Jurnal Fokus Elektroda : Energi Listrik, Telekomunikasi, Komputer, Elektronika dan Kendali)
Volume 08 No 04, Tahun 2023: Hal. 226-233.
e-ISSN: 2502-5562. Open Access at: http://ojs.uho.ac.id/index.php/jfe/
Penerbit : Jurusan Teknik Elektro Universitas Halu Oleo Kendari Sulawesi Tenggara,

Implementasi Algoritma Bubble Sort Untuk Ekstraksi Kode Biner Pada Intel Hex File

Putut Son Maria[1], Elva Susianti[2]

[1]Teknik Elektro, Universitas Islam Negeri Sultan Syarif Kasim Riau, [2]Teknik Elektronika, Politeknik Caltex Riau

Correspondent Author :putut.son@uin-suska.ac.id

Abstract **— Microprocessors or microcontrollers run programs in the form of binary code stored in hexa files after compilation. The sequence of binary code data is very important during the burning or loading process on a microprocessor or microcontroller so that there is no wrong location of data that fails in program execution by the microprocessor or microcontroller. The .hex file compiled from the small device c compiler (sdcc) turns out that the address offsets are not sequential so extracting the hex code will result in program execution failure or the program not functioning normally as it should if directly loaded into the microprocessor/microcontroller. This is a serious problem if it is to be applied to EPROM programmer devices or programmer chips that are made independently or non-commercially. This research aims to implement the bubble sort algorithm to sort lines in the contents of a .hex file based on the address offset produced by sdcc. Implementation was carried out by creating an application program using the Pascal language written in the DELPHI IDE (integrated development environment). The working steps start by copying the contents of the original .hex file to the memory(RAM) buffer and then making a copy of the original file to produce the target .hex file. The test uses 2 (two) .hex files as input and creates a target .hex file as the program output. The test results show that the contents of the target .hex file created are arranged sequentially according to the address offset.**

Keyword **— Bubble sort, Hex file, Microprocessor, Microcontroller, Sdcc.**

Abstrak — Mikroprosesor atau mikrokontroler menjalankan program berupa kode biner yang tersimpan dalam file heksa setelah melalui kompilasi. Urutan data kode biner sangat penting pada saat proses *burning* atau *loading* pada mikroprosesor atau mikrokontroler agar tidak terjadi salah lokasi data yang mengakibatkan kegagalan eksekusi program oleh mikroprosesor atau mikrokontroler. File .hex hasil kompilasi dari *small device c compiler*(sdcc) ternyata *offset* alamatnya tidak berurutan**,** sehingga ekstraksi kode heksa akan mengakibatkan kegagalan eksekusi program atau program tidak berfungsi normal sebagaimana seharusnya jika langsung diisikan ke mikroprosesor/mikrokontroler. Hal ini menjadi masalah serius ingin diaplikasikan pada perangkat *eprom programmer* atau *chip programmer* yang dibuat secara mandiri atau non-komersil. Penelitian ini bertujuan untuk mengimplementasikan algoritma *bubble sort* untuk mengurutkan baris dalam isi file .hex berdasarkan *offset* alamat yang dihasilkan oleh sdcc. Implementasi dilaksanakan dengan membuat program aplikasi menggunakan bahasa Pascal yang ditulis dalam IDE(*integrated development environtment*) DELPHI. Langkah kerjanya dimulai dengan menyalin isi file .hex asal ke *buffer* memori ram dan sampai membuat salinan file asal untuk menghasilkan file .hex target. Pengujian menggunakan 2(dua) file .hex sebagai masukan dan membuat file .hex target sebagai hasil luaran program. Hasil pengujian menunjukkan bahwa isi file .hex target yang dibuat telah tersusun secara berurutan menurut *offset* alamatnya.

Kata kunci — Bubble sort, File heksa, Mikroprosesor, Mikrokontroler, Sdcc.

I. PENDAHULUAN

Mikroprosesor dan mikrokontroler adalah piranti digital yang semakin populer dipelajari oleh kalangan akademisi, hobbyist dan praktisi. Mikroprosesor dan mikrokontroler harus diprogram dalam bentuk baris perintah atau program menggunakan bahasa assembly atau bahasa c agar dapat menjalankan fungsi kontrol seperti yang diinginkan oleh programmer.

