차세대 마이크로프로세서를 위한 어셈블러의 객체화에 대한 연구

정 태 의[†] · 이 지 영[††] · 이 광 엽[†††] · 이 용 석[††††]

요 약

소프트웨어 위기에 대한 해결 방법으로 객체지향 방법론이 빠르게 보급되고 있다. 객체화된 시스템은 객체라는 독립적인 모듈들의 집합으로 이루어짐으로 해서 객체 단위의 재사용이 가능하며 수정시 수정 대상 범위가 독립된 객체로 제한되어 타 객체들에 대한 영향이 비교적 적다는 장점을 가지고 있다. 본 논문에서는 객체지향의 이러한 장점들을 이용하여 차세대 마이크로프로세서를 위한 매크로어셈블러의 객체화에 관한 연구 내용을 다루고자 한다. 새로운 마이크로프로세서가 등장할 때마다 우리는 어셈블러를 새로이 개발하거나 혹은 기존의 어셈블러를 수정해야 한다. 전자는 중복 개발에 따른 손실이 발생하고, 후자는 특히 기존의 어셈블러의 경우 수정시 발생하는 타 모듈들에 대한 영향을 분석해야 하며 이에 따른 비효율적 생산성 문제를 가지고 있다. 이러한 문제에 대한 대안으로 본 논문에서 제시한 객체지향 매크로어셈블러는 객체지향 방법론에 준하여 독립적인 객체들로 이루어져 있어 객체들의 재사용이 가능하며 또한 타 객체들에 대한 영향을 최소화함으로서 수정시 발생하는 비효율적 생산성 문제를 어느 정도 해결할 수 있다. 또한 본 논문에서 제시한 객체지향 매크로어셈블러는 컴파일링 속도의 향상을 위해 매크로 프리프로세서와 어셈블러를 통합하여 구성하였으며 입력파일에 대한 어휘분석기를 오토마타를 이용하여 설계하였다.

The Study of an Object-Oriented Macro Assembler for Next-Generation Microprocessors

Tae-Eui Jeong[†] · Jee-Young Lee[††] · Kwang-Youb Lee[†††] · Yong-Surk Lee[††††]

ABSTRACT

The object-oriented methods are being rapidly accepted as the solution for the software crisis. Object-oriented systems are composed of the integration of independent object modules; their merits are such that it is possible to reuse objects already developed, and that, when changes are required, modifications are restricted to some independent objects such that their affects to other objects are so little. In this paper, we deal with the macro assembler for next-generation microprocessors which has the merits of object methods. Whenever a microprocessor is newly developed, new assembler should be developed or the existing assembler be modified. In the former, it leads to the loss of time and money by the repeated developments, and, in the latter, it causes the problem of inefficient productivity since other modules are to be analyzed for the affections followed by modifications of one module, especially in the existing assemblers. To resolve

※ 본 논문은 한국과학재단 특정기초연구과제(과제번호 : 97-0100-0701-2)의 연구 결과물입니다.
† 종신회원 : 서경대학교 컴퓨터과학과 교수
†† 준 회 원 : 서경대학교 컴퓨터과학과
††† 정 회 원 : 서경대학교 컴퓨터공학과 교수
†††† 정 회 원 : 연세대학교 전자공학과 교수
논문접수 : 1998년 6월 15일, 심사완료 : 1998년 12월 31일

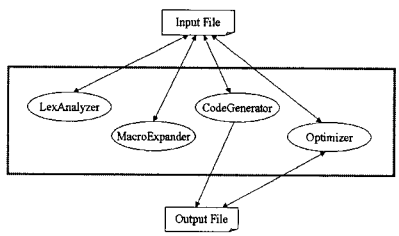
these problems, the object-oriented macro assembler suggested in this paper consists of independent objects separable such that it shows reusability and reduces the inefficient productivity by minimizing the affects to other objects. Moreover, the object-oriented macro assembler integrates a macro pre-processor into an assembler, and uses automata for analyzing input streams to reduce the compile time.

