Mikroprocesszorok gyorsítása

Tartalomjegyzék

Témakör	Főbb tartalmak	
Szorzás és osztás	Az összeadásra visszavezetett szorzás. 2 hatványaival történő szorzás. Dedikált szorzó áramkör. Kivonásra visszavezetett maradékos osztás. 2 hatványaival történő osztás.	

A szorzás problémája

Noha a szorzás az általában emlegetett négy alapművelet része mégis sokáig jelentős problémákat okozott a gyors számításokat igénylő alkalmazásokban. Az ok igen egyszerű. Míg magát a szorzást épp oly könnyedén fel tudjuk írni mint mondjuk egy összeadást (C=A+B, illetve C=A*B), a szorzás műveletének elvégzése az egyszerű ALU-val rendelkező processzorok számára időigényes, azaz lassú!

A régi mikroprocesszorok nem rendelkeztek az ALU-ban kialakított szorzó áramkörrel. Ennek alapvető oka a szűkös tranzisztorszám ami nem tette lehetővé az amúgy más úton is megoldható funkciók megvalósítását. A szorzást ugyanis ismételt összeadásokkal is meg lehet valósítani.

$$A*B = \underbrace{B+B+\ldots+B}_{A}$$

Látható, hogy az egyszerű írásmód ellenére a végrehajtandó műveletek szám A-szorosa egy összeadásnak.

Természetesen a szorzás kommutativitása okán a műveletvégrehajtás optimalizálható a szorzó és a szorzandó felcserélésével, így egy nagyobb szám kevesebbszer történő összeadása gyorsabb, mint egy kisebb szám többszöri összeadása.

A szorzás problémája

Speciális esetekben a szorzás jelentős mértékben gyorsítható. Amennyiben a 2 hatványival (2, 4, 8, 16, stb.) szorzunk, akkor azt egyszerűen a szorzandó értékét tartalmazó regiszter bitenkénti eltolásával meg lehet oldani.

Például a 8*3 műveletet az alábbi módon végezhetjük:

A szorzandó (3) egy 8 bites regiszterben ábrázolva →

A szorzó értéke 8, azaz 2^3 ezért a szorzandó regiszterének a bitjeit 3 bittel balra mozgatjuk.
A táblázatban háromszor mozgatjuk balra egy bittel!

Minden balra mozgatáskor az 1-es helyiértéken keletkező „üres” bitet 0-val töltjük fel. Az ábrán piros 0-val jelöltük.

helyiértékek									decimális
128	64	32	16	8	4	2	1		
0	0	0	0	0	0	1	1		3
eltolás balra egy bittel									
0	0	0	0	0	1	1	0		6
eltolás balra egy bittel									
0	0	0	0	1	1	0	0		12
eltolás balra egy bittel									
0	0	0	1	1	0	0	0		24

A kapott eredmény 24, azaz 8*3.
Az eltolások során a bal oldalon „kieső” bit általában a Carry bitet változtatja. Így ha nulla a kieső bit, akkor nincs túlcsordulás, ha egy, akkor túlcsordulás keletkezik amit a számításoknál figyelembe kell venni.

A szorzás kombinációs hálózattal

Léteznek a szorzás műveletét elvégző hardveres megoldások melyek bizonyos esetekben kellően gyorsak is lehetnek, ugyanakkor igen sok elemi kapuáramkörből épülnek fel, azaz sok helyet foglalnak az integrált áramkörön.

A mikroptocesszorok gyártásának kezdetén éppen a helyszűke okán ezek a szorzó áramkörök nem kerültek megvalósításra magában a processzorban. Ennek hiánya funkcionális hátrányt nem okoz, mivel az előzőekben látható módon egy egyszerű ALU segítségével is kiszámítható a szorzat. A számítási sebesség azonban a dedikált szorzó áramkörök nélkül sok szorzási műveletet tartalmazó programok esetében alacsony.

A szorzó áramkör esetében eljárhatunk úgy, ahogy papíron, kézzel végezzük a szorzást. Mivel itt kettes számrendszerben áll rendelkezésre a szorzó és a szorzandó, a tanult szorzást is kettes számrendszerben kell végezni.

		A_1	A_0	*	B_1	B_0
		A_1*B_0	A_0*B_0			
	A_1*B_1	A_0*B_1				
Carry	A_1*B_1	A_1*B_0 + A_0*B_1	A_0*B_0			
P_3	**P_2**	**P_1**	**P_0**			

Általános esetben két darab kettő bites számot (A_0, A_1 és B_0, B_1) a fenti ábra szerint szorzunk össze.

