프로그램형 논리 제어기의 고속화를 위한 래더 언어 해석기의 구현

김 형 석, 권 욱 현
서울대학교 전기공학부 제어정보시스템 연구실

Implementation of Ladder Diagram Translator for High-Speed Programmable Logic Controller

Hyungseok Kim, Wookhyun Kwon
Control Information Systems Lab., Seoul National University, Seoul, Korea

Abstract - This paper proposes a translation method that converts ladder diagrams to binary executable codes for PLC (programmable logic controller)s. A PLC based on general purpose DSP(digital signal processor) validates the method. We performed a benchmark on the system that compares the execution time of the interpretation method and ours. Experimenal result shows how fast this method executes programs that consist of codes generated.

1. 서 론

프로그램형 논리 제어기(Programmable Logic Controller, 이하 PLC)는 계전기 회로(electromagnetic relay circuit)를 사용하는 순차 제어기를 교체하기 위해 개발되었는데, 현재 논리, 순차, 타이밍, 카운트, 연산 등을 기본으로 하는 제어 시스템을 구현하는데 널리 사용되고 있으며, 그 응용 분야는 조립 공정, 화학 공정, 운송 시스템, 에너지 시스템 등의 고기능 자동화를 위한 디지털 시스템 제어기의 여러 분야에 이른다[1,2]. 자동화 시스템이 더욱 발전함에 따라, PLC가 처리해야 하는 작업들은 더욱 더 복잡해져 왔으며, 이러한 작업들을 보다 능률적으로 처리하기 위해서, PLC는 기본 래더 다이어그램(Ladder Diagram, 이하 LD) 외에 100여종 이상의 다양한 응용 명령어를 보유하게 되었고, SFC(Sequantial Function Chart), FBD(Function Block Diagram), ST(Structured Text) 등과 같은 차세대 고급 언어를 지원하게 되었다. 이러한 고급 언어들은 래더 언어의 수행 방법을 근간으로 하며, 프로그램 작성시의 복잡도를 덜어준다[1].

실시간 제어를 수행하여야 하는 PLC에서는 수행 속도가 무엇보다도 중요한 요인이 되나, 수행하여야 하는 작업이 복잡해짐에 따라 수행 속도에 대한 요구사항은 고성능의 마이크로프로세서를 손쉽게 사용할 수 있는 현재에도 여전히 해결해야 할 과제로 남아 있다. PLC가 발전함에 따라서 그 적용 범위 또한 넓어져, PLC 프로그램에서 복잡한 실수 계산 등 연산 시간이 오래 걸리는 응용 명령어가 차지하는 비중도 높아지고 있다. PLC 언어의 특성상 흐름 제어, 단순 메모리 입출력 등이 복잡한 응용 명령어와 긴밀히 결합되어, 일반 컴퓨터 응용보다 프로그램 수행 속도의 증가는 용이하지 않다[1].

PLC의 성능 향상에 대한 기본적인 접근 방법 중의 하나로, 새로운 구조의 연산 프로세서에 관한 연구가 계속되어 왔다. 병렬 처리 기법을 이용하거나[3,4,5] 전용 집적회로(ASIC) 설계 기법을 이용하여[6,7,8], PLC 전용의 프로세서를 개발하여 성능 향상을 추구하는 것이 그 예이다. 이와 같은 기법은 하드웨어 자원을 투자하여 프로세서 성능을 근본적으로 개선하는 접근 방법이 되나 비용 면에서 불리하며, 전용 하드웨어에의 의존도가 높아 확장 및 변경이 용이하지 않다는 단점을 수반한다.

명령 축약형 프로세서(RISC)에 관한 연구들[6,7]은 범용 프로세서에 PLC의 논리 명령어와 기본적인 응용 명령어를 수행하는 명령어를 추가하였다. 이러한 프로세서는 PLC의 수행 속도를 증가시키는 가장 효과적인 방법이 된다. 그러나, 빠르게 발전하는 복잡한 응용 명령어들을 추가로 수용해야 할 문제와 가격대 성능비가 눈부시게 발전하는 범용의 마이크로프로세서와는 달리 짧은 기간 내에 추가 설계,제작하는데 매우 불리하므로, 빠르게 변화하는 수요를 충족시키는데 어려움이 발생한다.

