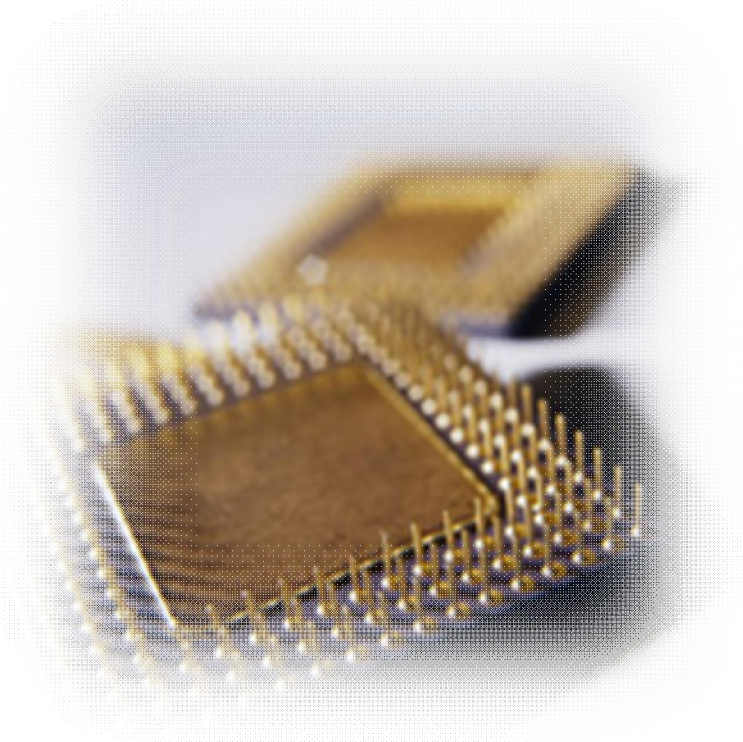
به نام خدا

پروژه طراحی مدار CPU



طراح: سید مصطفی رضاتوفیق

استاد: دکتر کامرانی

تیرماه 1386

فهرست

مقدمه

پروژه حاضر را برای درس معماری کامپیوتر و بر اساس کتاب مانو انجام داده‌ام. زیبایی موضوع باعث شد تا با وجود مشغله زیاد، وقت زیادی روی آن صرف کنم.

اهم کارهای انجام شده شامل تعیین توابع کنترل CPU، طراحی مدار CPU، طراحی یک اسمبلر تک‌گذره، و در نهایت طراحی و برنامه‌نویسی یک ماشین حساب با CPU طراحی شده بوده است.

برای دانلود نرم‌افزار و دریافت آخرین نسخه این سند به سایت بنده mrtofigh.googlepages.com یا مستقیما به آدرس mrtofigh.googlepages.com/CPUDesign مراجعه کنید.

در پایان لازم می‌دانم از استاد محترم آقای دکتر کامرانی به دلیل تسلط ایشان و تدریس زیبای درس تشکر کنم.

سید مصطفی رضاتوفیق

تیرماه 1386

کامپیوتر پایه[1]

سازمان کامپیوتر به وسیله ثبات های داخلی آن، زمانبندی و ساختار کنترل، و مجموعه دستوراتی که به کار می برد تعریف می گردد. در زیرعنوان های بعد هر یک از این قسمت ها تشریح می شود.

ثبات ها

تنها تغییر ثبات‌های CPU مورد بحث نسبت به طرح کتاب، اندازه ثبات‌های INPR و OUTR است. توضیح بیشتر در قسمت طراحی مدار خواهد آمد.

نام ثبات	تعداد بیت	سمبل ثبات
ثبات داده	16	DR
ثبات آدرس	12	AR
انباره	16	AC
ثبات دستورالعمل	16	IR
شمارنده برنامه	12	PC
ثبات موقت	16	TR
ثبات ورودی	16	INPR
ثبات خروجی	16	OUTR

قالب دستورالعمل

سه قالب دستورالعمل در کامپیوتر پایه وجود دارد. این قالب‌ها 16 بیتی هستند.

- دستورالعمل‌های حافظه‌ای

15	14 ... 12	11 ... 0
I	کد عمل	آدرس

- دستورالعمل‌های ثباتی

15	14	13	12	11 ... 0
0	1	1	1	عمل ثباتی

- دستورالعمل‌های ورودی-خروجی

15	14	13	12	11 ... 0
1	1	1	1	عمل I/O

[1] Basic Computer

دستورالعمل‌ها

جدول زیر مجموعه دستورالعمل‌های کامپیوتر پایه را نشان می‌دهد. دستور NOP با کد F020 به مجموعه دستورات CPU اضافه شده است. این دستور هیچ عملی انجام نمی‌دهد.

سمبل	کد شانزده شانزدهی $I = 0$	$I = 1$	شرح
AND	0xxx	8xxx	AND کردن کلمه حافظه با AC
ADD	1xxx	9xxx	جمع کردن کلمه حافظه با AC
LDA	2xxx	Axxx	بارکردن کلمه حافظه در AC
STA	3xxx	Bxxx	ذخیره محتوای AC در حافظه
BUN	4xxx	Cxxx	انشعاب نامشروط
BSA	5xxx	Dxxx	انشعاب و ضبط آدرس بازگشت
ISZ	6xxx	Exxx	افزایش و گذر در صورت نتیجه صفر
CLA	7800		پاک کردن AC
CLE	7400		پاک کردن E
CMA	7200		متمم کردن AC
CME	7100		متمم کردن E
CIR	7080		چرخش AC و E به راست
CIL	7040		چرخش AC و E به چپ
INC	7020		افزایش AC
SPA	7010		گذر از دستور بعدی اگر AC مثبت باشد
SNA	7008		گذر از دستور بعدی اگر AC منفی باشد
SZA	7004		گذر از دستور بعدی اگر AC صفر باشد
SZE	7002		گذر از دستور بعدی اگر E صفر باشد
HLT	7001		توقف کامپیوتر
INP	F800		دریافت کاراکتر و انتقال آن به AC
OUT	F400		برداشتن کاراکتر از AC و انتقال آن به خروجی
SKI	F200		گذر مبتنی بر پرچم ورودی
SKO	F100		گذر مبتنی بر پرچم خروجی
ION	F080		فعال‌کردن وقفه‌ها
IOF	F040		غیرفعال کردن وقفه‌ها