Secara *default* sebenarnya mikroprosesor atau mikrokontroler wajib diprogram menggunakan bahasa Assembly, tetapi karena struktur dan karakter bahasa Assembly yang rumit sehingga memprogram mikroprosesor menggunakan bahasa C lebih banyak memberikan kemudahan bagi programmer karena strukturnya yang lebih teratur dan menggunakan sintak yang mirip dengan bahasa program *high level* lainnya[1]. Agar perintah-perintah program dalam bahasa C dapat dipahami oleh mikroprosesor maka perintah tersebut harus dialihbahasakan menggunakan *tool* yaitu sdcc atau *small device cross compiler*.

Biasanya tool sdcc sudah disematkan dalam compiler dan akan menghasilkan file .hex standar dimana kode binernya dapat diekstraksi dan dituliskan ke dalam memori program. Masalahnya adalah bahwa penggunaan sdcc menghasilkan file .hex yang urutan *offset* alamatnya tidak teratur, sehingga hal ini akan mengakibatkan ekstraksi kode biner dari file .hex menjadi riskan salah penempatan pada memori program, efeknya adalah mikroprosesor tidak akan menjalankan perintah seperti yang seharusnya. Hal ini bertolak belakang dengan bahasa Assembly yang *offset* alamat dari file .hex-nya sudah teratur membentuk pola dari nilai kecil menuju besar (*descending*).

Penelitian ini bertujuan untuk mengimplementasikan algoritma *Bubble sort* untuk mengurutkan baris yang mengandung kode biner yang ada dalam file .hex. Pengurutan didasarkan pada *offset* alamat awal setiap barisnya dan kemudian disusun ulang sehingga file .hex yang baru akan tersusun berurutan secara *descending*. Hasil file .hex inilah yang dapat diekstraksi langsung kode binernya sehingga resiko ketidaksesuaian antara *offset* alamat dengan isi kode binernya dapat dihindari.

II. LANDASAN TEORI

2.1 *Intel hex file*

Intel hex file adalah file yang berisi karakter-karakter dalam sistem bilangan heksadesimal yang dibingkai dengan simbol dan karakter khusus yang merupakan transformasi dari kode perintah yang akan dijalankan pada sistem mikroprosesor atau mikrokontroler. Baris perintah atau program yang dibuat oleh seorang programmer pada saat selesai dikompilasi akan menghasilkan kode biner yang merupakan representasi bahasa mesin yang artinya hanya dapat dipahami oleh mikroprosesor.

Kode biner adalah salah satu dari 4(empat) informasi utama dari setiap baris isi file .hex selain *byte* jumlah data, alamat awal, dan *checksum* dari baris tersebut. Kode biner harus ditempatkan secara berurutan sesuai dengan *offset* alamat awal agar mikroprosesor dapat menjalankan sesuai algoritma program yang ditulis oleh *programmer*. Kode biner tersemat dalam file dengan ekstensi .hex setelah baris perintah atau program yang telah ditulis oleh *programmer* selesai dikompilasi. Kode biner harus disimpan ke memori program secara berurutan sesuai dengan *offset* alamat awalnya[4][5]. Gambar 1 menunjukkan contoh struktur baris standar file .hex. Kode biner harus diekstrak dari file .hex mulai dari karakter ke-10 untuk setiap barisnya.

```
:BBAAAATT[DDDDDDDD]CC

where
: is start of line marker
BB is number of data bytes on line
AAAA is address in bytes
TT is type discussed below but 00 means data
DD is data bytes, number depends on BB value
CC is checksum (2s-complement of number of
bytes+address+type+data)
```

