어 셈 블 리 H O W T O V 0 . 3 C

원저자 : Francois-Rene Rideau *rideau@ens.fr* <mailto:rideau@ens.fr> 버 젼 : 1996.6.15 0.3C

번역자 : 한동훈 *ddoch@hitel.kol.co.kr* <mailto:ddoch@hitel.kol.co.kr> 번역일 : 1996.9.27

(번역주: 이 번역문서는 부족한 저의 문장실력으로 필요성에 의하여 번역한 결과로 잘못된 오역과 의미에 이상없는 부분이 빠질 수도 있음을 미리 알립니다. 따라서 잘못된 번역으로 오는 책임은 저에게 없으며, 만일 수정해야 할 곳이 있다면 저에게 연락을 해주시기 바라며, 원저의 오류라면 원저자에게 연락해주시기 바랍니다.) 어셈블리 HOWTO aka *Free* 32-bit x86 어셈블리 FAQ aka Linux x86 어셈블리 HOWTO == 프리 프로그래밍 툴을 사용하는 x86 어셈블리 하우투 == 검색어 :assembly, assembler, free, macroprocessor, preprocessor, asm, inline asm, 32-bit, x86, i386, gas, as86, nasm Copyright (C) 1996 Francois-Rene Rideau. 여러분들은 이 문서를 변경하지 않는 선에서 마음대로 배포할 수 있으며, 조금의 주석은 가해도 상관없다. 여러분들은 다른 경우에 나에게 이문서의 배포에 대해 자유롭게 물어볼 수 도 있다. 리눅스 문서화 계획의 진행자들은 이 문서를 자유롭게 할 것이며, 다른 LDP 문서들도 곧 바로 허락이 될 것이다. 중요한 점: 이 문서는 어디까지나 베타 버젼이다. 여러분들에게는 다음과 같은 툭수한 권리 가 주어진다. 질문하기, 질문에 답하기, 주어진 답변을 바로 정정하기, 새로운 FAQ 답변들에 추가하기, 다른 소프트웨어에 암시를 주기, 현재의 개발자(나)에게 욕하기, 그리고 FAQ진행을 나누어서 하기, 왜냐하면 나는 다른 일을 하고 있기 때문이다... 다른 경우, 나에게 연락해 달라.(메일: *rideau@ens.fr* <mailto:rideau@ens.fr>) 아마도 우리는 Raymond Moon 에게 comp.lang.asm.x86에 있는 거의 FAQ에 이부분을 보태는 것을 납득시킬 수 있을 것이다.

목 차

제 1 절 시작 **3**

1.1 이 문서를 활용하는 방법 . . . 3

1.2 다른 관련 문서 . . . 3

제 2 절 어셈블러들 **3**

2.1 GCC 인라인 어셈블리 . . . 3

2.1.1 GCC가 있는 곳 . . . 4

2.1.2 GCC 인라인 어셈블리 DOC들을 구하는 곳 . . . 4

2.1.3 어떻게 GCC 안에서 정확히 어셈블리 코드를 불러내는가? . . . 4

2.2 GAS . . . 5

2.2.1 어디서 찾을 수 있는가 . . . 5

2.2.2 AT&T 문법은 무엇을 말하는가 . . . 5

2.2.3 2.2.3 제한된 16비트 모드 . . . 5

2.3 2.3 GASP . . . 5

2.3.1 2.3.1 GASP를 어디서 찾을 것인가 . . . 5

2.3.2 2.3.2 GASP는 어떠한 일을 하는가 . . . 6

2.4 2.4 AS86 . . . 6

2.4.1 2.4.1 AS86을 어디서 얻을 수 있는가 . . . 6

2.4.2 doc들을 어디서 찾을 수 있는가 . . . 7
2.4.3 어떻게 어셈블러를 불러오는가 ? . . . 7
2.4.4 이 새로운 버젼으로 리눅스를 컴파일 할수 없다면 어떻게 해야 하는가? . . . 7
2.5 다른 어셈블러들 . . . 7
2.5.1 Win32Forth 어셈블러 . . . 7
2.5.2 NASM . . . 7
2.5.3 Tunes . . . 8
2.5.4 프리가 아니거나 32비트가 아닌 x86 어셈블러들 . . . 8

제 3 절 META 프로그래밍/ 매크로 처리 8
3.1 위에서 통합된 것이란 무엇을 뜻하는가 . . . 8
3.1.1 GCC . . . 8
3.1.2 GAS . . . 8
3.1.3 GASP . . . 9
3.1.4 AS86 . . . 9
3.1.5 다른 어셈블러들 . . . 9
3.1.6 외부 필터 . . . 9
3.1.7 CPP . . . 9
3.1.8 M4 . . . 9
3.1.9 필터를 통한 매크로 처리 . . . 9
3.1.10 메타 프로그래밍 . . . 9

제 4 절 컨벤션 부르기 10
4.1 4.1 Linux . . . 10
4.1.1 GCC로 링크하기 . . . 10
4.1.2 ELF vs a.out 문제들 . . . 10
4.1.3 직접적인 리눅스 시스템 콜 . . . 10
4.2 도스 . . . 10
4.3 여러분들의 OS . . . 10
4.3.1 부트 로더 코드와 32비트 모드로 들어가기 . . . 11
4.3.2 프로텍션의 기초 . . . 11
4.3.3 인터럽트 다루기 . . . 11
4.3.4 16비트 시스템 서비스를 사용하는 V86/R86 모드 . . . 11
4.3.5 이것들에 대한 정보를 어디서 얻을 것인가 . . . 11