1. Elvégezzük a bitenkénti szorzásokat.
2. Összeadjuk az adott helyiértéken a részszorzatokat.

A szorzás kombinációs hálózattal

		1	1	*	1	0
		0*0	1*0			
	1*1	1*1				
Carry	1*1	0*0 + 1*1	1*0			
0	**1**	**1**	**0**			

Konkrét esetben elvégezzük a 3*2 műveletet. Eredményül 6-ot kapunk.

		1	1	*	1	1
		1*1	1*1			
	1*1	1*1				
Carry	1*1	1*1 + 1*1	1*1			
1	**0**	**0**	**1**			

Konkrét esetben elvégezzük a 3*3 műveletet. Eredményül 9-et kapunk.

Figyeljük meg, hogy ebben az esetben többször is előfordul átvitel!

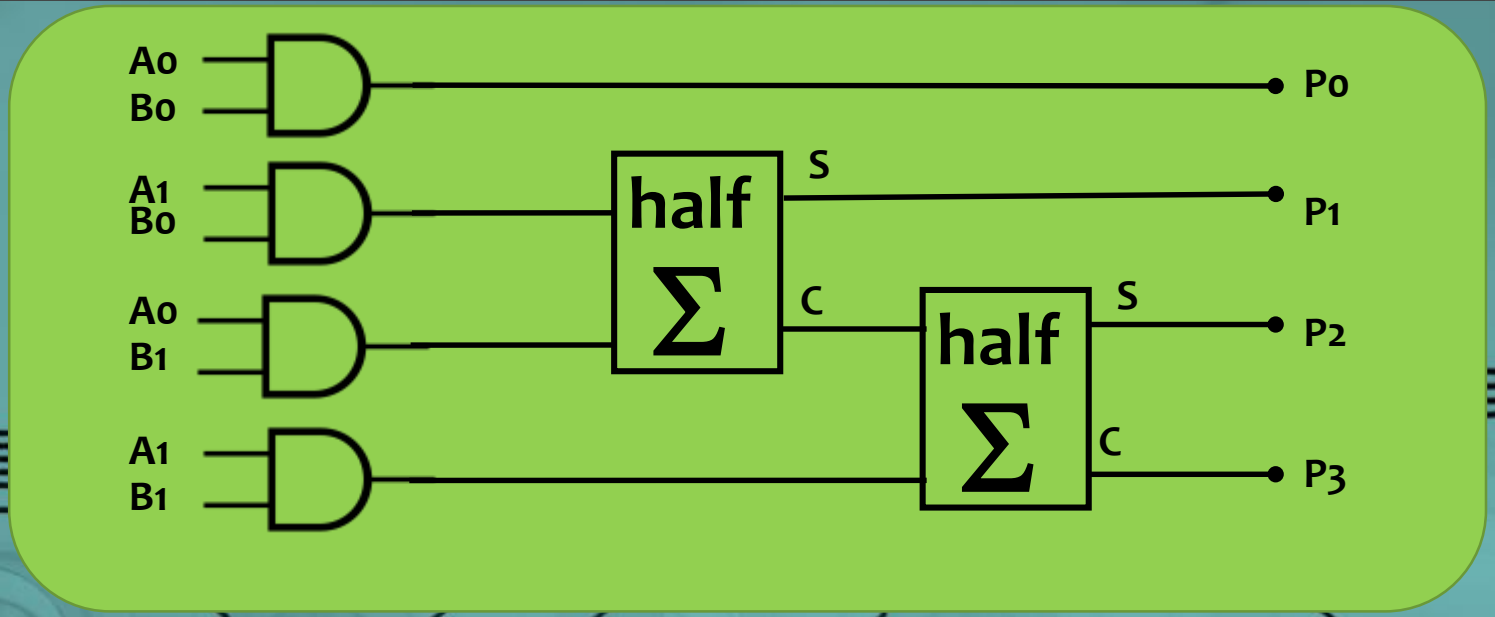
A szorzás kombinációs hálózattal

Az előzőek alapján a szorzást két lépésben tudjuk végrehajtani:

1. Bitenkénti szorzás.
   A szorzás megfelel a megfelelő bitpárok logikai ÉS (AND) műveletének.

2. A keletkezett szorzatok helyiértékeinek megfelelő összeadása.
   Az összeadásokat a már megismert összeadó áramkörökkel lehet elvégezni.

