Manit Gandhi 6280004

System Skill Final Quiz

Date: Tuesday, December 7th, 2021
Due: Thursday, December 9th, 2021 at 11.59PM
Instructor: Rachata Ausavarungnirun

Problem 1 (15 Points):	
Problem 2 (20 Points):	
Problem 3 (15 Points):	
Problem 4 (20 Points):	
Problem 5 (30+5 Points):	
Extra Credit (20 Points):	
Total (100+25 Points):	

Instructions:

1. This is a 72-hour exam. **If you get 100, you get a full score. Any points above 100 goes to your extra credit at the conversion rate of 50% per point.**
2. The total points is far greater than 100. This is intended so that 1) you can pick the questions you are more comfortable and 2) allows rooms for extra credit if you know all the class material. Please read all the questions first.
3. Submit your work as a pdf file on Canvas.
4. Clearly indicate your final answer for each conceptual problem.
5. **DO NOT CHEAT.** If we catch you cheating in any shape or form, you will be penalized based on **my plagiarism policy** ($N*$ 10% of your total grade, where N is the number of times you plagiarized previously).

Tips:

- **Read everything.** Read all the questions on all pages first and formulate a plan.
- **Be cognizant of time.** It is a sad day if you click submit when the submission site close.
- **Canvas allows resubmission.** I will take a look at the last version you submit.
- **Show work when needed.** You will receive partial credit at the instructors' discretion.

Initials: Manit Gandhi 6280004

1. Datalab ... Again! (15 points)

We are going to ask you to implement more bit arithmetic manipulation in a similar fashion as assignment 3.

Unlike assignment 3, you are to write the answer on this exam.

Similar to Assignment 3, you are *only* allowed to use the following eight operators:

```
! ~ & ^ | + << >>
```

Also, you are not allowed to use any constants longer than 8 bits.

Table 1 describes a set of functions that manipulate and test sets of bits. The "Rating" field gives the difficulty rating (the number of points) for the puzzle, and the "Max ops" field gives the maximum number of operators you are allowed to use to implement each function.

Name	Description	Rating	Max Ops
`bitXor(x,y)`	return x^y without using only ~ and &	1	14
`isEqual(x,y)`	return 1 of x == y, 0 otherwise	2	5

Table 1: Bit-Level Manipulation Functions.

For every question, **Please explain what you are trying to do**.

Write down the function body for `bitXor(x,y)` below.

```
int bitXor(int x, int y){
    int p = x&y;
    int q = ~x&~y;
    int r = ~p&~q;
    return r;
}
```

XOR is basically ~(X AND Y) AND ~(X NOR Y).
Variable p stores X AND Y.
Variable q stores X NOR Y.
Variable r stores p NOR q which is x XOR y.
Returning r as the result.

Initials: Manit Gandhi 6280004

Write down the function body for `isEqual(x,y)` below.

```
int isEqual(int x, int y){
    return !(x^y);
}
```

After performing XOR on 2 variables that are equal to each other, the result is 0.
We then apply NOT on that result so the result is 1.

For this part, you can only use logical operation and loops. Assume you have a list of pairs in an array `arr[]` of size `n`. However, there is one sad sad sad number in there that does not have it's pair (for example, your array can be `[]1, 2, 8 ,1, 8]` and the sad number is `2`). Write a code to find this sad number. There is no limit on how many operations you want to use but **you can only loop through the entire array once!**. You get zero credit if you loop through the array more than once.

```
int findSad(int arr[], int n) {
    int result = arr[0];
    for (int i = 1; i<n; i++){
        result = result^arr[i];
    }
    return result;
}
```

Initials: Manit Gandhi 6280004

2. Jump Table [20 points]

In this question, consider the following assembly codes below. We now assume the 32-bit x86 ISA, which upon a function call, the caller will place the first input at `%ebp+8` and the second input at `%ebp+12`. Fill in the rest of the C code for each of the switch cases. Write "NOTHING HERE" if the space should be left blank or if that line of code should not exist (i.e., the program does not suppose to modify `result` at that line).

```
blah:
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %edx
movl 12(%ebp), %eax
cmpl $7, %edx
ja .L8
jmp *.L9(,%edx,4)
.section .rodata
.align 4
.align 4
.L9:
.long .L8
.long .L4
.long .L5
.long .L5
.long .L8
.long .L7
.long .L6
.long .L4
.text
.L4:
mov (%edx), %eax
jmp .L2
.L5:
mov (%eax), %eax
jmp .L2
.L6:
mov (%edx), %ecx
add %ecx, %eax
jmp .L2
.L7:
addq %eax, %eax
.L8:
incl %eax
.L2:
popl %ebp
ret
```

Initials: Manit Gandhi 6280004

In the space below, fill in the blank to reflect the assembly code above.

```
int blah(int a, int b)
{
int result;

switch(  a  )
{

    case  1  :

    case  7  :

        result =  a  ;
        break;

    case  6  :

        result =  a+b  ;
        break;

    case  2  :

    case  3  :

        result =  a+b  ;
        break;
    case  5  :

        result =  a+b  ;

    default:
        result =  ++b  ;
}

return result;
}
```

Initials: Manit Gandhi 6280004

3. Code Size [15 points]

For the following code, please fill in the number (in **hexadecimal base**) for the address of each instruction.

