PROGRAMAREA ÎN LIMBAJ DE ASAMBLARE

GENGE BÉLA

Capitolul 1
Introducere

Limbajul de asamblare

- Reprezintă forma simbolică a limbajului mașină (codul binar, cod-mașină).
- Instrucțiuni sub forma unor șiruri de caractere inteligibile de operatorul uman și care sunt ușor translatabile în cod binar (mașină), i.e., secvențe de 0/1.
- Asigură acces direct la resursele fizice ale calculatorului (memorie, placa video, întreruperi).
- Indiferent de limbajul de programare, aplicațiile sunt translatate într-un limbaj de asamblare.

Ierarhia limbajelor de programare

Limbaje de nivel înalt:
C++, PHP, Java

Limbajul de asamblare

```
      GOTO Main

Setare_Port:
      MOVLW 0x7F            ;0x7F->WREG
      MOVWF TRISB           ;WREG->TRISB
      RETURN

Main:
      MOVLW 0x55            ;0x55->WREG    (pas 1)
      CALL Setare_Port      ;0x7F->WREG !!!
      MOVWF 0x20            ;WREG->F[20h]  (pas 2)
```

Limbajul mașină

```
00101110011111110110001011010111001011000
01001000101011011101100001001111010000001
00111111000101010111000001001110011111000
01100100111100011110100011011000001011100
11110011001010100011100000010100111110000
10100010110000110010111111011001011100011
11010110110111111111110000000110000111000
10100110011010010110100111001110101001010
00111010010100111111100000101101101011100
01011111111101110110001101101011000101101
01000011101001101111001100110001011110110
10110100001100010000010101011101001101100
00101000101000101111101111111100011011010
00100101000100011110110100011000110111111
```

Hardware

De ce se evită LA? (prejudecăți)

- Slide preluat din “Programare în limbaj de asamblare, Prof. Sebestyen Gheorghe, UTCN”.

este prea greu	Orice lucru nou este greu – la inceput
este greu de citit si de inteles	Comentariile pot imbunatati ac. lucru
este greu de scris	Da, …. pentru incepatori
este greu de depanat si de intretinut	Da
programarea este ineficienta	Da, se scriu mai multe linii de cod

De ce se evită LA? (prejudecăți)

- Slide preluat din “Programare în limbaj de asamblare, Prof. Sebestyen Gheorghe, UTCN”.

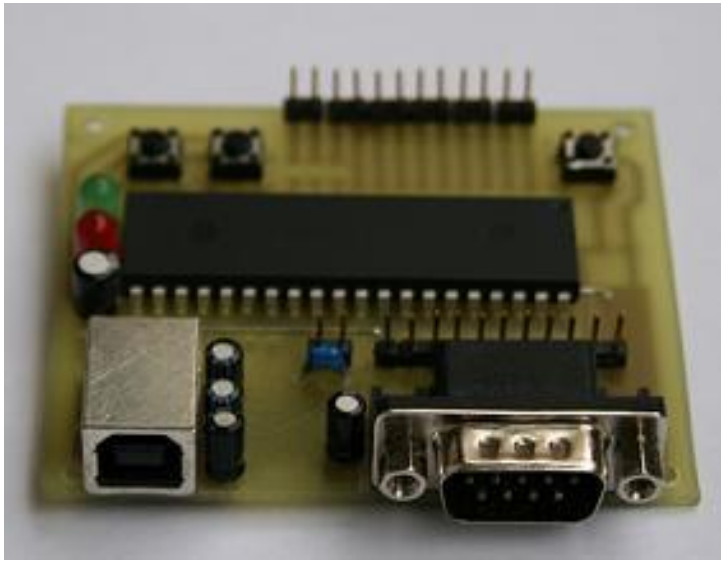
Viteza nu mai constituie o problema	Sunt aplicatii (ex. procesare de imagini) la care conteaza
Memoria nu mai constituie o problema	Exceptie fac sistemele incapsulate (enbedded), de control, microcontroloare,etc.
Compilatoarele actuale genereaza cod eficient	Niciodata mai bun decat programul scris in limbaj de asamblare
Limbajul de asamblare nu este portabil	Este portabil pe toate calculatoarele care au acelasi tip de procesor (ex. PC-uri)

De ce studiem LA?

- Viteza: scrierea programelor în limbaj de asamblare va duce la aplicații rapide.
- Dimensiunea programelor: programele scrise în LA ocupă cea mai puțină memorie.
- Oferă o paradigmă diferită de programare față de limbajele uzuale.
- Deschide posibilitatea programării sistemelor încorporate (embedded).

