GCC 생성기를 이용한 C 컴파일러 개발

정삼진*
*백석대학교 정보통신학부
e-mail:sjjeong@bu.ac.kr

Development of C Compiler using GCC generator

Sam-Jin Jeong*
*Dept of Information and Communication, Baekseok University

요 약

본 연구는 앞으로 개발될, 혹은 현재 개발이 진행중인 16 비트 CPU를 위한 C 컴파일러를 개발하고자 한다. 본 연구는 다양하고 특별한 용도의 새로운 CPU를 위한 새로운 C 컴파일러들을 보다 쉽게 개발할 수 있게 한다.

공개 소프트웨어인 GNU C 컴파일러 생성기를 사용하여 새로운 CPU의 기능들을 명세하고, 기계 의존 원시 파일들을 수정함으로서 새로운 컴파일러를 개발할 수 있다. 개발된 컴파일러는 16 비트 CPU가 지원하는 16 비트 산술 연산 뿐만 아니라, 16 비트 CPU가 지원하지 않는 16 비트 산술 연산, 32 비트 Data Movement 연산, 32 비트 산술 연산, 32 비트 floating point 연산까지 가능하다. 그러나, 배열, 포인터, 구조체 등과 같은 고급 기능들을 지원하기 위해서는 더 많은 연구가 필요하다.

1. 서론

컴퓨터와 정보통신산업은 대부분의 사람들이 예상했던 것보다 더욱 빠르게 발전되고 확장되고 있다. 각종 전자 제품, 통신기기등과 같은 특수한 목적에 범용 프로세서를 사용할 경우 상당히 비효율적이기 때문에, 사용 목적에 따라 그 목적에 적합한 프로세서를 제작하고자 하는 주문형 프로세서(ASIC)에 관한 연구가 매우 활발하게 진행되고 있어, 앞으로 다양한 용도의 주문형 프로세서의 출현될 것으로 예상된다.

새로운 프로세서가 개발되면 기존의 프로세서와는 다른 새로운 명령어(기계어)를 인식하므로 이에 적절한 어셈블러가 필요하게 되고, 고급 언어의 사용을 위해 새로운 컴파일러가 필요하게 된다. 새로운 프로세서를 위한 어셈블리어와 컴파일러 개발 작업은 프로그램 개발 환경이 잘 갖추어져 있는 기존의 범용 컴퓨터 시스템에서 이루어진다.

새로운 프로세서를 위한 컴파일러를 개발하는 방법에는 직접 설계 및 프로그램으로 작성하는 방법과 public domain에 공개된 컴파일러를 이용하여 관련된 부분을 수정하여 개발하는 방법이 있다.

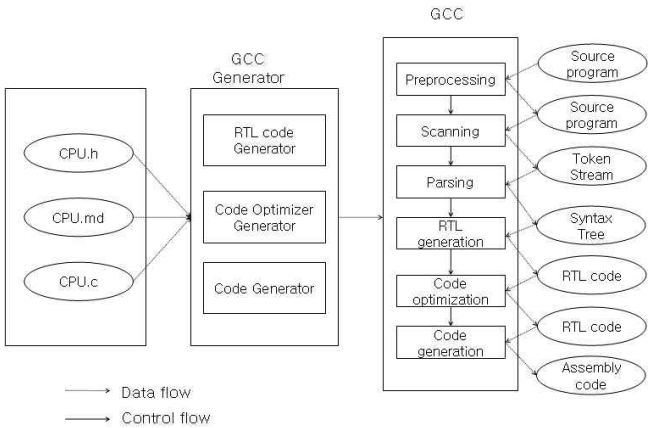
본 연구는 개발 기간이 단축이 되고, 안정성이 있고, 또한 향후 새로운 프로세서를 위한 컴파일러 개발의 장래성을 위해 두 번째 방법을 택하여 UNIX 환경 하에서 사용할 수 있는 GNU C 컴파일러(간단히 GCC라고 부른다)를 이용하여 개발하였다. 일반적으로 컴파일러는 대상이 되는 프로세서와 무관한 전반부(front end)와 프로세서와 밀접한 관계가 있는 후반부(back end)로 크게 두 부분으로 나누어진다[1][2]. GCC는 대단히 방대한 분량의 소프트웨어이지만, 후반부는 비교적 간단하여 여러 가지 대상 프로세서를 위한 컴파일러를 적은 노력으로 개발할 수가 있다[5].

본고의 2장에서는 GCC의 동작원리에 관하여 살펴보고, 3장에서는 실질적인 C 컴파일러의 구현방법에 대해서 설명하며, 4장은 새로 개발된 C 컴파일러에 의해서 floating point 산술 연산에 의해 생성된 어셈블리 코드를 보여 준다.