Lenti ábrán egy kétbites szorzó áramkör látható. A szorzat eredménye 4 bit. Általában n*m bites szorzat n+m bites.

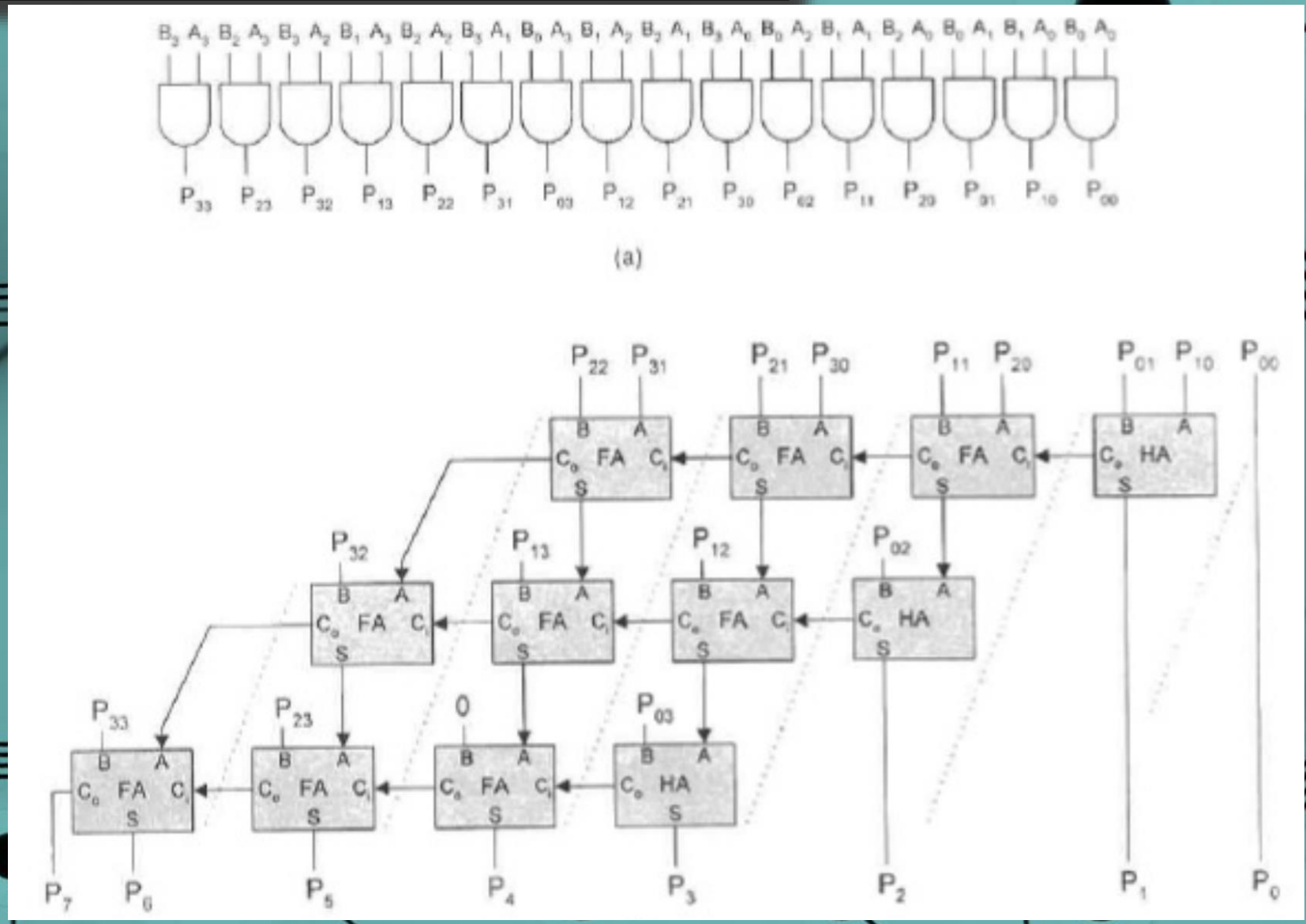


		A1	A0	*	B1	B0
		A1*B0	A0*B0			
	A1*B1	A0*B1				
Carry	A1*B1	A1*B0 + A0*B1	A0*B0			
P3	**P2**	**P1**	**P0**			

A szorzás kombinációs hálózattal

Az előző dián megismertetett kétbites szorzó egy speciális eset ezért nem tartalmaz teljes összeadót. A jobb érthetőség kedvéért érdemes megnézni egy nagyobb bitszélességű szorzó áramkört ahol már a teljes struktúra egyben megfigyelhető.

