Review Session

CSCI 2021, Spring 2020

Structure

Review Problems 1-9 with their solutions.

Buffer Overflows

```
int main(){
   char x[4];
   char y[4];
   gets(x);
   gets(y);
   printf("%s %s", x, y);
   return 0;
}
```

```
main:
        pushq   %rbp
        movq %rsp, %rbp
        subq  $16, %rsp
        #Location 1
        leaq   -4(%rbp), %rax
        movq %rax, %rdi
        movl  $0, %eax
        call    gets
        leaq   -8(%rbp), %rax
        movq %rax, %rdi
        movl  $0, %eax
        call    gets
        #Location 2
        movl  $0, %eax
        ...
```

Address	Variable	Initial Value
0x108	y[0]	0x0
0x109	y[1]	0x0
0x10a	y[2]	0x0
0x10b	y[3]	0x0
0x10c	x[0]	0x0
0x10d	x[1]	0x0
0x10e	x[2]	0x0
0x10f	x[3]	0x0

Fill in the third column of the table for the following input -

i. abc efgh

What is printed on the screen by the printf statement ?

Input - abc efgh

Address	Variable	Value
0x108	y[0]	‘e’
0x109	y[1]	‘f’
0x10a	y[2]	‘g’
0x10b	y[3]	‘h’
0x10c	x[0]	‘\0’
0x10d	x[1]	‘b’
0x10e	x[2]	‘c’
0x10f	x[3]	‘\0’

Code Optimization - Machine Independent Techniques

```
int dot_product_0(char *x, char*y, int size){
        int sum=0;
        for (int i=0; i<size; i++){
                sum = sum + x[i]*y[i];
                }
        return sum;
}
```

```
int dot_product_1(char *x, char*y, int size){
        int sum=0;
        int sum_1=0
        for (int i=0; i<size; i+=2){
                sum = sum + x[i]*y[i];
                sum_1 = sum_1 + x[i+1]*y[i+1];
                }
        return sum+sum_1;
}
```

Which code would you expect faster ? Why ?
How do we increase speed even further ?

Code Optimization - Machine Independent Techniques

```
int dot_product_1(char *x, char*y, int size){
        int sum=0;
        int sum_1=0
        for (int i=0; i<size; i++){
                sum = sum + x[i]*y[i];
                sum_1 = sum_1 + x[i+1]*y[i+1];
                }
        return sum+sum_1;
}
```

```
int dot_product_1(int *x, int*y, int size){
        int sum=0;
        int sum_1=0;
        int sum_2=0;
        int sum_3=0;
        int sum_4=0;
        int sum_5=0;
        int sum_6=0;
        int sum_7=0;
        for (int i=0; i<size; i+=8){
                sum = sum + x[i]*y[i];
                sum_1 = sum_1 + x[i+1]*y[i+1];
                sum_2 = sum_2 + x[i+2]*y[i+2];
                sum_3 = sum_3 + x[i+3]*y[i+3];
                sum_4 = sum_4 + x[i+4]*y[i+4];
                sum_5 = sum_5 + x[i+5]*y[i+5];

                sum_6 = sum_6 + x[i+6]*y[i+6];
                sum_7 = sum_7 + x[i+7]*y[i+7];

        }
        return sum + sum_1 + sum_2 + sum_3 + sum_4 + sum_5 +
sum_6 + sum_7;
}
```

Faster due to loop unrolling optimization.
Speed can be increased even further by unrolling the loop even further. Experimentally found that it was about 20% faster with 8-way unrolling.

Linking -- Symbols, local and global

```
/* foo5.c */
#include <stdio.h>

void f(void);

int x= 65536;
int y= 1024;

int main(){

        f();
        printf("x= 0x%x y= 0x%x\n", x, y);
        return 0;
}
```

```
/*  bar5.c */
double x;


void f(){
        x=-0.0;
}
```

What is the expected result upon running this program on a 64 bit machine ?
Can you explain why ?

Linking -- Symbols, local and global

x= 0x0 y= 0x0

The main cause of this unexpected result is that x is declared as both type ‘int’ and ‘double’. It is initialized as an int type in foo5.c, but it is updated by f(), which thinks it is a double (8 bytes). Therefore, 4 bytes of ‘y’ get overwritten.

Dynamic Memory Allocation - C memory usage mistakes

```
void  initialize(int *x, int s){
        x = (int *)malloc(s*sizeof(int));
        for (int i=0; i<s;i++) {x[i]=rand();}
        return ;
}

int main(){
        srand(time(0));
        int *x;
        initialize(x,10);
        for (int i=0; i<10; i++){
                printf("Initialized x[i] to %d\n", x[i]);
        }
        return 0;
}
```

What are the errors in this code and the possible fixes ?

Dynamic Memory Allocation - C memory usage mistakes

```
void  initialize(int *x, int s){
        for (int i=0; i<s;i++) {x[i]=rand();}
      return ;
}

int main(){
      srand(time(0));
      x = (int *)malloc(s*sizeof(int));
      int *x;
      initialize(x,10);
      for (int i=0; i<10; i++){
            printf("Initialized x[i] to %d\n", x[i]);
      }
      free(x);
      return 0;
}
```

Solution 1

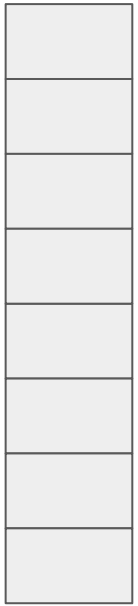
```
void  initialize(int **x, int s){
      *x = (int *)malloc(s*sizeof(int));
      for (int i=0; i<s;i++) {(*x)[i]=rand();}
      return ;
}

int main(){
      srand(time(0));
      int *x;
      initialize(&x,10);
      for (int i=0; i<10; i++){
            printf("Initialized x[i] to %d\n", x[i]);
      }
      return 0;
}
```

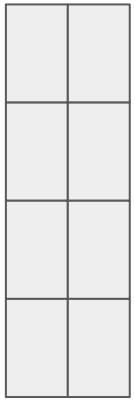
Solution 2

Caches - cache parameter and operation

Direct Mapped



Set Associative
(2-way)



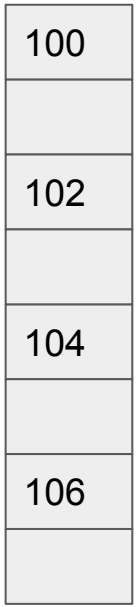
100, 102, 104, 106, 100, 102, 104, 106 ...

