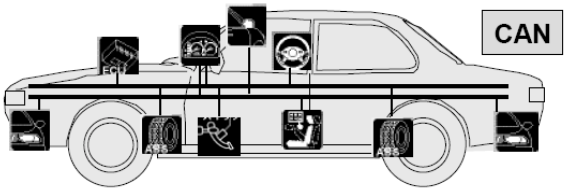
PLC – Introduzione

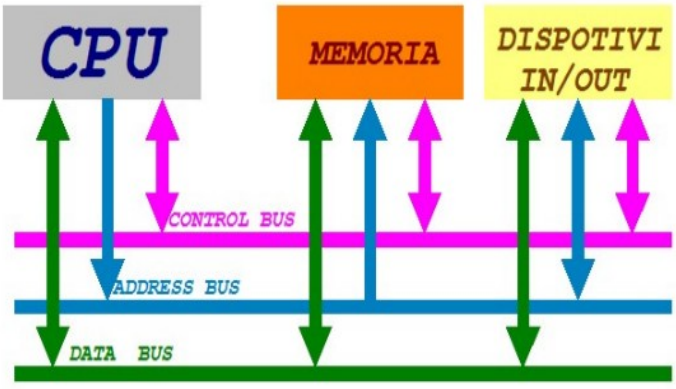
PLC è l'acronimo di **P**rogrammable **L**ogic **C**ontroller, un dispositivo programmabile molto utilizzato nel settore dell'atuomazione industriale.

Come tutti i dispositivi programmabili, il PLC è dotato di una CPU, di una memoria e di dispositivi di input ed output per interagire con il mondo esterno.

Essendo un dispositivo nato per controllare macchinari e dispositivi automatici, la sua interazione con il mondo esterno avviene essenzialmente leggendo lo stato di sensori o controllando degli attuatori, ma la sua evoluzione nel corso degli anni ha reso il PLC sempre più ricco di risorse e di potenzialità, svillupando nel tempo non solo la possibilità di controllare sensori ed attuatori, ma anche di interagire e comunicare con svariati **BUS di comunicazione**.

Apriamo una piccola parentesi sulla parola BUS. Con questo termine indichiamo un canale hardware, composto da diversi collegamenti elettrici, dove transitano dei dati. Tutti noi conosciamo l'USB (Universal Serial Bus) utiizzato per connettere le nostre Pendrive al PC, potremmo fare altri esempi un po' meno noti, come il CANBus, utilizzato nel settore automobilistico per connettere tra di loro i dispositivi elettronici di un'autovettura, o come il bus PCI express utilizzato per collegare delle schede aggiuntive nel nostro PC. Un BUS risponde a delle regole, definite protocolli, che specificano la tipologia dei segnali elettrici, le tempistiche e le priorità con cui essi vengono inviati agli altri dispositivi ad esso collegati.



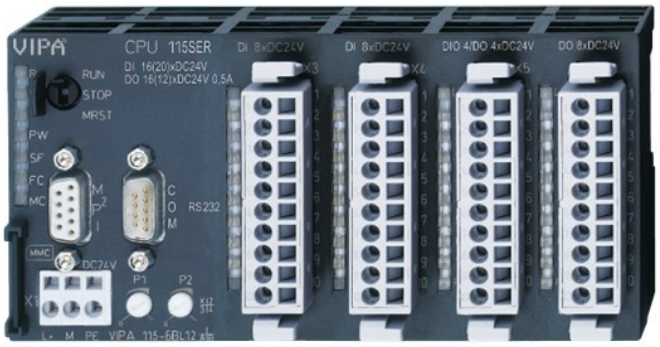


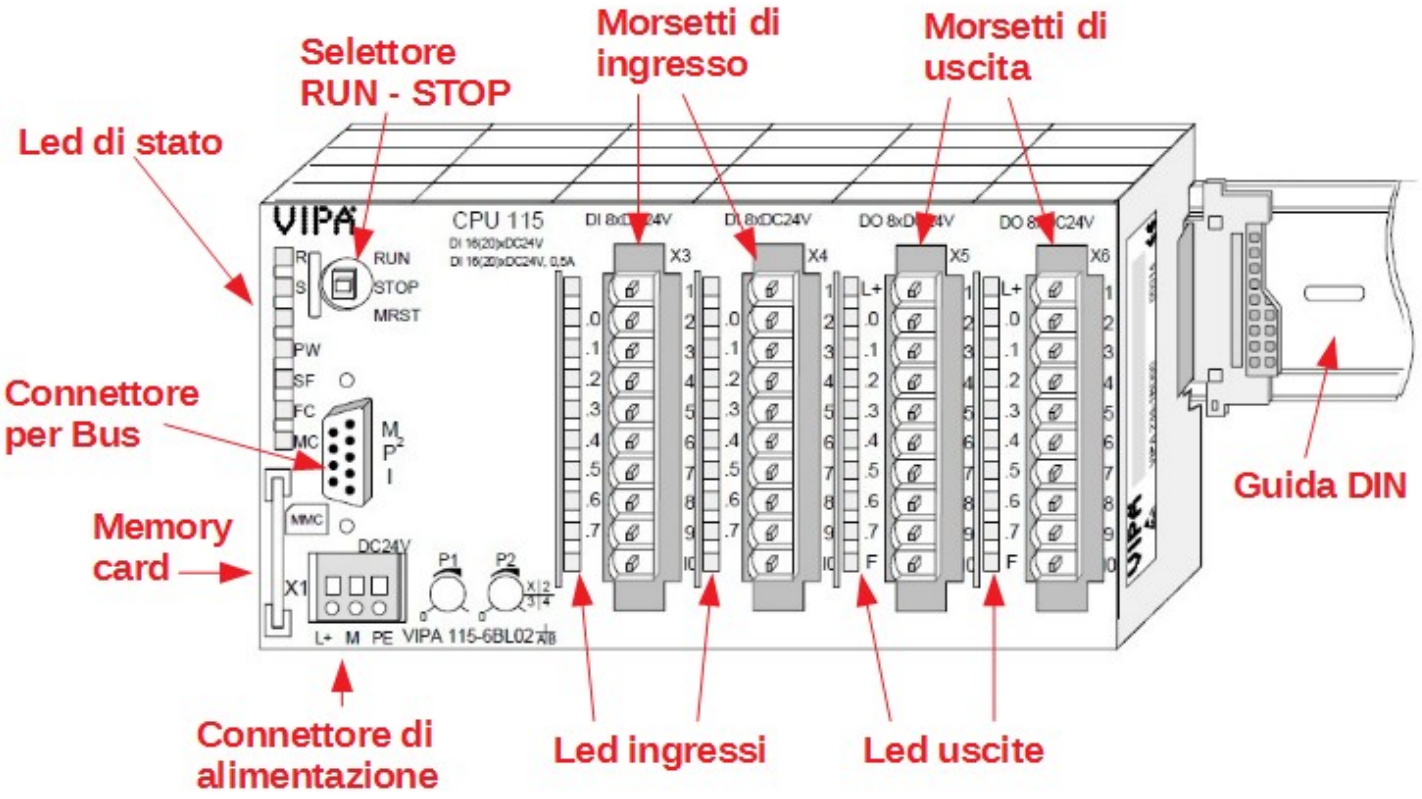
I moderni PLC sono dotati di connettori elettrici per la connessione ad uno o più BUS di comunicazione, che consentono non solo il controllo di ingressi ed uscite digitali (sensori ed attuatori) o di ingressi ed uscite analogiche, ma anche l'interazione con altri dispositivi, come azionamenti per motori, pannelli operatore o altri PLC e PC.

Insomma oggi un PLC può comunicare con qualsiasi dispositivo utilizzando i più noti canali hardware che conosciamo, seriale RS232, seriale RS485, ethernet ecc…

Vista la possibilità di interagire con atri dispositivi, un PLC può sembrare simile ad un normale PC, ma c'è una differenza sostanziale. Mentre un PC è normalmente dotato di un sistema operativo, che gli consente di avviare diversi programmi anche simultaneamente, in un PLC non c'è alcun sistema operativo, o meglio c'è solo un programma (firmwmare) che si occupa di mandare in esecuzione il nostro algoritmo, cioè quel codice che abbiamo scritto e caricato nella memoria per fargli eseguire determinate operazioni.

Entriamo ora un po' più nel dettaglio della sua struttura, prendiamo per questo l'immagine di uno dei tanti PLC in commercio, in una versione abbastanza semplificata.





CONNETTORE PER BUS

Le parti indicate nell'immagine, sono in genere disponibili in ogni tipologia di PLC. Il connettore per il collegamento al BUS è in questo caso quello utilizzato per la programmazione, ma potrebbero esserci anche altre tipologie di connettori per il collegamento ad altri tipi di BUS. Nei più recenti PLC non è raro trovare anche un connettore ethernet, per controllare e programmare il dispositivo tramite rete.

CONNETTORE DI ALIMENTAZIONE

Ormai lo standard di alimentazione di tutti i dispositivi industriali, è di 24 Volt in corrente continua, il connettore presenta infatti i due terminali di alimentazione, positivo e negativo, ed un terminale per il collegamento equipotenziale (PE).

LED DI STATO

Tramite questi led in genere vengono fornite delle indicazioni sullo stato del PLC, se il programma è attivo o no, o se ci sono errori di qualche genere.

MEMORY CARD

Spesso è presente uno slot per memory card utilizzato per eeguire il salvataggio del programma o dei dati.
La perdita di dati potrebbe compromettere il funzionamento del dispositivo controllato, ed in un PLC la perdita dei dati e del programma può avvenire, in quanto la memoria è dotata di una batteria tampone che in caso di inutilizzo prolungato potrebbe non essere sufficiente a mantenere i dati.

SELETTORE RUN STOP

Anche questo selettore è spesso presente su un PLC, per spegnere o riavviare il programma.

GUIDA DIN

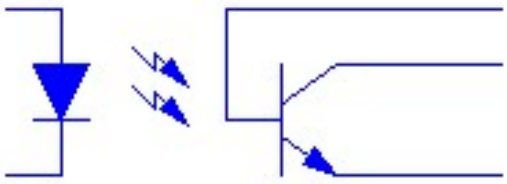
Essendo un componente industriale, esso andrà installato in un quadro elettrico insieme ad altri dispositivi. Per questo motivo un PLC ha la possibilità di essere montato sulla classica guida DIN per dispositivi da quadro.

MORSETTI E LED DI INGRESSO – USCITA

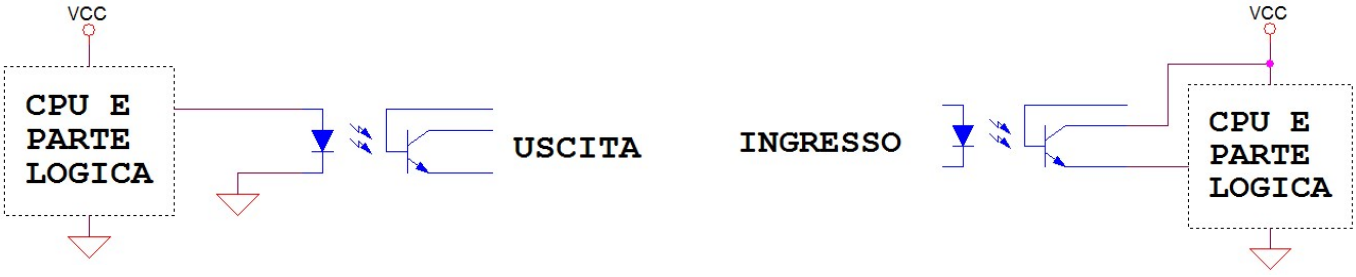
Come abbiamo detto, un PLC deve potersi connettere a sensori ed attuatori, cioè dispositivi di ingresso ed uscita digitali ed analogici.
Nella figura sono presenti 2 connettori con 8 terminali di ingressi per ognuno di loro e 2 con 8 terminali di uscita.
I moduli di input/output, come anche i moduli analogici, o quelli per funzioni particolari, possono essere aggiunti alla CPU, pertanto la configurazione è variabile a seconda delle necessità.
Vicino ai morsetti sono sempre presenti dei LED per vedere lo stato dell'ingresso o dell'uscita.