زمانبندی و کنترل

زمانبندی همه ثبات‌ها در کامپیوتر پایه بوسیله یک مولد پالس ساعت اصلی (بخش طراحی مدار) کنترل می‌شود. پالس‌های ساعت حالت هیچ ثباتی را عوض نمی کنند مگر اینکه ثبات توسط سیگنال کنترل فعال شود. سیگنال‌های کنترل در واحد کنترل تولید می‌شوند. مشخصات سیگنال‌های کنترل از لیست عبارات انتقال ثبات جدول بعد به دست می آید.

قبل از ادامه مثالی از روش به دست آوردن ورودی‌های کنترل ثبات‌ها را ذکر می‌کنم. برای تعیین ورودی‌های کنترل AR، همه عباراتی که محتوی AR را تغییر می‌دهند لیست می کنیم:

R'T0: AR←PC
R'T2: AR← IR(0-11)
D7'IT3: AR← M[AR]
RT0: AR← 0
D5T4: AR← AR + 1

ورودی‌های کنترل ثبات‌ها از جمله AR، LD (بارشدن)، INR (افزایش) و CLR (پاک کردن) هستند:

LD (AR) = R'T0 + R'T2 + D7'IT3
CLR (AR) = RT0
INR (AR) = D5T4

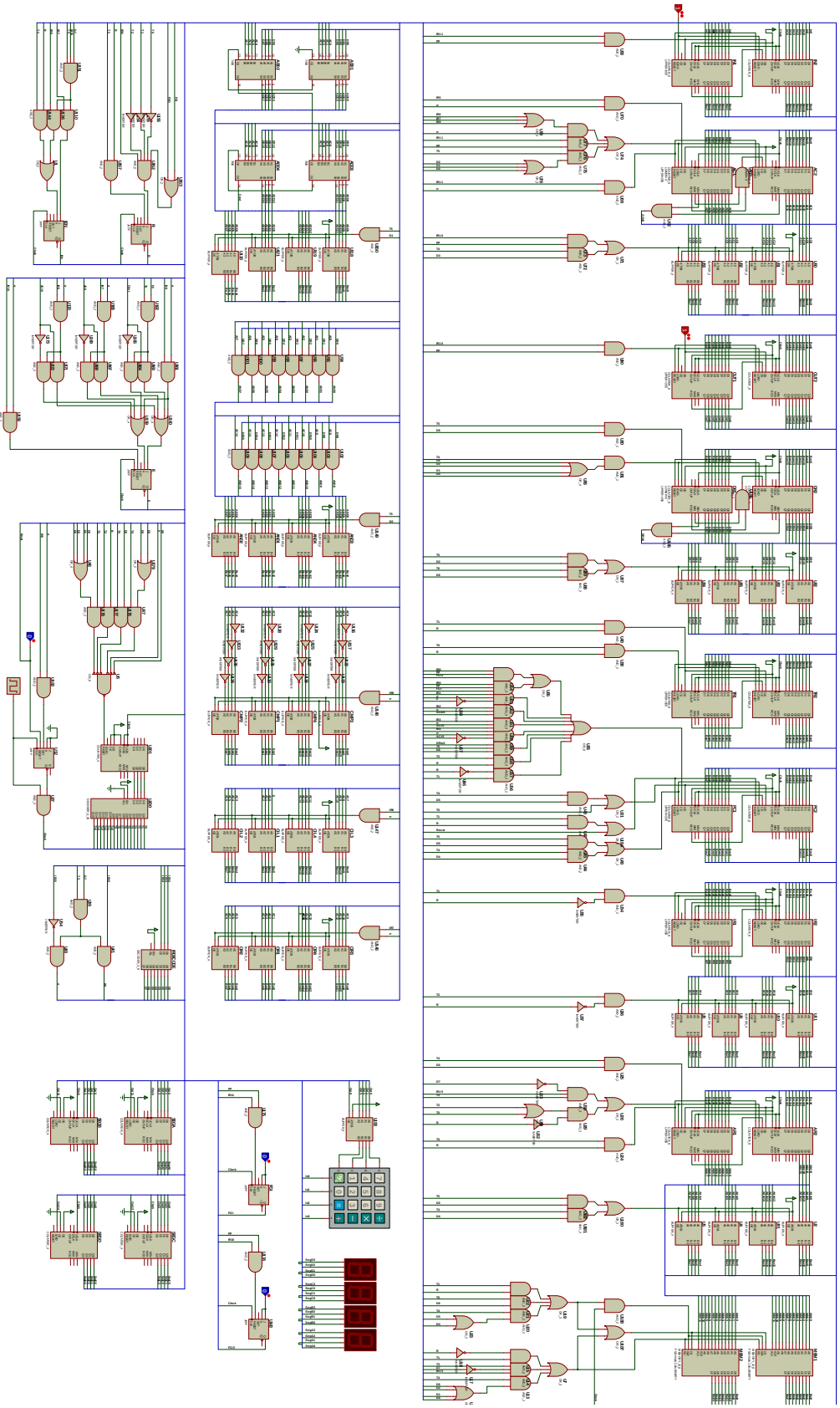
برداشت	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR],\ PC \leftarrow PC + 1$
دیکد	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{-}14)$, $AR \leftarrow IR(0\text{-}11),\ I \leftarrow IR(15)$
غیرمستقیم	D_7IT_3:	$AR \leftarrow M[AR]$
وقفه:		
	$T_0T_1T_2(IEN)(FGI + FGO)$:	$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0,\ TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR,\ PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC + 1,\ IEN \leftarrow 0,\ R \leftarrow 0,\ SC \leftarrow 0$
حافظه‌ای:		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR,\ SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR,\ E \leftarrow C_{out},\ SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR,\ SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC,\ SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR,\ SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC,\ AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR,\ SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR,\ \text{if}(DR = 0)\ \text{then}\ (PC \leftarrow PC + 1),\ SC \leftarrow 0$
ثباتی:		
	$D_7I'T_3 = r$ (مشترک در همه دستورالعمل‌های ثباتی)	
	$IR(i) = B_i\ (i = 0, 1, 2, \ldots, 11)$	
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow \overline{AC}$
CME	rB_8:	$E \leftarrow \overline{E}$
CIR	rB_7:	$AC \leftarrow \text{shr}\ AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow \text{shl}\ AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
ورودی - خروجی:		
	$D_7IT_3 = p$ (مشترک در همه دستورالعمل‌های ورودی خروجی)	
	$IR(i) = B_i\ (i = 6, 7, 8, 9, 10, 11)$	
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR,\ FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7),\ FGO \leftarrow 0$
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

