PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK UNTUK MEMANFAATKAN KERTAS SEBAGAI MEDIA PENYIMPANAN DATA DIGITAL (PAPER DISK)

TUGAS AKHIR

Oleh :

MOCH. ANANG KARYAWAN
NRP. 26.90.100.039

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
1999

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK UNTUK MEMANFAATKAN KERTAS SEBAGAI MEDIA PENYIMPANAN DATA DIGITAL (PAPER DISK)

TUGAS AKHIR

Diajukan Untuk Memenuhi Persyaratan
Memperoleh Gelar Sarjana Teknik Informatika
Pada
Jurusan Teknik Informatika
Fakultas Teknologi Industri
Institut Teknologi Sepuluh November
Surabaya

Mengetahui/Menyetujui,

Pembimbing I	Pembimbing II
Dr. Ir. Arif Djunaidy, M.Sc.	Ir. Aris Tjahyanto, M.Kom.
NIP. 131 633 403	NIP. 131 933 299

Surabaya, Agustus 1999

Dengan Menyebut Asma Alloh Yang Maha Pengasih lagi Maha Penyayang.

Tugas Akhir ini kupersembahkan untuk
Bapak dan Ibu yang setia mengiringi langkahku dengan do'a keduanya,
serta sebelas saudaraku.
Mudah-mudahan Alloh mengumpulkan kita di surganya kelak.

KATA PENGANTAR

KATA PENGANTAR

Bismillahirrohmanirrohim.

Alhamdulillah, segala puji syukur penulis panjatkan ke hadirat *Alloh Subhanahu Wa Ta'ala*, Tuhan Yang Maha *Rohman* dan Maha *Rohim*, yang telah melimpahkan banyak sekali karunia-Nya kepada kita, khususnya kepada penulis sehingga penulis dapat menyelesaikan Tugas Akhir dengan judul:

PERANCANGAN DAN PEMBUATAN
PERANGKAT LUNAK UNTUK MEMANFAATKAN KERTAS
SEBAGAI MEDIA PENYIMPANAN DATA DIGITAL
(PAPER DISK)

Sholawat serta salam semoga tetap tercurahkan kepada junjungan kita, Nabi *Muhammad Shollallohu 'Alaihi Wasallam*, --Rosul mulia yang telah membimbing manusia dari lembah kejahiliyaan menuju cahaya kebenaran dan keimanan--, para sahabat, para tabi'in dan para pengikutnya yang setia menjalankan syari'at Islam sampai akhir zaman.

Tugas Akhir ini dikerjakan untuk memenuhi salah satu persyaratan akademis bagi mahasiswa untuk meraih gelar kesarjanaan di Jurusan Teknik Informatika, Fakultas Teknologi Industri, Institut Teknologi Sepuluh November Surabaya.

Berangkat dari artikel *Trend Teknologi* yang dimuat di Harian Umum Republika tahun 1997 dan pengalaman bertahun-tahun penulis membantu beberapa mahasiswa yang sedang mengerjakan Tugas Akhir, masalah yang paling sering dihadapi oleh mahasiswa tersebut adalah kehilangan data yang mereka simpan di

disket. Dari sini ide tersebut dikembangkan, karena untuk data yang disimpan di dalam media magnetis kita tidak tahu di posisi-posisi manakah data kita disimpan. Kita juga tidak tahu mana data yang 0 dan mana yang 1. Sehingga kalau sebuah data pada media penyimpanan tersebut rusak sedikit saja kita tidak tahu letak kesalahannya. Ini sangat jauh berbeda bila dibandingkan dengan data yang tercetak pada kertas yang ditandai dengan titik "hitam" dan "putih" untuk mewakili satu bit data, karena terlihat dengan jelas.

Walaupun dalam Tugas Akhir ini perangkat lunak yang telah dibuat berhasil melakukan proses penyimpanan data ke dalam kertas dan membacanya ulang dengan baik, namun masih banyak sekali kekurangan yang terdapat dalam Tugas Akhir ini. Karena itu segala saran dan kritik dari berbagai pihak sangat penulis harapkan. Akhirnya hanya kepada Alloh Subhanahu Wa Ta'ala penulis berlindung dari kesalahan dan kekhilafan serta berharap semoga tulisan ini bermanfaat bagi kita semua.

Ya Alloh, berikanlah manfaat dari apa yang telah Engkau ajarkan kepada kami
Dan ajarkanlah kepada kami ilmu yang bermanfaat
Dan tambahkanlah ilmu kepada kami
Dan berikanlah kami kemampuan bersyukur atas segala keadaan.
Amin, amin ya Robbal 'alamin.
Walhamdulillahirobbil'alamin.

Surabaya, Agustus 1999

Mochammad Anang K.

UCAPAN TERIMA KASIH

Terima kasih yang tidak terkira penulis sampaikan kepada banyak pihak yang telah membantu dan mendorong penulis untuk segera menyelesaikan Tugas Akhir ini, khususnya kepada :

1. Bapak Dr. Ir. Arief Djunaidy, M.Sc., selaku Ketua Jurusan Teknik Informatika, Dosen Wali sekaligus Dosen Pembimbing Tugas Akhir ini yang selalu konsisten mengingatkan penulis untuk cepat-cepat menyelesaikan Tugas Akhir, memberikan bimbingan akademis, dan bimbingan Tugas Akhir. Semoga amal Bapak yang ikhlas tersebut diterima dan dibalas oleh Alloh dengan pahala yang besar.
2. Bapak Ir. Aris Tjahyanto, M.Kom. selaku Dosen Pembimbing II, yang telah banyak memberikan pengarahan dan bimbingan kepada penulis sehingga muncul inspirasi saat-saat terjadi "*deadlock*" waktu mengerjakan Tugas Akhir. Terima kasih atas bimbingannya, semoga Alloh membalas dengan pahala yang besar.
3. Bapak dan Ibu Dosen Jurusan Teknik Informatika (Pak Supeno, Bu Handayani, Bu Esther, Pak Khakim, Pak Agus, Pak Dwi, Pak Iyan, Pak Victor, Pak Hari, Bu Nanik, Pak Rully, Pak Fajar, Pak Suhadi, Pak Djoko, Pak Husni, Pak Oscar, Pak Yudi) dan Bapak dan Ibu Dosen lain yang telah dengan ikhlas membagikan ilmunya kepada penulis selama masa kuliah.
4. Bapak dan Ibu Staff Tata Usaha Jurusan Teknik Informatika yang telah memberikan kelancaran administrasi kepada penulis selama kuliah.
5. Bapak dan Ibu tercinta yang selalu mengiringi setiap langkah penulis dengan do'a, cinta, kasih sayang yang tulus dan air mata.
6. Kakak-kakakku (Mas Usman, Mas Yono, Mas Anwar (*Makasih uang SPP-nya dulu*), Mas Fendy, Mas Edy, Mas Hari (*Bimbing istri sampeyan*), Mbak Lina (*Moga cepat terkabul do'a Bapak Ibu buat Sampeyan*), Mas Amin (*Moga cepat lulus S2-nya*), Mbak Atik (*Moga cepat pindah tugas ke Jawa dan Faldy punya*

Adik)) dan khusus buat Adikku Udin (*Moga jadi seorang penyiar*) yang selalu mendo'akan dan mendorong penulis untuk lulus.

7. K.H. Ihya 'Ulumiddin yang banyak memberikan dorongan semangat, do'a serta nasihat-nasihat khususnya menjelang Seminar Tugas Akhir.
8. Ustadz H. M. Rofi'i Hasan atas dorongan dan do'anya dan terima kasih atas dibangunkannya penulis di waktu malam.
9. Keluarga Paklik Ghofur, Paklik Mahmud dan Paklik Maksum terima atas segala do'anya yang tulus.
10. Teman-teman seperjuangan di Vde Group (Guk Umar, Cak Handaka, Mas Ainul, Akoer, Iksan, Sukar-jadi dan Cak Choes) yang telah banyak memberikan dorongan tidak sehat kepada penulis.
11. Teman-teman di Yayasan Al Haromain (Ust. Djunaidi, Mas Ilyas, Mas Agung, Mas Arif, Yuda dan lain-lain (*Jangan berhenti berdakwah dan ngajinya*).
12. Teman di MDF Al Mu'tashim (*Khususnya Syarifuddin, moga habis menikah tidak lupa Mu'tashimnya*) dan PP Nurul Haromain, terima kasih atas segala do'a yang tulus, khususnya saat menjelang Seminar Tugas Akhir.
13. Teman-teman Remush dan TPAI Nurul Hidayah (Farid, Imam, Nonok, Andrian, Toni, Wawan, dan Mbak-mbaknya) *Gimana TPAI? Kasihan adik-adik nggak ada yang ngajar.*
14. Adik-adik Pesma Baitul Hikmah, Hakim, Taufik dan Afif (*Moga sekolahnya tidak molor seperti saya*).
15. Teman-teman di Vde Press khususnya Sugi' yang banyak membantu ngetikkan naskah dan di Cano Print (Pak Indro, Parman, Pur dan Yoto) dan Dani Abadi Offset atas bantuan ilmu cetaknya.
16. Teman-teman di Cafe Vde khususnya Cak Tris beserta Nyonya yang memasak-kan penulis dengan "Tempe Penyet"nya.
17. Teman-teman seangkatan C06, khusus Pak Beh yang banyak membantu dalam mengerjakan TA dan lebih khusus Pak Pramu yang banyak telepon untuk mengingatkan TA (*Maaf aku maju duluan, mudah-mudahan Pak Arif man*

memberi dispensasi lagi buat kamu, insya Alloh tak ewangi).

18. Mas Ghofir yang telah menjanjikan untuk mentraktir teman-teman bila penulis lulus.
19. Buat Devis makasih atas buku Borland Delphinya, Mas Purnadi (*Moga cepat menikah, udah tua jangan mikir bisnis aja*).
20. Mas Noor Al 'Azam, terima kasih atas kemudahan yang diberikan kepada penulis selama melakukan Kerja Praktek di RadNet Surabaya.
21. Semua teman yang ikut Yudisium periode September, terima kasih atas dorongan semangatnya.
22. Buat Adik-adik di Madani terima kasih atas do'anya.
23. Buat Bowo makasih atas kesediaannya menjadi moderator.
24. Dan semua sahabat, teman, seseorang yang tidak mungkin penulis sebutkan satu per satu. Terima kasih semuanya.

DAFTAR ISI

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR TABEL

DAFTAR LAMPIRAN

DAFTAR LAMPIRAN

ABSTRAK

ABSTRAK

Pemakaian kertas sebagai penyimpan data digital sebenarnya sudah dilakukan sejak lama, tetapi pemakaiannya masih memiliki keterbatasan. Keterbatasan tersebut berupa kapasitas data yang mampu ditampung pada selembar kertas yang masih sangat sedikit. Dengan ditemukannya teknologi pencetak laser dan pembaca data (*scanner*), maka kedua alat tersebut dapat dimanfaatkan untuk meningkatkan kapasitas data yang mampu disimpan di kertas.

Untuk meningkatkan kapasitas penyimpanan data pada kertas, maka kertas digunakan sebagai media penyimpanan data digital bukan data analog. Sebagai sebuah data digital, maka data yang disimpan di kertas harus mewakili bit-bit data yang terdapat pada komputer. Pengaturan bit-bit data dilakukan dengan menggunakan aturan-aturan tertentu yang mampu mengatasi segala persoalan pada saat pembacaan, seperti gangguan (*noise*), ukuran yang berubah, kemiringan data, hilang atau bertambahnya titik-titik data (*spot*) pada kertas. Oleh karena itu, teknik-teknik pendeteksian kesalahan (*Error Detecting Code*) dan teknik pengoreksian kesalahan (*Error Correcting Code*) harus digunakan untuk mengatasi persoalan tersebut.

Hasil uji coba menunjukkan bahwa dengan mengatur lebar *spot* sebesar 5 x 5 *pixel* untuk resolusi 600 DPI, maka data yang mampu disimpan pada kertas berukuran A4 adalah 600 / 5 * 600 / 5 / 12 * 11 x 7.5 = 99.000 byte. Ukuran tersebut setara dengan kurang lebih 50 lembar naskah ketikan biasa. Dengan demikian mampu menghemat pemakaian kertas.

BAB I

PENDAHULUAN

BAB I

PENDAHULUAN

1.1. Latar Belakang

Teknologi komputer telah mengalami perkembangan yang sangat pesat dan pemakaiannya pun telah meluas ke berbagai sektor kehidupan. Pemakaiannya tidak lagi sekedar dilakukan oleh kantor-kantor besar, tetapi telah merambah ke rumah-rumah dengan munculnya komputer personal yang ada di hampir setiap rumah. Keberadaan komputer tersebut selain digunakan sebagai media hiburan, juga -- yang bahkan lebih banyak -- digunakan untuk menyimpan data.

Media penyimpan data yang banyak digunakan saat ini adalah media *magnetis* seperti disket dan harddisk. Selain itu sudah pula digunakan disk optik (*optical disk)*. Di antara media-media penyimpanan data yang ada, disket merupakan media yang paling banyak digunakan. Hal ini tidak mengherankan apabila ditinjau dari alasan berikut ini

1. Harganya murah
2. Fleksibilitas dan mobilitas yang tinggi

Selain memiliki kelebihan-kelebihan di atas, penggunaan disket sebagai media penyimpanan data memiliki kelemahan utama yaitu data mudah rusak atau hilang akibat kelalaian pengguna atau akibat kesalahan dalam penyimpanan. Dengan adanya kerusakan atau kehilangan data tersebut, maka secara otomatis fungsi disket

sebagai media penyimpanan tentu tidak berarti lagi.

Berangkat dari kondisi seperti tersebut di atas para ahli telah mengembangkan beberapa teknologi penyimpanan yang lebih baik; seperti *zip drive*, *optical drive* dan *compact disk*. Namun, semua media penyimpanan di atas masih memiliki beberapa kekurangan, diantaranya harga yang mahal dan masih mengandung resiko kehilangan data yang tinggi akibat kesalahan penyimpanan.

Untuk itu perlu dikembangkan suatu teknologi alternatif yang lebih murah namun memiliki daya tahan yang tinggi sebagai media penyimpanan. Salah satu media yang banyak dipakai sebagai media penyimpanan data dengan tingkat kehandalan yang tinggi adalah kertas.

Kertas banyak digunakan sebagai media penyimpanan data, seperti untuk arsip pegawai, arsip ujian atau untuk menyimpan data lainnya. Pemakaian di atas masih terbatas untuk penyimpanan data analog, sehingga kapasitasnya masih sangat terbatas bila dibandingkan dengan media penyimpanan digital yang lain.

Dengan ditemukannya teknologi pencetak *laser* dan *scanner*, maka kemampuan kertas yang masih sangat terbatas tersebut dapat ditingkatkan dengan memanfaatkannya sebagai media penyimpanan digital. Sebagai media penyimpanan digital, maka selembar kertas berukuran A4 (215,6 mm x 279,8 mm) diharapkan mampu menyimpan data setara dengan 80 halaman naskah dengan ukuran A4. Dengan demikian, hal ini akan mampu menghemat pemakaian kertas sekaligus mengurangi resiko kehilangan data.

Pada Tugas Akhir ini dibuat sebuah perangkat lunak untuk mengakomodasi kebutuhan media penyimpanan dengan memanfaatkan kertas sebagai media penyimpanan alternatif.

1.2. Permasalahan

Berangkat dari latar belakang di atas, maka ada 2 (dua) permasalahan pokok yang dapat dikemukakan, yaitu :

1. Pengkodean data komputer menjadi data format bitmap (*DataTile*).

   Merancang dan membuat metode pengkodean data komputer menjadi data format bitmap.

2. Metode pembacaan data bitmap (DataTile)

   Merancang dan membuat metode untuk menerjemahkan data-data yang telah tersimpan (tercetak) pada kertas ke bentuk data komputer (berupa file) kembali.

1.3. Tujuan dan Manfaat

Tujuan dari Tugas Akhir ini adalah pembuatan perangkat lunak untuk memanfaatkan kertas sebagai penyimpanan data digital. Sedang manfaat yang diperoleh adalah penghematan pemakaian kertas dan merupakan media penyimpanan data alternatif yang relatif aman dan murah.

1.4. Batasan Permasalahan

Berbagai batasan yang dipakai dalam Tugas Akhir ini adalah sebagai berikut :

- Format gambar yang digunakan adalah format standar dari Windows yaitu *Windows Bitmap* (BMP).
- Perangkat lunak ini hanya dapat bekerja pada Sistem Operasi Windows 95 dengan konfigurasi perangkat keras minimal 486 DX2 dengan memory 8 MB (disarankan 16 MB), monitor VGA, dan dilengkapi dengan printer dan scanner 600 DPI.
- Data hasil scanner mempunyai resolusi BitMap (*Black and White*) minimal 600 DPI.

1.5. Sistematika Pembahasan

Susunan buku Tugas Akhir tentang Perancangan dan Pembuatan Perangkat Lunak untuk Memanfaatkan Kertas sebagai Media Penyimpanan Data Digital (*Paper Disk*) adalah sebagai berikut ;

Bab I Pendahuluan, yang menerangkan Latar Belakang, Permasalahan, Tujuan dan Manfaat, Batasan Masalah dan Sistematika Pembahasan dari Tugas Akhir ini.

Bab II Landasan Teori, yang menerangkan teori-teori dasar yang digunakan dalam penyusunan Tugas Akhir seperti Teori Matriks, Teknik Pendeteksian Kesalahan (*Error Detecting Code*), Teknik Pengoreksian Kesalahan (*Error*

Correcting Code), Grafik, Pengaksesan Windows BitMap, dan Teknik Kompresi Data dengan Menggunakan Format ZIP.

Bab III Perancangan dan Pembuatan Perangkat Lunak, yang menerangkan tahap-tahap Perancangan dan Pembuatan Perangkat Lunak meliputi Disain Format DataTile, Struktur Data, Algoritma serta Disain Komponen dari perangkat lunak.

