MICROCONTROLERUL
-continuare curs 1-

2.1. Structura unui microcontroler (continuare curs 1)

2.1.1. Unitatea timer

Pentru folosirea în industrie a microcontrolerului mai este nevoie de câteva blocuri. Unul din acestea este blocul timer care este important pentru că ne dă informația de timp, durată, protocol, etc.

Unitatea de bază a timer-ului este un contor liber (free-run) care este de fapt un registru a cărui valoare numerică creşte cu unu la intervale egale, aşa încât luându-i valoarea după intervalele T_1 şi T_2 şi pe baza diferenței lor să putem determina cât timp a trecut. Acesta este o parte foarte importantă a microcontrolerului al cărui control cere cea mai mare parte a timpului nostru.

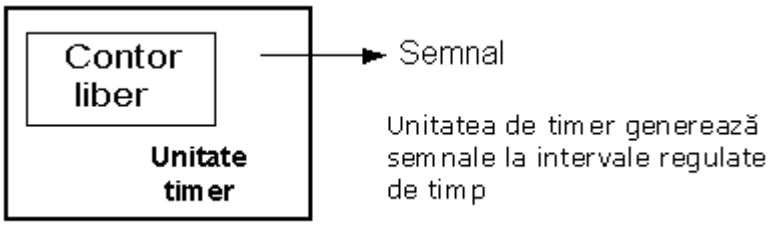


Fig. 2.1. Unitatea timer

2.1.2. Watchdog-ul

Un lucru important este funcționarea fără defecte a microcontrolerului. În cazul interferențelor (ce adesea se întâmplă în industrie) microcontrolerul se poate oprii din executarea programului, sau începe să funcționeze incorect.

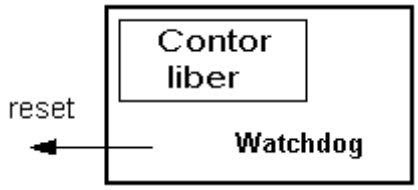


Fig. 2.2. Watchdog-ul

Dacă acest lucru se întâmplă cu un calculator, îl resetăm pur şi simplu şi el va continua să lucreze. Totuşi, nu există buton de resetare pe care să-l apăsăm în cazul microcontrolerului care să rezolve această problemă.

Pentru a depăşi acest obstacol, avem nevoie de a introduce încă un bloc numit watchdog-câinele de pază. Acest bloc este de fapt un alt contor liber (free-run) unde programul nostru trebuie să scrie un zero ori de câte ori se execută corect. În caz că programul se „blochează", nu se va mai scrie zero, iar contorul va reseta singur microcontrolerul la atingerea valorii sale maxime. Aceasta va duce la rularea programului din nou, şi corect de această dată pe toată durata. Acesta este un element important al fiecărui program ce trebuie să fie fiabil fără supravegherea omului.

2.1.3. Convertorul Analog-Digital

Pentru că semnalele de la periferice sunt în general semnale analogice, substanțial diferite de cele pe care le poate înțelege microcontrolerul semnale numerice (zero şi unu), ele trebuie convertite într-un mod care să fie înțeles de microcontroler.

Această sarcină este îndeplinită de un bloc pentru conversia analog-digitală sau de un convertor AD.

Acest bloc este responsabil pentru convertirea unei informații despre o anumită valoare analogică într-un număr binar şi pentru a o urmări pe tot parcursul până la blocul CPU care o va procesa.

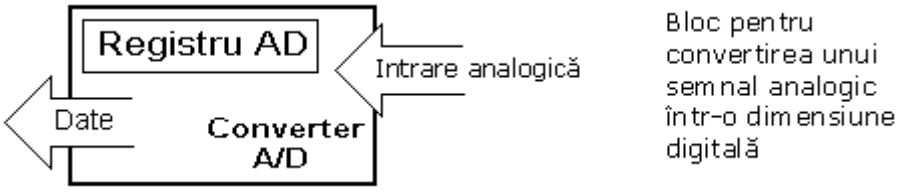


Fig. 2.3. Convertorul Analog-Digital

2.1.4. Comunicația serială

Interfața de comunicație serială este un periferic deosebit de util pentru realizarea conectarii microcontrolerului cu alte echipamente (de exemplu un calculator personal sau chiar un alt microcontroler). Este o interfață serială asincronă care poate funcționa în mai multe moduri. Cel mai frecvent se utilizeaza modul ”clasic” care este compatibil cu toate interfețele seriale asincrone existente. Un mod mai special îl constituie „modul comunicație multiprocesor” care permite realizarea unei magistrale de comunicație pe care să fie instalate mai multe microcontrolere şi care să poată comunica între ele fiecare având o anumită adresă.