1. 서 론

현재 소프트웨어 공학의 최대 관심사는 소프트웨어 위기를 극복하고자 하는, 즉 소프트웨어 생산성을 어떻게 높일 수 있느냐에 초점을 맞추고 있다. 소프트웨어는 하드웨어의 발전에 비해 매우 속도가 느린 발전을 보여왔다. 60년대의 하드웨어 제 1세대는 일괄처리 기술을 개발하여 메인 프레임 컴퓨터의 활용도를 높였고, 70년대의 제 2세대에는 터미널을 통해 온라인으로 프로그램 수행을 가능케 하는 시분할 기술을 도입하여 컴퓨터의 활용도를 높였으며, 마치 개인용 컴퓨터를 사용하는 것과 같은 속도와 편리함을 사용자에게 제공하였고, 제 3세대인 80년대 초에는 개인용 컴퓨터의 대량 보급을 이루었다. 그리고 80년대 후반부터 지금까지의 하드웨어 제 4세대는 분산처리와 이종환경에서의 상호 연관 동작과 호환성을 제공하는 개방형 시스템의 개발이 활발히 진행 중에 있다. 이처럼 하드웨어는 뚜렷한 기술의 발전을 이루어 오면서 이를 충분히 지원할 수 있는 소프트웨어의 발전이 요구되어왔다. 그러나 소프트웨어의 발전의 속도는 하드웨어의 발전 속도에 미치지 못하고 있다. 소프트웨어의 생산성 부진문제를 해결하기 위해서는 소프트웨어를 재사용 가능한 독립적인 부분으로 생산하여 생산성을 높이고 수정에 의한 영향을 줄일 수 있는 방법이 필요하다[16]. 기존의 방법으로 개발된 소프트웨어는 사용될 시스템에 종속적이어서 변경된 시스템이나 다른 시스템에 대하여 재개발해야만 하는 중복 개발의 비효율성을 가지고 있다. 어셈블러 또한 마찬가지이다. 어셈블러는 마이크로프로세서 발전에 따른 새로운 어셈블러 요구에 대하여 중복 개발되거나 기존의 어셈블러를 수정하여 왔다. 중복 개발시 발생하는 비용의 손실은 막대하며, 수정할 경우 기존의 어셈블러는 수정에 따른 영향이 상대적으로 크기 때문에 비효율적인 생산성 문제를 안고 잇다. 이러한 소프트웨어 생산의 비효율성 문제를 해결하기 위해 선택한 방법이 객체지향 방법이다[14, 19]. 이는 객체 지향에서의 객체가 자신의 상태와 동작을 스스로 가지고 있어서 독립적인 소프트웨어 모듈로서 재사용이 가능하며 필요한 객체만의 개발과 활용이 가능하기 때문이다[6,17,22]. 이에 대하여 본 논문에서는 객체지향 매크로어셈블러를 제안함으로써, 이제까지 마이크로프로세서의 발전에 따라 중복 개발되어오던 어셈블러의 생산성 문제를 해결하고자 한다. 객체지향 매크로어셈블러는 본 연구와 관련하여 개발중인 차세대 마이크로프로세서를 기본으로 하여 설계되었으며, 차세대 마이크로프로세서는 펜티엄 프로세서에 디지털 신호 프로세서(Digital Signal Processor)가 결합되어 고속 연산까지도 가능한 마이크로프로세서를 의미한다. 본 논문은 다음과 같이 구성된다. 2장에서 객체지향 매크로어셈블러의 구조를 기능적 측면과 객체지향 설계적 측면으로 나누어 설명하고, 3장에서는 객체지향 매크로어셈블러의 전체적인 구조를 보인다. 그리고 4장에서 본 논문의 결론을 검토하고 향후 연구 방향에 대하여 기술한다.

2. 객체지향 매크로어셈블러의 구성

본 논문에서 제시하는 객체지향 매크로어셈블러는 기능적인 측면에서 어셈블러의 기능을 모두 가지면서 구조적으로는 객체지향 시스템의 구조를 갖는다. 어셈블리 파일을 입력으로 받아서 목적 코드를 출력하는 객체지향 매크로어셈블러는 매크로어셈블러의 기능을 위해서 내부에 (그림 1)과 같은 어휘분석부와 매크로 확장부, 코드생성부, 코드최적화부를 갖는다.



(그림 1) 객체지향 매크로 어셈블러의 구성도
(Fig. 1) Structure of Object-Oriented Macro Assembler

2.1 기능적 측면

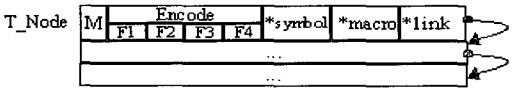
(그림 2)는 기능적인 면에서 일반적인 매크로어셈블러와 객체지향 매크로어셈블러의 기본 구조를 비교한 것이다. 일반적인 매크로어셈블러와 객체지향 매크로어셈블러의 가장 큰 차이는 매크로프로세서의 위치이다. 일반적인 매크로어셈블러는 매크로프로세서를 어셈블러의 전처리기(pre-processor)로 두어서 어셈블리 언어로 작성된 입력 파일에 대한 매크로 처리를 먼저 수행하며 이로 인해 확장된 어셈블리 파일을 만든 후, 후프로세서(post-processor)인 어셈블러는 이 파일을 입력으로 읽어서 컴파일 작업을 시작한다[1,3,9,10,13, 15,20]. 즉, 어셈블리 파일에 대한 두 번의 입력과 두 번의 분석 작업이 필요하다. 이에 반해, 객체지향 매크로어셈블러는 분석기에 매크로프로세서를 포함시켜 입력 파일을 토큰으로 분석하는 과정에서 매크로 처리까지 병행하여 수행하므로 한번의 파일 입력에 대해서 한번의 분석 작업이 필요하다. 객체지향 매크로어셈블러는 다음과 같이 어휘 분석부, 매크로 확장부, 코드생성부, 코드 최적화부의 4부분으로 구성되며 각각의 기능은 다음과 같다.

