دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارشکار آزمایشگاه معماری کامپیوتر – شماره ۱۱

عنوان آزمایش: پیاده‌سازی یک نسخه ساده‌سازی شده از کامپیوتر پایه مانو

نام و نام خانوادگی گردآورندگان : حسنا اویارحسینی و پویا محمدی

استاد آزمایشگاه: جناب آقای مهندس عاروان

تاریخ آزمایش: ۱۴۰۰/۱۰/۶

آزمایش۱)

نام آزمایش: پیاده‌سازی یک نسخه ساده‌سازی شده از کامپیوتر پایه مانو

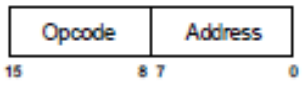
شرح آزمایش:

در این آزمایش میخواهیم یک نسخه ساده از کامپیوتر پایه را که عملیات های پایه ای را انجام میدهد بررسی و پیاده سازی کنیم:

بخش اول) توضیح opcode ها:

ابتدا قالب دستور را برای این کامپیوتر مشخص میکنیم:

در اینجا حافظه اصلی 256word * 16bits/word در نظر گرفته شده است پس برای مشخص کردن آدرس هر word در حافظه اصلی نیاز به ۸ بیت داریم، همچنین میدانیم قالب دستور باید مضربی از اندازه یک word (در اینجا یعنی ۱۶ بیت باشد) با توجه به اینکه در پیاده سازیمان ۵ دستور داریم و هر کدام یک عملوند دارند با ۱۶ بیت یا یک word میتوان قالب را پیاده سازی کرد. قالب دستور را طوری طراحی میکنیم که ۸ بیت پر ارزش آن مربوط به opcode(برای تشخیص دستور) و ۸ بیت کم ارزش تر مربوط به آدرس عملوند می باشد:



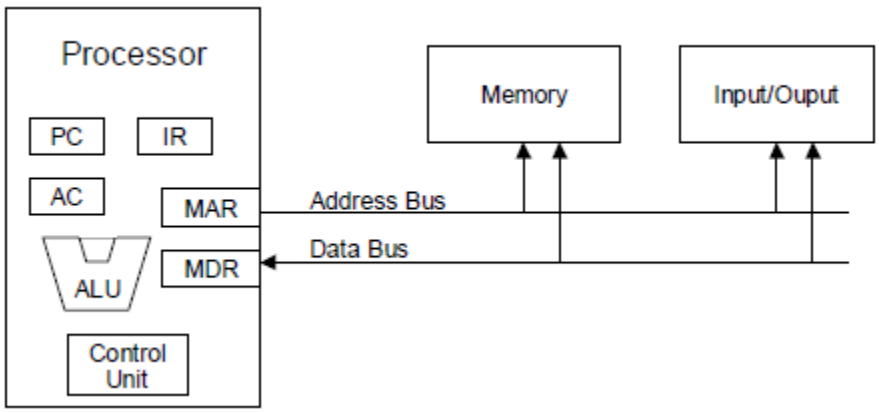
سپس opcode ها را به صورت زیر به هر یک از دستور ها منتصب میکنیم:

دستور	توضیح دستور	Opcode(Hex)
ADD address	AC <- AC + address محتوای حافظه اصلی در	00
STORE address	address محتوای حافظه اصلی در <- AC	01
LOAD address	AC <- address محتوای حافظه اصلی در	02
JUMP address	PC <-address	03
JNEG address	IF AC<0 THEN PC <- address	04

پیش نیاز بخش دوم)

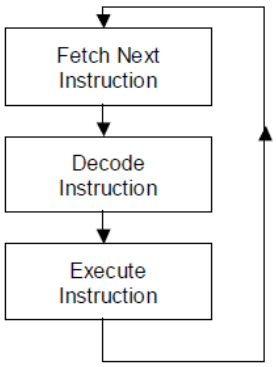
در جدول زیر ثبات هایی که در این ماشین وجود دارند ذکر شده اند تا در ادامه به کمک آنها به توضیح مراحل fetch, decode, Execute بپردازیم:

