



ארגון המחשב לאוניברסיטה הפתוחה



לקורס המלא ולעוד עשרות סיכומים בחינם:

Click קליק 
by Yehoran

<https://click-go-easy.click/>

לשאלות, הערות,
שיעורים פרטיים ושיעורים קבוצתיים:
יהורן - [0506798719](https://www.callme4u.com/0506798719)



- מבוא
- שפת האסמבלי של מעבד ה-MIPS
- מעבד חד מחזורי
- מעבד צנרת
- זיכרון

(היכנסו לקישורים לסרטוני חזרה על החומר בחינם)

מבוא לארגון המחשב - סיכום



סימון	פירוש	יחידות	מושג
CCT	Clock Cycle Time	[sec]	זמן מחזור שעון
CR	Clock Rate	[1/sec]=[Hz]	תדירות השעון
CC	Clock Cycle	[Cc/program]	מספר מחזורי שעון בתוכנית
IC	Instruction Count	[ins/program]	מספר פקודות מכונה בתוכנית
CPI	Clock Per Instruction	[Cc/ins]	מספר ממוצע של מחזורי שעון לפקודה אחת
CPU Time	Central Processing Unit Time	[sec/program]	זמן הריצה של תוכנית
Speedup			גורם ההאצה (חלוקת הביצועים האיטיים במהירים)
MIPS	Million Instructions Per Second	[MIPS]	מיליון פקודות בשנייה

מספרים גדולים או קטנים מאוד						
m	10^{-3}	מילי		K	10^3	קילו
μ	10^{-6}	מיקרו		M	10^6	מגה
n	10^{-9}	ננו		G	10^9	גיגה
p	10^{-12}	פיקו		T	10^{12}	טרה

נוסחאות	יחידות	מושג
$CPI = \sum_i CPI_i \cdot \omega_i$	[Cc/ins]	מספר ממוצע של מחזורי שעון לפקודה אחת
$CPU\ Time = CC \cdot CCT = \frac{CC}{CR} = IC \cdot CPI \cdot CCT = \frac{IC \cdot CPI}{CR}$	[sec/program]	זמן הריצה של תוכנית
$MIPS = \frac{IC}{CPU\ Time \cdot 10^6} = \frac{CR}{CPI \cdot 10^6}$	[MIPS]	מיליון פקודות בשנייה
$Speedup = \frac{CPU\ Time\ SLOW}{CPU\ Time\ FAST}$		מדד האצה (להשוואה בין מעבד איטי למעבד מהיר)
$Speedup = \frac{1}{(1 - F_{\text{החלק המשופר}}) + \frac{F_{\text{החלק המשופר}}}{Speedup_{\text{אחוז השיפור}}}}$		חוק אמדל (לחישוב השיפור הכללי של תוכנית שרק חלק ממנה עבר שיפור)

מבוא לארגון המחשב - סיכום



המרה מבסיס עשרוני לכל בסיס אחר

מחלקים את המספר העשרוני בבסיס r . שארית החלוקה ב- r יוצרת את המספר בבסיס r אליו אנו מנסים להמיר. המספר נבנה מימין לשמאל. בכל פעם רושמים את שארית החלוקה ומחלקים שוב את התוצאה בבסיס r . ממשיכים לחלק שוב ושוב עד שתוצאת החילוק שווה 0 (שהמחולק קטן מהבסיס):

לדוגמה, בכדי להמיר את המספר 158 לבסיס בינארי:

$$\begin{array}{r|l} \frac{1}{2} = 0 & \frac{2}{2} = 1 \\ \frac{4}{2} = 2 & \frac{9}{2} = 4 \\ \frac{19}{2} = 9 & \frac{39}{2} = 19 \\ \frac{79}{2} = 39 & \frac{158}{2} = 79 \\ \hline 1 & 0 \end{array}$$

המרה מבסיס כלשהו לבסיס עשרוני

כופלים כל מספר בבסיס r בחזקת המיקום שלו (המספר הימני ביותר במיקום 0, והמספר השמאלי ביותר במיקום n):

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + a_{n-2} \cdot r^{n-2} + \dots + a_0 \cdot r^0$$

לדוגמה, בכדי להמיר את המספר 158 בבסיס בינארי לבסיס עשרוני, נשתמש בטור חזקות:

$$1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 1 \cdot 128 + 1 \cdot 16 + 1 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 = 158$$

מבוא לארגון המחשב - סיכום



המרה מהירה בין בסיסים 2, 4, 8, 16

כיצד לזכור?	שווה ל- ___ ספרות בבסיס בינארי	ספרה בבסיס
$2^2 = 4$	2	4
$2^3 = 8$	3	8
$2^4 = 16$	4	16

לדוגמה:

2	1	3	2	4	158 בבסיס 4			
1	0	0	1	1	1	1	0	158 בבסיס בינארי

2	3	6	158 בבסיס אוקטלי						
0	1	0	0	1	1	1	1	0	158 בבסיס בינארי

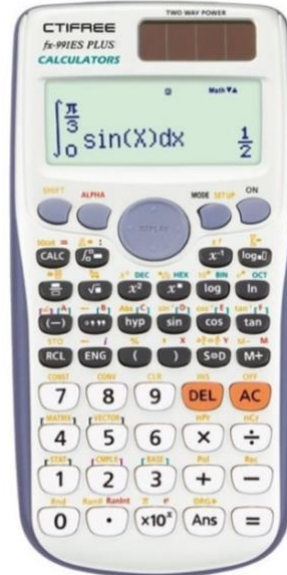
9	E	158 בבסיס הקסאדצימלי						
1	0	0	1	1	1	1	0	158 בבסיס בינארי

מספרים חשובים בבסיסים נפוצים																		
10000	1111	1110	1101	1100	1011	1010	1001	1000	111	110	101	100	11	10	1	0	(2)	בינארי
20	17	16	15	14	13	12	11	10	7	6	5	4	3	2	1	0	(8)	אוקטלי
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	(10)	דצימלי
10	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	(16)	הקסאדצימלי

מבוא לארגון המחשב - סיכום



שימוש במחשבון להמרה בין בסיסים 2, 8, 16 (מספרים שליליים בבסיס בינארי בשיטת משלים ל-2)



$MODE \rightarrow 4: BASE - N \rightarrow$ בחירת בסיס ($DEC / HEX / BIN / OCT$) \rightarrow כתיבת מספר $\rightarrow = \rightarrow$ בחירת בסיס אחר
(לחזרה למצב חישוב רגיל: $MODE \rightarrow 1: COMP$)

מחשבון אינטרנטי להמרה בין כל הבסיסים (לא עובד במספרים שליליים)

http://www.unitconversion.org/unit_converter/numbers.html

מבוא לארגון המחשב - סיכום



הצגת מספרים עם סימן (חיובי ושילולי) בשיטת משלים ל-2

9	7	10	6	11	5	12	4	13	3	14	2	15	1
1001	0111	1010	0110	1011	0101	1100	0100	1101	0011	1110	0010	1111	0001
-7		-6		-5		-4		-3		-2		-1	
1 1 1	1 1	1 1 1	1 1	1 1 1	1	1	1 1 1	1 1 1	1 1	1 1	1 1	1 1 1	1 1 1
+ 0 1 1 1	+ 0 1 1 0	+ 1 0 0 1	+ 1 0 1 0	+ 0 1 0 1	+ 0 1 0 0	+ 1 1 0 0	+ 0 0 1 1	+ 1 1 0 1	+ 0 0 1 0	+ 1 1 1 0	+ 1 1 1 0	+ 0 0 0 1	+ 1 1 1 1
1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0

הפיכת מספר חיובי למספר שלילי בשיטת משלים ל-2

$$\begin{array}{r} -1 \quad -1 \quad -1 \\ - \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \\ \quad \quad \quad 1 \quad 0 \quad 1 \\ \hline 1 \quad 0 \quad 1 \quad 1 \end{array}$$

1. על פי ההגדרה המתמטית: $-N = (2^n - N)$, לדוגמה, המספר -5 בהצגה של 4 ביטים הוא:
2. ביצוע הפעולה "NOT" על המספר החיובי והוספת המספר 1:

$$NOT(0101) = 1010 + 1 = 1011$$

בהצגה של 4 ביטים הוא:

3. מעבר על כל הסיביות של המספר החיובי מימין לשמאל עד לסיבית הראשונה שערכה 1, העתקה של סיבית זו כפי שהיא והפיכת כל הסיביות שמשמאלה באמצעות הפעולה "NOT"

תחום המספרים להצגה ב-n ביטים

מספרים עם סימן בשיטת משלים ל-2		מספרים ללא סימן	
2^{n-1}	$2^{n-1} - 1$	0	$2^n - 1$
שליליים	חיוביים	חיוביים	

זיהוי גלישה בחיבור שני מספרים

מספרים עם סימן בשיטת משלים ל-2	מספרים ללא סימן
הנשא (carry) האחרון ואחד לפניו שונים זה מזה	הנשא (carry) האחרון שווה 1

מבוא לארגון המחשב - סיכום



Buffer	OR (+)	AND (·)	NOT (A' או \bar{A})																																										
<table border="1"> <thead> <tr> <th>A</th> <th>A</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	A	0	0	1	1	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>A + B</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table>	A	B	A + B	0	1	1	1	0	1	1	1	1	0	0	0	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>A · B</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table>	A	B	A · B	0	1	0	1	0	0	1	1	1	0	0	0	<table border="1"> <thead> <tr> <th>A</th> <th>\bar{A}</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	\bar{A}	0	1	1	0
A	A																																												
0	0																																												
1	1																																												
A	B	A + B																																											
0	1	1																																											
1	0	1																																											
1	1	1																																											
0	0	0																																											
A	B	A · B																																											
0	1	0																																											
1	0	0																																											
1	1	1																																											
0	0	0																																											
A	\bar{A}																																												
0	1																																												
1	0																																												

XNOR (\odot) = $(AB) + (\bar{A}\bar{B})$	XOR (\oplus) = $(\bar{A}B) + (A\bar{B})$	NOR (\downarrow) = $\overline{(A + B)}$	NAND (\uparrow) = $\overline{(A \cdot B)}$																																																												
<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>A \odot B</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> </tbody> </table>	A	B	A \odot B	0	1	0	1	0	0	1	1	1	0	0	1	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>A \oplus B</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table>	A	B	A \oplus B	0	1	1	1	0	1	1	1	0	0	0	0	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>A \downarrow B</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> </tbody> </table>	A	B	A \downarrow B	0	1	0	1	0	0	1	1	0	0	0	1	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>A \uparrow B</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> </tbody> </table>	A	B	A \uparrow B	0	1	1	1	0	1	1	1	0	0	0	1
A	B	A \odot B																																																													
0	1	0																																																													
1	0	0																																																													
1	1	1																																																													
0	0	1																																																													
A	B	A \oplus B																																																													
0	1	1																																																													
1	0	1																																																													
1	1	0																																																													
0	0	0																																																													
A	B	A \downarrow B																																																													
0	1	0																																																													
1	0	0																																																													
1	1	0																																																													
0	0	1																																																													
A	B	A \uparrow B																																																													
0	1	1																																																													
1	0	1																																																													
1	1	0																																																													
0	0	1																																																													

4	← 3	← 2	← 1
OR (+)	AND (·)	NOT (\bar{A})	()

מבוא לארגון המחשב - סיכום



AND (·)		OR (+)	כלל
$A \cdot 1 = A$		$A + 0 = A$	איבר יחידה
$A \cdot 0 = 0$		$A + 1 = 1$	איון
$A \cdot A = A$		$A + A = A$	אידמפוטנט
$A \cdot \bar{A} = 0$		$A + \bar{A} = 1$	משלים
$A \cdot B = B \cdot A$		$A + B = B + A$	חילוף
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$		$A + (B + C) = (A + B) + C$	קיבוץ
$A + (B \cdot C) = (A + B) \cdot (A + C)$		$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	פילוג
$A \cdot (A + B) = A$		$A + (A \cdot B) = A$	ספיגה
$\overline{(A \cdot B)} = \bar{A} + \bar{B}$		$\overline{(A + B)} = \bar{A} \cdot \bar{B}$	משפט דה מורגן
$A \cdot (\bar{A} + B) = A \cdot B$		$A + (\bar{A} \cdot B) = A + B$	ההופכי הנעלם
$\overline{(\bar{A})} = A$			הופכי כפול

מחשבון לאלגברה בוליאנית: <https://www.boolean-algebra.com>

שפת האסמבלי של מעבד ה-MIPS - סיכום



אוגרים נוספים לשימוש המעבד בלבד		
כיצד לזכור?	שם אוגר	שימוש
program counter	pc	הכתובת של הפקודה הנוכחית בתוכנית
high	hi	החצי הגדול של תוצאת מכפלה / שארית החלוקה של תוצאת חילוק
low	lo	החצי הקטן של תוצאת מכפלה / החלק השלם של תוצאת חילוק

מקבץ האוגרים - registers			
שימוש	מספר אוגר	שם אוגר	כיצד לזכור?
ערך 0 קבוע (ברמת הברזל - לא ניתן לשינוי)	\$0	\$zero	
שמור לאסמבלר לפסידו פקודות	\$1	\$at	at - assembler temp
תוצאות חישובים	\$2	\$v0	v - value
	\$3	\$v1	
ארגומנטים	\$4	\$a0	a - arguments
	\$5	\$a1	
	\$6	\$a2	
	\$7	\$a3	
משתנים זמניים	\$8	\$t0	t - temporaries
	\$9	\$t1	
	\$10	\$t2	
	\$11	\$t3	
	\$12	\$t4	
	\$13	\$t5	
	\$14	\$t6	
	\$15	\$t7	
שמירת נתונים לשימוש בהמשך	\$16	\$s0	s - save for later
	\$17	\$s1	
	\$18	\$s2	
	\$19	\$s3	
	\$20	\$s4	
	\$21	\$s5	
	\$22	\$s6	
	\$23	\$s7	
עוד משתנים זמניים	\$24	\$t8	t - temporaries
	\$25	\$t9	
רגיסטרים שמורים למערכת ההפעלה	\$26	\$k0	k - kernel
	\$27	\$k1	
Global pointer	\$28	\$gp	
Stack pointer	\$29	\$sp	
Frame pointer	\$30	\$fp	
Return address	\$31	\$ra	

