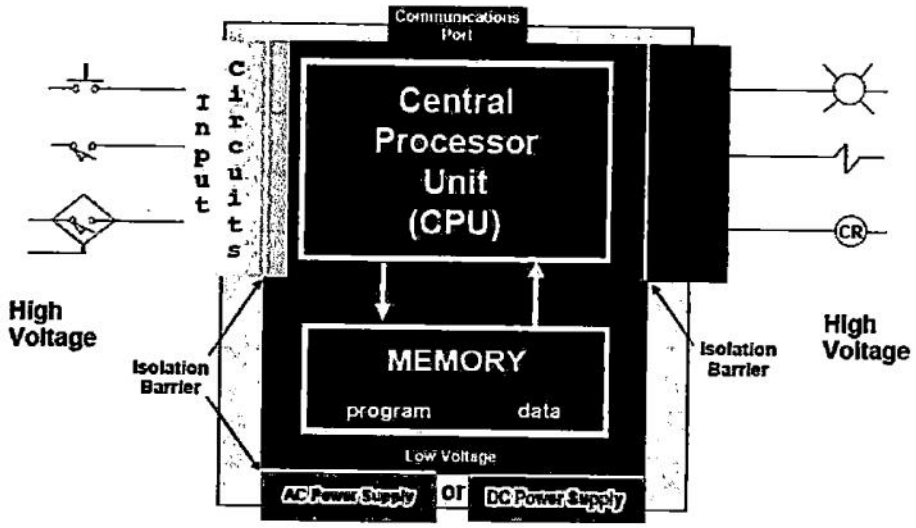
BAB II

STUDI PUSTAKA DAN LANDASAN TEORI

2.1 Definisi PLC

Programmable Logic Controller (PLC) merupakan suatu bentuk khusus alat kendali berbasis mikroprosesor yang memanfaatkan memori yang dapat diprogram untuk menyimpan instruksi-instruksi dan untuk mengimplementasikan fungsi-fungsi seperti logika, sekuensial, pewaktuan, pencacahan dan aritmetika guna mengendalikan mesin-mesin di dalam suatu sistem kendali proses (Angga, 2011).

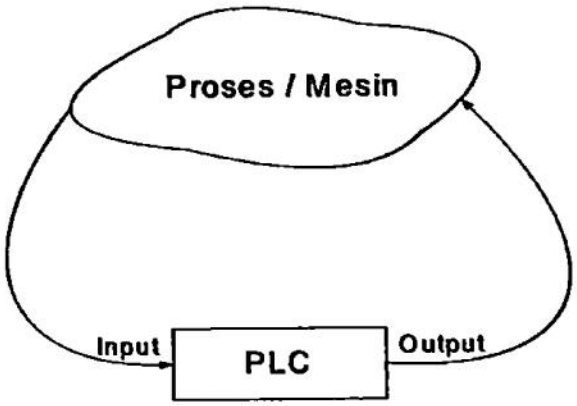


Gambar 2.1. Gambaran Umum Sebuah PLC

PLC sebenarnya merupakan suatu piranti berbasis mikroprosesor yang dikembangkan secara khusus untuk menjawab tantangan di dunia industri. Jika dibandingkan dengan *microcontroller*, PLC jauh lebih mudah digunakan dan sudah dikenal secara umum oleh pelbagai kalangan di dunia industri (Angga, 2011).

Adapun Juwana(2006) menjelaskan, PLC sesungguhnya merupakan sistem mikrokontroler khusus untuk industri, artinya seperangkat perangkat lunak dan keras yang diadaptasi untuk keperluan aplikasi dalam dunia industri.

Menurut Setiawan(2006) dan Endaryoko(2011), *Programmable Logic Controller* (PLC) pada dasarnya adalah sebuah komputer yang khusus dirancang untuk mengontrol suatu proses atau mesin. Proses yang dikontrol ini dapat berupa regulasi variabel secara kontinyu seperti pada sistem-sistem servo atau hanya melibatkan kontrol dua keadaan (*On/Off*) saja tetapi dilakukan secara berulang-ulang seperti umumnya kita jumpai pada mesin pengeboran, sistem konveyor, dan lain sebagainya. Gambar berikut memperlihatkan konsep pengontrolan yang dilakukan oleh sebuah PLC.



Gambar 2.2. Diagram Konseptual Aplikasi PLC

Setiawan(2006) menambahkan, walaupun istilah PLC secara bahasa berarti pengontrol logika yang dapat diprogram, akan tetapi pada kenyataannya PLC secara fungsional tidak lagi terbatas pada fungsi-fungsi logika saja. Sebuah PLC dewasa ini dapat melakukan perhitungan-perhitungan aritmatika yang relatif

kompleks, fungsi komunikasi, dokumentasi dan lain sebagainya (Sehingga dengan alasan ini dalam beberapa buku manual, istilah PLC sering hanya ditulis sebagai PC - *Programmable Controller*).

Adapun Caniago(2008) menjelaskan bahwa secara definitif, menurut NEMA (*National Electrical Manufactures Association*), PLC adalah suatu alat elektronika digital yang berbasis mikrokontroller dan menggunakan memori yang dapat diprogram untuk menyimpan dan mengaplikasikan instruksi – instruksi dari suatu fungsi tertentu, seperti logika, sekuensial, pewaktuan (*timing*), pencacahan (*counting*), dan aritmatika dalam rangka mengendalikan mesin-mesin ataupun suatu proses.

Juwana(2006) menjelaskan, sebuah PLC (kepanjangan: *Programmable Logic Control*) adalah sebuah alat yang digunakan untuk menggantikan rangkaian sederetan *relay* yang dijumpai pada sistem kontrol proses konvensional.

Widjiantoro, dkk.(2012) memberikan keterangan bahwa dengan demikian konsep yang digunakan pada rangkaian *relay* digunakan pula pada PLC.

Tabel 2.1. Tabel Hubungan *Relay* dengan PLC

Relay	PC
Kontak	Input
Koil	Output
Relay	Bit

Utomo(2013) dan Hutahaean(2010) memberikan pengertian bahwa PLC yang awalnya berfungsi menggantikan peran *relay*, dapat diartikan sesuai kata penyusunnya adalah sebagai berikut:

a. *Programmable* yaitu menunjukkan kemampuannya yang dapat dengan mudah

diubah-ubah sesuai program yang dibuat dan kemampuannya dalam hal memori program yang telah dibuat.

b. *Logic* yaitu menunjukkan kemampuannya dalam memproses *input* secara aritmatik (ALU) dengan melakukan proses membandingkan, menjumlahkan, mengkalikan, membagi, dan mengurangi.

c. *Controller* yaitu menunjukkan kemampuannya dalam mengontrol dan mengatur proses sehingga menghasilkan *output* yang diinginkan.

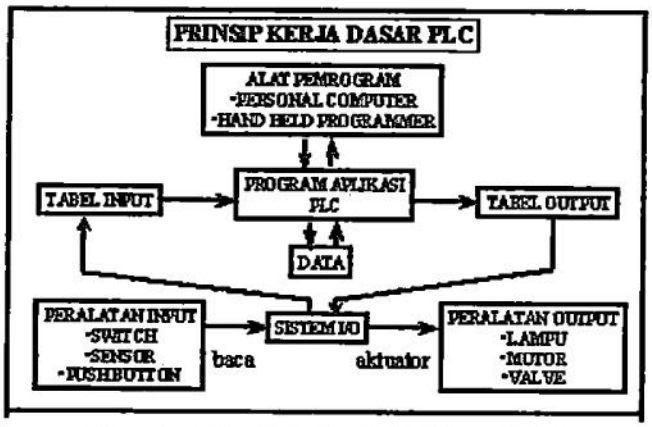
PLC merupakan elemen unit pengendali yang fungsi pengendaliannya dapat diprogram sesuai dengan keperluan. Jadi, sebelum digunakan PLC diprogram terlebih dahulu agar proses pengendalian yang terjadi sesuai dengan yang diinginkan (Caniago, 2008).

Piranti ini juga dirancang agar dapat dioperasikan oleh para insinyur yang memiliki kemampuan terbatas mengenai pemograman bahasa komputer. Oleh sebab itu para perancang PLC sudah menempatkan sebuah program awal (*pre-program*) yang memungkinkan program-program kontrol dapat dimasukkan dengan menggunakan bahasa pemrograman yang sederhana dan mudah dipahami (Caniago, 2008).

2.2 Prinsip Kerja Dasar PLC

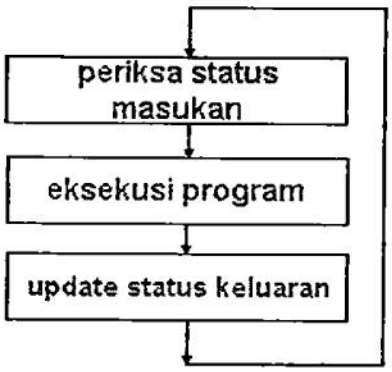
Caniago(2008) menjelaskan, PLC menerima sinyal *input* dari peralatan diskrit (*on/off*) atau analog (sensor). Modul *input* mengidentifikasi serta mengubah sinyal tersebut ke dalam bentuk tegangan yang sesuai dengan modul *input* dan mengirimkannya ke CPU (*Central Processing Unit*). Sinyal *input* tersebut diolah, kemudian dikirim ke modul *output* berdasarkan program yang

telah disimpan di CPU. Bentuk sinyal *output* diubah menjadi tegangan yang sesuai dan dipakai untuk menjalankan peralatan *output* (*actuator*).



Gambar 2.3. Prinsip Kerja Dasar PLC

Angga(2011) mengungkapkan, PLC bekerja dengan cara melakukan *program scanning*. Secara umum satu siklus *scanning* meliputi 3 tahapan utama yaitu: (i) memeriksa status masukan; (ii) melakukan eksekusi program; dan (iii) memperbaharui status keluaran. Umumnya lebih dari 3, tetapi secara garis besar ada 3 tahap tersebut, sebagaimana ditunjukkan pada gambar dibawah (Juwana, 2006).



Gambar 2.4. Proses *Scanning* Program PLC

2.2.1 Memeriksa Status Masukan (*Check Input Status*)

Pada tahapan ini, PLC akan memeriksa seluruh keadaan masukan yang

berasal dari piranti eksternal, seperti: saklar tekan, sensor, dsb. (Angga, 2011). PLC membaca data masukan (*input*) melalui perangkat yang disebut modul *input* (Caniago, 2008). PLC melihat keadaan setiap masukan yang ada untuk menentukan kondisi setiap masukan tersebut apakah pada keadaan aktif atau non-aktif (Rubiyanto, dkk., 2004). Hasil pemeriksaan status *input* ini kemudian akan disimpan di dalam memori PLC (Angga, 2011).

2.2.2 Melakukan Eksekusi Program (*Execute Program*)

PLC akan mengeksekusi program kontrol yang telah dirancang dan tersimpan pada memori PLC (Caniago, 2008). Pada prinsipnya eksekusi program ditentukan oleh status masukan. Status masukan merupakan syarat wajib untuk memastikan bahwa suatu bagian program perlu dieksekusi (Angga, 2011).

PLC akan mengerjakan atau mengeksekusi program (diagram tangga) per instruksi (Juwana, 2006). Misalkan program menginginkan jika masukan pertama aktif, maka program tersebut harus mengaktifkan keluaran pertama. Program yang ada telah mengetahui masukan-masukan mana saja yang aktif/tidak aktif dari langkah sebelumnya. Setelah itu, program akan menyimpan hasil eksekusi tersebut dengan tujuan untuk dapat digunakan pada langkah selanjutnya (Rubiyanto, dkk., 2004).

2.2.3 Memperbaharui Status Keluaran(*Update Output Status*)

Hasil dari eksekusi suatu program kemudian akan menentukan perubahan status keluaran dari elemen yang terpasang pada rangkaian keluaran PLC (Angga, 2011). PLC akan memperbaharui data-data pada modul *output* PLC (Caniago, 2008).

Pembaharuan keluaran ini bergantung pada masukan mana yang *ON*

selama langkah 1 dan hasil dari eksekusi program di langkah 2 (Juwana, 2006). Pada langkah terakhir ini, PLC memperbaharui kondisi keluaran berdasarkan masukan mana yang aktif pada langkah pertama dan hasil eksekusi program yang dimasukkan pada langkah kedua (Rubiyanto, dkk., 2004).

Setelah langkah 3, PLC akan mengulangi lagi *scanning* program-nya dari langkah 1, demikian seterusnya. Waktu *scan* didefinisikan sebagai waktu yang dibutuhkan untuk mengerjakan 3 langkah tersebut (Juwana, 2006). Masing-masing langkah bisa memiliki waktu tanggap (*response time*) yang berbeda-beda. Waktu total tanggap atau total *response time* adalah jumlah semua waktu tanggap masing-masing langkah (Juwana, 2006):

waktu tanggap masukan + waktu eksekusi program + waktu tanggap keluaran = waktu tanggap total

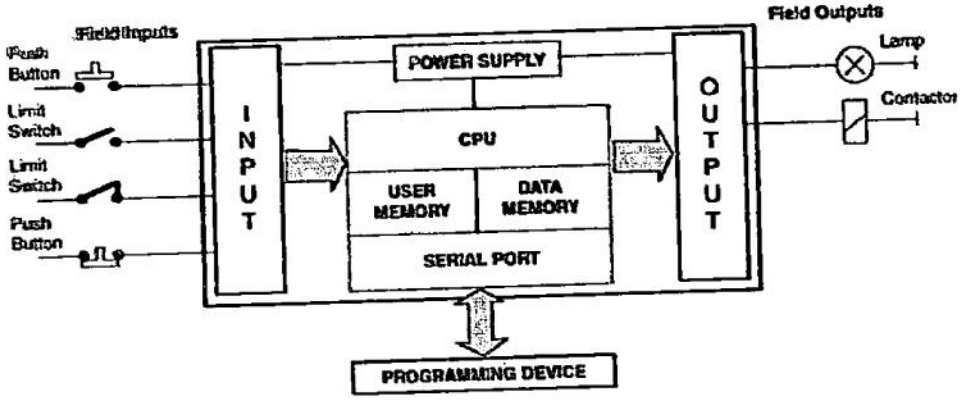
Adapun Endaryoko(2011) menjelaskan bahwa prinsip kerja sebuah PLC adalah menerima sinyal masukan proses yang dikendalikan lalu melakukan serangkaian instruksi logika terhadap sinyal masukan tersebut sesuai dengan program yang tersimpan dalam memori lalu menghasilkan sinyal keluaran untuk mengendalikan aktuator atau peralatan lainnya.

