

## TITLE OF THE INVENTION:

### METHOD AND APPARATUS FOR UTILIZING A SEQUENCE INTERPRETER APPROACH TO CONTROL LOGIC OF A PROGRAMMABLE LOGIC CONTROLLER

## BACKGROUND OF THE INVENTION

[0001] The present invention pertains to programmable logic controllers (PLCs), and more particularly, to PLCs that utilize a sequential interpreter approach in executing their control logic.

5 [0002] The use of PLCs has revolutionized the process control industry. One example of the use of PLCs in process control is in the manufacture of semiconductors. The blending of two or more fluids to form a blended gaseous mixture for combining a semiconductor liquid or gas with a carrier gas for delivery to a fabrication tool requires precise process control. Typically, controlling this type of process by PLC involves using  
10 sequence control logic (i.e., opening/closing valves, checking alarm setpoints, etc.) in the PLC and is accomplished via sequential ladder logic programming. This is the standard approach and works well for a fixed control scheme architecture. However, any changes to the control scheme requires ladder logic modifications by an individual with this skill set. Moreover, the memory usage by such fixed control scheme architecture is  
15 inefficient.

[0003] In particular, classical PLC software utilizes a ladder logic design, which has accounted for a large portion of programming for standard automation applications. The hierarchy of this type of application is shown in Fig. 1. Applications can be constructed which are fairly complex using this programming technique, and the applications which  
20 result are able to be maintained by a large cross-section of the process control community, as well as local engineers and technicians with a minimum of training. Although this programming technique is easily constructed, it is not particularly efficient in memory usage when compared with other available languages. There has remained a need to construct code segments that could be used for multiple gas stream control  
25 such as first and second input flows of a gas cabinet where the input flows are blended. Unfortunately, ladder logic does not easily accomplish this.

[0004] By way of example only, the Assignee of the present invention, namely, Air Products & Chemicals, Inc., has used PLCs in its product line and with the release of its Gasguard® 450 (approximately 1990) product line, the technique of using external configurable data to control system operation has been used for gas cabinet-related equipment. Subsequent generations of Gasguard®, including Gasguard® 250, 500, AP2 and AP3, have all used this same method. However, the primary difference is that each of the aforementioned products utilizes an embedded control system of various types and manufacturers.

[0005] Attempts at overcoming these problems included developing a code for a PLC application for a mass-customized (MC) gas cabinet but that application rapidly ran out of memory when using this direct programming method. It was suggested to use custom STL (statement list) routines and SCL (structured control language) routines to supplement a basic top-level ladder logic routine in areas that would result in a savings of memory and would allow code segment reuse when possible. One of the areas of the memory usage problem is the standard call sequence in the PLC. Each call to a routine resulted in a large overhead memory usage and a low level STL routine was constructed that allowed multiple-called code to be used without this overhead memory usage. This reduced the memory usage by a significant amount. The second area of the memory usage problem was the lack of timers available in the chosen hardware set. Because of the low number of timers available, a custom timing routine was constructed in STL. Although the inclusion of STL in the application was not desired, it was included because of memory limitations. Many of the analog scaling and alarm functions would be coded using SCL, which, although less desirable than ladder logic, was clearly more desirable than the STL segments. The resulting hierarchy is shown in Fig. 2.

[0006] The resulting PLC application software utilized ladder logic for the overall execution loop, with SCL custom sequences and low-level STL service routines. The application configuration data was used to customize the individual application with selected options and alarm limits, etc. However, even with these compromises in programming approach, the application clearly was not able to fit within the constraints of the selected hardware. The resulting code consisted of three separate programming languages and the STL segments were seen as a potential problem for code maintenance.

[0007] Thus, there remains a need for a PLC that utilizes the external control data technique on a PLC-based control system which is not typically utilized in this manner.

[0008] All references cited herein are incorporated herein by reference in their entireties.

5

#### BRIEF SUMMARY OF THE INVENTION

[0009] A method for controlling a process using a programmable logic controller (PLC) having a central processing unit (CPU) which includes a load memory which stores all instructions and data required for controlling the process and an interpreter that  
10 evaluates and executes instructions in a work memory to control the process. The method comprises the steps of: (a) creating instructions comprising a format that includes at least one option code that is coded as being enabled or disabled; (b) copying one instruction from the load memory to the work memory; (c) evaluating, by the interpreter, the at least one option code to determine if the option code is true; (d)  
15 executing the instruction if the option code is evaluated by the interpreter to be true, or not executing the instruction if the option code is evaluated by the interpreter to be false and if so to copy the next instruction from the load memory to the work memory and repeating step (c) for this next instruction; (e) checking to see if the instruction whose option code has been evaluated to be true has been completed and if not to continue to  
20 execute the instruction to completion; and (f) if the instruction whose option code has been evaluated to be true has been completed, repeating steps (b)-(f) until all of the instructions have been evaluated and executed where evaluation of said at least one option code in the corresponding instruction has been determined to be true.