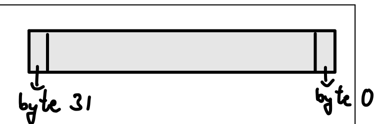
Address	Instruction (in binary)	Instruction (in Assembly)
5fa:	55	push %rbp
5fb:	48 89 e5	mov %rsp,%rbp
5fe:	89 7d fc	mov %edi,-0x4(%rbp)
601:	89 75 f8	mov %esi,-0x8(%rbp)
604:	8b 55 fc	mov -0x4(%rbp),%edx
607:	8b 45 f8	mov -0x8(%rbp),%eax
60a:	01 d0	add %edx,%eax
60c:	5d	pop %rbp
60d:	c3	retq
60e:	55	push %rbp
60f:	48 89 e5	mov %rsp,%rbp
612:	48 83 ec 08	sub $0x8,%rsp
616:	89 7d fc	mov %edi,-0x4(%rbp)
619:	89 75 f8	mov %esi,-0x8(%rbp)
61C:	8b 55 f8	mov -0x8(%rbp),%edx
61F:	8b 45 fc	mov -0x4(%rbp),%eax
622:	89 d6	mov %edx,%esi
624:	89 c7	mov %eax,%edi
626:	e8 cf ff ff ff	callq 5fa

4. Caching [20 points]

In this question, let's assume that we have a 16-bit system with a single level 2-way set associative cache with 4 sets, and a cache block size of 32 bytes.

How many bits are needed for the setID and the tags? Draw the breakdown of the tag/index/byte-in-block bits.

BIB → $\log_2$(CB size) = $\log_2(32)$ = 5 bits
index → $\log_2$(No. of sets) = $\log_2(4)$ = 2 bits (set id)
tag bits → remaining = 16 − (5+2) = 16 − 7 = 9 bits



For the following program, assume that an integer is 4 bytes.

```
int i; // Assume these variables are stored in the registers.
int a[2048]; // Assume that a = 0x1000
int b[2048]; // Assume that b = 0x8000

for(i=0;i<2048;i++)
    a[i] = i;

for(i=0;i<2048;i++)
    b[i] = a[i]++;
```

What is the total number of cache accesses? What is the number of cache hits and what is the number of cache misses? **Show your work.**

Total no. of access
= 2048 (a) + 2048(b) + 2048(c)
= 6144
Here, c is a[i]++

Cache miss = $\frac{6144}{32/4}$ = 768

Cache hit = 6144 − 768
= 5376

Initials: Manit Gandhi 6280004

Can I modify this program in order to make sure the program behaves the same but have higher cache hit rate? If so, how? If not, explain why?

Yes. By changing the datype from int to short, we will be using 2 bytes instead of 4.

$$\text{Cache miss} = \frac{6144}{32/2} = 384$$

$$\text{Cache hit} = 6144 - 384 = 5760$$

$$\text{Previous cache hit} = 5376$$

$$\text{Increase} = 5760 - 5376 = 384$$

Initials: Manit Gandhi 6280004

5. Virtual Memory [30+5 points]

Let's create a simple **BIG endian** machine that still utilize 2MB page size, and 32-bit address. Assuming the following data in the memory and the page table root is at `0x10`, and the page table entries is 32-bit long, where the n most significant bits after the page offset are used for the physical page number.