Endaryoko(2011) menambahkan, alat ini bekerja berdasarkan *input-input* yang ada dan tergantung dari keadaan pada suatu waktu tertentu yang kemudian akan meng-*ON* atau meng-*OFF*-kan *output-output*. 1 menunjukkan bahwa keadaan yang diharapkan terpenuhi, sedangkan 0 berarti keadaan yang diharapkan tidak terpenuhi (Endaryoko, 2011).

2.3 Komponen-komponen/Bagian-bagian Penyusun PLC

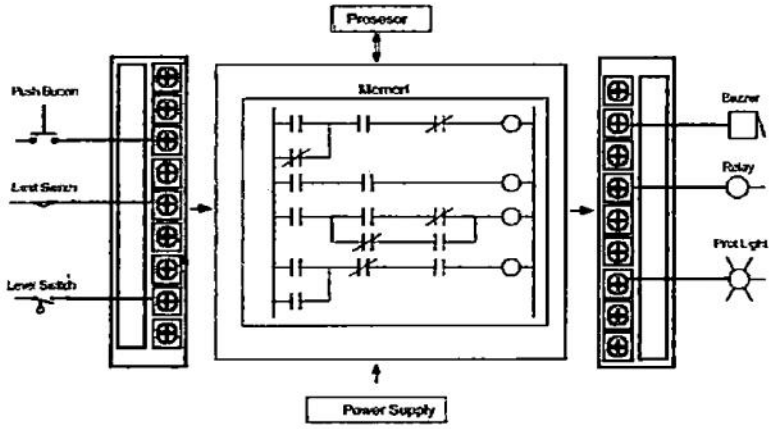
Secara umum, bagian utama suatu PLC adalah sebagai berikut: prosesor, catu daya, memori, modul *input* dan *output*, serta perangkat pemrograman(Angga, 2011). Caniago(2008) menuliskan, PLC memiliki empat komponen utama, yaitu: *Power Supply* (catu daya), *Processor*, Memori, dan Modul *Input / Output*.

Sedikit berbeda menurut Sonjaya(2011), komponen utama atau perangkat keras penyusun PLC adalah (1) Catu Daya / *Power Supply*, (2) CPU (*Central Processing Unit*) yang di dalamnya terdapat prosesor, dan memori, (3) Modul Masukan (*Input Modules*), dan Modul Keluaran (*Output Modules*), dan (4) Perangkat Pemrograman.



Gambar 2.5. Komponen-komponen Utama PLC

Adapun menurut Setiawan(2006), perangkat keras PLC pada dasarnya tersusun dari empat komponen utama berikut: prosesor, *power supply*, memori dan modul *input/output*. Secara fungsional interaksi antara ke-empat komponen penyusun PLC ini dapat diilustrasikan pada gambar berikut.



Gambar 2.6. Interaksi Komponen-komponen Sistem PLC

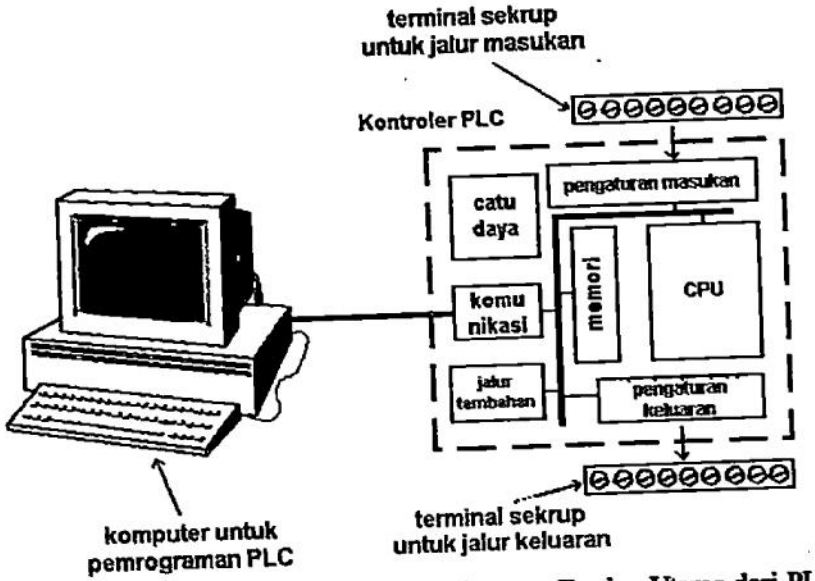
Dalam hal ini prosesor akan mengontrol peralatan luar yang terkoneksi dengan modul *output* berdasarkan kondisi perangkat *input* serta program *ladder* yang tersimpan pada memori PLC tersebut (Setiawan, 2006).

Sementara Poha, Buko(2013) menjelaskan bahwa dalam sistem PLC terdapat empat komponen bagian utama, yaitu:

1. *Central Processing Unit* (CPU), merupakan otak dari PLC yang terdiri dari 3 (tiga) bagian, yaitu:
   a) Mikroprosesor, merupakan otak dari PLC yang difungsikan untuk operasi matematika dan operasi logika.
   b) Memori, merupakan daerah CPU yang digunakan untuk melakukan proses penyimpanan dan pengiriman data pada PLC.
   c) Catu daya, yang berfungsi untuk mengubah sumber masukan tegangan bolak-balik menjadi tegangan searah.
2. *Programmer/Monitor*, adalah perangkat pemrograman yang digunakan untuk pemrograman ini umumnya tidak tersambung secara permanen ke dalam PLC. Jalannya program juga dapat diamati melalui perangkat ini.

3. *Input/Output Modules*, adalah antarmuka antara PLC dan perangkat eksternal (peralatan *input* dan peralatan *output*) dimana prosesor menerima informasi dari perangkat-perangkat eksternal tersebut dan mengkomunikasikan informasi kontrol ke perangkat-perangkat eksternal tersebut.
4. *Racks* dan *Chassis*, adalah sebagai rumah untuk PLC dan sebagai dudukan PLC agar posisinya stabil.

Elemen-elemen dasar sebuah PLC ditunjukkan pada gambar berikut(Juwana, 2006). Adapun Poha, Buko(2013) memberikan keterangan bahwa secara blok diagram, hubungan utama dari PLC dapat dilihat pada gambar berikut.



Gambar 2.7. Elemen Dasar PLC dan Hubungan Bagian Utama dari PLC

2.3.1 Perangkat Pemrograman (*Programming Device*)

Sumbodo(2008) menyatakan *Programming Device* PLC sebagai *device* masukan program yang berfungsi menjadi sarana untuk memasukkan atau mengisikan program ke dalam prosesor PLC yang disebut dengan pengisi

program (*program loader*).

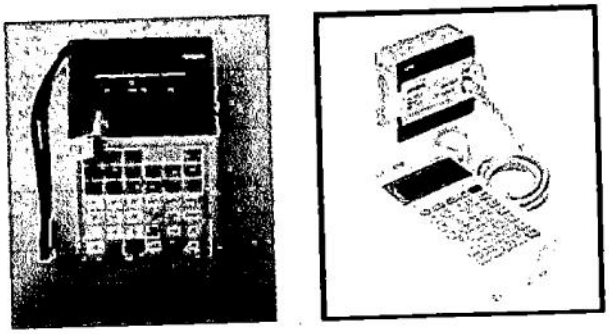
Sonjaya(2011) memberikan keterangan bahwa terdapat 2 perangkat program yang biasa digunakan, yaitu *Miniprogrammer* atau *Programming Console* dan Komputer.

2.3.1.1 *Miniprogrammer* atau *Console*

Miniprogrammer atau *Programming Console* (biasa disebut konsol) adalah sebuah perangkat seukuran kalkulator saku yang berfungsi untuk memasukkan instruksi-instruksi program ke dalam PLC (Sonjaya, 2011).

Umumnya, instruksi-instruksi program dimasukkan dengan mengetikkan simbol-simbol diagram tangga dengan menggunakan kode mnemonik (*Mnemonic Code*) (Sonjaya, 2011).

Selain digunakan untuk memasukkan program diagram *ladder*, beberapa jenis *miniprogrammer* juga dilengkapi fasilitas untuk *monitoring* dan tugas-tugas *diagnostic* (Sonjaya, 2011).



Gambar 2.8. *Miniprogrammer*

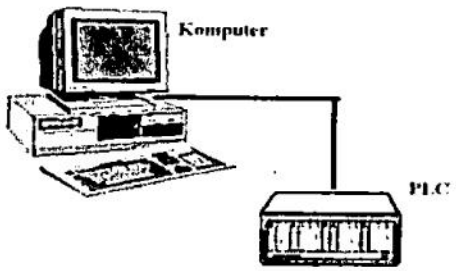
2.3.1.2 Komputer

Pemrograman PLC dengan menggunakan *miniprogrammer* ini akan sangat melelahkan jika jumlah anak tangga pada diagram *ladder* yang akan diprogram berukuran relatif besar. Umumnya, penggunaan konsol ini biasa

digunakan hanya untuk pengeditan program saja (Sonjaya, 2011).

Untuk memasukkan program secara keseluruhan pada PLC, dapat digunakan Komputer. *Vendor-vendor* PLC umumnya menyertakan perangkat lunak (*Software* atau *Support Software*) untuk mengimplementasikan pemasukan program diagram tangga, pengeditan, dokumentasi dan *monitoring* ke dalam PLC (Sonjaya, 2011).

Angga(2011) menambahkan bahwa pada komputer di dalamnya terdapat *support software* PLC yang bersesuaian. Setelah program yang dibutuhkan selesai dirancang, maka program tersebut dapat ditransfer ke PLC melalui kabel koneksi ke saluran komunikasi PLC. Jenis *port* yang sering digunakan adalah *port serial* dengan spesifikasi yang beragam (Angga, 2011).

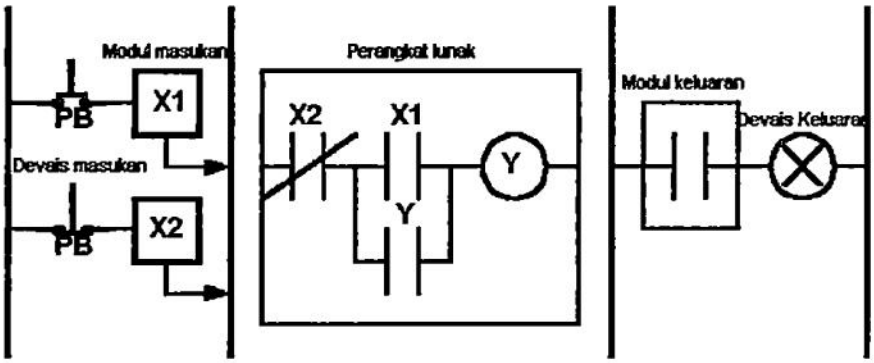


Gambar 2.9. Pemrograman PLC dengan Menggunakan Komputer

2.3.2 Hubungan *Input/Output* (I/O) dengan Perangkat Lunak

Pada saat pemrogram (*programmer*) bekerja dengan bahasa *ladder logic*, *programmer* harus mengerti hubungan I/O dengan perangkat lunak. Gambar di bawah memperlihatkan bahwa apabila *push button* 1 ditekan maka unit *input* X1 menjadi *ON*. Sesuai dengan prinsip pemahaman bahwa titik masukan sebagai *coil relay* yang mempunyai kontak di perangkat lunak, sehingga jika keadaan *ON* maka sinyal mengalir menuju modul masukan (dengan anggapan pemahaman

bahwa terdapat *coil*) hal tersebut mengakibatkan kontak dari unit *input* di dalam perangkat lunak akan bekerja (Sumbodo, 2008).

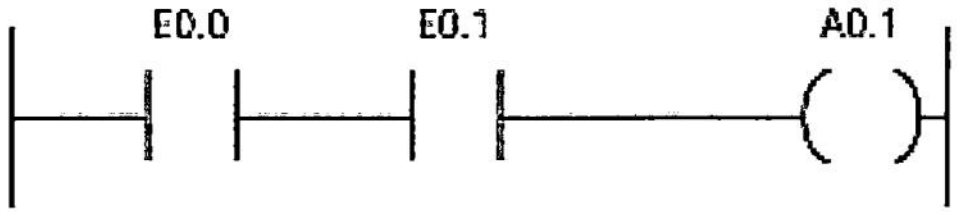


Gambar 2.10. Hubungan antara I/O dengan Perangkat Lunak

Penulis merujuk pada beberapa panduan praktikum, modul praktikum, dan karya tulis ilmiah yang terkait untuk menjelaskan tentang dasar pemrograman *ladder* PLC. Berikut penjelasan khusus tentang pemrograman *ladder* PLC.

2.4 Dasar Pemrograman *Ladder Diagram Language*/Bahasa Diagram Tangga/*Ladder Logic*

Penggambaran *ladder diagram* sesungguhnya berdasar pada diagram rangkaian. Tiga elemen penting yang biasa disajikan: kontak *Normaly Open*, kontak *Normaly Closed*, dan hasil keluaran (Endaryoko, 2011).