2. GCC의 동작 원리

GCC(GNU C Compiler)는 Free Software Foundation에서 개발하여 공개한 32비트용 범용 컴파일러이며, 대부분의 32비트 머신에서 그 머신의 특성에 맞게 GCC를 생성할 수 있는 GCC 생성기(Generator)에 의해서 만들어 진다.

GCC 생성기는 [그림 1]에서 보여주는 것과 같이 구문 트리(Syntax Tree)를 RTL 코드로 변환해 주는 RTL Code Generator를 생성하는 RTL Code Generator Generator, RTL Code를 최적화하는 생성기인 Code Optimizer Generator, 그리고 최적화된 RTL Code를 Assembly Code로 전환해 주는 Code Generator를 생성하는 Code Generator Generator가 있다.



[그림 1] GCC의 동작

새로 개발된 CPU를 위한 GCC를 개발하고자 한다면, 새로운 CPU의 하드웨어적 특성에 맞게 CPU,H를 작성하고, CPU의 어셈블리 코드를 적용한 CPU.C와 CPU.MD를 만들어야 한다. 또한 새로운 CPU가 현재 GCC에서 지원하고 있는 CPU와 현저하게 차이가 날 경우에는 RTL 코드를 생성하는 RTL Code Generator Generator를 수정하여야 한다.

GCC는 [그림 1]에서 오른편에서 보여주는 것과 같이 C 언어로 작성된 원시 프로그램을 여러 단계를 거쳐 컴파일한다.

GCC에서는 Free Software Foundation에서 개발하여 공개한 파서 생성기인 GNU bison을 이용하여 생성된 파서를 사용한다[3][4]. GCC의 파싱단계는 어휘 분석 단계에서 생성된 토큰 열을 받아서 프로그램을 문장 단위로 처리하여 각각의 문장을 구문 트리로 변환한다.

RTL은 Register Transfer Language의 약자로서, gcc가 구문 트리를 중간 코드로 변환하는데 중간코드로 사용되는 언어이다. RTL의 형태는 lisp 언어의 표현 방법인 리스트(list)이다. 따라서 lisp 언어의 특징인 패턴 매칭을 용이하게 처리할 수가 있다. 이 단계를 거쳐 구문 트리가 RTL로 변환되면, 이후에 이루어지는 컴파일러의 모든 작업은 RTL을 대상으로 이루어지게 된다.

코드 최적화 단계에서는 RTL을 대상으로 여러 가지 최적화 작업을 하게 된다. 최적화단계에서는 loop 최적화, 공통수식 제거, 레지스터(register) 할당 등과 같은 작업을 통하여 최종적으로 효율이 좋은 코드를 생성하기 위한 단계이다. RTL을 대상으로 작업을 하여 효율이 좋은 RTL을 생성해 낸다.

마지막 단계는 코드 생성 단계로서, RTL로부터 대상 프로세서를 위한 어셈블리코드를 생성한다[6].

3. C 컴파일러의 구현 방법

본 장에서는 실제 32비트 컴파일러인 GCC를 이용하여 새로운 16비트 CPU를 위한 C 컴파일러를 만들기 위한 구현 방법이 제시된다.

다음은 CPU가 지원하는 명령어들과 CPU 가 지원하지 않는 명령어들의 구현에 대해서 설명한다.

3.1. CPU 지원 명령어 구현

CPU가 하드웨어적으로 직접 처리할 수 있는 명령어들, 즉 곱하기와 나누기를 제외한 16비트 정수 연산과, 하나의 명령으로 많은 작업을 하여야 하는 기능들이 있다.

Data Movment 명령어에는 LOAD, STORE, LINK, ALLOCATE 와 UNLK 명령어가 있고, 산술(Arithmetic) 명령어에는 ADD, SUB ,MUL, DIV, MOD, CMP와 TST 명령어가 있다. 논리 명령어에는 AND, OR, XOR 명령어가 있고, Shift & Rotate 명령어에는 ASHIFT, ASHIFTRT, LSHIFT, LSHIFTRT 명령어가 있으며, 비교(Comparision) 명령어에는 Jump on equal(beq), Jump on not equal(bne), Jump on above(bgt), Jump on above equal(bge), Jump on below(blt), Jump on below equal(ble)등의 명령어들이 있다. Program Control

명령어에는 Unconditional jump(Absolute branch, goto), Call, Return 명령어가 있다. 위의 명령어들은 다음 장에 소개된 명령어들을 구현하기 위해서 사용되며 구현 내용은 생략한다.

