國立清華大學資訊工程學系

CS 410000 --- 計算機結構

105 學年度上學期

Final Project

(Due: to be announced)

Topic

(Programming Project)

The indexing algorithm design for processor cache

Goal

    i.   Study and implement least recently used replacement policy (LRU).

    ii.   Design a cache indexing scheme to minimize cache conflict miss.

Introduction

In hierarchy memory system, the small memory, cache, is used to keep data temporarily for increasing the system performance. If the data is in cache, processor can get the data with high accessing speed. If the data is not in the cache, called cache miss, processor will read data from main memory. Accessing data form main memory is slower than from cache.

In set-associative or direct mapping scheme, when multiple frequently used data blocks compete for a same cache location, those data blocks will keep kick others out from cache, and results in lots of cache miss, this kind of cache misses is called the conflict misses.
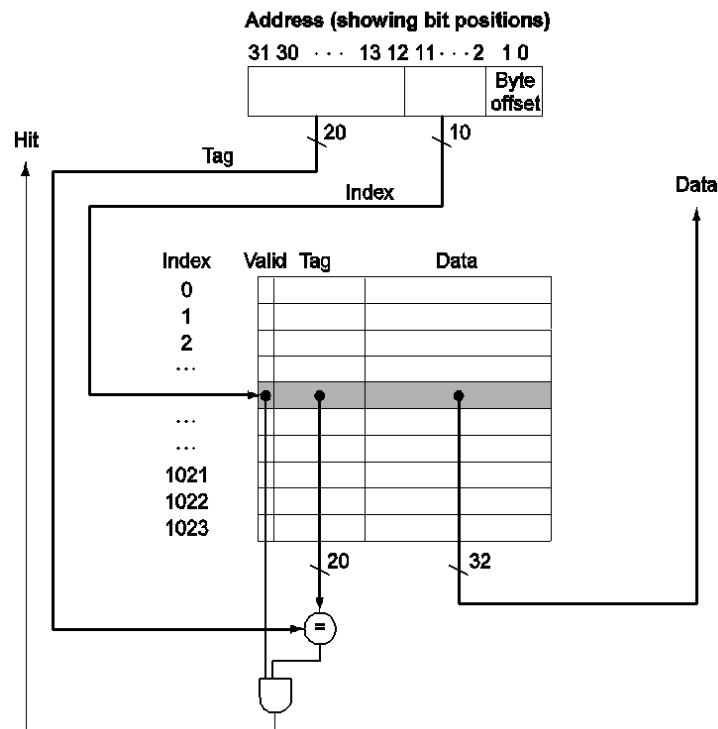
The above figure shows the direct mapping scheme with 1024 cache entries. In direct mapping scheme, each entry has one cache block. Byte offset takes two bits. The cache indexing needs 10 bits of 32 address bits for indexing 1024 entries. The other 20 bits are cache tag.

The cache indexing scheme decides the cache entry in which a data to be stored. If different data accessed frequently are mapped into a same location, the system performance will be degraded due to cache conflict misses. That is, the design target of the cache indexing scheme is to reduce cache conflict misses.

Moreover, in some SOC design, the application software running on the embedded processor is always dedicated, for example, the Fourier transforms in digital signal processing. It is possible to analyze the memory access behavior of the software to optimize the cache indexing to reduce the cache miss. The objective of this project is to design the cache indexing scheme with the minimal cache miss count from the given addressing sequence.

One of the most famous replacement policies is least recently used (LRU) replacement policy. You are required to implement LRU replacement policy in this project.

Problem Definition

Give a cache with E entries and A-way set associativity. And, the addressing bus is M-bit. We need to select $K = \log_2 E$ bits among all address bits for indexing the cache.

There are totally $\binom{M}{K}$ possible valid solutions. Your job is to find a valid solution with minimal cache misses under LRU replacement policy. To facilitate this problem, we assume that the offset is 0-bit, i.e. the block size is 1 byte.

Example 1 – (1-way associative)

If the addressing bus is 6 bits $(a_0\ a_1\ a_2\ a_3\ a_4\ a_5)$ and the cache has 8 Entries and uses direct map scheme, we need $\log_2 8 = 3$ bits for indexing 8 entries (#0~#7). There are $\binom{6}{3} = 20$ possible solutions.

