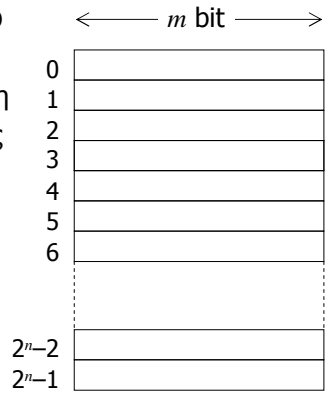
Σχεδίαση Υπολογιστικών Συστημάτων

Σχεδίαση μνημών

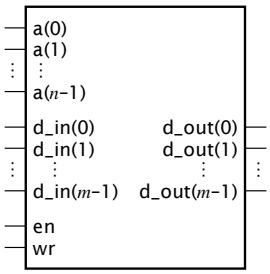
Μιχάλης Ψαράκης

Γενικές έννοιες

- Η μνήμη είναι ένας πίνακας από θέσεις αποθήκευσης
  - Καθεμία με μια μοναδική διεύθυνση
  - Σαν μια συλλογή από καταχωρητές
- Η διεύθυνση είναι κωδικοποιημένη σε δυαδικό
  - n bit διεύθυνσης ⇒ 2^n θέσεις
- Όλες οι θέσεις έχουν το ίδιο μέγεθος
  - $2^n \times m$ bit μνήμης



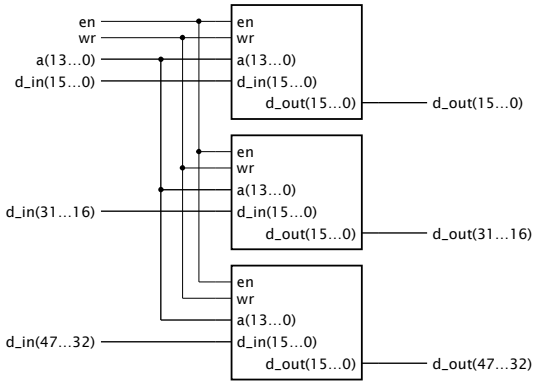
Βασικές λειτουργίες μνήμης



- είσοδοι a : διεύθυνση
- d_in και d_out
  - ο τύπος εξαρτάται από την εφαρμογή
- Λειτουργία εγγραφής (write)
  - en = 1, wr = 1
  - η τιμή d_in αποθηκεύεται στη θέση που καθορίζεται από τη διεύθυνση
- Λειτουργία ανάγνωσης (read)
  - en = 1, wr = 0
  - στο d_out οδηγείται η τιμή της θέσης που καθορίζεται από τη διεύθυνση
- Αδράνεια (idle): en = 0

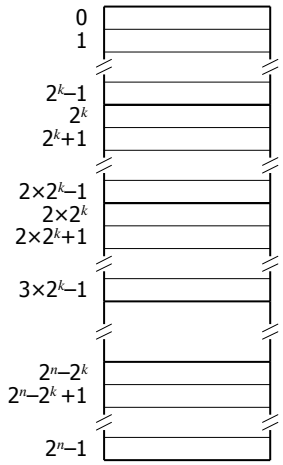
Μνήμες με μεγαλύτερο εύρος

- Τα τσιπ μνήμης έχουν ένα σταθερό εύρος
  - Π.χ., ×1, ×4, ×8, ×16, ...
- Χρησιμοποιούμε τσιπ μνήμης παράλληλα για να σχηματίσουμε μια μνήμη με μεγαλύτερο εύρος
  - Π.χ, τρία τσιπ 16K×16 για μια μνήμη 16K×48