제 5 절 해야 할일 11

제 1 절 시작

이 문서는 리눅스 운영체제에서 프리 어셈블러를 사용한 32비트 x86 어셈블리 프로그래밍이나 프로그램들에 대해 질문하는 사람들에게 답변하기 위해 제작 이 되었다. 그것은 또한 프리가 아닌, x86이 아닌, 32비트 어셈블러가 아닌 그러한 것들에 대한 문서들을 언급할 수 있다. 왜냐하면 어셈블리 프로그래밍에서 주로 흥미있는 분야는 운영체제, 언어, 게임, C 컴파일러가 표현을 제대로 하기 힘든 부분들을 건더리는 것이다. 우리는 그러한 소프트웨어 개발을 목표로 하고 있다.

1.1 이 문서를 활용하는 방법

이 문서는 종종 질문하는 것들에 대한 답변들을 포함하고 있다. 많은 경우, 소프트웨어나 문서 사이트에 URL이 주어진다. 가장 유용한 사이트가 미러 되었고, 가까이의 미러 사이트를 통해 이용할 수 있 다면, 여러분들은 아까운 자신의 시간을 버리지 않고, 불필요한 노력을 줄일 수 잇을 것이다. 특이한 경우에, 인기 사이트를 미러하는 세계에서 아주 큰 사이트가 있다. 보통은 미러리스트들이 파일로 제공될 수도 있고 또는, 로긴 메세지에 나타난다. 충고를 받아들여라. 그밖에, 여러분들은 찾고자 하는 것을 archie를 이용함으로써 해결할 수 있을 것이다. 가장 최근의 문서버젼이 있는 곳은 다음과 같다.

http://www.eleves.ens.fr:8080/home/rideau/Assembly

그러나 리눅스 하우투 싸이트는 정말 번개같이 업데이트 된다. 나도 모르는 사이에.. *sunsite.unc.edu/pub/linux/docs/HOWTO/* <ftp://sunsite.unc.edu/pub/linux/docs/HOWTO/> (?)

1.2 다른 관련 문서

* 만일 여러분들이 프리 소프트웨어에 대해 잘 모른다면 GNU 공공 라이센스를 주의 깊게 읽어 보길 바란다. 그것은 많은 프리 소프트웨어들과 대부분의 모 델에서 사용되어 지고 있다: 보통 "COPYING"이라는 파일이름으로 들어 있으며, 라이브러리버젼이라면 "COPYING.LIB" 라는 이름으로 되어 있을 것이다. FSF(free software foundation)의 문장가들은 또한 당신을 도울 것이다. * 특별하게, 흥미있는 프리 소프트웨어들은 대부분 소스를 포함하고 있는 데, 여러분들은 이것을 참고하거나, 수정하거나, 때로는 그냥 그대로 차용해서 사용할 수 있다. 여러분들에게 주어지는 라이센스를 주의깊게 읽고 그기에 따르는 것이 좋다. * x86 어셈블리 프로그래밍에 대한 일반적인 질문과 몇몇의 상업적인 어셈블러 (16비트 도스환경의..)에 대한 질문에 대한 답변들을 담고 있는 comp.lang. asm.x86을 위한 FAQ가 있다. 그중에서 몇개는 프리 32비트 어셈블리 프로그래밍에 적용이 되는 데, 여러분 들은 이 FAQ를 다음에서 읽을 수 있다. *www2.dgsys.com/~raymoon/faq/asmfaq.zip* <http://www2.dgsys.com/~raymoon/faq/asmfaq.zip> * FAQ들과 doc들은 독자 여러분들의 플랫폼상의 프로그래밍에 관련된 것들이 있다. 그리고 플랫폼 의존적인 것들은 어셈블러에서 프로그래밍에 바로 적용 이 되지 않는 다는 점을 참조하여야 한다.

제 2 절 어셈블러들

2.1 GCC 인라인 어셈블리

잘 알려진 바와 같이, GNU 프로젝트에서 중요한 위치를 차지하고 있는 최적화된 32비트 컴파일러인 GNU C/C++ 컴파일러 (GCC)는 x86 아키텍쳐를강력하게 지원 한다. 그리고 C 프로그램안에 어셈블리 코드를 삽입할 수 있는 능력을 제공한다. GCC 는 대부분의 가능한 플랫폼에서 동작한다. 그중에서도 Linux, *BSD, VSTa, OS/2, *DOS, Win*, 등등..

