어셈블리프로그래밍설계 및 실습

Term project 결과보고서

실험제목: Sorting Floating pointer

실험일자: 2016년 11월 10일 (목요일)

제출일자: 2016년 11월 30일 (수요일)

학 과: 컴퓨터공학과

담당교수: 이형근 교수님

실습분반: 화 5 목 6, 7

학 번: 2013722095

성 명: 최 재 은

1. 제목 및 일정
   1. 제목

Sorting floating pointer

* 1. 일정

|  |  |  |  |
| --- | --- | --- | --- |
|  | Week(11월) | | |
| 일정 | 3주차 | 4주차 | 5주차 |
| Sorting 방식 선택  및 원리 이해 |  |  |  |
| 정수로 sorting 구현 |
| Floating pointer 구현 |
| 보고서 작성 |

1. Project specification
   1. Pseudo code

R0에 Random\_data lable의 주소를 load

R1에 Result\_data의 값을 load

Sp에 TEMP(stack pointer의 시작 주소)의 값을 load

R4에 r0값을 mov하고 -4byte함

R5에 250을 mov하고 lsl #4하여 1000으로 만든 뒤 r4와 더하여 끝 주소 저장

Oper 로 jump

**- POP\_P**

Sp가 가리키는 위치의 값을 r4에 load하고 sp위치를 4byte내림

Sp의 위치를 0x10000000과 비교-> 낮으면 SAVE\_DATA로 이동

Sp가 가리키는 위치의 값을 r5에 load하고 sp위치를 4byte내림

**-oper**

R10에 r4를 mov, R11에 r5를 mov

R11가 가진 값의 위치에 있는 실제 값을 r9에 load

R5 - R4를 R12에 저장 -> 그 값이 4이하이면 pop\_p로 이동

**-shift\_I**

R4값을 4 증가하고 R4가 가진 값의 위치의 실값을 R8에 load

R8을 LSR #31하여 SIGN BIT 추출하고 그를 R12에 MOV

R9을 LSR #31하여 SIGN BIT 추출하고 그를 R3에 MOV

R12, R3을 비교->R12가 더 크면 shift\_I로 이동 / 작으면 shift\_J로 이동

SIGN BIT이 1이면 one\_case\_I로 이동

**-one\_case\_I**

R8, R9을 비교-> R8이 더 작으면 shift\_I로 이동

**- shift\_J**

R5를 4 감소하고 그 값의 위치에 저장된 실값을 R7에 load

R7을 LSR #31하여 SIGN BIT을 추출하고 그를 R12에 MOV

R9을 LSR #31하여 SIGN BIT을 추출하고 그를 R3에 저장

