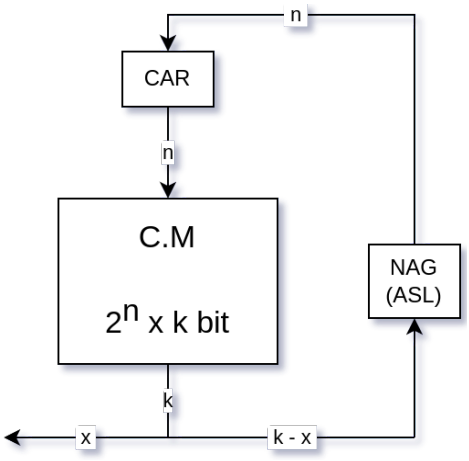
حل مسائل فصل ۷

کلاس حل تمرین معماری کامپیوتر دکتر زینالی، رامتین کوثری،۸ آذر ۱۴۰۳ کلاس ۳۱۰

طراحی واحد CU به روش Micro Program :

در روش میکرو پروگرم بر خلاف روش هارد وایر ما نیاز به معادلات و مدارات که با سیم به همدیگر متصل شده اند نداریم، بلکه یک حافظه جدیدی به نام حافظه کنترلی داریم که با C.M نمایش می دهیم که تمام سیگنال های کنترلی را داخل آن ذخیره کرده ایم. توجه داشته باشید که این حافظه کنترلی از آن حافظه Memory که در معماری Memory Reference یا M.R داشتیم متفاوت است.

حافظه کنترلی مانند حافظه موری نیست و قابلیت Write ندارد، بلکه قابلیت Program شدن دارد، یعنی در حین فعال بودن داده ای در آن نوشته نمی شود بلکه داده ها از قبل در حالتی که خاموش است روی آن پروگرم شده است، به این نوع از موری PROM میگوییم که همان ROM می باشد ولی حافظه موری RAM بود البته می تواند از جنس PAL و یا PLA نیز باشد.



مجموعه واحد CU به روش میکرو پروگرم به شکل روبرو است :

در میانه، ما یک حافظه کنترلی داریم (C.M) که 2^n خانه دارد و هر خانه k بیت می باشد که به هر خانه، **کلمه کنترلی** میگوییم.

یادتان باشد که از این روش برای ذخیره عملیات ها استفاده می کنیم پس هر خانه این حافظه حافظه کنترلی که k بیت است باید شامل ۲ دسته اطلاعات باشد، **اطلاعات عملیاتی** و **اطلاعات حرکتی**، حال چرا این ۲ دسته را داریم ؟ وقتی زمان بندی ای به صورت زیر داشته باشیم :

$$T_0 : AR \leftarrow PC \qquad T_1 : IR \leftarrow M[AR] \ , \ PC \leftarrow PC + 1$$

دو دسته عملیات داریم، یکی آنکه به ما میگوید از T_0 برویم به T_1 که داریم حرکت انجام می دهیم

و دیگری آنکه عملیات $AR \leftarrow PC$ را باید انجام دهیم. حال ممکن است دسته عملیاتی ۲ بخش باشد مانند عملیات در T_1.

توجه داشته باشید که از k بیت ما x بیت را به بخش عملیاتی اختصاص میدهیم (یعنی x بیت به DPU می رود) از طرفی پس بخش حرکتی $k - x$ بیت خواهد بود که به کمک آن باید حرکت کنیم. حال اینکه چگونه و به کجا حرکت کنیم را ماژول های NAG و CAR برای ما مشخص میکنند و به کمک آنها می توانیم بین خانه های حافظه کنترلی حرکت کنیم.

وقتی در سوال امتحان از ما میخواهند که یک DPU را به روش میکرو پروگرم طراحی کنیم و سپس کلمه کنترلی را به دست بیاوریم باید در شکل بالا مقادیر k، x و n را به دست بیاوریم تا بتوانیم اندازه کلمه کنترلی را بدست آوریم. برای به دست آوردن این مجهولات باید از بخش حرکتی به این اعداد برسیم. همانطور که گفته شد بخش حرکتی ما $k - x$ بیت می باشد که این بیت ها به ماژول NAG یا Next Address Generator ما می روند که آدرس بعدی را می سازد که n بیت است و این n بیت به ماژول CAR می روند. برای حل سوال این بخش گفتیم که باید متغیر هایمان را بدست بیاوریم، **برای اینکار ابتدا از بخش عملیاتی شروع میکنیم**، برای نوشتن بخش عملیاتی ما نیاز داریم که RTL اجرای دستورات DPU مان را داشته باشیم که با توجه به جزوه های قبل به این شکل می باشد :

