

7-1

Chapter 7: Memory System Design

Topics

- 7.1 Introduction: The Components of the Memory System
- 7.2 RAM Structure: The Logic Designer's Perspective
- 7.3 Memory Boards and Modules
- 7.4 Two-Level Memory Hierarchy
- 7.5 The Cache
- 7.6 Virtual Memory
- 7.7 The Memory Subsystem in the Computer

1

7-2

Why does a Memory Hierarchy Work?

- **Phenomenon of locality of reference:**
 - **temporal locality** - referenced now, likely to be referenced in the near future
 - **spatial locality** - reference this now, things at neighboring addresses are likely to be referenced soon....

for (k=0; k < n; k++) sum = sum + A[k];

- **temporal** - sum, k, instructions
- **spatial** - elements of A, instructions

2

Introduction

So far, we've treated memory as an array of words limited in size only by the number of address bits. Life is seldom so easy...

Real world issues arise:

- cost
- speed
- size
- power consumption
- volatility

...

In this chapter we will cover—

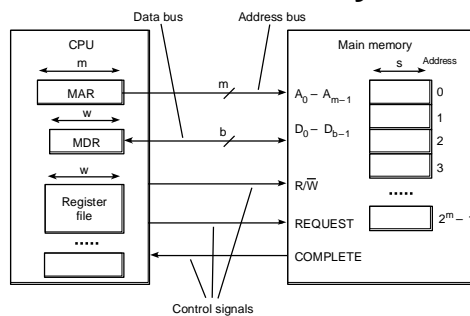
- Memory components:
 - RAM memory cells and cell arrays
 - Static RAM—more expensive, but less complex
 - Tree and matrix decoders—needed for large RAM chips
 - Dynamic RAM—less expensive, but needs “refreshing”
 - Chip organization
 - Timing
 - ROM—Read-only memory
- Memory boards
 - Arrays of chips give more addresses and/or wider words
 - 2-D and 3-D chip arrays
- Memory modules
 - Large systems can benefit by partitioning memory for
 - separate access by system components
 - fast access to multiple words

—more—

In this chapter we will also cover—

- The memory hierarchy: from fast and expensive to slow and cheap
 - Example: Registers → Cache → Main Memory → Disk
 - At first, consider just two adjacent levels in the hierarchy
 - The cache: High speed and expensive
 - Kinds: Direct mapped, associative, set associative
 - Virtual memory—makes the hierarchy transparent
 - Translate the address from CPU's logical address to the physical address where the information is actually stored
 - Memory management—how to move information back and forth
 - Multiprogramming—what to do while we wait
 - The “TLB” helps in speeding the address translation process
- Overall consideration of the memory as a subsystem

Fig 7.1 The CPU–Memory Interface



Sequence of events:

Read:

1. CPU loads MAR, issues Read, and REQUEST
2. Main memory transmits words to MDR
3. Main memory asserts COMPLETE

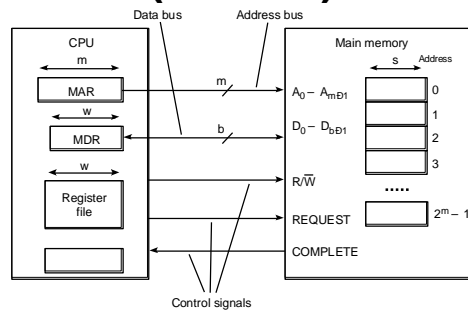
Write:

1. CPU loads MAR and MDR, asserts Write, and REQUEST
2. Value in MDR is written into address in MAR
3. Main memory asserts COMPLETE

—more—

7-7

Fig 7.1 The CPU–Memory Interface (cont'd.)



Additional points:

- If $b < w$, main memory must make w/b b-bit transfers
- Some CPUs allow reading and writing of word sizes $< w$
Example: Intel 8088: $m = 20$, $w = 16$, $s = b = 8$
8- and 16-bit values can be read and written
- If memory is sufficiently fast, or if its response is predictable, then COMPLETE may be omitted
- Some systems use separate R and W lines, and omit REQUEST

7

7-8

Tbl 7.1 Some Memory Properties

| Symbol | Definition | Intel 8088 | Intel 8086 | PowerPC 601 |
|----------------|--------------------------------------|------------------------|------------------------|------------------------|
| w | CPU word size | 16 bits | 16 bits | 64 bits |
| m | Bits in a logical memory address | 20 bits | 20 bits | 32 bits |
| s | Bits in smallest addressable unit | 8 bits | 8 bits | 8 bits |
| b | Data bus size | 8 bits | 16 bits | 64 bits |
| 2^m | Memory word capacity, s -sized wds | 2^{20} words | 2^{20} words | 2^{32} words |
| $2^m \times s$ | Memory bit capacity | $2^{20} \times 8$ bits | $2^{20} \times 8$ bits | $2^{32} \times 8$ bits |

8

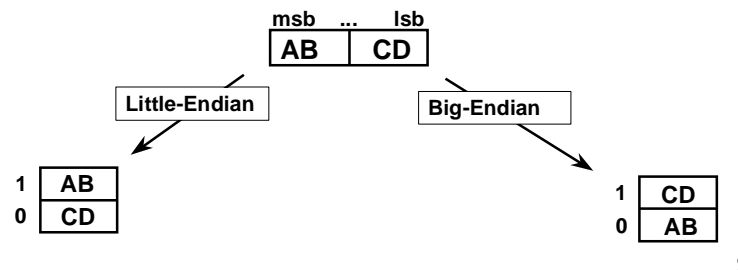
Big-Endian and Little-Endian Storage

When data types having a word size larger than the smallest addressable unit are stored in memory the question arises,

“Is the least significant part of the word stored at the lowest address (*little-Endian, little end first*) or—

is the most significant part of the word stored at the lowest address (*big-Endian, big end first*)”?

Example: The hexadecimal 16-bit number ABCDH, stored at address 0:



Tbl 7.2 Memory Performance Parameters



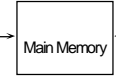
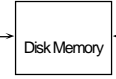

| <u>Symbol</u> | <u>Definition</u> | <u>Units</u> | <u>Meaning</u> |
|---------------------------|-------------------|--------------|---|
| t_a | Access time | time | Time to access a memory word |
| t_c | Cycle time | time | Time from start of access to start of next access |
| k | Block size | words | Number of words per block |
| ω | Bandwidth | words/time | Word transmission rate |
| t_l | Latency | time | Time to access first word of a sequence of words |
| $t_{bl} = t_l + k/\omega$ | Block access time | time | Time to access an entire block of words |

(Information is often stored and moved in blocks at the cache and disk level.)

7-11

Tbl 7.3 The Memory Hierarchy, Cost, and Performance

Some Typical Values:

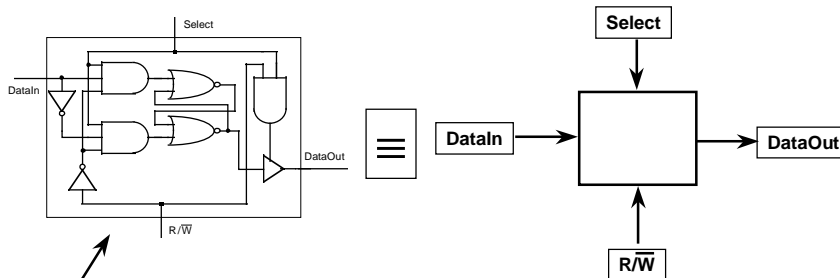
| | | | | | |
|------------------------|---|---|---|--|---|
| Component |  |  |  |  |  |
| Access type | Random access | Random access | Random access | Direct access | Sequential access |
| Capacity, bytes | 64–1024 | 8–512 KB | 8–64 MB | 1–10 GB | 1 TB |
| Latency | 1–10 ns | 20 ns | 50 ns | 10 ms | 10 ms–10 s |
| Block size | 1 word | 16 words | 16 words | 4 KB | 4 KB |
| Bandwidth | System clock rate | 8 MB/s | 1 MB/s | 1 MB/s | 1 MB/s |
| Cost/MB | High | \$500 | \$30 | \$0.25 | \$0.02 |

11

7-12

Fig 7.3 Conceptual Structure of a Memory Cell

Regardless of the technology, all RAM memory cells must provide these four functions: Select, DataIn, DataOut, and R/W.



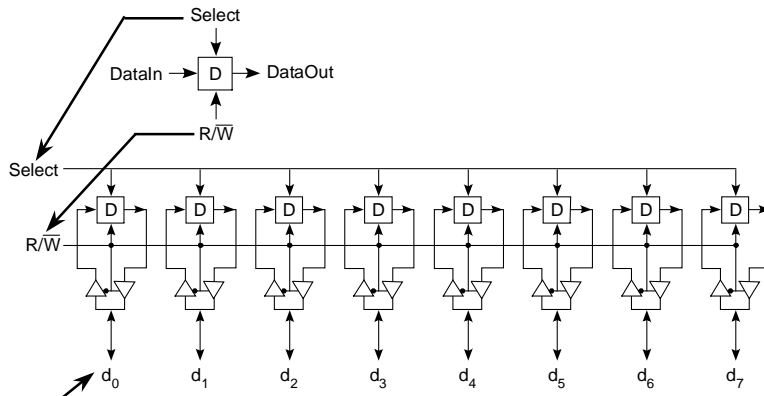
This “static” RAM cell is unrealistic in practice, but it is functionally correct. We will discuss more practical designs later.

