Part 1

**4.4 Problems in this exercise assume that logic blocks needed to implement a processor’s datapath have the following latencies:**

A screenshot of a cell phone

Description automatically generated

**4.4.1 If the only thing we need to do in a processor is fetch consecutive instructions (Figure 4.6), what would the cycle time be?**

Solution:

A screenshot of a social media post

Description automatically generated

Fetch consecutive instruction:

IF:

-> I-Mem

-> Add (PC = PC + 4)

Cycle Time = max(200, 70) = 200 ps

**4.4.2 Consider a datapath similar to the one in Figure 4.11, but for a processor that only has one type of instruction: unconditional PC-relative branch. What would the cycle time be for this datapath?**

Solution:

A close up of text on a white background

Description automatically generated

Unconditional PC-relative Branch:

IF:

-> I-Mem

-> Add(PC = PC + 4)

ID:

-> Sign-Extend

EX:

-> Shift-Left-2 -> Add(PC = PC + offset) -> Mux

Cycle Time = max(200, 70) + 15 + (10 + 70 + 20)

= 200 + 15 + 10 + 70 + 20

= 315 ps

**4.4.3 Repeat 4.4.2, but this time we need to support only conditional PC-relative branches.**

Solution:

Conditional PC-relative Branch:

IF:

-> I-Mem

-> Add(PC = PC + 4)

ID:

-> Regs

-> Sign-Extend

EX:

-> Mux -> ALU

-> Mux

-> Shift-Left-2 -> Add(PC = PC + offset)

Cycle Time = max(200, 70) + max(90, 15) + (max(20 + 90, 10 + 70) + 20)

= 200 + 90 + 20 + 90 + 20

= 420 ps

**The remaining three problems in this exercise refer to the datapath element Shift-left-2:**

**4.4.4 Which kinds of instructions require this resource?**

Solution:

1) Load/Store instructions

2) Branch Instructions

3) Jump Instructions (if taken into consideration) (transfer 26-bit to 32-bit address, but maybe another Shift-Left-2 not shown in Figure 4.11)

**4.4.5 For which kinds of instructions (if any) is this resource on the critical path?**

Solution:

Critical path: Longest delay path between state elements in the circuit. (online explanation)

The Registers take more time than Shift-left-2, so only instructions do not use the Registers count.

1) Unconditional PC-relative Branch Instructions

2) Jump Instructions (if taken into consideration)

**4.4.6 Assuming that we only support beq and add instructions, discuss how changes in the given latency of this resource affect the cycle time of the processor. Assume that the latencies of other resources do not change.**

Solution:

beq:

IF:

-> I-Mem

-> Add(PC = PC + 4)

ID:

-> Regs

-> Sign-Extend

EX:

-> Mux -> ALU

-> Mux

-> Shift-Left-2 -> Add(PC = PC + offset)

Cycle Time = max(200, 70) + max(90, 15) + (max(20 + 90, TShift-Left-2 + 70) + 20)

= 200 + 90 + max(110, TShift-Left-2 + 70) + 20 ps

= max(420, 380 + TShift-Left-2) ps

add:

IF:

-> I-Mem

-> Add(PC = PC + 4)

ID:

-> Regs

-> Sign-Extend

EX:

-> Mux -> ALU

WB:

-> Mux -> Regs

Cycle Time = max(200, 70) + max(90, 15) + (20 + 90) + (20 + 90)

= 200 + 90 + 110 + 110

= 510 ps

For the sake of convenience, I will use LSL2 to refer to the latency of Shift-Left-2 later.

When LSL2 is less than or equal to 40 ps, the cycle time is constant . The cycle time of beq is 420 ps, while the one of add is 510 ps. 90 ps extra time for an extra step to write back to Registers.

When LSL2 is more than 40, the result of max(110. TShift-Left-2) is changed. This time, the cycle time of beq is affected by LSL2, but there is nothing related to the one of add.

So, the latency of Shift-Left-2 would affect the cycle time of the whole processor if and only if the latency of Shift-Left-2 is more than 40 ps which will change the time cost in ID. I will not consider the more complicated pipeline included situation for now.

**4.5 For the problems in this exercise, assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:**

A screenshot of a cell phone

Description automatically generated

**4.5.1 In what fraction of all cycles is the data memory used?**

Solution:

Data memory is only used in sw and lw.

