A MIKROPROCESSZOR

A mikroprocesszor olyan nagy bonyolultságú félvezető eszköz, amely a digitális számítógép központi egységének a feladatait végzi el. Dekódolja az uatasításokat, vezérli a műveletek elvégzéséhez szükséges belső adatforgalmat és a csatlakozó perifériális berendezések tevékenységét. Felépítése hasonló a nagygépek központi felépítéséhez.
Részei:

- ˇ vezérlőegység (továbbiakban lásd: CU);
- ˇ aritmetikai - logikai egység (továbbiakban lásd: ALU);
- ˇ regiszterek.

A CU belül értelmezi az utasításokat és végrehajtásuk céljából összehangoltan vezérli a számítógép többi egységének működését úgy, hogy az események a programnak megfelelő helyes sorrendben és időben következzenek be. Ezt vezérlőjelek segítségével teszi. Biztosítja, hogy a megfelelő adatok a megfelelő helyen és időben rendelkezésre álljanak. Irányítja az áramkörök működését. Az utasításszámláló regiszter (lásd később) segítségével kiolvastatja a memóriából annak a memóriarekesznek a tartalmát, amely a soron következő utasítást tárolja. Az utasítás műveleti kódrésze alapján meghatározza, hogy sorrendben milyen műveletet kell végrehajtani. Az utasítás alapján értelmezi, hogy milyen címen találhatók a műveletben résztvevő adatok, vezérli ezek kiolvasását, és a megfelelő regiszterbe történő továbbításukat. Az aritmetikai - logikai egységgel (lásd később) végrehajtatja a megfelelő műveletet és beállítja az utasításszámláló új tartalmát. Engedélyező jeleinek sorrendjét az utasításregiszterben (lásd később) tárolt utasításkód bitmintáinak megfelelően generálja és ezzel irányítja az adatfolyamatot. Időzítő jelekkel ütemezve működik.
Az ALU - mint a nevében is benne van - a CPU-n belül a számítási és logikai műveletekt végzi el. Az ALU alkalmas bináris és többnyire decimális összeadásra, Boole algebrai műveletek elvégzésére, komplemensképzésre, valamint adatok léptetésére bitenként jobbra vagy balra. Minden egyéb adatkezelési művelet, amelynek elvégzése a CPU feladata, felbontható az előbb felsorolt alapműveletekre. Rendszerint fixpontos bináris összeadóból, komplementálóból, léptető regiszterből és logikai műveleteket végző részből áll.
A regiszterek gyors írható - olvasható munkatárak. A különböző regiszterek szigorúan meghatározott feladatokhoz vannak hozzárendelve, emiatt korlátozott funkció betöltésére alkalmasak. A belső sínrendszeren keresztül tartanak kapcsolatot a processzor más részeivel.
Felépítésük: A regiszterek felépítése lehet statikus memóriaelemek (pl.: flip - flop áramkörök) valamilyen rendszere vagy egy RAM memória része, ami lehet dinamikus vagy statikus. Néhány mikroprocesszor típusnál egyetlen chipben mind a két megoldást alkalmazzák úgy, hogy pl. az akkumulátort statikus memóriaelemekből állítják elő, míg az összes többi regisztert egy általános, dinamikus RAM-ba lehet kombinálni.
Tipikus regiszterek:
Akkumulátor
Az akkumulátor az aritmetikai és logikai műveletek operandusait, vagyis a műveletek tárgyát képező mennyiségeket vagy azoknak az eredményeit tárolja. A korszerű számítógépekben az akkumulátor helyett már egy vagy több regisztertömb van, amelyben akár 512 regiszter is elhelyezkedhet. Így csökkenthető a tárhozfordulások száma, illetve növelhető a végrehajtás sebessége. Adatszámláló regiszter
Az adatok kiolvasásakor vagy beírásakor azonosított memóriarekesz címét tárolja. Mérete függ a

