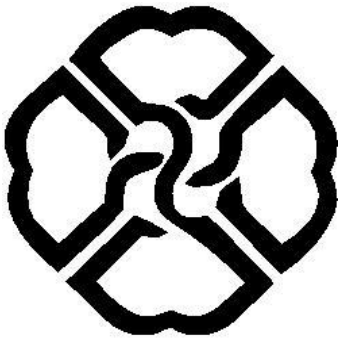
به نام خدا



دانشگاه کردستان

دانشکده مهندسی

آزمایشگاه معماری کامپیوتر

طراحی پردازنده MIPS تک سیکله

موضوع آزمایش:

دراین بخش یک CPU ساده را طراحی و پیاده سازی می‌کنیم که مشابه پردازنده MIPS تدریس شده در کلاس معماری کامپیوتر می‌باشد. این پردازنده دارای چهار رجیستر همه منظوره به نامهای R0تا R3، و یک شمارنده برنامه (PC) برای مشخص کردن دستورالعمل بعدی که باید واکشی شود، می‌باشد.

این پردازنده شبیه MIPS است با این تفاوت که هم دستورالعملها و هم مسیر داده و هم آدرسهای حافظه ۱۶ بیتی هستند.

اجزای اصلی مسیرداده عبارتند از: PC، حافظه دستورالعمل، فایل رجیستر، ALU و حافظه داده.

کل این مدار به کمک نرم افزار logisim شبیه سازی می‌گردد.

مجموعه دستورالعملها ISA

همانطور که گفته شد، دستورالعملها وآدرسهای تولید شده توسط پردازنده ۱۶ بیتی هستند. لذا، ما عرض حافظه دستورالعمل و داده را برابر ۱۶ بیت در نظر می‌گیریم. به عنوان مثال، منظور ما از آدرس 000A یازدهمین خانه حافظه است که یک کلمه ۱۶ بیتی در آن قرار دارد. در ۱۰ خانه قبلی (یعنی 0000 تا 0009) نیز ۱۰ کلمه ۱۶ بیتی قرار گرفته‌اند. ساختار دستورالعمل ها در جدول زیر داده شده است. چهار بیت باارزش دستورالعمل، یعنی بیتهای ۱۲ تا ۱۵ نشان دهنده کد دستور (OP-Code) است. دقت کنید تمام دستورات نوع R دارای کد صفر هستند و از طریق ۳ بیت کم ارزش (فیلد func) کار آنها مشخص می‌گردد که در جدول بعدی داده شده است.

15-12	11 10	9 8	7 6	5 4 3	2 1 0	
0000:0	rs	rt	rd	Not used	funct	دستورات نوع R
0001:1	Not used					دستور NOP
0010:2	rs	rt	immediate (signed)			دستور lui rt imm برای بار گذاری نیمه بالایی rt با imm $rt = imm << 8
0011:3	rs	rt	immediate (signed)			دستور ori rt rs imm برای or کردن rs و imm و ذخیره نتیجه در rt $rt = $rs | imm
0100:4	rs	rt	immediate (signed)			دستور addi rt rs imm $rt = $rs + imm
0101:5	rs	rt	immediate (signed)			دستور andi rt rs imm مثل ori است.
0110:6	rs	rt	immediate (signed)			دستور lw rt imm(rs) $rt = MEM[$rs + imm]
0111:7	rs	rt	immediate (signed)			دستور sw rt imm(rs) MEM[$rs+imm] = $rt
1000:8	target address					دستور jump برای پرش به target
1001:9	rs	rt	offset (signed)			دستور beq. این دستور شبیه MIPS است با این تفاوت که offset در ۴ ضرب نمی‌شود.
1010:A	rs	rt	offset (signed)			دستور bne

دستورات نوع R

funct	معنی
000	or: $rd = $rs | $rt
001	and: $rd = $rs & $rt
010	add: $rd = $rs + $rt
011	sub: $rd = $rs - $rt
100	sllv: $rd = $rs << $rt
101	srlv: $rd = $rs >> $rt
110	srav: $rd = $rs >> $rt
111	slt: $rd = ($rs < $rt) ? 1 : 0

در ادامه، راجع به بعضی دستورالعملها نکاتی ذکر می‌شود. اولاً، دستور بعدی در PC+1 قرار دارد نه در PC+4. در دستور jump مقدار target در ۴ ضرب نمی‌شود. هم چنین، ۴ بیت باارزش آدرس پرش از روی PC+1 به دست می‌آید. به عنوان مثال اگر PC=F001 و target=2A3 باشد، آدرس پرش با F2A3 برابر است.

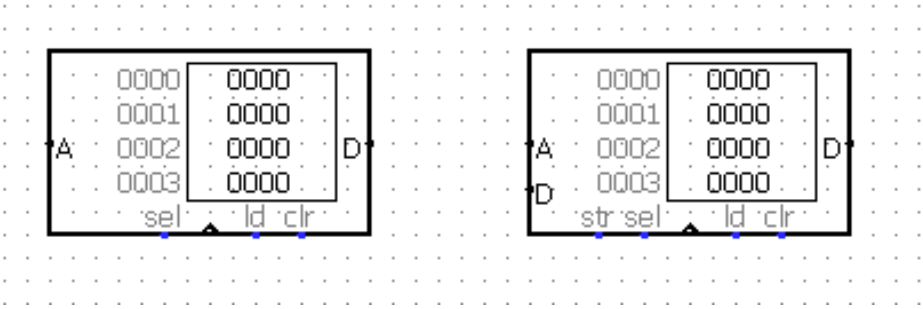
در دستورات beq و bne مقدار offset به صورت نسبی و علامت‌دار در نظر گرفته می‌شود و در ۴ ضرب نمی‌شود. هم‌چنین، با PC+1 جمع می‌شود. به عنوان مثال، beq $r3, $r1, 100 به این معنا است که اگر $r3 و $r1 برابر بودند به آدرس PC+101 پرش کنید. در غیر این صورت، دستور بعدی در PC+1 است.

دقت کنید فیلد immediate یک فیلد ۸ بیتی است. لذا، لازم است که قبل از ارسال به ALU گسترش داده شود. در جاهایی که immediate بدون علامت است، باید ۸ صفر به آن اضافه شود. در جاهایی که immediate دارای علامت باشد، باید طبق بیت علامت گسترش داده شود.

ماژولهای مورد نیاز برای حافظه داده و دستورالعمل

در نرم افزار Logisim ماژول RAM به صورت یک کتابخانه پیش‌ساخته (به همراه سایر ماژولهای دیگر حافظه) موجود است. از RAM به عنوان حافظه داده استفاده کنید. برای حافظه دستورالعمل از ROM استفاده کنید.

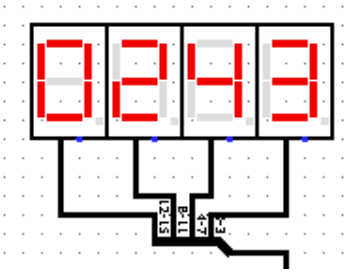
ماژول RAM را می‌توان هم به صورت یک پورت داده و هم به صورت دوپورت داده مجزا (برای load و store) استفاده کرد. ورودیهای این ماژول به صورت زیر هستند.



