Abstrak

Kriptografi AES-128 siap di-*porting* ke prosesor 8 bit. Algoritma ini dapat langsung diterjemahkan ke bahasa pemrograman C. Akan tetapi, melakukan hal tersebut untuk sebuah program mikrokontroler 8051 mengharuskan kita bergantung pada *compiler* C. Dengan demikian, metoda optimasi yang berfokus pada *constraint* kecepatan diimplementasikan di sini dengan menggunakan bahasa C dan *assembly*. Sejalan dengan *constraint* reduksi siklus mesin, penurunan ukuran program juga dapat diraih.

Generik 8051 adalah sebuah mikrokontroler 8 bit dengan keterbatasan kapasitas program 4 kB dan memori data 128 byte, tersedia dalam aneka varian dan versi *enhancement* dari chip berkemasan hingga IC *smartcard*. Di sini penulis telah mengimplementasikan optimasi hingga mencapai ukuran program total 3407 byte; kecepatan komputasi 3658 *cycle* dan 5648 *cycle*, berturut-turut untuk enkripsi dan dekripsi. *Throughput* 31.6 kbps dapat dicapai untuk kristal osilator 11.059 MHz. Tersedia pula versi *enhancement* 8051 yang dapat menggunakan kristal hingga 40 MHz. Penangkalan *Timing Analysis* juga dilakukan oleh perangkat lunak sebagai bagian dari isu keamanan.

Abstract

AES-128 cryptography is ready for porting to 8 bit processor. The algorithm can be directly translated to C language programming. However, doing so for an 8051 microcontroller program would result in the situation where we have to rely on the C compiler. Hence, methods of optimizations are implemented here using both C and assembly language focusing on speed improvement constraint. Along with the machine cycles reduction constraint, the decrease in code size are also achievable.

The 8051 generic is an 8 bit microcontroller with 4kB code memory and 128 byte data memory limitations, available in enormous number of variants and enhancement versions from packaged chips to smartcard ICs. Here the author has implemented optimization which reaches 3407 bytes of code size; 3658 cycles and 5648 cycles computation speeds, subsequently for encryption and decryption. A throughput of 31. kbps is therefore achievable for 11.059 MHz oscillator crystal. Available as well enhancement version of 8051 which can use up to 40 MHz crystal. Timing analysis countermeasure is also done by the software as part of security issues.

Kata Pengantar

Dengan menyebut nama Allah Yang Maha Pengasih lagi Maha Penyayang, puji dan syukur hanya milik-Nya. Rahmat dan berkah-Nya semata yang membawa penulis menyelesaikan tugas berat ini.

Kesempatan ini akan saya manfaatkan untuk mengucapkan terima kasih pada berbagai pihak yang dengan tulus-ikhlas membantu penyelesaian tugas akhir ini. Secara khusus perlu saya sebutkan di sini:

- Orang tua yang mendukung studi di Bandung sepenuhnya. Ananda telah memilih untuk menjadi orang yang paling berutang kepada Bapak-Ibu. Keluarga yang lain, Bandhin, adik tersayang, *mbah* Madiun, Paman-Bibi, serta sepupu-sepupuku.
- Dr. Ir. Sarwono Sutikno dan Mahmud Galela, S.T. (Laboratorium VLSI) selama bimbingan tugas akhir ini, interaksi sosial sehari-hari, ‘lawan’ diskusi.
- Penguji yang berkenan hadir saat sidang tugas akhir ini: Ir. M. Sarwoko Suraatmadja, MSc., Ir. Achmad Fuad Mas’ud, MSc., dan Ir. Waskita Adjiarto, M.T..
- Dr. Ir. Budi Rahardjo yang bukan sekedar menjadi wali akademik, tetapi juga menyumbangkan banyak paradigma baru buat saya.
- Seluruh dosen di Dept. Teknik Elektro ITB yang telah menjadi ‘guru’ penulis semoga dibalaskan segala limpahan ilmunya.
- Tata usaha departemen: Haji Taopik Hidayat, Bu Ci, Bu Ati, Mas Bayu, dll.; lab elka: Bu Nenden, Pak Nende, dll.
- ‘Keluarga’ saya selama setengah windu di Bandung: Andreas, Bayu, Hardian, Baud (komputer kalian sering saya ‘jajah’ beserta fasilitasnya) serta keluarga Pondok Abah Ale: Yudik, Heru, Nara, Keluarga Pak Asep – Bu Lelly.
- Rekan-rekan VLSI-RG: Arif Masnandar, I. Gayuh, dan Hosni (*class of July 04*); Miftakur, Irsyad, Rifki, Sigit, dan Yani (mahasiswa TA); Ma’muri, Ariya,

dan Irwan (*class of March 04*); Indra Simalango, Andhi H., dan Fikri (admin dan 'calon')

- Elektro 99, rekan BP-HME Indra Poernama, rekan BP-HME W. Ari, rekan kepanitiaan dan *event organizer*, kader dan '*rookie*' kami.
- *Frequency Cut Off* plus: Dimas, Ilham, Willy K., Johan Paul, Fathin, Wisnu, Widya.

Sebagai manusia biasa yang tak luput dari kekurangan, baik dalam pelaksanaan penelitian tugas akhir ini, maupun pelaporannya, layak penulis mohon maaf atas kesalahannya. Saran dan kritik akan menjadi masukan yang berarti buat saya. Semoga tulisan dan karya ini memberikan manfaat bagi masyarakat luas.

Maju Terus Kriptografi Indonesia!

Bandung, 9 Juni 2004,

Penulis

Daftar Isi

Halaman

Daftar Gambar

Halaman

Daftar Tabel

Halaman

Bab 1
Pendahuluan

1.1. Latar Belakang Penelitian

"WinZip® 9.0. *Now with AES Encryption*," begitu bunyi *tag* iklan program kompresi *file* yang sangat populer (lebih dari 115 juta *download*)[1]. Tampaknya, tidak salah bila dikatakan bahwa enkripsi telah menjadi kosa kata yang sangat sehari-hari pula (populer). Bagaimana halnya dengan enkripsi AES?

Rijmen dan Daemen (sebagai non-Amerika) cukup dikejutkan dengan diumumkannya algoritma Rijndael sebagai *Advanced Encryption Standard* (AES) oleh *National Institute of Standards and Technology* (NIST) pada November 2001. Proposal Rijndael menyisihkan empat finalis lainnya yaitu *MARS*, *RC6*, *Serpent*, dan *Twofish*. Sebagai standar baru enkripsi [9], sejak diberlakukan secara efektif (2002), telah lahir sejumlah implementasi AES. Yang mengantongi sertifikat dari NIST saja sudah mencapai 144 produk (per Mei 2004)[2]. AES sendiri memang diperuntukan bagi implementasi *software*, *firmware*, *hardware* atau kombinasinya. Jadi, wajar saja bila usaha pengembangannya banyak dan bervariasi.

Enkripsi sendiri berasal dari kebutuhan akan keamanan (*security*). Dengan melakukan enkripsi, akses terhadap informasi yang dilindunginya menjadi berbatas (*confidential*) karena informasi tersebut telah disandikan[3]. Hanya yang memiliki pengetahuan tentang penyandian tersebutlah yang dapat mengerti isinya. Tinggal kemudian seberapa canggih (*strong*) teknik penyandian dibandingkan dengan usaha pembongkarannya (*attack*) karena apapun dapat terjadi ketika informasi tersebut ditransmisikan. Seberapa 'kuat' informasi harus dijaga

[1] http://www.winzip.com/winzip.htm
[2] http://csrc.nist.gov/cryptval/aes/aesval.html
[3] http://www.itsecurity.com/dictionary/dictionary.htm

keamanannya bergantung pada tingkat kerahasiaannya/aksesibilitasnya (*sensitive/unclassified*, *confidential*, *secret* atau *top secret*) .

Orang-orang bisa latah melafalkan *enkripsi* dan butuh keamanan bukan berarti telah memiliki kesadaran akan keamanan. Oleh karena itu, kita harus realistis juga dalam mengimplementasikan keamanan, informasi senilai Rp 1 juta tidak kita lindungi dengan ongkos Rp 3 juta. Implementasi pada mikrokontroler turut memberikan jawaban atas dilemma tersebut. Sejak dilepas Intel pada 1980, varian dan turunan mikrokontroler 8051 telah hadir hingga ke bentuk *smartcard*. Versi Intel sendiri dapat diperoleh dalam kisaran US$ 4 – US$ 40. Telah 126 juta lebih 8051 (beserta variannya) dikapalkan pada 1993, ia sangat populer dan mudah didapat[4].

Walaupun sejumlah piranti kriptografi beserta data teknisnya menjadi subyek kontrol ekspor Federal AS, terbukanya algoritma AES bagi publik memberikan keuntungan tersendiri. Periset seolah-olah menjadi auditor eksternal bagi algoritma maupun implementasinya untuk masalah keamanan terhadap serangan. Dengan begitu, setiap riset metoda serangan akan ditindaki dengan riset penangkalan (*countermeasure*). Sejauh ini AES masih memiliki keunggulan baik matematis maupun praktis, setidaknya sampai umur nilai informasi yang dilindunginya berakhir.

1.2. Tujuan Penelitian

Secara umum, tujuan dari penelitian ini adalah menghasilkan implementasi algoritma AES-128 pada mikrokontroler keluarga 8051.

1.3. Rumusan Masalah

AES-128 ditujukan untuk dapat diimplementasikan (*porting*) ke prosesor 8 bit. Sebagai mikrokontroler 8 bit, 8051 memiliki sejumlah batasan:

[4] http://www.esacademy.com/automation/faq/8051

- *data memory* internal 128 byte dan *code memory* 4kB.
- Tingkat kesulitan berikutnya adalah bagaimana memperoleh unjuk kerja yang layak sehingga diperoleh modul komputasi AES-128 yang dapat dipakai kembali.

1.4. Batasan Masalah

Pada pelaksanaan tugas akhir ini, implementasi dibatasi pada dihasilkannya modul perangkat lunak yang dapat dimuat mikrokontroler keluarga 8051 untuk komputasi AES-128. Selama pengembangan, sistem mikrokontroler AT89s8252 digunakan untuk menguji modul komputasi AES-128 yang dimuat di dalamnya (emulasi). Sistem mikrokontroler dilengkapi dengan antarmuka komunikasi serial ke PC. Lewat program terminal yang ada di PC, kebenaran fungsional modul komputasi divalidasi. Ukuran unjuk kerja yang digunakan adalah jumlah *cycle* dan panjang *code*. Isu keamanan yang tidak diatur standar ini harus turut menjadi perhatian (prioritas berikutnya) dan diekspresikan pada *code* perangkat lunak yang dihasilkan sebagai bentuk penangkalan (*software countermeasure*).

1.5. Metodologi Penelitian

Untuk memecahkan masalah di atas, langkah-langkah berikut ini dilakukan.

1. Mempelajari standar AES-128 dengan FIPS-197 [9] sebagai dokumen utama.
2. Mempelajari pemrograman untuk set intruksi MCS-51™ dan bahasa C serta *software* pengembangan program mikrokotrolernya.
3. Penelusuran makalah-makalah yang memuat implementasi AES pada mikrokontroler 8-bit untuk memperoleh metoda-metoda penerapan algoritma yang lebih spesifik sekaligus untuk memperoleh pembanding.
4. Membuat terlebih dahulu program mikrokontroler yang akan menjadi dasar/pembangun pengembangan selanjutnya.

5. Melakukan evaluasi sebagai umpan balik pengembangan program mikrokontroler.
6. Mempelajari masalah-masalah keamanan yang mungkin muncul dalam implementasi mikrokontroler 8 bit sebagai prioritas berikutnya.

1.6. Sistematika Pembahasan

Bab 1 Pendahuluan

Pada bab I ini, dijelaskan mengenai latar belakang, tujuan, batasan masalah, dan metoda pelaksanaan penelitian serta sistematika pembahasan laporan.

Bab 2 Dasar Teori Implementasi AES-128

Bab ini merupakan tinjauan pustaka dari standar AES-128 dan isu implementasi yang berkaitan untuk mikrokontroler 8 bit.

Bab 3 Perancangan dan Implementasi Perangkat Lunak

Perancangan dimulai dari deskripsi masalah dan persyaratan pengguna (*user requirements*). Perangkat pengembangan program μC, metoda optimasi, dan interpretasi algoritma dibahas di sini.

Bab 4 Pengujian dan Analisis

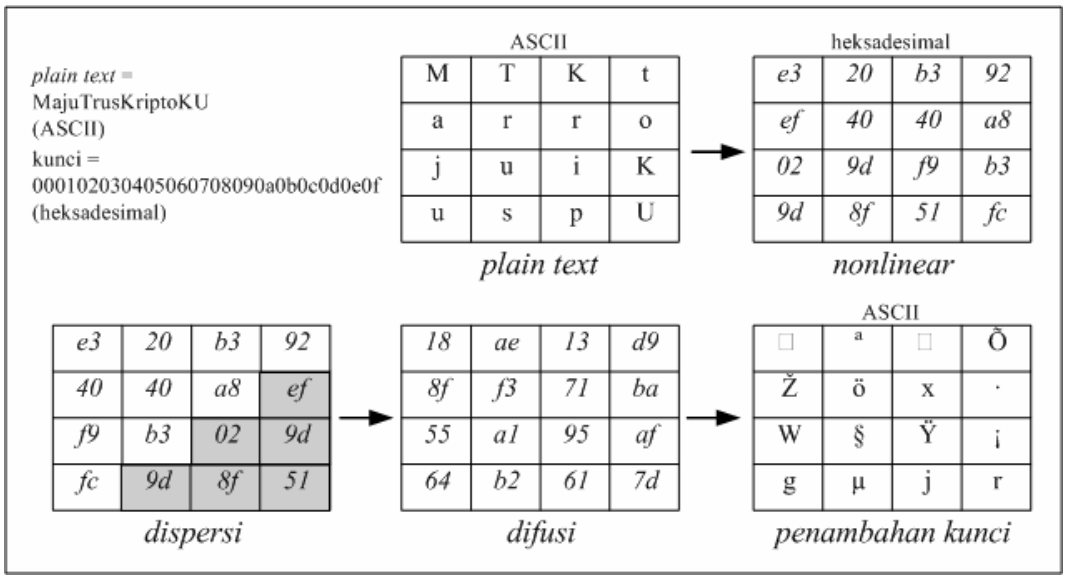
Evaluasi program mikrokontroler yang dihasilkan dibahas di sini. Beserta analisis spesifikasi yang berhasil dicapai.

Bab 5 Simpulan dan Saran

Bab ini berisi simpulan dari implementasi yang dilakukan serta saran untuk pengembangan di masa mendatang.

Bab 2
Dasar Teori Implementasi AES-128

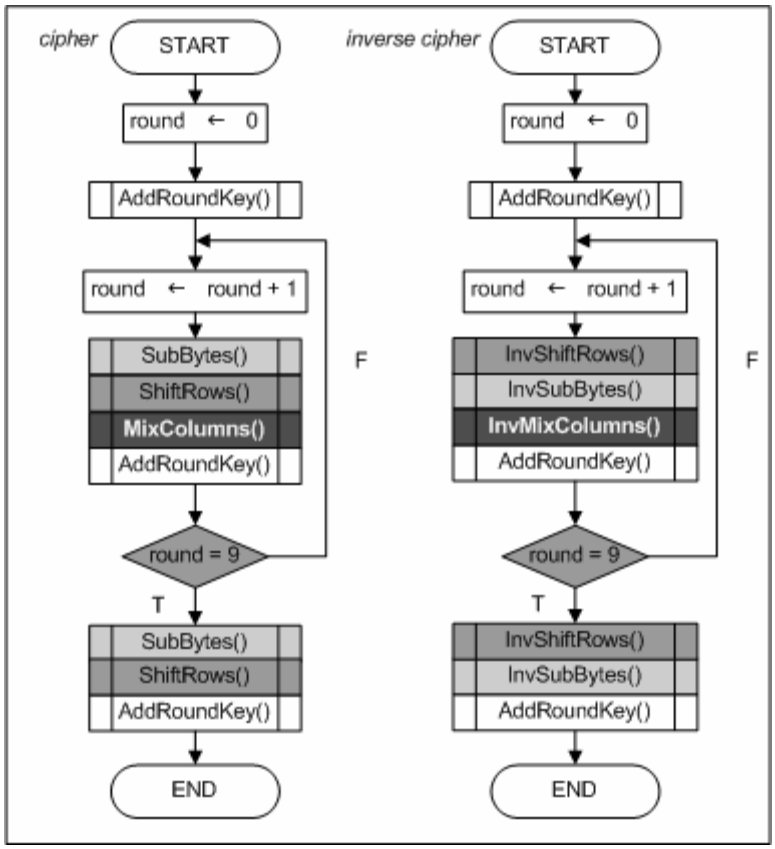
Algoritma Rijndael yang terpilih menjadi AES menyandikan data dalam empat langkah dasar yaitu, langkah **nonlinear**, langkah **dispersi**, langkah **difusi**, dan penambahan **kunci** [5]. Langkah-langkah tersebut dapat dideskripsikan lebih mudah dengan memvisualisasikan data yang akan dikonversi dalam *array* byte segi empat (Gambar 2-1). Nonlinearisasi diperoleh dengan memakai tabel subsitusi nonlinear. Dispersi dilakukan lewat permutasi byte-byte data dari kolom *array* yang berbeda. Langkah difusi menyandikan data menjadi kombinasi linear dari byte-byte data dalam satu kolom *array* tersebut. Penambahan kunci dilakukan dengan operasi XOR antara data dengan kunci. Keempat langkah tersebut akan memiliki nama khusus dalam algoritma yang diterangkan AES (Gambar 2-2).



Gambar 2-1. Langkah Nonlinear, Dispersi, Difusi, dan Penambahan Kunci terhadap Sebuah Teks.

Advanced Encryption Standard (AES) adalah nama standar yang diterbitkan NIST untuk kategori keamanan komputer. Standar algoritma ini oleh penerbitnya ditujukan bagi informasi sensitif (*unclassified*) di lingkungan Federal (AS) yang membutuhkan perlindungan kriptografi [9].

Algoritma AES adalah *cipher* blok simetrik, kunci rahasia yang sama digunakan untuk menyandikan data maupun untuk memperoleh kembali data tersebut dari data tersandinya. Istilah "AES-128" merujuk pada algoritma Rijndael dengan panjang blok (***block***) data dan panjang kunci 128 bit. AES sendiri menggunakan panjang blok data 128 bit, tetapi panjang kunci bisa berbeda-beda (ada AES-128, AES-129, dan AES-256).



Gambar 2-2. Diagram Alir *cipher* dan *inverse cipher*.

Enkripsi memproses masukan 128 bit dan kunci 128 bit. Keluaran tersandi (teks *cipher*) dengan panjang yang sama akan diperoleh melalui ***cipher*** (serangkaian transformasi). Setiap transformasi tersebut memiliki kebalikan, teks tak tersandi (*plain*) dapat diperoleh kembali lewat dekripsi (*inverse cipher*) (Gambar 2-2). Pembahasan berikut akan dimulai dengan sejumlah konvensi dan notasi yang akan digunakan seterusnya.

2.1. Representasi Data

Susunan byte dan bit data diturunkan dari urutan input 128 bit (blok). Bit-bit tersebut dinomeri mulai dari 0 sampai dengan 127 ($0 \le i \le 128$). Setiap urutan 8 bit (**byte**) diperlakukan sebagai entitas tunggal, sebagai elemen *finite field* dengan representasi polinomial

$$b(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 \qquad (2.1)$$

Sebagai contoh {01100011} direpresentasikan oleh polinomial $x^6 + x^5 + x + 1$. Dengan posisi MSB yang konsisten, byte tersebut bernilai heksadesimal {63}. Pengindeksan bit dalam byte dan penomeran rentetan byte dalam blok dapat dilihat lebih jelas dalam Tabel 2-1.

Tabel 2-1. Pengindeksan Aliran Input.

bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
	in_0								in_1								in_2						
indeks	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1
	$s_{0,0}$								$s_{1,0}$								$s_{2,0}$						

Cipher AES dilakukan pada array byte-2 dimensi yang disebut *state*. Blok data disusun dalam ***state*** yang terdiri atas empat baris-*Nb* byte (*Nb* = panjang blok/2 adalah 4 untuk AES-128). Setiap byte diberi dua indeks yang menyatakan

posisinya, dinyatakan sebagai $s_{r,c}$ atau $s[r,c]$, dengan nomer baris r dalam interval $0 \leq r < 4$, sedangkan nomer kolom c dalam $0 \leq c < Nb$. Data dalam *state* menginformasikan hasil setiap tahap transformasi (*intermediate result*).

Input dikopi ke *state array* pada permulaan *cipher* dan *inverse cipher*, kemudian *state* diperbarui pada akhir setiap transformasi (Gambar 2-1). Nilai *state* pada transformasi yang terakhir kemudian dikopi ke output kembali dengan pengindeksan yang serupa Tabel 2-1.