Fetch : $IR \leftarrow M[PC] \ , \ PC \leftarrow PC + 1$

$T_0 : \quad AR \leftarrow PC$

$T_1 : \quad DR \leftarrow M[AR] \ , \ PC \leftarrow PC + 1$

Decode : $I \leftarrow IR_{(15)} \ , \ D_0 D_1 D_2 \dots D_7 \leftarrow IR_{(14-12)} \ , \ AR \leftarrow IR_{(11-0)}$

$T_2 : \quad I \leftarrow DR_{(15)} \ , \ D_0 D_1 D_2 \dots D_{15} \leftarrow DR_{(14-11)} \ , \ AR \leftarrow DR_{(10-0)}$

Effective Address : $if\ (I = 1)\ then\ AR \leftarrow M[AR]$

$I'T_3 :\quad NOP$

$IT_3 :\quad DR \leftarrow M[AR]$

$IT_4 :\quad AR \leftarrow DR$

Execute : $D_0 D_1 D_2 \dots D_7$

$D_0(ADD) :\quad AC \leftarrow AC + M[AR]$

$D_0T_5 :\quad DR \leftarrow M[AR]$

$D_0T_6 :\quad AC \leftarrow AC + DR\ ,\ SC \leftarrow 0$

$D_1(Branch) :\quad if\ (AC < 0)\ then\ PC \leftarrow AR$

$sD_1T_5 :\quad PC \leftarrow AR$

$D_1T_5 :\quad SC \leftarrow 0$

$D_2(Store) :\quad M[AR] \leftarrow AC$

$D_2T_5 :\quad DR \leftarrow AC$

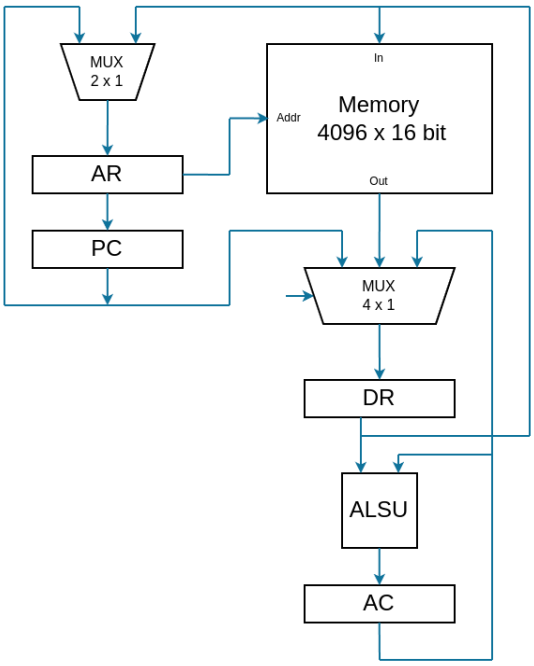
$D_2T_6 :\quad M[AR] \leftarrow DR\ ,\ SC \leftarrow 0$

$D_3(Exchange) :\quad AC \leftarrow M[AR]\ ,\ M[AR] \leftarrow AC$

$D_3T_5 :\quad DR \leftarrow M[AR]$

$D_3T_6 :\quad AC \leftarrow DR\ ,\ DR \leftarrow AC$

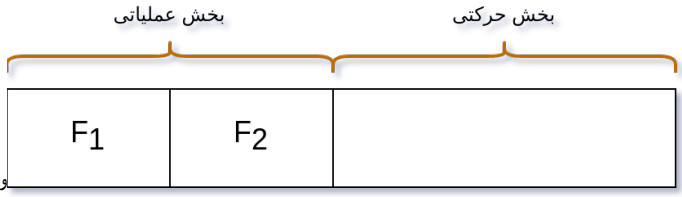
$D_3T_7 :\quad M[AR] \leftarrow DR\ ,\ SC \leftarrow 0$