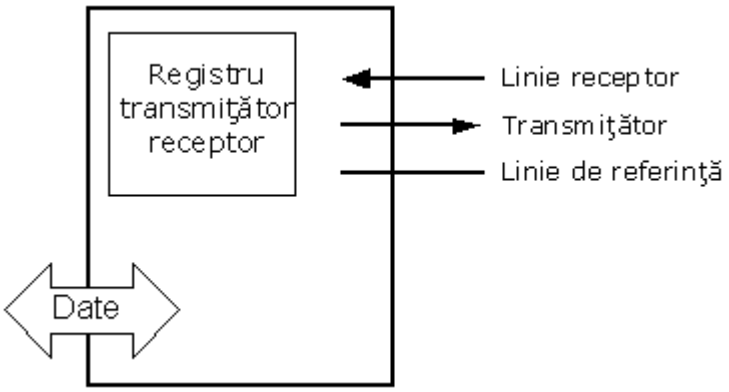


Fig. 2.4. Unitatea serială folosită pentru a trimite date, dar numai prin trei linii

Spre deosebire de transmisia paralelă, datele sunt mutate aici bit cu bit, sau într-o serie de biți, de unde vine şi numele de comunicație serială.

Schema următoare reprezintă secțiunea centrală a microcontrolerului.

