Miskolci Egyetem
Irányítástechnikai tanszék

Szakdolgozat

DP

i8080 mikroprocesszor assembly oktatási program (1. rész)

Perlaki Attila

Tartalom

1. Bevezetés

A dolgozatban ismertetésre kerülô program az Irányítástechnika tanszék "Vezérlés- és mikroprocesszortechnika" tárgyához készült, mint oktatási segédlet. A jelenlegi V1.2 változat elsôsorban bemutató jellegű alkalmazásra, illetve rövidebb programok igen részletes tesztelésére készült fel. A rendszer tartalmazza a számonkéréshez szükséges programelemeket és az utasításokhoz tartozó rövid ismertetôket (hypertext eljárással), illetve a további igények szerint bôvíthetô, ennek lehetôségeit ismertetem.

A dolgozathoz tartozó programrendszert Vincze Gáborral közösen készítettük, a kiírásnak megfelelôen. A programozási feladatokat egyenlô mértékben osztottuk meg egymás közt, ezt a továbbiakban minden társszerzôre vonatkozó utaláskor (V.G.) jelzéssel külön is feltüntetem.

A fejlesztés során nyújtott értékes tanácsaikért s a szükséges "erôforrások" biztosításáért ezúton is szeretnénk köszönetet mondani a tanszék munkatársainak.

2. Történeti áttekintés

2.1 Vezérlési elvek

Évszázadok óta ismertek azok az elvek, melyek segítségével bonyolult óraszerkezetek építhetôk, köztük olyanok is, melyek a bolygók járását éppúgy jelezhetik, mint a naptárban vándorló vallási ünnepeket. Vezérelhetnek összetett harangjátékot, vagy apróbb-nagyobb mozgó figurákat is. De ugyanígy vezérelhetôk más eszközök is,sôt e korai idôkben is létezett a programvezérlés a szövôgépeken.

A ma ismert fô vezérlési formák közül a mechanikus a legidôsebb, és csaknem teljesen kiszorult a használatból. A pneumatikus és hidraulikus technikát is csak viszonylag szűk területen alkalmazzák, de ott általában mással nem is helyettesíthetô (pl. robbanásveszélyes térben). A legelterjedtebb és legrugalmasabb az elektonikus vezérléstechnika különbözô fajtái.

Az elektronikus vezérlések lehetnek:

- Reléhálózatok (elektromechanikus)
- Kapuhálózatok
- diszkrét elemek (tranzisztor)
- SSI áramkörök (74xx sorozat)
- PLA áramkörök
- Programvezérlésű eszközök
- PLC-k
- CPU kártyák
- egychipes számítógépek
- mikrovezérlôk

A gyakorlatban a programozható eszközök használata vált általánossá.

2.2 Számítógép generációk

A "számítógépek korát" az ENIAC-tól szokás számítani. Az ezt megelôzô ún. nulladik generációt a mechanikus elvű gépek alkotják. (pl. Charles Babbage gépe) Az elsô generációs gépek elektroncsövek ezreit tartalmazták s elég megbízhatatlanok

voltak. Ekkor alakult ki a napjainkig legjelentôsebb elv, a Neumann-architektura.

Az elektroncsöveket felváltotta a tranzisztor a második generációban, majd az IC a harmadikban. Az áramköri egységek általában nem kifejezetten számítástechnikai célokra készültek.

A mai számítógépek nagy többsége a negyedik generációba tartozik. Általános jellemzôjük a nagy integráltság, a berendezésorientált áramkörök, és a mikroprocesszor,mint az egész rendszer működéséért felelôs IC.

A fejlesztések során már kialakultak az ötödik generáció körvonalai. Fontos lépés, hogy szakítanak a Neumann-architektúrával, mivel ennek teljesítménykorlátai már jelentôs problémákat okoznak. Fôleg az igen számításigényes alkalmazásoknál, ahol a valósidejű működés is követelmény, csak a multiprocesszoros rendszereknek vannak esélyeik. (pl. képfeldolgozás)

2.3 Az Intel család

Hivatalosan az elsô mikroprocesszort 1971-ben jelentették be. A számítástechnika tömeges elterjedéséhez vezetô út lényeges állomása volt ez az "elektronikus százlábú", hogy a szükséges méret, a megbízhatóság és a költségtényezôk piacképesek legyenek.

A négybites 4004-tôl, a 4040-esen és a már nyolcbites 8008-ason át, jutott el a fejlesztés a 8080-as processzorig, amely "dinasztia-alapítóvá" vált.

A "nagy generáció" táblázata:

INTEL	ZILOG	NEC
4 bites mikroprocesszorok		
i4004		
8 bites mikroprocesszorok		
i8008		
i8080		µPD8080
i8048	Z80	
i8051	Z8	
i8085		µPD8085
	NSC800	µPD780

16 bites mikroprocesszorok		
i8086/88		V20, µPD8086
	Z8000	V25, V30
i80186	Z800	
i80286		V40, V50
32 bites mikroprocesszorok		
i80386	Z+80000	V60, V70
		V8x
i80486		

[CWI Mikrovilág 1990/6 alapján]

A 8080-as család "szoros rokoni kötelékébe" csak a 8085-ös és a Z80-as sorolható. A 8080-as processzorra megírt programok változtatás nélkül futtathatók, példa erre a CP/M operációs rendszer. A 8085 csak két, a Z80 ellenben több száz új kódot tartalmaz, így visszafelé csak a többletutasítások elhagyásával maradhatunk kompatibilisek, ami Z80 esetén nagyon szigorú megkötés.Hardware tekintetében mindkét processzor jelentôsen eltér az alaptípustól.

A 8080-as család "távolba szakadt rokonai" elsôsorban a 80x86 család tagjai. Itt elsôsorban a már az alaptípusnál megismert módszerek továbbfejlesztése észlelhetô. Az assembly nyelv sok hasonlóságot mutat,a regiszterkészlet,a flagkezelés és a címzési módok többsége is ismerôs,így az átállás jelentôs nehézségek nélkül megoldható.

2.4 Fejlesztések

Mind az Intelnél, mind konkurrenseinél folynak a kutatások és fejlesztések az egyre nagyobb teljesítményű processzorok kibocsátása céljából. Az Intel új felépítésű sorozatot indított a 80860-assal, mely egy lebegôpontos és egy grafikai koprocesszort, valamint külön buszkezelôt és gyorsítótárat is tartalmaz. A mag egy igen gyors RISC processzor, mely minden utasítást egy óraciklus alatt végre tud hajtani, miközben a koprocesszorok is zavartalanul dolgozhatnak.

3. Az i8085 CPU

3.1 Hardware jellemzôk

Az elôbb felsoroltak alapján célszerűnek látszik a 8080-as assembly-vel megismerkedni. Ugyan akkor ez a típus még nem tartalmazza a rendszervezérléshez szükséges összes elemet, alkalmazásakor kiegészítô áramkörökre (i8224 és i8228) van szükség. Ezért a gyakorlatban inkább i8085-öt és Z80-at használnak.

Az i8085-ös alkalmazásását elsôsorban a processzor hardware felépítése indokolja, mivel a fejlettebb technológiának köszönhetôen sokkal áttekinthetôbb (s így a felhasználó szempontjából egyszerűbb), mint az alaptípusé. Az assemblyben nincs eltérés (a plusz két utasítás kivételével, ellentétben a Z80 több száz "rendszeridegen" utasításával), viszont a megszakításkezelés eltér, amit a programozás során figyelembe kell venni.

A 8085-ös az Intel elsô NMOS mikroprocesszora volt. Ez a technológiai váltás jelentôs elônyöket biztosított az alaptípushoz képest:

- egyetlen tápfeszültség (három helyett)
- beépített órajelgenerátor (külsô kétfázisú vezérlô helyett)
- beépített megszakításkezelô egység
- hatékonyabb buszvezérlés (segédáramkörök helyett)
- beépített soros I/O

A processzor 40 kivezetéses DIP tokozással került megvalósításra. Ennek következménye, hogy nincs elég láb az összes adat-, cím-, és vezérlôjel különálló kialakítására, ezért multiplexelt cím/adatbuszt alkalmaznak. (A tizenhatbites címsín alsó fele egyúttal a nyolcbites adatsínt is tartalmazza.) Ez a tesztelést és egyes áramkörök illesztését megnehezíti, de ehhez a processzorhoz több olyan speciális áramkör (általában a nagy sorozatszámú processzoroknál megszokott kiegészítôk, mint pl. PIO, SIO) készült, ami a rendszerépítést megkönnyíti.

A processzor kivezetései: [1.ábra]

A kivezetések értelmezése:

Tápellátás:

- **5V** tápfeszültség (±5%)
- **GND** közös testpont

Cím/adatbusz:

- **AD0...AD7** multiplexelt cím/adatbusz
- **A8...A15** címbusz felsô fele

Órajelek:

- **X1, X2** belsô órajelgenerátor csatlakozói
- **CLK** az elôállított órajel kimenete

Rendszervezérlés:

- **WR** írás jelzés
- **RD** olvasás jelzés
- **IO/M** periféria/memória kiválasztás
- **ALE** a multiplex sínen cím engedélyezés
- **READY** készenli jel külsô hozzáféréskor
- **HOLD** buszátadás kérés
- **HLDA** buszátadás igazolása

Megszakításkezelés:

- **RESETIN** alaphelyzetbe állító bemenet
- **RESETOUT** alaphelyzetbe állás jelzés
- **INTR** 8080 kompatibilis megszakításkérés
- **INTA** megszakítás elfogadás igazolása
- **RST x.5** fix című megszakítások kérése
- **TRAP** nem maszkolható megszakításkérés
- **S0, S1** processzor állapotjel

Soros I/O:

- **SID** soros adatbemenet
- **SOD** soros adatkimenet

A belsô felépítés vázlata: [2. ábra]

A mikroprocesszor működése a beolvasott utasítástól és az azt megelôzôen keletkezett állapotoktól együttesen függ. A folyamat a vezérlôegységen keresztül játszódik le,ütemezett módon. A vezérlôegység programja a processzorba van "behuzalozva", ezen csak a gyártó változtathat. Az ütemezés alapjele az órajel, amelyet az i8085 a külsô idôzítôtagok segítségével önmaga állít elô. A processzor normális működését bizonyos külsô jelek is döntôen befolyásolják (RESET, INT, HOLD, stb.), ezek folyamatos figyelése is a vezérlô feladata.

A vezérlô tíz különbözô ún. gépi ciklust ismer:

- **FETCH** utasítás beolvasás
- **MEM. READ** memóriaolvasás
- **MEM. WRITE** memóriaírás
- **STACK READ** memóriaolvasás veremmódszerrel
- **STACK WRITE** memóriaírás veremmódszerrel
- **I/O READ** perifériaolvasás
- **I/O WRITE** perifériaírás
- **INTERRUPT** megszakítás
- **HALT** leállás
- **HOLD** buszátadás, "lebegés"

Ezen ciklusok kombinációiból a processzor minden utasítása és egyéb működése összeállítható.

A beolvasott utasítás az utasításregiszterbe kerül,majd az utasítás dekóder értelmezi és végrehajtja. Ennek során a processzor összes elemére hatással lehet. A leggyakoribb művelet az adatmozgató és az aritmetikai/logikai utasítások végrehajtása. Az elôbbi során a regisztertömb állapota, az utóbbi során a jelzôbitek is változnak. A regisztertömb két eleme, az 'A' és 'F' regiszter kitüntetett fontosságú: közvetlenül az aritmetikai és logikai egységhez (ALU) vannak rendelve. Az utasítások a növekvô címek felé sorosan hajtódnak végre, mely során a programszámláló (PC) tartalma az utasítás hosszának megfelelôen növekszik. Ez egy esetben másként is lejátszódhat: amikor a PC tartalmát változtatjuk meg valamely utasítással (azaz "ugrunk" a programban). A vezérlésátadás tehát speciális adatmozgatás.

A külvilággal az adat- és címbuffereken keresztül teremt kapcsolatot a processzor. Az ehhez tartozó vezérlôjelek kibocsátása és fogadása is a belsô vezérlôegység feladata.

3.2 Software jellemzôk

A processzor egy akkumulátort és hat általános célú nyolcbites regisztert, két tizenhatbites speciális címmutatót és öt flag-et tartalmaz. Az általános regiszterek (B, C, D, E, H, L) páronként tizenhat bites regiszterként is kezelhetôk. A HL regiszterpár,címmutatóként használva, az M jelzésü látszólagos regiszter értékére mutat. A két speciális mutató a PC (Program Counter), amely a végrehajtandó kódra mutat,és az SP (Stack Pointer), amely a veremtár aljára mutat. (A verem a csökkenô címek felé építkezik.)

A regiszterblokk felépítése:

'A' regiszter	'F'jelzôbitek
'B' regiszter	'C' regiszter
'D' regiszter	'E' regiszter
'H' regiszter	'L' regiszter

'PC' regiszter
'SP' veremtár

A processzor ezen kívül tartalmaz ún. nem címezhetô regisztereket is (pl. az utasításregiszer), melyek a programozó számára elérhetetlenek.

Az 'F' regiszter felépítése:

- 7. bit S elôjelbit
- 6. bit Z zérusbit
- 5. bit nem publikált
- 4. bit AC félbyte-os átvitel
- 3. bit nem publikált
- 2. bit P paritás
- 1. bit nem publikált
- 0. bit CY átvitel

Az ún. nem publikált jelzôbitek közül az 5. és az 1. felhasználásra került, mint X5 - bôvítô elôjelbit - és V - bôvítô túlcsordulásbit -, de ezek felhasználása inkompatibilitással jár.

Az i8085 az ún.egycímes processzorok közé tartozik, azaz egy műveletben (az utasítás beolvasás műveletét nem számítva) csak egy memóriaelérés lehetséges. Ezért szükséges a regiszterkészletet viszonylag nagyra méretezni, ellentétben pl. a 65xx sorozattal, melyben csupán egy(!) általános és két indexregiszter van. Az i8085 továbbá nem képes a memóriával közvetlen aritmetikai/logikai műveletet végezni, csak közvetve (pl. az 'M' látszólagos regiszteren keresztül). Mindezek következménye, hogy az utasításkészlet csaknem fele adatmozgató, és ez a teljesítménymutatókat jelentôsen rontja.

4. Az i8085 assembly nyelve

4.1 Címzésmódok

Az assembly nyelvben valamilyen gépi objektumon (regiszter, memóriarekesz, periféria) valamilyen műveletet kívánunk elvégezni. Az i8085 assemblyben szigorúan csak utasítások és paraméterek használhatók. Minden utasítástípushoz tartozik egy vagy esetleg több paramétertípus.

Az utasítások paraméterezése (ahol szükséges) a címzésmódok szerint történik, amelyet az utasítás kódja határoz meg. Ebbôl következik,hogy két szempont szerint is csoportosítható a címzésmód.

Az adatelérés módja szerint megkülönböztetünk:

- címzés nélküli,
- konstans paraméteres,
- direkt címzésű,
- indirekt címzésű utasításokat.

Az utasítások felépítése szerint beszélhetünk:

- paraméter nélküli,
- beleértett paraméteres,
- regiszteres,
- akkumulátorhoz rendelt regiszteres,
- kétregiszteres,
- regiszterpáros,
- kétregiszteres regiszterpáros utasításokról.

A 'DP' program ez utóbbi rendszert követi, programozástechnikai megfontolások miatt.

4.2 Utasításrendszer

Az utasításrendszer a processzor által elvégezhetô műveleteket tartalmazza, ami ebbôl hiányzik, azt programmal általában pótolni lehet. Az i8085-ös processzor (az i8080-ashoz hasonlóan) nem képes néhány igen fontos műveletet utasításszinten elvégezni: hiányzik az indirekt I/O címzés, a relatív címzés, a szorzás/osztás műveletpár és a közvetlen bitmanipuláció. A Z80-as processzor (a szorzás és osztás kivétlével) mindezen lehetôségekkel

rendelkezik,de ezt csak a kapcsolókódos rendszerrel volt lehetséges megvalósítani (a nyolcbites felépítés miatt), ez pedig bonyolultabbá tette az assembly-t is. Az i8085 assembly az itt felsorolt hiányosságok ellenére egy csaknem teljes és könnyen elsajátítható nyelv.

Az utasításokat, működési módjuk szerint,a következôképpen lehet csoportosítani:

I. Adatmozgató utasítások családja

MOV típus: adatmozgató

- MOV reg, reg
- MOV M, reg
- MOV reg, M

MVI típus: értékadó

- MVI reg, data
- MVI M, data

LXI típus: értékadó

XCHG: csere

XTHL: spec.csere (verem)

SPHL: spec.adatmozgató (SP)

PCHL: spec.adatmozgató (PC)

Jxx típus és JMP: spec.értékadó (PC)

Cxx típus és CALL: spec.értékadó (PC, verem)

Rxx típus és RET: spec.értékadó (PC, verem)

RST típus: spec.értékadó (PC, verem)

LDxx típus: spec.adatmozgató (A, HL)

- LDA addr
- LDAX rp
- LHLD

STxx típus: spec.adtamozgató (A, HL)

- STA addr
- STAX rp
- SHLD

PUSH típus: spec.adatmozgató (verem)

POP típus: spec.adatmozgató (verem)

IN port: spec.adatmozgató (I/O)

OUT port: spec.adatmozgató (I/O)

II. Aritmetikai és logikai utasítások családja

AxxA típus: aritmetikai

- ADD reg és ADD M
- ADC reg és ADC M
- SUB reg és SUB M
- SBB reg és SBB M

AxxI típus: aritmetikai

- ADI data
- ACI data
- SUI data
- SBI data

LxxA típus: logikai

- ANA reg és ANA M
- XRA reg és XRA M
- ORA reg és ORA M
- CMP reg és CMP M

LxxI típus: logikai

- ANI data
- XRI data
- ORI data
- CPI data

ROT típus: léptetô

- RAL
- RAR
- RLC
- RRC

INR típus: számláló

DCR típus: számláló

INX típus: számláló

DCX típus: számláló

DAD típus: aritmetikai

III. Speciális utasítások családja

DAA: helyesbítô

CMA: spec.logikai

CMC és STC: flagkezelô

DI és EI: megszakításkezelô

SIM és RIM: soros I/O kezelô

NOP: hatástalan

HLT: processzor stop

5. A 'DP' program használata

5.1 Oktatási elvárások

A "Vezérlés- és mikroprocesszortechnika" tárgy oktatása során a gépi kód és az assembly nyelv elsajátításához olyan segédeszköz készült,mely egyesíti magában a szemléltetôeszköz (pl. fóliák) és a mérôpanel elônyeit. A programmal szemben támasztott követelmények a következôk voltak:

a. Legyen lehetôség az egymást követô állapotok szemléletes megjelenítésére.

b. Minden utasítás legyen megvalósítva.

c. Minden regiszter legyen megvalósítva.

d. Legyen lehetôség a memória megjelenítésére.

e. Minden szokásos számrendszer legyen alkalmazható. (bináris, decimális, hexadecimális)

f. Legyen lehetôség bonyolultabb (több utasításból álló) feladat szemléltetésére.

g. Legyen lehetôség a hallgató számonkérésére.

h. Legyen a program több irányban is könnyen továbbfejleszthetô.

