

# מבנה המחשב | 67200

הרצאות | אוהד פאליד ורונ גבר

כתביה | נמרוד רק

תשפ"ג סמסטר ב'

## תוכן העניינים

4	I מבוא ומוליכים-למחאה
4	הרצאה . . . . .
4	רקע כימי לרנזיסיטורים . . . . .
5	רקע פיזיקלי לטרנזיסיטורים . . . . .
6	מוליכים למחאה . . . . .
7	צומת ח-ק . . . . .
8	MOS-FET . . . . .
9	בנייה של שערים מטרנזיסיטורים . . . . .
10	תרגול . . . . .
11	II שערים
11	הרצאה . . . . .
12	סכום מכפלות ומכפלת סכומים . . . . .
14	יחידות סטנדרטיות בمعالגים לוגיים . . . . .
15	معالגים סדרתיים . . . . .
17	معالגים סדרתיים מורכבים על בסיס SR-Latch . . . . .
18	DFF . . . . .
18	תרגול . . . . .
20	מפות קרנו . . . . .
24	פונקציות שלמות . . . . .
24	III
24	הרצאה . . . . .
27	FSM . . . . .
30	זמן מעבד והגדירות זמני פעולה . . . . .
34	תרגול . . . . .
37	MIPS IV
37	הרצאה . . . . .
38	RISC vs CISC . . . . .
38	ריגיסטרים-ב-MIPS . . . . .
39	פקודות אריתמטיות . . . . .
39	פעולות לוגיות . . . . .
40	פעולות זכרון . . . . .
41	פילוסופיות ארגון זכרון . . . . .
41	פקודות קפיצה והתניות . . . . .
42	שימוש פונקציות באסבלי . . . . .
43	פקודות קראיה לפונקציה . . . . .
44	תרגול . . . . .
44	אנליזה של מעגלים סינכרוניים . . . . .
47	синטזה של מעגלים סינכרוניים . . . . .

## מעבד V Single-Cycle

49	הרצאה
49	קידוד פקודות MIPS
50	ביצוע פקודות ב-MIPS
50	IMPLEMENTATION OF MIPS INSTRUCTIONS
51	שימוש השלבים ב-MIPS
54	תרגול

# שבוע II | מבוא ומוליכים-למחצה

## הרצאה

המחשבים הראשונים היו עצומים, כבדים ויקרות, ויחידת הבסיס של המעבד שלහן היה שפופורת קטודיות (אבי הטרנזיסטור) ואז שפופורת ריק, וכיום משתמשים בטכנולוגיית CMOS שמאפשרת גדילה בעשרות סדרי גודל בזמן המוחזר, מהירות השעון, מספר הטרנזיסטורים ועוד. הקורס עוסק במבנה המעבד בעיקר.

בתוך כל מחשב יש אביזרי קלט ופלט (חישוני סונאר, מסך, עכבר), אמצעי אחסון (נדף כMO RAM ולא נדי' כמו דיסק קשיח), מעבד, מערכת הפעלה, דרייברים ותוכנה. בקורס עוסק במעבדים ותוכנה ונזכיר מערכות הפעלה ואמצעי אחסון.

קצב ההתקדמות עד לשנים האחרונות התנהג לפי חוק מור (1965) - כל שנה مضליים להכפיל את מספר הטרנזיסטורים כל שנתיים. העליה במספר הטרנזיסטורים מספקת גם עלייה אקספ' בביטויים, ביעילות אנרגיה וגם בגודל האחסון שאפשר לייצר.

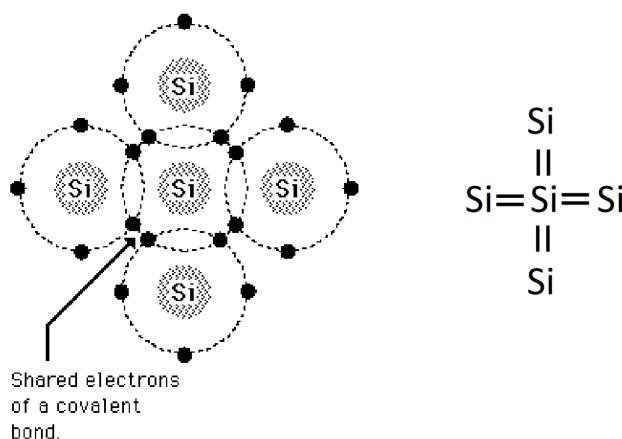
## רקע כימי לטרנזיסטורים

המודל של בוחר ניסה להסביר את תופעת התכוונות המשותפות ליסודות באוטה עמודה של הטלחה המחזוריית אותה בנה מנדليب כמה עשרות שנים קודם לכן. המודל קובע שככל חומר מורכב מאטומים, שלהם יש גרעין עם פרוטונים (חלקיים עם מטען חיובי חיובי), ניטרונים (חלקיים ללא מטען) ואלקטרונים (עם מטען שלילי) ששובבים את הגרעין.

**דוגמה** סיליקון (צורן) הוא מספר 14 בטבלה המחזוריית, כלומר יש לו 14 פרוטונים (ובמקרה זה גם 14 ניטרונים) וכן אלקטرونים בשכבות שונות סביבה הגרעין עם מספר שונה של אלקטرونים בכל שכבה.

בזה גילה בוסף שהמסלול (המעגל, השכבה של אלקטرونים) האחרון של כל אטום במצב יציב הוא מלא, אך אטום ירצה לחת או לחת אלקטرونים מאטומים אחרים כדי לקבל מסלול מלא.

**דוגמה** הרבה אטומים של סיליקון יוצרים יחד במצב יציב שmorכב משציג אטומי סיליקון עם מסלול אחד מלא לכולם כך שהם חולקים אלקטرونים (ראו איור במקורה של חמישה אטומים)



קשר בו אטומים חולקים (sharing) אלקטرونים נקרא קשר קוולנטי (covalent bond).

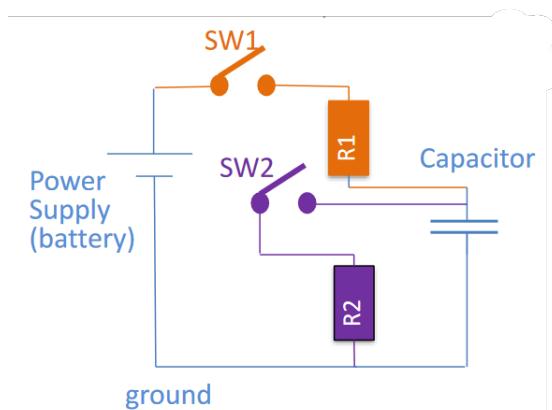
יונ הוא אטום או מולוקולה עם מספר לא שווה של פרוטונים (+) ואלקטרונים (-) והמטען של החלקיק הוא הפרש הערכיים האלו.

קשר יוני הוא קשר בין אטומים שנוצר כאשר אחד מהמסלול האחרון של אטום אחר (כדי להשלים מסלול) אבל מספר הפרוטונים באטומים שווה למספר האלקטרונים בכל אטום למעבר האלקטרון. כך לשניים כתה יש מטען מנוגד לשני ומושם ניגודיות המטענים מקרבת (באמצעות כוח משיכה אלקטرون-סטטי) בין האטומים לכדי קשר.

## רקע פיזיקלי לטרנזיסטורים

- זרם חשמלי הוא תנועה של אלקטرونים (או באנלוגיה תנועה של חורים, כאשר האלקטרונים השיליליים עוברים בין חורים אחרים חיוביים). בקונבנצייה הזרם נע מה-+-.
- מתח (פוטנציאל) נמדד בולט והוא יכולת להזור, כאשר סוללה עצם מחזיקה מתח וכמו מגדל מים באנלוגיה מים, אפשר לעשות בו חור וכך לגרום לו זורמה אבל כל עוד לא מחברים/מחוררים את הכליל לא יקרה שום דבר.
- מטרון הוא מספר האלקטרונים שמאוחסנים בחומר כלשהו.
- קיבול זו היכולת להחזיק מטען, כאשר בעת אחסון מטען נוצר מתח בקבל (מקביל למיכל מים ולחץ).

**דוגמא** נביט בمعالג הבא. R1 הוא נגד (סקול לצינור צר יותר, שיוצר התנגדות). אם נסגור את המתג הראשון, נסגור מעגל בין הסוללה לקבל כך שאלקטרונים יזרמו מהсолלה לקבל מלמעלה ומהקבל לסלולה מלמטה והקבל מקבל את אותו המתח של הסוללה. אם ננטק את המתג הקבל ימשיך להחזיק את המתח, ואם נדליך את המתג השני יהיה לנו זרם קצר מהקבל אליו עם כיוון השעון (בחילקו העליון של הקבל היונים החיוביים ומתחתיו השיליליים, והאלקטرونים הם אלו שענים).



בתקשר של מחשבים נקבע מתח נמוך (עד 1.5V) להיות 0 ומתוך גבוה (פחות 3.5V) להיות 1 - "כִּי כָּה". בין 1.5 ל-3.5 וולט לא אמורים להיות במצב יציב, רק במעבר בין המצבים. האנלוגיה כאן היא האם מיכל מים הוא מלא או ריק.

## מוליצים למחצה

מוליך למחצה הוא חומר שלא מוליך חשמל מאד טוב (מוליכותו היא בין חומר לא מוליך לחומר מוליך).

**דוגמא** סיליקון טהור הוא מוליך למחצה כי בצורת הגביש עלייה דיברנו יש מעט מאוד אלקטرونים חופשיים (הרבה מהם תפוזים בין כמה אטומים בקשר קוולנטי) ולכן קשה להזיר זרם.

איילוח (doping) הוא תהליך שבו "מלכלכים" את הסיליקון.

- **P-type :** נוסיף לסיליקון קצת אלומיניום (לו 3 אלקטرونים מיותר 4 במסלול האחרון) כך שייקשר לאטומי הסיליקון ועתה לחומר יהיה חסר אלקטרון והוא ישmach לקבל אותו, ואז אלקטرونים יעבו בклות בין האי-שלים האלה (פעם ישלים את המחשור במוקד כלולוך אחד, ואז יקפוץ לאחר, וכו'). החוררים שנוצרים כשאין את האלקטרון הנדרש ליציבות נעים (לשם התיאוריה), בכיוון ההפוך מהאלקטرونים וכן נוצר זרם חיובי בכיוון ההפוך מתנועת האלקטרונים.

- **N-type :** העיקרון הנ"ל רק עם זרחן (לו 5 אלקטرونים כלומר 1 במסלול האחרון) כך שייצור עודף אלקטرونים והאלקטرونים העודפים חופשיים לנوع לאן שירצטו ויצרו זרם.

لمוליך למחצה מסוג P ו-N אין מטען חיובי או שלילי אלא רק סיבובות שונות לתנועת האלקטרונים כתוצאה מהרצון להשלים מסלולים אחרים בתאומים.

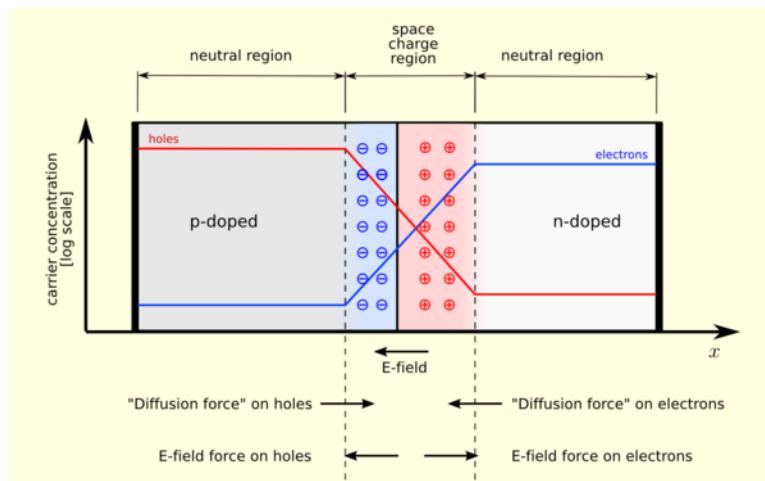
הצמדה של חומרים מסוג P ו-N יוצרת יונים - האלקטרונים מ-N שמחים לקפוץ ל-P שם "צרי" אותם. שני כוחות פועלים בעת הצמדת חומרים שכזו:

- הרצון של המסלולים להתמלא - מה שגורם לאלקטרונים לקפוץ מ-N ל-P.
- הכוח החשמלי שיוצרים האלקטרונים - כוח דחיה בין מטען שליליים שמונע קפיצה אלקטרונים מ-P ל-N (N אומר "יש לי מספק שליליים כבר").

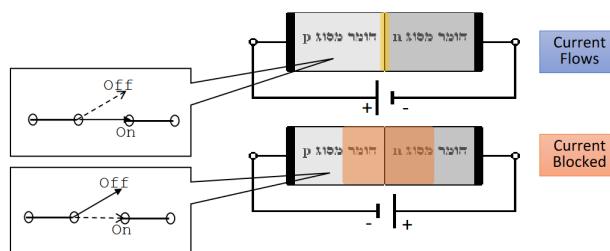
## צומת PN

הצמדה זו נקרא צומת PN junction (PN junction) והוא מוליכה מכיוון אחד וחוסמת זרם מכיוון אחר (כמו שסתום!). המנגנון שהצומה מספק נקרא דיודה (diode) ומסומן באירום הבאים

בשל תזוזה מקרית של אלקטرونים ופロטוניים בסמוך לצומת, מצליכים לעבר אטומים יוניים שליליים מ-N ל-P וחובבים להפוך (בניגוד לכיוון הזרימה). כשהמספיק התחלפותו כליה קורוט, ישנה מסה של אטומים יוניים חיוביים ב-N ומסה של שליליים ב-P שלא עברו לצד השני בගליהם חלק מהגביש כבר. במצב זה נוצר מחסום מכוח כוח הדחיה החשמלי שגורם להפסקת הזרימה דרך הצומה והחזקת מתח בו (ראו איור)



עתה חיבור סוללה לדiode נותן לנו תכונות מעניינות.



- חיבור סוללה עם + ל-P ו- ל-N יגרום להרבה מאד יונים חיוביים לזרום מ-P ל-N ויוניים שליליים מ-N ל-P (בהתאם לכיוון הזרימה לפני שנחנכה) וכן יצטמצם המרווח באמצעות עד ל-0 ותהייה לנו זרימה רגילה, כאילו סגרנו מתג.

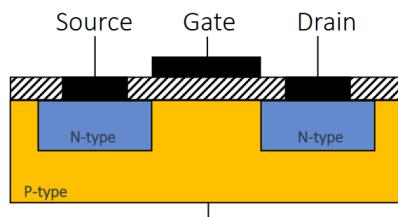
- חיבור סוללה עם  $+L-N$  ו- $-L-P$  יגרום ליוניים חיוביים ושליליים להגיע לחומרים ולחזק את היוניים במרוחה שכרגע מונעים מעבר ורק יחוקו את החסימה, כך שלמעשה דימינו התנאות של פתיחת מתג.

## MOS-FET

טרנזיסטור MOS-FET עשוי שלושה חומרים: מוליך למחצה; תחמושת מבודדת; ומתקת. יש לו בנוסף שלוש רגליים: source ; drain ; gate (ורgel הארקה שמחוברת תמיד).

### N-Channel ה-ORIANT

באיור ניתן לראות NMOS, וריאנט מבין שניים של MOS-FET (השני משלים לו רק ש-N ו-P במקומות הפוכים). החומר המוקוק הוא המבודד והשחור הוא המתקת.

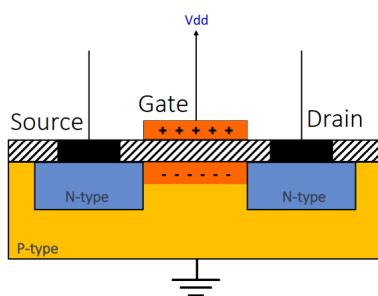


זכור כי זרימה ניתן לקבל רק מ-P ל-N וכן אי אפשר אף פעם להזירם שום דבר מהמקור (source) לשפך (drain) ולהפך. יש לנו למעשה שתי דיזודות גב אל גב (שकצוותיהן המקור והשפך).

אם המתקת של השער מקבלת מטען, האזורה בין השער למוליך-למחצה נהייה קובל כי הוא מחזיק הפרשי מטען!

**הערה** את כל ארבעת הרגליים תמיד לחבר לאנשהו - או לאדמה או לסוללה או לטרנזיסטור אחר.

- אם לחבר את השער לאדמה (הארקה), יהיו לנו שני צמחיי-VG שלא ניתן יהיה להזירם דרךן (ובפרט בין S ל-D דבר).
- אם לחבר את השער למתח, הקובל שהזכרנו לעיל מקבל מתח ועכשו יש מטען חיובי מעליו ושלילי מתחתיו, ככלומר ישנים אלקטرونים חופשיים ב-p-type, וכשהלו נוגעים בחומר n-type משני הצדדים יוצרץ אלקטرونים חופשיים דרךן יכול לעבור זרם, ומכאן השם n-channel.



מה שקיבלו בסופו של דבר הוא סוויצ' שאנחנו יכולים לשולוט בו באמצעות זרם לשער (0 או 1). את הטרנזיסטור נסמן בסימול הבא -

P-channel עובד אותו הדבר רק הפוך - הזרמת "0" לשער תסגור את המתג ו-"1" תפתח אותו. ההבדל העיקרי היחיד חוץ מהחומרים הוא שיש מתח שמחובר מלמטה במקום הארץקה. להלן טבלת סיכום של דרך הפעולה של הטרנזיסטורים.

Gate	"0"	"1"
N-MOS		
P-MOS		

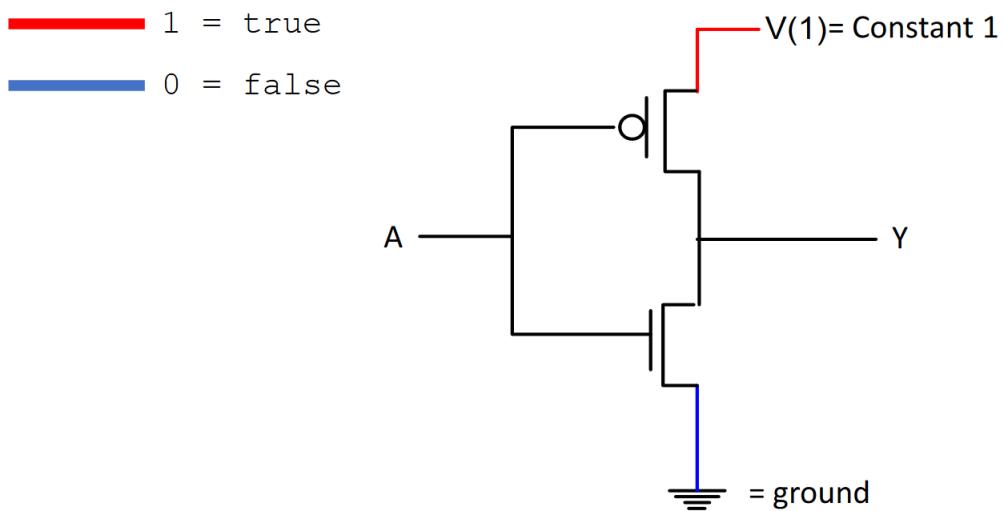
הערה בתחום הכללי, מסמן שליליה, ב-PMOS למשל "שוללים" את המתח ומתחנגים הפוך ממנו. גם שערי NOT מסומנים עם עיגול.

