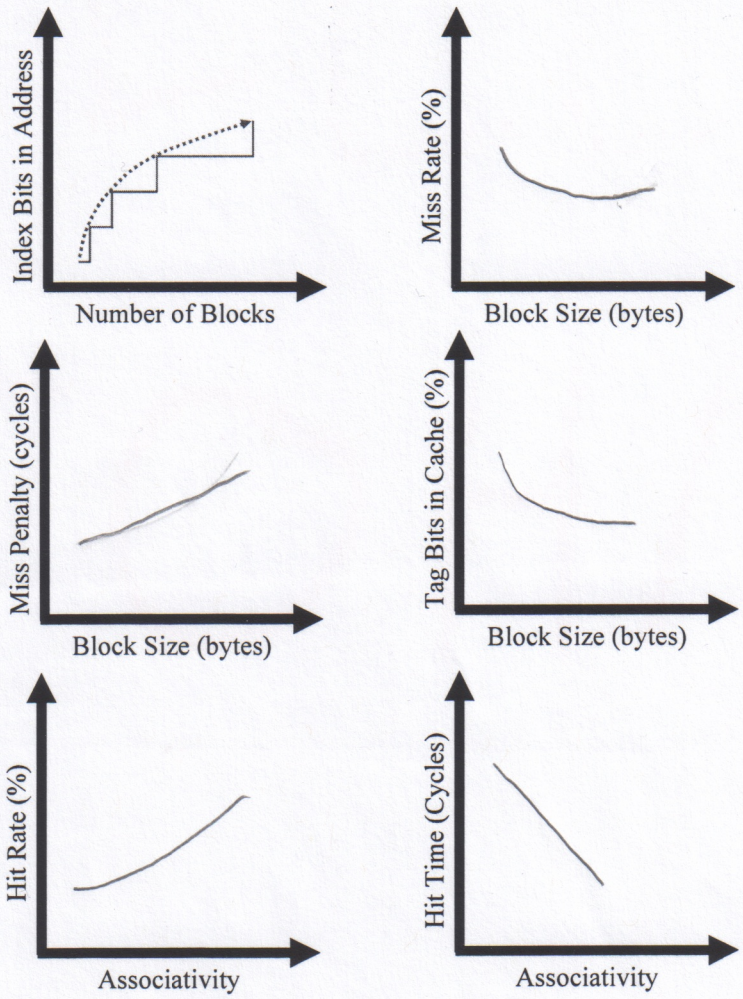
Cpre 381 Final Exam

Trevor Lund – 212265053

1. Cache Short Answer
    a. Indicate temporal or spatial locality and explain.
        i. This code snippet shows temporal locality because every time it loops, it accesses the same two memory locations ($s0[32767] and $s0[0]) and loads them into registers $t1 and $t0, respectively.
        ii. This code snippet uses both temporal and spatial locality. Every time it loops back to LP1, it uses temporal locality by loading the same 4 bytes for each iteration. When the code loops back to LP2, it uses spatial locality by decrementing the memory offset by 1 each iteration.
        iii. This code snippet uses spatial locality because each iteration of the loop decrements the offset of the load word operation by 1, keeping each load operation loading words that are close to each other in memory.

b. Graphs (See scanned image)

(b) Consider the following six graphs. The first of these represents the relationship between the number of blocks in a cache and the number of required index bits in the memory address (the relationship is generally logarithmic). Fill in the remaining graphs based on your understanding of the general relationship between the X-axis cache configuration parameter and the corresponding Y-axis performance / efficiency metric. [5 points]



c. A write-back cache only has to write to the main memory when there's a conflict miss, or when there's a read/write from a different memory address that uses the same cache block. A write-through cache, on the other hand, writes to the main memory every time there's a write command. Write-back exploits temporal locality of programs, only needing to write to and read from the cache, streamlining the memory processes.

d. If the MSBs of an address were used as the cache index, it would nullify the benefits of spatial locality. Normally, if a memory address is loaded into the cache, consecutive bytes are also loaded into the block. If the program is accessing a large chunk of consecutive memory (e.g. a large array), a large portion of it would be loaded into the cache. However, with the MSBs as the index, large portions of consecutive memory would be indexed to the same block in the cache, forcing a miss for reading things like arrays.

e. (Not sure if test meant 2^N or 2N, so I gave both answers)

   True, because a cache with 2^N blocks would be the same as the main memory module.

   True, because a cache with 2N blocks will function largely the same as one with only N blocks, the only difference being the former will have more blocks in which to store data.

2. MIPS Programming

a. Fliprow:

```
addi $t0, $a1, -1
add $t0, $a0, $t0       #t0 = address of last pixel in the row
frl:
lb $s0, 0($a0)          #s0 = pixel k
lb $s1, 0($t0)          #s1 = pixel n-k-1
sb $s0, 0($t0)
sb $s1, 0($a0)          # swap
addi $a0, $a0, 1
addi $t0, $t0, -1       # increment memory locations
sub $t1, $t0, $a0
bgtz $t1, frl           # continue flipping if not done
jr $ra
```

b. Flipimage:

```
addi $s0, $0, 0
fil:
addi $sp, $sp, -12
sw $a0, 0($sp)
sw $a1, 4($sp)
sw $s0, 8($sp)          # save the original arguments before calling the row flip
mul $t1, $a2, $s0       # set the offset for row flip
add $a0, $a0, $t1       # set the start position for row flip
add $a1, $0, $a2        # set the number of pixels for row flip
jal fliprow
lw $a0, 0($sp)
lw $a1, 4($sp)
lw $s0, 8($sp)
addi $sp, $sp, 12       # pop variables off the stack
addi $s0, $s0, 1        # increment row counter
bne $a1, $s0 fil        # continue to loop if all the rows haven't been flipped
```

