Willis Allstead
4/29/18
CS 219

HW11

1.

a. **PURPOSE: I am trying to understand how a 16-bit memory address is divided up into the different parts: line number, tag number, byte number.**

Main memory of 2^16 bytes means the memory address length is 16 bits. Since the block size is 8 bytes, or 2^3 bytes, that means the byte number is 3. A direct mapped cache mechanism with 32 cache lines, or 2^5 lines means the line number is 5 bits. The rest of the bits are for the tag.

b. **PURPOSE: I am trying to learn where different addressed bytes are stored in which lines in memory.**

0001 0001 0001 1011 -> cache line = 00011 or the 3rd line.
1100 0011 0011 0100 -> cache line = 00110 or the 6th line.
1101 0000 0001 1101 -> cache line = 00011 or the 3rd line.
1010 1010 1010 1010 -> cache line = 10101 or the 21st line.

c. **PURPOSE: I am trying to learn what bytes are stored along with a byte stored in cache memory. This might be due to what professor Egbert talked about, how code is sequential so sequential memory is important to cache.**

If 0001 1010 0001 1010 is stored in the cache then bytes with addresses
0001 1010 0001 1000 -> 0001 1010 0001 1111 are also stored with it for efficiency purposes.

d. **PURPOSE: I am trying to learn how many bytes can be stored in the cache maximally.**

Since the tag line was 8 in this example, 2^8 = 256 which means 256 bytes of memory can be stored in the cache.

e. **PURPOSE: I am trying to learn the actual reason that the tag is stored in the cache.**

A location in the cache does not store just a single item. Memory to cache address mapping is not one-one. Because of this, the tag is used to differentiate between items in the cache when accessing it.

2.

a. **PURPOSE: Given direct mapping, tag size, line size, and address size, I am trying to learn how an address is formatted and how to calculate the number of addressable units, number of blocks in main memory, number of lines in the cache, and the tag size.**

With direct mapping given the tag field in the address is 20 bits,

Since the cache uses a 64-byte line size,
64 = 2^r => r = 6 bits
Since the total address bits is 32,
32 = tag bits + line bits + word bits
32 = 20 + 6 + w
w (word bits) = 6 bits

number of addressable units = 2^(s+w)
tag bits = s-r
20 = s – 6 => s = 20 + 6 = 26
number of addressable units = 2^(26+6) = 2^32 bytes

number of blocks in main memory = number of addressable units/(2^w)
2^w = 2^6
number of blocks in main memory = (2^32)/(2^6) = 2^26 bytes

number of lines in cache = 2^r = 2^6 = 64 lines

size of tag (given) = 20 bytes.

FORMAT: 20 tag bits | 6 line bits | 6 word bits

b. PURPOSE: Given associative cache, line size, and address size, I am trying to learn how an address is formatted and how to calculate the number of addressable units, number of blocks in main memory, number of lines in the cache, and the tag size.

With associative cache,
Since the total address bits is 32,
32 = tag bits + word bits
32 = tag bits + 6
tag bits = 26

number of addressable units = 2^(s+w) = 2^(26+6) = 2^32 bytes

number of blocks in main memory = number of addressable units/(2^w)
2^w = 2^6
number of blocks in main memory = (2^32)/(2^6) = 2^26 bytes

number of lines in cache can't be determined with associative memory.

Size of tag bits = 26 bytes

FORMAT: 26 tag bits | 6 word bits

c. PURPOSE: Given 4-way set associative cache with 9 bit tag field, line size, and address size, I am trying to learn how an address is formatted and how to calculate the number of addressable units, number of blocks in main memory, number of lines in the cache, and the tag size.

