- Problem #1: we have latch (or flip-flop) based registers, but they aren't **addressable**.
  - An **address** (or **index**) allows dynamic rather than static reference to some stored datum.
  - By rough analogy to a C program, we have the left-hand side

Listing (C)

```c
int A0, A1, A2, A3;

A0 = 0;
A1 = 0;
A2 = 0;
A3 = 0;
```

Listing (C)

```c
int A[ 4 ];

A[ 0 ] = 0;
A[ 1 ] = 0;
A[ 2 ] = 0;
A[ 3 ] = 0;
```

but want the right-hand side.

- Problem #2: latches and flip-flops need a (relatively) large number of transistors per-bit; to support large capacities, it can make sense to use different components.

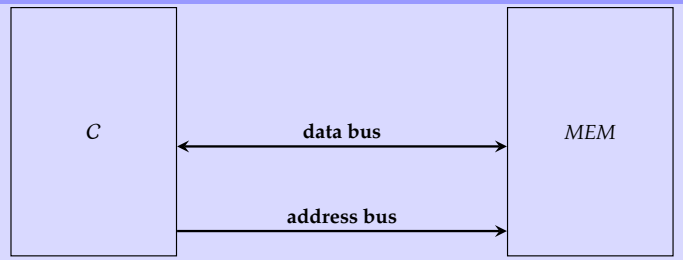
Notes:

- There are various ways to classify a given memory component, e.g.,
  1. **Volatility**
     - **volatile**, meaning the content is lost when the component is powered-off, or
     - **non-volatile**, meaning the content is retained even after the component is powered-off.
  2. **Access type**
     - random versus constrained (e.g., sequential access to content),
     - **Random Access Memory (RAM)** which we can read from *and* write to, and
     - **Read Only Memory (ROM)** which, as suggested by the name, supports reads only.
  3. **Interface type**
     - **synchronous**, where a clock or pre-determined timing information synchronises steps, or
     - **asynchronous**, where a protocol synchronises steps.

Notes:

- We're (mainly) interested in a how a (volatile, synchronous) RAM component works:

Example



- Goal: a concrete implementation of *MEM* with capacity for $n = 2^{n'}$ addressable words each of w bits (where $n \gg w$).

Notes:

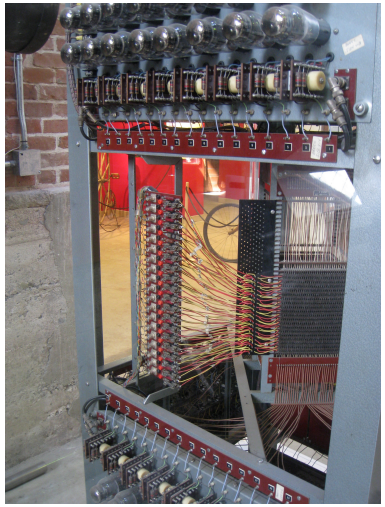
An Aside: History

- The EDSAC used **delay line** memory, where the rough idea is:
  - Each "line" is a tube of mercury (or something else in which sound waves propagate fairly slowly).
  - Put a speaker at one end to store sound waves into the line, and a microphone at the other to read them out.
  - Values are stored in the sense the corresponding waves take time to propagate; when they get to one end they are either replaced or fed back into the other.
- This is **sequential access** (cf. **random access**): you need to *wait* for the data you want to appear!

http://www.cl.cam.ac.uk/Relics/jpegs/delay_lines.jpg

Notes:

An Aside: History



- The Whirlwind used **magnetic-core** memory, where the rough idea is:
  - The memory is a matrix of small magnetic rings, or "cores", which can be magnetically polarised to store values.
  - Wires are threaded through the cores to control them, i.e., to store or read values.
  - The magnetic polarisation is retained, so core memory is non-volatile!
- You might still hear main memory termed **core memory** (cf. **core dump**) which is a throw-back to this technology.

http://en.wikipedia.org/wiki/File:Project_Whirlwind_-_core_memory,_circa_1951_-_detail_1.JPG

Notes:

Low-level Implementation (1) – SRAM and DRAM Cells

Definition (SRAM)

Static RAM (SRAM) is

- manufacturable in lower densities (i.e., smaller capacity),
- more expensive to manufacture,
- fast(er) access time (resp. lower access latency),
- easy(er) to interface with,
- ideal for use in **register files** and **cache memory**.

Definition (DRAM)

Dynamic RAM (DRAM) is

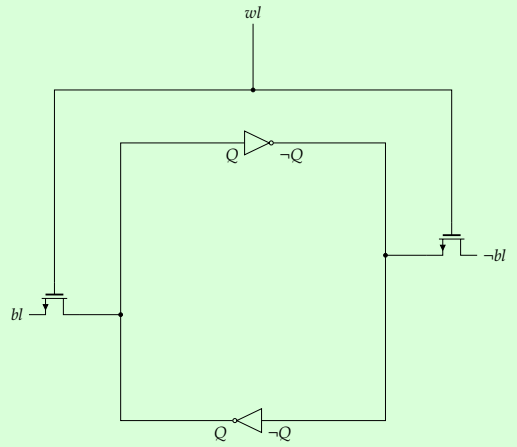
- manufacturable in higher densities (i.e., larger capacity),
- less expensive to manufacture,
- slow(er) access time (resp. higher access latency),
- hard(er) to interface with,
- ideal for use as **main memory**.

Notes:

Low-level Implementation (2) – SRAM and DRAM Cells

- Abstractly, an **SRAM cell** resembles two NOT gates ...

Circuit (SRAM cell)



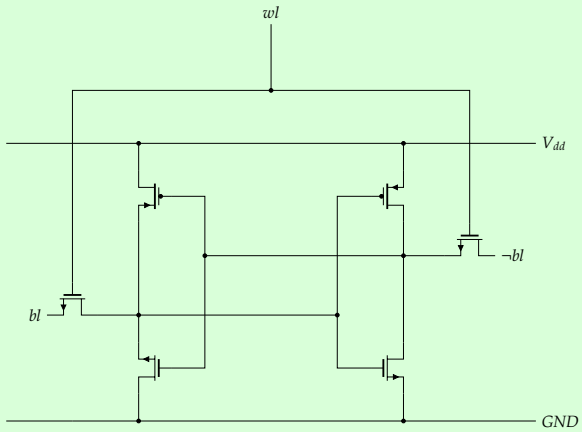
- ... concretely, we can fill in the NOT gates with the transistor-level equivalents; each "6T SRAM cell" requires 6 transistors (cf. ~ 30 or 40 for a flip-flop).

