Arsitektur dan Organisasi Komputer

RISC (Reduced Instruction Set Computer)

PROGRAM STUDI TEKNIK INFORMATIKA

PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER

UNIVERSITAS BRAWIJAYA

MALANG

2014

Daftar Isi

A. Abstrak

Reduced Instruction Set Computer (RISC) memilikki suatu kecenderungan umum ke arah komputer dengan set instruksi yang semakin kompleks. Dengan set instruksi yang tepat dan arsitektur yang sesuai desain, keefektifan mesin dapat dicapai. Kesederhanaan instruksi set memungkinkan sebagian instruksi untuk mengeksekusi dalam satu mesin siklus, dan simplicity dari setiap waktu siklus instruksi.

B. Pendahuluan

RISC yang bisa disebut sebagai (reduced atau pengurangan) karena RISC awalnya menawarkan instruksi yang lebih kecil dan mudah diatur dibandingkan dengan mesin CISC. Mesin RISC saat ini paling gencar dikembangkan. Karena ide awalnya adalah untuk menyediakan satu set instruksi minimal yang bisa melaksanakan semua operasi penting yaitu memindahkan data, Operasi ALU, dan percabangan. Instruksi *LOAD* dan *STORE* yang eksplisit saja yang diijinkan mengakses langsung ke memori.

Desain set instruksi yang kompleks termotivasi oleh tingginya biaya memori. Memiliki kompleksitas yang lebih dan dikemas ke dalam setiap instruksi, itu berarti bahwa program bisa lebih kecil, sehingga menghemat memori. CISC ISA’s menggunakan panjang variabel instruksi, yang membuat instruksi singkat dan sederhana, sementara juga memungkinkan lagi untuk instruksi yang lebih rumit. Selain itu, arsitektur CISC termasuk sejumlah besar instruksi yang langsung mengakses memori.

Pada pertengahan 1970-an melalui karya IBM John Cocke. Cocke mulai membangun eksperimental Model 801 mainframe pada tahun 1975. Sistem ini awalnya mendapat sedikit perhatian, rinciannya diungkapkan hanya beberapa tahun kemudian. Untuk sementara, David Patterson dan David Ditzel diterbitkan mereka diakui secara luas "Kasus untuk Reduced Instruction Set Computer" pada tahun 1980. Tulisan ini melahirkan cara berpikir baru yang radikal tentang arsitektur komputer, dan membawa akronim *CISC* dan *RISC* ke dalam leksikon ilmu komputer. Arsitektur baru diusulkan oleh Patterson dan Ditzel menganjurkan sebuah instruksi yang

sederhana. Setiap instruksi akan melakukan sedikit pekerjaan, tetapi waktu yang dibutuhkan untuk eksekusi instruksi akan konstan dan dapat diprediksi. Dukungan untuk mesin RISC datang dengan cara pengamatan pemrograman pada Mesin CISC. Studi ini mengungkapkan bahwa instruksi data movement menyumbang sekitar 45% dari semua instruksi, operasi ALU (termasuk aritmatika, perbandingan, dan logis) menyumbang 25%, dan bercabang (atau aliran kontrol) sebesar 30%.

Temuan ini, ditambah dengan munculnya memori yang lebih besar dan murah, serta perkembangan teknologi VLSI, menyebabkan beragamnya jenis arsitektur. Memori yang lebih murah berarti bahwa program bisa menggunakan penyimpanan yang lebih. Program yang lebih besar yang terdiri dari sederhana, instruksi yang bisa diprediksi bisa menggantikan program pendek yang terdiri dari variabel instruksi panjang dan rumit. Instruksi sederhana, instruksi yang akan memungkinkan penggunaan siklus clock lebih pendek. Selain itu, memiliki instruksi lebih sedikit berarti lebih sedikit transistor yang diperlukan pada chip. Sedikit transistor berarti biaya produksi lebih murah dan lebih Chip real estate tersedia untuk penggunaan lainnya. Instruksi prediktabilitas ditambah dengan kemajuan VLSI akan memungkinkan berbagai peningkatan kinerja, seperti pipelining, untuk di implementasikan ke dalam perangkat keras.

C. Evolusi RISC

Pada awal tahun 1980, teknologi ISA(*Instruction Set Architecture*) sederhana mulai mendominasi dan para desainer tertarik akan hal tersebut. Karena, ISA ini cenderung menghasilkan set instruksi dengan lebih sedikit instruksi, mereka menciptakan istilah Reduced Instruction Set Computer (RISC). Bahkan meskipun tujuan utama bukanlah untuk mengurangi jumlah instruksi, tetapi lebih diutamakan adanya kompleksitas pada RISC. Kemudian, untuk mengetahui karakteristik dari RISC maka akan dijelaskan bagaimana gambaran karakteristik RISC.

1. Mengidentifikasi prinsip-prinsip desain RISC ini setelah melihat mengapa desainer mengambil rute dari CISC di tempat pertama.

2. Karena CISC dan RISC memiliki kelebihan dan kekurangan masing-masing, prosesor modern mengambil fitur dari kedua kelas. Sebagai

contoh, Power PC yang mengikuti filosofi RISC dan beberapa instruksi yang kompleks.

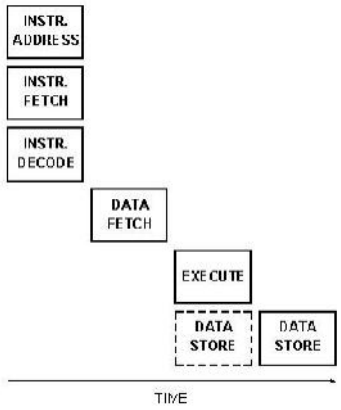


Figure 1 Typical RISC Architecture based Machine - Instruction phase overlapping

D. Definisi RISC (Reduced Interuction Set Architecture)

RISC merupakan sebuah tipe arsitekture mikroprosesor dengan utilitas yang kecil, set instruksi yang optimal, dan terkadang juga set instruksi yang biasa ditemukan pada jenis arsitektur yang lain.

a. Evolution/History

Sebuah proyek RISC pertama yang dibangun oleh IBM, Stanford, dan UC-Barkeley sekitar tahun 70’ dan 80’an. IBM 801, Stanford MIPS, dan Barkeley RISC 1 dan 2 yang di desain sedemikian kompleks yang dapat kita temui RISC saat ini.

