MIKRO BÁZE

1989 8

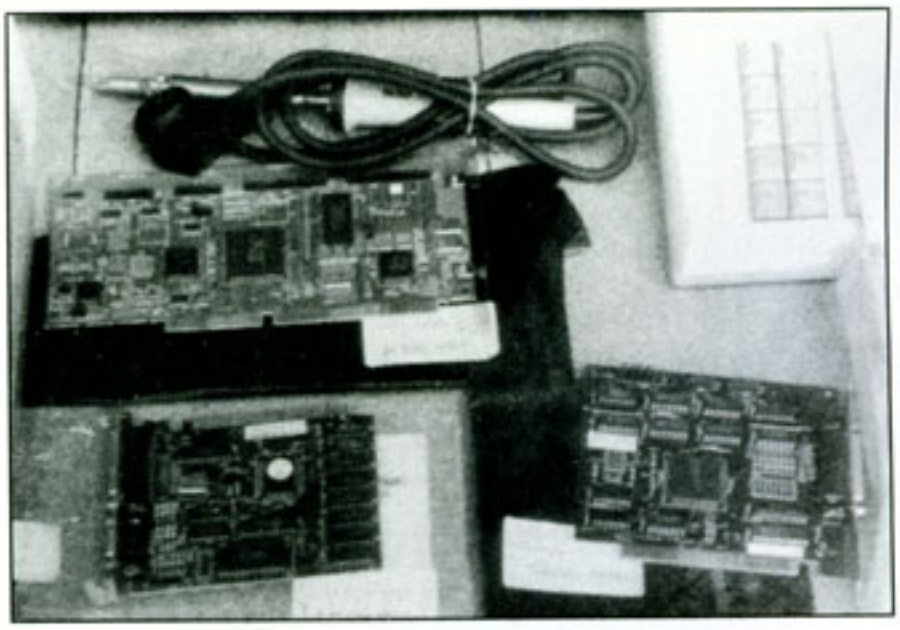
technický zpravodaj pro zájemce o mikropočítače

Cena 12 Kčs

Z podzimní návštěvy Budapešti přinášíme několik pohledů do obchodů a výkladních skříní. Srdce počítačových nadšenců jistě nad nimi zaplesá. A ceny? Slušné - například za řadič WD 1793 zaplatíte něco mezi 900 a 1500 forintů, za EPROM 27128 tak 300 až 500 forintů. Jak kde. Finální výrobky jsou ovšem dost drahé. Pokud na to máte, můžete však dostat téměř cokoliv.

Během krátké návštěvy jsme navázali kontakt s redakcí tamnějšího radioamatérského časopisu RÁDIÓTECHNIKA a dohodli se na vzájemné spolupráci. Výsledkem by měla být vzájemná výměna nejzajímavějších článků a informací.

Text i foto: D. Meca



MIKRO BÁZE

1989/8

OBSAH

Technický zpravodaj Svazarmu pro zájemce o mikropočítače. Vydává 602. ZO Svazarmu ve spolupráci s redakcí časopisu Amatérské radio. Povoleno ÚVTEI pod evidenčním číslem 87 007. Sestavil vedoucí redaktor Daniel Meca. Obálka ak. grafik Jiří Blažek a Daniel Meca, grafická úprava Zdeňka Perglerová. Sekretářka redakce Zdeňka Válková. Redakční rada: Petr Horský, ing. Jan Klabal, ing. Petr Kratochvíl, Josef Kroupa, Rudolf Mach, Daniel Meca, ing. Alois Myslík, ing. Josef Truxa. Za původnost a správnost příspěvků ručí autoři. Ročně vyjde 10 čísel. Cena výtisku 12 Kčs podle ČCÚ a SCÚ č. 1030/202/86. Roční předplatné 120 Kčs. Objednávky přijímá a zpravodaj rozšiřuje 602. ZO Svazarmu, Wintrova 8, 160 41 Praha 6.

V dopisech, kde nabízíte své příspěvky, se velmi často objevuje otázka - jak má vlastně rukopis vypadat. Normalizovaná úprava rukopisů je stanovena příslušnou ČSN. Její požadavky jsou poměrně přísné a vycházejí z klasického způsobu úpravy a zpracování textu. My však jsme počítačový časopis a tak i zpracování textu probíhá na počítači. Proto i naše požadavky na rukopisy jsou jiné.

Doporučená úprava rukopisů pro Mikrobázi.

Autorský arch (AA) = 20 normalizovaných rukopisných stránek.
Rukopisná stránka = 30 řádků po 60 úhozech = 1800 znaků.
Tisková stránka Mikrobáze je pokryta asi čtyřmi rukopisnými.

Rukopisy do Mikrobáze je možno dodat jednak v klasické úpravě, tj. psané na psacím stroji (30x60zn.). Přijmeme však i rukopisy psané počítačovou tiskárnou v libovolném formátu, podmínkou je však dobrá čitelnost. V takovém případě je vhodné (i když ne nutné) psát na šíři 64 znaků. Slova na konci řádku raději nedělte. V tabulkách a pevně formátovaných výpisech se pokuste vtěsnat do 50 znaků na řádek, ne však za cenu srozumitelnosti a přehlednosti. Pokud si přejete změnu písma, vyznačte ji před i za změnou nejlépe takto:

>t< pro tučné písmo
>k< pro kurzívu
>p< pro podtržené
>tp< pro tučné podtržené
>kp< pro kurzívu podtrženou

Příklad:
Toto je *psáno* na ukázku změny **písma** v *textu*.
V rukopisu vyznačte takto:
>t<Toto>t< je >k<psáno>k< na >p<ukázku>p< změny
>tp<písma>tp< v >kp<textu>kp<.

Navíc je vhodné každou změnu na výpisu podtrhnout barevnou tužkou.

Budeme však raději, když nám texty (zvláště delší) dodáte na disketě, nebo na kazetě. Nejenže nám tím ušetříte práci s přepisem, ale vyloučí se vznik chyb (je to důležité hlavně ve výpisech programů). Formát záznamu na kazetě musí odpovídat ZX Spectru a musí být použito editoru Tasword CS, D-text, nebo kompatibilního. Může to být i záznam textu z WordStaru pod CP/M, pořízený programem MSAVE. Je třeba uvést typ editoru, případně způsob kódování diakritiky.

Disketa může být 3", 3½", nebo 5¼". Formát záznamu musí odpovídat Beta-Disku, IBM PC 360KB, nebo CP/M (CPC 6128 - Amstrad data, Robotron 1715, TNS, Sharp, Spectrum, PMD a pod.). Jiné formáty jen po dohodě. Formát uveďte na štítku. Bez problému bude ASCII výpis, kódovaný podle KOI8-čs2, LATIN2, nebo KEYBCS2 (čeština Kamenických). Pokud používáte editor TEXT 602, případně WordStar, bude nejlépe dodat text ve vnitřním formátu těchto editorů. Pak samozřejmě není nutno zvlášť vyznačovat typy písma.

Disketu i kazetu nezapomeňte čitelně označit jménem a adresou, aby mohly být po přepsání vráceny.

A ještě něco k obrázkům. Velice uvítáme, když své příspěvky doplníte obrázky, grafy a fotografiemi. Obrázky stačí načrtnout tužkou, necháme si je překreslit. Pokud hodláte dodat definitivní předlohu k obrázku, nakreslete ji tuší na pauzovací papír v měřítku asi 2:1. Nepoužívejte příliš tenké čáry ani příliš drobné popisy. Řiďte se podle už uveřejněných obrázků. Výpisy programů raději dodejte na kazetě, či disketě. Obrázky a grafy z počítačové tiskárny musí být dostatečně výrazné, jinak nám je v tiskárně nezreprodukují. Pozor, modrá barva se obtížně reprodukuje.

Černobílé fotografie musí být lesklé, dostatečně kontrastní, formátu asi 13x18 cm, jen výjimečně menší. Barevné fotografie na obálku se zhotovují z dodaných diapozitivů, případně z dostatečně kvalitních větších neprůhledných předloh. Nutnou podmínkou je dokonalá ostrost a dobré barevné podání.
>Kvůli proplacení honorářů je nutno uvést rodné číslo a ČOP.<

Daniel Meca

CO MOŽNÁ NEVÍTE

O MIKROPROCESORECH Z 80 A 18085

Jakub Vaněk

1. Rozdíly v činnosti

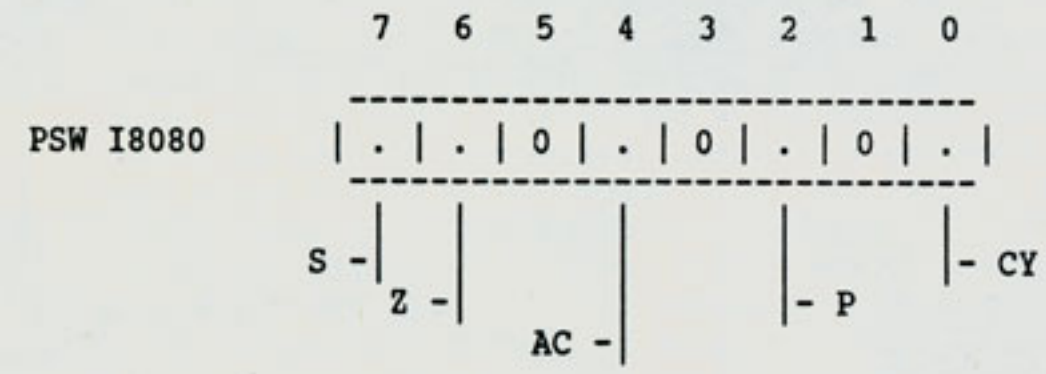
Mikroprocesory Z80 a I8085 vycházejí z původního procesoru I8080. Je obecně známo, že instrukční soubor Z80 i I8085 obsahuje všechny instrukce I8080. Ne každý si však uvědomuje, že ve způsobu provádění některých těchto společných instrukcí jednotlivými procesory jsou jisté, někdy však nezanedbatelné rozdíly.

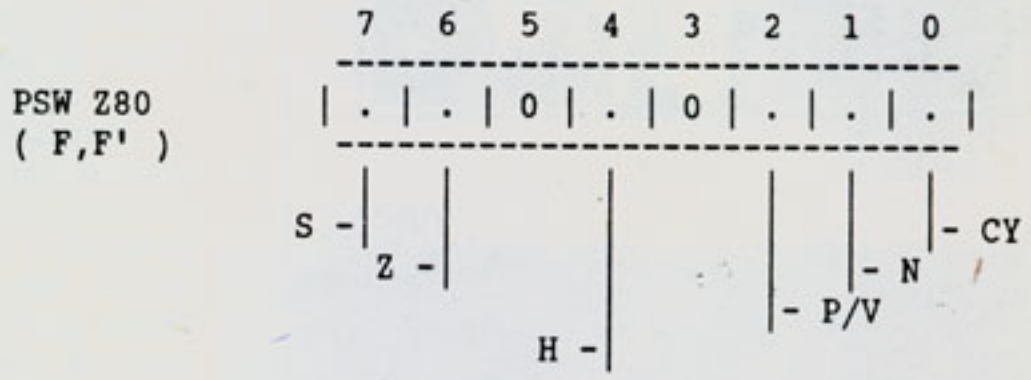
První rozdíl je v různé době trvání jednotlivých instrukcí. To je důležité vědět při psaní časových smyček. Přehled všech časově rozdílných instrukcí je uveden v následující tabulce:

symbol instrukce		počet T na instrukci		
I8080/85	Z80	I8080	I8085	Z80
MOV r1,r2	LD r1,r2	5	4	4
SPHL	LD SP,HL	5	6	6
PUSH qq	PUSH qq	11	12	11
XTHL	EX SP,HL	18	16	19
INR r	INC r	5	4	4
DCR r	DEC r	5	4	4
INR M	INC (HL)	10	10	11
DCR M	DEC (HL)	10	10	11
DAD dd	ADD HL,dd	10	10	11
INX dd	INC dd	5	6	6
DCX dd	DEC dd	5	6	6
Jcc nn	JP cc,nn	10	10/7	10
PCHL	JP (HL)	5	6	4
CALL nn	CALL nn	17	18	17
Ccc nn	CALL cc,nn	17	18/9	17
RST p	RST p	11	12	11
Rcc	RET cc	11/5	12/6	11/5
HLT	HALT	7	5	4

T je počet taktů hodin procesoru.

Další rozdíl je ve významu jednotlivých bitů stavového registru PSW. U CPU I8085 je pouze rozšířen, u Z80 je rozšířen a je pozměněn význam bitu 'P'. Rozdíly jsou patrny z následujícího obrázku:





Význam jednotlivých bitů :

S ... znaménko (sign)
Z ... nula (zero)
CY ... přenos (carry)
P ... parita (parity)
AC ... pomocný přenos (auxiliary carry)

X5 ... přeplnění INX,DCX
V ... přeběh (overflow)

H ... pomocný přenos (half-carry)
N ... sčítání/odečítání (subtract flag)
P/V .. parita/přeběh (parity/overflow)

U CPU I8085 jsou indikatory S,Z,CY,P a AC nastavovány stejně jako u I8080. Indikátory X5 a V je vhodně doplňují. Indikátor X5 je nastavován při přeplnění či podtečení v instrukcích INX a DCX. Indikátor V je vlastně aritmetické přeplnění.

U Z80 má indikátor P/V dvojí funkci. Po logických operacích indikuje paritu a po aritmetických operacích aritmetické přeplnění. Ve většině případů je tento rozdíl nepodstatny. Indikátor H je totožný s AC, a indikátor N umožňuje dekadickou korekci po sčítání i po odčítání.

2. Maličkosti ve strojovém kódu Z80

Pro lepší a efektivnější využívání CPU Z80 je dobré si uvědomit jak provádí některé instrukce. Všimněme si třeba opakovaných instrukcí. Jsou to instrukce LDIR a LDDR pro blokový přesun, CPIR a CPDR pro blokové vyhledávání a OTIR, OTDR, INIR, INDR pro blokové I/O operace. CPU při vykonávání těchto instrukcí provádí v mikroprogramu při dokončení jednoho průchodu instrukce relativní skok PC=PC-2. Dostává se tedy opět na začátek instrukce a ta je pokaždé znova čtena z operační paměti CPU. Tím je umožněno přerušení programu během provádění blokových instrukcí.

Adresa pro relativní skok se vypočítává v ALU CPU a tento výpočet trvá 5 taktů hodin. Je to zvlášté patrné u podmíněných relativních skoku (12/7 taktů hodin). Z tohoto vyplývá, že při ukončení opakované instrukce je poslední průchod o 5 taktů hodin kratší tj. 16 taktů místo 21 taktů při jednotlivých dílčích přenosech.

Ne každý také ví, že je možno programově zjistit, zda byla provedena instrukce DI či EI. Při použití instrukcí LD A,R a LD A,I je flag P/V na-

staven podle stavu registru IFF. Pokud se to zjišťuje v obslužné rutině pro NMI, udává flag P/V stav před příchodem NMI.

Při provádění instrukce EI je přerušení povoleno až po instrukci následující EI. Typické použití je na konci obslužné rutiny přerušení, kde sled instrukcí je EI a RET, přičemž přerušení je povoleno až po návratu.

A ještě něco o časování sběrnicových cyklů CPU Z80. Při dekódováni shiftu instrukce následuje opět M1. Po dvou shiftech však již není cykl M1 indikován. Cykl M1 bez Wait taktů trvá 4 takty systémových hodin. Následující pamětové cykly 3 takty. Cykly I/O trvají 4 takty hodin.

3. Neznámé instrukce CPU I8085

Firma Intel uvádí u tohoto CPU dvě nové instrukce oproti CPU I8080, a to instrukce RIM a SIM. Jsou určeny pro práci s maskou rozšířeného přerušovacího systému a sériového portu. Byly popsány například v [2].

Přesto je však v instrukčním kódu ještě 10 volných pozic. Jsou zde umístěmy velice užitečné instrukce, popsané v [3] a [4]. Jak již bylo řečeno, byl spolu s těmito instrukcemi doplněn i PSW. Následující tabulka ukazuje jejich funkci, včetně jejich vlivu na PSW. Mnemotechnické zkratky vycházejí ze zvyklostí tvoření názvů Intel a jsou převzaty z [3].

Symbol instrukce	Popis operace	C Z P S AC V X5	B/T	op.c.
LDHI d8	DE <- [HL+d8]	- - - - - - -	2/10	28h
LDSI d8	DE <- [SP+d8]	- - - - - - -	2/10	38h
SHLX	[DE] <- HL	- - - - - - -	1/10	d9h
LHLX	HL <- [DE]	- - - - - - -	1/10	edh
DSUB	HL <- HL-BC	* * * * * * *	1/10	08h
ARHL	ar.sh.r.HL	* - - - - - -	1/7	10h
RDEL	rot.l. DE	* - - - - * -	1/10	18h
RSTV	if V -> PC=40h else PC=PC+1	- - - - - - -	1/12 /6	cbh
JNX5 nn	if NX5 -> PC=nn else PC=PC+3	- - - - - - -	3/10 /7	ddh
JX5 nn	if X5 -> PC=nn else PC=PC+3	- - - - - - -	3/10	fdh

T značí počet taktů hodin procesoru a B počet bytů istrukce.

Literatura :

[1] Dědina, B.- Valášek, P.: Mikroprocesory a mikropočítače; SNTL 1983, str. 53-71, 110-132.

[2] Fadrhons, Jan : Mikroprocesory Z80, 8085 a NSC 800; Sdělovací technika č. 7,8/1982, str. 259-261, 301-304.

[3] Dehnhardi, W.: Unbekannte 8085 - Instruktione; Elektronik č. 15/1978.

[4] Neznámé instrukce mikroprocesoru 8085; Sdělovací technika č. 9/1985.

****

ODSTRANĚNÍ ROZTRHANÉHO TISKU U TISKÁRNY BT-100

Ing. Karel Hrazdil

Po dlouhém shánění jsem zakoupil tiskárnu BT-100 (moji dva kamarádi také), pak jsem několik dní vyráběl destičku plošných spojů, drátoval, letoval, zapojil interface a tiskárnu, a ono to fungovalo... (kamarádům také). Po napsání asi 50 stránek A4 mi začala tiskárna roztrhávat tisk krátkých řádků (tak asi do 40 znaků) a kopií obrazovky po jednotlivých linkách (kamarádům také). Obrátil jsem se na Teslu Přelouč - odpověď zněla: "Používejte náš obslužný program!", ale ani tato metoda neměla žádaný účinek. Potom jsem si vzpomněl na studentská léta, vzal tužku, papír a začal počítat. Setrvačnost rotujících součástek pohonu pojezdu vozíku měla několikrát větší hodnotu, než hybnost vozíku. Potom už bylo řešení na dosah ruky:

a) vystřihl jsem z 3 mm plsti kroužek o vnějším průměru 10 mm a vnitřním průměru 3 mm

b) odšrouboval jsem dno tiskárny, potom 4 vnitřní šrouby a sejmul kryt tiskárny

c) povolil jsem 2 červíky unášeče lanka pojezdu hlavičky a to ty blíže k ložisku (bílému příčníku)

d) unášeč jsem opatrně sejmul a na vyčnívající hřídel jsem nasadil plstěný kroužek; potom jsem unášeč vrátil na původní místo

e) unášeč jsem opatrně přitlačil proti dělícímu kotouči tak, abych plst stlačil na cca 2 mm a červíky jsem opět přitáhl

f) opět jsem tiskárnu zakrytoval, zapojil ... a ono to opět tisklo, jak by mělo. Opravu jsem dvakrát zopakoval a i moji kamarádi měli po starostech

Montáž třecího kroužku v ceně zanedbatelné, trvající ani ne půl hodiny, tedy odstranila závadu, shodně se opakující na třech tiskárnách, a to zcela uspokojivě.

%%%

Pozn. red.: Na stejné téma nám poslal příspěvek i Michal Bechyně, technik prodejny Mikropočítače ve Spálené ulici v Praze. Jeho řešení je snad ještě jednodušší a navíc umožňuje zvýšit rychlost tisku. Opublikujeme ho co nejdříve.

DŘU, DŘEŠ, DŘEME... CÉČKO /8/

Pole číselná

Syntakticky mají mnoho společného s poli znakovými. Hlavní rozdíl mezi oběma typy polí je v tom, že číselné prvky pole obsahují vždy stejný počet bajtů (podle konkrétní implementace jazyka - typ int bývá dvoubajtový, float int čtyřbajtový atd.).

Pole můžeme definovat s uvedením počtu budoucích prvků v hranatých závorkách:

```
int čísla[5];
```

nebo současně inicializovat jeho obsah:

```
int čísla[]={3,66,123,4,5};
```

Povšimněte si, že tady v hranatých závorkách nic není. Když pole inicializujeme, kompilátor si sám spočítá, kolik prvků v poli je, a uloží je do paměti. V definici však musíme počet prvků uvést, aby pro ně kompilátor mohl předem vyhradit místo v paměti (v horní definici to bude 5*2=10 bajtů). Můžeme ale rovněž definovat přidělení paměti poli a zároveň inicializovat menší počet jeho prvků, např.:

```
int čísla[5]={18,3,25};
```

První tři prvky budou mít uvedené hodnoty, do zbývajících některé kompilátory uloží nuly, jiné tam nechají "smetí".

Inicializovat pole nelze v automatické paměti (uvnitř těla funkce) - tam je můžeme jen definovat a pak mu programově přidělit obsah. **Přímá inicializace pole v těle funkce je možná jen při použití statické paměti:**

```
static int pole[]={22,3,55};
```

Externí pole můžeme definovat i inicializovat.

Nyní si opět pohrajeme s ukazateli na pole i ukazateli pole na pole ukazatelů. Jistě si ještě vzpomenete na to, jak dostaneme adresu uložení proměnné v paměti pomocí unárního operátoru &:

```
&prom
```

Pole můžeme pojímat jako pole proměnných. Adresu uložení první až třetí proměnné předchozího pole dostaneme operacemi:

```
&pole[0]      &pole[1]      &pole[2]
```

Adresu uložení pole dostaneme buď tak, jak je uvedeno na řádce zkraje (protože adresa prvního prvku je adresou pole), nebo přímo jménem pole. Čili pole==&pole[0]. Tím Céčko nabízí možnost zjednodušeného syntaktického zápisu pro písařský plezír.

