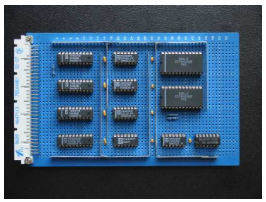
Digital- och datorteknik



Föreläsning #24

Biträdande professor Jan Jonsson

Institutionen för data- och informationsteknik
Chalmers tekniska högskola

Multiplikation och division

Allmänt

Behovet av processorinstruktioner för multiplikation och division är mycket stort i moderna datortillämpningar, t ex för digital signalbehandling och datorgrafik. Därför har processorer för sådana tillämpningar (t ex StarCore DSP, NVidia GPU) utrustats med speciella hårdvaruenheter som gör dessa operationer mycket snabba. Även den generella processorn ARM Cortex har processorinstruktioner för multiplikation och division.

Om man har behov av att utföra multiplikation eller division i sitt program, men processorn inte tillhandahåller motsvarande instruktioner, måste man istället skapa en programvarulösning.

Multiplikation och division

Multiplikation

Vi kommer att visa den grundläggande metoden för hur binär multiplikation av heltal utan tecken går till. Denna metod kan, med en mindre modifikation, också användas för multiplikation av heltal med tecken (se KMP 4.8).

Exempel:

Multiplikand: $Y = 1011_2$

Multiplikator: $X = 1101_2$ $\quad X = (x_3x_2x_1x_0)_2$

Princip:

$$P = X \cdot Y = (x_3 \cdot 2^3 + x_2 \cdot 2^2 + x_1 \cdot 2^1 + x_0 \cdot 2^0) \cdot Y =$$
$$= x_3 \cdot Y \cdot 2^3 + x_2 \cdot Y \cdot 2^2 + x_1 \cdot Y \cdot 2^1 + x_0 \cdot Y \cdot 2^0$$

Multiplikation och division

Multiplikation – manuell räkning

					1	0	1	1	Y
					1	1	0	1	X
	1	1	1	1	1	0	1	1	
				0	0	0	0		
			1	0	1	1			
+		1	0	1	1				
	1	0	0	0	1	1	1	1	P

Kontroll:

$Y = 1011_2 = 11_{10}$

$X = 1101_2 = 13_{10}$

$P = 13_{10} \cdot 11_{10} = 143_{10} = 10001111_2$

Multiplikation och division

Multiplikation – programvarulösning

Multiplikationen kan utföras med en iterativ procedur där två enkla operationer används: addition och högerskift.

Partialprodukten initieras till 0. Vi undersöker multiplikatorns bitar iterativt, med början på minst signifikant bit. Vi antingen adderar noll till partialprodukten, då aktuell multiplikatorbit är 0, eller adderar multiplikanden, då aktuell multiplikatorbit är 1. Därefter skiftas partialprodukten ett steg åt höger, vilket är synonymt med att vikten på multiplikanden dubbleras inför nästa iteration.

Multiplikation och division

Multiplikation – programvarulösning (forts.)

Multiplikatorn X (8 bitar, ≠0) finns i variabel XVAR.

Multiplikanden Y (8 bitar, ≠0) finns i variabel YVAR.

Produkten P (16 bitar) finns efter operationen i register A (hög del) och variabel PLO (låg del).

Register A och variabel PLO har från början värdet 0.

Variabeln COUNT är en skifträknare, som från början har värdet 8.

Multiplikation och division

Multiplikation – programvarulösning (forts.)

```
LOOP    LSR     XVAR    Minst signifikant bit i X (lsb) till carry
        BCC     NOADD   Carry = 0. Ej addition

        ADDA    YVAR    Carry = 1. Addera Y till produkt hög byte

NOADD   RORA            Skifta produkten (16 bitar) ett steg åt höger
        ROR     PLO                        -  "  -

        DEC     COUNT   Ett skift mindre
        BNE     LOOP    8 skift? Nej.
        …               Ja. Färdigt
```

Multiplikation och division

Division

Vi kommer att visa den grundläggande metoden för hur binär division av heltal utan tecken går till. Denna metod kan inte användas för tal med tecken, utan andra metoder måste då användas (beskrivs ej i denna kurs).

Exempel:

Dividend: Y = 10011000_2

Divisor: X = 1010_2

Kvot: Q (8 bitar)

Rest: R (4 bitar)

Multiplikation och division

Division

Princip:

$$Y/X = Q + R/X \Leftrightarrow Y = Q \cdot X + R \Leftrightarrow R = Y - Q \cdot X \qquad (X \neq 0)$$

$$Q = (q_7 q_6 q_5 q_4 q_3 q_2 q_1 q_0)_2$$

$$R = Y - Q \cdot X =$$

$$= Y - (q_7 \cdot 2^7 + q_6 \cdot 2^6 + q_5 \cdot 2^5 + q_4 \cdot 2^4 + q_3 \cdot 2^3 + q_2 \cdot 2^2 + q_1 \cdot 2^1 + q_0 \cdot 2^0) \cdot X$$

$$= Y - q_7 \cdot X \cdot 2^7 - q_6 \cdot X \cdot 2^6 - \ldots - q_3 \cdot X \cdot 2^3 - q_2 \cdot X \cdot 2^2 - q_1 \cdot X \cdot 2^1 - q_0 \cdot X \cdot 2^0$$

Divisionen går till så att man väljer q_i för i = 7, 6, …, 2, 1, 0, så att R blir så litet som möjligt, men ≥ 0.

Om R blir = 0 under processen så är Y jämnt delbart med X och eventuella återstående q_i sätts till 0.

Multiplikation och division

Division – manuell räkning

	q_7	q_6	q_5	q_4	q_3	q_2	q_1	q_0					
Q =	0	0	0	0	1	1	1	1					
Y =	1	0	0	1	1	0	0	0	1	0	1	0	= X
-		1	0	1	0								
	0	1	0	0	1	0							
-			1	0	1	0							
		0	1	0	0	0	0						
-				1	0	1	0						
			0	0	1	1	0	0					
-					1	0	1	0					
				0	0	0	1	0	=	R			

Kontroll:

$Y = 10011000_2 = 152_{10}$

$X = 1010_2 = 10_{10}$

$Y / X = 152_{10} / 10_{10} = 15_{10} + 2/10_{10}$

$Q = 15_{10} = 00001111_2$

$R = 2_{10} = 0010_2$

Multiplikation och division

Division – programvarulösning

Divisionen kan utföras med en iterativ procedur där några enkla operationer används: subtraktion och vänsterskift, samt addition (vid återställning.)

Partialresten initieras med dividenden. Divisorn har initialt samma vikt som dividendens mest signifikanta bit. Kvotens bitar beräknas sedan iterativt, med början på mest signifikant bit. Efter subtraktion av divisorn från aktuell partialrest sätts aktuell kvotbit till 1 om skillnaden blir positiv; om skillnaden blir negativ sätts aktuell kvotbit till 0 och partialresten återställs genom addition av divisorn. Därefter skiftas partialresten ett steg åt vänster, vilket är synonymt med att vikten på divisorn halveras inför nästa iteration.

Multiplikation och division

Division – programvarulösning (forts.)

Divisorn X (8 bitar, ≠ 0) finns i variabel XVAR.

Dividenden Y(16 bitar) finns från början i variabel YHI (hög del) och variabel YLO (låg del).

Kvoten Q (16 bitar) finns efter operationen i variabel QHI (hög del) och variabel QLO (låg del).

Resten R (8 bitar) finns efter operationen i register A och variabel REST.

Register A har från början värdet 0.

Variabeln COUNT är en skifträknare, som från början har värdet 16.

Multiplikation och division

Division – programvarulösning (forts.)

```
LOOP    LSL     YLO     2*dividend (msb till C-flaggan)
        ROL     YHI          -" -

        ROLA            MS-bit till A (C-flaggan till A-registret)
        BCS     SUBTR   Overflow i A-reg! Subtrahera divisor från dividendens ms-bitar

        CMPA    Xval    Skall vi subtrahera divisorn från ms-bitar från dividenden?
        BHS     SUBTR   Ja!

        LSL     QLO     Nej, skifta kvoten vänster med nolla in från höger
        ROL     QHI     Hög byte
        BRA     NEXT
```

Multiplikation och division

Division – programvarulösning (forts.)

```
SUBTR   SUBA    Xval    Utför subtraktionen. Blir korrekt även vid overflow!
        ORCC    #1      Sätt carry (ger etta i kvoten)
        ROL     QLO     Skifta kvoten vänster med etta in från höger
        ROL     QHI                          -" -

NEXT    DEC     COUNT   Skifträknare
        BNE     LOOP    Färdigt? Nej

        STA     REST    Ja. Uppdatera rest
         …              Färdigt
```

Talrepresentation

Fixpunkt (”fixed point”)

Det vi hittills arbetat med i kursen är s k fixpunktsystem, där en binärpunkt har antagits ligga på en fast plats i ett givet binärt kodord. För heltal är den naturliga platsen för binärpunkten till höger om den minst signifikanta biten. Generellt sett kan dock binärpunkten ligga placerad var som helst i kodordet.

Exempel:

Det binära kodordet 10111_2 kan tolkas olika beroende på var i kodordet binärpunkten antas ligga:

$(10111.)_2 = (23)_{10}$ eller

$(101.11)_2 = (5{,}75)_{10}$ eller

$(1.0111)_2 = (1{,}4375)_{10}$

Talrepresentation

Fixpunkt (”fixed point”)

En fördel med fixpunktsystemet är att det kan användas i även de enklaste datorer så länge som de har stöd för heltalsaritmetik.

En nackdel med fixpunktsystemet är att det inte är lämpligt för att representera system med tal som kan vara antingen väldigt stora eller väldigt små.

Exempel:

För att kunna representera både talet $2^{128} \approx 1{,}7 \cdot 10^{38}$ och talet $2^{-126} \approx 1{,}2 \cdot 10^{-38}$ skulle ett binärt kodord i ett fixpunktsystem behöva bestå av hela 254 bitar (128 + 126), d v s nästan 32 bytes.

Talrepresentation

Flyttal (”floating point”)

Nackdelen med fixpunktsystemet identifierades tidigt i datorns historia, och ett s k flyttalssystem föreslogs redan i början av 1900-talet för mekaniska räknemaskiner. Tysken Konrad Zuse byggde den första elektroniska datorn med flyttal på 1940-talet.

Med ett flyttalssystem kan man representera såväl väldigt stora tal som väldigt små tal, utan att det binära kodordet blir orimligt stort.

Exempel:

För att kunna representera både talet $2^{128} \approx 3{,}4 \cdot 10^{38}$ och talet $2^{-126} \approx 1{,}2 \cdot 10^{-38}$ skulle ett binärt kodord i ett flyttalssystem bara behöva bestå av 32 bitar, d v s 4 bytes.

Talrepresentation

Flyttal (”floating point”)

Hemligheten med ett flyttalssystemet är inte låsa binärpunkten till en fast plats i talets kodord utan istället låta dess position vara så nära början av talet som möjligt.

För att detta skall kunna realiseras måste kodordet också innehålla en del som indikerar var denna position är. Det önskade resultatet fås genom att man normaliserar talet och samtidigt multiplicerar det med en lämplig potens av 2.

Exempel:

$(10111.)_2 = (23)_{10}$ skrivs istället som $(1.0111)_2 \cdot 2^4 = (1{,}4375 \cdot 16)_{10}$

$(101.11)_2 = (5{,}75)_{10}$ skrivs istället som $(1.0111)_2 \cdot 2^2 = (1{,}4375 \cdot 4)_{10}$

$(1.0111)_2 = (1{,}4375)_{10}$ skrivs som $(1.0111)_2 \cdot 2^0 = (1{,}4375 \cdot 1)_{10}$

Talrepresentation

Format för flyttal (IEEE 754 standard)

IEEE definierade 1985 ett standardformat (IEEE 754) för representation av flyttal, som används av i stort sett alla moderna datorer. Standarden reviderades senast 2008.

Standarden utgår från att ett tal X skrivs på formen

$$X = k \cdot 2^e$$

där k är mantissa (el. signifikand) och e är exponent.

Mantissan antas vara normaliserad, d v s $(1)_2 \leq s < (10)_2$.

Minsta värde på mantissan är alltså $(1.0000000\ldots)_2$

Största värde på mantissan är alltså $(1.11111111\ldots)_2$

Talrepresentation

Format för flyttal (IEEE 754 standard)

Ett flyttal lagras i ett kodord enligt IEEE 754 i formatet

s / c / f

s (1 bit): teckenbit, där 0 motsvarar ’+’ och 1 motsvarar ’−’.

c (n bitar): karakteristika, ett heltal ≥ 0 enligt $c = e + 2^{n-1} - 1$

f (m bitar): fraction, mantissa med mest signifikanta bit borttagen

Kommentarer:

- Karakteristikan c är alltså exponenten e uttryckt i excess-kod
- Eftersom mantissan antas vara normaliserad kommer dess mest signifikanta bit alltid att ha värdet 1. Denna bit behöver därför inte lagras i kodordet.

Talrepresentation

Format för flyttal (IEEE 754 standard)

Ett flyttal lagras i ett kodord enligt IEEE 754 i formatet

s / c / f

Högsta värde ($2^n - 1$) och lägsta värde (0) på karakteristikan är reserverade för ∞ respektive 0. Antag t ex n = 8:

± ∞ : s / 255 / 0

± 0 : s / 0 / 0

För vanliga tal gäller alltså att $1 \leq c \leq 254$.

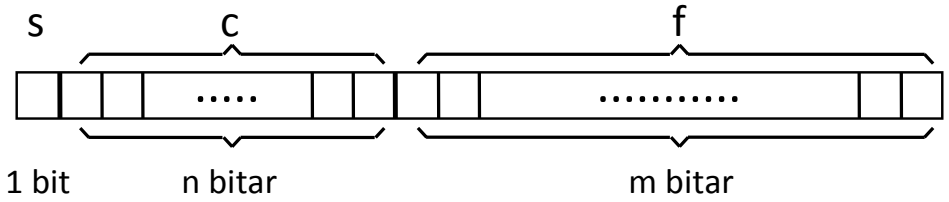
Användbart område för exponenten blir därför $-126 \leq e \leq 127$.

Minsta belopp blir $1.0000000\ldots0 \cdot 2^{-126} = 2^{-126}$

Största belopp blir $1.1111111\ldots1 \cdot 2^{+127} \approx 2^{+128}$

Talrepresentation

Format för flyttal (IEEE 754 standard)



Format	*enkelt*	*dubbelt*	*fyrdubbelt*
ordlängd (bitar)	32	64	128
m = antal bitar i f	23	52	112
antal signifikanta decimala siffror	7	15	34
n = antal bitar i c	8	11	15
$exponent_{min} = -(2^{n-1}-1)$	-127	-1023	-16383
$exponent_{max} = 2^{n-1}$	+128	+1024	+16384
talområde, $\|N\|$	$[2^{-126}, 2^{128})$ $\approx (10^{-38}, 10^{38})$	$[2^{-1022}, 2^{1024})$ $\approx (10^{-308}, 10^{308})$	$[2^{-16382}, 2^{16384})$ $\approx (10^{-4932}, 10^{4932})$