Gambar 1. Struktur baris file .hex

Simbol ‘:’ adalah sebagai penanda awal baris, dua karakter berikutnya(BB) menunjukkan jumlah byte dalam baris tersebut, empat karakter berikutnya(AAAA) menunjukkan *offset* alamat, kode biner ditunjukkan dengan (DDDD…) dan dua karakter berikutnya(CC) menunjukkan kode baris yang kemungkinan berisi jumlah data atau pertanda akhir dari file. Contoh isi file .hex ditunjukkan seperti pada Gambar 2, *offset* alamat pada baris ke-1 berada di 0000 hex, baris ke-2 di 0006 hex, baris ke-3 di 0003 hex dan seterusnya. *Offset* alamat ini seharusnya berurutan mulai dari alamat kecil menuju besar (*descending*), sehingga jelas bahwa *offset* alamat dalam file .hex pada contoh Gambar 2 tidak beraturan dan hal ini dapat mengakibatkan ekstraksi dan penulisan kode biner ke memori program mikroprosesor/mikrokontroler akan mengakibatkan kegagalan eksekusi.

```
:03000000020008F3
:0300610002000397
:0500030012006480FE04
:0E006400758920758DFD759850D28E75A89007
:0200720080FE0E
:06003700E478FFF6D8FD9D
:080015007900E94400601B7A48
:05001D0000900078785E
:030022000075A0C6
:0A00250000E493F2A308B8000205FE
:08002F00A0D9F4DAF275A0FF7C
:08003D007800E84400600A7934
:030045000075A0A3
:0600480000E4F309D8FCFE
:08004E007800E84400600C7921
:0B00560000900000E4F0A3D8FCD9FAF1
:03000800758107F8
:0A000B00120074E582600302000396
:04007400758200226F
:00000001FF
```

Gambar 2. Contoh file .hex hasil kompilasi sdcc

2.2 *Bubble Sort*

Algoritma *bubble sort* atau *exchange sort* adalah salah satu diantara 6(enam) algoritma sortasi yang populer. *Bubble sort* relatif mudah dipahami dan sesuai untuk aplikasi data yang jumlahnya tidak terlalu banyak[7]

Algoritma *bubble sort* dijalankan menggunakan *pseudo-code* berikut :

1. Misalnya terdapat *array* dengan m anggota.
2. Elemen *array* ke-n(ε_n) dibandingkan dengan elemen ke-(n+1), jika $\varepsilon_{(n+1)} < \varepsilon_{(n)}$ maka posisi kedua elemen ditukar.

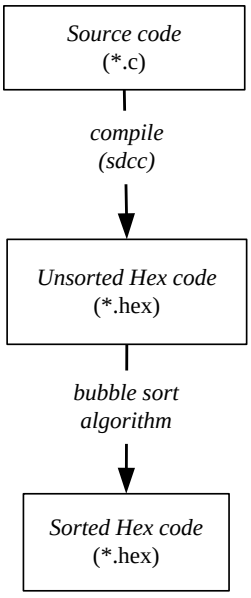
3. Elemen *array* ke-(n+1) dibandingkan dengan elemen ke-(n+2), jika $\varepsilon_{(n+2)} < \varepsilon_{(n+1)}$ maka posisi kedua elemen ditukar.
4. Proses berlangsung terus untuk elemen berikutnya dalam *array* sampai akhir indek *array*.
5. Jika tidak ada lagi yang ditukarkan, maka algoritma berhenti. [3][8].

Pada penelitian ini dipilih menggunakan algoritma *bubble sort* karena beberapa pertimbangan berikut, yaitu : 1) data yang akan diolah berukuran relatif kecil atau maksimal hanya 32 kilobyte yang mana ini adalah ukuran *flash* memori arduino berbasis ATMega328[2]. 2) kecepatan pengurutan bukanlah prioritas utama dalam ekstraksi, dan 3) kesederhanaan algoritma *bubble sort* memudahkan untuk penulisan kode program.

III. METODE PENELITIAN

3.1 Blok diagram urutan kerja

Urutan kerja pada penelitian ini ditunjukkan pada Gambar 3. Eksperimen diawali dengan membuat *source code* atau baris program yang ditulis menggunakan bahasa C, kemudian program dikompilasi menggunakan sdcc untuk menghasilkan file .hex.



Gambar 3. Blok diagram urutan kerja

File .hex yang dihasilkan oleh sdcc sampai di tahap ini adalah file yang *offset* alamatnya belum berurutan seperti pada Gambar 2. File .hex dari sdcc inilah yang akan disusun ulang sehingga menghasilkan file .hex baru yang *offset* alamatnya telah berurutan, dengan demikian dari hasil sortasi akan dapat dilihat dengan membandingkan file .hex sebelum sortir dan sesudah sortir.

