بسمه تعالی

تمرین سری 4 معماری

طراحی پردازنده با روش میکرو پروگرمینگ

محمد صادق خراسانی

95108008

فرضیات مورد نیاز

قبل از هر کاری برای پیشروی کار میبایست برای میکرو اپریشن ها نام انتخاب کرد و فرمت میکرو اینستراکشن ها را تعیین کرد.

به همین دلیل من برای هر میکرو اینستراکشن (یا ورد contrl rom) 3فیلد برای حداکثر 3 میکرو اپریشن همزمان انتخاب کردم و سپس مانند کتاب 2 بیت برای conditionو2بیت برای branchدر نظر گرفتم.در 7 بیت کم ارزش ورد رام هم ادرس محل بعدی که باید از رام خوانده شود قرار دارد.

همانطور که تخمین زده شده بود و همانطور که در نهایت دیده شد نیازی به رام با بیشتر از 7 بیت ادرس نبود جرا که با 7 بیت میتوانیم 128 خانه را ادرس دهی کنیم در حالی که بیشتر از 90 خانه برای دستورات داده شده نیازی نبود.

به همین دلیل رام با ورد 23 بیت و ادرس دهی 7 بیتی برای کنترل یونیت انخاب کرده ام.

سپس تمام دستورات را با نام میکرو اپریشن ها در اکسل وارد کرده و کد وریلاگ مربوط به آن را در فرمولی در اکسل که بسته به فیلد های نام میکرو اپریشن هاست گرفته ام تا قابلیت تغییر دادن میکرو اینستراکشن ها را داشته باشم.

فایل مربوط به اکسل پیوست نیز شده است.

قبل از نوشتن میکرو اینستراکشن ها طبیعتا نیاز بود برای هر کدام از فیلد های F1وf2وf3میکرو اپریشن ها را تعینن کنیم و برای هر کدام نامی بگذاریم .

در زیر تقسیم بندی تمام میکرو اپریشن های استفاده شده را در 3 فیلد جدا میبینید .

در این تقسیم بندی سعی شده است از حداکثر موازات بهره گرفته شود.

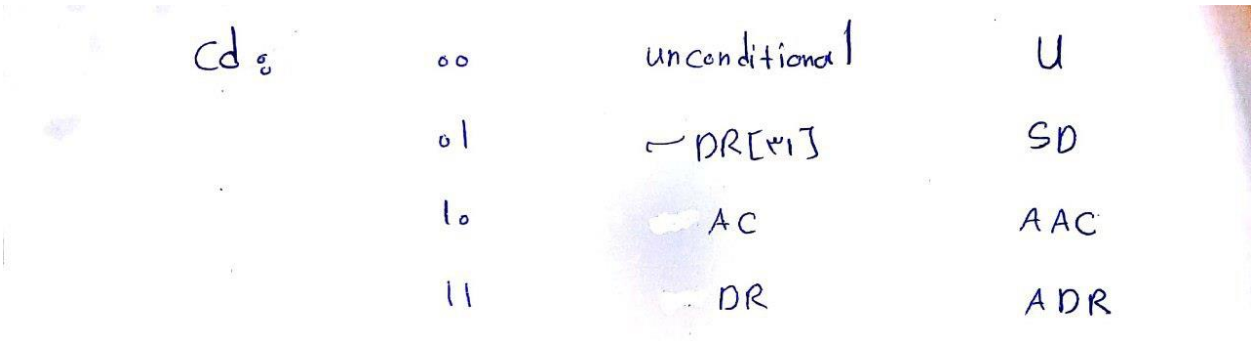
علاوه بر این چون هر کدام از فیلد ها f1,f2,f3 4 بیتی میباشند ینی جمعا 48 دستور را میتوانند ساپورت کنند که در تقسیم بندی بنده 46 جایگاه پر شده است.

