Programarea microcontrollerelor

Limbaje utilizate (1)

- limbaj de asamblare
  - de obicei diferit de cele ale microprocesoarelor de uz general
  - oferă acces complet la hardware
  - dificil de scris cod
  - de multe ori apar restricții
  - ex.: toate operațiile au ca operand obligatoriu un anumit registru (acumulator)

Limbaje utilizate (2)

- limbaje de nivel înalt
- în general C
  - mai rar C++
- depinde de nivelul de complexitate al microcontrollerului
- majoritatea microcontrollerelor sunt însoțite de medii de programare specifice

Intrări-ieșiri

- chiar în limbajul C/C++, nu putem apela la
  - `scanf/printf`
  - `cin/cout`
- de obicei nu avem tastatură sau monitor
- citirea și scrierea - comunicare cu anumite periferice care pot juca aceste roluri
- este posibil să avem la dispoziție funcții predefinite pentru aceste acțiuni

Structura unui program (1)

- un microcontroller lucrează în principiu la nesfârșit
- programul principal - buclă infinită
- sunt testate anumite intrări/condiții și se execută ca răspuns anumite acțiuni
- condiții testate
  - valori ale unor semnale externe (senzori, ...)
  - valori citite de la unele periferice

Structura unui program (2)

- acțiunile realizate
  - calcule - de obicei nu foarte complexe
  - comenzi către periferice
- testarea condițiilor se poate face la anumite intervale de timp, bine stabilite
- pot fi mai multe intrări independente care sunt preluate
  - ex.: automobil - informații de la diverși senzori

Structura unui program (3)

```
int main() {
  inițializări variabile, periferice
  while(1) {
     if(condiție1) acțiune1;
     if(condiție2) acțiune2;
     ...}
}
```

Accesul la periferice (1)

- perifericele interne ale microcontrollerului: au asociate adrese fixe
  - mediul de programare folosește constante predefinite pentru a accesa aceste adrese
- perifericele externe: microcontrollerul comunică prin intermediul unor porturi
  - uneori specifice (ex. interfață serială)
  - alteori de uz general (mai des)

Accesul la periferice (2)

- de unde știm adresele asociate perifericelor?
- mediul de programare are definite informațiile necesare
  - ex.: fișiere header (`.h`)
- putem avea la dispoziție și funcții predefinite pentru lucrul cu unele periferice

Accesul la periferice (3)

- adresele perifericelor pot fi

a) mapate în memorie
   - perifericele și locațiile de memorie ocupă același spațiu de adrese

b) izolate
   - spații de adrese separate - fiecare începe de la 0

- depinde de procesor

Accesul la periferice (4)

- în primul caz, perifericele sunt accesate prin operații de atribuire (citire/scriere)
  - la fel ca orice locație de memorie
- în al doilea caz
  - limbajele de nivel înalt nu au instrucțiuni pentru accesul la I/O; soluții:
  - limbaj de asamblare
  - funcții predefinite de acces la I/O

Documentații (1)

- pentru a afla adresele și funcțiile specifice, trebuie consultată documentația
  - a microcontrollerului - pentru perifericele interne
  - a plăcii de dezvoltare - pentru perifericele externe microcontrollerului, dar care sunt plasate pe placa de dezvoltare de la crearea acesteia
    - compilatorul poate avea informații despre acestea

Documentații (2)

- trebuie înțeles modul de funcționare al unui periferic pentru a putea lucra cu el
  - nu doar cum este conectat
- ce date și comenzi trebuie să primească
  - plus - în ce ordine și la ce momente
- ce date poate transmite
  - și la ce momente

Documentații (3)

- exemple de detalii care trebuie avute în vedere
- la multe elemente de afișare (în principal LED), de obicei trebuie transmisă valoarea 0 pentru aprins, respectiv 1 pentru stins
- unele comenzi sau ieșiri ale unor periferice sunt active pe 0
  - valoarea 0 arată activarea comenzii; 1 - inactiv

Exemplu

- specificație
  - preluarea unei valori numerice de la portul P1
  - transmiterea dublului valorii sale către portul P2
  - ori de câte ori bitul 6 din portul P3 are valoarea 1

Exemplu (cont.)

```
unsigned char x;
while(1) {
  if(P3 & 0x40) {
  // uneori if(P3_6 == 1)
     x=P1;
     P2=x+x;}
}
```

Măsurarea timpului

- realizarea testelor/acțiunilor la intervale regulate de timp
- implică utilizarea timerelor
- de obicei timerele sunt incluse în microcontroller
  - deci sunt periferice interne
  - deoarece sunt necesare practic întotdeauna

Funcționarea unui timer (1)

- structura de bază - numărător
  - incrementare/decrementare - la comanda unui semnal periodic, de frecvență cunoscută
  - derivat din semnalul de ceas al sistemului
- programatorul controlează
  - valoarea inițială încărcată în numărător
  - momentul de la care numărarea este validată
  - dacă numărătorul se reîncarcă la terminare

Funcționarea unui timer (2)

- tot programatorul testează terminarea numărării
  - semnal cu valoarea 0 sau 1, provenit de la numărător
- la terminarea numărării, se poate reîncărca valoarea inițială
  - comportament periodic, nu singular
  - configurat prin program

Exemplu - Intel 8051

```
TR0=0; // blocare timer
TH0=4; // inițializare constantă timer
TL0=0;
TR0=1; // validare numărare
while(TF0==0) ; // așteptare terminare
... // acțiuni după expirarea timpului
```

Întreruperi (1)

- exemplul anterior - cât timp așteptăm exprirarea perioadei timerului, nu putem realiza alte acțiuni
- dacă sunt mai multe evenimente periodice independente care trebuie tratate?
- soluție - timerul este lansat, iar la finalul perioadei generează o întrerupere
  - între timp se pot executa alte instrucțiuni

Întreruperi (2)

- microcontrollerele permit generarea de cereri de întrerupere
  - de către unele periferice interne (ex. timer)
  - din surse externe - de obicei periferice externe
- în general toate întreruperile pot fi deazactivate
  - procesorul poate refuza tratarea lor
  - dar nu este recomandabil pe termen lung

Rutina de tratare a întreruperii

- este scrisă în cea mai mare parte ca o funcție obișnuită
- dar trebuie indicat faptul că se apelează la apariția unei cereri de întrerupere
  - și pentru care există sursă de întreruperе
- sintaxa depinde de compilator
- ex.: timerul 0 al Intel 8051

```
void f() interrupt 1 {...}
```

Tratarea întreruperilor

- în timpul tratării unei întreruperi, poate apărea o nouă cerere din altă sursă
  - poate cauza probleme
  - se recomandă dezactivarea întreruperilor în timpul tratării uneia
- variabile globale folosite în rutinele de tratare
  - trebuie declarate ca `volatile` - nu sunt optimizate de compilator