SIEMENS

Använda pekare i S7-300/400

Det här exemplet beskriver hur man kan använda indirekt adressering med pekare i S7-300/400 system. Både hur pekare är uppbyggda och hur de används beskrivs.

OBS!
Detta är ett tips/exempel på lösning som skall hjälpa användaren att komma igång och se möjligheterna att använda SIMATIC.
Du måste själv anpassa detta tips/exempel till din applikation.

Siemens tar inget ansvar om material eller personal skadas i samband med användning av detta tips/exempel. Vi kan heller inte garantera att innehållet är helt felfritt och vi förbehåller oss rätten att ändra tipset/exemplet vid behov.

Det finns flera olika pekareformat:

16 bitars pekare	- är ett 2 bytes format. Används för att peka ut block, timrar och countrar.
32 bitars pekare	- är ett 4 bytes format. Används för att peka ut operander.
Pointer	- är ett 6 bytes format. Samma som 32 bitars pekaren, men här är DB:t inkluderat. Detta är inte detsamma som datatypen pointer.
Any pointer	- är ett 10 bytes format. ANY formatet används för att peka ut ett data område samt tala om dess storlek.

Ackumulatorn i plc:n är på 32 bitar. Det betyder att man endast kan hantera 32 bitar åt gången. Dom pekarna man använder när man programmerar är 32 bitars pekaren och 16 bitars pekaren.
Pekarna av formaten ANY och Pointer använder man oftast bara vid parametrering av block.

16 bitars pekare

16 bitars pekare formatet är ett vanligt heltal. Man har det för att kunna peka ut block, timrar och räknare.

Block

De blocken man kan peka ut är:

- Datablock
- Funktioner – utan parametrar
- Funktionsblock – utan parametrar och utan statiska variabler

Pekaren som man kan använda sig av är:

- MW - Memory ord
- DBW – Datablocks ord
- DIW – Datablocks ord
- LW – Lokaldata ord

Syntaxen ser ut som följer:

OPN DB[MW10]

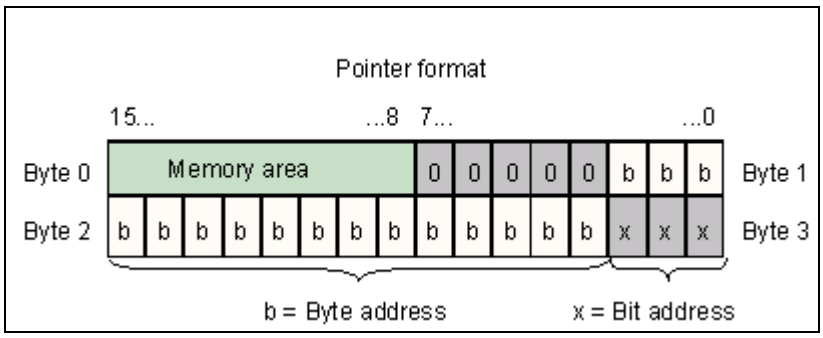
Om MW10 har värdet 22 öppnas DB22.

Andra exempel är:

OPN DI [DBW10]	(DB:n till DBW 10 måste redan vara öppen)
UC FC [MW10]	
CC FB [MW10]	(OBS – Inget DB anrop. STAT variabler kan inte användas)
A T [LW0]	
A C [DIW 20]	

32 bitars pekare

32 bitars pekaren används för att peka ut en adress i plc:n. Det finns 2 varianter på 32 bitars pekaren, en med adressidentifierare och en utan. Det enklaste sättet att beskriva en 32 bitars pekare är att använda datatypen pointer (P#M11.3).



Byte 0	- Vilken dataarea som datan ligger i (adressarea identifierare):	
B#16#00		Ingen adressidentifierare används
B#16#80	P	Periferi
B#16#81	I	Ingång
B#16#82	Q	Utgång
B#16#83	M	Minne
B#16#84	DB	Datablock
B#16#85	DI	Instans Datablock
B#16#86	L	Lokal data
B#16#87	V	Föregående lokaldata

Byte 1 - 3 mest signifikanta bitarna på byte adressen.

