제 19 회 한국정보처리학회 춘계학술발표대회 논문집 제 10 권 제 1 호 (2003. 5)

가상기계를 위한 어셈블리 언어

남동근°, 윤성림, 오세만
동국대학교 컴퓨터공학과
E-Mail : {capscrom, yslhappy, smoh}@dongguk.edu

Assembly Language for Virtual Machine

Dong-Keun Nam°, Sung-Lim Yun, Se-Man Oh
Dept. of Computer Engineering, Dongguk University

요 약

가상기계란 하드웨어로 이루어진 물리적 시스템과는 달리 소프트웨어로 제작되어 논리적인 시스템 구성을 갖는 개념적인 프로세서이다. 가상기계 기술은 기계의 프로세서나 운영체제가 바뀌더라도 응용프로그램을 변경하지 않고 사용할 수 있다는 장점이 있다. 최근에는 GVM, KVM 등 모바일 단말기를 위한 가상기계들이 개발되면서 그 중요성이 더욱 부각되고 있으며 특히, 임베디드 시스템을 위한 가상기계 기술은 모바일 디바이스와 디지털 TV 등의 다운로드 솔루션에 꼭 필요한 소프트웨어 기술이다.

본 논문에서는 바이트코드, MSIL 등 기존의 가상기계를 위한 어셈블리 언어들의 분석을 기반으로 하여 임베디드 시스템을 위한 가상기계의 표준 중간 언어인 SIL(Standard Intermediate Language)을 제안하고 니모닉(Mnemonic)을 정의한다. SIL 은 SIL Assembler 에 의해 EVM 의 실행 파일인 *.evm 의 형태로 번역되며 객체지향 프로그래밍 언어와 순차적인 프로그래밍 언어를 모두 수용할 수 있는 특징을 지닌다.

1. 서론

가상기계란 하드웨어로 이루어진 물리적 시스템과는 달리 소프트웨어로 제작되어 논리적인 시스템 구성을 갖는 개념적인 컴퓨터이다. 가상기계 기술을 이용하면 응용프로그램 실행 환경인 프로세서나 운영체제가 바뀌더라도 응용프로그램을 수정하지 하지 않고 사용할 수 있다. 대표적인 가상기계로서 자바 바이트코드를 실행하는 JVM(Java Virtual Machine)이 있다.

최근에는 GVM, KVM 등 모바일 단말기를 위한 가상기계들이 개발되면서 그 중요성이 더욱 부각되고 있다. 특히, 임베디드 시스템을 위한 가상기계 기술은 모바일 디바이스와 디지털TV 등에 탑재할 수 있는 핵심기술로서 다운로드 솔루션에서는 꼭 필요한 소프트웨어 기술이다.

이러한 가상기계는 실제 물리적 시스템과 같이 고유의 어셈블리 언어를 갖고 있다. 몇몇 가상기계의 어셈블리 언어를 살펴보면, JVM의 바이트코드, GVM의 SAL코드, .NET의 MSIL, EVM의 SIL 등이 있다.

현재 Microsoft의 C#언어와 SUN사의 자바언어 등을 모두 수용할 수 있는 가상기계에 대한 연구가 진행중이다. EVM(Embedded Virtual Machine)이라 명명되어진 이 가상기계 솔루션은 C#, 자바 등 객체지향 언어 뿐만 아니라 C언어와 같이 순차적인 언어로 작성된 프로그램을 어셈블리 언어 형태인 *.sil을 거쳐서 *.evm 파일 형태로 변환하여 임베디드 시스템에 탑재된 가상기계에서 실행할 수 있도록 한다.

이와 같은 프로젝트의 일환으로 본 논문에서는 바이트코드, MSIL 등 기존의 가상기계를 위한 어셈블리 언어들의 분석을 기반으로 하여 임베디드 시스템을 위한 가상기계의 표준 중간 언어인 SIL(Standard Intermediate Language)을 제안하고 니모닉을 정의한다.

본 논문의 2장에서는 기존의 가상기계 어셈블리 언어와 EVM에 대한 관련연구를 소개한다. 3장에서는 가상기계의 어셈블리 언어의 특징에 대해서 기술하고 4장에서는 SIL을 설계한다. 마지막으로 5장에서는 본 연구의 결론과 향후 연구 과제에 대해서 기술한다.

