liter i znaków (jest tzw. *kodem alfanumerycznym*). Większość kombinacji kodowych może mieć dwa różne znaczenia; o wyborze właściwego decyduje to, który z symboli „Litery", „Cyfry" występował ostatni w ciągu sygnałów. Zastępując dla uproszczenia wyrażenia kodowe ich numerami z przecinkiem, można np. nazwę

F I A T 1 2 5

zakodować

29, 6, 9, 1, 20, 31, 30, 17, 23, 20

1.4. DZIAŁANIA ARYTMETYCZNE

1.4.1. DODAWANIE I ODEJMOWANIE DWÓJKOWE

W układach cyfrowych automatyki występuje często konieczność wykonywania operacji arytmetycznych na sygnałach, przedstawionych w postaci liczb dwójkowych lub dwójkowo-dziesiętnych. Zazwyczaj nie są to operacje bardzo złożone, a zakresy zmian zarówno argumentów jak i wyników działań można z góry wyznaczyć. Z tego też powodu, w przypadku występowania liczb mieszanych (całkowitych i ułamkowych) położenie przecinka jest znane i niezmienne, jeśli nie w całym procesie przetwarzania, to przynajmniej w poszczególnych fazach tego procesu. Stały przecinek nie ma wpływu na sposób zapisywania liczb i wykonywania działań, a zatem może być całkowicie pomijany w układach przetwarzających i dopiero w końcowej fazie — drukowania lub przekazywania wyników — może znów być odtwarzany.

Reguły dodawania i odejmowania cyfr dwójkowych są bardzo proste:

dodajna	0	0	1	1	odjemna	0	0	1	1
dodajnik	0	1	0	1	odjemnik	0	1	0	1
suma	0	1	1	0	różnica	0	1	1	0
przeniesienie	0	0	0	1	pożyczka	0	1	0	0

W przypadku liczb wielocyfrowych przeniesienie jest dodawane do cyfr bardziej znaczących, a pożyczka jest odejmowana od cyfr bardziej znaczących. Pewne komplikacje powstają wówczas, gdy nie wszystkie argumenty działań są dodatnie lub gdy odjemna jest mniejsza od odjemnika. Proste rozwiązanie tych problemów uzyskuje się przez pewną

modyfikację zapisu liczb ujemnych, w związku z czym wyróżnia się trzy podstawowe sposoby zapisu liczb dwójkowych dodatnich i ujemnych:

— zapis znak-moduł (zapis modułowy),
— zapis znak-uzupełnienie do jedności (zapis odwrotny),
— zapis znak-uzupełnienie do dwóch (zapis dopełniający).

Zapis znak-moduł zawiera n bitów naturalnego kodu dwójkowego wyrażających bezwzględną wartość liczby oraz znak + albo −, niekiedy w formie bitu znaku: 0. (plus) albo 1. (minus), np.:

$$(+3)\quad 0.011\qquad (-3)\quad 1.011$$

Zapis znak-uzupełnienie do jedności dla liczb dodatnich jest taki jak zapis modułowy, natomiast dla liczb ujemnych zawiera negacje cyfr zapisu modułowego i bit znaku, np.:

$$(+3)\quad 0.011\qquad (-3)\quad 1.100$$

Zapis znak-uzupełnienie do dwóch dla liczb dodatnich jest taki jak zapis modułowy, a dla liczb ujemnych o n bitach przedstawia dopełnienie modułów tych liczb do liczby 2^n, np.:

$$(+3)\quad 0.011\qquad (-3)\quad 1.101\qquad (2^3-3=5)$$

Bit znaku 1. można tu więc traktować jako zapis liczby -2^n obok dodatniego modułu, co w sumie daje właściwą liczbę, np.:

$$(-3)_{10} = (1.101)_d = (-1000+101)_2 = (-011)_2$$

Zapis znak-uzupełnienie do dwóch dla liczby ujemnej można też łatwo wyznaczyć, dodając do zapisu znak-uzupełnienie do jedności, na najmniej znaczącej pozycji, liczbę 1.