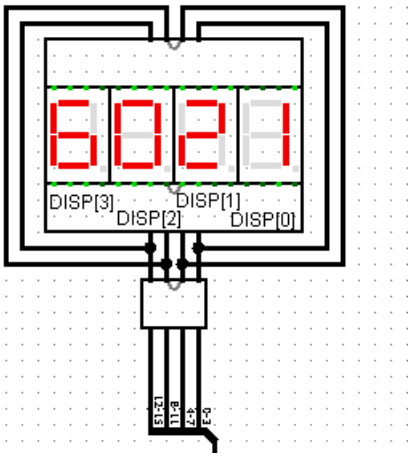
- ورودی sel تعیین می‌کند که آیا حافظه فعال است یا خیر.
- ورودی A مشخص کننده آدرس مورد نظر از حافظه است.
- ورودی clr تمام عناصر حافظه را برابر 0 قرار می‌دهد.
- ورودی ld به خط کنترلی Memory_Read وصل می‌شود. اگر ld یک باشد، خروجی برابر محتویات حافظه در خانه A خواهد بود.
- ورودی str نقش Memory_Write را بازی می‌کند. اگر یک باشد، داده وارد شده از پورت ورودی D در خانه A حافظه نوشته می‌شود.

می‌توانید از ابزار poke برای تغییر محتوی حافظه استفاده کنید. همچنین، می‌توانید روی حافظه کلیک راست کنید و برای بارگذاری مقادیر حافظه از یک فایل گزینه Load Image را انتخاب کنید.

جهت راحتی کار شما و نمایش محتویات باسها، یک فایل به اسم converter.circuit به شما داده می‌شود که ۱۶ بیت را به ۴ رقم هگزادسیمال تبدیل می‌کند و روی چهار 7-segment نمایش داده می‌شود. همچنین، می‌توانید از ابزار Hex digit display استفاده نمود.



ابزار Hex digit display



ابزار converter

پردازنده دارای سه عنصر ترتیبی فایل رجیستر، حافظه داده و PC می‌باشد که اصطلاحا **المانهای نگهدارنده حالت** نامیده می‌شوند. سایر عناصر مسیر داده، ترکیبی هستند.

در این روش پیاده‌سازی، تمامی دستورالعملها برای اجرا به زمان یکسانی احتیاج دارند (یک پالس ساعت). پس طول کلاک باید برابر با طولانیترین دستورالعمل باشد.
دستورات مراحل زیر را برای اجرا طی می‌کنند:

- **واکشی دستورالعمل از حافظه دستورالعمل**
- **کدبرداری (دیکد کردن) دستورالعمل**
- **خواندن عملوندها**
- **اجرای دستورالعمل**
- **نوشتن یا خواندن در حافظه داده (در دستورات load و store)**
- **ثبت نتیجه در رجیستر مقصد (در صورت لزوم)**

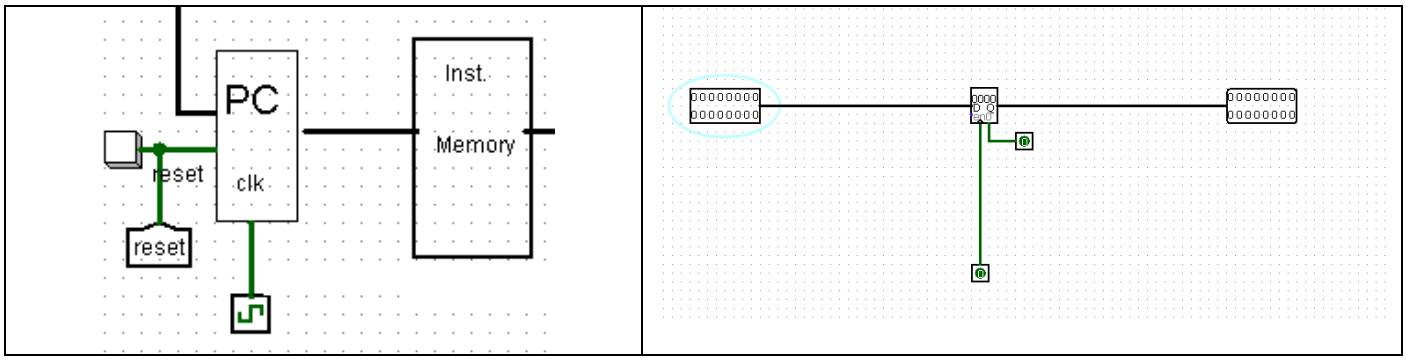
در پیاده سازی Single-cycle تمامی این مراحل در یک کلاک انجام می‌شوند.

۱

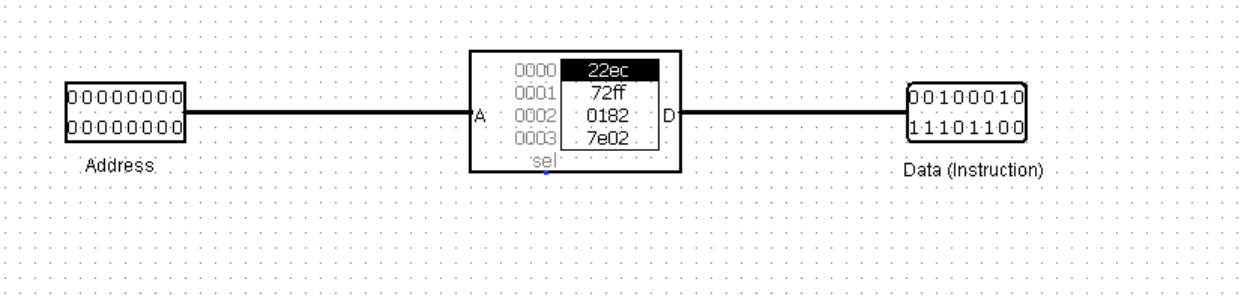
آزمایش اول

شامل شمارنده برنامه (PC) وحافظه دستورالعمل (مراحل واکشی و دیکود)

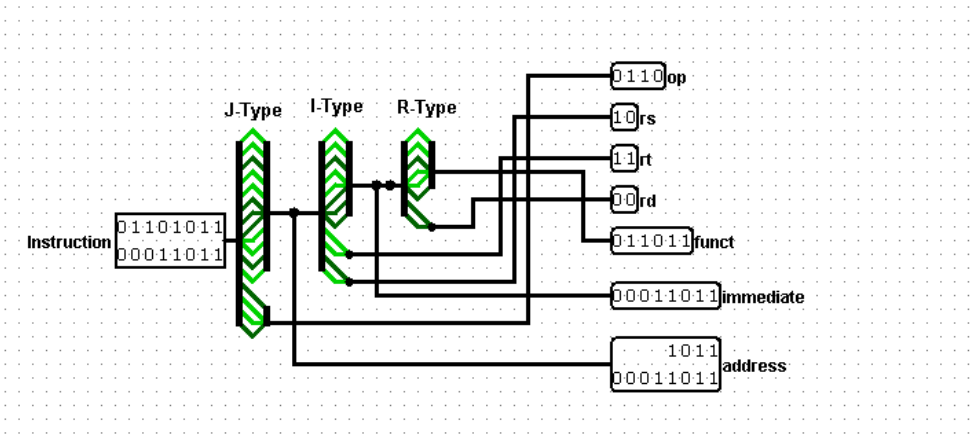
در این آزمایش با نحوه واکشی دستورات و دیکود آنها آشنا می‌شوید. یک رجیستر به نام PC (شمارنده برنامه) وجود دارد که آدرس دستورالعمل بعدی را نگهداری می‌کند. دقت کنید که این رجیستر از دید برنامه‌نویس مخفی بوده و توسط واحد کنترل مقداردهی می‌شود. این رجیستر دارای ورودی و خروجی ۱۶ بیتی و همچنین قابلیت reset کردن می‌باشد.