Conținutul cursului

- Arhitectura sistemelor dedicate și încorporate.
- Formatul instrucțiunilor.
- Organizarea și adresarea memoriei.
- Structuri de program.
- Porturi de intrare/ieșire.
- Temporizări:
  - Temporizări software.
  - Temporizări hardware.
- Întreruperi.
- Convertorul analog-digital.
- Comunicații seriale.



Bibliografie

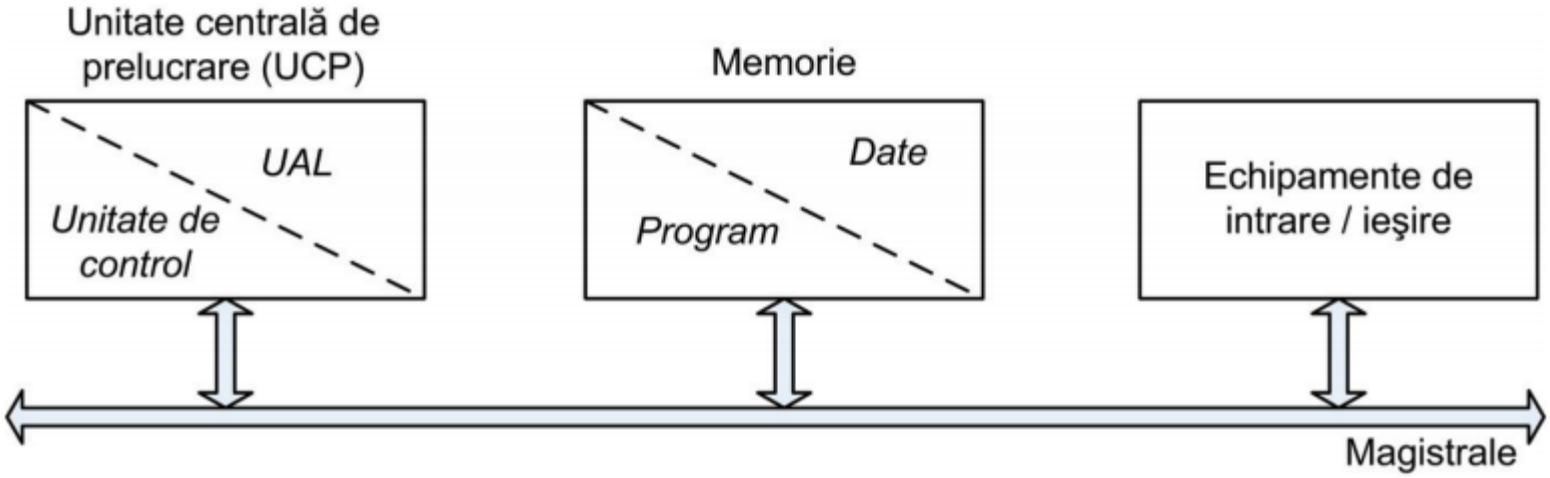
- Genge Bela, Haller Piroska: Proiectarea sistemelor dedicate și încorporate cu microcontrolerul PIC. Ed. Univ. Petru Maior, 2008.
- Adrian-Vasile Duka, Genge Bela, Haller Piroska: Sisteme cu microprocesoare. Microcontrolerul PIC18F4455. Ed. Univ. Petru Maior, 2013.
- Ambele cărți sunt (temporar) disponibile aici:
  - www.ibs.ro/~bela/Carti.zip
- www.intel.com – pentru procesoare Intel (x86,x64)
- www.microchip.com – pentru procesoare din familia PIC
- http://www.ibs.ro/~bela/Teachings/Microprocessors/microprocessors.html

Administrativ

- Conf. dr.ing. Béla GENGE (titular curs, laborator).
- Adresa email: bela.genge@ing.upm.ro.
- Examen: scris (50% nota finală).
- Laborator: evaluări pe parcurs (+ proiect final).
- Condiție prezentare la examen: min. nota 5 la laborator