Tutti i tipi di PLC hanno gli ingressi ed uscite optoisolati, cioè separati elettricamente dalla parte logica, quella composta da CPU e memoria.

Un optoisolatore è un componente composto da un diodo emettitore e da un fototransistor, in pratica il segnale viene trasferito dai morsetti in ingresso collegati al diodo, ai morsetti di uscita collegati al transistor, mediante la luce emessa dall'emettitore.



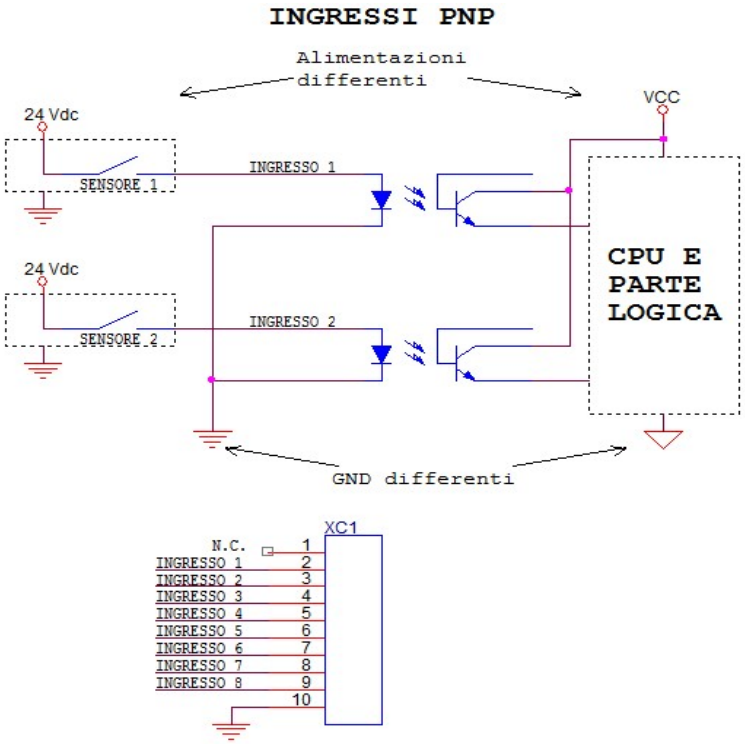
Questo consente di avere una totale separazione delle alimentazioni tra gli ingressi o uscite e la parte logica interna.



TIPOLOGIE DI INGRESSI, PNP, NPN E PUSH-PULL

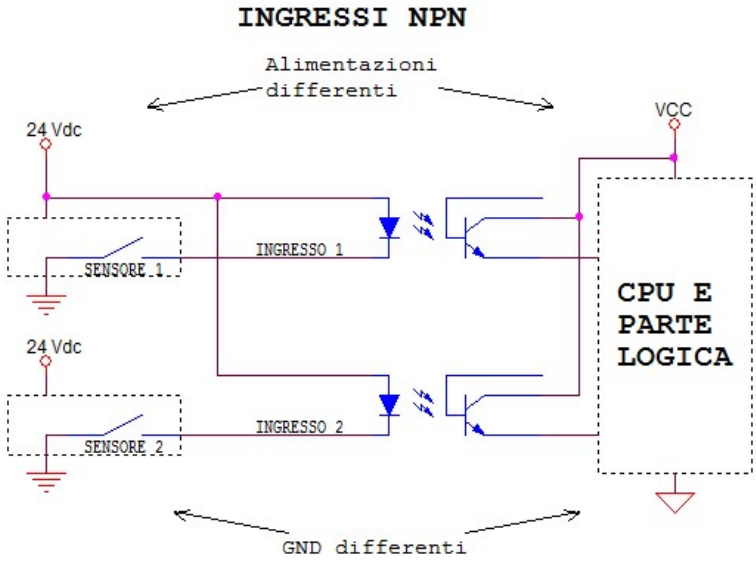
Un ingresso PNP, fornisce corrente all'ingresso del PLC, e l'optoisolatore trasferirà lo stato dell'ingresso alla parte logica.

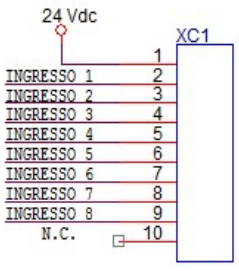
In questo caso sul connettore di ingresso del PLC, dovremo collegare il riferimento a 0Volt (GND) dell'alimentazione dei sensori.



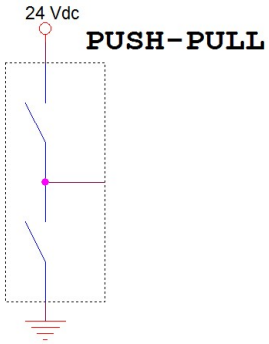
Un ingresso NPN, Assorbe corrente all'ingresso del PLC, e l'optoisolatore trasferirà lo stato dell'ingresso alla parte logica.

In questo caso sul connettore di ingresso del PLC, dovremo collegare il riferimento positivo dell'alimentazione dei sensori.





Esiste un'ulteriore tipologia di uscita di sensori collegabile ad entrambi gli ingressi, e cioè i sensori con uscita PUSH-PULL.

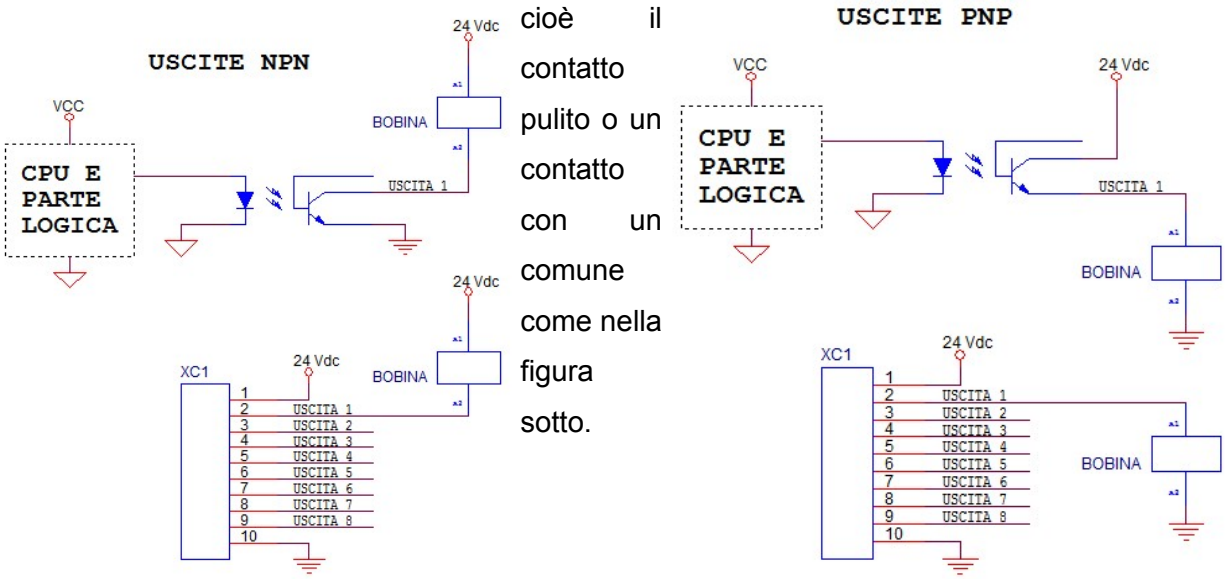


La configurazione più comune e più utilizzata, prevista anche dalle normative, è quella con ingressi PNP.

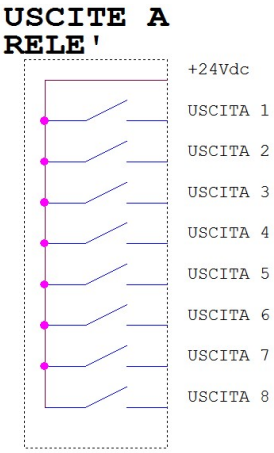
TIPOLOGIE DI USCITE PNP,NPN ED A RELE'

Analogo discorso per le uscite, un'uscita PNP eroga corrente verso il dispositivo esterno, un'uscita NPN assorbe corrente dal dispositivo esterno ed un'uscita PUSH-PULL è in grado di fornire ed assorbire corrente.

Nel PLC le uscite sono sempre di tipo PNP anche se c'è la possibilità di avere delle uscite a relè, che forniscono cioè il contatto pulito o un contatto con un comune come nella figura sotto.



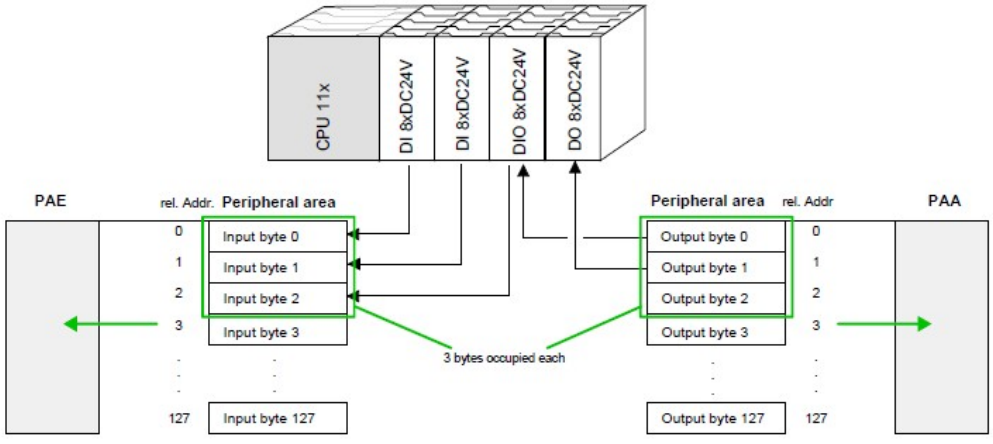
Le uscite PNP, NPN e le uscitè a relè sono sempre comunque uscite per gestire basse correnti, in genere parliamo al massimo di 1 A per le uscite a relè o di 0,5 A per le uscite PNP o NPN.



Come si può notare dalle immagini viste fino ad ora, gli ingressi e le uscite sono organizzate in blocchi da 8, in qualche caso anche in blocchi da 16, comunque multipli di 8.

Questo perché come vedremo più avanti gli ingressi e le uscite sono corrispondenti a dei bit della memoria, che è organizzata in byte o in word.

Nell'immagine vediamo le due aree di memoria interne Peripherial Area Input e Peripherial Area Output, nell'immagine sopra chiamate **PAE** per gli **ingressi** e **PAA** per le **uscite**.



In questo modo si potrà accedere ad un ingresso o un uscita, accedendo al corrrispondente bit di memoria digitando la locazione di memoria e il numero del bit.

es. al bit 0.0 dell'area di memoria degli ingressi è associato l'ingresso 1 al bit 0.7 dell'area di memoria delle uscite è associata l'uscita 8.

STRUTTURA DELLA MEMORIA INTERNA

La memoria di un PLC è suddivisa nei seguenti blocchi a seconda del loro utilizzo:

- Area di memoria degli ingressi digitali ed analogici.
- Area di memoria delle uscite digitali ed analogiche.
- Area di memoria delle variabili ritentive.
- Area di memoria delle variabili, organizzate in bit, byte, word e double word, area di lavoro.
- Area di memoria dei blocchi funzionali (funzioni).
- Area di memoria dei blocchi dati.
- Area di memoria dei blocchi organizzativi. Programmi con differenti priorità di esecuzione.
- Area di memoria dei contatori.
- Area di memoria dei temporizzatori.