Byte 2 - Mellandelen på byte adressen.

Byte 3 - 5 minst signifikanta bitarna på byte adressen, samt bit adressen.

Anledningen till att byte och bit adresserna hamnar som dom gör är att om man skall peka ut en bit via en integer så är det bara att ladda integern, sen är det en relativ pekare direkt.

Exempel:

Integer:	Binär kod:	Pointer:
5	0101	P#0.5
7	0111	P#0.7
8	1000	P#1.0
12	1100	P#1.4

Adressregister

I Step7 finns det speciella minnesceller för att hantera pekare vid programmering. De heter Adress Register 1, AR1 och Adress Register 2, AR2. AR2 används internt vid användning av Funktions Block. AR2 får inte användas i FB (om man inte är absolut säker på hur det fungerar).
Man måste inte använda sig av AR registren, men kan använda sig av ett dubbelord istället.
Fördelen med AR är att:

- AR finns med i avbrottsstacken.
- Det finns kommando där man direkt kan plussa på ett värde på AR.
- Man kan lätt använda sig av en offset.
- Vid byte av prioritetsgrupp hanterar systemet det så att AR inte skrivs över även om man använder ”samma” AR i interrupt rutinen.
- Man kan använda adressidentifierare i pekarna.

När man adresserar med AR måste man alltid ange en offset.

Kommando för att hantera adressregistren

LAR 1	- Kopierar Acku1 till AR1.
LAR 2	- Kopierar Acku1 till AR2.

Alla följande kommando fungerar även för AR2.

LAR 1 P#M10.0	- Kopierar adressen M10.0 till AR1.
LAR 1 AR2	- Kopierar AR2 till AR1.
LAR 1 DB10.DBD20	- Kopierar DB10.DBD20 till AR1.
LAR 1 DI20.DBD30	- Kopierar DI20.DBD30 till AR1.
LAR 1 MD 180	- Kopierar MD180 till AR1.
LAR 1 LD 6	- Kopierar LD 6 till AR1.

TAR 1	- Kopierar AR1 till Acku1.
TAR 2	- Kopierar AR2 till Acku1.

Till TAR kommandot kan man även använda sig av dom ovan angivna direktadresserade kommandona.

+AR1	- Summerar Acku1 med AR1. Summan läggs i AR1.
+AR2	- Summerar Acku1 med AR2. Summan läggs i AR2.

+AR1 P#2.0	- Summerar ett fast värde till AR1.
+AR2 P#2.0	- Summerar ett fast värde till AR2.

När man hanterar en pekare är det enklast att använda sig av datatypen pointer, P#0.0.

Exempel på AR adressering

Avfrågning av en bool:

```
L P#12.3              // Ladda adressen 12.3 till Acku1.
LAR 1                 // Adressen 12.3 lägger sig i AR1.
A I[AR1,P#0.0]        // Frågar av om I 12.3 = 1.

LAR 1 P# M 203.4      // Laddar adressen M 203.4 i AR1(med area identifierare).
A   [AR1,P#0.0]       // Frågar av om M203.4 = 1.
```

Avfrågning av byte, ord och dubbelord

```
L L#4                 // Laddar DINT 4 till Acku1
SLD 3                 // Skiftar 3 steg till vänster. Detta gör om 4:an till
                      pekareformat.
LAR 1                 //Flyttar P#4.0 till AR1.
L MB[AR1,P#0.0]       // Laddar MB 4 i acku1.
L IW[AR1,P#0.0]       // Laddar IW 4 i acku1.
L LD[AR1,P#0.0]       // Laddar LD 4 i acku1 (Lokaldata).
```

Offset vid AR användning

```
LAR 1 P# M 100.0      // Laddar adressen M 100.0 i AR1
L   W[AR1,P#0.0]      // Laddar MW 100 till Acku1.
L   W[AR1,P#50.0]     // Laddar MW 150 till Acku1( 100.0 + 50.0).
```

Offseten är bra att använda om man skall flytta data från ett ställe till ett annat.