Egy négy bites szorzó áramkör ÉS (AND) kapukból félösszeadókból (HA) és teljes összeadókból (FA) felépítve. Az ábrából jól érzékelhető a szorzó áramkör alkatrészigénye ami miatt sok egyszerűbb mikroprocesszorba ezek az áramkörök nem kerültek beépítésre.



Az osztás problémája

Az osztás a szorzáshoz hasonlóan időigényes művelet. Az osztást kivonásokra vezetjük vissza, azaz az osztandóból annyiszor vonjuk ki az osztót ahányszor az lehetséges. Megszámolva a kivonásokat megkapjuk az osztás egész részét és megvizsgálva az osztandót amely most már kisebb értékű mint az osztó vagy éppen nulla, megkapjuk a maradékot. Tehát az iskolában tanult klasszikus maradékos osztást végezzük el.

Az osztásnál azonban nem lehet felcserélni az osztót az osztandóval, azaz a művelet nem kommutatív. Figyelni kell továbbá arra is, hogy az osztó nem lehet nulla!

A kettő hatványaival (2, 4, 8, 16, stb.) történő osztás a szorzásnál megismert módon bit eltolással könnyen megvalósítható.

Mindezek ellenére az osztás a szorzáshoz képest még időigényesebb művelet!

Az osztás problémája

Speciális esetekben az osztás a szorzásnál megismert eljáráshoz hasonlóan jelentős mértékben gyorsítható. Amennyiben a 2 hatványival (2, 4, 8, 16, stb.) osztunk, akkor azt egyszerűen az osztandó értékét tartalmazó regiszter bitenkénti eltolásával meg lehet oldani.

Például a 96/16 műveletet az alábbi módon végezhetjük:

Az osztandó (96) egy 8 bites regiszterben ábrázolva →

Az osztó 16, azaz 2^4 ezért az osztandó regiszterének a bitjeit 4 bittel jobbra mozgatjuk.
A táblázatban négyszer mozgatjuk jobbra egy bittel!

Minden jobbra mozgatáskor a legmagasabb (128) helyiértéken keletkező „üres” bitet 0-val töltjük fel.
Az ábrán piros 0-val jelöltük.

helyiértékek									decimális
128	64	32	16	8	4	2	1		
0	1	1	0	0	0	0	0		96
	eltolás jobbra egy bittel								
0	0	1	1	0	0	0	0		48
	eltolás jobbra egy bittel								
0	0	0	1	1	0	0	0		24
	eltolás jobbra egy bittel								
0	0	0	0	1	1	0	0		12
	eltolás jobbra egy bittel								
0	0	0	0	0	1	1	0		6

A kapott eredmény 6, azaz 96/16.
Az eltolások során a jobb oldalon „kieső” bit általában a Carry bitet változtatja. Így ha nulla a kieső bit, akkor nincs túlcsordulás, ha egy, akkor túlcsordulás keletkezik amit a számításoknál figyelembe kell venni.

A szorzás-osztás képessége

Megjelenés éve	Processzor típusa	Fejlesztő	Órajele		Szorzás/osztás
1971	i 4004	Intel	740 kHz	4 bit	n/n
1972	i 8008	Intel	500-800 kHz	8 bit	n/n
1973	IMP-16	National Semiconductor	715 kHz	Bitszelet (4 bit)	i/i
1974	i 3002	Intel	8,33-16,67 MHz	Bitszelet (2 bit)	n/n
1974	MC 6800	Motorola	1-2 MHz	8 bit	n/n
1974	i 4040	Intel	500-740 kHz	4 bit	n/n
1974	i 8080	Intel	2-3,125 MHz	8 bit	n/n
1975	F8	Fairchild	1-2 MHz	8 bit	n/n
1975	IM 6100	Intersil	0-8 MHz	12 bit	n/n
1976	CDP 1802	RCA	3,2-5 MHz	8 bit	n/n
1976	Z 80	Zilog	2-4 MHz	8 bit	n/n
1976	N8x300	Signetics	6,66-8 MHz	8 bit	n/n
1976	INS 8900	National Semiconductor	2 MHz	16 bit	n/n
1976	TMS 9900	Texas Instruments	3.3 MHz	16 bit	i/i
1978	i8086	Intel	5 MHz	16 bit	i/i
1979	I8088	Intel	5 MHz	8(16) bit	i/i