## בנייה של ריצוף מטרנזיסטוריים

טרנזיסטורים הם שימושיים לנו כי אנחנו יכולים לבנות מהם ריצוף לוגיים.

דוגמה: בניית ריצוף NOT (Inverter) שמסומן ב-

הבנייה דורשת שני NMOS-ים (למעלה ו-N-Channel) למטה, ו-HMOS-ים (למטה), והוא כבאיור.



זהו הקלט ו-Y זה הפלט. כאשר  $A = 1$ , יעבור זרם של 0 (שהוא זניח) דרך ה-N-Channel למטה (השער דלוק) ולא יזרום שום

דבר דרך ה-P-Channel למטה (השער דלוק שכן אין זרימה). סה"כ אין שום זרימה עם מתח לא זניח ולכן  $Y$  יהיה "0".

בהתאם אם  $A$  הוא "0" אז ה-PMOS יזרום דרכו זרם קבוע של "1" ו- $Y$  יהיה "1".

הערה: צריך את ה-NMOS התחתון כי אחרת  $-A = 1$  יש לנו מעגל פתוח שיכול להיות לו כל זרם ולא בהכרח "0" כמו שאנחנו רוצים.

## תרגול

בבסיס ספירה הוא דרך ליצוג מספרים ממשיים. בסיס עשרוני נבע את ההמורה הבאה

$$(a_4a_3a_2a_1a_0.a_{-1}a_{-2}a_{-3})_{10} = a_410^4 + \dots a_010^0 + a_{-1}10^{-1} + a_{-2}10^{-2} + a_{-3}10^{-3}$$

בסיס בינארי משתמש בסיביות (ביטים) שערן 0 או 1, בעשרוני 0 עד 9, ובקסדצימלי  $A - F$ .

טוח המספרים בעל  $n$  ספרות בסיס  $r$  הוא  $\{0, \dots, r^n - 1\}$

$$\text{במקורה הכללי } a_n \dots a_0.a_{-1} \dots a_m \text{ בסיס } r \text{ מייצג את המספר} \\ \sum_{i=-n}^m a_i r^i$$

פעולות חשבוניות אפשר לעשות באותו האופן כבסיס עשרוני (חיבור ארוך שבו עברת ספרה הלאה, כאשר הספרות נגמרות לא בהכרח אחריו).  
(9)

מעבר לבסיס 10 הוא די פשוט (פרישת הייצוג וחישוב מכפלות וחיבור בסיס עשרוני). מעבר מבסיס 10 לבסיס אחר הוא פשוט תהליך איטרטיבי של פעולות מודולו ( $r$  הבסיס) כאשר מה שהוא לא השארית יהיה הספרה הראשונה (משמאלו), השניה, וכו' עד שנגמרות הספרות.

לא כלתי כאן אינסוף דוגמאות להמורות בין בסיסים כי לא מספיק משעמים לי.

במקורה הכללי נמיר מני בסיסים כלשהם דרך בסיס 10 כאשר את הרכיב משמאלו ומימין למספר העשרונית נטפל באופן נפרד וזהה (עד כדי חלוקה איטרטיבית בשמאלו ומכפלה איטרטיבית בימין).

**דוגמה** נמיר את  $_{10}(0.79272)$  לבסיס 4. נכפיל את המספר ב-4 ונקבל 3.17088. لكن הספרה הראשונה היא 3 והשארית היא 0.17088. נבעז זאת שוב ועתה נקבל 0 ושארית 0.68352, וחזור חיליה עד שהשארית תהיה 0, כאשר עתה הספרות הנו מלמעלה למטה במקום למטה מלמעלה בספרות הרגליות.

## שיטות ליצוג מספרים

- **שיטה גודל וסימן**: מספר יתחל בביט סימן (0 חיובי 1 שלילי) ושאר הביטים יהיו ייצוג בינארי של המספר.

$$\text{דוגמה } 01001 = (-1)^0 (2^3 + 2^0)$$

טוח היצוג בשיטה זו הוא  $(2^{n-1} - 1), \dots, 0, \dots, (2^{n-1} - 1)$

• **שיטת המשלים לאחד**: בית סימן, שלפי ערכו נדע אם שאר הספרות הן הערך הבינארי של המספר וזהו שזהו הערך הבינארי של המשלים של המספר  $-1 - (2^{n-1} - 1)$ , ובנוסף הפיכת כל בית תספק לנו את השילילה של המספר. לדוגמה,  $4 = 00100$ , ואם נהפוך כל בית נקבל  $-4 = 11011$ .

חישור מספרים הוא די פשוט: צריך לחבר את המספרים, ואם יש carry לאחר החישוב נמחק אותו ונוסיף אחד לתוצאה.

$$\text{דוגמה } 00101 - 4 = 9 - 4 = 01001 + 11011 = 100100 \text{ זהה הופך ל-00101.}$$

טוווח הייצוג הוא כמו בשיטת גודל וסימן ויש בו, כמו בשיטה הקודמת 0 חיובי ו-0 שלילי.

- **שיטת המשלים לשתיים:** בית סימן, שעבור מספרים שליליים ערכו הנדי למספר שਮתקבל מהיפוך הביטים והוספה אחד.

$$\text{דוגמה } -4 = -(00011 + 1) = -00100 = 11100.$$

לחולופין לחישוב המשלים אפשר ללקת מימין לשמאל עד ל-1 הראשון, לא לשנות אותו, ואז להפוך את כל מה שמשמאלו (ואז לא צריך להוציאי).<sup>1)</sup>

**דוגמה** למה שווה 5 – בשיטת המשלים ל-2?  $00101 = 5$ , נוסיף אחד ונשנה את בית הסימן ונקבל 10110.

כדי לחסר מספרים, נסכום אותם ונמתק **carry** אם יש.

**הגדירה** overflow הוא מצב שבו אנחנו סוכמים שני מספרים באותו הסימן ומקבלים מספר בסימן הפוך, במקרה זה התוצאה כMOVן שגוייה.

**דוגמה** משתמש בשיטת המשלים ל-1 עם 5 ביטים,  $11000 + 01101 = 01011$  קלומר סכמנו חיוביים וקיבלו תוצאה שלילית!

**הערה** הפתרון ל-overflow הוא הוספת ביטים כך שיידל טווח הייצוג.

## שבוע III | שערים

### הרצאה

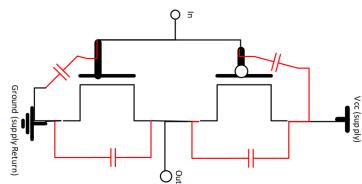
**הערה** כל שער שנבנה נבנה סימטרית מבחינה השימוש ב-NMOS ו-PMOS, אך השערים עם טרנזיסטורים אלה נקראים על-שם נקודות.

.MOS

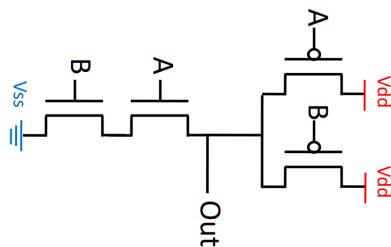
**הערה** לעולם לא נרצה להזרים זרם אל תוך טרנזיסטור אחר דלוק (מהכיון הלא נכון) כי זה גורם לסתור.

שער אחד בפני עצמו זה נחמד, אבל מעבדים בונים ע"י חיבור שערים. את השערים נחבר עם חומר מוליך בין הקלט של שער אחד לפט של השער שמןנו אנחנו לוקחים את התוצאה.

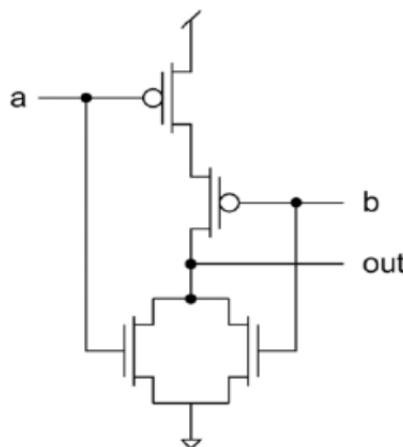
בין שער למקור, בין מקור לשפק ובין שער לשפק נוצרים קבליים אפקטיביים (לא הוספנו שום דבר, פשוט יש רכיבים פיזיים שנמצאים עצמם מחזיקים מתח בין הצדדים). את הקבליים האלה לוקחים זמן לטעון או לפרק מתחה בעת שנייה מצב (שינויי הזרם מהשער, הזרם מהמקור). לכן במצבים רגילים יש זרימה שאנו לא בהכרח מצלפים לה במהלך המעבר (10–15 שניות). ראו באירור באדום את הקבליים שנוצרו.



השער היסודי שמאפשר לבנות את כל שער השערים הוא NAND (Not And) - שנותן 0 אם "ס שני הפלטים 1 (ואחרת 1), ומסומן כך:



בדומה ניתן לבנות שער NOR באופן הבא (קו אלכטוני הוא (1) V כלומר מתח קבוע דלוק וחץ הוא (0) GND כלומר האركה)



הגדרה פ' בוליאנית היא פ' שמקבלת קלטים בוליאניים (משתנה שיכול לקבל אחד מבין שני ערכים, ביטים לדוגמה) ופולטת פלט בוליאני אחד בדיק.

הערה מעתה נסמן ' $x'$  להיות ההיפוך ל- $x$  (כלומר שהפעלו עליו שער NOT).

דוגמא פ' שבהינתן  $y, x$ , פולטת פלט שערכו  $x$  וגם לא  $y$ .

## סכום מכפלות ומכפלת סכומיים

הגדרה בהינתן משתנים בוליאניים לפ' בוליאנית כלשהי, minterm הוא מכפלה (AND) של ליטרלים, כאשר ליטרל הוא משתנה או היפוכו וכל משתנה מופיע בדיק פעם אחת כליטרל או בתוך ליטרל מהופך.

דוגמא עבור שלושה משתנים וההשמה  $x = 1, y = 1, z = 0$ ,  $m_6 = xyz'$  יהיה 1, והוא המינטרם היחיד שערכו 1, ועבור 1 זה יהיה  $m_7 = xyz$  (אינדקס ה-minterm עם ערך 1 מתקבל ע"י הערך הדצימלי של הסטריאג הבינארי המתקבל מהצמדת ההשומות במשתנים).

נשים לב שש-minterm מקבל ערך 1 בבדיקה שורה אחת של טבלת אמת מלאה על המשתנים.

**הגדירה** בהינתן השמה במשתנים, הוא סכום (OR) של המשתנים או היפוכיהם כך שכל משתנה מופיע פעם אחת בדיק.

**דוגמה** עבור שלושה משתנים עם ההשמה  $x = 1, y = 1, z = 0$  הוא  $M_6 = x' + y' + z$  וכו'.

**הערה**  $m_i$  משלים ל- $M_i$ .

**הגדירה** Sum of Products הוא סכום מכפלות minterm-ים ו-maxterm-ים.

**דוגמה** הפ'  $F_1$  עם טבלת האמת הבאה

x	y	z	$F_1$	Minterm	Maxterm
0	0	0	0	$m_0 = x'y'z'$	$M_0 = x + y + z$
0	0	1	1	$m_1 = x'y'z$	$M_1 = x + y + z'$
0	1	0	0	$m_2 = x'yz'$	$M_2 = x + y' + z$
0	1	1	1	$m_3 = x'yz$	$M_3 = x + y' + z'$
1	0	0	1	$m_4 = xy'z'$	$M_4 = x' + y + z$
1	0	1	0	$m_5 = xy'z$	$M_5 = x' + y + z'$
1	1	0	1	$m_6 = xyz'$	$M_6 = x' + y' + z$
1	1	1	0	$m_7 = xyz$	$M_7 = x' + y' + z'$

ניתנת לייצור ע"י

$$F_1(x, y, z) = m_1 + m_3 + m_4 + m_6 = x'y'z + x'yz + xy'z' + xyz'$$

הטכנית היהתה לסכום את כל המינט-ים שמקבילים 1 בפ' (בכחול).

חלופין נוכל לייצג את הפ' עם PoS ע"י המכפלת כל המקסט-ים שמקבילים ערך 0 (באדום) כולם

$$F_1(x, y, z) = M_0 \cdot M_2 \cdot M_5 \cdot M_7$$

כלומר הצנו בצורה קנונית פ' לכאהורה מורכבת!

**הערה** למה משתמשים כל הייצוגים השונים (טבלת אמת, PoP, SoP ומפת קרנו שנלמד בתרגול)? נרצה בסופו של דבר את הייצוג המינימלי, כך שנדרש למספר הקטן ביותר של שערים כדי למשמש אותו.

**דוגמה** מולטייפלסקר (MUX) מקבל שלושה קלטיים :  $S(elect), D0, D1$ . בטבלה  $X$  משמעו "לא משנה"

"מה הערך"

S	D1	D0	Y
0	X	0	0
0	X	1	1
1	0	X	0
1	1	X	1

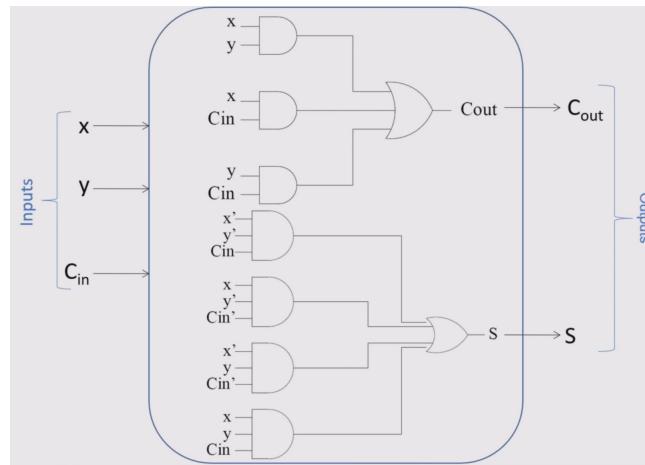
וניתן לרשום את הפ' כ- $C' D_0 + S' D_1$ , ואת זה ניתן למשב באמצעות שערים שכבר בנוינו. עם זאת, ניתן למשב את המעגל עם פחות טרנזיסטורים מאשר בIMPLEMENTATION אובייקטIVE-USING NOT-AND-OR.

**דוגמה** הוא מעגל שמקבל  $x, y, C_{in}$  (כאשר  $S$  נשא מסכימה קודמת) ופולט  $C_{out}$  כאשר  $S$  הוא הסכום ו- $C_{out}$  הוא הנשא (ראו טבלת אמת overflow)

Truth Table					
x	y	$C_{in}$	S	$C_{out}$	
0	0	0	0	0	
0	0	1	1	0	
0	1	0	1	0	
0	1	1	0	1	
1	0	0	1	0	
1	0	1	0	1	
1	1	0	0	1	
1	1	1	1	1	

נוקח את  $S = 0, C_{out} = 1$   $x + y + C_{in} = 0 + 1 + 0 = (10)_2$ , או  $x = 0, y = 1, C_{in} = 1$

למעשה, בגלל שיש למעגל שני פלטים, הרי שהוא מורכב משתי פ' בוליאניות. קרגיל אפשר למשב את המעגל באמצעות SoP (סכימת minterm), ובIMPLEMENTATION אובייקטIVE-USING NOT-AND-OR מופיע במעגל OR עם שלושה וארבעה כניסה - הסתודנטית המשקיעה תראה כיצד ניתן לעשות זאת עם פי 2 טרנזיסטורים ממספר הכניסות).



**הערה** קל לשדרר כמה FA-ים כדי לקבל מעגל שישוכם מספרים עם מספר ביטים גדול יותר (לוקחים את  $C_{out}$  של ה-FA על זוג הביטים הראשונים, מכנים אותו ל-FA וחוזר חלילה).

## פתרונות סטנדרטיות במעגלים לוגיים

1. מפענח (Decoder) : הקלט הוא  $n$  ביטים  $x_0, \dots, x_{n-1}$  והפלט הוא  $k = 2^n$  כאשר  $d_j = 1$

$$d_j = \begin{cases} 1 & (j)_{10} = (x_{n-1} \dots x_0)_2 \\ 0 & \text{אחרת} \end{cases}$$

כלומר פורש וקטור של  $n$  ביטים על  $2^n$  ביטים שכל אחד מייצג מספר מותך  $\{0, \dots, 2^n - 1\}$ .

ברגע שיש לנו בлок של מפענח, אפשר להשתמש בו כדי למשוך פ' בוליאנית באופן טריוויאלי, כי כל מה שצריך לעשות זה לעשות OR על כל ה- $d_i$ -ים שמייצגים  $x_0, \dots, x_{n-1}$  שמקבל ערך 1 בטבלת האמת של הפ' הבוליאנית.

2. מפלג (DeMultiplexer) : הקלט הוא  $f_0, \dots, f_{k-1}$  כלאחד בגודל בית והפלט הוא  $s_0, \dots, s_{n-1}$  כאשר  $k = 2^n$

$$f_j = \begin{cases} x & (j)_{10} = (s_{n-1} \dots s_0)_2 \\ 0 & \text{אחרת} \end{cases}$$

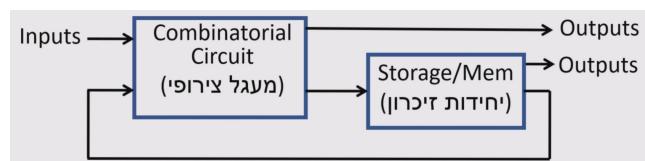
כלומר בהינתן בית, מוחזיר מהרזהות שבה הכל אפסים חוץ מאולי הבית ה- $(s_{n-1} \dots s_0)_2$  שערכו  $x$ .

3. מקודד (Encoder) : הקלט הוא  $n$  ביטים  $x_0, \dots, x_{k-1}$  והפלט הוא  $k = 2^n$  כאשר  $e_i = 1$  עבור  $i$  אחר בדיקוק (וכל השאר אפסים) או  $e_{n-1} \dots e_0 = (i)_{10}$ , כלומר מוחזיר את האינדקס (בביניاري) של הבית היחיד הדлок בקלט.

4. מריבב (Multiplexer) : הקלט הוא  $k = 2^n$  ביטים  $s_0, \dots, s_{n-1}$  וביטים  $x_0, \dots, x_{k-1}$  והפלט הוא  $f$  שהוא ערך הבית עם אינדקס  $.x \cdot (s_{n-1} \dots s_0)_2$

## מעגלים סדרתיים

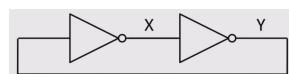
**הגדרה** מעגל סדרתי הוא מעגל קומבינטוררי שחלק מהקלטים שלו הם פלטים של ייחידת זיכרון שמחזיק השער (ראו איור)



מעגלים סדרתיים אסינכראוניים יכולים לשנות מצב בכל זמן, ואילו מעגלים סינכראוניים משנים מצב בהתאם לשעון.

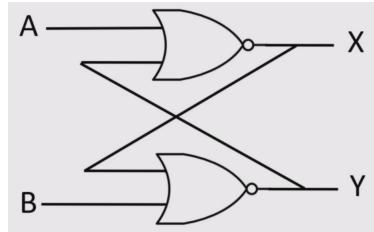
**הגדרה** שעון סיגナル בצורת גל מחזורי (0 ואז 1 ואז 0 וכו').