범용 마이크로 프로세서를 사용하는 PLC의 경우에는 전용 프로세서를 사용하는 PLC에 비하여 성능면에서 불리하나, 일반적으로 성능의 향상 주기가 특수 프로세서에 비해 빠르고 값싼 비용으로 빠른 기간 내에 설계, 제작할 수 있는 장점이 있다. 범용 마이크로 프로세서를 사용하는 PLC에서 수행 속도의 향상 문제는 운영 체제와 그 내부의 명령어 해석, 실행 방법의 개선으로 정리된다. 이와 같은 소프트웨어에 의한 성능 향상은 비용면에서 매우 유리할 뿐 아니라 확장성과 변경의 용이성도 동시에 제공한다.

범용의 마이크로 프로세서를 사용하여 PLC를 구성하는 경우에는 PLC 명령어와 마이크로프로세서의 명령어에 내재하는 문법의 격차에서 많은 성능의 손실이 발생한다. 현재까지 널리 사용되는 해석 방법은 이와 같은 문법 격차에서 오는 성능 손실을 크게 고려하지 않는다. 과거의 프로세서와는 달리 파이프라인 구조를 사용하는 고성능 프로세서의 경우, 성능 손실은 더욱더 크게 증폭된다. 따라서, 이러한 문제를 극복하는 해석 방식이 필요하게 된다.

본 논문의 내용과 구성은 다음과 같다. 2장에서는 프로그램형 논리 제어기에서의 일반적인 LD의 해석 방법을 서술하고 성능을 향상시킬 수 있는 LD의 해석과 실행을 위한 구조를 제안한다. 3장에서는 실험을 위하여 제작한 PLC 시스템에 대해 간단히 언급하고 실험을 통한 평가로 두 방식 간의 성능 차이를 비교, 분석한다. 4장에서는 성능 향상과 앞으로의 추세 등을 언급하고 끝을 맺는다.

2. LD 해석기의 구조

2.1 일반적 구조

그림 1은 응용 프로그램 메모리와 시스템 프로그램 메모리에 저장되어 있는 프로그램과 데이터의 동작 원리를 자세히 보여준다. PLC가 제어를 시작하면 시스템 프로그램은 다음과 같은 동작을 반복한다.

ㄱ) 응용 프로그램 메모리에서 명령어 단위로 읽어오고 해독한다.
ㄴ) 각 명령어와 대응하는 실행 루틴을 호출한다.
ㄷ) 오퍼랜드(operand)를 해석하고 입출력 테이블 메모리에서 데이타를 읽는다.
ㄹ) 지정된 동작을 수행한다.

ㅁ) ㄱ)으로 돌아가서 반복한다.

PLC는 제어기기 형태로, 내장된 사용자 인터페이스를 가지고 있지 않은 것이 일반적이다, 대신에 계산기나 컴퓨터의 자판과 흡사한 프로그래밍 패널이 통신 포트를 통하여 연결되어 있다. PLC에서의 프로그래머는 직접 래더 다이어그램을 그리거나 인스트럭션을 입력하여 응용 프로그램을 구성하고 응용 프로그램 메모리에 이를 저장한다, 그러면 프로그램형 논리 제어기는 이 프로그램을 중간 코드로 변환하고 자신의 메모리에 로드시키게 된다. 그리고 난 후에 프로그램형 논리 제어기를 동작시키게 되면, 프로그램형 논리 제어기 내부의 명령어 번역기가 중간 코드를 하나씩 가져와서 이를 해독한 후 그 명령을 수행하는 서브 루틴을 호출하여 그 동작을 수행하게 된다.

