**1. Generikus mikrovezérlő (PIC16) utasításkészlete. Byte orientált utasítások, bit orientált utasítások, műveletek konstanssal, vezérlő utasítások.**

**Generikus mikrovezérlő (PIC16) utasításkészlete:**

Egy 8 bites generikus mikrovezérlő (egy mikrovezérlő modell, amely a kis és közepes bonyolultságú elveket tükröz és ezek szerkezeti elemeit tartalmazza) utasításkészlete **RISC típusú (csökkentett bonyolultságú - Reduced Instruction Set Computer)** utasításokat tartalmaz.

Jellemző ezekre, hogy két részből épül fel:

1. **utasítás kódja**: ami arra utal, hogy mit kell elvégezni
2. **operandus**: ami általában az adatmemória egy regisztere, ami arra utal, hogy mivel kell elvégezni az utasítás műveletét.

Pl. Utasítás: MOV Reg, A (akkumulátor)

**Byte orientált utasítás:**

A két regiszteres utasítások esetén a második regiszternek kitüntetett (fix) címe van, ezt nevezzük **akkumulátornak** és A-val jelöljük. Ezek byte orientált utasítások, mert byte-ok közötti műveletek elvégzését vezérlik.

A műveletek lehetnek:

* *Logikai*:pl. AND Reg, A => Reg = Reg x A
* *Aritmetikai*: pl. ADD Reg, A => Reg = Reg +A
* *Regiszter eltoló*: pl. MOV Reg, A => A = Reg

**Bit orientált utasítások:**

Egy adott adatmemória regiszter adott bitjére vonatkoznak és ez a két információ az utasítás operandusában kell szerepeljen.

Két típusa van:

* *a bit beállító utasítások* (set vagy clear): pl. BC Reg, bit
* *a feltételes (bit értéktől függő) utasítások:* pl. RR Reg (rotate right)

**Műveletek konstanssal:**

Ha az utasítás készlet tartalmazza az L (Literal) indexet, akkor az operandus nem egy adott regiszter címe, hanem egy konstans. Az utasítás műveletét ebben az esetben az akkumulátor és a konstans között kell elvégezni.

Pl. MOVL A, bit

Pl. ADDL A, bit

**Vezérlő utasítások:**

Vannak olyan utasítások, amelyek nem tartalmaznak regiszter típusú operandusokat, hanem a **mikrovezérlő egy adott állapotát koordinálják**. Ilyenek az ugró utasítások, a szubrutin hívó és a visszatérő (szubrutinból, vagy megszakításból) utasítások, valamint az állapot szinkronizáló utasítások (pl. watchdog, meg a stanby mód).

Pl. JMP cím

Pl. CLRWDT: clear watchdog timer

Pl. RETi: return from interrupt

**2. Generikus mikrovezérlő memória szervezése. Programmemória szerkezete, adatmemória felépítése, speciális funkciójú regiszterek (SFR).**

Egy 8 bites generikus mikrovezérlő memória szervezése Harvard típusú. Ez azt jelenti, hogy a program és adatmemória (RAM) különálló egységeket képeznek és a szerkezetük sem egyforma.

A **programmemória** EEPROM típusú (elektromosan újraprogramozható, lassan írható, gyorsan olvasható, de a tápfeszültség lekapcsolásakor nem veszti el a tartalmát) és 16 bit szóhosszúságú regiszterekből épül fel. Minden regiszter egy-egy gépi kódú utasítást tartalmazhat. A programot a 0000H címtől kezdődően kell beírni a programmemóriába, mivel a mikrovezérlő alapállapotba tevésekor (Reset) mindig ez címeződik meg. A programmemória címzése a programszámlálóval (PC) történik. A megcímzett memóriarekesz tartalma az utasításregiszterbe (IREG) íródik át, hogy a teljes utasítás ciklus alatt az utasítás dekóder (IDEC) rendelkezésére tudjon állni.

