PEMANFAATAN METODE ITERASI MATEMATIS UNTUK PENGUJIAN KINERJA PROCESSOR

Edhy Sutanta
Jurusan Teknik Informatika, Fakultas Teknologi Industri,
Institut Sains & Teknologi AKPRIND Yogyakarta

INTISARI

Teknologi *processor* sebagai piranti utama sistem komputer mengalami perkembangan yang begitu pesat. Masing-masing vendor selalu memberikan resume terbaik dari produk yang dipasarkannya. Hal ini seringkali justru menimbulkan kebingungan para konsumen ketika menentukan pilihannya. Konsumen umumnya ingin mengetahui bagaimana kinerja masing-masing *processor* tersebut agar mendapatkan *processor* terbaik yang sesuai kebutuhannya. Kinerja sebuah *processor* sebenarnya dapat diuji menggunakan pemrograman matematis konvensional dengan metode iterasi. Ada tiga macam teknik yang dapat digunakan untuk menguji kinerja *processor*, yaitu MIPS, SPEC dan MFLOPS.

Penelitian ini akan menguji kinerja *processor* menggunakan metode MFLOPS (*Mega Floating Point per Second*) yaitu berdasarkan kemampuan memproses perhitungan *integer* dan *real* (32 bit) yang dilakukan dalam waktu satu detik. Pengujian dilakukan menggunakan empat buah operator matematis, yaitu penjumlahan (+), pengurangan (-), perkalian (*) dan pembagian (/). Aplikasi dibangun dengan menggunakan *compiler Borland Delphi 5.0*. Aplikasi ini akan menghasilkan informasi penilaian yang menunjukkan kekuatan FPU (*Floating Point Unit*) *processor* sebagai *co-processor* yang terintegrasi sebagai sistem komputer. Aplikasi ini diperkuat dengan modul-modul deteksi *processor* melalui teknik pemrograman dalam kode *assembler* dan *Win32 API*. Hasil uji beberapa sistem komputer yang telah diuji sebelumnya juga disediakan sebagai informasi pembanding.

Kata kunci: Metode MFLOPS, iteration matematis, *processor*

PENDAHULUAN

CPU merupakan salah satu bagian terpenting dalam sistem komputer dan menarik untuk diuji. Secara teoritis, kecepatan *processor* telah menunjukkan sebuah penilaian bagi masing-masing *processor*. Semakin tinggi kecepatan (MHz) sebuah *processor* umumnya diyakini akan memiliki kinerja yang semakin baik. Karena itu, konsumen hanya akan berpatokan pada angka-angka yang ditampilkan oleh produsen dan jarang yang sempat menguji bagaimana kinerja *processor* yang sesungguhnya. Perkembangan teknologi *processor* sangatlah cepat dan vendor semakin banyak memproduksi dan memasarkan berbagai varian *processor* dengan kode sandi tersendiri. Bisa jadi, *processor* yang memiliki kecepatan tinggi pun tidak mampu menunjukkna kinerja yang baik sebagaimana yang informasi yang disertakannya. Hal ini disebabkan oleh perbedaan jenis instruksi yang terkandung dalam setiap *processor*, dan instruksi tersebut akan terus berkembang seiring perkembangan teknologi *processor* yang semakin kecil, murah, dan cepat. Untuk semuanya itu, maka para konsumen akan sangat memerlukan informasi obyektif tentang hasil pengujian *processor* yang mengambarkan kinerja sistem komputer dalam penggunaan keseharian.

Untuk mengetahui isi yang terdapat dalam sebuah PC, tidak cukup hanya dengan menilai komponen penyusunnya satu per satu. Produsen *hardware* cenderung menampilkan hasil pengujian terbaik yang menunjukkan keunggulan produk yang dihasilkannya. Sebagian bahkan sengaja melengkapi desain kemasannya dengan *benchmark* yang banyak dikenal, sehingga umumnya menampilkan informasi dan nilai-nilai yang prestisius. Namun demikian, dalam keseharian komponen PC dengan merk terkenal pun seringkali tidak memberikan kinerja sebagaimana informasi yang disampaikannya. Setiap sistem komputer mempunyai kinerja yang berbeda-beda, bahkan sekalipun disusun oleh komponen-komponen yang sama, bisa jadi akan memiliki kinerja yang bervariasi. Kombinasi dan *tuning* akhir yang tepat akan menghasilkan kinerja yang sebenarnya. Dalam aplikasi *real-time* pada *game* 3D yang berat, sebenarnya *processor* akan “kepanasan” karena harus melakukan perhitungan *floating point unit* untuk

menghasilkan keakuratan data 3 dimensi, meskipun angka *real* yang dihasilkan tidak terlalu besar. Di sinilah peran *benchmark* (hasil pengujian) murni yang dilakukan untuk menguji seberapa banyak operasi perhitungan matematis yang mampu dikerjakan oleh *processor*.

Berdasarkan latar belakang tersebut, maka penelitian ini akan mengembangkan sebuah aplikasi yang dapat digunakan untuk menilai kinerja komputer dengan melakukan pengujian pada *processor* melalui sistem *benchmark* yang memanfaatkan metode iterasi matematis dengan menggunakan *compiler* Borland Delphi 5.0. Aplikasi dibatasi untuk pengukuran kinerja *processor* (*benchmark*) dengan menggunakan metode MFLOPS, yang terdiri atas sembilan jenis prosedur pengujian matematis yang merupakan kombinasi operasi *integer* dan *real*.

TINJAUAN PUSTAKA

Komputer dapat digolongkan ke dalam beberapa jenis berdasarkan *processor* yang digunakannya. Komputer *IBM* menggunakan *processor* yang dibuat oleh perusahaan Intel dan beberapa perusahaan lain seperti *AMD* dan *Cyrix*, sedangkan komputer *Macintosh* menggunakan *processor* yang dibuat oleh perusahaan Motorola. *Processor* yang digunakan dalam sistem komputer akan menentukan jenis komputer tersebut. Jenis, kualitas, dan harga perangkat komputer secara umum ditentukan oleh jenis *processor* yang ada di dalamnya. Semakin tinggi kecepatan *processor* yang digunakan dalam komputer, umumnya mempunyai kemampuan yang semakin baik, dan sudah pasti harganya akan semakin mahal. Kecepatan *processor* diukur dengan satuan MHz (*Mega Hertz*), yang menunjukkan jumlah *clock* / frekuensi operasi yang mampu dilaksanakan dalam setiap detiknya. Kemampuan yang dimiliki oleh masing-masing *processor* dapat dibuktikan dengan melakukan serangkaian pengujian. Sistem pengujian *processor* menunjukkan hasil penilaian yang berbeda yang mengacu pada tingkat kecepatan kerja aplikasi (Pramono, 1999). Pengujian yang akurat akan mencerminkan kualitas *processor* yang sesungguhnya. *Processor* berfungsi mengolah masukan menjadi keluaran. Untuk mengolah masukan diperlukan pekerjaan menghitung, misal menjumlah, mengurangi, membagi dan sebagainya. Bagian *processor* yang bertugas untuk menghitung dinamakan ALU (*Arithmetic Logic Unit*). Secara spesifik, penilaian yang dihasilkan dapat diperoleh dari total sistem *counter* yang hanya bisa diuji melalui teknik iterasi (perulangan) dalam interval waktu tertentu. Dalam standar industri internasional, terdapat tiga cara yang dapat digunakan untuk melakukan pengujian *processor*, yaitu (http://www.futuretech.vuurwerk.nl, 2002):

