Tartalomjegyzék

"Amit nem lehet Pascal-ban megírni, azt C/C++-ban meg lehet.
Amit nem lehet C/C++-ban megírni, azt Assembly-ben meg lehet.
Amit nem lehet Assembly-ben megírni, azt nem lehet megírni!"

„Az igazi programozó nem fél a GOTO-tól”

II. számú melléklet

Mikrovezérlő programozása assembly nyelven

Bevezetés

Úgy érzem a fenti cím egy kis magyarázatra szorul. Mikor ezt a szakdolgozat témát választottam többen is felvetették, hogy miért assembly nyelven szeretném megírni az egyes periféria lekezeléseket, s az egész projekt miért assemblyben van? Sokat gondolkoztam rajta, hogy talán tényleg egyszerűbb lenne egy magas szintű nyelv segítségével kezelni a perifériákat. Szakmámból adódóan azonban elsősorban a hardver felől tudok megközelíteni egy adott programozási problémát – a leg-hardverközelibb nyelv pedig minden kétséget kizáróan az assembly (teljesen egyenértékű a gépi kóddal, de kicsit emészthetőbb forma). Természetesen ez még nem lenne elegendő ok, hiszen a szakdolgozat elsősorban nem az én örömömre készül – sokkal inkább más kollégáknak segítség, útmutatás, az érdeklődés felkeltése gyanánt. Véleményem szerint egy mikroprocesszoros rendszert a leghatékonyabban assembly nyelven tudunk programozni – és kijelenthetjük, hogy ezen nyelv segítségével tudjuk nem csak a hardver, de saját képességeinket is a legteljesebb mértékben kihasználni.

Vizsgáljuk meg a mikrovezérlő felépítését, architektúráját, hogy az előbbi megállapítás bizonyítást nyerjen. A PIC mikrovezérlő egy egycímes, harvard architektúrára épülő, vezérlésre optimalizált RISC processzort tartalmazó mikroszámítógép.

Egycímes számítógép

Ahhoz, hogy egy számítógép műveleteket hajtson egymás után végre (program fusson), minden utasításnak négy címet kell tartalmaznia – ezek a következők:

- Első operandus címe
- Második operandus (amennyiben van) címe
- Eredmény címe
- Következő utasítás címe

Műveleti kód OP. COD	I. Operandus címe	II. Operandus címe	Eredmény címe	Következő utasítás címe

Négy címes architektúrát sehol a világon nem alkalmaztak, és nem is alkalmaznak. Gondoljunk bele, minden utasításhoz négy címet le kellene tárolni, túl hosszú lenne egy utasítás – arról nem is beszélve, hogy mindez komolyan hátráltatná a programozó munkáját is.

Próbáljuk redukálni a szükséges címeket. Első lépésként használjuk ki a következő Neumann alapelvet: „A számítógép legyen soros utasítás-végrehajtású”. Amennyiben az utasításokat egymás után sorban hajtjuk végre, akkor nincs más teendőnk, mint a memóriába is ugyanebbe a sorrendbe beírni őket. A következő utasítás címéhez felhasználunk egy egyszerű számlálót, mely reset esetén nullázódik, és minden utasítás végrehajtás során eggyel növekszik. Ennek a számlálónak a neve *PC* (Program Counter - Program számláló), vagy *IP* (Instruction Pointer – Utasítás mutató). PIC mikrovezérlőknél következetesen ***PC***-nek nevezzük. Ezen kitüntetett számláló segítségével megalkottuk a háromcímes architektúrát.

Műveleti kód OP. COD	I. Operandus címe	II. Operandus címe	Eredmény címe

Műveleteink jelentős hányadánál az elvégzés után nincs szükség mindkét operandusra, ezért az eredményt írjuk vissza a második operandus címére. Ezzel eljutottunk a kétcímes architektúrához – azonban szükséges egy új utasítást alkotnunk: *MOVE* (mozgató) utasítás, mely segítségével a második operandusunkat is le tudjuk kérdezni a művelet elvégzése után.

Műveleti kód OP. COD	I. Operandus címe	II. Operandus és Eredmény címe

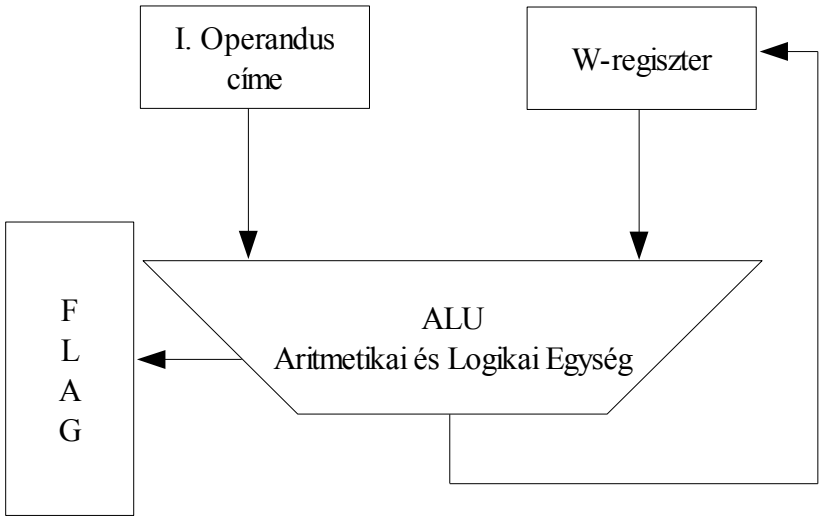
A PIC mikrovezérlők – a mai számítógépek többségével egyetemben az egycímes struktúrát alkalmazzák, ahhoz hogy ezt létre tudjuk hozni szükség van egy átmeneti regiszterre, ahol az egyik operandust letároljuk a művelet elvégzése előtt. Ez egy kitüntetett regiszter - a szakirodalom akkumulátor (*ACC-accumulator*), vagy munkaregiszter (*W-Work*) néven ismeri. Mi a *W* jelölést fogjuk használni.

Műveleti kód OP. COD	I. Operandus címe

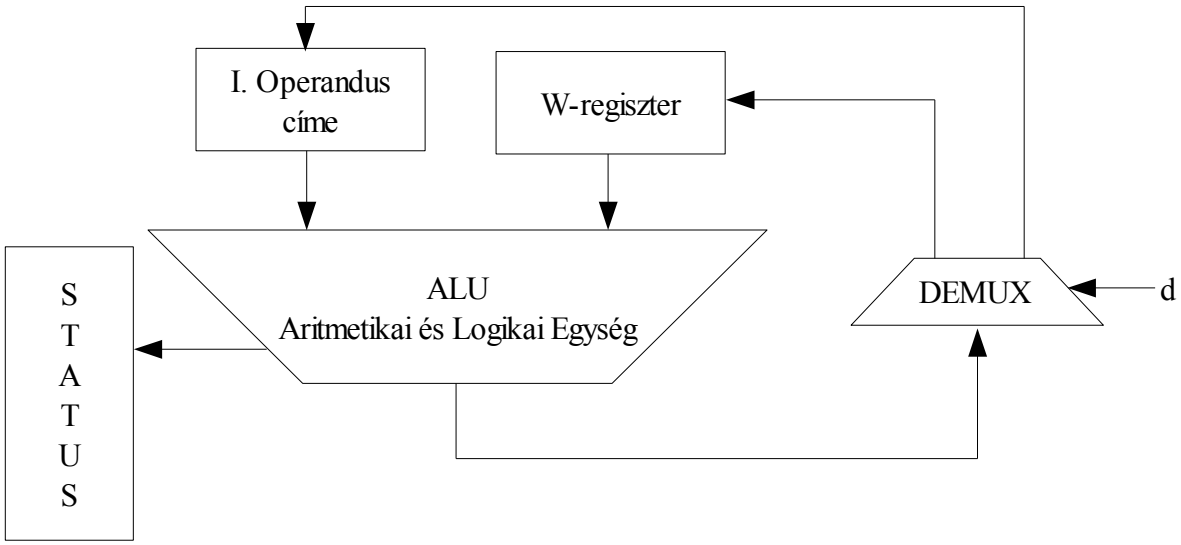
Az itt ismertetett architektúrán minden kétoperandusú művelet csakis az akkumulátoron keresztül valósulhat meg. Először az első operandust lehívjuk a *W* regiszterbe (*LOAD*), majd elvégezzük a műveletet (*OPERATION*), és kiírjuk az eredmény (*STORE*)

Bevezetés

Mikroszámítógépek művelet-végrehajtásának folyamata:



PIC mikrovezérlők művelet-végrehajtása:



PIC mikrovezérlő esetén a Flag-regisztert státusz (*STATUS*) regiszternek nevezzük, valamint az eredményt nem csak a *W* regiszterbe, hanem az első operandust tartalmazó file regiszterbe is visszaírhatjuk, erre használjuk a destination (cél) bitet. Az eddig leírtakat figyelembe véve egy összeadás a következőképpen néz ki a fenti architektúrán:

```
MOVLW   20          ;'20'h betöltése az akkumulátorba (1. operandus)
ADDWF   PORTA,0     ;PORTA-hoz 20h hozzáadása, eredmény: akkumulátor
```

Az MPLAB fordítója lehetőséget nyújt a *W* és *F* szimbólumok használatára is ilyenkor d=W esetén a munkaregiszterbe, d=F esetén a fájlregiszterbe kerül az eredmény. Ahhoz, hogy ez a programozástechnikai eljárás működjön a gyári include fájlt hozzá kell adnunk a forráshoz.

Fontos megjegyezni, hogy amennyiben a cél egyértelműen megállapítható a műveleti kódból (pl. konstanssal végzett művelet) a destination bitnek nincs értelme, a fordító a használatát hibának fogja értelmezni.

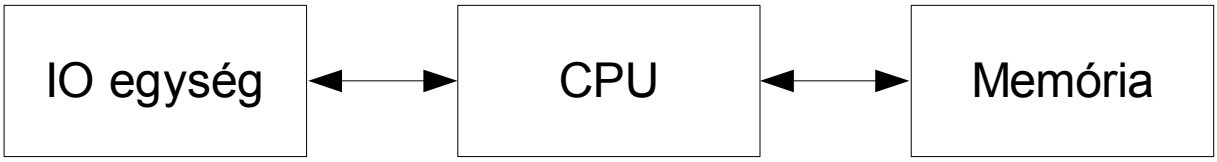
Harvard architektúra

Tekintsük át először a Neumann elveket[1] (a legtöbb mai számítógép is többé kevésbé alkalmazza ezeket az elveket, bár a felhasználástól függően esetleg nem mindegyiket):

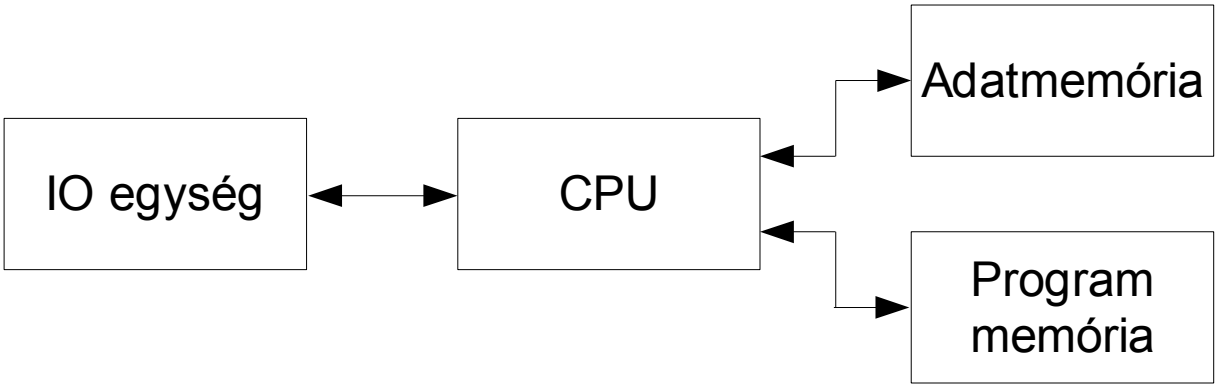
- A számítógép legyen bináris (használja a kettes számrendszert)
- A számítógép legyen teljesen elektronikus
- Az utasítás-végrehajtás legyen soros (egyszerre csak egy utasítást hajtson végre)
- Legyen univerzális Turing-gép (bármely aritmetikai és logikai műveletet végre tudjon hajtani az alapműveletek véges számú ismétlésével)
- A tárolt program elve - vagyis rendelkezzen belső tárral, az utasítások és az adatok a memóriában egymástól megkülönböztethetetlen módon legyenek tárolva.

A Harvard architektúra az itt felsorolt elvek közül az utolsót írja át, mikor kimondja: Az adat és a programmemória hardveresen különüljön el. Ezáltal különböző hosszúságú lehet az utasítás és az adat. Példaképpen a *16F84*-es mikrovezérlőnél 8 bites adatok mellett 14 bites utasításokkal tudunk dolgozni – így el tudjuk érni, hogy minden utasításunk egy szavas legyen. Fontos azonban megemlíteni, hogy a Harvard architektúra létjogosultsága igencsak behatárolt. Használata csak célszámítógépek, speciális alkalmazásra gyártott mikrogépek esetén lehetséges. Egy mikrovezérlőt nevéből adódóan vezérlési feladatokra fejlesztettek ki. Statisztikailag bizonyítható, hogy vezérlési feladatok esetén kevés átmeneti adattal, és sok vezérlő utasítással kell dolgoznunk, így előre meg tudjuk tervezni az adat- és programmemória relatív viszonyát.

Neumann elvű számítógép felépítése:



Harvard architektúrás gép felépítése:



1 Az itt felsorolt elvek csak a működésre vonatkoznak. Neumann János megadta a megfelelő hardverfelépítést is (ALU, CU, MEM) – melyet mind a mai napig alkalmaznak a digitális számítógép esetében.

RISC *jelleg*

A *RISC* mozaikszó jelentése: *Reduced Instruction Set Computer* – Csökkentett utasításkészletű számítógép.

Jellemzői:

- Egyszerű, kevés számú utasítás
- Minden utasítás (lehetőleg) egy gépi ciklus alatt hajtódjon végre
- Egyszerűsített címzési mód
- Minden utasítás egyszavas
- Nagyszámú általános célú regiszter

Egy *RISC* processzor nem képes bonyolult (több műveletből álló) utasításokat végrehajtani, azonban az utasításkészlete minden művelet elvégzését lehetővé teszi – mégpedig sebességre optimalizálva.

A RISC és CISC processzorok közötti verseny a mai napig tart. A mai korszerű CISC mikroprocesszorok már RISC magra épülnek így gyorsítva a működést.

Összefoglalva: ahhoz, hogy kihasználjuk az architektúra előnyeit a gép nyelvén kell programoznunk. Erre alkalmazhatjuk a bináris, a hexa (gépi) kódolást, ill. az ezek előnyeit megtartó, de az ember számára emészthetőbb mnemonikus kódolást - vagyis az assembly programnyelvet. Feladatra orientált nyelvek használata (*C*; *BASIC*; stb.) a kódunkat nagyon redundánssá teszi, ezáltal vannak feladatok, amiket nem, vagy csak nehézkesen tudunk megoldani (pl. sebességre érzékeny műveletek).

Az assembly nyelv megjelenése, létjogosultsága

A „hőskorban” a számítógépek processzorait bináris kódban, később a valamivel könnyebben megjegyezhető hexadecimális kódban írták. Hamar kiderült ezen programozás korlátozottsága. Az emberi agy nehezen jegyez meg számokat, még nehezebben kapcsolja össze őket különböző fogalmakkal – ahhoz, hogy egy adott processzor gépi kódjában programozói jártasságra tegyünk szert, igen hosszú idő és sok gyakorlás szükségeltetik. Meg kell még említeni, hogy ha áttérünk egy másik processzorra, annak gépi kódja számunkra ismeretlen lesz, kezdhetjük a tanulást elölről. Gondot okoz még a címszámítás, ill. a későbbi programmódosítás is. Aki programozott már processzort gépi kódban, tisztában van az egy sor utasítás, öt sor *NOP* (*No OPeration* – nincs művelet) szabállyal – erre azért van szükség, hogy ha később be kényszerülünk szúrni egy, vagy két utasítást, ne kelljen az összes ugrást a programban újra kiszámítani. Ezen áldatlan állapoton próbált meg segíteni az assembly nyelv. Ez a programnyelv a későbbi feladatorientált, ill. objektumorientált nyelvekkel szemben nem vezetett be új (ún. virtuális) utasításokat (absztrakciót), csupán a meglévőekhez rendelt egy rövid „emlékeztetőt” ún. mnemonikot. Természetesen egyetlen processzor sem ismeri az assembly nyelvet, szükségünk van egy fordítóra, ami az általunk használt mnemonikokat visszaalakítja gépi kódra. Az átalakítás nem redundáns - vagyis az assemblyben megírt program ugyanolyan hatékony lesz, mintha gépi kódban programoztunk volna. A *RISC* jellegű processzor miatt a *PIC* mikrovezérlők assembly nyelve egy kicsit eltér a sokak által ismert *Intel* filozófiától.