Az **adatmemória** RAM típusú (nagy sebességgel írható és olvasható, de a tápfeszültség lekapcsolásakor elveszti a tartalmát) és 8 bit hosszúságú regiszterekből épül fel, ez határozza meg az adat szélességét. Az adatbusza az aritmetikai-logikai egységgel (ALU) kommunikál, a címbuszra pedig az utasításdekóder küldi az utasításokban szereplő operandus címét. Mivel a generikus mikrovezérlő esetén az operandus címe 8 bites, ezért egy adatmemória lap 2exp8=256 byte-ot tartalmaz és lapozással lehet a memória kapacitását növelni.

Az **adatmemória** kétféle regisztert tartalmaz:

* *általános célú* regisztereket (adat regiszterek)
* *speciális célú* regisztereket (SFR). Az SFR regiszterek hardveresen vannak huzalozva a mikrovezérlő szerkezeti elemeihez és periféria moduljaihoz, és az a szerepük, hogy az utasítások segítségével (szoftveresen) lehessen beállítani és követni a hardver elemeket. A SFR regiszterek vezérlő és állapot biteket tartalmaznak.

**3. Generikus mikrovezérlő regiszter címzése. Direkt címzés, indirekt címzés, lapválasztás, táblakezelés, veremkezelés.**

Egy 8 bites generikus mikrovezérlő a gépi kódú utasításokkal éri el az adat regisztereit. Az utasítások operandusa rendszerint egy adatmemória regiszter, ennek a tartalmával történik az utasításban szereplő feladat elvégzése. Az operandusok címzése kétféleképpen történhet:

**Direkt címzés** esetén az utasítás tartalmazza az operandus regiszter abszolút címét (hányas regiszteren van). Mivel az utasítás az operandus címét 8 biten kódolja, összesen 2exp8=256 byte-ot lehet megcímezni, tehát egy adott memórialap regisztereit.

Pl. MOV AX, 06F2h - AX-et belerakom a 06F2h címen található regiszterbe

**Indirekt címzés** esetén az utasítás nem tartalmazza az operandus regiszter abszolút címét, ez a cím az FSR-ben van benne. Az az előnye, hogy iterációkat (loop) tudunk csinálni.

Pl. Ha nem adom meg a direkt címet, akkor többször meghívható

**A lapválasztás** esetén kevés memória van, csupán 8 bit, amivel 256 byteos lapokat tudunk megcímezni. A státusz regiszterben vannak a lapválasztó bitek, amelyekkel az adatmemórialapjainak a dekóderét címezzük meg, ezáltal mindig más lapot választunk.

**A táblakezelés** RETLA utasításokon alapszik, amelyek az akkumulátorba betesznek egy konstantst. Táblázatokat tudunk beírni a programmemóriába, az utasításoknak értékeket adunk, ezen értékeket tesszük a táblázatba. Visszatéréskor az akkuba beírt utasítással térünk vissza.

**A** **veremtár** egy speciális adatmemória (tehát RAM), ahova a szubrutin, illetve a megszakítás hívásakor a soron következő utasítás címét elmentjük. Ez egy LIFO (Last In First Out) típusuk memória, mivel a beágyazott hívások esetén mindig a legutoljára betett memóriacím hívódik elő először, a RET típusú utasítás hatására. A generikus mikrovezérlő veremtára 8 byte mélységű és egy számláló (SP – Stack Pointer) címezi meg, amit az utasításdekóder vezérel. Ebből következik, hogy 8 réteg mélységben ágyazhatunk be ismételt meghívásokat.

**4.  Generikus mikrovezérlők megszakításrendszere. Megszakításvektorok, prioritások, vezérlő regiszterek, időbeli viszonyok.**

Egy 8 bites generikus mikrovezérlő **megszakítás rendszere** az utasítás dekóder köré van szervezve, hiszen a megszakítás aszinkron folyamat, ezért a vezérlő hardver kell szabályozza.

Minden perifériának van egy Interrupt Enable Bit-je (az SFR-ben van), ami egy engedélyező mechanizmus. Ezt a bitet szoftverből állítjuk be a GIE bittel együtt megszakításkor (mindkettőt 1-re állítjuk be).

