

LAB 1

ADVANCED CPU ARCHITECTURE AND HARDWARE ACCELERATORS LAB

IDO RON : 322384330

BENYAMIN OUMANSKY: 322699946

מבוא:

מטרת מעבדה 1 היא ללמוד לדעת איך להשתמש בapbh , במעבדה ניצור מערכת קוד ש邏輯ית מתח מערכות , כך שכל תת מערכת מבצעת פעולות בנפרד לפי בחירתנו.

איך המערכת מורכבת:

: **TOP**

כニיסות: - אות X, אות Y וקו בקרה [3:4]ALUFN שמטרתו לברור הפונקציה שאנו רוצים להשתמש בה.

addersub 01

shifter 10

logic 11

לכל פונקציה יש תת אופציות שוות נבחר על ידי [0:2]ALUFN.

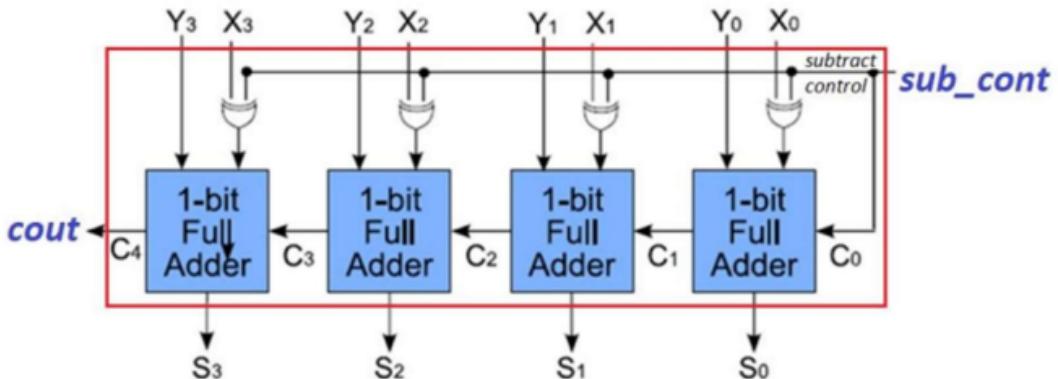
מוצא :

1. מוצא של אחת מהפונקציות שנבחר לפי [4:3]ALU

2. דגלים: בהתאם למוצא יצאו הדגלים המתאים. הדגלים הם: N,Z,C,NEGATIVE,CARRY,ZERO

הרכיבים :

ADDERSUB



משתמשים בפונקציה כאשר $"01" = ALUFN[4:3]$, ויש לה 3 אפשרויות לביצוע: חיבור, חיסור, neg.

אנחנו השתמשנו במבנה של FA וripple adder שהוא נתונים לנו בדוגמאות. והוספנו והתאמנו את הزادה ripple כדי שיכל לבצע גם חיסור וneg.

בררנו לפיה [0:2]ALUFN כדי לבחור איזה תט פולה להשתמש.

האות X נכנס לripple adder שמורכב מ-4 רכיבי FA, בהתאם לאIOR תיארנו את החומרה החדשה - לכל בית של הוקטור x שנכנס ל-FA הושיפו שער xor כך שכאשר sub_cont שווה ל0 יבוצע חיבור וכאש הוא שווה ל1 יבוצע חיסור. (החיסור מתבצע על ידי שיטת המשלים ל2 כי כל הסיביות של X מתחולפות ואנחנו סוכמים $1=0c$).

כדי למשוך את neg השתמשנו גם בפעולות החיסור ובכך שהגדכנו ש $y=0$.

איך מימשנו את זה בקוד?

הגדכנו סיגנלים מקומיים לקלטים, כדי שנוכל להתאים אותם בהתאם לתת הפעולה שנשתמש בה, ולאחר מכן השתמשנו בfa עבור כל בית כך שנעבור על כל וקטור הקלטים המקומיים החדשים כדי לבצע את הפעולה הנדרשת.