Gambar 2.11. *Ladder Diagram* (IEC, "IEC 61131-3, 3, 2nd Ed, 2003)

Ladder diagram terdiri dari garis vertikal yang di sebut garis bar. Instruksi yang dinyatakan dengan simbol digambarkan dan disusun sepanjang garis horizontal dimulai dari kiri dan dari atas ke bawah (Utomo, 2013).

Ladder diagram digunakan untuk menggambarkan rangkaian listrik dan dimaksudkan untuk menunjukkan urutan kejadian, bukan hubungan kabel antar komponen. Pada *ladder diagram* memungkinkan elemen-elemen elektrik dihubungkan sedemikian rupa sehingga keluaran (*output*) tidak hanya terbatas pada ketergantungan terhadap masukan (*input*) tetapi juga terhadap logika (Utomo, 2013).

Ladder language merupakan bahasa pemrograman yang menuliskan instruksi kontrol secara grafis. Untuk menggambarkan *ladder language/diagram* ada beberapa ketentuan yang perlu diperhatikan yaitu (Utomo, 2013):

a. Daya mengalir dari kiri ke kanan.

b. *Output* ditulis pada bagian yang paling kanan.

c. Tidak ada kontak yang diletakkan di sebelah kanan *output*.

d. Setiap *output* disisipkan satu kali dalam setiap program.

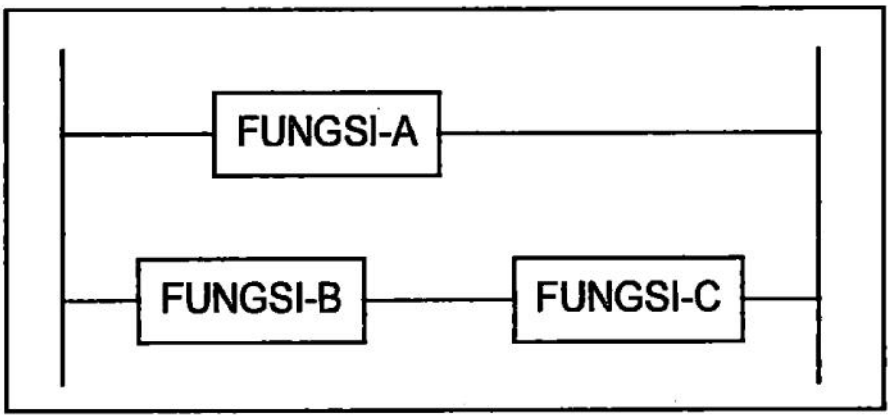
Ladder logic adalah bahasa pemrograman dengan bahasa grafik atau bahasa yang digambar secara grafik. Diagram ini menyerupai diagram dasar yang digunakan logika kendali sistem kontrol panel dimana ketentuan instruksi terdiri dari *coil-coil*, N.O., N.C. dan dalam bentuk penyimbolan. Pemrograman tersebut akan memudahkan *programmer* dalam mentransisikan logika pengendalian khususnya bagi *programmer* yang memahami logika pengendalian sistem kontrol panel. Simbol-simbol tersebut tidak dapat dipresentasikan sebagai komponen, tetapi dalam pemrogramannya simbol-simbol tersebut dipresentasikan sebagai fungsi komponen sebenarnya (Sumbodo, 2008).

2.4.1 Sistem Aliran Daya

Caniago(2008) menjelaskan bahwa sistem aliran daya merupakan prinsip

yang digunakan pada pemrograman PLC. Seperti arus yang mengalir pada rangkaian listrik, garis vertikal pada posisi kiri dan kanan adalah rel daya yang diasumsikan sebagai sumber daya untuk mengaktifkan fungsi-fungsi yang terdapat di dalam program yang dibuat.

Fungsi-fungsi tersebut secara langsung berhubungan dengan rel daya. Kemudian dieksekusi setiap satu kali *scan* operasi. Gambar berikut merupakan sistem aliran daya yang menjelaskan fungsi-A aktif jika ada aliran daya melewatinya. Sedangkan agar fungsi-C dapat aktif, maka fungsi-B harus aktif terlebih dahulu untuk melewatkan daya ke fungsi-C (Caniago, 2008).



Gambar 2.12. Sistem Aliran Daya

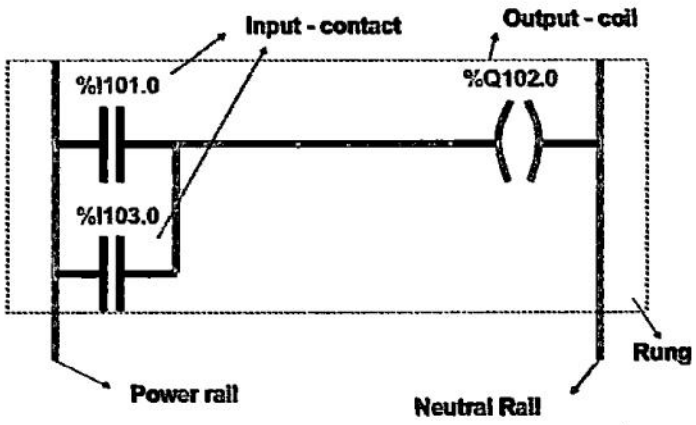
Widjiantoro, dkk.(2012) memberikan keterangan bahwa saat menentukan peralatan yang akan dipakai, yang harus dipikirkan adalah bagaimana hubungan rangkaian yang satu dengan yang lainnya. Dalam PLC rangkaian pengaturan tersebut digambarkan pada diagram tangga.

2.4.2 Konvensi tentang *Ladder Diagram*

Hal ini dijelaskan oleh Wicaksono(2008) dalam materi presentasinya berjudul "Dasar-Dasar Pemrograman PLC". Pada presentasi tersebut

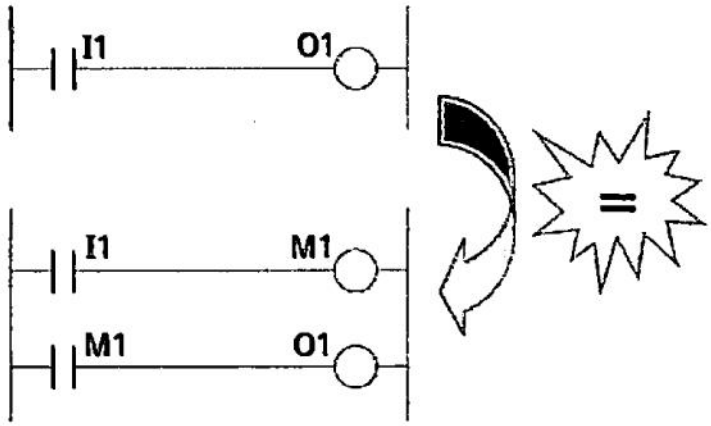
Wicaksono(2008) menuliskan beberapa hal tentang konvensi atau kesepakatan pada *ladder diagram* yang antara lain sebagai berikut.

1. *Ladder diagram* terdiri dari:
   a) *power rail* dan *neutral rail*;
   b) anak tangga (*rung*)



Gambar 2.13. *Ladder Diagram* dan Keterangannya

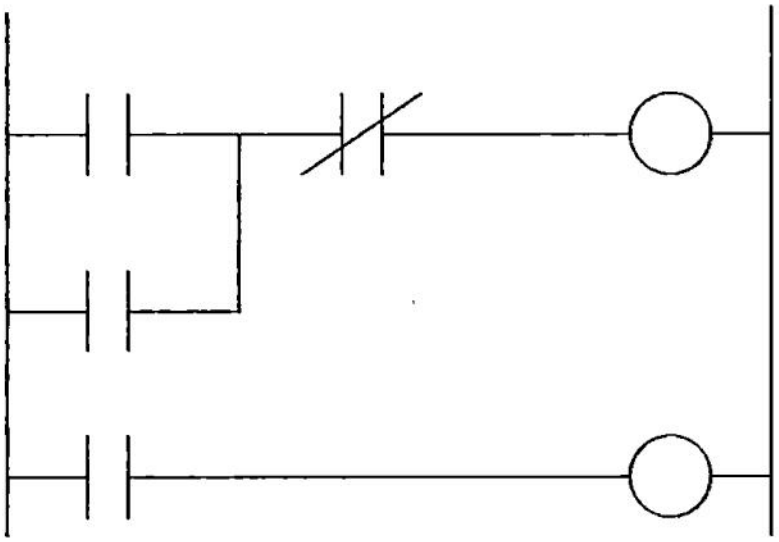
2. Dibaca dari kiri ke kanan, dari atas ke bawah.
3. *Rung* tidak boleh diakhiri dengan lebih dari satu *output*.
4. *Input/output* diidentifikasikan melalui alamatnya.
5. Penggunaan *Internal Relay* pada *Ladder*.



→ M1 = Internal relay

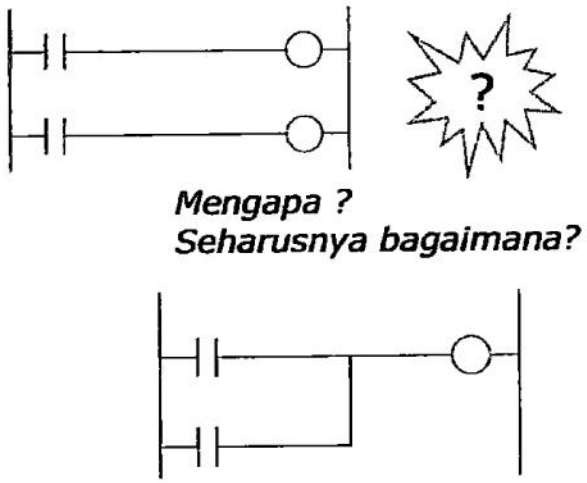
Gambar 2.14. Penggunaan *Internal Relay*

6. *Contact* dapat muncul berkali-kali.



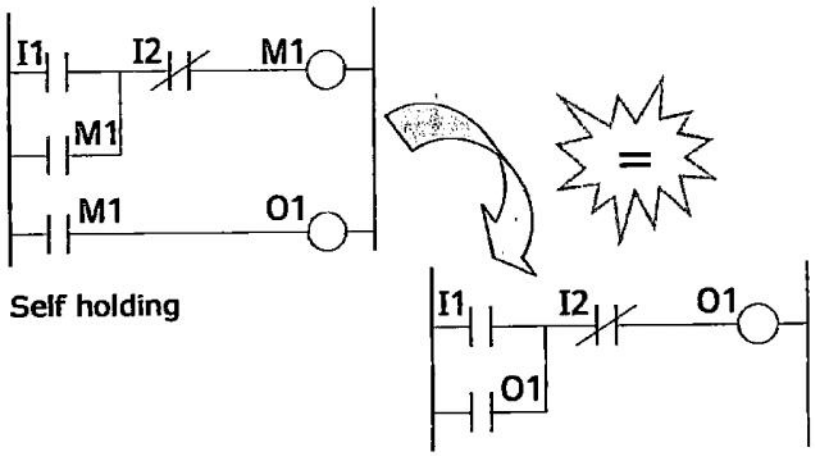
Gambar 2.15. Konvensi untuk Pemasangan *Contact*

7. *Coil* hanya dapat muncul sekali.



Gambar 2.16. Konvensi untuk Pemasangan *Coil* (Asumsi: Kedua *Coil* Beralamat Sama)

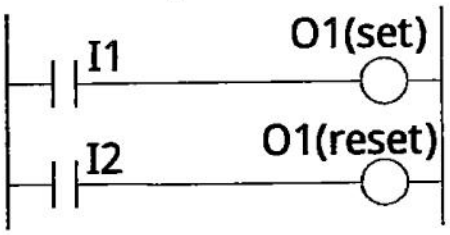
8. *Self Holding* – Sifat Khusus *Coil* di PLC (*Ladder Diagram*)



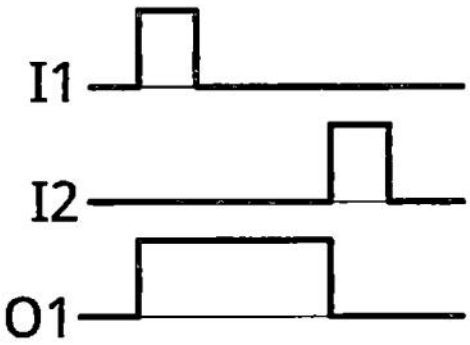
Note : NC Contact is ***the killer...***

Gambar 2.17. *Self Holding*

9. Bentuk Lain *Self Holding – Special Coil* (*Set – Reset*)



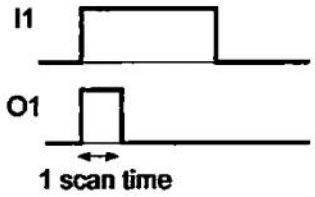
Gambar 2.18. *Ladder Diagram* dengan *Special Coil* (*SET* dan *RESET*)



Gambar 2.19. *Timing Diagram* dengan *Special Coil* (SET dan RESET)

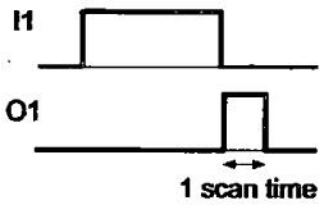
10. *Special Contact - Positive & Negative Transition Contact*

Gambar 2.20. Diagram *Ladder* dari *Positive Transition Contact*



Gambar 2.21. *Timing Diagram* dari *Positive Transition Contact*

Gambar 2.22. *Ladder Diagram* dari *Negative Transition Contact*



satu siklus *ladder* PLC

Gambar 2.23. Timing Diagram dari ***Negative Transition Contact***

2.4.3 Komponen-komponen Dasar *Ladder Diagram*

Wicaksono(2008) menuliskan dalam presentasinya bahwa komponen-komponen dasar dalam *ladder diagram* adalah sebagai berikut.

1. *Contact / input*
2. *Coil / output*
3. *Timer*
4. *Counter*

Berikut penjelasan tentang masing-masing komponen tersebut.