3. Cache Organization and Performance
   a. See scan

i. `lw $t0, 01000100two ($zero)`

Index	D	V	Tag	Data	D	V	Tag	Data
0	0	0			0	0		
1	0	1	0100		0	0		

Memory Access? : Read, 01000100

ii. `sw $t0, 10000100two ($zero)`

Index	D	V	Tag	Data	D	V	Tag	Data
0								
1	0	1	0100		0	1	1000	

Memory Access? : Write, 10000100

iii. `lw $t1, 01000100two ($zero)`

Index	D	V	Tag	Data	D	V	Tag	Data
0								
1	0	1	0100		0	1	1000	

Memory Access? :

iv. `lw $t2, 11000100two ($zero)`

Index	D	V	Tag	Data	D	V	Tag	Data
0								
1	0	1	0100		0	1	1100	

Memory Access? : Read, 11000100

v. `sw $t1, 11000000two ($zero)`

Index	D	V	Tag	Data	D	V	Tag	Data
0	0	1	11		0	0		
1	0	1	0100		0	1	11	

Memory Access? : Write, 11000000

b. AMAT = Hit Time + (Miss Rate * Miss PenaltAMAT = 8 + (.05 * 80) = 12 cycles
c. This occurs because of an increase in conflict misses. These occur more frequently because there are more memory blocks mapped to the same cache block, reducing block diversity.

4. MIPS Architecture and Performance
   a. Technically, branch and jump could write to a register when linking, but for the majority they don't.

(a) Consider a simple, non-pipelined, *single-cycle* implementation of MIPS. Assume instruction and data memory accesses take the same amount of time. The operation time for the major functional units for this machine are as follows:

- Memory units: 6 ns
- ALU and adders: 4 ns
- Register file (read or write): 2 ns

Mark the entries in the following table to indicate the stages of the critical path taken by the following instruction types. [5 points]

Instruction Type	Functional Units used by the Instruction Type				
	Instruction Fetch	Register Read	ALU	Data Access	Register Write
R-Format (ALU)	X	X	X		X
Load Word	X	X	X	X	X
Store Word	X	X	X	X	X
Branch	X	X	X		
Jump	X				

   b. With instruction fetch taking 6ns, register read taking 2ns, the ALU taking 4ns, data access taking 6ns, and register write taking 2ns, each clock cycle will take 6+2+4+6+2=20 ns
   c. Pipelined and Multi-Cycle processing are similar in that they use one cycle per instruction step. However, pipelined processors can execute multiple instructions faster than a multi-cycle processor can. There will be a significant amount of stalls in the pipelined processor due to the loads, but with forwarding, each ALU instruction can be executed one after another. Multi-cycle processors will have a much higher CPI when running multiple instructions (Especially of the same type) in a row.

d. Labeling wasn't very clear. I assumed a blank space for no forwarding.

ID: 212265053 CprE 381 S12 Final Exam 10

(d) In the code below, label each instruction with the forwarding paths it requires (1, 2, 3, 4: as labeled on the next page of this exam). [5 points] Labeling isn't clear. Assuming blank for no forward.

```
loop: sub   $t3, $t3, 1
      add   $t0, $t0, 4
      add   $t1, $a0, $t0      3
      lw    $t2, 0($t1)        2
      add   $t2, $t2, 1        1
      sw    $t2, 0($t1)        2
      bgt   $t3, $zero, loop
```

(e) Re-write the code from part (d) for optimal performance. In addition, try to minimize the number of required forwarding paths and explain which forwarding paths are still required. [5 points]

e. Unfortunately, the way this code is written, it cannot be changed lest the function itself is changed. Therefore, all forwarding is needed, and one stall is required.

5. Cache Construction and Behavior
   a. The index would be 5 bits (for 256 / 8 = 32 blocks), the block offset would be 3 bits (8 bytes), and the tag would be the remainder of 24 bits (32 – 5 – 3 = 24).
   b. The bits needed to implement such a cache are as follows: 1 bit for the LRU, 1 bit for "dirty" Boolean, 1 bit for a valid Boolean, 32 bits for the data, and 24 bits for the tag. Since it's a 2-way set associative cache, double this sum to get the final result. Therefore, the total number of bits required for storage is (1 + 1 + 1 + 32 + 24) * 2 = 118 bits.
   c. Each element of each array is 4 bytes long. The cache has an 8-byte block size. Each array has 32 elements, and the cache has 32 blocks. The cache can easily store one entire array (i.e. Block 1: x[0] & x[1], Block 2: x[3] & x[4]…), and since it's a 2-way set associative cache, it can easily store all three arrays. X and Y could be in the first set, while Z gets put into the second. One array would only take up 16 blocks, so there will even be an extra 16 blocks left over. After the initial compulsory misses, this code should have a hit rate of 100%.
   d. The cache has room for 32 bytes of data, judging from the sudden spike in time from a length of 32 to 64 (at least with a stride of 1). Seeing how the time also increases when the stride is less than Length / 4, I would say that the block size is 4 bytes. I'd wager that this cache is direct-mapped as well, judging from the fact that the time doubles after 32 length in the 1 stride column, suggesting that a lot of conflict misses were happening. A hit time for this cache is about 7 ns, and a miss time would be around 45 ns.