Adelson Chua 5 months ago

Yes.

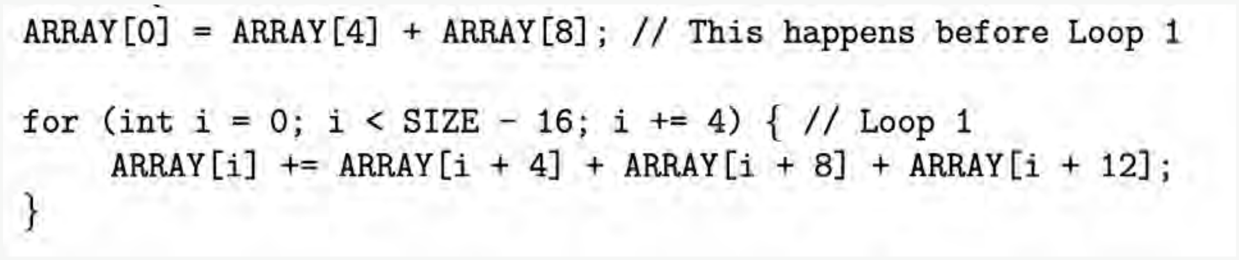
good comment 1

Reply to this followup discussion

Resolved Unresolved **@2125_f32**

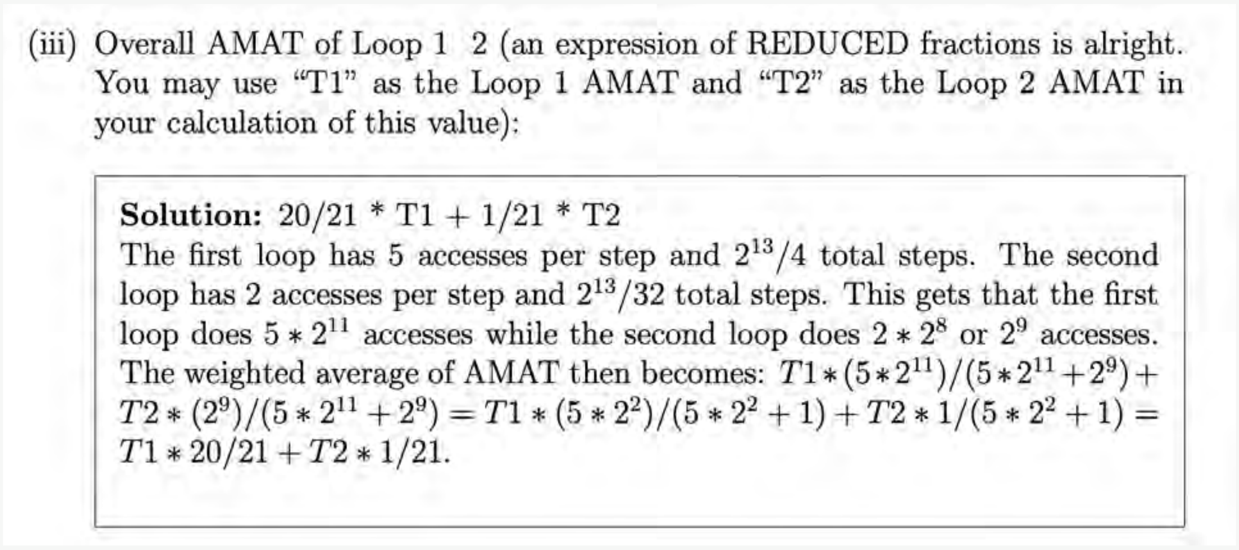
Anonymous Calc 5 months ago

SP18-Final-Q7.d.ii



```
ARRAY[0] = ARRAY[4] + ARRAY[8]; // This happens before Loop 1

for (int i = 0; i < SIZE - 16; i += 4) { // Loop 1
      ARRAY[i] += ARRAY[i + 4] + ARRAY[i + 8] + ARRAY[i + 12];
}
```



(iii) Overall AMAT of Loop 1 2 (an expression of REDUCED fractions is alright. You may use "T1" as the Loop 1 AMAT and "T2" as the Loop 2 AMAT in your calculation of this value):

Solution: 20/21 * T1 + 1/21 * T2
The first loop has 5 accesses per step and $2^{13}/4$ total steps. The second loop has 2 accesses per step and $2^{13}/32$ total steps. This gets that the first loop does $5 * 2^{11}$ accesses while the second loop does $2 * 2^8$ or 2^9 accesses. The weighted average of AMAT then becomes: $T1*(5*2^{11})/(5*2^{11}+2^9)+T2*(2^9)/(5*2^{11}+2^9) = T1*(5*2^2)/(5*2^2+1)+T2*1/(5*2^2+1) = T1*20/21+T2*1/21$.

Isn't the number of accesses for the first loop supposed to be 5 * (2^13 - 2^4) / 2^2? We don't run the loop all the way to the end of the array because we would have indexing errors. (also SIZE = 2^13)

helpful! | 0

Adelson Chua 5 months ago

Hmmm, oh yeah. That makes sense.

Yeah, I think you're right.

We'll note this to make sure we address this if we start to write/record some exam guides.

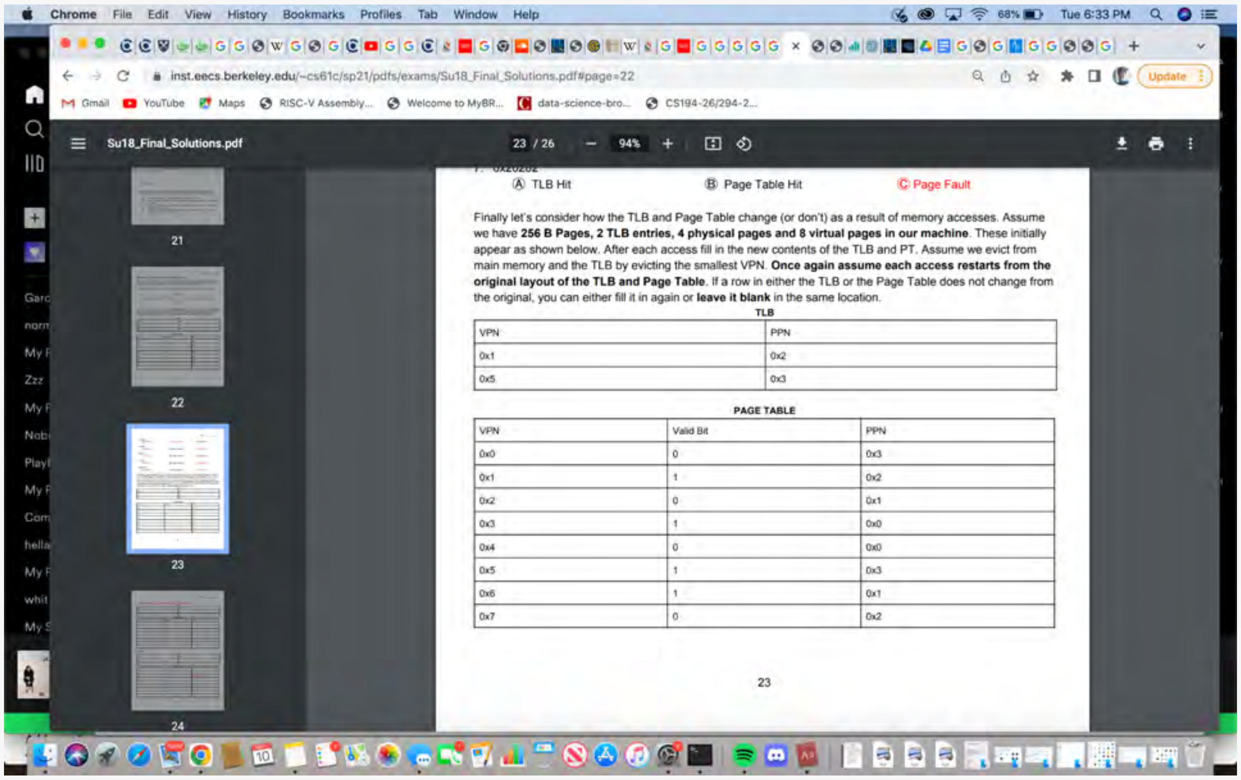
good comment | 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f33