Dodawanie i odejmowanie liczb w zapisie modułowym wykonuje się zwykle na samych modułach, realizując odpowiednie działania w zależności od znaków składników (np. odejmowanie liczb o różnych znakach sprowadza się do dodawania itp.). Przykłady wyjaśniają zasady postępowania:

(−3)	011	(+4)	100	(+3)	011	
+(−4)	+100	+(−3)	−011	−(+4)	−100	
(−7)	111	(+1)	001	(−1)	1.111	(dop.)

Jak widać, działania sprowadzające się do dodawania, albo odejmowania liczby mniejszej od większej, dają wynik (moduł) w zapisie modułowym, natomiast przy odejmowaniu liczby większej od mniejszej otrzymuje się przekroczenie zakresu działań o bit pożyczki, a rezultat jest przedstawiony w zapisie dopełniającym. Oddzielnym zagadnieniem jest ustalenie znaku wyniku działań; można zauważyć, że w przypadku przekroczenia zakresu o bit pożyczki wynik jest ujemny, a w pozostałych przypadkach znak wyniku jest taki jak znak pierwszego argumentu.

Przy dodawaniu i odejmowaniu liczb w zapisie odwrotnym działania wykonuje się na wszystkich bitach, łącznie z bitem znaku, a wynik uzyskuje się zawsze w zapisie odwrotnym. Jeśli — po wykonaniu działań — przed bitem znaku pojawi się jedynka (tzw. *pożyczka cykliczna* lub *przeniesienie cykliczne*), należy ją przesunąć na pozycję najmniej znaczącą i powtórzyć działanie. Na przykład:

(+3)	0.011	(−3)	1.100	(+3)	0.011
+(−4)	+1.011	+(+4)	+0.100	−(+4)	−0.100
(−1)	1.110		(1) 0.000		(1) 1.111
			+1		−1
		(+1)	0.001	(−1)	1.110

Łatwość operowania znakami umożliwia tu całkowite wyrugowanie odejmowania, gdyż każde odejmowanie można sprowadzić do dodawania, ze zmiennym znakiem drugiego argumentu, np.

$$(+3)-(+4) = (+3)+(-4)$$

Przy dodawaniu i odejmowaniu liczb w zapisie dopełniającym również wykonuje się działania na wszystkich bitach, a wynik jest przedstawiony w zapisie dopełniającym. Jeśli po wykonaniu działań przed bitem znaku pojawi się jedynka przeniesienia albo pożyczki, należy ją pominąć; np.

(+3)	0.011	(−3)	1.101	(+3)	0.011
+(−4)	+1.100	+(+4)	+0.100	−(+4)	−0.100
(−1)	1.111	(+1)	(1) 0.001	(−1)	(1) 1.111

Tu również odejmowanie można łatwo zastąpić dodawaniem.

Podane wyżej przykłady działań umożliwiają porównanie wad i zalet poszczególnych zapisów:

— w zapisie modułowym liczby występują w najprostszej postaci, ale niezbędna jest realizacja układowa zarówno dodawania jak i odejmowania, określenie znaku wyniku bywa kłopotliwe, a wynik nie zawsze występuje w zapisie modułowym;

— w zapisie odwrotnym utworzenie liczby ujemnej jest stosunkowo proste, ale pewne kłopoty powoduje przeniesienie (pożyczka) cykliczne oraz podwójna reprezentacja zera (0.000 i 1.111);

— w zapisie dopełniającym wszystkie działania są proste, ale samo utworzenie zapisu liczby ujemnej wymaga pewnych zabiegów. Tak więc każdy z zapisów ma zalety i wady; przy wyborze trzeba uwzględniać czynniki dodatkowe.