حال که RTL اجرای دستورات را نوشتیم (در امتحان در بخش الف سوال از قبل نوشته اید) باید تعیین کنیم که بخش عملیاتی ما چند قسمت است، چگونه ؟ به این شکل که باید RTL هایی که نوشته ایم را بررسی کنیم و ببینیم **حداکثر چند دستور با یکدیگر موازی انجام شده اند**، توجه داشته باشید که مبحثی به نام M.I یا Micro Instruction یا ریز دستور داریم مثلا در $DR \leftarrow AC\ ,\ AC \leftarrow DR$ ، ما ۲ ریز دستور داریم که میتوانی به شکل $DRTAC$ و $ACTDR$ نمایش دهیم. نکته مهمی که باید در نظر بگیرید این است که **فلگ ها هیچ کدام ریز دستور نیستند.**

نکته : ریز دستور ها دستوراتی هستند که روی DPU جریان پیدا میکنند و یا دستور عملیاتی هستند نه حرکتی ($SC \leftarrow 0$ حرکتی است زیرا زمان را به T_0 می برد ولی $PC \leftarrow PC + 1$ چون عملیاتی است ریز دستور حساب می شود)

با نگاه کردن به RTL اجرای دستورات نوشته شده درمیابیم که حداکثر تعداد ریز دستور که موازی اجرا شده اند، ۲ ریز دستور در یک کلاک است پس بخش عملیاتی کلمه کنترلی ما ۲ بخش دارد که این ۲ بخش را F_1 و F_2 می نامیم. تا اینجا بخش عملیاتی به شکل زیر طراحی شده است :



حال باید بخش حرکتی را مشخص کنیم ولی پیش از آنکه این بخش را مشخص کنیم باید ابتدا انواع بخش های حرکتی را بدانیم و سپس تمام بخش عملیاتی مان را به صورت Micro Instruction یا ریز دستور بنویسیم و سپس دریابیم که بخش حرکتی چند بیت است.

همچنین توجه داشته باشید که بخش عملیاتی ما x بیت بود و حال باید این x بیت را مشخص کنیم پس باید ببینیم F ها چند بیت هستند :

مرحله ۱ : ابتدا تمامی ریز دستور هایمان را از مرحله Fetch تا آخرین مرحله اجرای دستور بیرون میکشیم و نام گذاری میکنیم :

$AR \leftarrow PC \quad \Rightarrow \quad PCTAR$

F_1	F_2
NOP PCTAR Read DRTAR ~~ADD~~ ARTPC ACTDR Write DRTAC	NOP INC ACTDR

ead
IC
$RTAR$
DD
$RTPC$
$CTDR$
$rite$
$RTAC$

F_1	F_2
NOP PCTAR Read DRTAR ARTPC ACTDR Write DRTAC	NOP INC ACTDR ADD

حال باید مشخص کنیم که این دستورات هر کدام در کدام بخش از بخش های F_1 یا F_2 اجرا میشوند، برای این کار یک جدول میکشیم و دوباره RTL را 4 bit 3 bit 2 bit بررسی میکنیم تا ببینیم هر ریز دستور در کدام بخش انجام می شود :

F_1	F_2
NOP PCTAR Read DRTAR ADD ARTPC ACTDR Write DRTAC	NOP INC ACTDR
4 bit	2 bit

همانطور که مشاهده میکنید دستوراتی که در بخش اول اجرا شده اند را در F_1 و دستوراتی که در بخش دوم انجام شده اند را در F_2 نوشته ایم. حال F_1 ما ۹ ریز دستور را شامل شده و این ۹ حالت را با چند بیت می توان کد کرد ؟ با ۴ بیت زیرا $۲^۳ < ۹ < ۲^۴$ و همچنین F_2 را چون ۳ حالت دارد می توانیم با ۲ بیت کد کنیم، ۴ بیت بخش اول و ۲ بیت بخش دوم در مجموع میشود ۶ بیت پس بخش عملیاتی را توانستیم با ۶ بیت کد کنیم ولی آیا این ۶ بیت بهینه است ؟ بخش اول از ۱۶ بیت ۹ بیت استفاده شده پس ۷ بیت پرت دارد همچنین بخش دوم ۳ بیت از ۴ بیت استفاده شده پس ۱ بیت پرت دارد پس **در مجموع ۸ بیت پرت و بلا استفاده** داریم. چگونه میتوانیم این بیت های پرتمان را کاهش دهیم ؟ به این شکل :

