بسمه تعالی

گزارش طراحی سی پی یو

محمدصادق خراسانی

95108008

تمرین سری سوم

<u>شیوه ی کار و فرضیات اولیه باتوجه به طرح کلی :</u>

ابتدا DATAPATHرا طراحی میکنیم.

در ادامه شمای کلی DATAPATHرا میبینید.

در این شما از دو مالتی پلکسر 2 به 4 استفاده شده است که یکی مخصوص بردن حامل بر روی باس و دیگری مخصوص قرار دادن ادرس مطلوب بر روی ورودی ادرس مموری است.

اما شمای کنترل کلی دستورات با توجه به این که در ابتدا گفته شده بود تنها از خروجی 8 بیتی مموری برای دستورات میتوان استفاده کرد به صورت

fetch

Decode

Execute

میباشد.

چراکه در ابتدا گفته شده بود دستورات را 8 بیتی باید بخوانید در نتیجه فهمیدن تعداد operandها بسته به این که دستور چه باشد دارد.

که بعدا در cwپاسخ داده شد که میتوان 32 بیتی گرفت و مهم کارکرد مطلوبات است.

برخی از توضیحات وفرضیات:

نکته ی اول در مورد ویرایش حافظه میباشد.

بدلیل محدودیت iterationها تعداد بایت های حافظه را از 65535به 4096رساندم که نهایتا با 12 بیت ادرس دهی میشود اما بدلیل ارائه ی ماژول با ادرس 32 بیت در ساخت دیتاپس از رجیستر های 32 بیتی استفاده شده است.

<u>البته برای تست کردن نهایی مجبور شدم حافظه را به 256خانه کوتاه کنم تا کوارتیوس توان شبیه سازی نهایی را داشته باشد.</u>

<u>در غیر این صورت کوارتیوس ارور کمبود فضا میدهد و روند کاپایل و شبیه سازی بسیار کندی خواهد داشت.</u>

دوراه برای استفاده از حافظه ارائه شده است که هر دو قابل برنامه ر یزی هستند

در دیتا پس کنونی برای جلوگیری از دسترسی قسمت های حافظه راه دوم بهتر شمارده شده است اما بدلیل اینکه مقدار حافظه را 256 بایت کردم(برای تست کردن)از راه اول استفاده کرده ام(تا بتوانم تست بگیرم)

راه اول :

حدود 10 خانه ی اول حافظه به initialکردن مقادیر اولیه ی پوینتر ها اختصاص داده شود که با توجه به این قضیه به راحتی میتوان محدوده هایی که برای stackو دیگر قسمت های حافظه اشاره میکند را تغییر داد.

علاوه بر این برای varnum 256 خانه ی اول یه دوم اختصاص داده شده است.در واقع امکان برای هر دوتا باز است و با اضافه کردن یک گیت میتوان ان را تغییر داد.یعنی دیتاپس برای هر دو سازگار است.برای constها 256تا خانه ی دوم یا 10 ام حافظه اشاره میشود مانند varnumقابل تغییر است.و برای stackهم از یک جایی که در همان initialمشخص میشود تاا انتهای حافظه در نظر گرفته شده است. و برای قسمت دستورات هم با استفاده از همان قسمت initialبه نقطه ی شروع دلخواه jumpمیشود.

البته میتوانیم این 10 خانه ی اول حافظه را کاملا از varnumجدا کنم ولی به دو دلیل دارد که 256خانه ی اول را با صرف نظر از 10 خانه ی initialبه وارنیوم و 256 تای دوم به constها اشاره شده است که هیچ کدام از 4 قسمت گفته شده از حافظه به هم دسترسی ندارند

یک :جلو گیری از اتلاف فضای بینابینی 10 خانه ی initialو نقطه ی شروع varnum

دو :جلو گیری از پیچیدگی ادرس دهی حافظه که به سادگی انجام شده است.

این تقسیم بندی این گونه عمل شده است که

توضیحات راجع به 10خانه ی اول initial

در ابتدا که پردازنده شروع به کار میکند

همه ی مقادیر رجیستر ها صفر میشود و این یعنی pcمقدار 0 را اتخاذ میکند...در خانه ی اول حافظه دستوری در نظر گرفته شده است برای ست کردن مقدار sp ...پس عددی از ادرس حافظه را میدهیم که تا انتها مال stackباشد. سپس دستور unconditional jumpبرای pcاجرا میشود و ان را به ناحیه ایی که دستورات را در ان نوشتیم(طبیعتا بعد از constها و قبل از stackها)میبرد.و در ان جا به ترتیب دستورات اجرا میشوند.و این تا جایی پیش میرود که دستورات تمام شوند یا مانند کامپیوتر پایه به دستوری مانند hultبربخوریم(که انتهای هر برنامه نوشته میشود)پایان کار سی پی یو میباشد.