Tag bits = 9
Word bits = 6
Total address bits = tag bits + set bits + word bits
32 = 9 + set bits + 6
set bits = 17

number of addressable units = 2^(s+w+r)
= 2^(17+9+6) = 2^32 bytes

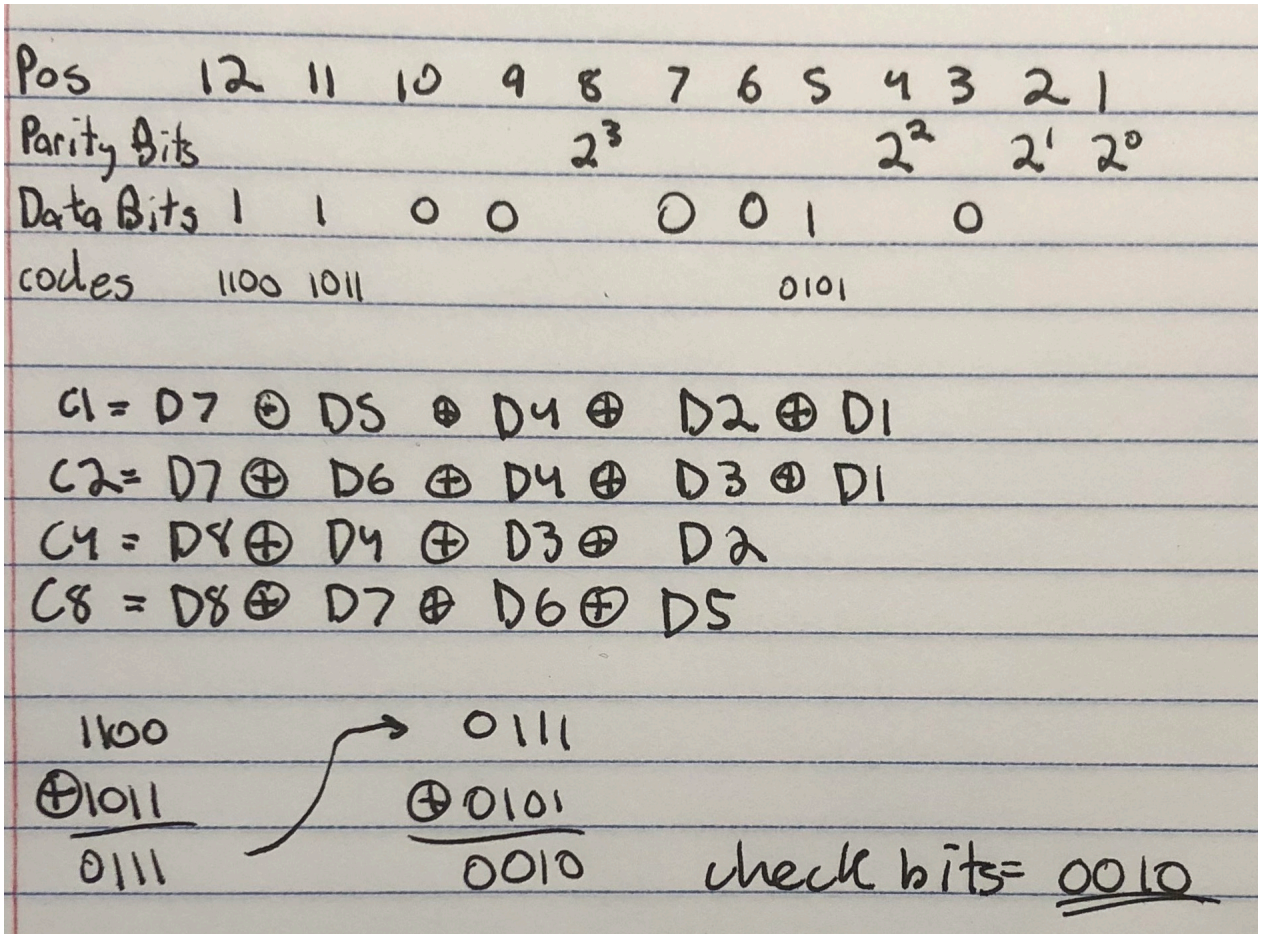
number of blocks of main memory = 2^(s+w) = 2^(17+9) = 2^26 bytes

k = 4 for 4-way set associative
number of sets in cache = 2^d
number of lines in cache = k * 2^d
= 4 * 2^17 = 2^19

size of tag bits = 9

FORMAT: 9 tag bits | 17 line bits | 6 word bits

3. **PURPOSE: I am trying to learn how to use the Hamming algorithm to determine which check bits would be stored in memory along with the data word.**



4. **PURPOSE: I am trying to learn how to use the Hamming algorithm to determine which bits have an error, and how to correct those bits read from memory.**