اگر بخواهیم مثلا D_0T_5 را به صورت موازی بنویسیم داریم :

$$D_0T_5: \quad DR \leftarrow M[AR] \ , \ NOP$$

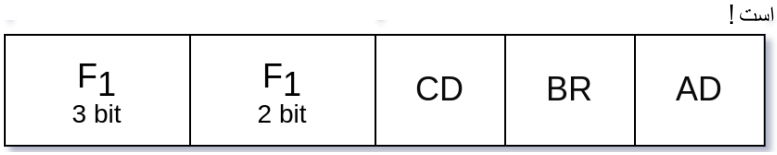
یعنی در بخش دوم RTL ما هیچ کاری انجام نداده ایم و این درست است همچنین این نکته هم به یاد داشته باشید که ما دستور NOP را در هر دو بخش نوشته ایم زیرا مثلا در $I'T_3$: NOP هر ۲ بخش ما هیچ کاری انجام نمیدهد و NOP است. حال آیا قبول دارید میتوانیم D_0T_5 را به این شکل بنویسم :

$$D_0T_5: \quad NOP \ , \ DR \leftarrow M[AR]$$

یعنی بخش اول هیچ کاری انجام نده ولی بخش دوم بیا Read انجام بده. اگر این کار را بکنیم انگار میتوانیم دستور Read را به بخش دوم جدول انتقال دهیم و آنجا بنویسیم اگر این کار را بکنیم بخش F_1 ما ۸ حالت می شود پس می توان آن را با ۳ بیت کد کرد و همچنین بخش F_2 ما ۴ حالت میشود که با همان ۲ بیت قبل کد میشود پس در مجموع ۳ بیت برای بخش اول و ۲ بیت برای بخش دوم داریم یعنی بخش عملیاتی ما ۵ بیت میشود پس ۱ بیت توانستیم صرفه جویی کنیم. حال مسئله ای پیش می آید و آن این است که نمی توانیم ریز دستور Read را به بخش دوم انتقال دهیم زیرا برای مثال در T_1 بخش دوم عملیات را ریز دستور INC اشغال کرده پس نمی توانیم جا به جا کنیم بنابراین باید به دنبال ریز دستوری بگردیم که بخش دوم عملیات در آن زمان بندی NOP باشد (یا هرچه غیر از ریز دستور باشد) که بهترین گزینه ریز دستور ADD در D_0T_6 می باشد. زیرا بخش دوم عملیاتی آن فلگ است و فلگ را ریز دستور حساب نمی کنیم بنابراین شکل جدول بخش عملیاتی کلمه کنترلی ما به شکل صفحه بعد می شود :

همانطور که میبینید صرفا با انتقال ریز دستور
ADD به بخش دوم، ۸ بیت را صرفه جویی کردیم

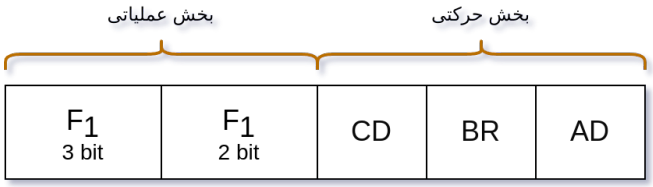
این کار دقیقا کار مهندسی کردن است !



حال که مشخص شد بخش عملیاتی ما ۵ بیت است باید به سراغ بخش حرکتی کلمه کنترلی برویم، پیش از آنکه به این بخش بپردازیم باید ببینیم که خود این بخش چند قسمت است یعنی ببینیم چه عواملی در حرکت ما موثر اند و حرکت ها را تعیین میکنند :

- **نوع حرکت (BR) :** در این بخش می توانیم انواع حرکت هایمان را با توجه به RTL اجرای دستوراتی که داریم مشخص کنیم که میتواند شامل Next ،Jump Call ،Return و Map باشد.
- **شرط حرکت (CD) :** هر شرطی که باعث حرکت متفاوتی می شود مانند I یا s، اگر شرطی نداشتیم با U نمایش می دهیم که به معنای Unconditional می باشد یعنی بدون شرط.
- **آدرس حرکت (AD) :** آدرس حرکت ما که اگر شکل اول این جزوه را ببینید می بینید که n بیت است و برای مثال می تواند آدرس مرحله Fetch و یا ... باشد.