A program az egymást követô állapotokat két képernyôablakban jeleníti meg:a végrehajtás elôtti és a végrehajtás utáni helyzet szerint.

A regiszterek s az értékük szerinti címek tartalma ezen ablakokon belül lett feltüntetve, mégpedig úgy, hogy a bal oldalon a regiszterek párba szervezve hexadecimálisan és binárisan (a logikai műveletek miatt) jelennek meg, a jobb oldalon a megcímzett memória és a rá következô hét cím tartalma látható (szintén hexadecimális formában), valamint a flagek bitenként kiemelve. A regiszterekben és a memóriában a program 'EDIT' funkciójával szabadon javíthatunk.

A program az összes i8085 utasítást tartalmazza a nem publikáltak kivételével.

Az utasítások bevitelekor a paraméterek a három használt számrendszer közül bármelyikben tetszôlegesen megadhatók.

A több utasításból álló feladatok (programok) szemléltetése a 'RUN' és 'STEP' funkciókkal lehetséges.A programokról assembly listát is lehet kérni, illetve ezek a programok lemezre menthetôk vagy onnan tölthetôk. A memória teljes 64K méretben megvalósításra került, az I/O 256 címet tartalmaz, ezek közül néhány, a szemléletesség kedvéért, speciális feladatokat lát el. (07: hang, 0A-0D: belsô óra, FF: megszakítási utasítás)

A hallgatók számonkérésére egy, a gyakorlatban sokszor használt, négytippes tesztrendszer került beépítésre. A kérdések a teljes utasításkészletet lefedik, a felajánlott válaszok pedig minden esetben tartalmazzák a helyes megoldást is. A teszt eredménye (százalékban kifejezve) az utolsó kérdés után leolvasható,majd a gép a hallgató nevével egy rekordot illeszt a 'DP.SCT' file-ba. Ennek tartalma: a hallgató neve és eredménye százalékban illetve táblázatszerűen, a jó és rossz válaszok feltüntetésével.

Az elkészült V1.2 verzió a felsorolt elvárásoknak, a fentiek szerint, megfelel, de az esetleges további igényeknek megfelelôen átalakítható, illetve fejleszthetô.

A program moduláris felépítésű, 'C' nyelven íródott. A felépítésbôl következôen más nyolcbites processzor is megvalósítható a rutinok nagy többségének csekély mértékű átalakításával vagy változatlanul hagyásával. Tizenhatbites processzor, nem teljes körű megvalósítása sem okoz lényeges nehézségeket. (Azonban védett módokkal, kizárási állapotokkal és társprocesszorokkal kapcsolatos feladatok valószínűleg komolyabb problémákat vetnek fel.) A továbbfejlesztés történhet egy teljes nyolcbites rendszer, tehát PIO, SIO, stb., beintegrálásával is.

Továbbfejleszthetô a program hardware szemléltetésére is, így logikai analizátornál megszokott módon nyomon követhetôkké válnának a processzor jelváltozásai.

5.2 Kezelés

A program a betöltés után rövid ideig a tömbök felépítésével foglalkozik, ezalatt a 'PLEASE WAIT' felirat látszik. Ha ez sikertelen a memória telítettsége miatt, akkor hibajelzéssel leáll. Ekkor a bent lévô rezidens programok eltávolításával kell helyett

biztosítani. Amennyiben az indulás sikeres volt, a kezdôkép jelenik meg. (C.1 kép)

A program két fô beviteli ponttal rendelkezik: a 'MNEMO.' jelű utasítássorral és a funkcióbillentyűk sorával.

A 'MNEMO.' sorba i8085 assembly utasításokat lehet írni, amelyek végrehajtódnak és eltárolódnak az aktuális programszámláló értékének megfelelôen. Amennyiben az utasítás valamilyen okból nem értelmezhetô, akkor a hibának megfelelô figyelmeztetés jelenik meg, majd a 'HELP' menüoldala kerül a képernyôre,ahol kikereshetjük a jó utasítást. (A bevitt adatokat egyébként nagy rugalmassággal kezeli a program, ha csak a mnemonik kerül bevitelre, úgy a gép illeszt hozzá egy megfelelô paramétert.)

A bevitel során a 'BACKSPACE' billentyűvel törölhetjük az utolsó karaktert, a 'DEL' billentyűvel pedig az egész sort. A bevitt sort az 'ENTER' zárja le, de amennyiben ezt külön bevitel nélkül ütjük le, akkor a PC aktuális címén lévô utasítás (ami az 'INS' billentyűvel bármikor visszakérhetô a 'MNEMO.' sorba) hajtódik végre.

A PC tartalma a nyilakkal léptethetô, a fel/le 256 byte-os, a jobbra/balra egy byte-os eltolást jelent.Az 'END' a PC alsó felét FFH értékkel teszi egyenlôvé, a 'HOME' kinullázza.

A program szolgáltatásai:

- F1: 'HELP' fômenü
- Ctrl F1 v. Alt h: aktuális 'HELP' ablak
- PgUp v. PgDn: lapozás a 'HELP'-ben
- F2 v. Alt r: programfuttatás
- Alt a: assembly programlista
- F3 v. Enter: lépésenkénti végrehajtás
- F4 v. Alt e: regiszterjavítás
- F5 v. Alt l: file töltés
- F6 v. Alt s: file mentés
- F7 v. Alt i: i8080-as megszakítás
- Shift,Alt,Ctrl F7: RST 5.5, 6.5, 7.5
- Ctrl F8: NMI
- F8: RESET
- F9 v. Alt t: teszt 36 v. 12 kérdéssel
- F10: kilépés a programból

A 'HELP' szolgáltatást igénybe véve egy 'MENU' ablak jelenik meg, amelybôl kiválasztható (nyilak és az 'ENTER' billentyűk

segítségével) az utasításcsoport, majd ezen belül, a 'MNEMONICS' ablakból, az utasítás, melynek leírása ezek után megjelenik egy 'HELP' ablakban. A végrehajtott utasításokról azonnali help is kérhetô 'Ctrl F1' beütésével. (C.2, C.3, C.4 képek)

A memóriában lévô programok lépésenként (F3) és folyamatosan (F2) is futtathatók. Ez utóbbi esetben a program futása megszakad 'NOP' és 'HLT' végrehajtásakor, vagy megszakítható a billentyűzet tetszôleges leütésével. (C.5 kép)

Regiszterjavításkor egy kis ablak jelenik meg az alsó képfélen, melybe az új értéket lehet beírni (hexadecimálisan), vagy más regiszterre lehet átmozgatni a nyilak segítségével. A regisztermezô és a memória közt a 'TAB' billentyűvel kell átkapcsolni. A beírt értéket 'ENTER' rögzíti, a kilépés pedig 'ESC' leütésével lehetséges. (C.6 kép)

A memóriában lévô programot disassembly listaként az 'Alt a' képes megjeleníteni az adott PC értékétôl, tíz utasítás terjedelemben (C.7 kép). A PC tartalma a megfelelô értékkel növekszik.

A memória tartalma (program, adat) az 'F5' és 'F6' segítségével tölthetô be, illetve menthetô ki. Lemezre mentéskor megadandó a terület kezdô és végcíme, valamint a filenév (az esetleges meghajtó és útvonalazonosítókkal együtt), de kiterjesztés nélkül, mivel az minden esetben: '.85'. Betöltéskor a végcím a file hosszából automatikusan kialakul, tehát erre nem kérdez rá a program. (C.9 kép)

A megszakítások, amennyiben engedélyezettek (kivéve a nem maszkolható megszakítást), végrehajtódnak, tiltás esetén pedig figyelmeztetô üzenet jelenik meg. A 'RESET' (F8) alaphelyzetbe állítja a programot, de a memóriát nem törli.

A számonkérés során az elôlap megjelenése után tizenkét (Alt t) vagy harminchat (F9) kérdésre kell választ adni, a megfelelô megoldás kiválasztásával. A teszt végén a program értékel és fileba menti az eredményt. (C.10, C.11, C.12 képek)

Kilépni az 'F10' billentyűvel lehet.

6. A 'DP' program leírása

6.1 Szerkezet

A program, a feladat összetettsége és a közös fejlesztés miatt, logikailag jól elkülöníthetô alcsoportokba lett szervezve, melyek mindegyike (egy kivételtôl eltekintve) külön file-ba került. A részek a fôbb globális változók deklarációival, az általánosan használt makródefiníciókkal, a könyvtári rutinok behívásával és a 'main' függvénnyel együtt egy "gyűjtôbe" (DP.C) épülnek be:

I. Utasítás-leírók:

- DPD.C mnemoniktömb
- DPH.C help-ablakok (V.G.)

II. i8085 szimuláció:

- DPM.C adatmozgató utasítások (V.G.)
- DPA.C aritmetikai/logikai utasítások
- DPS.C speciális utasítások (V.G.)

III. Felhasználói felület:

- DPI.C inicializáló rutinok
- DPP.C parancsértelmezô rutinok
- DPK.C képernyôkezelô rutinok (& V.G.)
- DPW.C ablakkezelô rutinok (V.G.)
- DPT.C számonkérés rutinjai

IV. Segédrutinok:

- DPF.C file kezelô rutinok
- DPZ.C átszámító rutinok (V.G.)

A 'program' függvény (DPP.C rész) a magja az egész rendszernek.Az inicializálás után egy állandó ciklusba kerül a gép,melybôl csak a kilépés parancs (F10) vezet ki. Az elágazásoknak két csoportja van: a kisebb a 'MNEMO.' kezelôi, a nagyobb a szolgáltatások hívása. Csaknem minden függvényhívás továbbiakat is hív, pl.megjelenítô rutinokat.

A rendszer működéséhez szükség van néhány igen fontos globális változóra. A memória (64kbyte-os mérete miatt) nem hagyományos módon lett deklarálva, hanem ún. távoli mutatóként, amelyre a 'main' függvényben egy allokáció épül rá, amennyiben van elég szabad memória. A portok egyszerű tömbként, a 'com' (utasításkód tároló), a 'par' (utasításhoz tartozó paraméter),az 'ist' (megszakítás állapot) és a 'hlp' (aktuális 'HELP' ablak száma) pedig egészértékként kerültek meghatározásra. A szövegműveletek számára egy 80 byte-os pufferterület lefoglalásra került.A regiszterek egy tizenkét elemű tömbben helyezkednek el, melyre összetett makródefiníciók épülnek. Létezik továbbá egy függvénymutató tömb,mely a szimulációs rutinok hívását segíti. Globális változónak számítanak még a DPD.C és DPH.C file-ok teljes egészében,valamint még néhány egyéb helyen lévô segédváltozó.

A makródefiníciók két csoportja található a gyűjtôben: a regiszterre valamint a jelzôbitekre vonatkozók. A regiszterek egy és kétbyte-os alakja (regisztернév és általános pl. RX alak) szerint kerül deklarálásra, a jelzôbitek neveik és feltételkódjaik szerint. A makrók használata jelentôsen megkönnyítette mind a programozást, mind a forráslista megértését. További jelentôs makródefiníciók találhatók még a DPK.C és DPI.C részekben.

A 'main' rutin csupán indítófeladatot lát el.

6.2 Rutinok

A program szempontjából fontos rutinok részletes leírása elsôsorban a további fejlesztések és módosítási igények miatt szükséges. A programszerkezetbôl adódóan az egyes rutinok csaknem teljesen elszigeteltek a működésüktôl "idegen" rutinoktól. A mag sugaras hívási szerkezete (amelyhez hasonló az utasítás végrehajtási rutin is) viszonylag nem túl mély hívási vermet hoz létre. A programban rekurzív vagy kereszthivatkozó részek nincsenek.

A függelék tartalmazza az összes létrehozott függvény teljes listáját, amelyhez a melléklet MAP listája is kapcsolódik.

DPA.C rész

Két függvénycsoport található itt: a flagkezelôké és az utasításrutinoké. A flagkezelô az 'A' regiszterre és az egyéb regiszterekre külön rutint tartalmaz ('flagsA' és 'flagsR') eltérô viselkedésük miatt. Külön függvény a 'parity', mivel túl összetett a művelet, amit végez. Az utasítások a 4.2 fejezet felosztását követik.

A flagkezelôkhöz tartozik néhány makrodefiníció is: a félbyte-os maszkolók ('HA','Hr','Hx'). Ide számít egy globális változó,az 'x' melyek feladata az összehasonlítás során a végrehajtást megelôzô állapot tárolása,hogy a megfelelô jelzôbitek helyesen álljanak be. A rutinok az 'F' makró által jelzett 7. regisztertömb-elemen (ami a flagregiszternek felel meg) végzik el a változtatásokat.

Az utasításrutinok szintén használnak makrókat a gyűjtôben meghatározottakon kívül is, elsôsorban az összetett hívások szétbontásakor. Ilyen eset az 'axxa' (továbbá a 'axxi', 'lxxa', 'lxxi' és 'rot' is) mely 'case' struktúrával pontosítja az utasítás jelentését. Az 'EXC' makró a kódrendszer egy furcsa következetlensége miatt szükséges: a kódban hatos számmal nem az 'A' regiszter (mely sorrendben itt következnék), hanem az 'M' látszólagos regiszter hívódik, s így az 'A' az 'F' helyén, hetes számmal azonosítható.

Az utasításrutinok általános felépítése: változódeklaráció, kódbontás (a szereplô regiszterek azonosítása), esetleges állapotmentés (x), művelet, szükség szerint flagkezelô hívása, végül visszatérés a megfelelô help-ablak számával.

DPI.C rész

Az inicializáló egyetlen feladata, hogy feltöltse a függvénymutató-tömböt. A 4.2 fejezet csoportosításának megfelelô makródefiníciók azt a maszkot tartalmazzák, melyek a kódtábla elemeibôl a rájuk vonatkozó kódokat kiválogatják. (Pl. a MOV a 40H-tól a 7FH-ig minden kódra igaz.) Az 'iccp' egy ciklusban feltölti a tömböt, amely ezután az elemére történô hivatkozáskor a megfelelô rutint hívja a DPM.C, DPA.C, vagy DPS.C részekbôl.

A 'mask' segédrutin a makrókhoz rendelt maszkokból és a ciklusértékbôl logikai értéket képez, mely szerint eldönthetô,hogy az adott maszk fedi-e a ciklusértéket.

DPP.C rész

A rendszer magja a 'program' függvény, melynek egyetlen feladata a billentyűzet figyelése, és az ennek megfelelô rutin hívása. A billentyűzetet a 'bioskey' könyvtári rutin kezeli, és egy speciális kóddal tér vissza.Ezekre a kódokra épül fel az itt található, 'K' kezdetű makródefiníciók. A rutinok kiválasztása 'switch' szerkezettel történik.

A rutin meghívása során lefut egy rövid bevezetô szakasz, mely letiltja a cursort, meghívja az inicializáló rutint ('iccp'), felépíti az alapképet és elhelyezi rajta az "intro" ablakot (C.1 kép), végül alaphelyzetbe állítja a munkaváltozókat. Ezután elindul a figyelôciklus.

A rutin egyrészt hívhatja a 'MNEMO.' beviteli sort kezelô függvényeket (karakter beillesztése, törlése, sorzárás, stb.),másrészt hívhat programszolgáltatásokat (F1 ... F10).

A további rutinok három csoportba sorolhatók: a bevitel, a kódvégrehajtás, és a szolgáltatások rutinjai. Ezek legnagyobb része ún. fejrutin, azaz csak további hívásokat tartalmaznak.

A 'ccp' rutin a kódvégrehajtás feladatait látja el, azaz meghívja a PC által mutatott kódnak megfelelô rutint a 'ccf' tömbön át. Ez a vezérlésátadási mód igen hatékonynak bizonyult. (A 'ccf' tömb feltöltése a DPI.C részben található.)

DPK.C rész

Ebben a részben az alapképernyô, a hozzá tartozó kiemelt pozíciók makrói és a kezelôfüggvények találhatók. Ugyanitt helyezkednek el az ablakkezelôk fejrutinjai is.

DPT.C rész

Itt a programrendszer egyik fontos szolgáltatásának működtetôrutinjai találhatók: ez a számonkérés helye.

A makródefiníciók hasonlóak a DPI.C részben alkalmazottakhoz, és abból jónéhányat fel is használ e rész,a többi pedig a 'cques' kérdéstömb felépítéséhez igazodik.