Anonymous Beaker 5 months ago

Summer 2018 Final Question 11/12: Why is it that when we add 0x2F4 in the second entry we invalidate the 0x1 vpn entry?


inst.eecs.berkeley.edu/~cs61c/sp21/pdfs/exams/Su18_Final_Solutions.pdf#page=22

Su18_Final_Solutions.pdf 23 / 26 94%

Ⓐ TLB Hit Ⓑ Page Table Hit Ⓒ Page Fault

Finally let's consider how the TLB and Page Table change (or don't) as a result of memory accesses. Assume we have **256 B Pages, 2 TLB entries, 4 physical pages and 8 virtual pages in our machine**. These initially appear as shown below. After each access fill in the new contents of the TLB and PT. Assume we evict from main memory and the TLB by evicting the smallest VPN. **Once again assume each access restarts from the original layout of the TLB and Page Table**. If a row in either the TLB or the Page Table does not change from the original, you can either fill it in again or **leave it blank** in the same location.

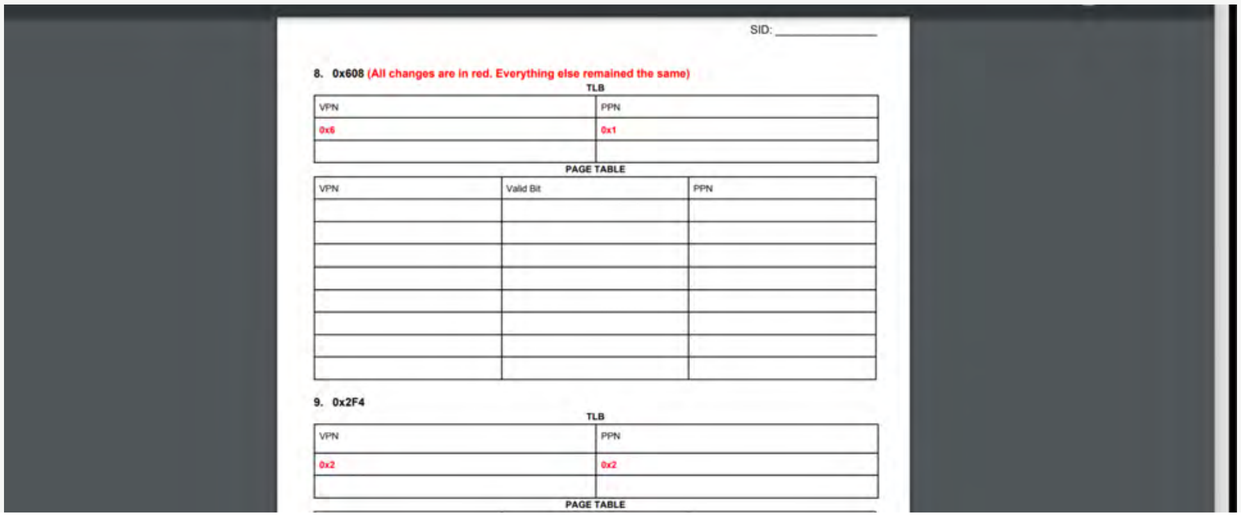
TLB

VPN	PPN
0x1	0x2
0x5	0x3

PAGE TABLE

VPN	Valid Bit	PPN
0x0	0	0x3
0x1	1	0x2
0x2	0	0x1
0x3	1	0x0
0x4	0	0x0
0x5	1	0x3
0x6	1	0x1
0x7	0	0x2

23


SID: ____________

8. 0x608 (All changes are in red. Everything else remained the same)

TLB

VPN	PPN
0x6	0x1

PAGE TABLE

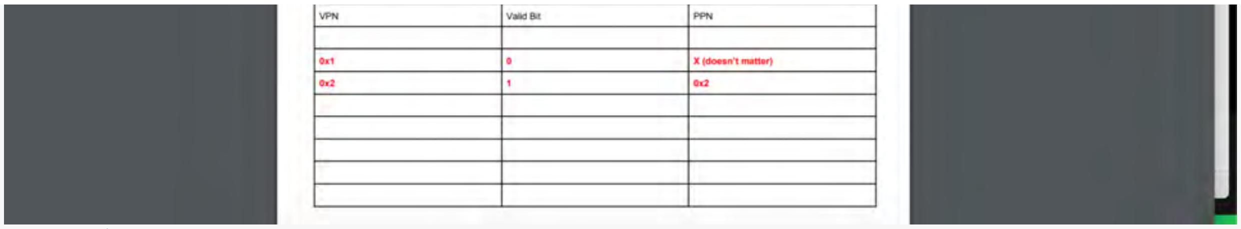
VPN	Valid Bit	PPN

9. 0x2F4

TLB

VPN	PPN
0x2	0x2

PAGE TABLE



VPN	Valid Bit	PPN
0x1	0	X (doesn't matter)
0x2	1	0x2

helpful! | 0

Adelson Chua 5 months ago

Problem states that there are 4 physical pages. The initial state already shows that there are already 4 pages being used (4 valid bits =1).

So when we access 0x2F4, we need a new page for that one.

The problem states that we evict the one with the smallest VPN, so we evict 0x1.