تغییر مهمی که در مدار اعمال شده این است که **مدار اضافه‌ای برای کنترل گذرگاه در نظر گرفته نشده است**. توابع مستقیما به ورودی OE رجیسترها متصل می‌شوند که باعث می‌شود مقدار خروجی رجیستر روی گذرگاه یا در حالت امپدانس بالا قرار گیرد. با استفاده از این روش (به جای استفاده از Mux) توسعه و تغییرات مدار سریعتر انجام شد.

نام قطعه	پایه کنترل	تابع کنترل
حافظه	RD	R'T1 + D7'IT3 + (D0 + D1 + D2 + D6)T4
	WR	RT1 + D6T6 + (D3 + D5)T4
	CS	R'T1 + D7'IT3 + (D0 + D1 + D2 + D6)T4
AC	LD	(D0 + D1 + D2)T5 + pB11 + (B9+B7+B6)r
	INR	rB5
	CLR	rB11
	OE	D3T4 + pB10
AR	LD	D7'IT3 + (T2 + T0)R'
	INR	D5T4
	CLR	RT0
	OE	D4T4 + D5T5
PC	LD	D4T4 + D5T5
	INR	R'T1 + RT2 + (DR=0)D6T6 + AC15'rB4 + AC15rB3 + (AC=0)rB2 + E'rB1 + (FGI)pB9 + (FGO)pB8
	CLR	RT1
	OE	T0 + D5T4
DR	LD	(D0 + D1 + D2 + D6)T4
	INR	D6T5
	OE	D2T5 + D6T6
IR	LD	R'T1
	OE	R'T2
TR	LD	RT0
	OE	RT1
OUTR	LD	pB10
INR	OE	pB11
SC	CLR	RT2 + r + p + (D0+D1+D2+D5)T5 + (D3 + D4)T4 + D6T6
R	J	R'T1 + D7'IT3 + (D0 + D1 + D2 + D6)T4
	K	RT2
E	J	rB8 + D1T5Cout + rB7AC0 + rB6AC15
	K	rB8 + D1T5Cout' + rB7AC0' + rB6AC15'
	CLR	rB10
FGI	CLR	pB11
FGO	CLR	pB10

طراحی مدار

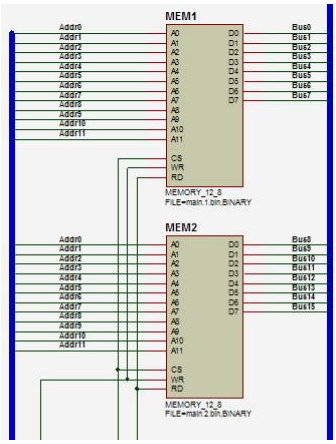
مدار زیر با نرم افزار Proteus 5.2 طراحی شده است.



نکته: به جز در مواردی که تذکر داده می‌شود، منظور از گذرگاه، خطوطی است که با Bus0 تا Bus15 مشخص شده اند.

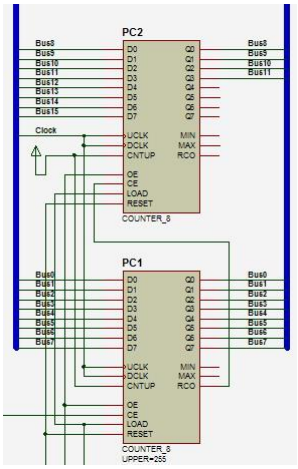
- حافظه

دو IC حافظه 4 کیلو بایتی استفاده شده است. هر کلمه (word) از ترکیب دو بایت هم آدرس از این دو IC تشکیل می‌شود. هنگام شروع بکار مدار، محتوی فایل‌های main.1.bin و main.2.bin در این ICها بارگذاری می‌شود.