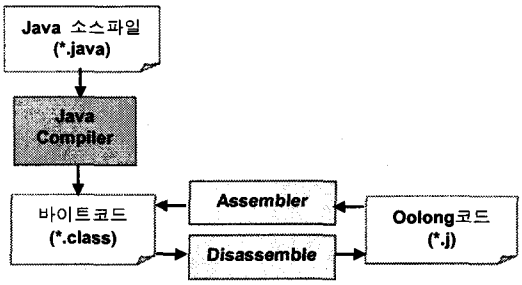
본 연구는 한국과학재단 목적기초연구(R01-2002-000-00041-0)지원으로 수행되었음.

2. 관련연구

2.1 Oolong

Oolong은 JVM의 어셈블리 언어에 해당된다. JVM은 자바 소스 프로그램을 컴파일한 클래스 파일을 실행한다. 그러나 클래스 파일 형식은 프로그래머가 분석하거나 수정하기가 매우 힘들기 때문에 Oolong언어가 만들어졌다. Oolong은 클래스 파일 형식과 거의 동일하지만 클래스 파일에 비해서 읽고 쓰기가 훨씬 쉽다.

Oolong은 John Meyer의 Jasmin언어를 기반으로 만들어졌으며 Jasmin의 니모닉을 그대로 사용하고 있다. Oolong으로 작성된 프로그램은 Oolong 어셈블러를 통해 클래스 파일로 어셈블되어 JVM에서 실행된다. Gnoloo 디스어셈블러를 사용하면 클래스 파일로부터 Oolong 구문을 추출해 낼 수 있다. Oolong 파일의 확장자는 관습적으로 *.j를 사용한다. [그림1]은 Oolong 코드의 추출과정을 나타내고 있다.



[그림1] Oolong코드의 추출 과정

Oolong의 의사코드는 15개로 구성되어 있으며 '.'으로 시작한다[표1].

[표1] Oolong의 의사코드

지시자	의미
.source	클래스 파일에 Sourcefile이라는 속성을 생성
.class	생성되는 클래스 이름과 접근 수정자를 정의
.interface	인터페이스를 선언
.end class	클래스의 끝을 나타내는데 사용되는 지시자
.super	해당 클래스의 수퍼 클래스를 선언
.implements	해당 클래스가 구현하는 인터페이스 명을 선언
.field	클래스의 필드를 선언
.method	클래스의 메소드를 선언
.limit locals	해당 메소드의 최대 지역변수의 개수를 제한
.limit stack	해당 메소드의 최대 스택의 개수를 제한
.line	소스코드에서의 라인번호를 선언
.var	디버깅을 위해 각 번호에 해당하는 지역 변수에 이름을 할당
.throws	해당 메소드에서 발생하는 예외를 선언
.catch	해당 메소드에서 예외 검사를 위한 구역을 설정
.end method	메소드의 끝

하나의 지시자는 한 줄을 정의하며 다만 .class 지시자와 .end class 지시자, .method 지시자와 .end method 지시자는 각각 지시자 사이의 부분을 정의 한다. ';'이후의 구문은 주석으로 처리된다.

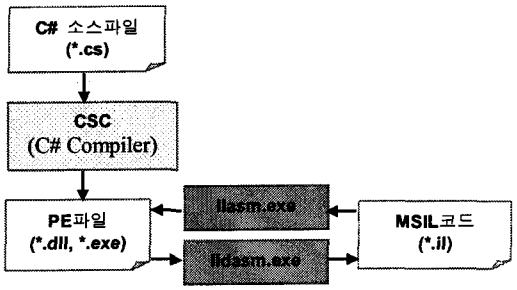
Oolong의 연산코드는 228개이며 Jasmin언어의 니모닉을 사용하고 있다. 각각의 니모닉은 클래스 파일의 opcode와 대응된다. 니모닉에는 인자가 따라올 수 있으며 인자들의 종류와 개수는 니모닉에 따라 다르다.

2.2 MSIL(Microsoft Intermediate Language)

MSIL은 .NET에서 사용되는 중간 언어이며, 컴파일러나 다른 도구에 의해 원시코드로부터 쉽게 생성될 수 있도록 설계된 스택 기반의 명령어 집합이다.

MSIL은 하드웨어 및 플랫폼에 독립적이고, JIT(Just-in-time)컴파일러를 목적으로 설계 되었다. Oolong과는 다르게 처음부터 언어독립적으로 설계되어 포괄적인 프로그래밍(Generic Programming)을 목적으로 하기 때문에 프로그램의 기능 및 구조의 변화에 잘 적응하는 언어이다.

