Penggunaan Kode Huffman dalam Kompresi Audio MP3

Willy Setiawan (13508043)

Jurusan Teknik Informatika ITB
Alamat : Jl.Ganesha 10,Bandung
e-mail: if18043@students.if.itb.ac.id

ABSTRAK

Makalah ini berisi tentang kompresi audio,yaitu MP3 dan penggunaan kode Huffman di dalamnya. Kode Huffman merupakan salah satu teknik kompresi yang masih seringkali digunakan hingga sekarang. Kompresi data merupakan teknik untuk memperkecil ukuran data seminimal mungkin, tapi masih berisi pesan yang sama. Kode Huffman seringkali digunakan untuk kompresi gambar dan audio. Format MP3 (MPEG-1 Audio Layer 3) adalah salah satu contoh format yang seringkali digunakan dalam kompresi audio. Ada banyak lagi jenis kompresi audio , seperti Vorbis dan FLAC, midi , dan lain-lain, tapi format MP3 lah yang paling banyak digunakan oleh orang-orang dalam mendengarkan data audio.

Kata kunci: mp3, kode huffman , kompresi

1. PENDAHULUAN

Dalam zaman yang makin maju ini,teknologi baru yang belum pernah dibayangkan sebelumnya makin banyak bermunculan. Mulai dari munculnya komputer , *handphone*,dan masih banyak lagi. Salah satu teknologi baru yang muncul adalah MP3 Player.MP3 Player ini bisa membuat seseorang mendengarkan lagu secara digital,tidak perlu menggunakan kaset atau CD seperti zaman dahulu.

MP3,format yang bisa dimainkan oleh MP3 Player adalah salah satu format audio yang paling sering digunakan dalam penyimpanan data audio.MP3 merupakan bentuk audio yang paling sering digunakan,karena data yang disimpan menyerupai data yang asli pada saat direkam, dan memiliki ukuran yang tidak terlalu besar dibandingkan format lain.

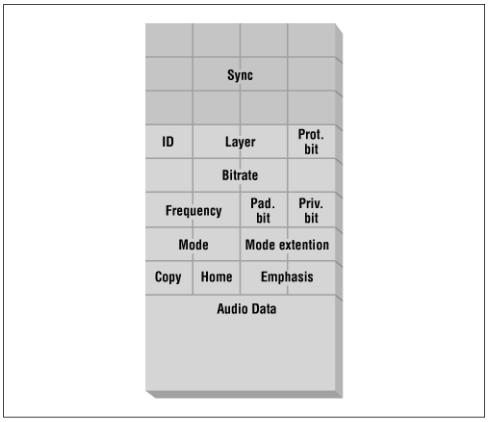
Tapi, bagaimana dengan cara pembentukan MP3 bisa memiliki ukuran data yang kecil? Untuk memiliki ukuran data yang tidak besar , diperlukan metode kompresi. Kompresi memiliki arti untuk mepersingkat ukuran bit yang diperlukan dalam suatu data. Metode kompresi data dalam audio yang paling sering digunakan adalah metode pengkodean Huffman. Kode Huffman merupakan salah satu metode kompresi data yang terkenal. Kompresi data ini memperkecil bit untuk kata yang sering muncul dan memperbesar bit yang jarang muncul.Sebagai contoh, data yang awalnya memiliki panjang 56 bit, bisa dikompresi menjadi 11 bit tanpa ada informasi yang hilang.

2. Pengenalan Format MP3 dan Kode Huffman

2.1. Pengenalan MP3

MP3 merupakan format yang menarik karena bisa mempertahankan kualitas suara sementara memiliki ukuran yang tidak terlalu besar. Teknologi ini dikembangkan oleh seorang insinyur Institut Fraunhofer di Jerman, Karlheinz Brandenburg. MP3 terdiri dari banyak sekali frame ,dimana setiap frame mengandung sebagian detik dari data audio yang berguna,yang siap dikonstruksi ulang oleh *decoder*. Yang dimasukkan ke setiap bagian awal dari frame data adalah “header frame”,yang mengandung 32 bit meta-data yang berhubungan dengan frame data yang masuk.

