CS 352 Computer Architecture

Midterm #2 – Prof. Mark April 5, 2005

Name: SAMPLE SOLUTION
(please print)

1.	18 points	_____
2.	24 points	_____
3.	30 points	_____
4.	16 points	_____
5.	12 points	_____
	TOTAL (100 pts)	_____

In recognition of University regulations regarding academic dishonesty, I certify that I have neither given nor received any un-permitted aid on this examination. This exam is open notes (yours and lectures notes and previous HW/exams from this semester), open textbook. Calculators may be used, but only in non-programmable mode. No other materials may be used.

Signed: _____

General Instructions:
If there is an ambiguity in a problem, please either ask the instructor for clarification or clearly state a reasonable assumption that resolves the ambiguity, and proceed with the problem.

Problem #1 – Simple cache behavior (18 points)

Suppose that we have a computer with a 2-way set associative cache that holds a total of four blocks. The block size is one word. We use the usual algorithm for mapping addresses to cache lines for a set associative cache. The replacement policy is least-recently-used (LRU).

a) (6 points) For 32-bit memory addresses, how many bits are used for each of the tag, index and byte-offset (aka byte-select) fields of the address?

Tag: 29 (32 bit address - 1(index bit) - 2(byte offset) - 0(log of words/block))

Index: 1 (only 2 sets)

Byte-offset: 2 (block size is 4 bytes)

b) (12 points) What addresses and data are stored in the cache after the following stream of addresses and data are read? For simplicity, write the full address in the space for the cache tag, even though the several of the low order bits would normally be omitted. Also, assume that at machine startup time, the leftmost block in the set (as seen in the diagram below) is the one considered least-recently-used.

Address/Data stream (all are reads):
Address = 0, Data= 23
Address = 9, Data= 120
Address = 8, Data = 31
Address = 6, Data = 49
Address = 9, Data = 120
Address = 7, Data = 17
Address = 2, Data = 203

Write your answer here:

index 0: address = 6 Data = 49 address = 2 Data = 203

index 1: address = 9 Data = 120 address = 7 Data = 17

Problem #2 -- Memory hierarchies (24 points)

(a) (12 points) Consider a two-level hierarchy of fast and slow memory. The fast memory is 100 times faster than the slow memory and can be used 70% of the time. Now consider an alternative design with three memory levels at three speeds. The fast level is 120 times faster than the slow level and the medium level is 50 times faster than the slow level. The fast level can be used 50% of the time, and the medium level can be used 25% of the time. Compare the execution time in both cases. (Note that the access time for the "slow" memory is the same in both cases).

A $\underbrace{.3 \times 1}_{\text{slow access}} + \underbrace{.7 \times .01}_{\text{fast access}} = \boxed{0.307}$

B is 1.185 times as fast as A.

B $\underbrace{.25 \times 1}_{\text{slow access}} + \underbrace{.25 \times .02}_{\text{medium access}} + \underbrace{.5 \times 0.0083}_{\text{fast access}} = \boxed{0.259}$

(b) (12 points) For an n-level memory hierarchy, derive a formula for *speedup* -- the ratio of the speed of the system using the entire hierarchy to the speed of the system using only the slowest level. Assume the information you have is similar to that in part (a). That is, you have the relative speed of each level compared to the slowest level and the percentage of the time each level can be used, with these percentages adding up to 100%. Of course, you will always use the fastest level whenever possible.

S = speed of slowest level, level n

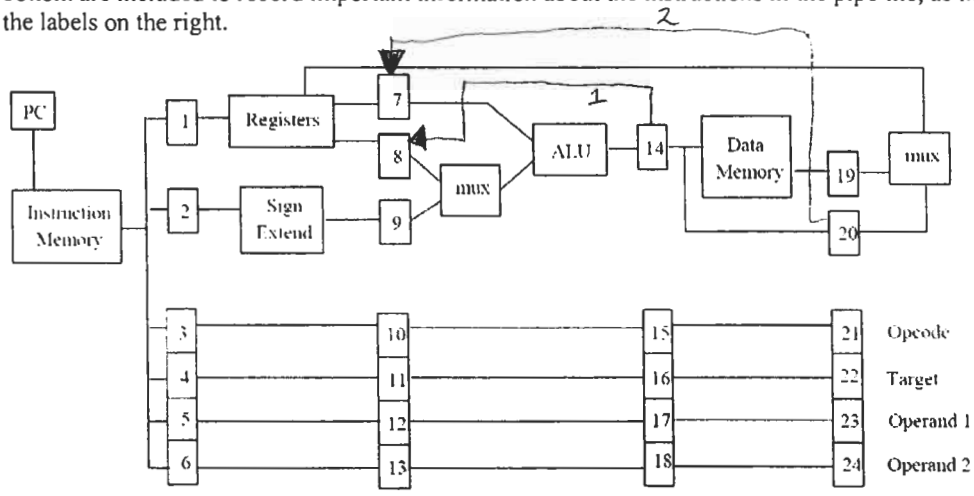
P_i = percentage of requests that can use level i, $\sum_{i=1}^{n} P_i = 1$