3.2 *Source code*

Kode biner yang menjadi obyek penelitian ini adalah hasil kompilasi *source code* atau program untuk mikroprosesor Intel mcs-51 series. Jumlah program dipersiapkan sebanyak 2(dua) file yaitu *source code* tanpa interupsi seperti pada Gambar 4 dan *source code* dengan interupsi seperti pada Gambar 5. Hasil kompilasi program pada Gambar 4 menghasilkan file .hex yang isinya ditunjukkan pada Gambar 2, sedangkan hasil kompilasi dari kode program Gambar 5 ditunjukkan pada Gambar 6. Isi file .hex Gambar 2 dan Gambar 6 terlihat menunjukkan *offset* alamat yang tidak berurutan. Kedua file .hex tersebut akan menjadi obyek uji pada penelitian ini.

```
#include<8051.h>

void main()
  {
   TMOD=0x20;
   TH1=0xFD;
   SCON=0x50;
   TR1=1;
   IE=0x90;
   while(1);
   }
```

Gambar 4. Kode program tanpa interupsi

```
#include<8051.h>

void main()
  {
   TMOD=0x20;
   TH1=0xFD;
   SCON=0x50;
   TR1=1;
   IE=0x90;
   while(1);
   }

void serial(void)__interrupt(4)
 {
  char c;
  c=SBUF;
  IE=0x00;

   SBUF = c;

   if(c==0x0d)
   {
   P0=~P0;
   SBUF='A';
   while(TI==0);
   TI=0;
   SBUF='C';
   while(TI==0);
   TI=0;
   SBUF='K';
    while(TI==0);
    TI=0;
    }
   RI=0;
   IE=0x90;
  }
```

Gambar 5. Kode program dengan interupsi

```
:0400000002002B329D
:01000B0032C2
:0100130032BA
:01001B0032B2
:0300230002009741
:0300840002002651
:0500260012008780FEBE
:0E008700758920758DFD759850D28E75A890E4
:0200950080FEEB
:0E009700C0E0C002C0D075D000AA9975A800C4
:0B00A500BA0D1DE580F4F5807599414F
:0500B00010990280FB25
:0300B500759943F7
:0500B80010990280FB1D
:0300BD0075994BE7
:0500C00010990280FB15
:0C00C500C29875A890D0D0D002D0E032D4
:06005A00E478FFF6D8FD7A
:080038007900E94400601B7A25
:05004000009000D578DE
:030045000075A0A3
:0A00480000E493F2A308B8000205DB
:08005200A0D9F4DAF275A0FF59
:080060007800E84400600A7911
:030068000075A080
:06006B0000E4F309D8FCDB
:080071007800E84400600C79FE
:0B00790000900000E4F0A3D8FCD9FACE
:03002B00758107D5
:0A002E001200D1E5826003020026F3
:0400D1007582002212
:00000001FF
```

Gambar 6. Isi file .hex hasil kompilasi program gambar 5

3.3 Tahap pemrograman

Pemrograman untuk sortasi file .hex terdiri dari 5(lima) tahapan yaitu : a) *buffering*, b) perhitungan jumlah baris, c) sortir, d) pencocokan dan pencatatan indek, dan e) finalisasi file. Implementasi pemrograman *bubble sort* pada penelitian ini menggunakan *compiler* Delphi 7 dengan default bahasa program Pascal. Delphi dipilih karena telah memiliki fitur *graphical user interface*(GUI) yang memudahkan untuk memonitor perubahan nilai variabel dalam program.

a) *Buffering*

Buffering adalah tahap dimana isi dari file .hex asal(sumber) akan disalin ke dalam variabel dalam pemrograman yaitu berupa jenis *random access memory*(ram) pada komputer. Tujuan dari *buffering* adalah agar komputer tidak terlalu sering mengakses(baca-tulis) *harddrive* sehingga meminimalkan resiko rusak pada *harddrive*. *Pseudo-code* dari langkah *buffering* ditunjukkan pada Gambar 7.