Další nám napoví tento program:

```
int a=5;
main()
{static int pole[]={1,2,3};
fce(&a,&pole[1]);
printf("%u %u",&a,pole);}
fce(uka,ukpole)
int *uka,*ukpole;
{printf("%u %u %u %u %u %u\n",
        &uka,uka,*uka,&ukpole,ukpole,*ukpole);}
```

Výpis z Hisoftu ZX Spectra:

```
65520 65334 5 65518 65530 2
65534 65528
```

Vezmeme význam a obsahy adres, které nás eminentně zajímají, zdola nahoru:

65518/65519 jsou adresy uložení ukazatele **uka** - jeho obsah je adresou uložení proměnné a, tedy 65534.

65520/65521 jsou adresy uložení ukazatele **ukpole** - jeho obsah je adresou uložení prvku **pole(1)**, tedy 65530.

65528/65529 jsou adresy uložení prvku **pole[0]**

65530/65531 " **pole[1]**

65532/65533 " **pole[2]**

65534/65535 " proměnné **a** (výsledek operace &a). Na vyšší adrese leží nižší bajt čísla a, tedy 5, na nižší adrese (&a) bajt vyšší, což je 0.

Z přehledu je patrný i postup ukládání parametrů programu.

Funkci fce() byla předána adresa prvku pole[1], která se uložila do ukazatele ukpole. Proto žádost o výpis obsahu adres, na které tento ukazatel ukazuje (*ukpole), přinesla výsledek 2.

Podobně se do ukazatele **uka** uložila adresa uložení proměnné **a**, čehož důkazem je výpis čísla 5 při žádosti o sdělení obsahu adres (*uka), na které **uka** ukazuje.

Myslím, že víc nic k tomu netřeba. Snad jen jednu mnemotechnickou pomůcku. Jak už jsem uvedl, pro zjištění adresy běžné proměnné i prvku "čistého" pole (tedy nikoli "přes" pole ukazatelů) použijeme naprosto stejný postup:

```
&a      je adresováním totéž, co      &pole[x]
```

V duchu to berte tak, že obojí je prostá proměnná ležící na adrese, kterou zjistíme unárním operátorem & v poli všech proměnných (je "jedno", jestli se to týká prvku pole nebo běžné proměnné). Teprve při adresování pomocí ukazatelů (resp. polem ukazatelů) se situace při zjišťování adres prvků a jejich obsahů adekvátně mění. Přičemž ukazatel ale opět není ničím jiným, než jen a jen proměnnou v poli všech proměnných (samozřejmě v souvislosti s typem proměnné i třídou použité paměti).

U ukazatelů (ostatně jako u všech proměnných) je třeba dbát na to, abychom si jejich nevhodnou změnou při práci s adresovou aritmetikou nepřipravili nepříjemná překvapení.

Pokud jsem vám minule trochu zamotal hlavu v příkladu se čtyřmi různými definicemi řetězce "mikro", dosaďte si do výkladu předchozí odstavce,

a hned bude vše jasné. Ovšem i nadále trvám na tom, že název "čistého" pole není ukazatelem, jak je uvedeno v některých příručkách Céčka. O ukazateli se dá mluvit až tehdy, když definujeme pole ukazatelů na některý z možných typů, nebo když adresu "čistého" pole převedeme do ukazatele na toto pole (jako ve výše uvedeném programu).

Na závěr teoretizování kolem ukazatelů si nemůžeme odpustit pohled na pole ukazatelů na ukazatele na typ int.

```
main()
(static int **pole[]={0,16384}
printf("%u %u",**pole,*pole[1]);}
```

Výpis:

```
45043 0
```

To, že jsou zde adresy 0 a 16384 použity ve vztahu k ZX Spectru, nehraje v principu nijak důležitou roli.

Bližším prozkoumáním opět odhalíme logiku věci:

```
  pole=65359          &pole[1]=65361
 *pole=0               pole[1]=16384
**pole=45043          *pole[1]=0
```

Obdobný rozbor ukazatelů na ukazatele typu int je v závěru 6. pokračování, proto jen stručně. Deklarace *pole[] nám říká, že jde o pole ukazatelů na daný typ (viz minulé pokračování). Přidáním další hvězdičky definujeme pole ukazatelů na ukazatele na daný typ. Z toho je zřejmé, že tu jde o dvoustupňové odkazování. Důkazem je rozvoj uvedený v předchozích sloupcích. V prvním je ukazatel uložen na adresách 65359 a 65360. Odtud odkazuje na adresu 0. A obsahem adres 0 a 1 je číslo 45043. Pro lepší představu o vztazích mezi oběma sloupci:

```
  pole      je totéž, co      &pole[0]
 *pole           "             pole[0]
**pole           "            *pole[0]
```

Druhý prvek pole ukazatelů leží vedle prvního na adresách 65361 a 65362 (prvky s vyšším indexem se ukládají na vyšší adresy).

Obsah ukazatelů kteréhokoli z obou stupňů lze programově měnit. Tím jsou takové ukazatele přímo předurčeny pro práci s dynamickou pamětí, která velmi často mění svůj obsah i rozložení (adresy) prvků v ní uložených.

Syntaxe incializace vícerozměrných polí má svá pravidla. Např. obsah každé řádky dvourozměrného pole se píše do svorek, mezi sebou oddělených čárkami:

```
pole[2][3]={{1,2,3},{4,5,6}};
```

Zde je každá řádka pole naplněna. S ohledem na rozmístění prvků v paměti (viz níže) můžeme takto vyplněné pole inicializovat jednodušeji:

```
pole[2][3]={1,2,3,4,5,6};
```

Když budeme chtít určit jen první dva sloupce každé řádky, napíšeme:

```
pole[2][3]={{1,2},{4,5}};
```

Důležité je to, že první prvek za vnitřní levou svorkou je zároveň prvním prvkem dané řádky. Např. inicializace:

```
pole[2][3]={1,2,3,4};
```

je totéž, co:

```
pole[2][3]={{1,2,3},{4}};
```

V dalším zcela odhlédněte od termínů řádky a sloupce pole. Dvourozměrné pole pojímejte jako pole jednorozměrných polí, čili jako definovaný počet jednorozměrných polí, jejichž první prvky leží na adresách prvního sloupce dvourozměrného pole.

Adresu vícerozměrného pole předáváme jménem pole, resp. adresou jeho prvního prvku, tedy podobně jako u pole jednorozměrného. Příklady získání adresy uložení různých prvků dvourozměrného pole:

prvek		jeho adresa
pole[23][60]		&pole[23][60]
pole[12][0]		&pole[12][0]
	nebo	pole[12]
pole[0][0]		&pole[][]
	nebo	pole[0]
	nebo	pole

U dvourozměrného pole je adresou jeho začátku nejen název pole sám, ale i výraz **pole[0] jako adresa prvního jednorozměrného pole.** Pak např. **pole.1. je adresou druhého jednorozměrného pole.** Výraz &pole[12][0] v tabulce je shodný s pole[12], protože jde o první prvek 13. pole. Analogicky by tento rozvoj pokračoval při přidávání dalších dimenzí pole.

Při předávání adresy dvourozměrného pole volané funkci je nutné kompilátoru sdělit, jak má s polem nakládat, resp. jaké jsou jeho dimenze (prvky číselného pole nejsou zakončeny binární nulou jako u znakových polí). Např.:

```
main()
(...
static int pole[2][5]={{ inic.pole }};
fce(pole);
...)

fce(číspole)
int číspole[][5];
(...)
```

Kompilátoru jsme sdělili, že jde o "několik" jednorozměrných polí s 5 prvky. On už si sám vypočte, že tato pole budou dvě.

Adresování číselných polí

Prvky číselného pole daného typu mají stejnou délku, proto je s nimi v lecčem jednodušší práce než se znakovými poli, obsahujícími alfanumerické řetězce různé délky. Musíme však znát uložení prvků v paměti v kontextu s jejich číslováním (ve vztahu k jejich indexům). Ukážeme si to na dvourozměrném poli **int pole[4][2].** Uložení jeho prvků v paměti je:

pole[0][0] pole[0][1] pole[1][0] pole[1][1] atd.

Když tedy chceme využít adresovou aritmetiku, přeneseme napřed adresu pole (jeho prvního prvku) do ukazatele na toto pole (třeba *ukpole, kde

ukpole==&pole[0][0]==&pole[][]==pole[0]==pole

což je adresa prvního prvku dvourozměrného pole). Zvyšováním adresového ofsetu sledujeme prvky přesně podle jejich uložení v paměti:

```
ukpole  ==&pole[0][0]
ukpole+1==&pole[0][1]
ukpole+2==&pole[1][0]
ukpole+3==&pole[1][1]
ukpole+4==&pole[2][0] atd.
```

Adresovou aritmetikou Céčka se v paměti pohybujeme o uvedený ofset*2, protože jde o dvoubajtový int (skáčeme po dvou adresách). Tak se v poslední řádce k adrese **ukpole** přičte číslo 8 a výsledkem bude adresa uložení prvku **pole**[2][0]. Když budeme chtít hodnotu samotného prvku na této adrese, dostaneme ji buď s pomocí ukazatele příkazem ***(ukpole+4)** nebo přímo **pole**[2][0]. Tento prvek můžeme dostat i jen pomocí **pole**[2], kde je nula zamlčená. Tak třeba pro získání adresy prvního prvku čtvrtého pole (tj. prvku **pole**[3][0]) stačí zadat jen **pole**[3] (pozor na rozdíl jedné mezi číslováním počtu prvků a jeho pozičním číslem v poli). V cyklu můžeme pro adresování i získání hodnot prvků použít i prefix či postfix pro zvyšování (resp. snižování) hodnoty ukazatele:

```
*ukpole++   resp.   *++ukpole
```

Tady je ovšem nutno vzít v potaz, že - oproti výše uvedenému prostému přičítání ofsetu - **zvyšujeme přímo obsah ukazatele**, který tak ztratí svou původní hodnotu (adresu začátku pole). Někdy nám to nemusí vadit, naopak s tím budeme vřele souhlasit. Chyba se však objeví, když na to zapomeneme a budeme se tímtéž ukazatelem domněle odkazovat na začátek pole, na nějž už ukazatel samozřejmě neodkazuje.

Jaký z uvedených způsobů adresování prvků zvolíte, záleží jen na vás, resp. na tom, který z nich se v závislosti na konkrétní programové situaci ukáže být vhodnější.

Závorky nejsou v předchozím případě adresování prvku nutné, protože **unární operátory (zde * a ++) se vyhodnocují zprava doleva** (ale radši se u vašeho kompilátoru přesvědčete, zda je tomu opravdu tak). Kdybyste však chtěli zvýšit obsah adresovaného prvku, bez závorek by to nešlo: **(*ukpole)++**, což je totéž jako ***ukpole+=1**. V ofsetovém adresování ***(ukpole+1)** závorka být musí, protože * **má vyšší prioritu. Příkaz *ukpole+1 by nepřičetl 2 k ukazateli, ale 1 k adresovanému prvku pole.**

Pozor rovněž při použití zkráceného zápisu početních operací. Např.:

```
pole[prom++]+=5;
```

není totéž, co:

```
pole[prom++]=pole[prom++]+5;
```

ale je to sled příkazů:

```
pole[prom]+=5;
prom++;
```

Proměnná prom se zvyšuje jen jednou! Základní schéma zkráceného zápisu je:

```
výraz1 op= výraz2
```

kde op je operátor. V rozvedeném tvaru:

```
(výraz1) = ((výraz1) op (výraz2))
```

Levá strana se na výpočtu pravé strany neúčastní. Teprve výsledek výpočtu na pravé straně bude přiřazen výrazu na levé straně. Z toho plyne, že v předchozím příkladu se operace prom++ provede skutečně jen jednou.

Myslím ale, že než sám sebe plést podobnými "vánočkami" ve vlastním programu, je lepší dbát na jeho zřetelnost.

Dál si probereme všechny možné zápisy při použití ukazatelů. Ještě jednou si zopakujme, že definici:

```
int pole[8][10];
```

nemusíme nutně říkat dvourozměrné pole 8*10, ale i že je to pole polí - zde jednorozměrné pole s osmi prvky, z nichž každý je prvním prvkem osmi jednorozměrných polí s deseti prvky (pole polí).

```
int **ukaz;
```

už známe - je to ukazatel na ukazatele typu int.

```
int *pole[5];
```

pole pěti ukazatelů na typ int.

```
int (*ukaz)[7];
```

novinka - jde o vytvoření jednoho ukazatele na sedmiprvkové pole typu int.

```
int *pole[3][4];
```

je tříprvkové pole ukazatelů na čtyřprvkové pole typu int (tedy nikoli "prosté" pole 3*4).

```
int (*ukaz)[4][8];
```

vytvoří jednoho ukazatele na pole 4*8 prvků typu int.

Ukazatele a funkce

```
char *fce();
```

funkce vracející ukazatele na typ char.

```
char (*fce)();
```

je ukazatel na funkci, která vrací typ char.

```
char *fce()[3];
```

funkce vracející ukazatele na tříprvkové pole typu char.

```
char *fce[5]();
```

je pětiprvkové pole ukazatelů na funkci, která vrací typ char.

Když si z uvedených deklarací odmyslíte středník, dostanete normální uvození funkcí. Nám už dobře známé uvození funkce svým názvem:

```
funkce()
```

není ničím jiným, než:

```
int funkce()
```

což je funkce vracející typ int (kompilátor si zamlčený typ doplní na int). Jak je vidět z uvedeného přehledu, funkce mohou vracet i leccos jiného. Když chceme vrátit třeba typ char z funkce vracející typ int, musíme v ní použít konverzní operaci, jejíž výsledek uložíme jako argument příkazu return (třeba pomocí typedef a cast). Tím se poněkud složitě vyhneme přirozené intenci funkce vrátit int. Když ve volající funkci deklarujeme

funkci vracející typ char a pak ji zavoláme, tato funkce požadovaný typ vrátí "sama o sobě" (deklarace funkce není povinná). Na první pohled se deklarování volaných funkcí v těle funkce volající může zdát jako zbytečnost. Tyto deklarace však poskytují kompilátoru (nakonec i programátorovi) velmi cennou a hlavně jednoznačnou informaci pro další práci s daty a funkcemi programu. Někdy bez takové informace (díky určitému stupni víceznačnosti) může dojít k nežádoucímu zmatení chodu programu.

Novější kompilátory ve standardu ANSI umožňují ukládat do kulatých závorek za názvem funkce i typ přenášených parametrů. Např.:

```
int *fce(char *ukaz, int *pole)
```

My, co nemáme přístup k ANSI, musíme provést následnou deklaraci parametrů ve starém stylu:

```
int *fce(ukaz,pole)
char *ukaz; int *pole;
```

A co víc - deklarovat tuto funkci ve funkci volající můžeme jen bez parametrů, tedy s prázdnou závorkou.

Tady nejde o "míň/víc psaní", ale o to, že formálně stejnou deklarací volané funkce ve funkci volající je vše předem jednoznačně určeno, pravděpodobnost nečekaných záludností v běhu rozsáhlejšího programu se blíží nule. Zde nelze než doporučit značně upravené druhé vydání céčkové bible B. W. Kernighana a D. M. Ritchieho - The C Programming Language, Second Edition (vydal Prentice Hall v edici Software Series v roce 1988). První vydání, jehož slovenský překlad vzešel z nakladatelství ALFA, obsahuje standard vyšlý z módy (i tak budiž ALFA velebena!).

Heapsort

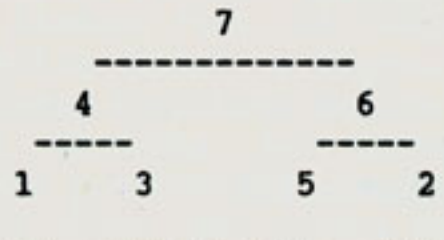
je jedním z typů asociativního třídění. Ukážeme si na něm spolupráci číselných a znakových polí.

Heapsort má dvě vlastnosti, jejichž spojením vyniká nad ostatními typy třídění - vedle vysoké rychlosti dané výrazem O(N log N) (N je počet tříděných dat, O symbol složitosti algoritmu) nevyžaduje ani bajt paměti navíc. Což se nedá říci např. o quick sortu, který pro svou rekurzi potřebuje dost velký zásobník.

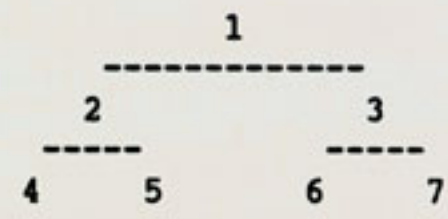
Základem heapsortu je binární strom. Z každého prvku vedou dvě "větvičky", na nichž sedí dvě děti. Ty jsou vždy menší (při opačné iteraci větší) než rodič nad nimi. Je-li pod dítětem opět nějaký prvek (jeden či dva), stává se dítě jeho (jejich) rodičem, atd... Binární strom musí být vyvážený. To znamená, že tvarově musí připomínat rovnostranný trojúhelník s (takřka) horizontální základnou, v níž prvky přibývají umisťováním do nejbližší volné pozice zprava. Heap (čte se hip) je česky hromada, halda. Mějme např. haldu neutříděných prvků:

4 7 6 1 3 5 2

Algoritmus třídění je dvoufázový. V první fázi se do špičky stromu dostane největší prvek. Po první fázi bude pole seřazeno podle výše uvedeného pravidla:



Úlohou druhé fáze je pole dotřídit. Algoritmus vždy odebere ze špičky stromu nejvyšší prvek a prohodí jej s prvkem na konci pole aktuálně zpracovávaných prvků. Poprvé tedy prohodí číslo 7 s číslem 2. Sedmička se tak dostane na svou správnou "utříděnou" pozici v poli (jako největší je na jeho konci). Dvojka ze špičky se "propadne" vpravo dolů, protože z obou dětí pod špičkou je 6 větší než 4. 6 se dostane špičku, 2 na jeho původní pozici. O patro níž se 2 porovná už jen s číslem 5, protože to je poslední aktuální pozice pole (v programu proměnná dol). Sedmička už je tabu. 2 je míň než 5, proto se 2 prohodí s 5. A níž už není co porovnávat. Prvek ze špičky stromu, tedy 6, se prohodí s posledním aktuálním prvkem, tj. s 2. Sestka je na správné pozici, dolní mez se opět posune a pokračuje se až do konce. Výsledkem 2. fáze je setřídění:



Prvky ve stromu se číslují (indexují) stejně, jak ukazují záměrně vybrané hodnoty setříděných prvků. Proto platí základní jednoduché vztahy:

```
rodič=int(dítě/2)
levédítě=2*rodič
pravédítě=levédítě+1
```

V paměti jsou prvky seřazeny podle svých indexů:

1 2 3 4 5 6 7

Vše je nesmírně jednoduché a všechny početní operace jsou založeny jen na uvedených třech vztazích a na porovnávání rodiče s jeho dětmi, resp. porovnáním obou dětí. (Heapsortem a jeho sestrou - řadou priorit - se zevrubněji zabývám v článku, který vyjde v Elektronice přibližně v únoru 90).

Připojený výpis programu je ukázkou třídění alfanumerických řetězců pomocí ukazatelů, takže řetězce se v paměti ani nehnou a prohazují se jen jejich ukazatele (podobně jako při třídění quick sortem či bubble sortem v minulém pokračování). Díky nulové prostorové náročnosti heapsortu můžeme na malém počítači utřídit všechno, co se nám do něj podaří "nacpat". Volba tohoto třídění je vhodná v případě potřeby rychlého utřídění dat nějakého programu, který by bylo obtížné nebo i zbytečné spojovat s běžně užívanými databázemi. Heapsort je velmi účinný při spojování dvou či více polí do jednoho. Pole buď připojíme v paměti za sebe a necháme je "vrůst" do sebe, nebo jejich páry spojíme pod společnou špičkou (jeden strom bude jeho levou, druhý jeho pravou částí - problémem je tu vyvážení stromu).

Heapsort napsaný v assembleru utřídil za vteřinu asi 300 řetězců, Céčko je přibližně čtyřikrát pomalejší. Oba údaje se týkají ZX Spectra. Program má jeden drobný nedostatek - např. řetězce CCCC a CCCCCCCC jsou pro něj shodné. V případě potřeby to lze ošetřit jednoduchým testem.

Program v první fázi prosívá strom zdola (funkce siftup()), ve druhé fázi shora (funkce siftdown()). Retězce porovnává funkce porovn(). V případě potřeby výměny pozic ukazatelů ve stromu se volá funkce prohoz(). Pro možnost okamžitého odzkoušení programu jsou do pole řetězců uloženy názvy dnů v týdnu. Na výstupu dostanete jejich abecední výpis.

```
#define POCET
char *dny[]={0,"pondělí","úterý","středa",
             "čtvrtek","pátek","sobota",
             "neděle");

main()
```

```
{
int i;
for(i=2,i<=POCET,i++)
  siftup(i);
for(i=POCET;i>1;i--)
 {prohoz(&dny[1],&dny[i]);
  siftdown(i-1);}
for(i=1;i<=POCET;i++)
  printf("%s\n",dny[i]);
}
siftup(dol)
int dol;
{
int dítě,rodič;
for(;;)
  {
  if(dítě==1) break;
  rodič=dítě/2;
  if(porovn(dny[rodič],dny[dítě])) break;
  prohoz(&dny[rodič],&dny[dítě]);
  dítě=rodič;
  }
}
siftdown(dol)
int dol;
{
int dítě,rodič;
rodič=1;
for(;;)
  {
  dítě=2*rodič;
  if(dítě>dol) break;
  if(dítě+1<=dol && porovn(dny[dítě+1],
                           dny[dítě])) break;
  prohoz(&dny[rodič],&dny[dítě]);
  rodič=dítě;
  }
}
porovn(rodič,dítě)
char *rodič,*dítě;
{
while(*rodič==*dítě)
 {if(dite=='\0') break;
  *rodič++; *dítě++;}
if(*rodič>=*dítě)
  return 1;
return 0;
}
prohoz(rodič,dítě)
int *rodič,*dítě;
{
int x;
x=*dítě;
*dítě=*rodič;
*rodič=x;
}
```

Makro **POČET** obsahuje počet alfanumerických řetězců tříděného pole. Po přepracování programu pro přímé vkládání řetězců z klávesnice by **POCET** narůstal programové.