A fôrutin ('tprog') működése vázlatosan a következô: a munkaváltozók deklarálása és alaphelyzetbe állítása,a kezdôkép ("intro") megjelenítése, majd a kérdezôciklus elindítása, melynek végén az elért eredmény kijelzôdik és file-ba mentôdik.

A kérdezôciklusban a gép keres egy még fel nem tett kérdést, majd megjelöli a táblázatában. A kiválasztott kérdésre megkeresi a jó megoldást, majd erre háromféle módszerrel egy véletlenszámot kever rá. A kérdést és a négy válaszlehetôséget ezután megjeleníti a képernyôn (C.10 kép), majd várja a tippet. A kapott választ kiértékeli (ugyanis elôfordulhat, hogy a kérdésnek több különbözô jó megoldása is van), majd feljegyzi a táblázatába ill. gyűjti a pontokat.Az utolsó kérdés után a pontokat százalékba számítja át és a táblával együtt elmenti.

DPF.C rész

Itt két file-kezelô található. A 'mio' a memória be- és kivitelét végzi, a 'qio' a számonkérés eredményét naplózza. Ez utóbbi file-felépítése a következô: hallgató neve (19 karakter), eredménye százalékban (3 karakter), és a teszttábla (55 karakter), amely 'G' jelzéssel a jól, 'E' jelzéssel a hibásan megválaszolt és '-' jelzéssel a fel nem tett kérdéseket ábrázolja. A file-t egy struktúra változó (CST) alapján tölti fel a program.

Függelék:

A 'DP' program függvényeinek jegyzéke:

DP.C rész

main: gépi fôprogram

DPD.C rész

(függvényt nem tartalmaz)

DPH.C rész

(függvényt nem tartalmaz)

DPM.C rész

mov: adatmozgatás regiszterek közt
mvi: értékadás regiszternek
lxi: értékadás regiszterpárnak
ldxx: spec. értékadás memóriából
stxx: spec. adattárolás memóriába
jxx: feltételes vezérlésátadás
jmp: feltétlen vezérlésátadás
cxx: feltételes szubrutinhívás
call: feltétlen szubrutinhívás
rxx: feltételes visszatérés szubrutinból
ret: feltétlen visszatérés szubrutinból
rst: spec. szubrutinhívás
push: regiszterpár verembe mentése
pop: értékadás regiszterpárnak verembôl
in: I/O olvasás
out: I/O írás
xchg: csere HL és DE közt
xthl: csere HL és a verem közt
sphl: SP = HL
pchl: PC = HL (spec. vezérlésátadás!)

DPA.C rész

flagsA: flagek beállítása A regiszter esetén
flagsR: flagek beállítása más esetben
parity: paritásvizsgálat
axxa: aritmetikai művelet regiszterrel
axxi: aritmetikai művelet adattal

lxxa: logikai művelet regiszterrel
lxxi: logikai művelet adattal
inr: regiszter csökkentése eggyel
dcr: regiszter növelése eggyel
inx: regiszterpár csökkentése eggyel
dcx: regiszterpár csökkentése eggyel
dad: HL összeadása regiszterpárral
rot: léptetô művelet (RAL,RAR,RLC,RRC)

DPS.C rész

daa: BCD helyesbítés
cma: akkumulátor negálása
stc: CY flag beállítása egybe
cmc: CY flag negálása
rim: soros I/O olvasása
sim: soros I/O írása
ei: megszakítás engedélyezése
di: megszakítás tiltása
nop: üres
hlt: processzor stop

DPI.C rész

iccp: függvénymutató tömb feltöltôje
mask: mintaillesztô

DPP.C rész

program: parancsértelmezô
binl: karakter illesztése a stringbe
dinl: karakter törlése a stringbôl
dell: string törlése
ment: string végrehajtása
mtoc: kódkeresés a stringben
mtop: paraméter a stringben
minc: spec. stringösszehasonlító
btoc: kód vétele a memóriából
ccp: kódvégrehajtó
help: helpkezelô (F1)
run: programkód futtató (F2)
step: egylépéses programvégrehajtás (F3)
edit: regiszterjavítás (F4)
list: disassembly lista (Alt a)
load: memória töltés (F5)

save: memória mentés (F6)
trap: megszakításkezelés (F7)
res: reset (F8)
test: számonkérés (F9)

DPK.C rész

shscr: alapkép megjelenítése
shcom: kód és mnemonik megjelenítése
shcpu: regiszterállapot megjelenítése
shclp: beviteli string megjelenítése
shmsg: üzenet megjelenítése
shlsc: egyéb bevitelek megjelenítése
shtest: számonkérés megjelenítése
shlis: disassembly lista megjelenítése
shhlp: helpablak megjelenítése
shmen: helpmenü megjelenítése
selmen: kért utasításcsoport kiválasztása
shmnem: utasításcsoport megjelenítése
selmnem: kért utasítás kiválasztása
shedit: regiszterjavítás megjelenítése

DPW.C rész

mkwin: ablakkészítés
frame: keretkészítés
clrwin: ablaktörlés

DPT.C rész

ttext: teszt "intro"
tprog: teszt fôrutin
qmtr: válasz ellenôrzése

DPF.C rész

mio: memória-file kezelése
qio: teszt-file kezelése

DPZ.C rész

strdec: string - szám konverzió
decbin: szám - bináris alakú string konverzió
fbin: flag bináris konverziója
itor: szám regiszterbe töltése
rtoi: regiszter számmá alakítása

Irodalomjegyzék:

[1] U. Tietze - Ch. Schenk: Analóg és digitális áramkörök. Műszaki könyvkiadó 1990

[2] T. Moto-oka - M. Kitsuregawa: Az ötödik generációs számítógép. [A japán kihívás] Műszaki könyvkiadó 1987

[3] Madarász László: µP-hobby. Műszaki könyvkiadó 1987

[4] Krizsán György: A ZILOG mikroprocesszor családok. LSI Atsz 1984

[5] dr. Ajtonyi István: Vezérléstechnika II. Tankönyvkiadó 1988

[6] dr. Ajtonyi István: Vezérléstechnika gyakorlatok és adatgyűjtemény. Tankönyvkiadó 1986

[7] MCS-85 User's Manual. Intel 1978

[8] B. W. Kernighan - D. M. Ritchie: A C programozási nyelv. Műszaki könyvkiadó 1985

