دانشگاه صنعتی شریف

دانشکده مهندسی برق

درس ساختار کامپیوتر

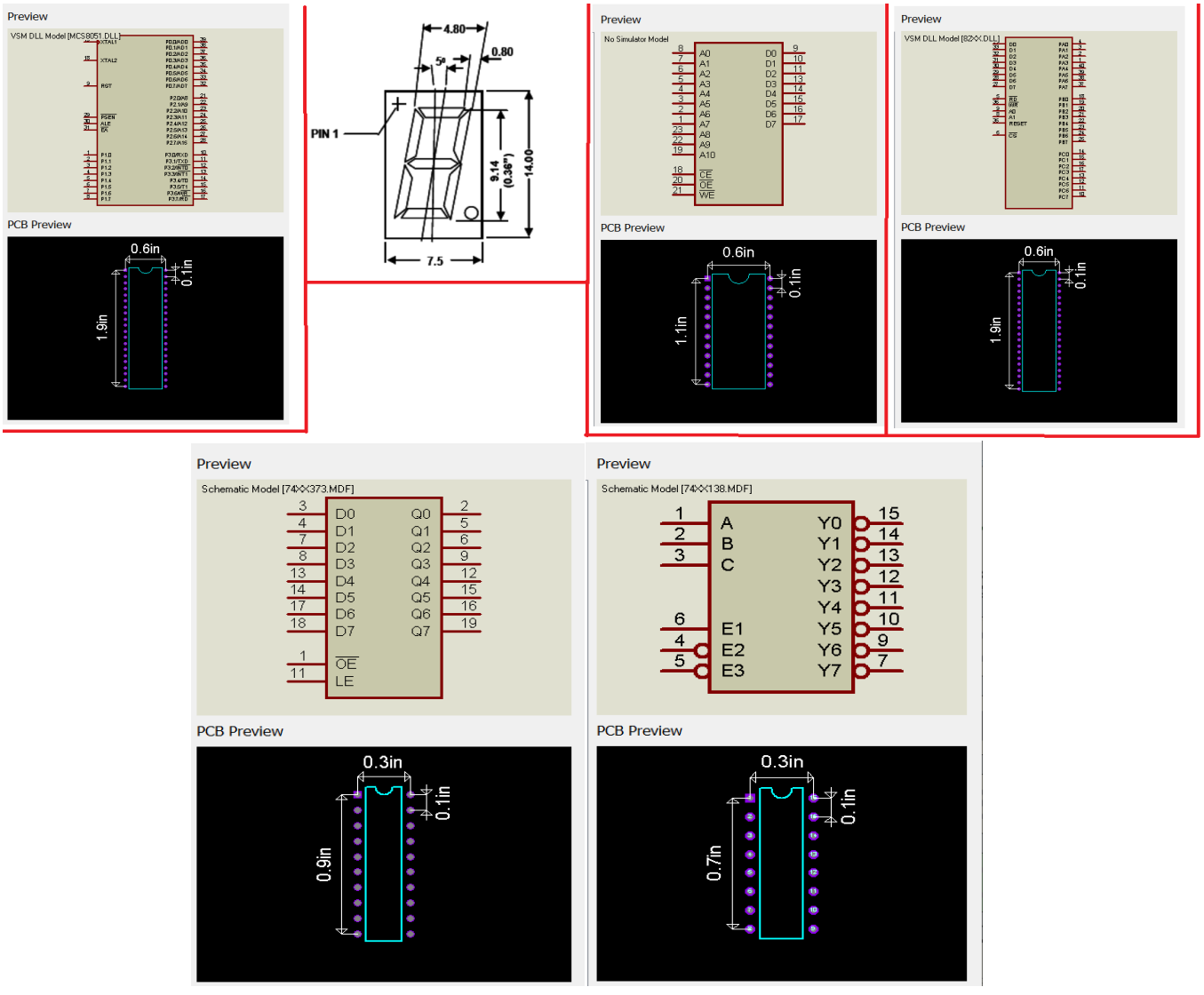
پروژه شماره یک

استاد درس : دکتر باقری

تهیه کنندگان :
محمدامین مطهری نیا 96102404
محمدحسن مزیدی 96102375
علیرضا رفیعی ساردوئی 97101723

بخش اول : الف) تبیین مساله به زبان خود : هدف پروژه در دستور کار مشخص شده است، مساله ای که باید به آن پرداخت طراحی سخت افزار و نرم افزاری است که سخت افزار امکان ورودی گرفتن از کاربر را فراهم کند همچنین به نمایشگر مجهز باشد. نرم افزار میبایست قابلیت کار در دو مد کاری را داشته باشد. در مد اول باید بتواند داده های صفحه کلید را در آدرس مشخصی از حافظه ی بیرونی بنویسد و در مد دوم امکان واکشی داده از حافظه بیرونی و تشخیص opcode, operand ها و اجرای آن ها به دو صورت مختلف که در بخش هدف پروژه (در صورت پروژه) ذکر شده است را داشته باشد. همچنین میبایست دستورهایی به کاربر برای استفاده از سیستم نمایشگر داده شود.

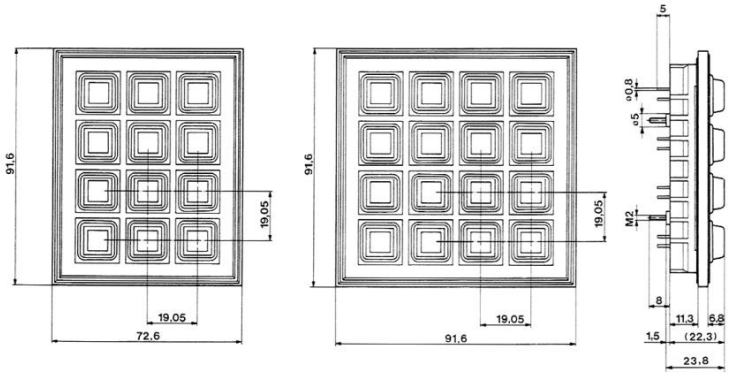
ب) ابعاد فنی محصول : سیستم ما حداقل از میکروکنترلر 8051 و 8 عدد $Segment - 7$ و یک کیبورد 4×4 تشکیل شده است اما برای اینکه بتوانیم سیستم با مشخصات خواسته شده را طراحی کنیم نیازی به چند IC دیگر نیز داریم از جمله این IC ها می‌توان به دو $P.P.I$ و یک حافظه خارجی و یک $Latch$ برای تمایز باس آدرس و دیتا اشاره کرد . حال فرض کنیم تنها از قطعات ذکر شده استفاده می‌کنیم و میخواهیم ابعاد فنی خروجی را بدست آوریم باید به دیتا شیت هر قطعه مراجعه کنیم و ابعاد فنی هر قطعه را بدست آوریم پس داریم :



ابعاد فنی یک صفحه کلید 4×4 نیز به صورت زیر است :

Series KNM3 without PCB

Monobloc 3x4 **Monobloc 4x4**



پس اکثر ابعاد فنی محصول مربوط به صفحه کلید خواهد شد و $Segment - 7$ با انجام محاسبه ابعاد فنی محصول چیزی در حدود $16 \times 12\ cm^2$ خواهد شد که البته محصول نهایی ممکن است بزرگ تر باشد زیرا برای گوشه ها ممکن است حجم اضافه در نظر گرفته شود .

چالش‌های محصول : چالش های زیر از جمله چالش هایی که در ابتدا با آن ها روبرو شدیم :

1- وقتی که یک دستور توسط پردازنده اجرا شود طبیعتا خروجی خواهید داشت و باید مشخص کنیم که در اثر این دستور چه نتایجی حاصل می‌شود یعنی در اثر اجرای یک دستور باید مشخص کنیم که مثلا برای رجیستر های داخلی مانند Acc , ... و رجیستر های مربوط به اعمال شرطی مانند $flag\ Register$ چه تغییری خواهیم داشت و مقدار جدید آن ها چه خواهد بود .