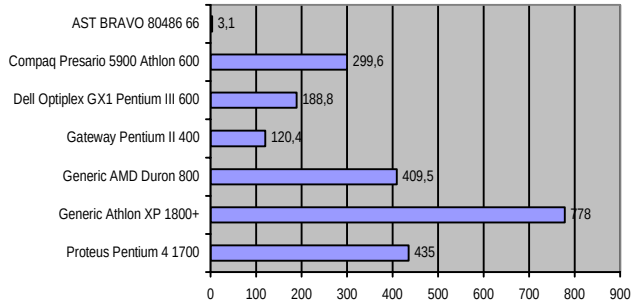
1. *MIPS*, yaitu menghitung total jumlah instruksi yang dapat dikerjakan oleh *processor* dalam setiap satu detik. Metode *MIPS* ini memiliki kelemahan, yaitu tidak dapat dipakai untuk pengujian semua jenis *processor*, karena setiap *processor* mempunyai instruksi yang berbeda. Contoh: instruksi arsitektur *processor* 32 bit (*Pentium II*) akan berbeda dengan instruksi arsitektur *processor* 64 bit (*Pentium IV*), sehingga hasilnya cenderung kurang akurat.
2. *SPEC*, yaitu menghitung total jumlah proses perhitungan matematis yang dapat dikerjakan oleh *processor* dalam interval waktu tertentu dengan standar sistem komputer minimal yang memiliki spesifikasi RAM 32MB (*SPEC92*) atau 64 MB (*SPEC95*). Perhitungan skor *SPEC92* dan *SPEC95* tidak didasarkan pada standar perhitungan, sehingga hasil yang diperoleh pada *SPEC92* dan *SPEC95* akan berbeda.
   Kedua *SPEC* di atas menghasilkan standar penilaian yang berbeda, sehingga masih banyak skor yang disembunyikan nilainya (bukan nilai umum). Pengujian *SPEC* difokuskan pada aplikasi *scientific* dan *engineering* serta banyak melibatkan operasi 64 bit.
3. *MFLOPS*, merupakan sebuah pengujian dengan melibatkan operasi matematis (*integer* dan *real*) dengan operator penjumlahan, pengurangan, perkalian dan pembagian yang dapat dikerjakan dalam setiap satu detik (huruf M berarti *Million* = jutaan). Secara teoritis, penjumlahan dan pengurangan angka *real* lebih cepat diproses daripada pembagian angka *real* itu sendiri. Metode *MFLOPS* merupakan sistem pengujian yang fleksibel karena dapat melayani pengujian berbagai kondisi CPU (variasi tipe *processor*).

MFLOPS merupakan metode yang dapat digunakan secara umum pada kondisi perangkat keras yang bermacam-macam dan tidak tergantung pada jumlah instruksi yang dimiliki. Berdasarkan hasil pantauan, penelitian seperti ini belum terlihat / pernah dilakukan oleh peneliti lain hingga saat ini. Sekalipun *tool-tool* komersial untuk pengujian *processor* bisa didapatkan di internet (www.spec.com, http://pages.tca.net/ roelof/setispy). Penelitian ini merupakan pembuktian

teori-teori yang diperoleh dari berbagai sumber, yaitu penggunaan teknik iterasi matematis sebagai alat utama pengujian. Secara singkat, ada sembilan operasi dasar yang digunakan dalam metode MFLOPS (www.passmark.com, 2002). Operasi inilah yang digunakan dalam standar internasional untuk menilai kecepatan perhitungan sebuah sistem CPU dalam satu detik, yaitu:

1. *Maths-Addition* (penjumlahan *integer*), merupakan iterasi penjumlahan *integer* dalam jangka waktu satu detik. Jumlah operasi yang dapat dikerjakan dalam satu detik tersebut merupakan skor dalam satuan *FLOPS* (*Floating Unit per Second*).
2. *Maths-Subtraction* (pengurangan *integer*), merupakan iterasi pengurangan *integer* dalam jangka waktu satu detik. Jumlah operasi yang dapat dikerjakan dalam satu detik tersebut merupakan skor dalam satuan *FLOPS* (*Floating Unit per Second*).
3. *Maths-Multiplication* (perkalian *integer*), merupakan iterasi perkalian *integer* dalam jangka waktu satu detik. Jumlah operasi yang diperoleh dalam satu detik tersebut merupakan skor dalam satuan *FLOPS* (*Floating Unit per Second*).
4. *Maths-Division* (pembagian *integer*), merupakan iterasi pembagian *integer* dalam jangka waktu satu detik. Jumlah operasi yang dapat dikerjakan dalam satu detik tersebut merupakan skor dalam satuan *FLOPS* (*Floating Unit per Second*).
5. *Maths-Floating Point Addition* (penjumlahan *real*), merupakan iterasi penjumlahan *real* dalam jangka waktu satu detik. Jumlah operasi yang dapat dikerjakan dalam satu detik tersebut merupakan skor dalam satuan *FLOPS* (*Floating Unit per Second*).
6. *Maths-Floating Point Substraction* (pengurangan *real*), merupakan iterasi pengurangan *real* dalam jangka waktu satu detik. Jumlah operasi yang dapat dikerjakan dalam satu detik tersebut merupakan skor dalam satuan *FLOPS* (*Floating Unit per Second*).
7. *Maths-Floating Point Multiplication* (perkalian *real*), merupakan iterasi perkalian *real* dalam jangka waktu satu detik. Jumlah operasi yang dapat dikerjakan dalam satu detik tersebut merupakan skor dalam satuan *FLOPS* (*Floating Unit per Second*).
8. *Maths-Floating Point Division* (pembagian *real*), merupakan iterasi pembagian *real* dalam jangka waktu satu detik. Jumlah operasi yang dapat dikerjakan dalam satu detik tersebut merupakan skor dalam satuan *FLOPS* (*Floating Unit per Second*).
9. *Maths-Maximum MegaFLOPS*, merupakan gabungan dari keseluruhan operasi di atas.

Delapan prosedur pengujian akan digabung dalam sebuah prosedur dan dieksekusi secara berurutan. Selanjutnya hasil yang diperoleh dibagi dengan 8 (sama dengan jumlah prosedur). Jumlah operasi yang diperoleh dalam satu detik tersebut merupakan skor dalam satuan *FLOPS* (*Floating Unit per Second*). Pengujian dilakukan dalam tiga buah kategori, masing-masing kategori pengujian terdiri atas operasi perhitungan *integer*, pengujian *floating point unit* dan *FLOPS* (Standar perhitungan operasi *real* per detik). *Integer* adalah susunan bilangan bulat sebagaimana bilangan cacah, misal 23, 459532, -26. Sedangkan *floating point* terdiri atas bilangan *real* yang memuat titik desimal, misal 1.003, 98394.2. Kedua tipe ini akan memerlukan fungsi perhitungan yang berbeda dalam sebuah *processor*. Untuk menghitung nilai maksimum pengujian *FLOPS*, dilakukan serangkaian pengujian proses perulangan (*looping*) dan percabangan (*branching*) yang sering digunakan dalam aplikasi komputer. Sebuah contoh hasil pengujian *processor* menggunakan software *PerformanceTest* dari *Passmark Software* yang telah dilakukan oleh Majalah CHIP adalah ditunjukkan oleh Gambar 1 (Majalah CHIP, edisi Maret 2002).



