

000

# Programování ve strojním kódu

## Pro Sinclair ZX Spectrum

$\frac{1}{\text{mole}}$ 
 $\frac{1}{\text{mole}}$ 
 $\frac{1}{\text{mole}}$ 
 $\frac{1}{\text{mole}}$ 
 $\frac{1}{\text{mole}}$

## ASSEMBLER Z 80

[illegible]

Obese

**Keywords:** *work engagement; organizational commitment; turnover intentions; job satisfaction*

1.	Na úvod . . . . .	2
2.	Základní matematické pojmy . . . . .	3
2.1	Binární soustava . . . . .	3
2.2	Hexadecimální soustava . . . . .	8
2.3	Ostatní číselné soustavy . . . . .	9
2.4	Ostatní pojmy - parita . . . . .	10
3.	Hardware počítače Sinclair ZX Spectrum . . . . .	10
3.1	Mikroprocesor Z80 . . . . .	10
4.	Software počítače Sinclair ZX Spectrum . . . . .	12
4.1	Assembler Z80 . . . . .	12
4.2	Strojní instrukce Z80 . . . . .	14
	Instrukce řídicí . . . . .	16
	Instrukce vstupu a výstupu . . . . .	17
	Instrukce přenosu dat . . . . .	21
	Instrukce blokového přenosu dat . . . . .	27
	Aritmetické a logické operace . . . . .	28
	Instrukce blokového prohledávání . . . . .	34
	Inkrementace a dekrementace . . . . .	36
	Instrukce skoků . . . . .	37
	Instrukce volání a návratu . . . . .	40
	Instrukce rotací a posunů . . . . .	42
	Instrukce pro práci s jednotliv. bity . . . . .	49
4.3	Pseudoinstrukce assembleru . . . . .	50
5.	Sestavení strojního programu . . . . .	54
6.	Závěr . . . . .	57

1 2 3 4 5 6 7 8 9 10 11 12

Базисные	Эквивалентные	Средние	Эквивалентные	Базисные	Средние	Средние
----------	---------------	---------	---------------	----------	---------	---------

A - Převodní tabulky HEX - DEC . . . . .	58
B - Abecední seznam strojních instrukcí . . .	59
C - Možné kombinace instrukce LD . . . . .	67
D - Doby provádění strojních instrukcí . . . .	68

## 1. Na úvod

Předpokládám, že většina z Vás, kteří začínáte číst tuto příručku, vlastníte Spectrum a suverénně ovládáte programování v Basicu, ale při zmínce o programu ve strojním kódu Vám běhá mráz po zádech a připadáte si poněkud méněcenní.

Verze Basicu používaná u počítače Sinclair se sice poněkud odchyluje od (neexistujícího) standardu, ale nepracuje se s ní špatně. Určitá funkční omezení počítače Sinclair, vynucená požadavkem na jeho nízkou cenu, se promítají i do jeho Basicu. Verze Basicu Spectra má sice většinu příkazů, ale chybí mu např. příkazy pro počítání s čísly z dvojitou přesností, příkazy pro počítání s komplexními čísly nebo s maticemi. Tato omezení však nejsou podstatná. Hlavní nevýhodou všech verzí Basiců, i těch nejdokonalejších, je jejich pomalost a větší náročnost na paměť.

Při provádění basicového příkazu musí počítač vždy použít překladače z Basicu do strojového kódu procesoru (který je zakódován v ROM). Opakuje-li se stejný příkaz stokrát za sebou, počítač jej musí stokrát přeložit do řeči mikroprocesoru.

Tato pomalost je sice určitou nevýhodou při provádění obsáhlejších matem. výpočtů, ale hlavní překážku tvoří při programování většiny her, u kterých se obvykle požaduje rychlá reakce. Basicový program je příliš pomalý a nestačí reagovat na rychle se měnící herní situaci. V tomto případě sjedná nápravu pouze program sestavený ve strojním kódu. Strojním programem je též možno eliminovat chybějící basicové příkazy u Spectra, kde je možno naprogramovat např. operace s komplexními čísly, maticemi, čísly s dvojnásobnou přesností, výstupy na floppy disk a podobně.

Možnosti stroj. programování jsou prakticky neomezené a teprve programováním ve stroj. kódu můžeme plně využít všech schopností mikroprocesoru Z 80, který je základem našeho Sinclairu. Stačí si uvědomit, že tentýž Z 80 pracuje v řadě jiných počítačů, mnohonásobně výkonnějších (a bohužel dražších) než je naše Spectrum. Různá omezení Spectra nejsou dána použitým mikroprocesorem, ale tím, co ho obklopuje (a toho je u Spectra málo). Existují dva krajní názory na mikroprocesor. Jeden názor tvrdí, že mikroprocesor je univerzální génius, druhý říká, že mikroprocesor je geniální blbec. V obou názorech je část pravdy. Celou pravdou je to, že mikroprocesor umí jen to, co mu formou programu předloží programátor a je tak dobrý, jak dobrý je jeho programátor.

Nic nestojí v cestě tomu, aby i Vy jste se stali experty v programování ve strojním kódu. Není to nic světoborného. Chce to jen trochu logického uvažování a hodně trpělivosti. Musíte vyjít od těch nejjednodušších programů a postupně experimentovat se stále složitějšími aplikacemi. Zjistíte, že původní strašidelnost se vytrácí, a že jste schopni ve str. kódu řešit i komplikované programy, resp. jejich kombinace s programy Basicovými.

Ale POZOR! Tento stručný návod není samospasitelný a jeho přečtením se z Vás nestanou rázem experti. Je to pouze začátek a předpokládá další studium.

## 2. Základní matematické pojmy

Protože z vlastní zkušenosti vím, jak je někdy obtížné získat odpovídající literaturu, resp. potřebné informace bez konzumace spousty dalších nepotřebných údajů, začnu od Adama a v první kapitole se pokusím pro ty, kteří se doposud podobnými problémy nezabývali, stručně shrnout základní matematické pojmy, bez nichž není možno úspěšně zvládnout programování ve stroj. kódu.

Mikroprocesor je elektrické zařízení, pracující (zjednodušeně řečeno) na bázi "vypnuto - zapnuto". Základními pracovními prvky všech mikroprocesorů jsou tzv. registry, což jsou soustavy elektrických prvků, které jsou schopny uschovávat, resp. různé kombinovat a převádět právě tyto dva stavy a nic víc. Matematicky toto "vypnuto - zapnuto" vyjadřujeme hodnotami "0" a "1". Musíme tedy pro všechny další úvahy vystačit pouze s nulou a jedničkou.

### 2.1 Binární soustava

Matematický systém používající pouze hodnot 0 a 1 je nazýván binárním nebo dvojkovým systémem. V tomto systému jsou všechny číselné hodnoty vyjádřeny jako mocniny čísla 2:

$2 \text{ na } 0 = 1$ ,  $2 \text{ na } 1 = 2$ ,  $2 \text{ na } 2 = 4$ ,  $2 \text{ na } 3 = 8$ ,  $2 \text{ na } 4 = 16$  atd.

Kombinací těchto mocnin pak můžeme vyjádřit jakékoliv číslo. Forma tohoto vyjádření je soustava nul a jedniček, např. 1010. Toto číslo však nevyjadřuje hodnotu 1010, jak jsme zvyklí z běžné desítkové soustavy, ale hodnotu 10. Aby nedocházelo k omylům, označují se čísla v dvojkové (binární) soustavě písmenem B a čísla v desítkové či decimální soustavě písmenem D (případně b a d). Můžeme pak napsat  $1010_b = 10_d$ ; číselná řada ve dvojkové soustavě vypadá následovně :

$$\begin{aligned}
 0_b &= 0_d \\
 1_b &= 1_d \\
 10_b &= 2_d \\
 11_b &= 10_b + 1_b = 2_d + 1_d = 3_d \\
 100_b &= 4_d \\
 101_b &= 100_b + 1_b = 4_d + 1_d = 5_d \\
 110_b &= 100_b + 10_b = 4_d + 2_d = 6_d \\
 111_b &= 100_b + 10_b + 1_b = 4_d + 2_d + 1_d = 7_d \\
 1000_b &= 8_d \\
 1001_b &= 1000_b + 1_b = 8_d + 1_d = 9_d
 \end{aligned}$$

I když to vypadá složitě, systém je jednoduchý: 0 znamená nula, 1 znamená, že na daném místě je odpovídající mocnina dvou. Mocniny dvou se postupně zvyšují z prava doleva. Zcela vpravo je vždy  $2^0$ , každý další posun doleva znamená násobení dvěma, tj. o řád vyšší mocnina 2:

. . . . .	2 na 5	2 na 4	2 na 3	2 na 2	2 na 1	2 na 0
. . . . .	32	16	8	4	2	1

Převod kladných čísel na záporná se provádí tak, že se invertují všechny bity včetně znaménkového (tj. místo 1 se napíše 0 a naopak) a k takto vzniklému číslu se přičte 1.

$$\begin{array}{r} +85D = 01010101b \\ \quad \quad 10101010b \quad - \text{invertováno} \\ + \quad \quad \quad 1b \\ \hline -85D = 10101011b \end{array}$$

Při převodu záporných čísel na kladná se postupuje tak, že od daného čísla se odečte 1 a takto vzniklé číslo se invertuje:

$$\begin{array}{r} -103D = 10011001b \\ \quad \quad \quad 1b \\ \hline 10011000b \\ +103D = 01100111b \quad - \text{invertováno} \end{array}$$

Můžeme postupovat i následujícím způsobem, který platí jak pro převod kladných čísel na záporná, tak i pro převod záporných čísel na kladná. Postupujeme zprava doleva. Nuly ponecháváme až narazíme na první jedničku. Tu také ještě ponecháme, všechny ostatní bity invertujeme:

$$\begin{array}{l} +84d = 01010100b \\ \text{invertujeme} \rightarrow 1 \quad 11 \quad 1 \quad \leftarrow \text{ponecháme první tři čísla zprava} \\ -84d = 10101100b \\ \\ -103d = 10011001b \\ \text{invertujeme} \rightarrow 1 \quad 11 \quad \leftarrow \text{jedničku ponecháme} \\ +103d = 10011001b \end{array}$$

#### Aritmetické operace s binárními čísly

Binární čísla můžeme sečítat, odečítat, násobit i dělit. Platí pro to následující pravidla:

- sčítání binárních čísel:

$0b + 0b = 0b$   
 $0b + 1b = 1b$   
 $1b + 1b = 10b$ , což můžeme také vyjádřit  $1b + 1b = 0b + \text{přenos}$  do dalšího místa.

Tyto zásady platí pro jakoukoliv pozici bitu v binárním čísle. Nejlépe je to zřejmé na příkladech:

$$\begin{array}{r} 1d \quad \quad 01b \\ +1d \quad \quad +01b \\ \hline 2d \quad \quad 10b \\ \quad \quad \quad | \\ \quad \quad \quad \text{přenos} \end{array}$$

Náš uvedený příklad, tj. 1010b vyjadřuje:

$$1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 8 + 0 + 2 + 0 = 10d$$

Jedničky a nuly v binárním čísle nazýváme bity. Osm bitů tvoří bajt (anglicky byte). Osm bitů proto, že procesor Z80 a řada dalších pracuje právě s osmimístními registry. Maximální hodnota jednoho bajtu může být 11111111b = 255d

Při práci se strojním kódem budeme potřebovat nejen kladná, ale i záporná čísla. V binární soustavě se používá k vyjádření záporného čísla tzv. doplňkového kódu. Je to další bit přidáný k binárnímu číslu. Je-li jeho hodnota 0, jedná se o kladné číslo, je-li to 1, jedná se o záporné číslo. Při běžně používaných osmibitových slovech budeme nejvyšší bit považovat za znaménkový bit a k vyjádření hodnoty pak zbývá jen sedm bitů. To znamená, že v tomto případě můžeme osmi bity vyjádřit maximální hodnotu od -128 do +127.

Pozn.: osm bitů někdy nazýváme slovem, čtyři bity slabikou.

Zápornou hodnotu čísla získáme tak, že toto číslo odečteme od čísla 2 na n, kde n je počet významových bitů. V případě osmibit. čísla je první bit znaménkový a zbývajících sedm bitů významových. Hodnota od které odečítáme, je potom  $2^7 = 128$ . Pak  $-1 = 128 - 1 = 127$ , což vyjádřeno binárně je:

$$11111111b = -1d$$

?

první jednička je znaménkový bit

$$-2 \text{ je } 128 - 2 = 126, \text{ binárně } 11111110b$$

$$-3 \text{ je } 128 - 3 = 125, \text{ binárně } 11111101b$$

atd. Nejlépe je to vidět na číselné řadě:

$$127d = 01111111b$$

:

:

$$5d = 00000101b$$

$$4d = 00000100b$$

$$3d = 00000011b$$

$$2d = 00000010b$$

$$1d = 00000001b$$

$$0d = 00000000b$$

$$-1d / 127d / = 11111111b$$

$$-2d / 126d / = 11111110b$$

$$-3d / 125d / = 11111101b$$

$$-4d / 124d / = 11111100b$$

$$-5d / 123d / = 11111011b$$

:

:

$$-127d / 1d / = 10000001b$$

$$-128d / 0d / = 10000000b$$

$$\begin{array}{r}
 3d \\
 +1d \\
 \hline
 4d
 \end{array}
 \qquad
 \begin{array}{r}
 11b \\
 +01b \\
 \hline
 10 \\
 +1 \\
 \hline
 100b
 \end{array}
 \qquad
 \begin{array}{l}
 1 \text{ krok} \\
 \text{přenos} \\
 2 \text{ krok}
 \end{array}$$
  

$$\begin{array}{r}
 5d \\
 +7d \\
 \hline
 12d
 \end{array}
 \qquad
 \begin{array}{r}
 101b \\
 +111b \\
 \hline
 010 \\
 +1 \\
 \hline
 1100b \\
 | \\
 \text{přenos}
 \end{array}
 \qquad
 \begin{array}{l}
 1 \text{ krok} \\
 \text{přenos} \quad 2 \text{ krok}
 \end{array}$$

- odečítání binárních čísel:

Odečítáme tak, že sčítáme dvojkové doplňky, tj. záporná čísla, podle pravidla  $A-B = A+(-B)$ , nebo  $-A-B = +(-A)+(-B)$

$$\begin{array}{r}
 12d \\
 + -7d \\
 \hline
 5d
 \end{array}
 \qquad
 \begin{array}{r}
 00001100b \\
 11111001b \\
 \hline
 00000101b
 \end{array}$$

Vystačíme tak vlastně pro sečítání i odečítání s jedním početním úkonem.

- násobení a dělení binárních čísel:

Každý posun bitů o jedno místo doleva znamená násobení dvěma, naopak posun o jedno místo doprava znamená dělení dvěma:

$$\begin{array}{ll}
 0001b = 1d & 1000b = 8d \\
 0010b = 1 * 2 = 2d & 0100b = 8 / 2 = 4d \\
 0100b = 2 * 2 = 4d & 0010b = 4 / 2 = 2d \\
 1000b = 4 * 2 = 8d & 0001b = 2 / 2 = 1d
 \end{array}$$

Můžeme pokračovat do prava za desetinnou tečku, takže:

$$\begin{array}{lll}
 0,1b & = 1 / 2 = 0,5d & = 2 \text{ na } -1 \\
 0,01b & = 0,5 / 2 = 0,25d & = 2 \text{ na } -2 \\
 0,001b & = 0,25 / 2 = 0,125d & = 2 \text{ na } -3 \text{ atd.}
 \end{array}$$

Tímto způsobem můžeme vyjádřit i desetinná čísla. Chceme-li násobit dvě binární čísla, musíme násobitele rozdělit na násobky dvou. Např.  $10d * 10d = 10d * (2d + 8d)$ .

$$\begin{array}{rcl}
 10d & = & 1010b \\
 10d * 2d & = & 10100b \quad 1 * \text{posun doleva tj. přidá se jedna nula} \\
 +10d * 8d & = & +1010000b \quad 3 * \text{posun doleva tj. přidají se tři nuly} \\
 \hline
 10d * 10d & = & 1100100b
 \end{array}$$

Jiný příklad:  $10d * 5d = 10d * (1d + 4d)$

$10d$	$=$	$1010b$	
$10d * 1d$	$=$	$1010b$	nic se nemění
$+10d * 4d$	$=$	$+101000b$	2 * posun doleva tj. přidají se dvě nuly
-----		-----	
$10d * 5d$		$110010b$	

Poznámka - místo desetinné čárky se zásadně používá tečka a pro násobení hvězdičky a dělení zlomkové čáry.

Dělení dvou binárních čísel je opakem násobení. Dělíme tak, že od dělence postupně odečítáme jeho násobky dvou, podle toho, je-li příslušný násobek obsažen v děliteli formou jedničky. Je-li na daném místě dělitele nula, neodečítáme nic. Násobky dvou dostaneme posunem dělence o jedno místo doprava.

Při všech aritmetických operacích musíme dbát na to, aby nedošlo k tzv. přetečení. Přetečení nastane tehdy, dojde-li k přenosu do neexistujícího bitu, nebo do znaménkového bitu a dojde tak ke zkreslení výsledku. Ve čtyřbitovém registru bude např. při součtu  $1111b + 0001b = 1\ 0000b$ . Výsledek v registru je  $0000b$ , což je nepravdivé.

|  
neexistující bit

Logické operace.

Vedle běžné algebry, tak jak ji znáte ze školy, existuje ještě Booleova algebra, která nepracuje s čísly, ale s pojmy pravda, nepravda. Pravda je vyjádřena jedničkou, nepravda nulou. V Booleově algebře existují tři operace logický součin, označovaný AND, logický součet, označovaný OR a binární nebo exklusivní součet, označovaný XOR. Pro jednotlivé funkce platí:

- logický součin:  $1 \text{ AND } 0 = 0$                        $0 \text{ AND } 0 = 0$   
 $1 \text{ AND } 1 = 1$                                        $0 \text{ AND } 1 = 0$
- logický součet:  $1 \text{ OR } 0 = 1$   
 $0 \text{ OR } 0 = 0$   
 $1 \text{ OR } 1 = 1$   
 $0 \text{ OR } 1 = 1$
- exklusivní (binární) součet:  $1 \text{ XOR } 0 = 1$   
 $0 \text{ XOR } 0 = 0$   
 $1 \text{ XOR } 1 = 0$   
 $0 \text{ XOR } 1 = 1$

Příklady jednotlivých operací:

AND	OR	XOR
0011	0011	0011
0101	0101	0101
----	----	----
0001	0111	0110

## 2.2 Hexadecimální soustava

Práce s kolonami jedniček a nul je značně nepřehledná a vede ke vzniku chyb. Pro ulehčení této práce byly vytvořeny další soustavy, z nichž je nejpoužívanější soustava hexadecimální či šestnáctková.

Tato soustava vznikne tak, že se sečtou hodnoty vždy čtyř bitů a takto vzniklé číslo je číslicí hexadecimální soustavy. Různými kombinacemi 4 bitů dostaneme nejvýše 16 hodnot, odtud tedy název soustavy - šestnáctková. Poněvadž číslice základního modulu soustavy musí být jednociferné, čehož nemohlo být dosaženo normálními číslicemi 0 až 9, bylo přibráno prvních 6 písmen abecedy tak, že po číslici 9 nenásleduje číslice 10, ale A a pokračuje se postupně až po F. Hodnoty vyjádřené v hexadecimální soustavě označujeme písmenem H (nebo h) připojeným za číslice této soustavy.

Číselná řada v šestnáctkové soustavě vypadá následovně:

0d	=	0000b	=	0h
1d	=	0001b	=	1h
2d	=	0010b	=	2h
3d	=	0011b	=	3h
4d	=	0100b	=	4h
5d	=	0101b	=	5h
6d	=	0110b	=	6h
7d	=	0111b	=	7h
8d	=	1000b	=	8h
9d	=	1001b	=	9h
10d	=	1010b	=	Ah
11d	=	1011b	=	Bh
12d	=	1100b	=	Ch
13d	=	1101b	=	Dh
14d	=	1110b	=	Eh
15d	=	1111b	=	Fh

Chceme-li vyjádřit v hexadecimální (HEX) soustavě hodnoty vyšší než 15, musíme použít dvou a více HEX-číslic:

16d	=	0001 0000b	=	10h
17d	=	0001 0001b	=	11h
				:
				:
27d	=	0001 1011	=	1Bh

Každá čtveřice bitů má svoji odpovídající Hex-číslici. Můžeme tak vytvořit libovolnou hodnotu. Prakticky nejvyšší Hex-číslo, které budeme používat, je FFFFh, což odpovídá 16-ti bitovému číslu 1111111111111111b nebo 65535d. Nejvyšší proto, že procesor Z80 může pracovat maximálně se šestnáctibitovými čísly (v binární soustavě).