F_1 :

0000	Nop	"Nothing"
0001	RIR	IR ← M[PC]
0010	MSPTDR	DR ← M[SP]
0011	MSPTAC	AC ← M[SP]
0100	CLROP$_{r-1}$	OP$_{r-1}$ ← 0
0101	OPTLV	LV ← OP
0110	PCTT$_1$	T$_1$ ← PC
0111	And	TMP ← AC ∧ DR
1000	OR	TMP ← AC ∨ DR
1001	MLVTDR	DR ← M[LV]
1010	MPCTSP	SP ← M[PC]
1011	FlagON	wFlag ← 1
1100	MSPTSP	SP ← M[SP]
1101	CLROP$_{r-2}$	OP$_{r-2}$ ← 0
.....	.....	.....
1111	MSPTLV	LV ← M[SP]

F_2 s			
	0000	NOP	"Nothing"
	0001	INCPC	PC ← PC+1
	0010	INC4SP	SP ← SP+4
	0011	DEC4SP	SP ← SP−4
	0100	TMPTMSP	M[SP] ← TMP
	0101	TMPTMLV	M[LV] ← TMP
	.....	.....	.....
	0111	ACTTMP	TMP ← AC
	1000	DRTMLV	M[LV] ← DR
	1001	MPCTPC	PC ← M[PC]
	1010	LVTT$_2$	T$_2$ ← LV
	1011	SPTAC	AC ← SP
	1100	TMPTSP	SP ← TMP
	1101	T$_2$TMSP	M[SP] ← T$_2$
	1110	TMPTDR	DR ← TMP
	1111	T$_2$TSP	SP ← T$_2$

Fn:			
	0000	Nop	"Nothing"
	0001	MTOP$_0$	OP$_0$ ← M[PC]
	0010	OPTMSP	M[sp] ← OP
	0011	SelfAddPC	PC ← PC + M[PC]
	0100	ACMDR	TMP ← AC − DR
	0101	ACSDR	TMP ← AC + DR
	0110	MSPTPC	PC ← M[sp]
	0111	DRTM	M[sp] ← DR
	1000	TMPTAC	AC ← TMP
	1001	OPTAC	AC ← OP
	1010	MCPPTDR	DR ← M[CPP]
	1011	MPCTCPP	CPP ← M[PC]
	1100	MPCTOP$_{1-0}$	OP$_{1-0}$ ← M[PC]
	1101	∈ACTTMP	TMP ← ∈ * AC
	1110	TMPTLV	LV ← TMP
	1111	T$_1$TMSP	M[SP] ← T$_1$



لازم است وضیحاتی راجع به دوبیت CDبیان کنم.

Branchبعدی بسته به شراطی انجام میشود که cdاعلام میدارد.

در طراحی من:

مورد اول یعنی بدون هیچ شرطی branchانجام بشود.

مورد دوم یعنی اگه بیت 31 ام DRصفر بود branchانجام شود

مورد سوم یعنی اگر ACغیر صفر بود برنچ انجام شود

مورد چهارم یعنی اگر DRغیر صفر بود برنچ انجام شود.

دوبیتBRهم حالات jumpوmapوcallوretرا مشخص میکنند

البته در اینجا چون از callوretدر رام استفاده نشده است از دو حالت دو بیت brاستفاده ایی نمیشود اما برای نبستن امکان اضافه کردن حالات جدید در نظر گرفته شده اند

تمامی دستورات مورد نیاز در اکسل وارد شده و رام اماده شده ی خود را در سمت راست آن میبینیم(در شکل زیر)

فایل اکسل با کیفیت بهتر به شکل پی دی اف در فایل های پروژه پیوست شده است.

