시스템프로그래밍

- 개인 과제 -

이름: 천윤서

학번: 201520882

제출일: 2019년 11월 01일

1. 알고리즘에 대한 개요

1) 의사 코드

본 과제는 정수 배열을 받아서 받은 모든 정수에 대해 소인수분해를 진행하는 과제이다. 이번 과제에 대한 의사코드를 작성하면 아래와 같이 나타낼 수 있다.

```
N = 입력 받은 정수;
if(!(1<= N <= 10)) 예외처리;
키보드에서 받은 문자를 정수로 변환하고 배열에 파싱하여 집어넣는다;
if(정수의 개수가 N보다 많거나, 입력이 범위 내의 정수가 아니라면) 예외처리;
pflag = 1; //소인수 분해가 한번도 일어나지 않으면 소수로 판단하는 플래그.
for(i=0; i<정수의 개수; i++)
{
        multflag = 0;
        for(j = 2; j <= Array[i]; j++)
        {
                //입력이 예를 들어 4일 때 "*2*2" 형태가 아닌 "2*2"형태를 위한 플래그.
                while(Array[i] % j == 0)
                {
                        if(multflag == 0) multflag = 1;
                        else print('*');
                        Array[i] /= j; print(j); pflag = 0; //소인수 분해 실행.
                }
        }
        if(pflag == 0) print('p')
}
```

의사 코드는 C 스타일로 작성되어 있지만, 실제로 구현할 때는 논리 구조만 동일하고, 코드는 어셈블리 코드로 바뀔 것이다.

2) 실제 설계 모듈화와 구조

이번 과제에서는 어셈블리 코드의 가독성과 재사용성을 높이기 위해서 JSUB 과 RSUB 을 적극적으로 사용하여 모듈화를 적극 도입했다. 모듈들을 표로 정리하면 아래와 같다.

모듈명	입력값	출력값	설명
ATON	SLEN1, SBUF1	NBUF1	SLEN1 길이의 SBUF1 의 스트링을 NBUF 에 정수값으로 변환하여 넣는다.
NTOA	NBUF1	SBUF1	NBUF 의 정수값을 SBUF 에 스트링으로 변환하여 넣는다. 단, 오른쪽에서 왼쪽으로 읽는다.
REVERS	SBUF1	SBUF2	SBUF1 의 스트링의 좌우를 반전시킨다.
MOD	MBUF1, MBUF2	MBUF3	MBUF1 % MBUF2 를 진행하여 MBUF3 에 나머지를 반환한다. MBUF 1 의 값이 더 작다면 음수가 반환될 수도 있다.
FACT	NBUF1	-	NBUF 로 들어온 값을 소인수분해 시킨다.
PARSE	-	ARY	입력으로 들어온 스트링을 받아서 N 크기의 Array 에 스페이스바로 구분하여 정수로 변환 후 집어넣는다.
ISNUM	CBUF	-	CBUF 에 있는 값이 숫자가 아니라면 ERRFOM 으로 이동한다.
SCANN	-	N	STDIN 으로 들어온 문자열을 숫자로 바꿔서 N 에 넣는다. N 이 범위를 초과할 경우 에러를 발생.
ERFOM	-	-	숫자가 아닌 문자가 입력됐을 때 호출하는 예외처리
ERNUM	-	-	N 과 입력 수의 개수가 다르면 호출하는 예외처리.
EROVR	-	-	지정된 숫자 범위를 초과될 때 호출하는 예외처리.
PRINTE	ERSTR	-	ERROR OCCUR! 라는 경고문을 출력
PRINT2	SBUF2	-	SBUF2 의 내용을 출력한다.

각 변수들의 네이밍은 StringBUFfer, NumberBUFfer, ModBUFfer, ARraY 의 약자이다.