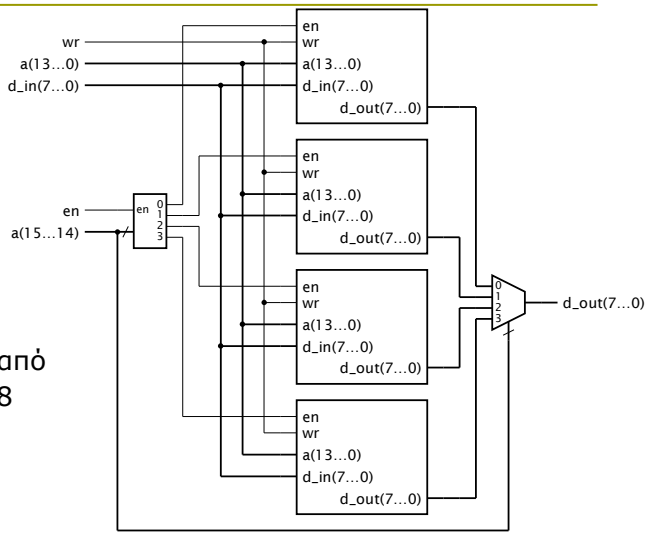
Περισσότερες θέσεις

- Για να έχουμε 2^n θέσεις με τσιπ μνήμης των 2^k θέσεων
  - Χρησιμοποιούμε $2^n/2^k$ τσιπ
- Διεύθυνση A
  - σχετική απόσταση A mod 2^k
    - τα k λιγότερο σημαντικά bit του A
  - στο τσιπ $\lfloor A/2^k \rfloor$
    - τα $n-k$ περισσότερο σημαντικά bit της διεύθυνσης A
    - με αποκωδικοποίηση επιλέγουμε τσιπ



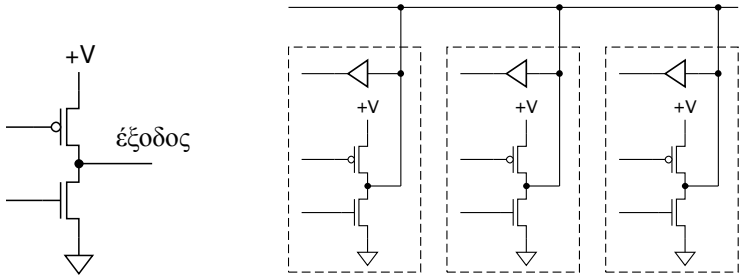
Περισσότερες θέσεις

- Παράδειγμα: μνήμη 64K×8 αποτελούμενη από τσιπ των 16K×8



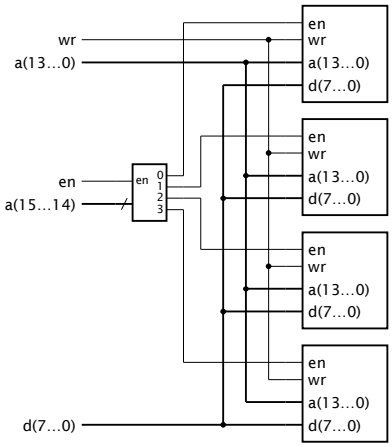
Οδηγοί τριών καταστάσεων

- Επιτρέπουν να συνδεθούν μαζί πολλαπλοί οδηγοί
  - Μόνο ένας είναι ενεργός κάθε στιγμή
  - Οι υπόλοιπες έξοδοι είναι σε υψηλή εμπέδηση (hi-Z)
    - Και τα δύο τρανζίστορ είναι κλειστά (off)
- Επιτρέπουν εισόδους/εξόδους διπλής κατεύθυνσης



Μνήμες με θύρες τριών καταστάσεων

- Κατά την εγγραφή
  - οι οδηγοί της θύρας d από τη μνήμη είναι hi-Z
  - η μνήμη διαβάζει την d
- Κατά την ανάγνωση
  - η επιλεγμένη μνήμη οδηγεί τη θύρα d
- Λιγότεροι ακροδέκτες και καλώδια
  - μειωμένο κόστος του PCB