راه دوم:

این است که مقدار بیت های خروجی از رجیستر های PCیاSPو..را کنترل کنیم .CPPوVARNUMمانند حات قبل است اما برای SPیا PCبه ترتیب بیت 17 و 18 هر کدام را به 1 منطقی وصل کنیم تا و آن ده خانه ی اول حافظه را برای initialکردن آن ها قرار ندهیم که این موضوع باعث این مورد میشود که بیت های ادرس دهی برای هر کدام متغییر باشد (به ترتیب 16 و17) و علاوه براین مقداری از حافظه هدر خواهد رفت .پس در این جا همان راه اول در نظر گرفته میشود.

برای دستورات که دارای اپرند های دوبایتی یا تک بایی هستنداین گونه در حافظه قرار میگیرند که ابتدا کد 8 بیتی دستور میآید و سپس اپرند آن(درصورت وجود)

اگر اپرند دو بایتی باشد ابدا بایت کم ارزش و سپس بایت پر ارزش قرار خواهد گرفت.

در صورتی که دستوری با اپرند offsetباشد با توجه به اینکه pcجمع کننده ی داخلی دارد(در ادامه توضیح داده خواهد شد)عملیات branchدر یک کلاک انجام خواهد گرفت .علاوه بر این نیازی به ویرایش آن وردی که دو بایت اولش offsetهست نمیباشد چراکه با توجه به ماژول حافظه فقط 16 بیت اول آن برداشت میشود و بقیه ی بیت ها مورد استفاده نخواهند بود.

اما روند کار:

ایتدا رجیستر هایی ساخته شده است که ورودی LOADوINCوCLEARرا شامل میشوند.

این رجیستر ها در اندازه های 8 ,16و 32 بیتی ساخته شده اند که همه ی ان ها از Dفلیپ فلاپ ها بهره میبرند.

اما در این ساختار دورجیستر هست که بدیل خاص بودن ویژگی هایی که دارند لازم به توضیحات بیشتری راجع به ان ها دیدم.

رجیستر SP:

این رجییستر برای اشاره به قسمت استک حافظه استفاده میشود که بدلیل انکه ماژول حافظه 32بیت خروجیش با هم همپوشانی دارند باید پوینتر استک 4 تا جابجا شود.

پس کنترل لود(برای مقدار دهی اول) و اینکریمنت4 تایی برای این پونتر گذاشته شده تا از پیچیدگی جمع و تفریق 4 تایی کاسته شود.

رجیستر :PC

این رجیستر عملیات INCرا دارد این کنترل با افزایش یدونه ایی ادرس حافظه بایت به بایت جلو میرود و دستورات را میخواند.

اما نکته ایی که دارد این است که دستوری که از این رجیستر زیاد استفاده میکند دستور branchمیباشد.

بدیل اینکه گفته شده این دستور در صورت اجرا باید مقدار فعلی PCرا با خود جمع کند پس کنترل SelfAdderبرای ان قرار داده شده است تا با یک کلاک جمع کقدار فعلی با مقدار offsetانجام شود.

رجیستر های دیگری هم وجود دارند که برای محاسبه و خواندن دستورات استفاده میشوند.

اما توضیح راجع به رجیستر OP

این رجیستر برای کنترل خروجی مموری استفاده میشود.

مثلا برای حالتی که قرار است یک بایت به استک پوش شود باید 24 بیت باقی صفر شوند پس این رجیستر فقط باید 8 بیت اول را لود کند و بقیه ی بیت های خروجی مموری را صفر کند.

درواقع ساختار این رجیستر 32 بیتی از کنار هم ماندن دو رجیستر 8 بیتی (OP0,OP1)و یک رجیستر16 بیتی دارد.

ورودی ادرس مموری با توجه به یک مالتی پلکسر 2 به 4 انتخاب میشود .

مقدار آنی باس هم یکی از 4 رجیستر OPوTMPوDRویا خروجی مموری تامین میشود که این هم به یک مالتی پلکسر 2 به 4 نیاز دارد.

در ادامه شمای دیتا پس و RTLها و MICRO operationها را میبینیم.

رجیسترTMP

برای قرار دادن حاصل محاسبات در آن انجام میشود که در مواقع لازم میتواند بر روی باس برود.

اما شیوه ی decodeکردن دستور ها :

ابتدا با ریست شدن سی پی یو مقادیر اولیه رجیستر ها ها همه صفر در نظر گرفته میشود...

10 خانه ی اول رم را برای initialکردن اولیه ی مقادیر اشاره گر ها ست میشود.

به همین منظور دستوری به دستور ها برای ست کردن spاضافه شده است.(با کد دستوری 0x11)

با قرار گرفتن باید ها در رجیستر 8 بیتی IRآن را DECODEمیکنیم.

