Целочислено деление. Логически структури и операционни схеми за апаратно изчисление на частно и остатък

Димитър Тянев

Резюме: Представен е проект на изчислител за бързо изпълнение на операция деление на цели числа със знак. Крайният резултат от синтеза представлява цялостна и единствена комбинационна схема. Синтезът изисква представяне на теоретичното основание за операция деление и произтичащите от него алгоритми за изчисляване на частното и на остатъка. Операндите и резултатите на операцията са числа, представени в допълнителен код. В първата част на изложението е изложен синтезът на логическата структура и на комбинационната схема за изчисление на първия резултат – частното. Във втората част на изложението е представен синтезираният алгоритъм и логическата схема за изчисление на втория резултат – остатъка. Цялата логическа схема за изпълнение на операция деление, описана в заключение показва, че тази операция е изпълнима в течение на времето за превключване на комбинационната схема. Така изчислението на двата резултата е възможно най-бързото, което може да бъде постигнато. Представена е още примерна логическа структура на изчислителя с микроконвейерна организация, която е подходяща при серийно изпълнение на операция деление. Работоспособността на схемата за деление е илюстрирана с числени примери.

Ключови думи: операция деление, частно, остатък, алгоритъм, логическа схема.

Основни съображения

В научните публикации както и в академичните монографии (чиито огромен списък не прилагаме тук), теоретичната, а така и алгоритмичната страна на операция деление, са концентрирани главно върху изчисляването на първия резултат - частното. Но при тази операция има и втори резултат - остатък. Изчислението на частното се търси по пътя на два основни подхода: въз основа на определението за операцията (Z=X/Y) [1], [4], или на определението (Z=(1/Y).X) [3]. В повечето случаи синтезираните алгоритми се основават на положителни операнди или такива без знак [2], [3], [6], което се определя като деление по модул. Въпреки че цифровата аритметика е научна област, която се изследва от десетилетия [1], [4], голяма рядкост са публикациите, в които се намират алгоритми за определяне и на остатъка. Операция деление може да се определи още и чрез операция умножение така X=Y.Z+R, където с R е означен остатъка. Изводът от това определение е, че той е цяло число, и за да бъде вярно равенството, следва да притежава знака на делимото X. За операндите и за резултатите се има предвид един и същи n-битов формат на разрядната мрежа.

Операция деление (Z=X/Y), която е сравнително рядко срещана (около 2,5%) и поради сложния си алгоритъм, е значително по-бавна в сравнение с останалите операции върху цели числа, което се признава дори сега [10]. По тези причини за нейното изпълнение все още се проектират многотактови последователни устройства [5]. Тази тенденция обаче е на изживяване, така че напълно са оправдани усилията за нейното прекратяване [6]. Желанието за гарантирано ускорение води и нас до избор на апаратна реализация.

Трябва да се обърне още внимание и на факта, че целите числа се съхраняват в паметта на компютърните системи в допълнителен код. В този код те се явяват операнди и при изчисления резултатите автоматично се получават в допълнителен код. Това се отнася и за операция деление. Използването на прав код за изобразяване на операндите, който придържа повечето автори към алгоритмите по модул, не е актуално. Типичен е примерът с нехомогенното схемно решение, показано в [6], при което се налага редуване на суматори и събстрактори. Основен недостатък на последните са по-големите апаратни разходи в сравнение със суматорите. Този недостатък ние ще се постараем да избегнем. За отбелязване е и фактът, че машинните команди за деление на цифровите процесори изискват изчисляване и на двата резултата в допълнителен код, независимо от желанията на потребителя. Двете числа остават на негово разположение в два различни регистъра на АЛУ.

За илюстрация, в нашия проект е избран алгоритъмът за деление на числа в допълнителен код, без възстановяване на частичния остатък въз основа на схемата с неподвижен делител. Използваният от нас подход обаче успешно може да бъде приложен и върху други алгоритми, като например тези, позволяващи едновременното определяне на няколко цифри в частното [4]. Показано е, че синтезираният в крайна сметка схемен делител изчислява частното както за цели, така и за дробни числа, което е предпоставка при деление на числа, представени във форма с плаваща запетая.

Част 1. Изчисляване на частното

Логическият синтез, който е представен тук, умишлено пропуска подробното изложение на избрания алгоритъм, тъй като той е основен в цифровата аритметика. Ще припомним, че непосредственото изчисляване на цифрите на частното Z започва след лявата нормализация на делимото X и на делителя Y. Ляво нормализираните операнди тук са означени с имената Wx и Wy. На всеки такт от итерационния алгоритъм се формира една цифра (0 или 1) на частното z_m , m=1,2,3,...n, и се определя следващата операция (събиране или изваждане). Правилата за това са представени в таблица 1.1.

Таблица 1.1. *Определяне на текущата цифра на частното и на следващата операция*

Знак на Остатъка R_m	Знак на Делителя Wy	Текуща цифра на Частното z_m	Следваща операция R_{m+1}=
+	+	1	$2.R_m$ - Wy
+	-	0	$2.R_m$ + Wy
-	+	0	$2.R_m$ + Wy
-	-	1	$2.R_m$ - Wy

Както и при умножение, идеята за апаратно ускоряване при деление се състои в използването на множество суматори. Така реализацията на същинското деление ще бъде търсена във вид на комбинационна схема.

