# SIMULATOR EKSEKUSI INSTRUKSI PADA SISTEM KOMPUTER VIRTUAL

**Soetrisno**
Teknik Informatika,Universitas Pelita Harapan
sutrisno.fik@uph.edu

**ABSTRAK**

Sistem komputer modern saat ini semakin canggih dan mudah digunakan karena kompleksitas perangkat keras, dan piranti lunak, semakin tersembunyi dari pengguna. Sehingga, untuk memahami bagaimana sebuah komputer melakukan eksekusi instruksi yang dilakukan oleh mikroprosesor tidak secara mudah dilakukan. Salah satu solusinya adalah dalam bentuk piranti lunak simulator, atau emulator, suatu mikroprosesor tertentu. Penulis merancang dan membuat piranti lunak sistem komputer virtual, terdiri dari mikroprosesor virtual, dan memori virtual. Mikroprosesor virtual dirancang dan dibuat mulai dari rancangan instruction set, dengan format ukuran instruksi yang seragam 32 bits, menyediakan 16 register berukuran 32 bits. Memori virtual yang dibuat pada simulator sebesar 32 kilo bytes. Sistem komputer yang dibuat memungkinkan lebih dari satu program berada di memori, dan mikroprosesor dapat menjalankan program-program yang ada di memori tersebut, meskipun masih dieksekusi secara sekuensial. Satu program dikerjakan hingga selesai, dilanjutkan program berikutnya. Dalam simulator ini, program dapat ditulis menggunakan bahasa mesin berdasarkan rancangan instruction set yang ada, atau program ditulis berupa pseudocode yang akan dikompilasi oleh piranti lunak compiler yang telah dibuat. Menggunakan compiler ini dapat diperlihatkan hasil translasi dari pseudocode ke kode dalam bahasa assembly, juga dalam bahasa mesin, dan serta status register, maupun memori, dari tiap instruksi yang dieksekusi.

**Keyword :** *mikroprosesor, instruction set, compiler, sistem komputer, komputer virtual*

## 1. PENDAHULUAN

Komputer modern saat ini seperti *desktop*, *laptop*, hingga gawai (*gadget*), semakin canggih dan mudah digunakan, dan diupayakan handal (*reliable*). Upaya menjadikan sebuah komputer handal, dan implikasinya adalah pemakai dihindarkan dari kerumitan akses ke perangkat keras secara langsung, dan pada tingkatan tertentu juga pada pirani lunak juga, misalnya pada sistem operasi. Kemudahan pemakaian dan penghindaran kerumitan pada kompleksitas perangkat keras, maupun piranti lunak, sangat membantu pemakai untuk fokus pada apa piranti lunak aplikasi yang dibutuhkannya.

Pemakai komputer dalam hal hususnya akses ke perangkat keras, seperti prosesor, semakin tidak mudah dilakukan. Pertimbangannya karena untuk menjamin kehandalan, maka perlu ada perlindungan dengan pelapisan proteksi (*shield*) dari kemungkinan terganggunya sistem karena akses yang tidak tepat. Salah satu contoh, meskipun pada sistem operasi Windows, masih disediakan akses ke prosesor melalui utility "Debug" yang diakses melalui *command line*, namun hanya diberikan akses terbatas. Pemanggilan Interrupt tertentu sudah tidak dapat dilakukan, karena diproteksi.

Hal ini menjadi kendala khsususnya untuk keperluan mendukung proses belajar mengajar, seperti pada mata kuliah Sistem Operasi ataupun mata kuliah Organisasi dan Arsitektur Komputer. Pada dua mata kuliah tersebut, masih diperlukan untuk mengenalkan beberapa hal mengenai perangkat keras khususnya prosesor. Beberapa hal tersebut diantaranya *register*, *program counter*, *addressing*, *big* dan *little endian data*. Pada program studi tertentu, topik-topik ini tidak menjadi kendala, tetapi pada program studi lainnya, dapat dikatakan sesuatu yang asing.