Počítání v HEX-soustavě je značně jednodušší, než v soustavě binární. Problém je při přechodu z HEX na desetinnou soustavu nebo při aritmetických operacích. Při převodech decimálních



čísels na hexadecimalní se používá tabulek nebo různých programů pro převody (příloha A). Při aritmetických nebo logických operacích musíme obvykle HEX číslo převést do binárního tvaru (někdy také postačí do desetinného tvaru), provést operaci a výsledek znovu převést do HEX-formy.

Bylo-li v binární soustavě 11111111b rovno -1d, platí to obdobně i v HEX tvaru; FFh je rovno 255d, ale také -1d, FEh je jednak 254d, ale i -2d. Nejlépe je to zase zřejmé z číselné řady:

127d	=	0111 1111b	=	7Fh
:				
3d	=	0000 0011b	=	03h
2d	=	0000 0010b	=	02h
1d	=	0000 0001b	=	01h
0d	=	0000 0000b	=	00h
-1d	=	1111 1111b	=	FFh
-2d	=	1111 1110b	=	FEh
-3d	=	1111 1101b	=	FDh
-4d	=	1111 1100b	=	FDh
:				
-127d	=	1000 0001b	=	81h
-128d	=	1000 0000b	=	80h

Tohoto způsobu se používá např. při zadání velikosti odskoku u relativních skoků, proto je třeba si systém záporných čísel v HEX soustavě dobře osvojit. V počítačové literatuře se používá pro binární čísla znaku % a pro HEX čísla znaku # a píšs se před číslo. Tyto symboly budou dále uváděny ve výpisech assembleru a v textu ve spojení s instrukcemi.

## 2.3 Ostatní číselné soustavy

Vedle binární a hexadecimalní soustavy se v počítačové praxi užívají ještě další dvě soustavy. Je to soustava osmičková či oktalová a soustava "BCD". Při další práci je nebudeme potřebovat, proto jen stručně o obou:

### Oktalová soustava

Je to obdoba šestnáctkové soustavy, ale vychází jen ze tří bitů. Maximální počet kombinací tří bitů je osm, proto osmičková. Pracuje pouze s číslicemi 0 až 7, číslice 8 a 9 nezná.

### BCD soustava

Soustava BCD převádí jednotlivé číslice desítkového čísla do 4 bitů binární soustavy a naopak: 327d = 0011 0010 0111 BCD. Každá číslice je tedy představována vždy čtveřicí bitů. Procesor Z80 má speciální příkazy pro počítání v BCD soustavě.

## 2.4 Ostatní pojmy - parita

Paritou rozumíme součet jedniček v binárním čísle. Je-li tento součet sudý, je parita sudá, je-li tento součet lichý, je parita lichá. Používá se jako kontrolní hodnota při práci s binárními hodnotami.

## 3. Hardware počítače Sinclair ZX Spectrum =====

ZX Spectrum se vyznačuje tím, že obsahuje minimální počet integrovaných obvodů. A sice 13 integrovaných obvodů, z čehož 10 obvodů je 16k bajtů paměti a její pomocné obvody. Je to pravděpodobně světový rekord a je tím podmíněna nízká cena. Na druhé straně to představuje určitá omezení. Poněvadž tato práce se nezabývá hardwarovou stránkou Sinclairu, t.j. možnostmi připojení dalších obvodů a zařízení, uvedu jen to nejnnutnější.

### 3.1 Mikroprocesor Z 80

Hlavní součástí každého počítače je centrální procesorová jednotka (CPU), která jednak vykonává veškeré početní operace, jednak řídí činnost celého počítače. U Spectra je použito CPU firmy Zilog, typ Z80A, což je rychlejší verze původního typu Z80. Je to jeden z nejrozšířenějších a nejvýkonnějších osmibitových procesorů. Umožňuje provádět 158 strojových instrukcí. Z80 může adresovat paměť do rozsahu 64k bajtů, čehož ZX Spectrum plně využívá.

Nejdůležitější součástí každého procesoru jsou jeho registry. Z80 má celkem 22 registrů :

- registr A (akumulátor). Osmibitový registr, v němž se provádí většina aritmetických a logických operací.
- registr F (stavový registr). Jednotlivé bity tohoto registru informují systém o výsledcích jednotlivých operací. Osm bitů tohoto registru, tzv. příznakové bity nebo stavové bity, mají následující označení a význam:
  - bit 7 : znaménkový bit. Je nastaven, t.j. má hodnotu 1, je-li při matematických operacích nastaven nejvyšší bit registru (t.j. je-li v nejvyšším bitu 1, což znamená zápornou hodnotu).
  - bit 6 : indikátor nuly. Bit je nastaven, je-li výsledkem aritmetických nebo logických operací v akumulátoru roven nule.
  - bit 5 : neobsazen, nepoužívá se

- bit 4 : indikátor polovičního přenosu. Používá se při instrukci DAA a indikuje přenos mezi 3 a 4 bitem.
- bit 3 : neobsazen, nepoužívá se.
- bit 2 : indikátor parity a přetečení. Bit je nastaven, dojde-li při aritmetických operacích k přetečení, t.j. je-li výsledkem číslo větší než kapacita registrů. Při logických operacích a rotacích je nastaven, je-li parita sudá.
- bit 1 : indikátor sčítání a odčítání. Používá se při DAA a je nastaven, byla-li předchozí operace odčítání.
- bit 0 : indikátor přenosu. Bit je nastaven, dojde-li při aritmetických operacích k přenosu ze znaménkového bitu.

Podrobněji budou jednotlivé funkce stavových bitů tohoto registru popsány při popisu strojových instrukcí. Jejich stav podmiňuje provedení některých strojních instrukcí.

- registry B, C, D, E, H, L. Osmibitové všeobecné registry pro manipulaci s daty. Tyto registry je možno spojovat do šestnáctibitových registrových páru BC, DE, HL. Registrový pár BC je pro nás zajímavý tím, že po návratu ze strojního programu do Basicu lze jeho obsah vypsat na obrazovku, bylo-li ke spuštění strojního programu použito instrukce PRINT USR.
- registry IX, IY. Šestnáctibitové indexové registry pro tzv. indexové adresování.
- registr PC. Programový čítač, šestnáctibitový registr obsahující adresu prováděné instrukce. Při každém zapojení počítače je vynulován a jeho obsah se postupně zvyšuje o jedničku.
- registr SP. Ukazatel zásobníku, šestnáctibitový registr obsahující adresu uživatelského zásobníku (stack).
- registr I. Osmibitový registr vektoru přerušení.
- registr R. Osmibitový registr, jehož prostřednictvím provádí Z80 automatické osvěžování dynamických pamětí. Na sběrnici je vyvedeno ale jen sedm bitů.

Vedle registru A, F, B, C, D, E, H a L má Z80 ještě registry A', F', B', C', D', E', H' a L'. Jsou to tzv. zdvojené registry. Jsou shodné se základními registry a mohou si s nimi vyměňovat obsah. Programově přímo přístupné nejsou.

Pro lepší názornost ještě tabulka všech registrů Z80:

```

.....
.  A  .  F  .  A' .  F' .
.....
.  B  .  C  .  B' .  C' .
.....
.  D  .  E  .  D' .  E' .
.....
.  H  .  L  .  H' .  L' .
.....
.  IX  .
.....
.  IV  .
.....
.  PC  .
.....
.  SP  .
.....
.  I  .  R  .
.....

```

## 4. Software počítače Sinclair ZX Spectrum =====

### 4.1 Assembler Z80 -----

Každý mikroprocesor rozumí pouze pevně stanovenému souboru strojních instrukcí, které jsou vyjádřeny kombinacemi jednoho až čtyř bajtů. Tak pro Z80 znamená bajt 76h strojní instrukci HALT. Program ve strojním kódu, kterému jediné mikroprocesor rozumí, je pak posloupnost strojních instrukcí, vyjádřená jako posloupnost bajtů. Takový jednoduchý program může vypadat např. následovně:

```
01h 00h 1Bh 21h 00h 40h 11h 00h 80h 00h EDh 80h C9h
```

(tento program přeneseme obsah obrazovky do oblasti za adresu 32768d). Pamatovat si, že instrukce pro "HALT" je 76h je dost nepohodlné a při práci s jednotlivými bajty se dopustíme určité řady chyb. Proto byl vyvinut jazyk symbolických adres, nazývaný též assemblerem, který značně usnadňuje práci se strojními instrukcemi mikroprocesoru. Původně je assembler označení programu, který překládá program napsaný v jazyku symbolických adres do strojního kódu procesoru, ale pro jednoduchost se pro jazyk symbolických adres používá označení assembler. Pokud bude někde potřeba rozlišení těchto dvou pojmů, budeme pro jazyk symbolických adres používat výrazu assembler a pro pomocný program výrazu Assembler.

Pro sestavování programu v assembleru platí určitá obecná pravidla. Každá část - návěští, instrukce, operandy a poznámky je od ostatních oddělena mezerami a před poznámkou musí být středník. Program uvedený v předchozím odstavci vypadá v assembleru takto:

návěští	instrukce	operandy	poznámky
začátek	LD BC,	6912	; do reg. BC hodnota 6912
	LD HL,	#4000	; do reg. HL počátek obrazu
	LD DE,	32768	; do reg. DE kam přenést
	NOP		; žádná operace
	LDIR		; blokový přenos
konec	RET		; zpět do BASICu

Nyní musíme provést to, co normálně provádí Assembler, t.j. jednotlivým instrukcím assembleru musíme přiřadit odpovídající hexadecimální kódy strojních instrukcí Z 80. Výsledkem bude relativní strojní program, t.j. program, kterému už bude Z 80 rozumět, který ale ještě nefunguje, poněvadž chybí počáteční adresa programu:

relat. stroj. program	návěští	instr.	operandy	poznámky
01 00 1B	začátek	LD BC,	6912	;do reg. BC 6912
21 00 40		LD HL,	#4000	;do reg. HL #4000
11 00 80		LD DE,	32768	;do reg. DE 32768
00		NOP		;žádná operace
ED B0		LDIR		;blokový přenos
C9	konec	RET		;zpět do Basicu

Zvolením počáteční adresy, např. #7000, dostaneme absolutní strojní program, který můžeme vložit do počítače a který již počítač provede. Poněvadž se v programech obvykle používá i absolutních skoků, píšeme pak takový program tak, že na první místo ještě napíšeme jednotlivé adresy. Definitivní výsledek naší práce pak bude vypadat následovně :

adresa	kód	náv.	instr.	operand	poznámka
#7000	01 00 1B	zač	LD BC	6912	;do reg. BC 6912
#7003	21 00 40		LD HL	#4000	;do reg. HL #4000
#7006	11 00 80		LD DE	32768	;do reg. DE 32768
#7009	00		NOP		;žádná operace
#700A	ED B0		LDIR		;blokový přenos
#700C	C9	kon	RET		;návrat do Basicu

Tento program přeneseme 6912 bajtů z paměti obrazovky tj. od adresy 16386 do oblasti od adresy 32768, kde je uložen pro pozdější použití. Před vyvoláním podprogramu je potřeba snížit RAMTOP instrukcí CLEAR v Basicu např.: CLEAR 32767.

To bylo na úvod. K assembleru se ještě na závěr kapitoly vrátíme, ale nyní se musíme věnovat jednotlivým strojním instrukcím Z 80, poněvadž bez jejich perfektní znalosti nám ani nejlepší Assembler nepomůže.

## 4.2 Strojní instrukce Z 80

Mikroprocesor Z 80 může provádět 158 strojních instrukcí. Tyto instrukce můžeme rozdělit do následujících skupin :

- řídicí instrukce      NOP, HALT, DI, EI, IM, CCF, SCF
- vstup a výstup      IN, INI, INIR, IND, INDR, OUT, OUTI, OTIR, OUTD, OTDR
- přenos dat            LD, PUSH, POP, EX, EXX
- blokový přenos      LDI, LDIR, LDD, LDDR
- aritmetické a logické operace      ADD, ADC, SUB, SBC, DAA, CPL, NEG, AND, OR, XOR, CP
- blokové prohledávání      CPI, CPIR, CPD, CPDR
- inkrementace a dekrementace      INC, DEC
- skoky                  JP, JR, JR DIS, DJNZ
- volání a návraty      CALL, RET, RETI, RETN, RST
- rotace a posuny      RLCA, RLA, RRLA, RRA, RLC, RL, RRC, RLD, RR, SRA, SRC
- bitové operace      BIT, SET, RES

Jednotlivé instrukce se označují mnemotechnickými zkratkami, např. LD pro load (= přenes, vlož ), INC pro increment (= zvýš ) JP pro jump (= skoč ) a pod. Ten kdo zná alespoň základy angličtiny to má jednodušší, ale ani ten kdo o angličtinu v životě nezavádil si nemusí zoufat, poněvadž toho zase není tak moc.

V dalším textu bude použito následujícího způsobu značení jednotlivých pojmů:

- mnemotechnické zkratky budou psány velkými písmeny
- označení registrů a čísel bude psáno velkými písmeny (někde v literatuře jsou malými písmeny i registry)
- registr obecně bude označen r
- jednobajtové číslo bude obecně značeno n
- hodnota odskoku u relativních skoků bude vyjadřovaná také jednobajtovým číslem n
- dvoubajtové číslo bude obecně nn, případně mn
- index bude obecně označován d (jednobajtové číslo)
- číslo v závorce znamená obsah pamětového místa v závorce
- registr v závorce znamená obsah pamětového místa, jehož adresa je v daném registru
- A ← H znamená, že obsah registru H se přenese do registru A
- JP 1.,2. znamená, že skok na operand 2 se provede tehdy, je-li splněná podmínka daná operandem 1
- pro rozlišení registru C a přenosového bitu C v registru F bude pro přenosový bit použito označení CY
- -CY znamená opačnou hodnotu než má CY=C (je-li CY=0, je -CY=1)

Než se pustíme do jednotlivých strojních instrukcí Z 80, musíme se vrátit ke stavovým indikátorům, což jsou některé bity registru F, o kterém byla zmínka v kapitole 3.1. Tyto indikátory poskytují informace o stavu CPU a především o způsobu dokončení poslední aritmetické operace. Jejich poloha v registru F je na následujícím schématu :

## registř F

č. bitu	indikátor											
	1	2	3	4	5	6	7	8	9	10	11	12
	1	0	1	1	2	1	1	1	3	1	2	1
	1	0	1	1	4	1	3	1	4	1	5	1
	1	0	1	1	5	1	4	1	5	1	6	1
	1	0	1	1	6	1	5	1	6	1	7	1
	1	0	1	1	7	1	6	1	7	1	8	1
	1	0	1	1	8	1	7	1	8	1	9	1
	1	0	1	1	9	1	8	1	9	1	10	1
	1	0	1	1	10	1	9	1	10	1	11	1
	1	0	1	1	11	1	10	1	11	1	12	1
	1	0	1	1	12	1	11	1	12	1	13	1
	1	0	1	1	13	1	12	1	13	1	14	1
	1	0	1	1	14	1	13	1	14	1	15	1
	1	0	1	1	15	1	14	1	15	1	16	1
	1	0	1	1	16	1	15	1	16	1	17	1
	1	0	1	1	17	1	16	1	17	1	18	1
	1	0	1	1	18	1	17	1	18	1	19	1
	1	0	1	1	19	1	18	1	19	1	20	1
	1	0	1	1	20	1	19	1	20	1	21	1
	1	0	1	1	21	1	20	1	21	1	22	1
	1	0	1	1	22	1	21	1	22	1	23	1
	1	0	1	1	23	1	22	1	23	1	24	1
	1	0	1	1	24	1	23	1	24	1	25	1
	1	0	1	1	25	1	24	1	25	1	26	1
	1	0	1	1	26	1	25	1	26	1	27	1
	1	0	1	1	27	1	26	1	27	1	28	1
	1	0	1	1	28	1	27	1	28	1	29	1
	1	0	1	1	29	1	28	1	29	1	30	1
	1	0	1	1	30	1	29	1	30	1	31	1
	1	0	1	1	31	1	30	1	31	1	32	1
	1	0	1	1	32	1	31	1	32	1	33	1
	1	0	1	1	33	1	32	1	33	1	34	1
	1	0	1	1	34	1	33	1	34	1	35	1
	1	0	1	1	35	1	34	1	35	1	36	1
	1	0	1	1	36	1	35	1	36	1	37	1
	1	0	1	1	37	1	36	1	37	1	38	1
	1	0	1	1	38	1	37	1	38	1	39	1
	1	0	1	1	39	1	38	1	39	1	40	1
	1	0	1	1	40	1	39	1	40	1	41	1
	1	0	1	1	41	1	40	1	41	1	42	1
	1	0	1	1	42	1	41	1	42	1	43	1
	1	0	1	1	43	1	42	1	43	1	44	1
	1	0	1	1	44	1	43	1	44	1	45	1
	1	0	1	1	45	1	44	1	45	1	46	1
	1	0	1	1	46	1	45	1	46	1	47	1
	1	0	1	1	47	1	46	1	47	1	48	1
	1	0	1	1	48	1	47	1	48	1	49	1
	1	0	1	1	49	1	48	1	49	1	50	1
	1	0	1	1	50	1	49	1	50	1	51	1
	1	0	1	1	51	1	50	1	51	1	52	1
	1	0	1	1	52	1	51	1	52	1	53	1
	1	0	1	1	53	1	52	1	53	1	54	1
	1	0	1	1	54	1	53	1	54	1	55	1
	1	0	1	1	55	1	54	1	55	1	56	1
	1	0	1	1	56	1	55	1	56	1	57	1
	1	0	1	1	57	1	56	1	57	1	58	1
	1	0	1	1	58	1	57	1	58	1	59	1
	1	0	1	1	59	1	58	1	59	1	60	1
	1	0	1	1	60	1	59	1	60	1	61	1
	1	0	1	1	61	1	60	1	61	1	62	1
	1	0	1	1	62	1	61	1	62	1	63	1
	1	0	1	1	63	1	62	1	63	1	64	1
	1	0	1	1	64	1	63	1	64	1	65	1
	1	0	1	1	65	1	64	1	65	1	66	1
	1	0	1	1	66	1	65	1	66	1	67	1
	1	0	1	1	67	1	66	1	67	1	68	1
	1	0	1	1	68	1	67	1	68	1	69	1
	1	0	1	1	69	1	68	1	69	1	70	1
	1	0	1	1	70	1	69	1	70	1	71	1
	1	0	1	1	71	1	70	1	71	1	72	1
	1	0	1	1	72	1	71	1	72	1	73	1
	1	0	1	1	73	1	72	1	73	1	74	1
	1	0	1	1	74	1	73	1	74	1	75	1
	1	0	1	1	75	1	74	1	75	1	76	1
	1	0	1	1	76	1	75	1	76	1	77	1
	1	0	1	1	77	1	76	1	77	1	78	1
	1	0	1	1	78	1	77	1	78	1	79	1
	1	0	1	1	79	1	78	1	79	1	80	1
	1	0	1	1	80	1	79	1	80	1	81	1
	1	0	1	1	81	1	80	1	81	1	82	1
	1	0	1	1	82	1	81	1	82	1	83	1
	1	0	1	1	83	1	82	1	83	1	84	1
	1	0	1	1	84	1	83	1	84	1	85	1
	1	0	1	1	85	1	84	1	85	1	86	1
	1	0	1	1	86	1	85	1	86	1	87	1
	1	0	1	1	87	1	86	1	87	1	88	1
	1	0	1	1	88	1	87	1	88	1	89	1
	1	0	1	1	89	1	88	1	89	1	90	1
	1	0	1	1	90	1	89	1	90	1	91	1
	1	0	1	1	91	1	90	1	91	1	92	1
	1	0	1	1	92	1	91	1	92	1	93	1
	1	0	1	1	93	1	92	1	93	1	94	1
	1	0	1	1	94	1	93	1	94	1	95	1
	1	0	1	1	95	1	94	1	95	1	96	1
	1	0	1	1	96	1	95	1	96	1	97	1
	1	0	1	1	97	1	96	1	97	1	98	1
	1	0	1	1	98	1	97	1	98	1	99	1
	1	0	1	1	99	1	98	1	99	1	100	1
	1	0	1	1	100	1	99	1	100	1	101	1
	1	0	1	1	101	1	100	1	101	1	102	1
	1	0	1	1	102	1	101	1	102	1	103	1
	1	0	1	1	103	1	102	1	103	1	104	1
	1	0	1	1	104	1	103	1	104	1	105	1
	1	0	1	1	105	1	104	1	105	1	106	1
	1	0	1	1	106	1	105	1	106	1	107	1
	1	0	1	1	107	1	106	1	107	1	108	1
	1	0	1	1	108	1	107	1	108	1	109	1
	1	0	1	1	109	1	108	1	109	1	110	1
	1	0	1	1	110	1	109	1	110	1	111	1
	1	0	1	1	111	1	110	1	111	1	112	1
	1	0	1	1	112	1	111	1	112	1	113	1
	1	0	1	1	113	1	112	1	113	1	114	1
	1	0	1	1	114	1	113	1	114	1	115	1
	1	0	1	1	115	1	114	1	115	1	116	1
	1	0	1	1	116	1	115	1	116	1	117	1
	1	0	1	1	117	1	116	1	117	1	118	1
	1	0	1	1	118	1	117	1	118	1	119	1
	1	0	1	1	119	1	118	1	119	1	120	1
	1	0	1	1	120	1	119	1	120	1	121	1
	1	0	1	1	121	1	120	1	121	1	122	1
	1	0	1	1	122	1	121	1	122	1	123	1
	1	0	1	1	123	1	122	1	123	1	124	1
	1	0	1	1	124	1	123	1	124	1	125	1
	1	0	1	1	125	1	124	1	125	1	126	1
	1	0	1	1	126	1	125	1	126	1	127	1
	1	0	1	1	127	1	126	1	127	1	128	1
	1	0	1	1	128	1	127	1	128	1	129	1
	1	0	1	1	129	1	128	1	129	1	130	1
	1	0	1	1	130	1	129	1	130	1	131	1
	1	0	1	1	131	1	130	1	131	1	132	1
	1	0	1	1	132	1	131	1	132	1	133	1
	1	0	1	1	133	1	132	1	133	1	134	1
	1	0	1	1	134	1	133	1	134	1	135	1
	1	0	1	1	135	1	134	1	135	1	136	1
	1	0	1	1	136	1	135	1	136	1	137	1
	1	0	1	1	137	1	136	1	137	1	138	1
	1	0	1	1	138	1	137	1	138	1	139	1
	1	0	1	1	139	1	138	1	139	1	140	1
	1	0	1	1	140	1	139	1	140	1	141	1
	1	0	1									

## Z indikátor nuly

---

Při osmibitových aritmetických a logických operacích dojde k nastavení indikátoru ( $Z=1$ ), je-li výsledkem operace v registru A nula. Je-li výsledek nenulový, je  $Z=0$ . (pozor, plete se to)

Při instrukcích srovnávání bude indikátor nastaven ( $Z=1$ ) při shodě porovnávaných hodnot.