State juga dapat dipandang sebagai ***word*** 4 byte, dengan nomer baris r dari $s_{r,c}$ menyatakan indeks dari keempat byte dalam setiap *word*. Dengan kata lain, *state* ekivalen dengan *array* dari empat *word* yang berindeks c (nomer kolom dari $s_{r,c}$) seperti dinyatakan di bawah ini.

$$w_0 = s_{0,0}s_{1,0}s_{2,0}s_{3,0} \qquad w_2 = s_{0,2}s_{1,2}s_{2,2}s_{3,2}$$
$$w_1 = s_{0,1}s_{1,1}s_{2,1}s_{3,1} \qquad w_3 = s_{0,3}s_{1,3}s_{2,3}s_{3,3}$$

2.2. Operasi Dasar

Walaupun secara keseluruhan hasil antara setiap tahap transformasi melibatkan *state* (1 blok), unit dasar operasi AES adalah byte. Setiap byte sebagai elemen *finite field* dapat dijumlah maupun dikalikan.

2.2.1. Penjumlahan

Penjumlahan dua elemen *finite field* diimplementasikan sebagai operasi XOR per bit. Sebagai konsekuensinya, pengurangan adalah operasi yang identik. Ekspresi berikut ini adalah ekivalen antara satu dengan lainnya (notasi polinomial, biner, dan heksadesimal).

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$$
$$\{01010111\} \oplus \{10000011\} = \{11010100\}$$
$$\{57\} \oplus \{83\} = \{d4\}$$

2.2.2. Perkalian

Perkalian elemen Galois Field (2^8) (notasi ●) dalam representasi polinomial adalah perkalian dengan modulo $m(x) = x^8 + x^4 + x^3 + x + 1$. Bilangan modulo $m(x)$ adalah *irreducible polynomial* GF(2^8).

$$xb(x) = b_8x^8 + b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x \qquad (2.2)$$

Perkalian x ● $b(x)$ dapat diwujudkan sebagai *left shift* yang diikuti XOR kondisional dengan {1b}, jika $b_8 = 1$, maka XOR dilakukan, jika $b_8 = 0$, maka XOR tidak dilakukan. *Exclusive-OR* kondisional tersebut tidak lain adalah operasi modulo dengan $m(x)$. Serangkaian *left shift* yang disusul operasi XOR tersebut dapat digunakan untuk perkalian antar elemen *finite field*. Operasi x ● $b(x)$ dinotasikan sebagai xtime().

2.3. Key Expansion

Key Expansion merupakan rutin untuk menghasilkan ***Round Key***, set kunci yang ditambahkan pada setiap *round*. Dari 4 *word* ***Cipher Key***, ***K***, akan dihasilkan 44 *word* ekspansinya, w_i, dengan $0 \le i < 44$. Empat *word* ekspansi pertama adalah *Cipher Key* itu sendiri.

Setiap *word* berikutnya, w[i], dihasilkan dari operasi XOR *word* sebelumnya, w[i-1], dengan *word Nk* posisi sebelumnya, w[i-*Nk*]. *Nk* sama dengan 4 untuk AES-128. Untuk *i* kelipatan *Nk*, sejumlah transformasi dilakukan pada w[i-*Nk*] sebelum operasi XOR di atas, diikuti oleh XOR dengan *word* konstanta *round*, Rcon.

Transformasi yang pertama adalah **SubWord()**, *word* w[i-4] tersebut dipetakan ke nilai S-Box-nya. Keluaran SubWord() kemudian dipermutasi secara siklik oleh fungsi **RotWord()**, *word* [a_0, a_1, a_2, a_3] akan menjadi [a_1, a_2, a_3, a_0] setelah RotWord(). Konstanta Rcon di atas adalah *word* [{02}$^{i-1}$, {00}, {00}, {00}].

2.4. Enkripsi

Cipher (Gambar 2-2) berlangsung dalam rentetan empat fungsi pembangun (primitif), **SubBytes()**, **ShiftRows()**, **MixColumns()**, dan **AddRoundKey()**. Rentetan tersebut dijalankan sebanyak ***Nr*** - 1 sebagai loop utama (*Nr* = 10 untuk AES-128). Setiap loop disebut ***round***. AddRoundKey() dieksekusi sebagai *round* inisial sebelum loop utama. Setelah loop utama tersebut berakhir (sembilan *round*), SubBytes(), ShiftRows(), MixColumns() , dan AddRoundKey(), dieksekusi secara berturut-turut sebagai *final round*.

2.4.1. AddRoundKey()

Penjumlahan (Bagian 2.3) dilakukan antara *state* dengan Round Key hasil ekspansi (Bagian 2.3). Persamaan berikut ini menjabarkan penjumlahan tersebut.

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round*4+c}] \qquad (2.3)$$

dengan $0 \le c < 4$ (penjumlahan per blok).

2.4.2. SubBytes()

Tabel substitusi byte untuk langkah **nonlinear** (lihat pendahuluan bab ini) tersedia sebagai S-Box [9]. Transfomasi yang telah ditabelkan tersebut mengambil invers multiplikatif GF(2^8) tiap byte, kemudian diikuti dengan transformasi *affine*.

2.4.3. ShiftRows()

ShiftRows() merupakan langkah **permutasi** yang dieksekusi lewat pergeseran siklik tiga baris terakhir *state* (baris pertama, r = 0, tidak digeser). Baris ke dua digeser siklik ke kanan sekali, baris ke tiga dua kali, baris ke empat tiga kali. (Gambar 2-3).

2.4.4. MixColumns()

Difusi diperoleh lewat transformasi MixColumns() yang mengoperasikan *state* kolom-demi-kolom.

$$\left.\begin{aligned} s'_{0,c} &= (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\ s'_{1,c} &= s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c} \\ s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c}) \\ s'_{3,c} &= (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}) \end{aligned}\right\} \quad (2.4)$$

2.5. Dekripsi

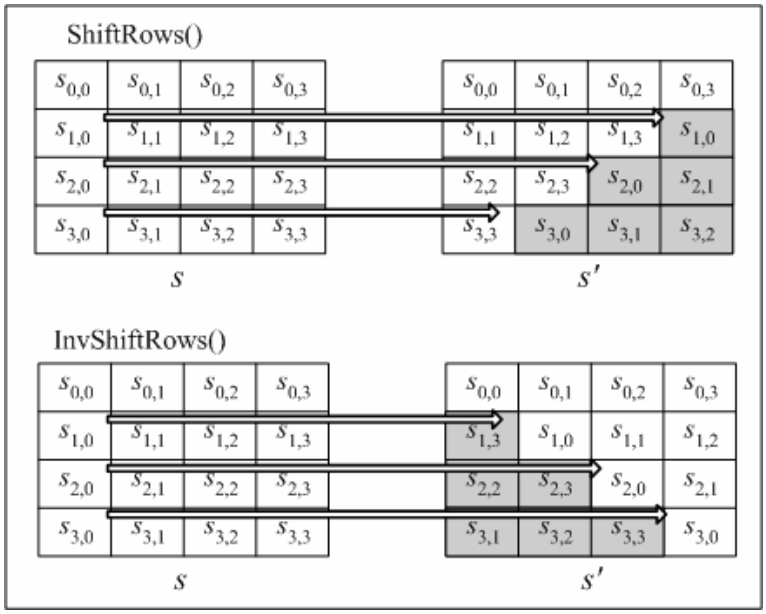
Setiap fungsi enkripsi di atas memiliki kebalikan, dekripsi berlangsung dengan kebalikan dari setiap primitif (*inverse cipher*) (Gambar 2-2). **AddRoundKey()** dieksekusi sebagai *initial round*, diikuti sembilan *round* rentetan **InvShiftRows()**, **InvSubBytes(), InvMixColumns()**, dan AddRoundKey(). *Round* ke-10 yang mengikutinya tidak menyertakan InvMixColumns serupa dengan *final round* enkripsi.

2.5.1. InvSubBytes()

Invers dari tabel S-Box yang digunakan untuk SubBytes tersedia sebagai S-Box^{-1} [9].

2.5.2. InvShiftRows()

Kebalikan ShiftRows() ini (Bagian 2.4.3) berlangsung dengan menggeser siklik ke arah berlawanan. Baris ke dua digeser siklik ke kiri sekali, baris ke tiga dua kali, baris ke empat tiga kali (Gambar 2-3).



Gambar 2-3. Tranformasi ShiftRows() dan InvShiftRows().

2.5.3. InvMixColumns()

Operasi *state* per kolom yang diwujudkan MixColumns() (Bagian 2.4.4) memiliki kebalikan berupa persamaan berikut ini.

$$\left.\begin{aligned} s'_{0,c} &= (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c}) \\ s'_{1,c} &= (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c}) \\ s'_{2,c} &= (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c}) \\ s'_{3,c} &= (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c}) \end{aligned}\right\} \quad (2.5)$$

2.6. Isu Implementasi

Sebagai catatan, dokumen AES menyebutkan batasan yang cukup luas perihal implementasi, yaitu dibolehkannya variasi dari standar. Implementasi apapun dengan hubungan keluaran-masukan (teks *cipher* maupun *plain*) yang sesuai spesifikasi standar ini, dapat diterima sebagai implementasi AES.

Tidak ada persyaratan khusus untuk pemilihan kunci yang dapat digunakan. Restriksi tidak ada karena AES tidak mengenal istilah *weak key* maupun *semiweak key*.

AES-128 merupakan standar yang menerangkan algoritma, aspek lain yang terkait dengan implementasi kriptografi tidak dibahas di sini. Secara terpisah NIST [14] mempersyaratkan sejumlah tes[5] yang harus dipenuhi jika hendak diakui sebagai implementasi yang memenuhi AES. Tes akan dilakukan oleh lab CMT (*Cryptographic Module Testing*) yang berakreditasi. Untuk masalah keamanan, terdapat standar yang memuat *security requirement*[6] bagi modul kriptografi masih dari NIST. Mode operasi yang digunakan untuk menangani aliran blok juga direkomendasikan dalam publikasi NIST tersendiri[7].

2.6.1. Kekuatan terhadap Serangan

Kekuatan matematis algoritma menjadi bahasan *cryptanalysis* tersendiri ketika proposal Rijndael untuk AES diajukan [6]. Bagi perancang implementasi, serangan spesifik terhadap implementasi perlu pula mendapat tempat khusus. Informasi rahasia yang dilindungi perangkat kriptografi dapat bocor lewat serangan *side channel*. Beberapa di antaranya yang perlu diberi perhatian adalah *timing analysis* dan *power analysis* [5,7]. Serangan *side channel* memperlakukan modul kriptografi sebagai *black box* yang hanya dapat diukur sinyal-sinyal eksternalnya: durasi eksekusi, konsumsi daya, radiasi elektromagnetik, temperatur, dll. Sukses serangan *side channel* membutuhkan gabungan pengetahuan dari beberapa bidang: statistik, pemrograman, intrumentasi, dan algoritma kriptografi itu sendiri [7].

Sejumlah penangkalan (*countermeasure*) dapat dilakukan terhadap kemungkinan serangan. Walaupun kebocoran side channel bersifat fisik, *software countermeasure* juga harus dilakukan berbarengan dengan usaha-usaha fisik.

---

[5] "AESAVS (The Advanced Encryption Algorithm Validation Suite)," 2002.
[6] "FIPS 140-2 : Security Requirements for Cryptographic Modules," 2001.
[7] "Recommendation for Block Cipher Modes of Operation," 2001.

2.6.1.1. Serangan Timing Analysis

Korban-korban awal dari *timing analysis attack* adalah kartu *TV-pay* yang 'dibongkar'dengan program *Sigpro* dan *Timeit*. Serangan dilakukan dengan membandingkan **konsumsi waktu** yang dibutuhkan kartu untuk memberikan jawaban autentifikasi atas sejumlah masukan tebakan [8]. *Timing analysis* memanfaatkan kebergantungan durasi komputasi terhadap data masukan.

2.6.2. Perbandingan Implementasi AES-128 pada Prosesor 8 Bit

Publikasi implementasi AES-128 untuk kelas prosesor 8 bit menampilkan *cycles* dan *code length* sebagai ukuran unjuk kerja. Di antaranya adalah pada Intel 8051, Motorola 68HC08, Atmel AVR dan PIC [5,10,13]. Keempatnya diimplementasikan dalam bahasa *assembly*. Yang diimplementasikan hanya enkripsi.

Tabel 2-2. Implementasi AES-128 pada Prosesor 8 bit.

implementasi AES-128	*cycle*	*byte*
Intel 8051 a)	4065	768
Intel 8051 b)	3744	826
Intel 8051 c)	3168	1016
68HC08	18390	919
Atmel AVR a)	6658	866
Atmel AVR b)	3815	504
PIC	3969	

sumber: V. Rijmen [13]; K. Sung Ha [13]; R. Ward [10].

Implementasi pada 8051 dan 68HC08 yang dibuat Rijmen *et al.* ditujukan bagi aplikasi *smartcard* berprosesor. Untuk yang bertipe keluarga 8051, optimasi penggunaan ROM dilakukan sehingga dapat diperbandingkan tiga panjang *cycle* berbeda untuk tiga macam panjang *code*. Pada implementasi prosesor 68HC08, optimasi panjang *code* tidak dilakukan. Pembuat implementasi Atmel AVR

melakukan pembandingan antara dua cara alokasi *state*. Hasilnya, proses *state* dengan register lebih cepat daripada dengan memori. Akan tetapi, total *cycle-code size* implementasi AVR tersebut merupakan netto sepuluh *round* enkripsi, ekspansi kunci dan input tidak diperhitungkan.

2.6.3. Mikrokontroler 8051

Nama generik 8051 diberikan untuk IC mikrokontroler keluarga MCS-51™ Intel [11]. Lisensi manufaktur 8051 juga dimiliki oleh vendor lain di antaranya Philips, Siemens, Advanced Micro Devices, dan Fujitsu. *Library* generik 8051 untuk semua varian sudah tersedia pada *compiler-linker* Keil yang digunakan untuk pengembangan program µC (Bagian 3.3). Pada *library* Keil masih terdapat tiga generik MCS-51™ lainnya, 8031, 8032, dan 8052. Perbedaan di antaranya terangkum dalam Tabel 2-3. Varian maupun versi *enhancement* 8051 biasanya masih menyertakan angka 51 (dari 8051) pada *part number*-nya.

Program yang dibuat dengan set intruksi MCS-51™ bisa dipakai untuk semua varian selama tidak mengeksploitasi fitur-fitur spesifiknya. Fitur tersebut dikontrol lewat *Special Function Register* (SFR) tambahan yang tidak ada pada generik 8051 [12]. Instruksi program yang dependen terhadap jenis 8051 yang dipakai akan berupa *storing* nilai ke SFR tambahan tersebut. Sebagai contoh, SAB80C517 (Siemens) memiliki delapan `DPTR`[8] yang dapat dipilih melalui `DPSEL` (alamat {92})[9]; 83C751 (Philips) menggunakan tiga SFR: `I2CON`, `I2DAT`, dan `I2CFG` untuk mendukung komunikasi *Inter-Integrated Circuit* (I^2C); seri 89s (Atmel) memiliki *watchdog timer* yang diatur melalui SFR `WMCON`.

Jika hanya untuk komputasi AES, perbedaan-perbedaan di atas tidak akan dieksploitasi sebagaimana halnya implementasi pembanding yang telah dipublikasikan (Bagian 2.6.2). Ia baru akan dilihat ketika komputasi harus berhubungan dengan dunia luar melalui periferal atau jalur I/O yang khusus.

[8] Data pointer untuk instruksi-instruksi mengambil dari atau menyimpan ke memori.
[9] Alamat {92} untuk SFR ini tidak didefinisikan pada generik 8051. Pada 8051 hanya ada 21 SFR.

Kelebihan versi *enhancement* yang dapat dimanfaatkan adalah berlipatgandanya kecepatan, seri 89 Atmel bisa bekerja dengan kristal 24 MHz, seri C50x (Siemens) bisa hingga 40 MHz.

Tabel 2-3. Varian Keluarga MCS-51™ Intel.

part number	on-chip code memory	on-chip data memory	timers/ counters	interrupts
8051	4kB ROM	128 bytes	2	5/2 level
8031	0	128 bytes	2	5/2 level
8751	4kB EPROM	128 bytes	2	5/2 level
8052	8kB ROM	256 bytes	3	6/4 level
8032	0k	256 bytes	3	6/4 level
8752	8kB EPROM	256 bytes	3	6/4 level

sumber: *Library* Keil; Scott MacKenzie [11].

Enhancement 8051 juga hadir sebagai CPU *smartcard* yang memiliki antarmuka ISO7816. Di antaranya adalah Theseus (EM Microelectronics)[10], keluarga WE (Philips)[11], dan SLE seri 44s/22s (Infineon)[12]. Di antara kelebihan yang ditawarkan misalnya saja, seri Theseus EMGTG96-3G yang memiliki 96kB *flash memory*; versi *smartcard* yang lebih mutakhir yang dilengkapi pula dengan modul kriptografi (*co-processor*) seperti 3-DES[13] (sehingga prosesor utama dibebani tugas lain). Adanya variasi pilihan tersebut adalah untuk memenuhi *requirements* yang berbeda-beda. Infineon misalnya, membagi jangkauan produknya dalam kelas *low end* hingga *high end security* untuk aneka kebutuhan: GSM, perbankan, *TV-pay*, dll.

[10] http://www.emmarin.com/
[11] http://www.semiconductors.philips.com/markets/identification/products/we/
[12] http://www.infineon.com/
[13] "FIPS46-3: Data Encryption Standard," http://csrc.nist.gov/cryptval/des.htm

Bab 3
Perancangan dan Implementasi Perangkat Lunak

3.1. Deskripsi Masalah

Set instruksi yang diadopsi program μC adalah dari MCS-51™ dan bahasa C untuk *compiler* Keil. Dengan mikrokontroler AT89s8252 yang digunakan untuk pengujian, kapasitas memori untuk program μC berganda dua kali lipat, yaitu 256 byte *on-chip data memory*. Delapan kB *code memory* yang tersedia juga dua kali lipat generik 8051 (Bagian 2.6.3). *Compiler* yang ada memungkinkan ditulisnya perangkat lunak μC dalam dua lingkungan sekaligus, bahasa C dan bahasa *assembly*. File *.hex maupun listing *assembly* dapat dihasilkan kemudian. *Source code* dibuat modular sehingga akan dihasilkan bagian komputasi enkripsi-dekripsi yang terpisah dari bagian I/O-nya. Dengan demikian, siklus instruksi dan ukuran program komputasi dapat dievaluasi per fungsi secara terpisah.

AES-128 [9] telah menyediakan panduan implementasi dan contoh vektor uji. Vektor uji AESAVS akan digunakan untuk emulasi dengan sistem mikrokontroler AT89s8252. Selama pengembangan dan untuk kemudahan emulasi tersebut, PC digunakan sebagai antarmuka dengan pengguna. Isu keamanan terhadap serangan-spesifik mikrokontroler menjadi prioritas berikutnya setelah komputasi memperoleh kecepatan dan ukuran yang layak.

3.2. Persyaratan Pengguna

- Mikrokontroler mampu melakukan komputasi enkripsi/dekripsi data berdasarkan masukan kuncinya, sesuai AES-128. Namun, variasi dari AES-128 diperbolehkan. Jadi, notasi, persamaan, dan transformasi boleh diubah asalkan masukan/keluaran sesuai spesifikasi standar.

- Modul enkripsi/dekripsi yang dihasilkan dapat dipakai kembali (*reusable*) pada sistem-sistem lain yang berbasis mikrokontroler 8051. Jika dipakai kembali, adaptasi cukup dilakukan pada modul I/O dan kontrol dari program μC.
- Siklus mesin dan ukuran program dievaluasi. Implementasi lain yang dipublikasikan digunakan sebagai pembanding.
- Sistem mikrokontroler mendukung komunikasi serial dengan PC selama pengembangan dan pengujian.
- Enkripsi/dekripsi divalidasi dengan vektor uji yang tersedia. Kombinasi vektor uji tersebut dimasukkan lewat program terminal di PC yang mengirimkannya secara serial ke sistem. Keluaran enkripsi/dekripsi dikirim kembali ke PC pada program terminal yang sama.
- Port-port μC digunakan sebagai indikator status maupun kontrol pilihan mode (enkripsi/dekripsi).

3.3. Perangkat Pengembangan Program Mikrokontroler

Perangkat yang digunakkan untuk pengembangan program adalah μVision 2 V2.23 (Keil Software Inc.) [15]. Perangkat lunak ini sudah memuat program *C compiler* dan *assembler* untuk set instruksi MCS-51™. *Compiling* dan *assembling* akan menghasilkan file obyek dari masing-masing modul *source code*. *Linker/locator* akan menggabungkan obyek masing-masing modul tersebut. Kemudian, *object hex converter* menghasilkan file *.hex yang dapat di-*download* ke mikrokontroler target.