Notes:

Low-level Implementation (2) – SRAM and DRAM Cells

- Abstractly, an **SRAM cell** resembles two NOT gates ...

Circuit (SRAM cell)



- ... concretely, we can fill in the NOT gates with the transistor-level equivalents; each "6T SRAM cell" requires 6 transistors (cf. ~ 30 or 40 for a flip-flop).

Notes:

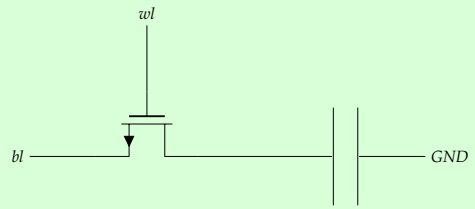
- Note that:
  - The initial NOT-based circuit might look odd, but clearly has two stable states: either $Q = 1$ and $\neg Q = 0$, or $Q = 0$ and $\neg Q = 1$.
  - The transistors re-enforce each other; the state is maintained as long as the cell is powered-on.
  - bl and $\neg bl$ are termed the **bit lines**, wl is the **word line** which controls access to the state.
  - The pre-charging steps are managed by extra **bit line conditioning** logic which we gloss over from here on.
- So
  - to read the cell we pre-charge $bl = 1$ and $\neg bl = 1$ then set $wl = 1$, after which $\neg bl$ (resp. bl) is discharged if state is 1 (resp. 0), or
  - to write x into the cell we pre-charge $bl = x$ and $\neg bl = \neg x$ then set $wl = 1$, after which the state matches x.

Notes:

Low-level Implementation (4) – SRAM and DRAM Cells

- A **DRAM cell** is constructed using 1 transistor and a capacitor:

Circuit (DRAM cell)



Notes:

- Note that:
  - The state *decays* when the cell is read, *and* also over time (even if it's not read) due to leakage; this implies a need to **refresh** the cell periodically.
  - The capacitor holds a tiny charge: this must be amplified to use as a driver in whatever circuit uses the cell.
  - The speed at which the capacitor charges and discharges is (relatively) slow.
- So
  - to read the cell we set $wl = 1$, after which current flows (resp. does not flow) on bl if the capacitor is charged (resp. not charged) meaning the state is 1 (resp. 0), or
  - to write x into the cell we set $bl = x$ then $wl = 1$, after which the capacitor charges (resp. discharges) and the state matches x.

Notes:

High(er)-level Implementation (1) – Cells ⇝ Device

- A **memory device** is constructed from (roughly) three components
  1. a **memory array** (or matrix) of replicated cells with
     - r rows, and
     - c columns

     meaning a $(r \cdot c)$-cell capacity.
  2. a **row decoder** which given an address (de)activates associated cells in that row, and
  3. a **column decoder** which given an address (de)selects associated cells in that column

  plus additional logic to allow use (depending on cell type), e.g.,
  1. **bit line conditioning** to ensure the bit lines are strong enough to be effective,
  2. **sense amplifiers** to ensure output from the array is usable.

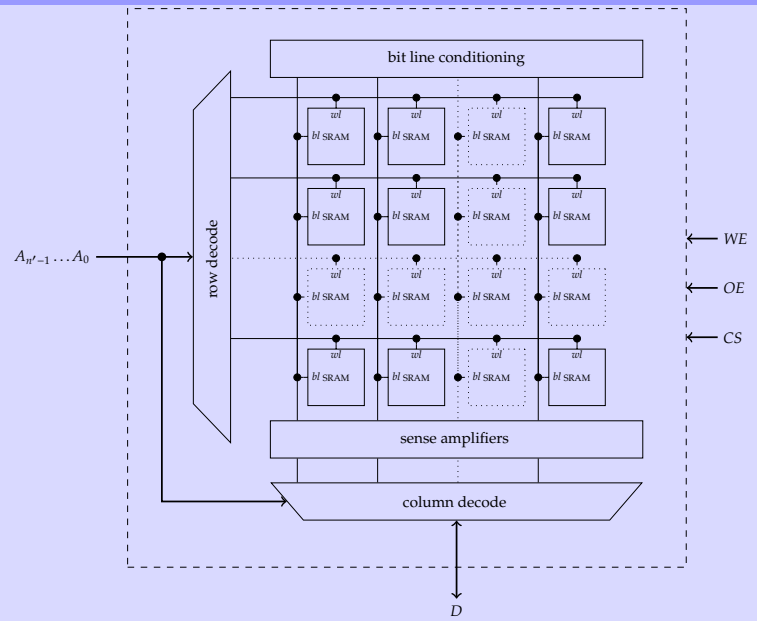
Notes:

- A 1-bit **SRAM device** with $n = 2^{n'}$ cells has a (somewhat) standard physical interface:
  1. auxiliary pin(s) for power and so on,
  2. D, a single 1-bit **data pin** (sometimes split into two separate D_{in} and D_{out} pins),
  3. A, a collection of n' **address pins** where A_i is the i-th such pin,
  4. a **Chip Select (CS)** pin, which enables the device,
  5. a **Output Enable (OE)** pin, which signals the device is being read from, and
  6. a **Write Enable (WE)** pin, which signals the device is being written to.

Notes:

High(er)-level Implementation (3) – Cells ⇝ Device, SRAM

Example (an n-cell SRAM memory device)



Notes:

Algorithm (SRAM-Read)

Having performed the following steps

1. drive the address onto A,
2. set WE = **false**, OE = **true** and CS = **true**

1-bit of data is read and made available on D, then we set CS = **false**.

Algorithm (SRAM-Write)

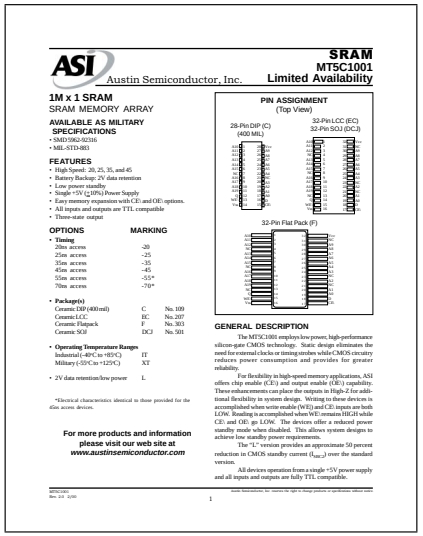
Having performed the following steps

1. drive the address onto A,
2. drive the data onto D,
3. Set WE = **true**, OE = **false** and CS = **true**

1-bit of data is written, then we set CS = **false**.

