어셈블리프로그램설계 및 실습 보고서

Term Project

학과: 컴퓨터정보공학부

담당교수: 이준환 교수님

실습분반: 화 6,7 목 5

학번: 2022202065

성명: 박나림

1. Introduction

본 프로젝트에서는 임의의 부동 소수점(Floating point number)으로 이루어진 배열을 이용하여, 두 가지의 종류로 정렬을 하고 메모리의 저장하는 코드를 구현하여 비교한다. 임의의 부동 소수점을 랜덤으로 생성시키는 코드는 주어지며 이 코드를 이용하여 10000 개의 숫자를 생성해 저장한다. 이 숫자들을 정규화 된 부동 소수점 배열로 보고 정렬을 한다. 이때 정렬은 병합 정렬(merge sort), 삽입 정렬(insertion sort)을 구현하도록 한다. 최종 정렬 결과는 라벨 final_result_series부터 시작하여 메모리에 저장한다. 또한 프로그램은 'MOV pc, #0 ;Program end' 이 라인으로 종료된다.

2. Background

1) 병합 정렬(merge sort)

병합 정렬, 또는 합병 정렬이라고도 불리는 정렬은 비교 기반 정렬 알고리즘이다. 최악, 최선, 평균 모두 O(n log n)의 시간 복잡도를 가지며, O(n)의 공간 복잡도를 가진다. 분할 정복 알고리즘을 사용하는데, 이러한 알고리즘은 그 상태로 해결할 수 없는 큰 문제를 작은 단위 문제들로 분할하여 해결하는 알고리즘을 말한다. 보통 재귀 함수(recursive function)를 통해 구현되기 때문에 재귀 호출을 사용하여 실행 속도가 늦어진다는 단점이 있다. 빠른 실행을 위해 스택, 큐 등의 자료구조를 이용하여 분할 정복법을 구현하기도 한다. n-way의 병합 정렬이라고 불리며, 일반적으로 쓰이는 하향식 2-way 병합 정렬의 순서는 다음과 같다.

먼저, 배열의 길이가 1 이하가 되면 이미 정렬된 것으로 판단한다. 그렇지 않은 경우 정렬되지 않은 배열을 절반으로 분할하여 비슷한 크기의 부분 배열로 나눈다. 그 다음, 각 부분 배열을 재귀적으로 병합 정렬시킨다. 각 부분 배열에서 정렬이 끝나면 다시 하나의 정렬된 배열로 합병시킨다. 이때 정렬된 배열 결과가 임시 배열에 저장되는 형식이다. 이러한 임시 배열에 저장된 결과를 다시 원래 배열에 복사하는 작업까지 끝마치면, 이 과정들을 재귀적으로 반복하여 병합 정렬을 마무리한다.

2) 삽입 정렬(insertion sort)

자료 배열의 모든 요소들을 앞에서부터 차례대로 이미 정렬된 배열 원소와 비교하여, 자신의 위치를 찾아서 알맞은 곳에 삽입함으로써 정렬을 완성하는 알고리즘을 삽입 정렬이라 한다. 삽입 정렬의 과정은 다음과 같다.

먼저, 배열의 두번째 원소부터 시작하여 첫번째 원소와 비교하여 삽입할 위치를 찾은 뒤 배열들을 뒤로 옮기고 지정한 자리에 자료를 삽입하여 정렬한다. 이런 식으로 두번째 원소는 첫번째 원소, 세번째 원소는 두번째와 첫번째 원소, 네번째 원소는 세번째, 두번째, 첫번째 원소들과 비교한다. 마지막 원소까지 이러한 과정을 차례대로 반복한다.

삽입 정렬은 구현이 비교적 간단하는 장점이 있지만 정렬하고자 하는 배열이 길어질수록 효율이 떨어진다는 단점이 있다. 따라서 이미 정렬된 최선의 경우는 이동 없이 1번의 비교씩만 이루어지므로 O(n)이지만, 역순으로 된 최악의 경우와 평균적인 시간 복잡도는 $O(n^2)$을 가진다.