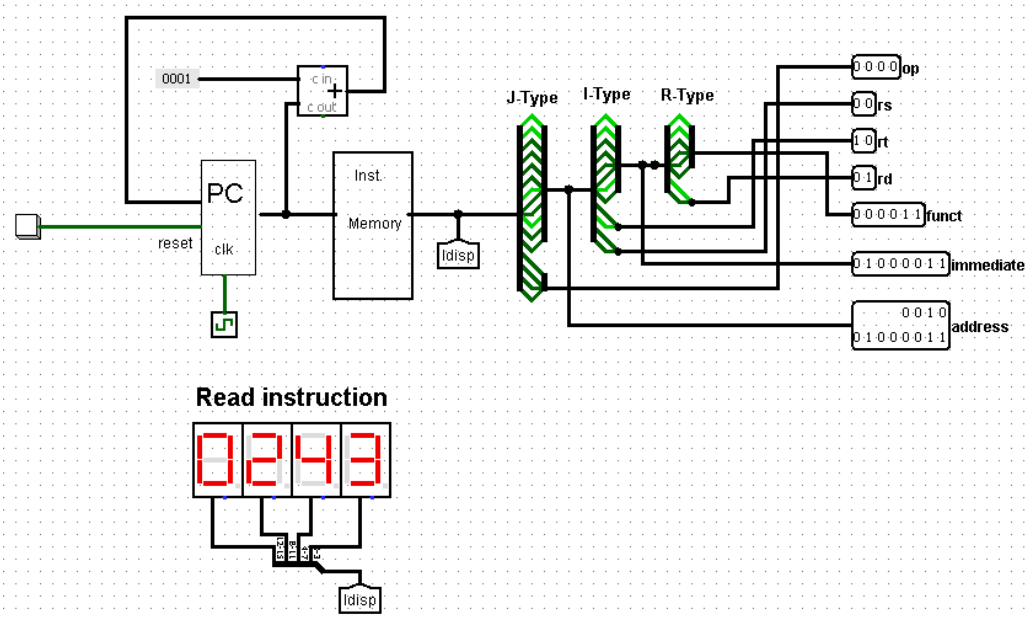
خروجی PC به عنوان آدرس ورودی به حافظه دستورالعمل داده می‌شود. حافظه دستورالعمل از نوع ROM بوده که ورودی آن آدرس ذخیره شده در PC و خروجی آن، دستورالعمل مربوطه می‌باشد.



پس از واکشی دستور العمل باید آن را دیکود نمود. برای دیکود کردن دستورالعمل می‌توان از مداری مشابه شکل زیر استفاده نمود (هر روش ابتکاری دیگری را نیز می‌توان استفاده کرد):



تلفیق دو فاز واکشی و دیکود در شکل زیر داده شده است.



دقت کنید که فعلا آدرس بعدی PC همان PC+1 می‌باشد. اما در ادامه خواهیم دید که این آدرس می‌تواند یک آدرس نسبی (در دستورات beq و bne) و یا مقصد پرش (در دستور jump) باشد.

برای دستورات beq و bne آدرس مقصد به صورت زیر محاسبه می‌گردد:

beq.Zero + bne.Zero' : Next PC ← (PC+1) + SignExten(Imm)

برای دستور Jump آدرس مقصد به صورت زیر محاسبه می گردد:

Jump: Next PC ← PC(15-12) . 12 bit target address

مدارات مورد نیاز برای دو مورد فوق را طراحی کنید.

۲

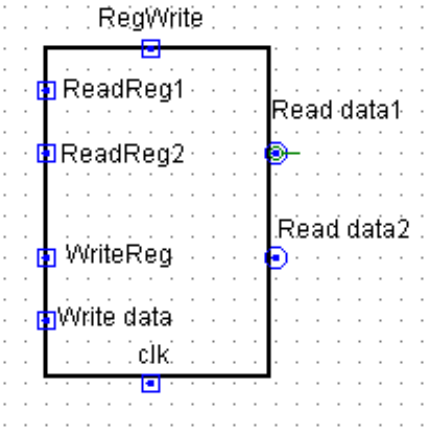
آزمایش دوم (طراحی فایل رجیستر)

فایل رجیستر دارای ۴ رجیستر ۱۶ بیتی است. ورودیهای این ماژول به صورت زیر هستند:

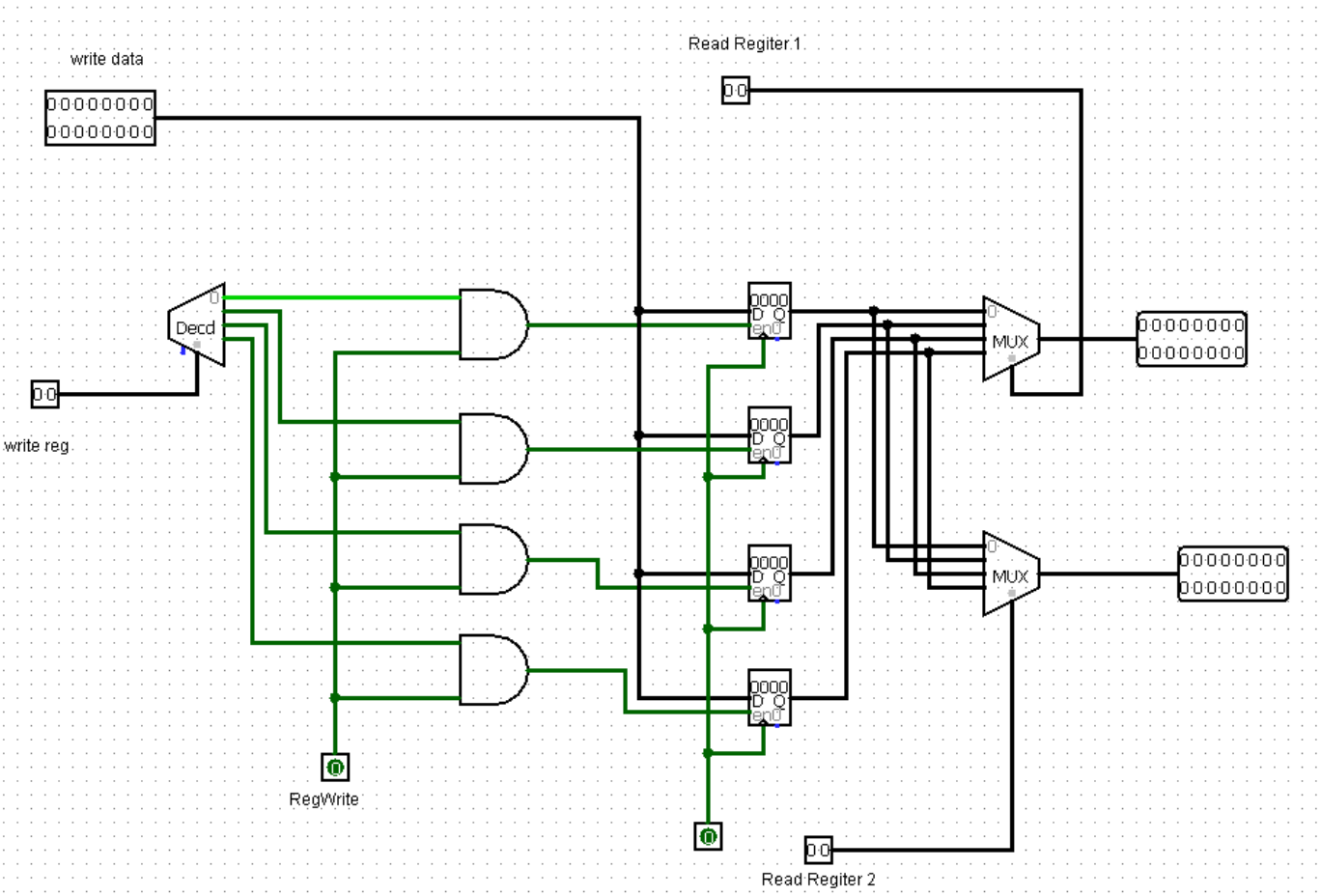
Read Register 1 (2 bits)	شماره رجیستر خواندنی اول
Read Register 2 (2 bits)	شماره رجیستر خواندنی دوم
Write Register (2 bits)	شماره رجیستر نوشتن
Write data (16 bits)	داده‌ای که باید در رجیستر نوشتن نوشته شود.
RegWrite (1 bit)	اگر این سیگنال یک باشد، write data در رجیستر نوشتن نوشته می‌شود.
clk (1 bit)	کلاک