[9] Benkô Tiborné - Urbán Zoltán: IBM PC programozása Turbo C nyelven. BME Mérnöktovábbképzô Int. 1989

```
#define DEDICATED "\nUSER: Szoftvermúzeum 1995\n"

/* PRL & VG LTD. */
/*    DP V1.3    */

/* #define HERC */
#define COLOR

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include <bios.h>
#include <alloc.h>
#include <dos.h>
#include <conio.h>

/* Common Area */

unsigned int (*ccf[ 256 ])();

union RS
{
    struct
    {
        unsigned int bcr: 2;
        unsigned int der: 2;
        unsigned int hlr: 2;
        unsigned int afr: 2;

        unsigned int pcr: 2;
        unsigned int spr: 2;

        unsigned int    : 0;

        unsigned int bca: 1;      unsigned int    : 1;
        unsigned int dea: 1;      unsigned int    : 1;
        unsigned int hla: 1;      unsigned int    : 1;
        unsigned int afa: 1;      unsigned int    : 1;

        unsigned int pca: 1;      unsigned int    : 1;
        unsigned int spa: 1;      unsigned int    : 1;

        unsigned int    : 0;
    } rs;

    struct
    {
        unsigned int reg;
        unsigned int mem;
    } rm;
};

#define RS__B 0x0001
#define RS__C 0x0002
#define RS__D 0x0004
#define RS__E 0x0008
#define RS__H 0x0010
#define RS__L 0x0020
#define RS__A 0x0040
```



```
#define RS__F 0x0080

#define RS__X ( 0x0001 << ( i == 7 ? 6 : i ) )
#define RS__Y ( 0x0001 << ( j == 7 ? 6 : j ) )

#define RS_BC 0x0003
#define RS_DE 0x000C
#define RS_HL 0x0030
#define RS_AF 0x00C0

#define RS_PC 0x0300
#define RS_SP 0x0C00

#define RS_XX ( 0x0003 << ( i == 7 ? 6 : i ) )

#define RS_bc 0x0001
#define RS_de 0x0004
#define RS_hl 0x0010

#define RS_pc 0x0100
#define RS_sp 0x0400

#define RS_xx ( 0x0001 << ( i == 7 ? 6 : i ) )

union RS      com_s, com_d;

unsigned char far  *memory;
unsigned char port[  256 ];

unsigned char buffer[ 80 ];

unsigned char reg[ 12 ];

unsigned int  com;
unsigned int  par;
unsigned int  ist;
unsigned int  hlp;

/* Registers */

#define B   reg[     0 ]
#define C   reg[     1 ]
#define D   reg[     2 ]
#define E   reg[     3 ]
#define H   reg[     4 ]
#define L   reg[     5 ]
#define A   reg[     6 ]
#define F   reg[     7 ]

#define PCH reg[     8 ]
#define PCL reg[     9 ]
#define SPH reg[    10 ]
#define SPL reg[    11 ]

#define RX  reg[ i ==7 ? 6 : i ]
#define RY  reg[ j ==7 ? 6 : j ]
#define RH  reg[ i     ]
#define RL  reg[ i + 1 ]

/* DRegs */

#define BC  reg[     1 ] + reg[  0 ] * 256
```

```
#define DE  reg[     3 ] + reg[  2 ] * 256
#define HL  reg[     5 ] + reg[  4 ] * 256
#define AF  reg[     7 ] + reg[  6 ] * 256

#define PC  reg[     9 ] + reg[  8 ] * 256
#define SP  reg[    11 ] + reg[ 10 ] * 256

#define RP  reg[ i + 1 ] + reg[  i ] * 256

/* Flags */

#define  S  0x80
#define  Z  0x40
#define AC  0x10
#define  P  0x04
#define CY  0x01

#define  V  0x02
#define X3  0x08
#define X5  0x20

/* Conditions */

#define XXNZ 0
#define XXZ  1
#define XXNC 2
#define XXC  3
#define XXPO 4
#define XXPE 5
#define XXP  6
#define XXM  7

/* Parts */

#include "dpd.c"
#include "dph.c"

#include "dpp.c"

#include "dpm.c"
#include "dpa.c"
#include "dps.c"

#include "dpi.c"
#include "dpk.c"
#include "dpw.c"
#include "dpt.c"
#include "dpf.c"
#include "dpz.c"

main()
{
    DEDICATED;

    if( memory = farcalloc( 65536L, sizeof( char ) ) )
        program();
    else
        printf( "\nNot enough memory. "
                "\nNincs elég memória!" );

    return( 0 );
}
```



```
/* PRL dpa */

#define  ADD 0
#define  ADC 1
#define  SUB 2
#define  SBB 3

#define  ADI 0
#define  ACI 1
#define  SUI 2
#define  SBI 3

#define  ANA 0
#define  XRA 1
#define  ORA 2
#define  CMP 3

#define  ANI 0
#define  XRI 1
#define  ORI 2
#define  CPI 3

#define RLC 0
#define RRC 1
#define RAL 2
#define RAR 3

#define EXC 6

#define HA  ( A & 0x0f )
#define Hr  ( r & 0x0f )
#define Hx  ( x & 0x0f )

/* Flags */

unsigned char x;

flagsA()
{
    if(  A >  127 ) F |=  S; else F &=  ~S;
    if( !A        ) F |=  Z; else F &=  ~Z;
    if( !parity() ) F |=  P; else F &=  ~P;
    if(  A <    x ) F |= CY; else F &= ~CY;
    if( HA <   Hx ) F |= AC; else F &= ~AC;

    F |=   V;
    F &= ~X3; F &= ~X5;

    return( 0 );
}

flagsR( r )
unsigned char r;
{
    if(  r >  127 ) F |=  S; else F &=  ~S;
    if( !r        ) F |=  Z; else F &=  ~Z;
    if( Hr <   Hx ) F |= AC; else F &= ~AC;

    F |=   V;
    F &= ~X3;
    F &= ~X5;
```

```
    return( 0 );
}

parity()
{
    int i, j;
    unsigned char a;

    a = A;

    for( i = j = 0; i < 8; i++ )
    {
        if( a & 1 ) j++;
        a >>= 1;
    }

    return( j % 2 );
}

/* Arithmeticals */

axxa()
{
    int i, j;

    HELP( 14 );

    j = ( com & 0x18 ) / 8;
    i =   com & 0x07;

    if( i == EXC )
        com_s.rm.reg = ( j == ADC || j == SBB ) ? RS_AF : RS__A,
        com_s.rm.mem =                            RS_hl,
        com_d.rm.reg =                            RS_AF;
    else
        com_s.rm.reg = ( j == ADC || j == SBB ) ? RS_AF | RS__X :
RS__A | RS__X,
        com_d.rm.reg =                            RS_AF;

    x = A;

    if( i == EXC ) switch( j )
    {
    case ADD: A += memory[ HL ];              break;
    case ADC: A += memory[ HL ] + ( F & CY ); break;
    case SUB: A -= memory[ HL ];              break;
    case SBB: A -= memory[ HL ] + ( F & CY ); break;
    }
    else switch( j )
    {
    case ADD: A += RX;              break;
    case ADC: A += RX + ( F & CY ); break;
    case SUB: A -= RX;              break;
    case SBB: A -= RX + ( F & CY ); break;
    }

    flagsA();

    if( j == SUB || j == SBB ) F ^= ( AC + CY );

    return( 14 );
```



```
}

axxi()
{
    int j;

    HELP( 15 );

    j = ( com & 0x18 ) / 8;

    com_s.rm.reg = ( j == ACI || j == SBB ) ? RS_AF : RS__A;
    com_d.rm.reg =                            RS_AF;

    x = A;

    switch( j )
    {
    case ADI: A += par % 256;              break;
    case ACI: A += par % 256 + ( F & CY ); break;
    case SUI: A -= par % 256;              break;
    case SBI: A -= par % 256 + ( F & CY ); break;
    }

    flagsA();

    if( j == SUI || j == SBI ) F ^= ( AC + CY );

    return( 15 );
}

/* Logicals */

lxxa()
{
    int i, j;

    HELP( 16 );

    j = ( com & 0x18 ) / 8;
    i =   com & 0x07;

    if( i == EXC )
        com_s.rm.reg =                RS__A,
        com_s.rm.mem =                RS_hl,
        com_d.rm.reg = ( j != CMP ) ? RS_AF : RS__F;
    else
        com_s.rm.reg =                RS__A | RS__X,
        com_d.rm.reg = ( j != CMP ) ? RS_AF : RS__F;

    if( j == CMP ) x = A;

    if( i == EXC ) switch( j )
    {
    case ANA: x = ( A &= memory[ HL ] ); break;
    case XRA: x = ( A ^= memory[ HL ] ); break;
    case ORA: x = ( A |= memory[ HL ] ); break;
    case CMP:       A -= memory[ HL ];   break;
    }
    else switch( j )
    {
    case ANA: x = ( A &= RX );           break;
    case XRA: x = ( A ^= RX );           break;
```

```
    case ORA: x = ( A |= RX );          break;
    case CMP:       A -= RX;            break;
    }

    flagsA();

    if( j == CMP ) A = x, F ^= ( AC + CY );

    return( 16 );
}

lxxi()
{
    int j;

    HELP( 17 );

    j = ( com & 0x18 ) / 8;

    com_s.rm.reg =                RS__A;
    com_d.rm.reg = ( j != CMP ) ? RS_AF : RS__F;

    if( j == CPI ) x = A;

    switch( j )
    {
    case ANI: x = ( A &= par % 256 ); break;
    case XRI: x = ( A ^= par % 256 ); break;
    case ORI: x = ( A |= par % 256 ); break;
    case CPI:       A -= par % 256;   break;
    }

    flagsA();

    if( j == CPI ) A = x, F ^= ( AC + CY );

    return( 17 );
}

/* Counters */

inr()
{
    int i;
    unsigned char r;

    HELP( 18 );

    i = ( com & 0x38 ) / 8;

    if( i == EXC )
        com_s.rm.mem = RS_hl,
        com_d.rm.reg = RS__F,
        com_d.rm.mem = RS_hl;
    else
        com_s.rm.reg = RS__X,
        com_d.rm.reg = RS__F | RS__X;

    if( i == EXC )
        x = memory[ HL ], r = ++memory[ HL ];
    else
        x = RX,           r = ++RX;
```



```
    flagsR( r );

    return( 18 );
}

dcr()
{
    int i;
    unsigned char r;

    HELP( 18 );

    i = ( com & 0x38 ) / 8;

    if( i == EXC )
        com_s.rm.mem = RS_hl,
        com_d.rm.reg = RS__F,
        com_d.rm.mem = RS_hl;
    else
        com_s.rm.reg = RS__X,
        com_d.rm.reg = RS__F | RS__X;

    if( i == EXC )
        x = memory[ HL ], r = --memory[ HL ];
    else
        x = RX,           r = --RX;

    flagsR( r );

    F ^= AC;

    return( 18 );
}

inx()
{
    int i;

    HELP( 18 );

    i = ( com & 0x30 ) / 8;

    if( i == EXC )
        com_s.rm.reg = RS_SP,
        com_d.rm.reg = RS_SP;
    else
        com_s.rm.reg = RS_XX,
        com_d.rm.reg = RS_XX;

    if( i == EXC )
        SPH += !( ++SPL );
    else
         RH += !(  ++RL );

    return( 18 );
}

dcx()
{
    int i;
```

```
    HELP( 18 );

    i = ( com & 0x30 ) / 8;

    if( i == EXC )
        com_s.rm.reg = RS_SP,
        com_d.rm.reg = RS_SP;
    else
        com_s.rm.reg = RS_XX,
        com_d.rm.reg = RS_XX;

    if( i == EXC )
       SPH -= !( SPL-- );
    else
        RH -= !(  RL-- );

    return( 18 );
}

/* Offset */

dad()
{
    int i;
    unsigned j;

    HELP( 19 );

    i = ( com & 0x30 ) / 8;

    if( i == EXC )
        com_s.rm.reg = RS_SP | RS_HL,
        com_d.rm.reg = RS_HL | RS__F;
    else
        com_s.rm.reg = RS_XX | RS_HL,
        com_d.rm.reg = RS_HL | RS__F;

    x = H;

    if( i == EXC )
        j = L, L += SPL, H += SPH + ( L < j );
    else
        j = L, L +=  RL, H +=  RH + ( L < j );

    if( H < x ) F |= CY; else F &= ~CY;

    return( 19 );
}

/* Rotation */

rot()
{
    int j;

    HELP( 20 );

    j = ( com & 0x18 ) / 8;

    com_s.rm.reg = ( j == RAL || j == RAR ) ? RS_AF : RS__A;
    com_d.rm.reg =                            RS_AF;
```



```
    switch( j )
    {
    case RLC: x = A & 128; A <<= 1; A |= ( x / 128 );       break;
    case RRC: x = A &   1; A >>= 1; A |= ( x * 128 );       break;
    case RAL: x = A & 128; A <<= 1; A |= ( F &  CY );       break;
    case RAR: x = A &   1; A >>= 1; A |= ( F &  CY ) * 128; break;
    }

    if( x ) F |= CY; else F &= ~CY;

    return( 20 );
}
```

```
/* PRL dpd */

char code[ 256 ][ 16 ] =
{
   " NOP "          ," LXI B,xxxx "  ," STAX B "      ," INX B "
,
   " INR B "        ," DCR B "       ," MVI B,xx "    ," RLC "
,
   "not used"       ," DAD B "       ," LDAX B "      ," DCX B "
,
   " INR C "        ," DCR C "       ," MVI C,xx "    ," RRC "
,
   "not used"       ," LXI D,xxxx "  ," STAX D "      ," INX D "
,
   " INR D "        ," DCR D "       ," MVI D,xx "    ," RAL "
,
   "not used"       ," DAD D "       ," LDAX D "      ," DCX D "
,
   " INR E "        ," DCR E "       ," MVI E,xx "    ," RAR "
,
   " RIM "          ," LXI H,xxxx "  ," SHLD xxxx "   ," INX H "
,
   " INR H "        ," DCR H "       ," MVI H,xx "    ," DAA "
,
   "not used"       ," DAD H "       ," LHLD xxxx "   ," DCX H "
,
   " INR L "        ," DCR L "       ," MVI L,xx "    ," CMA "
,
   " SIM "          ," LXI SP,xxxx " ," STA xxxx "    ," INX SP "
,
   " INR M "        ," DCR M "       ," MVI M,xx "    ," STC "
,
   "not used"       ," DAD SP "      ," LDA xxxx "    ," DCX SP "
,
   " INR A "        ," DCR A "       ," MVI A,xx "    ," CMC "
,
   " MOV B,B "      ," MOV B,C "     ," MOV B,D "     ," MOV B,E "
,
   " MOV B,H "      ," MOV B,L "     ," MOV B,M "     ," MOV B,A "
,
   " MOV C,B "      ," MOV C,C "     ," MOV C,D "     ," MOV C,E "
,
   " MOV C,H "      ," MOV C,L "     ," MOV C,M "     ," MOV C,A "
,
   " MOV D,B "      ," MOV D,C "     ," MOV D,D "     ," MOV D,E "
,
   " MOV D,H "      ," MOV D,L "     ," MOV D,M "     ," MOV D,A "
,
   " MOV E,B "      ," MOV E,C "     ," MOV E,D "     ," MOV E,E "
,
   " MOV E,H "      ," MOV E,L "     ," MOV E,M "     ," MOV E,A "
,
   " MOV H,B "      ," MOV H,C "     ," MOV H,D "     ," MOV H,E "
,
   " MOV H,H "      ," MOV H,L "     ," MOV H,M "     ," MOV H,A "
,
   " MOV L,B "      ," MOV L,C "     ," MOV L,D "     ," MOV L,E "
,
   " MOV L,H "      ," MOV L,L "     ," MOV L,M "     ," MOV L,A "
,
```



```
   " MOV M,B "      ," MOV M,C "     ," MOV M,D "     ," MOV M,E "
,
   " MOV M,H "      ," MOV M,L "     ," HLT "         ," MOV M,A "
,
   " MOV A,B "      ," MOV A,C "     ," MOV A,D "     ," MOV A,E "
,
   " MOV A,H "      ," MOV A,L "     ," MOV A,M "     ," MOV A,A "
,
   " ADD B "        ," ADD C "       ," ADD D "       ," ADD E "
,
   " ADD H "        ," ADD L "       ," ADD M "       ," ADD A "
,
   " ADC B "        ," ADC C "       ," ADC D "       ," ADC E "
,
   " ADC H "        ," ADC L "       ," ADC M "       ," ADC A "
,
   " SUB B "        ," SUB C "       ," SUB D "       ," SUB E "
,
   " SUB H "        ," SUB L "       ," SUB M "       ," SUB A "
,
   " SBB B "        ," SBB C "       ," SBB D "       ," SBB E "
,
   " SBB H "        ," SBB L "       ," SBB M "       ," SBB A "
,
   " ANA B "        ," ANA C "       ," ANA D "       ," ANA E "
,
   " ANA H "        ," ANA L "       ," ANA M "       ," ANA A "
,
   " XRA B "        ," XRA C "       ," XRA D "       ," XRA E "
,
   " XRA H "        ," XRA L "       ," XRA M "       ," XRA A "
,
   " ORA B "        ," ORA C "       ," ORA D "       ," ORA E "
,
   " ORA H "        ," ORA L "       ," ORA M "       ," ORA A "
,
   " CMP B "        ," CMP C "       ," CMP D "       ," CMP E "
,
   " CMP H "        ," CMP L "       ," CMP M "       ," CMP A "
,
   " RNZ "          ," POP B "       ," JNZ xxxx "    ," JMP xxxx "
,
   " CNZ xxxx "     ," PUSH B "      ," ADI xx "      ," RST 0 "
,
   " RZ "           ," RET "         ," JZ xxxx "     ,"not used"
,
   " CZ xxxx "      ," CALL xxxx "   ," ACI xx "      ," RST 1 "
,
   " RNC "          ," POP D "       ," JNC xxxx "    ," OUT xx "
,
   " CNC xxxx "     ," PUSH D "      ," SUI xx "      ," RST 2 "
,
   " RC "           ,"not used"      ," JC xxxx "     ," IN xx "
,
   " CC xxxx "      ,"not used"      ," SBI xx "      ," RST 3 "
,
   " RPO "          ," POP H "       ," JPO xxxx "    ," XTHL "
,
   " CPO xxxx "     ," PUSH H "      ," ANI xx "      ," RST 4 "
,
   " RPE "          ," PCHL "        ," JPE xxxx "    ," XCHG "
,
```

```
    " CPE xxxx "    ,"not used"      ," XRI xx "      ," RST 5 "
,
    " RP "          ," POP PSW "     ," JP xxxx "     ," DI "
,
    " CP xxxx "     ," PUSH PSW "    ," ORI xx "      ," RST 6 "
,
    " RM "          ," SPHL "        ," JM xxxx "     ," EI "
,
    " CM xxxx "     ,"not used"      ," CPI xx "      ," RST 7 "
};
```



```
/* PRL dpf */

mio( mode )
int mode;
{
    FILE *file;
    unsigned int mptr, msta, mend;
    unsigned char *c;

    do
    {
        shlsc( "From: _____ ", 0 );
        shlsc( " ",            5 );
        highvideo();
        c = cgets( buffer );
        if( !*c ) return( shlsc( "", 0 ) );
        msta = strdec( strupr( c ), strlen( c ) );

        if( mode == SAVE )
        {
            shlsc( "To:   _____ ", 12 );
            shlsc( " ",             17 );
            highvideo();
            c = cgets( buffer );
            if( !*c ) return( shlsc( "", 0 ) );
            mend = strdec( strupr( c ), strlen( c ) );
        }
    }
    while( mode != SAVE && msta > mend );

    while( 1 )
    {
        shlsc( "   File: ________ ", 28 );
        shlsc( " ",                  36 );
        highvideo();
        c = cgets( buffer );
        if( !*c ) return( shlsc( "", 0 ) );

        strcat( c, ".85" );

        if( mode == LOAD )
        {
            if( file = fopen( c, "rb" ) )
            {
                for( mptr = msta; !feof( file ); mptr++ )
                    memory[ mptr ] = fgetc( file );
                fclose( file );

                return( shlsc( "", 0 ) );
            }
            else
            {
                shmsg( "FILE NOT FOUND      " );
            }
        }
        else
        {
            if( !fopen( c, "rb" ) && ( file = fopen( c, "wb" ) ) )
            {
                for( mptr = msta; mptr < mend; mptr++ )
                    fputc( memory[ mptr ], file );
```

```
                fputc( memory[ mend ], file );
                fclose( file );

                return( shlsc( "", 0 ) );
            }
            else
            {
                shmsg( "FILE EXISTS         " );
            }
        }
    }
}

qio( t )
char *t;
{
    FILE *file;
    unsigned char *c, d[ 4 ];
    int i;

    union
    {
        char cfl[ 80 ];
        struct CST
        {
            char cid[ 19 ];
            char cp1      ;
            char csc[  3 ];
            char cp2      ;
            char cqt[ 55 ];
            char cse      ;
        } cst;
    } f;

    f.cst.cse ='\n';
    f.cst.cp1 = ' ';
    f.cst.cp2 = ' ';

    do
    {
        shlsc( "  Name: ____________________ ", 29 );
        shlsc( " ",                             36 );
        highvideo();
        c = cgets( buffer );
    }
    while( !*c );

    itoa( par, d, 10 );
    setmem(    f.cfl, 79, ' ' );

    strncpy(   f.cst.cid,  c, strlen(  c ) );
    strncpy(   f.cst.csc,  d, strlen(  d ) );
    strncpy(   f.cst.cqt,  t,         55   );

    if( file = fopen( "DP.SCT", "at" ) )
    {
        for( i = 0; i < 80; i++ )
            fputc( f.cfl[ i ], file );

        fclose( file );
    }
```



```
    shlsc( "", 0 );

    return( 0 );
}
```

```
/* VG dph */

char chlp[ 30 ][ 605 ] =
{

/* DP */

    "                                                          "