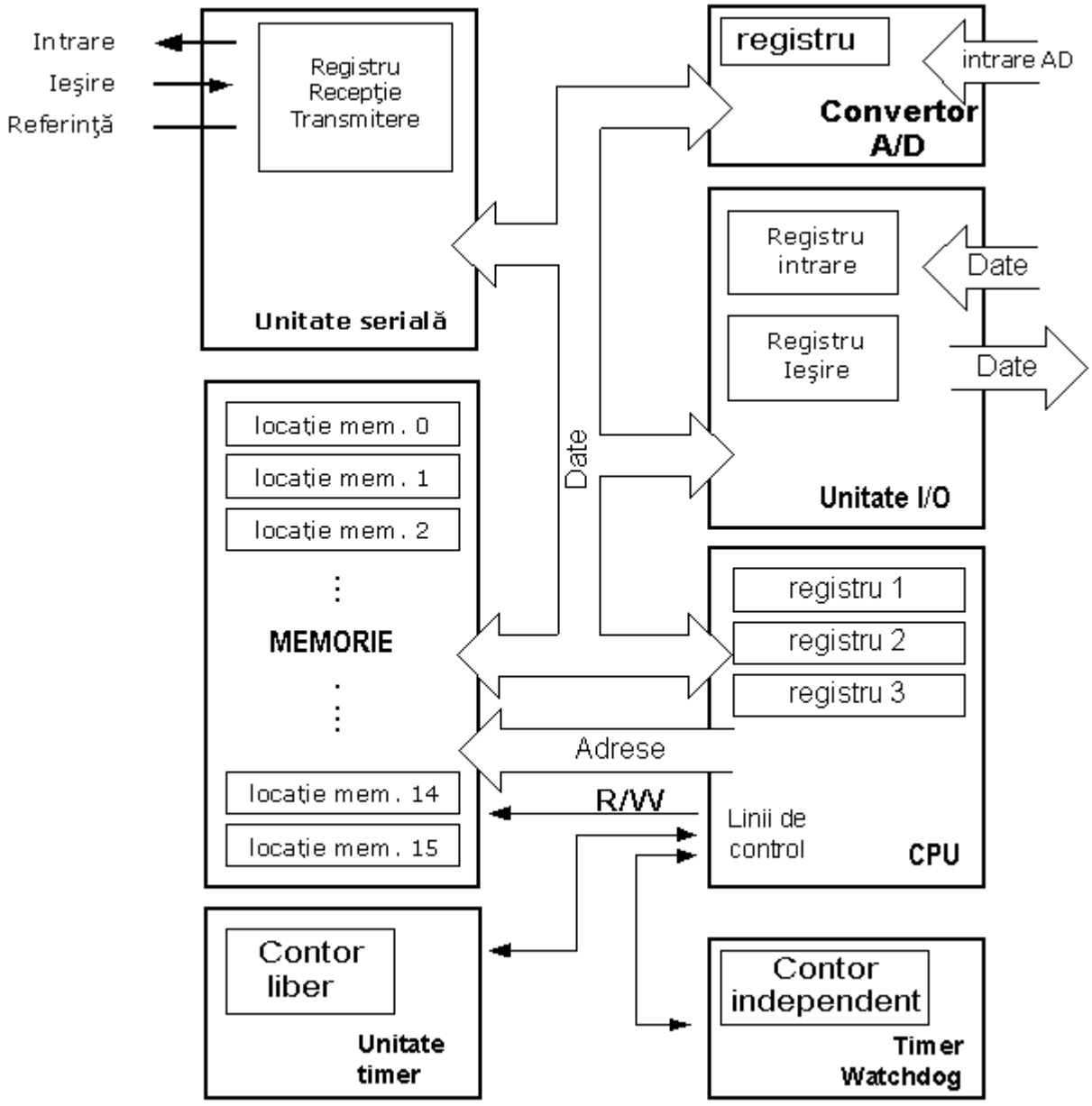


Fig. 2.5. Elementele de bază şi conexiunile interne ale microcontrolerului

Ceea ce deosebeşte fundamental un microcontroler de un circuit integrat analogic sau digital, este faptul că el nu poate face nimic dacă nu este programat. Programul software conferă microcontrolerului, abilitatea de a realiza funcții diferite cu aceeaşi configurație hardware.

2.1.5. Programul

Exemplu de program:

```
START
REGISTER1=MEMORY LOCATION_A
REGISTER2=MEMORY LOCATION_B
PORTA=REGISTER1 + REGISTER2

END
```

Programul adună conținutul a două locații de memorie, şi vede suma lor la portul A.

Prima linie a programului este pentru mutarea conținutului locației de memorie „A" într-unul din regiştri unității de procesare centrale, iar ce-a dea doua linie este pentru mutarea conținutului locației de memorie „B" în celălalt registru al unității de procesare centrale.

Următoarea instrucțiune determină ca unitatea de procesare centrală să adune conținutul celor doi regiştri să trimită rezultatul obținut la portul A, încât suma acestei adunări să fie vizibilă (să poată fi afişată).

Scrierea programului se realizează de obicei intr-un editor ce permite salvarea liniilor de comandă introduse.

Există mai multe opțiuni pentru scrierea programului de control al aplicației şi anume:

- cod maşină (cod hexadecimal);
- limbaj de asamblare;
- limbaj de nivel înalt (C, Pascal, Basic etc).

Comenzile recunoscute de microcontroler sunt cele scrise în cod maşină. Limbajul de asamblare şi limbajele de nivel înalt sunt mai evoluate, conțin instrucțiuni ce sunt uşor de reținut, dar pentru transformarea acestora în cod maşină avem nevoie de un compilator.

Compilatorul este program software, de obicei oferit gratuit de producătorii microcontrolerelor. Pentru a transfera codul hexadecimal rezultat în urma compilării, în memoria program a microcontrolerului este nevoie de un programator. Programatorul este compus dintr-un modul electronic care asigură interfațarea între aplicația ce conține microcontrolerul şi calculator (PC), şi un program software ce rulează pe PC.

Assembler aparține limbajelor de nivel scăzut ce sunt programate lent, dar folosesc cel mai mic spațiu în memorie şi dă cele mai bune rezultate când se are în vedere viteza de execuție a programului.