원시코드가 CSC(C# Compiler)를 통해 *.dll이나 *.exe 형태로 번역되면 .NET에서 제공하는 Ildasm 디스어셈블러를 이용하여 MSIL코드를 추출할 수 있다. 반대로 Ilasm 어셈블러를 이용하여 MSIL코드를 PE(Portable/Executable)파일 형태로 변환할 수 있다. [그림2]는 이와 같은 과정을 나타내고 있다.



[그림2] MSIL코드의 추출과정

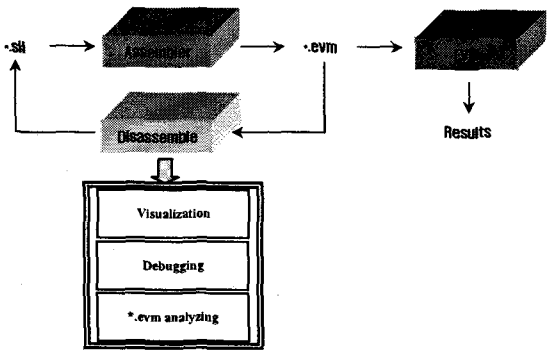
MSIL의 연산코드는 236개이다. 연산코드의 종류에는 산술/논리 연산, 흐름제어, 예외처리, 메소드 호출 등이 있고, 객체지향 프로그래밍 구조에 영향을 주는 가상 메소드 호출, 필드 접근, 배열 접근, 객체 할당과 초기화 등이 있다. MSIL 명령어 집합은 스택상에서 데이터 타입들을 탐색하고, JIT 컴파일러를 통해 실행된다.

2.3 EVM

EVM은 모바일 디바이스, 셋톱 박스, 디지털 TV 등에 탑재되어 동적 응용프로그램을 다운로드하여 실행할 수 있는 가상기계 솔루션이다.

EVM은 크게 세 부분으로 나뉘어진다. 첫째, C#이나 자바 등의 고급 프로그래밍언어로 작성된 프로그램을 본 논문에서 제안하는 SIL 형태로 번역하는 부분이다.

두번째는 SIL코드를 가상기계에서 실행가능 한 형태인 *.evm 형태로 변환하는 어셈블러 부분이다. 마지막 세번째는 실제 하드웨어에 탑재되어 *.evm파일을 실행하는 가상기계 부분이다. 이와 같은 EVM의 구현 모델 중 SIL 어셈블러 부분을 [그림3]에 나타내었다.



[그림3] SIL 어셈블러 구현 모델

3. 가상기계의 어셈블리 언어

어셈블리 언어란 프로그램을 기술하는 언어의 하나로서 사람이 판독하기 어려운 형태의 기계어를 알파벳, 정수 등 인간이 판독하기 쉬운 기호 형식으로 나타낸 것이다. 이러한 기호를 니모닉이라고 한다.

3.1 Real Machine의 어셈블리 언어

Real Machine의 어셈블리 언어는 기계 명령(machine instruction)에 대응하는 저급 언어이다. 각 명령문은 통상적으로 단일 기계 명령과 1:1로 대응한다. 어셈블리어로 프로그램을 작성한 후에는 어셈블러를 사용하여 기계어로 번역한다. Real Machine의 어셈블리 언어는 특정 프로세서의 기계어에 대응해서 디자인 되기 때문에 컴퓨터에 대한 정확한 제어를 제공한다. 또한 프로그래머가 프로세서, 메모리, 입 출력 등 하드웨어와 직접 상호작용 할 수 있기 때문에 고급 언어보다 빠른 속도를 제공한다.

그러나 프로세서마다 고유의 기계어와 어셈블리 언어를 가지고 있기 때문에 어셈블리 언어로 작성된 프로그램을 다른 프로세서에서 실행하기 위해서는 해당 프로세서의 어셈블리 언어로 재작성 해야 하는 단점이 있다. 또한 저급 언어이기 때문에 어셈블리 언어에 능숙하지 못한 사용자는 프로그램을 작성하기 어렵다는 단점이 있다.