    "  Miskolci        ________   ________     i8085 CPU      "
    "    Egyetem       ________   ________  oktatóprogram     "
    "                  ___   __  ___   __                     "
    "                  ___   ___ ___  ____                    "
    "                  ___   ___ _________                    "
    "                  ___   ___ ________                     "
    "                  ___  ____ ___                          "
    "  Irányítástech.  _________  ____      Perlaki Attila     "
    "    tanszék       ________   ____       Vincze Gábor     "
    "                                                          ",

/*  1 */

    "                                                          "
    "  MOV reg v. M, reg v. M        Adatmozgató utasítás      "
    "  MVI reg v. M, D8             Közvetlen adatbevitel      "
    "                                                          "
    "                                                          "
    "  A MOV utasítás hatására a forrás regiszter tartal-      "
    "  ma a célregiszterbe kerül.  Címzésre bármelyik re-      "
    "  gisztert fel lehet használni.  Az MVI hatására a 8      "
    "  bites adat a megcímzett regiszterbe íródik.  A me-      "
    "  móriaregisztert a HL regiszterpár címzi.                "
    "                                                         ",
/*  2 */

    "                                                          "
    "  LXI reg.p., D16   Regiszterpár közvetlen betöltése      "
    "                                                          "
    "                                                          "
    "  Az utasítás hatására a kijelölt regiszterpárba be-      "
    "  íródik a 16 bites adat. Mégpedig úgy, hogy a maga-      "
    "  sabb helyiértékü helyre  a felső byte,  az alacso-      "
    "  nyabb helyiértékü helyre az alsó byte kerül. A BC,      "
    "  DE, HL és az SP  regiszterpárok tölthetők be ennek      "
    "  az utasításnak a felhasználásával.                      "
    "                                                         ",

/*  3 */

    "                                                          "
    "  LDA A16            Az akkumulátor direkt betöltése      "
    "  LDAX reg.p.      Az akkumulátor indirekt betöltése      "
    "  LHLD A16             A HL reg.p. direkt betöltése      "
    "                                                          "
    "                                                          "
    "  Az utasítás végrehajtása után az adat az akkumulá-      "
    "  torba ill. a HL reg.p.-ba kerül  a memóriából.  Az      "
    "  adat címzése történhet direkt vagy indirekt módon,      "
    "  a címet az A16 v. a BC ill. DE reg.p. tartalmazza.      "
    "                                                         ",

/*  4 */
```



```
    "                                                          "
    "  STA A16            Az akkumulátor direkt tárolása       "
    "  STAX reg.p.       Az akkumulátor indirekt tárolása      "
    "  SHLD A16             A HL reg.p. direkt tárolása       "
    "                                                          "
    "                                                          "
    "  Az utasítás hatására az akkumulátor ill. a  HL re-      "
    "  giszterpár tartalma átkerül  arra a memória helyre      "
    "  amelynek a címét az A16 v. a BC ill. DE regiszter-      "
    "  pár tartalmazza.                                        "
    "                                                         ",

/*  5 */

    "                                                          "
    "  Jxx A16                          Feltételes ugrás       "
    "  JMP A16                    Feltétel nélküli ugrás       "
    "                                                          "
    "                                                          "
    "  A JMP utasítás hatására  az utasítás számláló (PC)      "
    "  tartalma A16 lesz,  így a vezérlés átadódik az A16      "
    "  címen lévő utasításra.  A Jxx utasításnál ha az xx      "
    "  feltétel teljesül, akkor PC tartalma A16 lesz. Más      "
    "  esetben a futás a következő címen folytatódik.          "
    "                                                         ",

/*  6 */

    "                                                          "
    "  Cxx A16                Feltételes szubrutin hívás       "
    "  CALL A16          Feltétel nélküli szubrutin hívás      "
    "                                                          "
    "  Ha az xx feltétel teljesül, akkor a PC tartalma az      "
    "  SP által meghatározott címre kerül oly módon, hogy      "
    "  az [SP-1]-re a magasabb és az [SP-2]-re az alacso-       "
    "  nyabb helyiértékü byte kerül. Az SP tartalma 2-vel      "
    "  csökken,  majd a PC-be bekerül  az A16.  Ellenkező      "
    "  esetben a következő utasítás hajtódik végre.            "
    "                                                         ",

/*  7 */

    "                                                          "
    "  Rxx          Feltételes visszatérés a szubrutinból      "
    "  RET    Feltétel nélküli visszatérés a szubrutinból      "
    "                                                          "
    "  Ha az xx szerint kódolt feltétel teljesül, akkor a      "
    "  PC-be kerül az SP által címzett  memória tartalma.      "
    "  Az [SP]-ről az alacsonyabb helyiértékü byte, és az      "
    "  [SP+1]-ről a magasabb helyiértékü byte.  Ezután az      "
    "  SP tartalma 2-vel megnövekszik.  Más esetben a fu-       "
    "  tás a következő utasítással folytatódik.                "
    "                                                         ",

/*  8 */

    "                                                          "
    "  RST n                            Megszakítás kérés      "
    "                                                          "
    "                                                          "
    "  Ez egy speciális egy byte-os CALL utasítás.  Hatá-      "
```

```
    "  sára a PC tartalma az SP által meghatározott memó-  "
    "  ria címre kerül úgy, hogy az [SP-1]-re a magasabb,  "
    "  az [SP-2]-re, az alacsonyabb helyiértékû byte ker-  "
    "  rül. Az SP tartalma 2-vel csökken, és a futás az n  "
    "  8 szorosával egyenlô címû utasítással folytatodik.  "
    "                                                      ",

/*  9 */

    "                                                      "
    "  PUSH reg.p.                          Stack írása    "
    "                                                      "
    "                                                      "
    "  Az utasítás hatására a kiválasztott  regiszter pár  "
    "  tartalma az SP által címzett  memória helyre kerül  "
    "  oly  módon, hogy  a magasabb helyiértékû  byte  az  "
    "  [SP-1] címû, az alacsonyabb helyiértékû byte pedig  "
    "  az [SP-2] címû helyre. Végrehejtás után az SP tar-  "
    "  ma 2-vel lecsökken. A reg.p. a PSW is lehet.        "
    "                                                      ",

/* 10 */

    "                                                      "
    "  POP reg.p.                          Stack olvasása  "
    "                                                      "
    "                                                      "
    "  Az utasítás hatására  az SP által címzett  memória  "
    "  helynek  a tartalma átkerül a kiválasztott regisz-  "
    "  ter párba oly módon,  hogy az [SP]-rôl  az alacso-  "
    "  nyabb helyiértékû, az [SP+1]-rôl a magasabb helyi-  "
    "  értékû byte származik. Végrehajtás után az SP tar-  "
    "  ma 2-vel  megnövekszik.  A reg.p.  a PSW is lehet.  "
    "                                                      ",

/* 11 */

    "                                                      "
    "  IN port                   Bevitel az akkumulátorba  "
    "  OUT port       Az akkumulátor tartalmának kivitele  "
    "                                                      "
    "                                                      "
    "  Az IN utasítás hatására a 8 bites címmel kiválasz-  "
    "  tott kapocs által  a 8 bites,  kétirányú adatsínre  "
    "  helyezett adat az akkumulátorba kerül. Az OUT uta-  "
    "  sítás  hatása  ellentétes  irányú  az  adatáramlás  "
    "  szempontjából, tehát az adat a kapocsra kerül.      "
    "                                                      ",

/* 12 */

    "                                                      "
    "  XCHG    A HL és DE reg.p. tartalmának felcserélése  "
    "  XTHL    A HL és  a stack  tartalmának felcserélése  "
    "                                                      "
    "                                                      "
    "  Az XCHG  utasítás  hatására  a HL reg.p.  tartalma  "
    "  felcserélödik  a  DE reg.p.  tartalmával.  Az XTHL  "
    "  utasítás hatására az L reg. tartalma az [SP],  a H  "
    "  reg. tartalma  az [SP+1] memória hely  tartalmával  "
    "  cserélödik fel.                                     "
    "                                                      ",
```



```
/* 13 */

    "                                                      "
    "  SPHL               A HL tartalmának átvitele SP-be  "
    "  PCHL               A HL tartalmának átvitele PC-be  "
    "                                                      "
    "                                                      "
    "  Az SPHL utasítás végrehajtása során a HL regiszter  "
    "  pár tartalma az SP regiszterbe kerül.  A PCHL  egy  "
    "  speciális JMP utasítás. Ennek végrehajtása során a  "
    "  HL regiszter pár tartalma a PC regiszterbe  kerül,  "
    "  azaz ugrás történik a HL reg.p. által adott címre.  "
    "                                                     ",

/* 14 */

    "                                                      "
    "               Az akkumulátor tartalmának módosítása  "
    "                                                      "
    "  ADD reg v. M             Reg. tart.-nak hozzáadása  "
    "  ADC reg v. M       Reg. és CY tart.-nak hozzáadása  "
    "  SUB reg v. M               Reg. tart.-nak kivonása  "
    "  SBB reg v. M        Reg. és CY tart.-nak kivonása   "
    "                                                      "
    "  Ezen  utasítások az összes jelzô bit tartalmát be-  "
    "  folyásolják. Az M reg.-t a HL reg.pár címzi.        "
    "                                                     ",

/* 15 */

    "                                                      "
    "               Az akkumulátor tartalmának módosítása  "
    "                                                      "
    "  ADI D8                         Közvetlen összeadás  "
    "  ACI D8                 Közvetlen összeadás CY-vel   "
    "  SUI D8                         Közvetlen kivonás    "
    "  SBI D8                 Közvetlen kivonás CY-vel     "
    "                                                      "
    "  Ezen  utasítások az összes jelzô bit tartalmát be-  "
    "  folyásolják. Az M reg.-t a HL reg.pár címzi.        "
    "                                                     ",

/* 16 */

    "                                                      "
    "               Az akkumulátor tartalmának módosítása  "
    "                                                      "
    "  ANA reg v. M                          ÉS kapcsolat   "
    "  XRA reg v. M                 Kizáró VAGY kapcsolat   "
    "  ORA reg v. M                      VAGY kapcsolat     "
    "  CMP reg v. M              Tartalom összehasonlítás   "
    "                                                      "
    "  A CMP utasítás a Z és CY jelzô bitek tartalmát be-  "
    "  folyásolja.  Az akkumulátor tartalma nem változik.  "
    "                                                     ",

/* 17 */

    "                                                      "
    "               Az akkumulátor tartalmának módosítása  "
    "                                                      "
```

```
    "  ANI D8                    Közvetlen ÉS kapcsolat  "
    "  XRI D8             Közvetlen Kizáró VAGY kapcsolat  "
    "  ORI D8                   Közvetlen VAGY kapcsolat  "
    "  CPI D8                 Közvetlen összehasonlítás  "
    "                                                     "
    "  A CPI utasítás a Z és CY jelzô bitek tartalmát be- "
    "  folyásolja.  Az akkumulátor tartalma nem változik. "
    "                                                     ",

/* 18 */

    "                                                     "
    "  INR reg v. M            Regiszter tartalom növelés "
    "  INX reg.p.          Regiszter pár tartalom növelés "
    "  DCR reg v. M         Regiszter tartalom csökkentés "
    "  DCX reg.p.       Regiszter pár tartalom csökkentés "
    "                                                     "
    "                                                     "
    "  Ezen utasítások  végrehajtása után  a kijelölt re- "
    "  giszter ill. regiszter pár tartalma növekszik ill. "
    "  csökken eggyel. Az M reg.-t  a HL reg. pár  címzi. "
    "                                                    ",

/* 19 */

    "                                                     "
    "  DAD reg.p.    Regiszter pár tartalmának hozzáadása "
    "                        HL regiszter pár tartalmához "
    "                                                     "
    "                                                     "
    "  Az utasítás végrehajtása során  a kiválasztott re- "
    "  giszter pár tartalma  a HL regiszter pár tartalmá- "
    "  hoz hozzáadódik.                                   "
    "  Az eredmény a HL regiszter párba kerül.  Az utasí- "
    "  tás csak a CY-t befolyásolja.                      "
    "                                                    ",

/* 20 */

    "                                                     "
    "                Az akkumulátor tartalmának léptetése "
    "                                                     "
    "  RRC                                        jobbra  "
    "  RLC                                         balra  "
    "  RAR                           CY-n keresztül jobbra "
    "  RAL                            CY-n keresztül balra "
    "                                                     "
    "  Ezek az utasítások  az akkumulátor tartalmának egy "
    "  helyiértékkel történô léptetését végzik.           "
    "                                                    ",

/* 21 */

    "                                                     "
    "  DAA            Az akkumulátor decimális módosítása "
    "  CMA                 Az akkumulátor komplementálása "
    "                                                     "
    "                                                     "
    "  A DAA utasítás  hatására  az akkumulátorban levô 8 "
    "  bites bináris szám  két 4 bites NBCD számmá alakul "
    "  át. Ez az utasítás a CY és az AC jelzô bitek érté- "
    "  két befolyásolja.  A CMA utasítás során az akkumu- "
```



```
    "  látor tartalma invertálódik.                       "
    "                                                    ",

/* 22 */

    "                                                     "
    "  STC                          A CY bit 1-be állítása "
    "  CMC                       A CY bit komplementálása "
    "                                                     "
    "                                                     "
    "  Az STC utasítás hatására  a CY ( carry ) jelzô bit "
    "  tartalma  1 lesz.  A  CMC utasítás  a CY jelzô bit "
    "  tartalmát invertálja.  Ezek az utasítások  a többi "
    "  jelzô bit értékét nem befolyásolják.               "
    "                                                     "
    "                                                    ",

/* 23 */

    "                                                     "
    "  SIM  A megszakításmaszk és a soros port beállítása "
    "  RIM    A megszakításmaszk és a soros port olvasása "
    "                                                     "
    "  Speciális i8085 utasítás! Az akkumulátor bitjei:   "
    "    - 0,1,2   RST X.5 maszk                          "
    "    - 3       INT eng.(RIM), maszk érvényesség (SIM) "
    "    - 4,5,6   várakozó megszakítások (RIM)           "
    "    - 4,6     RST 7.5 törlése & soros out. eng.(SIM) "
    "    - 7       I/O bit, input: RIM, output: SIM       "
    "                                                    ",

/* 24 */

    "                                                     "
    "  EI                Programmegszakítás engedélyezés "
    "  DI                    Programmegszakítás letiltás "
    "                                                     "
    "                                                     "
    "  Az EI utasítás a programmegszakítási rendszert en- "
    "  gedélyezi a következô utasítás végrehajtását köve- "
    "  tôen.  A programmegszakítási rendszert  a DI  uta- "
    "  sítás letiltja, közvetlenül elôfordulásának detek- "
    "  tálása után. A jelzô biteket nem befolyásolják.    "
    "                                                    ",

/* 25 */

    "                                                     "
    "  NOP                                 Nincs működés "
