EECS 370
Exam 2

Name:

University of Michigan uniqname:

Open book, open notes. Calculators are permitted, but no laptops, PDAs, cell phones, etc. This exam has 8 pages, 8 questions, and 100 points. Questions vary in difficulty; it is strongly recommended that you do not spend too much time on any one question.

1. *(12 points)*: Please circle TRUE or FALSE for each statement.

(a) TRUE FALSE ANSWER: FALSE
RANDOM cache line replacement is a great strategy to compare against other replacement algorithms, because it is the worst performing replacement strategy that exists.

(b) TRUE FALSE ANSWER: TRUE
For a given instruction set, a MULTI-CYCLE processor implementation requires less hardware than a PIPELINED processor implementation.

(c) TRUE FALSE ANSWER: TRUE
It is not possible to eliminate all data hazard stalls in the LC2K4 pipeline, even if the design is FULLY BYPASSED (has full forwarding).

(d) TRUE FALSE ANSWER: FALSE
Modifying a processor instruction set to reduce average CPI will always produce a faster design, assuming that clock speed is unaffected.

2. *(17 points)*:
Please compute the accuracy (i.e., % of correct predictions on average) of the following branch predictors, assuming that branches are taken 60% of the time, and 40% of the branches have targets in the forward direction (this statistic holds true for both taken and not taken branches):

(a) Predict direction of branch randomly.

ANSWER: 0.6 * 0.5 + 0.4 * 0.5 = 0.5, 50% accurate

(b) Predict all branches TAKEN.

ANSWER: 0.6 * 1.0 = 0.6, 60% accurate

(c) Predict all branches NOT TAKEN.

ANSWER: 0.4 * 1.0 = 0.4, 40% accurate

(d) Predict TAKEN if offset negative, NOT TAKEN otherwise.

ANSWER: 0.6 * 0.6 + 0.4 + 0.4 = 0.52, 52% accurate

(e) Why is it more expensive (e.g., in performance or hardware) to build a **predict TAKEN** branch predictor, compared to a **predict NOT TAKEN** branch predictor?

ANSWER: Predicting NOT TAKEN needs no extra hardware; just continue fetching sequential instructions. Predicting TAKEN requires keeping a cache of branch program counter to branch target adddress mappings.

3. *(12 points)*:
Consider a normal 5-stage LC-2K4 pipeline as discussed in class. Branches are resolved in the MEM stage and branches are predicted as not taken. 1/3 of load instructions incur a one cycle stall due to data hazards that can not be resolved via forwarding. Assume the instruction cache has a perfect hit rate, but the data cache has a 10% miss rate for both reads and writes. For a memory read, a data cache miss causes a 20 cycle stall. For a memory write, a data cache miss does not cause a stall. Analysis has shown the following dynamic instruction breakdown for program X: 15% not taken branches, 10% taken branches, 10% loads, 15% stores, 50% R-type instructions (add/nand).

(a) What is the CPI of this machine when running program X?

ANSWER: 1 + 0.1 * 3 + 0.1 * 1/3 * 1 + 0.1 * 0.1 * 20
= 1 + 0.3 + 0.033 + 0.2 = 1.533

(b) Now assume the pipeline was stretched out to 10 stages (numbered 1 to 10, with instruction fetch beginning in stage 1, and instructions finally completing in stage 10). Branches are resolved in stage 7. The clock frequency is increased to twice that of the original processor with the 5 stage pipeline. Data cache misses cause a 40 cycle stall. 1/3 of load instructions still incur a one cycle stall due to data hazards that can not be resolved via forwarding. What is the CPI for this new pipeline design?

ANSWER: 1 + 0.1 * 6 + 0.1 * 1/3 * 1 + 0.1 * 0.1 * 40
= 1 + 0.6 + 0.033 + 0.4 = 2.033

(c) Which of these machines performs better, the original 5 stage pipeline or the new 10 stage pipeline? How many times faster is it than the other one?

ANSWER: Considering both the CPI change and the doubled clock frequency, the new 10 stage pipeline is faster by a factor of:
1.533 * 2 / 2.033 = 1.508

4. *(12 points)*:
Consider the LC2 processor with the 5 stage pipeline as described in the lecture slides. Assume it runs programs with the following instruction mix: 40% R-type (add/nand), 20% LW, 20% SW, 20% BEQ. All load and store operations access memory in one cycle. Data hazards are resolved by forwarding whenever possible. 50% of LW instructions are followed by a dependent instruction. 50% of BEQ instructions are taken. The processor clock speed is 100 MHz.

(a) How many MIPS (millions of instructions per second) does this processor run at with this instruction mix?

ANSWER:
CPI: 1 (base) + 0.3 (taken branches) + 0.1 (lw stalls) = 1.4
100 MHz / 1.4 CPI = 71.4 MIPS

(b) The original LC2 processor is modified as follows: The branch hardware has been streamlined so that branches are resolved in the EX stage rather than the MEM stage. Nothing else is changed in the pipeline. The processor clock speed must be decreased to 80 MHz to accommodate this change. How many MIPS does this processor run at?