Notes:

High(er)-level Implementation (5) – Cells ⇝ Device, SRAM



ASI Austin Semiconductor, Inc.

SRAM
MT5C1001
Limited Availability

1M x 1 SRAM
SRAM MEMORY ARRAY

AVAILABLE AS MILITARY SPECIFICATIONS
- SMD 5962-92316
- MIL-STD-883

FEATURES
- High Speed: 20, 25, 35, and 45
- Battery Backup: 2V data retention
- Low power standby
- Single +5V (±10%) Power Supply
- Easy memory expansion with CE1 and OE options.
- All inputs and outputs are TTL compatible
- Three-state output

OPTIONS	MARKING
• **Timing**	
20ns access	-20
25ns access	-25
35ns access	-35
45ns access	-45
55ns access	-55*
70ns access	-70*
• **Package(s)**	
Ceramic DIP (400 mil)	C No. 109
Ceramic LCC	EC No. 207
Ceramic Flatpack	F No. 303
Ceramic SOJ	DCJ No. 501
• **Operating Temperature Ranges**	
Industrial (-40°C to +85°C)	IT
Military (-55°C to +125°C)	XT
• 2V data retention/low power	L

*Electrical characteristics identical to those provided for the 45ns access devices.

For more products and information please visit our web site at *www.austinsemiconductor.com*

PIN ASSIGNMENT
(Top View)

28-Pin DIP (C) (400 MIL)
32-Pin LCC (EC)
32-Pin SOJ (DCJ)
32-Pin Flat Pack (F)

GENERAL DESCRIPTION

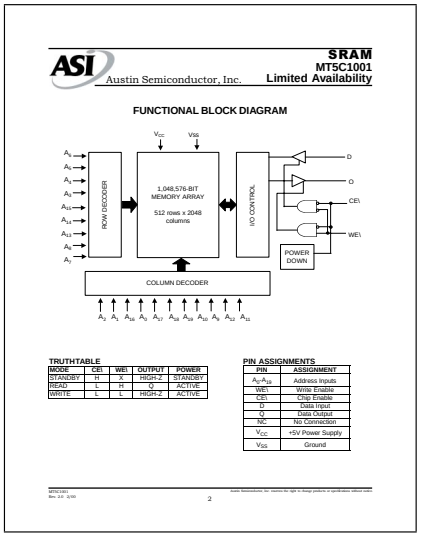
The MT5C1001 employs low power, high-performance silicon-gate CMOS technology. Static design eliminates the need for external clocks or timing strobes while CMOS circuitry reduces power consumption and provides for greater reliability.

For flexibility in high-speed memory applications, ASI offers chip enable (CE1) and output enable (OE1) capability. These enhancements can place the outputs in High-Z for additional flexibility in system design. Writing to these devices is accomplished when write enable (WE1) and CE1 inputs are both LOW. Reading is accomplished when WE1 remains HIGH while CE1 and OE1 go LOW. The devices offer a reduced power standby mode when disabled. This allows system designs to achieve low standby power requirements.

The "L" version provides an approximate 50 percent reduction in CMOS standby current (I_{SBC2}) over the standard version.

All devices operation from a single +5V power supply and all inputs and outputs are fully TTL compatible.

MT5C1001 Rev. 2.0 2/00 — Austin Semiconductor, Inc. reserves the right to change products or specifications without notice. — 1



ASI Austin Semiconductor, Inc.

SRAM
MT5C1001
Limited Availability

FUNCTIONAL BLOCK DIAGRAM

TRUTHTABLE

MODE	CE1	WE1	OUTPUT	POWER
STANDBY	H	X	HIGH-Z	STANDBY
READ	L	H	Q	ACTIVE
WRITE	L	L	HIGH-Z	ACTIVE

PIN ASSIGNMENTS

PIN	ASSIGNMENT
A_0–A_{19}	Address inputs
WE1	Write Enable
CE1	Chip Enable
D	Data Input
Q	Data Output
NC	No Connection
V_{CC}	+5V Power Supply
V_{SS}	Ground

MT5C1001 Rev. 2.0 2/00 — Austin Semiconductor, Inc. reserves the right to change products or specifications without notice. — 2

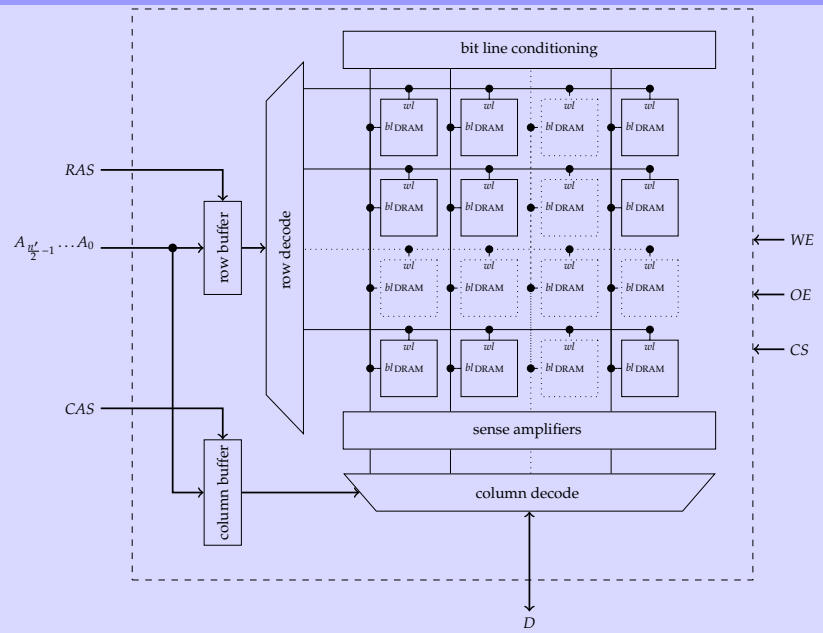
Notes:

- A 1-bit **DRAM device** with $n = 2^{n'}$ cells has a (somewhat) standard physical interface:
  1. auxiliary pin(s) for power and so on,
  2. D, a single 1-bit **data pin** (sometimes split into two separate D_{in} and D_{out} pins),
  3. A, a collection of $\frac{n'}{2}$ **address pins** where A_i is the i-th such pin,
  4. a **Chip Select (CS)** pin, which enables the device,
  5. a **Output Enable (OE)** pin, which signals the device is being read from,
  6. a **Write Enable (WE)** pin, which signals the device is being written to,
  7. a **Row Address Strobe (RAS)**, which controls the row buffer, and
  8. a **Column Address Strobe (CAS)**, which controls the column buffer.
- Note there are typically *half* the number of address pins versus the same sized SRAM device:
  - the pins are multiplexed to form a full address,
  - since DRAM is more dense, this acts to manage the number of pins required.

Notes:

High(er)-level Implementation (7) – Cells ⇝ Device, DRAM

Example (a n-cell DRAM memory device)



Notes:

Algorithm (DRAM-Read)

Having performed the following steps

1. Drive the row address onto *addr*.
2. Set *RAS* = **true**, which latches row address.
3. Drive the column address onto *addr*.
4. Set *CAS* = **true**, which latches column address.
5. Set *WE* = **false**, *OE* = **true** and *CS* = **true**.

1-bit of data is read and made available on *D*, and we set *CSRAS* = *CAS* = **false**.

Algorithm (DRAM-Write)

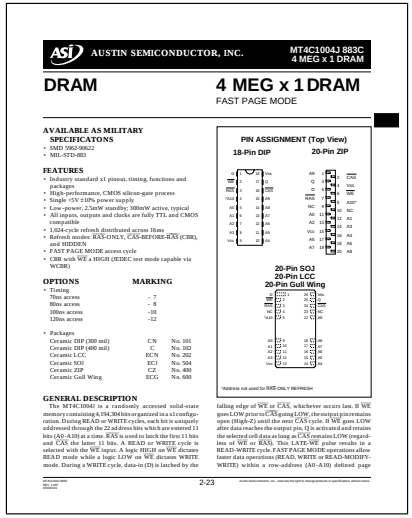
Having performed the following steps

1. Drive the row address onto *addr*.
2. Set *RAS* = **true**, which latches row address.
3. Drive the column address onto *addr*.
4. Set *CAS* = **true**, which latches column address.
5. Drive the data onto *data*.
6. Set *WE* = **false**, *OE* = **true** and *CS* = **true**.

1-bit of data is written, and we set *CSRAS* = *CAS* = **false**.

- Plus we need logic to implement DRAM refresh:
  - To cope with decay of cell content, we periodically read *all* the cells.
  - This amounts to activating each row in turn; the latency of refresh can therefore be optimised by selecting smaller *r* (resp. larger *c*) for the array.

Notes:



AUSTIN SEMICONDUCTOR, INC. MT4C1004J 883C 4 MEG x 1 DRAM

DRAM

4 MEG x 1 DRAM
FAST PAGE MODE

AVAILABLE AS MILITARY SPECIFICATIONS
- SMD 5962-90622
- MIL-STD-883

FEATURES
- Industry standard x1 pinout, timing, functions and packages
- High-performance, CMOS silicon-gate process
- Single +5V ±10% power supply
- Low-power, 2.5mW standby; 300mW active, typical
- All inputs, outputs and clocks are fully TTL and CMOS compatible
- 1,024-cycle refresh distributed across 16ms
- Refresh modes: RAS-ONLY, CAS-BEFORE-RAS (CBR), and HIDDEN
- FAST PAGE MODE access cycle
- CBR with WE a HIGH (JEDEC test mode capable via WCBR)

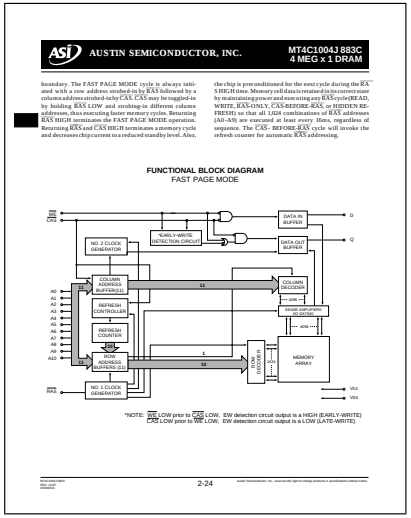
OPTIONS	MARKING
• Timing	
70ns access	- 7
80ns access	- 8
100ns access	-10
120ns access	-12
• Packages	
Ceramic DIP (300 mil)	C/N No. 101
Ceramic DIP (400 mil)	C No. 102
Ceramic LCC	ECN No. 202
Ceramic SOJ	ECJ No. 504
Ceramic ZIP	CZ No. 400
Ceramic Gull Wing	ECG No. 600

PIN ASSIGNMENT (Top View)
18-Pin DIP
20-Pin ZIP
20-Pin SOJ
20-Pin LCC
20-Pin Gull Wing

*Address not used for RAS-ONLY REFRESH

GENERAL DESCRIPTION
The MT4C1004J is a randomly accessed solid-state memory containing 4,194,304 bits organized in a x1 configuration. During READ or WRITE cycles, each bit is uniquely addressed through the 22 address bits which are entered 11 bits (A0–A10) at a time. RAS is used to latch the first 11 bits and CAS the latter 11 bits. A READ or WRITE cycle is selected with the WE input. A logic HIGH on WE dictates READ mode while a logic LOW on WE dictates WRITE mode. During a WRITE cycle, data-in (D) is latched by the falling edge of WE or CAS, whichever occurs last. If WE goes LOW prior to CAS going LOW, the output pin remains open (High-Z) until the next CAS cycle. If WE goes LOW after data reaches the output pin, Q is activated and retains the selected cell data as long as CAS remains LOW (regardless of WE or RAS). This LATE WE pulse results in a READ-WRITE cycle. FAST PAGE MODE operations allow faster data operations (READ, WRITE or READ-MODIFY-WRITE) within a row-address (A0–A10) defined page