2.4.3.1 *Contact*

Utomo(2013) menjelaskan bahwa *contact* dapat berupa kontak *input* (saklar, *push button*), kontak *internal variable* (*relay* otomatis) dan lain-lain, ada 4 macam tipe kontak yaitu:

a. Kontak N.O. (*Normally Open*) adalah kontak yang terdapat pada *ladder diagram* di mana pada saat keadaan sistem belum bekerja kondisi kontak dalam keadaan terbuka.

b. Kontak N.C. (*Normally Close*) adalah kontak yang terdapat pada *ladder diagram* di mana pada saat keadaan sistem belum bekerja kondisi kontak dalam keadaan tertutup.

c. Kontak *rising edge* adalah kontak yang terdapat pada *ladder diagram* di mana pada saat pada saat keadaan sistem mulai bekerja kondisi kontak berubah dari logika "0" menjadi logika "1".

d. Kontak *falling edge* adalah kontak yang terdapat pada *ladder diagram* di mana pada saat keadaan sistem mulai bekerja kondisi kontak berubah dari logika "1" menjadi logika "0".

Adapun Wicaksono(2008) menuliskan dalam presentasinya, klasifikasi *contact* adalah sebagai berikut.

a) *Normal Contact*

i. *Normally Open Contact*

ii. *Normally Close Contact*

b) *Transition Contact*

i. *Positive Transition Contact*

ii. *Negative Transition Contact*

2.4.3.2 *Coil*

Utomo(2013) menjelaskan, *coil* secara umum menyatakan *output*, terdapat 4 macam tipe *coil* yaitu:

a. *Coil*

b. *Negative Coil*

c. *SET Coil*

d. *RESET Coil*

Adapun Wicaksono(2008) mengklasifikasikan *coil* menjadi dua macam sebagai berikut.

a) *Normal Coil*

b) *Latching Coil*

Dengan demikian penulis dapat mengambil kesimpulan bahwa pada intinya *input* dan *output* diagram *ladder* PLC direpresentasikan oleh dua komponen dasar, yaitu *contact* dan *coil*.

2.4.3.3 *Timer*

Wicaksono(2008) dalam presentasinya berjudul "Timer : Teori dan Aplikasi" menjelaskan beberapa hal fundamental tentang *timer* pada *ladder diagram* sebagai berikut.

1. Macam-macam *sequence* (urutan) sistem
   a) *Event driven sequence*
      - Urutan proses ditentukan oleh *event* (peristiwa).
      - *Event* merupakan kejadian yang dialami oleh *input device* (*switch*, sensor)
   b) *Time driven sequence.*
      - Urutan proses ditentukan oleh waktu.
      - Waktu ditentukan oleh *timer* (*preset value*).
   c) Gabungan *Event-Time Driven Sequence*
2. Instruksi *Timer* menggantikan "time delay relay" di masa lalu.
3. *Timer* berfungsi untuk menunda terjadinya suatu aksi.
4. Lamanya penundaan ditentukan oleh *preset value*.
5. Cara Kerja *Timer*
   a) *Timer* bekerja jika *timer coil* mendapat logika 1 dari *input*-nya.
   b) *Timer* akan menghitung sampai *preset value* dan *timer contact* akan aktif.

c) Untuk jenis *On Delay Timer* (*Default*)

Timer akan mati (kembali ke nilai awal) jika *input*-nya dimatikan.

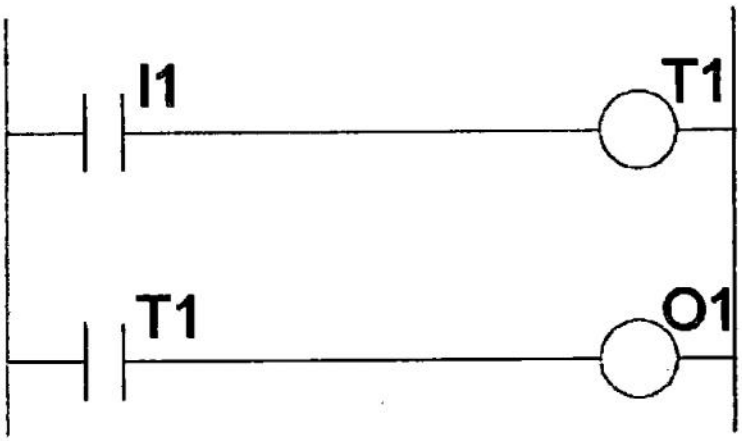
6. Beberapa Jenis *Timer*

a) *On Delay Timer*

b) *Off Delay Timer*

c) *Pulse Timer*

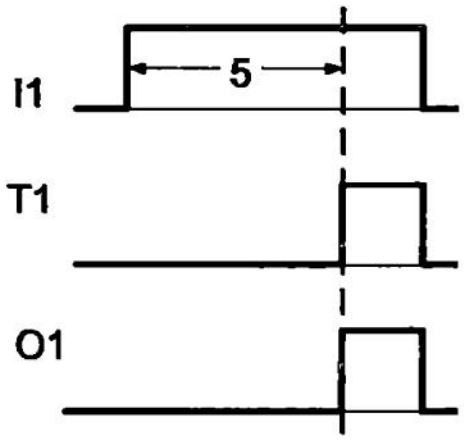
Pada gambar berikut dapat dilihat *ladder diagram* yang merepresentasikan penggunaan *timer*. I1 merupakan *input* (biasanya saklar), T1 merupakan *timer*, dan O1 merupakan *output* (biasanya paling sederhana memakai lampu LED). *Ladder diagram* ini penulis gunakan untuk menjelaskan 3 jenis *timer* sekaligus, yaitu *Timer Off Delay*, *Timer On Delay*, dan *Timer Pulse*.



Gambar 2.24. *Ladder Diagram Timer*

1. *Timer On Delay*

Secara prinsip, *timer* jenis ini dapat dijelaskan dengan gambar berikut.

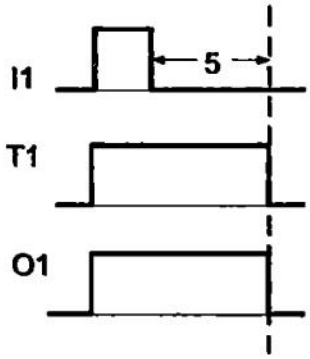


Gambar 2.25. *Timing Diagram* dari *Timer On Delay*

Pada gambar tersebut, posisi garis biru di atas menandakan kondisi aktif dan posisi di bawah menandakan kondisi nonaktif. I1 memicu *timer* untuk melakukan penghitungan waktu (asumsi: *preset* = 5 detik dan waktu mulai dari 0) selama 5 detik. Ketika penghitungan selesai, maka *output timer* akan aktif atau bernilai 1 dan menyebabkan *output* O1 aktif atau bernilai 1. Kemudian ketika *input* I1 dinonaktifkan, *timer* akan mengalami *reset*, *output timer* bernilai 0, dan *output* O1 akan nonaktif atau bernilai 0.

2. *Timer Off Delay*

Secara prinsip, *timer* jenis ini dapat dijelaskan dengan gambar berikut.

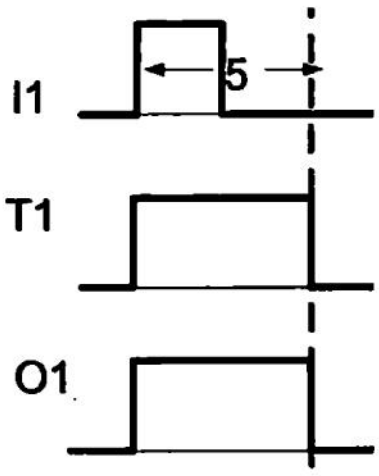


Gambar 2.26. *Timing Diagram* dari *Timer OFF Delay*

Pada gambar tersebut, posisi garis biru di atas menandakan kondisi aktif dan posisi di bawah menandakan kondisi nonaktif. I1 memicu *timer* untuk mengaktifkan *output* O1. *Output* akan selalu aktif selama *input* aktif. *Input* yang dinonaktifkan memicu *timer* melakukan penghitungan waktu mundur (asumsi: *preset* = 5 detik dan waktu mulai dari 5 ke 0) selama 5 detik. Ketika hitung mundur selesai, maka *output timer* akan nonaktif atau bernilai 0 dan menyebabkan *output* O1 tidak aktif atau bernilai 0. Setelah itu *timer* akan mengalami *reset*.

3. *Pulse Timer*

Secara prinsip, *timer* jenis ini dapat dijelaskan dengan gambar berikut.



Gambar 2.27. *Timing Diagram* dari *Timer Pulse*

Pada gambar tersebut, posisi garis biru di atas menandakan kondisi aktif dan posisi di bawah menandakan kondisi nonaktif. I1 memicu *timer* untuk mengaktifkan *output* O1. *Output* akan selalu aktif selama penghitungan waktu mundur (asumsi: *preset* = 5 detik dan waktu mulai dari 5 ke 0) selama 5 detik. *Input* yang dibaca oleh *timer* pulse berupa pulsa atau detak. Ketika hitung

mundur selesai, maka *output timer* akan nonaktif atau bernilai 0 dan menyebabkan *output* O1 tidak aktif atau bernilai 0. Setelah itu *timer* akan mengalami *reset*.

2.4.3.4 *Counter*

Wicaksono(2008) dalam presentasinya berjudul "Counter : Teori dan Aplikasi" menjelaskan beberapa hal fundamental terkait *counter* pada *ladder diagram*.

1. Fungsi *Counter*

   Menghitung banyaknya/jumlah kejadian tertentu, misal: menghitung jumlah barang untuk penyortiran, pengepakan, dll.

2. *Counter* mempunyai 2 *input*, yaitu *Pulse Input* (harus berbentuk pulsa) dan *Reset Input*.

3. Cara Kerja *Counter*

   a) *Counter coil* akan aktif dan menghitung jika *input* pulsa berubah dari 0 ke 1 (*rising edge*).

   b) *Counter coil* akan mati dan nilai kembali ke 0 jika *input reset* diaktifkan.

   c) Besar nilai yang akan dihitung *counter* ditunjukkan *preset value*.

   d) Ketika nilai *counter* mencapai *preset value*, *counter contact* akan aktif.

4. Jenis *Counter*

Tipe-tipe *counter* adalah sebagai berikut.

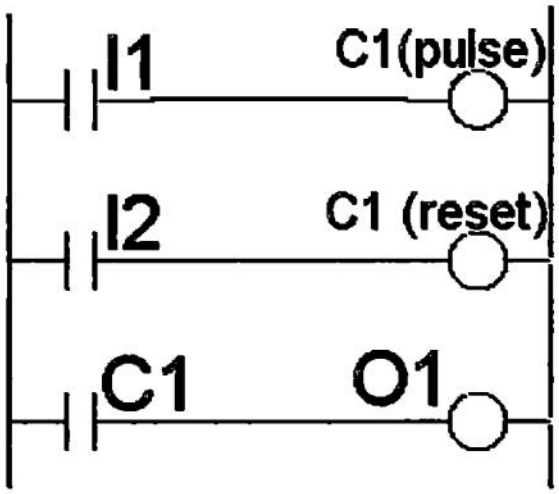
   a) *Count up* > hitungan naik

   b) *Count down* > hitungan turun

   c) *Count up - down* > hitungan naik - turun

   Pada gambar berikut dapat dilihat *ladder diagram* yang

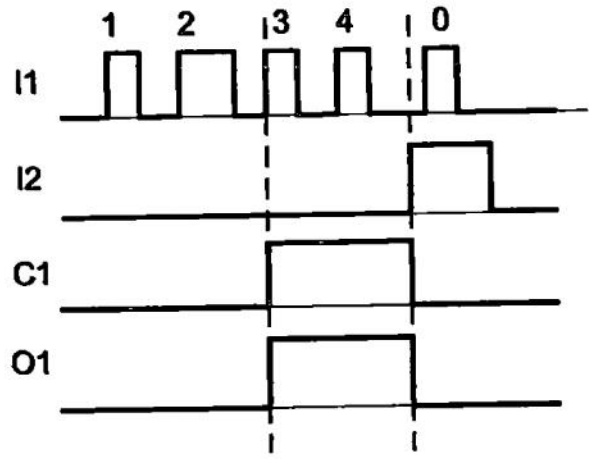
merepresentasikan penggunaan *counter*. I1 merupakan *input* (biasanya *pushbutton*), C1(*pulse*) merupakan bagian *input* pulsa dari *counter*, C1(*reset*) merupakan bagian *reset* dari *counter*, dan O1 merupakan *output* (biasanya paling sederhana memakai lampu LED). *Ladder diagram* ini penulis gunakan untuk menjelaskan 2 jenis *counter* sekaligus, yaitu *Up Count* dan *Down Count*.



Gambar 2.28. *Ladder Diagram Counter*

1. *Counter - Count Up*

Secara prinsip, *counter* jenis ini dapat dijelaskan dengan gambar berikut.

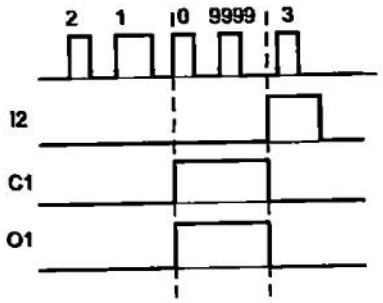


Gambar 2.29. *Timing Diagram Counter - Up Count*

Pada gambar tersebut, posisi garis biru di atas menandakan kondisi aktif dan posisi di bawah menandakan kondisi nonaktif. I1 berperan sebagai pulsa *input counter*. Saat I1 diaktifkan minimal sejumlah 3 kali, nilai aktual *counter* akan mencapai *preset*. Dengan demikian *output counter* akan bernilai 1 dan mengaktifkan O1. Adapun saat I2 diaktifkan, nilai aktual *counter* akan mengalami *reset* ke 0.