שפת האסמבלי של מעבד ה-MIPS - סיכום



פקודות בשפת מכונה						
פקודות מסוג	6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
R (register)	opcode	rs	rt	rd	shamt	funct
I (immediate)		immediate (16 bit)				
J (jump)		address (26 bit)				

(opcode = 000000) R (Registers) פקודות מסוג				funct		
פעולה	שם הפקודה	תחביר	משמעות (קידוד)	(dec)	(hex)	(bin)
אינטראקציה עם התקני הקלט והפלט של המחשב	syscall	syscall	system call (000000,000000,000000,000000,000000,001100)	12	0x0C	001100

שפת האסמבלי של מעבד ה-MIPS - סיכום



(opcode = 000000) R (Registers) מסוג פקודות				funct		
פעולה	שם הפקודה	תחביר	משמעות (קידוד)	(dec)	(hex)	(bin)
חיבור	add	add \$rd, \$rs, \$rt	rd=rs+rt (000000,sssss,ttttt,dddd,00000,100000)	32	0X20	100000
חיבור ללא התראה על הצפה	add unsigned	addu \$rd, \$rs, \$rt	rd=rs+rt (000000,sssss,ttttt,dddd,00000,100001)	33	0X21	100001
חיסור	sub	sub \$rd, \$rs, \$rt	rd=rs-rt (000000,sssss,ttttt,dddd,00000,100010)	34	0X22	100010
חיסור ללא התראה על הצפה	sub unsigned	subu \$rd, \$rs, \$rt	rd=rs-rt (000000,sssss,ttttt,dddd,00000,100011)	35	0X23	100011
כפל	multiply	mult \$rs, \$rt	{hi,lo}=rs*rt (000000,sssss,ttttt,000000,00000,011000)	24	0X18	011000
כפל ללא התראה על הצפה	multiply unsigned	multu \$rs, \$rt	{hi,lo}=rs*rt (000000,sssss,ttttt,000000,00000,011001)	25	0X19	011001
חילוק	divide	div \$rs, \$rt	lo=rs/rt, hi=rs%rt (000000,sssss,ttttt,000000,00000,011010)	26	0X1A	011010
חילוק ללא התראה על הצפה	divide unsigned	divu \$rs, \$rt	lo=rs/rt, hi=rs%rt (000000,sssss,ttttt,000000,00000,011011)	27	0X1B	011011
העתקת הערך באוגר hi לאוגר rd	move from hi	mfhi \$rd	rd=hi (000000,00000,00000,dddd,00000,010000)	16	0X10	010000
העתקת הערך באוגר lo לאוגר rd	move from lo	mflo \$rd	rd=lo (000000,00000,00000,dddd,00000,010010)	18	0X12	010010
פעולת "וגם" לוגית	and	and \$rd, \$rs, \$rt	rd=rs&rt (000000,sssss,ttttt,dddd,00000,100100)	36	0X24	100100
פעולת "או" לוגית	or	or \$rd, \$rs, \$rt	rd=rs rt (000000,sssss,ttttt,dddd,00000,100101)	37	0X25	100101
פעולת "שוני" לוגית	xor	xor \$rd, \$rs, \$rt	rd=rs ⊕ rt (000000,sssss,ttttt,dddd,00000,100110)	38	0X26	100110
פעולת "לא או" לוגית	nor	nor \$rd, \$rs, \$rt	rd=rs ↓ rt (000000,sssss,ttttt,dddd,00000,100111)	39	0X27	100111
הדלקת rd אם rs קטן ממש מ-rt	set on less than	slt \$rd, \$rs, \$rt	if (\$rs<\$rt) \$rd=1 else \$rd=0 (000000,sssss,ttttt,dddd,00000,101010)	42	0X2A	101010
כנ"ל תוך התעלמות מסימן	set on less than unsigned	sltu \$rd, \$rs, \$rt	if (\$rs<\$rt) \$rd=1 else \$rd=0 (000000,sssss,ttttt,dddd,00000,101011)	43	0X2B	101011
הזזה שמאלה וריפוד באפסים	shift left logical	sll \$rd, \$rt, shamt	rd=rt<<(shamt) (000000,00000,ttttt,dddd,shshs,000000)	0	0X00	000000
הזזה ימינה וריפוד באפסים	shift right logical	srl \$rd, \$rt, shamt	rd=rt>>(shamt) (000000,00000,ttttt,dddd,shshs,000010)	2	0X02	000010
הזזה ימינה וריפוד בסיבית הסימן	shift right arithmetic	sra \$rd, \$rt, shamt	rd=rt>>(shamt) (000000,00000,ttttt,dddd,shshs,000011)	3	0X03	000011
קפיצה ללא תנאי לכתובת שברגיסטר	jump register	jr \$rs	pc=\$rs (000000,sssss,00000,00000,00000,001000)	8	0X08	001000
שמירת הכתובת ב-\$ra וקפיצה	jump and link register	jalr \$rs	\$ra=pc+4, pc=\$rs (000000,sssss,00000,11111,00000,001001)	9	0X09	001001
עצירת התוכנית	break	break	stop program (000000,00000,00000,00000,00000,001101)	13	0X0D	001101

שפת האסמבלי של מעבד ה-MIPS - סיכום



פקודות מסוג I (immediate) ומסוג J (jump)				opcode			
פעולה	שם הפקודה	תחביר	משמעות (קידוד)	ריפוד	(dec)	(hex)	(bin)
חיבור	add immediate	addi \$rt, \$rs, imm	rt=rs+immediate (001000,sssss,ttttt,iiiiiiiiiiiiiii)	SE	8	0x08	001000
חיבור ללא התראה על הצפה	add immediate unsigned	addiu \$rt, \$rs, imm	rt=rs+immediate (001001,sssss,ttttt,iiiiiiiiiiiiiii)	SE	9	0x09	001001
פעולת "וגם" לוגית	and immediate	andi \$rt, \$rs, imm	rt=rs&rt (001100,sssss,ttttt,iiiiiiiiiiiiiii)	ZE	12	0x0C	001100
פעולת "או" לוגית	or immediate	ori \$rt, \$rs, imm	rt=rs rt (001101,sssss,ttttt,iiiiiiiiiiiiiii)	ZE	13	0x0D	001101
פעולת "שוני" לוגית	xor immediate	xori \$rt, \$rs, imm	rt=rs ⊕ rt (001110,sssss,ttttt,iiiiiiiiiiiiiii)	ZE	14	0x0E	001110
הדלקת rt אם rs קטן ממש מ-imm	set on less than imm	slti \$rt, \$rs, imm	if (\$rs<imm) \$rt=1 else \$rt=0 (001010,sssss,ttttt,iiiiiiiiiiiiiii)	SE	10	0x0A	001010
כנ"ל תוך התעלמות מסימן	set on less than imm unsigned	sltiu \$rt, \$rs, imm	if (\$rs<imm) \$rt=1 else \$rt=0 (001011,sssss,ttttt,iiiiiiiiiiiiiii)	SE	11	0x0B	001011
קפיצה אם \$rt=\$rs	branch if equal	beq \$rs, \$rt, label	if (\$rs=\$rt) pc=pc+4+4*imm (000100,sssss,ttttt,iiiiiiiiiiiiiii)	SE	4	0x04	000100
קפיצה אם \$rt!=\$rs	branch if not equal	bne \$rs, \$rt, label	if (\$rs!=\$rt) pc=pc+4+4*imm (000101,sssss,ttttt,iiiiiiiiiiiiiii)	SE	5	0x05	000101
קפיצה אם \$rs<=0	branch if less than or equal zero	blez \$rs, label	if (\$rs<=0) pc=pc+4+4*imm (000110,sssss,00000,iiiiiiiiiiiiiii)	SE	6	0x06	000110
קפיצה אם \$rs>0	branch if greater than zero	bgtz \$rs, label	if (\$rs>0) pc=pc+4+4*imm (000111,sssss,00000,iiiiiiiiiiiiiii)	SE	7	0x07	000111
טעינת בית (8) מכתובת \$rs+imm בזיכרון, SE	load byte	lb \$rt, imm(\$rs)	\$rt=memory(\$rs+imm) (100000,sssss,ttttt,iiiiiiiiiiiiiii)	SE	32	0x20	100000
טעינת בית (8) מכתובת \$rs+imm בזיכרון, ZE	load byte unsigned	lbu \$rt, imm(\$rs)	\$rt=memory(\$rs+imm) (100100,sssss,ttttt,iiiiiiiiiiiiiii)	SE	36	0x24	100100
טעינת חצי מילה (16) המתחילה בכתובת \$rs+imm, SE	load half-word	lh \$rt, imm(\$rs)	\$rt=memory(\$rs+imm) (100001,sssss,ttttt,iiiiiiiiiiiiiii)	SE	33	0x21	100001
טעינת חצי מילה (16) המתחילה בכתובת \$rs+imm, ZE	load half-word unsigned	lhu \$rt, imm(\$rs)	\$rt=memory(\$rs+imm) (100101,sssss,ttttt,iiiiiiiiiiiiiii)	SE	37	0x25	100101
טעינת מילה (32) המתחילה בכתובת \$rs+imm בזיכרון	load word	lw \$rt, imm(\$rs)	\$rt=memory(\$rs+imm) (100011,sssss,ttttt,iiiiiiiiiiiiiii)	SE	35	0x23	100011
שמירת 8 הביטים הנמוכים של \$rt בכתובת \$rs+imm בזיכרון	store byte	sb \$rt, imm(\$rs)	memory(\$rs+imm)=\$rt (101000,sssss,ttttt,iiiiiiiiiiiiiii)	SE	40	0x28	101000
שמירת 16 הביטים הנמוכים של \$rt בכתובת \$rs+imm בזיכרון ובכתובת שאחריה	store half-word	sh \$rt, imm(\$rs)	memory(\$rs+imm)=\$rt (101001,sssss,ttttt,iiiiiiiiiiiiiii)	SE	41	0x29	101001
שמירת 32 הביטים של \$rt בכתובת \$rs+imm בזיכרון וב-3 התאים שאחריה	store word	sw \$rt, imm(\$rs)	memory(\$rs+imm)=\$rt (101011,sssss,ttttt,iiiiiiiiiiiiiii)	SE	43	0x2B	101011
הכנסת הערך המייד ל-16 הביטים הגבוהים של \$rt	load upper immediate	lui \$rt, imm	\$rt=imm,0000000000000000 (001111,00000,ttttt,iiiiiiiiiiiiiii)		15	0x0F	001111
קפיצה ללא תנאי לכתובת של התווית	jump	j label	pc=pc+4[31...28] + imm*4 (000010,iiiiiiiiiiiiiiiiiiii)		2	0x02	000010
שמירת הכתובת ב-\$ra וקפיצה	jump and link	jal label	\$ra=pc+4, pc=pc+4[31...28]+imm*4 (000011,iiiiiiiiiiiiiiiiiiii)		3	0x03	000011

שפת האסמבלי של מעבד ה-MIPS - סיכום



פסידו פקודות				
פעולה	שם הפקודה	תחביר	פקודות שיתבצעו בפועל	משמעות
העתקת אוגר \$rs לאוגר \$rt	move	move \$rt, \$rs	add \$rt, \$rs, \$zero	\$rt=\$rs
איפוס המשתנה \$rt	clear	clear \$rt	add \$rt, \$zero, \$zero	\$rt=0
היפוך כל הסיביות באוגר \$rs והכנסת התוצאה לאוגר \$rt	not	not \$rt, \$rs	nor \$rt, \$rs, \$zero	\$rt = $\overline{\$rs}$
טעינת כתובת בת 32 סיביות לאוגר \$rd	load address	la \$rt, label	lui \$rt, label [31...16] ori \$rd, \$rd, label [15...0]	\$rt=label address
טעינת קבוע בן 32 סיביות לאוגר \$rd	load immediate	li \$rt, imm [31...0]	lui \$rt, imm [31...16] ori \$rt, \$rt, imm [15...0]	\$rt=32 bit imm
קפיצה ללא תנאי	branch unconditionally	b label	beq \$zero, \$zero, label	pc=label
קפיצה ללא תנאי ושמירת המיקום הנוכחי ב-\$ra	branch and link	bal label	bgezal \$zero, label	\$ra=pc+4, pc=label
קפיצה אם האוגר שווה למספר המייד	branch equal immediate	beq \$rt, imm, label	addi \$1, \$0, imm beq \$1, \$rt, label	if(\$rt=imm) pc=label
קפיצה אם האוגר שווה 0	branch equal zero	beqz \$rt, label	beq \$rt, \$0, label	if(\$rt=0) pc=label
קפיצה אם האוגר שונה מ-0	branch not equal zero	bnez \$rt, label	bne \$rt, \$0, label	if(\$rt!=0) pc=label
קפיצה אם $\$rs > \rt	branch if greater than	bgt \$rs, \$rt, label	slt \$1, \$rt, \$rs bne \$1, \$0, label	if(\$rs>\$rt) pc=label
קפיצה אם $\$rs \geq \rt	branch if greater or equal than	bge \$rs, \$rt, label	slt \$1, \$rs, \$rt beq \$1, \$0, label	if(\$rs>=\$rt) pc=label
קפיצה אם $\$rs < \rt	branch if less than	blt \$rs, \$rt, label	slt \$1, \$rs, \$rt bne \$1, \$0, label	if(\$rs<\$rt) pc=label
קפיצה אם $\$rs \leq \rt	branch if less or equal than	ble \$rs, \$rt, label	slt \$1, \$rt, \$rs beq \$1, \$0, label	if(\$rs<=\$rt) pc=label

שפת האסמבלי של מעבד ה-MIPS - סיכום



חישוב הכתובת לקפיצה בפקודות Branch ו-Jump

איך מחשבים את המספר ה-immediate שכתוב בקידוד של הפקודה בפקודות branch ו-jump?

בפקודות branch הקפיצה היא יחסית, ואילו בפקודות jump הקפיצה אבסולוטית. מה זה אומר בתכלס?

בפקודות branch, התווית הרשומה בפקודת האסמבלי מתורגמת למספר מידי ב-16 ביטים (עם סימן) המסמן את כמות השורות שיש לעלות או לרדת מ- $pc+4$ כדי להגיע לתווית הרשומה בפקודה.