R12, R3을 비교-> R12가 같거나 크면 shift\_J로 이동 / 더 작으면 Swap로 이동

SIGN BIT이 1이면 one\_case\_J로 이동

R7, R9을 비교 -> R7이 더 크면 shift\_J로 이동 / 같거나 작으면 Swap로 이동

**- one\_case\_J**

R7, R9을 비교 -> R7이 더 작으면 shift\_J로 이동

**- Swap**

R4, R5비교 -> R4가 같거나 크면 change\_end로 이동

R8값을 R5가 가진 값의 주소 위치에 저장

R7값을 R4가 가진 값의 주소 위치에 저장

Shift\_I로 이동

**- change\_end**

R8을 R11가 가진 값의 주소 위치에 저장

R9을 R4가 가진 값의 주소 위치에 저장

SP를 1증가시키고 그 위치에 R11값을 저장

SP를 1증가시키고 그 위치에 R4값을 저장

R5에 R4-4 값을 저장

R4에 R11값을 MOV

Oper로 이동

**- SAVE\_DATA**

R2에 0을 MOV

**- LOOP**

R2가 100인지 확인 -> 100이면 THE\_END로 이동

R0가 가진 값의 주소 위치의 실값을 R3에 LOAD하고 R0를 4증가

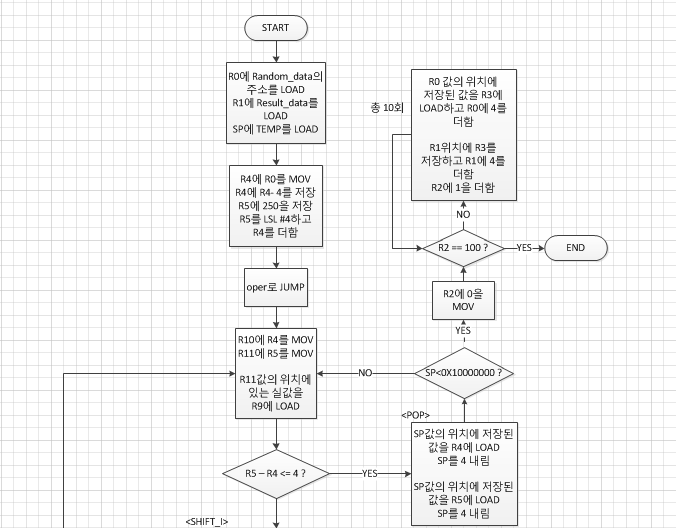
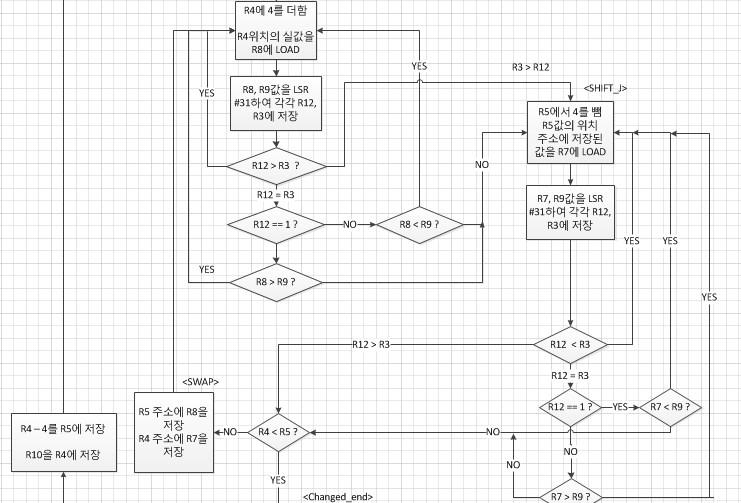
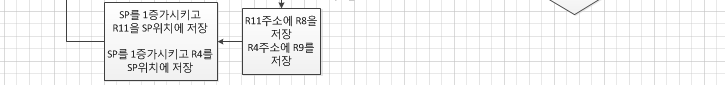
R3가 가진 값을 R1이 가진 값의 주소 위치에 저장하고 R1을 4증가

위의 내용을 총 10회 반복하고 LOOP로 이동

**- THE\_END**

PROGRAM COUNTER(PC)에 0을 MOV

* 1. Flow chart 작성



1. Algorithm
   1. Sorting algorithm

속도 면에서 가장 빠르다는 Quick sorting의 알고리즘을 사용하였습니다. 먼저 전체 데이터 블록의 시작과 끝 주소를 잡고, pivot을 데이터 블록의 오른쪽 끝으로 잡습니다. 이에 대해 추가적으로 블록의 시작 주소위치에 I를 두고 I가 pivot보다 크거나 같을 때까지 pivot을 향해 이동합니다. I가 pivot보다 크거나 같은 값을 찾게 되면, J를 pivot의 바로 왼쪽(4byte 아래) 으로 잡고 J가 pivot보다 작거나 같을 때까지 이동합니다. I가 pivot보다 크고 J가 pivot보다 작으면 I와 J를 swap하고 이 둘의 위치를 각각 +4 / -4 byte하여 이동시켜줍니다. 이러한 알고리즘으로 I와 J를 이동하다가 I가 J보다 큰 주소 위치에 있거나 같은 위치에 있을 경우 pivot과 I를 swap합니다.