Minnespekare

Man kan även används sig av ett dubbelord istället för AR som pekareminne.

Minnesareor som man kan använda är:

- MD - Minnes dubbelord.
- DBD - Data dubbelord. DB måste redan vara öppet.
- DID - Instans data dubbelord. DI måste redan vara öppet.
- LD - Lokaldata dubbelord.

Adressering med dubbelord fungerar i stort sett likadant som adressering med AR, med skillnaderna:

- AR finns i avbrottsstacken. Använder man minnespekare finns dom inte direkt i avbrottsstacken.
- Motsvarande kommando till +AR1 finns inte för minnespekare.
- Vid byte av prioritetsgrupp, te x interrupter, måste man själv se till att minnespekaren inte blir överskrivet.
- Man får inte någon offset automatiskt så som när man använder AR1.
- Man kan inte använda sig av pekare med adressarea identifierare när man använder sig av minnespekare.

Exempel på minnespekare adressering

Avfrågning av en bool:

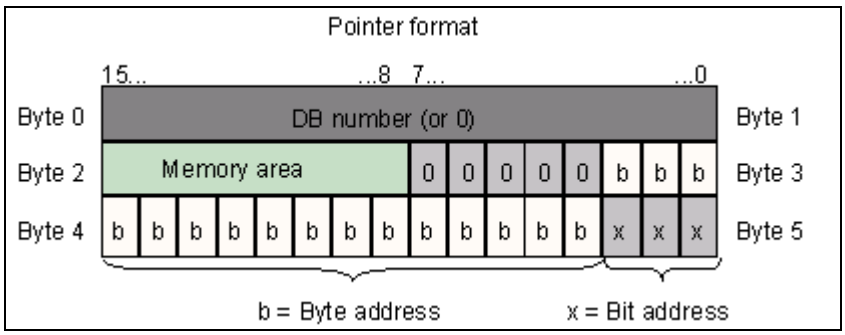
```
L P#12.3          // Ladda adressen 12.3 till Acku1.
T MD100           // Adressen 12.3 lägger sig i MD100.
A I[MD100]        // Frågar av om I 12.3 = 1.
```

Avfrågning av byte, ord och dubbelord

```
L L#4             // Laddar DINT 4 till Acku1
SLD 3             // Skiftar 3 steg till vänster. Detta gör om 4:an till
                  pekareformat.
T LD0             // Flyttar P#4.0 till Lokaldata 0.
L MB[LD0]         // Laddar MB 4 i acku1.
L IW[LD0]         // Laddar IW 4 i acku1.
L LD[LD0]         // Laddar LD 4 i acku1 (Lokaldata).
```

Pointer

Detta format används när man även måste ange vilket DB som ett dataord ligger i. Eftersom dataformatet är på 6 bytes används det vid parametrering och inte programmering.
För beskrivning av formatet hänvisas även till 32 bitars pekaren. Den är helt identisk förutom DB numret, som anges i byte 0 och 1. DB numret anges som en integer.



Byte 0,1 - DB numret till den datablocksarean man vill peka ut.

Byte 2 - Vilken dataarea som datan ligger i (adressarea identifierare):

B#16#80	P	Periferi
B#16#81	I	Ingång
B#16#82	Q	Utgång
B#16#83	M	Minne
B#16#84	DB	Datablock
B#16#85	DI	Instans Datablock
B#16#86	L	Lokal data
B#16#87	V	Föregående lokaldata

Byte 3,4,5 - bit och byte adressen till det som pekats ut.