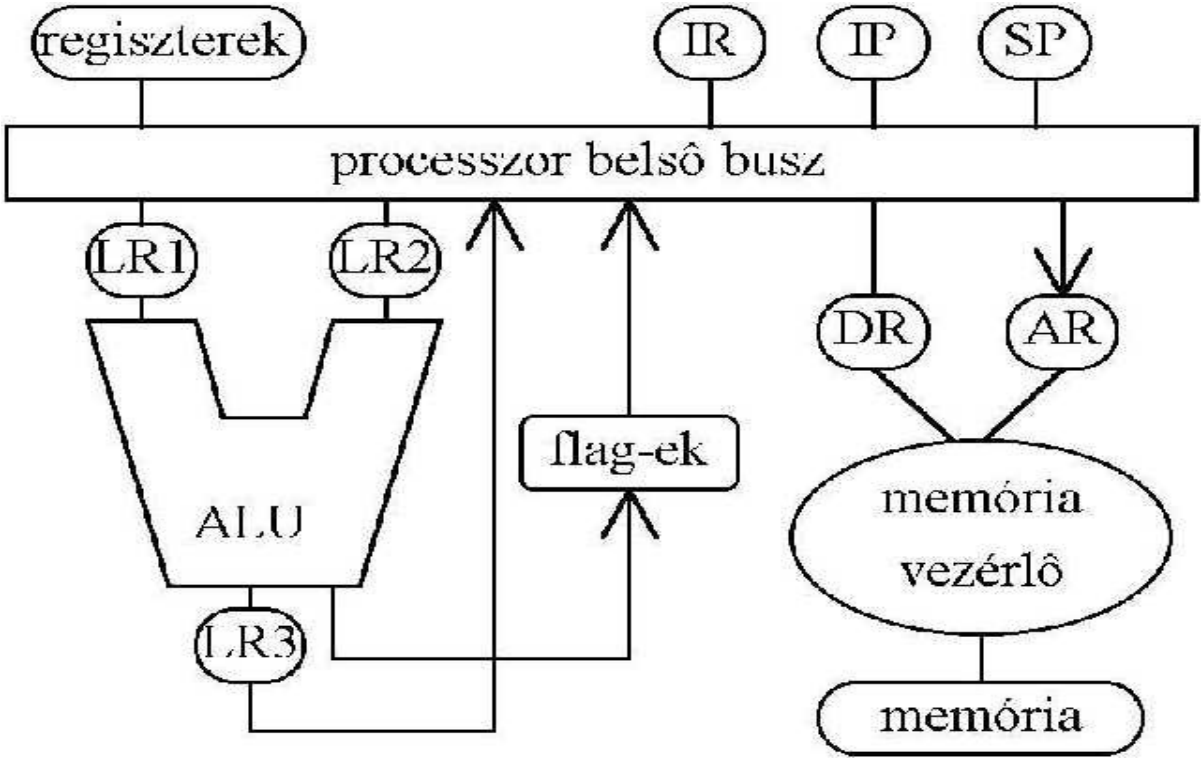
mikroprocesszor által címezhető memóriakapacitástól. Egyszerre több is lehet belőle a CPU-ban.
Utasításregiszter: A program szerint soron következő utasítás műveleti kód részét tárolja.
Utasításszámláló regiszter
A soron következő utasítás címét tárolja. Az utasításszámláló tartalmát a program maga is változtathatja. Fontos különbség az utasításszámláló és az adatszámláló regiszter között, hogy az utasításszámláló regiszternél a problémamegoldás az utasításkódok címeinek sorrendjében megy végbe. Vagyis az utasításszámláló által címzett első memóriarekesz elérésekor kihozunk a memóriából egy utasításkódot, így az utasításszámláló tartalma eggyel nő és így a memória következő rekeszét címezi, ahol a program szerint a következő utasításkód található. Az adatszámláló regiszter csak akkor fut végig az adatcímeken, ha az adatok sokszavas egységben vagy táblázatban vannak tárolva.
Bázis(cím)regiszter
Az operandusok címzéséhez felhasznált regiszter, amely nem általános használatú. A báziscím egy alapcím, amelyhez viszonyítva adhatjuk meg az utasításban az operandus helyét.
Indexregiszterek
Szintén nem minden processzorban találhatók és ezek is az operandusok címzését segítik elő, különösen adatsorok feldolgozásánál.
Állapotregiszterek, vezérlő regiszterek
Egy vagy több regiszteren belül tárolnak vezérlő és ellenörző jeleket.Az állapotregiszter az ALU műveleti eredményeit jellemzi.
Jellegzetes állapotbitek:

- előjelbit
- nulla bit
- túlcsordulásbit
- átvitelbit
- félbyte - átvitelbit
- megszakításbit
- paritásbit stb.