file_buffer adalah variabel bertipe *file of byte*, yaitu file dengan elemen berupa bilangan bulat(*byte*). Isi file .hex dari *harddrive* yang akan di-*sorting* disalin ke *file_buffer*, yaitu variabel yang secara fisik menggunakan alokasi memori ram.

```
assignfile(file_buffer,Form1.OpenDialog1.FileName);
{$I-}
reset(file_buffer);
{$I+}
```

Gambar 7. Buffering

b) Perhitungan jumlah baris

Proses perhitungan jumlah baris bertujuan untuk menghitung banyaknya baris dari isi file .hex, langkah ini penting karena untuk menentukan indek setiap baris dan menghitung *offset* alamat pada baris tersebut. *Pseudo-code* untuk menghitung jumlah baris ditunjukkan pada Gambar 8. Dalam kode program terdapat variabel bernama *before* dan *after* yaitu variabel bertipe larik atau *array*. Variabel *before* menyimpan semua *offset* alamat yang dibaca dari file .hex sebelum diurutkan. Variabel index adalah variabel bertipe *byte* yang mencatat jumlah baris. Baris terakhir dari file .hex tidak perlu dilibatkan dalam perhitungan karena isi pada baris tersebut pasti sama seperti ditunjukkan oleh baris terakhir pada Gambar 2 dan 6.

```
jumlah_baris := 0;
    while not eof(file_buffer) do
    begin
    read(file_buffer,data);
    if data = 58 then
       begin
       jumlah_baris := jumlah_baris + 1;
       read(file_buffer,data);read(file_buffer,data);
       alamat := read_offset_address;
       index := jumlah_baris;
       before[index] := alamat;
       end;
    end;
    jumlah_baris := jumlah_baris-1;

    for index := 1 to jumlah_baris do
    begin
    after[index] := before[index];
    end;
```

Gambar 8. Kode program perhitungan baris

c) *Sorting*

Pada langkah (b) menunjukkan bahwa variabel *after* akan diisi sama dengan variabel *before*. Berikutnya pada langkah *sorting* yang dilakukan adalah isi dari variabel *after* akan dimanipulasi menggunakan algoritma *bubble sort* menggunakan kode program seperti pada Gambar 9. Variabel *buf* diperlukan sebagai penyimpan nilai sementara jika ternyata ditemukan kondisi $\varepsilon_{(n+1)} < \varepsilon_{(n)}$. Jumlah iterasi yang ditetapkan dalam kode program ini sejumlah j x i dimana j adalah m-2, i adalah m-1 dan m adalah jumlah baris dalam file .hex. Hasil dari tahap ini adalah variabel

after akan berisi *offset* alamat yang tersusun berurutan secara *descending*[6].

```
procedure urutkan;
var i,j,buf : byte;
begin
    for j := 1 to jumlah_baris-2 do
    begin
        for i := 1 to jumlah_baris-1 do
        begin
            if after[i+1] < after[i] then
            begin
            // tukar tempat
            buf := after[i+1];
            after[i+1] := after[i];
            after[i] := buf;
            end;
        end;
    end;
end;
```

Gambar 9. pseudo code pengurutan

d) Pencocokan dan pencatatan indek(*match and indexing*)

Match and indexing adalah proses dimana mencari kecocokan isi variabel *after*(hasil setelah diurutkan) dengan isi variabel *before*(data sebelum diurutkan). Variabel *before* dan *after* adalah *array* berisi *offset* alamat dari setiap baris file .hex. Cara kerja *match and indexing* adalah setiap isi variabel *after* akan dicari kecocokannya dengan isi variabel *before*, jika ditemukan kecocokan maka indek pada variabel *before* akan dicatat dalam variabel *daftar_baris.* Proses akan diulang sampai sejumlah j x i seperti ditunjukkan pada kode program Gambar 10.

```
for j := 1 to jumlah_baris do
begin
  for i := 1 to jumlah_baris do
  begin
  if after[j] = before[i] then daftar_baris[j] := i;
  end;
end;
```

Gambar 10. Pencocokan isi dan pencatatan indek

e) Finalisasi file

Proses sebelumnya pada langkah (d) telah menghasilkan larik *daftar_baris* yang berisi urutan indek yang menunjukkan nomer baris yang *offset* alamatnya telah tersusun secara *descending*. Pada langkah finalisasi ini adalah tahap akhir dimana hasil *sorting* akan dimanfaatkan untuk menghasilkan file .hex baru yang merupakan hasil duplikat, dimana isinya nanti dapat dilihat perbedaannya dengan file .hex asli dari hasil *sdcc*. Langkah duplikasi file menggunakan pedoman berdasarkan urutan indek yang disimpan pada variabel *daftar_baris*. Gambar 11 menunjukkan *pseudo-code* untuk duplikasi file .hex.

