Curso S1601002

Introducción á programación de autómatas

Representación da información

1. Formatos de almacenamento da información nun PLC S7-1200

Os diferentes tipos de datos almacénanse e manéxanse nun autómata S7-1200 fundamentalmente en grupos de 8 bits (8, 16, 32 e 64 bits).

Dependendo do tipo de dato necesitarase máis ou menos espacio na memoria para o seu almacenaxe e tratamento.

Dado que nun dispositivo microprocesador só podemos almacenar dous posibles valores na memoria (0 e 1), preséntanse algúns "problemas" a resolver para traballar con datos do mundo real no que nos desenvolvemos. Por exemplo, ¿como representamos unha cantidade analóxica onde só podemos almacenar ceros e uns?, ¿será o mesmo almacenar unha cantidade enteira que unha real?, ¿como se fai para representar cantidades negativas onde só podemos ter ceros e uns?.

A continuación daremos sentido a todas estas preguntas.

1.1. Representación de números enteiros sin signo

Trátase do caso máis simple, onde o número convírtese a binario directamente. Nos S7-1200 temos varias posibilidades:

- **USInt (Unsigned Small Integer)**:Represéntase mediante un conxunto de 8 bits (1 byte), polo que, se facemos a conversión de binario a decimal temos 2^8 combinacións diferentes, o que nos da 256 posibilidades. Logo un dato **USInt** pode conter números enteiros (sempre positivos) comprendidos entre 0 e 255.
- **UInt (Unsigned Integer)**:Represéntase mediante un conxunto de 16 bits (2 bytes ou unha palabra), polo que teriamos 2^{16} posibilidades de representación. Se facemos os cálculos temos 65536 combinacións diferentes. Entón un dato **UInt** pode conter números enteiros (sempre positivos) comprendidos entre 0 e 65535.
- **UDInt (Unsigned Double Integer)**:Temos neste caso un conxunto de 32 bits (4 bytes), polo que, se facemos cálculos (2^{32}) obtemos 4294967296 combinacións posibles. Deste xeito, nun dato de tipo **UDInt** podemos almacenar números enteiros positivos comprendidos entre 0 e 4294967295.

1.2. Representación de números enteiros con signo

Como só podemos almacenar ceros e uns, cando apareceron os dispositivos microprocesadores, prantexouse a necesidade de "inventar" algún sistema para representar números negativos.

Foron desenvolvéndose tres métodos (**módulo e signo**, **complemento a un** e **complemento a dous**) e, aínda que son diferentes entre sí, os tres baséanse no mesmo principio: Utilizar o primeiro bit empezando pola esquerda para diferenciar se o número é positivo ou negativo.

Se o bit é un cero, o número considérase positivo.

Se o bit é un un, o número considérase negativo.

Hai que ter en conta polo tanto que, se traballamos con números con signo, para a mesma cantidade de bits que dispoñiamos nos enteiros sen signo, temos un bit menos para o módulo, logo as cantidades a representar son as mesmas que no caso anterior pero a metade positivas e a metade negativas.

Para 8 bits (7 para o módulo e 1 para o signo) temos:

$$2^7=128$$

O número positivo máis grande que se pode representar neste caso é:

$$0\ 1111111 = 1\cdot2^6+1\cdot2^5+1\cdot2^4+1\cdot2^3+1\cdot2^2+1\cdot2^1+1\cdot2^0 = 64+32+16+8+4+2+1 = 127$$

O sistema de representación de números negativos destes autómatas é o de complemento a 2, que consiste en converter todos os ceros en uns e viceversa, e ao resultado sumarlle 1.

Por exemplo, se queremos saber como se representaría o número -45 neste sistema colleriamos o 45 positivo:

$$0\ 0101101$$

Cambiamos os ceros por uns e viceversa:

$$1\ 1010010$$

E sumámoslle 1:

$$1\ 1010010 + 1 = 1\ 1010011$$

Para facer o proceso ao revés, é dicir, a partir dunha representación binaria dun número negativo, averiguar de que número se trata, fariamos o proceso contrario. Por ejemplo:

$$1\ 1010011$$

Restámoslle 1:

$$1\ 1010011 - 1 = 1\ 1010010$$

E cambiamos ceros por uns e uns por ceros:

$$0\ 0101101$$

O módulo do número é:

$$0\cdot2^7+0\cdot2^6+1\cdot2^5+0\cdot2^4+1\cdot2^3+1\cdot2^2+0\cdot2^1+1\cdot2^0 = 32+8+4+1 = 45$$

Segundo este criterio, o número negativo máis pequeno que se pode representar empregando 8 bits sería o:

$$1\ 0000000$$

Desfagamos o complemento para ver cal é:

$$1\ 0000000 - 1 = 0\ 1111111$$

Agora cambiamos ceros por uns e viceversa:

$$1\ 0000000$$

Temos polo tanto:

$$1\cdot2^7 = 128$$

Logo, o número máis pequeno que podemos representar con 8 bits é o -128.