Az assembly nyelv felépítése

Az assembly nyelv alapegysége a sor. Egy sorban egy és csak egy utasítás helyezkedhet el.

Utasítás: Az adott mikroprocesszor gépi kódját reprezentáló szimbólumkészlet, kiegészítve néhány – fordító függő - kényelmi szolgáltatással.

Címke	Mnemonic	Operandos(ok)	Megjegyzés

Címke: Címhivatkozási szimbólum. A gépi kóddal ellentétben az assembly nyelv megszabadít minket a fáradalmas címszámításoktól. Az elágazások során alkalmazott belépési pontokat saját szimbólumokkal láthatjuk el. A címke szabályosan ':'-tal záródik – az *MPASM* elfogadja nélküle is. Amennyiben 8 karakternél hosszabb címkét szeretnénk alkalmazni használjuk az '_' karaktert elválasztásra. Lehetőség szerint használjunk „beszédes” címkéket. Címke önállóan is állhat egy sorban – ekkor a következő assembly sort jelképezi (figyelem, ez lehet direktíva is).

Mnemonik: A műveleti kódot jelképező emlékeztető.

Operandusok: A művelet elvégzéséhez szükséges konstans, program- vagy adatmemória cím. A konstanst megadhatjuk a fordító által preferált számformátumokban (bináris, oktális, decimális, hexadecimális), vagy *ASCII* kódban aposztrófok között (pl. 'A'). A címkékkel hivatkozhatunk a programmemória címére. Direktívák segítségével szimbólumokat rendelhetünk az adatmemória címeihez is. Természetesen mindkét memóriacímet megadhatjuk a fent említett számformátumokban is. Az operandusok megadása során szükségünk lehet egyszerű számítási műveletekre – ahhoz, hogy ezeket ne nekünk kelljen elvégezni használhatunk operátorokat. Fontos azonban, hogy ezek a fordítónak szólnak. Tipikus hiba, hogy egy memóriarekeszben lévő értékhez hozzá szeretnénk adni egy számot, és ezt operátorok segítségével akarjuk megoldani. Az elgondolás hibás! Az *összeadás operátor* használatával ezt nem tudjuk végrehajtani csak az **összeadás utasítással**.

```
MOVF    TRISB+5,W   ;az utasítás hatására nem a TRISB 5-tel
                    ;megnövelt értéke kerül az
                    ;akkumulátorba, hanem a 11-es
                    ;memóriarekeszben lévő INTCON
                    ;regiszter értéke (TRISB címe: 6; a
                    ;radix: 16). PIC16F84-es típus esetén.
```

Operátorok:

- + összeadás
- - kivonás (vagy kettes komplemens)
- * szorzás
- / osztás
- % maradékos osztás, az eredmény a maradék
- (bal zárójel műveletek csoportosítására
-) jobb zárójel műveletek csoportosítására
- << balra léptetés a biteken
- >> jobbra léptetés a biteken
- ! logikai komplemens
- ^ bitenkénti KIZÁRÓ-VAGY kapcsolat
- | bitenkénti VAGY kapcsolat
- & bitenkénti ÉS kapcsolat
- && logikai ÉS kapcsolat
- || logikai VAGY kapcsolat
- > nagyobb?
- < kisebb?
- >= nagyobb, vagy egyenlő?
- <= kisebb, vagy egyenlő?
- == egyenlő?
- != nem egyenlő?
- ~ aritmetikai komplemens
- $ aktuális címmutató (PC)
- LOW alsó bájt (0:7)
- HIGH felső bájt (8:15)
- UPPER felső bájt (16:21)
- stb.

Megjegyzés: A programozó által az utasításokhoz írt komment. Az assembly nyelven írt programban nagyon fontos a megjegyzés – kódunk igen hosszúra nyúlhat, ráadásul első ránézésre elég nehéz eligazodni rajta. A programdokumentáció szerves része az egyes utasítások, modulok, források megfelelő eligazodási segítséggel való ellátása. A megjegyzés egy különleges karakterrel a ';'-tal kezdődik. Minden ami ez után van kommentnek számít, és a fordító már nem veszi figyelembe. Teljes sor(ok) is lehetnek megjegyzés(ek). Célszerű úgy elkészíteni ezt az előzetes dokumentációt, mintha egy másik szakembernek készítenénk útmutatót. A saját magunknak írt megjegyzések gyakran nagyon felületesek, és egy-két hónap elteltével már magunk sem igazodunk ki rajtuk. A magyarázatok ne a nyilvánvaló tényekre hívják fel a figyelmet (pl. az akkumulátorba 15h-t töltünk) – inkább a rutin, modul, program lényegét próbáljuk megfogni, és szemantikai megjegyzéseket tegyünk.

Direktíva: A fordítónak szóló speciális utasítás a programozótól. A direktívák nem kerülnek lefordításra – bár hatásaik megjelennek a tárgyprogramban.

Direktíva szimbóluma	Módosító rész	Módosító rész

Gyakori hiba programozás során a direktívával egy sorba címkét tenni – ez a legtöbb direktíva esetén tilos! A direktíva nem fordul be a kódba, ezért nincs is értelme az ő címére hivatkozó szimbólumnak – a fordító hibát fog jelezni. Amennyiben direktívát szeretnénk megjelölni belépési pontként a címkét írjuk egy sorral a direktíva fölé egy önálló sorba (valójában nem a direktíva fog erre a címkére való ugrás során végrehajtódni – hiszen ez már csak működés közben történik meg - csak az általa jelképezett hatások).

Megjegyzés: A számok számmal és csakis számmal, míg a szimbólumok betűvel és csakis betűvel kezdődhetnek egy assembly programban.

A következőkben megismerkedünk a *PIC* mikrovezérlők utasításaival és az *MPASM* direktíváival – majd röviden összefoglaljuk a fejlesztés menetét.

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik	Jelentés	Funkció	Gépi ciklus	Állított jelző bitek
Adatmozgató utasítások				
MOVLW k	MOVe Literal to W	konstans mozgatása W-be	1	
MOVWF f	MOVe W to File	W mozgatása fájlregiszterbe	1	
MOVF f,d	MOVe File	Fájlregiszter mozgatása	1	Z
Aritmetikai utasítások				
ADDLW k	ADD Literal and W	Konstans hozzáadása W-hez	1	C,DC,Z
ADDWF f	ADD W and File	W hozzáadása fájlregiszterhez	1	C,DC,Z
SUBLW k	SUBtract W from Literal	W kivonása konstansból	1	C,DC,Z
SUBWF f,d	SUBtract W from File	W kivonása fájlregiszterből	1	C,DC,Z
INCF f,d	INCrement File	Fájlregiszter növelése	1	Z
DECF f,d	DECrement File	Fájlregiszter csökkentése	1	Z
Logikai utasítások				
ANDLW k	AND Literal with W	Konstans és W ÉS kapcsolata	1	Z
ANDWF f,d	AND W with File	W és fájlregiszter ÉS kapcsolata	1	Z
IORLW k	Inclusive OR Literal with W	Konstans és W VAGY kapcsolata	1	Z
IORWF f,d	Inclusive OR W with File	W és fájlregiszter VAGY kapcsolata	1	Z
XORLW k	eXclusive OR Literal with W	Konstans és W kizáró-vagy kapcsolata	1	Z
XORWF f,d	eXclusive OR W with File	W és regiszter kizáró-vagy kapcsolata	1	Z
COMF f,d	COMplement File	Fájlregiszter bitjeinek invertálása	1	Z
SWAPF f,d	SWAP nibbles in File	Regiszter alsó-felső 4 bitjének cseréje	1	-
RLF f,d	Rotate Left File through carry	Forgatás balra az átvitelbiten keresztül	1	C
RRF f,d	Rotate Left File through carry	Forgatás jobbra az átvitelbiten keresztül	1	C
BCF f,b	Bit Clear File	Fájlregiszter bitjének törlése	1	-
BSF f,b	Bit Set File	Fájlregiszter bitjének egybe állítása	1	-
CLRF f	CLeaR File	Fájlregiszter törlése	1	Z
CLRW -	CLeaR W	W törlése	1	Z
CLRWDT -	CLeaR Watch Dog Timer	WDT törlése	1	$\overline{TO}$, $\overline{PD}$
Vezérlésátadó utasítások				
GOTO k	GO TO address	Feltétel nélküli ugrás	2	-
BTFSC f,b	Bit Test File, Skip if Clear	Bit tesztelése és ugrás, ha 0	1,2	-
BTFSS f,b	Bit Test File, Skip if Set	Bit tesztelése és ugrás, ha 1	1,2	-
INCFSZ f	INCrement File, Skip if Zero	Fájlregiszter növelése és ugrás, ha 0	1,2	
DECFSZ f	DECrement File, Skip if Zero	Fájlregiszter csökkentése és ugrás, ha 0	1,2	
CALL k	CALL Subrutine	Szubrutin meghívása	2	-
RETURN -	RETURN from subrutine	Visszatérés szubrutinból	2	-
RETLW k	RETurn from subrutine with Literal in W	Visszatérés szubrutinból konstans w-ben	2	-
RETFIE -	RETurn From Interrupt	Visszatérés megszakításból	2	-
SLEEP -	SLEEP (Go into standby mode)	Alvó mód	1	$\overline{TO}$, $\overline{PD}$
NOP	No OPeration	Nincs művelet	1	-

PIC mikrovezérlő utasításkészlete (16-os sorozat)

A táblázatban szereplő jelölések:

- k: konstans (literál)
- b: bitcím egy 8 bites regiszterben
- f: fájlregiszter
- d: destination (cél)
- -: nincs operandus

- Z: Zero bit
- C: Carry (átvitel) bit
- DC: Digit Carry bit
- $\overline{TO}$: Time Out bit
- $\overline{PD}$: Power Down bit

Az utasítások részletes leírása

Mnemonik: **MOVLW**

Szintaxis:	[címke] MOVLW k
Operandus:	$0 \leq k \leq 255$
Szimbolikus jelölés:	$k \rightarrow (W)$
Állított állapotbitek:	Nincs
14 bites kód:	11 00xx kkkk kkkk
Megjegyzés:	8 bites konstans értéket tudunk betölteni a munkaregiszterbe. A nem meghatározott bitek értéke: 0.
Szó:	1
Ciklus:	1
Példa:	MOVLW .6 ;az akkumulátor 6-tal való feltöltése

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **MOVWF**

Szintaxis:	[címke]	MOVWF	f
Operandus:	$0 \leq f \leq 127$		
Szimbolikus jelölés:	(W) → (f)		
Állított állapotbitek:	Nincs		
14 bites kód:	00 0000 1fff ffff		
Megjegyzés:	Adat betöltése a munkaregiszterből egy fájlregiszterbe (az adatmemória meghatározott memóriarekeszébe).		
Szó:	1		
Ciklus:	1		
Példa:	MOVWF	PORTB	;akkumulátor értéke a portb-be

Mnemonik: **MOVF**

Szintaxis:	[címke]	MOVF f,d	
Operandus:	$0 \leq f \leq 127$ $d \in [0,1]$		
Szimbolikus jelölés:	(f) → (cél)		
Állított állapotbitek:	Z		
14 bites kód:	00 1000 dfff fff		
Megjegyzés:	Regiszterben tárolt adat mozgatása a célbit függvényében. D=0 esetén az adat a munkaregiszterbe kerül, d=1 esetén az adat a fájlregiszterbe kerül. Az önmagába mozgatás után tesztelhetjük a zéró bitet - így el tudjuk dönteni egy számról, hogy nulla-e.		
Szó:	1		
Ciklus:	1		
Példa:	MOVF	PORTA,W	;porta értéke az akkumulátorba

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **ADDLW**

Szintaxis:	[címke] ADDLW k
Operandus:	$0 \leq k \leq 255$
Szimbolikus jelölés:	(W) + k → (W)
Állított állapotbitek:	C, DC, Z
14 bites kód:	11 111x kkkk kkkk
Megjegyzés:	8 bites konstans érték hozzáadása a munkaregiszterhez, az eredmény a munkaregiszterben képződik.
Szó:	1
Ciklus:	1
Példa:	ADDLW 9 ;9 hozzáadása az akkumulátorhoz

Mnemonik: **ADDWF**

Szintaxis:	[címke] ADDWF f,d
Operandus:	$0 \leq f \leq 127$ $d \in [0,1]$
Szimbolikus jelölés:	(W) + (f) → (cél)
Állított állapotbitek:	C, DC, Z
14 bites kód:	00 0111 dfff ffff
Megjegyzés:	Az 'f' helyén álló fájlregiszter tartalmát hozzáadjuk a munkaregiszterhez. Az eredményt a destination bit határozza meg ('d'=0 → W, 'd'=1 → f).
Szó:	1
Ciklus:	1
Példa:	ADDWF COUNT,F ;count regiszterhez akkumulátor ;hozzáadása, eredmény a count- ;ban

Mnemonik: **SUBLW**

Szintaxis:	[címke]	SUBLW	k
Operandus:		$0 \leq k \leq 255$	
Szimbolikus jelölés:		k - (W) → (W)	
Állított állapotbitek:		C, DC, Z	
14 bites kód:		11 110x kkkk kkkk	
Megjegyzés:		A munkaregiszter kivonása (kettes komplemens segítségével) egy 8 bites konstans értékből. Az eredmény a munkaregiszterben képződik.	
Szó:		1	
Ciklus:		1	
Példa:		SUBLW 5	;5-ből az akkumulátor kivonása

Mnemonik: **SUBWF**

Szintaxis:	[címke]	SUBWF	f,d
Operandus:		$0 \leq f \leq 127$ $d \in [0,1]$	
Szimbolikus jelölés:		f – (W) → (cél)	
Állított állapotbitek:		C, DC, Z	
14 bites kód:		00 0010 dfff ffff	
Megjegyzés:		A munkaregiszter kivonása (kettes komplemens segítségével) az 'f' helyén álló fájlregiszterből. Az eredmény a célbit függvénye ('d'=0 → W, 'd'=1 → f).	
Szó:		1	
Ciklus:		1	
Példa:		SUBWF COUNT,F	;count-ból az akkumulátor ;kivonása az eredmény a ;countban

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **INCF**

Szintaxis: [címke] INCF f,d

Operandus: $0 \leq f \leq 127$
$d \in [0,1]$

Szimbolikus jelölés: (f) + 1 → (cél)

Állított állapotbitek: Z

14 bites kód: 00 1010 dfff ffff

Megjegyzés: Az 'f' helyén álló fájlregiszter eggyel történő növelése (ha az eredeti érték 255 volt, az eredmény 0 lesz - aminek helyét a célbit határozza meg: d=0 → W, d=1 → f).

