시스템 프로그래밍 4주차 과제 보고서

이름 : 유성민

학번 : 201520908

학과 : 소프트웨어

학년 : 3

1) 알고리즘 개요

메인 루틴 CLAC, 두 자리 수 변환 서브 루틴 TWODIGIT, 입력으로 숫자가 아닌 것이 들어온 경우 처리하는 서브 루틴 NOTNUM, 스페이스 입력을 처리하는 서브루틴 ISSPACE

각각의 연산자를 처리하는 서브루틴 PLUSLOU(+ 처리), MINUSLOU(- 처리),

MULLOU(* 처리), DIVLOU(/ 처리)와 최종 결과 출력을 위한 서브 루틴 PRINT와 두 자리 수 출력을 위한 TWOPRINT, 마지막으로 멈추는 서브루틴 STOP으로 총 11개의 루틴으로 구성됩니다. 알고리즘의 흐름은 저는 알고리즘에서 S와 T 레지스터를 각각 다른 값으로 초기화 합니다. S레지스터와 T레지스터를 숫자 두개가 연속으로 들어오면 두 자리 수로 만들어야 하기 때문에 이 두 레지스터를 FLAG로 사용했기 때문입니다. 처음에 입력에 필요한 A 레지스터를 0으로 초기화 합니다. #0을 IMMEDIATE ADDRESSING을 사용했습니다. 그리고 입력 장치를 검사합니다. 입력이 제대로 되면, 입력으로 들어온 것을 읽고 아스키코드 '0' 0x30과 비교하여 숫자인지 아닌지를 판단합니다. 숫자인 경우 미리 지시어로 선언해둔 ARR 배열에 저장합니다. 그리고 처음 숫자가 들어왔을 때 S와 T레지스터를 같은 값으로 세팅합니다. 그리고 다시 메인 루틴으로 점프하여 입력을 받습니다. 만약 숫자가 연속으로 2번들어온 경우 두 자리 수 입력을 의미하므로 S와 T 레지스터가 같은 값으로 셋팅되어 있으면 두 자리 수임을 의미하여 두 자리 수 생성 루틴인 TWODIGIT로 점프합니다. TWODIGIT에서는 현재 입력 값 + 이전 입력 값 * 10을 하여 두 자리 수를 생성합니다. ARR[I]에는 십의자리 숫자가 ARR[I+1]에는 일의 자리 숫자가 들어갈 것이므로 배열에 저장 되어있는 값들을 꺼내어 레지스터 연산을 시킨 후에 다시 ARR[I]에 집어넣고 인덱스는 +1 시켜 다음 입력을 준비시킵니다. 세 자리 수는 없으므로 S와 T 레지스터를 각각 다른 값으로 세팅합니다. 다음으로 숫자가 아닌 입력이 들어오는 경우입니다. + - / * 공백 이렇게 5가지 경우에 대하여 NOTNUM 루틴에서 각각 입력을 처리하는 서브루틴으로 점프시킵니다. 공백을 처리할 때는 다음입력으로 숫자가 올 수 있으므로 인덱스를 +1 시켜놓고 S레지스터와 T레지스터를 다른 값으로 셋팅합니다. 그리고 다시

메인루틴으로 점프시킵니다. 플러스 루틴에서는 ARR[I-1]값과 ARR[I-2]값을 가져와서 더한 후에 ARR[I-2]에 저장합니다. 왜 ARR[I-1]과 ARR[I-2]인가하면, 위의 입력 루틴에서 인덱스를 증가시켰기 때문에 -1을 하고 시작해야 하기 때문입니다. - / *도 마찬가지의 루틴으로 이루어져 있습니다.(중복 내용 지양하라고 하셔서 쓰지는 않겠습니다.) 다음으로 출력 루틴입니다. 일단은 출력장치를 검사하고 최종 결과가 ARR[0]에 저장되 있을 것이므로 인덱스 레지스터 X를 0으로 초기화 시킵니다. 그리고 A 레지스터로 옮긴 후에 이것이 9보다 크다면 두 자리 수결과가 나온 것이므로 두 자리 수 출력 루틴으로 점프하고

아니라면 '0'을 더한 후에 출력 후 정지 루틴으로 점프합니다. 두 자리 수 출력 루틴에서는 최종결과/10을 하여 십의 자리 수를 임시 저장하고(일의 자리 수 출력을 위하여) 출력합니다. 그리고 최종결과 – 10*(임시저장한 10의자리수)를 하여 1의 자리 수를 출력시키고 정지 루틴으로 점프합니다