Pada penelitian ini file hasil duplikasi diberi nama code.hex yang disimpan di *drive* H. Variabel *baris_target* adalah variabel bertipe *byte* yang nilainya akan tergantung pada variabel *daftar_baris* dengan indek tertentu. Pada iterasi awal, nilai indek akan bernilai 1 dan mengakibatkan nilai pada variabel *baris_target* akan mengikuti sesuai nilai dari variabel *daftar_baris*. Berikutnya terdapat variabel *ptr_baris* yaitu variabel yang akan mencatat jumlah baris yang telah dilewati oleh *pointer* setelah perintah *read* dalam instruksi program bahasa pascal. Pada program bahasa pascal pembacaan *byte* menggunakan perintah *read* dalam sebuah file bersifat *increment* dan tidak berlaku *reverse*, artinya untuk menuju ke baris tertentu dalam file .hex maka pointer harus diarahkan ulang kembali ke awal file.

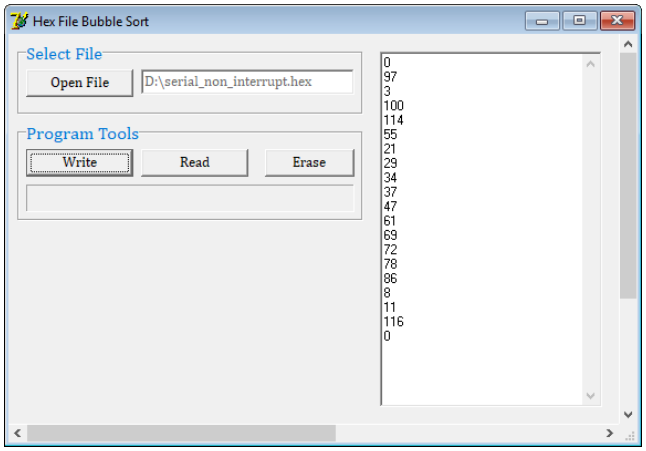
```
assignfile(file_arranged, 'H:\code.hex');
  reset(file_buffer);
  rewrite(file_arranged);

  for index := 1 to jumlah_baris do
  begin
    baris_target := daftar_baris[index];

    ptr_baris := 0;
    reset(file_buffer);
    while ptr_baris < baris_target do
    begin
      if ptr_baris = jumlah_baris then
        begin
        MessageDlg('End of File', mtInformation,[mbOk], 0);
        closefile(file_arranged);
        exit;
        end
      else
        if (eof(file_buffer) = false) then
        begin
        read(file_buffer, data);
        end;

      if data = 58 then
        begin
        ptr_baris := ptr_baris + 1;
        end;
    end;

    //copy 1 baris
    write(file_arranged, data);
    repeat
    read(file_buffer, data);
     if data <> 58 then write(file_arranged, data);
    until data = 58;
```

Gambar 11. Finalisasi file .hex

Duplikasi isi file asal/sumber .hex ke file target code.hex terjadi jika *ptr_baris* telah mencapai nilai yang sama dengan yang ditunjukkan oleh variabel *baris_target* seperti ditunjukkan oleh 5(lima) baris terakhir dari Gambar 11.