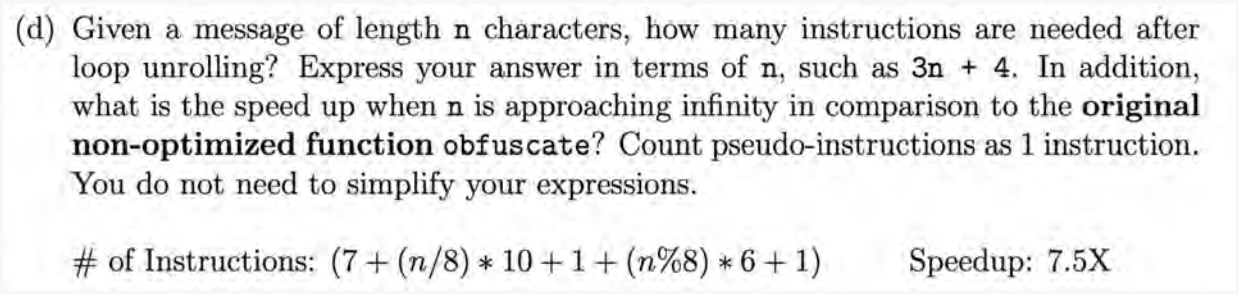
good comment | 0

Reply to this followup discussion

Resolved Unresolved @2125_f34

Anonymous Calc 5 months ago

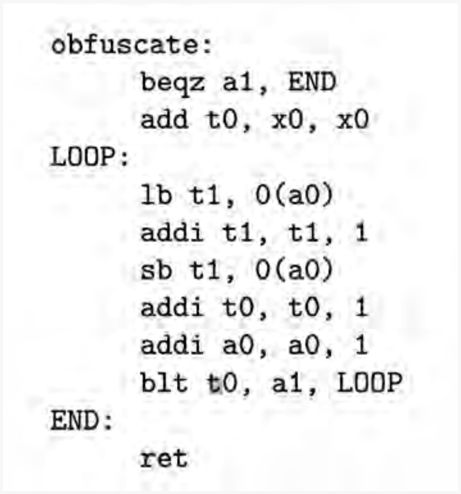
SP18-Final-Q10.d



(d) Given a message of length n characters, how many instructions are needed after loop unrolling? Express your answer in terms of n, such as 3n + 4. In addition, what is the speed up when n is approaching infinity in comparison to the **original non-optimized function** obfuscate? Count pseudo-instructions as 1 instruction. You do not need to simplify your expressions.

# of Instructions: $(7 + (n/8) * 10 + 1 + (n\%8) * 6 + 1)$ Speedup: 7.5X

I correctly for the number of instructions. As n --> oo, the (n/8)*10 part dominates. For the non-unrolled version this is code:



```
obfuscate:
      beqz a1, END
      add t0, x0, x0
LOOP:
      lb t1, 0(a0)
      addi t1, t1, 1
      sb t1, 0(a0)
      addi t0, t0, 1
      addi a0, a0, 1
      blt t0, a1, LOOP
END:
      ret
```

and since we are basically looping over all the bytes in char* d array, and there are 6 instructions per loop, we have 6n instructions. So then the speed up should be calculated as 6n / (10 * (n/8)) = 48/10 = 4.8?

helpful! | 0

Peyrin Kao 5 months ago

@2125_f5

good comment 1

Anonymous Calc 5 months ago

sorry! missed it.

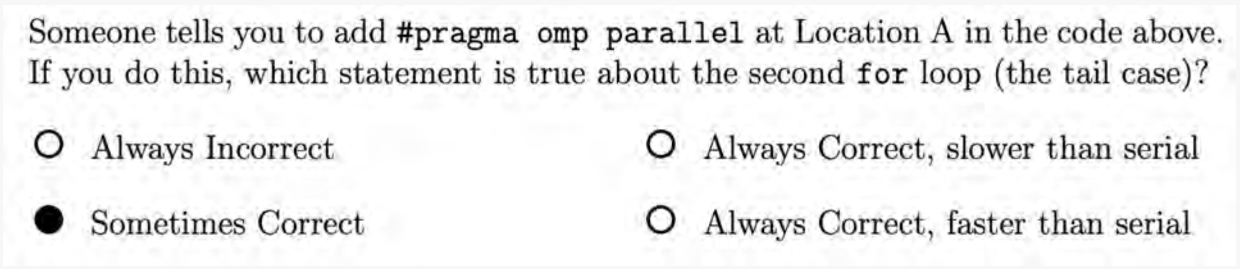
helpful! 0

Reply to this followup discussion

Resolved [✓] Unresolved @2125_f35

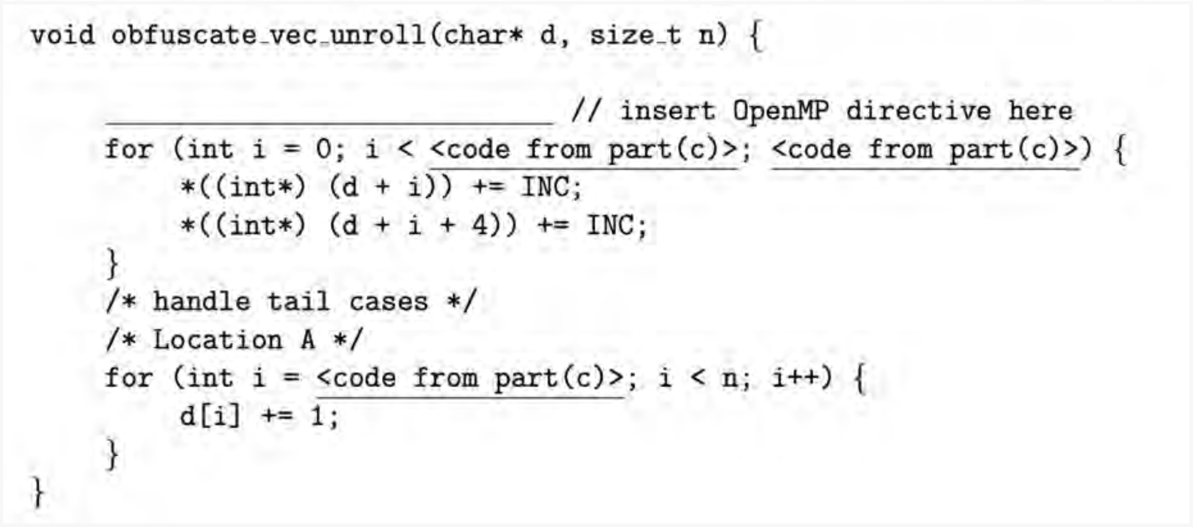
Anonymous Calc 5 months ago

SP18-Final-Q10.e



Someone tells you to add **#pragma omp parallel** at Location A in the code above. If you do this, which statement is true about the second **for** loop (the tail case)?

○ Always Incorrect ○ Always Correct, slower than serial [✓ Sometimes Correct] ○ Always Correct, faster than serial



```
void obfuscate_vec_unroll(char* d, size_t n) {

    ___________________________ // insert OpenMP directive here
    for (int i = 0; i < <code from part(c)>; <code from part(c)>) {
        *((int*) (d + i)) += INC;
        *((int*) (d + i + 4)) += INC;
    }
    /* handle tail cases */
    /* Location A */
    for (int i = <code from part(c)>; i < n; i++) {
        d[i] += 1;
    }
}
```

I'm confused why it isn't always correct. In the tail case for loop, we never have data racing issues as each iteration we are accessing a different location in memory. The only thing I was unsure about is whether or not it was faster or slower that serial (I thought if the tail case had a small enough number of iterations, then multi-threading would actually be slower). But I didn't know that the answer would be "sometimes correct". Could someone explain this?

helpful! 0

Adelson Chua 5 months ago

Location A is above the tail case loop.