2.1.1 GCC가 있는 곳

GCC 원본사이트는 *prep.ai.mit.edu/pub/gnu/* <ftp://prep.ai.mit.edu/pub/gnu/> 에 다른 GNU 계획에 의한 어플리케이션 소프트웨어들과 같이 발표된다. 그리고, 또한 많은 미러 사이트들이 존재하고 있다. 여러분들의 개방적인 OS에 맞도록 개정된 소스들과 미리 컴파일된 바이너리들이 통상적인 FTP 사이트에있다. inux 의 GCC를 사용한다면 아래에 가본다. *www.linux.org.uk/* <http://www.linux.org.uk/> 가장 인기 있는 도스기반의 GCC는 DJGPP이다. 다음 사이트의 디렉토리에서 발견할 수 있다: *www.delorie.com/djgpp/* <http://www.delorie.com/djgpp/> OS/2 기반의 도스에서 작동하는 GCC는 또한 EMX라 불린다; *www.leo.org/pub/comp/os/os2/gnu/emx+gcc/* <http://www.leo.org/pub/comp/os/os2/gnu/emx+gcc/> *warp.eecs.berkeley.edu/os2/software/shareware/emx.html* <http://warp.eecs.berkeley.edu/os2/software/shareware/emx.html>

2.1.2 GCC 인라인 어셈블리 DOC들을 구하는 곳

GCC 문서는 texinfo 포멧으로 문서파일들을 포함한다. 당신은 그것을 텍스로 변 환할 수 있고, 텍스로 컴파일 할수도 있으며, 프린트를 하던지, 이막스 .info파일 이나 브라우저, 기타 여러분들이 좋아하는 포멧으로 변환할 수 있다. .info 파일은 GCC의 괜찮은 설치본에는 들어 있는 것 같다. 그 부분은 다음과 같다: C 확장::확장된 Asm:: 부분 GCC 불러오기::서브모델 옵션::i386 옵션:: 이러한 것들이 도움이 될 것이다. 세세하게 보면, 그것은 i386의 레지스터를 위 해 특별히 규정된 이름을 제공한다: abcdSDB 는 %eax, %ebx, %ecx, %edx,%esi, %edi, %ebp 와 하나씩 일치한다. (%esp에는 글자가 배당되어 있지 않다.) HTML 포멧으로 변환된 이러한 문서들과 부분들의 URL은 다음과 같다. *www.cygnus.com/doc/usegcc_89.html* <http://www.cygnus.com/doc/usegcc_89.html>#SEC92 DJGPP 게임리소스 (게임 해커들에게 뿐만이 아니라)는 특별하게 어셈블리에 관한 이러한 페이지를 가지고 있다: *www.rt66.com/~brennan/djgpp/djgpp_asm.html* <http://www.rt66.com/~brennan/djgpp/djgpp_asm.html> 마지막으로, 이러한 웹페이지들은 "DJGPP Quick ASM Programming Guide" 로 불리워지고 FAQ들과 AT&T x86 어셈블리 문법, 몇몇의 인라인 어셈블리 정보, .obj/.lib 파일들을 변환하는 것들에 대한 것들로 가득차 있다. *remus.rutgers.edu/~avly/djasm.html* <http://remus.rutgers.edu/~avly/djasm.html>

GCC 는 어셈블링에서는 GAS에 의존하고 아래의 문법을 따른다; 인라인 어셈블리는 인용된 퍼센트 문자를 필요로 한다. 그래서 그것들은 GAS 에게로 건네진다. 아래의 GAS에 대한 부분을 보자. 많은 유용한 예제들을 리눅스의 linux/include/asm-i386/ 소스 서브디렉토리 에서 찾아보자.

2.1.3 어떻게 GCC 안에서 정확히 어셈블리 코드를 불러내는가?

최적화와 인라인 어셈블리를 가능하게 하기 위해서 GCC 를 "-O" 플래그와 같이 불러내자. 그렇게 하지 않는 다면, 코드는 컴파일되기는 하나, 정확히 동작하지 않을 것 이다. 좀 더 일반적으로, x86 플랫폼에서 좋은 컴파일 플래그는

```
gcc -O2 -fomit-frame-pointer -m386
```

정도가 될 것이다. -O2 는 좋은 최적화 레벨이다. 최적화에 더하여 컴파일러는 코드를 크게 만든다. 그러나 그것은 단지 bit faster일 뿐이다; 그러한 과다한 최적화는 루프를 타이트하게 만드는 데 정도에만 유용할 수 있다. 여러분들이 어떻게든 어셈블리에서 사용한다면 말이다. 그것이 필요하다면 필요한 만큼만의 루틴들에 사용하라. -fomit-frame-pointer 는 stupid frame pointer maintenance를 건너뛰게 코드를 생성시키고, 코드를 좀더 작고 빠르게 만들며, 그 이상의 최적화를 위해서 레지스터를 자유롭게 한다. 이것은 디버깅 툴(gdb)들을 사용하기 어렵게 만들긴 하나, 더이상 사이즈와 속도 를 향상 시킬 수 없을 것이다. -m386 은 어떠한 속도의 저하없이 좀 더 콤팩트한 코드를 생성해 낸다. (작은 코드는 또한 디스크 입출력을 적게 수행하고 빠른 실행을 한다는 것을 기억하자.) 그러나 아마도 위에 언급한 타이트한 루프상에서 일 것이다. 좀더 최적화 하려면, -mregparm=2 옵션이나 이에 대응하는 함수가 도움이 될 것이다. 그러나 외부 코드와 링킹을 할때에는 많은 문제점들이 여러분들을 괴롭히게 될 것이다. 여러분들은 이러한 플래그들을 기본설정파일인

```
/usr/lib/gcc-lib/i486-linux/2.7.2/specs
```

에 추가할 수 있다. (이 파일은 시스템에 따라 조금 틀릴 수 있다.)