Prvním prvkem pole řetězců je nula, protože z hlediska indexování prvků ve stromu je jednodušší počítat prvky od jedné. Prvek na pozici 0 a jemu odpovídající ukazatel jsou tedy ignorovány.

Základní algoritmus celého programu je obsahem prvních dvou cyklů **for** funkce **main()**. Třetí vypisuje utříděné pole řetězců (přesněji: vypisuje řetězce podle programem sestavené posloupnosti pole jejich ukazatelů).

V obou prosívacích funkcích je užit cyklus for s prázdnými příkazy. To je vhodné vždy, když potřebujeme něco dostat do smyčky bez stanovení vstupních podmínek cyklu. Uvnitř takového cyklu ovšem musíme mít iterace, které nás z něj v pravý moment vyvedou - zde jsou na řádkách, končících příkazem ukončení cyklu break. Místo for(;;) může-

me použít třeba while(něco=1). Podmínka bude vždy splněna, protože něco bude mít stále hodnotu 1, bude tedy vždy "pravdivé". Ovšem to je svázáno se zbytečnou deklarací jedné proměnné navíc.

V **siftup()** se porovnává rodič postupně s každým dítětem. V **siftdown()** se napřed zjistí, které ze sousedících dětí (jsou-li k dispozici obě) je větší, a teprve s ním je porovnán jeho rodič. V obou prosívacích funkcích dojde k prohozu vždy, když je rodič menší než dítě.

Funkce **porovn()** postupně porovnává oba zvolené řetězce znak po znaku. Když je řetězec rodiče větší než nebo stejný jako řetězec dítěte, funkce vrací hodnotu 1. Podmínka porovnání if ve volající funkci bude pravdivá a provede se příkaz ukončení cyklu break. Jinak **porovn()** vrací 0 - podmínka if bude nepravdivá a cyklus pokračuje dál.

K prohozu netřeba nic dodávat. Povšimněte si jen jednoho základního rozdílu - ve funkci **porovn()** jsou deklarovány ukazatele na typ char, ve funkci **prohoz()** ukazatele na typ int. Funkce **porovn()** operuje se znaky řetězců, **prohoz()** s číselnými ukazateli na řetězce.

Program lze o dost zkrátit, když použijeme základní algoritmus, který pracuje jen s prosíváním shora (uvedeno v pseudojazyce):

```
for i=int(N/2) downto 1
  SiftDown(i,N);
for i=N downto 2
  Prohoz(X[1],X[i]);
  Siftdown(1,i-1);
```

Časová složitost programu zůstává O(N log N) a opět není třeba ani bajtu navíc. Předělat uvedený program podle tohoto algoritmu jistě dokážete sami.

Ještě jednou rekurze

V některém z pokračování seriálu jsem se o ní zmínil, ale zůstal jsem dlužen jedno vysvětlení. Upozornil mne na ně čtenář, kterému "to" nechtělo fungovat. Potýkal se s hierarchií volání a návratů.

Pro demonstraci jsem jako základ zvolil rekurzní program pro dekadický výpis obsahu proměnné (ve slovenském vydání Kernighana na str.95). Na první pohled může zadání vypadat jako nesmyslné - když mám třeba n==123, pak se při požadavku na dekadický výpis obsahu proměnné n vypíše 123, ne? Na obrazovce ano. Ale jak takové n převést na sled tří kódů pro výpis na tiskárně? Nebo co když budu chtít dát stovky sem, desítky tam a jednotky někam jinam? Apod.

Program jsem trochu upravil kvůli výpisu parametrů:

```
int x=0;
main()
{int n; n=123;
printd(n)}
printd(n)
int n;
{int i;
++x;
printf("n%u=%u ",x,i);
if(n<0)
 (printf("%c",'-');
  n=-n;.
if((i=n/10)!=0)
 (printf("i%u=%u\n",x,i);
 printd(i);}
printf("\n%c",n%i0+'\0');.
```

Výsledný výpis:

```
n1=123 i1=12
n2=12  i2=1
n3=1

123
```

Proměnná i se potřetí už nevypsala, protože měla nulový obsah. Test if na různost od nuly byl nepravdivý, funkce už nezavolala samu sebe a programové řízení přešlo na závěrečný příkaz tisku. Tento příkaz se provede třikrát za sebou. Proč? Protože po každém zavolání funkce **printd(i)** se do zásobníku uloží návratová adresa, tj. adresa příkazu za příkazem volajícím funkci. Do zásobníku se dvakrát po sobě uložila adresa posledního příkazu **printf();**. Vždy, když po jeho provedení programové řízení padne na pravou svorku za ním, odebere adresu ze spodku zásobníku. A tak příkaz "skáče sám na sebe", dokud se na spodku zásobníku neobjeví adresa návratu do funkce main().

Na toto pravidlo ukládání adres do zásobníku a na jejich odebírání ze zásobníku v opačném sledu nesmíte při konstrukci rekurze zapomenout (obdoba assemblerových instrukcí CALL a RET).

I když program pracuje s automatickými proměnnými, ty nezmizí, dokud se každá z otevřených úrovní rekurze opět neuzavře. Proto si jednotlivá n ponechávají svou hodnotu, přestože jich v automatické paměti bude tolik, kolik bude rekurzí. Mezi sebou se nijak neovlivňují, protože každé z nich lokálně patří "jiné" funkci. Při otevírání jednotlivých úrovní rekurze jednotlivá n postupně nabývají různých hodnot, při zpětném pohybu (uzavírání rekurze) od nich tyto hodnoty postupně odebíráme.

V posledním příkazu **printf()** tak proběhnou tyto operace:

```
  1%10=1   1+'\0'=49, tj. ASCII kód čísla 1
 12%10=2   2+'\0'=50, tj.        "        2
123%10=3   3+'\0'=51, tj.        "        3
```

V prvním sloupci jsou celočíselné zůstatky po dělení deseti. ASCII kódy se zobrazí ve znakové reprezentaci.

Basicoví vyznavači si mohou rekurzi prověřit ve své oblíbené mluvě, i když v ní nejde jednoduše zařídit, aby se proměnné různých úrovní rekurze "neničily" (Basic nemá lokální proměnné). Jen pro zažití základního principu zkuste tenhle program, prohlédněte si jeho výpis a určitě pochopíte:

```
10 GOSUB 50: STOP
50 LET A=5
60 LET A=A-1: IF A THEN PRINT A: GOSUB 60
70 LET A=A+1: PRINT A
80 RETURN
```

RETURN hraje roli pravé svorky se čtyřmi návraty na řádku 70 (GOSUB 50 uložil do zásobníku 1 adresu návratu, GOSUB 60 4 adresy návratů). Poslední, pátý návrat proto skočí na příkaz STOP.

Pokud znáte princip quick sortu, můžete prozkoumat Céčkovou podobu jeho rekurze (quick sort je v minulé části seriálu).

Tímto pokračováním dávám dalšímu zkoumání ukazatelů vale - na úrovni céčkového adepta se o nich víc už sdělit nedá. Příště ahoj u struktur a unionů.

-elzet-

● ● ●

ZNÁTE KLUB dB?

V dubnu konečně došlo k založení nové klubové sekce 602. ZO Svazarmu Praha, nazvané Klub dB. Jak sám název napovídá, jde o sdružení uživatelů mikropočítačových databází, zejména typu dBase. Na rozdíl od dosavadních klubů zde však jde vesměs o profesionální uživatele, tedy o ty, kteří s databázemi pracují ve svém zaměstnání. Není rozhodující zda jako uživatelé, či jako programátoři.

Vytvořením klubu došlo k vyplnění citelné mezery, reprezentované všeobecnou neinformovaností v oblasti databázových systémů a roztříštěností při užívání a programování v nich. Z toho vyplývá také poslání, které si vzal nový klub za své. Měl by sjednotit úsilí všech uživatelů profesionálních databází, co se týče informací a tvorby aplikací.

Adresa: Klub dB, 602. ZO Svazarmu,
Wintrova 8
160 41 Praha 6

● ● ●

ZÁVADA MAGNETOFONU U SHARP MZ-821

L. Jeřábek, 602. ZO

Poměrně častou závadou data-magnetofonu v osobním počítači SHARP MZ-821 je ulamování kontaktů, které spínají po zmačknutí klávesy PLAY na 5. kontakt mgf. konektoru log. nulu. Tím počítač dostane na vědomí, že tato klívesa je stisknuta a že může spustit vstup či výstup dat. Závada se zprvu projevuje při hlášení "IPL is looking for a program" ještě návěštím "PLAY" s šipkou. Posléze dochází i k samovolnému rozbíhání motorku. V této fázi poruchy jsou kontakty jen unaveny nebo nalomeny. Příčinou bývá ponechávání zmačknutých kláves mgf. déle, než je třeba pro přenos dat, nebo přirozená únava materiálu.

Pomoci si lze tím, že kontakty vypájíme spolu se základovou destičkou z plošného spoje a asi do výšky 1 mm naneseme na tuto destičku, kterou kontakty pravoúhle procházejí, dvousložkové lepidlo. Tím změníme i místo ohybu kontaktů, které po opětném zapájení budou pracovat k naší plné spokojenosti. tuto úpravu doporučuji udělat i preventivně, protože v místě, kde kontakty procházejí destičkou, jsou zeslabeny a tedy logicky se zde ulamují. Závěrem doporučuji méně zdatným kutilům, aby vyhledali některý SHARP klub Svazarmu a požádali o pomoc. *

Příjem teletextu pomocí osobního počítače /2/

Tomáš Laška (hardware) a
Michal Matyska (software)

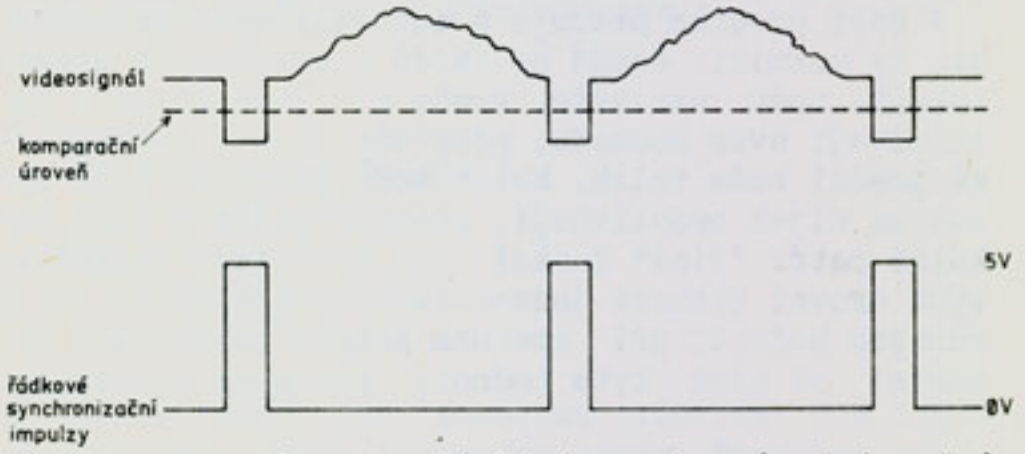
(pokračování)

Vývoj adaptéru

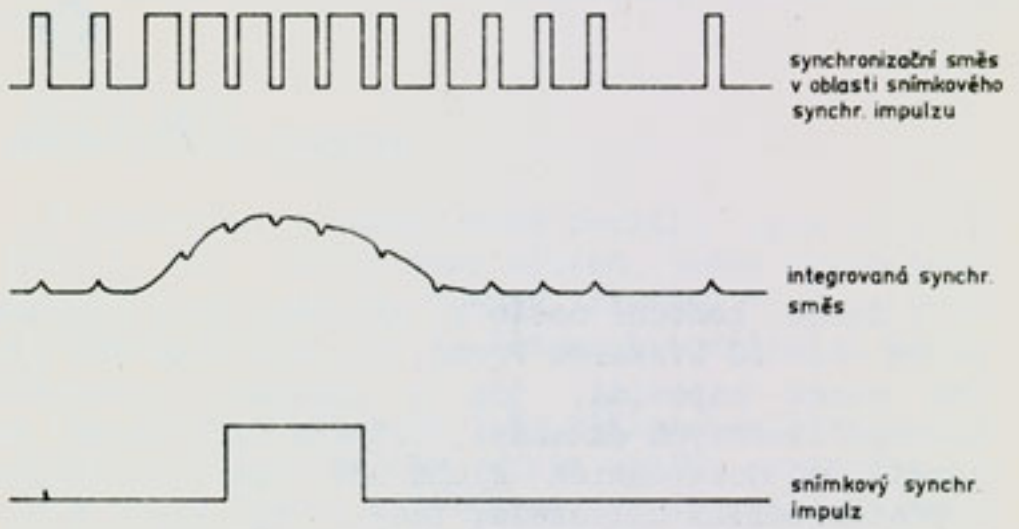
Při návrhu TTX adaptéru jsme vycházeli z toho, že TTX data se mohou vyskytovat jen v zatemněných řádcích a po zbytek půlsnímku videosignál nese obrazovou informaci. Vyjímečně se na západě využívá možnost vysílat Teletext celý půlsnímek a to když nevysílá normální program. Tato možnost u nás asi nepřichází v úvahu. Proto jsme volili takovou koncepci, že se v době kdy videosignál přenáší Teletext, TTX data přesunou do statické paměti a ve zbytku půlsnímku, kdy se přenáší obraz, jsou data přenesena do operační paměti počítače, kde budou po načtení celé stránky dekódována a zobrazena.

Z technického hlediska je příjem Teletextu velmi složitá problematika. Je to dáno především velkou přenosovou rychlostí a množstvím rušivých a nežádoucích vlivů, které vznikají přenosem a distribucí televizního signálu a v konečném důsledku i kvalitou přijímače na straně spotřebitele informace. Proto bychom rádi čtenáře seznámili s naším postupem a s problémy, na které jsme při vývoji adaptéru narazili. Vývoj našeho adaptéru podnítil článek v AR A 3/88 od ing. Macka, kde je ovšem několik chybiček (nesouhlasí čísla řádků na kterých se TTX vysílá). Také nám zůstalo záhadou proč je na všech obrázcích TTX řádku (i v naší normě) zakreslen signál burst, který slouží k synchronizaci barev v normě PAL, přestože se u nás vysílá v SECAM.

Než se zmíníme o našich verzích, popíšeme části, které jsou společné pro každý adaptér či dekodér TTX. Vstupním signálem adaptéru je úplný televizní signál o mezivrcholové úrovni 1 V (dále jen videosignál). Tento signál je u většiny modernějších barevných TV přijímačů vyveden, stejně jako videovstup. Pokud není videosignál vyveden, nečiní jeho vyhledání a vyvedení potíže ani méně zkušenému amatérovi. (Zde znovu upozorňujeme na bezpečnostní předpisy - pozn. red.) Prvním blokem adaptéru je obnovitel stejnosměrné složky. Na jeho vstup je videosignál přiveden stíněným kablíkem o impedanci 75 ohmů. Tento obvod je nutný, protože signál je vázán kapacitně a změna pozadí obrazu by způsobila nabíjení nebo vybíjení kondenzátoru a tím i kolísání stejnosměrné složky signálu. Na tomto místě v našem případě zcela vyhověl jednoduchý upínací obvod s diodou. Dalším obvodem je širokopásmový zesilovač, který signál asi 2,5x zesílí. Zesílení signálu je výhodné z hlediska dalších obvodů (nevadí tolik kolísání a odezva obvodů je rychlejší). Pro činnost adaptéru je nezbytný další blok a to oddělovač synchronizačních impulzů. Do tohoto bloku je přiveden zesílený videosignál. Jak známo, ve videosignálu jsou řádkové a snímkové synchronizační impulzy a právě tyto impulzy oddělovač (separátor) oddělí. Na jeho výstupu již jsou v úrovni TTL. Rádkové synchronizační impulzy se oddělí komparátorem, jehož komparační úroveň je pevně nastavena děličem. Tento způsob postačí, protože obnovitel stejnosměrné složky upíná řádkové synchronizační impulzy. Do druhého vstupu komparátoru je přiveden zesílený videosignál. Na jeho výstupu se objeví synchronizační směs. Tento proces je patrný z obr. 3. Pomocí dvojitého integračního článku se ze směsi oddělí snímkový synchronizační impulz - viz obr. 4.



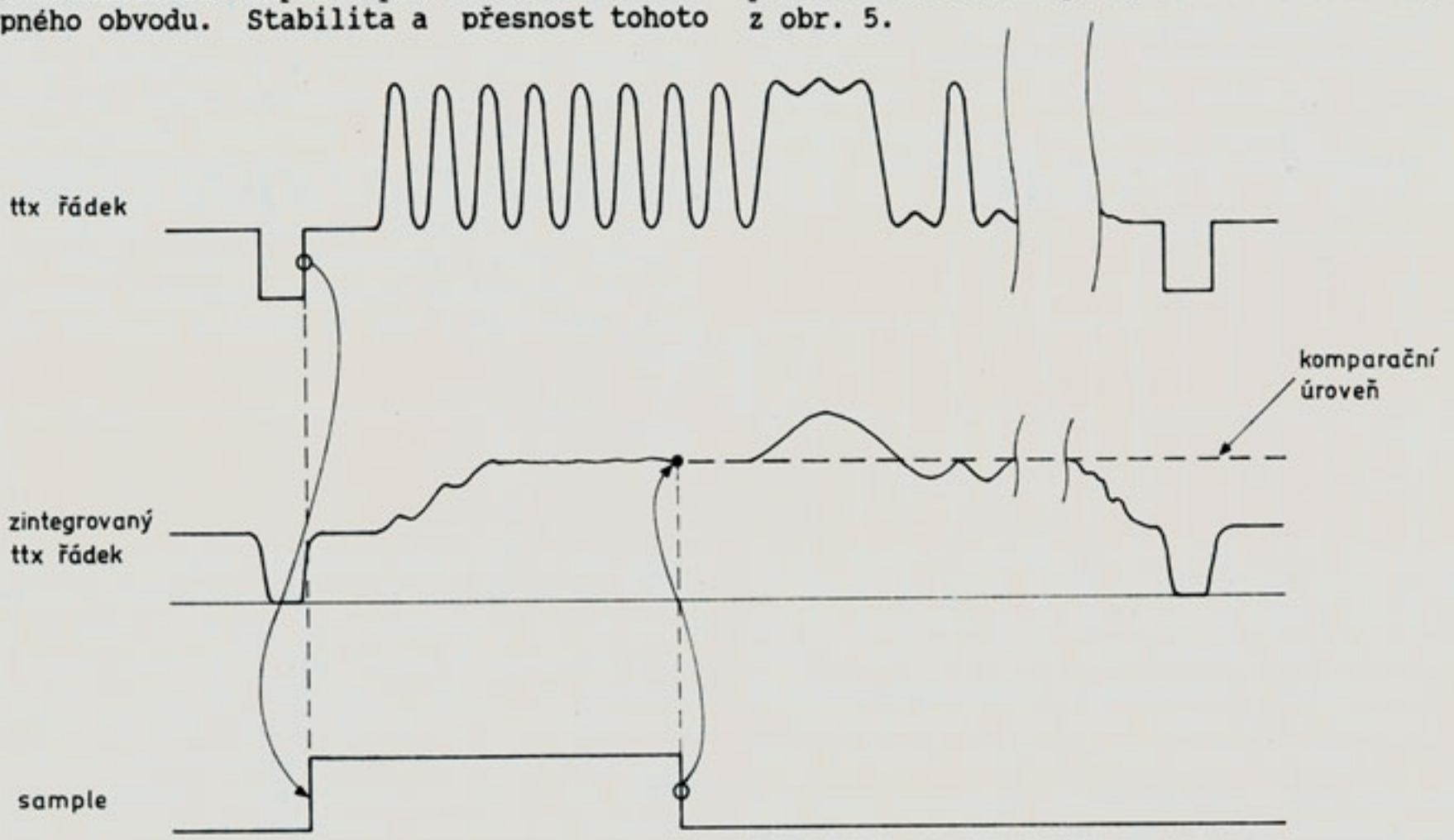
Obr. 3 - Získání synchronizační směsi



Obr. 4 - Oddělení snímkových synchron. impulzů

Velmi důležitou částí je blok obnovy (regenerace) Teletextu. Tato část má za úkol převést Teletextové úrovně v TTX řádku do úrovní logiky TTL. To je zajištěno velmi rychlým komparátorem. Na jeho neinvertujícím vstupu je zesílený videosignál, na invertujícím vstupu je navzorkována jeho střední hodnota v okamžiku, kdy se v TTX signálu nacházejí dva bajty CRI. Tak na výstupu komparátoru velmi efektně a vtipně získáme obnovený signál TTX v logice TTL. Tímto postupem je výsledný signál nezávislý na kolísání amplitudy a stejnosměrné složky vstupního videosignálu, v konečném důsledku i na kmitočtové charakteristice přijímacího zařízení. U tohoto stěžejního místa se chvilku zastavíme. Jak bylo popsáno dříve, dva byte CRI představují harmonický signál o kmitočtu 3,46875 MHz. Pokud videosignál integrujeme RC členem s touto konstantou, získáme v místě kde se nachází CRI stejnosměrnou úroveň, která je dána střední hodnotou CRI plus stejnosměrná složka videosignálu. Je to v podstatě ideální komparační úroveň. Musíme jí ovšem zachovat po celou dobu TTX řádku. To se děje pomocí obvodu sample & hold (vzorkuj a drž). Režim tohoto obvodu je ovládán logickým

signálem nazvaným SAMPLE. Ten je odvozen od řádkového synchronizačního impulzu pomocí monostabilního klopného obvodu. Stabilita a přesnost tohoto signálu není kritická, protože integrační článek prvního řádu má rychlý náběh. To vše je patrné z obr. 5.