نام ثبات	متغیر در نظر گرفته شده در کد	توضیحات(وظیفه ثبات)
PC	program_counter	ثباتی که به دستور فعلی در حافظه اصلی اشاره میکند
IR	instruction_registe	ثباتی که دستور فعلی در آن قرار میگیرد
MDR	memory_data_register	ثباتی که داده هایی که از حافظه اصلی خوانده میشوند در آن قرار میگیرند
AC	register_ac	ثبات accumulator برای محاسبات
MAR	memory_address_register	تنها ثباتی که میتواند به حافظه اصلی تقاضای دریافت داده موجود در یک آدرس مشخص را بدهد



بخش دوم) توضیح چگونگی اجرای چرخه fetch, decode, Execute :

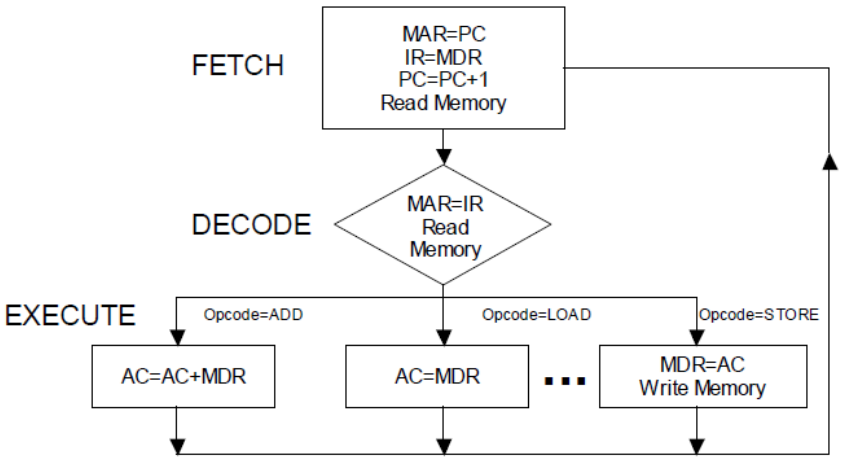
سه گام اصلی در کامپیوتر پایه fetch, decode, Execute می باشد. دستور ها(در قالبی که در بخش قبل ذکر کردیم) در حافظه اصلی وجود دارد ابتدا در مرحله fetch باید دستور ها را از حافظه اصلی بخوانیم و وارد ثبات IR بکنیم سپس دستور را دیکود کنیم، یعنی بخش opcode و عملوند را با توجه به قالب دستور جدا کنیم و با توجه به opcode تصمیم بگیریم در مرحله execute چه کاری باید انجام دهیم و در نهایت در مرحله execute آن عملیات را انجام دهیم، به این صورت یک خط از دستور خوانده و اجرا میشود سپس به خط بعد رفته و این چرخه را دوباره برای دستور جدید انجام میدهیم.



برای پیاده سازی این چرخه در کد، از یک ماشین حالت استفاده کرده ایم این ماشین حالت شامل state های زیر میشود:

```
reset_pc, fetch, decode, execute_add, execute_load,
execute_jneg,execute_jneg2, execute_store,execute_store2, execute_jump
```

و به این صورت کار میکند که در هر حالت ریزعملیات های مربوط به آن حالت انجام میشود و state بعدی هم مشخص میشود در ادامه به توضیح جزیی تر پیاده سازی دو حالت fetch و Decode میپردازیم:



Fetch:

همان طور که در فلوچارت مشاده میشود برای fetch کردن اول، آدرس دستور را که در PC ذخیره شده است به AR انتقال میدهیم (زیرا فقط AR توانایی درخواست از حافظه اصلی را دارد) و سپس دستور را از حافظه اصلی به IR منتقل میکنیم، در این مرحله PC را نیز یکی افزایش میدهیم تا برای چرخه بعد به دستور بعدی دسترسی داشته باشیم.