function	add	F1	F2	F3	cd	br	br_Add	br_add	f1[22:19]	f2[18:15]	f3[14:11]	cd[10:9]	br[8:7]	add[6:0]	verilog codes	
fetch	0	RIR	INCPC	NOP	U	MAP	0	0	01	01	00	00	11	0	array[0] = 23'b00010001000000110000000;	//fetch
nop	1	NOP	NOP	NOP	U	JMP	FETCH	0	00	00	00	00	00	0	array[1] = 23'b00000000000000000000000;	//nop
BIPUSH	2	CLROP3-1	INC4SP	MTOP0	U	JMP	NEXT	3	100	10	01	00	00	11	array[2] = 23'b01000010000100000000011;	//BIPUSH
	3		INCPC	OPTMSP	U	JMP	FETCH	0	00	01	10	00	00	0	array[3] = 23'b00000001001000000000000;	//
GOTO	4			SELFADDPC	U	JMP	FETCH	0	00	00	11	00	00	0	array[4] = 23'b00000000001100000000000;	//GOTO
IADD	5	MSPTDR	DEC4SP		U	JMP	NEXT	6	10	11	00	00	00	110	array[5] = 23'b00100011000000000000110;	//IADD
	6	MTAC			U	JMP	NEXT	7	00	00	00	00	00	111	array[6] = 23'b00000000000000000000111;	//
	7			ACSDR	U	JMP	NEXT	8	00	00	101	00	00	1000	array[7] = 23'b00000000010100000001000;	//
	8		TMPTMSP		U	JMP	FETCH	0	00	100	00	00	00	0	array[8] = 23'b00000100000000000000000;	//
IFEQ	9	MSPTDR	DEC4SP		U	JMP	NEXT	10	10	11	00	00	00	1010	array[9] = 23'b00100011000000000001010;	//IFEQ
	10				ADR	JMP		12	00	00	00	11	00	1100	array[10] = 23'b00000000000011000001100;	//
	11			SELFADDPC	U	JMP	NEXT	12	00	00	11	00	00	1100	array[11] = 23'b00000000001100000001100;	//
	12		INCPC		U	JMP	NEXT	13	00	01	00	00	00	1101	array[12] = 23'b00000001000000000001101;	//
	13		INCPC		U	JMP	FETCH	0	00	01	00	00	00	0	array[13] = 23'b00000001000000000000000;	//
IFLT	14	MSPTDR	DEC4SP		U	JMP	NEXT	15	10	11	00	00	00	1111	array[14] = 23'b00100011000000000001111;	//IFLT
	15				SD	JMP		18	00	00	00	01	00	10010	array[15] = 23'b00000000000001000010010;	//
	16		INCPC		U	JMP	NEXT	17	00	01	00	00	00	10001	array[16] = 23'b00000001000000000010001;	//
	17		INCPC		U	JMP	FETCH	0	00	01	00	00	00	0	array[17] = 23'b00000001000000000000000;	//
	18			SELFADDPC	U	JMP	FETCH	0	00	00	11	00	00	0	array[18] = 23'b00000000001100000000000;	//
If_icmpeq	19	MSPTDR	DEC4SP		U	JMP	NEXT	20	10	11	00	00	00	10100	array[19] = 23'b00100011000000000010100;	//If_icmpeq
	20	MSPTAC	DEC4SP		U	JMP	NEXT	21	11	11	00	00	00	10101	array[20] = 23'b00110011000000000010101;	//
	21			ACMDR	U	JMP	NEXT	22	00	00	100	00	00	10110	array[21] = 23'b00000000010000000010110;	//
	22			TMPTAC	U	JMP	NEXT	23	00	00	1000	00	00	10111	array[22] = 23'b00000000100000000010111;	//
	23				AAC	JMP		25	00	00	00	10	00	11001	array[23] = 23'b00000000000010000011001;	//
	24			SELFADDPC	U	JMP	FETCH	0	00	00	11	00	00	0	array[24] = 23'b00000000001100000000000;	//
	25		INCPC		U	JMP	NEXT	26	00	01	00	00	00	11010	array[25] = 23'b00000001000000000011010;	//
	26		INCPC		U	JMP	FETCH	0	00	01	00	00	00	0	array[26] = 23'b00000001000000000000000;	//
linc	27	CLROP3-1		MTOP0	U	JMP	NEXT	28	100	00	01	00	00	11100	array[27] = 23'b01000000000100000011100;	//linc
	28	OPTLV	INCPC		U	JMP	NEXT	29	101	01	00	00	00	11101	array[28] = 23'b01010001000000000011101;	//
	29	MLVTDR			U	JMP	NEXT	30	1001	00	00	00	00	11110	array[29] = 23'b10010000000000000011110;	//
	30	CLROP3-1	INCPC	MTOP0	U	JMP	NEXT	31	100	01	01	00	00	11111	array[30] = 23'b01000001000100000011111;	//
	31			OPTAC	U	JMP	NEXT	32	00	00	1001	00	00	100000	array[31] = 23'b00000000100100000100000;	//
	32			ACSDR	U	JMP	NEXT	33	00	00	101	00	00	100001	array[32] = 23'b00000000010100000100001;	//
	33		TMPTMLV		U	JMP	FETCH	0	00	101	00	00	00	0	array[33] = 23'b00000101000000000000000;	//