2- تعداد پین های محدود میکروکنترلر و تعداد بالای سخت افزارهای جانبی: 8 عدد $Segment - 7$و 4 عدد صفحه کلید 4×4

3- عدم اطمینان از فشرده شدن کلید : مکانیزم استفاده شده برای صفحه کلید می‌بایست $Polling$ باشد (جزییات در پیوست 2)، مکانیزم معمول نمایش روی $Segment - 7$ ها، $Polling$ است. در نتیجه مساله‌ی زمان بندی صفحه کلید سبب ایجاد شک در طراحان نسبت به عملکرد کاملا صحیح صفحه کلید شد. همچنین کاربر (که یک انسان است) ممکن است در فشردن کلید ها در یکی از مراحل طولانی و طاقت فرسای وارد کردن صد ها خط کد به زبان ماشین دچار خطا در تعیین میزان فشار انگشت و مدت زمان لازم برای فشردن کلید بشود و در نتیجه نیاز به تکرار همه ی مراحل از ابتدا بشود.

4- دستورهایی که از آدرس (relative OR absolute) استفاده می کنند به دلیل $Memmory\ Maping$ و تفاوت آدرس کاربر وَ آدرس میکروکنترلر.

5- شرط های زیادی که باید چک بشود! : دو مد کاری و هر مد کاری وضعیت های گوناگون اجرا دارند.

6- تشخیص $Opcode$ و $Operand$ ها در فضای حافظه در مد اجرای تک خط.

7- تعداد خطوط بالای کد و کارگروهی.

راه‌حل های احتمالی برای این چالش‌ها : راه‌حل هایی که برای این چالش‌ها به کار بردیم به صورت زیر است :

1. هنگامی که $Opcode$ موجود در حافظه را به پردازنده می‌دهیم ، پردازنده ابتدا باید تشخیص دهد که این $Opcode$ مربوط به کدام دستور است و آن را تشخیص دهد و با استفاده از $Operand$ ها موجود آن را اجرا می‌کند حال اگر بخواهیم مقدار موجود در رجیستر ها حاوی نتیجه دستور اجرا شده باشند کافی است به میکروکنترلر بفهمانیم که $Opcode$ وارد شده مربوط به کدام دستور است و $Operand$ های مورد نیاز را نیز در اختیار آن قرار دهیم با این کار میکروکنترلر آن دستور را انجام را می دهد و خود میکروکنترلر نتایج را در رجیستر های داخلی خود ذخیره می‌کند اما در حین اینکه تشخیص دهیم که $Opcode$ وارد شده مربوط به کدام دستور و $Operand$ های مورد نیاز را به میکروکنترلر بدهیم ممکن است مقدار رجیسترهای داخلی تغییر کند بنابراین باید چاره‌ای اندیشید تا در اثر اعمال پس زمینه ای داده مربط به رجیسترهای داخلی تغییر نکند و برنامه خروجی صحیح داشته باشد . راه حلی که ما اتخاد کردیم استفاده از حافظه RAM داخلی میکروکنترلر است و در پایان هر عمل مقدار رجیسترهای داخلی را در مکان مشخصی از حافظه RAM داخلی میکروکنترلر ذخیره می‌کنیم و در شروع هر عمل مقدار این رجیستر ها را استخراج می‌کنیم .

2.

2.1. پس از صحبت با آقای شایگانی (دستیار مرتبط) به جای استفاده از 4 صفحه کلید 4×4 از یک صفحه کلید 4×4 استفاده شد.

2.2. استفاده از دو PPI 8255

2.3. ... $time\ domain\ multiplexing\ using$

3.

3.1. خروجی های صفحه کلید را به $4\ pin\ AND\ GATE$ متصل می کنیم. به این صورت کاربر متوجه فشرده شدن کلید می شود.

3.2. هر زمان که سیستم یک ورودی از صفحه کلید دریافت می کند $DELAY\ SUBROUTINE$را فراخوانی می کنیم.

3.3. تغییر مکانیزم نمایش روی نمایشگر ها از $Polling$: با اضافه کردن دومین PPI نیاز به $Polling$ رفع می شود.

4. عملیات حسابی پیش از اجرای این دستور ها. (در قسمت دوم توضیح داده شده است.)

5. رسم فلوچارت دقیق پیش از شروع برنامه نویسی

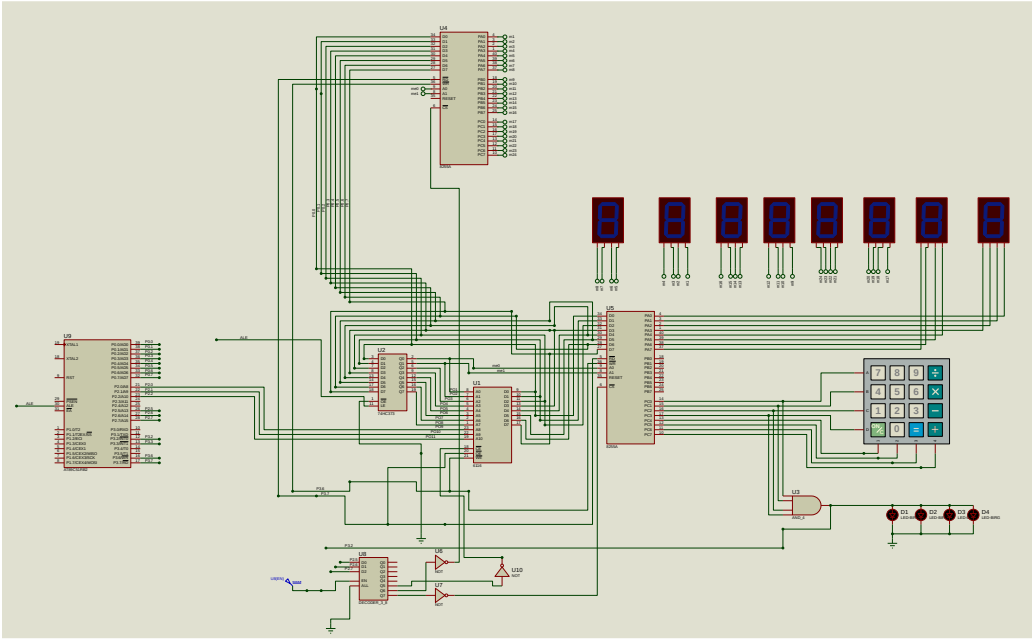
6. آدرس شروع برنامه که توسط کاربر وارد می‌شود اولین $Opcode$ را نشان می‌دهد و از روی این $Opcode$ میتوان پی برد که این دستور چند بایتی است و پس آدرس $Opcode$ بعدی را می‌توانیم بدست آوریم بدین ترتیب میتوان آدرس هر $Opcode$ را برای اجرای خط به خط پیدا کرد خانه های حافظه بین دو $Opcode$ مربوط به $Operand$ ها را دستور است . پس با داشتن این اطلاعات می‌توان کد اسمبلی وارد شده را به صورت خط به خط اجرا کرد .

7.

7.1. نوشتن کد با subroutine های متنوع.

7.2. تخصیص رجیسترهای مجاز برای هر کدام از اعضای گروه پیش اَز شروع پروژه، جدول رجیسترها در پیوست اول آمده است.

شمای کلی محصول : با توجه به توضیحات قبلی شمای کلی به صورت زیر می‌باشد :



****

بخش دوم : الف) دستورات مطرح شده در کلاس و آزمایشگاه به صورت زیر است .