Terdapat 3 desain fitur yang terdapat pada RISC yaitu:

(1) **One Cycle Execution Time.** Prosesor RISC memiliki CPI yang (clock per instruksi) dari satu siklus. Hal ini disebabkan oleh optimasi dari setiap instruksi pada CPU.

(2) **Pipelining.** Sebuah teknik yang memungkinkan untuk eksekusi simultan bagian, atau tahapan, instruksi untuk lebih efisien pada proses instruksi.

(3) **Large Number of Register.** RISC desain umumnya menggabungkan sejumlah besar register untuk mencegah terjadinya jumlah interaksi dengan memori yang terlalu banyak.

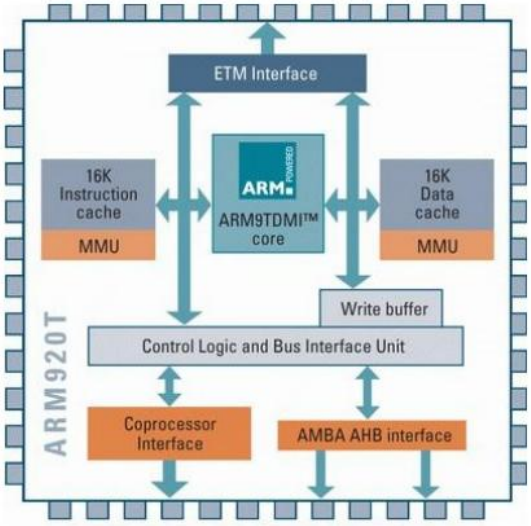
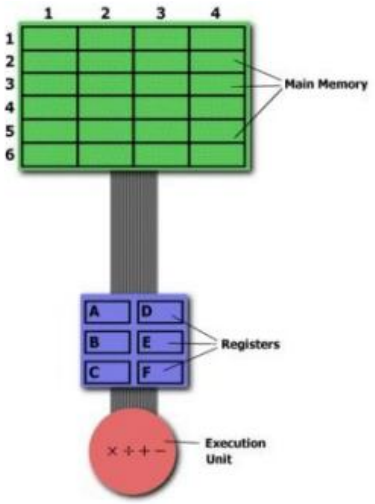


Figure 2 **Advanced RISC Machine (ARM)**

E. Perbandingan RISC dengan CISC

Cara paling sederhana untuk memeriksa keuntungan dan kerugian dari RISC arsitektur adalah dengan pendahulunya, arsitektur CISC (Complex Instruction Set Computer).

Mengalikan Dua Bilangan dalam Memori. Memori utama dibagi menjadi lokasi bernomor dari (baris) 1: (kolom) 1 (baris) 6: (kolom) 4. Eksekusi unit bertanggung jawab untuk melaksanakan semua perhitungan. Namun, unit eksekusi dapat beroperasi pada data yang telah dimasukkan ke dalam salah satu dari enam register (A, B, C, D, E, atau F). Katakanlah kita ingin mencari produk dari dua angka - satu disimpan di lokasi yang 2:3 sedangkan lainnya disimpan dalam lokasi 5:2 - dan kemudian menyimpan produk kembali di lokasi 2:3.



Representation of Storage Scheme for a Generic Computer

Pendekatan Arsitektur CISC. Tujuan utama dari arsitektur CISC adalah untuk menyelesaikan tugas beberapa baris perakitan dalam sebuah arsitektur. Hal ini dicapai dengan membangun prosesor hardware yang mampu memahami dan menjalankan beberapa rangkaian operasi. Untuk tugas tertentu ini, prosesor CISC sudah dilengkapi dengan spesifik instruksi (mengatakan " MUL ").

a. Ketika dijalankan, instruksi akan membaca dua nilai menjadi terpisah register, mengalikan operan di unit eksekusi , dan kemudian menyimpannya produk dalam register yang sesuai .
b. Dengan demikian, seluruh tugas mengalikan dua angka dapat diselesaikan dengan satu instruksi :
   MUL 2:3 , 5:2
c. MUL adalah apa yang dikenal sebagai " instruksi yang kompleks"
d. Maskapai ini mengoperasikan langsung pada bank memori komputer dan tidak

memerlukan programmer untuk memanggil secara eksplisit setiap pemuatan atau menyimpan fungsi .

e. Seperti perintah dalam bahasa tingkat yang lebih tinggi. Misalnya, jika kita membiarkan "a" mewakili nilai 2:3 dan " b " mewakili nilai 5:2, maka perintah ini identik dengan pernyataan C " a = axb "

Salah satu keuntungan utama dari sistem ini adalah bahwa compiler harus melakukan sangat sedikit pekerjaan untuk menerjemahkan pernyataan bahasa tingkat tinggi ke dalam perakitan. Karena panjang kode yang relatif singkat, sangat sedikit RAM diperlukan untuk menyimpan instruksi. Penekanan diletakkan pada pembangunan instruksi yang kompleks langsung ke hardware.

Pendekatan Arsitektur RISC. Prosesor RISC hanya menggunakan instruksi sederhana yang dapat dieksekusi dalam satu siklus clock. Dengan demikian, "MUL" perintah tersebut dapat dibagi menjadi tiga perintah yang terpisah:

a. "LOAD", yang memindahkan data dari bank memori ke register,

b. "PROD", yang menemukan produk dari dua operan yang terletak di dala register,

c. "STORE", yang memindahkan data dari register ke bank memori.