Library Keil untuk 8051 tersedia dari berbagai varian dan vendor, misalnya seri-seri Intel sendiri, seri 89C dan 89S Atmel, seri C500 Infineon, Phillips, dll. Terdapat pula *library* generik untuk semua varian 8051. Perincian target *library* generik tersebut adalah:

- mikrokontroler *8051-based* (CMOS atau NMOS)
- 32 jalur I/O, *2 timers*, *5 interrupts/2 priority levels*
- 4 kB ROM, 128 *bytes on-chip* RAM

Juga tersedia generik varian lainnya yaitu 8031, 8032, dan 8052 (Tabel 2-3).

Fasilitas yang sangat berguna adalah perangkat simulasi di lingkungan *debugger*. Dalam lingkungan ini status register saat simulasi dapat dimonitor pada *project window*. Perubahan isi variabel saat eksekusi baris-demi-baris program bisa dijejak dengan menambahkan *watch*. Selama simulasi, eksekusi baris-baris program tersebut dapat dikontrol langkahnya (*go*, *step into*, *step out*, dll.). Cara kontrol lain adalah dengan menempatkan sejumlah titik berhenti (*break*) pada baris-baris program. Tertera pula siklus mesin berjalan dan sekon ekivalennya.

Debugger juga menyediakan perangkat evaluasi. Yang dapat dimanfaatkan selama simulasi adalah *Performance Analyzer (PA)* dan monitor *code coverage*. Hasil evaluasi tersebut dapat dilaporkan sebagai teks log maupun langsung ditampilkan di tengah-tengah jalannya simulasi. *PA window* memberikan informasi visual yang sangat berguna dengan menampilkan grafik batang waktu eksekusi semua fungsi yang dipantau *on the fly*. Dengan begitu, secara visual dapat langsung ditentukan fungsi-fungsi mana saja yang mendominasi siklus mesin keseluruhan program.

Simulasi I/O serial dapat diotomasi [15] dengan terlebih dahulu membuat aliran input serialnya. Data yang akan menjadi masukan serial dapat dibuat dalam file tersendiri. Tombol *custom* yang mengontrolnya dapat ditambahkan pada koleksi *toolbox button*.

Penulisan program dapat menyertakan beberapa direktif kompilasi (*pragma*). Ekstensi bahasa (dari ANSI C) dan *library* yang mendukung arsitektur 8051 juga tersedia. Namun, beberapa rutin *library* standar ANSI C tidak disertakan, misalnya *fopen* dan *fprintf*.

3.4. Metoda Optimasi Program

Optimasi program dilakukan untuk meraih kecepatan komputasi. Ukuran program menjadi pertimbangan sekunder. Pengubahan dilakukan pada fungsi-fungsi yang dominan terhadap waktu eksekusi keseluruhan. Untuk

mengetahuinya, dibutuhkan umpan balik dari kegiatan simulasi, yaitu evaluasi PA (Bagian 4.2).

Dua pendekatan optimasi yang pertama terkait dengan gaya menulis program (*coding*) dan *compiler C* yang digunakan. Pendekatan yang ketiga terkait dengan metoda penerapan algoritma. Dalam prakteknya teknik-teknik optimasi tersebut tidak berdiri sendiri-sendiri.

3.4.1. Penulisan dalam Campuran C – Assembly

Satu proyek Keil dapat memuat sejumlah *source code* untuk target yang sama. Grup *source code* tersebut dapat terdiri atas (campuran) file-file berbahasa C maupun file-file *assembly*.

Pendekatan optimasi yang pertama adalah dengan mengubah penulisan C ke bahasa *assembly*. Pendekatan ini dilakukan jika ternyata listing *assembly* yang dihasilkan *compiler* terlihat boros instruksi. Selama hasilnya sama, baris-baris *assembly* yang tidak perlu dapat dibuang. Optimasi tersebut sangat mengandalkan pemrogram (subyektif). Baru dapat diketahui efisien tidaknya setelah diperbandingkan secara kuantitatif dengan pencapaian implementasi milik orang lain (obyektif).

Listing *assembly* yang dihasilkan *compiler* dapat digunakan sebagai acuan penulisan ulang fungsi yang hendak dioptimasi. Harga yang harus dibayar untuk instruksi-instruksi yang dapat dipakai tertera dalam Tabel 3-1 berikut ini (hanya sebagian).

Tabel 3-1. Harga Sejumlah Instruksi MCS-51™.

instruksi	*cycle*	*byte*
`MOV A,Rn`	1	1
`MOV A,direct`	1	2
`MOV A,@Ri`	1	1
`MOV A,#data`	1	2
`MOV Rn,A`	1	1

`MOV Rn,#data`	1	2
`MOV direct,A`	1	2
`MOV @Ri,A`	1	1
`MOV @Ri,#data`	1	1
`RRC A`	1	1
`RLC A`	1	1
`MOV Rn,direct`	2	2
`MOV direct,Rn`	2	2
`MOV direct,direct`	2	3
`MOV direct,@Ri`	2	2
`MOV direct,#data`	2	3
`MOV @Ri,direct`	2	2
`MOVX` (move external)	2	1

Secara umum, set instruksi yang '**murah**' (Tabel 3-1) untuk memindahkan variabel byte adalah set instruksi berikut ini.

```
MOV A,<byte-asal>
MOV <byte-tujuan>,A
```

Harga instruksi di atas hanya satu siklus mesin. Akan tetapi, optimasi tidak bisa dilakukan hanya dengan memperhatikan baris-baris instruksi secara individual. Contoh berikut ini mengilustrasikan hal tersebut. Instruksi di bawah ini mengekspresikan perpindahan variabel yang sama dengan dua cara berbeda. Dengan memakai akumulator menjadi

```
MOV A,R7
MOV 08h,A
```

tanpa akumulator menjadi

```
MOV 08h,R7
```

Versi pertama (dua baris) memindahkan isi R7 ke alamat {08} dalam dua *cycle* dan panjang *code* tiga byte, sedangkan versi kedua (satu baris) melakukannya dalam dua *cycle* dan panjang *code* dua byte. Dalam konteks panjang *cycle*, keduanya setara, tetapi dalam konteks ukuran, versi kedua lebih kecil.

Listing *assembly* yang dibangkitkan *compiler* sangat bergantung kepada **gaya penulisan** program C-nya. Sebagai ilustrasi (potongan program dari fungsi `InvMixColumns()`), kedua set instruksi di bawah ini menghasilkan operasi yang

sama tetapi *assembly* yang dibangkitkan kompilasinya berbeda. Yang pertama adalah

```
state[0][0] = mul_D[x0_2] ^ mul_9[x1_3] ^ state[0][0];
state[0][1] = mul_D[x1_3] ^ mul_9[x0_2] ^ state[0][1];
```

Instruksi kedua yang ekivalen adalah

```
state[0][0] = mul_D[x0_2] ^ state[0][0];
state[0][1] = mul_D[x1_3] ^ state[0][1];
state[0][0] = mul_9[x1_3] ^ state[0][0];
state[0][1] = mul_9[x0_2] ^ state[0][1];
```

Set intruksi yang kedua (dalam empat baris bahasa C) telah memiliki '**kemiripan**' dengan *mnemonic*-nya. Dengan menempatkan dua operasi memanggil array (dari tabel) `mul_D[]` secara berurutan maka inisiasi *pointer* array cukup dilakukan sekali untuk dua operasi sekaligus, mirip dengan 'cara berpikir' *assembly*-nya. Berikut ini adalah *assembly* yang dibangkitkan *compiler* untuk dua baris pertama yang memakai array `mul_D[]`.

```
MOV A,R7            ;isi x0_2 di-store ke akumulator
MOV DPTR,#mul_D     ;inisiasi pointer ke mul_D
MOVC A,@A+DPTR      ;look up table mul_D[]
XRL state,A         ;state[0][0] <- mul_D[x0_2] ^ state[0][0]
MOV A,R4            ;isi x1_3 di-store ke akumulator
MOVC A,@A+DPTR      ;masih look up table mul_D[]
XRL state+01H,A     ;state[0][1] <- mul_D[x1_3] ^ state[0][1]
```

Setelah itu, baru *pointer* (`DPTR`) diinisiasi menunjuk ke `mul_9[]` dengan *assembly* berikut ini.

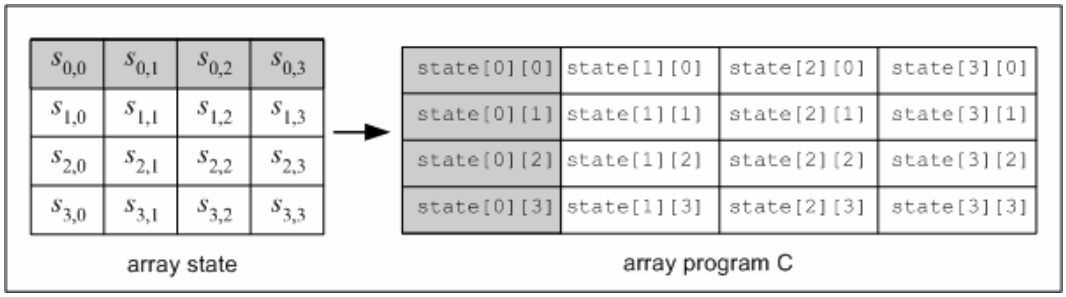
```
MOV A,R4            ;isi x1_3 di-store ke akumulator
MOV DPTR,#mul_9     ;inisiasi pointer ke mul_9
MOVC A,@A+DPTR      ;look up table mul_9[]
XRL state,A         ;state[0][0] <- mul_9[x0_2] ^ state[0][0]
MOV A,R7            ;isi x0_3 di-store ke akumulator
MOVC A,@A+DPTR      ;masih look up table mul_9[]
XRL state+01H,A     ;state[0][1] <- mul_9[x1_3] ^ state[0][1]
```

Versi empat baris sintaks C ini menghasilkan baris-baris *assembly* yang lebih sedikit daripada versi sintaks C dua baris sebelumya. Otomatis tidak perlu ada

optimasi kecepatan lagi (penulisan ulang fungsi bahasa C dalam *assembly* yang lebih singkat dan murah *cycle*-nya).

3.4.2. Alokasi Variabel dan Konstanta

Konvensi pengindeksan array $s_{\text{baris,kolom}}$ dalam AES-128 dipetakan ke dalam array `state[kolom][baris]` dalam program C (Gambar 3-1). Pengindeksan **dibalik** setelah menganalisis *assembly* hasil kompilasi. Sebagai contoh, `state[0][0]` setelah kompilasi akan diberi label[14] `state`, `state[0][3]` akan dilabeli dengan `state+03h`, dst. Jadi, jika pointer sedang menyimpan alamat awal array (`state`), maka *incremental* pointer akan menunjuk pada label `state+01h`, `state+02h`, dst. Jika digambarkan sebagai array dua dimensi, kenaikan tersebut adalah kenaikan indeks dalam arah baris pada kolom yang sama (Gambar 3-2).



Gambar 3-1. Pengindeksan Array State dalam Program.

Array *state* dideklarasikan (dalam bahasa C) sebagai array dua dimensi yang menempati alamat absolut {08} s.d. {17}. Jadi, label `state` memiliki alamat absolut {08}. Susunan seperti ini menempatkan setengah blok *state* dalam bank register 1 dan setengahnya lagi dalam bank register 2 (Gambar 3-2). Setiap byte

[14] *Label* dalam program *assembly* menyatakan alamat dari baris-baris program atau data yang mengikutinya.

elemen *state* dapat diakses dengan alamat *direct*-nya ({08} – {17}) maupun sebagai register (R7 – R0). Alamat-alamat tersebut dikenali sebagai register jika bank register yang bersangkutan sedang aktif. Instruksi berikut memindahkan isi A ke $s_{0,0}$ dengan alamat langsungnya.

```
MOV 08h,A
```

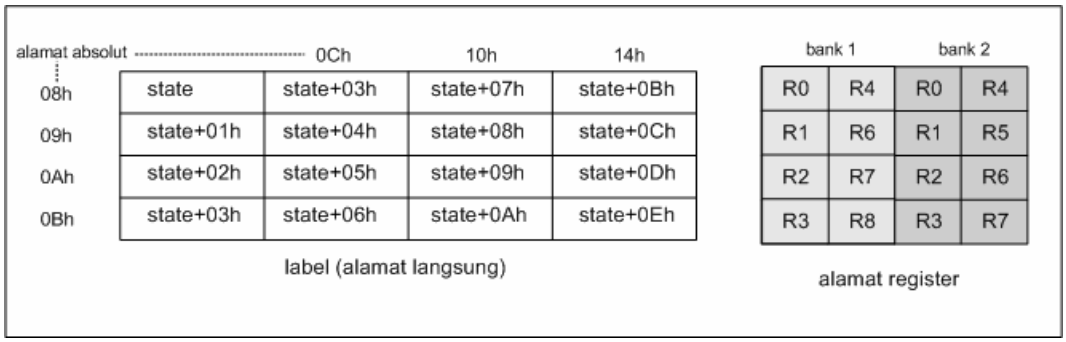
atau dengan menggunakan label seperti

```
MOV state,A
```

Dua cycle dan dua byte diperlukan oleh instruksi di atas. Jika bank register 1 aktif (lewat inisialisasi SFR `PSW`[15]), maka ekspresi berikut adalah ekivalen.

```
MOV R7,A
```

Satu cycle dan satu byte diperlukan oleh instruksi tersebut.



Gambar 3-2. Pengalamatan State yang Bisa Dipakai.

Bank register 0 adalah bank register *default* yang aktif setelah reset, komputasi akan memanfaatkan register-register ini. Itulah alasan digunakannya bank register 1 dan 2.

Penggunaan RAM internal yang lain dideklarasikan sebagai *relocatable data segment* (bukan alamat absolut seperti `state`).

[15] Pada Program Status Word, seleksi bank register yang aktif dikontrol lewat bit PSW.4 dan PSW.3.

Konstanta yang digunakan sebagai *look up table* dialokasikan pada memori program (*on chip-code memory*). Ukuran program keseluruhan akan menyatakan ukuran instruksi ditambah dengan tabel. Perhitungan yang ditabelkan, misalnya xtime() (Bagian 2.2.2), dapat dilakukan lebih cepat. Hasil perhitungan sudah berada di akumulator dalam lima *cycle*. Itupun sudah termasuk inisialisasi alamat awal *look up table*. Jadi, jika tabel yang sama masih digunakan oleh baris program berikutnya, kecepatan perhitungan menjadi tiga *cycle*. Harga yang harus dibayar dari penabelan konstanta adalah pemakaian memori program.

3.4.3. Metoda Implementasi Algoritma

Pendekatan ketiga dalam usaha meraih kecepatan komputasi adalah dengan mengatur kembali persamaan yang dipakai untuk memperoleh bentuk yang lebih sederhana jika dilihat dari sintaks program yang akan dipakai. Pendekatan spesifik implementasi ini dimulai dari tiap fungsi pembangun program. Dengan demikian, optimasi per primitif akan menghasilkan pelipatgandaan kecepatan sesuai pembobotannya (Tabel 3-2). Jadi, algoritma tidak diterjemahkan begitu saja ke bahasa C.

Tabel 3-2. Pembobotan *Cycle* Primitif AES-128.

primitif		bobot
enkripsi	dekripsi	
AddRoundKey()	AddRoundKey()	11
ShiftRows()	InvShiftRows()	10
SubBytes()	InvSubBytes()	10
MixColumns()	InvMixColumns()	9

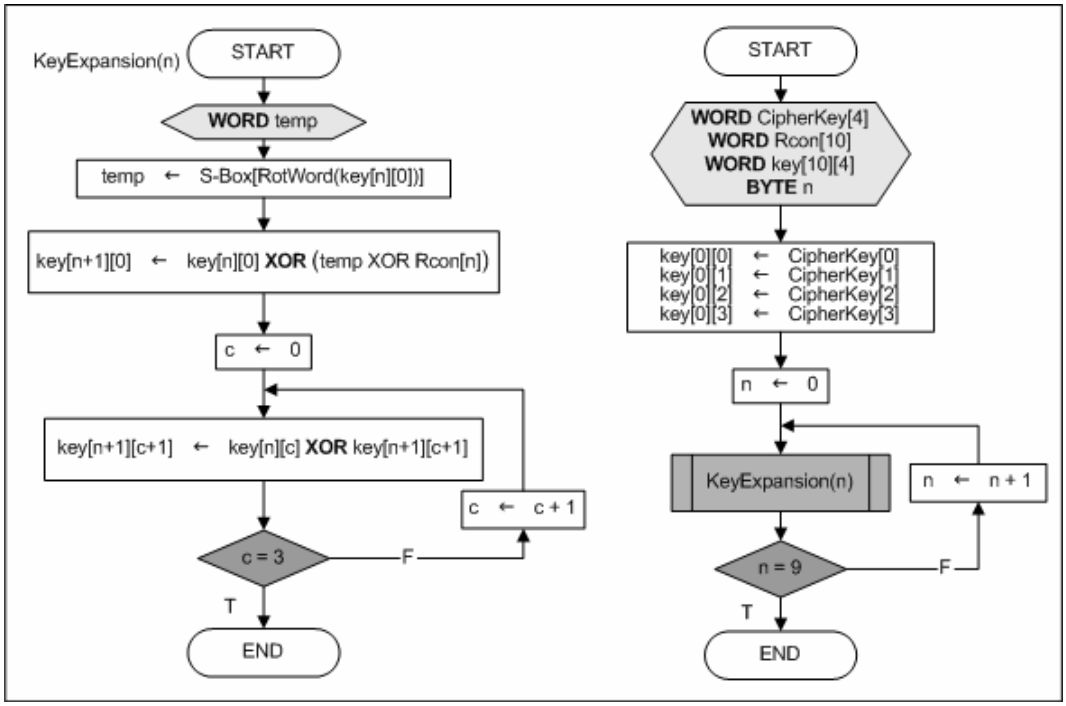
3.5. Key Expansion

Memori internal 8051 generik (Tabel 2-3) tidak dapat memuat 44 *word* key (176 byte) hasil ekspansi sekaligus. Rutin ini harus berbagi RAM dengan rutin

lain yang membutuhkan memori temporer juga. Penggunaan memori eksternal, selain memakan *cycle* panjang (Tabel 3-1), akan mengurangi keamanan. Yang dimaksud adalah munculnya data-data ekspansi kunci di bus data eksternal (port mikrokontroler). Padahal, **kerahasiaan** berada di kunci, algoritma dan mode operasi dipublikasikan terbuka.

Untuk mengatasi masalah ini, kunci disimpan dalam buffer secara **siklik**. *Word* kunci yang sudah tidak digunakan dapat dibuang dari memori. Proses menghasilkan *word* ekspansi hanya membutuhkan pengetahuan hingga empat *word* kunci sebelumnya. Dengan begitu, besar buffer minimal lima *word* (empat *word* sebelumnya ditambah satu *word* hasil ekspansi). Pengaruhnya adalah AddRoundKey() dilakukan per *word*, setiap 1 *word* usai, *word* berikutnya harus menunggu ekspansi kunci berikutnya selesai. Pilihan lain adalah menggunakan buffer yang lebih besar.

Dengan buffer yang lebih besar (di atas lima *word*), siklus buffer dapat dibuat tidak per satu word seperti di atas, tetapi per empat *word*. Fungsi ekspansi kunci dapat ditulis dengan menggunakan alamat absolut maupun dengan pointer. Cara kedua (dengan pointer) memungkinkan ukuran buffer di atas delapan *word*. Diagram alir (Gambar 3-3) berikut mendeskripsikan ekspansi kunci.



Gambar 3-3. Diagram Alir Ekspansi Kunci.

Untuk AddRoundKey() dekripsi, ekspansi kunci dijalankan hingga diperoleh Round Key yang terakhir. Dengan pengetahuan empat *word* terakhir itu, kunci dihasilkan secara mundur pada setiap *round* dekripsi. Fungsi yang menghasilkannya merupakan kebalikan ekspansi kunci enkripsi. Fungsi ini perlu dibuat mengingat tidak keseluruhan 44 *word* kunci disimpan. Jika 44 *word* tersebut disimpan, tentunya tinggal 'dipanggil' saja secara terbalik.

3.6. Enkripsi

Fungsi enkripsi secara keseluruhan (10 *round*) baru dibuat setelah fungsi-fungsi pembangunnya selesai dibuat dan diuji kebenaran fungsionalnya. Enkripsi dibangun lebih dahulu daripada dekripsi. Tujuannya adalah untuk mencapai hasil implementasi yang layak. Kelayakan tersebut diukur dengan perbandingan terhadap publikasi implementasi enkripsi pada μC 8 bit (Bagian 2.6.2).

3.6.1. AddRoundKey()

Jika buffer ekspansi kunci yang digunakan hanya lima word (Bagian 3.5), otomatis AddRoundKey() diwujudkan sebagai fungsi yang menambahkan kunci per kolom *state*. Jika hasil ekspansi kunci dapat diketahui empat word sekaligus, maka AddRoundKey() dibuat seperti dispesifikasikan algoritma (Bagian 2.4.1).