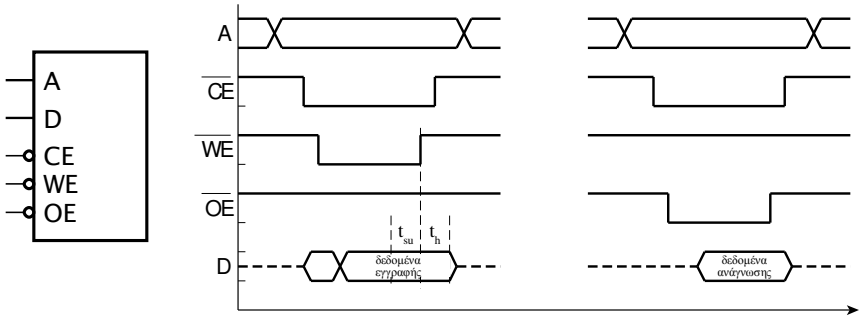


Τύποι μνήμης

- Μνήμη Τυχαίας Προσπέλασης (Random-Access Memory - RAM)
  - Μπορεί να γραφτεί και να διαβαστεί
  - Στατική RAM (Static RAM - SRAM)
    - Διατηρεί τα δεδομένα όσο εφαρμόζεται τροφοδοσία
    - Ασύγχρονη SRAM (Asynchronous SRAM): χωρίς ρολόι
    - Σύγχρονη SRAM (Synchronous SRAM - SSRAM): με ρολόι
  - Δυναμική RAM (Dynamic RAM - DRAM)
    - Χρειάζεται να ανανεώνεται περιοδικά
- Μνήμη Μόνο για Ανάγνωση (Read-Only Memory - ROM)
  - Συνδυαστική
  - Προγραμματιζόμενη και επανεγγράψιμη Flash

Ασύγχρονη SRAM

- Τα δεδομένα αποθηκεύονται σε κελιά μανδάλωσης του 1 bit
  - Η διεύθυνση αποκωδικοποιείται για να ενεργοποιήσει ένα κελί
- Συνήθως, οι είσοδοι ελέγχου είναι χαμηλού ενεργού
- Δεν είναι διαθέσιμες ως στοιχεία σε ASIC ή FPGA

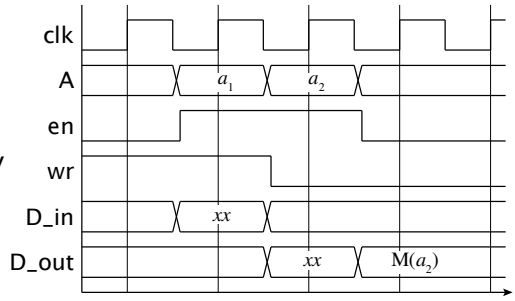


Χρονισμός ασύγχρονης SRAM

- Οι παράμετροι χρονισμού δημοσιεύονται στα φύλλα δεδομένων
- Χρόνος προσπέλασης (access time)
  - Από έγκυρη διεύθυνση/ενεργοποίηση μέχρι έγκυρα δεδομένα στην έξοδο
- Χρόνος κύκλου (cycle time)
  - Από την αρχή μέχρι το τέλος της προσπέλασης
- Προπαρασκευή και διατήρηση δεδομένων (data setup and hold)
  - Πριν/μετά το τέλος του παλμού WE
  - Κάνει δύσκολη τη χρήση των ασύγχρονων SRAM σε σύγχρονες σχεδιάσεις με ρολόι

Σύγχρονη SRAM (SSRAM)

- Καταχωρητές αποθήκευσης με ρολόι για τις εισόδους
  - διεύθυνση, δεδομένα και είσοδοι ελέγχου
  - αποθηκεύονται σε μια ακμή ρολογιού
  - παραμένουν για τον κύκλο ανάγνωσης/εγγραφής
- Flow-through SSRAM (διαμέσου ροής)
  - χωρίς καταχωρητή στην έξοδο δεδομένων



