Name _____

# Computer Architecture and Engineering
# **CS152 Quiz #2**
## March 5th, 2013
## Professor Krste Asanović

## Name: _____<ANSWER KEY>_____

This is a closed book, closed notes exam.
80 Minutes
16 pages

Notes:

- Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.
- Please carefully state any assumptions you make.
- Please write your name on every page in the quiz.
- You must not discuss a quiz's contents with other students who have not taken the quiz. If you have inadvertently been exposed to a quiz prior to taking it, you must tell the instructor or TA.
- You will get no credit for selecting multiple-choice answers without giving explanations if the instructions ask you to explain your choice.

| | | |
|---|---|---|
| Writing name on each sheet | _____ | 1 Point |
| Question 1 | _____ | 25 Points |
| Question 2 | _____ | 26 Points |
| Question 3 | _____ | 12 Points |
| Question 4 | _____ | 16 Points |
| TOTAL | _____ | 80 Points |

Name _____

# Question 1: Sub-blocked Cache [25 points]

In this problem we analyze two different proposals for a cache to be used in a processor. We care about both the cycle time (which is limited by the cache hit access time) and the miss rate.

The first proposal is to use a regular direct-mapped cache (see Appendix A). The second design is to use a sub-blocked cache (see below). A sub-blocked cache reduces the tag overhead by increasing the line size, but adds an individual valid bit to every sub-block to avoid increasing the miss penalty. The extra valid bits are stored in the tag array (more specifically in the status bits). Both caches are write back caches with write allocate.

The baseline cache (i.e., without sub-blocking) is a direct-mapped 32KB data cache with 32-byte lines. We will consider the impact of sub-blocking by keeping the same capacity but increasing the cache line size to 64-bytes with an individual valid bit on every 32-byte sub-block. Addresses are 32 bits long. Figure 1, below, show the modified hardware of the sub-blocked cache.