Arhitecturi de sisteme

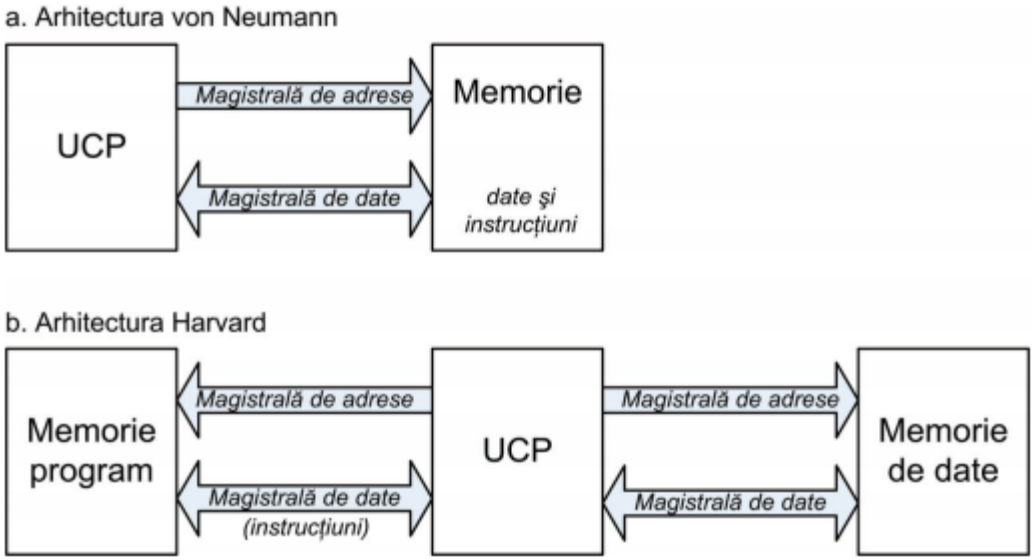
- Istoric, sistemele de calcul moderne își au originea în cel de-al doilea război mondial.
  - Funcționalitate dedicată, proiectate pentru o singură sarcină dedicată: Ex: Mașina lui Turing pentru spargerea criptografiei mașiniii Enigma.
  - Schimbarea funcționalității se realizează prin schimbarea circuitelor electrice.
- Neumann János Lajos (John von Neumann):
  - Arhitectul sistemlor moderne în care instrucțiunile și datele sunt stocate



Arhitecturi de sisteme

- Arhitectura Harvard.
  - Denumită după arhitectura calculatorului Harvard Mark 1 dezvoltat la Universitatea Harvard.

- Comparație arhitecturi:



Unitatea Centrală de Prelucrare

- UCP (en: CPU – Central Processing Unit).
  - Execută o secvență de instrucțiuni stocată în memorie.
- Ciclul de instrucțiune:
  - Extragerea instrucțiunilor într-un registru intern.
  - Decodificarea instrucțiunilor.
  - Execuția instrucțiunilor.
- Extragerea succesivă a instrucțiunilor se realizează sub controlul unui circuit:
  - **Program Counter:** stochează adresa următoarei instrucțiuni din memorie ce urmează a fi extrasă.
- Structura:
  - Unitate de control.
  - Unitate aritmetică și logică.
  - Regiștrii interni – memoria internă a UCP.

Magistralele

- Reprezintă căile de comunicare.
- Magistrala de date:
  - Asigură transferul de date pentru UCP.
  - Dimensiunea magistralelor: 8 biți, 16 biți, 32 biți, 64 biți.
  - Execuția instrucțiunilor.
- Magistrala de adrese:
  - Asigură adresarea locațiilor de memorie.
- Magistrala de control:
  - Conduce semnalele de control ale UCP către memorie și periferice.
  - Ex.: semnale citire/scriere, tactul sistem, linii de întrerupere, linii de stare.

Memoria, echipamente I/O

- Memoria e caracterizată prin:
  - Conținutul pe care îl stochează.
  - Locația (adresa) fiecărui registru.
- Echipamentele de Intrare/Ieșire:
  - Asigură comunicarea cu exteriorul.
  - Comunicarea se realizează prin magistrale, adrese de memorie, porturi.

Arhitecturi RISC

- Din punct de vedere a setului de instrucțiuni sistemele de calcul pot fi clasificate:
  - Arhitecturi CISC: Complex Instruction Set Computer.
  - Arhitecturi RISC: Reduced Instruction Set Computer.
- Arhitecturi CISC: set complex de instrucțiuni, fiecare instrucțiune implementează funcții apropiate limbajelor de nivel înalt.
  - Cod de dimensiuni reduse, set complex de instrucțiuni.
  - Timpul de execuție al unei instrucțiuni mult mai mare decât la RISC.
- Arhitecturi RISC: set redus de instrucțiuni, fiecare asigurând operații de bază.
  - Cod de dimensiune mare, set redus de instrucțiuni.
  - Timp de execuție redus.

