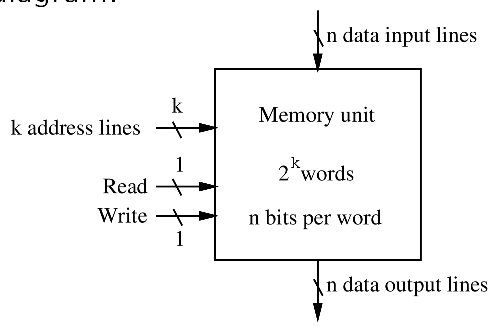
Types of Memory

- **Random-access memory (RAM):** A memory unit organized to allow data to be transferred **to or from any desired collection** of cells.
  - RAM can perform both the **read** and the **write** operations.
  - RAM takes the same time to access memory. (**Serial memory** takes different times to access information, e.g., disk, tape.)
- **Read-only memory (ROM):**
  - ROM can perform only the memory **read** operation.
  - A programmable logic device (PLD) that stores information by **programming** the device.

Memory Organization

- A memory block diagram:

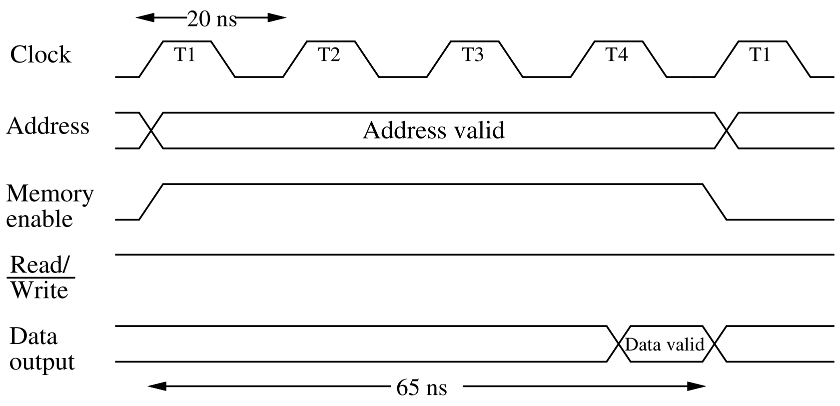


- A memory of 10 address bits, 16 data bits (1024×16 memory; $k = 10, n = 16$):

Memory address: Binary	Memory address: Decimal	Memory contents
0000000000	0	10110101 01011100
0000000001	1	10101011 10001001
0000000010	2	00001101 01000110
	⋮	⋮
1111111101	1021	10011101 00010101
1111111110	1022	00001101 00011110
1111111111	1023	11011110 00100100

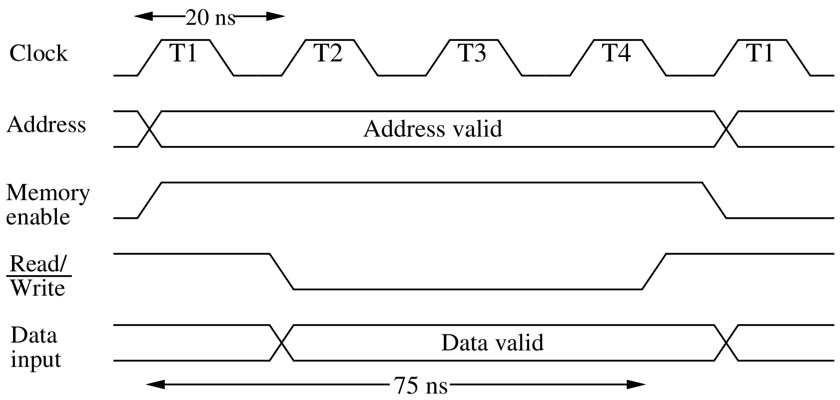
Memory Read Operation

- **Read memory:** A transfer of a copy of a stored word out of memory.
  1. Place the address of the desired word to the address line.
  2. Activate the READ input.