ILOAD	34	CLROP3-1		MTOP0	U	JMP	NEXT	35	100	00	01	00	00	100011	array[34] = 23'b01000000000100000100011;	//ILOAD
	35	OPTLV	INCPC		U	JMP	NEXT	36	101	01	00	00	00	100100	array[35] = 23'b01010001000000000100100;	//
	36	MLVTDR	INC4SP		U	JMP	NEXT	37	1001	10	00	00	00	100101	array[36] = 23'b10010010000000000100101;	//
	37			DRTM	U	JMP	FETCH	0	00	00	111	00	00	0	array[37] = 23'b00000000011100000000000;	//
IStore	38	MSPTDR	DEC4SP		U	JMP	NEXT	39	10	11	00	00	00	100111	array[38] = 23'b00100011000000000100111;	//IStore
	39	CLROP3-1			U	JMP	NEXT	40	100	00	00	00	00	101000	array[39] = 23'b01000000000000000101000;	//
	40			MTOP0	U	JMP	NEXT	41	00	00	01	00	00	101001	array[40] = 23'b00000000000100000101001;	//
	41	OPTLV	INCPC		U	JMP	NEXT	42	101	01	00	00	00	101010	array[41] = 23'b01010001000000000101010;	//
	42		DRTMLV		U	JMP	FETCH	0	00	1000	00	00	00	0	array[42] = 23'b00001000000000000000000;	//
ISUB	43	MSPTDR	DEC4SP		U	JMP	NEXT	44	10	11	00	00	00	101100	array[43] = 23'b00100011000000000101100;	//ISUB
	44	MSPTAC			U	JMP	NEXT	45	11	00	00	00	00	101101	array[44] = 23'b00110000000000000101101;	//
	45			ACMDR	U	JMP	NEXT	46	00	00	100	00	00	101110	array[45] = 23'b00000000010000000101110;	//
	46		TMPTMSP		U	JMP	FETCH	0	00	100	00	00	00	0	array[46] = 23'b00000100000000000000000;	//
DUP	47	MSPTDR	INC4SP		U	JMP	NEXT	48	10	10	00	00	00	110000	array[47] = 23'b00100011000000000110000;	//DUP
	48			DRTM	U	JMP	FETCH	0	00	00	111	00	00	0	array[48] = 23'b00000000011100000000000;	//
LAND	49	MSPTDR	DEC4SP		U	JMP	NEXT	50	10	11	00	00	00	110010	array[49] = 23'b00100011000000000110010;	//LAND
	50	MSPTAC			U	JMP	NEXT	51	11	00	00	00	00	110011	array[50] = 23'b00110000000000000110011;	//
	51	AND			U	JMP	NEXT	52	111	00	00	00	00	110100	array[51] = 23'b01110000000000000110100;	//
	52		TMPTMSP		U	JMP	FETCH	0	00	100	00	00	00	0	array[52] = 23'b00000100000000000000000;	//
swap	53	MSPTDR	DEC4SP		U	JMP	NEXT	54	10	11	00	00	00	110110	array[53] = 23'b00100011000000000110110;	//swap
	54	MSPTAC			U	JMP	NEXT	55	11	00	00	00	00	110111	array[54] = 23'b00110000000000000110111;	//
	55		INC4SP	DRTM	U	JMP	NEXT	56	00	10	111	00	00	111000	array[55] = 23'b00000010011100000111000;	//
	56		ACTTMP		U	JMP	NEXT	57	00	111	00	00	00	111001	array[56] = 23'b00000111000000000111001;	//
	57		TMPTMSP		U	JMP	FETCH	0	00	100	00	00	00	0	array[57] = 23'b00000100000000000000000;	//
wide	58	FLAGON			U	JMP	FETCH	0	1011	00	00	00	00	0	array[58] = 23'b10110000000000000000000;	//wide
IOR	59	MSPTDR	DEC4SP		U	JMP	NEXT	60	10	11	00	00	00	111100	array[59] = 23'b00100011000000000111100;	//IOR
	60	MSPTAC			U	JMP	NEXT	61	11	00	00	00	00	111101	array[60] = 23'b00110000000000000111101;	//
	61	OR			U	JMP	NEXT	62	1000	00	00	00	00	111110	array[61] = 23'b10000000000000000111110;	//
	62		TMPTMSP		U	JMP	FETCH	0	00	100	00	00	00	0	array[62] = 23'b00000100000000000000000;	//