8051/8052 irregular instructions

Opcode	x0	x1	x2	x3	x4
0y	NOP		LJMP addr16	RR A (rotate right)	INC A
1y	JBC bit,offset (jump if bit set with clear)		LCALL addr16	RRC A (rotate right through carry)	DEC A
2y	JB bit,offset (jump if bit set)		RET	RL A (rotate left)	ADD A,#data
3y	JNB bit,offset (jump if bit clear)		RETI	RLC A (rotate left through carry)	ADDC A,#data
4y	JC offset (jump if carry set)		ORL address,A	ORL address,#data	ORL A,#data
5y	JNC offset (jump if carry clear)		ANL address,A	ANL address,#data	ANL A,#data
6y	JZ offset (jump if zero)		XRL address,A	XRL address,#data	XRL A,#data
7y	JNZ offset (jump if non-zero)	AJMP addr11 ,	ORL C,bit	JMP @A+DPTR	MOV A,#data
8y	SJMP offset (short jump)	ACALL addr11	ANL C,bit	MOVC A,@A+PC	DIV AB
9y	MOV DPTR,#data16		MOV bit,C	MOVC A,@A+DPTR	SUBB A,#data
Ay	ORL C,/bit		MOV C,bit	INC DPTR	MUL AB
By	ANL C,/bit		CPL bit	CPL C	CJNE A,#data,offset
Cy	PUSH address		CLR bit	CLR C	SWAP A
Dy	POP address		SETB bit	SETB C	DA A (decimal adjust)
Ey	MOVX A,@DPTR		MOVX A,@R0	MOVX A,@R1	CLR A
Fy	MOVX @DPTR,A		MOVX @R0,A	MOVX @R1,A	CPL A

دستورات بالا از جمله تمامی دستوراتی است که برای میکروکنترلر مطرح شده است . حال باید بررسی کنیم که اگر این دستورات را برای سیستم طراحی در نظر بگیریم این دستورات دچار تغییر خواهند شد و یا خیر ؟

در مجموعه دستورات بالا اگر دستور وارد شده به نوعی با آدرس خانه حافظه بیرونی در مرتبط باشد باید در این دستور تغییر ایجاد کنیم مثلا برای دستور $JMP\ ,JB\ ,JNB\ ,JZ\ ,JNZ$... باید در روند اجرای دستورات تغییر ایجاد کنیم زیرا آدرس حافظه از دید کاربر با آدرس حافظه از دید میکروکنترلر یکسان نیست و دلیل آن این است که حافظه بیرونی را توسط $Memmory-Map$ به میکروکنترلر معرفی کردیم مثلا کاربر خانه شماره 40 را به عنوان مقصد JMP انتخاب می‌کند اما وقتی که میکروکنترلر باید به این آدرس پرش کند اگر مقدار 40 را بر روی باس آدرس خود قرار دهد ممکن هست حتی حافظه بیرونی انتخاب نشود و عدد 40 در محدوده $Memmory-Map$ مربوط به حافظه نباشد و در واقع باید میکروکنترلر به خانه ای اشاره کند که 40 شماره از شروع $Memmory-Map$ مربوط به حافظه خارجی جلوتر باشد . در جدول زیر تمامی دستوراتی که به نوعی با آدرس دهی مرتبط می‌شوند و در سیستم ما دچار تغییر می‌شوند آورده شده است .

JBC	*JNC*	*PUSH*	*LJMP*	*XRL address,A*	*CJNE*
JB	*JZ*	*POP*	*LCALL*	*ORL address,data*	*MOVX A,@R0,1*
JNB	*JNZ*	*AJMP*	*ORL address,A*	*ANL address,data*	*MOVX @R0,1 ,A*
JC	*SJMP*	*ACALL*	*ANL address,A*	*XRL address,data*	*MOVC*

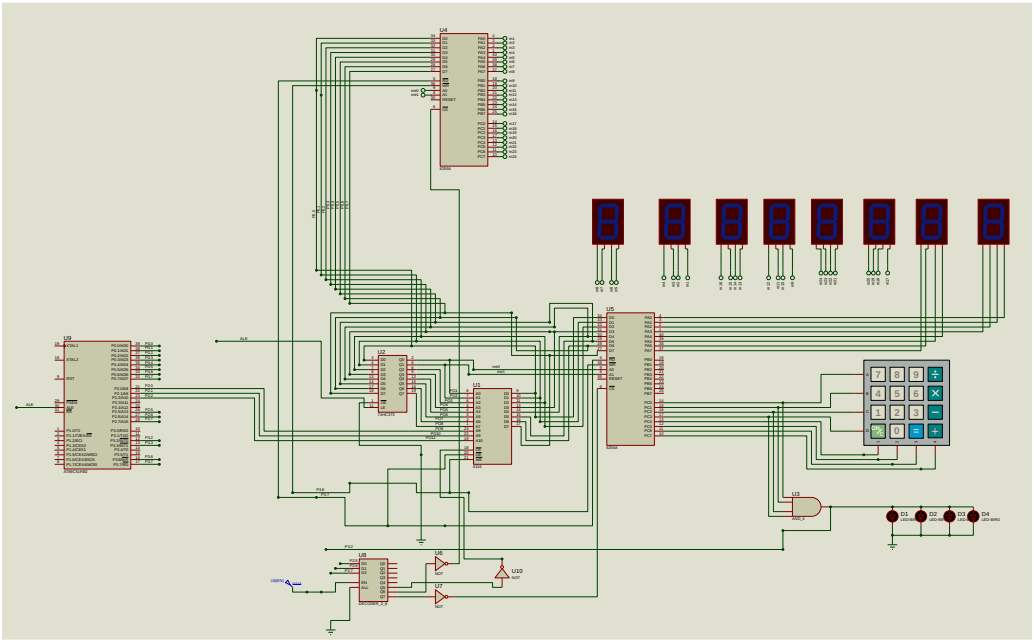
درباره بقیه دستورات که با آدرسی دهی ارتباطی ندارند مانند دستورات حسابی این دستورات دچار تغییری نخواهند شد زیرا شرایط برای آن‌ها تفاوتی نکرده است دلیل تغییر نکردن به طور دقیق تر را میتوان اینگونه توجیح کرد که $Operand$ های این دستورات و همچنین نتیجه این عملیات ها در رجیستر های داخلی میکروکنترلر ذخیره خواهند شد پس تغییری با حالت عادی نخواهند داشت.

تغییری که در دستورات ذکر شده باید اعمال کرد به این صورت است ، وقتی کاربر مقصد و یا آدرس حافظه بیرونی را وارد سیستم کرد و میکروکنترلر قصد داشت که به آن آدرس پرش کند و یا دیتایی را بخواند و عملیات مطلوب را انجام دهد باید آن آدرس را تا حدی تغییر دهد به این صورت که ابتدا آدرس حافظه بیرون که توسط کاربر وارد شده است را با اولین آدرس شروع محدوده آدرس دهی حافظه بیرونی $(Memmory-Map)$ جمع میکند و سپس این مقدار را بر روی باس آدرس قرار می‌دهد و این مقدار قرار گرفته بر روی آدرس باس اکنون به همان خانه حافظه‌ای اشاره می‌کند که مطلوب کاربر است و ادامه عملیات را از سر میگیریم .

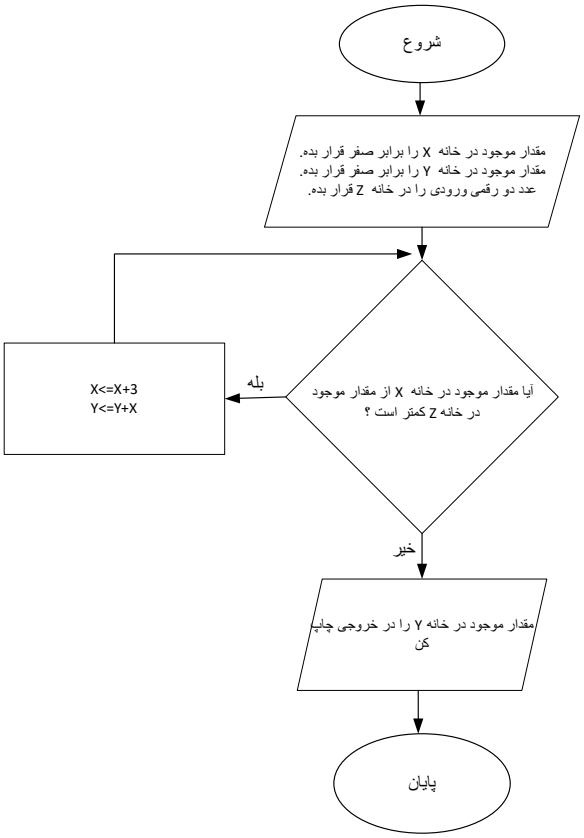
- علاوه بر تغییری که در دستورات بالا ایجاد کردیم باید به یک نکته نیز توجه کنیم و آن این است که دیتا هر خانه حافظه و آدرس آن خانه توسط کاربر وارد می‌شود پس باید عملیاتی مشابه با عملیات بالا برای آدرس ورودی نیز انجام دهیم به این صورت آدرس ورودی کاربر را با اولین مقدار آدرس دهی حافظه بیرونی جمع می‌کنیم و سپس این مقدار را بر روی باس آدرس قرار می‌دهیم و این آدرس به همان خانه‌ای اشاره می‌کند که مطلوب کاربر است حال دیتا وارد شده از سوی کاربر بر روی باس دیتا قرار می‌دهیم و سپس سیگنال $Write$ را فعال می‌کنیم تا این دیتا ورودی در خانه حافظه مطلوب قرار گیرد .
- نکته دیگری که کاربر باید به آن دقت کند این است در مواقعی که از دستوراتی که پرشی هستند استفاده میکند مقدار آدرس خانه مقصد را به گونه ای تنظیم کند که این آدرس در محدوده آدرس خانه های حافظه باشد . مثلا آدرس به گونه ای نباشد که هنگامی که سیستم با آن کار می‌کند آدرس قرار گرفته شده بر روی باس آدرس خارج از $Memmory-Map$ مربوط به حافظه خارجی قرار گیرد در این صورت سیستم دچار مشکل خواهد شد و نکته دیگر این است حتما خانه های مقصد پرش را با آپکود مناسب پر کند .