Obr. 5 - Získání komparační úrovně

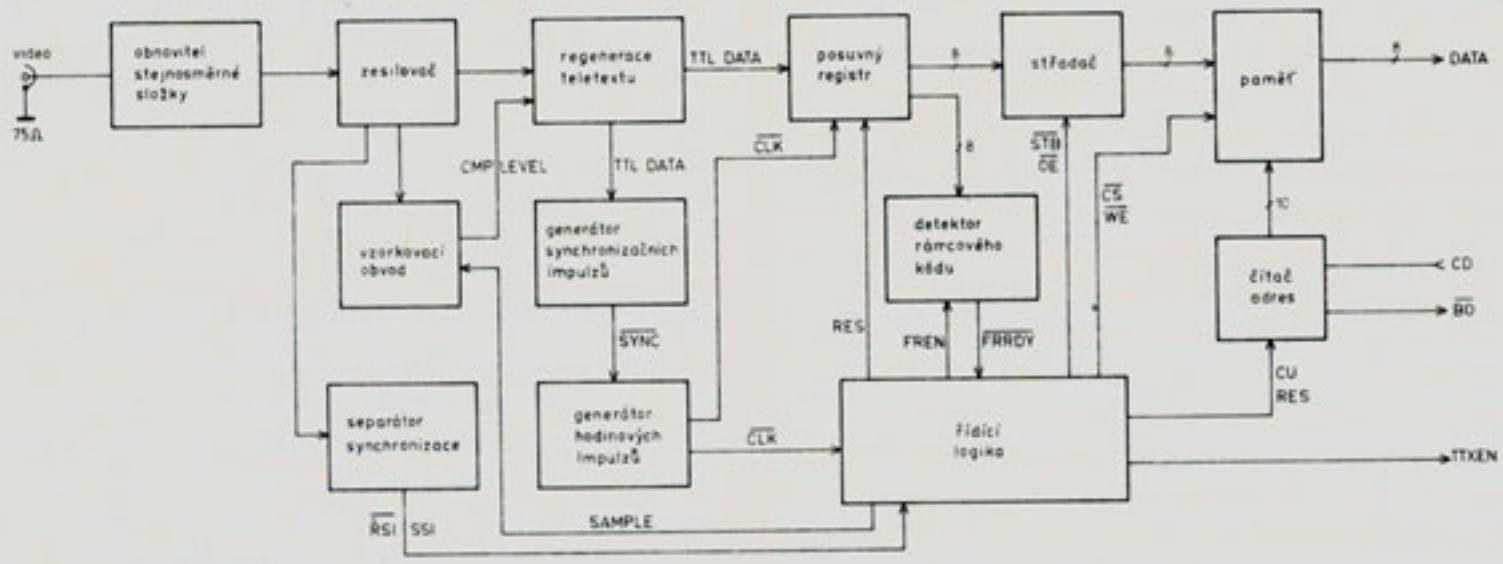
Výše popsaná část by se dala nazvat jako analogová. Nyní se věnujeme popisu naší první verze digitální části, kterou jsme měli zhotovenou velmi rychle. Vycházeli jsme z toho, že TTX řádky se nacházejí na přesně určených pozicích. Proto jsme počítali čítačem TV řádky a v místě platných řádků jsme data přesunuli bit po bitu do rychlých Schottky pamětí s organizací 256x1 bit. Úkolem software pak bylo nalézt v datech FC (framing code - rámcový kód) a pak narotovat data do bajtu. Tato verze měla mnoho nevýhod. Paměti a čítače byly pomalé, softwér byl pomalý (mnoho dat, hledání framingu, rotování). Toto řešení se ukázalo jako nefunkční, nepružné a ideově pochybené.

Rozhodli jsme se proto provést částečný sériově paralelní převod pomocí čtyřbitového posuvného registru. To proto,že jsme chtěli použít statické paměti s organizací 1024x4 bity. Díky shift registru odpadly potíže s rychlostí paměti a čítačů. Identifikace platných TTX řádků zůstala stejná - čítač řádků a software také. S touto verzí se nám podařilo 24.11.1988 zachytit poprvé bezchybná data.

Obě verze používaly v obvodu regenerace TTX trimru pro nastavení správné rozhodovací úrovně a obsahovaly další nastavovací prvky, které se musely nastavovat pomocí kvalitního dvoukanálového osciloskopu do 100 MHz.

Naše radost však netrvala dlouho, protože se začaly vysílat 4+4 řádky (místo dvou identifikačních impulzů SECAM, což je prohřešek proti normě, ale přinesl významné zvýšení rychlosti přístupu k informaci - viz výše). Naše druhá verze taktéž neumožňovala přijímat zahraniční TTX s jinou pozicí řádků. Kacířskou myšlenku, nastavovat pozice pomocí DIP SWITCHe, jsme po krátké úvaze zavrhli a v rekordní době jsme zhotovili verzi třetí, zatím nejdokonalejší a snad poslední, která je maximálně optimalizována z hlediska součástek a zapojení.

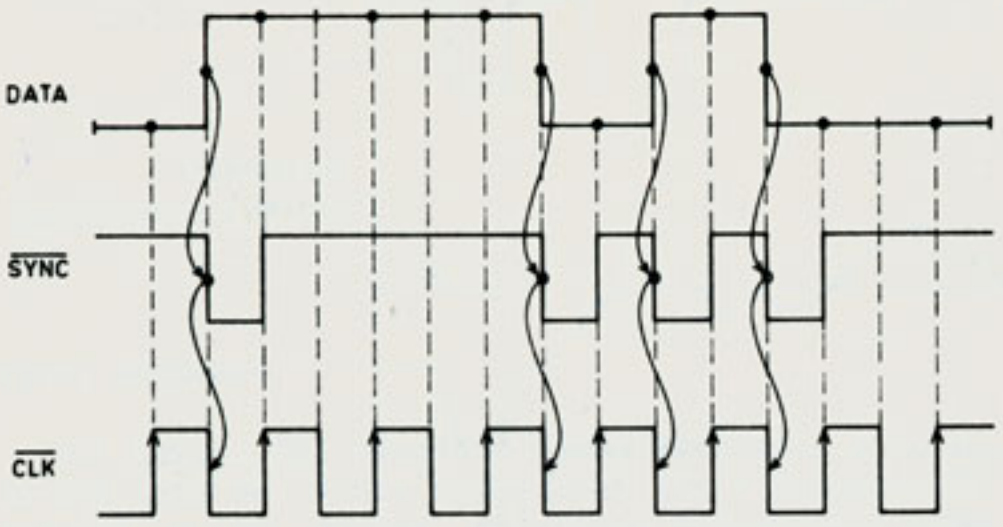
Blokové schéma poslední verze je na obr. 6.



Obr. 6 - Blokové schema adaptéru pro Teletext

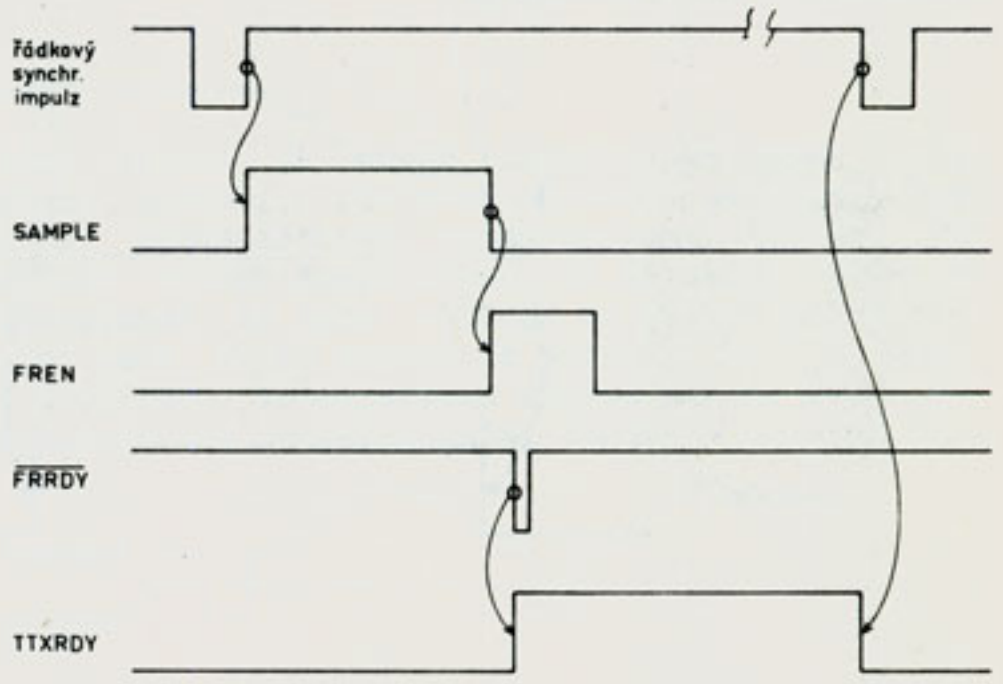
Analogová část je shodná s dříve popsanou a tak dále popíšeme pouze digitální část. Dá se říci, že srdcem adaptéru je generátor hodinových impulzů, který kmitá na frekvenci 6,9375 MHz. Na jeho stabilitě, ale zejména na jeho schopnosti synchronizovat se, závisí celá činnost adaptéru. Z tohoto generátoru je odvozeno vzorkování dat, časování paměti a posun v shift registru. Byly zkoušeny tři typy oscilátorů, dva RC a jeden LC. Nakonec se osvědčil nejlépe LC oscilátor, strhávaný v rytmu synchronizačních impulzů. Synchronizační impulzy pro generátor jsou odvozeny od datového signálu,

respektive od každé jeho změny pomocí derivačních RC článků. Ještě v bloku generáoru hodinových impulzů jsou hodiny invertovány, aby v bloku posuvného registru docházelo k správnému vzorkování dat. Posun v registru se totiž děje vzestupnou hranou hodin. Proces odvození synchronizačních impulzů, synchronizace hodin a vzorkování dat je na obr. 7.



Obr. 7 - Časování důležitých signálů

Na výstupu osmibitového shift registru jsou již data v paralelním tvaru. Na tento výstup je připojen detektor rámcového kódu. Pomocí diodové logiky je zde navolena kombinace odpovídající bajtu FC. Přitom framing se detekuje pouze asi ve třech bajtech po skončení dvou bajtech CRI. De facto je to odvozeno od signálu SAMPLE. Tento způsob se používá proto, aby nedošlo k výskytu falešného framingu na řádcích, kde TTX vůbec není (signál FREN - framing enable - framing povolen). Výstupní signál bloku FRRDY - framing ready - framing nalezen, který je zaveden do bloku řídící logiky, překlopí R/S klopný obvod. Signálem TTXRDY - teletext ready se odblokuje zápis dat do paměti. Paměti jsou samozřejmě spojeny s čítačem adres, který je rovněž řízen blokem řídící logiky. Tento blok se také stará o časování pamětí, nulování čítačů a řízení latch (střadače). Střadač je spojen s paralelním výstupem shift registru a je nezbytný, protože data jsou na výstupu registru pouze po dobu 144,44 ns, což nestačí k provedení zapisovacího cyklu paměti. Takto jsou data na výstupu střadače 1,155 mikrosekundy (8*144,44 ns tj. po dobu jednoho bajtu) a to je bohatě postačující. Detekce framingu a další výše jmenované signály jsou vidět na obr. 8. Neuvádíme odvození časování pamětí a střadače, přestože i tam je použito poměrně vtipné řešení.



Obr. 8 - Časování důležitých signálů

V bloku řídící logiky je rovněž generován signál TTXEN - teletext enable - teletext povolen, který je odvozen od snímkového synchronizačního impulzu a trvá po dobu možného výskytu teletextové informace, to je asi 22 řádků. Jeho ' 'vání ani stabilita nejsou kritické.

(pokračování příště)

● ● ●

SBC VE STROJOVÉM KÓDU

Filip Dvořák

V instrukčním souboru tohoto mikroprocesoru je velmi často používanou instrukcí SBC HL,RR (RR je párový registr BC, DE, HL, SP) zejména pro porovnávání obsahu dvou registrových párů.

Výsledek této operace ovlivňuje příznaky znaménka (S), nuly (Z) a přenosu (CY). V odborné literatuře je obvykle uváděno, že příznak znaménka je dosazen, je-li výsledkem operace záporné číslo (S=1).

Tato informace je však pravdivá jen za určité okolnosti a to tehdy, je-li výsledek v registru HL menší než 32768, to je 8000H. Bude-li výsledek větší nebo roven 32768, bude dosazen příznak znaménka i když je výsledkem kladné číslo!

Následuje-li test příznaku znaménka po této instrukci např. s podmíněným skokem JP M,nn (JP P,nn), může dojít k nepříjemnému zhroucení zdánlivě bezchybného strojového programu.

Tato "chyba" je způsobena tím, že po instrukci SBC je do příznaku znaménka dosazen nejvyšší bit registru H. Ten však při hodnotě 8000H a vyšší je roven 1. Důsledkem je také např. na počítači AMSTRAD CPC 6128 zajímavý převod čísla 8000H a většího v Basicu příkazem PRINT &8000. Výsledek je -32768.

Stejně je tomu i v případě instrukce ADC HL,RR. Proto doporučuji při 16 bitových operacích SBC a ADC testovat zásadně příznak přenosu CY, resp. příznak nuly Z a vyhnout se testování příznaku znaménka S. Výsledek testu je pak stoprocentní.

Věřím, že tato maličkost ušetří nejednu pernou chvilku při programování ve strojovém kódu Z80 těm, kteří o ní dosud nevěděli. *

Postavte si s námi diskový řadič

/4/

Daniel Meca

S využitím služeb TR-DOSu napsal Jirka Lamač koncem roku 1988 první verzi BIOSu do CP/M. Byla odvozena z jeho CP/M pro Microdrive a to od verze 3.3, ze které odstranil vše kolem Microdrivů a doplnil volání služeb TR-DOSu. Mělo to však své mouchy. Nebyla zde ještě prozatím implementována čeština. Byly to vlastně dva samostatné operační systémy, kde jeden volal služby druhého. To způsobilo velké potíže se systémem chybových hlášeni, který v případě chybného přístupu na disk vypisoval svá hlášení normálním písmem přes vše ostatní. Navíc jsem horko-těžko upravoval zpracování chyb tak, aby se v případě chyby nevyskakovalo do Basicu. Nejvíc nám však vadilo, že práce s diskem je dost pomalá. Ani přečíslení sektorů ještě nepřineslo kýžený výsledek. Navíc pak disketa nebyla kompatibilní s žádným známějším systémem. To mně, fanatikovi na kompatibilitu všeho se vším, vadilo nejvíc.

Tady pomohlo přečíslovat sektory už při formátování, tak jak to dělají některé modernější systémy. Nejlépe vyhovělo číslování přes dva sektory. Stopa přitom byla přečtena, či zapsána v průběhu tří otáček disku. Problém byl v tom, že TR-DOS v. 5.xx formátuje jen přes jeden sektor. Napsat nový formátovací program není zas tak velký problém, ale jde o to, jak ovládat přímo řadič. Již dříve jsem se zmínil o tom, že řadič je normálně odpojený a připojí se jen při činnosti DOSu, pokud je aresována EPROM. Jsou dvě možnosti jak to obejít:

1. Zasáhnout do EPROM.
2. Zasáhnout do hardware.

Protože jsem neměl v tu dobu možnost přeprogramování EPROM, zvolil jsem druhou variantu. Po trošce bádání jsem na desce systému přerušil jeden spoj a přidal dvě diody a jeden odpor. Zapojení je vidět na obrázku 4, kde je do výřezu z obr. 2 silnou čarou zakreslena přidaná část. Diody by opět měly být nejlépe Schotkyho (kvůli malému úbytku napětí v propustném směru - vyhoví i germaniové). Opor by měl mít hodnotu asi 10÷22KΩ. Funguje to tak, že se v době činnosti DOSu vyšle na port #FD log. nula, čímž se resetuje klopný obvod D1 (IO5). Ten pak svým výstupem Q (pin 5) přes diodu podrží vstup 13 hradla IO1 na nule. To znemožní odpojení řadiče při adresování RAM. Vysláním #10 na tento port se řadič opět může odpojit. Diody s odporem zde realizují logickou funkci AND. Někteří přátelé, kteří zapojení zkoušeli, použili místo diod integrovaný obvod. Slíbil jsem jim, že i na tuto možnost upozorním. Použití integrovaného hradla je jistě správnější, ale jak jste si mohli všimnout, v celkovém zapojení Betadisku jsou diody použity úspěšně na více místech. Trochu jsem se pobavil, když jsem hledal vhodný typ IO ve velkém katalogu Tesla. Podle něj se vyrábí 74ALS08, jenže je v obsahu označen jako NAND. Na stránce 130 je už označení AND, také schematická značka odpovídá, ale je uvedena logická funkce NAND. Ověřil jsem si však v zahraničních katalozích, že je to AND, takže Tesla buď laškuje, nebo neví co vyrábí.

Programovou obsluhu zapnutí a vypnutí řadiče jsem vyřešil těmito dvěma podprogramy:

```
RON   DI
      PUSH  HL
      PUSH  BC
      LD    BC,#00FD
      LD    HL,REST
      PUSH  HL
      LD    HL,#2A53
      PUSH  HL
      LD    HL,#3D2F
      PUSH  HL
      LD    H,A
      XOR   A
      RET
REST  LD    A,H
      POP   BC
      POP   HL
      RET

ROFF  PUSH  AF
      LD    A,#10
      OUT   (#FD),A
      POP   AF
      EI
      RET
```

Místo obsáhlých komentářů stačí říci co je v TR-DOSu na adresách #2A53 a #3D2F. Na #3D2F je v obou verzích NOP a RET. V tom je právě ten vtip. Instrukce NOP je totiž na adrese, jejíž volání způsobí přistránkování EPROMky DOSu. RET je pak použit pro nepřímý skok na adresu, která je připravená na STACKu. Tak je možno vyvolat libovolnou rutinu v celé EPROM. Na adrese #2A53 jsou ve verzi 5.03 intrukce OUT (C),A a RET (v 5.01 jso tyto instrukce na adrese #2A09).

Oba podprogramy neporušují obsah žádného registru. U rutiny RON se stará podprogram REST o zrestaurování použitých registrů po návratu z EPROM. Po dobu práce s řadičem je zakázáno přerušení, protože při použitém způsobu čistě programové obsluhy řadiče nejsou velké časové rezervy. V programu se vždycky zavolá RON, provede se operace s řadičem a pak se zavolá ROFF.

Pro takto upravený Betadisk jsem pak už snadno napsal formátovací program, v němž si mohu zvolit přečíslování a počet sektorů na stopě. Zároveň v něm odstraňuji drobný nedostatek původního formátování v TR-DOSu. Betadisk totiž formátuje obě strany naprosto stejně. Projeví se to při použití takto naformátovaných disket na počítačích, které kontrolují označení strany v adresové značce

(např. IBM PC). Ty totiž nenalézají stranu 1. K tomuto formátovacímu programu se ještě příležitostně vrátím. Je totiž zajímavý tím, že dokáže nejrůznější formáty záznamu. V jeho menu si můžete vybrat z 22 různých formátů, včetně IBM PC (i s boot sektorem), všech formátů Robotronu, Amstrad system i Amstrad data a řady dalších.

Co je však hlavní - popsaná úprava Betadisku otevřela možnost napsat zcela nové rutiny pro celou obsluhu řadiče. Zde opět zapracoval Jirka Lamač, který upravil své diskové rutiny, které původně napsal do CP/M pro Sharp. V Mikrobázi jsme o implementaci na Sharp MZ 800 už stručně referovali).

Výsledek je vynikající. Nová verze CP/M pro ZX Spectrum s Betadiskem si v rychlosti a eleganci obsluhy disketových jednotek nijak nezadá s tou na Sharpu a je určitě jednou z nejrychlejších vůbec. Formát záznamu na disku byl při té příležitosti změněn na 9 sektorů po 512 bytech, takže je kompatibilní s CP/M na Sharpu a vlastně i s formátem IBM PC. Dokonce i CCP a BDOS začínají na stejném místě jako u Sharpu, takže je možný REBOOT ze stejné diskety na Sharpu i ZX Spectru. Navíc zároveň stoupla kapacita diskety z 320 na 360KB (resp. z 640 na 720KB při 80 stopách). Je počítáno se současným připojením až čtyř různých mechanik. Mechanické parametry se však nastavují podle té nejpomalejší z nich.

Také způsob zpracování chyb je oproti původnímu Betadisku vylepšen. Pro ilustraci - když Betadisk nenalezne sektor, zkouší ho hledat desetkrát, ale vžy předtím provede HOME a SEEK, tedy vrátí se na stopu 0 a pak znova hledá žádanou stopu. To vše provádí s krokem 30 mS, což trvá dlouho a po tu dobu mechanika silně připomíná starou telefonní ústřednu. Na obranu TR-DOSu musím podotknout, že je to naprosto běžný postup i u jiných systémů. Lamačův BIOS zkouší sice hledání také desetkrát, ale dělá vždy jenom jeden krok stranou a zpět (možná se Jirka v tomhle nechal inspirovat Járou Cimrmanem). Samozřejmě se přitom hlídá zda nejsou hlavy v krajní stopě a podle toho se určuje směr onoho kroku. Takových "chytrostí" je zde celá řada. Ale o tom si povíme jindy.

Určitě by stálo zato použít Lamačovy rutiny pro obsluhu řadiče i v systému Betadisku. Již delší dobu se mi honí hlavou několik námětů na různá vylepšení systému Betadisku, kvůli práci na Mikrobázi jsem však neměl zatím čas na jejich realizaci. Možná by pomohla týmová spolupráce. Ostatně, na podrobnosti také dojde.

Můžeme jen doufat, že se tato verze CP/M objeví co nejdříve v naší programové nabídce.

Musím ještě podotknout, že jsem se samozřejmě zabýval také myšlenkou použít k ovládání neupraveného řadiče jednotlivé příkazy, nebo části rutin z EPROM. Rozborem obsahu EPROM jsem našel např. kompletní rutiny pro vyslání sektoru do řadiče i pro jeho načtení, které by bylo možné využít. K dispozici je na několika místech též důležitá sekvence OUT (C),A a RET. Nenašel jsem však použitelný způsob načtení stavového registru řadiče a registrů stopy a sektoru. Na druhou stranu z tohoto rozboru vyplývá, že by stačilo do EPROM naprogramovat sekvenci příkazů IN A,(C) a RET, aby byla možná úplná programová obsluha řadiče bez hardwarových úprav. Místa je v EPROM dost. Popravdě řečeno, po bližším rozboru tohoto, na první pohled velice elegantního řešení, docházím stále více k názoru, že vlastně nepřináší téměř žádné výhody proti řešení hardwarovému. Naopak, některé operace s řadičem se zdají být tímto způsobem obtížněji řešitelné. Jediná možnost by byla volat přímo některé rutiny, které jsou už v EPROM hotové. Ostatně, některé rutiny TR-DOSu jsou napsány docela vtipně a není tedy nic špatného je využít.