Při testování bitu bude se indikátor rovnat komplementu (t.j. opačné hodnotě) testovaného bitu.

Při instrukcích INI, INA a OUTD bude indikátor nastaven, bude-li obsah registru B roven 1.

Při instrukci IN r,(C) bude indikátor nastaven v případě nulového vstupu.

## S indikátor znaménka

---

Tento indikátor udává stav nejvýznamnějšího bitu registru ve kterém je uložen výsledek aritmetické operace. Při záporném výsledku je  $S=1$ .

Dále bude stav jednotlivých indikátorů označován následovně (u podmíněných instrukcí):

indikátor	I	S	Z	P/V	C
nenastaven (0)	I	P	NZ	PD	NC
nastaven (1)	I	M	Z	PE	C

Nyní se musíme detailně seznámit s jednotlivými strojními instrukcemi.

## Instrukce řídící

---

Slouží k nastavení pracovního režimu procesoru, povolení a zakázání maskovatelného přerušování, k nastavení CPU a k manipulaci se stavovým indikátorem CY.

NOP      žádná operace (No OPeration)  
 ===

Nenásleduje žádná činnost, procesor čeká na další instrukci. Nastavení indikátorů zůstává nezměněno.

HALT      zastavení  
 ====

Zastaví CPU a provádí operace NOP až do přijetí přerušování nebo vynulování CPU. Nastavení indikátorů zůstává nezměněno.

DI      zamaskování přerušování (Disables the mascable Interrupt)  
 ==

Instrukce znemožňuje přijetí zamaskovatelného přerušování až do jeho uvolnění instrukcí EI.  
 Nastavení indikátorů zůstává nezměněno.



EI uvolnění přerušení (Enable the mascale Interrupt)  
==

Instrukce uvolní přerušení zakázané instrukcí DI.  
Nastavení indikátorů zůstává nezměněno.

IM nastavení režimu přerušení (Interrupt Mode)  
==

Instrukcí IM je možno nastavit režim přerušení 0, 1 nebo 2, který se uvede do argumentu instrukce.  
Nastavení indikátorů se nezmění.

OCF inverze indikátoru C (CY) (Complement Carry Flag)  
===

Instrukce provede inverzi bitu C v registru F.

Indikátory S, Z a P/V nezměněny

N nulován

C invertován

H převezme předchozí nastavení C

SCF nastavení indikátoru C (CY) (Set Carry Flag)  
===

Instrukce nastaví indikátor C na hodnotu 1.

Indikátory S, Z a P/V nezměněny

N nulován

C nastaven

H nulován

## **Instrukce vstupu a výstupu (z periferních zařízení) =====**

IN instrukce vstupu (INput)  
==

Instrukce provádí přenos jednoho bajtu dat z periferního zařízení do registru Z 80. Jsou možné následující kombinace:

IN A, (n)	IN A, (C)	IN B, (C)
IN C, (C)	IN D, (C)	IN E, (C)
IN H, (C)	IN L, (C)	IN F, (C)

kde (C) znamená adresu periferního zařízení, která je uložena v registru C a (n) znamená adresu periferního zařízení. Instrukce IN F, (C) pouze nastavuje indikátory, ale nikam nezapisuje bajt ze vstupního zařízení.

Nastavení indikátorů je v případě IN A, (n) nezměněno, v případě IN r, (C) bude

H nulován

N nulován

C nezměněn,

S, Z a P/V nastaveny podle hodnot vstupujících dat.

INI instrukce blokového vstupu s inkrementací bez opakování  
 === (INput + Increment)

Instrukce provede přenos jednoho bajtu z periferního zařízení, jehož adresa je v registru C, na adresové místo, jehož adresa je v registru HL. Po skončení přenosu dekrementuje registr B a inkrementuje registr HL. Po skončení operace je stav registrů následující:

(HL) <- (C) ; B <- B-1 ; HL <- HL+1

Nastavení indikátorů:

S, H a P/V nedefinovány,  
 C nezměněn,  
 N nastaven  
 Z nastaven v případě, je-li B-1=0.

INIR instrukce blokového vstupu s inkrementací a opakováním  
 ==== (INput + Increment + Repeat)

Instrukce provádí totéž jako INI opakovaně tak dlouho, pokud je B<>0. Po dosažení stavu B=0 se pokračuje další instrukcí. U této instrukce se používá registr B jako počítadla: před instrukcí INIR musíme do registru B uložit počet bajtů, které chceme přenést.

Nastavení indikátorů je jako u INI.

IND instrukce blokového vstupu s dekrementací bez opakování  
 === (INput + Decrement)

Instrukce provádí přenos jednoho bajtu z periferního zařízení podobně jako instrukce INI s tím rozdílem, že po provedení přenosu dekrementuje jak registr B, tak registr HL. Po skončení operace je tudíž stav registrů následující:

(HL) <- (C) ; B <- B-1 ; HL <- HL-1

Nastavení indikátorů jako u instrukce INI.

INDR instrukce blokového vstupu s dekrementací a opakováním  
 ==== (INput + Decrement + Repeat)

Instrukce provádí přenos jednoho bajtu z periferního zařízení jako instrukce IND, ale opakovaně tak dlouho, pokud je B<>0. Po dosažení stavu B=0 se pokračuje další instrukcí. Registru B se opět používá jako počítadla.

Nastavení indikátorů jako u instrukce INI.

Instrukce vstupu mají jednu zvláštnost, a sice, že se při jejich provádění přivádí na adresovou sběrnici nejen obsah registru C, popřípadě hodnota  $n$  u instrukcí s  $A, (n)$  na dolních osm bitů, ale současně se na horních osm bitů přivádí obsah registru B, popřípadě u instrukce  $A, (n)$  obsah registru A, který je pak přepsán načtenou hodnotou. Instrukce by se vlastně měly zapisovat např.  $IN A, (BC)$  popřípadě  $IN A, (An)$ . Většina aplikací používá však jen dolní polovinu adresové sběrnice pro adresování periférií. Naš Sinclair ZX Spectrum je však výjimkou, protože využívá celou adresu při čtení klávesnice.

OUT instrukce výstupu (OUTput)  
===

Instrukce provádí přenos jednoho bajtu z registru Z 80, daného druhým operandem, do periferního zařízení daného prvním operandem. Jsou možné následující kombinace:

OUT (C),A	OUT (C),E
OUT (C),B	OUT (C),H
OUT (C),C	OUT (C),L
OUT (C),D	OUT (n),A

kde (C) znamená adresu periferního zařízení uloženou v registru C a (n) znamená adresu periferního zařízení.

Nastavení indikátorů nezměněno.

OUTI instrukce blokového výstupu s inkrementací bez  
==== opakování (OUTput + Increment)

Instrukce provede přenos jednoho bajtu z pamětového místa, jehož adresa je v registru HL, do periferního zařízení, jehož adresa je v registru C. Po skončení přenosu je dekrementován registr HL t.j. stav registrů po skončení operace je následující:

$(C) \leftarrow (HL) ; B \leftarrow B-1 ; HL \leftarrow HL+1$

Nastavení indikátorů jako u instrukce INI.

OTIR instrukce blokového výstupu s inkrementací a opakováním  
==== (OUTput + Increment + Repeat)

Provádí totéž jako instrukce OUTI opakovaně tak dlouho, pokud je  $B > 0$ . Po dosažení stavu  $B=0$  pokračuje další instrukcí. Registr B je opět použit jako počítadlo.

Nastavení indikátorů je jako u instrukce INI.

OUTD      instrukce blokového výstupu s dekrementací bez  
====      opakování (OuTput + Decrement)

Instrukce provádí totéž co instrukce OUTI s tím rozdílem, že po provedení přenosu dekrementuje jak registr HL, tak registr B, takže po skončení operace je stav registrů:

(C) <- (HL) ; B <- B-1 ; HL <- HL-1

Nastavení indikátorů jako u instrukce INI.

OTDR      instrukce blokového výstupu s dekrementací a opakováním  
====      (OuTput + Decrement + Repeat)

Instrukce provádí totéž jako instrukce OUTD opakovaně tak dlouho, pokud je B<>0. Po dosažení stavu B=0 pokračuje další instrukcí. Registru B je opět použito jako počítadla.

Nastavení indikátorů jako u instrukce INI.

Instrukce výstupu mají jednu zvláštnost, a sice, že se při jejich provádění přivádí na adresovou sběrnici nejen obsah registru C, popřípadě hodnota n u instrukce s A,(n) na dolních osm bitů, ale současně se na horních osm bitů přivádí obsah registru B, popřípadě u instrukcí A,(n) obsah registru A. Instrukce by se vlastně měly zapisovat např. OUT (BC),A popřípadě OUT (An),A. Většina aplikací používá však jen dolní polovinu adresové sběrnice pro adresování periférií.

Příklad na použití instrukcí vstupu a výstupu:

-----  
Přenesme blok 20 bajtů počínaje adresou #4082 do periferního zařízení o adrese #10.

```
LD B,20      ; nastavení počítadla
LD C,#10     ; nastavení adresy periferního zařízení do C
LD HL,#4000  ; nastavení adresy paměti od níž začne přenos
OTIR        ; provedení blokového přenosu
```

Stejný případ můžeme řešit použitím instrukce OUTI:

```
LD B,20      ; nastavení počítadla
LD C,#10     ; nastavení adresy periferního zařízení do C
LD HL,#4000  ; nastavení adresy paměti od níž začne přenos
op OUTI      ; přenos jednoho bajtu, inkř. HL, dekr. B
JR NZ,op     ; test indikátoru Z, je-li Z=0, t.j. B<>0 skok
              ; na návěští op, je-li Z=1, t.j. B=0,
              ; pokračování další instrukcí
```

Můžeme také použít pouhé instrukce OUT:

```

LD B,20      ; nastavení počítadla
LD C,#10     ; nastavení adresy periferního zařízení do C
LD HL,#4000  ; nastavení adresy paměti od níž začne přenos
op LD A,(HL)  ; přenos obsahu pamětového místa #4000 do A
OUT (C),A    ; přenos jednoho bajtu z reg. A do perif. zař.
INC HL       ; adresa dalšího pamětového místa
DJNZ op      ; dekrementuje počítadlo v B, testuje, je-li
              ; B=0 . Je-li B<>0 skok na návěští op, je-li
              ; B=0 pokračuje další instrukcí

```

Vidíme, že stejnou úlohu je možno řešit nejméně třemi i když různě složitými způsoby. Tyto příklady platí jen tehdy, když je pro dekódování periférie použito jen dolních osm bitů, jinak musí být jako počítadlo použit jiný registr.

## Instrukce přenosu dat

LD 1.,2. instrukce přenosu dat (Load)

=====

Instrukce provádí přenos jednoho nebo dvou bajtů z registru nebo pamětového místa daného operandem 2 do registru nebo pamětového místa daného operandem 1, případně čísla daného operandem do registru nebo pamětového místa daného operandem 1.

Instrukce LD můžeme rozdělit do několika skupin:

-----

- instrukce, které uloží do osmibitového registru jednobajtové číslo n. Jsou možné tyto kombinace:

LD A,n	LD B,n
LD C,n	LD D,n
LD E,n	LD H,n
LD L,n	

- instrukce, které uloží do dvojice registrů dvoubajtové číslo nn. Jsou možné tyto kombinace:

```

LD BC,nn
LD DE,nn
LD HL,nn

```

U této instrukce a u všech dalších instrukcí používajících dvoubajtových čísel jsou v operačním kódu uvedeny bajty v opačném pořadí než jak je píšeme v assembleru. Chceme-li do registru BC uložit číslo #4000, napíšeme instrukci v assembleru ve tvaru LD BC,#4000. Při převedení do strojového kódu je na prvním místě uveden kód instrukce, t.j. 01 a pak oba bajty v obráceném pořadí, instrukce ve strojovém kódu bude mít tvar:

01h 00h 40h !

- instrukce, které přenesou obsah jednoho registru do druhého registru (aniž se tím vymaže obsah registru, ze kterého se informace přenáší). V následující tabulce jsou sestaveny všechny možnosti.

LD A,A	LD A,B	LD A,C	LD A,D	LD A,E	LD A,H	LD A,L
LD B,A	LD B,B	LD B,C	LD B,D	LD B,E	LD B,H	LD B,L
LD C,A	LD C,B	LD C,C	LD C,D	LD C,E	LD C,H	LD C,L
LD D,A	LD D,B	LD D,C	LD D,D	LD D,E	LD D,H	LD D,L
LD E,A	LD E,B	LD E,C	LD E,D	LD E,E	LD E,H	LD E,L
LD H,A	LD H,B	LD H,C	LD H,D	LD H,E	LD H,H	LD H,L
LD L,A	LD L,B	LD L,C	LD L,D	LD L,E	LD L,H	LD L,L

- instrukce pro nepřímé adresování. Jsou to instrukce typu LD A,(nn) nebo LD A,(BC) a jejich opak, t.j. instrukce typu LD (nn),A nebo LD (BC),A. Instrukce typu LD A,(nn) přenesou do registru A obsah pamětového místa z adresy nn, instrukce LD A,(BC) přenesou do registru A obsah pamětového místa, jehož adresa je v registru BC, instrukce LD (nn),A naopak přenesou obsah registru A do pamětového místa daného adresou nn a instrukce LD (BC),A přenesou obsah registru A do pamětového místa, jehož adresa je v registru BC.

V následující tabulce jsou všechny možné kombinace

LD (nn),A	LD A,(nn)
LD (BC),A	LD A,(BC)
LD (DE),A	LD A,(DE)
LD (HL),A	LD A,(HL)
LD (HL),B	LD B,(HL)
LD (HL),C	LD C,(HL)
LD (HL),D	LD D,(HL)
LD (HL),E	LD E,(HL)
LD (HL),H	LD H,(HL)
LD (HL),L	LD L,(HL)

Opět platí, že v instrukci LD (nn),A a LD A,(nn) je adresa ve strojním kódu v opačném pořadí.

- instrukce pro nepřímé adresování, pracují se čtyřmi bajty. Jsou to instrukce typu LD BC,(nn), které uloží do registru C obsah pamětového místa na adrese nn a do registru B obsah pamětového místa na adrese nn+1 (opět se ukládání děje v opačném pořadí) a instrukce typu LD (nn),BC, které uloží obsah registru C do pamětového místa na adrese nn a obsah registru B do pamětového místa na adrese nn+1. Jsou možné tyto kombinace:

LD BC,(nn)	LD (nn),BC
LD DE,(nn)	LD (nn),DE
LD HL,(nn)	LD (nn),HL

V strojním kódu se udává pouze adresa prvního pamětového místa (jako obvykle v opačném pořadí), instrukce sama provede přechod na následující adresu.

- instrukce pro přenos dat z a do registru SP, které dovolují manipulaci s uživatelským zásobníkem (ale pozor, tyto instrukce si smí dovolit jen ten, kdo skutečně ovládá perfektně mechaniku uživatelského zásobníku, poněvadž jejich nevhodné použití znamená zhroucení systému).

Existují následující možnosti:

LD SP,nn	LD SP,(nn)
LD SP,HL	LD (nn),SP
LD SP,(IX+d)	LD SP,(IY+d)

Instrukce LD SP,nn uloží do registru SP dvoubajtové číslo nn. Instrukce LD SP,(nn) přenesou obsah pamětového místa na adrese nn do nižšího bajtu registru SP a obsah adresového místa na adrese nn+1 do vyššího bajtu registru SP (SP je šestnácti-bitový registr). Instrukce LD (nn),SP přenesou obsah nižšího bajtu registru SP do pamětového místa na adrese nn a obsah vyššího bajtu registru SP do pamětového místa na adrese nn+1. Instrukce LD SP,HL přenesou obsah dvojice registrů HL do registru SP.

- Instrukce pro práci s registry IX, IY a pro indexové adresování.

Existují následující kombinace:

LD A,(IX+d)	LD (IX+d),A
LD B,(IX+d)	LD (IX+d),B
LD C,(IX+d)	LD (IX+d),C
LD D,(IX+d)	LD (IX+d),D
LD E,(IX+d)	LD (IX+d),E
LD H,(IX+d)	LD (IX+d),H
LD L,(IX+d)	LD (IX+d),L
	LD -(IX+d),n
LD IX,(nn)	LD (nn),IX
LD IX,nn	LD SP,IX
LD A,(IY+d)	LD (IY+d),A
LD B,(IY+d)	LD (IY+d),B
LD C,(IY+d)	LD (IY+d),C
LD D,(IY+d)	LD (IY+d),D
LD E,(IY+d)	LD (IY+d),E
LD H,(IY+d)	LD (IY+d),H
LD L,(IY+d)	LD (IY+d),L
	LD (IY+d),n
LD IY,(nn)	LD (nn),IY
LD IY,nn	LD SP,IY

Tyto instrukce se používají pro tzv. indexové adresování. Instrukce LD A,(IX+d) přenesou do registru A obsah pamětového místa, jehož adresou je hodnota v registru IX, zvětšená o hodnotu indexu d. Podobně instrukce LD (IX+d),A přenesou obsah registru A do pamětového místa, jehož adresa je obsah registru IX zvětšený o hodnotu indexu d. Index d může nabývat krajních hodnot od 80h (=-128d) do 7Fh (="+127d), t.j. používá se

dvojkového doplňku. Je-li v registru IX např. adresa 4000h, a index má hodnotu 20h pak se instrukcí LD A,(IX+#20) přičte hodnota indexu k adrese a přenesse do registru A obsah pamětového místa s adresou #4020. Instrukce LD A,(IX+#FA) odečte od adresy #4000, uložené v registru IX, dvojkový doplněk FAh, t.j. 6 a přenesse do registru A obsah pamětového místa na adrese #2FEA ( $\#4000 - \#0006 = \#2FEA$ ).

Pozor při strojním kódu! Kód indexu d je vždy na třetím místě, i v případě instrukce LD (IX+d),n, kde jednobajtové číslo n je až na posledním místě.

- instrukce pro práci s registry I a R. Instrukce umožňují naplnit registr I nebo přenést obsah registru I (registr stránkové adresy přerušení) tj. vyšších osm bitů adresy přerušení do registru A. Další instrukce umožňují naplnit registr R nebo přenést obsah registru R (refresh) do registru A.

Možné kombinace:

LD A,I	LD A,R
LD I,A	LD R,A

Instrukce typu LD nemění indikátory. Vyjimkou jsou pouze instrukce LD A,I a LD A,R.

Nastavení indikátorů po instrukci LD A,I nebo LD A,R

S	nastaven je-li obsah registru záporný, jinak nulován
Z	nastaven je-li obsah registru nula, jinak nulován
H	nulován
P/V	naplněn stavem klopného obvodu přerušení
	nastaven je-li přerušení povoleno, jinak nulován
N	nulován
C	nezměněn

V příloze je uvedena přehledná tabulka všech možných kombinací instrukce LD.

