2.3. Output/Input List

Output과 input은 "constraint" (variable)들의 쉼표로 구분된 리스트로 구성됩니다.

Constraint는 아래의 문자들과 몇가지 modifier들의 조합으로 허용되는 operand의 종류와 그 operand가 inline assembly에서 어떻게 사용되는지를 나타냅니다. 아래의 리스트는 완전하지 않습니다. Gcc 메뉴얼을 참조하세요.

2.3.1. Constraints

2.3.1.1. Basic constraints

'm'

Architecture가 일반적으로 지원하는 모든 addressing mode중 하나를 사용하는 memory operand.

'r'

범용 레지스터 operand.

'd', 'a', 'f', ...

레지스터를 적접 지정하는 constraint. Architecture마다 다른 문자들을 정의합니다. 'd', 'a', 'f'는 68000/68020에서 쓰는 문자들입니다.

'i'

정수 immediate operand. Assemble할 때가 되어야 알 수 있는 symbol값(주소)들도 해당됩니다.

'n'

값을 정확히 알고 있는 정수. Symbol값들은 해당되지 않습니다.

'I', 'J', 'K', ... 'P'

미리 정해진 범위안의 정수 immediate operand. 예를 들어 68000에서 'I'는 1에서 8사이의 immediate operand를 뜻하며 shift operation의 shift cound로 쓰입니다.

'E'

Compling machine과 target machine의 floating 표현 방식이 같을 때 immediate floating operand를 허용합니다.

'F'

Immediate floating operand.

'G', 'H'

Machine에 따라 정의되는 특정한 범위내의 immediate floating point operand.

'g'

범용 레지스터, memory, immediate operand 중 무엇이라도 됩니다.

'X'

어떠한 operand라도 허용합니다.

'p'

유효한 주소가 허용됩니다. Load address, push address등의 instruction에 쓰입니다.

'Q', 'R', 'S', ... 'U'

Machine-dependent.

2.3.1.2. i386 specific

'q'

a, b, c, or d register

'A'

a, or d register (for 64-bit ints)

'f'

Floating point register

't'

First (top of stack) floating point register

'u'

Second floating point register

'a'

a register

'b'

b register

'c'

c register

'd'

d register

'D'

di register

'S'

si register

'I'

Constant in range 0 to 31 (for 32bit shifts)

'J'

Constant in range 0 to 63 (for 64bit shifts)

'K'

0xff

'L'

0xffff

'M'

0, 1, 2, or 3 (shifts for lea instruction)

'N'

Constant in range 0 to 255 (for out instruction)

'G'

Standard 80387 floating point constant

2.3.1.3. Modifiers

Constraint modifier들은 그 변수가 어떻게 사용되는 지를 compiler에게 알려줍니다. 아래의 리스트는 완전하지 않습니다. Gcc 메뉴얼을 참조하세요.

'='

변수의 값이 바뀜을 나타냅니다. Output들에 대해서는 이 modifier가 반드시 지정되어 있어야 합니다.

'&'

Early clobber. 다음 절에서 자세히 설명하겠습니다.

2.3.2. Early clobber

```
#include <stdio.h>
```

```
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char **argv)
{
        int a, sum;

        a = atoi(argv[1]);

        __asm__ __volatile__(
                "movl   %1, %0          ₩n₩t"
                "addl   %2, %0          ₩n₩t"
                "addl   %3, %0          ₩n₩t"
                "addl   %4, %0          ₩n₩t"
                "addl   %5, %0          "
                : "=g" (sum)
                : "g" (a), "g" (a+1), "g" (a+2), "g" (a+3), "g" (a+4));

        printf("a=%d, sum=%d₩n", a, sum);
        return 0;
}
```

위의 프로그램을 컴파일하면

```
        .file   "early_clobber.c"
        .version        "01.01"
gcc2_compiled.:
.section        .rodata
.LC0:
        .string "a=%d, sum=%d₩n"
.text
        .align 4
.globl main
        .type    main,@function
main:
        pushl %esi
        pushl %ebx
        movl 16(%esp),%eax
        pushl $0
        pushl $10
        pushl $0
        pushl 4(%eax)
        call __strtol_internal
        movl %eax,%esi
        addl $16,%esp
        leal 1(%esi),%edx
        leal 2(%esi),%ebx
        leal 3(%esi),%ecx
        leal 4(%esi),%eax
#APP
        movl    %esi, %edx
        addl    %edx, %edx
        addl    %ebx, %edx
        addl    %ecx, %edx
        addl    %eax, %edx
#NO_APP
        pushl %edx
        pushl %esi
        pushl $.LC0
        call printf
        xorl %eax,%eax
        addl $12,%esp
        popl %ebx
        popl %esi
        ret
.Lfe1:
```

```
        .size    main,.Lfe1-main
        .ident  "GCC: (GNU) 2.95.4 20010902 (Debian prerelease)"
```

#APP와 #NOAPP 사이의 inline assembly에서 %0 (sum)에 %edx가 할당되었음을 알 수 있습니다. 그런데 #APP아래 두번째줄이 addl %edx, %edx로 %2 (a+1)도 %edx로 할당되었습니다. Gcc는 항상 모든 input 변수들이 다 사용된 후에 output 변수들이 쓰인다고 가정해서 input 변수와 output 변수를 같은 operand에 할당하기도 합니다. Input, output이 하나의 operand에 할당되고 위의 예처럼 input보다 output으로 먼저쓰이게 되면 틀린 결과가 나옵니다.

이런 경우에는 gcc에게 그 output 변수는 input의 값들이 모두 사용되기 전에 값이 바뀔 수 있다는 것을 알려주어야 합니다. 이것을 알려주기 위한 modifier가 early clobber modifier '&' 입니다. 위의 프로그램에서 output constraint "=g" (sum)을 "=&g" (sum)으로 바꾸고 다시 컴파일하면 다음과 같은 결과가 나옵니다.

```
        .file   "early_clobber.c"
        .version        "01.01"
gcc2_compiled.:
.section        .rodata
.LC0:
        .string "a=%d, sum=%dWn"
.text
        .align 4
.globl main
        .type    main,@function
main:
        pushl %edi
        pushl %esi
        pushl %ebx
        movl 20(%esp),%eax
        pushl $0
        pushl $10
        pushl $0
        pushl 4(%eax)
        call __strtol_internal
        movl %eax,%esi
        addl $16,%esp
        leal 1(%esi),%ebx
        leal 2(%esi),%ecx
        leal 3(%esi),%edx
        leal 4(%esi),%eax
#APP
        movl    %esi, %edi
        addl    %ebx, %edi
        addl    %ecx, %edi
        addl    %edx, %edi
        addl    %eax, %edi
#NO_APP
        movl %edi,%eax
        pushl %eax
        pushl %esi
        pushl $.LC0
        call printf
        xorl %eax,%eax
        addl $12,%esp
        popl %ebx
        popl %esi
        popl %edi
        ret
.Lfe1:
        .size    main,.Lfe1-main
        .ident  "GCC: (GNU) 2.95.4 20010902 (Debian prerelease)"
```

Output 변수는 %edi로 할당되었고 어떤 input도 겹치게 할당되지 않았음을 알 수 있습니다.

이전
Assembly

처음으로
위로

다음
Clobber list