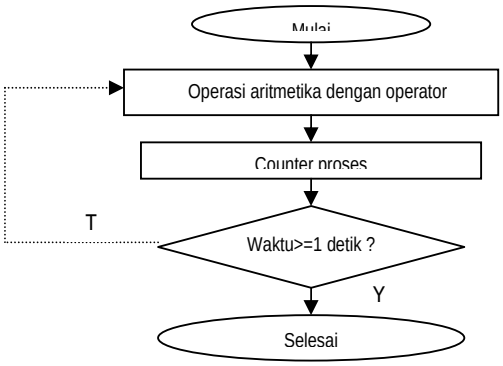
Gambar 1. Hasil Pengujian *Maths-Maximum MegaFLOPS* menggunakan Program Kinerja *PerformanceTest* dari *Passmark Software*

FLOPS adalah operasi angka *floating point* yang bisa dihasilkan dalam waktu satu detik. Secara teoritis, sebuah *processor* biasanya dapat menghasilkan jumlah hingga jutaan FLOPS untuk tipe-tipe i586 dan i686. Ini adalah hasil pengujian standar internasional yang biasanya digunakan untuk membandingkan kinerja dari berbagai sistem komputer. Angka teoritis tersebut akan dihasilkan dalam kondisi maksimal, yaitu pada saat *processor* belum mendapatkan banyak instruksi di luar rutin perhitungan. Dengan kalimat lain angka tersebut akan diperoleh saat komputer baru dinyalakan atau *boot* ulang.

Pada beberapa tipe *processor*, secara otomatis pengujian akan diproses oleh beberapa instruksi yang memang sengaja disediakan oleh masing-masig vendor *processor*. Sebagai contoh, Intel telah menyediakan instruksi *MMX* (*Multimedia Extensions*), *SSE* (*SIMD Streaming Extensions*) dan *SIMD* (*Single Instruction Multiple Data*) pada *processor*nya untuk *processor* Intel Pentium III. Instruksi-instruksi tersebut sengaja dirancang dengan tujuan mengoptimalkan *processor* untuk meng-*handle* berbagai aplikasi yang membutuhkan operasi perhitungan seperti *floating point unit*. Kecepatan *processor* diukur dalam satuan MHz dan MIPS. MHz adalah singkatan dari *Mega Hertz*, dimana semakin besar angkanya, semakin tinggi pula kecepatannya. Pemakai komputer seringkali merasakan perbedaan kecepatan *game* yang dijalankan di komputer generasi i386, i486, i586 atau i686. Hal tersebut terjadi karena terdapat perbedaan kecepatan pada *processor*nya. Sedangkan MIPS adalah singkatan dari *Million Instruction Per Second* atau banyaknya operasi yang dapat dikerjakan dalam satuan jutaan perintah yang dapat diproses dalam waktu satu detik. Hasil dari setiap pengujian dilakukan dari unit perhitungan yang berbeda. Kedelapan pengujian pertama, digunakan untuk menghitung jumlah operasi *integer* dan *real* dan biasanya digunakan juga untuk menguji instruksi MMX dari sebuah *processor*.

Sistem yang dikembangkan dalam penelitian ini direncanakan untuk pengujian secara *single user* dan mampu menangani:

1. Pengujian *processor* secara berulang menggunakan 9 metode *MFLOPS*.
   Sistem akan memulai pengujian dengan menerapkan konsep *realtime* (dedikasi *resources* tertinggi) pada saat berjalan (*runtime*).
2. Deteksi perangkat keras dan perangkat lunak secara minimal.
   Deteksi perangkat keras meliputi, jumlah *processor* yang digunakan, desain *processor*, tipe *processor*, vendor *processor*, serta memori fisik yang digunakan, Perangkat lunak yang dideteksi termasuk dalam cakupan sistem operasi yang digunakan, yaitu nama user aktif, direktori penting (windows, sistem, aplikasi, dan temporari) yang diikutsertakan sebagai bahan pertimbangan dan informasi.
3. Uji perbandingan yang memuat hasil pengujian sistem lain yang telah dimasukkan ke dalam kode program dilengkapi dengan grafik dan statistik angka penilaian. Sistem dapat melakukan perbandingan (*comparing*) dengan data *internal* berupa data hasil uji sistem lain yang telah dimasukkan ke dalam aplikasi serta data *external* berupa data hasil uji sistem lain dalam bentuk file dengan ekstensi *.BM.



Gambar 2. Alur utama proses pengujian *processor*

Gambar 2 menunjukkan alur utama proses pengujian *processor* yang dilakukan pada saat pengujian berlangsung. Sistem akan mencatat *counter* secara berulang (iterasi) selama waktu satu detik, sementara itu sistem tetap akan mengerjakan operasi matematis dengan jenis operator yang berbeda. Variabel *counter* pada akhir pengujian menunjukkan hasil penilaian dalam satuan *FLOPS* (jumlah operasi yang dilakukan dalam satu detik). Satuan *MFLOPS* menganut perhitungan yang berbeda, yaitu hasil pembagian *FLOPS* dengan bilangan 1.000.000 (Mega=1.000.000) dan kemudian hasilnya dikalikan dengan konstanta 100 (konstanta max). Hasil akhir tersebut menunjukkan satuan *MFLOPS*.

Secara singkat, program yang dikembangkan memuat:

1. Kode *assembler*, digunakan untuk mendeteksi kecepatan *processor* yang akan diuji dalam satuan MHz, misal 649,0 MHz.
2. Rutin *Win32 API*, beberapa rutin Win32 API yang digunakan antara lain:
   a. *ShellExecute*, berfungsi untuk mengeksekusi program lain. Pada sistem, rutin ini terdapat pada *window* "About", yaitu pada *shortcut link* alamat e-mail dan alamat *website*.
   b. *GetTickCount*, berfungsi untuk mendapatkan perbedaan waktu dalam level *milisecond*. Fungsi ini digunakan pada saat mendeteksi kecepatan *processor* dan dalam proses pengujian.
   c. *SetPriorityClass*, berfungsi untuk mengatur prioritas sistem agar sumber daya komputer (*resources*) yang digunakan berada dalam tingkatan maksimal.
   d. *GetWindowsDirectory*, berfungsi untuk mendapatkan direktori Windows. Biasanya nilai yang dihasilkan berupa *string* yang berisi "C:\Windows".
   e. *GetSystemDirectory*, berfungsi untuk mendapatkan direktori system Windows. Biasanya nilai yang dihasilkan berupa *string* "C:\Windows\System".
   f. *GetTempPath*, berfungsi untuk mendapatkan direktori *temporary*. Biasanya nilai yang dihasilkan berupa *string* "C:\Windows\Temp".
   g. *GetUserName*, berfungsi untuk mendapatkan nama pengguna komputer yang aktif.
   h. *GetVersionEx*, berfungsi untuk menentukan versi dari sistem operasi Windows yang digunakan.
   i. *GlobalMemoryStatus*, berfungsi untuk mendapatkan informasi memori, yaitu RAM atau tambahan *swap memory harddisk* lainnya yang terdapat dalam CPU.
3. Unit DCU standar Borland Delphi 5.0
   a. *ShellApi*, digunakan untuk mengakses rutin *Win32API*.
   b. *Registry*, digunakan untuk mengakses *registry Windows*. Fungsi ini terdapat dalam pendeteksian tipe dan vendor *processor*, dalam *window* "My Computer" dan *window* "Compare System".

PEMBAHASAN

Sistem aplikasi yang dihasilkan terdiri atas empat antar muka utama, yaitu:

1. Menu Utama
2. Menu Diagnosa, terdiri dari 4 pilihan yaitu: My Computer, Run Benchmark, Compare System dan Exit .
3. Menu Setup, hanya terdiri dari 1 pilihan yaitu: Print Setup, yaitu digunakan untuk pengaturan dan konfigurasi printer.
4. Menu Windows, yaitu merupakan fasilitas dari form MDI, terdiri atas: Tile, Cascade, Arrange All, Help

Konfigurasi perangkat keras dan perangkat lunak minimal yang direkomendasikan agar sistem aplikasi dapat bekerja dengan baik adalah:

1. PC dengan prosessor *Pentium 133 MMX*
2. RAM (*Memory*) 16 *MegaByte*
3. Kapasitas Harddisk 500 *MegaByte*
4. Sistem operasi: *Windows9x, WindowsNT 4.0*, atau *Windows2000*.