Veremmutató regiszter (SP) A verem legfelső elemét jelöli ki.

Működése

Az egyes részek szükségességének megértéséhez vizsgáljuk meg a processzor utasításciklusát (*instruction cycle*), azaz egy művelet végrehajtásának fázisait!



1. Az utasítás beolvasása a processzorba

Ahhoz, hogy ezt képes legyen a processzor megtenni, szüksége van arra, hogy hol találja meg ezt az utasítást a memóriában. Ez a memóriacím található az IP (*instruction pointer - utasítás mutató*) regiszterben. Ez még kevés, ugyanis ha megtalálta az utasítást és kiolvasta a memóriából, akkor a kiolvasott utasítást valahol el kell tárolni. Erre való az IR (instruction - utasítás) regiszter. A pontos működés megértéséhez még annyit kell tudnunk, hogy a memória (az ábrán jól láthatóan, ha szembe ülsz a monitorral) mindössze két regiszterhez kapcsolódik, azaz az AR és a DR regiszterhez. Az AR (address - cím) regiszter tartalmazza azt a címet, amelyről olvasni szándékozunk illetve ahova írni szeretnénk. A DR (data - adat) regiszter pedig az olvasás után tartalmazni fogja a memória AR című helyén levő adatot, míg az írás során a DR-ben levő adatot az AR címre fogja írni a memóriát vezérlő egység.

Tehát az utasítás beolvasása valójában a következő műveletek végrehajtását jelenti: IP --> AR ; READ ; DR-->IR

2. A beolvasott utasítás dekódolása, elemzése

Miután beolvastuk az utasítást, nekiláthatunk az értelmezésének. Az utasítások mindig két részből állnak: az *utasítás kódjából*, amely meghatározza, hogy milyen műveletet kell végrehajtani, illetve az *operandusokból*, amelyek meghatározzák, hogy milyen adatokkal kell elvégezni az adott műveletet, illetve azt, hogy az eredmény hol tárolódjon.
A beolvasott utasítás (IR) nem biztos, hogy a teljes utasítást tartalmazza, az viszont biztos, hogy az utasítás kódját igen. Azért nem tartalmazhatja minden esetben a szükséges adatokat (illetve azok címeit), mert az olvasáskor még nem tudjuk azt, hogy milyen műveletet olvasunk, így arról

sincs információnk, hogy az adott műveletnek hány és milyen operandusa van. Utóbbi tulajdonságok viszont benne vannak az utasítás kódjában. (Nyilván egy összeadás utasítás több operandust tartalmaz, mint egy olyan utasítás, amely eggyel megnöveli egy regiszter értékét.) Azaz megtörténik az utasításkód értelmezése, melynek alapján az *ALU (arithmetical and logical unit - aritmetikai és logikai műveletvégző egység)* tudomást szerez arról, hogy milyen műveletet kell majd elvégeznie és kiderül, hogy még milyen egyéb részek tartoznak az utasításhoz, azaz mennyi adatot kell még beolvasni a memóriából ahhoz, hogy komplett legyen az utasítás és meghatározhatók legyenek az operandusok.

3. Az operandusok beolvasása

Ebben a fázisban kiolvasásra kerülnek a memóriából az operandusok címei (ha ez szükséges) illetve maguk az operandusok. Ezek kétfélék lehetnek, vagy a memóriában vannak (és akkor onnan ki kell olvasni őket) vagy regiszterek és akkor már a processzorban tartózkodnak. Az ALU két segédregisztere szolgál arra, hogy a kiolvasott operandusokat tárolja (*LR1 és LR2, latch - segédregiszter*). Általában az ALU maximum kétoperandusú műveleteket képes végrehajtani, természetesen, ha speciális műveletvégző egységgel van dolgunk, akkor a maximális operandusszámnak megfelelő segédregiszterre van szükségünk.

4. A művelet végrehajtása

Miután összeállt, hogy mit kell csinálni és az, hogy milyen adatokkal, az ALU elvégezheti az utasítást. Az eredményt - a tárolás céljára - egy harmadik segédregiszterbe teszi (LR3).