2) 실행 결과 스크린 샷

1)기본

```
20^Cseongminyoo@ubuntu:~/workspace/SicTools$ java -jar out/make/sictools.jar 20
08.asm
Gtk-Message: 03:57:39.346: Failed to load module "canberra-gtk-module"
5 4 + 2 -
7
```

결과 설명 : (5+4) – 2 = 7

2)문자열 최대

```
1^Cseongminyoo@ubuntu:~/workspace/SicTools$ java -jar out/make/sictools.jar 2015
08.asm
Gtk-Message: 03:09:01.324: Failed to load module "canberra-gtk-module"
1 1 1 1 1 + + + +
5
```

결과 설명 : (((((1 + 1) + 1) + 1) + 1) = 5

3) 결과 중간 음수

```
5^Cseongminyoo@ubuntu:~/workspace/SicTools$ java -jar out/make/sictools.jar 2015
08.asm
Gtk-Message: 03:09:57.092: Failed to load module "canberra-gtk-module"
1 2 - 3 +
2
```

결과 설명 : (1 – 2) + 3

두 자리 수 끼리 연산

```
34^Cseongminyoo@ubuntu:~/workspace/SicTools$ java -jar out/make/sictools.jar 201
08.asm
Gtk-Message: 03:52:23.208: Failed to load module "canberra-gtk-module"
10 50 + 40 -
20
```

결과 설명 : (10 + 50) – 40 = 20

사칙연산 모두 사용

```
7^Cseongminyoo@ubuntu:~/workspace/SicTools$ java -jar out/make/sictools.jar 2015
08.asm
Gtk-Message: 04:03:13.948: Failed to load module "canberra-gtk-module"
2 3 * 4 - 2 / 1 +
2
```

결과 설명: (((2*3) – 4) / 2)+1) = 2

최대 / 최소 숫자

```
Gtk-Message: 04:05:50.180: Failed to load module "canberra-gtk-module"
88 11 +
99
```

결과 설명: 88 + 11 = 99

```
99^Cseongminyoo@ubuntu:~/workspace/SicTools$ java -jar out/make/sictools.jar 201
08.asm
Gtk-Message: 04:06:21.398: Failed to load module "canberra-gtk-module"
1 1 -
0
```

결과 설명 1 -1 = 0

3) 구현사항 / 미구현사항 / 버그 / 개선점 등에 대한 설명

<구현사항>

1. 사칙연산 모두가능

2. 두 자리 수 포함 연산 가능

4. 입력길이제한포함 연산 가능

5. 제시된 최대 최소 숫자 연산 가능

6. 연산 도중 음수 처리 가능

<미 구현 사항>

소수점 연산 불가능 F 레지스터 미 사용

<버그 / 개선점>

소수 연산 가능하게 바꾸기

문자도 #사용하여 IMMEDIATE ADDRESSING MODE 가능하게 만들어 코드길이 감축, 메모리 레퍼런스 감소

4) 미 구현사항/버그에 대한 타당한 이유 제시

제가 처음에 연산은 무조건 A레지스터 이용이라는 편견이 머리에 박혀 있어서 F 레지스터를 사용하지 않고 알고리즘을 작성하였습니다. F레지스터를 사용함으로써 사용한 배열 크기도 맞춰야 복잡요소가 얽혀서 제 실력으로는 구현이 불가능 했습니다. 아직 실력이 많이 부족한 것을 깨닫고 더 열심히 공부에 정진하도록 하겠습니다...

5)고찰

교수님께서 수업도중 직접 써봐야 실력이 는다고 하셨는데 과제를 하면서 뼈저리게 느꼈습니다. SIC/XE 명령어들을 이론으로 보는 것보다 실제로 해보니 훨씬 어렵고 많은 시행착오 끝에 과제를 완성시켰습니다. 특히 배열 인덱싱하는 부분이 헷갈려서 실수를 많이 했던 것 같습니다. 과거 컴퓨터 과학자들이 HIGH LEVEL언어를 만들었는지 정말 이해가 잘 되었습니다. 확실히 어셈블리언어로 알고리즘을 작성하려고 하니 굉장히 어려웠으나

명령어에 대해 보다 정확한 이해를 할 수 있는 계기가 되어 좋은 경험이었다고 생각합니다.