ECE 3058
Architecture, Systems, Concurrency, and Energy in Computation
Lab 3

Part A: Caches

Direct-mapped Cache

The following diagram shows how a direct-mapped cache is organized. To read a word from the cache, the input address is set by the processor. Then the index portion of the address is decoded to access the proper row in the tag memory array and in the data memory array. The selected tag is compared to the tag portion of the input address to determine if the access is a hit or not. At the same time, the corresponding cache block is read and the proper line is selected through a MUX.

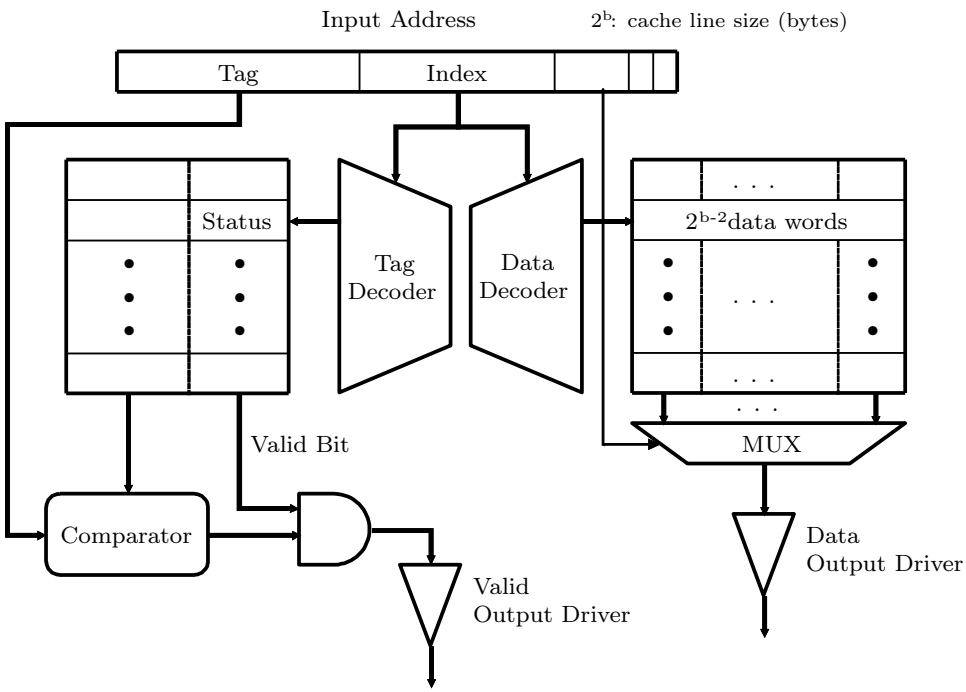

Figure A.1: A direct-mapped cache implementation

In the tag and data array, each row corresponds to a line in the cache. For example, a row in the tag memory array contains one tag and two status bits (valid and dirty) for the cache line. For direct-mapped caches, a row in the data array holds one cache line.

Four-way Set-associative Cache

The implementation of a 4-way set-associative cache is shown in the following diagram. (An n-way set-associative cache can be implemented in a similar manner.) The index part of the input address is used to find the proper row in the data memory array and the tag memory array. In this case, however, each row (set) corresponds to four cache lines (four ways). A row in the data memory holds four cache lines (for 32-bytes cache lines, 128 bytes), and a row in the tag memory array contains four tags and status bits for those tags (2 bits per cache line). The tag memory and the data memory are accessed in parallel, but the output data driver is enabled only if there is a cache hit.

