마이크로프로세서 설계 무작정 따라하기 part-II (2)

부제: 명령어 시뮬레이터, 어셈블러, 컴파일러의 개발

KAIST 전자전산학과 박사과정 배영돈(donny@ics.kaist.ac.kr), 이종열(jylee@ics.kaist.ac.kr)

이번 강좌에서는 GNU C 컴파일러에 관련된 여러 가지 사항들에 대하여 간단히 짚어보고, GNU C 컴파일러를 이용한 컴파일러의 개발 과정에 대해서 살펴봄으로써 이어지는 다음 강좌에서 필요한 기본 지식을 마련해 보도록 한다.

1. GNU C 컴파일러란?

GNU C 컴파일러는 미국의 Free Software Foundation 이란 비영리 단체에서 무료로 제공되는 C 컴파일러이다. (http://www.gnu.org 참조) GNU C 컴파일러는 약 20 년의 역사를 가지고 있으며, 최초 개발 시부터 포팅(porting)을 염두에 두고 개발되었기 때문에, 컴파일러의 전위(front end, 프론트 엔드)와 후위(back end, 벡엔드)를 수정하여 다양한 종류의 컴파일러를 다시 만들어 내는 것이 가능하다. 표 1에서 볼 수 있듯이 현재 많은 수의 컴파일러들이 전위와 후위를 변경하여 개발 되었다.

표 1. GNU C 컴파일러의 여러 가지 전위와 후위들

전위 (front end)	C, C++, Objective C, Ada 95, Fortran, Modula-3, PASCAL, Chill
후위 (back end)	Acorn RISC, DEC, Am29000, AMD21xx, DSP16xx, TMS320C4x, SPARC, MIPS, Pentium, MIPS, RS6000, ARM, MCORE, …

GNU C 컴파일러가 C 프로그램을 컴파일하는 과정은 크게 세 개의 과정, RTL 생성, 최적화, 코드 생성으로 구분할 수 있으며, 각 과정은 다시 몇 개의 세부 과정으로 구분된다. 그림 1에는 GNU C 컴파일러의 각 단계가 표시되어 있다.

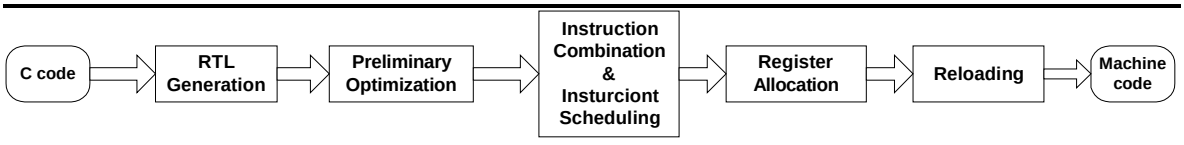

그림 1. GNU C 컴파일러의 단계

(1) RTL 생성 (RTL generation) 단계

C 원시 프로그램(source code)으로부터 초기 단계의 RTL 이 생성된다. RTL (Register Transfer Language)이란 GNU C 컴파일러에서 내부적으로 사용되는 중간 단계 표현 방식(Intermediate representation, IR)이다. RTL 생성 이후의 모든 컴파일러의 단계는 C 원시 프로그램을 표현한 RTL 코드를 기반으로 수행된다. 이 단계에서는 YACC 기반의 파서(parser)에 의하여 소스 코드가 분석되고 이로부터 parse tree 가 생성되며, RTL 코드는 이 parse tree 로부터 생성된다.

(2) 예비 최적화(preliminary optimization)

이 단계에서는 분기 최적화를 통하여 불필요한 코드가 제거 된다. 그리고 공통 부분식 제거(common sub-expression elimination), 상수 전파(constant propagation), 루프 최적화, 자료의 흐름분석(data flow analysis) 등의 최적화가 수행된다. 이 단계에서 수행되는 최적화는 대부분이 컴파일러를 포팅하고자 하는 목표기계(target machine)에 의존하지 않는 최적화 (machine independent optimization)이다. 이 단계에서 수행되는 최적화에 대해서는 V. Aho, M. R. Sethi, J. D. Ullman 등이 저술한 책(*Compiler—Principles, Techniques, and Tools)* 을 참조하기 바란다.