Exempel på Pekare

Pekaren skall läggas på ett ben som skall peka ut en startadress på en viss area. Detta kan t ex vara källområdet till FC81 – IBLKMOV. IBLKMOV ligger i Standardbiblioteket under TI-S7 converting blocks. På benet skall endast startadressen anges för var pekaren (som pekar ut var källområdet ligger) finns. Blocket förutsätter att pekaren är i formatet Pointer.

På benet anges te x DB10.DBX12.0.
Man vill att källan skall vara DB22.DBX30.0
Pekaren byggs upp som följer:

```
L 22                  // 22 till Acku1.
T DB10.DBW12          // Lägg det i byte 0 och 1 för pekaren.
L P#30.0              // Ladda adressen 30.0 (vanligen är denna variabel).
T DB10.DBD14          // Lägg adressen 30.0 i byte 3,4,5 och 6 i pekaren.
L B#16#84             // Kod för DB area.
T DB10.DBB14          // Lägg koden i byte 3 i Pekaren.
```

Om man använder samma aderssidentifierare hela tiden kan man skriva ihop det med adressen direkt. Man slipper då ange koden för minnesarean man skall använda. Samma pekare som ovan kan då beskrivas så här:

```
L 22                  // 22 till Acku1.
T DB10.DBW12          // Lägg det i byte 0 och 1 för pekaren.
L P#DBX 30.0          // Ladda adressen 30.0 (vanligen är denna variabel).
T DB10.DBD14          // Lägg adressen 30.0 i byte 3,4,5 och 6 i pekaren.
```

ANY Pointer

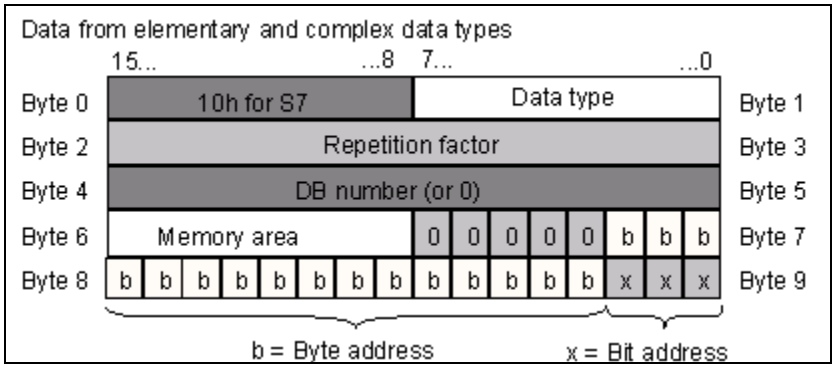
ANY pointern används när man vill peka ut en minnesarea i plc:n. Man vill dels berätta vilken startadress datan har, dels hur stor den är. Detta formatet används inte när man programmerar, utan används endast vid parametrering av ett block, te x SFC 20.
ANY pointern är 10 bytes lång. Detta gör att man inte kan gå via Acku1 när man skall flytta pekaren.

Om man vill att datan i pekaren skall vara dynamisk så får man mellanlagra det i blockets lokaldata, för att sedan anpassa den där.

För att definiera pekaren måste man antingen använda sig av lokaldata, IN parameter eller IN/OUT parameter. Man kan även skicka en ANY pointer vidare via en OUT parameter.

När man har definierat pekaren kan man ange den via dess symbol.

Pekaren är uppbyggd som följer:



Byte 0 - Alltid 10h i S7.

Byte 1 - Data typ som pekas ut:

B#16#00	NIL	Noll pekare
B#16#01	BOOL	Bitar (1 bit)
B#16#02	BYTE	Byte (8 bitar)
B#16#03	CHAR	Tecken (8 bitar)
B#16#04	WORD	Ord (16 bitar)
B#16#05	INT	Integer (16bitar)
B#16#06	DWORD	Dubbel ord (32 bitar)
B#16#07	DINT	Dubbel Integer (32 bitar)
B#16#08	REAL	Flyttal (32 bitar)
B#16#09	DATE	Datum (32bitar)
B#16#0A	TOD	Klockslag (Time of day – 32 bitar)
B#16#0B	TIME	Time format (32 bitar)
B#16#0C	S5TIME	S5TIME – Timer format (16bitar)
B#16#0E	DT	Datum och tid (Date and time – 64 bitar)
B#16#13	STRING	Sträng

Byte 2-3 - Antalet av datatypen som pekas ut.