Bab IV Evaluasi dan Uji Coba, yang menerangkan hasil menjalankan (*running*) dari perangkat lunak serta evaluasi dari hasil tersebut.

Bab V Kesimpulan dan Saran, yang berisi tentang kesimpulan yang dapat ditarik serta saran-saran yang dapat diberikan tentang perbaikan Tugas Akhir ini.

BAB II
LANDASAN TEORI

BAB II

LANDASAN TEORI

Dalam Bab II ini dijelaskan tentang teori-teori dasar yang digunakan dalam penyusunan Tugas Akhir ini seperti Teori Matriks, Teknik Pendeteksian Kesalahan (*Error Detecting Code*), Teknik Pengoreksian Kesalahan (*Error Correcting Code*), Grafik, Windows BitMap dan Teknik Kompresi Data.

2.1. Matriks[1]

Matriks adalah susunan segi empat siku-siku dari bilangan-bilangan. Bilangan-bilangan dalam susunan ini disebut entri dari matriks atau elemen matriks. Matriks dapat digambarkan sebagai berikut :

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \cdot & \cdot & \cdot & \cdots & \cdots \\ \cdot & \cdot & \cdot & \cdots & \cdots \\ \cdot & \cdot & \cdot & \cdots & \cdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

Gambar 2.1.
Sistem Matriks

[1] Anton, Howard, *Elementary Linear Algebra*, John Wiley & Sons Ltd., 1996, hal. 54.

Ukuran matriks dapat ditetapkan dengan menyatakan jumlah kolom dan baris. Dengan demikian ukuran matriks di atas adalah m x n.

2.2. Sistem Koordinat

Sistem grafik selalu dipecah menjadi elemen-elemen yang paling kecil yang disebut pixel (*picture element*). Letak suatu pixel dibedakan dengan pixel yang lain berdasarkan lokasi pixel tersebut. Untuk membedakan lokasi atau posisi sebuah pixel digunakan sistem koordinat.

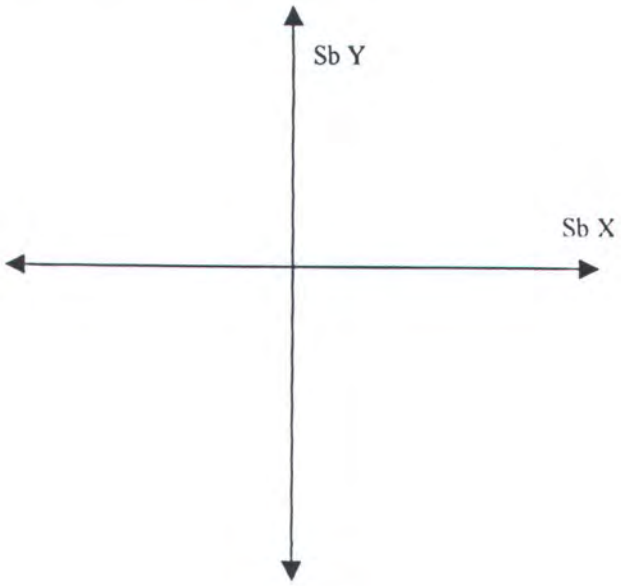
Sistem koordinat yang digunakan untuk menyatakan letak suatu pixel adalah sistem koordinat Kartesius dan koordinat sudut. Dalam sistem koordinat Kartesius, khususnya sistem koordinat Kartesius dua dimensi, letak sebuah pixel ditentukan oleh dua besaran. Nilai dari besaran tersebut apabila digambar akan membentuk suatu sumbu koordinat mendatar yang disebut *absis*, dinyatakan sebagai sumbu x, dan koordinat tegak yang disebut *ordinat*, dinyatakan sebagai sumbu y. Untuk lebih jelas dapat digambarkan sebagaimana pada Gambar 2.2.

2.3. Grafik

Grafik adalah suatu bentuk data yang berupa gambar. Grafik banyak digunakan dalam kehidupan karena grafik merupakan suatu representasi yang paling mudah untuk dimengerti.

Dalam dunia komputer, informasi visual disimpan sebagai sebuah *array* dari kumpulan bit. Setiap bit menggambarkan gerbang elektronik yang bernilai *on* atau

off. Setiap titik dalam sebuah gambar dipetakan ke dalam satu bit atau lebih dalam memory komputer. Gambar yang disimpan dan ditampilkan dengan cara ini disebut gambar *bitmap* atau bitmap sederhana.[2]



Gambar 2.2.
Sistem Koordinat Kartesius

Grafik bitmap terdiri atas kumpulan titik-titik yang disebut sebagai pixel. Dengan demikian, sebuah citra dapat digambarkan sebagai sebuah matriks dengan ukuran *m x n* pixel. Masing-masing pixel adalah elemen matriks tersebut dan dapat disebut sebagai elemen matriks (p[i,j]), dengan i dan j adalah skalar non negatif.

2.4. Teknik Pendeteksian Kesalahan

Pendeteksian kesalahan sangat diperlukan dalam Tugas Akhir ini, karena

[2] Dick Oliver, et. al., *Tricks of The Graphics Gurus*, Sams Publishing, 1993, hal.5.

seringkali hasil dari proses *scanning* gambar Data Tile tersebut menghasilkan gambar atau *image* yang tidak sama dengan image asli. Gambar hasil scanning tersebut apabila didekodekan kembali, maka akan menghasilkan data yang salah. Kesalahan tersebut mengakibatkan data dari file asli akan terkorupsi atau terbentuk *corupted file* sehingga hasil akhir file tersebut tidak dapat dibuka atau dijalankan ulang.

Untuk mencegah agar tidak terjadi *error* selama proses *decoding*, maka digunakan teknik pendeteksian kesalahan (*error detection tecnique*). Di antara teknik pendeteksian kesalahan yang umum digunakan adalah :

- Bit paritas (*parity bit*)
- *Block redudancy check*
- *Cyclic redudancy check*

2.4.1. Bit Paritas

Bit paritas merupakan metode yang paling sederhana, yaitu dengan menambah sebuah bit paritas (*parity bit*) pada akhir setiap *word* dalam sebuah *frame*. Contoh yang umum adalah transmisi data ASCII, di mana sebuat bit paritas ditambahkan untuk setiap 7-bit ASCII karakter. Nilai dari bit ini dipilih sehingga satu word mempunyai jumlah angka 1 genap (untuk *even parity*) atau jumlah angka 1 ganjil (untuk *odd parity*). Pada umumnya even parity digunakan untuk transmisi data *asynchronous* dan odd parity digunakan untuk transmisi data *synchronous*.[3]

---

[3] Stalling, William, *Data and Computer Comunications*, Macmillan Publishing Company, 1991, hal. 125.

Sebagai contoh untuk sebuah karakter ASCII G (1110001) jika menggunakan odd parity maka karakter tersebut akan ditambah dengan angka 1, sehingga ditransmisikan sebagai 11100011. Penerima akan menguji karakter yang diterima dan bila jumlah angka 1 adalah ganjil, maka diasumsikan tidak terjadi kesalahan. Jika satu bit atau sejumlah ganjil bit tersebut mengalami inversi (berubah menjadi 0 atau sebaliknya) maka penerima akan mendeteksi adanya kesalahan. Persoalan muncul apabila bit yang mengalami inversi tersebut dua atau sejumlah genap, maka kesalahan tidak akan terdeteksi.

Probabilitas kesalahan dapat dirumuskan sebagai berikut :

$$P_1 = \left(1 - P_B\right)^{N_B N_C} \qquad \ldots\ldots\ldots (2.1.)$$

$$P_2 = \sum_{k=1}^{N_C} \binom{N_C}{k} \left[\sum \binom{N_B}{j} P_B^j (1 - P_B)^{(N_B - j)} \right]^k [(1 - P_B)^{N_B}]^{(N_C - k)} \qquad \ldots\ldots\ldots (2.2.)$$

$$P_3 = 1 - P_2 - P_1 \qquad \ldots\ldots\ldots (2.3.)$$

di mana

N_B = jumlah bit per karakter (termasuk bit paritas)

N_C = jumlah karakter tiap frame.

2.4.2. Block Redudancy Check

Block redudancy check merupakan perbaikan dari bit paritas. Perbaikan tersebut dilakukan dengan menambah kumpulan bit paritas kedua sebagaimana terlihat pada gambar 2.3. Frame dipandang sebagai sebuah blok karakter yang

disusun dalam ruang dua dimensi. Untuk tiap karakter ditambahkan satu bit paritas, dan sebagai tambahan, sebuah bit paritas ditambahkan untuk setiap bit dengan posisi menyilang semua karakter. Dengan demikian terbentuk karakter tambahan dengan bit yang ke-i adalah bit paritas untuk bit ke-i dari semua karakter lain dalam blok. Proses penentuan bit ke-i tersebut dapat dinyatakan secara matematis menggunakan operasi $\oplus$ (*exclusive or*).

Exclusive or dari dua digit bilangan biner adalah 0 jika keduanya 0 atau keduanya 1, dan jika kedua digit tersebut berbeda, maka hasilnya adalah 1. Dengan demikian bit paritas dari setiap karakter pada baris bit paritas dapat dirumuskan sebagai berikut :

$$R_j = b_{1j} \oplus b_{2j} \oplus \cdot \cdot \cdot b_{nj} \qquad \ldots\ldots\ldots (2.4.)$$

di mana

R_j = bit paritas untuk karakter ke-j

b_{ij} = bit ke-i pada karakter ke-j

n = jumlah bit dalam karakter.

Persamaan di atas digunakan untuk menentukan bit paritas pada even parity. Sedang untuk paritas karakter cek (*parity character check*) digunakan rumus :

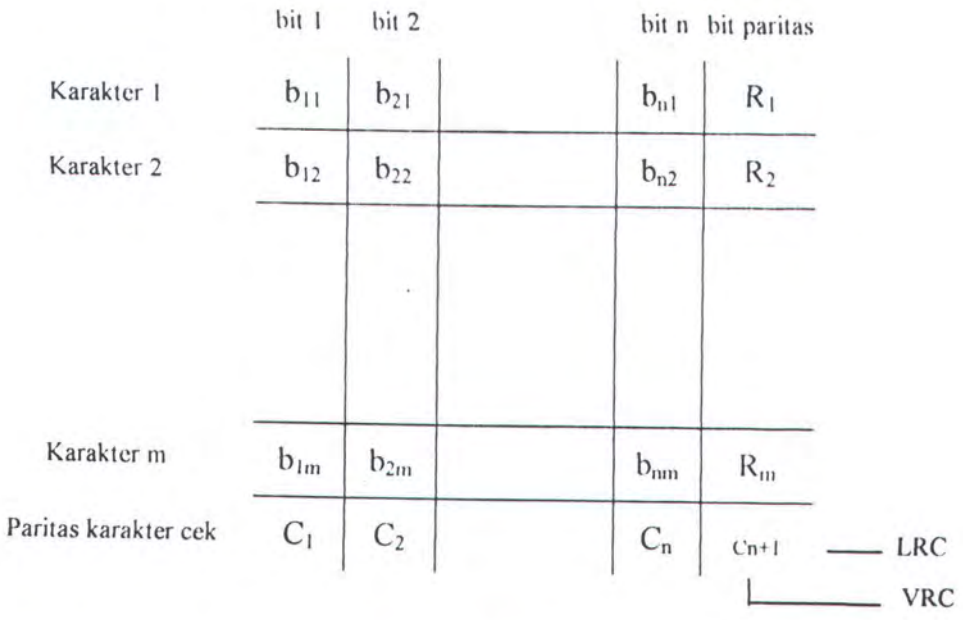
$$C_i = b_{i1} \oplus b_{i2} \oplus \cdot \cdot \cdot \oplus b_{im} \qquad \ldots\ldots\ldots (2.5.)$$

di mana

C_i = bit ke-i dari paritas karakter cek

m = jumlah karakter dalam satu frame.

Dalam format ini, bit-bit paritas pada akhir setiap karakter disebut sebagai *vertical redudancy check* (VRC) dan paritas karakter cek disebut sebagai *longitudinal character check* (LRC).



Gambar 2.3.
Format Block Redudancy Check

2.4.3. Cyclic Redudancy Check

Untuk mendapatkan peningkatan yang lebih baik, sebuah teknik yang sangat baik dan mudah diimplementasikan adalah *cyclic redudancy check* (CRC). Prosedur CRC dapat dijelaskan sebagai berikut:

Diberikan sebanyak k bit pesan atau frame, maka pemancar akan membangkitkan sebanyak n bit sebagai sequence, disebut sebagai *frame check sequence* (FCS), sehingga hasil akhir dari frame atau pesan yang terdiri atas $k + n$ bit dapat dibagi

dengan bilangan penentu (*predetermined number*). Penerima kemudian membagi frame yang datang dengan bilangan yang sama, dan jika tidak ada sisa (*remainder*), berarti tidak terjadi kesalahan.

Untuk lebih jelasnya, dapat digunakan prosedur-prosedur di bawah ini:

- Aritmatika modulo 2
- Polinomial
- *Shift register* dan gerbang exclusive-or

2.4.3.1. Aritmatika Modulo 2

Aritmatika modulo 2 menggunakan penjumlahan bilangan biner tanpa *carry generator*, hanya menggunakan operasi exclusive or, dan didefinisikan :

T = (k + n) bit frame yang akan ditransmisikan, dengan $n < k$

M = k-bit pesan, yaitu k bit dari T

F = n bit FCS, n bit terakhir dari T

P = kombinasi (*pattern*) dari $n + 1$ bit, disebut sebagai bilangan pembagi.

Jika T/P harus tidak mempunyai sisa, maka dapat dirumuskan bahwa:

$$T = 2^n M + F \qquad \ldots\ldots\ldots (2.6.)$$

Dengan mengalikan M dengan 2^n, berarti menggeser ke kiri (*shift left*) sebanyak n bit, dan mengisinya dengan 0. Penambahan F, sama dengan menggabung M dengan F. Jika T harus habis dibagi dengan P dan $2^n M$ dibagi P juga, maka :

$$\frac{2^n M}{P} = Q + \frac{R}{P} \qquad \ldots\ldots\ldots (2.7.)$$

Dari persamaan 2.7. diperoleh hasil (Q) bagi dan sisa (R). Karena pembagian biner, maka sisa (Q) selalu satu bit lebih sedikit dibandingkan dengan pembagi (P). Kemudian sisa (P) tersebut digunakan sebagai FCS, sehingga diperoleh :

$$T = 2^n M + R \qquad \ldots\ldots\ldots (2.8.)$$

Untuk membuktikan apakah R memenuhi persyaratan, maka persamaan 2.8. tersebut dibagi dengan T, dan diperoleh:

$$\frac{T}{P} = \frac{2^n M + R}{P} \qquad \ldots\ldots\ldots (2.9.)$$

Dengan substitusi persamaan 2.7. diperoleh :

$$\frac{T}{P} = Q + \frac{R}{P} + \frac{R}{P} \qquad \ldots\ldots\ldots (2.10.)$$

Dan menurut aritmatika modulo 2, setiap bilangan biner apabila ditambahkan dengan dirinya sendiri akan dihasilkan bilangan biner nol, sehingga persamaan 2.10. dapat ditulis:

$$\frac{T}{P} = Q + \frac{R + R}{P} = Q \qquad \ldots\ldots\ldots (2.11.)$$

Karena tidak terdapat sisa, maka T dapat dibagi habis dengan P, sehingga FCS dengan mudah dapat dihasilkan. Secara sederhana pembagian $2^n M$ dengan P menghasilkan sisa dan sisa tersebut digunakan sebagai FCS. Pada saat penerimaan (pembacaan), penerima akan membagi T dengan P dan tidak akan menghasilkan sisa bila tidak terjadi kesalahan.

Kombinasi P harus dipilih satu bit lebih lebih panjang dibandingkan dengan FCS yang diinginkan, dan kombinasi tersebut dipilih tergantung pada tipe kesalahan

yang diharapkan. Pemilihan kombinasi biner tersebut mempunyai ketentuan bahwa untuk kedua *most significant bit* (MSB) dan *least significant bit* (LSB) dari P harus 1.

Kesalahan yang terjadi dapat dengan mudah dideteksi. Kesalahan biasanya terjadi karena perubahan satu bit data. Secara matematis hal tersebut sama dengan melakukan operasi exclusive or pada bit tersebut dengan 1: $0 \oplus 1 = 1$; $1 \oplus 1 = 0$. Dengan demikian kesalahan dalam frame $(n + k)$ bit dapat direpresentasikan dengan sebuah *field* $(n + k)$ bit dengan 1 untuk setiap posisi kesalahan. Frame hasil T_r dapat dinyatakan dengan:

$$T_r = T + E \qquad \ldots\ldots\ldots (2.12.)$$

di mana

T = frame yang dikirim

E = kombinasi kesalahan dengan 1 dalam posisi di mana kesalahan muncul

T_r = frame yang diterima.

Penerima akan gagal mendeteksi sebuah kesalahan, jika dan hanya jika T_r habis dibagi P, dan jika dan hanya jika E juga habis dibagi P.

2.4.3.2. Polinomial

Proses CRC dapat dinyatakan sebagai polinomial dalam variabel *dummy* dengan koefisien biner. Koefisien-koefisien tersebut sesuai dengan bit-bit dalam bilangan biner. Dengan demikian untuk $M = 110011$, didapatkan $M(X)$ $X^5 + X^4 + X + 1$, dan untuk $P = 11001$ didapatkan $P(X)$ $X^4 + X^3 + 1$. Operasi aritmatika yang

digunakan adalah aritmatika modula 2. Proses CRC tersebut dapat dituliskan sebagai berikut :

$$\frac{X^n M(X)}{P(X)} = Q(X) + \frac{R(X)}{P(X)} \qquad \ldots\ldots\ldots (2.13.)$$

$$T(X) = X^n M(X) + R(X) \qquad \ldots\ldots\ldots (2.14.)$$

Sebuah kesalahan *E(X)* tidak akan terdeteksi jika habis dibagi dengan *P(X)*. Dan jika tidak habis dibagi *P(X)* maka kesalahan yang dapat dideteksi adalah :

1. Semua kesalahan bit tunggal.
2. Semua kesalahan bit ganda, selama *P(X)* mempunyai faktor sedikitnya tiga *term* (koefisien biner 1)
3. Beberapa kesalahan bit ganjil, selama *P(X)* mengandung faktor $(X + 1)$
4. Beberapa kesalahan panjang, dengan panjang kesalahan kurang dari panjang FCS.
5. Kesalahan umum yang lebih besar.

