EEC 170

Assignment 4 – Chapter 5

1. **(3 pt)** How many words are in a direct-mapped cache with 32-bit addresses, 22-bit tags, 5-bit block offsets and a 2-bit byte offsets.

**Note that a direct-mapped cache is effectively a 1-way set associative cache. This means that it will have 1 block per set. The number of index bits is not provided but with knowledge of the size of the address it can be calculated.**

**\*Also, could have solved by noting that the index and block offset bits together specify a word. They combine to make 8 bits, which provides space for words.**

1. **(10 pts)** The following address format is implemented in a 2-way set associative cache with 32-bit addresses and a true LRU policy:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tag** | **Index** | **Block Offset** | **Byte Offset** |
| 31-5 | 4-3 | 2 | 1-0 |

* 1. **(1 pt)** How many words are in each block?

**See figure 5.12 to see a cache with the block offset hardware included. Block offsets select a word within a block. With a 1-bit block offset there can only be 2 words to select from in each block.**

* 1. **(2 pt)** How many words can be present in each set? How many total words can there be there in this cache?

**There are 2 blocks in each set, shown above to be 2 words each. The index bits select the set, and we have 2 bits, so we have = 4 sets.**

* 1. **(6 pts)** The following table represents a series of cache references. Complete the missing entries of the table.  
     1. In the Way 0 and Way 1 columns, show which tags are present in the format T(x)=tag, where “x” is the index and “tag” is the tag value.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Address | Binary  Address | Tag | Index | Block  Offset | Hit/Miss | Way 0 | Way 1 |
| 0x8C | 100\_01\_1\_00 | 4 | 1 | 1 | M | T(1)=4 |  |
| 0x58 | 10\_11\_0\_00 | 2 | 3 | 0 | M | T(1)=4  T(3)=2 |  |
| 0x8A | 100\_01\_0\_10 | 4 | 1 | 0 | H | T(1)=4 T(3)=2 |  |
| 0xBC | 101\_11\_1\_00 | 5 | 3 | 1 | M | T(1)=4 T(3)=2 | T(3)=5 |
| 0xF4 | 111\_10\_1\_00 | 7 | 2 | 1 | M | T(1)=4 T(3)=2  T(2)=7 | T(3)=5 |

**Note that there was no replacement needed. If, for example, there was another T(3) item, the least recently used policy would replace the T(3) with the least recent access, which would be the T(3)=2 block.**

1. **(10 pts)** The program described below runs on a multiple issue processor with a 3-level CPU cache, a 4 GHz clock frequency, and the following performance metrics:

|  |  |  |  |
| --- | --- | --- | --- |
| Level | Miss Penalty R/W | Data Miss Rate | Instruction Miss Rate |
| L1 cache | 2 | 5% | 2% |
| L2 cache |  | 2% | 1.3% |
| Main |  | .3% | .1% |

|  |  |  |
| --- | --- | --- |
| Instruction Class | Instruction Count | IPC |
| Branch/Jump |  | 1.1 |
| Load/Store |  | .9 |
| Arithmetic |  | 1.5 |
| Float Add |  | .25 |

**\* Note IPC not CPI is provided due to multiple issue**

* 1. **(2 pts)** What is the base CPU time of the program?   
     **Hint:** This is the standard ET neglecting the memory-stall clock cycles.  
       
     **Standard ET problem, only using IPC instead of CPI, because CPI < 1 in some cases:**

* 1. **(6 pts)** What is the amount of time consumed by memory-stall clock cycles? Assume equal read/write miss penalty and negligible write buffer stalls.

**See Section 5.4 “Performance of Multilevel Caches”, and Chapter 5 lecture slides 36, 37, 49, & 50.   
  
Number of stalls (miss cycles) per instruction can be treated as a portion of CPI:**

**To get this value in terms of time either multiply this rate by the clock period (as shown below) or you may have multiplied the final solution:**

**Now, we can use this equation to get the memory-stall time (MST):**

**Memory stall cycles/times are always subdivided into Data and Instruction, because the data memory and instruction memory will have different miss rates. The instruction memory is typically a lower miss rate than the data memory. Only instructions that query the data memory are included in the data MST. All instructions come from instruction memory, so all instructions are included in the instruction MST.**

**\*A number of alternative methods to solve this problem exist. You could directly find the MST of each instruction, and sum those together (note that load/store would have both Data and Instruction MST components). You may also have found the memory-stall clock cycles and waited until the end to multiply by to convert it to time.**

* 1. **(2 pts)** What is the total CPU time of the program? What percentage of CPU time was taken by memory stall cycles?  
       
     **From the same example as part b, the total CPI is based on the base CPI plus the memory-stall clock cycles per instruction. This implies that ET is based on the sum of the base ET and the MST.**