2.2 GAS

GAS는 GCC와 한쌍으로 움직이는 GNU 어셈블러이다.

2.2.1 어디서 찾을 수 있는가

binutils라 이름붙여진 패키지 속에 GCC가 있는 같은 곳에서 찾을 수 있다.

2.2.2 AT&T 문법은 무엇을 말하는가

GAS는 32비트 유닉스 컴파일러를 제공하기 위해 창안되었기 때문에 표준 AT&T 문법을 이용한다. AT&T 문법은 많은 것들이 표준 680x0 어셈블러와 닮았다. 이 문법은 "Intel" 문법에 비해서 좋지도 않고, 나쁘지도 않다. 단지 다를 뿐이다. 여러분들이 그것을 사용해보게 되면, 인텔 문법에 비해 좀더 많은 규칙이 있음 을 알게 될 것이다. 여러분들이 프로그램을 변환하는 것을 돕기 위한 프로그램이 존재한다. 이것은 TASM 문법을 AT&T 문법으로 변환하는 것이다. *ftp.oulu.fi/pub/msdos/programming/convert/ta2asv08.zipgas.doc* <ftp://ftp.oulu.fi/pub/msdos/programming/convert/ta2asv08.zipgas.doc> 이나 as.doc파일(GAS를 찾은 같은 곳 주위에 있을 것이다)은 그 문법을 기술한다. 다음의 FTP 디렉토리에 있다 *sunsite.unc.edu/pub/linux/GCC/* <ftp://sunsite.unc.edu/pub/linux/GCC/> *sunsite.doc.ic.ac.uk/packages/linux/sunsite.unc-mirror/GCC/* <ftp://sunsite.doc.ic.ac.uk/packages/linux/sunsite.unc-mirror/GCC/> (?) 다시한번 이야기하지만, 리눅스에는 괜찮은 예들이 들어있다; 아래의 linux/arch/i386의 다음 파일들을 보라: kernel/entry.S, kernel/head.S, boot/compressed/head.S, mathemu/*.S

2.2.3 2.2.3 제한된 16비트 모드

GAS 는 32비트 어셈블러이며 32비트 컴파일러를 제공한다는 것을 의미한다. GAS 는 현재 제한된 16 비트모드를 제공하는데, 그 16비트모드는 미리 예비된, 명령어들의 32비트 접두어로 이루어져 있으며, 따라서 32비트 CPU상의 16비트 모드에서 돌아가는 32비트 모드를 쓸 수 있다. 양모드에서 공히, 그것은 16비트 레지스터 사용이 가능하나, 16비트 어드레싱 은 제공하지 않는다. 모드사이를 전환하려면 "code16"과 "code32"의 지시자를 사용하라. 인라인 어셈블리에서의 asm("code16\n") 상태는 32비트 GCC로 하여금 리얼모드! 에서 돌아가는 코드를 만들것을 허용한다. 여러분들이 필요하다고 느낀다면 풀 16비트를 사용함으로써 마음껏 누려보자.

2.3 2.3 GASP

GASP 는 GAS의 전처리기이다. 이것은 GAS에 매크로와 몇몇 괜찮은 문법을 추가시킨다.

2.3.1 2.3.1 GASP를 어디서 찾을 것인가

나는 정확하게 모른다. GNU 사이트 (prep.ai.mit.edu & mirrors)를 보기바란다. 아마도 GAS와 같이 binutils 패키지에 같이 있을 것이다.

2.3.2 2.3.2 GASP는 어떠한 일을 하는가

난 아무 생각이 없다, 그러나 그것은 자체의 texinfo 문서가 따라올 것이다. 그래서 그것을 프린트 해서 보라, 아니면 .info 파일들을 브라우즈하기 바란다. 매크로 어셈블러의 규칙을 나에게 보내주면 좋겠다.

2.4 2.4 AS86

AS86은 Bruce Evans' C 컴파일러 부분중의 16비트, 32비트를 다 같이 제공하는 80x86 어셈블러이다. 그것은 최근의 인텔 문법을 가지고 있으며 as와 조금 어드레싱 모드를 달리한다.