این ماژول دارای خروجیهای زیر است:

Read Data 1 (16 bits)	مقدار رجیستر خواندنی اول
Read Data 2 (16 bits)	مقدار رجیستر خواندنی دوم



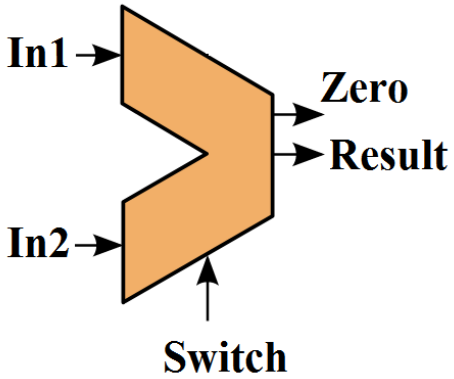
ماژول فایل رجیستر را می‌توان به شکل زیر طراحی نمود:



۳
آزمایش سوم

طراحی واحد محاسبه و منطق (ALU)

ALU یکی از مهمترین قسمتهای هر پردازنده بوده و یک مدار ترکیبی است که عملیات محاسباتی و منطقی مانند جمع، تفریق، AND و ... را انجام می‌دهد. در این بخش به کمک گیتهای پایه AND ، NOT،OR و مالتی پلکسر یک واحد ALU را طراحی خواهیم نمود.



ورودیهای واحد ALU عبارتند از:

In1 (16 bits)	ورودی ۱۶ بیتی اول
In2 (16 bits)	ورودی ۱۶ بیتی دوم
switch (3 bits)	عملی که باید روی ورودیهای فوق انجام شود.

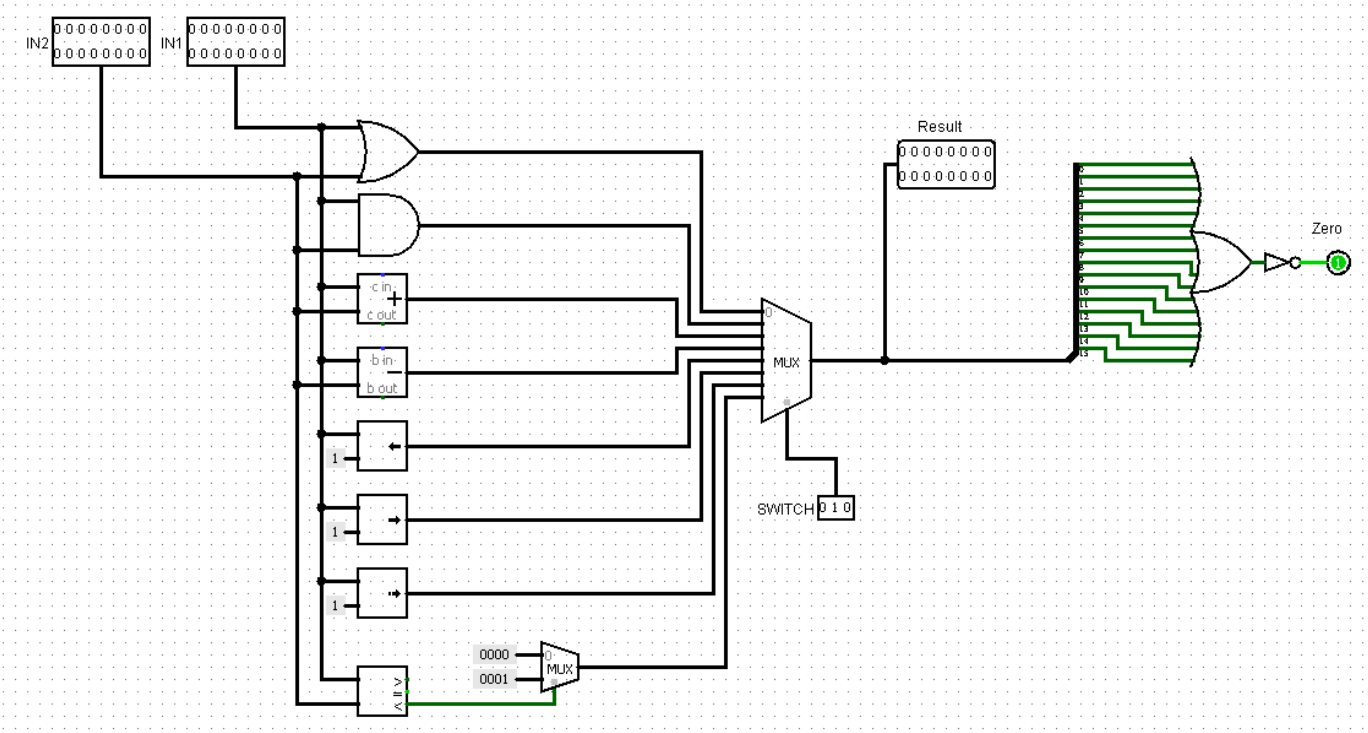
خروجیهای ALU عبارتند از:

Result (16 bits)	نتیجه ۱۶ بیتی
Zero (1 bit)	اگر نتیجه ALU صفر باشد، این خروجی برابر با ۱ خواهد شد.

عملی که ALU باید انجام دهد از طریق ورودی سه بیتی switch به صورت جدول زیر معلوم می‌گردد:

switch	معنی
0	or: result = In1 | In2
1	and: result = In1 & In2
2	add: result = In1 + In2
3	sub: result = In1 – In2
4	sllv: result = In1 << 1
5	srlv: result = In1 >> 1 (zero-fill)
6	srav: result = In1 >> 1 (sign-fill)
7	slt: result = (In1 < In2) ? 1 : 0 (treat In1 and In2 as *signed*)

مدار طراحی شده در logisim می‌تواند مشابه شکل زیر باشد:



۴

آزمایش چهارم (تکمیل مسیر داده)

ابتدا المانهای اصلی را از چپ به راست به ترتیب زیر بچینید:

۱. شمارنده برنامه (PC)