5. Az eredmény tárolása

Az utasítás tulajdonságának függvényében az eredmény az LR3 regiszterből vagy valamelyik regiszterbe vagy az utasításban meghatározott memóriacímre kerül (persze két lépésben, először a DR regiszterbe és csak utána a memóriába).

6. A következő utasítás címének meghatározása

Miután az utasítást elvégeztük, meg kell határozni az a címet, ahol a programvégrehajtás folytatódni fog. Ha a programunk szekvenciális, azaz a memóriában a következő címen levő utasítást kell végrehajtani, akkor az IP-t annyival kell megnövelni, amilyen hosszú az adott utasítás volt. Ha valahol egészen máshol kell a programvégrehajtást folytatni, az általában olyan utasítás, amelynek végrehajtása után valamely regiszter tartalmazza a folytatás memóriacímét. Így ezt a címet kell az IP-be beírni.
Az utasításciklus végetért és a processzor rátérhet a következő utasítás végrehajtására, azaz visszatérhet az 1. pontra.
Az ábrán (feljebb egy kicsit) már csak három homályos pont van: a *belső busz*, a *flag-ek*, illetve az *SP regiszter*. A *belső busz* szolgál arra, hogy a processzoron belül adatok áramoljanak rajta az egyik helyről a másikra. A *flag-ek (jelzők)* olyan speciális, 1 bites regiszterek, amelyek a processzor működését befolyásolják illetve állapotát mutatják. Nevezetes flag például a *carry flag*, amely azt mutatja, hogy az utolsó művelet (mondjuk összeadás vagy kivonás) alkalmával keletkezett-e átvitel/áthozat, vagy a *zero-flag*, amely akkor igaz, ha az utoljára végrehajtott

művelet eredménye nulla volt.
Az *SP (stack pointer - verem mutató) regiszter* működésére a későbbiek során még visszatérünk, az eljáráshívások és az eljárásokból való visszatérések során játszik jelentős szerepet.

Az fenti ciklust szokás két, ún. *félciklusra bontani*, ezek a *beolvasás (fetch - 1,2,3) illetve a végrehajtás (execute - 4,5,6) félciklus*.

A processzor utasításkészlete

Ahhoz, hogy teljesen megértsük a processzor működését, szükségünk van arra, hogy tudjuk miféle utasítások végrehajtását várhatjuk el tőle. Azonban az utasítások nem önmagukban álló teendők, hanem műveletek a hozzájuk tartozó adatokkal, így szükségünk van arra, hogy megvizsgáljuk, milyen módon határozhatjuk meg ezeket az adatokat! Az adatok meghatározásának módjait nevezzük *címzési módoknak (addressing modes)*.

Címzési módok

Közvetlen / direkt adat (immediate addressing)
: Olyan adat, amelynek értéke vesz részt a műveletben

Regiszter (register addressing)
: A sorszámával vagy nevével meghatározott regiszter értéke vesz részt a műveletben

Direkt cím (direct addressing)
: A megadott érték egy memóriacímet határoz meg és az itt található adat vesz részt a műveletben

Regiszter indirekt cím (register indirect addressing)
: A sorszámával vagy nevével megadott regiszter értéke meghatároz egy memóriacímet és az itt található adat vesz részt a műveletben

Indirekt memóriacím (indirect addressing)
: A megadott érték egy memóriacímet határoz meg, az itt található adat határozza meg azt a címet, amelyen levő adat résztvesz a műveletben

Indexelt cím (indexed addressing) - Bázisrelatív cím (base relative addressing)
: A cím két komponensből áll: egy *kezdőcímből (base)* és egy *eltolásból* (offset, index). A művelet során a két cím összege határozza meg azt a helyet, ahol az operandus található a memóriában. Attól függően szokás *indexelt illetve báziscímzésről* beszélni, hogy a kezdőcím egy direkt cím avagy egy regiszter tartalma.

Relatív cím (relative addressing)
: Hasonló az indexelt címzéshez, a címzésben szereplő értéket hozzá kell adni (előjelesen) valamilyen címhez, többnyire az utasítás-mutató értékéhez. Ez a címzés a feltételes vezérlésátadó utasításokra jellemző.

A címzési módok (bocs, de ilyen sok van)után nézzük meg a processzor utasításainak csoportjait!

