情報学郡実験第 2 アセンブリ言語を用いたソートアルゴリズムの実装

1250372 三上柊

2022 年 11 月 5 日

第 0 章

本課題に関して

本課題で作成するアルゴリズムが満たすべき条件は,

- 指定された主記憶領域のダブルワード列を**昇順**に整列すること
- 整列対象の先頭番地が ebx に，ダブルワードの個数が ecx に格納されて sort に渡される
- sort は ebx が指す番地から始まる ecx 個のダブルワードからなる領域の中身を書き換える
- 整列対象のダブルワードは 0 以上 2^{31} 未満である
- ダブルワードの個数は 1 個以上 30 万個以下である
- sort.s には，サブルーチン sort の定義（及び，整列対象以外にこのサブルーチンが必要とするデータ領域等の定義）のみ記述すること。特に，sort.s の中でラベル _start を定義してはいけない。
- ラベル sort を global 宣言すること。
- sort は，標準出力に何も出力しないこと。
- sort の呼び出し前と復帰後で汎用レジスタの値が変化しないようにすること。

である．これを満たすよう作成したソートアルゴリズム（sort.s）に対して実験を行う．
また，すべての実験を行うに際して使用したコンピュータを以下に示しておく．

Listing 1　使用したコンピュータとソフトウェア

```
$ uname -a
Linux KUT20VLIN -462 5.4.0 -70 - generic #78~18.04.1 - Ubuntu SMP Sat Mar 20
    14:10:07 UTC
2021 x86_64 x86_64 x86_64 GNU/ Linux
$ nasm --version
NASM version 2.13.02
$ ld --version
GNU ld (GNU Binutils for Ubuntu ) 2.30
```

第 1 章

コンパイラを使用せず，アセンブリ，機械語のみでのソートアルゴリズムの記述

1.1 目的

コンパイラとは，高級言語を「機械がすぐに実行できる形」に翻訳するものである．本課題は，これを使用せず直接「機械がすぐに実行できる形」の言語（今回はアセンブリ言語）で記述できることを示すことである．

1.2 実験方法

第 0 章に示した条件を満たすよう，高級言語で記述したものと同じ挙動をするようなソートアルゴリズムをアセンブリ言語で記述する．これを実行可能ファイルに変換し，期待された結果が出力されるか確認する．
実行コマンドは以下であり．print_eax.s，test_sort.s，sort.s に対して操作を行う．

Listing 1.1 実行コマンド

```
$ nasm -felf print_eax.s
$ nasm -felf test_sort.s
$ nasm -felf sort.s
$ ld -m elf_i386 print_eax.o test_sort.o sort.o
$ ./a.out
```

1.3 実験結果

a.out を実行した結果，右のような結果が得られた．

入力	1	3	5	7	9	2	4	6	8	0	1	2
期待出力	0	1	1	2	2	3	4	5	6	7	8	9
出力	0	1	1	2	2	3	4	5	6	7	8	9

1.4 考察

実験結果から，「1 以上 30 万個以下の 0 以上 2^{31} 未満のダブルワード」に関してはアセンブリ言語でソートアルゴリズムが記述可能であると言える．この条件を満たさない場合のソートアルゴリズムも記述可能であるかは定かでない．

第 2 章

高級言語との比較

2.1 目的

高級言語とは，人間が理解しやすい命令や規則が設定されたプログラミング言語（抽象度が低級言語よりも高い言語）のことであり、低級言語はコンピュータが実行できる形の言語である．これらで同じアルゴリズムを記述した時，両者にどのような（ソースコード記述量，コードの複雑さ等）差が現れるかを検証するのが本実験の目的である．今回は高級言語に Java 言語，低級言語にアセンブリ言語を使用する．

2.2 実験方法

実際にソートアルゴリズムを高級言語，低級言語で記述する．その中で同処理をしている部分に着目し，行数，コードの複雑さを比較する．コードの複雑さは同じ処理をするのに必要な手順の量によって判断する．

2.3 実験結果

実際に記述，比較した結果，2.1 と 2.2 という 2 つの表が得られた．

2.4 考察

表 2.1 を参照すると，同じ処理をするのに必要なコード量はアセンブリ言語のほうが増加している．

またコードの複雑さに関して，表 2.2 を参照すると，同じ処理を行うのにかかる手順はアセンブリ言語のほうが多いため，コードの複雑さにも差が現れている． 選択ソートアルゴリズムに関してアセンブリ言語の方がコードが長くなるのは，Java 言語では i 番目の配列の値とその他の値の比較が 1 行で行えるのに対し，アセンブリ言語では同じ処理を行うのに「指定したアドレス番地の値を一度レジスタに保存」と「保存されたレジスタの値とその他の値を比較」という 2 手順が必要であり，これらは 2 行に分けて書かれることが多いためだと考察できる．データの入れ替えも同様である．