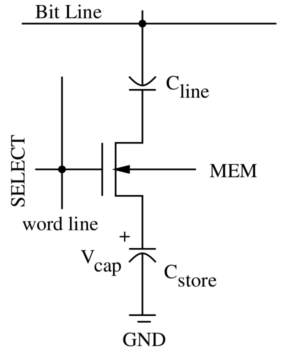
Memory Write Operation

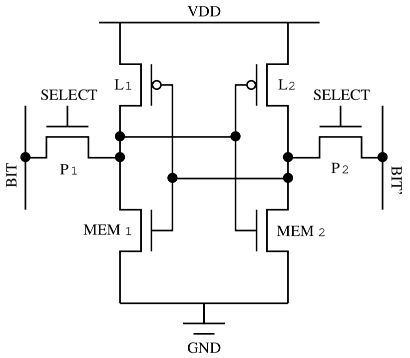
- **Write memory:** A transfer into memory of a new word to be stored.
  1. Apply the address of the desired word to the address line.
  2. Place the data to be stored to the data input lines.
  3. Activate the WRITE input.



Properties of Memory

- **Static RAM (SRAM)** consists of internal latches that stores the binary information.
  - The stored information remains valid as long as power is applied to the RAM.
- **Dynamic RAM (DRAM)** stores the binary information in the form of electric charges on capacitors.
  - The capacitors must be periodically recharged by **refreshing** the DRAM.
- DRAM cell vs. SRAM cell



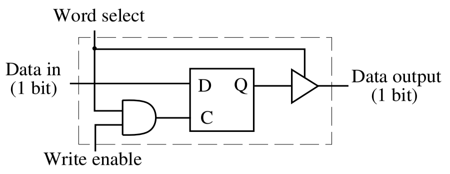


Properties of Memory (Cont'd)

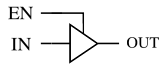
- Capacity: DRAM/SRAM ≈ 4–8; Cycle time: DRAM/SRAM ≈ 8–16; Cost: SRAM/DRAM ≈ 8–16.
- Typical applications: Cache ⟷ SRAM; Main memory ⟷ DRAM.
- **Volatile memory:** Memory units that **lose** stored information when power is turned off.
  - SRAM, DRAM, etc.
- **Non-volatile memory:** Memory units that **retain** stored information when power is turned off.
  - ROM, disk, tape, etc.

RAM Cell

- Modeling for a RAM cell (**Tri-state logic:** high, low, **high impedance (Hi-Z)** states)



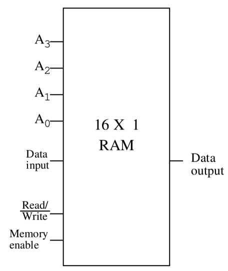
(a) RAM cell model



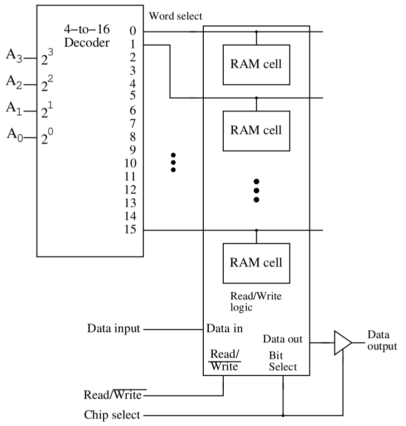
EN	IN	OUT
0	X	Hi–Z
1	0	0
1	1	1

(b) Three–state buffer

- A 16-word by 1-bit (16×1) RAM chip:



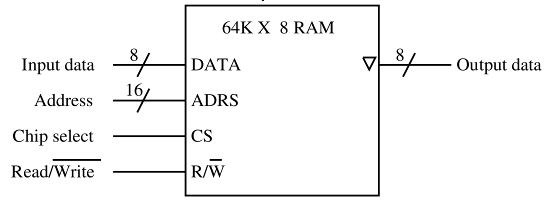
(a) Symbol



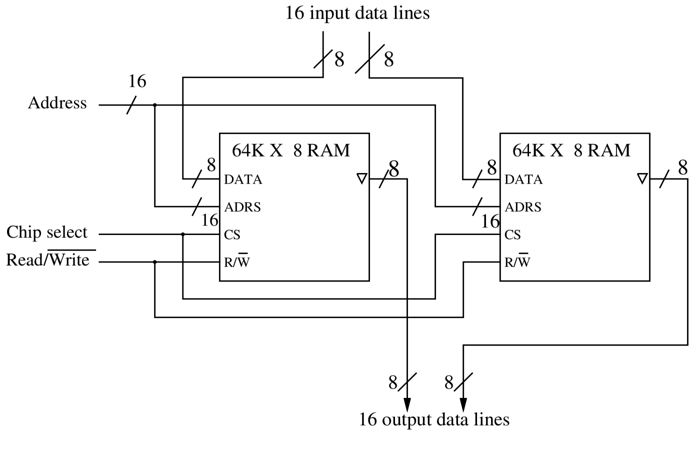
(b) Block diagram

Making Wider RAM

- Symbol for a $64K \times 8$ RAM chip (16 address lines, 8 data lines):

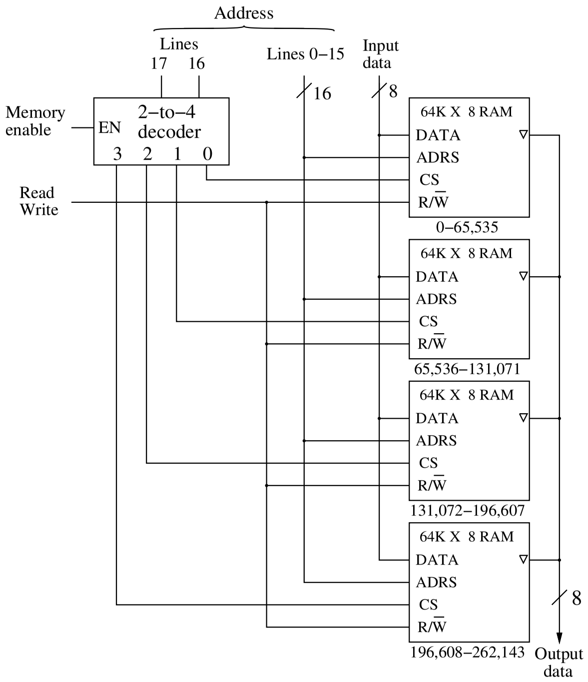


- A $64K \times 16$ RAM chip (16 address lines, 16 data lines):



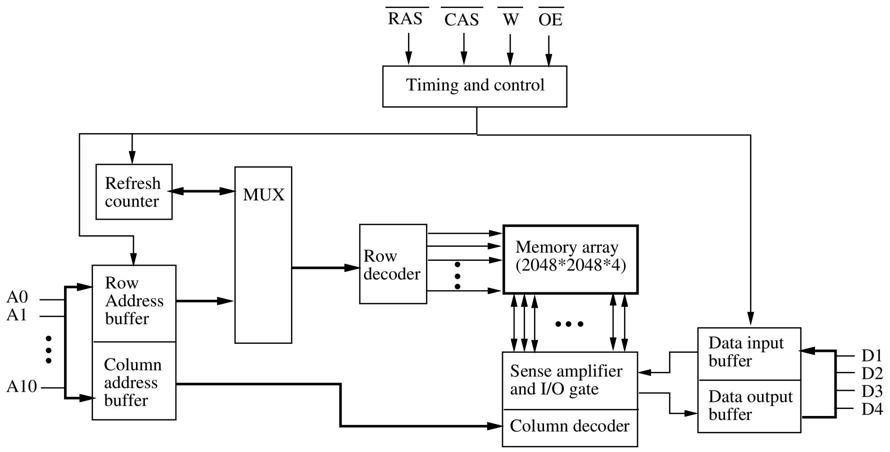
Making Larger RAM

- A $256K \times 8$ RAM chip (18 address lines, 8 data lines):



Typical DRAM Structure

- Organization for a typical $4M \times 4$ (16 megabit) DRAM
  - Address lines are multiplexed (RAS, then CAS), so only 11 address lines are needed.
  - 4 bits of data that may be read/written at a time.