Pro zajímavost, rutiny pro zápis a čtení sektoru v EPROM TR-DOSu vypadají takto:

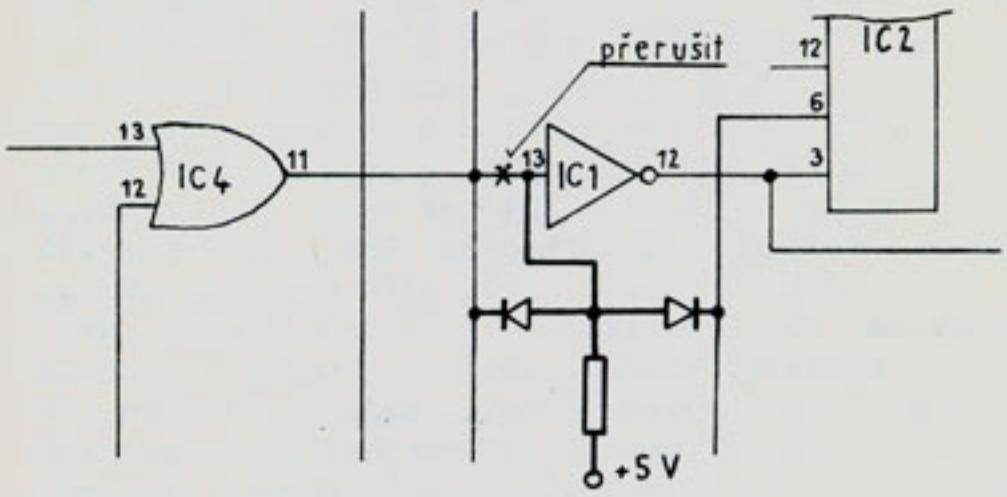
```
WRITE   LD    B,4         ;Časování pro timeout.
INFF1   IN    A,(#FF)     ;Čtení pomocného portu.
        AND   #C0         ;Maskování bitů 0-5.
        JR    NZ,OUTDA    ;Je-li DRQ, jdi na zapis.
        INC   DE          ;Čítač pro timeout.
        LD    A,E
        OR    D
        JR    NZ,INFF1    ;Další pokus, dokud nevyprší
        DJNZ  INFF1       ;timeout, jinak
        RET               ;návrat bez zápisu.
INFF2   IN    A,(#FF)     ;Nové čtení pomocného portu.
        AND   #C0         ;Maskování bitů 0-5.
        JR    Z,INFF2     ;Není-li DRQ, zpět.
        RET   M           ;Návrat při INTRQ.
OUTDA   OUTI              ;Zapiš bajt.
        JR    INFF2       ;Další.
```

```
READ    LD    B,4         ;Časování pro timeout.
INFF3   IN    A,(#FF)     ;Čtení pomocného portu.
        AND   #C0         ;Maskování bitů 0-5.
        JR    NZ,INDA     ;Je-li DRQ, jdi na čtení.
        INC   DE          ;Čítač pro timeout.
        LD    A,E
        OR    D
        JR    NZ,INFF3    ;Další pokus, dokud nevyprší
        DJNZ  INFF3       ;timeout, jinak
        RET               ;návrat bez čtení.
INFF4   IN    A,(#FF)     ;Nové čtení pomocného portu.
        AND   #C0         ;Maskování bitů 0-5.
        JR    Z,INFF4     ;Není-li DRQ, zpět.
        RET   M           ;Návrat při INTRQ.
INDA    INI               ;Načti bajt.
        JR    INFF4       ;Další.
```

Z výpisu je zřejmé, že se nikde nepočítá délka sektoru. Operace je ukončena řadičem, který vyšle INTRQ na konci sektoru a délku si ohlídá sám. Požadovanou délku sektoru zjistí řadič z takzvané adresové oblasti. O jejím uspořádání si povíme až bude čas. Takto napsané čtecí a záznamové rutiny umožňují čtení a záznam sektorů různé délky v rozsahu 128 až 1024 bajtů. To funguje i při volání služeb TR-DOSu. Jenom není možno použít těchto služeb pro multisektorové operace s jinou délkou sektoru než 256 bajtů, protože DOS tuto délku předpokládá a inkrementuje buffer vždy o tuto hodnotu.

Upozorňuji, že se jedná jen o hlavní výkonné rutiny. Před jejich zavoláním musí být již hlava nastavena na správné stopě, v registru sektoru musí být správná hodnota a do příkazového registru musí být vyslán příkaz ke čtení, či zápisu sektoru. To všechno není zas tak jednoduché, jak by se na první pohled zdálo. Ale o tom si zase povíme příště.

(pokračování příště)



Obr. 4 - Hardwarová úprava verze 5.xx pro samostatné ovládání řadiče (výřez z obr. 2 - přidaná část je zakreslena silnou čarou)

```
        RST  #10
        CALL PAZVUK
        CALL WAIT
        JP   ZPET
NECHCI
        LD   A,15
LD      A,15
        JR   ZPRAVA
PARERR
        LD   A,13
        RST  16
        LD   A,26
        JR   ZPRAVA
;---------------------
WAIT
        CALL #028E
        LD   A,E
        INC  E
        JR   Z,WAIT
        RET
;---------------------
PAZVUK
        LD   C,#FF
ZVUK
        LD   A,6
        OUT  (#FE),A
        CALL CEKA
        LD   A,22
        OUT  (#FE),A
        CALL CEKA
        DEC  C
        JR   NZ,ZVUK
        RET
CEKA    LD   B,C
SMYCKA  DJNZ SMYCKA
        RET
;---------------------

SAVE
        LD   BC,#0E02
        CALL ATTR1
        CALL PAZVUK
        CALL OPEN2
        LD   DE,TNAME
        LD   BC,9
        CALL #203C
        CALL ASCII
        LD   A,3
        LD   (#5B08),A
        XOR  A
        LD   DE,17
        LD   IX,#5B08
        CALL #04C2
        CALL TIME
        LD   A,#FF
        LD   DE,(#5B04)
        LD   IX,#6000
        CALL #04C2
        CALL TIME
        JP   ZPET
TNAME
        DEFB 22,#14,2
        DEFM "NAME: "
TIME
        LD   B,45
PRR     HALT
        DJNZ PRR
        RET
;---------------------

ODPOJ
        LD   A,155
        OUT  (127),A
        LD   A,79
        OUT  (223),A
        RET
;PGM=OE=CE=1 Vpp=5V
;D4=X=0 RELE=0 Vcc=5V D7=0
;---------------------

BASIC
        LD   IY,#5C3A
        LD   BC,#1013
        CALL ATTR1
        CALL PAZVUK
        RST  #08
        DEFB #FF
;---------------------
```

```
ASCII
        RES  5,(IY+1)
        LD   HL,#5B09
        LD   B,10
ASC
        BIT  5,(IY+1)
        JR   Z,ASC
        RES  5,(IY+1)
        LD   A,(IY-50)
        CP   13
        JR   Z,DOST
        LD   (HL),A
        INC  HL
        PUSH HL
        PUSH BC
        RST  #10
        POP  BC
        POP  HL
        DJNZ ASC
DOST
        XOR  A
        CP   B
        JR   Z,STTAP
        LD   A,32
PLN     LD   (HL),A
        INC  HL
        DJNZ PLN
STTAP   CALL OPEN1
        LD   DE,#09A1
        XOR  A
        CALL #0C0A
        SET  5,(IY+2)
        CALL #15B4
        RET
;---------------------

CTIEPR
        LD   A,(#5B00)
        CP   2
        LD   A,(BYT)
        OUT  (223),A
        HALT
        HALT
        HALT
        JR   Z,CT132
;PGM=1 OE=CE=0 Vpp=5v
;D4=0 RELE=1 Vcc=5V D7=0
CTI
        RES  1,A
        RES  2,A
        LD   (BYT),A
        OUT  (223),A
        CALL DATAZ
        CALL CEKEJ
BUDE    LD   HL,(#5B02) ; OD
        LD   DE,(#5B04) ; KOLIK
        LD   A,E
        OR   D
        RET
; test na ZERO=1 => ZPET
CT132
        RES  0,A ; #68 pro 27256
; PGM=A14=0
        JR   CTI
;---------------------
ODRAM
        LD   IX,#6000
        PUSH HL
        POP  BC
        AND  A
        ADD  IX,BC
        RET
; test na CY=1 => ZPET
;---------------------

ADRESA
        LD   A,L
        OUT  (63),A
        LD   A,H
        OUT  (95),A
        CP   #40
        CALL Z,VELADR
        RET
VELADR
        LD   A,(BYT)
        SET  0,A
        LD   (BYT),A
        OUT  (223),A
        RET
;---------------------
CEKEJ
        CALL OPEN1
        LD   B,31
        LD   HL,TXT
PISE    LD   A,(HL)
        RST  #10
        HALT
        HALT
        INC  HL
        DJNZ PISE
        RET
TXT
        DEFM "  1988   EPROG 2.3 by JRM HK"
        DEFB 13
;---------------------
OPEN1
        LD   A,1
OPEN    CALL #1601
        RET
OPEN2
        LD   A,2
        JR   OPEN
;---------------------
DATAZ
        LD   A,144
DATA    OUT  (127),A
        RET
DATADO
        LD   A,128
        JR   DATA

CISTA
        LD   BC,#0A13
        CALL ATTR1
        CALL CTIEPR
        LD   BC,0
        JR   Z,ENDT
TEST
        CALL ADRESA
        IN   A,(31)
        CP   #FF
        CALL NZ,CHYB
        INC  HL
        DEC  DE
        LD   A,E
        OR   D
        JR   NZ,TEST
ENDT    PUSH BC
        CALL ODPOJ
        CALL PAZVUK
        POP  BC
        LD   A,B
        OR   C
        JR   Z,EPROK
EPRER
        CALL #2D2B
        CALL OPEN1
        LD   DE,TEPRER
        LD   BC,9
        CALL #203C
        CALL #2DE3
VEPR    CALL WAIT
        JP   ZPET
EPROK
        CALL OPEN1
        LD   DE,TEPROK
        LD   BC,KTEPR-TEPROK
        CALL #203C
        JR   VEPR
TEPRER
        DEFM " ERRORS: "
TEPROK
        DEFM "         EPROM IS O.K."
KTEPR
CHYB
        INC  BC
        RET
;---------------------

READ
        LD   BC,#0C02
        CALL ATTR1
        CALL CTIEPR
        JP   Z,ZPET
JP      Z,ZPET
```

```
        CALL ODRAM
        JP   C,ZPET
CTU
        CALL ADRESA
        IN   A,(31)
        LD   (IX+0),A
        INC  HL
        INC  IX
        DEC  DE
        LD   A,E
        OR   D
        JR   NZ,CTU
        RET
;-----------------------

TVRF
        DEFB 21,0,20,0,22,18,2
        DEFM "ADR  OK ERR"
TVAD
        DEFB 22,20,2
VERIFY
        LD   BC,#1002
        CALL ATTR1
        CALL DARST
        LD   DE,TVRF
        LD   BC,TVAD-TVRF
        CALL #203C
        LD   HL,0 ; CHYB
        LD   (#5B19),HL
        CALL CTIEPR
        JP   Z,ZPET
        CALL ODRAM
        JP   C,ZPET
COMP
        CALL ADRESA
        IN   A,(31)
        LD   B,(IX+0)
        CP   B
        JR   NZ,COERR
NOTBRK
        INC  IX
        INC  HL
        DEC  DE
        LD   A,E
        OR   D
        JR   NZ,COMP
ENDVRF
        CALL DACALL
        LD   HL,(#5B19)
        PUSH HL
        POP  BC
        JP   ENDT
COERR
        PUSH DE
        PUSH IX
        PUSH HL
        PUSH AF
        PUSH BC
        PUSH HL
        LD   HL,(#5B19)
        INC  HL
        LD   (#5B19),HL
        LD   A,2
        CALL #1601
        LD   DE,TVAD
        LD   BC,3
        CALL #203C
        POP  BC
        CALL BCHEXP
        LD   A,32
        RST  16
        POP  BC
        LD   (#5C94),BC
        LD   B,2
        CALL CTYRI
        LD   A,32
        RST  16
        POP  BC
        LD   (#5C94),BC
        LD   B,2
        CALL CTYRI
        CALL WAIT
        LD   B,5
        CALL PRR
        POP  HL
        POP  IX
        POP  DE
        CP   #20
        JR   Z,ENDVRF
        JR   NOTBRK
ATTR1
        LD   A,22 ; AT
        RST  16
        LD   A,B
        RST  16
        LD   A,C
        RST  16
        LD   A,20 ; INVERSE
        RST  16
        LD   A,1
        RST  16
        LD   A,21 ; OVER
        RST  16
        LD   A,1
        RST  16
        LD   A,32
        RST  16
        RET
;-----------------------
;$L+

UPCMOS
        SET  4,(HL) ; D4=1 => Vpp=12,5V po D3=0
        RET
UP6V
        RES  6,(HL) ; D6=0 => Vcc=6V
        RES  0,(HL) ; D0=A14=0 pri zahajeni
        RET
CENUL
        RES  2,(IY+$1B) ; CE=0 pro 2764 a 27128
        LD   A,(BYT)
        OUT  (223),A
        HALT
        RET

PROG
        LD   BC,#0E13
        CALL ATTR1
        LD   HL,BYT
        LD   A,(HL) ; #6F
        OUT  (223),A ; ZAP
        PUSH HL
        CALL CEKEJ ; po 5V
        POP  HL
        LD   A,(#5B01)
        CP   0
        CALL Z,UPCMOS
        LD   A,(#5B00)
        CP   2
        CALL Z,UP6V
        LD   A,(HL)
        OUT  (223),A
        HALT
        HALT
        RES  3,(HL)
        LD   A,(HL)
        OUT  (223),A ; Vpp
        HALT
        HALT
        CALL BUDE
        JP   Z,KONPRG
        CALL ODRAM
        JP   C,KONPRG
        LD   IY,#5B00
        LD   A,(#5B00)
        CP   2
        CALL NZ,CENUL
ZACPAL
        LD   (IY+7),0
JESTE
        LD   A,#7F ; BREAK?
        IN   A,(#FE)
        RRA
        JR   C,NOTBR
        JR   KONPRG
NOTBR   CALL ZOBRAZ
        CALL NASTAV
        CP   #FF
        JR   Z,DALE
        OUT  (31),A
        INC  (IY+7)
        CALL PULZ1
        CALL DATAZ
        CALL ADRESA
        CALL DOBRE
        JR   NZ,OPAKUJ
PALPAL
        CALL NASTAV
        OUT  (31),A
        CALL PULZ2
DALE    DEC  DE
        LD   A,E
        OR   D
        JR   NZ,DALADR
KONPRG  LD   A,#6F
        LD   (BYT),A
        OUT  (223),A
        LD   IY,#5C3A
        HALT
        JP   ZPET
NASTAV
        CALL DATADO
        CALL ADRESA
        LD   A,(IX+0)
        RET
DALADR
        INC  HL
        INC  IX
        JR   ZACPAL
OPAKUJ
        LD   A,(IY+6) ; MAX.
        CP   (IY+7)
        JR   Z,PALPAL
        JR   JESTE
PULZ1
        DI
        SET  1,(IY+$1B) ;OE/=1
        LD   A,(#5B00)
        CP   2
        LD   A,(BYT)
        JR   Z,PULCE
        AND  #FE
        OUT  (223),A
        CALL MS
        OR   #01
KOPUL   OUT  (223),A
        EI
        RET
PULCE   AND  #FB
        OUT  (223),A
        CALL MS
        OR   #04
        JR   KOPUL
PULZ2
        CALL KRAT2
        LD   A,(#5B00)
        CP   2
        CALL NZ,KRAT2
        LD   B,(IY+7)
TVRD    CALL PULZ1
        DJNZ TVRD
        RET
KRAT2
        SLA  (IY+7)
        RET

MS
        PUSH BC ; 11 T
        LD   B,249 ; 7 T
VRTSE   DJNZ VRTSE ; 13*B+8
        POP  BC ; 10 T
        RET  ;10

; T = 3*E-7 sec
; + CALL + OUT
;-----------------------

DOBRE
        RES  1,(IY+$1B) ; OE=0
        CALL BYTVEN
        LD   A,(#5B00)
        CP   2
        CALL Z,CE0
        IN   A,(31)
        PUSH AF
        CALL Z,CE1
CALL    Z,CE1
        SET  1,(IY+$1B) ; OE=1
        CALL BYTVEN
        POP  AF ; z EPROM
        LD   C,(IX+0) ; z RAM
        CP   C
        RET
```

```
CE0
        RES   2,(IY+$1B)
BYTVEN  LD    A,(BYT)
        OUT   (223),A
        RET
CE1
        SET   2,(IY+$1B)
        JR    BYTVEN
ZOBRAZ
        PUSH  HL
        PUSH  DE
        PUSH  BC
        PUSH  HL
        POP   BC
        CALL  BCHEXP
        POP   BC
        POP   DE
        POP   HL
        RET
PRASC
        PUSH  BC
        SUB   32
        LD    B,3
        LD    C,0
        AND   A
ROT     RLA
        RL    C
        DJNZ  ROT
        LD    B,C
        LD    C,A
        LD    HL,$3D00
        ADD   HL,BC
PRTAB
        LD    DE,$509B
        LD    B,8
PRCHAR
        LD    A,(HL)
        LD    (DE),A
        INC   HL
        INC   D
        DJNZ  PRCHAR
        LD    HL,PRTAB+1
        LD    A,(HL)
        INC   A
        CP    $9F
        JR    NZ,NOENDL
        LD    A,$9B
NOENDL
        LD    (HL),A
        POP   BC
        RET
BCHEXP
        LD    ($5C94),BC
        LD    B,4
CTYRI   XOR   A
        LD    HL,$5C94
        RLD
        INC   HL
        RLD
        ADD   A,$30
        CP    $3A
        JR    NC,MENSI
VETSI   CALL  PRASC
        DJNZ  CTYRI
        RET
MENSI   ADD   A,7
        JR    VETSI
DARST
        LD    HL,VETSI
        LD    (HL),0
        INC   HL
        LD    (HL),0
        INC   HL
        LD    (HL),$D7
        RET
DACALL
        LD    HL,VETSI
        LD    (HL),$CD
        LD    DE,PRASC
        LD    (VETSI+1),DE
        RET
```

BOBO 64 K (3)

Ing. Josef Blahůt,

ing. Václav Daněček

Příloha 2 - seznam součástek

Číslicové integrované obvody

SN74LS00	(K555LA3)	105,108,111,305*
SN74LS04	(K555LN1)	101,107,109,204,401
SN74LS08	(K555LI1)	106,207
SN74LS10	(K555LA4)	304,406
SN74LS20	(K555LA1)	203*
SN74LS32	(K555LL1)	402,407
SN74LS74	(K555TM2)	202*,303*
SN74LS86	(K555LP5)	110
SN74LS90	(K555IE2)	302*
SN74LS93	(K555IE5)	201*,205*,206*,301*,306*, 307*
SN74LS138	(K555ID7)	303,405
SN74LS174	(K555TM9)	104,313
SN74LS175	(K555TM8)	404
SN74LS193	(K555IE7)	102
SN74LS244	(K555AP5)	606,607
SN74LS257	(K555KP11)	208*,303*,309*,502,605
SN74LS295	(K555IR16)	310,311,312
SN74LS298	(K555KP13)	209,210,211
SN74LS373	(K555IR22)	408

Pozn.: Obvody označené * nemusí být typu LS

MH74S00		112
MH74188		103
Z80A CPU	(UA 880)	602
MHB8251		501
MHB8255A		601
HYA 4164	(K565RU5)	503,504,505,506,507,508,509, 510
I2764	(K574RF4)	603,604
(I27128		603)

Tranzistory

KC 507	T2,T3
KC 508	T1,T5
KF 507	T6
KSY 71	T7
KSY 81	T4

Diody

KA 206	D1,D18,D21,D23,D24,D25
GA 201	D2 ÷ D20
LQ 100	D22

Ostatní

MAB 311	IO1
WK 164 12	OP1
TX 787 5401	601,602 (objímka 40 vývodů)
TX 787 5281	603,604 (objímka 28 vývodů)
TY 513 301 1/30	K1,K2,K3,K4,K5 (FRB konektor)
krystal 14 MHz	Xtal