2.4.1 2.4.1 AS86을 어디서 얻을 수 있는가

완전히 시대에 뒤떨어진 AS86은 HJLu에 의해 배포되며 바로 리눅스 커널을 컴 파일 할 수 있다. 이 패키지는 bin86(현재버젼 0.3)로 되어 있으며 Linux GCC 사이트에서 발견할 수 있다. 그러나 내가 보기에는 리눅스 컴파일링을 제외하고는 아무곳에도 사용되지 않고 있다. 이 버젼은 오로지 해킹된 미닉스 오브젝트 파일 포멧을 제공하며 32비트모드에서 는 조금의 버그가 있는데, 단지 리눅스를 컴파일만 하기 위해서라면 괜찮으리라. 가장 최근의 버젼은 FreeBSD 배포판과 같이 출시되었다. 나는 그것을 다음에서 구했다. *ftp.ibp.fr/pub/FreeBSD/packages-2.1/development/bcc-95.3.12.tgz* <ftp://ftp.ibp.fr/pub/FreeBSD/packages-2.1/development/bcc-95.3.12.tgz> 그러나 그 버젼이 이제 많이 발전했을 것이다. 여러개들 중에서 AS86도 이제 리눅스 GNU a.out 포맷을 지원한다. 그래서 여러분 들도 코드를 리눅스 프로그램에 링크를 시킬 수 있고, 데이타를 다루기 위해 GNU binutil 패키지의 보통의 툴을 사용할 수 있을 것이다. 이 버젼은 이전의 것들과 함께 아무런 손상없이 공동으로 존재할 수 있다. (아래의 질문 2.4.4를 보라). BCC 의 1995.3.12 이전의 버젼들이 실수한 이유는 32비트 모드 프로그래밍 시에 모든 세그먼트 팝과 푸쉬를 16비트로 처리함으로써 매우 번거롭게 된 데 있었다. 그 패치가 Tunes 프로젝트에 의해 다음에 발표되었다. *www.eleves.ens.fr:8080/home/rideau/Tunes/* <http://www.eleves.ens.fr:8080/home/rideau/Tunes/> 보조페이지는 files/tunes.0.0.0.25.src.tgz 이다. 풀린 서브디렉토리의 LLL/i386/ 그 패치는 또한 바로 다음에서 찾을 수 있다. *www.eleves.ens.fr:8080/home/rideau/files/as86.bcc.patch.gz* <http://www.eleves.ens.fr:8080/home/rideau/files/as86.bcc.patch.gz> Bruce Evans 는 이 패치를 받아들였는데, 가장 최근의 bcc 버젼은 이 패치 를 포함하고 있다. 도스 사용자를 위한 주의사항:

- 도스에서 컴파일 하려면 POSIX_HEADERS_MISSING 를 먼저 정의하라.
- bcc/as에 있어서 DJGPP를 사용하지 않는다면 mops.c의 mcall() 함수에서 "far"라고 이름 붙여진 것을 값을 바꿔야 한다. 왜냐하면 몇몇 도스 컴파일러에서는 "far"는 예약어이기 때문이다. bcc/ld 디렉토리로부터 typeconf.obj로 링크를 시켜야 한다는 것을 주의하라.
- bcc/ld에 있어서 아마도 a.out.h와 ar.h의 복사본이 필요할 것이다. DJGPP는 그것을 가지고 있는데, 다른 C 컴파일러 일 경우에는 다른 GCC(도스, 리눅스, VSTa, 등등)에서 몰래 살짝 해야 할 것이다. (^.%)
- bcc/ld에 있어서 BSD_A_OUT 매크로정의를 모든 파일에서 해야 할 필요가 있을 것이다. 그리고 STADARD_GNU_A_OUT 정의를 writebin.c에 해두고 리눅스 a.out.h 복사본을 사용가능한 도스 이름으로 변경한다.
- turns의 리눅스 a.out은 asm/a.out.h에 포함이 되어 있는데, 이것도 포함이 되 도록 해야 한다. 16비트 어셈블러에서는 asm/a.out.h를 24비트 보다 적은 비트 필드로 세트된 것에 대응하는 24비트 비트필드를 변경하여야 한다.
- 나는cc1을 시도해 보지 않았는데, 그러나 실제로 여러분들이 해보고 싶다면 cc1을 컴파일 할 수 있을 것이다. 그러나, 아마도 bcc frontend를 다시 적어야 하거나 바로 cc1을 사용해야 할것이다.왜냐하면 그것은 컴파일 시 cc1, as, ld를 작동시킬 때 fork()/exec()/wait() 트레블에 의존하기 때문이다.

- 전처리된 도스버젼은 다음에서 발견할 수 있다. *www.eleves.ens.fr:8080/home/rideau/files/asld86.zip* <http://www.eleves.ens.fr:8080/home/rideau/files/asld86.zip> 만일 그것들을 프리 컴파일러로 제컴파일을 하게 된다면 나에게 보내달라. 그러면 매우 고마울 것이다.

2.4.2 doc들을 어디서 찾을 수 있는가

doc들은 bcc 패키지에 포함이 되어 있다. 맨페이지도 또한 FreeBSD 사이트의 어 느 곳이던지 구할 수 있을 것이다. 의심이 간다면, 그들 소스에 가끔 괜찮은 doc들이 있다: 그리 잘 설명이 된 것은 아니지만, 프로그래밍 스타일은 깨끗하게 되어 있다. 관심이 있다면 Tunes 0.0.0.25 에서 어떻게 as86이 사용되었는 지를 살펴보라.

2.4.3 어떻게 어셈블러를 불러오는가 ?

bcc를 사용해서 .s 어셈블리 소스 파일을 GNU a.out .o 오브젝트 파일, .l 리스팅 파일로 변환하는 GNU 메이크 파일의 예를 하나 들어보자.

```
%.o %.l:        %.s
        bcc -3 -G -c -A-d -A-l -A$*.l -o $*.o $<
```

어떠한 리스팅 파일도 뭔하지 않는다면 "%.l", "-A-l", "-A$*.l"을 없애라. GNU a.out 이외의 것을 얻고자 한다면 bcc doc중에서 다른 제공하는 포맷에 대한 글을 보거나 GNU binutils 패키지의 objcopy 유틸리티를 사용할 수 있다.

