実験レポート４

1270360:稗田隼也

1月4日

1　目的

私達が普段使っている高級言語を使わずにアセンブリ言語を使うことでプログラミング上で起きていることのイメージを養うとともにアセンブリ言語や機械語で高級言語と同じ動作が可能かどうかをコンパイラを使わず、アセンブリ言語や機械語によって整列アルゴリズムを直接記述することで示すことを目的にする。また疑似コードや高級言語での実装と比べ、大きく異なる点があるかどうかを明らかにしコードの複雑さやコード量がどの程度増すかを明らかにしてアセンブリ言語や機械語で直接記述した場合も、実行時間は理論的計算量に従うことを示す。そして高級言語での実装と比べて計算時間に差があるかどうかを明らかにし上記の事柄を総合して，アセンブリ言語で整列アルゴリズムを直接記述することに利点があるかどうかを明らかにしていく。

2　方法

目的を達成するためにアセンブリ言語で記述した以下のソートアルゴリズムについて考察していく。

Listing 1　switch

```
1  section .text
2          global  _start
3  _start:
4    mov ebx,0
5    mov ecx,data
6    mov edx,ndata
7    sub edx,1
8    mov esp,0
9    mov ebp,0
10   mov esi,ndata
```

```
11
12 loops:  mov esp,0    ;ru-pu 0
13
14   mov ecx,data     ;ecxに場所
15   add ebp,1    ;何重目か
16   cmp ebp,esi    ;n kai
17   je endif
18 loop:
19
20   mov eax,[ecx]      ;a
21   mov ebx,[ecx+4]    ;b
22   cmp eax,ebx    ;比較
23   ja else      ;b<aならelse
24
25   add ecx,4    ;次
26   add esp,1    ;ru-pu count
27
28   cmp edx,esp    ;比較
29   je loops     ;回数と個数が同じならloopsへ
30
31           jmp loop     ;繰り返し
32
33 else:
34   mov [ecx],ebx    ;ecxの場所にb
35   mov [ecx+4],eax          ;ecx+4にa
36         add ecx,4    ;次
37   add esp,1    ;count
38   cmp edx,esp    ;比較
39   je loops   ;回数と個数が同じならloopsへ
40
41   jmp loop     ;繰り返し
42
43
44
45
46 endif:
```

```
47          mov ebx,0
48    mov eax,1
49          int 0x80
50
51    section .data
52
53
54
55  data: dd 12,134,11,56
56
57  ndata equ ($-data)/4
```

また、高級言語との相違点やコードの複雑性、計算時間等を比べるために高級言語のソートアルゴリズムも用意し比較していく。また、計算量についてある変数を用意し処理が行われるたびに1回ずつ更新していく方法で計算する。具体的にはアセンブリでは edi に処理をするごとに加算していくようにする。高級言語も同様だ。計算量について調べるためにデータ数を3パターンほど用意する。データ数の違う例で計算量を求めることで法則性から計算量を調べる。それぞれ、12,134,11,56 と 13,14,15,11,154 と 15,11,2,15,1,19 について議論する。また、高級言語についても計測を行い、違いを調べる。

3 結果

まずはじめに、実際にアセンブリでバブルソートのソートアルゴリズムを記述したファイルを作成し、ターミナルで a.out という実行ファイルにして./a.out とコマンドを打つことで実行し、data にある値が順番どおりになっていたことから直接記述できるといえる。また、確認方法として echo で一つ一つ出力していった。その結果、12,134,11,56 が 11,12,56,134 になったことが確認できた。

次に高級言語との違いについて検討していく。以下が c++という高級言語で書かれたソートアルゴリズムだ。

Listing 2　switch

```
1  #include <iostream>
2  using namespace std;
3  int main(void){
4      int s[]={12,134,11,56};
5      int n=4;
6      for(int i=0;i<n;i++){
7          for(int t=0;t<n-1;t++){
8              if(s[t]>s[t+1]){
```

```
9                   int a=s[t];
10                  s[t]=s[t+1];
11                  s[t+1]=a;
12              }
13          }
14      }
15
16      for(int i=0;i<n;i++){
17          cout<<s[i]<<" ";
18      }
19
20      return 0;
21  }
```

比べて見ると高級言語では配列を用いてデータの整理をしているがアセンブリでは配列は用いていない。また、ループの処理の際、高級言語ではループを一行でかけるが、アセンブリでは回数の保持と目標回数の比較をしないとループがかけない。また、高級言語では変数を無制限に使えるがアセンブリでは制限がある。また、高級言語では型指定しているがアセンブリではしていない。コードの長さもアセンブリではとても長くなっていて、一目見ても何をしているかがわかりにくい。次にアセンブリの計算量だ。結果は 83,129,195 という計算量になった。これは n=4,5,6 ということを考えて考察していく。また、それぞれのデータについての高級言語の計算量は、23,31,53（出力時の処理はノーカウント）

4 考察

高級言語とアセンブリ言語の相違点について述べたがここではなぜ違うかを考察していく。まずはじめに、高級言語が配列を扱えるのに対し、アセンブリでは扱えないのはアセンブリにおいて配列とは連続する番地を指すのでアセンブリのように番地指定を直接するような言語では必要ないからだと考えた。また、高級言語の配列とはあくまでコードを書く際に便利なツールとして発展したものだと考えた。また、変数に制限があることに関しては限られた変数だけで値を入れ変えていけば問題ないのでメモリの節約を考慮して設計されていると考えた。次に、アセンブリの計算量についてだ。5×n×n+7 について考えると n=4,5,6 のときで 87,132,187 というようになる。これは、アセンブリの計算量の結果と近い値で有ることからオーダーは O(n2) であることがわかる。またバブルソートは O(n2) であることから理論的計算量に従うことがわかる。また、高級言語と計算量を比べると N2 の係数が 1 で有ることがわかる。これは、高級言語では一つの処理の記述で終わることもアセンブリでは何回かに分けて実行しているからだと考えた。また、以上のことから、アセンブリが流れをイメージしにくく長くなってしまうことや自分たちの書くアセンブリによっては非効

率な作業をして計算量が多くなる観点から、プログラム的な観点からは利点がないと言える。ただし、プログラムを内部まで理解することを目的としたときは学習できるという面で良いと言える。

参考文献

[1] C とアセンブリ言語で学ぶ計算機プログラミングの基礎概念 : プログラムはプロセッサ上でどのように実行されるのか, 角川裕次著
[2] アセンブリプログラミング, 大駒誠一 [ほか] 著