[0010] A programmable logic controller (PLC) for controlling a process coupled thereto  
25 and wherein the PLC includes a central processing unit (CPU) which includes a load memory that stores all instructions and data required for controlling the process and an interpreter that evaluates and executes instructions in a work memory to control the process. The PLC comprises: the load memory containing read-only instructions wherein each instruction comprises a format that includes at least one option code that is coded  
30 as being enabled or disabled; and the work memory that receives only one instruction at a time, wherein the interpreter evaluates the at least one option code to determine if the at least one option code is true and to execute the one instruction if the at least one

option code is true, or receiving the next instruction if the least one option code is evaluated to be false.

#### BRIEF DESCRIPTION OF SEVERAL VIEWS OF THE DRAWINGS

- 5    **[0011]** The invention will be described in conjunction with the following drawings in which like reference numerals designate like elements and wherein:
- [0012]** Fig. 1 is a hierarchy of a conventional PLC software application;
- [0013]** Fig. 2 is a hierarchy of another PLC software application directed to solving memory usage concerns;
- 10    **[0014]** Fig. 3 is a hierarchy of the present invention PLC software application that overcomes the memory usage concerns while reducing the level of complexity to implement a change;
- [0015]** Fig. 4 is a front view of an exemplary gas cabinet having a PLC mounted inside an upper portion thereof;
- 15    **[0016]** Fig. 5 is a front view of an exemplary touch screen for communicating with the PLC;
- [0017]** Fig. 6 is a functional diagram of the central processing unit (CPU) of a PLC comprising the hierarchy of the present invention;
- [0018]** Fig. 7 is a diagram of the contents of the load memory of the CPU of a PLC
- 20    comprising the hierarchy of the present invention;
- [0019]** Fig. 8 is a chart that defines the format of the sequence command structure of the present invention;
- [0020]** Fig. 9 is a diagram depicting a sequence data block in the load memory;
- [0021]** Fig. 10 is a diagram of the contents of the work memory of the CPU of a PLC
- 25    comprising the hierarchy of the present invention;
- [0022]** Figs. 11A-11C together constitute a chart of the 96 option codes in an exemplary configuration showing the unique setting level (yes or not) for each option code which is set by the configuration programmer; and
- [0023]** Fig. 12 is a flow diagram of the PLC interpreter application that overcomes the
- 30    memory usage concerns while reducing the level of complexity to implement a change.

## DETAILED DESCRIPTION OF THE INVENTION

[0024] The present invention uses a non-traditional approach to programming sequential control logic in a PLC whereby sequential control is accomplished using  
5 separate, pre-configured binary data. In particular, the present invention concentrates on SCL (structured control language) and ladder logic, but also introduces the concept of a sequence interpreter and sequence datablocks. This approach is chosen to minimize the memory usage and to allow a more flexible configuration environment using pre-tested code in a mass-customized environment.

10 [0025] As mentioned earlier, this approach is different from the traditional straight-line programming model and utilizes a meta-code interpreter. Under this model, individual sequences (or events, e.g., purge valve #1, purge valve #2, open vent, etc.) can be represented in data blocks that are interpreted by a meta-code interpreter. The data to represent the sequence is only a fraction of the size that would be needed to construct  
15 custom code to accomplish the sequence. Although the size of the meta-code interpreter may be larger than any one sequence, the same code is used for all sequences and this results in a tremendous reduction in required memory. The use of this interpreter model also allows for modifications to be made in individual sequences without the need for the underlying code to be re-verified because individual code  
20 segments are not being changed, only data is changed. The hierarchy 20 of this approach of the present invention is shown in Fig. 3.

[0026] Fig. 4 depicts a front view of an exemplary gas cabinet 2 that contains various valves, pressure switches, gas cylinders, etc. (all of which are hereinafter referred to as  
25 "equipment 4"). This equipment 4 is controlled by the PLC (not shown) which is located inside the upper portion 6 of the gas cabinet 2. Although a variety of input/output (I/O) devices for communicating with the PLC are known, one exemplary I/O device is a touch screen 8, as shown in Fig. 5.