1.4.2. DODAWANIE I ODEJMOWANIE DWÓJKOWO-DZIESIĘTNE

Liczby dwójkowo-dziesiętne ze znakiem, podobnie jak dwójkowe, są przedstawiane w trzech podstawowych zapisach:

— *znak-moduł*,

— *znak-uzupełnienie do dziewięciu*,

— *znak-uzupełnienie do dziesięciu*.

We wszystkich tych zapisach liczby dodatnie są przedstawiane w taki sam sposób, ujemne zaś tworzy się na podstawie modułu.

Aby uzyskać uzupełnienie do dziewięciu, należy każdą cyfrę modułu odjąć od 9. Uzupełnienie do dziesięciu jest większe o 1 na najmniej znaczącej pozycji, a więc dla liczb n-cyfrowych oznacza różnicę między 10^n a modułem. W zapisie moduł-uzupełnienie do dziesięciu bit znaku ma zatem symboliczną wartość -10^n. Oznaczając — dla uproszczenia — wartość tetrady odpowiednim znakiem dziesiętnym, a bit znaku zerem lub jedynką, można np. liczbę — 18 przedstawić w postaciach

$$-18 \text{ lub } 1.18 \quad 1.81 \quad 1.82$$

Zasady dodawania i odejmowania liczb wielocyfrowych są w tych zapisach takie same jak w zapisach liczb dwójkowych, przy czym uzupełnieniu do jedności odpowiada uzupełnienie do dziewięciu, a uzupełnieniu do dwóch — uzupełnienie do dziesięciu. Oto przykłady działań (dziesiętny zapis tetrad przyjęto dla uproszczenia)

Znak-moduł	(+18)	18	(+18)	18	
	−(−24)	+24	+(−24)	−24	
	(+42)	42	(−6)	1.94	(uzup. do 10)

znak-uzupełnienie do dziewięciu

(+18)	0.18	(−18)	1.81	(+18)	0.18
+(−24)	+1.75	+(+24)	+0.24	−(+24)	−0.24
(−6)	1.93		(1) 0.05		(1) 1.94
			+1		−1
		(+6)	0.06	(−6)	1.93

znak-uzupełnienie do dziesięciu

(+18)	0.18	(−18)	1.82	(+18)	0.18
+(−24)	+1.76	+(+24)	+0.24	−(+24)	−0.24
(−6)	1.94	(+6)	(1) 0.06	(−6)	(1) 1.94

W układach cyfrowych zamiast cyfr dziesiętnych występują w tych działaniach odpowiednie tetrady kodów dwójkowo-dziesiętnych, w związku z czym pozostają do wyjaśnienia dwa problemy:

— jak najprościej utworzyć uzupełnienie do 9 i do 10 w ramach tetrady;

— jak przeprowadza się dodawanie i odejmowanie w ramach tetrady.

Pierwszy problem rozwiązuje się łatwo przez zastosowanie jednego z kodów samouzupełniających (np. IV lub V), w których uzupełnienie do dziewięciu uzyskuje się przez zanegowanie wszystkich bitów. Uzupełnienie do dziesięciu wymaga dodania jedynki. W innych kodach uzupełnienia można realizować w specjalnych układach logicznych.

Nieco trudniejsze jest rozwiązanie drugiego problemu, gdyż kody dwójkowo-dziesiętne, stanowiące odcinki naturalnego kodu dwójkowego, dodawane wg zasad przyjętych dla kodu naturalnego, nie zawsze dają poprawne wyniki. Na przykład działanie 3+9 na liczbach czterobitowych daje 12 w postaci czterobitowej, a powinno dać 2 i przeniesienie do starszej dekady. Zasady uzyskania właściwego wyniku wprowadzania korekcji są różne dla różnych kodów.