Tabel 1. Hasil penilaian dalam satuan *MFLOPS* untuk *RAM* 128

No	Operasi Dasar dalam Metode *MFLOPS*	Uji Komputer	*P233 MMX, RAM 128 MB*	*PII 350 Mhz, RAM 128 MB*	*AMD K6-2 450 MHz, RAM 128 MB*	*AMD K7 650 MHz, RAM 128 MB*
1.	*Maths – Addition*	*+ Integer*	204	403	320	649
2.	*Maths – Subraction*	*- Integer*	205	403	312	610
3.	*Maths-Multiplication*	** Integer*	226	403	355	626
4.	*Maths-Division*	*/ Integer*	203	389	311	600
5.	*Maths-Floating Point Addition*	*+ Real*	189	325	304	587
6.	*Maths-Floating Point Subraction*	*- Real*	204	381	298	594
7.	*Maths - Floating Point Multiplication*	** Real*	188	369	306	592
8.	*Maths - Floating Point Division*	*/ Real*	105	290	213	575
9.	*Maths - Maximum MegaFLOPS*	MaxMFL	40	120	143	287

Sumber: Ekawati, 2002

Sistem aplikasi selanjutnya digunakan untuk menguji beberapa komputer dengan spesifikasi berbeda dan menggunakan sistem operasi yang sama, yaitu *Windows98*. Hasil penilaian dalam satuan *MFLOPS* untuk *RAM* 128 ditunjukkan oleh Tabel 1. Berdasarkan data dalam Tabel 1, maka penilaian dalam satuan *MFLOPS* untuk *RAM* 128 adalah sebagai berikut:

1. Hasil penilaian menunjukkan bahwa hasil uji *processor AMD K6-2 450 MHz, RAM 128 MB* rata-rata lebih unggul dibandingkan dengan *processor Pentium II 350 MHz, RAM 128 MB*. Dari hasil ini, dapat disimpulkan bahwa kecepatan *processor* jenis *Pentium* tidak selalu lebih unggul, tetapi masih dipengaruhi oleh partisipasi *RAM* dalam hal kecepatan transfer data.
2. Rata-rata kecepatan operasi untuk bilangan *integer* pada konfigurasi *RAM 128*, lebih cepat dibandingkan dengan bilangan *real*.
3. Untuk kategori bilangan *integer*, operasi perkalian umumnya lebih cepat dilakukan daripada operasi yang lain, sedangkan operasi pembagian lebih lambat daripada jenis operasi yang lain.
4. Untuk kategori bilangan *real*, operasi pengurangan pada *processor Pentium* dan operasi perkalian pada *processor AMD* lebih cepat dilakukan daripada operasi yang lain dan operasi pembagian lebih lambat daripada operasi yang lain.
5. Operasi yang paling lama dilakukan adalah operasi pembagian *real*.

Selanjutnya, hasil penilaian dalam satuan *MFLOPS* untuk *RAM* 64 ditunjukkan oleh Tabel 2.

Tabel 2. Hasil penilaian dalam satuan MFLOPS untuk RAM 64

No	Operasi Dasar dalam Metode *MFLOPS*	Uji Komputer	*P233 MMX, RAM 64 MB*	*Celeron 333 Mhz, RAM 64 MB*	*AMD K6-3 500 MHz, RAM 64 MB*	*PIII 667 MHz, RAM 64 MB*
1.	*Maths – Addition*	*+ Integer*	200	399	315	644
2.	*Maths – Subraction*	*- Integer*	200	398	306	606
3.	*Maths – Multiplication*	** Integer*	220	397	351	620
4.	*Maths – Division*	*/ Integer*	198	384	306	594
5.	*Maths - Floating Point Addition*	*+ Real*	183	321	298	582
6.	*Maths - Floating Point Subraction*	*- Real*	200	375	294	588
7.	*Maths - Floating Point Multiplication*	** Real*	183	364	300	588
8.	*Maths - Floating Point Division*	*/ Real*	99	284	209	570
9.	*Maths – Maximum MegaFLOPS*	MaxMFL	36	116	138	282

Sumber: Ekawati, 2002

Berdasarkan data hasil pengujian dalam Tabel 2 untuk satuan *MFLOPS* dengans *RAM* 64 adalah:

1. Hasil penilaian *menunjukkan* bahwa hasil uji *processor AMD K6-2 500 MHz, RAM 64 MB* rata-rata lebih unggul dibandingkan dengan *processor Celeron 333 Mhz, RAM 64 MB*, tetapi masih dipengaruhi oleh partisipasi *RAM* dalam hal terhadap kecepatan transfer data.
2. Rata-rata kecepatan operasi untuk bilangan *integer* pada konfigurasi *RAM 64*, lebih cepat dibandingkan dengan bilangan *real*.
3. Untuk kategori bilangan *integer*, operasi perkalian umumnya lebih cepat dilakukan daripada operasi yang lain dan operasi pembagian lebih lambat dilakukan daripada operasi yang lain.
4. Untuk kategori bilangan *real*, operasi pengurangan pada *processor Pentium* serta dan operasi perkalian pada *processor AMD* lebih cepat dilakukan daripada operasi yang lain umumnya

lebih cepat dilakukan daripada operasi yang lain dan operasi pembagian lebih lambat dilakukan daripada operasi yang lain.

5. Operasi yang paling lama lama dilakukan adalah operasi pembagian *real*.

Perhitungan cacah operasi yang dapat dilaksanakan untuk setiap pengujian dalam satuan detik adalah menggunakan *semantic code* berikut:

```
StartTime:=GetTickCount;
repeat
 inc(i);
 [kode perhitungan matematis dan operatornya diletakkan disini]
 busy:=busy+1;
 EndTime:=GetTickCount - StartTime;
until ((EndTime/CLOCK_TICK)>=1);
```

Hasil yang diperoleh merupakan skor dengan penggunaan operator tertentu untuk kemudian ditampilkan dalam bentuk grafik (dengan komponen *TChart*) serta secara opsional dapat dibandingkan dengan konfigurasi komputer yang memiliki *processor* berbeda yang telah disediakan sebagai pembanding. Masing-masing hasil pengujian dapat dijelaskan sebagai berikut:

1. *Maths-Integer Addition* (Diagnosa Penjumlahan *Integer*), merupakan operasi penjumlahan *integer* yang prosesnya dikerjakan berulang-ulang dalam waktu satu detik. *Operand* yang dipakai adalah bilangan 5 dengan tipe data *double* serta nilai inisialisasi awal 1000. Untuk hasil yang optimal, proses ini sendiri akan di kerjakan sebanyak 3x. Kemudian, skor-nya dibagi 3 untuk mendapat skor rata-rata. Hal ini dilakukan karena kemampuan *processor* dalam melakukan operasi penjumlahan dalam satu waktu tertentu, tidak dapat menjamin dihasilkan skor yang optimal. Bilangan yang dipakai merupakan bilangan *integer* mulai bilangan dengan angka satuan sampai bilangan dengan angka ratusan ribu (+ dan -).
2. *Maths-Integer Subtraction* (Diagnosa Pengurangan *Integer*), merupakan operasi pengurangan *integer* yang prosesnya dikerjakan berulang-ulang dalam waktu satu detik. *Operand* yang dipakai adalah bilangan 5 dengan tipe data *double* serta nilai inisialisasi awal 1000. Untuk hasil yang optimal, proses ini sendiri akan di kerjakan sebanyak 3x. Kemudian, skor-nya dibagi 3 untuk mendapat skor rata-rata. Hal ini dilakukan karena kemampuan sebuah *processor* dalam melakukan sebuah tugas pengurangan dalam satu waktu, tidak dapat menjamin dihasilkan skor yang optimal. Bilangan yang dipakai merupakan bilangan *integer* mulai bilangan dengan angka satuan sampai bilangan dengan angka ratusan ribu (+ dan -).
3. *Maths-Integer Multiplication* (Diagnosa Perkalian *Integer*), merupakan operasi perkalian *integer* yang prosesnya dikerjakan berulang-ulang dalam waktu satu detik. *Operand* yang dipakai adalah bilangan 2 dengan tipe data *longint* serta nilai inisialisasi awal 2. Untuk hasil yang optimal, proses ini sendiri akan di kerjakan sebanyak 3x. Kemudian, skor-nya dibagi 3 untuk mendapat skor rata-rata. Hal ini dilakukan karena kemampuan sebuah *processor* dalam melakukan sebuah tugas perkalian dalam satu waktu, tidak dapat menjamin dihasilkan skor yang optimal. Bilangan yang dipakai merupakan bilangan *integer* mulai bilangan dengan angka satuan sampai bilangan dengan angka ratusan ribu (+ dan -).
4. *Maths-Integer Division* (Diagnosa Pembagian *Integer*), merupakan operasi pembagian *integer* yang prosesnya dikerjakan berulang-ulang dalam waktu satu detik. *Operand* yang dipakai adalah bilangan 5 dengan tipe data *double* serta nilai inisialisasi awal 1000. Untuk hasil yang optimal, proses ini sendiri akan di kerjakan sebanyak 3x. Kemudian, skor-nya dibagi 3 untuk mendapat skor rata-rata. Hal ini dilakukan karena kemampuan sebuah *processor* dalam melakukan sebuah tugas pembagian dalam satu waktu, tidak dapat menjamin dihasilkan skor yang optimal. Bilangan yang dipakai merupakan bilangan *integer* mulai bilangan dengan angka satuan sampai bilangan dengan angka ratusan ribu (+ dan -).
5. *Maths-Floating Point Addition* (Diagnosa Penjumlahan *Real*), merupakan operasi penjumlahan *real* yang prosesnya dikerjakan berulang-ulang dalam waktu satu detik. Operand yang dipakai adalah bilangan 0,999999 dengan tipe data *double* serta nilai inisialisasi awal 0,000001. Untuk hasil yang optimal, proses ini sendiri akan di kerjakan sebanyak 3x. Kemudian, skor-nya dibagi 3 untuk mendapat skor rata-rata. Hal ini dilakukan karena kemampuan sebuah *processor* dalam melakukan sebuah tugas penjumlahan dalam satu waktu, tidak dapat menjamin dihasilkan skor yang optimal. Bilangan yang dipakai merupakan bilangan *real* mulai bilangan dengan angka satuan (+ dan -) sampai bilangan dengan angka ratusan ribu serta pemakaian *real* sampai dengan 5 digit angka di belakang koma.

6. *Maths-Floating Point Substraction* (Diagnosa Pengurangan *Real*), merupakan operasi pengurangan *real* yang prosesnya dikerjakan berulang-ulang dalam waktu satu detik. *Operand* yang dipakai adalah bilangan 0,005 dengan tipe data *double* serta nilai inisialisasi awal 1000. Untuk hasil yang optimal, proses ini sendiri akan di kerjakan sebanyak 3x. Kemudian, skor-nya dibagi 3 untuk mendapat skor rata-rata. Hal ini dilakukan karena kemampuan sebuah *processor* dalam melakukan sebuah tugas pengurangan dalam satu waktu, tidak dapat menjamin dihasilkan skor yang optimal. Bilangan yang dipakai merupakan bilangan *real* mulai bilangan dengan angka satuan (+ dan -) sampai bilangan dengan angka ratusan ribu (+ dan -) serta pemakaian *real* sampai dengan 5 digit angka di belakang koma.
7. *Maths-Floating Point Multiplication* (Diagnosa Perkalian *Real*), merupakan operasi perkalian *real* yang prosesnya dikerjakan berulang-ulang dalam waktu satu detik. *Operand* yang dipakai adalah bilangan 0,005 dengan tipe data *double* serta nilai inisialisasi awal 1000. Untuk hasil yang optimal, proses ini sendiri akan di kerjakan sebanyak 3x. Kemudian, skor-nya dibagi 3 untuk mendapat skor rata-rata. Hal ini dilakukan karena kemampuan sebuah *processor* dalam melakukan sebuah tugas perkalian dalam satu waktu, tidak dapat menjamin dihasilkan skor yang optimal. Bilangan yang dipakai merupakan bilangan *real* mulai bilangan dengan angka satuan sampai bilangan dengan angka ratusan ribu (+ dan -) serta pemakaian *real* sampai dengan 5 digit angka di belakang koma.
8. *Maths-Floating Point Division* (Diagnosa Pembagian *Real*), merupakan operasi pembagian *real* yang prosesnya dikerjakan berulang-ulang dalam waktu satu detik. Operand yang dipakai adalah bilangan 5 dengan tipe data *double* serta nilai inisialisasi awal 1000. Untuk hasil yang optimal, proses ini sendiri akan di kerjakan sebanyak 3x. Kemudian, skor-nya dibagi 3 untuk mendapat skor rata-rata. Hal ini dilakukan karena kemampuan sebuah *processor* dalam melakukan sebuah tugas pembagian dalam satu waktu, tidak dapat menjamin dihasilkan skor yang optimal. Bilangan yang dipakai merupakan bilangan *real* mulai bilangan dengan angka satuan sampai bilangan dengan angka ratusan ribu (+ dan -) serta pemakaian *real* sampai dengan 5 digit angka di belakang koma.
9. *Maths-Maximum MegaFLOPS*, merupakan nilai maximum yang dapat dicapai oleh keseluruhan proses diatas. Seluruh proses dikerjakan secara berurutan dengan metode yang sama, sehingga akan diperoleh sebuah angka dimana proses yang dikerjakan dalam kondisi *non-stop*.

