Chapter 7- Memory System Design

- **Introduction**
- **RAM structure: Cells and Chips**
- **Memory boards and modules**
- **Two-level memory hierarchy**
- **The cache**
- **Virtual memory**
- **The memory as a sub-system of the computer**

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
- etc.

What other issues can you think of that will influence memory design?
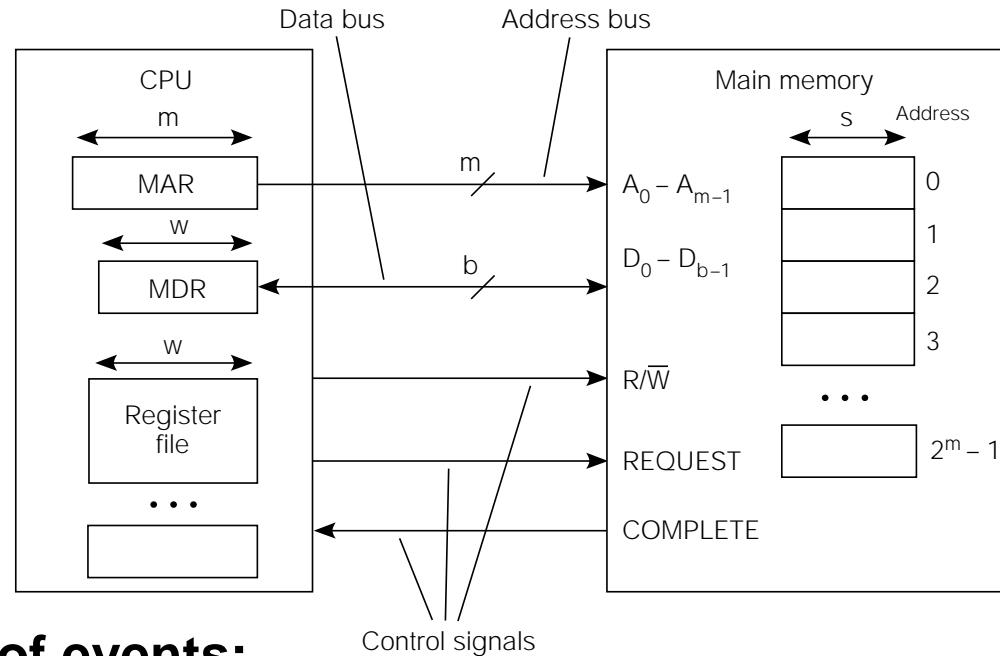
In This Chapter we will cover–

- 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–Main Memory Interface

Data bus   Address bus

CPU

Main memory

m

MAR

m

$A_0 - A_{m-1}$

w

MDR

b

$D_0 - D_{b-1}$

w

Register file

$R/\overline{W}$

REQUEST

COMPLETE

s   Address

0
1
2
3

. . .

$2^m - 1$

. . .

Control signals
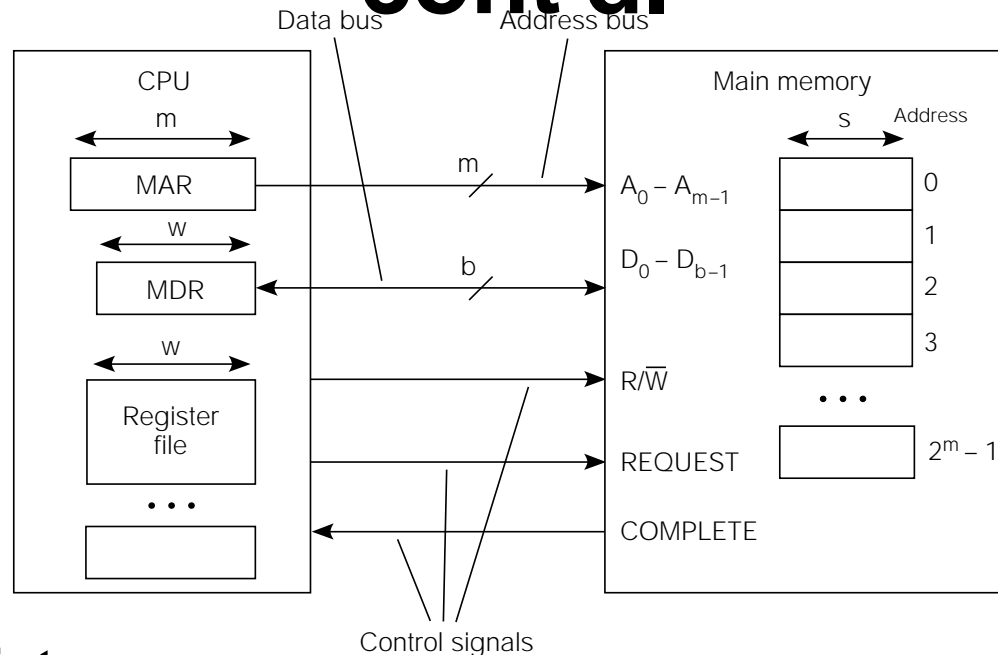
Sequence of events:

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.

-more-

3. Main Memory asserts COMPLETE.

The CPU–Main 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.**

Table 7.1   Some Memory Properties

Symbol	Definition	Intel 8088	Intel 8086	IBM/Moto. 601
w	CPU Word Size	16bits	16bits	64 bits
m	Bits in a logical memory address	20 bits	20 bits	32 bits
s	Bits in smallest addressable unit	8	8	8
b	Data Bus size	8	16	64
2^m	Memory wd capacity, s-sized wds	2^{20}	2^{20}	2^{32}
2^mxs	Memory bit capacity	2^{20}x8	2^{20}x8	2^{32}x8

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*)"?

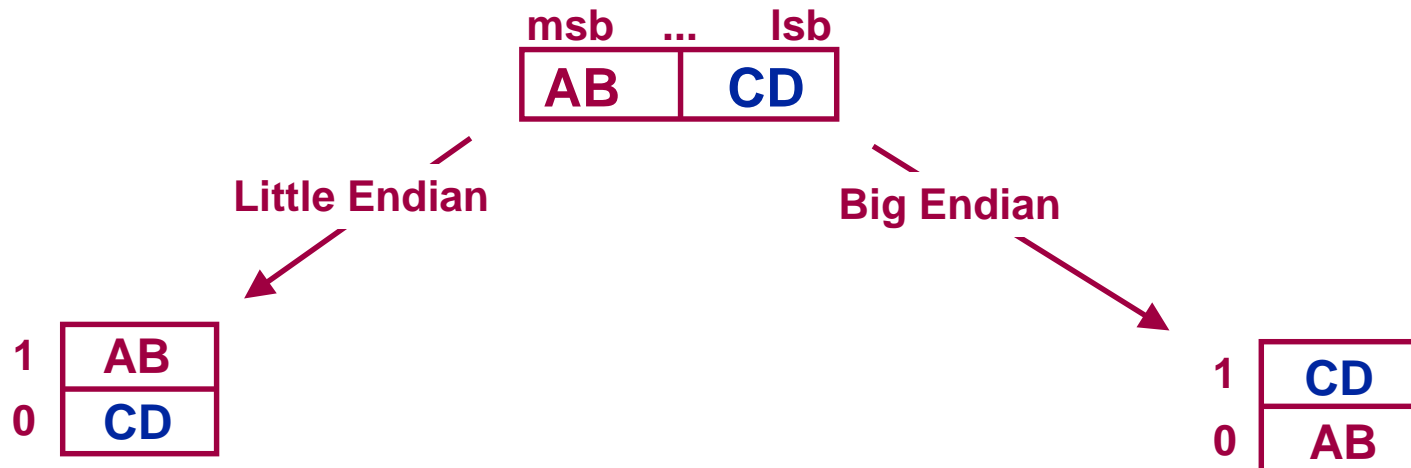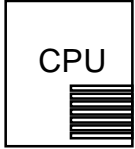**Example: The hexadecimal 16-bit number ABCDH, stored at address 0:**

msb	...	lsb
AB		CD

Little Endian

1	AB
0	CD

Big Endian

1	CD
0	AB
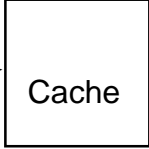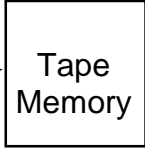
Table 7.2   Memory Performance Parameters

Symbol	Definition	Units	Meaning
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
b	Bandwidth	words/time	Word transmission rate
t_l	Latency	time	Time to access first word of a sequence of words
$t_{bl} = t_l + k/b$	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.)

Table 7.3 The Memory Hierarchy, Cost, and Performance

Some Typical Values:

Component	CPU	Cache	Main Memory	Disk Memory	Tape Memory
Access	Random	Random	Random	Direct	Sequential
Capacity, bytes	64-1K	4MB	4GB	85GB	1TB
Latency	10ns	20ns	50ns	10ms	10ms-10s
Block size	1 word	16 words	16 words	4KB	4KB
Bandwidth	System clock rate	666MB/s	200MB/s	160MB/s	4MB/s
Cost	High	$50	$0.75	$0.08	$0.001

Intel Architecture Over Time

Processor	Release Date	MIPS	Max. CPU Frequency at Introduction	# of Xtors on Die	Main CPU Register Size	External Data Bus Size	Max, External Address Space	Caches in CPU Package
8086	1978	0.8	8 MHz	29 K	16	16	1 MB	None
Intel286	1982	2.7	12.5 MHz	134 K	16	16	16 MB	None
Intel386 DX	1985	6	20 MHz	275 K	32	32	4 GB	None
Intel486 DX	1989	20	25 MHz	1.2 M	32	32	4 GB	8 KB L1
Pentium	1993	100	60 MHz	3.1 M	32	64	4 GB	16 KB L1
Pentium Pro	1995	440	200 MHz	5.5 M	32	64	64 GB	16 KB L1; 256KB or 512 KB L2
Pentium II	1997	466	266 MHz	7 M	32	64	64 GB	32KB L1; 256 KB or 512 KB L2
Pentium III	1999	1000	500 MHz	8.2 M	32 GP 128 SIMD-FP	64	64 GB	32 KB L1; 512 KB L2

Fig. 7.3 Memory Cells - a conceptual view

Regardless of the technology, all RAM memory cells must provide these four functions: Select, DataIn, DataOut, and R/W.
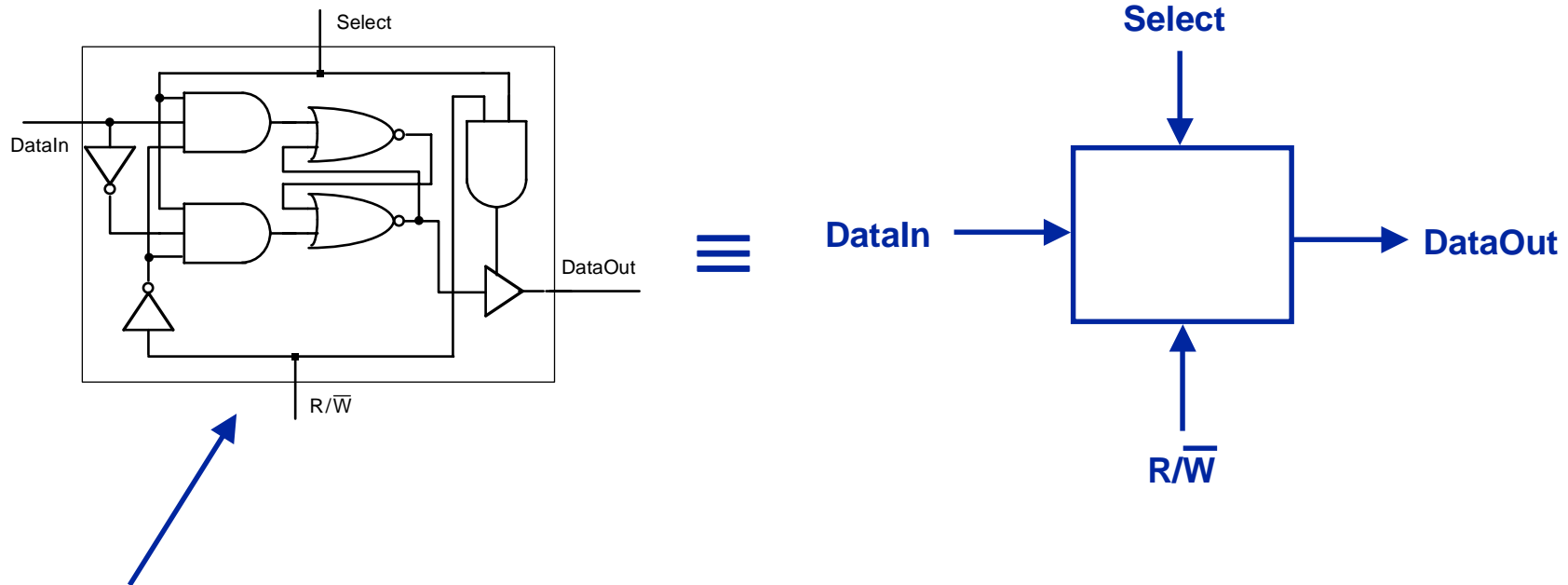
This "static" RAM cell is unrealistic in practice, but it is functionally correct. We will discuss more practical designs later.

Fig. 7.4 An 8-bit register as a 1D RAM array

The entire register is selected with one select line, and uses one R/W line
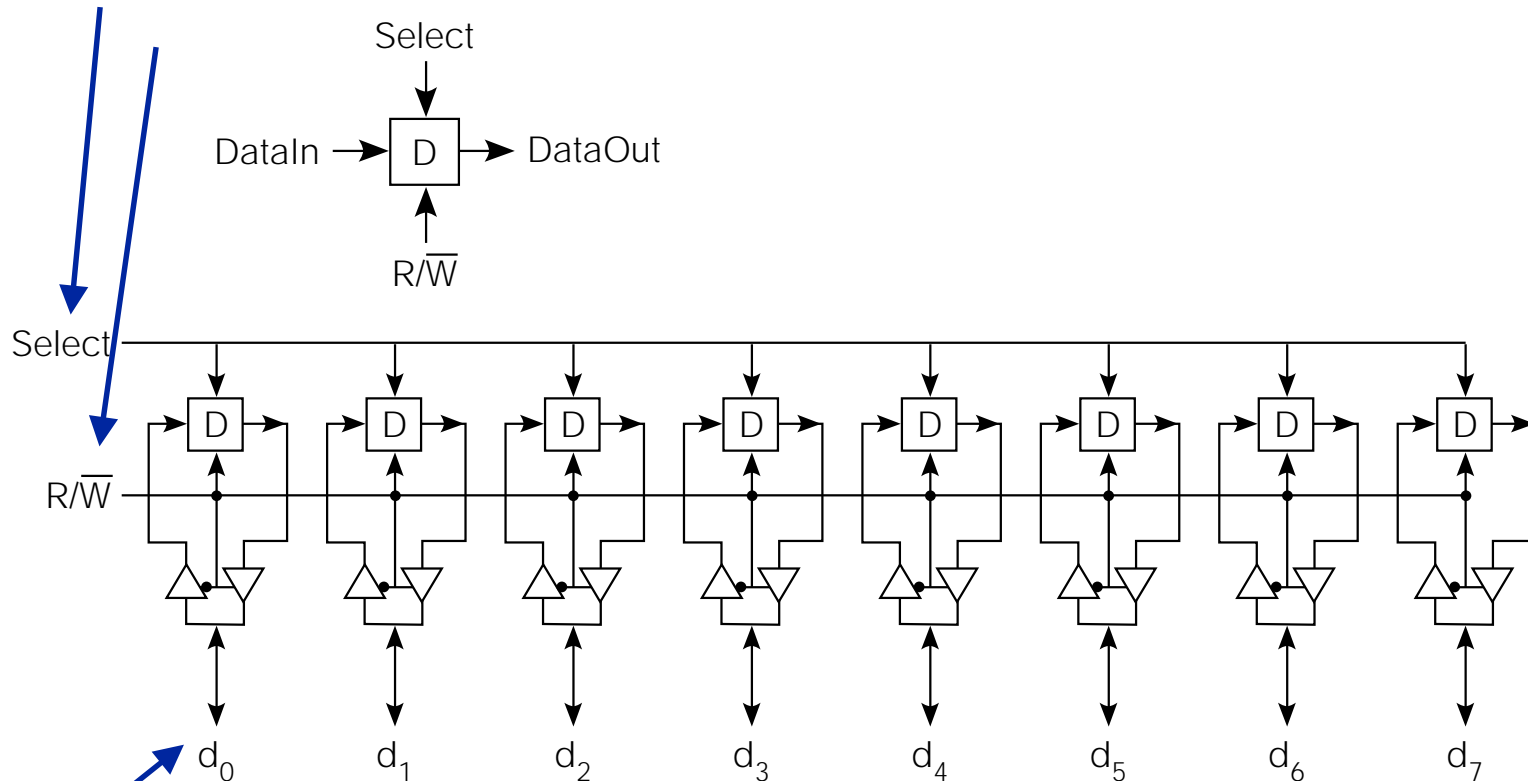
Data bus is bi-directional, and buffered. (Why?)

Fig. 7.5   A 4x8 2D Memory Cell Array

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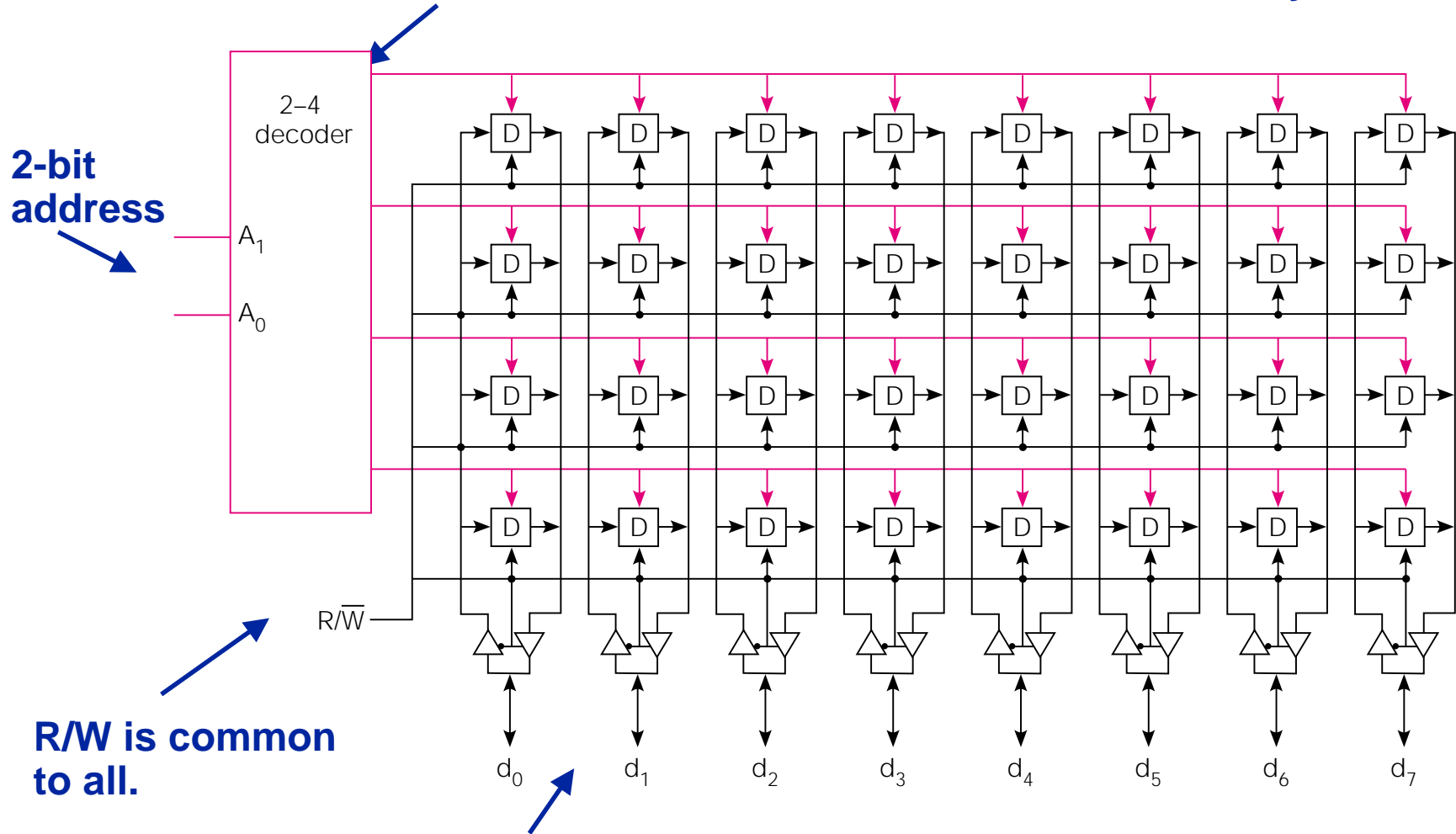
2-bit address

2–4 decoder

A_1

A_0

$R/\overline{W}$

R/W is common to all.

d_0 d_1 d_2 d_3 d_4 d_5 d_6 d_7

Bi-directional 8-bit buffered data bus

Fig. 7.6 A 64Kx1 bit static RAM (SRAM) chip

~square array fits IC design paradigm

Row address: 8   A$_0$–A$_7$

8–256 row decoder

256

256 × 256 cell array

Selecting rows separately from columns means only 256x2=512 circuit elements instead of 65536 circuit elements!

256

Column address: 8   A$_8$–A$_{15}$

1 256–1 mux
1 1–256 demux

R/$\overline{\text{W}}$

CS, Chip Select, allows chips in arrays to be selected individually

CS

1

This chip requires 21 pins including power and ground, and so will fit in a 22 pin package.