Segundo este razonamento, temos, nos S7-1200:

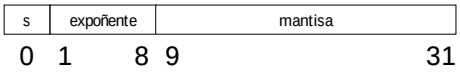
- **SInt (Signed Integer):** Empregamos 8 bits (7+1), polo que as cantidades a representar van desde o -128 ata o 127.
- **Int (Integer):** Empregamos 16 bits (15+1), polo que teriamos as cantidades desde -32768 ata 32767.
- **DInt (Double Integer):** Con 32 bits (31+1), entón podemos representar números comprendidos entre -2147483648 ata 2147483647.

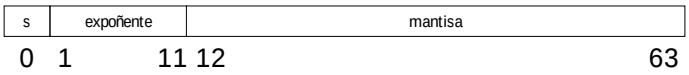
1.3. Representación de números reais

Neste caso ideouse un algoritmo para representar números reais en coma flotante (IEEE754), que ten dúas variantes: números de simple precisión (32 bits) e dobre precisión (64 bits).

Estes autómatas poden traballar con calquera dos dous tipos de datos.

O algoritmo consiste en utilizar o bit máis á esquerda para o signo, os seguintes bits para definir un exponente, e o resto unha mantisa.





Mediante uns exemplos veremos o funcionamento do algoritmo.

1.3.1. Conversión dun número en formato IEEE754 a coma flotante

Supoñamos que temos unha cadea de 32 bits que representa un dato en coma flotante en formato IEEE 754 simple precisión tal como a seguinte:

0 1 0 0 0 0 1 0 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 0 0 0 1 0 1 0 0 1

Se queremos saber de que número se trata procedemos do seguinte xeito:

O signo determinámolo igual que para números negativos (0 = positivo, 1 = negativo).

A combinación de bits correspondente ao expoñente é $1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ _{\angle 2} = 132_{\angle 10}$, polo tanto para tomamos como potencia de 2:

$$2^{132-127} = 2^5$$

Como mantisa temos, considerando as potencias de 2 sucesivas:

$$2^{20}+2^{19}+2^{18}+2^{16}+2^{14}+2^{12}+2^{11}+2^{10}+2^5+2^3+2^0 = 1924137$$

Tomamos:

$$(1+1924137\cdot 2^{-23}) = 1{,}22937$$

O número que buscamos é o:

$$+\ 1{,}22937\cdot 2^5 = 39{,}34$$

1.3.2. Conversión dun número en coma flotante a formato IEEE 754

O proceso inverso é algo máis laborioso.

Supoñamos que queremos converter o mesmo número anterior.

Primeiro miramos o signo tomando un 0 para positivos e 1 para negativos.

Buscamos un número *n* tal que o noso número dividido entre 2^n nos dea un cociente maior ou igual que 1 e menor que 2.

No caso anterior:

$$39{,}34 / 2^5 = 1{,}229375$$

Polo tanto n = 5

Para decidir o expoñente facemos á inversa, e dicir, sumámoslle 127 ao n que atopamos:

$$\text{Expoñente: } 127+5 = 132_{\angle 10} = 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0_{\angle 2}$$

O traballo máis laborioso consiste en averiguar a mantisa. Para esto collemos a parte decimal que nos quedou ao facer a división anterior e transformámola en binario. Collemos, polo tanto 0,229375 que transformaremos seguindo as sucesivas potencias negativas de 2 desde a potencia -1 ata a -23.

Nº	Potencia de 2	Bit	Diferencia
-1	0,5	0	0,229375
-2	0,25	0	0,229375
-3	0,125	1	0,104375
-4	0,0625	1	0,041875
-5	0,03125	1	0,010625
-6	0,015625	0	0,010625
-7	0,0078125	1	0,0028125
-8	0,00390625	0	0,0028125
-9	0,00195313	1	0,000859375
-10	0,00097656	0	0,000859375
-11	0,00048828	1	0,000371094
-12	0,00024414	1	0,000126953
-13	0,00012207	1	4,88281E-06
-14	6,1035E-05	0	4,88281E-06
-15	3,0518E-05	0	4,88281E-06

-16	1,5259E-05	0	4,88281E-06
-17	7,6294E-06	0	4,88281E-06
-18	3,8147E-06	1	1,06812E-06
-19	1,9073E-06	0	1,06812E-06
-20	9,5367E-07	1	1,14441E-07
-21	4,7684E-07	0	1,14441E-07
-22	2,3842E-07	0	1,14441E-07
-23	1,1921E-07	0	1,14441E-07

Como vemos pode haber un pequeno erro nos últimos bits.