Szó: 1

Ciklus: 1

Példa:
```
INCF   PORTA,W    ;PORTA eggyel való megnövelése az
                  ;eredmény az akkumulátorban
```

Mnemonik: **DECF**

Szintaxis: [címke] DECF f,d

Operandus: $0 \leq f \leq 127$
$d \in [0,1]$

Szimbolikus jelölés: (f) – 1 → (cél)

Állított állapotbitek: Z

14 bites kód: 00 0011 dfff ffff

Megjegyzés: Az 'f' helyén álló fájlregiszter eggyel történő csökkentése (ha az eredeti érték 0 volt, az eredmény 255 lesz - aminek helyét a célbit határozza meg: d=0 → W, d=1 → f)

Szó: 1

Ciklus: 1

Példa:
```
DECF  COUNT,F     ;count eggyel való csökkentése, az
                  ;eredmény a countban
```

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **ANDLW**

Szintaxis:	[címke] ANDLW k
Operandus:	$0 \leq k \leq 255$
Szimbolikus jelölés:	(W) AND k → (W)
Állított állapotbitek:	Z
14 bites kód:	11 1001 kkkk kkkk
Megjegyzés:	A munkaregiszter és egy 8 bites konstans logikai ÉS kapcsolatát adja eredményül. Az eredmény a munkaregiszterben képződik.
Szó:	1
Ciklus:	1
Példa:	ANDLW 7 ;7 és az akkumulátor ÉS kapcsolata

Mnemonik: **ANDWF**

Szintaxis:	[címke] ANDWF f,d
Operandus:	$0 \leq f \leq 127$ $d \in [0,1]$
Szimbolikus jelölés:	(W) AND (f) → (cél)
Állított állapotbitek:	Z
14 bites kód:	00 0101 dfff ffff
Megjegyzés:	Logikai ÉS kapcsolat a munkaregiszter és az 'f' helyén álló fájlregiszter között az eredmény d=0 esetén a munkaregiszterben, d=1 esetén a fájlregiszterben kerül tárolásra.
Szó:	1
Ciklus:	1
Példa:	ANDWF PORTA,W ;porta és az akkumulátor ÉS ;kapcsolata, eredmény az ;akkumulátorban

Mnemonik: **IORLW**

Szintaxis:	[címke] IORLW k
Operandus:	$0 \leq k \leq 255$
Szimbolikus jelölés:	(W) VAGY k → (W)
Állított állapotbitek:	Z
14 bites kód:	11 1000 kkkk kkkk
Megjegyzés:	A munkaregiszter és egy 8 bites konstans logikai VAGY kapcsolatát adja eredményül. Az eredmény a munkaregiszterben képződik.
Szó:	1
Ciklus:	1
Példa:	IORLW 7 ; 7 és az akkumulátor VAGY kapcsolata

Mnemonik: **IORWF**

Szintaxis:	[címke] IORWF f,d
Operandus:	$0 \leq f \leq 127$ $d \in [0,1]$
Szimbolikus jelölés:	(W) VAGY (f) → (cél)
Állított állapotbitek:	Z
14 bites kód:	00 0100 dfff ffff
Megjegyzés:	Logikai VAGY kapcsolat a munkaregiszter és az 'f' helyén álló fájlregiszter között az eredmény d=0 esetén a munkaregiszterben, d=1 esetén a fájlregiszterben kerül tárolásra.
Szó:	1
Ciklus:	1
Példa:	IORWF PORTA,W ;porta, és az akkumulátor VAGY ;kapcsolata eredmény az ;akkumulátorban

Mnemonik: **XORLW**

Szintaxis:	[címke]	XORLW	k
Operandus:	$0 \leq k \leq 255$		
Szimbolikus jelölés:	(W) KIZÁRÓ-VAGY k → (W)		
Állított állapotbitek:	Z		
14 bites kód:	11 1010 kkkk kkkk		
Megjegyzés:	A munkaregiszter és egy 8 bites konstans KIZÁRÓ-VAGY kapcsolata. Az eredmény a munkaregiszterben képződik.		
Szó:	1		
Ciklus:	1		
Példa:	XORLW	8	;8 és az akkumulátor KIZÁRÓ-VAGY ;kapcsolata

Mnemonik: **XORWF**

Szintaxis:	[címke]	XORWF	f,d
Operandus:	$0 \leq f \leq 127$ $d \in [0,1]$		
Szimbolikus jelölés:	(W) KIZÁRÓ-VAGY (f) → (cél)		
Állított állapotbitek:	Z		
14 bites kód:	00 0110 dfff ffff		
Megjegyzés:	KIZÁRÓ-VAGY kapcsolat a munkaregiszter és az 'f' helyén álló fájlregiszter között az eredményt d=0 esetén a munkaregiszterben tároljuk, d=1 esetén pedig visszaírjuk a fájlregiszterbe.		
Szó:	1		
Ciklus:	1		
Példa:	XORWF	PORTB,W	; portb és az akkumulátor ;KIZÁRÓ VAGY kapcsolata az ;eredmény az akkumulátorban

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **COMF**

Szintaxis: [címke] COMF f,d

Operandus: $0 \leq f \leq 127$
$d \in [0,1]$

Szimbolikus jelölés: $(\bar{f}) \rightarrow$ (cél)

Állított állapotbitek: Z

14 bites kód: 00 1001 dfff ffff

Megjegyzés: Az 'f' helyén álló fájlregiszter egyes komplemensét adja eredményül (invertálja a fájlregiszter tartalmát). Az eredmény d=0 esetén a W regiszterbe, d=1 esetén a fájlregiszterbe kerül.

Szó: 1

Ciklus: 1

Példa:

```
COMF    PORTB,F    ;portb-n minden bit invertálása
```

Mnemonik: **SWAPF**

Szintaxis: [címke] SWAPF f,d

Operandus: $0 \leq f \leq 127$
$d \in [0,1]$

Szimbolikus jelölés: (f <3:0>) → (cél<7:4>)
(f <7:4>) → (cél<3:0>)

Állított állapotbitek: Nincs

14 bites kód: 00 1110 dfff ffff

Megjegyzés: Az 'f' helyén álló fájlregiszterben megcseréli az alsó és a felső 4 bitet. Az eredmény d=0 esetén a munkaregiszterben, d=1 esetén a fájlregiszterben kerül tárolásra.

Szó: 1

Ciklus: 1

Példa:

```
SWAPF   STATUS,W   ;status regiszter alsó és felső
                   ;négy bitjének cseréje, eredmény
                   ;az akkumulátorban
```

Mnemonik: **RLF**

Szintaxis: [címke] RLF f,d

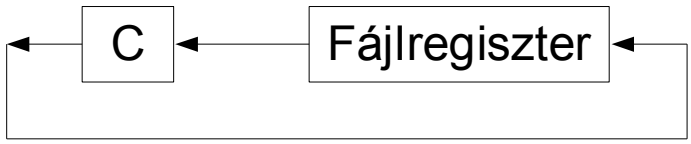
Operandus: $0 \leq f \leq 127$
$d \in [0,1]$

Szimbolikus jelölés: Lásd a megjegyzésben

Állított állapotbitek: C

14 bites kód: 00 1101 dfff ffff

Megjegyzés: Az 'f' helyén álló fájlregiszter bitjein egyet balra forgat a Carry (átvitel) biten keresztül. Az eredmény d=0 esetén a munkaregiszterben, d=1 esetén a fájlregiszterben kerül tárolásra.



Szó: 1

Ciklus: 1

Példa:
```
RLF    COUNT,F    ;bitforgatás a count regiszteren balra, az
                  ;MSB a Carry bitbe kerül, eredmény a
                  ;countban
```

Mnemonik: **RRF**

Szintaxis: [címke] RRF f,d

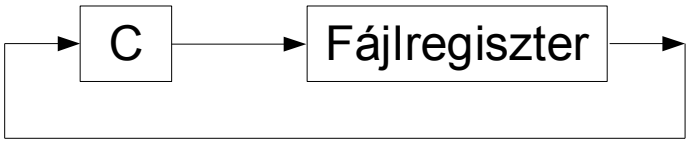
Operandus: $0 \leq f \leq 127$
$d \in [0,1]$

Szimbolikus jelölés: f<n> → cél<n-1>
f<0> → C
C → cél<7>

Állított állapotbitek: C

14 bites kód: 00 1100 dfff ffff

Megjegyzés: Az 'f' helyén álló fájlregiszter bitjein egyet jobbra forgat a Carry (átvitel) biten keresztül. Az eredmény d=0 esetén a munkaregiszterben, d=1 esetén a fájlregiszterben kerül tárolásra.



Szó: 1

Ciklus: 1

Példa:
```
RRF    COUNT,F     ;bitforgatás a count regiszteren jobbra,
                   ;az LSB a Carry bitbe kerül, eredmény a
                   ;countban
```

Mnemonik: **BCF**

Szintaxis: [címke] BCF f,b

Operandus: $0 \leq f \leq 127$
$0 \leq b \leq 7$

Szimbolikus jelölés: $0 \rightarrow (f<b>)$

Állított állapotbitek: Nincs

14 bites kód: 01 00bb bfff ffff

Megjegyzés: Az 'f' helyen álló fájlregiszter 'b' bitjének törlése (nullába állítása).

Szó: 1

Ciklus: 1

Példa:
```
BCF    PORTA,0     ;porta nulladik bitje:0
```

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **BSF**

Szintaxis:	[címke] BSF f,b
Operandus:	$0 \leq f \leq 127$ $0 \leq b \leq 7$
Szimbolikus jelölés:	$1 \rightarrow f(<b>)$
Állított állapotbitek:	Nincs
14 bites kód:	01 01bb bfff ffff
Megjegyzés:	Az 'f' helyen álló fájlregiszter 'b' bitjének egybe állítása.
Szó:	1
Ciklus:	1
Példa:	BSF PORTA,0 ;porta nulladik bitje:1

Mnemonik: **CLRF**

Szintaxis:	[címke] CLRF f
Operandus:	$0 \leq f \leq 127$
Szimbolikus jelölés:	$0 \rightarrow (f)$ $1 \rightarrow (Z)$
Állított állapotbitek:	Z
14 bites kód:	00 0001 1fff ffff
Megjegyzés:	Az 'f' helyen álló fájlregiszter törlése (mind a 8 bit kinullázása) és a zéró bit egybe állítása.
Szó:	1
Ciklus:	1
Példa:	CLRF COUNT ;count regiszter törlése

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **CLRW**

Szintaxis: [címke] CLRW

Operandus:

Szimbolikus jelölés: $0 \rightarrow (W)$
$1 \rightarrow Z$

Állított állapotbitek: Z

14 bites kód: 00 0001 0xxx xxxx

Megjegyzés: A munkaregiszter törlése (mind a 8 bit kinullázása) és a zéró bit egybe állítása.

Szó: 1

Ciklus: 1

Példa: CLRW ;akkumulátor törlése

Mnemonik: **CLRWDT**

Szintaxis: [címke] CLRWDT

Operandus: Nincs

Szimbolikus jelölés: $0 \rightarrow$ WDT
$0 \rightarrow$ WDT előosztó
$1 \rightarrow \overline{TO}$
$1 \rightarrow \overline{PD}$

Állított állapotbitek: $\overline{TO}$, $\overline{PD}$

14 bites kód: 00 0000 0110 0100

Megjegyzés: A Watch Dog Timer és a hozzá tartozó előosztó törlése.
$\overline{TO}$, $\overline{PD}$ bitek egybe állítása

Szó: 1

Ciklus: 1

Példa: CLRWDT ;Watch Dog Timer törlése

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **GOTO**

Szintaxis:	[címke]	GOTO k	
Operandus:	$0 \leq k \leq 2047$		
Szimbolikus jelölés:	k → PC<10:0> PCLATH<4:3> → PC<12:11>		
Állított állapotbitek:	Nincs		
14 bites kód:	10 1kkk kkkk kkkk		
Megjegyzés:	Feltétel nélküli elágazás. A 'k' helyen megadott 11 bites közvetlen címet betölti a PC-be (PC<10:1>). A felső biteket a PCLATH <4:3> bitjeiből tölti be. Kétciklusos utasítás.		
Szó:	1		
Ciklus:	2		
Példa:	GOTO	CIKLUS	;CIKLUS címkével ellátott ;programrészre történő ugrás

Mnemonik: **BTFSC**

Szintaxis:	[címke]	BTFSC	f,b
Operandus:	$0 \leq f \leq 127$ $0 \leq b \leq 7$		
Szimbolikus jelölés:	átlépés, ha (f<b>) = 0		
Állított állapotbitek:	Nincs		
14 bites kód:	01 10bb bfff ffff		
Megjegyzés:	Feltételes elágazás. Amennyiben az 'f' által jelölt fájlregiszter 'b' bitje 1 értékű, akkor a következő utasítás hajtódik végre – ebben az esetben egyciklusos utasítás. Amennyiben az 'f' által jelölt fájlregiszter 'b' bitje 0 értékű, akkor a következő utasítást elhagyjuk és helyette egy NOP utasítást hajtunk végre - ebben az esetben kétciklusos utasítás.		

Szó:	1		
Ciklus:	1,2		
Példa:	BTFSC	STATUS,C	;Carry bit tesztelése, ha 0 a ;következő utasítás átugrása

Mnemonik: **BTFSS**

Szintaxis:	[címke]	BTFSSf,b	
Operandus:	$0 \leq f \leq 127$ $0 \leq b \leq 7$		
Szimbolikus jelölés:	átlépés ha f(<b>) = 0		
Állított állapotbitek:	Nincs		
14 bites kód:	01 11bb bfff ffff		
Megjegyzés:	Feltételes elágazás. Amennyiben az 'f' által jelölt fájlregiszter 'b' bitje 0 értékű, akkor a következő utasítás hajtódik végre – ebben az esetben egyciklusos utasítás. Amennyiben az 'f' által jelölt fájlregiszter 'b' bitje 1 értékű, akkor a következő utasítást elhagyjuk és helyette egy NOP utasítást hajtunk végre - ebben az esetben kétciklusos utasítás.		
Szó:	1		
Ciklus:	1,2		
Példa:	BTFSS	STATUS,C	;Carry bit tesztelése, ha 1 a ;következő utasítás átugrása

Mnemonik: **INCFSZ**

Szintaxis:	[címke]	INCFSZ	f,d
Operandus:	$0 \leq f \leq 127$		
Szimbolikus jelölés:	(f) + 1 → (cél) átlépés ha az eredmény 0		
Állított állapotbitek:	Nincs		
14 bites kód:	00 1111 dfff ffff		

Megjegyzés: Feltételes elágazás. Az utasítás növeli az 'f' helyen álló fájlregiszter tartalmát eggyel, majd az eredményt a cél helyére írja. Amennyiben az eredmény nem egyenlő nullával, akkor a következő utasítás hajtódik végre – ebben az esetben egyciklusos utasítás. Ha az utasítás hatására az eredmény nulla, akkor a következő utasítást elhagyjuk és helyette egy NOP utasítást hajtunk végre - ebben az esetben kétciklusos utasítás.

Szó: 1

Ciklus: 1,2

Példa:

```
INCFSZ    COUNT,F    ;count regiszter növelése
                     ;eggyel az eredmény visszakerül
                     ;a count-ba, ha elértük a nullát a
                     ;következő utasítást átugorjuk
```

Mnemonik: **DECFSZ**

Szintaxis: [címke] DECFSZ f,d

Operandus: $0 \leq f \leq 127$

Szimbolikus jelölés: (f) - 1 → (cél)
átlépés ha az eredmény 0

Állított állapotbitek: Nincs

14 bites kód: 00 1011 dfff ffff

Megjegyzés: Feltételes elágazás. Az utasítás csökkenti az 'f' helyen álló fájlregiszter tartalmát eggyel, majd az eredményt a célregiszterbe írja. Amennyiben a célregiszter értéke nem egyenlő nullával, akkor a következő utasítás hajtódik végre – ebben az esetben egyciklusos utasítás. Ha az utasítás hatására elérte a nullát, akkor a következő utasítást elhagyjuk és helyette egy NOP utasítást hajtunk végre - ebben az esetben kétciklusos utasítás.