Address	Values (in hexadecimal) [Lowest bit – Highest bit]															
0x00	00	10	20	30	40	50	60	70	80	90	a0	b0	c0	d0	e0	f0
0x10	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
0x20	00	10	20	30	40	50	60	70	80	90	a0	b0	c0	d0	e0	f0
0x30	30	31	32	33	34	35	36	37	38	39	3a	3b	3c	3d	3e	3f
0x40	00	10	20	30	40	50	60	70	80	90	a0	b0	c0	d0	e0	f0
0x50	19	15	12	0a	6b	3a	4b	12	91	ac	ff	fe	3c	3d	3e	4f
0x60	12	50	62	8a	5e	5f	df	ea	99	ac	74	6b	91	44	33	ef
0x70	70	71	72	73	74	75	76	77	78	79	7a	7b	7c	7d	7e	7f
0x80	91	40	8a	00	8c	14	fe	ff	74	13	02	ba	6b	12	4b	31
0x90	90	91	92	93	94	95	96	97	98	99	9a	9b	9c	9d	9e	9f
0xa0	80	00	8a	00	8c	14	fe	ff	72	14	0a	6b	10	02	e1	ba
0xb0	30	31	32	33	34	35	36	37	38	39	3a	3b	3c	3d	3e	3f
0xc0	80	00	8a	00	8c	14	fe	ff	72	14	0a	6b	10	01	e1	ba
0xd0	91	40	8a	00	8c	14	fe	ff	74	13	02	ba	6b	12	4b	31
0xe0	70	00	8a	00	8c	14	fe	ff	72	14	0a	6b	10	03	e1	ba
0xf0	91	40	8a	00	8c	14	fe	ff	74	13	02	ba	6b	12	4b	31

offset = $\log_2(2\text{MB})$ = 21 bits

remaining = 32 − 21 = 11 bits

n = 11

59

(a) What is the physical address for a virtual address `0x0000beef`? Put in **Not enough information** if the table does not provide enough information to get the physical address.

0x0000beef

VPN = 0000 0000 000 → 0

PO = 0 0000 1011 1110 1110 1111

Page Table Entry → 10 11 12 13

PPN = 0001 0000 000

PA = 0001 0000 0000 0000 1011 1110 1110 1111

1 0 0 0 b e e f

= 0x1000beef

Initials: Manicit Gandhi 6280004

(b) What is the physical address for a virtual address 0x0fffffff? Put in **Not enough information** if the table does not provide enough information to get the physical address.

0x0fffffff

VPN = 0000 1111 111 → 127 (Not present in table)

0 f f

PO = 1 1111 1111 1111 1111 1111

f f f f f f

Not enough info

(c) What is the physical address for a virtual address 0x0000beef if this system were to use 16KB page instead? Put in **Not enough information** if the table does not provide enough information to get the physical address.

If this system used 16 KB page, offset = $\log_2(16kb)$ = 14 bits
remaining = 32 − 14 = 18 bits

0x0000beef

VPN = 0000 0000 0000 0000 10 → 2

0 0 0 0 b

PO = 11 1110 1110 1111

b e e f

Page Table Entry → 18 19 1a 16

PPN = 0001 1000 0001 1001 00

1 8 1 9 1

PA = 0001 1000 0001 1001 0011 1110 1110 1111

1 8 1 9 3 e e f

= 0x 18193eef

(d) Assuming that the memory access takes 100 cycles to access DRAM, the system has 4-level page table (i.e., a page walk have to access the memory 4 times before it can access its data), an TLB access takes 1 cycle, and a L1 cache access to the set takes 1 cycle and the tag comparison in the L1 cache takes another 1 cycle. How long does it takes to load a data that has a TLB miss and a L1 cache hit in a virtualized environment where nested page table is being used? Feels free to explain your answer.

Check TLB = 1 cycle
Page walk = 100 cycles × 4 levels = 400 cycles
get set = 1 cycle
tag comparison = 1 cycle

Since its VIPT, get set and check TLB are parallel, so we consider that as 1 cycle.

400 + 1 + 1 = 402 cycles

(e) (Extra credit: 4 points) Assuming the same setup as part d, but in this case, we utilize a container that runs all it's processes under the host machine's user space (i.e., all virtual addresses used in the container are already in the host's user space). What is the latency of a cache access if you incur a TLB miss and an L1 cache hit? Please explain your answer.

container | host

VA
↓
PTE lvl 4 → host VA_1 → host PA_1
↓
lvl 3 → host VA_2 → host PA_2
↓
lvl 2 → host VA_3 → host PA_3
↓
lvl 1 → host VA_4 → host PA_4
↓
PA → host VA_5 → host PA_5

From the above diagram, we see that 2 page walks are required for each level of page table.
2 additional page walk is required at the end.
Consider 1 lvl = 2 page walks = 100 × 2 = 200 cycles

Total page walks = 200 × 5 = 1000 cycles
Total TLB access, get set, tag comparison = 2 × 2 = 4

Total latency = 1000 + 4 = $\boxed{1004 \text{ cycles}}$

Initials: Manit Gandhi 6280004

(f) (Extra credit: 1 points) What are the names of my cats?