- تغییرکه گفته شد در کد اسمبلی اعمال شده است که فایل کد اسمبلی پیوست شده است .

بخش سوم : مینیمم سیستم ما در پروتئوس به صورت زیر می‌باشد :



حال میخواهیم به مینیمم سیستمی که طراحی کردیم برنامه‌ای بدهیم تا جمع مضارب عدد 3 را تا یک عدد مشخص را حساب کند . در صورت پروژه ذکر نشده است که عدد حاصل را باید در خروجی نشان دهیم اما ما در اینجا فرض می‌کنیم که باید عدد وارد شده را باید در خروجی نشان دهیم . کد اسمبلی این برنامه ضمیمه شده است (همچنین جدول مربوط به $Opcode, Operand$ دستورات ورودی نیز در یک فایل جدا ضمیمه شده است .) و در اینجا تنها فلوچارت مربوط به این کار را رسم می‌کنیم و خروجی را نشان می‌دهیم ، فلوچارت کلی این برنامه به صورت زیر است :



در فلوچارت بالا آدرس‌های X, Y, Z باید توسط کاربر وارد شود و همچنین مقدار عدد دو رقمی نیز توسط کاربر مشخص می‌شود . حال این برنامه را بر روی مینیمم سیستم خود پیاده می‌کنیم و آن را در دو $Mode$ که قبلاً ذکر کردیم اجرا می‌کنیم . کد اسمبلی مربوطه پیوست شده است .

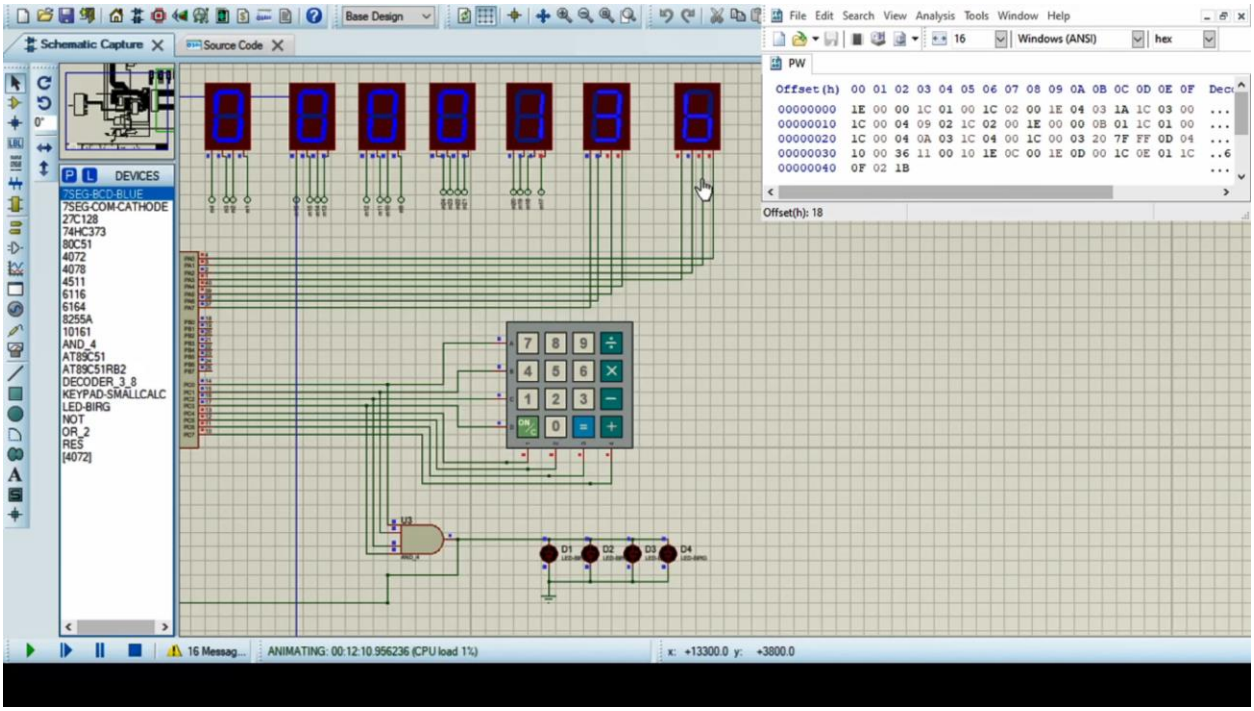
$Mode\ 1 : Run\ Totally\ Assemble\ Code$

در این حالت عدد ورودی را مطابق شکل زیر برابر با 44 قرار می‌دهیم پس برنامه ورودی باید جمع دنباله زیر را حساب کند :

$$3, 6, 9, 12, 15, \dots, 42$$

جمع دنباله بالا برابر با 315 می‌شود . حال اگر از سیستم کمک بگیریم داریم : ورودی را برابر با $2C_{HEX}$ قرار میدهیم .

حال با اجرا کردن برنامه خروجی به صورت زیر می‌باشد :



که مقدار $13b_{HEX}$ برابر با 315 است پس خروجی درست می‌باشد .

$Mode\ 2 : Run\ Single\ Step\ Assemble\ Code$

وقتی که مد اجرا را بر روی این حالت قرار دهیم . مقدار موجود در حافظه‌ای که خروجی را در بر دارد به صورت مرحله به مرحله آپدیت می‌شود . تمامی مراحل اعم از ورودی دادن و خواندن مقدار رجیستر ها د هر مرحله در فیلمی به همراه فایل ها ضمیمه شده است و به این دلیل از آوردن خروجی برای این بخش خودداری میکنیم و به فیلم موجود در پوشه $Others$ ارجاع می‌دهیم .

بخش چهارم : در کدینگ و پیاده‌سازی با چالش‌های زیادی روبرو شدیم در زیر چند نمونه از این چالش ها را ذکر کردیم :

1- در پیاده سازی بخش مربوط به نمایش‌گر با مشکل $Overlap$ بین $Segment-7$ مواجه شدیم به این صورت که برای نمایش یک رشته بر روی 8 عدد $Segment-7$ که داشتیم در یک لحظه فقط یکی از این 8 عدد $Segment-7$ را روشن کرده و مقدار مطلوب به آن را بر روی آن قرار داده و سپس آن را خاموش می‌کنیم و چنین کار مشابهی را برای دیگر $Segment-7$ نیز تکرار می‌کنیم و این کار را بسیار سریع انجام می‌دهیم به گونه‌ای که چشم متوجه خاموش و روشن $Segment-7$ ها نشود و چیزی که در نهایت میبینیم 8 عدد $Segment-7$ روشن که رشته مورد نظر ما را نمایش می‌دهد . اما در پیاده سازی این روش به این مشکل خوردیم که مقدار هر $Segment-7$ ممکن است با مقدار $Segment-7$ دیگر همپوشانی کند مثلا میخواستم عدد 7 را برای $Segment-7$ شماره 1 و عدد 3 را بر روی $Segment-7$ شماره 2 نمایش دهیم اما در اثر همپوشانی مقدار ها بر روی هر دو $Segment-7$ عدد 8 چاپ می‌شد با مرور کد متوجه شدیم دلیل این همپوشانی عدم هماهنگی بین تعویض مقدار بر روی $Segment-7$ و تعویض اینکه کدام $Segment-7$ روشن باشد رخ می‌داد و برای مثال ذکر شده هر دو مقدار 7 و 3 برای هر $Segment-7$ در نظر گرفته می‌شد که این باعث نمایش عدد 8 می‌شد . با جدا کردن بخش مشترک بین $Segment-7$ و استفاده از دو عدد PPI این مشکل حل شد .