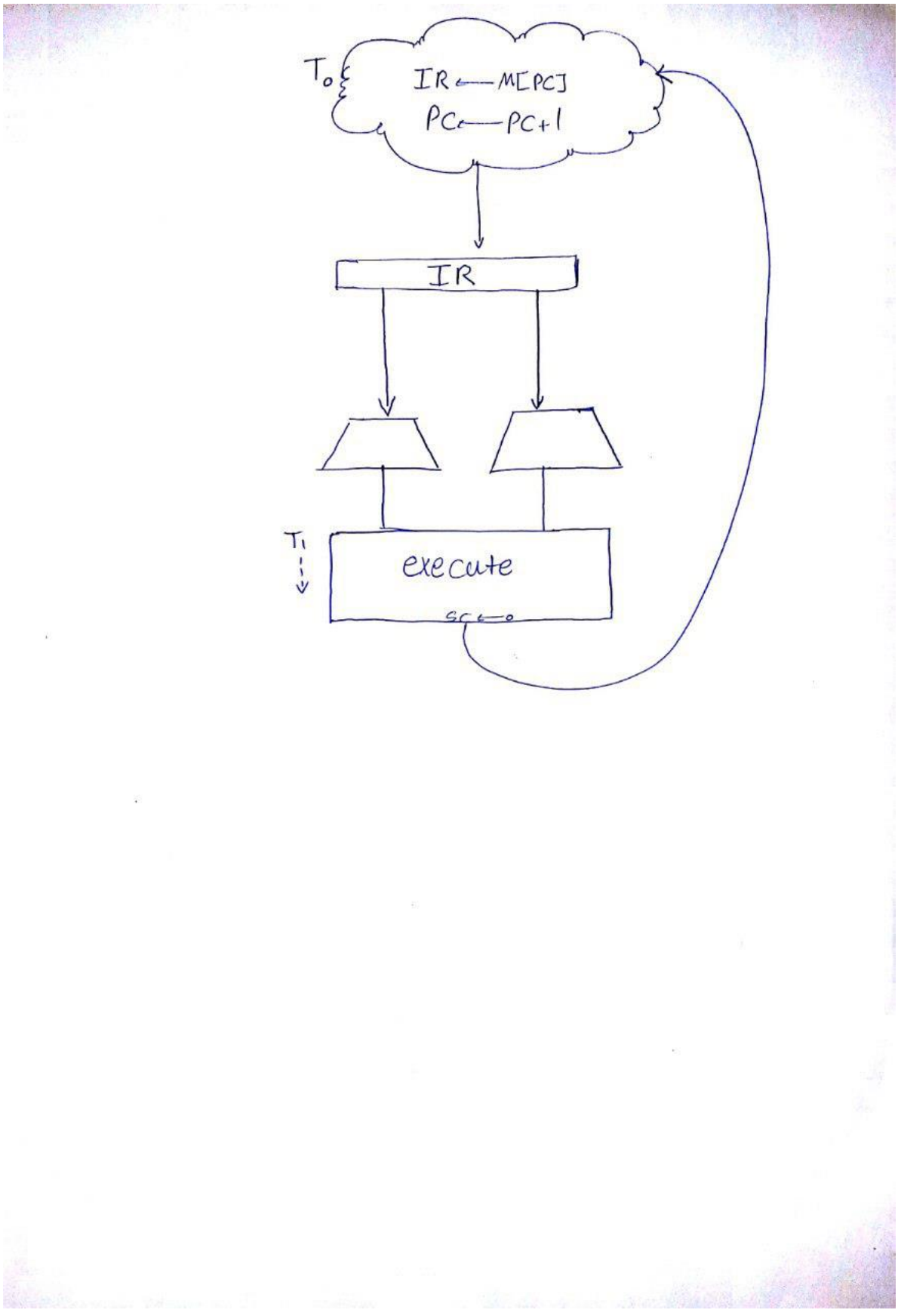
دیکود کردن این رجیستر با استفاده از دو انکودر ☺انجام میشود که قابلیت گرفتن 256تا دستور را میتوان از آن داشت.