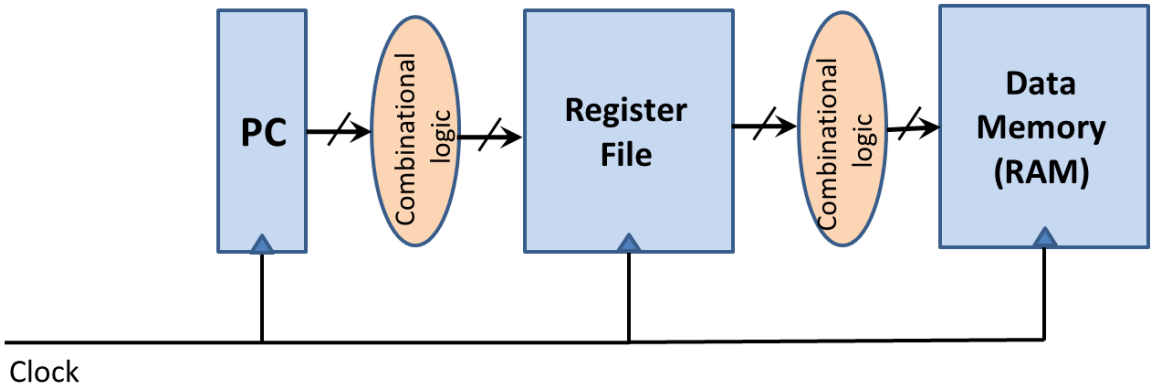
۲. حافظه دستورالعمل

۳. فایل رجیستر

۴. ALU

۵. حافظه داده

دقت کنید که در المانهای ترتیبی(۱ و ۳ و ۵) برای تغییر حالت (نوشتن) کلاک لازم است و برای خواندن نیازی با کلاک نیست. درک این موضوع برای فهم بهتر عملکرد single-cycle بسیار مهم است.



سپس اتصالات را برای اجرای دستورات نوع R برقرار کنید. در ادامه، دستورات LW و SW را هم اضافه نمایید (برای این کار لازم است عناصر دیگری مانند مالتی پلکسر و توسعه دهنده علامت را به مدار اضافه کنید).

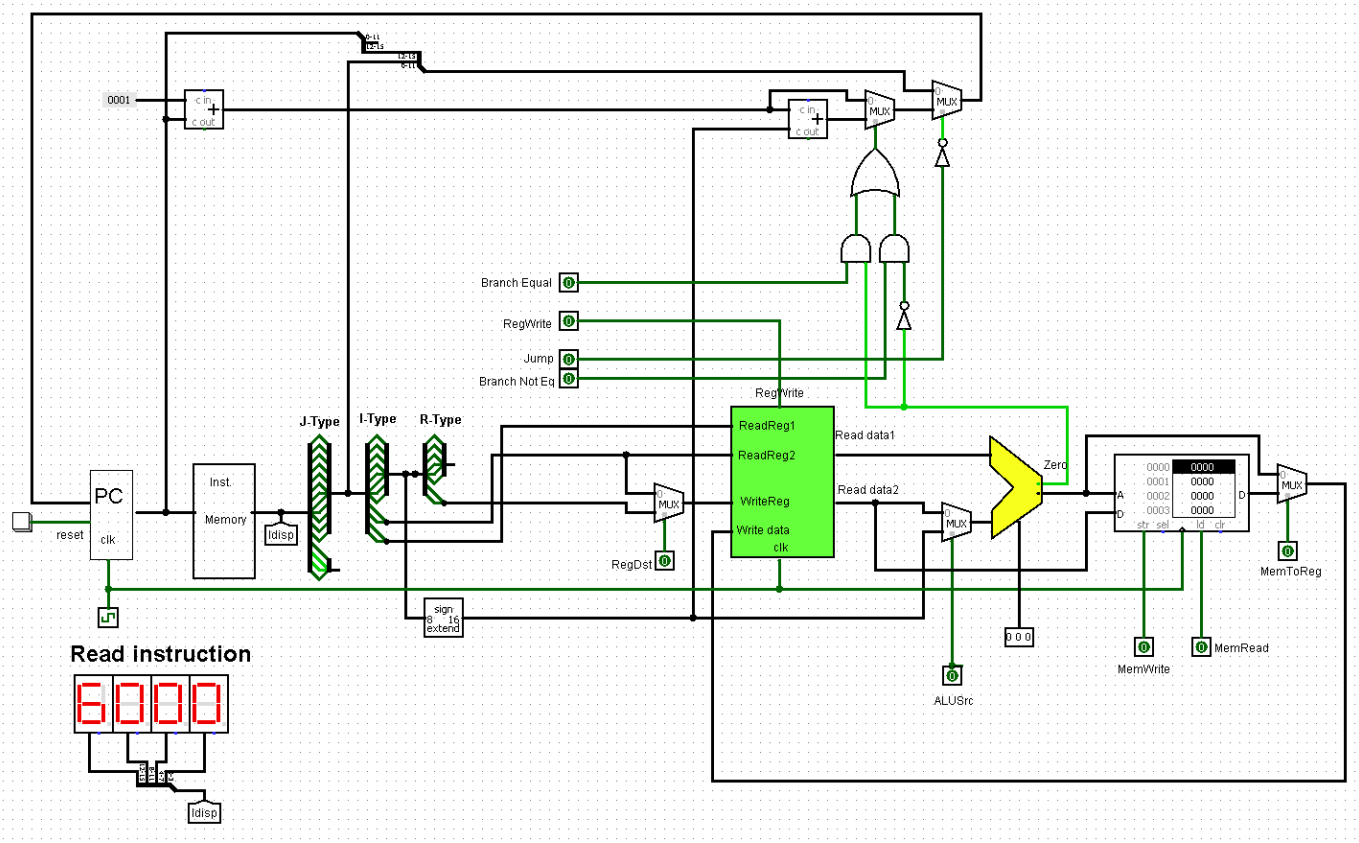
تمامی سیگنالهای کنترلی که لازم است را مشخص نمایید و بسته به نوع دستور به آنها بصورت دستی مقدار بدهید.

مقادیر 12 و 09 را به صورت دستی در دو خانه ابتدایی حافظه داده وارد نمایید.

چهار دستور زیر را به ترتیب توسط مسیر داده خود اجرا کنید:

LW r1 , 0(r0) : 6100
LW r2 , 1(r0) : 6201
ADD r3, r1, r2 : 06c2
SW r3,2(r0)   : 7302

دقت کنید که فعلا در این مرحله سیگنالهای کنترلی به صورت دستی برای هر دستور مشخص می‌گردند.
در آزمایش بعدی این کار توسط واحد کنترل و از روی Op-code دستور انجام می‌گردد.



برای دستورات jump و beq و bne نیازی به اضافه نمودن المانهای دیگر به مسیر داده نمی‌باشد و تنها با تنظیم نمودن سیگنالهای کنترلی مربوطه می‌توان آنها را اجرا نمود.
برای پیاده‌سازی دستور lui نیاز به اضافه نمودن چه عناصری داریم؟ برای سایر دستورات چطور؟
دستورات زیر را پس از تکمیل مسیر داده اجرا نمایید:

LUi r1 , 22(r0)
LUi r2 , 11(r0)
ADD r3, r1, r2

۵

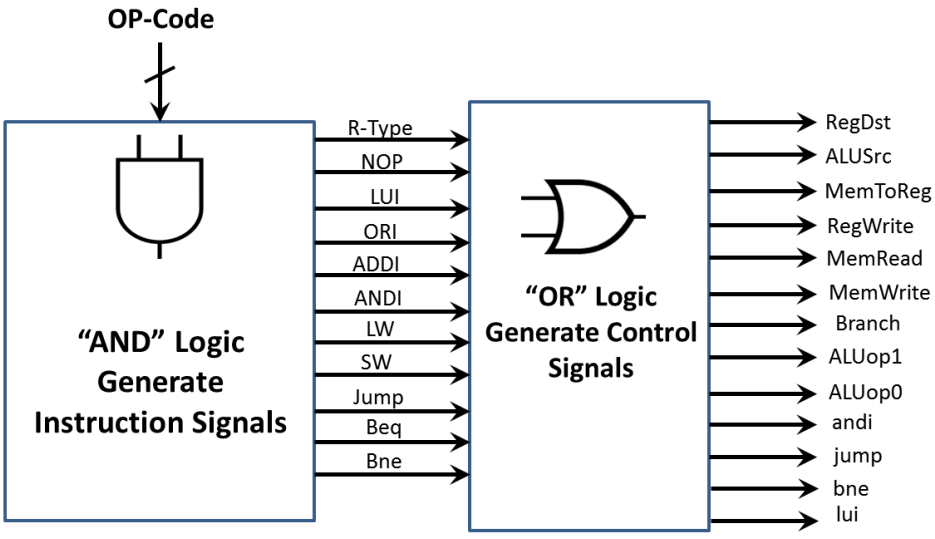
آزمایش پنجم (طراحی واحد کنترل)