Berikut adalah prosedur-prosedur yang digunakan dalam program aplikasi yang dikembangkan:

```
unit UBenchmark;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs,
  TeEngine, Series, ExtCtrls, TeeProcs, Chart, ComCtrls, StdCtrls,
IniFiles;
const CLOCK_TICK : Double = 1000;
type
  TFBenchmark = class(TForm)
    ProgressBar1: TProgressBar;
    State: TLabel;
    Label1: TLabel;
    Image1: TImage;
    Timer1: TTimer;
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    procedure CreateMDIChild(const Name: string);
  public
    { Public declarations }
    procedure state1();
    procedure state2();
    procedure state3();
    procedure state4();
    procedure state5();
    procedure state6();
    procedure state7();
    procedure state8();
    procedure state9();
    procedure filetmp();
  end;
var
  FBenchmark: TFBenchmark;
  step,
score,score1,score2,score3,counter,scoretotal1,scoretotal2,scoretotal3
: integer;
  StartTime,EndTime,i : Double;
  pWinIni : TIniFile;
implementation
uses UGraph;
{$R *.DFM}

procedure TFBenchmark.CreateMDIChild(const Name: string);
var
  Child: TFGraph;
begin
  { create a new MDI child window }
  Child := TFGraph.Create(Application);
  Child.Caption := Name;
  Child.Left:=0;
  Child.Top:=0;
end;

procedure TFBenchmark.Timer1Timer(Sender: TObject);
begin
state1();
Timer1.Enabled:=False;
end;
procedure TFBenchmark.filetmp();
begin
pWinIni:=TiniFile.Create(ExtractFilePath(Application.ExeName)+'Benc
hMath.tmp');
case (step) of
      1 : begin
        pWinIni.WriteInteger('state1','score1',score1);
```

```
            pWinIni.WriteInteger('state1','score2',score2);
            pWinIni.WriteInteger('state1','score3',score3);
            end;
        2 : begin
            pWinIni.WriteInteger('state2','score1',score1);
            pWinIni.WriteInteger('state2','score2',score2);
            pWinIni.WriteInteger('state2','score3',score3);
            end;
        3 : begin
            pWinIni.WriteInteger('state3','score1',score1);
            pWinIni.WriteInteger('state3','score2',score2);
            pWinIni.WriteInteger('state3','score3',score3);
            end;
        4 : begin
            pWinIni.WriteInteger('state4','score1',score1);
            pWinIni.WriteInteger('state4','score2',score2);
            pWinIni.WriteInteger('state4','score3',score3);
            end;
        5 : begin
            pWinIni.WriteInteger('state5','score1',score1);
            pWinIni.WriteInteger('state5','score2',score2);
            pWinIni.WriteInteger('state5','score3',score3);
            end;
        6 : begin
            pWinIni.WriteInteger('state6','score1',score1);
            pWinIni.WriteInteger('state6','score2',score2);
            pWinIni.WriteInteger('state6','score3',score3);
            end;
        7 : begin
            pWinIni.WriteInteger('state7','score1',score1);
            pWinIni.WriteInteger('state7','score2',score2);
            pWinIni.WriteInteger('state7','score3',score3);
            end;
        8 : begin
            pWinIni.WriteInteger('state8','score1',score1);
            pWinIni.WriteInteger('state8','score2',score2);
            pWinIni.WriteInteger('state8','score3',score3);
            end;
        9 : begin
            pWinIni.WriteInteger('state9','score1',scoretotal1);
            pWinIni.WriteInteger('state9','score2',scoretotal2);
            pWinIni.WriteInteger('state9','score3',scoretotal3);
            end;
        end;
pWinIni.Free();
end;

procedure TFBenchmark.state1();
begin
State.Caption:='Integer Addition [Diagnosa Penjumlahan Integer]';
State.Refresh();
for counter:= 1 to 3 do
    begin
    score:=0;
    i:=1000;
    StartTime:=GetTickCount;
    repeat
        inc(score);
        i:=i+5;
        EndTime:=GetTickCount-StartTime;
    until ((EndTime/CLOCK_TICK)>1);
    case counter of
        1 : score1:=score;
        2 : score2:=score;
        3 : score3:=score;
        end;
    ProgressBar1.Position:=ProgressBar1.Position+1;
    end;
filetmp();
//ShowMessage(Format('Elapsed Time : %0.2f
seconds',[EndTime/CLOCK_TICK]));
step:=2;
state2();
end;

procedure TFBenchmark.state2();
begin
State.Caption:='Integer Subtraction [Diagnosa Pengurangan Integer]';
State.Refresh();
for counter:= 1 to 3 do
    begin
    score:=0;
    i:=1000;
    StartTime:=GetTickCount;
    repeat
        inc(score);
        i:=i-5;
        EndTime:=GetTickCount-StartTime;
    until ((EndTime/CLOCK_TICK)>1);
    case counter of
        1 : score1:=score;
        2 : score2:=score;
        3 : score3:=score;
        end;
    ProgressBar1.Position:=ProgressBar1.Position+1;
    end;
filetmp();
step:=3;
state3();
end;

procedure TFBenchmark.state3();
var i : longint;
begin
State.Caption:='Integer Multiplication [Diagnosa Perkalian Integer]';
State.Refresh();
for counter:= 1 to 3 do
    begin
    score:=0;
    i:=2;
    StartTime:=GetTickCount;
    repeat
        inc(score);
        i:=i*2;
        EndTime:=GetTickCount-StartTime;
    until ((EndTime/CLOCK_TICK)>1);
    case counter of
        1 : score1:=score;
        2 : score2:=score;
        3 : score3:=score;
        end;
    ProgressBar1.Position:=ProgressBar1.Position+1;
    end;
filetmp();
step:=4;
state4();
end;

procedure TFBenchmark.state4();
begin
State.Caption:='Integer Division [Diagnosa Pembagian Integer]';
State.Refresh();
for counter:= 1 to 3 do
    begin
    score:=0;
    i:=1000;
    StartTime:=GetTickCount;
    repeat
        inc(score);
        i:=i/5;
        EndTime:=GetTickCount-StartTime;
    until ((EndTime/CLOCK_TICK)>1);
    case counter of
        1 : score1:=score;
        2 : score2:=score;
        3 : score3:=score;
        end;
    ProgressBar1.Position:=ProgressBar1.Position+1;
    end;
filetmp();
step:=5;
state5();
end;

procedure TFBenchmark.state5();
begin
State.Caption:='Floating Point Addition [Diagnosa Penjumlahan
Desimal]';
State.Refresh();
for counter:= 1 to 3 do
```

```
        begin
        score:=0;
        i:=1000;
        StartTime:=GetTickCount;
        repeat
            inc(score);
            i:=i+0.005;
            EndTime:=GetTickCount-StartTime;
        until ((EndTime/CLOCK_TICK)>1);
        case counter of
            1 : score1:=score;
            2 : score2:=score;
            3 : score3:=score;
            end;
        ProgressBar1.Position:=ProgressBar1.Position+1;
        end;
filetmp();
step:=6;
state6();
end;

procedure TFBenchmark.state6();
begin
State.Caption:='Floating Point Subtraction [Diagnosa Pengurangan
Desimal]';
State.Refresh();
for counter:= 1 to 3 do
        begin
        score:=0;
        i:=1000;
        StartTime:=GetTickCount;
        repeat
            inc(score);
            i:=i-0.005;
            EndTime:=GetTickCount-StartTime;
        until ((EndTime/CLOCK_TICK)>1);
        case counter of
            1 : score1:=score;
            2 : score2:=score;
            3 : score3:=score;
            end;
        ProgressBar1.Position:=ProgressBar1.Position+1;
        end;
filetmp();
step:=7;
state7();
end;

procedure TFBenchmark.state7();
begin
State.Caption:='Floating Point Multiplication [Diagnosa Perkalian
Desimal]';
State.Refresh();
for counter:= 1 to 3 do
        begin
        score:=0;
        i:=1000;
        StartTime:=GetTickCount;
        repeat
            inc(score);
            i:=i*0.005;
            EndTime:=GetTickCount-StartTime;
        until ((EndTime/CLOCK_TICK)>1);
        case counter of
            1 : score1:=score;
            2 : score2:=score;
            3 : score3:=score;
            end;
        ProgressBar1.Position:=ProgressBar1.Position+1;
        end;
filetmp();
step:=8;
state8();
end;