Gambar 1. Data yang mendeskripsikan bentuk struktural dari frame tersebut;data inilah yang disebut *header* dari frame



Gambar 2.Frame Header MP3 secara visual

Tabel 2.Karakteristik file header

Posisi	Tujuan	Bit
A	Sinkronisasi frame	11
B	Versi MPEG	2
C	Layer MPEG	2
D	Proteksi	1
E	Index bitrate	4
F	Frekuensi tingkat sampel	2
G	Bit padding	1
H	Bit privat	1
I	Mode channel	2
J	Mode lanjutan	2
K	Copyright	1
L	Originalitas	1
M	Emphasis	2

Walau MP3 terlihat begitu hebat, ternyata format ini juga memiliki keterbatasannya tersendiri :
* Bit rate terbatas, maksimum 320 kbit/s (beberapa encoder dapat menghasilkan bit rate yang lebih tinggi, tetapi sangat sedikit dukungan untuk mp3-mp3 tersebut yang memiliki bit rate tinggi
* Resolusi waktu yang digunakan mp3 dapat menjadi terlalu rendah untuk sinyal-sinyal suara yang sangat transient, sehingga dapat menyebabkan noise.
* Resolusi frekuensi terbatasi oleh ukuran window yang panjang kecil, mengurangi efisiensi coding.
* Tidak ada scale factor band untuk frekuensi di atas 15,5 atau 15,8 kHz.
* Mode jointstereo dilakukan pada basis per frame.
*Delay bagi encoder/decoder tidak didefinisikan, sehingga tidak ada dorongan untuk gapless playback (pemutaran audio tanpa gap). Tetapi, beberapa encoder seperti LAME dapat menambahkan metadata tambahan yang memberikan informasi kepada MP3 player untuk mengatasi hal itu.

2.1. Pengenalan Kode Huffman

Pertama,, akan dikenalkan lebih dalam tentang kompresi. Kompresi merupakan proses melakukan *encoding* informasi menggunakan bit yang lebih sedikit dari informasi awal. Terdapat dua jenis tipe kompresi, yaitu : *lossless dan lossy* . Dalam kompresi *lossless*, awalnya data akan dipecah menjadi ukuran yang lebih kecil dan pada akhirnya data disatukan kembali. Sedangkan, dalam kompresi *lossy*, ada bit informasi yang dieliminasi setelah dilakukan kompresi.Kompresi tipe ini sering dilakukan untuk kompresi gambar.Prinsip umum dalam proses kompresi adalah mengurangi duplikasi data sehingga memori untuk merepresentasikan menjadi lebih sedikit daripada representasi data digital semula.
Beberapa perbandingan antar *lossy* dan *lossless* :
• Keuntungan dari metode lossy atas lossless adalah dalam bebeapa kasus metode lossy dapat menghasilkan file kompresi yang lebih kecil dibandingkan dengan metode lossless yang ada, ketika masih memenuhi persyaratan aplikasi.

• Metode lossy sering digunakan untuk mengkompresi suara, gambar dan video. karena data tersebut dimaksudkan kepada human interpretation dimana pikiran dapat dengan mudah “mengisi bagian-bagian yang kosong” atau melihat kesalahan masa lalu sangat kecil atau inkonsistensi-idealnya lossy adalah kompresi transparan, yg dapat diverifikasi dengan tes ABX. Sedangkan lossless digunakan untuk mengkompresi data untuk diterima ditujuan dalam kondisi asli seperti dokumen teks.

• *Lossy* akan mengalami *generation loss* pada data sedangkan pada *lossless* tidak terjadi karena data yang hasil dekompresi sama dengan data asli.