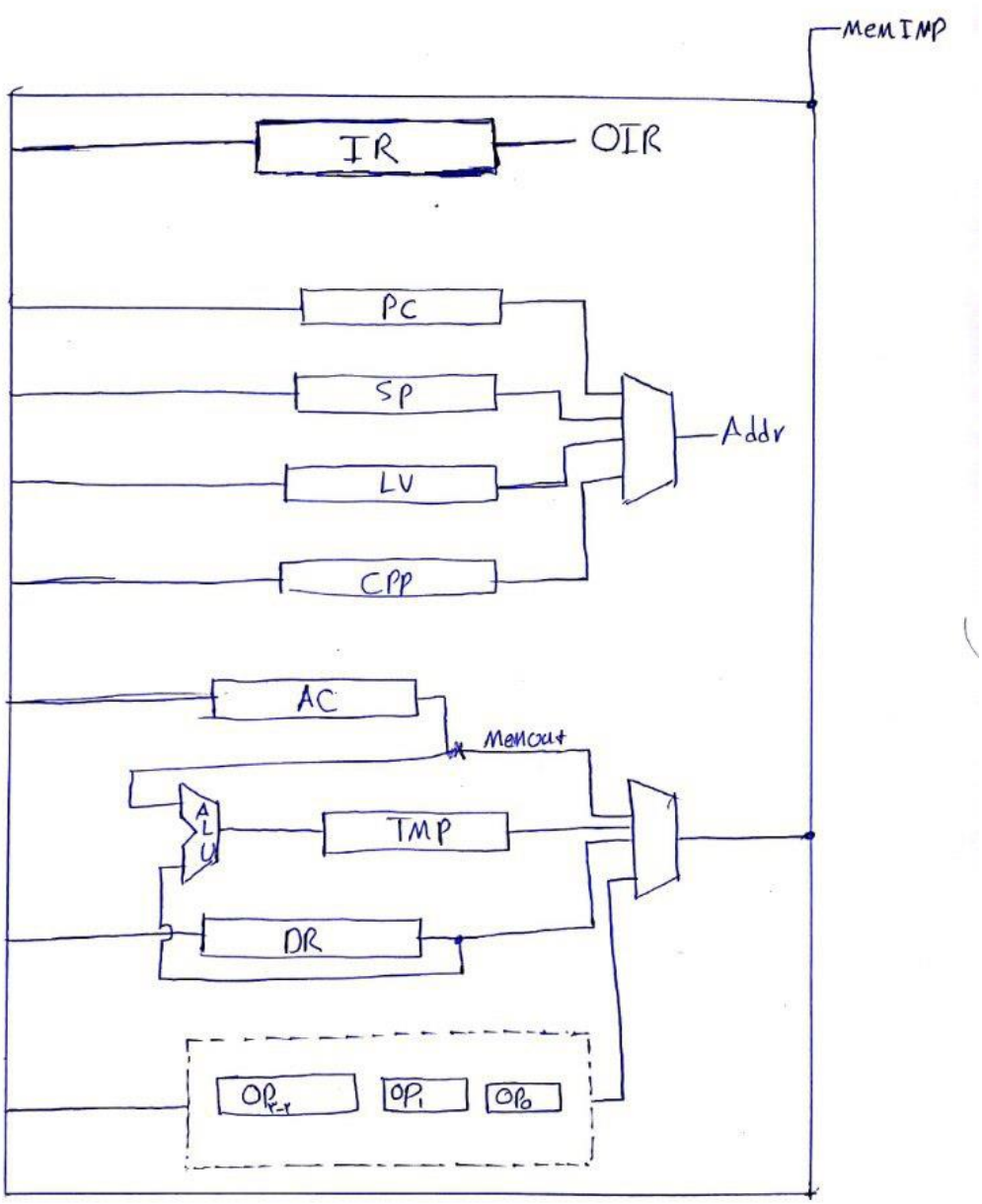
به اینصورت که 4 بیت اول و 4 بیت دوم هر کدام به یک انکودر 4به 16 میرود هر انکودر خروجی هایی به پارامتر D,Sمیدهند.(با اندیس های 0 تا 15)

.....