پس بخش حرکتی ما خودش دارای ۳ بخش است بنابراین داریم :

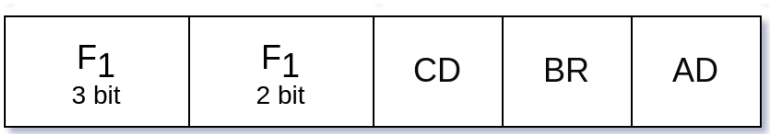


حال برای اینکه ببینیم بخش حرکتی چند بیت است باید ببینیم هرکدام از ۳ بخش آن چند بیت است یعنی مثلا بخش CD چه تعداد شرط دارد و آن تعداد را با چند بیت می توان کد کرد یا همچنین مقلا بخش BR چه تعداد حرکت داریم و آن را نیز با چه تعداد می توان کد کرد و ... . برای مشخص کردن بیت های لازم باید قسمت اصلی سوال ج) امتحان یعنی "طراحی واحد CU به روش میکرو پروگرم" را انجام دهیم، برای این کار میدانیم که فرمت بالا فرمت کلمه کنترلی می باشد و در هر کدام از خانه های حافظه کنترلی یک خط از RTL اجرای دستورات قرار دارد (انگار برنامه نوشته اید و خط های برنامه را ذخیره کرده اید) پس بیایم تمامی دستورات را به همان شکلی که در حافظه کنترلی قرار میگیرند بنویسیم، گفتیم هر خانه حافظه کنترلی شامل کلمه کنترلی می شود پس به عنوان مثال برای خط اول Fetch داریم :

$$Fetch: \quad PCTAR \quad NOP \quad U \quad JMP \quad NEXT$$

یعنی بیا ریز دستو PCTAR را انجام بده بدون هیچ شرطی و سپس برو (بپر یا جامپ کن) به آدرس بعدی

توجه داشته باشید که در هر خطی که می نویسیم باید ابتدا شرط را چک کنیم یعنی هیچ کاری نکن (۲ بخش اول NOP) شرط رو چک کن بعد ادامه رو انجام بده. حال کل دستورات را می نویسیم :



$$\begin{aligned} Fetch: \quad & PCTAR \quad NOP \quad U \quad JMP \quad NEXT \\ & Read \quad INCPC \quad U \quad JMP \quad NEXT \\ Decode: \quad & DRTAR \quad NOP \quad U \quad JMP \quad NEXT \end{aligned}$$

$E.A:$	NOP	NOP	I	JMP	P_1
$P_1:$	$Read$	NOP	U	JMP	$NEXT$
	$DRTAR$	NOP	U	MAP	$-$

توجه کنید که در ۲ بخش اول دستور E.A هیچ کاری انجام نمی دهیم چون شرط داریم و مستقیم می خواهیم شرط را بررسی کنیم و اگر شرط برقرار بود میگوییم برود به P_1 که خودمان طراحی کرده ایم تا بتوانیم در محیطی جدا مانند (Fetch یا Decode) شرط را انجام دهیم (به گونه ای Label هایی که در سمت چپ تعریف میکنیم همگی انگار نام توابع برنامه نویسی ما هستند). در **۲ خط سبز رنگ** ما ابتدا پیش از انجام عملیاتی بررسی کردیم آیا شرط برقرار است و اگر برقرار بود رفتیم به P_1 که خط بعد تعریف کردیم که خود P_1 در ۲ خط نوشته می شود منتهی اگر شرط برقرار نبود چه ؟ آن را هنوز انجام نداده ایم ولی جلوتر می نویسیم. پیش از نوشتن آن به یک نقطه در **خط صورتی رنگ** باید توجه شود و آن هم MAP می باشد که چرا آدرس آن خط تیره است و هیچ آدرسی مشخص نشده ؟ زیرا وقتی مرحله Effective Address را انجام می دهیم سپس باید دستور مربوطه را انجام دهیم ($D_0 D_1 D_2 \dots D_7$). حال هر کدام از این دستور ها در خانه به خصوصی از حافظه ذخیره شده اند و ما به صورت مستقیم نمی دانیم که آدرس آن کجاست ولی وقتی می نویسیم MAP انگار داریم به ماژول NAG یا Next Address Generator که آدرس بعدی را می ساخت فرمان می دهیم که آدرس دستور مربوطه را به ما بدهد که به همین دلیل است ما در بخش آدرس از خط تیره استفاده کرده ایم. پس :

نکته : یادتان باشد که فقط در آخرین مراحل Effective Address (اگر داشتیم، اگر نداشتیم در مرحله Decode) باید Branch یا BR را به صورت MAP بنویسیم.