25% + 10% = 35%

**4.5.2 In what fraction of all cycles is the input of the sign-extend circuit needed? What is this circuit doing in cycles in which its input is not needed?**

Solution:

I-format instructions use sign-extend circuit: addi, beq, sw and lw.

20% + 25% + 25% + 10% = 80%

Part 2

**5.2 Caches are important to providing a high-performance memory hierarchy to processors. Below is a list of 32-bit memory address references, given as word addresses.**

**3, 180, 43, 2, 191, 88, 190, 14, 181, 44, 186, 253**

**5.2.1 For each of these references, identify the binary address, the tag, and the index given a direct-mapped cache with 16 one-word blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.**

Solution:

Word address << 2 = Memory address

One-word block -> 0-bit offset

16 blocks -> 4-bit Index

|  |  |  |  |
| --- | --- | --- | --- |
| Tag | Index | Block Offset | Byte Offset |
| 26-bit | 4-bit | 0-bit | 2-bit |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index (4-bit) | Valid (1-bit) | Tag (26-bit) | Referenced (Word Address) | Mem[] |
| 0000 | 0 |  |  |  |
| 0001 | 0 |  |  |  |
| 0010 | 1 | 00 0000 0000 0000 0000 0000 0000 | 2 | 2 |
| 0011 | 1 | 00 0000 0000 0000 0000 0000 0000 | 3 | 3 |
| 0100 | 1 | 00 0000 0000 0000 0000 0000 1011 | 180 | 180 |
| 0101 | 1 | 00 0000 0000 0000 0000 0000 1011 | 181 | 181 |
| 0110 | 0 |  |  |  |
| 0111 | 0 |  |  |  |
| 1000 | 1 | 00 0000 0000 0000 0000 0000 0101 | 88 | 88 |
| 1001 | 0 |  |  |  |
| 1010 | 1 | 00 0000 0000 0000 0000 0000 1011 | 186 | 186 |
| 1011 | 1 | 00 0000 0000 0000 0000 0000 0010 | 43 | 43 |
| 1100 | 1 | 00 0000 0000 0000 0000 0000 0010 | 44 | 44 |
| 1101 | 1 | 00 0000 0000 0000 0000 0000 1111 | 253 | 253 |
| 1110 | 1 | 00 0000 0000 0000 0000 0000 0000 | 14 | 14 |
| 1111 | 1 | 00 0000 0000 0000 0000 0000 1011 | 191 | 191 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Word Addr | Binary Address (32-bit) | Index  (4-bit) | BO  (0-bit) | Tag (26-bit) | Hit/  Miss | Ref |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 0011 |  | 00 0000 0000 0000 0000 0000 0000 | Miss |  |
| 180 | 0000 0000 0000 0000 0000 0010 1101 0000 | 0100 |  | 00 0000 0000 0000 0000 0000 1011 | Miss |  |
| 43 | 0000 0000 0000 0000 0000 0000 1010 1100 | 1011 |  | 00 0000 0000 0000 0000 0000 0010 | Miss |  |
| 2 | 0000 0000 0000 0000 0000 0000 0000 1000 | 0010 |  | 00 0000 0000 0000 0000 0000 0000 | Miss |  |
| 191 | 0000 0000 0000 0000 0000 0010 1111 1100 | 1111 |  | 00 0000 0000 0000 0000 0000 1011 | Miss |  |
| 88 | 0000 0000 0000 0000 0000 0001 0110 0000 | 1000 |  | 00 0000 0000 0000 0000 0000 0101 | Miss |  |
| 190 | 0000 0000 0000 0000 0000 0010 1111 1000 | 1110 |  | 00 0000 0000 0000 0000 0000 1011 | Miss |  |
| 14 | 0000 0000 0000 0000 0000 0000 0011 1000 | 1110 |  | 00 0000 0000 0000 0000 0000 0000 | Miss |  |
| 181 | 0000 0000 0000 0000 0000 0010 1101 0100 | 0101 |  | 00 0000 0000 0000 0000 0000 1011 | Miss |  |
| 44 | 0000 0000 0000 0000 0000 0000 1011 0000 | 1100 |  | 00 0000 0000 0000 0000 0000 0010 | Miss |  |
| 186 | 0000 0000 0000 0000 0000 0010 1110 1000 | 1010 |  | 00 0000 0000 0000 0000 0000 1011 | Miss |  |
| 253 | 0000 0000 0000 0000 0000 0011 1111 0100 | 1101 |  | 00 0000 0000 0000 0000 0000 1111 | Miss |  |