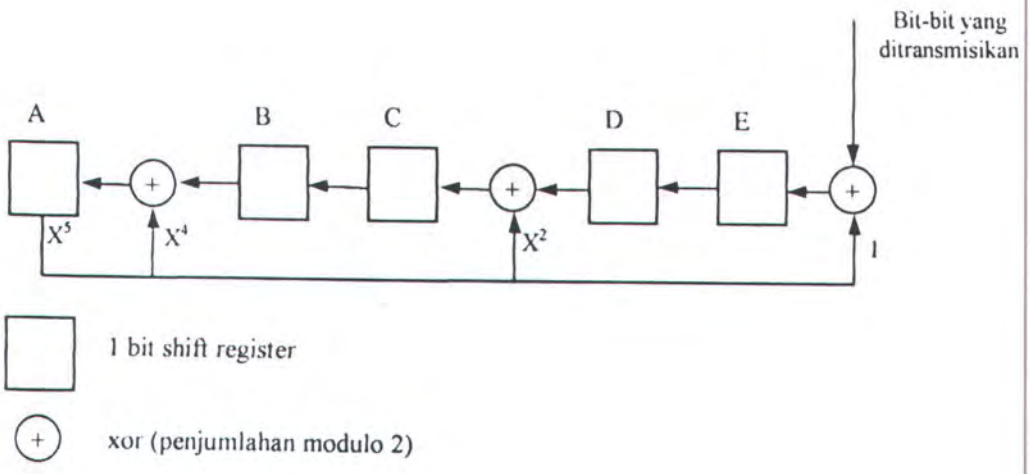
Beberapa versi untuk *P(X)* yang secara luas digunakan adalah :

- CRC-12 $= X^{12} + X^{11} + X^3 + X^2 + X + 1$
- CRC-16 $= X^{16} + X^{15} + X^2 + X + 1$
- CRC-CCITT $= X^{16} + X^{12} + X^5 + X + 1$
- CRC-32 $= X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$

2.4.3.3. Shift Register dan Gerbang Exclusive-OR

Proses CRC dengan mudah dapat diimplementasikan dengan membuat rang-

kaian yang terdiri atas gerbang exclusive or (xor) dan shift register. Gambar rangkaian tersebut dapat dilihat pada gambar 2.4.



Gambar 2.4.
Rangkaian Shift Register dengan Gerbang XOR
untuk Pembagian dengan Polinomial ($X^5 + X^4 + X + 1$)

Rangkaian tersebut diimplementasikan aturan sebagai berikut :

1. Register terdiri atas *n* bit, sama dengan panjang dari FCS
2. Terdapat sampai *n* gerbang exclusive-or (XOR)
3. Jumlah gerbang tergantung dari jumlah term dalam polinomial pembagi *P(X)*.

Proses dimulai dengan shift register dinolkan (*clear*). Pesan atau pembilang dimasukkan satu bit, dimulai dengan MSB. Karena tidak ada umpan balik (*feed back*) sampai terdapat bit pembilang bernilai 1 pada register terakhir, empat operasi awal adalah operasi shift biasa. Pada saat satu bit 1 data di register terakhir, 1 dikurangkan (exclusive-or) dari bit kedua, dan kelima pada shift register berikutnya. Hal tersebut

identik dengan proses pembagian biner sebelumnya. Proses dilanjutkan untuk semua bit dalam message ditambah dengan empat bit 0. Proses akhir tersebut bertujuan untuk menggeser *M* ke posisi kiri untuk mendapatkan FCS. Setelah proses selesai, maka register akan berisi sisa hasil bagi (*reminder*) atau FCS yang akan ditransmisikan.

Proses yang sama dilakukan pada saat pembacaan (penerimaan). Setiap bit data dimasukkan dalam shift register A. Jika tidak terjadi kesalahan, shift register harus mengandung kombinasi untuk *R* sebagai kesimpulan dari *M*. Bit-bit *R* yang dikirim mulai datang, dan akibatnya adalah meng-nol-kan semua register. Sehingga pada saat menerima, maka semua register berisi angka 0.

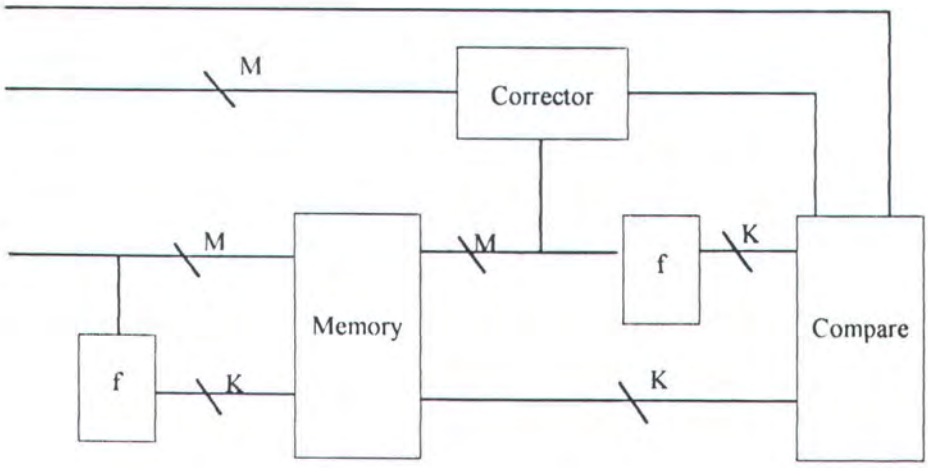
2.5. Teknik Pengoreksi Kesalahan

Hampir semua sistem memori utama modern memiliki logik untuk mendeteksi dan mengoreksi error-error yang terjadi, baik error berat maupun ringan.[4]

Ketika data dibaca ke dalam Memori, data tersebut dihitung dengan sebuah fungsi *f*. Dari hasil perhitungan tersebut terbentuk kode. Baik data maupun kode tersebut kemudian disimpan. Jadi, apabila sebuah data M bit akan disimpan dan kode memiliki panjang K bit, maka ukuran data sesungguhnya yang disimpan adalah M + K bit (gambar 2.5.).

---

[4] Stalling, Wiliam, *Computer Organization and Architecture, 4e: Designing for Performance*, Prentice-Hall Inc., 1996, hal. 117.



Gambar 2.5.
Fungsi Koreksi Kode Kesalahan

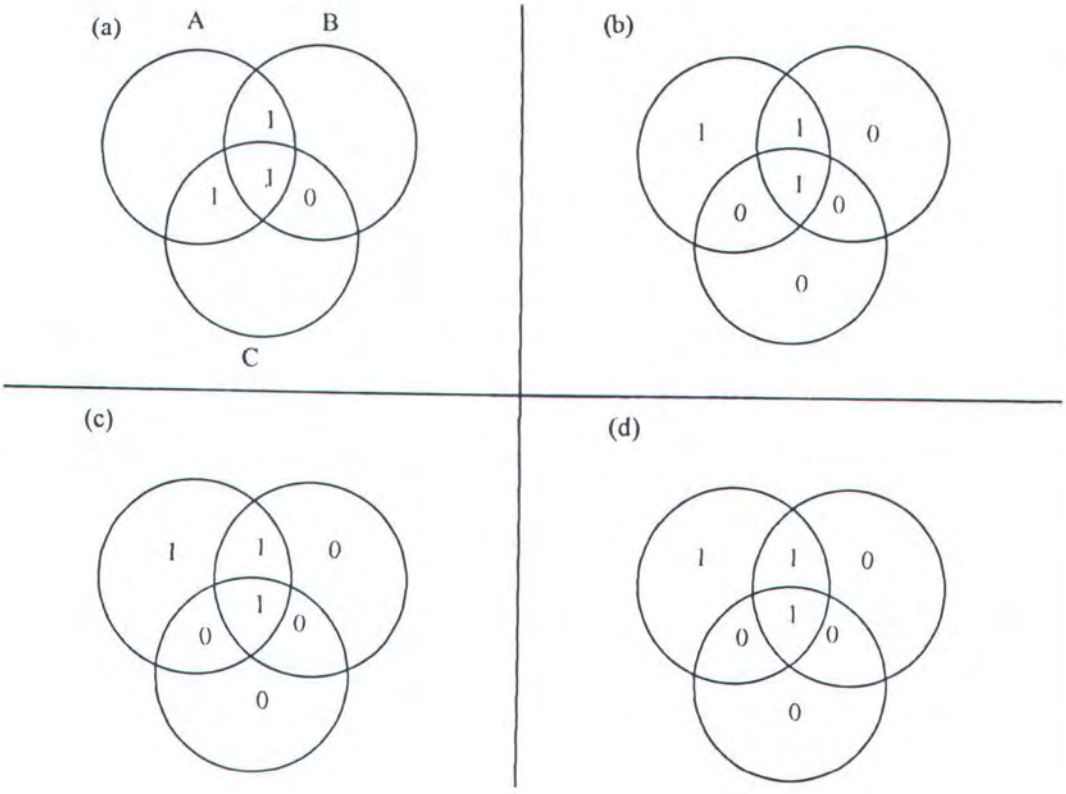
Ketika data yang tersimpan akan dibaca, maka kode tersebut digunakan untuk mendeteksi dan mengoreksi error yang mungkin terjadi. Bit-bit kode K yang dihasilkan dari M bit data dibandingkan dengan bit kode yang diambil dari M+K bit. Hasil perbandingan tersebut akan menghasilkan salah satu dari tiga kemungkinan berikut :

- Tidak terdapat error. Data bit yang diambil akan dikirimkan.
- Terjadi error, dan dimungkinkan untuk dikoreksi. Bit-bit data dan bit-bit koreksi error dimasukkan ke korektor sehinga menghasilkan bit-bit M koreksi yang akan dikirimkan.
- Terjadi error, namun tidak dimungkinkan untuk mengoreksinya. Keadaan ini cukup dilaporkan.

.Kode yang dihasilkan tersebut dinamakan sebagai *Error Correcting Code* (kode pengoreksi kesalahan), yaitu kode dengan sejumlah error bit dalam data yang dapat dideteksi dan dikoreksi.

Kode perbaikan error yang paling sederhana adalah kode Hamming yang diciptakan oleh *Richard Hamming* di Laboratorium Bell. Kode Hamming menggunakan diagram Venn untuk menjelaskan penggunaan kode ini untuk data 4 bit (M=4) (gambar 2.6.). Dengan tiga buah lingkaran yang berpotongan, terdapat tujuh kompartemen. Untuk kompartemen tengah diberikan 4 buah bit data (gambar 2.6.a). Sedangkan kompartemen sisanya diisi dengan bit paritas. Setiap bit paritas dipilih sehingga bilangan 1 di dalam lingkaran berjumlah genap (gambar 2.6.b). Jadi, karena lingkaran A terdiri dari 3 buah data bilangan 1, maka bit paritas di dalam lingkaran itu diisi 1. Apabila suatu error mengubah salah satu bit data (gambar 2.6.c), maka error itu akan dengan mudah dideteksi. Dengan memeriksa bit paritas, kesalahan berhasil ditemukan pada lingkaran A dan C, namun tidak pada B. Karena itu, error dapat dikoreksi dengan mengubah bit tersebut.

Untuk menjelaskan konsep tersebut, berikut diberikan contoh dengan membuat kode yang dapat mendeteksi dan mengoreksi error bit tunggal di dalam data 8 bit. Langkah pertama adalah dengan menentukan dulu panjang kode sesungguhnya. Dari gambar 2.5. blok *compare* menerimanya input dengan 2 nilai K bit. Perbandingan bit demi bit dilakukan dengan operasi exclusive-or kedua input tersebut. Hasilnya disebut sebagai *syndrome word*. Syndrome word digunakan untuk menentukan terdapat kesalahan atau tidak.



Gambar 2.6.
Diagram Venn Koreksi Kesalahan Hamming

Dengan demikian, syndrome word dengan lebar K bit memiliki jangkauan 2^K yang berada antara 0 dan 2^K - 1. Nilai 0 berarti bahwa tidak terdeteksi error, sedang yang menyisakan 2^K - 1 mengindikasi bit mana yang mengalami error, bila terdapat error. Karena suatu error dapat terjadi pada sembarang M bit data atau K check bit, maka panjang data harus memiliki :

$$2^K - 1 \geq M + K \qquad \ldots\ldots\ldots (2.15.)$$

Persamaan ini menunjukkan jumlah bit yang diperlukan untuk mengoreksi

error bit tunggal di dalam sebuah data dengan panjang M bit. Tabel 2.1. merupakan daftar panjang *check bit* yang diperlukan untuk bermacam-macam panjang data.

Tabel 2.1.
Panjang Check Bit untuk Beberapa Panjang Data

Data bits	Single error correction		Single error correction Double error correction	
	Check bits	% Increase	Check bits	% Increase
8	4	50.00	5	62.50
16	5	31.25	6	37.50
32	6	18.75	7	21.88
64	7	10.94	8	12.50
128	8	6.25	9	7.03
256	9	3.52	10	3.91

Dari tabel 2.1. diketahui bahwa data 8 bit memerlukan 4 check bit. Berikut diturunkan syndrome 4 bit dengan memakai ketentuan di bawah ini :

- Bila syndrome semuanya berisi 0, maka tidak terdapat error
- Bila syndrome berisi sebuah dan hanya sebuah bit yang diubah ke 1, maka telah terjadi suatu error pada salah satu dari 4 buah bit check
- Bila syndrome berisi lebih dari sebuah bit yang berubah ke 1, maka nilai numerik syndrome mengindikasikan posisi bit data yang mengalami error. Bit data ini kemudian diinversikan untuk keperluan koreksi.

Bit position	Position Number				Check bit	Data bit
12	1	1	0	0		M_8
11	1	0	1	1		M_7
10	1	0	1	0		M_6
9	1	0	0	1		M_5
8	1	0	0	0	C_8	
7	0	1	1	1		M_4
6	0	1	1	0		M_3
5	0	1	0	1		M_2
4	0	1	0	0	C_4	
3	0	0	1	1		M_1
2	0	0	1	0	C_2	
1	0	0	0	1	C_1	

Gambar 2.7.
Letak Bit Data dan Bit Check

Untuk memperoleh ketentuan tersebut, maka bit data dan check bit diatur menjadi word 12 bit seperti yang terlihat pada gambar 2.7. Posisi-posisi bit diberi nomor 1-12. Posisi-posisi bit yang memiliki bilangan pangkat 2 tersebut dikenal sebagai bit *check-bit*. *Check-bit* tersebut dihitung persamaan 2.16. – 2.19., dengan simbol $\oplus$ menandakan operasi *exclusive-or*.

$$C_1 = M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7 \quad \ldots\ldots\ldots (2.16.)$$

$$C_2 = M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7 \quad \ldots\ldots\ldots (2.17.)$$

$$C_4 = M_2 \oplus M_3 \oplus M_4 \oplus M_8 \ldots\ldots\ldots (2.18.)$$

$$C_8 = M_5 \oplus M_6 \oplus M_7 \oplus M_8 \ldots\ldots\ldots (2.19.)$$

Setiap check bit menunjukkan posisi bit data yang nomor posisinya berisi bilangan 1. Jadi, posisi-posisi bit data 3, 5, 7, 9, dan 11 semuanya berisikan suku 2^0, posisi-posisi bit 3, 6, 7, 10, dan 11 semuanya berisi suku 2^1; posisi-posisi bit 5, 6, 7, dan 12 seluruhnya berisi suku 2^2 ; dan posisi-posisi bit 9, 10, 11 dan 12 berisi suku 2^3. Ditinjau dari sisi lain, posisi bit n diperiksa oleh bit-bit C_i, sehingga $\sum i = n$. Misalnya, posisi bit 7 diperiksa oleh bit-bit yang berada pada posisi 4, 2 dan 1; dan 7 = 4 + 2 + 1.

Contoh data dengan panjang input 8-bit berisi 00111001, dengan bit data M1 berada pada posisi paling kanan. Maka perhitungannya adalah sebagai berikut :

$C_1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$

$C_2 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$

$C_4 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$

$C_4 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$

Bila bit data 3 (M_3) mengalami error dan berubah dari 0 menjadi 1, maka akan diperoleh check bit:

$C_1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$

$C_2 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$

$C_4 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$

$C_4 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$

Bila check bit yang baru dibandingkan dengan check bit-bit yang lama, maka akan terbentuk syndrome word :

	C_8	C_4	C_2	C_1
	0	1	1	1
$\oplus$	0	0	0	1
	0	1	1	0

Hasilnya adalah 0110, yang menunjukkan bahwa posisi bit 6 (bit data ke-3) mengalami error (kesalahan).

Kode tersebut dikenal sebagai kode *single error correcting* (SEC). Umumnya, memori semikonduktor dilengkapi dengan kode *single error correcting, double error correcting* (SEC-DEC).

2.6. BitMap Windows

Microsoft Windows mendefinisikan format file BitMap tersendiri, yang banyak digunakan untuk gambar-gambar kecil (*icon*) atau untuk latar belakang layar (*wallpaper*). Format ini mendukung data dengan ukuran 1, 4, atau 8 bit per pixel, dan untuk versi terakhir mendukung 24 bit.

2.6.1. Struktur Data File BMP

Borland Delphi mendefinisikan data file BMP dengan sebuah *class* dengan nama TBitMap. Untuk mendeklarasikan data dengan tipe BitMap pada Borland Delphi cukup dengan menuliskan :

```
myBMP: TBitMap;
```

Dengan deklarasi *class* tersebut, tidak diperlukan lagi deklarasi type data untuk mengakes atau membuka file BitMap. Semua perintah untuk mengakses data BitMap, baik berupa prosedur, fungsi dan variabel sudah tercakup dalam class `TBitMap`.