- 어휘 분석부 : 입력 파일을 분석하여 토큰 테이블을 작성한다.
- 매크로 확장부 : 매크로 호출시 매크로 정의에 따라 토큰 테이블을 확장한다.
- 코드 생성부 : 토큰 테이블을 기계어 코드로 인코딩 한다.
- 코드 최적화부 : 기계어 코드로 인코딩 된 출력에 대하여 핍홀 최적화기법을 적용하여 코드를 최적화 한다.

객체지향 매크로어셈블러는 어셈블 작업을 위해 다음과 같은 세가지 종류의 테이블 자료구조를 갖는다.

(1) 토큰 테이블

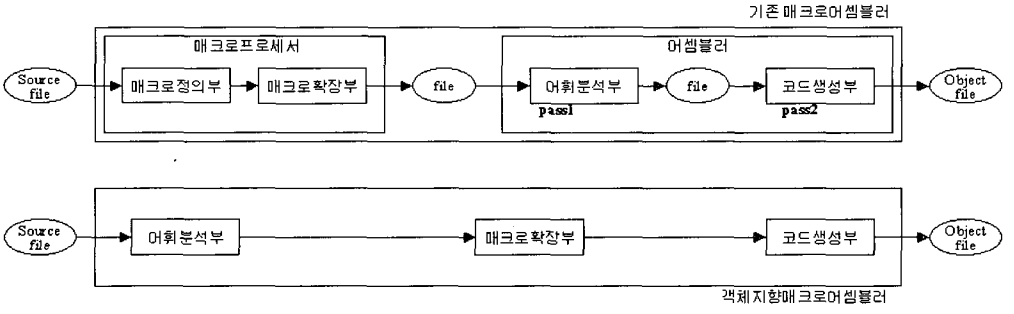
입력 파일에 대하여 어휘 분석이 끝난 토큰들에 대한 테이블로서 테이블의 각 튜플은 입력 파일의 한 라인에 대한 분석결과를 담고 있다. (그림 3)은 토큰 테이블의 형태를 보인 것으로 입력 파일의 한 라인에 대한 토큰 테이블의 튜플은 구조체 T_Node로구성된다. M은 매크로 호출에 대한 정보를 나타내며 매크로 호출이 있는 경우 TRUE 값으로 설정된다. Encode는 디지털 신호 프로세서의 인코딩 형식으로 최종 출력 파일로 출력될 부분이다[11]. 심볼에 대하여 *symbol을 통해 심볼 테이블과 연결되며, 매크로에 대한 매크로 정의 테이블과의 연결은 *macro에 의해 수행된다. 토큰테이블의 튜플(T_Node)들은 *link로 서로 연결된다.



(그림 3) 토큰 테이블
(Fig. 3) Token Table

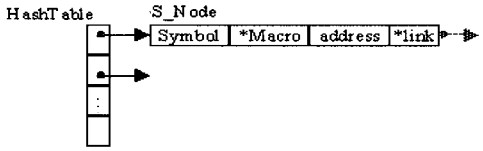
(2) 심볼 테이블

입력 파일에 나타나는 심볼(identifier)들을 등록하는 테이블로서, 인덱싱 방법으로 해쉬 테이블(Hash Table)을 사용한다[12]. 심볼 테이블의 각 노드는 (그림 4)에서와 같이 심볼을 저장하는 Symbol과 심볼의 주소에 대한 address, 그리고 심볼이 매크로 이름인 경우 매크로 정의 테이블과의 연결점인 *Macro로 구성된다. 심볼 테이블에 Macro 정의를 가리키는 포인터를



(그림 2) 객체지향 매크로 어셈블러와 기존 매크로 어셈블러의 기본구조
(Fig. 2) Basic Architectures of Object-Oriented Macro Assembler and the Existing Macro Assembler

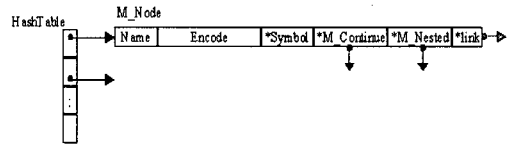
포함시킴으로써 매크로 이름 테이블을 따로 구성하지 않는다.