Fig 7.7   A 16Kx4 SRAM Chip

Row address: 8
A_0–A_7

8–256
row
decoder

256

4 64 × 256
cell arrays

64 each
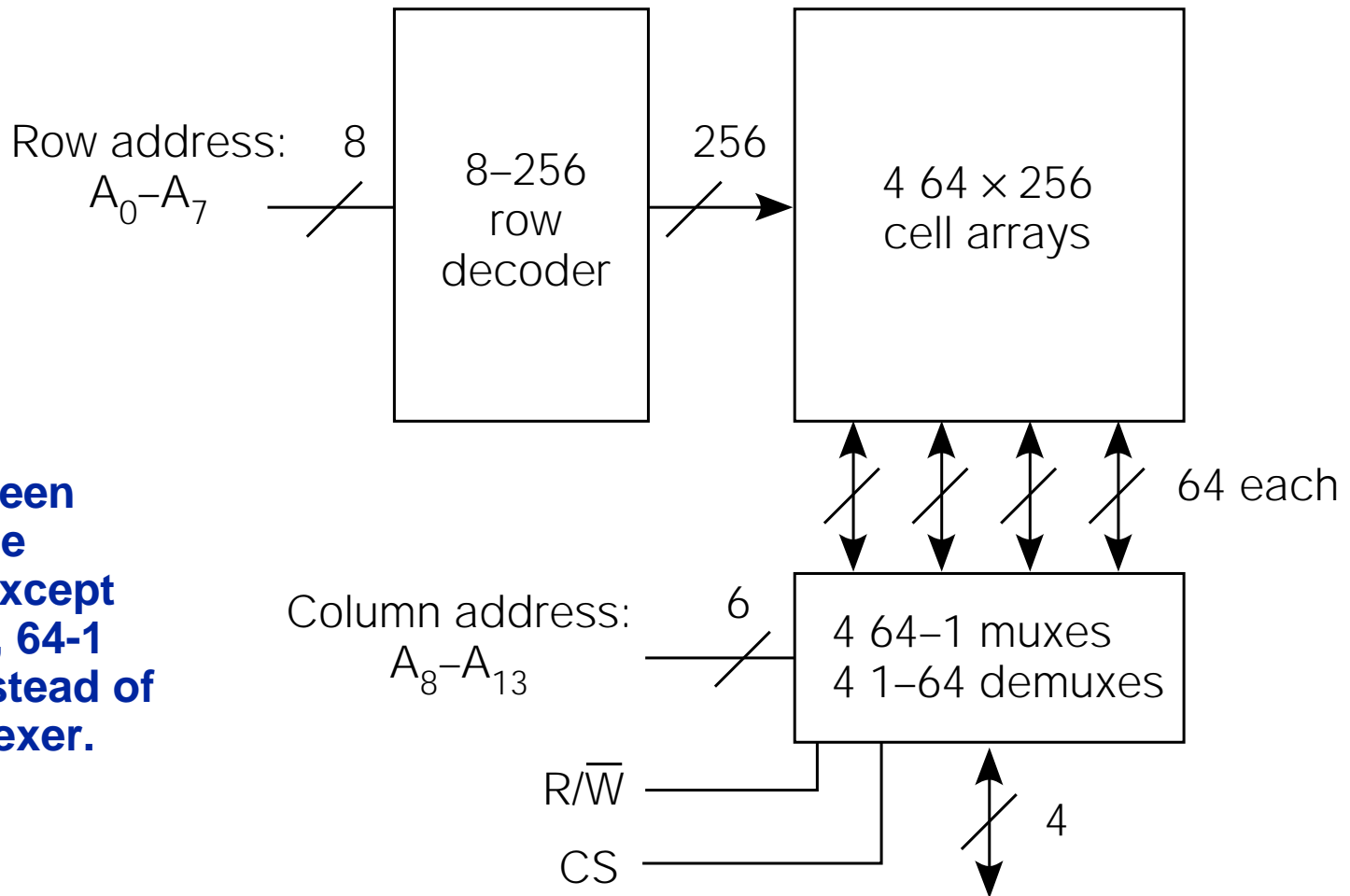
**There is little
difference between
this chip and the
previous one, except
that there are 4, 64-1
Multiplexers instead of
1, 256-1 Multiplexer.**

Column address: 6
A_8–A_{13}

4 64–1 muxes
4 1–64 demuxes

$R/\overline{W}$

CS

4

**This chip requires 24 pins including power and ground, and so will require
a 24 pin pkg. Package size and pin count can dominate chip cost.**

Fig 7.8 Matrix and Tree Decoders

•**2-level decoders are limited in size because of gate fanin.**
Most technologies limit fanin to ~8.
•**When decoders must be built with fanin >8, then additional levels**
 of gates are required.
•**Tree and Matrix decoders are two ways to design decoders with large fanin:**
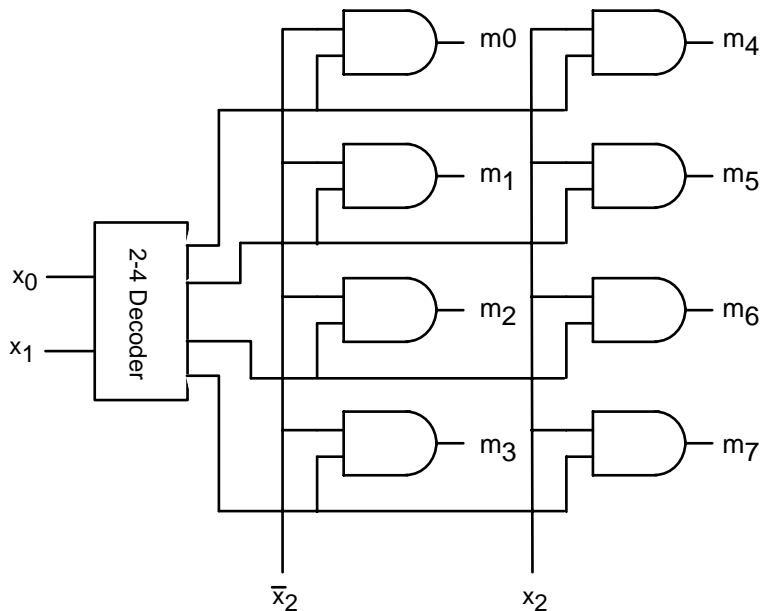
3-to-8 line tree decoder constructed
from 2-input gates.
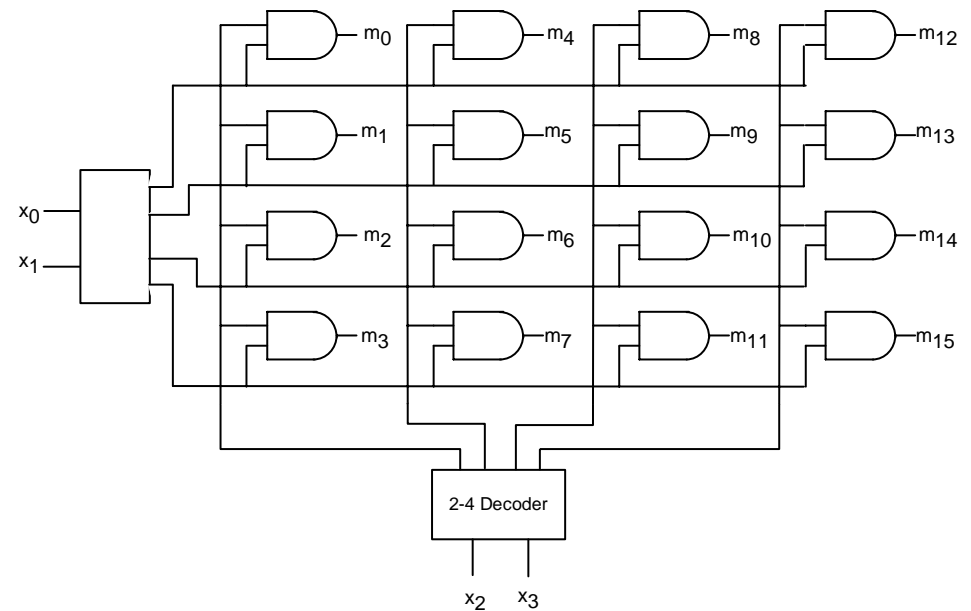
4-to-16 line matrix decoder
constructed from 2-input gates.

Fig 7.9 A 6 Transistor static RAM cell

Dual rail data lines for reading and writing

b_i $\bar{b}_i$

+5

Active loads

Storage cell

Word line w_i

Switches to control access to cell

Additional cells

Column select (from column address decoder)

Sense/write amplifiers — sense and amplify data on Read, drive b_i and $\bar{b}_i$ on write

$R/\overline{W}$

CS

d_i

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.
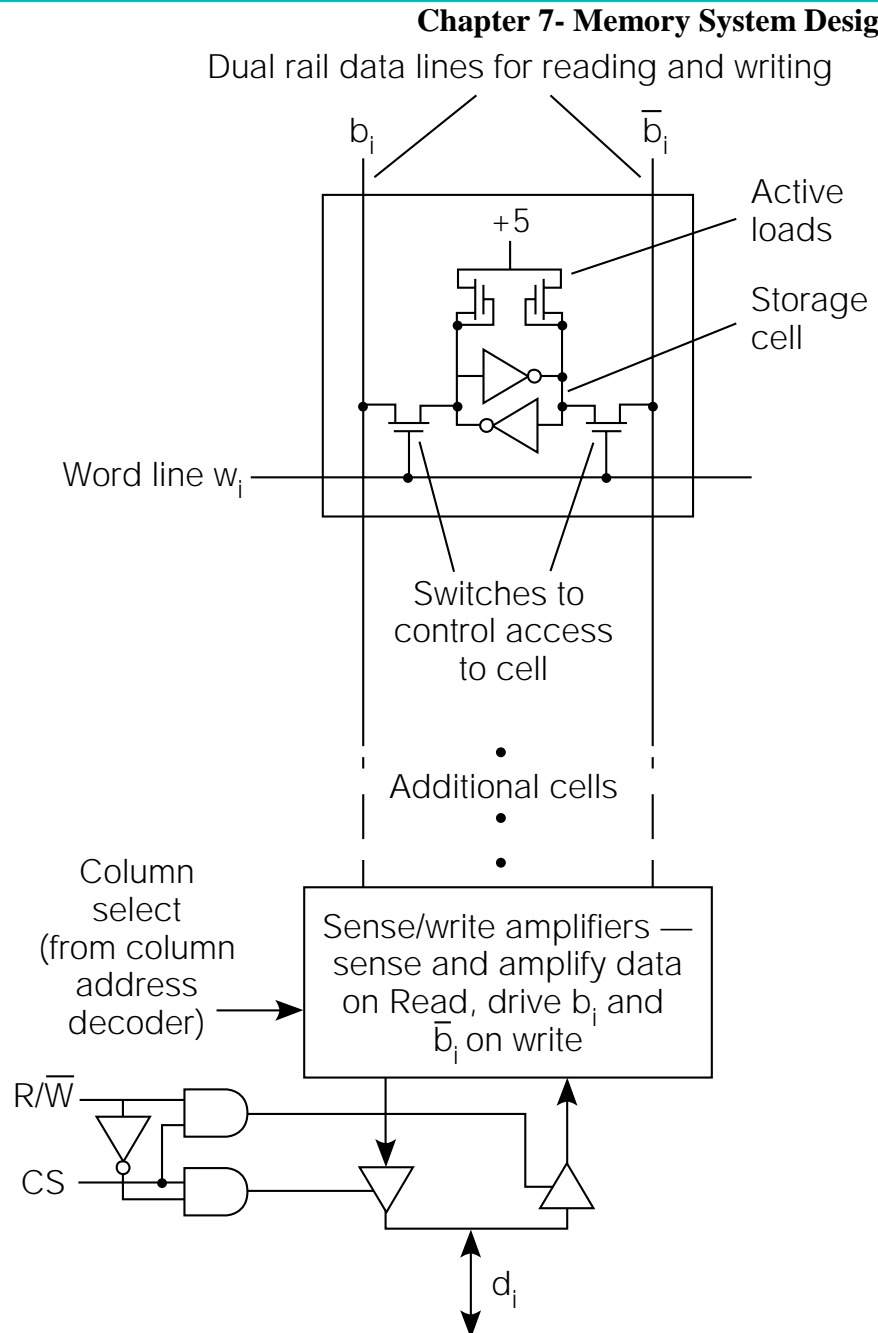
Figs 7.10 Static RAM Read Timing

Memory
address

Read/write

CS

Data

t_{AA}
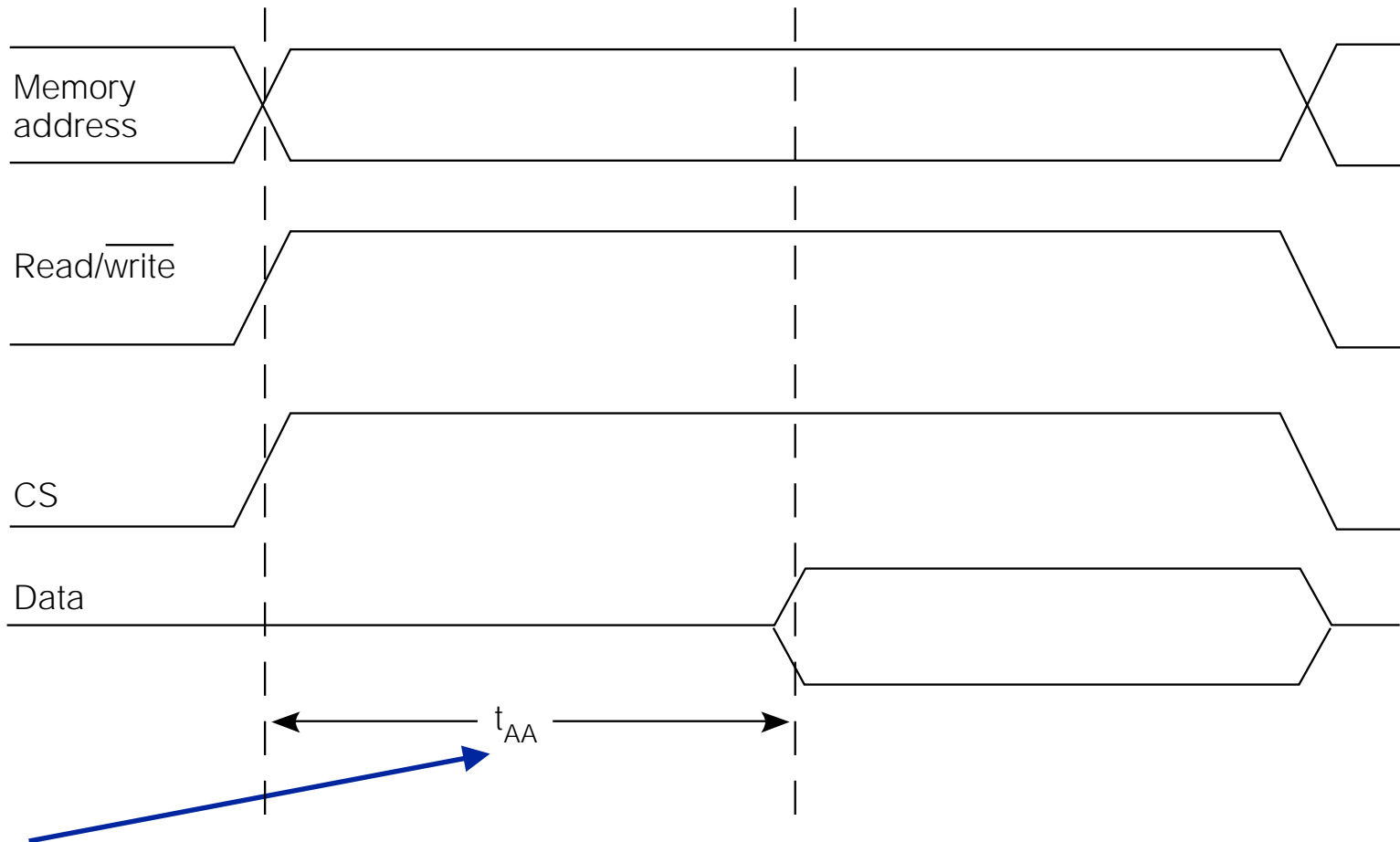
Access time from Address– the time required of the RAM array to
decode the address and provide value to the data bus.

Figs 7.11 Static RAM Write Timing
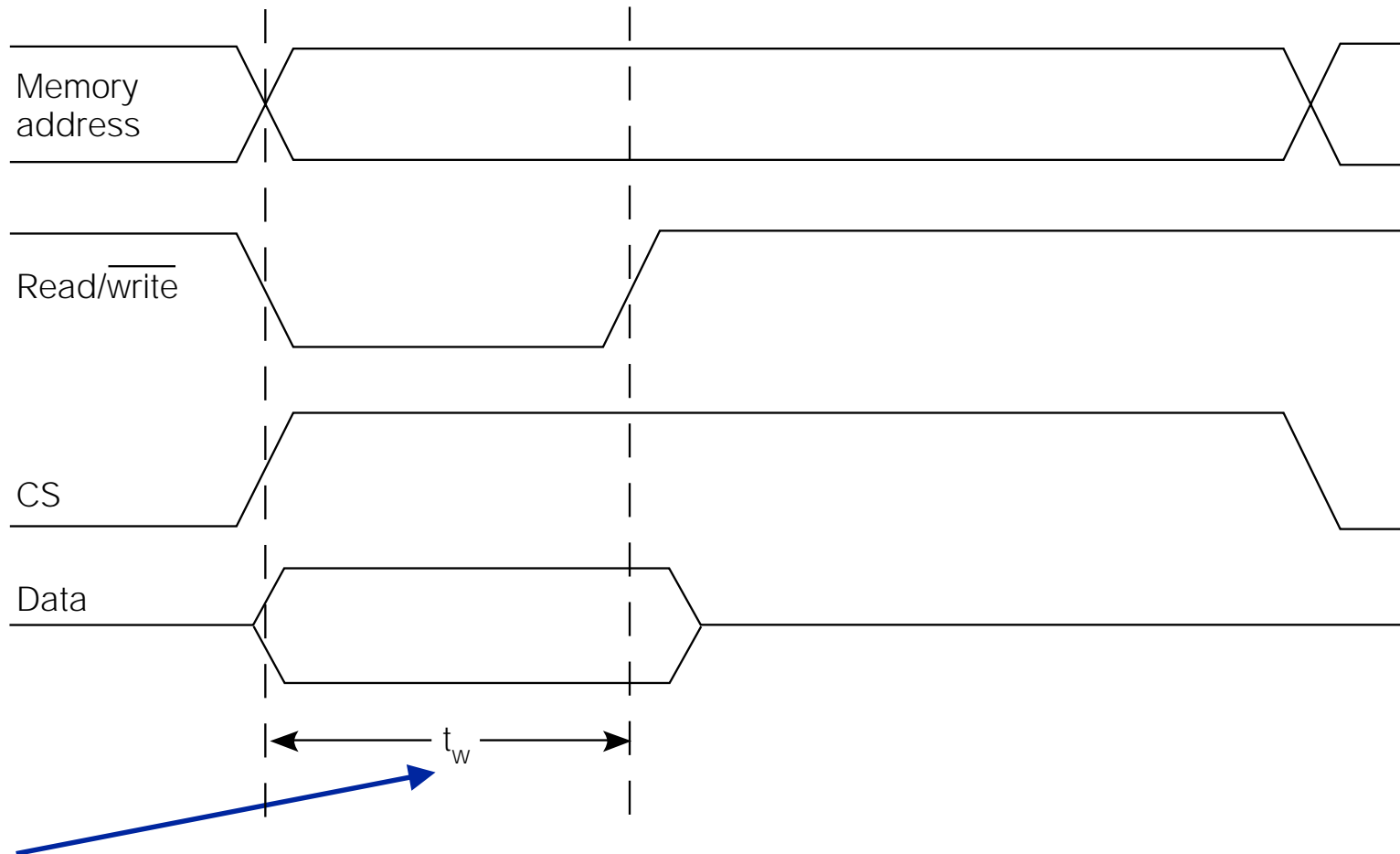
Write time–the time the data must be held valid in order to decode address and store value in memory cells.

Fig 7.12   A Dynamic RAM (DRAM) Cell

Capacitor will discharge in 4-15ms.

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, assert word line, sense value on bit line with sense/amp.

