توضیحات آزمایش ۹: کامپیوتر پایه (۱)

هدف از این آزمایش طراحی یک کامپیوتر پایه ساده و مشاهده عملی چگونگی عملکرد آن است. ورودی و خروجی‌های کامپیوتر پایه در شکل زیر نشان داده شده است.

برای پیاده‌سازی کامپیوتر پایه از دو روش structural و behaviorl می‌توان استفاده کرد. در روش structural اجزای داخلی پردازنده را به صورت مجزا طراحی و پیاده‌سازی شده و در نهایت آنها کنار هم قرار می‌گیرند. در روش behavioral ، پردازنده توسط توصیف رفتار آن طراحی می‌شود. در این آزمایش، کامپیوتر پایه به صورت رفتاری توصیف می‌شود.

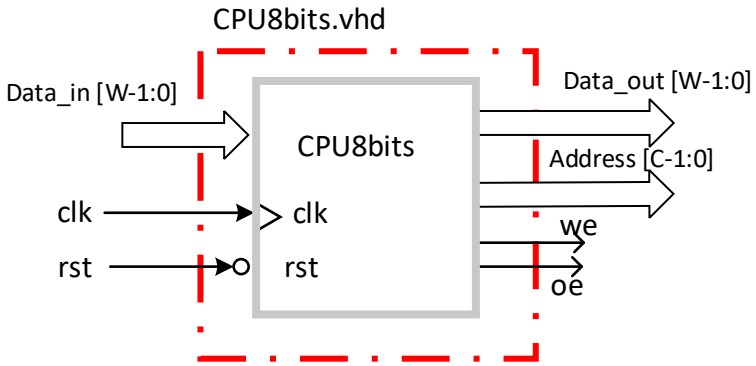
کامپیوتر پایه طراحی شده از بخش حافظه و پردازنده تشکیل شده است.

واحد حافظه

حافظه کدهای برنامه و داده اشتراکی است و از نوع RAM است. حافظه RAM دارای خطوط آدرس ۵ بیتی، خطوط داده ورودی و خروجی هر کدام ۸ بیت و خطوط کنترلیw، r و clk است. برای طراحی حافظه از RAM طراحی شده در آزمایش‌های قبل استفاده کنید. با این تفاوت که زمانی که حافظه rst می‌شود، با دستورات مختلف مشخص شده در جدول ۱ نوشته شده است، reset شود.

واحد پردازنده:

ورودی و خروجی واحد پردازنده در شکل زیر نشان داده شده است.



شکل ۱: پردازنده ۸ بیتی و پورت‌های ورودی و خروجی

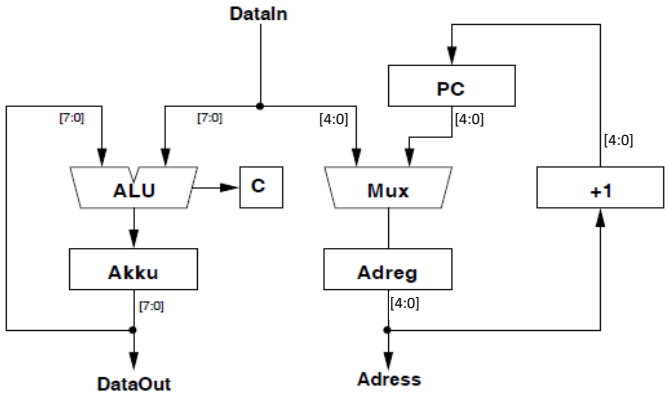
این پردازنده ۸ بیتی است و توانایی آدرس‌دهی ۳۲ بایت را دارد. تعداد خطوط آدرس ۵ بیت و پهنای گذرگاه داده ۸ بیت است.

خطوط داده، آدرس we و oe برای ارتباط با حافظه هستند. زمانی که پردازنده بخواهد داده‌ای را از حافظه بخواند، خطوط آدرس را در Adress می‌ریزد و سیگنال oe را فعال می‌کند. زمانی که بخواهد داده را در حافظه بنویسد، آدرسی را در Address، داده را در Data می‌ریزد و سیگنال we را فعال می‌کند.