לאחר מכן, המעבד הופך את המספר הזה לכתובת שאליה יש לקפוץ ושם אותה באוגר pc באמצעות החישוב הבא: $pc = pc + 4 + 4 * imm$

בפקודות branch המספר שיהיה בקידוד של הפקודה הוא כמות השורות שיש לקפוץ ביחס ל- $pc+4$ בכדי להגיע לפקודה שהתווית של הפקודה מצביעה עליה. זאת אומרת, שאם נניח את האצבע על הפקודה הבאה בקוד (זו שבאה מיד אחרי פקודת ה-brunch) ונספור כמה שורות יש לעלות או לרדת ביחס אליה עד שנגיע לפקודה שעליה התווית מצביעה, זה המספר שיהיה כתוב בקידוד של הפקודה. (אם צריך לרדת 5 שורות למטה, המספר יהיה 5, ואם צריך לעלות 3 פקודות למעלה, המספר יהיה מינוס 3).

בפקודת jump הערך של הכתובת אליה יש לקפוץ נתון ב-26 סיביות. לעומת זאת, כתובת של פקודה בזיכרון נתון ב-32 סיביות. כיצד מתגברים על המכשול?

התשובה: לוקחים את הכתובת שאליה יש לקפוץ, מוחקים ממנה את 2 הסיביות הימניות ואת 4 הסיביות השמאליות.

לאחר מכן, המעבד יוסיף 2 אפסים מימין מכיוון שהכתובות של הפקודות מתחלקות ב-4 שהרי כל פקודה היא מילה ולכן תופסת 4 תאים בזיכרון.

בנוסף, המעבד ייקח את 4 הסיביות השמאליות ביותר של $pc+4$ ויצרף אותם לצד שמאל של המספר בן ה-26 סיביות הנתון בפקודה.

01010010001111101010111000100100

בפקודות jump יש לחשב את הכתובת הממשית של הפקודה בזיכרון. אבל כתובת בזיכרון נתונה ב-32 ביטים, ואילו לנו בקידוד של הפקודה יש מקום רק ל-26 ביטים (שהרי ה-opcode תופס לנו 6 ביטים). אז איך עושים את זה?

מאוד פשוט: מציגים את הכתובת של הפקודה שאליה רוצים לקפוץ בהצגה בינארית ב-32 ביטים, מוחקים את שני הביטים הימניים ואת ארבעת הביטים השמאליים - המספר שיתקבל זהו המספר שיהיה בקידוד של הפקודה. (לאחר מכן, המעבד יוסיף 2 אפסים מימין (כי כתובות של פקודות מתחלקות ב-4), ויעתיק את 4 הביטים השמאליים של הפקודה שאחרי הפקודה שבה אנחנו נמצאים ל-4 הביטים השמאליים של הכתובת שאליה קופצים).

שפת האסמבלי של מעבד ה-MIPS - סיכום



כל התווים הניתנים להדפסה

BINARY	DECIMAL	HEXADECIMAL	SYMBOL	BINARY	DECIMAL	HEXADECIMAL	SYMBOL
010 0000	32	20	SPACE	101 0000	80	50	P
010 0001	33	21	!	101 0001	81	51	Q
010 0010	34	22	"	101 0010	82	52	R
010 0011	35	23	#	101 0011	83	53	S
010 0100	36	24	\$	101 0100	84	54	T
010 0101	37	25	%	101 0101	85	55	U
010 0110	38	26	&	101 0110	86	56	V
010 0111	39	27	'	101 0111	87	57	W
010 1000	40	28	(101 1000	88	58	X
010 1001	41	29)	101 1001	89	59	Y
010 1010	42	2A	*	101 1010	90	5A	Z
010 1011	43	2B	+	101 1011	91	5B	[
010 1100	44	2C	,	101 1100	92	5C	\
010 1101	45	2D	-	101 1101	93	5D]
010 1110	46	2E	.	101 1110	94	5E	^
010 1111	47	2F	/	101 1111	95	5F	_
011 0000	48	30	0	110 0000	96	60	`
011 0001	49	31	1	110 0001	97	61	a
011 0010	50	32	2	110 0010	98	62	b
011 0011	51	33	3	110 0011	99	63	c
011 0100	52	34	4	110 0100	100	64	d
011 0101	53	35	5	110 0101	101	65	e
011 0110	54	36	6	110 0110	102	66	f
011 0111	55	37	7	110 0111	103	67	g
011 1000	56	38	8	110 1000	104	68	h
011 1001	57	39	9	110 1001	105	69	i
011 1010	58	3A	:	110 1010	106	6A	j
011 1011	59	3B	;	110 1011	107	6B	k
011 1100	60	3C	<	110 1100	108	6C	l
011 1101	61	3D	=	110 1101	109	6D	m
011 1110	62	3E	>	110 1110	110	6E	n
011 1111	63	3F	?	110 1111	111	6F	o
100 0000	64	40	@	111 0000	112	70	p
100 0001	65	41	A	111 0001	113	71	q
100 0010	66	42	B	111 0010	114	72	r
100 0011	67	43	C	111 0011	115	73	s
100 0100	68	44	D	111 0100	116	74	t
100 0101	69	45	E	111 0101	117	75	u
100 0110	70	46	F	111 0110	118	76	v
100 0111	71	47	G	111 0111	119	77	w
100 1000	72	48	H	111 1000	120	78	x
100 1001	73	49	I	111 1001	121	79	y
100 1010	74	4A	J	111 1010	122	7A	z
100 1011	75	4B	K	111 1011	123	7B	{
100 1100	76	4C	L	111 1100	124	7C	
100 1101	77	4D	M	111 1101	125	7D	}
100 1110	78	4E	N	111 1110	126	7E	~
100 1111	79	4F	O	111 1111	127	7F	DEL

מספרים

bin	hex	dec	סימן
0011 0000	0X30	48	0
0011 0001	0X31	49	1
0011 0010	0X32	50	2
0011 0011	0X33	51	3
0011 0100	0X34	52	4
0011 0101	0X35	53	5
0011 0110	0X36	54	6
0011 0111	0X37	55	7
0011 1000	0X38	56	8
0011 1001	0X39	57	9

אותיות גדולות וקטנות באנגלית

bin	hex	dec	סימן	bin	hex	dec	סימן
0100 0001	41	65	A	0110 0001	61	97	a
0100 0010	42	66	B	0110 0010	62	98	b
0100 0011	43	67	C	0110 0011	63	99	c
0100 0100	44	68	D	0110 0100	64	100	d
0100 0101	45	69	E	0110 0101	65	101	e
0100 0110	46	70	F	0110 0110	66	102	f
0100 0111	47	71	G	0110 0111	67	103	g
0100 1000	48	72	H	0110 1000	68	104	h
0100 1001	49	73	I	0110 1001	69	105	i
0100 1010	4a	74	J	0110 1010	6a	106	j
0100 1011	4b	75	K	0110 1011	6b	107	k
0100 1100	4c	76	L	0110 1100	6c	108	l
0100 1101	4d	77	M	0110 1101	6d	109	m
0100 1110	4e	78	N	0110 1110	6e	110	n
0100 1111	4f	79	O	0110 1111	6f	111	o
0101 0000	50	80	P	0111 0000	70	112	p
0101 0001	51	81	Q	0111 0001	71	113	q
0101 0010	52	82	R	0111 0010	72	114	r
0101 0011	53	83	S	0111 0011	73	115	s
0101 0100	54	84	T	0111 0100	74	116	t
0101 0101	55	85	U	0111 0101	75	117	u
0101 0110	56	86	V	0111 0110	76	118	v
0101 0111	57	87	W	0111 0111	77	119	w
0101 1000	58	88	X	0111 1000	78	120	x
0101 1001	59	89	Y	0111 1001	79	121	y
0101 1010	5a	90	Z	0111 1010	7a	122	z

שפת האסמבלי של מעבד ה-MIPS - סיכום



פקודות syscall

להפעלת הפקודה יש:

1. להציב באוגר \$v0 מספר בהתאם לטבלה להלן: li \$v0, code
 2. להציב באוגרים \$a0, \$a1 את הנתונים לפונקציה: add \$a0, \$t1, \$zero (אם יש)
 3. לקרוא לפקודה syscall
- לקלוט את הנתונים מהפקודה (אם יש)

שם הפקודה	פעולה	קוד ב-\$v0	ארגומנטים	תוצאות
print integer	הדפסת מספר שלם	1	\$a0 - מספר שלם להדפסה	
print string	הדפסת מחרוזת	4	\$a0 - הכתובת למחרוזת asciiz	
read integer	קריאת מספר שלם	5		\$v0 - מכיל את המספר שהוקלד
read string	קריאת מחרוזת	8	\$a0 - הכתובת בזיכרון שבה יש לאחסן את הקלט, \$a1 - כמות התווים המקסימלית לקליטה	
exit	סיום התוכנית	10		
print character	הדפסת תו	11	\$a0 - הערך של האוגר יודפס	
read character	קריאת תו	12		\$v0 - מכיל את התו שהוקלד
print integer in hex	הדפסת מספר בבסיס 16	34	\$a0 - מספר שלם להדפסה	מציג את המספר ב-8 ספרות
print integer in bin	הדפסת מספר בבסיס 2	35	\$a0 - מספר שלם להדפסה	מציג את המספר ב-32 ספרות
print integer as unsigned	הדפסת מספר ללא סימן	36	\$a0 - מספר שלם להדפסה	מציג את המספר בהצגה עשרונית

שפת האסמבלי של מעבד ה-MIPS - סיכום



מאקרו	פרוצדורה
- קטע קוד המתחיל בהנחיה ".macro" ומיד אחריה שם המאקרו.	- קטע קוד המתחיל בתווית.
- המאקרו מסתיים בהנחיה ".end_macro".	- הפרוצדורה יכולה להיות רשומה אחרי סיום התוכנית.
- קריאה למאקרו באמצעות השם שלו.	- קפיצה לפרוצדורה באמצעות jal.
- האסמבלר מחליף את הקריאה בפקודות שרשומות בגוף המאקרו.	- חזרה מהפרוצדורה באמצעות jr \$ra.
- אפשר לקרוא למאקרו רק אם הוא הוגדר לפני הקריאה אליו.	- העברת ארגומנטים לפרוצדורה באמצעות \$a0-\$a3.
- העברת ארגומנטים למאקרו באמצעות %argName. בכל מקום שהארגומנט מופיע בתוך המאקרו, הוא יוחלף בערך שיישלח אליו בזמן הקריאה למאקרו.	- החזרת תוצאות מהפרוצדורה באמצעות \$v0-\$v1.
- הגדרת הארגומנטים בתוך סוגריים בזמן ההכרזה על המאקרו.	- לפרוצדורה אסור לשנות את אוגרים \$s0-\$s7, במידה והיא רוצה לשנות אותם היא צריכה לשמור את הנתונים במחסנית ולהחזיר אותם לפני סיום הפרוצדורה.
- אין להגדיר מאקרו בתוך מאקרו.	- לפרוצדורה מותר לשנות את אוגרים \$t0-\$t9, במידה והתוכנית שקוראת לפרוצדורה רוצה לשמור על הערכים שיש באוגרים אלו, עליה לשמור אותם במחסנית לפני הקריאה לפרוצדורה ולהחזירם לאחר החזרה מהפרוצדורה.
- אפשר לזמן מאקרו מתוך מאקרו אחר בתנאי שהוא הוגדר לפניו.	- במידה ופרוצדורה קוראת לפרוצדורה אחרת, היא צריכה לשמור במחסנית את הערך באוגר \$ra לפני הקריאה ולהחזיר אותו בסיום החזרה מהפרוצדורה הפנימית.

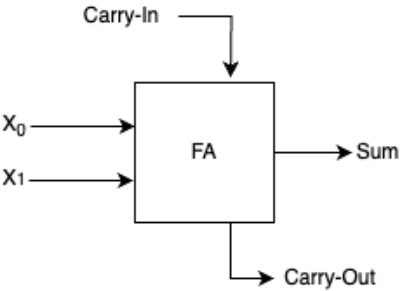
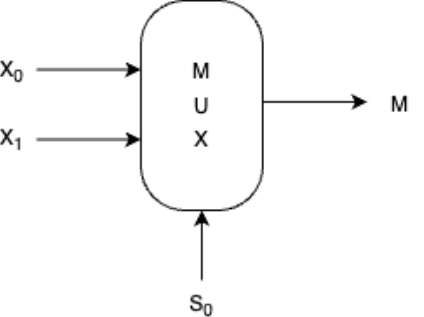
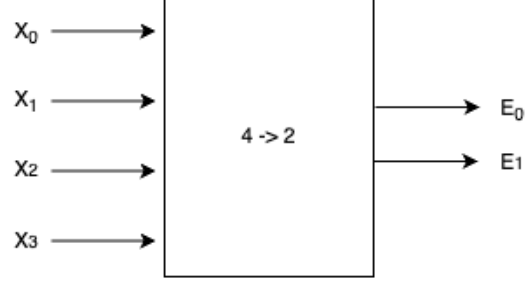
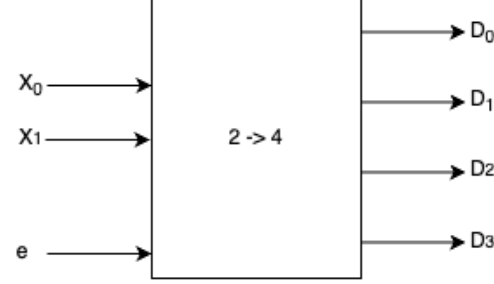
ניהול הזיכרון במחסנית באחריותו הבלעדית של המתכנת(!) התאים במחסנית בגודל של 4 בתים, ולכן הקפיצה בכתובות תמיד ב-4.