Programele în limbajul C sunt mai uşor de scris, mai uşor de înțeles, dar sunt mai lente în executare decât programele în Assembler.

Basic este cel mai uşor de învățat, şi instrucțiunile sale sunt cele mai aproape de modul de gândire a omului, dar ca şi limbajul de programare C este de asemenea mai lent decât Assembler-ul.

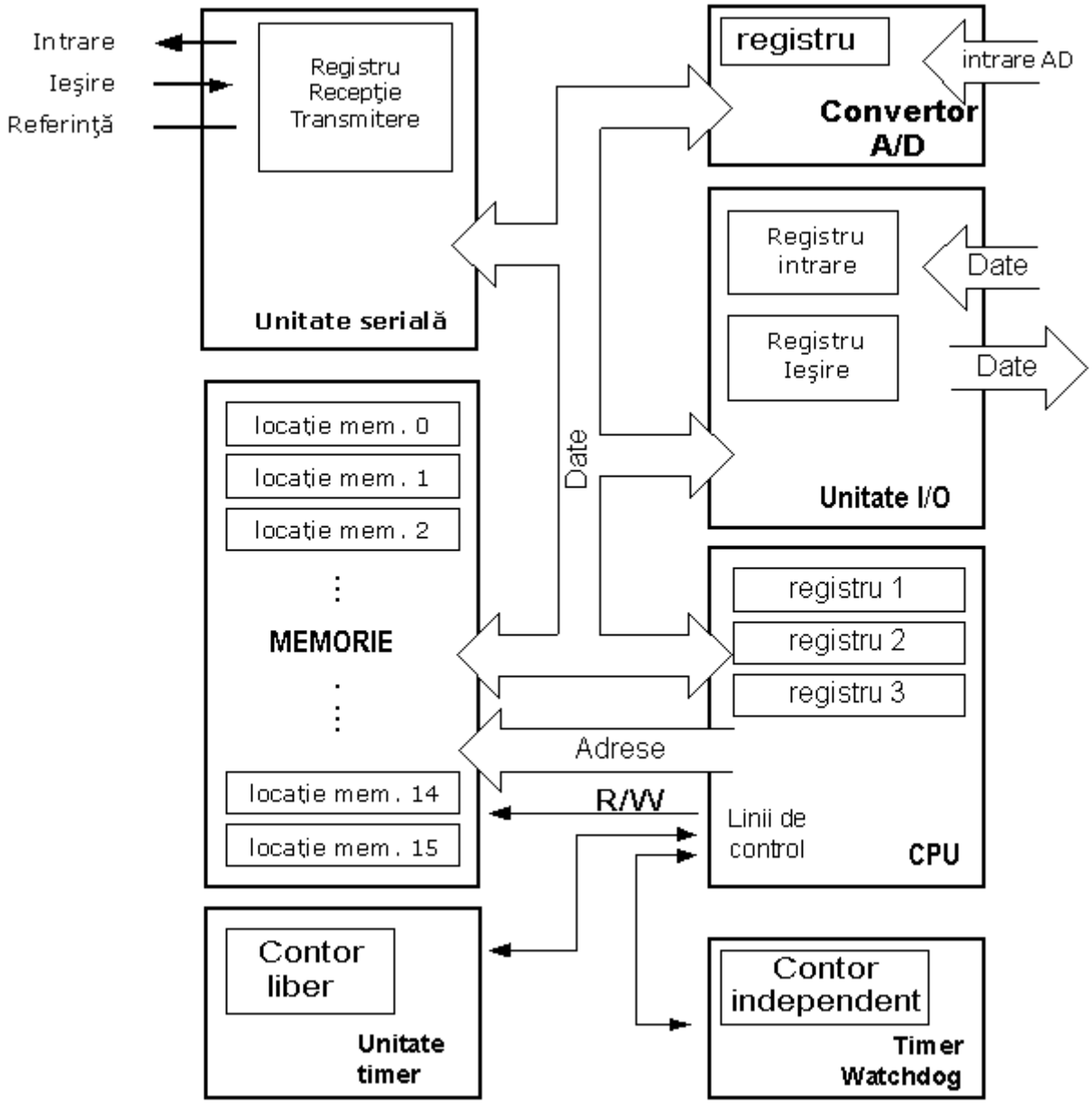
După ce este scris programul, trebuie ca microcontrolerul să fie instalat într-un aparat Mai întâi trebuie ca microcontrolerul să fie conectat la o sursă (tensiune necesară pentru operarea tuturor instrumentelor electronice) şi la oscilator al cărui rol este similar inimii din corpul uman.