2. *Counter - Count Down*

Secara prinsip, gambar berikut dapat menjelaskan *counter - count down*.



Gambar 2.30. *Timing Diagram* dari *Counter - Count Down*

Pada gambar tersebut, posisi garis biru di atas menandakan kondisi aktif dan posisi di bawah menandakan kondisi nonaktif. I1 berperan sebagai pulsa *input counter*. Saat I1 diaktifkan minimal sejumlah 3 kali, nilai aktual *counter* akan menurun mencapai angka 0. Dengan demikian *output counter* akan bernilai 1 dan mengaktifkan O1. Adapun saat I2 diaktifkan, nilai aktual *counter* akan mengalami *reset* ke nilai *preset*-nya (dalam kasus di atas *preset* = 3).

2.5 *Software* Classic Ladder

2.5.1 Profil

Sumber: https://sites.google.com/site/classicladder/home

Sebuah proyek *open source* untuk memiliki *ladder* bebas dan pemrograman perangkat lunak sekuensial (*grafcet*) yang dikodekan dalam bahasa C (yang akan digunakan untuk pendidikan, pelatihan, di *software* PLC pada komputer PC atau *embedded platform*, ...). Umumnya, ditemukan jenis bahasa tersebut di PLC untuk membuat program-program proses otomasi. Hal ini memungkinkan untuk mewujudkan program kecil atau yang lebih besar secara elektrik dengan *ladder*. Proyek ini dirilis di bawah persyaratan lisensi LGPL. Situs *web* asli yang lama berada pada alamat berikut.

http://membres.multimania.fr/mavati/classicladder/

Namun sekarang, situs *web* tersebut sudah tidak aktif. *Update* terbaru terdapat pada situs baru Classic Ladder (https://sites.google.com/site/classicladder/home). Perangkat lunak ini, untuk *download* yang tersedia pada sourceforge.net, berada pada alamat *web* di bawah ini.

http://sourceforge.net/projects/classicladder/

Sumber: http://www.linuxcnc.org/docs/2.4/html/ladder_classic_ladder.html

Classic Ladder telah diadaptasi untuk bekerja dengan HAL EMC2/LinuxCNC, dan saat ini masih didistribusikan bersama dengan EMC2/LinuxCNC. Jika terdapat masalah(*issues/problems/bugs*) dianjurkan untuk melaporkannya kepada proyek *Enhanched Machine Controller*.

2.5.2 Konsep *Ladder*

Sumber: http://www.linuxcnc.org/docs/2.4/html/ladder_classic_ladder.html

Classic Ladder adalah jenis bahasa pemrograman awalnya diimplementasikan pada industri PLC (itu disebut *Ladder Programming*). Hal ini didasarkan pada konsep kontak *relay* dan *coil*, dan dapat digunakan untuk membangun cek(*checks*) dan fungsi logika dengan cara yang familiar bagi banyak integrator sistem. Penting untuk mengetahui bagaimana program tangga dievaluasi ketika dijalankan.

Tampaknya wajar bahwa setiap baris akan dievaluasi dari kiri ke kanan kemudian baris berikutnya turun dll., tetapi hal ini tidak bekerja seperti ini. Semua *input* dibaca, semua logika diketahui, maka semua *output* ditetapkan. Ini dapat menimbulkan sebuah masalah dalam situasi tertentu jika *output* dari satu baris menyuplai masukan dari yang lain. Masalah lainnya dengan pemrograman tangga adalah aturan "Last One Wins". Jika pengguna memiliki *output* yang sama pada lokasi yang berbeda dari *ladder*-nya, keadaan dari satu yang terakhir akan menjadi hal yang diatur pada *output*-nya. Classic Ladder versi 0.7.124 telah diadaptasi untuk EMC 2.3. Dokumen ini menjelaskan versi tersebut.

2.5.3 Bahasa

Sumber: http://www.linuxcnc.org/docs/2.4/html/ladder_classic_ladder.html

Bahasa yang paling umum digunakan saat bekerja dengan Classic Ladder adalah 'ladder'. Classic Ladder juga mendukung *Sequential Function Chart* (*Grafcet*).

2.5.4 *File*

Sumber: http://www.linuxcnc.org/docs/2.4/html/ladder_classic_ladder.html

Format *file* yang digunakan pada proyek Classic Ladder adalah ".clp", sedangkan untuk versi terbarunya (0.9.014) adalah ".clprj". *File Ladder* (.clp) tidak boleh mengandung spasi kosong dalam namanya.

Adapun berikut beberapa informasi lain tentang *software* Classic Ladder ini yang penulis dapatkan dari beberapa sumber lainnya.

Sumber: e-Gizmo_PLC64_28I/O *Programmable Logic Controller_Hardware Manual* Rev 1 r0_"Common hardware platform for use with Ladder Logic and ClassicLadder open source PLC ladder programming software."

Classic Ladder yang dikembangkan oleh Marc Le Douarain ini memiliki:

- *website* resmi:

https://sites.google.com/site/classicladder/

- alamat *web* untuk *download*:

https://sites.google.com/site/classicladder/home/downloads

Dokumentasi PDF juga tersedia dari *link* yang sama, sayangnya bagi masyarakat pada umumnya (global), kebanyakan dari dokumentasi tersebut ditulis dalam bahasa Prancis. *Google Translate* dapat membantu untuk

menyelesaikan permasalahan ini.

- pedoman singkat (bahasa Inggris):

http://users.teledisnet.be/web/rlo05343/umanual/umanual_for_classicladder.html

Versi Classic Ladder yang telah dikompilasi(*compiled*) dapat berjalan baik pada OS Windows maupun Unix.

Sumber: http://leachy.homeip.net/olinuxino/classic_ladder.html

Pada Classic Ladder pengguna dapat membuat program dengan menggunakan diagram *ladder* atau *grafcet*. Proyek ini terdiri atas 2 bagian: bagian windows dan bagian linux.

Sumber: http://www.sharewareconnection.com/classicladder.htm

Classic Ladder merupakan sebuah *software open source* yang memungkinkan pengguna untuk membuat program-program pada PLC secara elektrik. Program-program tersebut dipisahkan dalam bagian-bagian/*sections* (utama/*main* atau subrutin/*sub-routines*), di mana pengguna memilih bahasa(*ladder* atau *sequential*).

Elemen-elemen berikut tersedia pada *ladder* tersebut.

a) Elemen-elemen *Boolean*(*Boolean elements*)

b) *Rising / Falling Edges*

c) *Timers*

d) *Monostables*

e) *Counters*

f) Membandingkan ekspresi aritmatika (*Compare of arithmetic expressions*)

g) Keluaran-keluaran Boolean (*Boolean Outputs*)

h) *Coil Set / Reset*

i) Lompatan (*Jumps*)

j) Pemanggilan ke bagian subrutin (*Calls to sub-routines sections*)

k) Mengoperasikan ekspresi aritmatika (*Operate of arithmetic expressions*)

URL info program: http://sourceforge.net/projects/classicladder

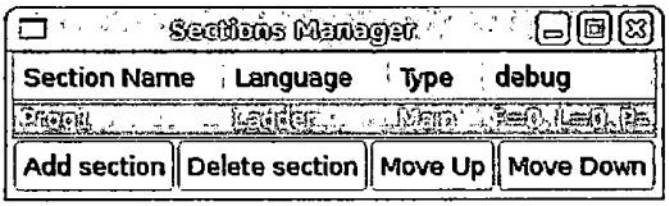
2.5.5 Tampilan / Antarmuka Grafis/ *Graphical User Interface*

Sumber: http://www.linuxcnc.org/docs/2.4/html/ladder_classic_ladder.html

Jika pengguna menjalankan Classic Ladder dengan GUI-nya, *software* ini akan menampilkan dua jendela: tampilan bagian (*section display*), dan manajer bagian (*section manager*).

2.5.5.1 Manajer Bagian(*Sections Manager*)

Ketika pengguna pertama kali memulai Classic Ladder, pengguna akan mendapatkan jendela manajer bagian(*sections manager window*) yang kosong.

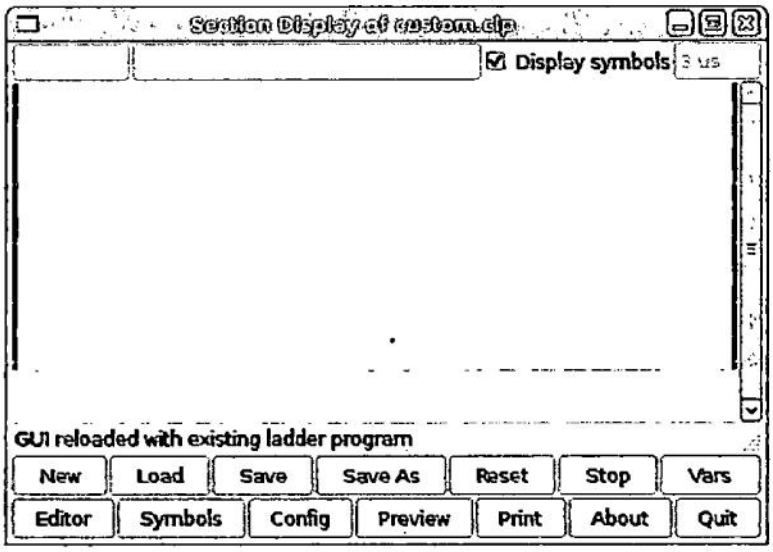


Gambar 2.31. Jendela *Default* Manajer Bagian(*Section Manager*)

Jendela ini memungkinkan kita untuk menamai, membuat atau menghapus bagian(*section*) dan memilih apa bahasa yang kita gunakan pada bagian(*section*) tersebut. Jendela ini juga terkait tentang bagaimana kita menamai *subroutine* untuk memanggil *coil-coil*.

2.5.5.2 Tampilan Bagian(*Section Display*)

Ketika kita pertama kali memulai Classic Ladder, kita mendapatkan jendela tampilan bagian (*Section Display*) yang kosong.



Gambar 2.32. Jendela *Default* Tampilan Bagian(*Section Display*)

Sebagian besar tombol yang mudah untuk dijelaskan:

1. **Tombol *Vars*** adalah untuk melihat variabel, mengalihkan untuk menampilkan satu, yang lain, keduanya, kemudian tidak satupun yang ada pada jendela tersebut.
2. **Tombol *Config*** digunakan untuk *modbus* dan menunjukkan jumlah elemen tangga maksimum yang penuh dengan modul *real time*.
3. **Tombol *Symbol*** akan menampilkan daftar simbol yang dapat di-*edit* untuk variabel-variabel tersebut (petunjuk kita dapat menamai *input*, *output*, *coil*, dll.).
4. **Tombol *Quit*** akan menutup program pengguna yang berarti Modbus dan tampilannya - program *ladder real time* masih akan berjalan di *background*.
5. **Kotak Centang(*Check Box*)** di bagian kanan atas tampilan memungkinkan kita untuk memilih antara nama variabel atau nama simbol yang ditampilkan.

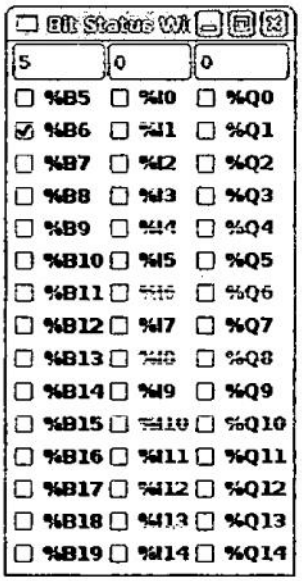
Pengguna mungkin melihat bahwa terdapat garis bawah pada tampilan

program *ladder* yang bertuliskan "Project failed to load..." Itu adalah *status bar* yang memberikan info tentang unsur-unsur program *ladder* yang pengguna klik pada jendela tampilan. *Status bar* ini akan menampilkan nama sinyal HAL untuk variabel %I, %Q dan %W pertama (dalam sebuah persamaan).

Pengguna mungkin melihat beberapa label unik, seperti (103) di dalam anak tangga(*rung*). Hal ini ditampilkan (disengaja) karena suatu kesalahan lama (*old-bug*) ketika menghapus elemen pada versi lama terkadang tidak menghapus objek dengan kode yang benar. Pengguna mungkin telah memperhatikan bahwa tombol koneksi horizontal panjang terkadang tidak bekerja dalam versi yang lebih lama/tua. Hal ini dikarenakan pencarian kode 'bebas', tetapi ditemukan sesuatu yang lain. Nomor yang berada dalam kurung adalah kode yang belum diakui atau dikenali. Program *ladder* akan tetap bekerja dengan baik, untuk memperbaikinya dibutuhkan penghapusan kode dengan *editor* dan penyimpanan program.

2.5.5.3 *Variable Window* (Jendela Variabel)

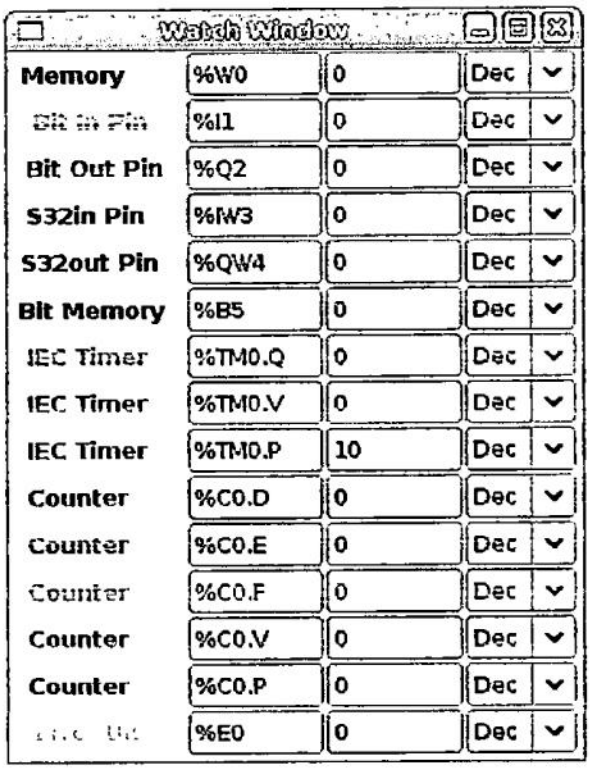
Ini adalah dua jendela variabel: *bool* dan bilangan bulat yang ditandai(*signed integer*). Dengan menggunakan tombol "Vars" di jendela tampilan bagian(*Section Display*), pengguna dapat melakukan pengalihan tombol 'Vars' untuk menampilkan satu, yang lain, keduanya, bahkan tidak satupun variabel yang ada pada jendela tersebut.



