Assignment 3: Virtual Cache Memory

Due: 11:59PM Tuesday, November 22, 2022

Grader (TA): Ruxin Wang <rwang31@lsu.edu>

In this assignment, you will write an IA32 assembly routine using gnu assembly. First, create the directory **`prog3`**, which will be used for submission. Change to this directory, and issue the following command to get the files you need (Note: Do not ignore the period at the end of the command):

```
cp ~cs3501_lee/cs3501_F22/p3/* .
```

View and understand the provided `Makefile` and `cache.c` so that you can write your own in the future.

In this assignment, you will write an assembly program in the provided file `prog3.s`. The file will implement a function with the prototype:

```
unsigned char check_cache(line cache[4], unsigned char addr);
```

This function will check whether the given cache stores the data at the given memory address. If the cache stores the data at the given memory address (i.e., cache hit), this function will return the stored data. Otherwise (i.e., cache miss), this function will return `0xFF`. Here are important assumptions about the function.

- Memory addresses are **8 bits** wide (`m=8`). Therefore, we use `unsigned char` to store memory addresses.
- Memory accesses are to **1-byte words** (not to 4-byte words). Therefore, the return type of the function is `unsigned char`.
- The cache is directed-mapped (`E=1`), with a 4-byte block size (`B=4`) and four sets (`S=4`).
- A cache line is defined using `struct` as follows. `valid` can store only 0 or 1. If the number of tag bits (i.e., `t`) is less than 8 bits (the size of `char`), `t` low-order bits of `tag` will be used to store the tag bits, and the unused bits will be zero.

```
typedef struct {
        char block[4];
        char valid;
        char tag;
} line;
```

- Cache blocks do not store byte data `0xFF` because it is used for indicating cache miss in this function.

Note: You are required to write a **comment** for **each instruction** to explain its purpose. You can write a comment by beginning with the "**#**" sign in your assembly code. In addition, you should include your **full name** and **LSU ID** number as a comment in your code.

In addition, when you modify any callee-save register (`%ebx, %esi, or %edi`), you need to save and restore its old value.

I suggest you get this working in the following steps:

1. Find the number of tag bits, the number of set index bits, and the number of block offset bits using the given information above.
2. Store the given memory address in a byte-sized register and then split the address into three components (tag bits, set index bits, and block offset bits). You may want to use bit-level operations such as `andb` and `shrb` and 1-byte move instruction `movb` to split the address. You can check each component's value by putting it in the register used for storing return values (e.g., `%al` for 1-byte return values) before moving onto the next steps.
3. Find the start address of the corresponding cache set using the set index bits. You need to use the size of `struct line` to find the start address of the cache set. It is okay to use multiplication instructions. You may want to use `movzbl` for converting 1-byte offsets to 4-byte offsets (e.g., `movzbl %ch, %ebx`).
4. Read the valid bit from the cache set using `movb`. If the valid bit has `0` (i.e., cache miss), store `0xFF` in the return register (i.e., `%al`) and finish the function. If the valid bit has `1`, move onto the next step. You may want to use `testb` for this comparison.
5. Read the tag bits from the cache set using `movb` and then compare them with the tag bits in the memory address. If they are not the same (i.e., cache miss), store `0xFF` in the return register (i.e., `%al`) and finish the function. Otherwise (i.e., cache hit), move onto the next step. You may want to use `cmpb` for this comparison.
6. Read one byte from the cache block using the block offset bits and then put it in the return register. Again, you need to use `%al` for the 1-byte return value. You may want to use `movzbl` for converting 1-byte offsets to 4-byte offsets (e.g., `movzbl %ch, %ebx`).

⇨ You can hand in with '**`make submit`**'.

Here is an example run of my completed tester:

```
>make
gcc -Wall -g -m32 -c cache.c
gcc -Wall -g -m32 -c prog3.s
gcc -Wall -g -m32 -o xtest cache.o prog3.o

> ./xtest
Enter a memory address (0-255) for cache access: 1
cache HIT for 0x1: 0xb
Enter a memory address (0-255) for cache access: 2
cache HIT for 0x2: 0xc
Enter a memory address (0-255) for cache access: 3
cache HIT for 0x3: 0xd
```

```
Enter a memory address (0-255) for cache access: 4
cache MISS for address 0x4
Enter a memory address (0-255) for cache access: 60
cache HIT for 0x3c: 0x2a
Enter a memory address (0-255) for cache access: 63
cache HIT for 0x3f: 0x2d
Enter a memory address (0-255) for cache access: 64
cache MISS for address 0x40
Enter a memory address (0-255) for cache access: 0
cache HIT for 0x0: 0xa
```

When you are ready to submit, you can do so with "**`make submit`**". Note that you can submit multiple times, but don't do so after the due date! Note that you will be graded using the provided `Makefile`, not your copy of it, so do not count on any changes you make to any file except `prog3.s`.

Important notes:

- You should **NOT** send any assignment-related emails to the instructor. The TA has full control over the assignment grading process. The instructor will NOT reply to assignment-related emails and will NOT answer to assignment-related questions during his office hours. You can still ask high-level concepts.
- You should **NOT** send the TA (or the instructor) any emails enclosing your source code. In fairness to other students, the TA will only check your submitted file(s) after the assignment deadline. If you send any email enclosing your source code, there may be a penalty for your assignment grade. You can still ask high-level concepts.
- If there is no comment in your code, a grade of "zero" will be recorded.
- Your program will be compiled and tested only on the *classes* server for grading. You need to make sure that your code is running well on the server. In other words, even though your program is running well on another machine, that will not be considered during grading.
- It is your responsibility to remember and securely keep your password of the server. If you forget yours, you need to check with the computer manager in the department to reset it. This process may take several days, and you cannot request an extension for this reason.
- Since this assignment is posted about two weeks before its deadline, last-minute requests for extension will not be granted. Being busy can never be a valid reason to request an extension when you have about two weeks for the assignment. The instructor/TA will not reply to such requests.
- It is very likely that last-minute questions sent on the due date cannot be answered by TA before the deadline.

No Additional Task for Honors Option