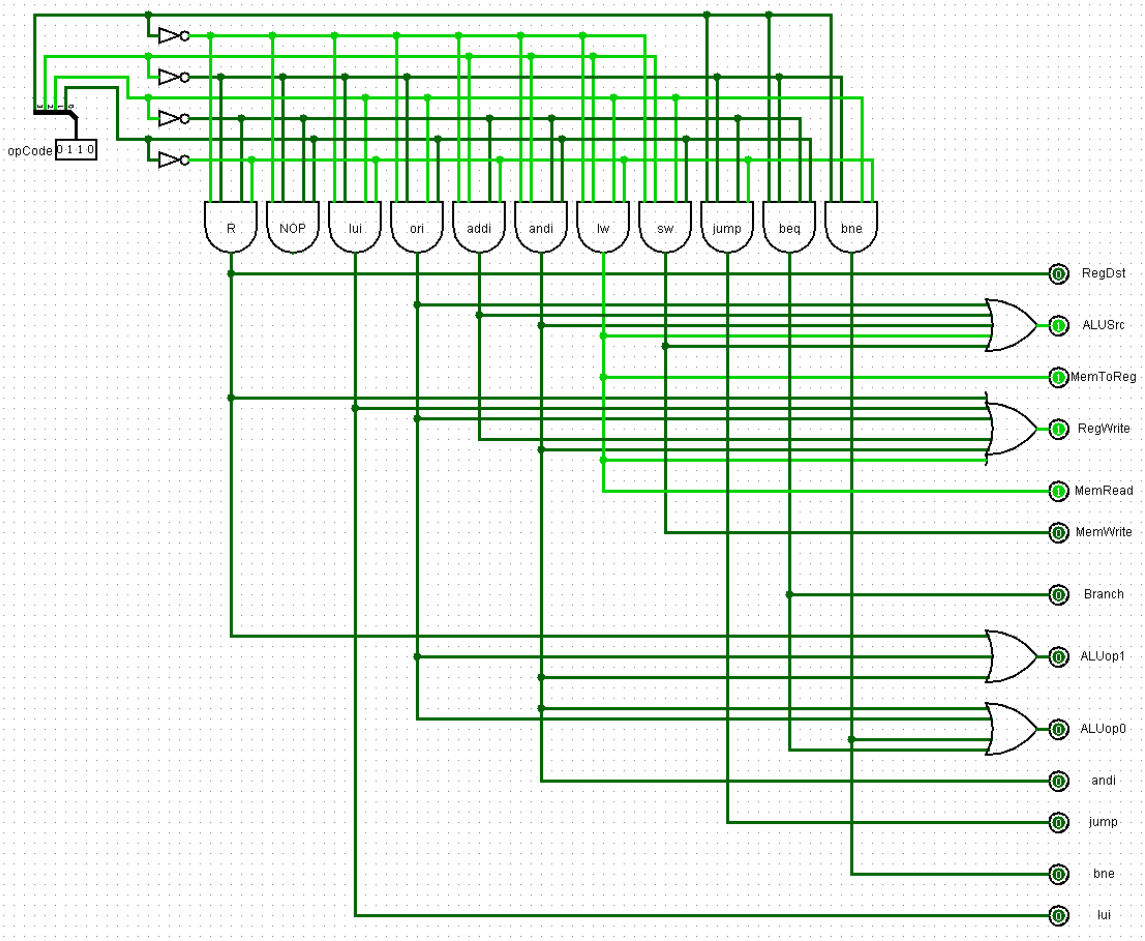
وظیفه این واحد تولید سیگنالهای کنترلی برای مسیرداده است. سیگنالهای کنترلی برای دستورات مختلف متفاوت بوده و بنابراین، قسمت op-code دستور به عنوان ورودی به واحد کنترل داده می‌شود.
توجه به این نکته ضروری است که این واحد خود دستورات را اجرا نمی‌کند بلکه دیگر اعضای سیستم را وادار به انجام آنها می‌کند. به این منظور، این واحد باید با سایر المانهای مسیرداده ارتباط داشته باشد.

واحد کنترل پردازنده MIPS تک سیکله به صورت Hardwired و با المانهای ترکیبی پیاده‌سازی می‌گردد. جدول زیر ورودی و بیتهای خروجی واحد کنترل را نشان می‌دهد.

OP-code		RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	AlUOp0	andi	Jump	bne	LUi
R-Type	0	1	0	0	1	0	0	0	1	0	0	0	0	0
NOP	1	0	0	0	0	0	0	0	0	0	0	0	0	0
LUi	2	0	0	0	1	0	0	0	0	0	0	0	0	1
ORi	3	0	1	0	1	0	0	0	1	1	0	0	0	0
ADDi	4	0	1	0	1	0	0	0	0	0	0	0	0	0
ANDi	5	0	1	0	1	0	0	0	1	1	1	0	0	0
LW	6	0	1	1	1	1	0	0	0	0	0	0	0	0
SW	7	0	1	0	0	0	1	0	0	0	0	0	0	0
Jump	8	0	0	0	0	0	0	0	0	0	0	1	0	0
Beq	9	0	0	0	0	0	0	1	0	1	0	0	0	0
Bne	10	0	0	0	0	0	0	0	0	1	0	0	1	0

واحد کنترل را می‌توان به صورت یک مدار ترکیبی با استفاده از PLA پیاده سازی نمود.





واحد کنترل ALU

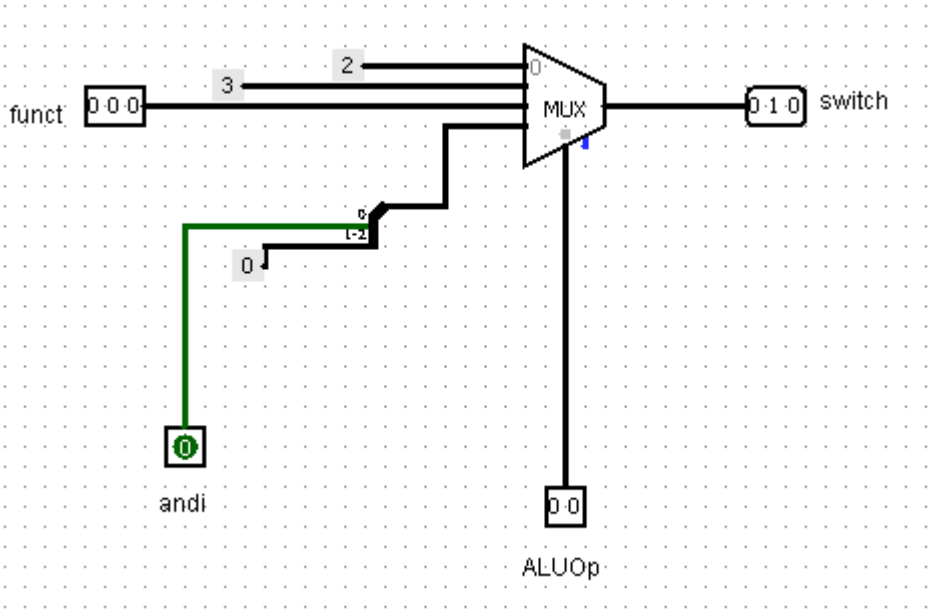
برای طراحی این قسمت به نکات زیر توجه کنید:

دو بیت خروجی ALUop0 و ALUop1 که توسط واحد کنترل اصلی تولید می‌شوند، چهار حالت داشته و به عنوان ورودی به این بخش داده می‌شود:

- اگر دستور از نوع R-type باشد(حالت 10) ، عمل ALU توسط فیلد function مشخص می‌گردد.
- اگر دستور beq یا bne باشد (حالت 01)، ALUعمل تفریق را انجام می‌دهد. (switch =011)
- اگر دستور LW یا SW یا ADDi باشد (حالت 00)، ALUعمل جمع را انجام می‌دهد. (switch =010)
- اگر دستور ORi یا ANDi باشد (حالت 11)، ALUعمل مربوطه (OR یا AND) را انجام می‌دهد. (001 یا switch =000)

OP-code		ALUOp1	AlUOp0
R-Type	0	1	0
NOP	1	0	0
LUi	2	0	0
ORi	3	1	1
ADDi	4	0	0
ANDi	5	1	1
LW	6	0	0
SW	7	0	0
Jump	8	0	0
Beq	9	0	1
Bne	10	0	1

مدار کنترل ALU می‌تواند مشابه شکل زیر باشد:

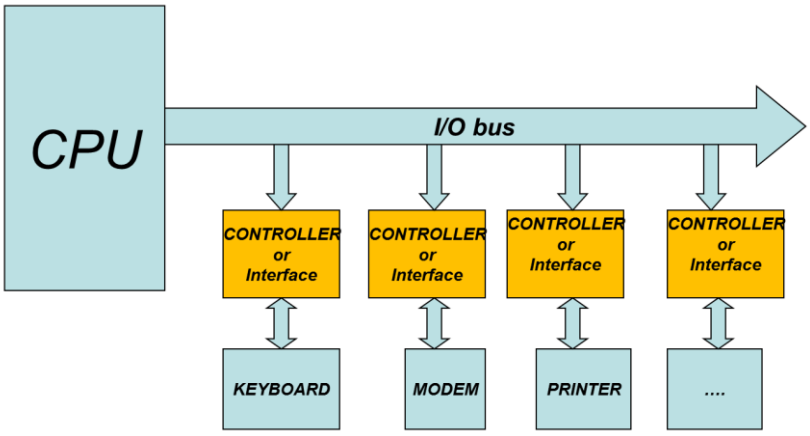


۶

آزمایش ششم

(I/O نگاشته شده به حافظه)

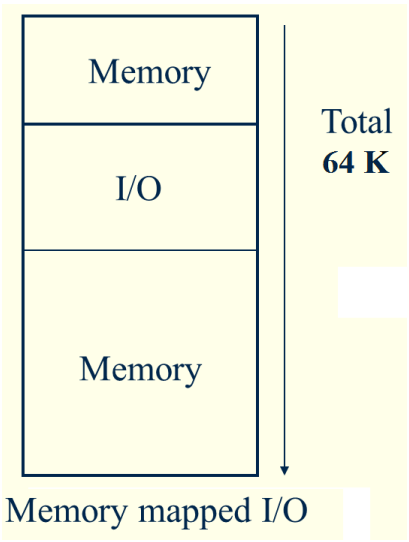
در این بخش می‌خواهیم قابلیت خروج اطلاعات را به پردازنده اضافه کنیم. برای این کار چند روش وجود دارد که از جمله متداولترین آنها می‌توان به روش I/O نگاشته شده به حافظه (memory-mapped I/O) اشاره نمود. در این روش دستورات خاصی برای I/O وجود ندارد و فقط آدرس مشخص می‌کند که هدف برقراری ارتباط با I/O است یا حافظه. به طوری که بخشی از آدرسهای حافظه به I/O نگاشته شده‌اند. پردازنده‌ها از طریق یک واسط(کنترل کننده) با I/O مطابق شکل زیر در ارتباط هستند.

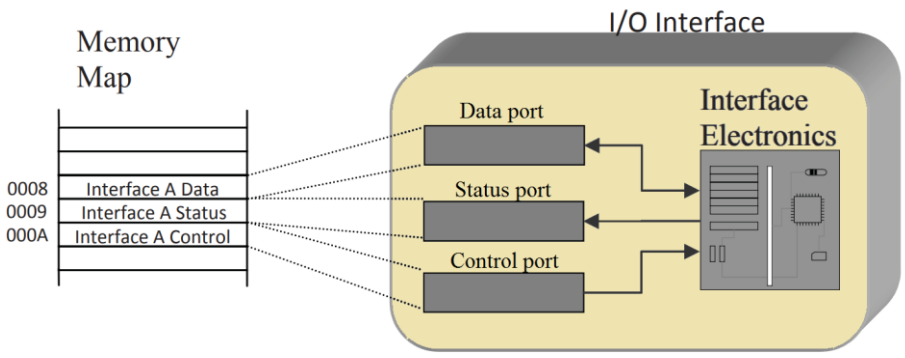


از دلایل لزوم کنترل کننده یا واسط می‌توان به موارد زیر اشاره نمود:

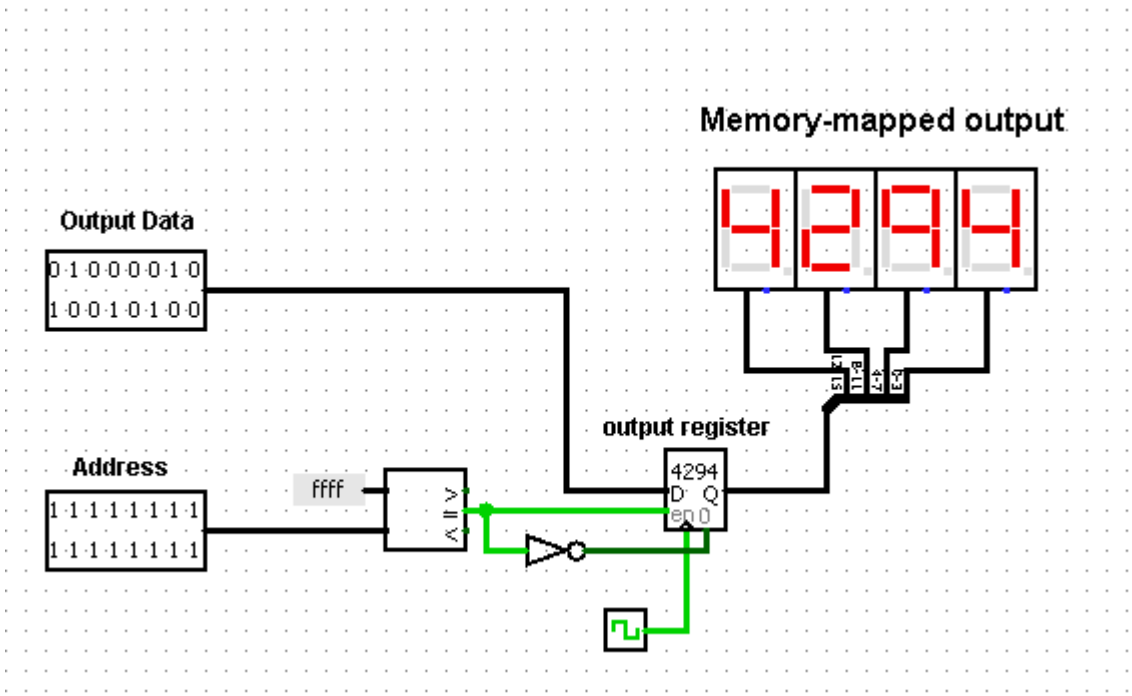
- ابزارهای غیرالکترونیکی (الکترومکانیکی یا الکترومغناطیسی) ← لزوم تبدیل سیگنالها
- تفاوت در سرعت انتقال ← لزوم همزمانی
- انواع قالب متفاوت
- مدهای عملیاتی مختلف