3.6.2. SubBytes()

Pilihan implementasi untuk transformasi SubBytes() jatuh pada tabel S-Box. Ke-256 byte elemen tabel dimasukkan sebagai kontanta dalam *code memory*.

```
code unsigned char S_Box[256] = { 99, 124, ...<dihapus>
```

Dalam sintaks C, S-Box dideklarasikan sebagai array *unsigned char* berukuran 256x1. Setiap *state* dipetakan S-Box menggunakan alamat langsungnya, misalnya

```
state[0][0]=S_Box[state[0][0]];
```

3.6.3. ShiftRows()

Pergeseran dilakukan per baris *state* menggunakan satu byte memori temporer. *State* yang akan digeser ditempatkan secara bergantian dalam memori tersebut untuk kemudian dipertukarkan isi *state*-nya.

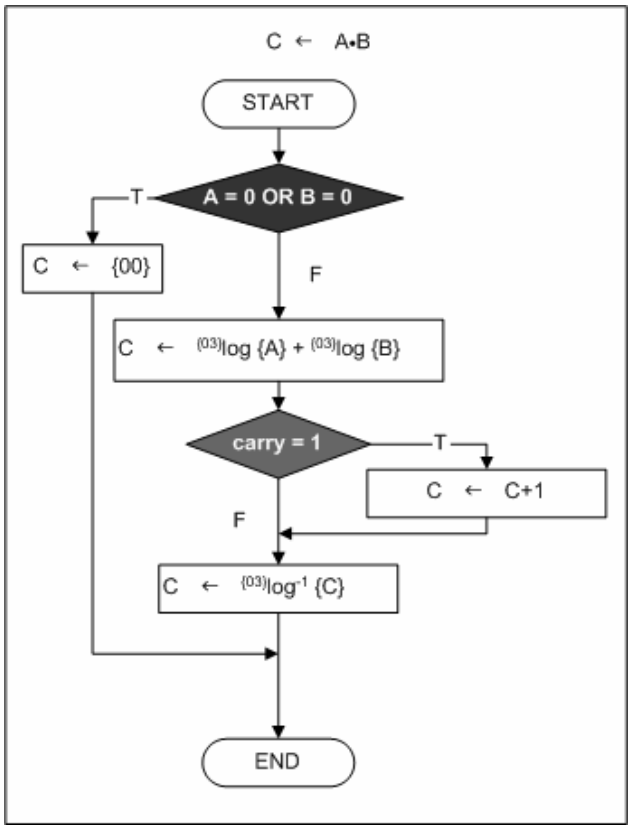
Alternatif lain adalah membuat fungsi ini implisit dalam fungsi lain [10]. Untuk enkripsi, setiap ShiftRows() diikuti oleh MixColumns() kecuali untuk *final round*. Jadi, pergeseran baris implisit dapat diwujudkan dengan mengubah *state* yang ditransformasikan MixColumns().

3.6.4. MixColumns()

Metoda perkalian dari implementasi yang sudah ada dieksploitasi di sini. Dari beberapa publikasi, cara mengalikan dapat dibagi menjadi dua, dengan menggunakan tabel log-antilog dan dengan pergeseran bit-berulang.

3.6.4.1. Perkalian dengan Tabel Log-Antilog

Tabel log-antilog yang dimaksud sudah tersedia sebagai tabel $^{\{03\}}\log \{xy\}$ dan $^{\{03\}}\text{antilog } \{xy\}$[3]. Penggunaan kedua tabel tersebut dimaksudkan agar tidak ada instruksi yang eksplisit mengekspresikan perkalian. Jika perkalian dilakukan dengan cara mengalikan kedua byte, kemudian diikuti modulonya, maka hasil antara perkalian (sebelum dimodulo) dapat melebihi ukuran 1 byte. Tabel tersebut mencegah komputasi berlangsung di atas unit byte.



Gambar 3-4. Diagram Alir Perkalian dengan Tabel Log-Antilog.

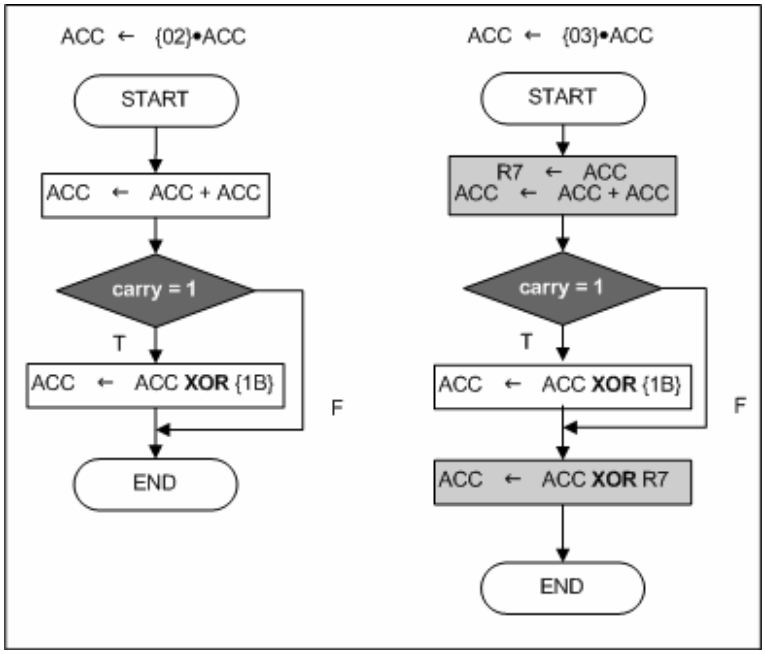
Jika tabel log-antilog digunakan, maka operasi aritmatiknya adalah penambahan (bukan perkalian). Idenya adalah perkalian memanfaatkan penjumlahan pangkat, jika $a = g^{\alpha}$ dan $b = g^{\beta}$ maka $a \bullet b = g^{\alpha+\beta}$. Hasil perkalian

(notasi ●) diperoleh dengan terlebih dahulu mengambil nilai log tiap bilangan yang dikalikan. Jumlah kedua nilai log tersebut akan menjadi hasil dari perkalian sesungguhnya setelah diambil antilognya. Komputasi selengkapnya dideskripsikan Gambar 3-4. Sejumlah penyederhanaan langkah dari algoritma semula [3] telah dilakukan. Dalam tabel $^{\{03\}}\log\{xy\}$ jumlah terbesar dari dua elemen-elemennya adalah $\{1fc\}^{16}$. Lebih lanjut, tabel ini analog dengan perputaran jarum jam (bilangan modulo), jika penjumlahan tersebut melebihi {ff} (ada *carry*), maka nilai sebenarnya diperoleh dengan menguranginya dengan {ff}. Pengurangan dengan mengambil *2's complement* {ff} sama dengan sekali inkremen. Tabel log-antilog dapat digunakan untuk semua perkalian dalam $GF(2^8)$.

3.6.4.2. Perkalian dengan Pergeseran Berulang

Pergeseran bit-berulang diwujudkan melalui serangkaian xtime() (Bagian 2.2.2). MixColumns() **hanya** melibatkan pengali {02} dan {03}. Jika xtime() dipakai, perkalian dengan {03} hanya selangkah lebih jauh dari perkalian dengan {02} seperti diilustrasikan Gambar 3-5.

[16] Digit '1' dalam {1fc} adalah bit ke-9 (indeks MSB) yang merupakan carry dari penjumlahan dua byte.



Gambar 3-5. Diagram Alir Perkalian {02} dan {03} dengan Pergeseran Berulang.

Dengan mengaplikasikan pergeseran berulang, perkalian {03} dapat dijabarkan sebagai

$$\{03\} \bullet b(x) = (\{02\} \bullet b(x)) \oplus b(x) \tag{3.1}$$

Persamaan (2.4) dapat disusun kembali dalam bentuk di bawah ini [5].

$$\left.\begin{aligned} p &= s_{0,c} \oplus s_{1,c} \oplus s_{2,c} \oplus s_{3,c} \\ s'_{0,c} &= \{02\} \bullet (s_{0,c} \oplus s_{1,c}) \oplus p \oplus s_{0,c} \\ s'_{1,c} &= \{02\} \bullet (s_{1,c} \oplus s_{2,c}) \oplus p \oplus s_{1,c} \\ s'_{2,c} &= \{02\} \bullet (s_{2,c} \oplus s_{3,c}) \oplus p \oplus s_{2,c} \\ s'_{3,c} &= \{02\} \bullet (s_{3,c} \oplus s_{0,c}) \oplus p \oplus s_{3,c} \end{aligned}\right\} \tag{3.2}$$

Apabila *look up table* digunakan untuk persamaan (3.2), hanya dibutuhkan tabel perkalian {02} (256 byte) untuk MixColumns().

3.7. Dekripsi

Transformasi-transformasi yang merupakan kebalikan dari *cipher* diterapkan dalam program dekripsi. Fungsi AddRoundKey() untuk enkripsi digunakan kembali untuk dekripsi. Yang harus dibuat lagi adalah InvSubBytes(), InvShiftRows(), dan InvMixColumns(). Beberapa bagian cukup dikopi dari fungsi kebalikannya yang digunakan saat enkripsi. Untuk InvSubBytes(), perubahan hanya pada *look up table* yang digunakan (Bagian 3.6.1), yaitu S-Box^{-1}. InvShiftRows() juga hanya dibedakan oleh indeks *state* yang digeser (Bagian 2.5.3). Hanya modifikasi tersebut yang diperlukan dalam penulisan fungsi InvSubBytes() dan InvShiftRows(). Seperti halnya ShiftRows() (Bagian 3.6.3), InvShiftRows() dapat dibuat implisit [10] dalam fungsi lain yang mengikutinya, yaitu dalam InvSubBytes().

3.7.1. InvMixColumns()

Pengali dalam InvMixColumns() adalah {09}, {0b}, {0d}, dan {0e}. Jika metoda log-antilog digunakan (Bagian 3.6.4.1), maka MixColumns()-InvMixColumns() akan berbagi tabel yang sama untuk perkalian (2x256 byte *code memory*).

Cara kedua, dengan tabel ekstra, adalah dengan menyusun ulang persamaan (2.5) seperti yang dilakukan pada persamaan (3.2) (Bagian 3.6.4.2). Tabel perkalian {02} yang digunakan MixColumns() dapat digunakan bersama-sama (256 byte), sedangkan khusus InvMixColumns() ditambahkan tabel perkalian {09} dan perkalian {0d} (2x256 byte). Perkalian {0b} dan {0e} dijabarkan sebagai kombinasi perkalian {02}, {09} atau {0d}. Penjabaran perkalian tersebut dilakukan dengan melakukan sejumlah pergeseran berulang (Bagian 2.2.2). Penjabaran ini serupa dengan yang dilakukan persamaan (3.1) (Bagian 3.6.4.2).

$$\left.\begin{aligned} \{0b\} \bullet b(x) &= (\{09\} \bullet b(x)) \oplus (\{02\} \bullet b(x)) \\ \{0e\} \bullet b(x) &= (\{0d\} \bullet b(x)) \oplus (\{02\} \bullet b(x)) \oplus b(x) \end{aligned}\right\} \quad (3.3)$$

Persamaan (3.3) tersebut menjabarkan perkalian {0b} sehingga dapat diwujudkan dengan tabel $\{09\} \bullet b(x)$ dan $\{02\} \bullet b(x)$. Perkalian {0e} juga dijabarkan memakai prinsip yang sama. Penyusunan ulang persamaan (2.4) yang telah menyertakan substitusi persamaan (3.3) adalah seperti berikut ini.

$$\left.\begin{aligned}
s'_{0,c} &= \left(\{0d\} \bullet (s_{0,c} \oplus s_{2,c})\right) \oplus \left(\{09\} \bullet (s_{1,c} \oplus s_{3,c})\right) \oplus \left(\{02\} \bullet (s_{0,c} \oplus s_{1,c})\right) \oplus s_{0,c} \\
s'_{1,c} &= \left(\{0d\} \bullet (s_{1,c} \oplus s_{3,c})\right) \oplus \left(\{09\} \bullet (s_{0,c} \oplus s_{2,c})\right) \oplus \left(\{02\} \bullet (s_{1,c} \oplus s_{2,c})\right) \oplus s_{1,c} \\
s'_{2,c} &= \left(\{0d\} \bullet (s_{0,c} \oplus s_{2,c})\right) \oplus \left(\{09\} \bullet (s_{1,c} \oplus s_{3,c})\right) \oplus \left(\{02\} \bullet (s_{2,c} \oplus s_{3,c})\right) \oplus s_{2,c} \\
s'_{3,c} &= \left(\{0d\} \bullet (s_{1,c} \oplus s_{3,c})\right) \oplus \left(\{09\} \bullet (s_{0,c} \oplus s_{2,c})\right) \oplus \left(\{02\} \bullet (s_{0,c} \oplus s_{3,c})\right) \oplus s_{3,c}
\end{aligned}\right\} \quad (3.4)$$

Total jumlah tabel yang diperlukan InvMixColumns() adalah tiga buah (3x256 byte), lebih kecil daripada jika kesemuanya ditabelkan (empat tabel perkalian, {09}, {0b}, {0d}, dan {0e}).

3.8. Kontrol, Status, dan I/O

Fungsi-fungsi kontrol dan status proyek yang berisi fungsi `main()`. Kontrol diwujudkan dalam fungsi `Mode()` (semua diagram terlampir) yang mengecek port 1. P1.0 dan P1.1 digunakan untuk memilih mode komputasi, P1.0 *active low* untuk memilih enkripsi, P1.1 *active low* untuk memilih dekripsi. Kedua port tersebut tidak dibaca sebagai bit, tetapi dalam byte P1 (port 1).

Port 0 menjadi indikator status sistem ke luar. Status dibuat dalam enam tahana, `READY`, `ENCRYPT`, `DECRYPT`, `INKEY`, `INTEXT`, dan `COMMANDERROR`. Status inisial setelah reset adalah `READY`, status berikutnya mengikuti nilai *return* fungsi `Mode()`. Status `COMMANDERROR` diberikan jika `Mode()` gagal membaca atau menerjemahkan kontrol. Indikator status ini ditulis dalam program C sebagai fungsi `Status(ModeOpr)` dengan argumen variable `ModeOpr`. Status yang aktif ditandai sebagai bit '0' (*low*) di port 0. Pengubahan nilai P0 (port 0) dilakukan per byte.

Fungsi `Status()` dan `Mode()` ditulis dalam bahasa C tanpa ada optimasi lebih jauh. `InputToState()` dan `StateToOutput()` ditulis dalam bahasa

assembly-nya. Kedua fungsi inilah yang memetakan aliran data serial ke *state* dan kebalikannya. Dua fungsi tersebut mengakses variabel *state* sebagai register (Gambar 3-2). Secara bergantian bank register 1 dan 2 diaktifkan saat input-ouput blok data. Kesemua fungsi tersebut diatur dalam fungsi `main()` dari keseluruhan proyek Keil ini.

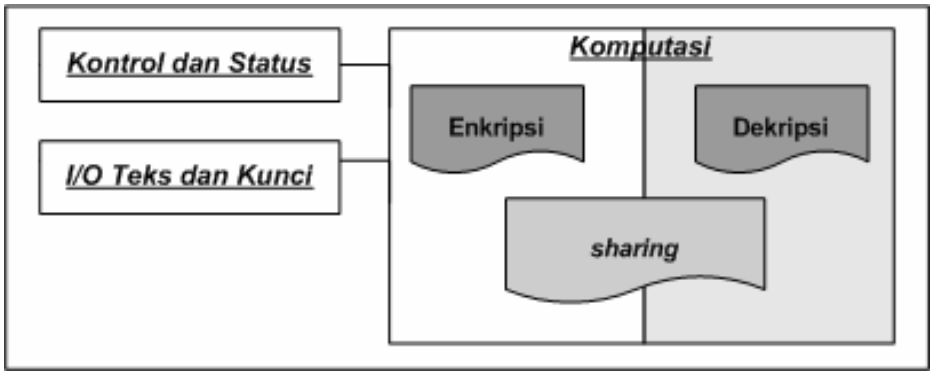
Tidak ada perhatian khusus yang diberikan dalam pembuatan fungsi-fungsi di atas. Fungsi-fungsi tersebut ditujukan untuk digunakan selama pengembangan saja dan dapat diganti-ganti sesuai adaptasi yang dibutuhkan implementasi.

3.9. Struktur Proyek Keil Program Mikrokontroler

Agar komputasi dapat dievalusi terpisah, program yang memuat fungsi-fungsi enkripsi-dekripsi ditulis dalam *source code* tersendiri. Lebih jauh, pemisahan dalam bentuk modul-modul tunggal memungkinkan penulisan masing-masing dalam bahasa C maupun *assembly* (Gambar 3-6).

Selain itu, modul komputasi dapat dipakai kembali dengan adaptasi pada bagian kontrol, status atau I/O-nya. Apapun adaptasi yang dilakukan, modul komputasi AES-128 ini tidak perlu mengalami perubahan apa-apa. Jadi, jumlah *cycle* dan ukuran program komputasi tidak berubah ketika adaptasi dilakukan.

Sejumlah fungsi maupun *look up table* akan dipakai secara bersama-sama oleh enkripsi maupun dekripsi. Enkripsi-dekripsi juga berbagi sumber daya lain, yaitu alamat-alamat RAM untuk ekspansi kunci dan penyimpanan *state*.



Gambar 3-6. Struktur Program dalam Proyek Keil.

Bab 4
Pengujian dan Analisis

4.1. Simulasi Fungsional

Selama pengembangan program, contoh vektor uji FIPS-197 [9] digunakan untuk pengujian kebenaran fungsional lewat simulasi pada *debugger* Keil. Contoh *plain text*, *key*, dan *cipher text* yang digunakan selama simulasi tersebut adalah

```
KEY=000102030405060708090A0B0C0D0E0F
PT =00112233445566778899AABBCCDDEEFF
CT =69C4E0D86A7B0430D8CDB78070B4C55A
```

Perubahan isi variabel *state* per transformasi beserta tiap Round Key juga tertera lengkap pada dokumen tersebut sehingga hasil antara per transformasi dapat dicek kebenarannya dengan mengacu ke situ.

Contoh *Key Expansion* panjang kunci 128 disediakan beserta *intermediate result* per transformasi. Vektor uji kunci tersebut tertera bawah ini.

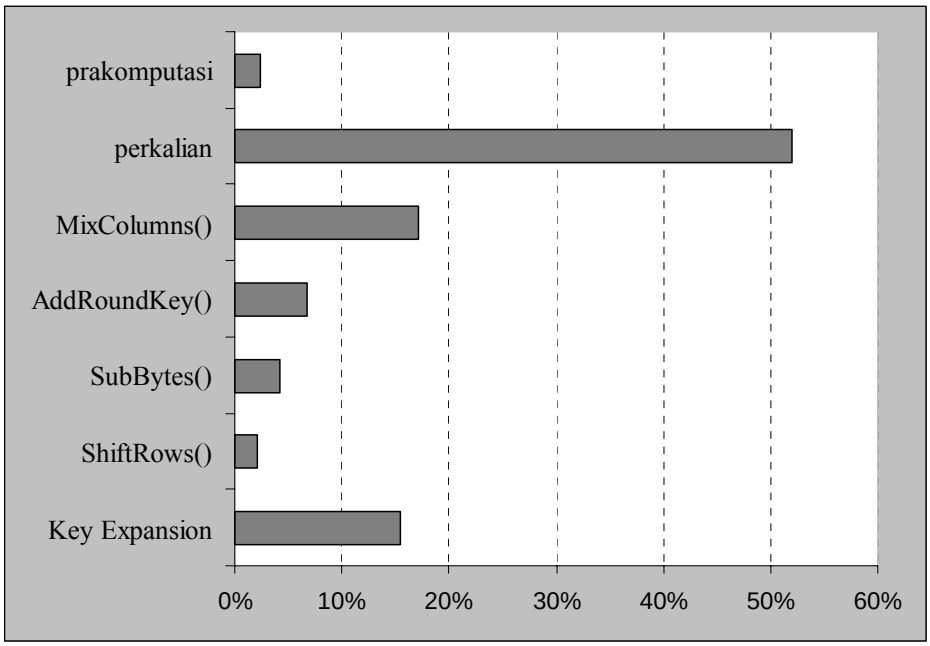
```
KEY=2B7E151628AED2A6ABF7158809CF4F3C
```

4.2. Pusat Perhatian Optimasi Program

Program *cipher* versi pertama ditulis dalam bahasa C. Setelah kebenaran fungsional terpenuhi dalam simulasi (Bagian 4.1), analisis dilakukan untuk memutuskan bagian program mana yang menjadi pusat perhatian optimasi. Analisis terhadap program yang hanya memuat enkripsi ini menjadi umpan balik bagi perancangan program enkripsi-dekripsi versi lengkap.

Sekedar untuk simulasi program tersebut, masukan teks dan kunci ditulis sebagai operasi penyimpanan (*storing/loading*) ke RAM internal (`MOV state,#data`). Jadi, waktu eksekusi keseluruhan tidak merepresentasikan I/O, kontrol, dan status yang sesungguhnya (*dummy* saja).