Typical 16 megabit DRAM (4M * 4)

Memory Error Detection/Correction

- With larger memories, it is highly likely that data errors will occurs.
- These errors can be detected and corrected by using error-detecting and -correcting codes in RAMs.
  - **Single error detection:** A code which finds one data bit in error (e.g., parity).
  - **Single error correction:** A code which can detect a single error **and** correct it.
  - **Single error correction/double error detection:** A code which can detect single or double errors **and** correct single errors.
- **Add redundancy** to data, producing a code word such that even the existence of error(s) will allow you to identify the original data word.
- The most common memory error correcting codes in use are **Hamming Codes** (By R. W. Hamming in 1950).

Hamming Codes

- A Hamming code has n data bits and k check bits (total code word size: $n+k$ bits).
  - $2^k - 1 \geq n + k$.
- Exp: Deriving a single error correction code for the 8-bit data word **11000100**.

Bit position	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	1	P_4	1	0	0	P_8	0	1	0	0

- Each parity bit P_i is computed as follows (P_1: 001**1**, 010**1**, 011**1**, 100**1**, 101**1**):

$$
\begin{aligned}
P_1 &= XOR\ of\ bits(3,5,7,9,11) = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0 \\
P_2 &= XOR\ of\ bits(3,6,7,10,11) = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0 \\
P_4 &= XOR\ of\ bits(5,6,7,12) = 1 \oplus 0 \oplus 0 \oplus 0 = 1 \\
P_8 &= XOR\ of\ bits(9,10,11,12) = 0 \oplus 1 \oplus 0 \oplus 0 = 1
\end{aligned}
$$

- The resulting 12-bit word which is written to memory:

Bit position	1	2	3	4	5	6	7	8	9	10	11	12
	<u>0</u>	**<u>0</u>**	1	**<u>1</u>**	1	0	0	**<u>1</u>**	0	1	0	0

Hamming Codes (Cont'd)

- How to check a 12-bit word read from memory for error correction?
- Compute the 4 check bits as follows (C_1: 000**1**, 001**1**, 010**1**, 011**1**, 100**1**, 101**1**):

$$\begin{aligned} C_1 &= XOR\ of\ bits(1,3,5,7,9,11) \\ C_2 &= XOR\ of\ bits(2,3,6,7,10,11) \\ C_4 &= XOR\ of\ bits(4,5,6,7,12) \\ C_8 &= XOR\ of\ bits(8,9,10,11,12) \end{aligned}$$

- For single error detection: If $C = C_8C_4C_2C_1 = 0$, no error; otherwise, bit C is in error.

Bit	1	2	3	4	5	6	7	8	9	10	11	12	
Case 1	0	0	1	1	1	0	0	1	0	1	0	0	No error
Case 2	<u>**1**</u>	0	1	1	1	0	0	1	0	1	0	0	Error in bit 1
Case 3	0	0	1	1	<u>**0**</u>	0	0	1	0	1	0	0	Error in bit 5

Case	Error?	C	C_8	C_4	C_2	C_1
1	No error	0	0	0	0	0
2	Error in bit 1	1	0	0	0	1
3	Error in bit 5	5	0	1	0	1

Single Error Correction/Double Error Detection

- The above code cannot detect **double** errors.
- By adding an **overall parity bit**, we can generate a code which **detects** double errors.
  - E.g., 001110010100P_{13}
- For even parity,

Syndrome C	Parity (overall)	Meaning
$C = 0$	$P = 0$	No errors
$C \neq 0$	$P = 1$	A single, correctable error occurred
$C \neq 0$	$P = 0$	A double error is detected and cannot be corrected
$C = 0$	$P = 1$	The parity bit is in error

IC Fabrication Dilemma

- IC fabrication dilemma:
  - IC logic density increases **exponentially** with time.
  - The more ICs you make, the **cheaper** they get.
  - Complex logic ICs have very **specific** functions, so you make **fewer**.
- **Question:** How do I make very high volume parts that are very dense?
  - Ans #1: Microprocessors that have very large volume (e.g., automobile ignition controllers, etc).
  - Ans #2: **Programmable logic.**