2- عدم آشنایی کامل با دستورات پیش از شروع پروژه همچنین خطاهای عددی در وارد کردن مقادیر در بعضی از جاها (برای مثال خطا در وارد کردنِ محتوای صحیح $CWR\ of\ 8255\ PPI$) که هر دو این موارد به وسیله مینیمم سیستم فعلی قابل رفع کردن است. تنها کافی است از یکی از نمایشگرها (برای نمایش محتوای خانه حافظه ی خواسته شده) و دستور $here: jmp\ here$ برای ایجاد $break\ point$ در محل های مورد شک برنامه استفاده کرد. لیکن تنها مساله این است که صبر ایوب می خواهد!

3- مشکل در استفاده از حافظه بیرونی برای اجرای برنامه کاربر داشتیم به این صورت که در حین استفاده از این حافظه نباید محتویات خانه ها تغییر کند و اطلاعات ورودی دچار تغییر شود و مثلا در هنگامی که با نمایشگر و یا صفحه کلید کار می‌کنیم مقدارهای موجود در حافظه دچار تغییر نشود .

4- تعریف کردن $function$ های $SETB\ ,CLR$ کاری سخت بود و نتوانستیم عمل $Indirect\ Adsressing$ را برای بیت ها پیاده‌سازی کنیم و حجم کد برای پیاده‌سازی این کار زیاد می‌شد در عوض دو بایت $addressable$ در اختیار کاربر قرار دادیم تا بتواند عمل مورد نظر خود را انجام دهد .

5- مسئله دیگر پیاده سازی $Interrupt$ بود برای اینکار دو دستور $Input\ ,Output$ برای کاربر در نظر گرفتیم که بتواند کار خود را انجام دهد .

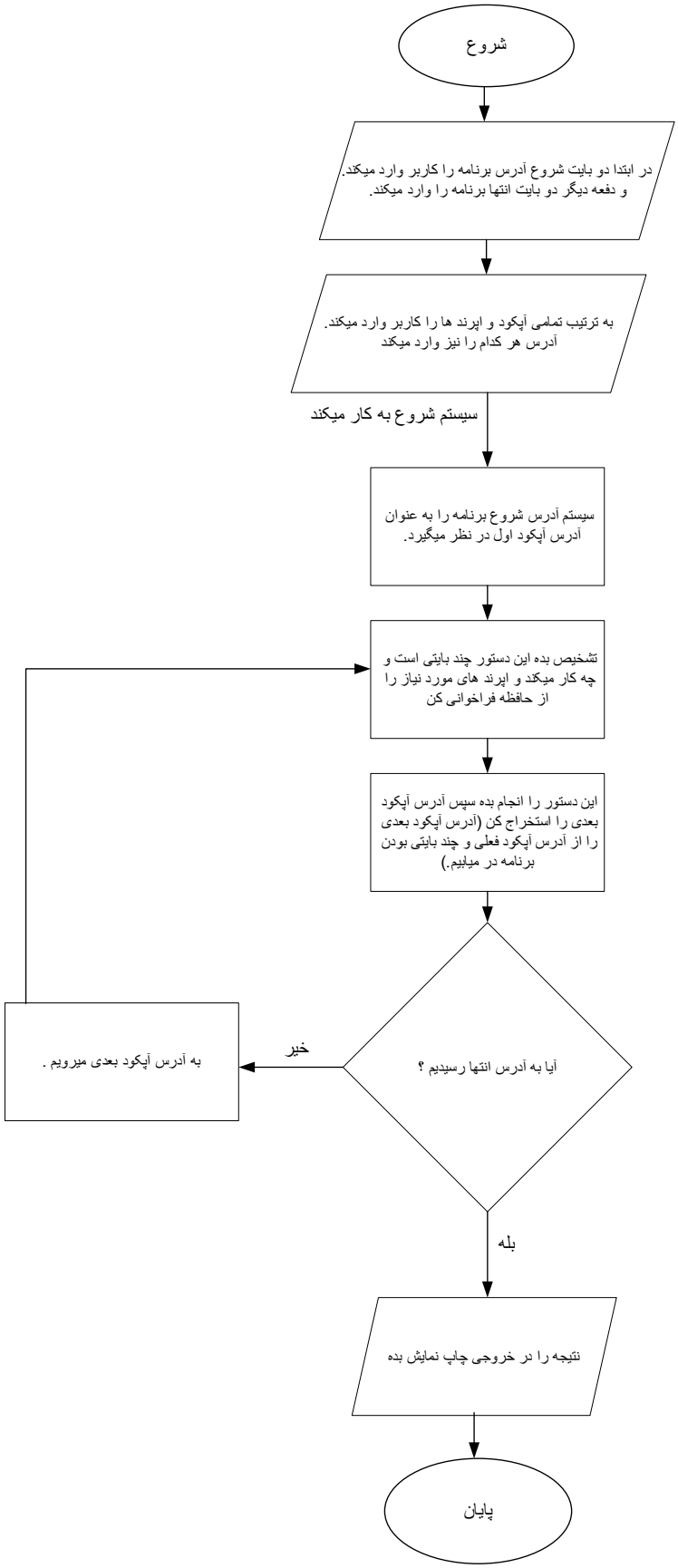
پیوست :

الف) کد اسمبلی بخش سوم به صورت زیر است :

```
mov A,0                 30/0/0
mov R1,A                28/1/0
mov R2,A                28/2/0
mov R4,3                30/4/3
Input A                 26
mov R3,A                28/3/0
E2:
mov A,R4                28/0/4
add A,R2                9/2
mov R2,A                28/2/0
mov A,0                 30/0/0
addc A,R1               11/1
mov R1,A                28/1/0
mov A,R4                28/0/4
add A,3                 10/3
mov R4,A                28/4/0
mov A,R3                28/0/3
clr c                   32/7Fh/FFh
subb R4                 13/4
jc E1                   16/0/54
jmp E2                  17/0/16
```

توضیحات نرم افزاری: فلوچارت صفحه بعد توضیح روش به کار رفته در پیاده‌سازی سیستم است . توضیحات بیشتر را برای هر قسمت با کامنت گذاری در کد مشخص کرده‌ایم .