LDC_W	63			MPCTCPP	U	JMP	NEXT	64	00	00	1011	00	00	1000000	array[63] = 23'b00000000101100001000000;	//LDC_W
	64		INC4SP	MCPPTDR	U	JMP	NEXT	65	00	10	1010	00	00	1000001	array[64] = 23'b00000010101000001000001;	//
	65			DRTM	U	JMP	FETCH	0	00	00	111	00	00	0	array[65] = 23'b00000000011100000000000;	//
POP	66		DEC4SP		U	JMP	FETCH	0	00	11	00	00	00	0	array[66] = 23'b00000011000000000000000;	//POP
LDASP	67	MSPTSP	INCPC		U	JMP	NEXT	68	1100	01	00	00	00	1000100	array[67] = 23'b11000001000000001000100;	//LDASP
	68		INCPC		U	JMP	NEXT	69	00	01	00	00	00	1000101	array[68] = 23'b00000001000000001000101;	//
	69		INCPC		U	JMP	NEXT	70	00	01	00	00	00	1000110	array[69] = 23'b00000001000000001000110;	//
	70		INCPC		U	JMP	FETCH	0	00	01	00	00	00	0	array[70] = 23'b00000001000000000000000;	//
ENVOKE	71	PCTT1	LVTT2		U	JMP	NEXT	72	110	1010	00	00	00	1001000	array[71] = 23'b01101010000000001001000;	//ENVOKE
	72		MPCTPC		U	JMP	NEXT	73	00	1001	00	00	00	1001001	array[72] = 23'b00001001000000001001001;	//
	73	CLROP3-2	INCPC	MPCTOP1-0	U	JMP	NEXT	74	1101	01	1100	00	00	1001010	array[73] = 23'b11010001110000001001010;	//
	74			OPTAC	U	JMP	NEXT	75	00	00	1001	00	00	1001011	array[74] = 23'b00000000100100001001011;	//
	75		SPTAC	4ACTTMP	U	JMP	NEXT	76	00	1011	1101	00	00	1001100	array[75] = 23'b00001011110100001001100;	//
	76		TMPTDR		U	JMP	NEXT	77	00	1110	00	00	00	1001101	array[76] = 23'b00001110000000001001101;	//
	77			ACMDR	U	JMP	NEXT	78	00	00	100	00	00	1001110	array[77] = 23'b00000000010000001001110;	//
	78		INCPC	TMPTLV	U	JMP	NEXT	79	00	01	1110	00	00	1001111	array[78] = 23'b00000001111000001001111;	//
	79			MPCTOP1-0	U	JMP	NEXT	80	00	00	1100	00	00	1010000	array[79] = 23'b00000000110000001010000;	//
	80		INCPC	OPTAC	U	JMP	NEXT	81	00	01	1001	00	00	1010001	array[80] = 23'b00000001100100001010001;	//
	81		SPTAC	4ACTTMP	U	JMP	NEXT	82	00	1011	1101	00	00	1010010	array[81] = 23'b00001011110100001010010;	//
	82		TMPTDR		U	JMP	NEXT	83	00	1110	00	00	00	1010011	array[82] = 23'b00001110000000001010011;	//
	83		INCPC	ACSDR	U	JMP	NEXT	84	00	01	101	00	00	1010100	array[83] = 23'b00000001010100001010100;	//
	84		TMPTSP		U	JMP	NEXT	85	00	1100	00	00	00	1010101	array[84] = 23'b00001100000000001010101;	//
	85		DEC4SP	T1TMSP	U	JMP	NEXT	86	00	11	1111	00	00	1010110	array[85] = 23'b00000011111100001010110;	//
	86		T2TMSP		U	JMP	FETCH	0	00	1101	00	00	00	0	array[86] = 23'b00001101000000000000000;	//
RETURN	87		LVTT2		U	JMP	NEXT	88	00	1010	00	00	00	1011000	array[87] = 23'b00001010000000001011000;	//RETURN
	88	MSPTLV			U	JMP	NEXT	89	1111	00	00	00	00	1011001	array[88] = 23'b11110000000000001011001;	//
	89			MSPTPC	U	JMP	NEXT	90	00	00	110	00	00	1011010	array[89] = 23'b00000000011000001011010;	//
	90		T2TSP		U	JMP	FETCH	0	00	1111	00	00	00	0	array[90] = 23'b00001111000000000000000;	//