Byte 4-5 - Vilket DB som pekas ut.

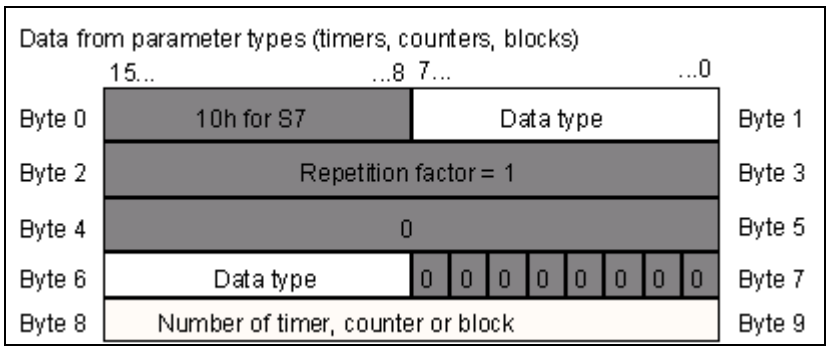
Byte 6 - Vilken dataarea som datan ligger i (adressarea identifierare):

B#16#80	P	Periferi
B#16#81	I	Ingång
B#16#82	Q	Utgång
B#16#83	M	Minne
B#16#84	DB	Datablock
B#16#85	DI	Instans Datablock
B#16#86	L	Lokal data
B#16#87	V	Föregående lokaldata

Byte 7, 8, 9 - Bit och byte startadress

Föregående lokaldata (V) används om man t ex vill läsa ut lokaldatan från ett block med SFC20. När SFC20 körs är det ju dess lokaldata som avses när man använder L. Vill man ha lokaldatan från blocket som SFC20 anropas ifrån måste man använda sig av V.

När man skall peka ut ett block, timer eller räknare och vill använda sig av en ANY pointer, skall den vara strukturerad som följer:



Byte 0 - 10 hex för S7.

Byte 1 - Data typ som pekas ut:

B#16#17	BLOCK_FB	Funktions block
B#16#18	BLOCK_FC	Funktion
B#16#19	BLOCK_DB	Datablock
B#16#1A	BLOCK_SDB	Standard Datablock
B#16#1C	COUNTER	Räknare
B#16#1D	TIMER	Timer

Byte 2-3 - Skall vara 1.

Byte 4-5 - Skall vara 0.

Byte 6 - Data typ som pekas ut:

B#16#17	BLOCK_FB	Funktions block
B#16#18	BLOCK_FC	Funktion
B#16#19	BLOCK_DB	Datablock
B#16#1A	BLOCK_SDB	Standard Datablock
B#16#1C	COUNTER	Räknare
B#16#1D	TIMER	Timer

Byte 7 - Skall vara 0.

Byte 8-9 - Numret på blocket, timern eller räknaren.

Exempel på ANY pointer

Pekaren skall läggas på ett ben som skall peka ut en startadress på en viss area. Detta kan t ex vara källområdet till SFC20 – BLKMOV. På benet anges en lokaldata variabel som ligger på adresserna LB0-LB9.

Arean som skall pekas ut är:
Startadress: DB24.DBW8
Det skall vara av typen ord och arean skall vara 20 ord långt.

Blocket förutsätter att pekaren är i formatet ANY.