Keuntungan dari RISC memberikan dampak yang sangat penting pada arsitektur sebuak prosesor, karena setiap instruksi hanya membutuhkan satu clock cycle untuk mengeksekusi. Seluruh program akan mengeksekusi dalam jumlah waktu yang sama seperti multi - cycle pada perintah " MUL ". RISC membutuhkan lebih sedikit transistor dari ruang hardware dari pada CISC, sehingga menyisakan lebih banyak ruang untuk tujuan secara umum sebuah register.

(1) Memisahkan " LOAD " dan " STORE" sesungguhnya mengurangi jumlah pekerjaan yang komputer harus dilakukan .

(2) Setelah CISC -style " MUL " perintah dieksekusi, prosesor secara otomatis menghapus register. Jika salah satu kebutuhan operan yang akan digunakan untuk perhitungan lain, prosesor harus menerima beban data dari memori ke register . Dalam RISC, operan akan tetap di register sampai nilai lain yang dimuat di tempatnya .

Tabel berikut akan membedakan kedua arsitektur dan berdasarkan pada analisis keuntungan secara keseluruhan.

CISC	RISC
Emphasis on hardware	Emphasis on software
Includes multi-clock complex instructions	Single-clock, reduced instruction only
Memory-to-memory: "LOAD" and "STORE" incorporated in instructions	Register to register: "LOAD" and "STORE" are independent instructions
Small code sizes, high cycles per second	Low cycles per second, large code sizes
Transistors used for storing complex instructions	Spends more transistors on memory registers

Kita bisa mengukur perbedaan antara RISC dan CISC menggunakan dasar persamaan kinerja komputer sebagai berikut:

$$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{instructions}}{\text{program}}$$

Kinerja komputer, yang diukur oleh lama eksekusi program, berbanding lurus Untuk waku siklus, jumlah siklus clock per instruksi, dan jumlah instruksi dalam program ini. Memperpendek siklus clock, bila mungkin, menghasilkan peningkatan kinerja RISC serta CISC. Jika tidak, mesin CISC meningkatkan kinerja dengan mengurangi jumlah nomor instruksi program. Komputer RISC meminimalkan jumlah siklus per instruksi. Namun kedua arsitektur dapat menghasilkan hasil yang identik dalam jumlah waktu yang sama. Pada level gate, kedua sistem melakukan kuantitas setara dengan pekerjaan.

Mesin CISC mengandalkan microcode untuk mengatasi kompleksitas instruksi. Microcode memberitahu prosesor bagaimana melaksanakan setiap instruksi. Untuk alasan kinerja, microcode bersifat kompak, efisien, dan tentu saja harus benar. Bagaimanapun instruksi microcode dibatasi oleh instruksi panjang variabel, yang memperlambat proses decoding, dan berbagai jumlah siklus clock per instruksi yang membuatnya sulit untuk melaksanakan instruksi pipeline. Proses translasi tambahan akan membutuhkan waktu. Semakin kompleks set instruksi, semakin banyak waktu yang dibutuhkan untuk mencari instruksi dan melibatkan perangkat keras yang cocok untuk pelaksanaannya. Arsitektur RISC mengambil pendekatan yang berbeda. Kebanyakan instruksi RISC mengeksekusi dalam satu siklus clock. Untuk mencapai kecepatan ini, kontrol microprogrammed digantikan oleh kontrol terprogram, yang lebih cepat di mengeksekusi instruksi. Ini membuatnya lebih mudah untuk melakukan instruksi pipelining, tetapi lebih sulit untuk berurusan dengan kompleksitas pada tingkat hardware. Dalam sistem RISC, kompleksitas dihapus dari set instruksi didorong naik tingkat ke domain dari compiler.

Untuk ilustrasi, mari kita lihat sebuah instruksi. Misalkan kita ingin menghitung produk,

5 X 10. Kode pada mesin CISC mungkin terlihat seperti ini:

```
mov ax, 10
```

mov bx, 5

mul bx, ax

Sebuah minimalis RISC ISA tidak memiliki instruksi perkalian. Dengan demikian, pada sistem RISC masalah perkalian akan diubah seperti ini:

```
        mov ax, 0
        mov bx, 10
        mov cx, 5
Begin:  add ax, bx
        loop Begin    ;causes a loop cx times
```

Kode CISC, meskipun pendek, membutuhkan waktu siklus lebih untuk mengeksekusi. Seperti pada setiap arsitektur, perpindahan register ke register, penambahan, dan operasi loop masing-masing mengkonsumsi satu siklus clock. Misalkan juga bahwa operasi perkalian membutuhkan 30 siklus clock. Membandingkan dua fragmen kode yang kita miliki:

CISC instructions:

$$\text{Total clock cycles} = (2\ \texttt{movs} \times 1 \text{ clock cycle}) + (1\ \texttt{mul} \times 30 \text{ clock cycles})$$
$$= 32 \text{ clock cycles}$$

RISC instructions:

$$\text{Total clock cycles} = (3\ \texttt{movs} \times 1 \text{ clock cycle}) + (5\ \texttt{adds} \times 1 \text{ clock cycle}) + (5\ \texttt{loops} \times 1 \text{ clock cycle})$$
$$= 13 \text{ clock cycles}$$

Ditambah fakta bahwa siklus clock RISC seringkali lebih pendek dari siklus clock CISC, dan itu harus jelas meskipun ada petunjuk lebih lanjut, waktu eksekusi yang sebenarnya kurang untuk RISC dibandingkan CISC. Ini adalah inspirasi utama dibalik desain RISC. Transistor sebelumnya digunakan dalam pelaksanaan instruksi CISC adalah digunakan untuk pipeline, cache, dan register. Dari ketiganya, register menawarkan potensi terbesar peningkatan kinerja, sehingga masuk akal untuk meningkatkan jumlah register dan menggunakannya dalam cara-cara inovatif. Salah satu inovasi tersebut adalah penggunaan *Register window set.* Meskipun tidak secara luas diterima sebagai inovasi lain yang terkait dengan arsitektur RISC, register windowing ini tetap menarik Ide dan diperkenalkan secara singkat. Bahasa tingkat tinggi tergantung pada modularisasi

