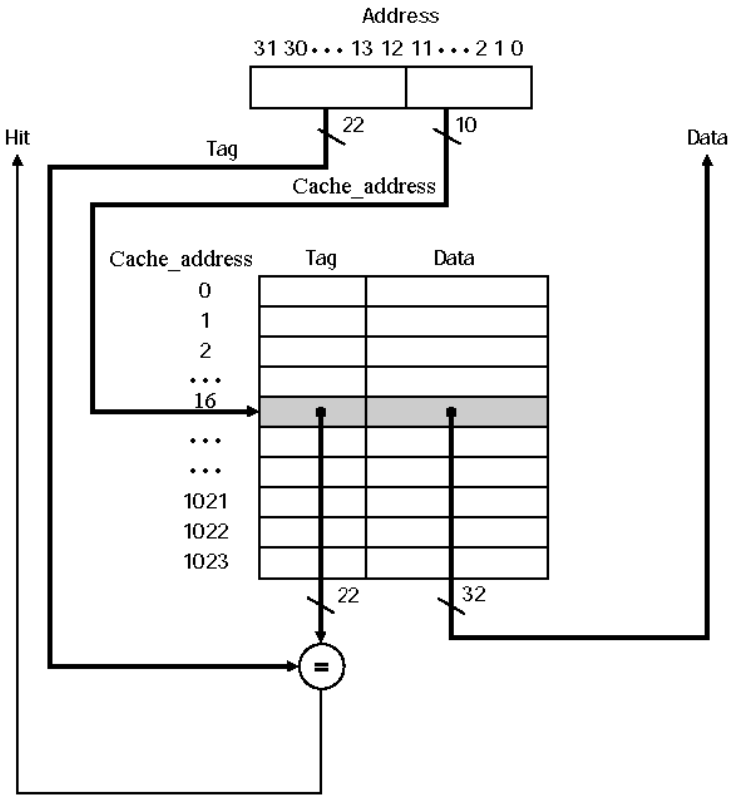
להלן תאור של מערכת בקרת זיכרון מטמון (Cache Controller) של זיכרון הפקודות (Instruction Cache Memory). המעבד (CPU) נעזר ב-Instruction Cache Memory העובד במקביל לזיכרון ראשי (Main Memory). שיטת ניהול ה-Cache היא – direct mapping מתוארת מטה. ה-CPU באמצעות מונה הכתובות (PC- Program Counter) המייצג כתובת (address) והסיגנל memory_read פונה לזכרון בכל מחזור הבאת הפקודה (fetch). הפניה מתבצעת ל-Main Memory וגם ל-Cache במקביל. ביצועי ה-Cache (זמן הנדרש לקבלת מילה רצויה מהזיכרון) טובים יותר מביצועי ה- Main Memory, לכן במידה וה-Cache מכיל את הפקודה המבוקשת, הפקודה נשלפת מה-Cache ולא מה-Main Memory במטרה להגדיל ביצועים. ה- Cache Controller במחזור ה-fetch פועל באופן הבא:

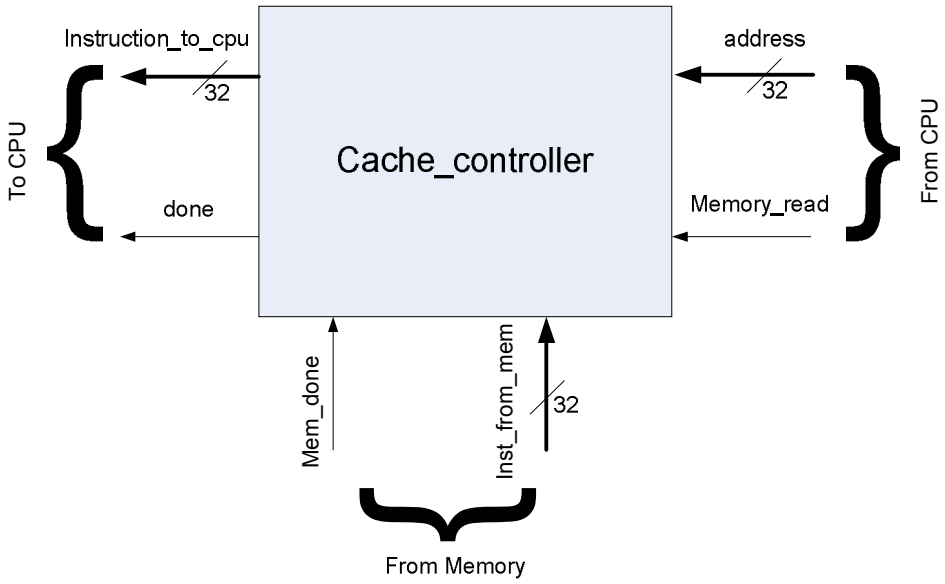
ה-Cache מחובר ל-address bus שגודלו m באמצעות אפיק שגודלו- cache_address_size ביטים נקבע לפי מספר המלים-N אותם מכיל ה-Cache, אשר $cache_address_size = \lceil \log_2 N \rceil$. ה-Cache קטן בצורה משמעותית מה- Main Memory, לכן בכל פניה של ה-CPU ל-Main Memory, כל מילה ב-Cache מזוהה עם מספר כתובות ב-Main Memory השווה ל- $2^{(m-cache_address_size)}$. על מנת להחליט על המצאות של הפקודה המבוקשת ב-Cache, יש להשלים cache_address לכתובת המלאה. ההשלמה- tag נמצאת ב-Cache יחד עם הפקודה המבוקשת. גודל מילת ה-Cache הוא גודל הפקודה וה-tag, או $data_size + (m - cache_address_size)$. בכל fetch של הפקודה מה-cache נשלפת מילה המכילה גם את הפקודה וגם את ה-tag, במידה וה-tag זהה לתת-כתובת העליונה, כלומר להפרש בין הכתובת המלאה לבין cache_address מצב זה נחשב כ-hit, והפקודה (instruction_to_cpu) נקראת מה-cache. במקרה של אי התאמה מדובר במצב miss, ממתינים למילה מ-Main Memory, אינדיקציה לכך ע"י סיגנל mem_done, והפקודה המגיע (inst_from_mem) נכתבת ל-cache לפי cache_address, ובשדה- של ה-tag נכתב הפרש בין הכתובת המלאה לבין ה-cache_address.



איור 1

באיור 1 מוצג מבנה לוגי של ה- Cache המכיל 1024 מילים, בעל cache_adders של 10 ביט וה-tag של 22 ביט המשרת CPU בעל פקודה של 32 ביט. בדוגמה זו ה-CPU פונה ל-Main Memory בכתובת "x"00fa3810. ניתן לחשב את ה- cache_address שהוא "b"00_0001_0000 או 16 עשרוני. במידה ובכתובת ה-16 של ה-Cache בשדה ה-tag נימצא ערך "b"00_0000_0011_1110_1000_1110 יש לנו

hit, אחרת יש miss. בכל מקרה, ללא תלות במקור של הפקודה המגיעה ל- CPU, Cache או MainMemory, ה-CPU מקבל הודעה done ברגע שיש פקודה מוכנה מה-Cache Controller.



איור 2

המשימה:

- ✓ תכנן מערכת ספרתית, סינכרונית המתארת Cache Controller של Instruction Cache Memory ואת כל המכלול של הזכרון ה-Cache
- ✓ גודל ה-Cache הוא 4096 מילים.
- ✓ ה- CPU העובד עם ה-Cache זה הוא בעל address bus של 32 ביט והפקודות שגודלן 32 ביט.
- ✓ לייצוג ה-cache יש להשתמש ב- type המוגדר באופן הבא:

$$\textbf{type}\ cache_mem\ \textbf{is array}(0\ \textbf{to}\ 4095)\ \textbf{of}\ std_logic_vector(51\ \textbf{downto}\ 0)$$

- ✓ ממש את מערכת על פי המוגדר באיור 2, ע"י חלוקה לתת מבנים: יחידת בקרה ויחידות הביצוע.
- ✓ את יחידת הבקרה יש לממש כמכונת מצבים (FSM).
- ✓ כל המבנים יש לכתוב ב-processes **נפרדים בלבד**.
- ✓ לממש את המערכות באופן מלא הכולל הגדרת entity ו-architecture.
- ✓ **יש לכתוב מודל קביל לסינתזה**.

שלבי העבודה מוערכים באופן הבא:

א) חלוקה לתת מבנים וסרטוט של המערכת כולה ברמת תת מבנים -**10 נקודות**.

ב) תיאור דיאגרמאת מצבים של יחידת הבקרה (FSM)-**10 נקודות**

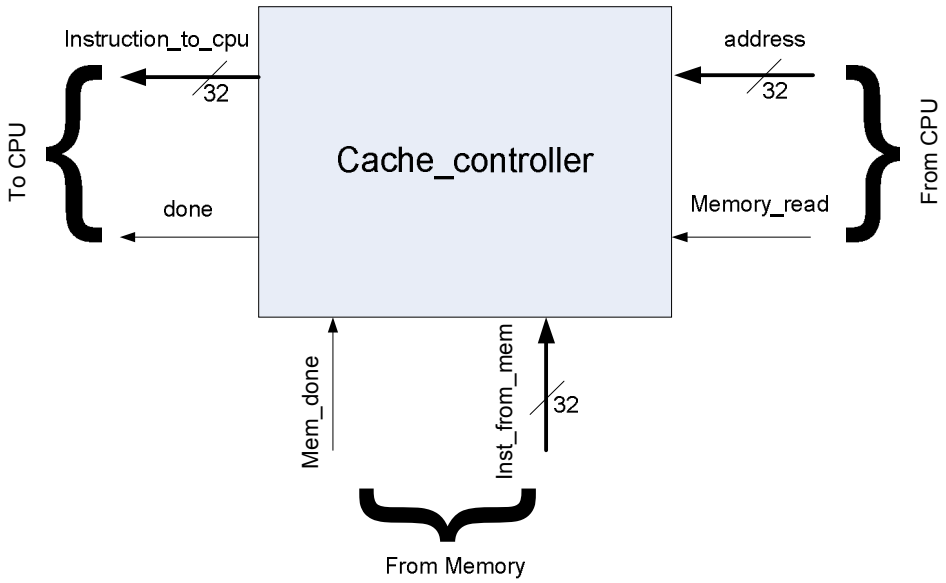
ג) כתיבת קוד הכולל entity, architecture וה-processes המתארים את המערכת-**50 נקודות**

ד) **בונוס של 10 נקודות** בעבור מימוש המערכת הנתונה לאייפיון באמצעות **generic** המגדיר:

1. גודל הפקודה של המעבד
2. גודל ה-Cache כערך שהוא תמיד 2^n, אשר n הוא מספר שלם

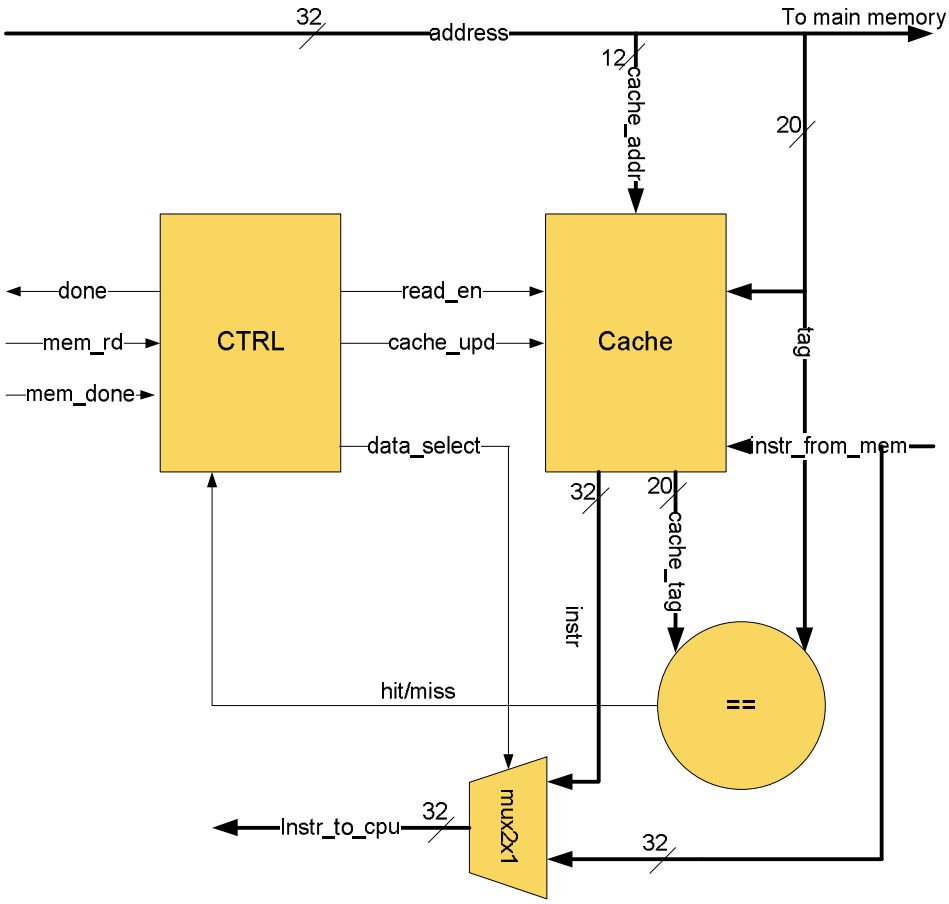
הפתרון.

ראשית נגדיר מסגרת entity.



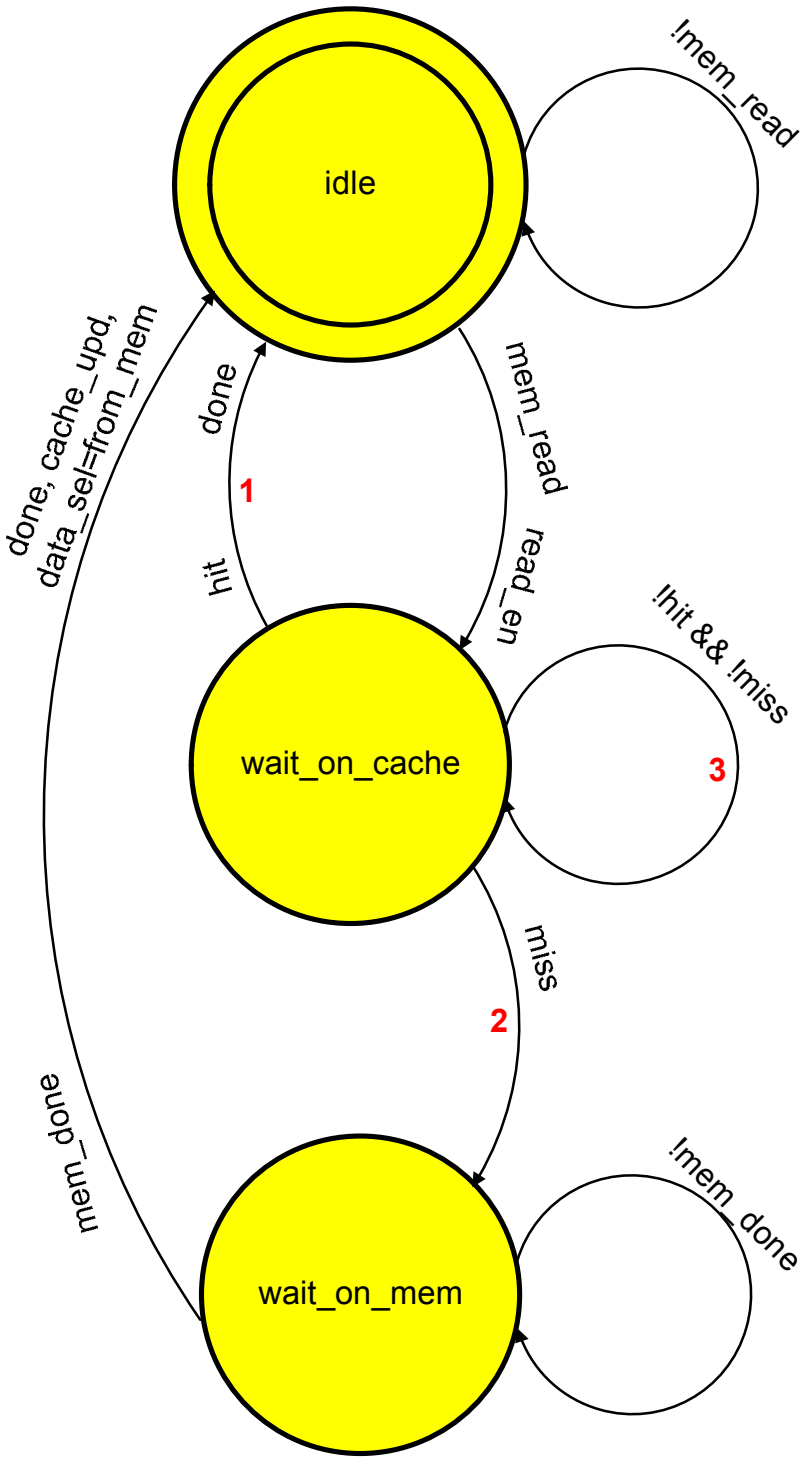
מסגרת זו בנוסף לקווים שצויינו, תכלול גם clock וה-reset.

נציג מבנה של המערכת באמצעות דיאגרמת בלוקים :



נממש יחידת בקרה כמכונת מצבים ונציג דיאגמת מעברים שלה :

Default assignment:
cache_upd='0',
done='0',
data_sel=from_cache



תאור המערכת בשפת VHDL

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity cache_ctrl is
7     generic( instr_width          : integer := 32;
8              addr_width           : integer := 32;
9              log2_of_cache_depth  : integer := 12);
10    port (
11       clk           : in  std_logic;
12       rst           : in  std_logic;
13       addr          : in  std_logic_vector(addr_width-1 downto 0);
14       memory_read   : in  std_logic;
15       inst_from_mem : in  std_logic_vector(instr_width-1 downto 0);
16       mem_done      : in  std_logic;
17       inst_to_cpu   : out std_logic_vector(instr_width-1 downto 0);
18       done          : out std_logic);
19 end entity cache_ctrl;
```

הגדרת ארכיטקטורה ומשאבי המערכת

```
21 architecture arc_cache_ctrl of cache_ctrl is
22    signal hit, miss : std_logic;
23    signal cache_upd : std_logic;
24    signal read_en   : std_logic;
25    signal data_sel  : std_logic;
26    type   mem_arr is array (0 to ((2**log2_of_cache_depth)-1)) of
27       std_logic_vector((instr_width+(addr_width-log2_of_cache_depth)-1) downto 0);
28    signal cache_mem : mem_arr;
29    alias cache_addr : std_logic_vector(log2_of_cache_depth-1 downto 0) is
30                       addr(log2_of_cache_depth-1 downto 0);
31    alias tag        : std_logic_vector((addr_width-log2_of_cache_depth)-1 downto 0) is
32                       addr(addr_width-1 downto log2_of_cache_depth);
33    type   fsm_st is (idle, wait_on_cache, wait_on_mem);
34    signal cs        : fsm_st;
35 begin
```

מימוש מערך הזכרון מטמון ודגלי סטטוס hit/miss

```
37    process (clk, rst) is
38    begin
39       if (rst = '0') then
40          hit       <= '0';
41          miss      <= '0';
42          cache_mem <= (others => (others => '0'));
43       elsif rising_edge(clk) then
44          hit  <= '0';
45          miss <= '0';
46          if (read_en = '1') then
47             if (tag = cache_mem(conv_integer(cache_addr))
48                      ((addr_width+instr_width-log2_of_cache_depth-1) downto instr_width)) then
49                hit <= '1';
50             else
51                miss <= '1';
52             end if;
53          end if;
54          if (cache_upd = '1') then
55             cache_mem(conv_integer(cache_addr)) <= tag & inst_from_mem;
56          end if;
57       end if;
58    end process;
```

ריבוב מבוקר של מידה משני מקורות:
ריבוב מבוקר של מידה משני מקורות:

```
60  inst_to_cpu<=cache_mem(conv_integer(cache_addr))(inst_to_cpu'range) when
61              (data_sel='1') else inst_from_mem;
```

מימוש יחידת בקרה באמצעות מכונת מצבים Mealy בפרוסס אחד ואותת בקרה כרגיסטרים

```
63  process (clk, rst) is
64     variable ns : fsm_st;
65  begin
66     if (rst = '0') then
67        done      <= '0';
68        cache_upd <= '0';
69        read_en   <= '0';
70        data_sel  <= '1';
71        cs        <= idle;
72     elsif rising_edge(clk) then
73        ns        := cs;
74        done      <= '0';
75        cache_upd <= '0';
76        read_en   <= '0';
77        data_sel  <= '1';
78        case cs is
79           when idle => if (memory_read = '1') then
80                           ns      := wait_on_cache;
81                           read_en <= '1';
82                        end if;
83           when wait_on_cache => if (hit = '1') then
84                                    ns   := idle;
85                                    done <= '1';
86                                 elsif (miss = '1') then
87                                    ns := wait_on_mem;
88                                 end if;
89           when wait_on_mem => if (mem_done = '1') then
90                                  ns        := idle;
91                                  done      <= '1';
92                                  cache_upd <= '1';
93                                  data_sel  <= '0';
94                               end if;
95           when others => null;
96        end case;
97        cs<=ns;
98     end if;
99  end process;
```

וסיום:

```
101 end architecture arc_cache_ctrl;
```