MT4C1004J 883C
REV. 1/99
2-23



AUSTIN SEMICONDUCTOR, INC. MT4C1004J 883C 4 MEG x 1 DRAM

boundary. The FAST PAGE MODE cycle is always initiated with a row address strobed-in by RAS followed by a column address strobed-in by CAS. CAS may be toggled-in by holding RAS LOW and strobing-in different column addresses, thus executing faster memory cycles. Returning RAS HIGH terminates the FAST PAGE MODE operation. Returning RAS and CAS HIGH terminates a memory cycle and decreases chip current to a reduced standby level. Also, the chip is preconditioned for the next cycle during the RAS HIGH time. Memory cell data is retained in its correct state by maintaining power and executing any RAS cycle (READ, WRITE, RAS-ONLY, CAS-BEFORE-RAS, or HIDDEN REFRESH) so that all 1,024 combinations of RAS addresses (A0–A9) are executed at least every 16ms, regardless of sequence. The CAS- BEFORE-RAS cycle will invoke the refresh counter for automatic RAS addressing.

FUNCTIONAL BLOCK DIAGRAM
FAST PAGE MODE

*NOTE: WE LOW prior to CAS LOW; EW detection circuit output is a HIGH (EARLY WRITE)
CAS LOW prior to WE LOW; EW detection circuit output is a LOW (LATE-WRITE)

MT4C1004J 883C
REV. 1/99
2-24

Notes:

- Externally, the configuration of a device is described as something like

$$\delta \times \omega \times \beta$$

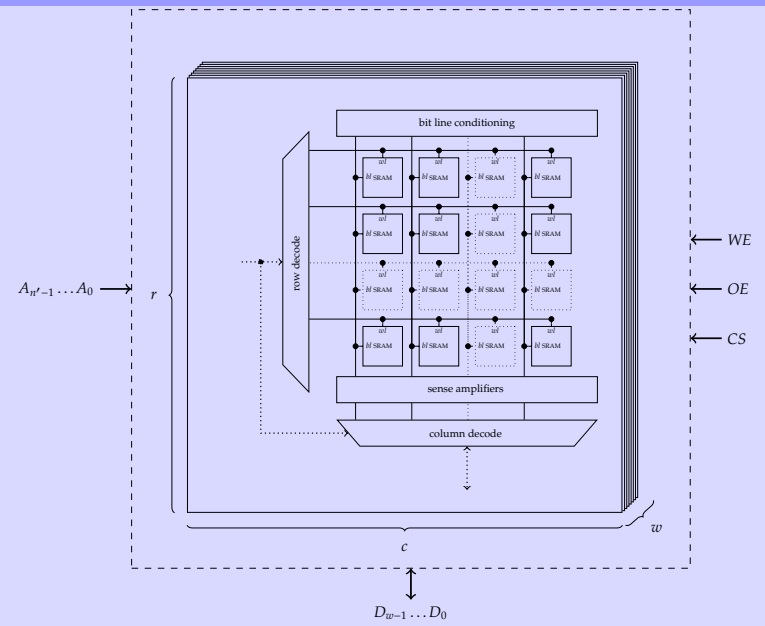
  (plus maybe some timing information) where
  - δ relates to capacity, usually measured in (large multiples of) bits,
  - ω describes the width of words, measured in bits,
  - β is the number of internal **logical banks**.
- Internally, this implies some organisational choices:
  1. for $\omega > 1$, we replicate the memory device internally to give ω arrays (each copy relates to one bit of a ω-bit word),
  2. for the arrays, r and c can be selected to match physical requirements (e.g., to get "square" or "thin" arrays), and
  3. for $\beta > 1$, each array is split into logical banks

  where in the latter case, the goal is to distribute the range of addresses across multiple smaller banks.

Notes:

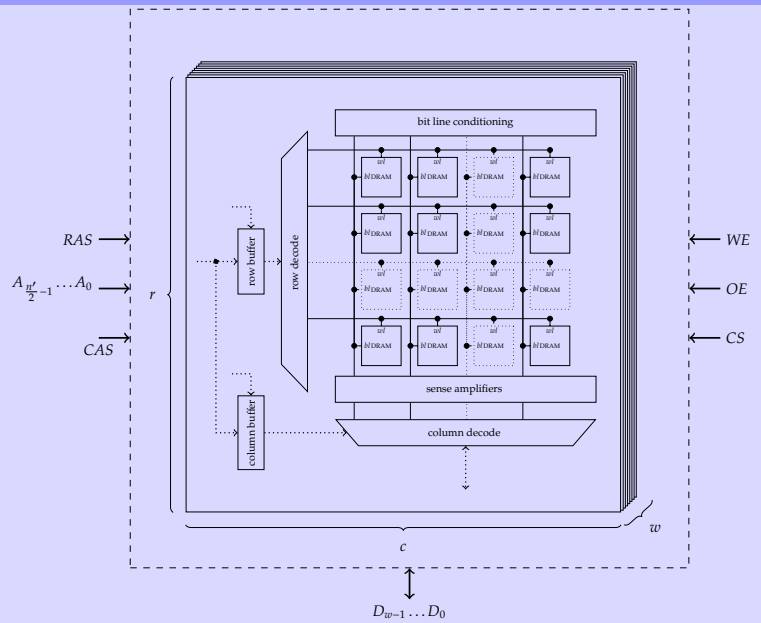
High(er)-level Implementation (11) – Cells ⇝ Device

Example (from a 1-bit to w-bit SRAM device via replication)



Notes:

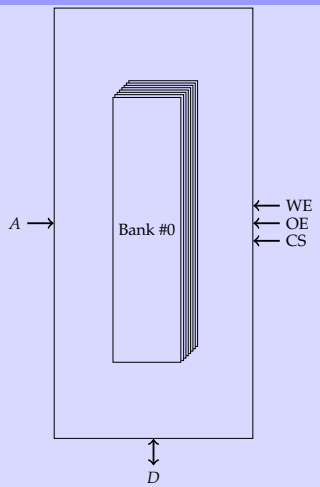
Example (from a 1-bit to w-bit DRAM device via replication)



Notes:

Example (a 512 Mbit, 8-bit Device)

- A 64 Mbit × 8 × 1 internal configuration:
  - The total capacity is 64 Mbit · 8 · 1 = 512 Mbit.
  - There is $\beta = 1$ logical bank, consisting of $\omega = 8$ arrays; each array has $\delta = r \cdot c = 64 \cdot 1024 \cdot 1024$ cells.
  - Address x refers to cell x of each array in the bank; for example $x = 42_{(10)}$ refers to cell 42 of each array, and therefore to an 8-bit word overall.