12

7-13

Fig 7.4 An 8-Bit Register as a 1-D RAM Array

The entire register is selected with one select line, and uses one $\overline{R/W}$ line



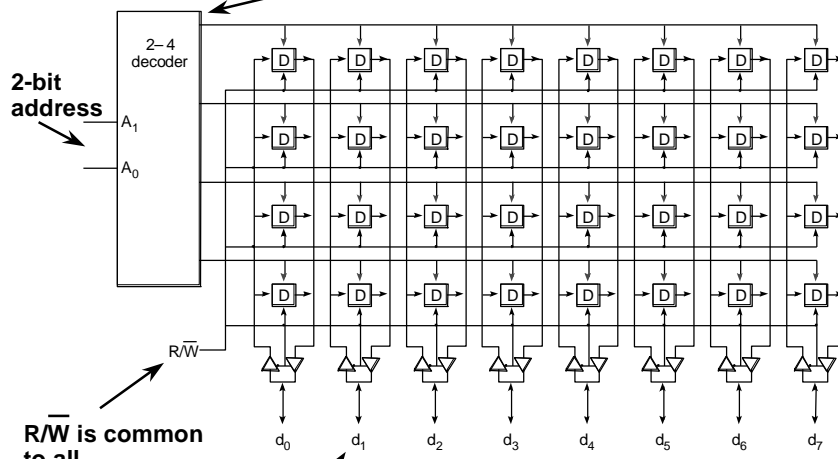
Data bus is bidirectional and buffered. (Why?)

13

7-14

Fig 7.5 A 4 x 8 2-D Memory Cell Array

2-4 line decoder selects one of the four 8-bit arrays



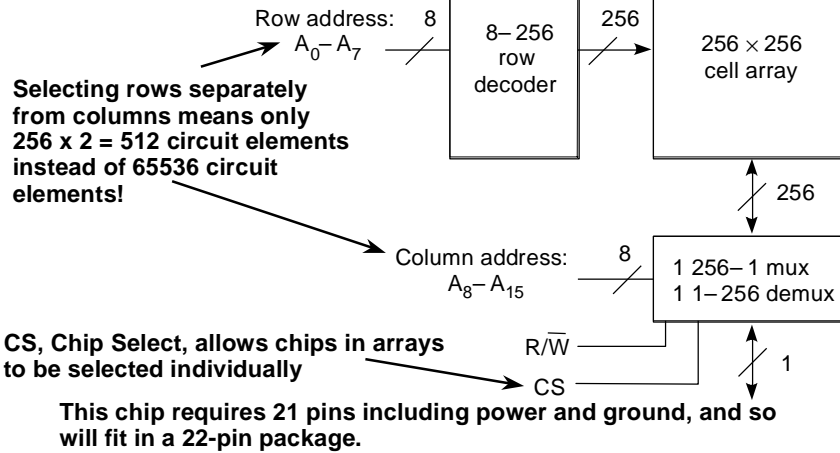
$\overline{R/W}$ is common to all

Bidirectional 8-bit buffered data bus

14

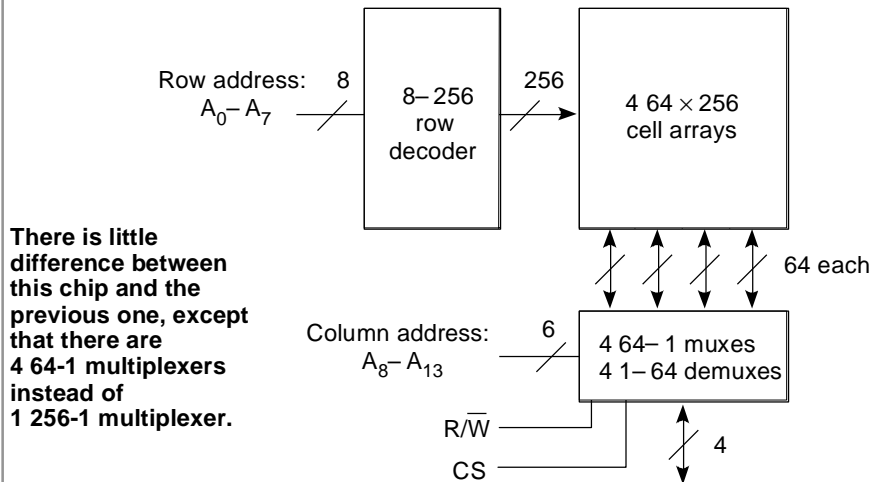
Fig 7.6 A 64 K x 1 Static RAM Chip

~square array fits IC design paradigm



15

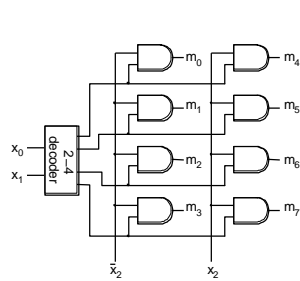
Fig 7.7 A 16 K x 4 SRAM Chip



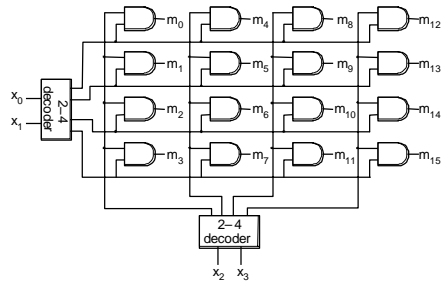
16

Fig 7.8 Matrix and Tree Decoders

- 2-level decoders are limited in size because of gate fan-in. Most technologies limit fan-in to ~8.
- When decoders must be built with fan-in >8, then additional levels of gates are required.
- Tree and matrix decoders are two ways to design decoders with large fan-in:



3-to-8 line tree decoder constructed from 2-input gates.

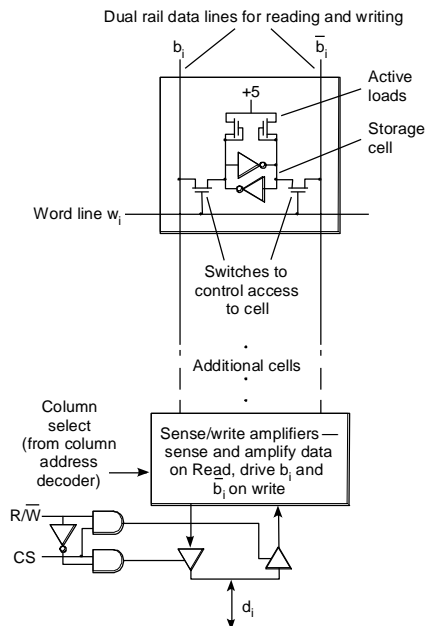


4-to-16 line matrix decoder constructed from 2-input gates.

Fig 7.9 Six-Transistor Static RAM Cell

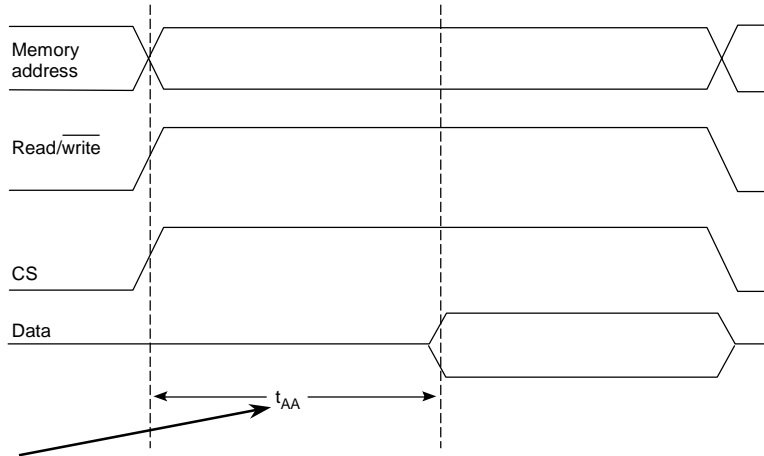
This is a more practical design than the 8-gate design shown earlier.

A value is read by *precharging* the bit lines to a value 1/2 way between a 0 and a 1, while asserting the word line. This allows the latch to drive the bit lines to the value stored in the latch.



7-19

Fig 7.10 Static RAM Read Operation

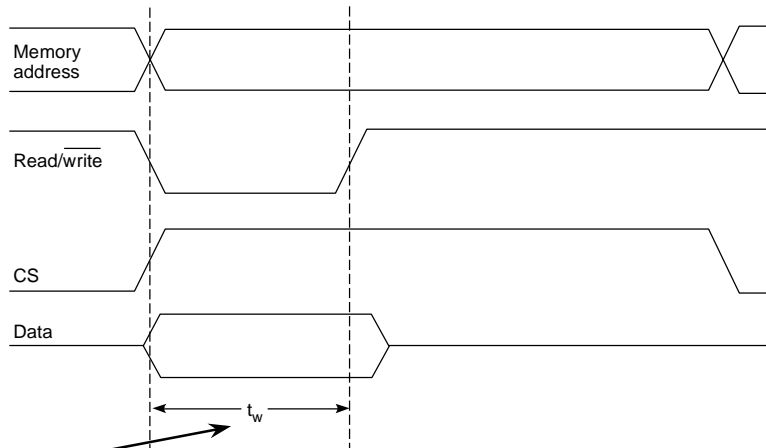


Access time from Address—the time required of the RAM array to decode the address and provide value to the data bus.

