1. Fully associative (FA) caches
    a. Any block of main memory can go into any line
    b. W = LIC, S = 1
        i. Entire cache is a whole set
        ii. Any line of cache can go into any way of that set
        iii. Number of set bits are always 0
    c. Won't have conflicts like with a DM cache
        i. Any line can go anywhere
    d. However, extremely expensive to implement in terms of both power and money
        i. Must check all tags at once to see if data is in cache
    e. Example
        i. Same cache parameters as before, except now a FA cache
        ii. 8-byte FA cache with line size of 2, and 4-bit address
        iii. LIC = C / LS = 8 / 2 = 4
        iv. For a FA cache, S = 1

Tag	Set	Offset		Address Bits
4 – 1 – 0 = 3 bits	$\log_2 S = \log_2 1 = 0$ bits	$\log_2 LS = \log_2 2 = 1$ bit	=	4 bits

2. FA cache mapping example *(not covered in lecture)*
    a. Use the same cache specifications and address format we calculated in the previous problem
    b. Use same addresses and data from DM cache example previously

Address	Data	Address	Data
0110	0x1B	0111	0x59
1000	0xFE	1001	0x3D
1010	0x25	1011	0x98
1110	0x0C	1111	0x3A

    c. Load addresses into cache in same fashion
        a. However, this time there's no set bits
        b. Any line of cache can go anywhere, as we see in the next table

Memory Access	Tag	Offset	Access Type	Data	Hit or Miss
0110	011	0	Read		Miss
1000	100	0	Read		Miss
1110	111	0	Read		Miss
1010	101	0	Write	0xBE	Miss
1011	101	1	Write	0xEF	Hit

d. Place values from first three accesses into table below
   i. First three accesses fill up first three previously invalid lines of cache
   ii. Each one is a miss, but they bring in the entire line
   iii. No conflict between 0110 and 1110 this time
   iv. Dirty bits are all 0 for same reason as before

Line in Cache	Tag	Byte 0	Byte 1	Valid	Dirty
00	011	1B	59	~~0~~ 1	~~0~~ 0
01	100	FE	3D	~~0~~ 1	~~0~~ 0
10	111	0C	3A	~~0~~ 1	~~0~~ 0
11				0	0

e. Now we introduce writes
   i. Write changes the value in the cache
   ii. The first write misses, so we must bring the original values in from memory
      1. Known as *write-allocate* policy
      2. Once we bring it in, *then* we modify the 0th byte
      3. Set dirty bit for this line since we modified the value from memory
   iii. Second write hits, so we don't need to bring the line in
      1. Change the 1st byte to the new value
      2. Dirty bit stays 1 here

Line in Cache	Tag	Byte 0	Byte 1	Valid	Dirty
00	011	1B	59	~~0~~ 1	~~0~~ 0
01	100	FE	3D	~~0~~ 1	~~0~~ 0
10	111	0C	3A	~~0~~ 1	~~0~~ 0
11	101	~~25~~ BE	~~98~~ EF	~~0~~ 1	~~0~~ 1

3. Set associative (SA) caches
   a. Compromise between DM and FA
   b. N-way SA cache means there's N lines in each set
      i. Thus, there's N different places a line can go into
      ii. W = N(-way)
   c. Advantages and disadvantages of both DM and FA
      i. More options to place lines, so less conflicts than a DM cache
      ii. Will still have conflicts compared to a FA cache
      iii. More expensive in power and cost than a DM cache
      iv. Cheaper and less power than a FA cache
   d. Example
      i. Same cache parameters as before, except now a 2-way SA cache
      ii. 8-byte 2-way SA cache with line size of 2, and 4-bit address
      iii. LIC = C / LS = 8 / 2 = 4
      iv. For a SA cache, S = LIC / W = 4 / 2 = 2

Tag	Set	Offset		Address Bits
4 – 2 – 0 = 2 bits	$\log_2 S = \log_2 2 = 1$ bit	$\log_2 LS = \log_2 2 = 1$ bit	=	4 bits

4. SA cache mapping example *(not covered in lecture)*
    a. Use the same cache specifications and address format we calculated in the previous problem
    b. Use same addresses and data from previous two examples

Address	Data	Address	Data
0110	0x1B	0111	0x59
1000	0xFE	1001	0x3D
1010	0x25	1011	0x98
1110	0x0C	1111	0x3A

    c. Load addresses into cache into same fashion
        i. One set bit this time

Memory Access	Tag	Set	Offset	Access Type	Data	Hit or Miss
0110	01	1	0	Read		Miss
1000	10	0	0	Read		Miss
1110	11	1	0	Read		Miss
1010	10	1	0	Read		Miss
1110	11	1	0	Write	DE	Hit

    d. Place values from accesses into table below
        i. Table stops right before the fourth read
        ii. How do we know which line to evict in set 1?

