7/4/03

תרגיל 2 בעיבוד אות I

2.1.2 • מהו ההספק של - $S(t) = A \cdot \sin wt$?

האנרגיה היא על כל הציר ולכן ההספק הוא של טווח זמן של מחזור.

לפי השאלה אנו צריכים לבדוק על מחזור אחד בלבד.

• יש שתי דרכים:

דרך א' -

$$A^2\int_0^T \sin^2 wt\,dt = A^2\int_0^T \cos^2 wt\,dt = \frac{1}{2}A^2\int_0^T \sin^2 wt + \cos^2 wt\,dt$$

$$A^2\int_0^T \frac{1-\cos 2wt}{2} = A^2\int_0^T \frac{1}{2}dt + \underbrace{\int_0^T \cos 2wt\,dt}_{\text{מספר שלם של מחזורים ולכן שווה ל-0.}} = A^2 \cdot T/2$$

$$\lim_{T\to 0} \frac{1}{T}\int_0^T S^2(t)dt = \int_0^T A^2 \sin^2 wt\,dt = A^2/2$$

$$RMS = \sqrt{A^2/2}$$

ובדומה בדיסקרטי - אותו מושג ואותו דבר.

2.3.1

• אם יחס התדירויות בין מחזורי הוא רציונלי אזי הסכום הוא מחזורי.

• אם יחס " " " " שלם אזי זה אולי מינן.

2.3.2 $\sin\alpha t + \sin\beta t$

אם היחס בין α ו-β הוא רציונלי הם מחזוריים יחסית,

אחרת שני הסינוסים לא יסתכמו.

2.4.1

$$S_n = C_1 Z^{-1} S_n + C_2 Z^{-2} S_n$$

$$S_n = C_1 S_{n-1} + C_2 S_{n-2}$$

$$S_n = \boxed{(C_1 Z^{-1} + C_2 Z^{-2})} S_n$$

$S_n = \sin wn$ כאשר ← אנו צריכים אלגברי לפתור את המערכת:

$$\sin(wn) = C_1 \sin(w(n-1)) + C_2 \sin(w(n-2))$$

$$\sin(wn) = C_1(\sin wn \cos w - \cos wn \sin w) + C_2(\sin wn \cos 2w - \cos wn \sin 2w)$$

יש לנו 2 משוואות עם 2 נעלמים ונוכל לפתור את המשוואות

$$\begin{cases} C_1 \cos w + C_2 \cos 2w = 1 \\ C_1 \sin w + C_2 \sin 2w = 0 \end{cases}$$

$C_2(\cos^2 w - \sin^2 w)$

מהמשוואה השנייה מקבלים: $C_1 = -2C_2\cos w$

נציב במשוואה הראשונה:

$$-2C_2\cos^2 w + 2C_2\cos^2 w - C_2 = 1$$

$$\boxed{C_2 = -1} \Longrightarrow \boxed{C_1 = +2\cos w}$$

2.5.3 כיצד לבטא את השימוש בסכום (Σ) כאשר בפועל יש לנו רק הזזה בזמן (Z^{-n}) והלם ($\delta_{n,0}$)?

$$X_n = \sum_{m=-\infty}^{\infty} X_{-m}\,\delta_{n,-m} = \sum_{m=-\infty}^{\infty} X_{+m}\,Z^{+m}\,\delta_{n,0}$$

2.6.3 לא בהכרח: כל פונקציה רציפה $f(x)$ ניתן לפתח בטור הבא:

$f(x) = \sum_{m=0}^{\infty} a_m x^m \Longrightarrow$ ובנוסף ה-a-ים הללו הם יחידים.

אנו רואים שזה לא רק עבור פולינומים אלא לכל פונקציה רציפה.
למרחב הנ"ל יש בסיס (כמו לכל מרחב) ובלת"ז של הפולינומים.

לא אורתונורמלי כיוון שהוא אפילו לא אורתוגונלי (אחד משני התנאים).

זה מרחב וקטורי (מרחב מכפלה פנימית שהוא גם) יש מכפלה סקלרית שהיא $\int_{-\infty}^{\infty} f(x)g(x)\,dx$.

$$\int_{-\infty}^{\infty} x^n x^m\,dx = \int_{-\infty}^{\infty} x^{n+m}\,dx = \left[\frac{x^{n+m+1}}{n+m+1}\right]_{-\infty}^{\infty}$$

אם אחד זוגי והשני אי זוגי הם אורתוגונליים. אחרת, אם שניהם זוגיים או שניהם אי זוגיים הם לא אורתוגונליים.

2.6.4 כן. אוסף סינוסים המתאימים בזמנים אל הנתון.
כי יש 4 סינוסים, וכל פונקציה $\emptyset$ נקודה סוף.

תורת הגרפים ב- DSP

ב- DSP אנו משתמשים באלגוריתמים שונים ממבני-נתונים.

programming = algorithms + data structures

במבני-נתונים רגילים אין התייחסות לבסיס הנתונים. ב- DSP יש התייחסות לנתונים ולכן אנו חוזים באלגוריתם זהו.
תמיד ניתן לבנות פתרון ע"פ הדיאגרמה צריך לקבל אלגוריתמיקה.

- ב- DSP יש לנו אוסף נתונים מגרף, כאשר בין הנתונים יש קשר שיש לו כיוון.

$x \xrightarrow{1} \bullet \longrightarrow \bullet$ - x היה מקדם 1 לאחר אין צורך לציין.

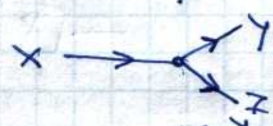
$x \longrightarrow y \iff y_n = x_n \ \forall n$

אנו רואים פונקציה מורכבת.

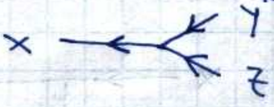
$z = f(x)\ ,\ y = g(z) \implies y = g(f(x))$

סימנים מיוחדים:

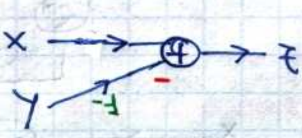
$x \xrightarrow{a} y$ - זהו ריבוי הכל קבועה מכפלה. $xa = y$ $\quad y = ax$



- $x = y = z$



- לזה אין משמעות.



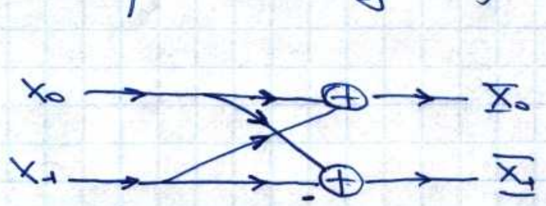
- חיבור בין שני signal-ים. $z = x + y$

$z = x - y$

$z = x - y$

הכפלה בין שני signal-ים $z = x \cdot y$

למשל פרפר הלוי:



$X_0 = x_0 + x_1$
$X_1 = x_0 - x_1$ } DFT

דיאגרמה קלאסית של פרפר.

אנו רואים שזהו DFT מסדר שני (N=2).

$$\boxed{X_k = \sum_{n=0}^{N-1} x_n W_N^{nk}} \qquad W_N = e^{-i\frac{2\pi}{N}}$$

נלמד מאוחר יותר את ה-DFT ואת ה-FFT לעומק.

אופן סימון:

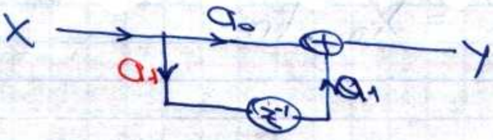
$Y_n = X_{n-1} \quad \forall n$

נראה מסנן כללי נוסף:

$$Y = X - Z^{-1}X = \Delta X$$

זהו המסנן הפשוט הראשון - הקירוב לנגזרת (תכונה חשובה של מסנן).

מסנן מסדר ראשון:



$$Y_n = a_0 X_n + a_1 X_{n-1}$$

זהו ה-FIR מסדר ראשון הפשוט ביותר.

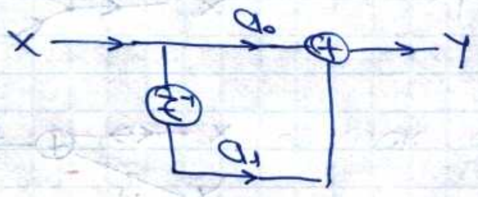
$$Y_n = a_0 X_n + a_1 X_{n-1}$$

אנו רואים שהזזות ומכפלות קובעות את סדר משוואה קומוטטיבית

- נראה כעת שניתן לצייר את אותו הדבר:

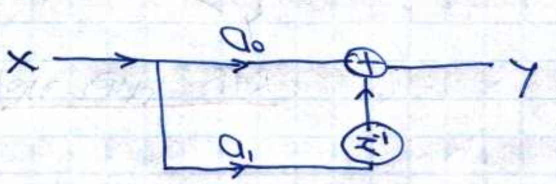
4 צורות קנוניות:

a)

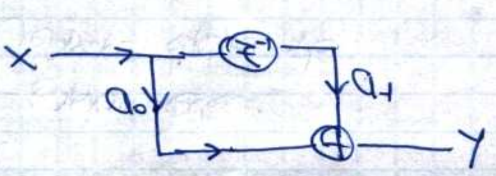


c)

b)

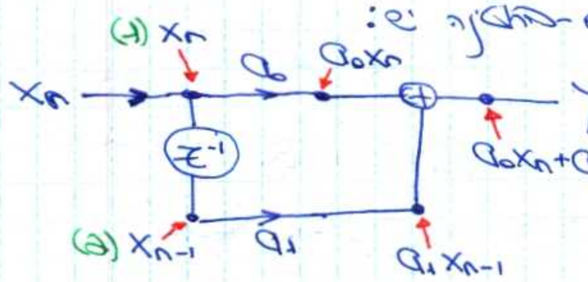


d)



אין פה משוב!!

לכל a, יש נתמש בצורה של אלגוריתם-בתבנית יש:



המסנן נראה כך לכל שני

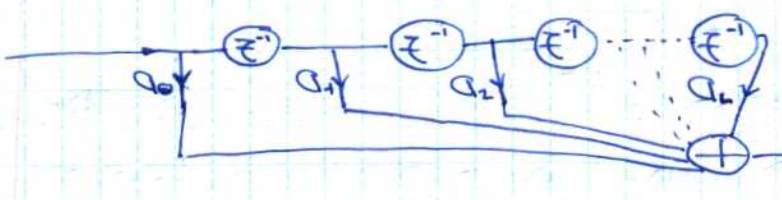
צריכים לזכור את המצב של הזיכרון. $a_0x_n+a_1x_{n-1}$

x_n נכנס ולכן ב- (1) יש x_n.

באופן כללי נצייר נקודה רק בנקודות בהן צריך לשמור את הערך אחרי התבנית הבאה, ולכן הנק' היחידה שמסומנת תהיה (2) ולא כי השאר לא נצייר באופן מפורש.

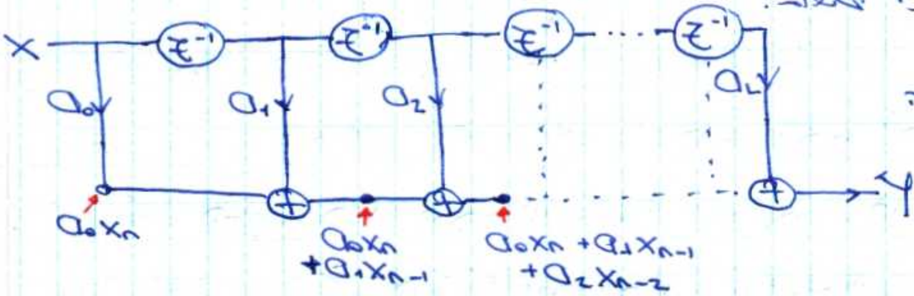
מימוש FIR כללי:

$$y_n = a_0x_n + a_1x_{n-1} + a_2x_{n-2} + \dots + a_Lx_{n-L}$$



לא ניתן לעשות tapped delay

מה שכן ניתן לעשות:

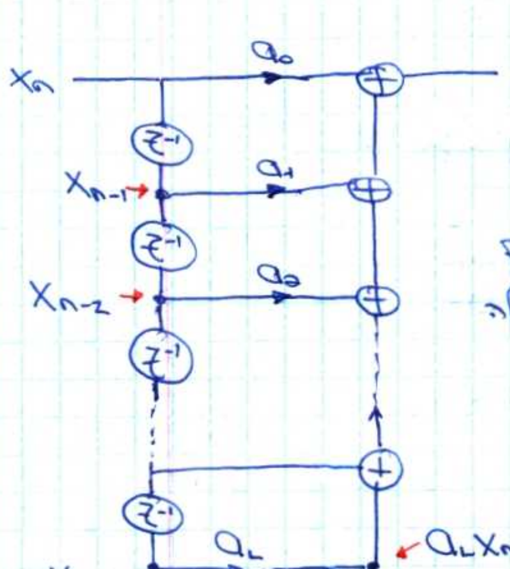


סכום מההתחלה עד הסוף

כאשר שרשרנו היה צריך לקחת גם את הערך אחרי השלב הקודם אבל רצינו להיפטר מסכימים. ישנם L-n נקודות זיכרון בדיוק.

ניתן לראות כאן כשרשור של אחד מהבלוקים שראינו קודם.

מתקיים כי קריאה לרוטינה מספר פעמים.



סיכום מהסוף להתחלה

6 פונקציונלית כמו אותו הפילטר שכתבנו את a.

[illegible]

הנקודות מסכימות מהסוף להתחלה.

ההבדלים בין שני הציורים יכולים להיות נומריים.

28/4/03

נושא חדש - המעבד של עיבוד נתונים

- האפליקציות של DSP הן אפליקציות של Realtime ולכן צריך להשתמש במעבד מיוחד (ולא במעבד רגיל).
- Realtime רך - לעמוד בקצב שמגיעים הנתונים החיצון בקצב ממוצע. לא מדובר על המקרה הגרוע ביותר.
- Realtime קשה - יש הגבלה לגבי המקרה הגרוע ביותר למרות שהקצב הממוצע יכול להיות שווה.
- למעבד כזה יש ממשקים מיוחדים לעולם החיצון.
- מעבד DSP יודע לבצע קונבולוציה בצורה מהירה.

$$Y_n = \underbrace{\sum_i a_i X_{n-i}}_{FIR} + \underbrace{\sum_m b_m Y_{n-m}}_{IIR}$$

MAC - אנו מכפילים ומוסיפים לסכום הקודם שהיה.

יש הרבה פעולות ב-DSP שהן MAC. הייחוד ב-DSP הוא שהמעבד יודע לבצע פעולת MAC בזמן אחד.

- יש לו סט פעולות מצומצם יותר מעבדים אחרים, אך מצד שני יש לו פעולות שאינן קיימות בהם ובהן הוא מתמחה.
- מעבד DSP הוא הרבה יותר זול, ההספק של הרבה יותר נמוך (פחות אנרגיה), הוא לא מתחמם ויש לו תקשורת טובה עם העולם החיצון.
- <u>כיצד המעבד עושה MAC</u>?

איך כותבים MAC:

$$Y_n = \sum_{l=0}^{L-1} a_l X_{n-l} \Longrightarrow$$

```
a[]
x[]
y ← 0
ctr ← L
loop
    dec ctr
    if ctr<0 goto xxx
    update xptr
    load x, xptr
    update aptr
    load a, aptr
    z = x·a
    y ← y+z
    goto loop
xxx
```

- בפקודה פשוטה כזאת יכולה לקחת 30 שעונים.
- יש לנו בזבוז של פעולות. אנו רוצים לבצע את כל הפקודה הזאת בפעולה אחת בלבד.

- לכל המעבדים יש איזשהו מנגנון לאיפוס הנתונים בהתחלה ולביצוע הלולאה ולכן זה לא יקח זמן מה- ALU.

כל מה שנשאר לנו בתוך הלולאה:

loop
1. update xptr
2. update aptr
3. load x ← *xptr
4. load a ← *aptr
5. fetch op (mult)
6. decode op (mult)
7. mult z ← x·a
8. fetch op (add)
9. decode op (add)
10. add y ← y + z

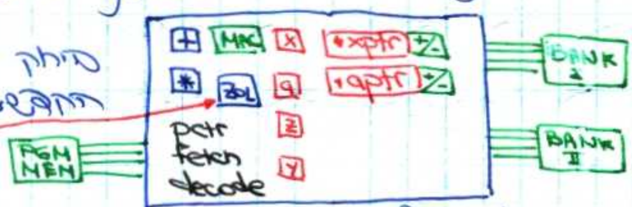
MAC (5–10)

אנו רואים שיש כאן מינימום 10 פעולות שונות. רוב ה- fetch ולוקחים הרבה יותר מאשר פעולות שבאמת עושות.

- אנו רואים שמימוש הפקודות שיש לנו מבצעות ה-MAC הם:

מיד אנו רואים את הפקודות הנדרשות במעבד.

Zero Overhead Loop



א. כי מעבד שיודע לבצע פעולת MAC בפקודה אחת נקרא מעבד MAC.

MAC:
fetch op (MAC)
decode op (MAC)
MAC y ← x, a

הוספת הפעולה MAC צריך לה הופך את המעבד ל- DSP.

ב. נוסיף יחידה שיודעת להוסיף ולהוריד לפוינטרים מבלי להעמיס ALU.

זה נקרא: pau
Pointer arithmetic unit

+/-:
update xptr || update aptr
במקביל

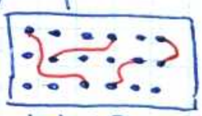
ג. האם אנו יכולים לבצע את שתי הטעינות בבת אחת ה-load?

לכאורה לא ניתן לבצע כי זה זמני כיוון ששניהם עוברים על ה- bus. לכן נפריד את הזיכרון לשני נפרדים כך שהאוגר נפרד ל-ALU כך ש-a ו-x לא יהיו על אותו החלק (בנק) של זיכרון. בפועל זה שונה את החלק של מבנה הזיכרון (שבו יש רק אפיק לזיכרון אחד).

:

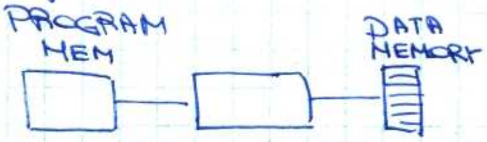
load x ← *xptr || load a ← *aptr

ב. במחשבים הישנים התכנית לא היה צ' הרצה מתוך הזיכרון של הפקודות אלא צ' חיבור בין כניסות הפיס השונות

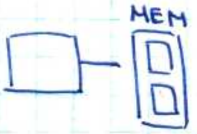


במחשב.

ב-MARK1 השתמשו לראשונה בפקודות שישבו בזיכרון.



ב- ENIAC היה חיבור של Bus אחד:



ואז ניתן לראות את הרעיון הן מכך שהתכנית יכולה לפנות ל את הזיכרון של עצמה ומכך שהזמן התכנית לשנות את עצמה. ואז ניתן להמציא את ה- LISP.

הארכיטקטורה של ENIAC היא המקובלת כיום בכל המעבדים פרט למעבדים של ה-DSP. בהם משתמשים בארכיטקטורה MARK1.

בגלל שני ה-BUS-ים ניתן להריץ את פעולת ה-fetch וה-decode (נראה בהמשך הבא).

נסתכל כעת מה הפעולות שעשינו לנו:

loop:
1. update xptr || update aptr
2. load x || load a
3. MAC

אנו רואים שבמחזור אחד לא ניתן לבצע את 3 הפעולות במקביל.

ג. נראה מה זה pipeline:

1	2	3	4	5	6	7	8
update I	update II	update III	update IV	update V	update VI		
	load I	load II	load III	load IV	load V	load VI	
		MAC I	MAC II	MAC III	MAC IV	MAC V	MAC VI

אנו רואים שהפעולה לוקחת 1-(אורך ה-pipeline)+n .

לוקח הזמן כדי למלא וכדי לרוקן את ה-pipeline. אנו מניחים שה- pipeline ישאר כל הזמן מלא ובשביל כך אנו משתמשים ב- NOP. הפעולה העיקרית של המעבד ב- DSP היא לבצע

לנצל כמו שצריך את ה-pipeline.

- יש DSP-ים פתוחים ויש DSP-ים לא פתוחים מבחינת אורך ה-pipeline.

- נראה כאן כיצד עושים ה-fetch וה-decode:
  שלב נוסף כאן מבצע pipeline: $z \leftarrow x+y$

fetch op (+)
decode op (+)
add x+y

1	2	3	4	5
f_1	f_2	f_3		
	d_1	d_2	d_3	
		add1	add2	add3

- אם נוסיף כאן ל-pipeline הקודם, אנו רואים שהוא בנוי pipeline בעומק 5.
- ה-pipeline מסובך כך שיש חומרה שעושה לו פעולה שיכולה לעבוד במקביל ל-ALU הראשי.
- למה ה-DSP לא משתמש ב-cash?
  כיוון שב-cash יש חוסר פרדיקטביליות - אנחנו לא יודעים מראש בדיוק כמה זמן יקח הדבר. יכולים להיות מקרים של Miss או HIT שבהתאם נצטרך לחכות לתשובה של בלוק שלם מהזיכרון.
- ה-DSP החדשים, יש שימוש מחדש ב-cash אם הוא יש, או בתוכנה, או בזיכרון ולא שימוש, ויש כלים חדשים להשתמש ב-cash בתוכנה כך שניתן להפעיל ולהפסיק אותו בהתאם לצורך.

דברים נוספים ב-DSP

- למעבד יש תקשורת עם העולם החיצוני בעזרת port-ים. שני סוגים עיקריים של port: IN ו-OUT.
  לכל port יש דגל שאומר האם יש בתוכו נתון שמחכה לנו. מצד אחד יש לנו מקור input שהוא לוקח את הפעולות הבאות, אך מצד שני אסור לנו להפסיד אף input. בשביל כך משתמשים בפסיקה: הפסיקה מודיעה למעבד שיש בו נתון מוכן.

- כאשר יש לנו פסיקה, הבעיה היא שאנו שומרים תמונה שלמה של כל מה שהיה ב- CPU, מטפלים בפסיקה ואז שוב מעלים חזרה מהזיכרון את התמונה.
- ב- DSP מטפלים בזה ע"י: Zoir. מוריד את כמות השעונים עבור (הטיפול) בפסיקה מ- 20 ל- 4. אבל זה מכריח אותנו שלא תהיה פסיקה בתוך פסיקה.
- לפיכך יש רק כמות מאוד קטנה ומוגבלת של אוגרים שאנו שומרים.

יש DSP של Fixed Point וגם של Floting Point. הרוב הם מהסוג הראשון. זה מקשה על המתכנת בכך שהוא צריך גם לדאוג לדברים נוספים עקב כך.