**5.2.2 For each of these references, identify the binary address, the tag, and the index given a direct-mapped cache with two-word blocks and a total size of 8 blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.**

Solution:

Two-word block -> 1-bit offset

8 blocks -> 3-bit Index

|  |  |  |  |
| --- | --- | --- | --- |
| Tag | Index | Block Offset | Byte Offset |
| 26-bit | 3-bit | 1-bit | 2-bit |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index (3-bit) | Valid (1-bit) | Tag (26-bit) | Referenced (Word Address) | Mem[] |
| 000 | 0 |  |  |  |
| 001 | 1 | 00 0000 0000 0000 0000 0000 0011 | 3 | 2 |
| 010 | 1 | 00 0000 0000 0000 0000 0000 1011 | 180 | 180 |
| 011 | 0 |  |  |  |
| 100 | 1 | 00 0000 0000 0000 0000 0000 0101 | 88 | 88 |
| 101 | 1 | 00 0000 0000 0000 0000 0000 1011 | 186 | 186 |
| 110 | 1 | 00 0000 0000 0000 0000 0000 1111 | 253 | 252 |
| 111 | 1 | 00 0000 0000 0000 0000 0000 0000 | 14 | 14 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Word Addr | Binary Address (32-bit) | Index  (3-bit) | BO  (1-bit) | Tag (26-bit) | Hit/  Miss | Ref |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 001 | 1 | 00 0000 0000 0000 0000 0000 0000 | Miss |  |
| 180 | 0000 0000 0000 0000 0000 0010 1101 0000 | 010 | 0 | 00 0000 0000 0000 0000 0000 1011 | Miss |  |
| 43 | 0000 0000 0000 0000 0000 0000 1010 1100 | 101 | 1 | 00 0000 0000 0000 0000 0000 0010 | Miss |  |
| 2 | 0000 0000 0000 0000 0000 0000 0000 1000 | 001 | 0 | 00 0000 0000 0000 0000 0000 0000 | Hit | 3 |
| 191 | 0000 0000 0000 0000 0000 0010 1111 1100 | 111 | 1 | 00 0000 0000 0000 0000 0000 1011 | Miss |  |
| 88 | 0000 0000 0000 0000 0000 0001 0110 0000 | 100 | 0 | 00 0000 0000 0000 0000 0000 0101 | Miss |  |
| 190 | 0000 0000 0000 0000 0000 0010 1111 1000 | 111 | 0 | 00 0000 0000 0000 0000 0000 1011 | Hit | 190 |
| 14 | 0000 0000 0000 0000 0000 0000 0011 1000 | 111 | 0 | 00 0000 0000 0000 0000 0000 0000 | Miss |  |
| 181 | 0000 0000 0000 0000 0000 0010 1101 0100 | 010 | 1 | 00 0000 0000 0000 0000 0000 1011 | Hit | 180 |
| 44 | 0000 0000 0000 0000 0000 0000 1011 0000 | 110 | 0 | 00 0000 0000 0000 0000 0000 0010 | Miss |  |
| 186 | 0000 0000 0000 0000 0000 0010 1110 1000 | 101 | 0 | 00 0000 0000 0000 0000 0000 1011 | Miss |  |
| 253 | 0000 0000 0000 0000 0000 0011 1111 0100 | 110 | 1 | 00 0000 0000 0000 0000 0000 1111 | Miss |  |

**5.2.3 You are asked to optimize a cache design for the given references. There are three direct-mapped cache designs possible, all with a total of 8 words of data: C1 has 1-word blocks, C2 has 2-word blocks, and C3 has 4-word blocks. In terms of miss rate, which cache design is the best? If the miss stall time is 25 cycles, and C1 has an access time of 2 cycles, C2 takes 3 cycles, and C3 takes 5 cycles, which is the best cache design?**

Solution:

C1:

1-word block -> 0-bit offset

8/1 blocks -> 3-bit Index

|  |  |  |  |
| --- | --- | --- | --- |
| Tag | Index | Block Offset | Byte Offset |
| 27-bit | 3-bit | 0-bit | 2-bit |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index (3-bit) | Valid (1-bit) | Tag (27-bit) | Referenced (Word Address) | Mem[] |
| 000 | 1 | 000 0000 0000 0000 0000 0000 1011 | 88 | 88 |
| 001 | 0 |  |  |  |
| 010 | 1 | 000 0000 0000 0000 0000 0001 0111 | 186 | 186 |
| 011 | 1 | 000 0000 0000 0000 0000 0000 0101 | 43 | 43 |
| 100 | 1 | 000 0000 0000 0000 0000 0000 0101 | 44 | 44 |
| 101 | 1 | 000 0000 0000 0000 0000 0001 1111 | 253 | 253 |
| 110 | 1 | 000 0000 0000 0000 0000 0000 0001 | 14 | 14 |
| 111 | 1 | 000 0000 0000 0000 0000 0001 0111 | 191 | 191 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Word Addr | Binary Address (32-bit) | Index  (3-bit) | BO  (0-bit) | Tag (27-bit) | Hit/  Miss | Ref |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 011 |  | 000 0000 0000 0000 0000 0000 0000 | Miss |  |
| 180 | 0000 0000 0000 0000 0000 0010 1101 0000 | 100 |  | 000 0000 0000 0000 0000 0001 0110 | Miss |  |
| 43 | 0000 0000 0000 0000 0000 0000 1010 1100 | 011 |  | 000 0000 0000 0000 0000 0000 0101 | Miss |  |
| 2 | 0000 0000 0000 0000 0000 0000 0000 1000 | 010 |  | 000 0000 0000 0000 0000 0000 0000 | Miss |  |
| 191 | 0000 0000 0000 0000 0000 0010 1111 1100 | 111 |  | 000 0000 0000 0000 0000 0001 0111 | Miss |  |
| 88 | 0000 0000 0000 0000 0000 0001 0110 0000 | 000 |  | 000 0000 0000 0000 0000 0000 1011 | Miss |  |
| 190 | 0000 0000 0000 0000 0000 0010 1111 1000 | 110 |  | 000 0000 0000 0000 0000 0001 0111 | Miss |  |
| 14 | 0000 0000 0000 0000 0000 0000 0011 1000 | 110 |  | 000 0000 0000 0000 0000 0000 0001 | Miss |  |
| 181 | 0000 0000 0000 0000 0000 0010 1101 0100 | 101 |  | 000 0000 0000 0000 0000 0001 0110 | Miss |  |
| 44 | 0000 0000 0000 0000 0000 0000 1011 0000 | 100 |  | 000 0000 0000 0000 0000 0000 0101 | Miss |  |
| 186 | 0000 0000 0000 0000 0000 0010 1110 1000 | 010 |  | 000 0000 0000 0000 0000 0001 0111 | Miss |  |
| 253 | 0000 0000 0000 0000 0000 0011 1111 0100 | 101 |  | 000 0000 0000 0000 0000 0001 1111 | Miss |  |

Miss Rate = 12/12 = 100%

C2:

2-word block -> 1-bit offset

8/2 blocks -> 2-bit Index

|  |  |  |  |
| --- | --- | --- | --- |
| Tag | Index | Block Offset | Byte Offset |
| 27-bit | 2-bit | 1-bit | 2-bit |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index (2-bit) | Valid (1-bit) | Tag (27-bit) | Referenced (Word Address) | Mem[] |
| 00 | 1 | 000 0000 0000 0000 0000 0000 1011 | 88 | 88 |
| 01 | 1 | 000 0000 0000 0000 0000 0001 0111 | 186 | 186 |
| 10 | 1 | 000 0000 0000 0000 0000 0001 1111 | 253 | 252 |
| 11 | 1 | 000 0000 0000 0000 0000 0000 0001 | 14 | 14 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Word Addr | Binary Address (32-bit) | Index  (2-bit) | BO  (1-bit) | Tag (27-bit) | Hit/  Miss | Ref |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 01 | 1 | 000 0000 0000 0000 0000 0000 0000 | Miss |  |
| 180 | 0000 0000 0000 0000 0000 0010 1101 0000 | 10 | 0 | 000 0000 0000 0000 0000 0001 0110 | Miss |  |
| 43 | 0000 0000 0000 0000 0000 0000 1010 1100 | 01 | 1 | 000 0000 0000 0000 0000 0000 0101 | Miss |  |
| 2 | 0000 0000 0000 0000 0000 0000 0000 1000 | 01 | 0 | 000 0000 0000 0000 0000 0000 0000 | Miss |  |
| 191 | 0000 0000 0000 0000 0000 0010 1111 1100 | 11 | 1 | 000 0000 0000 0000 0000 0001 0111 | Miss |  |
| 88 | 0000 0000 0000 0000 0000 0001 0110 0000 | 00 | 0 | 000 0000 0000 0000 0000 0000 1011 | Miss |  |
| 190 | 0000 0000 0000 0000 0000 0010 1111 1000 | 11 | 0 | 000 0000 0000 0000 0000 0001 0111 | Hit | 191 |
| 14 | 0000 0000 0000 0000 0000 0000 0011 1000 | 11 | 0 | 000 0000 0000 0000 0000 0000 0001 | Miss |  |
| 181 | 0000 0000 0000 0000 0000 0010 1101 0100 | 10 | 1 | 000 0000 0000 0000 0000 0001 0110 | Hit | 180 |
| 44 | 0000 0000 0000 0000 0000 0000 1011 0000 | 10 | 0 | 000 0000 0000 0000 0000 0000 0101 | Miss |  |
| 186 | 0000 0000 0000 0000 0000 0010 1110 1000 | 01 | 0 | 000 0000 0000 0000 0000 0001 0111 | Miss |  |
| 253 | 0000 0000 0000 0000 0000 0011 1111 0100 | 10 | 1 | 000 0000 0000 0000 0000 0001 1111 | Miss |  |

Miss Rate = 10/12 = 83.3%

C3:

4-word block -> 2-bit offset

8/4 blocks -> 1-bit Index

|  |  |  |  |
| --- | --- | --- | --- |
| Tag | Index | Block Offset | Byte Offset |
| 27-bit | 1-bit | 2-bit | 2-bit |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index (1-bit) | Valid (1-bit) | Tag (27-bit) | Referenced (Word Address) | Mem[] |
| 0 | 1 | 000 0000 0000 0000 0000 0001 0111 | 186 | 184 |
| 1 | 1 | 000 0000 0000 0000 0000 0001 1111 | 253 | 252 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Word Addr | Binary Address (32-bit) | Index  (1-bit) | BO  (2-bit) | Tag (27-bit) | Hit/  Miss | Ref |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 0 | 11 | 000 0000 0000 0000 0000 0000 0000 | Miss |  |
| 180 | 0000 0000 0000 0000 0000 0010 1101 0000 | 1 | 00 | 000 0000 0000 0000 0000 0001 0110 | Miss |  |
| 43 | 0000 0000 0000 0000 0000 0000 1010 1100 | 0 | 11 | 000 0000 0000 0000 0000 0000 0101 | Miss |  |
| 2 | 0000 0000 0000 0000 0000 0000 0000 1000 | 0 | 10 | 000 0000 0000 0000 0000 0000 0000 | Miss |  |
| 191 | 0000 0000 0000 0000 0000 0010 1111 1100 | 1 | 11 | 000 0000 0000 0000 0000 0001 0111 | Miss |  |
| 88 | 0000 0000 0000 0000 0000 0001 0110 0000 | 0 | 00 | 000 0000 0000 0000 0000 0000 1011 | Miss |  |
| 190 | 0000 0000 0000 0000 0000 0010 1111 1000 | 1 | 10 | 000 0000 0000 0000 0000 0001 0111 | Hit | 190 |
| 14 | 0000 0000 0000 0000 0000 0000 0011 1000 | 1 | 10 | 000 0000 0000 0000 0000 0000 0001 | Miss |  |
| 181 | 0000 0000 0000 0000 0000 0010 1101 0100 | 1 | 01 | 000 0000 0000 0000 0000 0001 0110 | Miss |  |
| 44 | 0000 0000 0000 0000 0000 0000 1011 0000 | 1 | 00 | 000 0000 0000 0000 0000 0000 0101 | Miss |  |
| 186 | 0000 0000 0000 0000 0000 0010 1110 1000 | 0 | 10 | 000 0000 0000 0000 0000 0001 0111 | Miss |  |
| 253 | 0000 0000 0000 0000 0000 0011 1111 0100 | 1 | 01 | 000 0000 0000 0000 0000 0001 1111 | Miss |  |