Kode Huffman merupakan salah satu metode kompresi data yang diciptakan oleh David A. Huffman. Kode Huffman menggunakan metode tertentu untuk merepresentasikan tiap simbol dimana ekspresi yang paling sering muncul mendapat ukuran bit terkecil dan ekspresi yang jarang muncul mendapat ukuran bit yang lebih banyak.
Proses pembentukan dari kode Huffman adalah :

1. Pilih 2 simbol dengan peluang paling kecil. Kedua simbol tadi dikombinasikan sebagai simpul orangtua dan peluang nya dijumlahkan. Simbol baru ini
2. diperlakukan sebagai simpul baru dan diperhitungkan dalam mencari simbol selanjutnya yang memiliki peluang paling kecil.
3. Selanjutnya, pilih dua simbol berikutnya,termasuk simbol baru, yang mempunyai

peluang terkecil. Lakukan hal yang sama seperti langkah sebelumnya.

4. Ulangi hingga seluruh tulisan terkode.

Sebagai contoh,akan diberikan tabel dari banyaknya kemunculan suatu tulisan “ABCCAAD”:

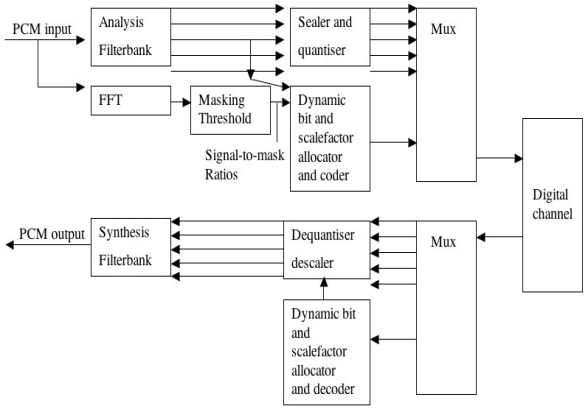
Tabel 1. Tabel Kemunculan Tulisan

Simbol	Kekerapan	Peluang
A	3	3/7
B	1	1/7
C	2	2/7
D	1	1/7

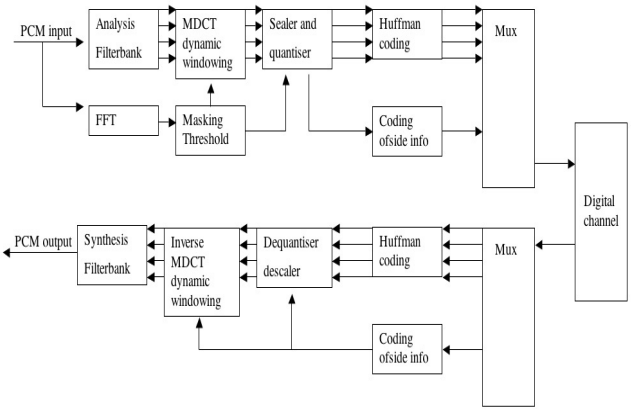
Pada saat dilakukan langkah - langkah pengkodean Huffman seperti diatas, maka hasilnya adalah :
Kode A : 0
Kode B : 110
Kode C : 10
Kode D : 111

3. Kompresi Audio MP3

Untuk melakukan kompresi audio , terdapat beberapa metode yang dapat digunakan,antara lain :model psikoakustik ,*auditory masking*,critical band,dan joint stereo.Dalam melakukan kompresi MP3,metode yang dilakukan adalah model psikoakustik.Beberapa percobaan pada psikoakustik adalah MPEG Layer-1,diikuti MPEG Layer-2. Kedua model pertama ini mengurangi secara drastis tingkat data yang diperlukan untuk memroduksi ulang suara berkualitas CD.MPEG Layer-3 menguranginya lebih jauh(Layer-1 : 384 kbps,Layer-2: 192 kbps,Layer-3 :112 kbps). Perbedaan antara MPEG Layer 1,2 dan 3 dari segi desain :