Gambar 2.33. Jendela Status Bit

Jendela *Bool* tersebut menampilkan beberapa dari semua data variabel *bool* (*on / off*). Hal yang perlu diperhatikan yaitu semua awal variabel menggunakan tanda %. Variabel %I merepresentasikan pin-pin bit masukan(*input*) HAL. %Q merepresentasikan kumparan(*coil*) *relay* dan pin-pin bit *output* HAL. %B merupakan kumparan(*coil*) *relay* internal atau hubungan(*contact*) internal. Tiga tempat/area peng-*edit*-an pada bagian paling atas memungkinkan kita untuk memilih 15 variabel yang akan ditampilkan di setiap kolom. Misalnya jika terdapat 30 variabel %B dan kita masukkan 5 di bagian atas kolom, variabel %B5 sampai %B19 akan ditampilkan.

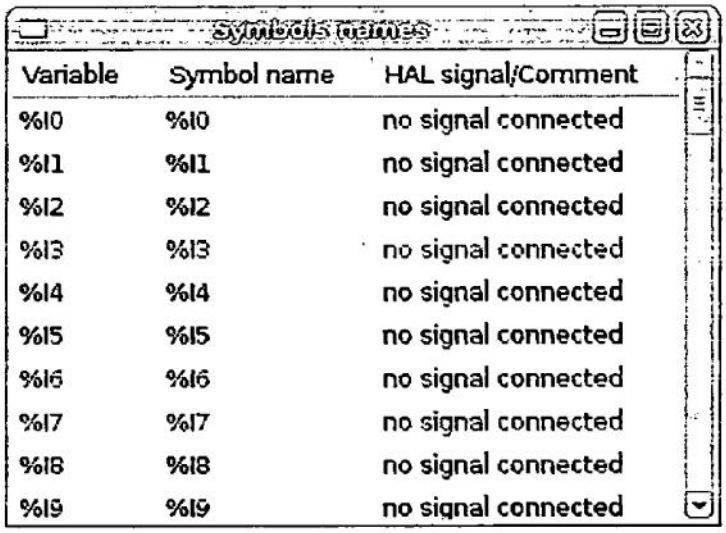
Kotak centang (*Check Box*) memungkinkan pengguna untuk mengatur dan mengatur/menetapkan variabel %B secara manual selama program tangga tidak menetapkannya sebagai *output*. Setiap bit yang ditetapkan sebagai *output* oleh program ketika Classic Ladder berjalan tidak dapat diubah dan akan ditampilkan sebagai tercentang(*checked*) jika *on* dan tidak tercentang jika *off*.



Gambar 2.34. Jendela Pemantau (*Watch Window)*

Watch Window menampilkan status variabel. *Textbox* di samping tersebut adalah nomor yang tersimpan dalam variabel dan kotak *drop-down* di samping tersebut yang memungkinkan pengguna untuk memilih nomor yang akan ditampilkan dalam hex, desimal atau biner. Jika terdapat nama simbol yang didefinisikan pada jendela simbol untuk variabel kata(*Word*) yang tampil dan terdapat centang pada 'Display Symbol' yang memeriksa jendela tampilan bagian (*section display*), nama simbol akan ditampilkan. Untuk mengubah variabel yang ditampilkan, pengguna perlu mengetikkan nomor variabelnya, misalkan %W2 (jika kotak centang 'Display Symbol' tidak dicentang) atau nama simbol (jika 'Display Symbol' dicentang) pada nomor/nama variabel dan kita menekan tombol "Enter".

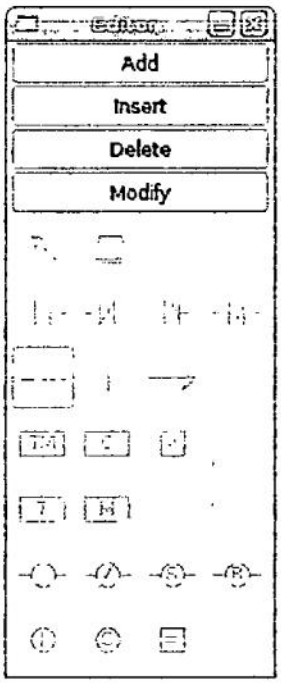
2.5.5.4 Jendela Simbol



Gambar 2.35. Jendela Nama Simbol

Ini adalah daftar nama 'symbol' yang gunakan, bukan nama variabel yang akan ditampilkan di jendela bagian (*Section Window*) ketika kotak cek 'Display Symbol' dicentang. Pengguna menambahkan nama variabel (ingat simbol '%' dan huruf kapital) dan nama simbol. Jika variabel dapat memiliki sinyal HAL terhubung (%I, %Q, dan %W - jika pengguna telah memuat pin S32 dengan modul *real time*) maka bagian komentar(*comment section*) akan menampilkan nama sinyal HAL saat itu atau kurang dari itu. Nama simbol harus tetap pendek untuk tampilan yang lebih baik. Perlu diingat bahwa pengguna dapat menampilkan lagi nama sinyal HAL %I, %Q dan variabel %W dengan mengkliknya di jendela bagian (*Section Window*). Di antara dua hal tersebut dihidupkan/diaktifkan, maka pengguna seharusnya dapat memantau dengan apa program *ladder* tersebut terhubung.

2.5.5.5 *Editor Window*



Gambar 2.36. Jendela *Editor*

Berikut keterangan tentang tombol-tombol paling atas.

- Tombol "Add" berfungsi untuk menambahkan sebuah *rung* setelah *rung* yang dipilih oleh pengguna.
- Tombol "Insert" berfungsi untuk menyisipkan sebuah *rung* sebelum *rung* yang dipilih oleh pengguna.
- Tombol "Delete" berfungsi untuk menghapus *rung* yang dipilih.
- Tombol "Modify" berfungsi untuk membuka rung yang dipilih untuk pengubahan/*editing*.

Mulai dari gambar kiri atas:

1. *Object Selector*, *Eraser*
2. N.O. *Input*, N.C. *Input*, *Rising Edge Input*, *Falling Edge Input*
3. *Horizontal Connection*, *Vertical Connection*, *Long Horizontal Connection*

4. IEC *Timer Block*, *Counter Block*, *Compare Variable*
5. *Old Timer Block*, *Old Monostable Block* (ini telah digantikan oleh IEC *Timer*)
6. *Coil* - N.O. *Output*, N.C. *Output*, *Set Output*, *Reset Output*
7. *Jump Coil*, *Call Coil*, *Variable Assignment*

Berikut deskripsi singkat dari masing-masing tombol.

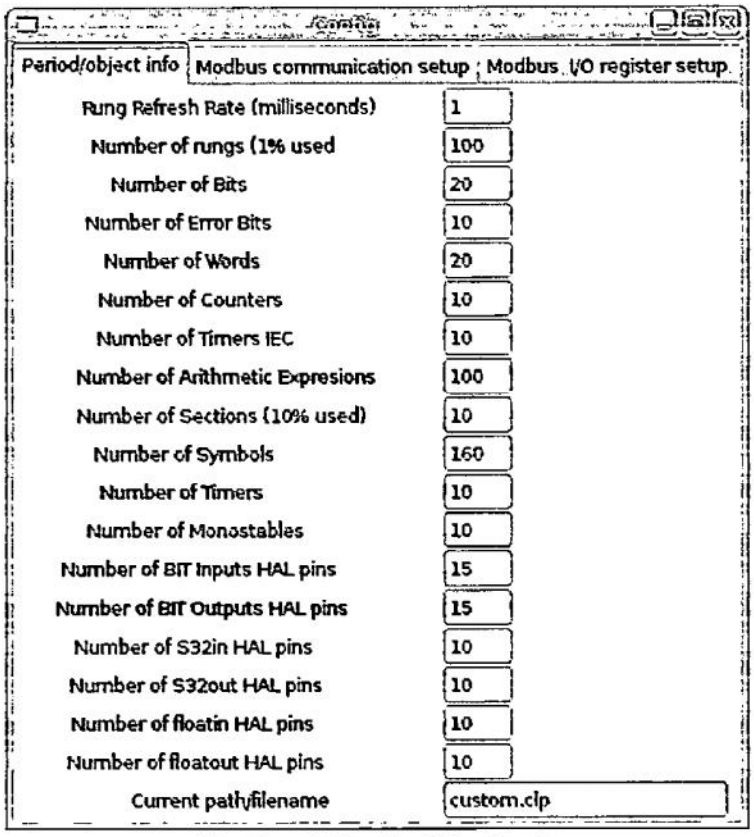
1. Tombol **ANAK PANAH PEMILIH (*SELECTOR ARROW*)** memungkinkan pengguna untuk memilih obyek yang tersedia dan memodifikasi informasinya.
2. **PENGHAPUS(*ERASER*)** dapat menghapus sebuah obyek. Namun berdasarkan pengamatan penulis, penghapusan pada obyek *output/coil/*kumparan harus dilakukan dengan tombol *coil* tersebut sendiri. Jika penghapusan *coil* menggunakan Eraser, maka aplikasi Classic Ladder akan tertutup seketika. Mungkin hal ini merupakan satu *bug/error/*masalah dari Classic Ladder.
3. **KONTAK N.O.** tersebut adalah sebuah kontak normal terbuka(*normally open contact*). Hal ini dapat berupa pin-HAL (%I) kontak *input* eksternal, kontak *internal-bit coil* (%B) atau *coil* eksternal (%Q) kontak. Kontak *input* pin-HAL tersebut ditutup ketika pin-HAL bernilai benar(*True*). Kontak-kontak *coil* tersebut menutup ketika *coil* yang sesuai aktif (kontak %Q2 menutup ketika *coil* %Q2 aktif).
4. **KONTAK N.C.** tersebut adalah sebuah kontak normal tertutup. Kontak ini sama dengan kontak N.O. kecuali kontak ini terbuka ketika pin-HAL bernilai *True* atau *coil* aktif.
5. **KONTAK *RISING-EDGE*** tersebut adalah kontak yang tertutup ketika

pin-HAL beralih dari *False* ke *True*, atau *coil* dari tidak aktif menjadi aktif.

6. **KONTAK *FALLING-EDGE*** tersebut adalah kontak yang tertutup ketika pin-HAL beralih dari *True* ke *False* atau *coil* aktif menjadi tidak aktif.
7. **KONEKSI HORIZONTAL** tersebut menghubungkan 'sinyal' ke obyek secara horizontal.
8. **KONEKSI VERTIKAL** tersebut menghubungkan 'sinyal' ke obyek secara vertikal.
9. ***HORIZONTAL-RUNNING CONNECTION*** tersebut adalah cara cepat untuk menghubungkan dengan bersambung(*long-run*) 'kawat sinyal(*signal wire*)' secara horizontal.
10. ***TIMER* IEC** tersebut menggantikan *TIMER* dan *MONOSTABLE*.
11. ***TIMER*** adalah modul *timer*.
12. ***MONOSTABLE*** adalah modul *monostable* (*one-shot*).
13. ***COUNTER*** adalah modul *counter*.
14. ***COMPARE*** memungkinkan pengguna untuk membandingkan variabel dengan nilai-nilai atau variabel-variabel lainnya. (misalnya %W1 <= 5 atau %W1=%W2) Perbandingan tidak dapat ditempatkan di sisi paling kanan dari tampilan bagian(*Section Display*).
15. ***VARIABLE ASSIGNMENT*** memungkinkan pengguna untuk memberikan nilai-nilai pada variabel-variabel.(misalnya %W2 = 7 atau %W1 = %W2) Fungsi-fungsi *ASSIGNMENT* hanya dapat ditempatkan pada sisi paling kanan dari tampilan bagian(*Section Display*).

2.5.5.6 Jendela *Config*(Pengaturan)

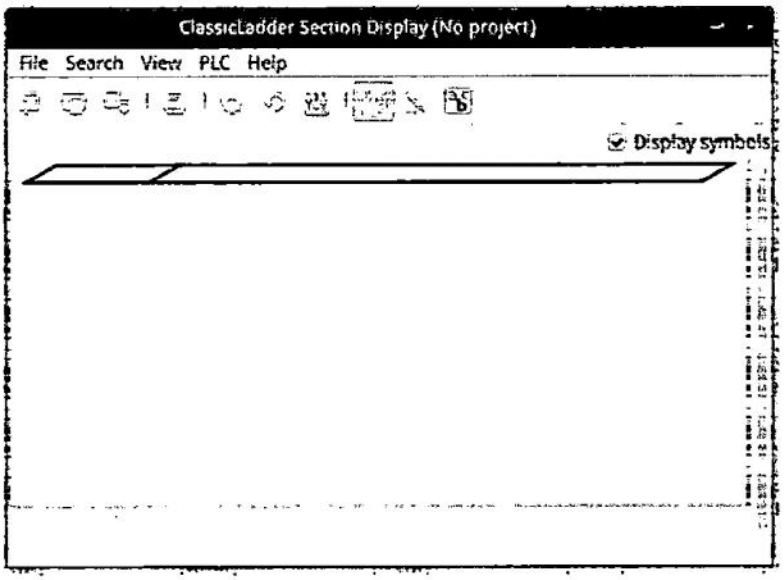
Jendela *config* menunjukkan status proyek yang aktual dan memiliki *tab-tab* pengaturan *Modbus* untuk komunikasi dengan *hardware* PLC.