לפני כל הכנסה של נתון למחסנית יש להקטין את אוגר \$sp ב-4. לאחר כל קריאה מהמחסנית יש להגדיל את אוגר \$sp ב-4.

```
callerProcedure:
addi $sp, $sp, -4
sw $ra, 0($sp)
jal calleeProcedure
lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra
```

מעבד חד מחזורי - סיכום

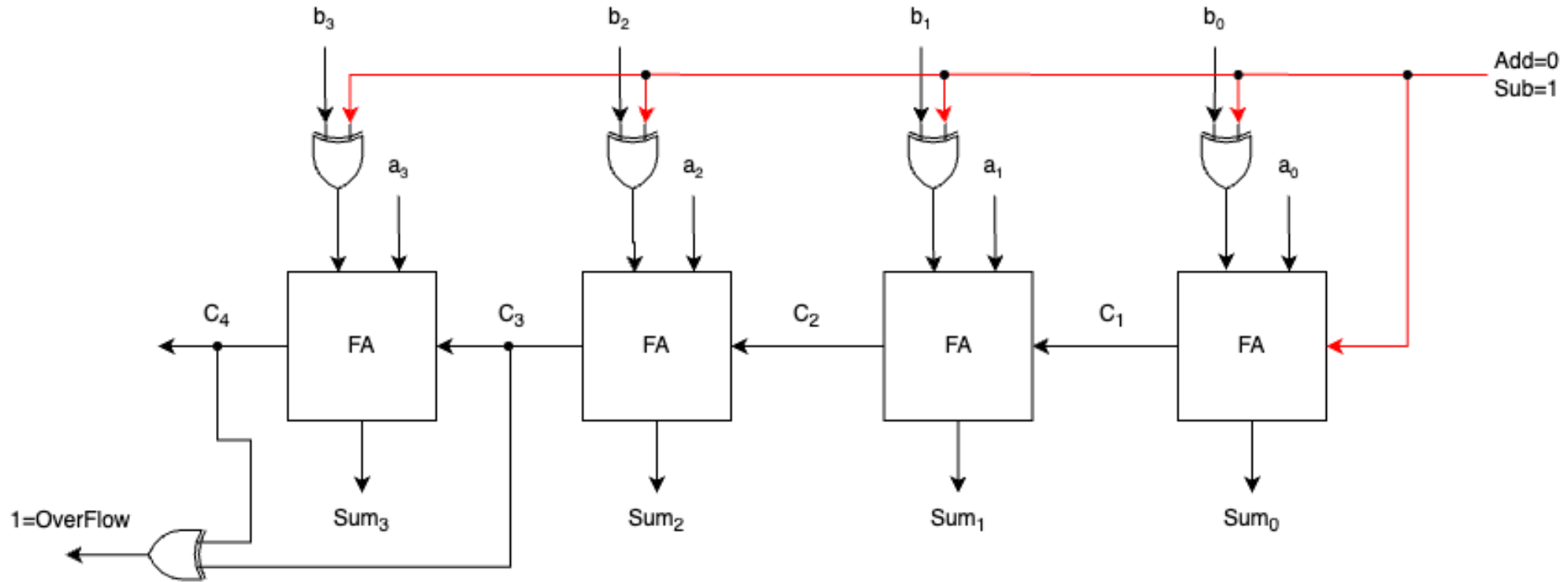


מחבר (Adder)	מרבב (Multiplexer)	מקודד (Encoder)	מפענח (Decoder)																																																																																																										
																																																																																																													
<table border="1" data-bbox="82 611 552 921"> <thead> <tr> <th>C-I</th> <th>X_1</th> <th>X_0</th> <th>C-O</th> <th>SUM</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	C-I	X_1	X_0	C-O	SUM	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	0	1	1	1	0	1	0	1	1	0	1	1	0	1	0	1	1	1	1	1	<table border="1" data-bbox="764 697 952 842"> <thead> <tr> <th>S_0</th> <th>M</th> </tr> </thead> <tbody> <tr><td>0</td><td>X_0</td></tr> <tr><td>1</td><td>X_1</td></tr> </tbody> </table>	S_0	M	0	X_0	1	X_1	<table border="1" data-bbox="1140 664 1669 875"> <thead> <tr> <th>X_3</th> <th>X_2</th> <th>X_1</th> <th>X_0</th> <th>E_1</th> <th>E_0</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> </tbody> </table>	X_3	X_2	X_1	X_0	E_1	E_0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	1	0	1	0	0	0	1	1	<table border="1" data-bbox="1705 664 2234 875"> <thead> <tr> <th>X_1</th> <th>X_0</th> <th>D_3</th> <th>D_2</th> <th>D_1</th> <th>D_0</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr> </tbody> </table>	X_1	X_0	D_3	D_2	D_1	D_0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	0	1	0	0	1	1	1	0	0	0
C-I	X_1	X_0	C-O	SUM																																																																																																									
0	0	0	0	0																																																																																																									
0	0	1	0	1																																																																																																									
0	1	0	0	1																																																																																																									
0	1	1	1	0																																																																																																									
1	0	1	1	0																																																																																																									
1	1	0	1	0																																																																																																									
1	1	1	1	1																																																																																																									
S_0	M																																																																																																												
0	X_0																																																																																																												
1	X_1																																																																																																												
X_3	X_2	X_1	X_0	E_1	E_0																																																																																																								
0	0	0	1	0	0																																																																																																								
0	0	1	0	0	1																																																																																																								
0	1	0	0	1	0																																																																																																								
1	0	0	0	1	1																																																																																																								
X_1	X_0	D_3	D_2	D_1	D_0																																																																																																								
0	0	0	0	0	1																																																																																																								
0	1	0	0	1	0																																																																																																								
1	0	0	1	0	0																																																																																																								
1	1	1	0	0	0																																																																																																								
<p>מקבל 2 כניסות ומוציא את תוצאת החיבור שלהן וכן את הנשא המתקבל כתוצאה מהחיבור. מבדילים בין חצי מחבר (Half Adder) שאיננו מקבל נשא נכנס אלא רק שתי כניסות, לבין מחבר מלא (Full Adder) המקבל גם 2 כניסות וגם נשא נכנס.</p>	<p>מקבל 2^n כניסות ו-n כניסות בקרה ומעביר את אחת הכניסות ליציאה על פי הערך בכניסת הבקרה. נקרא גם "בורר" או "mux".</p>	<p>מקבל 2^n כניסות, ומוציא n יציאות. המקודד מדליק את היציאה שהמספר שלה מיוצג בינארית בכניסות למפענח.</p>	<p>מקבל n כניסות, ומוציא 2^n יציאות. המפענח מדליק את היציאה שהמספר שלה מיוצג בינארית בכניסות למפענח. אפשר להוסיף כניסת enable למפענח וכאשר היא שווה 0, כל היציאות יהיו שוות 0.</p>																																																																																																										

מעבד חד מחזורי - סיכום



מחבר ומחסר מלא לכמה סיביות (בשיטת משלים ל-2)



מעבד חד מחזורי - סיכום



SOP (sum of product) סכום המכפלות ו-POS (product of sum) מכפלת הסכומים

בהינתן טבלת אמת של מספר משתנים, נוכל לבנות פונקציה לוגית המשתמשת בשערים AND, OR ו-NOT בכדי להציג את טבלת האמת. ישנן שתי גישות בכדי להגיע לביטוי אלגברי המשיג את דרישות טבלת האמת:

POS מכפלת הסכומים		SOP סכום המכפלות	
נתייחס רק לשורות שבהן ערך הפונקציה שווה 0. כל משתנה המופיע בשורה זו כ-0, יופיע בסכום כפי שהוא. כל משתנה המופיע כ-1, יופיע בסכום כהופכי של המשתנה. נכפול את כל הסכומים.		נתייחס רק לשורות שבהן ערך הפונקציה שווה 1. כל משתנה המופיע בשורה זו כ-1, יופיע במכפלה כפי שהוא. כל משתנה המופיע כ-0, יופיע במכפלה כהופכי של המשתנה. נסכום את כל המכפלות.	
0	שורות רלוונטיות	1	שורות רלוונטיות
\bar{x}	משתנה שווה 1	x	משתנה שווה 1
x	משתנה שווה 0	\bar{x}	משתנה שווה 0
maxterm	סכום סטנדרטי	minterm	מכפלה סטנדרטית
$F(x, y, z) = \prod M_i$	סימון הפונקציה	$F(x, y, z) = \sum m_i$	סימון הפונקציה

לדוגמה:

SOP סכום המכפלות

$$F(x, y, z) = \sum m_i = \sum (2,3,4) = \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}\bar{z} + xyz = x\bar{y}\bar{z} + yz + \bar{x}y$$

POS מכפלת הסכומים

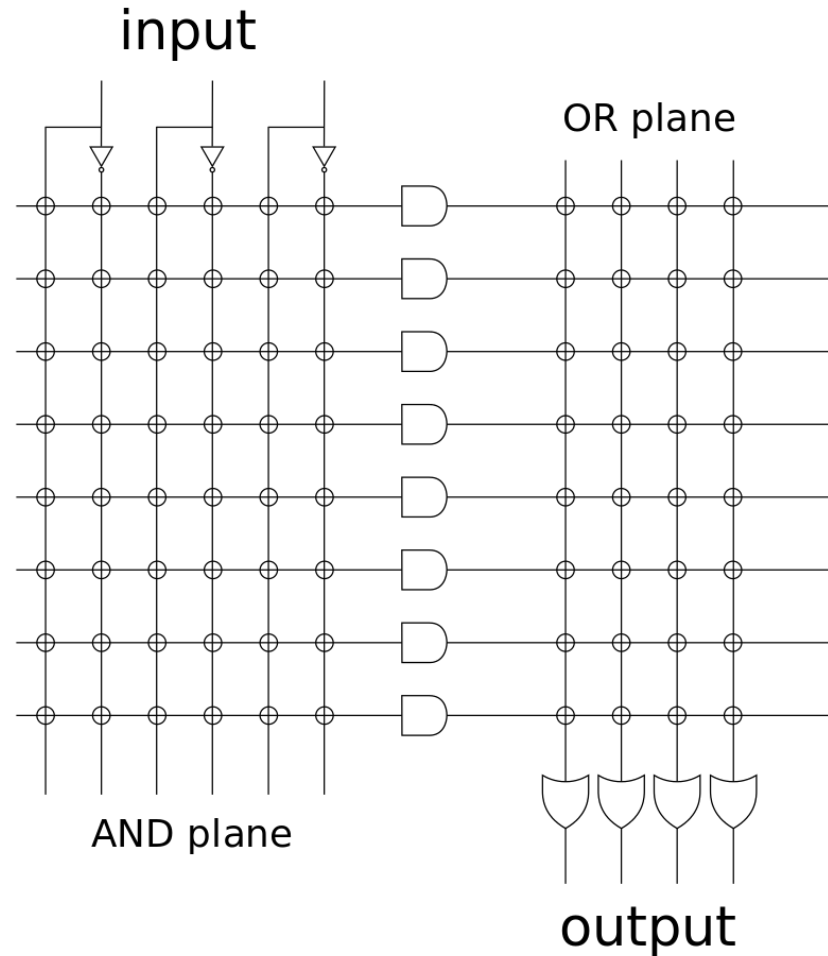
$$F(x, y, z) = \prod M_i = \prod (0,1,5,6) = (x + y + z)(x + y + \bar{z})(\bar{x} + y + \bar{z})(\bar{x} + \bar{y} + z) = x\bar{y}\bar{z} + yz + \bar{x}y$$

	x	y	z	f
m_0	0	0	0	0
m_1	0	0	1	0
m_2	0	1	0	1
m_3	0	1	1	1
m_4	1	0	0	1
m_5	1	0	1	0
m_6	1	1	0	0
m_7	1	1	1	1

מעבד חד מחזורי - סיכום



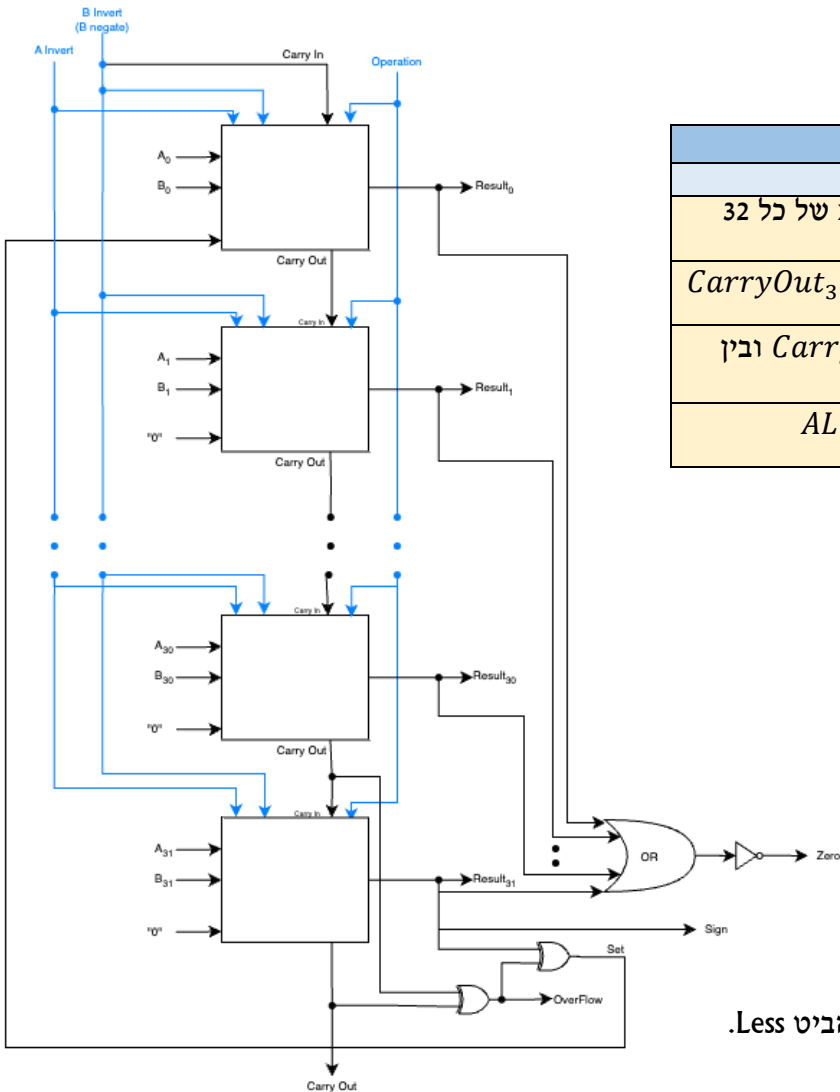
PLA - Programmable logic array



מעבד חד מחזורי - סיכום



שרשור 32 יחידות ALU



חיווי על תוצאות ה-ALU

סימון	שם	משמעות	בשביל מה זה טוב?	איך זה קורה?
Z	Zero	ביט השווה 1 אם כל הביטים של התוצאה שווים 0	חיווי בפקודות branch האם יש לבצע את הקפיצה או לא	היפוך סימן לפעולת xor של כל 32 התוצאות
C	Carry Out	ביט השווה 1 אם חיבור שני הביטים ב-MSB הוציא נשא	חיווי לגלישה בשיטת ייצוג ללא סימן	תוצאת הנשא האחרון $CarryOut_{31}$
O	Over Flow	ביט השווה 1 אם יש גלישה לפי משלים ל-2	חיווי לגלישה בשיטת משלים ל-2	פעולת xor בין $CarryOut_{30}$ ובין $CarryOut_{31}$
S	Sign	ביט השווה 1 אם תוצאת ה-MSB שווה 1	חיווי למספר שלילי בשיטת משלים ל-2	תוצאת החיבור של ALU_{31}

פעולות ה-ALU כתוצאה מכניסות הבקרה

למה זה קורה?	Operation	B Invert	A Invert	מה יקרה?
	00	0	0	"וגם"
	01	0	0	"או"
	10	0	0	"חיבור"
$A + (-B) = A - B$	10	1	0	"חיסור"
(לוגי) $\overline{(A + B)} = \overline{A} \cdot \overline{B}$	00	1	1	"לא או"
(לוגי) $\overline{(A \cdot B)} = \overline{A} + \overline{B}$	01	1	1	"לא וגם"
	11	0	1	"הדלק אם קטן מ"

הזמן שלוקח לבצע פעולה לוגית קצר מכיוון שכל הביטים מחושבים במקביל. הזמן שלוקח לבצע פעולה אריתמטית יותר ארוך כיוון שהחישוב מבוצע בטור.