PUSH instrukce vkládání do uživatelského zásobníku

====

Instrukce PUSH přenesse obsah dvojice registrů do uživatelského zásobníku. Děje se to tím způsobem, že nejprve se dekrementuje registr SP, na adresu jím určenou se na vrchol zásobníku přenesse obsah vyššího registru z dané dvojice (t.j. registru A, B, D, H nebo vyšších 8 bitů registru IX nebo IY), opět se dekrementuje registr SP a na dekrementovaný vrchol zásobníku se přenesse obsah nižšího z obou registrů (t.j. registru F, C, E, L nebo nižších osm bitů registru IX nebo IY - registry IX a IY jsou šestnácti-bitové. Možné kombinace instrukce PUSH:

PUSH AF	PUSH BC
PUSH DE	PUSH HL
PUSH IX	PUSH IY

Nastavení indikátorů zůstává nezměněno.



POP instrukce vybírání ze zásobníku  
==

Instrukce POP přenese obsah dvou pamětových míst z vrcholu zásobníku do dvojice registrů. Nejprve přenese obsah vrcholu zásobníku do nižšího z dvojic registrů nebo do nižších osmi bitů registru IX nebo IY, pak inkrementuje registr SP a přenese obsah inkrementovaného vrcholu zásobníku do vyššího z dané dvojice nebo do vyšších osmi bitů registru IX nebo IY. Tato instrukce je opakem instrukce PUSH, tj. informace vložené do zásobníku instrukcí PUSH získáme zpět instrukcí POP.

Možné kombinace instrukce POP:

POP AF	POP BC
POP DE	POP HL
POP IX	POP IY

Nastavení indikátorů se změní po instrukci POP AF, u dalších zůstává nastavení indikátorů nezměněno.

EX instrukce výměny obsahů registrů  
== (EXchange)

Instrukce EX provádí výměnu obsahů dvou registrových párů. Jsou možné tyto kombinace:

EX AF,AF'  
EX DE,HL  
EX (SP),HL  
EX (SP),IX  
EX (SP),IY

Instrukce EX (SP),HL, EX (SP),IX a EX (SP),IY provádí výměnu obsahů uvedených registrových párů a obsahu uživatelského zásobníku (adresa vrcholu zásobníku je v SP) způsobem popsaným u instrukcí PUSH a POP.

Nastavení indikátorů zůstává nezměněno.

EXX instrukce výměny obsahu hlavních a čárkovaných registrů  
==

Instrukce EXX provede současnou výměnu obsahu dvojic registrů BC - BC', DE - DE', HL - HL'.

Nastavení indikátorů zůstává nezměněno.

Příklady na použití instrukcí přenosu dat.  
-----

Příklady na použití instrukcí LD byly již vlastně v předchozích odstavcích, kdy pro vložení hodnot do registru bylo použito instrukci LD, např. LD B,#2C pro vložení hodnoty #2C do registru B, nebo LD HL,#4C82 pro vložení adresy začátku přenosu dat do registru HL. To byly ty nejjednodušší případy.

Chceme-li např. uložit do registru BC adresu začátku registru obrazovky, která je v paměti od adresy 4000h musíme použít instrukci pro vložení hodnoty do registru LD BC,#4000. V strojním kódu to bude 01 00 40. 01 je kód instrukce LD BC,nn a 00 40 je v opačném pořadí napsána adresa #4000 (t.j. to nn v instrukci). Při práci se strojními programy potřebujeme také vložit na určitou adresu novou hodnotu. Děje se tak např. při vkládání nové hodnoty do systémové proměnné. Toto se však nedá přímo, poněvadž neexistuje instrukce LD (nn),nn, ale musíme použít okliku přes některý registrový pár. Mohli bychom použít kterýkoliv ze tří registrových párů BC, DE, nebo HL, poněvadž všechny tři páry mají instrukci LD rr,nn a současně LD (nn),rr. Pro registrové páry BC a DE mají obě instrukce po čtyřech bajtech, zatímco pro registrový pár HL mají obě instrukce po třech bajtech. Je tudíž vhodnější použít registrového páru HL, čímž se ušetří celkem 2 bajty. Bude pak:

```
LD HL,32600      ; nová adresa syst. proměnné UGD do reg. HL
LD (23675),HL    ; nová adresa do systémové proměnné UGD
```

Jiný případ bude, pokud potřebujeme přenést strojní program vložený do instrukce REM na začátku basicové oblasti, za RAMTOP. Máme strojní program o délce 16 bajtů, který chceme přenést z adresy 23760 na novou adresu 65000. Jelikož neexistuje instrukce LD (nn),nn, musíme použít okliku přes některý registr, v tomto případě registr A, který jako jediný umožňuje přenosy. Postupujeme tak, že přeneseme obsah prvního pamětového místa do registru A, odtud ho další instrukcí přeneseme na novou adresu, zvýšíme o jedničku jak počáteční tak novou adresu a celý postup opakujeme šestnáctkrát. K odpočítání potřebného počtu opakovaných přenosů použijeme opět registr B jako počítadla spolu s instrukcí DJNZ, která bude vysvětlena později.

```
LD DE,23760      ; nastavení počátku přenosu
LD HL,65000      ; nastavení nové počáteční adresy
LD B,#10         ; nastavení počítadla
znovu LD A,(DE)   ; přenos obsahu první adresy do reg. A
LD (HL),A        ; přenos obsahu reg. A do první nové
                  ; adresy
INC DE           ; další adresa
INC HL           ; další nová adresa
DJNZ znovu       ; test, je-li B<>0, není provedeno 16
                  ; přenosů, skok na návěští znovu, t.j.
                  ; opakování přenosu. Je-li B=0, je přenos
                  ; ukončen a pokračuje se další instrukcí
```

Tento příklad lze daleko jednoduššeji řešit pomocí instrukcí blokového přenosu, které budou uvedeny v následující kapitole. Před spuštěním strojního programu si musíme vždy uvědomit, že v registrech procesoru jsou informace, které procesor potřebuje pro svoji další práci po ukončení strojního programu. Přepíšeme-li tyto informace nově vloženými hodnotami, může se stát, že se program po návratu ze strojního kódu zhroutí. Aby k tomu nedošlo musíme uchovat pro další použití obsahy registrů. Používáme obvykle instrukci PUSH a POP. Chceme-li uchovat a později do registru HL vrátit jeho obsah, použijeme PUSH HL, pak může pokračovat strojní kód a po jeho ukončení musí následovat POP HL

Obou instrukcí můžeme také použít pro přenos obsahu registrů. Chceme-li přenést obsah registrového páru BC do registrového páru HL, můžeme použít:

```
PUSH BC    ; přenese obsah BC na vrchol zásobníku
POP HL     ; přenese obsah vrcholu zásobníku, t.j. původní
           ; obsah BC, do HL
```

## Instrukce blokového přenosu dat

```
LDI        instrukce blokového přenosu s inkrementací bez
===        opakování (LoaD + Increment)
```

Instrukce provádí přenos jednoho bajtu z adresy, která se nachází v registrovém páru HL na adresu, která se nachází v registrovém páru DE. Instrukce současně inkrementuje HL a DE a dekrementuje BC, kterého se používá jako počítadla.

Stav po provedení instrukce je následující :

(DE) <- (HL) ; DE <- DE+1 ; HL <- HL+1 ; BC <- BC-1

Nastavení indikátorů:

S, Z a C nezměněn,  
H a N nulován.  
P/V nastaven, je-li BC-1 <> 0

```
LDIR       instrukce blokového přenosu s inkrementací a opakováním
===        (LoaD + Increment + Repeat)
```

Instrukce provádí činnost jako instrukce LDI, ale opakovaně pokud je BC-1 <> 0. Po dosažení stavu BC-1=0 se pokračuje v provádění následující instrukcí.

Nastavení indikátorů jako u instrukce LDI (po skončení je P/V nulován, poněvadž je BC-1=0).

```
LDD        instrukce blokového přenosu s dekrementací bez
===        opakování (LoaD + Decrement)
```

Instrukce provádí obdobnou činnost jako instrukce LDI, ale místo inkrementování dekrementuje oba registrové páry DE a HL a současně dekrementuje registrový pár BC.

Stav po provedení instrukce:

(DE) <- (HL) ; DE <- DE-1 ; HL <- HL-1 ; BC <- BC-1

Nastavení indikátorů jako u LDIR.

LDDR instrukce blokového přenosu s dekrementací  
==== (Load + Decrement + Repeat)

Instrukce provádí činnost jako instrukce LDD, ale opakovaně, pokud je BC-1  $\neq$  0. Po dosažení stavu BC-1=0 se pokračuje další instrukcí.

Nastavení indikátorů jako u LDIR.

Příklady použití blokového přenosu:

-----  
Příklad z předchozí kapitoly, t.j. přenos 16 bajtů, počínajících adresou #4000 na novou adresu 32768 vyřešíme pomocí blokového přenosu velmi jednoduše:

```
LD BC,#10           ; nastavení počítadla
LD HL,#4000         ; nastavení první adresy přenosu
LD DE,#8000         ; nastavení první adresy nové lokace
LDIR                ; blokový přenos až do stavu BC-1=0
```

## ----- Instrukce aritmetických a logických operací -----

ADD instrukce aritmetického sčítání  
===

Tato instrukce provádí součet dvou operandů a výsledek uloží do prvního operandu. Může pracovat s jednobajtovými i dvoubajtovými čísly.

Možné kombinace:

ADD A, (HL)	ADD A, A
ADD A, (IX+d)	ADD A, B
ADD A, (IY+d)	ADD A, C
ADD HL, BC	ADD A, D
ADD HL, DE	ADD A, E
ADD HL, HL	ADD A, H
ADD HL, SP	ADD A, L
	ADD A, n
ADD IX, BC	ADD IY, BC
ADD IX, DE	ADD IY, DE
ADD IX, IX	ADD IY, IY
ADD IX, SP	ADD IY, SP

Nastavení indikátorů:

8-bitová operace	16-bitová operace
S nastaven, je-li výsledek záporný	nezměněn
Z nastaven, je-li výsledek nulový	nezměněn
H nastaven, je-li přenos ze 3. bitu	nastaven, je-li přenos z 11. bitu
P/V nastaven, dojde-li k přetečení	nezměněn
N nulován	nezměněn
C nastaven, dojde-li k přenosu ze znaménkového bitu	

ADC instrukce aritmetického sčítání s přenosem  
 === (ADD with Carry)

Instrukce provádí sčítání dvou operandů jako instrukce ADD a navíc přičte obsah indikátoru CY. Výsledek uloží do prvního operandu.

Možné kombinace:

ADC A, (HL)	ADC A, A
ADC A, (IX+d)	ADC A, B
ADC A, (IY+d)	ADC A, C
ADC HL, BC	ADC A, D
ADC HL, DE	ADC A, E
ADC HL, HL	ADC A, H
ADC HL, SP	ADC A, L
	ADC A, n

Nastavení indikátorů:

Stejně jako u instrukce ADD u osmibitové operace. H je nastaven, dojde-li u šestnáctibitové operace k přenosu mezi 11 a 12 bitem.

SUB instrukce aritmetického odčítání  
 === (SUBtract)

Instrukce odečte operand od obsahu registru A a výsledek uloží do registru A.

Možné kombinace:

SUB A	SUB H
SUB B	SUB L
SUB C	SUB n
SUB d	SUB (HL)
SUB E	SUB (IX+d)
	SUB (IY+d)

Nastavení indikátorů:

S nastaven, je-li výsledek záporný  
 Z nastaven, je-li výsledek nula  
 H nastaven při přenosu mezi 3. a 4. bitem  
 P/V nastaven dojde-li k přetečení  
 N nastaven  
 C nastaven, dojde-li k přenosu ze znaménkového bitu.

SBC instrukce aritmetického odčítání s přenosem  
 === (SUBtract with Carry)

Instrukce odečte obsah druhého operandu a obsahu indikátoru C od prvního operandu a výsledek uloží do prvního operandu.

Možné kombinace:

SBC A, (HL)	SBC A, A
SBC A, (IX+d)	SBC A, B
SBC A, (IY+d)	SBC A, C
SBC HL, BC	SBC A, D
SBC HL, DE	SBC A, E
SBC HL, HL	SBC A, H
SBC HL, SP	SBC A, L
	SBC A, n

Nastavení indikátorů:

Stejně jako u instrukce SUB, navíc u šestnáctibitových operací bude H nastaven, dojde-li k přenosu z 11. do 12. bitu.

DAA instrukce opravy pro BCD soustavu  
 === (Decimal Adjust Accumulator)

Instrukce opravuje výsledek aritmetických operací ADD, ADC, SUB, SBC a dále výsledky instrukcí INC, DEC a NEG, týkající se registru A tak, aby odpovídaly výsledkům daných operací prováděných v BCD soustavě. Z 80 pracuje v binární soustavě a k jejímu vyjádření používáme HEX soustavu. BCD soustava pracuje pouze s číslicemi 0 až 9, HEX číslice A až F nezná. Pracujeme-li s čísly v soustavě BCD, musíme použít instrukci DAA, aby nám hodnoty A až F převedla na BCD vyjádření. Nejlépe je to zřejmé z příkladu. Máme v BCD (t.j. vlastně v desítkové soustavě) provést tři součty:

$28 + 11 = 39$  ,  $28 + 13 = 41$  a  $28 + 19 = 47$

BCD soustava převádí každou desítkovou číslici na 4 bity. 28d převede na 0010 1000b, 11d na 0001 0001b, 13d na 0001 0011b a 19d na 0001 1001b.

Registr A provede uvedené součty následovně:

28+11:      0010 1000b    28d  
           ---+0001 0001b\_\_\_11d  
           0011 1001b = 39d. Oprava není třeba

28+13:      0010 1000b    28d  
           ---+0001 0011b\_\_\_13d  
           0011 1011b =        nelze vyjádřit v BCD, instrukce DAA  
 musí provést opravu, aby výsledkem bylo

0100 0001b    41d

Provede to tak, že k výsledku binárního součtu přičte 6.

Bude pak 0011 1011b    3Bd  
           ---+0000 0110b\_\_\_06d  
           0100 0001b = 41d

```

28+19:    0010 1000b    28d
          ---+0001 1001b---19d
          0100 0001b = 41d

```

Výsledek je sice vyjádřitelný v BCD soustavě, ale nesprávný, poněvadž došlo k přenosu mezi 3. a 4. bitem, což v BCD neplatí. Instrukce DAA opět zjedná nápravu přičtením 6:

```

          0100 0001b    41d
          ---+0000 0110b---06d
          0100 0111b = 47d

```

U odečítání se 6 od výsledku operace odečítá. Důležité je to, že mezi aritmetickou operací a instrukcí DAA nesmí být instrukce, jejíž provedení ovlivní nastavení indikátorů N a H.

```

CPL      instrukce komplementu
===      (ComPLement)

```

Instrukce v registru A nahradí jedničky nulami a naopak.

Nastavení indikátorů zůstává nezměněno.

```

NEG      instrukce negace
===

```

Instrukce změní znaménko čísla v registru A (odečte obsah registru A od nuly).

Nastavení indikátorů:

```

S  nastaven, je-li výsledek záporný
Z  nastaven, je-li výsledek nula
H  nastaven, došlo-li k přenosu mezi 3. a 4. bitem
P/V nastaven, byl-li obsah registru A před operací 80h
N  nastaven
C  nastaven, nebyl-li před operací obsah registru A
    roven nule.

```

Příklady použití aritmetických operací:

Potřebujeme najít adresu začátku prvního řádku Basicu. Víme, že tuto adresu dostaneme přičtením hodnoty 5 k adrese uložené v systémové proměnné PROG, která je na adrese 23635.

Použijeme :

```

LD HL,(23635) ; adresa začátku Basicového programu
               ; v paměti do HL
LD BC,0005    ; přičítána hodnota
ADD HL,BC     ; sečtení obou hodnot a uložení do HL

```

Potřebujeme sečíst dvě 32-bitová čísla. K dispozici jsou však pouze 16-ti bitové registry. Musíme tudíž nejprve sečíst nižších 16 bitů obou čísel a pak vyšších 16 bitů obou čísel.

Při sečtení nižších 16 bitů může ale dojít k přetečení (t.j. výsledek je větší než je možno vyjádřit 16-ti bitovým registrem). V tom případě dojde k nastavení indikátoru C. Použijeme k tomu instrukce ADC. Zvolíme-li dvě konkrétní čísla například 23456789h a C12BA238h, bude pak:

```
LD HL,#6789      ; nižších 16 bitů 1.operandu do HL
LD BC,#A238      ; nižších 16 bitů 1.operandu do BC
ADD HL,BC        ; součet 16-ti nižších bitů
PUSH HL          ; obsah HL do zásobníku
LD HL,#2345      ; vyšších 16 bitů 1.operandu do HL
LD BC,#C12B      ; vyšších 16 bitů 2.operandu do BC
ADC HL,BC        ; součet 16-ti vyšších bitů s přičtením pře-
                  ; nosového bitu z předchozího součtu
LD DE,HL         ; výsledek druhého součtu do DE
POP BC           ; výsledek prvního součtu ze zásobníku do BC
```

Nižších 16 bitů výsledku bude v registrovém páru BC a vyšších 16 bitů výsledku v DE.

Instrukce DAA byla již dostatečně vysvětlena v textu, jen zápis v assembleru při součtu 28 + 13:

```
LD A,28          ; první operand do reg. A
ADD A,13         ; přičtení druhého operandu
DAA              ; převede výsledek do BCD soustavy
```

Máme-li v registru A hodnotu 01010101b, bude v něm po instrukci CPL 10101010b.

Je-li v registru A hodnota 000000001b /=1d/, bude v něm po provedení instrukce NEG 11111111b /=-1d/.

AND        instrukce logického součinu  
===

Instrukce provede logický součin odpovídajících si bitů registru A a operandu. Výsledek uloží do registru A.

Možné kombinace:

AND A	AND H
AND B	AND L
AND C	AND n
AND D	AND (HL)
AND E	AND (IX+d)
	AND (IY+d)

Nastavení indikátorů:

S	nastaven, je-li výsledek záporný, jinak nulován
Z	nastaven, je-li výsledkem nula, jinak nulován
H	nastaven
P/V	nastaven v případě sudé parity, jinak nulován
N	nulován
C	nulován



OR instrukce logického součtu  
==

Instrukce provede logický součet odpovídajících si bitů registru A a operandu. Výsledek uloží do registru A.

Možné kombinace:

OR A	OR E	OR (HL)
OR B	OR H	OR (IX+d)
OR C	OR L	OR (IY+d)
OR D	OR n	

Nastavení indikátorů:

S nastaven, je-li výsledek záporný, jinak nulován  
Z nastaven, je-li výsledkem nula, jinak nulován  
H nastaven  
P/V nastaven v případě sudé parity, jinak nulován  
N nulován  
C nulován

XOR instrukce binárního /exkluzivního/ součtu  
==

Instrukce provede binární součet odpovídajících si bitů registru A a operandu. Výsledek uloží do registru A.

Možné kombinace:

XOR A	XOR E	XOR (HL)
XOR B	XOR H	XOR (IX+d)
XOR C	XOR L	XOR (IY+d)
XOR D	XOR n	

Nastavení indikátorů:

S nastaven, je-li výsledek záporný, jinak nulován  
Z nastaven, je-li výsledkem nula, jinak nulován  
H nastaven  
P/V nastaven v případě sudé parity, jinak nulován  
N nulován  
C nulován

CP instrukce srovnávání  
== (ComPare)

Instrukce provede odečtení operandu od obsahu registru A. Výsledek nikam neukládá, pouze nastavuje indikátory.

Možné kombinace:

CP A	CP E	CP (HL)
CP B	CP H	CP (IX+d)
CP C	CP L	CP (IY+d)
CP D	CP n	

Nastavení indikátorů:

S nastaven při záporném výsledku, jinak nulován  
Z nastaven, je-li výsledek nulový, (je-li A=operand)  
H nastaven při přenosu ze 3. do 4. bitu  
P/V nastaven došlo-li k přetečení  
N nastaven  
C nastaven při přenosu ze znaménkového bitu

### Příklady použití logických operací:

Chceme-li vynulovat vyšší čtyři bity registru A a nižší čtyři bity ponechat, provedeme

AND #0F ; nuluje 4 vyšší bity reg. A, nižší 4 bity ponechá

Pro lepší představu jak to funguje:

```
obsah reg.A      0101 0101
AND #0F          ----0000 1111----
výsledek         0000 0101
```

Chceme-li vynulovat registr A, je XOR A lepší než LD A,00 (XOR má operační kód 1 bajt, LD A,00 však 2 bajty):

```
obsah reg.A      0101 0101
XOR A            ----0101 0101----
výsledek         0000 0000
```

Instrukci CP používáme při větvení programu, příp. při tvoření smyček. Máme provést skok na návěští "vpravo", bude-li v registru A hodnota 20, nebo skok na návěští "vlevo", bude-li v registru A hodnota 40:

```
CP 20           ; porovnání obsahu A s hodnotou 20
JR Z,vpravo     ; je-li A=20, t.j. Z=1, skoč na "vpravo"
CP 40           ; porovnání A s hodnotou 40
JR Z,vlevo      ; je-li A=40, t.j. Z=1, skoč na "vlevo"
```

### Instr. blokového prohledávání

CPI instrukce blokového prohledávání s inkrementací bez  
== opakování (ComPare + Increment)

Instrukce provede odečtení obsahu pamětového místa, jehož adresa je v registrovém páru HL od obsahu registru A. Současně inkrementuje registrový pár HL a dekrementuje registrový pár BC (kterého se používá jako počítadla).