Prosedur `Create` pada TBitMap digunakan untuk inisialisasi class TBitMap. Sebagai contoh untuk membuka file "Buku01.BMP" dari disk, maka perintahnya adalah :

```
Bitmap := TBitmap.Create;
Bitmap.LoadFromFile('Buku01.BMP');
```

Setelah sebuah data BitMap selesai dibuat, maka semua fungsi dan prosedur dapat dipanggil.

2.6.2. Class `TCanvas` pada TBitMap

Program Delphi dalam menghasilkan keluaran grafik menyediakan *object* yang dikenal sebagai *Canvas*. Canvas adalah sebuah object dengan sejumlah *property*, fungsi dan prosedur (disebut *methods*) untuk mengontrol keluaran grafik ke peralatan grafik seperti layar display, printer atau plotter. Canvas mengemas interaksi dengan mesin di Windows yang dikenal sebagai *Graphic Device Interface* (GDI).

GDI berisi kumpulan fungsi-fungsi (*subroutines*) untuk keluaran grafik dan struktur data sehingga sebuah aplikasi dapat menampilkan informasi secara visual. GDI adalah fasilitas penting untuk menghasilkan tampilan grafis dalam Windows.

Karena operasi langsung melalui GDI sangat kompleks, maka konsep Canvas, yang diimplementasikan dengan class `Tcanvas`, mengemas kompleksitas GDI tersebut menjadi lebih sederhana.

Salah satu komponen grafik yang penting adalah *pixel* (*picture element*). Pixel menunjukkan sebuah titik, atau kotak dengan ukuran paling kecil. Untuk melakukan operasi pixel pada Delphi dapat menggunakan fungsi yang terdapat pada class `TCanvas`. `TCanvas` dapat digunakan untuk operasi menggambar obyek pada permukaan bidang gambar. `TCanvas` menyediakan fungsi, prosedur dan property yang digunakan untuk proses manipulasi grafik. Class `TBitMap` secara otomatis memasukan class `TCanvas` dalam deklarasi tipe datanya.

Untuk mengakses fungsi, prosedur, atau property pada property `Canvas`, dapat dilakukan dengan menyebutkan variabel data yang bertipe class `TBitMap` dihubungkan dengan titik (.), diikuti dengan `Canvas`, dihubungkan dengan titik lalu disebutkan fungsi atau prosedur yang dibutuhkan. Sebagai contoh untuk membuat sebuah titik pada lokasi koordinat (100, 200) dan membuat kotak dengan koordinat (50, 50) dan (250, 150), perintah yang digunakan adalah :

```
BitMap.Canvas.Pixels[100,200]:=clBlack;
BitMap.Canvas.Rectangle(50,50,250,150);
```

Sedang untuk melihat isi atau warna pixel dengan koordinat (x, y) dapat dilakukan dengan membaca property `Pixels` yang terdapat pada class `TCanvas`. Sebagai contoh untuk melihat warna pixel dengan koordinat (270, 290) maka perintah

yang digunakan adalah :

```
myClr:= BitMap.Canvas.Pixels[270, 290];
```

Type data pengembalian operasi di atas adalah `integer`, hal tersebut karena Delphi mendefinisikan tiap pixel dalam BitMap sebagai 3-byte RGB, yang merepresentasikan intensitas warna *red, green, blue* (merah, hijau, biru) sebagai warna dasar dalam cahaya.

2.6.3. Mencetak File BitMap

File BitMap yang telah dibuat dapat dicetak printer dengan menggunakan object Canvas juga. Untuk mencetak file BitMap, langkah-langkahnya adalah sebagai berikut :

- Deklarasi variabel dengan tipe `TImage`
- Inisialisasi variabel tersebut
- Ambil file BitMap yang akan dicetak
- Akses *object* `Printer` dengan perintah :

```
Printer.BeginDoc;
Printer.Canvas.Draw(X,Y:Integer; Graphic:TGraphic);
Printer.EndDoc;
```

2.6.4. Pengaksesan Scanner

Salah satu alat input yang penting untuk memasukkan data grafik ke dalam

komputer adalah *scanner*. Selain menggunakan fungsi-fungsi dalam *Aplication Programming Interface* (API) untuk mengakses scanner, dapat juga menggunakan protokol TWAIN yang disediakan Windows.

Protokol TWAIN dibuat untuk menghindari ketergantungan sebuah aplikasi akan *device interface* bagi peralatan input grafik yang berbeda-beda. Dengan protokol TWAIN, setiap scanner atau peralatan input grafik yang memenuhi standar protokol TWAIN dapat diakses dengan mudah.

2.7. Teknik Kompresi Data

Teknik kompresi data dikembangkan karena ruang penyimpanan pada komputer boleh dikatakan terbatas, walaupun hard disk dengan ukuran *giga byte* telah ada di pasaran. Hard disk dengan kapasitas besar tersebut apabila digunakan untuk menyimpan data dan program komputer suatu saat akan penuh. Jika hard disk sudah penuh, maka data tidak dapat masuk. Salah satu pemecahan persoalan di atas adalah dengan membeli hard disk baru, tetapi ini memerlukan dana yang besar. Dari sini muncul ide untuk memampatkan (*compress*) data yang telah ada dan dibongkar (*expand*) pada saat diperlukan.

Kemudian ditemukanlah metode kompresi data. Dengan metode ini semua data yang terdapat pada media penyimpanan dapat dimampatkan besarnya, sehingga ruang penyimpanan yang tersisa semakin besar. Di antara metode kompresi data yang digunakan adalah :

- Algoritma *Run-Length* atau *Simple Coding*
- Algoritma *Lempel-Ziv*
- Algoritma Pohon *Huffman*
- Algoritma Aritmetika
- Algoritma LZRI
- Algoritma LZHUF

Di antara algoritma-algoritma tersebut, yang paling banyak digunakan adalah algoritma Lempel-Ziv. Algoritma ini kemudian dikembang oleh ilmuwan komputer sehingga muncul berbagai cabang algoritma Lempel-Ziv, di antaranya adalah LZSS (Lempel-Ziv-Storer-Szymanski), LZW (Lempel-Ziv-Welch), LZAR (Lempel-Ziv-Arithmetic) dan LZHUF (Lempel-Ziv-Huffman).

Algoritma LZW banyak dipakai karena cepatnya proses kompilasi yang dilakukan, hal ini terlihat pada perangkat lunak PKZIP yang begitu cepat dalam mengkompresi file. Tetapi kelemahan utama algoritma ini adalah hasil kompresi yang relatif kurang kecil (untuk versi awal PKZIP). Dari software PKZIP ini kemudian muncul perangkat lunak yang lain dengan menggunakan format standar dari PKZIP.

Pada saat muncul sistem operasi Windows, perangkat lunak pengkompresi data ini semakin berkembang dengan munculnya program-program pengkompresi data dan file dalam lingkungan Windows. Salah satu yang banyak dipakai adalah WinZip. Dengan munculnya sistem operasi Windows 95 dan NT yang mendukung penamaan file dengan panjang 255 karakter, perbaikan pada perangkat lunak

pengkompresi data dilakukan dengan keluarnya perangkat lunak pengkompresi data 32 bit yang mendukung penamaan file 255 karakter.

BAB III
PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK

BAB III
PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK

Perangkat lunak bertujuan untuk mengubah format data biner pada komputer menjadi format data biner pada kertas (DataTile) dan sebaliknya. Untuk mencapai tujuan tersebut, tahap yang perlu adalah Perancangan Perangkat Lunak yang meliputi Disain Format DataTile beserta Struktur Datanya, Algoritma dan Prosedur, dan Disain Antarmuka dari perangkat lunak, dan Pembuatan Perangkat Lunak yang merupakan implementasi dari perancangan yang telah dibuat.

3.1. Perancangan Perangkat Lunak

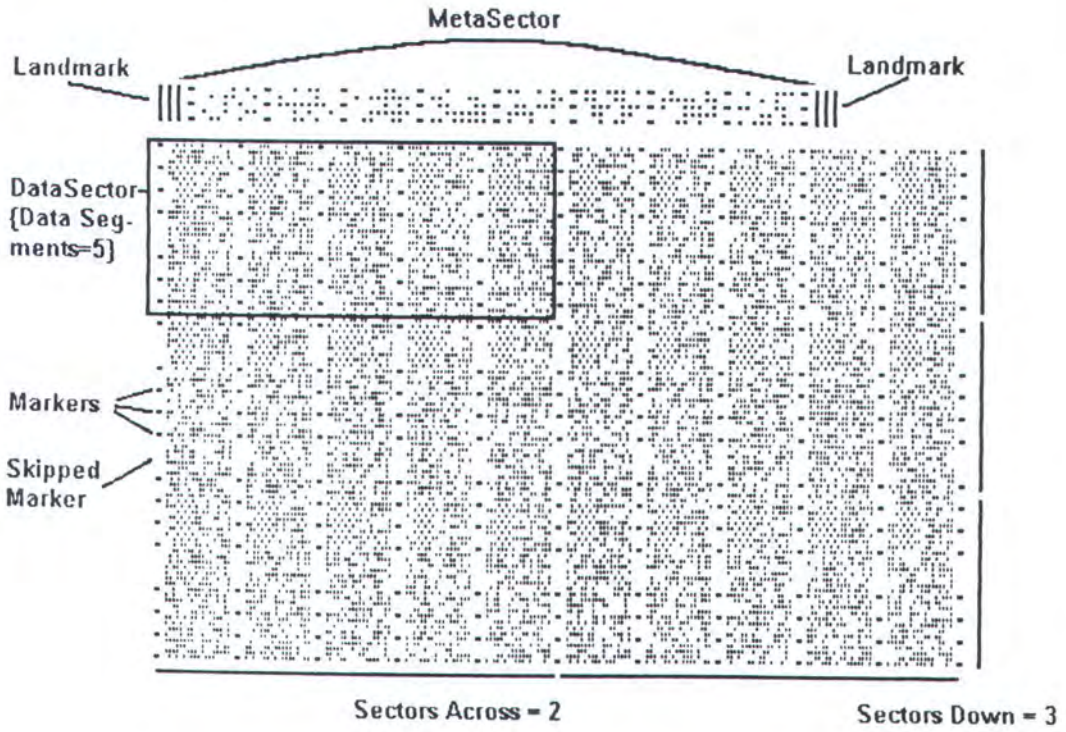
Beberapa hal yang perlu dipersiapkan dalam pembuatan perangkat lunak adalah Perancangan Format DataTile, Struktur Data, Disain Proses dan Algoritma, serta Disain Antarmuka.

3.1.1. Perancangan Format DataTile

Data yang tersimpan di media komputer adalah data digital yang dinyatakan dalam satuan terkecilnya adalah BIT, singkatan dari *binary digit*. Satuan data terkecil (BIT) tersebut kemudian diterjemahkan menjadi sebuah titik dalam kertas. Satuan data terkecil pada kertas tersebut diberi nama *spot*. Kertas yang terdiri atas kumpulan titik tersebut dengan aturan atau format tertentu tersebut disebut sebagai *DataTile*

(ubin data). DataTile ini dibuat dalam format Windows BitMap hitam putih (*Black and White*/BW), kemudian dicetak, dan dibaca ulang melalui scanner. Data BitMap hasil *scanning* tersebut kemudian dikenali kembali oleh komputer.

Agar data BitMap tersebut dapat dikenali secara otomatis oleh program, maka DataTile tersebut dibagi menjadi beberapa bagian. Bagian-bagian dari DataTile ditunjukkan dengan gambar 3.2.



Gambar 3.1.
Format DataTile

Dari gambar 3.1., format DataTile dibagi menjadi dua bagian utama, yaitu :

- **Meta Sector**

  Meta Sector berisi data utama dari DataTile, berisi semua variabel-variabel yang

diperlukan dalam proses pengenalan pada bagian *Data Sector*. Bagian Meta Sector ini berisi *Land Mark*, *Meta Sector Mark*, dan *Meta Sector Data*. Land Mark berfungsi untuk petunjuk awal (*starting point*) dalam pengenalan Meta Sector. Meta Sector Mark digunakan untuk petunjuk pada saat pengenalan data pada Meta Sector Data. Semua data untuk Meta Sector mempunyai ukuran tetap yang disesuaikan dengan jenis printer yang akan digunakan. Meta Sector berisi tentang data variabel yang digunakan untuk membaca *spot* pada DataTile. Variabel-variabel tersebut adalah:

- **Spot Height**

  Variabel ini digunakan untuk menentukan tinggi setiap spot ketika dicetak untuk setiap titik data (*dot*). Setiap titik tersebut mewakili satu bit.

- **Spot Length**

  Variabel ini digunakan untuk menentukan panjang atau lebar setiap spot ketika dicetak untuk setiap titik data.

- **Spot Distance**

  Variabel ini digunakan untuk menentukan jarak antara sisi kanan sebuah spot ketika dicetak dengan dengan spot berikutnya dalam satu *Data Segment*.

- **Segments Per Marker**

  Variabel ini digunakan untuk menentukan jumlah baris data antara dua buah *Marker*.

- **Data Segment**

Data Segment adalah kumpulan sel-sel data yang terdapat di antara dua buah Marker.

- **Marker Height**

  Variabel ini digunakan untuk menentukan tinggi dari sebuah Marker. Marker digunakan sebagai petunjuk terhadap barisan data di sebelahnya.

- **Marker Length**

  Variabel ini digunakan untuk menentukan panjang atau lebar dari sebuah Marker.

- **Row Distance**

  Variabel ini digunakan untuk menentukan jarak antara dua baris data (barisab spot) yang diukur dari puncak sebuah baris data dengan baris data berikutnya.

- **Marker to Spot**

  Variabel ini digunakan untuk menentukan jarak antara tepi Marker dengan sel data yang paling dekat.

- **Sectors Down**

  Variabel ini digunakan untuk menentukan jumlah Data Sector yang terdapat dalam DataTile. Semakin besar jumlah Data Sector, maka gambar hasil scan yang harus didekodekan semakin besar.

- **Spots Per Segment**

  Variabel ini digunakan untuk menentukan jumlah spot, berupa titik (dot) atau kosong (*blank*) antara dua Marker.

- **Sectors Across**

  Variabel ini digunakan untuk menentukan jumlah Data Sector pada DataTile.

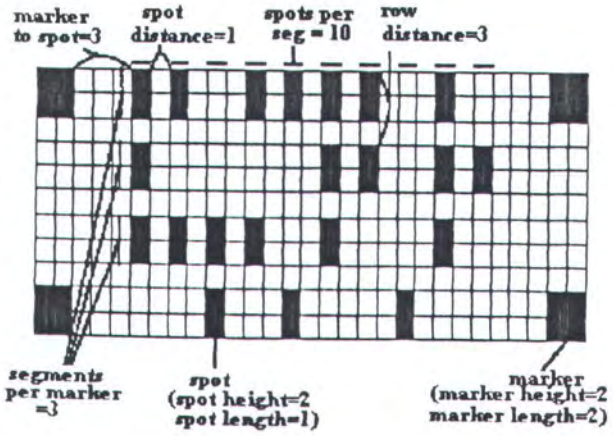
- **Skip Marker**

  Variabel ini digunakan untuk menentukan marker-marker vertikal yang perlu untuk tidak dicetak (dihilangkan).

- **Spot Depopulation**

  Variabel ini digunakan untuk menentukan jumlah pixel yang tidak dicetak dalam satu spot (dot).

  Untuk lebih jelas, dapat dilihat pada gambar 3.1. dan 3.2.



Gambar 3.2.
Format Data Sector DataTile

- **Data Sector**

  Data Sector berisi data dari file yang akan disimpan dalam DataTile. Ukuran spot yang tersimpan dalam Data Sector dibuat tidak tetap (*variabel*). Dalam Data Sector ini semua data file disimpan. Sedang ukurannya mengacu pada data yang

disimpan pada Meta Sector.

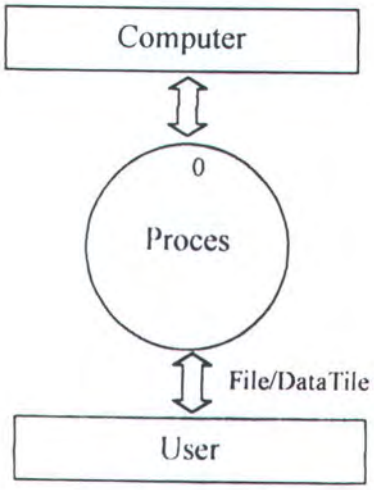
3.1.2. Perancangan Data

Sistem operasi menyimpan data pada komputer dalam bentuk file-file. File-file inilah selanjutnya yang akan diubah formatnya ke dalam bentuk DataTile. Agar memudahkan proses pengubahan tersebut maka tipe data yang digunakan adalah *byte* karena file disimpan dalam bentuk byte.

Sedang untuk proses pengembalian dari DataTile ke dalam file kembali, type data yang digunakan adalah *array* dengan panjang 65355 (0xFFFF).

