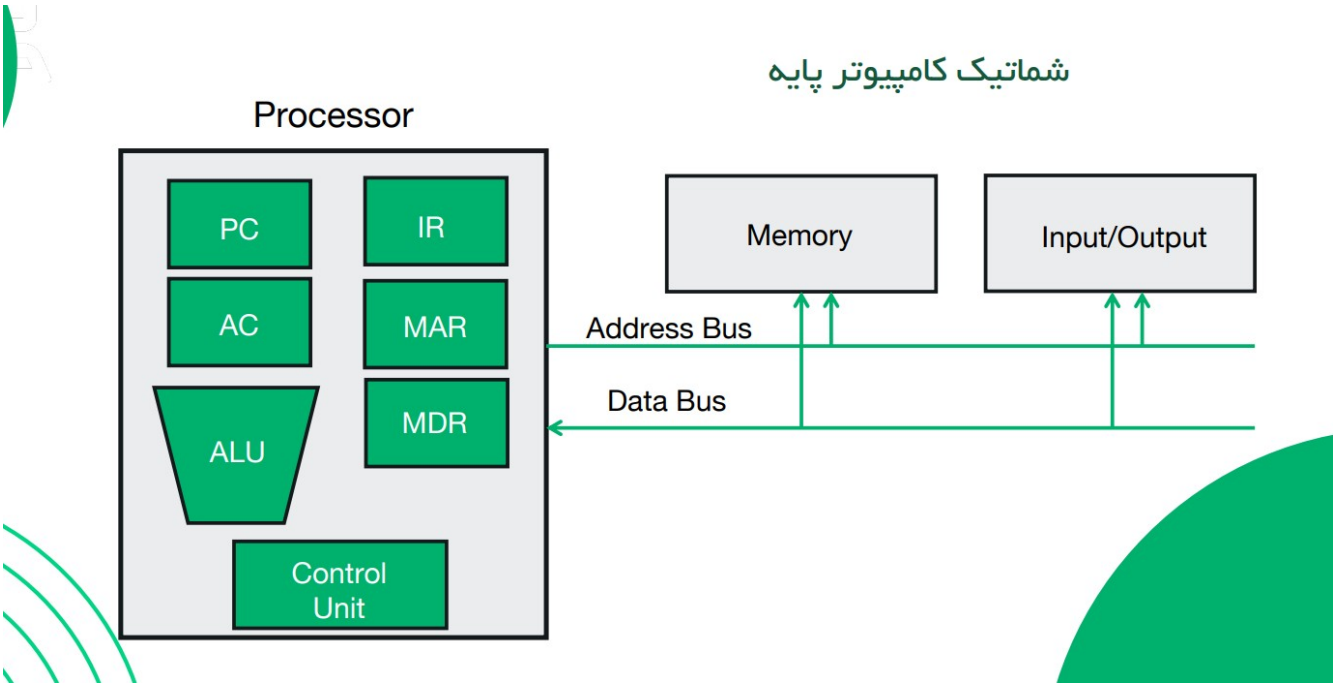
گزارش آزمایش ۴ معماری کامپیوتر

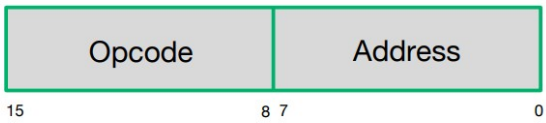
محمد حیدری راد – ۹۹۳۱۰۱۷
مهران غفاریان کلاهی – ۹۹۳۱۰۴۲
محمد علی خسروی — ۹۸۳۱۰۲۲
گروه ۵
استاد عاروان

ساخت کامپیوتر پایه:
کامپیوتر پایه ای که طراحی شده به صورت زیر است:
بنا بر گفته استاد درس، نیازی به پیاده سازی قسمت input و output و Main Memory نبوده و تنها cpu مد نظر بود اما ما برای تست کارکرد کامپیوتر پایه (همینطور از روی علاقه) بخش حافظه را نیز پیاده سازی کرده ایم.



.

ریزعملیات‌های کامپیوتر پایه



فرمت کامپیوتر پایه به صورت ۱۶ بیتی است که شامل

- ۸ بیت opcode که دستورات را مشخص می‌کند.
- ۸ بیت آدرس که آدرس یکی از عملگرها را مشخص می‌کند.
- حافظه با ظرفیت ۲۵۶ کلمه ۱۶ بیتی