Szó: 1

Ciklus: 1,2

Példa:

```
DECFSZ    COUNT,F    ;count regiszter csökkentése
                     ;eggyel az eredmény visszakerül
                     ;a count-ba ha elértük a nullát a
                     ;következő utasítást átugorjuk
```

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **CALL**

Szintaxis: [címke] CALL k

Operandus: $0 \leq k \leq 2047$

Szimbolikus jelölés: PC → TOS (Top Of the Stack)
k → PC<10:0>
PCLATH<4:3> → PC<12:11>

Állított állapotbitek: Nincs

14 bites kód: 10 0kkk kkkk kkkk

Megjegyzés: Szubrutinhívó utasítás. Első lépésként a visszatérési címet (PC) a **hardveres** verem tetejébe (Top Of the Stack) helyezi. Ezután a megadott 11 bites 'k' címet a PC-be teszi (PC <10:0>), a felső biteket a PCLATH-ből (PCLATH <4:3>) tölti a PC-be. Kétciklusos utasítás. (Az utasítás hatására az SP automatikusan állítódik)

Szó: 1

Ciklus: 2

Példa: CALL DELAY ;delay rutin meghívása

Mnemonik: **RETURN**

Szintaxis: [címke] RETURN

Operandus: Nincs

Szimbolikus jelölés: TOS → PC

Állított állapotbitek: Nincs

14 bites kód: 00 0000 0000 1000

Megjegyzés: Visszatérés szubrutinból. A **hardveres** verem tetején lévő értéket (TOS) a PC-be helyezi. Kétciklusos utasítás. (Az utasítás hatására az SP automatikusan állítódik)

Szó: 1

Ciklus: 2

Példa: RETURN ;visszatérés szubrutinból

Mnemonik: **RETLW**

Szintaxis:	[címke] RETLW k
Operandus:	$0 \leq k \leq 255$
Szimbolikus jelölés:	k → (W) TOS → PC
Állított állapotbitek:	Nincs
14 bites kód:	11 01xx kkkk kkkk
Megjegyzés:	Visszatérés szubrutinból konstans értékkel. A 'k' helyen álló 8 bites értéket betölti a munkaregiszterbe – majd a verem tetején lévő értéket (TOS) betölti a PC-be. (Az utasítás hatására az SP automatikusan állítódik.)
Szó:	1
Ciklus:	2

Példa:

```
RETLW   0       ;az akkumulátorba 0 betöltése
                ;visszatérés szubrutinból
```

Mnemonik: **RETFIE**

Szintaxis:	[címke] RETFIE
Operandus:	Nincs
Szimbolikus jelölés:	TOS → PC 1 → GIE
Állított állapotbitek:	Nincs
14 bites kód:	00 0000 0000 1001
Megjegyzés:	Visszatérés megszakításból. A verem tetején lévő értéket (TOS) betölti a PC-be, majd az INTCON <GIE> bitet egybe állítja. (Az utasítás hatására az SP automatikusan állítódik.)
Szó:	1
Ciklus:	2

Példa:

```
RETFIE  ;visszatérés megszakításból GIE bit egybe
        ;állítása
```

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **SLEEP**

Szintaxis:	[címke] SLEEP
Operandus:	Nincs
Szimbolikus jelölés:	0 → WDT 0 → WDT előosztó 1 → $\overline{TO}$ 0 → $\overline{PD}$
Állított állapotbitek:	$\overline{TO}$, $\overline{PD}$
14 bites kód:	00 0000 0110 0011
Megjegyzés:	Energiatakarékos alvó mód. A CPU oszcillátora megáll, az I/O portok őrzik állapotukat. A CPU törli a WDT-t és előosztóját is, a $\overline{PD}$ bitet 0-ba, a $\overline{TO}$ bitet 1-be váltja.
Szó:	1
Ciklus:	1
Példa:	SLEEP ;alvó mód, ébredés esetén következő utasítás, ;vagy megszakítási vektor

Mnemonik: **NOP**

Szintaxis:	[címke] NOP
Operandus:	Nincs
Szimbolikus jelölés:	Nincs művelet.
Állított állapotbitek:	Nincs
14 bites kód:	00 0000 0xx0 0000
Megjegyzés:	Nincs művelet. (1 gépi ciklus idejére a processzor várakozik.)
Szó:	1
Ciklus:	1
Példa:	NOP ; nincs utasítás, 1 gépi ciklusig üresjárat

Az itt felsorolt utasítások és az azokra vonatkozó információk a PIC mikrokontrollerek 16-os családjára vonatkoznak.

Láthatjuk, hogy igen kis számú (35 db) utasítással kell megelégednünk a fejlesztés során. A kérdés, hogy mit kapunk cserébe?!

- Minden utasításunk egyszavas – a harvard architektúrának köszönhetően.
- Minden utasításunk 1 gépi ciklus alatt végrehajtódik – kivéve, melyben a PC-t állítani kell (ugró és szubrutinhívó, -visszatérő utasítások).
- Egyszerű címzés.

PIC mikrovezérlők 18-as családjának változásai az utasításkészletre vonatkozóan:

- Megjelennek új utasítások.
- Megnövelt adatmemória (Fájlregiszter tartomány: 0÷4095).
- 16 bites utasításhossz.
- A *CALL*; *GOTO*; *MOVFF*; *LFSR* utasítások 2 szavasak (így átlátjuk a teljes memóriát).
- Az adatmemóriában nincs szükség bankváltásra, ha az *ACCESS RAM*-ot használjuk.
- Az eddig bemutatott utasítások használhatóak, kivéve az *RLF*; *RRF* – az *RLNCF* és az *RRNCF* utasítások a *18*-as megfelelőik.

Új utasítások és funkcióik

Mnemonik	Jelentés	Funkció	Gépi ciklus	Állított jelző bitek
Adatmozgató utasítások				
LFSR f,k	move Literal to FSR	Konstans érték FSR-be helyezése	2	-
MOVLB k	MOVe Literal to Bsr<3:0>	Konstans érték BSR-be helyezése	1	-
MOVFF fs,fd	MOVe File to File	Fájlregiszterből (fs) fájlregiszterbe (fd) töltés	2	-
TBLRD*	TaBLe ReaD	Tábla olvasása a táblamutató nem változik	2	-
TBLRD*+	TaBLe ReaD with post-increment	Tábla olvasása a táblamutató utólagos növelésével		-
TBLRD*-	TaBLe ReaD with post-decrement	Tábla olvasása a táblamutató utólagos csökkentésével		-
TBLRD+*	TaBLe ReaD with pre-increment	Tábla olvasása a táblamutató előzetes növelésével		-
TBLWT*	TaBLe WriTe	Tábla írása a táblamutató nem változik	2	-
TBLWT*+	TaBLe WriTe with post-increment	Tábla írása a táblamutató utólagos növelésével		-
TBLWT*-	TaBLe WriTe with post-decrement	Tábla írása a táblamutató utólagos csökkentésével		-
TBLWT+*	TaBLe WriTe with pre-increment	Tábla írása a táblamutató előzetes növelésével		-
Aritmetikai utasítások				
ADDWFC f,d,a	Add W and Carry to File	W és Carry hozzáadása a fájlregiszterhez	1	C,DC, Z,OV, N
MULLW k	MULtiply Literal with Wreg	Konstans és W szorzása	1	-
MULWF f,a	MULtiply Wreg with File	Fájlregiszter és W szorzása	1	-
SUBFWB f,d,a	Subtract file from W with Borrow	W-ből a fájlregiszter és a Borrow kivonása	1	C,DC, Z,OV, N
SUBWFB f,d,a	Subtract W from file with Borrow	Fájlregiszterből W és a Borrow kivonása	1	C,DC, Z,OV, N

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Logikai utasítások				
BTG f,b,a	Bit Toggle File	Fájlregiszter b bitjének invertálása	1	-
NEGF f,a	NEGate File	Fájlregiszter kettes komplemensének képzése	1	C,DC, Z,OV, N
RLCF f,d,a	Rotate Left through Carry File	Forgatás a biteken balra, átvitelbiten keresztül	1	C,Z,N
RLNCF f,d,a	Rotate Left Not through Carry File	Forgatás a biteken balra	1	Z,N
RRCF f,d,a	Rotate Right through Carry File	Forgatás a biteken jobbra, átvitelbiten keresztül	1	C,Z,N
RRNCF f,d,a	Rotate Right Not through Carry File	Forgatás a biteken jobbra	1	Z,N
SETF f,a	SET File	Fájlregiszter feltöltése FF_h értékkel	1	-
Vezérlésátadó utasítások				
BC n	Branch if Carry	Ugrás, ha az átvitel bit értéke 1	1 (2)	-
BN n	Branch if Negative	Ugrás, ha az előjel bit értéke 1	1 (2)	-
BNC n	Branch if Not Carry	Ugrás, ha az átvitel bit értéke 0	1 (2)	-
BNN n	Branch if Not Negative	Ugrás, ha az előjel bit értéke 0	1 (2)	-
BNOV n	Branch if Not OVerflow	Ugrás, ha a túlcsordulás bit értéke 0	1 (2)	-
BNZ n	Branch if Not Zero	Ugrás, ha a zéró bit értéke 0	1 (2)	-
BOV n	Branch if OVerflow	Ugrás, ha a túlcsordulás bit értéke 1	1 (2)	-
BRA n	BRAnch unconditionally	Feltétel nélküli elágazás	2	-
BZ n	Branch if Zero	Ugrás, ha a zéró bit értéke 1	1 (2)	-
CPFSEQ f,a	ComPare File with wreg Skip if =	Fájlregiszter és W összehasonlítása és ugrás, ha egyenlőek	1 (2,3)	-
CPFSGT f,a	ComPare File with wreg Skip if >	Fájlregiszter és W összehasonlítása és ugrás, ha a fájlregiszter nagyobb	1 (2,3)	-
CPFSLT f,a	ComPare File with wreg Skip if <	Fájlregiszter és W összehasonlítása és ugrás, ha a fájlregiszter kisebb	1 (2,3)	-
DAW	Decimal Adjust W	W decimális korrekciója	1	C
DCFSNZ f,d,a	DeCrement File Skip if Not Zero	Fájlregiszter csökkentése, ugrás ha nem nulla	1 (2,3)	-
INFSNZ f,d,a	INcrement File Skip if Not Zero	Fájlregiszter növelése, ugrás ha nem nulla	1 (2,3)	-

PIC mikrovezérlő utasításkészlete (16-os sorozat)

POP	POP top of return stack (TOS)	A verem tetején lévő érték eldobása	1	-
PUSH	PUSH top of return stack (TOS)	PC értékének mentése a verem tetejére	1	-
RCALL n	Relative CALL	Rövid szubrutinhívás	2	-
RESET	Software device RESET	Szoftveres reszet	1	All
TSTFSZ f,a	TeST fájl Skip if Zero	Fájlregiszter tesztelése, ugrás ha nulla	1 (2,3)	-

Megjegyzés:

BSR: Bank Select Register (Bankválasztó regiszter)

RAM hozzáférés bit:

a = 0: A hely az azonnali hozzáférésű RAM-ban (BSR regiszter kihagyása)
a = 1: Helyét a BSR regiszter segítségével kell meghatározni

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **LFSR**

Szintaxis:	[címke]	LFSR	f,k		
Operandus:	$0 \le f \le 2$ $0 \le k \le 4095$				
Szimbolikus jelölés:	k → FSRf				
Állított állapotbitek:	Nincs				
16 bites kód:	1110	1110	00ff	k_{11}kkk	(1. szó)
	1111	0000	$k_7$kk	kkkk	(2. szó)
Megjegyzés:	A 12 bites konstans értéket betöltjük az FSRf regiszterbe				
Szó:	2				
Ciklus:	2				

Mnemonik: **MOVLB**

Szintaxis:	[címke]	MOVLB	k	
Operandus:	$0 \le k \le 255$			
Szimbolikus jelölés:	k → BSR			
Állított állapotbitek:	Nincs			
16 bites kód:	0000	0001	kkkk	kkkk
Megjegyzés:	8-bites konstans betöltése a BSR (Bank Select Register) regiszterbe. A BSR <7:4> bitek mindig nullák maradnak, tekintet nélkül a k<7:4> bitekre.			
Szó:	1			
Ciklus:	1			

Mnemonik: **MOVFF**

Szintaxis:	[címke] MOVFF fs,fd
Operandus:	$0 \leq fs \leq 4095$ $0 \leq fd \leq 4095$
Szimbolikus jelölés:	(fs) → fd
Állított állapotbitek:	Nincs
16 bites kód:	1100 ffff ffff $ffff_s$ (1. szó) 1111 ffff ffff $ffff_d$ (2. szó)
Megjegyzés:	A forrás fájlregiszter (f_s) értékét a cél fájlregiszter-be (f_d) töltjük. Mind a forrás, mind a cél lehet a W. Célregiszterként nem használhatjuk a PCL, TOSU, TOSH, TOSL regisztereket.
Szó:	2
Ciklus:	2(3)

Mnemonik: **TBLRD***

Szintaxis:	[címke] TBLRD*
Operandus:	Nincs
Szimbolikus jelölés:	(Prog mem (TBLPTR)) → TABLAT
Állított állapotbitek:	Nincs
16 bites kód:	0000 0000 0000 0000
Megjegyzés:	A táblamutató (TBLPTR) által mutatott programmemória-rekesz tartalma a TABLAT regiszterbe kerül. A táblamutató értéke nem változik.
Szó:	1
Ciklus:	2

Mnemonik: **TBLRD*+**

Szintaxis:	[címke] TBLRD*+
Operandus:	Nincs
Szimbolikus jelölés:	(Prog mem (TBLPTR)) → TABLAT (TBLPTR) +1 → TBLPTR
Állított állapotbitek:	Nincs
16 bites kód:	0000 0000 0000 0001
Megjegyzés:	A táblamutató (TBLPTR) által mutatott programmemória-rekesz tartalma a TABLAT regiszterbe kerül – ezután a táblamutató értékét megnöveljük eggyel.
Szó:	1
Ciklus:	2

Mnemonik: **TBLRD*-**

Szintaxis:	[címke] TBLRD*-
Operandus:	Nincs
Szimbolikus jelölés:	(Prog mem (TBLPTR)) → TABLAT. (TBLPTR) – 1 → TBLPTR
Állított állapotbitek:	Nincs
16 bites kód:	0000 0000 0000 0010
Megjegyzés:	A táblamutató (TBLPTR) által mutatott programmemória-rekesz tartalma a TABLAT regiszterbe kerül – ezután a táblamutató értékét csökkentjük eggyel.
Szó:	1
Ciklus:	2

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **TBLRD+***

Szintaxis:	[címke] TBLRD+*
Operandus:	Nincs
Szimbolikus jelölés:	(TBLPTR) +1 → TBLPTR (Prog mem (TBLPTR)) → TABLAT
Állított állapotbitek:	Nincs
16 bites kód:	0000 0000 0000 0011
Megjegyzés:	A táblamutató (TBLPTR) értékét eggyel megnöveljük – ezután a táblamutató által kijelölt programmemória-rekesz tartalma a TABLAT regiszterbe kerül.
Szó:	1
Ciklus:	2

Mnemonik: **TBLWT***

Szintaxis:	[címke] TBLWT*
Operandus:	Nincs
Szimbolikus jelölés:	(TABLAT) → (Prog mem (TBLPTR))
Állított állapotbitek:	Nincs
16 bites kód:	0000 0000 0000 1100
Megjegyzés:	A TABLAT regiszter értékét a táblamutató (TBLPTR) által kijelölt programmemória-rekeszbe mozgatjuk. A táblamutató értéke nem változik.
Szó:	1
Ciklus:	2

Mnemonik: **TBLWT*+**

Szintaxis:	[címke] TBLWT*+
Operandus:	Nincs
Szimbolikus jelölés:	(TABLAT) → (Prog mem (TBLPTR)) (TBLPTR) +1 → TBLPTR
Állított állapotbitek:	Nincs
16 bites kód:	0000 0000 0000 1101
Megjegyzés:	A TABLAT regiszter értékét a táblamutató (TBLPTR) által kijelölt programmemória-rekeszbe mozgatjuk – ezután a táblamutató értékét eggyel megnöveljük.
Szó:	1
Ciklus:	2

Mnemonik: **TBLWT*-**

Szintaxis:	[címke] TBLWT*-
Operandus:	Nincs
Szimbolikus jelölés:	(TABLAT) → (Prog mem (TBLPTR)) (TBLPTR) -1 → TBLPTR
Állított állapotbitek:	Nincs
16 bites kód:	0000 0000 0000 1110
Megjegyzés:	A TABLAT regiszter értékét a táblamutató (TBLPTR) által kijelölt programmemória-rekeszbe mozgatjuk – ezután a táblamutató értékét csökkentjük eggyel.
Szó:	1
Ciklus:	2

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **TBLWT+***

Szintaxis:	[címke] TBLWT+*
Operandus:	Nincs
Szimbolikus jelölés:	(TBLPTR) +1 → TBLPTR (TABLAT) → (Prog mem (TBLPTR))
Állított állapotbitek:	Nincs
16 bites kód:	0000 0000 0000 1111
Megjegyzés:	A táblamutató (TBLPTR) értékét eggyel megnöveljük – majd a TABLAT regiszter értékét a táblamutató által kijelölt programmemória-rekeszbe mozgatjuk.
Szó:	1
Ciklus:	2

Mnemonik: **ADDWFC**

Szintaxis:	[címke] ADDWFC f{,d {,a}}
Operandus:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Szimbolikus jelölés:	(W) + (f) + (C) → (cél)
Állított állapotbitek:	N, OV, C, DC, Z
16 bites kód:	0010 00da ffff ffff
Megjegyzés:	Az akkumulátor értékéhez az f helyén álló fájlregiszter értékének, és a Carry bitnek a hozzáadása. Az eredmény d=0 esetén az akkumulátorba, d=1 esetén a fájlregiszterbe kerül.
Szó:	1
Ciklus:	1