Swap 이후 현재 블록의 끝 주소(지금까지 건들지 아니하였음) 와 현재 pivot의 주소를 stack에 순서대로 넣습니다. 그러면 pivot을 기준으로 오른쪽 블록이 stack에 들어가게 된 것이므로 pivot의 왼쪽 주소를 sorting하기 위하여 데이터 블록의 끝 주소 값을 현재 pivot의 바로 왼쪽(4byte 아래)으로 잡아주고 데이터 블록의 시작 주소를 초기의 값으로 reset합니다(블록이 분할 될 때마다 오른쪽 블록만 stack에 넣고 왼쪽 블록에 대한 정렬만 진행하기 위해서). 이러한 알고리즘으로 전체 데이터 블록을

작은 블록으로 세분화합니다. 이렇게 세분화하다가 블록의 끝 주소 값과 시작 주소

값의 차이가 4 이하가 된다면 (sorting한 이후이므로 남은 두 값 역시 정렬이 완료된 상태임) stack에서 시작, 끝 주소 순으로 pop하여 저장하고 이를 기준으로 pivot, I, J를 다시 setting하여 현재 블록에 대한 sorting을 진행합니다. 차후 발생하는 블록의 세분화는 위에서 설명한 알고리즘과 동일하며, pop했을 경우 stack pointer의 위치가 프로그램 초기에 설정했었던 주소 값(0x10000000)보다 낮아지면 모든 데이터에 대하여 정렬이 완료되었으므로 값을 저장하기 위해 SAVE\_DATA로 이동합니다. 이 때, 제일 처음 데이터를 받아왔을 때의 첫 주소를 시작으로 하여 데이터를 지정된 주소(0x60000000)에 저장합니다. 저장 시 loop는 10개를 저장하는 것을 1사이클로 하여 총 100번의 사이클을 돌며, 한 사이클을 돌 때마다 그 counter를 1씩 증가시켜 counter가 100이 되면 프로그램을 종료합니다. I, J와 pivot을 비교할 때는 sign bit을 추출하여 그를 통해서 1차적으로 크기를 비교하여 I를 더 움직일지 J를 이동할지, swap을 할지를 선택합니다. 위의 1차 비교 후 jump되지 않는다면 두 비교대상의 부호가 같다는 것이므로 이것이 음수인지 양수인지를 비교하여 해당되는 lable로 이동합니다. 이동 후에 그 값을 비교하는데 floating pointer를 표현하는 bit stream의 형태가 (sign bit + exponent + (mantissa))이므로 그 자체를 비교하는 것을 통해서 크기비교를 할 수 있습니다(exponent가 크면 무조건 크거나(+) 작습니다(-). 또한 exponent가 동일하다 하여도, mantissa를 비교함으로써 그 크기를 비교할 수 있습니다). 이러한 원리를 이용하여 그 크기를 비교하고 위에서 설명한 알고리즘에 따라서 I, J, Pivot을 이동하고 블록을 세분화 합니다.

1. Performance & result
   1. Performance

\* Code size : 

- Input 1



Score : 225427722600448

- Input 2



Score : 256997034917172

- Input 3



Score : 239827200847168

* 1. 결과 화면







- 첫 번째 사진이 I의 위치를 잡은 것이고, 두 번째 사진이 pivot의 위치를 잡은 사진입니다. 세 번째 사진을 보면 각각의 값이 register에 저장된 것을 확인할 수 있습니다.

- r8이 1/10001001/00101110110000000000000

R9이 1/10001001/01110010100000000000000인데, 이때 r8의 mantissa가 더 작기

때문에 실제 값은 I가 더 크다고 할 수 있습니다. 그래서 J를 확인하러 이동합니다.