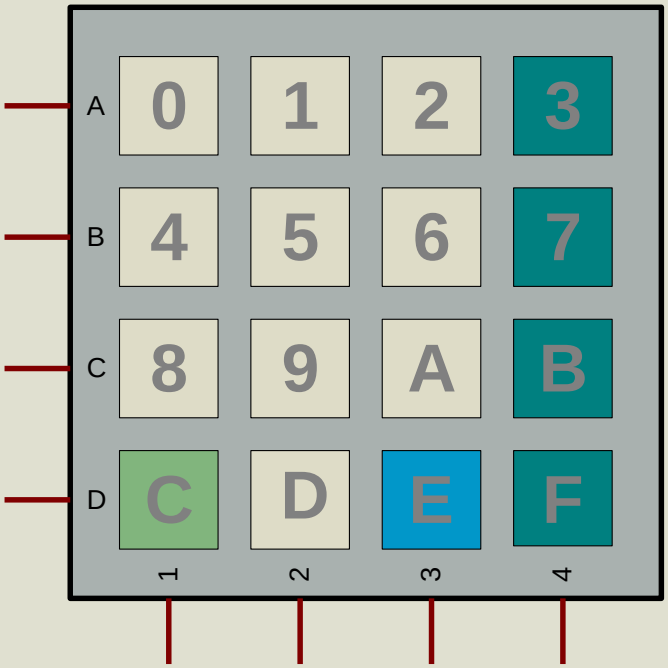
- برای پیاده سازی هر دستور یک $Subroutine$ تعریف کردیم و وقتی کاربر یک $Opcode$ را وارد می‌کند در حافظه ROM در خانه حافظه مربوط به $Opcode$ دستور JMP قرار دارد که مقصد آن $Subroutine$ مورد نظر می‌باشد .
- در پیاده‌سازی سیستم در استفاده از صفحه کلید از روش تاخیر برای صحت سنجی کلید ورودی استفاده کردیم . سیستم پس از آمدن کلید چک می‌کند که آیا پس از گذشت زمان $Bounce$ کلید همچنان فعال است یا خیر ؟
- آدرس رجیستر ها و بیت ها مورد نیاز برای استفاده از کاربر در جدول پیوست آمده است .


شروع
در ابتدا دو بایت شروع آدرس برنامه را کاربر وارد میکند.
و دفعه دیگر دو بایت انتها برنامه را وارد میکند.
به ترتیب تمامی آپکود و اپرند ها را کاربر وارد میکند.
آدرس هر کدام را نیز وارد میکند
سیستم شروع به کار میکند
سیستم آدرس شروع برنامه را به عنوان
آدرس آپکود اول در نظر میگیرد.
تشخیص بده این دستور چند بایتی است و
چه کار میکند و اپرند های مورد نیاز را
از حافظه فراخوانی کن
این دستور را انجام بده سپس آدرس آپکود
بعدی را استخراج کن (آدرس آپکود بعدی
را از آدرس آپکود فعلی و چند بایتی بودن
برنامه در میابیم.)
آیا به آدرس انتها رسیدیم ؟
خیر
به آدرس آپکود بعدی میرویم .
بله
نتیجه را در خروجی چاپ نمایش بده
پایان

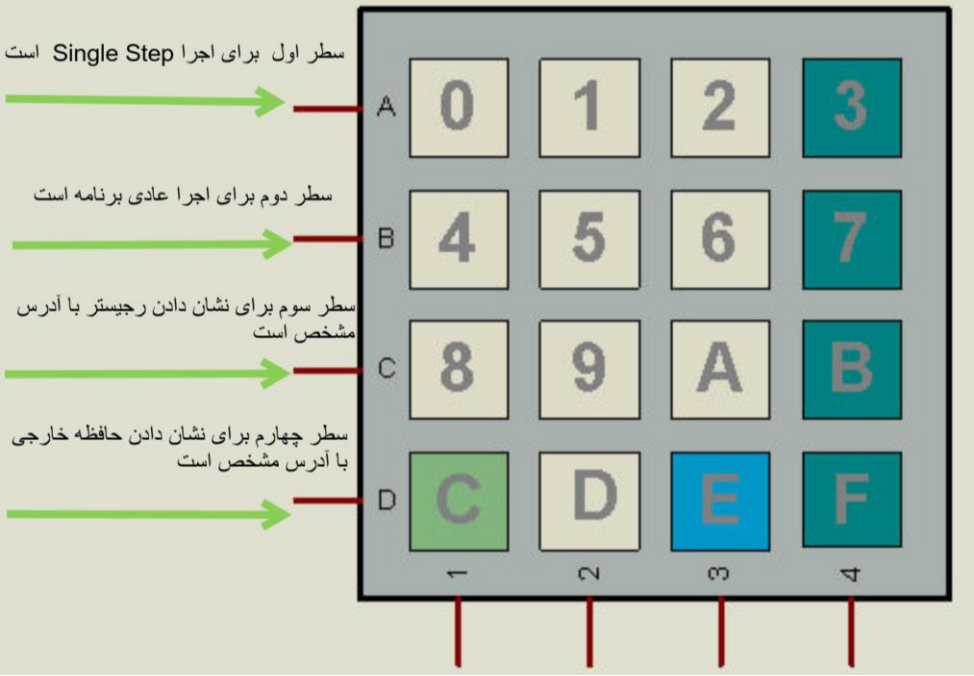
توضیحات سخت‌افزاری :

- به علت تعداد پین های زیاد وسایل جانبی و تعداد پین‌های محدود 8051 از روش $Memmory - Map$ و با استفاده از دو PPI توانستیم که وسایل جانبی (8 عدد $Segment - 7$) و صفحه کلید را به 8051 معرفی کنیم .و دلیل انتخاب دو عدد PPI نیز همین است .
- به دلیل اینکه در نرم‌افزار $Proteus$ صفحه کلید 64 تایی نداشتیم با راهنمایی گرفتن از آقای شایگانی از یک صفحه کلید 16 تایی استفاده کردیم و به دلیل کاهش تعداد کلید ها فرآیند دادن ورودی دادن توسط کاربر در تعداد مراحل بیشتری انجام خواهد شد .
- به دلیل اینکه کاربر از فشار دادن کلید و آگاه شدن از اینکه سیستم این کلید را خوانده است متوجه شود از 4 عدد LED کمکی استفاده کرده‌ایم .

کار با دستگاه : حال روش ورودی دادن به سیستم را توضیح میدهیم : در اولین قدم باید کار با صفحه کلید را یاد بگیریم صفحه کلید مورد نظر ما به صورت زیر است (البته این صفحه کلید در شبیه‌سازی به این صورت نیست.) از اعداد مشخص شده بر روی آن برای وارد کردن آدرس استفاده میکنیم . (با هر بار فشار دادن یک عدد هگز وارد سیستم می‌شود) .



کلید اجرای برنامه به صورت زیر است :



برای وارد کردن برنامه ابتدا دو عدد هگز به عنوان آدرس شروع برنامه و دو عدد هگز به عنوان آدرس پایان برنامه وارد کردیم سپس $Opcode$ و $Operand$ هر دستور را در خانه حافظه وارد کرد و هنگامی که این عمل به پایان رسید با استفاده از کلید های سطر یک یا دو برای اجرای برنامه استفاده می‌کنیم . در هنگام اجرای برنامه به صورت خط به خط با استفاده از کلید های سطر سه و یا چهارم می‌توان مقدار موجود در رجیستر های داخلی و یا حافظه خارجی را با وارد کردن آدرس آن‌ها مشاهده کنیم .

برای کار با دستگاه نیاز داریم تا معادل هگز دستورات را داشته باشیم در جدول زیر این دستورات آورده شده است :

$Command$	$OpCode$	$Operand$	$Operand$	$Operand$	$Bytes$
AND	0	Reg			2
$ANDI$	1	$Data$			2
OR	2	Reg			2
ORI	3	$Data$			2
NOT	4	Reg			2
RL	5	Reg			2
RLC	6	Reg			2
RR	7	Reg			2
RRC	8	Reg			2
ADD	9	Reg			2
$ADDI$	10	$Data$			2
$ADDC$	11	Reg			2

$ADDCI$	12	$Data$			2
$SUBB$	13	Reg			2
$SUBBI$	14	$Data$			2
JZ	15	ADD_H	ADD_L		3
JC	16	ADD_H	ADD_L		3
$LJUMP$	17	ADD_H	ADD_L		3
$LCALL$	18	ADD_H	ADD_L		3
INT	19	–	–	–	–
RET	20				1
$RETI$	21				1
$MASKINT$	22				1
$UNMASKINT$	23				1
$MOVRM$	24	Reg	ADD_H	ADD_L	4
$MOVMR$	25	ADD_H	ADD_L	Reg	4
$INPUT$	26				1
$OUTPUT$	27				1
$MOVRR$	28	Reg	Reg		3
NOP	29				1
$MovI$	30	Reg	$Data$		3
$SBIT$	31	Imm	Imm		3
CLR	32	Imm	Imm		3
CPR	33	Imm	Imm		3

جدول آدرس رجیستر ها و بیت های مورد نیاز کاربر به صورت زیر است :

$R0$	$00H$	$Carry$	$00H$
$R1$	$01H$	$B1$	$01H$
$R2$	$02H$	$B2$	$02H$
$R3$	$03H$	$B3$	$03H$
$R4$	$04H$	$B4$	$04H$
$R5$	$05H$	$B5$	$05H$
$R6$	$06H$	$B6$	$06H$
$R7$	$07H$	$B7$	$07H$
$R8$	$08H$	$B8$	$08H$
$R9$	$09H$	$B9$	$09H$
$R10$	$0AH$	$B10$	$0AH$
$R11$	$0BH$	$B11$	$0BH$
$R12$	$0CH$	$B12$	$0CH$
$R13$	$0DH$	$B13$	$0DH$
$R14$	$0EH$	$B14$	$0EH$

$R15$	$0FH$	$B15$	$0FH$

ستون $Opcode$ معادل $Decimal$ را در بر دارد.