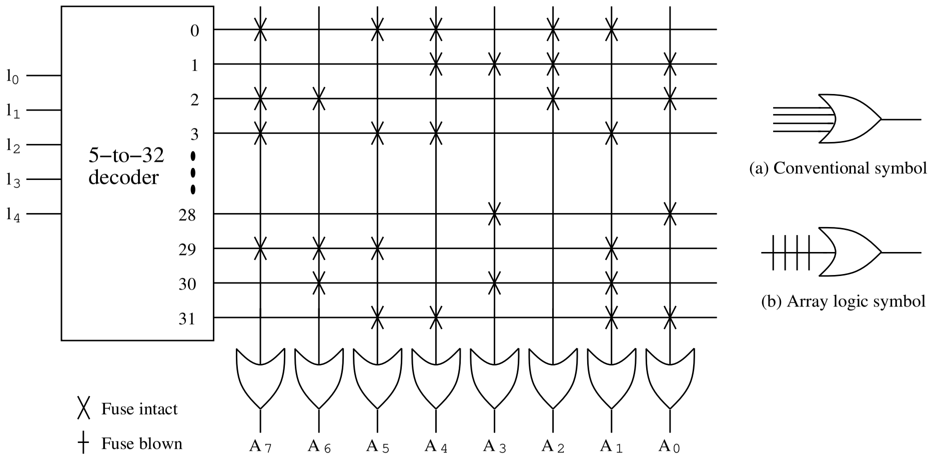
Programmable Logic Family

1. **Simple Programmable Logic Device (SPLD)** (mostly, 2-level)
   - **Read Only Memory (ROM):** A **fixed** array of **AND** gates and a **programmable** array of **OR** gates.
   - **Programmable Logic Array (PLA):** A **programmable** array of **AND** gates feeding a **programmable** array of **OR** gates.
   - **Programmable Array Logic (PAL):** A **programmable** array of **AND** gates feeding a **fixed** array of **OR** gates.
2. **Complex Programmable Logic Device (CPLD):** Multiple SPLDs incorporated on a single chip with programmable switching network interconnecting them.
3. **Field Programmable Gate Array (FPGA):** A **(re)programmable** logic device that can be used to implement multi-level logic and configured by the end users.

- Devices may be
  - **Permanently programmed** at the time of IC manufacture,
  - **Programmed at the time of use**, or
  - **Dynamically re-programmed** during use

Read Only Memory (ROM)

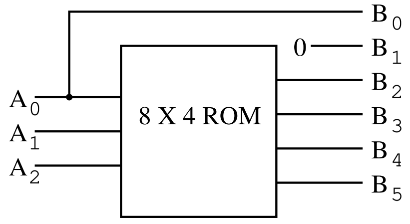
- ROM or Programmable ROM (PROM) have n **input,** m **output** lines and 2^n decoded minterms.



ROM Implementation

- Implement the circuit that accepts a 3-bit number and generates an output binary number equal to the square of the input number.

Inputs			Outputs						
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	Decimal
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49

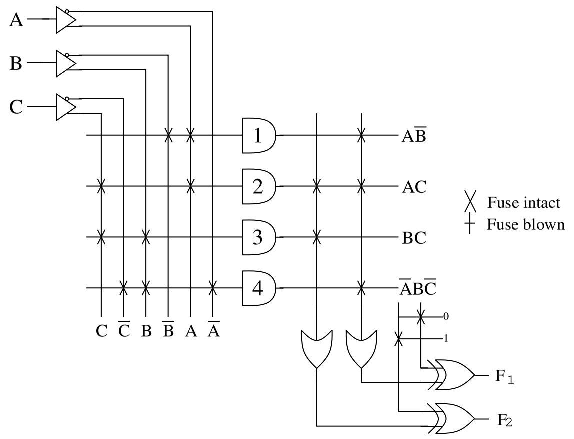


Block diagram

Programmable Logic Array (PLA)

- The AND terms limit the application of a PLA, so try to reduce the # of AND terms to fit in a PLA.
- Implement $F_1 = A\bar{B} + AC + \bar{A}B\bar{C}$ and $F_2 = \overline{AC + BC}$.