واسط معمولا شامل تعدادی رجیستر (داده – وضعیت – کنترل) برای ارتباط با پردازنده می‌باشد. هر رجیستر باید آدرس مشخص داشته باشد. در اینجا، پورت خروجی به صورت یک رجیستر مدل می‌گردد که با آدرس FFFF فعال می‌شود. برای نمایش محتویات رجیستر خروجی از ابزار Hex digit display استفاده می‌کنیم.



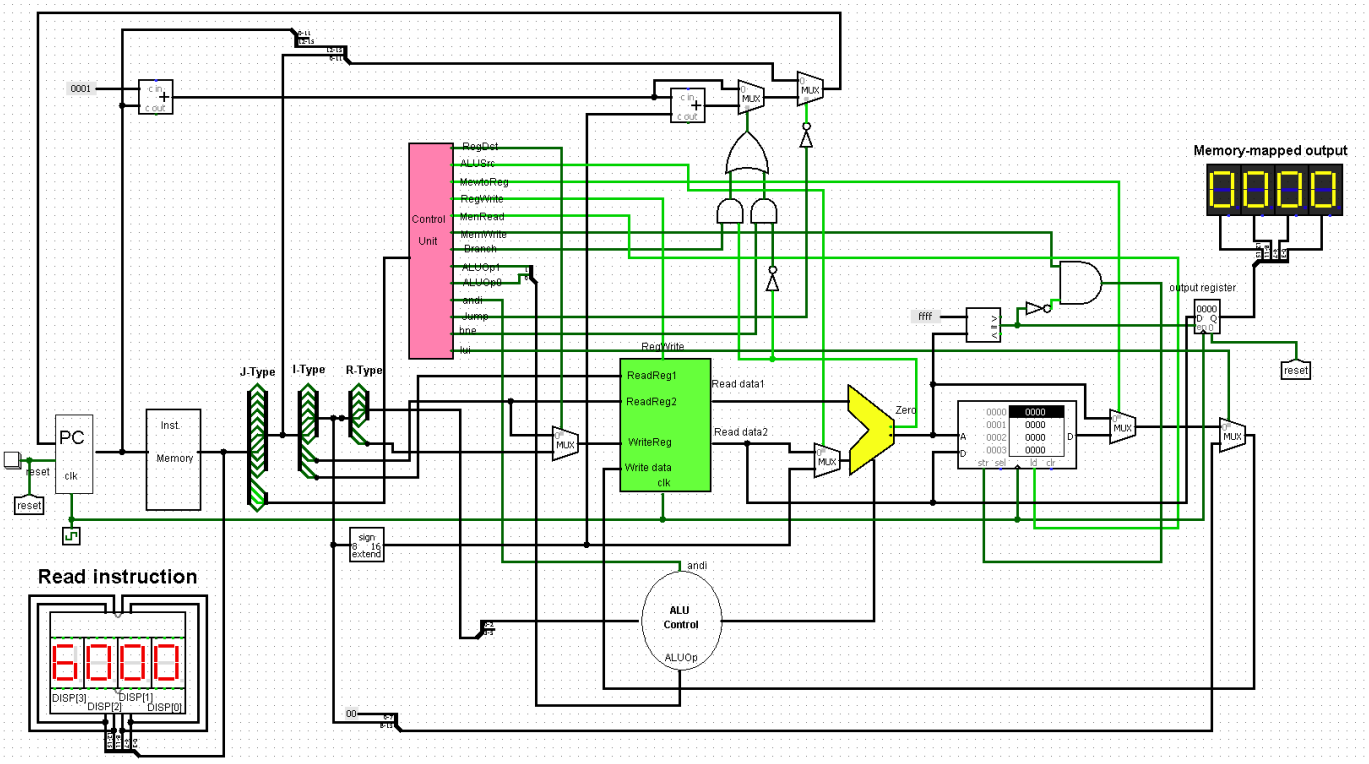


مداراتی را مشابه شکل زیر اضافه نمایید که با دستور SW به آدرس مشخصی (مثلا FFFF) به جای اینکه کلمه مورد نظر در حافظه ذخیره شود به پورت خروجی منتقل گردد. سپس محتوای پورت خروجی را نمایش دهید.



دستوراتی را برای تست این قسمت بنویسید.

مدار کامل CPU طراحی شده مشابه شکل زیر است:



برنامه زیر را جهت تست پردازنده طراحی شده بارگذاری و اجرا نمایید.

```
6000   lw  $r0, 0($r0);       The value of r0 after execution is 13
6501   lw  $r1, 1($r1);       The value of $ r1 after execution is 9
0182   add $r2, $r0, $r1;   The value of $ r2 after execution is 1c
7e02   sw  $r2, 2($r3);       The value of MEM [2] after execution is 1c
0183  sub $r2, $r0, $r1;    The value of $ r2 after execution is a
7e02   sw  $r2, 2($r3);      The value of MEM [2] after execution is a
0181  and $r2, $r0, $r1;   The value of $ r2 after execution is 1
7e02   sw  $r2, 2($r3);      The value of MEM [2] is 1 after execution
0180   or  $r2, $r0, $r1;    The value of $ r2 after execution is 1b
7e02   sw  $r2, 2($r3);      The value of MEM [2] after execution is 1b
3225   ori $r2, $r0, 37;      The value of $ r2 after execution is 37
4714   addi $r3, $r1, 20;   The value of $ r3 after execution is 1d
5943   andi $r1, $r2, 67;   The value of $ r1 after execution is 3
94fe   beq $r1, $r0, -2;       $ R1 = 3, $ r0 = 13, not equal, the result does not jump
ae04   bne $r3, $r2, 4;       $ R3 = 1d, $ r2 = 37, not equal, the instruction goes to the
current instruction + 1 + 4 instructions
```

4155 addi $r1, $r0, 85; Not performed
0241 and $r1, $r0, $r2; Not performed
3144 ori $r1, $r0, 68; Not performed
0243 sub $r1, $r0, $r2; Not performed
4510 addi $r1, $r1, 16; $ R1 = $ r1 + 10 after execution
94fe beq $r1, $r0, -2; After the last step, $ r1 = 13 = $ r0, the branch condition is set to go to the previous addition.

6021 lw $r0, 33($r0); After execution $r0=MEM[34]=0
6511 lw $r1,17 ($r1); After execution $r1=MEM[34]=0
6afd lw $r2, -3($r2); After execution $r2=MEM[34]=0
6f18 lw $r3, 26($r3); After execution $r3=MEM[35]=0
20AB lui $r0, AB;
74FF sw $r0, $r1(FF);
8032 jump 32; After execution, jump to Article 32 instructions