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

Single bit line

b_i

Switch to control access to cell

Capacitor stores charge for a 1, no charge for a 0

t

c

Word line w_j

Additional cells

Column select (from column address decoder)

Sense/write amplifiers — sense and amplify data on Read, drive b_i and $\bar{b}_i$ on write

$R/\overline{W}$

R

CS

W

d_i
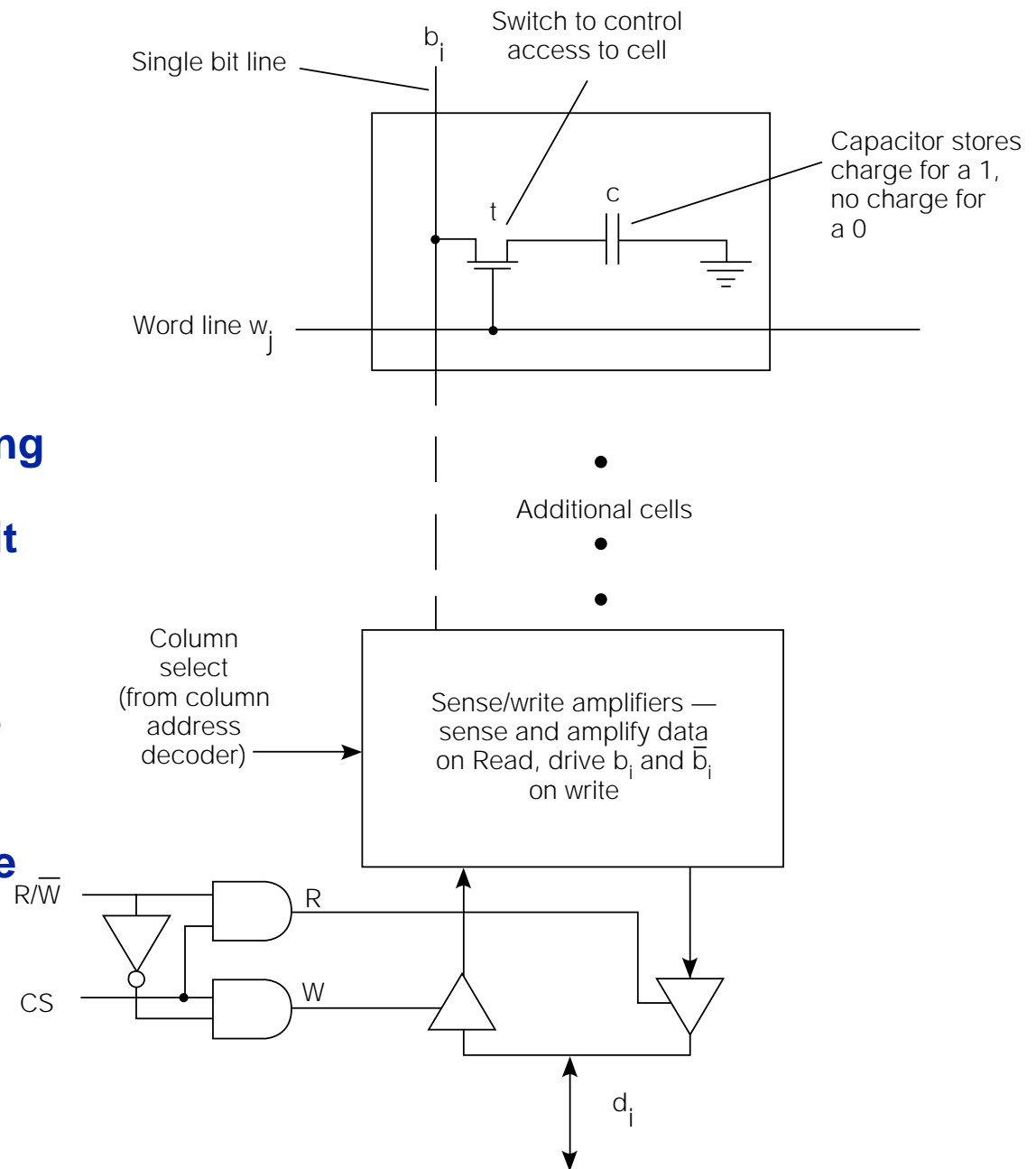
Fig 7.13 DRAM Chip organization

•**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.**

Row latches and decoder

1024

1024×1024 cell array

10

A_0-A_9

$\overline{RAS}$

$\overline{CAS}$

$R/\overline{W}$

Control logic

Control

1024

1024 sense/write amplifiers and column latches

1024

10

10 column address latches, 1–1024 muxes and demuxes

d_o

d_i
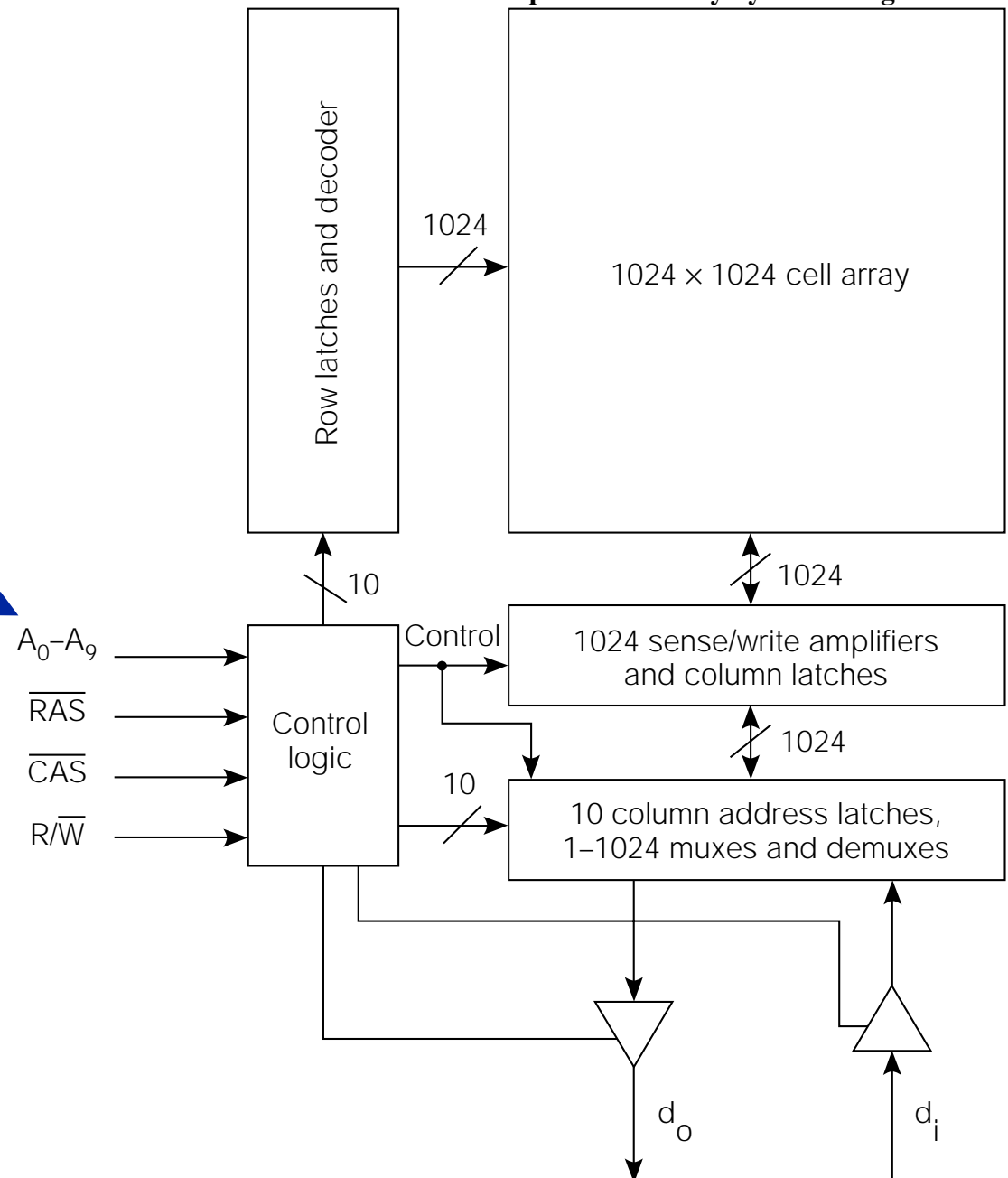
Figs 7.14, 7.15 DRAM Read and Write cycles

Typical DRAM Read operation   **Typical DRAM Write operation**

Memory Address | Row Addr | Col Addr

$\overline{RAS}$ t_{RAS} t_{Prechg}

$\overline{CAS}$

R/$\overline{W}$

Data

t_A t_C

Memory Address | Row Addr | Col Addr

$\overline{RAS}$ t_{RAS} Prechg

$\overline{CAS}$

$\overline{W}$

Data

t_{DHR} t_C
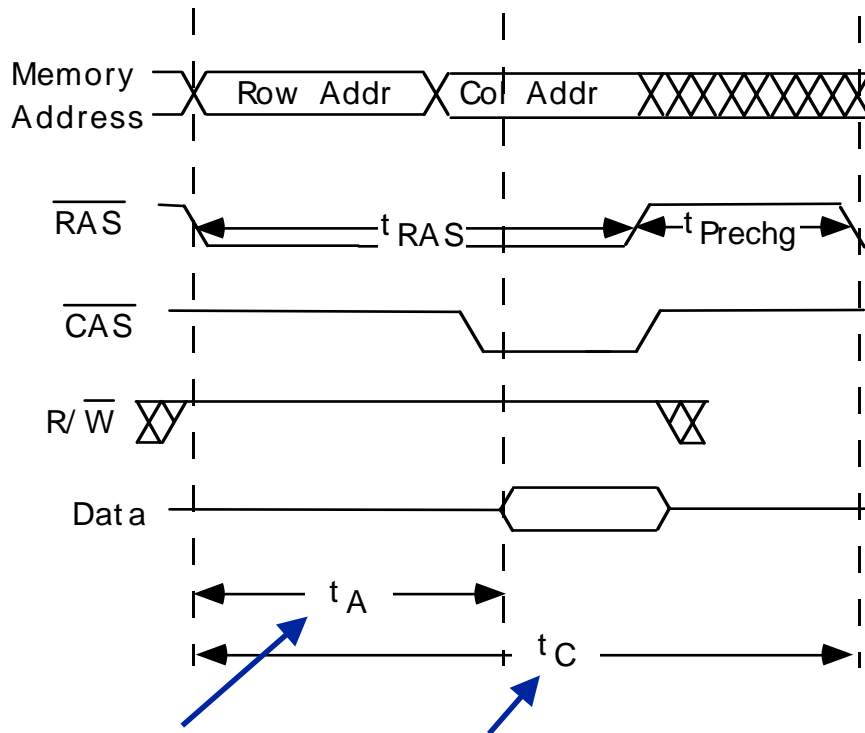
Access time   Cycle time
Notice that it is the bit line precharge operation that causes the difference between access time and cycle time.
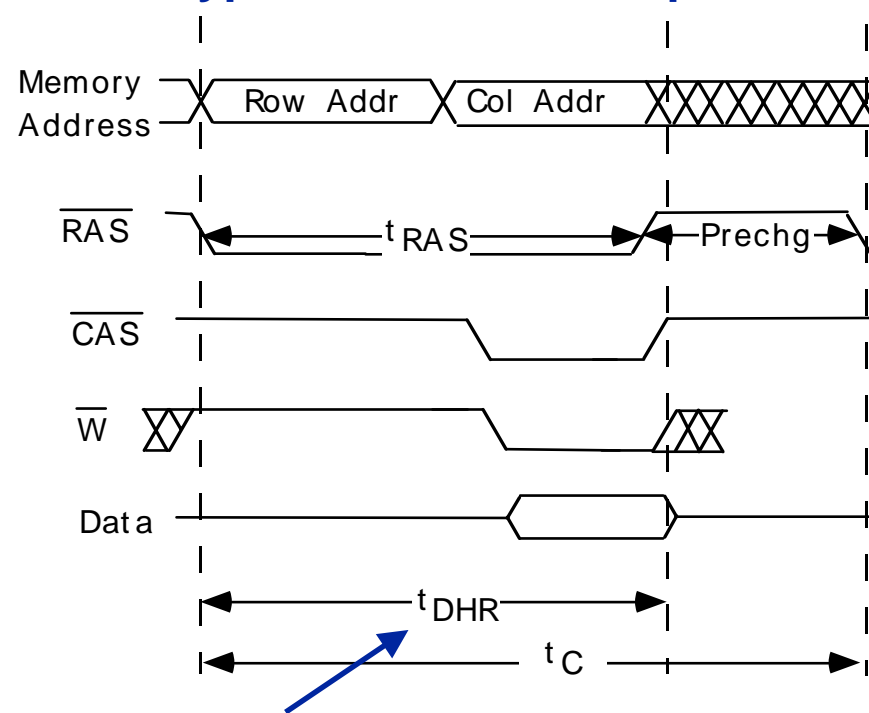
Data hold from RAS.

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.**

Fig 7.16   A CMOS ROM Chip

2-D CMOS ROM Chip

+V

00   Address   Row Decoder

$\overline{CS}$

1   0   1   0
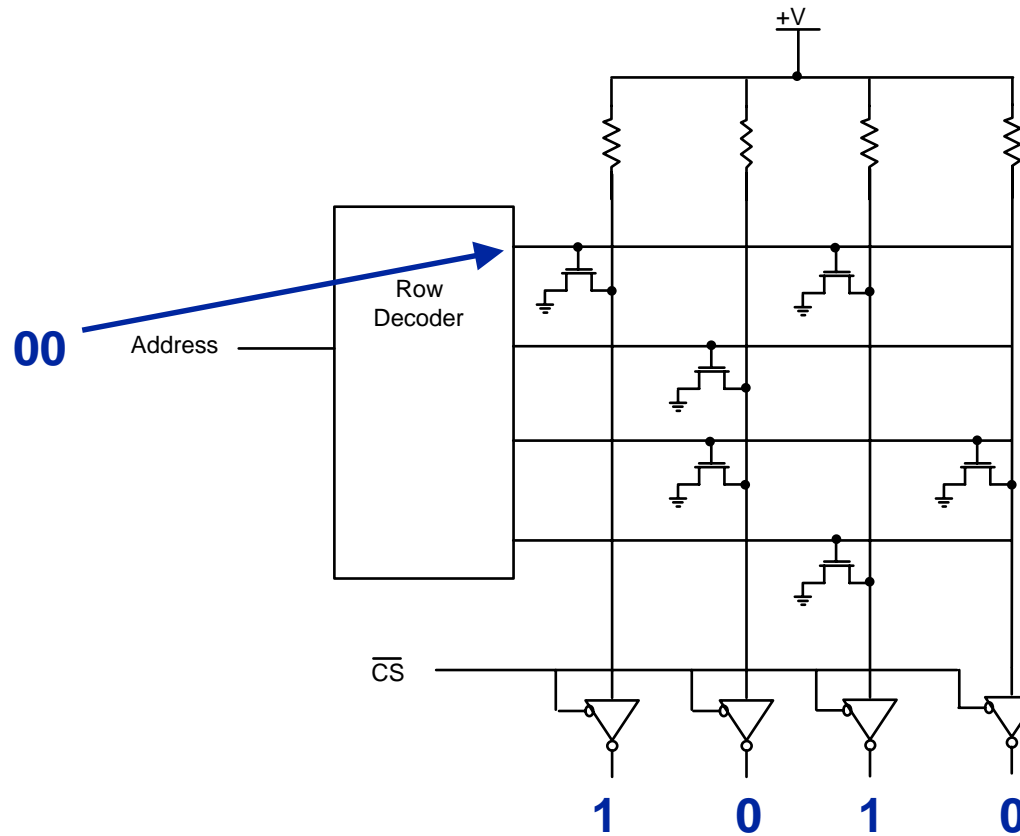
Tbl 7.4   Kinds of ROM

ROM Type	Cost	Programmability	Time to program	Time to erase
Mask pro-grammed	Very inexpensive	At the factory	Weeks (turn around)	N/A
PROM	Inexpensive	Once, by end user	Seconds	N/A
EPROM	Moderate	Many times	Seconds	20 minutes
Flash EPROM	Expensive	Many times	100 us.	1s, large block
EEPROM	Very expensive	Many times	100 us.	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.

How to Build a SIMM (or DIMM)

A<11..0>

DQ<31..0>

RAS.L

CAS.L

OE.L

WE.L

DRAM

A<11..0>

DQ<31..24> DQ<7..0>

RAS.L

CAS.L

OE.L

WE.L

DRAM

A<11..0>

DQ<23..16> DQ<7..0>

RAS.L

CAS.L

OE.L

WE.L

DRAM

A<11..0>

DQ<7..0> DQ<7..0>

RAS.L

CAS.L

OE.L

WE.L

SIMM

A<11..0>

DQ<31..0>

RAS.L

CAS.L

OE.L

WE.L

Computer Systems Design and Architecture by V. Heuring and H. Jordan   © 1997 V. Heuring and H. Jordan: Updated David M. Zar, February, 2001

SRC DRAM Design

SRC

SRC		
D<31..0>		
A<1..0>		
A<24..13>	A<11..0>	Y
A<12..2>	B<10..0>	
	B<11>	
	A.L/B.L	
A<31..25>	ROW.L	
WRITE.H	RAS.L	
READ.H	CAS.L / OE.L / WE.L	
DONE.H	DONE..H	
CLK	CLK	