**Megszakítás kéréskor** a feldolgozás alatt lévő utasítás még végrehajtódik, a következő utasítás címe a verembe kerül. Ezután a program átugrik egy speciális címre, a **megszakítás vektorra** ekkor a GIE (General Interrupt Enable) 0 lesz, hogy más ne kérhessen megszakítást. Ezen a címen levő utasításokat hajtja végre, amíg nem jön egy RETi utasítás.

A generikus mikrovezérlő egy megszakításvektorral rendelkezik, de vannak olyan mikrovezérlők, amelyekbe két vagy több megszakításvektort is beágyaztak. Ebben az esetben szintén egy SFR regiszterrel lehet beállítani, hogy melyik megszakításvektornak legyen nagyobb **prioritása** (ha két megszakítás érkezik egyszerre, melyik legyen hamarabb kezelve). A prioritás bitek beállítása hasonló módon történik, mint az engedélyező bitek beállítása.

**Időbeli viszonyok:** az elvégzést alatt álló utasítást még el lehet végezni. A megszakítás késhet egy utasításnyi időt (0-tól 2 ciklusig), nem egyforma a megszakítás kezelése (változhat).

**5. Generikus mikrovezérlők programozása. Tokok programozása, programozói állapot, konfigurációs bitek, hardverkörnyezet, áramkörben történő soros programozás (ICSP).**

Egy 8 bites generikus **mikrovezérlő programozásán** azt a folyamatot értjük, amely során az assembler nyelven megírt program átfordítódik gépi, bináris kóddá és ez átíródik a fejlesztő platformról a mikrovezérlő program memóriájába.

A **tokok (chip) programozása:**

* Az integrált fejlesztői platformon írjuk meg a programot.
* A szövegszerkesztőben megírt kódot lefordítja gépi kódra.
* A gépi kódot a programozó modul az USB porton keresztül tölti b e a programmemóriába.
* A mikrovezérlő felé öt szállal kapcsolódik, ezek a tápfeszültség, földelés stb.

A **programozói állapot:**

* A vezérlő bittel tesszük programozói állapotba a tokot. (Pl meg kell nyomni a gombot pl BOOT) Ez az a folyamat, amikor tokot éppen felprogramozzuk.
* A RESET-et 0-ról áttesszük 1-re, vagy egy nagyobb feszültséget teszünk a mikrovezérlőre.

A **konfigurációs bitek:**

* Olyan állapotbitek, amelyek a program teljes futása alatt nem módosulnak, az EEPROM-ba írjuk.
* A legfontosabb konfigurációs bitek az órajel generátor működését beállító bitek, a Reset áramkör működését meghatározó bitek, a Watchdog működését engedélyező bit és a programmemória adatvédelmét korlátozó bitek.

Pl. A programmemória legyen-e védett, a watchdogot használjuk-e

**Hardverkörnyezet:**

* PC => USB => Programozó modul => 5 vezérlőjel => tok (chip)

**6. Generikus mikrovezérlők speciális regiszterei (SFR). Hardverkörnyezet, STATUS regiszter, WDT (Watch Dog Timer) regiszter, FSR (File Select Regiszter), PCL/PCLATH regiszterek.**

Egy 8 bites generikus mikrovezérlő **speciális funkciójú regiszterein** (SFR) értjük azokat az adatmemória regisztereket, amelyek hardveresen huzalozva vannak a mikrovezérlő egyes áramköreihez. A szerepük, hogy az utasítások segítségével (szoftveresen) lehessen beállítani és követni a hardver elemeket. A SFR regiszterek vezérlő és állapot biteket tartalmaznak.

A SFR regisztereket az általános célú regiszterekhez hasonlóan szoftveresen lehet írni és olvasni a mikrovezérlő gépi kódú utasításai segítségével. Rendszerint a megfelelő vezérlő biteket írjuk (pl. a BS vagy BC utasításokkal) illetve a megfelelő állapot biteket olvassuk (pl. a BTSS vagy BTSC utasításokkal) és segítségükkel tudjuk beállítani egy-egy periféria modul helyes működését, illetve meg tudjuk állapítani, hogy éppen milyen állapot fázisban vannak.