(그림 4) 심볼 테이블
(Fig. 4) Symbol Table

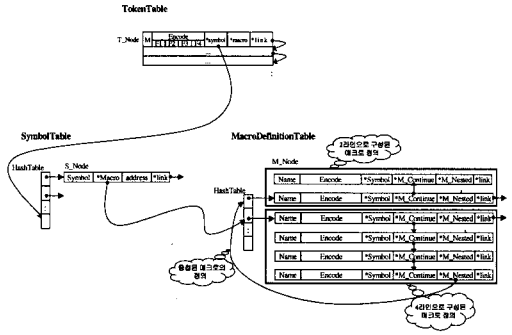
(3) 매크로 정의 테이블

매크로 확장을 위한 매크로 정의 내용을 가지고 있는 테이블로서 심볼 테이블에서와 같이 해쉬 테이블을 사용한다[12]. (그림 5)는 매크로 정의 테이블의 구성을 보여주고 있다. 매크로 정의 테이블에는 매크로 이름을 저장하는 Name, 매크로 정의를 라인별로 저장하는 Encode, 심볼에 대한 심볼 테이블 주소를 가지고 있는 *Symbol이 있다. 토큰 테이블과 마찬가지로 매크로 정의의 한 라인에 대하여 M_Node 하나가 작성되며, 매크로 정의의 다음 라인에 대한 다음 노드와는 M_Continue로 연결된다. M_Nested는 매크로 정의 안에 포함될 수 있는 중첩 매크로(nested macro) 정의로서 M_Node와 연결된다.



(그림 5) 매크로 정의 테이블
(Fig. 5) Macro Definition Table

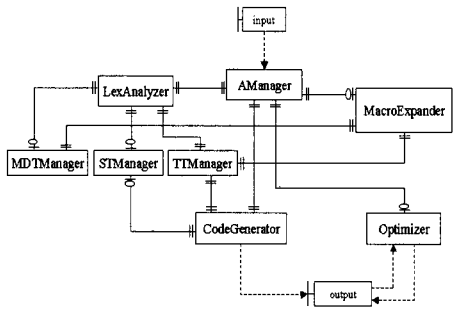
토큰 테이블, 심볼 테이블, 매크로 정의테이블은 (그림 6)과 같이 상호 동작한다. 입력파일의 어떤 라인에 심볼이 있으면, 그 라인에 대한 토큰테이블의 튜플이 심볼테이블과 연결된다. 입력파일의 어떤 라인에 매크로가 있으면, 그 라인에 대한 토큰테이블의 튜플이 매크로 이름이 있는 심볼테이블과 연결되고, 그 심볼테이블에서 매크로정의가 있는 매크로정의테이블과 연결된다. 매크로테이블에서 매크로 정의의 한 라인을 구성하는 M_Node들이 매크로 정의의 라인 수만큼 M_Continue로 연결되어 하나의 매크로 정의를 나타내며, 매크로 정의 내에 정의된 중첩 매크로는 M_Nested로 연결된다.



(그림 6) 객체지향 매크로 어셈블러의 테이블들의 상호 동작
(Fig. 6) Interactions among Tables in Object-Oriented Macro Assembler

2.2 객체지향 설계적 측면

객체지향 매크로어셈블러는 어셈블러의 기능을 수행하기 위해 어휘 분석부, 매크로 확장부, 코드 생성부, 코드 최적화부에 대하여 LexAnalyzer 객체와 MacroExpander 객체, CodeGenerator 객체, Optimizer 객체가 있으며 AManager 객체가 이들을 제어한다. 또한 객체지향 어셈블러에 필요한 3개의 테이블인 토큰 테이블, 심볼 테이블, 매크로 정의 테이블을 작성하고 관리하는 TTManager(Token Table Manager) 객체, STManager (Symbol Table Manager) 객체, MDTManager(Macro Definition Table Manager) 객체가 (그림 7) 및 <표 1> 에서와 같이 존재한다.