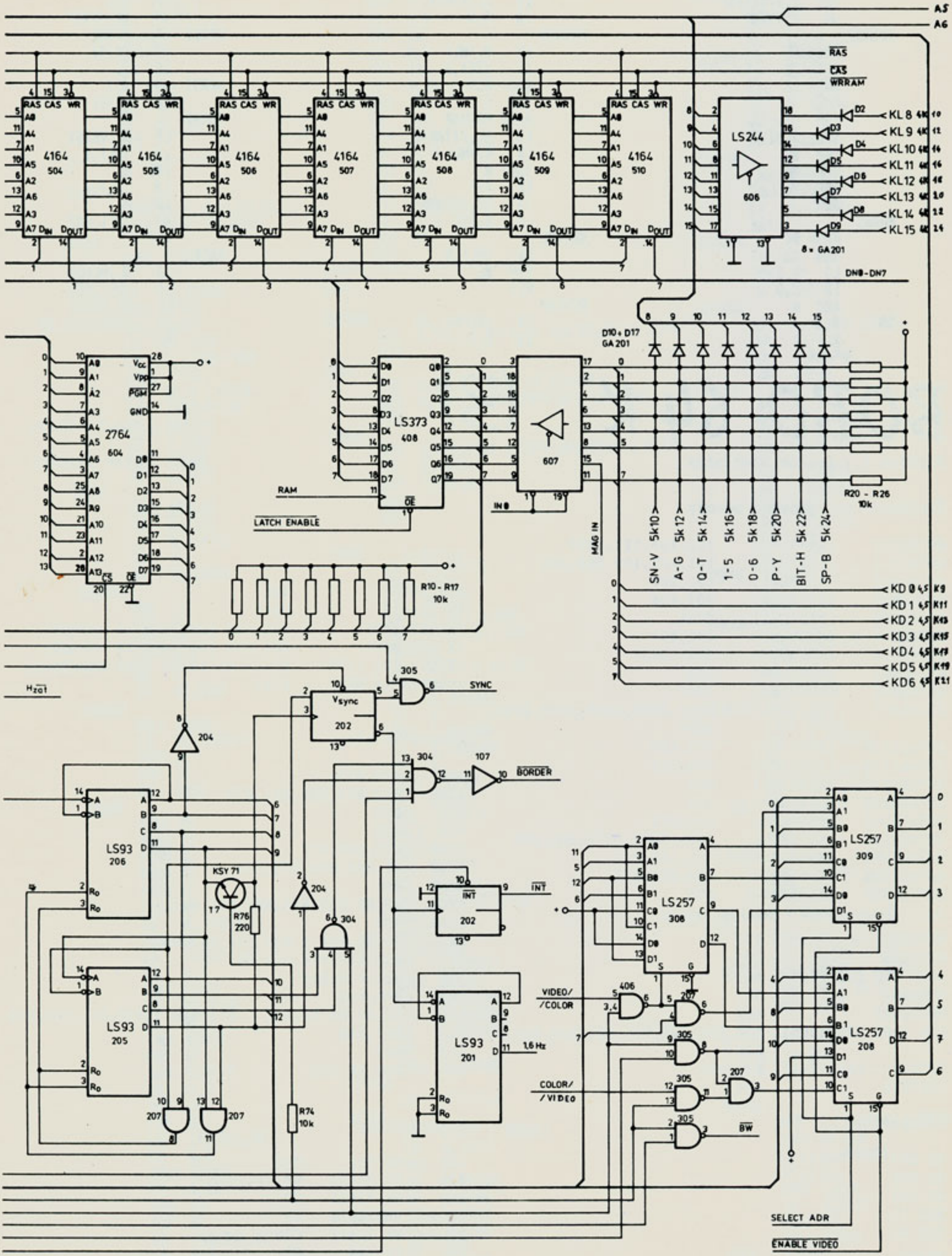
Odpory TR 191

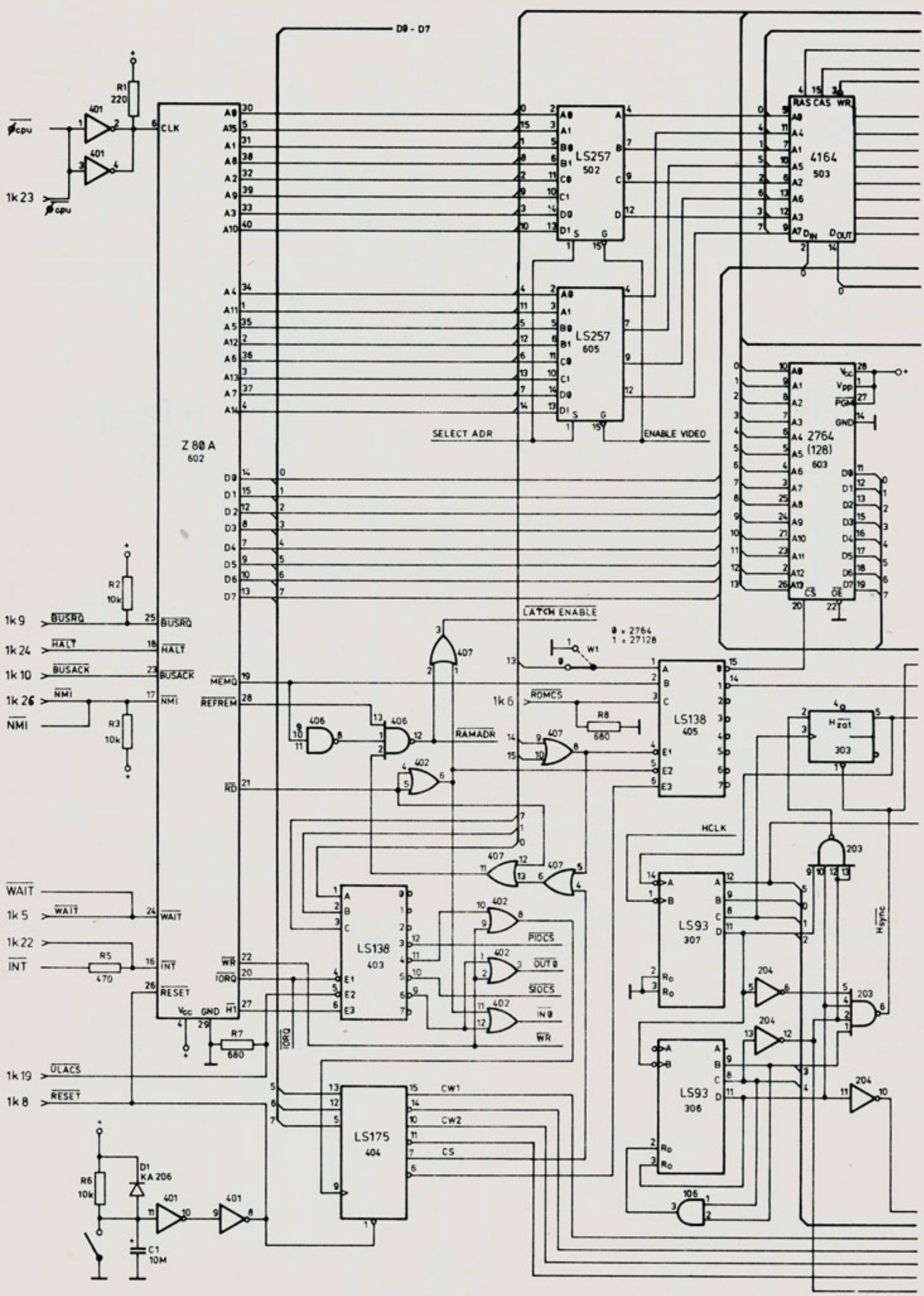
10R	R42,R75
22R	R52,R53,R54.R64,R73
27R	R43
47R	R62
100R	R18,R69,R74
220R	R1,R68,R70,R71,R76
330R	R61
470R	R5,R9,R29,R40,R55,R65,R72
680R	R56
1k	R7,R8,R30 ÷ R37,R39,R41,R57 R58,R59,R60,R63,R67
1k5	R27,R28,R66
3k3	R44,R51
10k	R2,R3,R4,R6,R10 ÷ R17,R19 ÷ R26,R45,R46,R38
M1	R47,R48
M47	R49
4M7	R50 TR 192

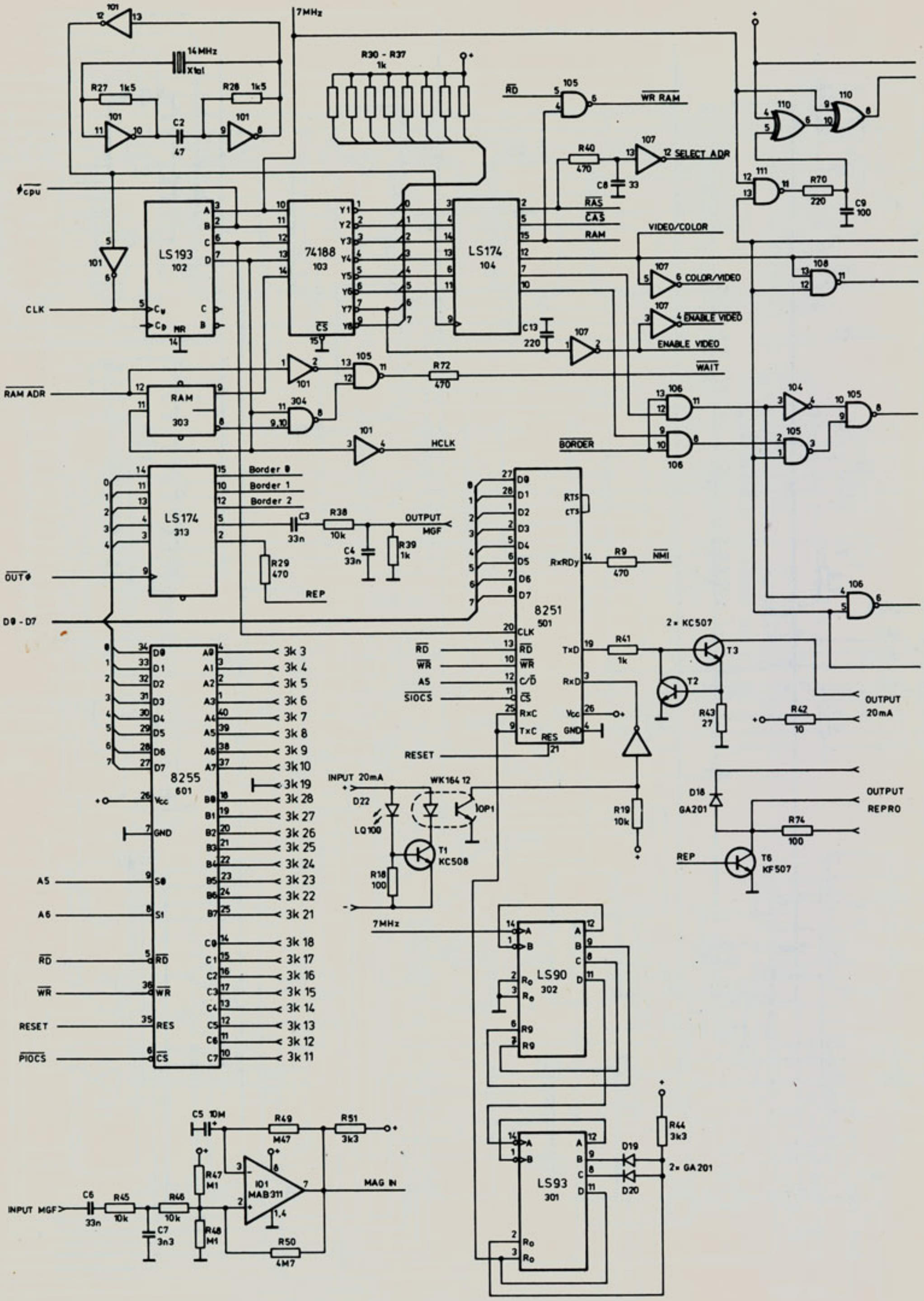
Kondenzátory

10p	Tk 754	C10
33p	Wk 71411	C8,C11,C12
47p	Wk 71411	C2
100p	Wk 71411	C9
220p	Wk 71411	C10
3n3	Tk 744	C7
33n	Tk 782	C4,C6,C3
47n	Tk 782	C20 ÷ C58
10M	TE 981	C1
47M	TE 121	C5,C16,C17,C18,C19

Příloha 1a - schema zapojení




D0 - D7
R1 220
401
401
1k 23
CLK
Z 80 A 602
A0 A15 A1 A8 A2 A9 A3 A10
A4 A11 A5 A12 A6 A13 A7 A14
D0 D1 D2 D3 D4 D5 D6 D7
LS257 502
LS257 605
SELECT ADR
ENABLE VIDEO
4164 503
RAS CAS WR
2764 (128) 603
R2 10k
1k 9 BUSRQ
1k 24 HALT
1k 10 BUSACK
1k 26 NMI
NMI
R3 10k
MEMQ
REFREM
RD
LATCH ENABLE
0 = 2764
1 = 27128
W1
407
406
406
RAMADR
1k 6 ROMCS
R8 680
LS138 405
402
402
HCLK
203
WAIT
1k 5 WAIT
1k 22
INT
R5 470
RESET
WR
IORQ
M1
Vcc GND
R7 680
LS138 403
PIOCS
OUTS
SIOCS
IN
WR
LS93 307
LS93 306
204
204
203
203
Hzat 303
Hsync
1k 19 ULACS
1k 8 RESET
LS175 404
CW1
CW2
CS
R6 10k
D1 KA 206
401
401
C1 10M
106


7MHz
14MHz
Xtal
R27 1k5
R28 1k5
C2
47
R30 - R37
1k
RD
WR RAM
R40
470
C8
33
SELECT ADR
R70
220
C9
100
VIDEO/COLOR
COLOR/VIDEO
ENABLE VIDEO
WAIT
RAS
CAS
RAM
CPU
CLK
LS193
102
74188
103
LS174
104
C13
220
R72
470
RAM ADR
RAM
303
HCLK
BORDER
Border 0
Border 1
Border 2
LS174
313
C3
33n
R38
10k
C4
33n
R39
1k
OUTPUT
MGF
R29
470
REP
OUT0
D0 - D7
8251
501
RTS
CTS
RxRDy
R9
470
NMI
CLK
RD
WR
A5
SIOCS
RESET
TxD
RxD
Vcc
GND
RES
R41
1k
2x KC507
T3
T2
R43
27
R42
10
OUTPUT
20mA
8255
601
A5
A6
RESET
PIOCS
3k 3
3k 4
3k 5
3k 6
3k 7
3k 8
3k 9
3k10
3k19
3k28
3k27
3k26
3k25
3k24
3k23
3k22
3k21
3k18
3k17
3k16
3k15
3k14
3k13
3k12
3k11
INPUT 20mA
WK164 12
D22
LQ100
T1
KC508
R18
100
R19
10k
7MHz
D18
GA201
R74
100
OUTPUT
REPRO
REP
T6
KF507
LS90
302
LS93
301
R44
3k3
D19
D20
2x GA201
C5 10M
R49
M47
R51
3k3
R47
M1
R48
M1
IO1
MAB311
MAG IN
INPUT MGF
C6
33n
R45
10k
R46
10k
C7
3n3
R50
4M7

DM0 - DM7

LS295 310

LS295 312

LS298 210

LS298 211

LS298 209

Border 0

Border 1

Border 2

CLK

BW

BORDER

1,6Hz

R52 22 R

R53 22 G

R54 22 B

R73 22 Y

R55 470

R56 680

T4 KSY81

R63 1k

R60 1k D23

R61 330 D24

R62 47 D25

R71 220

3× KA206

R57 1k

R58 1k

R59 1k

T5 KC508

SYNC

R65 470

R64 22 VIDEO

R68 220

C18 33

R75 10

C14 47M

C15 47n

R66 1k5

R69 100

C10 10

D21 KA206

C11 33 VHF

R67 1k

Obsazení konektoru 2K

1		2	
3	A15	4	A14
5	A13	6	A12
7	A11	8	A10
9	A9	10	A8
11	A7	12	A6
13	A5	14	A4
15	A3	16	A2
17	A1	18	A0
19		20	
21	D1	22	D0
23	D3	24	D2
25	D5	26	D4
27	D7	28	D6
29		30	

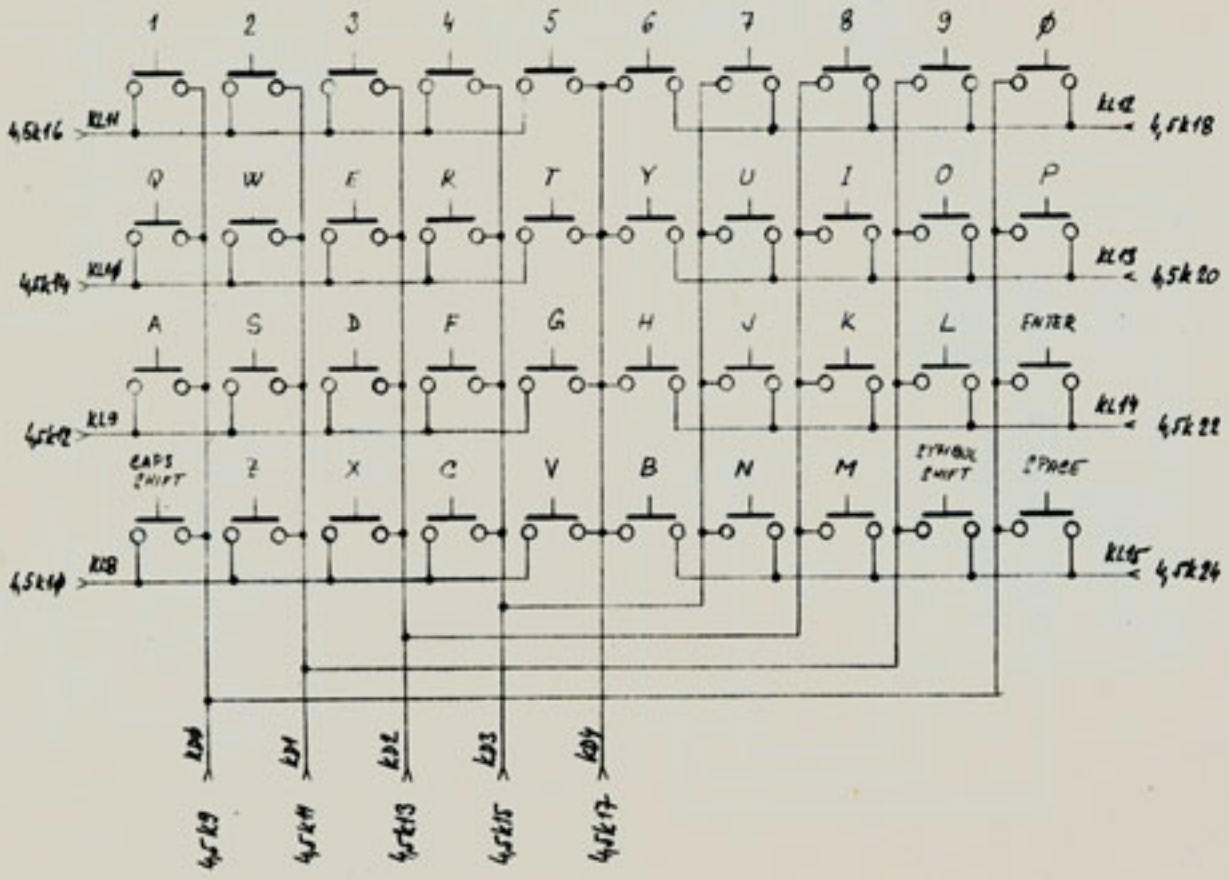
Obsazení konektoru 1K

1		2	
3		4	
5	WAIT	6	ROM CS
7	REFRESH	8	RESET
9	BUSRQ	10	BUSACK
11		12	
13		14	
15		16	RD
17		18	WR
19	ULA CS	20	M1
21	IORQ	22	INT
23	ΦCPU	24	HALT
25	MEMQ	26	NMI
27	ZEM	28	+5V
29		30	

Příloha 3 - obsah řídící paměti PROM MH74188

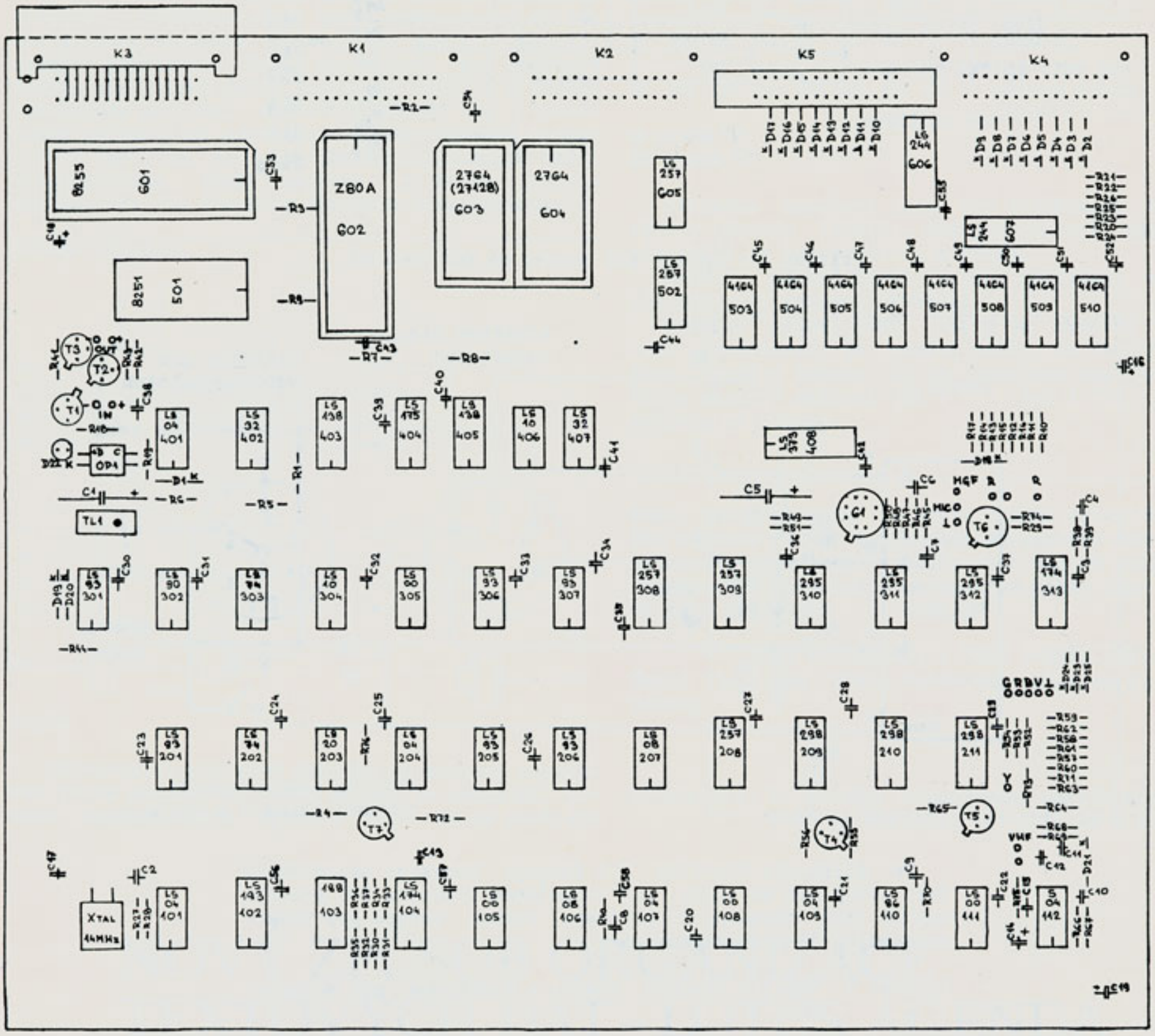
	Adresa					Výstup Y							
	E	D	C	B	A	8	7	6	5	4	3	2	1
0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	1	1	0	0	1	0	0	0	0
2	0	0	0	1	0	1	0	0	0	1	0	1	0
3	0	0	0	1	1	0	0	0	0	1	0	1	0
4	0	0	1	0	0	1	0	0	0	1	0	0	0
5	0	0	1	0	1	1	0	1	0	1	0	0	0
6	0	0	1	1	0	1	1	0	0	0	0	0	1
7	0	0	1	1	1	1	1	0	0	0	0	1	1
8	0	1	0	0	0	0	1	0	0	0	1	1	0
9	0	1	0	0	1	0	1	0	0	0	1	0	0
10	0	1	0	1	0	0	1	0	0	0	1	0	0
11	0	1	0	1	1	0	1	0	0	0	1	0	0
12	0	1	1	0	0	1	0	0	0	0	0	1	1
13	0	1	1	0	1	1	0	0	0	0	0	1	1
14	0	1	1	1	0	1	0	0	0	0	0	1	0
15	0	1	1	1	1	1	0	0	0	0	0	0	0
16	1	0	0	0	0	1	0	0	0	0	0	0	0
17	1	0	0	0	1	1	0	0	1	0	0	0	0
18	1	0	0	1	0	1	0	0	0	1	0	1	0
19	1	0	0	1	1	1	0	0	0	1	0	1	0
20	1	0	1	0	0	0	0	0	0	1	0	0	0
21	1	0	1	0	1	1	0	1	0	1	0	0	0
22	1	0	1	1	0	1	1	0	0	0	0	0	1
23	1	0	1	1	1	1	1	0	0	0	0	1	1
24	1	1	0	0	0	1	1	0	0	0	0	1	1
25	1	1	0	0	1	1	1	0	0	0	0	1	1
26	1	1	0	1	0	1	1	0	0	0	0	1	1
27	1	1	0	1	1	1	1	0	0	0	0	1	1
28	1	1	1	0	0	1	0	0	0	0	0	1	1
29	1	1	1	0	1	1	0	0	0	0	0	1	1
30	1	1	1	1	0	1	0	0	0	0	0	1	0
31	1	1	1	1	1	1	0	0	0	0	0	0	0