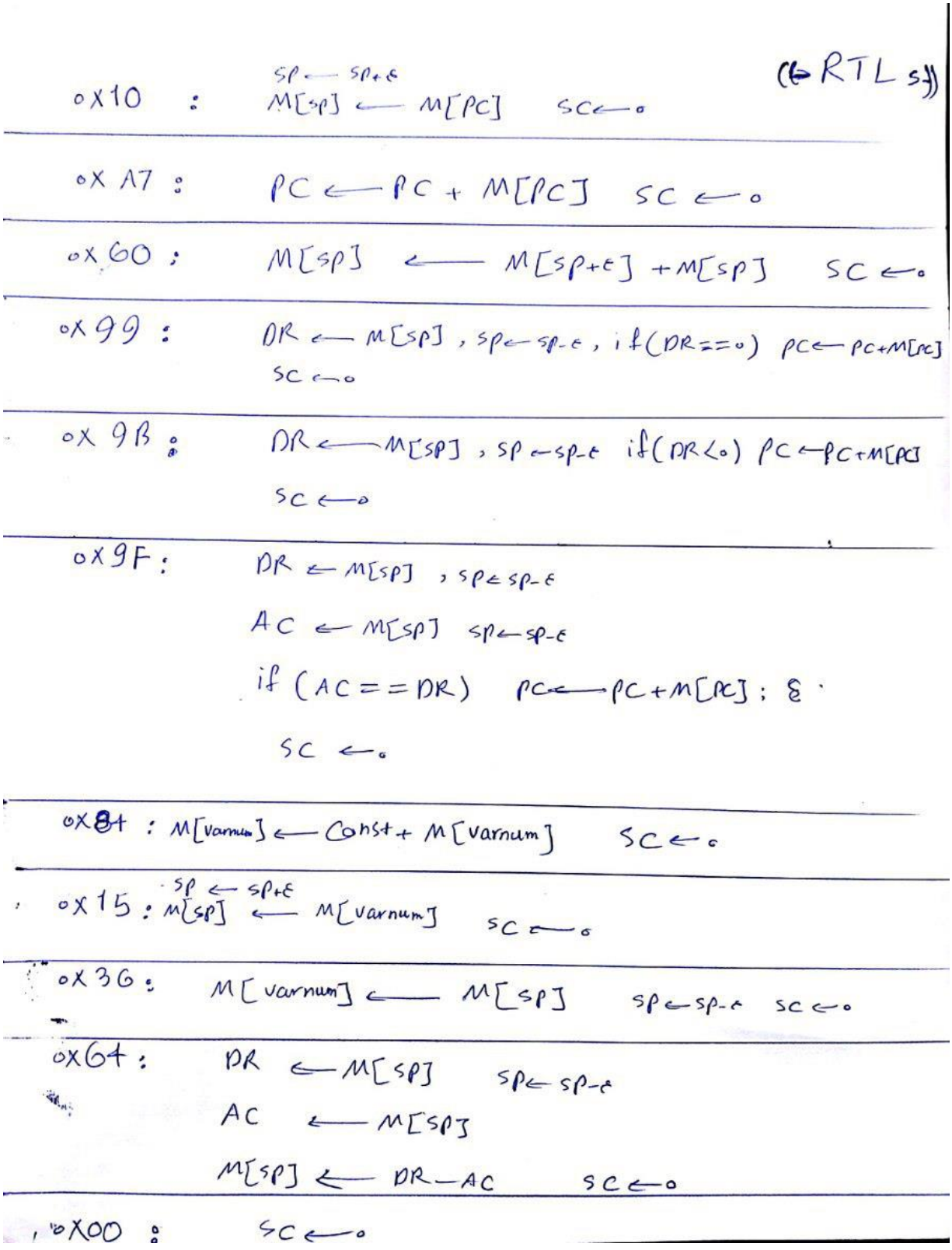
در مورد کنترل کردن سی پی یو در تعامل با حافظه این گونه عمل شده است که هر جا نیاز به استفاده از حافظه بوده است دکمه ی استارت زده میشود و تا گرفتن خروجی آماده شده صبر میشود و پس از آن کانتر اضافه میشود.

در بعضی از سیگنال های کنترلی در کنترل یونیت میبینیم که با ورودی enableکانتر andشده است...این به این دلیل است که در زمان هایی که منتظر یک شدن سیگنال readyمموری هستیم کلاک های اضافی باعث متفاوت شدن مقادیر داخل رجیستر هایمان نشود.


T0
IR ← M[PC]
PC ← PC+1
IR
T1
execute
SC ← 0


MemIMP
IR
OIR
PC
SP
LV
CPP
Addr
AC
MemOut
ALU
TMP
DR
OP_{n-1}
OP_1
OP_0

برای هر دستور RTLمربوطه نوشته شده و سپس برای هر کدام میکرواپریشن لازمه با توجه به دیتا پس ایجاد شده است.



در اینجا هم میکرواپریشن ها را مشاهده میکنیم.

دستورات برداشته (بجای میکرواپریشن) ، کنترل فانکشن ها را می بینیم.