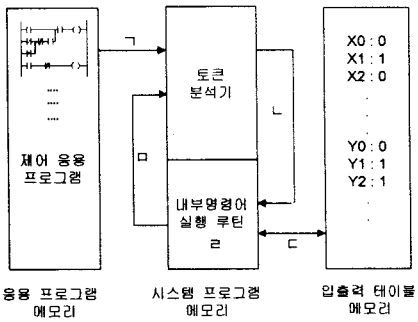

그림 1. LD 프로그램의 해석 과정

그림 1에서 보여졌듯이 기존의 해석 방식에서는 각 내부 명령어의 실행을 위한 서브 루틴들이 있어서 이를 호출하여 이용하기 때문에 여러 번의 호출(call), 회귀(return)가 필요하게 된다. 즉, 기존의 해석 방식은 제어 동작 중에 해석, 실행이 되기 때문에 제어기 내부에 저장된 응용 프로그램의 원시 코드를 하나씩 읽고 해석하여 미리 작성된 매크로 루틴으로 대체하여 수행할 수 밖에 없어 수행 속도가 저하될 뿐더러, 최근에 추가되는 특수 연산 등의 실행 시간이 매우 길고 크기가 큰 연산 명령의 경우 시간의 지연이 크게 되어 이 두 가지 시간 지연 요소의 조합으로 인해 프로그램형 논리 제어기의 수행 성능의 향상을 기대할 수 없게 된다. 따라서, 이러한 해석 방식의 느린 속도와 비효율성을 극복하기 위한 효율적이고 고속인 해석과 실행 방식이 필요하게 된다.

2.2 PLC 명령어 해석 방식의 제안

제안하는 명령어 해석 방식은 오프 라인(off-line) 상태에서 외부의 시스템 상에서 내부 명령어를 프로그램형 논리 제어기 시스템에서 동작할 수 있는 기계 언어인 실행 코드로 변환하는 것이다. 즉, 사용자가 래더 다이어그램을 작성하고 명령어 해석기를 실행하여 그것을 최종적으로 실행 가능한 기계 언어로 변환시켜서 프로그램형 논리 제어기에 다운로드하면 그대로 실행하는 방법이다. 그런데, 이 방식을 위해서는 래더 다이어그램에서 실행 코드를 만들어주고 이를 다운로드해 줄 수 있는 시스템이 필요하다. 이를 위해서 외부에서 상호간에 데이터들을 주고받을 수 있는 시스템을 구축하는데 쉽게 구할 수 있고 그래픽 유저 인터페이스(Graphic User Interface)가 뛰어나며 개방된 프로토콜을 갖추어 통신 방식이 쉽게 구현되는 보통의 개인용 컴퓨터가 적합하다. 이 외부의 시스템을 로더(loader)라고 정의하고 여기에 프로그램형 논리 제어기를 연결하여 적당한 통신 방법으로 목적 코드를 다운로드한다.

래더 프로그램이 그림 2와 같은 내부 명령어들로 치환되고, 이 내부 명령어들로 이루어진 프로그램은 로더 내부의 명령어 해석 프로그램에 의해 사용되는 프로세서와 호환되는 어셈블리 코드로 변환되어 어셈블리어 프로그램이 출력된다. 원시 코드(source code)에 일정한 포맷(format)이 있기 때문에 어휘 분석(lexical analysis)을 용이하게 할 수 있고, 어셈블리 코드를 생성하는 데 있어서는 매크로 어셈블러 형식을 이용하여 각 명령어에 해당하는 어셈블 코드로 대치한다. 이러한 번역 과정에서 원시 코드에서의 오류의 개입을 사용자에게 보고하는 것이 중요하고[9], 또 하나의 중요한 과정은 기계어로 변환된 코드의 효율성을 증가시키는 코드 최적화(code optimization)이다[10].

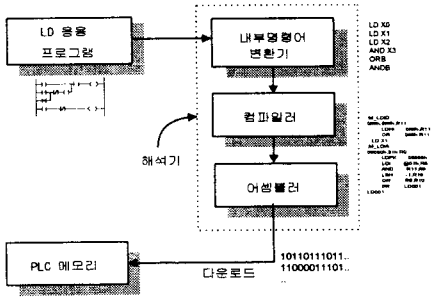

그림 2. LD 해석기의 구조

다시 이것을 어셈블러 프로그램이 프로그램형 논리 제어기의 프로세서에서 곧바로 실행할 수 있는 기계어로 변환시키는데, PLC 프로그램은 대체로 각 명령어 별로 독립적인 관계를 가지므로 각 명령어 블록 간에 상관 관계 있는 것들(라벨, 변수...)을 먼저 처리한 후에 명령어 단위로 끊어서 차례로 어셈블러를 통과시키면 메모리 절감과 해석 속도 면에서 향상된 성능을 볼 수 있다.

로더의 최종 출력인 실행 가능한 이진 코드(binary code)는 프로그램형 논리 제어기가 플랜트의 제어 동작을 하고 있지 않은 오프 라인 시에 그것의 메모리로 모두 전송된다. 이를 실행시키면 프로그램형 논리 제어기는 이 코드를 순차적으로 실행하여 플랜트 제어를 하게 된다. 여기서 로더 내부의 명령어 해석 프로그램의 입력은 LD로 이루어진 프로그램이며 출력은 동일한 작동을 하는 프로그램형 논리 제어기의 프로세서에서 작동하는 실행 가능한 이진 코드이다.