Stav po provedení instrukce:

A - (HL) ; HL <- HL + 1 ; BC <- BC - 1

Výsledek operace se neukládá do žádného z registrů, pouze dochází k nastavení indikátorů, které je následující:

```
S nastaven, je-li výsledek záporný
Z nastaven, je-li výsledek nula, t.j. je-li A = (HL)
H nastaven, došlo-li k přenosu mezi 3. a 4. bitem
P/V nastaven, je-li BC - 1 <> 0
N nastaven
C nezměněn
```

CPIR instrukce blokového prohledávání s inkrementací a  
==== opakováním (ComPare + Increment + Repeat)

Instrukce provádí stejnou činnost jako instrukce CPI, ale opakovaně pokud je  $BC-1 \neq 0$  nebo pokud je  $A-(HL) \neq 0$ . Pokud nastane některý z případů  $BC-1=0$  nebo  $A-(HL)=0$ , pokračuje se prováděním další instrukce.

Nastavení indikátorů stejně jako u instrukce CPI.

CPD instrukce blokového prohledávání s dekrementací bez  
=== opakování (ComPare + Decrement)

Instrukce provádí obdobnou činnost jako instrukce CPI, jenom s tím rozdílem, že dekrementuje jak registrový pár HL, tak registrový pár BC. Stav po provedení instrukce:

$A = (HL) ; HL \leftarrow HL - 1 ; BC \leftarrow BC - 1$

Nastavení indikátorů stejně jako u instrukce CPI.

CPDR instrukce blokového prohledávání s dekrementací a  
==== opakováním (ComPare + Decrement + Repeat)

Instrukce provádí stejnou činnost jako instrukce CPD, ale opakovaně pokud je  $BC-1 \neq 0$  nebo pokud je  $A-(HL) \neq 0$ . Pokud nastane jeden z případů  $BC-1=0$  nebo  $A-(HL)=0$ , opakování se ukončí a pokračuje se prováděním následující instrukce.

Nastavení indikátorů je stejné jako u instrukce CPI.

Příklady použití instrukcí blokového prohledávání:

Máme v paměťovém bloku 100 bajtů, končícím adresou 7FFFh, najít první nulový bajt odshora. Použijeme-li instrukce CPDR, bude:

```
LD HL,#7FFF ; nastavení počátku prohledávání
LD BC,0100 ; nastavení počítadla
LD A,0 ; nastavení hledané (porovnávané) hodnoty
CPDR ; blokové prohledávání až do stavu (HL)=A nebo
; BC-1=0
```

## Instrukce inkrementace a dekrementace

---

INC      instrukce inkrementace

===

Instrukce provede zvýšení hodnoty operandu o jedničku.

Možné kombinace :

INC A	INC BC	INC (HL)
INC B	INC DE	INC (IX+d)
INC C	INC HL	INC (IY+d)
INC D	INC IX	
INC E	INC IY	
INC H	INC SP	
INC L		

Nastavení indikátorů:

8-bitová operace

16-bitová operace

S nastaven, je-li výsledek záporný	nezměněn
Z nastaven, je-li výsledek nulový	nezměněn
H nastaven při přenosu mezi 3.a 4.bitem	nezměněn
P/V nastaven při přetečení	nezměněn
N nulován	nezměněn
C nezměněn	nezměněn

DEC      instrukce dekrementace

===

Instrukce provede snížení hodnoty operandu o jedničku.

Možné kombinace:

DEC A	DEC BC	DEC (HL)
DEC B	DEC DE	DEC (IX+d)
DEC C	DEC HL	DEC (IY+d)
DEC D	DEC IX	
DEC E	DEC IY	
DEC H	DEC SP	
DEC L		

Nastavení indikátorů:

8-bitová operace

16-bitová operace

S nastaven, je-li výsledek záporný	nezměněn
Z nastaven, je-li výsledek nulový	nezměněn
H nastaven při přenosu mezi 3.a 4.bitem	nezměněn
P/V nastaven při přetečení	nezměněn
N nastaven	nezměněn
C nezměněn	nezměněn

Příklad:

-----

Máme zvýšit adresu uloženou v registrovém páru HL o 2:

```
INC HL      ; zvýšení o 1
INC HL      ; zvýšení o 1
```

## Instrukce skoků

JP instrukce nepodmíněného skoku (Jump)

==

Instrukce provede skok na paměťové místo, jehož adresa je dána operandem (tj. uloží hodnotu danou operandem do registru PC).

Možné kombinace:

JP nn	JP (IX)
JP (HL)	JP (IY)

POZOR ! změna dohodnutého způsobu označování: U instrukcí JP neznamena (HL) obsah paměťového místa, jehož adresa je v registrovém páru HL, ale přímo adresu skoku, která je uložena v registrovém páru HL (podobně to platí u JP (IX) a JP (IY)).

Nastavení indikátorů zůstává nezměněno.

JP 1.,2. instrukce podmíněného skoku (Jump)

=====

Instrukce provede skok na adresu danou druhým operandem pouze tehdy, je-li splněna podmínka daná prvním operandem. Touto podmínkou v prvním operandu je stav indikátoru S, Z, P/V nebo C. Není-li splněna, provádí se další instrukce.

instrukce	skok při stavu indikátoru
JP M,nn	S nastaven /výsledek <0/
JP P,nn	S nulován /výsledek >=0/
JP Z,nn	Z nastaven /výsledek =0/
JP NZ,nn	Z nulován /výsledek <>0
JP PE,nn	P/V nastaven /sudá parita/
JP PO,nn	P/V nulován /lichá parita/
JP C,nn	C nastaven /přenos/
JP NC,nn	C nulován /bez přenosu/

Nastavení indikátorů zůstává nezměněno.

JR n instrukce relativního skoku  
==== (Jump Relative)

Instrukce provede relativní skok, jehož velikost je určena operandem. Adresu skoku získáme přičtením hodnoty operandu k adrese instrukce relativního skoku zvětšené o 2. Operand může nabývat hodnot od 80h (=-128d) do 7Fh (=127d).

U dokonalejších překladačů (Assemblerů) je možno zadat jako operand návěští a překladač si velikost odskoku vypočte sám (např. JR ZNOVU).

Nastavení indikátorů zůstává nezměněno.

JR 1.,n instrukce relativního podmíněného skoku  
 ===== (Jump Relative)

Instrukce provede relativní skok, jehož velikost je určena druhým operandem jen tehdy, je-li splněna podmínka daná prvním operandem. Není-li tato podmínka splněna, pokračuje se další instrukcí. Adresu skoku získáme přičtením hodnoty operandu k adrese instrukce relativního skoku zvětšené o 2. Operand může nabývat hodnot od 80h (= -128d) do 7Fh (= 127d). Podmínkou v prvním operandu je nastavení indikátoru Z nebo C.

Možné kombinace:

instrukce	skok při stavu indikátoru
JR Z,n	Z nastaven (výsledek nulový)
JR NZ,n	Z nulován (výsledek nenulový)
JR C,n	C nastaven (přenos)
JR NC,n	C nulován (bez přenosu)

Nastavení indikátorů zůstává nezměněno.

Některé překladače si velikost operandu "n" vypočítávají samy.

DJNZ n instrukce relativního skoku podmíněného čítačem  
 ===== (Decrement + Jump No Zero)

Instrukce provádí opakovaně dekrementaci registru B (kterého se používá jako počítadla) a v případě, že B-1 < 0 provede relativní skok na adresu danou operandem e. Dosáhne-li se stavu B-1 = 0, pokračuje se prováděním následující instrukce. Pro adresu odskoku a velikost operandu n platí totéž jako u instrukce JR n. Některé překladače si velikost odskoku vypočtou samy.

Nastavení indikátorů zůstává nezměněno.

Příklady použití instrukcí skoku:

Máme ze strojního programu skočit na podprogram v ROM, který je na adrese #0018. Postačí

JP #0018 ; skok na adresu #0018

Chceme-li upravit program uvedený v kapitole blokového prohledávání tak, aby při nalezení bajtu s obsahem FFh skočil do ROM na adresu 0020h:

```
LD HL,#7FFF ; počáteční adresa do HL
LD BC,0100 ; nastavení počítadla
LD A,#FF ; nastavení hledané hodnoty
CPDR ; blokové prohledávání
JP Z,#0020 ; je-li Z=1 (což nastane při A=FFh),
; skok na #0020
```

Adresy absolutních skoků jsou pevné a nemění se při posunu strojního programu v paměti. Používáme je většinou při skocích do ROM, t.j. na neměnné adresy, poněvadž při použití ke skokům na adresy strojního programu musíme při posunech strojního programu v paměti provést přeadresování.

Naproti tomu relativní skoky nemají pevnou adresu, ale mají určen pouze interval (počet bajtů) od adresy instrukce skoku zvětšené o 2. Při posunech strojního programu v paměti se posunují i adresy relativních skoků, takže není třeba relativní skoky přeadresovávat. Navíc zaberou v programu o bajt méně než skoky absolutní.

Velikost odskoku je omezena hodnotou  $-128d=80h$  a  $127d=7Fh$  bajtu (nikoliv instrukcí!) a obvykle ho musíme stanovit až po převedení assembleru do relativního strojního programu podle skutečného počtu přeskakovaných bajtů. Postupujeme podle schématu uvedeného v následující tabulce. Pokud potřebujeme realizovat větší odskok než  $-128d$  nebo než  $+127d$ , musíme zkombinovat dva nebo i více relativních skoků tak, že do adresy na kterou se provede skok umístíme další relativní skok.

op- kód	návěští	mnemotech. zkratka	hodnota n	
21		LD HL, NN	F4h	
FF			F5h	
7F			F6h	
01		LD BC, NN	F7h	
01			F8h	
01			F9h	
3E	ZNOVU	LD A, N	FAh	
FF			FBh	
ED		CPDR	FC h	
B9			FDh	
18		JR DIS	FEh	
?		[?]	FFh	!skok dozadu
3E		LD A, N	00h	
28			01h	!skok dopředu
C6		ADD A, N	02h	
13			03h	
27		DAA	04h	
C9	KONEC	RET	05h	v

Pokud chceme v tomto případě skočit na návěští ZNOVU, bude hodnota n rovna FAh, pokud chceme skočit na návěští KONEC, bude to 05. Tato hodnota se objeví místo otazníku.

Pracujeme-li s překladačem, který dovoluje skoky na návěští, je to jednodušší. Nemusíme nic počítat a stačí

JR ZNOVU nebo JR KONEC

Instrukce DJNZ odpovídá basicové instrukci FOR-NEXT.  
Chceme 3x provést podprogram uložený na adrese 7FE0h.

```

06          LD B,3      ; nastavení počítadla
03          ; na hodnotu 03
CD    ZNOVU    CALL #7FE0 ; volání podprogramu
E0          ; nižší bajt adresy
7F          ; vyšší bajt adresy
10          DJNZ ZNOVU  ; skok na ZNOVU
FB          ; hodnota n

```

### Instrukce volání a návratu

CALL nn instrukce volání podprogramu  
=====

Instrukce provádí vyvolání podprogramu uloženého na adrese nn (t.j. skok na adresu nn) a po instrukci návratu, kterou musí být každý podprogram ukončen, pokračuje program instrukcí následující po instrukci CALL. Jeden podprogram je možno vyvolat z více míst programu a z jednoho podprogramu lze volat další podprogram. Poněvadž instrukce CALL ukládá adresu instrukce následující po instrukci CALL na vrchol zásobníku obdobným způsobem jako instrukce PUSH, musíme dbát na to, aby při návratu z podprogramu byla takto uložena adresa opět na vrcholu zásobníku. Platí to pro případ, že v podprogramu manipulujeme instrukcemi PUSH a POP se zásobníkem.

Nastavení indikátorů zůstává nezměněno.

CALL 1.,nn instrukce podmíněného volání podprogramu  
=====

Instrukce provádí vyvolání podprogramu jako CALL nn, ale jen tehdy, je-li splněna podmínka daná 1. operandem, kterou je nastavení indikátoru. Není-li tato podmínka splněna, pokračuje program následující instrukcí.

Možné kombinace:

instrukce	stav indikátoru pro provedení
CALL M,nn	S nastaven (výsledek<0)
CALL P,nn	S nulován (výsledek >=0)
CALL Z,nn	Z nastaven (výsledek =0)
CALL NZ,nn	Z nulován (výsledek<> 0)
CALL PE,nn	P/V nastaven (sudá parita)
CALL PO,nn	P/V nulován (lichá parita)
CALL C,nn	C nastaven (přenos)
CALL NC,nn	C nulován (bez přenosu)

Nastavení indikátorů zůstává nezměněno.



RET instrukce návratu z podprogramu (RETurn)  
===

Instrukce provede návrat z podprogramu na instrukci následující po instrukci CALL, která podprogram vyvolala. Nastavení indikátorů zůstává nezměněno.

RET I. instrukce podmíněného návratu z podprogramu  
=====

Instrukce provede návrat z podprogramu podobně jako instrukce RET, ale pouze tehdy, je-li splněna podmínka daná v operandu. Touto podmínkou je nastavení indikátoru.

instrukce	stav indikátoru pro provedení
-----------	-------------------------------

RET M	S nastaven (výsledek <0)
RET P	S nulován (výsledek >=0)
RET Z	Z nastaven (výsledek =0)
RET NZ	Z nulován (výsledek <>0)
RET PE	P/V nastaven (sudá parita)
RET PO	P/V nulován (lichá parita)
RET C	C nastaven (přenos)
RET NC	C nulován (bez přenosu)

Není-li splněna podmínka v operandu, pokračuje program následující instrukcí. Nastavení indikátorů zůstává nezměněno.

RETI instrukce návratu z maskovaného přerušení  
==== (RETurn from Interrupt)

Instrukce provádí návrat z obslužného programu přerušení obdobně jako instrukce RET a současně oznamuje perifernímu zařízení, že končí jeho obsluhu. Instrukce neprovádí uvolnění přerušení, proto je před ní třeba vložit instrukci EI. Nastavení indikátorů zůstává nezměněno.

RETN instrukce návratu z nemaskovaného přerušení  
==== (RETurn from Non maskable interrupt)

Instrukce provádí návrat z nemaskovaného přerušení obdobně jako instrukce RET. Nastavení indikátorů zůstává nezměněno.

RST n instrukce restartu  
=====

Instrukce provede skok do podprogramu na adresu danou operandem.  
Možné tvary instrukce RST:

RST #00	RST #08	RST #10	RST #18
RST #20	RST #28	RST #30	RST #38

Nastavení indikátorů zůstává nezměněno.

Instrukce v této kapitole jsou tak jednoduché, že na ně není třeba uvádět příklady.

## Instrukce rotací a posunů

SLA     instrukce aritmetického posunu vlevo  
 ===     (Shift Left Arithmetic)

Instrukce posune bity v daném registru, který je uveden v operandu tak, že bit 7 se přesune do CY a bit 0 se naplní 0.

		CY	7	6	5	4	3	2	1	0
Obsah registru: před operací	[ ]		[h]	[g]	[f]	[e]	[d]	[c]	[b]	[a]
			/	/	/	/	/	/	/	/
po operaci		[h]		[g]	[f]	[e]	[d]	[c]	[b]	[a]
										[0]<0

Možné kombinace operandů:

SLA A	SLA H
SLA B	SLA L
SLA C	SLA (HL)
SLA D	SLA (IX+d)
SLA E	SLA (IY+d)

Nastavení indikátorů:

S    nastaven, je-li výsledek záporný, jinak nulován  
 Z    nastaven, je-li výsledek nulový, jinak nulován  
 H    nulován  
 P/V nastaven, je-li parita sudá, nulován je-li lichá  
 N    nulován  
 C    obsahuje 7.bit operandu

SRA r   instrukce aritmetického posunu vpravo  
 ===== (Shift Right Arithmetic)

Instrukce posune bity v registru daném operandem doprava tak, že bit 7 zůstane zachován a bit 0 se přesune do CY.

			7	6	5	4	3	2	1	0	CY
Obsah registru: před operací		[h]	[g]	[f]	[e]	[d]	[c]	[b]	[a]	[ ]	
			/	/	/	/	/	/	/	/	
po operaci		[h]	[h]	[g]	[f]	[e]	[d]	[c]	[b]	[a]	

Možné operandy:

SRA A	SRA H
SRA B	SRA L
SRA C	SRA (HL)
SRA D	SRA (IX+d)
SRA E	SRA (IY+d)

Nastavení indikátorů:

S    nastaven, je-li výsledek záporný, jinak nulován  
 Z    nastaven, je-li výsledek nulový, jinak nulován  
 H    nulován  
 P/V nastaven, je-li parita sudá, nulován je-li lichá  
 N    nulován  
 C    obsahuje 0.bit operandu

SRL r instrukce logického posunu vpravo  
==== (Shift Right Logic)

Instrukce posune bity v registru daném operandem doprava tak, že bit 7 se naplní nulou a bit 0 se přesune do CY.

		7	6	5	4	3	2	1	0	CY
Obsah registru: před operací	[h]	[g]	[f]	[e]	[d]	[c]	[b]	[a]	[ ]	
po operaci	0	[h]	[g]	[f]	[e]	[d]	[c]	[b]	[a]	

Možné kombinace operandů:

SRL A	SRL H
SRL B	SRL L
SRL C	SRL (HL)
SRL D	SRL (IX+d)
SRL E	SRL (IY+d)

Nastavení indikátorů:

S nastaven, je-li výsledek záporný, jinak nulován  
Z nastaven, je-li výsledek nulový, jinak nulován  
H nulován  
P/V nastaven, je-li parita sudá, je-li lichá nulován  
N nulován  
C obsahuje 0.bit operandu

RL r instrukce rotace operandu vlevo přes CY  
==== (Rotate Left)

Instrukce provádí rotaci bitů v registru daném operandem vlevo přes indikátor C. Obsah indikátoru C se přesune do bitu 0 operandu.

		CY	7	6	5	4	3	2	1	0
Obsah registru: před operací	-[i]	[h]	[g]	[f]	[e]	[d]	[c]	[b]	[a]	
		/	/	/	/	/	/	/	/	
po operaci	[h]	[g]	[f]	[e]	[d]	[c]	[b]	[a]	[i]<-	

Možné operandy:

RL A	RL H
RL B	RL L
RL C	RL (HL)
RL D	RL (IX+d)
RL E	RL (IY+d)

Nastavení indikátorů:

S nastaven, je-li výsledek záporný, jinak nulován  
Z nastaven, je-li výsledek nulový, jinak nulován  
H nulován  
P/V nastaven, je-li parita sudá, je-li lichá nulován  
N nulován  
C obsahuje 7.bit operandu

RLA instrukce rotace registru A vlevo přes CY  
 === (Rotate Left A)

Instrukce provádí totéž jako instrukce RL, ale platí pouze pro registr A.

Nastavení indikátorů:

S, Z a P/V nezměněny  
 R a N nulovány  
 C obsahuje bit 7 registru A

RLC r instrukce rotace operandu vlevo do CY  
 ===== (Rotate Left Carry)

Instrukce provede rotaci bitů operandu doleva tak, že bit 7 přesune do bitu 0 a současně do CY.

	CY	7	6	5	4	3	2	1	0
Obsah registru: před operací	[X]	[h]	[g]	[f]	[e]	[d]	[c]	[b]	[a]
	/	/	/	/	/	/	/	/	/
po operaci	[h]	[g]	[f]	[e]	[d]	[c]	[b]	[a]	[h]←
	-----								

Možné operandy:

RLC A	RLC H
RLC B	RLC L
RLC C	RLC (HL)
RLC D	RLC (IX+d)
RLC E	RLC (IY+d)

Nastavení indikátorů:

S nastaven, je-li výsledek záporný, jinak nulován  
 Z nastaven, je-li výsledek nulový, jinak nulován  
 H nulován  
 P/V nastaven, je-li parita sudá, je-li lichá nulován  
 N nulován  
 C obsahuje 7. bit operandu

RLCA instrukce rotace registru A vlevo do CY  
 ===== (Rotate Left Carry A)

Instrukce provádí totéž jako instrukce RLC, ale platí pouze pro registr A.