Μνήμες στην VHDL

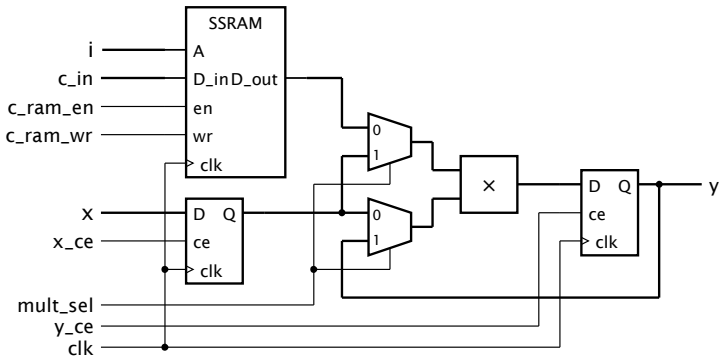
- Η αποθήκευση της RAM αναπαρίσταται από ένα σήμα πίνακα

```
type RAM_4Kx16 is array (0 to 4095) of std_logic_vector(15 downto 0);
signal data_RAM : RAM_4Kx16;
...
data_RAM_flow_through : process (clk) is
begin
  if rising_edge(clk) then
    if en = '1' then
      if wr = '1' then
        data_RAM(to_integer(a)) <= d_in;  d_out <= d_in;
      else
        d_out <= RAM(to_integer(a));
      end if;
    end if;
  end if;
end process data_RAM_flow_through;
```

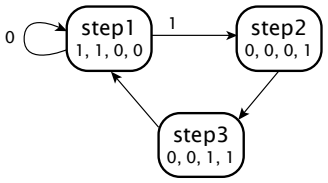

Παράδειγμα: Πολ/στής Συντελεστών

- Υπολογίστε τη συνάρτηση $y = c_i \times x^2$
  - Ο συντελεστής είναι αποθηκευμένος σε μια flow-through SSRAM
    - ακέραιος δείκτης των 12 bit για το i
  - x, y, c_i προσημασμένες τιμές σταθερής υποδιαστολής των 20 bit
    - 8 bit πριν και 12 bit μετά τη δυαδική υποδιαστολή
  - Είσοδος *start*: δηλώνει την άφιξη νέων τιμών στις εισόδους x και i
  - Χρησιμοποιείστε έναν απλό πολλαπλασιαστή
    - Πολλαπλασιάστε $c_i \times x \times x$

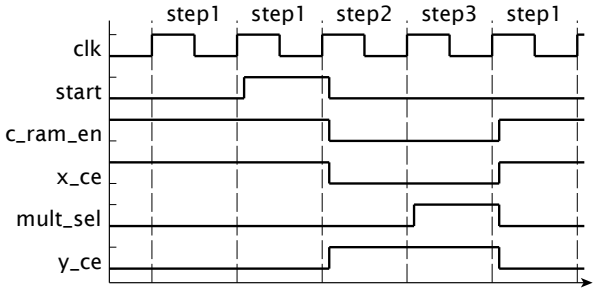
Διαδρομή Δεδομένων Πολ/στή



Χρονισμός και Έλεγχος Πολ/στή



Σήματα ελέγχου:
`c_ram_en, x_ce, mult_sel, y_ce`



Παράδειγμα: Πολ/στής Συντελεστών

```
library ieee; use ieee.std_logic_1164.all,
                  ieee.numeric_std.all, ieee.fixed_pkg.all;

entity scaled_square is
  port ( clk, reset : in std_logic;
         start : in std_logic;
         i : in unsigned(11 downto 0);
         c_in, x : in sfixed(7 downto -12);
         y : out sfixed(7 downto -12) );
end entity scaled_square;
```

```
architecture rtl of scaled_square is

  signal c_ram_en, c_ram_wr, x_ce, mult_sel, y_ce : std_logic;
  signal c_out, x_out : sfixed(7 downto -12);
  signal y_out : sfixed(7 downto -12);

  type c_array is array (0 to 4095) of sfixed(7 downto -12);
  signal c_RAM : c_array;

  type state is (step1, step2, step3);
  signal current_state, next_state : state;
```


Παράδειγμα: Πολ/στής Συντελεστών

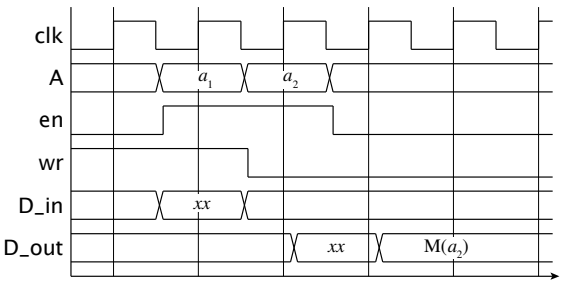
```
begin

  c_ram_wr <= '0';

  c_RAM_flow_through : process (clk) is
  begin
    if rising_edge(clk) then
      if c_ram_en = '1' then
        if c_ram_wr = '1' then
          c_RAM(to_integer(i)) <= c_in;
          c_out <= c_in;
        else
          c_out <= c_RAM(to_integer(i));
        end if;
      end if;
    end if;
  end process c_RAM_flow_through;
```

Παράδειγμα: Πολ/στής Συντελεστών

```
  y_reg : process (clk) is
    variable operand1, operand2 : sfixed(7 downto -12);
  begin
    if rising_edge(clk) then
      if y_ce = '1' then
        if mult_sel = '0' then
          operand1 := c_out;  operand2 := x_out;
        else
          operand1 := x_out;  operand2 := y_out;
        end if;
        y_out <= operand1 * operand2;
      end if;
    end if;
  end process y_reg;

  y <= y_out;

  state_reg : process ...
  next_state_logic : process ...
  output_logic : process ...
end architecture rtl;
```


SSRAM με διοχέτευση

- Και η έξοδος δεδομένων έχει καταχωρητή
  - Πιο κατάλληλη για συστήματα υψηλής ταχύτητας
  - Προσπελαύνει την RAM σε ένα κύκλο, χρησιμοποιεί τα δεδομένα στον επόμενο κύκλο



SSRAM με διοχέτευση στην VHDL

```
data_RAM_pipelined : process (clk) is
  variable pipeline_en : std_logic;
  variable pipeline_d_out : std_logic_vector(15 downto 0);
begin
  if rising_edge(clk) then
    if pipelined_en = '1' then
      d_out <= pipelined_d_out;
    end if;
    pipeline_en := en;
    if en = '1' then
      if wr = '1' then
        data_RAM(to_integer(a)) <= d_in; pipeline_d_out := d_in;
      else
        pipeline_d_out := RAM(to_integer(a));
      end if;
    end if;
  end if;
end process data_RAM_pipelined;
```

καταχωρητής εξόδου

SSRAM

Μνήμες Πολλαπλών Θυρών

- Πολλαπλές συνδέσεις διεύθυνσης, δεδομένων και ελέγχου στις θέσεις αποθήκευσης
- Επιτρέπει ταυτόχρονες προσπελάσεις
  - Αποφεύγει την πολυπλεξία και την ακολουθία ελέγχου
- Σενάριο
  - Ένα σύστημα παράγει, ένα σύστημα καταναλώνει δεδομένα
- Τι θα συμβεί ένα συμβούν ταυτόχρονα δύο εγγραφές σε μία θέση;
  - Το αποτέλεσμα μπορεί να είναι απρόβλεπτο
  - Κάποιες μνήμες πολλαπλών θυρών περιλαμβάνουν κύκλωμα διαιτησίας (arbiter)

Παράδειγμα: dual-port SSRAM

- Αναπτύξτε το μοντέλο VHDL μίας dual-port (διπλής θύρας) SSRAM μεγέθους 4K x 16 bit
  - Θύρα a1: εγγραφή και ανάγνωση
  - Θύρα a2: μόνο ανάγνωση

```
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;

entity dual_port_SSRAM is
  port ( clk      : in  std_logic;
         en1, wr1 : in  std_logic;
         a1       : in  unsigned(11 downto 0);
         d_in1    : in  std_logic_vector(15 downto 0);
         d_out1   : out std_logic_vector(15 downto 0);
         en2      : in  std_logic;
         a2       : in  unsigned(11 downto 0);
         d_out2   : out std_logic_vector(15 downto 0) );
end entity dual_port_SSRAM;
```


Παράδειγμα: dual-port SSRAM

```
architecture synth of dual_port_SSRAM is
  type RAM_4Kx16 is array (0 to 4095)
                    of std_logic_vector(15 downto 0);
  signal data_RAM : RAM_4Kx16;

begin

  read_write_port : process (clk) is
  begin
    if rising_edge(clk) then
      if en1 = '1' then
        if wr1 = '1' then
          data_RAM(to_integer(a1)) <= d_in1;  d_out1 <= d_in1;
        else
          d_out1 <= data_RAM(to_integer(a1));
        end if;
      end if;
    end if;
  end process read_write_port;
...
```

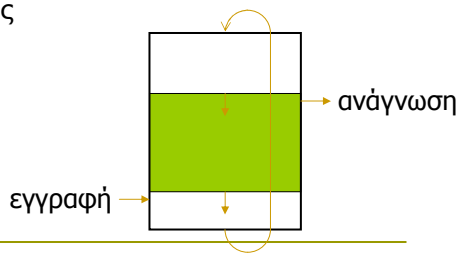
Παράδειγμα: dual-port SSRAM

```
...

  read_only_port : process (clk) is
  begin
    if rising_edge(clk) then
      if en2 = '1' then
        d_out2 <= data_RAM(to_integer(a2));
      end if;
    end if;
  end process read_only_port;

end architecture synth;
```

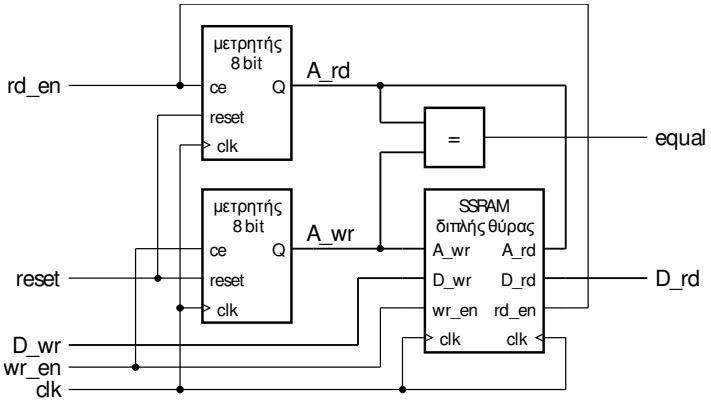

Μνήμες FIFO

- First-In/First-Out (FIFO) προσωρινή μνήμη (buffer)
  - Συνδέει το σύστημα που παράγει με αυτό που καταναλώνει
  - Αποσυνδέει τους ρυθμούς παραγωγής και κατανάλωσης

- Υλοποίηση με RAM διπλής θύρας
  - Κυκλική προσωρινή μνήμη
  - Γεμάτη (full):
    write address = read address
  - Άδεια (empty):
    write address = read address



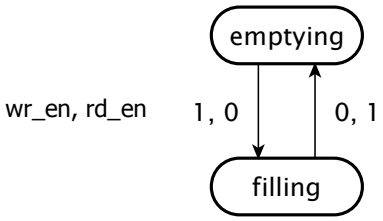
Παράδειγμα: FIFO - Datapath



- Equal = γεμάτη ή άδεια
  - Χρειάζεται να διακρίνουμε αυτές τις καταστάσεις — Πώς;

Παράδειγμα: FIFO - Control

- FSM ελέγχου
  - → filling (γεμίζει): εγγραφή χωρίς ταυτόχρονη ανάγνωση
  - → emptying (αδειάζει): ανάγνωση χωρίς ταυτόχρονη εγγραφή
  - αμετάβλητη όταν συμβεί ταυτόχρονη εγγραφή και ανάγνωση



full = filling and equal

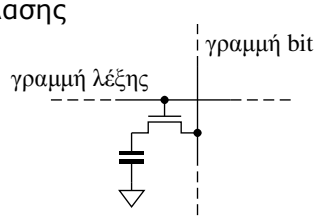
empty = emptying and equal

Πολλαπλά Πεδία Ρολογιού

- Χρειάζεται να επανασυγχρονίσουμε δεδομένα που διασταυρώνονται μεταξύ πεδίων ρολογιού
  - Χρήση καταχωρητών επανασυγχρονισμού
- Μπορεί να συμβεί υπερχείλιση εάν το ρολόι του αποστολέα είναι γρηγορότερο από το ρολόι του παραλήπτη
- Μια FIFO εξισορροπεί αυτές τις διαφορές στους ρυθμούς ροής δεδομένων
  - Τα κελιά μέσα στην RAM της FIFO γράφονται με το ρολόι του αποστολέα και διαβάζονται με το ρολόι του παραλήπτη