19

7-20

Fig 7.11 Static RAM Write Operations



Write time—the time the data must be held valid in order to decode address and store value in memory cells.

20

7-21

Fig 7.12 Dynamic RAM Cell Organization

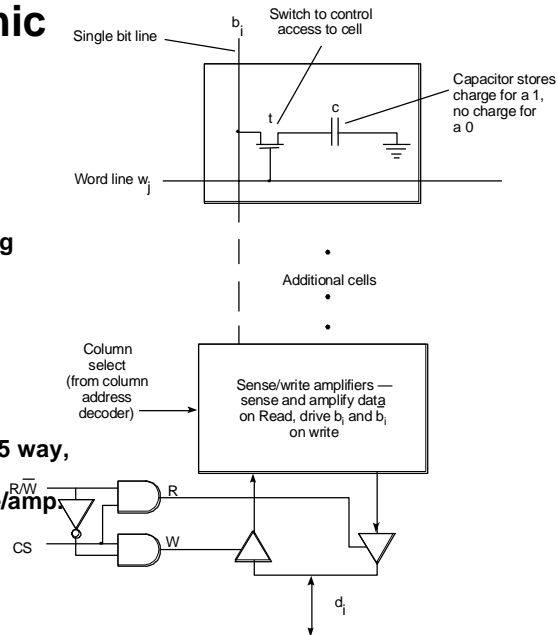
Capacitor will discharge in 4–15 ms.

Refresh capacitor by reading (sensing) value on bit line, amplifying it, and placing it back on bit line where it recharges capacitor.

Write: place value on bit line and assert word line.

Read: precharge bit line to 0.5 way, assert word line, sense value change on bit line with sense/amp.

This need to refresh the storage cells of dynamic RAM chips complicates DRAM system design.



21

7-22

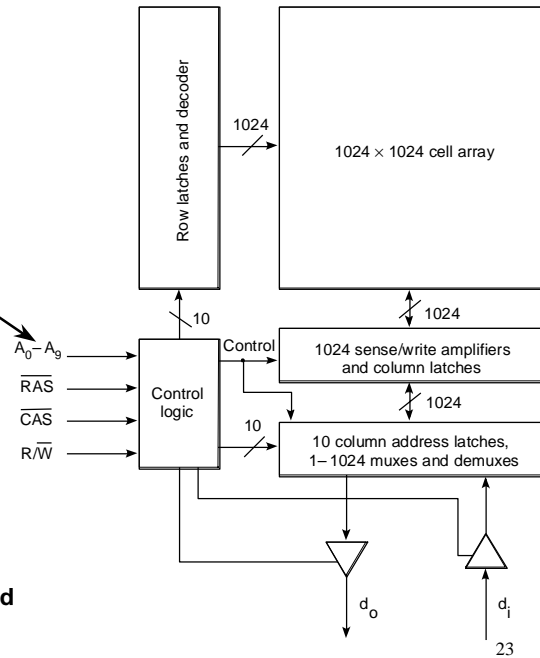
The Refresh

- **Do an entire ROW at once:**
 - the word line is connected to all bits in the row
 - put the halfway value on all the bit lines
 - Assert the word line and sense the bit values
 - amplify these and write internally (nothing gets put on the data lines)
- **Consider a 4Mbit x 1 DRAM, organized as 2048 x 2048,**
- **and all 2048 rows must be refreshed every 4 ms.**
 - If refresh takes 80ns. -- do 2048 refreshes every 4 ms.
 - .. Thus refresh consumes 2048 x 80 ns of each 4 ms.
 - Do the arithmetic --- about 4% of available time

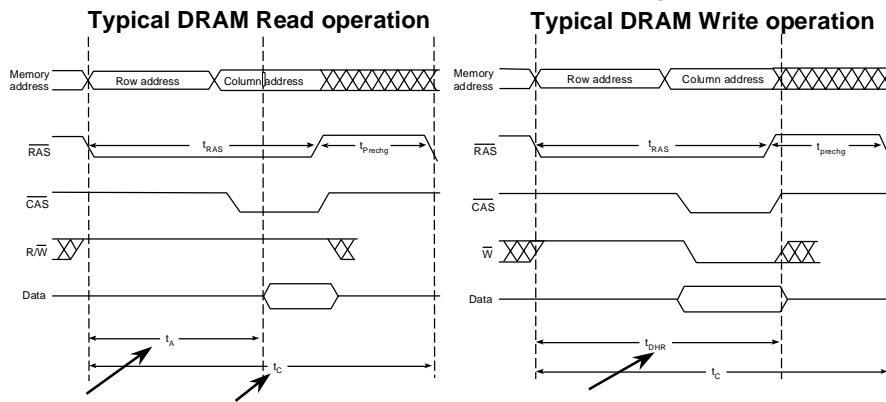
22

Fig 7.13 Dynamic RAM Chip Organization

- Addresses are time-multiplexed on address bus using RAS and CAS as strobes of rows and columns.
 - CAS is normally used as the CS function.
- Notice pin counts:
- Without address multiplexing: 27 pins including power and ground.
 - With address multiplexing: 17 pins including power and ground.



Figs 7.14, 7.15 DRAM Read and Write Cycles



Access time **Cycle time**
 Notice that it is the bit line precharge operation that causes the difference between access time and cycle time.

Data hold from RAS.

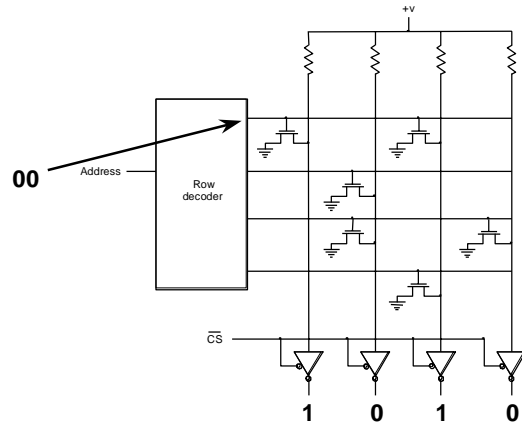
DRAM Refresh and Row Access

- Refresh is usually accomplished by a “RAS-only” cycle. The row address is placed on the address lines and RAS asserted. This refreshed the entire row. CAS is not asserted. The absence of a CAS phase signals the chip that a row refresh is requested, and thus no data is placed on the external data lines.
- Many chips use “CAS before RAS” to signal a refresh. The chip has an internal counter, and whenever CAS is asserted before RAS, it is a signal to refresh the row pointed to by the counter, and to increment the counter.
- Most DRAM vendors also supply one-chip DRAM controllers that encapsulate the refresh and other functions.
- Page mode, nibble mode, and static column mode allow rapid access to the entire row that has been read into the column latches.
- Video RAMS, VRAMS, clock an entire row into a shift register where it can be rapidly read out, bit by bit, for display.

Page Mode DRAM, etc.

- Can get faster access to columns **IN THE SAME ROW** by RAS, CAS+col1, CAS+coli, CAS+coln, etc., since the entire row is already stored in the “column latches”
 - called “Page Mode Memory” - not random access any more!! EDO (extended data out) memories are like this.
- Also “static column mode” where col is automatically incremented by one -- RAS, CAS + col1, CAS, CAS, etc.
- Also “nibble mode” where get four successive bits in the row - A Nibble is a little bi(y)te.....
- SDRAMs (synchronous DRAMS) now common -- specify initial row and column, stream out the row bits starting at that address - down to ~ 25ns. Like VRAM.

Fig 7.16 A 2-D CMOS ROM Chip



Tbl 7.4 ROM Types

| <u>ROM Type</u> | <u>Cost</u> | <u>Programmability</u> | <u>Time to Program</u> | <u>Time to Erase</u> |
|---------------------|------------------|------------------------|------------------------|----------------------|
| Mask-programmed ROM | Very inexpensive | At factory only | Weeks | N/A |
| PROM | Inexpensive | Once, by end user | Seconds | N/A |
| EPROM | Moderate | Many times | Seconds | 20 minutes |
| Flash EPROM | Expensive | Many times | 100 μ s | 1 s, large block |
| EEPROM | Very expensive | Many times | 100 μ s | 10 ms, byte |

Memory Boards and Modules

- There is a need for memories that are larger and wider than a single chip
- Chips can be organized into “boards.”
 - Boards may not be actual, physical boards, but may consist of structured chip arrays present on the motherboard.
- A board or collection of boards make up a memory module.
- Memory modules:
 - Satisfy the processor–main memory interface requirements
 - May have DRAM refresh capability
 - May expand the total main memory capacity
 - May be *interleaved* to provide faster access to blocks of words

Fig 7.17 General Structure of a Memory Chip

This is a slightly different view of the memory chip than previous.