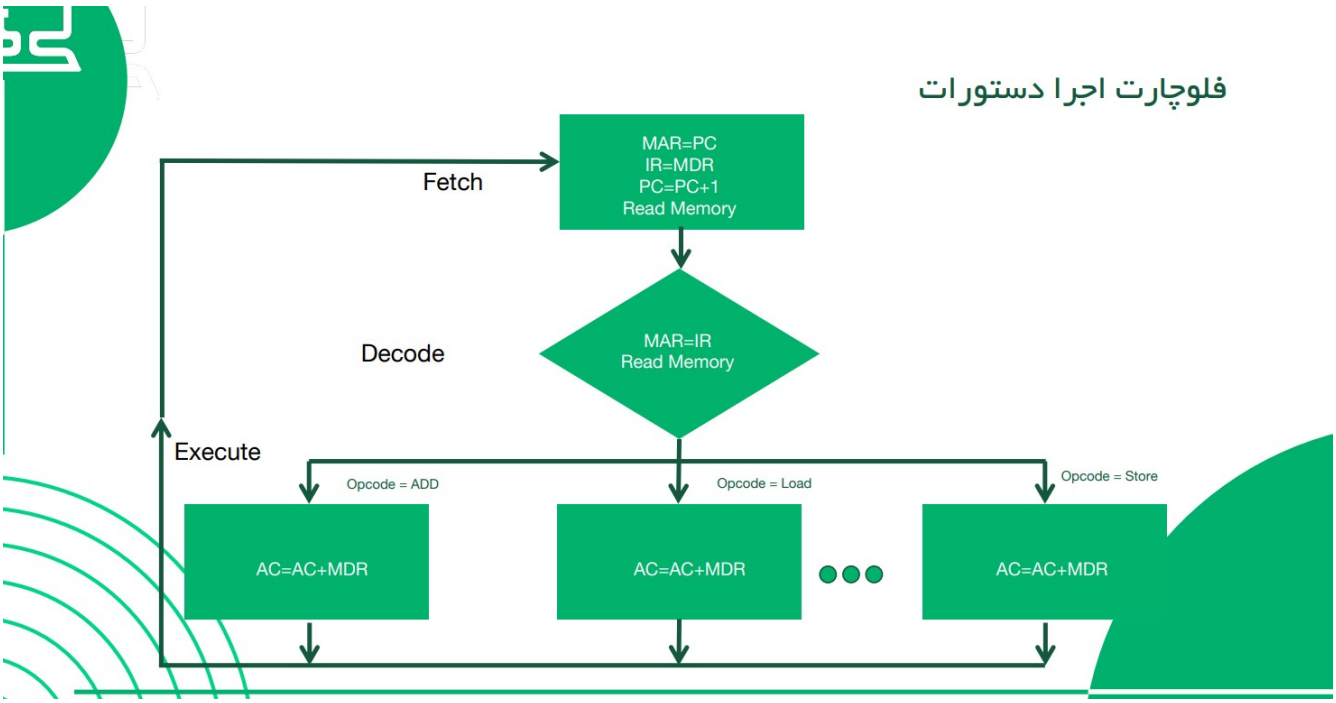
.

ریزعملیات‌های کامپیوتر پایه

• ADD Address AC ⟵ AC + content of Memory Address	00
• Store Address Contents of Memory Address ⟵ AC	01
• Load Address AC ⟵ Content of Memory Address	02
• Jump Address PC ⟵ Address	03
• JNEG Address if AC<0 THEN PC ⟵ address	04

.

باید ذکر کنیم که برای اینکه کامپیوتر بعد از اجرای دستورات نوشته شده در Main Memory دیگر دستوری اجرا نکند، یک دستور halt با کد 05 نیز شخصا اضافه کرده ایم.



.

ALU این کامپیوتر صرفا دستور ADDرا انجام میدهد

کد ALU :

```
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use ieee.numeric_std.all;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  --use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity alu is
34  port (
35  a: in std_logic_vector(15 downto 0);
36  b: in std_logic_vector(15 downto 0);
37  c: out std_logic_vector(15 downto 0)
38  );
39  end alu;
40
41  architecture Behavioral of alu is
42  begin
43  c <= std_logic_vector(signed(a) + signed(b));
44  end Behavioral;
```

.

PROCESSOR

processor نوشته شده از 16 حالت (state) استفاده میکند که بفهمد در چه حالتی است و چه کاری باید انجام شود:

اگر reset برابر 1 باشد، حالت بعدی کامپیوتر start میشود.

start رجیستر ها initialize شده و r و w برابر مقدار 0 میشوند و حالت بعدی برابر fetch_0 قرار میگیرد