Compoñendo o número neste caso queda como:

0 1 0 0 0 0 1 0 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 0 0 0 1 0 1 0 0 0

Polo tanto, nos autómatas S7-1200 temos:

- **Real:**Números en coma flotante de simple precisión (32 bits). Podemos representar números negativos desde $-3{,}402823 \cdot 10^{38}$ ata $-1{,}175495 \cdot 10^{-38}$, e números positivos desde $1{,}175495 \cdot 10^{-38}$ ata $3{,}402823 \cdot 10^{38}$.
- **LReal:**Números en coma flotante de dobre precisión (64 bits). Neste caso podemos representar cantidades negativas desde $-1{,}7976931348623158 \cdot 10^{308}$ ata $-2{,}2250738585072014 \cdot 10^{-308}$, e cantidades positivas desde $2{,}2250738585072014 \cdot 10^{-308}$ ata $1{,}7976931348623158 \cdot 10^{308}$

2. Sistemas de numeración

- **Decimal:** O que emprega o ser humano para representar cantidades (0-9).
- **Binario:** O que empregan os dispositivos dixitais como PCs ou PLCs (0-1).
- **Hexadecimal:** Sistema “inventado” para traballar con dispositivos dixitais e representar cantidades elevadas de xeito simple.

Decimal	Binario	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

A conversión entre os sistemas binario-hexadecimal é simple. Consiste en coller grupos de catro bits (empezando pola dereita) e mirar na táboa anterior a equivalencia.

Exemplo: Converter a hexadecimal o número binario: 1011010101011

Completamos tantos ceros á esquerda como falten para ter un múltiplo de catro:

0001 0110 1010 1011

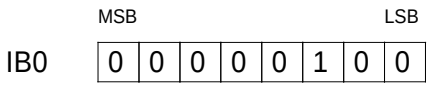
Buscamos na táboa a equivalencia: 16AB

3. Formato byte, palabra e dobre palabra

Como se indicou ao principio deste documento, nos autómatas S7-1200 os datos almacénase na memoria en formatos de 8, 16, 32 e 64 bits. Chámanse byte (8 bits), palabra (16 bits), dobre palabra (32 bits) e cuádruple palabra (64 bits).

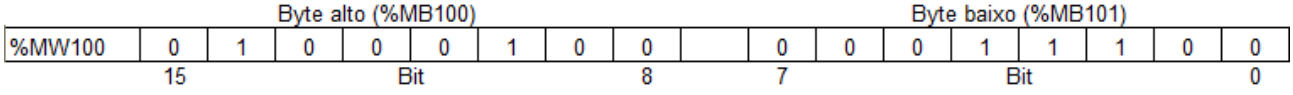
Os datos gárdanse do seguinte xeito.

- **Byte**: Como vemos na figura seguinte, o bit menos significativo colócase á dereita, de xeito que, se temos a entrada %I0.2 activada e as demáis desactivadas, teremos:



Se nese momento consultamos o valor de %IB0 obteremos o valor 4 (2^2).

- **Palabra**: O byte máis alto colócase primeiro e logo o byte máis baixo, como vemos na figura seguinte.



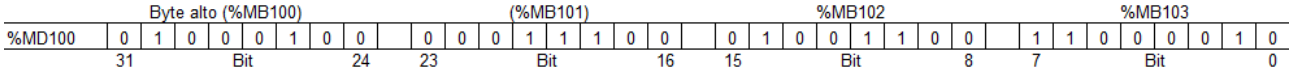
Se o dato que temos almacenado na figura anterior é un **Uint** entón trátase do 17436 ($1\cdot2^{14}+1\cdot2^{10}+1\cdot2^4+1\cdot2^3+1\cdot2^2$).

Pero tamén podemos acceder aos bytes por separado, e dicir, a %MB100 e %MB101. Nese caso teriamos en %MB100 o dato 68 ($1\cdot2^6+1\cdot2^2$) e na posición %MB101 o dato 28 ($1\cdot2^4+1\cdot2^3+1\cdot2^2$).

Temos que ter en conta que 68·256+28 = 17436

En todo caso seremos nós como programadores os que teñamos que ter coidado en como se interpreta a información coa que estamos a traballar.

- **Dobre palabra**: O razoamento é similar ao anterior pero con catro bytes.



Neste caso podemos atoparnos con varias situacións: Se a secuencia de 32 bits é considerada como un número enteiro dobre si signo (**UDInt**) enton sería o 1.142.705.346 (sumemos as potencias de 2 que conteñen un 1 na figura e obtemos este número).

Pero poidera tratare dun número **Real** de simple precisión en formato IEEE754. Neste caso sería o 625,199340820313.

Seguimos podendo acceder aos bytes por separado, e neste caso ás palabras tamén por separado. Haberá que ter coidado coas posicións de memoria ás que accedemos.

No exemplo da figura teremos claro que os bytes teñen os valores de maior a menor (68, 28, 76 e 194).