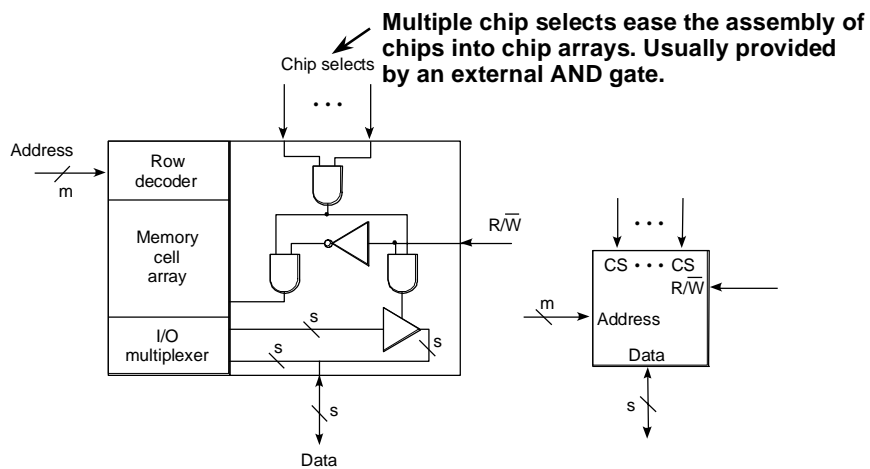
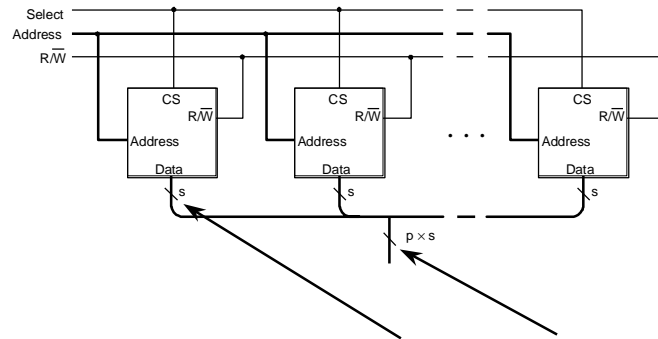


Fig 7.18 Word Assembly from Narrow Chips

All chips have common CS, $\overline{R/W}$, and Address lines.



P chips expand word size from s bits to $p \times s$ bits.

Why “Narrow” Chips??

- For SRAM, adding a data pin to go from s to $s+1$ bits gives a relative increase of $(s+1)/s$ in capacity
- adding an address pin sends memory from
- 2^m to $2^{(m+1)}$ i.e. it doubles capacity
- For DRAM, adding an address pin doubles both the number of rows and the number of columns, so chip capacity increased by a factor of 4

Fig 7.19 Increasing the Number of Words by a Factor of 2^k

The additional k address bits are used to select one of 2^k chips, each one of which has 2^m words:

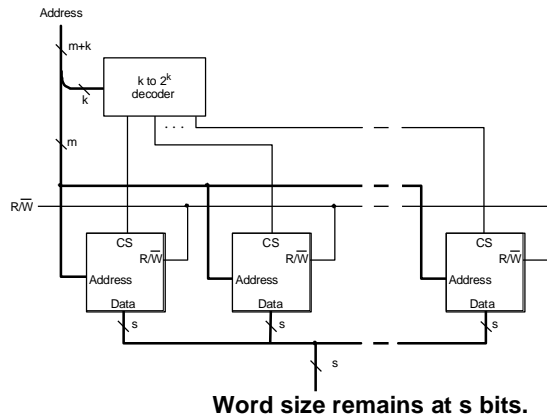
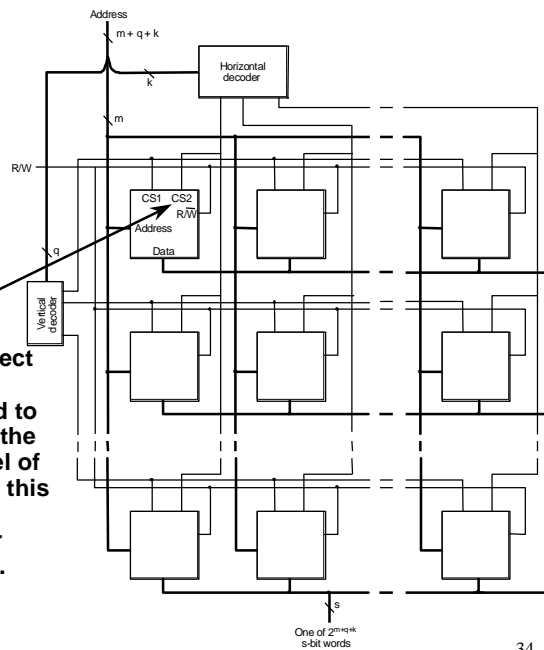


Fig 7.20 Chip Matrix Using Two Chip Selects

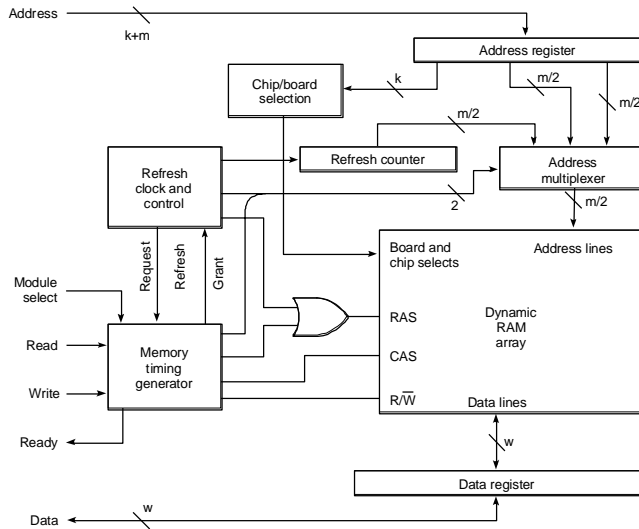
This scheme simplifies the decoding from use of a $(q+k)$ -bit decoder to using one q -bit and one k -bit decoder.

Multiple chip select lines are used to replace the last level of gates in this matrix decoder scheme.



7-37

Fig 7.23 Dynamic RAM Module with Refresh Control

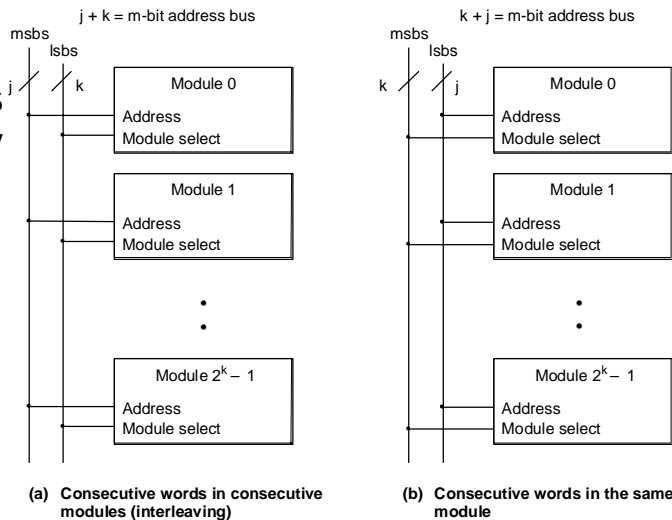


37

7-38

Fig 7.24 Two Kinds of Memory Module Organiz'n.

Memory modules are used to allow access to more than one word simultaneously.



38

Fig 7.25 Timing of Multiple Modules on a Bus

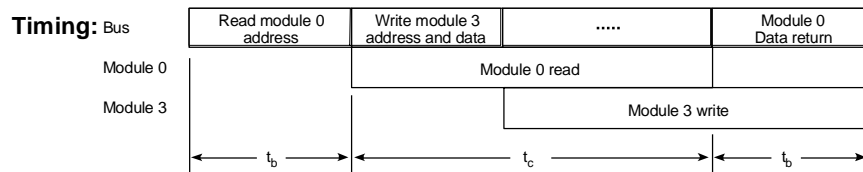
If time to transmit information over bus, t_b , is $<$ module cycle time, t_c , it is possible to time multiplex information transmission to several modules;

Example: store one word of each cache line in a separate module.

Main Memory Address:

| | |
|------|------------|
| Word | Module No. |
|------|------------|

This provides successive words in successive modules.



With interleaving of 2^k modules, and $t_b < t_c/2^k$, it is possible to get a 2^k -fold increase in memory bandwidth, provided memory requests are pipelined. DMA satisfies this requirement.

Memory System Performance

Breaking the memory access process into steps:

For all accesses:

- transmission of address to memory
- transmission of control information to memory (R/W, Request, etc.)
- decoding of address by memory

For a Read:

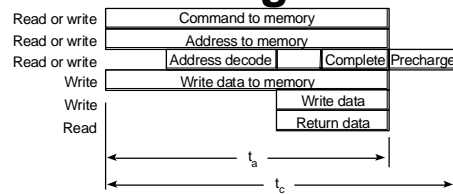
- return of data from memory
- transmission of completion signal

For a Write:

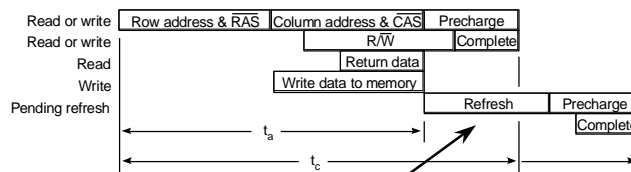
- transmission of data to memory (usually simultaneous with address)
- storage of data into memory cells
- transmission of completion signal

The next slide shows the access process in more detail.