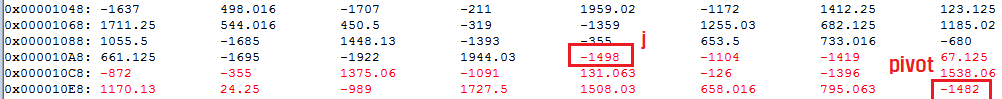


- 위의 사진은 r7에 J값을 넣은 것입니다.

- r7은 0/10001000/10001101100010000000000

R9이 1/10001001/01110010100000000000000인데, 이때 두 sign bit을 비교하면

r7이 양수로 pivot보다 크기 때문에 J를 다시 이동시킵니다.





- 이때의 R7은 이 1/10001001/01110110100000000000000으로, sign bit이 1로 음수이고, mantissa가 pivot보다 크기 때문에 실제 값은 더 작다고 볼 수 있습니다.

- 이때 I와 J를 swap하러 갑니다.





- 위의 사진을 보시면 I와 J가 swap되었음을 확인할 수 있습니다.

- 이후 I와 J를 각각 +4 / -4 byte하여 위의 알고리즘을 진행합니다.

- 아래의 사진은 순서대로 I와 pivot





- 아래의 사진은 swap한 결과





- I와 pivot을 swap하면 현재 블록의 끝 주소와 바뀐 pivot의 주소를 stack에 넣습니다.



- 이후 끝 주소를 pivot의 왼쪽으로 옮기고 왼쪽 블록에 대한 sorting을 위의 알고리즘을 이용하여 다시 시작합니다.

- 이렇게 pivot을 기준으로 블록을 세분화하며 정렬을 진행하다가 더 이상 왼쪽블럭을 정렬할 필요가 없으면 pop을 합니다.

- 아래의 사진은 pop하여 시작 주소와 끝주소를 resetting하는 사진입니다. 사진을 보시면 끝에서 두 개가 pop되어서 시작주소를 저장하는 r4와 끝 주소를 저장하는 r5에 들어갔음을 확인할 수 있습니다.