Gambar 2.37. Jendela *Config*

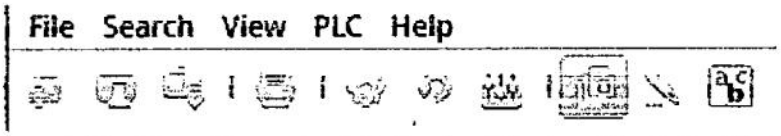
Tampilan yang dibahas di atas berlaku untuk Classic Ladder versi lama (0.7.124). Adapun untuk versi terbarunya (0.9.014) terdapat 2 jendela yang memiliki perbedaan cukup signifikan. Keduanya adalah jendela *Section Display* dan jendela *Editor*. Berikut penjelasan mengenai kedua jendela tersebut.

2.5.5.7 Jendela *Section Display* Versi Terbaru (0.9.014)



Gambar 2.38. Jendela *Section Display* Classic *Ladder* Versi Terbaru (0.9.014)

Pada jendela tersebut, bagian paling penting adalah sebagai berikut.



Gambar 2.39. Menu Utama Jendela Section Display Classic Ladder Versi Terbaru (0.9.014)

1. *Menu* (mulai dari kiri)

   a) *File*

      i. *New* - Membuat dan memulai proyek baru.

      ii. *Load* - Memuat proyek yang telah tersedia atau pernah dibuat sebelumnya.

      iii. *Save* - Menyimpan proyek aktual.

      iv. *Save As* - Menyimpan proyek aktual dengan nama tertentu.

      v. *Export to* - Ekspor proyek aktual ke format tententu.

- Svg: Ekspor proyek aktual ke sebuah *file* vektor dengan format SVG.
- Png: Ekspor proyek aktual ke sebuah *file* gambar *bitmap* dengan format PNG.
- *Clipboard*: Ekspor proyek aktual ke *clipboard* untuk dimasukkan ke aplikasi yang mendukung gambar.(Ctrl+C)

vi. *Preview* - Melihat pratinjau pencetakan proyek.

vii. *Print* - Mencetak proyek.

viii. *Quit* - Keluar dari aplikasi Classic Ladder.

b) *Search* - Menu Pencarian

i. *Find* - Pencarian obyek *ladder* dengan kata kunci.

ii. *Find Next* - Mencari hasil pencarian selanjutnya.

iii. *Find Previous* - Mencari hasil pencarian sebelumnya.

iv. *Go to First Rung* - Menuju ke *rung* pertama atau paling atas.

v. *Go to Last Rung* - Menuju ke *rung* terakhir atau paling bawah.

vi. *Go to Previous Section* - Mengarahkan menuju *section*/bagian sebelum(urutan) *section*/bagian aktual.

vii. *Go to Next Section* - Mengarahkan menuju *section*/bagian sesudah/setelah(urutan) *section*/bagian aktual.

c) *View* - Menu Tampilan

i. *Sections Window* - Menampilkan atau menyembunyikan jendela *Section Manager*.

ii. *Editor Window* - Menampilkan atau menyembunyikan jendela

Editor.

iii. *Symbols Window* - Menampilkan atau menyembunyikan jendela *Symbols Names*.

iv. *Bools Vars Window* - Menampilkan atau menyembunyikan jendela "Spy Bools Vars" (jendela pemantau variabel).

v. *Free Vars Window* - Menampilkan atau menyembunyikan jendela "Spy Free Vars" (pemantau variabel bebas).

vi. *Log Window* - Menampilkan jendela "Events Log" (laporan kejadian/peristiwa)

vii. *Frames Log Windows* - Menampilkan atau menyembunyikan jendela-jendela laporan untuk *hardware* PLC yang terhubung.

d) PLC - Menu terkait simulasi logika, pengaturan, dan hubungan *hardware* PLC.

i. *Run Logic* - Menjalankan simulasi rangkaian logika.

ii. *Run Logic Only One Cycle* - Menjalankan simulasi satu siklus.

iii. *Reset Logic* - Mengatur ulang logika-logika variabel yang terlibat dalam simulasi.

iv. *Configuration* - Membuka jendela pengaturan Classic Ladder.

e) *Help* - Menu bantuan.

i. *About* - Menampilkan informasi terkait rilis Classic Ladder aktual dan pengembangnya.

2. Tombol *Icon* (mulai dari kiri)

a)  (*New*)

b)  (*Load*)

c)  (*Save*)

d)  (*Print*)

e)  (*Run Logic*)

f)  (*Reset Logic*)

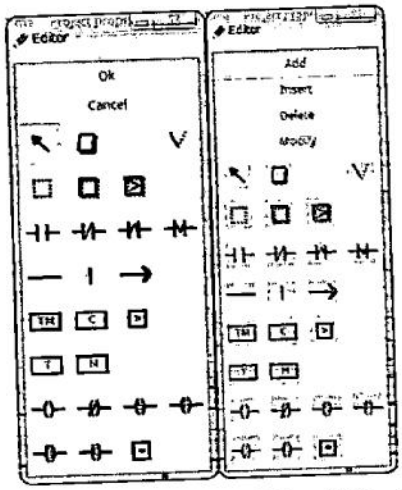
g)  (*Configuration*)

h)  (*Section Manager*)

i)  (*Editor Window*)

j)  (*Symbols Names*)

2.5.5.8 Jendela *Editor* Versi Terbaru (0.9.014)



Gambar 2.40. (a)Jendela *Editor* Aktif dan (b)Jendela *Editor* Inaktif

Bagian yang berbeda pada jendela ini dari versi lamanya adalah sebagai berikut.(dari kiri ke kanan)

Gambar 2.41. *Tool-tool Editor* yang Berbeda di Versi Terbaru Classic Ladder (0.9.014)

1. *Invert Logic of Object* - Membalikkan Logika dari Obyek.
2. *Select Rung Part* (*Drag and Release*) - Memilih/Menyeleksi bagian dari *rung*.
3. *Copy Rung Part* - Menyalin bagian dari *rung*.
4. *Move Rung Part* - Memindahkan bagian dari *rung*.

2.5.6 Obyek *Ladder* dalam Classic Ladder

Sumber: http://www.linuxcnc.org/docs/2.4/html/ladder_classic_ladder.html dan http://linuxcnc.org/docs/html/ladder/classic_ladder.html

2.5.6.1 KONTAK

Obyek ini mewakili *switch* atau kontak *relay*. Kontak-kontak tersebut dikendalikan oleh huruf variabelnya dan nomor yang ditetapkan pada kontak-kontak tersebut.

Huruf variabel tersebut dapat berupa B, I, atau Q dan angkanya dapat mencapai tiga digit misalnya %I2, %Q3, atau %B123. Variabel I dikendalikan oleh pin *input* HAL dengan angka yang sesuai. Variabel B adalah untuk kontak internal, yang dikendalikan oleh *coil* B dengan angka yang sesuai. Variabel Q dikendalikan oleh *coil* Q dengan angka yang sesuai. (seperti *relay* dengan beberapa kontak). Misalnya. Jika pin-HAL classicladder.0.in-00 bernilai benar, maka kontak N.O. %I0 akan menyala (tertutup, benar, apa pun sebutannya). Jika

coil %B7 diberi aliran listrik (*energized*) (menyala, benar, dll.) maka kontak N.O. %B7 akan menyala. Jika *coil* %Q1 diberi aliran listrik(*energized*) maka kontak N.O. %Q1 akan menyala (dan pin-HAL classicladder.0.out-01 akan bernilai benar.)

1. **KONTAK N.O.** (*Normally Open*) ⊣⊢ ketika variabelnya bernilai salah(*false*), saklarnya mati.
2. **KONTAK N.C.** (*Normally Close*) ⊣/⊢ - Ketika variabelnya bernilai salah(*false*), saklarnya menyala.
3. **KONTAK *RISING EDGE*** ⊣↑⊢ - Ketika perubahan variabel dari *false* ke *true*, saklar berdetak menyala.
4. **KONTAK *FALLING EDGE*** ⊣↓⊢ - Ketika perubahan variabel dari *true* menjadi *false*, saklar berdetak menyala.

2.5.6.2 *TIMER* IEC TIM

Obyek ini mewakili *timer* hitung mundur yang baru. *Timer* IEC menggantikan *timer* dan *monostable*. *Timer* IEC memiliki dua kontak sebagai berikut.

1. **I** = *input*
2. **Q** = *output*

Terdapat tiga mode, yaitu TON, TOF, dan TP. Berikut penjelasannya.

1. **TON:** Bila masukan *timer* bernilai benar, hitung mundur dimulai dan berlanjut selama masukan tetap bernilai benar. Setelah hitung mundur dilakukan dan selama masukan *timer* masih bernilai benar, *output* akan bernilai benar.

2. **TOF:** Bila masukan *timer* bernilai benar, *output* ditetapkan bernilai benar. Ketika *input* bernilai salah(*false*), *timer* menghitung mundur kemudian menetapkan keluaran menjadi bernilai salah(*false*).
3. **TP:** Bila masukan *timer* berdetak ke nilai benar atau memegang nilai benar, *timer* menetapkan keluaran bernilai benar sampai *timer* menghitung mundur.(*one-shot*)

Interval waktu dapat diatur dalam kelipatan 100ms(*milisecond*/milidetik), detik, atau menit.

Adapun berikut variabel untuk *timer* IEC yang dapat dibaca atau ditulis dalam blok-blok *in compare* atau *operate*.

1. **%TMxxx.Q**: waktu selesai (*Boolean*, baca tulis)
2. **%TMxxx.P**: waktu yang ditetapkan untuk *timer* xxx (baca tulis)
3. **%TMxxx.V**: nilai *timer* xxx (baca tulis)

Keterangan: baca tulis adalah *read write* dari sumber referensi.

2.5.6.3 *TIMER*

Obyek ini mewakili *timer* hitung mundur. *Timer* ini sudah ditinggalkan dan digantikan oleh *timer* IEC.

Timer memiliki empat kontak sebagai berikut.

1. **E** = *enable*/aktifkan (*input*) - mulai waktu ketika bernilai benar, me-*reset* ketika bernilai salah
2. **C** = *control*/kendalikan (*input*) - wajib pada *timer* untuk berjalan (biasanya terhubung ke E)
3. **D** = *done*/selesai (*output*) - bernilai benar ketika waktu *timer* habis dan

selama E masih bernilai benar

4. **R** = *running*/sedang berjalan (*output*) - bernilai benar ketika waktu berjalan

Basis waktu dapat berupa kelipatan milidetik, detik, atau menit.

Adapun berikut variabel untuk *timer* yang dapat dibaca atau tertulis dalam blok-blok *in compare* dan *operate*.

1. **%Txx.R**: Timer xx sedang berjalan (*boolean*, hanya baca)
2. **%Txx.D**: Timer xx selesai (*boolean*, hanya baca)
3. **%Txx.V**: nilai aktual Timer xx (*integer*, hanya baca)
4. **%Txx.P**: Timer xx ditetapkan (*integer*, baca atau tulis)

2.5.6.4 *Monostable* M

Obyek ini mewakili *one-shot timer*. *Monostable* sudah ditinggalkan dan digantikan oleh Timer IEC.

Monostable memiliki dua kontak, yaitu I dan R. I merupakan kontak *rising-edge* yang sensitif memulai timer hanya ketika nilai berubah dari *false* ke *true* (atau *off* ke *on*). Saat *timer* berjalan, kontak I dapat berubah dengan tidak berpengaruh terhadap waktu yang sedang berjalan. R akan bernilai benar dan tetap sama sampai *timer* tersebut selesai menghitung ke angka nol.

Basis waktu dapat kelipatan milidetik, detik, atau menit.

Adapun berikut variabel untuk *monostables* yang dapat dibaca atau tertulis dalam blok-blok *in compare* dan *operate*.

1. **%Mxx.R**: *monostable* xx berjalan(*running*) (*boolean*, hanya baca)
2. **%Mxx.V**: nilai aktual(*current value*) xx *Monostable* (*integer*, hanya baca)

3. **%Mxx.P**: ketetapan(*preset*) *Monostable* xx (*integer*, baca atau tulis)

2.5.6.5 *COUNTER* C

Obyek ini merupakan *counter* naik/turun(*up/down*). Pada obyek ini, terdapat tujuh kontak sebagai berikut.

1. **R** (*input*) - *reset* - akan mengatur kembali hitungan ke angka 0.
2. **P** (*input*) - *preset* - akan mengatur hitungan ke angka yang telah ditetapkan dari menu *edit*.
3. **U** (*input*) - *up count* - akan menambahkan satu ke hitungan.
4. **D** (*input*) - *down count* - akan mengurangi satu dari hitungan.
5. **E** (*output*) - *under flow* - akan bernilai benar ketika hitungan berpindah(*roll over*) dari angka 0 ke angka 9999.
6. **D** (*output*) - *done* - akan bernilai benar ketika hitungan sama dengan nilai yang ditetapkan.
7. **F** (*output*) - *overflow* - akan bernilai benar ketika hitungan berpindah(*roll over*) dari angka 9999 ke angka 0.

Kontak-kontak hitung naik dan turun tersebut sensitif tepi(*edge sensitive*) yang berarti bahwa kontak-kontak tersebut hanya menghitung ketika terjadi perubahan nilai kontak dari *false* ke *true* (atau *off* ke *on* dalam sebutan lain). Rentangnya adalah dari 0 sampai 9999.

Adapun berikut variabel untuk *counter* yang dapat dibaca atau tertulis dalam blok-blok *in compare* dan *operate*.