IV. HASIL DAN PEMBAHASAN

Hasil *running* program aplikasi ditunjukkan pada Gambar 12. Evaluasi nilai variabel ditampilkan pada komponen GUI berjenis *memo* pada bagian kanan antarmuka aplikasi. Pengujian pertama dilakukan untuk mensortir file .hex yang isinya seperti pada Gambar 2. Hasil evaluasi variabel *before* ditunjukkan seperti pada Gambar 12(bagian kanan). Terlihat urutan *offset* alamat pada variabel *before* sesuai dengan file .hex yang diolah, tampilan angka pada kotak memo adalah dalam sistem bilangan desimal, sedangkan dalam file .hex adalah dalam sistem bilangan heksadesimal. Hasil perbandingan antara *offset* alamat pada file .hex dengan tampilan dari *memo* ditunjukkan pada Tabel 1, terlihat bahwa nilai yang dibaca dari file sumber .hex dengan hasil dari variabel menunjukkan besaran yang sama, hal ini berarti bahwa program telah berhasil membaca semua *offset* alamat di semua baris dalam file .hex. Isi variabel *before* kemudian disalin ke variabel *after* dan disortir sehingga isi variabel after seperti ditunjukkan pada Gambar 13(a). Sampai pada tahap ini terlihat bahwa urutan data sudah menunjukkan pola *descending*.



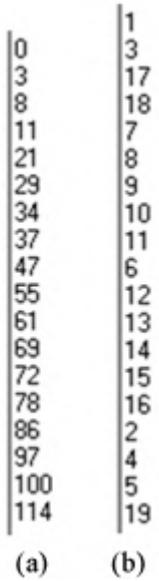
Gambar 12. Tampilan aplikasi GUI

Langkah berikutnya adalah pencocokan alamat antara variabel *before* dengan *after*, hal ini diperlukan untuk mencatat urutan baris ke berapa yang harus disalin dari file sumber ke file target. Daftar urutan baris yang harus disalin ke file target dicatat oleh variabel *daftar_baris* yang isinya ditunjukkan seperti pada Gambar 13(b).

Langkah berikutnya dengan mengacu pada gambar 2 dan 13(b), maka untuk menghasilkan file target *code.hex* harus disalin dari file sumber dengan mengikuti urutan baris seperti yang ditunjukkan pada Gambar 13(b).

TABEL 1. DAFTAR *OFFSET* ALAMAT PADA VARIABEL *BEFORE*

File .hex	(variabel *before*)
0000	0
0061	97
0003	3
0064	100
0072	114
0037	55
0015	21
001D	29
0022	34
0025	37
002F	47
003D	61
0045	69
0048	72
004E	78
0056	86
0008	8
000B	11



Gambar 13. Isi variabel after dan daftar_baris

Dalam penelitian ini, langkah terakhir dari program sorting adalah pembuatan file target code.hex yang berisi salinan dari file sumber. Yang membedakan file target

```
:03000000020008F3
:0500030012006480FE04
:0300800758107F8
:0A000B00120074E582600302000396
:080015007900E94400601B7A48
:05001D0000900078785E
:030022000075A0C6
:0A00250000E493F2A308B8000205FE
:08002F00A0D9F4DAF275A0FF7C
:06003700E478FFF6D8FD9D
:08003D007800E84400600A7934
:030045000075A0A3
:0600480000E4F309D8FCFE
:08004E007800E84400600C7921
:0B00560000900000E4F0A3D8FCD9FAF1
:0300610002000397
:0E006400758920758DFD759850D28E75A89007
:0200720080FE0E
:04007400758200226F
:00000001FF
```

Gambar 14. Hasil algoritma bubble sort terhadap file .hex(source code 1)

```
:0400000002002B329D
:01000B0032C2
:0100130032BA
:01001B0032B2
:0300230002009741
:0500260012008780FEBE
:03002B00758107D5
:0A002E001200D1E5826003020026F3
:080038007900E94400601B7A25
:05004000009000D578DE
:030045000075A0A3
:0A00480000E493F2A308B8000205DB
:08005200A0D9F4DAF275A0FF59
:06005A00E478FFF6D8FD7A
:080060007800E84400600A7911
:030068000075A080
:06006B0000E4F309D8FCDB
:080071007800E84400600C79FE
:0B00790000900000E4F0A3D8FCD9FACE
:0300840002002651
:0E008700758920758DFD759850D28E75A890E4
:0200950080FEEB
:0E009700C0E0C002C0D075D000AA9975A800C4
:0B00A500BA0D1DE580F4F5807599414F
:0500B00010990280FB25
:0300B500759943F7
:0500B80010990280FB1D
:0300BD0075994BE7
:0500C00010990280FB15
:0C00C500C29875A890D0D0D002D0E032D4
:0400D1007582002212
:00000001FF
```