Mnemonik: **MULLW**

Szintaxis:	[címke] MULLW k
Operandus:	$0 \leq k \leq 255$
Szimbolikus jelölés:	k x (W) → PRODH:PRODL
Állított állapotbitek:	Nincs
16 bites kód:	0000 1101 kkkk kkkk
Megjegyzés:	8 bites konstans szorzása az akkumulátorral. A 16 bites eredmény a PRODH (felső bájt) és PRODL (alsó bájt) regiszterbe kerül. Az akkumulátor értéke nem változik.
Szó:	1
Ciklus:	1

Mnemonik: **MULWF**

Szintaxis:	[címke] MULWF f{,a}
Operandus:	$0 \leq f \leq 255$ $a \in [0,1]$
Szimbolikus jelölés:	(f) x (W) → PRODH:PRODL
Állított állapotbitek:	Nincs
16 bites kód:	0000 001a ffff ffff
Megjegyzés:	Fájlregiszter szorzása az akkumulátorral. A 16 bites eredmény a PRODH (felső bájt) és PRODL (alsó bájt) regiszterbe kerül. Az akkumulátor és a fájlregiszter értéke nem változik.
Szó:	1
Ciklus:	1

Mnemonik: **SUBFWB**

Szintaxis:	[címke] SUBFWB f{,d{,a}}
Operandus:	$0 \leq f \leq 255$ $d \in [0,1]$ $a \in [0,1]$
Szimbolikus jelölés:	(W) – (f) – (B) → cél
Állított állapotbitek:	N, OV, C, DC, Z
16 bites kód:	0101 01da ffff ffff
Megjegyzés:	Az akkumulátorból az f regiszter értékének, és a Borrow bitnek a kivonása (PIC mikrovezérlők esetén a Borrow bit a Carry negáltja). Az eredmény d=0 esetén az akkumulátorban, d=1 esetén a fájlregiszterben képződik.
Szó:	1
Ciklus:	1

Mnemonik: **SUBWFB**

Szintaxis:	[címke] SUBWFB f{,d{,a}}
Operandus:	$0 \leq f \leq 255$ $d \in [0,1]$ $a \in [0,1]$
Szimbolikus jelölés:	(f) – (W) – (B) → cél
Állított állapotbitek:	N, OV, C, DC, Z
16 bites kód:	0101 10da ffff ffff
Megjegyzés:	Az f regiszterből az akkumulátorból értékének, és a Borrow bitnek a kivonása (PIC mikrovezérlők esetén a Borrow bit a Carry negáltja). Az eredmény d=0 esetén az akkumulátorban, d=1 esetén a fájlregiszterben képződik.
Szó:	1
Ciklus:	1

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **BTG**

Szintaxis: [címke] BTG f,b{a}

Operandus: $0 \le f \le 255$
$0 \le b \le 7$
$a \in [0,1]$

Szimbolikus jelölés: $\overline{(f<b>)} \rightarrow f<b>$

Állított állapotbitek: Nincs

16 bites kód: 0111 bbba ffff ffff

Megjegyzés: Az f regiszter b helyén álló bitjének invertálása.

Szó: 1

Ciklus: 1

Mnemonik: **NEGF**

Szintaxis: [címke] NEGF f{,a}

Operandus: $0 \le f \le 255$
$a \in [0,1]$

Szimbolikus jelölés: $\bar{f} + 1 \rightarrow f$

Állított állapotbitek: N, OV, C, DC, Z

16 bites kód: 0110 110a ffff ffff

Megjegyzés: Az f helyén álló fájlregiszter kettes komplemensének képzése.

Szó: 1

Ciklus: 1

Mnemonik: **RLCF**

Szintaxis: [címke] RLCF f{,d{,a}}

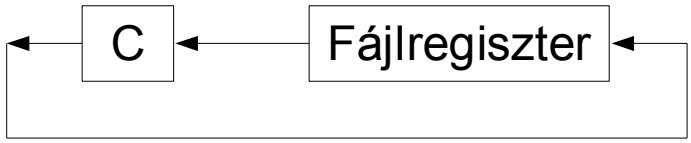
Operandus: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Szimbolikus jelölés: f<n> → cél<n+1>
f<7> → C
C → cél<0>

Állított állapotbitek: C, N, Z

14 bites kód: 0011 01da ffff ffff

Megjegyzés: Az 'f' helyén álló fájlregiszter bitjein egyet balra forgat a Carry (átvitel) biten keresztül. Az eredmény d=0 esetén a munkaregiszterben, d=1 esetén a fájlregiszterben kerül tárolásra.



Szó: 1

Ciklus: 1

Mnemonik: **RLNCF**

Szintaxis: [címke] RLNCF f{,d{,a}}

Operandus: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Szimbolikus jelölés: f<n> → cél<n+1>
f<7> → cél<0>

Állított állapotbitek: N, Z

16 bites kód: 0100 01da ffff ffff

Megjegyzés: Az 'f' helyén álló fájlregiszter bitjein egyet balra forgat. Az eredmény d=0 esetén a munkaregiszterben, d=1 esetén a fájlregiszterben kerül tárolásra.

Szó: 1

Ciklus: 1

Mnemonik: **RRCF**

Szintaxis: [címke] RRCF f{,d{,a}}

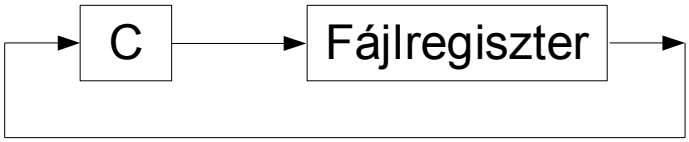
Operandus: $0 \leq f \leq 255$
$d \in [0,1]$
$a \in [0,1]$

Szimbolikus jelölés: f<n> → cél<n-1>
f<0> → C
C → cél<7>

Állított állapotbitek: C, N, Z

14 bites kód: 0011 00da ffff ffff

Megjegyzés: Az 'f' helyén álló fájlregiszter bitjein egyet jobbra forgat a Carry (átvitel) biten keresztül. Az eredmény d=0 esetén a munkaregiszterben, d=1 esetén a fájlregiszterben kerül tárolásra.



Szó: 1

Ciklus: 1

Mnemonik: **RRNCF**

Szintaxis: [címke] RRNCF f{,d{,a}}

Operandus: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Szimbolikus jelölés: f<n> → cél<n-1>
f<0> → cél<7>

Állított állapotbitek: N, Z

16 bites kód: 0100 00da ffff ffff

Megjegyzés: Az 'f' helyén álló fájlregiszter bitjein egyet jobbra forgat. Az eredmény d=0 esetén a munkaregiszterben, d=1 esetén a fájlregiszterben kerül tárolásra.

Szó: 1

Ciklus: 1

Mnemonik: **SETF**

Szintaxis: [címke] SETF f{,a}

Operandus: $0 \le f \le 255$
$a \in [0,1]$

Szimbolikus jelölés: $FF_h \rightarrow f$

Állított állapotbitek: Nincs

16 bites kód: 0110 100a ffff ffff

Megjegyzés: Az f helyén álló fájlregiszterbe FF_h értéket töltünk.

Szó: 1

Ciklus: 1

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **BC**

Szintaxis:	[címke] BC n
Operandus:	$-128 \leq n \leq 127$
Szimbolikus jelölés:	ha C = 1: (PC) + 2n → PC
Állított állapotbitek:	Nincs
16 bites kód:	1110 0010 nnnn nnnn
Megjegyzés:	Feltételes elágazás (relatív ugrással). Amennyiben a Carry bit értéke 0, akkor a következő utasítás hajtódik végre – ebben az esetben egyciklusos utasítás. Amennyiben a Carry bit értéke 1, akkor az n értékének kétszeresét hozzáadjuk a programszámlálóhoz kettes komplemensben – ha n helyén címke áll, akkor az adott címre ugrunk, feltéve, hogy a kiindulási címtől ±127 ciklusra van. Ebben az esetben kétciklusos az utasítás.
Szó:	1
Ciklus:	1 (2)

Mnemonik: **BN**

Szintaxis:	[címke] BN n
Operandus:	$-128 \leq n \leq 127$
Szimbolikus jelölés:	ha N = 1: (PC) + 2n → PC
Állított állapotbitek:	Nincs
16 bites kód:	1110 0110 nnnn nnnn
Megjegyzés:	Feltételes elágazás (relatív ugrással). Amennyiben a Negative bit értéke 0, akkor a következő utasítás hajtódik végre – ebben az esetben egyciklusos utasítás. Amennyiben a Negative bit értéke 1, akkor az n értékének kétszeresét hozzáadjuk a programszámlálóhoz kettes komplemensben – ha n helyén címke áll, akkor az adott címre ugrunk, feltéve, hogy a kiindulási címtől ±127 ciklusra van. Ebben az esetben kétciklusos az utasítás.
Szó:	1
Ciklus:	1 (2)

Mnemonik: **BNC**

Szintaxis:	[címke] BNC n
Operandus:	$-128 \leq n \leq 127$
Szimbolikus jelölés:	ha C = 0: (PC) + 2n → PC
Állított állapotbitek:	Nincs
16 bites kód:	1110 0011 nnnn nnnn
Megjegyzés:	Feltételes elágazás (relatív ugrással). Amennyiben a Carry bit értéke 1, akkor a következő utasítás hajtódik végre – ebben az esetben egyciklusos utasítás. Amennyiben a Carry bit értéke 0, akkor az n értékének kétszeresét hozzáadjuk a programszámlálóhoz kettes komplemensben – ha n helyén címke áll, akkor az adott címre ugrunk, feltéve, hogy a kiindulási címtől ±127 ciklusra van. Ebben az esetben kétciklusos az utasítás.
Szó:	1
Ciklus:	1 (2)

Mnemonik: **BNN**

Szintaxis:	[címke] BNN n
Operandus:	$-128 \leq n \leq 127$
Szimbolikus jelölés:	ha N = 0: (PC) + 2n → PC
Állított állapotbitek:	Nincs
16 bites kód:	1110 0111 nnnn nnnn
Megjegyzés:	Feltételes elágazás (relatív ugrással). Amennyiben a Negative bit értéke 1, akkor a következő utasítás hajtódik végre – ebben az esetben egyciklusos utasítás. Amennyiben a Negative bit értéke 0, akkor az n értékének kétszeresét hozzáadjuk a programszámlálóhoz kettes komplemensben – ha n helyén címke áll, akkor az adott címre ugrunk, feltéve, hogy a kiindulási címtől ±127 ciklusra van. Ebben az esetben kétciklusos az utasítás.
Szó:	1
Ciklus:	1 (2)

Mnemonik: **BNOV**

Szintaxis:	[címke] BNOV n
Operandus:	$-128 \leq n \leq 127$
Szimbolikus jelölés:	ha OV = 0: (PC) + 2n → PC
Állított állapotbitek:	Nincs
16 bites kód:	1110 0101 nnnn nnnn
Megjegyzés:	Feltételes elágazás (relatív ugrással). Amennyiben az OVerflow bit értéke 1, akkor a következő utasítás hajtódik végre – ebben az esetben egyciklusos utasítás. Amennyiben az OVerflow bit értéke 0, akkor az n értékének kétszeresét hozzáadjuk a programszámlálóhoz kettes komplemensben – ha n helyén címke áll, akkor az adott címre ugrunk, feltéve, hogy a kiindulási címtől ±127 ciklusra van. Ebben az esetben kétciklusos az utasítás.
Szó:	1
Ciklus:	1 (2)

Mnemonik: **BNZ**

Szintaxis:	[címke] BNZ n
Operandus:	$-128 \leq n \leq 127$
Szimbolikus jelölés:	ha Z = 0: (PC) + 2n → PC
Állított állapotbitek:	Nincs
16 bites kód:	1110 0001 nnnn nnnn
Megjegyzés:	Feltételes elágazás (relatív ugrással). Amennyiben az Zero bit értéke 1, akkor a következő utasítás hajtódik végre – ebben az esetben egyciklusos utasítás. Amennyiben az Zero bit értéke 0, akkor az n értékének kétszeresét hozzáadjuk a programszámlálóhoz kettes komplemensben – ha n helyén címke áll, akkor az adott címre ugrunk, feltéve, hogy a kiindulási címtől ±127 ciklusra van. Ebben az esetben kétciklusos az utasítás.
Szó:	1
Ciklus:	1 (2)

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **BOV**

Szintaxis:	[címke] BOV n
Operandus:	$-128 \leq n \leq 127$
Szimbolikus jelölés:	ha OV = 1: (PC) + 2n → PC
Állított állapotbitek:	Nincs
16 bites kód:	1110 0100 nnnn nnnn
Megjegyzés:	Feltételes elágazás (relatív ugrással). Amennyiben az OVerflow bit értéke 0, akkor a következő utasítás hajtódik végre – ebben az esetben egyciklusos utasítás. Amennyiben az OVerflow bit értéke 1, akkor az n értékének kétszeresét hozzáadjuk a programszámlálóhoz kettes komplemensben – ha n helyén címke áll, akkor az adott címre ugrunk, feltéve, hogy a kiindulási címtől ±127 ciklusra van. Ebben az esetben kétciklusos az utasítás.
Szó:	1
Ciklus:	1 (2)

Mnemonik: **BRA**

Szintaxis:	[címke] BRA n
Operandus:	$-1024 \leq n \leq 1023$
Szimbolikus jelölés:	(PC) + 2n → PC
Állított állapotbitek:	Nincs
16 bites kód:	1101 0nnn nnnn nnnn
Megjegyzés:	Feltétel nélküli ugrás (relatív). Az n értékének kétszeresét hozzáadjuk a programszámlálóhoz kettes komplemensben – ha n helyén címke áll, akkor az adott címre ugrunk, feltéve, hogy a kiindulási címtől ±1023 ciklusra van.
Szó:	1
Ciklus:	2

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **BZ**

Szintaxis:	[címke] BZ n
Operandus:	$-128 \leq n \leq 127$
Szimbolikus jelölés:	ha Z = 1: (PC) + 2n → PC
Állított állapotbitek:	Nincs
16 bites kód:	1110 0000 nnnn nnnn
Megjegyzés:	Feltételes elágazás (relatív ugrással). Amennyiben az Zero bit értéke 0, akkor a következő utasítás hajtódik végre – ebben az esetben egyciklusos utasítás. Amennyiben az Zero bit értéke 1, akkor az n értékének kétszeresét hozzáadjuk a programszámlálóhoz kettes komplemensben – ha n helyén címke áll, akkor az adott címre ugrunk, feltéve, hogy a kiindulási címtől ±127 ciklusra van. Ebben az esetben kétciklusos az utasítás.
Szó:	1
Ciklus:	1 (2)

Mnemonik: **CPFSEQ**

Szintaxis:	[címke] CPFSEQ f{,a}
Operandus:	$0 \leq f \leq 255$ $a \in [0,1]$
Szimbolikus jelölés:	(f) összehasonlítás (W) átlépés, ha (f) = (W)
Állított állapotbitek:	Nincs
16 bites kód:	0110 001a ffff ffff
Megjegyzés:	A megadott fájlregiszter és az akkumulátor értékének összehasonlítása – a következő utasítás átlépése, ha egyenlőek. Átlépés esetén kétciklusos az utasítás, ellenkező esetben egy. Amennyiben 2 szavas utasítást kell átlépnünk, akkor ezt csak 3 ciklusból tudjuk megoldani.
Szó:	1
Ciklus:	1 (2, 3)

Mnemonik: **CPFSGT**

Szintaxis:	[címke] CPFSGT f{,a}
Operandus:	$0 \leq f \leq 255$ $a \in [0,1]$
Szimbolikus jelölés:	(f) összehasonlítás (W) átlépés, ha (f) > (W)
Állított állapotbitek:	Nincs
16 bites kód:	0110 010a ffff ffff
Megjegyzés:	A megadott fájlregiszter és az akkumulátor értékének összehasonlítása – a következő utasítás átlépése, ha a fájlregiszter értéke nagyobb, mint az akkumulátoré. Átlépés esetén kétciklusos az utasítás, ellenkező esetben egy. Amennyiben 2 szavas utasítást kell átlépnünk, akkor ezt csak 3 ciklusból tudjuk megoldani.
Szó:	1
Ciklus:	1 (2, 3)

Mnemonik: **CPFSLT**

Szintaxis:	[címke] CPFSLT f{,a}
Operandus:	$0 \leq f \leq 255$ $a \in [0,1]$
Szimbolikus jelölés:	(f) összehasonlítás (W) átlépés, ha (f) < (W)
Állított állapotbitek:	Nincs
16 bites kód:	0110 000a ffff ffff
Megjegyzés:	A megadott fájlregiszter és az akkumulátor értékének összehasonlítása – a következő utasítás átlépése, ha a fájlregiszter értéke kisebb mint az akkumulátoré. Átlépés esetén kétciklusos az utasítás, ellenkező esetben egy. Amennyiben 2 szavas utasítást kell átlépnünk, akkor ezt csak 3 ciklusból tudjuk megoldani.