(그림 7) 객체지향 매크로 어셈블러의 객체 구조 다이어그램
(Fig. 7) Object Structure Diagram of Object-Oriented Macro Assembler

〈표 1〉 객체지향 매크로 어셈블러의 객체들
〈Table 1〉 Objects of Object-Oriented Macro Assembler

Control Object	Entity Object	Interface Object
AManager	LexAnalyzer	Input
	MacroExpander	Output
	CodeGenerator	
	Optimizer	
	TTManager	
	STManager	
	MDTManager	

AManager 객체가 외부와의 인터페이스를 제공하는 입력 인터페이스 객체로부터 처리를 요청하는 메시지를 받으면 객체지향 어셈블러는 작동을 시작한다. AManager 객체는 입력 파일을 분석하기 위해 LexAnalyzer 객체에게 analyze() 메시지를 보내어 어휘분석을 요청하고 LexAnalyzer 객체는 어휘 분석과 함께 매크로 정의 테이블을 작성한다. 매크로 호출시 AManager 객체는 MacroExpander 객체에게 expand() 메시지를 보냄으로서 매크로 정의에 따라 토큰 테이블을 확장한다. 또한 AManager 객체는 CodeGenerator 객체에게는 generate() 메시지를, Optimizer 객체에게는 optimize() 메시지를 각각 보냄으로서 전체 시스템을 제어한다.

2.2.1 어휘분석부

LexAnalyzer 객체와 함께 TTManager 객체, STManager 객체, MDTManager 객체들이 포함되어 어휘 분석부를 구성한다. AManager 객체로부터 어휘 분석 요청을 받은 LexAnalyzer 객체는 유한 오토마타를 사용하여 파일을 분석하고 분석된 토큰에 대한 토큰 테이블을 작성하도록 TTManager 객체에게 메시지를 보낸다. 분석된 심볼과 매크로 정의에 대해서는 각각 STManager 객체와 MDTManager 객체에게 메시지를 보내어 심볼 테이블과 매크로 정의 테이블을 작성하도록 한다. 각각의 테이블 작성과 함께 파일에 대한 어휘 분석이 끝나면, 매크로 호출에 대한 확장을 위해 AManager 객체가 MacroExpander 객체에게 매크로 확장 처리를 요청한다.

2.2.2 매크로 확장부

MacroExpander 객체와 TTManager 객체, MDTManager 객체가 매크로확장부를 구성한다. 요청을 받은 MacroExpander 객체는 토큰 테이블에서 M 필드를 검사하여 매크로를 호출한 튜플을 찾는다. 찾은 튜플의 Macro 필드는 어휘 분석부에서, 호출된 매크로에 대한 매크로 정의가 있는 매크로 정의 테이블의 주소로 설정되어 있으므로 매크로정의테이블에 대한 검색 작업 없이 바로 매크로 정의 테이블의 내용을 읽어서 매크로 확장을 처리한다. 토큰 테이블 전체에 대한 매크로 처리가 완료되면 AManager 객체가 CodeGenerator 객체에게 코드 생성을 요청한다.

2.2.3 코드 생성부

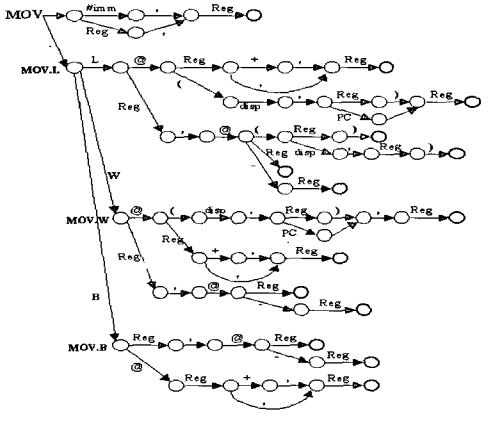
CodeGenerato 객체와 TTManager 객체, STManager 객체가 코드 생성부를 구성한다. 요청을 받은 CodeGenerator 객체는 TTManager 객체에게 요청하여 토큰 테이블을 순차적으로 읽으면서 인코딩 작업을 한다. 토큰 테이블의 Symbol 필드는 심볼에 대한 정보를 가지고 있는 심볼 테이블의 주소이므로 심볼 테이블에 대한 검색 작업 없이 바로 심볼 테이블에서 필요한 정보를 얻어 인코딩에 사용할 수 있다. 인코딩된 내용을 파일로 출력한다. 코드 생성이 끝나면 AManager 객체가 Optimizer 객체에게 코드의 최적화를 요청한다.

2.2.4 코드 최적화부

코드 최적화부는 Optimizer 객체로 구성한다. 코드의 최적화 요청을 받은 Optimizer 객체는 인코딩된 파일에 대하여 핍홀 최적화 기법을 사용하여 파일의 코드를 최적화한다.