Bazat pe ceasul său microcontrolerul execută instrucțiunile programului. Îndată ce este alimentat microcontrolerul va executa un scurt control asupra sa, se va uita la începutul programului şi va începe să-l execute.

Cum va lucra aparatul depinde de mulți parametri, cel mai important fiind priceperea dezvoltatorului de hardware, şi de experiența programatorului în obținerea maximului din aparat cu programul său.

Subiecte:

1. Rolul unității timer. Desen.
2. Rolul Watchdog-ului. Desen.
3. Rolul Convertorului analog-digital. Desen.
4. Exemplu de program. Explicați liniile programului.
5. Ce opțiuni pentru scrierea programului microcontrolerului există?



Elementele de bază şi conexiunile interne ale microcontrolerului

Zecimal	Binar	Hexazecinal
0	0000	0×0
1	0001	0×1
2	0010	0×2
3	0011	0×3
4	0100	0×4
5	0101	0×5
6	0110	0×6
7	0111	0×7
8	1000	0×8
9	1001	0×9
10	1010	0×A
11	1011	0×B
12	1100	0×C
13	1101	0×D
14	1110	0×E

Zecimal	Binar	Hexazecinal
15	1111	0×F

De exemplu 10110110≡0xB6 (0x - reprezintă convenția de scriere a numerelor hexazecimale, B- este quartetul cel mai semnificativ (1011) iar 6 reprezintă quartetul cel mai puțin semnificativ (0110).

a. Programul scris in limbaj de asamblare (pic)

```
#DEFINE  LED      PORTB,0          ;definire port de iesire, pinul RB0
         CBLOCK   0CH              ;inceput de declaratii variabile in memoria
                                   RAM
                  TEMP1, TEMP2
         ENDC                      ;sfarsit de declaratii variabile in memoria
                                   RAM
START:                             ;eticheta de start
         BSF      LED              ;setarea portului de iesire la nivel
                                   logic ridicat (5 V)
         CALL     DELAY            ;salt rutinei DELAY
         CALL     DELAY            ;salt rutinei DELAY
         BCF      LED              ;setarea portului de iesire la nivel
                                   logic scazut (0 V)
         CALL     DELAY            ;salt rutinei DELAY
         CALL     DELAY            ;salt rutinei DELAY
         GOTO     START            ;revenirea la eticheta START
.....
```

b. Acelasi program scris in limbajul Basic:

```
--------------------------------------------------------------------------
loop:                              'eticheta de start
         high     portb.0          'setarea portului de iesire la nivel logic
                                   ridicat (5 V)
         pause    500              'stare de asteptare 500 milisecunde
         low      portb.0          'setarea portului de iesire la nivel logic
                                   scazut (0 V)
         pause    500              'stare de asteptare 500 milisecunde
         goto     loop             'salt la eticheta loop
--------------------------------------------------------------------------
```

c. fisierul hex rezultat din fisierul de asamblare

```
:1000000000308316860083128601861700C200C20900
:1000100086130C200C200528FF308D00FF308C004B
:0C002000000008C0B10288D0B0E2808002F
:02400E00F83F79
:00000001FF
```

d. program in C

```
#include "16F690.h"          // header-ul pentru microcontroler
#use delay(clock=4000000)    // setarea frecvenței de tact
void main ()                 // funcția principală
{
while(1)                     // bucla fără sfârşit
{
output_high(PIN_C0);         // aprinderea ledului conectat la pinul RC0 (trecerea în 1 logic)
delay_ms(1000);              // întârziere de 1000 ms

output_low(PIN_C0);          // stingerea ledului conectat la pinul RC0 (trecerea în 0 logic)
delay_ms(1000);              // întârziere de 1000 ms
}
}
```