3. 시스템 구현과 평가

3.1 시스템의 구현

프로그램형 논리 제어기를 구현하였으며, 프로세서로 디지털 시그널 프로세서(DSP:Digital Signal Processor) TMS320C40-40을 사용하는 제어기 시스템을 제작하였다.

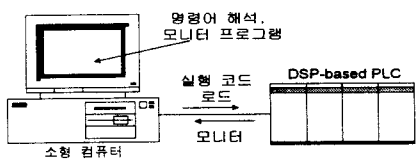

그림 3. 테스트 베드 시스템

3.2 내부 명령어의 실행 속도

해석 프로그램의 제어 응용 프로그램 해석 속도를 비교하여 보면 기존의 해석 방식이 제안된 해석 방식에 비해 그 해석 속도가 빠르다. 왜냐하면 제안된 해석 방식은 컴파일 과정과 어셈블 과정을 거칠 뿐만 아니라 파일 입출력을 여러 번 하기 때문에 미리 명령어에 대한 변환

루틴이 준비되어 있는 기존의 해석 방식에 비해 그 속도는 현저히 느리다. 그러나, 이 방식의 프로그램은 제어를 위한 응용 프로그램을 실행하고 있을 때 해석하는 것이 아니라 실행 전에 이미 그 해석을 모두 마치고 실행 가능한 결과 코드를 제어기에 넘겨주기 때문으로, 그 해석에 걸리는 시간이 PLC의 동작시의 성능에는 관련이 없다. 따라서, 두 해석 방식의 비교를 위해서는 해석에 지연되는 시간이 아닌 제어 동작 시, 즉 실행 시간의 비교를 행하는 것이 효과적인 비교이다.

실제 프로그램형 논리 제어기의 동작 시에 실행 시간의 감소 효과를 분석하기 위해서는 동작 중인 응용 프로그램에 들어 있는 내부 명령어들의 분포를 이용하여 계산되어야 한다. 또한 전술한 바와 같이 동 종류별 명령어들은 그 실행 시간이 비슷하여 평균 실행 시간으로 대표될 수 있기 때문에 각 종류별 명령어의 사용 빈도를 이용하는 것으로 전체 실행 시간에 큰 오차를 주지는 않는다. 여러 개의 실제 공정에서 사용 중인 프로그램들을 분석한 내부 명령어 의 분포도는 표 2와 같으며, 이를 바탕으로 전체 프로그램 내의 명령어 개수, 기본 명령어의 분포 등을 변화시키면서 응용 프로그램의 실행 시간을 분석하였다.

다음의 식 (1)은 실험의 결과인 기본 명령어의 평균 실행 시간인 $T_{기본명령}$, 기본 명령 이외의 명령어의 종류별 평균 실행 시간인 $T_{응용명령}$과 그림 9의 명령어 빈도 $P_{기본명령}$, $P_{응용명령}$을 이용하여 프로그램 내의 명령어의 전체 개수 N_{exe}를 가정하였을 때의 프로그램 실행 시간을 계산한 것이다. 기본 명령어의 비율 $P_{기본명령}$은 프로그램에 따라 변할 수 있다고 가정하였다.

$$\text{프로그램 평균 실행 시간} = T_{기본명령} \times P_{기본명령} \times N_{exe} + \sum_{응용명령} \left\{ T_{응용명령} \times \frac{P_{응용명령}}{(1-0.689)} \times (1-P_{기본명령}) \times N_{exe} \right\} \quad (1)$$

그림 4는 실행되는 응용 프로그램 내의 명령어의 수, Nexe, 즉 프로그램 크기를 300씩 증가시키면서 프로그램의 크기를 변화시키면서 실행 코드 내의 기본 명령어의 비율과 프로그램 실행 시간과의 관계를 나타낸 것이다.