حال اگر یادتان باشد حاتی که شرط برقرار نباشد را ننوشتیم پس به آن ۳ خط رنگی ۱ خط دیگر که با آبی نمایش داده شده اضافه می شود به شکل زیر :

$E.A:$	NOP	NOP	I	JMP	P_1
	NOP	NOP	U	MAP	$-$
$P_1:$	$Read$	NOP	U	JMP	$NEXT$
	$DRTAR$	NOP	U	MAP	$-$

همانطور که می بینید چه شرط برقرار باشد چه نباشد در آخرین مرحله باید MAP کنیم تا دستورات اجرا شوند. حال میرسیم به اجرای خود هر دستور پس داریم :

$D_0(ADD):$	$READ$	NOP	U	JMP	$NEXT$
	NOP	ADD	U	JMP	$Fetch$

و به این شکل دستور اول را نوشتیم ولی ۲ نکته دارد، اول آنکه Add را در بخش دوم نوشتیم پس حواستان باشد، دوم آنکه در آخرین مرحله اگر یادتان باشد SC را صفر می کردیم یعنی می رفتیم به مرحله Fetch، پس در قسمت آدرس Fetch را نوشتیم که به آنجا بازگردیم. حال به ادامه نوشتن دستورات می پردازیم :

$D_1(Branch):$	NOP	NOP	s	JMP	P_2
	NOP	NOP	U	JMP	$Fetch$
$P_2:$	$ARTPC$	NOP	U	JMP	$Fetch$

همانطور که مشاهده می کنید در این دستور شرط ما s بود، پس چون شر داشتیم در خط اول هیچ کاری نمیکردیم (NOP در هر بخش) و فقط شرط را بررسی می کردیم، اگر شرط برقرار بود می رفت به P_2 و خط **سبز رنگ** را اجرا میکرد ولی اگر نبود می رفت به NEXT و خط **صورتی رنگ** را اجرا میکرد. ادامه دستورات :

$D_2(Store):$	$ACTDR$	NOP	U	JMP	$NEXT$
	$Write$	NOP	U	JMP	$Fetch$

$$D_3(Exchange): \quad READ \quad NOP \quad U \quad JMP \quad NEXT$$
$$ACTDR \quad DRTAC \quad U \quad JMP \quad NEXT$$
$$Write \quad NOP \quad U \quad JMP \quad Fetch$$

حال باید مشخص کنیم که هر ۳ بخش عملیاتی ما چند بیت است :

- **به دست آوردن چند بیت بودن بخش CD :** با توجه به میکرو پروگرم نوشته شده بررسی میکنیم که در مجموع چند شرط داریم که اگر ملاحظه کنید میبینید که فقط I، s و U را داریم که ۳ حالت است پس با ۲ بیت کد می شود.
- **به دست آوردن چند بیت بودن بخش BR :** نگاه می کنیم و می بینیم که در ستون BR چند نوع حرکت داریم که ملاحظه می کنید فقط JMP و MAP را داریم پس ۲ حالت است و با ۱ بیت کد می شود.
- **به دست آوردن چند بیت بودن بخش AD :** برای به دست آوردن تعداد آدرس ها باید ببینیم که کل RTL اجرای دستورات ما در جدول میکرو پروگرم در مجموع چند کلاک طول کشیده (تعداد خط ها) که داریم :

۷ کلاک بدنه مشترک + ۲ کلاک D_0 + ۳ کلاک D_1 + ۲ کلاک D_0 + ۳ کلاک D_0 = ۱۷ کلاک

یعنی بخش AD ما **باید ۵ بیت باشد** که بتواند ۱۷ حالت را کد کند، البته می توانیم خلاصه تر بنویسیم که ۱۶ حالت شود که بتوان با ۴ بیت کد کرد که برای آن به جزوه مراجعه کنید.

حال کلمه کنترلی ما به شکل روبرو خواهد شد :

پس در کل کلمه کنترلی ما ۱۳ بیت میشود (قسمت د در امتحانات که از ما خواسته "کلمه کنترلی را به دست آورید").

با عملیات های انجام شده داریم :

$$n = 5 \ , \ x = 5 \ , \ k = 13$$

حال باید Decoder های DPU طراحی شود (صفحه ۲۳ جزوه). اگر سوال آمد که "Sequencer را رسم کنید" باید صفحه۲۲ جزوه رسم شود.