The directive given by the problem is **#pragma omp parallel** (note, it's not for), so each thread that was spawned will do the same loop at the same time. That's what causes the problem.

good comment 0

Anonymous Calc 5 months ago

I see, so then it is "sometimes correct" because it is correct when that directive only spawns a single thread?

helpful! | 0

Adelson Chua 5 months ago

Basically yes.

There's also a case where all threads run in a perfectly interleaved fashion, basically replicating what a single thread can do.

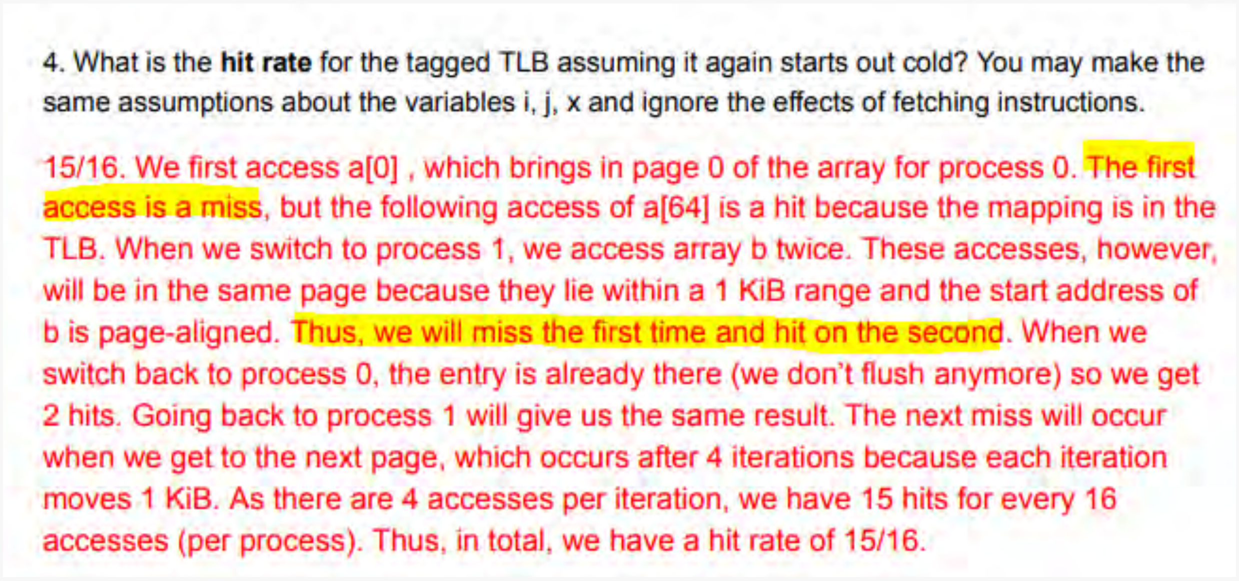
good comment | 0

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f36

Anonymous Poet 5 months ago

Fa17-Final-Q12.4



4. What is the **hit rate** for the tagged TLB assuming it again starts out cold? You may make the same assumptions about the variables i, j, x and ignore the effects of fetching instructions.

15/16. We first access a[0] , which brings in page 0 of the array for process 0. The first access is a miss, but the following access of a[64] is a hit because the mapping is in the TLB. When we switch to process 1, we access array b twice. These accesses, however, will be in the same page because they lie within a 1 KiB range and the start address of b is page-aligned. Thus, we will miss the first time and hit on the second. When we switch back to process 0, the entry is already there (we don't flush anymore) so we get 2 hits. Going back to process 1 will give us the same result. The next miss will occur when we get to the next page, which occurs after 4 iterations because each iteration moves 1 KiB. As there are 4 accesses per iteration, we have 15 hits for every 16 accesses (per process). Thus, in total, we have a hit rate of 15/16.

I am kind of confused how they got 15/16 based on this explanation. They said we miss twice (one for each process) on the first iteration. Wouldn't we continue to miss once for each process every four iterations, so two misses for every sixteen accesses? I don't understand why the hit rate isn't 14/16.

helpful! | 0

Adelson Chua 5 months ago

The loop accessing the array a[] has 4 memory accesses, same as b[]. Solution says there are 4 iterations in total before we miss again.

For array a, there are 4 * 4 accesses = 16, 1 of which is a miss. Same for array b.

So it is 15/16 for both of them.

good comment | 1

Anonymous Poet 5 months ago

ohh i think i see it, so the overall hit rate is the average of the hit rates of both threads, right? i think that's where i was confused.

helpful! | 0

Adelson Chua 5 months ago

It's just written in simplest terms, don't take the average.