procedure TFBenchmark.state8();
begin
State.Caption:='Floating Point Division [Diagnosa Pembagian
Desimal]';
State.Refresh();
for counter:= 1 to 3 do
        begin
        score:=0;
        //i:=1000;
        i:=0.000001;
        StartTime:=GetTickCount;
        repeat
            inc(score);
            i:=i/0.999999;
            EndTime:=GetTickCount-StartTime;
        until ((EndTime/CLOCK_TICK)>1);
        case counter of
            1 : score1:=score;
            2 : score2:=score;
            3 : score3:=score;
            end;
        ProgressBar1.Position:=ProgressBar1.Position+1;
        end;
filetmp();
step:=9;
state9();
end;

procedure TFBenchmark.state9();
var dumm : longint;
begin
State.Caption:='Max MegaFLOPS';
State.Refresh();
for counter:= 1 to 3 do
        begin
        score:=0;
        i:=1000;
        StartTime:=GetTickCount;
        repeat
            inc(score);
            i:=i+5;
            EndTime:=GetTickCount-StartTime;
        until ((EndTime/CLOCK_TICK)>1);
        ProgressBar1.Position:=ProgressBar1.Position+1;
        //score:=0;
        i:=1000;
        StartTime:=GetTickCount;
        repeat
            inc(score);
            i:=i-5;
            EndTime:=GetTickCount-StartTime;
        until ((EndTime/CLOCK_TICK)>1);
        ProgressBar1.Position:=ProgressBar1.Position+1;
        //score:=0;
        dumm:=2;
        StartTime:=GetTickCount;
        repeat
            inc(score);
            dumm:=dumm*2;
            EndTime:=GetTickCount-StartTime;
        until ((EndTime/CLOCK_TICK)>1);
        ProgressBar1.Position:=ProgressBar1.Position+1;
        //score:=0;
        i:=1000;
        StartTime:=GetTickCount;
        repeat
            inc(score);
            i:=i/5;
            EndTime:=GetTickCount-StartTime;
        until ((EndTime/CLOCK_TICK)>1);
        ProgressBar1.Position:=ProgressBar1.Position+1;
        score:=0;
        i:=1000;
        StartTime:=GetTickCount;
        repeat
            inc(score);
            i:=i+0.005;
            EndTime:=GetTickCount-StartTime;
        until ((EndTime/CLOCK_TICK)>1);
        ProgressBar1.Position:=ProgressBar1.Position+1;
        //score:=0;
        i:=1000;
        StartTime:=GetTickCount;
        repeat
            inc(score);
```

```
            i:=i-0.005;
            EndTime:=GetTickCount-StartTime;
        until ((EndTime/CLOCK_TICK)>1);
        ProgressBar1.Position:=ProgressBar1.Position+1;
        //score:=0;
        i:=1000;
        StartTime:=GetTickCount;
        repeat
            inc(score);
            i:=i*0.005;
            EndTime:=GetTickCount-StartTime;
        until ((EndTime/CLOCK_TICK)>1);
        ProgressBar1.Position:=ProgressBar1.Position+1;
        //score:=0;
        i:=0.000001;
        //i:=0.1;
        StartTime:=GetTickCount;
        repeat
            inc(score);
            i:=i/0.999999;
            //i:=i/2;
            EndTime:=GetTickCount-StartTime;
        until ((EndTime/CLOCK_TICK)>1);
        ProgressBar1.Position:=ProgressBar1.Position+1;
        case counter of
            1 : scoretotal1:=score div 8;
            2 : scoretotal2:=score div 8;
            3 : scoretotal3:=score div 8;
            end;
        end;
filetmp();
score:=0;
i:=1000;
step:=1;
State.Caption:='State : Initializing...';
Timer1.Enabled:=False;
Close();
Screen.Cursor:=crDefault;
CreateMDIChild('Summary System : ');
ProgressBar1.Position:=0;
end;

procedure TFBenchmark.FormCreate(Sender: TObject);
begin
step:=1;
if SetPriorityClass(Application.Handle,REALTIME_PRIORITY_CLASS)
THEN
      Application.MessageBox('REALTIME
GAGAL','BenchMath',MB_OK);
end;

end.

unit UMyComputer;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs,
  StdCtrls, Buttons, shellapi, Registry;
type
  TFMyComputer = class(TForm)
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    GroupBox2: TGroupBox;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    BitBtn1: TBitBtn;
    NumProcessor: TLabel;
    ProcessorDesign: TLabel;
    ProcessorType: TLabel;
    ProcessorVendor: TLabel;
    TotalMemory: TLabel;
    AvailMemory: TLabel;
    MemoryInUse: TLabel;
    OperatingSystem: TLabel;
    UserName: TLabel;
    WinPath: TLabel;
    SysPath: TLabel;
    ProgramPath: TLabel;
    TempPath: TLabel;
    GroupBox3: TGroupBox;
    Label14: TLabel;
    Label15: TLabel;
    Label16: TLabel;
    Label17: TLabel;
    Label18: TLabel;
    Label19: TLabel;
    Label20: TLabel;
    Label21: TLabel;
    Label22: TLabel;
    Score1: TLabel;
    Score2: TLabel;
    Score3: TLabel;
    Score4: TLabel;
    Score5: TLabel;
    Score6: TLabel;
    Score7: TLabel;
    Score8: TLabel;
    Score9: TLabel;
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FMyComputer: TFMyComputer;
  siSysInfo : SYSTEM_INFO;

implementation
{$R *.DFM}

procedure TFMyComputer.FormShow(Sender: TObject);
const
    cnMaxLen=254;
var memStatus : TMemoryStatus;
    pRegIni : TRegIniFile;
    TempDir: array[0..255] of Char;
    versiinfo : TOsVersionInfo;
    sUserName : string;
    dwUserNameLen : DWord;
    t: DWORD;
    mhi, mlo, nhi, nlo: DWORD;
    t0, t1, chi, clo, shr32: Comp;

begin
//baca kecepatan processor dengan kode assembler
  shr32 := 65536;
  shr32 := shr32 * 65536;
  t := GetTickCount;
  while t = GetTickCount do begin end;
  asm
    DB 0FH
    DB 031H
    mov mhi,edx
    mov mlo,eax
  end;
  while GetTickCount < (t + 1000) do begin end;
  asm
    DB 0FH
    DB 031H
    mov nhi,edx
    mov nlo,eax
  end;
  chi := mhi;
  if mhi < 0 then chi := chi + shr32;
  clo := mlo;
  if mlo < 0 then clo := clo + shr32;
  t0 := chi * shr32 + clo;
```

```
  chi := nhi;
  if nhi < 0 then chi := chi + shr32;
  clo := nlo;
  if nlo < 0 then clo := clo + shr32;
  t1 := chi * shr32 + clo;
  t1:=(t1 - t0) / 1E6;

//arsitektur CPU
  GetSystemInfo(siSysInfo);

NumProcessor.Caption:=IntToStr(Ord(siSysInfo.dwNumberOfProcesso
rs));

ProcessorDesign.Caption:=IntToStr(Ord(siSysInfo.dwProcessorType));
  pRegIni := TRegIniFile.Create;
  pRegIni.RootKey:=HKEY_LOCAL_MACHINE;

pRegIni.OpenKey('Hardware\Description\System\CentralProcessor\',tr
ue);
  ProcessorType.Caption:=(pRegIni.ReadString('0', 'Identifier',
'<none>')+Format(' @  %.1f MHz', [t1]));
  ProcessorVendor.Caption:=(pRegIni.ReadString('0',
'VendorIdentifier', '<none>'))  ;
  pRegIni.CloseKey;
  pRegIni.Free ;

//Cek memori
  memStatus.dwLength := SizeOf(memStatus) ;
  GlobalMemoryStatus(memStatus);
  TotalMemory.Caption:=IntToStr(memStatus.dwTotalPhys)+' byte';
  AvailMemory.Caption:=IntToStr(memStatus.dwMemoryLoad)+' %';
  MemoryInUse.Caption:=IntToStr(memStatus.dwAvailPhys)+' byte';

//cek versi Windows
  versiinfo.dwOSVersionInfoSize := sizeof(TOsVersionInfo);
  GetVersionEx(versiinfo);
  case versiinfo.dwPlatformId of
      VER_PLATFORM_WIN32S :
OperatingSystem.Caption:='Windows 3.x (+ Win32s)';
      VER_PLATFORM_WIN32_WINDOWS :
OperatingSystem.Caption:='Windows 95 or newer';
      VER_PLATFORM_WIN32_NT :
OperatingSystem.Caption:=Format('Windows NT
%d.%d',[versiinfo.dwMajorVersion,versiinfo.dwMinorVersion]);
  end;

  dwUserNameLen:=cnMaxLen-1;
  SetLength(sUserName, cnMaxLen);
  GetUserName(Pchar(sUserName), dwUserNameLen);
  SetLength(sUserName, dwUserNameLen);
  UserName.Caption:=sUserName;
  if dwUserNameLen=cnMaxLen-1 then
      UserName.Caption:='<none>';

//direktori system
  GetWindowsDirectory(@TempDir,sizeof(TempDir));
  WinPath.Caption := StrPas(TempDir);
  GetSystemDirectory(@TempDir,sizeof(TempDir));
  SysPath.Caption := StrPas(TempDir);
  ProgramPath.Caption:=ExtractFilePath(Application.ExeName);
  GetTempPath(255, @TempDir);
  TempPath.Caption := StrPas(TempDir);
end;
end.
```