סימולציה:

צרנו `test` עבור הפעולה הזו עם כמה קלטים כדי לבדוק האם היא עובדת.

+◆ /addersub_tb/x	11111111	00010100	00001100	00101100	00001100	00000000	11111111
+◆ /addersub_tb/y	00000001	00000001	00000000	00010010	01010010	00000000	00000001
+◆ /addersub_tb/sub_...	000	000	000	011	100	000	000
+◆ /addersub_tb/cout	1	001					
+◆ /addersub_tb/s	00000000	00001011	11110110	00010011	01010001	00000000	00000000

ניתן לראות חלק מהדוגמאות הבאות (משמאלי ימין):

1. חיבורנו בין 20 ל 3 וקיבלנו 23 עם `cout=0`

2. חיסרנו בין 2 ל 12 וקיבלנו בפלט 10.-.

3. פועלות `neg` ל 48 עם `cout=0`

6. חיבור בין 2 מספרים עם `cout=1`.

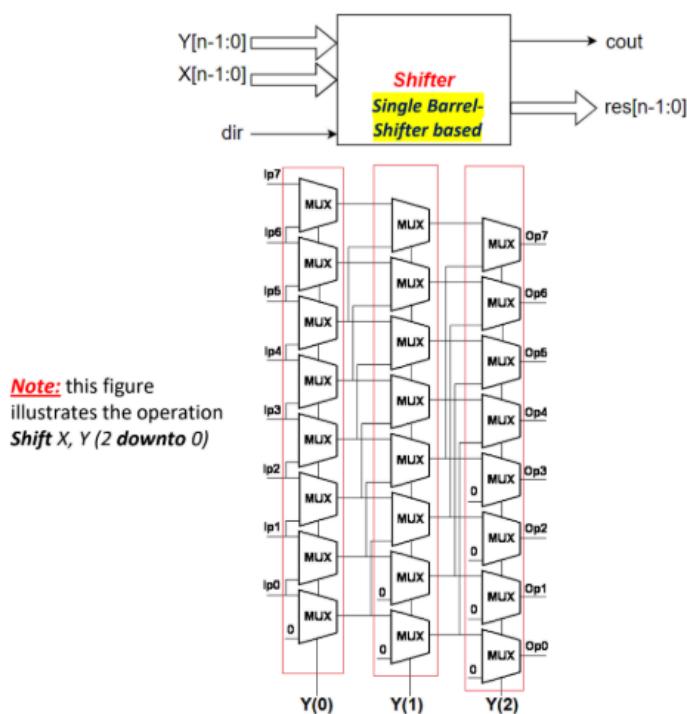
Shifter

הרכיב יכול לבצע הזזה ימינה או שמאלת של הקלט Z .

שלושת הביטים הראשונים של ALUFN ישרמו בסיגנל **direction** וזה מה שיגדר את הכוון. נגידר מטריצה שכלי שורה בה שומרת את הערך המוזז של Z . שורה 0 שומרת את הערך המקורי של Z כלומר ללא הזזה אם ההזזה הינה שמאלת, אחרת נשמר את Z הפוך (כי מבצעים הזזה ימינה, לאחר מכן נהפוך חזרה). כך בוצע לוולאה שבכל שלב תשמור את הערך המוזז של Z בהזחות שהם בחזקת השלב, ככלומר בשלב הראשון נשמר בשורה 1 של המטריצה את Z מוזז פעמי אחת או 0, בשורה השנייה הזחות של 2, בשורה 3 הזחות של 4 וכן הלאה. בנוסף בכל שלב נוספים אפסים משמאלי אם צריך בהתאם לשלב. הקלט X הוא זה שמנדר את מספר הזחות שנעשה בכל שלב ככלומר בית ה LSB קובע הזזה של 1, הביט השני הזחות של 2 וכן הלאה.

