מבוא למחשבים
Lecture 5

arithmetic

Dr. Ron Shmueli

חלק נכבד מהשקפים מבוסס על הספר:

- Heuring and Jordan: "Computer System Design and Architecture", **Prentice Hall**, 2004
- Chapter 6

Ripple carry adder

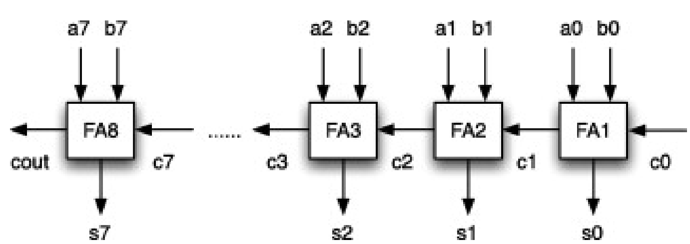


Fig. 3. A 8-bits carry ripple adder.

- חישוב זמן ביצוע.
- 2dt למימוש כל פונקציה לוגית (בהתעלמות מ Fan In)
- מעבר למסכם בבסיס גבוה יותר 2^k - שיפור זמן ביצוע

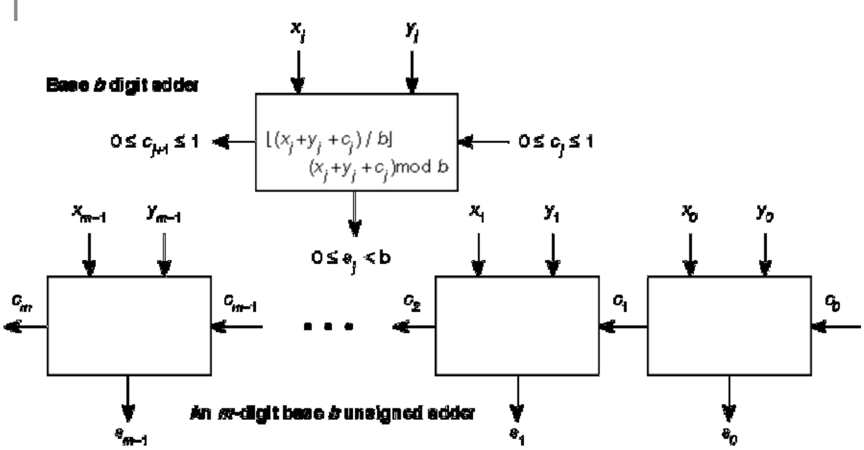
Two Level Logic Design of a Base 4 Digit Adder

x_b	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
x_a	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
y_b	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
y_a	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
c_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
c_1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1
s_b	0	0	0	1	1	1	1	0	0	1	1	1	1	0	0	0	1	1	1	0	0	0	0	1	1	0	0	0	0	1	1	1
s_a	0	1	1	0	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1	1	0	0	1

- The base 4 digit x is represented by the 2 bits x_b x_a, y by y_b y_a, and s by s_b s_a
- s_a is independent of x_b and y_b, c_1 is given by $y_b y_a c_0 + x_a y_b c_0 + x_b x_a c_0 + x_b y_a c_0 + x_b x_a y_a + x_a y_b y_a + x_b y_b$,

while s_b is a 12 input OR of 4 input ANDs

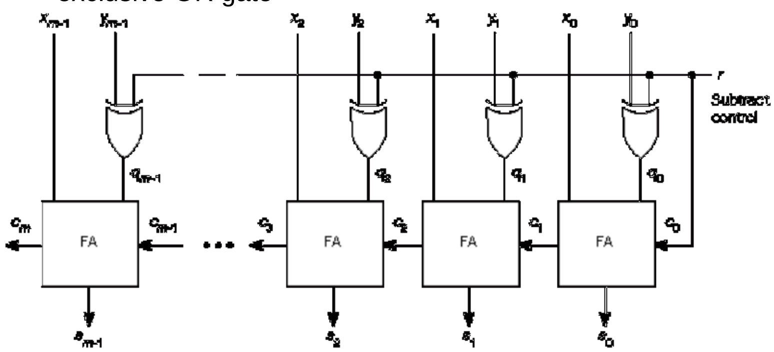
Fig. 6.1 An m-digit base B unsigned adder



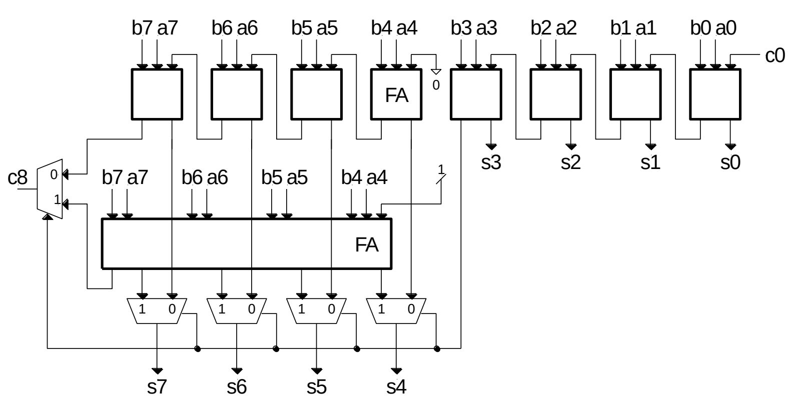
- Typical cell produces $s_j = (x_j + y_j + c_j) \bmod b$ and $c_{j+1} = \lfloor (x_j + y_j + c_j)/b \rfloor$

Fig. 6.3 2’s Complement Adder/Subtracter

- A multiplexer to select y or its complement becomes an exclusive OR gate

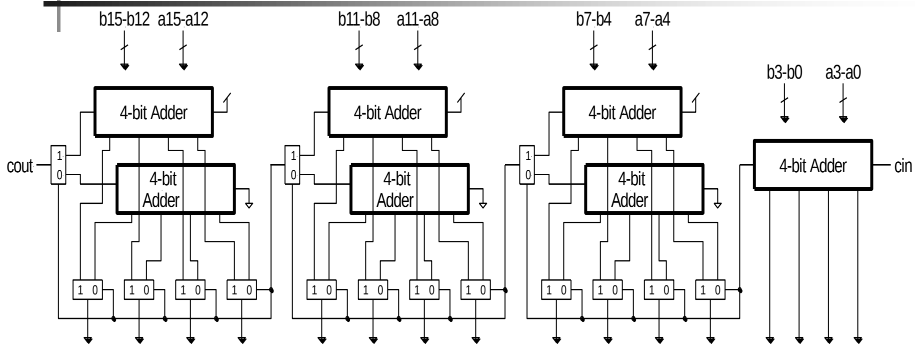


Speeding Up Addition With CSA
Carry Select Adder



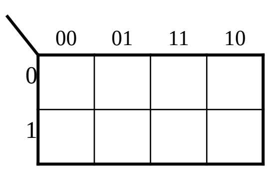
זמן ביצוע:

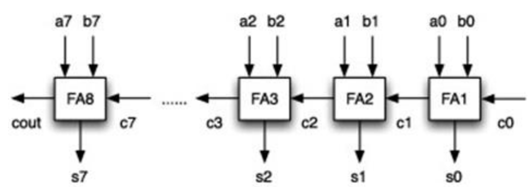
Extending Carry-select to multiple blocks



- גודל קבוצה אופטימלי:
- b- מס ביטים בבלוק, n- מס ביטים כולל , n/b - מס בלוקים , dt – זמן ביצוע
- T=2(dt)b+2(dt)(n-b)/b
- גודל קבוצה אופטימלי: dT/db= 2dt+2ndt/(b^2) = 0 → b=sqrt(n)

Speeding Up Addition With CLA
Carry Lookahead Adder



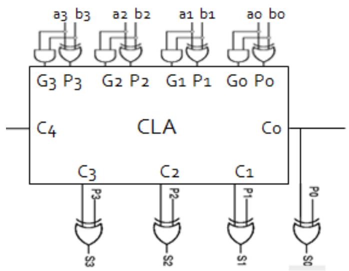


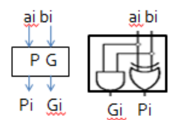
Xi	Yi	Ci+1
0	0	
0	1	
1	0	
1	1	

$G_j = x_j \cdot y_j$

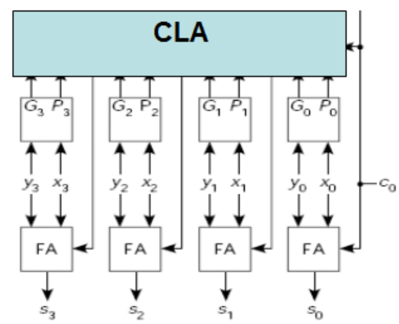
$P_j = x_j \text{ xor } y_j = x_j + y_j$

CLA - Carry Look Ahead adder





מימוש מותאם לספר:



חישוב זמן ביצוע

FAN IN מגבלת

Recursive Carry Lookahead Scheme

Given

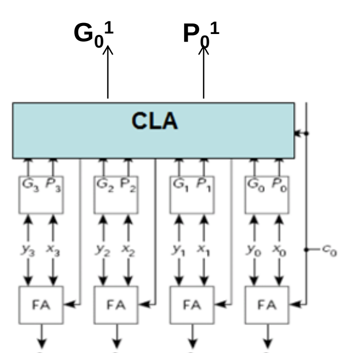
$c_1=g_0+(p_0\cdot c_0)$

$c_2=g_1+(p_1\cdot c_1)=g_1+(p_1\cdot(g_0+(p_0\cdot c_0)))=g_1+p_1\cdot g_0+p_1\cdot p_0c_0$

$c_3=g_2+p_2\cdot g_1+p_2\cdot p_1\cdot g_0+p_2\cdot p_1\cdot p_0c_0$

$c_4=\underbrace{g_3+p_3\cdot g_2+p_3\cdot p_2\cdot g_1+p_3\cdot p_2\cdot p_1\cdot g_0}_{G}+\underbrace{p_3\cdot p_2\cdot p_1\cdot p_0c_0}_{P}$

G_0^1 = G3 + P3G2 + P3P2G1 +P3P2P1G0
P_0^1 = P3P2P1P0



P_j^l יקבל 1 אם נשא הקבוצה מועבר ליציאה

G_j^l - יקבל 1 אם הקבוצה הכלולה מתחת ל CLA מייצרת נשא 1.

Carry Lookahead Adder for Group Size k = 2

- c1= $G_0+P_0c_0$
- c2= $G_1+P_1G_0+P_1P_0c_0 = G_0^1+ P_0^1c_0$
- $\underbrace{G_1+P_1G_0}_{G_0^1}$ $\underbrace{P_1P_0}_{P_0^1}$
- c4= $G_1^1+P_1^1 G_0^1 + P_1^1 P_0^1 c_0 = G_0^2+ P_0^2c_0$

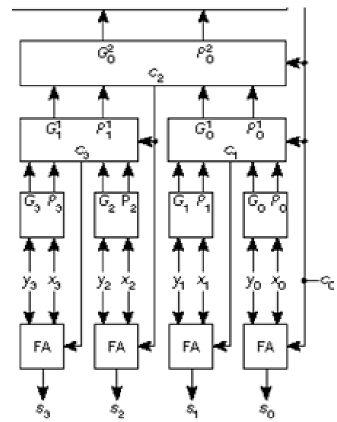
מספר הרמות:

עבור מסכם של M סיביות וגודל קבוצה k

מספר רמות = $\log_k M$

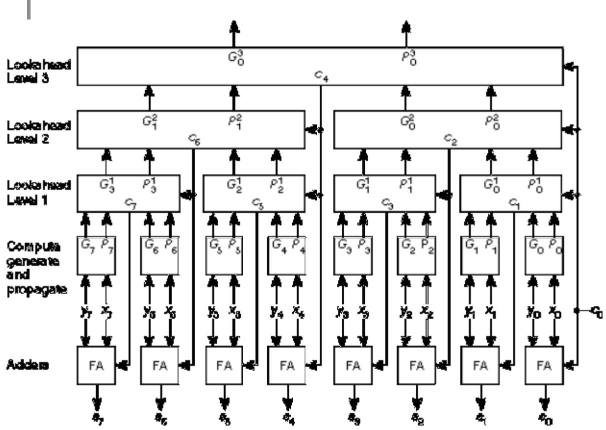
לדוגמא: מסכם 8 סיביות בגודל קבוצה 2

מספר הרמות = 3



חישוב זמן ביצוע

Fig. 6.4 Carry Lookahead Adder for Group Size k = 2

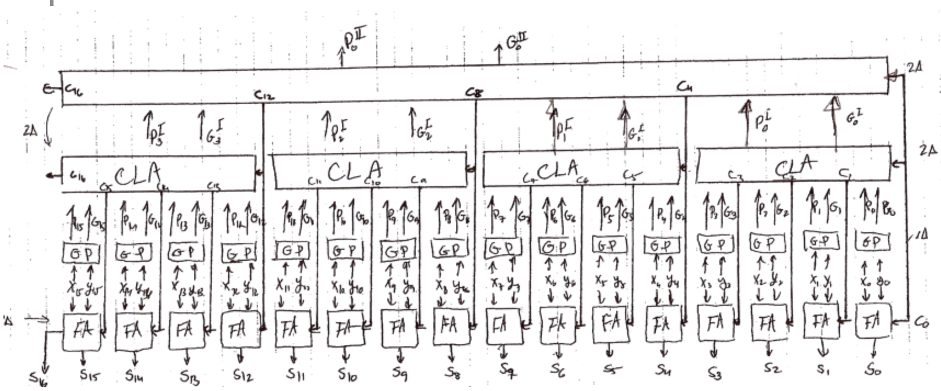


חישוב זמן ביצוע
מסכם 8 סיביות

[זמן ביצוע]= 1dt +[$\log_k M$ (2) - 1] 2dt+2dt= [1 + 4 $\log_k M$]dt

PG #levels עולים (ויורדים) רמה אחרונה רק עלייה FA

מסכם 16 סיביות K=4

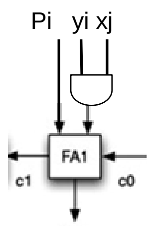


- השוואת זמן ביצוע עם מסכם גלי.

דוגמא

- לרשותך יחידות לסיכום 4 סיביות משני סוגים CLA ו- Ripple - נדרש לתכנן מסכם 16 סיביות – מה זמן הביצוע בכל אחד מהמקרים?

Fig. 6.5 Digital Multiplication Schema

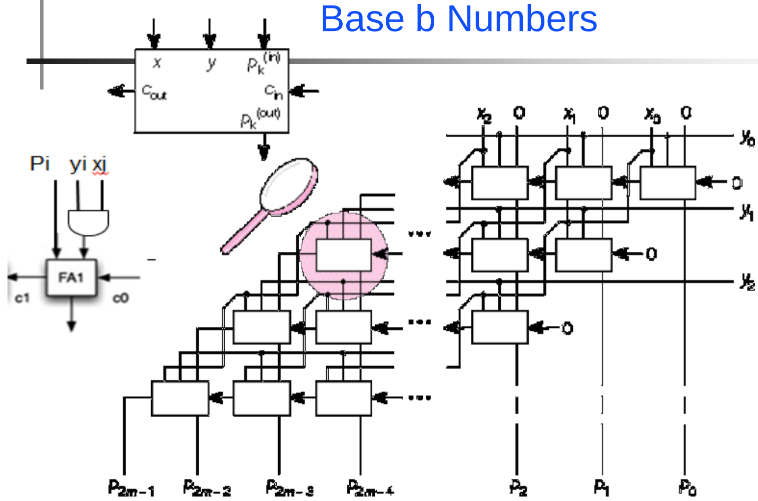


				x_3	x_2	x_1	x_0	**multiplicand**
				y_3	y_2	y_1	y_0	**multiplier**
			$(xy_0)_4$	$(xy_0)_3$	$(xy_0)_2$	$(xy_0)_1$	$(xy_0)_0$	pp_0
		$(xy_1)_4$	$(xy_1)_3$	$(xy_1)_2$	$(xy_1)_1$	$(xy_1)_0$		pp_1
	$(xy_2)_4$	$(xy_2)_3$	$(xy_2)_2$	$(xy_2)_1$	$(xy_2)_0$			pp_2
$(xy_3)_4$	$(xy_3)_3$	$(xy_3)_2$	$(xy_3)_1$	$(xy_3)_0$				pp_3
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0	

p: product   pp: partial product

לעשות דוגמא 3 x 3

Fig. 6.6 Parallel Array Multiplier for Unsigned Base b Numbers



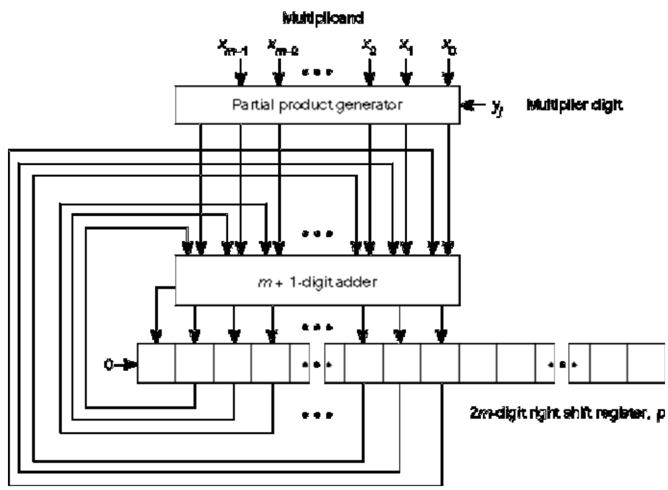
מס מודולים
שנותרו בשורה

[זמן ביצוע]= dt + 2mdt + (m-1) 2 2dt = 3(2m-1)dt

(and) מס סיביות בשורה שורות שנותרו

Fig. 6.7 Unsigned Series-Parallel Multiplication Hardware

לתאר כתיבת תוכנית לביצוע כפל



Steps for Using the Unsigned Series-Parallel Multiplier

1) Clear product shift register p.
2) Initialize multiplier digit number j=0.
3) Form the partial product xy_j.
4) Add partial product to upper half of p.
5) Increment j=j+1, and if j=m go to step 8.
6) Shift p right one digit.
7) Repeat from step 3.
8) The 2m digit product is in the p register.

למידה עצמית

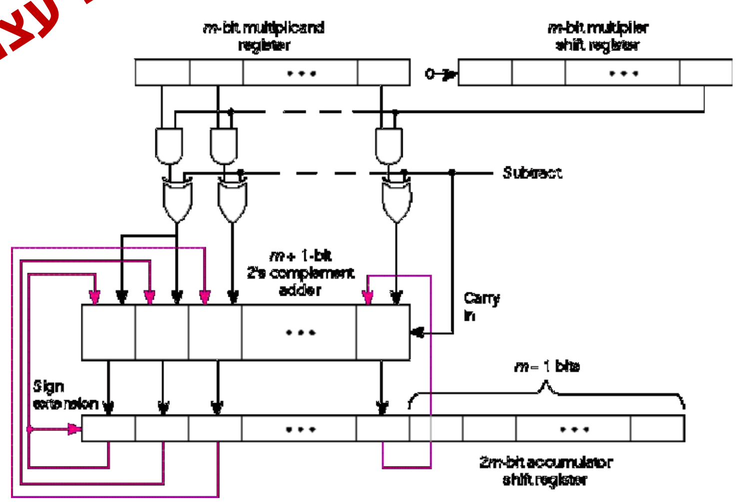
Signed Multiplication

- The sign of the product can be computed immediately from the signs of the operands
- For complement numbers, negative operands can be complemented, their magnitudes multiplied, and the product recomplemented if necessary
- A complement representation multiplicand can be handled by a b's complement adder for partial products and sign extension for the shifts
- A 2's complement multiplier is handled by the formula for a 2's complement value: add all PP's except last, subtract it.

$$\text{value}(x) = -x_{m-1}2^{m-1} + \sum_{i=0}^{m-2} x_i 2^i \qquad \textbf{Eq. 6.25}$$

למידה עצמית

Fig. 6.8 2's Complement Signed Multiplier Hardware



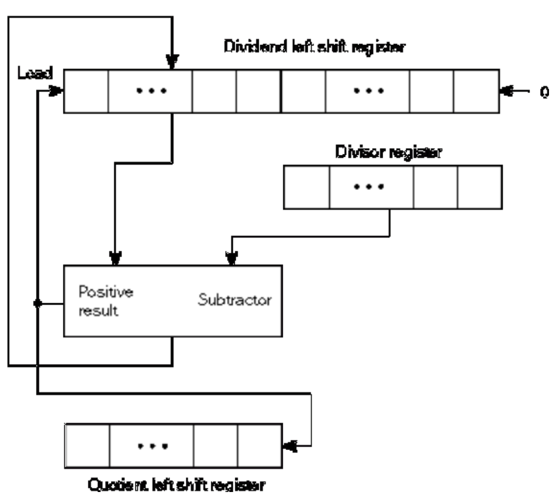
למידה עצמית

Steps for Using the 2's Complement Multiplier Hardware

1) Clear the bit counter and partial product accumulator register.
2) Add the product (AND) of the multiplicand and rightmost multiplier bit.
3) Shift accumulator and multiplier registers right one bit.
4) Count the multiplier bit and repeat from 2 if count less than m-1.
5) Subtract the product of the multiplicand and bit m-1 of the multiplier.

Note: bits of multiplier used at rate product bits produced

Fig 6.9 Unsigned Binary Division Hardware



- 2m bit dividend register
- m bit divisor
- m bit quotient
- Divisor can be subtracted from dividend, or not

Use of Division Hardware for Integer Division

1) Put dividend in lower half of register and clear upper half. Put divisor in divisor register. Initialize quotient bit counter to zero.
2) Shift dividend register left one bit.
3) If difference positive, shift 1 into quotient and replace upper half of dividend by difference. If negative, shift 0 into quotient.
4) If fewer than m quotient bits, repeat from 2.
5) m bit quotient is an integer, and an m bit integer remainder is in upper half of dividend register.

Integer Binary Division Example: D=45, d=6, q=7, r=3

	D	0 0 0 0 0 0 1 0 1 1 0 1		
	d	0 0 0 1 1 0		
Init.	D	0 0 0 0 0 1 0 1 1 0 1 -		
	d	0 0 0 1 1 0		
diff(-)	D	0 0 0 0 1 0 1 1 0 1 - -	q	0
	d	0 0 0 1 1 0		
diff(-)	D	0 0 0 1 0 1 1 0 1 - - -	q	0 0
	d	0 0 0 1 1 0		
diff(-)	D	0 0 1 0 1 1 0 1 - - - -	q	0 0 0
	d	0 0 0 1 1 0		
diff(+)	D	0 0 1 0 1 0 1 - - - - -	q	0 0 0 1
	d	0 0 0 1 1 0		
diff(+)	D	0 0 1 0 0 1 - - - - - -	q	0 0 0 1 1
	d	0 0 0 1 1 0		
diff(+)	rem.	0 0 0 0 1 1	q	0 0 0 1 1 1

יישום shifter בעזרת מרבבים

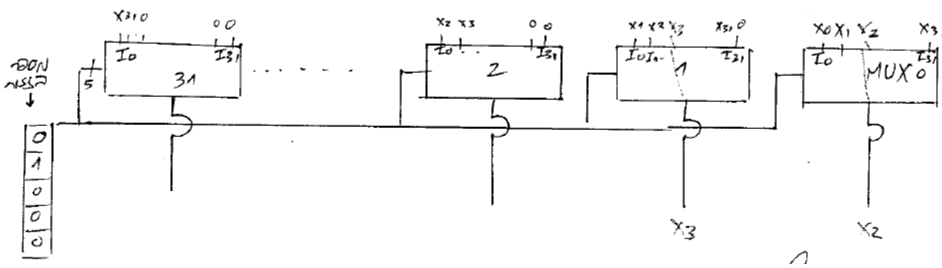


Fig 6.12 Barrel Shifter with a Logarithmic Number of Stages

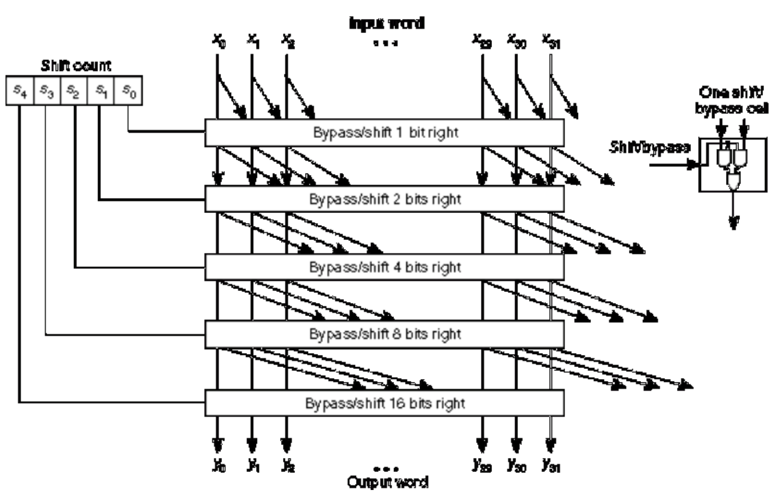
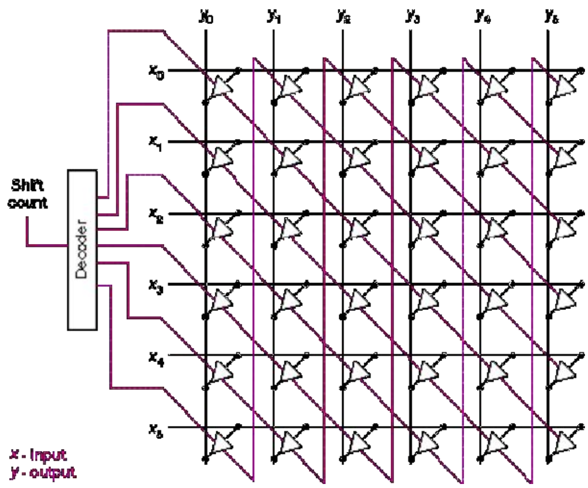


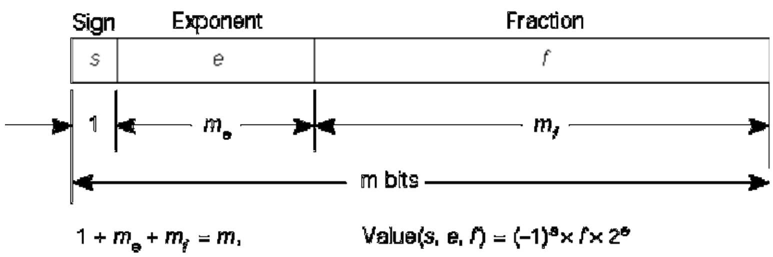
Fig 6.11 A 6 Bit Crossbar Barrel Rotator for Fast Shifting



Floating Point Numbers

- נקודה קבוע והשוואה עם נקודה צפה
- ישר המספרים וגודל שגיאה

Fig 6.14 Floating Point Numbers Include Scale & Number in One Word



- All floating-point formats follow a scheme similar to the one above
- s is sign, e is exponent, and f is significand
- We will assume a fraction significand, but some representations have used integers

Signs in Floating Point Numbers
Normalized Floating Point Numbers
Comparison of Normalized Floating Point Numbers

Normalized Floating Point Numbers

- There are multiple representations for a FP #
- If f_1 and $f_2 = 2^d f_1$ are both fractions & $e_2 = e_1$-d, then (s, f_1, e_1) & (s, f_2, e_2) have same value
- Scientific notation example: $.819\times10^3 = .0819\times10^4$
- A normalized floating point number has a leftmost digit non-zero (exponent small as possible)
- Zero cannot fit this rule; usually written as all 0s
- In norm. base 2 left bit =1, so it can be left out
  - So called **<u>hidden bit</u>**

Normalized FP numbers can be compared for <,≤,>,≥,=,≠ as if they were integers. This is the reason for the s,e,f ordering

Fig 6.15 IEEE Single-Precision Floating Point Format

sign	exponent	fraction
s	ê	$f_1 f_2 \ldots f_{23}$
0	1 … 8	9 … 31

ê	e	Value	Type
255	none	none	Infinity or NaN
254	127	$(-1)^s\times(1.f_1f_2\ldots)\times2^{127}$	Normalized
...	...	...	...
2	-125	$(-1)^s\times(1.f_1f_2\ldots)\times2^{-125}$	Normalized
1	-126	$(-1)^s\times(1.f_1f_2\ldots)\times2^{-126}$	Normalized
0	-126	$(-1)^s\times(0.f_1f_2\ldots)\times2^{-126}$	Denormalized

- Exponent bias is 127 for normalized #s

Special Numbers in IEEE Floating Point

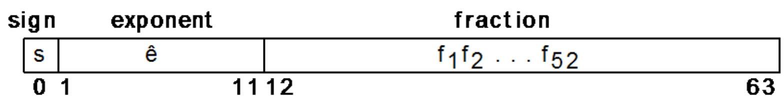
- An all zero number is a normalized 0
- Other numbers with biased exponent e = 0 are called denormalized
- Denorm numbers have a hidden bit of 0 and an exponent of -126; they may have leading 0s
- Numbers with biased exponent of 255 are used for ±∞ and other special values, called NaN (not a number)
- For example, one NaN represents 0/0

דוגמאות

- $(-1.5)_{10}=(\ ?\ )_{IEEE}$
- $(3825x10^{2})_{10}=(\ ?\ )_{IEEE}$
- $(00111111001000000000000000000000)_{IEEE}=$
- $(00000000001000000000000000000000)_{IEEE}=$

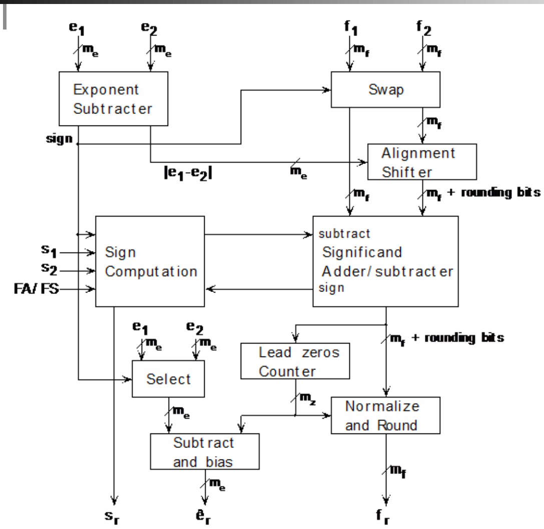
Play Store - floating point app.

Fig 6.16 IEEE Standard Double Precision Floating Point



- Exponent bias for normalized #s is 1023
- The denorm biased exponent of 0 corresponds to an unbiased exponent of -1022
- Infinity and NaNs have a biased exponent of 2047
- Range increases from about $10^{-38} \le |x| \le 10^{38}$ to about $10^{-308} \le |x| \le 10^{308}$

Fig 6.17 Floating Point Add & Subtract Hardware



- Adders for exponents and significands
- Shifters for alignment and normalize
- Multiplexers for exponent and swap of significands
- Lead zeros counter

Fig 6.13 Complete Arithmetic Logic Unit Block Diagram