CSU-34021 Computer Architecture II

Name : Abhay Singh Khanka

Q1

1. We have block size = 2

Number of cache lines = cache size/ block size = 16/2 = 8

Number of cache lines in a set = 2

Number of sets = 8/2 = 4 (so 2 bits for set)

Then tag is = 6 – (1+2) = 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Decimal | Binary | Offset | Set | Tag | Status |
| 15 | 00 1111 | 1 | 11 | 001 | MISS |
| 17 | 01 0001 | 1 | 00 | 010 | MISS |
| 1 | 00 0001 | 1 | 00 | 000 | MISS |
| 17 | 01 0001 | 1 | 00 | 010 | HIT |
| 15 | 00 1111 | 1 | 11 | 001 | HIT |
| 1 | 00 0001 | 1 | 00 | 000 | HIT |
| 0 | 00 0000 | 0 | 00 | 000 | HIT |
| 9 | 00 1001 | 1 | 00 | 001 | MISS |
| 15 | 00 1111 | 1 | 11 | 001 | HIT |
| 8 | 00 1000 | 0 | 00 | 001 | HIT |
| 7 | 00 0111 | 1 | 11 | 000 | MISS |
| 9 | 00 1001 | 1 | 00 | 001 | HIT |
| 15 | 00 1111 | 1 | 11 | 001 | HIT |

HITS = 8

MISSES = 5

b) Hit rate = 8/13 = 0.62 = 62%

Miss rate = 5/13 = 0.38 = 38%

c) We have,

16Kb cache size

4-way set associative cache (let’s call this x=4)

Line size = 4\*32 bits = 16 bytes

Offset bits = 2 for 4 words

Set size = no. of lines/x

No. of lines = cache size/line size = 2^10

Set size = 2^10/4 = 256

It is divided in the following way :

A white sheet of paper with writing on it

Description automatically generated with low confidence

d) Converting the given hexadecimal number in binary we get :

*1010101111001101111010* **00111110** 00

Where Set = numbers in **Bold**

Offset = numbers underlined

Tag = numbers in *Italic*

And thus by converting the set into decimal we get the memory location to be 62.

e) A structural hazard occurs when two or more instructions that are already in the pipeline need the same resource. This results in instructions needing to be executed in a series rather than parallel for a portion of the pipeline. Structural hazards are also referred to as resource hazards.

By having a state of the art processor, we have access to more resources such as having multiple ports to main memory and multiple arithmetic logic units (ALU). These extra resources help control and prevent structural hazards and as a result give us a better performance.

f) Sometimes it becomes necessary to introduce a no operation (NOP) instruction. Taking the following instructions as an example :

ADD R1, R1, R3

LD R2, R1, R0

NOP

BEQZ R2, 08

A NOP is needed because there is a data dependency between the load instruction and the instruction following the load instruction. Thus the pipeline would be stalled for 1 ns.

Instead of having to put a NOP we can put another useful instruction. For example

BEQZ R2, 24

ST R2, R0, 00

The store instruction is in the delay slot of the BEQZ instruction. Thus instead of adding a NOP instruction which would increase the time of the program, we add another useful instruction there which keeps the same time and increases number instructions executed.

g) i) Instruction executed = 14

Ticks = 22

These two numbers are different due to stalls taking place in the program. For example a

stall of 1ns takes place between the instructions LD and SLLI due to data dependency.

ii) A total of 4 stalls are present in the program. The first and last stalls are introduced to avoid data hazards, while the rest are inserted to avoid control hazards.

Stalls are introduced to delay execution of an instruction to avoid an hazard, especially when working with load and store instructions which are present in the beginning and at end of the code. Similarly stalls are also introduced to avoid control hazards which occur due to control dependencies in our program.

iii) First of all we would put a NOP after the instruction LD R3,R0,02. This is done in order to avoid data hazard as the next instruction involves the use of R3.

Q2 a) It is given that,

CPI = 1

Clock rate = 5ghz

Main mem access time = 100ns

Miss rate (primary cache) = 2%

Secondary cache access time = 5ns

% of reduction of miss rate to main mem = 0.5%

Miss penalty to main memory = (100\*10^-9) / (1(5000\*10^6)) = 500 clock cycles

Total CPI = base cpi + memory stalls per instruction

= 1 + 2%\*500 = 11

Miss penalty of level two cache = (5\*10^-9) / (1(5000\*10^6)) = 25 clock cycles

CPI for level two cache = base cpi + level 1 memory stalls + level 2 memory stalls

= 1 + (2%\*25) + (0.5%\*500)

= 4

Required ratio = 11/4 = 2.75 times faster

b) First of all we have to establish stack frame using prologue (i.e. push ebp and move ebp, esp).

We then need to push ebx for storing the second argument as edx will be used by the imul instruction.

Finally we dismantle the stack frame by using the epilogue.

The corrected code is given below :

pow: push ebp

mov ebp, esp

push ebx

mov eax, 1

mov eax, [ebp+12]

mov eax. [ebp+8]

L1: imul ebx

loop L1

pop ebx

mov esp, ebp

pop ebp

ret

c) code 1 edx = 1

This is because the JL instruction is a signed instruction so the value of 8000h is out of bounds and thus edx remains 1.

code 2 edx = 1

This is because the JB instruction is an unsigned instruction so when the values are

compared, 7fffh is smaller than 8000h thus the jump takes place and edx remains 1.

d) Byte addressable memory refers to architectures where data can be accessed and addressed in units that are narrower than the bus. An eight-bit processor addresses eight bits, but as this is the full width of the bus, this is regarded as word-addressable.

e) Bits allocated to page1 = 10

Bits allocated to page2 = 10

Bits allocated to offset = 12

Size of page table for each page = 1024\*4 = 4KB

f) Paging based memory allocation divides virtual memory or all processes into equal sized pages and physical memory into fixed size frames. Thus a page has a fixed size, however processes may request more or less space.

For example, the page size is 2048 bytes, a process of 72,766 bytes will need 35 pages plus 1086 bytes. It will be allocated 36 times leading to an internal fragmentation of 2048-1086 = 962 bytes.

External fragmentation occurs when a process, which was allocated in contiguous memory, is unloaded from physical memory, which creates a free space in the memory. If a new process comes which requires more memory than the free space, we will not be able to allocate contiguous memory to that process due to the non-contiguous nature of free memory.

In Paging based memory allocation a process is allowed to be allocated space that are non-contiguous in the physical memory, meanwhile the logical representation of those blocks would be contagious in the virtual memory.

g) The write once protocol uses write-back caches and a write-invalidate protocol. Each cache line is in one of the following states :

INVALID

VALID

RESERVED

DIRTY

Meanwhile, the MESI protocol uses a shared bus signal so cache lines can enter the cache in the shared or exclusive state, which is not possible in write-once. Its cache line are in one the following states :

INVALID

SHARED

EXCLUSIVE

MODIFIED

Also, Write-once write-through cycles are no longer necessary if the state is EXCLUSIVE.