In base al tipo di PLC queste aree possono essere differenti, anche se in linea di massima la struttura rimane abbastanza simile.
La suddivisione di queste aree è fissa come la loro dimensione, pertanto è fondamentale scegliere il PLC adatto per il tipo di applicazione scelta.
Quando si sceglie un PLC lo si fa in base alla CPU, ed ad ogni tipo di CPU è associata una determinata configurazione della memoria di ingresso. La CPU si occuperà di elaborare i dati contenuti nelle aree secondo una tempistica ben definita.
Nel PLC visto sopra ad esempio abbiamo la seguente conigurazione della memoria interna:

• Area di memoria degli ingressi	1024 bit	128 Byte
• Area di memoria delle uscite	1024 bit	128 Byte
• Area di memoria delle variabili (area di lavoro)	8192 bit	1024 Byte
• Area dei contatori	256 contatori a 16 bit	512 Byte
• Area dei temporizzatori	256 temporizzatori a 16 bit	512 Byte
• Area blocchi dati, e blocchi funzionali		4096 Byte

Le capacità di memoria di un PLC viene in genere sintetizzata, riportato due soli valori e cioè:

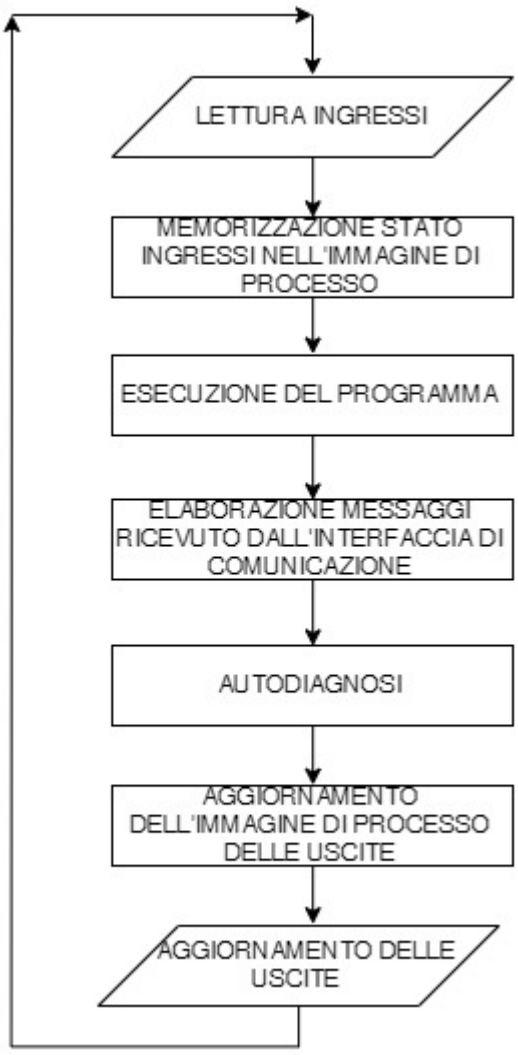
- Area di Memoria (work memory) la corrspondente RAM del PC in questo caso **16Kbyte**
- Area di Programma (load memory) il corrispondente Hard Disk nel PC in questo caso **24Kbyte**

Nel PLC è inoltre presente una memoria non volatile, contenente il firmware, e cioè quello che gestisce la scansione temporale dei blocchi di programma, e la relativa decodifica delle istruzioni scelte.
Si può notare da subito un'altra differenza rispetto ad un normale PC e cioè la quantità di memoria che può apparire irrisoria. Nella realtà la memoria indicata sopra ci consente di controllare svariati dispositivi di ingresso e di uscita, ad una velocità molto alta. Bisogna considerare ad esempio che in questo caso la tempistica di elaborazione dei dati è dell'ordine dei msec.

CICLO DI SCANSIONE

Un programma memorizzato in un PLC viene eseguito in modo ciclico, denominato “ciclo di scansione”.
Durante un ciclo di scansione, il PLC opera le seguenti azioni:

- *Il PLC legge lo stato di tutti gli ingressi*
- *Memorizza lo stato degli ingressi nella loro area di memoria (immagine di processo).*
- *Esegue il programma caricato.*
- *Elabora eventuali messagi o dati ricevuti nell’interfaccia di comunicazione.*
- *Esegue un’autodiagnosi di funzionamento.*
- *Aggiorna l’immagine di processo relativa alle uscite.*
- *Aggiorna lo stato delle uscite.*



La durata del ciclo di scansione, dipende da tanti fattori, dalla velocità della CPU, dalla quantità di dati trattati, e dalla complessità del programma, ma tipicamente in un normale PLC, la durata varia da 1 a 3msec.
Questo fatto ci impedisce di poter realizzare delle temporizzazioni inferiori alla durata del ciclo, e per questo motivo, nei comuni PLC la risoluzione dei temporizzatori interni è tipicamente di 10msec.

Questo modo di elaborare i dati e di eseguire le operazioni contenute nel programma, è differente dal classico modo sequenziale. Le operazioni scritte nel programma **è come se venissero eseguite in simultanea**, così come la lettura degli ingressi e l’aggiornamento delle uscite.

Vedremo più avanti che questo può creare qualche difficoltà le prime volte che si scrive un programma.
Anziché utilizzare un ragionamento sequenziale, dovremo considerare ad esempio che la lettura degli ingressi è simultanea, e dovremo immaginare che anche le stesse operazioni da noi scritte, avvengono nello stesso istante, sempre all’interno del ciclo di scansione.

PROGRAMMAZIONE DEL PLC

Per la programmazione, ogni marca di PLC, utilizza il proprio ambiente di sviluppo che ovviamente presenta delle differenze a livello di interfaccia, esiste comunque uno standard a livello normativo e più esattamente lo standard IEC 61131.

La parte 3 di questo standard, prevede i seguenti 5 linguaggi:

- **IL** - **I**struction **L**ist, lista di istruzioni *tipo testuale*
- **ST** - **S**tructured **T**ext, linguaggio strutturato *tipo testuale*
- **LD** - **L**adder **D**iagram, il linguaggio a contatti *tipo grafico*
- **FBD** - **F**unction **B**lock **D**iagram, diagrammi a blocchi funzionali *tipo grafico*
- **SFC** - **S**equential **F**unctional **C**hart, gli schemi funzionali sequenziali *tpo grafico*

Esistono anche altri linguaggi creati da aziende costruttrici di PLC, che non rispondono esattamente allo standard della IEC61131, come ad esempio i linguaggi **AWL**, **FUB** e **KOP**, sviluppati dalla Siemens, che sono però associabili rispettivamente ai linguaggi **IL**, **FBD** e **LD**.

Questo standard oltre che definire i linguaggi, indica i tipi di dati a disposizione, come indicato nella seguente tabella:

TIPO DI VARIABILE	DESCRIZIONE	BIT
BOOL	Variabile booleana, singolo bit valori possibili 0-1	1
SINT	Signed short integer, variabile short intera con segno, range --128 +127	8
INT	Signed integer, variabile intera con segno, range -32768 +32767	16
DINT	Signed double integer, variabile doppia intera con segno, range -2147483648 + 2147483647	32
LINT	Signed long integer, long intera con segno, range -9223372036854775808 +9223372036854775807	64
USINT	Unsigned short int, variabile short intera senza segno, range 0 +255	8
UINT	Unsigned int, variabile intera senza segno, range 0 +65535	16
UDINT	Unsigned double int, variabile doppia intera senza segno, range 0 +4294967296	32
ULINT	Unsigned long int, variabile ling intera senza segno, range 0 +18446744073709551616	64
REAL	Variabile a virgola mobile, range -3.4e+38 +3.4e+38	32
LREAL	Variabile a virgola mobile doppia precisione	64
TIME LTIME	**Variabile tempo.** TIME# oppure T# seguito dagli operatori d,h,m,s,ms (day, hour, minute, second, millisecond) *es. T#2s T#2d4h10m TIME#50ms* Nel software Step7 della Siemens, la sintassi è diversa ES. S5T#2S	32 64
DATE	**Variabile data.** DATE# oppure D# seguito da anno, mese, giorno. *es. D#2019-06-10 DATE#2019-06-10*	
TIME_OF_DAY	**TIME_OF_DAY# o TOD** es.TOD#11:50:30.40 TIME_OF_DAY# 11:50:30.40	
DATE_AND_TIME	**DATE_AND_TIME# o TOD#** es. DT#2019-06-10-16:28:50.20 DATE_AND_TIME# 2019-06-10-16:28:50.20	
STRING WSTRING	Variabile stringa Variabile Wide stringa	
BYTE	8 bit senza segno, range 0 +255	8
WORD	16 bit senza segno, range 0 +65535	16
DWORD	32 bit senza segno, range 0 +4294967296	32
LWORD	64 bit senza segno, range 0 +18446744073709551616	64

Le aree di memoria possono essere indirizzate anche in maniera diretta, ed a seconda del tipo di variabile la sintassi può essere differente.

- **Variabili booleane di ingresso**

  In questo caso si utilizza il simblo **%IX** , seguito dall'indirizzo della variabile, essendo una variabile ti tipo bit, occorre indicare l'indirizzo della memoria ed il bit.

  %IX2.4 *indica il bit* ***4*** *all'indirizzo* ***2*** *della memoria dell'immagine di processo degli ingressi*

- **Variabili booleane di uscita**

  In questo caso si utilizza il simblo **%QX** , seguito dall'indirizzo della variabile, essendo una variabile ti tipo bit, occorre indicare l'indirizzo della memoria ed il bit.

  %QX2.4 *indica il bit* ***4*** *all'indirizzo* ***2*** *della memoria dell'immagine di processo delle uscite.*

- **Variabili booelane dell'area di lavoro, area Merker**

  In questo caso si utilizza il simblo **%MX** , seguito dall'indirizzo della variabile, essendo una variabile ti tipo bit, occorre indicare l'indirizzo della memoria ed il bit.

  %MX120.2 *indica il bit* ***2*** *all'indirizzo* ***120*** *della memoria di lavoro*

Il meccanismo è analogo anche per gli altri tipi di variabili, solo che al posto della X, va indicato lo spazio occupato dalla variabile con le lettere B per 1 byte, W per 2 byte e D per 4 byte.
Avremo ad esempio i seguenti casi:

- %IW3 2 byte all'indirizzo 3 dell'immagine di processo degli input
- %OB4 1 byte all'indirizzo 4 dell'immagine di processo degli output
- %MB4 1 byte all'indirizzo 4 dell'area di lavoro
- %MW10 2 byte all'indirizzo 10 dell'area di lavoro
- %MD200 4 byte all'indirizzo 200 dell'area di lavoro

Nel software Step7 della Siemens per gli ingressi si utilizza solo il simbolo E per gli ingressi e A per le uscite senza indicare il carattere %.

Le variabili ritentive vengono trattate come le altre ma occupano determinati indirizzi, che vengono definiti in fase di progetto all'interno dell'ambiente di sviluppo utilizzato.
Anche la dichiarazione delle variabili avviene in maniera differente a seconda dell'ambiente di sviluppo e del linguaggio utilizzato.
Vedremo più avanti qualche esempio concreto di programmazione con la relativa dichiarazione delle variabili.