Miss Rate = 11/12 = 91.7%

Miss Rate:

C1: 12/12 = 100%

C2: 10/12 = 83.3% (C2 is best in terms of miss rate)

C3: 11/12 = 91.7%

Cycle Time: (Access Time + Miss Stall Time)

C1: 12 \* 2 + 12 \* 25 = 324 cycles

C2: 12 \* 3 + 10 \* 25 = 286 cycles (C2 is best in terms of cycle time)

C3: 12 \* 5 + 11 \* 25 = 335 cycles

**There are many different design parameters that are important to a cache’s overall performance. Below are listed parameters for different direct-mapped cache designs.**

**Cache Data Size: 32 KiB**

**Cache Block Size: 2 words**

**Cache Access Time: 1 cycle**

**5.2.4 Calculate the total number of bits required for the cache listed above, assuming a 32-bit address. Given that total size, find the total size of the closest direct-mapped cache with 16-word blocks of equal size or greater. Explain why the second cache, despite its larger data size, might provide slower performance than the first cache.**

Solution:

1 KiB = 2^13 bits

2-word block -> 1-bit offset

32 KiB / (2 words) = 2^18 bits / 2^6 bits = 2^12 blocks -> 12-bit Index

|  |  |  |  |
| --- | --- | --- | --- |
| Tag | Index | Block Offset | Byte Offset |
| 17-bit | 12-bit | 1-bit | 2-bit |

Cache Size = block number \* (block size + tag size + valid field size)

= 2^12 \* (2^1 \* 32 + 17 + 1)

= 335872 bits

16-word block -> 4-bit offset

|  |  |  |  |
| --- | --- | --- | --- |
| Tag | Index | Block Offset | Byte Offset |
| (26 - n)-bit | n-bit | 4-bit | 2-bit |

Cache Size = block number \* (block size + tag size + valid field size)

= 2^n \* (2^4 \* 32 + (26 - n) + 1) >= 335872 bits

n = 8, Left = 135936 bits

n = 9, Left = 271360 bits

n = 10, Left = 541696 bits

n = 11, Left = 1081344 bits

-> nmin = 10

Data Size = 2^10 \* 2^4 \* 2^5 = 2^19 bits = 2^6 KiB = 64 KiB > 32 KiB (Larger)

For a 16-word blocks cache, we need more access time to get the Data, although we can reduce the miss rate. If the weight that access time increases is greater than the miss stall time reduces, than it will performance slow.

**5.2.5 Generate a series of read requests that have a lower miss rate on a 2 KiB 2-way set associative cache than the cache listed above. Identify one possible solution that would make the cache listed have an equal or lower miss rate than the 2 KiB cache. Discuss the advantages and disadvantages of such a solution.**

Solution:

2 KiB 2-way set associative cache

1-word -> 0-bit offset

2KiB / (1 word) = 2^14 / 2^5 = 2^9 blocks -> 9-bit Index

|  |  |  |  |
| --- | --- | --- | --- |
| Tag | Index | Block Offset | Byte Offset |
| 21-bit | 9-bit | 0-bit | 2-bit |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Word Addr | Binary Address (32-bit) | Index  (9-bit) | Way | Tag (21-bit) | Hit/  Miss | Ref |
| Series 1 | | | | | | |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 0 0000 0011 | 0 | 0 0000 0000 0000 0000 0000 | Miss |  |
| 8194 | 0000 0000 0000 0000 0000 1000 0000 1000 | 0 0000 0010 | 1 | 0 0000 0000 0000 0000 0001 | Miss |  |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 0 0000 0011 | 0 | 0 0000 0000 0000 0000 0000 | Hit | 3 |
| 8194 | 0000 0000 0000 0000 0000 1000 0000 1000 | 0 0000 0010 | 1 | 0 0000 0000 0000 0000 0001 | Hit | 8194 |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 0 0000 0011 | 0 | 0 0000 0000 0000 0000 0000 | Hit | 3 |
| 8194 | 0000 0000 0000 0000 0000 1000 0000 1000 | 0 0000 0010 | 1 | 0 0000 0000 0000 0000 0001 | Hit | 8194 |
| Series 2 | | | | | | |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 0 0000 0011 | 0 | 0 0000 0000 0000 0000 0000 | Miss |  |
| 2 | 0000 0000 0000 0000 0000 0000 0000 1000 | 0 0000 0010 | 1 | 0 0000 0000 0000 0000 0000 | Miss |  |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 0 0000 0011 | 0 | 0 0000 0000 0000 0000 0000 | Hit | 3 |
| 2 | 0000 0000 0000 0000 0000 0000 0000 1000 | 0 0000 0010 | 1 | 0 0000 0000 0000 0000 0000 | Hit | 2 |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 0 0000 0011 | 0 | 0 0000 0000 0000 0000 0000 | Hit | 3 |
| 2 | 0000 0000 0000 0000 0000 0000 0000 1000 | 0 0000 0010 | 1 | 0 0000 0000 0000 0000 0000 | Hit | 2 |