2.4.4 이 새로운 버젼으로 리눅스를 컴파일 할수 없다면 어떻게 해야 하는가?

리누스는 메일로 왕성하게 활동하고 있다. 리눅스 a.out as86으로 리눅스를 컴 파일하는 나의 패치도 그것을 만들지 못한다. 이제, 이것들이 성공하지 못한다면: bin86 페키지안의 /usr/bin에 있는 여러분들의 as86을 가지고 있고, 더 좋은 as86을 /usr/local/libexec/i386/bcc/as에 가져다 놓아라. 아마도 이 "좋은" as86을 부를 필요가 더이상 없을 것이다.

2.5 다른 어셈블러들

2.5.1 Win32Forth 어셈블러

Win32Forth 는 Win32s, Win95, Win/NT에서 훌륭하게 돌아가는 프리 32비트 FORTH 시스템이다. 그것은 어셈블러에 통합된 프리 32비트 어셈블러(접두사 또는 접미사 문법)를 포 함하고 있다. 매크로 처리는 사려깊은 FORTH 언어의 full power에 의해 처리된다. 그러나 단지 입출력은 Win32For 그것 자체를 이용할 뿐이다. 다음에서 찾을 수 있다. *ftp.forth.org/pub/Forth/win32for/* <ftp://ftp.forth.org/pub/Forth/win32for/>

2.5.2 NASM

네트워크 상에서 벌어지고 있는 어셈블러 프로젝트는 아직 다른 어셈블러를 만들기 위해 노력하고 있는 중이다. 이것은 C에서 쓰여졌는데, 모든 알려진 문법들과 오브젝트 포맷들을 제공하는 데 충분한 모듈러가 될것이다. 현재 버젼은 조금의 매우 간단한 문법들과 평이한 바이너리 출력에 있어서는 잘돌아간다; 매크로 프로세싱에는 흥미가 없어보인다. 확실히 NASM은 이 하우투가 업데이트 되는 것 보다 빠르게 발전하고 있다; 그렇다고 NASM이 현재 요구되는 모든 특징들을 가지고 있을 것이라고 기대하 지는 말라. 적어도 여러분들이 그것의 계획이행을 도울 준비없이는... *www.dcs.warwick.ac.uk/~jules/nasm1.html* <http://www.dcs.warwick.ac.uk/~jules/nasm1.html>

2.5.3 Tunes

Tunes OS 프로젝트는 그들 자신의 어셈블러를 Scheme 언어를 사용함으로써 전체 개발 수행의 일부분으로 개발하고 있다. 그것은 아직 잘 돌아가지 않고 있으며 도움을 필요로 하고 있다. 이 어셈블러는 심볼릭 문법 트리를 처리하는 데, 이것은 어셈블리 문법의 기초로 동등하게 서비스 될 수 있으며, 디스어셈블러, 공통 어셈블러/컴파일러 back-end 등등의, 그리고 전정한 언어인 Scheme의 풀파워는 매크로처리/메트로처리에 있어서 타의 추종을 불허한다. *www.eleves.ens.fr:8080/home/rideau/Tunes/* <http://www.eleves.ens.fr:8080/home/rideau/Tunes/>

2.5.4 프리가 아니거나 32비트가 아닌 x86 어셈블러들

여러분들은 x86 어셈블리 프로그래밍의 기본으로서 그것들에 대해 좀더 찾아 볼 수 있다. Raymond Moon의 comp.lang.asm.x86을 위한 FAQ 를 찾아보기 바란 다. *www2.dgsys.com/~raymoon/faq/asmfaq.zip* <http://www2.dgsys.com/~raymoon/faq/asmfaq.zip>

제 3 절 META 프로그래밍/ 매크로 처리

여러분들은 이러한 작업을 위해서 적당한 툴을 사용할 수 있으며, 그것이 적당하지 않다면 어셈블리를 선택할 수 없다; C, OCAML, perl, Scheme 가 당신의 프로그래밍의 최선의 선택이 될 것이다. 그러나, 이러한 툴들이 기계상에서 충분히 쓸만한 제어기능을 주지 않는다면, 어셈블리는 유용하거나 필요할 것이다. 다른 경우에, 만일 여러분들이 안전한 프로그래밍, 편안한 수정 등등을 허락하는 인라인 확장에 의해 자동적으로 시스템의 매크로처리와 메타프로그래밍을 이해하 고자 한다면 그것은 한번 정의되는 재귀패턴을 허락할 것이며, 다중 시간을 재사용 할 것이다. "평이한" 어셈블러는 종종 작은 루틴을 C로 링크시킬 때 조차 충분히 않은때가 있다.