Gambar 15. Hasil algoritma bubble sort terhadap file .hex(source code 2)

dengan file sumber adalah bahwa urutan baris pada file target telah berurutan secara *descending*. Hasil file target ditunjukkan pada Gambar 14. Program aplikasi yang sama juga diuji untuk mengolah file .hex hasil kompilasi *sdcc* seperti pada Gambar 6, dimana *source code*-nya ditunjukkan pada Gambar 5. Dengan baris perintah *source code* yang lebih banyak, maka kode biner yang dihasilkan dalam file .hex juga semakin panjang. Mengacu pada gambar 6 dan membandingkannya dengan gambar 15 maka terlihat bahwa terlihat perbedaan pada *offset* alamat Gambar 15 sudah tersusun berurutan membentuk pola *descending*

Isi file .hex sumber pada Gambar 6 berhasil disalin secara utuh dan *offset* alamat telah tersusun secara runut pada file .hex target. Ini membuktikan bahwa algoritma *bubble sort* yang diimplementasikan pada penelitian ini sudah memadai untuk menangani file .hex yang dihasilkan oleh *sdcc*. Hasil dari *sorting* selanjutnya dapat diekstraksi untuk mendapatkan kode biner untuk proses penulisan pada memori program, seperti misalnya pada *eprom programmer* atau *microcontroller loader*.

V. KESIMPULAN

Algoritma *bubble sort* cukup dapat diandalkan dan telah memadai untuk *sorting* data dengan ukuran hingga 32 *kilobyte*. Beberapa keunggulan menggunakan algoritma *bubble sort* adalah mekanismenya yang sederhana sehingga mudah dipahami dan diimplementasikan dalam bentuk program aplikasi, kinerjanya yang cukup stabil terbukti dengan pengujian untuk kondisi dimana elemen yang bernilai sama akan tetap menempati *member array* dengan indek yang bersebelahan walaupun iterasi diulang-ulang, dan algoritma *bubble sort* dapat dijalankan tanpa harus menambahkan memori dalam komputer. Walaupun algoritma ini kurang efisien untuk pengolahan data besar tetapi untuk ukuran data yang ditetapkan pada penelitian ini ternyata algoritma *bubble sort* dapat menghasilkan kinerja yang baik.

DAFTAR ACUAN

[1] Devie I., E. Apriaskar, Djuniadi., “Sistem Jemuran Otomatis Menggunakan Mikrokontroler Berbasis Arduino,” Jurnal Fokus Elektroda, Vol. 06 No. 01 : 43-47. 2021.

[2] Dadi, Kusno U., Andhy P., M. A. Aziz., “Palang Pintu Dengan Absensi Barcode dan Deteksi Suhu Badan Berbasis Arduino,” Orbith : Majalah Ilmiah Pengembangan Rekayasa dan Sosial., Vol. 18 No. 2 : 130-141. 2022.

[3] Eko Nur W., “Algoritma Sederhana dalam Memahami Proses Pengurutan Data. Jurnal Teknologi Informasi,” DINAMIK Vol. XIV No. 1 : 14-22. 2009.

[4] Intel. 2012. [online] https://www.intel.com/content/www/us/en/support/programmable/articles/000076770.html. [Diakses April 2023]

[5] Kanda, 2016, [online]: https://www.kanda.com/blog/microcontrollers/intel-hex-files-explained/. [Diakses Mei 2023]

[6] M. Farhan A., I. Hafiza, R. Wahyuni, A. Syahputra, ”Penggunaan Algoritma Bubble Sort dalam Pengurutan Nomor Induk Mahasiswa”, Jurnal Ilmu Komputer Hello World, Vol 2 No 1 : 14-19, 2023.
[7] Panny A.R., “Analisis Perbandingan Kompleksitas Algoritma Pengurutan Nilai,” Jurnal Evolusi Vol. 4 No 2: 64-75, 2016.
[8] Retnoningsih E., “Algoritma Pengurutan Data (Sorting) Dengan Metode Insertion Sort dan Selection Sort,” Jurnal Information Management For Educators And Professionals. 3(1): 95 – 106, 2018.