32 KiB 2 words direct-mapped cache

2-word block -> 1-bit offset

32 KiB / (2 words) = 2^18 bits / 2^6 bits = 2^12 blocks -> 12-bit Index

|  |  |  |  |
| --- | --- | --- | --- |
| Tag | Index | Block Offset | Byte Offset |
| 17-bit | 12-bit | 1-bit | 2-bit |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Word Addr | Binary Address (32-bit) | Index  (12-bit) | BO  (1-bit) | Tag (17-bit) | Hit/  Miss | Ref |
| Series 1 | | | | | | |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 0000 0000 0001 | 1 | 0 0000 0000 0000 0000 | Miss |  |
| 8194 | 0000 0000 0000 0000 1000 0000 0000 1000 | 0000 0000 0001 | 0 | 0 0000 0000 0000 0001 | Miss |  |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 0000 0000 0001 | 1 | 0 0000 0000 0000 0000 | Miss |  |
| 8194 | 0000 0000 0000 0000 1000 0000 0000 1000 | 0000 0000 0001 | 0 | 0 0000 0000 0000 0001 | Miss |  |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 0000 0000 0001 | 1 | 0 0000 0000 0000 0000 | Miss |  |
| 8194 | 0000 0000 0000 0000 1000 1000 0000 1000 | 0000 0000 0001 | 0 | 0 0000 0000 0000 0001 | Miss |  |
| Series 2 | | | | | | |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 0000 0000 0001 | 1 | 0 0000 0000 0000 0000 | Miss |  |
| 2 | 0000 0000 0000 0000 0000 0000 0000 1000 | 0000 0000 0001 | 0 | 0 0000 0000 0000 0000 | Hit | 3 |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 0000 0000 0001 | 1 | 0 0000 0000 0000 0000 | Hit | 3 |
| 2 | 0000 0000 0000 0000 0000 0000 0000 1000 | 0000 0000 0001 | 0 | 0 0000 0000 0000 0000 | Hit | 3 |
| 3 | 0000 0000 0000 0000 0000 0000 0000 1100 | 0000 0000 0001 | 1 | 0 0000 0000 0000 0000 | Hit | 3 |
| 2 | 0000 0000 0000 0000 0000 0000 0000 1000 | 0000 0000 0001 | 0 | 0 0000 0000 0000 0000 | Hit | 3 |

|  |  |  |
| --- | --- | --- |
| Miss Rate | Series 1 | Series 2 |
| 2 KiB 2-way set associative cache | 2/6 = 33% | 6/6 = 100% |
| 32 KiB 2 words direct-mapped cache | 2/6 = 33% | 1/6 = 16.7% |

N-words direct-mapped caches are good at handling requests which address only change in a small address range. While n-words direct-mapped caches are really good at dealing with several small amount but called at high frequencies ones.

**5.2.6 The formula shown in Section 5.3 shows the typical method to index a direct-mapped cache, specifically (Block address) modulo (Number of blocks in the cache). Assuming a 32-bit address and 1024 blocks in the cache, consider a different indexing function, specifically (Block address[31:27] XOR Block address[26:22]). Is it possible to use this to index a direct-mapped cache? If so, explain why and discuss any changes that might need to be made to the cache. If it is not possible, explain why.**

Solution:

I think it is possible but unnecessary. Because XOR is an irreversible operation, you must record additional information and pay an extra price for the real address. So, yes, for example, just add a 3-bit Mem Bit to record the Block address[31:27], and then we are done. In some ways, it saves space in the cache because it is folded from 2^6 to 2^3. But it costs a lot on obtaining the address.