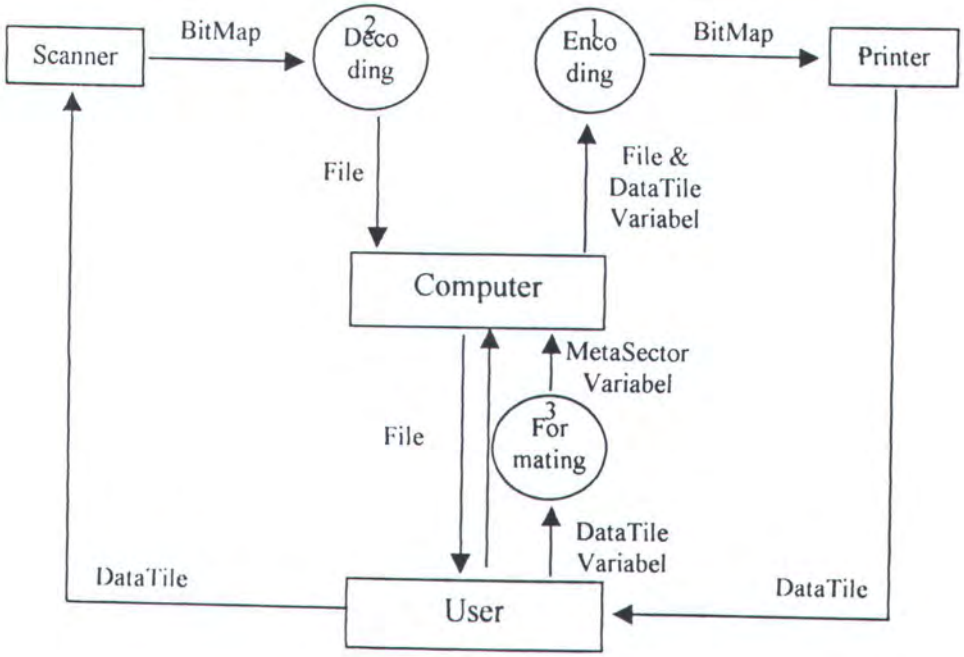
3.1.3. Diagram Aliran Data (DAD)

Proses aliran data pada perangkat lunak dapat digambarkan dengan gambar 3.3.



Gambar 3.3.
Diagram Aliran Data Level 0

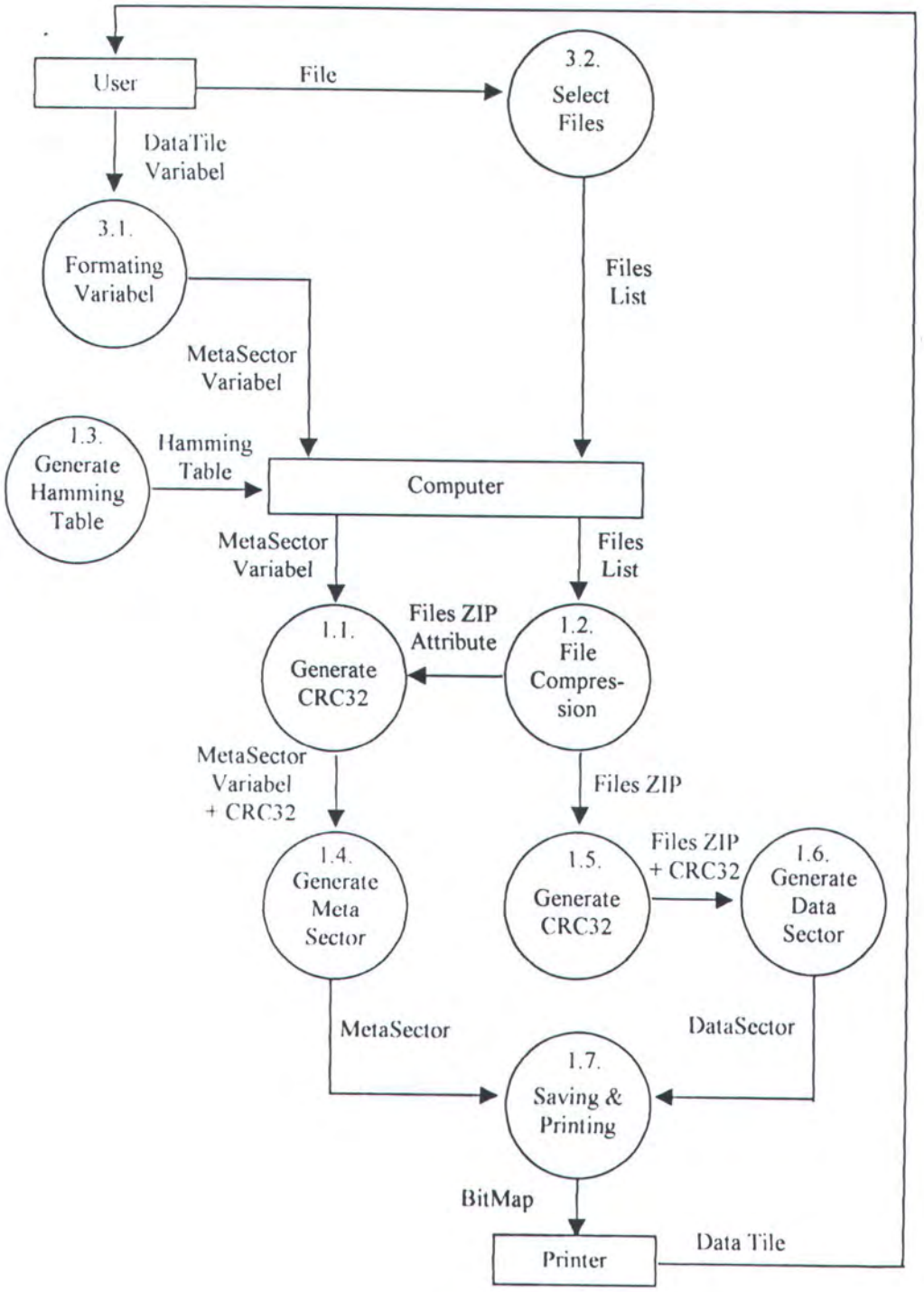
Berdasarkan DAD Level 0 tersebut dibuat DAD Level 1 seperti pada gambar 3.4.



Gambar 3.4.
Diagram Aliran Data Level 1

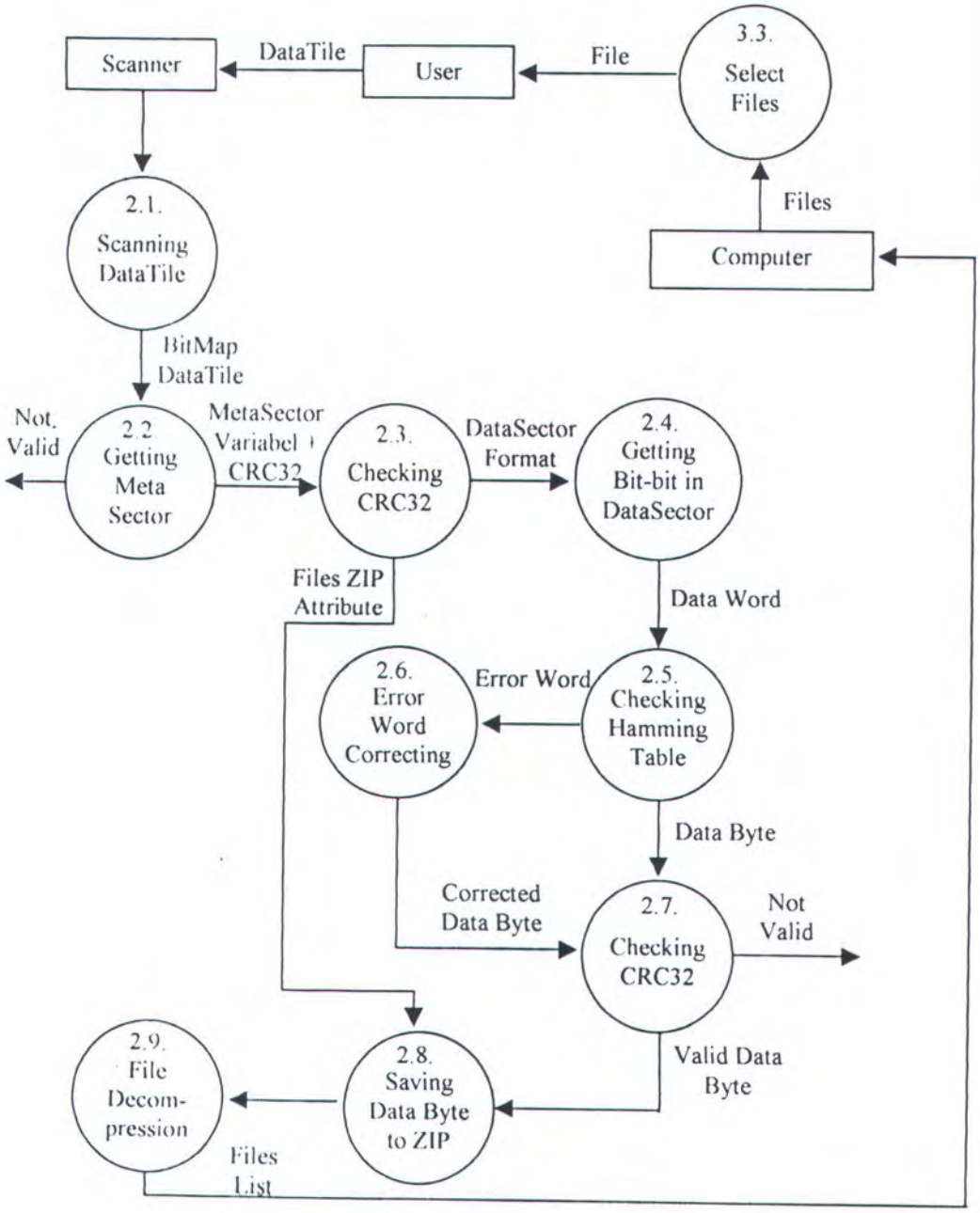
Dari Diagram Aliran Data Level 1 dipecah menjadi dua buah proses, yaitu proses 1 dan 2. Proses pertama *encoding* (pengkodean), yaitu proses mengubah file-file yang terdapat pada komputer menjadi bentuk data BitMap yang selanjutnya dicetak ke kertas yang disebut DataTile. Sedang proses yang kedua adalah kebalikan dari proses pertama.

Diagram Aliran Data untuk proses pertama digambarkan dengan gambar 3.5.



Gambar 3.5.
Diagram Aliran Data Level 1.1.

Sedang untuk proses kedua, yang merupakan kebalikan proses pertama digambarkan dengan gambar 3.6.



Gambar 3.6.
Diagram Aliran Data Level 2.1.

3.1.4. Disain Antarmuka

Berdasarkan Diagram Aliran Data yang telah dibuat, maka perangkat lunak mempunyai disain antarmuka yang digambarkan sebagai struktur menu berikut :

Tabel 3.1.
Disain Layout Menu Perangkat Lunak

GroupMenu	Menu Item	Proses
File	Select Source	Inisialisasi Scanner
	Print Stored DataTile ...	Mencetak file DataTile yang tersimpan
	Printer Setup ...	Setup printer
	Exit	Keluar dari program
Create	Create DataTile ...	Membuat DataTile baru
	Data Format ...	Mendefinisikan variabel-variabel untuk DataTile
Extract	Scan, Decode and Run	Ambil gambar dari scanner, decoding DataTile, jalankan program
	Open, Decode and Run	Buka file DataTile, decoding DataTile, jalankan program
	Scan, Restore Data to Disk	Ambil gambar dari scanner, decoding DataTile, simpan dalam disk.
	Open, Restore Data to Disk	Buka file DataTile, decoding DataTile, simpan dalam disk
	Extract Zip Files	Dekompresi file ZIP
	Scan to Disk	Pilihan untuk menyimpan DataTile hasil scanning ke disk
	Scan to Memory	Pilihan untuk tidak menyimpan DataTile hasil scanning
Help	About	Menampilkan versi program

3.2. Pembuatan Perangkat Lunak

Berdasarkan rancangan-rancangan yang telah dibuat, langkah selanjutnya

adalah mengimplementasikan hasil disain tersebut ke dalam perangkat lunak. Implementasi disain tersebut menggunakan Borland Delphi versi 3.0.

3.2.1. Pembuatan Struktur Data

Berdasarkan rancangan data yang telah dibuat, maka struktur data untuk DataTile adalah TbitMap. Sedang untuk variabel-variabel pada DataTile yang disimpan dalam MetaSector dideklarasikan:

```
spotheight, spotlength, spotdepop: byte;
spotdist, rowdist, skipmark: byte;
marklength, markheight: byte;
marktospot, segpermark: byte;
spotperseg, datasegs: byte;
sectoraccros, sectordown: byte;
maxwidth: integer;
targetprinter: byte;
hscanner: boolean;
fscanner: boolean;
redudancy: byte;
leftmargin, topmargin: integer;
dpiprinter: integer;
tshift: byte;
markmetalength: integer;
markmetaheight: integer;
```

Tidak semua variabel yang telah dideklarasikan di atas disimpan dalam MetaSector. Yang disimpan dalam MetaSector adalah indeks DataTile, jumlah DataTile, ukuran file yang disimpan dalam DataTile, nama file ZIP, sebagian variabel-variabel DataTile, ditambah dengan CRC32. Jumlah seluruh data tersebut sebanyak 40 byte.

Urutan 40 byte data pada MetaSector adalah:

- 1 byte indeks
- 1 byte jumlah DataTile
- 2 byte jumlah yang disimpan pada DataTile
- 12 byte nama file ZIP
- 4 byte CRC32
- 16 byte variabel untuk MetaSector yang terdiri atas `spotheight`, `spotlength`, `spotdepop`, `spotdist`, `rowdist`, `skipmark`, `marklength`, `markheight`, `marktospot`, `segpermark`, `spotperseg`, `datasegs`, `sectoraccros`, `sectordown`, `targetprinter`, dan `redudancy`.
- 4 byte CRC32.

Sedang untuk proses pembacaan file ZIP digunakan pembacaan per byte, sedang untuk proses decoding proses penulisannya dengan menggunakan array.

3.2.2. Pembuatan Directory dan File

Beberapa sub *directory* (*folder*) yang perlu dibuat untuk menjalankan perangkat lunak adalah:

- Directory **ZIPEncod**

  Directory ini digunakan untuk menyimpan file-file ZIP yang digunakan untuk membuat beberapa DataTile. Nama file ZIP sama dengan nama pertama dari file yang dimasukkan ke dalam file ZIP. Misalnya nama file pertama yang dipilih

adalah MYCOVER.DOC maka nama file ZIP adalah MYCOVER.ZIP.

- Directory **GenBMPS**

  Directory ini digunakan untuk menyimpan file-file DataTile yang telah dibuat dalam format BitMap. Nama file BitMap diambil dari nama file ZIP sebanyak enam karakter, kemudian ditambah dengan dua digit bilangan sesuai dengan seri DataTile yang telah dibuat. Misalnya nama file ZIP adalah MYCOVER.ZIP maka seri nama file BMP-nya adalah MYCOVE01.BMP, MYCOVE02.BMP dan seterusnya.

- Directory **GetBMPS**

  Directory ini digunakan untuk menyimpan file-file DataTile hasil proses scanning.

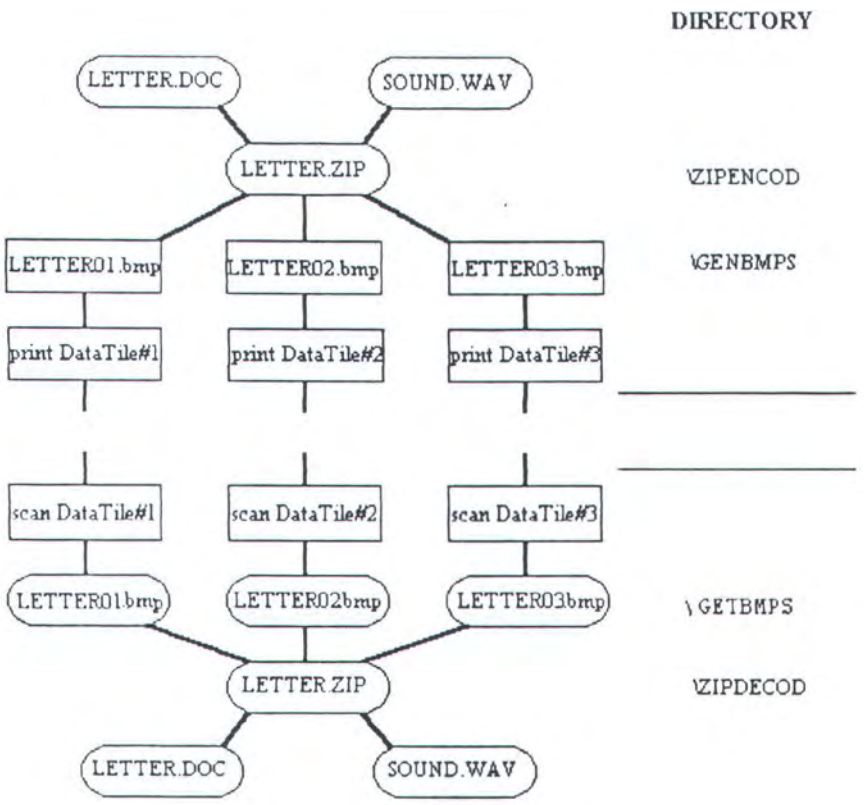
- Directory **ZIPDecod**

  Directory ini digunakan untuk menyimpan hasil akhir dari proses decoding yaitu file-file *original* (asli) yang telah didekompresi dari data yang terrepresentasi dalam DataTile.

Pada proses kompresi file, nama file ZIP adalah nama pertama dari file yang dipilih. Panjang nama file ZIP dibuat standar 8 karakter, sebagai contoh untuk file dengan nama **BukuTugas1.doc** akan dibuat file ZIP-nya dengan nama **BukuTuga.ZIP**. Dari file ZIP tersebut kemudian dibuat file DataTile dengan format BitMap, dengan nama **BukuTu01.BMP**. Apabila jumlah DataTile lebih dari satu, maka nama file DataTile diberi indeks, dengan urutan mulai dari **01**, sehingga file

DataTile akan menjadi **BukuTu01.BMP**, **BukuTu02.BMP**, **BukuTu03.BMP** dan seterusnya.

Proses pembuatan dan penyimpanan data yang diperlukan perangkat lunak beserta directory-directorynya dapat dilihat pada gambar 3.7.