ANSWER:
CPI: 1 (base) + 0.2 (taken branches) + 0.1 (lw stalls) = 1.3
80 MHz / 1.3 CPI = 61.5 MIPS

(c) The original LC2 processor is modified to shorten the pipeline from five stages to four as follows: Data memory access is moved into the EX stage. A special dedicated adder (separate from the ALU) is put into this stage to calculate the **base register plus offset** value and send it as an address to the data memory. This effectively merges EX and MEM into one stage and removes the load-use stall, but with a slower clock as a result. The old MEM stage goes away entirely; the WB stage immediately follows the EX stage. Branches are resolved in the WB stage. Data hazards are resolved by forwarding whenever possible. The processor clock speed must be decreased to 90 MHz to accommodate this change. How many MIPS does this processor run at?

ANSWER:
No more LW stalls, forwarding can resolve this hazard.
CPI: 1 (base) + 0.3 (taken branches) = 1.3
90 MHz / 1.3 CPI = 69.2 MIPS

5. *(14 points)*:
The alpha version of the LC2K4 processor did not support using immediate values in the main ALU as operands. Hence, the LC2K4-Alpha had an extra stage in the pipeline (with its own dedicated adder separate from the main ALU). This stage is called **AC** (Address Calculation); the **base register + immediate offset** address calculation is done in this stage. Given that a program has the following properties:

- 20% of all instructions are loads and stores that use an address with a base register whose value is set in the immediately-preceding instruction, which is an r-type instruction
- 10% of all instructions are loads and stores that use an address with a base register whose value is set in the immediately-preceding instruction, which is an **lw** instruction
- 20% of all instructions are **nand** and **add** instructions that immediately follow a **lw** and depend on the result of that load.
- 20% of all instructions are **nand** and **add** instructions that follow two instructions after a **lw** and depend on the result of that load (e.g. **lw**, something, **add**)

These four groups listed above are totally disjoint. More specifically, there are no **lw** instructions that are followed by two dependent instructions in a row; something like the following never occurs: **lw 0 3 one, add 1 3 4, add 2 3 5.** There are no instruction combinations other than those specifically enumerated above that could cause any stalling. There are no BEQ instructions; the program runs once through in consecutive address order from beginning to end. There is full forwarding to the extent physically (temporally) possible. Calculate the CPI of a program with the preceding statistics on the following two pipeline implementations of the LC2K4.

(a) IF ID AC EX MEM WB

(b) IF ID EX AC MEM WB

ANSWER (for both parts):

Hazard Type	Hazard Frequency	Stalls part (a)	Stalls part (b)
R lw	0.2	1	0
lw lw	0.1	2	1
lw R	0.2	1	2
lw X R	0.2	0	1

```
Part (a) stall total:  0.2 * 1 + 0.1 * 2 + 0.2 * 1 + 0.2 * 0 = 0.6

Part (b) stall total:  0.2 * 0 + 0.1 * 1 + 0.2 * 2 + 0.2 * 1 = 0.7

Total CPI:
Part (a):  1.6
Part (b):  1.7
```

6. *(9 points)*:

A certain new, high-performance computer has a cache with the following characteristics:

- 48 bit addresses
- Byte addressable memory
- 8 megabyte cache size (1 megabyte = 2^{20} bytes = 1,048,576 bytes)
- 4 kilobyte cache block size (1 kilobyte = 2^{10} bytes = 1,024 bytes)
- 16-way set associative cache
- Address bit numbering: most significant bit is bit 47, least significant bit is bit 0.

ANSWERS:

```
4 kilobyte cache block size
4 kilobytes = 2**12 bytes
so 12 bits for the block offset
8 megabyte cache size / 4 kilobyte block size = 2K blocks in the cache
= 2**11 blocks in the cache
16 way set associative = 2**4 blocks per set
2**11 blocks / 2**4 blocks per set = 2**7 sets
so 7 bits for the set index
rest of the bits (48 - (7 + 12) = 29) are the tag
```

(a) What range of address bit numbers are used for each of the following?

Set index: bits 18 - 12

Block offset: bits 11 - 0

Tag: bits 47 - 19

(b) For 0x3f2c84de7b29, show the set index, block offset, and tag. You may state your answers in either binary or hexadecimal.

```
0x3f2c84de7b29 in binary:
001111110010110010000100110111001111011001010 01
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTSSSSSSSBBBBBBBBBBBB
```

Set index: 110 0111 binary = 0x67

Block offset: 1011 0010 1001 binary = 0xb29

Tag: 0 0111 1110 0101 1001 0000 1001 1011 binary = 0x07e5909b

7. *(9 points)*:
You are designing a data cache with a single objective: minimize the total number of cache blocks transferred between the cache and main memory. Each copy of a block from the main memory to the cache or from the cache to the main memory counts as one transfer (regardless of the block size). For situations when a CPU write operation goes directly through to main memory, each such write operation counts as one transfer. You are limited to a specific total cache data size, and this size is significantly smaller than the size of the program that you are going to run on this system, but you have complete freedom on all other cache parameters. You know that the program you are going to run has a high degree of both spatial and temporal locality, for both reads and writes. For each of the three design decisions below, circle the BEST answer:

5.1. Write policies:

(a) You should use a write back, allocate on write policy

(b) You should use a write back, no allocate on write policy

(c) You should use a write through, allocate on write policy

(d) You should use a write through, no allocate on write policy

(e) The write policy is not likely to have much effect on the total number of blocks transferred.