3. Algorithm

1) 병합 정렬

병합 정렬을 구현할 때의 주요 함수 수도코드는 다음과 같다.

```
merge_sort(arr, first, last)
 if first < last
   mid=(first+last)/2
   merge_sort(arr, first, mid)
   merge_sort(arr, mid+1, last)
   merge(arr, first, mid, last)
merge(arr, first, mid, last)
 i=first; j=mid+1; k=0;
 while i<=mid && j<=last
   if arr[i] <= arr[j]
     sorted[k]=arr[i]; k=k+1; i=i+1;
   else sorted[k]=arr[j]; k=k+1; j=j+1;
 if i> mid
   while j <= last
     sorted[k]=arr[j]; k=k+1; j=j+1;
else while i<mid
```

```
 sorted[k]=arr[i]; k=k+1; i=i+1;

for i=first, k=0 to i<last; I,k+=1

 arr[i]=sorted[k]
```

위 함수에 따라서 라벨을 ms_init, merge_sort, ms_end, merge, cmp_1, cmp_2, while_1, sort_Arr_LE, sort_Arr_GT, while1_end, cmp_if, cmp_else, sort_Arr_LE_i, sort_Arr_LE_j, result_Arr, cmp_for, for_result_Arr, exit로 나누어 구현하였다.

먼저 ms_init에서 각 라벨을 LDR로 저장하고 first, last를 초기화 한 뒤 merge_sort로 이동한다. merge_sort에선 mid를 계산하고 각 함수들을 재귀적으로 호출하기 위해 BL명령어를 사용한다. 또한 각 register를 저장해놓기 위해서 PUSH, POP함수를 사용하였다. merge가 호출되면 I, j, k index를 초기화 시킨 뒤 각 cmp, while, sort 라벨을 이용해서 연산을 진행한다. 수도코드에 따라 같은 방식으로 연산을 구현하였으며, 연산이 다 완료된 후에는 result라벨로 넘어가서 최종 결과를 저장할 라벨을 불러온다. 이후 sorted에 저장된 결과들을 result라벨로 copy하여 저장한다. 이러한 과정이 끝나면 exit로 넘어가서 mov pc, #0과 함께 프로그램을 종료한다.

2) 삽입 정렬

삽입 정렬을 구현할 때의 주요 함수 수도코드는 다음과 같다.

```
for i=1 to i<n

key = arr[j]

  for j=i-1 to j>=0 (j--)

    if arr[j] > key

      arr[j+1]=arr[j]

    else break

  arr[j+1]=key
```

위 함수에 따라서 라벨을 is_init, for_1(out loop), for_2(in loop), end_for_2, end_for_1, exit로 나누어 구현하였다.

먼저 is_init에서 LDR을 통해 각각 랜덤 수 배열과 최종 결과 라벨들을 불러와 저장하였다. 또한 index를 나타낼 register도 초기화를 해주었다. 그 다음 for_1으로 들어가면, index와 배열의 사이즈(10000)를 비교하여 만약 사이즈를 넘어갈 시 end_for_1로 건너뛰게 하였다. 그렇지 않은 경우엔 arr[i]를 key에 저장한 뒤 j를 초기화하고 for_2로 간다. 이때 index에 따라 배열 숫자를 불러올 때는, 각 수의 크기가 4byte이므로 LDR key, [주소,

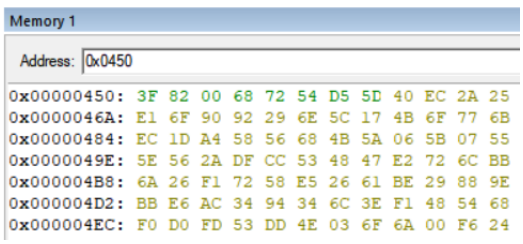
index, LSL #2]식으로 쓴다. LSL #2를 하면 index에 x4하여 수의 크기에 맞게 index를 설정할 수 있기 때문이다. for_2에서는 j의 index를 비교하면서 조건에 안 맞으면 바로 end_for_2로 건너뛰고, 그렇지 않으면 연산을 진행한다. break문도 마찬가지로 CMP로 비교하여 조건에 안 맞을 시 바로 끝나는 라벨로 넘어가게 한다. 반복 조건에 계속 맞는 동안은 해당 함수로 다시 돌아가도록 branch문을 쓴다. 전체적으로 out loop가 끝나는 end_for_1에서는 sort end로, 정렬된 결과를 copy하여 최종 결과 라벨에 다시 작성한다. 삽입 정렬은 중간 배열을 거치지 않고 바로 원래 랜덤 배열에서 정렬이 되므로, 결과를 저장하기 위해서는 다시 copy를 해야 한다. 이 과정까지 끝나면 exit로 가서 mov pc, #0과 함께 프로그램을 종료한다.