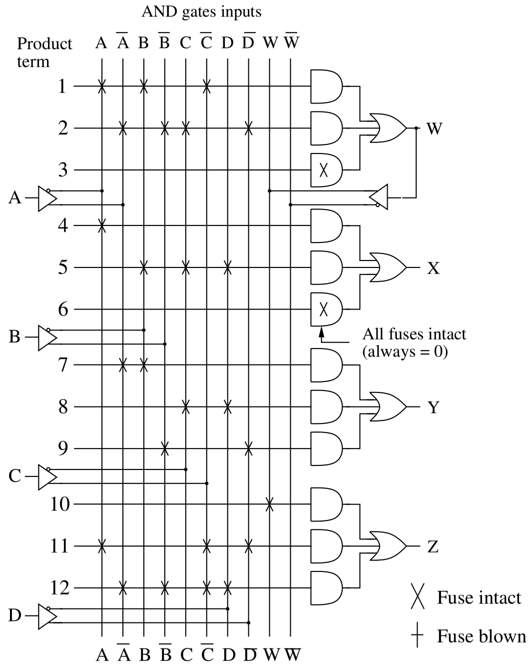
	Product term	Inputs A	B	C	Outputs (T) F_1	(C) F_2
$A\bar{B}$	1	1	0	–	1	–
AC	2	1	–	1	1	1
BC	3	–	1	1	–	1
$\bar{A}B\bar{C}$	4	0	1	0	1	–



Programmable Array Logic (PAL)

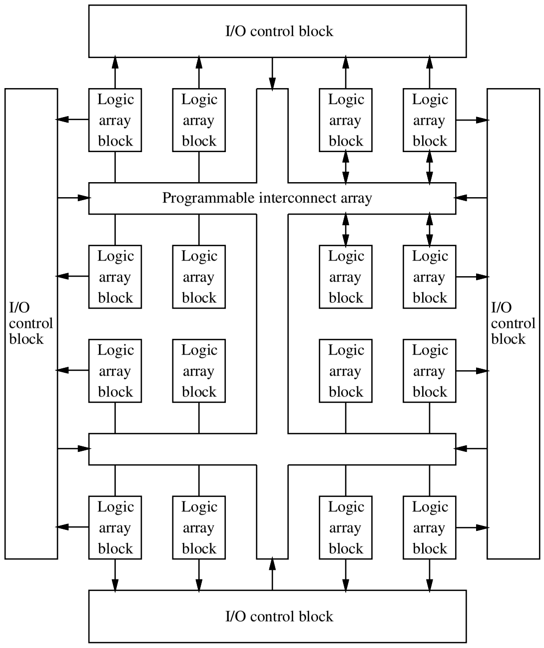
- The OR terms limit the application of a PAL, so try to reduce the # of OR terms to fit in a PAL.

Product term	AND inputs A	B	C	D	W	Outputs
1	1	1	0	–	0	$W = AB\bar{C}$
2	0	0	1	0	–	$+ \bar{A}\bar{B}C\bar{D}$
3	–	–	–	–	–	
4	1	–	–	–	–	$X = A$
5	–	1	1	1	–	$+ BCD$
6	–	–	–	–	–	
7	0	1	–	–	1	$Y = \bar{A}B$
8	–	–	1	1	–	$+CD$
9	–	0	–	0	–	$+\bar{B}\bar{D}$
10	–	–	–	–	–	$Z = W$
11	1	–	0	0	–	$+ A\bar{C}\bar{D}$
12	0	0	0	1	–	$+\bar{A}\bar{B}\bar{C}D$



Complex Programmable Logic Device (CPLD)

- A programmable logic device that incorporates multiple SPLDs on a single chip with programmable switching network interconnecting them.



Field Programmable Gate Array (FPGA)

- A **(re)programmable** logic device that can be used to implement multi-level logic and configured by the end users.