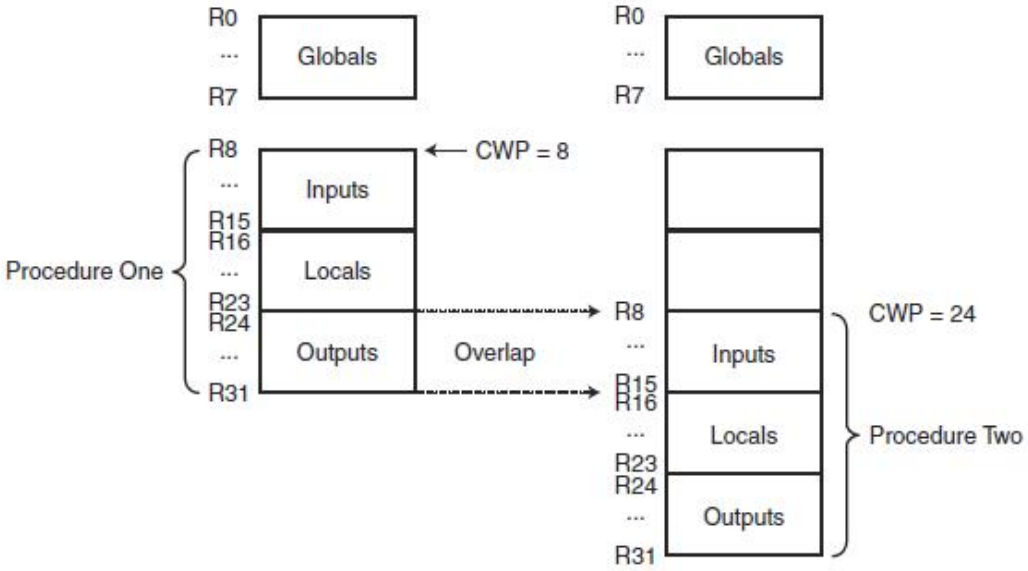
untuk efisiensi. Panggilan Prosedur bukanlah tugas yang sepele. Ini melibatkan menyimpan alamat pengirim, nilai-nilai register , melewati parameter (baik dengan mendorong mereka pada stack atau menggunakan register), bercabang ke subroutine, dan menjalankan subrutin tersebut. Setelah subroutine selesai, modifikasi nilai parameter harus diselamatkan, dan sebelumnya nilai-nilai register harus dipulihkan sebelum kembali eksekusi ke program panggilan.

Menyimpan register, melewati parameter, dan mengembalikan register melibatkan cukup usaha dan sumber daya. Dengan chip RISC memiliki kapasitas untuk ratusan register, yang menyelamatkan dan memulihkan urutan dapat dikurangi menjadi hanya dengan mengubah lingkungan register. Untuk memahami konsep ini, cobalah untuk membayangkan semua register sebagai yang dibagi ke set. Ketika program mengeksekusi dalam satu lingkungan, hanya satu set register tertentu yang terlihat. Jika program perubahan ke lingkungan yang berbeda (misalnya prosedur), untuk set register terlihat perubahan lingkungan yang baru. Misalnya, program utama sedang berjalan, mungkin itu hanya terlihat register 0 sampai 9. Ketika prosedur tertentu disebut, mungkin akan terlihat register 10 sampai dengan 19. Nilai-nilai khas untuk arsitektur RISC yang nyata termasuk 16 register set (atau *windows) dari 32 register masing-masing. CPU ini dibatasi untuk operasi hanya dalam satu* window tunggal pada waktu tertentu. Oleh karena itu, dari sudut pandang programmer, hanya ada 32 register yang tersedia. Register windows, dengan sendirinya, tidak selalu membantu dengan prosedur panggilan atau parameter passing. Namun, jika windows tersebut tumpang tindih maka hati-hati, melewati parameter dari satu modul ke yang lain menjadi masalah sederhana yaitu pergeseran dari satu register set yang lain, yang memungkinkan dua set tumpang tindih justru dalam register yang harus dibagi untuk melakukan parameter passing. Hal ini dilakukan dengan membagi windows register set ke partisi yang berbeda, termasuk *register yang global* (Umum untuk semua windows), *register lokal* (Lokal ke windows aktif), *register masukan* (Yang tumpang tindih dengan windows sebelumnya yang register output), dan *register keluaran* (Yang tumpang tindih dengan windows depan masukan register). Ketika CPU beralih dari satu prosedur ke depan, beralih ke windows register yang berbeda, tetapi tumpang tindih windows memungkinkan parameter untuk "Lulus" hanya dengan mengubah dari register output dalam panggilan tersebut modul masukan register dalam apa yang disebut modul. A *window saat*

pointer (CWP) menunjuk ke windows register diatur untuk digunakan pada waktu tertentu.

Pertimbangkan skenario di mana Prosedur satu memanggil Prosedur Dua. Dari 32 register di setiap set, asumsikan 8 bersifat global, 8 lokal, 8 adalah untuk masukan, dan 8 adalah untuk output. Kapan Prosedur satu memanggil Prosedur Dua, setiap parameter yang perlu diteruskan dimasukkan ke dalam set keluaran register Prosedur satu . Sekali Prosedur Dua dimulai eksekusi, register ini menjadi set masukan register untuk Prosedur Dua.