3.2 Virtual Machine 의 어셈블리 언어

3.2.1 가상기계의 어셈블리 언어의 특징

가상기계의 어셈블리 언어는 프로그램 작성의 편의보다는 프로그램의 하드웨어 독립성 확보 및 단계적 컴파일을 통한 컴파일러 제작의 용이성 확보에 그 목적이 있다. 서로 호환이 되지않는 이기종이라도 가상기계를 탑재하면 프로그램의 재작성 이나 수정 없이 실행이 가능하다. 따라서 어떠한 프로그램이라도 가상기계의 어셈블리 언어로 번역이 되면 어셈블러를 통해 가상기계에서 실행 가능한 형태로 변환하여 하드웨어 플랫폼에 상관없이 실행할 수 있다.

객체지향 언어 등의 고급 언어를 가상기계의 어셈블리 언어로 번역한 후 다시 어셈블러를 통해 실행파일 형태로 변환 함으로써 컴파일러 제작 과정의 복잡도를 줄일 수 있다.

3.2.2 SIL의 특징

SIL은 EVM의 어셈블리 언어로서 프로그래밍 언어의 종류에 상관없이, 작성된 프로그램이 번역기를 통해 SIL로 번역되면 EVM의 실행 파일인 *.evm 형태로의 변환이 가능하다.

SIL은 다양한 프로그래밍 언어를 수용하기 위해서 바이트코드, MSIL등 기존의 가상기계 어셈블리 언어들의 분석을 토대로 정의 되었으며, 객체지향 언어와 순차적 언어를 모두 수용하기 위한 연산코드 집합을 갖고있다. 또한 어셈블리 언어 수준의 디버깅을 용이하게 하기위해 일관성 있는 이름 규칙을 적용하여 가독성 높은 연산코드를 정의하였다.

4. SIL 의 설계

EVM의 어셈블리 언어인 SIL은 기존의 가상기계 어셈블리 언어들의 표준화 모델로 설계하였다. SIL은 클래스 선언 등 특정 작업의 수행을 나타내는 의사코드와 실제 명령어에 대응되는 연산코드로 이루어져 있다.

C#, 자바 등의 고급 언어로 작성된 프로그램은 번역기를 통해 표준 중간 언어인 SIL로 번역되고 SIL은 어셈블러에 의해 *.evm 형태로 변환되어 시스템의 운영체제나 아키텍쳐에 상관없이 EVM에 의해 실행된다 [그림4].

[그림4] SIL 어셈블러

4.1 의사코드(Pseudo code)

의사코드는 '.'으로 시작하고 주석은 '//'으로 시작한다. 의사코드의 니모닉은 원시코드 수준의 키워드를 그대로 사용하여 가독성을 높였다. SIL의 의사코드는 총 21개이며 각 의사코드와 그 의미를 [표2]에 기술하였다.

[표2] SIL의 의사코드

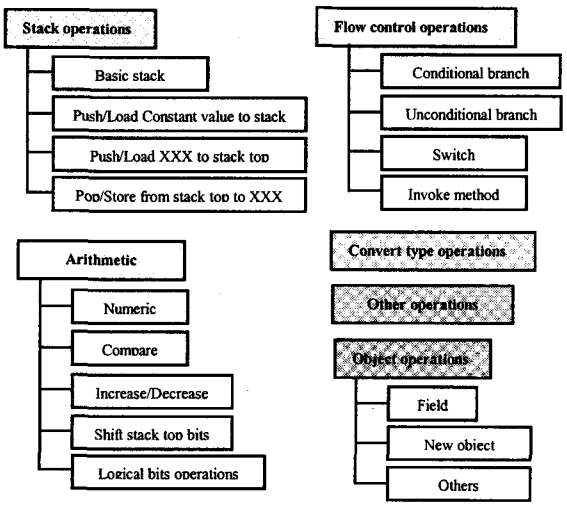
지시자	의미
.source	목적 파일에 Sourcefile이라는 속성을 생성
.namespace	네임 스페이스를 선언

.end namespace	네임 스페이스의 끝
.class	생성되는 클래스 이름과 접근 수정자를 정의
.interface	인터페이스를 선언
.end class	클래스의 끝을 나타내는데 사용되는 지시자
.super	해당 클래스의 수퍼 클래스를 선언
.implements	해당 클래스가 구현하는 인터페이스 명을 선언
.field	클래스의 필드를 선언
.method	클래스의 메소드를 선언
.limit locals	해당 메소드의 최대 지역변수의 개수를 제한
.limit stack	해당 메소드의 최대 스택의 개수를 제한
.line	소스코드에서의 라인번호를 선언
.var	디버깅을 위해 각 번호에 해당하는 지역 변수에 이름을 할당
.throws	해당 메소드에서 발생하는 예외를 선언
.catch	해당 메소드에서 예외 검사를 위한 구역을 설정
.end method	메소드의 끝
.event	이벤트 선언
.property	프로퍼티 선언
.set	set 프로퍼티 선언
.get	get 프로퍼티 선언

4.2 연산코드(Operation code)

연산코드는 크게 6개의 카테고리로 나눌 수 있으며 각각의 카테고리는 세부 카테고리를 갖는다.

