2.4. Clobber list

Input이나 output으로 사용되지 않지만 어떤 레지스터를 inline assembly에서 임시로 사용할 때 clobber list에 그 레지스터를 적습니다. Clobber list에 있는 레지스터는 input, output에 있는 레지스터와 겹칠 수 없습니다. 즉, clobber list에 지정된 레지스터는 input, output을 위한 레지스터 할당에서 빠지게 됩니다.

만약 어떤 레지스터가 input으로 쓰이고 그 값이 바뀌지만 output으로 쓰이진 않는다면 clobber list에 그 레지스터를 정해줄 수 없습니다. 이런 경우엔 dummy 변수를 하나 선언한 후 output 인자로도 지정해주어야 합니다.

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

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

        a = atoi(argv[1]);
        b = atoi(argv[2]);

        __asm__ __volatile__(
                "movl   %1, %%eax       ₩n₩t"
                "addl   %2, %%eax       ₩n₩t"
                "mull   %%eax           "
                : "=&a" (r)
                : "g" (a), "g" (b));

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

두 정수를 입력받아 합의 제곱을 구하는 프로그램입니다. 위의 프로그램을 컴파일하면 다음과 같은 결과가 나옵니다.

```
        .file   "clobber_list.c"
        .version        "01.01"
gcc2_compiled.:
.section        .rodata
.LC0:
        .string "a=%d, b=%d, r=%d₩n"
.text
        .align 4
.globl main
        .type    main,@function
main:
        pushl %esi
        pushl %ebx
        movl 16(%esp),%ebx
        pushl $0
        pushl $10
        pushl $0
        pushl 4(%ebx)
        call __strtol_internal
        movl %eax,%esi
```

```
        addl $16,%esp
        pushl $0
        pushl $10
        pushl $0
        pushl 8(%ebx)
        call __strtol_internal
        movl %eax,%edx
        addl $16,%esp
#APP
        movl    %esi, %eax
        addl    %edx, %eax
        mull    %eax
#NO_APP
        pushl %eax
        pushl %edx
        pushl %esi
        pushl $.LC0
        call printf
        xorl %eax,%eax
        addl $16,%esp
        popl %ebx
        popl %esi
        ret
.Lfe1:
        .size    main,.Lfe1-main
        .ident  "GCC: (GNU) 2.95.4 20010902 (Debian prerelease)"
```

b가 %edx로 할당되었음을 알 수 있습니다. b는 input이므로 값이 변하지 않은 것으로 생각해 inline assembly후에 printf를 부를 때도 %edx의 값을 그대로 사용하는 것을 볼 수 있습니다. 위의 프로그램을 실행해보겠습니다.

```
$ ./a.out 4 6
a=4, b=0, r=100
```

결과값은 맞지만 b의 값이 0으로 출력됩니다. 이는 mull instruction이 결과값을 %edx와 %eax에 걸쳐 저장하기 때문입니다. 위쪽 결과값인 %edx가 0이 되지만 컴파일러는 그 값이 변하지 않았다고 생각하기 때문에 b의 값이 엉뚱하게 된 것 입니다. 이처럼 input, output 어느 것으로도 쓰이지 않지만 그 값이 변하는 경우에는 clobber list에 그 레지스터의 이름을 적어주면 됩니다.

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

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

        a = atoi(argv[1]);
        b = atoi(argv[2]);

        __asm__ __volatile__(
                "movl   %1, %%eax       ₩n₩t"
                "addl   %2, %%eax       ₩n₩t"
                "mull   %%eax           "
                : "=&a" (r)
                : "g" (a), "g" (b)
                : "edx" );

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

컴파일된 결과는 다음과 같습니다.

```
        .file   "clobber_list.c"
        .version        "01.01"
gcc2_compiled.:
.section        .rodata
.LC0:
        .string "a=%d, b=%d, r=%dWn"
.text
        .align 4
.globl main
        .type    main,@function
main:
        pushl %esi
        pushl %ebx
        movl 16(%esp),%ebx
        pushl $0
        pushl $10
        pushl $0
        pushl 4(%ebx)
        call __strtol_internal
        movl %eax,%esi
        addl $16,%esp
        pushl $0
        pushl $10
        pushl $0
        pushl 8(%ebx)
        call __strtol_internal
        movl %eax,%ecx
        addl $16,%esp
#APP
        movl    %esi, %eax
        addl    %ecx, %eax
        mull    %eax
#NO_APP
        pushl %eax
        pushl %ecx
        pushl %esi
        pushl $.LC0
        call printf
        xorl %eax,%eax
        addl $16,%esp
        popl %ebx
        popl %esi
        ret
.Lfe1:
        .size    main,.Lfe1-main
        .ident  "GCC: (GNU) 2.95.4 20010902 (Debian prerelease)"
```

b의 값을 ecx로 할당한 것을 볼 수 있습니다.

물론 결과값도 제대로 나옵니다.

```
$ ./a.out 6 4
a=6, b=4, r=100
```

위의 예에서 볼 수 있듯이 clobber list에는 레지스터의 이름을 직접 적습니다. 쓰이는 이름들은 다음과 같습니다.

```
i386 specific

 ah, al, ax, eax
 bh, bl, bx, ebx
 ch, cl, cx, ecx
```

```
dh, dl, dx, edx
si, esi
di, edi
```

(floating 레지스터들은 어떻게 지정하는 지 모르겠습니다. 아시는 분?)

Condition code의 값이 바뀜을 나타내는 cc가 있지만, ix86에선 필요하지 않습니다. 또, stack, frame pointer인 esp/sp, ebp/bp도 지정은 할 수 있지만 아무런 효력도 없습니다. esp, ebp의 값을 변경하는 경우엔 원래의 값으로 복원해야합니다.

이전
Output/Input List

처음으로
위로

다음
Applications