A **legfontosabb SFR regiszterek** a STAUS, WDT, FSR és PC regiszterek.

A **STATUS** regiszter az utasításdekóderhez (IDEC) kapcsolódó regiszter, többnyire a jelző (flag) biteket és a lapválasztó biteket tartalmazza. A jelző bitek közül a legfontosabbak a Z és a CY flagek, amelyek akkor jeleznek (1-esre állnak). A lapválasztó bitek (P0, P1, P2) segítségével az adatmemória aktuális lapját lehet beállítani. Minden adatmemória lap 256byte kapacitású, hogy 8 bittel lehessen megcímezni egy-egy regisztert.

A **WDT** regiszter a watchdog áramkör időzítését végzi. Ebbe a regiszterbe beírt konstans határozza meg, hogy a watchdog áramkör inicializálása után (a CLRWDT utasítással) mennyi idő kell eltelnie amíg az áramkör nem generál egy Reset jelet, amellyel a mikrovezérlőt alapállapotba hozhatja. Ahhoz, hogy ez ne történjen meg, a programba megfelelő idöközönként egy-egy CLRWDT utasítást kell elhelyezni, ami újraindítja a watchdog számlálóját.

A **FSR** regiszter az adatmemória indirekt címzést szolgálja. Indirekt címzés esetén az utasítás nem tartalmazza az operandus regiszter abszolút címét, hanem csak egy utalást (IND) arra, hogy az utasításdekóder hol keresse azt. A mikrovezérlő hardverstruktúrája úgy van megtervezve, hogy az operandus címét egy speciális funkciójú regiszter, a FSR (File Select Register) regiszter tartalmazza. Ezért, mielőtt egy indirekt címzésű utasítást használunk, előbb fel kell töltenünk a FSR regiszter tartalmát az operandus megfelelő címével.

A **PC** (Program Counter) regisztereket a programmemória címzésére használjuk. Két 8 bites SFR regiszterre van szükségünk, hogy a tartalmát írni és olvasni tudjuk. Ezek a PCL és a PCLATH nevű regiszterek, melyek a programszámláló alsó, illetve felső 8 bitjét tartalmazzák. Ezek mind a 16 bit egyszerre íródik be a programszámlálóba.

**7. Az assembler nyelv szintaktikája. Címkemező, utasításmező, operandusmező, megjegyzésmező, direktívák. A gépi kódra fordítás lépései.**

Mikrovezérlők esetén, a gépi kódú programokat **assembler** nyelven írjuk meg, a fejlesztői platform szövegszerkesztőjében. Egy assembler nyelven írt utasítás négy mezőben helyezkedik el a szövegszerkesztő egy sorában. Ezek a **címke, utasítás kód, operandus és komment mezők.**

Jellemző ezekre, hogy két részből épül fel:

1. **utasítás kódja**: ami arra utal, hogy mit kell elvégezni
2. **operandus**: ami általában az adatmemória egy regisztere, ami arra utal, hogy mivel kell elvégezni az utasítás műveletét.
3. **címkét** az nem mindig használjuk, csak ugró utasítások esetén. Kettőspont van utána és újrahívható.
4. **Komment** azt jelzi, hogy mit ír le az adott gépi kódú utasítás. Pontosvesszővel kezdődik.

A **direktívák** azok nem végrehajtható utasítások és az assambler nyelvet segítik a kompilálásban. A legfontosabb direktívák:

* EQU - A címkének adunk egy abszolút címet. (abszolút cím: nem változó cím)
* END - vége van az utasításoknak
* CONFIG - konfigurációs biteket állítunk be

Az assemblerben írt gépi kódú program bináris kódra való fordítása (**kompilálása**) két lépésben történik.