Gambar 3.7.
Direktori Perangkat Lunak

3.2.3. Fungsi dan Prosedur Program

Beberapa prosedur program yang dibutuhkan adalah prosedur untuk menghasilkan tabel Hamming, prosedur pembangkit CRC-32, prosedur pembuatan

DataTile dan prosedur pembacaan DataTile.

3.2.3.1. Prosedur Tabel Hamming Code

Prosedur ini digunakan untuk membuat tabel Hamming dari karakter yang dibaca dari file ZIP yang akan dibuat DataTilenya. Implementasi prosedurnya adalah sebagai berikut:

```
function GetHammingCode(myByte: Byte): Word;
var
  c1, c2, c4, c8: byte;
begin
  c1:= GetBit(myByte, 0) xor GetBit(myByte, 1) xor
       GetBit(myByte, 3) xor GetBit(myByte, 4) xor
       GetBit(myByte, 6);
  c2:= GetBit(myByte, 0) xor GetBit(myByte, 2) xor
       GetBit(myByte, 3) xor GetBit(myByte, 5) xor
       GetBit(myByte, 6);
  c4:= GetBit(myByte, 1) xor GetBit(myByte, 2) xor
       GetBit(myByte, 3) xor GetBit(myByte, 7);
  c8:= GetBit(myByte, 4) xor GetBit(myByte, 5) xor
       GetBit(myByte, 6) xor GetBit(myByte, 7);
  GetHammingCode:=
       GetBit(myByte, 7) shl 11 + GetBit(myByte, 6) shl 10 +
       GetBit(myByte, 5) shl 9 + GetBit(myByte, 4) shl 8 +
       c8 shl 7 + GetBit(myByte, 3) shl 6 +
       GetBit(myByte, 2) shl 5 + GetBit(myByte, 1) shl 4 +
       c4 shl 3 + GetBit(myByte, 0) shl 2 + c2 shl 1 + c1;
end;
```

Prosedur ini digunakan untuk menghasilkan kode Hamming untuk tiap-tiap karakter. Kode ini berfungsi sebagai *Error Correcting Code* (ECC).

3.2.2.2. Prosedur Pembangkit CRC

Prosedur ini digunakan untuk mendeteksi ada tidaknya kesalahan yang terjadi

pada saat proses decoding data. Bila nilai akhir dari CRC adalah nol, berarti tidak ada kesalahan pada data yang didecoding.

```
procedure TForm1.GetCRC(var CRC: Integer; Input: Byte);
begin
  if (CRC shr 31) > 0 then
    CRC := ((CRC shl 1) xor CRC32) xor Input
  else
    CRC := (CRC shl 1) xor Input;
end;
```

3.2.3.3. Prosedur Pembuatan DataTile

Pembuatan DataTile dilakukan setelah file ZIP terbentuk. DataTile tersebut dibuat berdasarkan variabel-variabel yang dimasukkan dalam Form `Customized Data Format`. Langkah-langkah pembuatan DataTile adalah:

1. Simpan semua variabel yang diperlukan Meta Sector, yaitu sebanyak 40 byte seperti yang telah didefinisikan dalam Pembuatan Struktur Data. Buat LandMark, MarkMetaSector, dan DataMetaSector dengan menerjemahkan 40 byte data MetaSector dalam bentuk spot, hitam untuk bit 1 dan putih untuk bit 0 dengan ukuran standar.
2. Tiap byte data file ZIP, dibuat kode Hamming
3. Ambil per bit data kode Hammingnya, terjemahkan dalam bentuk spot, hitam untuk bit 1 dan putih untuk bit 0.
4. Ulangi langkah 3, sampai jumlah spot yang dibuat sama dengan jumlah spot per segment. Bila telah sama beri jarak lokasi untuk Marker DataSector dengan data segmen berikut.

5. Ulangi langkah 4, sampai jumlah segmen yang terbentuk sama dengan jumlah Data Segment
6. Beri Marker DataSector sesuai dengan jumlah Data Segment ditambah 1 (satu) untuk beberapa kelompok baris data.
7. Ulangi langkah 3 sampai jumlah data yang tersimpan dalam DataTile sama dengan kapasitas satu DataTile atau data sudah habis.
8. Simpan DataTile dengan disertai indeks.
9. Bila masih terdapat data yang belum tersimpan dalam DataTile, buat DataTile lanjutannya, ulangi langkah pertama

3.2.3.4. Prosedur *Decoding* DataTile

Prosedur ini digunakan untuk membaca ulang DataTile yang telah dicetak, dan dimasukkan melalui scanner. DataTile hasil scanning tersebut harus disimpan dalam format BMP. Langkah-langkah untuk mengenali DataTile hasil scanning adalah sebagai berikut :

- *GetLandMark*, prosedur ini digunakan untuk mencari titik awal dari DataTile (*starting point*), dan dihitung tingkat kemiringannya.
- *GetMetaFile*, prosedur digunakan untuk membaca data pada Meta Sector, didasarkan pada nilai (x, y) yang ditemukan pada saat *GetLandmark*.
- Hasil pembacaan pada Meta Sector tersebut digunakan untuk membaca data hasil scanning. Data dibaca dengan mencari posisi marker yang terdapat pada DataTile

dengan perkiraan posisi didasarkan pada perhitungan variabel pada Meta Sector.

- Data dibaca tiap spot, didekodekan ke dalam `Word`, dicek kode Hammingnya, bila terjadi kesalahan (error), kirim pesan, bila dapat dikoreksi dilanjutkan untuk data berikutnya, setelah selesai satu *frame* dicek nilai CRC data tersebut. Langkah ini diulang-ulang sampai data terakhir pada DataTile.
- Jika tidak terdapat kesalahan dan DataTile lebih dari satu, maka ulangi langkah pertama, jika hanya satu atau DataTile telah dimasukkan semua, simpan file ZIP asli ke dalam directory ZipDecod.

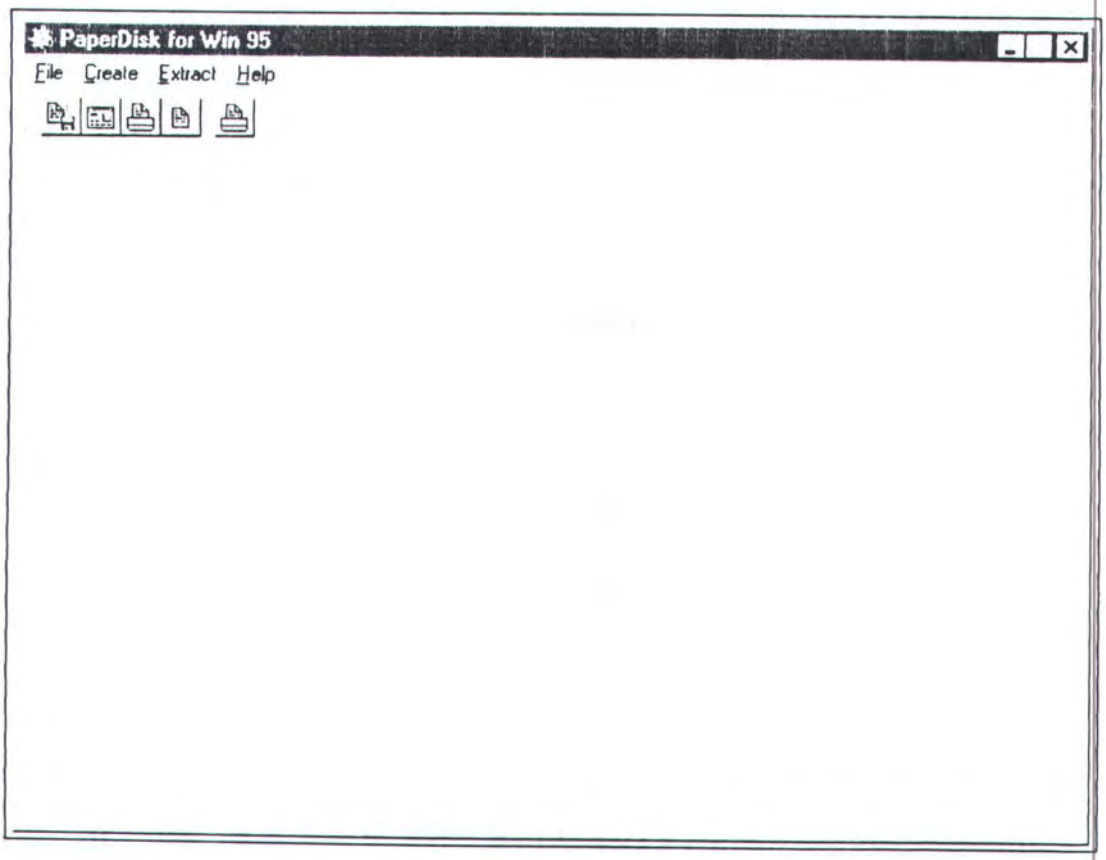
Jika data hasil decoding tersebut selesai disimpan, data tersebut dapat didekompresi dengan perangkat lunak atau dijalankan langsung.

3.2.4. Disain Antarmuka

Rancangan disain menu yang dibuat diimplementasikan dengan menggunakan struktur menu *pulldown*. Selain itu juga dilengkapi dengan beberapa *form* sebagai antarmuka, seperti form pengisian variabel untuk DataTile, form pembuatan DataTile dan form untuk dekompresi file ZIP.

3.2.4.1. Disain Menu Utama

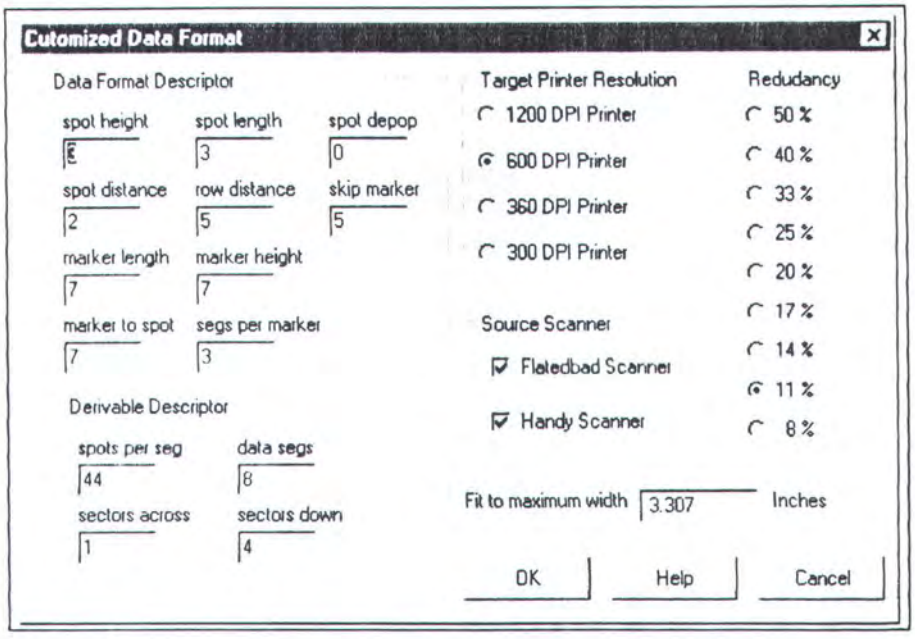
Disain dibuat dengan model *pulldown menu*. Dengan menu utama seperti pada gambar 3.8. Untuk form utama diberi judul **PaperDisk for Win 95**.



Gambar 3.8.
Form dan Menu Utama Perangkat Lunak

3.2.4.2. Form untuk Pengisian Format DataTile

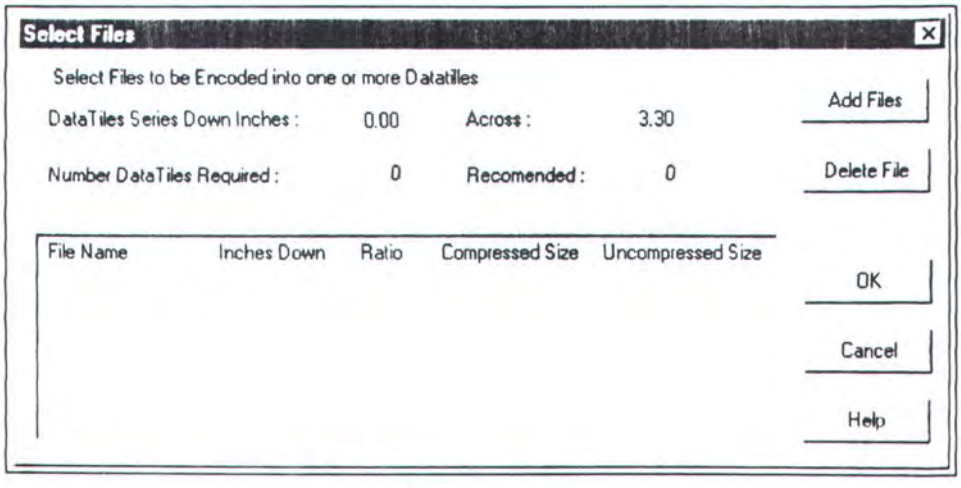
Form untuk pengisian variabel-variabel yang diperlukan dalam proses pembuatan DataTile dapat dilihat pada gambar 3.9. Variabel-variabel tersebut dapat diubah-ubah disesuaikan dengan kebutuhan. Semakin besar data-data yang dimasukkan, berarti semakin berar pula ukuran DataTile yang dihasilkan.



Gambar 3.9.
Form Pengisian Variabel untuk Pembuatan DataTile

3.2.4.3. Form untuk Pembuatan DataTile

Form ini digunakan untuk memilih file yang akan dibuat DataTilenya, dengan terlebih dahulu dikompresi. Bentuk form seperti pada gambar 3.10.



Gambar 3.10.
Disain Form Pembuatan DataTile

BAB IV
HASIL UJI COBA DAN EVALUASI PERANGKAT LUNAK

BAB IV

HASIL UJI COBA DAN EVALUASI PERANGKAT LUNAK

Perangkat lunak PaperDisk yang telah dibuat, selanjutnya diuji coba. Uji coba dilaksanakan untuk menganalisis proses-proses atau prosedur yang terdapat pada perangkat lunak. Analisis yang dilakukan meliputi analisis proses kompresi file, penerjemahan data hasil kompresi ke dalam format DataTile, proses mencetak DataTile ke printer, proses pengambilan data dari scanner, proses *decoding* DataTile dan dekompresi file menjadi file asli (*original*).

4.1. Analisis Proses Kompresi File

Proses kompresi file dilakukan dengan tujuan agar file-file yang akan diterjemahkan ke dalam format DataTile mempunyai ukuran yang kecil. Dengan demikian maka jumlah data yang tersimpan dalam kertas menjadi lebih banyak.

PaperDisk secara otomatis melakukan kompresi file pada saat pengguna membuat DataTile (proses *Create DataTile*). Proses kompresi file dimulai dengan memasukkan nama-nama file-file yang akan dijadikan DataTile. Hasil proses kompresi file ditampilkan dalam tabel daftar file (*file list*) yang berisi nama file, ukuran panjang DataTile yang dihasilkan, rasio kompresi, ukuran file terkompresi dan ukuran sesungguhnya. Hasil file kompresi (ZIP file) disimpan di dalam *directory* **ZipDecod**. Jumlah DataTile yang diperlukan ditunjukkan pula pada saat proses

kompresi file.

Sebagai uji coba dilakukan pembuatan DataTile dari beberapa file seperti yang terdapat dalam tabel 4.1.

Tabel 4.1.
Hasil Kompresi File, Ukuran dan Jumlah DataTile yang Dihasilkan

Nama File	Ukuran Asli (Byte)	Ukuran Kompresi (Byte)	Ukuran DataTile	Jumlah DataTile (Halaman)
KPBab1.rtf	10.253	3.152	3,3" x 0,92"	1
KPBab2.rtf	12.966	3.430	3,3" x 1,0"	1
KPBab3.rtf	25.414	7.262	3,3" x 2,1"	1
KPBab4.rtf	45.796	6.389	3,3" x 1,8"	1
BukuTA2.rtf	599.065	61.214	3,3" x 18"	5
Skripsi.chi	169.238	42.384	3,3" x 12"	4
Calc.exe	59.392	33.770	3,3" x 9,7"	3

Setelah masing-masing file dikompresi, maka proses selanjutnya adalah pembuatan file DataTile berdasarkan file kompresi ZIP yang telah dibuat.

4.2. Analisis Proses Pembuatan DataTile

Proses pembuatan DataTile dilakukan setelah pengguna menyetujui pembuatan DataTile dengan menekan *button* **OK**. Apabila pengguna menekan button **Cancel**, maka file hasil kompresi tersebut dihapus.

Proses pembuatan DataTile dilakukan dengan membaca byte-byte dari file ZIP, selanjutnya byte-byte data tersebut diubah ke dalam kode Hamming dan dihitung nilai CRC dengan. Hasil perhitungan CRC dan kode Hamming yang telah

dihasilkan diterjemahkan menjadi spot-spot (titik-titik) pada data BitMap. Dari masing-masing file ZIP yang dibuat DataTile yang disimpan pada folder **GenBMPS**. Jumlah DataTile yang dihasilkan sesuai dengan tabel 4.1.

Selain diujicobakan dengan satu file, proses uji coba juga dilakukan terhadap beberapa file sekaligus seperti contoh di atas. Pada file yang besar atau perangkat lunak secara otomatis membuat DataTile lebih dari satu (*Muliple DataTile*). Indeks diberikan secara otomatis pada nama file DataTile dengan kode 01 - 99 dan informasi tersebut dimasukkan pula ke dalam Meta Sector DataTile.

4.3. Analisis Proses Pencetakan DataTile

Setelah sebuah file DataTile dibuat, proses selanjutnya adalah mencetak DataTile tersebut. Proses pencetakan diperlukan agar DataTile tersebut secara permanen disimpan dalam kertas. Pencetakan dilakukan dengan printer HP LaserJet 6L dengan resolusi 600 DPI. Uji coba dilakukan dengan mencetak beberapa file DataTile yang telah dibuat sebelumnya. Proses mencetaknya dengan printer HP LaserJet dan menunjukkan hasil sesuai dengan yang diharapkan.