در کد بخش منتقل کردن PC به MAR را در خط های آخر(که مربوط به مقدار دهی MAR می باشد) انجام داده ایم. به این صورت که پس از اجرای عملیات هر state انجام میشود.(دستورات مربوط به مقدار دهی MAR به رنگ نارنجی در ابتدای جدول هر حالت آورده شده این دستورات پس از دستتورات مربوط به ماشین حالت انجام میشوند).

و همچنین با هر کلاک به کمک دستور زیر آدرس موجود در MAR از حافظه اصلی خوانده شده و محتوای آن به MDR منتقل میشود:

```
memory_data_register <= ram_block(to_integer(memory_address_register));
```

و بقیه مراحل در دستورات مربوط به حالت fetch در ماشین حالت انجام میشود:

FETCH		
`memory_address_register <= program_counter`		when fetch, آدرس دستور بعدی که در PC قرار دارد وارد MAR میشود.
Micro operation	**Code**	**توضیحات**
IR <- MDR	`instruction_register <= memory_data_register;`	ابتدا دستور را که در حافظه اصلی وجود دارد میخوانیم و به ثبات IR که نگهدارنده دستور در CPU است میدهیم
AC <- 0	`program_counter <= program_counter + 1;`	سپس PC را یکی زیاد میکنیم تا برای رفتن به دستور بعد در چرخه بعدی آماده باشد
	`state <= decode;`	به مرحله بعد یعنی decode میرویم

Decode:

DECODE		
memory_address_register <= instruction_register(address_width-1 downto 0) when decode, آدرس عملوند که در IR قرار دارد وارد MAR میشود تا از حافظه اصلی محتوای این خانه درخواست شود.		
Micro operation	**Code**	**توضیحات**

DECODE(IR[15:8])	case instruction_register(15 downto 8) is
	
	when "00000000" =>
	
	state <= execute_add;
	
	when "00000001" =>
	
	state <= execute_store;
	
	when "00000010" =>
	
	state <= execute_load;
	
	when "00000011" =>
	
	state <= execute_jump;
	
	when "00000100" =>
	
	state <= execute_jneg;
		در بخش Decode میخواهیم تشخیص دهیم دستور مربوط به چه opcode است برای این کار با توجه به اینکه در قالب دستور مشخص کردیم ۸ بیت پرارزش برای opcode است به بیت ۸ تا ۱۵ نگاه میکنیم و تصمیم میگیریم که باید به چه حالتی برویم و ریز عملیات های مربوط به کدام دستور را در ادامه انجام دهیم برای این کار به کمک دستور case به بیت های مربوط به opcode نگاه میکنیم و تصمیم میگیریم مرحله بعد کدام مرحله است

	when others => state <= fetch; end case;	اگر بیت های مربوط به opcode در ISA مشخص شده نبود(opcode بزرگتر از ۴) یعنی دستوری به ازای این opcode مشخص نشده پس دوباره به مرحله fetch بر میگردیم

Execute:

و در بخش Execute با توجه به اینکه opcode چه بوده است به یکی از state های زیر منتقل شده ایم و حال دستور مربوط به آن حالت را باید انجام دهیم که در ادامه توضیح داده شده اند:

ADD

`memory_address_register <= program_counter when execute_add,`

آدرس دستور بعدی که در PC قرار دارد وارد MAR میشود.

Micro operation	Code	توضیحات
AC <- AC + MDR	`register_ac <= register_ac + signed(memory_data_register);`	برای جمع یک داده که از حافظه اصلی خوانده ایم با AC کافیست داده را که در ثبات MDR قرار دارد با AC جمع کنیم و حاصل را داخل AC قرار دهیم
	`state <= fetch;`	به مرحله بعد یعنی fetch میرویم تا چرخه برای دستور بعدی دوباره تکرار شود

LOAD

`memory_address_register <= program_counter when execute_load,`

آدرس دستور بعدی که در PC قرار دارد وارد MAR میشود.