RAPPRESENTAZIONE DEI NUMERI INTERI

I numeri senza virgola, vengono memorizzati utilizzando il sistema binario, per comprendere in quale maniera, possiamo considerare il numero più piccolo con segno, cioè il tipo SINT, con due esempi:

se il numero è positivo: char prova=98; in binario=0110 0010

se il numero è negativo: char prova=-98; in binario=1001 1110

Nel primo caso il numero è positivo ed il codice binario è la semplice trasformazione del valore in decimale:

98=0110 0010 = $0 * 2^7 + 1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$ *=64+32+2=98*

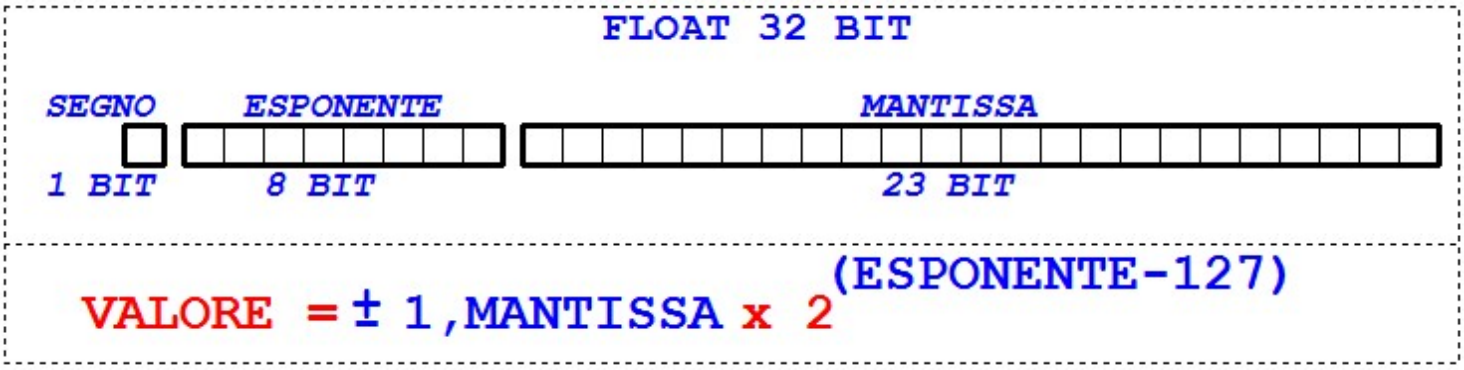
Nel secondo caso il numero negativo viene rappresentato facendo il complemento a due del valore 98 convertito in binario, cioè si ricava il complemento del numero binario invertendo ogni singolo bit e successivamente si somma il valore 1:

complemento di 0110 0010 = 1001 1101

1001 1101 + 1 = 1001 1110

RAPPRESENTAZIONE DEI NUMERI REALI

I numeri REAL utilizzano invece la rappresentazione in virgola mobile secondo lo standard IEE754 con mantissa ed argomento.



SEGNO*:* *0=positivo 1=negativo*

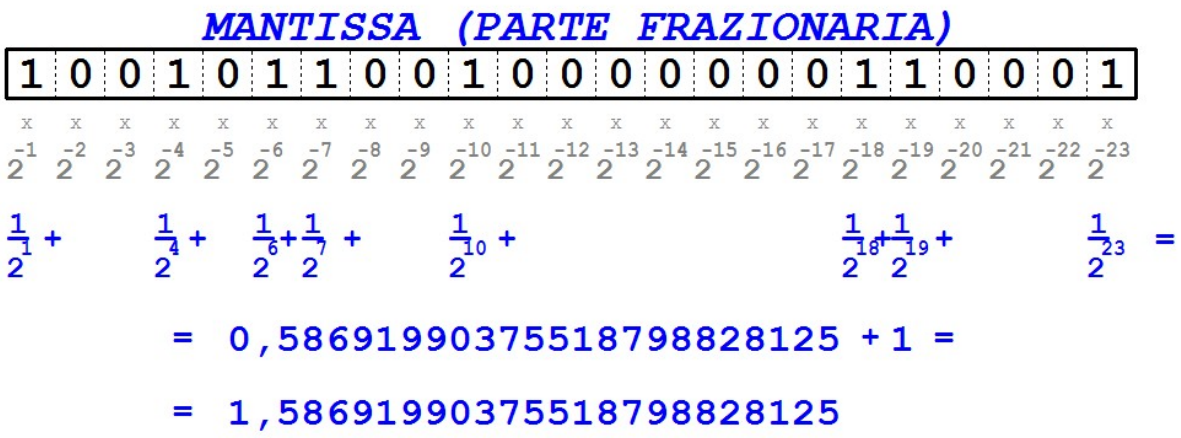
ESPONENTE: *L'esponente deve rappresentare valori positivi e negativi, per fare questo nello spazio dedicato viene messo il valore dell'esponente reale sommato al valore 127.*

In questo modo volendo memorizzare un esponente pari a +2 ,nel campo troveremo 129 che in binario corrisponde a 1000 0001.

Volento invece memorizzare un esponente pari a -2, nel campo troveremo 125 che in binario corrisponde a 0111 1101.

MANTISSA: *La mantissa è normalizzata, la normalizzaione si ottiene moltiplicando per 2 l'effettivo valore (in binario la moltiplicazione per 2 avviene shiftando a sinistra di una posizione) fino a quando il bit più a sinistra della mantissa diventa 1, eliminando in questo modo tutti i bit a zero non significativi a sinistra. Ciò significa che il bit più a sinistra vale sempre 1, per questo motivo è inutile memorizzare questa informazione, i 23 bit pertanto serviranno solo a rappresentare la parte frazionaria del numero.*

*Ad esempio una mantissa pari a **100 1011 0010 0000 0011 0001** trasformata come parte frazionaria in decimale corrisponde a:*



*Se ad esempio nei 4 byte abbiamo la seguente sequenza **0100 0100 0100 1011 0010 0000 0011 0001** , significa che il valore sarà ottenuto come segue:*

***S=0**=+*

$$VALORE = \pm 1,M \times 2^{(E-127)}$$

***E=100 0100 0**=136*

***M=100 1011 0010 0000 0011 0001** =0,5869199037551879882812 5*

Sostituendo i valori otteniamo: $VALORE = +1,586919903755187988281 25 \times 2^{(136-127)}$

Eseguendo il calcolo avremo: VALORE = 812,50299072265625

RAPPRESENTAZIONE DEI NUMERI IN BCD (Binary Coded Decimal)

In questa rappresentazione si utilizzano 4 bit per ogni cifra decimale. Pertanto i bit utilizzati dipenderanno dalla quantità di cifre necessaria per rappresentare il numero.

DECIMALE	BINARIO	BCD
0	0	0000
1	1	0001
2	10	0010
3	11	0011
4	100	0100
5	101	0101
6	110	0110
7	111	0111
8	1000	1000
9	1001	1001

Qualche esempio.

DECIMALE	BCD
253	0010 0101 0011
2619	0010 0110 0001 1001
9999	1001 1001 1001 1001

RAPPRESENTAZIONE DEI NUMERI IN CODICE GRAY e CODICE ASCII

Il codice GRAY viene utilizzato solo per rappresentare dei numeri, a differenza del classico codice binario, tra una cifra e l'altra può cambiare solo un il valore di una cifra.

Il codice ASCII invece serve per rappresentare dei simboli, a cui viene associato un numero.

BINARIO	GRAY
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
01100	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	SPC	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	€		‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ		Ž	
	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9		'	'	"	"	•	–	—	˜	™	š	›	œ		ž	Ÿ
	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ
	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Esaminiamo ora i linguaggi di programmazione sopra menzionati.

IL - Istruction List

E' un linguaggio testuale, assomiglia molto al linguaggio assembler dei microprocessori, anche se la sintassi è differente. Le istruzioni principali sono le seguenti:

- LD - Load, cioè carica una variabile o una costante.
- LDN . Load Not, cioè carica il valore negato della variabile.
- NOT - Operatore booleana di negazione.
- AND - Operazione booleana di AND logico, è possibile la negazione con il simbolo N
- OR - Operazione booleana di OR logico, è possibile la negazione con il simbolo N
- XOR - Operazione booleana di XOR logico
- ADD - Operazione di somma
- SUB - Operazione di sottrazione
- MUL - Operazione di moltiplicazione
- DIV - Operazione di divisione
- GT - Compara, se maggiore, Greater Than
- GE - Compara, se maggiore uguale, Greater Equal
- LT - Compara se minore, Less Than
- LE - Compara se minore uguale, Less Equal
- EQ - Compara se uguale, EQual
- NE - Compara se diverso, Enuqual
- JMP - Salta, Jump. Il salto può essere condizionato in base al risultato dell'operazione precedente JMPNC (risultato FALSE) JMPC (risutato TRUE)
- CAL - Richiama una subroutine. Può essere condizionato CALLNC, CALLC
- RET - Ritorno da una funzione
- S - Set di un bit
- R - Reset di un bit
- ST - Memorizza un valore, Store.

ST – Structured Text

Anche questo è un linguaggio testuale, assomiglia al linguaggio Pascal.

Le assegnazioni di valori vengono fatte con il simbolo :=. Le operazioni avvengono digitando gli operatori tra gli operandi, ed il controllo del flusso avviene mediante i noti cicli ***if then else*** e ***while do***, con la seguente sintassi:

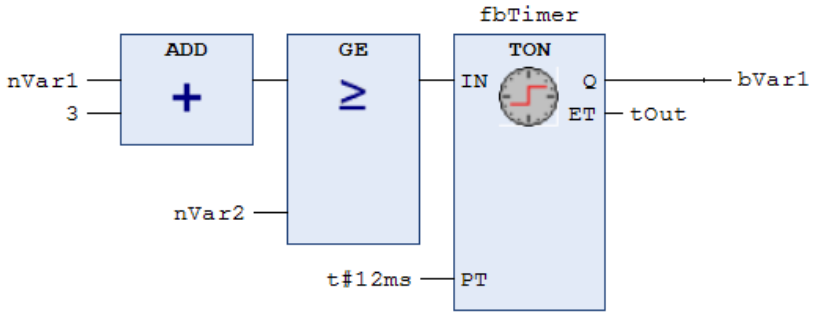
WHILE (condizione booleana**) DO**
…….
END_WHILE;

IF (condizione booleana) THEN
…….
ELSE
…….
ENDIF;

FBD – Function Block Diagram

Questo è un linguaggio grafico, e come tale utiizza dei simboli per rappresentare le operazioni tra le variabili e gli operandi. Tra i tanti è probabilmente il linguaggio meno utilizzato, probabilmente il più adatto a rappresentare algoritmi non molto complessi. Senza scendere nella simbologia che può cambiare tra le varie marche di PLC, vediamo un esempio per avere un'idea del suo funzionamento.

In questo caso la variabile nVar1 si somma al valore 3 e se il risultato è superiore al valore della variabile nVar2 viene fatto partire un temporizzatore di 12msec, con un ritardo all'inserzione, perciò l'uscita andrà a livello alto scaduto questo tempo.



SFC – Sequential Functional Chart

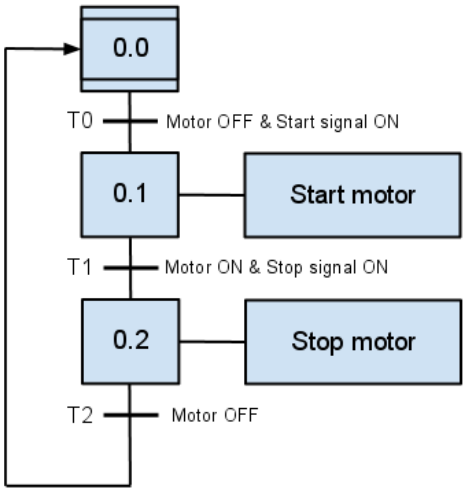
Anche questo è un linguaggio grafico, una sorta di schema a blocchi sequenziale del sistema organizzato in fasi e transizioni, anche questo non è un linguaggio molto usato, ed anche in questo caso vediamo solo un esempio.

Gli elementi fondamentali sono le fasi e cioè gli step, rappresentati con un rettangolo contenente il numero della fase, e le transizioni rappresentati con una linea orizzontale che riassume le condizioni per passare da una fase alla successiva.

Nell'esempio di fianco dopo la fase 0.0 se arriva il segnale di start e se il motore è spento, si passa alla fase 0.1 dove viene acceso il motore.