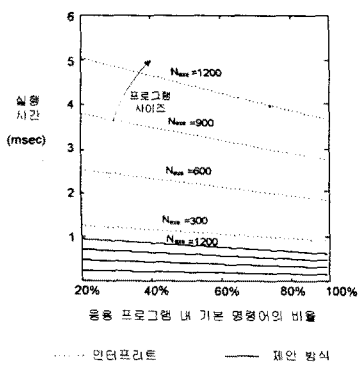

그림 4. 응용 프로그램의 실행 시간 변화

전체적으로 보아 제안된 해석 방식의 적용 시 실행 속도의 상승 효과가 두드러짐을 알 수 있으며, 전체적으로 보아 실행 프로그램의 크기가 커질수록 두 방식의 실행 시간의 차이는 더 커지는 것을 알 수 있다. 또한, 응용 프로그램 내에서 기본 명령어를 더 많이 사용할수록 그 실행 시간은 감소하고 이 감소폭은 그 프로그램의 크기가 커질수록 더 증가한다는 것을 그림을 통하여 알 수 있다.

4. 결 론

본 논문에서는 고속을 요구하는 프로그램형 논리 제어기에서 발생하는 프로그램 수행 속도 향상에 대한 문제를 고찰하여 제어 프로그램의 수행 속도를 빠르게 하기 위한 새로운 해석 구조와 실행 방식 등을 제안하고 이를 구현하고 실험하여 기존의 방법과의 비교를 통하여 성능을 평가하였다.

제안된 해석 방법은 래더 응용 프로그램으로부터 중간 코드를 만들지 않고 순차적으로 그대로 실행이 가능하도록, 즉 제어기의 운영 체제 내부에 해석부가 내재되지 않도록, 소형 컴퓨터 등의 오프라인 상에서 결과 코드를 생성해 내어 제어기에 통신을 통하여 로딩하여 실행시키는 방식이다. 각 방식의 실행 시간을 측정, 비교하기 위하여 프로그램형 논리 제어기를 제작하였는데, 제안된 해석 방식이 기존의 방식에 비해 그 평균 프로그램 실행 시간이 적은 것으로 실험으로 관측되었다. 이것은 호출, 회귀가 거의 없고 기존 방식과 같이 중간 코드 억세스를 위하여 메모리를 참조하지 않기 때문에 산출된 결과로 분석할 수 있다.

프로그램형 논리 제어기의 역할이 확대되어 고속을 요구하고 복잡한 연산의 필요성이 대두되며 내부 명령어의 수가 증가하여 프로그램의 실행 시간과 메모리 사용량이 늘어날 수 있기 때문에 효율적인 해석 방식의 연구는 앞으로 더더욱 중요한 과제가 될 것이다

[참 고 문 헌]

[1] I. Warnock, *Programmable Controllers - Operation and application*, Prentice Hall, 1988

[2] International Electrotechnical Commission, *Programmable Controllers - Part 3 : Programming language*, IEC Publication1131-3, 1993

[3] J. Park, N. Chang, G .S. Rho, and W. H. Kwon, "Implementation of a Parallel Algorithm for Event Driven Programmable Controllers," *Control Eng. Practice*, vol. 1, no. 4, pp. 663-670, 1993

[4] Jong-il Kim, J. Park, and W. H. Kwon, "Architecture of a ladder solving processor for programmable controllers", *Microprocessors and Microsystems*, vol. 16, no. 7, pp. 369-379, 1992

[5] G. Rho, J. Park, and W. H. Kwon, "Load Balancing of a Data-flow Based Programmable Controllers," *Proc. of IECON 93, Hawaii, U.S.A*, 1993.

[6] K. Koo, G. Rho, J. Park, and W.H. Kwon, and N. Chang, "Architectural Design of a RISC Processor for Programmable Logic Controller," *Journal of Systems Architecture*, Elsevier, pp. 311-325, No. 44, 1998

[7] G. Rho, K. Koo, N. Chang, J. Park, and W.H. Kwon, "Implementation of a RISC Processor for Programmable Controllers," *Microprocessors and Microsystems*, 1994.

[8] Y. Shimokawa, T. Matsushita, H. Furuno, and Y. Shimanuki, "A High-Performance VLSI chip for Instrumentation and Electric Control," *Proc. of IECON 91*, pp.884-889, 1991

[9] R.W. Lewis, *Programming Industrial Control Systems using IEC 1131-3*

[10] Alfred V.Aho, Ravi Sethi, Jeffrey D.Ullman, *Compilers*, Addison-Wesley Publishing Company,1986

[11] Peter Calingaert, *Assemblers, Compilers, and Program Translation*, Computer Science Press., 1979