Szó: 1

Ciklus: 1 (2, 3)

Mnemonik: **DAW**

Szintaxis: [címke] DAW

Operandus: Nincs

Szimbolikus jelölés: ha [W<3:0> > 9] vagy [DC = 1]
akkor (W<3:0>) + 6 → W<3:0>
különben (W<3:0>) = W<3:0>;
ha [W<7:4> +DC > 9] vagy [C = 1]
akkor (W<7:4>) + 6 + DC → W<7:4>
különben (W<7:4>) + DC = W<7:4>

Állított állapotbitek: C

16 bites kód: 0000 0000 0000 0111

Megjegyzés: Az akkumulátor decimális kiegészítése (BCD számok esetén használatos).

Szó: 1

Ciklus: 1

Mnemonik: **DCFSNZ**

Szintaxis: [címke] DCFSNZ f{,d{,a}}

Operandus: $0 \leq f \leq 255$
$d \in [0,1]$
$a \in [0,1]$

Szimbolikus jelölés: (f) – 1 = cél
ugrás, ha az eredmény nem nulla.

Állított állapotbitek: Nincs

16 bites kód: 0100 11da ffff ffff

Megjegyzés: Feltételes elágazás. Az utasítás csökkenti az 'f' helyen álló fájlregiszter tartalmát eggyel, majd az eredményt a célregiszterbe írja. Amennyiben az eredmény egyenlő nullával, akkor a következő utasítás hajtódik végre – ebben az esetben egyciklusos utasítás. Ha az utasítás hatására az eredmény nem egyenlő nullával, akkor a következő utasítást elhagyjuk és helyette egy NOP utasítást hajtunk végre - ebben az esetben kétciklusos utasítás. Kétszavas utasítás átlépésére 3 gépi ciklus szükséges.

Szó: 1

Ciklus: 1 (2, 3)

Mnemonik: **INFSNZ**

Szintaxis: [címke] INFSNZ f{,d{,a}}

Operandus: $0 \leq f \leq 255$

$d \in [0,1]$

$a \in [0,1]$

Szimbolikus jelölés: (f) + 1 = cél
ugrás, ha az eredmény nem nulla

Állított állapotbitek: Nincs

16 bites kód: 0100 10da ffff ffff

Megjegyzés: Feltételes elágazás. Az utasítás növeli az 'f' helyen álló fájlregiszter tartalmát eggyel, majd az eredményt a célregiszterbe írja. Amennyiben az eredmény egyenlő nullával, akkor a következő utasítás hajtódik végre – ebben az esetben egyciklusos utasítás. Ha az utasítás hatására az eredmény nem egyenlő nullával, akkor a következő utasítást elhagyjuk és helyette egy NOP utasítást hajtunk végre - ebben az esetben kétciklusos utasítás. Kétszavas utasítás átlépésére 3 gépi ciklus szükséges.

Szó: 1

Ciklus: 1 (2, 3)

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **POP**

Szintaxis:	[címke] POP
Operandus:	Nincs
Szimbolikus jelölés:	TOS → kuka
Állított állapotbitek:	Nincs
16 bites kód:	0000 0000 0000 0110
Megjegyzés:	A verem tetején lévő érték eldobása. Az utasítás lehetővé teszi, hogy a felhasználó szoftveres úton állítsa a vermet.
Szó:	1
Ciklus:	1

Mnemonik: **PUSH**

Szintaxis:	[címke] PUSH
Operandus:	Nincs
Szimbolikus jelölés:	PC → TOS
Állított állapotbitek:	Nincs
16 bites kód:	0000 0000 0000 0101
Megjegyzés:	A programszámláló értékét (a következő utasítás címét) a verembe helyezzük.
Szó:	1
Ciklus:	1

Mnemonik: **RCALL**

Szintaxis:	[címke] RCALL n
Operandus:	$-1024 \leq n \leq 1023$
Szimbolikus jelölés:	PC → TOS (PC) + 2n → PC
Állított állapotbitek:	Nincs
16 bites kód:	1101 1nnn nnnn nnnn
Megjegyzés:	Relatív szubrutinhívó utasítás. A programszámlálót a verem tetejére helyezzük. Az n értékének kétszeresét hozzáadjuk a programszámlálóhoz kettes komplemensben. Amennyiben n helyén címke áll az utasítás működése megegyezik a CALL utasítással, de nem látja át a teljes memóriatartományt.
Szó:	1
Ciklus:	2

Mnemonik: **RESET**

Szintaxis:	[címke] RESET
Operandus:	Nincs
Szimbolikus jelölés:	Hatása megegyezik a Master Clear reszettel.
Állított állapotbitek:	Összes.
16 bites kód:	0000 0000 1111 1111
Megjegyzés:	Hatása megegyezik a Master Clear reszettel.
Szó:	1
Ciklus:	1

PIC mikrovezérlő utasításkészlete (16-os sorozat)

Mnemonik: **TSTFSZ**

Szintaxis:	[címke] TSTFSZ f{,a}
Operandus:	$0 \leq f \leq 255$ $a \in [0,1]$
Szimbolikus jelölés:	Átlépés, ha f = 0
Állított állapotbitek:	Nincs
16 bites kód:	0110 011a ffff ffff
Megjegyzés:	Az 'f' helyén álló fájlregiszter értékének vizsgálata. Amennyiben nulla, akkor a következő utasítás átlépése (kétciklusos).
Szó:	1
Ciklus:	1 (2, 3)

MPASM belső makrói

Az MPLAB fordítójába integráltak néhány – a programozást megkönnyítő – belső makródefiníciót.

Makró neve	Leírás	Makró kifejtése	Állított jelzőbitek
ADDCF f,d	Carry hozzáadása a fájlregiszterhez	BTFSC STATUS,C INCF f,d	Z
ADDDCF f,d	Digit Carry hozzáadása a fájlregiszterhez	BTFSC STATUS,DC INCF f,d	Z
B k	Feltétel nélküli ugrás	GOTO k	-
BC	Ugrás, ha a Carry 1	BTFSC STATUS,C GOTO k	-
BDC	Ugrás, ha a Digit Carry 1	BTFSC STATUS,DC GOTO k	-
BNC	Ugrás, ha a Carry 0	BTFSS STATUS,C GOTO k	-
BNDC	Ugrás, ha a Digit Carry 0	BTFSS STATUS,DC GOTO k	-
BNZ	Ugrás, ha a ZERO 0	BTFSS STATUS,Z GOTO k	-
BZ	Ugrás, ha a ZERO 1	BTFSC STATUS,Z GOTO k	-
CLRC	Carry törlése	BCF STATUS,C	-
CLRDC	Digit Carry törlése	BCF STATUS,DC	-
CLRZ	Zero törlése	BCF STATUS,Z	-
LCALL k	Hosszú szubrutinhívás	BCF/BSF PCLATH,3 BCF/BSF PCLATH,4 CALL k	-
LGOTO k	Hosszú ugrás	BCF/BSF PCLATH,3 BCF/BSF PCLATH,4 GOTO k	-
MOVFW f	A fájregiszter tartalmának W-be töltése	MOVF f,W	Z
NEGF f,d	Fájlregiszter kettes komplemensének képzése	COMF f,d INCF f,d	Z
SETC	Carry 1-be állítása	BSF STATUS,C	-
SETDC	Digit Carry 1-be állítása	BSF STATUS,DC	-
SETZ	Zero 1-be állítása	BSF STATUS,Z	-

MPASM belső makrói

SKPC	Következő utasítás átugrása, ha Carry 1	BTFSS STATUS,C	-
SKPDC	Következő utasítás átugrása, ha Digit Carry 1	BTFSS STATUS,DC	-
SKPNC	Következő utasítás átugrása, ha Carry 0	BTFSC STATUS,C	-
SKPNDC	Következő utasítás átugrása, ha Digit Carry 0	BTFSC STATUS,DC	-
SKPNZ	Következő utasítás átugrása, ha Zero 0	BTFSC STATUS,Z	-
SKPZ	Következő utasítás átugrása, ha Zero 1	BTFSS STATUS,Z	-
SUBCF f,d	Carry kivonása a fájlregiszterből	BTFSC STATUS,C DECF f,d	Z
SUBDCF f,d	Digit Carry kivonása a fájlregiszterből	BTFSC STATUS,DC DECF f,d	Z
TSTF f	Fájlregiszter tesztelése, hogy nulla e.	MOVF f,F	Z

MPASM direktívák

Az MPLAB fordítója számos a programozást megkönnyítő direktívával van ellátva – ezek közül itt csak a legfontosabbakat soroljuk fel.

Direktívák csoportosítása:

Vezérlő direktívák:	Megadhatjuk, hogy miként kerüljön lefordításra a forráskód.
Feltételes fordítás direktívái:	Megadhatjuk, hogy a kód egy része, vagy egésze bizonyos feltételek teljesülése esetén fordításra kerüljön-e vagy sem.
Adat direktívák:	Különbözőcélokra lefoglalhatunk helyet az adatmemóriában, szimbólumokat rendelhetünk az adatmemória címeihez.
Jegyzékbe vétel (listing) direktívái:	Kiválaszthatjuk, hogy milyen lista fájlok keletkezzenek a fordítás során.
Makró direktívák:	Makrók létrehozásához, beállításához és a hozzájuk tartalmazó adatmemória lefoglalásához szükséges direktívák.
Tárgy (object) fájl direktívái:	A tárgy fájl létrehozásához tartalmazó direktívák.

A direktívákat akkor alkalmazzuk, ha valóban megkönnyítik a programozást. A „divat kedvéért”, vagy csak azért, hogy bizonyítsuk a direktívák terén való jártasságunkat ne használjuk őket. Törekedjünk az átláthatóságra, és a forrás könnyű érthetőségére.

MPASM direktívák

Direktívák		
Vezérlő	**Feltételes fordítás**	**Adat**
#DEFINE	ELSE	__BADRAM
#INCLUDE	ENDIF	__BADROM
#UNDEFINE	ENDW	__CONFIG
CONSTANT	IF	__IDLOCS
END	IFDEF	__MAXRAM
EQU	IFNDEF	__MAXROM
ORG	WHILE	CBLOCK
PROCESSOR		DA
RADIX		DATA
SET		DB
VARIABLE		DE
		DT
		DW
		ENDC
		FILL
		RES

Jegyzék	**Makró**	**Tárgy fájl**
ERROR	ENDM	ACCESS_OVR
ERRORLEVEL	EXITM	BANKISEL
LIST	EXPAND	BANKSEL
MESSG	LOCAL	CODE
NOLIST	MACRO	CODE_PACK
PAGE	NOEXPAND	EXTERN
SPACE		GLOBAL
SUBTITLE		IDATA
TITLE		IDATA_ACS
		PAGESEL
		PAGESELW
		UDATA
		UDATA_ACS
		UDATA_OVR
		UDATA_SHR

Vezérlő direktívák

#DEFINE: Szöveghelyettesítő címke megadása. A *#DEFINE* direktívával beszédes neveket tudunk konstanshoz, és memóriacímekhez rendelni. A fordító a forrásprogramban a definiált neveket helyettesíti az előre megadott karakterlánccal. Általában a mikrovezérlőre csatlakoztatott perifériákat szoktuk beszédes címkével ellátni a segítségével. Megadhatunk a programokban előre meghatározott konstansokat is (BAUD; MAX_SEBESSEG; MIN_HOSSZ; stb.)

Szintaxis: #DEFINE NÉV [karakterlánc]

Példa:
```
#DEFINE LED PORTB,0      ;LED néven tudunk hivatkozni a
                         ;PORTB 0. bitjére
#DEFINE SZAM 20          ;SZAM helyére 20 fordul be
```

#INCLUDE: Kiegészítő forrásfájlt adhatunk meg. Hatására a megadott fájlban lévő forrás is fordításra kerül. Külön oda kell figyelnünk hogy az elsődleges forrásban és az *include* fájl(ok)ban megadott memóriacímek ne ütközzenek egymással. A kiegészítő forrás is tartalmazhat kiegészítő forrást (maximum 5 szintű lehet az egymásba ágyazás). Egy forrás maximum 255 *include* fájlt tartalmazhat. Segítségével átláthatóbbá tehetjük forrásprogramunkat. Az általunk már lekezelt perifériákhoz tartozó algoritmusokat célszerű *include* fájlba tenni, ezáltal későbbi fejlesztésekhez is felhasználhatjuk ezeket. A fordítás során mindig először a kiegészítő forrás(ok) kerülnek lefordításra, az eredeti forrás csak utána következik.

Szintaxis:
```
#INCLUDE kiegészítő_forrás
#INCLUDE ”kiegészítő_forrás”
#INCLUDE <kiegészítő_forrás>
```

Példa:
```
#INCLUDE P16F84.INC          ;Gyári kiegészítő forrás
                             ;megadása
#INCLUDE <D:\PIC\BOCI.INC>   ;Saját kiegészítő forrás megadása
                             ;elérési úttal
```

#UNDEFINE: Szöveghelyettesítő címke feloldása. Elképzelhető, hogy a forrás egy részénél ugyanahhoz a szimbólumhoz már egy másik karakterláncot szeretnénk rendelni – pl. különböző adatátviteli sebességek. Ebben az esetben az *#UNDEFINE* direktíva segítségével feloldhatjuk a szöveghelyettesítőt, majd a *#DEFINE* direktívával egy másik karakterláncot rendelhetünk hozzá.

Szintaxis: #UNDEFINE NÉV

Példa:
```
#DEFINE LED PORTB,0          ;LED néven tudunk hivatkozni a
                             ;PORTB 0. bitjére
#DEFINE FENY_ERO 90          ;FENY_ERO helyére 90 kerül
;programrész
#UNDEFINE LED                ;LED szimbólum feloldása
#UNDEFINE FENY_ERO           ;FENY_ERO feloldása
#DEFINE LED PORTB,1          ;LED a B port 1. bitjén
#DEFINE FENYERO 50           ;FENY_ERO új értéke 50
```

CONSTANT: Konstans értékhez rendelhetünk szimbólumot. Ezt a konstanst a későbbiek során nem tudjuk felülírni. A *#DEFINE* direktívával ellentétben itt egy szimbólumhoz érték, és nem karakterlánc rendelődik hozzá.

Szintaxis: CONSTANT szimbólum=konstans […,szimbólum=konstans]

Példa:
```
CONSTANT UT=63               ;UT szimbólum helyett a 63
                             ;konstans fog befordulni
```

END: Forráskód végét jelölő direktíva – azt ami utána áll a fordító már nem veszi figyelembe. **Figyelem, ha egy program több forrást (pl. include) tartalmaz, akkor is egy és csak egy *END* szerepelhet benne.**

Szintaxis: END

Példa:
```
;program
;END                         ;program vége
```

MPASM direktívák

EQU: Segítségével szimbólumot rendelhetünk egy konstanshoz – ez akár egy másik szimbólum is lehet. Az adatmemória regisztereit, és bitjeit szokás az *EQU* direktívával címkézni.

Szintaxis: szimbólum EQU konstans

Példa:
```
SZAM EQU 0X20          ;az adatmemória 20h címén
                       ;elhelyezkedő regiszterre ezentúl
                       ;tudunk a SZAM szimbólummal
                       ;hivatkozni
IRANYB EQU TRISB       ;A B port irányregiszterére
                       ;tudunk hivatkozni az IRANYB
                       ;szimbólum segítségével
;megjegyzés: az SFR regiszterszimbólumait nem illik átírni!
```

ORG: Meghatározhatjuk, hogy a programmemória mely címétől kezdve legyen befordítva a kód.