    "  HLT                                       Leállás "
    "                                                     "
    "                                                     "
    "  A NOP utasítás hatására a CPU nem hajt végre mûve- "
    "  letet egy ciklusban.                               "
    "  A HLT utasítás a CPU működését leállítja.  Vissza- "
    "  állítás interrupt-tal vagy reset-tel.  A regiszte- "
    "  rek és a jelzô bitek tartalma változatlan marad.   "
    "                                                    ",

/* 26 */

    "                                                     "
```

```
    "  Ismeretlen utasításkód !                              "
    "                                                        "
    "                                                        "
    "                                                        "
    "  Az utasítás nem szerepel  az Intel által közzétett   "
    "  táblázatban. Felhasználása inkompatibilitással ill.  "
    "  kiszámíthatatlan működéssel jár, mivel a CPU ezt a    "
    "  kódot is értelmezi valamiképpen.                      "
    "                                                        "
    "                                                       ",

/* 27 */

    "                                                       "
    "  Kiértékelhetetlen bevitel !                          "
    "                                                       "
    "                                                       "
    "                                                       "
    "                                                       "
    "  A leírt szöveg nem hasonlít egyik mnemonikra sem.    "
    "  Kérem ellenőrizni! ( Lásd még a HELP-et! )           "
    "                                                       "
    "                                                       "
    "                                                      ",

/* 28 */

    "                                                       "
    "  Paraméterhiba !                                      "
    "                                                       "
    "                                                       "
    "                                                       "
    "  Az utasítás nyolcbites adatot vár, tehát abszolut-   "
    "  értékben 255-nél nem nagyobb számot.  Az utasítást   "
    "  oly módon kell kijavítani,  hogy vagy kisebb para-   "
    "  métert adunk meg, vagy más kódot választunk.         "
    "                                                       "
    "                                                      "
};

char mehlp[ 252 ] =
{
    "                          "
    "                          "
    "  Adatmozgató utasítások  "
    "                          "
    "  Aritmetikai utasítások  "
    "                          "
    "  Speciális utasítások    "
    "                          "
    "                         "
};

char mnhlp[ 3 ][ 294 ] =
{

/* 1 */

    "                                      "
    " CALL    IN      JMP     LDA     LDAX   "
    " LHLD    LXI     MOV     MVI     OUT    "
    " PCHL    POP     PUSH    RET     RST    "
```



```
    " SHLD    SPHL    STA     STAX    XCHG   "
    " XTHL                                   "
    "                                       ",

/* 2 */

    "                                       "
    " ACI     ADC     ADD     ADI     ANA    "
    " ANI     CMP     CPI     DAD     DCR    "
    " DCX     INR     INX     ORA     ORI    "
    " RAL     RAR     RLC     RRC     SBB    "
    " SBI     SUB     SUI     XRA     XRI    "
    "                                       ",

/* 3 */

    "                                       "
    " CMA     CMC     DAA     DI      EI     "
    " HLT     NOP     RIM     SIM     STC    "
    "                                       "
    "                                       "
    "                                       "
    "                                      "
};

int index[ 3 ][ 26 ] =
{
     3,  6, 11,  5,  3,  3,  3,  2,  1,  1, 11, 13, 10,
     9,  7,  8,  4, 13,  4,  4, 12, 12,  0,  0,  0,  0,

     4, 15, 14, 14, 15, 16, 17, 16, 17, 19, 18, 18, 18,
    18, 16, 17, 20, 20, 20, 20, 14, 15, 14, 15, 16, 17,

     1, 21, 22, 21, 24, 24, 25, 25, 23, 23, 22,  0,  0,
     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
};
```

```
#define HELP( x ) if( key == KF1 ) return( x )
/* PRL dpi */

#define MOV  "01xxxxxx"
#define MVI  "00xxx110"
#define LXI  "00xx0001"
#define LDxx "00xx1010"
#define STxx "00xx0010"
#define Jxx  "11xxx010"
#define JMP  "11000011"
#define Cxx  "11xxx100"
#define CALL "11001101"
#define Rxx  "11xxx000"
#define RET  "11001001"
#define RST  "11xxx111"
#define PUSH "11xx0101"
#define POP  "11xx0001"
#define IN   "11011011"
#define OUT  "11010011"
#define XCHG "11101011"
#define XTHL "11100011"
#define SPHL "11111001"
#define PCHL "11101001"
#define AxxA "100xxxxx"
#define AxxI "110xx110"
#define LxxA "101xxxxx"
#define LxxI "111xx110"
#define INR  "00xxx100"
#define DCR  "00xxx101"
#define INX  "00xx0011"
#define DCX  "00xx1011"
#define DAD  "00xx1001"
#define ROT  "000xx111"
#define DAA  "00100111"
#define CMA  "00101111"
#define STC  "00110111"
#define CMC  "00111111"
#define RIM  "00100000"
#define SIM  "00110000"
#define DI   "11110011"
#define EI   "11111011"
#define NOP  "00000000"
#define HLT  "01110110"

/* Routines */

iccp()
{
    for( com = 0; com < 256; com++ )
    {
        if( mask( MOV  ) ) ccf[ com ] =  mov;
        if( mask( MVI  ) ) ccf[ com ] =  mvi;
        if( mask( LXI  ) ) ccf[ com ] =  lxi;
        if( mask( LDxx ) ) ccf[ com ] = ldxx;
        if( mask( STxx ) ) ccf[ com ] = stxx;
        if( mask( Jxx  ) ) ccf[ com ] =  jxx;
        if( mask( JMP  ) ) ccf[ com ] =  jmp;
        if( mask( Cxx  ) ) ccf[ com ] =  cxx;
        if( mask( CALL ) ) ccf[ com ] = call;
        if( mask( Rxx  ) ) ccf[ com ] =  rxx;
        if( mask( RET  ) ) ccf[ com ] =  ret;
```



```
        if( mask( RST  ) ) ccf[ com ] =  rst;
        if( mask( PUSH ) ) ccf[ com ] = push;
        if( mask( POP  ) ) ccf[ com ] =  pop;
        if( mask( IN   ) ) ccf[ com ] =   in;
        if( mask( OUT  ) ) ccf[ com ] =  out;
        if( mask( XCHG ) ) ccf[ com ] = xchg;
        if( mask( XTHL ) ) ccf[ com ] = xthl;
        if( mask( SPHL ) ) ccf[ com ] = sphl;
        if( mask( PCHL ) ) ccf[ com ] = pchl;
        if( mask( AxxA ) ) ccf[ com ] = axxa;
        if( mask( AxxI ) ) ccf[ com ] = axxi;
        if( mask( LxxA ) ) ccf[ com ] = lxxa;
        if( mask( LxxI ) ) ccf[ com ] = lxxi;
        if( mask( INR  ) ) ccf[ com ] =  inr;
        if( mask( DCR  ) ) ccf[ com ] =  dcr;
        if( mask( INX  ) ) ccf[ com ] =  inx;
        if( mask( DCX  ) ) ccf[ com ] =  dcx;
        if( mask( DAD  ) ) ccf[ com ] =  dad;
        if( mask( ROT  ) ) ccf[ com ] =  rot;
        if( mask( DAA  ) ) ccf[ com ] =  daa;
        if( mask( CMA  ) ) ccf[ com ] =  cma;
        if( mask( STC  ) ) ccf[ com ] =  stc;
        if( mask( CMC  ) ) ccf[ com ] =  cmc;
        if( mask( RIM  ) ) ccf[ com ] =  rim;
        if( mask( SIM  ) ) ccf[ com ] =  sim;
        if( mask( EI   ) ) ccf[ com ] =   ei;
        if( mask( DI   ) ) ccf[ com ] =   di;
        if( mask( NOP  ) ) ccf[ com ] =  nop;
        if( mask( HLT  ) ) ccf[ com ] =  hlt;

        if( !ccf[ com  ] ) ccf[ com ] =  hlt;
    }

    return( 0 );
}

mask( cm )
char *cm;
{
    int i, ix, i1;
    char cx[ 9 ], c1[ 9 ];

    *( cx + 8 ) = '\0';
    *( c1 + 8 ) = '\0';

    for( i = 0; i < 8; i++ )
    {
        *( cx + i ) = *( cm + i ) == 'x' ? '0' : '1';
        *( c1 + i ) = *( cm + i ) == '1' ? '1' : '0';
    }

    ix = strdec( cx, 8 );
    i1 = strdec( c1, 8 );

    return( ( com & ix ) == i1 );
}
```

```
/* PRL dpk */

#define POZCOM 38,20
#define POZDIS 64,20
#define POZLSC  1,22
#define POZMSG  3,20

#define POZRB   7, 4
#define POZRD   7, 5
#define POZRH   7, 6
#define POZRA   7, 7
#define POZRP   7, 9
#define POZRS   7,10

#define POZBB  12, 4
#define POZBC  21, 4
#define POZBD  12, 5
#define POZBE  21, 5
#define POZBH  12, 6
#define POZBL  21, 6
#define POZBA  12, 7
#define POZBF  21, 7

#define POZXF  63, 7

#define POZAB  38, 4
#define POZAD  38, 5
#define POZAH  38, 6
#define POZAP  38, 9
#define POZAS  38,10

#define POZL1  10,23
#define POZL2  24,23

#define POZHX  70,22

char screen[] =
{
    "+-----------------------------------------------------------
---------------+"
    "¦ INTEL 8080/8085 CPU
(C) '91 VG & PRL ¦"
    "¦-----------------------------------------------------------
---------------¦"
    "¦ BC: ____ ........ ........ ( BC ): __ __ __ __ __ __ __ __
¦"
    "¦ DE: ____ ........ ........ ( DE ): __ __ __ __ __ __ __ __  S
Z   Ac  P   Cy ¦"
    "¦ HL: ____ ........ ........ ( HL ): __ __ __ __ __ __ __ __
¦"
    "¦ AF: ____ ........ ........                                  .
. x . x . x .  ¦"
    "¦
¦"
    "¦ PC: ____                   ( PC ): __ __ __ __ __ __ __ __
FLAG        ¦"
    "¦ SP: ____                   ( SP ): __ __ __ __ __ __ __ __
¦"
    "¦-----------------------------------------------------------
---------------¦"
```



```
    "¦ BC: ____ ........ ........ ( BC ): __ __ __ __ __ __ __ __
¦"
    "¦ DE: ____ ........ ........ ( DE ): __ __ __ __ __ __ __ __  S
Z   Ac  P   Cy ¦"
    "¦ HL: ____ ........ ........ ( HL ): __ __ __ __ __ __ __ __
¦"
    "¦ AF: ____ ........ ........                                  .
. x . x . x .  ¦"
    "¦
¦"
    "¦ PC: ____                   ( PC ): __ __ __ __ __ __ __ __
FLAG        ¦"
    "¦ SP: ____                   ( SP ): __ __ __ __ __ __ __ __
¦"
    "¦-----------------------------------------------------------
---------------¦"
    "¦                            Mnemo.: _____________________ [
___: __ __ __ ]¦"
    "+-----------------------------------------------------------
---------------+"
    "
"
    "
"
    "
"
    " 1.HELP  2.RUN   3.STEP  4.EDIT  5.LOAD  6.SAVE  7.INT.  8.RES.
9.TEST  0.QUIT"
};

/* Routines */

shscr()
{

#if defined( COLOR )

    textcolor( YELLOW );

#endif

    clrscr();
    lowvideo();
    cprintf( "%s", screen );

    return( 0 );
}

shcom()
{
    int ic, ie;

    ic = com % 256;
    ie = com / 256;

#if defined( COLOR )

    textcolor( YELLOW );

#endif

    lowvideo();
```

```
    gotoxy( POZCOM );
    cprintf( "\b_____________________" );

    highvideo();
    gotoxy( POZCOM );
    cprintf( "\b%s", code[ ic ] );

    gotoxy( POZDIS );
    cprintf( "%04X:", PC );

    switch( ie )
    {
    case  0: cprintf( " %02X" "   "  "   " , ic
); break;
    case  1: cprintf( " %02X"" %02X" "   " , ic, par % 256
); break;
    case  2: cprintf( " %02X"" %02X"" %02X", ic, par % 256, par / 256
); break;

    default: cprintf( " pardon?  " );
             sound( 220 ), delay(  80 );
             sound( 110 ), delay(  80 );
             sound( 440 ), delay( 240 );
             nosound();
    }

    return( 0 );
}

shsta( page )
int page;
{

#if defined( COLOR )

    textcolor( BROWN );

#endif

    gotoxy( -4 + POZRB + page );

    switch( page ? com_d.rs.bcr : com_s.rs.bcr )
    {
    case 0:  lowvideo(); cprintf(   "BC"   ); break;
    case 1: highvideo(); cprintf(   "B"    );
              lowvideo(); cprintf(    "C"   ); break;
    case 2:  lowvideo(); cprintf(   "B"    );
            highvideo(); cprintf(    "C"   ); break;
    case 3: highvideo(); cprintf(   "BC"   ); break;
    }

    gotoxy( -4 + POZRD + page );

    switch( page ? com_d.rs.der : com_s.rs.der )
    {
    case 0:  lowvideo(); cprintf(   "DE"   ); break;
    case 1: highvideo(); cprintf(   "D"    );
             lowvideo(); cprintf(    "E"   ); break;
    case 2:  lowvideo(); cprintf(   "D"    );
            highvideo(); cprintf(    "E"   ); break;
    case 3: highvideo(); cprintf(   "DE"   ); break;
    }
```



```
    gotoxy( -4 + POZRH + page );

    switch( page ? com_d.rs.hlr : com_s.rs.hlr )
    {
    case 0:  lowvideo(); cprintf(   "HL"   ); break;
    case 1: highvideo(); cprintf(   "H"    );
             lowvideo(); cprintf(    "L"   ); break;
    case 2:  lowvideo(); cprintf(   "H"    );
            highvideo(); cprintf(    "L"   ); break;
    case 3: highvideo(); cprintf(   "HL"   ); break;
    }

    gotoxy( -4 + POZRA + page );

    switch( page ? com_d.rs.afr : com_s.rs.afr )
    {
    case 0:  lowvideo(); cprintf(   "AF"   ); break;
    case 1: highvideo(); cprintf(   "A"    );
             lowvideo(); cprintf(    "F"   ); break;
    case 2:  lowvideo(); cprintf(   "A"    );
            highvideo(); cprintf(    "F"   ); break;
    case 3: highvideo(); cprintf(   "AF"   ); break;
    }

    gotoxy( -4 + POZRP + page );

    switch( page ? com_d.rs.pcr : com_s.rs.pcr )
    {
    case 0:  lowvideo(); cprintf(   "PC"   ); break;
    case 3: highvideo(); cprintf(   "PC"   ); break;
    }

    gotoxy( -4 + POZRS + page );

    switch( page ? com_d.rs.spr : com_s.rs.spr )
    {
    case 0:  lowvideo(); cprintf(   "SP"   ); break;
    case 3: highvideo(); cprintf(   "SP"   ); break;
    }

    gotoxy( -8 + POZAB + page );

    switch( page ? com_d.rs.bca : com_s.rs.bca )
    {
    case 0:  lowvideo(); cprintf( "( BC )" ); break;
    case 1: highvideo(); cprintf( "( BC )" ); break;
    }

    gotoxy( -8 + POZAD + page );

    switch( page ? com_d.rs.dea : com_s.rs.dea )
    {
    case 0:  lowvideo(); cprintf( "( DE )" ); break;
    case 1: highvideo(); cprintf( "( DE )" ); break;
    }

    gotoxy( -8 + POZAH + page );

    switch( page ? com_d.rs.hla : com_s.rs.hla )
    {
    case 0:  lowvideo(); cprintf( "( HL )" ); break;
```

```
    case 1: highvideo(); cprintf( "( HL )" ); break;
    }

    gotoxy( -8 + POZAP + page );

    switch( page ? com_d.rs.pca : com_s.rs.pca )
    {
    case 0:  lowvideo(); cprintf( "( PC )" ); break;
    case 1: highvideo(); cprintf( "( PC )" ); break;
    }

    gotoxy( -8 + POZAS + page );

    switch( page ? com_d.rs.spa : com_s.rs.spa )
    {
    case 0:  lowvideo(); cprintf( "( SP )" ); break;
    case 1: highvideo(); cprintf( "( SP )" ); break;
    }

    return( 0 );
}

shcpu( page )
int page;
{
    int i;

#if defined( COLOR )

    textcolor( YELLOW );

#endif

        highvideo();

/* Regs in H */

    gotoxy( POZRB + page );
    cprintf( "%04X", BC );

    gotoxy( POZRD + page );
    cprintf( "%04X", DE );

    gotoxy( POZRH + page );
    cprintf( "%04X", HL );

    gotoxy( POZRA + page );
    cprintf( "%04X", AF );

    gotoxy( POZRS + page );
    cprintf( "%04X", SP );

    gotoxy( POZRP + page );
    cprintf( "%04X", PC );

    lowvideo();

/* Flags in B */

    gotoxy( POZXF + page );
    cprintf( "%s", fbin() );
```



```
/* Regs in B */

    gotoxy( POZBB + page );
    cprintf( "%s", decbin( B ) );

    gotoxy( POZBC + page );
    cprintf( "%s", decbin( C ) );

    gotoxy( POZBD + page );
    cprintf( "%s", decbin( D ) );

    gotoxy( POZBE + page );
    cprintf( "%s", decbin( E ) );

    gotoxy( POZBH + page );
    cprintf( "%s", decbin( H ) );

    gotoxy( POZBL + page );
    cprintf( "%s", decbin( L ) );

    gotoxy( POZBA + page );
    cprintf( "%s", decbin( A ) );

    gotoxy( POZBF + page );
    cprintf( "%s", decbin( F ) );

    highvideo();

/* Memory in H */

    gotoxy( POZAB + page );
    for( i = 0; i < 8; i++ )
        cprintf( "%02X ", memory[ BC + i ] );

    gotoxy( POZAD + page );
    for( i = 0; i < 8; i++ )
        cprintf( "%02X ", memory[ DE + i ] );

    gotoxy( POZAH + page );
    for( i = 0; i < 8; i++ )
        cprintf( "%02X ", memory[ HL + i ] );

    gotoxy( POZAS + page );
    for( i = 0; i < 8; i++ )
        cprintf( "%02X ", memory[ SP + i ] );

    gotoxy( POZAP + page );
    for( i = 0; i < 8; i++ )
        cprintf( "%02X ", memory[ PC + i ] );

    return( 0 );
}

shport()
{
    int i;

    lowvideo();

    gotoxy( -4 + POZL1 ); cprintf( "C1H:" );
    gotoxy( -4 + POZL2 ); cprintf( "C2H:" );
```

```
    highvideo();

    for( i = 0; i < 8; i++ )
    {
        gotoxy( 7 - i + POZL1 );
        if( port[ 0xC1 ] & ( 1 << i ) ) highvideo(); else lowvideo();
        cprintf( "_" );

        gotoxy( 7 - i + POZL2 );
        if( port[ 0xC2 ] & ( 1 << i ) ) highvideo(); else lowvideo();
        cprintf( "_" );
    }

    shhx();

    return( 0 );
}

char digt[] =
{
    "+-+  ++-++-+-  +- +- +-++-++-++-++  +-+  ++-++-+"
    "¦ ¦  ¦+-+ -¦+-¦+-+¦-+  ¦¦-¦+-¦¦-¦¦-+¦  +-¦¦- ¦- "
    "+-+  -+-++-+  - -++-+  -+-+ -+- -+-++-++-++-+-  "
};

#define D11 ( digt + d1 * 3        )
#define D12 ( digt + d1 * 3 + 0x30 )
#define D13 ( digt + d1 * 3 + 0x60 )
#define D21 ( digt + d2 * 3        )
#define D22 ( digt + d2 * 3 + 0x30 )
#define D23 ( digt + d2 * 3 + 0x60 )

shhx()
{
    int d1, d2;

    d1 = port[ 0xC0 ] / 16;
    d2 = port[ 0xC0 ] % 16;

    lowvideo();

    gotoxy( -4 + POZHX + 1 ); cprintf( "C0H:" );

    highvideo();

    gotoxy(      POZHX     ); cprintf( "%.3s%.3s", D11, D21 );
    gotoxy(      POZHX + 1 ); cprintf( "%.3s%.3s", D12, D22 );
    gotoxy(      POZHX + 2 ); cprintf( "%.3s%.3s", D13, D23 );

    return( 0 );
}

shclp()
{

    shmsg( "" );

#if defined( COLOR )

    textcolor( WHITE );

#endif
```



```
    lowvideo();
    gotoxy( POZCOM );
    cprintf( "\b_____________________" );

    highvideo();
    gotoxy( POZCOM );
    cprintf( "\b%s", buffer );

    gotoxy( POZDIS );
    cprintf( "%04X:          ", PC );

    return( 0 );
}

shmsg( msg )
char *msg;
{