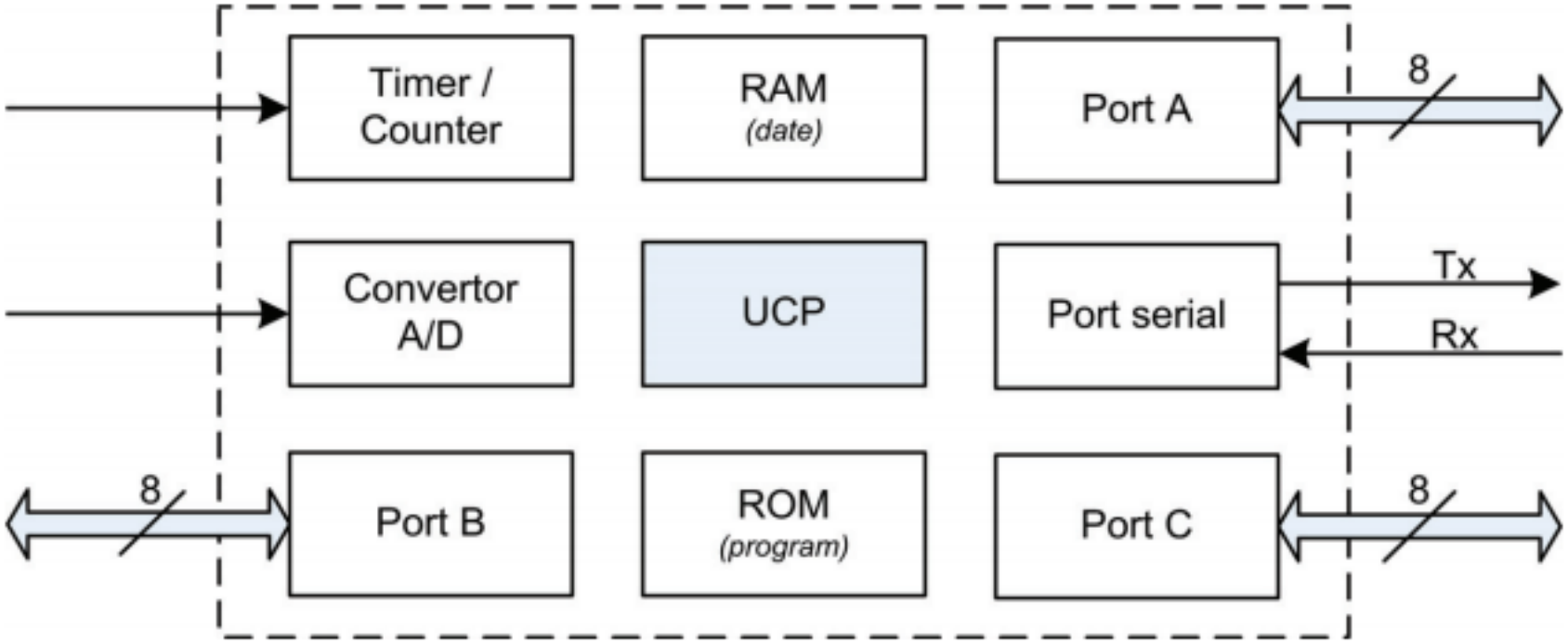
Arhitectura pe care o vom studia

- Vom studia arhitectura microcontrolerelor PIC.
  - Arhitectura Harvard, RISC.
- Ce este un microcontroler:
  - Este un microsistem ce conține toate cele trei componente principale (UCP, memorie, echipamente I/O) într-un singur circuit integrat.
  - Circuitul include toate componentele necesare funcționării.
- Caracteristici microcontrolere:
  - Set redus de instrucțiuni.
  - Dimensiune memorie redusă (KB/MB).
  - Capacitate de adresare redusă.
  - Consum redus de energie.
  - Preț redus.

Microcontrolerele PIC

- PIC: Programmable Intelligent Controller.
- Fabricate de Microchip Technology.
- Succes foarte mare datorat suportului oferit de producător dar și al mediului de dezvoltare în limbaj de asamblare și C.
- PIC 18:
  - Familie de uC (microcontrolere) cu instrucțiuni pe 16 biți, magistrală internă pe 8 biți, memorie program de tip Flash, memorie de date adresabil liniar, capacitate de până la 2MB.
  - Număr mare de protocoale de comunicații: CAN, Ethernet, USB, SPI, I2C, etc.

