Andrew Scott
E155
Lab 4
October 5, 2015

Lab 4: Life of Pi

Assembly Code

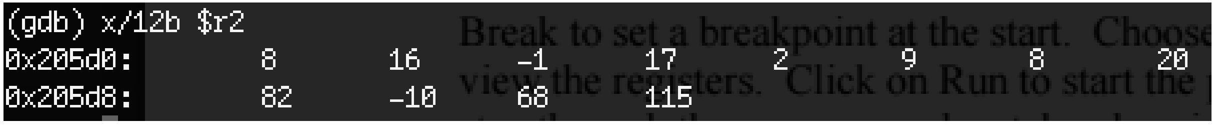
```
// lab4_as.s
// 4 October 2015 ascott@hmc.edu
// sort twelve numbers for E155 lab 4 using an insertion sort approach

.text
.global main
main:
    LDR r2, =array              // load base address of array into R2
    MOV r0, #1                  // int i = 1
loopi:
    CMP r0, #12                 // loop comparison, checking that i < 12
    BGE endi                    // loop comparison, checking that i < 12
    MOV r1, r0                  // int j = i
loopj:
    CMP r1, #0                  // loop comparison, checking that j > 0
    BLE endj                    // loop comparison, checking that j > 0
    LDRSB r3, [r2, r1]          // r3 = array[j]
    SUB r5, r1, #1              // r5 = j - 1
    LDRSB r4, [r2, r5]          // r4 = array[r5] (aka array[j - 1])
    CMP r3, r4                  // loop comparison, checking if array[j - 1] > array[j]
    BLE endj                    // loop comparison, checking if array[j - 1] > array[j]
    STRB r3, [r2, r5]           // array[j - 1] = r3
    STRB r4, [r2, r1]           // array[j] = r4 (swapped array[j], array[j - 1])
    SUB r1, r1, #1              // j--
    B loopj                     // continue the inner loop
endj:
    ADD r0, r0, #1              // i++
    B loopi                     // continue the outer loop
endi:
    BX LR                       // return from main
.data


array:                          // array of unsorted bytes
    .byte 0x08
    .byte 0x10
    .byte 0xFF
    .byte 0x11
    .byte 0X02
    .byte 0x09
    .byte 0x08
    .byte 0x14
    .byte 0x52
    .byte 0xF6
    .byte 0X44
    .byte 0x73
.end
```

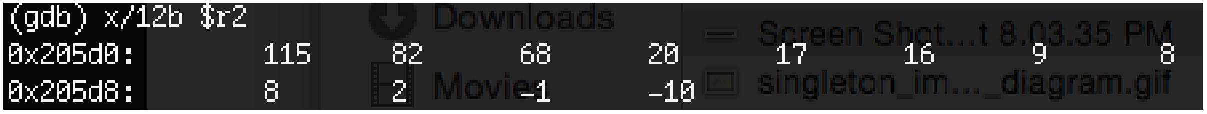
Testing

To test my sorting algorithm, I used three separate tests (by separate tests I am referring to the input array of unsorted bytes). I started with an array that had no particular order, but did have one number that repeated. This test was to ensure that the algorithm was generally working, given a random input, and that it could handle duplicates fine. Before running the algorithm, the array appeared as follows:



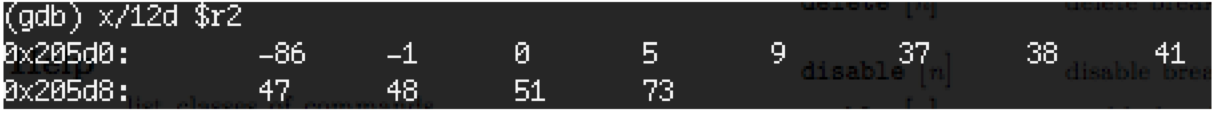
```
(gdb) x/12b $r2
0x205d0:        8       16      -1      17      2       9       8       20
0x205d8:        82      -10     68      115
```

After running the algorithm, the array was properly sorted, including the repeating 8 showing up next to itself as it was supposed to:



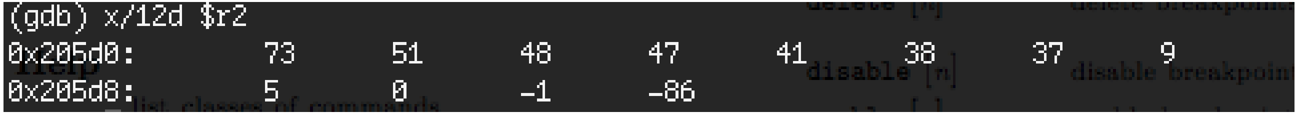
```
(gdb) x/12b $r2
0x205d0:        115     82      68      20      17      16      9       8
0x205d8:        8       2       -1      -10
```

The second test I ran was with an array of bytes that was in completely reverse order. This would ensure that the algorithm was considering every element of the array when doing the sorting and not skipping over any of them. Before running the algorithm, the array appeared as follows:



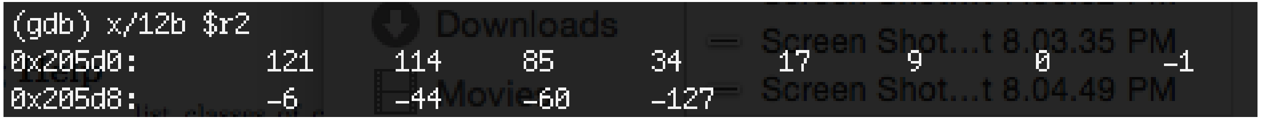
```
(gdb) x/12d $r2
0x205d0:        -86     -1      0       5       9       37      38      41
0x205d8:        47      48      51      73
```

After running the algorithm, the array was completely reversed and thus sorted:



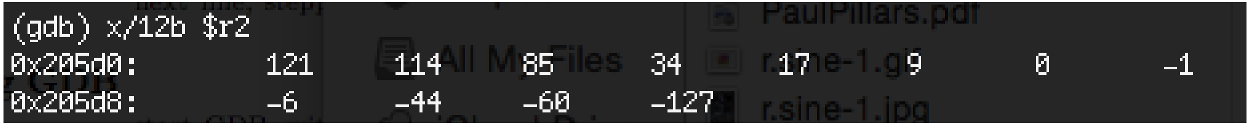
```
(gdb) x/12d $r2
0x205d0:        73      51      48      47      41      38      37      9
0x205d8:        5       0       -1      -86
```

Finally, I ran a test on an already sorted array to ensure that the algorithm would not make any moves when it was not necessary to do so. Before running the algorithm, the array appeared as follows:



```
(gdb) x/12b $r2
0x205d0:        121     114     85      34      17      9       0       -1
0x205d8:        -6      -44     -60     -127
```

After running the algorithm, the array was unchanged:



```
(gdb) x/12b $r2
0x205d0:        121     114     85      34      17      9       0       -1
0x205d8:        -6      -44     -60     -127
```

Time

I spent approximately 7 hours on this lab.