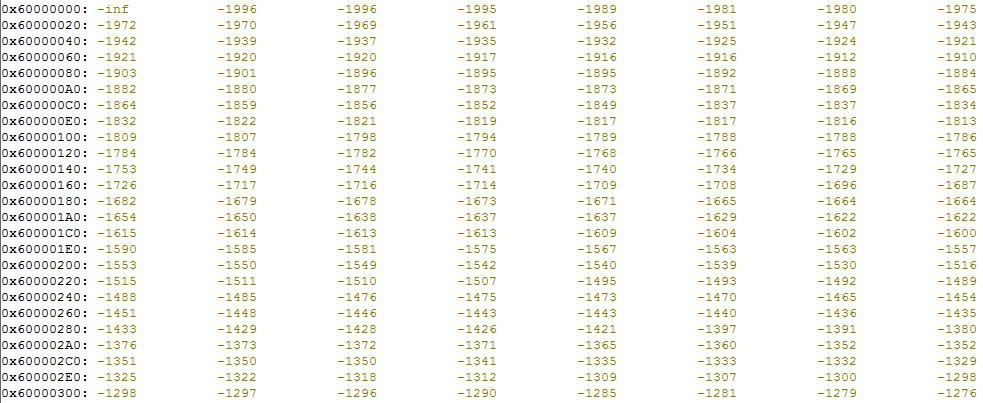


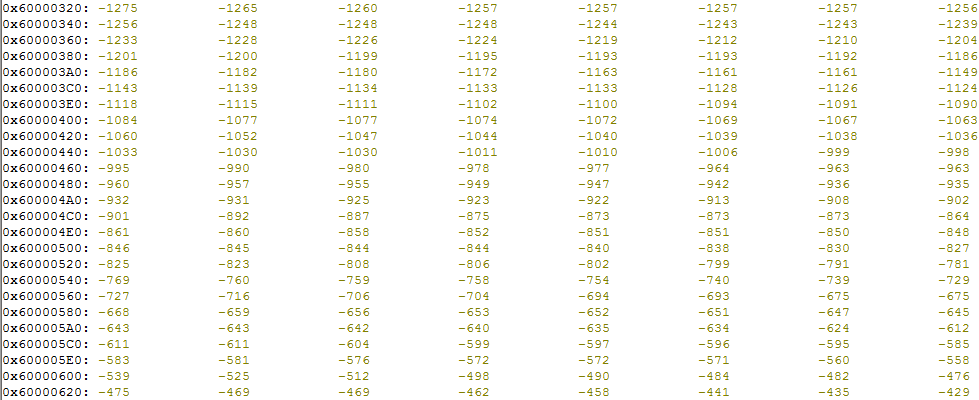
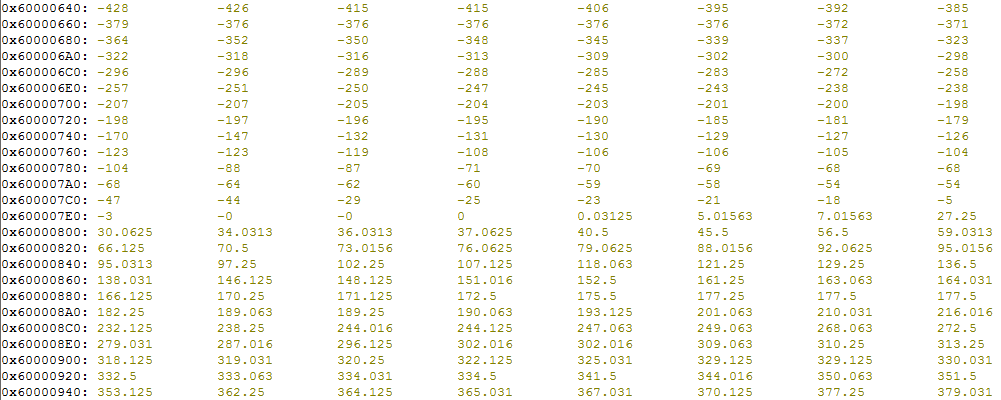
- 위의 시작, 끝 주소를 통해 새로운 블록을 형성하고 그 블록에 대한 정렬을 다시 진행합니다.

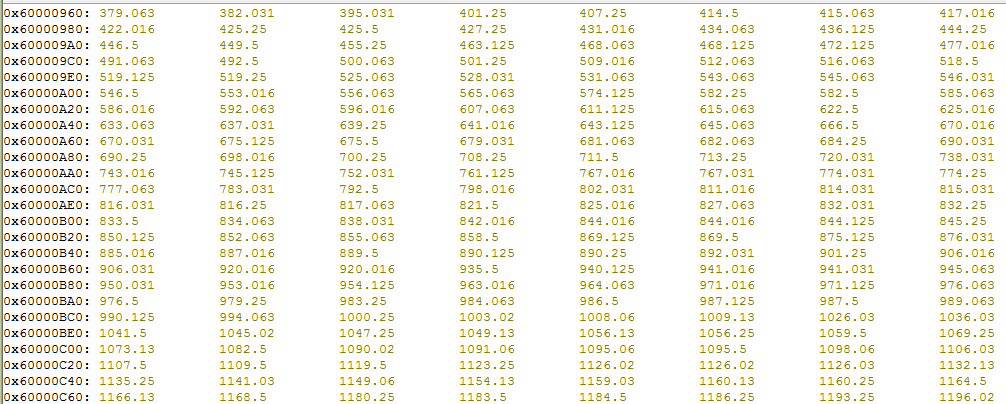
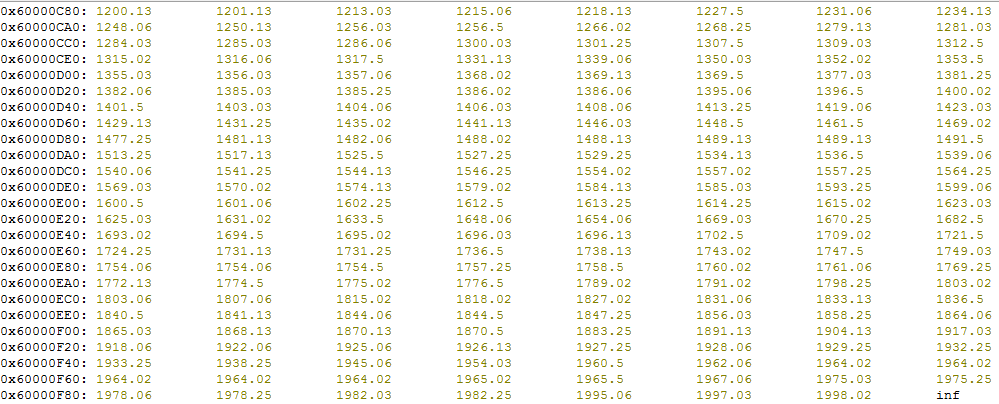
- 이렇게 모든 정렬을 진행하게 되면 stack pointer가 초기 설정 값인 0x10000000보다 낮아지는데 이때 모든 정렬이 완료된 것으로 보고 저장을 위해 탈출합니다.