Adatmozgató utasítások
: Ebbe a csoportba három utasítás tartozik: a két regiszter közti (RR), a regiszter és memória közti (RM) illetve a két memóriaterület közti (MM) adatmozgatás utasítása. Ezek közül az utolsót nem minden processzor tudja (hiszen az RM segítségével

előállítható M-->R-->M). Az utasításkészlet többnyire megengedi a memória tetszőleges címzési móddal történő megcímzését.

Aritmetikai utasítások

Ebbe a csoportba tartoznak a matematikai műveleteknek megfelelő utasítások, mint az *összeadás (ADD), kivonás(SUB), szorzás(MUL), osztás(DIV), komplemensképzés(NEG, COMPL)*. A szorzás de még inkább az osztás nem szükségszerűen található meg minden processzor utasításkészletében, hiszen ezek a műveletek visszavezethetők összeadásra és kivonásra. Ide sorolhatók az eggyel növelő (INC) illetve a csökkentő (DEC) utasítások is. Bár ezekre nincs szükség, ha van összeadás (és az mindig van!), mégis szinte minden processzor utasításkészlete tartalmaz ilyen utasításokat az igen gyakori használat és a gyorsabb végrehajtás végett. Az utasítások általában megengedik tetszőleges címzési mód használatát, kivéve a szorzást és osztást, amelyeknél gyakran csak speciális regisztereket használhatunk.

Logikai utasítások

Ebbe a csoportba a *logikai és (AND), logikai vagy (OR), kizáró vagy (XOR), negálás (NOT)* tartozik. Ide sorolhatók a *léptető utasítások*, amelyek az adott operandus bitjeit egy magadott számmal eltolják, léptetik. Ezek a *logikai léptetés balra/jobbra (LSL/LSR), aritmetikai léptetés balra/jobbra (ASL/ASR)*. A különbség a logikai és az aritmetikai jobbra léptetés között annyi, hogy míg logikai esetben a legfelső bitre minden esetben 0 lép be, aritmetikai esetben a belépő bit a szám előjelbitje. A balra léptetésnél nincs különbség a logikai és aritmetikai léptetés között. Az eggyel való léptetés megfelel a kettővel való szorzásnak illetve osztásnak. Az utasítások általában megengedik tetszőleges címzési mód használatát, bár a léptetés mennyiségét többnyire csak direkt adattal vagy egy regiszterrel lehet meghatározni.

Összehasonlító utasítások

Ez a csoport mindössze két utasítást tartalmaz, a *COMP (összehasonlítás)* és a *TEST (bitenkénti összehasonlítás)* utasítást. Egyik sem végez tényleges műveletet, csupán a flag-eket állítja be. A COMP úgy tesz, mintha elvégezne egy kivonást, a TEST pedig, mintha elvégezne egy logikai és műveletet az operandusai között.

Veremkezelő utasítások

A *verem (stack)* egy speciális memóriaterület, amelyet két művelettel tudunk kezelni. Az egyik művelet a *PUSH*, amely egy adatot ráhelyez a verem tetejére, a másik pedig a *POP*, amely leemeli a verem tetején levő adatot. A lényeg az, hogy a veremben nem lehet "kotorászni", csak a teteje látszik. A verem tetejét egy speciális regiszter, az SP (stack pointer) mutatja. A vermet szokás még *LIFO (last in first out*, azaz ami legutoljára ment be az jön ki legelőször) szerkezetnek is hívni.

Processzorvezérlő utasítások

Ide sorolhatók a flag-eket állító utasítások, ezek egy flag törlését (clear) vagy beállítását (set) végzik. Speciális utasítás a *NOP*, amely nem csinál semmit, vagy a *HALT*, amely felfüggeszti a processzor működését. A címzés ezeknél az utasításoknál hiányzik.

I/O utasítások

Ebbe a csoportba két utasítás tartozik: az *IN*, amely beolvasást végez egy adott eszközről illetve az *OUT*, amely kiírást végez egy adott eszköz felé. A címzési mód (melyik eszközről van szó) általában tetszőleges lehet, de az adat többnyire egy kitüntetett regiszterből megy az output eszköz felé illetve ide érkezik input esetén.