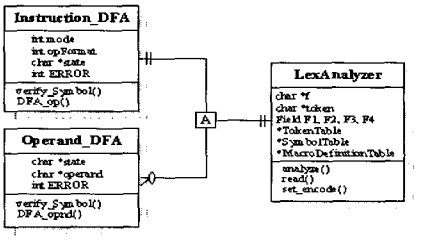
2.3 유한오토마타를 이용한 객체지향 매크로어셈블러의 어휘분석부

객체지향 매크로어셈블러에서는 번역속도를 향상시키기 위해 어휘분석부에서 유한오토마타를 사용한다[4, 21]. 유한오토마타의 각 상태에서 입력 문자에 따라 그에 대한 다음 상태로 전이가 이루어지면서 토큰이 인식된다. MOV 명령에 대한 전이도를 (그림 8)에서 보여주고 있다. 유한오토마타의 각 상태에서 결정 가능한 토큰 테이블의 필드에 대해서는 그 코드가 토큰 테이블의 필드에 할당된다. 유한오토마타를 사용하여 입력 파일 분석 단계에서 가능한 코드 할당 작업이 이루어지게 함으로써 번역 작업이 간소화되어 CodeGenerator 객체의 부담을 줄이고, 전체 객체지향 매크로어셈블러의 속도가 향상된다.



(그림 8) 'MOV'의 전이 다이어그램
(Fig. 8) Transition Diagram of 'MOV'

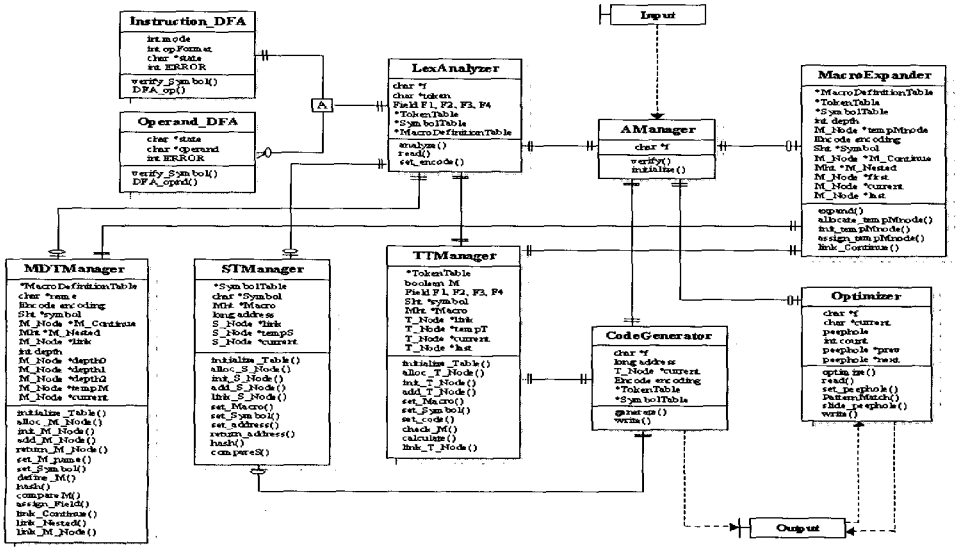
유한오토마타를 사용하여 설계한 어휘분석부를 객체 구조 다이어그램으로 나타내면 (그림 9)와 같다. LexAnalyzer 객체에 인코딩 형식을 나타내는 변수로 Field를 멤버변수로 두어 인코딩 형식에 대한 투명성을 LexAnalyzer 객체에서 제공할 수 있도록 한다. 어휘분석부의 유한 오토마타는 두 객체로 구성한다. 명령어에 대한 유한 오토마타 처리부분인 Instruction_DFA 객체와 오퍼랜드에 대한 유한 오토마타 처리 부분인 Operand_DFA 객체가 그것이다. 어휘분석부를 작은 객체들로 나누어 구성함으로써 이해와 구현이 용이해진다[2]. 즉 각 객체들이 사용자를 위한 단위 기능을 수행할 수 있도록 하며 객체의 멤버함수를 통하여 명확한 외부 인터페이스를 정의하여 독립적인 프로그램 모듈의 역할을 하도록 한다[7]. 또한 객체지향 방법의 캡슐화와 정보은폐의 장점을 제공한다. 따라서 변경사항에 대하여 객체의 독립적이고 제한적인 수정이 가능하다.