Proses ini diilustrasikan pada Gambar dibawah



Overlapping Register Windows

Salah satu bagian yang lebih penting dari informasi untuk dicatat mengenai register windows pada mesin RISC adalah sifat melingkar dari set register. Untuk program bersarang yang tingkat tinggi, adalah mungkin untuk mengambil kebutuhan register. Ketika ini terjadi, memori utama mengambil alih, menyimpan windows bernomor terendah, yang mengandung nilai-nilai dari aktivasi prosedur tertua. Register bernomor tertinggi (yang aktivasi terbaru) kemudian membungkus ke register bernomor terendah. Seperti kembali dari prosedur dijalankan, tingkat bersarang menurun, dan nilai-nilai register dari memori dikembalikan dalam urutan

yang dimana mereka tersimpan. Selain sederhana, instruksi tetap-panjang, pipeline efisien dalam mesin RISC telah menyediakan arsitektur ini dengan peningkatan besar dalam kecepatan. Instruksi sederhana telah membebaskan Chip real estate, sehingga tidak hanya lebih ruang yang dapat digunakan, tetapi juga dalam chip yang lebih mudah dan lebih memakan waktu untuk merancang manufaktur.

Hal ini menjadi semakin sulit untuk mengkategorikan prosesor terkini baik sebagai RISC atau CISC. Garis yang memisahkan arsitektur ini tidak terlalu jelas perbedaannya. Beberapa arsitektur saat ini menggunakan kedua pendekatan tersebut. Jika menelusuri beberapa manual chip yang lebih baru, bisa melihat mesin RISC yang saat ini memiliki instruksi lebih boros dan lebih kompleks dari beberapa mesin CISC. PowerPC RISC, misalnya, memiliki set instruksi yang lebih besar daripada Pentium CISC. Sebagai teknologi VLSI terus membuat transistor lebih kecil dan lebih murah, ekspansif dari set instruksi sekarang menjadi kurang dari sebuah isu di CISC dibandingkan RISC, sedangkan register penggunaan dan arsitektur load / store adalah menjadi lebih menonjol.

Berikut adalah Perbedaan klasik antara RISC dan CISC.

RISC	CISC
Multiple register sets, often consisting of more than 256 registers	Single register set, typically 6 to 16 registers total
Three register operands allowed per instruction (e.g., `add R1, R2, R3`)	One or two register operands allowed per instruction (e.g., `add R1, R2`)
Parameter passing through efficient on-chip register windows	Parameter passing through inefficient off-chip memory
Single-cycle instructions (except for `load` and `store`)	Multiple-cycle instructions
Hardwired control	Microprogrammed control
Highly pipelined	Less pipelined
Simple instructions that are few in number	Many complex instructions
Fixed length instructions	Variable length instructions
Complexity in compiler	Complexity in microcode
Only load and store instructions can access memory	Many instructions can access memory
Few addressing modes	Many addressing modes

Seperti yang telah disebutkan, meskipun banyak sumber memuji inovasi revolusioner desain RISC, banyak ide-ide yang digunakan dalam mesin RISC

(termasuk pipelining dan petunjuk sederhana) yang diterapkan pada mainframe pada tahun 1960 dan 1970. Ada banyak disebut desain baru yang tidak benar-benar baru, tetapi hanya daur ulang. Inovasi tidak selalu berarti menciptakan proses baru, melainkan mungkin kasus sederhana mencari tahu cara terbaik untuk menggunakan proses yang sudah ada.

F. Versi Arsitektur RISC

F.1. ARSITEKTUR DASAR RISC I

RISC I set instruksi berisi beberapa operasi sederhana (aritmatika, logis, dan shift) yang beroperasi pada register. Instruksi, data, alamat, dan register 32 bit . Instruksi RISC dibagi dalam empat kategori aritmatika - logika (ALU), akses memori, cabang, dan lainnya. Waktu eksekusi RISC I memberikan waktu yang dibutuhkan untuk membaca ,mendaftar, melakukan operasi ALU, dan menyimpan hasilnya kembali ke register. Register 0, yang selalu berisi 0. Memungkinkan kita untuk mensintesis berbagai operasi dan mode pengalamatan. Beban dan menyimpan instruksi memindahkan data antar register dan memori. Instruksi ini menggunakan dua siklus CPU. Kami memutuskan untuk membuat pengecualian terhadap kendala eksekusi kami, siklus tunggal memperpanjang dan mengizinkan akses memori lengkap. Ada 8 variasi instruksi akses memori untuk mengakomodasi sign-diperpanjang atau nol - diperpanjang 8 - bit, 16 - bit. Dan data 32-bit. Meskipun tampaknya hanya ada satu mode pengalamatan, dalam x ditambah perpindahan, absolut dan disintesis menggunakan daftar 0. Selalu mengandung 0 setidaknya ke CDC - 6600 pada tahun 1964. Hal ini juga muncul dalam lebih desain baru. Instruksi cabang termasuk Call, Back, Konditional dan Jump tanpa syarat. Instruksi bersyarat adalah set standar yang digunakan awalnya di PDP - 11 dan ditemukan di kebanyakan mikroprosesor 16 - bit hari ini. Sebagian besar fitur inovatif RISC ditemukan di Call Back, dan Jump.

Jika IMM sama dengan 0, orde rendah 5 bit Source2 register lain: jika IMM sama dengan 1 , Source2 mengungkapkan tanda, 13 - bit konstan. Karena frekuensi terjadinya konstanta integer dalam program bahasa tingkat tinggi, bidang langsung telah membuat pilihan dalam setiap instruksi. SCC menentukan kode kondisi yang ditetapkan. Petunjuk menggunakan akses SOURCE1 untuk menentukan indeks mendaftar dan Source2 untuk menentukan offset. Format

menggabungkan tiga bidang terakhir untuk membentuk 19 - bit PC - relative address, digunakan terutama untuk instruksi cabang. Meskipun pengukuran perbandingan dari benchmark yang efektivitas, menunjukkan bahwa banyak petunjuk penting VAX dapat disintesis dari sederhana RISC mode pengalamatan dan Kode operasi. Ingat bahwa register 0 (r0) selalu berisi 0; menetapkan 10 sebagai tujuan tidak mengubah nilainya .