1. Az első átfutáskor az assembler megoldja a címkék átfordítását abszolút címekké, valamint ellenőrzi a szintaktikai (alaki) és szemantikai (értelmezésbeli) hibákat, mint pl.: téves mnemonik jelölés, operandus hiánya, operandus helytelen használata, akkumulátor helytelen használata, címke hiánya vagy téves jelölése. Ezeket hibaüzenetek, vagy figyelmeztetések formájában meg is jeleníti.
2. Ha nem talál hibát, akkor a második átfutáskor elkészíti a fordítási listát (az utasítások kód és cím megfeleltetését, valamint a címkék táblázatát a hozzárendelt abszolút értékkel együtt) és a forrásprogramot (src), valamint megjelenít egy statisztikát a lefordított programról.

**8. Generikus mikrovezérlő időzítő perifériái. Hardverkörnyezet, számláló üzemmód, időzítő üzemmód, kiolvasás/összehasonlítás (Capture/Compare) üzemmód, PWM moduláció.**

Egy 8 bites generikus mikrovezérlő egyik leggyakrabban használt periféria modulja az időzítő (**Timer**) modul. A használata elősegíti a legtöbb időzítő, számlál, impulzus generáló funkciókat hardveresen oldjuk meg, vagyis incializáljuk a modul működését, ezt követően ez a mikrovezérlő utasításai nélkül képes lesz megoldani a problémát.

Egy **Timer modul 3 fő egységre** oszlik:

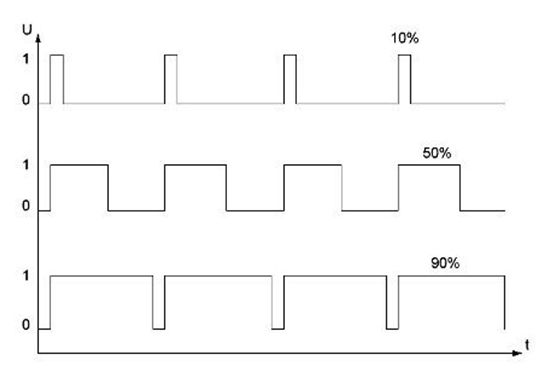
* bementi kontrol áramkör,
* soros bináris számlálók,
* kimeneti (megszakítást kontroláló) áramkör.

A generikus mikrovezérlő számláló része három sorbakapcsolt 8 bites számlálót tartalmaz, vagyis egy 24 bites bináris számlálót alkot, amelyik 000000H-tól FFFFFFH-ig tud számolni, felfele, vagy lefele.

Egy **Timer modulnak 3 fő üzemmódja** van:

* Számláló üzemmód:
  + Timer modul bemenetét valamelyik PORT lábra kötjük
  + A belső oszcillátorra kötjük, időt fog mérni
* Capture: A számláló tartalmát egyszerre menti el a lementő regiszter
* Compare: A Capture tartalmát összehasonlítja a Timer tartalmával.

Impulzus szélesség modulációs üzemmód (PWM) az az, amikor egy jel esetében a szélességet moduláljuk, tehát csökkentjük, vagy növeljük. A kitöltési tényező az, ami alapján modulálunk.



**9. Generikus mikrovezérlő kommunikációs perifériái. Hardverkörnyezet, I/O portok, párhuzamos portok, szinkron soros portok, aszinkron soros portok. Az üzenetek szerkezete.**

Egy 8 bites generikus mikrovezérlő rendelkezik számos kommunikációs perifériával, amelyeken keresztül tartja a kapcsolatot a külvilággal. Ezek kétféle típusúak: **párhuzamos** vagy **soros** adatátviteli egységek, aszerint, hogy az adatok átvitele egyszerre történik 8 biten, vagy ütemesen, bitenként viszik át egyetlen adat vonalon. Minden mikrovezérlő rendelkezik legalább egy hardveres párhuzamos és egy hardveres soros kommunikációs megoldással.

A mikrovezérlők alapvető kommunikációs eszköze a párhuzamos **I/O port**. Ez 8 bit hosszúságú és egy mikrovezérlő egyszerre több porttal is rendelkezhet (PortA, PortB, PortC stb.). Ezeken keresztül tartja a kapcsolatot a külvilággal. Minden port hardveresen kapcsolódik két, speciális funkciójú (SFR) regiszterhez. Az egyik (DIRx) meghatározza a port bitjeinek az irányát, a másik (PORTx) az értékét.