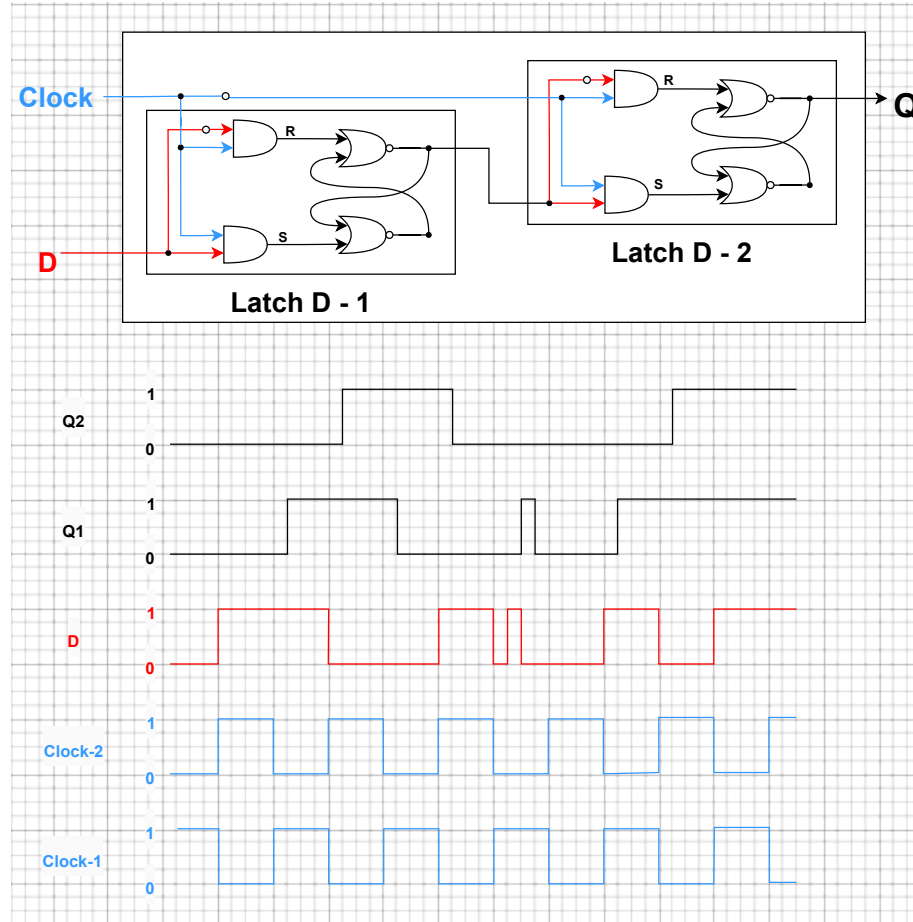
זמן הפעולה של ה-ALU הוא קצת יותר ארוך מפעולה אריתמטית כיוון שבפקודת SLT יש להוסיף את הזמן של הדלקת הביט Less.

מעבד חד מחזורי - סיכום



מנעול בקצה שעות - Latch D Edge Triggered

מנעול זה משורשר 32 פעמים עבור כל סיבית באוגר ואז מוכפל 32 פעמים עבור 32 אוגרים

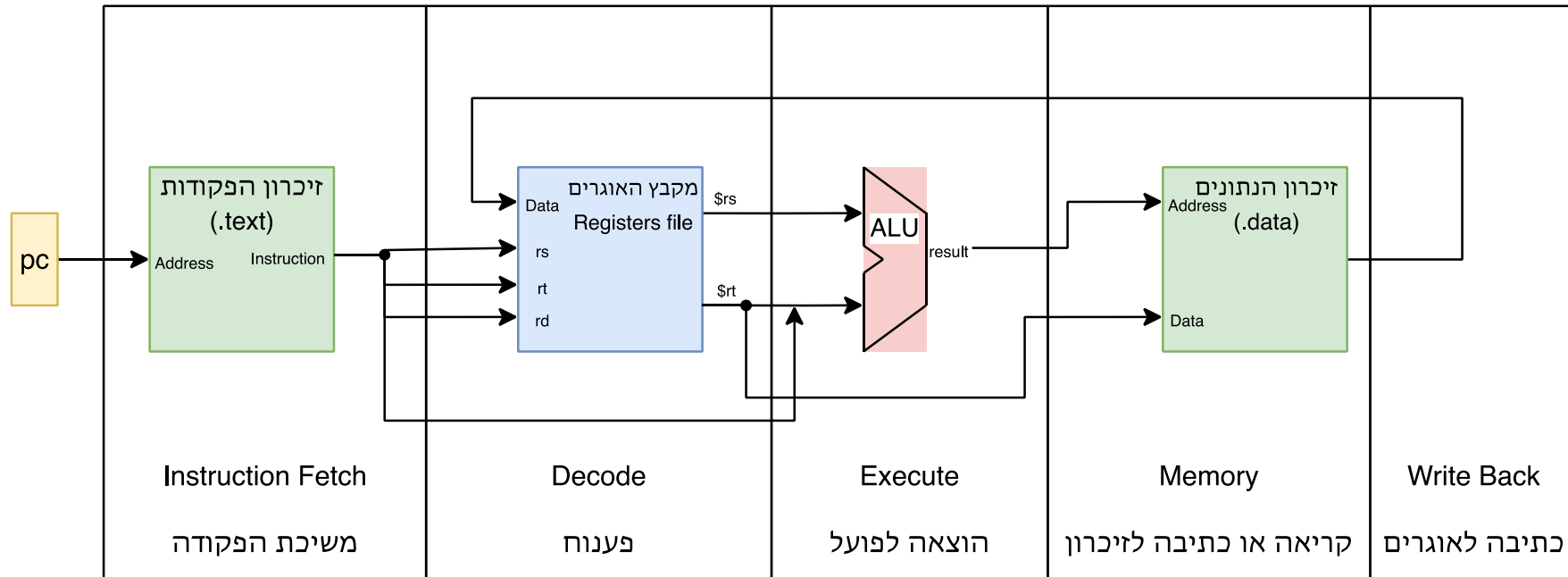


מעבד חד מחזורי - סיכום



חמשת שלבי הביצוע של פקודות במעבד

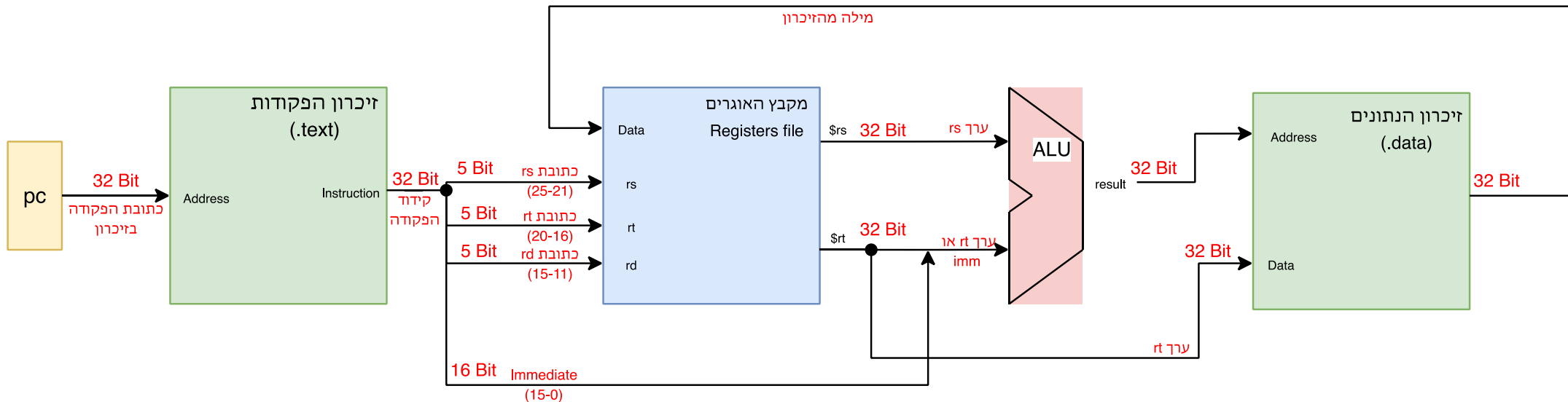
מתי רלוונטי?	שלב		
תמיד	משיכת הפקודה	Instruction Fetch	1
	פענוח (וקריאת האוגרים)	Decode	2
תמיד חוץ מפקודת jump	הוצאה לפועל	Execute	3
רק בפקודות lw, sw	קריאה או כתיבה לזיכרון	Memory	4
בכל הפעולות האריתמטיות / לוגיות ובפקודה lw	כתיבה לאוגרים	Write Back	5



מעבד חד מחזורי - סיכום



רוחב הפסים ומשמעות המידע הזורם בהם

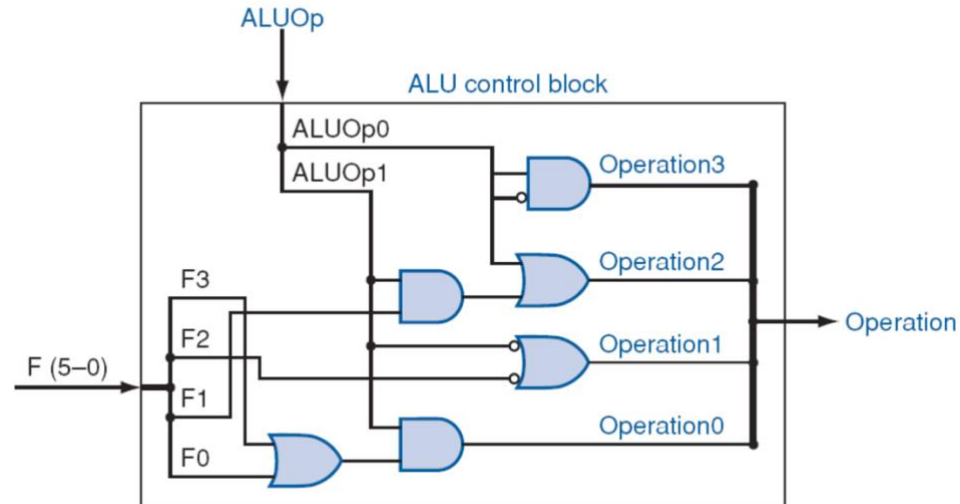


מעבד חד מחזורי - סיכום



כניסות ויציאות הבקרה של יחידת הבקרה המשנית (יחידת הבקרה של ה-ALU) במעבד עם הפקודות - ADD, SUB, AND, OR, LW, SW, BEQ, J

פקודה	ALUOp1	ALUOp0	func	F3	F2	F1	F0	A invert op-3	B invert op-2	Operation op-10
add	1	0	100000	0	0	0	0	0	0	10
sub	1	0	100010	0	0	1	0	0	1	10
and	1	0	100100	0	1	0	0	0	0	00
or	1	0	100101	0	1	0	1	0	0	01
slt	1	0	101010	1	0	1	0	0	1	11
beq	0	1	xxxxxx	x	x	x	x	0	1	10
lw	0	0	xxxxxx	x	x	x	x	0	0	10
sw	0	0	xxxxxx	x	x	x	x	0	0	10