با استفاده از جداول بالا میتوان کد اسمبلی دلخواه را به سیستم داد و سپس آن را اجرا کرد اما فرآیند وارد کردن کد اسمبلی بسیار زمان‌بر است .

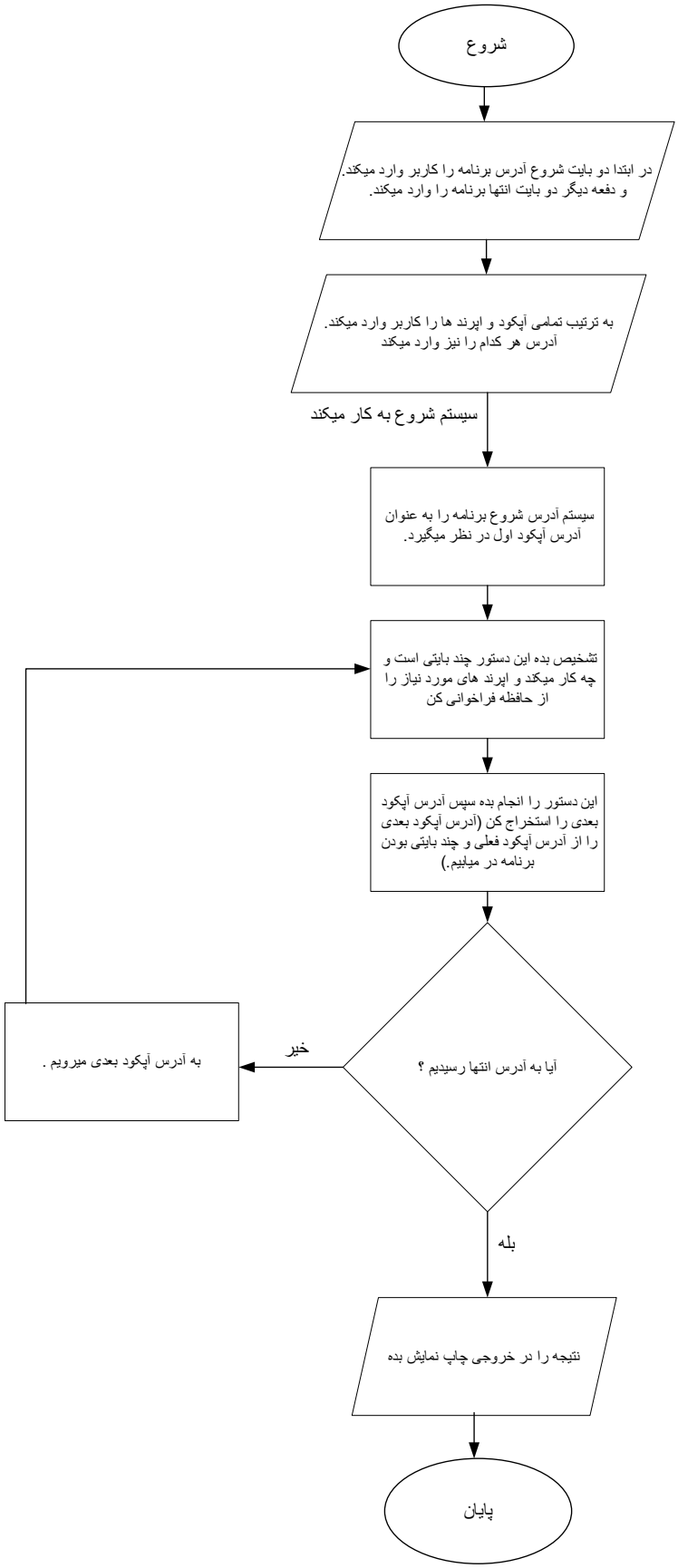
پیوست :

الف) کد اسمبلی بخش سوم به صورت زیر است :

```
mov A,0                 30/0/0
mov R1,A                28/1/0
mov R2,A                28/2/0
mov R4,3                30/4/3
Input A                 26
mov R3,A                28/3/0
E2:
mov A,R4                28/0/4
add A,R2                9/2
mov R2,A                28/2/0
mov A,0                 30/0/0
addc A,R1               11/1
mov R1,A                28/1/0
mov A,R4                28/0/4
add A,3                 10/3
mov R4,A                28/4/0
mov A,R3                28/0/3
clr c                   32/7Fh/FFh
subb R4                 13/4
jc E1                   16/0/54
jmp E2                  17/0/16
```

توضیحات نرم افزاری: فلوچارت صفحه بعد توضیح روش به کار رفته در پیاده‌سازی سیستم است . توضیحات بیشتر را برای هر قسمت با کامنت گذاری در کد مشخص کرده‌ایم .

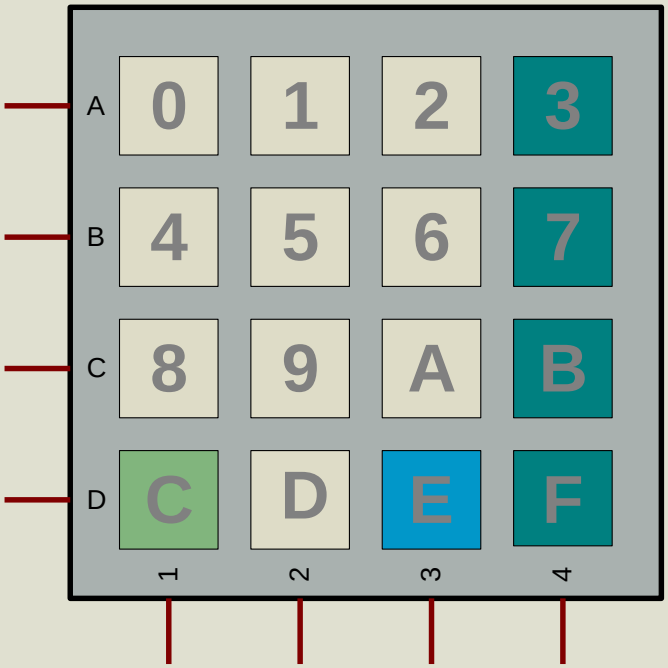
- برای پیاده سازی هر دستور یک $Subroutine$ تعریف کردیم و وقتی کاربر یک $Opcode$ را وارد می‌کند در حافظه ROM در خانه حافظه مربوط به $Opcode$ دستور JMP قرار دارد که مقصد آن $Subroutine$ مورد نظر می‌باشد .
- در پیاده‌سازی سیستم در استفاده از صفحه کلید از روش تاخیر برای صحت سنجی کلید ورودی استفاده کردیم . سیستم پس از آمدن کلید چک می‌کند که آیا پس از گذشت زمان $Bounce$ کلید همچنان فعال است یا خیر ؟
- آدرس رجیستر ها و بیت ها مورد نیاز برای استفاده از کاربر در جدول پیوست آمده است .