Structura generală



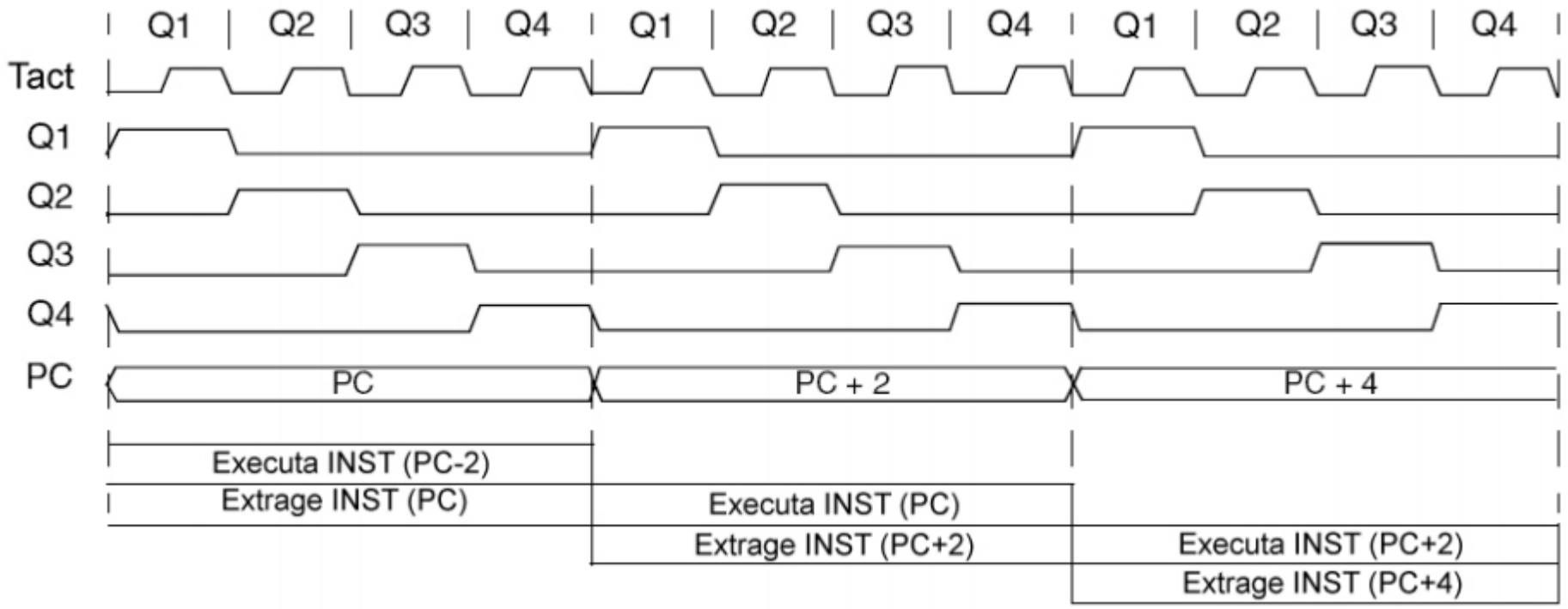
Execuția instrucțiunilor

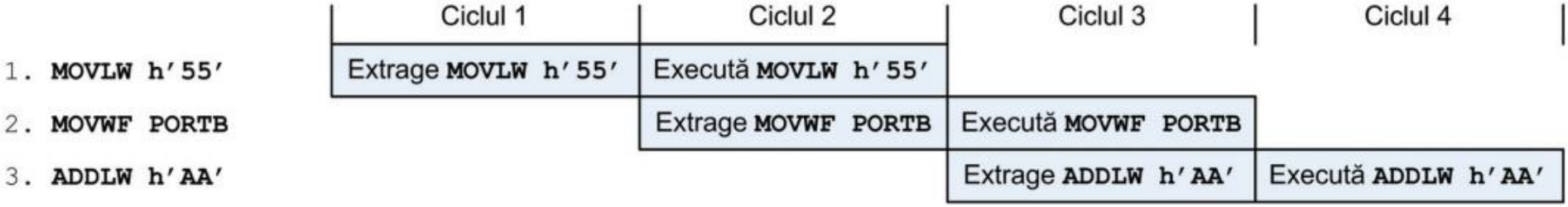
- Un ciclu instrucțiune presupune două faze:
  - Faza de extragere a instrucțiunilor din memorie.
  - Faza de execuție a instrucțiunilor.
- Extragerea: se copiază instrucțiunea din memoria program într-un registru intern al UCP.
- Execuția: decodificarea instrucțiunii, extragerea operanzilor, execuția operației, scrierea rezultatului în memorie.
- Durata de execuție depinde de frecvența de lucru.