S_i = speed of level i, $S_i = k_i S$, $0 < k < 1$

$$\text{Speedup} = \frac{S}{\sum_{i=1}^{n} P_i S_i} = \boxed{\frac{1}{\sum_{i=1}^{n} P_i k_i}}$$

Problem #3 – Pipelines (30 points)

Consider a MIPS-like machine with the organization shown below, where pipeline latch registers are labelled with numbers, and also consider the following instruction sequence for this machine. The first argument of an ALU operation is the target, the second denotes the upper ALU input (Operand 1) in the figure, and the third denotes the lower ALU input (Operand 2). Note that the pipeline registers along the bottom are included to record important information about the instructions in the pipeline, as indicated in the labels on the right.



1	SW R4,10(R2)	Store word in R4 to address (R2)+10
2	LW R1,100(R3)	Load word at address (R3)+100 into R1
3	ADD R2,R1,R4	(R2) = (R1) + (R4)
4	SUB R4,R5,R3	(R4) = (R5) - (R3)
5	SUB R3,R1,R2	(R3) = (R1) - (R2)

(a) (8 points) List the data hazards in this code, assuming no out of order execution (so these are all RAW hazards). Give a pair of line numbers and a register id for each hazard.

2, 3 R1

3, 5 R2

ALSO THERE IS A DEPENDENCE (BUT NO HAZARD) FROM 2 → 5 ON R1.

(b) (12 points) Each of the following instruction sequences contains a single hazard. Indicate the EARLIEST pipline registers to be compared in order to detect the hazard in each case. You don't have to use the notation from the book here, just say something like "check 5=12".

1. ADD R1,R2,R3
   SUB R4,R2,R1
   LW R1,15(R3)

   CHECK 6=11

2. SUB R1,R2,R3
   SW R2,10(R3)
   ADD R4,R1,R3

   CHECK 5=16

3. LW R1,40(R2)
   SUB R3,R1,R4
   ADD R2,R2,R4

   CHECK 5=11

   → STALL FOR ONE CYCLE, THEN FORWARD FROM 19 TO 7

(c) (10 points) Draw data forwarding paths onto the figure above for each of the hazards in (b). That is, draw an arrow from the source latch to the destination latch of the forwarding path in each case. Label each path with the hazard number from (b) that it eliminates. If data cannot be forwarded without stalling, write that down and do not draw a path for that hazard.

Problem #4 – Cache design (16 points)

Suppose that you know that a program will execute a stream of LW instructions with the following pattern:

LW from address 0x800000
LW from address 0x800004
LW from address 0x800008
LW from address 0x80000C
LW from address 0x900000
LW from address 0x900004
LW from address 0x900008
LW from address 0x90000C
LW from address 0xA00000
LW from address 0xA00004
LW from address 0xA00008
LW from address 0xA0000C
...

a) (8 points) What is the optimal cache block size (in bytes) for a machine executing this program? **Why?** Assume that the total cache size is limited to 1KB.

16 bytes = 4 words

This gives maximal spatial locality. The requests are spaced too far apart to give any more than 4 requests in a block, so we choose the smallest size that gives that much locality.

Note also that this sequence has no temporal reuse.

b) (8 points) What is the average memory access time for LW instructions in this program if the cache hit time is 2 cycles and the miss penalty is 50 cycles? Show your work.

$AMAT = TIME_{HIT} + MISS\ RATE \times TIME_{MISS}$

$2\ \text{cycles/request} + .25\ \text{miss/request} \times 50\ \text{cycles/miss}$

$2\ \text{cycles/request} + 12.5\ \text{cycles/request} = \boxed{14.5\ \text{cycles/request}}$

Problem #5 – Branch prediction in a deep pipeline (12 points)

The recently announced IBM Cell processor has a long pipeline. As a result, the branch mis-prediction penalty is large: 18 cycles.

Suppose that typical code running on this processor executes a branch instruction every nine instructions. Also suppose that all instructions have a CPI of 1.0, unless there is a branch misprediction penalty.

How accurate does the branch predictor have to be (i.e. what percent of the time does it have to predict correctly) in order to keep total machine performance at 90% of what it would be without any branch misprediction penalties? Show your work.

CPI GOOD = 1.0

90% performance => CPI = 1.11

THAT MEANS THE BRANCH MISPREDICTION PENALTIES MUST BE ≤ .11 cycle/instruction

TOTAL PENALTIES:

(% of instructions which are branches) x (mispredict rate) x (Penalty for each mispredict) : .11 cycles/instruction

$\frac{1}{9} \times m \times 18 \text{ cycles/mispredict} = 0.11 \text{ cycles/instruction}$

$m = \frac{1}{18} \approx 0.0556$, so the hit rate must be **94%**