Exercicis resolts

A partir de dos exemples concrets es mostra el plantjament d'algorismes de resolució de programes en assemblador. És una metodologia fàcil d'aprendre que convé seguir sempre. Com es veu a continuació, els passos a resoldre són, primer trobar el diagrama de flux i després passar-lo a assemblador. Aquesta manera de resoldre problemes en assemblador s'emprarà a partir del proper capítol.

Exercici resolt 1. El primer algorisme.

Es volen sumar dos nombres que s'agafen d'un perifèric d'entrada i es vol mostrar el el resultat en un perifèric de sortida.

Solució

Plantejament

El processador es troba connectat a l'entrada/sortida del sistema a través de ports d'entrada. En aquest exercici s'empra un port d'entrada i un de sortida anomenats PortA i PortB respectivament.

El plantejament en aquest exercici és molt simple. Es guarden en dos registres diferents les dues dades que entren pel PortA. Aleshores es sumen, guardant el resultat en un dels registres, i es treu pel PortB.

Diagrama de flux

El diagrama de flux és un mètode gràfic d'expressió de l'algorisme. Es basa en la construcció d'un graf amb símbols predefinits que indica la tipologia de tasca a realitzar. En general es solen emprar els següents símbols:

- Rectangle → Indica execució.
- Rectangle inclinat → Implica acció d'entrada/sortida
- Rombo → Pregunta per una condició, actuant de diversa manera segons el valor d'aquesta.
- Oval → S'empra per indica inici i fi de programa
- Cercle → Emprat per connectar diferents parts del programa.

El diagrama de flux de l'exercici es mostra en la figura 1. Es pot observar que en aquest cas és molt directe. Cal observar com cada sómbil indica l'acció que es realitza. L'execució de l'algorisme soluciona el problema plantejat.

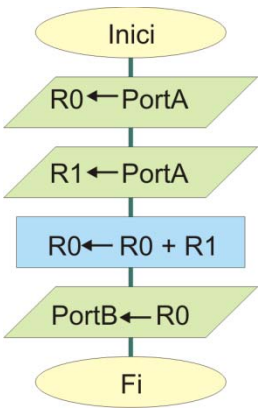

Figura 1. Diagrama de flux de l'exercici 1.

Un diagrama de flux per a un programa en assemblador sol ser convenient fer-lo a baix nivell. Com que les instruccions assemblador es basen en transferència de dades entre registres, les accions convé especificar-les en lleguatge trasferència de registres. És útil emprar la nomenclatura establerta pel processador.

Programa en assemblador

La traducció del diagrama de flux en un programa assemblador és directa. Donat que les instruccions ja s'han posat en format adequat pel processador, només cal traduir els símbols en les instruccions assemblador adequades. La figura 2 mostra el programa assemblador d'aquest exercici.

```
IN R0, PortA;
IN R1, PortB;
ADD R0, R1;
OUT PortB, R0;
```

Figura 2. Codificació en assemblador de l'exercici 1.

Exercici resolt 2. Cap a un algorisme genèric

Treure pel PORTB el resultat de sumar tots els nombres primers entre 0 i 100.

Solució

Plantejament

L'exercici en aquest cas empra una iteració sobre una variable: la variable *sum* (que tindrà el resultat final) anirà sumant la variable índex *a*, fins que aquesta arribi al final de la iteració. Per tant, es realitzen les operacions:

sum ← sum + a
a ← a+2

Un punt fonamental és trobar les condicions inicials i finals. En aquest cas les condicions inicials són: la variable *sum* ha de partir de 0 i la variable índex del primer valor a sumar *a* val 1. La condició final es troba quan la variable índex sobrepassa el límit, que en aquest exercici és 100.

Diagrama de flux

El diagrama de flux es dóna en la figura 3. S'hi dientifiquen clarament les variables sum i a. Es pot observar que, tot i seguir el format RTL, en aquest cas s'han emprat variables normals enlloc de registres del processador. El seu ús permet identificar més fàcilment les variables emprades. Un cop es passa a l'assemblador és fàcil assignar registres del processador a les variables normals.