A szorzás-osztás időigénye (IMP-16)

OP CODE BASE	MNEMONIC AND ASSEMBLER FORMAT		EXECUTION CYCLES	MEMORY CYCLES READ	MEMORY CYCLES WRITE	COMMAND TYPE	FORMAT GROUP
0000	HALT		–	1	–	BASIC	8
0080	PUSHF		4	1	–	↓	8
0100	RTI	[+IMMED]	5	1	–		8
0200	RTS	[+IMMED]	4	1	–		8
0280	PULLF		5	1	–		8
0300	JSRP	+IMMED	8	3	–	EXTENDED	11
0380	JSRI	ADDRESS	4	1	–	BASIC	8
0400	RIN	+IMMED	7	1	–	↓	8
0480	MPY	ADDRESS [(xr)]	106 TO 122	3	–	EXTENDED	9
0490	DIV	ADDRESS [(xr)]	125 TO 159	3	–		9
2800	JSR	ADDRESS [(xr)]	4	1	–		5
2C00	JSR	@ ADDRESS [(xr)]	6	2	–		5
3000	RADD	SOURCE REGISTER, DESTINATION REGISTER	3	1	–		1
3080	RXCH	SOURCE REGISTER, DESTINATION REGISTER	8	1	–		1
3081	RCPY	SOURCE REGISTER, DESTINATION REGISTER	6	1	–		1
3081	NOP		6	1	–		1
3082	RXOR	SOURCE REGISTER, DESTINATION REGISTER	6	1	–		1
3083	RAND	SOURCE REGISTER, DESTINATION REGISTER	6	1	–		1
4000	PUSH	REGISTER	3	1	–		3
4400	PULL	REGISTER	3	1	–		3

Az ábrákon a National Semiconductor IMP-16 processzor utasításainak egy részlete látható. Ez a korai mikroprocesszor igen fejlett volt, igaz nem egy chipes hanem úgynevezett bitszelet technikával készült processzorról beszélünk. Ennek ellenére figyelemreméltó, hogy már 1973-ban olyan processzorral jelent meg a piacon amely utasításai között szerepelt szorzás és osztás.

Látható, hogy az egyik legegyszerűbb 16 bites összeadó utasítás (ADD) ciklusideje 3,
a 16 bites szorzásé (MPY) 106-122, míg a 16 bites osztásé (DIV) 125-159).

A szorzás-osztás időigénye (i8088)

A jobb oldalon az Intel i8086/i8088 (az egykori IBM XT számítógépek) processzorának három utasítás leírása látható.

Az ábrákból jól kiolvasható az egyes utasítások végrehajtási ideje. (Ezen processzorok esetében még a meghajtó órajel jó közelítéssel meghatározta a processzor számítási teljesítményét)

Látható, hogy az egyik legegyszerűbb 16 bites összeadó utasítás (ADD) ciklusideje 3, a 16 bites szorzásé (MUL) 118-133, míg a 16 bites osztásé (DIV) 144-162).

ADD destination,source
Addition

Flags: O D I T S Z A P C
X X X X X X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3	—	2	ADD CX, DX
register, memory	9 + EA	1	2-4	ADD DI, [BX].ALPHA
memory, register	16 + EA	2	2-4	ADD TEMP, CL
register, immediate	4	—	3-4	ADD CL, 2
memory, immediate	17 + EA	2	3-6	ADD ALPHA, 2
accumulator, immediate	4	—	2-3	ADD AX, 200

MUL source
Multiplication, unsigned

Flags: O D I T S Z A P C
X U U U U X

Operands	Clocks	Transfers*	Bytes	Coding Example
reg8	70-77	—	2	MUL BL
reg16	118-133	—	2	MUL CX
mem8	(76-83) + EA	1	2-4	MUL MONTH [SI]
mem16	(124-139) + EA	1	2-4	MUL BAUD_RATE

DIV source
Division, unsigned

Flags: O D I T S Z A P C
U U U U U U

Operands	Clocks	Transfers*	Bytes	Coding Example
reg8	80-90	—	2	DIV CL
reg16	144-162	—	2	DIV BX
mem8	(86-96) + EA	1	2-4	DIV ALPHA
mem16	(150-168) + EA	1	2-4	DIV TABLE [SI]