KESIMPULAN

Kehadiran produk *processor* yang sangat cepat saat ini telah menimbulkan kebingungan sebagian konsumen yang awam pada saat harus menentukan pilihannya. Apalagi masing-masing vendor menyajikan teknologi baru yang lebih menonjolkan sisi kelebihannya saja. Aplikasi yang dikembangkan dalam penelitian ini dapat membantu untuk menilai dan mengukur kinerja *processor* terutama dari unit FPU (*Floating Point Unit*) yang menentukan kinerja CPU. Dengan demikian aplikasi ini akan memberikan masukan, khususnya bagi para konsumen awam untuk mengetahui kinerja *processor* dan selanjutnya dapat menentukan pilihannya secara tepat.

Terdapat beberapa hal yang masih dapat digali untuk pengembangan penelitian ini, yaitu, penambahan pengujian lain dan detektor otomatis. Pengujian lain yang dapat ditambahkan misalnya, pengujian grafik 3D, kecepatan baca tulis *storage media* (*harddisk*, *CDROM*, dll), akses *RAM* dan banyak lainnya. Sedangkan detektor otomatis merupakan fungsi yang dapat menyempurnakan sifat pendeteksian *hardware*.

DAFTAR PUSTAKA

Ekawati, R., 2002, Pengujian Processor Dengan Metode Iterasi Matematis, Skripsi, IST AKPRIND, Yogyakarta

Majalah CHIP, edisi Januari-Maret 2002

Manual Help SiSoft Sandra, 2001, Standard Version 2001.5.8.11, SiSoftware, http://www.sisoft.co.uk/sandra

Manual Help KinerjanceTest versi 3.5, PassMark Software, www.passmark.com

Pramono, D., 1999, Mudah Menguasai Delphi, Jilid 1 & 2, PT. Elexmedia Komputindo, Jakarta

http://www.anandtech.com update Maret 2002

http://www.sysmark.com update Maret 2002

http://www.tomshardware.com update Maret 2002

http://www.mathworks.com/company/pentium/Smith.txt update Juni 2002

http://www.midnightbeach.com/jon/pubs/Chapters.html update Juni 2002

http://pages.tca.net/roelof/setispy/default.htm#download update Juni 2002

http://www.futuretech.vuurwerk.nl/perf.html update Juni 2002

BIODATA PENULIS

Identitas:

Nama : Edhy Sutanta
Tempat/Tgl Lahir : Kulon Progo, 08 Maret 1972
Minat Khusus : Database, Management Information Systems, Algorithm
Pekerjaan : Staf Pengajar
Kantor : Jurusan Teknik Informatika, Fakultas Teknologi Industri, IST AKPRIND Yogyakarta
Jabatan Akademik: Lektor / IIIC

Riwayat Pendidikan:

1. 1978-1984 : SD Negeri Sentolo, Kulon Progo, Yogykarta
2. 1984-1987 : SMP Negeri I Sentolo, Kulon Progo, Yogykarta
3. 1987-1990 : SMA Negeri I Sentolo, Kulon Progo, Yogykarta
4. 1990-1995 : Jur. Manajemen Informatika & Teknik Komputer, IST AKPRIND, Yogykarta
5. 2004-sekarang : Mahasiswa Pasca Sarjana, Program Studi Ilmu Komputer, UGM, Yogyakarta

Riwayat Pekerjaan:

1. 1996-sekarang : Dosen Tetap Jurusan Teknik Informatika, IST AKPRIND Yogyakarta
2. 1998-2000 : Pelaksana Program Diploma III, Prodi. Manajeman Informatika & Teknik Komputer, IST AKPRIND Yogyakarta
3. 1998-2001 : Sekretaris Jurusan Teknik Informatika, IST AKPRIND Yogyakarta
4. 2000-2004 : Dosen Tidak Tetap Jurusan Teknik Industri, Universitas Sarjanawiyata Tamansiswa, Yogyakarta
5. 2001-2003 : Ketua Jurusan Teknik Informatika IST AKPRIND Yogyakarta
6. 2001-2004 : Redaksi Pelaksana Jurnal Teknologi ACADEMIA ISTA, IST AKPRIND Yogyakarta

Karya Ilmiah:

1. Publikasi melalui Seminar

No	Judul	Waktu
1	Tinjauan tentang metoda penyisipan biner untuk pengurutan data	07-01-1997
2	Efek normalisasi dalam basis data relasional	10-06-1997
3	Pemanfaatan field memo untuk mengurangi redudansi rinci data akibat penerapan teknik normalisasi	31-05-1999

2. Publikasi melalui Jurnal

No	Judul	Nama Jurnal	Waktu
1	Efek normalisasi dalam basis data relasional	ACADEMIA ISTA, vol. 3, No.2, ISSN : 1410-5829, Desember 2000	2000
2	Pemanfaatkan field memo untuk minimalisasi duplikasi rinci data pada kunci penghubung	ACADEMIA ISTA, vol. 5, No.2, ISSN : 1410-5829, Akreditasi No.52/DIKTI/Kep/2002 Desember 2002	2002
3	Mendayagunakan informasi sebagai sumberdaya untuk mencapai keunggulan kompetitif organisasi	DASI vol……, No……. ISSN: 1411-3201 ……..	2004

3. Laporan Penelitian

No	Judul	Waktu
1	Efek Normalisasi Dalam Basis Data Relasional	19-06-1999 s/d 05-10-1999
2	Perancangan Basis Data Model Relasional Pada Entitas Tidak Homogen	10-03-2000 s/d 25-04-2000
3	Perancangan Basis Data Relasional Untuk Model Data Entity	19-02-2001

	Relationship Model (ER_Model)	s/d 5-05-2001
4	Pemanfaatan Field Memo Untuk Minimalisasi Duplikasi Rinci Data Pada Kunci Penghubung	01-02-2002 s/d 30-07-2002
5	Menyigi Penggunaan Metode Penyisipan Biner Dalam Pengurutan Data	25-11-2002 s/d 30-05-2002
6	Menyigi Penggunaan Metode Shell Sort Dalam Pengurutan Data	20-10-2003 s/d 30-05-2004
7	Studi Perbandingan Sistem Perdagangan Di Internet	01-10-2004 s/d 25-02-2005

4. Buku Diterbitkan

No	Judul	Nama Penerbit	Waktu
1	Sistem Basis Data: Konsep & Peranannya Dalam SIM	ANDI Yogyakarta	1996
2	Sistem Informasi Manajemen	Graha Ilmu Yogyakarta	2003
3	Algoritma: Teknik Penyelesaian Permasalahan Untuk Komputasi	Graha Ilmu Yogyakarta	2004
4	Sistem Basis Data	Graha Ilmu Yogyakarta	2004
5	Pengantar Teknik Informatika: Teori dan Aplikasi Komputer Dasar	Graha Ilmu Yogyakarta	2004
6	Komunikasi Data dan Jaringan Komputer	Graha Ilmu Yogyakarta	2004
7	Statistik dan Probabilitas: Teori dan Praktek Komputer	AMUS Yogyakarta	2004