Evaluasi laporan PA memberikan total waktu eksekusi 16078 *cycle*. Dari jumlah tersebut, 11110 *cycle* disumbangkan oleh `MixColumns()`. Lebih jauh lagi, pembengkakan waktu eksekusi `MixColumns()` disebabkan oleh lambatnya perkalian. Fungsi perkalian jika dihitung terpisah dari `MixColumns()` mendominasi lebih dari 52% waktu eksekusi enkripsi (Gambar 4-1). Walaupun persentase tersebut tidak menyertakan eksekusi I/O yang sesungguhnya, ia sudah menyatakan fakta lambatnya `MixColumns()`. Proses masukan teks-kunci serta fungsi `main()` pada Gambar 4-1 diakumulasi sebagai prakomputasi () yang bernilai 380 cycle.



Gambar 4-1. Evaluasi PA yang Memperlihatkan Lambatnya Perkalian.

Berdasarkan Gambar 4-1, prioritas optimasi muncul dalam urutan `MixColumns()` (termasuk perkalian), ekspansi kunci, dan `AddRoundKey()`. Dari evaluasi *assembly* hasil kompilasi, fungsi-fungsi *round* lainnya secara subyektif

sudah dianggap efisien. Evaluasi *cycle* untuk fungsi-fungsi *round* dan ekspansi kunci dapat dilihat pada Tabel 4-1 (prakomputasi tidak disertakan).

Tabel 4-1. Kecepatan Cipher Program Pertama.

Fungsi Round	*cycle*
Key Expansion	2494
ShiftRows()	340
SubBytes()	680
AddRoundKey()	1074
MixColumns()	11110
TOTAL	15698

Pusat perhatian program diletakkan dalam *constraint* kecepatan (*cycle*), karena sulit (secara praktis) untuk menulis program dalam dua fokus sekaligus: menaikkan kecepatan dan menurunkan ukuran program.

4.3. Perbandingan Variasi Pendekatan Optimasi

Telah disebutkan (Bagian 4.2) bahwa optimasi dilakukan pada fungsi-fungsi MixColumns(), Key Expansion, dan AddRoundKey(). Variasi teknik optimasi yang telah disinggung sebelumnya (Bagian 3.4) dievaluasi di sini. Sebuah catatan khusus adalah opsi *code optimization* pada Keil dibiarkan ***default***.

4.3.1. Analisis Optimasi MixColumns()

Perkalian dengan tabel log-antilog memungkinkan `InvMixColumns()` memanggil subrutin perkalian yang sama dengan `MixColumns()`. Namun, jika berpikir dari sisi *cycle* `MixColumns()` saja, metoda ini tidak menguntungkan (Bagian 3.6.4.2).

Penggunaan pergeseran berulang untuk `MixColumns()` memberikan hasil berikut ini.

Tabel 4-2. MixColumns() Menggunakan Perkalian dengan Pergeseran Berulang.

MixColumns()	Bahasa Pemrograman	Indeks *state*	*cycle*
	C	`state[baris][kolom]`	5317
	assembly	`state[kolom][baris]`	3752

Semula `MixColumns()` ditulis dalam bahasa C, penulisan ulang dengan bahasa assembly menghasilkan peningkatan kecepatan 29.43%. Setelah optimasi tersebut, perkalian {02} berlangsung maksimal enam cycle, sedangkan perkalian {03} berlangsung maksimal delapan cycle. Keuntungan yang diperoleh dari perkalian dengan pergeseran berulang ini[17] adalah `MixColumns()` (*assembly*) lebih cepat 66.23% daripada jika tabel log-antilog digunakan (Tabel 4-1).

Kerugiannya adalah kecepatan komputasi `MixColumns()` tidak seragam (dianalisis lebih lanjut pada Bagian 0). Kecepatannya bergantung operand dari masing-masing perkalian. Untuk vektor uji yang dicobakan (Bagian 4.1), `MixColumns()` berlangsung minimal dalam 412 cycle, maksimal dalam 422 cycle. Jumlah 3752 cycle pada Tabel 4-2 adalah akumulasi kesembilan *round*. `MixColumns()` tiap *round* berlangsung dalam kecepatan antara 412 – 422 cycle.

Masih semata-mata dari sisi `MixColumns()`, perkalian yang lebih cepat dapat diwujudkan dengan menggunakan tabel. Seperti telah disinggung sebelumnya (Bagian 3.4.2), setiap perkalian dapat berlangsung dalam tiga cycle melihat tabel.

[17] Dalam program, perkalian diwujudkan sebagai dua fungsi, `mul_2()` dan `mul_3()`

Tabel 4-3. MixColumns dengan Tabel Perkalian.

	Tabel Perkalian	Indeks *state*	*cycle*
MixColumns()	{02} dan {03}	`state[baris][kolom]`	2457
	{02}	`state[kolom][baris]`	1332

Kecepatan 2457 cycle (Tabel 4-3) dicapai dengan menggunakan tabel perkalian {02} dan perkalian {03} untuk persamaan (2.4) (Bagian 2.4.4). Hasil yang 45.79% lebih baik dicapai walau hanya dengan satu tabel (perkalian {02}), yaitu lewat persamaan (3.2) (Bagian 3.6.4.2). Persamaan (3.2) hanya membutuhkan 15 XOR per kolom *state* yang dioperasikan, sedangkan persamaan (2.4) membutuhkan 16 XOR per kolomnya. Penabelan dari sebagian pergeseran berulang tersebut digunakan pula untuk `InvMixColumns()`.

4.3.2. Analisis Optimasi Key Expansion

Program Key Expansion yang ditulis dengan menggunakan pointer terbukti lebih lambat daripada yang menggunakan alamat langsung (Tabel 4-4). Ekspansi kunci yang tercepat dilakukan dengan menggunakan delapan word buffer siklik. Bukan hanya pada kecepatan, pilihan ukuran buffer siklik yang dipakai akan berpengaruh juga pada jumlah RAM yang dipakai. Masing-masing pilihan akan menghabiskan ruang RAM masing-masing lima word (20 byte), delapan word (32 byte) atau enambelas word (64 byte). Khusus untuk program AES-128 saja, RAM masih cukup, buffer Key Expansion berbagi tempat hanya dengan sejumlah variabel (Bagian 4.4).

Tabel 4-4. Kecepatan Ekspansi Kunci.

Key Expansion	Bahasa Pemrograman	Ukuran Buffer Siklik	*cycle*
	C	5 word	2494
	assembly	16 word	1906

	C	8 word	840

4.3.3. Analisis Optimasi AddRoundKey()

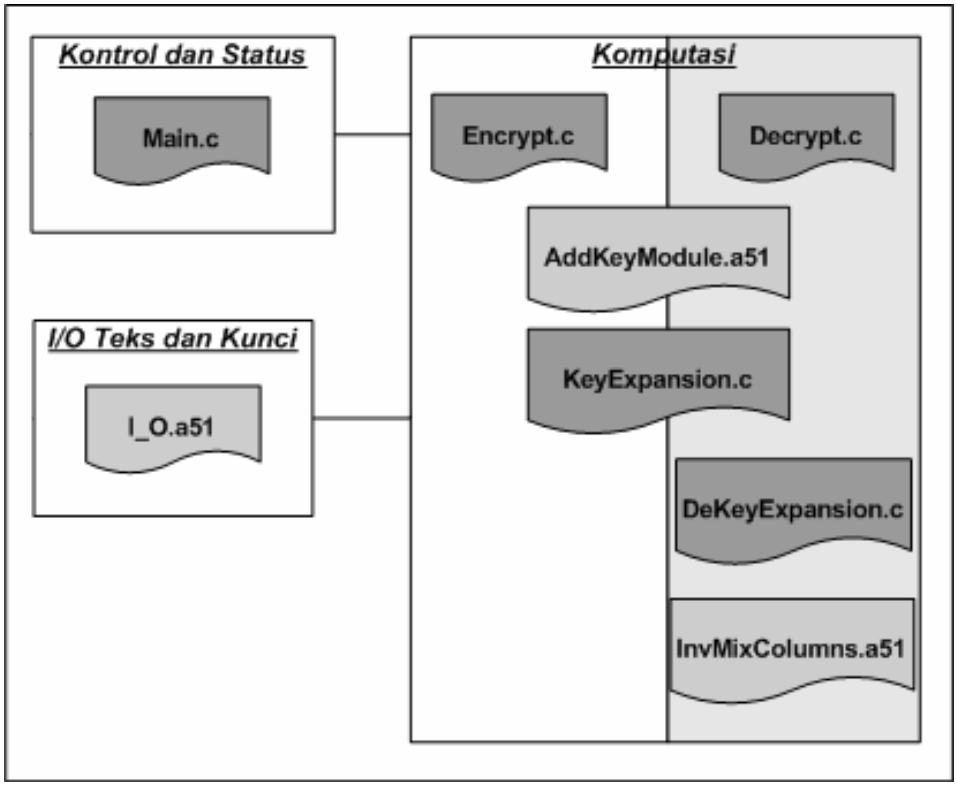
AddRoundKey() yang dipakai bergantung dari Key Expansion yang dipakai (Bagian 3.6.1). Laporan PA untuk optimasi yang dilakukan tertera dalam Tabel 4-5. Penjumlahan per blok dapat dilakukan apabila *Round Key* yang dihasilkan ekspansi per blok pula, yaitu dengan ukuran buffer siklik delapan atau enam belas *word* (Tabel 4-4). Penjumlahan per *word* dilakukan jika ukuran buffer siklik ekspansi kunci adalah lima *word*.

Tabel 4-5. Kecepatan AddRoundKey().

	Bahasa Pemrograman	Unit Penjumlahan	Indeks *state*	*cycle*
AddRoundKey()	C	*word*	`state[baris][kolom]`	1074
	assembly	*word*	`state[kolom][baris]`	834
	assembly	blok	`state[kolom][baris]`	572

4.4. Evaluasi Unjuk Kerja Program

Evaluasi program keseluruhan dilakukan lewat analisis versi terakhir program mikrokontroler. Umpan balik yang diberikan dari analisis Bagian 4.2 dan Bagian 4.3 menghasilkan versi terakhir program (Gambar 3-6). Fungsi komputasi yang ditulis dalam bahasa *assembly* (teroptimasi) adalah `AddRoundKey()` dan `InvMixColumns()` masing-masing dalam *file* `AddKeyModule.a51` dan `InvMixColumns.a51`. Fungsi I/O ditulis dalam bahasa *assembly* tanpa alasan khusus. Akan tetapi, `InputToState()` dan `StateToOutput()` dalam `I_O.a51` menggunakan konsep bank register (Bagian 3.4.2).



Gambar 4-2. Struktur Program Versi Terakhir dalam Proyek Keil.

Fungsi yang mengatur pemanggilan fungsi-fungsi round (komputasi) adalah `EncryptRound()` dan `DecryptRound()`, masing-masing untuk enkripsi dan dekripsi. Fungsi-fungsi tersebut seolah-olah menjadi `Main()` untuk masing-masing komputasi (enkripsi/dekripsi). Akan dihasilkan satu blok keluaran teks untuk satu blok masukan kunci dan satu blok masukan teks sehingga jumlah *cycle* enkripsi/dekripsi dihitung terpisah sebagai sekali `EncryptRound()` atau sekali `DecryptRound()`.

Perbandingan dengan implementasi enkripsi mikrokontroler 8 bit yang lain terangkum dalam Tabel 4-6. Perbandingan dilakukan per fungsi untuk sekali `EncryptRound()` dengan mengubah `Main.c` dan menghilangkan `I_O.a51` dari program versi terakhir. Ini dilakukan untuk memperoleh perbandingan yang obyektif. Pada tiap-tiap implementasi pembanding, masukan teks-kunci, ekspansi

kunci, dan percabangan diakumulasi sebagai prakomputasi. Juga disebutkan publikasi masing-masing [10,13] bahwa masukan teks-kunci merupakan proses penyimpanan ke memori (*load/store/move*) pasca-*reset*/inisial. Jadi, ia tidak menggambarkan I/O yang sebenarnya ('fiktif').

Pengubahan kontrol, status, dan I/O pada pembandingan di atas sekaligus mencontohkan **adaptasi** program yang dapat dilakukan tanpa mengubah-ubah modul komputasi AES-nya.

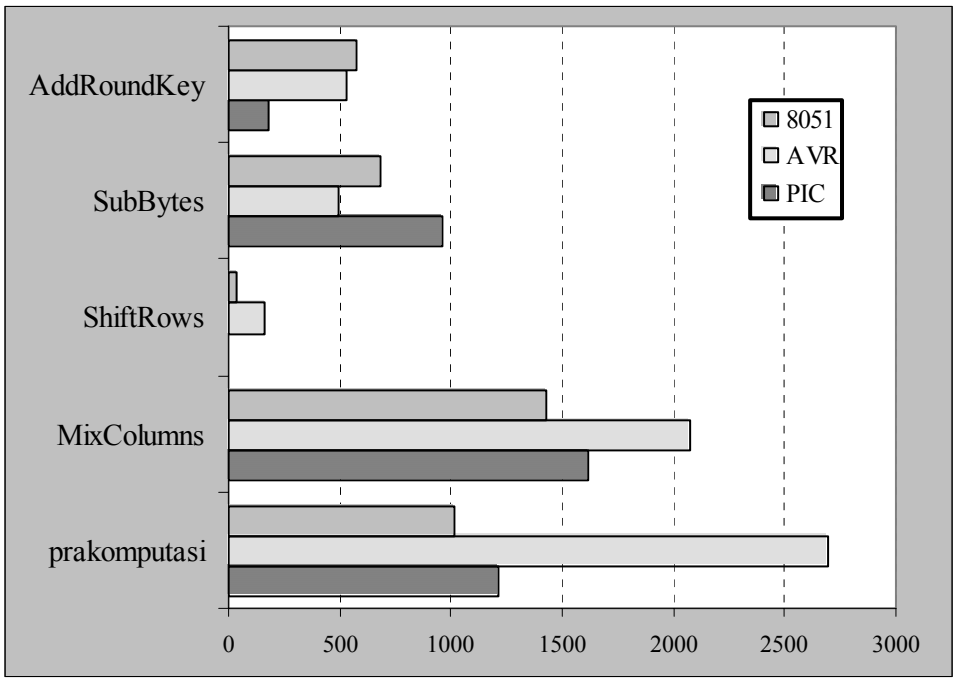
Tabel 4-6. Tabel Perbandingan Unjuk Kerja Tiap Fungsi Enkripsi.

Fungsi	*8051*†		*AVR*		*PIC*
	cycle	*byte*	*cycle*	*byte*	*cycle*
`prakomputasi`	1015	1029	2693	1848	1213
`AddRoundKey()`	572	68	528	192	176
`SubBytes()`	680	84	490	132	960
`ShiftRows()`	34	41	160	64	0
`MixColumns()`	1431	229	2079	104	1620
Jumlah	3732	1451	5950	2340	3969

† implementasi AES-128 yang dibuat
sumber: K. Sung Ha [13]; R. Ward [10].

Dari visualisasi perbandingan enkripsi (Gambar 4-3), sejumlah analisis muncul. Fungsi `AddRoundKey()` (pada 8051) yang menggunakan pointer adalah lebih lambat, setiap XOR (`XRL state+xxh, A`) selalu didahului dengan inkremen pointer yang menunjuk ke kunci yang akan dijumlahkan dan pemindahannya ke akumulator. Walaupun diperbandingkan dengan mikrokontroler lain, pembandingan masih relevan karena operasi XOR 8051 tersebut bernilai satu *cycle* saja. Fungsi `ShiftRows()` yang dibuat implisit dalam `MixColumns()` (Bagian 3.6.3) masih dipanggil sekali pada *round* final, sedangkan pada implementasi PIC `ShiftRows()` tidak dibuat sama sekali (implisit semuanya)

[10]. Dengan demikian, pembobotan pada Tabel 3-2 menjadi berubah, sembilan kali `ShiftRows()` dibebankan ke `MixColumns()`. Catatan lain, prakomputasi yang panjang pada AVR disebabkan oleh penyimpanan tabel S-Box ke RAM eksternal saat inisial dan ekspansi kunci 44 *word* ke RAM sekaligus (*single shot*) [13].



Gambar 4-3. Diagram Batang Perbandingan Fungsi-fungsi Enkripsi.

Ukuran program untuk implementasi PIC adalah 1739 *word* (menggunakan *single word instructions program memory*). Jadi, secara keseluruhan program 8051 yang dihasilkan memiliki jumlah *cycle* dan ukuran program **terkecil** dari tiga implementasi yang diperbandingkan.

Perbandingan berikutnya (Tabel 4-7) adalah dengan tugas akhir sebelumnya yang memakai Atmel AT90s8535 untuk dekripsi AES-128 [4]. Untuk fungsi-fungsi dekripsi hanya `AddRoundKey()`, `InvShiftRows()`, dan ekspansi kunci

yang memiliki *cycle* lebih kecil. Prakomputasi pada implementasi AVR tidak menyertakan I/O serialnya.

Tabel 4-7. Tabel Perbandingan Unjuk Kerja Tiap Fungsi Dekripsi.

Fungsi	*8051*†		*AVR*	
	cycle	*byte*	*cycle*	*byte*
`prakomputasi`	119	175	2693	
`InvShiftRows()`	0	0	55	
`AddRoundKey()`	572	68	575	3870
ekspansi kunci	1587	768‡	14534	
`InvSubBytes()`	760	344‡	313	
`InvMixColumns()`	2610	1117‡	1402	1546‡
Jumlah	5648	2472	22973	5416

† implementasi AES-128 yang dibuat, fungsi *round* dekripsi saja
‡ sudah termasuk tabel yang digunakannya
sumber: I. Hermawan [4].

Evaluasi program lengkap (enkripsi-dekripsi) yang sudah menyertakan I/O serial, kontrol, dan status tertera pada Tabel 4-8. Modul yang dipakai bersama (penjumlahan dan ekspansi kunci) berukuran 320 byte, sedangkan semua *look up table* yang dipakai enkripsi dipakai lagi oleh dekripsi yaitu, 522 byte. Jadi, total *code memory* yang dipakai untuk tabel konstanta adalah 1290 byte, sedangkan 2117 byte lagi untuk program.

Jumlah RAM internal yang dipakai adalah 52 byte dengan 16 byte alamat absolut (untuk *state*), 33 byte alamat *relocatable* (untuk ekspansi kunci), dan 3 byte alamat *overlayable*. Tiga byte yang terakhir digunakan oleh `MixColumns()` sebagai memori temporer, apabila fungsi ini tidak sedang dipanggil, maka alamat-alamat tersebut dapat digunakan fungsi lain (di-*overlay*). Alokasi RAM yang lain tidak dapat diganggu-gugat selama komputasi AES-128 berjalan.

Komputasi AES-128 untuk enkripsi berlangsung dalam 3658 *cycle*, sedangkan untuk dekripsi berlangsung dalam 5648 *cycle*. I/O serial menggunakan 19200 bps (kristal 11.059 MHz) maka untuk tiga blok (masukan teks, kunci, dan keluaran teks) dapat diambil taksiran nilai 0.02 sekon ≈ 18432 cycle.

Tabel 4-8. Evaluasi Program AES-128 pada Mikrokontroler 8051.

			Modul/Fungsi	*byte*	*cycle*
kontrol, status, I/O			`Main`	168	
			I/O	265	
komputasi AES-128	enkripsi		`EncryptRound()`	148	101
			`ShiftRows()`	41	34
			`MixColumns()`	229	1431
			`SubBytes()`	84	680
			`AddRoundKey()`	68	572
			Key Expansion	252	840
	dekripsi		`DecryptRound()`	175	119
			`InvShiftRows()`	0	0
			`InvMixColumns()`	349	2610
			`InvSubBytes()`	88	760
			Inverse Key Expansion	250	747
look up table	dekripsi	enkripsi	`S_Box`	256	
			`Rcon`	10	
			`mul_2`	256	
			`IS_Box`	256	
			`mul_9`	256	
			`mul_D`	256	
Jumlah				3407	

Campuran C – assembly yang digunakan terbukti masih efektif untuk menghasilkan kecepatan dan ukuran program yang representatif (Tabel 4-9). Program (fungsi *round* saja) yang dihasilkan hanya 13.4 % lebih lambat dan 30 % lebih besar terhadap implementasi 8051 tercepat yang dipublikasikan. *Throughput* komputasi (31.6 kbps) bisa mencapai 114.3 kbps pada versi *enhancement* 8051 yang dapat bekerja dengan kristal 40 MHz (Bagian 2.6.3).

Bagian yang berbahasa C akan dependen terhadap *compiler*. Untuk fleksibilitas dibangkitkan pula versi *assembly* dari keseluruhan modul sehingga program dapat dipakai kembali.