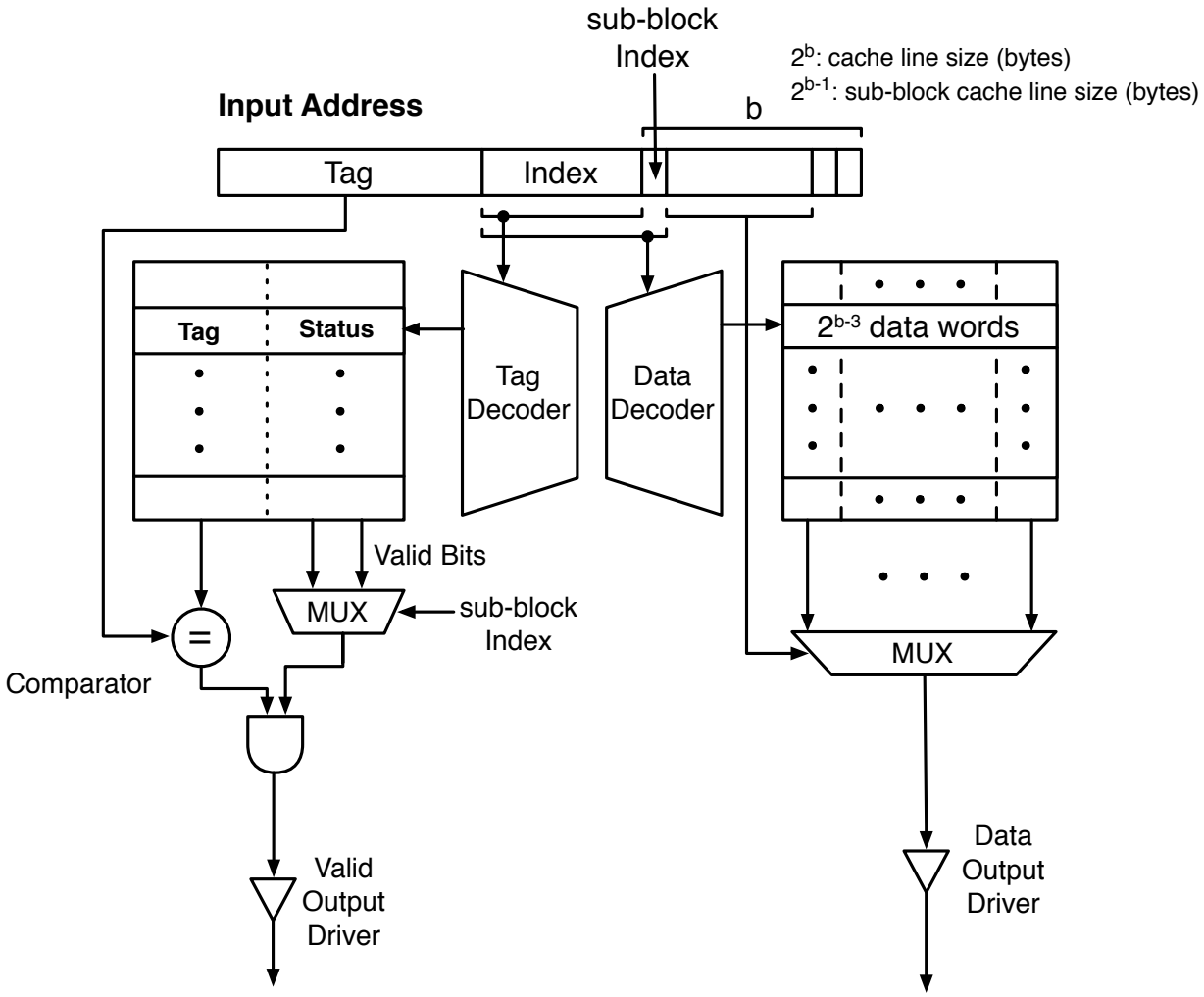


Figure 1: A sub-blocked cache implementation

Name _____

**Q1.A Three C's of Cache Misses [5 points]**
State which of the 3C's of cache misses are affected by sub-blocking the cache, and how. How will the cache miss rate compare to the baseline? Explain your reasoning to receive credit.

Conflict misses can increase, as sub-blocking reduces the number of sets (when the capacity is kept the same). Compulsory, and Capacity misses don't change.

**Q1.B Cache Parameters [4 points]**
Fill out the following table showing how many bits are in each of the caches.

| | Baseline | Sub-blocked |
|---|---|---|
| # tag array bits | (17+1+1) * 2^10 = 19Kbits<br>-0.5 if missed dirty bit<br>-0.5 if you didn't multiply 2^10 | (17+2+1) * 2^9 = 10Kbits<br>-0.5 if missed dirty bit<br>-0.5 if you didn't multiply 2^9 |
| # data array bits | 32*8*2^10 = 256Kbits<br>-0.5 if you didn't multiply 2^10 | 64*8*2^9 = 256Kbits<br>-0.5 if you didn't multiply 2^9 |

Name _____

**Q1.C Critical Path [7 points]**
Using Figure 1, Figure A-1 (found in Appendix A), determine and explain what the critical path is for both the baseline cache and the sub-blocked cache. Also, determine the cache access time (ns) for both configurations (i.e., the delay through the critical path).

| Component | Delay Equation (in ps) | | Baseline | Sub-blocked |
|---|---|---|---|---|
| Decoder | 30 × (# of index bits) + 50 | Tag | 350 | 320 |
| | | Data | 350 | 350 |
| Memory array | 30 × $log_2$(# of rows) + 20 × ⌊$log_2$(# of bits in a row)⌋ + 50 | Tag | 430 | 400 |
| | | Data | 510 | 510 |
| Comparator | 20 × (# of tag bits) + 100 | | 440 | 440 |
| N-to-1 Mux | 50 × $log_2 N$ + 100 | Tag | N/A | 150 |
| | | Data | 250 | 250 |
| 2-input AND | 50 | | 50 | 50 |
| Data output driver | 50 x (associativity) + 100 | | 150 | 150 |
| Valid output driver | 100 | | 100 | 100 |

Table is worth 5 points. -0.4 for incorrect box.