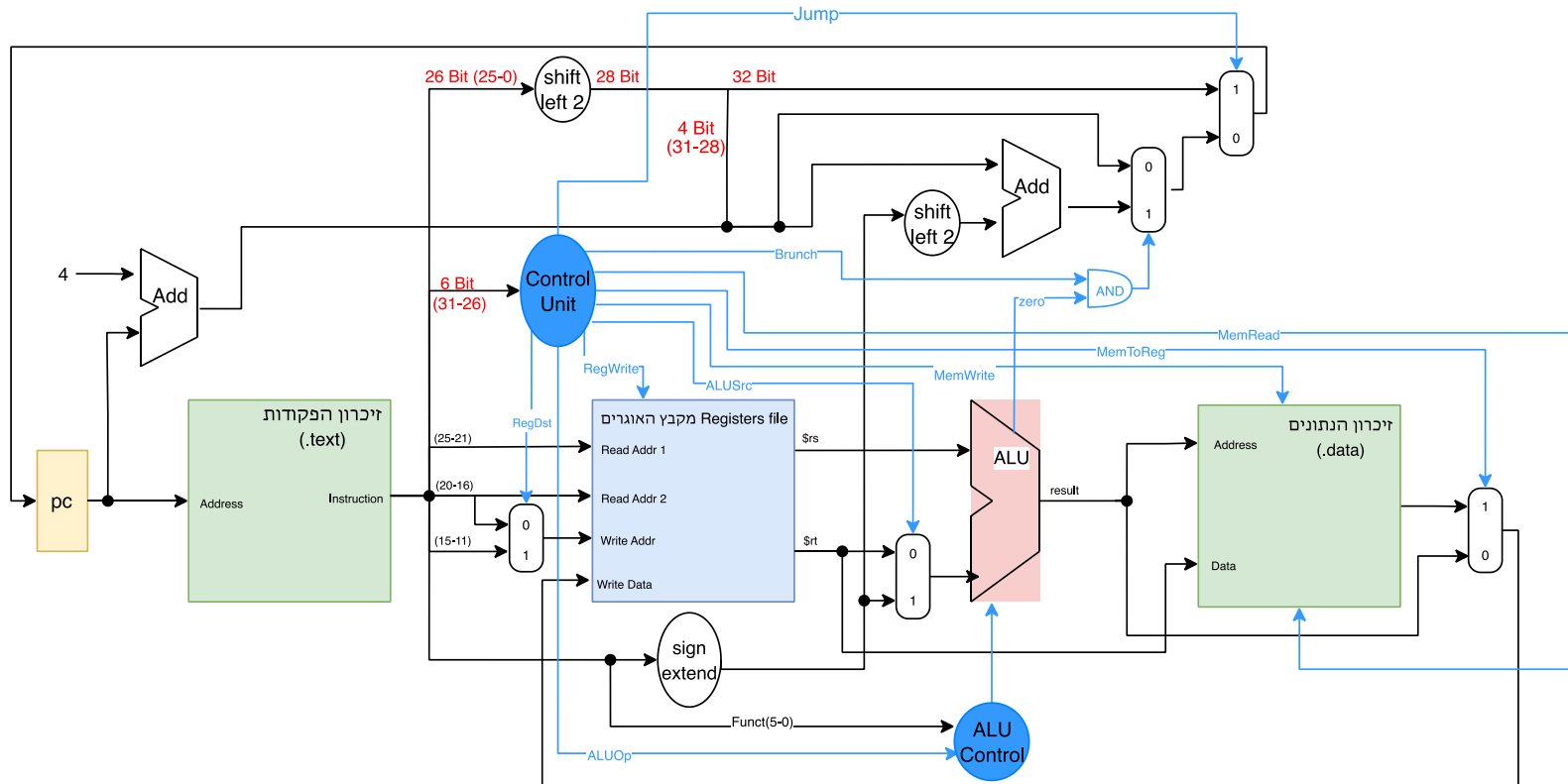


מעבד חד מחזורי - סיכום



מעבד המסוגל לבצע את הפקודות - ADD, SUB, AND, OR, LW, SW, BEQ, J

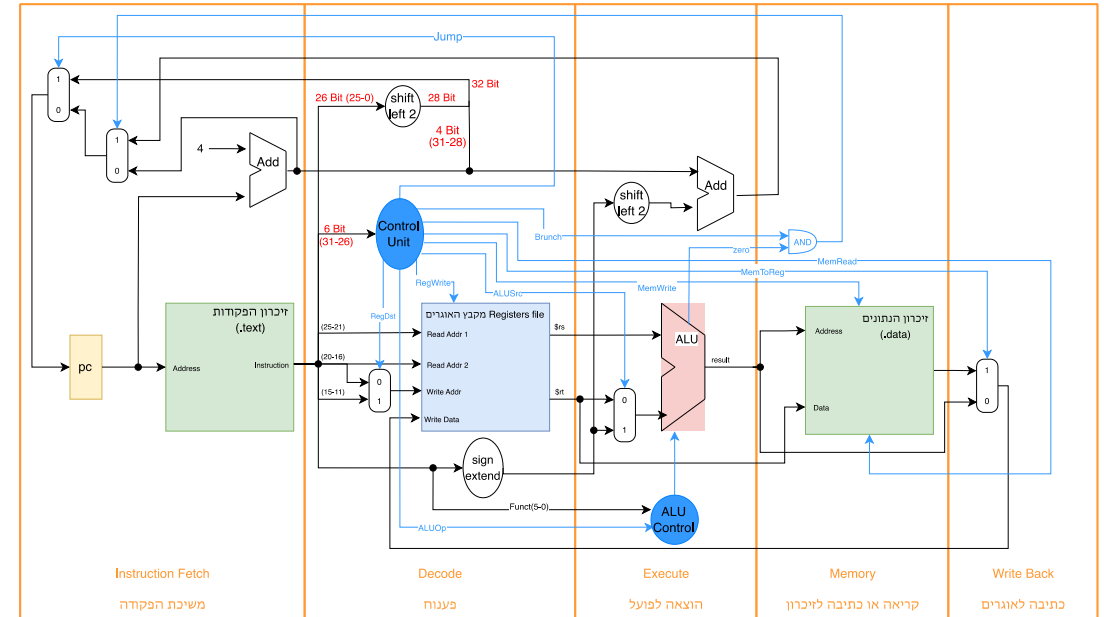
פקודה	RegDst	ALUSrc	MemToReg	RegWrite	MemRead	MemWrite	Branch	Jump	ALUOp1	ALUOp0
טיפוס R	1	0	0	1	0	0	0	0	1	0
sw	x	1	x	0	0	1	0	0	0	0
lw	0	1	1	1	1	0	0	0	0	0
beq	x	0	x	0	0	0	1	0	0	1
j	x	x	x	0	0	0	x	1	x	x



מעבד צנרת - סיכום



שלבי הפקודה במעבד והצגתם באופן ויזואלי באמצעות תרשימים



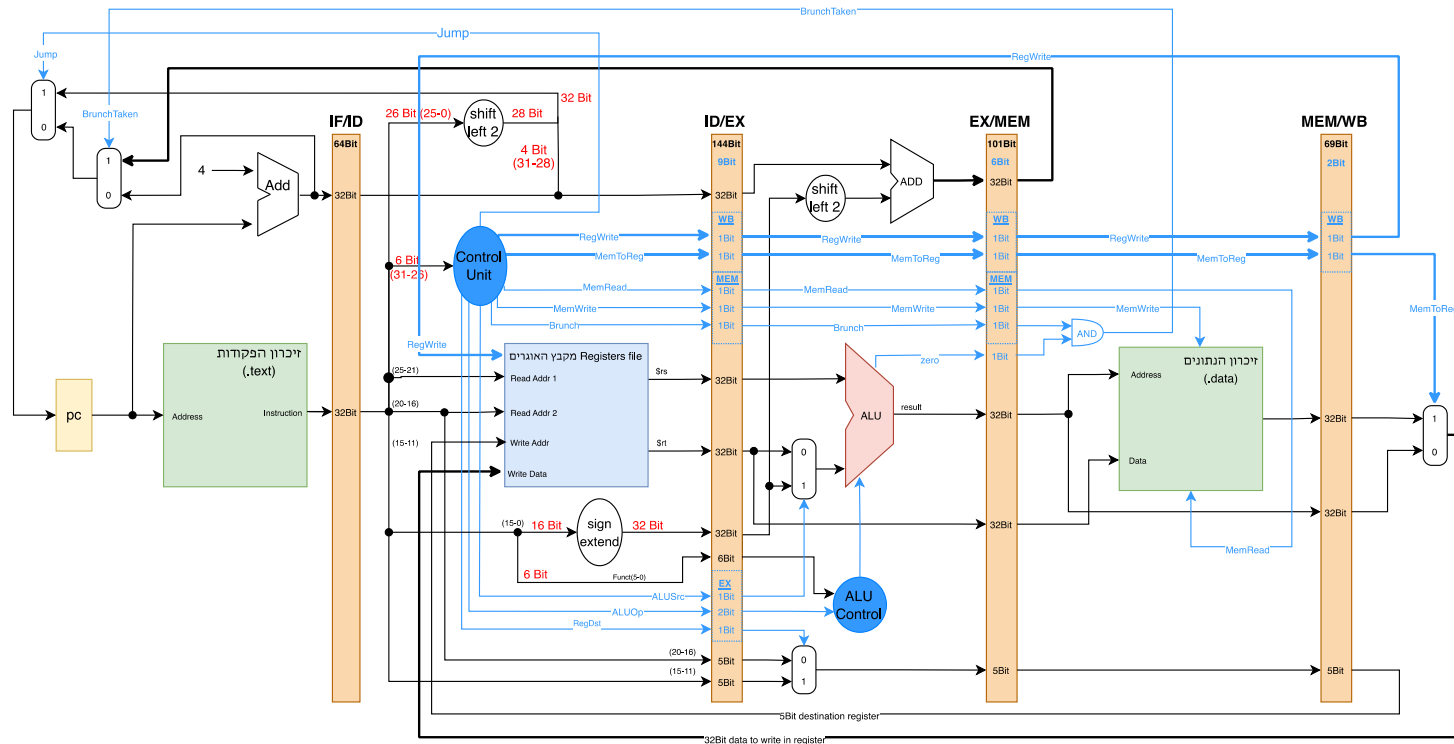
שלב	שם	ראשי תיבות	הרכיב המרכזי	ובקצרה
1	Instruction Fetch	IF	Instruction Memory	IM
2	Instruction Decode	ID	Registers	Rge
3	Execute	EX	ALU	ALU
4	Memory	MEM	Data Memory	DM
5	Write Back	WB	Registers	Reg

מעבד צנרת - סיכום



שימוש באוגרי צנרת וחלוקת אותות הבקרה לפי שלבי הביצוע

	ID	EX				MEM			WB	
פקודה	Jump	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	MemRead	MemWrite	RegWrite	MemToReg
טיפוס R	0	1	1	0	0	0	0	0	1	0
sw	0	X	0	0	1	0	0	1	0	X
lw	0	0	0	0	1	0	1	0	1	1
beq	0	X	0	1	0	1	0	0	0	X
j	1	X	X	X	X	X	0	0	0	X