Simulator, atau emulator, merupakan suatu jalan keluar, akses ke perangkat keras dalam benuk piranti lunak. Sejumlah simulator, dan emulator, telah banyak dibuat, bahkan tidak saja pada tingkatan mikroprosesor, bahkan hingga ke sistem operasi. Simulator mikroprosesor salah satu diantaranya adalah SPIM yang dikembangkan untuk mensimulasikan prosesor MIPS, seperti yang dibahas pada bukunya Patterson dan Hennessy [1]. Emulator untuk prosesor Intel, khususnya prosesor 8086 cukup banyak tersedia di internet.

Secara umum, pada emulator 8086, memungkinkan kita memelajari mikroprosesor dan akses ke prosesor 8086, dengan membuat program *assembly*. Kita dapat memelajari di antaranya akses sejumlah *interrupt*, *register*, dan lainnya. Emulator ini sangat menarik, dan bermanfaat dalam mendukung proses belajar.

Penulis mengembangkan simulator komputer *virtual* terdiri dari mikroprosesor *virtual*, dan memori *virtual*. Istilah "*virtual*" yang dimaksudkan penulis adalah bukan akses ke sumber daya perangkat keras mikroprosesor (yaitu Intel prosesor, AMD, dan lainnya), ataupun riil perangkat keras memori (RAM, DDR3, atau DDR4). Komputer *virtual* yaitu sebuah komputer dalam bentuk piranti lunak yang mana program (dalam bentuk *machine code*) akan dieksekusi di atas prosesor *virtual*, dan memanfaatkan memori *virtual* juga. Pemakai berinteraksi ke komputer *virtual* ini. Komputer *virtual* dikembangkan menggunakan bahasa pemrograman Java.

Tujuan dibuatnya komputer virtual ini, untuk membantu mahasiswa dalam proses belajar mata kuliah yang telah disebutkan di atas. Khususnya yang ingin dicapai adalah membantu mahasiswa untuk memahami bagaimana suatu mikroprosesor mengeksekusi instruksi berdasarkan *machine code*, dan memahami konversi atau translasi perintah bahasa *assembly* ke *machine code*. Oleh karenanya, pada simulator yang dibuat, mikroprosesor *virtual* perlu dirancang himpunan atau kumpulan instruksi (*instruction set*) yang sederhana diharapkan mudah dipahami, sehingga mendapatkan esensinya.

Sebagaimana diketahui ada dua pola rancangan instruksi yaitu CISC (*Complex Instruction Set Code*), dan RISC (*Reduced Instruction Set Code*). Prosesor Intel dirancang menggunakan pendekatan CISC [2][3]. Pada CISC salah satu cirinya adalah ukuran panjang instruksi bervariasi. Sedangkan pada RISC biasanya menggunakan ukuran panjang instruksi yang seragam atau sama, seperti pada prosesor MIPS [1]. Untuk keperluan mendukung tujuan penelitian yang dilakukan, maka mikroprosesor yang dibuat menggunakan ukuran yang sama untuk setiap instruksinya yaitu 32 bits (4 bytes).

Setiap instruksi yang berukuran 32 bits tersebut, dibagi ke beberapa *fields*, tergantung operasi apa yang diterapkan padanya. Satu *field* yang pasti ada adalah kode operasi (*operation code*, atau disingkat *opcode*) sepanjang 8 bits. Penamaan operasi berupa *mnemonic* yang dapat digunakan unuk menuliskan kode *assembly*. Penamaan instruksi terinspirasi dan mengadopsi penamaan pada prosesor Intel dan MIPS [1][2][3].

Himpunan instruksi yang disediakan, dirancang sedikitnya dapat merepresentasikan elemen instruksi dasar dari suatu pemrogram, yaitu sekuens, pencabangan, repetisi menggunakan pencabangan, operasi aritmatika, akses data ke dan dari register dan memori, operasi dasar *stack* yaitu *push* dan *pop*, pemanggilan modul, meskipun ini sesungguhnya dapat dilakukan menggunakan kombinasi *conditional* dan *unconditional jump*.