Baseline cache critical path (in ns): (1 point)
Tag: 350 + 430 + 440 + 50 + 100 = 1370ps = 1.37ns
Data: 350 + 510 + 250 + 150 = 1260ps = 1.26ns
Critical path = 1.37ns
+0.5 if you got the tag critical path right (decoder + memory array + comparator + 2-input-AND + valid output driver)
-0.1 if trivial math error

Sub-blocked cache critical path (in ns): (1 point)
Tag path through comparator: 320 + 400 + 440 + 50 + 100 = 1310ps = 1.31ns
Tag path through valid mux: 320 + 400 + 150 + 50 + 100 = 1020ps = 1.02ns
Data: 350 + 510 + 250 + 150 = 1260ps = 1.26ns
Critical path = 1.31ns
+0.5 if you got the tag critical path right (decoder + memory array + comparator + 2-input-AND + valid output driver)
-0.1 if trivial math error
Some people added the 2-to-1 mux path to the critical path, but the 2-to-1 mux is evaluated parallel with the comparator, so it shouldn't be added to the critical path

**Q1.D AMAT [4 points]**
Temporarily assume that both the baseline cache and the sub-blocked cache have the same hit rate, 75%, and the same average miss penalty, 20ns. Using the cycle times computed in Q1.C as the hit time, compute the average memory access time for both caches.

Baseline cache AMAT (in ns):

AMAT = hit time + miss rate * miss penalty = 1.37 + 0.25 * 20 = 6.37ns

Didn't penalize you if you got Q1.C wrong
-0.5 for trivial math error

Sub-blocked cache AMAT (in ns):

AMAT = hit time + miss rate * miss penalty = 1.31 + 0.25 * 20 = 6.31ns

Didn't penalize you if you got Q1.C wrong
-0.5 for trivial math error

**Q1.E Crossover Point in Performance [5 points]**
If the baseline cache miss rate is 5%, how much greater must the miss rate for the sub-blocked cache be to give worse overall performance as measured by AMAT? Assume the miss penalty is 20ns in both cases.

1.37 + 0.05 * 20 = 1.31 + MP * 20
20MP = 1.06
MP = 0.053

Sub-blocked cache will perform worse than the baseline cache if the miss rate is 0.3% greater.

Didn't penalize you if you got Q1.C wrong
-1 for trivial math error

Name _____

# Question 2: Page Tables [26 points]

For this question, you are going to study a 64-bit RISC-V machine that uses a 4-level page table scheme. Virtual addresses are 64-bits long, but the system requires and enforces that the upper 11-bits of a valid virtual address are always 0. This 64-bit RISC-V machine uses 8KB pages, and hence the page offset is 13-bits in size. A PTE or page-table entry is 8-bytes long, and hence level-1, 2, 3, and 4 page table indices are 10 bits each. Please see Figure 2 and 3 for the virtual address, and page-table entry breakdown. Assume that the operating system is smart enough to only allocate the minimal needed physical memory.

| 11 bits | 10 bits | 10 bits | 10 bits | 10 bits | 13 bits |
|---|---|---|---|---|---|
| All zeros | L1 page index | L2 page index | L3 page index | L4 page index | Page offset |

Figure 2. Virtual Address Breakdown (MSB on the left, LSB on the right)

| 21 bits | 30 bits | 9 bits | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|
| reserved | PPN | reserved | R | W | X | V |

R = read permission bit, W = write permission bit, X = execute permission bit, V = valid bit

Figure 3. Page-table Entry Breakdown (MSB on the left, LSB on the right)

**Q2.A DRAM Size [4 points]**
Given that 30 bits are allocated for the PPN in a PTE, what is the largest amount of DRAM that we can usefully put in a system?

2^30 * 2^13 = 2^43 = 8TB

**Q2.B Time for a TLB Refill [4 points]**
Assume that the AMAT during a page table walk is 2ns. How long (in ns) would it take for a TLB refill?

2ns * 4 = 8ns

To reduce the TLB refill time, let's assume you have decided to implement a hashed page table.