3.2. CPU 지원하지 않는 명령어 구현

3.2.1 16비트 산술 연산

CPU의 어셈블리어로 지원되지 않는 기본적인 16비트 산술 연산에는 곱하기, 나누기, 모듈러 연산 등이 있다. m68k에서는 이러한 연산에 대한 RTL 코드가 생성되면 m68k.md 파일에 있는 'mulhi3', 'divhi3' 'modhi3' 등의 insn과 패턴 매치되어 해당 어셈블리어를 출력하나, 본 컴파일러에서는 해당 어셈블리어가 없으므로 라이브러리를 호출하는 방식으로 구현 하였다. 즉, 관련되는 오퍼랜드를 스택에 저장하고, 해당 라이브러리를 호출하여 어셈블리어를 출력하도록 하였다.

3.2.2. 32비트 Data Movement 연산

int로 선언된 변수는 16비트의 HImode로 RTL 코드가 생성되고, long int로 선언된 변수는 32비트의 SImode로 RTL 코드가 생성된다. 본 CPU는 16비트의 데이터 레지스터를 사용하기 때문에, HImode의 RTL 코드의 경우 해당 어셈블리어만 존재한다면 m68k.md 파일에서 매치되는 insn에 의해 어셈블리가 바로 출력될 수 있다. 그러나, SImode의 RTL 코드는 모든 32비트 오퍼랜드를 16비트씩 나눈 다음 현재 제공되는 16비트용 어셈블리어들의 조합으로 처리해 주어야 하는 부가적인 작업이 더 필요하다.

32비트 오퍼랜드를 16비트 레지스터로 다루는데 기초적인 역할을 하는 assign expression 즉 데이터 이동과 관련된 명령어들의 구현이 필요하다. 32비트 데이터의 이동은 16비트 데이터의 이동 명령어인 LOAD, STORE로 출력되는데, 이것은 m68k.md 파일에 있는 'movsi' insn에서 처리된다.

HImode를 처리하는 'movhi' insn에서는 출력 패턴에서 해당 어셈블리어를 직접 출력 했으나, 'movsi'에서는 m68k.c 파일에 새로 생성해 놓은 hi_output_move_double() 함수를 호출한다. 이 함수 안에서 32비트 오포랜드를 16비트로 분할하는데 필요한 주된 작업이 수행된다. 이 함수 외에 32비트 데이터의 처리와 관련되어 새로 생성한 함수들로는 split_single(), hi_singlemove_string() 등이 있다.

3.2.3. 32비트 정수형 산술 연산

모든 연산은 그 오퍼랜드가 상수 간의 연산인지 변수 간의 연산인지 아니면 변수와 상수간의 연산인지에 따라 구현 방법이 조금씩 달라지게 되며, 여기에 따라 RTL 코드도 다르게 생성된다. 이 중에서 상수간의 연산은 GCC에서 RTL 코드 생성 전에 미리 계산하여 그 계산된 값으로 RTL 코드를 생성하므로 특별히 연산 과정에서 처리해야 할 사항은 없고, 다만 계산된 상수를 16비트씩 분리하여 읽어 들이는 작업이 필요한데 이것은 split_single() 함수에 의해 처리된다. 그리고 상수와 변수간의 연산은 RTL 코드 생성 자체를 재조절해야 하는 작업이어서 아직 구현되지 않았다. 그러므로 본 연구에서 구현된 연산식은 변수간의 연산식이 되겠다.

3.2.4. 32비트 floating point 산술 연산

floating point number는 32비트의 SFmode로 RTL 코드가 생성된다.

본 시스템에서는 이러한 floating point number들을 다루기 위한 floating point register가 따로 존재하지 않으므로, 16비트 데이터 레지스터로 처리해야 한다. 그러므로, long int 타입의 처리에서와 마찬가지로 32비트 오퍼랜드를 16비트씩 분할하는 과정이 필요하다.

상수간의 연산은 다른 데이터 형에서처럼, GCC에서 RTL 코드 생성 전에 미리 계산하여 그 계산된 값으로 RTL 코드를 생성하므로, 계산된 데이터 값을 16비트씩 읽어 들이는 작업만이 필요하다. 그 과정을 간단히 설명하면, 계산된 상수 값을 갖는 SFmode의 RTL 코드가 생성되면 m68k.md 파일의 'movsf' insn과 패턴매치 되고, 그 매치된 insn의 출력 패턴에서는 hi_output_move_double()함수를 호출한다. 그러면 그 함수에서는 상수를 16비트씩 나누기 위해 split_single() 함수를 호출하고, split_single() 함수내에서는 floating point number 상수를 다루기 위한 함수 split_fp()를 호출한다.

