CPSC 3300 - Fall 2013 – Exam 2 ***<with answers; blue parts are relevant for exam 3 for Spring 2014>***

True/False. Circle T or F. (1.5 pts. each)

1. T / F A microprogrammed control unit is more flexible than hardwired control unit.
2. T / F A microprogrammed control unit is typically slower than a hardwired control unit.
3. T / F A control store word must contain a bit position for each control point in the datapath.
4. T / F Pipeline stages are separated by pipeline latches.
5. T / F The five-stage pipeline we studied needed a register file with two write ports and one read port.
6. T / F Even with forwarding, the five-stage pipeline we studied had a load-use penalty of one cycle.
7. T / F A predict-untaken is easier to implement than predict-taken since a BTA is not needed.
8. T / F A predict-untaken scheme is better to implement since most branches are untaken.
9. T / F A two-level adaptive branch predictor first accesses a finite-state predictor that will determine the correct branch history shift register to use.
10. T / F SRAM memory is typically cheaper than DRAM.
11. T / F SRAM memory must be periodically refreshed to not lose values.
12. T / F The "static" part of SRAM means that SRAM is read-only and you cannot change its contents.
13. T / F The "static" part of SRAM means that SRAM is write-once and you cannot change the contents of a word after the first write to that word.
14. T / F Locality of reference is a property of the CPU on which a program is running (e.g., how big the cache is).
15. T / F Temporal locality means that when a memory location is accessed once, a neighboring location is likely to be accessed next.
16. T / F A miss penalty is the ratio of the number of memory accesses not found in the cache to the total number of memory accesses.
17. T / F Burst memory transfers are useful for I/O devices but not for caches.
18. T / F Most caches in microprocessors we studied implement read-allocate as the read-miss policy.
19. T / F All caches in microprocessors we studied implement read-allocate as the read-miss policy.
20. T / F An "*n*-way" set associative cache requires *n* to be a power of two (e.g., 2-way, 4-way, 8-way, ...).
21. T / F A set-associative cache needs a replacement (or eviction) policy, but a fully-associative cache doesn't.
22. T / F Steps for accessing a fully-associative cache are 1) index lookup, 2) tag match, and 3) byte selection.
23. T / F Fully associative caches can have conflict and capacity misses but not compulsory misses.
24. T / F A write-back cache needs a valid bit, but a write-through cache doesn't.

***<answers>***

1,2,4,6,7,18 are true. The rest are false.

***<discussion of false statements from chapter 4 material>***

3. A control store word can contain a bit position for each control point in the datapath. *However, this is so inefficient in bit representation that most control stores use words with encoded fields. These fields are then decoded into the individual control signals using decoders. E.g., consider all the possible sources to an internal bus – only one of the associated control points can be activated in a given cycle.*

5. The five-stage pipeline we studied needed a register file with two read ports and one write port.

8. A predict-untaken scheme is not as good since most branches are taken.

9. A two-level adaptive branch predictor first accesses a branch history shift register that will determine the correct finite-state predictor to use.

***<discussion of false statements from chapter 5 material>***

10. SRAM memory is typically more expensive than DRAM.

11. DRAM memory must be periodically refreshed to not lose values.

12/13. The "static" part of SRAM *means that SRAM does not require periodic refreshing and holds it values as long as it stays powered on.*

14. Locality of reference is a property of program behavior. *The cache exploits whatever locality is present.*

15. Spatial locality means that when a memory location is accessed once, a neighboring location is likely to be accessed next.

16. A miss rate is the ratio of the number of memory accesses not found in the cache to the total number of memory accesses.