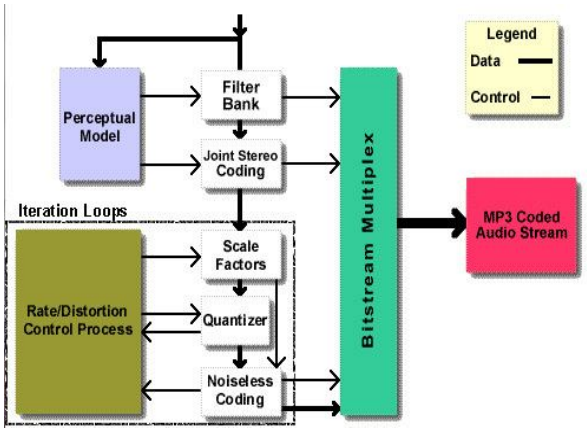


Gambar 3. MPEG Layer 1 dan 2



Gambar 4. MPEG Layer 3

Proses pembentukan MP3 :



Gambar 5. Proses Pembentukan File MP3

Blok pertama yang dilihat adalah blok Filter Bank. Fraunhofer menggunakan solusi hibrid untuk MP3 yang mengkombinasikan sebuah bank filter polifase dan sebuah *Modified Discrete Cosine Transform* (MDCT).Bank filter polifase adalah pemisahan awal audio stream menjadi frekuensi sub-band yang berjarak sama.Dengan sampel ini,frekuensi inti dipilih dan kemudian digunakan untuk kompresi. Satu masalah yang muncul dari pemilihan ini adalah frekuensi inti yang bertumpang tindih dengan sub-band,yang menyebabkan masalah kualitas dan kejernihan suara. Untuk mencegah masalah ini,MDCT digunakan pada 32 sub-band dari bank filter.Setiap jarak sub-band dilebarkan untuk bertumpang tindih dengan sub-band tetangga.Sampel baru diproses melalui MDCT dan ditambah dengan fungsi inversnya untuk membatalkan setiap kesalahan pada MDCT.Proses pembatalan ini dinamakan *time domain aliasing cancellation*.Ini adalah proses yang memakan waktu,tapi kalkulasi dan waktu yang dibutuhkan dikurangi menggunakan faktorisasi rekursif.

Sampel hasil dilewatkan ke porsi dari encoder *joint stereo coding*.Joint stereo coding adalah dimana informasi yang berlebihan dan tidak penting dibuang dari bit stream.

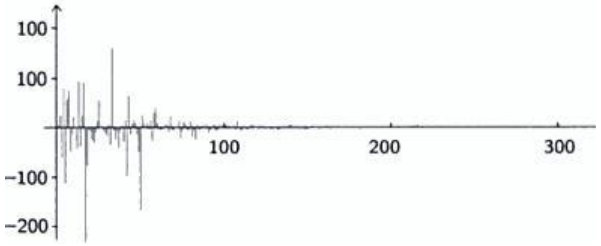
Model perseptual (perceptual model) mengkalkulasikan akan menjadi apa frekuensi limit untuk tiap sub-band,yang digunakan dalam penentuan frekuensi kritis.Fungsi lainnya adalah menentukan permulaan masking untuk setiap sub-band.Masking adalah ketika satu suara membuat suara yang lain tidak dapat didengar,atau menutupi suara lain.Jadi berdasarkan model psikoakustik,encoder dapat menentukan untuk melakukan eliminasi atau mengurangi jumlah bit yang dialokasi untuk setiap suara dibawah permulaan masking untuk setiap sub-band.Faktor lain yang menentukan apakah yang dapat dihilangkan adalah bit rate dari encoding.Semakin rendah bit rate,semakin banyak data yang dihilangkan.

Aliran data dialirkan ke sepasang loop:noise control/distortion loop(control loop)dan rate loop.Keseluruhan aliran data dibagi menjadi beberapa potong,dan juga sub-band.Control loop mengecek setiap potongan untuk menentukan noise berada dalam tiap sub-band melebihi batas masking,atau noise yang diperbolehkan.Jika sub-band melebihi noise yang diperbolehkan,maka faktor skala untuk sub-band tersebut harus disesuaikan.Faktor skala menyediakan *gain* yang diperoleh untuk setiap sub-band.*Gain* adalah perbandingan sinyal keluaran dan masukan. Untuk mengganti syarat bit rate yang meningkat,*rate loop* diinisialisasikan.