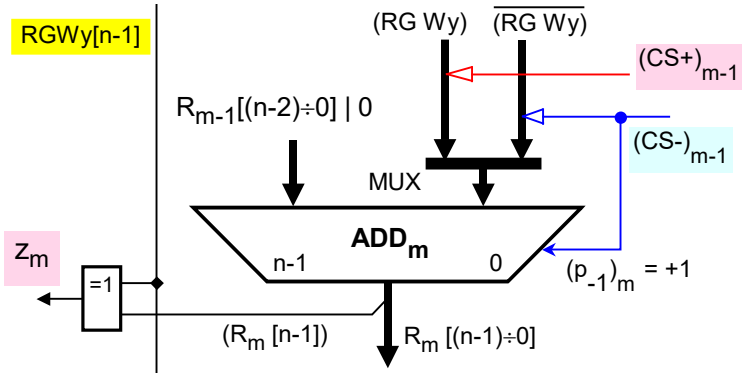
От сравнението на знака на делителя със знака на текущия частичен остатък R_m, се определя поредната цифра z_m в частното. Приемайки таблица 1.1 като таблица на истинност, можем да синтезираме логическа функция, която определя стойността на поредната цифра в частното. Нейното уравнение е следното

$$z_m = \left(R_m[n-1] \cap Wy[n-1]\right) \cup \left(\overline{R_m[n-1]} \cap \overline{Wy[n-1]}\right), \qquad m = \overline{0,n-2} \quad . \tag{1.1}$$

Това е елементарната логическа функция на равнозначността, т.е. ако знаците на текущия частичен остатък и на делителя съвпадат, в частното се записва единица (1), в противен случай нула (0). Заедно с това се извършва изместване наляво на частичния остатък ($2.R_m$). Изместването наляво възстановява реда на полинома на частичния остатък до (n-2).

Следващият частичен остатък R_{m+1} се получава след събиране или изваждане на нормализирания делител, както е посочено в дясната колонка на таблица 1.1 ($R_{m+1}=2.R_m \pm Wy$). Аритметическата операция изисква суматор. Изборът на втория операнд в тази сума извършва функция (1.1).

И така, завършваме този етап от обясненията с произтичащата от тях логическа структура.



Фиг. 1.1. Логическа структура на текущ суматор в делителя (текущо m-то ниво)

Както може да се види, знакът на делителя RGWy[n-1] и знакът на новия частичен остатък R_m[n-1] формират според (1.1) текущата цифра в частното z_m .

За да се получи текущият частичен остатък R_m, на левия вход на текущия суматор ADD_m се подава изместеният наляво на един бит предходен частичен остатък R_{m-1}, т.е. $2.R_{m-1}=(R_{m-1}[(n-2)\ldots 0] \mid 0)$.

По десния вход на суматора, чрез двувходовия мултиплексор MUX, се подава за събиране или за изваждане нормализираният делител Wy. Той се явява съдържание на регистър RGWy, където остава до края на операцията. Управлението на десния вход на суматора се реализира чрез управляващите сигнали (CS+) и (CS-). Логическите стойности на тези сигнали са кодирани според следната логическа функция:

$$if \quad z_m = \begin{cases} 0\ , & then \quad (CS+)=1\ ,\ (CS-)=0\ ; \\ 1\ , & then \quad (CS-)=1\ ,\ (CS+)=0\ . \end{cases} \qquad (1.2)$$

Частното е число със знак и се получава автоматично в допълнителен код. Неговата цифрова част има дължината (n-1)[b]. От тук следва, че схемният делител трябва да има (n-1) на брой нива, за да изчисли (n-1) на брой цифри. Знакът на частното се определя предварително. Той определя началното съдържание на регистъра на частното RGZ.

$$\begin{cases} if \quad SignZ=0\ , & then \quad RGZ := 000...000 = 0\ ; \\ if \quad SignZ=1\ , & then \quad RGZ := 111...111 = 2^{n-1}-1\ . \end{cases} \qquad (1.3)$$

Последното, което трябва да бъде изяснено, е свързано с това, че при този алгоритъм, частното не винаги се получава точно, което се дължи на несиметричността на допълнителния код. Правилата за корекция на частното са представени в таблица 1.2.

Таблица 1.2. *Определяне на корекцията*

Знак на делимото X	Знак на делителя Y	Стойност на корекцията
+	+	няма корекция
+	-	+1
-	+	+1, *if* $R_{k-l+1} \neq 0$
-	-	+1, *if* $R_{k-l+1} = 0$

В тази таблица с R_{k-l+1} е означен последният частичен остатък, където с k е означен реда на полинома на делимото X, а с l, съответно реда на полинома на делителя Y. За операндите и за резултатите се има предвид формат от n[b]. Резултат от лявата нормализация на операндите е и числото N, (N=k-l+1), показващо броя на неизвестните цифри на частното.