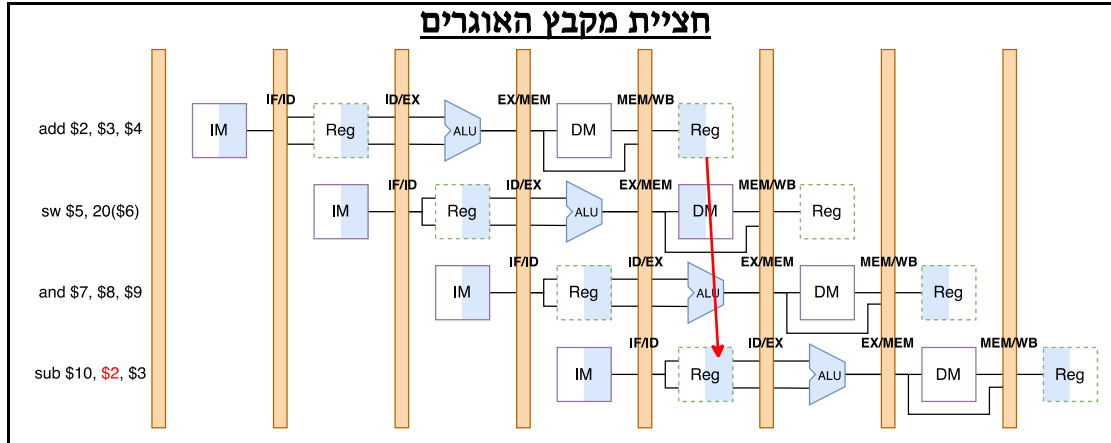


מעבד צנרת - סיכום

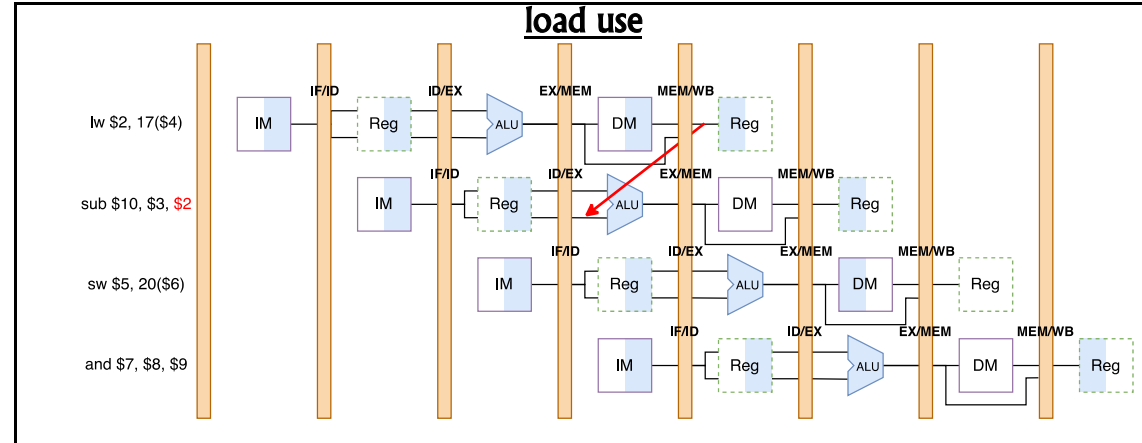


סיכוי נתונים מסוגים שונים

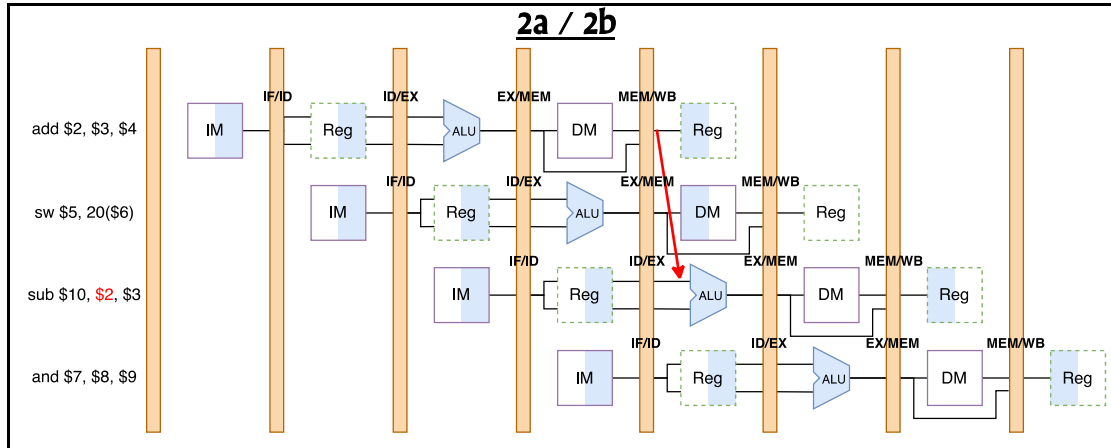
חציית מקבץ האוגרים



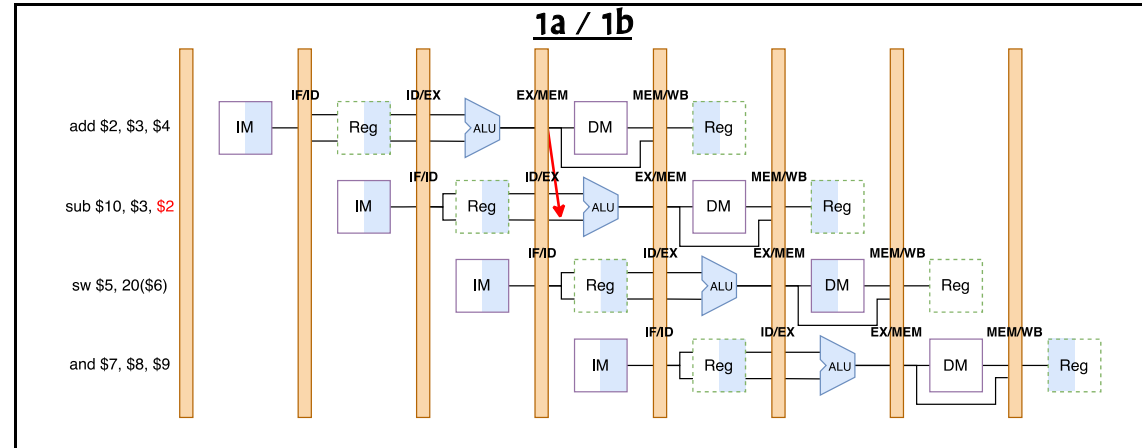
load use



2a / 2b



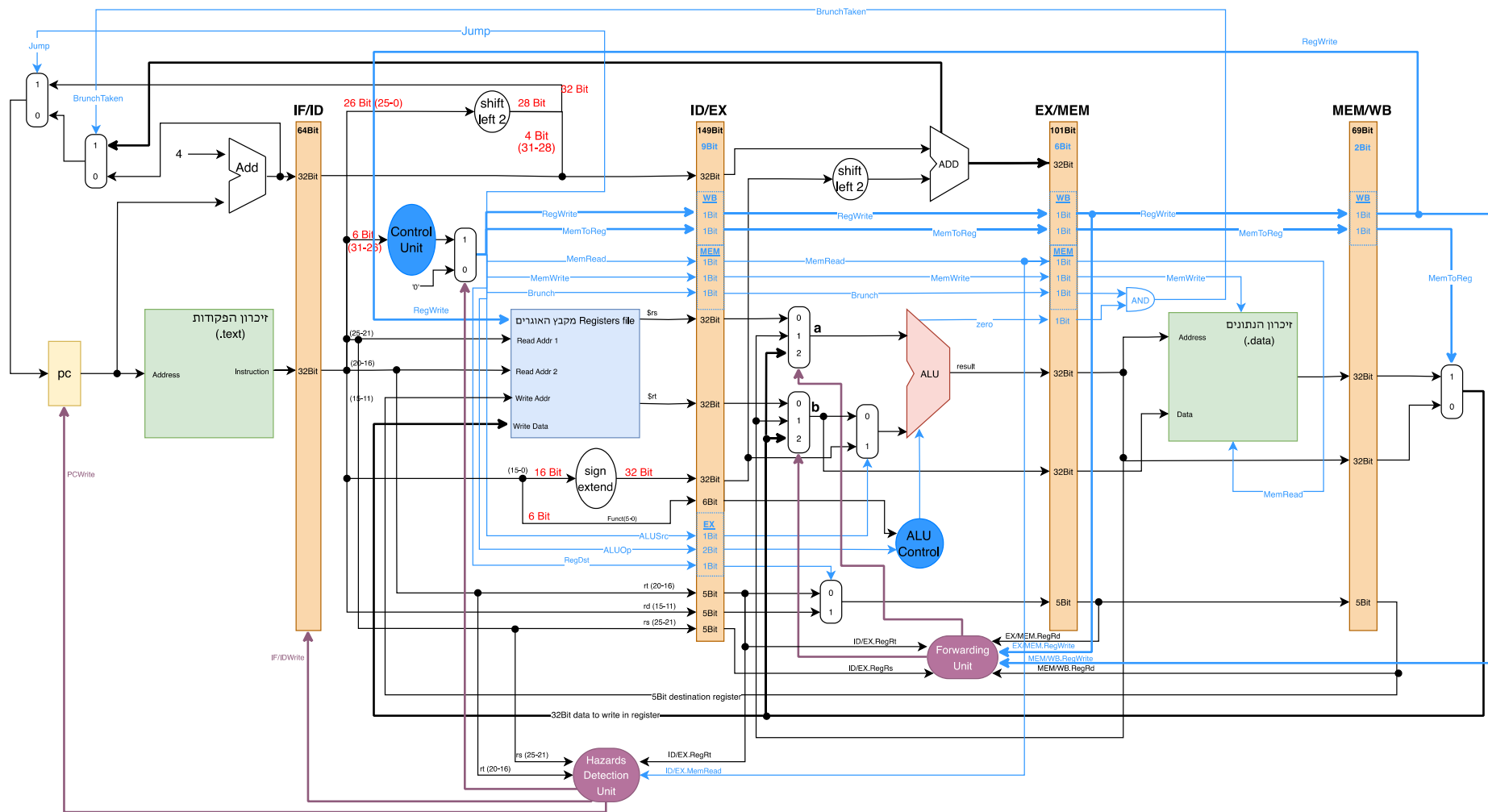
1a / 1b



מעבד צנרת - סיכום



יחידת העברה קדימה לאיתור וטיפול בסיכונים נתונים ויחידה לזיהוי סיכונים לאיתור וטיפול בסיכון load use



מעבד צנרת - סיכום



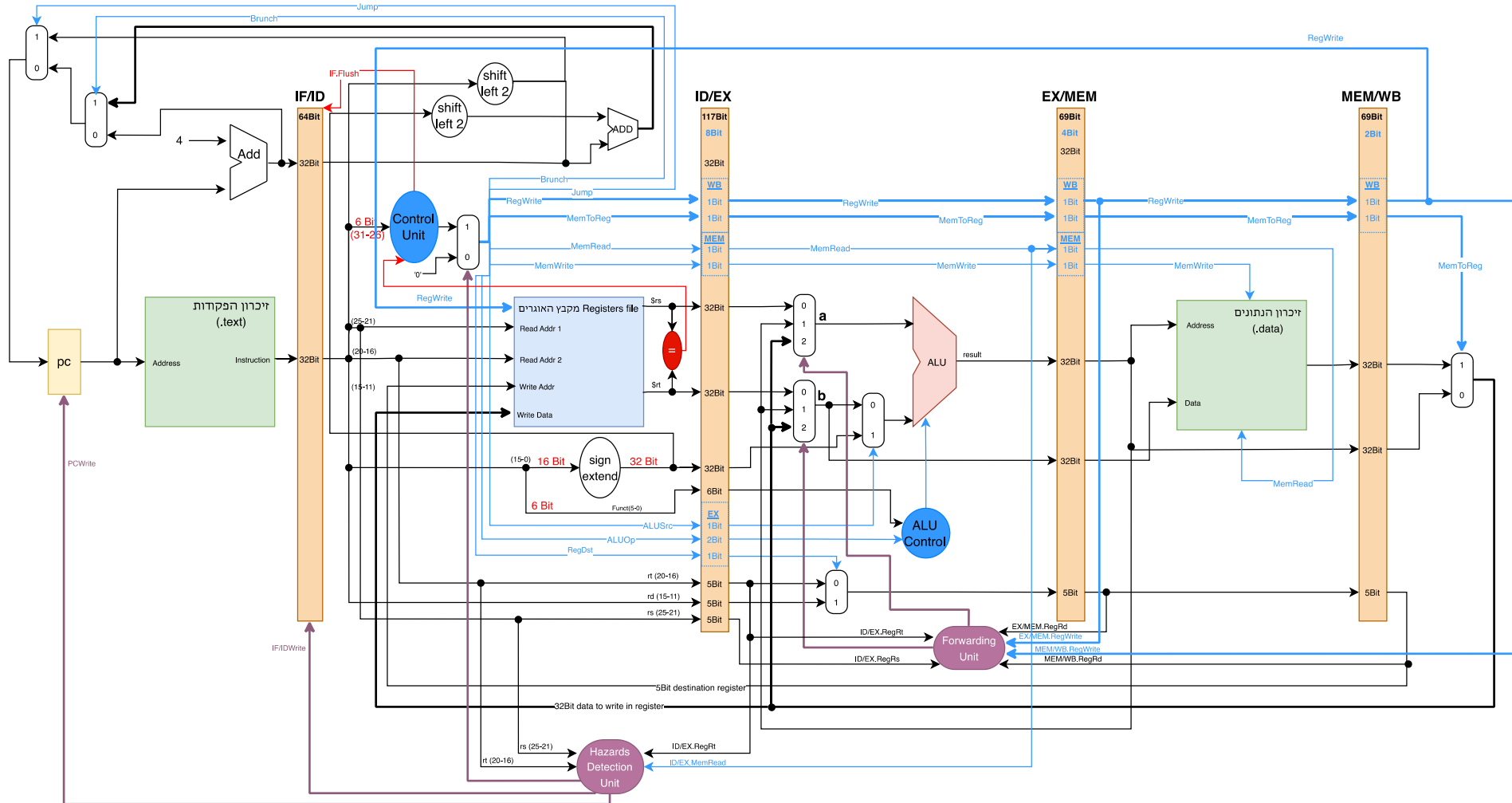
בדיקות שהיחידה להעברה קדימה - Forwarding Unit - מבצעת				
הבדיקות שיחידת העברה קדימה עושה		אוגר המקור של הפקודה הבאה		מרחק בין פקודות
1a	EX/MEM.RegRd=ID/EX.RegRs & EX/MEM.RegWrite=1 & EX/MEM.RegRd!=0	rs אוגר מקור = rd	a	פקודה אחרי פקודה
1b	EX/MEM.RegRd=ID/EX.RegRt & EX/MEM.RegWrite=1 & EX/MEM.RegRd!=0	rt אוגר מקור = rd	b	
2a	MEM/WB.RegRd=ID/EX.RegRs & MEM/WB.RegWrite=1 & MEM/WB.RegRd!=0	rs אוגר מקור = rd	a	פקודה עם רווח של פקודה אחת באמצע
2b	MEM/WB.RegRd=ID/EX.RegRt & MEM/WB.RegWrite=1 & MEM/WB.RegRd!=0	rt אוגר מקור = rd	b	

בדיקות שהיחידה לזיהוי סיכונים - Hazards Detection Unit - מבצעת	
הבדיקות שיחידת העברה קדימה עושה	אוגר המקור של הפקודה הבאה
ID/EX.RegRd=IF/ID.RegRs & ID/EX.MemRead=1	rs אוגר מקור = rd
ID/EX.RegRd=IF/ID.RegRt & ID/EX.MemRead=1	rt אוגר מקור = rd

מעבד צנרת - סיכום



סיכוי בקרה - הקדמת Branch לשלב 2



מעבד צנרת - סיכום



Scheduling					
הנחות שיש להניח	דוגמה לקוד אחרי	דוגמה לקוד לפני	הנחת האופטימיזר	מקרה	
אין תלות בתוצאת ה-Brunch ולכן אין צורך להניח שום דבר	beq \$4, \$5, Target sub \$1, \$2, \$3 ... add \$6, \$7, \$8 Target: ... and \$9, \$10, \$11 or \$12, \$13, \$14	sub \$1, \$2, \$3 beq \$4, \$5, Target ... add \$6, \$7, \$8 Target: ... and \$9, \$10, \$11 or \$12, \$13, \$14		ייבוא פקודה הנמצאת לפני ה-Brunch	a
1. מניחים שהמהדר מסוגל לקבוע בצורה דיי מובהקת האם ה-Brunch ילקח או לא, ולכן הוא יכול למלא בפקודה שרוב הסיכויים שהיא באמת צריכה להתבצע. 2. מניחים שהמהדר יודע לבחור פקודות למילוי שגם אם הן תתבצעה הן לא יגרמו לטעות לוגית בתוכנית אלא רק יבזבו מחזור שעון אחד.	sub \$1, \$2, \$3 beq \$4, \$5, Target and \$9, \$10, \$11 ... add \$6, \$7, \$8 Target: ... or \$12, \$13, \$14	sub \$1, \$2, \$3 beq \$4, \$5, Target ... add \$6, \$7, \$8 Target: ... and \$9, \$10, \$11 or \$12, \$13, \$14	Brunch Taken	ייבוא פקודה הנמצאת בכתובת היעד של פקודת ה-Brunch	b
	sub \$1, \$2, \$3 beq \$4, \$5, Target add \$6, \$7, \$8 ... Target: ... and \$9, \$10, \$11 or \$12, \$13, \$14	sub \$1, \$2, \$3 beq \$4, \$5, Target ... add \$6, \$7, \$8 Target: ... and \$9, \$10, \$11 or \$12, \$13, \$14	sub \$1, \$2, \$3 beq \$4, \$5, Target ... add \$6, \$7, \$8 Target: ... and \$9, \$10, \$11 or \$12, \$13, \$14	Brunch NOT Taken	ייבוא פקודה הנמצאת אחרי פקודת ה-Brunch