Tabel 4-9. Perbandingan dengan Implementasi Enkripsi Rijmen et al.

implementasi AES-128	*cycle*	*byte*	*throughput*
8051†	3732	1676	31.6 kbps
8051‡	3658	1451	32.2 kbps
8051 Rijmen *et al.* [5]	3168	1016	37.2 kbps

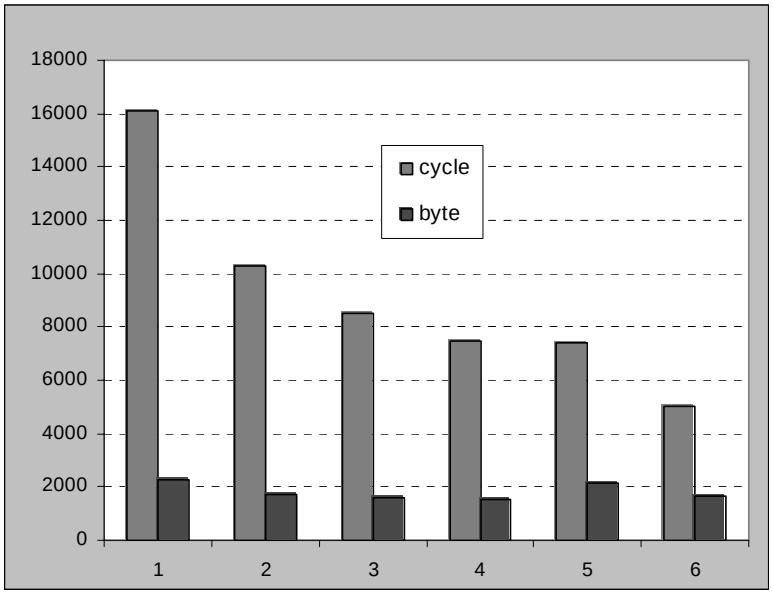
† dengan prakomputasi
‡ fungsi *round* saja
throughput[18] untuk kristal 11.059 MHz

Pengujian kebenaran modul komputasi AES-128 dilakukan dengan 568 kombinasi vektor uji (AESAVS dan proposal Rijndael) pada Lampiran B. Validasi tersebut otomatis turut menguji modul kontrol, status, dan I/O. Pengujian dilakukan pada sistem mikrokontroler AT89s8252 (Lampiran A).

[18] *Throughput* dihitung untuk masukan teks 128 bit per komputasi, 1 cycle = 1.085 μs.

4.5. Analisis Riwayat Optimasi Program

Riwayat optimasi program enkripsi dan dekripsi (Tabel 4-10 dan



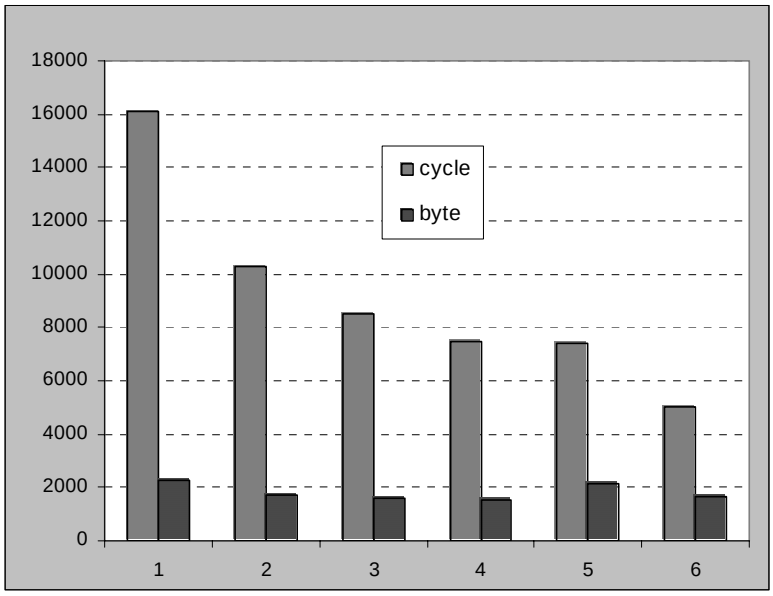
Gambar 4-4. Visualisasi Riwayat Optimasi Enkripsi.

Tabel 4-11) menunjukkan bahwa sejalan dengan upaya penurunan *cycle* program, terjadi penurunan ukuran program pula. Pengecualiannya hanya pada penggunaan tabel perkalian {02} dan {03} karena kita harus menambahkan 256 byte ekstra dibandingkan dengan yang hanya menggunakan satu *look up table* perkalian. Riwayat tersebut juga menunjukkan bahwa penggunaan indeks *state* yang dibalik (`state[col][row]`) turut mendukung langkah optimasi lainnya, misalnya dari riwayat no. 1 ke no. 2 (Tabel 4-10), tidak ada perubahan metoda, hanya penulisan ulang beberapa modul dalam bahasa *assembly* yang disertai perubahan indeks *state*.

Berikut ini adalah riwayat optimasi program sebelum diperoleh versi terakhir program (Bagian 4.4).

Tabel 4-10. Riwayat Optimasi Program Enkripsi.

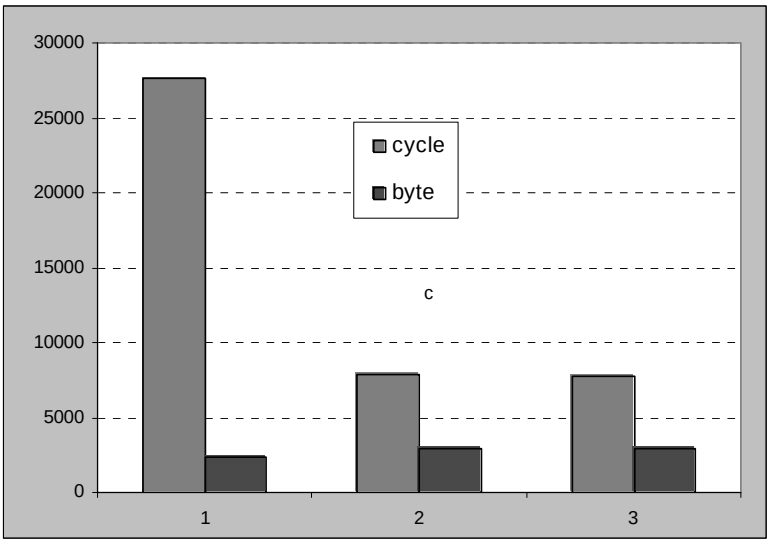
no.	keterangan	*cycle*	*byte*
1.	*indexing*: `state[row][col]` perkalian dengan tabel log-antilog buffer siklik ekspansi kunci: 5 *word* modul *assembly*: -	16078	2264
2.	*indexing*: `state[row][col]` perkalian dengan pergeseran berulang buffer siklik ekspansi kunci: 5 *word* modul *assembly*: -	10285	1685
3.	*indexing*: `state[col][row]` perkalian dengan pergeseran berulang buffer siklik ekspansi kunci: 5 *word* modul *assembly*: >`AddRoundKey()` per *word* >`MixColums()` >`mul_2()` >`mul_3()`	8480	1620
4.	*indexing*: `state[col][row]` perkalian dengan pergeseran berulang buffer siklik ekspansi kunci: 16 *word* modul *assembly*: >`AddRoundKey()` per blok >`MixColums()` >`mul_2()` >`mul_3()` >ekspansi kunci	7446	1514
5.	*indexing*: `state[row][col]` perkalian dengan tabel {02} dan {03} buffer siklik ekspansi kunci: 5 *word* modul *assembly*: -	7425	2156
6.	*indexing*: `state[col][row]` pergeseran berulang yang ditabelkan (tabel xtime()) buffer siklik ekspansi kunci: 16 *word* modul *assembly*: >`AddRoundKey()` per blok >ekspansi kunci	5026	1631



Gambar 4-4. Visualisasi Riwayat Optimasi Enkripsi.

Tabel 4-11. Riwayat Optimasi Program Dekripsi.

no.	keterangan	*cycle*	*byte*
1.	*indexing*: `state[row][col]` perkalian dengan tabel log-antilog buffer siklik ekspansi kunci: 5 *word* modul *assembly*: - Cipher Key yang digunakan adalah Round Key terakhir dari ekspansi kunci enkripsi	27668	2380
2.	*indexing*: `state[col][row]` pergeseran berulang yang ditabelkan buffer siklik ekspansi kunci: 16 *word* modul *assembly*: >`AddRoundKey()` per blok >ekspansi kunci	7836	2924
3.	*indexing*: `state[col][row]` pergeseran berulang yang ditabelkan buffer siklik ekspansi kunci: 16 *word* modul *assembly*: >`AddRoundKey()` per blok `>InvMixColums()` > ekspansi kunci	7764	2916



Gambar 4-5. Visualisasi Riwayat Optimasi Dekripsi.

4.6. Analisis Keamanan terhadap Serangan Spesifik Mikrokontroler

Implementasi perangkat keras dilakukan sebatas untuk kepentingan uji kebenaran program komputasi AES-128 di dalamnya. Sistem mikrokontroler keseluruhan (termasuk perangkat lunaknya) belum aplikatif mengerjakan fungsi tertentu selain pengujian. Jadi, isu keamanan ditinjau dari modul komputasi AES-128. Program mikrokontroler yang dibuat memiliki sejumlah kekuatan yang berupa *software countermeasure*.

Dengan komputasi yang dijalankan internal, satu-satunya peristiwa data ada di port mikrokontroler adalah saat pemasukan kunci dan I/O teks. Pemasukan kunci pun menjadi tidak perlu jika ia ditanam dalam *code memory* (tentunya menjadi tidak bisa diganti tanpa pemrograman ulang), *download* program ke μC dapat dilakukan dengan *lock protect bit* [2] sehingga isi *code* tidak dapat dibaca lagi dari μC.

4.6.1. Penangkalan Serangan Timing Analysis

Perkalian dengan tabel log-antilog maupun pergeseran berulang (Bagian 3.6.4.2) memiliki dependensi terhadap operand, ada percabangan program (*conditional jump*) yang tidak dieksekusi untuk operand tertentu (sebagaimana dideskripsikan Gambar 3-4 dan Gambar 3-5). Sebagai hasilnya, muncul variasi *cycle* `MixColumns()` untuk *plain text* yang disimulasikan (Bagian 4.3.1).

Dependensi waktu komputasi terhadap data dihindari dengan membuat *code coverage* fungsi 100% untuk semua operan. Pada `MixColumns()` yang menggunakan tabel log-antilog maupun pergeseran berulang di atas, sejumlah `NOP` perlu disisipkan agar tiap operan perkalian menempuh panjang *cycle* yang sama untuk tiap percabangan yang dilaluinya [5,8]. Program versi terakhir tidak memerlukan penyisipan `NOP` tersebut karena tidak ada *conditional jump* dalam `MixColumns()` yang dibuat (persamaan (3.2)). Begitu halnya dengan `InvMixColumns()` (persamaan (3.4)). Laporan *code coverage* 100% ini dilihat pada *debugger* Keil (Bagian 3.3).

Bab 5
Simpulan dan Saran

5.1. Simpulan

- Metoda optimasi program yang berfokus pada *constraint* kecepatan telah berhasil meningkatkan unjuk kerja program komputasi AES-128 keseluruhan.
- Enkripsi-dekripsi AES-128 telah berhasil diimplementasikan pada mikrokontroler 8051 dengan total ukuran program 3407 byte dan kecepatan komputasi enkripsi-dekripsi berturut-turut 3658 *cycle* dan 5648 *cycle*.
- Secara keseluruhan program AES-128 yang dihasilkan dapat dimuat ke dalam mikrokontroler 8051 generik (4kB *code memory* dan 128 byte *data memory*) dengan menyisakan kurang dari 600 byte *code memory* untuk program lainnya. Dengan begitu, semua varian 8051 lain dapat menggunakan modul komputasi ini.

5.2. Saran

- Penggunaan bahasa *assembly* sepenuhnya akan sangat mungkin mempercepat komputasi dan mereduksi ukuran program.
- Pada implementasi ini telah dilakukan *software countermeasure* terhadap *timing analysis attack*. Peningkatan keamanan selanjutnya adalah dengan melakukan *countermeasure* terhadap *power analysis attack*. Namun hingga saat ini, *power analysis attack* untuk AES masih prematur, belum praktis, dan merupakan polemik yang belum usai.

Daftar Pustaka

[1] AEC Electronics, "PC Based 89s8252 ISP," 1997.

[2] Atmel Corp., "Datasheet: Atmel AT8928252, 8-bit Microcontroller with 8Kbytes Flash," 2000.

[3] Brian Gladman, "A Specification for Rijndael, the AES Algorithm," 2002.

[4] I. Hermawan, "Implementasi Algoritma Dekripsi AES-128 pada Mikrokontroler Atmel AT90s8535," Februari 2004.

[5] J. Daemen, V. Rijmen, "Efficient Block Ciphers for Smartcards," http://www.usenix.org/publications/library/proceedings/smartcard99/full_papers/daemen/daemen_html, 1999.

[6] J. Daemen, V. Rijmen, "AES Proposal: Rijndael," 1999.

[7] M. Aigner, E. Oswald, "Power Analysis Tutorial," http://www.iaik.tu-graz.ac.at/aboutus/people/oswald/papers/dpa_tutorial.pdf

[8] M. Janke, P. Laackmann, "Power and Timing Analysis Attack Againts Security Controllers," http://www.silicon-trust.com/pdf/secure_5/40_techno_3.pdf

[9] NIST, "Advanced Encryption Standard," *Federal Information Processing Standards Publication 197*, November 2001.

[10] R. W. Ward, T. C. A. Molteno, "A CPLD Coprocessor for Embedded Cryptography," http://www.physics.otago.ac.nz/electronics/papers/WardMolteno103.pdf

[11] Scott MacKenzie, "The 8051 Microcontroller, 2nd Ed.," *Prentice Hall*, 1995.

[12] Sencer Yeralan, Ashutosh Ahluwalia, "Programming and Interfacing the 8051 Microcontroller," *Addison-Wesley Publishing Co.*, 1995.

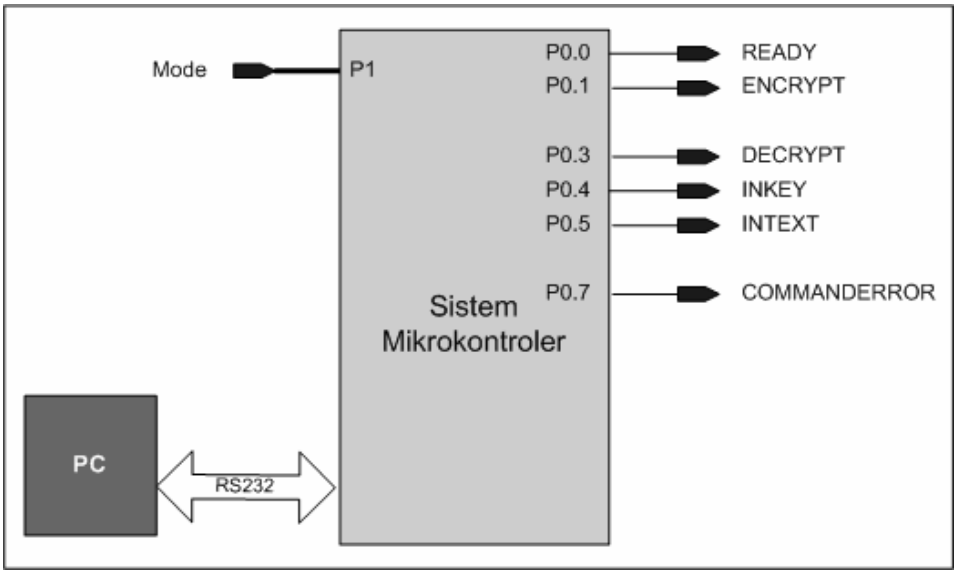
[13] Sungha Kim, Ingrid Verbauwhede, "AES Implementation on 8-bit Microcontroller," 2003.

[14] Situs *NIST Computer Security Resource Center* http://csrc.nist.gov/CryptoToolkit/

[15] Situs FAQ *debugger* Keil http://www.keil.com/uvision2/debuggerfaq.asp

Lampiran A
Perancangan dan Implementasi Perangkat Keras Pengujian

Sistem mikrokontroler terdiri atas mikrokontroler AT89s8252, driver TTL – RS232, regulator tegangan dan koneksi *In System Programming* (ISP). Mikrokontroler 8051 lain dapat dipakai di sini, AT89C51 – 52, AT89S51 – S52, dan uC lain yang kompatibel dengan keluarga MCS-51™ 40 pin DIP (*Dual Inline Package*) [11]. Akan tetapi, ISP-nya hanya dapat dipakai oleh AT89s8252. Keuntungannya adalah saat *debugging* dan *troubleshooting* chip mikrokontroler ini tidak perlu dicabut untuk dipindah ke *EEPROM/flash programmer*. Jadi, program pada *flash memory* cukup diganti dengan men-*download* lagi lewat koneksi ISP. Pemrograman dilakukan dengan *software* AEC_ISP.exe (*freeware* yang berbasis DOS). Koneksi ISP untuk program ini adalah lewat port paralel [1].



Gambar A-1. Blok Diagram Sistem Mikrokontroler Pengujian.

Perbedaan AT89s8252 dengan *library* generik 8051 atau 8052 (Bagian 3.3) adalah disediakannya kemampuan *Serial Peripheral Interface* (SPI), *watchdog timer* (SFR `WMCON`), dua `DPTR`, sembilan sumber *interrupt*, dll. Akan tetapi, fitur yang dieksploitasi di sini hanya ISP-nya.

B. 1. Selektor Mode dan Indikator Status

Untuk kebutuhan display dan monitoring status *on-board*, digunakan *array* LED yang terhubung ke port 0. Nyala LED memanfaatkan arus port saat *active-low* (I_{OL}) [2] yang dapat ditarik hingga 3.2 mA untuk port 0, pembatas arusnya adalah R 820Ω.

Untuk menyimulasikan pemilihan mode (enkripsi-dekripsi) selama pengujian program, digunakan saklar yang terhubung ke P1.0 dan P1.1. Status juga dapat dilihat pada kondisi LED yang terhubung ke port 0. LED menyala jika status program yang bersangkutan aktif (Bagian 3.8).

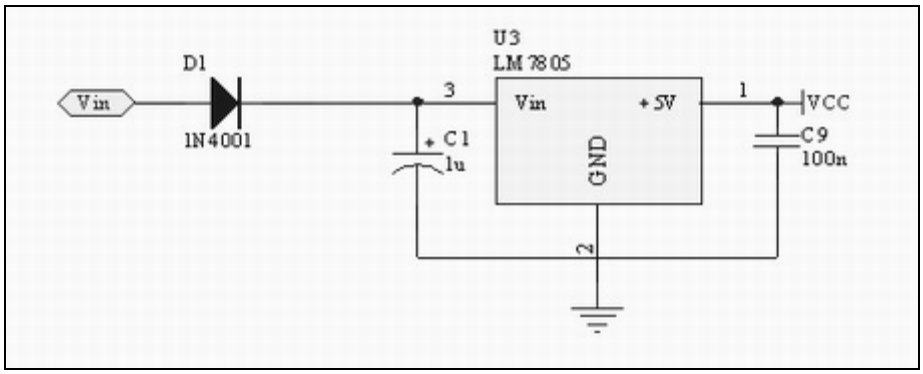
B. 2. Komunikasi Serial

Driver TTL-RS232 yang digunakan adalah MAX232. Komunikasi serial memakai kanal pertama (1) dari dua kanal yang tersedia. Kanal T1 IN terhubung ke pin TxD µC, sedangkan R1 OUT terhubung ke pin RxD µC. Lewat antarmuka ini TxD terhubung ke pin RD (*Received Data*) port COM PC, sedangkan RxD ke pin TD (*Transmitted Data*) port COM PC. Pin-pin *handshaking* berikut di-*loopback* pada sisi PC, RTS (*Request To Send*) dengan CTS (*Clear To Send*), DSR (*Data Set Ready*) dengan DTR (*Data Terminal Ready*) dan DCD (*Data Carrier Detect*).

Program µC yang digunakan mengatur port serialnya dalam mode 1 (8-bit UART). Transmisi tiap karakter data serial dimulai dengan sebuah *start bit* (*low*) dan diakhiri dengan sebuah *stop bit* (*high*).

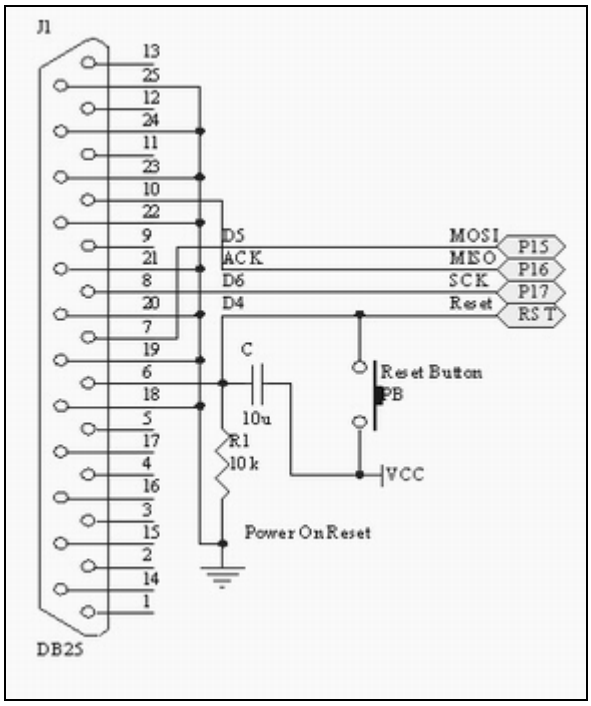
Program *ProAnz Serial Communication Raw Data I/O*[19] digunakan untuk mengirim dan menerima data serial dalam tampilan heksadesimal. Konfigurasi koneksi pada program adalah 19200 bps, *stop bit* '0', tanpa kontrol dan tanpa *parity* (yang lain diisi '0').