3.1 위에서 통합된 것이란 무엇을 뜻하는가

3.1.1 GCC

GCC 에서 인라인 어셈블리 코드를 사용한다면 그 속에서의 레지스터 규칙을 꼼꼼 하게 살펴보아야 한다. 최적화실행자는 항상 그것에 대해 알고 있으므로, 인라인 어셈블리 코드는 정확하지 않는 코드를 만들어 낸다. 그러면, 여러분들의 어셈블리를 CPP 매크로에 놓을 수 있으며, 모든 사람들도 그것을 C 함수/매크로로 사용할 수 있을 것이다. 인라인 함수는 매크로와 매우 많이 비슷한데, 그것은 가끔 사용을 하기 위해 깨끗하게 청소될 때가 있다. 그러한 경우에 코드가 복사가 될 수 있으니 주의해야 한다. 그래서 로컬 라벨("1:" 스타일의)에서만은 어셈블리 코드에서 정의되어 있어야 한다. 그렇다 할지라도, 매크로는 로컬이 아닌 정의라벨의 이름도 허용을 한다. 또한, 여러분들의 코드나 GCC의 조금의 버그는 레지스터 규정이 아마도 선언되 지 않았을 때, 인라인 함수를 어셈블리 코드로 사용할 때, 나타날 수 있을 것 이며 GCC를 혼동에 빠뜨린다. 마지막으로, C 언어는 그차체가 어셈블리 프로그래밍에 있어서 제법 괜찮은 추상화 라고 대우받고 있다. 어셈블링의 고충에서 많은 부분을 들 수도 있을 것이다. 그리고 함수로 인자를 레지스터를 통해 넘겨주는 몇몇 최적화는 어셈블리에서 그 함수를 부를 때 부적당 할 수도 있다. 적어도 여러분들이 어트리뷰 asmlinkage 에게 그것들을 주어야 한다. 리눅스 커널 소스의 부분을 참고 할 수 있다.

3.1.2 GAS

GAS는 절대로 매크로 기능을 포함하고 있지 않다. 그러나, GCC 와 건네지는 .S 파일들은 그것들이 GAS에게 건네지기 전에 CPP를 통 할 수도 있다. .s 파일은 하나를 생성시키고 바로 GAS에게로 건네진다. 다시한번 말하지만, 예를 볼려면 리눅스 소스들을 보라.

3.1.3 GASP

보통의 거의 모든 매크로어셈블리를 GAS에게 보탠다. texinfo doc를 보라.

3.1.4 AS86

이것은 간단한 매크로를 제공한다. 나는 doc들을 못찾았다. 이 소스들은 아주 간결하고 깔끔하다. 여러분들이 흥미를 가진다면 그것들을 쉽게 이해할 수 있을 것이다. 초보적인 것보다 더 많이 알고 싶다면 외부필터를 이용할 수 있다. (아래의 3.2 부분을 보라.)

3.1.5 다른 어셈블러들

Win32FORTH: CODE 와 END-CODE는 해설모드로부터 선택할 수 없는 매크로이다. 그래서 어셈블링 동안은 FORTH 단어의 모든 것들에 접근을 할 수 있을 것이다. NASM: 아직 매크로를 제공하지 않는다. 아래의 외부 필터 부분을 보라. TUNES: 이것도 아직 제공하지는 않는다. 그러나 Scheme언어는 기분내키는 데로의 메타 프로그래밍을 허용하는 진정한 고급언어이다.

3.1.6 외부 필터

어셈블러에서 어떤 매크로도 제공을 하던지, 또는 여러분들이 어떤 언어를 사용한다고 하더라도(C 조차!) 그 언어는 표현하는 데 충분치 않을 것이다. 여러분들은 Makefile 규칙으로 다음과 같이 파일을 외부필터를 통해 건네줄 수 있다.

```
%.s:    %.S other_dependencies
        $(FILTER) $(FILTER_OPTIONS) < $< > $@
```

3.1.7 CPP

CPP 는 표현력에 있어서 좀 약하지만 쉬운 것들에는 충분하다. 그것이 표준이라면 GCC에 의해 불리워 질 것이다. CPP의 제한된 점에서 보듯이, 오브젝트를 선언 할 수 없으며, 따라서 파괴자 (destructors)는 자동적으로 선언블럭의 마지막에서 콜된다. 그리고 데이타나 코드를 처리하기 위해 그것을 공통 선언을 할 수 없다. CPP는 C 컴파일러에 따라온다. GCC는 여러분들이 가져올 수 있는 자유로운 C 컴파일러이다.

3.1.8 M4

M4는 매크로처리에 있어서 탁월한 능력을 보여준다. 재귀적 표현, 질서정연한 규칙등에 있어서 괜찮아 보인다. CPP가 할 수 없는 모든 것들을 그것으로 할 수 있다. macro4th/This4th 를 아래에서 보라. *ftp.forth.org/pub/Forth/ in Reviewed/ ANS/* <ftp://ftp.forth.org/pub/Forth/inReviewed/ANS/> (?), 또는 Tunes 0.0.0.25 소스를 m4를 사용하는 진보한 매크로 처리의 예제로서 보는 것도 괜찮을 것이다. m4의 라이트버젼은 GNU m4 1.4 (이상)으로 구할 수 있다.

3.1.9 필터를 통한 매크로 처리

여러분들은 간단한 매크로 확장 필터를 쓸 수 있다. perl, awk, sed 등을 사용함으로써 가능하다. 이렇게 하는 것이 빠르며, 거의 모든 것을 제어 할 수 있을 것이다.