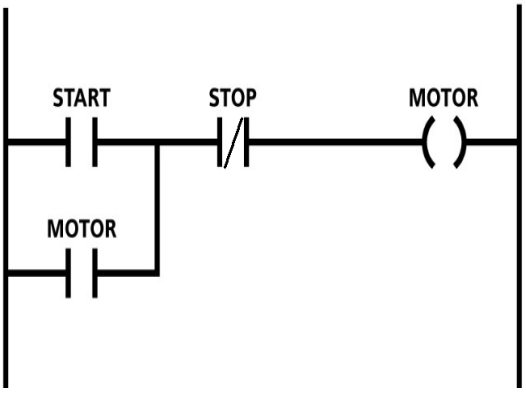
Successivamente se il motore è ON e se arriva il segnale di stop, si passa alla fase 0.2 dove il motore viene spento.

Con il motore spento si torna poi alla fase 0.0.



LD - Ladder

Dei linguaggi grafici utilizzati, il più comune e forse anche facilmente comprensibile è il linguaggio LD, e cioè il Ladder. Un linguaggio che ricorda molto la logica elettrica con contatti e bobine, per questo viene anche definito linguaggio a contatti.



Qui a fianco possiamo vedere un esempio di un circuito di autoritenuta, normalmente realizzato con un rele'.
In pratica se viene premuto il pulsante di START, e STOP non è premuto, si attiva l'uscita MOTOR, la quale è in configurazione OR con il pulsante di START, perciò una volta rilasciato questo pulsante, l'uscita MOTOR si autoalimenta.
Il pulsante STOP posto in serie e negato, spegne l'uscita quando viene premuto.

Di fianco lo stesso programma di autoritenuta, con lo schema di collegamento ed il rispettivo schema funzionale realizzato con un relè.

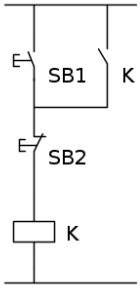


Fig.1.a. schema funzionale

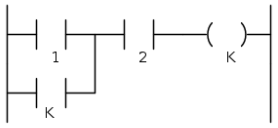
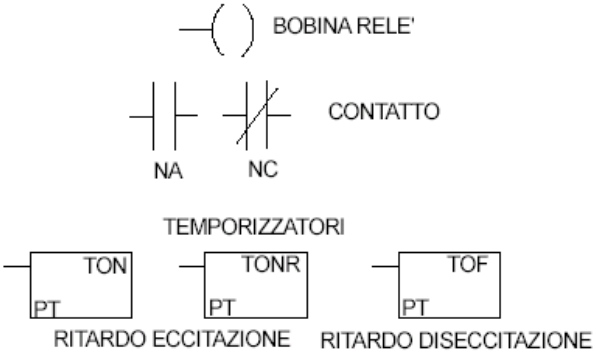


Fig.1.b. schema ladder di programmazione

Anche con questo linguaggio si possono fare operazioni di qualsiasi tipo, assegnazione di valori alle variabili, operazioni aritmetiche e logiche. Si possono gestire temporizzatori e contatori, e richiamare altri blocchi funzione.



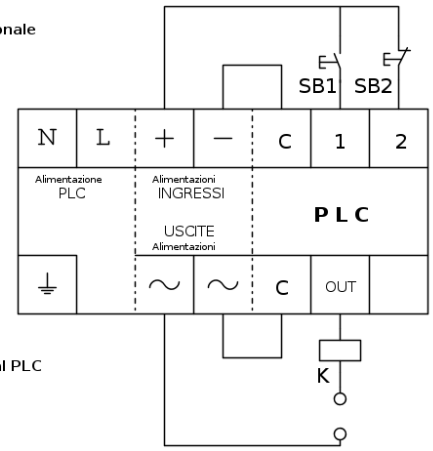


Fig.1.c. collegamenti al PLC

Nel linguaggio a contatti il programma va sviluppato su più linee, come fossero tanti circuiti dove la corrente parte da sinistra ed arriva a destra verso l'uscita.
In realtà come vedremo avanti la situazione può essere più complessa, soprattutto perché tutto quello che scriviamo **non avviene in maniera sequenziale, ma simultanea.**
Approfondiamo ora questo aspetto, e vediamo le funzioni principali presenti nei vari linguaggi per PLC.
Successivamente approfondiremo la struttura di un programma per PLC, e cioè i Programi Organization Unit, i Function Block e tutto quello che la normatica IEC 61131 prevede per standardizzare al massimo la programmazione di un PLC.

PROGRAMMI IN LINGUAGGIO LADDER

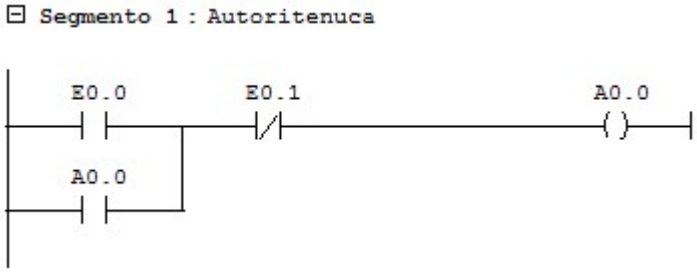
Come vedremo più avanti, un programma pu essere suddiviso in più parti, per semplicità e per comprendere la logica di funzionamento di un programma scritto in Ladder, proviamo a vedere qualche esempio scritto con il linguaggio Step7 della Siemens, struttturato inizialmente in un unico blocco contenente il codice.

Cominciamo con quello che può essere paragonato al classico “Hello World”, cioè un semplice esempio per verificare il funzionamento d un sistema.

Ricordiamo che nel linguaggio Step la sintassi per definire un ingresso è ***E INDIRIZZO, BIT***, pertanto scrivendo E0.0 indichiamo il bit 0 del’indirizzo 0 della memoria immagine di processo degli ingressi, e con E0.1 indichiamo il bit 1 collegato sempre allo stesso indirizzo.

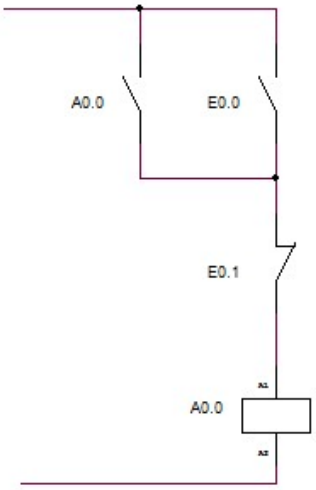
Analogamente scrivendo A0.0 indichiamo il bit 0 della memoria immagine di processo delle uscite.

Questi bit saranno associati agli ingressi ed uscite fisicamente presenti sui connettori del PLC.



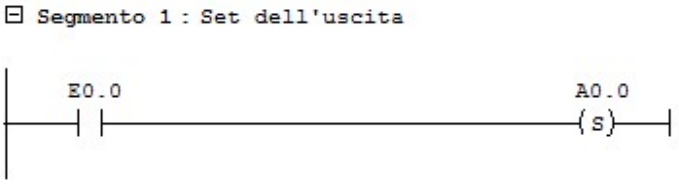
Dobbiamo interpretare il linguaggio a contatti, come dei veri e propri contatti N.O. o N.C. che alimentano una o più bobine.

Nell’esempio vediamo un contatto N.O. all’indirizzo E0.0 ed uno N.C. all’indirizzo E0.1.

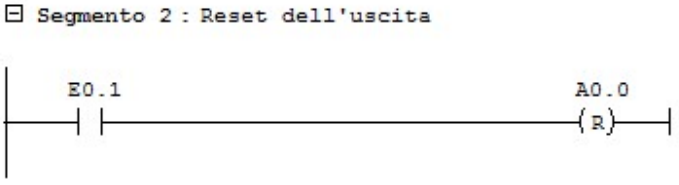


Per comprendere il funzionamento dobbiamo considerare il programma scritto come fosse il seguente circuito elettrico.

Ogni variabie di tipo booleano viene vista come un contatto normalmente aperto, normalmente chiuso o come un uscita.



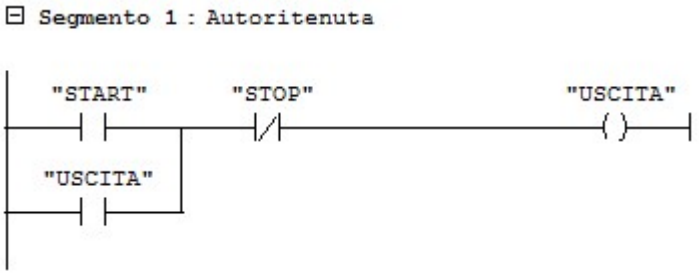
Lo stesso tipo di programma può essere scritto in linguaggio Ladder, utilizzando le funzioni Set e Reset su due righe di programma.



Entrambi i programmi eseguono la stessa funzione, anche se il programma viene scritto su più righe, la sua esecuzione non è sequenziale, riga per riga, ma avviene ciclicamente acquisendo lo stato degli ingressi per poi elaborare la logica scritta con il linguaggio Ladder.

In genere alle variabili di ingresso/uscita come anche a tutte le altre variabili, possono essere associati degli identificatori. Nel caso del software della Siemens, ciò viene effettuato con la tabella dei simboli.

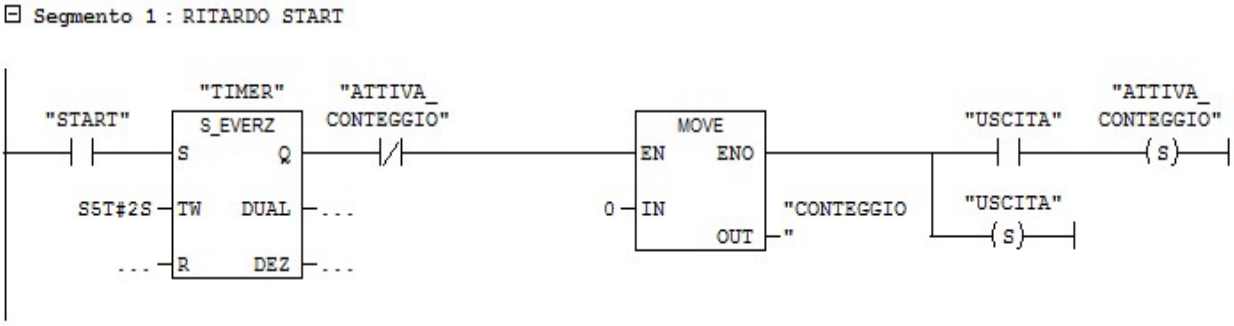
Simbolo	Indirizzo	Tipo di dati	Commento
START	E 0.0	BOOL	Pulsante di START
STOP	E 0.1	BOOL	Pulsante di STOP
USCITA	A 0.0	BOOL	Uscita

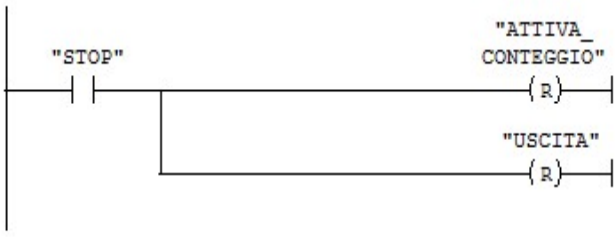


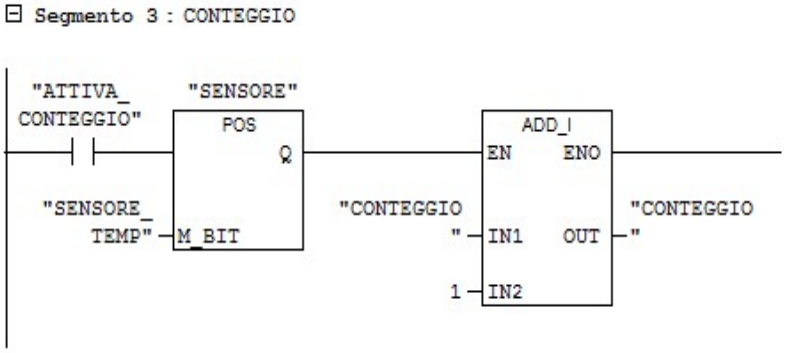
Con l'utilizzo degli identificatori il programma risulta più leggibile e come in ogni programma, nella dichiarazione delle variabili, bisogna dichiarare il tipo di dato, in questo caso booleano.