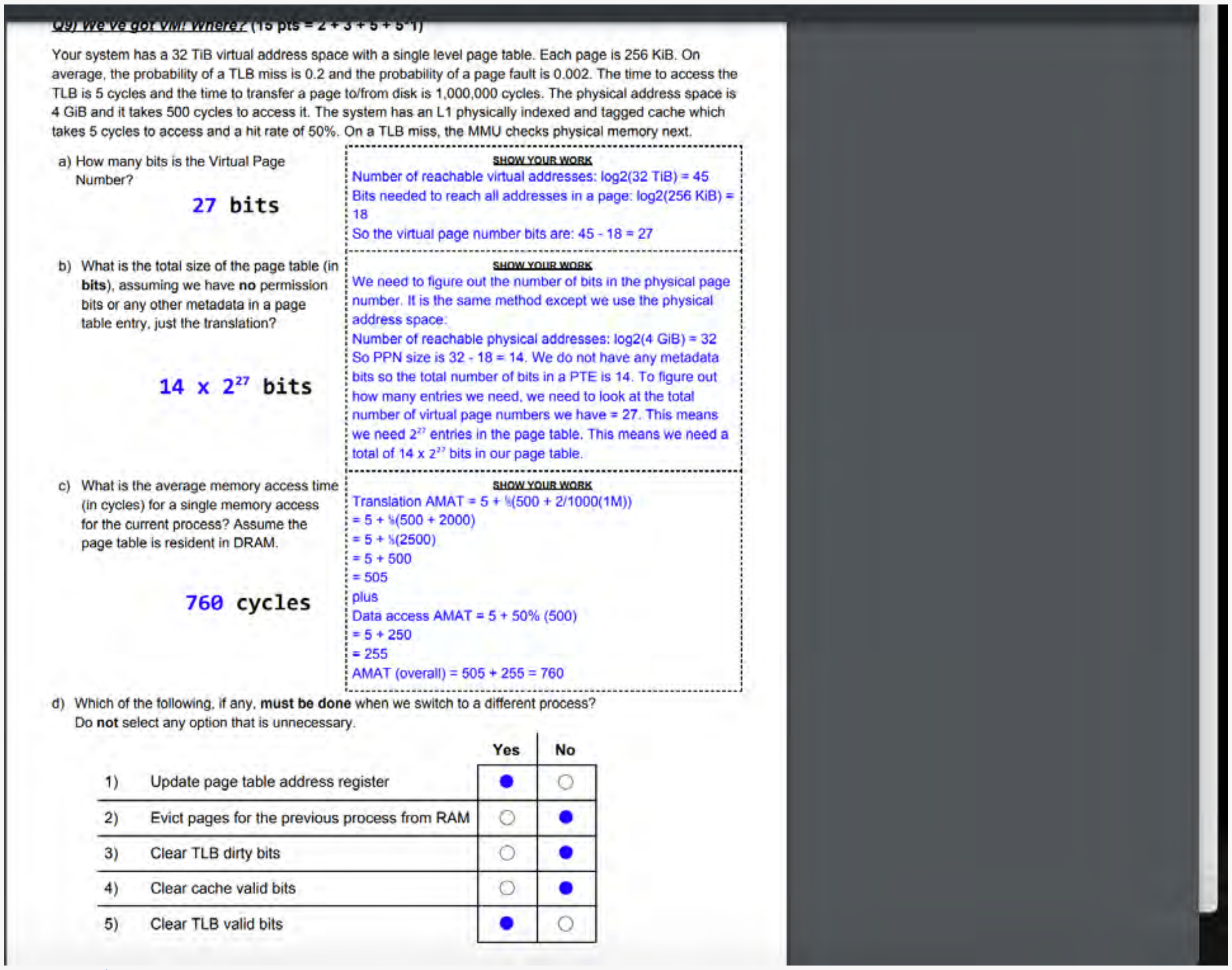
good comment | 0

Reply to this followup discussion

Resolved Unresolved @2125_f37

Anonymous Beaker 5 months ago

Fall 2019 Final Q9: Why do we not flush the dirty bits of tlb for part d? Also for part b, why isn't the page table size just virtual address space / page size? ie 2^45/2^18 = 2^27?



Q9) We've got VM! Where? (15 pts = 2 + 3 + 5 + 5*1)

Your system has a 32 TiB virtual address space with a single level page table. Each page is 256 KiB. On average, the probability of a TLB miss is 0.2 and the probability of a page fault is 0.002. The time to access the TLB is 5 cycles and the time to transfer a page to/from disk is 1,000,000 cycles. The physical address space is 4 GiB and it takes 500 cycles to access it. The system has an L1 physically indexed and tagged cache which takes 5 cycles to access and a hit rate of 50%. On a TLB miss, the MMU checks physical memory next.

a) How many bits is the Virtual Page Number?

27 bits

SHOW YOUR WORK

Number of reachable virtual addresses: log2(32 TiB) = 45
Bits needed to reach all addresses in a page: log2(256 KiB) = 18
So the virtual page number bits are: 45 - 18 = 27

b) What is the total size of the page table (in **bits**), assuming we have **no** permission bits or any other metadata in a page table entry, just the translation?

14×2^{27} bits

SHOW YOUR WORK

We need to figure out the number of bits in the physical page number. It is the same method except we use the physical address space:
Number of reachable physical addresses: log2(4 GiB) = 32
So PPN size is 32 - 18 = 14. We do not have any metadata bits so the total number of bits in a PTE is 14. To figure out how many entries we need, we need to look at the total number of virtual page numbers we have = 27. This means we need 2^{27} entries in the page table. This means we need a total of 14×2^{27} 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

d) Which of the following, if any, **must be done** when we switch to a different process? Do **not** select any option that is unnecessary.

		Yes	No
1)	Update page table address register	●	○
2)	Evict pages for the previous process from RAM	○	●
3)	Clear TLB dirty bits	○	●
4)	Clear cache valid bits	○	●
5)	Clear TLB valid bits	●	○

helpful! | 0

Adelson Chua 5 months ago

@2126_f6

Also for part b, why isn't the page table size just virtual address space / page size?

Why are you trying to compute for the number of pages? The question is asking for the total page table size which is 2^VPN * PTE