در ادامه هم به ساختن سخت افزار مورد نیاز پرداخته شده است.

نکته ی مهم و قابل توضیح نحوه ی هندل دستور wideمیباشد.

با امدن این دستور فلیپ فلاپی به نامflagیک میشود.

رجیستر هایی که قرار است oprandدوبایتی دریافت کنند از flag ورودی میگیرند و در صورت یک بودن آن 2 بایت از خروجی مموری لود میکنند و در غیر این صورت یک بایت لود میکنند.

علاوه بر این پس از این اتفاق (یعنی بعد از لود شدن دوبایتی)مقدار flagصفر میشود و pcیکی زیاد میشود.

این مورد را در سخت افزارپیاده سازی کرده ام.

توضیحاتی در مورد قسمت های مختلف سخت افزار:

نکته ی اول اینکه دیتاپس من نیاز به کمی تغییر داشت چرا که در تمرین قبل نیاز به این نبود که از رجیستر sp و pcروی باس اصلی خروجی گرفته شود. اما در اینجا نیاز به این قضیه در دستور envokeایجاد شد و نتیجتا دیتا پس کمی میبایست تغییر میکرد.

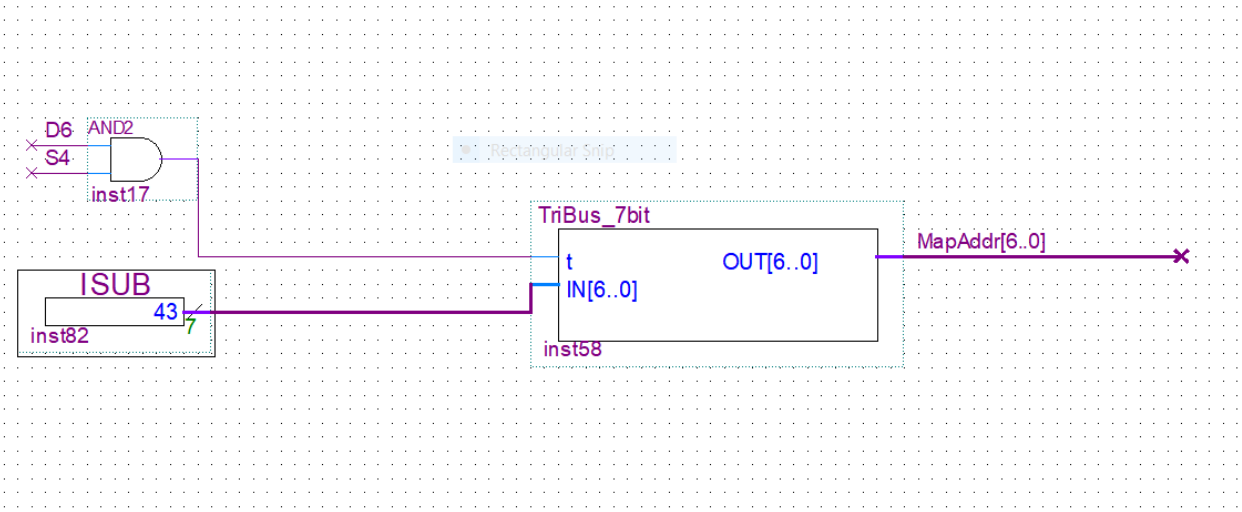
در مورد کنترل یونیت لازم است مطالبی گفته شود.

یک اینکه در کنترل یونیت از یک رام و یک رجیستر ادرس برای رام و دو مالتی پلکسر برای انتخاب ادرس بعدی رام و واحد mapو واحد incrementorو واحد genSignalاستفاده شده است .

واحد mapدستورات را از رجیستر IRگرفته و به عنوان خروجی ادرس نقطه ی شروع دستور مورد نظر در رام را تولید میکند.

در ساخت این واحد از CONSTANTهای کوارتیوس (LMP)استفاده شده است که به را حتی قابلیت تغییر ادرس شروع دستور دلخواه را به ما میدهد.