100, 104, 100, 104, 100, 104, 100, 104 ...

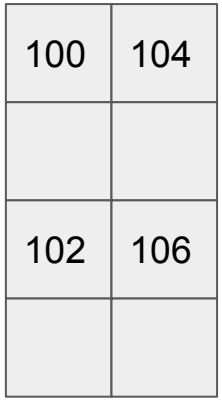
100, 101, 102, 103, 100 ...

The cache lines are 1 byte in size.
What is the cache hit rate for the direct mapped and set associative cache, for each access pattern ?

Direct Mapped

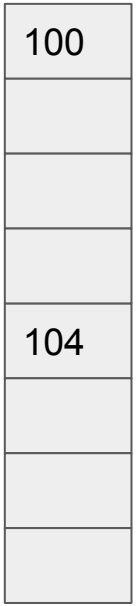


Set Associative (2-way)

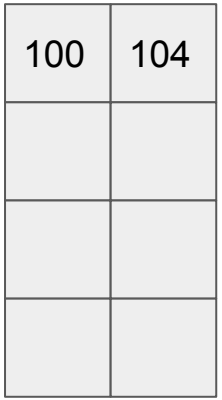


Hit Rate = 100 ; Hit Rate = 100

Direct Mapped

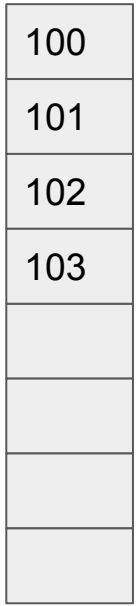


Set Associative (2-way)

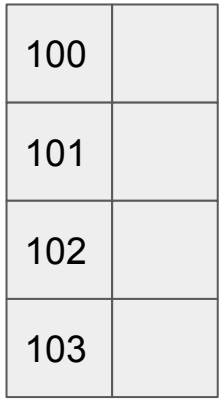


Hit Rate = 100 ; Hit Rate = 100

Direct Mapped



Set Associative (2-way)



Hit Rate = 0 ; Hit Rate = 0

Number Representation - Bits and Bitwise Operators

Write a C program to check if the number of bits in a char type is even or odd.

If it is even, then it should return 0, otherwise, it should return 1.

A Possible Solution :

```
int oddEven(char x){

        int r=0;
        while (i < 8){
           r = r + (x >> 1) & 1;
        }
        return r & 1;

}
```

Virtual Memory -- Page Tables and TLBs

Tag	PPN
0x15	0x2b
0x15	0x1b

TLB

VPN	PPN	Valid
0x2a	0x17	Y
0x2f	0x26	Y
--	--	N

Page Table

Virtual address access List :

0xabcd, 0xabba, 0xbeef, 0xdead

16 bit address --
i. Page Offset (10 bits)
ii. PPN (6 bits)

Please specify for each access if a TLB hit/miss occurred, and whether a Page Fault occurred, and the physical address. The initial contents of the TLB and the page table are shown on the left.

Virtual Memory -- Page Tables and TLBs

Virtual Address	VPN	Physical Address	TLB Hit	Page Fault	PPN
0xabcd	0x2a	0x5fcd	Y	N	0x2b
0xabba	0x2a	0x5fba	Y	N	0x2b
0xbeef	0x2f	0x9aef	N	N	0x26
0xdead	0x37	--	N	Y	--

Tag	PPN
0x17	0x26
0x15	0x1b

TLB final state

Virtual Address	VPN	Offset	PPN	Offset	Physical Address
0xabcd -- 1010 1011 1100 1101	-- 101010	1111001101	-- 010111	1111001101	-- 0101 1111 1100 1101
0xabba -- 1010 1011 1011 1010	-- 101010	1110111010	-- 010111	1110111010	-- 0101 1111 1011 1010
0xbeef -- 1011 1110 1110 1111	-- 101111	1011101111	--100110	1011101111	-- 1001 1010 1110 1111
0xdead -- 1101 1110 1010 1101	-- 110111	1010101101			

Logic Design -- Boolean Functions

Design a logic circuit that finds the second smallest value among the set of 3 words, A, B, and C, using an HCL case expression.

Logic Design -- Boolean Functions

A possible solution is as follows :

```
(A <= B && A >= C) || (A >= B && A <= C): A;

(B <= C && B >= A) || (B >= C && B <= A): B;

1: C;
```

CPU architecture -- Y86-64

Please write a Y86-64 program that implements a swap function, similar to the C code :

```
void swap(int *x, int *y){
    int temp_1 = *x;
    int temp_2  = *y;
    *y = temp_1;
    *x = temp_2;
}
```

CPU architecture -- Y86-64

A possible solution is as follows :

```
swap:

        mrmovq  (%rdi), %rcx
        mrmovq  (%rsi), %rax
        rmmovq  %rdx, (%rsi)
        rmmovq  %rcx, (%rdi)
        ret
```

Thank You

Questions ?