Kristal 11.059 MHz dipilih karena alasan *Baud rate*. Pengujian dimaksudkan untuk validasi kebenaran fungsional dari program AES-128 yang dibuat, sejumlah vektor uji akan dimasukkan lewat komunikasi serial. Dengan nilai 11.059 MHz tersebut dapat diperoleh nilai transmisi data 19200 bps [11]. Misalkan digunakan 12 MHz, untuk 2400 bps saja (dari PC) nilai sebenarnya adalah 2403 bps (dari μC). Jadi, nilai 11.059 MHz digunakan untuk memperoleh jumlah *cycle* yang tepat untuk UART 19200 bps.

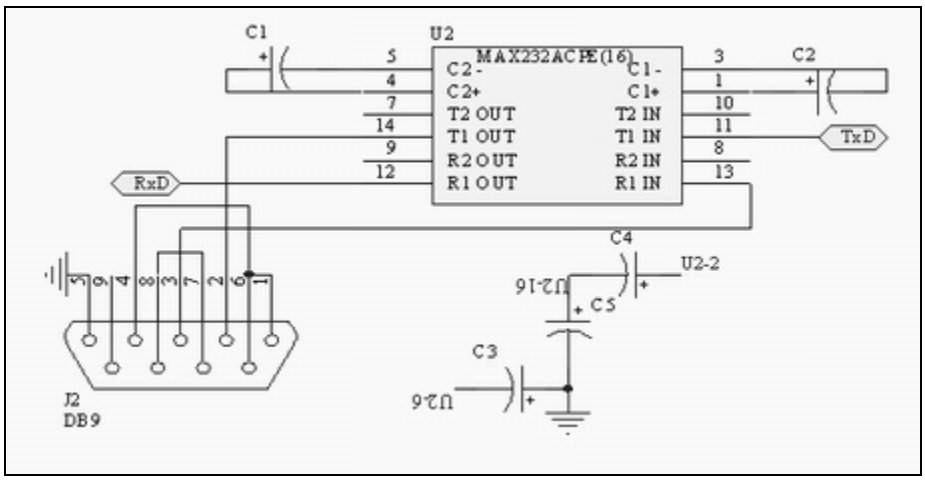


Gambar A-2. Skematik Regulator Tegangan.

[19] Pembuat program anonim (kur_whs@yahoo.com).



Gambar A-3. Skematik Koneksi ISP.



Gambar A-4. Skematik Antarmuka RS-232.

Lampiran B
Pengujian Kebenaran Fungsional

Dokumen FIPS-197 untuk standar AES-128 hanya menyediakan contoh vektor uji yang representatif untuk pengembangan implementasi.

Terpisah dari AES-128, NIST mengeluarkan *AES Algorithm Validation Suite* (AESAVS) bagi pembuat implementasi AES yang ingin memperoleh validasi **formal** (Federal AS). Implementasi yang divalidasi AESAVS akan diumumkan dan disertifikasi oleh NIST[20]. Namun, validasi resmi tersebut baru dapat diperoleh lewat *Cryptographic Module Validation Program* (CMVP)[21], sebuah catatan lagi, sertifikat tersebut bukan untuk masalah keamanan. Desain I/O yang tidak menangani aliran blok maupun *feedback* digolongkan sebagai mode *Electronic Code Book* (ECB)[22]. Berikut ini adalah vektor uji ECB yang dipakai untuk AES-128 (dari AESAVS).

B. 1. GFSBox Known Answer Test Values

`KEY=00000000000000000000000000000000`

plain text	*cipher text*
f34481ec3cc627bacd5dc3fb08f273e6	0336763e966d92595a567cc9ce537f5e
9798c4640bad75c7c3227db910174e72	a9a1631bf4996954ebc093957b234589
96ab5c2ff612d9dfaae8c31f30c42168	ff4f8391a6a40ca5b25d23bedd44a597
6a118a874519e64e9963798a503f1d35	dc43be40be0e53712f7e2bf5ca707209
cb9fceec81286ca3e989bd979b0cb284	92beedab1895a94faa69b632e5cc47ce
b26aeb1874e47ca8358ff22378f09144	459264f4798f6a78bacb89c15ed3d601
58c8e00b2631686d54eab84b91f0aca1	08a4e2efec8a8e3312ca7460b9040bbf

B. 2. KeySBox Known Answer Test Values

[20] "*AES Algorithm Validation List*," http://csrc.nist.gov/cryptval/aes/aesval.html
[21] http://csrc.nist.gov/cryptval/1401labs.htm
[22] SP800 – 38A: "*Recommendation for Block Cipher Modes of Operation*" http://csrc.nist.gov/CryptoToolkit/tkmodes.html

PT=00000000000000000000000000000000

key	*cipher text*
10a58869d74be5a374cf867cfb473859	6d251e6944b051e04eaa6fb4dbf78465
caea65cdbb75e9169ecd22ebe6e54675	6e29201190152df4ee058139def610bb
a2e2fa9baf7d20822ca9f0542f764a41	c3b44b95d9d2f25670eee9a0de099fa3
b6364ac4e1de1e285eaf144a2415f7a0	5d9b05578fc944b3cf1ccf0e746cd581
64cf9c7abc50b888af65f49d521944b2	f7efc89d5dba578104016ce5ad659c05
47d6742eefcc0465dc96355e851b64d9	0306194f666d183624aa230a8b264ae7
3eb39790678c56bee34bbcdeccf6cdb5	858075d536d79ccee571f7d7204b1f67
64110a924f0743d500ccadae72c13427	35870c6a57e9e92314bcb8087cde72ce
18d8126516f8a12ab1a36d9f04d68e51	6c68e9be5ec41e22c825b7c7affb4363
f530357968578480b398a3c251cd1093	f5df39990fc688f1b07224cc03e86cea
da84367f325d42d601b4326964802e8e	bba071bcb470f8f6586e5d3add18bc66
e37b1c6aa2846f6fdb413f238b089f23	43c9f7e62f5d288bb27aa40ef8fe1ea8
6c002b682483e0cabcc731c253be5674	3580d19cff44f1014a7c966a69059de5
143ae8ed6555aba96110ab58893a8ae1	806da864dd29d48deafbe764f8202aef
b69418a85332240dc82492353956ae0c	a303d940ded8f0baff6f75414cac5243
71b5c08a1993e1362e4d0ce9b22b78d5	c2dabd117f8a3ecabfbb11d12194d9d0
e234cdca2606b81f29408d5f6da21206	fff60a4740086b3b9c56195b98d91a7b
13237c49074a3da078dc1d828bb78c6f	8146a08e2357f0caa30ca8c94d1a0544
3071a2a48fe6cbd04f1a129098e308f8	4b98e06d356deb07ebb824e5713f7be3
90f42ec0f68385f2ffc5dfc03a654dce	7a20a53d460fc9ce0423a7a0764c6cf2
febd9a24d8b65c1c787d50a4ed3619a9	f4a70d8af877f9b02b4c40df57d45b17

B. 3. VarTxt Known Answer Test Values

KEY=00000000000000000000000000000000

plain text	*cipher text*
80000000000000000000000000000000	3ad78e726c1ec02b7ebfe92b23d9ec34
c0000000000000000000000000000000	aae5939c8efdf2f04e60b9fe7117b2c2
e0000000000000000000000000000000	f031d4d74f5dcbf39daaf8ca3af6e527
f0000000000000000000000000000000	96d9fd5cc4f07441727df0f33e401a36
f8000000000000000000000000000000	30ccdb044646d7e1f3ccea3dca08b8c0
fc000000000000000000000000000000	16ae4ce5042a67ee8e177b7c587ecc82
fe000000000000000000000000000000	b6da0bb11a23855d9c5cb1b4c6412e0a
ff000000000000000000000000000000	db4f1aa530967d6732ce4715eb0ee24b
ff800000000000000000000000000000	a81738252621dd180a34f3455b4baa2f
ffc00000000000000000000000000000	77e2b508db7fd89234caf7939ee5621a
ffe00000000000000000000000000000	b8499c251f8442ee13f0933b688fcd19
fff00000000000000000000000000000	965135f8a81f25c9d630b17502f68e53
fff80000000000000000000000000000	8b87145a01ad1c6cede995ea3670454f
fffc0000000000000000000000000000	8eae3b10a0c8ca6d1d3b0fa61e56b0b2
fffe0000000000000000000000000000	64b4d629810fda6bafdf08f3b0d8d2c5
ffff0000000000000000000000000000	d7e5dbd3324595f8fdc7d7c571da6c2a
ffff8000000000000000000000000000	f3f72375264e167fca9de2c1527d9606
ffffc000000000000000000000000000	8ee79dd4f401ff9b7ea945d86666c13b
ffffe000000000000000000000000000	dd35cea2799940b40db3f819cb94c08b

fffff000000000000000000000000000	6941cb6b3e08c2b7afa581ebdd607b87
fffff800000000000000000000000000	2c20f439f6bb097b29b8bd6d99aad799
fffffc000000000000000000000000000	625d01f058e565f77ae86378bd2c49b3
fffffe000000000000000000000000000	c0b5fd98190ef45fbb4301438d095950
ffffff000000000000000000000000000	13001ff5d99806efd25da34f56be854b
ffffff800000000000000000000000000	3b594c60f5c8277a5113677f94208d82
ffffffc000000000000000000000000000	e9c0fc1818e4aa46bd2e39d638f89e05
ffffffe000000000000000000000000000	f8023ee9c3fdc45a019b4e985c7e1a54
fffffff000000000000000000000000000	35f40182ab4662f3023baec1ee796b57
fffffff800000000000000000000000000	3aebbad7303649b4194a6945c6cc3694
fffffffc000000000000000000000000000	a2124bea53ec2834279bed7f7eb0f938
fffffffe000000000000000000000000000	b9fb4399fa4facc7309e14ec98360b0a
ffffffff000000000000000000000000000	c26277437420c5d634f715aea81a9132
ffffffff800000000000000000000000000	171a0e1b2dd424f0e089af2c4c10f32f
ffffffffc000000000000000000000000000	7cadbe402d1b208fe735edce00aee7ce
ffffffffe000000000000000000000000000	43b02ff929a1485af6f5c6d6558baa0f
fffffffff000000000000000000000000000	092faacc9bf43508bf8fa8613ca75dea
fffffffff800000000000000000000000000	cb2bf8280f3f9742c7ed513fe802629c
fffffffffc000000000000000000000000000	215a41ee442fa992a6e323986ded3f68
fffffffffe000000000000000000000000000	f21e99cf4f0f77cea836e11a2fe75fb1
ffffffffff000000000000000000000000000	95e3a0ca9079e646331df8b4e70d2cd6
ffffffffff800000000000000000000000000	4afe7f120ce7613f74fc12a01a828073
ffffffffffc000000000000000000000000000	827f000e75e2c8b9d479beed913fe678
ffffffffffe000000000000000000000000000	35830c8e7aaefe2d30310ef381cbf691
fffffffffff000000000000000000000000000	191aa0f2c8570144f38657ea4085ebe5
fffffffffff800000000000000000000000000	85062c2c909f15d9269b6c18ce99c4f0
fffffffffffc000000000000000000000000000	678034dc9e41b5a560ed239eeab1bc78
fffffffffffe000000000000000000000000000	c2f93a4ce5ab6d5d56f1b93cf19911c1
ffffffffffff000000000000000000000000000	1c3112bcb0c1dcc749d799743691bf82
ffffffffffff800000000000000000000000000	00c55bd75c7f9c881989d3ec1911c0d4
fffffffffffc000000000000000000000000000	ea2e6b5ef182b7dff3629abd6a12045f
fffffffffffe000000000000000000000000000	22322327e01780b17397f24087f8cc6f
ffffffffffff000000000000000000000000000	c9cacb5cd11692c373b2411768149ee7
ffffffffffff800000000000000000000000000	a18e3dbbca577860dab6b80da3139256
fffffffffffc000000000000000000000000000	79b61c37bf328ecca8d743265a3d425c
fffffffffffe000000000000000000000000000	d2d99c6bcc1f06fda8e27e8ae3f1ccc7
ffffffffffffff00000000000000000000000000	1bfd4b91c701fd6b61b7f997829d663b
ffffffffffffff80000000000000000000000000	11005d52f25f16bdc9545a876a63490a
fffffffffffffc0000000000000000000000000	3a4d354f02bb5a5e47d39666867f246a
fffffffffffffe0000000000000000000000000	d451b8d6e1e1a0ebb155fbbf6e7b7dc3
ffffffffffffff0000000000000000000000000	6898d4f42fa7ba6a10ac05e87b9f2080
ffffffffffffff8000000000000000000000000	b611295e739ca7d9b50f8e4c0e754a3f
fffffffffffffffc000000000000000000000000	7d33fc7d8abe3ca1936759f8f5deaf20
fffffffffffffffe000000000000000000000000	3b5e0f566dc96c298f0c12637539b25c
ffffffffffffffff000000000000000000000000	f807c3e7985fe0f5a50e2cdb25c5109e
ffffffffffffffff800000000000000000000000	41f992a856fb278b389a62f5d274d7e9
fffffffffffffffc000000000000000000000000	10d3ed7a6fe15ab4d91acbc7d0767ab1
fffffffffffffffe000000000000000000000000	21feecd45b2e675973ac33bf0c5424fc
ffffffffffffffff000000000000000000000000	1480cb3955ba62d09eea668f7c708817
ffffffffffffffff800000000000000000000000	66404033d6b72b609354d5496e7eb511

fffffffffffffffc00000000000000	1c317a220a7d700da2b1e075b00266e1
fffffffffffffffe00000000000000	ab3b89542233f1271bf8fd0c0f403545
ffffffffffffffff00000000000000	d93eae966fac46dca927d6b114fa3f9e
ffffffffffffffff80000000000000	1bdec521316503d9d5ee65df3ea94ddf
fffffffffffffffc0000000000000	eef456431dea8b4acf83bdae3717f75f
fffffffffffffffe0000000000000	06f2519a2fafaa596bfef5cfa15c21b9
ffffffffffffffff0000000000000	251a7eac7e2fe809e4aa8d0d7012531a
ffffffffffffffff8000000000000	3bffc16e4c49b268a20f8d96a60b4058
fffffffffffffffc000000000000	e886f9281999c5bb3b3e8862e2f7c988
fffffffffffffffe000000000000	563bf90d61beef39f48dd625fcef1361
ffffffffffffffff000000000000	4d37c850644563c69fd0acd9a049325b
ffffffffffffffff800000000000	b87c921b91829ef3b13ca541ee1130a6
fffffffffffffffc00000000000	2e65eb6b6ea383e109accce8326b0393
fffffffffffffffe00000000000	9ca547f7439edc3e255c0f4d49aa8990
ffffffffffffffff00000000000	a5e652614c9300f37816b1f9fd0c87f9
ffffffffffffffff80000000000	14954f0b4697776f44494fe458d814ed
fffffffffffffffc0000000000	7c8d9ab6c2761723fe42f8bb506cbcf7
fffffffffffffffe0000000000	db7e1932679fdd99742aab04aa0d5a80
ffffffffffffffff0000000000	4c6a1c83e568cd10f27c2d73ded19c28
ffffffffffffffff8000000000	90ecbe6177e674c98de412413f7ac915
fffffffffffffffc000000000	90684a2ac55fe1ec2b8ebd5622520b73
fffffffffffffffe000000000	7472f9a7988607ca79707795991035e6
ffffffffffffffff000000000	56aff089878bf3352f8df172a3ae47d8
ffffffffffffffff800000000	65c0526cbe40161b8019a2a3171abd23
fffffffffffffffc00000000	377be0be33b4e3e310b4aabda173f84f
fffffffffffffffe00000000	9402e9aa6f69de6504da8d20c4fcaa2f
ffffffffffffffff00000000	123c1f4af313ad8c2ce648b2e71fb6e1
ffffffffffffffff80000000	1ffc626d30203dcdb0019fb80f726cf4
fffffffffffffffc0000000	76da1fbe3a50728c50fd2e621b5ad885
fffffffffffffffe0000000	082eb8be35f442fb52668e16a591d1d6
ffffffffffffffff0000000	e656f9ecf5fe27ec3e4a73d00c282fb3
ffffffffffffffff8000000	2ca8209d63274cd9a29bb74bcd77683a
fffffffffffffffc000000	79bf5dce14bb7dd73a8e3611de7ce026
fffffffffffffffe000000	3c849939a5d29399f344c4a0eca8a576
ffffffffffffffff000000	ed3c0a94d59bece98835da7aa4f07ca2
ffffffffffffffff800000	63919ed4ce10196438b6ad09d99cd795
fffffffffffffffc00000	7678f3a833f19fea95f3c6029e2bc610
fffffffffffffffe00000	3aa426831067d36b92be7c5f81c13c56
ffffffffffffffff00000	9272e2d2cdd11050998c845077a30ea0
ffffffffffffffff80000	088c4b53f5ec0ff814c19adae7f6246c
fffffffffffffffc0000	4010a5e401fdf0a0354ddbcc0d012b17
fffffffffffffffe0000	a87a385736c0a6189bd6589bd8445a93
ffffffffffffffff0000	545f2b83d9616dccf60fa9830e9cd287
ffffffffffffffff8000	4b706f7f92406352394037a6d4f4688d
fffffffffffffffc000	b7972b3941c44b90afa7b264bfba7387
fffffffffffffffe000	6f45732cf10881546f0fd23896d2bb60
ffffffffffffffff000	2e3579ca15af27f64b3c955a5bfc30ba
ffffffffffffffff800	34a2c5a91ae2aec99b7d1b5fa6780447
fffffffffffffffc00	a4d6616bd04f87335b0e53351227a9ee
fffffffffffffffe00	7f692b03945867d16179a8cefc83ea3f

ffffffffffffffffffffffffffffff00	3bd141ee84a0e6414a26e7a4f281f8a2
ffffffffffffffffffffffffffffff80	d1788f572d98b2b16ec5d5f3922b99bc
ffffffffffffffffffffffffffffffc0	0833ff6f61d98a57b288e8c3586b85a6
ffffffffffffffffffffffffffffffe0	8568261797de176bf0b43becc6285afb
fffffffffffffffffffffffffffffff0	f9b0fda0c4a898f5b9e6f661c4ce4d07
fffffffffffffffffffffffffffffff8	8ade895913685c67c5269f8aae42983e
fffffffffffffffffffffffffffffffc	39bde67d5c8ed8a8b1c37eb8fa9f5ac0
fffffffffffffffffffffffffffffffe	5c005e72c1418c44f569f2ea33ba54f3
ffffffffffffffffffffffffffffffff	3f5b8cc9ea855a0afa7347d23e8d664e