今回の実験により，「選択ソートアルゴリズム」を記述する際は高級言語を使用ことが好ましいことが明らかになったが，その他のソート（ソートに限らないが）アルゴリズムも高級言語を使用するのが好ましいのかは定かでない．

表 2.1　コード量の比較

Listing 2.1　Java 言語での記述例

```
1  int max_index = 0;
2  int max = 0;
3  for (int i = data.length - 1; i > 0; i--) {
4      max = data[0];
5      max_index = 0;
6      for (int j = 1; j <= i; j++) {
7          if (data[j] >= max) {
8              max = data[j];
9              max_index = j;
10         }
11     }
12     int m = data[max_index];
13     data[max_index] = data[i];
14     data[i] = m;
15 }
```

Listing 2.2　アセンブリ言語での記述例

```
1  loop0:
2    cmp ecx, 0
3    jle endp
4    mov edx, [ebx]
5    mov eax, 0
6    mov edi, 1
7    loop1:
8      cmp edi, ecx
9      jg loop0l
10     mov esi, [ebx + edi*4]
11     cmp esi, edx
12     jge then
13     jmp endif
14     then:
15       mov edx, [ebx + edi*4]
16       mov eax, edi
17     endif:
18       inc edi
19       jmp loop1
20   loop0l:
21     mov esi, [ebx + eax*4]
22     mov edi, [ebx + ecx*4]
23     mov [ebx + eax*4], edi
24     mov [ebx + ecx*4], esi
25     dec ecx
26     jmp loop0
```

表 2.2　ソースコードの同処理手順行対応表

処理内容	Java	アセンブリ
ループ 2 内条件比較	7	10 - 11
最大値の比較，更新	7 - 11	10 - 17
data[max_index] と data[i] の入れ替え	12 - 14	21 - 24

第 3 章

低級言語で記述した際も実行時間は理論的計算量に従うか

3.1 目的

本実験の目的は，低級言語で記述したプログラムも，高級言語と同様に実行時間が理論的計算量に従うことを示すことである．今回はアセンブリ言語で記述した選択ソートアルゴリズムを用いて実験を行う．

3.2 実験方法

1.1 のコマンドで生成した a.out に対し，Linux 標準コマンドである time を実行し実行時間を計測する．計測は各ダブルワード列に対して 3 回行い，その平均を取る．
計測された時間をグラフ化する．ダブルワード列の生成，time コマンドの実行の仕方は以下に示す．

Listing 3.1 ダブルワード列の生成（N=データ数)

```
    section .data
data: times N dd 0
ndata: equ ($ - data) / 4
```

Listing 3.2 time の実行 (real が実行時間)

```
$ time ./a.out
-- 実行結果（略）--
real 0m0 .002s
user 0m0 .001s
sys 0m0 .000s
```

今回はダブルワードの個数のパターンを 1,10,100,1000 個までの 4 つと，1 万個から 10 万個まで 1 万刻みでの 10 個，計 14 パターンとする．

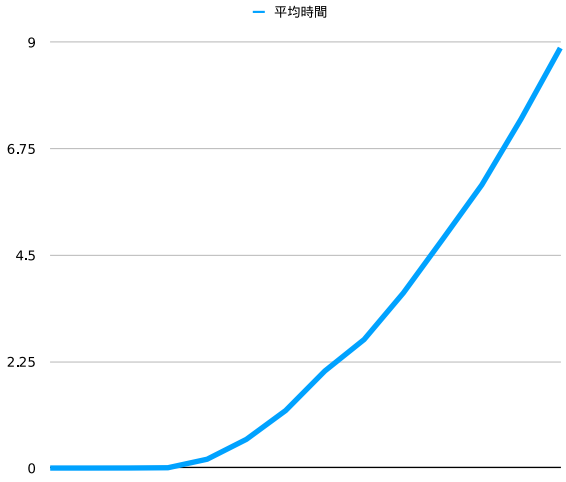
3.3 実験結果

得られた結果を以下に示す.

図 3.1 計測結果

データ個数	実行時間
1	0.001
10	0.001
100	0.002
1,000	0.0073
10,000	0.1873
20,000	0.6077
30,000	1.2164
40,000	2.0527
50,000	2.7173
60,000	3.7017
70,000	4.8357
80,000	5.9860
90,000	7.3787
100,000	8.8790

図 3.2 データの個数と実行時間の関係



3.4 考察

実験結果の 3.2 より，データの個数と実行時間の関係は n^2 に従っていると考えられる．これは理論的時間計算量である $O(n^2)$ の n^2 と一致するため，アセンブリ言語で直接コードを記述した際もその実行時間は理論的時間計算量に従うと言える． これは，高級言語と低級言語でコードの量に差はあるが，コンピューター内部での処理は同一であるからだろうと考えられる．