Vezérlést módosító utasítások

Ebbe a csoportba azok az utasítások tartoznak, amelyek a szekvenciális programvégrehajtástól való eltérést idézhetnek elő. Lehetnek *feltétel nélküliek* (ilyenkor minden esetben eltérünk a szekvenciális sorrendtől) illetve *feltételesek*. A feltételek nagyon egyszerűek lehetnek csak, többnyire valamelyik flag állására vonatkozhatnak.
Két osztálya van a vezérlést módosító utasításoknak: az *ugrás (JUMP vagy BRANCH)*, amelynek végrehajtása során nem jegyezzük meg azt, hogy honnan érkeztünk illetve a *szubrutinhívás (CALL)*, amely esetben feljegyezzük a veremben, hogy honnan jöttünk. A CALL utasítás párja a *RET (return - visszatérés)*, amelynek hatására a vezérlés visszakerül oda, ahonnan jött. A verem itt azért fontos, hogy a szubrutinok egymásba ágyazhatók legyenek, azaz ha egy szubrutin meghivott egy másikat, az egy harmadikat stb, akkor is vissza tudjon a program találni az elsőhöz. A szubrutinok a programozási nyelvekben megszokott eljárás illetve függvény megvalósításának gépi eszközei.
Egy nagyon speciális vezérlést módosító utasítás a *INT (interrupt - megszakítás)* utasítás. Az interrupt ebben az esetben nem más, mint egy ugrási címeket tartalmazó táblázat valamelyik elemére való ugrás. Abban több, mint a szubrutinhívás, hogy ezt a táblázatot minden program, egymástól függetlenül láthatja, a táblázat helye nem függ a konkrét memóriaelrendezéstől. Ez a táblázat - helyzetéből adódóan - az operációs rendszer mindenki által használható rutinjainak címeit tartalmazza. Így az operációs rendszer rutinjainak csupán a sorszámát kell tudnunk ahhoz, hogy végrehajtsuk őket, nem pedig a rutinok tényleges, memóriabeli címét. Ez a lehetőség nagyban segít abban, hogy egyrészt az operációs rendszerek továbbfejleszthetők legyenek, másrészt abban, hogy bárhova elhelyezhetők a memóriában a megfelelő rutinok, ügyelve persze arra, hogy a táblázat megfelelő eleme mindig az aktuálisan érvényes címet tartalmazza. Az INT utasítás párja az *IRET*, amely a végrehajtott szubrutinból való visszatérést biztosítja.

Egy kitalált, de a valóságot modellezni képes processzor modellje

Azért, hogy a fenti gondolatmenetet még jobban megértsd, képzeljünk el egy processzort, ruházzuk fel regiszterekkel és definiáljunk egy nem túl bonyolult, de mégis használható utasításkészletet!
Tartalmazzon a processzorunk 16 olyan regisztert, amely képes egy 16 bites számot tárolni. A regisztereket sorszámukkal jelöljük, azaz a R0, R1, ...R15. Az aritmetikai és logikai egység legyen olyan, amelyik 16 bites adatokkal tud műveletet végezni, és természetesen az eredmény is 16 bites lesz.
A memóriavezérlő kétfajta adatokkal is tudjon műveletet végezni, azaz legyen mód 16 bites és 8 bites adatátvitelre is. 8 bites adatátvitelkor a DR regiszternek csak a fele játszik szerepet. A memóriacím mindig 16 bites legyen (tehát az AR és az IP regiszternek is ilyennek kell lennie)
Az utasítások kódrésze legyen 8 bites (azaz az IR is 8 bites). Maga az utasításkód nézzen ki a következőképpen:

4 bit	2 bit	2 bit
műveleti kód	1. operandus címzési módja	2. operandus címzési módja