Jeśli liczby są przedstawione w *kodzie „8421"* (I) to dodawanie w zakresie, gdzie suma nie przekracza liczby 9, daje wynik poprawny. Korekcja jest niezbędna, gdy suma jest większa od 9 i polega na dodaniu liczby 6, co przesuwa wynik na skali liczb, dając właściwe przeniesienie i sumę w kodzie „8421", np.:

3	0011	3	0011	8	1000
+5	+0101	+9	+1001	+9	+1001
8	1000		1100		1 0001
			+0110 kor.		+0 0110 kor.
		12	1 0010	17	1 0111

Jeśli cyfry dziesiętne są przedstawione w *kodzie Aikena* (IV), można w nich wyróżnić dwa zakresy: od 0 do 4 — gdzie mają postać jak kod „8421", i od 5 do 9 — gdzie w stosunku do kodu „8421" są przesunięte o +6. Dodanie liczby z pierwszego zakresu do liczby z drugiego zakresu daje wynik przesunięty w stosunku do kodu „8421" o +6, czyli poprawny — w kodzie Aikena. Dodanie dwóch liczb z zakresu pierwszego daje wynik poprawny, jeśli suma mieści się w pierwszym zakresie; w innych przypadkach trzeba wynik skorygować o +6. Dodanie dwóch liczb z drugiego zakresu daje wynik poprawny, jeśli suma (bez przeniesienia) mieści się w drugim zakresie; w innych przypadkach wynik trzeba skorygować o −6. Zasady te można przedstawić za pomocą następujących warunków (S* oznacza sumę przed korekcją, S — po skorygowaniu):

jeśli $00101 \leqslant S^* \leqslant 01000$ to $S = S^* + 0110$,
jeśli $10110 \leqslant S^* \leqslant 11010$ to $S = S^* - 0110$,
w pozostałych przypadkach $S = S^*$.

Jeszcze inaczej można te warunki przedstawić tak:
— korekcja występuje, gdy $0101 \leqslant S^* \leqslant 1010$,
— przy przeniesieniu $0 : S = S^* + 0110$,
— przy przeniesieniu $1 : S = S^* - 0110$.

Na przykład:

3	0011	3	0011	6	1100
+2	+0010	+4	+0100	+7	+1101
5	0101		0111		1 1001
			+0110 kor.		−0 0110 kor.
		7	1101	13	1 0011

Jeśli cyfry dziesiętne są przedstawione w *kodzie „+3"* (V) toprzy dodawaniu dwóch takich cyfr otrzymuje się wynik przesunięty o +6 względem kodu „8421", co zapewnia prawidłowe pojawianie się przeniesienia. Aby jednak wynik zachował postać kodu „+3" należy skorygo-

wać sumę o −3, jeśli przeniesienie nie występuje, albo o +3, jeśli występuje; np.

3	0110	6	1001
+4	+0111	+7	+1010
	1101		(1) 0011
	−0011 kor.		+ 0011 kor.
7	1010	13	(1) 0110

W ostatnim działaniu bit przeniesienia ma postać symboliczną: w kodzie „+3" jedynka jest przedstawiana inaczej.

Podobne zasady korekcji można wyprowadzić również dla innych kodów oraz dla odejmowania, ale kody opisane wyżej są w działaniach arytmetycznych stosowane najczęściej, a odejmowanie bywa zwykle sprowadzane do dodawania, dla uproszczenia warunków korekcji. Oto przykład działania wielocyfrowego z zastosowaniem kodu „+3" i zapisu moduł-uzupełnienie do dziesięciu:

(+53)	0. 1000	0110	
−(+22)	+ 1. 1010	1011	
	1 0. 0010	(1) 0001	
	+0011	+0011	korekcja
	0. 0101	0100	
	+0100		przeniesienie
	0. 1001	0100	
	−0011		korekcja
+31	0. 0110	0100	

Przy wykonywaniu dodawania należy pamiętać, że wynik dodania dwóch liczb n-bitowych może mieć $n+1$ bitów.