(3) 명령어 결합과 스케쥴링(instruction combination and scheduling)

RTL 코드 명령어들의 다양한 결합을 시도함으로써 좀더 최적화된 명령어들을 생성하고 명령어들의 순서를 재배치하여 기능 유닛(functional unit)이나 데이터의 충돌을 막는 단계이다.

(4) 레지스터 할당(Register allocation)

GNU C 컴파일러는 레지스터 할당이 수행되기 전까지의 컴파일 단계에서는 레지스터의 수가 무한하며 한 종류의 레지스터만이 있는 것으로 가정한다. 이렇게 가정된 레지스터를 가상 레지스터(pseudo register)라고 한다. (이에 대하여 머신에 존재하는 레지스터를 실재 레지스터라고 한다.) 이 단계에서는 가상 레지스터에 실재 레지스터를 매핑(mapping)하게 된다.

(5) 리로딩(Reloading)

레지스터 할당 단계를 거친 후에도 실재 레지스터를 할당받지 못 한 가상 레지스터들을 메모리에 할당하는 과정인 리로딩이 수행된다.

2. GNU C 컴파일러의 포팅(porting) 과정

GNU C 컴파일러의 벡엔드, 즉 코드 생성과 관련된 부분을 사용자가 수정하거나 또는 필요한 내용을 새로 작성함으로써 목표기계(target machine)에 맞는 컴파일러를 개발하는 것을 포팅이라고 한다. 예를 들면, SPARC 워크스테이션에서 ARM 용 컴파일러를 구현한다면 SPARC 은 주기계(host machine)이 되며 ARM 은 목표기계(target machine)이 된다. 그리고 여기서 생성된 ARM 용 컴파일러(이 컴파일러는 SPARC 에서 수행된다.)를 GNU C 컴파일러에서는 크로스 컴파일러(cross compiler)라고 한다.

GNU C 포팅은 다양한 목표기계에서 가능하지만, 모든 머신에서 가능한 것은 아니다. 예를 들어, MIPS 전용 어셈블러를 사용하는 MIPS 용 크로스 컴파일러를 GNU C 컴파일러를 포팅하여 구현하는 것은 불가능하다. 그 이유는 GNU C 컴파일러의 일부 코드가 MIPS 컴퓨터 상에서만 컴파일되기 때문이다. 그러나 만약, MIPS 용 GNU 어셈블러와 링커를 사용한다면 크로스 컴파일러의 생성이 가능하다. (어셈블러와 링커에 관한 내용은 컴파일러에 관한 강좌가 끝난 후에 이어진다.)
주기계와 목표기계의 부동 소수점 형식이 다른 경우 크로스 컴파일러가 정상적으로 동작하지 않을 수도 있다. 또 두 머신에서 워드의 크기가 달라지는 경우에도 문제점이 발생할 수 있다. 따라서 이와 같은 경우에는 주의가 요망된다.

GNU C 컴파일러의 포팅은 컴파일러 후위부 (back end) 작성 단계, 컴파일 환경 설정 단계, 컴파일러 생성단계로 구성된다.

(1) 컴파일러의 후위부 (back end) 작성 단계

컴파일러의 코드 생성은 목표기계에 따라서 크게 달라지기 때문에 목표기계에 대한 정확하고 충실한 정보가 요구된다. 이때 컴파일러가 목표기계를 모두 이해할 필요는 없으며, 컴파일러 입장에서 필요한 정보만 제공되면 된다. GNU C 컴파일러는 코드 생성에 필요한 정보를 목표 설명 매크로(target description macro)와 머신 설명(machine description)을 통하여 얻게 된다. 새로 생성되는 GNU C 컴파일러는 목표기계의 정보를 코드로 가지고 있는데, 이것은 최적화 과정에서 목표기계의 정보를 필요로 하기 때문이다.