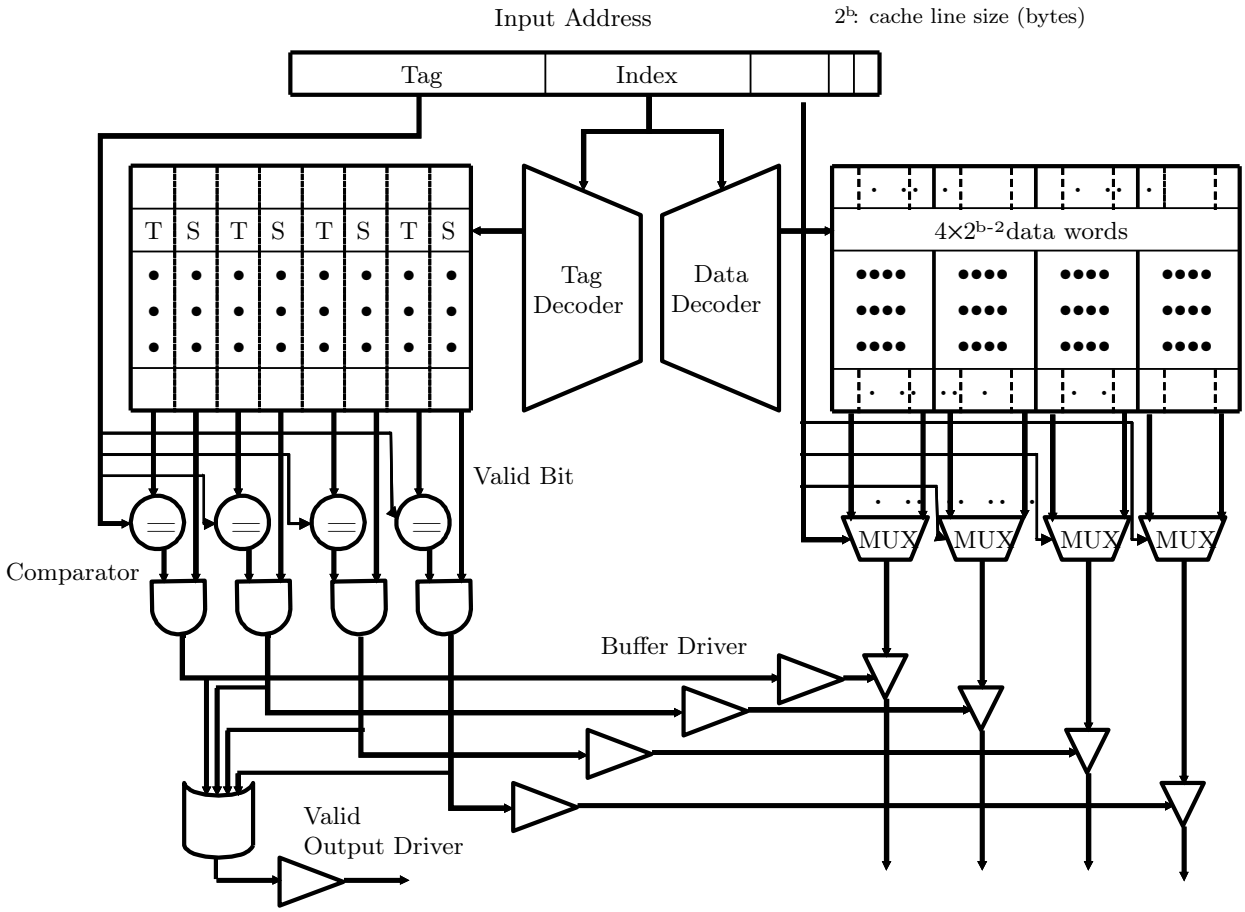

Figure A.2: A 4-way set-associative cache implementation

Problem A.1:
We want to compute the access time of the direct-mapped (DM) cache. Assume a 128-KB cache with 8-word (32-byte) cache lines. The data output is also 32 bits, and the MUX selects one word out of the eight words in a cache line. Using the delay equations given in Table A.1, fill in the column for the direct-mapped (DM) cache in the table. In the equation for the data output driver, 'associativity' refers to the associativity of the cache (1 for direct-mapped caches, A for A-way set-associative caches).

Component	Delay equation (ps)		DM (ps)	SA (ps)
Decoder	200×(# of index bits) + 1000	Tag	3400	3000
		Data	3400	3000
Memory array	200×$\log_2$(# of rows) +	Tag	4217.5	4249.5
	200×$\log_2$(# of bits in a row) + 1000	Data	5000	5000
Comparator	200×(# of tag bits) + 1000		4000	4400
N-to-1 MUX	500×$\log_2$N + 1000		2500	2500
Buffer driver	2000			2000
Data output driver	500×(associativity) + 1000		1500	3000
Valid output driver	1000		1000	1000

DM
- Index bits: 12
- Tag bits: 15
- Data rows: 32*8 = 256
- Tag bits in a row: 15+2 = 17

SA (N = 4)
- Index bits: 10
- Tag bits: 17
- Data rows: 32*8*4 = 1024
- Tag bits in a row: (17+2) * 4 = 76

Table A.1: Delay of each Cache Component

What is the critical path of this direct-mapped cache for a cache read? What is the access time of the cache (the delay of the critical path)? To compute the access time, assume that a 2-input gate (AND, OR) delay is 500 ps. If the CPU clock is 150 MHz, how many CPU cycles does a cache access take?

Critical Path and Time Calculations:

tag decoder + tag array + and gate + VoP = 3400 + 4217.5 + 500 + 1000 = 9117.5 ps
tag decoder + Comparator + tag array + and gate + vop = 3400 + 4000 + 4217.5 + 500 + 1000 = 13,117.5 ps
data decoder + data array + mux + dop = 3400 + 5000 + 2500 + 1500 = 12400 ps

Critical Path: Tag decoder --> Tag Array --> Comparator --> AND gate --> Valid output driver
Access Time: 13,117.5 ps

CPU cycles = access time / clock time = access time * frequency = 13117.5 ps * 150 MHz
= 13117.5 * 10^-12 * 150 * 10^6 = 1.967625 = 1.968 CPU = 2 CPU cycles approx

Problem A.2:

Now George P. Burdell is studying the effect of set-associativity on the cache performance. Since he now knows the access time of each configuration, he wants to know the miss-rate of each one. For the miss-rate analysis, George is considering two small caches: a direct-mapped cache with 8 lines with 16 bytes/line, and a 4-way set-associative cache of the same size. For the set-associative cache, George tries out a least recently used (LRU) policy.

George tests the cache by accessing the following sequence of hexadecimal byte addresses, starting with empty caches. For simplicity, assume that the addresses are only 12 bits. Complete the following tables for the direct-mapped cache and the LRU 4-way set-associative cache showing the progression of cache contents as accesses occur (in the tables, 'inv' = invalid, and the column of a particular cache line contains the {tag,index} contents of that line). You only need to fill in elements in the table when a value changes.

D-map									
	line in cache								hit?
Address	L0	L1	L2	L3	L4	L5	L6	L7	
110	inv	11	inv	inv	inv	inv	inv	inv	no
136				13					no
202	20								no
1A3			1A						no
102	10								no
361							36		no
204	20								no
114									yes
1A4									yes
177								17	no
301	30								no
206	20								no
135									yes

D-map	0.77 = 77% miss rate
Total Misses	10
Total Accesses	13

4-way									LRU
Address	line in cache hit?								hit?
	Set 0				Set 1				
	way0	way1	way2	way3	way0	way1	way2	way3	
110	inv	inv	inv	inv	11	inv	inv	inv	no
136					11	13			no
202	20								no
1A3		1a							no
102			10						no
361				36					no
204									yes
114									yes
1A4									yes
177							17		no
301			30						no
206									yes
135									yes

4-way LRU	0.61 = 61% miss rate
Total Misses	8
Total Accesses	13