#if defined( COLOR )

    textcolor( RED );

#endif

    highvideo();
    gotoxy( POZMSG );
    if( *msg )
    {
        cprintf( "%s", msg );
    }
    else
    {
        cprintf( "                           " );
    }

    return( 0 );
}

shlsc( msg, i )
int i;
char *msg;
{
    if( *msg )
    {

#if defined( COLOR )

        textcolor(  YELLOW );

#endif

        lowvideo();
        gotoxy( i + POZLSC );
        cprintf( "%s", msg );
    }
    else
    {
        gotoxy(     POZLSC );
        clreol();
    }
```

```
    return( 0 );
}

#define POZTHM  1, 1
#define POZSCR 16, 8
#define POZTTX  4, 2
#define POZTTQ 12, 4

shtest( x )
int x;
{
    int i;

    mkwin( 14, 5, 69, 17, 8, 7, " TEST " );

    highvideo();

    switch( x )
    {
    case   0:
    case 255: gotoxy( POZTHM );
              cprintf( "%s", ttext() );

              if( x )
              {
                  gotoxy( POZSCR - 1 ); cprintf( "
" );
                  gotoxy( POZSCR     ); cprintf( "  Elért eredmény:
%4d%%  ", par );
                  gotoxy( POZSCR + 1 ); cprintf( "
" );
              }

              break;

    default:  gotoxy( POZTTX );
              cprintf( "%2d. %s", x, buffer );

              for( i = 0; i < 4; i++ )
              {
                  gotoxy( POZTTQ + i * 2 );
                  cprintf( "%c. %s", 'a' + i, code[ port[ 160 + i ] ]
);
              }
    }

    lowvideo();

    while( !kbhit() );

    clrwin();

    return( 0 );
}

shlis()
{
    int i, ic, iw;

    mkwin( 14, 5, 69, 17, 8, 7, " LIST " );

    highvideo();
```



```
    do
    {
        clrscr();

        for( i = 3; i < 11; i++ )
        {
            gotoxy( 10, i );

            btoc();

            printf( "%04X:   ", PC );

            iw = com / 256 + 1;
            ic = com % 256;

            switch( iw )
            {
            case  1: cprintf( " %02X" "   "  "   " "    %s", ic,
code[ ic ] ); break;
            case  2: cprintf( " %02X"" %02X" "   " "    %s", ic, par
% 256,          code[ ic ] ); break;
            case  3: cprintf( " %02X"" %02X"" %02X""    %s", ic, par
% 256, par / 256, code[ ic ] ); break;

            default: cprintf( " %02X not used", memory[ PC ]
); break;
            }

            if( com !=1024 ) PCL += iw, PCH += ( PCL < iw ); else PCH
+= !( ++PCL );
        }

        key = bioskey( 0 );
    }
    while( key != KES );

    clrwin();

    return( 0 );
}

/* VG dpk */

shhlp()
{
    mkwin( 14, 5, 69, 17, 8, 7, hlp ? " HELP " : "" );

    highvideo();

    cputs( chlp[ hlp ] );

    lowvideo();

    while( !kbhit() );

    clrwin();

    return( 0 );
}

#define INVR 0x70
```

```
#define NORM 0x0f

shmen( s )
int s;
{
    lowvideo();
    gotoxy( 1, 1 );
    cprintf( "%s", mehlp );

    highvideo();
    textattr( INVR );
    gotoxy( 1, 2 * s + 3 );
    cprintf( "%.26s", ( mehlp + ( 2 * s + 2 ) * 26 ) );
    textattr( NORM );

    return( 0 );
}

selmen( s, x, y )
int s, x, y;
{
    lowvideo();
    mkwin( 5, 2, 32, 12, 0, 7, " MENU " );

    shmen( s );

    while( 1 ) switch( key = bioskey( 0 ) )
    {
    case KAD: s += s < 2 ? 1 : -2;
              shmen( s );
              break;

    case KAU: s -= s > 0 ? 1 : -2;
              shmen( s );
              break;

    case KCR: selmnem( s, x, y);
              break;

    case KES: clrwin();
              return( 0 );
    }
}

shmnem( s, x, y )
int s, x, y;
{
    lowvideo();
    gotoxy( 1, 1 );
    cprintf( "%s", mnhlp[ s ] );

    highvideo();
    textattr( INVR );
    gotoxy( x * 8 + 1, y + 2 );
    cprintf( "%.8s", ( mnhlp[ s ] + ( ( ( y + 1 ) * 5 + x ) * 8 ) )
);
    textattr( NORM );

    return( 0 );
}

selmnem( s, x, y )
```



```
int s, x, y;
{
    lowvideo();
    mkwin( 8, 10, 49, 18, 0, 7, " MNEMONICS " );

    shmnem( s, x, y );

    while( 1 ) switch( key = bioskey( 0 ) )
    {
    case KAD: if( y < index[ s ][ 0 ] || ( !s && !x && y <= index[ 0
][ 0 ] ) ) y++;
              shmnem( s, x, y );
              break;

    case KAU: if( y ) y--;
              shmnem( s, x, y );
              break;

    case KAR: if( ( s && ( y < index[ s ][ 0 ] && x < 5 ) || ( y ==
index[ s ][ 0 ] && x < 4 ) )
               || ( !s && y <= index[ 0 ][ 0 ] && x < 5 ) )
              {
                  x++;
                  if( x == 5 ) x = 0, y++;
              }

              shmnem( s, x, y );
              break;

    case KAL: if( y || ( x && !y ) )
              {
                  x--;
                  if( x == -1 ) x = 4, y--;
              }

              shmnem( s, x, y );
              break;

    case KCR: hlp = index[ s ][ y * 5 + x + 1 ];
              shhlp();
              bioskey( 0 );
              break;

    case KES: x = y = 0;
              clrwin();
              return( 0 );
    }
}

shedit( c, x, y )
char *c;
int x, y;
{
    lowvideo();

    if( x )
    {
        mkwin( x * 3 + 33, y + 11, x * 3 + 38, y + 13, 8, 7, "" );

        gotoxy( 2, 1 );
        if( *c )
            cprintf( "%.2s", c   );
```

```
        else
                cprintf( "%02X", par );

        while( !kbhit() );

        clrwin();
    }
    else
    {
        mkwin( 5, y + 11, 12, y + 13, 8, 7, "" );

        gotoxy( 2, 1 );
        if( *c )
            cprintf( "%.4s", c   );
        else
            cprintf( "%04X", par );

        while( !kbhit() );

        clrwin();
    }

    return;
}
```



```
/* VG dpm */

#define EXC 6

mov()
{
    int i, j;

    HELP(  1 );

    i = ( com & 0x38 ) / 8;
    j =   com & 0x07;

    if( i == EXC && j != EXC )
        memory[ HL ] = RY,
        com_s.rm.reg = RS__Y,
        com_d.rm.mem = RS_hl;

    if( i != EXC && j == EXC )
        RX = memory[ HL ],
        com_s.rm.mem = RS_hl,
        com_d.rm.reg = RS__X;

        if( i != EXC && j != EXC )
        RX = RY,
        com_s.rm.reg = RS__Y,
        com_d.rm.reg = RS__X;

    return(  1 );
}

mvi()
{
    int i;

    HELP(  1 );

    i = ( com & 0x38 ) / 8;

    if( i == EXC )
        memory[ HL ] = par % 256,
        com_d.rm.mem = RS_hl;
    else
        RX = par % 256,
        com_d.rm.reg = RS__X;

    return(  1 );
}

lxi()
{
    int i;

    HELP(  2 );

    i = ( com & 0x30 ) / 8;

    if( i == EXC )
        SPH = par / 256, SPL = par % 256,
        com_d.rm.reg = RS_SP;
    else
```

```
        RH  = par / 256, RL  = par % 256,
        com_d.rm.reg = RS_XX;

    return(  2 );
}

ldxx()
{
    int i;

    HELP(  3 );

    i = ( com & 0x30 ) / 8;

    switch( i )
    {
    case 0:
    case 2: A = memory[  RP ];
            com_s.rm.mem = RS_xx;
            com_d.rm.reg = RS__A;
            break;
    case 4: H = memory[ par + 1 ];
            L = memory[ par     ];
            com_d.rm.reg = RS_HL;
            break;
    case 6: A = memory[ par ];
            com_d.rm.reg = RS__A;
            break;
    }

    return(  3 );
}

stxx()
{
    int i;

    HELP(  4 );

    i = ( com & 0x30 ) / 8;

    switch( i )
    {
    case 0:
    case 2: memory[  RP ] = A;
            com_s.rm.reg = RS__A;
            com_d.rm.mem = RS_xx;
            break;
    case 4: memory[ par + 1 ] = H;
            memory[ par     ] = L;
            com_s.rm.reg = RS_HL;
            break;
    case 6: memory[ par ] = A;
            com_s.rm.reg = RS__A;
            break;
    }

    return(  4 );
}

jxx()
{
```



```
    int i;

    HELP(  5 );

    i = ( com & 0x38 ) / 8;

    switch( i )
    {
    case XXNZ: if( ( F & Z  ) ==  0 )  jmp();  break;
    case  XXZ: if( ( F & Z  ) ==  Z )  jmp();  break;
    case XXNC: if( ( F & CY ) ==  0 )  jmp();  break;
    case  XXC: if( ( F & CY ) == CY )  jmp();  break;
    case XXPO: if( ( F & P  ) ==  0 )  jmp();  break;
    case XXPE: if( ( F & P  ) ==  P )  jmp();  break;
    case  XXP: if( ( F & S  ) ==  0 )  jmp();  break;
    case  XXM: if( ( F & S  ) ==  S )  jmp();  break;
    }

    return(  5 );
}

jmp()
{
    HELP(  5 );

    com_d.rm.reg = RS_PC;

    PCH = par / 256;
    PCL = par % 256;

    return(  5 );
}

cxx()
{
    int i;

    HELP(  6 );

    i = ( com & 0x38 ) / 8;

    switch( i )
    {
    case XXNZ: if( ( F & Z  ) ==  0 ) call();  break;
    case  XXZ: if( ( F & Z  ) ==  Z ) call();  break;
    case XXNC: if( ( F & CY ) ==  0 ) call();  break;
    case  XXC: if( ( F & CY ) == CY ) call();  break;
    case XXPO: if( ( F & P  ) ==  0 ) call();  break;
    case XXPE: if( ( F & P  ) ==  P ) call();  break;
    case  XXP: if( ( F & S  ) ==  0 ) call();  break;
    case  XXM: if( ( F & S  ) ==  S ) call();  break;
    }

    return(  6 );
}

call()
{
    HELP(  6 );

    com_s.rm.reg = RS_PC;
    com_d.rm.reg = RS_PC | RS_SP;
```

```
    com_d.rm.mem = RS_sp;

    memory[ SP - 1 ] = PCH;
    memory[ SP - 2 ] = PCL;

    SPH -= !( SPL-- );
    SPH -= !( SPL-- );

    PCH = par / 256;
    PCL = par % 256;

    return(  6 );
}

rxx()
{
    int i;

    HELP(  7 );

    i = ( com & 0x38 ) / 8;

    switch( i )
    {
    case XXNZ: if( ( F & Z  ) ==  0 )  ret();  break;
    case  XXZ: if( ( F & Z  ) ==  Z )  ret();  break;
    case XXNC: if( ( F & CY ) ==  0 )  ret();  break;
    case  XXC: if( ( F & CY ) == CY )  ret();  break;
    case XXPO: if( ( F & P  ) ==  0 )  ret();  break;
    case XXPE: if( ( F & P  ) ==  P )  ret();  break;
    case  XXP: if( ( F & S  ) ==  0 )  ret();  break;
    case  XXM: if( ( F & S  ) ==  S )  ret();  break;
    }

    return(  7 );
}

ret()
{
    HELP(  7 );

    com_s.rm.mem = RS_sp;
    com_d.rm.reg = RS_PC | RS_SP;

    PCH = memory[ SP + 1 ];
    PCL = memory[ SP     ];

    SPH += !( ++SPL );
    SPH += !( ++SPL );

    return(  7 );
}

rst()
{
    int i;

    HELP(  8 );

    i = ( com & 0x38 );

    com_s.rm.reg = RS_PC;
```



```
    com_d.rm.reg = RS_PC | RS_SP;
    com_d.rm.mem = RS_sp;

    memory[ SP - 1 ] = PCH;
    memory[ SP - 2 ] = PCL;

    SPH -= !( SPL-- );
    SPH -= !( SPL-- );

    PCH = 0;
    PCL = i;

    return(  8 );
}

push()
{
    int i;

    HELP(  9 );

    i = ( com & 0x30 ) / 8;

    com_s.rm.reg = RS_XX;
    com_d.rm.reg = RS_SP;
    com_d.rm.mem = RS_sp;

    memory[ SP - 1 ] = RH;
    memory[ SP - 2 ] = RL;

    SPH -= !( SPL-- );
    SPH -= !( SPL-- );

    return(  9 );
}

pop()
{
    int i;

    HELP( 10 );

    i = ( com & 0x30 ) / 8;

    com_s.rm.mem = RS_sp;
    com_d.rm.reg = RS_XX | RS_SP;

    RH = memory[ SP + 1 ];
    RL = memory[ SP     ];

    SPH += !( ++SPL );
    SPH += !( ++SPL );

    return( 10 );
}

in()
{
    HELP( 11 );

    com_d.rm.reg = RS__A;
```

```
    A = port[ par % 256 ];

    return( 11 );
}

out()
{
    HELP( 11 );

    com_s.rm.reg = RS__A;

    port[ par % 256 ] = A;

    return( 11 );
}

xchg()
{
    char h, l;

    HELP( 12 );

    com_s.rm.reg = com_d.rm.reg = RS_DE | RS_HL;

    h = D, D = H, H = h;
    l = E, E = L, L = l;

    return( 12 );
}

xthl()
{
    char h, l;

    HELP( 12 );

    com_s.rm.reg = com_d.rm.reg = RS_HL;
    com_s.rm.mem = com_d.rm.mem = RS_sp;

    h = H, H = memory[ SP + 1 ], memory[ SP + 1 ] = h;
    l = L, L = memory[ SP     ], memory[ SP     ] = l;

    return( 12 );
}

sphl()
{
    HELP( 13 );

    com_s.rm.reg = RS_HL;
    com_d.rm.reg = RS_SP;

    SPH = H;
    SPL = L;

    return( 13 );
}

pchl()
{
    HELP( 13 );
```



```
    com_s.rm.reg = RS_HL;
    com_d.rm.reg = RS_PC;

    PCH = H;
    PCL = L;

    return( 13 );
}
```

```
/* PRL dpp */

/* Keys */

#define KXH  8960
#define KXR  4864

#define KXE  4608
#define KXL  9728
#define KXS  7936
#define KXI  5888

#define KXT  5120
#define KXQ  4096

#define KXA  7680

#define KF1 15104
#define KF2 15360
#define KF3 15616
#define KF4 15872
#define KF5 16128
#define KF6 16384
#define KF7 16640
#define KF8 16896
#define KF9 17152
#define KF0 17408

#define Kc1 24064

#define Ks7 23040
#define Ka7 28160
#define Kc7 25600

#define Kc8 25856

#define KBS  3592
#define KDL 21248
#define KIN 20992
#define KHM 18176
#define KEN 20224
#define KPU 18688
#define KPD 20736
#define KCR  7181
#define KES   283
#define KTB  3849

#define KAU 18432
#define KAD 20480
#define KAL 19200
#define KAR 19712

/* Others */

#define LOAD  KF5
#define SAVE  KF6

#define HLPMIN  0
#define HLPMAX 25

#define OLD     0
```



```
#define NEW     8

#define WAIT while( !kbhit() )

#define INTR 0x08

int key;

/* Routines */

program()
{
    int i;
    char *c;

    _AX = 0x0100;
    _CX = 0x0f00;

    geninterrupt( 0x10 );

#if defined( COLOR )

    textcolor( LIGHTRED + BLINK );

#endif

    cprintf( "PLEASE WAIT!" );

    iccp();

    shscr();
    shcpu( OLD );
    shcpu( NEW );

    hlp = 0;
    shhlp();

    c = buffer;

    i = 1;

    while( 1 )
    {
        *( c + 0 ) = ' ';
        *( c + i ) ='\0';

        switch( key = bioskey( 0 ) )
        {
            case KES:                       break;

            case KPU:
            case KPD:
/* Help */  case KF1: if( i > 1 ) { ment( c, i ); break; }
            case Kc1: case KXH:     help();break; /* Only V1.1 */
/* Run  */  case KF2: case KXR: i =  run();break;
/* Step */  case KF3:           i = step();break;
/* Edit */  case KF4: case KXE: i = edit();break; /* Only V1.2 */
/* List */            case KXA: i = list();break; /* Only V1.2 */
/* Load */  case KF5: case KXL: i = load();break;
/* Save */  case KF6: case KXS: i = save();break;
/* Int. */  case KF7: case KXI: i = trap();break;
/* Res. */  case KF8:           i =  res();break;
```

```
/* Test */  case KF9: case KXT: i = test();break; /* Only V1.2 */
/* Quit */  case KF0: case KXQ:   clrscr();return( 0 );

/* Rst5 */  case Ks7:
/* Rst6 */  case Ka7:
/* Rst7 */  case Kc7:
/* Nmi  */  case Kc8:           i = trap();break;

            case KHM: case KEN:

            case KAU: case KAD:
            case KAL: case KAR:

            case KIN:
/* CR   */  case KCR: i = ment( c, i );    break;
/* BS   */  case KBS: i = dinl( c, i );    break;
/* DEL  */  case KDL: i = dell( c );                break;

/* Text */  default:  i = binl( c, i );    break;
        }
    }
}

binl( c, i )
int i;
char *c;
{
    if( i < 24 && isprint( key & 0x00FF ) )
    {
        *( c + i     ) = key;
        *( c + i + 1 ) ='\0';

        i++;
    }

    shclp();

    return( i );
}

dinl( c, i )
int i;
char *c;
{
    if( i >  1 )
    {
        *( c + i - 1 ) ='\0';

        i--;
    }

    shclp();

    return( i );
}

dell( c )
char *c;
{
    *( c + 1 ) ='\0';

    shclp();
```



```
    return( 1 );
}

ment( c, i )
int i;
char *c;
{
    if( i != 1 && ( key == KCR || key == KF1 || key == KIN ) )
    {
        *( c         ) = ' ';
        *( c + i     ) = ' ';
        *( c + i + 1 ) ='\0';

        c = strupr( c );

        par = mtop( c );
        com = mtoc( c );

        if( com==1024                )              hlp = 27;
        if( com < 512 && par > 65279 ) par %=  256;
        if( com < 512 && par >   256 ) com  = 1024, hlp = 28;

        shcom();

        shcpu( OLD );

        if( com < 1024 ) ccp();

        shsta( OLD );
        shsta( NEW );

        shcpu( NEW );
    }
    else
    {
        switch( key )
        {
        case KHM: PCL  = 0x00;      break;
        case KEN: PCL  = 0xff;      break;

        case KAD: PCH--;            break;
        case KAU: PCH++;            break;
        case KAL: PCH -= !( PCL-- );       break;
        case KAR: PCH += !( ++PCL );       break;
        }

        btoc();

        shcom();

        if( key == KCR )
        {
            shcpu( OLD );

            ccp();

            shsta( OLD );
            shsta( NEW );

            shcpu( NEW );
        }
```

```
    }

    if( hlp > 25 ) help();

    return( 1 );
}

mtoc( c )
char *c;
{
    int i, j;

    if( strstr( c, "RST" ) )
        strcat( c, "0.. " );
    else
        strcat( c, "B.. " );

    for( i = 0; i < 256; i++ )
        if( j = minc( code[ i ], c ) ) break;

    if( strstr( code[ i ], "xxxx" ) ) i += 512; else
    if( strstr( code[ i ], "xx"   ) ) i += 256;

    return( j ? i : 1024 );
}

mtop( c )
char *c;
{
    char *d, *e;

    d = strchr( ( c + 1 ), ',' );
    e = strchr( ( c + 1 ), ' ' );

    c = d ? ( d + 1 ) : ( e + 1 );

    return( *c ? strdec( c, strlen( c ) - 1 ) : 0 );
}

minc( a, b )
char *a, *b;
{
    int i;

    i = 0;

    while( 1 )
    {
        if( !*( a + i ) || *( a + i ) == 'x' )
            return( 1 );

        if( !*( b + i ) )
            return( 0 );

        if( *( a + i ) == *( b + i ) || *( b + i ) == '.' || isspace(
*( b + i ) ) )
            i++;
        else
            b++, i = 0;
    }
}
```



```
btoc()
{
    com = memory[ PC     ];
    par = memory[ PC + 1 ]
        + memory[ PC + 2 ] * 256;

    if( strstr( code[ com ], "not used" ) ) return( com  = 1024 );
    if( strstr( code[ com ], "xxxx"     ) ) return( com +=  512 );
    if( strstr( code[ com ], "xx"       ) ) return( com +=  256 );
}

ccp()
{
    int iw;

    com_s.rm.reg = com_d.rm.reg = com_s.rm.mem = com_d.rm.mem = 0;

    iw = ( com == 1024 ) ? 0 : com / 256 + 1;

    if( key == KCR || key == KIN ) switch( iw )
    {
    case 3: memory[ PC + 2 ] = par / 256;
    case 2: memory[ PC + 1 ] = par % 256;
    case 1: memory[ PC     ] = com % 256;
    }

    if( key != KF1 ) PCL += iw, PCH += ( PCL < iw );

    time( port + 10 );

    if( key != KIN ) hlp = ( com == 1024 ) ? 26 : ccf[ com % 256 ]();

    if( key == KF1 ) shhlp();

    if( key == Ks7
     || key == Ka7
     || key == Kc7 ) PCL += 0x04;

    if( key == Kc8 ) PCL  = 0x56;

    if( port[ 7 ] ) sound( port[ 7 ] * 16 ); else nosound();

    shport();