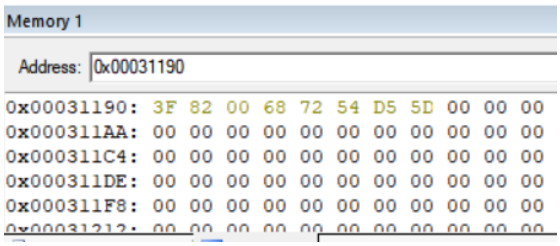
4. Performance & Result

1) 병합 정렬

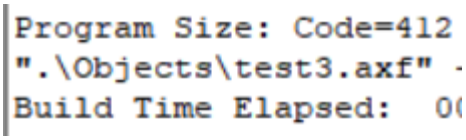
-memory

10000개의 랜덤 수를 저장하고 이를 임의의 배열에 분리하여 정렬하는 과정을 거치려하였으나, 재귀함수에서 문제가 발생하여 완성하지 못하였다.





-code size: 412



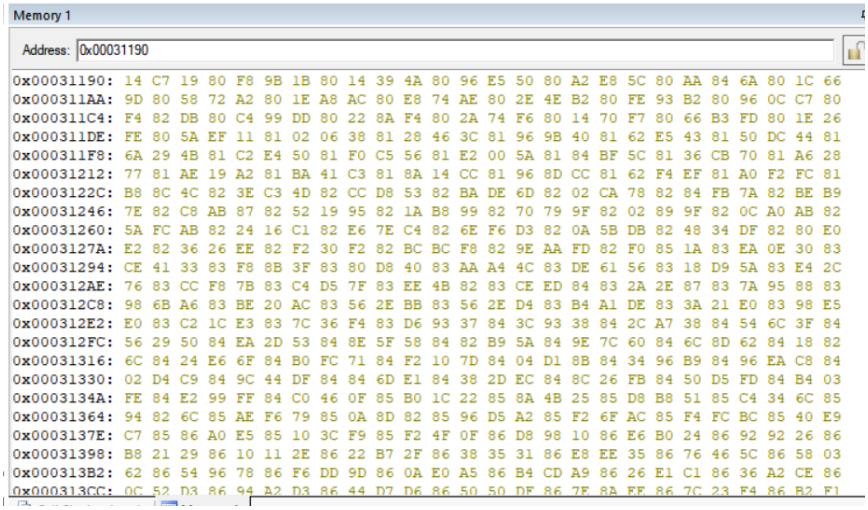
-state: 381

임의의 수를 저장하는 state가 270014로, 전체 state에서 이를 제외한 state는 위와 같이 나온다. 재귀함수 부분에서 프로그램이 더 동작하지 않아서 미완성인 상태이다.

2) 삽입 정렬

-memory

10000개의 랜덤 수를 생성한 다음에 이를 삽입 정렬하여 final_result_series라벨의 메모리에 저장한 결과이다. 첫 줄부터 4byte씩 끊어서 보면 8019C714, 801B9BF8, 804A3914...식으로 정렬이 된 것을 볼 수 있다.

