1.3. Inline assembly?

System programming을 하거나 성능이 중요한 프로그램의 tight loop을 짤 때 assembly를 써야하는 경우가 생깁니다. 즉, 프로그램의 대부분을 C나 C++등으로 만들고 고급언어로는 할 수 없거나, 그 부분을 assembly로 써서 속도 향상이 가능할 때 그 부분만을 assembly로 만들어서 C로 (C/C++ 모두 적용되지만 앞으로는 그냥 C라고만 하겠습니다) 만든 나머지 부분과 같이 쓰게 됩니다. C로 된 부분과 assembly로 된 부분을 같이 동작시키는 데는 크게 두 가지 방법이 있는데 하나는 assmbly로 쓴 부분을 독립된 함수로 만들어 따로 어셈블 한 후에 오브젝트 화일을 링크시키는 방법이 있고, 나머지 하나는 inline assembly를 쓰는 방법이 있습니다.

따로 assembly화일을 만들면 C의 함수호출방식에 맞추어 함수의 entry와 exit에서 인자들을 받고 return값을 돌려주는 부분만 신경쓰면 됩니다. gcc(gasm)외의 nasm 같은 다른 assembler를 쓸 수도 있으므로 어느정도 크기가 되는 부분을 assembly로 작성할 때는 이 방법을 쓰는 것이 좋습니다.

하지만 많은 경우에 전체 로직의 극히 일부분에서만 assembly가 필요하고 특히 compiler가 사용하지 못하는 processor의 특정한 기능을 쓰기위해 assembly를 쓸 때는 적게는 몇개, 많아도 이삼십개 정도의 instruction만을 assembly로 만들면 되는 경우가 대부분이고 이를 위해서 따로 함수를 만들어 링크하는 것은 번거로운데다가 자주 호출되는 경우라면 부가적인 function entry/exit 때문에 성능에도 좋지않습니다. 이런 경우에 inline assembly를 쓰게됩니다. 우선 어떤 것인지 감을 잡기 위해 예제를 보겠습니다.

```
void die(void)
{
        dump_processor();
        printk("halting the machine₩n");
        __asm__ __volatile__("cli; hlt;");
}
```

dump_processor, printk의 호출까지는 일반적인 C함수의 모습을 가지고 있습니다. __asm__으로 시작하는 부분이 inline assembly인데 괄호안의 문자열 "cli; hlt;"가 compile을 된 assembly 코드의 해당하는 자리에 그대로 출력이 되어 같이 어셈블됩니다. cli는 clear interrupt이고 hlt 는 halt입니다. 즉 cli; hlt;의 두 인스트럭션을 수행하게 되면 프로세서는 귀를 막고 잠들게 됩니다. 즉 위의 함수는 프로세서의 상태를 dump하고 멈출 것이라는 것을 출력한 다음에 기계를 멈춥니다.

위의 경우처럼 몇개의 특수한 instruction을 써야하는 경우에 매우 간편하게 쓸 수 있습니다. 이외에도 inline assembly를 시용하면 C의 변수를 assembly에서 쓸 수 있고 레지스터들을 어떻게 다룰지를 지정할 수도 있습니다. 차차 살펴보도록 하겠습니다.