בסוף הלולאה בשורה האחורונה של המטריצה נקבל את הסופי המוזז של Z אך אם אנחנו מבצעים הזזה ימינה אנו הופכים את הוקטור ושמורים אותו בפלט של הרכיב.

בנוסף מבחינת ה HDOUT בכל שלב נשמר את הספרה האחורונה ש"יצאת" בתווך סיגナル (לדוגמא בשלב השני זו הספרה השנייה ש"יצאת", בשלב השלישי הספרה הרביעית ש"יצאת"), כך בלולאה עוברים שלב שלב על כל שורה של המטריצה ואם המבצעת הזזה באותו השלב אז נעדכן את ה HDOUT אחרת ללא שינוי. בסוף נוציא אותו כפלט.



סימולציה:

+◆ /shifter_tb/x	00000010	00000001	00000010	00000100	00000100	00000010
+◆ /shifter_tb/y	00100001	000001000	00010010	00011100	00100001	
+◆ /shifter_tb/direction	001	000	001	000	001	
+◆ /shifter_tb/cout	0					
+◆ /shifter_tb/res	00001000	00010000	00000100	11000000	00001000	

בדוגמה השמאלית הראשונה ניתן לראות שלקחנו את הוקטור 00001000 והזינו אותו במקום אחד שמאלה.

בדוגמה האחרונה מימין זהנו את 2 יביטים ימינה.

: Logic

מבצע פועלות לוגיות על X, Y :

X xor Y, X or Y, X and Y, X nor Y, X xor Y, X and Y, X or Y, X not Y, X AND Y, X OR Y, X NOT Y.

חלק מהפעולות הלוגיות לא קיימות בחבילה, לכן יצרנו אותן על ידי NOT, AND, OR, XOR.

סימולציה : לקחנו רוח של 40 ms לכל דוגמא

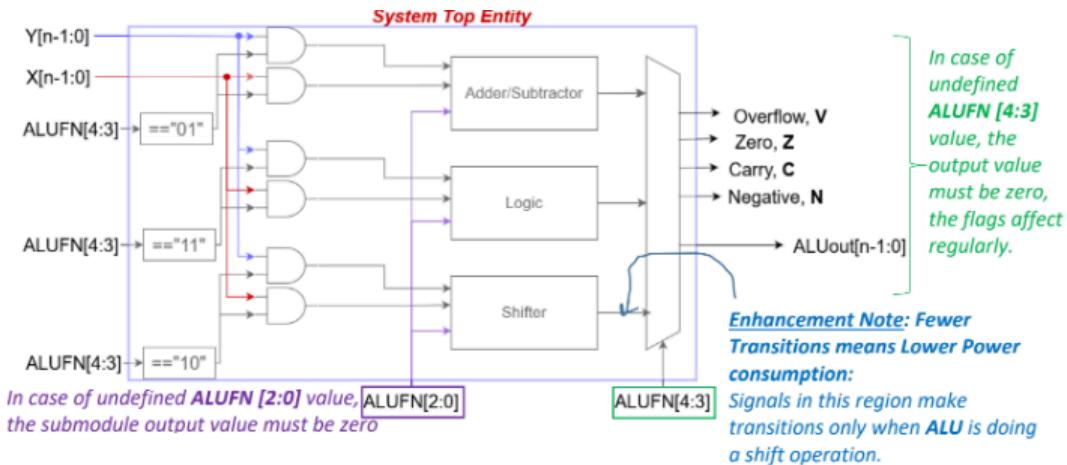
+◆ /logic_tb/x	00000000	000000100	00000010	00100100	00000011	11000100	00010100	11111111	00000000
+◆ /logic_tb/y	00000000	000000100	000000100	00001010	000000111	01000010	00000011	00000001	00000000
+◆ /logic_tb/res_logic	11111111	11111101	000000000	000001000	00111001	111111111	00000001	111111111	
+◆ /logic_tb/alufn_logic	000	000	000001100	011	100	101	111	000	

בדוגמה הרביעית משמאלי אנחנו מבצעים XOR בין $X=00000011$ ו- $Y=11111111$ ולפי טבלת האמת של xor אכן קיבלנו במצב 00001100

TOP

זו המערכת שעוטפת את כל הפעולות שהגדנו, היא מקבלת 2 קלטים Z, X , קלט שஅחראי לברור איזה תט מודול להשתמש. מוציאה בפלט דגלים ואת המוצא. נגידר שرك פועלה אחת תעבוד בהתאם לחץ $alufn$ בזמן שהשאר לא.

