T-107-TOLH, Homework Assignment V

October 28, 2015

"There are only two hard problems in Computer Science: cache invalidation and naming things."

-Phil Karlton

Instructions:

- This assignment will be answered through Skel.
- You *must* hand in this assignment through Skel or it will **not** be graded.

Handout instructions: Connect to Skel using your favorite `ssh` client and unpack the assignment into your home directory by running the following command:

```
[student15@skel ~]$ tar xzvf /labs/tolh15/hw5/$(whoami)/hw5.tar.gz
[student15@skel ~]$ cd tolh15-hw5
[student15@skel tolh15-hw5]$ ls
answers  assignment  problem1  problem2  problem3  problem4  problem5
[student15@skel tolh15-hw5]$
```

To hand in the assignment, you **must** run the "./assignment handin" command inside the "hw5" directory. This will archive a copy of your assignment into a file called "/labs/tolh15/.handin/hw5/$(whoami)/handin.tar.gz". If the handin file does not exist, then you will not get a grade for this assignment.

```
[student15@skel ~]$ cd tolh15-hw5
[student15@skel tolh15-hw5]$ ./assignment handin
[student15@skel tolh15-hw5]$ ls /labs/tolh15/.handin/hw5/$(whoami)/handin.tar.gz
/labs/tolh15/.handin/hw5/student15/handin.tar.gz
```

To see when you last handed inn the assignment, then run the "./assignment check" command.

```
[student15@skel tolh15-hw5]$ ./assignment check
Last handin: 2015-09-16 16:13:37
```

This is the same as running the command.

```
[student15@skel tolh15-hw5]$ rutool check -c tolh15 -p hw5
Last handin: 2015-09-16 16:13:37
```

Before you start this homework assignment, you must first set your preferred editor using the "./assignment set editor" command. The classical choices are nano, vim, and emacs.

```
[student15@skel tolh15-hw5]$ ./assignment config set editor nano
# or
[student15@skel tolh15-hw5]$ ./assignment config set editor vim
# or
[student15@skel tolh15-hw5]$ ./assignment config set editor emacs
```

Every problem should be answered using the "./assignment" program which is located in the tolh15-hw5 directory. You can answer each question individually or all of them by running "./assignment" with the following parameters.

```
# Only answer problem1
[student15@skel tolh15-hw5]$ ./assignment problem1
# Answer all problems in the assignment
[student15@skel tolh15-hw5]$ ./assignment all
```

Question 1 (13 points)

Consider the following (awesome) cache problem.

You may assume the following:

- The memory is byte addressable.
- Memory accesses are to **1-byte words** (not 4-byte words).
- Physical addresses are 9 bits wide.
- The cache is 4-way set associative, with a 2-byte block size and 16 total lines.

In the following tables, **all numbers are given in hexadecimal**. The contents of the cache are as follows:

4-way Set Associative Cache																
Index	Valid	Tag	Byte 0	Byte 1	Valid	Tag	Byte 0	Byte 1	Valid	Tag	Byte 0	Byte 1	Valid	Tag	Byte 0	Byte 1
0	1	3E	F7	54	0	1F	7A	15	1	19	47	83	1	04	BD	F5
1	1	2A	C9	2C	1	18	E6	53	0	10	80	A4	1	3C	13	2E
2	0	1F	D7	F5	1	23	3B	3A	1	33	A6	35	1	0C	66	5B
3	1	3C	1F	F4	1	07	3E	18	1	28	8D	89	1	3A	8E	AE

Part 1

The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following:

CO The block offset within the cache line
CI The cache index (Also called set index)
CT The cache tag

8	7	6	5	4	3	2	1	0

Part 2

For the given physical address, indicate the cache entry accessed and the cache byte value returned **in hex**. Indicate whether a cache miss occurs.

If there is a cache miss, enter "-" for "Cache Byte returned".

Physical address: `0x064`

(a) Physical address format (one bit per box)

8	7	6	5	4	3	2	1	0

(b) Physical memory reference

Parameter	Value
Cache Offset (CO)	0x
Cache Index (CI)	0x
Cache Tag (CT)	0x
Cache Hit? (Y/N)	
Cache Byte returned	0x

Physical address: `0x147`

(a) Physical address format (one bit per box)

8	7	6	5	4	3	2	1	0

(b) Physical memory reference

Parameter	Value
Cache Offset (CO)	0x
Cache Index (CI)	0x
Cache Tag (CT)	0x
Cache Hit? (Y/N)	
Cache Byte returned	0x

Question 2 (16 points)

You are writing a new 3D game that you hope will earn you fame and fortune. You are currently working on a function to blank the screen buffer before drawing the next frame. The screen you are working with is a 1920x1080 array of pixels. The machine you are working on has a 64 KiB direct mapped cache with 4 byte lines. The C structures you are using are:

```
struct pixel {
    char r;
    char g;
    char b;
    char a;
};

struct pixel buffer[1920][1080];
register int i, j;
register char *cptr;
register int *iptr;
```

Assume:

- `sizeof(char) = 1`
- `sizeof(int) = 4`
- `buffer` begins at memory address 0
- The cache is initially empty.
- The only memory accesses are to the entries of the array `buffer`. Variables `i`, `j`, `cptr`, and `iptr` are stored in registers.