3.1.10 메타 프로그래밍

[생략]

제 4 절 컨벤션 부르기

4.1 4.1 Linux

4.1.1 GCC로 링크하기

32비트 인자는 32비트 near리턴 어드레스위로 스택상에 푸쉬된다. %ebp, %esi, %edi, %ebx 가 저장된다. %eax에 결과가 담기거나 %edx:%eax에 64비트 결과가 담긴다. GCC는 저장된 레지스터에 의해 호출되는 컨벤션을 변경할 수 있는 옵션을 가지고 있으며, 레지스터에 인자를 가지고 있으나 FPU를 생각하지 않는다. i386 인포 페이지를 체크하라. GCC가 이러한 표준 컨벤션을 제공할려면 함수를 위해 asmlinkage attribute를 선언해야 한다. (나는 어떻게 그것이 컨벤션을 불러서 변경하는 지를 알지 못한다.)

4.1.2 ELF vs a.out 문제들

어떤 C 컴파일러는 모든 심볼 전에 강조를 준비한다. 특별히, 리눅스 a.out GCC 는 리눅스 ELF GCC가 없을 동안 그러한 준비를 한다. 리눅스 소스 트리가 그것을 어떻게 관리하는 지를 보라. (linux/include/linux/linkage.h). 여러분들은 C->asm 이름 변경을 다음과 같은 기술을 삽입함으로써 뛰어넘을 수 있다. void foo asm("bar") (void); foo 함수를 확실히 하기 위해서는 어셈블리에서 bar를 호출해야 한다. binutils 패키지안의 objcopy 유틸리티는 a.out 목적파일은 ELF 목적파일로 만들수 있으며 반대의 경우도 가끔 가능 할 것이다.

4.1.3 직접적인 리눅스 시스템 콜

이것은 재명명할 수 없다. 왜냐하면 이것은 바뀌면 호환성이 없어지기 때문이다. 그리고 이것은 libc의 고정적인 것들과 확장된 것들을 방해한다. 정석으로 하자면, 리눅스 시스템 서비스 콜을 재명명하는 것은 libc를 통해서 한다. 이제, libc로 링크를 시키는 것을 원하지 않을 것이다. 아래에서 linux-eforth- 1.0c.tgz 을 보라. *ftp.forth.org/pub/Forth/Linux/* `<ftp://ftp.forth.org/pub/Forth/Linux/>` 리눅스 소스가 따라올 것이다. 어떻게 시스템 콜을 사용할 것인지를 설명하는 asm/unistd.h 헤더파일도 따라온다. 기본적으로 여러분들은 %eax에 __NR_syscallname 번호를 넣고, 파라메타를 %ebx, %ecx, %edx, %esi, %edi에 각각 넣고 int $0x80을 사용 할 수 있을 것이다. 결과는 %eax에 리턴되고, 에러시에는 libc가 errno를 세팅하는 것에 해당하는 결과를 음수로 %eax에 놓는다. 사용자 스텍은 건더리지 않는데, 시스템 콜이 불리워 지는 동안 별다는 것이 필 요하지 않을 것이다.

4.2 도스

최근의 도스 익스텐더들에는 도스서비스를 위한 인터페이스가 따라온다. 그에 대한 doc파일들을 읽어보라. 보통은 int 0x21을 흉내내는데, 따라서 리얼모드를 사용할 수 있게 된다. (나는 그것들이, 필요할 때 32비트 작동자를 사용하여 16비트 도스 서비스를 호출함으로써 작업을 할 수 있으리라고는 생각치 않는다.) DPMI에 관한 doc은 다음에서 찾을 수 있다. *ftp.oulu.fi/pub/msdos/programming/* `<ftp://ftp.oulu.fi/pub/msdos/programming/>` DJGPP 에는 자체의 (제한된) 교체된 libc가 따라온다. 이것은 리눅스에서 도스로 크로스컴파일이 가능하다. 그러나 현재 있는 패치는 a.out GCC용이며 최근의 ELF GCC는 패치가 필요없을 것이다. 만일 그것들이 새로운 패치를 필요로 한다면 난모르겠다..

4.3 여러분들의 OS

[어셈블리 프로그래머들이 많이 그것에 대해 이야기 하고 있는 것이다.]

4.3.1 부트 로더 코드와 32비트 모드로 들어가기

4.3.2 프로텍션의 기초

4.3.3 인터럽트 다루기

4.3.4 16비트 시스템 서비스를 사용하는 V86/R86 모드

4.3.5 이것들에 대한 정보를 어디서 얻을 것인가

[다른 문서에 이부분들에 대한 안내표시를 보태주기 바란다] 정보의 중요한 소스는 OS에 존재하고 있다. 많은 안내표시들이 아래의 WWW 페이지에 있다. *www.eleves.ens.fr:8080/home/rideau/Tunes/Review/OSes.html* <http://www.eleves.ens.fr:8080/home/rideau/Tunes/Review/OSes.html>

제 5 절 해야 할일

- 불완전한 부분 채우기
- 소프트웨어에 대한 더 많은 안내표지 보태기
- 실제 생활에 보탬이 되도록 제안된 각각의 솔루션들의 문법, 파워, 제한점 등을 설명할 수 있는 간단한 예제 보태기 끝.