Himpunan instruksi (*instruction set*) yang dirancang diimplementasi menjadi mikroprosesor *virtual*. Program sederhana yang meliputi elemen instruksi: sekuens, selection, repetition, memungkinkan dilakukan. Dengan penataan penulisan program dalam bentuk *machine code* yang benar, dimungkinkan akses data dalam bentuk array dapat pula dilakukan, sebagaimana diketahui array merupakan koleksi data di memori. Mikroprosesor yang dibuat masih merupakan mikroprosesor integer, yaitu hanya dapat mengolah data numerik integer.

Semua fasilitas operasi yang disediakan pada mikroprosesor virtual ini, dapat digunakan jika dituliskan secara manual dalam bentuk *machine code* langsung. Maka untuk membantu memudahkan belajar, instruksi-instruksi *machine code* cukup dituliskan dalam file teks (ASCII), misalnya menggunakan Notepad. Setiap instruksi dituliskan dalam satu baris berupa angka numerik desimal. Sehingga jika ada dua puluh instruksi maka ada dua puluh baris numerik decimal, lihat Gambar 2.

Mengalihkan ide instruksi program, biasanya dalam bentuk *pseudocode*, ke dalam *machine code* membutuhkan upaya ekstra bila dilakukan secara manual. Sebuah kakas (*tool*) yaitu *compiler* telah dibuat yang dikembangkan menggunakan JavaFX, sehingga pengguna dapat menggunakan komputer *virtual* ini, dan menuliskan program dalam gaya penulisan *pseudocode*. Sebagai catatan, pada versi *compiler* yang ada saat ini, ada sebagian fasilitas operasi yang tersedia pada mikroprosesor belum dapat dimanfaatkan. Jika dikehendaki, maka terpaksa harus menuliskan dalam *machine code* langsung.

## 2. PEMBAHASAN

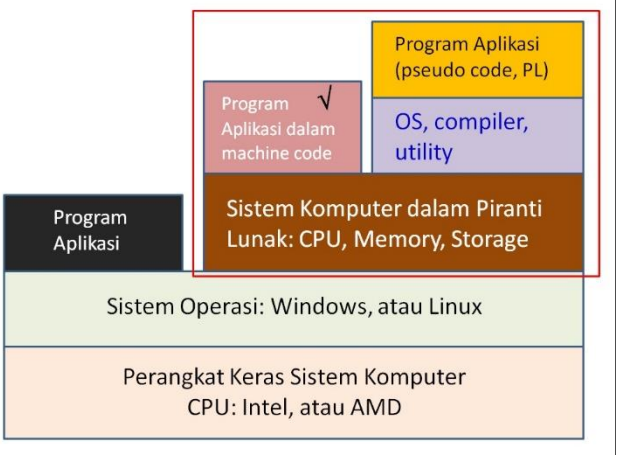
Mikroprosesor *virtual* yang dirancang dan diimplementasi merupakan mikroprosesor 32 bits, dilengkapi sejumlah *register*, menyediakan 16 buah register sebagai *general purpose registers*, register untuk *stack*, *program counter*, dan register untuk menangani multi program yang berada di memori. Penamaan 16 buah *general purpose registers* yaitu r0 hingga r15. Setiap register berukuran 32 bits.

Setiap instruksi berukuran sama yaitu 32 bits. Dari 32 bits tersebut dibagi ke dalam sejumlah fields, dan salah satunya adalah *opcode* sepanjang 8 bits, berada pada lokasi bit 24 hingga 31. Untuk instruksi yang termasuk operasi register, maka dari 32 bits disusun menjadi 5 fields, dan salah satunya dalah opcode, dan tiga fields lainnya adalah identitas register-register yang digunakan, dan satu field lainnya untuk sementara tidak digunakan. Instruksi kategori operasi register di antaranya adalah operasi penambahan (ADD), pengurangan (SUB), perkalian (MUL). Setiap field register menggunakan 4 bits. Pada instruksi yang melibatkan akses ke dan dari memori, dari 32 bits dibagi menjadi 4 fields, dan satu di antaranya adalah *opcode*. Dua fields digunakan untuk identitas register, dan satu field digunakan untuk alamat atau *immediate data*. Beberapa instruksi melibatkan memori di antaranya adalah MOV, MOVB, MOVW, dan masih ada lainnya.

