Lund University

Computer Architecture, EITF20

Exercise 2

Mohammad Attari (Masoud)

December 17, 2019

Problem 1

The following code fragment is given:

```
loop: ld    x1, 0(x2)         ;load x1 from address 0+x2
      addi  x1, x1, 1         ;x1=x1+1
      sd    x1, 0(x2)         ;store x1 at address 0+x2
      addi  x2, x2, 4         ;x2=x2+4
      sub   x4, x3, x2        ;x4=x3-x2
      bnez  x4, Loop          ;branch to Loop if x4!= 0
```

Assume that the initial value of x3 is x2+40.

a) Show the timing table for the 1st iteration plus the 1st instruction of the 2nd iteration of this instruction sequence for the 5-stage RISC pipeline without any forwarding hardware, but assuming that a register read and a write in the same clock cycle "forwards" through the register file. Assuming that the branch target and outcome are unknown until the end of the execution stage, calculate how many cycles does this loop take to execute?

b) Repeat the above tasks for the same pipeline with full forwarding hardware. Note that the bypassing happens through the pipeline registers. Assume that the branch is handled by predicting it as not taken. The branch target and outcome are now calculated in the ID stage.

c) This time we adopt a taken prediction scheme, but with the branch target calculation performed in the ID stage (branch condition still tested in the EX stage). Show the timing table and calculate the number of cycles needed to finish the loop.

Problem 2

Consider following part of an assembly language program:

```
1: LW       R5, (R3)          ;Load word R5=Mem(R3)
2: LW       R6, 8(R3)         ;Load word R6=Mem(R3+8)
3: MUL      R5, R5, R6        ;Multiply R5=R5*R6
4: ADD      R3, R3, R2        ;Add R3=R3+R2
5: SW       R5, (R3)          ;Store word Mem(R3)=R5
```

Basic architecture setup: A standard 5-stage pipeline architecture with stalling but no forwarding; One arithmetic functional unit (for both multiplication and addition); All instructions take one cycle in the EXE stage, except MUL and ADD, which take 5 cycles and 2 cycles in the EXE stage respectively (The EXE stage of MUL and ADD is not pipelined); Assume a perfect memory system (all memory accesses are cache hits).

a) Draw an instruction execution timing table (clock number vs instruction number with entries showing the pipeline stage, similar to the table below) for the above code snippet.
b) Now, improve the basic pipeline architecture with 2 arithmetic functional units and the Tomasulo technique with 2 reservation stations for each arithmetic functional unit and enough reservation stations and units for loads/stores. Assume the execution time for each individual instruction remains the same.
c) Repeat the above with scoreboarding hardware.

Problem 3

Which has the lower miss rate: a 16 KiB instruction cache with a 16 KiB data cache or a 32 KiB unified cache? Use the miss rates in the figure below to help calculate the correct answer, assuming 36% of the instructions are data transfer instructions. Assume a hit takes 1 clock cycle and the miss penalty is 200 clock cycles. A load or store hit takes 1 extra clock cycle on a unified cache if there is only one cache port to satisfy two simultaneous requests. The unified cache leads to a structural hazard. What is the average memory access time in each case? Assume write-through caches with a write buffer and ignore stalls due to the write buffer.

Misses per 1000 instructions

Size (KiB)	Instruction cache	Data cache	Unified cache
8	8.16	44.0	63.0
16	3.82	40.9	51.0
32	1.36	38.4	43.3
64	0.61	36.9	39.4
128	0.30	35.3	36.2
256	0.02	32.6	32.9

Problem 4

An in-order CPU running at 2.5 GHz is given. The average CPI=2.3 for all instructions and the average frequency (ratio) of each operation type is shown in the following table. Assume no stalls due to data dependencies and branches.

Instruction mix	
instruction type	**% of all instructions**
load	23.75
store	9.66
uncond branch	5.84
cond branch	18.45
int computation	42.30
fp computation	0.00

Now we add a unified cache to the CPU. The cache has 128 sets and associativity 1. The cache system has a miss-penalty of 50 clock cycles. We are interested to quickly compare improvements (in average) of 3 mutually exclusive enhancements to the CPU.

Enhancements and data regarding our intended application are:

1) Enhancing integer computation performance by a factor 1.8 (assume no effects on miss-ratio and clock cycle time)
2) Decreasing execution time for data access instructions by a factor 2 (assume no effects on miss-ratio and clock cycle time)
3) Changing cache associativity from 128 sets, associativity 1 to 32 sets, associativity 4.

Cache miss-ratio			
No. of sets	**Associativity**		
	1	**2**	**4**
16	0.778109	0.594727	0.303048
32	0.597252	0.312289	0.136385
64	0.397689	0.145433	0.083915
128	0.207784	0.088052	0.051590

Clock cycle time dependence on associativity is:

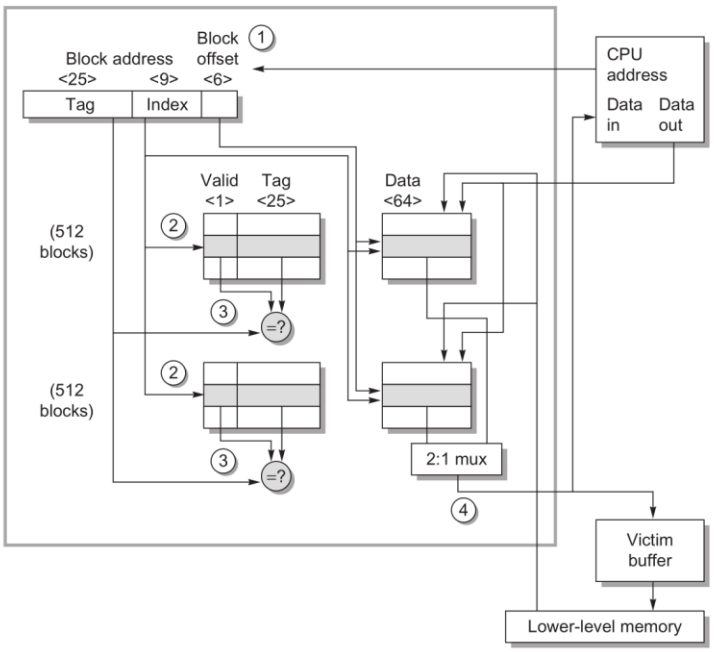
	Associativity		
	1	**2**	**4**
relative clock cycle time	1	1.04	1.1

Which of the 3 enhancements will give the highest speedup? (Show all calculations!)

Problem 5

a) Show the address bit partitioning into fields for a memory system with parameters:
Memory size = 1GB
Cache size = 512 KB
Block size = 32 Bytes
Set associative with 4 blocks per set.

b) List the micro-operations (4 steps in the following picture) done during a successful cache access.



c) In the previous example (b), what would the address breakdown look like if you were to make the cache direct-mapped, but kept the cache size intact? How about a fully associative cache?

d) Now imagine you have a 128B direct-mapped data cache with the block size being 4 bytes. The following code is starts running on the processor and the data cache is clean (that is, empty), but ignore the instruction cache effects. Is increasing the block size (with same cache size) to 8 bytes beneficial for this code? If yes, then can you push the block size to its limit? Motivate your answer with examples (for instance how array placement in memory can affect the performance).

```
      Mov   R10, 32
      Mov   R4, 0
Loop:
      LW    R1, (R5)
      LW    R2, (R6)
      Add   R3, R1, R2
      Add   R4, R3, R4
      Addi  R5, R5, 4
      Addi  R6, R6, 4
      Subi  R10, 1
      Bneq  Loop
```