    outportb( 0x03BC, port[ 0x01 ] );
    outportb( 0x0378, port[ 0x02 ] );
    outportb( 0x037C, port[ 0x03 ] );

    return( 0 );
}

help()
{
    int s, x, y;

    if( key == KPU ) hlp += ( hlp < HLPMAX );
    if( key == KPD ) hlp -= ( hlp > HLPMIN );

    if( key != KF1 && hlp ) shhlp();

    if( key == KF1 || hlp > 25 )
    {
```

```
        x = y = s = 0;

        selmen( s, x, y );

        hlp = 0;
    }

    return( 1 );
}

run()
{
    int j;

    shcpu( OLD );

    shmsg( "Program is running.    " );

    while( 1 )
    {
        btoc();

        if( com == 0x00 || com == 0x76 || kbhit() ) break;

        shcom();

        ccp();

        shcpu( NEW );
    }

    if( hlp > 25 ) bioskey( 0 ), help();

    shmsg( "" );

    return( 1 );
}

step()
{
    shmsg( "Stepping in the memory." );

    btoc();

    shcpu( OLD );

    shcom();

    ccp();

    shsta( OLD );
    shsta( NEW );

    shcpu( NEW );

    if( hlp > 25 ) help();

    WAIT;
    shmsg( "" );

    return( 1 );
}
```



```
edit()
{
    char *c;
    int i, x, y;

    c = buffer;

    *c = '\0';

    x = y = i = 0;

    shmsg( "Editing." );

    rtoi( x, y );
    shedit( c, x, y );

    while( 1 )
    {
        switch( key = bioskey( 0 ) )
        {
        case KAD: if( y < 6 && !*c )
                  {
                      if( x )
                          y += ( y == 2 ) ? 3 : 1;
                      else
                          y += ( y == 3 ) ? 2 : 1;
                  }

                  break;

        case KAU: if( y > 0 && !*c )
                  {
                      if( x )
                          y -= ( y == 5 ) ? 3 : 1;
                      else
                          y -= ( y == 5 ) ? 2 : 1;
                  }

                  break;

        case KAL: if( x > 1 && !*c )
                      x--;

                  break;

        case KAR: if( x && x < 8 && !*c )
                      x++;

                  break;

        case KTB: if( y != 3 && !*c )
                      x = !x;

                  break;

        case KDL: i  =  0 ;
                  *c ='\0';

                  break;

        case KBS: if( i )
```

```
                {
                    *( c + i - 1 ) ='\0';

                    i--;
                }

                break;

        case KCR: if( *c )
                {
                    itor( c, x, y );
                    shcpu( NEW );
                    i = 0;
                    *c = '\0';
                }

                break;

        case KES:
        case KF0: shmsg( "" );
                  shcpu( NEW );
                  return( 1 );

        default:  if( isxdigit( ( char )key ) && ( i < x ? 4 : 2 ) )
                  {
                      *( c + i     ) = key;
                      *( c + i + 1 ) ='\0';

                      i++;
                  }

                  break;
        }

        rtoi( x, y );
        shedit( c, x, y );
    }
}

list()
{
    unsigned char l, h;

    l = PCL, h = PCH;

    shlis();

    PCL = l, PCH = h;

    return( 1 );
}

load()
{
    shcpu( OLD );

    shmsg( "Loading memory.        " );

    mio( LOAD );

    shcpu( NEW );
```



```
    shmsg( "" );

    return( 1 );
}

save()
{
    shcpu( OLD );

    shmsg( "Saving memory.         " );

    mio( SAVE );

    shcpu( NEW );

    shmsg( "" );

    return( 1 );
}

trap()
{
    if( ( ist & INTR ) || key == Kc8 )
    {
        shmsg( "Interrupt.             " );

        switch( key )
        {
        case KF7:
        case KXI: com = port[ 0xff ];      break;

        case Ks7: com = 0xef; ist |= 0x01; break;
        case Ka7: com = 0xf7; ist |= 0x02; break;
        case Kc7: com = 0xff; ist |= 0x04; break;

        case Kc8: com = 0xff;              break;
        }

        shcom();
        shcpu( OLD );
        ccp();
        shcpu( NEW );
    }
    else
    {
        shmsg( "Disable interrupt.     " );
    }

    WAIT;
    shmsg( "" );

    return( 1 );
}

res()
{
    shscr();

    shmsg( "Processor reset.       " );
    shcpu( OLD );

    setmem(  reg,  12, 0 );
```

```
    setmem( port, 256, 0 );

    nosound();

    shcpu( NEW );

    WAIT;
    shmsg( "" );

    return( 1 );
}

test()
{
    par = ( key == KXT ) ? 12 : 36;

    tprog();

    return( 1 );
}
```



```
/* VG dps */

#define INTR 0x08

daa()
{
    int i, j;

    HELP( 21 );

    com_s.rm.reg = com_d.rm.reg = RS_AF;

    x = A;

    i = ( A & 0x0f );

    if( i > 9 || ( F & AC ) ) A += 0x06;

    j = ( A & 0xf0 ) / 16;

    if( j > 9 || ( F & CY ) ) A += 0x60;

    flagsA();

    F &= ( S + Z + P );

    return( 21 );
}

cma()
{
    HELP( 21 );

    com_s.rm.reg = com_d.rm.reg = RS__A;

    A = ~A;

    return( 21 );
}

stc()
{
    HELP( 22 );

    com_d.rm.reg = RS__F;

    F |= CY;

    return( 22 );
}

cmc()
{
    HELP( 22 );

    com_s.rm.reg = com_d.rm.reg = RS__F;

    F ^= CY;

    return( 22 );
}
```

```
rim()
{
    HELP( 23 );

    com_d.rm.reg = RS__A;

    A = ist;

    return( 23 );
}

sim()
{
    HELP( 23 );

    com_s.rm.reg = RS__A;

    switch( A & 0x48 )
    {
    case 0x00: ist = A & 0x08; break;
    case 0x08: ist = A & 0x0f; break;
    case 0x40: ist = A & 0x88; break;
    case 0x48: ist = A & 0x8f; break;
    }

    return( 23 );
}

ei()
{
    HELP( 24 );

    ist |=  INTR;

    return( 24 );
}

di()
{
    HELP( 24 );

    ist &= ~INTR;

    return( 24 );
}

nop()
{
    HELP( 25 );

    return( 25 );
}

hlt()
{
    shmsg( "Processor stop.       " );

    HELP( 25 );

    return( 25 );
}
```



```
/* PRL dpt.c */

#define qLDAX "000x1010"
#define qSTAX "000x0010"
#define qLDA  "00111010"
#define qSTA  "00110010"
#define qLHLD "00101010"
#define qSHLD "00100010"

#define qADD  "10000xxx"
#define qADC  "10001xxx"
#define qSUB  "10010xxx"
#define qSBB  "10011xxx"
#define qANA  "10100xxx"
#define qXRA  "10101xxx"
#define qORA  "10110xxx"
#define qCMP  "10111xxx"
#define qADI  "11000110"
#define qACI  "11001110"
#define qSUI  "11010110"
#define qSBI  "11011110"
#define qANI  "11100110"
#define qXRI  "11101110"
#define qORI  "11110110"
#define qCPI  "11111110"

#define qRLC  "00000111"
#define qRRC  "00001111"
#define qRAL  "00010111"
#define qRAR  "00011111"

unsigned char cques[ 56 ][ 52 ] =
{
     MOV    "Adatmozgatás regiszterek közt:           ",
     MVI    "Értékadás regiszternek:                  ",
     LXI    "Értékadás regiszterpárnak:               ",
    qLDA    "Akkumulátor feltöltése címről:           ",
    qSTA    "Akkumulátor tárolása címre:              ",
    qLDAX   "Akkumulátor feltöltése regp. mutatóról:  ",
    qSTAX   "Akkumulátor tárolása regp. mutatóra:     ",
    qLHLD   "HL regiszterpár feltöltése címről:       ",
    qSHLD   "HL regiszterpár tárolása címre:          ",
     Jxx    "Feltételes ugrás:                        ",
     JMP    "Feltétlen ugrás:                         ",
     Cxx    "Feltételes szubrutinhívás:               ",
     CALL   "Feltétlen szubrutinhívás:                ",
     Rxx    "Feltételes visszatérés szubrutinból:     ",
     RET    "Feltétlen visszatérés szubrutinból:      ",
     RST    "Fixcímü szubrutinhívás megszakításokhoz: ",
     PUSH   "Regiszterpár verembe mentése:            ",
     POP    "Regiszterpár feltöltése veremből:        ",
     IN     "Akkumulátor feltöltése címzett portról:  ",
     OUT    "Akkumulátor kiírása címzett portra:      ",
     XCHG   "Csere DE és HL regiszterpárok közt:      ",
     XTHL   "Csere a verem és HL regiszterpár közt:   ",
     SPHL   "SP feltöltése HL értékével:              ",
     PCHL   "PC feltöltése HL értékével: ( ugrás! )   ",
    qADD    "Regiszter összeadása A-val, átv. nélkül: ",
    qADC    "Regiszter összeadása A-val, átvitellel:  ",
    qSUB    "Regiszter kivonása A-ból, átv. nélkül:   ",
    qSBB    "Regiszter kivonása A-ból, átvitellel:    ",
```

```
    qANA    "Regiszter 'ÉS' akkumulátorral:           ",
    qXRA    "Regiszter 'KIZ. VAGY' akkumulátorral:    ",
    qORA    "Regiszter 'VAGY' akkumulátorral:         ",
    qCMP    "Regiszter összehasonlítása akkumulátorral",
    qADI    "Adat összeadása A-val, átv. nélkül:      ",
    qACI    "Adat összeadása A-val, átvitellel:       ",
    qSUI    "Adat kivonása A-ból, átv. nélkül:        ",
    qSBI    "Adat kivonása A-ból, átvitellel:         ",
    qANI    "Adat 'ÉS' akkumulátorral:                ",
    qXRI    "Adat 'KIZ. VAGY' akkumulátorral:         ",
    qORI    "Adat 'VAGY' akkumulátorral:              ",
    qCPI    "Adat összehasonlítása akkumulátorral:    ",
     INR    "Regiszter értékének növelése eggyel:     ",
     DCR    "Regiszter értékének csökkentése eggyel:  ",
     INX    "Regiszterpár értékének növelése eggyel:  ",
     DCX    "Regiszterpár értékének csökkentése eggyel",
     DAD    "HL összeadása regiszterpárral:           ",
    qRRC    "Akkumulátor körbeléptetése jobbra:       ",
    qRLC    "Akkumulátor körbeléptetése balra:        ",
    qRAR    "Akkumulátor eltolása jobbra és CY flagbe:",
    qRAL    "Akkumulátor eltolása balra és CY flagbe: ",
     DAA    "Akkumulátor BCD helyesbítése:            ",
     CMA    "Akkumulátor invertálása:                 ",
     STC    "CY flag beállítása '1'-be:               ",
     CMC    "CY flag invertálása:                     ",
     DI     "Megszakítás tiltása:                     ",
     EI     "Megszakítás engedélyezése:               ",
     HLT    "Processzor megállítása:                  "
};

ttext()
{
    return( "                                                  "
            "  ________  ________    _____   ________  ________  "
            "  ________  ________  ________  ________  ________  "
            "     ___     ___     ____   __     ____    ___     "
            "     ___     ___     ____         ____    ___     "
            "     ___     ______    _____      ____    ___     "
            "     ___     _______     ___    ____     ___     "
            "     ___     ___      __   ____  ____     ___     "
            "    ____    ________  _______  ________  ____    "
            "     ____    ________   _____    ________  ____  "
);
}

#define QPP 160 + ofs

tprog()
{
    unsigned int i, j, x, qvo;
    unsigned char tabl[ 80 ], tipp, mix, good, ofs;

    setmem( tabl, 80, '-' );
    qvo = par;
    par = 0;

    randomize();

    shtest( 0 );
    bioskey( 0 );
```



```
    for( i = 0; i < qvo; i++ )
    {
        while( 1 )
        {
            x = rand() % 55;

            if( tabl[ x ] == '-' ) break;
        }

        tabl[ x ] = 'E';

        mix  = rand();
        good = qmtr( x );
        tipp = 0xff;

        for( j = 0; j < 4; j++ )
        {
            ofs = ( j + mix / 64 ) % 4;

            switch( j )
            {
            case 0: port[ QPP ] =   good;         break;
            case 1: port[ QPP ] = ( good & mix ); break;
            case 2: port[ QPP ] = ( good | mix ); break;
            case 3: port[ QPP ] = ( good ^ mix ); break;
            }

        }

        strcpy( buffer, ( cques[ x ] + 8 ) );

        while( 1 )
        {
            shtest( i + 1 );

            key = bioskey( 0 );

            if( key == KES || key == KF0 || key == KXQ ) return( 1 );

            tipp = key;

            if( tipp >= 'a' && tipp <= 'd' ) break;

            sound( 220 ), delay(  80 );
            sound( 110 ), delay(  80 );
            sound( 440 ), delay( 240 );
            nosound();
        }

        com = port[ 160 + tipp - 'a' ];

        if( mask( cques[ x ] ) ) tabl[ x ] = 'G', par++;
    }

    par = par * 100 / qvo;

    shtest( 255 );

    qio( tabl );

    return( 0 );
}
```

```
qmtr( x )
int x;
{
    int j;
    unsigned char c[ 9 ];

    strncpy( c, cques[ x ], 8 );

    for( j = 0; j < 8; j++ )
        if( *( c + j ) == 'x' ) *( c + j ) = '0';

    j = strdec( c, 8 );

    return( j );
}
```



```
/* VG dpw */

#define MAXWINDOW 100

struct savewin
{
    char *save;
    int  x1, y1, x2, y2, cb, cp;
};

signed stp = -1;

struct savewin winfo[ MAXWINDOW ];
mkwin( x1, y1, x2, y2, back, pen, header )
int x1, y1, x2, y2, back, pen;
char *header;
{
    int wsize;
    char *wptr;

    window( 1, 1, 80, 25 );
    wsize = ( x2 - x1 + 1 ) * ( y2 - y1 + 1 );
    wptr = ( char * )malloc( 2 * wsize );
    gettext( x1, y1, x2, y2, wptr );

    winfo[ ++stp ].save = wptr;

    winfo[ stp ].x1 = x1;
    winfo[ stp ].y1 = y1;
    winfo[ stp ].x2 = x2;
    winfo[ stp ].y2 = y2;
    winfo[ stp ].cb = back;
    winfo[ stp ].cp = pen;

    textbackground( back );
    textcolor( pen );

    frame( x1, y1, x2, y2, header );

    window( x1 + 1, y1 + 1, x2 - 1, y2 - 1 );

    clrscr();
    gotoxy( 1, 1 );

    return( 0 );
}

int frame( x1, y1, x2, y2, h )
int x1, y1, x2, y2;
char *h;
{
    int i, hx, wx;
    char *sor;

    lowvideo();

    sor = ( char * )malloc( 80 );

    memset( sor, 205, x2 - x1 );

    sor[ 0 ] = 201;
```

```
    sor[ x2 - x1 ] = 187;
    sor[ x2 - x1 + 1 ] = '\0';
    gotoxy( x1, y1 );
    cprintf( "%s", sor );

    sor[ 0 ] = 200;
    sor[ x2 - x1 ] = 188;
    gotoxy( x1, y2 );
    cprintf( "%s", sor );

    for( i = y1 + 1; i <= y2 - 1; i++ )
    {
        gotoxy( x1, i); putch( 186 );
        gotoxy( x2, i); putch( 186 );
    }

    wx = x2 - x1;
    hx = strlen( h );
    gotoxy( x1 + ( wx - hx ) / 2 + 1 , y1 );

    highvideo();

    cprintf( "%s", h );
    free( sor );

    return( 0 );
}

clrwin()
{
    int x1, x2, y1, y2;

    x1 = winfo[ stp ].x1;
    y1 = winfo[ stp ].y1;
    x2 = winfo[ stp ].x2;
    y2 = winfo[ stp ].y2;

    puttext( x1, y1, x2, y2, winfo[ stp ].save );
    free( winfo[ stp-- ].save );

    if( stp >= 0 )
    {
        x1 = winfo[ stp ].x1;
        y1 = winfo[ stp ].y1;
        x2 = winfo[ stp ].x2;
        y2 = winfo[ stp ].y2;
        window( x1 + 1, y1 + 1, x2 - 1, y2 - 1 );
        textbackground( winfo[ stp ].cb );
        textcolor( winfo[ stp ].cp );
    }
    else
    {
        window( 1, 1, 80, 25 );
    }

    return( 0 );
}
```



```
/* VG dpz */

strdec( idat, i )        /*  a stringet decimalis szamma alakitja  */
char *idat;
int i;
{
    int j, besz, sg;

    besz = 0;

    if( *( idat ) == '-' )
        sg = 1;
    else
        sg = 0;

    if( *( idat + i - 1 ) != 'H' )
    {
        if( i < 6 )
            for( j = 0 + sg; j < i; j++ )
                if( *( idat + j ) >= '0' && *( idat + j ) <= '9' )
                    besz = 10 * besz + *( idat + j ) - '0';
                else
                    return( 0 );

        else
            for( j = 0 + sg; j < i; j++ )
                if( *( idat + j ) == '0' || *( idat + j ) == '1' )
                    besz = 2 * besz + *( idat + j ) - '0';
                else
                    return( 0 );
    }

    if( *( idat + i - 1 ) == 'H' )
        for( j = 0 + sg; j < i - 1; j++ )
        {
            if( *( idat + j ) >= '0' && *( idat + j ) <= '9' )
                besz = 16 * besz + *( idat + j ) - '0';
            else if( *( idat + j ) >= 'A' && *( idat + j ) <= 'F' )
                besz = 16 * besz + *( idat + j ) - 'A' + 10;
            else
                return( 0 );
        }

    if( sg == 1 ) besz = -besz;

    return( besz );
}


decbin( d )                         /*  dec-bin konverziot vegez  */
int d;                              /*   8 bites szamok eseten    */
{
    int i, b;
    unsigned char *conv;

    conv = buffer;

    b = 0;

    for( i = 7; i >= 0; i-- )
    {
```

```
        b = d / pow( 2, i );
        *( conv + 7 - i ) = b + '0';
        d -= b * pow( 2, i );
    }

    *( conv + 8 ) = '\0';

    return( conv );
}


fbin()                          /*  flag binaris atalakitasa  */
{
    int i, b, d;
    unsigned char *fconv;

    fconv = buffer;

    b = 0;
    d = F;

    for( i = 7; i >= 0; i-- )
    {
        b = d / pow( 2, i );
        *( fconv + 2 * ( 7 - i ) ) = b + '0';
        *( fconv + 2 * ( 7 - i ) + 1 ) = ' ';
        d -= b * pow( 2, i );
    }

    *( fconv + 4 ) = *( fconv + 8 ) = *( fconv + 12 ) = 'x';

    *( fconv + 15 ) = '\0';

    return( fconv );
}

itor( c, x, y )
char *c;
int x, y;
{
    unsigned int h, num;

    h = strlen( c );

    *( c + h     ) = 'H';
    *( c + h + 1 ) = '\0';

    strupr( c );

    num = strdec( c, h + 1 );

    if( !x ) switch( y )
    {
    case 0: B   = num / 256; C   = num % 256; break;
    case 1: D   = num / 256; E   = num % 256; break;
    case 2: H   = num / 256; L   = num % 256; break;
    case 3: A   = num / 256; F   = num % 256; break;
    case 5: PCH = num / 256; PCL = num % 256; break;
    case 6: SPH = num / 256; SPL = num % 256; break;
    }
    else switch( y )
    {
```



```
    case 0: memory[ BC + x - 1 ] = num % 256; break;
    case 1: memory[ DE + x - 1 ] = num % 256; break;
    case 2: memory[ HL + x - 1 ] = num % 256; break;
    case 5: memory[ PC + x - 1 ] = num % 256; break;
    case 6: memory[ SP + x - 1 ] = num % 256; break;
    }

    return( 0 );
}

rtoi( x, y )
int x, y;
{
    if( !x ) switch( y )
    {
    case 0: par = BC; break;
    case 1: par = DE; break;
    case 2: par = HL; break;
    case 3: par = AF; break;
    case 5: par = PC; break;
    case 6: par = SP; break;
    }
    else switch( y )
    {
    case 0: par = memory[ BC + x - 1 ]; break;
    case 1: par = memory[ DE + x - 1 ]; break;
    case 2: par = memory[ HL + x - 1 ]; break;
    case 5: par = memory[ PC + x - 1 ]; break;
    case 6: par = memory[ SP + x - 1 ]; break;
    }

    return( 0 );
}
```