4.4. Analisis Proses Pengambilan DataTile dari Scanner

Proses pengambilan DataTile dari scanner sangat diperlukan untuk proses decoding. Proses pengambilan data dari scanner ini dengan menggunakan protokol TWAIN, melalui fasilitas yang terdapat pada komponen TEnhTwain. Proses

pengambilan DataTile selain menggunakan fasilitas langsung dari perangkat lunak, dapat pula dilakukan melalui program yang disediakan oleh vendor scanner yang terpasang.

Proses pengambilan gambar DataTile dilakukan menggunakan HP ScanJet 5P. Data gambar hasil scanning disimpan dalam format BMP. Proses pengambilan gambar bila melalui perangkat lunak, dilakukan dengan dua pilihan, yaitu:

- Scan ke memory komputer, gambar hasil scan tidak disimpan ke disk, tetapi langsung di-decoding oleh perangkat lunak.
- Scan ke disk, gambar hasil scan langsung disimpan ke disk, bila diperlukan dapat dibuka oleh perangkat lunak untuk proses decoding.

4.5. Analisis Proses Decoding DataTile

Pada saat proses decoding DataTile, ada dua kemungkinan yang dapat terjadi atas DataTile hasil scanning. Kemungkinan tersebut adalah :

1. DataTile hasil scanning sama persis dengan DataTile asli
2. DataTile hasil scanning mengalami distorsi (penyimpangan)

Kemungkinan pertama sangat sulit terjadi, dan yang terjadi pada uji coba adalah kemungkinan kedua. Penyimpangan yang terjadi antara lain:

1. Ukuran spot pada DataTile berubah
2. DataTile hasil scanning tidak tegak lurus (miring)
3. Terdapat noise, akibat kualitas tinta yang turun atau terdapat debu pada kaca

Scanner.

4. Terdapat spot yang hilang atau bertambah akibat salah penyimpanan (terkena tinta ballpoint, spidol dan lain-lain)
5. Jarak antar spot berubah

Berdasarkan kemungkinan yang terjadi tersebut, langkah pertama adalah melakukan uji coba dengan menggunakan DataTile ideal, kemudian dilakukan perubahan-perubahan terhadap DataTile ideal tersebut. Perubahan-perubahan dilakukan dengan mengubah kemiringannya, mengubah ukuran spot, menambah noise, menambah dan mengurangi spot, dan menggeser jarak antar spot.

4.5.1. Uji Coba DataTile Ideal

Proses decoding DataTile pertama diujicobakan untuk DataTile ideal, yaitu DataTile yang tidak mengalami perubahan sama sekali. Perubahan yang dimaksud adalah tingkat kemiringan DataTile, ukuran spot dan perubahan lebar dan tinggi DataTile. Untuk mendapatkan data ideal, dilakukan dengan membuka ulang DataTile yang telah dibuat perangkat lunak. Proses ini dilakukan berulang-ulang dengan menggunakan DataTile yang berbeda-beda, yaitu menggunakan DataTile untuk file-file seperti yang terdapat dalam tabel 4.1. Dari beberapa kali proses uji coba decoding untuk DataTile ideal, perangkat lunak berhasil melakukannya dengan baik. Dengan hasil tersebut, berarti prosedur untuk proses decoding DataTile telah berjalan sesuai dengan yang diharapkan.

4.5.2. Uji Coba dengan Mengubah Ukuran Spot

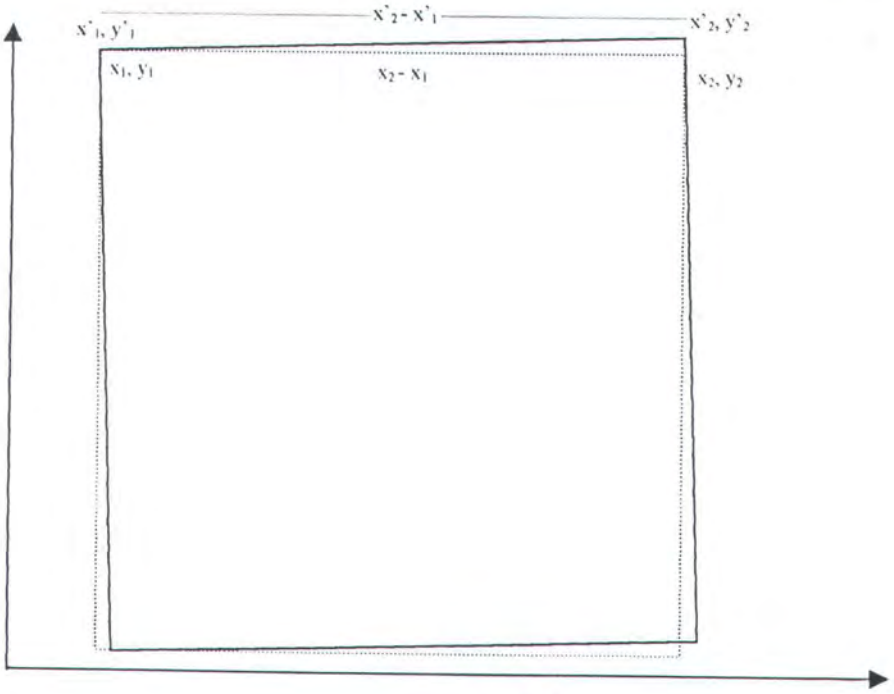
DataTile yang telah berhasil didecoding tersebut, kemudian diubah ukuran spotnya. Pengubahan tersebut dilakukan dengan mengedit DataTile tersebut melalui *Image Editor* (program pengedit gambar) seperti *Microsoft Paint*, *Adobe Photoshop* atau *Corel PhotoPaint*. Perubahan ukuran spot tersebut dilakukan dengan menambah atau mengurangi jumlah pixel dalam spot. Beberapa DataTile yang telah diubah tersebut kemudian didecoding.

Uji coba dilakukan dengan mengedit lima buah DataTile, yaitu **KPBab301.BMP**, **KPBab401.BMP**, **Calc0001.BMP**, **Calc0002.BMP**, dan **Calc0003.BMP**. Masing-masing DataTile yang diubah dengan menggunakan **Microsoft Paint** dengan menambah atau mengurangi ukuran spot. Setelah proses decoding dilakukan untuk lima DataTile tersebut. Dari hasil uji coba, perangkat lunak dapat melakukan pengenalan DataTile dengan baik, dan apabila terdapat kesalahan, secara otomatis membetulkannya.

4.5.3. Uji Coba dengan Memutar DataTile Ideal

DataTile hasil scanning seringkali miring (tidak tegak lurus), walaupun sudah dilakukan usaha-usaha untuk membuat data hasil scanning tersebut tegak lurus. Untuk itu perangkat lunak harus mempunyai kemampuan mendeteksi kemiringan DataTile. Kemampuan tersebut diperoleh dengan menggunakan perhitungan matematis. Kemiringan yang ditoleransi adalah kemiringan dengan sudut kecil, karena untuk sudut yang kecil perubahan jarak terhadap sumbu x menjadi sangat

kecil, sehingga dapat diabaikan. Pendekatan ini digunakan untuk penyederhanaan perhitungan dan proses. Untuk lebih jelasnya dapat dilihat pada gambar 4.1.



Gambar 4.1.
Diagram DataTile yang Diputar dengan Sudut Kecil

Berdasarkan gambar 4.1., nilai $\mathbf{x'_2 - x'_1}$ ($\Delta \mathbf{x'}$) dibandingkan dengan $\mathbf{x_2 - x_1}$ ($\Delta \mathbf{x}$) mempunyai selisih yang sangat kecil, sehingga nilai $\Delta \mathbf{x'}$ dapat digantikan dengan nilai $\Delta \mathbf{x}$.

Dari hal tersebut di atas, dilakukan uji coba dengan DataTile yang miring. Uji coba dilakukan dengan memutar DataTile ideal beberapa derajat searah jarum jam atau berlawanan jarum jam. Pemutaran DataTile tersebut dilakukan dengan *Image Editor* (**Adobe Photoshop**). Uji coba dilakukan dengan memutar DataTile **KPBab301.BMP** dari 0° sampai 2,0° searah jarum jam dan berlawanan jarum jam

dan dilakukan secara bertingkat dengan selisih 0,1°. Dari hasil uji coba, perangkat lunak hanya dapat mengenali dengan baik DataTile yang mempunyai kemiringan di bawah-1,6°.

4.5.4. Uji Coba dengan Memberi *Noise* pada DataTile

Pada DataTile hasil scanning seringkali ditemukan noise berupa spot-spot kecil. Noise tersebut terjadi karena kualitas tinta dari printer yang digunakan rendah atau kertas yang digunakan tidak putih bersih (masih terdapat noda/bercak kotor).

Untuk itu dilakukan uji coba terhadap DataTile yang telah diberi noise. Proses penambahan noise dilakukan dengan Image Editor (**Adobe Photoshop**), secara bertingkat mulai dari 0% sampai dengan 7,5% --dalam **Adobe Photoshop**, besar *noise* tidak dinyatakan dalam persen (%), tetapi dinyatakan dengan angka 1 sampai dengan 999-- dengan peningkatan sebesar 0,5%. Uji coba dilakukan terhadap DataTile **KPBab301.BMP** dengan penambahan noise mulai dari 0,5% (skala 5) sampai dengan 7,5% (skala 75).

Berdasarkan hasil uji coba yang telah dilakukan, perangkat lunak hanya mampu mengenali DataTile yang mempunyai noise di bawah 5%. Semakin tinggi noise yang terdapat pada DataTile, semakin sering perangkat lunak menampilkan jendela pesan kesalahan. Pada tingkat noise di bawah 3% adalah kondisi yang optimal.

Kegagalan pendeteksian DataTile dengan noise lebih dari 4% disebabkan terlalu banyaknya spot-spot tambahan sehingga menyulitkan proses pendeteksian data

baik dalam Meta Sector maupun dalam Data Sector.

4.5.5. Uji Coba dengan Menambah dan/atau Mengurangi Spot

Uji coba ini dilakukan terutama untuk membuktikan prosedur Error Corecting Code. DataTile yang telah dibuat kemudian diedit dengan menggunakan Image Editor (**Microsoft Paint**). Beberapa spot yang terdapat pada DataTile dihilangkan dan di beberapa bagian ditambahkan. Hasil *editing* DataTile tersebut kemudian diujicobakan ke dalam program PaperDisk.

Apabila program menemukan kesalahan data, maka program akan menampilkan pesan berupa *Window Message* (jendela pesan). Pada Window Message tersebut ditampilkan data hasil koreksi dengan data hasil yang dibaca pada DataTile, dilengkapi dengan lokasi atau koordinat di mana terjadi kesalahan bit data. Informasi ini selanjutnya dapat digunakan sebagai acuan untuk mengedit DataTile yang tidak dapat dikoreksi dengan benar oleh program.

Uji coba dilakukan dengan mengubah jumlah spot yang terdapat pada DataTile **KPBab301.BMP**. Pengubahan tersebut dilakukan dengan menambah atau mengurangi satu, dua, tiga dan empat *spot* atau lebih, baik secara berurutan, bertetangga maupun dilakukan secara acak. Untuk kasus-kasus tertentu, perangkat lunak gagal mengenali DataTile. Kegagalan tersebut terutama untuk kasus lebih dari satu spot tersebut ditambahkan atau dikurangkan secara berurutan. Kegagalan tersebut karena prosedur yang digunakan untuk Error Correcting Code (kode pengoreksian kesalahan) adalah kode Hamming. Kode Hamming hanya mampu

mengoreksi kesalahan satu bit. Kecuali bila dua spot berurutan yang ditambahkan atau dihilangkan tersebut berbeda kelompok data (*frame*), di mana yang pertama adalah spot terakhir kelompok data pertama dan yang kedua adalah spot pertama dari kelompok data kedua pada DataTile.

4.5.6. Uji Coba dengan Mengubah Ukuran DataTile

Pada saat pencetakan DataTile, seringkali hasil cetak tersebut apabila discan ulang, menghasilkan DataTile yang mempunyai ukuran yang berbeda. Untuk itu program harus mampu mengantisipasi kejadian tersebut. DataTile yang telah dibuat kemudian diubah ukuran jarak antar spotnya dengan menggeser sebagian spot tersebut sehingga ukurannya berubah.

Uji coba dilakukan dengan mengubah ukuran lebar dan tinggi dari DataTile. Pengubahan tersebut dilakukan dengan program **Adobe Photoshop**. Uji coba dilakukan pada file DataTile **KPBab301.BMP**, yang mempunyai lebar **84 mm** dan **65 mm** pada resolusi 600 DPI. Penambahan ukuran DataTile dimulai dari 0,1 mm sampai dengan 2 mm, karena berdasarkan kenyataan saat DataTile yang sudah dicetak yang selanjutnya discan mengalami perubahan ukuran yang tidak lebih dari 1 mm. Perubahan ukuran DataTile apabila dihitung dalam satuan pixel dan diasumsikan berubah sebesar 0,5 mm maka besar perubahan jumah pixel adalah **0,5 x 600 / 25,3 = 11 pixel**.

Berdasarkan hasil uji coba seperti yang dilakukan, perangkat lunak hanya mampu mengenali dengan baik DataTile yang mengalami perubahan ukuran kurang

dari **2 mm**.

4.5.7. Uji Coba dengan DataTile Hasil Scanning

Setelah proses-proses uji coba di atas dilakukan, maka perangkat lunak diujicobakan dengan DataTile sesungguhnya, yaitu DataTile yang diperoleh dari proses scanning.

DataTile yang telah dibuat kemudian dicetak dengan HP Laser Jet 6L dengan resolusi 600 DPI. Dalam uji coba file yang digunakan adalah file-file DataTile yang telah dibuat sebelumnya. Dalam uji coba file yang dicetak adalah **KPBab301.BMP** dan **Calc0001.BMP**, **Calc0002**, dan **Calc0003.BMP**. Hasil cetak tersebut kemudian discan dengan menggunakan HP ScanJet 5P. Khusus **KPBab301.BMP** discan sebanyak lima kali. Untuk setiap kali scanning DataTile tersebut disimpan dengan nama **Scan1.BMP** sampai dengan **Scan5.BMP**. Sedang yang lainnya hanya sekali.

Hasil uji coba yang dilakukan beberapa kali, perangkat lunak telah berhasil melakukan proses decoding dengan baik, walaupun terkadang dalam proses tersebut ditemukan kesalahan dalam menerjemahkan data yang terdapat pada DataTile. Ini ditunjukkan dengan fasilitas jendela pesan (*Window Message*) kesalahan. Pada jendela kesalahan tersebut ditampilkan kesalahan bit, lokasi kesalahan dan data hasil koreksi. Apabila data hasil koreksi tersebut tidak sesuai dengan data asli, maka perangkat lunak secara otomatis menghentikan proses decoding.

Terhadap kesalahan yang sudah tidak dapat ditoleransi tersebut, pengguna dapat melakukan pengoreksian DataTile dengan menggunakan image editor semacam

Microsoft Paint, yang didasarkan pada data yang ditampilkan pada jendela kesalahan. Pengeditan dapat dilakukan untuk data-data yang salah, sesuai dengan lokasi yang ditunjukkan jendela pesan perangkat lunak. Setelah proses pengoreksian tersebut, proses decoding dapat dilakukan dengan baik.

Untuk kasus multiple DataTile (file-file dengan lebih dari satu DataTile), seperti pada DataTile **Calc0001.BMP** perangkat lunak secara otomatis meminta kepada pemakai untuk memasukkan file-file DataTile untuk proses *decoding* selanjutnya. Pengenalan multiple DataTile dapat dilakukan dengan baik.

4.6. Analisis Proses Dekompresi File

Proses dekompresi file hasil decoding dapat dilakukan secara langsung melalui perangkat lunak langsung. Proses dekompresi ditampilkan dengan antarmuka yang *user friendly* sehingga memudahkan pemakaian. Selain itu dapat pula proses dekompresi dilakukan dengan perangkat lunak lain seperti **WinZip** for Windows atau perangkat lunak sejenis.

4.7. Analisis Waktu yang Dibutuhkan

Program diuji coba untuk membuat DataTile dengan berbagai ukuran file data, kemudian dilakukan perhitungan waktu yang dibutuhkan untuk membuat file DataTile yang diperlukan. Uji coba dilakukan untuk file-file seperti yang terdapat pada tabel 4.1. Hasil uji coba seperti yang tercantum dalam tabel 4.2.

Tabel 4.2.
Waktu yang Dibutuhkan untuk Pembuatan DataTile

Nama File	Ukuran Kompresi (Byte)	Ukuran DataTile	Waktu Pembuatan DataTile
KPBab1.rtf	3.152	3,3" x 0,92"	7 detik
KPBab2.rtf	3.430	3,3" x 1,0"	7 detik
KPBab3.rtf	7.262	3,3" x 2,1"	10 detik
KPBab4.rtf	6.389	3,3" x 1,8"	8 detik
BukuTA2.rtf	61.214	3,3" x 18"	50 detik
Skripsi.chi	42.384	3,3" x 12"	40 detik
Calc.exe	33.770	3,3" x 9,7"	30 detik

Untuk hasil proses decoding, percobaan hanya dilakukan untuk satu DataTile, karena untuk kasus-kasus Multiple DataTile, waktu yang dibutuhkan tidak dapat dihitung, karena sangat tergantung dengan user. Demikian juga untuk waktu yang dibutuhkan proses mencetak tidak dilakukan uji coba karena faktor yang sama.

Untuk satu *DataTile* yang sempurna, yaitu ukuran standar diperlukan waktu selama 18 detik.

BAB V

KESIMPULAN DAN SARAN

BAB V

KESIMPULAN DAN SARAN

Dalam bab ini dijelaskan beberapa kesimpulan yang dapat diambil dari uji coba dan evaluasi terhadap perangkat lunak PaperDisk dan pengembangan lebih lanjut yang perlu dilakukan untuk penyempurnaan perangkat lunak.