Namtaan.
Sorry krub, I forgot the other cat's name

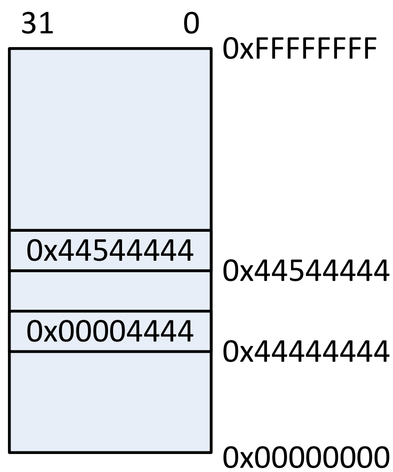
6. Extra Credit: 0x44444444 [20 points]

Do not attempt this until you are done with other questions.

A 32-bit processor implements paging-based virtual memory using a single-level page table. The following are the assumptions about the processor's virtual memory.

- A page table entry (PTE) is 4-bytes in size.
- A PTE stores the physical page number in the least-significant bits.
- The base address of the page tables is page-aligned.

The following figure shows the physical memory of the processor at a particular point in time.



4GB Physical Memory

At this point, when the processor executes the following piece of code, it turns out that the processor accesses the page table entry residing at the physical address of `0x44444444`.

```
char *ptr = 0x44444444;
char val = *ptr; // val == 0x44
```

Initials: Manit Gandhi 6280004

What is the page size of the processor? Show work in detail.

Let $n = \log_2(\text{page size})$

$VA = 0x44444444$

$VPN = VA >> n$

PTE $PA = 0x44444444$

PTE Size = 4

$\underbrace{\text{PT Base Address}}_{\text{PTBA}} = \text{PTE PA} - \text{VPN} * \text{PTE size}$

Since base address is page aligned:

$PTBA \ \& \sim(1<<n) = 0$

$\therefore (PTE\_PA - VPN * \text{PTE Size}) \ \& \sim(1<<n) = 0$

$\therefore (0x44444444 - (0x44444444 >> n)*4) \ \& \sim(1<<n) = 0$

$\therefore (0x44444444 - (0x44444444 >> (n-2))) \ \& \sim(1<<n) = 0$

Therefore the form n is $4k+2$. k is an integer.

The possible values of n: 6, 10, 14, 18, 22, 26, 30

We will now verify the values to see if its equal to 0.

$n=6$: $(0x44444444 - (0x44444444 >> 4)) \ \& \sim(1<<6) - 0x40000000 \ \& \ 0x0000003F = 0$

$n=10$: $(0x44444444 - (0x44444444 >> 8)) \ \& \sim(1<<10) - 0x44000000 \ \& \ 0x000003FF = 0$

$n=14$: $(0x44444444 - (0x44444444 >> 12)) \ \& \sim(1<<14) - 0x44400000 \ \& \ 0x00003FFF = 0$

$n=18$: $(0x44444444 - (0x44444444 >> 16)) \ \& \sim(1<<18) - 0x44440000 \ \& \ 0x0003FFFF = 0$

$n=22$: $(0x44444444 - (0x44444444 >> 20)) \ \& \sim(1<<22) - 0x44444000 \ \& \ 0x003FFFFF != 0$

$n=26$: $(0x44444444 - (0x44444444 >> 24)) \ \& \sim(1<<26) - 0x44444400 \ \& \ 0x03FFFFFF != 0$

$n=30$: $(0x44444444 - (0x44444444 >> 28)) \ \& \sim(1<<30) - 0x44444440 \ \& \ 0x3FFFFFFF != 0$

Therefore, we have now obtained the possible values of n.
6, 10, 14, 18.

Physical Address 0x44454444 is not a PTE.
We will check whether the last PTE of the table is stored at a lower physical address than 0x44454444.

PA of last PTE (LPTE PA)

$$= PTBA + ((1 << (32-n)) - 1) * PTE\_Size$$

$$LPTE\_PA = PTBA + ((1 << (34-n)) - 4)$$

$n = 6$:

$0x40000000 + ((1<<28)-4) = 0x40000000 + 0x0ffffffc = 0x4ffffffc > 0x44454444$

$n = 10$:

$0x44000000 + ((1<<24)-4) = 0x44000000 + 0x00fffffc = 0x44fffffc > 0x44454444$

$n = 14$:

$0x44400000 + ((1<<20)-4) = 0x44400000 + 0x000ffffc = 0x444ffffc > 0x44454444$

$n = 18$:

$0x44440000 + ((1<<16)-4) = 0x44440000 + 0x0000fffc = 0x4444fffc < 0x44454444$

Therefore n is 18

Log Table

N	$log_2 N$
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1024 (1k)	10
2048 (2k)	11
4096 (4k)	12
8192 (8k)	13
16384 (16k)	14
32768 (32k)	15
62236 (64k)	16
131072 (128k)	17
262144 (256k)	18
524288 (512k)	19
1048576 (1M)	20
2097152 (2M)	21
4194304 (4M)	22
8388608 (8M)	23
16777216 (16M)	24

Stratchpad

Stratchpad