Nastavení indikátorů:

S, Z a P/V nezměněny  
 H a N nulovány  
 C obsahuje bit 7 registru A

RR r instrukce rotace operandu vpravo přes CY  
 ==== (Rotate Right)

Instrukce provádí rotaci bitů operandu vpravo přes CY tak, že bit 0 operandu se přesune do CY a obsah CY do bitu 7 operandu.

	7	6	5	4	3	2	1	0	CY
Obsah registru: před operací	[h]	[g]	[f]	[e]	[d]	[c]	[b]	[a]	[i]-
		\	\	\	\	\	\	\	\
po operaci	->[i]	[h]	[g]	[f]	[e]	[d]	[c]	[b]	[a]

Možné operandy:

RR A	RR H
RR B	RR L
RR C	RR (HL)
RR D	RR (IX+d)
RR E	RR (IY+d)

Nastavení indikátorů:

S nastaven, je-li výsledek záporný, jinak nulován  
 Z nastaven, je-li výsledek nulový, jinak nulován  
 H nulován  
 P/V nastaven, je-li parita sudá, je-li lichá nulován  
 N nulován  
 C obsahuje 0. bit operandu

RRA instrukce rotace registru A vpravo přes CY  
 === (Rotate Right A)

Instrukce provádí totéž jako instrukce RR, ale platí pouze pro registr A.

Nastavení indikátorů:

S, Z a P/V nezměněny  
 H a N nulovány  
 C obsahuje bit 0 registru A.

RRC r instrukce rotace operandu vpravo do CY  
 ===== (Rotate Right Carry)

Instrukce provádí rotaci bitů operandu doprava tak, že bit 0 přeneseme do bitu 7 a současně do CY.

	7	6	5	4	3	2	1	0	CY
Obsah registru: před operací	[h]	[g]	[f]	[e]	[d]	[c]	[b]	[a]	[X]
		\	\	\	\	\	\	\	\
po operaci	->[a]	[h]	[g]	[f]	[e]	[d]	[c]	[b]	[a]

Možné operandy:

RRC A	RRC H
RRC B	RRC L
RRC C	RRC HL
RRC D	RRC (IX+d)
RRC E	RRC (IY+d)

Nastavení indikátorů:

S nastaven, je-li výsledek záporný, jinak nulován  
 Z nastaven, je-li výsledek nulový, jinak nulován  
 H nulován  
 P/V nastaven, je-li parita sudá, je-li lichá nulován  
 N nulován  
 C obsahuje 0.bit operandu

RRCA instrukce rotace registru A vpravo do CY  
 ==== (Rotate Right Carry A)

Instrukce provádí totéž jako instrukce RRA, ale platí pouze pro registr A.

Nastavení indikátorů:

S, Z a P/V nezměněny  
 H a N nulovány  
 C obsahuje bit 0 registru A.

RLD instrukce BCD rotace vlevo  
 === (Rotate Left bcd)

Instrukce provádí BCD rotaci mezi nižšími čtyřmi bity registru A a bity pamětového místa, jehož adresa je v registru HL. Způsob přenosu je následující: nižší čtyři bity pamětového místa se přenesou do vyšších čtyř bitů tohoto pamětového místa, vyšší čtyři bity pamětového místa se přenesou do nižších čtyř bitů registru A a nižší čtyři bity registru A se přenesou do nižších čtyř bitů pamětového místa. Vypadá to složitě a názorněji je to vidět na schématu:

	registr A		pamět.místo (HL)	
Obsah: před operací	[7:4]	[3:0]	[7:4]	[3:0]
po operaci	[   ]	[   ]	[   ]	[   ] <-
	-----			

Nastavení indikátorů:

S nastaven, je-li obsah registru A záporný jinak nulován  
 Z nastaven, je-li obsah registru A nula jinak nulován  
 H nulován  
 P/V nastaven, má-li registr A sudou paritu jinak nulován  
 N nulován  
 C nezměněn

RRD instrukce BCD rotace vpravo  
 === (Rotate Right bCD)

Instrukce provádí rotaci nižších čtyř bitů registru A a bitů pamětového místa, jehož adresa je v registru HL. Způsob přenosu je následující: nižší čtyři bity registru A se přenesou do vyšších čtyř bitů pamětového místa, vyšší čtyři bity pamětového místa se přenesou do jeho nižších čtyř bitů a nižší čtyři bity pamětového místa se přenesou do nižších čtyř bitů registru A.

	registr A	pamět.místo (HL)
Obsah: před operací	[7654][3210]	[7654][3210]--
po operaci	[   ] [   ]	[   ] [   ]

Nastavení indikátorů:

S nastaven, je-li obsah registru A záporný jinak nulován  
 Z nastaven, je-li obsah registru A nula jinak nulován  
 H nulován  
 P/V nastaven, má-li registr A sudou paritu jinak nulován  
 N nulován  
 C nezměněn

U všech instrukcí rotací a posunů platí, že (HL) znamená obsah pamětového místa, jehož adresa je v registru HL, a že (IX+d) je obsah pamětového místa jehož adresu dostaneme přičtením indexu d k adrese v registru IX. Stejně tak pro registr IV.

Příklady na použití instrukcí rotací a posunu:

Chceme na obrazovku vypsat v BCD vyjádření obsah registru. Počítač hodnotu obsaženou v registru interpoluje jako kód znaku a tento znak, jehož kód je v registru, vypíše na obrazovku. Chceme-li na obrazovku dostat BCD obsah registru, musíme jeho obsah rozdělit na dvě poloviny. Horní čtyři bity reprezentují první BCD číslici, dolní čtyři bity druhou BCD číslici. Tyto čtyři bity musíme samostatně vyjádřit jako kód znaku a přičíst k tomuto kódu kód nuly. U Spectra je to 48d = 30h. Postupujeme tak, že po uložení obsahu registru do zásobníku přesuneme čtyřnásobnou instrukcí SRL horní čtyři bity do dolních čtyř bitů. V horních čtyřech bitech jsou nuly. Nyní k takto získanému kódu můžeme přičíst kód nuly a máme první číslici, kterou můžeme přenést do části paměti vyhrazené pro obsah obrazovky. V assembleru to bude :

```
LD A,#93      ;hodnota #93 do registru A
PUSH AF       ;uchování registru A pro pozdější použití
SRL A         ;posun reg. A o jeden bit vpravo
SRL A         ;dtto
SRL A         ;dtto
SRL A         ;dtto
ADD A,#30     ;přičtení kódu nuly k kódu znaku v A
RST #10       ;výpis na obrazovku
```

Získání druhé HEX číslice bude o něco jednodušší. Po obnovení původního obsahu registru A instrukcí POP v něm máme opět obě BCD číslice, t.j. #93. První číslici "9" již máme. Horní čtyři bity tudíž nepotřebujeme a můžeme je vynulovat instrukcí AND #0F. Pak už jen zbývá přičíst 30h a přenést takto získaný kód na obrazovku. Assembler bude pokračovat:

```
POP AF          ;obnovení původního obsahu registru A
AND #0F         ;vynulování horních čtyř bitů registru A
ADD A,#30       ;přičtení kódu nuly k kódu znaku v A
RST #10        ;výpis dalšího znaku na obrazovku
```

Jinak se instrukcí z této kapitoly používá převážně při provádění aritmetických operací násobení a dělení. Máme vynásobit dvě 8-bitová kladná čísla, např. 02 a 06. Bude to vypadat následovně:

```
LD E,02         ;první číslo do E
LD C,03         ;druhé číslo do C
LD HL,00        ;vynulování HL pro ukládání výsledku
LD D,0          ;vynulování D
LD B,8          ;nastavení počítadla
ZNOVU SRL C     ;další bit do CY
JR NC,DAL       ;je-li CY=0, nesčítat
ADD HL,DE        ;je-li CY=1, přičíst k HL
DAL SLA E        ;násobení DE
RL D             ;dvěma
DJNZ ZNOVU      ;test na B=0, t.j. na provedení osmi cyklů
```

Komentář snad není třeba, nechť se každý procvičí přemýšlením jak to funguje.

Ještě snad malé vysvětlení k instrukcím RLD a RRD. Mějme například obsah reg. A = 56, obsah reg. HL = #4038 a obsah pamětového místa s adresou #4038 roven #21.

Stav registru a paměti bude:

před provedením operace:

H	L	adresa 4038	A
[0100:0000]	[0011:1000]	-[0010:0001]	[0101:0110]
[0100:0000]	[0011:1000]	[0001:0110]	[0101:0010]

před provedením operace:

H	L	adresa 4038	A
[0100:0000]	[0011:1000]	-[0010:0001]	[0101:0110]---
[0100:0000]	[0011:1000]	[0110:0010]	<-- [0101:0001]<--



## Instrukce pro práci s jednotlivými bity

---

Následující instrukce umožňují testování, nastavování nebo nulování libovolného bitu v registrech a paměťových místech.

BIT b,r instrukce pro test bitu  
=====

Instrukce přenesle komplement bitu "b" operandu do indikátoru Z.

Možné kombinace:

BIT 0,A	BIT 0,B	BIT 0,C	BIT 0,D	BIT 0,E	BIT 0,H
BIT 1,A	BIT 1,B	BIT 1,C	BIT 1,D	BIT 1,E	BIT 1,H
BIT 2,A	BIT 2,B	BIT 2,C	BIT 2,D	BIT 2,E	BIT 2,H
BIT 3,A	BIT 3,B	BIT 3,C	BIT 3,D	BIT 3,E	BIT 3,H
BIT 4,A	BIT 4,B	BIT 4,C	BIT 4,D	BIT 4,E	BIT 4,H
BIT 5,A	BIT 5,B	BIT 5,C	BIT 5,D	BIT 5,E	BIT 5,H
BIT 6,A	BIT 6,B	BIT 6,C	BIT 6,D	BIT 6,E	BIT 6,H
BIT 7,A	BIT 7,B	BIT 7,C	BIT 7,D	BIT 7,E	BIT 7,H

BIT 0,L	BIT 0,(HL)	BIT 0,(IX+d)	BIT 0,(IY+d)
BIT 1,L	BIT 1,(HL)	BIT 1,(IX+d)	BIT 1,(IY+d)
BIT 2,L	BIT 2,(HL)	BIT 2,(IX+d)	BIT 2,(IY+d)
BIT 3,L	BIT 3,(HL)	BIT 3,(IX+d)	BIT 3,(IY+d)
BIT 4,L	BIT 4,(HL)	BIT 4,(IX+d)	BIT 4,(IY+d)
BIT 5,L	BIT 5,(HL)	BIT 5,(IX+d)	BIT 5,(IY+d)
BIT 6,L	BIT 6,(HL)	BIT 6,(IX+d)	BIT 6,(IY+d)
BIT 7,L	BIT 7,(HL)	BIT 7,(IX+d)	BIT 7,(IY+d)

Nastavení indikátorů:

S a P/V nedefinovány  
H nastaven  
N nulován  
C nezměněn  
Z obsahuje komplement specifikovaného bitu.

SET b,r instrukce nastavení bitu  
=====

Instrukce provede nastavení bitu "b" operandu, t.j. uloží do něj hodnotu 1.

Možné kombinace:

SET 0,A	SET 0,B	SET 0,C	SET 0,D	SET 0,E	SET 0,H
SET 1,A	SET 1,B	SET 1,C	SET 1,D	SET 1,E	SET 1,H
SET 2,A	SET 2,B	SET 2,C	SET 2,D	SET 2,E	SET 2,H
SET 3,A	SET 3,B	SET 3,C	SET 3,D	SET 3,E	SET 3,H
SET 4,A	SET 4,B	SET 4,C	SET 4,D	SET 4,E	SET 4,H
SET 5,A	SET 5,B	SET 5,C	SET 5,D	SET 5,E	SET 5,H
SET 6,A	SET 6,B	SET 6,C	SET 6,D	SET 6,E	SET 6,H
SET 7,A	SET 7,B	SET 7,C	SET 7,D	SET 7,E	SET 7,H

SET 0,L	SET 0,(HL)	SET 0,(IX+d)	SET 0,(IY+d)
SET 1,L	SET 1,(HL)	SET 1,(IX+d)	SET 1,(IY+d)
SET 2,L	SET 2,(HL)	SET 2,(IX+d)	SET 2,(IY+d)
SET 3,L	SET 3,(HL)	SET 3,(IX+d)	SET 3,(IY+d)
SET 4,L	SET 4,(HL)	SET 4,(IX+d)	SET 4,(IY+d)
SET 5,L	SET 5,(HL)	SET 5,(IX+d)	SET 5,(IY+d)
SET 6,L	SET 6,(HL)	SET 6,(IX+d)	SET 6,(IY+d)
SET 7,L	SET 7,(HL)	SET 7,(IX+d)	SET 7,(IY+d)

Nastavení indikátorů zůstává nezměněno.

RES b,r instrukce nulování bitu (RESet)

=====

Instrukce provede nulování bitu "b" operandu, t.j. uloží do něj hodnotu 0.

Možné kombinace:

RES 0,A	RES 0,B	RES 0,C	RES 0,D	RES 0,E	RES 0,H
RES 1,A	RES 1,B	RES 1,C	RES 1,D	RES 1,E	RES 1,H
RES 2,A	RES 2,B	RES 2,C	RES 2,D	RES 2,E	RES 2,H
RES 3,A	RES 3,B	RES 3,C	RES 3,D	RES 3,E	RES 3,H
RES 4,A	RES 4,B	RES 4,C	RES 4,D	RES 4,E	RES 4,H
RES 5,A	RES 5,B	RES 5,C	RES 5,D	RES 5,E	RES 5,H
RES 6,A	RES 6,B	RES 6,C	RES 6,D	RES 6,E	RES 6,H
RES 7,A	RES 7,B	RES 7,C	RES 7,D	RES 7,E	RES 7,H

RES 0,L	RES 0,(HL)	RES 0,(IX+d)	RES 0,(IY+d)
RES 1,L	RES 1,(HL)	RES 1,(IX+d)	RES 1,(IY+d)
RES 2,L	RES 2,(HL)	RES 2,(IX+d)	RES 2,(IY+d)
RES 3,L	RES 3,(HL)	RES 3,(IX+d)	RES 3,(IY+d)
RES 4,L	RES 4,(HL)	RES 4,(IX+d)	RES 4,(IY+d)
RES 5,L	RES 5,(HL)	RES 5,(IX+d)	RES 5,(IY+d)
RES 6,L	RES 6,(HL)	RES 6,(IX+d)	RES 6,(IY+d)
RES 7,L	RES 7,(HL)	RES 7,(IX+d)	RES 7,(IY+d)

Nastavení indikátorů zůstává nezměněno.

## 4.3 Pseudoinstrukce assembleru

V předchozí kapitole jsme probrali všechny strojní instrukce mikroprocesoru Z 80. Nyní se ale ještě musíme vrátit k assembleru. Každý "dospělý" Assembler (t.j. překladač) může vedle strojních instrukcí pracovat i s pseudoinstrukcemi. I když při ručním vkládání strojního programu nejsou pseudoinstrukce nic platné, je třeba je znát. Jednak pro případ, že budete pracovat s takovým Assemblerem, který dovoluje použití pseudoinstrukcí (i když ne třeba všech), jednak pro případ, že budete aplikovat strojní programy z různých pramenů, ve kterých také může být použito pseudoinstrukcí.

Pseudoinstrukce nemají přímý vliv na činnost procesoru, jako strojní instrukce, ale pouze řídí překládací program, t.j. assembler.

Jejich formát je stejný, jako formát strojních instrukcí, t.j. píšeme je do stejného místa a stejným způsobem jako strojní instrukce. Podobně jako u strojních instrukcí i u pseudo-instrukcí můžeme používat návěští, operandy a komentáře. Nyní k jednotlivým pseudoinstrukcím.

```
DEFB    pseudoinstrukce definice bajtu
====    (DEFine Byte)
```

Pseudoinstrukce generuje jeden bajt s hodnotou danou operandem v tom místě, kde je pseudoinstrukce v programu uvedena. Poněvadž se jedná pouze o 1 bajt, může být generovaná hodnota (t.j. hodnota daná operandem) v mezích -128d=80h až 255d=FFh.

Příklad:

-----

```
adresa    návěští    instrukce    komentář
4520      DVE        DEFB 02      ; uloží na adresu #4520 hodnotu
                                   ; tu 02
```

```
DEFW    pseudoinstrukce definice slova
====    (DEFine Word)
```

Pseudoinstrukce generuje dva bajty s hodnotou danou operandem v tom místě, kde je pseudoinstrukce v programu uvedena. Nejprve se ukládá nižší bajt a pak vyšší bajt.

Příklad:

-----

```
4520      CISO      DEFW #025D    ; uloží na adresu #4520
                                   ; hodnotu #5D a na adresu #4521
                                   ; hodnotu #02
```

U obou pseudoinstrukcí DEFB a DEFW může být více operandů. Pseudoinstrukce vykonává svoji funkci opakovaně pro každý operand.

Příklad:

-----

```
4520      STO      DEFB 01,00,00  ; uloží postupně na adresy
                                   ; #4520 hodnotu 01
                                   ; #4521      00
                                   ; #4522      00
4520      DEFW #0100,#0200 ; uloží postupně na
                                   ; adresy: #4520      00
                                   ; #4521      01
                                   ; #4522      00
                                   ; #4523      02
```

DEFM    pseudoinstrukce definice hlášení  
 ====    (DEFine Message)

Pseudoinstrukce generuje posloupnost bajtů obsahujících ASCII kódy znaků operandu. Operand musí být řetězec, t.j. výraz musí být uzavřen v uvozovkách.

Příklad:

```
-----
4520          DEFM "TEXT" ; generuje na adresách:
                ; #4520      54 (ASCII kód T)
                ; #4521      45      E
                ; #4522      58      X
                ; #4523      54      T
```

DEFS    pseudoinstrukce definice paměti  
 ====    (DEFine String)

Pseudoinstrukce rezervuje v paměti počet bajtů daný operandem, počínaje místem, ve kterém je pseudoinstrukce v programu uvedena.

Příklad:

```
-----
4520    VYSL    DEFS 4    ;rezervuje 4 bajty na adresách 4520 až
                        4523 pro uložení výsledku
```

EQU    pseudoinstrukce definice symbolu  
 ===

Pseudoinstrukce přiřadí hodnotu operandu k návěští. V dalším textu již nemusíme vypisovat vždy hodnotu operandu, ale stačí uvést návěští.

Příklad:

```
-----
4520    BUFFER   EQU #FF00 ; přiřadí BUFFER hodnotu #FF00
        :
        :
5200              LD HL,(BUFFER) ; provede LD HL,(#FF00)
```

ORG    pseudoinstrukce nastavení referenčního čítače  
 ===

Pseudoinstrukce nastaví referenční čítač na adresu danou operandem. Znamená to, že ta část programu, která následuje za pseudoinstrukcí ORG se překládá až od adresy udané operandem.

Příklad:

```
-----
26EB           ORG #26EB        ; program se začne překládat
26EB                          ; od adresy #26EB
26EB    ED4B00FF   LD BC,(BUFFER) ; BUFFER do BC
26EF    C9         RET         ; konec podprogramu
```

END        pseudoinstrukce konce zdrojového programu

===

Pseudoinstrukce END označuje konec zdrojového programu. Další text za touto pseudoinstrukcí není překládán.

Následující pseudoinstrukce Assembler Gens 3 nemá, ale je vhodné se s nimi seznámit.

DEFT       pseudoinstrukce definice textu

====       (DEFine Text)

Pseudoinstrukce vykonává stejnou činnost jako DEFM, ale navíc do prvního bajtu ukládá délku řetězce (operandu).

Příklad:

-----

```

4520                    DEFT "TEXT" ; generuje na adresách:
                              ; #4520 hodnota #04
                              ; #4521                #54
                              ; #4522                #45
                              ; #4523                #58
                              ; #4524                #54

```

DEFL       pseudoinstrukce definice návěští

====       (DEFine Label)

Pseudoinstrukce má stejnou funkci jako EQU s tím rozdílem, že operand DEFL může mít na různých místech programu různé hodnoty. Hodnota operandu platí vždy až do dalšího předefinování.

Příklad:

-----

```

4520    ZNOVU    DEFL 05       ; přiřadí ZNOVU hodnotu 05
          ;
4550                DJNZ ZNOVU ; provede DJNZ 05
          ;
4580    ZNOVU    DEFL #FD      ; přiřadí ZNOVU hodnotu #FD
          ;
459F                DJNZ ZNOVU ; provede DJNZ #FD

```

GLOBAL     pseudoinstrukce definice globalu

=====

Pseudoinstrukce slouží k definování jmen, na které je možno se odvolat z jiných modulů.

EXTERNAL      pseudoinstrukce externalu  
=====

Pseudoinstrukce umožňuje označit jména zadaná v operandu jako vnější. Vnější jména jsou taková, která jsou použita v překládacím modulu v poli operandů, ale definována jsou v jiném modulu.