**5.3 For a direct-mapped cache design with a 32-bit address, the following bits of the address are used to access the cache.**

A screenshot of a cell phone

Description automatically generated

**5.3.1 What is the cache block size (in words)?**

Solution:

Block Offset Bits= Offset – Byte Offset = (4 – 0 + 1) – 2 = 3 bits

Cache Block Size = 2^3 = 8-word

**5.3.2 How many entries does the cache have?**

Solution:

Index Bits= 9 – 5 + 1 = 5 bits

Number of Entries = 2^5 = 32

**5.3.3 What is the ratio between total bits required for such a cache implementation over the data storage bits?**

Solution:

Tag Bits = 31 -10 + 1 = 22 bits

Cache Size = block number \* (block size + tag size + valid field size)

= 2^5 \* (2^3 \* 32 + 22 + 1)

= 8928 bits

Cache Data Size = block number \* block size

= 2^5 \* 2^3 \* 32

= 8192 bits

Cache Size / Cache Data Size = 8928 / 8192 = 1.09

/\*

Simple Solution:

Cache Size / Cache Data Size = (block size + tag size + valid field size) / block size

= (2^3 \* 32 + 22 + 1) / (2^3 \* 32)

= 279 / 258

= 1.09

\*/

**Starting from power on, the following byte-addressed cache references are recorded.**

A screenshot of a cell phone

Description automatically generated

**5.3.4 How many blocks are replaced?**

Solution:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Byte-Addr | Binary Address (32-bit) | Index  (5-bit) | BO  (3-bit) | Tag (22-bit) | Hit/  Miss | Ref | Replaced? |
| 0 | 0000 0000 0000 0000 0000 0000 0000 0000 | 0 0000 | 000 | 00 0000 0000 0000 0000 0000 | Miss |  | N |
| 4 | 0000 0000 0000 0000 0000 0000 0000 0100 | 0 0000 | 001 | 00 0000 0000 0000 0000 0000 | Hit | 0 | N |
| 16 | 0000 0000 0000 0000 0000 0000 0001 0000 | 0 0000 | 100 | 00 0000 0000 0000 0000 0000 | Hit | 0 | N |
| 132 | 0000 0000 0000 0000 0000 0000 1000 0100 | 0 0100 | 001 | 00 0000 0000 0000 0000 0000 | Miss |  | N |
| 232 | 0000 0000 0000 0000 0000 0000 1110 1000 | 0 0111 | 010 | 00 0000 0000 0000 0000 0000 | Miss |  | N |
| 160 | 0000 0000 0000 0000 0000 0000 1010 0000 | 0 0101 | 000 | 00 0000 0000 0000 0000 0000 | Miss |  | N |
| 1024 | 0000 0000 0000 0000 0000 0100 0000 0000 | 0 0000 | 000 | 00 0000 0000 0000 0000 0001 | Miss |  | Y |
| 30 | 0000 0000 0000 0000 0000 0000 0001 1110 | 0 0000 | 111 | 00 0000 0000 0000 0000 0000 | Miss |  | Y |
| 140 | 0000 0000 0000 0000 0000 0000 1000 1100 | 0 0100 | 011 | 00 0000 0000 0000 0000 0000 | Hit | 132 | N |
| 3100 | 0000 0000 0000 0000 0000 1100 0001 1100 | 0 0000 | 111 | 00 0000 0000 0000 0000 0011 | Miss |  | Y |
| 180 | 0000 0000 0000 0000 0000 0000 1011 0100 | 0 0101 | 101 | 00 0000 0000 0000 0000 0000 | Hit | 160 | N |
| 2180 | 0000 0000 0000 0000 0000 1000 1000 0100 | 0 0100 | 001 | 00 0000 0000 0000 0000 0010 | Miss |  | Y |

4 replacements happened on 2 different block entries(00000, 00100).

**5.3.5 What is the hit ratio?**

Solution:

Hit Ratio = 4 / 12 = 33%

**5.3.6 List the final state of the cache, with each valid entry represented as a record of <index, tag, data>.**

Solution:

Omit the invalid entries:

<00000, 00 0000 0000 0000 0000 0011, mem[3072]>

<00100, 00 0000 0000 0000 0000 0010, mem[2176]>

<00101, 00 0000 0000 0000 0000 0000, mem[160]>

<00111, 00 0000 0000 0000 0000 0000, mem[224]>