- 이후 10개를 저장하는 것을 1사이클로하여 100번을 looping하여 값들을 저장합니다. 아래의 사진은 input3에 대한 정렬이 완료된 사진입니다.

- Input 3





1. Conclusion
   1. 고찰 및 결론

이번 프로젝트를 진행하기에 앞서 퀵 솔팅이 가장 빠르다는 이야기를 들은 이상, 퀵으로 짜지 않으면 좋은 성적은 안되겠구나…..싶어 퀵소트에 대한 공부부터 시작해야 했습니다. 수업시간에 조교님께서 언급하셨던 ‘재귀방식’에 대해서 좀 더 고찰해야 했습니다. 먼저 처음 프로젝트에 착수했을 때에, pivot을 잡고 I와 J가 움직이는데 I가 pivot보다 크거나 같은 값을 찾고 나서 J가 pivot보다 크면 다시 I를 움직이는 것으로 알고리즘을 이해해서 구현하는 데에 큰 장애가 되었습니다. 하지만 이후에 실습시간에 조교님들께서 제대로 된 알고리즘을 알려주시고 나서 무릎을 탁 치고는 바로 도서관으로 달려가 알고리즘을 수정하였습니다. 먼저 정수로 10개짜리를 정렬하는 것을 시작으로 floating pointer 10개, floating pointer 1000개로 순차적으로 검증하면서 코드를 수정, 작성해 나갔습니다. 재귀방식에 대해서는 데이터 구조 시간에 배웠던 ‘재귀함수방식을 스택을 사용하여 구현하는 방법’을 통해서 이해했으며, 이 또한 ‘무릎을 탁 치게 하는 교육 알고리즘이구나’하고 깨달았습니다. 그리고 코딩을 하면서 또 하나의 문제점이었던 ‘과연 솔팅을 끝냈다고 어떻게 확인할 것인가…..’를 초기 sp값을 설정해두고 sp가 그 초기값보다 낮아지면 탈출하여 저장을 시작하도록 함으로써 해결하였습니다. 이는 스택이 비어있는 상태는 프로그램이 시작할 때와 모든 값에 대한 솔팅이 끝난 경우뿐이기 때문입니다. 사실상 프로그램이 진행되면서 왼쪽 블록이 2칸짜리 or pivot이 해당 블록에서 가장 큰 값인 case에만 stack에 저장되는 블록의 시작과 끝 주소차가 4이하가 됩니다. 즉, pop으로 간다 하더라도 그 시작, 끝 주소의 차이가 4이하라면 다시 pop을 반복하므로 이를 종합해보면 stack pointer는 모든 정렬이 끝나기 전에는 최소 초기값의 위치를 갖게 됩니다. 그렇기 때문에 저장하러 가는 조건을 sp가 초기값보다 낮아질 때로 잡아주었습니다.