כל תט מודול מחזיר תוצאה, I cout, ה-Top בוחר מביניהם את התוצאה הסופית בהתאם לALUOUT. ובנוסף ה-Top מחשב ארבעה דגלים (Zero, Carry, Negativ, Cout) – על סמך התוצאה הסופית, cout של תט המודול שבחרנו והתנאים המתאימים לכל דgal.



סימולציה:

נريץ את הטעט : tb_ref1

	Msgs
+ /tb/Y	11011000
+ /tb/X	01111001
+ /tb/ALUFN	00100
+ /tb/ALUout	00000000
/tb/Nflag	0
/tb/Cflag	0
/tb/Zflag	1
/tb/Vflag	0

ואכן קיבלנו את התוצאות שציפנו להן.

ונסתכל גם המאגר `list`, ונשווה אותו למאגר שבסמישמה ונשים לב שם זהים מלבד הדלתאות (שהן ניתנות להזנהה). ימין - מהמשימה שמאל-פלט שלנו

ps delta	/tb/Y /tb/X /tb/ALUout /tb/ALUFN /tb/Nflag /tb/Cflag /tb/Zflag /tb/Vflag	ps delta	/tb/Y /tb/X /tb/ALUout /tb/ALUFN /tb/Nflag /tb/Cflag /tb/Zflag /tb/Vflag
0 +7	11111111 11111111 01000 11111110 1 1 0 0	0 +8	11111111 11111111 01000 11111110 1 1 0 0
50000 +8	11111110 1110101 01000 1110011 1 1 0 0	50000 +7	11111110 1110101 01000 1110011 1 1 0 0
100000 +7	11111101 1101011 01001 00010010 0 1 0 0	100000 +6	11111101 1101011 01001 00010010 0 1 0 0
150000 +8	11111100 11000001 01001 00010111 0 1 0 0	150000 +6	11111100 11000001 01001 00010111 0 1 0 0
200000 +8	11111101 11010111 01010 00101001 0 0 0 0	200000 +8	11111101 11010111 01010 00101001 0 0 0 0
250000 +6	111111010 11001101 01010 00101001 0 0 0 0	250000 +6	111111010 11001101 01010 00101001 0 0 0 0
300000 +7	111111011 10000011 01011 1111010 1 0 0 0	300000 +9	111111011 10000011 01011 1111010 1 0 0 0
350000 +7	11111100 1011001 01011 1111001 1 0 0 0	350000 +6	11111100 10111001 01011 1111001 1 0 0 0
400000 +8	111110111 10101111 01100 11110110 1 1 0 0	400000 +8	111110111 10101111 01100 11110110 1 1 0 0
450000 +7	11110110 1010101 01100 11110101 1 1 0 0	450000 +6	11110110 1010101 01100 11110101 1 1 0 0
500000 +10	11110101 10011011 01000 10010000 1 1 0 0	500000 +10	11110101 10011011 01000 10010000 1 1 0 0
550000 +8	11110100 10010001 01000 10000101 1 1 0 0	550000 +8	11110100 10010001 01000 10000101 1 1 0 0
600000 +8	11110011 00000000 01001 01101100 1 0 0 0	600000 +7	11110011 00000000 01001 01101100 1 0 0 0
650000 +9	11110010 0111101 01001 0111010 0 1 0 1	650000 +9	11110010 0111101 01001 0111010 0 1 0 1
700000 +7	11110001 01100011 01001 00000000 0 0 1 0	700000 +7	11110001 01100011 01001 00000000 0 0 1 0
750000 +1	11110000 01010001 01111 00000000 0 0 1 0	750000 +1	11110000 01010001 01111 00000000 0 0 1 0
800000 +9	11110111 01011111 10000 10000000 1 1 0 0	800000 +9	11110111 01011111 10000 10000000 1 1 0 0