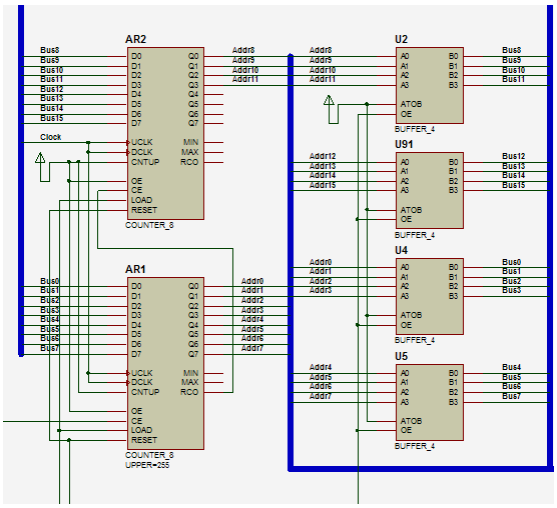
- رجیسترها

برای داشتن رجیسترهایی که فرمان inc داشته باشند، به جای شیفت رجیستر از شمارنده استفاده کرده‌ام. با توجه به اینکه نرم افزار دارای شمارنده 16 بیتی نبود برای هر رجیستر 16 بیتی (رجیستر داده) یا 12 بیتی (رجیستر آدرس) از دو شمارنده 8 بیتی استفاده کرده‌ام. رجیستر PC در زیر نشان داده شده است.

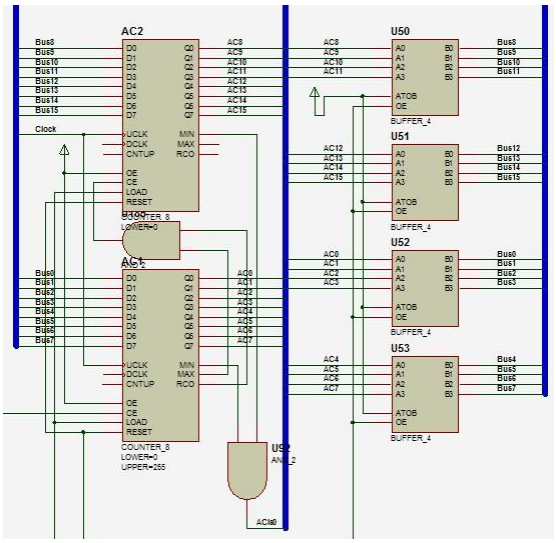


اتصال پین RCO رجیستر پایینی به پین CE رجیستر بالایی در واقع بیت نقل است (در زمان فعال شدن فرمان inc). ضمنا توجه کنید که وقتی مقدار این رجیستر آدرس روی گذرگاه قرار می گیرد 4 بیت بالایی آزاد هستند.

در مورد AR (شکل بعد) چون خروجی علاوه بر اینکه روی گذرگاه قرار می گیرد، برای آدرس دهی هم استفاده می شود، از طریق بافرهایی به گذرگاه و نیز مستقیما به حافظه متصل است.



در مورد رجیسترهای AC و DR چون لازم بود وضعیت صفر بودنشان در دسترس باشد، و با توجه به ساختار این شمارنده ها (به راهنمای نرم افزار مراجعه کنید) اتصالات تغییر داده شدند:



- ورودی – خروجی

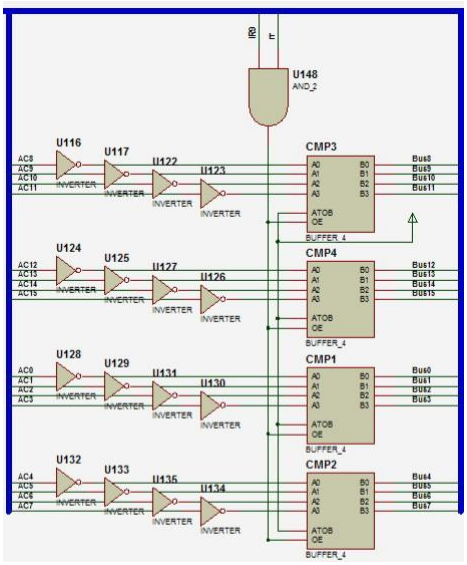
رجیسترهای OutR و InR 16 بیتی در نظر گرفته شده‌اند. ضمنا ورودی OutR و خروجی InR از طریق گذرگاه از/به AC منتقل می‌شوند (نه بطور مستقیم).

8 بیت پایین OutR و InR برای تبادل داده و 8 بیت بالا برای انتخاب وسیله خارجی در نظر گرفته شده اند:

وسیله	خط انتخاب
صفحه کلید	Out8
رجیستر مربوط به 7-segment های 1 تا 4	Out11-Out14

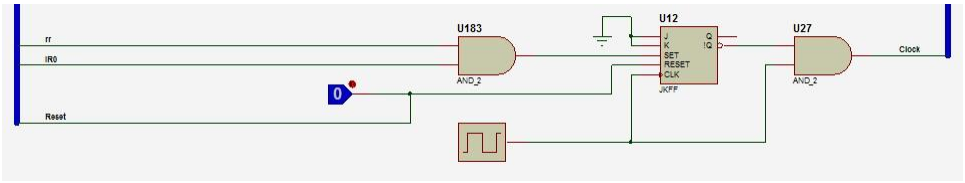
- واحد حساب و منطق (ALU)

خروجی ALU (قسمت وسط مدار) هم روی گذرگاه قرار می گیرد. بافرهایی برای کنترل خروجی‌ها در نظر گرفته شده‌اند. شکل بعد معکوس کننده AC را نشان می دهد:



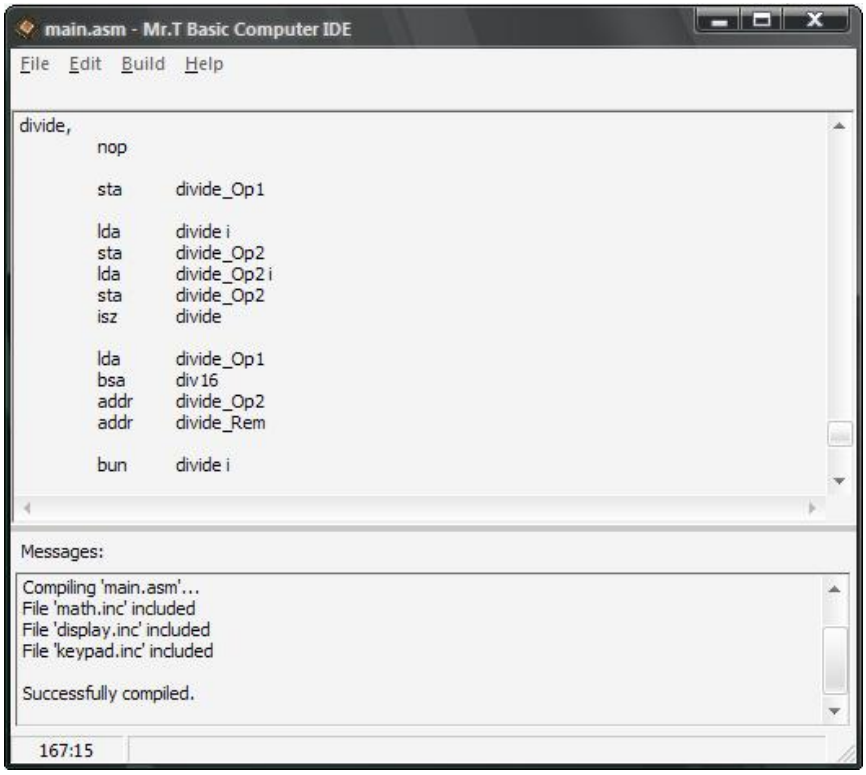
- مولد پالس ساعت

شکل زیر مدار تولید پالس ساعت را نشان می‌دهد. بعد از اجرای دستور HLT با زدن دکمه ریست، مدار مجددا راه‌اندازی می‌شود.



نرم افزار IDE

IDE ساده‌ای طراحی کرده‌ام که در آن می توان فایل برنامه را ویرایش و ترجمه کرد.



لازم به ذکر است که در مدار طراحی شده توسط اینجانب، حافظه اصلی شامل دو IC حافظه 4 کیلو بایتی است. هر کلمه از ترکیب بایت های موجود در آدرس های یکسان از دو IC ذخیره می شود. بنابراین برای ترجمه برنامه برای این مدار باید گزینه Dual Memory از منوی Build فعال باشد. اگر نام برنامه اصلی main.asm باشد دو فایل با نام های main.1.bin و main.2.bin ایجاد می شوند که باید در کنار فایل طراحی مدار قرار گیرند.

اسمبلر

یک اسمبلر تک‌گذره نوشته‌ام که برنامه اسمبلی کامپیوتر پایه را به کد ماشین معادل ترجمه می کند. در اینجا توضیحات کتاب مانو درباره قالب برنامه اسمبلی را تکرار نمی‌کنم و به تشریح تفاوت‌ها و بهبودهای اعمال شده و جزئیات لازم درباره اسمبلر می پردازم.

- رهنمود include در هر جای برنامه قرار گیرد، با محتوای فایلی که نام آن ذکر شده جایگزین می شود. توجه کنید که برچسب های استفاده شده در این فایل نباید با برنامه اصلی همنام باشند.

```
include 'file.inc'
```

اسمبلر در مسیرهای زیر به دنبال نام فایل می‌گردد:

مسیر فایل برنامه، فولدر Lib در مسیر نرم افزار، مسیر نرم افزار

- برای حفظ پویایی برنامه‌ها در تعریف اشاره‌گرها شبه دستور addr اضافه شده است. addr و به دنبال آن یک برچسب، آدرس آن برچسب را در محل شبه دستور ذخیره می کند.

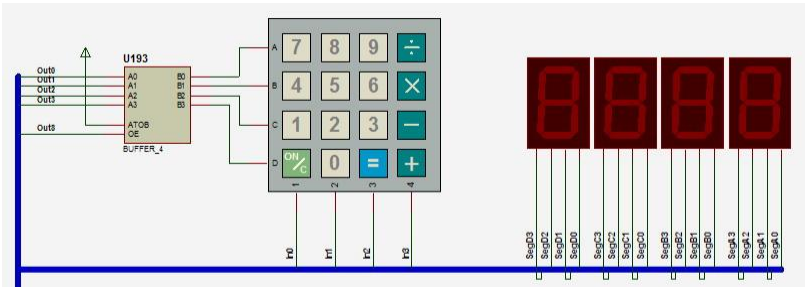
بدون addr	با addr

`Ptr,  dec 100`
`…`
`org 100`
`Array, …`
	`Ptr,  addr Array`
	`…`
	`Array, …`

- دستور بی اثر NOP با کد F020 اضافه شده است. این دستور در 4 پالس ساعت اجرا می‌شود.
- عدد مبنای 16 که بعد از شبه دستور hex ذکر می‌شود باید با یک رقم شروع شود. بعد از شبه دستور bin باید یک عدد دودویی (رشته ای از صفر و یک) بیاید. سایر اعداد استفاده شده در برنامه دهدهی فرض می‌شوند.
- پیغام‌های خطای اسمبلر شامل نام فایل، شماره سطر و توضیح خطا است. این در مورد فایل‌های include شده هم درست است. یعنی اسمبلر شماره سطرهای فایل include شده را جز شماره خطوط برنامه اصلی اعلام نمی‌کند.
- اسمبلر نسبت به بزرگی و کوچکی حروف حساس نیست.
- آخرین دستور برنامه باید یک end باشد. در فایل های include شده نباید end ظاهر شود.

ماشین حساب

برای آزمایش CPU طراحی شده با اضافه کردن یک صفحه کلید و 4 قطعه 7-segment یک ماشین حساب 4 رقمی طراحی کرده ام.