17. Burst memory transfers are useful for both I/O devices and caches.

19. Most caches in microprocessors we studied implement read-allocate as the read-miss policy, *but the hint bits in the Itanium load instructions can cause the hardware to not copy the missing data into given levels of the caches on read misses. The data is bypassed directly to the CPU instead, without replacing data in the cache(s) that likely has better temporal locality.*

20. An "n-way" set associative cache can have an arbitrary number, n, of banks.

21. A set-associative cache needs a replacement (or eviction) policy, but a direct-mapped cache doesn't. *Likewise a fully-associative cache needs a replacement policy.*

22. Steps for accessing a direct-mapped cache are 1) index lookup, 2) tag match, and 3) byte selection. *Steps for accessing a fully-associative cache are 1) tag match, 2) routing, and 3) byte selection.*

23. Fully associative caches can have compulsory and capacity misses but not conflict misses.

24. A write-back cache needs a modified bit, but a write-through cache doesn't. *All caches need valid bits.*

25. Identify the five stages of the simple scalar pipeline we studied and explain what each stage does when processing the following instruction. (15 pts.)

```
load R2,0(R1)    // R2 <- memory[R1+0]
```

***<answers>***

IF – instruction fetch – the instruction word is obtained from the icache using the address in the PC, and the PC is incremented

ID – instruction decode – the load opcode is decoded, control signals generated, register R1 is read, and the offset 0 is sign-extended

EX – execute – the value in R1 and the offset 0 are added by the ALU to produce the effective address

MEM – memory – the data word is obtained from the dcache using the address R1+0

WB – writeback – the data word is stored into register R2

26. Associate each term or statement below with a type of dependency. Circle one or more of RAW, WAR, or WAW. (Destination registers are listed first for load and add instructions.) (8 pts.)

a) RAW / WAR / WAW True data dependency
b) RAW / WAR / WAW False data dependency
c) RAW / WAR / WAW load R2,0(R1) followed by add R1,R1,4
d) RAW / WAR / WAW load R2,0(R1) followed by add R2,R2,4

***<answers>***

a) RAW
b) WAR, WAW
c) WAR
d) RAW, WAW

***<the following questions are from chapter 5 material, which would be material for exam 3 in Spring 2014>***

27. Consider a 4 GB byte-addressable main memory with a four-way set-associative cache of 2 MB and 32 bytes per line.

a) How many total lines are there in cache? (not just per bank) (2 pt.)
b) Show how the main memory address is partitioned into fields for the cache access and give the bit lengths of those fields. (6 pts.)
c) How many total tag comparators are there in the cache? (2 pt.)

***<answers>***

a) 2MB / (32 bytes/line) = 2^21 / 2^5 lines = 2^16 lines

b) offset bits: log2( 32 ) = 5
index bits: log2( 2^16 / 4 ) = log2( 2^14 ) = 14
tag bits: log2( 4G ) - 14 - 5 = 32 - 19 = 13


c) 4 (one per bank)

28. Assume a 256-word main memory and a four-line direct-mapped cache with two words per line. The cache is initially empty. For the word address reference stream given below, circle which of the references are hits. Also, show the final contents of the cache. (The word addresses are in decimal.) (15 pts.)

2, 10, 3, 26, 12, 4, 12, 10, 5, 12, 6, 10

***<answer>***

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | | 0,1 | 8,9 | 16,17 | 24,25 … | empty |
| 1 | | 2,3 | 10,11 | 18,19 | 26,27 … | ~~2,3~~ ~~10,11~~ ~~2,3~~ ~~26,27~~ 10,11 |
| 2 | | 4,5 | 12,13 | 20,21 … | | ~~12,13~~ ~~4,5~~ ~~12,13~~ ~~4,5~~ 12,13 |
| 3 | | 6,7 | 14,15 | 22,23 … | | 6,7 |

only the last reference (10) is a hit

29. Consider a computer system that contains a cache with 1-clock-cycle access time and a memory with 20-clock-cycle access time. If the hit rate is 90% and a cache miss requires both a single cache access and a single memory access, what is the average memory access time (AMAT) in terms of clock cycles? (6 pts.)