Rate loop melakukan dan mengecek kode Huffman. Kode Huffman adalah metode menganalisa sebuah set data dan memroduksi sebuah set bit yang optimal untuk merepresentasikan data.Semakin sering sampel input digunakan,diberikan string bit yang lebih kecil,dengan demikian melakukan kompresi *lossless* pada data *lossy* dari porsi enkoding psikoakustik,dan tabel pengkodean Huffman yang berbeda digunakan untuk spektrum frekuensi yang berbeda untuk memperoleh pengurangan ukuran maksimum.Jika jumlah bit dari kode Huffman lebih besar dari bitrate yang diperbolehkan,maka jumlah potongan waktu dikurangi.

4. Peran kode Huffman

Data yang terkode terdiri dari 576 baris frekuensi tiap channel dan bagian kecil yang disimpan sebagai 16 bit signed integer.Sebagai contoh,akan diberi gambaran spektrum frekuensi pada file MPEG Layer 3:



Gambar 6. Frekuensi layer 3 yang terkuantisasi

Lihat dari 576 nilai numerik nya :

4, 24, −63, 79, −114, 55, 75,
−51, −13, 21, −1, −43, 94, −41, 22, 92, −231, −77, −4, −7, −26, −31, −11, 12, 52,
0, 2, −6, −12, −18, 6, −36, 159, −6, −25, −6, −43, −6, 3, −29, 14, −99, 64, −15, 7,
14, 9, −15, −52, −168, 25, 6,
1, 10, 3, 15, −28, 30, 38, 8, −3, −6, −5, 2, 5, 4, −9, 24, −6, −6, −1, 2, 13, −22,
−8, 12, 5, −5, −6, 12, −20, −6, −24, 7, −5, −5, 3, 0, 5, 3, −11, −5, −1, 6, −7, −6,
5, −2, 3, 5, 4, 2, 2, −2, 1, 3, −5, 14, 0, −8, −5, 0, 1, −4, 2, 3, 5, −1, 1, −1, 2, 0, 4,
−2, 1, 0, 0, −3, −3, −1, 0, −6, −5, −2, −1, 0, −1, −1, 3, 8, 2, 0, 2, 1, 1, 1, 0, 2, −4,
−3, −1, 2, 2, −1, −1, 0, 0, 0, 1, 1, 2, 2,
0, −2, 1, 0, −1, 2, 1, −1, 0, −1, 0, −1, −2, −1, 0, 0, 1, 1, −2, −2, 0, 0, 0, −1, 0, 0,
0, 1, −1, −1, 0, 0, 1, 0, 1, 1, −1, −1, 1, −1, 0, 2, 0, 0, −1, 0, 1, 2, 1, 2, −1, 2, 2, 3,
−1, 2, 2, 0, 0, −1, 1, 1, 1, 0, −2, 1, 0, 2, 1, 0, 0, −1, 1, 0, −1, 1, 0, −1, 0, 0, 1, 0, −1,
−1, 1, 1, 0, 1, 0, −1, 0, 1, 1, −1, 0, 0, 0, −1, −1, −2,
−1, 0, 0, −1, 0, 0, 0, 0, 0, 0, 1, −1, 0, 0, 0, 0, 0, −1, 1, −1, 0, 0, −1, 0, 1, 1, 0,
0, 1, 0, −1, 1, 0, 1, 0, 0, −1, 1, 0, 0, 1, −1, −1, 0, 0, 0, 1, 1, 0, 0, 0, −1, 0, 0, 0, −1,
1, 0, 0, 1, −1, −1, 1, 0, 0, −1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, −1, 1, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, −1, 0, 0, 0, 0, 0, 0, 0, 1, 1, −1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, −1, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, −1, 0, −1, 0, 0, −1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, −1,
0, 0, 0, 0, 0, 0, 0, −1, 0, 1, 0, 0, 0, −1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, −1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, −1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

Dapat dilihat dari bagian akhir paling atas spektrum,semua nilai nol diawali oleh banyak ”nilai kecil”.Bagian dari spektrum dari kiri nilai nol,dimana spektrum mulai diisi oleh nilai 1 dan -1 adalah bagian ”nilai kecil”.Nilai ini tidak dapat mendengar suara ini,jadi encoder biasanya menghilangkannya.