Berikut ini arsitektur dari komputer *virtual*, lihat Gambar 1. Komputer *virtual* adalah yang berada di dalam segiempat merah. Komputer *virtual* berjalan di atas komputer riil, tidak terkoneksi akses langsung ke sumber daya perangkat komputer sesungguhnya. Komputer *virtual* pada dasarnya adalah sebuah program aplikasi, seperti layaknya program-program aplikasi lainnya yang berajalan di atas *host computer*. Lapisan terbawah adalah sistem komputer *virtual*, terdiri dari mikroprosesor *virtual*, dan memori virtual. Lapisan di atasnya, adalah

program aplikasi (√) yang sudah dalam *machine code*, yang dituliskan dalam file teks (ASCII), dan dapat dieksekusi oleh komputer *virtual*. Atau alternatif lain memanfaatkan *compiler* yang mengalihkan *pseudocode* ke dalam *machine code*.

Berikut adalah beberapa instruksi yang difasilitasi dan dapat dikerjakan oleh mikroprosesor virtual, lihat Tabel 1.




Gambar 1. Arsitektur Komputer Virtual

Tabel 1. Beberapa instuksi yang ada pada mikroprosesor virtual

| No | Jenis Heuristik | Fungsi | OpCode |
|---|---|---|---|
| 1 | ADD | Penjumlahan register | 0x0 |
| 2 | ADDI | Penjumlahan melibatkan immediate data | 0x01 |
| 3 | HALT | Penghentian eksekusi instruksi | 0x7F |
| 4 | CALL | Pemanggilan module | 0x22 |
| 5 | RET | Kembali dari module | 0x23 |
| 6 | MOV | Transfer data antar memori dan memori | 0x04 |
| 7 | JMP | Unconditional jump | 0x15 |
| 8 | JE | Conditional jump if equal | 0x17 |
| 9 | JLT | Conditional jump if less than | 0x19 |

Berikut ini sebuah contoh *pseudocode* yang dialihkan ke *assembly code*, dan *machine code*, lihat Gambar 2. *Pseudocode* program, lihat Gambar 2-a, yaitu menjumlahkan angka 1 hingga 10, semua data disimpan di register (yaitu r1 dan r2), dan hasilnya disimpan di register r2. Gambar 2-b, merupakan assembly language dari pseudocode kolom pertama. Dan, pada Gambar 2-c merupakan *machine code* dalam bentuk angka numerik desimal, sedangan angka pada tanda kurung siku adalah alamat memori di mana instruksi-instruksi tersebut akan ditempatkan di memori (Gambar 2-c).

```
R1 = 0
r2 = 10;
While (r1 < 10)
    r2 = r2 + r1
    r1 = r1 +1
Endwhile
End program
```

(a) *pseudocode*

```
movi r1, 0
movi r2, 10
movi r3, 10
ulang: je r1, r3, endprogram
jgt r1, r3, endprogram
add r2, r2, r1
addi r1, r1, 1
jmp ulang
endprogram: halt
```