Set	Line in Cache	Tag	Byte 0	Byte 1	Valid	Dirty
0	0	10	FE	3D	~~0~~ 1	~~0~~ 0
	1				0	0
1	2	01	1B	59	~~0~~ 1	~~0~~ 0
	3	11	0C	3A	~~0~~ 1	~~0~~ 0

5. Cache replacement algorithms
    b. How do we evict lines from a given set for non-DM caches?
        i. Why do we not need a replacement policy for DM caches?
    c. Least recently used (LRU)
        i. Whichever line that has gone the longest without being touched is evicted
        ii. 2-way cache example: when we touch a line, set its *use* bit and clear the other line's bit
    d. First in, first out (FIFO)
        i. Evict whichever line has been in the cache the longest
        ii. Use a queue to implement
    e. Least frequently used (LFU)
        i. Add a counter to each line
        ii. Evict whichever line has the fewest accesses
    f. Random
        i. Works almost as well as any of the others

6. Finishing the SA cache mapping example *(not covered in lecture)*
    a. Will assume LRU for our previous example
        i. Have added correct use bit history below
        ii. Any access sets that line's use bit to 1 and the other line's bit to 0
            1. Set 1's use bits switch twice with the second and third accesses

Set	Line in Cache	Tag	Byte 0	Byte 1	Valid	Dirty	Use
0	0	10	FE	3D	~~0~~ 1	~~0~~ 0	~~0~~ 1
	1				0	0	~~0~~ 0
1	2	01	1B	59	~~0~~ 1	~~0~~ 0	~~0~~ ~~1~~ 0
	3	11	0C	3A	~~0~~ 1	~~0~~ 0	~~0~~ ~~0~~ 1

    b. Fourth read needs to evict a line in set 1
        i. Choose one with use bit 0
            2. Was the first line we brought in, so was "least recently used"
        ii. After bringing in new line, set its use bit to 1 since we just accessed it
            3. Change other line's use bit to 0

Set	Line in Cache	Tag	Byte 0	Byte 1	Valid	Dirty	Use
0	0	10	FE	3D	~~0~~ 1	~~0~~ 0	~~0~~ 1
	1				0	0	~~0~~ 0
1	2	~~01~~	~~1B~~	~~59~~	~~0~~ ~~1~~	~~0~~ ~~0~~	~~0~~ ~~1~~ ~~0~~
		10	25	98	1	0	1
	3	11	0C	3A	~~0~~ 1	~~0~~ 0	~~0~~ ~~0~~ ~~1~~ 0

    c. Fifth write is a hit
        i. Set dirty bit appropriately
        ii. Use bit switches again

Set	Line in Cache	Tag	Byte 0	Byte 1	Valid	Dirty	Use
0	0	10	FE	3D	~~0~~ 1	~~0~~ 0	~~0~~ 1
	1				0	0	~~0~~ 0
1	2	~~01~~	~~1B~~	~~59~~	~~0~~ ~~1~~	~~0~~ ~~0~~	~~0~~ ~~1~~ ~~0~~
		10	25	98	1	0	~~1~~ 0
	3	11	~~0C~~ DE	3A	~~0~~ 1	~~0~~ ~~0~~ 1	~~0~~ ~~0~~ ~~1~~ ~~0~~ 1

7. Cache write policies
    a. If we change values that reside in main memory, we make changes in cache first
    b. What happens when we need to evict a line?
    c. Write-back
        i. Only update RAM if evicted line's dirty bit is set
        ii. Dirty bit set if CPU had to update value of line in cache
        iii. Does not require writing to RAM nearly as often
            1. If we don't write to a line, then evict it, don't need to update main memory
        iv. However, RAM value doesn't always match the updated cache value anymore
            1. Other devices that need the most up-to-date value must go through cache now
    d. Write-through
        i. All write operations are made to main memory as well as cache
        ii. Every time a value is changed, must update RAM as well
        iii. Requires writing to RAM extremely often
        iv. Write-through does *not* use a dirty bit