Arithmetic 카테고리는 수학적 연산과 비교연산, 비트연산, 그리고 논리연산에 관련된 연산코드를 포함한다. Flow control operations 카테고리는 branch와 메소드 호출에 관한 연산코드를 포함한다. Stack operations 카테고리는 기본적인 스택 제어 연산코드와 저장장소에 따른 push/load 및 pop/store 연산코드를 포함한다. Convert type operations 카테고리는 스택 top에 있는 데이터의 타입을 변환하는 연산코드를 포함한다. Object operations 카테고리는 객체 생성과 필드에 관련된 연산코드를 포함한다. 그 이외의 연산코드는 Other operations 카테고리 안에 포함된다[그림5].



[그림5] SIL 연산코드의 카테고리

연산코드의 니모닉은 해당 연산코드가 수행하는 일을 의미하는 영어 단어를 기본으로 정의한다. 약자를 의미하는 알파벳과 정수의 조합을 원칙으로 하고, 연산코드의 실행과 관련하여 타입에 대한 정보가 필요한 경우' .' 을 사용하여 타입을 표현하는 캐릭터를 덧붙이도록 한다. 연산코드는 반드시 알파벳으로 시작해야 하며,' .' 을 제외한 어떠한 특수 기호도 사용할 수 없다.

연산코드는 연산에 필요한 인자를 가질 수 있고 인자의 개수는 연산코드에 따라 다르다. 연산코드의 다형성을 제공하기 위해서 Arithmetic 카테고리와 Convert type operations 카테고리의 연산코드는 피연산자의 타입정보를 피연산자가 가지도록 하였다.

코드 사이즈의 최적화를 위해서 작은 범위의 피연산자 값을 갖는 short form 형태의 연산코드를 정의하였다. 또한 일부 load 관련 연산코드와 매개변수 관련 연산코드에 대해서는 연산코드에 피연산자 정보를 포함한 pseudo operation code를 정의하였다.

5. 결론 및 향후연구

가상기계 기술은 응용프로그램의 이식성 확보라는 장점을 가진 기술로서 특히, 임베디드 시스템을 위한 가상기계 기술은 모바일 디바이스와 디지털 TV등에 탑재되는 다운로드 솔루션에 꼭 필요한 소프트웨어 기술이다.

본 논문에서는 기존의 가상기계를 위한 어셈블리 언어들의 분석을 기반으로 하여, 임베디드 시스템을 위한 가상기계의 표준 중간 언어인 SIL을 제안하고 그 니모닉을 정의하였다. SIL은 객체지향 언어와 순차적 언어 등 다양한 프로그래밍 언어를 수용할 수 있도록 설계되었으며, 일관성 있는 이름 규칙을 적용하여 가독성 높은 연산코드를 정의하였다. 따라서, 가상기계를 위한 어셈블리 언어의 표준으로서 그 역할을 충분히 할 수 있으리라 판단된다.

향후 과제로는 SIL 연산코드의 보완 및 SIL코드 형태의 *.sil파일을 EVM의 실행 파일인 *.evm형태로 변환하는 어셈블러에 관한 연구가 필요하겠다.

참고문헌

[1] C# Language Specification, Microsoft Corporation, Nov. 20. 2000.

[2] Jon Meyer & Troy Downing, Java Virtual Machine, O'REILLY, Mar. 1997.

[3] Joshua Engel, Programming for the Java Virtual Machine, Addison Wesley, 1, 1999.

[4] MSIL Instruction Set Specification, Microsoft Corporation, Nov. 20. 2000.

[5] Serge Lindin, INSIDE MICROSOFT .NET IL ASSEMBLER, Microsoft Corporation, Feb. 6. 2002.

[6] The IL Assembly Language Programmers' Reference, Microsoft Corporation , Oct. 10. 2000.

[7] 민정현, "매핑테이블을 이용한 자바 바이트코드에서 MSIL로의 번역기", 동국대학교 석사학위 논문, 2001.