Az elméletileg három operandust igénylő műveleteknél (mint például az összeadás) a harmadik operandus (azaz az eredmény helye) legyen mindig az első operandus, azaz az eredmény minden esetben felülírja a kiindulási adatot. Azoknál a műveleteknél, ahol nincs szükség két operandus megadására, ott a felszabaduló biteket felhasználjuk újabb utasítások definiálására, így 16-nál jóval több utasítást fogunk meghatározni.
Az operandusok legyenek változatosak! Négyféle operandust (címzési módot) engedünk meg: regisztert, direkt adatot (konstans - immediate), regiszterben megadott memóriacímen levő adatot (regiszter indirekt) illetve memóriában szereplő adatot. Azaz össze tudunk adni egy regiszterben levő számot egy konstassal, avagy egy memóriában levő számhoz hozzá tudjuk adni azt a számot, amely azon a helyen van a memóriában, amelyet egy regiszter címez meg.
Természetesen az első operandus sohasem lehet direkt adat, mert igen érdekes jelenségeket eredményezne az, ha egy konstans értékét meg tudnák változtatni.
A 00 címzési mód jelölje a regisztert (R), a 01 a memóriacímet (M), 10 a regiszter indirekt (I) és 11 a direkt (konstans) adatot (D). A lehetséges címzési módokat mutatja a következő táblázat, melynek +1..+4 oszlopai azokat a byte-okat mutatják, amelyek az utasításkódot követik ahhoz, hogy minden operandus jól meghatározott legyen. A 4+4 azt jelzi, hogy az adott byte két 4 bites része 1-1 regisztert határoz meg, a 0+4 egy regisztert határoz meg.

típus	kód	+1	+2	+3	+4
RR	0000	4+4			
RM	0001	0+4	cím		
RI	0010	4+4			
RD	0011	0+4	adat		
MR	0100	cím		0+4	
MM	0101	cím		cím	
MI	0110	cím		0+4	
MD	0111	cím		adat	
IR	1000	4+4			
IM	1001	0+4	cím		
II	1010	4+4			
ID	1011	0+4	adat		
RR	00	0+4			
M	01	cím			
I	10	0+4			
D	11	adat			

A processzorunk utasításkészlete nézzen ki a következőképpen:

kód	*név*	*meghatározás*
0000	MOV	adatmozgatás
0001	ADD	összeadás
0010	SUB	kivonás
0011	CMP	összehasonlítás
0100	AND	logikai és
0101	OR	logikai vagy
0110	XOR	kizáró vagy
011100	NOT	logikai nem
011101	NEG	-1 szeres képzése, komplementálás
011110	INC	1-el növelés
011111	DEC	1-el csökkentés
1000	LSR	logikai léptetés jobbra
1001	LSL	logikai léptetés balra
1010	ASR	aritmetikai léptetés jobbra
101100	PUSH	verem tetejére helyezés
101101	POP	kivétel a veremből
10111000	PUSHF	flag-ek verembe helyezése
10111100	POPF	flag-ek visszatöltése a veremből
110000	JUMP	ugrás a megadott címre
110001	CALL	szubrutinhívás
110010	INT	interrupt hívás
11001100	RET	visszatérés szubrutinból
11001101	IRET	visszatérés interruptból
110100	JC	ugrás, ha a carry flag = 1
110101	JNC	ugrás, ha a carry flag= 0
110110	JZ	ugrás, ha a zero flag = 1
110111	JNZ	ugrás, ha a zero flag = 0
111000	OUT	R0 kivitele valamelyik output eszközre
111001	IN	adatbevitel valamelyik input eszközről R0-ba
11110000	NOP	üres utasítás
11110001	HALT	processzor működésének felfüggesztése
11110010	SETC	carry flag beállítása
11110011	CLC	carry flag törlése

A fenti táblázatból jól látszik, hogy vannak 2, 1 illetve 0 címet igénylő utasítások. Az is látható, hogy az utasításkódoknál nem használtuk ki az összes lehetőséget, például a 11101111 kódhoz nem tartozik utasítás.

Feladat: Próbáljuk meg megkomponálni azt az utasítást, amely a 8. regiszterhez hozzáadja a 128-as memóriacímen levő értéket és az eredményt az első operandus (azaz a 8. regiszter) helyén tárolja!

Az utasításkód összeadás kell, hogy legyen, azaz az első 4 bit: 0001.

Az első operandus egy regiszter, amelynek címzési módja 00, míg a másik egy memóriában levő adat, amelynek címzési módja 01. Így az utasítás kódja: 0001 0001. Az ilyen címzési módokat megkövetelő utasításkód után még három byte következik, amelyből az első a regiszter sorszámát, a másik kettő pedig a memóriacímet tartalmazza, azaz az operandusokat leíró 3 byte: 00001000 (=8) 00000000 10000000 (=128).

Tehát az utasítás teljes kódja:

ADD R8,M128 : 00010001 00001000 00000000 10000000

Feladat: Adjuk meg azt az utasítást, amely a zero flag igaz állása esetén elugrik a 3. regiszter által meghatározott címre és ott folytatja a program végrehajtását!