Pekaren byggs upp som följer:

```
L B#16#10          // Alltid 10 hex för Step7.
T LB 0             // Lägg 10 hex i LB 0.

L B#16#04          // Kod 04 hex för Word.
T LB 1             // Lägg det i LB 1.

L 20               // 20 ord.
T LW 2             // Lägg det i LW 2.

L 24               // DB 24.
T LW 4             // Lägg det i LW 4.

L P#DBX8.0         // Startadress DBX8.0
T LD 6             // Lägg det i LD 6.
```

Det kan däremot vara lite farligt att adressera absoluta adresser såsom är gjort i detta exempel. Det räcker med att man lägger till en TEMP variabel innan den aktuella ANY variabeln för att programmet skall sluta fungera.

Säkert sätt att bygga en ANY pointer

Istället för att gå in på absoluta adresser på där ANY pointern ligger kan man först läsa ut startadressen, för att sedan skriva in ny data till den.

Det har skapats en TEMP variabel med namnet TEMP_Any. Den är av typen ANY.

```
L    P##TEMP_Any        // Ladda TEMP_Any. Den absoluta startadressen läggs då
                        i Akku1.
LAR1                    // Lägg TEMP_Any:s startadress i AR1

L    B#16#10            // Lägg koden 10h i byte noll
T    B [AR1,P#0.0]

L    B#16#4             // Lägg koden 4h = Word, i byte ett
T    B [AR1,P#1.0]

L    25                 // Lägg 25 (i detta fall 25 word) i ord 2
T    W [AR1,P#2.0]

L    37                 // Lägg 37 (i detta fall DB37) i ord 4
T    W [AR1,P#4.0]

L    P#16.0             // Lägg startadressen 16.0 i dubbelordet 6
T    D [AR1,P#6.0]

L    B#16#84            // Lägg koden 84h = Datablock i byten 6
T    B [AR1,P#6.0]
```

Exempel på hur man flyttar en ANY pointer

Om man vill använda sig av en ANY pointer som en IN variabel i en FC eller FB behöver man ibland flytta den till blockets TEMP area. I just detta exempel användes AR2, så detta exempel skall endast användas i en FC. Det skall inte användas i en FB. FB använder AR2 internt i blocket.
Ett exempel en sådan flytt kan vara om man vill använda ANY pointern som en parameter på t ex SFC20. SFC20 accepterar inga formaloperander som är deklarerade som IN eller IN/OUT. SFC20 accepterar däremot TEMP variabler.

Flyttningen kan göras såhär:

I blockhuvud har följande variabler skapats:

Interface typ:	Namn:	Data typ:
IN	IN_Any	ANY
TEMP	TEMP_Any	ANY

```
L    P##IN_Any          // Den absoluta startadressen för var pekaren ligger (inte
                           vad den innehåller) laddas till Akku1.
LAR1                    // Startadressen för IN parametern läggs i AR1.
L    P##TEMP_Any        // Den absoluta startadressen för var pekaren ligger
                           laddas i Akku1.
LAR2                    // Startadressen för TEMP variabeln läggs i AR2.

L    W [AR1,P#0.0]      // Laddar bytarna 0 och 1 från IN pointern till Akku1.
T    LW [AR2,P#0.0]     // Skriver bytarna 0 och 1 till TEMP pointern.

L    W [AR1,P#2.0]      // Laddar bytarna 2 och 3 från IN pointern till Akku1.
T    LW [AR2,P#2.0]     // Skriver bytarna 2 och 3 till TEMP pointern.

L    W [AR1,P#4.0]      // Laddar bytarna 4 och 5 från IN pointern till Akku1.
T    LW [AR2,P#4.0]     // Skriver bytarna 4 och 5 till TEMP pointern.

L    W [AR1,P#6.0]      // Laddar bytarna 6 och 7 från IN pointern till Akku1.
T    LW [AR2,P#6.0]     // Skriver bytarna 6 och 7 till TEMP pointern.

L    W [AR1,P#8.0]      // Laddar bytarna 8 och 9 från IN pointern till Akku1.
T    LW [AR2,P#8.0]     // Skriver bytarna 8 och 9 till TEMP pointern.
```

Nu är TEMP pointern färdig för att användas.