06.02.08

הרצאה 3

חובת ההגשה בקורס היא 5 תרגילים.

חזרה על הרצאה קודמת (הרצאה 1):

- ראינו רכיבים בשם Full Adders אשר מחברים 2 ביטים, והשתמשנו בהם על מנת לחבר מספר בינארי בעל 4 ביטים עם מספר אחר בעל 4 ביטים.

- ראינו כי כאשר יש חיבור של שני מספרים בינריים (שבהם הסיבית השמאלית ביותר היא סיבית הסימן, הפיכת סימן של מספר נעשית באמצעות משלים ל-2) עלול להיווצר overflow, ואנו מכריזים על overflow ע"פ הזוגיות של סכום ה carriers עבור שתי פעולות החיבור השמאליות ביותר, כלומר, אם שתי פעולות החיבור האחרונות (או אף אחת מהן) נותנות Carry, אז אין overflow, אבל אם רק לאחת מהן יש Carry אז יש overflow.

  לכן כאשר מחברים מספר חיובי עם מספר שלילי לא יהיה overflow.

- למדנו על כך שקיימות 16 פונקציות בינאריות, הפונקציה f≡1 היא אחת מהפונקציות האלה. למדנו על מערכות אוניברסאליות, כאשר נרצה להוכיח כי קבוצה מסוימת היא קבוצה אוניברסאלית, לא נוכל להשתמש בפונקציה זו מבלי לייצר אותה.

  למשל המערכת {XoR, OR} אינה מערכת אוניברסאלית, אבל אם מוסיפים לה פונקציה קבועה היא הופכת להיות מערכת אוניברסלית.

מפות קרנו: מפות קרנו נועדו לסייע לנו לפשט פונקציות בינאריות.

בשולי הטבלה כותבים את כל הinput-ים האפשריים של הפונקציה, כך שכל שני ריבועים סמוכים במפה נבדלים בערך של משתנה אחד בדיוק, וכך נוצרת סימטריה במפה.

בתאי הטבלה רושמים את ה-output של הפונקציה במקום המתאים, כל שבכל תא יש אפס או אחד.

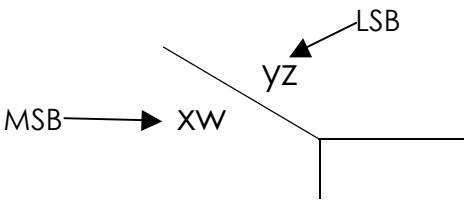
באמצעות מפת קרנו ניתן לפשט את הפונקציה באופן מקסימלי לסכום של מכפלות (SOP) או מכפלה של סכומים (POS). (פונקציות מהסוג XoR או XNoR לא ניתן לפשט באמצעות מפות קרנו).

SOP: נרצה להקיף את כל ה-1-ים במפה בקבוצות שגודלן הוא חזקה של 2, ע"פ החוקיות (שנלמדה בתרגול), נרצה להקיף כל פעם כמה שיותר תאים, מותר שאותו תא יהיה שייך ליותר מקבוצה אחת.

לעיתים יש יותר מאפשרות נכונה אחת להקיף את כל ה-1-ים במפה.

<u>קונבנציה:</u>

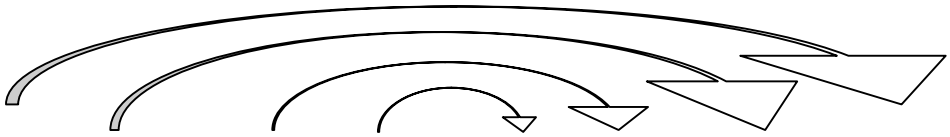
כאשר בונים מפת קרנו:



מפות קרנו של חמישה משתנים:

לדוגמה: (לשם הפשטות, זוהי מפת קרנו של ארבעה משתנים אבל יש שלושה משתנים בשורה, על מנת שנוכל ללמוד על המצבים החוקיים והלא חוקיים במפת קרנו של חמישה משתנים).

בשביל לבנות מפת קרנו יש לייצר
תמונת ראי



xyz / w	000	001	011	010	110	111	101	100
0								
1								

הנה חלק מהקבוצות החוקיות (חלקן אינן טריביאליות) במפת קרנו:

xyz / w	000	001	011	010	110	111	101	100
0								
1								

xyz / w	000	001	011	010	110	111	101	100
0								
1								

xyz / w	000	001	011	010	110	111	101	100

0								
1								

xyz / w	000	001	011	010	110	111	101	100
0								
1								

xyz / w	000	001	011	010	110	111	101	100
0								
1								

וכדומה....

<u>אבל למשל הקבוצה הבאה אינה חוקית:</u>

xyz / w	000	001	011	010	110	111	101	100
0								
1								

כי אם נסתכל על המשתנים המשותפים של הקבוצה הזאת, אז יש רק משתנה יחיד משותף וזה w,
אבל אמורים להיות 2 משתנים משותפים (לרביעייה שני משתנים משותפים).

הערה: ייתכן שבבחינה תהיה פונקציה עם 5 משתנים, אבל אז אחד המשתנים יהיה תמיד קבוע ואז נהיה צריכים לעשות מפת קרנו של 4 משתנים בלבד. זהירות, לא להסתבך!

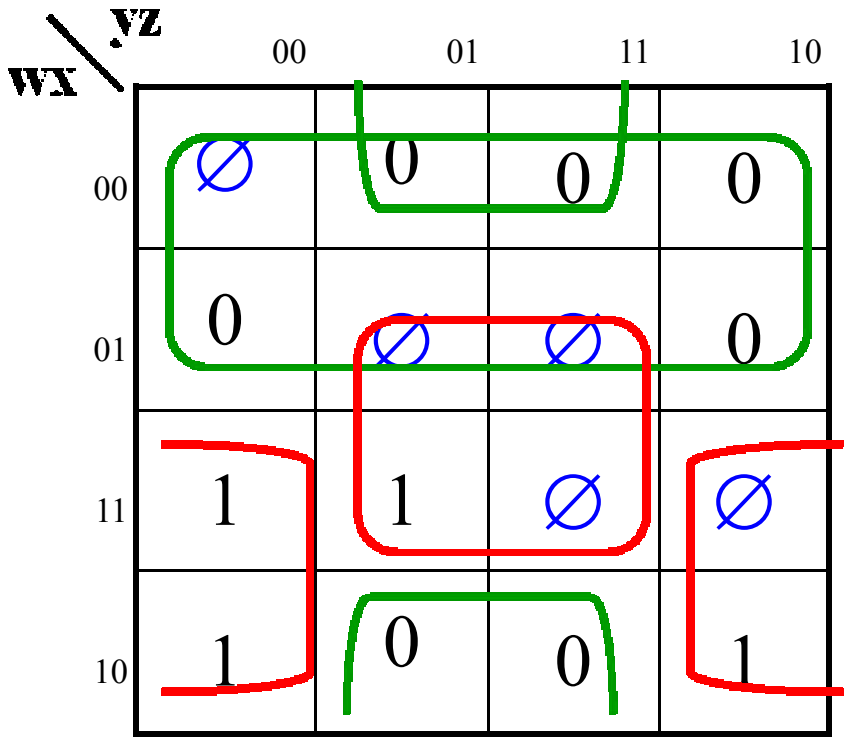
Don't Care:

אם יש פונקציה בינארית כלשהי שעבור input מסוים לא משנה לנו מהו ה output אז בתא של מפת קרנו המתאים לinput זה נשים את האות d או את הסימן של הקבוצה הריקה.

למשל אם צריך לתאר טבלת אמת עם 6 מצבים (שלמשל כל אחד מהם הוא ייצוג בינארי של מספר בין 0 ל5), אז נעשה זאת באמצעות 3 ביטים, אבל אז ישארו לנו עוד שני input-ים שלא רלוונטיים (המייצגים את המספרים 6 ו-7), אז במפת קרנו נשים עבורם don't care.

כאשר בתא מסוים יש סימן של don't care ניתן להתייחס אליו כאילו הוא 1 (אם זה עוזר לנו להרחיב קבוצה מסוימת של 1-ים) או כאילו הוא 0 (אחרת).

דוגמה (הסבר למטה):



סכום של מכפלות:

נתייחס רק לקבוצות של ה-1-ים (באדום). נקבל:

$$f = z'w + zx$$

סכום של מכפלות ניתן לממש באמצעות שערי NAND בלבד (ע"י שימוש בכללי דה מורגן):

$$f = z'w + zx = \overline{\overline{(w\bar{z})}\,\overline{(zx)}}$$

(הראינו כבר כי NOT(x)=NAND(x,x))

<u>מכפלה של סכומים:</u>

נתייחס רק לקבוצות של ה-0-ים (בירוק). נקבל:

$$f = w(x+z')$$

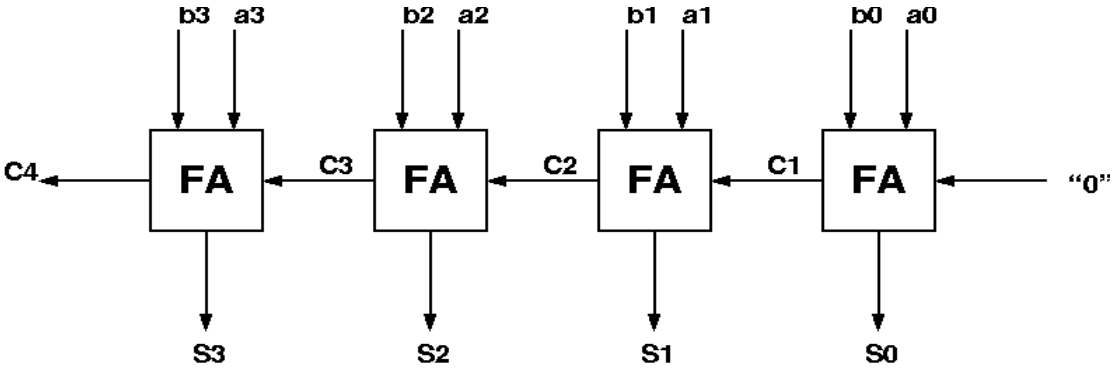
מכפלה של סכומים ניתן לממש באמצעות שערי NOR בלבד (ע"י שימוש בכללי דה מורגן):

$$f = w(x+z') = \overline{\bar{w} + \overline{(x+\bar{z})}}$$

(הראינו כבר כי NOT(x)=NOR(x,x))

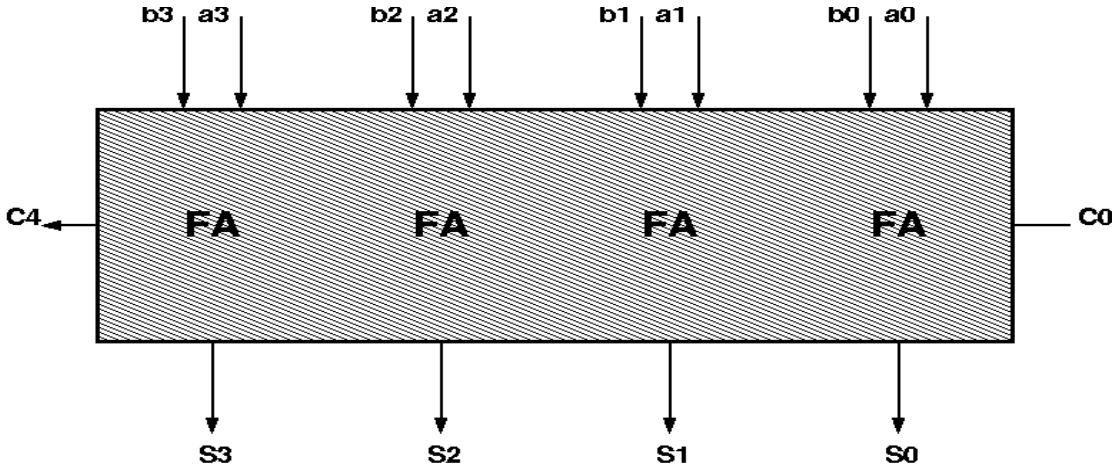
<u>חיבור מספרים בינאריים באמצעות רכיבי ה full-adder:</u>

שיעור קודם ראינו כי המימוש הבא עבור מחבר של שני מספרים בני 4 סיביות הוא איטי:



מימוש זה הינו איטי משום שכל FA (פרט לראשון) מחכה שה FA הקודם לו יסיים לבצע את הפעולה שלו (כי הוא צריך את הcarry מהפעולה הקודמת).

לשם כך משתמשים ב Look Ahead Carry על מנת לחבר שני מספרים בני 4 סיביות:



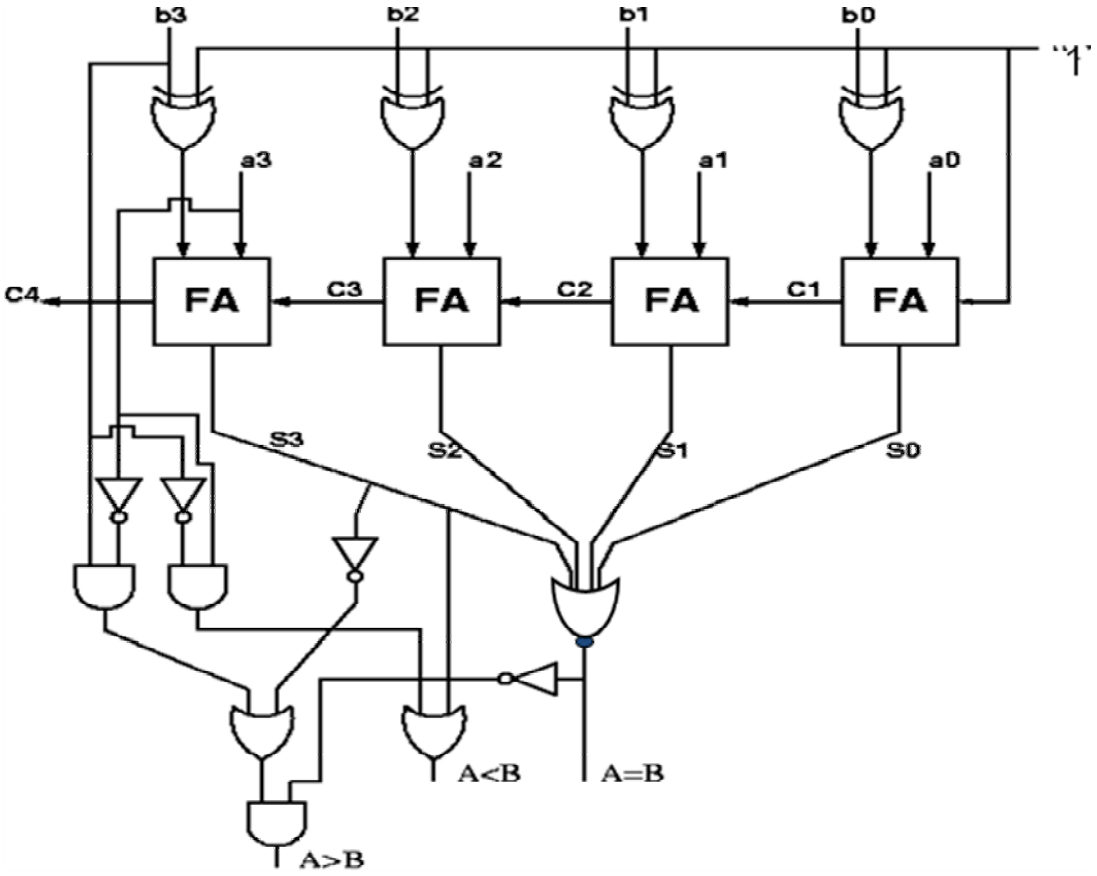
ברכיב זה כל FA אינו מחכה לCarry מהFA הקודם, אלא מחליט מהו הCarry לפי הinput בלבד. לפיכך, כל פעולת החיבור נעשית בעליית שעון אחת, לעומת 4 עליות שעון במימוש הקודם (כלומר במימוש הקודם כל הפעולות של ה-FA נעשו בצורה טורית, אחד אחרי השני וכעת הם מתבצעים במקביל). עם זאת, המימוש הקודם יותר מודולרי (כלומר משתמשים ברכיבי היסוד על מנת לבנות רכיב מתוחכם יותר, לעומת המימוש החדש שבו יוצרים רכיב חדש).

כיצד קורה שהמחשב מבצע הפעלה של פונקציית sin כמעט באותה מהירות של פעולת חיבור אחת? הרי פעם זה היה לוקח הרבה יותר זמן, שהרי מדובר בחישוב טור טיילור וחיבור איברים עד להתכנסות. הסוד הוא, נניח שאנו במחשב 32 ביט, שקיימת טבלה שגודלה 4GB בזיכרון ROM (Read Only Memory) שעבור כל מספר אפשרי x (כמות המספרים הממשיים במחשב הוא סופי) קיים תא בטבלה שהוא התוצאה של הפעלת הפונקציה sin עליו, ואז כשמפעילים את הפונקציה המחשב פשוט שולף מהטבלה.

<u>שאלה מבחינה:</u>

בהינתן שני מספרים של 3 ביטים (+סיבית סימן) A ו B, עלינו לבנות משווה גודל שיחליט מי מביניהם גדול יותר.

פיתרון:



הסבר:

נתונים שני מספרים בינאריים:

$A=a_3a_2a_1a_0$

$B=b_3b_2b_1b_0$

בכדי לגלות מי מהם גדול יותר נעשה את פעולת החיסור A-B.

לB עושים בהתחלה משלים ל1 באמצעות שערי הXOR. בשביל שיהיה משלים ל-2 מכניסים 1 בכניסה הראשונה של רכיב הFA הראשון.

באמצעות רכיבי הFA מחברים בין המספר A למשלים ל-2 של B, מקבלית את התוצאה:

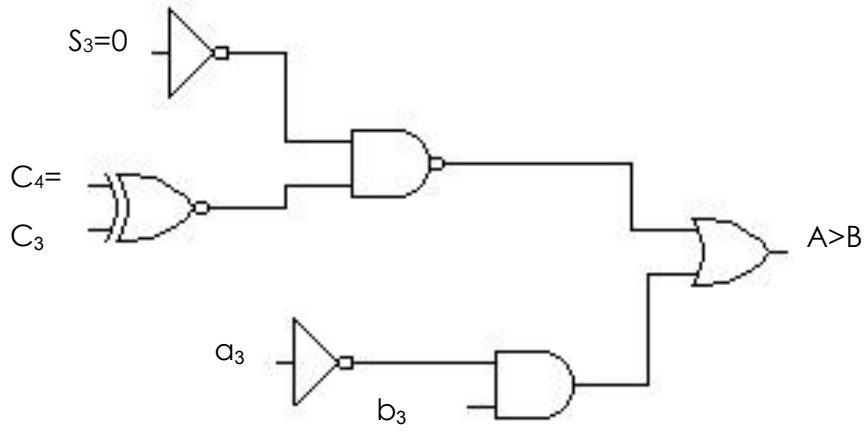
$S=S_3S_2S_1S_0$ ויכול להיות שיש Carry סופי C_4.

מכניסים את S_0, S_1, S_2, ו-S_3 לשער NOR. אם ביציאה של שער זה מתקבל 1 זה אומר ש $S_0=S_1=S_2=S_3=0$. כלומר A-B=0 ולכן A=B.

B>A אם ורק אם A-B<0, כלומר S שלילי, כלומר $S_3=1$, או אם A שלילי ו B חיובי (ואז $a_3=1$ וגם $b_3=0$).

A>B אם ורק אם A-B>0 וזה אם מקבלים בתוצאה שסיבית הסימן היא 0, כלומר שהתוצאה חיובית, ($S_3=0$ כלומר) או אם A>0 ו B<0 (כלומר אם סיבית הסימן של A, a_3 היא 0 וסיבית הסימן של B, b_3 היא 1.

כלומר, לדוגמה:



מקודדים (Encoders) ומפענחים (Decoders):

מקודד לוקח קוד פשוט והופך אותו לקוד מסובך לאחסון או לשליחה.

מפענח עושה בדיוק ההפך.

Endec היה בעבר בשימוש, הוא שימש גם כמפענח וגם כמקודד (רכיב משולב) והיה מבוסס על חומרה.

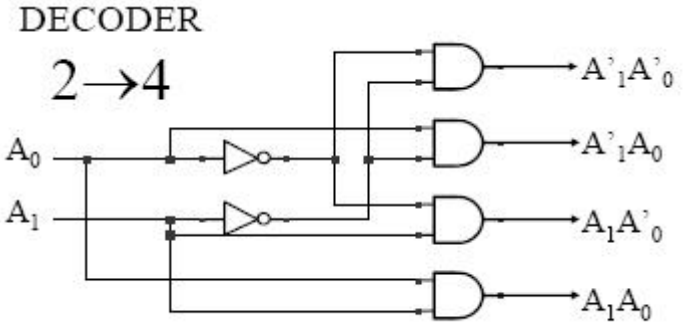
Codec הוא כמו Endec רק שהוא מבוסס על תוכנה. Codec פופולרי כיום- לדוגמה MP3, וידאו.

למשל בטלפון יש A to D (Analogue to Digital) אשר מקודד את הקול שמדברים לאותות דיגיטליים, לאחר מכן ע"י D to A (Digital to Analogue) מתבצע פענוח ע"י הצד השני.

מפענחים Decoders:

רכיבים בעלי n כניסות ראשיות ובעלי 2^n יציאות ראשיות, ממיר מייצוג בינארי לאונארי.

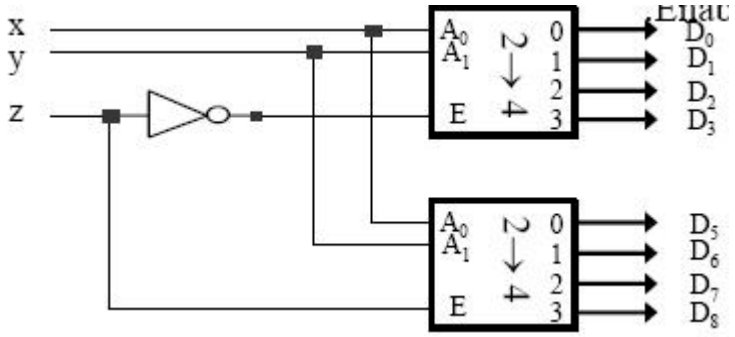
מימוש של מפענח 2→4:



מכניסים מספר בינארי A_1A_0 למפענח, והוא מוציא זרם ביציאה שמייצגת את המספר האונארי המתאים בלבד, לדוגמה אם המספר הבינארי הוא 00, היציאה העליונה בלבד תוציא זרם.

אם רוצים לעשות מפענח של 3→8, (או מפענח של יותר מ-2 כניסות) צריך להגדיר כניסת Enable לכל מפענח 2→4 (אם ל Enable נכנס 1 לוגי המקודד יבצע את הפעולה, אחרת המקודד יוציא 0 לוגי בכל היציאות).

מימוש של מפענח 3→8 ע"י שימוש במפענחים 2→4 (עיקרון המודולריות):

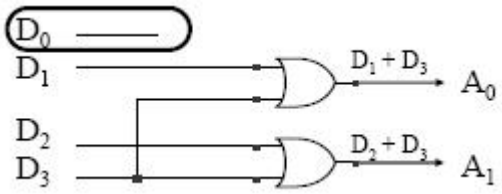


ניתן לממש גם ללא שימוש בשני מקודדים 2→4 אבל אז יפגע עיקרון המודולריות.

מקודדים Encoders:

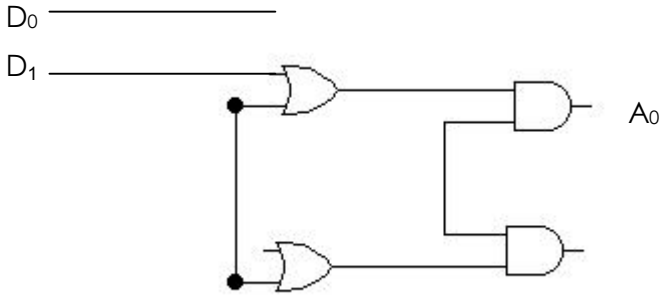
מקודד הוא פונקציה הפוכה למפענח, זהו רכיב בעל 2^n כניסות ו n יציאות, ממיר מייצוג אונארי לבינארי.

מימוש של מקודד 4→2 (בהנחה שהקלט תקין):



מכניסים באחת היציאות (בלבד) 1 לוגי, היציאה מייצגת מספר אונארי, והפלט הוא המספר הבינארי המתאים.

מכיוון שנרצה לייצג גם מקודדים עם יותר מ-4 כניסות, נרצה לייצר יציאת Enable, נעשה זאת באופן הבא:



A_1

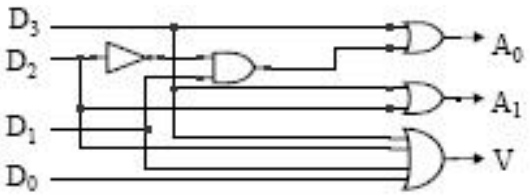
מקודד עדיפויות Priority Encoder:

זהו מקודד אשר בנוסף לפעולה הרגילה של ה-Encoder מכיל יציאה בשם Valid (מסומנת באות V) אשר ערכה הוא 1 לוגי אם הקלט תקין ו-0 לוגי אם הקלט אינו תקין (אם כל הכניסות ערכן 0 לוגי).

כלומר טבלת האמת של מקודד העדיפויות היא:

D_3	D_2	D_1	D_0	A_1	A_0	V		
0	0	0	0	0	0	0		not valid
0	0	0	1	0	0	1	0	Valid
0	0	1	∅	0	1	1	1	
0	1	∅	∅	1	0	1	2	
1	∅	∅	∅	1	1	1	3	

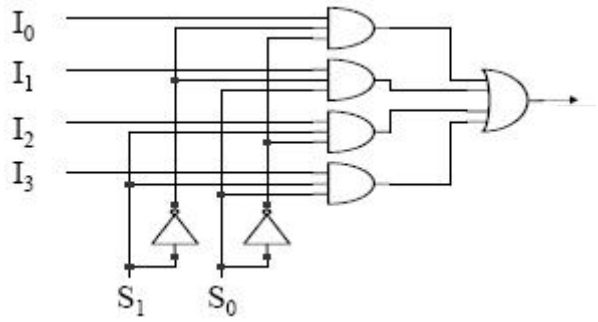
והמימוש:



מרבבים (Multiplexers):

לרכיב זה 2^n כניסות, n קווי בקרה, ויציאה אחת, הרכיב בורר את הכניסה שתצא ביציאה. בחירת הכניסה שתעבור ליציאה נעשית באמצעות n קווי הברירה, שהם הייצוג הבינארי של מספר הכניסה שצריכה לעבור ליציאה.

דוגמה למימוש ל1←4 mux, אשר S_0 ,S_1 הם כניסות הבקרה שלו, והמספר הבינארי S_1S_0 מייצד מספר אונארי n אזי הכניסה I_n תעבור ליציאה:



שימוש של Multiplexers במחשב: למחשב יש קלט מה RAM ומה DISC. באמצעות ה CONTROLER שולט על איזה מהכניסות האלה יעברו ל CPU באמצעות הBUS. כלומר MULTIPLEXERS הם אבן בניין עיקרית במחשב.