شیوه ی کار به این صورت است که مانند دیکود کردن هر دستور در تمرین قبل اگر بیت مربوط به یک دستور یک شد کنترل بافری که مربوط به این دستور هست یک میشود در نتیجه مقدار کانستنت مربوط به این دستور که همان نقطه ی شروع دستور مورد نظر در رام میباشد بر روی باس خروجی میرود.(مثلا در شکل زیر اگر andمورد نظر که نشان دهنده ی امدن دستور isubاست یک شود مقدار کانستن "43"به باس خروجی متصل میشود)



در نتیجه خروجی واحد mapperادرس شروع اولین اینستراکشن از دستور مورد نظر در رام خواهد بود.

در مورد واحد gen signalباید کفت که این واخد وظیفه ی تولید سیگنال های ترجمه شده برای دیتاپس را خواهد داشت .

در واقع خروجی هر ورد "کنترل رام" یا به عبارتی (f1,f2,f3)به این واحد داده میشود تا سیگنال های لود و...رجیستر ها یا مالتی پلکسر ها را در دیتاپس تولید کند.

در این واحد 3 دیکودر قرار داده شده است که نام میکرو اپریشن مربوط به خودشان را تولید میکنند .

و هر سیگنال تولید شده بسته به این که با میکرو اپریشن ها اشتراک داشته باشد یا نه حاصل orبعضی از ان ها خواهد بود که به طور کامل همگی در نظر گرفته شده اند.

واحد دیگری به نام input logic نیز وجود دارد که در آن انتخاب وردی ادرس بعدی رام انجام میشود.

کنترل زمان هایی که نیاز به برنچ میباشد و انتخاب مالتی پلکسری که ادرس بعدی را مشخص میکند در این واحد با گیت های منطقی انجام شده است.

فایل های مربوط به تمامی قسمت ها در فایل فرستاده شده پیوست شده است.

در این تمرین نیز مانند تمرین قبلی برای کنترل مموری از یک فلیپ فلاپ استفاده شده است که دقیقا مانند تمرین قبل پیاده سازی شده است و در مواقعی باید منتظر خروجی مموری بماند این کار انجام میشود.