Execuția instrucțiunilor

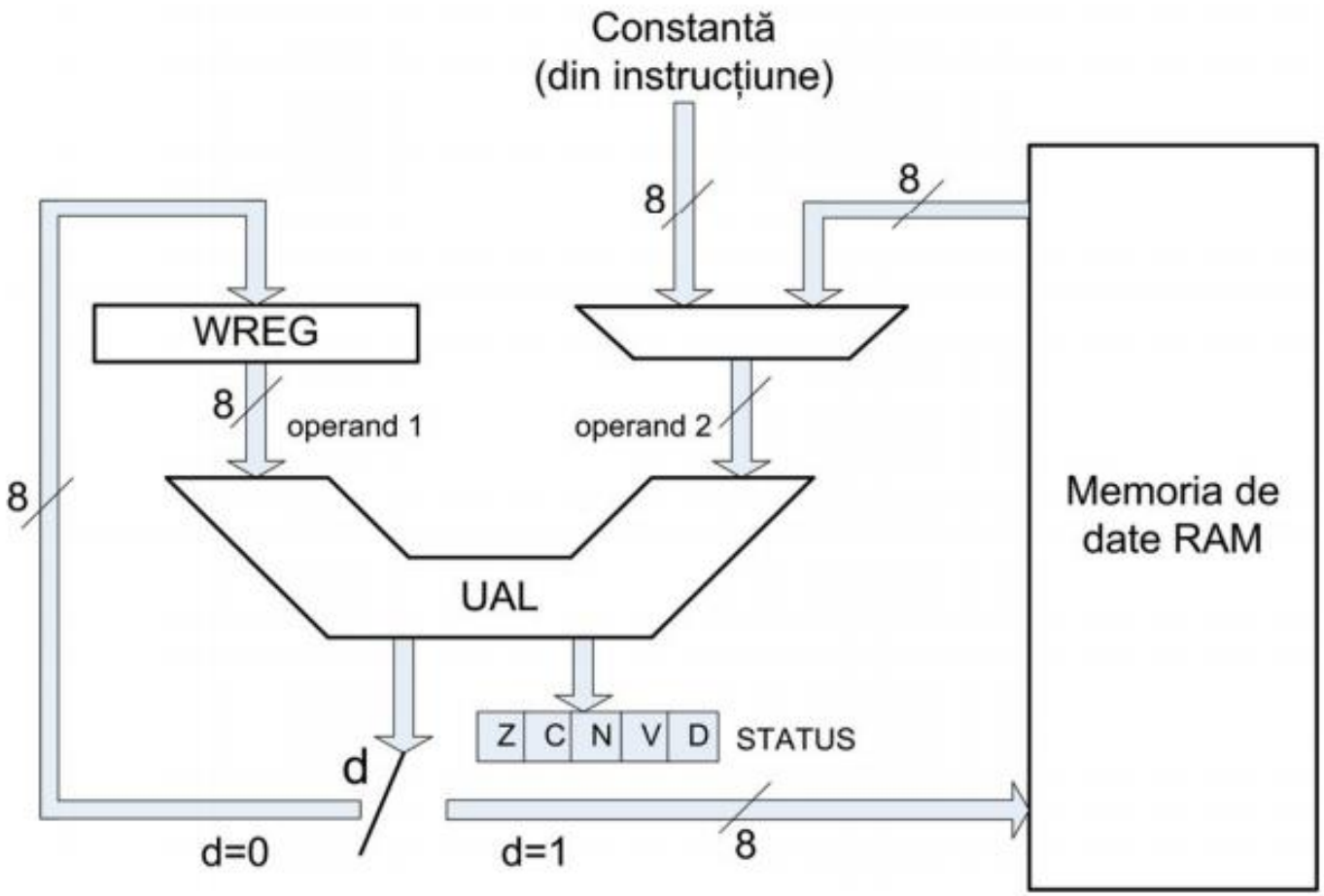
- Fiecare instrucțiune se execută în 2x4 cicluri instrucțiune (Q), adică în 2x4 tacturi sistem.
- În cât timp se execută o instrucțiune:
  - Frecv. = 1KHz o instrucțiune se execută în 8ms.
  - Frecv. = 1MHz o instrucțiune se execută în 8us.
- Tactul sistem este divizat intern la 4, fiind generate 4 semnale tact:
  - PC este incrementat în Q1, instrucțiunea este extrasă în memorie în registrul de instrucțiune în Q4.
- În următoarele Q1-Q4:
  - Q1: decodificare instrucțiune.
  - Q2: citire date.
  - Q3: procesare date.
  - Q4: scriere date.

Execuția instrucțiunilor





Unitatea aritmetică și logică

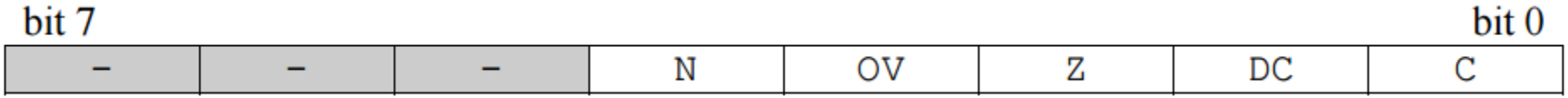


Unitatea aritmetică și logică

- O unitate aritmetică și logică (UAL) pe 8 biți și un registru (acumulator, en: Work Register) pe 8 biți.
- UAL execută operații aritmetice pe 8 biți (adunare, scădere, operații logice – AND,OR,XOR,etc - , deplasări, rotiri, înmulțiri.
- În cazul a doi operanzi, unul se va regăsi în WREG, celălalt poate fi o constantă (literal) sau un registru din memorie.
- Bitul de stare din instrucțiune (bit destinație d), rezultatul poate fi stocat în registrul de lucru (d=0) sau în memoria de date (d=1).
- Rezultatul înmulțirii se stochează în reg: PROD