: fetch_0

با مقدار دهی r <= 1 فرایند خواندن از حافظه آغاز شده و مقداری که باید خوانده شود که همان مقدار pc است در MAR قرار میگیرد تا حافظه مقدار ذخیره شده در همان آدرس را در MDR بریزد، حالت بعدی fetch_1 قرار میگیرد
fetch_1: در این حالت، حافظه مقدار خواسته شده را با سیگنال data_in به processor میفرستد، پس دیگر کاری با حافظه نداریم و r نیز 0 میشود و حالت بعدی fetch_2 قرار میگیرد
fetch_2: در این حالت مقدار data_in که از حافظه خوانده شده در IR ذخیره میشود، حالت بعدی load_data_0 قرار داده میشود
load_data_0: در این حالت قرار است فرایند خواند مقدار حافظه ای که آدرس ذخیره شده در ۸ بیت کم ارزش IR ذخیره شده خوانده شود و همینطور ۸ بیت پر ارزش IR در رجیستر opcode ذخیره میشود
نکته : این load_data آن دستور load که مقدار خانه حافظه را در AC میریزد نیست! بلکه اطلاعات ذخیره شده در ۸ بیت پایین IR را میخواند و در MDR میریزد.
load_data_1: در این حالت مقدار ذخیره شده در حافظه خوانده شده و از طریق data_in به processor فرستاده میشود، پس r را 0 میکنیم و حالت بعدی را load_data_2 قرار میدهیم.
load_data_2: در این حالت مقدار data_in را در رجیستر MDR ریخته و حالت بعدی را decode قرار میدهیم.
decode : بنا بر opcode، مشخص میشود حالت بعدی چه باید باشد
add: خروجی alu که حاصل جمع AC با MDR را محاسبه میکند در AC ذخیره میشود و حالت بعدی fetch_0 قرار میگیرد
store: سیگنال w برابر 1 میشود و در کلاک بعدی AC در خانه حافظه ای که MAR به آن اشاره میکند ذخیره میشود و حالت بعدی fetch_0 قرار میگیرد
load_0: سیگنال r برابر 1 میشود و در کلاک بعدی از طریق سیگنال data_in مقدار حافظه خواسته شده به processor فرستاده میشود، حالت بعدی load_1 میشود
load_1 : در این حالت داده در data_in قرار گرفته است، پس r برابر 0 میشود و حالت بعدی load_2 میشود
load_2: در این حالت data_in در AC قرار میگیرد و حالت بعدی fetch_0 میشود (درست است که در کامپیوتر مانو، نمیتوان مستقیم از رجیستری به AC داده ای ریخت، اما در این آزمایش این فرض جهت ساده سازی کامپیوتر برداشته شده)
jump: مقدار MAR در PC ریخته میشود و حالت بعدی fetch_0 قرار میگیرد.
Jneg: اگر بیت 16 ام AC برابر 1 بود پس عدد منفی است و MAR در PC قرار میگیرد
halt: این حالت تله است که کامپیوتر هیچ کاری را جلو نمیبرد.
سیگنال هایی که با see شروع میشوند صرفا جهت دیدن کارکرد کامپیوتر از بیرون هستند و فایده دیگری ندارند.

MAIN_MEMORY

حافظه نوشته شده صرفا یک آرایه از رجیستر هاست (منطقی نیست ولی فعلا برای تست کامپیوتر کارمان را راه می اندازد) همینطور یک برنامه تست کامپیوتر در آن با توضیحات در کامنت ارائه شده.

```
45
46  architecture Behavioral of memory is
47  type reg_ram is array(0 to 255) of std_logic_vector(15 downto 0);
48  signal ram: reg_ram := (
49  "0000001100100000", --0 inst: jump to 32
50  "0000000000001110", --1 inst: add with data in address 14
51  "0000000100010000", --2 inst: store in address 16
52  "0000000000010010", --3 inst: add with data in address 18
53  "0000001100001001", --4 inst: jump to 9
54  "0000000000000011", --5 inst: add with data in address 3
55  "0000010100000000", --6 inst: halt
56  "0000001000001111", --7 inst: load data in address 15
57  "0000001100000001", --8 inst: jump to 1
58  "0000001000001110", --9 inst: load data in address 14
59  "0000000000010010", --10 inst: add with data in address 18
60  "0000001000010000", --11 inst: load data in address 16
61  "0000000000010010", --12 inst: add with data in address 18
62  "0000001100010011", --13 inst: jump to 19
63  "0000000000000011", --14 data: 3
64  "0000000000001100", --15 data: 12
65  "0000000000000000", --16 data: 0 (0000000000001111 will be stored in it)
66  "0000000000000000", --17 data: 0
67  "1000000000000000", --18 data: -2**8
68  "0000010000010101", --19 inst: jneg address 21
69  "0000001100000000", --20 inst: jump to 0 (if 19 does not jump this, computer never halts :) )
70  "0000010100000000", --21 inst: halt
71  "0000000000000000", --22
72  "0000000000000000", --23
73  "0000000000000000", --24
74  "0000000000000000", --25
75  "0000000000000000", --26
76  "0000000000000000", --27
77  "0000000000000000", --28
78  "0000000000000000", --29
79  "0000000000000000", --30
80  "0000000000000000", --31
81  "0000000000010010", --32 add with data in address 18
82  "0000010000000001", --33 jneg to address 1
83  "0000010100000000", --34 halt
84  "0000000000000000",
85  "0000000000000000",
86  "0000000000000000",
87  "0000000000000000",
88  "0000000000000000",
89  "0000000000000000",
```

.

ماژول computer صرفا دو ماژول processor و main_memoryرا به یکدیگر متصل میکند.