$D_1 S_0 T_1$: $OP \leftarrow 0$, $OP_0 \leftarrow M[PC]$, $SP \leftarrow SP+1$

// // T_2 : $M[SP] \leftarrow OP$, $PC \leftarrow PC+1$, $SC \leftarrow 0$

$D_{10} S_V T_1$: $PC \leftarrow PC + M[PC]$, $SC \leftarrow 0$

$D_4 S_0 T_1$: $DR \leftarrow M[SP]$, $SP \leftarrow SP-1$

// // T_2 : $AC \leftarrow M[SP]$

// // T_3 : $TMP \leftarrow AC + DR$

// // T_4 : $M[SP] \leftarrow TMP$, $SC \leftarrow 0$

$D_9 S_9 T_1$: $DR \leftarrow M[SP]$, $SP \leftarrow SP-1$

// // T_2 : if $(DR == 0)$ $PC \leftarrow PC + M[PC]$

// // T_3 : $PC \leftarrow PC+1$

// // T_4 : $PC \leftarrow PC+1$, $SC \leftarrow 0$

$D_9 S_{11} T_1$: $DR \leftarrow M[SP]$, $SP \leftarrow SP-1$

// // T_2 : if $(DR[3] == 0)$ $PC \leftarrow PC+1$

// // T_3 : if $(DR[3] == 1)$ $PC \leftarrow PC+1$, $SC \leftarrow 0$

// // T_4 : $PC \leftarrow PC + M[PC]$, $SC \leftarrow 0$

$D_9 S_{10} T_1$: $DR \leftarrow M[SP]$, $SP \leftarrow SP-1$

// // T_2 : $AC \leftarrow M[SP]$, $SP \leftarrow SP-1$

// // T_3 : $TMP \leftarrow AC - DR$

// // T_4 : $AC \leftarrow TMP$

// // T_5 : if $(AC == 0)$ $PC \leftarrow PC + M[PC]$, $SC \leftarrow 0$

// // T_6 : $PC \leftarrow PC+1$

// // T_7 : $PC \leftarrow PC+1$, $SC \leftarrow 0$

$D_7 S_6 T_1$: $OP_{n-1} \leftarrow 0$, $OP_0 \leftarrow M[PC]$

" " T_2 : $LV \leftarrow OP$, $PC \leftarrow PC+1$

" " T_3 : $DR \leftarrow M[LV]$

" " T_4 : $OP_{n-1} \leftarrow 0$, $OP_0 \leftarrow M[PC]$, $PC \leftarrow PC+1$

" " T_5 : $AC \leftarrow OP$

" " T_6 : $TMP \leftarrow AC + DR$

" " T_7 : $M[LV] \leftarrow TMP$, $SC \leftarrow 0$

$D_1 S_5 T_1$: $OP_{n-1} \leftarrow 0$, $OP_0 \leftarrow M[PC]$

" " T_2 : $LV \leftarrow OP$, $PC \leftarrow PC+1$

" " T_3 : $DR \leftarrow M[LV]$, $SP \leftarrow SP+1$

" " T_4 : $M[SP] \leftarrow DR$, $SC \leftarrow 0$

$D_2 S_4 T_1$: $DR \leftarrow M[SP]$, $SP \leftarrow SP-1$, $OP_{n-1} \leftarrow 0$

" " T_2 : $OP_0 \leftarrow M[PC]$

" " T_3 : $LV \leftarrow OP$, $PC \leftarrow PC+1$

" " T_4 : $M[LV] \leftarrow DR$, $SC \leftarrow 0$

$D_4 S_3 T_1$: $DR \leftarrow M[SP]$, $SP \leftarrow SP-1$

" " T_2 : $AC \leftarrow M[SP]$

" " T_3 : $TMP \leftarrow AC - DR$

" " T_4 : $M[SP] \leftarrow TMP$, $SC \leftarrow 0$

$D_0 S_2 T_1$: $SC \leftarrow 0$

$D_1 S_1 T_1$: $SP \leftarrow M[PC]$, $PC \leftarrow PC+1$

" " T_2 : $PC \leftarrow PC+1$

" " T_3 : $PC \leftarrow PC+1$

" " T_4 : $PC \leftarrow PC+1$, $SC \leftarrow 0$

نحوه ی کار را در توضیحات آتی میبینیم:

برای هر رجیستر با استفاده از کنترل های آن در میکرو اپریشن ها نگاه میشود و کنترل فانکشن ها ی مربوطه با هم ORمیشوند.

برای کنترل یونیت باسی تحت عنوان CntrlSignalخارج میشود که سیگنال های کنترلی برای رجیستر ها و مالتی پلکسر ها را تولید میکند.

برای انجام این کار به هر کدام از کنترل ها یک شماره نسبت داده شده است که در کنترل یونیت همان شماره با استفاده از کنترل فانکشن ها تولید میشود.

در زیر شماره ی هر کدام از کنترل های رجیستر ها را میبینیم.

Load[IR] : ۵

(selPAddr)Load[PC] : ۶

INC[PC] : ۷

LD[SP] : ۲

INC+[SP] : ۸

DeC+[SP] : ۹

LD[LV] : ۱۰

LD[CPP] : ۱۱

LD[DR] : ۱۲

LD[AC] : ۱۳

LD[TMP] : ۱۴

LD[OP$_0$] : ۱۵

CLR[OP$_{e-1}$] : ۱۶

INC[OP$_1$] : ۱۷

MUX[۲:۶] : [۱..۵] (برای آدرس‌ها)

MUX[۲:۶] : [۶..۳] (برای باس)

ALUControl : [۲۳..۱۸]

و در ادامه کنترل های هر کدام را که در کنترل یونیت سیم بندی شده است را مشاهده میکنید.