A<11..0>
DQ<31..0>
RAS.L
CAS.L
OE.L
WE.L

SRC DRAM Timing

SRC DRAM Design with Refresh

SRC

D<31..0>

A<1..0>

A<24..13>   A<11..0>

                                    Y

A<12..2>   B<10..0>

                    B<11>

                    A.L/B.L

A<11..0>
DQ<31..0>
RAS.L
CAS.L
OE.L
WE.L

A<31..25>   ROW.L

WRITE.H   RAS.L
                    CAS.L
                    OE.L
READ.H   WE.L

DONE.H   DONE..H
                    GRNT.H
                    RQST.H

Refresh Counter

CLK   CLK
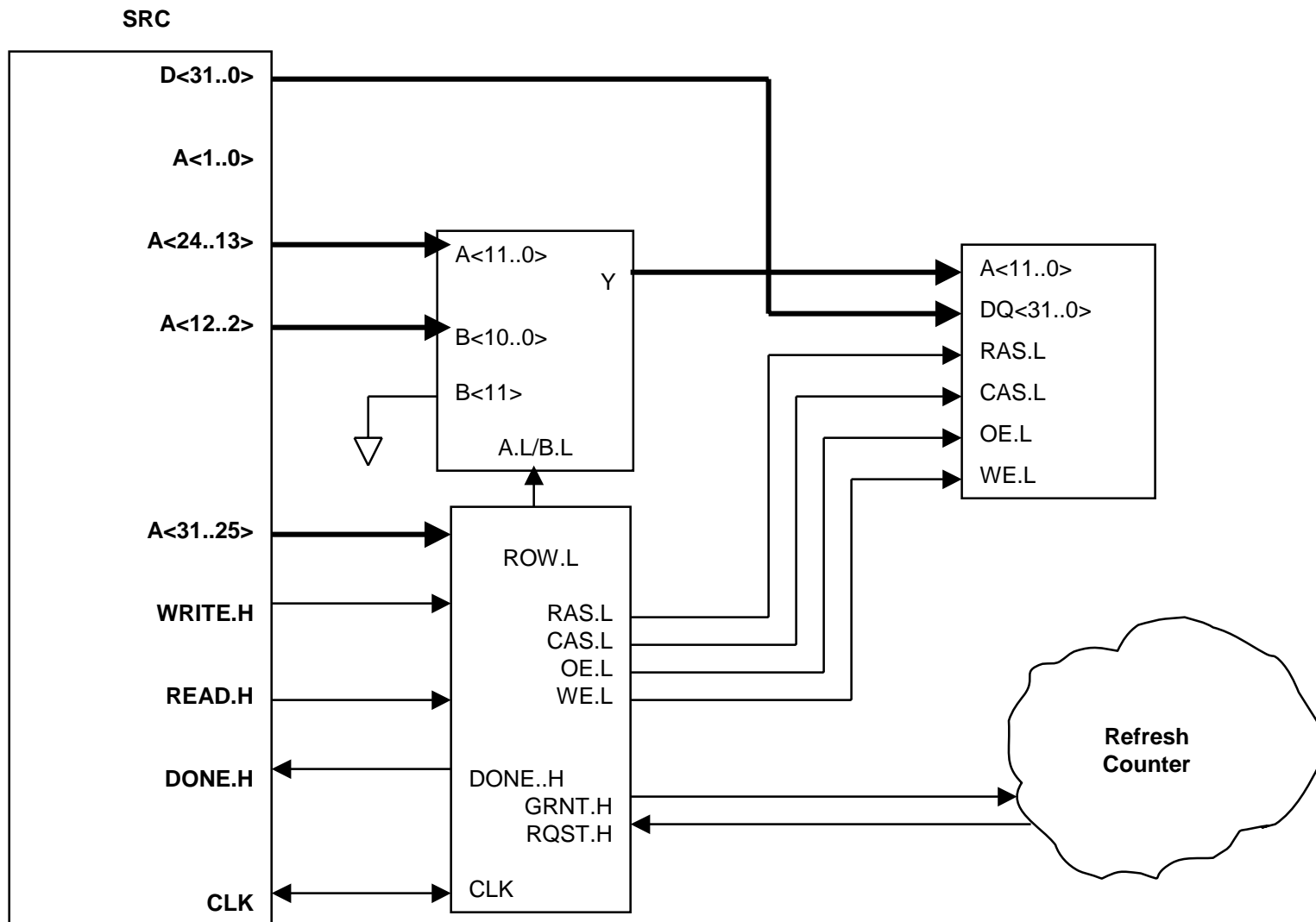
Refresh Counter

- **Each row needs to be refreshed every R** μ**s**
- **There are N rows in the DRAM so every R/N** μ**s we need to refresh one of them.**
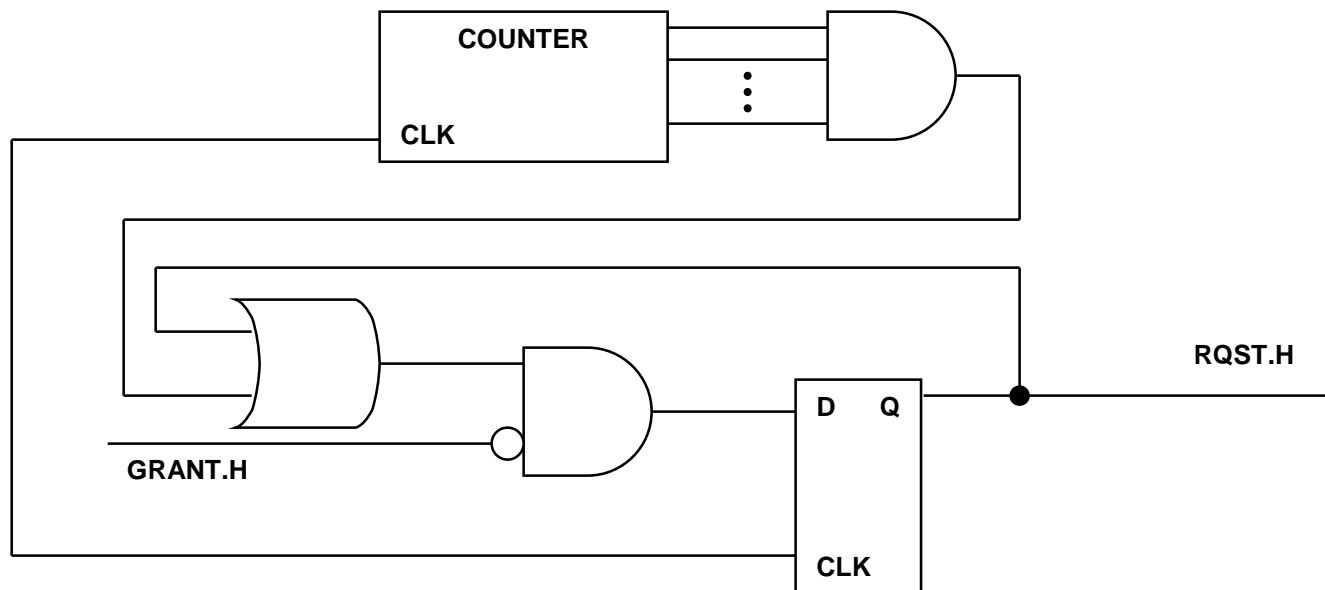
Memory Controller State Machine
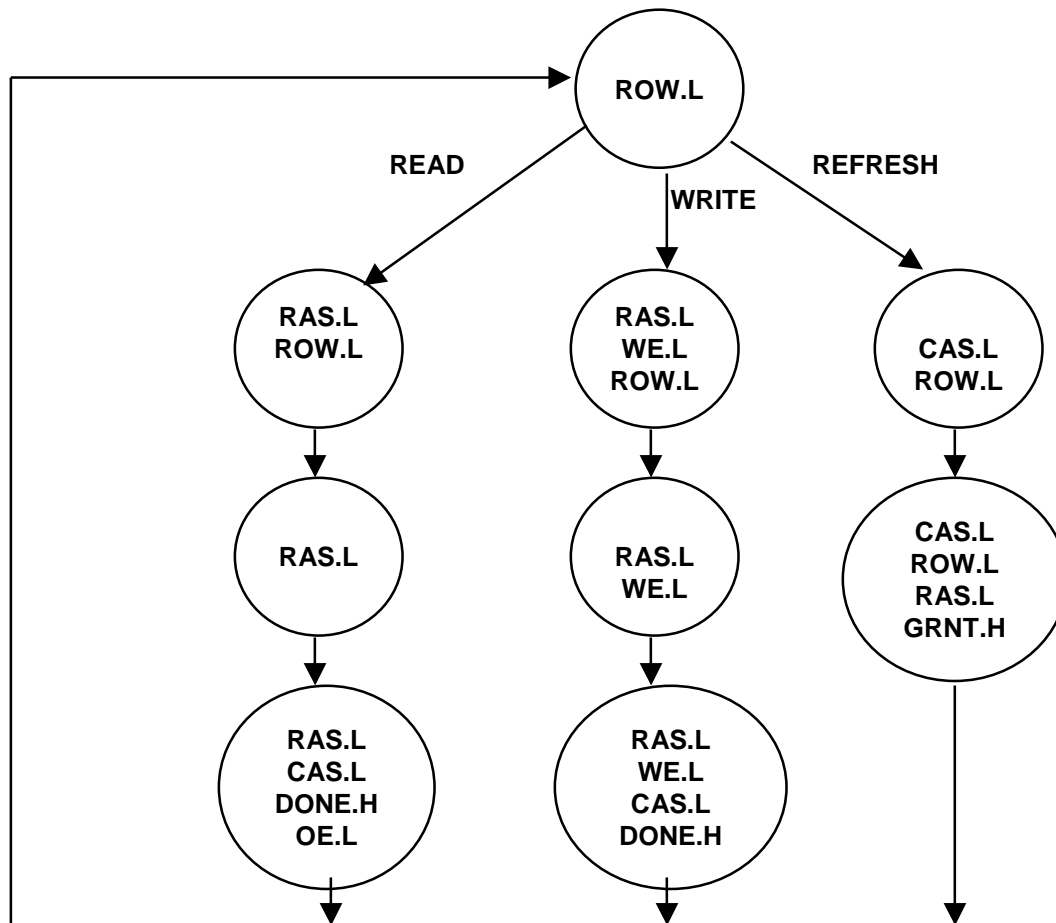

```
READ = READ.H * RQST.H' * A<31> * A<30> * A<29> * A<28> * A<27> * A<26> * A<25>
WRITE = WRITE.H * RQST.H' * A<31> * A<30> * A<29> * A<28> * A<27> * A<26> * A<25>
REFRESH = RQST.H
```

Fig 7.17   General structure of memory chip

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

Multiple chip selects ease the assembly of chips into chip arrays. Usually provided by an external AND gate.

Chip   Selects

. . .

Address

m

Address Decoder

Memory Cell Array

I/O Multiplexer

R/ $\overline{W}$

s

s

s

$\equiv$

. . .

CS

R/ $\overline{W}$

m

Address

Data

s

Data

Bi-directional data bus.
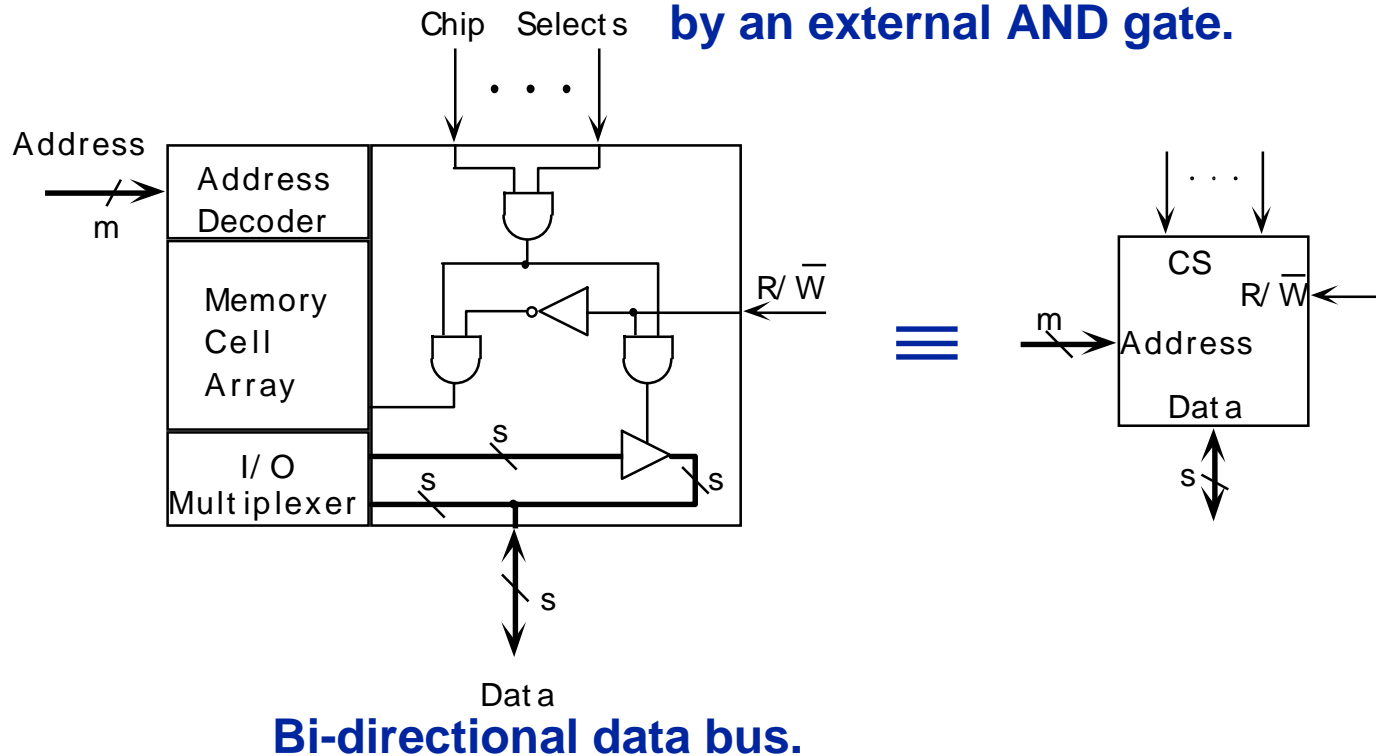
Fig 7.18   Word Assembly from Narrow Chips

All chips have common CS, R/W, and Address lines.

Select

Address

R/ $\overline{W}$

| CS |
| R/ $\overline{W}$ |
| Address |
| Data |

s

| CS |
| R/ $\overline{W}$ |
| Address |
| Data |

s

• • •

| CS |
| R/ $\overline{W}$ |
| Address |
| Data |

s

p×s
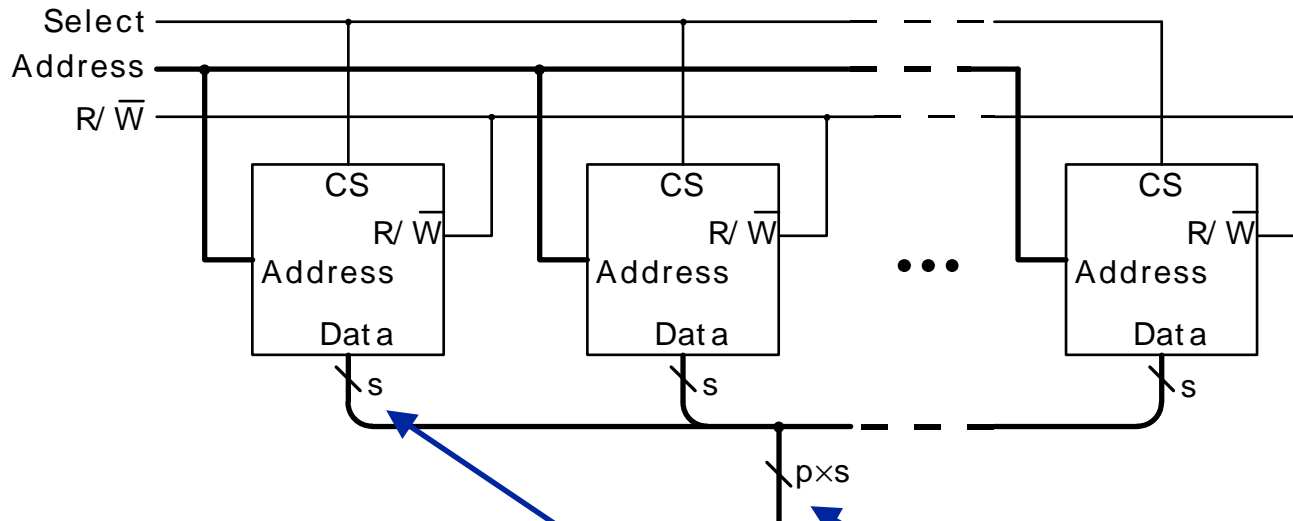
P chips expand word size from s bits to p x s bits.

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:

Address

m+k

k to 2^k
Decoder

k

m

R/$\overline{W}$

CS
R/$\overline{W}$
Address
Data

CS
R/$\overline{W}$
Address
Data

CS
R/$\overline{W}$
Address
Data

s

s

s

s
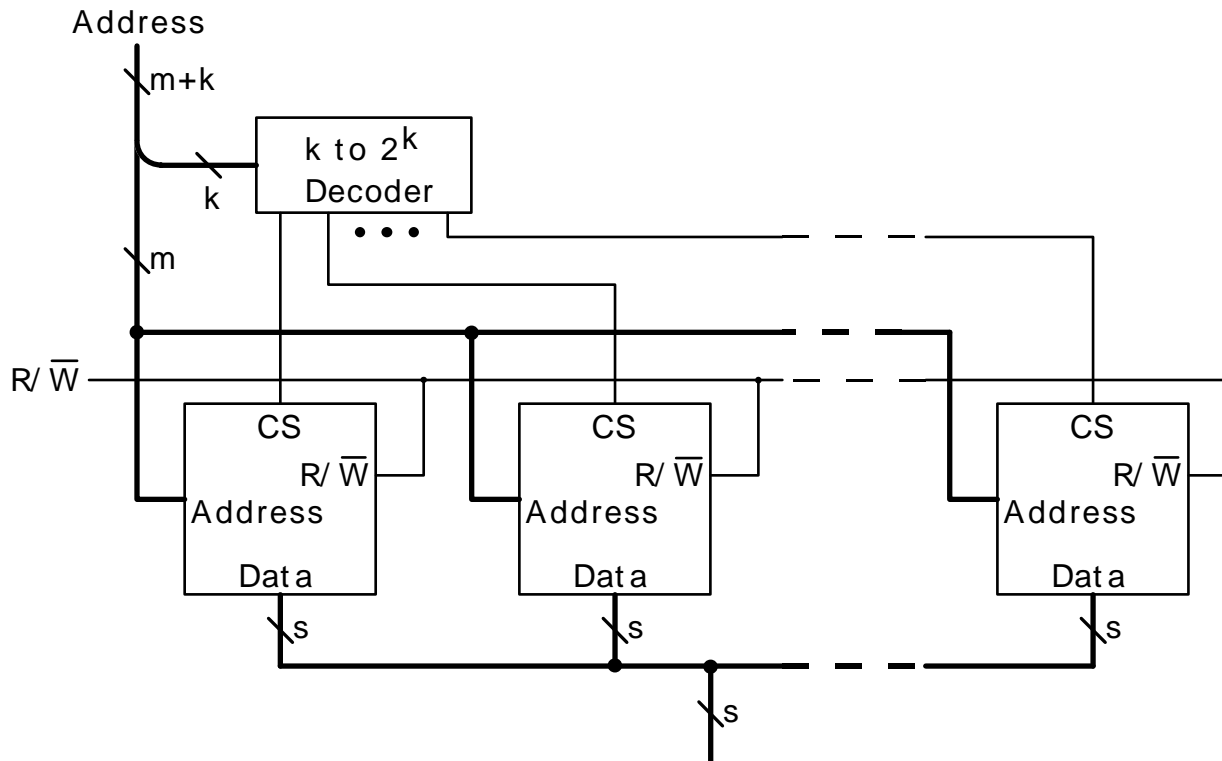
Word size remains at s bits.

Fig 7.20 Chip Matrix Using Two Chip Selects

Address

$m + q + k$

Horizontal decoder

k

m

R/W

CS1   CS2

R/$\overline{W}$

Address

Data

Vertical decoder

q

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.

s

One of 2^{m+q+k} s-bit words
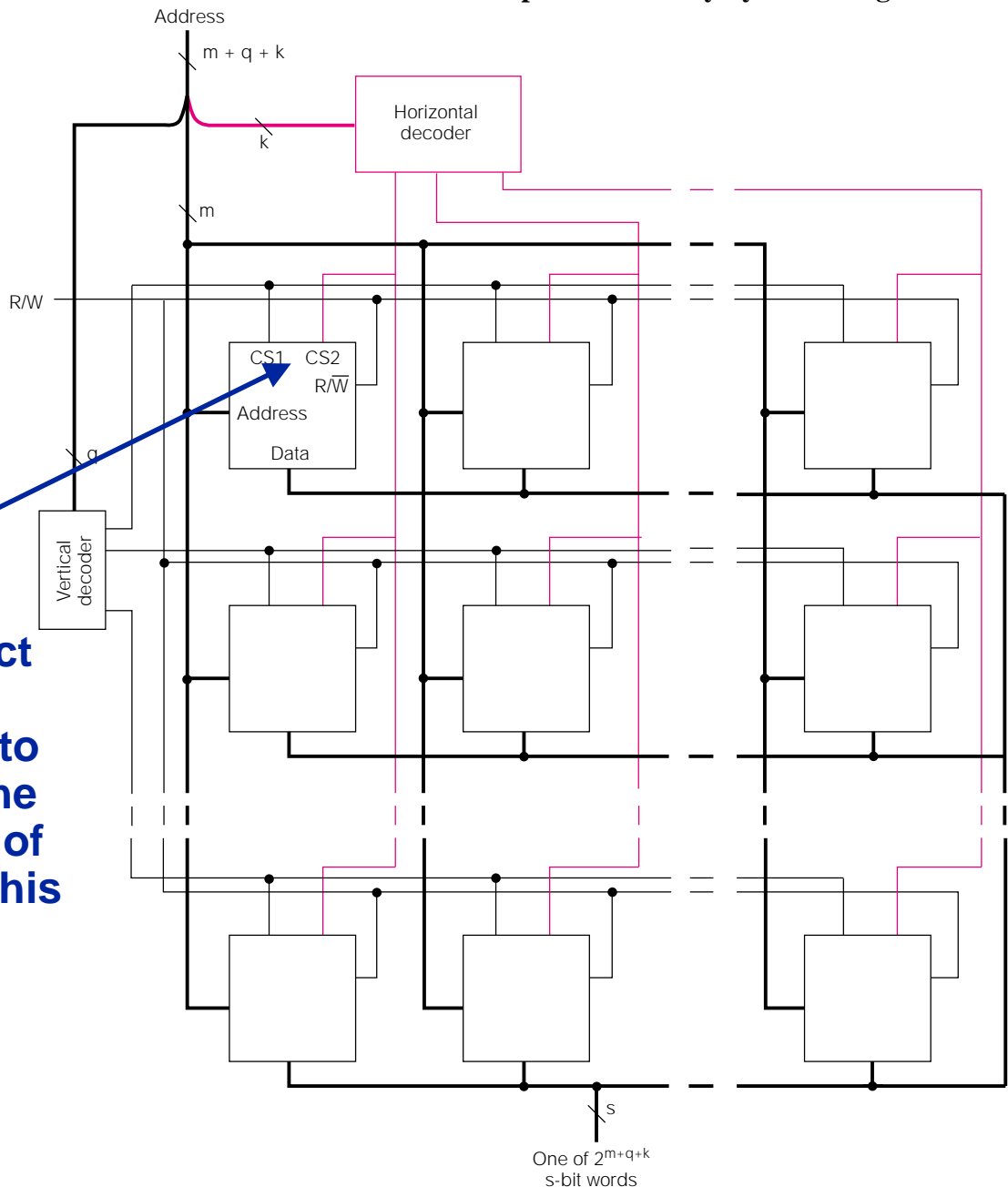
Fig 7.21   A 3-D DRAM Array

$\overline{CAS}$ ——————————————— Enable

k_c

2^{k_c} decoder

· · ·

High address $k_c + k_r$

k_r

2^{k_r} decoder

2^{k_r} decoder

$\overline{RAS}$

$R/\overline{W}$

Multiplexed address   m/2

$\overline{RAS}$ $\overline{CAS}$

$R/\overline{W}$

Address

Data

$\overline{RAS}$ $\overline{CAS}$

$R/\overline{W}$

Address

Data
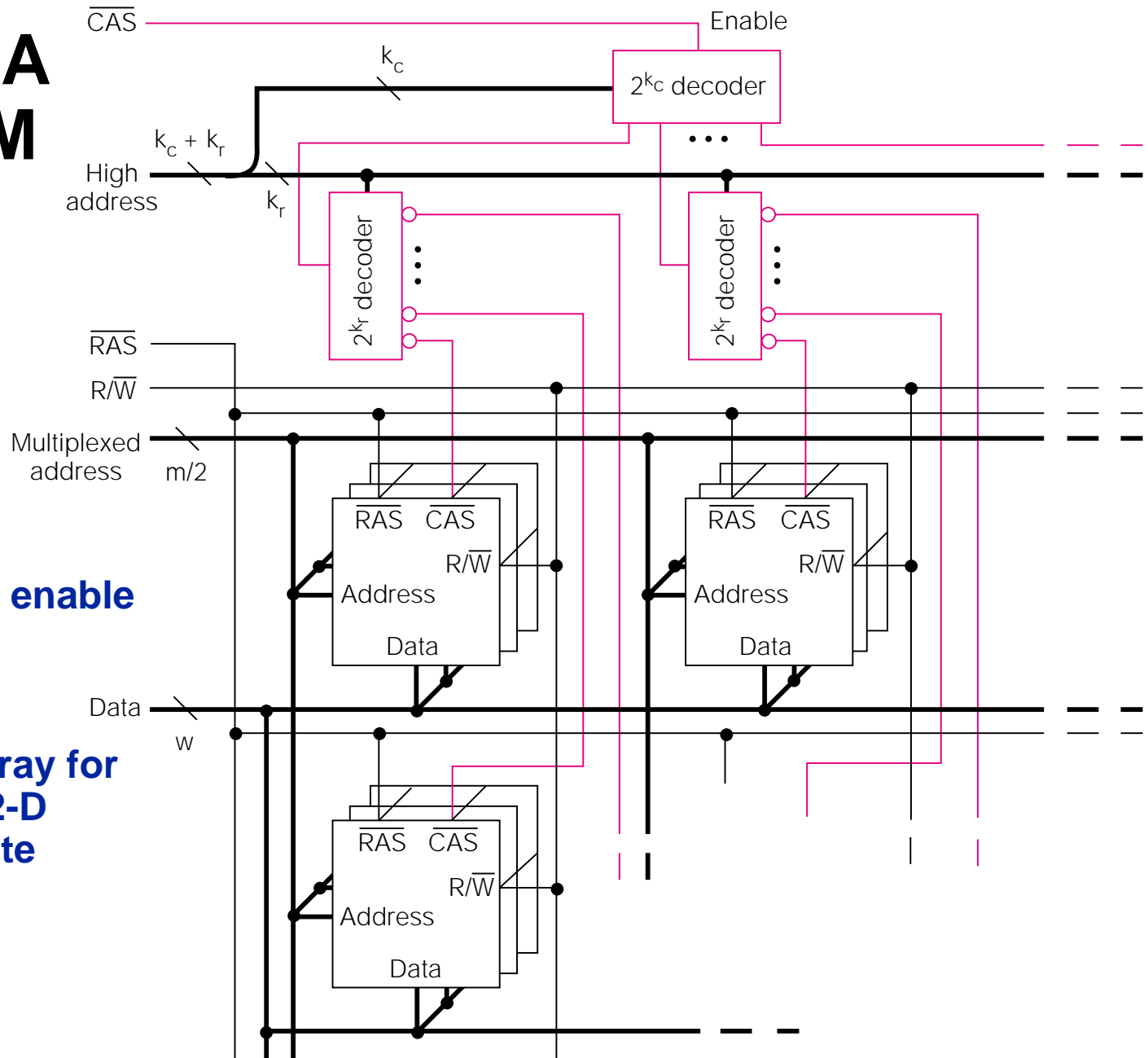
•**CAS is used to enable top decoder in decoder tree.**