4.1. Melakukan Decoding pada Nilai Kecil

Untuk melakukan *decoding* pada nilai kecil,kelompokkan nilai tersebut menjadi 4 (u0,u1,u2,u3) dan melakukan encode pada kelompok tersebut.Sebenarnya,hanya nilai absolut,0 atau 1,yang di encode,dan kode untuk kelompok diikuti sampai 4 sign bit.Untuk setiap nilai tidak nol pada kelompok tersebut ada 1 sign bit.Jika bit itu 0,nilai positif,jika 1,nilainya negatif.

Encoder dapat memilih 2 tabel berbeda untuk encode nilai kecil,*htabA* dan *htabB.htabA* adalah tabel byte diindekskan oleh nilai 6 bit.Setiap byte terdapat nilai absolut dari (u0,u1,u2,u3) di bagian kiri dan nilai Huffman yang sebenarnya digunakan pada bagian kanan.Misalkan terdapat sebuah variabel yang bernama *huffman_cache*,yang mengandung *huffman_cache_size* bit rata kiri.Disini kita asumsikan bahwa kita mengisi huffman_cache yang memastikan setidaknya ada 16 bit yang valid dari huffman_cache.

```
code = htabA[((unsigned int) huffman_cache) >>
(HUFFMAN_CACHE_SIZE - 6)];

huffman_cache = huffman_cache << (code & #F);

huffman_cache_size = huffman_cache_size - (code &#F);
```

Setelah kita load *code* dari htabA,4 bit terkanan berisi bit yang diperlukan.Mereka dibutuhkan untuk mengatur huffman_cache.Nilai dari variabel *code* perlu diatur dengan tanda yang sesuai sebelum dimasukkan ke dalam array dari frekuensi raw **u.** Untuk menyimpan nilai ke dalam posisi yang benar,kita taruh pointer u kedala **u.**Untuk nilai yang sudah dikodekan yang bukan nol,kode Huffman diikuti oleh bit bertanda.Bit-1 ditandai dengan nilai -1,dan bit nol ditandai dengan +1.Daripada mengetes tiap bit secara terpisah,kita kombinasikan bit dari *huffman_cache* dan 4 bit dari *code* ke 1 nilai 8 bit yang kita gunakan sebagai index ke tabel yang bernama *signed_small_values* yang mengandung nilai dari u0,u1,u2 dan u3 serta nomor dari n bit yang digunakan

```
code = (code & #F0) | (((unsigned int)
huffman_cache >>
(HUFFMAN_CACHE_SIZE - 4));
*u++ = signed_small_values[code].u0;
*u++ = signed_small_values[code].u1;
*u++ = signed_small_values[code].u2;
*u++ = signed_small_values[code].u3;
```

4.2.Melakukan Decoding pada Nilai Besar

Bagian pertama dari spektrum frekuensi mengandung ”nilai besar”.Disini nilai dapat berkembang menjadi 2^{13} = 8192 dan meletakkannya pada sebuah tabel kode Huffman yang sebagian besar tidak pernah digunakan.Untuk menjadi lebih efektif, skema pengkodean digunakan untuk ”nilai besar” yang kecil, yaitu nilai dari 0 sampai 15, menggunakan tabel kode Huffman,dan diikuti di aliran bit oleh unsigned integer,yang ditambahkan ke 15 untuk nilai yang sangat besar. Untuk mengetahui jumlah bit yang digunakan dalam pengkodean unsigned integer tersebut,setiap tabel terdapat nilai bernama ”linbit”yang paling banyak 13.Ini memberikan nilai maksimum untuk nilai besar sebesar 2^{13} + 15 = 8207. Nilai 0-15 bisa di enkode dengan 4 bit karena kode Huffman setidaknya memiliki panjang 1 bit,ini memberikan nilai maksimum kompresi sebesar 25%. Untuk mengimprovisasikannya,nilai besar di encode dalam pasangan (u0,u1) yang membuat nilai kompresan bisa menjadi 12.5%.