Szintaxis: ORG memóriacím

Példa:
```
ORG 0                  ;reszet vektor definiálása
GOTO START             ;reszet esetén ugrás start
ORG 4                  ;megszakítási vektor
RETFIE                 ;vissza a megszakításból
```

PROCESSOR: Processzor típusának meghatározása.

Szintaxis: PROCESSOR processzor_típus

Példa:
```
PROCESSOR 16F84        ;16F84 mikrovezérlő beállítása
```

RADIX: Beállíthatjuk hogy milyen számformátumot használunk a forráskódban. Megadható oktális (*OCT*), decimális (*DEC*) és hexadecimális (*HEX*) paraméter. Amennyiben nem használjuk, az alapértelmezett számformátum a hexadecimális (16). A későbbi kódolás minden számot a beállított, vagy alapértelmezett számformátum szerint értelmez a fordító – ha ettől el akarunk térni, akkor használjuk a következő szintaxist: B'10001111'; O'257'; D'100'; H'A2'. A jelölések: Binary (kettes); Octal (nyolcas); Decimal (tízes); Hexa (tizenhatos). Az *MPASM* fordítója decimálisként értelmezi azt a számot, mely előtt a '.' karakter áll.

Szintaxis: RADIX számformátum

Példa:
```
RADIX DEC                   ;az alapértelmezett
                            ;számformátum decimális
MOVLW 10                    ;10d betöltése az akkumulátorba
;programrész
MOVLW B'10001111'           ;10001111b betöltése a W-be
```

SET: Szimbólumot tudunk egy változó értékű konstanshoz rendelni. Funkcióját tekintve megegyezik az *EQU* direktívával, ezt azonban később felül tudjuk írni. Használható ciklusváltozóként.

Szintaxis: szimbólum SET konstans

Példa:
```
SET LOOP 15                 ;LOOP helyett 15 fordul be
;ciklus
SET LOOP 35                 ;LOOP helyére 35 fordul be
;ciklus
```

VARIABLE: Változók definiálására alkalmas direktíva. Fontos tisztázni, hogy ez nem futás időben történő változásra utal, arra csak az adatmemória regisztereit, vagy a programmemóriát használhatjuk. **Minden direktíva hatása már a fordítás során implementálódik a kódba!** A *VARIABLE* direktívával megadott szimbólumot a forráskódban később felülírhatjuk – ebben különbözik a *CONSTANS* szimbólumtól. Célszerűen ott lehet használni, ahol egy adott szimbólumhoz attól függően hogy a forráskód mely részén hivatkozunk rá, más konstans tartozik – pl. két soros portot használunk különböző sebességekkel, de úgy szeretnénk megírni a forrást, hogy mindegyiknél a BAUD szimbólummal tudjuk hivatkozni az adatátviteli sebességre.

Szintaxis: VARIABLE NÉV szimbólum=konstans […,szimbólum=konstans]

Példa:
```
VARIABLE BAUD=D'9600'       ;BAUD értéke 9600d
;Soros port1 inicializálása
VARIABLE BAUD=D'14400 '     ;BAUD értéke 14400d
;Soros port2 inicializálása
```

Feltételes fordítás direktívái

ELSE: A feltételes fordítás „különben” ága. Csak az *IF-ENDIF* direktívával együtt van értelme.

Szintaxis: ELSE

Példa:
```
IF      BAUD == 9600        ;ha a BAUD 9600
        MOVLW .XXX          ;akkor BRG regiszterbe
        MOVWF BRG           ;XXX betöltése
ELSE
        MOVLW .XXX          ;különben YYY betöltése
        MOVWF BRG           ;(11400 BAUD)
```

ENDIF: Feltételes fordítás végét jelző direktíva. Előtte kötelezően *IF*-nek kell állnia.

Szintaxis: ENDIF

Példa:
```
IF      SZAM>1              ;ha a SZAM szimbólum értéke
        INCF PORTB,F        ;nagyobb, mint 1, akkor
ENDIF                       ;megnöveljük PORTB értékét
```

ENDW: A *WHILE* ciklus végét jelző direktíva

Szintaxis: ENDW

Példa: l. WHILE

IF: Feltételes fordítás kezdete. Segítségével univerzális programokat tudunk írni. Az átláthatóságot azonban csökkenti. Csak működő kipróbált programok esetén tanácsos alkalmazni. Kötelezően *ENDIF*-el záródik

Szintaxis: IF kifejezés

Példa:
```
IF      SZAM>1              ;ha a SZAM szimbólum értéke
        INCF PORTB,F        ;nagyobb, mint 1, akkor
ENDIF                       ;megnöveljük PORTB értékét
```

IFDEF: Akkor fordul be, ha a megadott szimbólum definiálva van a forrásban. Tesztelésre, ill. több programváltozat esetén alkalmazható. Elképzelhető, egy perifériát egy másik alkalmazásban áthelyezünk egy másik portra. Ebben az esetben egy szimbólum definiálásával különbséget tudunk tenni a két programváltozat között. Kötelezően *ENDIF* direktívával záródik.

Szintaxis: IFDEF szimbólum

Példa:
```
#DEFINE SIM 0
IFDEF SIM
;programrész (például regiszterek feltöltése kezdőértékkel a
;szimuláció elvégzéséhez)
ENDIF
```

IFNDEF: Akkor fordul be, ha a megadott szimbólum definiálva van a forrásban. Használata megegyezik az *IFDEF* direktívával.

Szintaxis: IFNDEF

Példa: l. IFDEF

WHILE: Makrón belül használható while ciklus. Nem futási időben működik! A *WHILE* ciklusban a változó maximum 255 értékű lehet. A ciklus maximum 100 sort tartalmazhat. Használata kerülendő, mert a igen redundáns kódot eredményez!

Szintaxis: WHILE kifejezés

Példa:
```
SR      MACRO REG,N             ;jobbra léptető macro
I=0
        WHILE I<N               ;N-szer jobbra léptetünk
        RRF REG,W               ;a REG helyen megadott
        RRF REG,F               ;regiszteren
I+=1
ENDW
ENDM
```

Amennyiben sok feltételes fordítási direktívát alkalmazunk a forráskódban, az nehezen kezelhetővé, átláthatatlanná válik.

Adat direktívák

__BADRAM: A direktíva segítségével beállíthatjuk a nem használható adatmemória területet. Amennyiben a forráskód erre a RAM területre hivatkozik, „érvénytelen RAM” üzenetet kapunk a fordítótól. A gyári include fájl tartalmazza.

Szintaxis: __BADRAM kifejezés[-kifejezés] [, kifejezés[-kifejezés]]

Példa:
```
__BADRAM 0X24-0X26, 0X50      ;nem használt regiszterek
;programrész
MOVWF 0X25                    ;érvénytelen RAM veszély üzenet
                              ;a fordítótól
```

__BADROM: A direktíva segítségével beállíthatjuk a nem használható programmemória területet. Amennyiben eze(ke)n a címe(ke)n utasítás van, ugrás, vagy szubrutin hívás hivatkozik rá(juk) – a fordító „érvénytelen ROM” üzenetet ad. A gyári include fájl nem tartalmazza.

Szintaxis: __BADROM kifejezés[-kifejezés] [, kifejezés[-kifejezés]]

Példa:
```
__BADROM 0X24-0X26, 0X50      ;nem használt programmemória
;programrész
GOTO 0X25                     ;érvénytelen ROM veszély
ORG  0X50
NOP                           ;érvénytelen ROM veszély
```

__CONFIG: A mikrovezérlő konfigurációs bitjeit tudjuk beállítani. Minden típus adatlapjában szerepel a beállítható bitek neve és funkciója.

Szintaxis: __CONFIG kifejezés

Példa:
```
;16-os család esetén
__CONFIG XT_OSC & _WDT_OFF & _CP_OFF
;XT oszcillátor; wdt, kódvédelem kikapcs
;18-as család esetén
__CONFIG _CONFIG0, CP_OFF_0
__CONFIG _CONFIG3, WDT_OFF_3
;kódvédelem, wdt kikapcs
```

__IDLOCS: 4 hexa számjegyű processzorazonosítót adhatunk meg a forráskódban. 16-os család esetén programozás során, míg 18-as család esetén futásidőben is olvashatjuk. A használata előtt meg kell adnunk a processzor típusát.

Szintaxis: __IDLOCS kifejezés

Példa:
```
;16-os család esetén
__IDLOCS 0X1234                ;azonosító: 1234
;18-as család esetén
__IDLOCS _IDLOC0, 0X1          ;azonosító: 12345678
__IDLOCS _IDLOC1, 0X2
__IDLOCS _IDLOC2, 0X3
__IDLOCS _IDLOC3, 0X4
__IDLOCS _IDLOC4, 0X5
__IDLOCS _IDLOC5, 0X6
__IDLOCS _IDLOC6, 0X7
__IDLOCS _IDLOC7, 0X8
```

__MAXRAM: Segítségével megadhatjuk az adatmemória maximális címét. A gyári include fájl tartalmazza.

Szintaxis: __MAXRAM memóriacím

Példa:
```
__MAXRAM H'CF'                 ;utolsó regiszter CFh címen
```

__MAXROM: Segítségével megadhatjuk a programmemória maximális címét. A gyári include fájl nem tartalmazza.

Szintaxis: __MAXROM memóriacím

Példa:
```
__MAXROM 1FFF

SET LOOP 35                    ;LOOP helyére 35 fordul be
;ciklus
```

CBLOCK: Konstans blokkot tudunk definiálni. Kötelező lezárás: ENDC.

Szintaxis: CBLOCK kezdőcím

Példa:
```
CBLOCK 0X20                    ;LOOP helyett 15 fordul be
EGY, KETTO                     ;EGY: 0X20, KETTO: 0X21
HAROM, NEGY                    ;HAROM: 0X22, NEGY: 0X23
OT:2                           ;OT: kétbájtos regiszter
ENDC                           ;0X24-0X25
;OT alsó bájtja: OT felső bájtja: OT+1
```

MPASM direktívák

DA: Segítségével 7-bites ASCII karaktereket tudunk tárolni a programmemóriában. A memóriát 14 bitenként foglalja le a direktíva, amennyiben páratlan számú karaktert kívánunk tárolni, akkor az utolsó karakterként a 'NULL' kerül a memóriába. Figyelem, mivel az arcitektúránk 8-bites, ezért nem a tényleges ASCII kód kerül letárolásra. A kiolvasás után konverziót kell végrehajtani a helyes karakterek megkapásához.

Szintaxis: [címke] DA kifejezés [,kifejezés, …, kifejezés]

Példa:
```
DA ”abcdef”
;a programmemóriába 30E2; 31E4; 32E6 kerül
;a: 61h, b: 62h, c: 63h, d: 64h, e: 65h, f: 66h
;a: (0)1100001b, b: (0)1100010b
; letárolt kód: 11000011100010b  = 30E2h
```

DATA: Számok és szöveges információ letárolására alkalmas táblázatot generál a programmemóriában. A DA direktívához hasonlóan itt is 14 bites csomagokban tárolunk, azonban egy karakterhez rendeljük a 14 bitet. A karaktereket nem vonjuk össze – így a kiolvasott adat megegyezik a beírt értékkel, csak az üres helyeket kell eldobnunk.

Szintaxis: [címke] DATA kifejezés [,kifejezés, …, kifejezés]

Példa:
```
DATA ”abcdef”
;a programmemóriába 0061; 0062; 0063; 0064; 0065; 0066 kerül
```

DB: Adatot tárolhatunk a programmemóriában 1 bájton.

Szintaxis: [címke] DATA kifejezés [,kifejezés, …, kifejezés]

Példa:
```
DB ”ab”,'c',”def”
;letárolt értékek: 6162h, 6364h, 6566h
```

DE: Segítségével a belső adat EEPROM-ban tudunk információt tárolni. Nincs implementálva minden mikrovezérlőben. 16-os család esetén a 2100h, míg 18-asnál az F00000h-nál kezdődik az EEPROM adatmemória.

Szintaxis: [címke] DE kifejezés [,kifejezés, …, kifejezés]

Példa:
```
ORG 2100
DE ”Boci program”, '\n'
```

MPASM direktívák

DT: Adattáblázatot hozhatunk létre a programmemóriában. Soros RETLW utasítást is generál. A W értéke határozza meg, hogy melyikkel térünk vissza. Minden karaktert egy RETLW utasításhoz rendelünk. Az adatdirektívák közül ezt célszerű alkalmazni.

Szintaxis: [címke] DT kifejezés [,kifejezés, …, kifejezés]

Példa:
```
DT ”abcdef”
;ADDWF PCL,F
;RETLW 'a'
;RETLW 'b'
;RETLW 'c'
;RETLW 'd'
;RETLW 'e'
;RETLW 'f'
```

DW: Az adatot szavanként tároljuk a programmemóriában.

Szintaxis: címke DW kifejezés [,kifejezés, …, kifejezés]

Példa: DW ”abcdef”

ENDC: Konstans blokkot lezáró direktíva.

Szintaxis: ENDC

Példa: l. CBLOCK

FILL: Feltölthetjük n-szer egy értékkel a programmemóriát. 16-os család esetén szavanként, 18-as család esetén bájtonként.

Szintaxis: címke FILL kifejezés,n

Példa:
```
ORG 300
FILL (NOP),6                ;300-305: NOP utasítás
```

RES: Adott szót (18-asnál bájtot) tudunk lefoglalni egy általunk megadott szimbólumnak a programmemóriában.

Szintaxis: címke szimbólum RES memóriaegység

Példa:
```
ORG  50                     ;50: SZAM alsó bájtja
RES SZAM 2                  ;51: SZAM felső bájtja
```

Jegyzék direktívák

ERROR: Feltételek segítségével fejlesztői hibaüzenetet tudunk küldeni, melyet a fordító a fordítás során megjelenít számunkra. A szöveg minimálisan 1, maximálisan 80 karakter hosszú lehet.

Szintaxis: ERROR ”szöveges karakterlánc”

Példa:
```
IF      (BAUD!=D'1200')&&(BAUD!=D'2400')&&
        (BAUD!=D'4800')&&(BAUD!=D'9600')&&
        (BAUD!=D'11400')&&(BAUD!=D'19200')
ERROR ”Az általad választott Baud nem támogatott!”
ENDIF
```

ERRORLEVEL: Meg tudjuk adni, hogy mely üzeneteket jelenítse meg a fordító.

Szintaxis: ERRORLEVEL {0|1|2|+üzenetszám|-üzenetszám}

Példa:
```
ERRORLEVEL 0            ;üzenet, veszély és hiba
                        ;megjelenítése
ERRORLEVEL {-302}       ;bank kiválasztási üzenetet ne
                        ;jelenítse meg
;0: üzenet, veszély, hiba
;1: veszély, hiba
;2: hiba
;+üzenetszám: az üzenet megjelenítése
;-üzenetszám: az üzenet elrejtése
```

MPASM direktívák

LIST: A szövegszerkesztő beállításait és a fordítást tudjuk befolyásolni ezen direktíva segítségével.

Szintaxis: LIST opció=kifejezés [, …, opció=kifejezés]

Példa: LIST P=16F84, F=INHX8M, R=HEX

Opció	Alapértelmezett érték	Leírás
b	8	Tabulátor szélességének meghatározása (max. 999)
c	132	Oszlop szélességének meghatározása (max. 999)
f	INHX8M	A kimeneti hexa fájl formátuma (INHX8m, INHX32, INHX8S)
free	FIXED	
fixed	FIXED	
mm	ON	A LIST fájlban a memóriatérkép megjelenítése (ON, OFF)
n	60	Sorok száma egy oldalon (max. 999)
p	NONE	Processzor kiválasztása (l. PROCESSOR)
pe	NONE	Processzor kiválasztás és kiterjesztett utasításkészlet engedélyezése
r	HEX	Alapértelmezett számformátum (OCT, DEC, HEX)
st	ON	A LIST fájlban a szimbólumtábla megjelenítése (ON, OFF)
t	OFF	Csonka sorok kiírása (OFF: kiír; WRAP; feltölt)
w	0	Üzenetek szintjének beállítása (l. ERRORLEVEL)
x	ON	Makró kiterjesztések be és kikapcsolása (ON; OFF)
A táblázatban szereplő számok decimális formátumban értendők!		

MESSG: A fejlesztő által definiál üzeneteket tudunk vele megjeleníteni.