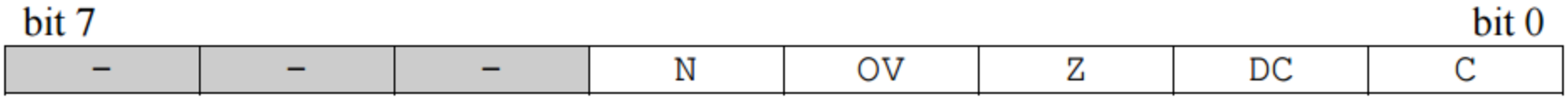
Unitatea aritmetică și logică

- Rezultatul înmulțirii se stochează în reg.: PRODH:PRODL.
- Introducerea WREG a simplificat arhitectura ALU. Lipsa aceastuia ar fi dus la instrucțiuni mult mai complexe:
  - Ex.: adunarea a două numere implică: locația celor două numere și locația rezultatului.
- Indicatorii operațiilor aritmetice și logice sunt stocați în registrul STATUS:



Unitatea aritmetică și logică

- Bitul N: Negative.
- Bitul OV: Overflow.
- Bitul Z: Zero.
- Bitul DC: Digital Carry.



Foaia de catalog a PIC18F4455

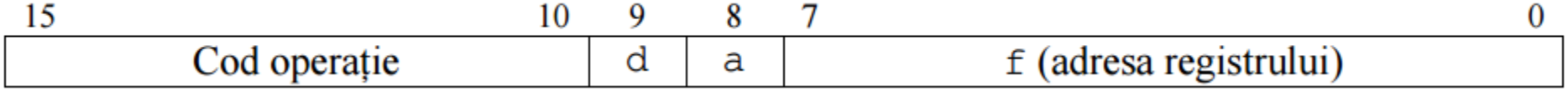
- Disponibil pe microchip.com
- http://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf

Instrucțiuni

- PIC18F4455 implementează un set redus de instrucțiuni (aprox. 78 de instrucțiuni)
- Tipuri de instrucțiuni:
  - Pe octet.
  - Pe bit.
  - Cu constante (literali).
  - De control.

Instrucțiuni pe octet

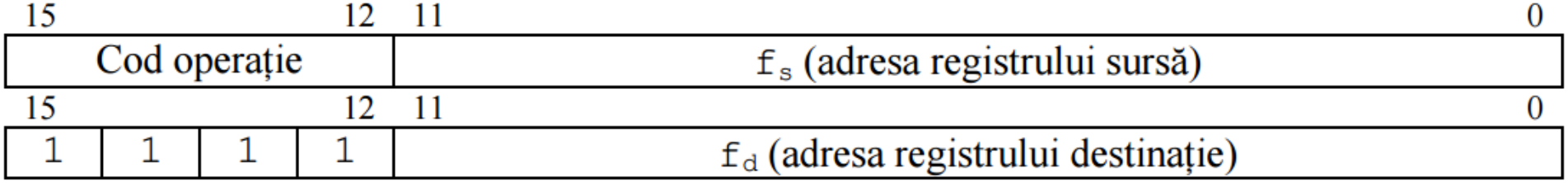
- Majoritatea instrucțiunilor au 3 operanzi:
  - Registrul: operandul **f** (registrul ce urmează a fi utilizat, prin nume simbolic sau adresă)
  - Destinația rezultatului: operandul **d** (d=0, d=W – rezultatul stocat în WREG, d=1, d=F – rezultatul stocat în registrul specificat prin f)
  - Zona de memorie accesată: operandul **a** (specifică modul de acces la registrul f, a=0, a=ACCESS – f se găsește în Access Bank, dacă a=1, a=BANKED, f segăsește în Bank-uri, iar pentru a-l accesa se folosește registrul BSR).