Notes:

Example (a 512 Mbit, 8-bit Device)

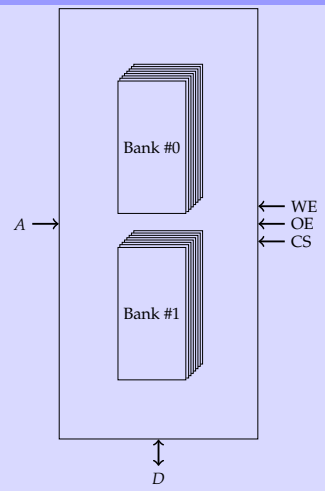
- A 32 Mbit $\times$ 8 $\times$ 2 internal configuration:
  - The total capacity is 32 Mbit $\cdot$ 8 $\cdot$ 2 = 512 Mbit.
  - There are $\beta = 2$ logical banks, each consisting of $\omega = 8$ arrays; each array has $\delta = r \cdot c = 32 \cdot 1024 \cdot 1024$ cells.
  - Address x refers to

$$x \mapsto \begin{cases} \text{bank} & x \bmod \beta \\ \text{cell} & x \operatorname{div} \beta \end{cases}$$

  of each array; for example,

  1. $x = 42_{(10)}$ refers to cell 21 of each array in bank 0, while
  2. $x = 43_{(10)}$ refers to cell 21 of each array in bank 1

  each case therefore referring to 8-bit word overall.



Notes:

High(er)-level Implementation (13) – Cells ⇝ Device

Example (a 512 Mbit, 8-bit Device)

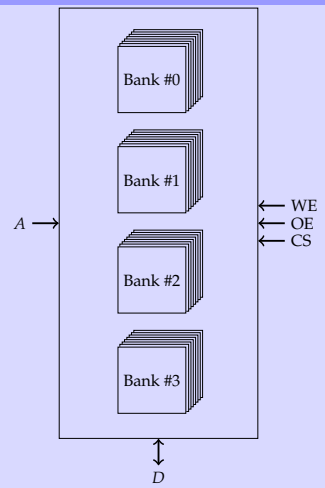
- A 16 Mbit $\times$ 8 $\times$ 4 internal configuration:
  - The total capacity is 16 Mbit $\cdot$ 8 $\cdot$ 4 = 512 Mbit.
  - There are $\beta = 4$ logical banks, each consisting of $\omega = 8$ arrays; each array has $\delta = r \cdot c = 16 \cdot 1024 \cdot 1024$ cells.
  - Address x refers to

$$x \mapsto \begin{cases} \text{bank} & x \bmod \beta \\ \text{cell} & x \operatorname{div} \beta \end{cases}$$

  of each array; for example,

  1. $x = 42_{(10)}$ refers to cell 10 of each array in bank 2, while
  2. $x = 43_{(10)}$ refers to cell 21 of each array in bank 3

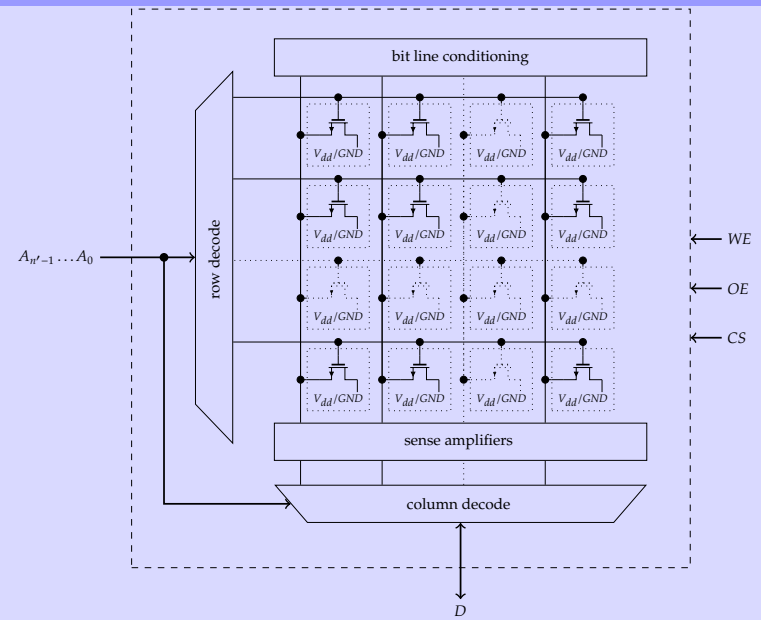
  each case therefore referring to 8-bit word overall.



Notes:

An Aside: "what about ROM devices?"

Example (an n-cell ROM memory device)



Notes:

High-level Implementation (1) – Device ⇝ Module

- To make devices easier to use (or integrate), they are typically packaged into a **memory module**:
  1. one or more memory devices, and
  2. an interface which controls access.
- There are lots of package types; two dominate
  1. **Single Inline Memory Module (SIMM)**, which is (roughly) 1-sided, has less pins and a narrower word size, and
  2. **Dual Inline Memory Module (DIMM)**, which is (roughly) 2-sided, has more pins and a wider word size

  and are capable of housing different device types (e.g., EDO *or* SDRAM devices in a given DIMM package).

Notes:

High-level Implementation (2) – Device $\rightsquigarrow$ Module

- Externally, the configuration of a module is described as something like

$$\Delta \times \Omega$$

  (plus maybe some timing information) where
  - Δ relates to capacity, usually measured in (large multiplies of) bytes,
  - Ω describes the width of words, measured in bits.
- Internally, some organisational choices exist
  1. usually $\Omega > \omega$ so the module is "filled" using multiple devices to form one **physical bank**, and
  2. depending on the module type, the devices are organised into one or more **ranks**

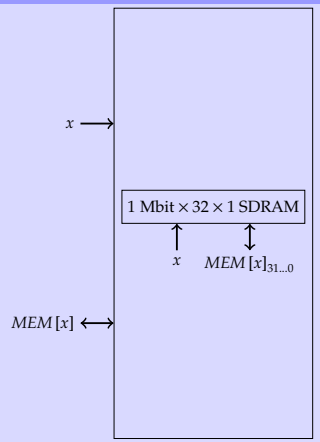
  where in the former case, the goal is to distribute content relating to a single address across multiple smaller devices.

