Discussion Problems

CSci 2021: Machine Architecture and Organization
April 27th, 2016
Substitute instructor: Prof. Pen-Chung Yew

Simple Memory System Example

1. Simple Memory System TLB
   - 16 entries
   - 4-way associative

2. Simple Memory System Page Table
   Only show first 16 entries (out of 256)

3. Simple Memory System Cache
   - 16 lines, 4-byte block size
   - Physically addressed
   - Direct mapped

Address Translation Example #1

Virtual Address: 0x030F

Addressing
- 14-bit virtual addresses
- 12-bit physical address
- Page size = 64 bytes
Address Translation Example #2

Virtual Address: 0x0214

<table>
<thead>
<tr>
<th>TLB1</th>
<th>TLB0</th>
<th>VPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

TLBI TLBT

<table>
<thead>
<tr>
<th>VPN</th>
<th>PXN</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
</tbody>
</table>

TLB Hit? Y

Page Fault? N

Physical Address

<table>
<thead>
<tr>
<th>CT</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PPO</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
</tr>
</tbody>
</table>

What’s wrong with this C code? #1

```c
int *add(int a, int b) {
    int sum = a + b;
    return &sum;
}
```

- Sum is a local variable on the stack, which will not exist after the function returns.

What’s wrong with this C code? #1 fix

```c
int *add(int a, int b) {
    int *sum = malloc(sizeof(int));
    if (!sum)
        /* handle error... */
    *sum = a + b;
    return sum;
}
```

What’s wrong with this C code? #2

```c
char *f(void) {
    char *buf = malloc(11);
    int i;
    for (i = 0; i <= 11; i++)
        buf[i] = 'A';
    return buf;
}
```

- Off by one access. The loop goes up to i = 11, but the legal array indexes are only 0 through 10.

What’s wrong with this C code? #2 fix

```c
char *f(void) {
    char *buf = malloc(11);
    int i;
    for (i = 0; i < 11; i++)
        buf[i] = 'A';
    return buf;
}
```

What’s wrong with this C code? #3

```c
char *f(void) {
    char *buf = malloc(11);
    int i;
    for (i = 0; i < sizeof(buf); i++)
        buf[i] = 'A';
    return buf;
}
```

- This won’t crash, but it doesn’t do what the previous code did. `sizeof()` on a pointer returns 8 (on 64-bit), not the size of the buffer pointed to.

N.B. To save space, we will leave out the malloc() return value checking. But if you do this, your program will crash when it runs out of memory.
What’s wrong with this C code? #4

```c
char *f(void) {
    char *buf = malloc(11);
    int i;
    for (i = 0; i < strlen(buf); i++)
        buf[i] = 'A';
    return buf;
}
```

- Read of uninitialized memory. strlen looks through the buffer for a null byte, but there might not even be one anywhere.

What’s wrong with this C code? #5

```c
int read_int(void) {
    int *x_ptr;
    scanf("%d", &x_ptr);
    return *x_ptr;
}
```

- Use of uninitialized variable: `x_ptr` never got a value before scanf tried to write through it.

Loop unrolling

- Here is some code for multiplication using addition in a loop:

```c
long mult(int a, int b) {
    long prod = 0;
    int i;
    for (i = 0; i < a; i++) {
        prod += b;
    }
    return prod;
}
```

(Even if C didn’t have a * operator, this is not a good algorithm.)

- How would you unroll this loop by 4?

Loop unrolling: basic idea

- Here’s the basic structure:

```c
for (i = 0; i < ...; i += 4) {
    prod += b;
    prod += b;
    prod += b;
    prod += b;
}
```

- The remaining parts are figuring out what goes in the ..., and how to handle the case when `a` is not a multiple of 4.

Loop unrolling: complete

```c
int limit = a - 3;
for (i = 0; i < limit; i += 4) {
    prod += b;
    prod += b;
    prod += b;
    prod += b;
}
for (; i < a; i++) {
    prod += b;
}
```

- `limit = a, limit = a – 1, or limit = a – 2` would give wrong answers
- `limit = a – 4, etc.,` would be correct but less optimized
- Notice the final loop is like the original one

Cache parameters

- Assume the following caches all have 64-byte blocks, and other parameters as shown:

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>E</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>32 KB</td>
<td>1</td>
<td>512</td>
</tr>
<tr>
<td>B</td>
<td>32 KB</td>
<td>8</td>
<td>64</td>
</tr>
<tr>
<td>C</td>
<td>32 KB</td>
<td>512</td>
<td>1</td>
</tr>
</tbody>
</table>

- Which cache needs the most gates? C
- Which cache has the fastest hit time? A
- Which cache has the lowest miss rate? C
- Which cache is found in a real Core i7? B
Y86-64 compiling

- Suppose you have the following C code:
  ```c
  long ary[10][10];
  ary[i][j]++;
  ```
- Assuming that ary is in %rax, i is in %rdi, and j is in %rsi, how would you write Y86-64 code to implement this?

  Hint: start by writing an expression for the address &ary[i][j]:
  ```c
  ary + 8*(10 * i + j)
  ```

Y86-64 compiling, steps

- On x86-64, how do you multiply by 2 without a multiply instruction? (There are several ways)
- Which way is possible with Y86-64 instructions?
- How about multiplying by 5?
- How about multiplying by 8 and 10?