가) 목표 설명 매크로(target description macro)

목표기계의 하드웨어적인 특성에 관한 정보를 나타낸다. 목표기계의 이름이 machine 이라면 “machine.h” 란 이름의 파일에 각종 정보들이 기술된다. 여기에는 엔디안(endian), 한 워드당의 비트수, 레지스터 정보, 스택 구성, 분기 방식등 하드웨어에 밀접하게 관련된 정보들이 “`#define`”으로 정의된 매크로의 형식으로 표현된다. 그림 2는 목표 설명 매크로의 일부 내용이다.

```
#define BYTES_BIG_ENDIAN        1
#define BITS_PER_UNIT           16
#define BITS_PER_WORD           16

#define POINTER_SIZE            16
#define STACK_BOUNDARY          16
```

그림 2. 목표 설명 매크로

목표 설명 매크로에는 500 개 이상의 매크로가 존재하지만, 이 것들을 모두 정의할 필요는 없고 목표기계와 관련된 매크로들만 정의하면 된다.

실제로 포팅하는 과정에서 매크로들의 의미를 정확히 파악하고 모두 다 독자적으로 정의하는 것은 상당히 힘든 작업이다. 따라서 가능하면 이미 컴파일러가 포팅된 기계 중에서 목표기계와 유사한 구조를 가지는 기계를 골라, 그 컴파일러의 목표 설명 매크로에서 시작하여, 달라지는 부분을 수정해 나가는 방식을 취하는 것이 바람직하다. 이미 다양한 목표기계에 대하여 GNU C 컴파일러가 포팅되어 있기 때문에 사용자가 원하는 목표기계와 유사한 구조를 지진 것을 찾는 것은 그다지 어렵지 않을 것이다.

나) 머신 설명(machine description)

머신 설명 매크로가 하드웨어의 구조에 관련된 정보를 제공하는 것이라면, 머신 설명은 기계에서 제공되는 명령어들의 종류와 동작에 관련된 정보를 제공하는 역할을 한다. 그림 3은 머신 설명 파일의 일부분으로 각각 C 원시코드(source code)에서 사용된 덧셈 연산과 부동소수점 수의 부호를 변경하는 연산은 어떤 머신 명령어로 변환되어야 하는 가를 기술하고 있다.

```
(define_insn "addsi3"
      [(set  (match_operand:SI   0   "register_operand"   "=r")
               (plus:SI (match_operand:SI   1   "register_operand"   "%r")
                           (match_operand:SI   2   "register_operand"   "r")))]
      ""
      "add %0, %1, %2"
      [(set_attr  "type" "arith")])

(define_expand "negsf2"
      [(set  (match_dup 2)
                (match_dup 3))
      (parallel [(set  (match_operand:SF  0   "register_operand"   "=r")
                                 (neg:SF (match_operand:SF   1   "register_operand"
"0")))
                     (use (match_dup 2))])]
      ""
      "operands[2] = gen_reg_rtx(SImode);
        operands[3] = GEN_INT(0x80000000);")
```

그림 3. 머신 설명 파일

머신 설명은 RTL 형식으로 작성되며 목표기계가 가지고 있는 명령어들을 자세하게 기술한다. 이 파일은 여러 개의 ".c" 및 ".h" 파일로 변환된다. 이렇게 변환된 파일들은 GNU C 컴파일러 원시 프로그램(source code)과 함께 컴파일되어 최종적인 컴파일러가 만들어 진다.

(2) 컴파일 환경 설정 단계

목표 설명 매크로 파일과 머신 설명 파일을 작성한 후, 이 것들을 GNU C 원시 프로그램과 함께 컴파일 하려면 여러 가지 컴파일 환경을 적절히 설정해야 한다.