JZ I3 kódja: 11010 10 (indirekt cím) 00000011 (3. regiszter)
Feladat:
Adjuk meg azt az utasítást, amely végrehajtja 13. interrupt-ot!
INT 13 kódja: 110010 11 (direkt adat) 00000000 00001101 (=13)

Most nézzünk meg egy olyan programhoz tartozó kódot, amely egy valós feladatocskát old meg!

A feladat legyen a következő: adjunk össze néhány számot. Azaz adott a memóriában valahol (mondjuk az 1000-es címtől kezdődően) egy csomó szám. A legelső közülük kitüntetett, megmondja, hogy hány darab szám van utána, amivel foglalkoznunk kell, azaz amelyeket össze kell adnunk. Az alábbi program elvégzi ezt a feladatot (feltéve, hogy legalább egy számot össze kell adni)
A programot úgy helyezzük el a memóriában, hogy a 100-as (hexadecimális) címen kezdődjön.

```
100:
        XOR     R0, R0          ; itt tároljuk az összeget (kinullázás)
        MOV     R1, 1000        ; a memóriában itt kezdődnek az adataink
        MOV     R2, I1          ; ennyi elemet kell összeadni
        INC     R1              ;
        INC     R1              ; most az R1 az első összeadandóra mutat,
                                ; minden szám 2 byte-os!!!!
        CLC                     ; carry flag törlése
UJRA:
        ADD     R0,I1           ; a soron következő elem hozzáadása az összeghez
        INC     R1              ; cím növelése, 2 byte-os számok!
        INC     R1
        DEC     R2              ; az összeadandók számának csökkentése
        JNZ     UJRA            ; ha még van mit összeadni, akkor vissza
VEGE:
        INT     10              ; kiíratás rutinjának meghívása
```

Az UJRA egy szimbolikus címet jelöl (az olvashatóság érdekében), jelen esetben a JNZ után álló konkrét érték -11 lesz, hiszen ennyi byte-ot kell visszalépni ahhoz, hogy az ADD utasításhoz kerüljünk. (2+2+2+2+3 = 11). A visszalépést a szekvenciális végrehajtásnál levő címhez (az INT utasítás címe) viszonyítva kell számolni, elágazás nélkül ugyanis itt folyatódna a program.
A fenti utasítássor gépi kódja a következő (hexadecimálisan):

```
100:    60 00           XOR     R0, R0          ; itt tároljuk az összeget (kinullázás)
102:    03 01 03 E8     MOV     R1, 1000        ; a memóriában itt vannak az adataink
106     02 21           MOV     R2, I1          ; ennyi elemet kell összeadni
108     78 01           INC     R1              ;
10A     78 01           INC     R1              ; R1 az elsô összeadandóra mutat,
10C     F3              CLC                     ; carry flag törlése
10D     42 01           ADD     R0,I1           ; a következô elem hozzáadása
10F     78 01           INC     R1              ; cím növelése, 2 byte-os számok!
111     78 01           INC     R1
113     7A 02           DEC     R2              ; az összeadandók számának csökkentése
115     BF FF F5  JNZ   UJRA                    ; ha még van mit összeadni, akkor vissza
118
```

Szemeljünk ki a fenti kódrészletből egy bonyolult címzési móddal rendelkező utasítást és nézzük meg még egyszer a processzor utasítás-ciklusát erre az egy utasításra! Legyen a kiszemelt utasítás az ADD R0,I1 utasítás.
Ekkor a processzor a következő mikroprogram-lépéseket fogja végrehajtani

```
(IP = 10D)
IP --->  AR
M-READ (42)
DR ---> IR
dekódolás (címzési mód: RI, azaz +1 byte!)
IP +1 ---> AR
M-READ (01)
R0 ---> LR1
R1 ---> AR
M-READ
DR ---> LR2
ALU-ADD!
LR3 ---> R0
IP+2 ---> IP
```

Megjegyzés: A gyakorlatban, valahányszor az IP-t memória-olvasásra használjuk fel, tartalma mindig nő 1-el (így az utolsó lépés elmaradhat).

Forrás: http://x3.hu/freeweb/frameset.x3?user=/zimonyi_tdk&page=/zimonyi/rendtech/mikropro.html