FUDAN UNIVERSITY, SOFTWARE SCHOOL

Design Document of Architecture Project 2

Wang Xin
10302010023

June 5, 2013

1 PREPARATION WORK

This project is based on the five-stage pipeline processor we designed in Project1. However, the instruction set implemented previously is not sufficient to support the new program **mmm**, which is a sparse matrix multiplication. Hence the first task is to implement these instructions, including addi, multu, mflo, beq, etc.
The implementation of multu is a little different from others. According to the definition of SimpleScalarToolSet, multu needs two extra registers: HI and LO to store the higher and lower bits of the result. Thus we need to add to more registers in the processor, but only the LO would be copied to RegisterRd in the following instruction mflo.

2 STRUCTURE OF CACHE BLOCK

According to the requirements, the cache we need to simulate is a four-way set-associative unified cache. We need to first analyze the structure of each cache line.

2.1 STRUCTURE OF CACHE LINE

The cache line is the basic unit of the cache organization.

1. **Valid Bit**: A bit to indicate whether the data stored in this line is valid.
2. **Dirty Bit**: Since we need to simulate the write-back policy, the dirty bit is set to indicate if the cache line should be written back to memory when replaced.

3. **Tag**: A field used to compare with the value of the tag field of the cache. In this project, there are totally $16 = 2^4$ sets. The block size is $4 = 2^2$ and two bits for the byte part of the address. Therefore, the length of the tag filed should be 32-4-2-2=24.

4. **ref_count**: Since we need to implement a FIFO replacement algorithm, we need a field to record the cycles since the cache line was replaced in. This field will increase by one every time when the set was accessed. The cache line with largest ref_count will be replaced out when a new line is to be written in a full set.

The structure of cache line is as follows:

Listing 1: structure of cache line

```
struct cache_line{
    unsigned int valid : 1;
    unsigned int dirty : 1;
    unsigned int tag : 24;
    /*used to implement the FIFO queue*/
    unsigned int ref_count ;
    unsigned int data[4];
};
```

2.2 Structure of Cache Set and Cache Block

A cache set consists of four cache lines and the total cache block contains 16 cache sets.

Listing 2: structure of cache sets and blocks

```
struct cache_set{
    struct cache_line lines[WAYS];
};

struct cache_block{
    struct cache_set sets[SETS];
};
```

3 Cache Read and Cache Write

3.1 Cache Read

1. **Unified Cache**: It is a unified cache including I-cache, is used to provide instructions for execution in instruction fetch stage and Data cache, or D-cache, which is used to provide data in memory access stage. Hence we need to simulate the cache read in both do_if and do_mem stages (LW instruction).

2. **Cache Hit and Miss**: LW instruction will invoke cache read operation. The processor will get the index and tag by the given address. Then use the index to get the corresponding cache set. First of all, all four cache lines in the referred cache set will have

their ref_count increased to represent that it has been replaced in this cache line for one more cycle. Then, if a valid line has the same tag with the address, a cache hit will happen and it will take 1 cycle latency. Otherwise, it will cause a cache miss and take 10 cycles latency.

3. **Cache Write Back**: When cache misses, the processor will check the set to find an empty line(valid = 0). If not found, it will cause a cache replacement. The cache line with largest ref_count will be replaced and the ref_count will be reset to zero. If the replaced line has dirty bit as 1, it will cause a write back.

3.2 CACHE WRITE

Since the cache takes write-back policy, the cache write-operation will firstly cause a cache-read operation initially to get the index and the ways of the writing target if it is already in the cache. If the target line was not in the cache, it will similarly be replaced in cache block. The dirty bit of the written cache line will be set to 1 ultimately.