[0027] As shown in Fig. 6, the central processing unit (CPU) of the PLC (by way of example only, a Siemens PLC model no. S7-300) comprises a load memory 22 and a  
30 work memory 24. Because the sequence data is contained completely within data blocks, it becomes possible to maintain sequence data blocks on the load memory (e.g., a flash card) and load data blocks for sequences into the work memory 24 as needed. This further reduces the memory usage. The only sequence that needs to be maintained

in the work memory 24 is the idle sequence, since this sequence is the default sequence for the termination of all other sequences. The PLC controls the gas cabinet equipment 2 for the process (e.g., a customer's fabrication tool). It should also be noted that a virtual manifold box (VMB) 2A can be coupled to the gas cabinet 2 which permits a single gas cabinet 2 to supply multiple use points.

[0028] By using a separate data source to configure the sequence of events in the PLC, two problems are solved:

[0029] 1. Internal PLC memory constraints are bypassed since the binary control data is stored in a separate, non-volatile flash memory area, which is available at a reasonable cost and can be expanded as needed; and

[0030] 2. Configuration of the PLC-based control system can be accomplished using the same proven tools and techniques utilized on previous generation embedded systems.

[0031] The load memory 22 is a non-volatile flash card and contains ready-only sequence data (Fig. 7). Each sequence data block consists of a series of command instructions. The sequence data block size is expandable but currently contains a maximum of 160 command instructions. Each individual sequence command instruction consists of, by way of example only, six bytes of data, as shown in Fig. 8. Fig. 9 provides an example of a sequence data block in the load memory 22 showing three sequence commands, namely, sequence commands #1, #2 and #3.

[0032] The work memory 24 is internal CPU volatile data of limited size. A unique current sequence data block and subsequence data block is maintained for each process line (see Fig. 10). The subsequence data block basically are internal loops or macros that are called by the sequence data block to perform any repetitive function. Current sequence data PLx block size is large enough to handle one command instruction (six bytes). Currently executing command remains in this block until completely executed.

[0033] Depending on the particular application, i.e., process to be controlled by the PLC, the configuration programmer, using a software tool, selects which option codes (also referred to throughout this Specification as "opcodes") are to be active in the process. By way of example only, there are currently 96 option codes available that can be used to configure the PLC to provide for particular control of a particular process.

Thus, before a particular PLC configuration is delivered to a customer, the configuration programmer selects those option codes (from the overall 96 available option codes) that are necessary to control the particular customer's process. "Selecting an option" code involves encoding the last two bytes of the command structure so that when these two  
5 last bytes are evaluated by the interpreter, they both evaluate to "true." If just one of these last two bytes do not evaluate to "true", the command they are associated with is not executed. Conversely, if both of these bytes evaluate to "true" the associated command is then executed. In this manner, the 96 overall Option Codes do not have to added or deleted from the actual software; instead, they are either enabled or disabled  
10 by use of the opcode evaluations. Figs. 11A-11C form a chart that provides an exemplary listing of the option codes for a particular process configuration. As can be seen, all of the option codes are available and it is up to the configuration programmer to select which option codes are to be implemented. As can also be appreciated, each option code can be designated with a unique hex value for "yes" or "no".

15 **[0034]** Fig. 12 provides a flow diagram of the PLC interpreter operation in executing the invention of the present application.

**[0035]** When a sequence is begun, the first command instruction (consisting of 6 bytes) is copied from the appropriate sequence data block in the load memory 22 to the corresponding process line's data block in the work memory 24. The format of the  
20 sequence command structure is given in Fig. 8. When the sequence interpreter runs (i.e., every scan), the interpreter attempts to execute the command in this local data block.

**[0036]** The last two bytes of the command structure, OpCode1 and OpCode2, are evaluated to determine if the command should be run or skipped; see Figs. 8 and 9. In  
25 order for the command to be executed, both opcode parameters must be true. These parameters contain values which relate to option codes which are utilized by the configuration programmer to determine if an option currently exists in the system. An opcode with a value of zero (0) indicates the parameter is not used and will always evaluate as true. By way of example only, there are currently 96 option codes available  
30 to be used to configure the software.

**[0037]** If the opcode check evaluates to true, the command is executed by the sequence interpreter. Conversely, if the opcode check evaluates to false, the command is not executed.