در صفحات بعد برنامه مربوط به این ماشین حساب آورده شده است. قراردادهای زیر در نوشتن روتین‌های برنامه به کار رفته است:

- اولین پارامتر ورودی/خروجی روتین‌ها در ثبات AC ارسال می شود. پارامتر یک بیتی از طریق پرچم E ارسال می‌شود. پارامترهای ورودی/خروجی بیشتر از طریق اشاره‌گرهایشان که بعد از دستور فراخوانی روتین ذخیره شده‌اند دستیابی می‌شوند.
- نام برچسب های درون هر روتین به صورت routineName_LabelName است تا با سایر برچسب‌های برنامه تداخل نداشته باشند.
- اولین حرف اسامی روتین‌ها و برچسب‌ها کوچک و حرف اول بقیه کلمات چسبیده به آنها (در صورت وجود) بزرگ است.

فایل اصلی برنامه

```
/**************************************
/ Calculator program for basic computer
/ Author: Mr.T (mrtofigh@gmail.com)
/ Last change: 1386.4.2
/ All rights reserved.
/**************************************

     org  0

/-----------------------------------------------
interrupt,
     bun  start

     bun  interrupt i

/-----------------------------------------------
     include   'math.inc'
     include   'display.inc'
     include   'keypad.inc'

/-----------------------------------------------
start,
reset,
     cla
     sta  operand
     sta  result
     sta  operator
     sta  eqWasEntered

     bsa  writeHex

loop,
     bsa  readNum16
     sta  newOperand
     cla
     cil
     sta  newOpEntered

     bsa  readKey
     sta  key

     add  resetKey
     sza
     bun  noReset
     bun  reset

noReset,
checkOpEntered,
     lda  newOpEntered
     sza
     bun  checkSuccess

checkEqKey,
```

```
     lda  key
     add  equalityKey
     sza
     bun  noOperand
     bun  doOperand

checkSuccess,
     lda  newOperand
     sta  operand

     lda  eqWasEntered
     sza
     bun  ignorePrevOperand
     bun  doOperand

ignorePrevOperand,
     cla
     sta  operator

doOperand,
     lda  operator
     bsa  lookup
     addr routines

     sta  routine
     sze
     bun  callRoutine

     lda  operand
     bun  afterCall

callRoutine,
     lda  result
     bsa  routine i
     addr operand

afterCall,
     sta  result
     bsa  writeNumber

     lda  key
     add  equalityKey
     sza
     bun  noEqKey
     bun  eqKey

noOperand,
noEqKey,
     lda  key
     sta  operator

     cla
     bun  loopEnd

eqKey,
```

```
     cla
     cma

loopEnd,
     sta  eqWasEntered

     bsa  waitNoKey
     bun  loop

key,
     hex  0
newOperand,
     hex  0
newOpEntered,
     hex  0
eqWasEntered,
     hex  0
operand,
     hex  0
result,
     hex  0
operator,
     hex  0
routine,
     hex  0
resetKey,
     dec  -13
equalityKey,
     dec  -15
routines,
     dec  4
     addr divide

     dec  8
     addr mul8

     dec  12
     addr sub16

     dec  16
     addr add16

     dec  0

/----------------------------------------
divide,
     nop

     sta  divide_Op1

     lda  divide i
     sta  divide_Op2
     lda  divide_Op2 i
     sta  divide_Op2
     isz  divide
```

```
     lda  divide_Op1
     bsa  div16
     addr divide_Op2
     addr divide_Rem

     bun  divide i

divide_Op1,
     hex  0
divide_Op2,
     hex  0
divide_Rem,
     hex  0

/---------------------------------------
     end
```

```
/****************************************
/ Math routines
/ Author: Mr.T (mrtofigh@gmail.com)
/ Last change: 1386.4.2
/ All rights reserved.
/****************************************

/----------------------------------------
/ div16 routine
/ divides one word by another
/ input:
/   ac        first operand >= 0
/   pointer   second operand <> 0
/ output:
/   ac        ac div second operand
/   pointer   ac mod second operand
/----------------------------------------
div16,
  nop

  sta   div16_Divided

  lda   div16 i
  sta   div16_DividedBy
  lda   div16_DividedBy i
  sta   div16_DividedBy
  isz   div16

  lda   div16 i
  sta   div16_Param3
  isz   div16

  cla
  sta   div16_Remainder
  sta   div16_Result

  lda   div16_Divided
  bsa   countBits
  addr  div16_Divided

  cma
  inc
  sta   div16_DBC

div16_Loop,
  lda   div16_DBC
  inc
  spa
  bun   div16_Cont
  sza
  bun   div16_End

div16_Cont,
```

```
  sta   div16_DBC

  lda   div16_Divided
  cil
  sta   div16_Divided

  lda   div16_Remainder
  cil
  sta   div16_Remainder

  lda   div16_DividedBy
  cma
  inc
  add   div16_Remainder

  cle

  sna
  bun   div16_DivideRem

  lda   div16_Result
  cil
  sta   div16_Result

  bun   div16_Loop

div16_DivideRem,
  sta   div16_Remainder

  lda   div16_Result
  cme
  cil
  sta   div16_Result

  bun   div16_Loop

div16_End,
  lda   div16_Remainder
  sta   div16_Param3 i

  lda   div16_Result

  bun   div16 i

div16_Result,
  hex   0
div16_Param3,
  hex   0
div16_Divided,
  hex   0
div16_DividedBy,
  hex   0
div16_Remainder,
  hex   0
div16_DBC,
```

```
  hex   0
/----------------------------------------

/----------------------------------------
/ countBits routine
/ Counts bits of a number
/ input:
/   ac        number
/ output:
/   ac        number of bits (position of leftmost bit + 1)