On a TLB miss, the first hash function is used to map a VPN to a hash table slot that has 8 PTE's <VPN, PPN> that are searched in parallel. If the first hash probe fails, a second hash function is used to look in another hash table slot. If both hash probes fail, a backup page table is read using a full hierarchical page table walk as before. Figure 4 and 5 summarizes this process:

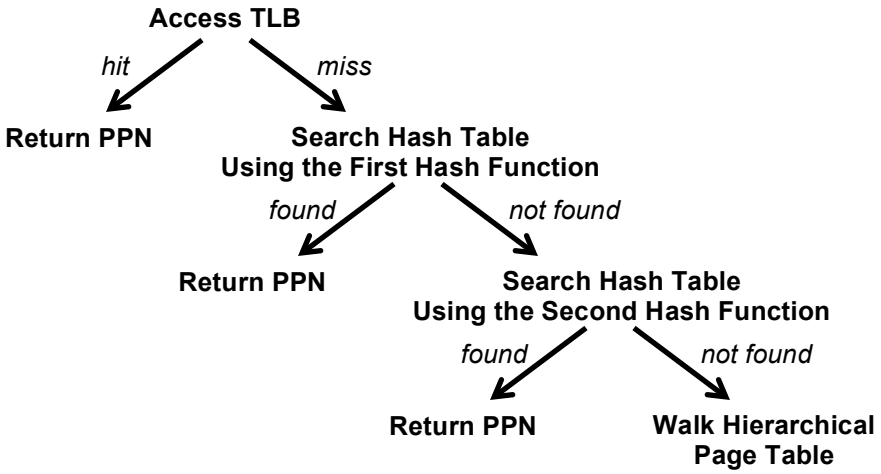


Figure 4: Walking a Hashed Page Table

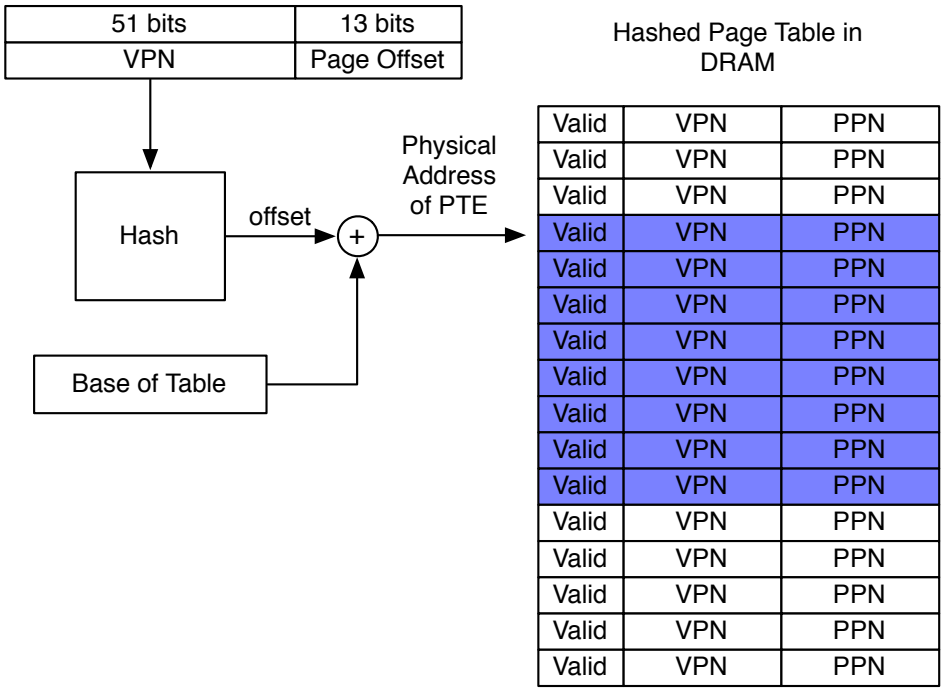


Figure 5: Searching a Hashed Page Table

Assume the following: 1) the first hash probe has a 95% hit rate, 2) the time to search a hashed page table is dominated by the memory access time, 3) AMAT of a hash table probe (fetching all 8 slot entries) is 4ns, 4) AMAT during a hierarchical page table walk is 100ns, since portions of the backup page table are likely to be swapped out to disk.

**Q2.C Hashed Page Tables [8 points]**
Given the assumptions above, calculate the minimum hit rate of the second hash probe required for the hashed page table to improve TLB refill time compared to the hierarchical page table in Q2.B.