Access Example:

Reference Address	Indexing by (a$_3$ a4 a5)	Cache Location
000100	000100	#4
001011	001011	#3

The reference sequence is as follows:

Reference sequence	Indexing by (a$_3$ a4 a5)		Indexing by (a$_0$ a1 a2)	
(a$_0$ a$_1$ a$_2$ a$_3$ a$_4$ a$_5$)	entry	status	entry	status
001011	#3	miss	#1	miss
001000	#0	miss	#1	miss
001011	#3	hit	#1	miss
001000	#0	hit	#1	miss
Cache miss	2		4	

Example 2 – (2-way associative)

If the addressing bus is 6 bits (a_0 a_1 a_2 a_3 a_4 a_5) and the cache has 4 Entries and uses 2-way associative scheme, we need $\log_2 4 = 2$ bits for indexing 4 entries (#0~#3). There are $\binom{6}{2} = 15$ possible solutions.

The cache organization will be:

Entries	Block 1		Block 2	
#0	tag	data	tag	data
#1	tag	data	tag	data
#2	tag	data	tag	data
#3	tag	data	tag	data

Access Example:

Reference Address	Indexing by (a4 a5)	Cache Location
000100	000100	#0
001011	001011	#3

The reference sequence is as follows:

Reference sequence (a_0 a_1 a_2 a_3 a_4 a_5)	Indexing by (a4 a5)		Indexing by (a2 a3)	
	entry (#block)	status	entry (#block)	status
000000	#0 (b1)	miss	#0 (b1)	miss
000100	#0 (b2)	miss	#1 (b1)	miss
001000	#0 (b1)	miss	#2 (b1)	miss
000000	#0 (b2)	miss	#0 (b1)	hit
001011	#3 (b1)	miss	#2 (b2)	miss
000000	#0 (b1)	hit	#0 (b1)	hit
001011	#3 (b1)	hit	#2 (b2)	hit
Cache miss	5		4	

Related Work

A previous work, "Zero Cost Indexing for Improved Processor Cache Performance " is proposed by Tony Givargis [1]. To refer to this algorithm in your project is welcome, or your new idea is encouraged and will be given additional bonus.

Requirements

- Using C/C++ language. (Your program will be recompiled by g++ to check the correctness, so please make sure your program is compliable to g++.)

- Your program should pass the argument from command, like :
  "./arch_final cache.org reference.lst" (You can open *.org and *.lst with notepad++.)

- Final cache indexing and cache miss count (index.rpt).

- Report; including algorithm description, flow-chart of algorithm, summary of result, and discussion.

- Demonstration. The time will be shown on class website before the final exam.

- Your program should finish in one minute.

- DO NOT COPY, we changed the replacement policy.

Input File Format cache.org

Addressing_Bus : 6

Entries : 4

Associativity : 2

reference.lst

.benchmark testcase1

000000

000100

001000

000000

001011

000000

001011

.end

Notice that, the index of the MSB (first bit) is always 0. Meanwhile, the index of the LSB (last bit) always has the biggest indexing number. For example, the last

address in above "reference.lst" is 001011, which means bit index 0, 1, 3 (a_0, a_1, a_3) are 0, and bit index 2, 4, 5 (a_2, a_4, a_5) are 1.

Output File Format index.rpt

Student ID: 991234

Addressing Bus: 6

Entries: 4

Associativity: 2

Indexing bits count: 2

Indexing bits: 2 3

Total cache miss: 4

.benchmark testcase1

000000 miss

000100 miss

001000 miss

000000 hit

001011 miss

000000 hit

001011 hit

.end

Please note that, indexing bits should be printed from MSB to LSB (from small to big).

Grading

- Oral report　–　You should describe the flow of your algorithm clearly.

- Output file format　–　The output file should be generated in output file format.

- Correctness　–　The number of indexing bits and the count of cache miss should be correct.

- Cache miss count　–　The count of cache miss is the main performance criterion of the cache system.

- Run Time　–　Your program should not run over a minute.

- Demonstration　–　You have to daemon your code to TAs.

- Creativity　–　If your new idea is good, than you get higher grade.

References

[1] Tony Givargis "Zero Cost Indexing for Improved Processor Cache Performance",*ACM Transaction on Design Automation of Electronic Systems, Vol. 11, No. 1, pp.3-25, 2006.*