/   pointer   right shifted ac (it's left-most bit is placed at msb)
/----------------------------------------
countBits,
  nop

  sta   countBits_Num

  lda   countBits i
  sta   countBits_Ptr2
  isz   countBits

  cla
  sta   countBits_Count

  lda   countBits_Num
  sza
  bun   countBits_Loop
  bun   countBits_End

countBits_Loop,
  lda   countBits_Num
  cle
  cil

  sze
  bun   countBits_First1
  sta   countBits_Num

  isz   countBits_Count
  bun   countBits_Loop

countBits_First1,
  lda   countBits_Count
  cma
  inc
  add   countBits_Const
  sta   countBits_Count

countBits_End,
  lda   countBits_Num
  sta   countBits_Ptr2 i

  lda   countBits_Count

  bun   countBits i
```

```
countBits_Num,
  hex   0
countBits_Ptr2,
  hex   0
countBits_Count,
  hex   0
countBits_Const,
  dec   16
/----------------------------------------

/----------------------------------------
/ toBCD routine
/ Converts a word to BCD format
/ input:
/   ac        number
/ output:
/   ac        BCD number
/----------------------------------------
toBCD,
  nop

  sta   toBCD_Num

  cla
  sta   toBCD_BCD

  lda   toBCD_DigitCount
  sta   toBCD_DC

toBCD_Loop,
  lda   toBCD_Num
  bsa   div16
  addr  toBCD_Ten
  addr  toBCD_Remainder
  sta   toBCD_Num

  lda   toBCD_BitCount
  sta   toBCD_BC

toBCD_NextBit,
  lda   toBCD_Remainder
  cir
  sta   toBCD_Remainder

  lda   toBCD_BCD
  cir
  sta   toBCD_BCD

  isz   toBCD_BC
  bun   toBCD_NextBit

  isz   toBCD_DC
  bun   toBCD_Loop
```

```
  lda   toBCD_BCD
  bun   toBCD i

toBCD_Num,
  hex   0
toBCD_BCD,
  hex   0
toBCD_Ten,
  dec   10
toBCD_Remainder,
  hex   0
toBCD_DigitCount,
  dec   -4
toBCD_DC,
  hex   0
toBCD_BitCount,
  dec   -4
toBCD_BC,
  hex   0
/----------------------------------------

/----------------------------------------
/ sub16 routine
/ subtracts two words
/ input:
/   ac        first operand
/   pointer   second operand
/ output:
/   ac        ac - second operand
/----------------------------------------
sub16,
  nop

  sta   sub16_Op1

  lda   sub16 i
  sta   sub16_Ptr2
  lda   sub16_Ptr2 i
  isz   sub16

  cma
  inc
  add   sub16_Op1

  bun   sub16 i

sub16_Op1,
  hex   0
sub16_Ptr2,
  hex   0
/----------------------------------------

/----------------------------------------
/ add16 routine
/ adds two words
```

```
/ input:
/   ac        first operand
/   pointer   second operand
/ output:
/   ac        ac + second operand
/-----------------------------------------
add16,
  nop

  sta   add16_Op1

  lda   add16 i
  sta   add16_Ptr2
  lda   add16_Ptr2 i
  isz   add16

  add   add16_Op1

  bun   add16 i

add16_Op1,
  hex   0
add16_ptr2,
  hex 0
/-----------------------------------------

/-----------------------------------------
/ mul8 routine
/ multiplies two small words
/ input:
/   ac        first operand
/   pointer   second operand
/ output:
/   ac        ac * second operand
/-----------------------------------------
mul8,
  nop

  sta   mul8_Op1

  lda   mul8 i
  sta   mul8_Op2
  lda   mul8_Op2 i
  sta   mul8_Op2
  isz   mul8

  cla
  sta   mul8_Result

mul8_Loop,
  lda   mul8_Op1
  sza
  bun   mul8_NoZero
  bun   mul8_End
```

```
mul8_NoZero,
  cle
  cir
  sta   mul8_Op1
  sze
  bun   mul8_Add
  bun   mul8_AfterAdd

mul8_Add,
  lda   mul8_Op2
  add   mul8_Result
  sta   mul8_Result

mul8_AfterAdd,
  lda   mul8_Op2
  cle
  cil
  sta   mul8_Op2

  bun   mul8_Loop

mul8_End,
  lda   mul8_Result
  bun   mul8 i

mul8_Result,
  hex   0
mul8_Op1,
  hex   0
mul8_Op2,
  hex   0
/----------------------------------------
```

```
/****************************************
/ Display routines
/ Author: Mr.T (mrtofigh@gmail.com)
/ Last change: 1386.4.2
/ All rights reserved.
/****************************************

/----------------------------------------
/ writeNumber routine
/ Displays a word in it's decimal format
/ input:
/   ac        number to be displayed
/----------------------------------------
writeNumber,
  nop

  bsa   toBCD
  bsa   writeHex

  bun   writeNumber i
/----------------------------------------

/----------------------------------------
/ writeHex routine
/ Displays a hex number
/ input:
/   ac        number to be displayed
/----------------------------------------
writeHex,
  nop

  sta   writeHex_Num

  lda   writeHex_Count
  sta   writeHex_C

  lda   writeHex_SegA
  cma
  sta   writeHex_Seg

writeHex_Loop,
  lda   writeHex_Num
  and   writeHex_f4
  cma

  and   writeHex_Seg
  cma
  out

  lda   writeHex_Seg
  cle
  cme
  cil
```

```
  sta   writeHex_Seg

  lda   writeHex_Num
  cir
  cir
  cir
  cir
  sta   writeHex_Num

  isz   writeHex_C
  bun   writeHex_Loop

  bun   writeHex i

writeHex_Num,
  hex   0
writeHex_segA,
  hex   800
writeHex_Seg,
  hex   0
writeHex_Count,
  hex   -4
writeHex_C,
  hex   0
writeHex_f4,
  hex   0f
/---------------------------------------
```

```