$4 + 0.05 * (4 + (1\text{-}HR) * 400) < 8$
$4 + 0.2 + 20 – 20*HR < 8$
$20*HR > 16.2$
$HR > 0.81$ or 81%

-2 if used 100ns rather than 400ns
-4 if you came up with a slightly wrong formula

**Q2.D Overheads of Hierarchical Page Table [5 points]**
Assume we have 100 processes running, and 4GB of physical memory. Calculate the minimum physical memory required to hold the hierarchical page tables needed to map contiguous 4GB virtual memory spaces for all 100 processes.

To hold 4GB of virtual memory, you need 2^32/2^13 = 2^19 pages.
To hold 2^19 pages, you need 2^19/2^10 = 2^9 L4 page tables.
To hold 2^9 L4 page tables, you need 1 L3 page table.
To hold 1 L3 page table, you need 1 L2 page table.
To hold 1 L2 page table, you need 1 L1 page table.

You need 2^9 + 3 pages to map 4GB of virtual memory space per process.

Total amount of physical memory to hold all hierarchical page tables for 100 processes
= 100 * (2^9 + 3) * 8KB
= 100 * 515 * 8KB
= 412 MB

+2 if you got # of L4 page tables correct
+1 if you got # of L3 page tables correct
+1 if you got # of L2 page tables correct
+1 if you got # of L1 page tables correct

**Q2.E Overheads of Hashed Page Table [5 points]**
To reduce hash table conflicts, the hash table is sized with 4 times more PTEs than the number of physical pages. Each entry in the hash table is 16 bytes long. Assume we have 100 processes, and 4GB of physical memory. Calculate the physical memory required for the hashed page table (ignore the backup hierarchical page table).

Total number of physical pages = 2^32/2^13 = 2^19 pages.
# of PTEs in hash table = 2^19 * 4 = 2^21 PTEs.

Total amount of physical memory to hold hash table
= 2^21 PTEs * 16 bytes/PTE
= 2^25 bytes
= 32 MB

Name _____

# Question 3: AMAT [12 points]

Mark whether the following modifications to an L1 cache will cause each of the categories to **increase**, **decrease**, or whether the modification will have **no effect**. You can assume the baseline cache is direct-mapped and that all other cache parameters remain unchanged in each case. **Explain your reasoning** to receive credit.

| | | L1 Hit Time | L1 Miss Rate | L1 Miss Penalty |
|---|---|---|---|---|
| A | Add a sub-blocking scheme (Divide line into sub-blocks with a valid bit for each sub-block) (***line size constant***) | OK - Same, as can forward data value from same size data array, with<br><br>tag/valid check off critical path, as cache is direct mapped.<br><br>OK (not as good as above, but no deduction) - Increases very slightly,<br><br>as need to also check valid bit, and tag array is slightly larger. | Increases, as less prefetching of data. | Reduces, as less data brought in on miss. |
| B | Add an L2 cache | Stays the same, as L1 unchanged. | Stays the same, as L1 unchanged. | Reduces, due to hits in L2. |

| | | | | |
|---|---|---|---|---|
| C | Add a victim cache<br><br>(All parts had to have consistent view of whether VC considered part of L1 or not. Graded using view that gave maximum points.) | If VC considered part of L1 (usual answer), hit time increases as hits in victim cache are slower than hits in regular cache. [ -0.5 is said hit time unchanged but considered VC part of L1 ]<br><br>If VC not considered part of L1:<br>L1 hit time unchanged. | If VC considered part of L1, miss rate drops due to victim cache providing effectively more associativity.<br><br>If VC not considered part of L1:<br>L1 miss rate unchanged. | If VC considered part of L1, unchanged (VC access penalty folded into hit time).<br><br>If VC not considered part of L1:<br>L1 miss penalty reduced due to hits in VC. OK to say miss penalty stays the same, if miss fetched in parallel with checking victim cache. |
| D | Add hardware prefetching | Unchanged. | Reduced, as data prefetched into L1. (Assume prefetching beneficial.) | OK - Remains the same.<br><br>OK - to say reduced, if might have started prefetch of line before demand miss.<br><br>OK - to say increase, if additional prefetch traffic slows demand fetches. |

Name _____

# Question 4: Code Optimizations [16 points]

Will the following transformations to C code **increase**, or **decrease** performance? Assume all arrays are 4-byte integer arrays. **Carefully explain how the transformation impacts performance** to receive credit.