Fig 7.26 Sequence of Steps in Accessing Memory



(a) Static RAM behavior



(b) Dynamic RAM behavior

“Hidden refresh” cycle. A normal cycle would exclude the pending refresh step. -more-

Example SRAM Timings

Approximate values for static RAM Read timing:

- Address bus drivers turn-on time: 40 ns.
- Bus propagation and bus skew: 10 ns.
- Board select decode time: 20 ns.
- Time to propagate select to another board: 30 ns.
- Chip select: 20 ns.

PROPAGATION TIME FOR ADDRESS AND COMMAND TO REACH CHIP: 120 ns.

- On-chip memory read access time: 80 ns.
- Delay from chip to memory board data bus: 30 ns.
- Bus driver and propagation delay (as before): 50 ns.

TOTAL MEMORY READ ACCESS TIME: 280 ns.

Moral: 70 ns chips do not necessarily provide 70 ns access time!

Considering Any Two Adjacent Levels of the Memory Hierarchy

Some definitions:

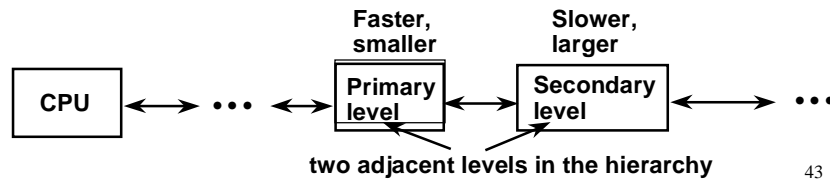
Temporal locality: the property of most programs that if a given memory location is referenced, it is likely to be referenced again, “soon.”

Spatial locality: if a given memory location is referenced, those locations near it numerically are likely to be referenced “soon.”

Working set: The set of memory locations referenced over a fixed period of time, or in a *time window*.

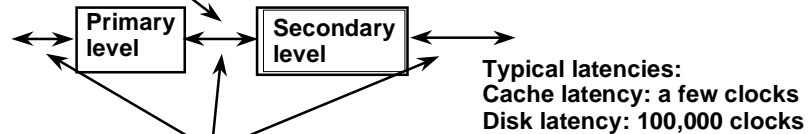
Notice that temporal and spatial locality both work to assure that the contents of the working set change only slowly over execution time.

Defining the primary and secondary levels:



Primary and Secondary Levels of the Memory Hierarchy

Speed between levels defined by latency: time to access first word, and bandwidth, the number of words per second transmitted between levels.



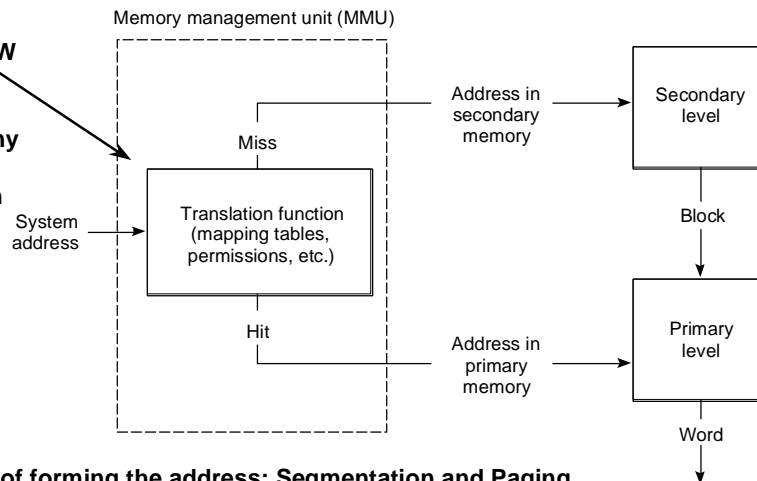
- The item of commerce between any two levels is the **block**.
- Blocks may/will differ in size at different levels in the hierarchy.
Example: Cache block size ~ 16–64 bytes.
 Disk block size: ~ 1–4 Kbytes.
- As working set changes, blocks are moved back/forth through the hierarchy to satisfy memory access requests.
- A complication: Addresses will differ depending on the level.
Primary address: the address of a value in the primary level.
Secondary address: the address of a value in the secondary level.

Primary and Secondary Address Examples

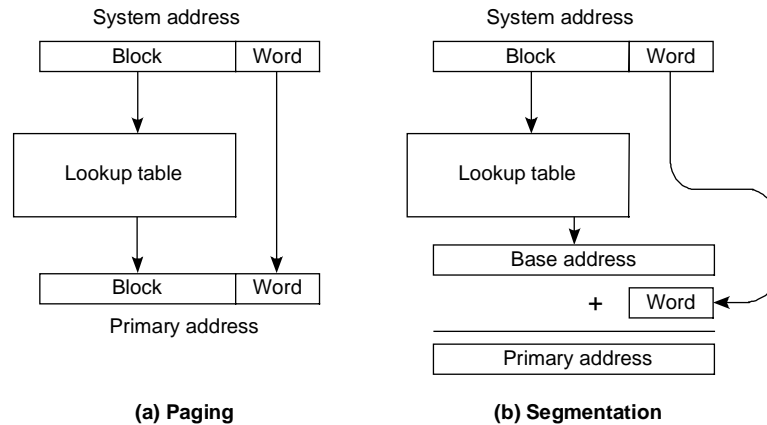
- **Main memory address:** unsigned integer
- **Disk address:** track number, sector number, offset of word in sector.

Fig 7.28 Addressing and Accessing a Two-Level Hierarchy

The computer system, HW or SW, must perform any address translation that is required:



Two ways of forming the address: Segmentation and Paging. Paging is more common. Sometimes the two are used together, one "on top of" the other. More about address translation and paging later...

Fig 7.29 Primary Address Formation

Hits and Misses; Paging; Block Placement

Hit: the word was found at the level from which it was requested.

Miss: the word was not found at the level from which it was requested. (A miss will result in a request for the block containing the word from the next higher level in the hierarchy.)

Hit ratio (or hit rate) = $h = \frac{\text{number of hits}}{\text{total number of references}}$

Miss ratio: $1 - \text{hit ratio}$

t_p = primary memory access time. t_s = secondary memory access time

Access time, $t_a = h \cdot t_p + (1-h) \cdot t_s$.

Page: commonly, a disk block. **Page fault:** synonymous with a miss.

Demand paging: pages are moved from disk to main memory only when a word in the page is requested by the processor.

Block placement and replacement decisions must be made each time a block is moved.

Virtual Memory

A virtual memory is a memory hierarchy, usually consisting of at least main memory and disk, in which the processor issues all memory references as effective addresses in a flat address space. All translations to primary and secondary addresses are handled transparently to the process making the address reference, thus providing the *illusion* of a flat address space.

Recall that disk accesses may require 100,000 clock cycles to complete, due to the slow access time of the disk subsystem. Once the processor has, through mediation of the operating system, made the proper request to the disk subsystem, it is available for other tasks.

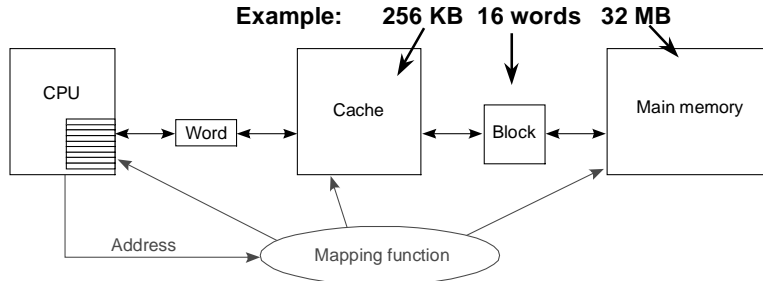
Multiprogramming shares the processor among independent programs that are resident in main memory and thus available for execution.

Decisions in Designing a 2-Level Hierarchy

- Translation procedure to translate from system address to primary address.
- Block size—block transfer efficiency and miss ratio will be affected.
- Processor dispatch on miss—processor wait or processor multiprogrammed.
- Primary-level placement—direct, associative, or a combination. Discussed later.
- Replacement policy—which block is to be replaced upon a miss.
- Direct access to secondary level—in the cache regime, can the processor directly access main memory upon a cache miss?
- Write through—can the processor write directly to main memory upon a cache miss?
- Read through—can the processor read directly from main memory upon a cache miss as the cache is being updated?
- Read or write bypass—can certain infrequent read or write misses be satisfied by a direct access of main memory without any block movement?

7-51

Fig 7.30 The Cache Mapping Function



The cache mapping function is responsible for all cache operations:

- Placement strategy: where to place an incoming block in the cache
- Replacement strategy: which block to replace upon a miss
- Read and write policy: how to handle reads and writes upon cache misses

Mapping function must be implemented in hardware. (Why?)