F.2. RISC Windows

Investigasi yang disebutkan sebelumnya menggunakan bahasa tingkat tinggi menunjukkan bahwa CALL prosedur mungkin operasi, memakan waktu dalam program bahasa RISC. Program mungkin memiliki jumlah yang lebih besar dari panggilan, karena instruksi kompleks yang ditemukan dalam ClSCs adalah subrutin di RISC. Dengan demikian, CALL prosedur harus secepat mungkin. RISC Skema register dekat dengan tujuan ini. Pada saat yang sama, skema ini juga mengurangi jumlah akses ke memori data. Menggunakan prosedur melibatkan dua kelompok operasi ini memakan waktu memulihkan registers pada setiap Call atau Back, dan hasil dari passing parameter, karena pengukuran pada bahasa tingkat tinggi program menunjukkan bahwa skalar lokal paling sering operan, kami ingin mendukung alokasi dari register. Mikroprosesor menyimpan beberapa register pada chip untuk menghindari penghematan restorating. Dengan demikian, masing-masing hasil prosedur Call dalam satu set baru register dialokasikan untuk digunakan untuk prosedur baru. Skema yang sama diadopsi oleh RISC I, namun beberapa register tidak disimpan atau dikembalikan pada masing-masing Prosedur Call. Register ini fr0 melalui r9 disebut register global.

Selain itu, set register yang digunakan oleh berbagai proses yang tumpang tindih untuk memungkinkan parameter menjadi lulus dalam register. Pada mesin lain, parameter biasanya diteruskan stack dengan prosedur panggilan menggunakan register (frame pointer) untuk menunjuk ke awal parameter (dan juga untuk akhir penduduk setempat) . Dengan demikian, semua referensi untuk parameter diindeks referensi ke memori. Untuk mematahkan set jendela register (r10 untuk R31) menjadi tiga bagian Register 26 sampai 31 (TINGGI) berisi parameter lulus dari "atas" prosedur saat ini, yaitu memanggil prosedur. Register 16 sampai 25 (LOKAL) digunakan untuk penyimpanan skalar lokal persis seperti

yang dijelaskan sebelumnya. Register 10 sampai 15 (LOW) digunakan untuk penyimpanan lokal dan parameter yang dikirimkan ke prosedur "bawah" prosedur saat ini (yang disebut prosedural). Pada setiap PANGGILAN prosedur satu set baru register, r10 untuk R31, dialokasikan, namun, kami ingin Register dari RENDAH menjadi TINGGI register. Hal ini dilakukan dengan memiliki hardware yang tumpang tindih register LOW dari memanggil frame dengan register TINGGI yang disebut frame: demikian. Tanpa memindahkan informasi, parameter dalam registers 10 sampai 15 muncul dalam register 25 melalui 31 di disebut frame. Untuk kasus dimana prosedur A memanggil procedure B, dan yang panggilan prosedur C. RISC I menangani ini dengan daftar terpisah stack overflow dalam memori dan stack pointer untuk itu Overflow dan underflow ditangani dengan perangkap untuk perangkat lunak yang menyesuaikan tumpukan itu, karena ini rutin dapat menyimpan atau mengembalikan beberapa set register, yang meluap/underflow frekuensi didasarkan pada lokal di kedalaman tumpukan bukan pada kedalaman mutlak Efektivitas skema ini tergantung pada frekuensi relatif melimpah dan underflows ; studi oleh Halbert dan Kessler13 menunjukkan overflow yang akan terjadi dalam waktu kurang dari 1 % dari panggilan dengan hanya 4 sampai 8 bank mendaftar. (Mesin lain, seperti sebagai BBN C/70, mengandung mendaftar bank, tetapi mereka tidak tumpang tindih jendela mereka)

Langkah terakhir Dalam mengalokasikan variabel Dalam register adalah penanganan masalah pointers. Pointer ke variable mengharuskan variabel memiliki alamat. Karena register biasanya tidak memiliki alamat, orang bisa membiarkan compiler menentukan apa variabel memiliki pointer dan menempatkan variabel tersebut dalam memori. Ini menghalangi kompilasi, memperlambat akses ke variabel tersebut, dan berada diluar of the art teknologi compiler yang ditemukan di kebanyakan perusahaan dan universitas. RISC I memecahkan bahwa masalah dengan memberikan alamat ke register jendela. Jika kita menyisihkan sebagian dari ruang alamat, kita bisa menentukan, dengan satu perbandingan, apakah alamat menunjuk ke register atau ke memori. Karena satu-satunya instruksi untuk mengakses memori beban dan menyimpan, dan mereka mengambil siklus ekstra sudah, kita dapat menambahkan fitur ini tanpa mengurangi kinerja beban dan petunjuk. Hal ini memungkinkan penggunaan

langsung teknologi compiler dan masih menyisakan sebagian besar variabel dalam register.

Kami akan meningkatkan performa prefetching berikutnya instruksi duing eksekusi instruksi saat ini. Kesulitan dengan instruksi cabang. Beberapa mesin high-end memiliki teknik yang rumit untuk prefetch instruksi yang tepat setelah cabang,tapi teknik ini terlalu rumit untuk chip tunggal RISC . Solusi kami adalah untuk mendefinisikan kembali sehingga mereka tidak berpengaruh sampai instruksi selanjutnya. Kembali ke MANIAC I tahun 1952 dan sekarang umum digunakan dalam microprogramming).Kemungkinkan RISC selalu prefetch instruksi berikutnya selama pelaksanaan arus instruksi. Kode bahasa mesin yang sesuai diatur sedemikian rupa sehingga hasil yang diinginkan diperoleh. Karena RISC I selalu dimaksudkan untuk diprogram dalam bahasa tingkat tinggi, kita tidak akan "beban" yang programmer dengan kompleksitas ini: beban akan dilakukan oleh programmer dari compiler tersebut, optimizer, dan debugger.