850000 +9	11101110 01010101 10000 11000000 1 0 0 0	850000 +7	11101110 01010101 10000 11000000 1 0 0 0
900000 +9	11101101 01001111 10000 00011101 0 1 0 0	900000 +8	11101101 01001111 10000 00011101 0 1 0 0
950000 +8	11101100 01000001 01001 0110110 0 0 0 0	950000 +8	11101100 01000001 01001 0110110 0 0 0 0
1000000 +5	11101011 00110111 10010 00000000 0 0 1 0	1000000 +5	11101011 00110111 10010 00000000 0 0 1 0
1050000 +1	11101010 00101101 10010 00000000 0 0 1 0	1050000 +1	11101010 00101101 10010 00000000 0 0 1 0
1100000 +9	11101001 00100011 10000 01000000 0 1 0 0	1100000 +8	11101001 00100011 10000 01000000 0 1 0 0
1150000 +9	11101000 00011001 10000 11010000 1 1 0 0	1150000 +7	11101000 00011001 10000 11010000 1 1 0 0
1200000 +9	11100111 00001111 10001 00000001 0 1 0 0	1200000 +9	11100111 00001111 10001 00000001 0 1 0 0
1250000 +7	11100110 00000101 10001 00000111 0 0 0 0	1250000 +1	11100110 00000101 10001 00000111 0 0 0 0
1300000 +5	11100101 11111011 10010 00000000 0 0 1 0	1300000 +9	11100101 11111011 10010 00000000 0 0 1 0
1350000 +1	11100100 11100001 10010 00000000 0 0 1 0	1350000 +5	11100100 11100001 10010 00000000 0 0 1 0
1400000 +6	11100011 11000111 10011 11000111 1 0 0 0	1400000 +5	11100011 11000111 10011 11000111 1 0 0 0
1450000 +6	11100010 1101101 10011 11111111 1 0 0 0	1450000 +5	11100010 1101101 10011 11111111 1 0 0 0
1500000 +6	11100001 11010011 10010 11000001 1 0 0 0	1500000 +5	11100001 11010011 10010 11000001 1 0 0 0
1550000 +6	11100000 11001001 10010 11000000 1 0 0 0	1550000 +5	11100000 11001001 10010 11000000 1 0 0 0
1600000 +6	11011111 10111111 11101 01100000 0 0 0 0	1600000 +5	11011111 10111111 11101 01100000 0 0 0 0
1650000 +6	11011110 10110101 11101 01101011 0 0 0 0	1650000 +5	11011110 10110101 11101 01101011 0 0 0 0
1700000 +6	11011101 10101011 11111 10001001 1 0 0 0	1700000 +5	11011101 10101011 11111 10001001 1 0 0 0
1750000 +6	11011100 10100001 11111 10000010 1 0 0 0	1750000 +5	11011100 10100001 11111 10000010 1 0 0 0
1800000 +6	11011011 10101111 11011 01001100 0 0 0 0	1800000 +5	11011011 10101111 11011 01001100 0 0 0 0
1850000 +6	11011010 10001101 11011 01010111 0 0 0 0	1850000 +3	11011010 10001101 11011 01010111 0 0 0 0
1900000 +3	11011001 10000011 00100 00000000 0 0 1 0	1900000 +1	11011000 01111001 00100 00000000 0 0 1 0
		2000000 +1	+11010111 00101111 00100 00000000 0 0 1 0

1950000 +1|11011000 01111001 00100 00000000 0 0 1 0
2000000 +1|11010111 01101111 00100 00000000 0 0 1 0