Three different types of mapping functions:

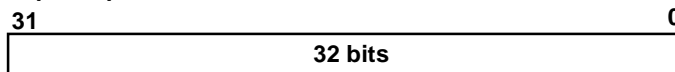
- Associative
- Direct mapped
- Block-set associative

51

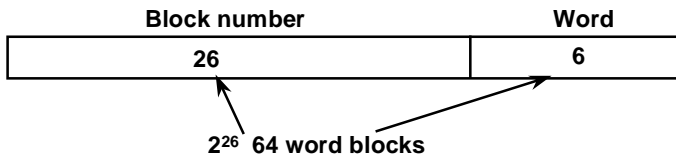
7-52

Memory Fields and Address Translation

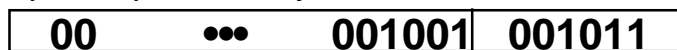
Example of processor-issued 32-bit virtual address:



That same 32-bit address partitioned into two fields, a block field, and a word field. The word field represents the offset into the block specified in the block field:



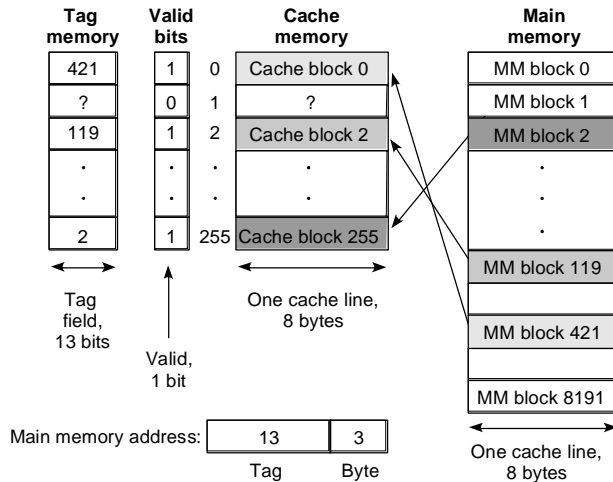
Example of a specific memory reference: Block 9, word 11.



52

Fig 7.31 Associative Cache

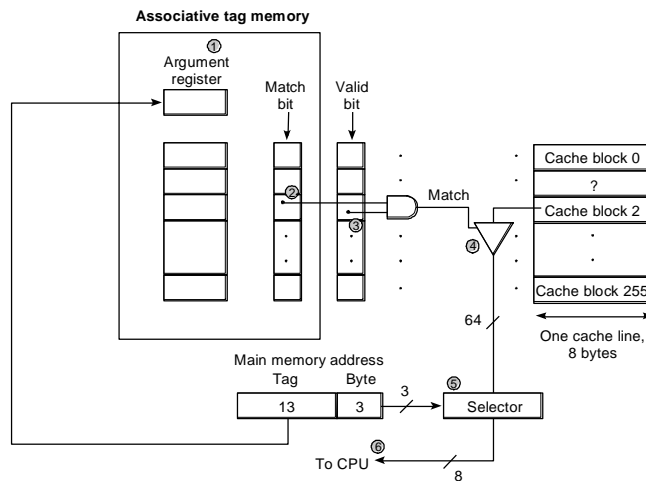
Associative mapped cache model: any block from main memory can be put anywhere in the cache. Assume a 16-bit main memory.*



*16 bits, while unrealistically small, simplifies the examples

Fig 7.32 Associative Cache Mechanism

Because any block can reside anywhere in the cache, an *associative* (content addressable) memory is used. All locations are searched simultaneously.



Advantages and Disadvantages of the Associative Mapped Cache

Advantage

- Most flexible of all—any MM block can go anywhere in the cache.

Disadvantages

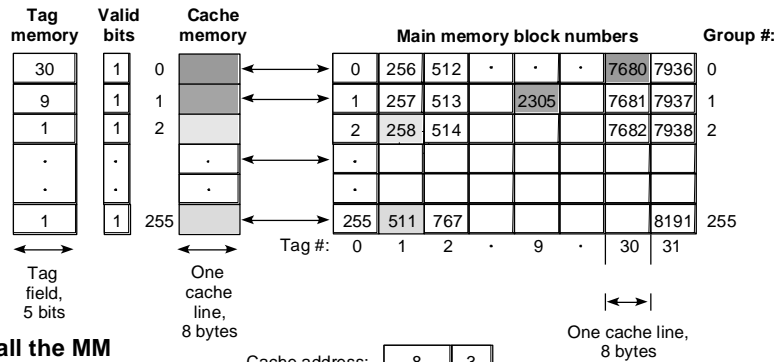
- Large tag memory.
- Need to search entire tag memory simultaneously means lots of hardware.

Replacement Policy is an issue when the cache is full. –more later–

Q.: How is an associative search conducted at the logic gate level?

Direct-mapped caches simplify the hardware by allowing each MM block to go into only one place in the cache:

Fig 7.33 Direct-Mapped Cache

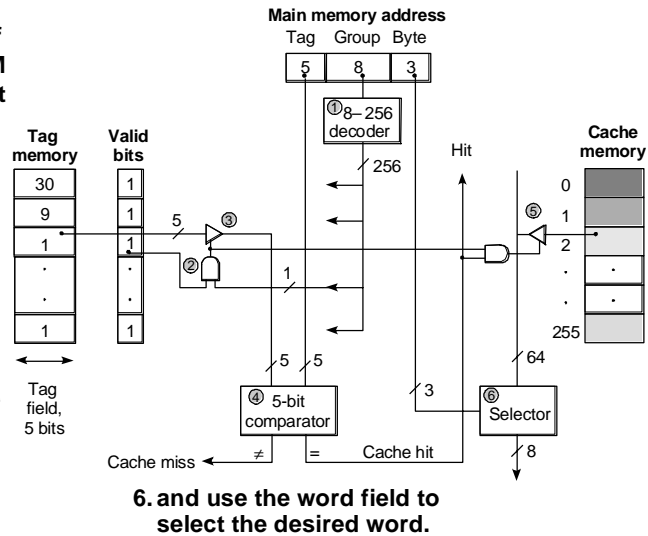


Key Idea: all the MM blocks from a given group can go into only one location in the cache, corresponding to the group number.

Now the cache needs only examine the single group that its reference specifies.

Fig 7.34 Direct-Mapped Cache Operation

1. Decode the group number of the incoming MM address to select the group
2. If Match AND Valid
3. Then gate out the tag field
4. Compare cache tag with incoming tag
5. If a hit, then gate out the cache line

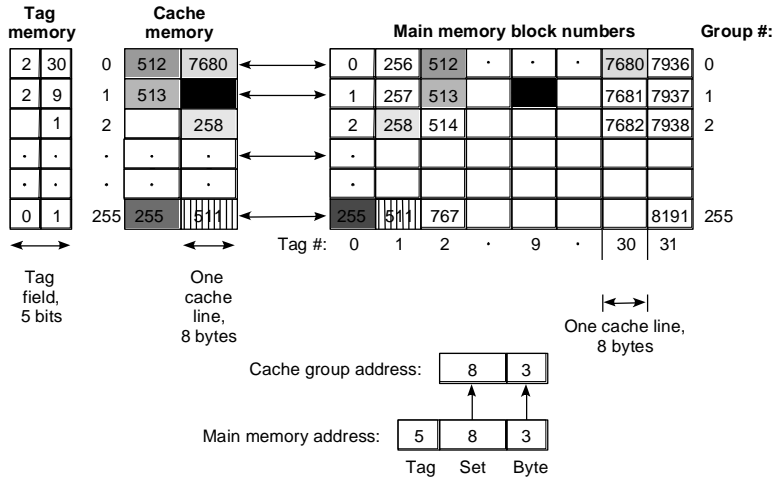


Direct-Mapped Caches

- The direct mapped cache uses less hardware, but is much more restrictive in block placement.
- If two blocks from the same group are frequently referenced, then the cache will “thrash.” That is, repeatedly bring the two competing blocks into and out of the cache. This will cause a performance degradation.
- Block replacement strategy is trivial.
- Compromise—allow several cache blocks in each group—the Block-Set-Associative Cache:

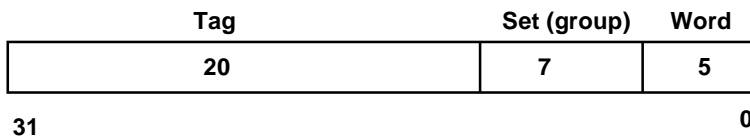
Fig 7.35 2-Way Set-Associative Cache

Example shows 256 groups, a set of two per group.
Sometimes referred to as a 2-way set-associative cache.



Getting Specific: The Intel Pentium Cache

- The Pentium actually has two separate caches—one for instructions and one for data. Pentium issues 32-bit MM addresses.
 - Each cache is 2-way set-associative
 - Each cache is 8 K = 2^{13} bytes in size
 - $32 = 2^5$ bytes per line.
 - Thus there are 64 or 2^6 bytes per set, and therefore $2^{13}/2^6 = 2^7 = 128$ groups
 - This leaves $32 - 5 - 7 = 20$ bits for the tag field:



This "cache arithmetic" is important, and deserves your mastery.

Cache Read and Write Policies

- Read and Write cache hit policies
 - Writethrough—updates both cache and MM upon each write.
 - Write back—updates only cache. Updates MM only upon block removal.
 - “Dirty bit” is set upon first write to indicate block must be written back.
- Read and Write cache miss policies
 - Read miss—bring block in from MM
 - Either forward desired word as it is brought in, or
 - Wait until entire line is filled, then repeat the cache request.
 - Write miss
 - Write-allocate—bring block into cache, then update
 - Write-no-allocate—write word to MM without bringing block into cache.

Block Replacement Strategies

- Not needed with direct-mapped cache
- Least Recently Used (LRU)
 - Track usage with a counter. Each time a block is accessed:
 - Clear counter of accessed block
 - Increment counters with values less than the one accessed
 - All others remain unchanged
 - When set is full, remove line with highest count
- Random replacement—replace block at random
 - Even random replacement is a fairly effective strategy

Cache Performance

Recall Access time, $t_a = h \cdot t_p + (1 - h) \cdot t_s$ for primary and secondary levels.