Data   w

•**Use one 2-D array for each bit. Each 2-D array on separate board.**

$\overline{RAS}$ $\overline{CAS}$

$R/\overline{W}$

Address

Data

Fig 7.22   A Memory Module interface

Must provide–

- **•Read and Write signals.**
- **•Ready: memory is ready to accept commands.**
- **•Address–to be sent with Read/Write command.**
- **•Data–sent with Write or available upon Read when Ready is asserted.**
- **•Module Select–needed when there is more than one module.**

Bus Interface:
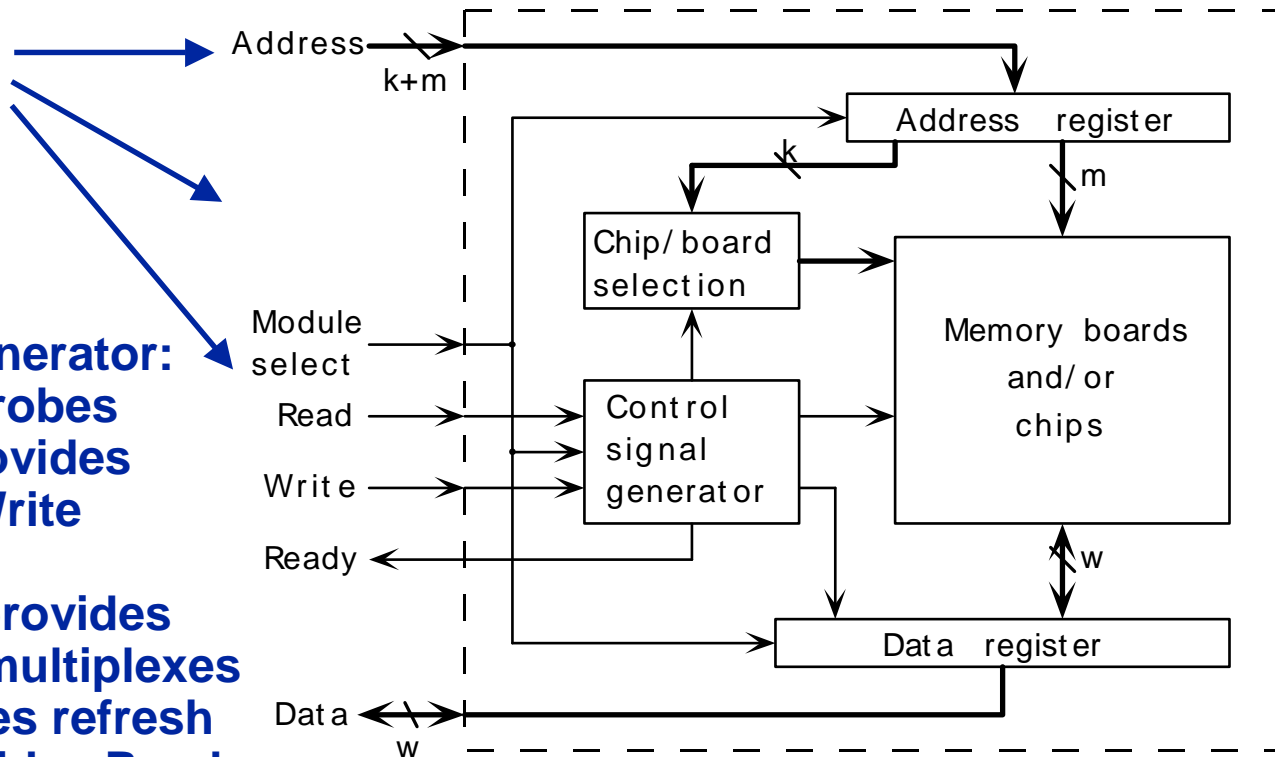
Control signal generator: for SRAM, just strobes data on Read, Provides Ready on Read/Write

For DRAM–also provides CAS, RAS, R/W, multiplexes address, generates refresh signals, and provides Ready.

Address

k+m

Address   register

k

m

Chip/ board
selection

Module
select

Memory boards
and/ or
chips

Read

Control
signal
generator

Write

Ready

w

Data   register

Data

w

Fig 7.23   DRAM module with refresh control

Address ———————————/————————————————————————→
                          k+m
                                                                    ┌─────────────────────────────────┐
                                                                    │   Address   Register   │
                                                                    └─────────────────────────────────┘
                          ┌──────────────┐   k   │   m/2   │ m/2
                          │ Chip/ board │ ←——/——————   │   │
                          │ selection   │   │   │
                          └──────────────┘   m/2   │   │
                                                         ┌─────────────│─────────────────┐   │
    ┌──────────────┐   ┌──────────────────────┐   │ ┌───────────────────────────┐
    │ Refresh   │———————   │   Refresh counter   │   │ │   Address   │
    │ clock and   │   └──────────────────────┘   └—→│   Multiplexer   │
    │ control   │———————————————   └───────────────────────────┘
    └──────────────┘   │   2   │ m/2
            │   ↑   │   ──/──   │
            │   │   │   ▼
            │   │   │   ┌──┐
            │   │   └———→│ Board and   Address   lines   │
  Module   │   │   │ chip selects   │
  select ———│——————   │   │   │
            ↓   R e G R s f e   │   Dynamic   │
  Read ——→┌──────────────┐   │ RAS   RAM Array   │
          │ Memory   │   │   │
          │ timing   │   │ CAS   │
  Write —→│ generator   │   │   │
          │   │   │ R/W̄   Data   lines   │
          └──────────────┘   └──┘
            │   │
  Ready ←———   │ ↕/ w
                                                         ┌────────────────────────────┐
                                                         │   Data   register   │
                                                         └────────────────────────────┘
          w   ↑
  Data ←——/——

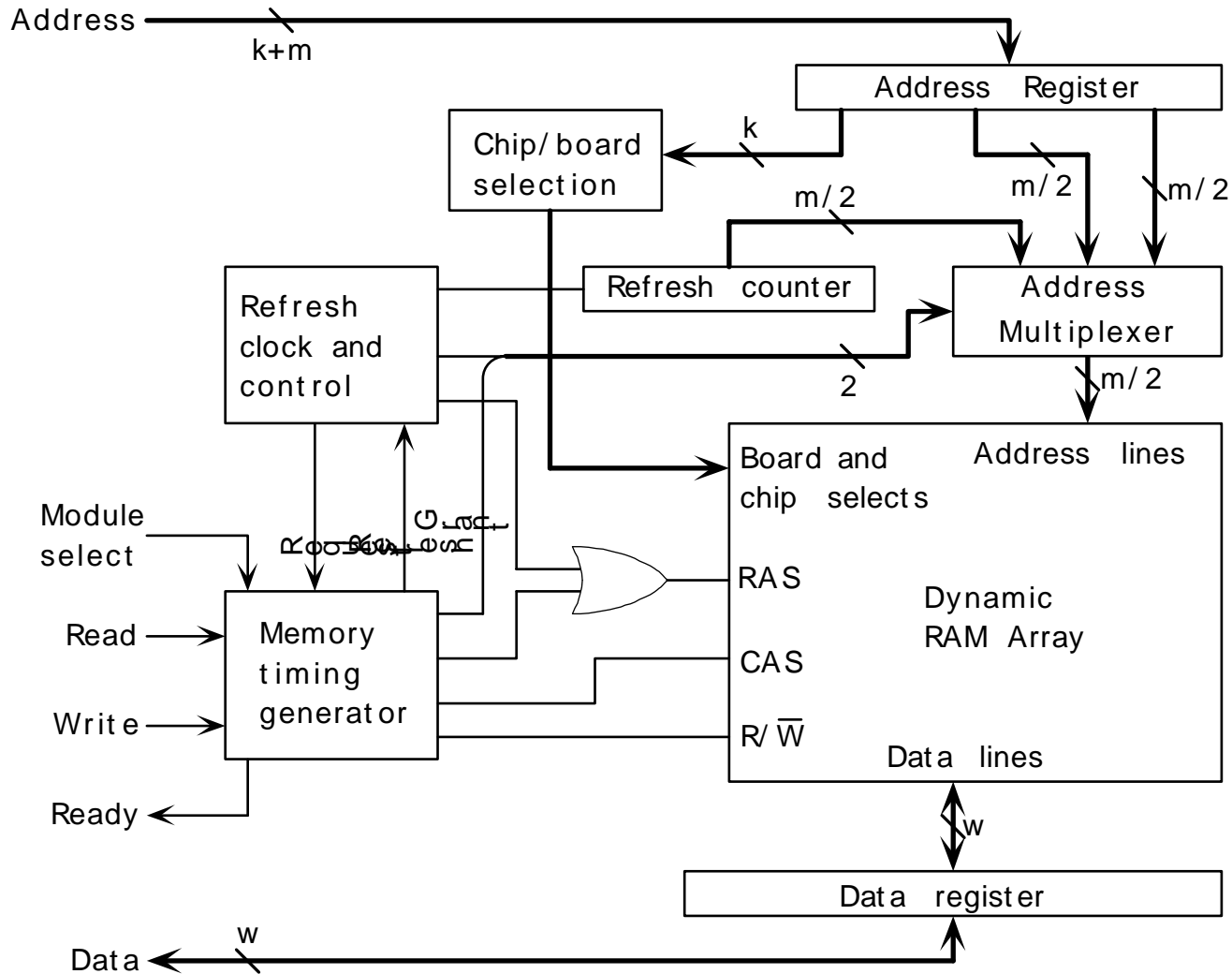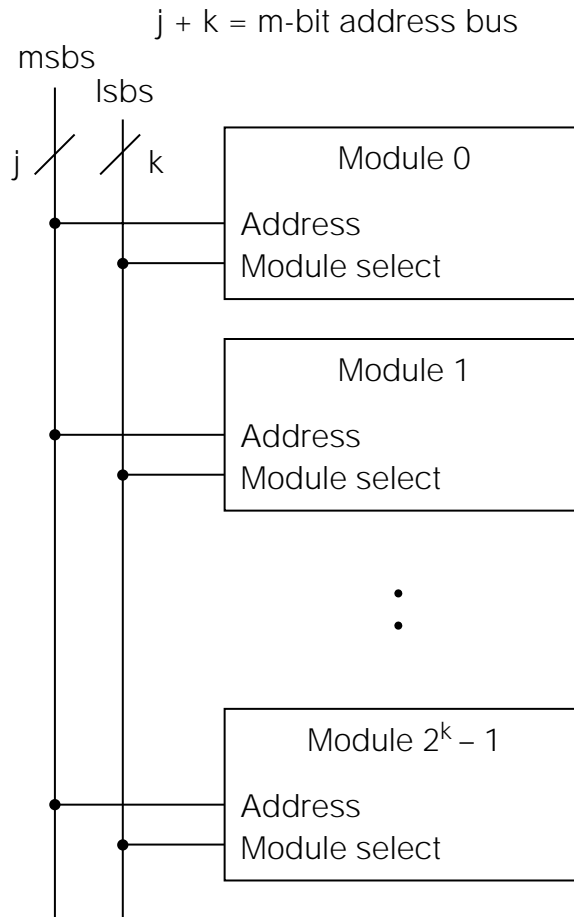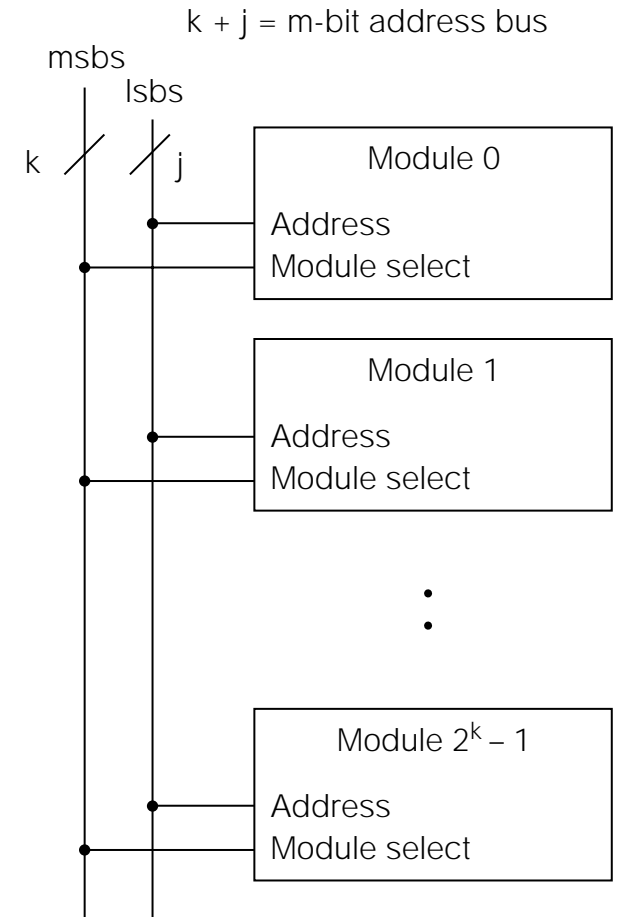# Fig 7.24 Two Kinds of Memory Module Organiz'n.

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

$j + k$ = m-bit address bus

msbs
lsbs

j / / k

Module 0

Address
Module select

Module 1

Address
Module select

.
.
.

Module $2^k - 1$

Address
Module select

(a) Consecutive words in consecutive modules (interleaving)

$k + j$ = m-bit address bus

msbs
lsbs

k / / j

Module 0

Address
Module select

Module 1

Address
Module select

.
.
.

Module $2^k - 1$

Address
Module select

(b) Consecutive words in the same module

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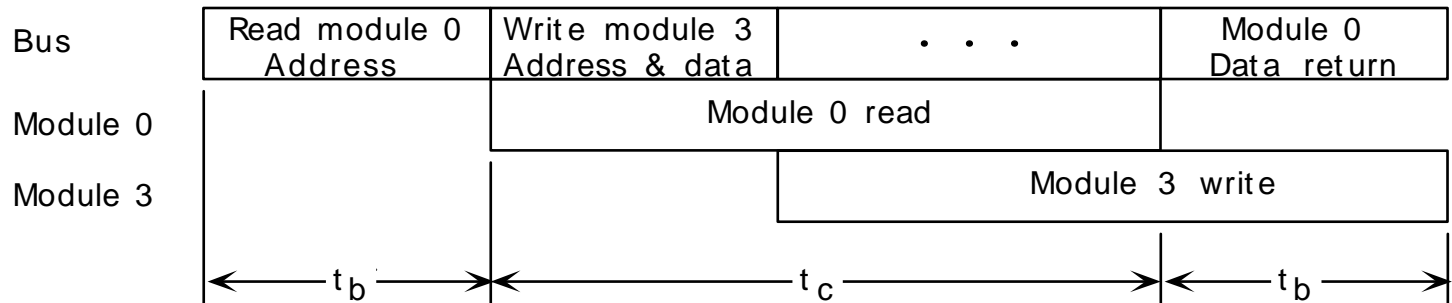**Example: store one word of each cache line in a separate module.**

Main Memory Address:

Word	Module No.

This provides successive words in successive modules.

Timing:

Bus	Read module 0 Address	Write module 3 Address & data	. . .	Module 0 Data return
Module 0		Module 0 read		
Module 3			Module 3 write	

$\leftarrow t_b \rightarrow \quad \leftarrow \qquad\qquad t_c \qquad\qquad \rightarrow \quad \leftarrow t_b \rightarrow$

With interleaving of 2^k modules, and $t_b < t_b/2k$, 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.)
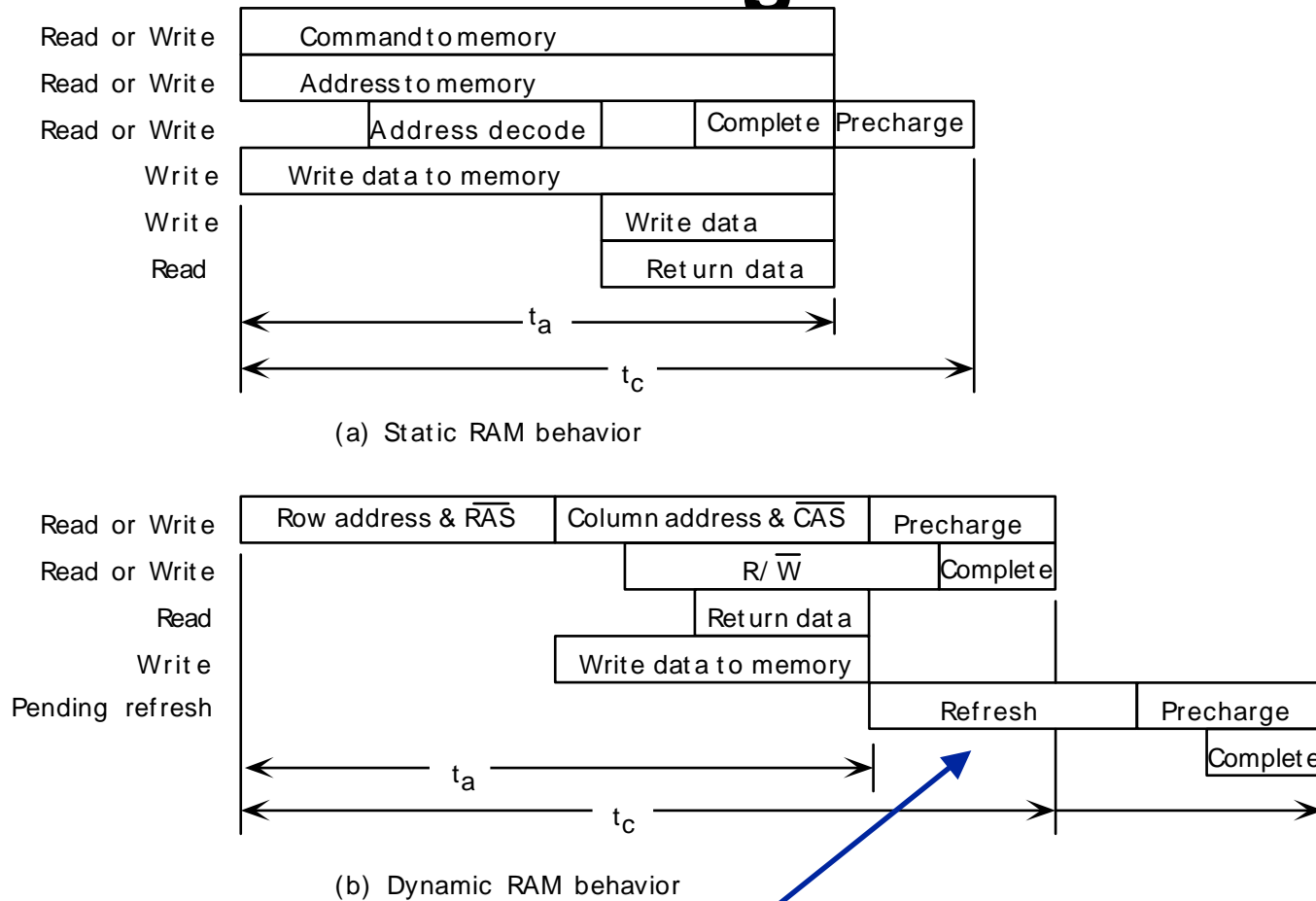- decoding of address by memory

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   Static and dynamic RAM timing

Read or Write	Command to memory		
Read or Write	Address to memory		
Read or Write		Address decode	Complete Precharge
Write	Write data to memory		
Write			Write data
Read			Return data

t_a

t_c

(a) Static RAM behavior

Read or Write	Row address & $\overline{RAS}$	Column address & $\overline{CAS}$	Precharge
Read or Write		R/$\overline{W}$	Complete
Read		Return data	
Write		Write data to memory	
Pending refresh			Refresh Precharge
			Complete

t_a

t_c

(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: 20ns.**

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: 70ns chips to not necessarily provide 70ns 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.
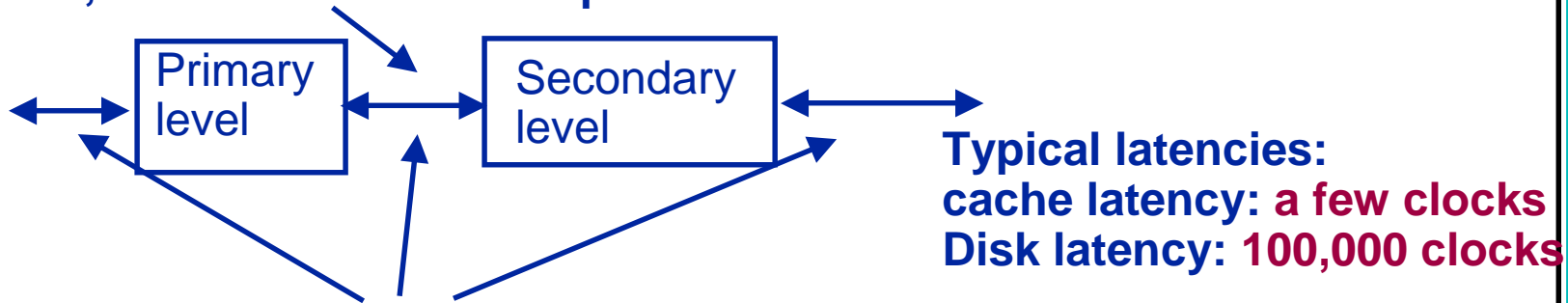
Defining the Primary and Secondary levels:

Faster, smaller   **Slower, larger**

| CPU | • • • | Primary level | Secondary level | • • • |

two adjacent levels in the hierarchy

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.
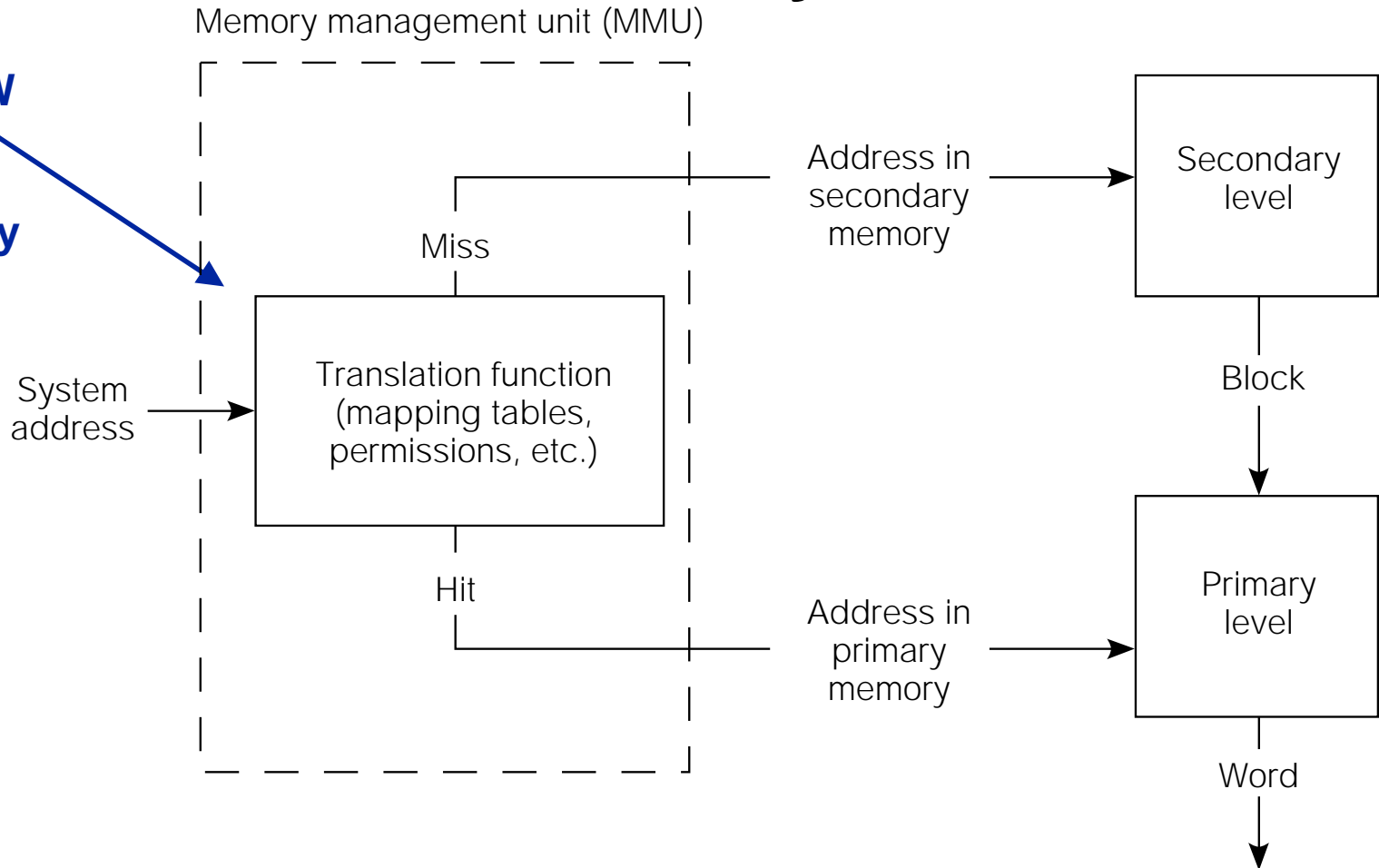
| Primary level | | Secondary level |

**Typical latencies:
cache latency: a few clocks
Disk latency: 100,000 clocks**

- **The item of commerce between any two levels is the <u>block</u>.**

- **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**

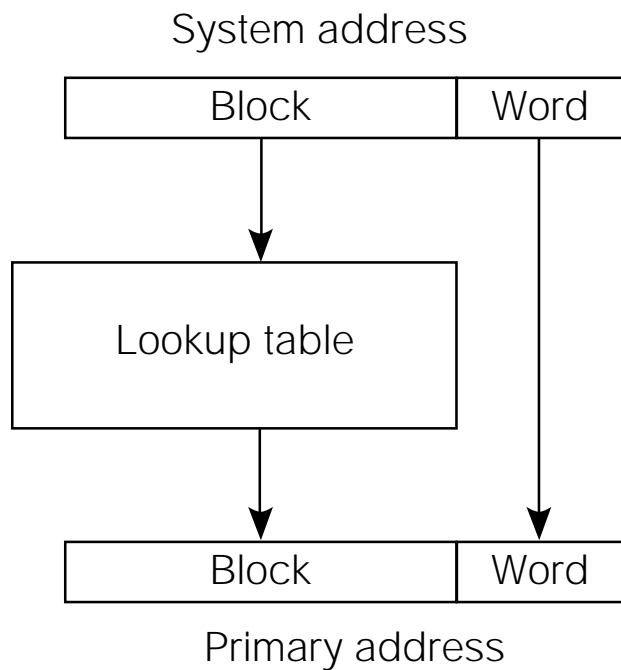- **Disk address: track number, sector number, offset of word in sector.**

Fig 7.28   Addressing and Accessing a 2-Level Hierarchy

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

Memory management unit (MMU)

System address

Miss

Translation function (mapping tables, permissions, etc.)

Hit

Address in secondary memory

Secondary level
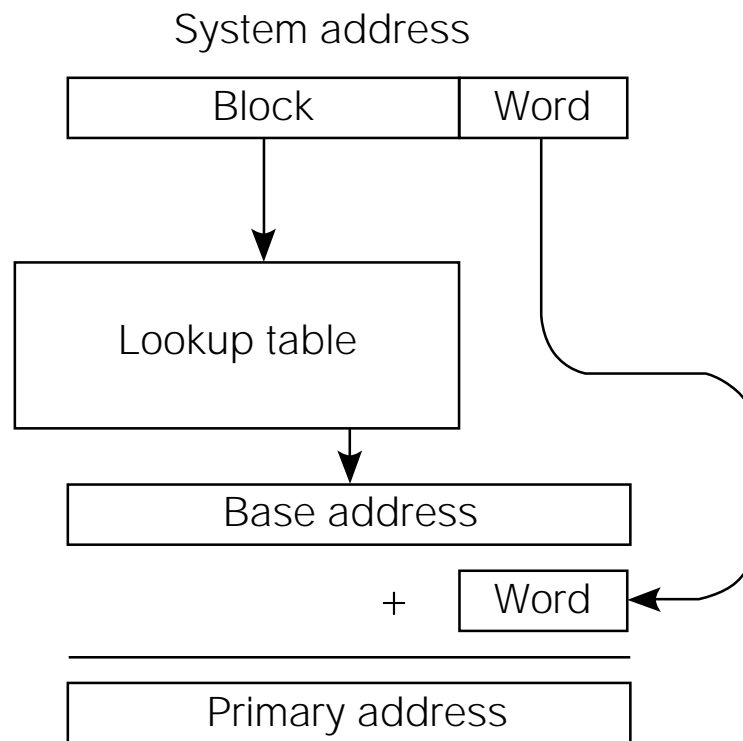
Block

Address in primary memory

Primary level

Word

**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

System address

Block	Word

Lookup table

Block	Word

Primary address

(a) Paging

System address

Block	Word

Lookup table

Base address

$+$ | Word |

Primary address

(b) Segmentation

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 = $\dfrac{\text{number of hits}}{\text{total number of references}}$

Miss ratio: 1 - 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?**

Fig 7.30 The Cache Mapping Function

Example: **256KB**   **16words**   **32MB**
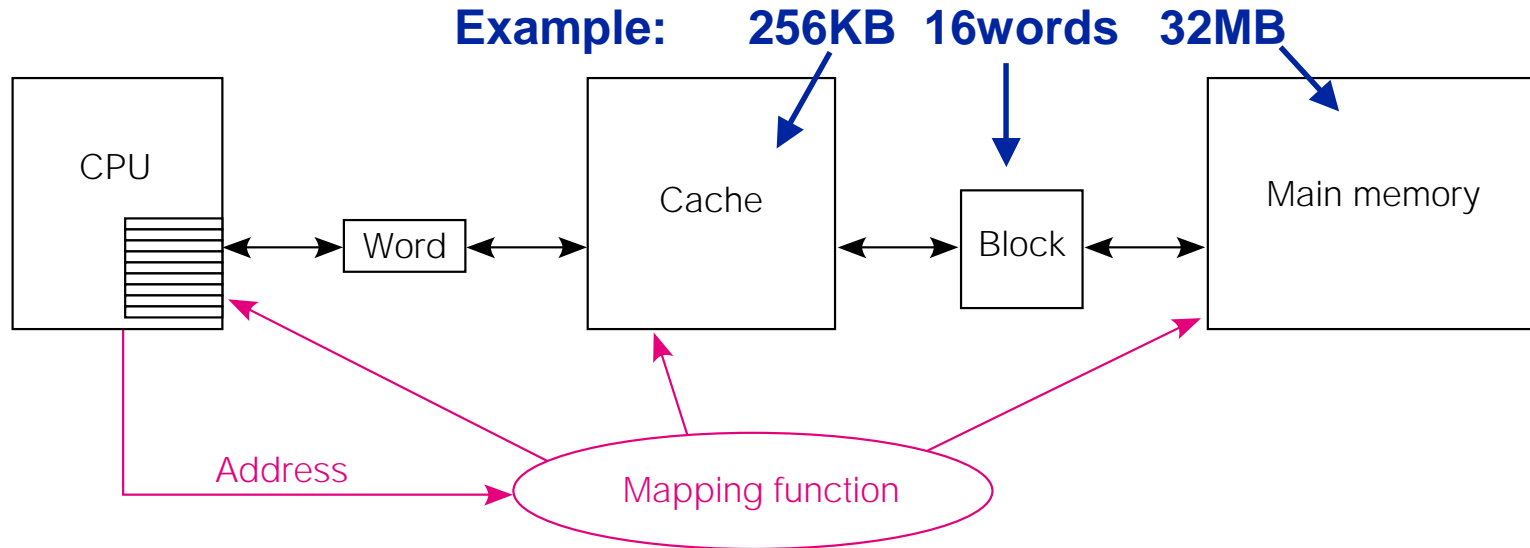
The cache mapping function is responsible for all cache operations:
- **Placement strategy: where to place an incoming block in the cache**
- **Replacement strategy: which block to replace upon a miss**
- **Read and write policy: how to handle reads and writes upon cache misses.**

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

**Three different types of mapping functions:**
- **Associative**
- **Direct mapped**
- **Block-set associative**

Memory fields and address translation

Example of processor-issued 32-bit virtual address:

31   **0**

32 bits

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:

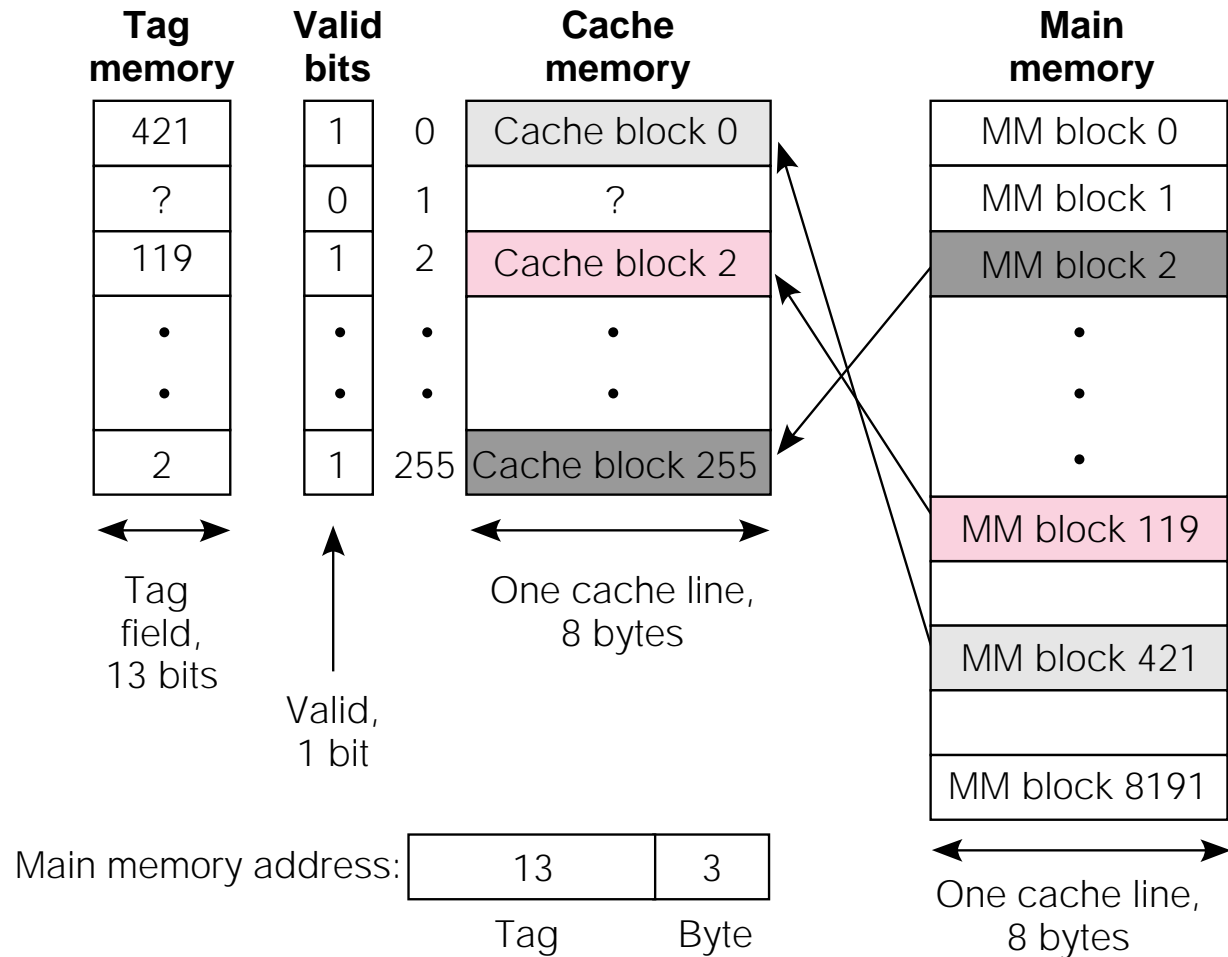| Block Number	Word
26	6

2^{26} **64 word blocks**
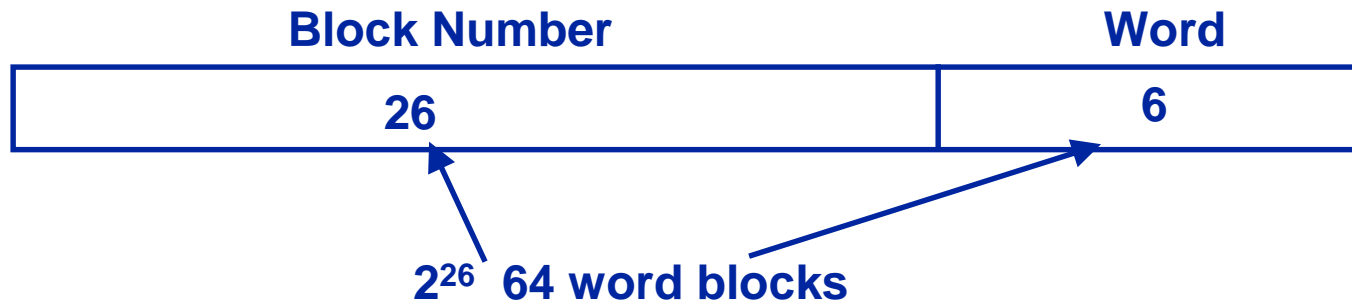
Example of a specific memory reference: Block 9, word 11.

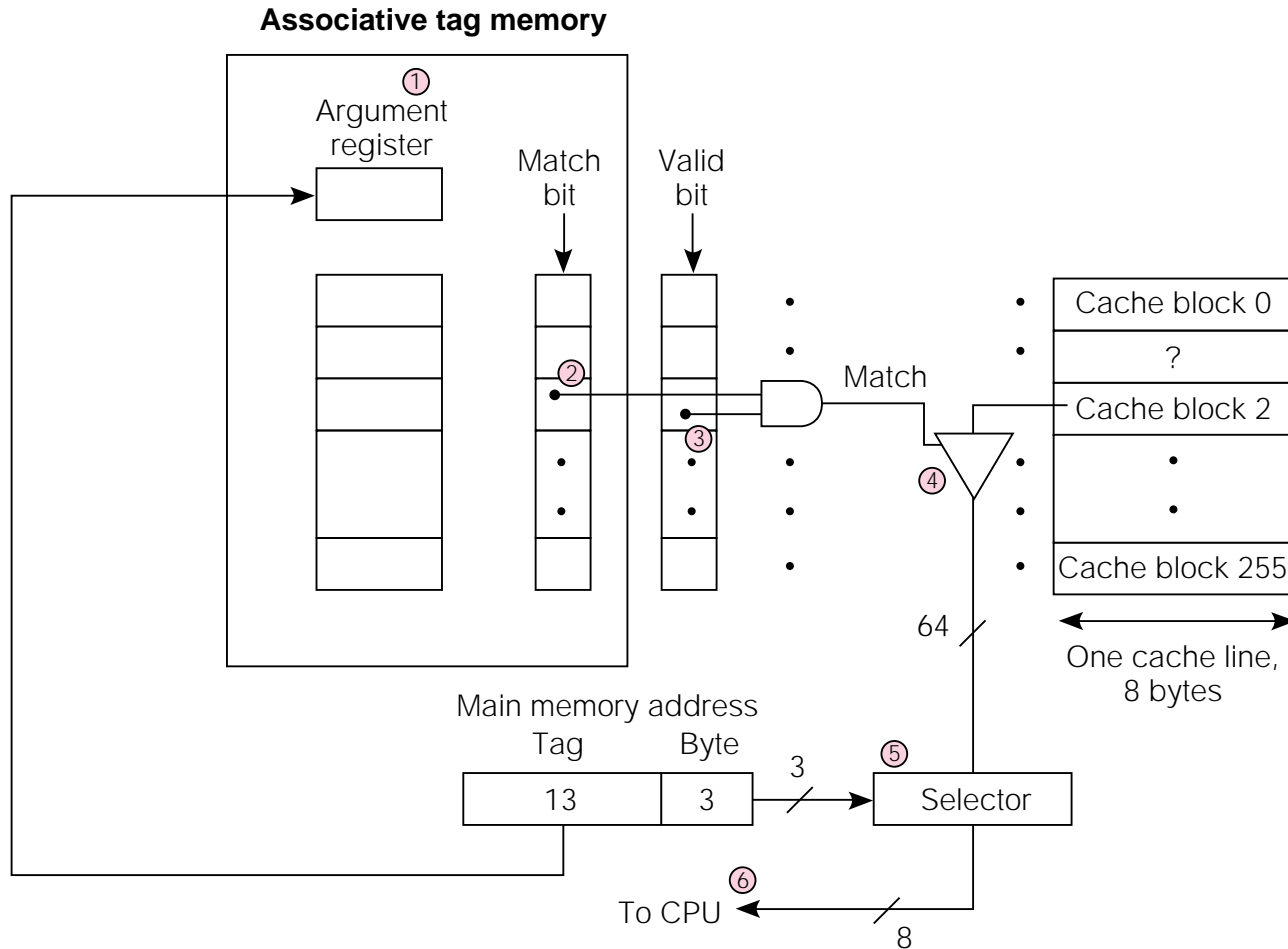| 00	•••	001001	001011

Fig 7.31 Associative mapped caches

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

Tag memory	Valid bits		Cache memory	Main memory
421	1	0	Cache block 0	MM block 0
?	0	1	?	MM block 1
119	1	2	Cache block 2	MM block 2
•	•	•	•	•
•	•	•	•	•
2	1	255	Cache block 255	•
				MM block 119
				MM block 421
				MM block 8191

Tag field, 13 bits

Valid, 1 bit

One cache line, 8 bytes

One cache line, 8 bytes

Main memory address: | 13 | 3 |

Tag   Byte

*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.

Associative tag memory

① Argument register

Match bit

Valid bit

②

③

Match

④

Cache block 0

?

Cache block 2

Cache block 255

64

One cache line, 8 bytes

Main memory address

Tag   Byte

| 13 | 3 |

3

⑤ Selector

⑥ To CPU

8

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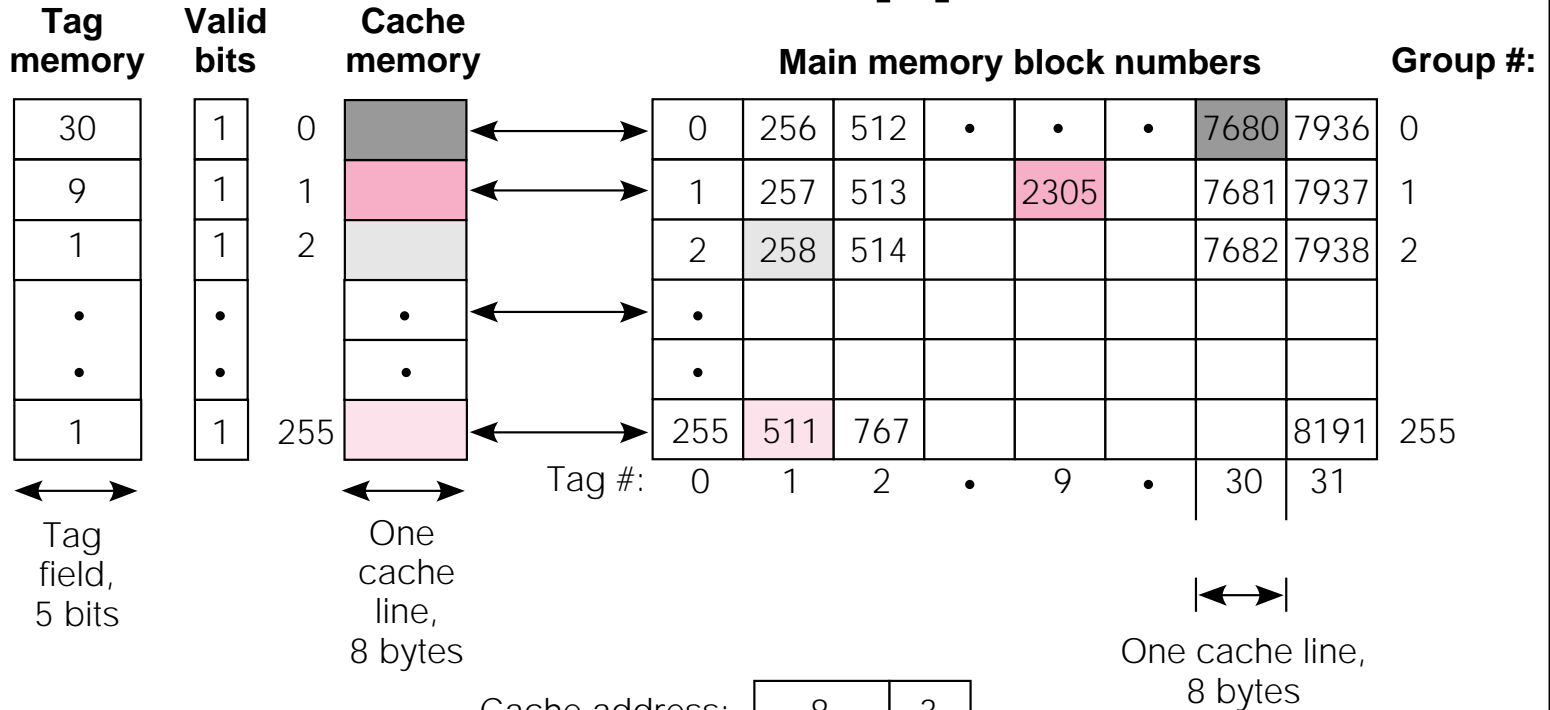- Need to search entire tag memory simultaneously means lots of hardware.