(a) (4 points) What percentage of the writes in the following code will miss in the cache?

```
for (j=0; j < 1920; j++) {
    for (i=0; i < 1080; i++){
        buffer[i][j].r = 0;
        buffer[i][j].g = 0;
        buffer[i][j].b = 0;
        buffer[i][j].a = 0;
    }
}
```

Miss rate for writes to `buffer`: ______________

(b) (4 points) What percentage of the writes in the following code will miss in the cache?

```
char *cptr;
cptr = (char *) buffer;
for (; cptr < (((char *) buffer) + 1920 * 1080 * 4); cptr++)
    *cptr = 0;
```

Miss rate for writes to `buffer`: ______________

(c) (4 points) What percentage of the writes in the following code will miss in the cache?

```
int *iptr;
iptr = (int *) buffer;
for (; iptr < (buffer + 1920 * 1080); iptr++)
    *iptr = 0;
```

Miss rate for writes to `buffer`: ______________

(d) (4 points) Which code (A, B, or C) should be the fastest? ______________

Question 3 (14 points)

In this problem, your awesome knowledge of pointers and pointer related arithmetic is tested.

Assume an x86 architecture.

```
int x[5] = {0, 1, 2, 3, 4}; // This array is at address 0x08048c00
short y[3] = {0, 1, 2}; // This array is at address 0x08048c14

a = x[0];
b = &x[0];
c = x + a + 3;
d = *c + 3;
e = (x + 2)[2];
f = &y[2] - 2;
g = *(&y + 1);
h = *(x + 5);
```

Fill in the following table. Give your answers in hexadecimal.

Denote any unknown variables by `?`.

Expression	Type	Value
a	int	0
d		
f		
h		
c		
g		
b		
e		

Question 4 (18 points)

Consider the source code bellow where we have a main function that calles the function foo that takes two parameters.

```
foo(int a, int *b)
{
    ...
}

int main()
{
    ...
    int a;
    int b;
    /* Missing code that sets values to variables a and b */
    foo(a,&b);
    ...
}
```

Assembly:

```
foo:
   0x08048475 <foo+0>:  push   %ebp
=> 0x08048476 <foo+1>:  mov    %esp,%ebp
```

A memory dump of the stack when the `%eip` register is 0x08048476 after the foo function has been called from main. This is the same as doing x/136xb $esp in gdb.

```
0xffffd5c8: 0x08 0xd6 0xff 0xff 0x45 0x84 0x04 0x08
0xffffd5d0: 0xae 0x08 0x00 0x00 0xec 0xd5 0xff 0xff
0xffffd5d8: 0x08 0xd6 0xff 0xff 0x99 0x84 0x04 0x08
0xffffd5e0: 0xe0 0x61 0x84 0x00 0x41 0x82 0x04 0x08
0xffffd5e8: 0xe0 0x8c 0x84 0x00 0x05 0x0d 0x00 0x00
0xffffd5f0: 0x80 0x84 0x04 0x08 0x40 0x83 0x04 0x08
0xffffd5f8: 0xf4 0x7f 0x84 0x00 0x00 0x00 0x00 0x00
0xffffd600: 0x80 0x84 0x04 0x08 0x00 0x00 0x00 0x00
0xffffd608: 0x88 0xd6 0xff 0xff 0xe6 0xcc 0x6c 0x00
0xffffd610: 0x03 0x00 0x00 0x00 0xb4 0xd6 0xff 0xff
0xffffd6e8: 0xe0 0x8c 0x84 0x00 0x05 0x0d 0x00 0x00
0xffffd6f0: 0x80 0x84 0x04 0x08 0x40 0x83 0x04 0x08
0xffffd6f8: 0xf4 0x7f 0x84 0x00 0x00 0x00 0x00 0x00
0xffffd600: 0x80 0x84 0x04 0x08 0x00 0x00 0x00 0x00
0xffffd608: 0x88 0xd6 0xff 0xff 0xe6 0xcc 0x6c 0x00
0xffffd610: 0x03 0x00 0x00 0x00 0xb4 0xd6 0xff 0xff
0xffffd618: 0xc4 0xd6 0xff 0xff 0xd0 0xd3 0xff 0xf7
```

Before foo is called, the variables a and b are assigned values in main (hidden in the above code).

Write your answer as hexadecimal numbers.

Find the values of old ebp, the return address, a ,and b by reading the above stack frame. Assume that bytes are in little-endian order.

old `ebp`: ______________ return address: ______________ value of `a`: ______________ value of `b`: ______________

Question 5 (18 points)

In this problem, you are to implement a couple of functions in x86_64 assembly. Use your favourite text editor on skel to work on the solution file, `solution64.s`.

You are provided a tool to test your solutions for correctness. It will test your implementation with random integers and inform you whether your output was correct or not.

```
[student15@skel problem5]$ make
[student15@skel problem5]$ ./asm 64
```

A reference implementation of these functions have been made available to you in the C programming language.

```
int add(int a, int b) {
    /* Compute the sum of the two integers ‘a’ and ‘b’. */
    return a + b;
}

int sub(int a, int b) {
    /* Subtract ‘b’ from ‘a’. */
    return a - b;
}

int sum(int a, int b, int c, int d, int e, int f, int g) {
    /* Compute the sum of the seven integers given as parameters. */
    return a + b + c + d + e + f + g;
}

int max(int a, int b) {
    /* Return the larger of the two integers ‘a’ and ‘b’. */
    if (a > b)
        return a;
    else
        return b;
}

int cmp(int a, int b) {
    /* Return -1, 0, or 1, if ‘a’ is less than, equal to,
     * or greater than ‘b’, respectively. */
    if (a > b)
        return 1;
    else if (a == b)
        return 0;
    else
        return -1;
}

int idiv(int a, int b) {
    return a / b;
}

int mod(int a, int b) {
    return a % b;
}
```

```
int fib(int a, int b) {
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return fib(n-1) + fib(n-2);
}
```