작성된 목표 설명 매크로 파일과 머신 설명 파일을 적절한 위치에 저장하여야 한다. GNU C 컴파일러의 원시 프로그램들이 담긴 디렉토리를 보면, 'config'라는 서브디렉토리를 찾을 수 있고 이 곳에 들어 가면 각종 목표 기계의 이름을 가진 디렉토리들을 발견하게 된다. 여기에 "SimpleCore"라는 이름으로 디렉토리를 만든다. 그리고 만들어진 "SimpleCore" 디렉토리 안에 목표 설명 매크로 파일과 머신 설명 파일을 저장한다.

다음으로 크로스 컴파일러를 지원할 여러 툴들과 라이브러리 파일을 적절한 위치에 저장한다. 어셈블러와 링커가 있다면 "~/SimpleCore/bin" 저장하도록 한다. 이때 각 프로그램은 다음과 같은 이름으로 저장되어야 한다.

- as : 어셈블러
- ld : 링커
- ar : 압축기(archiver)로 목표기계의 형식에 맞게 압축 파일를 다루는 프로그램
- ranlib : 압축파일에서 심볼 테이블을 생성하는 프로그램

크로스 컴파일러에서 사용할 수 있는 표준 C 라이브러리가 있으면, "~/SimpleCore/lib" 디렉토리에 저장한다. 또 'crt0.o'나 'crtn.o'와 같은 시작 파일(startup file : C 의 main 함수가 수행되기 전에 초기화를 위하여 수행되어야 하는 라이브러리 프로그램)들도 이 디렉토리에 저장한다. 표준 C 라이브러리 함수의 프로토타이핑 정보를 갖는 표준 헤더 파일들은 "~/SimpleCore/include" 디렉토리에 저장한다.

크로스 컴파일러를 개발할 때 중요하게 대두되는 문제는 라이브러리이다. 예를 들어, 고정소수점만을 지원하는 DSP 프로세서용 컴파일러를 개발할 때 C 프로그램 상의 부동소수점 연산을 지원하기 위해서는 소프트웨어적으로 부동소수점 연산을 처리하는 라이브러리 함수를 제공하여야 한다. GNU C 컴파일러에서 이런 소프트웨어 라이브러리를 사용하려면, 'libgcc1.a'라는 이름의 라이브러리를 컴파일러 개발자가 제공해야 한다. GNU C 컴파일러에서는 부동소수점 연산을 고정 소수점 연산으로 에뮬레이션해주는 C 라이브러리 함수를 적절히 수정함으로써 라이브러리 'libgcc1.a'를 개발할 수 있다.

라이브러리를 등록한 후, 'config.sub'와 'configure'라는 스크립트 파일에서 목표기계인 SimpleCore 에 관련된 정보를 등록하여야 한다. 이 스크립트들은 후에 Makefile 을 생성하는데 사용된다. 수정 부분은 그림 4와 그림 5은 각각 'config.sub'와 'configure' 파일에서 수정되어야 하는 부분을 보이고 있다.

```
...
...
cpu_type =`echo $machine | sed 's/-.*$//'`
case $machine in
SimpleCore*-*-*)
  cpu_type = SimpleCore
  ;;
alpha*_*_*)
  cpu_type = alpha
  ;;
...
...
```

그림 4. config.sub 의 수정

```
...
...
case $basic_machine in
    ....
    tahoe | i860 | ....          \
          ....                    \
          | spac | spaclet | SimpleCore)
...
...


...
...
tm_file=${cpu_type}/${cpu_type}.h
xm_file=${cpu_type}/xm-${cpu_type}.h
case $machine in
SimpleCore-*-*)
  tm_file=${cpu_type}/SimpleCore.h
  xm_file=${cpu_type}/xm-SimpleCore.h
  tmake_file=t-SimpleCore
  use_collect2=yes
  ;;
...
...
```