Příloha 5 - zapojení klávesnice ZX Spectrum



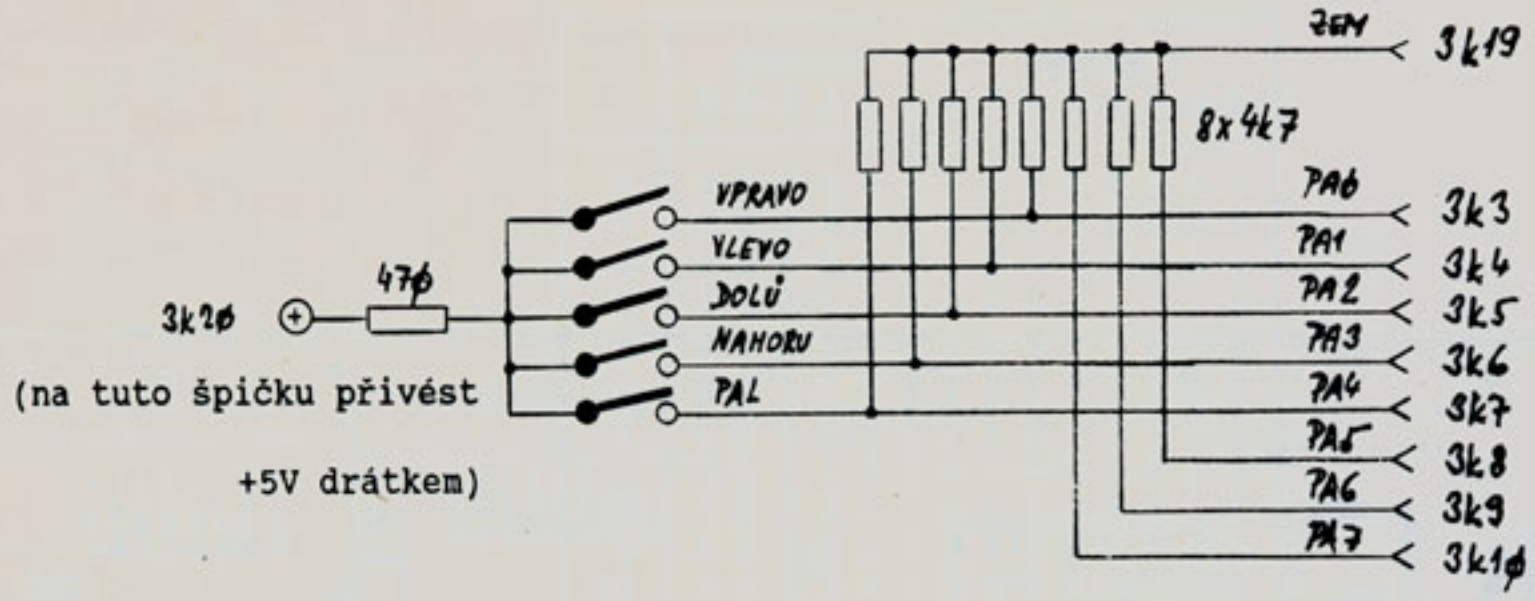
První ovladač Spectrum se připojí jako tlačítka 6,7,8,9,0 a druhý jako 1,2,3,4,5.

Příloha 4 - rozmístění součástek na desce

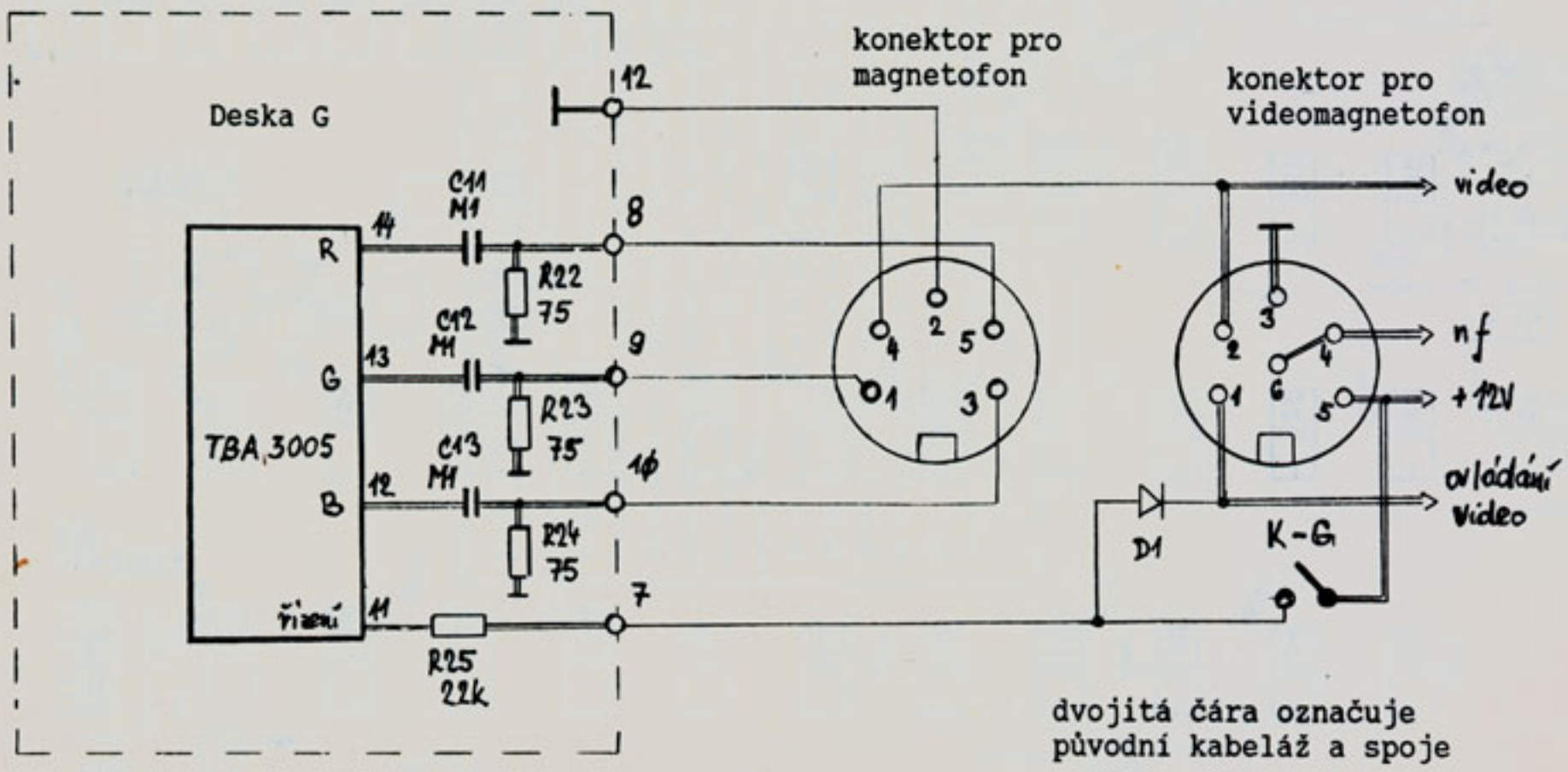


Pozn. red. - V současné době mají autoři hotovou úpravu BOBO 64 pro paměti 256Kb. Videopaměť pak vůbec nezasahuje do základních 64KB RAM pro procesor. RAMdisk má 172KB. Dále mají připraven k BOBO diskový řadič pro mechaniku 5¼". Kromě toho mají "v šuplíku" řadu velice zajímavých konstrukcí, někdy i jen užitečných drobností. To vše bychom rádi časem také otiskli.

Příloha 6 - připojení ovladače typu Kempston



Příloha 7 - úprava BTV Oravan na RGB monitor



Po stisknutí tlačítka K-G funguje konektor magnetofonu jako vstup RGB monitoru. Při rozpojeném tlačítku K-G pracuje ORAVAN jako normální TV přijímač. Funkce vstupu pro videomagnetofon je plně zachována.

Úpravy: 1. Odpojit všechny vodiče z konektoru pro magnetofon.

2. Odpojit tlačítko K-G (u BTV Mánes je možné využít tlačítko pro odpojení reproduktoru).

3. Na desku G osadit kondenzátory C11÷C13 a odpory R22÷R25 (na desce jsou pro ně připravené pozice).

4. Na základní desku osadit konektorové špičky 7,8,9,10 pro desku G (špičky lze vypájet z testovacích bodů desky G).

5. Kablíkem PNLY 5x0,30, délky asi 30 cm, propojit magnetofonový konektor se základní deskou dle schematu.

6. Propojit synchronizaci a řízení na konektor videomagnetofonu a tlačítko K-G podle schematu (dioda D1 slouží k otevření video vstupu pro sync. signál).

Nové zapojení konektoru
pro magnetofon: 1 - G
2 - ZEM
3 - B
4 - VIDEO (vlastně sync. směs)
5 - R

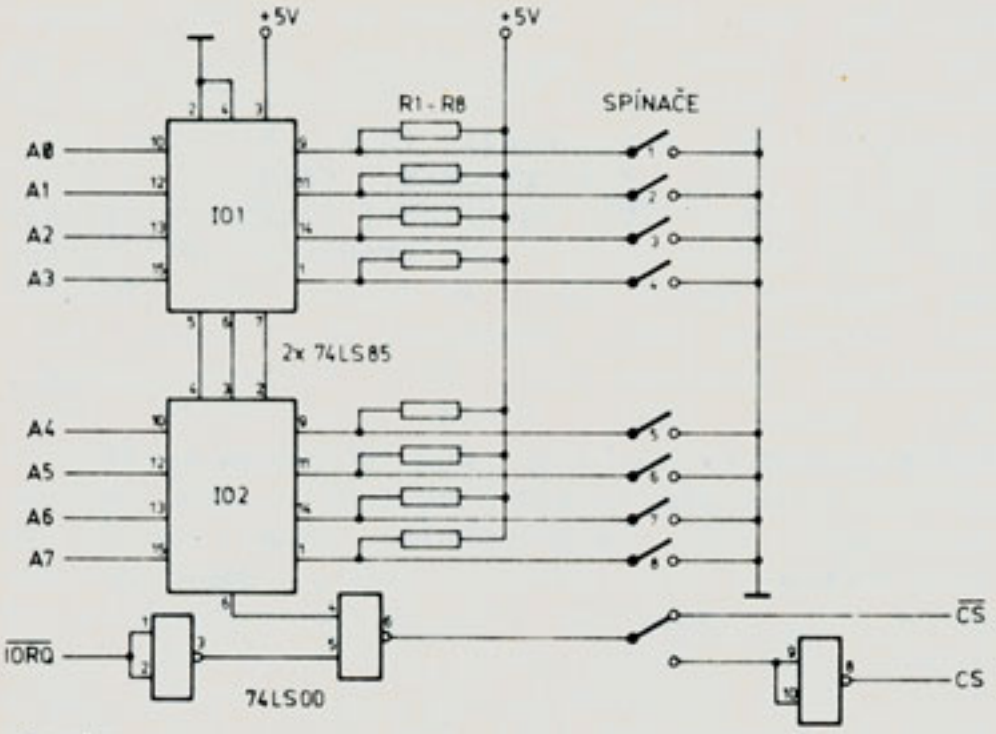
KONEC

UNIVERZÁLNÍ ADRESOVÝ DEKODÉR

Miroslav Bořík

Každý správný fanda počítačové techniky časem dospěje k tomu, že ho přestávají zajímat hry, začne shánět systémové a užitkové programy, návody a plánky na hardwarové doplňky a vylepšení svého počítače. Sám jsem se dostal do tohoto stadia a zjistil jsem, že hry se shánějí mnohem snadněji. Počítačový nadšenec má v tomto směru velmi ztíženou situaci vzhledem k omezenému množství dosažitelných publikací. Proto bych chtěl mně podobným fandům poskytnout další námět. Nejedná se o žádnou převratnou novinku, ale o užitečnou věc - jednoduchý univerzální adresový dekodér. Původně určený pro ZX Spectrum, ale použitelný i pro mnohé další mikropočítače. Námět pochází z knihy Lothara Schlüsslera "ZX Spectrum Hardware-Erweiterungen".

Schema zapojení



Popis

Univerzální adresový dekodér je schopen adresovat různé periferie. Volba adresy je manuální, pomocí osminásobného spínače DIL. Dá se nastavit jakákoliv adresa v rozsahu 00 - FF (hex.). Všechny adresy se vybírají signálem /IORQ a příkazy IN nebo OUT. Podle polohy spínačů lze vytvořit adresu např.:

A7	A6	A5	A4	A3	A2	A1	A0	adresový vodič
S8	S7	S6	S5	S4	S3	S2	S1	spínač DIL
ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF	nastavená hodnota 127(dek.)

ON znamená "0"
OFF znamená "1"

Základem zapojení jsou dva čtyřbitové komparátory (obvody 74LS85). Obě porovnávaná slova se přivádějí na vstupy A0÷A3 a B0÷B3.

Hodnoty odporů R1÷R8 nejsou kritické, vyhoví zde odpory v rozsahu 3k9÷10k.

Protože signál IORQ, který je aktivní "0" se musí invertovat, je použit další IO 74LS00. Pro plné využití tohoto IO je navrženo zapojení pro "Chip select" aktivní jak v "0", tak i "1" - podle potřeby uživatele.

Pozn. red.: Ne vždy je zcela vhodné vytvářet součin signálu /IORQ s výstupem adresového dekodéru, zejména v systémech s oddělovači datové sběrnice. Požadovaným dynamickým parametrům někdy lépe vyhoví tyto signály:

$\overline{IORD}$ (=$\overline{IORQ.RD}$) a $\overline{IOWR}$ (=$\overline{IORQ./WR}$).

Často je ovšem třeba dekódovat skupinu (2,4,16) adres.

Literatura: P. Horský: Připojování periferních obvodů ke sběrnici (e) STD. AR A 6-8/86

NOVÝ ZPŮSOB ORGANIZACE TEXTOVÝCH DAT - LOTUS AGENDA

Ota Luňák

Lotus Agenda je novým programovým produktem firmy Lotus. Oficiální název je Agenda - The Information Manager, v podstatě se však jedná o systém retrospektivní organizace textových dat. Agenda umožňuje uživateli uspořádat data podle jakéhokoliv myslitelného způsobu a to během, nebo až po vstupu dat. Odpadá tak nutnost definovat předem strukturu databáze jako u klasických databází typu dBASE III.

Strukturu Agendy tvoří POLOŽKA, KATEGORIE, POZNÁMKA a FORMÁT (VIEW). Základní jednotkou je POLOŽKA, kterou může tvořit textová informace o maximální délce 350 znaků a která může být přiřazena libovolnému počtu Kategorií. Kategorie s přiřazenými položkami tvoří SEKCI. KATEGORIE zajišťují strukturalizaci Položek a lze je obhospodařovat pomocí programové funkce Category Manager. POZNÁMKA obsahuje dodatečné informace o Položce či Kategorii a může obsahovat až 10 KB dat. FORMÁT zajišťuje volbu formátu zobrazení dat s možností jejich dynamické modifikace.

Do setříděných dat horizontálně zobrazených formou kategorií a položek lze vkládat SLOUPCE, které data dále rozčlení, nebo lze do Sloupců vložit data nová. Pokud data přiřazená určité kategorii změníme, Agenda je přiřadí automaticky odpovídající kategorii. Pokud například máme v kategorii "Telefony" položku "Telefonovat Petrovi v pátek" a definujeme sloupec "Kdy" s datovým formátem, Agenda do sloupce "Kdy" doplní skutečné datum. Data lze importovat a exportovat v mnoha formátech včetně WordPerfect a dBASE III.

Agenda je bezesporu revolučním způsobem dynamické organizace textových souborů a má všechny předpoklady stát se stejně populární, jako svého času Lotus 1-2-3. Podrobnější informace o programu Agenda najdete v časopisu PC Magazine (UK) 10/88. Přehled databázových systémů najdete v časopisu PC Magazin (US) (Středisko VTI Svazarmu, Praha 1, Martinská 5).

MONITOR informací střediska VTI SVAZARMU

**Středisko
vědeckotechnických informací Svazarmu
pro elektroniku
Martinská 5, 110 00 Praha 1**

Služby střediska

Jsou poskytovány pouze osobně. Vyřizování členství a hostování v 602.ZO Svazarmu, přístup ke knihovně časopisů na mikrofiších, pořizování ozalitových kopií z knihovny časopisů, prodej programových produktů Mikrobáze, zpravodaje Mikrobáze, nepájivých kontaktních polí, technické poradenství pro výrobky firmy AMSTRAD a poskytování informací o odborných akcích Svazarmu.

Pracovní doba

pondělí		zavřeno
úterý až čtvrtek	10 - 12	14 - 17
pátek	10 - 12	14 - 16
AMSTRAD pondělí až pátek	8 - 10	

>>>>>>> telefon 22 87 74 <<<<<<<

Seznam časopisů na mikrofiších

64'ER-Das Mag. fuer comp.fans (DE), ACM Siggraph: Comp. Graphics (US), Applied Mathematical. Modelling (GB), Bajtek (PL), Byte (US), Chip-Das Mikrocomp. Magazin (DE), Communication News (US), Communications of the ACM (US), Computer (IEEE) (US), Computer Aided Design (GB), Computer Design (US), Computer Graphics And Applicat. (US), Computer Journal (GB), Computer Networks (NL), Computing Reviews (US), Comsat Technical Review (US), Desktop publishing World (GB), Datamation (GB), Electr. Sound+Rte (CH), Electr. and Wirel. World (GB), Electri - onics (US), Electronics - Int. Edition (US), Elektor (DE), Elektronik (DE), Elektronique Praktique (F), Elo (DE), Elrad (DE), Ezermester (MLR), Funkamateur (NDR), Funkschau (DE), Happy Computer (DE), Hifi News and Rec. Review (GB), Hobby (Magazin der Technik) (DE), IBM Journal of R & D (US), IEEE Trans. Circ. Systems (US), IEEE Trans. Syst. Man Cybern. (US), IEEE Trans. on Softw. Engineering (US), Industrial Robot (GB), Industrial and Proc. Control Mag. (GB), Information and softw. technol. (GB), Intern. Business Equipment (BE), Journ. Acous. Soc. Amer. (US), Journ. Parall. Programming (US), Journ. of The Aud. Eng. Soc. (US), Kompjutr za vas (BG), Komputer (PL), Laser & Applications (US), MC - Die Mikrocomp. Zeitschrift (DE), Microelectr. and Reliability (GB), Mikromagazin (MLR), Mikrodok (DE), Mikroklan (PL), Mikroprocess. Sredstva I Sist. (SU), Moj Mikro (YU), Nachricht. Elektr. + Telematik (DE), Office Equip. and Products (JP), Personal Computer World (GB), PC Magazin (DE), Practical Computing (GB), Practical Electronics (GB), Practical Wireless (GB), RE - Radioelektronik (PL), Radio (SU), Radio Electronics (US), Radio-Amater (YU), Radio und Fernsehen Elektronik (NDR), Radio-Televizija-Elektronika (BG), Radioelektronik (PL), Rádiótechnika (MLR), Revija za mala računala (YU), Robotica (GB), Siemens R&D Reports (DE), Simulation (US), Software-Practice and Exper. (GB), Solid State Communications (GB), Solid State Technology (US), TB - Report (DE), Techniky Komputerowe (PL), The Office (GB), Toshiba Review (JP), ZX Computing Monthly (GB)

Použité zkratky:

US-USA, GB-Velká Británie, YU-Jugoslávie, JP-Japonsko, CH-Švýcarsko, BG-Bulharská lid. republika, F-Francie, BE-Belgie, DE-NSR, SU-SSSR

Knihy na mikrofiších

Forth (programovací jazyk - manuál v češtině) * Příručka jazyka Basic pro ZX Spectrum (kniha v češtině) * Przewodnik pro ZX Spectrum (v polštině) * Úvod do programování ve strojovém jazyku ZX 81, ZX Spectrum (kniha v češtině) * ZX Microdrive + Interface I (kniha v angličtině) * ZX Spectrum I + II (kniha v angličtině)

..... Informace ze zahraničí

Ota Luňák

AMSTRAD PPC640 SPRINT

je osobním přenosným počítačem pro náročné. SPRINT je totiž název pevného disku formátu 3.5" s kapacitou 20 megabajtů, kterým lze nahradit jednu ze dvou diskových mechanik populárního přenosného počítače Amstrad. Technická data:

přístup	- 25 msek
přenosová rychlost	- 250 kb/sek
dovolené přetížení	- 10 G při činnosti
	75 G pro transport

A cena? 550 GBP, tedy téměř cena počítače PPC640 DD.