Instrucțiuni pe octet

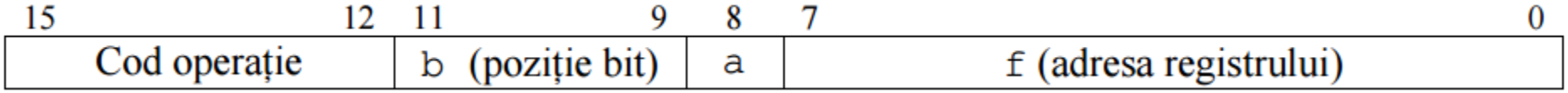
Format instrucțiuni orientate pe octet	Exemplu de utilizare

cod_operație f,d,a
cod_operație f,a
 	
f=adresa pe 8 biți a registrului
d=0 (d=W) destinația rezultatului este registrul de lucru WREG
d=1 (d=F) destinația rezultatului este registrul f
a=0 (a=ACCESS) registrul f se găseşte în Access Bank
a=1 (a=BANKED) registrul f se găseşte în Bank-uri
 	
Excepție: MOVFF f_s,f_d
f_s=adresa pe 12 biți a registrului sursă
f_d=adresa pe 12 biți a registrului destinație
	ADDWF REG1, W, BANKED
	CLRF 0x01, ACCESS
	
	MOVFF REG_s, REG_d



Instrucțiuni pe bit

- Toate instrucțiunile au 3 operanzi:
  - Registrul: operandul **f** (registrul ce urmează a fi utilizat, prin nume simbolic sau adresă)
  - Bitul de registru: operandul **b** (format din 3 biți, specifică poziția bitului din registrul f)
  - Zona de memorie accesată: operandul **a** (specifică modul de acces la registrul f, a=0, a=ACCESS – f se găsește în Access Bank, dacă a=1, a=BANKED, f segăsește în Bank-uri, iar pentru a-l accesa se folosește registrul BSR).

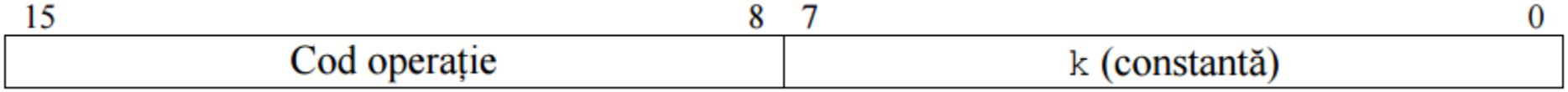


Instrucțiuni pe bit

Format instrucțiuni orientate pe bit	Exemplu de utilizare
cod_operație f,b,a f=adresa pe 8 biți a registrului b=poziția bitului (0:7) în cadrul registrului f a=0 (a=ACCESS) registrul f se găseşte în Access Bank a=1 (a=BANKED) registrul f se găseşte în Bank-uri	BSF REG, 4, ACCESS

Instrucțiuni cu constante (literali)

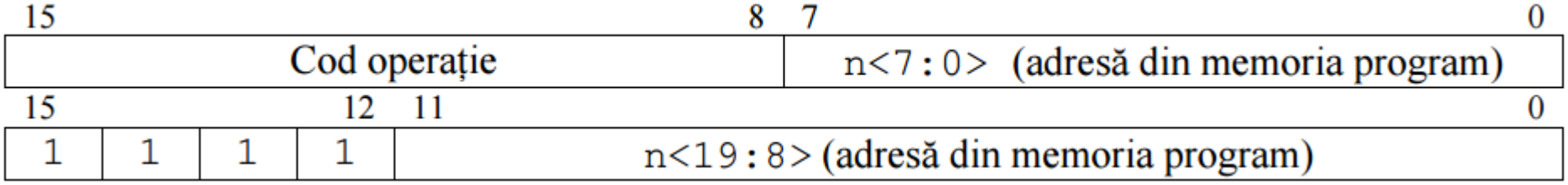
- Toate instrucțiunile au 3 operanzi:
  - Constanta (literalul): operandul **k**



Format instrucțiuni cu constante	Exemplu de utilizare
cod_operație k k=valoarea efectivă pe 8 biți Excepție: LFSR f,k f=registrul FSR utilizat k=adresa pe 12 biți	MOVLW d'23' LFSR FSR0, h'100'

Instrucțiuni de control

- În funcție de tipul lor pot include următorii operanzi:
  - O adresă de memoria program (etichetă): operandul **n** (instrucțiunea GOTO).
  - Utilizarea stivei rapide: operandul **s** (instrucțiunile CALL, RETURN, RETFIE).
  - Fără operand.
- Operandul asigură un salt în memoria program și modifică valoarea din PC.



Instrucțiuni de control

Format instrucțiuni de control	Exemplu de utilizare
cod_operație n **cod_operație n,{s}** **cod_operație s** **cod_operație** n=etichetă (adresă din memoria program) s= utilizare stivă rapidă Excepție: RETLW k k=constantă pe 8 biți	GOTO main_loop CALL rutina RETURN FAST NOP RETLW 0x15