2. 실행 결과 스크린샷

1) 정상 실행

```
^Cchun@chun-virtual-machine:~/SicTools$ java -jar out/make/sictools.jar
5
50 9 71 4 10
2*5*5
3*3
P
2*2
2*5
```

]

```
^Cchun@chun-virtual-machine:~/SicTools$ java -jar out/make/sictools.jar
6
8 73 256 83 96 3
2*2*2
P
2*2*2*2*2*2*2*2
P
2*2*2*2*2*3
P
```

```
1
12
2*2*3
```

```
4
6 3 7 54
2*3
P
P
2*3*3*3
```

```
2
110 144
2*5*11
2*2*2*2*3*3
```

2) 비정상 실행

a) N(1~10)이나 소인수 분해할 숫자(2~999)의 값이 범위를 벗어난 경우.

```
112
ERROR OCCUR!
```

b) N 과 숫자의 개수가 일치하지 않는 경우(N 과 숫자의 수가 같지 않으면 무조건 에러).

```
^Cchun@chun-virtual-machine:~/SicTools$ java -jar out/make/sictools.jar
5
11 6 23 4 17 22
ERROR OCCUR!
```

3. 결과에 대한 설명

어셈블리로 짜인 코드의 큰 흐름은 아래의 순서를 따른다.

1. SCANN 함수를 통해서 STDIN 으로 들어온 값을 받아서 숫자로 변환하여 N 에 집어넣는다.

2. PARSE 함수를 통해서 STDIN 에 들어온 값을 N 과 개수가 일치하나 검사한 뒤, ARY 배열에 스페이스바로 구분하여 하나씩 집어넣는다.

3. ARY 배열의 값을 배열 포인터를 하나씩 증가시키면서 NBUF1 를 통해서 FACT 함수에서 소인수 분해를 하며, 결과를 즉시 출력한다.

나머지 함수들은 정수->문자열 변환 함수 등의 동작을 도와주는 함수거나 예외 처리 함수들이었다. 모든 함수들이 정상적으로 동작하였고, 결과가 정상적으로 출력되었다.

4. 구현사항 및 미구현 사항, 버그 및 개선점

1) 구현 및 미구현 사항

해당 프로그램은 스페이스바로 구분되는 N개의 숫자를 받아서 소인수분해를 하며, 소수인 경우에는 P라고 출력한다. 또한 숫자의 범위가 지정된 한계를 벗어나거나, N과 입력된 숫자의 수가 일치하지 않으면 즉시, 에러 문구를 출력하고 동작을 정지한다. 따라서 모든 과제의 요구사항이 구현되었다.

2) 버그 및 개선점

사실 첫 어셈블리어 프로그램이라 구현상의 미흡한 점이 많았다. 우선 변수들을 NBUF, SBUF등의 버퍼를 전역으로 선언하고 JSUB 안에서 이를 처리하는 식으로 함수를 구현했는데, 차라리 이번 과제에서 거의 사용하지 않았던 T-Register나 S-Register등에 값을 반환하도록 처리하였으면 더 효율적으로 메모리를 사용했을 것 같다.

또, EROVR, ERFOM, ERNUM등의 에러처리 함수를 따로 선언했지만, 이 셋은 전부 ERROR OCCUR!를 출력하고 정지하며 따라서 기능상의 차이점이 없다. 이는 String을 출력하는 함수를 전부 따로 구현했기 때문인데, 예를 들어서, Error를 출력하는 PRINT 함수와, 결과를 출력하는 PRINT 함수가 나누어져 있고, 이 때문에 여러 String의 함수들을 각각 정의해주는 것이 큰 부담으로 다가왔다. 아마, Indirect Addressing을 적절하게 활용하면 String의 주소값을 받아서 출력하는 범용적인 함수의 설계가 가능하지 않았을까 싶다.

5. 고찰

어셈블리어는 C언어에 비해 무척이나 난독성이 심하고 코드 길이가 길었다. 잘 읽히지 않았는데, 이 때문에 모듈 위주의 설계가 이런 상황에서 매우 큰 도움이 된다는 사실을 알았다.

모듈 위주의 설계 방식을 사용하기로 하면서, JSUB과 RSUB을 자주 활용했는데 이 때 함수 안에서 레지스터가 변경되는 것을 조심해야 한다는 점을 알았다. 예를 들어서, JSUB 안에서 또 JSUB을 호출하면, L 레지스터의 값이 덮어씌워지면서 돌아갈 주소를 잃어버린다. 이러한 문제를 해결하기 위해 L 레지스터의 값 등을 따로 저장하는 버퍼를 만들어야 했다.

C언어에서 무척 자연스럽게 사용하는 printf(%s), 정수-문자열 변환 등의 함수가 어셈블리어에서는 실제로 구현하는 것이 얼마나 힘든지 몸소 체감했다. 이를 통해 라이브러리 기능의 소중함에 대해 다시 한번 생각해보게 되었다.