**Q4.A [4 points]**

**before:**

```
for (i=0; i<M; i++) {
   for (j=0; j<N; j++) {
       x[i*N+j] = 2 * x[i*N+j];
   }
}
```

**after:**

```
for (j=0; j<N; j++) {
   for (i=0; i<M; i++) {
       x[i*N+j] = 2 * x[i*N+j];
   }
}
```

**Decrease:** Unit strided loads and stores are transformed to strided loads and stores. This transformation reduces spatial locality.

**Q4.B [4 points]**

**before:**

```
for (i=0; i<N; i++)
   a[i] = b[i] * c[i];
for (i=0; i<N; i++)
   d[i] = a[i] * c[i];
```

**after:**

```
for (i=0; i<N; i++) {
   a[i] = b[i] * c[i];
   d[i] = a[i] * c[i];
}
```

**Increase:** Loop fusion increases temporal locality (accessing array a and c).

Name _____

**Q4.C [4 points]**

**before:**

```
for (i=0; i<N; i++)
   for (j=0; j<N; j++)
       B[j*N+i] = A[i*N+j];
```

**after:**

```
int BS=8;
for (i=0; i<N; i+=BS)
   for (j=0; j<N; j++)
       for (k=0; k<BS; k++)
           B[j*N+(i+k)] = A[(i+k)*N+j];
```

**Increase:** Cache blocking/tiling effectively reduces the working set size. In other words, cache blocking/tiling ensures that cache lines are reused before getting evicted.

**Q4.D [4 points]**

**before:**

```
for (i=0; i<N; i++)
   c[i] = a[i] + b[i];
```

**after:**

```
int P=1;
for (i=0; i<N; i++) {
   prefetch(&a[i+P]);
   prefetch(&b[i+P]);
   prefetch(&c[i+P]);
   c[i] = a[i] + b[i];
}
```

**Decrease:** Prefetching too close will likely slow down the execution, as prefetch instructions become pure overhead. Also prefetech instructions take up space in the I$.