מעבד צנרת - סיכום



חיזוי דינאמי במעבדים בעלי צנרת עמוקה

BHT (Branch History Table)			1-bit Prediction Accuracy		
כתובת הפקודה	תוצאת ה-Branch האחרונה				
BTB (Branch Target Buffer)					
כתובת הפקודה	כתובת היעד לקפיצה	תוצאת ה-Branch אחת לפני האחרונה	תוצאת ה-Branch האחרונה	מיקום	חיזוי
		0	0	0	0
		0	1	0	0
		1	1	0	1
		1	0	0	1
		0	1	1	1
		1	0	1	1
		0	0	1	0
		1	1	1	1


```

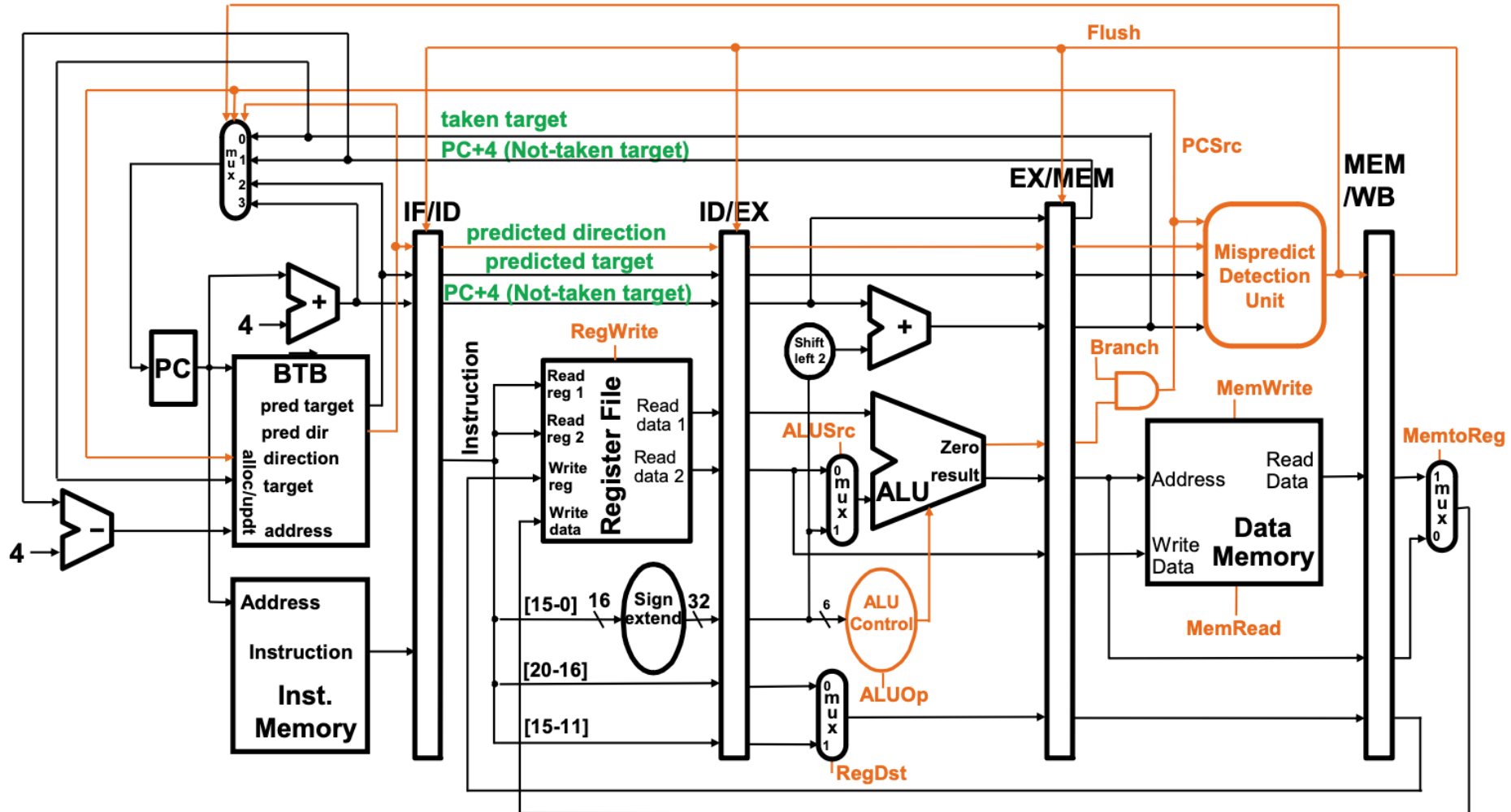
    graph TD
      00((חיזוי: לא נלקח  
00)) -- "תוצאה אחרונה 0" --> 00
      00 -- "תוצאה אחרונה 1" --> 01((חיזוי: לא נלקח  
01))
      01 -- "תוצאה אחרונה 0" --> 00
      01 -- "תוצאה אחרונה 1" --> 11((חיזוי: נלקח  
11))
      11 -- "תוצאה אחרונה 0" --> 10((חיזוי: נלקח  
10))
      11 -- "תוצאה אחרונה 1" --> 11
      10 -- "תוצאה אחרונה 0" --> 00
      10 -- "תוצאה אחרונה 1" --> 11
    
```

2-bit Prediction Accuracy

מעבד צנרת - סיכום



חיזוי דינאמי באמצעות BTB במעבד MIPS



מעבד צנרת - סיכום



פסיקות וחריגות

הטרמינולוגיה של ההפרעות במעבד ה-MIPS:

הטרמינולוגיה של MIPS	מהיכן הגיעה ההפרעה	סוג ההפרעה
פסיקות (interrupts)	חיצוני	I/O - פעולות קלט / פלט של רכיבי חומרה שונים
חריגות (exception)	פנימי	שימוש במערכת ההפעלה מתוך התוכנית
חריגות (exception)	פנימי	גלישה אריתמטית
חריגות (exception)	פנימי	שימוש בפקודה לא חוקית
פסיקות (interrupts) או חריגות (exception)	חיצוני / פנימי	תקלת חומרה

במעבד ה-MIPS קיימים מספר אוגרים המסייעים בטיפול בחריגות. אנחנו נזכיר שניים מהם:

1. אוגר הנקרא cause - נועד לשמירת הסיבה שגרמה לפסיקה / חריגה.
2. אוגר הנקרא EPC (Exception PC) - נועד לשמירת הכתובת של הפקודה המגיעה מיד אחרי הפקודה שגרמה לחריגה, למקרה שמערכת ההפעלה תרצה לחזור לתוכנית ולהמשיך להריץ אותה.

הטיפול בפסיקות וחריגות מתחלק למספר משימות שהמעבד מבצע:

1. עצירה התוכנית: שטיפת הפקודה שגרמה לתקלה וכן של כל הפקודות שבאות אחריה, סיום ביצוע הפקודות שהיו לפני הפקודה הבעייתית.
 2. שמירת המידע הנדרש בנוגע לתקלה: סיבת התקלה נשמרת באוגר cause וכתובת הפקודה שגרמה לתקלה נשמרת באוגר EPC.
 3. קפיצה לכתובת 0x80000180 שבה נמצא ה-exception handler של מערכת ההפעלה שהוא בודק את סיבת הפסיקה ובהתאם מחליט מה לעשות וכיצד לנהוג.
- במידה ושתי פקודות שונות הנמצאות בשני שלבים שונים של הצנרת גורמות לחריגה באותו הזמן, מעבד ה-MIPS יעדיף לטפל בחריגה שנוצרה מהפקודה הראשונה שנכנסה לצנרת (זאת אומרת: הפקודה הנמצאת בשלב מתקדם יותר בצנרת), אך יש מעבדים אשר בוחרים באיזו חריגה לטפל בשיטות שונות.



ניהול היררכיית זיכרון - סיכום

ניהול היררכיית זיכרון נועד כדי להרוויח את המהירות של זיכרונות נדיפים מסוג SRAM מבלי לשלם מחיר גבוה על נפח גדול של זיכרון מסוג זה אלא על ידי שימוש בעיקר בזיכרון מסוג DRAM שהוא אמנם איטי בצורה משמעותית אך גם זול מאוד.

מושגים שונים בניהול היררכיית זיכרון			
מושגי יסוד	גישה של המעבד לזיכרון המטמון ומציאת הנתון המבוקש	פגיעה	Hit
	גישה של המעבד לזיכרון המטמון וגילוי שהנתון המבוקש לא נמצא בו	החטאה	Miss
	הזמן שלוקח לגשת לזיכרון המטמון ולקבל ממנו נתון שנמצא בו	זמן פגיעה	Hit Time
	הזמן שלוקח לגשת לזיכרון המטמון, לגלות שאין בו את הנתון המבוקש, לחכות שהזיכרון יפנה לרמה שמתחתיו וייבא את הנתון המבוקש אליו וקבלת הנתון המבוקש אל המעבד	זמן החטאה	Miss Penalty
	אחוז הפעמים שהמעבד פונה לזיכרון המטמון ומוצא בו את הנתון המבוקש	שיעור הפגיעה	Hit Rate
	אחוז הפעמים שהמעבד פונה לזיכרון המטמון ולא מוצא בו את הנתון המבוקש ($Hit Rate + Miss Rate = 1$)	שיעור החטאה	Miss Rate
	לרוב, בלוק יהיה בגודל של מילה אחת / 2 מילים / 4 מילים / 8 מילים וכן הלאה	חבילה של נתונים בגודל כלשהו	Block
	במיפוי אסוציאטיבי חלקי, שדה האינדקס של הכתובת מהזיכרון הראשי יקבע לאיזה סט הכתובת תיכנס	כניסה לשורה ספציפית בזיכרון	Set
	במיפוי אסוציאטיבי חלקי, לאחר שמיפוינו כתובת כלשהי לסט מסוים, הנתון יוכל להישמר בכל אחד מה-Wayים שיש באותו סט	מקום בתוך הסט שאפשר לשמור בו בלוק	Way
	הזמן הממוצע שלוקח להביא נתון מהזיכרון למעבד: $AMAT = Hit Time + Miss Rate \cdot Miss Penalty$	זמן ממוצע לגישה לזיכרון	AMAT
סיבות להחטאה	החטאה שנגרמה כתוצאה מכך שהתא עדיין ריק וסיבית ה-Valid שלו שווה 0	החטאה כי התא ריק	Miss Valid
	החטאה שנגרמה כתוצאה מכך שהתא מלא (סיבית ה-Valid שלו שווה 1), אבל שדה ה-Tag שנמצא בו לא זהה לשדה ה-Tag של הכתובת המבוקשת	החטאה כי התא מלא במשהו אחר	Miss Tag (Miss Conflict)
שיטות כתיבה	שיטת כתיבה לזיכרון הראשי כך שבכל פעם שהמעבד כותב משהו לזיכרון המטמון המידע מועבר מיידית גם לזיכרון הראשי (על פי רוב משתמשים ב-Buffer כדי לא להשהות את המעבד סתם)	כתיבה מיידית	Write Through
	שיטת כתיבה לזיכרון הראשי כך שבכל פעם שהמעבד כותב משהו לזיכרון המטמון הבלוק שנכתב אליו מסומן ב-Dirty Bit, ורק כאשר ינסו להחליף את הבלוק בבלוק אחר, המידע שיש בו ייכתב לזיכרון הראשי	כתיבה רק בהחלפת בלוק	Write Back
שיטות מיפוי	שיטת מיפוי לפיה כל בלוק מהזיכרון הראשי יכול להיות במיקום אחד בלבד בזיכרון המטמון. שיטה שקלה מאוד לניהול ולשליפת נתונים מזיכרון המטמון אך עלולה לגרום להרבה Miss Tag	מיפוי ישיר	DMC - Direct Mapped Cache
	שיטת מיפוי לפיה כל בלוק מהזיכרון הראשי יכול להיות בכל מקום בזיכרון המטמון. שיטה זו אמנם מאפשרת להשתמש בכל מרחב הכתובות בזיכרון המטמון לפני שניתקל ב-Miss Tag הראשון, אך בפועל קשה מאוד לנהל אותה ויש לשלם מחיר כבד מידי של זמן שליפה ארוך וחומרה מסובכת ויקרה	מיפוי אסוציאטיבי מלא	Fully Associative Cache
	שיטת מיפוי לפיה כל בלוק מהזיכרון הראשי יכול להיות במיקום אחד בלבד בזיכרון המטמון אך בתוך כל מיקום כזה יש מספר מקומות פנויים ואפשר שיהיו בתוך הסט הזה כמה בלוקים שונים בעת ובעונה אחת	מיפוי אסוציאטיבי חלקי	K Way Set Associative Cache
שיטות פינוי בזיכרון אסוציאטיבי	פינוי אקראי	Random	
	פינוי הבלוק האחרון שהיה בשימוש	LRU	
	פינוי נכנס ראשון יוצא ראשון	FIFO	
	פינוי נכנס ראשון יוצא ראשון בתנאי שלא היה בשימוש לאחרונה	NMRU	

ניהול היררכיית זיכרון - סיכום



נוסחאות חישוב					נדרש למצוא
נוסחות					
	000000000000000000000000	0000	00	00	
	Tag N-n-m-2	Index n=?	Block Offset m=?	Byte Offset	
	נסמן את כמות הביטים שיש בכתובת ב-N, כמות המילים שיש בזיכרון הראשי הינה: 2^{N-2}				כמות מילים בזיכרון הראשי
	נסמן את כמות הביטים שיש בכתובת ב-N, כמות הבתים שיש בזיכרון הראשי הינה: 2^N				כמות בתים בזיכרון הראשי
	נסמן את כמות הביטים שיש בכתובת ב-N, כמות הביטים שיש בזיכרון הראשי הינה: $2^N \cdot 8$ או $2^{N-2} \cdot 32$				כמות ביטים בזיכרון הראשי
	כמות הביטים באינדקס היא n, וכמות הביטים ב-Word Offset היא m, אם יש מילה אחת בבלוק 2^n מילים, אם יש מספא מילים בבלוק 2^{n+m}				כמות המילים בזיכרון המטמון במיפוי ישיר
	אם יש מילה אחת בבלוק 2^{n+2} מילים, אם יש מספר מילים בבלוק 2^{n+m+2}				כמות הבתים בזיכרון המטמון במיפוי ישיר
	אם יש מילה אחת בבלוק $k \cdot 2^n$ מילים, אם יש מספא מילים בבלוק $k \cdot 2^{n+m}$				כמות המילים בזיכרון המטמון אסוציאטיבי עם k way בכל סט
	אם יש מילה אחת בבלוק $k \cdot 2^{n+2}$ מילים, אם יש מספר מילים בבלוק $k \cdot 2^{n+m+2}$				כמות הבתים בזיכרון המטמון אסוציאטיבי עם k way בכל סט
	$k \cdot 2^n \cdot (32 \cdot 2^m + valid + dirty + Tag) = k \cdot 2^n \cdot (32 \cdot 2^m + 1 + 1 + Tag)$ - יש לבדוק אם יש בזיכרון סיבית dirty, אם לא, להוריד 1				כמות הסיביות שיש במטמון באופן כללי (כולל הסיביות של המידע ושל המיפוי)

ניהול היררכיית זיכרון - סיכום