Szintaxis: MESSG ”szöveges karakterlánc”

Példa:
```
IF      (BAUD!=D'1200')&&(BAUD!=D'2400')&&
        (BAUD!=D'4800')&&(BAUD!=D'9600')&&
        (BAUD!=D'11400')&&(BAUD!=D'19200')
ERROR ”Az általad választott Baud nem támogatott!”
MESSG ”Támogatott sebességek: 1200, 2400, 4800, 9600, 11400,
19200”
ENDIF
```

MPASM direktívák

NOLIST: Segítségével kikapcsolhatjuk a kimeneti jegyzék fájlt.

Szintaxis: NOLIST

PAGE: Lapdobás beillesztése a jegyzék fájlba.

Szintaxis: PAGE

SPACE: Üres sorok beillesztése a jegyzék fájlba.

Szintaxis: SPACE n

Példa: SPACE 5 ; 5 üres sor beillesztése

SUBTITLE: Forráskódunkhoz alcímet rendelhetünk – maximálisan 60 karakter hosszúságú lehet.

Szintaxis: SUBTITLE ”szöveges karakterlánc”

Példa: SUBTITLE ”MECHA”

TITLE: Forráskódunkhoz címet rendelhetünk – maximálisan 60 karakter hosszúságú lehet.

Szintaxis: TITLE ”szöveges karakterlánc”

Példa: TITLE ”Kódzár”

Makró direktívák

ENDM: Makró végét jelző direktíva. Amennyiben a program erre fut, akkor a makró visszaadja a vezérlést a programnak.

Szintaxis: ENDM

Példa: l. MACRO

EXITM: Makróból történő feltételes visszatérésre alkalmazzuk. Előfordulhat, hogy olyan makrókat használunk, melyek paramétereinek egy adott tartományban kell elhelyezkedniük (pl. nem tudunk nullával osztani). A paraméterek ellenőrzése során nem támogatott érték esetén használjuk az EXITM direktívát. Hatása azonos az ENDM direktívával.

Szintaxis: EXITM

Példa:
```
OSZT MACRO OSZTO,OSZTANDO
IF OSZTO==0
EXITM
;oszto makró
ENDM
```

EXPAND: A makró egészének megjelenítése a jegyzék fájlban. Ez az alapértelmezett beállítás.

Szintaxis: EXPAND

LOCAL: Helyi makróváltozó deklarálása. A lokális változóra csak a makrón belül tudunk hivatkozni. Amennyiben ugyanolyan szimbólum van a főprogramban, melyet a makróban is szeretnénk használni, alkalmazzuk a LOCAL direktívát.

Szintaxis: LOCAL szimbólum [,..., szimbólum]

MPASM direktívák

MACRO: Makró definíciót lehetővé tévő direktíva. A makró definíciójának minden esetben meg kell előznie a makró meghívását.

Szintaxis: makró_neve MACRO paraméter1 [, …, paramétern]

Példa:
```
PLUS MACRO REG1, REG2      ;makró definíció
;makró kifejtése
;ENDM                      ;makró vége
```

NOEXPAND: Használata esetén a makrók kifejtése nem jelenik meg a jegyzék fájlban.

Szintaxis: NOEXPAND

A makró a szubrutinnal ellentétben a fordítónak szól, mely az ott található utasítássorozatot a makró meghívásának helyére másolja. Makrók helyett célszerű szubrutint alkalmazni a kisebb memóriahasználat okán.

Tárgy fájl direktívái

ACCESS_OVR: A 18-as család esetén alkalmazható direktíva. A közvetlen elérésű memória (ACCESS RAM) használatával megszabadulhatunk a fáradságos bankváltásoktól. Amennyiben ez a memória számunkra kevés, akkor vagy a BSR regiszter állításával másik bankba kell lépnünk, vagy – ha erre lehetőségünk van – az ACCESS adatmemória azonos címeihez különböző regisztereket rendelhetünk az overlay (átlapolás) technika alkalmazásával. Ebben az esetben a z ACCESS RAM terület felett rendelkezésünkre álló adatmemóriát virtuális memóriaként kezeljük.

Szintaxis: [címke] ACCESS_OVR RAM_cím

BANKISEL: Bankválasztó direktíva indirekt címzéshez. Az RP0, RP1 bitek mellett az IRP bitet is állítja.

Szintaxis: BANKISEL szimbólum

```
Példa:  BANKISEL REG        ;A REG regisztert tartalmazó
                            ;bankba lépés, IRP bit set/clear
```

BANKSEL: Bankválasztó direktíva közvetlen címzés esetén. Az IRP bitet nem állítja!

Szintaxis: BANKSEL szimbólum

```
Példa:  BANKSEL TRISA       ;bankválasztás az irányok
                            ;beállításához
```

CODE: A programrészt adott címre helyezhetjük a programmemóriában (16-os mikrovezérlő esetén).

Szintaxis: [címke] CODE ROM_cím

```
Példa:  RESET CODE 0        ;RESET címkére ugrás: 0-s
            GOTO START      ;címre ugrás
        START CODE 0X25     ;START programrész a 0X25-ös
                            ;címre fordul be
```

CODE_PACK: 18-as család esetén alkalmazható. Hatása megegyezik a CODE direktívával.

Szintaxis: [címke] CODE_PACK ROM_cím

Példa: l. CODE

EXTERN: Különböző asm modulok közötti átjárhatóságot teszi lehetővé. Amelyik szimbólumot globálisként definiáltuk egy másik modulban, azt az EXTERN direktívával elérhetővé tehetjük a saját modulunkban.

Szintaxis: EXTERN szimbólum1 [, ..., szimbólumn]

Példa:
```
EXTERN DELAY
CALL DELAY
```
;a delay rutin egy másik
;modulban lett megírva, de a
;GLOBAL direktívával a delay
;szimbólumot elérhetővé tettük
;más modul számára is.

GLOBAL: Szimbólumainkat elérhetővé tehetjük más asm modulok számára is.

Szintaxis: GLOBAL szimbólum [, ..., szimbólumn]

Példa:
```
GLOBAL TIME1
```
;a TIME1 szimbólumot más
;modul is elérheti, feltéve, hogy
;használja az EXTERN
;direktívát.

IDATA: Adott címtől foglalhatjuk le az adatmemóriát szimbólumaink számára. Kezdőértéket adhatunk.

Szintaxis: [címke] IDATA RAM_cím

Példa:
```
GROUP IDATA 0X20
      GROUP_VAR1 DB 1,2,3,4
```
;a 20h címtől négy hely
;lefoglalva értékek: 1,2,3,4

IDATA_ACS: 18-as család esetén alkalmazható direktíva. Az adatok kezdőcímét tudjuk definiálni az ACCESS RAM-ban. Kezdőértéket adhatunk.

Szintaxis: [címke] IDATA_ACS RAM_cím

Példa: l. IDATA

MPASM direktívák

PAGESEL: Lapválasztó direktíva. A PCLATH regisztert állítja be a megfelelő értékre. A direktíva a BSF, BSF parancsokat alkalmazza – az akkumulátor értéke nem változik.

Szintaxis: PAGESEL [címke]

Példa:
```
PAGESEL RUTINOK      ;RUTINOK címkét tartalmazó
                     ;lapra váltás
```

PAGESELW: Lapválasztó direktíva. A PCLATH regisztert állítja be a megfelelő értékre. A direktíva a MOVLW, MOVWF parancsokat alkalmazza – az akkumulátor értéke megváltozik.

Szintaxis: PAGESELW RUTINOK

Példa:
```
PAGESELW RUTINOK     ;RUTINOK címkét tartalmazó
                     ;lapra váltás, W felülírása
```

UDATA: Adott címtől foglalhatjuk le az adatmemóriát szimbólumaink számára. Kezdőértéket nem adhatunk.

Szintaxis: [címke] UDATA RAM_cím

Példa:
```
GROUP UDATA 0X20     ;a 20h címtől két hely
      GROUP_VAR1 RES 2 ;lefoglalva
```

UDATA_ACS: 18-as család esetén alkalmazható direktíva. Az adatok kezdőcímét tudjuk definiálni az ACCESS RAM-ban. Kezdőértéket nem adhatunk.

Szintaxis: [címke] UDATA_ACS RAM_cím

Példa: l. UDATA

UDATA_OVR: Overlay technikával definiálhatunk adatokat az adatmemóriában.

Szintaxis: [címke] UDATA_OVR RAM_cím

Programozás

A programozás sokak számára csupán egy kódsorozat begépelését jelenti – azonban ez egy nagyon helytelen elgondolás! A programozás hosszú folyamat, mely több lépésben jut el a tervezés fázisától a tényleges kódolásig, majd a működés ellenőrzéséig. Programot írni mindenki tud, de hatékony szellemes megoldások megalkotásához komoly tapasztalat, és rengeteg gyakorlás szükségeltetik. A fejlesztés során ne kapkodjunk, mindig adjunk magunknak – és másoknak is – időt a probléma részletes elemzésére. A hangsúlyt a tervezésre helyezzük – a gondos tervezés több időt igényel, ami azonban sokszorosan megtérül a folyamat későbbi fázisaiban.

Programfejlesztés lépései:

- Specifikáció
- Problémák definiálása, elemzése
- A leghatékonyabb megoldások kiválasztása
- Előzetes dokumentáció elkészítése
- Kódolás
- Hibakeresés, ellenőrzés (szimuláció, emuláció, funkcionális teszt)
- Teljes dokumentáció elkészítése

Specifikáció: A termék (itt, és a továbbiakban program) működési elvének leírása a megrendelő, felhasználó szemszögéből. A specifikáció általános ismérveket tartalmaz a program működéséről – de nem foglalkozik azok informatikai hátterével. Csak és kizárólag a megoldandó feladatok vannak felsorolva, a hogyan kérdésre már nekünk kell válaszolni. Egy specifikációnak mindig részletesnek és egyértelműnek kell lennie. Nem maradhatnak nyitott kérdések, félmondatok, amiket többféleképpen is lehet értelmezni.

Problémák definiálása, elemzése:
A kapott specifikáció alapján a programozó (fejlesztő) az adott problémákat ellátja informatikai jellemzőkkel. Ebben a fázisban már nem általános megfogalmazások, közérthető fogalmak, hanem algoritmusok szerepelnek. Itt kell kiderülnie annak is, ha az adott problémá(ka)t a rendelkezésre álló – szoftver és/vagy hardver – eszközzel lehetetlen kivitelezni.

A leghatékonyabb megoldások kiválasztása:
Minden problémára számos megoldás létezik – ezek költségben (ráfordított munkaóra, memóriaigény, sebesség) jelentősen eltérhetnek. A leghatékonyabb megoldás nem minden esetben a legkisebb és leggyorsabb algoritmust takarja. Először kiválasztjuk azokat a megoldásokat melyek megfelelnek a specifikációban megadott paramétereknek, majd ezek közül a számunkra legegyszerűbbet valósítjuk meg. Ebben a fázisban a programozó már – adott, de tág határok között – szabadon dönthet a folyamat későbbi irányáról.

Programozás

Előzetes dokumentáció elkészítése:

Fáradtságos – és talán a programozók számára a legkellemetlenebb – de elengedhetetlen munka. A kódolásnak csak részletes terv után kezdhetünk neki. Ez a terv ne csak a fejünkben létezzen, könnyítsük meg saját magunk számára a folyamat végrehajtását azzal, hogy részletes útmutatót készítünk. Az egyes algoritmusok, programok leírására többféle módszert is kifejlesztettek. Ezek közül nem lehet egyetemleges jót kiválasztani – a szerzőnek sem célja letenni egyik mellett sem a voksot. Mindenki a saját szemszögéből – és a probléma figyelembevételével – válasszon egy neki tetsző megoldást. Legyen az akár folyamatábra, struktogram, Jackson diagram, fa struktúra, pszeudó-kód, vagy egyéb leíró módszer.

Kódolás: A megtervezett algoritmust most egy általunk meghatározott programnyelv segítségével implementáljuk. Bármilyen programnyelvet is használjunk célszerű azt szemantikai megjegyzésekkel ellátni – ez megkönnyíti a későbbi hibakeresést, dokumentálást is.

Hibakeresés, ellenőrzés:

Compiler típusú programnyelvek esetén lehetőségünk van előzetes szintaktikai tesztelést végrehajtani, mely igen sok segítséget nyújt az alaki (nyelvtani) hibák felderítésében. A logikai hibák megtalálására több módszer is rendelkezésünkre áll. Amennyiben számkonverziót, egyik kommunikációs protokollból a másikba történő átalakítást, bonyolult aritmetikai műveleteket végzünk érdemes először szimulációs környezetben végrehajtani a tesztelést. Ne feledjük azonban, hogy a szimuláció matematikai modellekre épít, a számunkra bonyolultnak tűnő matematikai problémák elemzésében komoly segítség lehet, de a logikai hibakeresésre már nem mindig alkalmas. Mikroprocesszoros rendszerek fejlesztése során használhatunk valamilyen debugger (hibakereső), vagy emulátor eszközt is. A hibakeresés utolsó fázisa az áramkörben történő tesztelés valós körülmények között. Fontos: „ Az hogy nincs hiba, azt jelenti, hogy még nem találtuk meg azokat a körülményeket, melyek között a hiba fellép”. A tesztelés során minden elképzelhető eseményt próbáljunk meg modellezni – ne arra gondoljunk, mi hogyan használnánk az eszközt. Nincs kellemetlenebb élmény mikor egy általunk működő jelzővel ellátott eszköz az átadás során hibásodik meg.

Teljes dokumentáció:

Hangsúlyozni szeretném a teljes szót. A dokumentációnak a fejlesztés minden lényeges momentumát tartalmaznia kell. Úgy kell megalkotni, hogy abból egy másik szakember egyértelmű következtetéseket vonjon le, és reprodukálni is tudja az eszközt. A dokumentáció része a specifikáció is.

Programozás

Assembly program felépítése

FEJLÉC
INICIALIZÁLÁS
DEFINÍCIÓK
MAKRÓK
FŐPROGRAM
RUTINOK
MEGSZAKÍTÁSOK
TÁBLÁK
PROGRAM VÉGE

Programozás

Fejléc:

- Szerző
- Verziószám
- Dátum
- Ismertető

Inicializálás:

- Processzor típus
- Gyári include fájl
- Konfigurációs bitek
- Saját include fájlok

Definíciók: Szimbólumokat rendelhetünk számokhoz, és más szimbólumokhoz. Az adatmemória lefoglalására használhatjuk a CBLOCK és ENDC direktívákat, adott címre való hivatkozáshoz az EQU-t. Szimbólumok szimbólumokhoz rendelésére a #DEFINE direktíva áll rendelkezésünkre.

Makrók: A makró a szubrutinhoz hasonló gyakran ismételt kódsorozat. Különbség, hogy a makró annyiszor fordul be a tárgyprogramba, ahányszor meghívjuk (ezért nem tartalmazhat címkét). Előnye, hogy paraméterezhető – a használata azonban a memória megóvása érdekében kerülendő.

Főprogram: Itt helyezkedik el a tényleges tárgykód. A főprogram mindig egy végtelen ciklus. A processzor soha nem állhat meg – nem futhat az END direktívára.

Rutinok: Gyakran ismételt programrészek. Meghívásuk a CALL paranccsal történik visszatérés a RETURN-el. A rutin meghívásakor a programszámláló aktuális értéke elmentődik a hardveres verembe visszatéréskor az itt található cím töltődik a programszámlálóba. A verem mélysége határozza meg az egymásba ágyazható rutinok számát.

Megszakítások: Valamely belső vagy külső periféria okozhat megszakítást – ekkor a beolvasott utasítás még végrehajtódik, de a következő helyett – a PC értékének mentése után – a megszakítási vektorra ugrik a program. Visszatérés a RETFIE paranccsal történik. A megszakítás generálásának időpillanatáról általában nincsenek előzetes információink, ezért a közösen használt regiszterek értékét el kell mentenünk (W, STATUS). PIC mikrovezérlők esetén a megszakításjelző flag bitet nekünk kell szoftveres úton törölni. Itt szeretném felhívni a figyelmet, **ha a főprogramban bankváltás történik és engedélyezve van a megszakítás, akkor a megszakítási rutinban ellenőriznünk kell az aktuális bankot – szükség esetén kiválasztani a megfelelő bankot, majd a visszatérés előtt visszaállítani az eredeti értékre.**

Program vége: Az assembly programot kötelező lezárni – erre a célra használjuk az END direktívát.