1. **%Cxx.D** : *counter* xx selesai(*done*) (*boolean*, hanya baca)
2. **%Cxx.E**: *counter* xx menyimpang kosong(*empty overflow*) (*boolean*, hanya

baca)

3. **%Cxx.F**: *counter* xx menyimpang penuh(*full overflow*) (*boolean*, hanya baca)
4. **%Cxx.V**: nilai aktual(*current value*) *counter* xx (*integer*, baca atau tulis)
5. **%Cxx.P**: ketetapan(*preset*) counter xx (*integer*, baca atau tulis)

2.5.6.6 *COMPARE*

Obyek ini berfungsi untuk melakukan perbandingan aritmatika. Misalnya apakah variabel %XXX=angka tertentu(atau angka yang dievaluasi) Blok *COMPARE* akan bernilai benar ketika perbandingannya benar. Sebagian besar simbol matematika dapat digunakan, seperti: +, -, *, /, =, <, >, <=, >=, (,), ^(eksponen), %(modulus), &(dan), |(atau), !(tidak).

Fungsi matematika juga dapat digunakan. Fungsi-fungsi tersebut adalah ABS(mutlak), MOY(Bahasa Perancis, untuk rata-rata), AVG(rata-rata). Misalnya ABS(%W2)=1, MOY(%W1,%W2)<3. Tidak terdapat spasi yang diperbolehkan dalam persamaan perbandingan. Misalnya %C0.V>%C0.P adalah ekspresi perbandingan yang valid/sah, sedangkan %C0.V>%CO.P bukan ekspresi yang valid/sah.

Adapun daftar variabel pada halaman selanjutnya(bagian Variabel-variabel ClassicLadder) yang dapat digunakan untuk membaca atau menulis untuk obyek *ladder*. Ketika blok baru *COMPARE* dibuka, pengguna harus memastikan dan menghapus simbol # ketika memasukkan/menuliskan perbandingan. Untuk mengetahui jika variabel kata(*word*) #1 kurang dari 2 kali nilai aktual *counter* #0, sintaks akan menjadi:

```
%W1<2*%C0.V
```

Untuk mengetahui jika bit 2 S32in sama dengan 10, sintaks akan menjadi:

```
%IW2=10
```

Catatan: *COMPARE* menggunakan sama dengan(*equals*) aritmatika "=", bukan sama dengan ganda(*double equals*) "==" yang biasa digunakan oleh para *programmer*.

2.5.6.7 *VARIABLE ASSIGNMENT*

Untuk *VARIABLE ASSIGNMENT*, misalnya penetapan angka tertentu (atau angka yang dievaluasi) untuk variabel %xxx. Terdapat dua fungsi matematika MINI dan MAXI yang memeriksa variabel untuk nilai-nilai maksimum(0x80000000) dan nilai-nilai minimum(0x07FFFFFF) (dengan berpikir tentang nilai yang ditandai(*signed values*)) dan menjaganya dari penyimpangan.

Ketika blok baru *VARIABLE ASSIGNMENT* dibuka, pengguna harus memastikan dan menghapus simbol # ketika memasukkan/menuliskan sebuah penugasan(*assignment*).

Untuk menetapkan nilai 10 ke *timer preset*(nilai yang ditetapkan untuk *timer*) *Timer* IEC 0, sintaks akan menjadi:

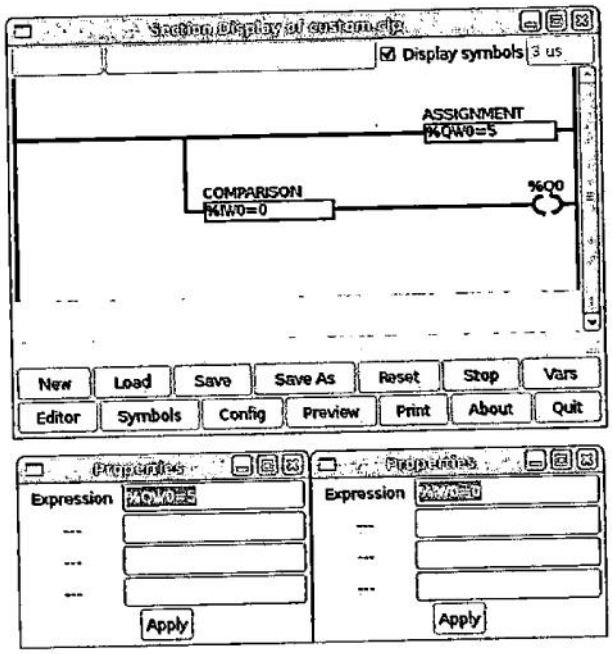
```
%TM0.P=10
```

Untuk menetapkan nilai 12 ke bit 3 S32out, sintaks akan menjadi:

```
%QW3=12
```

Gambar berikut menunjukkan penugasan(*assignment*) dan perbandingan(*comparison*). %QW0 adalah bit S32out dan %IW0 adalah bit S32in. Dalam hal ini pin-HAL classicladder.0.s32out-00 akan ditetapkan ke nilai

5 dan ketika pin-HAL classicladder.0.s32in-00 bernilai 0, pin-HAL classicladder.0.out-00 akan ditetapkan ke nilai benar(*True*).



Gambar 2.42. Contoh Menetapkan(*Assign*) / Membandingkan(*Compare*)

2.5.6.8 *COIL*

Coil merupakan kumparan/*coil relay*. *Coil-coil* tersebut dikendalikan oleh huruf variabelnya dan nomor yang ditugaskan. Huruf variabel dapat berupa B atau Q dan nomor dapat mencapai tiga digit nomor, misalnya %Q3, atau %B123. *Coil* Q mengendalikan pin-pin keluar HAL(HAL *out pins*). Misalnya jika &Q15 diberi energi(*energized*), maka pin-HAL classicladder.0.out-15 akan bernilai benar. *Coil* B adalah kumparan/*coil* internal yang digunakan untuk mengontrol aliran program.

1. ***COIL*** **N.O.** **–()–** - (kumparan/*coil relay*)

   Ketika kumparan tersebut diberi energi(*energized*), kontak N.O. akan ditutup/tertutup (*on*, benar, dll.)

2. **COIL N.C.** -|/|- - (kumparan/*coil relay* yang memiliki kontak invers.) Ketika kumparan/*coil* tersebut diberi energi(*energized*), kontak N.O. akan terbuka (*off*, salah, dll.)

3. ***COIL SET*** -(S)- - (kumparan/*coil relay* dengan kontak pengunci(*latching*)) Ketika kumparan/*coil* tersebut diberi energi(*energized*), kontak N.O. akan terkunci dan tertutup.

4. ***COIL RESET*** -(R)- - (kumparan/*coil relay* dengan kontak pengunci/*latching*) Ketika kumparan/*coil* tersebut diberi energi(*energized*), kontak N.O. akan terkunci dan terbuka.

5. ***COIL JUMP*** -(J)- - (sebuah *coil* "go to") Saat kumparan/*coil* diberi energi(*energized*), program tangga(*ladder program*) melompat ke anak tangga(*rung*) (pada bagian yang digunakan saat itu(*current section*)) - titik lompatan (*jump point*) ditunjukkan oleh label anak tangga(*rung label*). (Penambahan label anak tangga(*rung label*) dapat dilakukan dari tampilan bagian(*section display*), kotak label paling kiri dan paling atas(*top left label box*)).

6. ***COIL CALL*** -(C)- - (sebuah *coil* "go sub") Ketika kumparan/*coil* tersebut diberi energi(*energized*), program melompat ke bagian subrutin(*subroutine section*) yang ditunjukkan(*designated*) oleh nomor *subroutine* - *subroutine-subroutine* ditunjukkan(*designated*) oleh SR0 sampai SR9 (menandai(*designate*) *subroutine-subroutine* tersebut pada manajer bagian(*section manager*))

PERINGATAN Jika digunakan kontak N.C. dengan *coil* N.C.,

logika akan bekerja (saat kumparan/*coil* diberi energi(*energized*), kontak akan ditutup/tertutup), akan tetapi hal ini sangat sulit untuk diikuti.

2.5.6.9 *COIL JUMP* —(J)—

Sebuah *COIL JUMP* digunakan untuk 'melompat' ke bagian-lain seperti *go to* dalam bahasa pemrograman *BASIC*. Jika penggunan melihat bagian kiri atas dari jendela tampilan bagian(*section display window*), akan terlihat kotak label kecil dan kotak komentar yang lebih panjang di sampingnya. Jika pengguna mengakses bagian *Editor-->Modify* kemudian kembali ke kotak kecil, pengguna dapat mengetik nama. Jika pengguna meneruskan, pengguna dapat menambahkan komentar di bagian komentar(*comment section*) tersebut. Nama label ini adalah nama anak tangga(*rung name*) yang sedang dimodifikasi saja dan digunakan oleh *COIL JUMP* untuk mengidentifikasi ke mana tujuannya. Ketika menempatkan *COIL JUMP*, pengguna harus menambahkannya pada posisi yang paling tepat dan mengubah label ke anak tangga(*rung*) yang menjadi tujuan lompatan.

2.5.6.10 *COIL CALL* —(C)—

Sebuah *COIL CALL* digunakan untuk menuju ke bagian *subroutine* kemudian kembali - seperti "go sub " dalam bahasa pemrograman BASIC. Jika pengguna menuju ke jendela manajer bagian (*section manager window*), pengguna dapat menekan tombol *add*. Pengguna dapat menamai bagian tersebut, memilih bahasa yang akan digunakan (tangga/*ladder* atau *sequential*), dan memilih jenis (utama/*main*) atau subrutin/*subroutine*).

Jika pengguna memilih nomor *subroutine* (SR0 misalnya), sebuah bagian(*section*) yang kosong akan ditampilkan dan pengguna dapat

membangun/membuat *subroutine*-nya. Adapun jika pengguna telah melakukan hal tersebut, pengguna dapat kembali ke manajer bagian(*section manager*) dan mengklik pada bagian utama (nama bawaan/*default name*: prog1).

Dengan demikian, pengguna dapat menambahkan *COIL CALL* untuk programnya. *COIL CALL* harus ditempatkan pada posisi yang paling tepat pada anak tangga(*rung*).

Satu hal yang perlu diingat, yaitu mengubah label untuk nomor *subroutine* yang dipilih sebelumnya.

2.5.7 Variabel-variabel dalam Classic Ladder

Variabel-variabel ini digunakan dalam MEMBANDINGKAN(*COMPARE*) atau BEROPERASI(*OPERATE*) untuk mendapatkan informasi tentang atau mengubah spesifikasi dari obyek-obyek *ladder*. Seperti mengubah *preset counter* (nilai yang ditetapkan), atau melihat jika *timer* selesai berjalan. Berikut daftarnya:

1. **%Bxxx**: memori bit xxx (*Boolean*)
2. **%Wxxx**: memori kata(*word*) xxx (bilangan bulat 32 bit yang ditandai(32 *bit signed integer*))
3. **%IWxxx**: memori kata(*word*) xxx (pin masuk S32)
4. **%QWxxx**: memori kata(*word*) xxx (pin keluar S32)
5. **%IFxx**: memori xx kata(*word*) (*float in pin*) (dikonversi ke S32 pada ClassicLadder)
6. **%QFxx**: memori xx kata(*word*) (*float out pin*) (dikonversi ke S32 pada ClassicLadder)
7. **%Txx.R**: *timer* xx berjalan(*running*) (*Boolean*, pengguna hanya dapat

membaca)

8. **%Txx.D**: *timer* xx selesai(*done*) (*Boolean*, pengguna hanya dapat membaca)
9. **%Txx.V**: nilai aktual(*current value*) *timer* xx (integer, pengguna hanya dapat membaca)
10. **%Txx.P**: nilai yang ditetapkan untuk *timer* xx (bilangan bulat(*integer*))
11. **%TMxxx.Q**: *timer* xxx selesai(*done*) (*Boolean*, baca tulis)
12. **%TMxxx.P**: ketetapan(*preset*) *timer* xxx (bilangan bulat(*integer*), baca tulis)
13. **%TMxxx.V**: nilai aktual(*current value*) *timer* xxx (bilangan bulat(*integer*), baca tulis)
14. **%Mxx.R**: *monostable* xx berjalan(*running*) (*Boolean*)
15. **%Mxx.V**: *monostable* nilai aktual(*current value*) xx (bilangan bulat(*integer*), pengguna hanya dapat membaca)
16. **%Mxx.P**: ketetapan(*preset*) *monostable* xx (bilangan bulat(*integer*))
17. **%Cxx.D**: *counter* xx selesai(*done*) (*Boolean*, pengguna hanya dapat membaca)
18. **%Cxx.E**: *counter* xx *empty overflow* (*Boolean*, pengguna hanya dapat membaca)
19. **%Cxx.F**: Counter xx *full overflow* (Boolean, pengguna hanya dapat membaca)
20. **%Cxx.V**: nilai aktual(*current value*) *counter* xx (bilangan bulat/*integer*)
21. **%Cxx.P**: ketetapan(*preset*) *counter* xx (bilangan bulat/*integer*)
22. **%Ixxx**: masukan fisik xxx (*Boolean*) - bit masukan HAL
23. **%Qxxx**: keluaran fisik xxx (*Boolean*) - bit keluaran HAL

24. **%Xxxx**: aktivitas langkah(*activity of step*) xxx (bahasa sekuensial)
25. **%Xxxx.V**: waktu aktivitas dalam detik dari langkah xxx (bahasa sekuensial)
26. **%Exx**: kesalahan(*error*) (*Boolean*, baca tulis (akan ditimpa))
27. **Variabel yang Diindeks atau Divektorkan(*Vectorized and Indexed Variables*)**

Ini merupakan variabel yang diindeks oleh variabel lain. Beberapa sumber mungkin menyebutnya sebagai variabel yang divektorkan. Contoh: %W0[%W4] => jika %W4 sama dengan 23 , ini sesuai dengan %W23.