5.1. Kesimpulan

Kesimpulan yang dapat diambil dari hasil uji coba dan evaluasi perangkat lunak PaperDisk adalah :

a. Dengan menggunakan perangkat lunak PaperDisk, kapasitas penyimpanan data dalam kertas dapat ditingkatkan dengan baik, sehingga kebutuhan kertas dapat dikurangi. Dengan menggunakan PaperDisk satu lembar kertas berukuran A4 mampu menyimpan data sebesar 60 KBytes file data tidak terkompresi.

b. Waktu yang diperlukan perangkat lunak untuk melakukan proses decoding sebuah file data tergantung pada ukuran file data dan proses encoding DataTile tergantung pada besarnya ukuran DataTile yang digunakan.

c. Berdasarkan hasil scanning yang dilakukan berulang-ulang, DataTile hasil scanning tersebut selalu mengalami perubahan ukuran, kemiringan dan tambahan spot sehingga teknik pendeteksian kesalahan (Error Detecting Code) dan Teknik Pengoreksi Kesalahan (Error Correcting Code) sangat diperlukan untuk untuk

proses decoding DataTile. Kode Hamming berfungsi untuk mengoreksi kesalahan yang terjadi dan CRC berfungsi untuk menguji data yang telah dikoreksi tersebut terdapat kesalahan lagi atau tidak.

5.2. Pengembangan Lebih Lanjut

Adapun pengembangan perangkat lunak lebih lanjut adalah :

a. Pada perangkat lunak ini, resolusi printer yang digunakan hanya untuk 600 DPI, sehingga perangkat lunak ini perlu dikembangkan lagi untuk jenis printer lain yang mempunyai resolusi berbeda. Penggunaan resolusi yang berbeda tersebut dilakukan dengan mengubah variabel-variabel untuk DataTile, menambahkan pilihan dalam program serta mengujicobakan pada jenis printer yang sesuai sehingga diperoleh variabel-variabel untuk DataTile yang tepat. Demikian juga pengaksesan data *scanner* perlu dikembangkan untuk resolusi yang lain (tidak hanya 600 DPI), seperti untuk mesin *Fax* yang mempunyai kerapatan 200 DPI, 300 DPI maupun 1200 DPI.
b. Perangkat lunak dapat dikembangkan untuk sistem keamanan data (*Data Security System*) dalam sistem pengiriman pesan, yaitu dengan mengubah model atau bentuk dari format DataTile, penambahan fasilitas password untuk membuka DataTile dan pengacakan data.

DAFTAR PUSTAKA

DAFTAR PUSTAKA

1. Anton, Howard, *Elementary Linear Algebra*, John Wiley & Sons Ltd., United Stated of America, 1996.

2. Clayton, Walnum, *Object-Oriented Programming with Borland C++ 4*, QUE Corp., First Edition, United Stated of America, 1994.

3. Levine, John, *Programming for Graphics Files in C and C++*, John Wiley & Sons Inc., United Stated of America, 1993.

4. Mikrodata, Elex Media Komputindo, Volume 9 Edisi 9, Jakarta, 1994.

5. Nashelsky, Louis, *Introduction to Digital Computer Technology*, John Wiley & Sons, Second Edition, United Stated of America, 1977.

6. Oliver, Dick, et.al., *Tricks of The Graphics Gurus*, Sams Publishing, First Edition, United State of America, 1993.

7. Pranata, Antony, *Permrograman Borland Delphi*, Andi, Edisi Pertama, Yogyakarta, Indonesia, 1997.

8. _____, Tip dan Trik *Permrograman Delphi*, Andi, Edisi Pertama, Yogyakarta, Indonesia, 1997.

9. Rimmer, Steve, *Bit-Mapped Graphics*, Windcrest - McGrawHill, First Edition, United Stated of America, 1990.

10. Stallings, William, *Computer Organization and Architecture, Designing for Performance*, Prentice-Hall Inc., Fourth Edition, New Jersey, United Stated of America, 1996.

11. _____ , *Data and Computer Comunications*, Macmillan Publishing Company, Third Edition, New York, United Stated of America, 1991.

12. Wozniewicz, Andrew, et,al., *Teach Yourself Borland Delphi in 21 Days*, SAMS Publishing, First Edition, Indiana, United Stated of America, 1995.

LAMPIRAN

LAMPIRAN A
CONTOH-CONTOH DATATILE

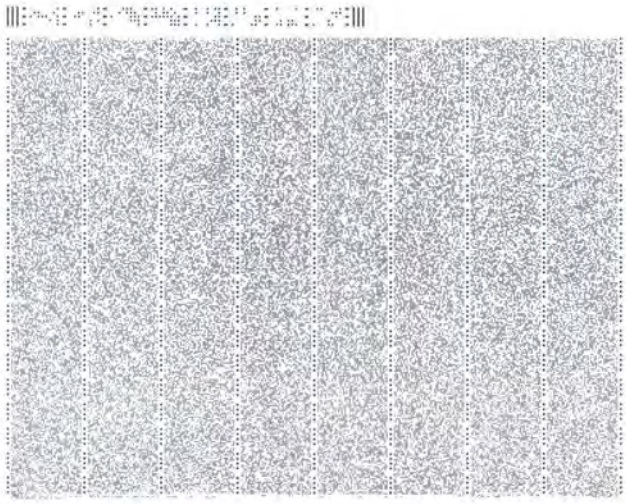
Prosedur decoding DataTile merupakan prosedur yang paling penting di antara prosedur-prosedur yang terdapat dalam program PaperDisk. Karena proses decoding merupakan penyimpanan ulang data yang telah disimpan dalam format DataTile di kertas menjadi file ke dalam komputer. Hal ini karena berdasarkan uji coba yang dilakukan berulang-ulang menunjukkan bahwa gambar atau image DataTile hasil scanning selalu mengalami perubahan dari bentuk idealnya (distorsi). Di antara distorsi yang terjadi adalah :

- DataTile tersebut miring
- Terdapat spot-spot kecil (noise) tambahan pada DataTile
- Ukuran spot mengalami perubahan dari yang sesungguhnya
- Perubahan jumlah spot
- Perubahan ukuran DataTile

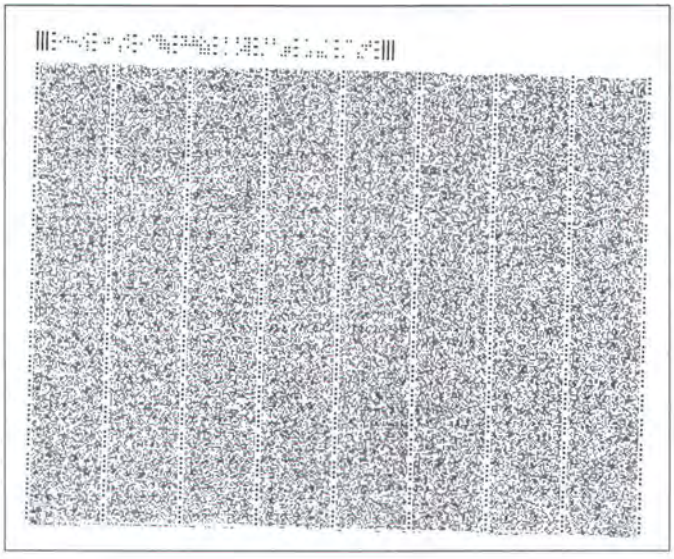
Dari hal tersebut di atas, maka perangkat lunak PaperDisk diujicobakan dengan data-data yang mendekati dengan kemungkinan di atas. Uji coba pertama dilakukan dengan DataTile ideal, yaitu DataTile hasil proses encoding program PaperDisk. Uji coba dilakukan berulang-ulang untuk beberapa DataTile.

Gambar A.1. adalah contoh DataTile ideal yang digunakan untuk proses uji coba. DataTile tersebut disimpan dengan nama **KPBab301.BMP**, mempunyai ukuran **380.744 KB**. Ukuran bila dicetak **81.26 mm x 64.1 mm**. DataTile tersebut digunakan untuk menyimpan file **KPBab3.rtf** yang mempunyai ukuran asli **25.414 Bytes**. File tersebut berisi naskah yang terdiri atas 10 halaman. Setelah dikompresi mempunyai ukuran **Bytes**.

Setelah proses decoding DataTile tersebut dapat dilakukan dengan benar, maka file **KPBab301.BMP** tersebut dibuka dengan program **Adobe Photoshop**. Melalui program **Adobe Photoshop** DataTile tersebut diputar searah dan berlawanan dengan jarum jam. Proses pemutaran ini dilakukan mulai dari $0{,}1^{o}$ sampai dengan $2{,}0^{o}$. Contoh DataTile yang diputar seperti dalam gambar A.2. Gambar A.2. adalah DataTile **KPBab301.BMP** yang diputar **$1{,}4^{o}$** searah jarum jam. Posisi miring DataTile tersebut dapat dibandingkan dengan bingkai luarnya. File tersebut diberi nama **KPBab301 Mir 1,4.BMP.**



Gambar A.1.
DataTile Ideal

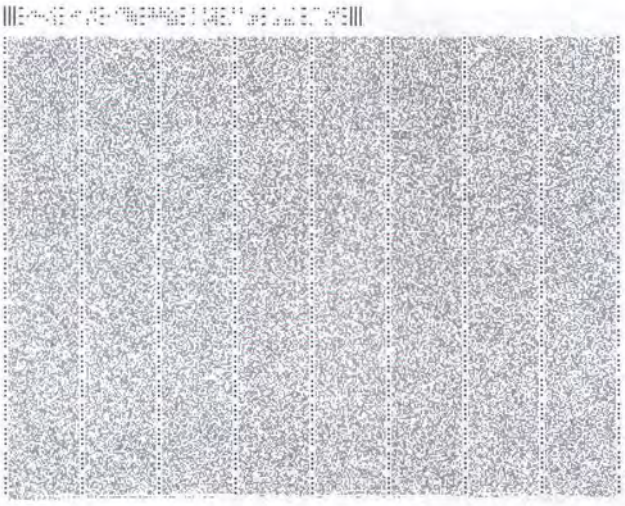


Gambar A.2.
DataTile yang Diputar 1,4° Searah Jarum Jam

Prosedur uji coba selanjutnya adalah dengan menambahkan noise pada DataTile. Penambahan noise dilakukan dengan menggunakan program **Adobe PhotoShop**. Caranya adalah dengan terlebih mengubah mode DataTile menjadi mode *Greyscale*,

diberi noise sebesar mulai 10 - 60 (pada **Adobe Photoshop** faktor noise dimulai dari 1 - 999).

Gambar A.3. adalah contoh DataTile yang diberi noise sebanyak 50 atau sekitar 5%. Kemudiam diubah ke mode BitMap dan disimpan dengan nama **KPBab301 Noise 5%.BMP**.

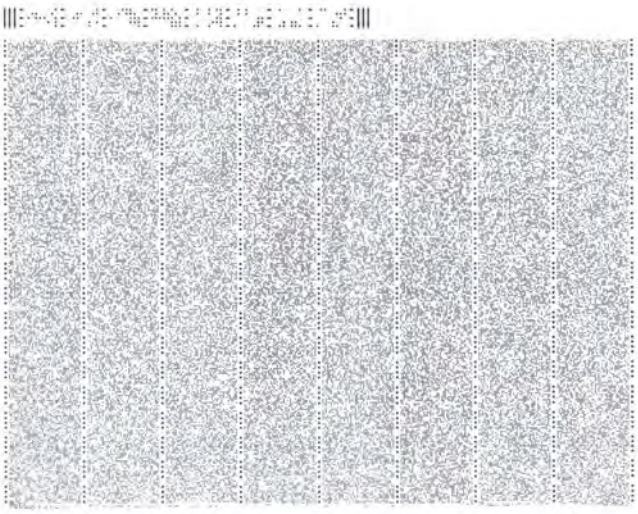


Gambar A.3.
DataTile yang Diberi Noise

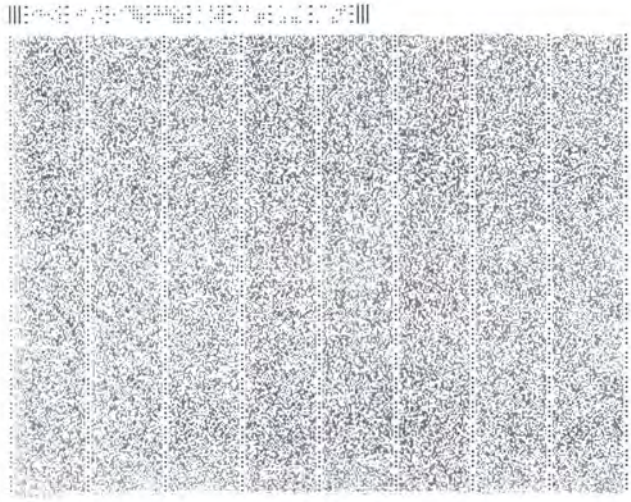
Uji coba selanjutnya adalah dengan mengubah ukuran dari DataTile. DataTile **KPBab301.BMP** yang mempunyai ukuran asli **81.26 mm x 64.1 mm** diubah secara bertahap mulai dari **81.36 mm x 64.2 mm** sampai **83,26 mm x 65,7 mm**.

Gambar A.4. adalah DataTile **KPBab301.BMP** yang diubah ukurannya dengan menambah sebesar 1,8 mm, lalu disimpan dengan nama **KPBab301 Medet 18.BMP**. Bila dibandingkan dengan ukuran asli yaitu **81.26 mm x 64.1 mm**, maka ukuran DataTile tersebut menjadi **83,03 mm x 65,5 mm**.

Proses terakhir adalah dengan uji coba dengan DataTile hasil scanning. Gambar A.5. adalah contoh DataTile hasil scanning dari HP ScanJet 5P dengan resolusi 600 DPI Hitam Putih (BW) dan disimpan dengan nama **Scan1.BMP**. Bila dibandingkan dengan ukuran asal yaitu **81,26 mm x 64.1 mm**, maka ukuran DataTile **Scan01.BMP** tersebut berubah menjadi 81,53mm x 64,47 mm.



Gambar A.4.
DataTile yang Ukurannya Diubah



Gambar A.5.
DataTile Hasil Scanning

LAMPIRAN B
PETUNJUK PENGGUNAAN PAPERDISK

B.1. Kebutuhan Sistem

Program PaperDisk membutuhkan Komputer dengan spesifikasi sebagai berikut :

- Prosesor 486 (disarankan Pentium)
- RAM 8 MB (disarankan 16 MB)
- 10 MB sisa ruang hard disk
- Windows 95 atau 97
- Printer Laser Jet 600 DPI
- Scanner 600 DPI untuk mode BW (Hitam Putih)

B.2. Instalasi Program PaperDisk

Disket instalasi terdiri atas file-file:

- Disk1.id
- Setup.ini
- Setup.pkg
- _setup.dll
- _isdel.exe
- Setup.exe
- Setup.ins
- _setup.lib
- inst32i.ex_
- _setup.1

Setelah proses instalasi, maka akan terbentuk program group **PaperDisk for Win 95.**

B.2. File-File yang Dibutuhkan

- PaperDisk.exe
- XcdUnz32.dll
- XcdZip32.dll
- Tsuz.dll

B.4. Menu Utama PaperDisk

Menu Utama program PaperDisk adalah :

- **File**

  Terdiri atas sub menu:

  - **Select Source**

    Digunakan untuk proses inisialisasi scanner yang terpasang pada komputer.

  - **Print Stored DataTile ...**

    Mencetak file DataTile yang telah dibuat oleh program PaperDisk. File-file DataTile disimpan dalam directory **\GenBMPS** pada directory **PaperDisk**.

  - **Printer Setup ...**

    Mengatur jenis printer yang digunakan untuk mencetak DataTile.

  - **Exit**

    Keluar dari program PaperDisk.

- **Create**

  Terdiri atas sub menu:

  - **Create DataTile ...**

    Membuat file DataTile baru dengan memasukkan nama-nama file yang akan disimpan dalam format DataTile, untuk selajutnya disimpan dan dicetak.

  - **Data Format ...**

    Digunakan untuk mengatur atau mendefinisikan variabel-variabel untuk DataTile, seperti lebar spot (*spot length*), tinggi spot (*spot height*), lebar marker (*marker length*), dan lain-lain. Untuk sementara variabel yang ada khusus untuk printer dan scanner 600 DPI. Angka-angka dalam form dapat diubah ukuran-ukurannya, tetapi sebaiknya tidak diubah.

- **Extract**

  Terdiri atas sub menu

  - **Scan, Decode and Run**

    Digunakan untuk proses pengambilan gambar DataTile dari scanner, hasilnya kemudian didecoding, setelah terbentuk file aslinya kemudian file tersebut dibuka atau dijalankan.

  - **Open, Decode and Run**

    File DataTile hasil scanning yang disimpan dalam pada directory GetBMPS dibuka, kemudian didecoding, setelah terbentuk file aslinya kemudian file tersebut dibuka atau dijalankan.

- **Scan, Restore Data to Disk**
  Digunakan untuk proses pengambilan gambar DataTile dari scanner, hasilnya kemudian didecoding, setelah terbentuk file aslinya kemudian file tersebut disimpan saja dalam disk atau hard disk.
- **Open, Restore Data to Disk**
  File DataTile hasil scanning yang disimpan dalam pada directory GetBMPS dibuka, kemudian didecoding, setelah terbentuk file aslinya kemudian file tersebut disimpan saja dalam disk atau hard disk.
- **Extract Zip Files**
  Form untuk proses dekompresi file ZIP. Fungsinya hampir sama dengan program **WinZip.**
- **Scan to Disk**
  Pilihan untuk menyimpan DataTile hasil scanning ke dalam disk.
- **Scan to Memory**
  Pilihan untuk tidak menyimpan DataTile hasil scanning.

- **Help**
  Terdiri atas sub menu **About**, yang berfungsi untuk menampilkan versi program PaperDisk.