(그림 9) LexAnalyzer의 객체 구조 다이어그램
(Fig. 9) Object Structure Diagram of LexAnalyzer

3. 객체지향 매크로어셈블러의 전체적인 구조

2장에서 설계한 독립적인 클래스들을 결합하여 객체지향 매크로어셈블러를 구성한다. (그림 6)의 간략한 객체 구조 다이어그램에 매크로어셈블러의 기능을 수행하기 위한 변수와 함수들을 추가한 상세 객체 구조 다이어그램이 (그림 10)이다. 객체지향 매크로어셈블러



(그림 10) 객체지향 매크로 어셈블러의 상세화된 객체 구조 다이어그램
(Fig. 10) Detailed Object Structure Diagram of Object-Oriented Macro Assembler

는 외부와의 인터페이스로 Input과 Output을 가진다. 즉, 객체지향 매크로어셈블러는 하나의 큰 캡슐로서 Input과 Output을 통해서 외부와의 메시지 전달이 이루어진다. 객체지향 매크로어셈블러의 내부를 보면 메시지를 통해 통신하는 작은 객체들로 각각 캡슐화되어 있는데, 이들 객체에 대한 수정작업은 독립된 객체에 대한 수정의 용이성을 제공한다[5,8,18,22]. 예를 들어 기계어 형식이 변경될 경우, LexAnalyzer 객체의 Field를 수정한다. Field가 LexAnalyzer 객체의 멤버변수이므로 내부정보가 은폐되어 그 변경 사항이 주변에 영향을 주지 않으므로 다른 객체는 수정할 필요가 없다. 명령어가 변경될 경우에는 Instruction_DFA 객체를 수정한다. Instruction_DFA 객체는 LexAnalyzer 객체의 DFA_op() 멤버함수를 통한 명확한 외부 인터페이스를 제공하기 때문에 독립적인 모듈의 역할을 하도록 해준다.

그리고 데이터나 레지스터가 변경될 경우에는 Operand_DFA 객체를 수정한다. Operand_DFA 객체도 Instruction_DFA 객체와 마찬가지로 독립적인 모듈의 역할을 하도록 구성함으로써 기능적 추상화를 제공한다.

4. 결론 및 향후 연구 방향

객체지향 방법의 장점은 요구사항이 변경되어 프로그램이 변경되어야 할 경우, 객체를 재사용하거나 독립된 객체로 수정의 범위를 제한함으로서 수정에 따른 영향을 최소화할 수 있다는 것이다. 이러한 객체지향 방법의 장점을 이용하면 중복 개발로 인한 비효율적 생산성의 소프트웨어 문제를 해결할 수가 있다. 즉 한번 작성한 어셈블러에 대하여 어셈블러를 구성하는 내부 객체들 중에서 일부 객체는 그대로 다시 사용하고, 수정이 필요한 객체들에 대해서는 독립된 객체 내의 국한된 수정으로 새로운 요구사항에 맞도록 변경할 수가 있는 것이다. 수정의 범위를 최소화하기 위한 방법으로 기존의 방식에서는 모듈화 방법을 사용하는데[2], 이 방법은 객체지향 방법과 달리 모듈내의 데이터와 그 데이터를 다루는 오퍼레이션이 다른 모듈로부터 독립되어 있지 않고 영향을 받기 때문에 한 부분을 수정하더라도 그 부분을 다시 테스트해야 할뿐만 아니라 프로그램 전체에 미칠지 모르는 영향분석 및 검증을 해야만 한다. 그러나 객체지향 방법에서는 각 객체가 데이터와 그 데이터를 운영하는 오퍼레이션이 독립적이고 각 객체에 대한 검증을 마친 상태이기 때문에 수정 한 객체에 대해서만 검증이 필요하다. 따라서 본 논문에서 제안하는 객체지향 매크로어셈블러는 마이크로프로세서를 위해 어셈블러를 객체들로 구성함으로써 기존의 어셈블러보다 더 높은 재사용성을 제공하고 수정에 따른 영향을 최소화한다. 또한 매크로프로세서와 어셈블러를 1패스로 구성하여 어휘 분석중 매크로 정의 테이블을 작성토록 하여 한번만 입력 파일을 분석하도록 하였으며, 어휘분석기를 결정적 유한오토마타로 구현함으로써 컴파일링 속도를 높였다. 향후 연구과제로는 객체지향 매크로어셈블러에 대한 성능 평가 및 효율성을 더욱 높이기 위해 마이크로프로세서의 변경 요구에 따라 객체지향 매크로어셈블러를 자동으로 생성할 수 있는 객체지향 매크로어셈블러 자동생성기에 대한 연구가 있다.

참 고 문 헌

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, *Compilers Principles, Techniques, and Tools*, Addison-Wessley, 1988.

[2] Andrew W. Appel, *Modern compiler implementation in ML*, Cambridge, 1997.

[3] Barkakati and Hyde, *The Waite Group's Microsoft Macro Assembler Bible*, 2nd edition, 1996.

[4] Seth Bergmann, *Compiler Design : Theory, Tools, and Examples*, Wm. C. Brown Publishers, 1994.