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

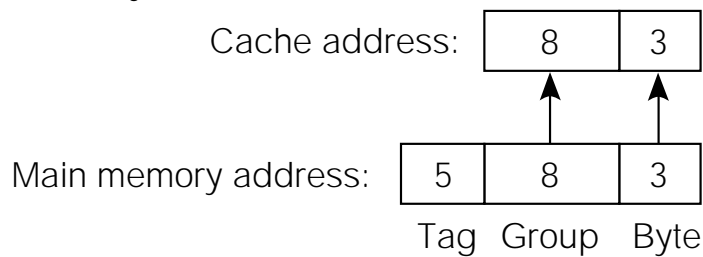**Q.: How is an associative search conducted at the logic gate level?**

–next–
Direct mapped caches simplify the hardware by allowing each MM block to go into only one place in the cache.

Fig 7.33   The 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
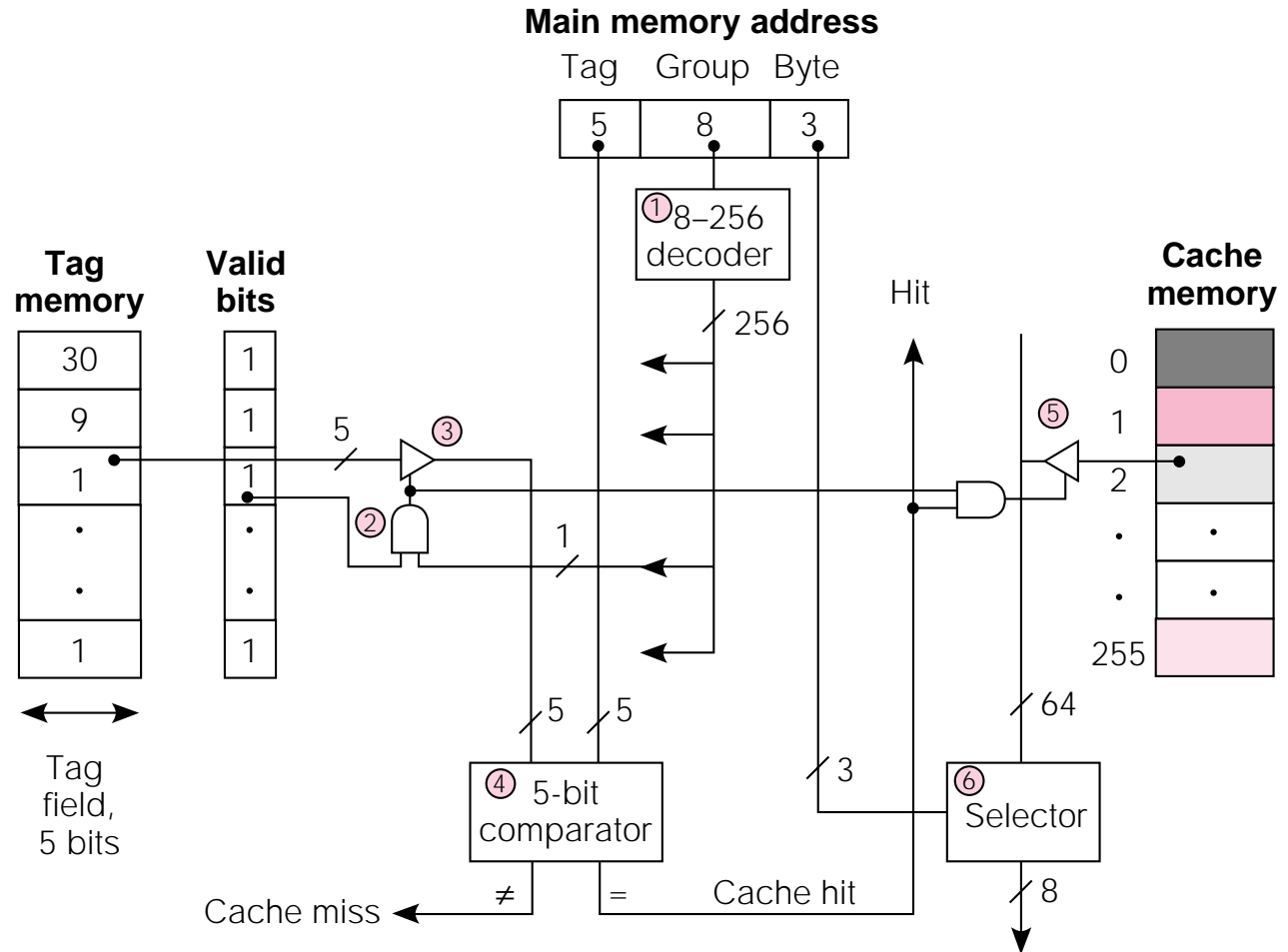
2. If Match AND Valid

3. Then gate out the tag field

4. Compare cache tag with incoming tag

5. If a hit, then gate out the cache line,

6. and use the word field to select the desired word.

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. –next–**

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.

Tag memory		Cache memory			Main memory block numbers								Group #:
2	30	0	512	7680	0	256	512	•	•	•	7680	7936	0
2	9	1	513	2304	1	257	513		2304		7681	7937	1
	1	2		258	2	258	514				7682	7938	2
•	•	•	•	•	•								
•	•	•	•	•	•								
0	1	255	255	511	255	511	767					8191	255

Tag #:   0   1   2   •   9   •   30   31

Tag field, 5 bits

One cache line, 8 bytes

One cache line, 8 bytes

Cache group address:	8	3

| Main memory address:	5	8	3

Tag   Set   Byte
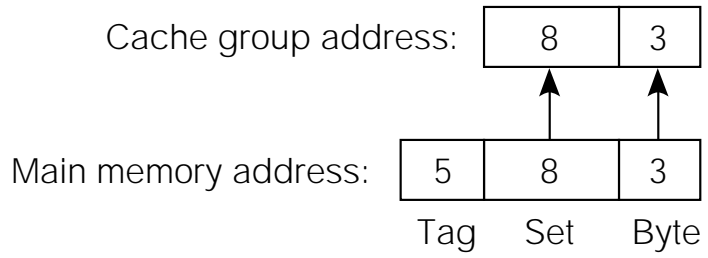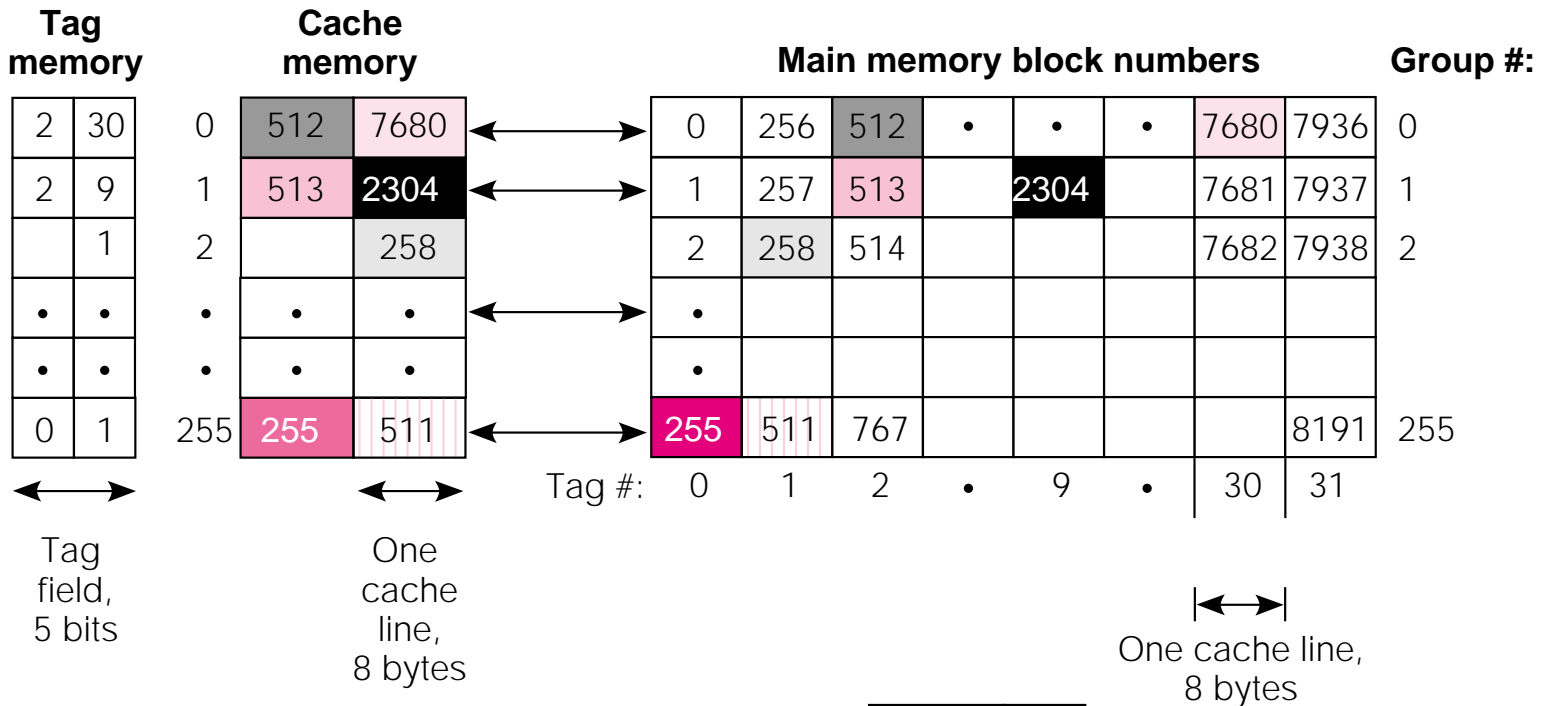
Getting Specific:
The Original Intel Pentium Cache

•**The Pentium actually has two separate caches–one for instructions and one for data. Pentium issues 32-bit MM addresses.**

<div style="float:right;border:1px solid red">

MMX Pentium:
•**16 KB code**
•**16 KB data**
•**4-way SA**

</div>

   •**Each cache is 2-way set associative**
   •**Each cache is 8K=2^{13} bytes in size**
   •**32 = 2^5 bytes per line.**
   •**Thus there are 64 or 2^6 bytes per line, and therefore $2^{13}/2^6$ or 2^7=128 groups**
   •**This leaves 32-5-7 = 20 bits for the tag field:**

Tag	Set (group)	Word
20	7	5

31   **0**
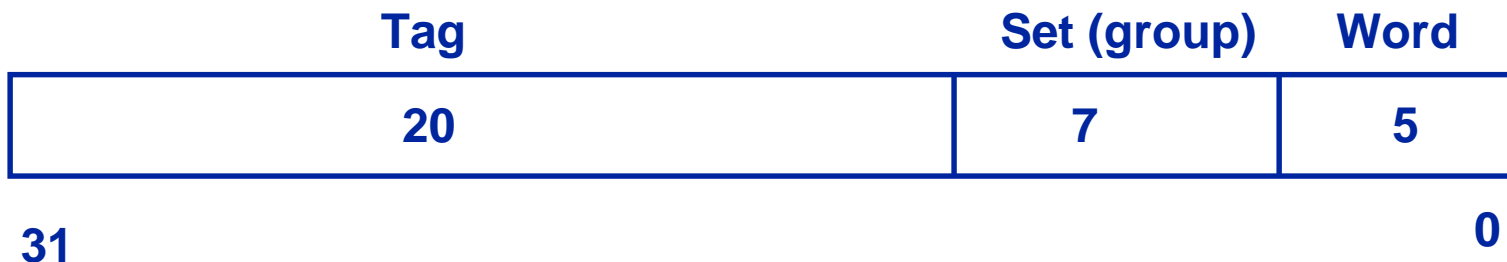
This "cache arithmetic" is important, and deserves your mastery.

Cache Read and Write policies

- **Read and Write cache hit policies**
    - **Write-through–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 $t_p =$ cache and $t_s =$ MM,

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

We define S, the speedup, as $S = T_{without}/T_{with}$ 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 Getting Specific: The PowerPC 601 Cache

Set of 8

Tag memory

Cache memory

Line 0

Sector 0

Sector 1

Address tag

64 sets

• • •

• • •

8 words

8 words

20 bits

64 bytes

Line 63

	Tag	Line (set) #	Word #
Physical address:	20	6	6
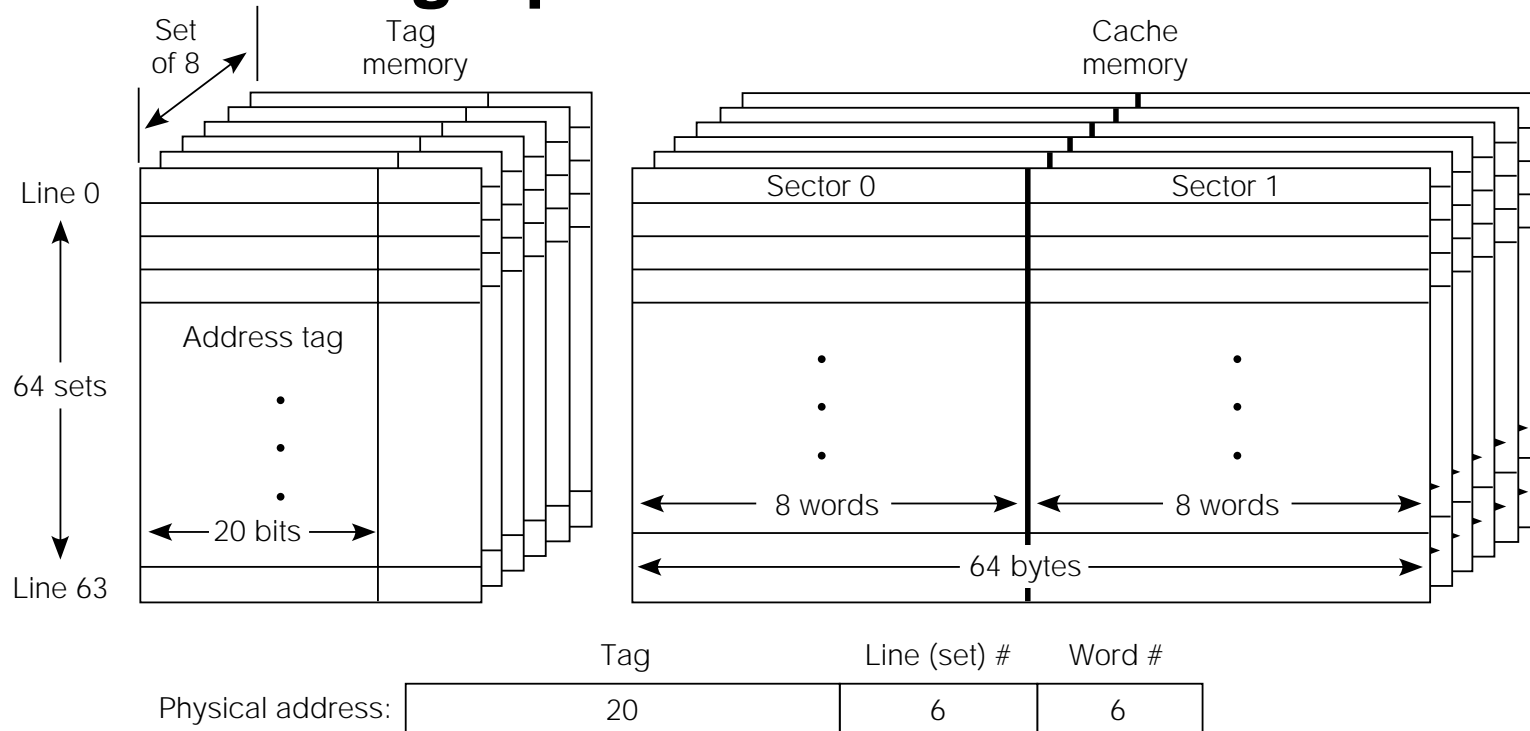
• **The PPC 601 has a *unified* cache - that is, a single cache for both instructions and data.**
• **It is 32KB in size, organized as 64x8block set associative, with blocks being 8 8-byte words organized as 2 independent 4 word sectors for convenience in the updating process**
• **A cache line can be 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.

CPU Chip

```
┌─────────────────────────────────────────────────┐
│  ┌─────────┐          ┌──────────┐    ┌────────┐ │   ┌────────────┐    ┌────────┐
│  │         │ Logical  │   MMU    │ Physical│    │ │   │            │    │        │
│  │  CPU    │ Address  │ ┌──────┐ │ Address │Cache│ │   │Main Memory │    │  Disk  │
│  │         │◄────────►│ │Mapping│◄────────►│    │ │◄─►│            │◄──►│        │
│  │         │          │ │Tables │ │        │    │ │   │            │    │        │
│  └─────────┘          │ └──────┘ │        └────┘ │   └────────────┘    └────────┘
│                       │ Virtual  │                │
│                       │ Address  │                │
│                       └──────────┘                │
└───────────────────────────────────────────────────┘
```
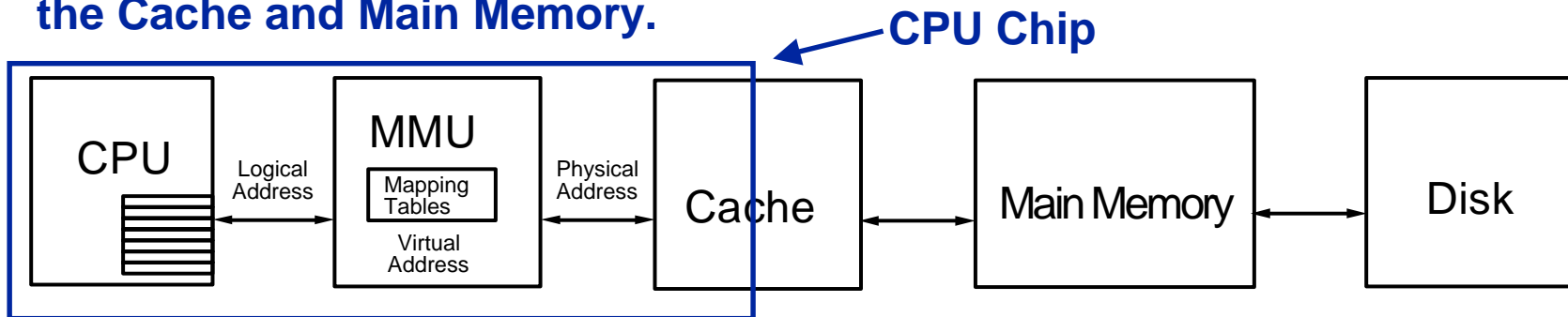
A word about addresses:

- **Effective Address** - an address computed by 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.**
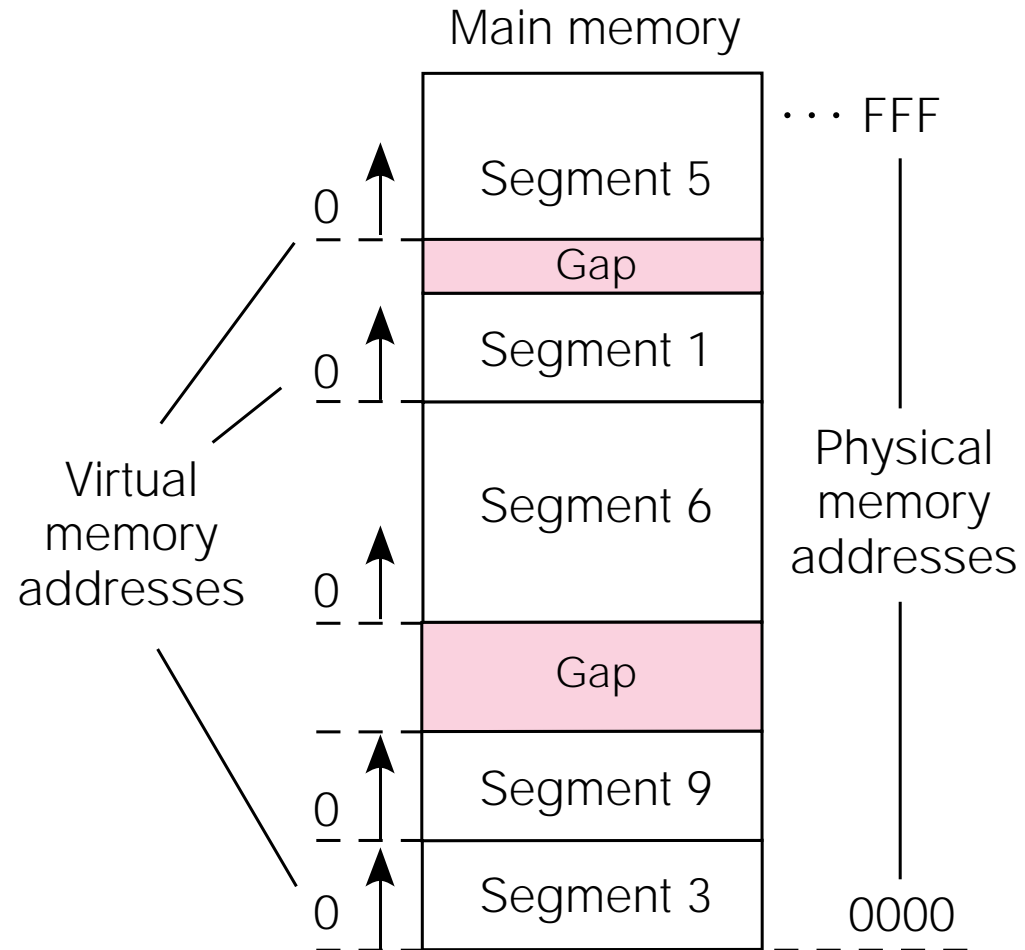
–more–