Notes:

High-level Implementation (3) – Module $\rightsquigarrow$ Bank

Example (a 4 MB, 32-bit SIMM)

- A 4 MB $\times$ 32, 1-device configuration:
  - The device has a capacity of 1 Mbit $\cdot$ 32 $\cdot$ 1 = 32 Mbit = 4 MB arranged into 32-bit words.
  - The bank, which has a 32-bit data-path, is filled by the single device: address x refers to the word $MEM[x]$.



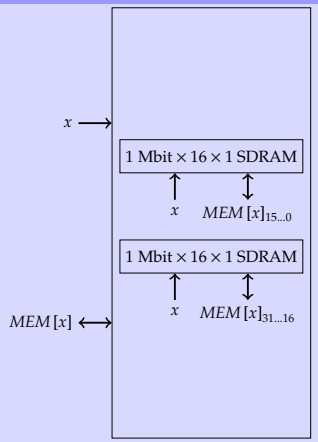
Notes:

Example (a 4 MB, 32-bit SIMM)

- A 4 MB × 32, 2-device configuration:
  - Each device has a capacity of 1 Mbit · 16 · 1 = 16 Mbit = 2 MB arranged into 16-bit words.
  - There are 2 devices; the total capacity is 2 · 2 MB = 4 MB.
  - The bank, which has a 32-bit data-path, is filled by merging the devices: address x refers to

$$x \mapsto \begin{cases} \text{bits} & 15 \ldots 0 & \text{of } MEM[x] \text{ from device 0} \\ \text{bits} & 31 \ldots 16 & \text{of } MEM[x] \text{ from device 1} \end{cases}$$

  merging each 16-bit part into the word $MEM[x]$.



Notes:

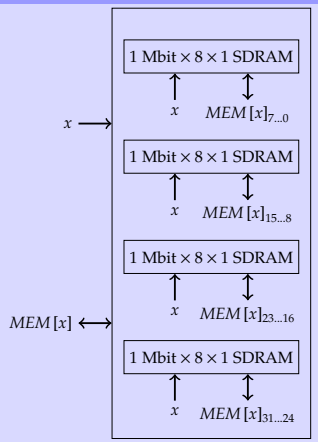
High-level Implementation (3) – Module ⇝ Bank

Example (a 4 MB, 32-bit SIMM)

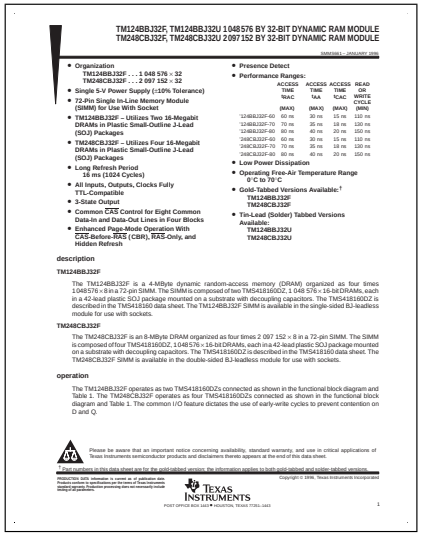
- A 4 MB × 32, 4-device configuration:
  - Each device has a capacity of 1 Mbit · 8 · 1 = 8 Mbit = 1 MB arranged into 8-bit words.
  - There are 4 devices; the total capacity is 4 · 1 MB = 4 MB.
  - The bank, which has a 32-bit data-path, is filled by merging the devices: address x refers to

$$x \mapsto \begin{cases} \text{bits} & 7 \ldots 0 & \text{of } MEM[x] \text{ from device 0} \\ \text{bits} & 15 \ldots 8 & \text{of } MEM[x] \text{ from device 1} \\ \text{bits} & 23 \ldots 16 & \text{of } MEM[x] \text{ from device 2} \\ \text{bits} & 31 \ldots 24 & \text{of } MEM[x] \text{ from device 3} \end{cases}$$

  merging each 8-bit part into the word $MEM[x]$.



Notes:



TM124BBJ32F, TM124BBJ32U 1 048 576 BY 32-BIT DYNAMIC RAM MODULE
TM248CBJ32F, TM248CBJ32U 2 097 152 BY 32-BIT DYNAMIC RAM MODULE

SMVS961 – JANUARY 1996

- Organization
  TM124BBJ32F . . . 1 048 576 × 32
  TM248CBJ32F . . . 2 097 152 × 32
- Single 5-V Power Supply (±10% Tolerance)
- 72-Pin Single In-Line Memory Module (SIMM) for Use With Socket
- TM124BBJ32F – Utilizes Two 16-Megabit DRAMs in Plastic Small-Outline J-Lead (SOJ) Packages
- TM248CBJ32F – Utilizes Four 16-Megabit DRAMs in Plastic Small-Outline J-Lead (SOJ) Packages
- Long Refresh Period
  16 ms (1024 Cycles)
- All Inputs, Outputs, Clocks Fully TTL-Compatible
- 3-State Output
- Common $\overline{CAS}$ Control for Eight Common Data-In and Data-Out Lines in Four Blocks
- Enhanced Page-Mode Operation With $\overline{CAS}$-Before-$\overline{RAS}$ (CBR), $\overline{RAS}$-Only, and Hidden Refresh
- Presence Detect
- Performance Ranges:

	ACCESS TIME t_{RAC} (MAX)	ACCESS TIME t_{AA} (MAX)	ACCESS TIME t_{CAC} (MAX)	READ OR WRITE CYCLE (MIN)
124BBJ32F-60	60 ns	30 ns	15 ns	110 ns
124BBJ32F-70	70 ns	35 ns	18 ns	130 ns
124BBJ32F-80	80 ns	40 ns	20 ns	150 ns
248CBJ32F-60	60 ns	30 ns	15 ns	110 ns
248CBJ32F-70	70 ns	35 ns	18 ns	130 ns
248CBJ32F-80	80 ns	40 ns	20 ns	150 ns

- Low Power Dissipation
- Operating Free-Air Temperature Range
  0°C to 70°C
- Gold-Tabbed Versions Available:†
  TM124BBJ32F
  TM248CBJ32F
- Tin-Lead (Solder) Tabbed Versions Available:
  TM124BBJ32U
  TM248CBJ32U

description

TM124BBJ32F

The TM124BBJ32F is a 4-MByte dynamic random-access memory (DRAM) organized as four times 1 048 576 × 8 in a 72-pin SIMM. The SIMM is composed of two TMS418160DZ, 1 048 576 × 16-bit DRAMs, each in a 42-lead plastic SOJ package mounted on a substrate with decoupling capacitors. The TMS418160DZ is described in the TMS418160 data sheet. The TM124BBJ32F SIMM is available in the single-sided BJ-leadless module for use with sockets.