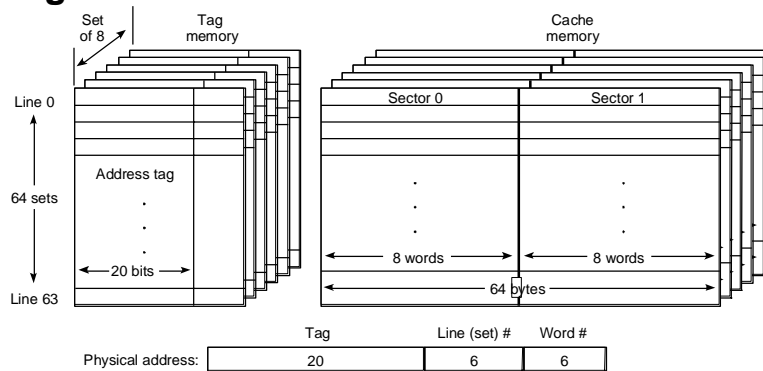
For $t_p = \text{cache}$ and $t_s = \text{MM}$,

$$t_a = h \cdot t_c + (1 - h) \cdot t_M$$

We define S , the speedup, as $S = T_{\text{without}} / T_{\text{with}}$ for a given process, where T_{without} is the time taken without the improvement, cache in this case, and T_{with} is the time the process takes with the improvement.

Having a model for cache and MM access times and cache line fill time, the speedup can be calculated once the hit ratio is known.

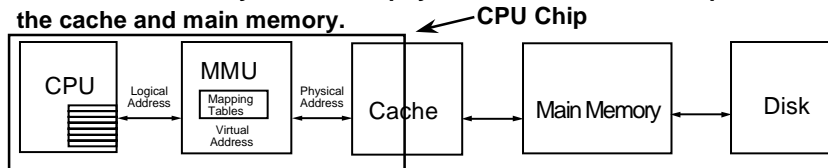
Fig 7.36 The PowerPC 601 Cache Structure



- The PPC 601 has a *unified* cache—that is, a single cache for both instructions and data.
- It is 32 KB in size, organized as 64 x 8 block-set associative, with blocks being 8 8-byte words organized as 2 independent 4-word sectors for convenience in the updating process
- A cache line can be updated in two single-cycle operations of 4 words each.
- Normal operation is write back, but write through can be selected on a per line basis via software. The cache can also be disabled via software.

Virtual Memory

The memory management unit, MMU, is responsible for mapping logical addresses issued by the CPU to physical addresses that are presented to the cache and main memory.



A word about addresses:

- **Effective address**—an address computed by the processor while executing a program. Synonymous with logical address.
 - The term effective address is often used when referring to activity inside the CPU. Logical address is most often used when referring to addresses when viewed from outside the CPU.
- **Virtual address**—the address generated from the logical address by the memory management unit, MMU.
- **Physical address**—the address presented to the memory unit.

(Note: Every address reference must be translated.)

Virtual Addresses—Why

The logical address provided by the CPU is translated to a virtual address by the MMU. Often the virtual address space is larger than the logical address, allowing program units to be mapped to a much larger virtual address space.

Getting Specific: The PowerPC 601

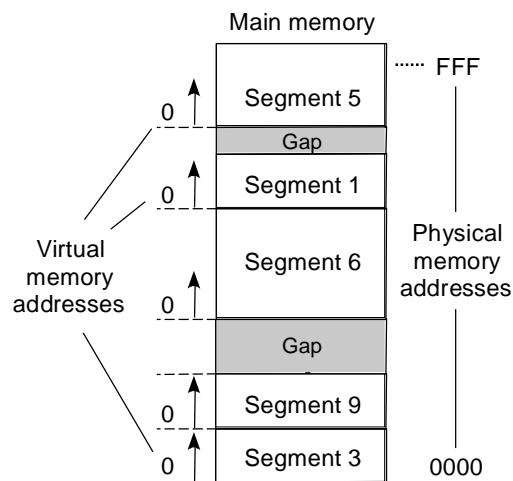
- The PowerPC 601 CPU generates 32-bit logical addresses.
- The MMU translates these to 52-bit virtual addresses before the final translation to physical addresses.
- Thus while each process is limited to 32 bits, the main memory can contain many of these processes.
- Other members of the PPC family will have different logical and virtual address spaces, to fit the needs of various members of the processor family.

Virtual Addressing—Advantages

- **Simplified addressing.** Each program unit can be compiled into its own memory space, beginning at address 0 and potentially extending far beyond the amount of physical memory present in the system.
 - No address relocation required at load time.
 - No need to fragment the program to accommodate

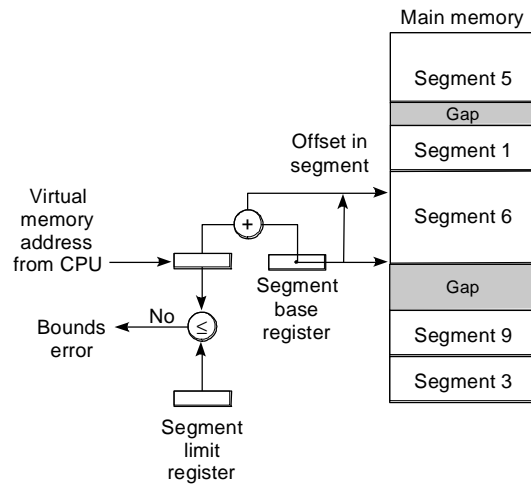
This is the origin of those “bus error” and “segmentation fault” messages.

**Fig 7.38
Memory
Management
by
Segmentation**



- Notice that each segment’s virtual address starts at 0, different from its physical address.
- Repeated movement of segments into and out of physical memory will result in gaps between segments. This is called external fragmentation.
- Compaction routines must be occasionally run to remove these fragments.

Fig 7.39 Segmentation Mechanism

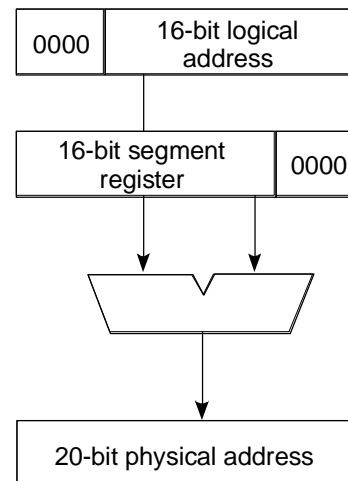


- The computation of physical address from virtual address requires an integer addition for each memory reference, and a comparison if segment limits are checked.
- Q: How does the MMU switch references from one segment to another?

69

Fig 7.40 The Intel 8086 Segmentation Scheme

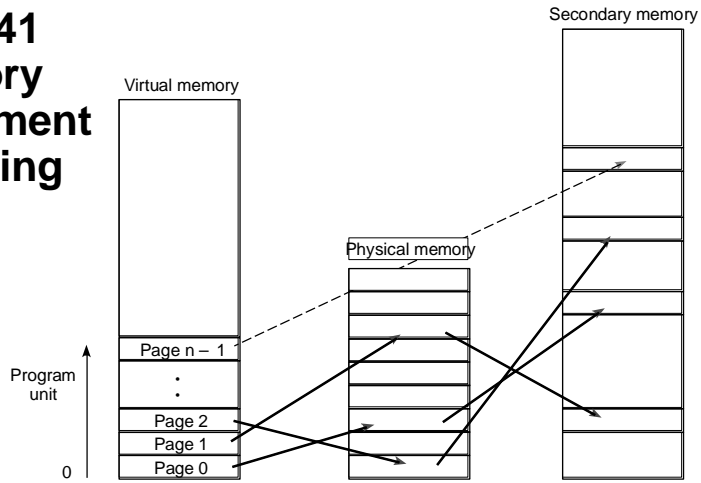
The first popular 16-bit processor, the Intel 8086 had a primitive segmentation scheme to “stretch” its 16-bit logical address to a 20-bit physical address:



The CPU allows 4 simultaneously active segments, CODE, DATA, STACK, and EXTRA. There are 4 16-bit segment base registers.

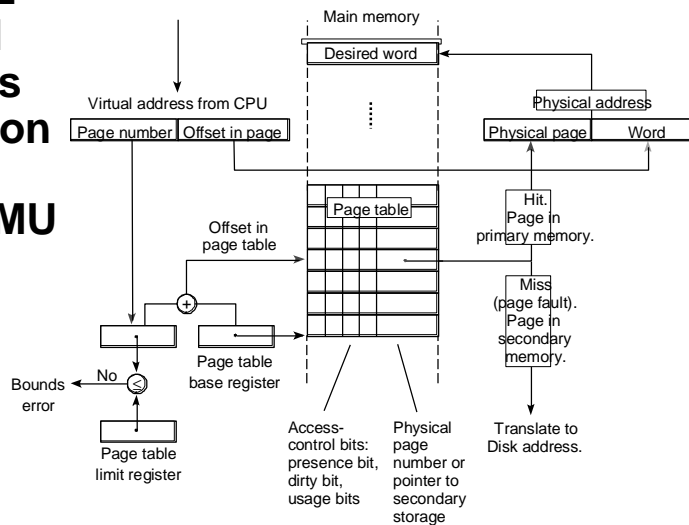
70

**Fig 7.41
Memory
Management
by Paging**



- This figure shows the mapping between virtual memory pages, physical memory pages, and pages in secondary memory. Page n - 1 is not present in physical memory, but only in secondary memory.
- The MMU manages this mapping.