- کامپیوتر پایه بر مبنای Accumulator هست .که دارای عرض بیت ۸ بیتی و یک بیت carry است (Akku (8 downto (0). بیت آخر همان بیت Carry است. Akku(8) = carry. : Akkuرجیستری که می‌تواند داده را از باس داخلی بگیرد یا بر آن قرار دهد.
- شمارنده برنامه : PC رجیستری است که آدرس دستور بعدی در آن قرار می‌گیرد. شمارنده برنامه (PC) که پنج بیتی است که می‌تواند ۳۲ خانه حافظه ۸ بیتی را آدرس‌دهی کند.
- سیگنال adreg که آدرس دستور و یا داده ای که باید ذخیره و یا از حافظه خوانده شود را نگه می‌دارد.
- زمانی که سیگنالrst ’0’ باشد، تمام متغیرهای AKKU، PC و Adreg برابر با صفر می‌شود، و ماشین حالت به state "0000" می‌رود.
- واحد کنترل (CU): بخش‌های گوناگون پردازنده را با هم هماهنگ می‌کند.
- ALU : محاسبات عددی و منطقی در این بخش صورت می‌گیرد.

مسیر داده

مسیر داده این پردازنده دز شکل زیر نشان داده شده است. (pc و adreg هرکدام ۵ بیتی هستند.)



شکل ۲) مسیر داده پردازنده

مجموعه دستورها

هر دستور در این پردازنده یک بایت طول دارد و قالب آن تک دستوره است و شامل دو بخش زیر است: بخش opcode ۳ بیت پرارزش دستور است که نوع دستور را مشخص می‌کند و ۵ بیت کم ارزش آن آدرس و یا داده که با توجه به نوع دستور می‌تواند آدرس و یا داده باشد.

7 5	4 0
opcode	Address\ Immediate

لیست دستورات که توسط پردازنده باید اجرا شود با توجه به بیت‌های Opcode در جدول زیر آورده شده است.

جدول ۱: لیست دستورات قابل اجرا توسط پردازنده

opcode	instruction	operation
000	LD A, [ADDR]	ACC ← [ADDR]
001	ST A, [ADDR]	[ADDR] ← ACC
010	JP ADDR	JUMP TO ADDR
011	JPC ADDR	IF Carry == 1: JUMP TO ADDR
100	AND A, [ADDR]	ACC ← ACC & [ADDR]
101	ADD A, [ADDR]	ACC ← ACC + [ADDR]
110	NOT	ACC ← not ACC
111	SHL	ACC ← ACC << 1

به عنوان مثال، ۸ بیتی "۱۰۱۰۰۱۱۰" دستور ADD می‌باشد که مقدار Akku را با داده‌ای که در آدرس "00110" قرار دارد جمع می‌کند و مقدار carry را در Akku(8) می‌ریزد.

واحد کنترل:

مسیر داده زا می‌توان با استفاده از یک ماشین حالت کنترل کرد. state های ماشین حالت نوع دستور و فراخوانی دستور را مشخص می‌کنند. به ازای هر دستور یک state و برای به روزرسانی PC نیز یک state نیاز است. در این آزمایش تعداد دستورات ۸ دستور است که ماشین حالت نیاز به ۸ حالت به علاوه یک حالت ابتدایی برای به روزرسانی PC و فراخوانی دستور است که در مجموع ماشین حالت ۹ حالت متفاوت خواهد داشت، و برای نمایش آن نیاز به ۴ بیت خواهد بود. حالت "0000" state فراخوانی دستور و بروزرسانی داده را انجام می‌دهد. ابتدا از این "0000" state شروع به کار می‌کند و بعد از آن با توجه مقدار opcode به state متناظر با اجرای آن دستور می‌رود. بعد از اجرای هر دستور، دوباره به "0000" state باز می‌گردد تا مقدار PC به روزرسانی شود و دستور بعد اجرا شود. کد شده حالت‌های ماشین حالت در جدول زیر نشان داده شده است:

جدول ۲: کد شده حالت‌های مختلف ماشین حالت

state		instruction	operation	Next state
S0	0000	Fetch instruction/ Operand address	pc ← adreg + 1, adreg = data oe ←0 , data ← Z	S0 if opcode = 011 , c=0 S1 if opcode = 000 … S4 if opcode = 011 , c=1 … S8 if opcode = 111
S1	1000	Load memory to akku	oe←1 , adreg ←pc akku←data	S0
S2	1001	Write akku to memory	we←1, data ←akku adreg← pc	S0
S3	1010	Read PC	adreg←PC	S0
S4	1011	Clear carry, read PC	Carry ←0 , adreg←PC	S0
S5	0100	Read operand, AND	oe←1, data ←Z , adreg ← PC akku←akku and data , update carry	S0
S6	0101	Read operand, ADD	oe←1, data ←Z , adreg ← PC akku←akku and data	S0
S7	0110	Not Akku	akku←not akku	S0
S8	0111	Shift left akku	akku←akku (7 downto 0) & '0'	S0

طراحی پردازنده با زبان توصف سخت افزار

نمونه‌ای از پردازنده ای که ۴ عملیات ADD،NOR ،JCC وSTA را انجام می‌دهد در دستور کار آورده شده است (خطوط آدرس در این طراحی ۵ بیتی در نظر گرفته شده است و opcode ۲ بیتی است.). با تغییر در کدهای آن، پردازنده‌ای را طراحی کنید که عملیات‌های خواسته شده را انجام دهد. (خطوط داده ۸ بیتی، خطوط آدرس ۵ بیتی و ۳ بیت برای opcode در نظر بگیرید)

ابتدا کتابخانه های مورد نیاز فراخوانی می‌شوند، Entity با نام CPU8BITS تعریف شده و پورت‌های ورودی و خروجی را مشخص می‌کند. در این ENTITY سیگنال data از نوع Inout تعریف شده است. که برای خواندن و نوشتن داده در حافظه کاربرد دارد. بهتر است که سیگنال‌های data_in و data_out به صورت جداگانه و از نوع in و out تعریف شوند.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity CPU8BIT2 is
port (
        data: inout std_logic_vector(7 downto 0);
        adress: out std_logic_vector(5 downto 0);
        oe: out std_logic;
        we: out std_logic; -- Asynchronous memory interface
        rst: in std_logic;
        clk: in std_logic);
end;

architecture CPU_ARCH of CPU8BIT2 is
        signal akku: std_logic_vector(8 downto 0); -- akku(8) is carry !
        signal adreg: std_logic_vector(5 downto 0);
        signal pc: std_logic_vector(5 downto 0);
        signal states: std_logic_vector(2 downto 0);
```

سپس در Architecture رفتار پردازنده توصیف می‌شود.

در ابتدا سیگنال‌های Akku،adreg ، status , PC تعریف می‌شود.

در بدنه Architecture، ابتدا سیگنال rst چک می‌شود تا در صورتی که مقدار آن صفر باشد، همه سیگنال‌های میانی (رجیسترها) را صفر کند. سیگنال rst ناهمگام با پالس ساعت و Low active است.

```
architecture CPU_ARCH of CPU8BIT2 is
        signal akku: std_logic_vector(8 downto 0); -- akku(8) is carry !
        signal adreg: std_logic_vector(5 downto 0);
        signal pc: std_logic_vector(5 downto 0);
        signal states: std_logic_vector(2 downto 0);