F.3. MEMORY INTERFACE

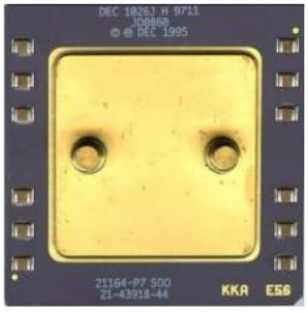
Pada kebanyakan komputer, antarmuka untuk memori utama sebuah kinerja bottleneck harus diberikan pertimbangan khusus. Sebenarnya kita dapat mengakses memori utama dalam siklus RISC CPU tunggal. Tergantung pada asumsi yang kita buat untuk CPU waktu siklus kita, dan ukuran memori utama, asumsi ini mungkin

terlalu optimis. Dengan demikian kita berasumsi bahwa ada dua siklus CPU yang diperlukan untuk mengakses memori data. Kinerja hanya terdegradasi 10%, karena skema jendela mendaftar mengurangi sejumlah referensi data yang off-chip. Referensi data yang dilakukan bukan merupakan masalah, tetapi memungkinkan dua siklus untuk mengambil instruksi dari memori akan mengurangi kinerja dengan hampir satu faktor . Jelas, memory interface ini akan menjadi semakin kritis seperti kecepatan intrinsik CPU meningkat dengan kemajuan teknologi. Akses ke memori dapat dipaksa untuk datang terutama dari on-chip, baik dengan besar mendaftar file atau dengan cache on-chip dan terkait hirarki memori.

Cache on-chip akan bermanfaat untuk RISC. Kadang-kadang lupa bahwa cache tidak efektif jika terlalu kecil. Menurut pendapat kami, cache data yang efektif akan sedikit lebih besar dari daftar kami direncanakan, terutama jika untuk memberikan jumlah yang sama port sebagai register file. Terjemahan yang lebih rumit dan decoding bahkan mungkin strech siklus CPU dasar. Mengingat jumlah terbatas sirkuit kita dapat menempatkan ke sebuah chip pada saat ini, dan diberikan universitas lingkungan dan desainer mahasiswa , file register jelas merupakan cara yang lebih aman untuk pergi meskipun masalah akses data telah dikurangi dengan jumlah besar register dan skema window yang efektif, jumlah instruksi fetch sebenarnya telah meningkat karena kesederhanaan instruksi individu. Instruksi menjemput dari main memori memang faktor membatasi *speedup*. Sebuah cache instruksi komoditas yang diinginkan. Karena tidak ada kebutuhan untuk CPU untuk wnte ke dalam cache ini, controller dapat lebih sederhana dibandingkan dengan cache data. Kami memutuskan bahwa RISC saya seharusnya tidak dibebani dengan desain cache full-blown on-chip, tetapi instruksi cache pasti akan menjadi ide yang baik untuk generasi RISC.

G. Contoh Prosesor RISC

1. **Digital Equipment Corporation (DEC) - Alpha**. Alpha, awalnya dikenal sebagai Alpha AXP, adalah 64-bit reduced instruction set computer (RISC) set instruksi arsitektur (ISA) yang dikembangkan oleh Digital Equipment Corporation (DEC), dirancang untuk menggantikan 32-bit set instruksi VAX kompleks komputer (CISC) ISA dan implementasinya.



DEC Alpha Microprocessor developed in 1995

2. **Advanced RISC Machine (ARM)**. ARM adalah 32-bit Reduced Instruction Set Computer (RISC) set instruksi arsitektur (ISA) yang dikembangkan oleh ARM Holdings. Hal ini dikenal sebagai Advanced RISC Machine, dan

sebelum itu sebagai Acorn RISC Machine. Hal ini telah membuatnya dominan dalam elektronik mobile dan embedded pasar dengan biaya yang relatif rendah dan mikroprosesor kecil dan mikrokontroler.

Advanced RISC Machine (ARM) Microprocessor developed by Conexant Computers

3. **Atmel AVR**. AVR adalah Modified Harvard Architecture 8-bit RISC tunggal chip mikrokontroler (µC) yang dikembangkan oleh Atmel pada tahun 1996. AVR adalah salah satu keluarga mikrokontroler pertama yang menggunakan on-chip flash memory untuk penyimpanan program, sebagai lawan One-Time Programmable ROM, EPROM, EEPROM atau yang menggunakan mikrokontroler pada saat itu.

Atmel AVR ATmega8 Microprocessor

4. **Performence Optimized with Enhanced RISC** – Kinerja Computing (POWER-PC). PowerPC adalah arsitektur RISC yang dibuat oleh 1991 Apple IBM-Motorola aliansi, yang dikenal sebagai AIM.
   a. Awalnya ditujukan untuk desain komputer pribadi
   b. Digunakan dalam prosesor kinerja tinggi.
   c. PowerPC sebagian besar didasarkan pada arsitektur POWER pada IBM sebelumnya, dan mempertahankan dengan kompabilitas yang tinggi.

PowerPC 600 Series developed by IBM Computer

H. Kesimpulan

Setelah dijelaskan semua tentang RISC (*Reduced Instruction Set Architecture*) dan memperkenalkan karakteristik penting yang membedakan desain RISC dengan CISC. Desain CISC memberikan instruksi yang kompleks dan besar jumlah mode pengalamatan. Alasan untuk kompleksitas ini adalah keinginan untuk menutup kesenjangan semantik yang ada antara bahasa tingkat tinggi dan bahasa mesin. Pada awalnya, efektifitas penggunaan prosesor dan sumber daya memori adalah sangat penting. Instruksi yang kompleks cenderung untuk meminimalkan persyaratan memori.

Data empiris bagimana menyarankan bahwa kompiler tidak menggunakan instruksi kompleks, melainkan mereka menggunakan instruksi sederhana untuk mensintesis instruksi yang kompleks.

Prinsip RISC, berdasarkan studi empiris pada prosesor CISC, telah diusulkan sebagai alternatif untuk CISC. Sebagian besar desain prosesor saat ini didasarkan pada prinsip RISC.

I. Soal & Jawaban

1. Sebutkan 3 perintah dasar sederhana RISC beserta penjelasan yang dapat dieksekusi dalam satu siklus clock ?

   "LOAD", yang memindahkan data dari bank memori ke register,

   "PROD", yang menemukan produk dari dua operan yang terletak di dala register

   "STORE", yang memindahkan data dari register ke bank memori.
2. Sebutkan dan jelaskan 3 fitur desain yang terdapat pada RISC?