شروع
در ابتدا دو بایت شروع آدرس برنامه را کاربر وارد میکند.
و دفعه دیگر دو بایت انتها برنامه را وارد میکند.
به ترتیب تمامی آپکود و اپرند ها را کاربر وارد میکند.
آدرس هر کدام را نیز وارد میکند
سیستم شروع به کار میکند
سیستم آدرس شروع برنامه را به عنوان
آدرس آپکود اول در نظر میگیرد.
تشخیص بده این دستور چند بایتی است و
چه کار میکند و اپرند های مورد نیاز را
از حافظه فراخوانی کن
این دستور را انجام بده سپس آدرس آپکود
بعدی را استخراج کن (آدرس آپکود بعدی
را از آدرس آپکود فعلی و چند بایتی بودن
برنامه در میابیم.)
آیا به آدرس انتها رسیدیم ؟
خیر
به آدرس آپکود بعدی میرویم .
بله
نتیجه را در خروجی چاپ نمایش بده
پایان

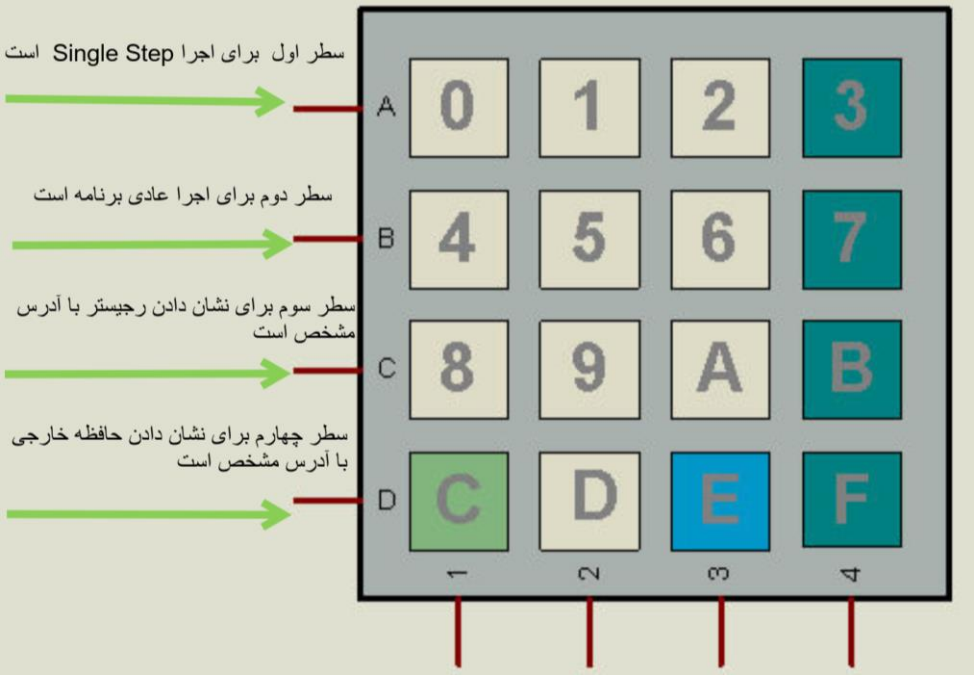
توضیحات سخت‌افزاری :

- به علت تعداد پین های زیاد وسایل جانبی و تعداد پین‌های محدود 8051 از روش $Memmory - Map$ و با استفاده از دو PPI توانستیم که وسایل جانبی (8 عدد $Segment - 7$) و صفحه کلید را به 8051 معرفی کنیم .و دلیل انتخاب دو عدد PPI نیز همین است .
- به دلیل اینکه در نرم‌افزار $Proteus$ صفحه کلید 64 تایی نداشتیم با راهنمایی گرفتن از آقای شایگانی از یک صفحه کلید 16 تایی استفاده کردیم و به دلیل کاهش تعداد کلید ها فرآیند دادن ورودی دادن توسط کاربر در تعداد مراحل بیشتری انجام خواهد شد .
- به دلیل اینکه کاربر از فشار دادن کلید و آگاه شدن از اینکه سیستم این کلید را خوانده است متوجه شود از 4 عدد LED کمکی استفاده کرده‌ایم .

کار با دستگاه : حال روش ورودی دادن به سیستم را توضیح میدهیم : در اولین قدم باید کار با صفحه کلید را یاد بگیریم صفحه کلید مورد نظر ما به صورت زیر است (البته این صفحه کلید در شبیه‌سازی به این صورت نیست.) از اعداد مشخص شده بر روی آن برای وارد کردن آدرس استفاده میکنیم . (با هر بار فشار دادن یک عدد هگز وارد سیستم می‌شود) .



کلید اجرای برنامه به صورت زیر است :



برای وارد کردن برنامه ابتدا دو عدد هگز به عنوان آدرس شروع برنامه و دو عدد هگز به عنوان آدرس پایان برنامه وارد کردیم سپس $Opcode$ و $Operand$ هر دستور را در خانه حافظه وارد کرد و هنگامی که این عمل به پایان رسید با استفاده از کلید های سطر یک یا دو برای اجرای برنامه استفاده می‌کنیم . در هنگام اجرای برنامه به صورت خط به خط با استفاده از کلید های سطر سه و یا چهارم می‌توان مقدار موجود در رجیستر های داخلی و یا حافظه خارجی را با وارد کردن آدرس آن‌ها مشاهده کنیم .

برای کار با دستگاه نیاز داریم تا معادل هگز دستورات را داشته باشیم در جدول زیر این دستورات آورده شده است :

$Command$	$OpCode$	$Operand$	$Operand$	$Operand$	$Bytes$
AND	0	Reg			2
$ANDI$	1	$Data$			2
OR	2	Reg			2
ORI	3	$Data$			2
NOT	4	Reg			2
RL	5	Reg			2
RLC	6	Reg			2
RR	7	Reg			2
RRC	8	Reg			2
ADD	9	Reg			2
$ADDI$	10	$Data$			2
$ADDC$	11	Reg			2

$ADDCI$	12	$Data$			2
$SUBB$	13	Reg			2
$SUBBI$	14	$Data$			2
JZ	15	ADD_H	ADD_L		3
JC	16	ADD_H	ADD_L		3
$LJUMP$	17	ADD_H	ADD_L		3
$LCALL$	18	ADD_H	ADD_L		3
INT	19	–	–	–	–
RET	20				1
$RETI$	21				1
$MASKINT$	22				1
$UNMASKINT$	23				1
$MOVRM$	24	Reg	ADD_H	ADD_L	4
$MOVMR$	25	ADD_H	ADD_L	Reg	4
$INPUT$	26				1
$OUTPUT$	27				1
$MOVRR$	28	Reg	Reg		3
NOP	29				1
$MovI$	30	Reg	$Data$		3
$SBIT$	31	Imm	Imm		3
CLR	32	Imm	Imm		3
CPR	33	Imm	Imm		3

جدول آدرس رجیستر ها و بیت های مورد نیاز کاربر به صورت زیر است :

$R0$	$00H$	$Carry$	$00H$
$R1$	$01H$	$B1$	$01H$
$R2$	$02H$	$B2$	$02H$
$R3$	$03H$	$B3$	$03H$
$R4$	$04H$	$B4$	$04H$
$R5$	$05H$	$B5$	$05H$
$R6$	$06H$	$B6$	$06H$
$R7$	$07H$	$B7$	$07H$
$R8$	$08H$	$B8$	$08H$
$R9$	$09H$	$B9$	$09H$
$R10$	$0AH$	$B10$	$0AH$
$R11$	$0BH$	$B11$	$0BH$
$R12$	$0CH$	$B12$	$0CH$
$R13$	$0DH$	$B13$	$0DH$
$R14$	$0EH$	$B14$	$0EH$

$0FH$	$B15$	$0FH$	$R15$

- ستون $Opcode$ معادل $Decimal$ را در بر دارد.

دستورهای $Input\ ,\ Output$ که در جدول مشخص است اعمال زیر را انجام می‌دهند:

- دستور $Input$: داده ورودی را در رجیستر $R0$ قرار می‌دهد .
- دستور $Output$: داده خروجی را در خروجی نمایش می‌دهد. (داده خروجی باید در مکان های $0CH - 0DH - 0EH - 0FH$ قرار گرفته باشد .)

با استفاده از جداول بالا میتوان کد اسمبلی دلخواه را به سیستم داد و سپس آن را اجرا کرد اما فرآیند وارد کردن کد اسمبلی بسیار زمان‌بر است .