**דוגמה** כיצד השער הבא יתנהג?



אם הקלט בהתחלה הוא  $X = 0, Y = 0$  אז  $X = 1$  ו $Y = 0$  ונקבל מצב יציב של  $(X, Y) = (0, 1)$ . בדומה לכך אם  $X = 1, Y = 1$  אז  $X = 0$  ו $Y = 1$  ונקבל מצב יציב של  $(X, Y) = (1, 0)$ . כלומר, השער יתנהג כצורה דו-יציבה (עם שני מצבים יציבים), שיכולה לשמש אותנו לאחסון זכרון.

**דוגמה** נביט בשער הבא, שנקרא SR Latch (惦記 השערים במעגל הם NOR-ים)



הפ' הזו מקיימת  $X(t+1) = (A + Y(t))'$ ,  $Y(t+1) = (B + X(t))'$  לאחר שינוי קלטיהם מסונכון עם שעון שביצעו  $t$  טיקים), או בטבלת אמת עמוסה

A	B	X(t)	Y(t)	X(t+1)	Y(t+1)
0	0	0	0	1	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	1	0	0
1	1	1	0	1	0
1	1	1	1	0	0

A	B	X(t)	Y(t)	X(t+1)	Y(t+1)
1	0	0	0	0	1
1	0	0	1	0	1
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0

ואם נשלוף רק את הערכים המעניינים, נוכל להציג בתופעה מעניינת (חצים מעגליים משמשים למשך טיק נושא באותם הערכים וחיצים אומרים שנעבור לזוג ערכים אחר)

	A=0, B=0		A=0, B=1		A=1, B=0		A=1, B=1	
	X=0	X=1	X=0	X=1	X=0	X=1	X=0	X=1
Y=0								
Y=1								
Stable		Clear (Y)		Set (Y)		Undefined		

ונשים לב שגם  $X = Y$  נקלט מצב לא יציב (לא משנה מה ערכי  $A, B$  תמיד יש חץ שיוציא אותו למצב אחר) ולכן תמיד נניח שאנו נמשתמשים ב-SR Latch כאשר  $Y = X' = X = Y = 0$  (זה דומה למצב בדוגמה הקודמת שבו  $X = Y = 1$  שווה לא מוגדר בכלל כי יש לנו מהפך עם אותו ערך משני הצדדים).

אם כן, עבור  $(X, Y) = (0, 0)$ , נקלט  $(A, B) = (0, 0)$  ואם  $(X, Y) = (1, 1)$  נקלט  $(A, B) = (1, 1)$  אז מודלים את  $Y$  ואם  $(A, B) = (1, 0)$  מופיעים את  $Y$ , ואם  $(A, B) = (0, 1)$  אז מודלים את  $X$ .

לכן, באמצעות  $(S, R) = (A, B)$  נוכל לשנות בערכו של  $Y$  כשייש לנו זכרון של המצב הקודם, עם טבלת אמת מצומצת נוחה ביותר, כאשר  $Q(t) = Y(t) = X(t)'$  (בהתעלם מהמקרים הלא חוקיים)

S	R	Q(t+1)	Q'(t+1)
0	0	Q(t)	Q'(t)
0	1	0	1
1	0	1	0
1	1	0	0

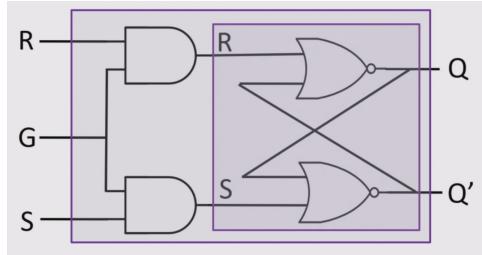
**הערה** מעגל סדרתיים ניתן לייצג באמצעות טבלת עירור (הטבלה הנ') וטבלת מעברים, שועונה על השאלה "אילו קלטים נדרשים כדי לעבור בין מצב כלשהו לאחר" ( $\Phi$  הוא ערך 'Don't care')

From $Q(t)$	To $Q(t+1)$	S	R
0	0	0	$\Phi$
0	1	1	0
1	0	0	1
1	1	$\Phi$	0

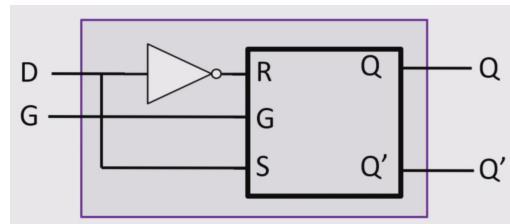
**הערה** יש היגיון בשמות הביטים S,R - אם רק (et) S דлок, קבועים את הפלט להיות 1, אם רק R (eset) דлок קבועים את הפלט להיות 0, ואם לא זה ולא זה דלוקים לא עושים כלום, כלומר שומרים על המצב כמוות שהיה. כMOVED שתחת סימולים אלה, גם S וגם R דלוקים וזה לא מוגדר היטב.

## מעגלים סדרתיים מורכבים על בסיס SR Latch

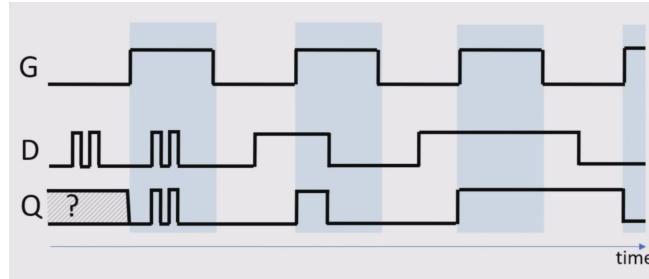
- **Gated SR Latch** זה שער שמנוע פליטת ערכים לא חוקיים מ-S,R, והוא מקבל קלטים S,R,G כאשר G הוא בית שער שאם הוא דлок אז המעגל מתפקיד כ-SR Latch רגיל, ואם כבוי אז הפלטים לא משתנים לא משנה מה. המימוש הוא די פשוט, כולל AND של G עם S,R עם G לפני הכניסה למעגל הפנימי (ראו איור)



- **Gated D-Latch** הוא גרסה אפילו יותר בוטואה ל-S-R Latch - במקומם לקבל קלטי S,R, מקבל D שיהיה מחובר ל-S ודרך מהפך ל-R, וכך לא מקבל מצב של (1,0), וכי לקלט (0,0) אפשר פשוט לכבות את G (ראו איור)



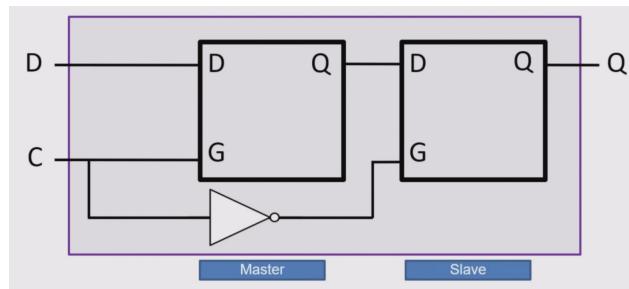
**הערה** לרוב נחבר את השעון ל-G, כלומר אפשר לשנות את הערך ורק כשחשעון בטיק שערכו 1. **דוגמא** בהינתן שעון שמחובר ל-G ובית D שערכו משתנה, נוכל לחשב מה יהיה הפלט Q, כפ' של הזמן. השטח המקוקו בהתחלה פירשו שלא ידוע לנו מה הערך של Q שכן הוא לא מוגדר בשלב זהה.



## D Flip Flop

בנה מעגל Shifter, שהוא מעגל שבו בית הקלט  $Z$  צעד אחד ימינה בכל מחזור. אם נשרשר D-Latch-G-ים שכלה-G-ים שלהם מחוברים לשעון, הקלט יופיע מיד בסוף השרשור (למעט זמן פגיעה של החישמל שהוא זניח).

הפתרון זהה הוא להוסיף מהפץ בין השעון ל-Latch השני כך שכשהשעון ב-1 רק ה-Latch הראשון יוכל לשנות את ערכו וכשהשעון ב-0 רק ה-Latch השני ישנה את ערכו והראשון לא ישנה. שער כזה נקרא D-Flip Flop.



לכן רק בעת ירידת השעון מקבל שינוי של הפלט  $Q$ , כי לאחר העליה ה-Master ישנה את הפלט ולאחר הירידה ה-Slave ישנה את הערך, הלא הוא פلت המעגל כולו. כל עוד השעון לא יריד, הערך ישאר זהה לערך שקיבל בירידת השעון האחורונה.

**הערה:** ניתן לבנות מעגל שישתנה רק בעליה באמצעות הזזת המהפץ לפני השער של המאסטר ולא העבד.

## תרגול

**הגדרה:** אלגברה בוליאנית היא מבנה אלגברי המוגדר על קבוצת איברים  $B$  עם שני אופרטורים בינאריים  $\cdot, +$ , מתקיימות אקסiomות Huntington: סגירות לכפל וחיבור, קיום איברי ייחידה לכפל וחיבור, קומוטטיביות בחיבור וכפל, דיסטרויטיביות (שני הנוסחים), קיום משלים (כך  $x \cdot x' = 0, x + x' = 1$ ) ו- $|B| \geq 2$ .

**הגדרה:** אלגברה בוליאנית דו-ערךית מוגדרת על קבוצה בת שני איברים  $B = \{0, 1\}$  עם האופרטורים  $x \cdot y = \text{AND}(x, y), x + y = \text{OR}(x, y)$  ו- $x' = \text{Not}(x)$ .

**טענה:** באלגברה בוליאנית דו-ערךית מתקיימות התכונות הבאות:

- אידempotentיות:  $x \cdot x = x + x = x$  לכל  $x$ .

- $x \cdot 0 = 0, x + 1 = 1$

- אסוציאטיביות לחיבור וכפל.

- חוק הצמצום :  $x(x+y) = x, x+xy = x$

- $(x')' = x$

**הערה** סדר האופטורים בחישוב ביטויי בוליאני הוא קודם סוגרים, אז NOT, אז AND ואז OR.

### דרכים לבטא פונקציה בוליאנית

- ביטוי בוליאני (ביטוי על המשתנים עם שני האופרטורים ושלילה).

- טבלת אמת : לטבלה יהיו  $2^n$  שורות כאשר יש  $n$  קלטים.

- סכום מכפלות : מספיק שמכפלה אחת תהיה 1 כדי שהסכום יהיה 1. כדי להציג סכום מכפלות, סוכמים את כל המכפלות הסטנדרטיות (minterm) שמקבלות 1 בטבלת האמת.

המשתנה ה- $j$  מקבל שליליה ב-minterm ה- $i$  אם "ס הבית ה- $j$ -ב- $_{(2)}^i$  הוא 0.

- מכפלת סכומים : מספיק שסכום אחד יהיה 0 ואז כל המכפלה היא 0. כדי להציג סכום מכפלות עם שליליה על הביטוי ושימוש בשני כללי דה-מורגן.

המשתנה ה- $j$  מקבל שליליה ב-maxterm ה- $i$  אם "ס הבית ה- $j$ -ב- $_{(2)}^i$  הוא 1.

נרצה לצמצם את מספר הליטרלים שלנו (מספר השערים).

**דוגמה** נביט ב- $F(x, y, z) = xy' + x'z - F_2(x, y, z) = x'y'z + x'yz + xy'$ . את הפ' השנייה ניתן למש עם משמעותית פחות שעריםalogisms (יש הרבה פחות פעולות) אבל הפ' שקוילות ולכן את השנייה תמיד!

את השקוילות אפשר להראות עם טבלת אמת או באמצעות אלגברה בוליאנית :

$$\begin{aligned}
 xy'z + x'yz + x' &= x'zy' + x'zy + xy' \\
 &= x'z(y' + y) + xy' \\
 &= x'z + xy' \\
 &= F_2(x, y, z)
 \end{aligned}$$

$$\begin{aligned}
 F(x, y, z) &= (x + y)[x'(y' + z')]' + x'y' + x'z' \\
 &= (x + y)[x + (y' + z')'] + x'y' + x'z' \\
 &= (x + y)(x + yz) + x'y' + x'z' \\
 &= (xx + xyz + yx + yyz) + x'y' + x'z' \\
 &= \dots = 1
 \end{aligned}$$

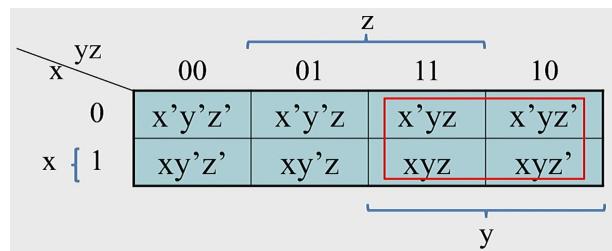
## מפות קרנו

מפת קרנו בונים פעם אחת לכל הפ' הبولיאניות ב- $n$  משתנים. ל- $n$  משתנים יש מפת קרנו עם  $2^n$  משבצות.

ראשית נבנה טבלת אמת לפ' הפוליאנית, ואז נציב את ה-minterm-ים בשבלונה של מפת הקרנו אם נצליח לשன אותה.

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

לחולופין נוכל לבנות מחדש את הטבלה עם האינטואיציה שבסיסנים הכהולים למשתנים, באמצעות ערכי  $x$  ו- $yz$  ניתן להסיק בקלות את המinterm-ים (יש לשים לב שערכי  $yz$  הם לא לפי סדר לקסיקוגרפי)



**הערה** ארבעת המשבצות ש- $z$  מסומן עליהם נסכום לליטרל ייחיד שהוא  $z$ , כך גם על  $y$ , וכך גם השורה התוחטונה נסכמת ל- $x$ . נשים לב גם כי כל שני ריבועים סמוכים נבדלים בליטרל אחד בלבד.

כדי לבצע צמצומים נמצא קבוצות של ריבועים סמוכים שערכם בטבלה האמת 1 עבור הפ', כאשר הקבוצה חייב להיות חזקה של 2 (כולל 1) ועלינו לבחור קבוצות גדולות ככל האפשר. קבוצות יכולות לחפות וצריך לכסות את כל הריבועים. הטבלה היא מעגלית ולכן ניתן לבצע חצחות את הקצה מימין ולהמשיך ממשאל.

**דוגמה** עבור  $F(x, y, z) = \sum(2, 3, 4, 5)$  (סכום מכפלות עם אינדקסים). מפת קרנו שלנו היא הבאה, כאשר הגענו אליה או באמצעות השבלונה או באופן הבא: הסימונים של  $x, y, z$  אומרים לנו Aiife המשטנה מקבל ערך 1, ולכן במשבצת השנייה מימין בשורה העליונה לדוגמה, גם  $y$  וגם  $z$  הם 1 אבל  $x$  הוא 0, כלומר מדובר במקרה של  $(0, 1, 1)$  שבמקרה שלו זה 1 (כי זהו ערכו של המinterm השלישי שמהגדרת הפ' הוא 1).

			$z$		
		00	01	11	10
$x$	0	0	0	1	1
$x$	1	1	1	0	0
				$y$	

כדי למצמצם את הטרבלה עכשו נcosa את הטרבלה עם שני מלבים, אחד משמאלי למיטה ואחד מימין למעלה וכן נקבל ייצוג מינימלי של הפ' הבוליאנית שהוא  $F(x, y, z) = x'y + xy'$  (במלבן משמאלי משותף הכל חוץ  $-z$  וכן גם בסמלון מימין).

**דוגמה** ( $F(x, y, z) = \sum(0, 2, 4, 5, 6)$ ) הביטוי הלא מצומצם מכיל 5 ביטויים. מלאו איקשחו את הטרבלה ונקבל

		$z$			
		00	01	11	10
$x$	0	1	0	0	1
$x$	1	1	1	0	1
				$y$	

נשתמש בחפיפות וגם במעגליות ונקבל מלבן אחד משמאלי למיטה ועוד מלבן שכולל את העמודה השמאלית והעמודה הימנית יחד, ואז הביטוי המינימלי הוא  $F(x, y, z) = xy' + z' + y$  (בשמאלי למיטה נופל  $z$  ובמעגלי נופלים  $x$  ו- $y$ , פשוט עוברים על זוג ריבועים אופקי וזוג אנכי ורואים מה משתנה בהם, ומה משותף בהם).

מפת קרנו באربعة משתנים נראה כך, כאשר אפשר לזכור אותה באמצעות העובדה שערכי הצירים שלה (גם אופקי וגם אנכי) הם 2, 0, 1, 3, 4, 5, 6, 7 (היצוג הבינארי של המספרים).

		$wx$			$z$	
			00	01	11	10
$wx$	00	0000	0001	0011	0010	
$wx$	01	0100	0101	0111	0110	
$wx$	11	1100	1101	1111	1110	
$wx$	10	1000	1001	1011	1010	

**דוגמה** ( $F(x, y, z) = (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$ ) מתקבלת מפת קרנו עם הערכים הבאים (לא משנה איך מגיעים לזה)

		00	01	11	10	y
		wx	yz	w	x	
w		00	01	11	10	
w	00	1	1		1	
	01	1	1		1	
	11	1	1		1	
	10	1	1		1	
z						

נבחר את חצי המפה השמאלית כי זו שמנינה והמלבן הגדול ביותר שיש, ובשביל הערכים מימין נבחר שתי רכיביות מעגליות (אי אפשר את כולם ביחד כי זה נותן לא חזקה של 2).

החצי השמאלי נבדל ב- $w$ ,  $z$ ,  $-x$ , ו- $y$  מקבל 0 כלומר הביטוי הראשון הוא  $y'$ .

הרכיביה המעגלית העליונה נבדلت ב- $-x$  ו- $y$  ו- $w$ ,  $z$  מקבלים ערכים 0 שניהם, כלומר יש לנו  $w'z'$  והרכיביה המעגלית התחתונה שונה ב- $w$  ו- $y$  ו- $x$  ו- $z$  מקבלים ערכים 1 ו-0 בהתאם, כלומר הביטוי השני הוא  $xz'$ .

$$\text{סה"כ קיבלנו } F(x, y, z, w) = y' + z'w' + xz'$$

**הأدירה** לפעמים עבור צירופים מסוימים, לא יהיה אכפת לנו מה הוא פלט הפ', צירופים כאלה נקראים **צירופים אדישים** ונitin להשתמש בערך שיותר נוח אליו כך שיבוטלו ליטרלים רבים ככל האפשר. נסמן צירוף כזה ב- $\emptyset$ .

**דוגמא** ((1, 1, 1))  $F(x, y, z)$  היא הפ' הבוליאנית והצירופים האדישים הם (7) (כלומר רק).

		00	01	11	10	z
		x	yz	w	x	
w		0	1	0	0	1
w	00	1	1		1	
	01	1	1		1	
	11	1	1		1	
	10	1	1		1	
y						

עתה נוכל לבחור מלבנים יותר נוחים (ראו או איור) מאשר בהיעדר הצירוף האדיש כי אז לא היינו יכולים לבחור את השורה התחתונה כמעט והיו לנו אמנים עדין שני ביוטוים אבל עם יותר ליטרלים, כלומר יותר שערירים שזה פחות טוב.

**הערה** צירוף אדיש מתקבל לדוגמה כsharpכיב אלקטронאי איזשהו מעגל בכל מקרה מחובר להארקה כך שלא משנה ערכו עבור צירוף מסוים כלשהו.

**דוגמא** נביט במפת הקרן הbhאה עם שני צירופים אדישים. הבחירה הכி נוחה היא 0 לשמאלי ו-1 לשמאלי כ-1 לימני כי כל קומבינציה אחרת הייתה דורשת מאייתנו יותר משני מלבנים או מלבנים קטנים יותר (יותר ליטרלים)\* שזה פחות אידיאלי.

		00	01	11	10	y
		wx	yz	w	x	
w		00	01	11	10	
w	00	1	1		1	
	01	1	1		1	
	11	1	1		1	
	10	1	1		1	
z						

**דוגמה** אפשר להשתמש בມפת קרנו גם כדי לצלצלים מכפלה של סכומים. נביט ב-(5)  $F(x,y,z) = \sum (0,1,2,3,4,6,7) = \Pi(5)$ . כפי שניתן למטה, צמצום של סכום המכפלות דרוש לפחות 3 סכומים ואילו מכפלה סכומים דרוש בדיקת יטרל אחד.

	00	01	11	10
x\y	00	01	11	10
0	1	1	1	1
1	1	0	1	1

	00	01	11	10
x\y	00	01	11	10
0	1	1	1	1
1	1	0	1	1

כשomezים פ' בוליאנית לפי מכפלה סכומים, מבצעים בדיקת התהיליך של סכום מכפלות רק שמשכים 0-ים במקומות 1-ים. לאחר הcisoi, ממיררים את הcisoi למכפלה של סכומים, כאשר כל סכום מתאים למילון אחד.

**הערה** ניתן להוכיח נכונות של צמצום לפי מפת קרנו למכפלה סכומים או באמצעות דה-מורגן לצמצום של סכום מכפלות, או פשוט באופן ישיר מטבלת האמת ונכונות ייצוג המ-terms.

**דוגמה** נתונה הפ' הבוליאנית

$$F(w,x,y,z) = \sum (1, 2, 3, 11, 12, 13, 15) + d(w,x,y,z), \quad d(w,x,y,z) = \sum (5, 9, 10, 14)$$

- ציירו את מפת קרנו עבור הפ'  $F$ .

המפה תראה כך

	00	01	11	10
w\x\y\z	00	01	11	10
00	0	1	1	1
01	0	Ø	0	0
11	1	1	1	Ø
10	0	Ø	1	Ø

- כמה פ' שונות מיוצגות ע"י המפה?

$2^4 = 16$  כי אפשר לבחור ערכים שרירותיים עבור כל אחד מהצירופים האדישים שהוא ב"ת באחרים.

- רשמו סכום מכפלות מינימאי עבור  $F$ , האם הסכום ייחיד?

הכי נוח יהיה שהשורה השנייה תהיה כולה 0 והשלישית כולה 1, ובשורה הרביעית כמה שיותר 1-ים כדי שנקבלים מבנים של רביעיות במקומות זוגות. כך נקבל סה"כ את הcisoi הבא

	00	01	11	10
w\x\y\z	00	01	11	10
00	0	1	1	1
01	0	Ø	0	0
11	1	1	1	Ø
10	0	Ø	1	Ø

שנותנו לנו את הביטוי  $z'x'y + x'y + x'w$ . זה לא ביוטי מינימלי כי אפשר לבחור שכל הצירופים האדיישים יהיו 1 חוץ מ-(0,0,1,1).

ואז קיבל את הכיסוי הבא עם אותו מספר ליטרלים

		y			
		00	01	11	10
wx		00	1	1	1
w	00	0	Ø	0	0
	01	0	Ø	1	Ø
	11	1	1	1	Ø
	10	0	Ø	1	Ø

## פונקציות שלמות

הגדירה קבוצה  $F$  של פ' בוליאניות נקראת שלמה אם ניתן למשתכל פ' בוליאנית בעורף  $F$  בקבוצת  $F$ .

**משפט**  $\{ \cdot, \cdot', +, \cdot' \}$  היא שלמה.

**הוכחה:** כל פ' בוליאנית ניתנת להציג כסכום מכפלות שדורש רק את שלושת הפעולות הללו.

**מסקנה**  $\{ \cdot, \cdot', +, \cdot' \}$  הן שלמות.

**הוכחה:** עם דה-מורגן אפשר ליציר  $+$  עם  $\cdot'$ , ו- $\cdot$  עם  $\cdot$ .

**דוגמה** NAND, קלומר  $(y \cdot x)'$  היא פ' שלמה. ראשית ניתן להשיג NOT ( $\cdot'$ ) באמצעות AND-ו- $x$  ( $x \cdot x = x'$ ). לכן ניתן להשיג באמצעות NOT על NAND, שניהם כבר יש לנו.

**דוגמה** הוכיחו כי  $f(x, y, z) = x' + yz$  והפ' הקבועות 0, 1 הן קבוצה שלמה.

ראשית ל-NOT ניתן להגיע באמצעות  $x' = f(x, 0, 0)$ . f(x, 0, 0) =  $x' + 0 \cdot 0 = x' + 0 = x'$ . f(x, 0, 0) אפשר להגיע ע"י  $f(1, x, y) = f(x, 0, 0)$ . לכן ניתן ליציר קבוצה שלמה כלומר הקבוצה המקורית היא שלמה. אפשר גם להגיע ל-OR ע"י  $f(f(x, 0, 0), y, 1) = (x')' + y \cdot 1 = x + y = x + y$ .

למעשה לא צריך את שני הקבועים כי ברגע שיש NOT עם אחד הקבועים אפשר להגיע לקבוע אחר עם NOT על הקבוע שכן יש לו.

## שבוע | IIIII

### הרצאה

**דוגמה** נתונה הפ' עם טבלת האמת הבאה

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

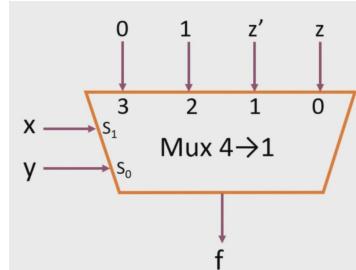
- נממש את הפ' עם 8 MUX → 1 יחיד.

כל מה שצרכי לעשות זה לחבר את  $z$  ל- $s_2, s_1, s_0$ , להציב בקלטים  $x_0, \dots, x_7$  של ה-MUX את ערכי  $f$  בטבלה האמתה לפי הסדר. כך נבחר עבור כל צירוף  $(x, y, z)$  בדיקות התוצאה מתוך ערכי טבלת האמתה של  $f$ .

- עתה נממש עם 4 MUX → 1 יחיד ועוד שער אחד בלבד. כאן צריך יותר להתחמץ. ראשית נביט בטבלה כאשר  $(x, y)$  מופרדים מ- $z$ , כך שלכל  $(x, y)$  יש שני צירופים עם  $z$  עם ערכים אפשריים.

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

בנייה MUX יחיד עם סלקטורים  $y, x$  ובקלט ה- $z$ - נשים או קבועים שני הצירופים עם  $z$  נתונים את אותו ערך, או  $z$  או  $z'$  בהתאם לערך השער משטנה (שכנו עצמכם שאכן אלו כל האפשרויות). כך קיבל את השער הבא



אנחנו מקיימים את הדרישות כי יש MUX אחד ושער אחד שהוא NOT על הקלט השני ל-MUX.

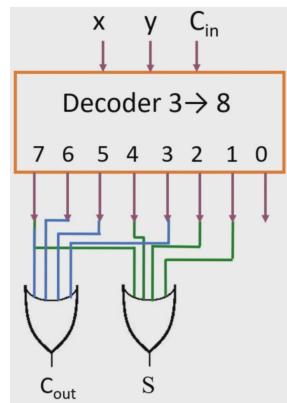
**דוגמה** נממש FA (שמקבל  $S, C_{out}$  ופולט  $x, y, C_{in}$ ) שיש לו את טבלת האמת הבאה (לצורך נוחות) עם :

x	y	$C_{in}$	S	$C_{out}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

• מפענה  $3 \rightarrow 8$  ושערי OR.

נבחר קלטים למפענה  $x, y, C_{in}$ , ואז על הפלטים נצמיד שער OR אחד לפט  $S$  ואחד לפט  $C_{out}$ . מה"כ זה יראה כך (יראה ב-

(1, 1, 1) מוציאה חוט לשני ה-OR-ים, בהתאם לטבלת האמת)



•  $8 \rightarrow 1$ -MUX .

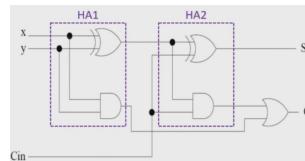
נבחר את הסלקטוריים להיות  $x, y, C_{in}$  ונשכץ בשמוות הקלטים של ה-MUX את ערכי  $S$  בטבלת האמת לכל צירוף של הסלקטורים

(הקלטים), והואתו הדבר שוב עם  $C_{out}$ .

•  $4 \rightarrow 1$ -MUX .

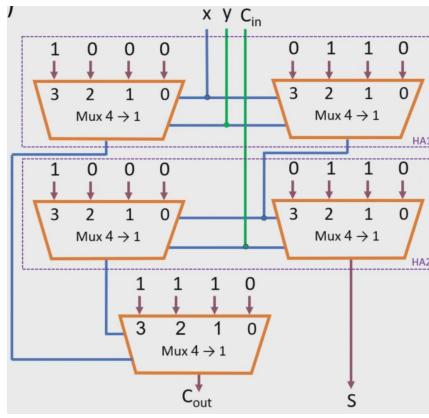
זה כבר יותר מרכיב, והורש פירוק של FA לשני Half Adder-ים שלא הזכירנו כאן, אבל הם שערים שמקבלים  $x, y$  ופלטים

$S, C_{out}$ . מימוש די פשוט ניתן לראות בתוך המלבן XOR הקלטים והנשא הוא AND על הקלטים.



כך נוכל לחבר יחד שניים כאלה כדי לחשב  $C_{in} = (x + y) + C_{in}$ .

עם  $MUX 4 \rightarrow 1$  (משבצים את ערכי טבלת האמת כאמור) וכל שנותר הוא להרכיב שני HA לאחד יותר גדול (ראו איור)

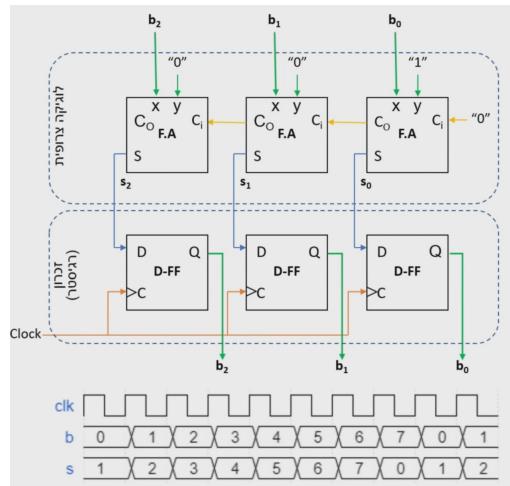


לאחר ראיינו DFF, שמקבל קלט D ושעון ומשנה את פלטו בעת עליית השעון לערך של D (זו וריאצית ה-Edge) נוכל על בסיס יחידה זו לבנות יחידות יותר מורכבות.

**דוגמה** Toggle FF מקבל T ושעון ומחשב בכל עלייה שעון  $Q(t+1) = \text{XOR}(T, Q(t))$  יחד עם  $T$  אל תוך XOR  $Q(t+1) = \text{XOR}(T, Q(t))$  ייחד עם  $T = 0$  אם  $Q(t)$  ואם  $T = 1$   $Q(t+1)' = Q(t)$  אחרת.

**דוגמה** JK-FF מקבל  $J, K$  ושעון ומחשב בכל עלייה שעון  $Q(t+1) = JQ'(t) + K'Q(t)$  והפלט  $Q(t+1)$  יתבצע בשער לוגי שתוצאתו מזונת- $D$  של ה-DFF הפנימי.

**דוגמה** נממש ספונ, שסופר את מספר עליות השעון.



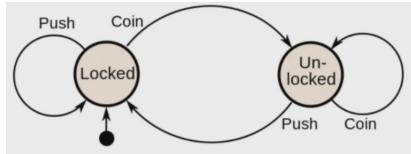
הרעיוון כאן הוא פשוט אבל המימוש לא כל כך: בכל עלייה שעון, ה-FA הימני ביותר מכניס עוד ערך 1 לסכום שמוחזק ע"י כל המעלג. הסכום מופיע בכל עלייה שעון ל-DFF הבא (זה-FA המתאים לו), וכך ה-DFFים מחזקים שלושה ביטים שמיצגים מספר שערכו עליה באחד בכל עלייה שעון (הסתודנטית המשקיעה תרים את שלושת המוחזרים בראש/על נייר ותראה שהוא אכן עובד).

## Finite State Machine

**הגדרה** FSM הוא מודל חישובי עם מספר סופי של מצבים, מצב התחלתי, מספר סופי של קלטים ופלטים, פ' מעברים  $\rightarrow \{\text{קלטים} \times \{\text{מצבים}\} \rightarrow \{\text{מצבים}\} \times \{\text{פלט}\}$ .

**דוגמה** שער מסטובב (Turnstile) יכול להיות פתוח או סגור, אם הוא מקבל מטבע והוא נועל, הוא נפתח, אם הוא לא נועל ונדחף, הוא נועל. כן המצבים הם "פתוח" ו"נועל", הקלטים הם מטבע ודחיפה, הפלט עבורו "פתוח" הוא "אפשר לעبور" ובהתאמה עבורו "נועל".

ונוכל לצייר את ה-*FSM* עם גראף



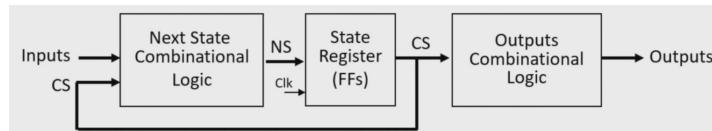
או טבלה (יש כמה דרכים)

State	Output	Input	Next State
Locked (init)	Closed-pass	Coin	Unlocked
		Push	Locked
Unlocked	Open-pass	Coin	Unlocked
		Push	Locked

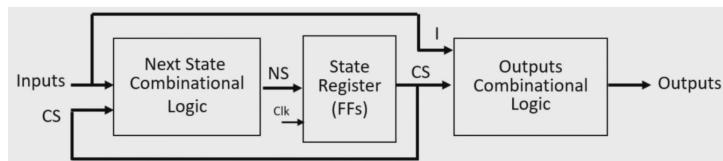
State	Output	Inputs	
		Coin	Push
Locked (Init)	Closed pass	Unlocked	Locked
Unlocked	Open pass	Unlocked	Locked

**דוגמה** מכונה כבאיור (Moore) היא מכונה שבה המצב הבא והווכחי בהתאם, שמה שמייחד אותה הוא שהפלטים תלויים אך ורק במצב הנוכחי.



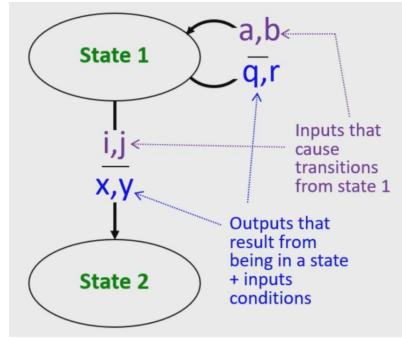
נשים לב שהרגיסטר שמורכב מ-FF-ים מחזיק את המצב הנוכחי, וערכו מחוות חזרה פנימה לחישוב המצב הבא שיכנס לרגיסטר במחזור הבא.

**דוגמה** מכונה Mealy היא מכונה שבה הפלטים תלויים גם במצב וגם בקלטים, והארQUITטורה שלו היא כבאיור



**הערה** את הייצוג של הגרפי של מכונת Moore מצירירים כמו אוטומט רגיל, רק שם כל מצב (מעגל) מכיל גם את הפלטים שהוא משרת. את הייצוג הגרפי של מכונת Mealy מצירירים כמו אוטומט רגיל, רק שעל החצים (המעברים) נוסיף את הפלטים שהקלטים על החץ יחד עם המצב שעוברים אליו משרים (ראו איור).

**הערה** הפלטים יושפעו מהקלט מהר יותר ב-Mealy כי הם מחוברים ישירות לפ' הפלטים, בעוד ב-Moore נדרש עלייה שעון כדי שישתנה המצב המשרת פלט.



**הערה** אפשר לכתוב מכונות Mealy ששוולות למכונות Moore עם פחרות מלבנים, אבל החסרונו הוא ש- Mealy גורם לביעיות עם תזומותים (שנלמד בהמשך).

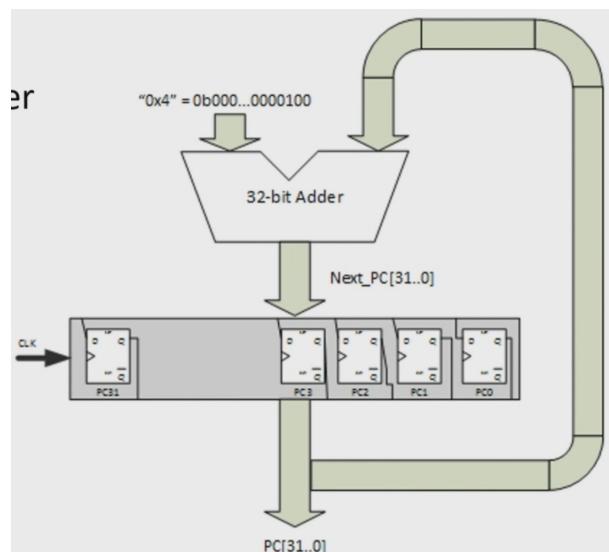
**דוגמה** נזכיר לדוגמת השער המסתובב. נגידר 0 השער נעל-ו-1 הוא פתוח. לכן המעגל שלפ' הפלט הוא חוט ישיר מהמצב לפלט כי הפלט זהה במצב. את המעברים ניתן לראות בטבלת האמת הבאה

Current State	Coin	Push	Next State
0	0	X	0
0	1	X	1
1	X	0	1
1	X	1	0

והמיוש של המעגל מעבר למצב הבא הוא די פשוט: אם  $Q$  הוא המצב הנוכחי ו-  $D$  המצב הבא (הקלט ל-DFF) אז  $D = \text{Coin} \cdot Q + \text{Push}' \cdot Q'$  שזה ניתן למימוש עם שערים מאוד פשוטים.

**דוגמה** Program Counter נתונים למעבד בכל פעם את הכתובת בזיכרון ממנה צריך לקרוא את הפקודה הבאה. הספרן פולט כתובות באורך 32 ביטים שkopatzת בקייזות של 4 (אלא אם הייתה קפיצה למקום אחר) בגלל שככל מילה היא באורך 32 ביטים (בית הוא 8 ביט).

המיוש הוא די פשוט: נחזק 32-DFF-ים שייחזקו את הכתובת, נחוות יישירות את ה-DFF ל-32 הפלטים והמעגל לחישוב המצב הבא הוא FA עם FA  $y = Q \cdot x = Q \cdot 4$  (המצב הנוכחי), או באior ברזולוציה נמוכה משום מה זה יראה כך



זהה מכונת Moore, שערכה מתעדכן פעם אחת בכל מחזור.

## תזמון מעבד

**הגדרה** התדר של מעבד הוא מספר המוחזורים של השעון בשנייה (ביחידות Hz).

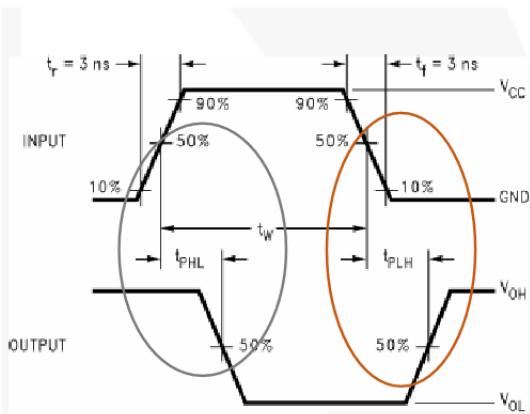
טעינה ופריקה של מטען לוקחים זמן ולכון מעבר של זרם בתוך טרנזיסטור אינם מיידיים (אלא מאוד מהירים). פרק הזמן זהה נקרא Propagation Delay.

## פרמטרים של זמן פעוף

- זמן הפעוף מנמוך לגובה ( $t_{PLH}$ ) : זמן הפעוף כשהפלט עובר מנמוך (0) לגובה (1). מודדים אותו החל משינוי ניכר בקלט ועד לעליית הפלט ל-50% מתח. בפועל מתח נחسب גובה (ומתפרש כ-1) רק כשהוא 90% ומעלה מערכו המקסימלי, וכך יש הנחה סבואה שהעליה והירידה של המתח הם מאד מהירים ולכון מ-50% ל-90% אין הבדל משמעותי.
- זמן הפעוף מגובה לנמוך ( $t_{PHL}$ ) : זמן הפעוף כשהפלט עובר מגובה לנמוך, מחושב ע"י הפרש הזמנים בין שינוי ניכר בקלט ועד לירידת הפלט למתחת ל-50%.
- זמן עלייה ( $t_r$ , Rise) : הזמן שלוקח לעלות מ-10% מתח ל-90% מתח.
- זמן ירידת ( $t_f$ , Fall) : הזמן שלוקח לרדת מ-90% ל-10% מתח.
- זמן פעוף ( $t_{pd}$ ) : כש- $t_{pd} = t_{PHL}$ , נקרא להם.

**הערה**  $t_r, t_f$  הם מאוד קטנים (ננו-שניות).

**דוגמה** בשער NOT כלשהו מקבלים את הגירף הבא של הפלט כתלות בקלט.

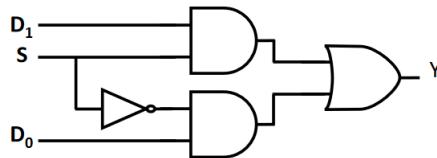


נשים לב שהשינוי הוא לא מיידי. במקרה הזה השינוי הניכר בקלט הוא חצייתו (מלמעלה או מלמטה) של הקלט את רף 50% המתח.

הגדירה עבור  $t_{pd}$  יש ערך טיפוסי, ערך מקסימלי ( $t_{pd\_max}$ ) שאומר אחרי כמה זמן בטוח הפלטים כבר ישתנו וערך מינימלי ( $t_{pd\_min}$ ) שمبתייח מתחת לאיזה רף בטוח הערכים לא ישתנו.

**הערה**  $t_{pd\_min/max}$  יכולים להיות שונים עבור שינוי בקלטים שונים (קלט  $x$  משפיע יותר מהר מאשר  $y$ ).

**דוגמא** נתון השער X שmmaומש באופן הבא



וחסמי זמן פעוף לשערים

$t_{pd\_max}$	$t_{pd\_min}$	שער
5ns	2ns	NOT
8ns	4ns	AND
10ns	5ns	OR

- מהו  $t_{pd\_max}$  של השער כולם?

עבור ההשפעה  $Y \rightarrow D_1$  (כמה זמן לאחר שינוי  $D_1$  ישנה נctrך לעבור דרך AND ו-OR כלומר סה"כ 18ns, וכן גם עבור

$D_0$

עבור  $Y \rightarrow S$  זמן הפעוף המקסימלי הוא  $\max(AND + OR, NOT + AND + OR) = \max(18, 23) = 23$  ns.

זמן הפעוף של השער כולם הוא המינימום של כל המסלולים, כלומר 23ns.

- מהו  $t_{pd\_min}$  של השער כולם?

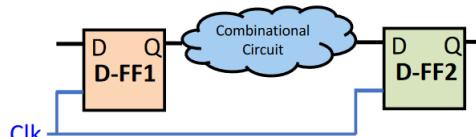
עתה נבחר את המסלול הקצר ביותר, שהוא כמובן  $D_0 \rightarrow Y$  ( $D_1 \rightarrow Y$  שדורש 4ns ועוד).

הגדירה זמן הפעוף של  $D \rightarrow Q$  מחושב (מודגר) החל מעלייה של השעון ל-50% ועד לשינוי  $Q$  והוא נקרא  $t_v$  (או  $t_{PCQ}$ ) והוא למעשה מגדי אחרי כמה זמן לאחר עליית השעון נוכל להчисיב את הקלט כחוקי.  $t_{v\_min}$  מבטיח עד מתי  $Q$  ישאר בערכו הקודם ו- $t_{v\_max}$  מבטיח החל ממתי יהיה הערך החדש.

הגדירה כדי  $Q$  יהיה תקין,  $D$  צריך להיות יציב ולא להשתנות במשך פרק זמן לפני עליית השעון, זהו (*Setup*,  $t_s$ ), וגם לאחר עליית השעון, זהו (*Hold*,  $t_h$ ).

**הערה**  $t_s > 0$  כי בתוך ה-DFF-ה-Master משנה את ערכו קצר לפני ה-Slave ולכן הערך שם צריך לא להשתנות כדי של-Slave יהיה את הערך הנכון.

**דוגמא** נביט בكونסטרוקציה הבאה



נניח שזמן מחזור השעון הוא  $t_{cyc}$  (זמן בין עליית שעון אחת לשניה), לכן קצב השעון הוא  $f = \frac{1}{t_{cyc}}$ . נניח כי זמן הפעוף המקסימלי של השער הוא  $t_{pd\_max}$ . מהו זמן המחזור המינימלי כדי שהמעגל יהיה תקין, כלומר כדי שההתוצאה תגיע ממהקלט ל-DFF1 עד לפולט של 2 תוך שני מחזורים (בראשו הקלטים עוברים את השער ובשני הם כבר מופיעים בצד השני)?

- זמן המחזור צריך לפחות  $t_{cyc}$ .

$$t_{cyc} \geq t_{v\_max} + t_{pd\_max} + t_{setup}$$

כדי שיהיה קודם לחכות שהפלט של DFF1 יהיה חוקי ( $t_{pd\_max}$ ), אז עלת לו לעבור את כל השער ( $t_{v\_max}$ ) ואז שהפלט יהיה ייחד מספיק זמן לפני עליית השעון הבאה כדי שייעבור בהצלחה ל-Q של DFF2 לאחר העליה.

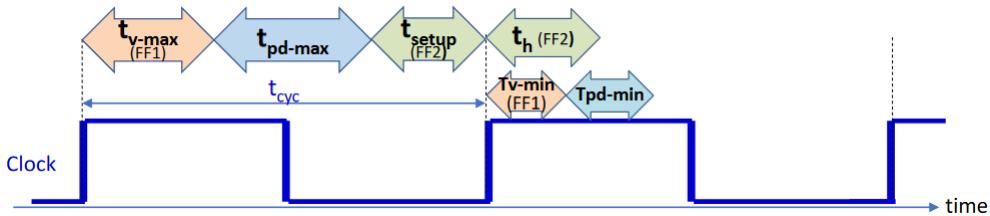
- מה צריך להיות  $t_h$  כדי שהשער יהיה תקין?

בפעם השנייה שבה ישנה הערך, נדרש ש- $t_h$  של DFF2 יהיה פחות מזמן שлокח ל-D של DFF2 להשתנות בהשפעת ה- $Q$ . החדש של DFF1. כלומר, נדרש ש- $t_h \leq t_{v\_min} + t_{pd\_min}$

$$t_h \leq t_{v\_min} + t_{pd\_min}$$

כדי הזמן שлокח לפחות לעבור מ- $D$  (DFF2) ל- $Q$  (DFF1) לא ישנה לאחר עליית השעון ( $t_{v\_min}$ ) ועוד הזמן המינימלי שעוברו ( $t_{pd\_min}$ ) לא יכול לקבל ערך חדש כפלט של המעגל ( $D$ ).

סה"כ מחלק התזומנים כפ' של השעון הוא באירוע



**הערה** אם אין לנו דרך לשלוט ב- $t_h$  ונרצה עדין מעגל חוקי, אפשר לחיבר את  $t_{pd\_min}$  להיות יותר גדול ע"י הוספה שני NOT-ים למסלול הקצר ביותר במעגל (הוא לרוב לא הארוך ביותר) וכך לא להשפיע על התוצאות אבל כן על התזמון המינימלי.

**דוגמא** נתונה הקונסטרוקציה הבאה עם MUX, שלו (הכלביות  $ns$ ,  $t_{pd\_max} = 9$ ,  $t_{pd\_min} = 23$ ,  $t_{v\_max} = 2$ ,  $t_{v\_min} = 7$ ,  $t_s = 3$ ,  $t_h = 5$ )

- מהו זמן המחזור המינימלי האפשרי?

כדי שהמעגל יהיה תקין, נדרש שמסלול הפעוף הארוך ביותר במבנה יהיה כולל מוכל במחזור אחד. המסלול הארוך ביותר הוא משינוי ב- $(S0, Q)$ , דרך חישוב ה-MUX ועד לשינוי הערך ב- $D$ ,ऋציך לקחת בחשבון שלפני תחילת המחזור הבא נדרש  $t_{setup}$  ש- $t_h$  יהיה ייחד ל- $t_{setup}$ . סה"כ נדרש שיתקיים (בדומה לדוגמה הקודמת)

$$t_{cyc} \geq t_{v(max)} + t_{pd(max)} + t_{setup} = 7 + 23 + 3 = 33$$

- מהי הדרישה על  $t_h$  כדי לקבל מבנה תקין?

נוצר שערך של  $D$  לא ישנה מוקדם מדי לאחר עליית השעון, בפרט שזמן שינוי הערך  $Q$  וחישוב ה-MUX יקחו יותר מאשר

$$(t_h = 5 \leq 9 \text{ ו } t_h \leq t_{v(min)} + t_{pd(min)} = 2 + 9) \text{ כלומר } t_h = 9$$

$$\text{סיה"כ} F_{max} = \frac{1}{33ns} = 30MHz \text{ ולכן } T_{cyc(min)} = 33ns$$

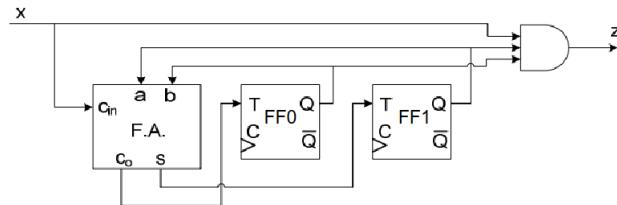
עד כה הטענו מהקלט האמיתי של המודול שמחובר ל-S1. הוא עצמו גם חייב לקיים דרישת זמן לשעון וכך נוכל להתייחס אליו כסיגナル שמסוגל מפלט אחר והניתוח יהיה כן".

כל קלט למעגל אחד הוא פלט של מעגל אחר, וכך יכולם להיות להם ערכי  $t_v, t_{v-min}$  שונים ממודול אחר.

כל פלט למעגל כלשהו הוא קלט למעגל אחר וכן הפלטים צריכים לעמוד בדרישות  $t_h, t_s$  מסוימות גם כן.

**הערה** כשהנתנו זמן של מעגל, נסתכל על כל המסלולים שמתחלים בכניסה לדFF ונגם

**דוגמה** נסתכל על המבנה הבא, כשהנתנו  $0 \leq t_{setup} = 9, t_{hold} = 0$  ודרישות על הפלט  $t_{v(max)}^x = 15, t_{v(min)}^x = 4$  (צריך להיות יציב לפחות 9ns לפני עליית השעון, ו-0ns אחרי)



עם הנתונים

Parameter	$t_{pd-max}$	$t_{pd-min}$	$t_{setup}$	$t_{hold}$	$t_v$	$t_{v-min}$
AND 3 inputs	3	1				
FA [a,b,Cin]->S	12	3				
FA [a,b,Cin]->Co	8	3				
T-FF			7	2	4	0

• מהו  $T_{cyc-min}$ ? נתנו את כל המסלולים שנגמרים בפלט (כי לפלט יש דרישות ביחס לשעון, בפרט לדFF יש

יש כזה שנדרש מאייתנו)

– מסלולים שנגמרים ב-z : המסלול המקסימלי הוא באורך

$$t_{setup} + t_{pd}^{AND} + \max \{t_v^{FF1}, t_v^{FF0}, t_v^x\} = 9 + 3 + \max \{4, 4, 15\} = 27ns$$

כדי להיות יציב זמן לפני עליית השעון, ערכו מחושב ע"י AND או מוסיפים את זמן הפעוע דרכו, וקלט

ה-AND הם פלטי FF0 ו-x בתאמה, שערכם מתעדכן למחוזר הנוכחי לאחר ה- $t_v$  של כל אחד מהם (כאן אנחנו

.( $t_v = t_{v(max)}$  מסומנים

– מסלולים שנגמרים ב- $T$ : המסלול המקסימלי הוא באורך

$$t_{\text{setup}}^{\text{FF1}} + t_{\text{pd}(\rightarrow S)}^{\text{FA}} + \max \{t_v^{\text{FF1}}, t_v^{\text{FF0}}, x_{\text{valid}}\} = 7 + 12 + \max \{4, 4, 15\} = 34ns$$

– מסלולים שנגמרים ב- $T$ : המסלול המינימלי לפעוף הוא

$$t_{\text{setup}}^{\text{FF0}} + t_{\text{pd}(\rightarrow C_0)}^{\text{FA}} + \max \{t_v^{\text{FF1}}, t_v^{\text{FF0}}, x_{\text{valid}}\} = 7 + 8 + \max \{4, 4, 15\} = 30ns$$

ולכן סה"כ נצרך  $T_{\text{cyc-min}} \geq \max \{27, 34, 30\} = 34ns$

- האם מתקיימת הדרישה על ה- $\text{Min Delay}$  ?

כן! עבור  $z$  מתקיים  $t_h^z = 0ns$  והזמן המינימלי לפעוף הוא

$$t_{\text{pd}(min)}^{\text{AND}} + \max \{t_{v(min)}^x, t_{v(min)}^{\text{FF}}\} = 1 + \min \{0, 0\} = 1ns$$

כלומר מתקיימת הדרישה ועבור  $t_h^{\text{FF}} = 2ns$  FF0, FF1 יש זמן המינימלי לפעוף הוא

$$t_{\text{fpd}(min)}^{\text{FA}} + \min \{t_{v(min)}^{\text{FF}}, t_{v(min)}^x\} = 3ns$$

(כי הערך החדש צריך לעבור דרך FA ולהגיע או משינוי ב- $x$  או משינוי ב- $Q$  של אחד ה-FF-ים) ולכן מתקיימת הדרישה גם כן.

## תרגול

**הגדרה** מעגל צירופי (קומבינטוררי) הוא מעגל שיש לו כניסה ויציאות כאשר האחרונות תלויות בכל ערך ורגע של הראשונות. מעגל סדרתי הוא מעגל שפלטיו תלויים גם ביחידת זכרון שמחוברת לשעון.

**דוגמה** כיצד נממש Full Adder (כזכור קלטים  $S, C_{out}$  ופלטים  $x, y, C_{in}$  (מיימן))?

$x$	$y$	$C_{in}$	00	01	11	10
0	0	0	0	1	0	1
0	1	0	1	0	1	0
1	0	1	0	1	0	0
1	1	1	1	1	0	1

$x$	$y$	$C_{in}$	00	01	11	10
0	0	0	0	0	1	0
0	1	0	1	1	1	1
1	0	1	1	1	1	0
1	1	1	1	1	0	1

ולכן אפשר למש את  $S$  עם OR על ארבעה פלטי AND (שכלולים קלטים שעוברים דרך מהפץ) ואת  $C_{out}$  אפשר קצת יותר עיל. ראיינו בהרצאה שהמיימוש של FA כולל בתוכו שני HA.

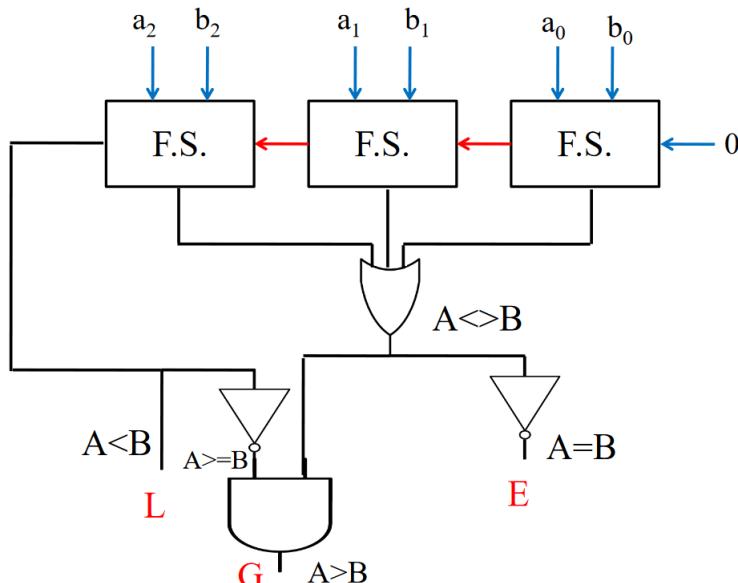
**הגדרה** מוחסרים מחשבים חישור ספרות בינהיות. עטן נפלוט את ההפרש (**בערך מוחלט**) ו-**Borrow Out** שיגיד לנו כמה יותר לחסר מעבר להפרש. הסטודנטית המשקיעה תמשח חצי-מוחסן ומוחסן מלא.

**הגדירה** משוויים הם מעגלים שבודקים איזה קלט יותר גדול.

- דרך אחרת היא באמצעות השוואת מוחסרים:  $A > B \iff A - B > 0$  וכן.
  - דרך נוספת היא באמצעות השוואת SB-MSB ל-LSB כאשר בפעם הראשונה שיש אי-שוויון בביטים נבחר את האחד שכללתו גדול יותר (1 לעומת 0).

**דוגמה** נתונם הקלטים  $A = a_2a_1a_0$ ,  $B = b_2b_1b_0$  ממשו עם מחסרים מעגל שפלט ביטים  $G, E, L$  שעריך כל אחד מהם 1 אם  $a_i > b_i$ ,  $A = B$ ,  $A < E$  בהתאם.

## נמש את המעל כבאיור



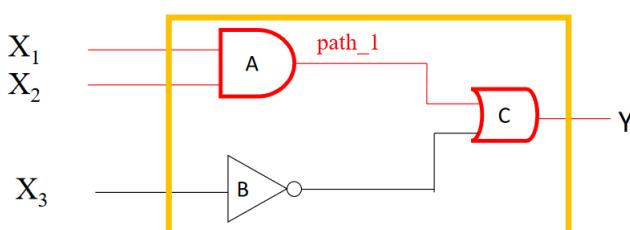
אם  $A < B$  בהכרח ישאר לנו Borrow Out כי אם  $a_2 = b_2$  אז ייה BO מ לפני שימוש הלהה ואחרת ולבו בודאי יהיה BO.

אם כל הביטים זהים נצטרך **NON** על כל הפרושים (ובן החיסוריים פוליטיים) ואנו זה מה שבנוינו.

בשיטות האלימינציה,  $G$  הוא 1 אם  $A \neq B$  ו-0 אם  $A = B$ . קלומר אמ-ה-OR הוא 1 וגם ה- $L$  הוא 0 וכך אנו מופיע במעגל.

**הערה** השימוש ב-10% ו-90% כרף לחישובי תצוםנו נובע מכך שקשה לאפיינו את פריקת הקבל כתהילד ל内幕י בקשות. ולכו מתעלמים ממה.

**דוגמה נבית בפ'**  $X_1 * X_2 + X'_2$  שэмומשת באופו הבא



מתתקיים  $t_{pd}(A) = \max\{t_{PLH}(A), t_{PHL}(A)\}$  ו- $X_3$  מ- $X_1, X_2$  והשני מ- $Y$ . כלומר  $t_{pd}(A) = \max\{t_{PLH}(A), t_{PHL}(A)\}$ .

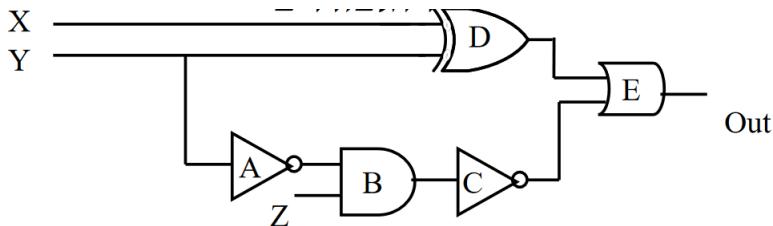
לכן (מסלול ראשון)  $t_{pd}(B) + t_{pd}(C)$  תחת הנתונים הבאים (כאשר מסלול שני)  $t_{pd}(A) + t_{pd}(C)$  ובההתאמה (Contamination מלשון  $t_{cd} = t_{tp-min}$ )

Data in ns	$X_2$	A	B	C
$t_{PHL}$	-	100	90	80
$t_{PLH}$	-	110	70	100
$t_{cd}$	-	12	8	10
$t_r$	14	20	12	18
$t_f$	15	17	13	19

נחשב את ה- $t_{pd}$  של המעגל כולו. עבור כל שער, וכן  $t_{pd} = \max\{t_{PHL}, t_{PLH}\}$  ו- $t_{pd}^{p2} = 90 + 100 = 190ns$ .

נחשב את ה- $t_{pd(min)}$  של המעגל.  $t_{pd(min)} = 12 + 10 = 22ns$ .

דוגמה נתונים המעגל והנתונים הבאים



Data in ns	A	B	C	D	E
$t_{pd}$	15	25	15	60	20
$t_{pd-min}$	5	5	5	10	5

המסלולים של שערים מהקלטים לפלייטים הם  $B \rightarrow C \rightarrow E$ ,  $A \rightarrow B \rightarrow C \rightarrow E$ ,  $D \rightarrow E$  ו- $Z \rightarrow$  ו- $Y \rightarrow$  בהתאמה.

$t_{pd(min)} = 15$  ו- $t_{pd} = 80$ .

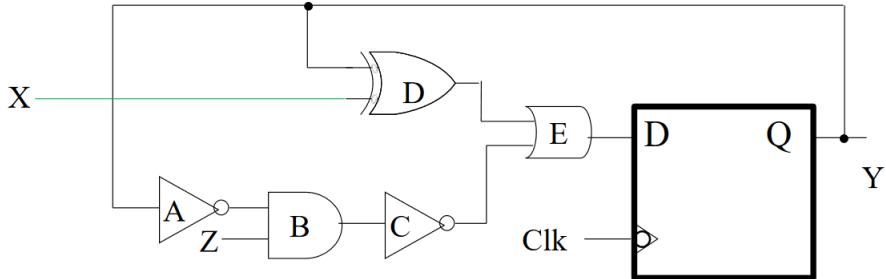
לכן  $t_{pd(min)} = 15$  ו- $t_{pd} = 80$ .

הערה כל עוד אין מעגלים סדרתיים, חישוב  $t_{pd}$  נעשה ע"י מקסימום הזמנים על כל המסלולים מפלטים לקלטים ו- $t_{pd(min)}$  ע"י מינימום.

הערה עבור כל מעגל סדרתי חיבר להתקיים  $t_{hold} \leq t_{v(min)}^{FF} + t_{pd(min)}^{לגיון}$  כדי שהשינוי הכי מהיר במעגל יקח יותר מאשר הזמן שהפלט צריך להישאר זהה אחרי עליית השעון.

הערה הקלטו למעגל חיבר להטייך לאחר לכל היוטר  $+ t_{setup}$  זמן כדי שה-FF יוכל לחשב באופן תקין את  $Q$ .

דוגמה נתון המעגל הסדרתי הבא עם הנתונים תחתוי



Data in ns	A	B	C	D	E
$t_{pd}$	15	25	15	60	20
$t_{pd-min}$	5	5	5	10	5

כדי להשלים את הניתוח של הזמן המוגדר נצטרך את האילוצים על ה-DFF  
 $t_{setup} = 10ns$ ,  $t_{v(min)} = 5ns$ ,  $t_v = 10ns$  DFF  $t_{hold} = 25ns$

• האם המוגל תקין?

לא! חיבר להתקיים  $25 = t_{hold} \leq 5 + t_{pd(min)} = 5 + t_{pd(min)}^{D \rightarrow E} = 5 + 10 + 5 = 20$  סתייה.

• כיצד ניתן את הבעיה?

ונסיף דילוי על המסלול הקצר ביותר, בפרט נוסיף שני NOT-ים על החוט בין  $Q$  לקלט העליון של  $D$  (ושל  $A$ ). עכשו המסלול הקצר ביותר יש לו  $25 = t_{pd(min)} = 5 + 25 = 30$  ועכשו קיבל  $25 = t_{hold} \leq 5 + 25 = 30$  כלומר זה כן תקין. החסרונו הוא שהתדר המקסימלי האפשרי ירד כי זמן המוחזר המינימלי עלה כתוצאה מהוספה שני NOT-ים.

## MIPS | VII שבוע

### הרצאה

**הגדרה** Instruction Set Architecture היא אוסף כללים שמתקנתן נדרש לענות להם כשהוא מפתח למעבד.

**דוגמה** אנחנו נלמד על MIPS, אבל במציאות פופולריים מאוד x86 ו-ARM, וגם V-RISC-Sh הוא פרויקט קוד פתוח.

**הגדרה** מיקרו-ארQUITקטורה היא מימוש של ISA.

**דוגמה** המימוש של אינטל ו-AMD ל-x86 הוא מיקרו-ארQUITקטורה.

לצורך האבstrקטציה שלנו, מעבד הוא מכונת מצבים המוחברת ל זיכרון, כאשר המცב כולל רגיסטרים שערכם משתנה על ידי פקודות. הזיכרון הוא מערך שניגשים אליו לפי אינדקס (בית אחד בכל פעם). כל מעבד מרים את התכנית הבאה:

1. קרא את הפקודה הבאה מהזיכרון בכתבota של ברגיסטר PC (Program Counter).

2. הוסף לרגיסטר PC את מספר הבטים שתופסת פקודה (התקדמות לפוקודה הבאה).
3. בצע את הפוקודה (וישנה את מצב המעבד).
4. חוזר לשלב 1.

בහינתן תוכנה בשפה עילית (שפתקמפלט), נתרגם את הקוד לסדרת פקודות אסמבלי באמצעות קומpileר. לאחר מכן נשתמש באסמבילר כדי לתרגם את קוד האסמביל לבינארי, שאותו המעבר כבר יודע להרץ.

**הערה** לעיתים הקוד שלנו ישמש בספריות חיצונית או בקבצי קוד אחרים, ועל אייחוי כל הפקודות לקובץ אובייקט יחיד. הפקודות באסמביל הן פקודות שהמעבד יודע לבצע (ח-ISA של המעבד), אבל לפני שהן מקודדות ל-1-ים ו-0-ים עבר המעבד.

## CISC vs RISC

- Complex Instruction Set Computer זו גישה לפיה פקודות האסמביל יהיו קרובות ככל הניתן לשפה עילית, וכך להוריד את מספר הפקודות בתוכנה. בפועל זה מומッシュ ע"י פקודות שפותחות למיקר-פעולות ע"י החומרה. ל-ISA הזה יש הרבה מאוד פקודות, בפורמטים שונים והרכבה של פקודות שונות אחת על השניה, ואפשר אפילו להריץ פקודות ישירות על הזיכרון שמסתירות את המעבד בריגיסטרים. אורך הפקודות יכול להשנות, כאשר נקודד פקודות שכיחות באמצעות בית אחד, עד לפקודות הנדירות ביותר שהן באורך 15 בתים.

8x DSP הולכים לפי גישת CISC.

- Reduced Instruction Set Computer היא גישה לפיה יש לשמור את מספר הפקודות מצומצם וכך לפחות את פעולות החומרה, ולתת לקומpileר לעשות את העבודה הקשה של בחירת הפעולות ואופטימיזציה. הפקודות הם די פשוטות והוא רק בין רגיסטרים (ולא על הזיכרון), ורק באופן מפורש לטעון ולשמור נתונים בזיכרון. אורך הפקודות הוא קבוע.

RISC-V MIPS, ARM RISC הולכים לפי גישת

**הערה** המגמה עם הזמן היא לעבור מ-CISC ל-RISC משום שבמעבר קשה היה כתוב קומpileרים עילאים ולכן נדרשה המורכבות של פקודות האסמביל, ואילו עם חלוף הזמן נהיה קל ויעיל יותר לכתוב קומpileרים (בשפה עילית) שיבצעו את הפעולה המורכבת עצמאם.

**דוגמה** עבור העתקה של 100 ערכים בין מערך אחד לאחד ב-c, יש ב-8x פקודה אחת שמקבלת את מספר הבטים להעתקה והפוניטרים והחומרה כבר תממש את ההעתקה, לעומת זאת RISC שם צריך למשולאה שמעיטה לרוגיסטר ואז לזכור מילה מילה.

## MIPS

במעבד MIPS יש 32 רגיסטרים, \$31, ..., \$0, שכל אחד מהם בגודל 32 ביט, המכונה "מילה" (4 בתים). מספר הרגיסטרים קבוע כך לאחר ניתוח של מספר המשתנים בתוכנות מדגמיות, כאשר אם יש יותר משתנים מרוגיסטרים נשתמש בזיכרון הראשי כדי להחזיק את ערכם. לחלק מהרגיסטרים יש יעודיים ספציפיים, כפי שניתן לראות בטבלה הבאה

Name	Register Number	Usage	Preserve on call?
\$zero	0	constant 0 ( <a href="#">hardware</a> )	n.a.
\$at	1	<a href="#">reserved</a> for assembler	n.a.
\$v0 - \$v1	2-3	returned values	no
\$a0 - \$a3	4-7	arguments	<a href="#">yes</a>
\$t0 - \$t7	8-15	temporaries	no
\$s0 - \$s7	16-23	saved values	<a href="#">yes</a>
\$t8 - \$t9	24-25	temporaries	no
\$gp	28	global pointer	<a href="#">yes</a>
\$sp	29	stack pointer	<a href="#">yes</a>
\$fp	30	frame pointer	<a href="#">yes</a>
\$ra	31	return addr ( <a href="#">hardware</a> )	<a href="#">yes</a>

ניסיונות רבים מהרגיסטרים משמשים פעילות תקינה של המחשבנית (SP, FP, RA), חלק שומרם ארגומנטים וחלק משומשים לצרכים אחרים.

כל פקודה היא בגודל מילה (32 ביט), וכל פקודה מבצעת פעולה פשוטה, בין היתר פעולות אРИטמטיות ולוגיות, גישה ל זיכרון (load, store) ופקודות ופוקודות מותניות. הפקודות שמורות בזיכרון ובכל פעם נקרא מהזיכרון את הפקודה ונರץ אותה.

## פקודות אРИטמטיות

- חיבור/חיסור : נביט בפקודה `add $d, $s, $t` (כאשר שלושת הפרמטרים הם רגיסטרים), ובדומה `sub $d, $s, $t`

ניסיונות להודענו למעבד בשום מקום שאנו מנסים להשתמשים בשיטת המשלים ל-2, כי אנחנו מניחים שניים שקיים קומפלט את הפקודה יודע מהקשר שהוא סוכם רגיסטרים שיש בהם כבר ערכים מיוצגים במשלים ל-2, ואם לא אז זה באג.

$$f = (g + h) - (i + j)$$

הקומפיילר יקצח רגיסטרים למשתנים ונקבל  $\$s0 = (\$s1 + \$s2) - (\$s3 + \$s4)$ , ואז לתרגם השורה לשפת אסמבלי יש כמה אפשרויות, הנאייה ביותר מתווכן היא

`add $t0, $s1, $s2`

`add $t1, $s3, $s4`

`sub $s0, $t0, $t1`

אבל יש דרכים אחרות (לדוגמה לפתיחת סוגרים). במתמטיקה אמנים התוצאות שקולות, אבל כאן יכול להיות שנתקבל תוצאה אחרת בגלגול-overflow.

- חיבור מיידי : הפקודה `i imm addi $t, $s + $t` מחשבת  $t = s + imm$  (כלומר חיבור בין רגיסטר וקובוע והשמה ברגיסטר).
- כאן ומן נטו לנו במשלים ל-2 אבל בגודל 16 ביט כדי שנוכל לכלול אותו בתוך קידוד הפקודה בגודל מילה אחת, ולכן נדרש להרחיב אותו למשלים ל-2 בגודל 32 ביטים, פועלה זו נקראת Sign Extend, וניתן למשה בклות ע"י ריפוד מצד של ה-MSB עם ערך קבוע של 0 לחוביים ו-1 לשיליינים (למעשה ערך ה-MSB לפני הריפוד).

הערה לא צריך `sub` כי אפשר למשה בקלות עם `addi` כאשר ה-`imm` הוא מספר שלילי.

## פעולות לוגיות

- הפקודה היא  $t = s \text{ and } d$  והוא מחשבת שער AND על כל שני ביטים מתאימים  $m-s$  ו- $t$  (בו זמנית על כל הביטים) ושומרת את התוצאה ב- $d$ .

- הזו לוגית: הפקודה  $a \ll d$  (מלשון  $a$  ביטים שמאל (לכיוון ה-MSB) את הביטים של  $t$  ושומרת את התוצאה ב- $d$ .

מימין מכנים אפסים ובנתיים מאבדים ביטים משמאלי, כאשר במקרה שמספרים מיוצגים במשלים ל-2 זה יכול לאבד את בית הסימן, ובכל מקרה נוכל לקבל overflow גם במקרה של טבעים.

- הזו אריתמטית: נשמר על בית הסימן גם לאחר ההזזה במקום להתעלם ממנו. ראו טבלה שמשווה את כל ההוצאות הקיימות ב-

.MIPS

Instruction	Operation	Description
<code>sll \$d, \$t, a</code>	<code>Shift Left Logical \$d = \$t &lt;&lt; a</code>	Shifts a register value left by the shift amount listed in the instruction and places the result in a third register. Zeroes are shifted in.
<code>sllv \$d, \$t, \$s</code>	<code>Shift Left Logical \$d = \$t &lt;&lt; \$s</code>	Shifts a register value left by the value in a second register and places the result in a third register. Zeroes are shifted in.
<code>sra \$d, \$t, a</code>	<code>Shift Right Arithmetic \$d = \$t &gt;&gt; a</code>	Shifts a register value right by the shift amount (shamt) and places the value in the destination register. The sign bit is shifted in.
<code>srav \$d, \$t, \$s</code>	<code>Shift Right Arithmetic \$d = \$t &gt;&gt; \$s</code>	Shifts a register value right by the value in a second register and places the value in the destination register. The sign bit is shifted in.
<code>srl \$d, \$t, a</code>	<code>Shift Right Logical \$d = \$t &gt;&gt;&gt; a</code>	Shifts a register value right by the shift amount (shamt) and places the value in the destination register. Zeroes are shifted in.
<code>srlv \$d, \$t, \$s</code>	<code>Shift Right Logical \$d = \$t &gt;&gt;&gt; \$s</code>	Shifts a register value right by the amount specified in \$s and places the value in the destination register. Zeroes are shifted in.

הזו אריתמטית של בית אחד שמאליה פרושה הכפלה ב-2 (עד כדי overflow) וימינה פרושה חלוקה ב-2 (עם עיגול כלפי מטה).

## פעולות זכרון

הזיכרון הוא מערך, שניגשים אליו באמצעות כתובות, כאשר כל כתובה מביאה לבית אחד של מידע, למרות שבפועל ב-MIPS נקרא ונכתב ביחידות של מילה (ארבעה בתים) מיושרות (כלומר כתובות שמתחלקות ב-4).

- קראת מילה:  $(s_i t_i) = \text{כתובת} + imm$  והוא ה-i(\$s + imm) והזזה אותה ב- $t$ .

דוגמה:  $0($a0), 0($t1, 0($t1, 0($t1,$  ו-

- כתיבת מילה  $(s_i t_i) = \text{כתובת} + imm$  לרגיסטר  $t$  לכתובת  $s + imm$  בזיכרון.

דוגמה יש לנו מערך A עם שלושה ערכים (מספרים, בגודל ארבעה בתים), שכתובת הבסיס שלו שומרה ב- $s$ . נוכל לגשת אל האיברים במערך באמצעות  $0($s3), 4($s3), 8($s3)$ .

דוגמה נניח שיש לנו את הפקודות ב-C

```
int A[100];
A[12] = h + A[8]
```

הקוד הזה יתורגם באסמבלי ל-

```

lw $t0, 32($s3) # load A[8] into a temporary register
add $t0, $s2, $t0 # add h to the temporary register
sw $t0, 48($s3) # save the result in A[12]

```

## פילוסופיות ארגון זכרון

- ארכיטקטורת ואן-ניומן : זכרון אחד לפקודות התוכנה וגם לששתני התוכנה. כש庫ראים מהזיכרון, משמעות התוכן (פקודה או מידע) תלוייה בפעולת השובילה לקריאה. אפ"פ שתיתכן הפרדה פיזית בין החלקים, לוגית הם ממופים לאותו המקום.

**יתרונות**  אזור זכרון מאוחד, וכל לדבג תוכנות ולשנות את אופן הטעינה.

**חסרונות**  קריאת פקודות וקריאת מידע קוראות על אותו המשאב.

ממומש ב-**x86, MIPS, ARM**.

- ארכיטקטורת הארוורד : הזכרון של הקוד מופרד מהזיכרון של המידע, כך ש-**lw** קורא רק מידע ו-**fetch** לפקודות קורא רק פקודות.

**יתרונות**  אין גישה במקביל למשאים שונים על אותו הפס - ביצועים יותר טובים.

**חסרונות**  חוסר גמישות בגודל הסגמנטים של קוד לעומת דאטא וקשה יותר לעורוך את הקוד לדיבוג.

ממומש בעיקר ב-**DSP**.

**הערה**  גם בוואן-ניומן המימוש בפועל יכול להיות באמצעות רכיבים פיזיים שונים, הגישה תהיה באמצעות אותו מרחב כתובות (אמנם כתובות אחרות, אבל באותו מיפוי).

## פקודות קפיצה והתנויות

- קפיצה בלתי-מוותנת : **label j** קופץ לאחר סיום הפקודה לשורת הקוד שמופיע לאחר ה-**label** (כפי שנכתב בקובץ ה-**asm**).).
- קפיצה מוותנת שוויזן : **beq \$s==\$t label** אם **\$s** שווה ל-**\$t** ואחרת ממשיכה לפקודה הבאה (ובדומה **bne** שקובצת אם **.\$(\$s!=\$t)**).

**דוגמה** נתון הקוד הבא ב-**C**

```

if (i == j)
    f = g + h;
else
    f = g - h;

```

הוא יתורגם לקוד אסמליל באופן הבא

```
bne $s0, $s1, not_eq # s0=i, s1=j

add $v0, $s2, $s4 # f = g+ h

j cont

not_eq:

sub $v0, $s2, $s4 # f = g - h

cont:
```

- השמה מותנת:  $\$s < imm\ 1\$t \rightarrow \$s = \$t$  אם  $\$s < \$t$  אחרת,  $\$s = imm$

## מימוש פונקציות במסבלי

**הגדולה הקוראת היא הפ' שקוראת לפ' אחרת, הנקראת היא הפ' שנקראת,** **הפרמטרים** הם הערכים שמועברים מהקורסית לנקרת, **התוצאות** הם הערכים שהנקראת מוחזירה **לקוראת וכותבת החזרה** היא הכתובת בזכרן הקוד של הפקודה שאחרי הקראיה לנקרת בקוד של הקוראת.

המחסנית היא אוצר בזיכרון שתוכנה יכולה לשמש בו, והיא משמשת שמירה של מידע לوكאלית בעת הרצת פ' באופן שמאפשר קריאה מקוונת וזרה מקריאות של פ'. מבנה הנתונים עובד בשיטת LIFO ואפשר או לדחוף (push) אלמנט לראש המחסנית, או להוציא (pop) את הערך העליון במחסנית.

**למיהה** העורה לעתים אפשר לגשת גם לערכים אקראיים במחסנית, ובכל מקרה ניתן לכתובות ביחס לכתובות ראש המחסנית - Top of Stack כאשר כל דבר מתחת אינו ואליidi. זאת משום שהמחסנית גדלה נגד כיוון הכתובות, כלומר הוספה ערך תזוז את TOS ארבעה בתים

**שמירת רגיסטרים בעת קריאה וחזרה מפונקציה**

(הקורסואט) הייתה באמצעות חישוב שקטור הא- $\alpha$  (הקורסואט). רק חלק מהרגיסטרים נשמרו, וע"י גורמים שונים:

- §\$7-\$§\$8 הם רגוליטריים סטטיים ולכון  $f$  מצפה שם לא ישתנו, כך שם  $g$  משתמש בהם היא תctrיך לשחזר אותם לערבים בטראם
  - §\$9-\$§\$10 הם רגוליטריים זמניים ולכון  $f$  (הקוראת) לשמר אותם במקום אחר במהלך הקריאה ל- $g$ .

- \$a0-\$a3 משמשים העברת והחזרת ארגומנטים ולכון הקוראת צריכה לשמור אותם אם היא רוצה להשתמש בערכיהם שהוא עצמה קיבלה (או תחזיר) בתור נקראת.

- \$ra הוא הרגיסטר שמחזיק את כתובת החזרה מהפ', והוא נדרשת ע"י הפקודה jal בעת הקראה לפ' הנקרה, כך שאחריות השימוש היא על הקראת.

- \$\$sp, המצביע לראש המחסנית, שנדרש לכל הפ' על מנת לפעול באופן תקין, ולכון הנקרה נדרשת לשחזר אותו בעת סיום הקראה לפ'.

את הערכים נשמר תמיד במחסנית.

- כדי לדוחו (ולשמור) את \$ra למחסנית נשימוש בפקודות

addi \$sp, \$sp, -4

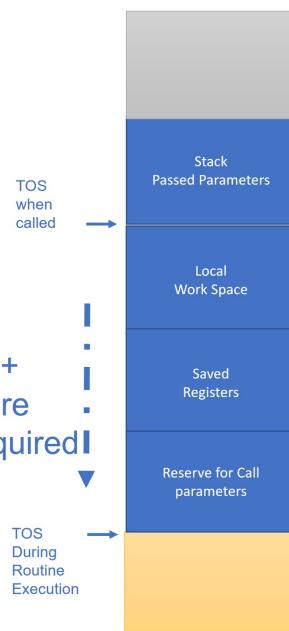
sw \$ra, (\$sp)

כלומר קודם מזיזים את המצביע לראש המחסנית מילה אחת למטה בזיכרון (מעליהם את גובה המחסנית) ואז כותבים לכתובת הבסיס של המחסנית את הערך של \$ra, כך שהוא עכשו בראש העירימה.

- כדי לגשת לערכים אפשר פשוט לבצעlw על כל היסט (חייב) ביחס ל\$sp.

- כדי לעשות kop קודם נקרה את המילה ואז נזיז כלפי מעלה את \$sp.

בעת קימפול נוכל לדעת כמה מקום פ' צריכה במחסנית ( מבחינת משתנים מקומיים, שמירת רגיסטרים ומקום לקרוא לפ' אחרות). מבחינה סידור הזכרון בעת קראה לפ', המחסנית תראה כך



## פקודות קראה לפונקציה

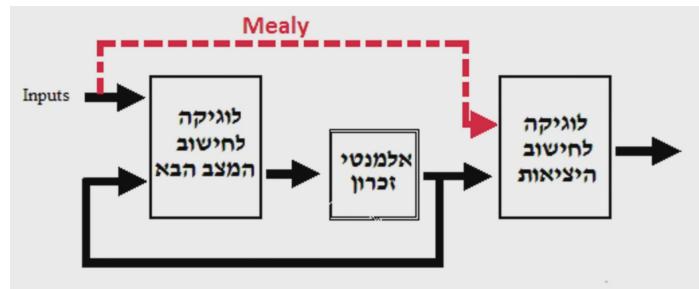
• `jal $ra $label` שם ב-`$ra` את כתובות הפקודה הבאה (הערך הבא שיקבל PC) ואז קופצת ל-`label`.

• `$s $r $j` קופצת לכתובות שברגייסטר.

כשנקרה לפ', נrix `jal`, וכשנচזור מפ', נrix `$ra $j`.

## תרגול

לודגמת ההבדל בין מכונות Moore ל-Mealy, ראו האיור הבא כאשר בשוחר Moore ובאdom התוספת שהופכת אותה למכונית Mealy.



## אנליזה של מעגלים סינכרוניים

בහינתן מעגל, נרצה להבין מה הוא עושה (איזה מכונה הוא מייצג, איך הוא מתנהג). השלבים לאנליהם הם :

1. הצגת הקלטים לזכרון ופלטי המעגל באמצעות הפלטים מהזיכרון והקלטים למעגל.

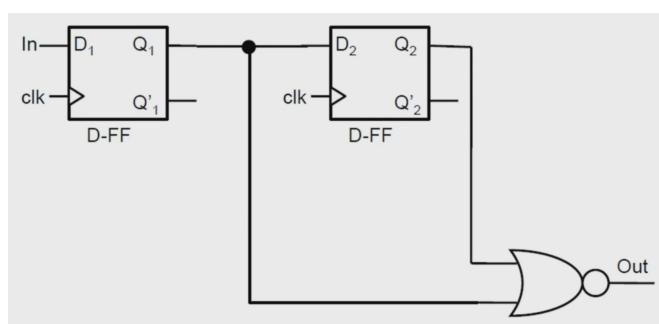
2. כתיבת טבלת מעברים.

3. כתיבת טבלת מעברים סימבולית (טבלה עם שמות למצבים).

4. רישום אוטומט מצבים.

5. ניתוח האוטומט באמצעות סדרת בוחן.

## דוגמה נתון המעגל הבא



1. בклטי הזיכרון מתקיים  $D_1 = In$ ,  $D_2 = Q_1$  ובפלט המעגל מתקיים  $Out = (Q_1 + Q_2)'$ .

2. טבלת המעברים תראה כך, כאשר אנחנו עוברים על כל אפשרות  $In, Q_1, Q_2, D_1, D_2$ .

		In=0		In=1		
$Q_1$	$Q_2$	$D_1$	$D_2$	$D_1$	$D_2$	Out
0	0	0	0	1	0	1
0	1	0	0	1	0	0
1	0	0	1	1	1	0
1	1	0	1	1	1	0

כאשר חלק מהקומבינציות לא הגייניות, אבל עדין כתוב אותן.

3. ניתן שמות למצבים, כאשר מצב מוגדר ע"י קיבוע ערכיהם של כל פלטי רכיבי הזכרון, במקרה זה יש לנו רק שני-DFFים

שמות המצבים	$Q_1$	$Q_2$
A	0	0
B	0	1
C	1	0
D	1	1

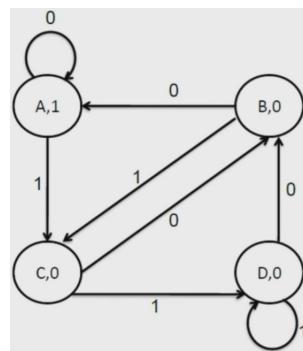
ועלシו נוכל לחשב את הפלט של המעלג בהינתן הקלט והמצב הנוכחי בטבלת המצבים הבאה (PS המצב הנוכחי ו-NS המצב

הבא)

	NS			
PS	In=0	In=1	Out	
A	A	C	1	
B	A	C	0	
C	B	D	0	
D	B	D	0	

4. עתה נבנה אוטומט, כאשר נctrיך להחליט האם לבנות אוטומט Moore או Mealy, ובשלב הראשון עוד יוכלו לשים לב שהמעגל

הelogio שמנדר את הפלט תלוי רק בזיכרון ולא בקלט ולכן נסתפק באוטומט Moore



5. נתח אוטומט, כshima שמשמעותו אחרת של דבר היא באילו מקירים המכונה פולטה 1. השתמש בסדרת בוחן, כולם טבלה עם שלוש שורות: מצב, פלט, והקלט שאיתו נבער למצב הבא בשורה. נdag שהמעברים בין כל שני תאים סמוכים בשורת המצבים יכסו את כל המעברים הקיימים באוטומט (כל הקשתות) לפחות פעם אחת. אין חשיבות לסדר המעבר.

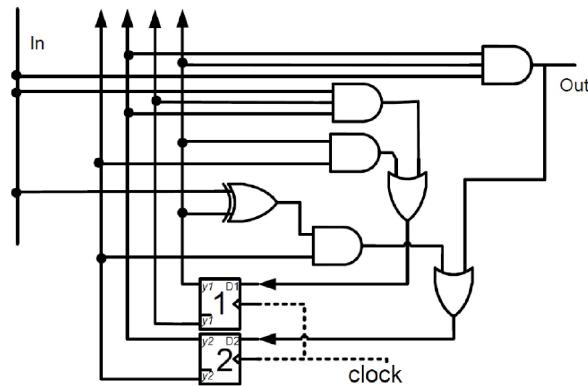
In	0	0	1	0	0	1	1	0	1	0	1	0	0	0
State	A	A	A	C	B	A	C	D	B	C	B	C	B	A
Out	φ	φ	1	0	0	1	0	0	0	0	0	0	0	1

כאשר שני הפלטים הראשוניים הם  $\Phi$  כי כל עוד לא עברו שתי ייחידות זמן, לא יוכל לדעת מה הפלט של רכיב הזכרון השני (כי הקלט שלו מוגדר רק אחרי המחוור הראשון) שMOVABLE לפלט ולכן הפלט לא מוגדר.

עתה נחפש את ה-1-ים בפלטים, ונשים לב שכדי לקבל 1 בפלט, צריך שהקלטים בשני מחזורי השעון הקודמים יהיו 0, וזהו כלל השינוי! נשים לב שהקלט בזמן המחוור הנוכחי לא משנה כי מדובר במצב Moore ולא Mealy.

איך נדע שחייב רק את שני הביטים שלפני המחוור הנוכחי ולא יותר או פחות? אפשר לבדוק כללי שינויי מרכיבים יותר (שמורכבים משלושה ביטים לדוגמה) ולשים לב שהם לא מתקיים, כי אפשר לקבל 1 גם עם 100 (זה קורה בסדרת הבדיקה) וגם באמצעות (ע"י היישאות ב- $A$  שוב ושוב). הכלל האופטימלי וה邏輯י הוא זה שמענין אותנו.

#### דוגמה נתון המעגל הבא



כאשר הפלט היחיד הוא Out (והחצאים למעלה חסרי משמעות).

1. ניצג את הפלטים והקלטים לזכרון. ובנוסף  $D_2 = y_2 \cdot y_1 \cdot In + y'_2 \cdot (y_1 \oplus In)$  ו-  $D_1 = y_1 \cdot y'_2 + y_2 y'_1 In$ .

2. מהצבת הערכים קיבל את טבלת המצבים

	In=0			In=1			
$y_1$	$y_2$	$D_1$	$D_2$	Out	$D_1$	$D_2$	Out
0	0	0	0	0	0	1	0
0	1	0	0	0	1	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	0	1	1

3. נבחר שמות למצבים (קיוב פלטי הזכרון) כרגע

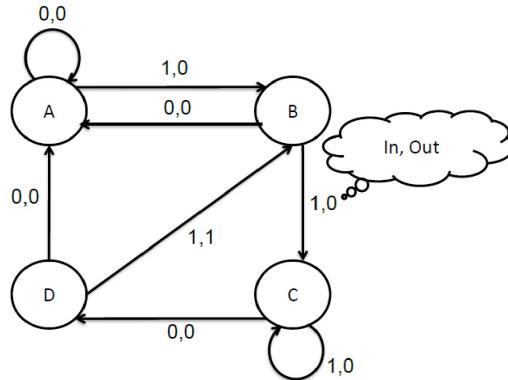
שמות המצבים	$Q_1$	$Q_2$
A	0	0
B	0	1
C	1	0
D	1	1

ועתה באמצעות הצבה של השמות בטבלת המצבים נקבל כאשר הפלט כן משפיע על הפלט, כלומר מדובר במצב Moore

	NS,Out	
PS	In=0	In=1
A	A,0	B,0
B	A,0	C,0
C	D,0	C,0
D	A,0	B,1

4. כדי לבנות את האוטומט נצטרך עכשו לשימוש באוטומט Mealy, ככלומר שעל הקשת נכתוב איזה קלט מעביר אותו למצב

הבא ואיזה פלט הוא מספק לנו



5. עתה נרשום סדרת בוון שתראה אותנו הדבר, רק שהפעם בחיפוש אחר כל השינוי נצטרך להתחשב גם בערך הנוכחי של הקלט

In	1	1	0	1	1	0	1	0	1	1
State	A	B	C	D	B	C	D	B	A	B
Out	φ	φ	0	1	0	0	1	0	0	0

כאשר 1 מתקבל רק כאשר הקלט בזמןים  $t+3, t+2, \dots, t+1$  היה .1, 1, 0, 1.

## סינטזה של מעגלים סינכרוניים

בහינתן אפין, נרצה למשתמש מעגל סינכרוני שמקיים את הדרישות. נשתמש בסכמה הבאה :

1. בניית אוטומט בהתאם לדרישות.

2. טבלת מצבים.

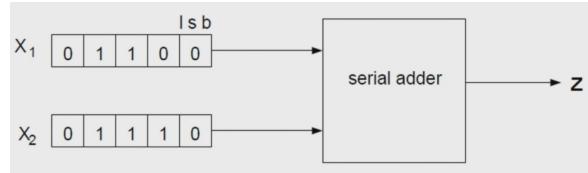
3. קידוד מצבים ובחירה סוג FF.

4. טבלת מעברים ופלט.

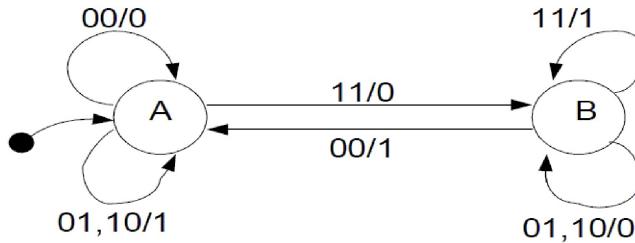
5. הגדרת פ' הכניסות של רכיבי הזיכרון ויציאת המעגל.

6. בניית המעגל.

**דוגמה** נרצה לבנות מסכם בינארי טורי, כולם מוגדר שמקבל קלטים  $x_1, x_2$  וזמן  $t_i$  מחשב את הביט  $h-i$  (מכיוון ה-LSB) בסכימת המספרים המתואימים על הסרט הנע של  $x_1, x_2$  (כל מהזור מקבלים שני ביטים חדשים, שכайлו מותוספים כ-MSB למספרים שאנו סוכמים).  
ראו איור (כל פעם סוכמים את הביטים המתואימים, כמוגן עם נשא מהסכימות הקודמות)



- ראשית נבנה אוטומט שמקיים את הדרישות, כאשר הקלט (מיימן ל-/-) הוא שני הביטים  $m-1 x_1$  ו- $x_2$  בהתאם. נבחר שהמצבים שלנו ייצגו האם יש לנו נשא מהחישוב הקודם, וכך נשמר את המידע הזה בזיכרון למשעה.  $A$  מייצג מצב שאינו בו נשא מהחישוב הקודם ו- $B$  מייצג מצב שבו יש נשא מהחישוב הקודם. בכל פעם נחשב את הסכימה יחד עם הנשא (אם יש כזה), ונבחר מה הפלט של התוצאה ובנוסף האם יש לנו נשא ונעביר מצב בהתאם.



- نبנה טבלה מצבים, ש מכילה גם את הפלט במצב אליו עוברים כי הפלט תלוי בקלט הנוכחי כי זו מכונת Mealy

		NS (Next State), z (Output)			
		Input			
		00	01	11	10
PS (Present State)	A	A,0	A,1	B,0	A,1
	B	A,1	B,0	B,1	B,0

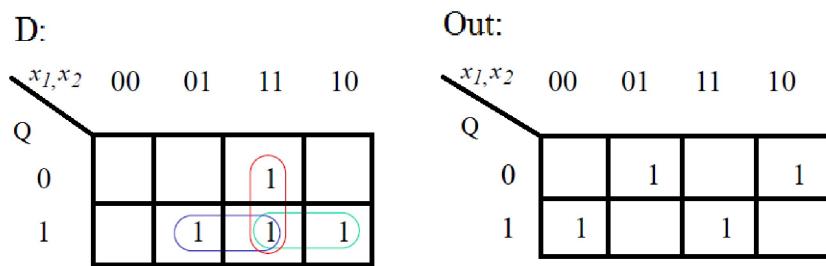
כאשר נשים לב שהקלטים לא מסודרים לקסיקוגרפיה אלא כמו בטבלה קרנו!

- קידוד המצבים דורש בחירות ייצוג בינארי למצבים, וכאשר יש שניים זה די קל - מספיק ביט אחד שייצג את  $A$  כשהוא 0 ואת  $B$  כשהוא 1.
- נחליף את  $A$  ו- $B$  בטבלה בביטויים המתואימים להם ונקבלת טבלה מעברים ופלט ש מכילה רק ביטים

		NS (Next State), z (Output)			
		Input			
		00	01	11	10
PS (Present State)	0	0,0	0,1	1,0	0,1
	1	0,1	1,0	1,1	1,0

5. נפצל את הטבלה לשתי מפות קרנו (אחת ל-D, הקלט לשילוב המצב הבא ואחת לפולט) וככשה את הטבלה כמו שעשינו

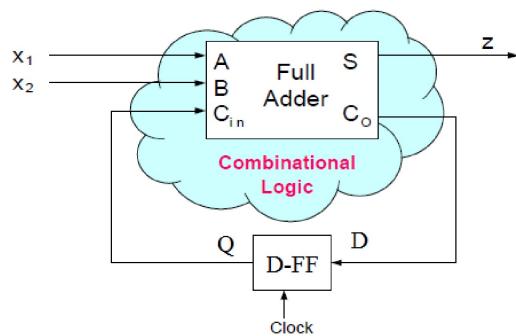
בתרגול 2



$$\text{Out} = Qx'_1x'_2 + Q'x'_1x_2 + Qx_1x_2 + Q'x_1x'_2 \quad \text{D} = x_1x_2 + Qx_1 + Qx_2$$

6. הסטודנטית הערנית תשים לב שהחישובים האלה הם בדיק הפלטים של FA ולכן (באופן שדי מותאים לאפיון) נקבל שימושו את המעגל

כך



## שבוע VII | מעבד Single-Cycle

### הרצאה

**הערה** למה שלא נשמר את כל הרגיסטרים לצד של הקוראת או הנקראת? כיזה מאד בזבזני בין היתר כי לא תמיד נדרש את כל הרגיסטרים שמורים. מה שענייפ הוא שכל צד ישמור חלק מהרגיסטרים שלו אצלנו ויישמר רגיסטרים אחרים בשילוב אחר. בצורה כזו לא נדרש

תמיד לשמר הכל, אלא רק מה שאחננו צריכים לשמר (לדוגמה אם הקוראת השתמשה ב-\$3\$ ולא צריכה אותה יותר אחרי הקריאה לפ', היא לא צריכה לשמור אותה עצמה).

**הערה** ישן פקודות שלא קיימות ב-MIPS אבל שניות למימוש באמצעות פקודת אחת שכן קיימת, לדוגמה `sh $d, $s` זה ניתן למימוש ע"י `nor $d, $s, $0` והאSEMBLER יוכל לתרגם את המקרים הפרטיים האלה.

## קידוד פקודות MIPS

כל פקודת מקודדת באמצעות 32 ביטים, ולא יכולה להיות בגודל דינامي, ולכן נדרש לנחל את תקציב הפקודות שלו ( $2^{32}$  סה'כ) באופן יעיל.

- פקודות עם שני רגיסטרים אופרנדים מקודדת ב-Type R.

opcode	rs	rt	rd	shamt	funct
--------	----	----	----	-------	-------

כאשר opcode הוא מזהה הפעולה (כל פקודות ה-R הן 0) שגודלו 6 ביטים ; rs ו-rt הם אינדקסי הרגיסטרים של האופרנדים ו-rd אינדקס רגיסטר היעד (5 ביטים כל אחד) ; shamt מספר הביטים ל-shift כמשמעותו בפקודה shift (5 ביטים) ; funct ו- 1 שגדיר איזו פעולה בדיק נבצע (לדוגמה ל-add ו-sub יהיו כאן ערכים שונים). funct תופס 6 ביטים כך שיש לנו לכל היותר 64 פקודות מסווג R. נשים לב שככל פקודה תופסת הרבה מילימ (5 ביטים לכל אינדקס רגיסטר, כולל 32,000 כולם שמייצגות פקודות סכימה כלשהי).

- פקודות עם רגיסטרים ו-immediates מקודדות ב-Type I.

opcode	rs	rt	immediate
--------	----	----	-----------

כאשר rt הוא אינדקס רגיסטר ה-operand השני יחד עם ה-immediate (שהוא 16 ביטים) ו-rs הוא רגיסטר היעד. הרוב המכרייע של opcodes משמש פקודות מסווג זה.

- פקודות קפיצה מקודדת ב-Type J.

opcode	immediate
--------	-----------

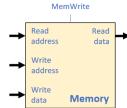
כדי שנוכל קופץ למגוון כתובות בזיכרון ( $2^{26}$  כתובות), כאשר רק j-ojal מקודדות כך (שאר הקפיצות כוללות immediate).

**הערה** כתובות R-Type הן חci זולות, כי אפשר להכניס הרבה סוגים שונים של פקודות באמצעות שדות ה-opcode וה-funct. אם נרצה עוד פקודה מסווג R-Type, נפנה מיד לפקודה Miyotra Type I-Type, שתופסת שימושית יותר מקום בעבר פחות פקודות, ונשבע את ה-opcode שלה לצורך פקודות R-Type.

## ביצוע פקודות MIPS

מעבד הוא בסה'כ מכונת מצבים ענקית, כאשר הבדיקה הברורה בין הלוגיקה למצב/זיכרון היא קצר יותר מרכיבת כי הפקודה רצתה איפשהו באמצעות (לפni ואחרי מצבים). עם זאת, ביצוע פקודה בסופו של דבר מעביר אותנו מצב (משנה את ה-PC וכו'). כזכור הזכרנו אמנים מאורגנים בביטחון, אבל הקריאה ממנו היא ביחידות של ארבעה בתים (מילה), וכך גם PC גדול בקביצות של 4.

ברמת הארכיטקטורה הנוכחית, רכיב הזכרון מציג את הממשק הבא



כאשר כתובות הקריאה והכתיבה הן 32 ביט, הוא המילה שנרצה לכתוב לכתובת הכתיבה (אם נכתב), ו-*MemWrite* הוא ביט שקובע האם נכתב או לא. הפלט הוא *Read data* שהוा המילה שנמצאת בכתובת הקריאה. משיק זה מאפשר לנו לספק תמיד את כל הקלטים, אבל לכתוב לזכרו רק אם אנחנו צריכים, וזה יקל علينا בהמשך בימוש המעבד.

בדומה, רגיסטרים הם אוסף-Flip-Flop.

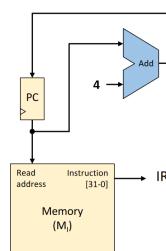
**הערה** כשנסמן חץ באופן הבא ← אל תוך רכיב ומתחתיו מספר, הכוונה שיש לנו בסם שמורכב ממספר חוטים (מספר שכתנו) ולא רק ביט אחד.

## שלבי הרצת פקודה

- קראית הפקודה (fetch) : המעבד מספק כתובות (ששמורה ב-PC) לזכרו ומקבל את תוכנה, שהיא קידוד הפקודה שצריך להריץ.
- פענוח הפקודה (decode) : להבין איזו פקודה צריך לבצע ואילו רגיסטרים היא דורשת. בנוסף נזיה את ה-PC בהתאם לפקודה (קפיצה לכתובת אם מדובר ב-branch או jump ואחרת איןקרמנטן).
- ביצוע הפקודה (execute) : לבצע את הפעולה המתמטית.
- גישה לזכרו (memory access) : נדרש כשהפעולה ניגשת לזכרו (load יקרא ו-store יכתוב).
- כתיבה חוזרת (write back) : לכתוב את תוצאות החישוב או ה-load או ה-branch. השלב הזה למעשה משתנה את מצב המכונה לקראת הפקודה הבאה.

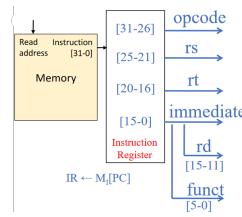
## מימוש השלבים

- המימוש דורש חיבור בין רגיסטר ה-PC לזכרו וגם איןקרמנטציה, כמו למשל יראה כך:



כאשר IR הוא רегистר שמכיל את הפקודה הנוכחית, כי  $IR = M_I[PC]$  (עבור  $M_I$  זיכרו הפקודות).

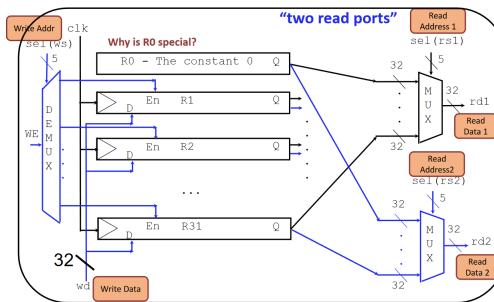
- פענוח: נפצל את הביטים ברегистר הפקודה (IR) למשמעויות הרלוונטיות שלם (ראו איור)



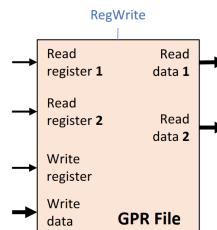
ובנוסף לביא את הרגיסטרים הנדרשים לפועלה, שממוקמים ב-FF-ים שבו הרגיסטר הראשון הוא בעצם קבוע 0 ולא רכיב זיכרונו וה-31 האחרים הם רכיבי זיכרונו אמתיים. כדי לקרוא מהם מידע, צריך להשתמש ב-Mux שבוחר אחד מבין 32 רגיסטרים ומוציאה את התוצאה החוצה. הכתובת שנקרה תלואה כموון ב-rt, rs ו-rd לפי הצורך. ה-Mux הוא בעצם מערך של 32 Mux-ים שכל אחד בורר בין 32 אפשרויות.

כל אחד מה-32-Mux-ים ניתן למימוש עם 32 NAND-ים ועוד NAND עם 32 כניות, כלומר למעלה מ-1000 שערים לכל לוגיקת הקראיה מה-File.

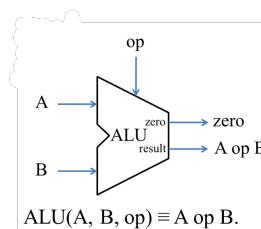
כדי לכתוב בהמשך לרוגיסטרים (בשלב ה-DeMux) נחבר את ה-D-FF-ים ל-32-RAM (memory access) שnbrור אליו את הרגיסטר הנוכחי אליו אנחנו רוצים לכתוב אליו באמצעות בית WriteEnable האמור.



כל הרכיב באירור נקרא יחד ה-Register File, והוא מציג את הממשק הבא

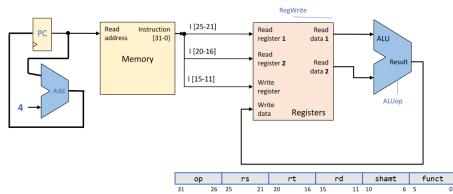


• **ביצוע:** הרכיב האחראי על החישוב האריתמטי נקרא Arithmetic Logic Unit (ALU), שמציג את הממשק הבא



כאשר הפלט zero הוא בית (דגל) שערךו 1 אם "ס פלט החישוב הוא 0 (שימושי מאד לקפיצות מותננות שוויז/אי-שוויז).

עבור הרצת פקודת R-Type, כבר יש לנו את כל הרכיבים הנדרשים והתהליך יראה כך (cronologית משמאל לימין).



**הערה** המוקם היחיד שבו נדרש לעליית שעון כדי להתקדם הוא בהתקדמות PC, שכן עד לחישוב התוצאה ב-ALU אין שום לוגיקה שאינה ציורפית, וכך לכנתוב את המידע לרגיסטרים צריך עוד עליית שעון. לעומת זאת מכוזר אחד לכל פקודה (עד כדי hold time ו-setup time).

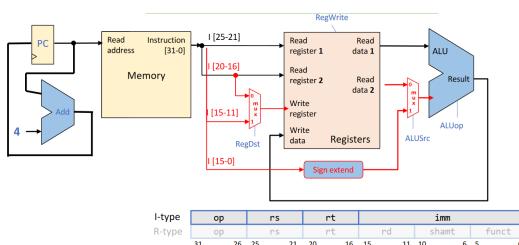
Single Cycle (time), מכאן בא השם

כדי לממש פקודת I-Type, נבצע את אותה הפעולה של R-Type עם כמה שינויים קטנים:

1. ניקח את תוכן ה-imm מהפקודה ונעשה לה sign extend (הרחבת המספר מ-16 ביטים ל-32 ביטים כפי שראינו בעבר) ולכן נctruck ל לקרוא רק מכנתוב אחת ב-I-Type (נספק זבל ל-bus כתובת הקリアה השנייה). את הבחירה במה להשתמש נמשם באמצעות mux שבורר לפי סוג הפקודה.
2. כתובת הכתיבה צריכה mux לפניה בדומה לנ"ל כי ב-R-Type כותבים לרגיסטר rd (האינדקס השלישי בקידוד) ואילו ב-I-Type מדובר ברגיסטר rt.

**הערה** בגלל ש-signature opcode, ה-imm funct כודב להגיד לנו מה אמ' מדובר ב-I-Type או R-Type כדי שנוכל לפרש את הביטים נכון.

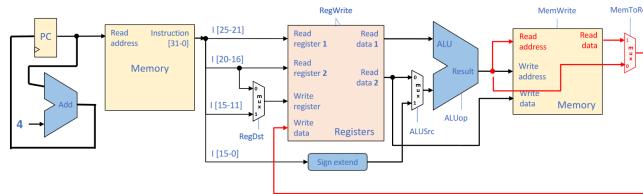
האיור הבא מציג מעבד שיכול להריץ פקודות I ו-R מותמיות, עם אינדקסים הביטים בקידודים למיטה



: השלב הזה חיוני למימוש כתיבה וקריאה, כאשר בשתי הפקודות מעתיקים את הערך של Reg [IR [rt]] ל-Reg [IR [rs]] memory או להפוך, בהתאם. כדי למשם את הפעולה נשתמש ברכיב הזכרון Data Memory שמקבל Addr קלטי ו-Din (32 ביט) ו-WE (ושעון WE=0) ו-WE (WE=0) (32 ביט) כאשר אם Dout=0 מכיל את תוכן הקリアה ואחרות הפלט לא מעניין אותו.

בשביל הקריאה וגם הכתיבה ניתן ל-Data Memory את כתובת הקリアה והכתיבה שהיא פלט ה-ALU (שיסכום את הרגיסטר יחד עם ה-imm לכדי כתובת אחת). תוכן הכתיבה יהיה פשוט תוכן הרגיסטר השני שקרהנו מהרגיסטרים. פלט הזכרון יבורר יחד עם פלט ה-ALU לפי האם אנחנו קוראים מהזיכרון (פלט הזכרון) או מבצעים חישוב מתמטי נתו (פלט ה-ALU).

האיור הבא מדגים את המעבד שבנו עד כה, שיכל להריץ כל פקוד מסוג R או I



## תרגול

### דוגמה הפקודה

100011 00110 00101 0000 0000 0000 0000

מבצעת (\$5, 7(\$6) כאשר 35 הוא opcode של קריאה מהזיכרון, rt, rs כרגיל מקודדים את אינדקס הרגיסטר ממנו נקרא ואילו כתוב ולבסוף ה-*mem* שהוא ההיסט מ-*rs* שנutan את הכתובת.

**הערה** J-type פקודת כתובות הפקודה ביחידות של מילה במקומות הביתי, ולכן אפ"פ שיש לו  $2^{28}$  ביטים, נוכל ליעց מרחב זיכרון בגודל  $.2^{28}$

**דוגמה** את השורה  $A[i] = g + A[j]$  נממש באמצעות שתי פקודות: קריאה של  $A[6]$  לרגיסטר זמני והשנת ההפרש ברגיסטר נוסף.

### דוגמה נבית בקוד הלולאה הבאה

```
do {
    g = g + A[i];
    i = i + j;
} while (i != h);
```

אותו נוכל לכתוב כפסאודו-קוד אסמבלי

```
Loop: g = g + A[i];
      i = i + j;
      if (i != h) goto Loop;
```

מכאן נקצה רגיסטר לכל אחד מה משתנים, לדוגמה  $g = \$s1, h = \$s2, i = \$s3, j = \$s4$  ונשמר ב- $\$s5$ . סיום

נוכל לתרגם זאת לשפת אסמבלי אמיתית

```
Loop: sll $t1,$s3,2          # $t1 = 4*i
      add $t1,$t1,$s5        # $t1 = addr of A[i]
      lw $t1,0($t1)           # $t1 = A[i]
      add $s1,$t1,$t1          # g = g + A[i]
      add $s3,$s3,$s4          # i = i + j
      bne $s3,$s2,Loop         # go to L1 if i!=h
```

כאשר שלושת השורות הראשונות הן עצם העניין: כדי לקרוא מהמערך, נחשב את היחסט  $i$  מראשיתו של המערך באמצעות הכפלת הרגיסטר ב-4 (כי הכתובות כשהן לא-*mem* הן ביחידות של בתים), נוסיף אותו לבסיס של A ונטען את המילה מהזיכרון. משם סתם עושים אריתמטיקה עד לתנאי הלולאה שkopץ חזרה להתחלה רק אם  $i \neq h$  לאחר השוואת הרגיסטרים המתאים להם ( $i \neq h$  משמע  $i \neq h$ ).

**דוגמה** switch-case אפשר למשה באופן הבא, והימוש כMOVED שקול לשרשור ארוך של if-elseים.

כל הסגר ב-switch-case : לפניו פקודות התוכן של החסגר, נסייף :

- פקודת addi שתחשב את ההפרש בין המשתנה למועדן ההשוואה בסהgor
- פקודת bne שתבדוק האם ההפרש שונה מאפס ואם כן תקופז לתוויות של החסגר הבא (אחרת לא נקופז ונשאר להריץ את תוכן החסגר הנוכחי).

לאחר סיום תוכן החסגר נסייף פקודת j לסוף ה-switch-case צולו, כך שלא נריץ בטיעות הסגרים אחרים (בדומה לפקודת ה-break ב-C).

**דוגמה** את ההתניה if(g < h) goto label בבחירה \$\$s0, \$\$s1 בבחירה (בהתאמה)

```
slt $t0, $s0, $s1 # g<h ? 1 : 0  
bne $t0, $0, label
```

**דוגמה** נפענח את הקוד הבא

```
begin: addi $t0, $zero, 0  
        addi $t1, $zero, 1  
loop:   slt $t2, $a0, $t1  
        bne $t2, $zero, finish  
        add $t0, $t0, $t1  
        addi $t1, $t1, 2  
        j loop  
finish: add $v0, $t0, $zero
```

ההערות ליד הין הפענוח (מעבר לפסאודו-אסמבלאי)

```
begin: addi $t0, $zero, 0 # $t0=0  
        addi $t1, $zero, 1 # $t1=1  
loop:   slt $t2, $a0, $t1 # If n<$t1 then $t2=1 else $t2=0  
        bne $t2, $zero, finish # If n<$t1 then goto finish  
        add $t0, $t0, $t1 # $t0 = $t0 + $t1  
        addi $t1, $t1, 2 # $t1 = $t1 + 2  
        j loop # goto loop  
finish: add $v0, $t0, $zero # $v0 = $t0
```

ועתה נבדוק מה הקוד עושה באמצעות טבלת מעקב לדוגמה, ונקבל שזה סוכם את כל המספרים האיזוגיים בין 1 ל-n.

**דוגמה** נמיר את הקוד הבא מ-C לאסמבלי MIPS כאשר הערכים ההתחלתיים של a ו-b נמצאים ב-\$a0 ו-\$a1 בהתאם, וכתוות הבסיס של \$.\$s0 ב-C.

```
int mult(int a, int b) {  
    int value;  
    value = 0;  
    if (a < 0) {  
        a = -a;  
        b = -b;  
    }  
    while (a != 0) {  
        value += b;  
        a--;  
    }  
    A[4] = value;  
}
```

את value נשים ב-\$t0, ונעתיק את ערכו של a ל-\$t1 ושם נבצע לו דיקרמןט. בלאה נקופז לסוף אם \$t1 הוא 0 ואחרת נריץ את תוכנה. לבסוף נכתוב את תוכנו של \$t0 = value לミילה החמישית במערך. הקוד צולו הוא

```
Begin: add $t0, $zero, $zero
       slt $t1, $a0, $zero
       beq $t1, $zero, Loop
       sub $a0, $zero $a0
       sub $a1, $zero $a1
Loop:  beq $a0, $zero, Finish
       add $t0, $t0, $a1
       sub $a0, $a0, 1
       j Loop
Finish: sw $t0, 16($s0)
```