1.4.3. MNOŻENIE I DZIELENIE

Przy wykonywaniu mnożenia i dzielenia liczb znak wyniku można jednoznacznie i łatwo wyznaczyć znając znaki argumentów, przy czym znaki te nie wpływają w żaden sposób na zasady wykonywania samych działań. W takiej sytuacji najdogodniejsze jest stosowanie zapisu znak-moduł i oddzielne wykonywanie potrzebnych działań na bitach znaku i bitach liczbowych. Iloczyn lub iloraz dwóch zmiennych jest ujemny tylko

wówczas, gdy jeden z argumentów jest ujemny, co — przy poprzednio przyjętych oznaczeniach — umożliwia stwierdzenie, że znak wyniku jest 1 jeśli znak jednego argumentu jest 1 a drugiego 0. W taki sam sposób w p. 1.2 była definiowana funkcja logiczna nierównoważności (suma modulo 2), a więc bit znaku iloczynu lub ilorazu można wyznaczyć jako sumę modulo 2 bitów znaku argumentów.

Mnożenie modułów, przedstawionych w naturalnym kodzie dwójkowym, wykonuje się tak jak zwykłe mnożenie liczb wielocyfrowych, np.:

13	1101
× 5	× 101
	1101
	0000
	+ 1101
65	1000001

Jak widać, mnożenie sprowadza się do wielokrotnego dodawania odpowiednio przesuniętej mnożnej. Jest to metoda najprostsza, ale przy długich liczbach wydłuża czas wykonywania działania. Jeśli czas ten jest zbyt długi, stosuje się specjalne metody przyspieszonego mnożenia. Istnieją również metody mnożenia liczb dwójkowych i dwójkowo-dziesiętnych w zapisach innych niż modułowy, ale — ze względu na większą złożoność — w specjalizowanych urządzeniach są stosowane rzadko.

Iloczyn jest zazwyczaj liczbą znacznie dłuższą niż każdy z argumentów i dlatego w układach powinno być przewidziane miejsce na przesuwający się w lewo rezultat działań. Brak tego miejsca może spowodować zagubienie bitów najbardziej znaczących. W większych urządzeniach cyfrowych, w których określenie długości wyniku działań jest trudne (lub celowo rezygnuje się ze zbyt wielkiej dokładności), przyjęto zasadę przedstawiania wszystkich liczb w postaci ułamka. Czyni się to przez umieszczenie przecinka zaraz za bitem znaku, przez co np. trzybitową liczbę +5 zastępuje się ułamkiem 0.101, czyli +5/8. Przy dodawaniu i odejmowaniu mianownik nie ulega zmianie, więc odczytanie właściwego wyniku nie sprawia trudności; przy mnożeniu mianownik zwiększa się, co przesuwa wynik w prawo. Jeśli przesunięcie przekracza wartość przewidzianą przez konstruktora, następuje zagubienie ostatnich bitów, ale są to pozycje najmniej znaczące, nie mające istotnego wpływu na

wynik. Umieszczenie przecinka tuż za bitem znaku ma jeszcze i tę zaletę, że podczas mnożenia zachowuje bit znaku na stałej pozycji, co ułatwia stosowanie zapisów ze znakiem.

Dzielenie modułów przedstawionych w naturalnym kodzie dwójkowym wykonuje się również wg tych samych zasad co zwykłe dzielenie wielocyfrowych liczb dziesiętnych. Dla urządzeń jest to zadanie znacznie trudniejsze niż mnożenie, gdyż zachodzi konieczność porównywania dzielnika z pośrednimi wynikami odejmowania.

LITERATURA

1. *Chu Y.*: Maszyny cyfrowe. Zasady projektowania. Warszawa 1967, PWN.
2. *Flores I.*: Arytmetyka maszyn cyfrowych. Warszawa 1970, WNT.
3. *Карцев М. А.*: Арифметика цифровых машин. Москва 1969, Наука.
4. *Mostowski A. W., Pawlak Z.*: Logika dla inżynierów. Warszawa 1970, PWN.