Както се вижда от таблица 1.2, когато частното не е вярно, то се коригира, като към младшия му бит се добавя единица. Това налага разпознаване на съответните ситуации, определени в таблицата. Разпознаването или още дешифрирането на необходимостта от корекция осъществява логическа функция, която зависи от знаците на операндите, а в последните два случая още и от това, дали делението е точно или не. Ако на таблица 1.2 се гледа като на таблица на истинност, може да се съобрази, че логическата функция, изразяваща необходимостта от корекция на частното, е дизюнкция от три логически терма

$$COR = cor1 \cup cor2 \cup cor3\ , \qquad (1.4)$$

където cor (*correction*). Първият терм изразява условието за корекция според втория ред на таблицата

$$cor1 = \overline{(X[n-1])} \cap (Y[n-1])\ . \qquad (1.5)$$

Вторият и третият терм (cor2 и cor3) са функции за корекция, отнасящи се за третия и съответно за четвъртия случай в таблицата. Тези функции зависят от знаците на операндите, както и от стойността на последния частичен остатък, т.е. от това дали делението е точно или не. Така за тях получаваме следните изрази:

$$cor2 = \left[(X[n-1]) \cap \overline{(Y[n-1])}\right] \cap \overline{EQ(R)}\ , \qquad (1.6)$$

и съответно

$$cor3 = \left[(X[n-1]) \cap (Y[n-1])\right] \cap EQ(R)\ . \qquad (1.7)$$

В горните изрази с EQ(R) е означена логическата стойност на функция, която дешифрира нулев частичен остатък в произволна итерация (произволно ниво) по време на делението. Тъй като този факт трябва да се дешифрира на всеки такт, тази функция трябва да има следния събирателен вид:

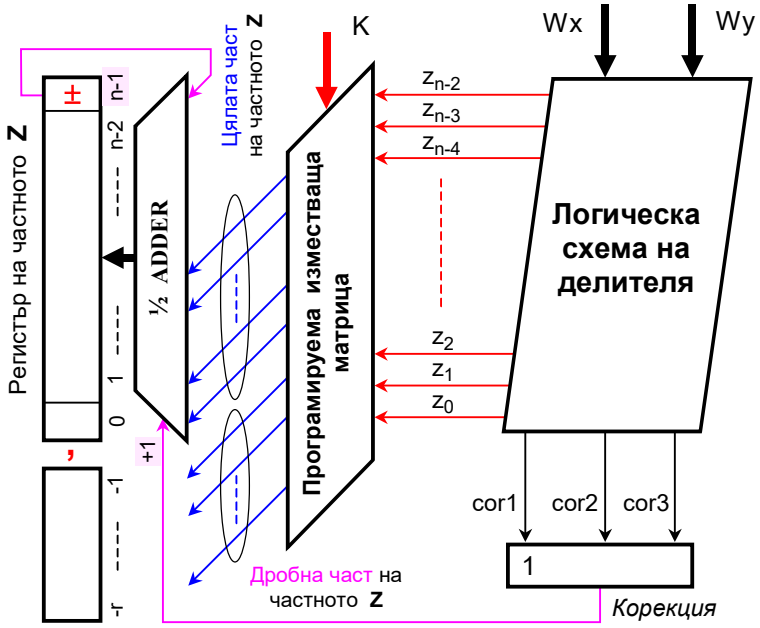
$$EQ(R) = \bigcup_{m=0}^{n-2} EQ(R_m) \ , \qquad (1.8)$$

където

$$EQ(R_m) = \bigcap_{m=0}^{n-1} \overline{z_m} \ , \qquad (1.9)$$

С други думи функция (1.8) изразява възможността за преждевременно точно деление, което може да се констатира на всеки такт.

Искаме да обърнем внимание на това, че според разбирането на традиционния алгоритъм за деление, последният частичен остатък има изключително сложно определение за това, че е *последен*. Например, той е последен, ако са получени всички неизвестни цифри на частното, т.е. ако са получени всички N на брой цифри. Но има случаи на точно *преждевременно* деление, при това след различен брой тактове, преди изчерпване на контролното число N. В смисъла на решаваната тук задача, т.е. в смисъла на апаратното реализиране на процеса на деление, *определянето на остатъка като последен е крайно неудобно*, тъй като той може да се получи в произволно ниво на логическата схема на делителя. Ето защо изказването, че *делението е точно, ако частното няма дробна част*, представлява изключително удобна интерпретация за това дали е необходима или не корекция в последните два случая на таблицата. Въз основа на изложеното по-горе може да представим синтезираната част от структурата на схемния делител.



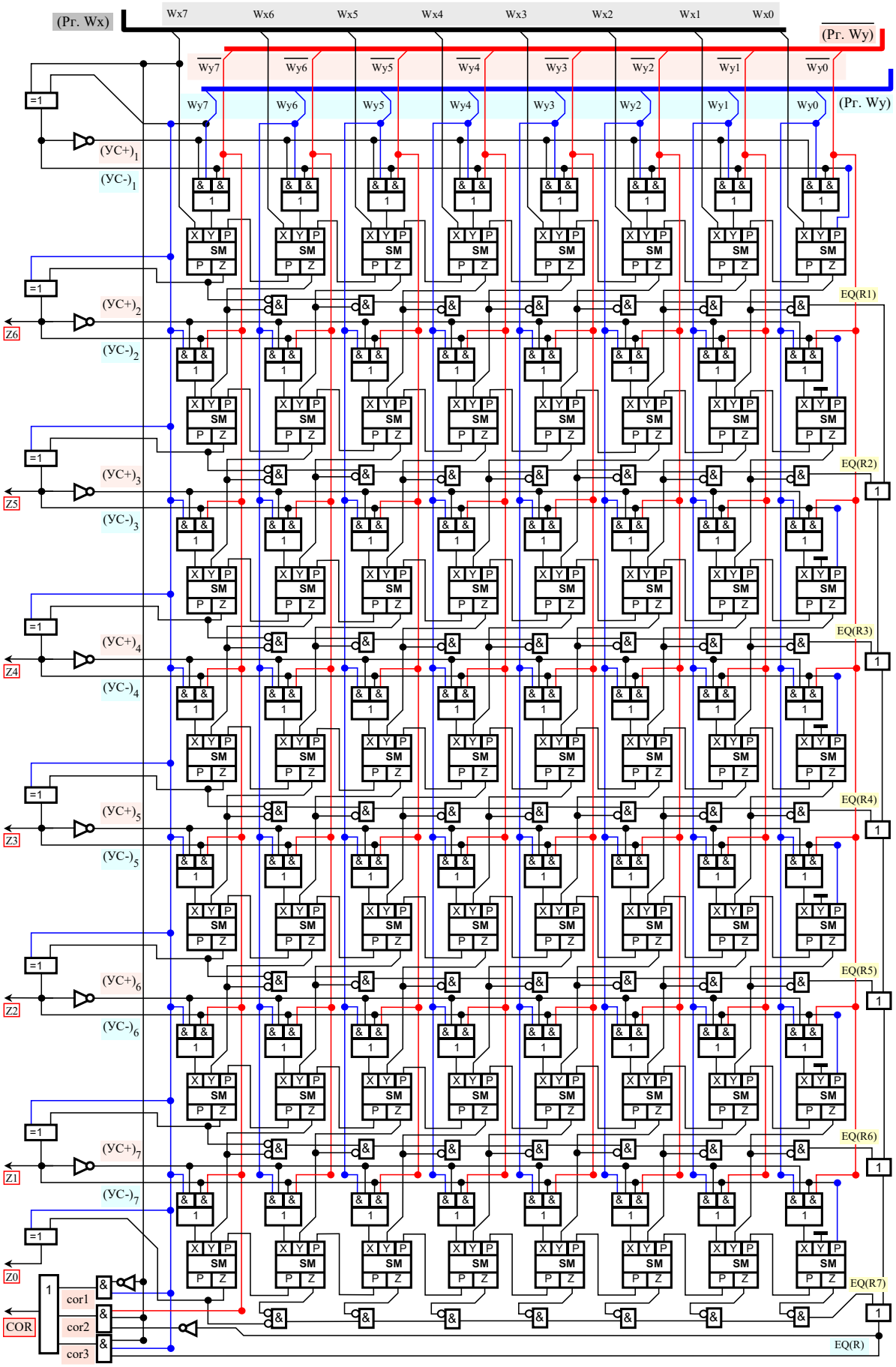
Фиг. 1.2. Логическа структура на схемния делител

На схемния делител се подават двата ляво нормализирани операнда, чиито знаци определят началното съдържание на регистъра на частното RGZ. От схемата излизат (n-1) на брой цифри на частното (z_{n-2}, z_{n-3}, … z_1, z_0), както и трите терма на функцията за корекция (cor1, cor2, cor3). Частното е подадено на дясно изместваща програмируема матрица. Изместването е в посоката на запетаята и е аритметическо, т.е. с така нареченото знаково разширение. Както се вижда от рисунката, тази схема се програмира за изместване от параметъра K. Броят на изместванията се определя така:

$$K = (n-1) - N \ . \qquad (1.10)$$

Стойността на параметъра K може да бъде изчислен още на етапа на лявата нормализация, когато се изчислява и стойността на N (брой на неизвестните цифри в частното). След това, тази стойност трябва да се съхранява в отделен регистър RGK до края на операцията.

Що се отнася до самата корекция, то горната схема илюстрира как изместеният и позициониран спрямо правилното място на запетаята резултат, се събира със стойността на функцията COR (0 или 1), подавана в младшия разряд №0 на полусуматора ½ADDER. Дробната част на частното в структурната схема е показана само за илюстрация. Тя без корекцията е невярна и реално следва да се изхвърли. Ако все пак искаме да я съхраним, то корекцията (+1) трябва да се подава не в младшия бит на цялата част (b_0), а в най-младшия фиксиран бит на реалното частно (b_{-r}).



Фиг. 1.3. Принципна логическа схема на схемен делител 8x8

По-горе на фигура 1.3 е представена синтезираната принципна логическа схема на схемен делител, който изчислява винаги 7 старши цифри от значещата част на частното (от z_6 до z_0). След изместването надясно на K бита, както е показано в примерите и в логическата структура на фигура 1.2, в регистъра на частното ще останат старшите N на брой цифри на цялото число. Както е известно, в знаковия бит се намира установеният като начална стойност знак на частното z_7, а следващите го по-младши цифри го повтарят, допълвайки отляво изместеното N битовото частно според правилото за знаковото разширение.

Както беше отбелязано по време на анализа, не е възможно апаратното решение на същинското деление да постигне онази динамика, която е присъща на алгоритъма. Има се предвид фактическата дължина на частното, която е променливата величина и се изразява чрез параметъра N. Фактическата дължина на частното се адаптира в схемата чрез параметъра K, който инициализира програмируемата изместваща матрица. Функционирането на апаратното решение обаче благоприятства използването му в случай на деление на числа с плаваща запетая. Както е известно, мантисите на операндите са винаги ляво нормализирани числа, от където следва, че частното се получава винаги с една и съща дължина. С други думи, при деление на числа с ляво фиксирана запетая, схемният делител не зависи в своя синтез от параметъра N и в неговата структура не е необходима програмируемата изместваща матрица. За да се уеднаквят двата случая, при деление на числа с ляво фиксирана запетая изместващата матрица може да се управлява допълнително за да стане прозрачна.

Част 2. Изчисляване на остатъка

Теоретично основание

Основавайки се на синтеза, изложен по-горе в първа част, тук представяме неговото продължение, отнасящо се до хардуерното изчисление на втория резултат – остатъка. В заключението е описана представата ни за пълната и цялостна структура на комбинационната схема на делителя, което прави операция деление идентична с операция умножение както по структура, така и по бързодействие. С цел оптимизация на комбинационната схема на делителя и минимизиране на нейната латентност, могат да бъдат използвани същите методи, които са възможни върху схемния умножител [4].

Вече посочихме, че операция деление Z=X/Y, като най-сложна, за разлика от всички други, генерира два самостоятелни резултата, изисквани в изчислителните алгоритми - частно Z и остатък R. Чрез тези два резултата може да бъде дадено следното определение на операция деление - частното и остатъкът са такива числа, щото Z.Y+R=X.

Ще припомним още, че $X_{(\mathrm{mod}Y)}=R$, при което Y се нарича модул за сравнение на числото X с други числа, които имат същия остатък R, както и още, че частното Z, в тази постановка, се определя като коефициент за кратност на модула Y. Точно в тази интерпретация, числото X може да се изрази чрез коефициента за кратност Z, модула за сравнение Y и остатъка, или още изображението R, така X=Z.Y+R.

Итерационният алгоритъм, по силата на който се получават цифрите на частното, се предхожда от лявата нормализация на операндите. В процеса на тази нормализация се определя цялото число N, показващо броя на неизвестните цифри на частното. Числото N се дефинира чрез равенството N=k-l+1.

В [4], при изложение на алгоритъма, е показано, че частичният остатък, който определя последната цифра в частното, може да послужи за извеждане на следното равенство

$$2^{-(k-l)}.R_{k-l+1} = X - Z.Y \quad .$$

Според даденото в началото определение, това равенство може да бъде записано така

$$2^{-(k-l)}.R_{k-l+1} = R \; . \qquad (2.1)$$

Равенството (2.1) е забележително с това, че изразява как може да се получи вторият резултат от операция деление, а именно остатъкът R от делението. Изводът е, че остатъкът R се съдържа в последния частичен остатък R_{k-l+1}, който следва да се измести (k-l) бита надясно, за да се представи правилно като цяло число. Това число е със знак и ще бъде получено автоматично в допълнителен код.

Необходимо е още едно пояснение. Ако последното изваждане, при което се получава последният частичен остатък R_{k-l+1}, е било успешно, то той съдържа търсения остатък R. Ако обаче изваждането не е било успешно, то от последния частичен остатък R_{k-l+1} следва да се възстанови предидущия частичен остатък R_{k-l}, в който трябва да се съдържа търсения остатък R.

Възстановяването на предидущия частичен остатък се постига чрез събиране или изваждане на делителя Wy, операция, която се избира според правилото, както следва

$$\begin{cases} \textit{if} \quad (R_{k-l+1}[n-1]) = Wy[n-1], & \textit{then} \quad R_{k-l} = R_{k-l+1} - Wy \ ; \\ \textit{if} \quad (R_{k-l+1}[n-1]) \neq Wy[n-1], & \textit{then} \quad R_{k-l} = R_{k-l+1} + Wy \ . \end{cases} \qquad (2.2)$$

В крайна сметка остатъкът R може да бъде изчислен с прилагане на следния алгоритъм:

Ако полученото частно Z е *нечетно* число, то остатъкът R се съдържа в последната разлика.

Ако полученото частно Z е *четно* число, то остатъкът R се съдържа в предидущата разлика. В този случай, за определяне на остатъка, се налага възстановяване на предидущата разлика.

Окончателният остатък се получава след аритметическо изместване надясно на (k-l) бита на съответния частичен остатък.

ПРИМЕРИ

Тук по-долу като илюстрация на изказания алгоритъм са представени два числени примера. И двата илюстрират неточно деление, т.е. деление с остатък. Първият пример илюстрира деление на две положителни числа, при което частното е четно число. Последният факт води алгоритъма към случая, когато остатъкът се съдържа в предидущата разлика (000100). Тя се възстановява, след което се прави (k-l)-битово изместване надясно и се формира остатъкът.

Пример 1. Да се изпълни операция деление Z=X/Y на числата X=31 и Y=5, които са представени във формат n=6[b]. Следва да получим този отговор: частно Z=6 и остатък R=1, т.е. 31=5.6 + 1.

$|X| = 0\ 11111$; $|Y| = 0\ 00101$.

Нормализиране на операндите X и Y.

0	11111	= X
0	11111	= Wx

Делимото е нормализирано

0	00101	= Y
0	10100	= Wy

2 ← Лява нормализация на Делителя (2 бита)

N = 2 – 0 + 1 = 3 (3 неизвестни цифри на частното)

k-l = 2-0 = 2 - изместване на 2[b] за формиране на остатъка.

$|Z| = 0\ 00zzz$ = ?

$0\ 00110,$ = 6.

Wy = 0 10100

Wx = 0 11111

изваждане +	1	01100
Ост. > 0	0	01011 ←
изваждане +	0	10110
	1	01100
Ост. > 0	0	00010 ←
изваждане +	0	00100
	1	01100
Ост. < 0	1	10000
Възстановяване на предидущия частичен остатък:		
събиране +	1	10000
	0	10100
	0	00100
k-l=2 →	0	00001 ,

възстановяване

R = 1 .

В този случай частното не се нуждае от корекция: Z=+6 .

Вторият пример илюстрира деление на две отрицателни числа, при което частното е нечетно число. Последният факт води алгоритъма към случая, когато остатъкът се съдържа в последната разлика (10100000). Това е случай, когато последната разлика съдържа остатъка, който се формира след нужното изместване.

Пример 2. Да се изпълни операция деление Z=X/Y на числата X=-97 и Y=-7, които са представени във формат n=8[b], в допълнителен код. Следва да получим този отговор: частно Z=+13 и остатък R=-6, т.е. (-97)=(-7).13 - 6 .

$$[X]_{ДК} = 1\ 0011111\ ; \qquad [Y]_{ДК} = 1\ 1111001\ .$$

Нормализиране на операндите X и Y.

1	0011111	= X
1	0011111	= Wx

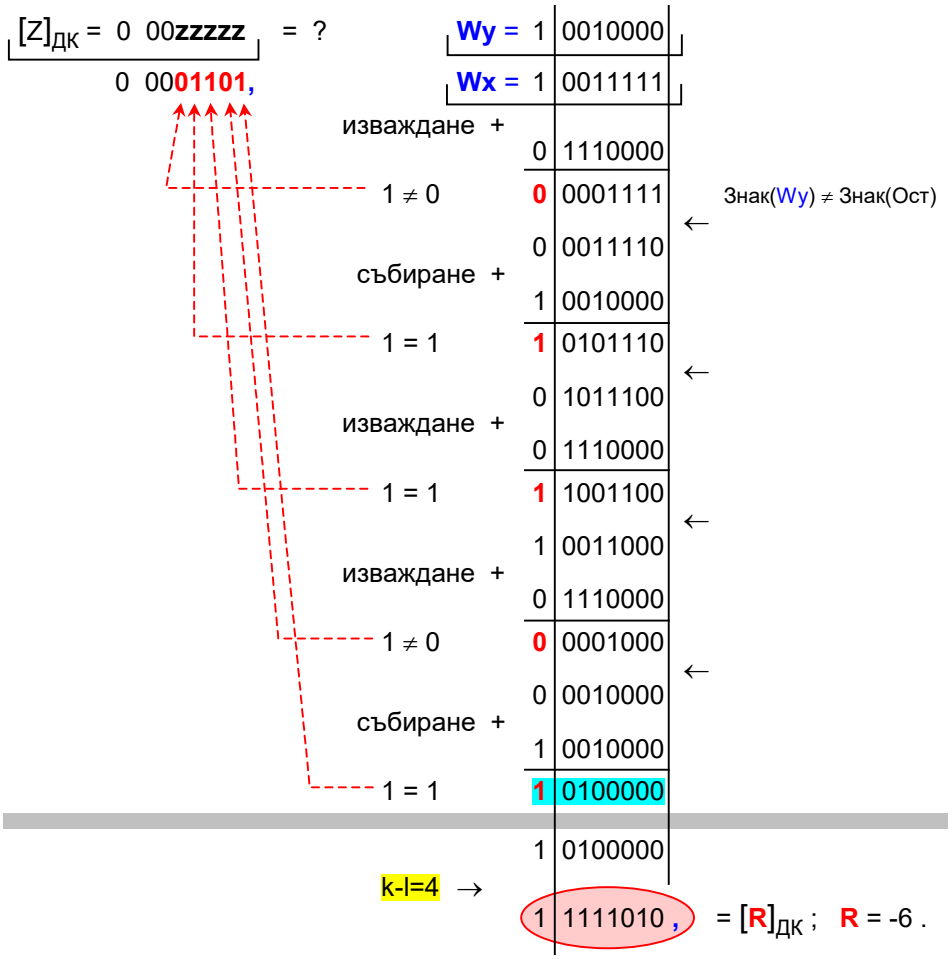
Делимото е нормализирано

1	1111001	= Y
1	0010000	= Wy

4 ← Лява нормализация на Делителя (4 бита)

N = 4 – 0 + 1 = 5 (5 неизвестни цифри на частното)

k-l = 4-0 = 4 - изместване на 4[b] за формиране на остатъка.

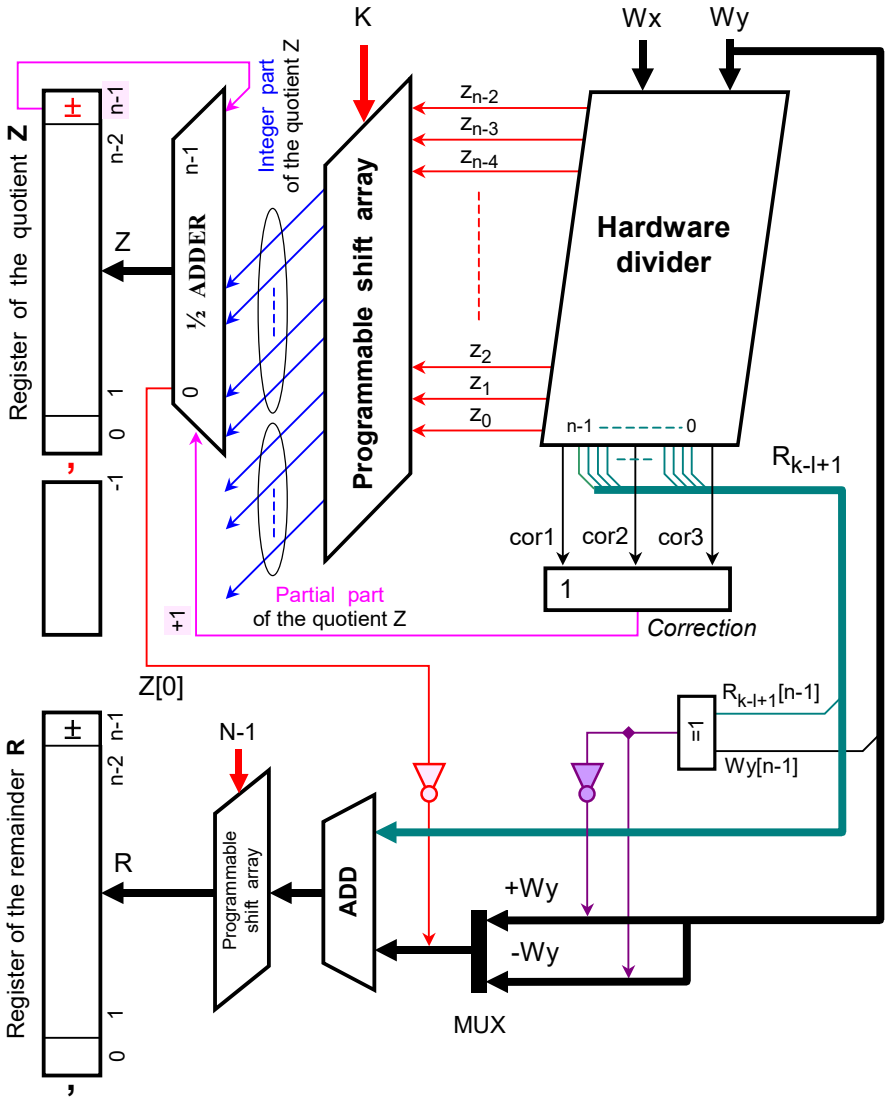


В този случай частното не се нуждае от корекция: Z=+13 .

Операция деление е най-сложната от всички операции върху цели числа. По време на нейното изпълнение могат да настъпят различни ситуации, при това в разнообразни съчетания. Такива ситуации са недефинираност (Y=0), препълване, точно деление, както и преждевременно точно деление. За да се илюстрират всички тези случаи следва да се изпълнят множество числени примери. Такива, в помощ на реалния синтез, могат да бъдат разгледани в [8].

Синтез на логическата структура

Изложените теоретични основания, алгоритми и числени примери ни позволява да синтезираме онази част от логическа структура, която допълва представената в първа част на изложението, до следния й окончателен вид.



Фиг. 2.1. Логическа структура на схемния делител,
допълнена с елементи, необходими за изчисляване и на остатъка

С казаното до тук считаме изложението на нашия проект за завършено. Разбира се има още няколко подробности, като например случая с недефинираност (Y=0), или пък случая с препълване при деление на цели числа и др.. Отразяването на тези тънкости в синтезираната схема е напълно възможно, което изисква само отлично познаване на алгоритъма.

Цялостното изпълнение на устройството за деление във вид на една единствена комбинационна схема е напълно възможно. Това превръща операция деление в еднотактна, което я прави аналогична с операция умножение, както и с някои структурни елементи в FPU.

Заключение

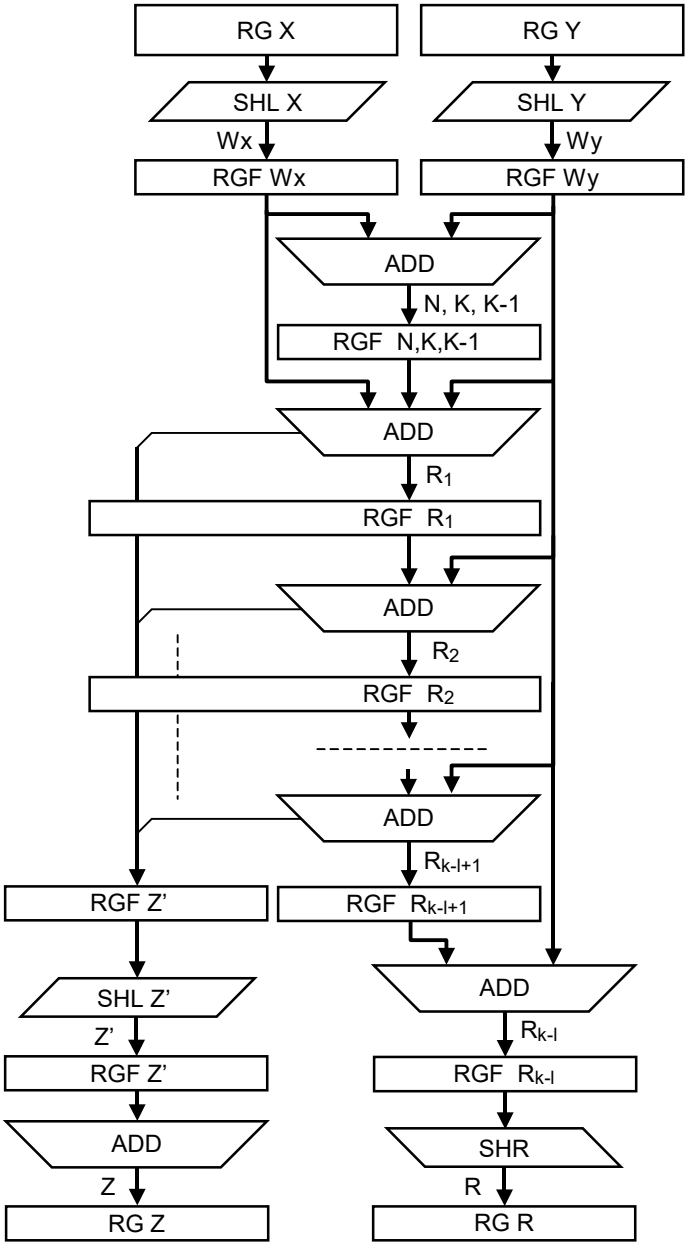
Ще опишем кратко целият състав на тази достатъчно сложна комбинационна схема. Необходими са само 4 регистъра – 2 два входни за входните операнди X,Y и два изходни за двата резултата Z,R. Между тези две двойки регистри се нареждат последователно следните съставни комбинационни схеми. Най-напред върху входните регистри се поставят схеми за определяне броя на старшите незначещи цифри на операндите. Това са комбинационни схеми, които са синтезирани по наш проект, публикуван в [7].

Формираните от тези две схеми числа постъпват на суматор, който изчислява споменатото по хода на изложението по-горе число N. Резултат на този суматор могат да бъдат още числата K и (K-1), необходими за инициализация на следващите в схемата функционални нейни елементи. Операндите постъпват още на входа на схемния делител, от изходите на който излизат две комбинации. Първата, преминавайки през изместващата матрица и полу-суматора ½ADDER (фигура 1.1), се записва в RGZ като първи резултат – частно Z. Втората, представляваща последната разлика R_{n-1}, преминава последователно през суматорът ADD и изместващата надясно матрица и се записва в изходния регистър RGR като втори резултат – остатък R.

В случай, че схемата се използва за деление на числа с плаваща запетая, остатъкът губи своя смисъл, а числото K приема стойност 0, която прави изместващата матрица прозрачна.

Представените два числени примера илюстрират функционирането на алгоритъма и на синтезираната логическа структура. Повече числени примери могат да бъдат видяни в [8].

Последователните елементи в описаната комбинационна схема могат да бъдат организирани в микроконвейер, показан на фигура 2.2. Конвейерната организация ще позволи да бъде повишена производителността на изчислителя в случаи, когато математическите изчисления съдържат множество последователни операции деление. Микроконвейерното управление на схемния делител може да бъде постигнато с методите и средствата, подробно изложени в монографията [9].



Фиг. 2.2. Примерна логическа структура за микроконвейерна организация на схемния делител

Както може да се види от структурната схема на конвейера, в него влизат регистри-фиксатори RGF, комбинационни схеми - суматори ADD, програмируеми матрици, изместващи наляво SHL и надясно SHR, както и необходимите за управление на отделните степени на конвейера конвейерни автомати, които не са изобразени на схемата.

Литература

[1]. Richard, P., Zimmermann, B., Zimmermann, P., *(2010), Modern Computer Arithmetic*, Cambridge Monographs on Computational and Applied Mathematics (No. 18), Cambridge University Press, November 2010, 236 pages. ISBN-13: 978-0521194693.

[2]. Cavagnino, D., Werbrouck, A., *(2008), Efficient Algorithms for Integer Division by Constants Using Multiplication*, The Computer Journal, Volume 51, Issue 4, 1 July 2008, Pages 470–480.

[3]. Kumar, D., Saha, P., Dandapat, A., *(2017), Hardware Implementation of Methodologies of Fixed Point Division Algorithms*, International Journal of Smart Sensing and Inteligent Systems, Vol. 10, No.3, September 2017.

[4]. Tyanev, D. S., *(2008), Computer Organization. Tom1, Tom 2,* Technical University of Varna, ISBN: 978-954-20-0412-7, ISBN 978-954-20-0413-4.
Достъпно от: http://www.tyanev.com/home.php?lang=bg&mid=18&mod=1&b=12 .

[5]. Trummer, R., Zinterhof, P., Trobec, R., *(2005), A High-Performance Data-Dependent Hardware Divider*, Parallel Numerics'05, 193-206 M. Vajteršic, R. Trobec, P. Zinterhof, A. Uhl (Eds.) Chapter 7: Systems and Simulation, ISBN: 961-6303-67-8.

[6]. Takagi N., Kadowaki, S., Takagi, K., *(2005), A hardware algorithm for integer division. Computer Arithmetic*. ARITH-17 2005, 17th IEEE Symposium, ISSN: 1063-6889.

[7]. Tyanev, D. S., Petkova, Y. P., *(2015), Logic scheme for determining the number of leftmost insignificant digits in a bit-set of any length*. SciTechnol: Journal of Computer Engineering & Information Technology, USA, ISSN: 2324-9307, 2015, Vol. 4, Issue 1. doi: 10.4172/2324-9307.1000123. Достъпно от: https://www.scitechnol.com/logic-scheme-for-determining-the-number-of-leftmost-insignificant-digits-in-a-bitset-of-any-length-hRjK.php?article_id=3259

[8]. Tyanev, D. S., *(2007), Computer organization. Digital arithmetics – exercises*, Technical University of Varna, ISBN: 954-20-0258-0.
Достъпно от: http://www.tyanev.com/home.php?lang=bg&mid=18&mod=1&b=7 .

[9]. Tyanev, D. S., *(2016), Asynchronous pipeline systems with common structures (Synthesis Methodology),* Technical University of Varna.
Достъпно от: http://www.tyanev.com/home.php?lang=bg&mid=18&mod=1&b=14

[10]. Lemire, D., Kaser, O., Kurz, N., *(2019), Faster Remainder by Direct Computation Applications to Compilers and Software Libraries.* Достъпно от: https://arxiv.org/abs/1902.01961.pdf .