/****************************************
/ Keypad routines
/ Author: Mr.T (mrtofigh@gmail.com)
/ Last change: 1386.4.2
/ All rights reserved.
/****************************************

/---------------------------------------
/ lookup routine
/ Searches a key in a table and returns it's associated data
/ input:
/   ac        key
/   pointer   table. a 0 value indicates end of table
/ output:
/   ac        data stored in table after found key
/   e flag    1 = found, 0 = not found
/---------------------------------------
lookup,
  nop

  sta   lookup_Key

  lda   lookup i
  sta   lookup_Table
  isz   lookup

  bun   lookup_LoopStart

lookup_Loop,
  isz   lookup_Table

lookup_LoopStart,
  lda   lookup_Table i
  isz   lookup_Table

  sza
  bun   lookup_Cont
  bun   lookup_NotFound

lookup_Cont,
  cma
  inc
  add   lookup_Key

  sza
  bun   lookup_Loop

lookup_Found,
  lda   lookup_Table i

  cle
  cme
```

```
  bun   lookup_End

lookup_NotFound,
  cle

lookup_End,
  bun   lookup i

lookup_Table,
  hex   0
lookup_Key,
  hex   0
/----------------------------------------

/----------------------------------------
/ waitKey routine
/ waits until a key being pressed
/----------------------------------------
waitKey,
  nop

waitKey_Loop,
  bsa   readKey
  sza
  bun   waitKey_End
  bun   waitKey_Loop

waitKey_End,
  bun   waitKey i
/----------------------------------------

/----------------------------------------
/ waitNoKey routine
/ waits until no key being pressed
/----------------------------------------
waitNoKey,
  nop

waitNoKey_Loop,
  bsa   readKey
  sza
  bun   waitNoKey_Loop

  bun   waitNoKey i
/----------------------------------------

/----------------------------------------
/ readNum16 routine
/ waits for entering a number,
/   displays number on 7-segments
/   returns when a non-digit key pressed
/ output
/   ac        number
/   e flag    1 = a number returned in ac, 0 = no number
/----------------------------------------
```

```
readNum16,
  nop

  cla
  sta   readNum16_NumEntered
  sta   readNum16_Num

readNum16_Loop,
  bsa   readKey

  sza
  bun   readNum16_Cont
  bun   readNum16_Loop

readNum16_Cont,
  sta   readNum16_Key

  bsa   keyToDigit
  sta   readNum16_Digit

  sze
  bun   readNum16_DigitKey
  bun   readNum16_NonDigitKey

readNum16_DigitKey,
  cla
  inc
  sta   readNum16_NumEntered

  lda   readNum16_Num
  bsa   sub16
  addr  readNum16_4Digit

  sna
  bun   readNum16_Loop

  lda   readNum16_Num
  bsa   Mul8
  addr  readNum16_Ten

  add   readNum16_Digit
  sta   readNum16_Num

  bsa   writeNumber
  bsa   waitNoKey

  bun   readNum16_Loop

readNum16_NonDigitKey,
readNum16_End,
  lda   readNum16_NumEntered
  cir

  lda   readNum16_Num
```

```
  bun   readNum16 i

readNum16_NumEntered,
  hex   0
readNum16_Digit,
  hex   0
readNum16_Num,
  hex   0
readNum16_Key,
  hex   0
readNum16_4Digit,
  dec   1000
readNum16_Ten,
  dec   10
/---------------------------------------

/---------------------------------------
/ keyToDigit routine
/ returns digit from its key code
/ input:
/   ac        Key code
/ output:
/   ac        digit
/   e flag    1 = digit, 0 = non-digit
/---------------------------------------
keyToDigit,
  nop

  bsa   lookup
  addr  keyToDigit_Digits

  bun   keyToDigit i

keyToDigit_Digits,
  dec   1
  dec   7

  dec   2
  dec   8

  dec   3
  dec   9

  dec   5
  dec   4

  dec   6
  dec   5

  dec   7
  dec   6

  dec   9
  dec   1
```

```
  dec   10
  dec   2

  dec   11
  dec   3

  dec   14
  dec   0

  dec   0
/---------------------------------------

/---------------------------------------
/ readKey routine
/ returns code of pressed key
/ output:
/   ac        0 = no key pressed, non-0 = key code
/---------------------------------------
readKey,
  nop

  cla
  sta   readKey_Result
  sta   readKey_Col

  lda   readKey_RowCount
  sta   readKey_RC

  lda   readKey_Row
  sta   readKey_R

readKey_RowLoop,
  lda   readKey_R
  and   readKey_Keypad
  cma
  out

  inp
  sza
  bun   readkey_Pressed

  lda   readKey_R
  cle
  cme
  cil
  sta   readKey_R

  lda   readKey_Result
  add   readKey_Four
  sta   readKey_Result

  isz   readKey_RC
  bun   readKey_RowLoop

/-- end of RowLoop --
```

```
  cla
  bun   readKey_End

readKey_Pressed,
  cle
  cir
  sze
  bun   readKey_Pressed2
  isz   readKey_Col
  bun   readKey_Pressed

readKey_Pressed2,
  lda   readKey_Result
  add   readKey_Col
  inc

readKey_End,
  bun   readKey i

readKey_Keypad,
  hex   0feff
readKey_Result,
  hex   0
readKey_Four,
  hex   4
readKey_Row,
  hex   0fffe
readKey_R,
  hex   0
readKey_RowCount,
  hex   -4
readKey_RC,
  hex   0
readKey_Col,
  hex   0
/---------------------------------------
```

مراجع

1. معماری کامپیوتر، موریس مانو، ترجمه دکتر قدرت سپیدنام، انتشارات خراسان، 1385.
2. طراحی کامپایلر، آلفرد وی. اهو، ترجمه مهدی آصفی، انتشارات ناقوس، 1380.
3. راهنمای نرم‌افزار PROTEUS (نسخه 5.20.07).