TM248CBJ32F

The TM248CBJ32F is an 8-MByte DRAM organized as four times 2 097 152 × 8 in a 72-pin SIMM. The SIMM is composed of four TMS418160DZ, 1 048 576 × 16-bit DRAMs, each in a 42-lead plastic SOJ package mounted on a substrate with decoupling capacitors. The TMS418160DZ is described in the TMS418160 data sheet. The TM248CBJ32F SIMM is available in the double-sided BJ-leadless module for use with sockets.

operation

The TM124BBJ32F operates as two TMS418160DZs connected as shown in the functional block diagram and Table 1. The TM248CBJ32F operates as four TMS418160DZs connected as shown in the functional block diagram and Table 1. The common I/O feature dictates the use of early-write cycles to prevent contention on D and Q.

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

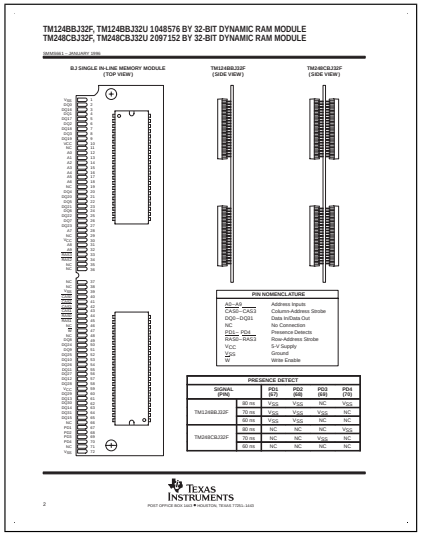
† Part numbers in this data sheet are for the gold-tabbed version; the information applies to both gold-tabbed and solder-tabbed versions.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

Copyright © 1996, Texas Instruments Incorporated

TEXAS INSTRUMENTS
POST OFFICE BOX 1443 • HOUSTON, TEXAS 77251–1443

1



TM124BBJ32F, TM124BBJ32U 1048576 BY 32-BIT DYNAMIC RAM MODULE
TM248CBJ32F, TM248CBJ32U 2097152 BY 32-BIT DYNAMIC RAM MODULE

SMVS961 – JANUARY 1996

BJ SINGLE IN-LINE MEMORY MODULE (TOP VIEW)

TM124BBJ32F (SIDE VIEW)

TM248CBJ32F (SIDE VIEW)

PIN NOMENCLATURE

A0–A9	Address Inputs
$CAS0$–$CAS3$	Column-Address Strobe
DQ0–DQ31	Data In/Data Out
NC	No Connection
PD1–PD4	Presence-Detects
$RAS0$–$RAS3$	Row-Address Strobe
V_{CC}	5-V Supply
V_{SS}	Ground
$\overline{W}$	Write Enable

PRESENCE DETECT

SIGNAL (PIN)		PD1 (67)	PD2 (68)	PD3 (69)	PD4 (70)
TM124BBJ32F	60 ns	V_{SS}	V_{SS}	NC	V_{SS}
	70 ns	V_{SS}	V_{SS}	V_{SS}	NC
	80 ns	V_{SS}	V_{SS}	NC	NC
TM248CBJ32F	60 ns	NC	NC	NC	V_{SS}
	70 ns	NC	NC	V_{SS}	NC
	80 ns	NC	NC	NC	NC

TEXAS INSTRUMENTS
POST OFFICE BOX 1443 • HOUSTON, TEXAS 77251–1443

2

Notes:

Conclusions

- Take away points:
  1. The initial goal was an n-element memory of w-bit words; the final solution is motivated by divide-and-conquer, i.e.,
     - 1.1 one or more channels, each backed by
     - 1.2 one or more physical banks, each composed from
     - 1.3 one or more devices, each composed from
     - 1.4 one or more logical banks, of
     - 1.5 one or more arrays, of
     - 1.6 many cells
  2. The major complication is a large range of increasingly detailed options:
     - lots of parameters mean lots of potential trade-offs (e.g., between size, speed and power consumption),
     - need to take care of detail: there are so many cells, any minor change can have major consequences!
  3. Even so, there is just one key concept: we have some cells, and however they are organised we just need to identify and use the right cells given some address.

Notes:

References and Further Reading

[1] Wikipedia: Computer memory.
http://en.wikipedia.org/wiki/Category:Computer_memory.

[2] Wikipedia: Dynamic random access memory.
http://en.wikipedia.org/wiki/Dynamic_random-access_memory.

[3] Wikipedia: Memory geometry.
http://en.wikipedia.org/wiki/Memory_geometry.

[4] Wikipedia: Static random access memory.
http://en.wikipedia.org/wiki/Static_random_access_memory.

[5] U. Drepper.
What every programmer should know about memory.
http://www.akkadia.org/drepper/cpumemory.pdf.

[6] D. Page.
Chapter 8: Memory and storage.
In *A Practical Introduction to Computer Architecture*. Springer-Verlag, 1st edition, 2009.

[7] W. Stallings.
Chapter 5: Internal memory.
In *Computer Organisation and Architecture*. Prentice-Hall, 9th edition, 2013.

[8] A.S. Tanenbaum.
Section 3.3.4: Memory organisation.
In *Structured Computer Organisation* [11].

Notes:

References and Further Reading

[9] A.S. Tanenbaum.
Section 3.3.5: Memory chips.
In *Structured Computer Organisation* [11].

[10] A.S. Tanenbaum.
Section 3.3.6: RAMs and ROMs.
In *Structured Computer Organisation* [11].

[11] A.S. Tanenbaum.
Structured Computer Organisation.
Prentice-Hall, 6th edition, 2012.

Notes: