

# Operace v FP a iterační algoritmy

INP 2025

FIT VUT v Brně



# Př. Latence instrukcí (mikroarchitektura Intel Haswell, 2013)

| Instrukce | Latence  |
|-----------|----------|
| MOV r,r   | 1        |
| MOV m,r   | 3        |
| ADD r,r   | 1        |
| MUL r32   | 4        |
| DIV r32   | 22 – 29  |
| ROR, ROL  | 1        |
| FADD      | 3        |
| FMUL      | 5        |
| FDIV      | 10 – 24  |
| FSQRT     | 10 – 23  |
| FSIN      | 47 – 106 |
| FPTAN     | 130      |

Jsou uvedeny minimální hodnoty latence (kdy nenastane cache miss, závislost mezi instrukcemi, NaN nebo jiné zvláštní případy apod.) jako počet hod. taktů jádra. [www.agner.org/optimize/instruction\\_tables.pdf](http://www.agner.org/optimize/instruction_tables.pdf)

# Operace FP

- Číslo  $X$  s pohyblivou řádovou čárkou  $X = M_X.B^{E_X}$  zapíšeme jako dvojici  $(M_X, E_X)$ , kde **mantisa**  $M_X$  je ve dvojkovém doplňkovém kódu, nebo v přímém kódu se znaménkem na  $n_M$  bitech, **exponent**  $E_X$  je v kódu s posunutím na  $n_E$  bitech. Třetím definičním údajem je hodnota **základu B**.
- Základní aritmetické operace pro dvojici čísel  $X, Y$  s plovoucí čárkou jsou:

$$X + Y = (M_X . 2^{E_X - E_Y} + M_Y) . 2^{E_Y}, \text{ kde } E_X \leq E_Y$$

$$X - Y = (M_X . 2^{E_X - E_Y} - M_Y) . 2^{E_Y}, \text{ kde } E_X \leq E_Y$$

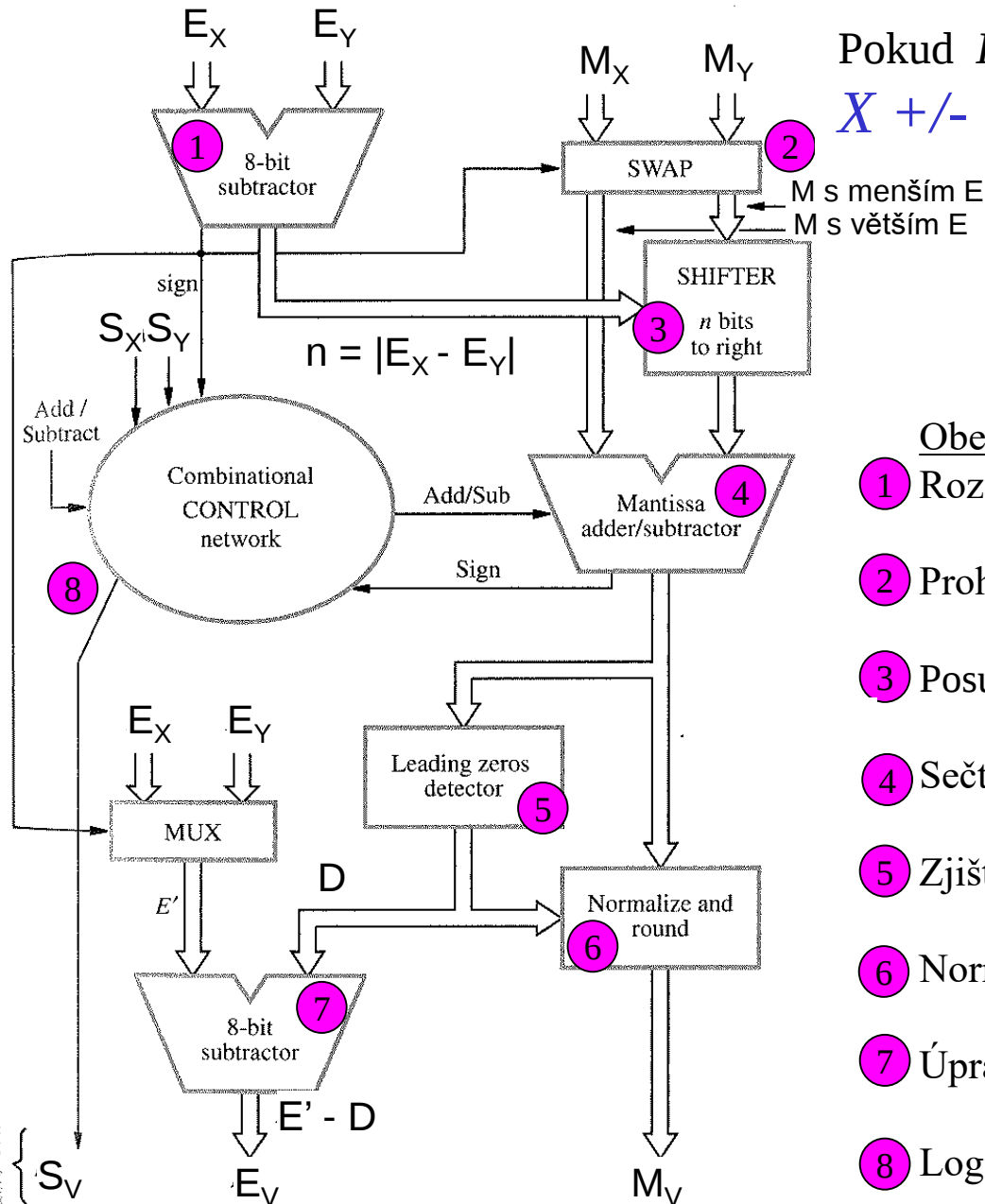
$$X * Y = (M_X . M_Y) . 2^{E_X + E_Y}$$

$$X : Y = (M_X : M_Y) . 2^{E_X - E_Y}$$

# Požadované operace

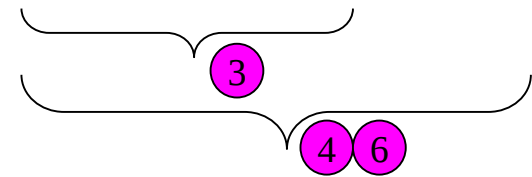
- Pro sčítání a odčítání:
  - 1. Vypočte se v pevné čárce rozdíl  $E_X - E_Y$
  - 2. Posune se  $M_X$  o  $E_X - E_Y$  bitů (tj. doprava, pokud je  $E_X \leq E_Y$ )
  - 3. Vypočte se v pevné čárce  $M_X \cdot 2^{E_X - E_Y} \pm M_Y$
- Dále je zapotřebí provést **normalizaci a zaokrouhlení výsledku** (viz přednáška Reprezentace dat). Necht'  $V = (M_V, E_V)$  označuje výsledek. Normalizovat výsledek znamená posouvat mantisu vlevo (vpravo) a podle toho zmenšovat (zvětšovat) exponent  $E_V$  tak dlouho, až se do sledovaného bitu ( $v_0$  nebo  $v_1$ ) dostane 1.
- **Sčítání exponentů** se provádí v binární sčítačce s korekcí, nebo ve speciální sčítačce pro posunutý kód.
- **Sčítání mantis** se provádí v binární sčítačce se šířkou  $n_M + 2$  bitů s doplněným záchytným klopným obvodem  $S$  (viz *Sticky bit* v přednášce o Reprezentaci dat).
- V případě násobení (dělení) v FP se využije pro násobení (dělení) mantis dedikovaná násobička (dělička) mantis pracující v FX.

# Princip obvodové realizace sčítačky/odčítačky FP



Pokud  $E_X \leq E_Y$ , potom

$$X \pm Y = (M_X \cdot 2^{\underbrace{E_X - E_Y}_{(1)}} \pm M_Y) \cdot 2^{E_Y} \quad (7)$$



Obecný postup (nevíme, zda  $E_X \leq E_Y$ ):

- 1 Rozdíl exponentů, nastavení *sign* a  $n$
- 2 Prohození mantis podle *sign*
- 3 Posuv mantisy o  $n$  bitů vpravo
- 4 Sečtení/odečtení mantis podle *Add/Sub*
- 5 Zjištění počtu nul  $D$  v horních bitech mantisy
- 6 Normalizace mantisy, detekce spec. případů
- 7 Úprava exponentu podle  $D$  a *sign*
- 8 Logika pro výpočet znaménka

# Obvodová realizace operací v FP

- Pro operace s pohyblivou čárkou prováděné v rámci uvedených algoritmů by bylo možno teoreticky použít sčítačku a násobičku ALU s pevnou čárkou. V praxi se to však takto neděje. Obvody aritmetiky s pohyblivou čárkou jsou konstruovány jako zcela **nezávislá** jednotka s vlastním řadičem.
- Dělení čísel s pohyblivou čárkou a celou řadu dalších operací (sinus, kosinus atd.) je možné v HW provádět **iteračními algoritmy**.

# Newtonův iterační algoritmus

Na základě odhadu  $x_i$  hledáme přesnější odhad  $x_{i+1}$  v bodě průsečíku tečny funkce  $f$  s osou  $x$ . Rovnice přímky procházející bodem  $f(x_i)$  je

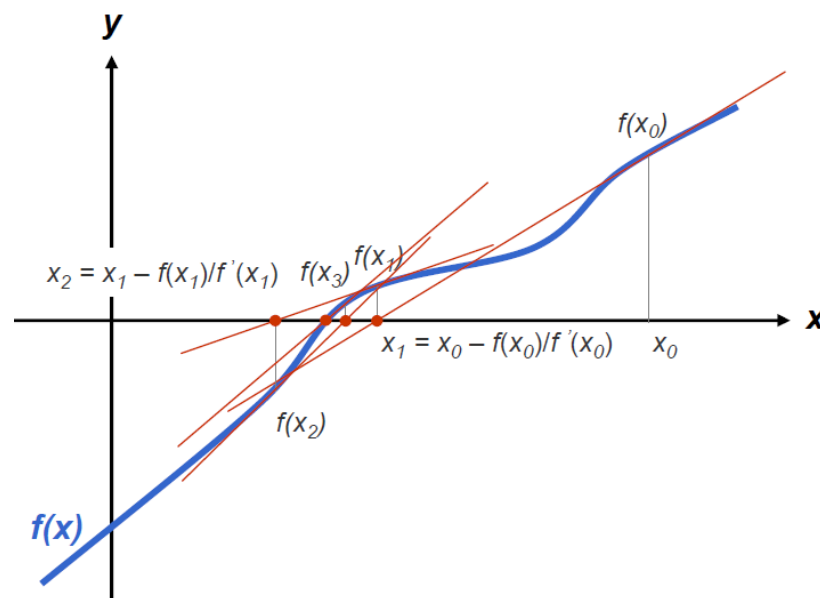
$$y - f(x_i) = f'(x_i)(x - x_i).$$

Pokládáme-li přímku za aproximaci funkce  $f(x)$ , můžeme psát

$$f(x_{i+1}) - f(x_i) = f'(x_i)(x_{i+1} - x_i).$$

V průsečíku tečny s osou  $x$  je  $f(x_{i+1}) = 0$ , takže odtud dostáváme iterační vzorec

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$



# Newtonův algoritmus - dělení

Iterační vzorec

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

použijeme pro případ dělení. Máme-li dělit číslem  $b$ , zvolíme  $f(x) = 1/x - b$ , a hledáme bod průchodu této funkce osou  $x$ , tedy bod  $x$ , kde  $f(x) = 0$ , takže pak platí  $1/x = b$ , resp.  $x = 1/b$ . Operaci dělení číslem  $b$  nahradíme násobením číslem  $1/b$ .

Derivace

$$f'(x) = -1/x^2, \text{ odtud}$$

$$\begin{aligned} x_{i+1} &= x_i - (1/x_i - b)/(-1/x_i^2) = \\ &= x_i + x_i - bx_i^2 = x_i (2 - bx_i) \end{aligned}$$

Rychlost konvergence závisí na volbě  $x_0$  a je typicky kvadratická.

# Newtonův algoritmus dělení - prakticky

**Postup výpočtu  $a / b$  je následující:**

1. Posune se  $b$  tak, aby padlo do intervalu  $<1, 2)$  a pomocí tabulky odhadů zvolíme první odhad  $x_0$ .
2. Provedeme krok iteračního výpočtu  $x_{i+1} = x_i (2 - bx_i)$ . Krok 2 opakujeme tak dlouho, až se dosáhne požadované přesnosti na  $p$  bitů, kdy je relativní chyba  $(x_i - 1/b)/(1/b) = 2^{-p}$ . V následujícím kroku  $i+1$  je relativní chyba
$$(x_{i+1} - 1/b)/(1/b) = 2^{-2p}$$
3. Výsledek  $n$ -té iterace  $x_n$  vynásobíme číslem  $a$ , výsledek  $x_n \cdot a$  posuneme o odpovídající počet bitů podle kroku 1.

# Newtonův algoritmus dělení - příklad

Spočtete binárně  $1/b$  pro  $b = 20$ .  $(20)_{10} = (10100)_2$      $x_{i+1} = x_i (2 - bx_i)$

- řádovou čárku posuneme tak, aby  $b \in <1, 2)$ , tedy:

$10100 \rightarrow 1,0100$  (o 4 bity doleva)

- zvolíme např.  $x_0 = 1$ , potom:

$$x_1 = x_0(2 - bx_0) = 1.(10 - 1,01) = 1.0,11 = 0,11$$

$$x_2 = 0,11(10 - 1,01.0,11) = 0,11(10 - 0,1111) = 0,11.1,0001 = 0,110011$$

$$\begin{aligned} x_3 &= 0,110011(10 - 1,01.0,110011) = 0,110011(10 - 0,11111111) = \\ &= 0,110011.1,00000001 = 0,11001100110011 \end{aligned}$$

atd.

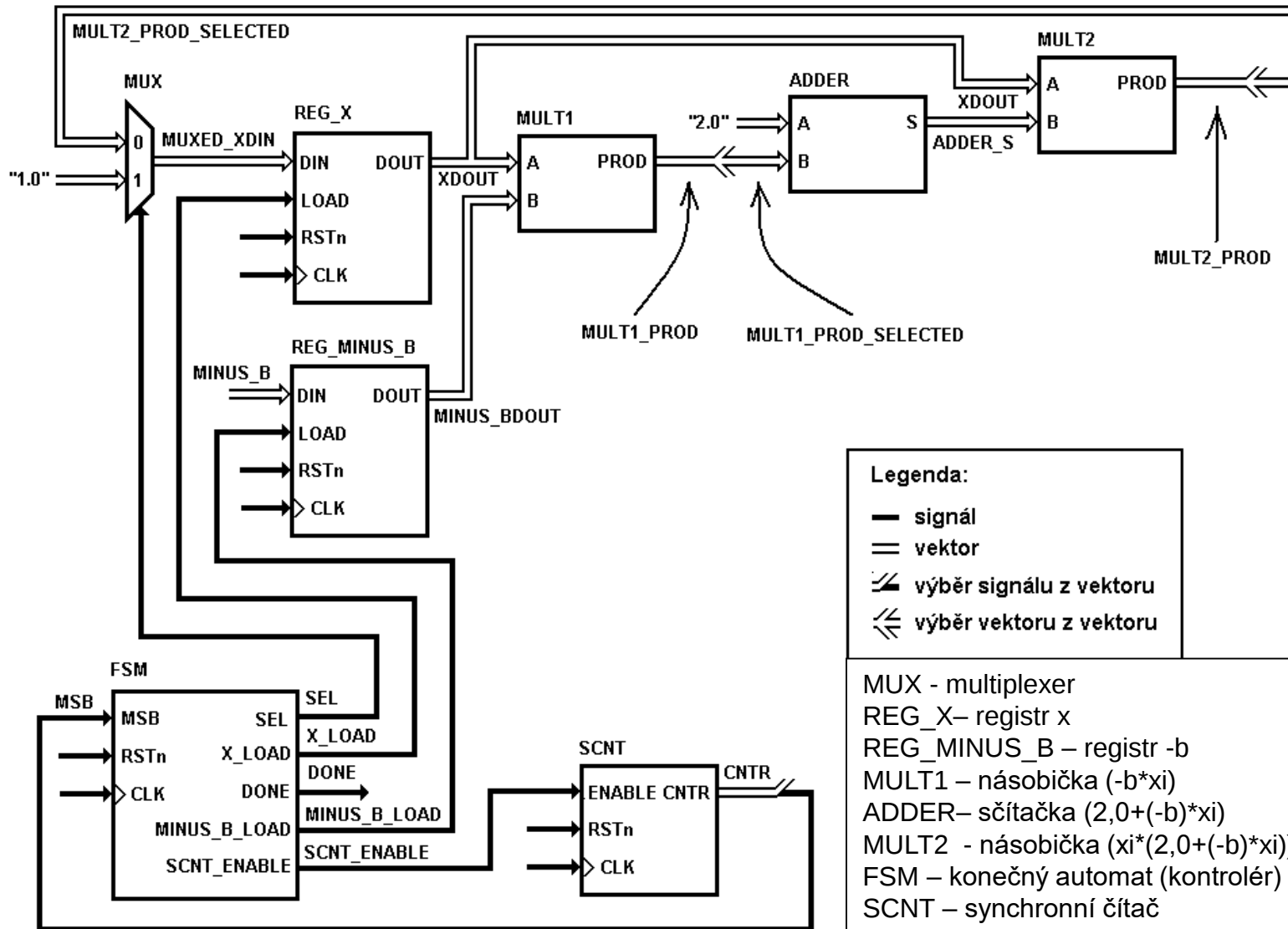
- řádovou čárku posuneme o 4 bity doleva:  $0,000011001100110011$

Ověření správnosti:  $1/20 = 0,05$  a

$$(0,000011001100110011)_2 = (0,051952362)_{10}$$

# Newtonův algoritmus dělení v HW

$$x_{i+1} = x_i (2 - bx_i)$$



# Další funkce pomocí iteračních algoritmů

Zde platí, že postupy vhodné pro programovou implementaci, jako MacLaurinův rozvoj nebo Čebyševovy polynomy atd., nemusí být pro obvodovou realizaci iteračních výpočtů výhodné, a proto byla odvozena řada modifikovaných nebo nových algoritmů, např. **Goldschmidtův algoritmus** apod.

***Všeobecná zásada je najít co nejrychlejší algoritmy, založené pouze na operacích sčítání (odčítání), posuvů a případně i násobení.***

# CORDIC

Algoritmus CORDIC ([Coordinate Rotational Digital Computer](#)) publikoval J. E. Volder v r. 1959. Následně byl zobecněn pro další typy výpočtů.

**Myšlenka:** Využitím jednoho algoritmu můžeme počítat řadu matematických funkcí pouhým vyčíslováním funkce ve tvaru

$$a \pm b \cdot 2^{-i}$$

tedy pomocí součtů, rozdílů a bitových posunů (tzn. rychlá a levná HW implementace).

**Použití:** CORDIC je používán typicky ve vestavěných zařízeních s jednoduchým procesorem (kapesní kalkulačky, jednočipové mikrokontrolery, apod.), čímž umožňuje efektivně počítat mnoho funkcí. Používá se také např. v koprocesech Intel počínaje I 8087, pro číslicovou Fourierovu transformaci, číslicovou filtraci signálů apod.

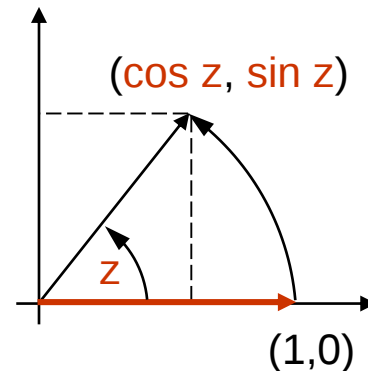
# CORDIC – rotační režim

Algoritmus lze provozovat ve dvou režimech – rotačním a vektorovém. Každý režim umožňuje vypočítat jiné funkce.

## Rotační režim

Princip: rotací vektoru  $(1,0)$  o úhel  $z$  získáváme hodnotu  $\cos z$  a  $\sin z$ .

Rotaci provádíme postupným přičítáním nebo odčítáním vhodně zvolených úhlů (postupně se zmenšující hodnoty) tak, aby získaná hodnota konvergovala k požadované hodnotě  $z$ .



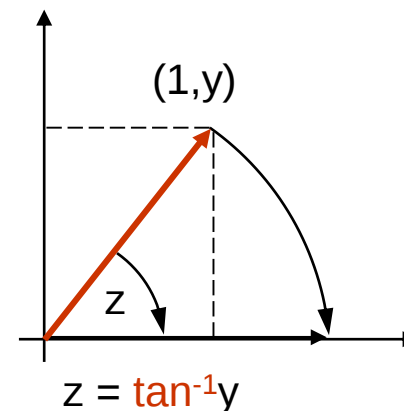
1. počátek v  $(1,0)$ ,
2. rotace tak, aby úhel byl  $z$ ,
3.  $x = \cos z$  a  $y = \sin z$

# CORDIC – vektorový režim

Algoritmus lze provozovat ve dvou režimech – rotačním a vektorovém. Každý režim umožňuje vypočítat jiné funkce.

## Vektorový režim

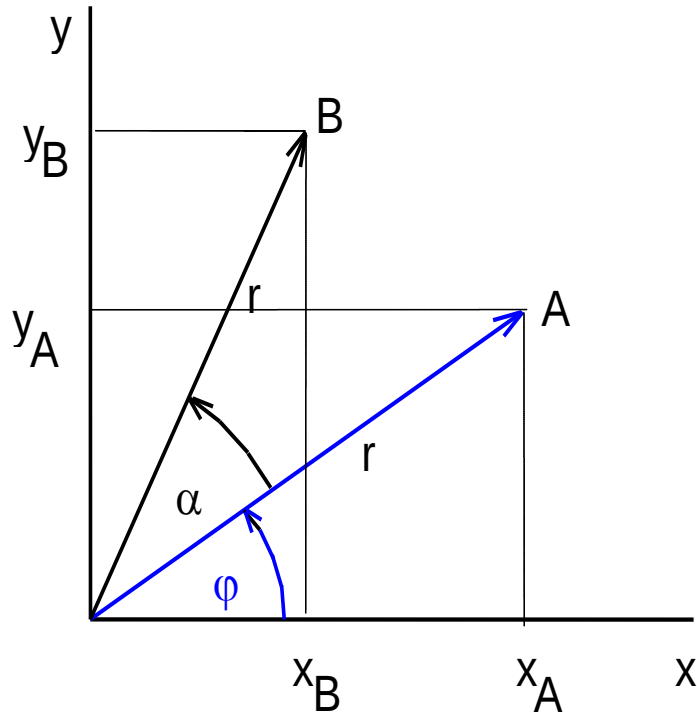
Princip: rotací vektoru  $(1,y)$  o úhel  $z$  tak, aby výsledný vektor byl rovnoběžný s osou  $X$ , získáváme hodnotu  $\arctan y$ .



Nulování hodnoty  $y$  provádíme postupným přičítáním nebo odčítáním vhodně zvolených hodnot tak, aby získaná hodnota postupně konvergovala k 0.

1. počátek v  $(1,y)$ ,
2. rotace o  $z$  tak, aby  $y$  bylo 0,
3.  $z$  odpovídá  $\tan^{-1} y$

# Rotace vektoru



**Přechod** od bodu A k bodu B lze obecně vyjádřit jako:

$$x_B = r \cdot \cos(\varphi + \alpha) = r \cdot \cos \varphi \cdot \cos \alpha - r \cdot \sin \varphi \cdot \sin \alpha \quad (1)$$

$$y_B = r \cdot \sin(\varphi + \alpha) = r \cdot \sin \varphi \cdot \cos \alpha + r \cdot \cos \varphi \cdot \sin \alpha \quad (2)$$

$$x_A = r \cdot \cos \varphi$$

$$y_A = r \cdot \sin \varphi$$

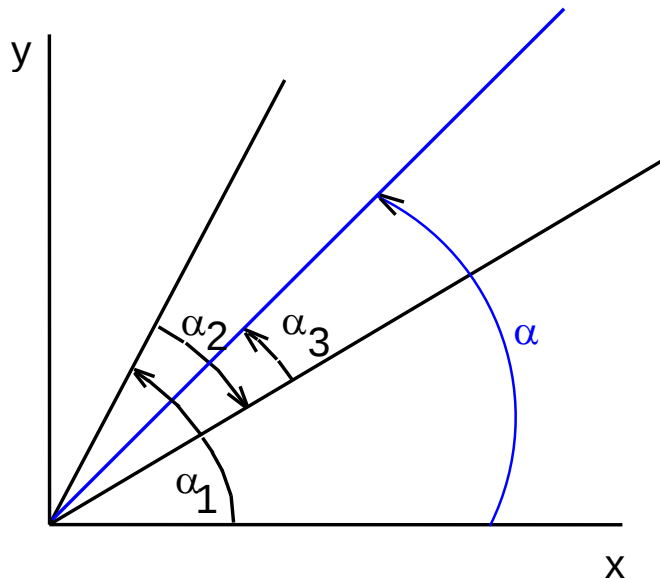
Dosadíme-li za  $r$  do (1)  $r = x_A / \cos \varphi$ ,  $r = y_A / \sin \varphi$  pořadě, a do (2) v opačném pořadí, dostaneme výrazy

$$x_B = \cos \alpha (x_A - y_A \cdot \operatorname{tg} \alpha)$$

$$y_B = \cos \alpha (y_A + x_A \cdot \operatorname{tg} \alpha)$$

# Rotace vektoru

Požadovaný úhel natočení  $\alpha$  můžeme složit z  $n$  úhlů s kladným i záporným znaménkem (viz obrázek), tedy z orientovaných úhlů  $\alpha_i'$ .



Příklad:

rotace vektoru orientovaného do osy x o úhel  $\alpha$  dosažená pomocí tří iterací s postupně se snižujícími absolutními hodnotami úhlů  $\alpha_i$

# CORDIC – rotace vektoru

Výsledný úhel natočení  $\alpha = \alpha_0' + \alpha_1' + \dots + \alpha_{n-1}'$

Postup iteračního výpočtu je pak následující:

Položíme  $x_0 = x_A$ ,  $y_0 = y_A$  a určíme  $x_{i+1}, y_{i+1}$  postupně pro  $i = 0, 1 \dots n-1$ .

Dostáváme pak iterační vzorce

$$x_{i+1} = \cos \alpha_i' \cdot (x_i - y_i \cdot \operatorname{tg} \alpha_i') \quad (3)$$

$$y_{i+1} = \cos \alpha_i' \cdot (y_i + x_i \cdot \operatorname{tg} \alpha_i') \quad (4)$$

**Problém:** uvedený vztah vyžaduje implementovat obecné násobičky:

- každý krok obsahuje násobení odlišným koeficientem  $\operatorname{tg} \alpha_i'$
- každý krok obsahuje násobení odlišným koeficientem  $\cos \alpha_i'$

**Řešení:**

- vhodná volba úhlů  $\alpha_i'$  tak, aby hodnota  $\operatorname{tg} \alpha_i'$  byla mocninou dvou
- zavedení pevného počtu iterací a odložení násobení koeficienty  $\cos \alpha_i'$

# CORDIC – rotace vektoru

## Vhodná volba úhlů $\alpha_i'$

Ve dvojkové soustavě volíme takové úhly  $\alpha_i'$ , že platí  $\text{tg } \alpha_i' = \pm 2^{-i}$ .

Tato volba nemá vliv na konvergenci, neboť je-li  $\text{tg } \alpha_i'$  klesající posloupnost, pak také  $\alpha_i'$  je klesající posloupnost (viz tabulka dále).

Tím se v rovnicích (3), (4) nahradí násobení **posuvem o  $i$  bitů** (vpravo).

## Odložení násobení

Další zjednodušení provedeme odložením násobení hodnotami  $\cos \alpha_i'$  nakonec výpočtu, kdy výsledek vynásobíme hodnotou (agregační konstanta)

$$K_{n-1} = \cos \alpha_0 \cdot \cos \alpha_1 \dots \cos \alpha_{n-1} = 0,60725\dots$$

kde  $\alpha_i = |\alpha_i'|$ , protože platí  $\cos \alpha_i' = \cos |\alpha_i'|$ .

Hodnota agregační konstanty závisí pouze na počtu kroků, který je fixní a je volen na základě požadované přesnosti.

# Tabulka úhlů CORDIC

Pro  $i = 0, 1, \dots$  sestavíme tabulku hodnot  $\alpha_i$ , pro něž platí  $\alpha_i = \arctg 2^{-i}$

| i   | $\alpha$ [°] | tg $\alpha$ | cos $\alpha$ | K        |
|-----|--------------|-------------|--------------|----------|
| 0   | 45,000       | 1           | 0,707107     | 0,707107 |
| 1   | 26,565       | 0,5         | 0,894427     | 0,632456 |
| 2   | 14,036       | 0,25        | 0,970143     | 0,613572 |
| 3   | 7,125        | 0,125       | 0,992278     | 0,608834 |
| 4   | 3,576        | 0,0625      | 0,998053     | 0,607648 |
| 5   | 1,790        | 0,03125     | 0,999512     | 0,607352 |
| 6   | 0,895        | 0,015625    | 0,999878     | 0,607278 |
| 7   | 0,448        | 0,0078125   | 0,999969     | 0,607259 |
| ... | ...          | ...         | ...          | ...      |

Agregační konstanta K konverguje se zvětšujícím se počtem iterací k hodnotě 0,60725293.

# CORDIC - finální podoba

Dosazením dostaneme konečný tvar iteračních vzorců

$$x_{i+1} = x_i - (+) y_i 2^{-i}$$

$$y_{i+1} = y_i + (-) x_i 2^{-i}$$

$$z_{i+1} = z_i + (-) \alpha_i$$

Rotační režim: vyjdeme-li z vektoru  $x_0 = 1$ ,  $y_0 = 0$ , přičemž  $z_0 = \alpha$ , pak aplikací iteračních vzorců tak, aby  $z_n \rightarrow 0$  (tzn. natočili jsme vektor o úhel  $\alpha$ ), dostáváme po  $n$  iteracích hodnoty

$$x_n = \cos \alpha / K_{n-1}, y_n = \sin \alpha / K_{n-1}.$$

Abychom získali požadované hodnoty kosinu a sinu, musíme obě složky násobit agregační konstantou  $K_{n-1}$ .

**Optimalizace:** Položíme-li  $x_0 = K_{n-1}$ ,  $y_0 = 0$ , vyhneme se závěrečnému násobení a výsledek je přímo roven

$$x_n = \cos \alpha, y_n = \sin \alpha.$$

**Poznámka:** Pro úhly větší než  $90^\circ$  využíváme symetrie a opakování grafu goniometrických funkcí.

# Výpočet sin a cos pomocí CORDIC v jazyce Python

```
def cordic(alpha, n=16): #úhel alpha, počet iterací

    alphatab = [math.atan(2**(-i)) for i in range(n)]    Tabulka úhlů

    K = reduce(lambda x,y: x*y, [math.cos(a) for a in alphatab])
    x, y, z = K, 0.0, alpha                               Výpočet agregační konstanty

    for i in range(n):
        if z < 0:
            xn = x + y * 2**(-i)
            yn = y - x * 2**(-i)
            z += alphatab[i]
        else:
            xn = x - y * 2**(-i)
            yn = y + x * 2**(-i)
            z -= alphatab[i]
        x,y = xn,yn

    return (x, y) # cos(alpha), sin(alpha)
```

Na základě hodnoty znaménka  
přičítáme nebo odčítáme úhel z  
tabulky úhlů.

Po  $n$  iteracích získáváme  $x$  a  $y$   
hodnoty  $\cos(\alpha)$  a  $\sin(\alpha)$

# Př. S využitím sedmi iterací algoritmu CORDIC vypočtete $\sin(28.027^\circ)$ a $\cos(28.027^\circ)$

Rotujeme vektor (1,0) o úhel  $28.027^\circ$ , průběžný úhel označme z.

$$X_0 = 1 \quad Y_0 = 0 \quad z_0 = 28.027^\circ$$

Odečti úhel  $\alpha_0 = 45^\circ \rightarrow z_1 = z_0 - \alpha_0 = 28.027 - 45 = -16.973^\circ$

Průběžný úhel je kladný, úhel  $\alpha_0$  odečítáme

$$X_1 = X_0 - Y_0 / 1 = 1 - 0 / 1 = 1$$

$$Y_1 = Y_0 + X_0 / 1 = 0 + 1 / 1 = 1$$

Přičti úhel  $\alpha_1 = 26.565^\circ \rightarrow z_2 = z_1 + \alpha_1 = -16.973 + 26.565 = 9.592^\circ$

Průběžný úhel byl záporný, přičítáme a měníme znaménka

$$X_2 = X_1 + Y_1 / 2 = 1 + 1 / 2 = 1.5$$

$$Y_2 = Y_1 - X_1 / 2 = 1 - 1 / 2 = 0.5$$

Odečti úhel  $\alpha_2 = 14.036^\circ \rightarrow z_3 = z_2 - \alpha_2 = 9.592 - 14.036 = -4.444^\circ$

$$X_3 = X_2 - Y_2 / 4 = 1.5 - 0.5 / 4 = 1.375$$

$$Y_3 = Y_2 + X_2 / 4 = 0.5 + 1.5 / 4 = 0.875$$

Přičti úhel  $\alpha_3 = 7.125^\circ \rightarrow z_4 = z_3 + \alpha_3 = -4.444 + 7.125 = 2.680^\circ$

$$X_4 = X_3 + Y_3 / 8 = 1.375 + 0.875 / 8 = 1.484375$$

$$Y_4 = Y_3 - X_3 / 8 = 0.875 - 1.375 / 8 = 0.703125$$

Odečti úhel  $\alpha_4 = 3.576^\circ \rightarrow z_5 = z_4 - \alpha_4 = 2.680 - 3.576 = -0.895^\circ$

$$X_5 = X_4 - Y_4 / 16 = 1.484375 - 0.703125 / 16 = 1.440429$$

$$Y_5 = Y_4 + X_4 / 16 = 0.703125 + 1.484375 / 16 = 0.795898$$

Přičti úhel  $\alpha_5 = 1.790^\circ \rightarrow z_6 = z_5 + \alpha_5 = -0.895 + 1.790 = 0.894^\circ$

$$X_6 = X_5 + Y_5 / 32 = 1.440429 + 0.795898 / 32 = 1.465300$$

$$Y_6 = Y_5 - X_5 / 32 = 0.795898 - 1.440429 / 32 = 0.750884$$

Odečti úhel  $\alpha_6 = 0.895^\circ \rightarrow z_7 = z_6 - \alpha_6 = 0.894 - 0.895 = -0.0007^\circ$

$$X_7 = X_6 - Y_6 / 64 = 1.465300 - 0.750884 / 64 = 1.453567$$

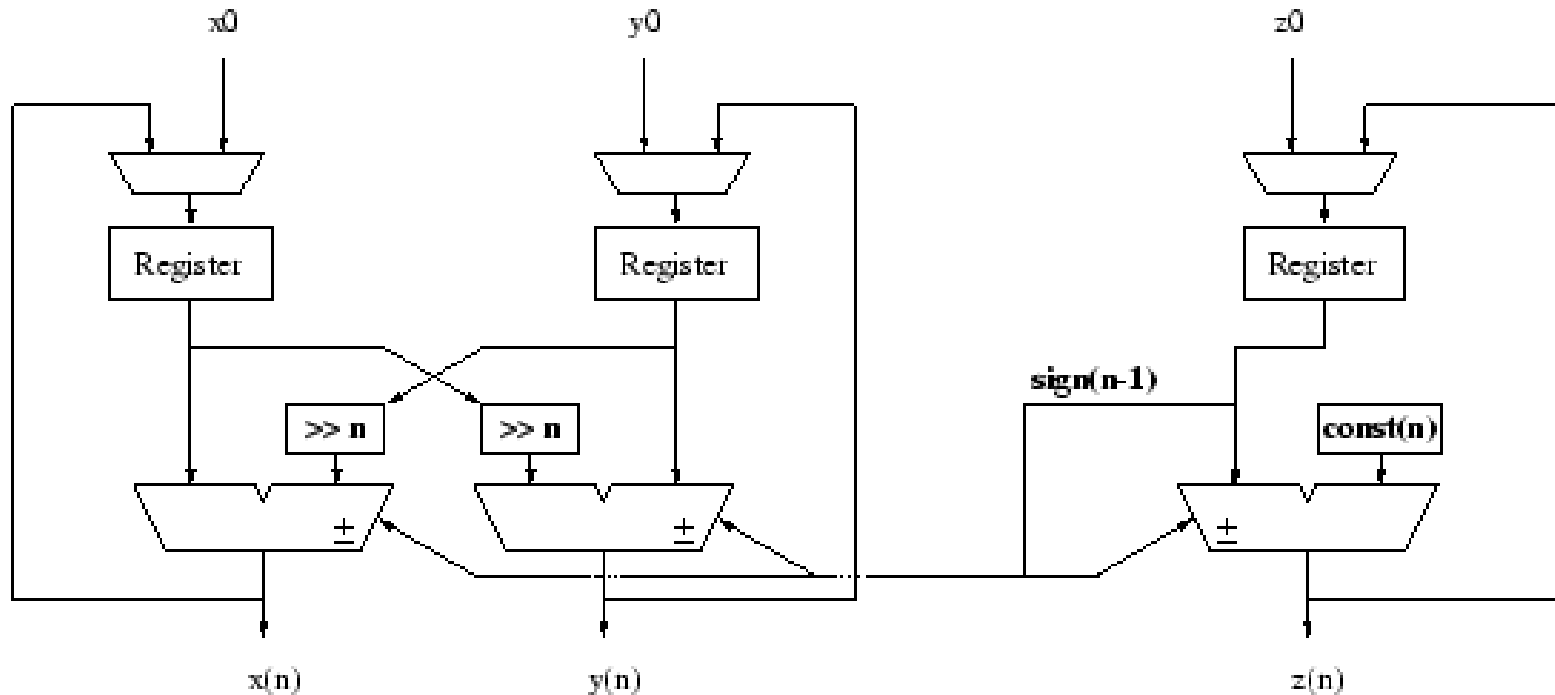
$$Y_7 = Y_6 + X_6 / 64 = 0.750884 + 1.465300 / 64 = 0.773779$$

Následuje násobení agregační konstantou odpovídající použitému počtu iterací, tzn.  $K_6 = 0.607278$  a získání požadovaných hodnot:

$$\sin(28.027^\circ) = 0.607278 * Y_7 = 0.46990$$

$$\cos(28.027^\circ) = 0.607278 * X_7 = 0.88272$$

# CORDIC – v HW (základní implementace)



úhel v kroku  $i$

$d_i$  je znaménko úhlu

$$x_{i+1} = x_i + y_i \cdot d_i \cdot 2^{-i}$$

$$y_{i+1} = y_i - x_i \cdot d_i \cdot 2^{-i}$$

$$z_{i+1} = z_i - d_i \cdot \arctanTab[i]$$

$$d_i = -1 \text{ pokud } z_i < 0, \text{ jinak } +1$$

# Zobecněný CORDIC

$$x_{i+1} = x_i - \mu d_i y_i 2^{-i}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i}$$

$$z_{i+1} = z_i + d_i \alpha_i$$

- $\mu = 1$  kruhové rotace (sin, cos)  $\alpha_i = \tan^{-1} 2^{-i}$
- $\mu = 0$  lineární rotace  $\alpha_i = 2^{-i}$
- $\mu = -1$  hyperbolické rotace  $\alpha_i = \tanh^{-1} 2^{-i}$
- Přímý výpočet funkcí:  
sin, cos,  $\tan^{-1}$ , sinh, cosh,  $\tanh^{-1}$ ,  $\tan^{-1}(y/x)$ ,  $y + xz$ ,  $\sqrt{x^2 + y^2}$ ,  $e^z$ ,  
násobení, dělení
- Nepřímý výpočet funkcí:  
tan, tanh, ln, log,  $a^b$ ,  $\cos^{-1}$ ,  $\sin^{-1}$ ,  $\cosh^{-1}$ ,  $\sinh^{-1}$ ,  $\sqrt{a}$

# Násobení a dělení pomocí algoritmu CORDIC (lineární rotace)

```
def cordicdiv (a,b, n=16):  
    x, y, z = b, a, 0.0  
  
    for i in range(n):  
        if y < 0:  
            y += x * 2**(-i)  
            z -= 2**(-i)  
        else:  
            y -= x * 2**(-i)  
            z += 2**(-i)  
  
    return z # a/b
```

Využíváme vektorový režim

```
def cordicmul (a,b, n=16):  
    x, y, z = a, 0.0, b  
  
    for i in range(n):  
        if z >= 0:  
            y += x * 2**(-i)  
            z -= 2**(-i)  
        else:  
            y -= x * 2**(-i)  
            z += 2**(-i)  
  
    return y # a*b
```

Využíváme rotační režim

Konec kapitoly o implementaci  
aritmetických operací v HW