good comment | 1

Reply to this followup discussion

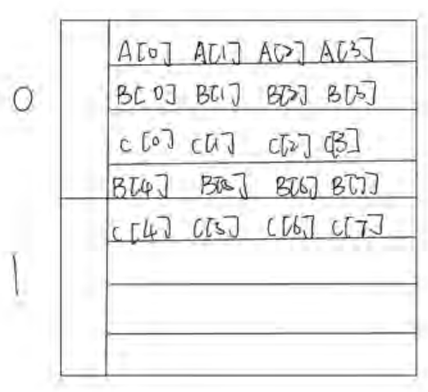
[✓ Resolved] Unresolved @2125_f38

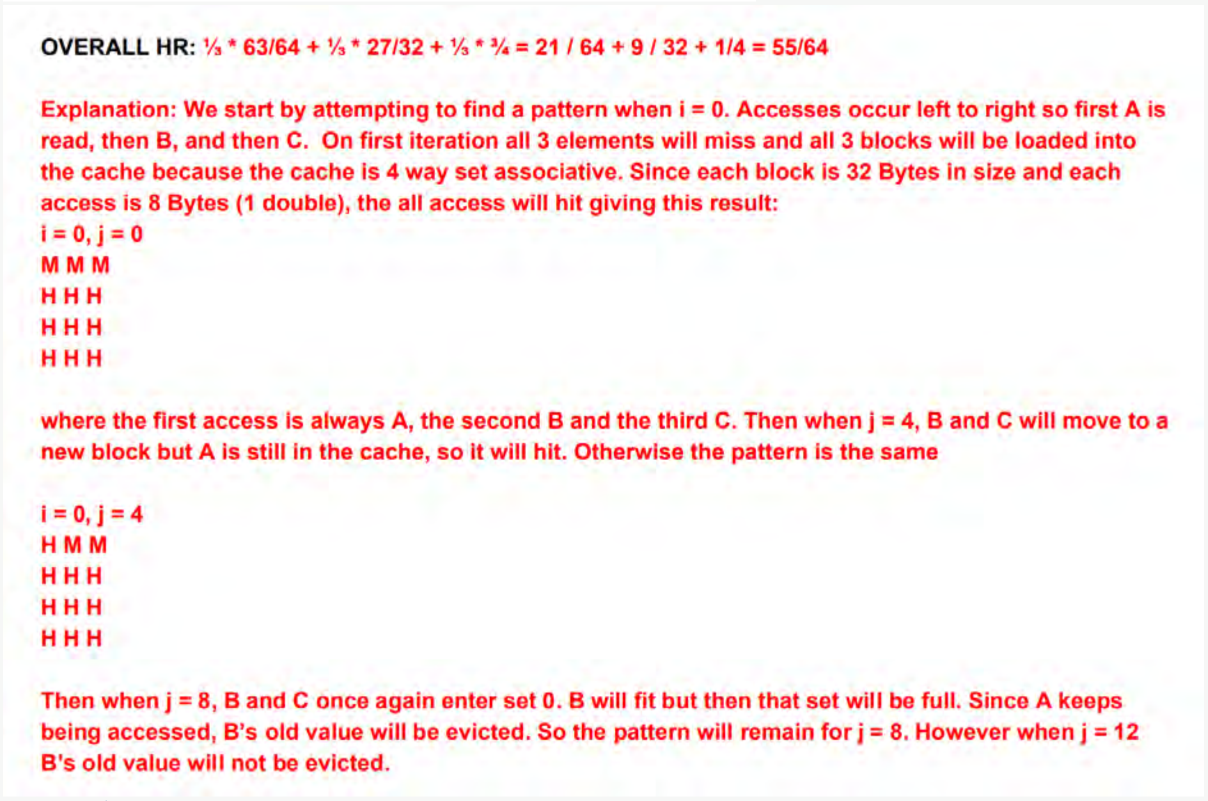
Anonymous Mouse 5 months ago

summer 2018 q6

I am actually not sure how the block is brought into the cache. Don't get the part why j=8, B, C back to set0.

That's my drawing:





OVERALL HR: ⅓ * 63/64 + ⅓ * 27/32 + ⅓ * ¾ = 21 / 64 + 9 / 32 + 1/4 = 55/64

Explanation: We start by attempting to find a pattern when i = 0. Accesses occur left to right so first A is read, then B, and then C. On first iteration all 3 elements will miss and all 3 blocks will be loaded into the cache because the cache is 4 way set associative. Since each block is 32 Bytes in size and each access is 8 Bytes (1 double), the all access will hit giving this result:

i = 0, j = 0
M M M
H H H
H H H
H H H

where the first access is always A, the second B and the third C. Then when j = 4, B and C will move to a new block but A is still in the cache, so it will hit. Otherwise the pattern is the same

i = 0, j = 4
H M M
H H H
H H H
H H H

Then when j = 8, B and C once again enter set 0. B will fit but then that set will be full. Since A keeps being accessed, B's old value will be evicted. So the pattern will remain for j = 8. However when j = 12 B's old value will not be evicted.

helpful! | 0

Anonymous Mouse 5 months ago

Block size is 32 bit, which means each time we can bring in 4 doubles. And it's 4 set associativity.

helpful! | 0

Adelson Chua 5 months ago

@2125_f12

good comment | 0

Anonymous Mouse 5 months ago

so the drawing represents what is the cache now? And 4 entris per line due to 4 set associativity?

helpful! | 0

Anonymous Mouse 5 months ago

I think I may be misunderstood the block size. Is that for size of one cache line, or the whole block size for a set, like index 0..

helpful! | 0

Adelson Chua 5 months ago

Block size just refers to a single cache line (1 way of the set). It doesn't include the whole set.

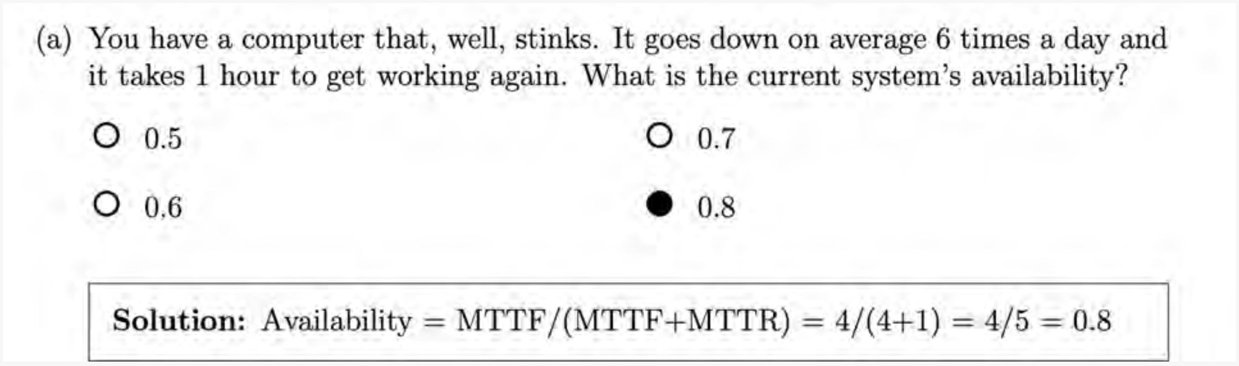
good comment | 0

Reply to this followup discussion

[✓ Resolved] Unresolved **@2125_f39**

Anonymous Calc 5 months ago

SP18-Final-Q13.a



(a) You have a computer that, well, stinks. It goes down on average 6 times a day and it takes 1 hour to get working again. What is the current system's availability?

○ 0.5 ○ 0.7

○ 0.6 [✓ 0.8]

Solution: Availability = MTTF/(MTTF+MTTR) = 4/(4+1) = 4/5 = 0.8

if it fails 6 times a day, then the mean time between failures should be 4 hours. Which means that the MTTF = 3 because MTTR is 1. 3/4 is 0.75. Am I missing something?

helpful! | 0

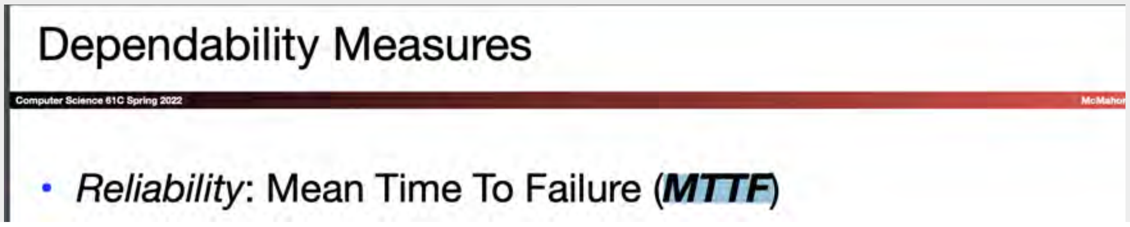
Peyrin Kao 5 months ago

~~MTTF includes repair time, so it's 4 hours, not 3. The first statement you have is correct: "if it fails 6 times a day, then the mean time between failures should be 4 hours."~~