Příklad:  
-----

Nazveme-li program pro násobení dvou čísel, uvedený v kapitole instrukcí rotací a posunů, NASOB, a označíme-li paměťová místa pro uložení dvou čísel které násobíme návěštím OP1 a OP2 a paměťové místo pro uložení výsledku návěštím VYSL, můžeme uvedený program upravit následovně:

```

GLOBAL NASOB                ; název programu
EXTERNAL OP1, OP2, VYSL     ; vstupy
NASOB  LD A,(OP1)            ; první vstup
      LD E,A                ; přesun prvního čísla do E
      LD A,(OP2)            ; druhý vstup
      LD C,A                ; přesun druhého čísla do C
      LD HL,0               ; program pokračuje dále
      :
      :
      DJNZ ZNOVU            ; testuje konec násobení
      LD (VYSL),HL          ; uloží výsledek na VYSL

```

## 5. Sestavení strojního programu =====

Když teď ovládáte jak strojní instrukce Z80, tak pseudo-instrukce assembleru, můžete klidně sestavovat programy ve strojovém kódu. Není to ovšem tak jednoduché. Prvním a základním předpokladem správné funkce strojního programu je důkladný rozbor dané úlohy a sestavení podrobného vývojového diagramu. V Basicu se u jednodušších programů bez vývojového diagramu klidně obejdeme, ale při sestavování strojního programu je téměř nezbytný, poněvadž na něm musíme "vyháčkovat" funkci jednotlivých registrů a indikátorů.

Vezměme si například program pro násobení dvou čísel. V Basicu vystačíme s jedním příkazem LET C = A \* B. Vývojový diagram je velice jednoduchý a může vypadat následovně:

```

      ( začátek )
      V
      [ zadání čísel ]
      V
      [ násobení ]
      V
      [ výpis výsledku ]
      V
      ( konec )

```

Odpovídající Basicový program bude vypadat následovně:

```
10 INPUT A
20 INPUT B
30 LET C = A * B
40 PRINT C
50 STOP
```

Velice jednoduchá záležitost. Horší to bude ve strojovém kódu. Pokusme se sestavit vývojový diagram a celý strojní program pro vynásobení dvou celých kladných čísel.

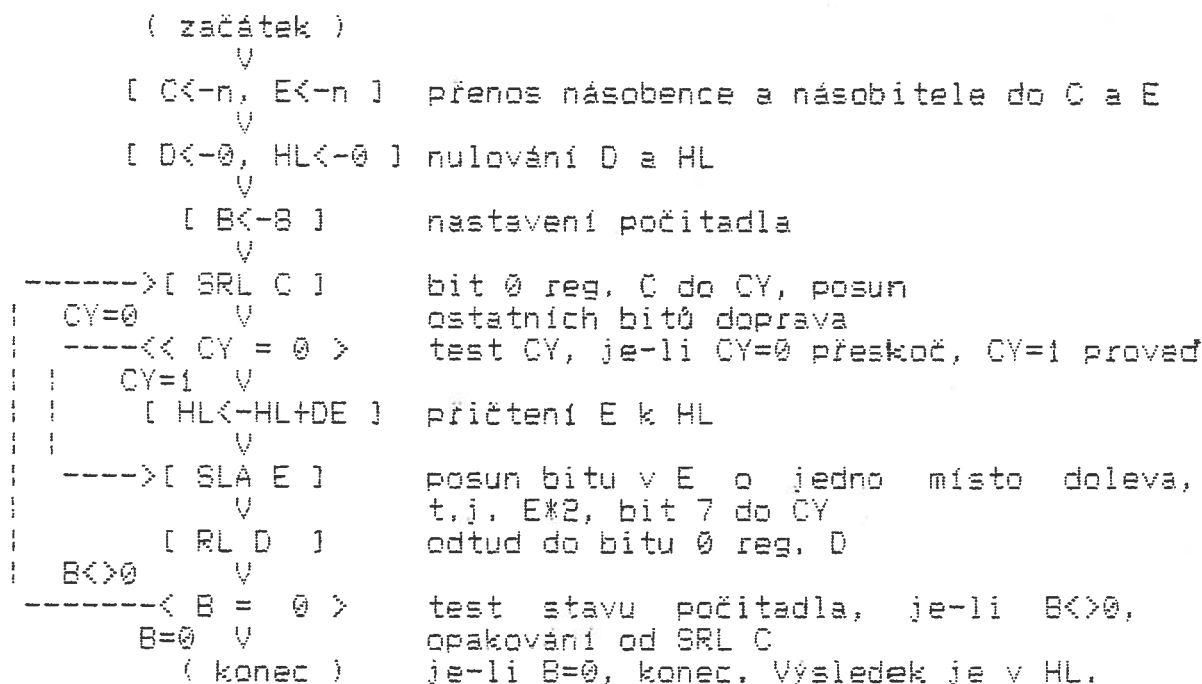
Nejprve se musíme vžít do role procesoru a představit si, jak bude danou úlohu řešit. Musíme zpět do druhé kapitoly k násobení binárních čísel, poněvadž procesor nezná nic jiného než binární soustavu. Pro lepší názornost zvolme konkrétní případ násobení dvou čísel  $7 * 5$ . Víme, že binárně násobíme tak, že sečítáme násobky dvou. V našem případě bude  $7 * 5 = 7 * (1 + 4)$ . Přeneseno do binární soustavy bude:

počáteční stav registru	0000 0000	
násobitel 05 =	0000 0101	
násobenec		
07 * 1 =	0000 0111	0000 0111 je obsaženo v násobiteli, přičteme
07 * 2 =	0000 1110	není v násobiteli, nepřičítáme
07 * 4 =	0001 1100	0001 1100 je obsaženo v násobiteli, přičteme
07 * 8 =	0011 1000	není v násobiteli, nepřičítáme
dál už jsou v násobiteli jen nuly, nic nepřičítáme		
-----		
Výsledek	0010 0011	= 23h = 35d

Ted se jedná o to realizovat tento postup v registrech Z80. Nejprve musíme uvážit, které registry použijeme. Víme, že postup přičítání se bude opakovat. Potřebujeme počítadlo, tudíž registr B. Násobenec budeme násobit tak, že ho budeme v registru posunovat doleva. Jeden registr nám ale nebude stačit, mohlo by dojít k jeho přetečení, tudíž to nemůže být C, ale registr E, přičemž D je rezervován pro případné přetečení. Nemůžeme použít H a L, poněvadž ty musíme použít pro uložení výsledku, t.j. pro postupné přičítání jednotlivých dvojkových násobků násobence, poněvadž pouze u HL existuje instrukce ADD HL,DE. (šlo by to provést i naopak a pak LD D,H a LD E,L, ale je to delší.) Pro násobitel nám zbývá buď A nebo C. Je v podstatě jedno, který registr použijeme, ale je lepší ponechat A volný pro případ dalšího použití (např. přenos dat z paměti, má jako jediný možnost LD A,(nn)). Zbývá nám tudíž registr C.

Když máme obsazeny registry, můžeme uvažovat dále. Rozhodující jsou jednotlivé bity násobitele. Musíme je tudíž postupně odprava otestovat, mají-li hodnotu 0 nebo 1. Při hodnotě 1 budeme hodnotu v registru E přičítat k HL, bude-li hodnota bitu 0, nepřičteme nic. Při každém dalším bitu násobitele musíme

násobenec vynásobit dvěma, t.j. posunout v registru o jedno místo doleva. Násobitel má 8 bitů, t.j. musíme tento postup 8x opakovat. K testování bitu násobitele bychom mohli použít např. BIT 0,C, BIT 1,C atd. až BIT 7,C ale bylo by to zdolouhavé. Lepší je pomocí instrukce posunu přesunout bit 0 do CY. Nesmíme zapomenout na ošetření přetečení registru E, t.j. přesun přetečeného bitu do registru D. Můžeme to provést tak, že bit 7, který se při posunu doleva dostane do CY přeneseme z CY do bitu 0 registru D pomocí instrukce posunu přes CY. Vývojový diagram pak může vypadat následovně:



Máme-li vývojový diagram, o kterém se domníváme, že je správně sestaven, můžeme se pustit do assembleru. Obvykle postupujeme ve čtyřech etapách. V první etapě sestavíme vlastní assembler. Místa skoků a volání podprogramů musíme označit návěštímí. Ve druhé etapě dopíšeme kódy jednotlivých instrukcí a operandů. Zatím neznáme hodnoty relativních skoků n a adresy volání CALL a absolutních skoků. Pro ně vynecháme volná místa. Je nejlepší tato místa označit čtverečky. Ve třetí etapě spočítáme počet přeskočených bajtů a určíme hodnoty n, které doplníme do čtverečků. V poslední čtvrté etapě doplníme absolutní adresy a podle nich i adresy absolutních skoků a volání (dopíšeme je do volných čtverečků). Takto sestavený assembler s počáteční adresou strojního programu 32768d = 8000h bude vypadat takto:

4.et.	[3.] a 2.et.	1. etapa
8000	0E 05	LD C,05 ; násobitel do C
8002	1E 07	LD E,07 ; násobenec do E
8004	16 00	LD D,00 ; nulování D
8006	21 00 00	LD HL,00 ; nulování HL
8009	06 08	LD B,08 ; nastavení počítadla
800B	CB 39 ZNOVU	SRL C ; bit 0 reg.C do CY
800D	30[01]	JR NC,DAL ; test CY=0 skok na DAL



800F	19		ADD HL,DE	; přičti DE k HL
8010	CB 23	DAL	SLA E	; násobení E dvěma
8012	CB 12		RL D	; přenos bitu do D
8014	10[F5]		DJNZ ZNOVU	; test počítadla, B<>0
				; skok na ZNOVU
8016	44		LD B,H	; přenos výsledku
8017	40		LD C,L	; z HL do BC
8018	C9		RET	; návrat do Basicu

Ještě k poslední instrukci RET. Zatím v předchozích ukázkách nebyla uváděna, poněvadž se jednalo o výseky strojních programů bez dalších vazeb. Tento program je vlastně prvním fungujícím programem a proto musí být ukončen instrukcí RET. Platí jednoznačná zásada:

Každý strojní program musí být ukončen instrukcí RET.

Je to proto, že naše strojní programy jsou vlastně podprogramy Basicového systému počítače a musí se poslední instrukcí vrátit do Basicu. I pouhé PRINT USR 32768, které vyvolá na Spectru náš vzorový program, je vlastně basicovým podprogramem a způsobí, že se na obrazovku vypíše obsah registrového páru BC.

Ve druhém sloupci assembleru máme zdrojový program, který pak obvykle píšeme jako sérii bajtů, t.j. 0E 05 1E 07 .... 44 40 C9, nebo ve formě tzv. HEX-DUMP. HEX-DUMP je absolutním zdrojovým programem, poněvadž obsahuje i adresy. Je uváděna vždy adresa prvního bajtu v každém řádku, každý další bajt má adresu o jedničku vyšší. Někdy bývá na konci každého řádku uváděno ještě kontrolní číslo, které vyjadřuje buď paritu nebo hodnotu součtu bajtů v řádku. Náš program bude v HEX-DUMP vypadat následovně:

8000	=	32768	-	0E 05 1E 07 16 00 21 00	=	111
8008	=	32776	-	00 06 0B CB 39 30 C1 01	=	348
8010	=	32784	-	CB 23 CB 12 10 F5 44 40	=	865
8018	=	32792	-	C9	=	201

S použitím Assembleru Gens3 samozřejmě odpadají etapy 2 - 4, protože je za nás provede Assembler. Je tedy při programování ve strojním kódu použití Assembleru podstatným ulehčením práce. Navíc nás Gens upozorňuje na chyby v zápisu mnemoniky, pokus o zapsání do osmibitového registru větší hodnoty než 255d, můžeme používat u instrukcí skoků a relativních skoků návěští atd.

Tímto by vlastně mělo skončit vlastní sestavování programu ve strojním kódu. Teď zbývá pouze vložit program do počítače, spustit a čekat, bude-li fungovat. Funguje-li, je vyhráno, jestliže nefunguje, čeká nás obtížné hledání chyb. Bohužel málokterý program funguje hned napoprvé a většina se musí pracně odladovat, t.j. hledat chyby.

## 6. Závěr

### =====

K doplnění a opravám byly použity:

Assebbler Z-80 vydaný JZD Slušovice  
Z 80 Programing manual firemní literatura SGS-ATES

# Priloha A =====

Decimální hodnoty HEX číslic:

4. číslice HEX	3.	2.	1.
HEX	DEC		
0	0	0	0
1	4096	256	16
2	8192	512	32
3	12288	768	48
4	16384	1024	64
5	20480	1280	80
6	24576	1536	96
7	28672	1792	112
8	32768	2048	128
9	36864	2304	144
A	40960	2560	160
B	45056	2816	176
C	49152	3072	192
D	53248	3328	208
E	57344	3584	224
F	61440	3840	240

Převod HEX-DEC hodnot jednobajtových čísel:

	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	XA	XB	XC	XD	XE	XF
0X	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
1X	1	17	33	49	65	81	97	113	129	145	161	177	193	209	225	241
2X	2	18	34	50	66	82	98	114	130	146	162	178	194	210	226	242
3X	3	19	35	51	67	83	99	115	131	147	163	179	195	211	227	243
4X	4	20	36	52	68	84	100	116	132	148	164	180	196	212	228	244
5X	5	21	37	53	69	85	101	117	133	149	165	181	197	213	229	245
6X	6	22	38	54	70	86	102	118	134	150	166	182	198	214	230	246
7X	7	23	39	55	71	87	103	119	135	151	167	183	199	215	231	247
8X	8	24	40	56	72	88	104	120	136	152	168	184	200	216	232	248
9X	9	25	41	57	73	89	105	121	137	153	169	185	201	217	233	249
AX	10	26	42	58	74	90	106	122	138	154	170	186	202	218	234	250
BX	11	27	43	59	75	91	107	123	139	155	171	187	203	219	235	251
CX	12	28	44	60	76	92	108	124	140	156	172	188	204	220	236	252
DX	13	29	45	61	77	93	109	125	141	157	173	189	205	221	237	253
EX	14	30	46	62	78	94	110	126	142	158	174	190	206	222	238	254
FX	15	31	47	63	79	95	111	127	143	159	175	191	207	223	239	255

## Příloha B

### Abecední seznam strojních instrukcí.

ADC A,A	8F	AND L	A5
ADC A,B	88	AND n	E6 n
ADC A,C	89	AND (HL)	A6
ADC A,D	8A	AND (IX+d)	FD A6 d
ADC A,E	8B	AND (IY+d)	FD A6 d
ADC A,H	8C	BIT 0,A	CB 47
ADC A,L	8D	BIT 0,B	CB 40
ADC A,n	CE n	BIT 0,C	CB 41
ADC A,(HL)	8E	BIT 0,D	CB 42
ADC A,(IX+d)	DD 8E d	BIT 0,E	CB 43
ADC A,(IY+d)	FD 8E d	BIT 0,H	CB 44
ADC HL,BC	ED 4A	BIT 0,L	CB 45
ADC HL,DE	ED 5A	BIT 0,(HL)	CB 56
ADC HL,HL	ED 6A	BIT 0,(IX+d)	DD CB d 46
ADC HL,SP	ED 7A	BIT 0,(IY+d)	FD CB d 46
ADC A,A	87	BIT 1,A	CB 4F
ADD A,B	80	BIT 1,B	CB 48
ADD A,C	81	BIT 1,C	CB 49
ADD A,D	82	BIT 1,D	CB 4A
ADD A,E	83	BIT 1,E	CB 4B
ADD A,H	84	BIT 1,H	CB 4C
ADD A,L	85	BIT 1,L	CB 4D
ADD A,n	C6 n	BIT 1,(HL)	CB 4E
ADD A,(HL)	86	BIT 1,(IX+d)	DD CB d 4E
ADD A,(IX+d)	DD 86 d	BIT 1,(IY+d)	FD CB d 4E
ADD A,(IY+d)	FD 86 d	BIT 2,A	CB 57
ADD HL,BC	09	BIT 2,B	CB 50
ADD HL,DE	19	BIT 2,C	CB 51
ADD HL,HL	29	BIT 2,D	CB 52
ADD HL,SP	39	BIT 2,E	CB 53
ADD IX,BC	DD 09	BIT 2,H	CB 54
ADD IX,DE	DD 19	BIT 2,L	CB 55
ADD IX,IX	DD 29	BIT 2,(HL)	CB 56
ADD IX,SP	DD 39	BIT 2,(IX+d)	DD CB d 56
ADD IY,BC	FD 09	BIT 2,(IY+d)	FD CB d 56
ADD IY,DE	FD 19	BIT 3,A	CB 5F
ADD IY,IX	FD 29	BIT 3,B	CB 58
ADD IY,SP	FD 39	BIT 3,C	CB 59
AND A	A7	BIT 3,D	CB 5A
AND B	A0	BIT 3,E	CB 5B
AND C	A1	BIT 3,H	CB 5C
AND D	A2	BIT 3,L	CB 5D
AND E	A3	BIT 3,(HL)	CB 5E
AND H	A4	BIT 3,(IX+d)	DD CB d 5E

BIT 3, (IY+d)	FD CB d 5E	CALL Z ,nn	CC n n
BIT 4,A	CB 67	CALL nn	CD n n
BIT 4,B	CB 60	CCF	3F
BIT 4,C	CB 61	CP A	BF
BIT 4,D	CB 62	CP B	B8
BIT 4,E	CB 63	CP C	B9
BIT 4,H	CB 64	CP D	BA
BIT 4,L	CB 65	CP E	BB
BIT 4, (HL)	CB 66	CP H	BC
BIT 4, (IX+d)	DD CB d 66	CP L	BD
BIT 4, (IY+d)	FD CB d 66	CP n	FE 20
BIT 5,A	CB 6F	CP (HL)	BE
BIT 5,B	CB 68	CP (IX+d)	DD BE d
BIT 5,C	CB 69	CP (IY+d)	FD BE d
BIT 5,D	CB 6A	CPD	ED A9
BIT 5,E	CB 6B	CPDR	ED B9
BIT 5,H	CB 6C	CPI	ED A1
BIT 5,L	CB 6D	CPIR	ED B1
BIT 5, (HL)	CB 6E	CPL	2F
BIT 5, (IX+d)	DD CB d 6E	DAA	27
BIT 5, (IY+d)	FD CB d 6E	DEC A	30
BIT 6,A	CB 77	DEC B	05
BIT 6,B	CB 70	DEC BC	08
BIT 6,C	CB 71	DEC C	00
BIT 6,D	CB 72	DEC D	15
BIT 6,E	CB 73	DEC DE	18
BIT 6,H	CB 74	DEC E	10
BIT 6,L	CB 75	DEC H	25
BIT 6, (HL)	CB 76	DEC HL	28
BIT 6, (IX+d)	DD CB d 76	DEC IX	DD 28
BIT 6, (IY+d)	FD CB d 76	DEC IY	FD 28
BIT 7,A	CB 7F	DEC L	20
BIT 7,B	CB 78	DEC SP	38
BIT 7,C	CB 79	DEC (HL)	35
BIT 7,D	CB 7A	DEC (IX+d)	DD 35 d
BIT 7,E	CB 7B	DEC (IY+d)	FD 35 d
BIT 7,H	CB 7C	DI	F3
BIT 7,L	CB 7D	DJNZ e	10 e
BIT 7, (HL)	CB 7E	EI	FB
BIT 7, (IX+d)	DD CB d 7E	EX AF,AF'	08
BIT 7, (IY+d)	FD CB d 7E	EX DE,HL	EB
CALL C,nn	DC n n	EX (SP),HL	E3
CALL M,nn	FC n n	EX (SP),IX	DD E3
CALL NC,nn	D4 n n	EX (SP),IY	FD E3
CALL NZ,nn	C4 n n	EXX	D9
CALL P,nn	F4 n n	HALT	76
CALL PE,nn	EC n n	IM 0	ED 46
CALL PD,nn	E4 n n	IM 1	ED 56