- **One Cycle Execution Time : Prosesor RISC memiliki CPI yang (clock per instruksi) dari satu siklus.**
- **Pipelining : Sebuah teknik yang memungkinkan untuk eksekusi simultan bagian, atau tahapan, instruksi untuk lebih efisien pada proses instruksi.**
- **Large Number of Register : RISC desain umumnya menggabungkan sejumlah besar register untuk mencegah terjadinya jumlah interaksi dengan memori yang terlalu banyak.**

3. Apa definisi dari arsitektur RISC pada sistem mikroprosesor?

   RISC merupakan sebuah tipe arsitekture mikroprosesor dengan utilitas yang kecil, set instruksi yang optimal, dan terkadang juga set instruksi yang biasa ditemukan pada jenis arsitektur yang lain.

4. Jelaskan perbedaan arsitektur terdahulu CISC dengan arsitektur RISC? (buat tabel)
5. T

CISC	RISC
Emphasis on hardware	Emphasis on software
Includes multi-clock complex instructions	Single-clock, reduced instruction only
Memory-to-memory:	Register to register:
"LOAD" and "STORE" incorporated in instructions	"LOAD" and "STORE" are independent instructions
Small code sizes, high cycles per second	Low cycles per second, large code sizes
Transistors used for storing complex instructions	Spends more transistors on memory registers

a

ma RISC ?

Untuk mendapatkan Kinerja optimal, mengurangi desain waktu , mengurangi jumlah kesalahan desain, dan waktu eksekusi instruksi individu, hasil dinamis yang menarik,maka harus menggunakan program C yang dikembangkan. Kita memerlukan sebuah Program yang pada dasarnya adalah sebuah rekursif Program bin -packing yang memecahkan tiga dimensi teka-teki dan menampilkan banyak fitur dari

program khas, selain itu ada kurang dari 0,2 % panggilan prosedur , panggilan stack dan ada sejumlah loop relatif besar.

6. Apa yang harus dilakukan untuk mengoptimalkan RISC ?

   Mengeksplorasi alternatif dengan kecenderungan umum ke arah kompleksitas arsitektur. Hipotesisnya adalah bahwa dengan mengurangi set instruksi , arsitektur VLSI dapat dirancang menggunakan sumber daya langka secara lebih efektif dibandingkan CISC.

7. Apa yang harus anda lakukan untuk memperoleh kesederhanaan dan keefektifan implementasi chip tunggal pada Arsitektur?

   - Mengeksekusi satu instruksi per siklus .Instruksi harus lebih cepat, dan tidak rumit daripada , instruksi mikro di mesin saat ini seperti PDP - 11 atau VAX . Selain itu, kesederhanaan ini tidak membutuhkan microcode kontrol. tingkat penafsiran muncul untuk meningkatkan kinerja sekaligus mengurangi ukuran chip.

   - Semua instruksi mempunyai ukuran yang sama. Menyederhanakan implementasi diperlukan untuk mengurangi ukuran program.

   - Hanya memuat dan menyimpan instruksi pengakses memori ;
   sisanya beroperasi antara register. pembatasan ini menyederhanakan desain. Kurang kompleksnya mode pengalamatan juga membuat lebih mudah untuk me-restart mode dan juga mudah untuk me-restart petunjuk

   - Mendukung bahasa tingkat tinggi (HLL).RISC I mendukung alamat 32 - bit , 8 - , 16 - , dan 32 – bit data, dan beberapa register 32-bit . Kami bermaksud untuk memeriksa dukungan untuk sistem operasi dan perhitungan floating -point dalam penerus RISC I. kendala tersebut akan menghasilkan mesin dengan kepadatan kode substansial kinerja antar keduanya . Terlepas dari kendala tersebut, Arsitektur yang dihasilkan bersaing baik dengan lainnya. State -of - the-art mesin seperti VAX 11/780.

8. Sebutkan Instruksi dalam RISC yang anda ketahui!

   - Aritmatika - logika (ALU)

   - Akses memori

   - Cabang dan lain-lain

9. Jelaskan yang anda ketahui tentang Register Global!

   Mikroprosesor menyimpan beberapa register pada chip untuk menghindari penghematan restorating.Dengan demikian, masing- masing hasil prosedur PANGGILAN dalam satu set baru register dialokasikan untuk digunakan untuk prosedur baru Skema yang sama diadopsi oleh RISC, namun beberapa register tidak disimpan atau dikembalikan pada masing-masing Prosedur PANGGILAN. Register ini melalui FR0 hingga R9 inilah yang disebut sebagai register global.

10. Apakah alasan penamaan nama arsitektur RISC?

    Reduced yang berarti pengurangan, karena intruksinya lebih kecil dan mudah diatur dibandingkan dengan arsitektur CISC

11. Apakah penyebab set instruksi yang didesain semakin kompleks?

    Karena semakin tingginya biaya memori

12. Siapakah yang pertama kali menggagas tentang arsitektur RISC?

    John Cocke

13. Manakah yang lebih besar set instruksi PowerPC RISC daripada Pentium CISC?

    PowerPC RISC

14. Apakah yang diandalkan Mesin CISC untuk mengatasi kompleksitas intruksi?

    Microcode

15. Apakah yang tejadi dengan erkembangnya teknologi VLSI?

    Mengakibatkan beragamnya jenis arsitektur

16. Apakah perbedaab control antara RISC dengan CISC?

    RISC adalah hardwired control sedangkan CISC adalah microprogrammed control

J. Daftar Pustaka

CISC, RISC, and DSP Microprocessors" by Douglas L. Jones 2000

Computer Organization Design RISC, by David A. Patterson 1998

The Essentials of Computer Organization and Architecture, by Linda Null 2008

The Evolution of RISC Technology at IBM, by John Cocke & V. Markstein 1990