ANSWER: (a)

5.2. Block size:

(a) You should use a 1 byte block size

(b) You should make the block size as large as possible.

(c) For your given program, there is likely to be some intermediate block size that is optimal.

(d) The block size is not likely to have much effect on the total number of blocks transferred.

ANSWER: (c)

5.3. Associativity:

(a) You should use a direct mapped cache

(b) You should use a 2-way set associative cache

(c) You should use a 4-way set associative cache

(d) You should use a fully associative cache

(e) There is likely to be some intermediate degree of associativity that is optimal.

(f) The associativity is not likely to have much effect on the total number of blocks transferred.

ANSWER: (d)

8. *(15 points)*:
For each of the following cache configurations, how many cache misses will result for the following series of word addressed references (given in decimal)? The cache is initially empty and **least recently used** (LRU) is used to evict blocks from the cache. All caches are fully associative. Show your work for partial credit.

1, 4, 8, 5, 1, 3, 2, 10, 9, 3, 4, 7, 6, 1, 8, 0, 6, 5, 7, 8, 2, 10, 1

(a) Cache contains eight one-word blocks.

ANSWER:

```
For each reference it is shown whether it hit or missed,
and the contents of the cache are listed after handling
that reference is completed, in MRU to LRU order.

  1  M   1  -  -  -  -  -  -  -
  4  M   4  1  -  -  -  -  -  -
  8  M   8  4  1  -  -  -  -  -
  5  M   5  8  4  1  -  -  -  -
  1  H   1  5  8  4  -  -  -  -
  3  M   3  1  5  8  4  -  -  -
  2  M   2  3  1  5  8  4  -  -
 10  M  10  2  3  1  5  8  4  -
  9  M   9 10  2  3  1  5  8  4
  3  H   3  9 10  2  1  5  8  4
  4  H   4  3  9 10  2  1  5  8
  7  M   7  4  3  9 10  2  1  5
  6  M   6  7  4  3  9 10  2  1
  1  H   1  6  7  4  3  9 10  2
  8  M   8  1  6  7  4  3  9 10
  0  M   0  8  1  6  7  4  3  9
  6  H   6  0  8  1  7  4  3 10
  5  M   5  6  0  8  1  7  4  3
  7  H   7  5  6  0  8  1  4  3
  8  H   8  7  5  6  0  1  4  3
  2  M   2  8  7  5  6  0  1  4
 10  M  10  2  8  7  5  6  0  1
  1  H   1 10  2  8  7  5  6  0

15 misses
```

(b) Cache contains four two-word blocks.

ANSWER:

```
The addresses are first converted to block numbers
(divide by 2 and discard the remainder).
Then, for each reference it is shown whether it hit or missed,
and the contents of the cache are listed (as block numbers)
after handling that reference is completed, in MRU to LRU order.

  1  0  M   0  -  -  -
  4  2  M   2  0  -  -
  8  4  M   4  2  0  -
  5  2  H   2  4  0  -
  1  0  H   0  2  4  -
  3  1  M   1  0  2  4
  2  1  H   1  0  2  4
 10  5  M   5  1  0  2
  9  4  M   4  5  1  0
  3  1  H   1  4  5  0
  4  2  M   2  1  4  5
  7  3  M   3  2  1  4
  6  3  H   3  2  1  4
  1  0  M   0  3  2  1
  8  4  M   4  0  3  2
  0  0  H   0  4  3  2
  6  3  H   3  0  4  2
  5  2  H   2  3  0  4
  7  3  H   3  2  0  4
  8  4  H   4  3  2  0
  2  1  M   1  4  3  2
 10  5  M   5  1  4  3
  1  0  M   0  5  1  4

13 misses
```

(c) Cache contains two four-word blocks.

ANSWER:

```
The addresses are first converted to block numbers
(divide by 4 and discard the remainder).
Then, for each reference it is shown whether it hit or missed,
and the contents of the cache are listed (as block numbers)
after handling that reference is completed, in MRU to LRU order.

  1  0  M   0  -
  4  1  M   1  0
  8  2  M   2  1
  5  1  H   1  2
  1  0  M   0  1
  3  0  H   0  1
  2  0  H   0  1
 10  2  M   2  0
  9  2  H   2  0
  3  0  H   0  2
  4  1  M   1  0
  7  1  H   1  0
  6  1  H   1  0
  1  0  H   0  1
  8  2  M   2  0
  0  0  H   0  2
  6  1  M   1  0
  5  1  H   1  0
  7  1  H   1  0
  8  2  M   2  1
  2  0  M   0  2
 10  2  H   2  0
  1  0  H   0  2

10 MISSES
```