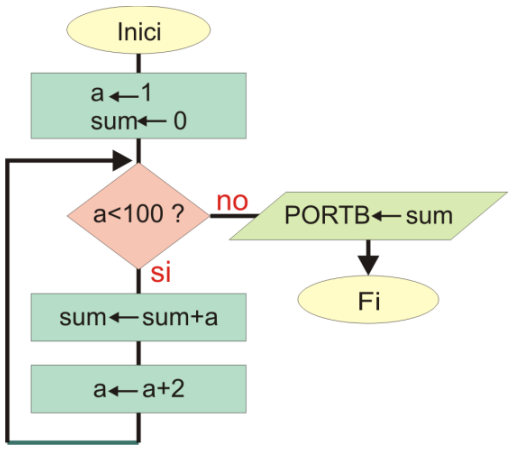

Figura 3. Diagrama de flux de l'exercici 2.

Programa en assemblador

La traducció en assemblador del programa es dóna en la figura 4.

```
.DEF sum = R10;
.DEF a = R11;
inici: LDI sum, 0;
       LDI a, 1;
proper: CPI a, 100;
       BRPL fi;
       ADD sum, a;
       ADDI a, 2;
       RJMP proper;
fi: OUT PORTB, sum;
       RJMP fi;
```

Figura 4. Codificació en assemblador de l'exercici 2.

Es veu clarament com l'ús de variables genèriques facilita la comprensió del programa. La conversió de variables genèriques a registres la realitza la directiva *.DEF*. Per altra part, el salt a parts del programa es fa amb instruccions de salt com *RJMP* o *RCALL*, i el lloc on es salta s'identifica amb etiquetes, aquí *inici*, *proper* i *fi*.

4.12 Exercicis

1. Trobar la velocitat lineal a què gira les pistes interna i externa d'un disc dur. I en el cas d'un DVD?

2. La UART és una unitat de transmissió sèrie d'informació entre dos dispositius. La figura següent mostra el format que se segueix: quan un byte es vol enviar, aquest s'empaqueta amb un 0 en la posició menys significativa i un 1 en la més significativa, formant un paquet de 10 bits que s'envien a través del port sèrie. Quan no es transmet res la línia queda en repós a 1-lògic. Adjunt en la taula es mostren velocitats de transmissió típiques.

Donat per suposat que la transmissió de cada bit dura 16 cicles de rellotge base del sistema, es demana:

i) Calcular la freqüència de rellotge mínima per a cada baudrate. S'entén com a *baudrate* el nombre de bits enviats per segon.

ii) Calcular quan temps es triga en transmetre un byte d'informació per a cada baudrate.

iii) Quants bytes es transmeten, en cada cas, per segon?

Freqüencies típiques(Baudrate)
2400
4800
9600
19200
38400
57600
115200

Exercici 2. Format de transmissió sèrie.

3. Donats els següents algorismes descrits en assemblador:

i) Trobar els corresponents diagrames de flux.
ii) Dir que realitza cadascun d'ells.
iii) Treballant amb una freqüència de rellotge de 20MHz i suposant que cada instrucció dura un cicle de rellotge, quan triga en executar-se cada algorisme?

Nota: Tot i que la funció que realitza cada instrucció és bastant evident, els apèndixs A1 i A2 detallen la seva funcionalitat.

```
--ALGORISME 1
inici:    LDI R0, 100;
bucle:    OUT PORTB, R0;
          DEC R0;
          BRPL bucle;
continuar:      ...
```

```
--ALGORISME 2
.DEF A = R0;
.DEF B = R1;
.DEF tmp = R2;

inici:    LDI A, 1;
          LDI B, 0;
bucle:    OUT PORTB, A;
          MOV tmp, A;
          ADD A, B;
          MOV B, tmp;
          CPI A, 1000;
          BRMI bucle;
continuar:      ...
```

```
--ALGORISME 3
.DEF tmp0 = R0;
.DEF tmp1 = R1;

inici:    LDI tmp0, 0xFFF;
bucle2: LDI tmp1, 0xFFF;
bucle1: DEC tmp1;
          BRNZ bucle1;
          DEC tmp0;
          BRNZ bucle2;
continuar:      ...
```

Exercici 3. Algorismes.