**Fig 7.42
Virtual
Address
Translation
in a
Paged MMU**



- 1 table per user per program unit
- One translation per memory access
- Potentially large page table

A page fault will result in 100,000 or more cycles passing before the page has been brought from secondary storage to MM.

Page Placement and Replacement

Page tables are direct mapped, since the physical page is computed directly from the virtual page number.

But physical pages can reside anywhere in physical memory.

Page tables such as those on the previous slide result in large page tables, since there must be a page table entry for every page in the program unit.

Some implementations resort to hash tables instead, which need have entries only for those pages actually present in physical memory.

Replacement strategies are generally LRU, or at least employ a “use bit” to guide replacement.

Fast Address Translation: Regaining Lost Ground

- The concept of virtual memory is very attractive, but leads to considerable overhead:
 - There must be a translation for every memory reference.
 - There must be *two* memory references for every program reference:
 - One to retrieve the page table entry,
 - one to retrieve the value.
 - Most caches are addressed by physical address, so there must be a virtual to physical translation before the cache can be accessed.

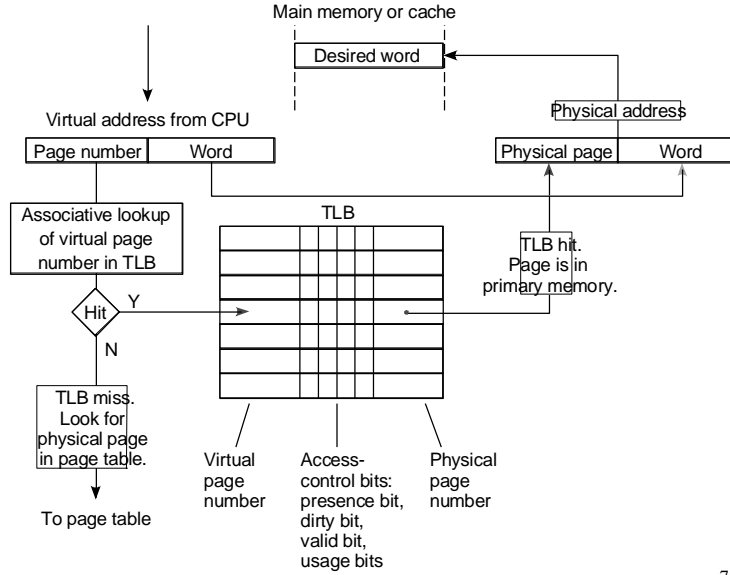
The answer: a small cache in the processor that retains the last few virtual to physical translations: a Translation Lookaside Buffer, TLB.

The TLB contains not only the virtual to physical translations, but also the valid, dirty, and protection bits, so a TLB hit allows the processor to access physical memory directly.

The TLB is usually implemented as a fully associative cache:

7-75

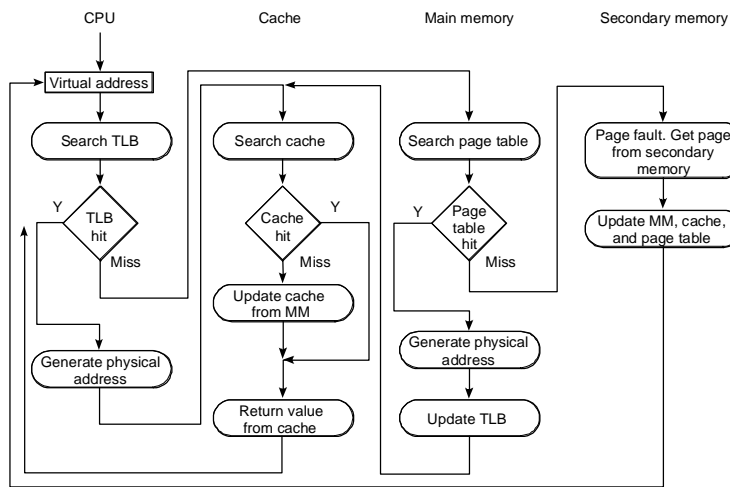
Fig 7.43 Translation Lookaside Buffer Structure and Operation



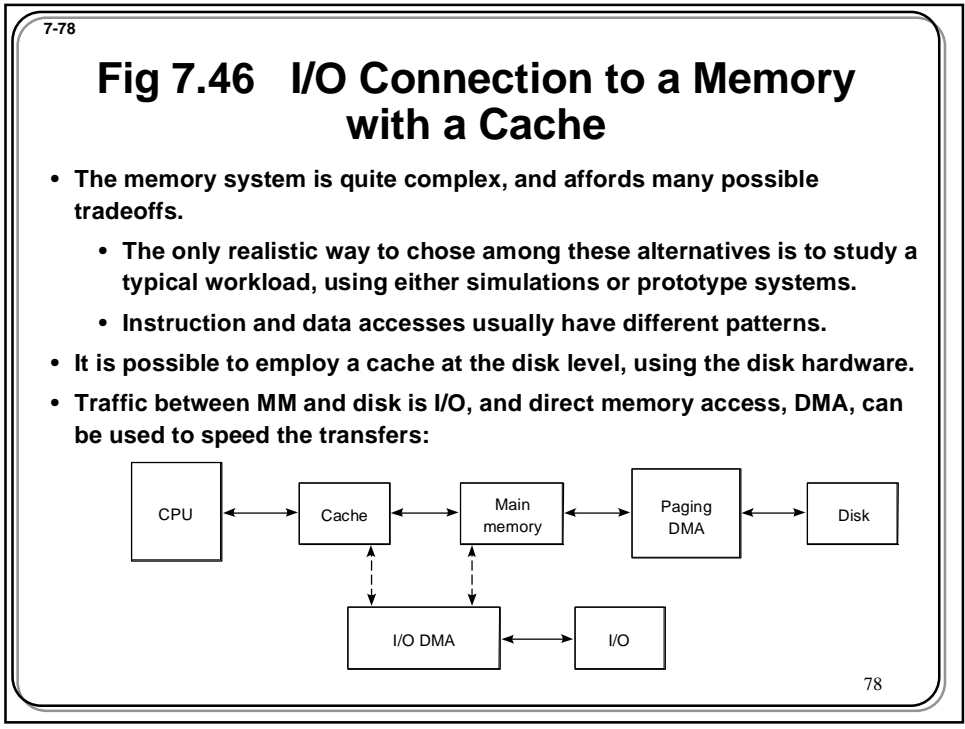
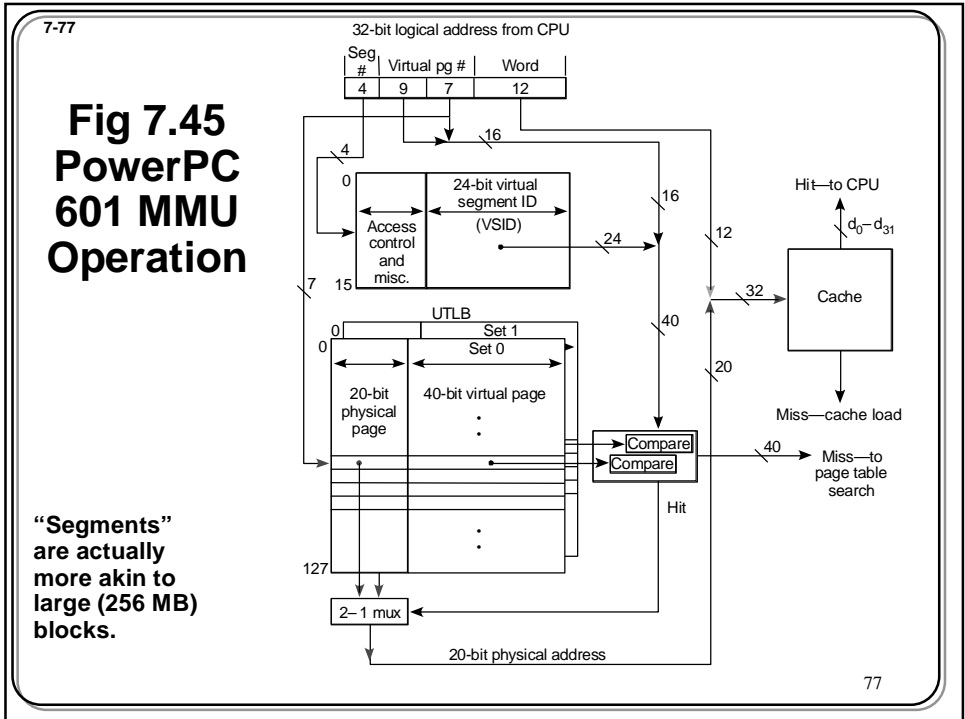
75

7-76

Fig 7.44 Operation of the Memory Hierarchy



76



Chapter 7 Summary

- Most memory systems are multileveled—cache, main memory, and disk.
- Static and dynamic RAM are fastest components, and their speed has the strongest effect on system performance.
- Chips are organized into boards and modules.
- Larger, slower memory is attached to faster memory in a hierarchical structure.
- The cache to main memory interface requires hardware address translation.
- Virtual memory—the main memory–disk interface—can employ software for address translation because of the slower speeds involved.
- The hierarchy must be carefully designed to ensure optimum price-performance.