B. 4. VarKey Known Answer Test Values

PT=00000000000000000000000000000000

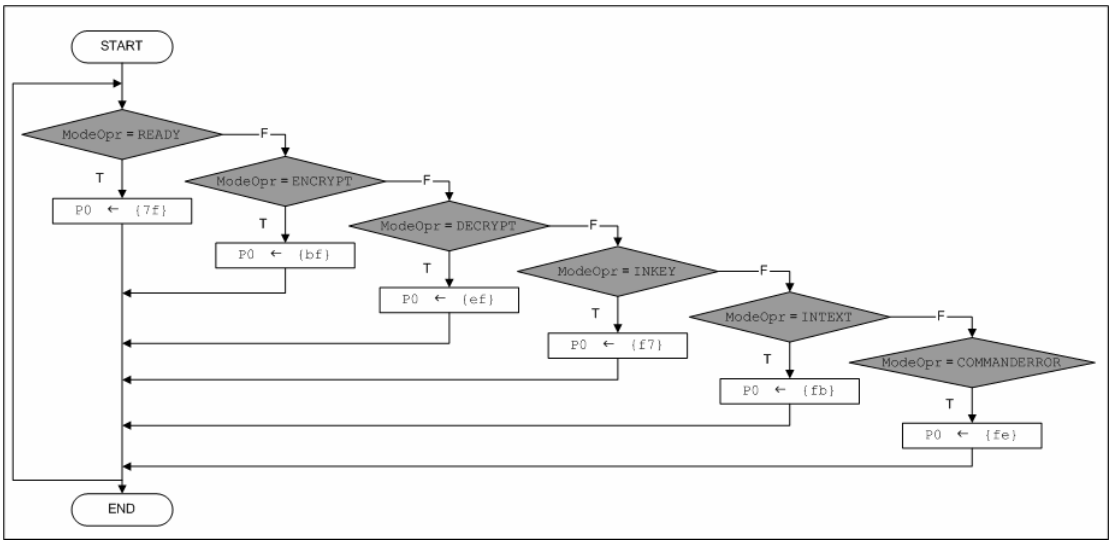
key	*cipher text*
80000000000000000000000000000000	0edd33d3c621e546455bd8ba1418bec8
c0000000000000000000000000000000	4bc3f883450c113c64ca42e1112a9e87
e0000000000000000000000000000000	72a1da770f5d7ac4c9ef94d822affd97
f0000000000000000000000000000000	970014d634e2b7650777e8e84d03ccd8
f8000000000000000000000000000000	f17e79aed0db7e279e955b5f493875a7
fc000000000000000000000000000000	9ed5a75136a940d0963da379db4af26a
fe000000000000000000000000000000	c4295f83465c7755e8fa364bac6a7ea5
ff000000000000000000000000000000	b1d758256b28fd850ad4944208cf1155
ff800000000000000000000000000000	42ffb34c743de4d88ca38011c990890b
ffc00000000000000000000000000000	9958f0ecea8b2172c0c1995f9182c0f3
ffe00000000000000000000000000000	956d7798fac20f82a8823f984d06f7f5
fff00000000000000000000000000000	a01bf44f2d16be928ca44aaf7b9b106b
fff80000000000000000000000000000	b5f1a33e50d40d103764c76bd4c6b6f8
fffc0000000000000000000000000000	2637050c9fc0d4817e2d69de878aee8d
fffe0000000000000000000000000000	113ecbe4a453269a0dd26069467fb5b5
ffff0000000000000000000000000000	97d0754fe68f11b9e375d070a608c884
ffff8000000000000000000000000000	c6a0b3e998d05068a5399778405200b4
ffffc000000000000000000000000000	df556a33438db87bc41b1752c55e5e49
ffffe000000000000000000000000000	90fb128d3a1af6e548521bb962bf1f05
fffff000000000000000000000000000	26298e9c1db517c215fadfb7d2a8d691
fffff800000000000000000000000000	a6cb761d61f8292d0df393a279ad0380
fffffc000000000000000000000000000	12acd89b13cd5f8726e34d44fd486108
fffffe000000000000000000000000000	95b1703fc57ba09fe0c3580febdd7ed4
ffffff000000000000000000000000000	de11722d893e9f9121c381becc1da59a
ffffff800000000000000000000000000	6d114ccb27bf391012e8974c546d9bf2
ffffffc000000000000000000000000000	5ce37e17eb4646ecfac29b9cc38d9340
ffffffe000000000000000000000000000	18c1b6e2157122056d0243d8a165cddb
fffffff000000000000000000000000000	99693e6a59d1366c74d823562d7e1431
fffffff800000000000000000000000000	6c7c64dc84a8bba758ed17eb025a57e3
fffffffc000000000000000000000000000	e17bc79f30eaab2fac2cbbe3458d687a
fffffffe000000000000000000000000000	1114bc2028009b923f0b01915ce5e7c4
ffffffff000000000000000000000000000	9c28524a16a1e1c1452971caa8d13476

ffffffff800000000000000000000000	ed62e16363638360fdd6ad62112794f0
fffffffc000000000000000000000000	5a8688f0b2a2c16224c161658ffd4044
fffffffe000000000000000000000000	23f710842b9bb9c32f26648c786807ca
ffffffff000000000000000000000000	44a98bf11e163f632c47ec6a49683a89
ffffffff800000000000000000000000	0f18aff94274696d9b61848bd50ac5e5
fffffffc000000000000000000000000	82408571c3e2424540207f833b6dda69
fffffffe000000000000000000000000	303ff996947f0c7d1f43c8f3027b9b75
fffffffff000000000000000000000000	7df4daf4ad29a3615a9b6ece5c99518a
fffffffff800000000000000000000000	c72954a48d0774db0b4971c526260415
fffffffffc000000000000000000000000	1df9b76112dc6531e07d2cfda04411f0
fffffffffe000000000000000000000000	8e4d8e699119e1fc87545a647fb1d34f
ffffffffff000000000000000000000000	e6c4807ae11f36f091c57d9fb68548d1
ffffffffff800000000000000000000000	8ebf73aad49c82007f77a5c1ccec6ab4
ffffffffffc000000000000000000000000	4fb288cc2040049001d2c7585ad123fc
ffffffffffe000000000000000000000000	04497110efb9dceb13e2b13fb4465564
fffffffffff000000000000000000000000	75550e6cb5a88e49634c9ab69eda0430
fffffffffff800000000000000000000000	b6768473ce9843ea66a81405dd50b345
fffffffffffc000000000000000000000000	cb2f430383f9084e03a653571e065de6
fffffffffffe000000000000000000000000	ff4e66c07bae3e79fb7d210847a3b0ba
ffffffffffff000000000000000000000000	7b90785125505fad59b13c186dd66ce3
ffffffffffff800000000000000000000000	8b527a6aebdaec9eaef8eda2cb7783e5
ffffffffffffc000000000000000000000000	43fdaf53ebbc9880c228617d6a9b548b
ffffffffffffe000000000000000000000000	53786104b9744b98f052c46f1c850d0b
ffffffffffffff000000000000000000000000	b5ab3013dd1e61df06cbaf34ca2aee78
ffffffffffffff800000000000000000000000	7470469be9723030fdcc73a8cd4fbb10
ffffffffffffffc000000000000000000000000	a35a63f5343ebe9ef8167bcb48ad122e
ffffffffffffffe000000000000000000000000	fd8687f0757a210e9fdf181204c30863
fffffffffffffff000000000000000000000000	7a181e84bd5457d26a88fbae96018fb0
fffffffffffffff800000000000000000000000	653317b9362b6f9b9e1a580e68d494b5
ffffffffffffffffc00000000000000000000000	995c9dc0b689f03c45867b5faa5c18d1
ffffffffffffffffe00000000000000000000000	77a4d96d56dda398b9aabecfc75729fd
fffffffffffffffff00000000000000000000000	84be19e053635f09f2665e7bae85b42d
fffffffffffffffff80000000000000000000000	32cd652842926aea4aa6137bb2be2b5e
ffffffffffffffffffc0000000000000000000000	493d4a4f38ebb337d10aa84e9171a554
ffffffffffffffffffe0000000000000000000000	d9bff7ff454b0ec5a4a2a69566e2cb84
fffffffffffffffffff0000000000000000000000	3535d565ace3f31eb249ba2cc6765d7a
fffffffffffffffffff8000000000000000000000	f60e91fc3269eecf3231c6e9945697c6
ffffffffffffffffffffc000000000000000000000	ab69cfadf51f8e604d9cc37182f6635a
ffffffffffffffffffffe000000000000000000000	7866373f24a0b6ed56e0d96fcdafb877
fffffffffffffffffffff000000000000000000000	1ea448c2aac954f5d812e9d78494446a
fffffffffffffffffffff800000000000000000000	acc5599dd8ac02239a0fef4a36dd1668
ffffffffffffffffffffffc0000000000000000000	d8764468bb103828cf7e1473ce895073
ffffffffffffffffffffffe0000000000000000000	1b0d02893683b9f180458e4aa6b73982
fffffffffffffffffffffff0000000000000000000	96d9b017d302df410a937dcdb8bb6e43
fffffffffffffffffffffff8000000000000000000	ef1623cc44313cff440b1594a7e21cc6
ffffffffffffffffffffffffc000000000000000000	284ca2fa35807b8b0ae4d19e11d7dbd7
ffffffffffffffffffffffffe000000000000000000	f2e976875755f9401d54f36e2a23a594
fffffffffffffffffffffffff000000000000000000	ec198a18e10e532403b7e20887c8dd80
fffffffffffffffffffffffff800000000000000000	545d50ebd919e4a6949d96ad47e46a80
ffffffffffffffffffffffffffc00000000000000000	dbdfb527060e0a71009c7bb0c68f1d44

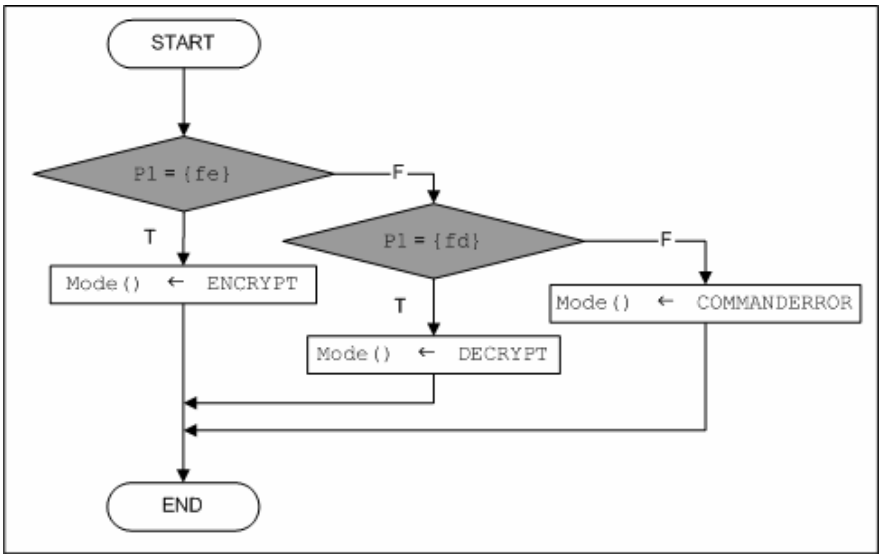
fffffffffffffffffffe00000000000	9cfa1322ea33da2173a024f2ff0d896d
fffffffffffffffffffff00000000000	8785b1a75b0f3bd958dcd0e29318c521
fffffffffffffffffffff80000000000	38f67b9e98e4a97b6df030a9fcdd0104
fffffffffffffffffffffc0000000000	192afffb2c880e82b05926d0fc6c448b
fffffffffffffffffffffe0000000000	6a7980ce7b105cf530952d74daaf798c
fffffffffffffffffffffff0000000000	ea3695e1351b9d6858bd958cf513ef6c
fffffffffffffffffffffff8000000000	6da0490ba0ba0343b935681d2cce5ba1
fffffffffffffffffffffffc000000000	f0ea23af08534011c60009ab29ada2f1
fffffffffffffffffffffffe000000000	ff13806cf19cc38721554d7c0fcdcd4b
fffffffffffffffffffffffff000000000	6838af1f4f69bae9d85dd188dcdf0688
fffffffffffffffffffffffff800000000	36cf44c92d550bfb1ed28ef583ddf5d7
fffffffffffffffffffffffffc00000000	d06e3195b5376f109d5c4ec6c5d62ced
fffffffffffffffffffffffffe00000000	c440de014d3d610707279b13242a5c36
fffffffffffffffffffffffffff00000000	f0c5c6ffa5e0bd3a94c88f6b6f7c16b9
fffffffffffffffffffffffffff80000000	3e40c3901cd7effc22bffc35dee0b4d9
fffffffffffffffffffffffffffc0000000	b63305c72bedfab97382c406d0c49bc6
fffffffffffffffffffffffffffe0000000	36bbaab22a6bd4925a99a2b408d2dbae
fffffffffffffffffffffffffffff0000000	307c5b8fcd0533ab98bc51e27a6ce461
fffffffffffffffffffffffffffff8000000	829c04ff4c07513c0b3ef05c03e337b5
fffffffffffffffffffffffffffffc000000	f17af0e895dda5eb98efc68066e84c54
fffffffffffffffffffffffffffffe000000	277167f3812afff1ffacb4a934379fc3
fffffffffffffffffffffffffffffff000000	2cb1dc3a9c72972e425ae2ef3eb597cd
fffffffffffffffffffffffffffffff800000	36aeaa3a213e968d4b5b679d3a2c97fe
fffffffffffffffffffffffffffffffc00000	9241daca4fdd034a82372db50e1a0f3f
fffffffffffffffffffffffffffffffe00000	c14574d9cd00cf2b5a7f77e53cd57885
fffffffffffffffffffffffffffffffff00000	793de39236570aba83ab9b737cb521c9
fffffffffffffffffffffffffffffffff80000	16591c0f27d60e29b85a96c33861a7ef
fffffffffffffffffffffffffffffffffc0000	44fb5c4d4f5cb79be5c174a3b1c97348
fffffffffffffffffffffffffffffffffe0000	674d2b61633d162be59dde04222f4740
fffffffffffffffffffffffffffffffffff0000	b4750ff263a65e1f9e924ccfd98f3e37
fffffffffffffffffffffffffffffffffff8000	62d0662d6eaeddedebae7f7ea3a4f6b6
fffffffffffffffffffffffffffffffffffc000	70c46bb30692be657f7eaa93ebad9897
fffffffffffffffffffffffffffffffffffe000	323994cfb9da285a5d9642e1759b224a
fffffffffffffffffffffffffffffffffffff000	1dbf57877b7b17385c85d0b54851e371
fffffffffffffffffffffffffffffffffffff800	dfa5c097cdc1532ac071d57b1d28d1bd
fffffffffffffffffffffffffffffffffffffc00	3a0c53fa37311fc10bd2a9981f513174
fffffffffffffffffffffffffffffffffffffe00	ba4f970c0a25c41814bdae2e506be3b4
fffffffffffffffffffffffffffffffffffffff00	2dce3acb727cd13ccd76d425ea56e4f6
fffffffffffffffffffffffffffffffffffffff80	5160474d504b9b3eefb68d35f245f4b3
fffffffffffffffffffffffffffffffffffffffc0	41a8a947766635dec37553d9a6c0cbb7
fffffffffffffffffffffffffffffffffffffffe0	25d6cfe6881f2bf497dd14cd4ddf445b
fffffffffffffffffffffffffffffffffffffffff0	41c78c135ed9e98c096640647265da1e
fffffffffffffffffffffffffffffffffffffffff8	5a4d404d8917e353e92a21072c3b2305
fffffffffffffffffffffffffffffffffffffffffc	02bc96846b3fdc71643f384cd3cc3eaf
fffffffffffffffffffffffffffffffffffffffffe	9ba4a9143f4e5d4048521c4f8877d88e
ffffffffffffffffffffffffffffffffffffffffff	a1f6258c877d5fcd8964484538bfc92c

Lampiran C

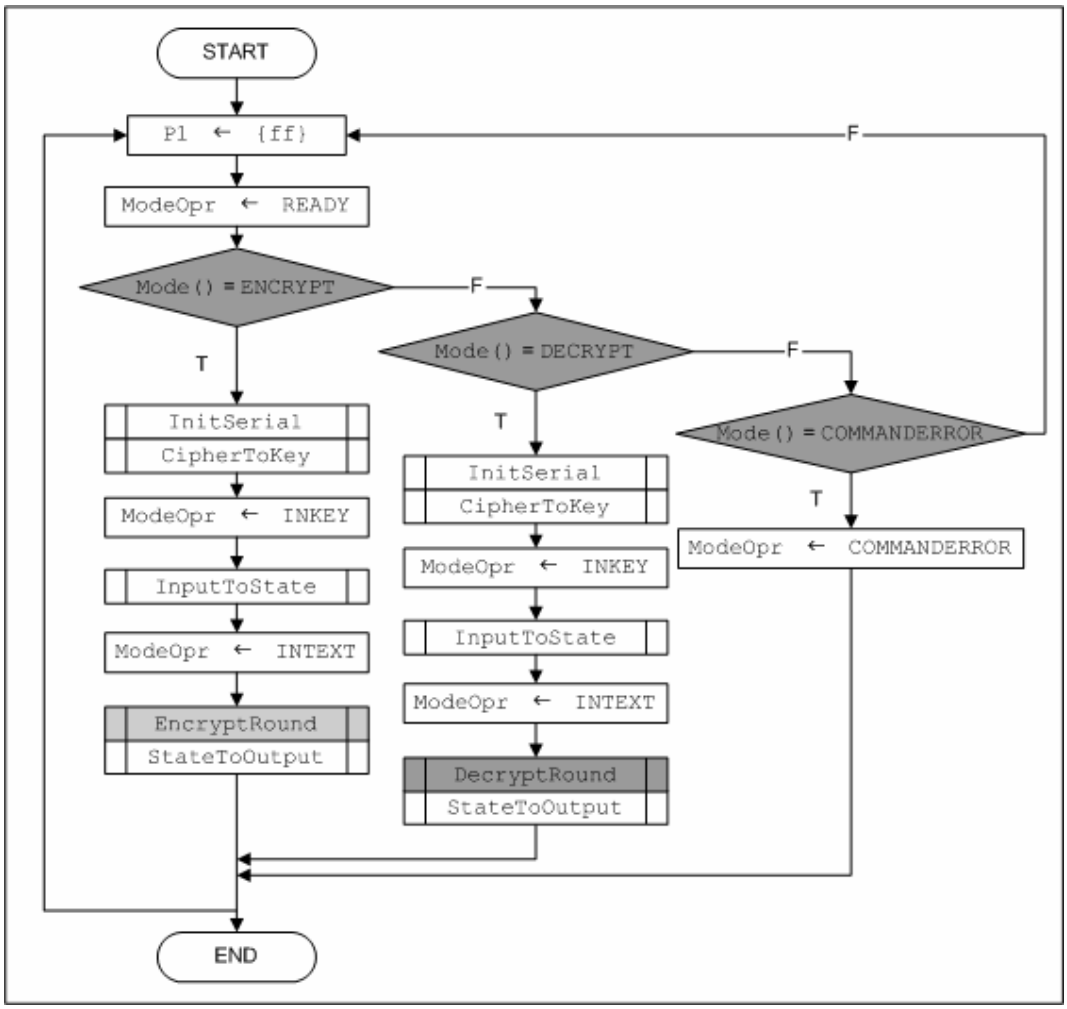
Diagram Alir Kontrol, Status, dan I/O



Gambar C-1. Diagram Alir Fungsi `Status()`.



Gambar C-2. Diagram Alir Fungsi `Mode()`.



Gambar C-3. Diagram Alir Fungsi `Main()`.

Lampiran D

Listing Program Mikrokontroler

```
//Implementasi Algoritma AES-128 pada Mikrokontroler 8051
//Arif Kusbandono      13299108
//VLSI-RG Dept. Teknik Elektro ITB
//Juni 2004

//Modul  : Definition.h


//definisi BYTE
typedef unsigned char   BYTE;
```

```
//Implementasi Algoritma AES-128 pada Mikrokontroler 8051
//Arif Kusbandono      13299108
//VLSI-RG Dept. Teknik Elektro ITB
//Juni 2004

//Modul  : Main.c
//         Modul berisi fungsi yang mengatur pembacaan mode komputasi
//         dari P1, mengatur status yang di-display ke P0, mengatur I/O,
//         dan komputasi yang dilakukan (enkripsi/dekripsi)

#pragma code
//Pragma Source dipakai untuk generate listing assembly
//dari modul C yang bersangkutan
//#pragma src  //pragma ini tidak boleh aktif saat Build Project
#include <stdio.h>
//file include register AT89X52 varian dari generik 8052
//dipakai khusus
//#include <AT89X52.H>
#include <REG51.H>            //file include register generik 8051

#include "Definition.h"              //file definisi tipe variabel
#include "LookUpTable.dat"    //look up table yang digunakan

//deklarasi fungsi-fungsi yang ada dalam modul lain
extern void EncryptRound();
extern void DecryptRound();

extern void InitSerial();
extern void InputToState();
extern void CipherToKey();
extern void StateToOutput();

void Status(BYTE ModeOpr);
BYTE Mode();

//direktif _at_ mengalokasikan variabel pada alamat absolut
BYTE state[4][4] _at_ 0x08;

//segmen relocatable
BYTE key[2][4][4];
BYTE m;

//definisi nilai untuk argumen Status() dan balikan Mode()
```

```
#define   READY 0
#define   ENCRYPT       1
#define   DECRYPT       2
#define   INKEY 3
#define   INTEXT        4
#define   COMMANDERROR 5

void main()
{
          while (1)     //super loop
          {
            P1 = 0xFF;
            Status(READY);
            Mode();
            if  (Mode() == ENCRYPT)
                {
                        Status(ENCRYPT);       //sekuen I/O enkripsi
                        InitSerial();
                        CipherToKey();
                        Status(INKEY);
                        InputToState();
                        Status(INTEXT);
                        EncryptRound();
                        StateToOutput();
                        P1 = 0xFF;
                }
            if  (Mode() == DECRYPT)
                {
                        Status(DECRYPT);       //sekuen I/O dekripsi
                        InitSerial();
                        CipherToKey();
                        Status(INKEY);
                        InputToState();
                        Status(INTEXT);
                        DecryptRound();
                        StateToOutput();
                        P1 = 0xFF;
                }
            if  (Mode() == COMMANDERROR)
                {
                        Status(COMMANDERROR);
                        P1 = 0xFF;
                }
          }
}

void Status(BYTE ModeOpr)
{
          switch        (ModeOpr)
          {
            case READY:
            {
                P0 = 0x7F;     //mengirimkan status (diemulasi display LED)
                break;
            }
            case ENCRYPT:
            {
                P0 = 0xBF;
                break;
            }
            case DECRYPT:
            {
                P0 = 0xEF;
                break;
            }
            case INKEY:
            {
                P0 = 0xF7;
```