IM 2	ED 5E	LD A,C	79
IN A, (C)	ED 78	LD A,D	7A
IN A, (n)	DB 20	LD A,E	7B
IN B, (C)	ED 40	LD A,H	7C
IN C, (C)	ED 48	LD A,I	ED 57
IN D, (C)	ED 50	LD A,L	7D
IN E, (C)	ED 58	LD A,R	ED 5F
IN H, (C)	ED 60	LD A,n	3E n
IN L, (C)	ED 68	LD A, (BC)	0A
INC A	3C	LD A, (DE)	1A
INC B	04	LD A, (HL)	7E
INC BC	03	LD A, (IX+d)	DD 7E d
INC C	0C	LD A, (IY+d)	FD 7E d
INC D	14	LD A, (nn)	3A n n
INC DE	13	LD B,A	47
INC E	1C	LD B,B	40
INC H	24	LD B,C	41
INC HL	23	LD B,D	42
INC IX	DD 23	LD B,E	43
INC IY	FD 23	LD B,H	44
INC L	2C	LD B,L	45
INC SP	33	LD B,n	06 20
INC (HL)	34	LD B, (HL)	46
INC (IX+d)	DD 34 d	LD B, (IX+d)	DD 46 d
INC (IY +d)	FD 34 d	LD B, (IY+d)	FD 46 d
IND	ED AA	LD BC,nn	01 n n
INDR	ED BA	LD BC, (nn)	ED 48 n n
INI	ED A2	LD C,A	4F
INIR	ED B2	LD C,B	48
JP C,nn	DA n n	LD C,C	49
JP M,nn	FA n n	LD C,D	4A
JP NC,nn	D2 n n	LD C,E	4B
JP NZ,nn	C2 n n	LD C,H	4C
JP P,nn	F2 n n	LD C,L	4D
JP PE,nn	EA n n	LD C,n	0E n
JP PD,nn	E2 n n	LD C, (HL)	4E
JP Z,nn	CA n n	LD C, (IX+d)	DD 4E d
JP nn	C3 n n	LD C, (IY+d)	FD 4E d
JP (HL)	E9	LD D,A	57
JP (IX)	DD E9	LD D,B	50
JP (IY)	FD E9	LD D,C	51
JR C,n	38 n	LD D,D	52
JR NC,n	30 n	LD D,E	53
JR NZ,n	20 n	LD D,H	54
JR Z,n	28 n	LD D,L	55
JR n	18 n	LD D,n	16 n
LD A,A	7F	LD D, (HL)	56
LD A,B	78	LD D, (IX+d)	DD 56 d

LD D, (IY+d)	FD 56 d	LD SP, (nn)	ED 78 n n
LD DE, nn	11 n n	LD (BC), A	02
LD DE, (nn)	ED 5B n n	LD (DE), A	12
LD E, A	5F	LD (HL), A	77
LD E, B	58	LD (HL), B	70
LD E, C	59	LD (HL), C	71
LD E, D	5A	LD (HL), D	72
LD E, E	5B	LD (HL), E	73
LD E, H	5C	LD (HL), H	74
LD E, L	5D	LD (HL), L	75
LD E, n	1E n	LD (HL), n	36 n
LD E, (HL)	5E	LD (IX+d), A	DD 77 d
LD E, (IX+d)	DD 5E d	LD (IX+d), B	DD 70 d
LD E, (IY+d)	FD 5E d	LD (IX+d), C	DD 71 d
LD H, A	67	LD (IX+d), D	DD 72 d
LD H, B	60	LD (IX+d), E	DD 73 d
LD H, C	61	LD (IX+d), H	DD 74 d
LD H, D	62	LD (IX+d), L	DD 75 d
LD H, E	63	LD (IX+d), n	DD 36 d n
LD H, H	64	LD (IY+d), A	FD 77 d
LD H, L	65	LD (IY+d), B	FD 70 d
LD H, n	26 n	LD (IY+d), C	FD 71 d
LD H, (HL)	66	LD (IY+d), D	FD 72 d
LD H, (IX+d)	DD 66 d	LD (IY+d), E	FD 73 d
LD H, (IY+d)	FD 66 d	LD (IY+d), H	FD 74 d
LD HL, nn	21 n n	LD (IY+d), L	FD 75 d
LD HL, (nn)	2A n n	LD (IY+d), n	FD 36 d n
LD I, A	ED 47	LD (nn), A	32 n n
LD IX, nn	DD 21 n n	LD (nn), BC	ED 43 n n
LD IX, (nn)	DD 2A n n	LD (nn), DE	ED 53 n n
LD IY, nn	FD 21 n n	LD (nn), HL	22 n n
LD IY, (nn)	FD 2A n n	LD (nn), IX	DD 22 n n
LD L, A	6F	LD (nn), IY	FD 22 n n
LD L, B	68	LD (nn), SP	ED 73 n n
LD L, C	69	LDD	ED A8
LD L, D	6A	LDDR	ED B8
LD L, E	6B	LDI	ED A0
LD L, H	6C	LDIR	ED B0
LD L, L	6D	NEG	ED 44
LD L, n	2E n	NOP	00
LD L, (HL)	6E	OR A	B7
LD L, (IX+d)	DD 6E d	OR B	B0
LD L, (IY+d)	FD 6E d	OR C	B1
LD R, A	ED 4F	OR D	B2
LD SP, HL	F9	OR E	B3
LD SP, IX	DD F9	OR H	B4
LD SP, IY	FD F9	OR L	B5
LD SP, nn	31 n n	OR n	F6 n

OR (HL)	B6		RES 2,B	CB 90
OR (IX+d)	DD B6	d	RES 2,C	CB 91
OR (IY+d)	FD B6	d	RES 2,D	CB 92
OTDR	ED 8B		RES 2,E	CB 93
OTIR	ED B3		RES 2,H	CB 94
OUT (C),A	ED 79		RES 2,L	CB 95
OUT (C),B	ED 41		RES 2,(HL)	CB 96
OUT (C),C	ED 49		RES 2,(IX+d)	DD CB d 86
OUT (C),D	ED 51		RES 2,(IY+d)	FD CB d 86
OUT (C),E	ED 59		RES 3,A	CB 9F
OUT (C),H	ED 61		RES 3,B	CB 98
OUT (C),L	ED 69		RES 3,C	CB 99
OUT (n),A	D3 n		RES 3,D	CB 9A
OUTD	ED AB		RES 3,E	CB 9B
OUTI	ED A3		RES 3,H	CB 9C
POP AF	F1		RES 3,L	CB 9D
POP BC	C1		RES 3,(HL)	CB 9E
POP DE	D1		RES 3,(IX+d)	DD CB d 9E
POP HL	E1		RES 3,(IY+d)	FD CB d 9E
POP IX	DD E1		RES 4,A	CB A7
POP IY	FD E1		RES 4,B	CB A8
PUSH AF	F5		RES 4,C	CB A1
PUSH BC	C5		RES 4,D	CB A2
PUSH DE	D5		RES 4,E	CB A3
PUSH HL	E5		RES 4,H	CB A4
PUSH IX	DD E5		RES 4,L	CB A5
PUSH IY	FD E5		RES 4,(HL)	CB A6
RES 0,A	CB 87		RES 4,(IX+d)	DD CB d A6
RES 0,B	CB 88		RES 4,(IY+d)	FD CB d A6
RES 0,C	CB 81		RES 5,A	CB AF
RES 0,D	CB 82		RES 5,B	CB A8
RES 0,E	CB 83		RES 5,C	CB A9
RES 0,H	CB 84		RES 5,D	CB AA
RES 0,L	CB 85		RES 5,E	CB AB
RES 0,(HL)	CB 86		RES 5,H	CB AC
RES 0,(IX+d)	DD CB d 86		RES 5,L	CB AD
RES 0,(IY+d)	FD CB d 86		RES 5,(HL)	CB AE
RES 1,A	CB 8F		RES 5,(IX+d)	DD CB d AE
RES 1,B	CB 88		RES 5,(IY+d)	FD CB d AE
RES 1,C	CB 89		RES 6,A	CB B7
RES 1,D	CB 8A		RES 6,B	CB B8
RES 1,E	CB 8B		RES 6,C	CB B1
RES 1,H	CB 8C		RES 6,D	CB B2
RES 1,L	CB 8D		RES 6,E	CB B3
RES 1,(HL)	CB 8E		RES 6,H	CB B4
RES 1,(IX+d)	DD CB d 8E		RES 6,L	CB B5
RES 1,(IY+d)	FD CB d 8E		RES 6,(HL)	CB B6
RES 2,A	CB 97		RES 6,(IX+d)	DD CB d B6

RES 6, (IY+d)	FD CB d BE	RR D	CB 1A
RES 7, A	CB BF	RR E	CB 1B
RES 7, B	CB B8	RR H	CB 1C
RES 7, C	CB B9	RR L	CB 1D
RES 7, D	CB BA	RR (HL)	CB 1E
RES 7, E	CB BB	RR (IX+d)	DD CB d 1E
RES 7, H	CB BC	RR (IY+d)	FD CB d 1E
RES 7, L	CB BD	RRA	1F
RES 7, (HL)	CB BE	RRC A	CB 0F
RES 7, (IX+d)	DD CB d BE	RRC B	CB 08
RES 7, (IY+d)	FD CB d BE	RRC C	CB 09
RET	C9	RRC D	CB 0A
RET C	D8	RRC E	CB 0B
RET M	F8	RRC H	CB 0C
RET NC	D0	RRC L	CB 0D
RET NZ	C0	RRC (HL)	CB 0E
RET P	F0	RRC (IX+d)	DD CB d 0E
RET PE	E8	RRC (IY+d)	FD CB d 0E
RET PO	E0	RRCA	0F
RET Z	C8	RRD	ED 67
RETI	ED 4D	RST #00	C7
RETN	ED 45	RST #08	CF
RL A	CB 17	RST #10	D7
RL B	CB 10	RST #18	DF
RL C	CB 11	RST #20	E7
RL D	CB 12	RST #28	EF
RL E	CB 13	RST #30	F7
RL H	CB 14	RST #38	FF
RL L	CB 15	SBC A, A	9F
RL (HL)	CB 16	SBC A, B	98
RL (IX+d)	DD CB d 16	SBC A, C	99
RL (IY+d)	FD CB d 16	SBC A, D	9A
RLA	17	SBC A, E	9B
RLC A	CB 07	SBC A, H	9C
RLC B	CB 00	SBC A, L	9D
RLC C	CB 01	SBC A, n	DE n
RLC D	CB 02	SBC A, (HL)	9E
RLC E	CB 03	SBC A, (IX+d)	DD 9E d
RLC H	CB 04	SBC A, (IY+d)	FD 9E d
RLC L	CB 05	SBC HL, BC	ED 42
RLC (HL)	CB 06	SBC HL, DE	ED 52
RLC (IX+d)	DD CB d 06	SBC HL, HL	ED 62
RLC (IY+d)	FD CB d 06	SBC HL, SP	ED 72
RLCA	07	SCF	37
RLD	ED 6F	SET 0, A	CB 07
RR A	CB 1F	SET 0, B	CB 08
RR B	CB 18	SET 0, C	CB 01
RR C	CB 19	SET 0, D	CB 02



SET 0,E	CB C3	SET 5,C	CB E9
SET 0,H	CB C4	SET 5,D	CB EA
SET 0,L	CB C5	SET 5,E	CB EB
SET 0,(HL)	CB C6	SET 5,H	CB EC
SET 0,(IX+d)	DD CB d C6	SET 5,L	CB ED
SET 0,(IY+d)	FD CB d C6	SET 5,(HL)	CB EE
SET 1,A	CB CF	SET 5,(IX+d)	DD CB d EE
SET 1,B	CB C8	SET 5,(IY+d)	FD CB d EE
SET 1,C	CB C9	SET 6,A	CB F7
SET 1,D	CB CA	SET 6,B	CB F8
SET 1,E	CB CB	SET 6,C	CB F9
SET 1,H	CB CC	SET 6,D	CB FA
SET 1,L	CB CD	SET 6,E	CB FB
SET 1,(HL)	CB CE	SET 6,H	CB FC
SET 1,(IX+d)	DD CB d CE	SET 6,L	CB FD
SET 1,(IY+d)	FD CB d CE	SET 6,(HL)	CB FE
SET 2,A	CB D7	SET 6,(IX+d)	DD CB d FE
SET 2,B	CB D8	SET 6,(IY+d)	FD CB d FE
SET 2,C	CB D9	SET 7,A	CB FF
SET 2,D	CB DA	SET 7,B	CB 00
SET 2,E	CB DB	SET 7,C	CB 01
SET 2,H	CB DC	SET 7,D	CB 02
SET 2,L	CB DD	SET 7,E	CB 03
SET 2,(HL)	CB DE	SET 7,H	CB 04
SET 2,(IX+d)	DD CB d DE	SET 7,L	CB 05
SET 2,(IY+d)	FD CB d DE	SET 7,(HL)	CB 06
SET 3,A	CB DF	SET 7,(IX+d)	DD CB d 06
SET 3,B	CB D8	SET 7,(IY+d)	FD CB d 06
SET 3,C	CB D9	SLA A	CB 07
SET 3,D	CB DA	SLA B	CB 08
SET 3,E	CB DB	SLA C	CB 09
SET 3,H	CB DC	SLA D	CB 0A
SET 3,L	CB DD	SLA E	CB 0B
SET 3,(HL)	CB DE	SLA H	CB 0C
SET 3,(IX+d)	DD CB d DE	SLA L	CB 0D
SET 3,(IY+d)	FD CB d DE	SLA (HL)	CB 0E
SET 4,A	CB E7	SLA (IX+d)	DD CB d 0E
SET 4,B	CB E8	SLA (IY+d)	FD CB d 0E
SET 4,C	CB E9	SRA A	CB 0F
SET 4,D	CB EA	SRA B	CB 10
SET 4,E	CB EB	SRA C	CB 11
SET 4,H	CB EC	SRA D	CB 12
SET 4,L	CB ED	SRA E	CB 13
SET 4,(HL)	CB EE	SRA H	CB 14
SET 4,(IX+d)	DD CB d EE	SRA L	CB 15
SET 4,(IY+d)	FD CB d EE	SRA (HL)	CB 16
SET 5,A	CB EF	SRA (IX+d)	DD CB d 16
SET 5,B	CB E8	SRA (IY+d)	FD CB d 16

SRL A	CB 3F
SRL B	CB 38
SRL C	CB 38
SRL D	CB 3A
SRL E	CB 3B
SRL H	CB 3C
SRL L	CB 3D
SRL (HL)	CB 3E
SRL (IX+d)	DD CB d 3E
SRL (IY+d)	FD CB d 3E
SUB A	97
SUB B	98
SUB C	91
SUB D	92
SUB E	93
SUB H	94
SUB L	95
SUB n	D6 n
SUB (HL)	96
SUB (IX+d)	DD 96 d
SUB (IY+d)	FD 96 d
XOR A	AF
XOR B	A8
XOR C	A9
XOR D	AA
XOR E	AB
XOR H	AC
XOR L	AD
XOR n	EE n
XOR (HL)	AE
XOR (IX+d)	DD AE d
XOR (IY+d)	FD AE d

# Příloha C

Možné kombinace instrukce LD 1.,2.

2. \ 1. (BC DE HL IX+d IY+d nn) A B C D E H L I R IX IY SP BC DE HL																				
(BC)	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.
(DE)	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.
(HL)	.	.	.	.	.	.	X	X	X	X	X	X	X	.	.	.	.	.	.	.
(IX+d)	.	.	.	.	.	.	X	X	X	X	X	X	X	.	.	.	.	.	.	.
(IY+d)	.	.	.	.	.	.	X	X	X	X	X	X	X	.	.	.	.	.	.	.
(nn)	.	.	.	.	.	.	X	.	.	.	.	.	.	.	X	X	X	X	X	X
A	X	X	X	X	X	X	X	X	X	X	X	X	X	X	.	.	.	.	.	.
B	.	.	X	X	X	.	X	X	X	X	X	X	X	.	.	.	.	.	.	.
C	.	.	X	X	X	.	X	X	X	X	X	X	X	.	.	.	.	.	.	.
D	.	.	X	X	X	.	X	X	X	X	X	X	X	.	.	.	.	.	.	.
E	.	.	X	X	X	.	X	X	X	X	X	X	X	.	.	.	.	.	.	.
H	.	.	X	X	X	.	X	X	X	X	X	X	X	.	.	.	.	.	.	.
L	.	.	X	X	X	.	X	X	X	X	X	X	X	.	.	.	.	.	.	.
I	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.
R	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.
IX	.	.	X	X	X	.	X	X	X	X	X	X	X	.	.	.	.	.	.	.
IY	.	.	.	.	.	X	.	.	.	.	.	.	.	.	X	.	.	.	.	.
SP	.	.	.	.	.	X	.	.	.	.	.	.	.	.	X	.	.	.	.	.
BC	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.
DE	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.
HL	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.
nn	.	.	.	.	.	.	.	.	.	.	.	.	.	X	X	X	X	X	X	X

# Příloha D

Doby provádění strojních instrukcí.

Instrukce	cykly	Instrukce	cykly	Instrukce	cykly
ADC A,r	4	EX (SP),HL	19	LD (HL),r	7
ADC A,n	7	EX (SP),I.	23	LD (HL),n	10
ADC A,(HL)	7	EXX	4	LD (I.+d),r	19
ADC A,(I.+d)	19	HALT	4	LD (I.+d),n	19
ADC A,HL,xx	15	IM c	8	LD (nn),A	13
ADD A,r	4	IN A,(n)	11	LD (nn),xx	20
ADD A,n	7	IN r,(C)	12	LD (nn),HL	16
ADD A,(HL)	7	INC r	4	LD (nn),I.	20
ADD A,(I.+d)	19	INC (HL)	11	LDD	16
ADD HL,xx	11	INC (I.+d)	23	LDDR	20 16
ADD I.,xx	15	INC xx	6	LDI	16
AND r	4	INC I.	10	LDIR	20 16
AND n	7	IND	16	NEG	8
AND (HL)	7	INDR	21 16	NOP	4
AND (I.+d)	19	INI	16	OR r	4
BIT b,r	8	INIR	21 16	OR n	7
BIT b,(HL)	12	JP nn	10	OR (HL)	7
BIT b,(I.+d)	20	JP (HL)	4	OR (I.+d)	19
CALL nn	17	JP (I.)	8	OTDR	21 16
CALL cc,nn	17 10	JP cc,nn	10	OTIR	21 16
CCF	4	JR nn	12	OUT (C),r	12
CP r	4	JR c,nn	12 7	OUT (n),A	11
CP n	7	LD r,r'	4	OUTD	16
CP (HL)	7	LD r,n	7	OUTI	16
CP (I.+d)	19	LD r,(HL)	7	POP AF	10
CPD	16	LD r,(I.+d)	19	POP rr	10
CPDR	21 16	LD A,I	9	POP I.	15
CPI	16	LD A,R	9	PUSH AF	11
CPIR	21 16	LD A,(BC)	7	PUSH rr	11
CPL	4	LD A,(DE)	7	PUSH I.	15
DAA	4	LD A,(nn)	13	RES b,r	8
DEC r	4	LD I,A	9	RES b,(HL)	15
DEC (HL)	11	LD R,A	9	RES b,(I.+d)	23
DEC (I.+d)	23	LD xx,nn	10	RET	10
DEC xx	6	LD xx,(nn)	20	RET cc	11 5
DEC I.	10	LD HL,(nn)	16	RETI	14
DI	4	LD I.,nn	14	RETN	14
DJNZ n	13 8	LD I.,(nn)	20	RL r	8
EI	4	LD SP,HL	6	RL (HL)	15
EX AF,AF'	4	LD SP,I.	10	RL (I.+d)	23
EX DE,HL	4	LD (rr),A	7	RLA	4

Instrukce	cykly	poznámky
RLC r	8	b = 0,1,2,3,4,5,6,7
RLC (HL)	15	c = C,NC,Z,NZ
RLC (I.+d)	23	cc = C,NC,Z,NZ,M,P,PO,PE
RLCA	4	d = 0 - 255
RLD	18	I. = IX,IY
RR r	8	n = 0 - 255
RR (HL)	15	nn = 0 - 65535
RR (I.+d)	23	r = A,B,C,D,E,H,L
RRA	4	rr = BC,DE,HL
RRC r	8	x = #00,#08,#10,#18,#20,#28,#30,#36
RRC (HL)	15	xx = BC,DE,SP
RRC (I.+d)	23	
RRCA	4	
RRD	18	
RST x	11	U některých instrukcí jsou uvedeny dvě hodnoty trvání instrukce. Je to proto, že instrukce má různou délku když je a když není podmínka splněna. U instrukce CALL, JR, RET je čas větší, když je podmínka pro vykonání splněna než když není, a u instrukcí CPDR, CPIR, DJNZ, INIR, INDR, LDIR, OTDR a OTIR je čas větší, když podmínka pro ukončení instrukce není splněna.
SBC A,r	4	
SBC A,n	7	
SBC A,(HL)	7	
SBC A,(I.+d)	19	
SBC HL,rr	15	
SBC HL,SP	15	
SCF	4	
SET b,r	8	
SET b,(HL)	15	
SET b,(I.+d)	23	
SLA r	8	
SLA (HL)	15	
SLA (I.+d)	23	
SRA r	8	
SRA (HL)	15	
SRA (I.+d)	23	
SRL r	8	
SRL (HL)	15	
SRL (I.+d)	23	
SUB r	4	
SUB n	7	
SUB (HL)	7	
SUB (I.+d)	19	
XOR r	4	
XOR n	7	
XOR (HL)	7	
XOR (I.+d)	19	

Při kmitočtu oscilátoru 3,5 MHz je délka jednoho cyklu 0.2857us.

Opravit a doplnit: Bohuslav Labaj, Zápotockého 789/9, Trinec 6

Natištěno pro ZO SVAZARM Karolinka

březen 1989