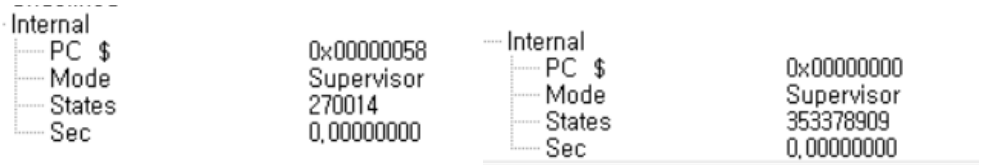


-code size: 220

```
Program Size: Code=220
".\Objects\test3.axf"
Build Time Elapsed:  0
```

-state: 353108895

임의의 수를 저장하는 state가 270014로, 전체 state에서 이를 제외한 state는 위와 같이 나온다.



-병합 정렬과 삽입 정렬의 결과 비교

병합 정렬이 미완성인 상태라 제대로 비교를 할 수는 없지만, 우선 code size로 볼 때 병합 정렬이 약 2배 크다는 것을 알 수 있었다. 실제로 코드를 구현할 때도 병합 정렬이 더 복잡하였었다. 하지만 state를 보면, 이 결과에서는 10000개로 실행해서 결과가 안 나왔지만 처음에 10개로 테스트했을 땐 삽입 정렬보다 병합 정렬이 state가 훨씬 작았었다. 이를 통해 병합 정렬이 삽입 정렬보다 구현하기에 복잡하고 code size도 크게 나오지만 속도는 훨씬 빠르고 효율적이라는 것을 알 수 있다.

5. consideration

코드를 작성한 뒤 실행을 할 때 excess write error등의 에러가 발생하였다. 이는 memory.ini 파일에서 MAP 범위를 수정해줌으로써 해결하였다. 이때도 한번에 너무 큰 범위로 설정하면 128 memory 초과라는 오류가 뜬다. 따라서 적절한 범위로 메모리 범위를 설정해주는 게 중요한 거 같다.

삽입 정렬을 구현할 때, 배열의 index를 설정하는 부분에서 처음에는 index만으로 구현을 하였다가 제대로 정렬이 안됐었다. 이에 메모리를 다시 살펴보니 임의의 부동소수점 배열은 각 수가 4byte씩이라는 것을 알고 정렬할 때도 index로 접근할 때 4칸씩 건너뛰면서 해야 된다는 점을 깨닫게 되었다. 따라서 LSL #2를 통해 4칸씩 index에 곱하여 배열에 접근하니 성공적으로 정렬될 수 있었다.

처음에 구현할 때에는 10개의 수로 테스트하여 각 라인마다 실행결과를 확인하였었는데, 마지막으로 10000개의 수로 검사를 할 때는 디버깅하는 데에 시행착오가 있었다. F5로 한번에 실행하니 무한로딩이 걸렸다. 확인해보니 프로그램이 종료된 후 다시 처음으로 돌아가서 실행을 하다 보니 끝나지 않았던 것이었다. 그래서 break point를 설정하고 확인하니 10000개의 정렬 결과를 확인할 수 있었다.

병합 정렬에서는 특히 여러 시행착오를 겪었었는데, 먼저 수도 코드를 따라 함수를 구현했을 때, PUSH를 사용하지 않으면 그 다음 줄부터 무한로딩이 걸리는 상황이 나타났었다. 그래서 PUSH로 register들을 저장하니 다음 라인으로 넘어갈 수 있었지만 전체적으로 결과가 제대로 저장이 되지 않았다. 처음 랜덤 수를 저장한 라벨을 살펴보니 두개의 수, 8byte까지만 접근이 됐었고 sorted 배열에서도 몇 개의 수만 저장이 된 상태였다. 최종 결과에서도 두 개의 수만 저장이 되고 정렬이 안된 상태였으니 아마 배열을 분리하는 과정에서 재귀적 호출이 잘못됐었던 것 같다. 이에 함수를 다시 구현해보고 코드를 살펴봤지만 결과에 큰 차이가 없었다. 아직 ARM에서 재귀 함수를 구현하는 것에 대해 미숙한 것 같다. 이 부분은 더 공부를 해봐야겠다고 느꼈다.

6. Reference

David Money Harris and Sarah L. Harris / Digital Design and Computer Architecture / Elsevier / 2007

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein / Introduction to Algorithms / MIT Press and McGraw-Hill / 2003