- 5      **[0038]** The sequence interpreter returns two Boolean parameters to the calling function; SeqStepResult and SeqEndResult. SeqStepResult indicates whether the command (step) has been completed. Some commands will take numerous CPU scans to complete and will return a false SeqStepResult until it eventually completes the command. When a SeqStepResult of true is received, the sequence command is incremented to the next step, while a SeqStepResult of false will prevent the next command from being retrieved and evaluated. The SeqEndResult indicates that the end of the sequence has been reached. When this parameter is true, the sequence interpreter stops evaluating commands for this process line.
- 10    **[0039]** Every CPU scan, the process line indicator variable is incremented. When the maximum process line is reached, the process line variable indicator returns to process line 1. Assuming sequences are running for every process line, the sequence interpreter first evaluates the current command for process line 1. On the next scan, the sequence interpreter evaluates the current command for the next process line. This continues to
- 15    the last process line at which time the sequence interpreter returns to process line 1 at the next CPU scan.

**[0040]** Using the system and method of the present invention provides the additional benefits of:

- 20      -lower system memory requirements translates to lower system costs;
- operational modifications to system need not be done by a PLC programmer;
- sequence interpreter logic is portable and can be reused on other product lines;
- configuration data can be assembled to accommodate all system options (master template);
- 25      -configuration data is encapsulated into several data blocks, which can be easily stored, validated and revision-controlled;
- invalid command instructions can be detected and subsequent appropriate error handling measures taken;
- future changes can be accomplished without modifying existing, tested
- 30      underlying logic code;

-additional sequence logic blocks can be added without impacting work memory;

-sequence interpreter command list can be expanded or enhanced as needed;

5            -sequence logic can be programmed once to handle all possible options and configurations; and

-individual logic instructions can be enabled or disabled via simple Boolean variables (option codes). This is performed at run-time.

10        **[0041]** Initial efforts have indicated that this approach allows both the MC Cabinet and code sets for the VMB 2A (in particular, the eight-stick VMB 2A, where the eight refers to eight individual flow lines) to be accomplished with the present hardware set, while preserving much of the flexibility of controller series of the Assignee.

15        **[0042]** The shift to this type of approach toward the integration of sequences through data blocks should allow the eight-stick VMB 2A to be fielded using the present hardware platform instead of requiring the use of a larger PLC. This further reduces the number of standard part numbers needed for manufacturing, since one controller can be used for both gas cabinets and eight-stick VMBs. This approach may also allow expanded use of existing editors as a basis for configuration, which would reduce the amount of retraining necessary for manufacturing configuration.

20        **[0043]** While the invention has been described in detail and with reference to specific examples thereof, it will be apparent to one skilled in the art that various changes and modifications can be made therein without departing from the spirit and scope thereof.

## CLAIMS

1. A method for controlling a process using a programmable logic controller (PLC) having a central processing unit (CPU) which includes a load memory which stores all instructions and data required for controlling the process and an interpreter that evaluates and executes instructions in a work memory to control the process, said method comprising the steps of:

(a) creating instructions comprising a format that includes at least one option code that is coded as being enabled or disabled;

(b) copying one instruction from said load memory to said work memory;

(c) independently evaluating, by said interpreter, said at least one option code to determine if said option code is true;

(d) executing said instruction if said option code is evaluated by said interpreter to be true, or not executing said instruction if said option code is evaluated by said interpreter to be false and if so to copy the next instruction from said load memory to said work memory and repeating step (c) for this next instruction;

(e) checking to see if said instruction whose option code has been evaluated to be true has been completed and if not to continue to execute said instruction to completion; and

(f) if said instruction whose option code has been evaluated to be true has been completed, repeating steps (b)-(f) until all of said instructions have been evaluated and executed where evaluation of said at least one option code in the corresponding instruction has been determined to be true.

2. The method of claim 1 wherein said step of creating instructions comprises generating sequence data blocks, each sequence data block comprising a series of instructions required to accomplish a unique sequence or event in controlling the process.

3. The method of claim 1 wherein said format comprises two option codes wherein each of said option codes is coded as being enabled or disabled and wherein said step of executing said instruction occurs only if said interpreter evaluates both of said option codes as being true and not executing said instruction if at least one of said option codes is evaluated to be false.
4. The method of claim 1 wherein each of said instructions comprises six bytes.
5. The method of claim 4 wherein said instruction format further comprises a plurality of parameters in addition to said at least one option code, each of said parameters and said at least one option code being one byte in size.
6. The method of claim 1 wherein an option code of zero is always evaluated by said interpreter to be true.
7. The method of claim 1 wherein the process is a gas flow process and wherein said PLC controls process equipment in a gas cabinet.