$$INC[PC]: D_1S_0T_2 + D_9S_9(T_8+T_9) + D_9S_{11}DR[0\text{-}1][T_2+T_6] + D_9S_{10}(T_9+T_7) + T_0 + D_8S_6T_6 + D_{12}S_7T_3 + D_1S_8T_2 + D_1S_4(T_1+T_2+T_6+T_6)$$

$$BelfAdder[PC]: D_{10}S_7T_1 + D_9S_9Z(T_6) + D_9S_{11}T_8 + D_9S_{13}T_8Z$$

$$INC+[SP] = D_1S_0T_1 + D_1S_8T_3$$

$$Dec4[SP] = D_4S_0T_1 + D_9S_9T_1 + D_9S_{11}(T_1) + D_9S_{10}(T_1+T_2) + D_9S_9T_1 + D_4S_6T_1$$

$$LD[LV] = D_8S_6T_2 + D_1S_8T_2 + D_{12}S_9T_3$$

$$LD[DR] = D_9S_0T_1 + D_9S_9T_1 + D_9S_{11}T_1 + D_9S_{10}T_1 + D_8S_6T_3 + D_1S_8T_3 + D_{12}S_9T_1 + D_4S_6T_1$$

$$LD[AC] = D_9S_0T_2 + D_9S_{10}(T_2+T_6) + D_9S_6T_2 + D_8S_6T_8$$

$$LD[TMP] = D_9S_8T_3 + D_9S_{10}T_3 + D_9S_6T_3$$

$$LD[OP.] = D_1S_8T_1 + D_8S_6(T_1+T_6) + D_1S_8T_1 + D_{12}S_9T_2$$

$$CLR[OP_{3-1}]: D_1S_0T_1 + D_8S_6T_1 + D_1S_8T_1 + D_{12}S_9T_1$$

[PC] Max[r:e]: $D_1 S_1 T_1 + D_3 S_6 (T_1 + T_4) + D_1 S_3 T_1 + D_6 S_4 T_7 + D_1 S_5 T_1 + D_9 S_9 (T_{10} + T_{12}) Z$

$+ D_9 S_{11} T_{13} + D_9 S_{13} T_{13} Z$

[SP] max[r:e]: $D_1 S_0 T_7 + D_4 S_6 (T_1 + T_7 + T_6) + D_9 S_9 (T_1 + T_7) + D_9 S_{11} (T_1 + T_7)$

$D_9 S_{10} (T_1 + T_7) + D_1 S_3 T_6 + D_6 S_4 T_1 + D_4 S_6 (T_1 + T_{10} + T_6)$

[LV] = $D_1 S_6 T_4 + D_1 S_3 T_6 + D_6 S_4 T_6 + D_1 S_6 T_6$

SC = $D_1 S_0 T_7 + D_1 S_5 T_1 + D_4 S_0 T_6 + D_9 S_9 Z (T_{10} + T_6) + D_9 S_9 T_4 + D_9 S_{11} T_6$

$DR[C1] + D_9 S_{11} T_{13} + D_9 S_{13} T_{13} Z + D_9 S_9 T_5 + D_1 S_6 T_4 +$

$D_1 S_3 T_6 + D_6 S_4 T_6 + D_4 S_6 T_6 + D_3 S_6 T_1 + D_1 S_1 T_6$

Start: $D_1 S_0 (T_1 + T_7) + D_1 S_5 T_1 + D_4 S_0 (T_1 + T_7 + T_6) + D_9 S_9 (T_1) + D_9 S_9 Z T_7$

$+ D_9 S_{11} (T_1 + T_6) + D_9 S_{10} (T_1 + T_7 + T_{13}) + D_1 S_6 (T_1 + T_6 + T_4 + T_5)$

$+ D_1 S_3 (T_1 + T_6 + T_6) + D_6 S_4 (T_1 + T_{10} + T_6) + D_4 S_6 (T_1 + T_7 + T_{16})$

$+ D_1 S_1 T_1 + T_0$

در انتها تستی برای شما قرار داده میشود که لود شدن مقدار SPو دستور BIPUSHپس از آن اجرا میشود.

در واقع در این تست خانه های مموری اینگونه که در شکل زیر میبینیم مقدار دهی اولیه شدند.

array[0] <= 8'b0001_0001;

array[1] <= 8'b0010_0000;

array[2] <= 8'b0000_0000;

array[3] <= 8'b0000_0000;

array[4] <= 8'b0000_0000;

array[5] <= 8'b0001_0000;