그림 5. configure 의 수정

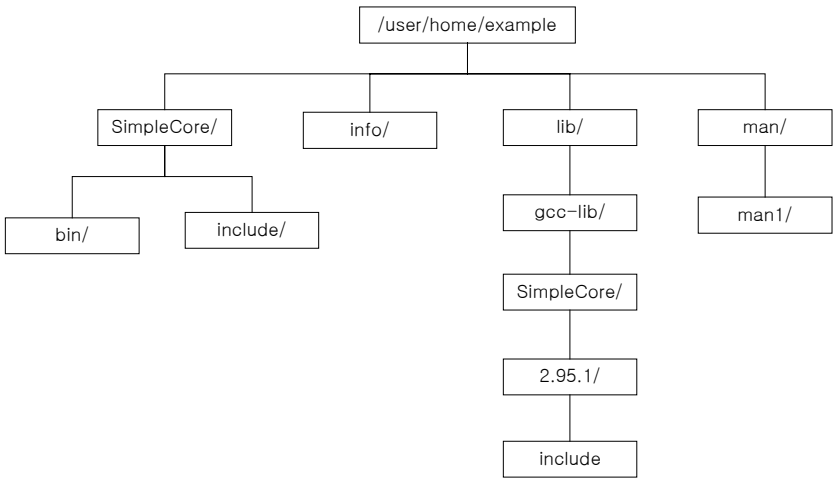

그림 6. GNU C 컴파일러의 설치

(3) 컴파일러 생성 단계

컴파일 환경 설정이 완료되면, GNU C 컴파일러 원시 프로그램을 컴파일하여 목표기계를 위한 GNU C 컴파일러를 생성한다. 여기서는 UNIX 운영체제 상에서 GNU C 컴파일러를 생성하는 과정에 대하여 설명한다.

- ‘make distclean’을 수행한다. 이 명령은 ‘Makefile’을 포함하여 이전 컴파일에서 생성된 모든 파일을 삭제한다.
- ‘configure’ 스크립트를 수행하여 개발 기계(build machine), 주기계, 목표기계를 설정한다. 이 스크립트를 수행할 때 ‘—target=SimpleCore —prefix=~/SimpleCore’라는 옵션을 주도록 한다. 앞의 옵션은 목표

머신을 지정하는 옵션이다. 그리고 '—prefix=~/SimpleCore' 옵션 은 컴파일러가 설치되는 위치를 지정한다. 이 스크립트의 결과로 'Makefile', 'config.h', 'tconfig-h', 'tm.h' 등이 생성된다.

- 'make LANGUAGES=c' 명령을 수행한다. 'LANGUAGES=c'는 C 컴파일러만을 생성하기 위한 것이다. 이 명령을 수행하면 앞에서 생성된 파일들을 이용하여 크로스 컴파일러가 만들어 진다.
- 마지막으로 'make LANGUAGES=c install' 명령을 사용하여 만들어진 컴파일러를 설치한다.

컴파일러가 성공적으로 설치되면, 최종적으로 그림 6과 같은 디렉토리가 구성된다. C 컴파일러인 'cc1'과 선행처리기인 'cpp'는 "~/SimpleCore/lib/gcc-lib/SimpleCore/version/" 디렉토리에 설치되며 컴파일러 드라이버인 'gcc'는 "~/SimpleCore/lib/gcc-lib/SimpleCore/bin/"에 설치된다.

[참고문헌]

- V. Aho, M. R. Sethi, and J. D. Ullman, *Compiler—Principles, Techniques, and Tools*, Addison-Wesly, Reading, MA, 1986.
- R. M. Stallman, *Using and Porting GNU CC for version 2.6*, Free Software Foundation Inc., 1996.
- 황승호, 이대현, 이종열, *컴파일러 개발—GNU C 컴파일러 포팅을 중심으로*, 시그마프레스, 2001

* 다음 회에 계속됩니다. 다음 호에서는 Machine description 과 Target machine macro 를 작성하여 Cross compiler 를 생성하는 과정을 설명합니다.

본 강좌에서 사용된 예제는 홈페이지(http://www.donny.co.kr/simplecore)를 통해서 제공됩니다.