Az **aszinkron soros** protokollok általában legalább egy START bitet használnak a kommunikáció indítására és időzítést használnak az adatbitek küldésére. Minden bit egy periódusnyi időre marad az adatbuszon. A vevő modul a START bittel szinkronizálja magát, és ugyanazt a periódusú időzítést generálja saját magának, mint az adó periódusa. A vevő az időzítés fázisát eltolja fél periódussal.

A **szinkron soros** protokollok az órajel frontjait használják az adatbitek szinkronizálására. Általában az adó modul az órajel felfutó élein teszi ki a következő adatbitet az adatvonalra. A vevő az adatbitek beolvasását az órajel lemenő élein fogja elvégezni. Mivel az órajel közös, valamint a kitöltési tényező 50% biztosítva van az olvasás félperiódusos eltolása, vagyis az olvasás mindig az adatbit vonalidejének a közepén történik.

**10. Mikrokontrolleres fejlesztés lépései és eszközei. Integrált fejlesztő környezetek (IDE), a programfejlesztés lépései, programozási alapelvek: címkiosztás, változók használata, paraméterátadás, feladat (task) kezelés, watchdog használata.**

A mikrovezérlős alkalmazások fejlesztése rendszerint egy kompatibilis **integrált fejlesztői platformon** történik (IDE – Integrated Development Environment).

A fejlesztéshez szükséges modulok:

* a **Szövegszerkesztő** (Editor): ahol a programot írjuk meg és szerkesztjük szöveges formában;
* a **Fordító** (Compilator): amelyik az assembler nyelven (vagy más magasabb szintű nyelven) megírt programot átkonvertálja bináris kódokká;
* a **Szimulátor**: amelyik a lefordított programot off-line futtatja akár lépésenként és kijelzi a különböző regiszterek pillanatnyi értékeit;
* az **Emulátor** (opcionális), amivel a programot on-line lehet futtatni akár az alkalmazás áramköri környezetében;
* a **Programátor**, amivel a lefordított gépi kódú programot fel lehet tölteni a mikrovezérlő programmemóriájába.

**A programfejlesztés lépései:**

1 – A feladat elemzése

2 – Az alkalmazás megtervezése

3 – Folyamatábra (Organigram) elkészítése

4 – Kódolás

5 – Fordítás, ellenőrzés, szimulálás (debugolás)

6 – Betöltés, felprogramozás

7 – Futtatás

8 – Korrekciók

9 – Dokumentálás

A **címkiosztás** az EQU direktívával történik és minden olyan változónak meg konstansnak adunk az elnevezésére jellemző nevű címkét, amelynek abszolút címét vagy értékét a programban használjuk.

A főprogram változóit feladat (task) kezelésre is használhatjuk (pl. a főprogramban változtatjuk a változó értékét, a megszakításrutin vagy valamelyik szubrutin pedig az adott értéknek megfelelő feladatot fogja elvégezni).

Fordításkor (5. lépésnél): Az assemblerben írt gépi kódú program bináris kódra való fordítása (kompilálása) két lépésben történik. Az első átfutáskor az assembler megoldja a címkék átfordítását abszolút címekké, valamint ellenőrzi a szintaktikai (alaki) és szemantikai (értelmezésbeli) hibákat, mint pl.: téves mnemonik jelölés, operandus hiánya, operandus helytelen használata, akkumulátor helytelen használata, címke hiánya vagy téves jelölése. Ezeket hibaüzenetek, vagy figyelmeztetések formájában meg is jeleníti. Ha nem talál hibát, akkor a második átfutáskor elkészíti a fordítási listát (az utasítások kód és cím megfeleltetését, valamint a címkék táblázatát a hozzárendelt abszolút értékkel együtt) és a forrásprogramot (src), valamint megjelenít egy statisztikát a lefordított programról.