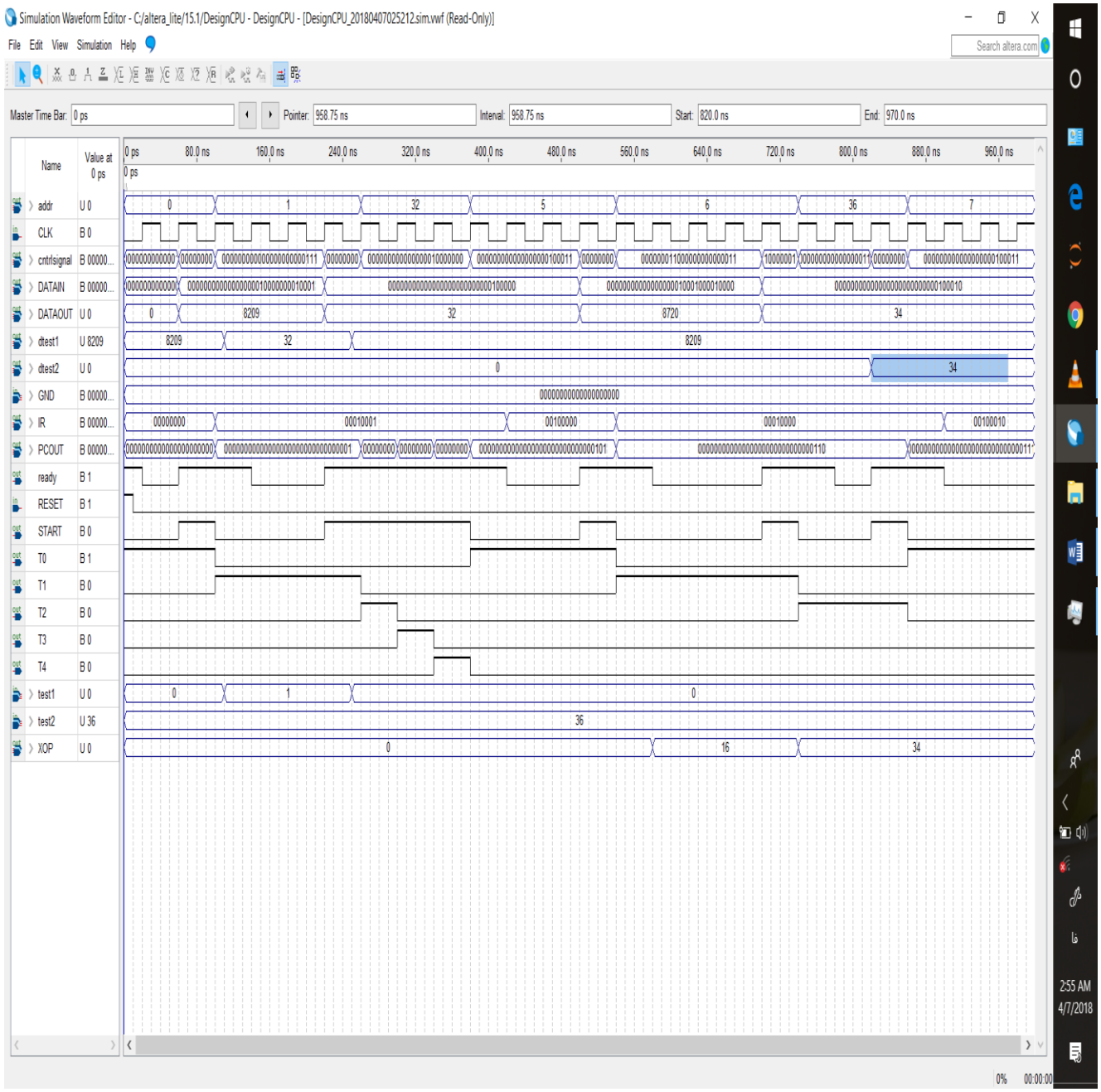
array[6] <= 8'b0010_0010;

کد بالا آدرس 32 را در SPذخیره میکند .

این نقطه ی شروع استک در این تست در نظر گرفته شده است (به عنوان پایه ی استک)که مقدار آن صفر خواهد بود و استک برای پوش کردن های بعدی ابتدا 4 واحد به ادرس خود اضافه میکند که همانطور که میبینید مقدار 34 در نهایت در خانه ی 36 حافظه ذخیره شده است.

خروجی این قطعه دستور را در شبیه ساز کورارتیوس میبینیم.

درواقع در کد بالا قرار است عدد 34 در خانه ی 36 حافظه (در جایی که پوینر SPبعد از اضافه شدن به عنوان نقطه ی بالای استک اشاره میکند)ذخیره شود که در قسمت ابی رنگ این نتیجه مشهود است.


Simulation Waveform Editor - C:/altera_lite/15.1/DesignCPU - DesignCPU - [DesignCPU_20180407025212.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 958.75 ns
Interval: 958.75 ns
Start: 820.0 ns
End: 970.0 ns
Name
Value at 0 ps
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
addr U 0
CLK B 0
cntrlsignal B 00000...
DATAIN B 00000...
DATAOUT U 0
dtest1 U 8209
dtest2 U 0
GND B 00000...
IR B 00000...
PCOUT B 00000...
ready B 1
RESET B 1
START B 0
T0 B 1
T1 B 0
T2 B 0
T3 B 0
T4 B 0
test1 U 0
test2 U 36
XOP U 0
2:55 AM
4/7/2018
0%
00:00:00