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 memory limitations.**

•**Cost effective use of physical memory.**
  •**Less expensive secondary (disk) storage can replace primary storage. (The MMU will bring portions of the program into physical memory as required)**

•**Access control. As each memory reference is translated, it can be simultaneously checked for read, write, and execute privileges.**
  •**This allows access/security control at the most fundamental levels.**
  •**Can be used to prevent buggy programs and intruders from causing damage to other users or the system.**
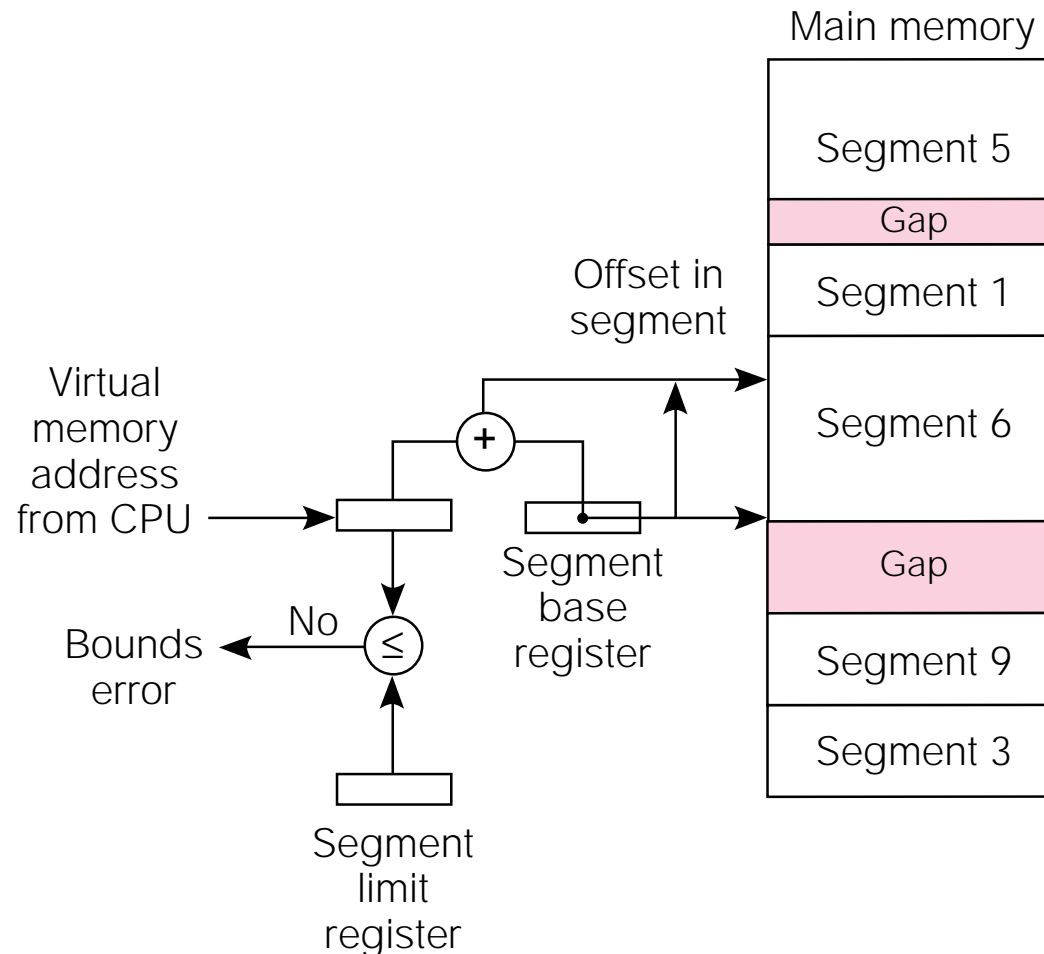
 This is the origin of those "bus error" and "segmentation fault" messages...

Fig 7.38 Memory management by segmentation

Main memory

Segment 5

Gap

Segment 1

Segment 6

Gap

Segment 9

Segment 3

0

0

0

0

0

· · · FFF

Virtual memory addresses

Physical memory addresses

0000

•**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.**

Main memory

Fig 7.39 Segmentation Mechanism

Segment 5

Gap

Offset in segment

Segment 1

Virtual memory address from CPU

$+$

Segment 6

Segment base register

Gap

No

$\leq$

Bounds error

Segment 9

Segment 3

Segment limit register

• **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?**
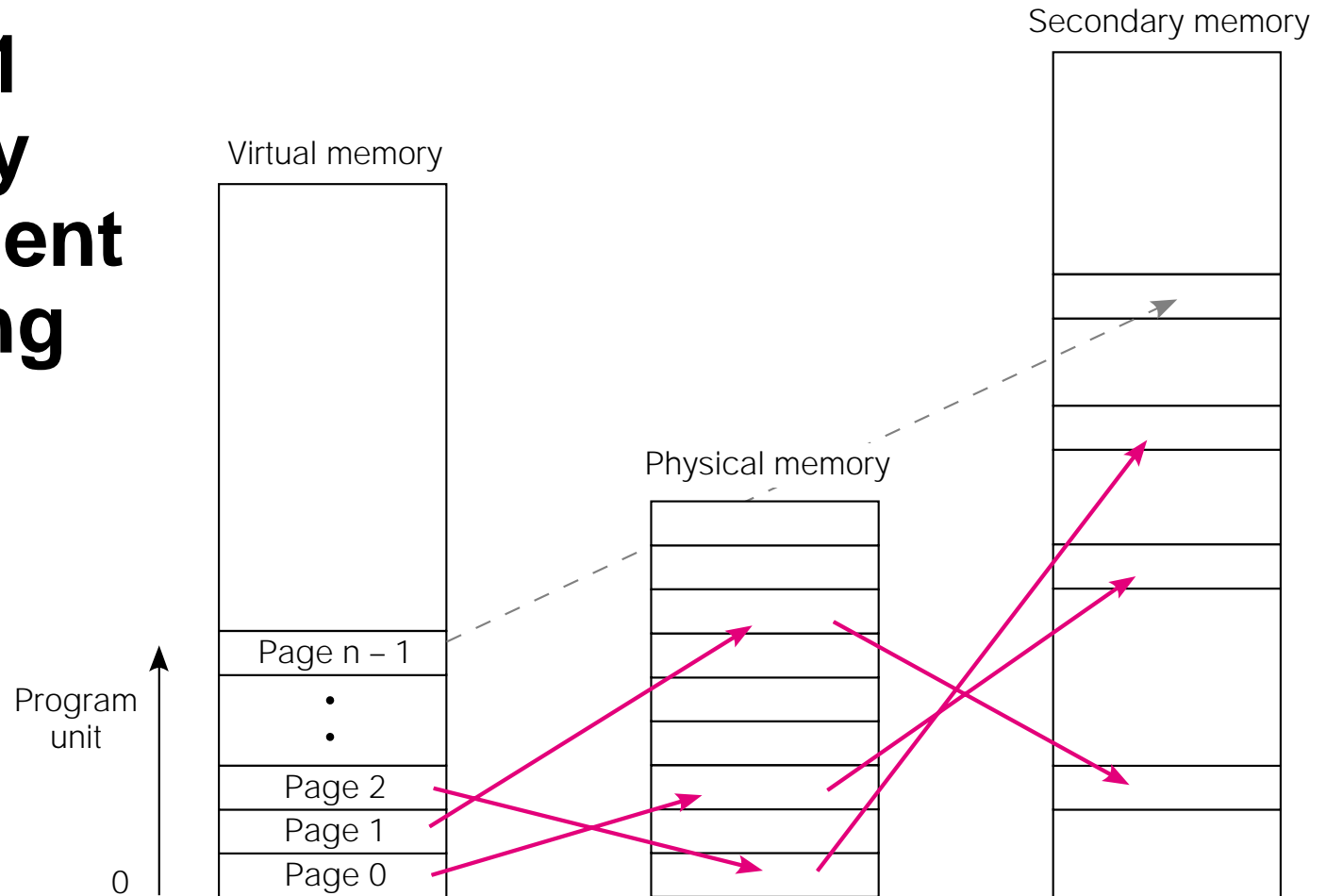
Fig 7.40   The Intel 8086 Segmentation Scheme

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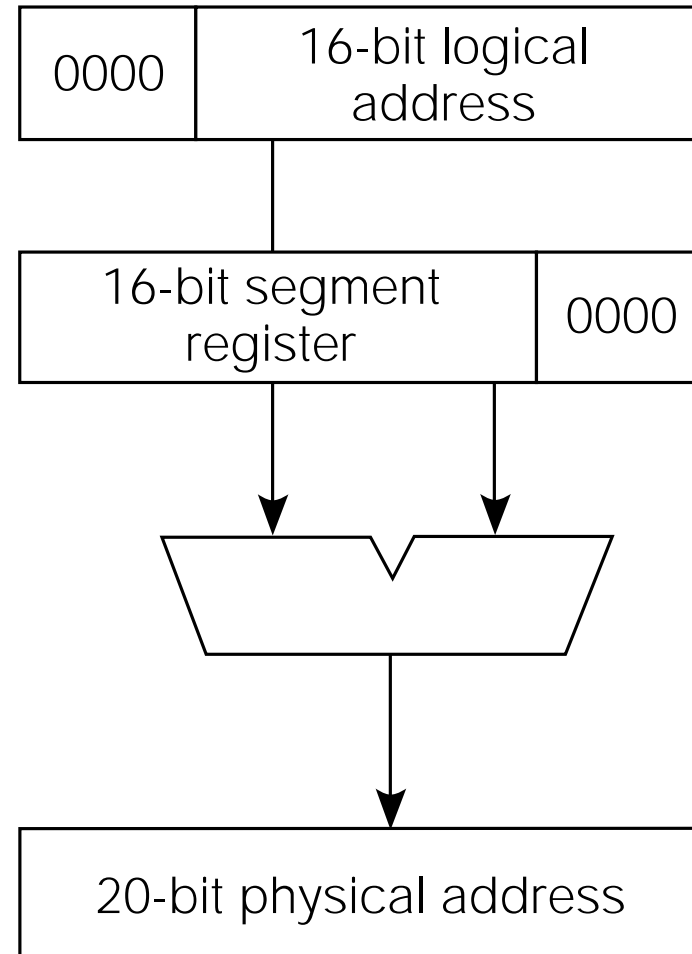
The CPU allows 4 simultaneously active segments, CODE, DATA, STACK, and EXTRA. There are 4 16-bit segment base registers.

Fig 7.41 Memory management by paging

Secondary memory

Virtual memory

Physical memory

Program unit

Page n – 1

Page 2

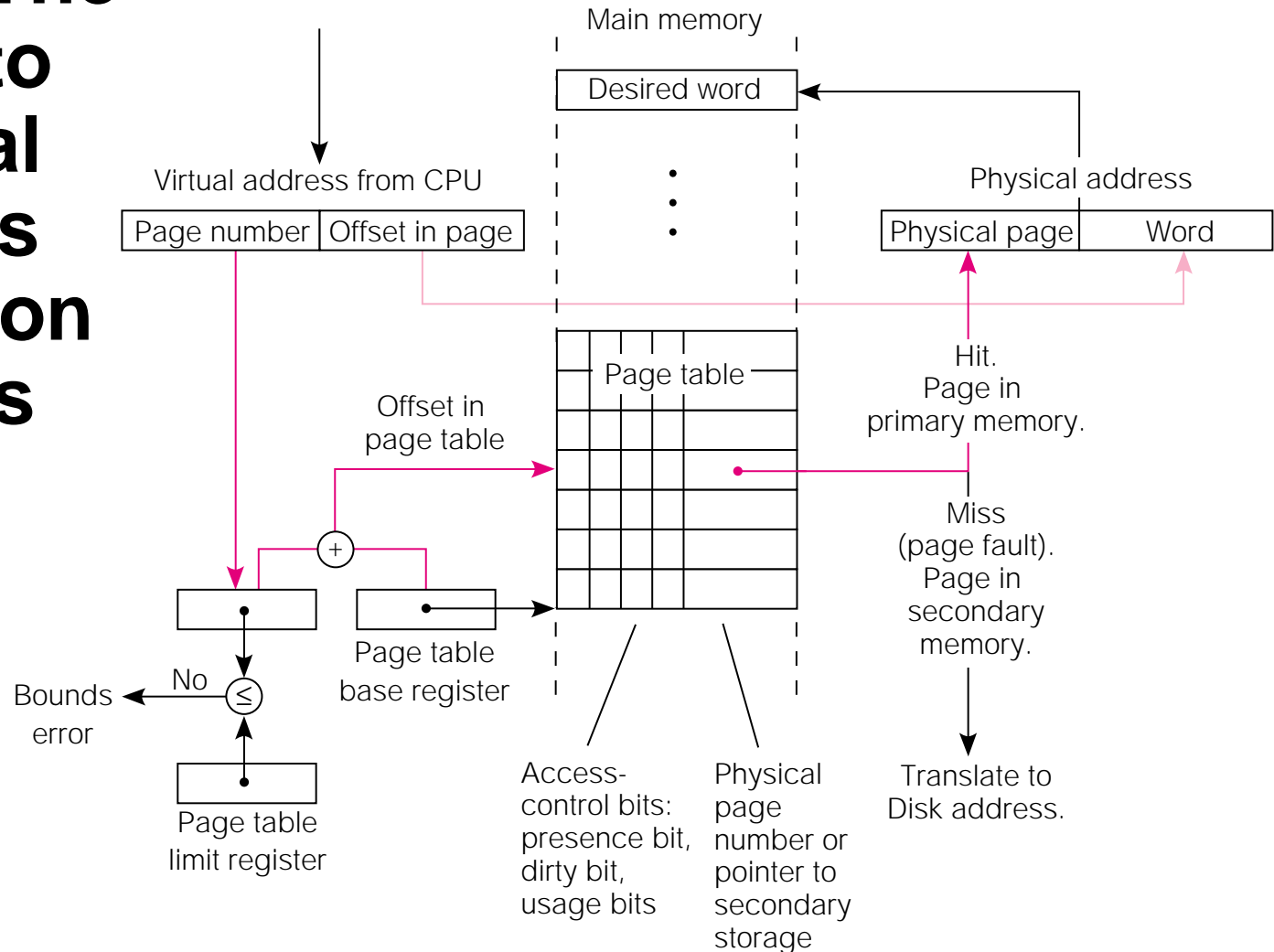Page 1

Page 0

0

•**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 that manages this mapping.**

Fig 7.42 The Virtual to Physical Address Translation Process

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

Main memory

Desired word

Virtual address from CPU

| Page number | Offset in page |

Physical address

| Physical page | Word |

Page table

Offset in page table

Page table base register

Page table limit register

Bounds error ← No ← ≤

Access-control bits: presence bit, dirty bit, usage bits

Physical page number or pointer to secondary storage

Hit. Page in primary memory.

Miss (page fault). Page in secondary memory.

Translate to Disk address.

- **A page fault will result in 100,000 or more cycles passing before the page has been brought from secondary storage to MM.**
- **Page tables are maintained by the OS**

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.

Fig 7.43 TLB Structure and Operation

Main memory or cache

Desired word

Physical address

Virtual address from CPU

| Page number | Word |

| Physical page | Word |

Associative lookup
of virtual page
number in TLB

TLB

Hit   Y

TLB hit.
Page is in
primary memory.

N

TLB miss.
Look for
physical page
in page table.

To page table

Virtual
page
number

Access-
control bits:
presence bit,
dirty bit,
valid bit,
usage bits

Physical
page
number

Fig 7.44 Operation of the Memory Hierarchy

CPU   Cache   Main memory   Secondary memory

Virtual address

Search TLB   Search cache   Search page table

TLB hit Y   Cache hit Y   Page table hit Y

Miss   Miss   Miss

Page fault. Get page from secondary memory

Update MM, cache, and page table

Generate physical address

Update cache from MM

Generate physical address

Return value from cache

Update TLB

Computer Systems Design and Architecture by V. Heuring and H. Jordan   © 1997 V. Heuring and H. Jordan: Updated David M. Zar, February, 2001

Fig 7.45 The PowerPC 601 MMU Operation

"Segments" are actually more akin to large (256 MB) blocks.

32-bit logical address from CPU

Seg #	Virtual pg #		Word
4	9	7	12

16

4

0

Access control and misc.

24-bit virtual segment ID (VSID)

15

16

24

16

12

Hit—to CPU

d_0–d_{31}

Cache

32

Miss—cache load

7

UTLB

0

0

Set 1

Set 0

20-bit physical page

40-bit virtual page

:

:

40

20

Compare

Compare

40

Miss—to page table search
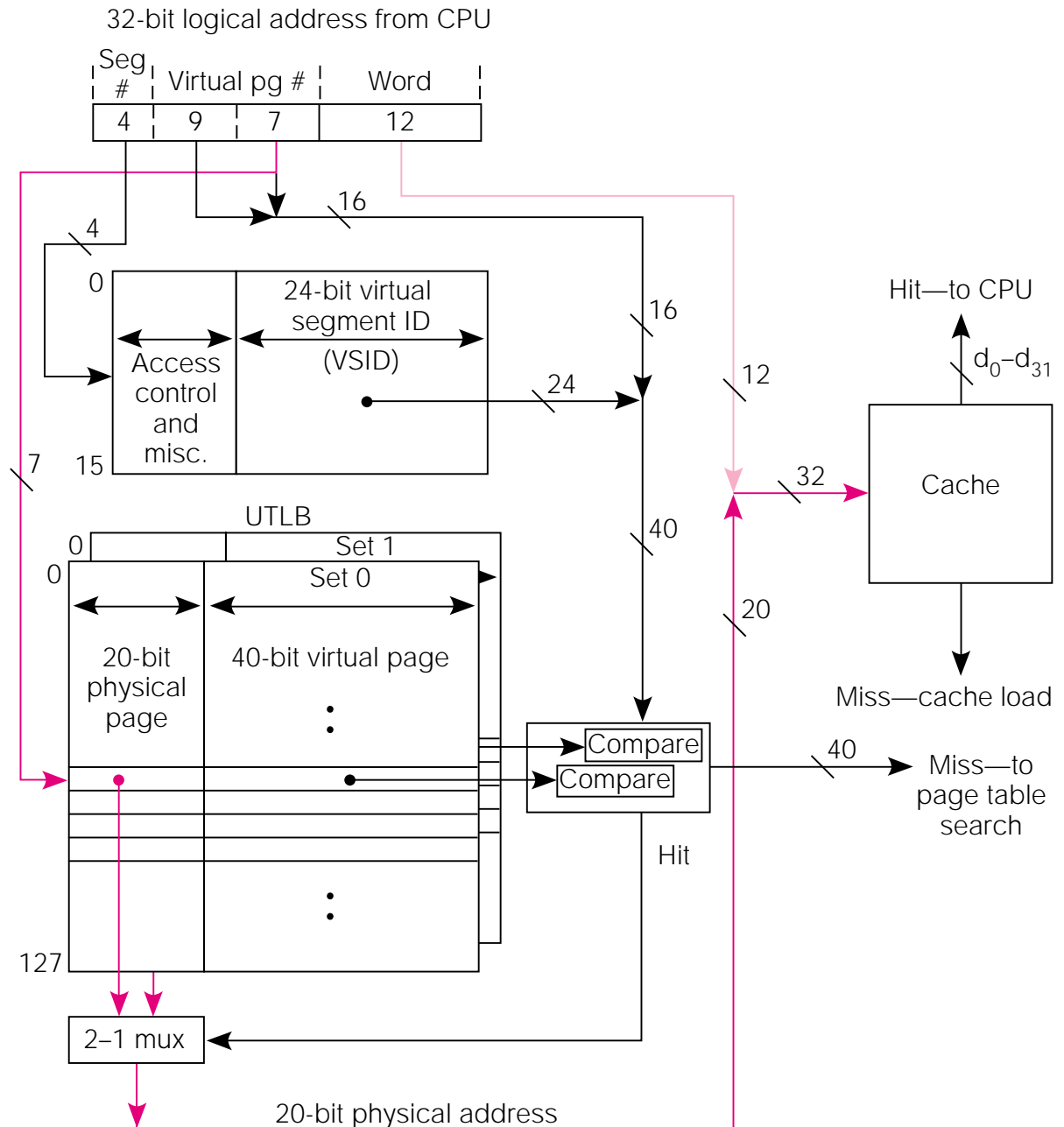
Hit

127

2–1 mux

20-bit physical address

Fig 7.46   I/O Connection to a Memory with a Cache

•**The memory system is quite complex, and affords many possible tradeoffs.**
   •**The only realistic way to chose among these alternatives is to study a typical workload, using either simulations or prototype systems.**
   •**Instruction and data accesses usually have different patterns.**
•**It is possible to employ a cache at the disk level, using the disk hardware.**
•**Traffic between MM and disk is I/O, and Direct Memory Access, DMA can be used to speed the transfers:**