4. 목적 프로그램 생성

본 장은 floating point 산술 연산에 관한 간단한 예제 프로그램을 통하여 C 원시 프로그램이 목적 머신에 맞는 어셈블리어로 변환된 예를 보여준다.

4.1 C 원시 프로그램

```
main()
  {
    float a, b, c, d;

     a=1.2;
     b=2.5 ;
     c=a*b;
    }
```

4.2 어셈블리 코드(Assembly Code)

```
#NO_APP
gcc2_compiled.:
___gnu_compiled_c:
.text
        .even
.globbl _main
_main:
  PUSH M(SP) = A1(L)          ;;link.w a6,#-16
  PUSH M(SP) = A1(H)
  LEA A1 = SP
  LEA R3 = SP(H)
  LEA R4 = SP(L)
  ADD R4 += -16
  LEA SP(H) = R3
  LEA SP(L) = R4
  LOAD R1 = 16281                    ;; MOVE.1
#0x3f99999A,-4(a6)
  STORE *(A1-4) = R1
  LOAD R1 = 39322
  STORE *(A1-2) = R1
  LOAD R1 = 16416                    ;; move.1
#0x40200000,-8(a6)
  STORE *(A1-8) = R1
  LOAD R0 =0
  STORE *(A1-6) =R0
  LOAD R1 = *(A1-6)                  ;;move.1
-8(a6),-(sp)
  PUSH M(SP) =R1
  LOAD R1 = *(A1-8)
  PUSH M(SP) = R1
  LOAD R1 = *(A1-2)                  ;; move.1
-4(a6),-(sp)
  PUSH M(SP) = R1
  LOAD R1 = *(A1-4)
  PUSH M(SP) = R1
  CALL ___mulsf3              ;; jsr ____mulsf3
  LEA R5 = SP(L)              ;; addq.w #8,sp
  LEA  R6 = SP(H)
  ADD R5 += 8
  ADC R6 = R6 + CARRY
  LEA SP(L) = R5
  LEA SP(H) = R6
  STORE *(A1-12) =  R0    ;;move.1 d0,-12(a6)
  STORE *(A1-10) = R1
L1:
  LEA SP = A1                 ;; unlk a6
  POP A1(H) = M(SP)
  POP A1(L) = M(SP)
  RET                         ;;rts
```

5. 결 론

본 논문에서는 새로 개발된 16비트 CPU를 위한 C 컴파일러를 개발하기 위해서 기존의 32비트 GNU C 컴파일러를 분석하고, 이를 수정하여 이용하는 방법을 제시하였다.

GNU C 컴파일러는 32비트 CPU용 범용 C 컴파일러이기 때문에 GNU C 컴파일러를 이용하여 새로운 프로세서를 위한 컴파일러를 개발하는 작업은 대상 프로세서가 32비트일 경우에 비교적 용이하게 C 컴파일러를 개발 할 수가 있다.

본 연구에서는 완전한 컴파일러까지는 완성하지 못하였지만, 프로세서가 하드웨어적으로 지원하고 있는 기능들에 대해서는 정확하게 동작하고 있다. 그리고 C 프로그램 문장으로 본다면, 산술 연산, 관계 연산, 논리 연산, 할당문, if문, for문, while문, 함수 호출등과 같은 문장들을 정확하게 컴파일하고 있다. 특히 CPU가 하드웨어로 지원하지 못하는 기능인 16비트 곱하기와 나누기, 그리고 32비트 산술연산들은 RTL Code Generator Generator를 분석, 수정하여 완벽하게 동작한다.

본 연구에서 개발된 C 컴파일러는 완전한 컴파일러는 아니지만 산술 연산, 함수 호출과 같은 부분은 완벽하여 특정 분야에 활용되어질 수 있고, 이 과정에서 축적된 기술을 적용하여, 앞으로 특정한 형태의 주문형 프로세서에 대해서 짧은 시간 내에 새로운 컴파일러를 개발할 수 있으리라 기대된다.

참고문헌

[1] Alfred V.Aho,Ravi Sethi and Jeffrey D. Ullman, "Compilers: Principles, Techniques, and Tools", Addison-Wesley, 1986.

[2] Andrew W. Appel, "Modern Compiler Implementation in Java", Cambridge University Press, 1998.

[3] Charles Donnelly and Richard Stallman, "Bison: The YACC-compatible Parser Generator", Free Software Foundation, Dec., 1992.

[4] MasBr90 Tony Mason and Doug Brown, "lex & yacc", O'Reilly & Associates, Inc., 1990.

[5] Richard M. Stallman, "Using and Porting GNU CC", Free Software Foundation, Oct., 1993

[6] 정삼진, “16-bit CPU용 C 컴파일러 개발”, 2009년 한국산학기술학회 춘계 학술발표논문집.