Jika *htab* adalah sebuah tabel Huffman,ini adalah sebuah tabel dari *short integer* diindekskan oleh bit HWIDTH dari huffman_cache. Tiap *short integer* terdapat di 8bit paling kiri,4bit selanjutnya nilai u0 dan paling kanan u1.Karena tabel Huffman menggunakan ekstensi,kita tidak dapat menyimpulkan bahwa tiap short int di tabel memiliki bentuk seperti yang disebutkan,bisa jadi sebuah pointer.Dalam kasus ini , dia memiliki nilai negatif,dan 12 bit paling kiri memberikan perbedaan antara index awal dan tabel ekstensi. Perbedaan ini negatif,karena tabel ekstensi berada setelah tabel utama.4 bit sisa memberikan nomor index digunakan untuk mengakses tabel ekstensi.

Cara membaca dua nilai besar dengan *htab* :

```
int width = HWIDTH;
short int *h = htab;
code = h[((unsigned int)huffman_cache) >>
(HUFFMAN_CACHE_SIZE -width)];
while (code <0)
{huffman_cache = huffman_cache<<width;
huffman_cache_size = huffman_cache_size-width;
h = h - (code>>4);
width = code & #0F;
code = h[((unsigned int)huffman_cache) >>
(HUFFMAN_CACHE_SIZE -width)];
}
```

Setelah memperoleh kode kanan,kita ubah huffman_cache dan mask kode untuk memperoleh nilai u0 dan u1 .

```
huffman_cache = huffman_cache << (code >> 8);
huffman_cache_size = huffman_cache_size - (code >>
8);
code = code & #FF;
```

Untuk membaca bit bertanda , kita cukup mengetes bit paling kiri dari *huffman_cache* ,yang merupakan bit tanda nya, dan mengetesnya seperti : **if** (huffman_cache < 0) ... **else** ... . Tetapi, sebenarnya lebih penggunaan cabang seperti itu membutuhkan waktu yang agak lama.

Menentukan tanda dari nilai

```
{
int tmp = ((signed int) huffman_cache ) >>
(HUFFMAN_CACHE_SIZE -1);
value = (value Â tmp)-tmp;
huffman_cache = huffman_cache << 1;
huffman_cache_size--;
}
```

Kode diatas menggunakan properti dari nilai biner yang direpresentasikan dalam format 2’s complement. Mengubah bagian paling kiri dari *huffman_cache* ke kanna menghasilkan sebuah nilai integer yang semua bitnya diset ke satu jika bagian paling kiri 1,atau 0 jika bagian paling kiri 0. Kita simpan hasilnya ke *tmp* dan gunakan 2 kali: sebagai *mask* dari operasi Â,dan menguranginya.Dalam kasus *sign bit* sudah diatur, nilai ini merupakan sebuah operasi komplemen biner dan sebuah penambahan terhadap 1,yang merupakan definisi ari representasi 2’s complement.

5. KESIMPULAN

Ada beberapa kesimpulan yang dapat diperoleh. Pertama, pembentukan format MP3 menggunakan model psikoakustik dimana mencocokan data dengan kualitas pendengaran manusia.Kedua, kode Huffman berguna dalam pengurangan ukuran file data audio MP3.

REFERENSI

[1] Munir,Rinaldi ,"Diktat Kuliah IF 2091 Struktur Diskrit", STEI,ITB,2008

[2] Ruckert,Martin , "Understanding MP3: syntax, semantics, mathematics, and algorithms", Birkhäuser,2005

[3]http://www.omninerd.com/articles/How_Audio_Compression_Works . Tanggal akses : 19 Desember 2009

[4] http://en.wikipedia.org/wiki/Audio_file_format .Tanggal akses : 15 Desember 2009.

[5] http://en.wikipedia.org/wiki/Huffman_coding . Tanggal akses : 15 Desember 2009

[6] http://computer.howstuffworks.com/file-compression3.htm Tanggal akses : 15 Desember 2009

[7] http://www.mp3-converter.com/mp3codec/huffman_coding.htm Tanggal akses : 19 Desember 2009