See Adel's answer, maybe

good comment | 0

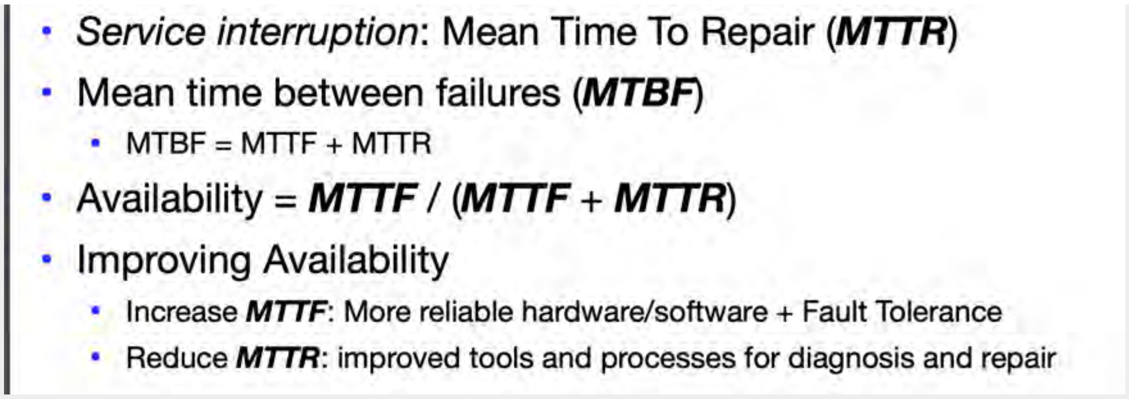
Anonymous Calc 5 months ago



Dependability Measures

Computer Science 61C Spring 2022 McMahon

- *Reliability*: Mean Time To Failure (***MTTF***)



- *Service interruption*: Mean Time To Repair (***MTTR***)
- Mean time between failures (***MTBF***)
  - MTBF = MTTF + MTTR
- Availability = ***MTTF*** / (***MTTF*** + ***MTTR***)
- Improving Availability
  - Increase ***MTTF***: More reliable hardware/software + Fault Tolerance
  - Reduce ***MTTR***: improved tools and processes for diagnosis and repair

It says here that MTBF is MTTF and MTTR, which means MTBF includes the mean time to repair. It doesn't make sense for MTTF to also include the MTTR.

helpful! | 0

Anonymous Calc 5 months ago

I have MTBF = 4, MTTR = 1, MTTF = 3. I'm still confused, sorry.

helpful! | 0

Adelson Chua 5 months ago

Hmm, your reasoning sounds correct.

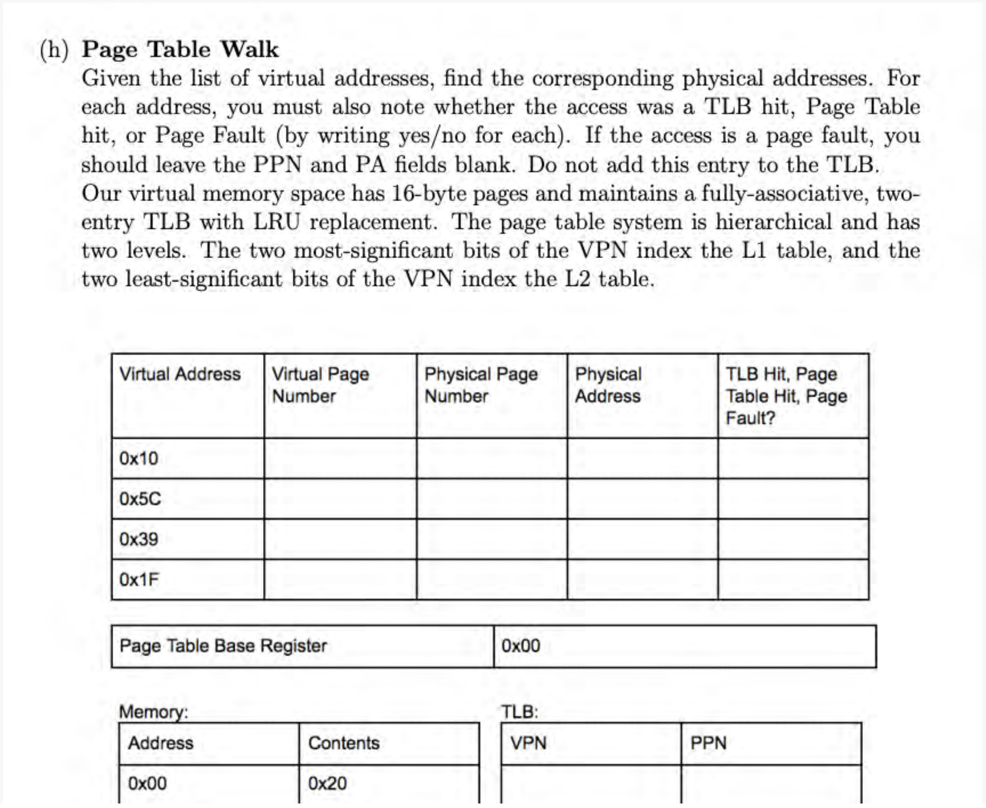
We'll check this. But, looks like you are correct.

good comment | 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f40

Anonymous Calc 5 months ago



(h) **Page Table Walk**
Given the list of virtual addresses, find the corresponding physical addresses. For each address, you must also note whether the access was a TLB hit, Page Table hit, or Page Fault (by writing yes/no for each). If the access is a page fault, you should leave the PPN and PA fields blank. Do not add this entry to the TLB. Our virtual memory space has 16-byte pages and maintains a fully-associative, two-entry TLB with LRU replacement. The page table system is hierarchical and has two levels. The two most-significant bits of the VPN index the L1 table, and the two least-significant bits of the VPN index the L2 table.

Virtual Address	Virtual Page Number	Physical Page Number	Physical Address	TLB Hit, Page Table Hit, Page Fault?
0x10				
0x5C				
0x39				
0x1F				

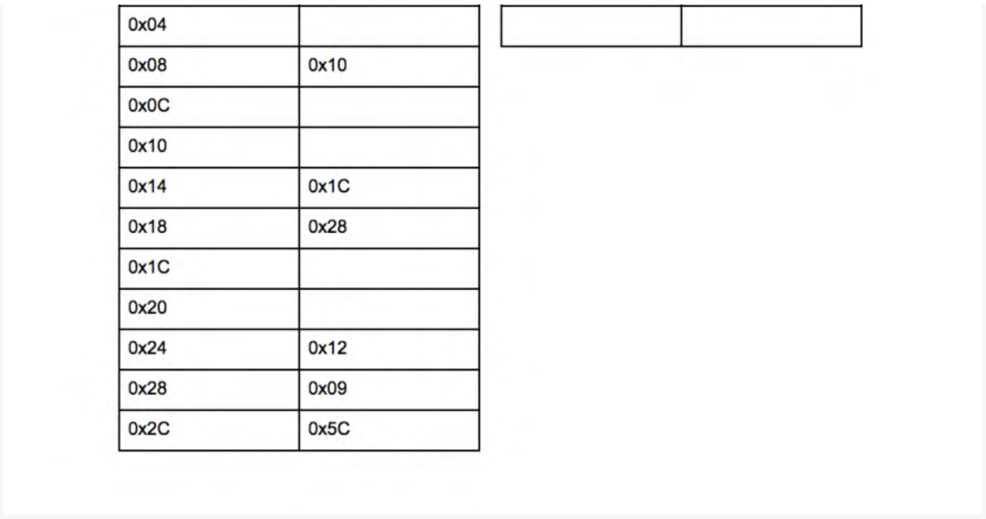
Page Table Base Register	0x00

Memory:

Address	Contents
0x00	0x20

TLB:

VPN	PPN



0x04	
0x08	0x10
0x0C	
0x10	
0x14	0x1C
0x18	0x28
0x1C	
0x20	
0x24	0x12
0x28	0x09
0x2C	0x5C

kinda confused about the memory table. Does every memory address the address of 2 bits? because the jumps go by 4 but the contents are only 8 bits.

helpful! 0

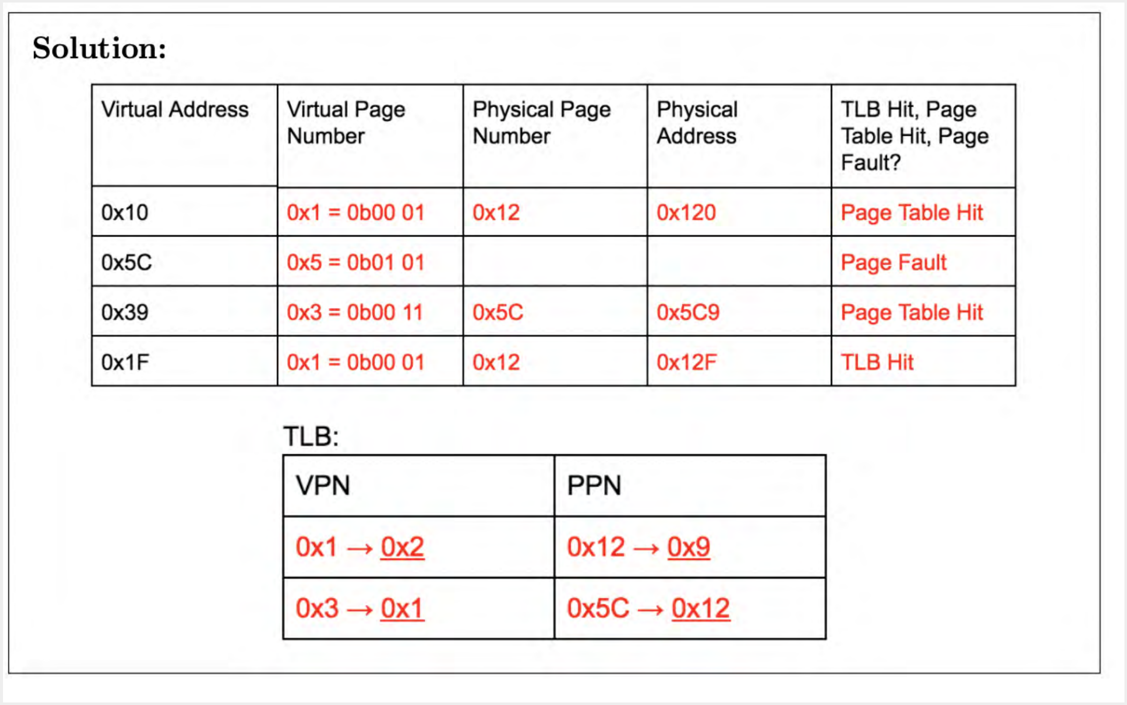
Anonymous Calc 5 months ago

SP18-Final-Q12.h sorry!

helpful! 0

Anonymous Calc 5 months ago

Okay so after thinking about it, I think its just the standard 32-bit system, just we assume that 0's are to the left of the MSB. So disregard my previous question.



Solution:

Virtual Address	Virtual Page Number	Physical Page Number	Physical Address	TLB Hit, Page Table Hit, Page Fault?
0x10	0x1 = 0b00 01	0x12	0x120	Page Table Hit
0x5C	0x5 = 0b01 01			Page Fault
0x39	0x3 = 0b00 11	0x5C	0x5C9	Page Table Hit
0x1F	0x1 = 0b00 01	0x12	0x12F	TLB Hit

TLB:

VPN	PPN
0x1 → <u>0x2</u>	0x12 → <u>0x9</u>
0x3 → <u>0x1</u>	0x5C → <u>0x12</u>

I don't understand this TLB though. Why is there a mapping on both the VPN and PPN side?

helpful! 0

Adelson Chua 5 months ago

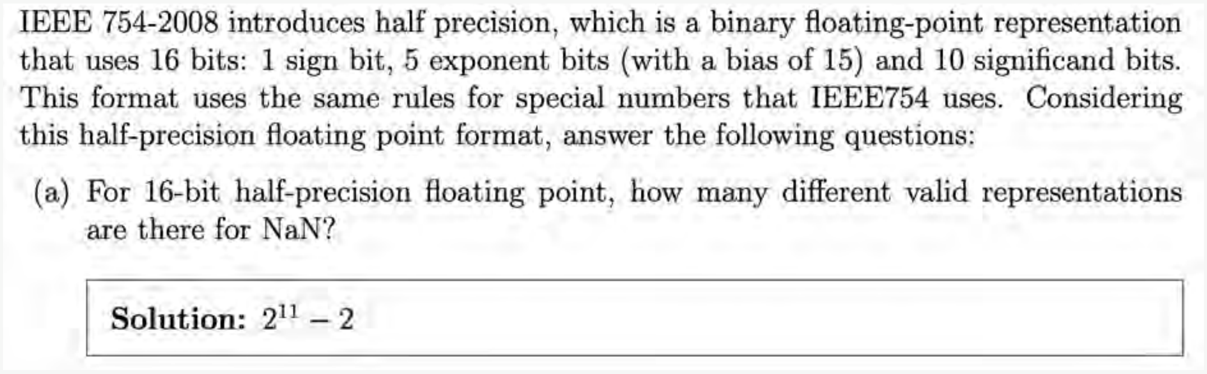
It's not a mapping, it's showing a transition of the TLB state as you do the memory accesses. The final state of the TLB is the underlined one.

good comment 1

Reply to this followup discussion

[✓ Resolved] Unresolved @2125_f41

Anonymous Gear 2 5 months ago


IEEE 754-2008 introduces half precision, which is a binary floating-point representation that uses 16 bits: 1 sign bit, 5 exponent bits (with a bias of 15) and 10 significand bits. This format uses the same rules for special numbers that IEEE754 uses. Considering this half-precision floating point format, answer the following questions:

(a) For 16-bit half-precision floating point, how many different valid representations are there for NaN?

Solution: $2^{11} - 2$

For Spring 2018 Final Q. 9a, where did the -2 come from in this answer?

helpful! 0

Adelson Chua 5 months ago

To not include the all 0 combination which corresponds to infinity.

good comment 0

Reply to this followup discussion

Start a new followup discussion

Compose a new followup discussion