ELEC 464 : MICROCOMPUTER SYSTEM DESIGN
FALL 1995

Cache Memory

Cache memory can greatly increase the processing speed of microprocessors whose CPU cycle time is significantly shorter than the memory access time. Most modern general-purpose microprocessors include some sort of cache memory. In this lecture we will introduce cache memory by looking at the design of a direct-mapped cache. Cache memory is covered in section 7.5 of the text.

Locality of Reference

During the execution of most programs the CPU typically accesses a few memory locations very often. Furthermore, these locations are often close together. As an example, consider the following C function:

```
int strlen(char *p)
{
        int len=0 ;
        while (*p++ != 0 ) len++ ;
        return len ;
}
```

the variables `len` and `p` will be accessed repeatedly[1] as will the instructions within the loop.

To exploit this locality of reference we can use small high-speed memories to temporarily store the values of these often-used memory locations. If access to this *cache* memory can be done more quickly (i.e. with fewer wait states) than access to main memory we can reduce the overall execution time.

The structure of a cached memory system is shown below:

The CPU can read from the cache memory or the main memory and can write to the cache, main memory or both. The main complexity of a cache is designing the controller because it must keep track of which values in the cache are valid and what memory locations they represent.

Effects of Hit Rate and Access Times

If we define k as the ratio of the time required to retrieve a value from cache as opposed to from memory and H as the average fraction of time a memory reference can be obtained from the cache, we can derive the speed improvement factor in memory access time due to the use of cache to be:

$$S = \frac{1}{1 - H(1-k)} \tag{1}$$

Note that H (hit rate) must be reasonably large for k to have an effect. Also note that k will depend not only on the memory access time but also on the length of the bus cycle. For example the bus cycle on a 68000 CPU at 8 MHz is long enough that no wait states are needed for main memory access. This means k is 1 and there is no advantage to using cache memory.

On the other hand consider a 50 MHz RISC processor that has a 20 ns bus cycle when reading from the cache and a main memory bus cycle of 100ns (k=0.2). Assuming the cache size and program structure result in a cache hit rate of 90%, then the use of the cache will speed up the processor by a factor of $1/(1 - 0.9(1 - 0.2)) = 3.6$.

Direct-Mapped Cache

The simplest type of cache memory organization (and the only one we will study) is the direct-mapped cache.

The main and cache memories are organized into *lines* consisting of a small number of bytes (e.g. 8). Lines are read from the cache as a complete unit and when a cache line is updated during a main memory read, the complete line must be updated (e.g. for a 32-bit-wide memory the main memory accesses need to be done in pairs).

A direct-mapped cache that holds 2^N bytes simply uses the least significant $N-3$ (for 8-byte lines)

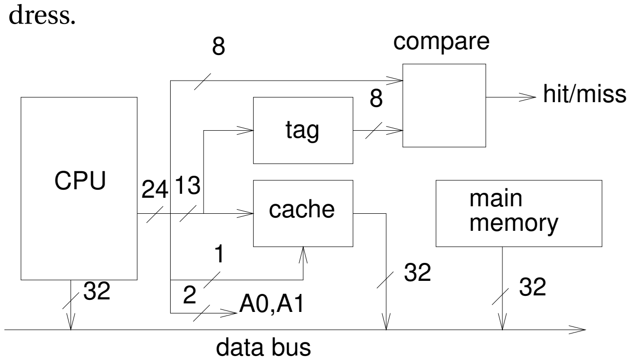
[1] In a proper system this function would be in-lined, and use only the variable 'p' which would be stored in a register, but the principle still applies anyways...

address lines to select a particular line. For example, a 64 kB cache with 8-byte lines would use the CPU address lines A15 to A3 to select a particular line from the cache.

When the CPU does a memory access, the cache control circuitry must decide whether the particular address is stored in the cache memory. If it is, then the CPU reads the cache, otherwise it must read the value from main memory. At the same time as a value is read from the main memory the appropriate cache line is updated.

The cache controller decides whether the particular line in the cache corresponding to that address is the desired one or not by using a lookup table stored in a "tag" RAM. The value stored in each entry in the tag RAM corresponds the remaining most-significant bits of the address that the line was read from.

For example, a 64kB cache with 8-byte lines holds 8k lines. If the cache can support a main memory size of 16 MBytes (24 bits) then the tag RAM must store 8 bits for each line. The 24 address bits going to the cache are divided into 3-bits to select a byte from a line, 13 bits to select a line from the cache and 8 bits that are compared to the tag for that line). The 13 bit line select forms the address input to an 8Kx8 tag RAM and the 8 output bits are compared to the 8 most significant bits of the desired (24-bit) address.



Write-Back versus Write-Through

When the CPU writes a value out the cache controller can either update both the main memory and the cache ("write through") or it can update just the cache ("write-back"). Although the latter is faster it means that we must keep track of cache lines that have been updated in cache but not in main memory. When we get a cache miss on a line we need to write that line back to main memory before it can be re-read from main memory. This means that the cache controller design is more complicated and requires a 1-bit memory (called a "dirty bit" memory, e.g. 8kx1) for each line in the cache.