Micro operation	Code	توضیحات
AC <- MDR	`register_ac <= signed(memory_data_register);`	داده موجود در حافظه اصلی که قبلا خوانده شده و در ثبات MDR قرار گرفته را در داخل AC قرار میدهیم
	`state <= fetch;`	به مرحله بعد یعنی fetch میرویم تا چرخه برای دستور بعدی دوباره تکرار شود

JUMP

`memory_address_register <=instruction_register(address_width-1 downto 0) when execute_jump,`

آدرس دستور بعدی وارد MAR میشود که این آدرس همان عملوند دستور jump است

Micro operation	Code	توضیحات
PC <- IR[7:0]	`program_counter <= instruction_register(address_width-1 downto 0);`	آدرسی که در دستور به عنوان عملوند مشخص کرده ایم را از IR که دستور را در خود دارد استخراج میکنیم(۸ بیت کم ارزش) و به PC میدهیم به این صورت PC در مرحله بعد به خط بعدی نمیرود و مستقیم به جایی میرود که در دستور مشخص کرده بودیم
	`state <= fetch;`	به مرحله بعد یعنی fetch میرویم تا چرخه برای دستور بعدی دوباره تکرار شود

JENG		
Micro operation	**Code**	**توضیحات**
execute_jneg		
memory_address_register <= program_counter when execute_jneg آدرس دستور بعدی که در PC قرار دارد وارد MAR میشود.		

PC <- IR[7:0]	if (register_ac < 0) then
	
	program_counter <= instruction_register(address_width-1 downto 0);
	
	end if;
		آدرسی که در دستور به عنوان عملوند مشخص کرده ایم را از IR که دستور را در خود دارد استخراج میکنیم(۸ بیت کم ارزش) و به PC میدهیم به این صورت PC در مرحله بعد به خط بعدی نمیرود و مستقیم به جایی میرود که در دستور مشخص کرده بودیم

	state <= execute_jneg2;	این دستور در دو کلاک انجام میشود پس به مرحله ای که باید در کلاک دوم برای این دستور اجرا شود میرویم
execute_jneg_2		
memory_address_register <= program_counter when execute_jneg_2 آدرس دستور بعدی که در PC قرار دارد وارد MAR میشود.		
	state <= fetch	به مرحله بعد یعنی fetch میرویم تا چرخه برای دستور بعدی دوباره تکرار شود

STORE		
Micro operation	**Code**	**توضیحات**
execute_store		
memory_address_register <= instruction_register(address_width-1 downto 0) when execute_store آدرس جایی که باید AC در آن ذخیره شود را برای کلاک بعد در MAR قرار میدهیم.		
	state <= execute_store2;	این دستور در دو کلاک انجام میشود پس حالت را عوض میکنیم و به مرحله ای که باید در کلاک دوم برای این دستور اجرا شود میرویم*
execute_store_2		
memory_address_register <= program_counter when execute_store2, آدرس دستور بعدی که در PC قرار دارد وارد MAR میشود.		
	state <= fetch	به مرحله بعد یعنی fetch میرویم تا چرخه برای دستور بعدی دوباره تکرار شود

*پس از اینکه در حال execute_store2 قرار میگیریم شرط کد زیر برقرار میشود و memory_write یک میشود

```
with state select
            memory_write <= '1' when execute_store,
                            '0' when others;
```

پس در کلاک شرط زیر برقرار شده و داده موجود در AC در حافظه اصلی نوشته میشود.

```
if (memory_write = '1') then
      ram_block(to_integer(memory_address_register)) <=unsigned(register_ac);
end if;
```

توضیح حالت resetکه جزو opcode ها نیست ولی از عملکرد های واحد کنترل هست:

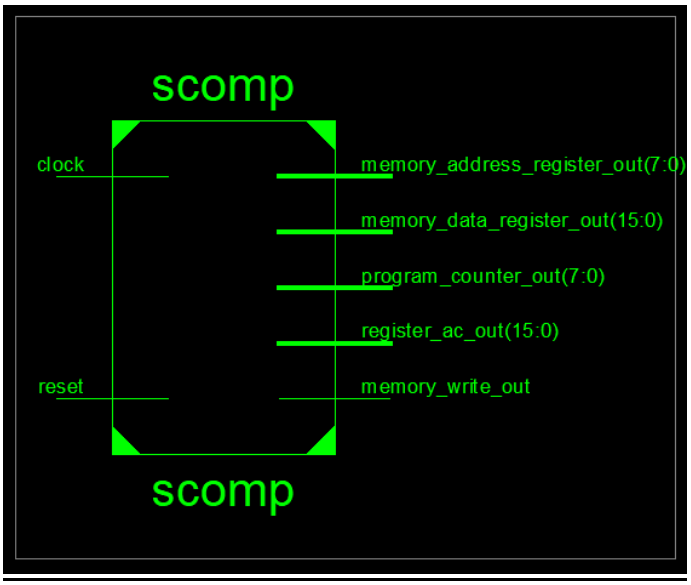
از این حالت برای ریست کردن کامپیوتر در ابتدا استفاده میشود.

RESET		
`memory_address_register <=۰` صفر در MAR قرار میگیرد تا در چرخه بعد از اولین دستور شروع به اجرا کنیم.		
Micro operation	**Code**	**توضیحات**
PC <- 0	`program_counter <= (others => '0');`	برای ریست کردن کامپیوتر باید PC را صفر کنیم تا همه مراحل از ابتدا اجرا شوند
AC <- 0	`register_ac <= (others => ‘0’);`	
	`state <= fetch;`	به مرحله بعد یعنی fetch میرویم تا چرخه برای دستور بعدی دوباره تکرار شود

بخش سوم) توضیح شمای پیاده‌سازی شده درRTL شماتیک نرم‌افزار:

- کلیت ماژول scomp :

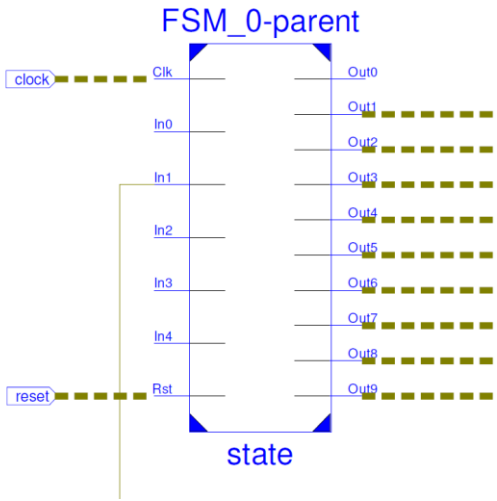
خروجی ها		ورودی ها	
PC	program_counter_out	ریست	Reset
داده موجود در AC	register_ac_out	کلاک	clk
داده موجود در MDR	memory_data_register_out		
داده موجود در MAR	memory_address_register_out		
نشان میدهد آیا داده ای در حافظه اصلی نوشته شده است یا خیر	memory_write_out		



شماتیک فوق کلیت ماژول Simple Computer را نشان می دهد.

- ماشین حالت محدود FSM :

در scomp از ماشین حالت محدود برای تعیین وضعیت و عملکرد CPU در هر کلاک استفاده می شود. در واقع FSM همان کار Control Unit را برای ما انجام می دهد.



- ثبات ها و MM:

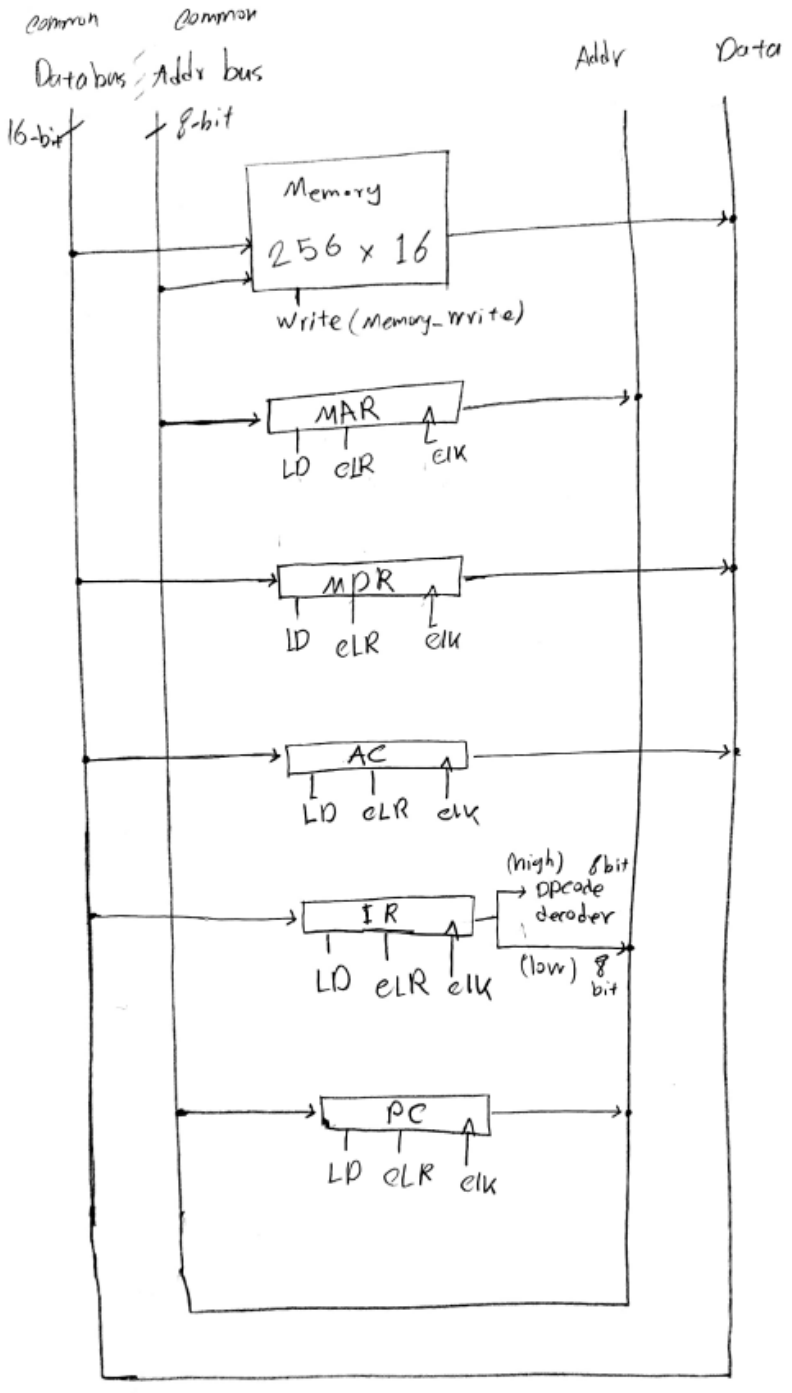
در شمای زیر می توان دید کلی از ارتباط ثبات ها و حافظه اصلی داشت (به جزییات دیگر پرداخته نشده). نکات قابل توجه در شمای زیر:

۱ – MAR به حافظه اصلی دسترسی دارد و میتواند آدرس یک word را به آن بدهد تا داده موجود در آن به عنوان خروجی برگردد و MDR داده خروجی حافظه اصلی در خود ذخیره میکند.

۲ – AC نتایج عملیات ها را در خود ذخیر می کند و برای ذخیره آن داده را به MDR پاس می دهد. (خروجی ALU در شمای زیر ترسیم نشده)

۳ – IR دستور های که از MM خوانده شده را از MDR می گیرد و در خود ذخیره می کند. ۸ بیت کم ارزش خود را به MAR می دهد و 8 بیت پر ارزش را به دیکدر برای اجرای دستور (در شمای زیر ترسیم نشده)

۴ – PC هم علاوه بر اینکه هر بار یک واحد اضافه می شود می تواند در حالت های خاص (مثل JUMP) از AC آدرس بخواند.



ارتباط های بین ریجستر ها.

شماتیک کامل مدار در فایل پی دی اف جداگانه ضمیمه شده است.