[5] Grady Booch, *Object-Oriented Analysis and Design with Applications*, Benjamin/Cummings, 1994.

[6] Grady Booch, *Object Oriented Design with Application*, Benjamin/Cummings, 1991.

[7] Timothy Buddm, *Object-Oriented Programming*, Addison-Wesley, 1997.

[8] P. Cord and Edward Yourdon, *Object-Oriented Analysis*, Yourdon Press, 1991.

[9] John J. Donovan, *Systems Programming*, MaGraw-Hill, 1972.

[10] Douglas V. Hall, *Microprocessors and Interfacing : Programming and Hardware*, McGraw-Hill, 1992.

[11] HITACHI, *SH-DSP SH7410 Programming*

Manual, 1997.

[12] Ellis Horowitz, Sartaj Sahni, and Dinesh Mehta, *Fundamentals of Data Structures in C++*, Computer Science Press, 1995.

[13] Adi J. Khambata, Microprocessors / Microcomputers : Architecture, Software, and Systems, John Wiley & Sons, 1987.

[14] Won Kim and Frederick H. Lochovsky, *Object-Oriented Concepts, Databases and Applications*, Addison-Wesley, 1989.

[15] Thomas W. Parsons, *Introduction to Compiler Construction*, W. H. Freeman and Company, 1992.

[16] Roger S. Pressman, *Software Engineering : A Practitioner's Approach*, McGraw-Hill, 1997.

[17] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

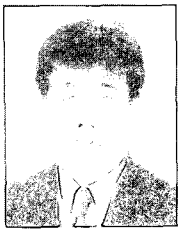
[18] S. Shlaer and S. Mellor, *Object-Oriented System Analysis-Modeling the World in Data*, Yourdon Press, 1988.

[19] David A. Taylor, *Object-Oriented Technology*, Addison-Wesley, 1990.

[20] Walter A. Triebel and Avtar Singh, *The 8088 and 8086 Microprocessors Programming, Interfacing, Software, Hardware, and Applications*, Prentice Hall, 1991.

[21] Reinhard Wilhelm, *Compiler Design*, Addison-Wesley, 1995.

[22] Edward Yourdon, Katharine Whitehead, Jim Thomann, Karin Oppel, and Peter Nevermann, *Mainstream Objects : an Analysis and Design Approach for Busines*, Pretice Hall, 1995.



정 태 의

e-mail : tejeong@bukak.seokyeong.ac.kr

1979년 2월 고려대학교 전자공학과졸업(학사)

1982년 6월 오하이오주립대 전기공학과 졸업(석사)

1989년 12월 오클라호마대학 전산학과 졸업(석사)

1994년 5월 오클라호마대학 전산학과 졸업(박사)

1983년 10월~1986년 6월 금성반도체 연구소 컴퓨터부문 선임연구원

1986년 7월~1987년 8월 United Microtek, Inc. (San Jose, CA) Engineer-ing Manager

1995년 3월~현재 서경대학교 컴퓨터과학과 조교수

관심분야 : 형식언어, 그래프알고리즘, 컴파일러, 객체지향방법론

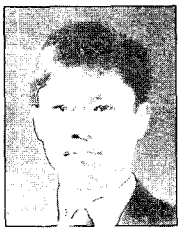
이 지 영

e-mail : jylee@bukak.seokyeong.ac.kr

1997년 2월 서경대학교 컴퓨터과학과 졸업(학사)

1999년 2월 서경대학교 컴퓨터과학과 졸업(석사)

관심분야 : 컴파일러, 객체지향방법론



이 광 엽

e-mail : kylee@bukak.seokyeong.ac.kr

1985년 8월 서강대학교 전자공학과 졸업(학사)

1987년 8월 연세대학원 전자공학과 졸업(석사)

1994년 2월 연세대학원 전자공학과 졸업(박사)

1989년 1월~1995년 2월 현대전자산업(주) 시스템IC 연구소 선임연구원

1995년 3월~현재 서경대학교 컴퓨터공학과 조교수

관심분야 : 마이크로프로세서, 캐쉬 및 컴퓨터 하드웨어 설계

이 용 석

e-mail : yonglee@bubble.yonsei.ac.kr

1973년 2월 연세대학교 전기공학과졸업(학사)

1977년 2월 University of Michigan, Ann Arbor 전자공학과 졸업(석사)

1981년 2월 University of Michigan, Ann Arbor 전자공학과 졸업(박사)

1982년 3월~1993년 2월 인텔(San Jose)등에서 연구

1993년 3월~현재 연세대학교 전자공학과 교수

관심분야 : 마이크로프로세서, SRAM, 캐쉬 및 DSP 설계