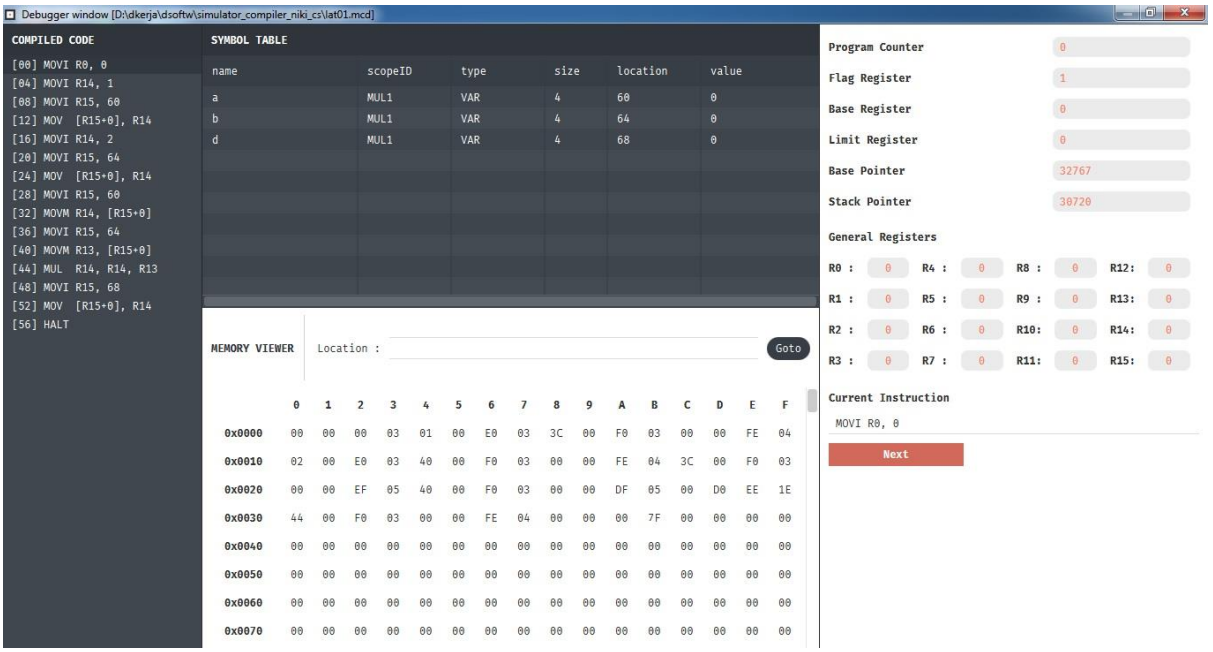
(b) *assembly code*

```
[00] 51380224
[04] 52428810
[08] 53477386
[12] 387121184
[16] 437452832
[20] 2232320
[24] 17891329
[28] 352321548
[32] 2130706432
```

(c) *machine code*

Gambar 2. Konversi *pseudocode* ke *machine code*

Berikut ini adalah contoh *compiler* yang juga menampilkan kode program yang menampilkan sejumlah status informasi, lihat Gambar 3. *Compiler* ini menyajikan status eksekusi instruksi dari program yang sedang dieksekusi oleh computer *virtual*. Pada kolom paling kiri ditampilkan assembly code dari program yang telah terkompilasi. Kolom berikutnya ditunjukkan variabel-variabel yang digunakan dalam program yang dituliskan dalam *pseudocode*. Baris di bawahnya adalah data memori *virtual*, code program ditempatkan mulai pada alamat 0 (0x0000). Kolom paling kanan, menampilkan status register-register yang dimiliki oleh prosesor *virtual*.




Gambar 3. Compiler dan status memori serta register dari program yang dieksekusi

## 3. KESIMPULAN

1. Mikroprosesor *virtual* berhasil dibuat berdasarkan rancangan arsitektur *instruction set* yang dibuat, merupakan mikroprosesor *integer*.
2. Sistem komputer *virtual* yang tersusun dari mikroprosesor *virtual*, dan memori *virtual*, dapat dibentuk, dan mampu mengeksekusi program yang ditulis dalam bahasa mesin berdasarkan mikroprosesor *virtual* tersebut.
3. Mikroprosesor virtual yang dibuat menyediakan sejumlah instruksi dasar yang diperlukan dalam pemrograman, yaitu operasi dasar aritmatika, *assignment* data ke lokasi memori (*variable*) ataupun ke *register*, serta antar *register*, *unconditional* dan *conditional jump*, operasi dasar stack, dan *call module*.
4. Sistem komputer *virtual* dapat menampung dua dua atau lebih program, meskipun eksekusinya masih dilakukan secara sekuensial.
5. Menggunakan sistem komputer virtual ini memungkinkan kita dapat memelajari untuk memahami konversi instruksi (*code*) assembly ke kode mesin (*machine code*).
6. Dapat digunakan untuk mendukung mata kuliah Sistem Operasi, dan atau mata kuliah Organisasi dan Arsitektur Komputer.

## DAFTAR PUSTAKA

[1]. David A. Patterson, John L. Hennessy. *Computer Organization and Design, The Hardware/ Software Interface*. Morgan Kaufmann, Fourth Edition, 2012.

[2]. Intel. Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture. Intel Corporation 2008.

[3]. Intel. Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-M. Intel Corporation 2008.