Analizziamo ora il seguente programma.

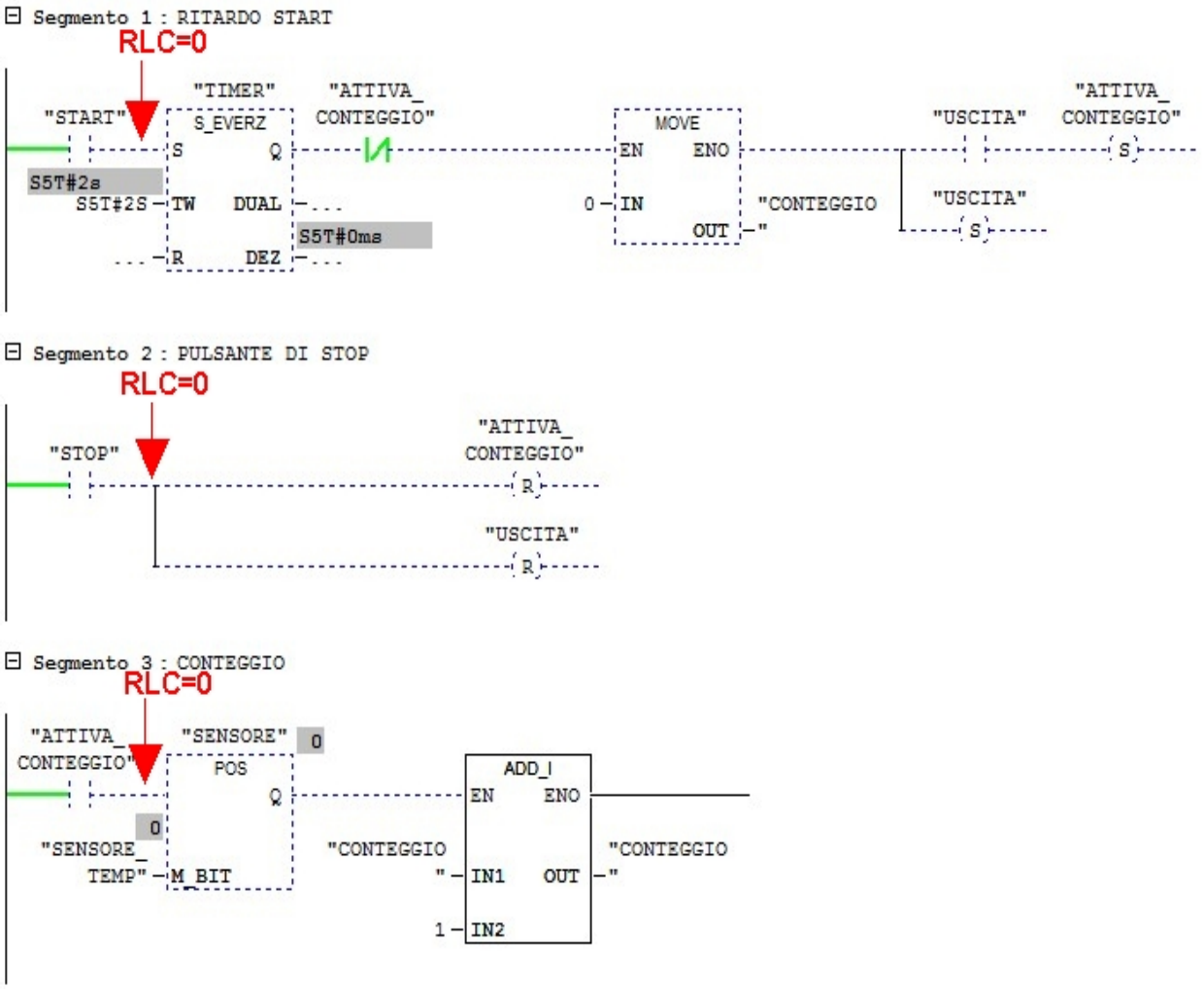
Simbolo	Indirizzo	Tipo di dati	Commento
START	E 0.0	BOOL	Pulsante di START
STOP	E 0.1	BOOL	Pulsante di STOP
SENSORE	E 0.2	BOOL	Sensore di conteggio
USCITA	A 0.0	BOOL	Uscita
ATTIVA_CONTEGGIO	M 0.0	BOOL	Bit attivazione conteggio
SENSORE_TEMP	M 0.1	BOOL	Bit per rilievo fronte
CONTEGGIO	MW 100	INT	Conteggio
TIMER	T 0	TIMER	Ritardo accensione



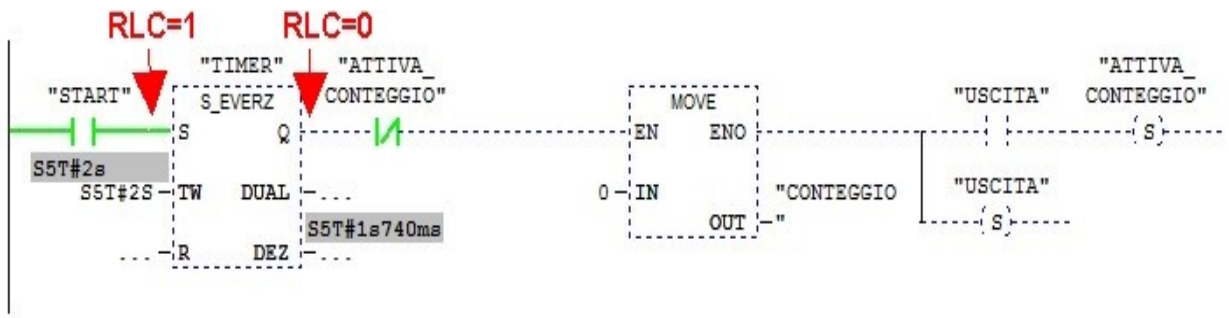




Il programma, è composto da 3 righe, che come detto prima vengono interpretate contemporaneamente.
Cerchiamo di capire cosa accade. Consideriamo all'avvio tutte le variabili al valore zero, in questo caso le 3 righe non eseguono alcuna operazione, in quanto i bit START e STOP non sono premuti ed il bit ATTIVA_CONTEGGIO vale 0. Pertanto essendo i 3 bit in configurazione NO (Normally Open), il **R**isultato **L**ogico **C**ombinatorio (RLC) della prima operazione (test dei 3 bit) è zero, perciò le restanti operazioni subordinate a questo risultato, non vengono eseguite. In pratica è come se in un analogo circuito elettrico la corrente non andasse oltre i 3 contatti.



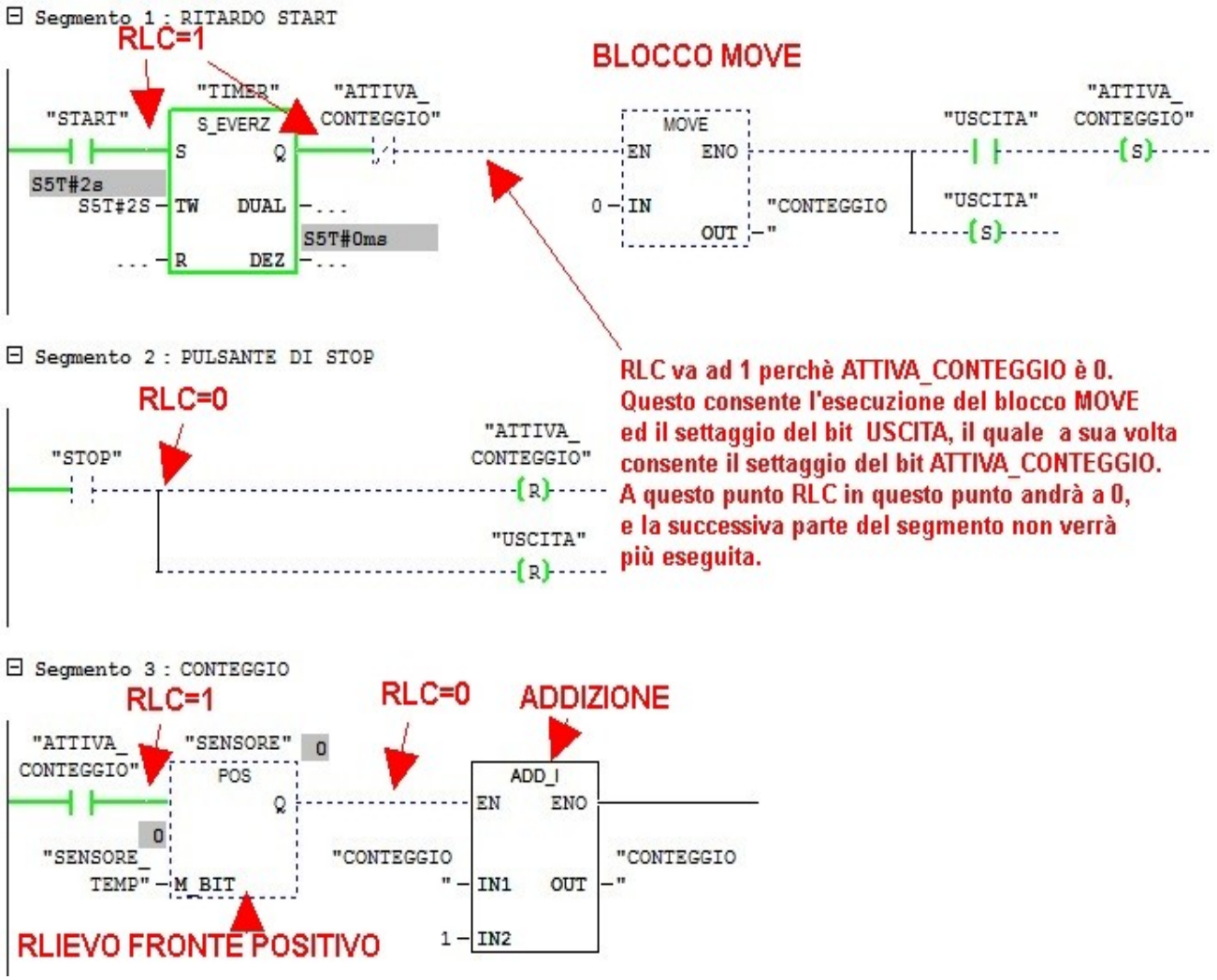
Se viene premuto il pulsante di START, il risultato RLC del primo segmento va ad uno e di conseguenza si attiva il blocco chiamato TIMER.



Il blocco timer è del tipo "ritardo all'inserzione" cioè se l'ingresso S è uno, inizia il conteggio e l'uscita andrà ad uno dopo il tempo impostato sull'ingresso TW. Sulle uscite DUAL e DEZ, c'è invece il valore attuale in formato binario o BCD. Al termine del tempo, l'uscita Q va ad uno, e viene eseguita la parte seguente del segmento.

Se si mantiene il pulsante START premuto per due secondi, verrà eseguito il blocco MOVE ed il SET delle due variabili ATTIVA_CONTEGGIO e USCITA.

Il blocco MOVE sposterà il valore in ingresso del blocco (IN) che è 0, sulla variabile di uscita e cioè CONTEGGIO e successivamente attiverà il bit USCITA e successivamente il bit ATTIVA_CONTEGGIO, a questo punto RLC dopo il bit ATTIVA_CONTEGGIO rimarrà a zero.



Il PLC esegue tutte le righe simultaneamente, pertanto appena il bit ATTIVA_CONTEGGIO va ad uno, nel terzo segmento, il Risultato Logico Combinatorio del test dello stesso bit va ad uno, e si attiva il blocco RILIEVO FRONTE POSITIVO.

Se ora ci sarà un attivazione del SENSORE, RLC in uscita del blocco andrà per un solo ciclo di scansione ad uno, e pertanto verrà eseguito una sola volta la successiva addizione.