Δυναμική RAM (DRAM)

- Τα δεδομένα αποθηκεύονται σε ένα κελί του 1 τρανζίστορ/1 πυκνωτή
  - μικρότερο κελί από την SRAM, άρα περισσότερα ανά τσιπ
  - αλλά μεγαλύτερος χρόνος προσπέλασης
- Λειτουργία εγγραφής
  - έλκει τη γραμμή bit στο υψηλό ή το χαμηλό (0 or 1)
  - ενεργοποιεί τη γραμμή λέξης
- Λειτουργία ανάγνωσης
  - προ-φορτίζει τη γραμμή σε μια ενδιάμεση τάση
  - ενεργοποιεί τη γραμμή λέξης και ανιχνεύει την ισοστάθμιση των φορτίων
  - επανεγγράφει για να αποκαταστήσει το φορτίο



Ανανέωση της DRAM

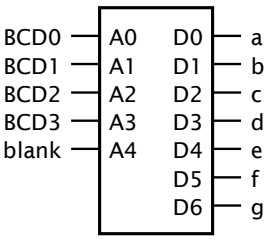
- Το φορτίο του πυκνωτή χάνεται με το χρόνο
  - Χρειάζεται να διαβάζει και να επανεγγράφει περιοδικά
    - Συνήθως, κάθε κελί ανά 64ms
  - *Ανανεώνει (refresh)* κάθε θέση
- Οι DRAM οργανώνονται σε σειρές (banks) από γραμμές
  - Ανανεώνει ολόκληρη τη γραμμή κάθε φορά
- Δεν μπορεί να προσπελαστεί κατά την ανανέωση
  - Η ανανέωση παρεμβάλλεται μεταξύ προσπελάσεων
  - ή γίνεται σε μια ριπή κάθε 64ms

Μνήμη Μόνο για Ανάγνωση (ROM)

- Για σταθερά δεδομένα ή προγράμματα CPU
- ROM με μάσκα (masked ROM)
  - Τα δεδομένα ενσωματώνονται στη μνήμη κατά την κατασκευή
- Προγραμματιζόμενη (programmable ROM - PROM)
  - Χρησιμοποιείται μια συσκευή προγραμματισμού PROM
- Διαγράψιμες PROM (erasable PROM - EPROM)
  - Διαγράψιμες με υπεριώδεις ακτίνες φωτός (UV)
  - Ηλεκτρικά διαγράψιμες (Electrically erasable - EEPROM)
  - Flash RAM

Συνδυαστικές ROM

- Μία ROM αντιστοιχίζει μια διεύθυνση εισόδου σε μια έξοδο δεδομένων
  - Αυτό είναι ένα συνδυαστικό κύκλωμα!
  - Καθορίζει τη συνάρτηση με έναν πίνακα
- Παράδειγμα: Αποκωδικοποιητής 7 τμημάτων



Διεύθυνση	Περιεχόμενο	Διεύθυνση	Περιεχόμενο
0	0111111	6	1111101
1	0000110	7	0000111
2	1011011	8	1111111
3	1001111	9	1101111
4	1100110	10–15	1000000
5	1101101	16–31	0000000

Παράδειγμα: ROM στην VHDL

```
library ieee; use ieee.numeric_std.all;
architecture ROM_based of seven_seg_decoder is
  type ROM_array is
      array (0 to 31) of std_logic_vector ( 7 downto 1 );
  constant ROM_content : ROM_array
    := ( 0 => "0111111", 1 => "0000110",
         2 => "1011011", 3 => "1001111",
         4 => "1100110", 5 => "1101101",
         6 => "1111101", 7 => "0000111",
         8 => "1111111", 9 => "1101111"
         10 to 15 => "1000000",
         16 to 31 => "0000000" );
begin
  seg <= ROM_content(to_integer(unsigned(blank & bcd)));
end architecture ROM_based;
```