begin
Process(clk,rst)
begin
        If (rst = '0') then
        adreg <= (others => '0'); -- start execution at memory location 0
        states <= "000";
        akku <= (others => '0');
        pc <= (others => '0');
```

در لبه بالارونده پالس ساعت، پردازنده شروع به کار می‌کند. "000" state برای فراخوانی دستور و به روزرسانی PC است، اگر در حال حاضر در "000" state باشد، باید یک واحد افزایش پیدا کند، (آدرس را یک واحد افزایش دهد) و بیت‌های آدرس داده ورودی را در adreg قرار دهد. در بقیه state ها و بعد از انجام عملیات‌ها، آدرس PC که آدرس بعد را مشخص می‌کند در Adreg قرار می‌گیرد و دستور بعدی برای اجرا فراخوانی می‌شود.

دستورات در دو پالس اجرا می‌شوند، در پالس اول، بر اساس مقدار state opcode مشخص می‌شود، و در پالس دوم، عملیات مربوط به آن پالس اجرا خواهد شد. پس در پالس اول، مقدار PC یک واحد اضافه می‌شود، و در پالس دوم، اگر در "state"0000، نباشد، مقدار PC در adreg قرارمی‌گیرد.

```
elsIf rising_edge(clk) then

-- PC / Adress path
If (states = "000") then
        pc <= adreg + 1;
        adreg <= data(5 downto 0);
else
        adreg <= pc;
end If;
```

در ALU/data Path ، بر حسب state که در حال حاضر در آن قرار دارد، عملیات ALU متناسب با آن state را انجام می‌دهد. به طور مثال، زمانی که در "state "010 است، عملیات جمع akku با داده ورودی انجام می‌شود.

```
-- ALU / Data Path
Case states is
        when "010" => akku <= ("0" & akku(7 downto 0)) + ("0" & data); -- ADD:  ACC ← ACC + [ADDR]
        when "011" => akku(7 downto 0) <= akku(7 downto 0) nor data; -- NOR :  ACC ← ACC + [ADDR]
        when "101" => akku(8) <= '0'; -- branch not taken, clear carry
        when others => null; -- instr. fetch, jcc taken (000), sta (001)
end Case;
```

در مرحله بعد state machine پردازنده تعریف شده است. ماشین حالت طراحی شده مدل Moore است و با توجه به توضیحات جدول ، در حر حالت با توجه به حالت های فعلی و بیت هایopcode و یا carry، حالت بعد مشخص می‌شود. به طور مثال زمانی در هر state به جز state "000" باشد، حالت بعدی "000" خواهد بود تا PCو Adreg به روز رسانی شوند. و دستور بعد فراخوانی شود. اگر مقدار opcode "۱۱" باشد وcarry برابر با "1" باشد، دستور پرش شرطی باشد انجام شود و به state مربوطه که "101" هست میرود. در آن state باید مقدار carry برابر با صفر شود.

```
-- State machine
        If (states /= "000") then
                states <= "000"; -- fetch next opcode
        elsIf (data(7 downto 6) = "11" and akku(8)='1') then
                states <= "101"; -- branch not taken
        else states <= "0" & not data(7 downto 6); -- execute instruction
        end If;
end If;  -- end of rst

end Process;
```

در نهایت باید خروجی‌های ماشین حالت Moore مشخص شوند. در هر حالت باید مشخص شود که مقدار سیگنال‌هایdata خروجی، Adress، oe و we چه مقداری باشد. در طراحی مثال ذکر شده در دستور کار، oe و we به صورت low active هستند. اما طراحی این آزمایش و با توجه به آنکه حافظه RAM طراحی شده در آزمایش‌های قبل خطوط خواندن و نوشتن high active بودند، oe و we را به صورت high active طراحی کنید.

```
-- output
adress <= adreg;
data <= "ZZZZZZZZ" when states /= "001" else akku(7 downto 0);
oe <= '1' when (clk='1' or states = "001" or rst='0' or states = "101") else '0';
-- no memory access during reset and state "101" (branch not taken)
we <= '1' when (clk='1' or states /= "001" or rst='0') else '0';

end CPU_ARCH;
```

با توجه به ماشین حالت طراحی شده، این ماشین هر دستور را در دو پالس ساعت انجام خواهد داد.