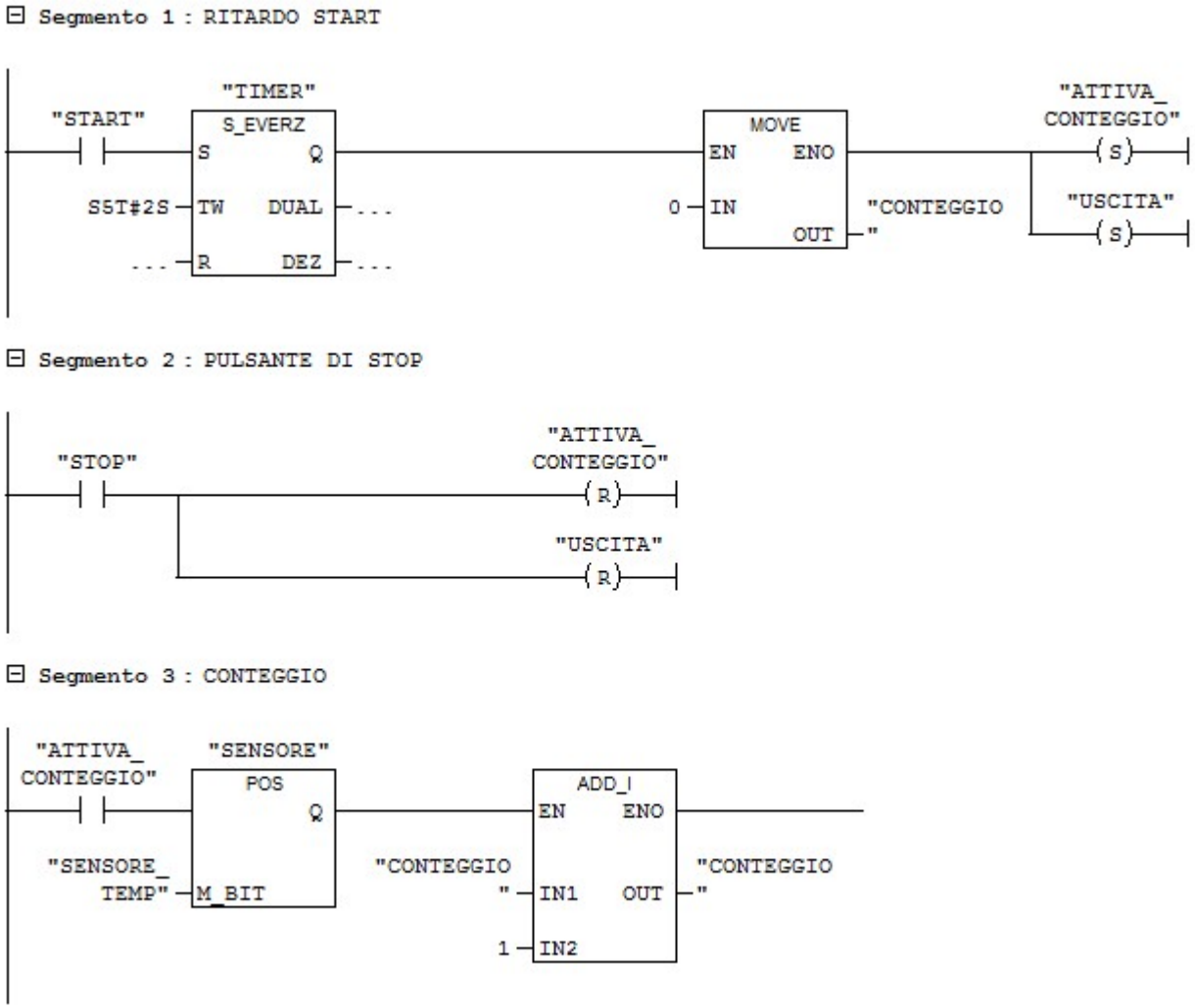
L'addizione ha in ingresso il valore della variabile CONTEGGIO, ed il valore uno, ed il risultato andrà nella stessa variabile CONTEGGIO. Effettueremo perciò un incremento della variabile ad ogni fronte di salita sul'ingresso SENSORE.

Se il bit di STOP viene attivato, verrà spento il bit ATTIVA_CONTEGGIO ed il bit USCITA, impedendo così altri conteggi all'attivazione del bit SENSORE.

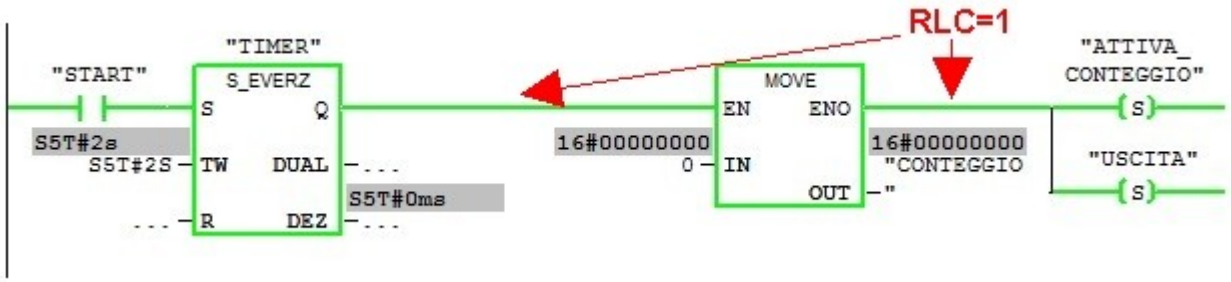
Si può facilmente intuire, che un programma scritto per un PLC segue una logica ben diversa da quella nota nei programmi scritti nei classici linguaggi per programmatori, dove l'esecuzione è spesso sequenziale.

In un programma per PLC, il Risultato Logico Combinatorio, RLC, viene verificato ad ogni ciclo di scansione, e pertanto è molto facile imbattersi in un errore.

Ad esempio un errore nel programma precedente potrebbe essere i seguente:



L'errrore sta nel fatto che dopo la pressione del pulsante di START per due secondi, il primo segmento rimane sempre attivo, perché il risultato logico combinatorio in ogni suo punto è 1.



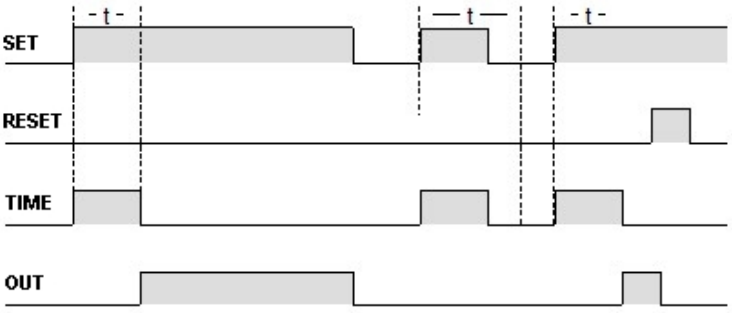
Questo comporta, che se il pulsante START rimanesse ad uno, la variabile CONTEGGIO rimarrebbe sempre a 0, in quanto il blocco MOVE viene sempre eseguito, pertanto ogni successivo conteggio sul fronte di salita del bit SENSORE, verrebbe ignorato.

Il PLC eseguendo le operazioni ad ogni ciclo di scansione, si troverebbe due operazioni da fare sempre e cioè lo spostamento del valore 0 su conteggio, e l'addizione con il valore 1. Ciò produrrà un risultato incerto sulla variabile CONTEGGIO, che sicuramente non supererà mai il valore di 1.

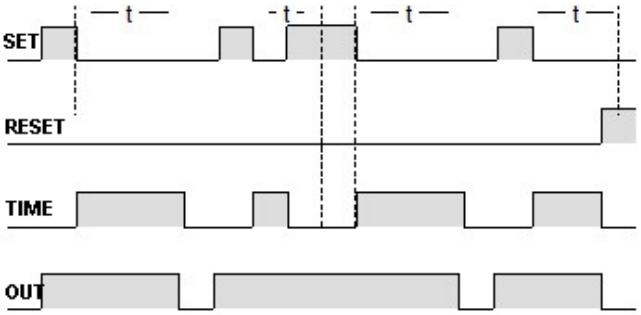
Vediamo ora le funzioni comuni previste dalla normativa IEC 61131 e comunque presenti in ogni linguaggio di programmazione per PLC.

- Set e Reset di un bit
- Rilievo del fronte di salita o di discesa di un bit
- Conversione di valore nei vari tipi disponibili.
- Operazioni di confronto tra variabili.
- Assegnazione di valore.
- Contatori in avanti,indietro ed avanti ed indietro.
- Timer con ritardo all'attivazione, alla disattivazione o di tipo impulsivo.
- Funzioni in virgola fissa come, somma, sottrazione, moltiplicazione e divisione.
- Funzioni in virgola mobile come:
  - Somma ADD
  - Sottrazione SUB
  - Moltiplicazione MUL
  - Divisione DIV
  - Valore assoluto ABS
  - Modulo MOD
  - Radice quadrata SQRT
  - Elevamento a potenza EXPT
  - Esponenziale naturale EXP
  - Logaritmo in base naturale LN
  - Logaritmo in base 10 LOG
  - Coseno COS
  - Seno SIN
  - Tangente TAN
  - Arcoseno ASIN
  - Arcococoseno ACOS
  - Arcotangente ATAN
- Funzioni booleane come:
  - Prodotto logico AND
  - Somma logica OR
  - Or esclusivo XOR
  - Shift dei bit a sinistra SHL
  - Shift dei bit a destra SHR

Queste funzioni sono comunque descritte in dettaglio in ogni ambiente di sviluppo, ad esempio nell'ambiente Step 7 di Siemens, il timer con ritardo all'attivazione viene descritto con il grafico qui a lato.



Invece questo è il grafico del ritardo alla disattivazione.

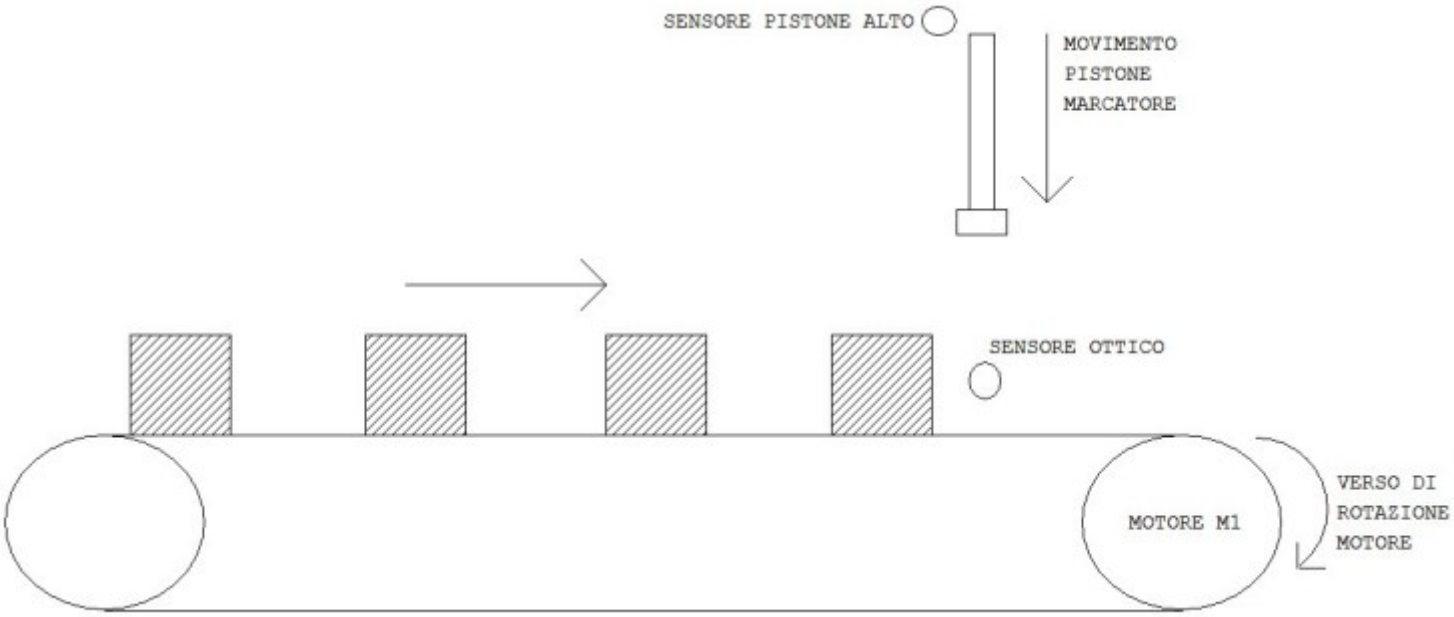


In entrambi i casi è presente il bit di Set e Reset, insieme all'uscita anch'essa vista come singolo bit, anche se nei temporizzatori è possibile conoscere il tempo residuo o quello trascorso.