***<answer>***

.9 * 1 cycle + .1 * (1 + 20) cycles = 1 + .1*20 cycles = 1 + 2 cycles = 3 cycles

30. Assume you are running the following program segment on a system with a 32 KB two-way set-associative data cache with 64-byte line size.

```
double a[2048],b[2048],c[2048],d[2048];  /* each double is 8 bytes */
...
for(i=0;i<2048;i++){
   a[i] = 0.0;
   b[i] = 0.0;
   c[i] = 0.0;
   d[i] = 0.0;
}
```

A friend suggests that it would be faster to write the program segment in this way:

```
double a[2048],b[2048],c[2048],d[2048];  /* each double is 8 bytes */
...
for(i=0;i<2048;i++){
   a[i] = 0.0;
   b[i] = 0.0;
}
for(i=0;i<2048;i++){
   c[i] = 0.0;
   d[i] = 0.0;
}
```

Is your friend correct? Why or why not? (10 pts.)

***<answer>***

Yes, your friend is correct.

Your program sees conflict misses for all array accesses. Your friend's program runs without conflict misses.

***<extended explanation – note that your answer does not have to be this detailed>***

Arrays that are a power of two in size should always be a red flag regarding possible conflict misses!

Note that each bank of the cache is 16 KB in size, and each array is 2K*8 B = 16 KB is size. There are also eight array values in a 64-byte line. Thus, because the arrays are allocated back to back, a[0]-a[7], b[0]-b[7], c[0]-c[7], and d[0]-d[7] are all mapped to the same set index. Likewise, a[8]-a[15], b[8]-b[15], c[8]-c[15], and d[8]-d[15] are all mapped to the next set index. Likewise for the rest of the memory blocks.

The reference behavior is: a[0], b[0], c[0], d[0], a[1], b[1], c[1], d[1], …
And the miss behavior is thus:

a[0] compulsory miss – place a[0]-a[7] in one bank
b[0] compulsory miss – place b[0]-b[7] in other bank
c[0] compulsory miss – place c[0]-c[7] in bank that held a[0]-a[7] (using either FIFO or LRU)
d[0] compulsory miss – place d[0]-d[7] in bank that held b[0]-b[7]
a[1] conflict miss – place a[0]-a[7] in bank that held c[0]-c[7] (conflict because a[1] had been in cache)
b[1] conflict miss – place b[0]-b[7] in bank that held d[0]-d[7] (conflict because b[1] had been in cache)
…

This continues for the complete loop – there are no cache hits for any of the memory blocks in the arrays because of the four-way competition for the two cache lines in each set.

Your friend's suggested code limits the competition to two array blocks for the two cache lines in a set. Both array blocks can be present at the same time in a set, and this eliminates the conflict misses.

The reference behavior of your friend's code is: a[0], b[0], a[1], b[1], …, c[0], d[0], c[1], d[1], …
And the new miss behavior is thus:

a[0] compulsory miss – place a[0]-a[7] in one bank
b[0] compulsory miss – place b[0]-b[7] in other bank
a[1] hit
b[1] hit
…
a[8] compulsory miss – place a[8]-a[15] in one bank
b[8] compulsory miss – place b[8]-b[15] in other bank
a[9] hit
b[9] hit
…
c[0] compulsory miss – place c[0]-a[7] in one bank
d[0] compulsory miss – place d[0]-b[7] in other bank
c[1] hit
d[1] hit
…

XC. Using the same initial program as in question 30 and the same cache (32 KB two-way set-associative data cache with 64-byte line size), change the data declarations (but not the data type of the arrays) to make the program run faster. Explain the effect of the changes on the cache behavior. (up to 10 pts.)

```
double a[2048],b[2048],c[2048],d[2048];  /* each double is 8 bytes */
...
for(i=0;i<2048;i++){
   a[i] = 0.0;
   b[i] = 0.0;
   c[i] = 0.0;
   d[i] = 0.0;
}
```

***<suggested answer>***

The key is to prevent the mapping of the four array blocks being accessed together in the body of the loop to the same set index. There are many possible solutions, including:

```
double a[2048],b[2048],pad[8],c[2048],d[2048];
```

where the pad array is at least a cache line in size (64B). Another approach is to never declare large arrays with sizes that are a power of 2. "`double a[2049],b[2049],c[2049],d[2049];`" would reduce the number of conflict misses, while "`double a[2056],b[2056],c[2056],d[2056];`" would eliminate the conflict misses for this cache.