**END OF QUIZ**

Name _____

**THIS PAGE WAS INTENTIONALLY LEFT BLANK**

Name _____

## Appendix A. 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.

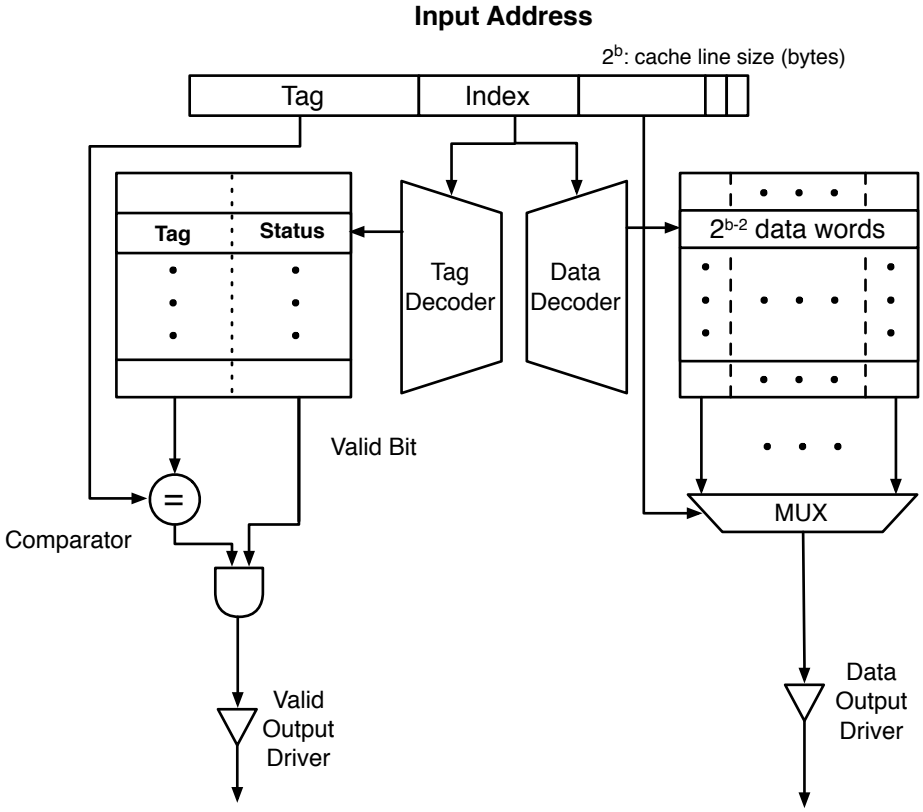


Figure A-1: A direct-mapped cache implementation

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.

**YOU MAY DETACH THIS PAGE**

Name _____

## Appendix B. Two-way Set-associative Cache

The implementation of a 2-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, each row (set) corresponds to two cache lines (two ways). A row in the data memory holds two cache lines (for 32-bytes cache lines, 64 bytes), and a row in the tag memory array contains two 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.

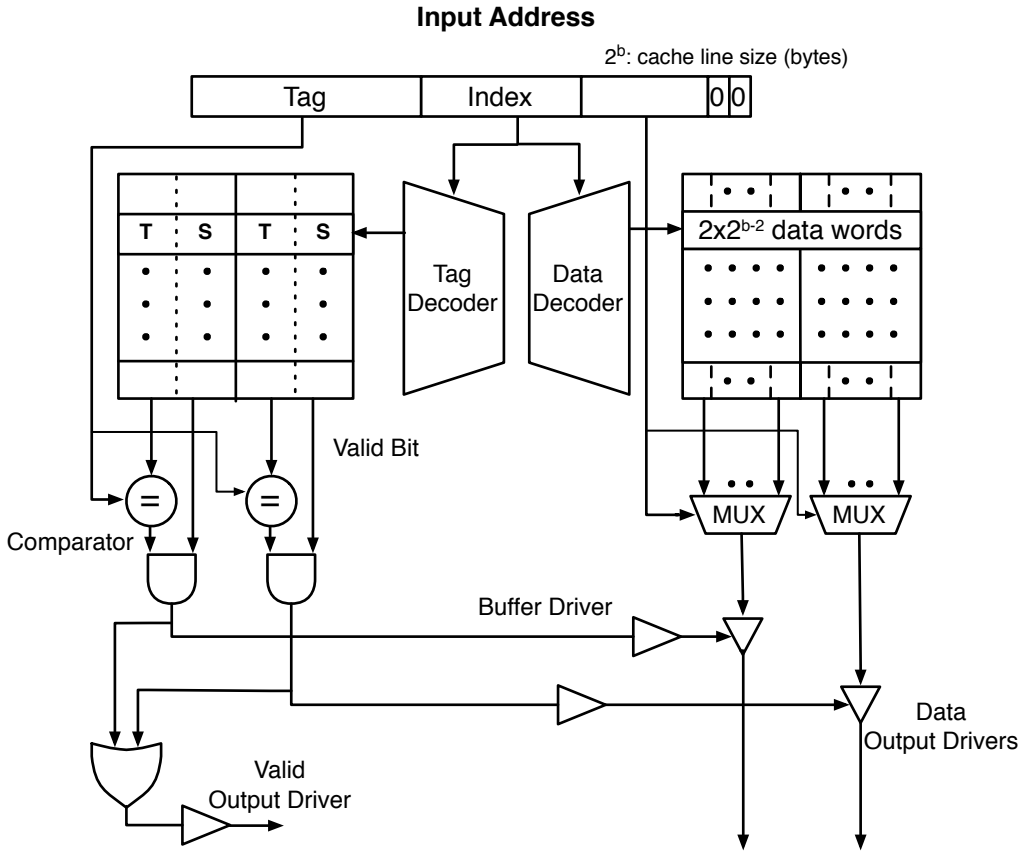


Figure B-1: A 2-way set-associative cache implementation

**YOU MAY DETACH THIS PAGE**