ATARI FOLIO - pécéčko do kapsy

Folio je novým hitem firmy Atari. Vyznačuje se speciální verzí MS-DOSu, která je spolu s užitkovými programy umístěna v paměti ROM. Folio si "umí" zapamatovat aplikaci, a tak po zapnutí počítače se ocitnete přesně tam, kde jste přestali pracovat. Data jsou uchovávána na výměnné paměťové kartě RAM (64 kilobajtů = 50 GBP). Pro práci s počítačem Folio je možné využít všechna standardní programová vybavení pracující pod operačním systémem MS-DOS. Slabou stránkou však je displej - zobrazující 40 znaků v osmi řádcích. A cena? 200 GBP.

ACER 1100/33

je nový osobní počítač největšího taiwanského výrobce osobních počítačů, firmy ACER (dříve MULTITECH). Firma ACER je zajímavá tím, že si vyrábí všechny potřebné komponenty pro sestavy osobních počítačů sama. Svým prodejním obratem (předpokládá se 1 miliarda USD v roce 1990), zaujala americkou firmu Intel tak, že první verze mikroprocesoru 80386 s frekvencí 33 MHz směřovaly na Taiwan. Systém je vybaven zápisníkovou (cache) pamětí (kontrolér 82385 / 33 MHz) s kapacitou 64 kilobajtů (20 ns). Operační paměť lze rozšířit ze 4 megabajtů na 24 MB. Grafický adaptér VGA (800 x 600 / 16 barev), mechanika 5 1/4" / 1,2 MB a ESDI hard disk 140 MB tvoří základní výbavu ACERu. Typ 1100/33 představuje vrchol řady 80386 a naznačuje, že nástup procesorů typu RISC lze předpokládat dříve, než se obecně předpokládalo. Bližší informace: PCW 5/89 - str. 134.

2.5 Gigabajtů na videokazetě

nabízí firma Digi-Data Corporation. Zařízení nazývané Gigastore je schopné zaznamenat 2.5 GB na kazetu T-120 VHS a to i ve spojení s počítači typu IBM PC.

SINCLAIR PC200

Pro kategorii domácích počítačů zachovala firma AMSTRAD známou obchodní značku Sinclair. PC200 má být stejně oblíbený, jako známý typ ZX Spectrum. Nový počítač je určen pro kategorii profesionálních domácích počítačů. Je standardu IBM PC (16bitový) a je vybaven klávesnicí typu AT (102 tlačítek) a výstupem na TV přijímač. Může sloužit nejen pro zábavu, ale i pro vážnější využití v mnoha oblastech. Pokud nechcete zatěžovat domácí TV přijímač, můžete PC200 zakoupit s barevným či monochromatickým monitorem - pak je součástí sestavy zdarma i joystick a programové vybavení (4 hry a PC Organizer). Data jsou uchovávána na vestavěné disketové jednotce 3 1/2" (720 Kb). Cena je 327 GBP (MEDIA, sklad Praha).

AMSTRAD PC 2000

Nová řada profesionálních šestnáctibitových a dvaatřicetibitových počítačů se skládá ze tří typů:

PC 2086 - mikroprocesor 8086 / 8MHz - 640KB RAM
PC 2286 - mikroprocesor 80286/12MHz - 1MB RAM
PC 2386 - mikroprocesor 80386/20MHz - 4MB RAM

Všechny typy používají novou řadu monitorů AMSTRAD VGA. Pomocí sítě AMSTRAD NETWORK využívající operační systém AMSNOS je možné vzájemné propojení všech počítačů AMSTRAD s počítači jiných typů a velikostí. Další zajímavou novinkou je zdvojení vnitřní sběrnice pro zrychlení přenosu dat mezi počítačem a periferiemi a vybavení pevných disků vyrovnávací pamětí. Tak je možné data načíst během jediné otáčky - klasická zařízení potřebují ke stejné operaci 2 až 3 otáčky. Také 64KB rychlé tzv. "zápisníkové" paměti umožňuje procesoru nepřetržitou činnost. Data jsou uchovávána na pružných discích 3½". Je však možné připojit vnější jednotku s klasickým formátem 5¼". Nové nároky na zpracování informací zvyšují nároky také na výrobce. Firma AMSTRAD je tedy opět vpředu - nejen nákupem licence na OS/2 od firmy IBM, ale také aplikací revolučních konstrukčních prvků při udržení nízké cenové hladiny.

Paralelní rozložené zpracování dat

V originále Parallel Distributed Processing je stále častějším tématem odborné literatury. S nástupem transputerů se objevuje množství literatury, zabývající se uvedenou tématikou. Do problematiky výběru vhodné literatury z této oblasti vás zavede rubrika BiblioFile časopisu PCW 9/88.

Motorola 6845 CRT

Popis uvedeného obrazového řadiče (použit v počítačích Amstrad CPC) včetně návodu k programovému nastavení požadovaných parametrů je zveřejněn v časopise PCW 9/88 na str. 170.

Kolik stojí "hardisk"?

Britská firma Kudos (Colindale Business Park, Carlisle Road, London NW9 0HW) prodává hardisk 20 MB za 169 GBP. 30MB stojí 179 GBP a 60MB 315 GBP. Streamer (zálohovací kazetopásková jednotka) 40MB stojí 449 GBP.

Přenos dat mezi ATARI ST a AMSTRAD CPC 6128

lze uskutečnit pomocí textového editoru TASWORD na straně počítače CPC a specálního programu na straně ST. K přenosu potřebujete upravený (na straně ST) kabel Centronics a program pro ATARI ST. Tento program zveřejnil jugoslávský časopis Moj mikro 2/1989, str.39.

Potřebné maličkosti

pro výpočetní techniku jsou u nás stále ještě vzácností. Jedná se nejen o drobné nábytkové doplňky, ale i prvky pro uložení disket, tiskových výstupů, rozvaděče dat (tzv.data switch), speciální nářadí, čisticí prostředky, antistatické filtry a další. Rozsáhlý sortiment tohoto příslušenství nabízí britská firma ACCODATA, Nepicar House, London Road, Wrotham Heath, Kent TN15 7RS, Velká Británie.

MULTI INERFACE

pro ZX Spectrum najdete v časopisu Moj mikro 12/1988, str.10. Toto užitečné zařízení umožňuje ovládání širokého spektra periferních zařízení roboty počínaje a měřením fyzikálních veličin konče. (Moj mikro je k dispozici na mikrofiších ve středisku VTI Svazarmu, Martinská 5)

Šachy v jazyce BASIC

Program pro tuto klasickou hru je, ve formě výpisu, součástí seriálu o principech programování šachových her, který zveřejňuje jugoslávský časopis Moj mikro. Zmíněný program je obsahem březnového čísla.

Kolik vydělává Amstrad?

Roční obrat firmy Amstrad byl v roce 1988 o 22% vyšší, než v roce 1987 a činil 625.4 milionů GBP. Firma prodala 700000 domácích počítačů, 700000 počítačů kategorie PC, 400000 audio zařízení, 450000 video zařízení a 250000 tiskáren. Kéž by bylo možné vidět alespoň třetinové hodnoty u našich státních podniků!

PROGRAMOVÁ NABÍDKA

(pokračování z čísla 7)

DAM +2 V1989

Assembler pro PMD 85-2 včetně celostránkového editoru, překladače, zpětného překladače a debuggeru s mozností krokování.

KASWORD V3.0

Textový editor pro PMD 85 s kompletním znakovým souborem s diakritikou.

KAREL V2.2

Názorný výukový programovací jazyk s pohodlným editorem, klíčovými klávesami, slovníkem a možností výpisů na tiskárně.

GRUTI

Základní grafické rutiny pro PMD 85-2 typu CIRCLE, BOX, přímkové objekty, animace atd.

WELLCOPY

Kopirovací program pro běžné i bezhlavičkové a autostart bloky, s mazáním i záměnou pořadí bloků a změnou hlavičky.

GRED

Celostránkový grafický editor s plně okénkovým ovládáním a s možností čtení záznamů ZX Spectra.

GREP

Bodově orientovaný grafický editor pro rozměr obrázků 36x42 bodů s možností animace. Ovládání klávesnicí nebo joystickem.

MUSICA

Hudební editor s jednoduchým notovým zápisem melodie, editací včetně blokových přesunů. Kapacita 20 tisíc not, generování hrající rutiny i s daty na pásek.

COLOSS MON

Program pro studium, modifikaci, tvorbu a snadné ladění programů ve strojovém kódu. Umožňuje též přenos programů z počítače ZX Spectrum.

KANTOR I

Prvních pět lekcí studia deskriptivní geometrie pro střední školy a přípravu ke studiu s plně interaktivním způsobem komunikace.

KANTOR Ia

Tři náročně sestavené zkušební programy k I. dílu souboru KANTOR s vyhodnocováním a měřením potřebného času.

KANTOR II

Lekce šestá až osmá studia deskriptivní geometrie pro střední školy včetně cvičení na přípravu ke studiu a k přijímacím zkouškám.

KANTOR III

Lekce devátá až jedenáctá studia deskriptivní geometrie s cvičením o vzájemné poloze bodů, přímek a rovin s normalizovanými konstrukcemi.

GEORUT I

Knihovna základních konstrukčních rutin pro kreslení a řešení geometrických úloh, část stereometrie.

UPOZORNĚNÍ: Každý program se smí provozovat v jednom čase pouze na jednom počítači. Kopírování programů, dat, nebo příruček dalším osobám není dovoleno.

Nabízené programy lze osobně zakoupit v prodejně 602. ZO Svazarmu (Praha 1, Martinská 5, telefon 22 87 74), nebo si je můžete na korespondenčním lístku objednat na adrese 602. ZO Svazarmu, Dr. Z. Wintra 8, 160 41 Praha 6. Na objednávce nezapomeňte uvést typ počítače a jeho verzi!

Do úplného výčtu programů naší nabídky už schází jen několik titulů, určených speciálně pro IBM PC. Protože je to kvalitativně naprosto nová kategorie programů, zaslouží si samostatný a rozsáhlejší popis. Ten naleznete v příštím čísle v naší nabídce, která bude věnována výhradně jim.

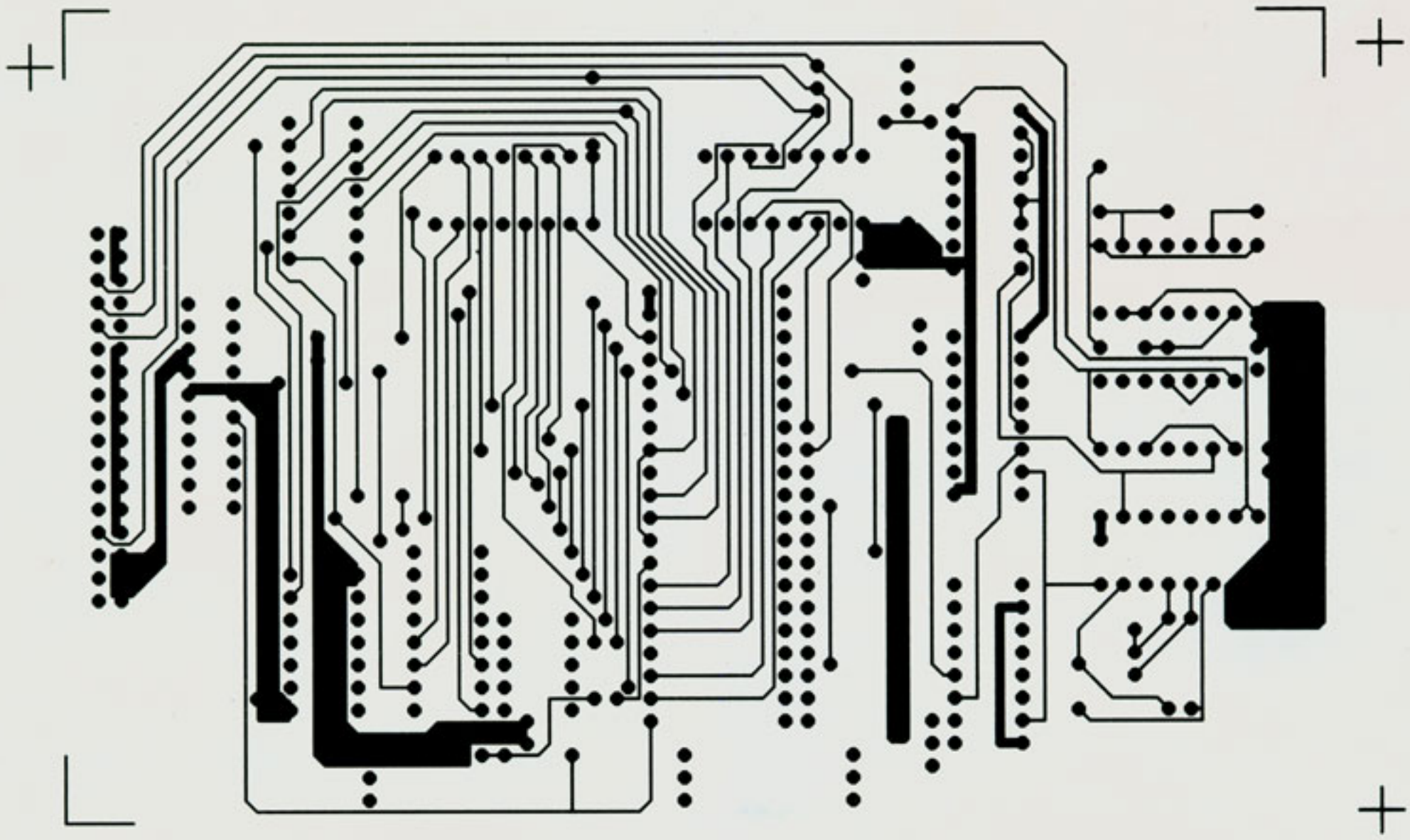
>>> >>> >>> >>> >>> *(pokračování v příštím čísle)*

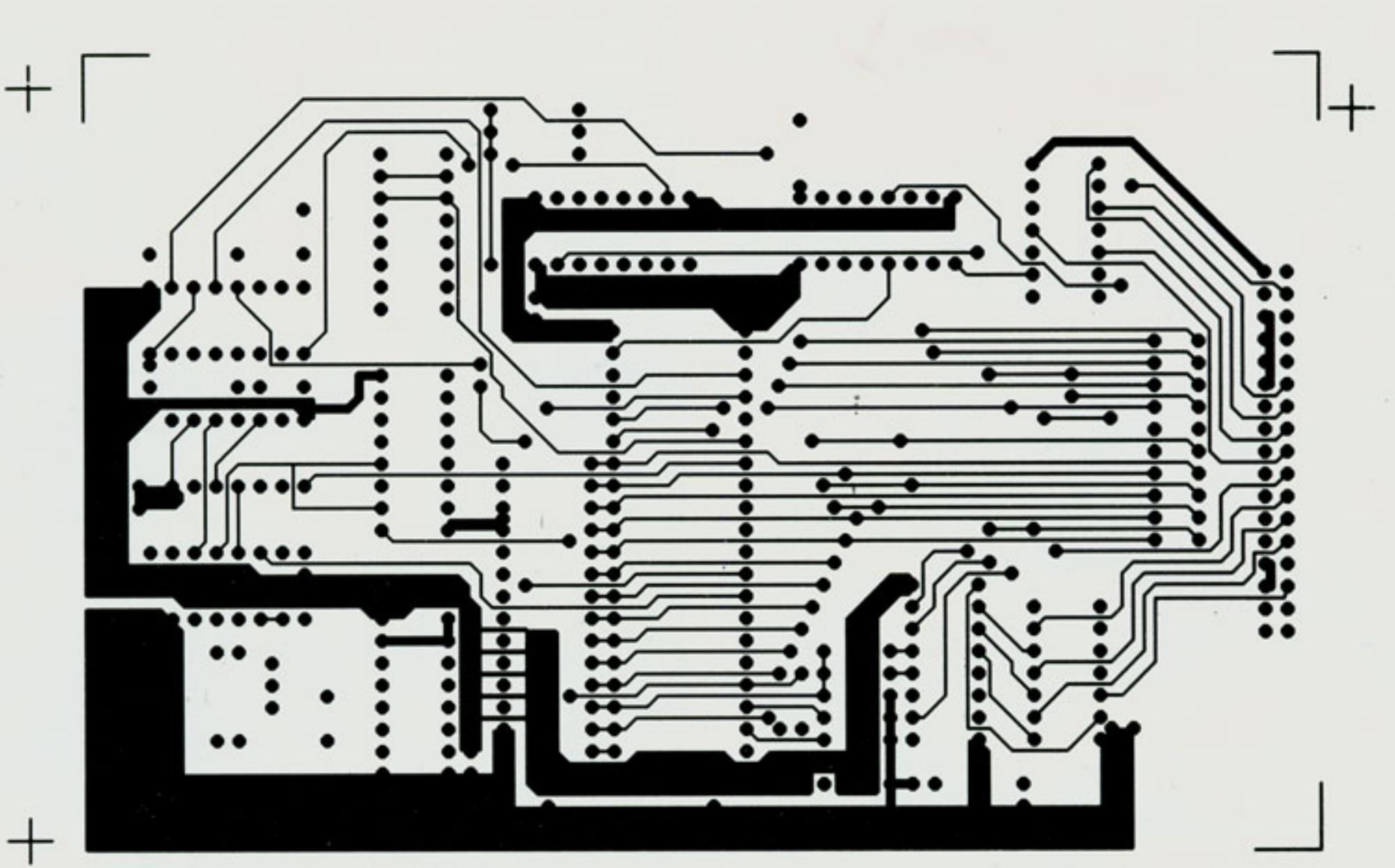
****

POSTAVTE SI S NÁMI DISKOVÝ ŘADIČ

****

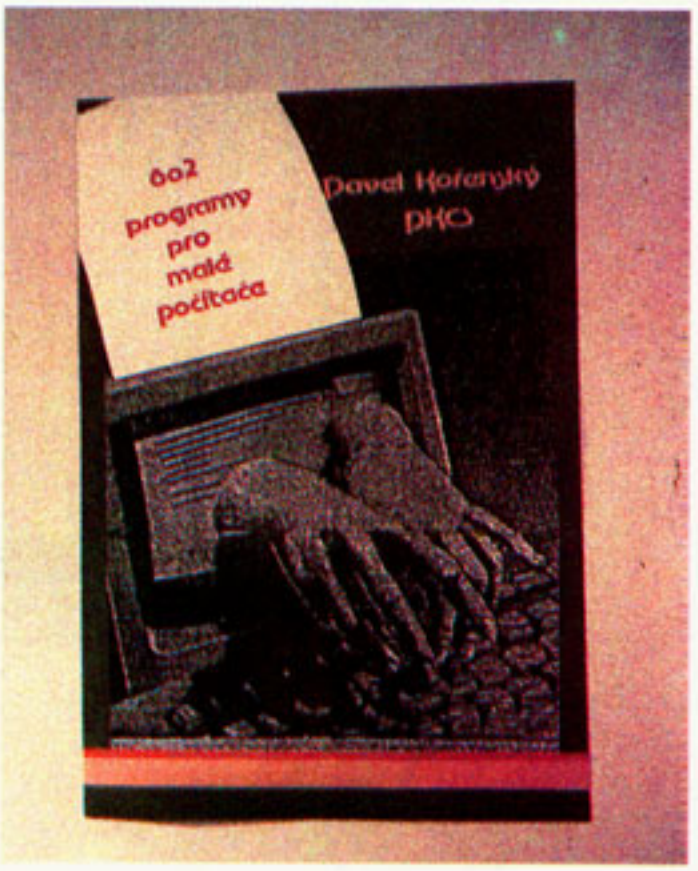
Obrazec plošných spojů desky řadiče.
Nahoře strana součástek.
Dole strana spojů.



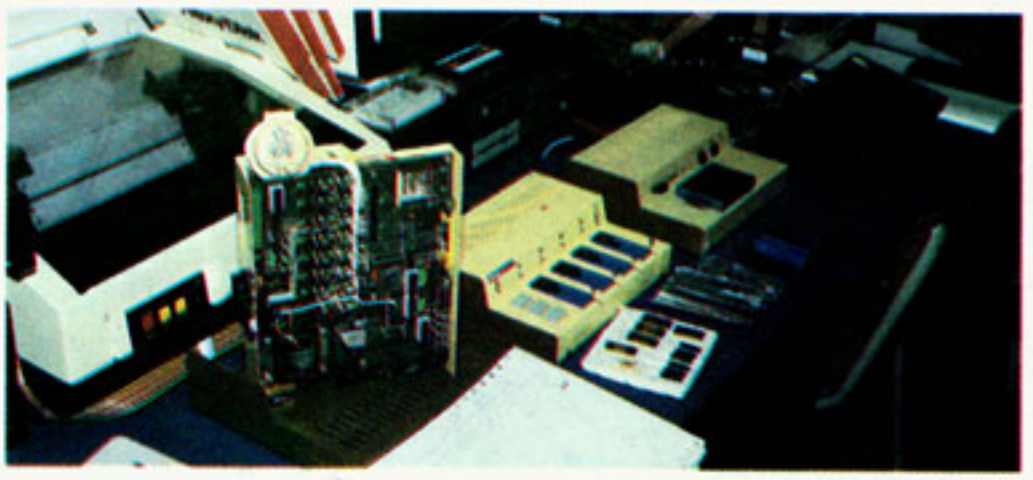



LEC
VIDÍME DO TOHO!


LogoStar LS
Electronic Products
LogoStar LS
Electronic Products
NetWare


602
programy
pro
malá
počítača
Pavel Hofanský
PHO

SOFTWARE 89




GENIE