Proviamo ora a risolvere un problema più complesso con un programma scritto in logica LADDER.

ESERCIZIO

Un nastro gestito da un motore, trasporta dei pezzi meccanici che vanno marcati tramite una punzonatura.



Per realizzare questo sistema occorrono due ingressi; SENSORE_PISTONE_ALTO e SENSORE_OTTICO, e due uscite; MOTORE_NASTRO e MOVIMENTO_PISTONE.

Per far partire e fermare il ciclo, servono inoltre due ingressi; PULSANTE_START e PULSANTE_STOP, e per visualizzare il ciclo in corso, servirà almeno un uscita per una spia luminosa che chiameremo CICLO_ATTIVO.

Il nostro PLC perciò dovrà almeno avere 4 ingressi e 3 uscite che possiamo mappare nel seguente modo.

MOTORE_NASTRO	A	0.0	BOOL	Motore di movimento nastro
MOV_PISTONE	A	0.1	BOOL	Movimento pistone in basso 1=giù 0=sù
CICLO_ATTIVO	A	0.2	BOOL	Spia ciclo in corso
PULS_START	E	0.0	BOOL	Pulsante di START
PULS_STOP	E	0.1	BOOL	Pulsante di STOP
SENSORE_ALTO	E	0.2	BOOL	Sensore pistone in alto
SENSORE_OTTICO	E	0.3	BOOL	Sensore di presenza pezzo
TEMPO_PISTONE	T	0	TIMER	Tempo necessario per la marcatura
TEMPO_SENSORE	T	1	TIMER	Tempo per posizionare il pezzo dopo il sensore ottico
ATTESA_SENSORE_OFF	T	2	TIMER	Attesa spegnimento sensore ottico
FRONTE_CICLO	M	0.0	BOOL	Rilievo del fronte
FRONTE_SENSORE	M	0.1	BOOL	Rilievo del fronte
ATTESA_USCITA	M	0.2	BOOL	Attesa uscita del pezzo dal sensore
AVVIO_MARCATURA	M	0.3	BOOL	Bit per avvio marcatura
AVVIO_MARCATURA_1	M	0.4	BOOL	Rilievo del fronte

Le variabili utilizzate saranno quelle sopra suddivise per tipologia, le prime 3 di tipo A, sono le uscite, poi le successive 4 di tipo E gli ingressi, di seguito 3 variabili di tipo TIMER per i temporizzatori, e 5 variabili bit di tipo BOOL dell'area di lavoro.

Nella pagina successiva vediamo l'intero programma.

Nel primo segmento c'è il classico circuito di autoritenuta, per accendere e spegnere il bit CICLO_ATTIVO tramite i pulsanti di START e di STOP.

Se il CICLO_ATTIVO è ad uno, se il SENSORE_ALTO è attivo e se il movimento del Pistone è spento, allora si accende anche il Motore del nastro.

Nel secondo segmento invece si attiva il bit ATTESA_USCITA se all'accensione del ciclo il pezzo è di fronte al sensore, e se il pistone si è mosso.

Il segmento 3 attende lo spegnimento del sensore, e dopo 1 secondo spegne il bit ATTESA_USCITA.

Il segmento 4 invece controlla se il sensore ottico di presenza pezzo è attivo, se il ciclo è attivo e se non sono in fase di attesa uscita dal sensore. Se sono vere queste condizioni, parte un timer che dopo 3 secondi accende il bit AVVIO_MARCATURA.

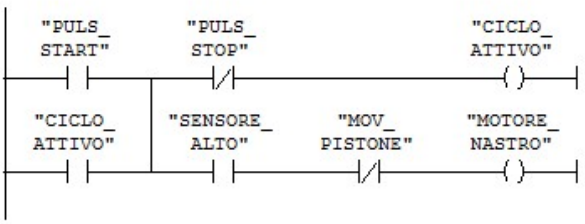
Il segmento 5 attiva il movimento del pistone, all'accensione del bit AVVIO_MARCATURA,

Il segmento 6 invece spegne l'uscita del pistone riportandolo in alto dopo 5 secondi che MOV_PISTONE si è acceso.

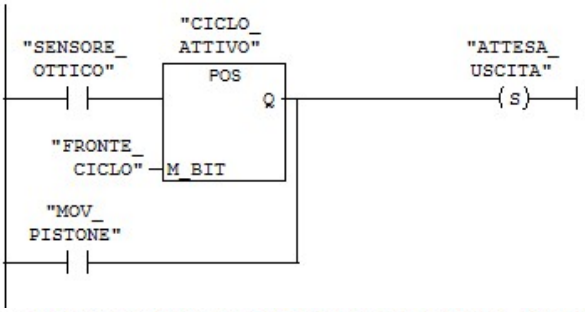
Tutti i timer utilizzati in questo programma sono come ritardo all'attivazione.

Questa non rappresenta ovviamente l'unica soluzione software possibili, ma è una delle tante che si possono realizzare con la logica LADDER.

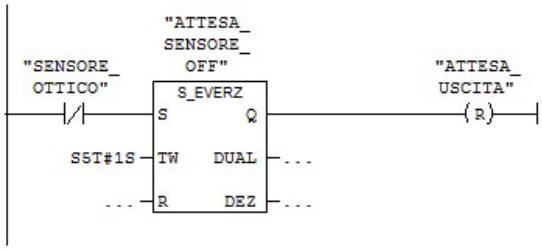
Segmento 1 : START e STOP ciclo, con autoritenuta



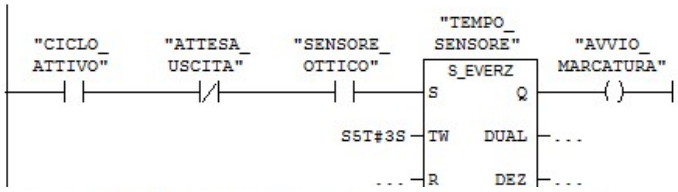
Segmento 2 : Il bit ATTESA_USCITA impedisce la doppia marcatura



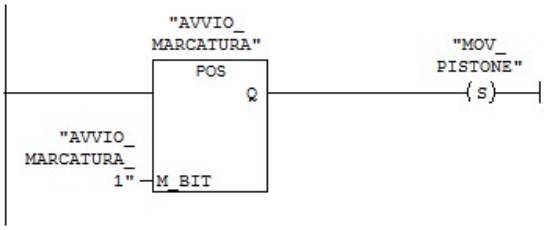
Segmento 3 : Dopo lo spegnimento del sensore, spengo il bit ATTESA_USCITA



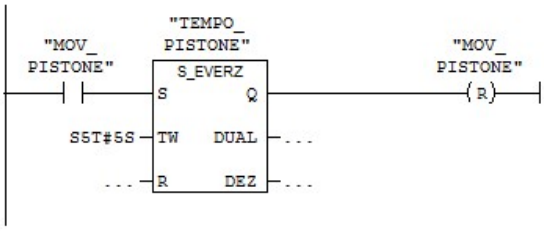
Segmento 4 : Aspetto 3 secondi dopo l'attivazione del sensore ottico



Segmento 5 : Allo scadere del tempo eseguo la marcatura



Segmento 6 : Dopo 5 secondi che il pistone va in basso spengo il movimento



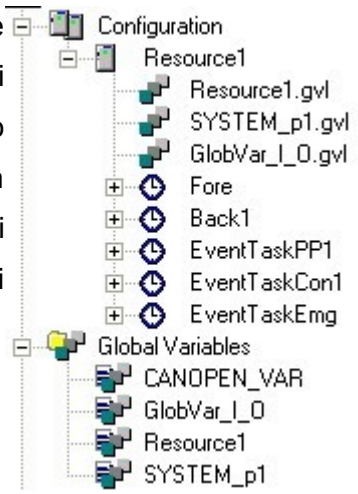
Nello studiare questo programma bisogna sempre ricordarsi che il PLC non esegue i segmenti in maniera sequenziale, ma ad ogni ciclo di scansione legge tutti i bit ed esegue tutte le istruzioni.

I programmi per PLC prevedono anche la suddivisione in sottoprogrammi o in funzioni, esistono funzioni già preimpostate e rese disponibili nell'ambiente di sviluppo ed esistono funzioni che possiamo realizzare e richiamare più volte all'interno del programma.
Esiste inoltre un blocco eseguito ciclicamente e dei blocchi di codice (Task) eseguiti secondo diverse priorità con scansione temporale o associati ad eventi.

Linguaggio strutturato ST – Structured Text

Come detto prima un altro linguaggio utilizzato, soprattutto nel settore della programmazione dei PLC all'interno dei Controlli numerici, è quello strutturato. Uno degli ambienti che si possono utilizzare e liberamente scaricabile, è il CODESYS https://www.codesys.com/ .
Questo ambiente di programmazione dei PLC è molto interessante, in quanto consente di scrivere programmi da far girare su dispositivi che hanno installato il suo runtime, cioè una sorta di interprete del linguaggio scritto.
Installando il runtime per Codesys all'interno di un qualsiasi PC o all'interno della Raspberry, rende quest'ultimo un potente PLC in grado di eseguire il software scritto in Codesys. Ovviamente un PC come la scheda Raspberry non hanno degli input ed output integrati, e dovremo aggiungere delle schede con ingressi ed uscite controllabili ad esempio da rete.
Anche molti costruttori di Controlli numerici utilizzano per la programmazione della logica PLC interna al controllo, il linguaggio strutturato, di seguito un esempio di una parte di programma scritta in linguaggio ST all'interno di un controllo numerico.

Di fianco una porzione di codice ed una parte della struttura del programma, ovviamente ogni ambiente avrà delle differenze, ma il linguaggio strutturato ha una sintassi standard, ed anche in questo caso, bisogna considerare che ogni riga di codice scritta, viene eseguita nel ciclo di scansione e non sequenzialmente.




```
VAR
TEMP_JOG   :LREAL;
STATUS     : INT;
END_VAR

(*IMPOSTAZIONE JOG INCREMENT*)
TEMP_JOG := WORD_TO_LREAL(JOG_INC);
STATUS := SJOG(PRO1,TEMP_JOG);

IF S_1MANU=TRUE THEN
    IF FLAG_USCITE=FALSE THEN
       FLAG_USCITE:=1;
       MEM_OUT_1:=OUTPUT_1;
       MEM_OUT_2:=OUTPUT_2;
       MEM_OUT_3:=OUTPUT_3;
    END_IF;
  OUTPUT_1:=MEM_OUT_1;
  OUTPUT_2:=MEM_OUT_2;
  OUTPUT_3:=MEM_OUT_3;
ELSE
  FLAG_USCITE:=0;
END_IF;
```

Nella norma IEC 61131, sentiamo anche parlare di POU
(Program Organitation Unit) che in estrema sintesi rappresentano
i blocchi di codice suddivisi per tipologia e presenti all'interno del programma.

Terminiamo così questa introduzione al PLC, rimandando ai vari ambienti di sviluppo tutti gli approfondimenti necessari, per chi fosse interessato comunque in rete si trovano diverse informazioni ed esempi applicativi relativi ai linguaggi di programmazione previsti nella norma IEC 61131.