**Modul Perkuliahan sesi 12**

**Organisasi dan Arsitektur Komputer**

**“Reduced Instruction Set Computer (RISC)”**

Tonggak perkembangan arsitektur computer adalah

* Family concept
* Microprogrammed control unit
* Cache memory
* Pipelining
* Multiprocessor
* RISC

Rancangan arsitektur CPU yang mengambil dasar filosofi bahwa proseor dibuat dengan arsitektur yang tidak rumit dengan membatasi jumlah instruksi hanya pada instruksi dasar yang diperlukan saja.

Kata “reduced” berarti pengurangan pada set instruksinya.

Rancangan ini berawal dari pertimbangan-pertimbangan dan analisa model perancangan lain yang kompleks, sehingga harus ada pengurangan set instruksinya.

Pemakai Teknik RISC

* Didominasi oleh IBM dengan Intel Inside-nya
* Processor PowerPC adalah processor buatan Motorola yang menjadi otaku tama computer apple machintosh memakai teknik RISC dalam desiannya
* Macintosh, DEC, SUN adalah computer yang handal dengan system pipelining, superscalar, operasi floating point

Karakteristik RISC

* Siklus instruksi
* Operasi pertukaran data
* Mode pengalamatan
* Format instruksi

Siklus instruksi

* Satu instruksi per siklus mesin.
* Siklus mesin ditentukan oleh waktu yang digunakan untuk mengambil dua buah opernd dari register, melakukan operasi ALU, dan menyimpan hasil operasinya ke dalam reister
* RISC adalah rancangan processor yang sederhana, tetapi dalam kesederhanaan tersebut didapatkan kecepatan operasi tiap-tiap siklus instruksinya.
* Instruksi dibatasi hanya menyediakan instruksi dasar saja.
* Fungsi-fungsi yang kompleks akan diterjemahkan dalam operasi instruksi-instruksi dasar.

Operasi pertukaran data

* Berbentuk pertukaran data dari register ke registr
* Dengan mengoptimalkan penggunaan memori register diharapkan siklus operasi semakin cepat
* Register adalah memori yang paling cepat dibandingkan cache maupun memori utama
* Dengan penyederhanaan instruksi maka operasi unit control juga akan sederhana dan cepat
* Penekanan penggunaaan operasi dari register ke register adalah hal yang unik pada rancangan RISC
* Rencangan konemporer lainnya memiliki instruksi register ke register juga, namun juga melibatkan operasi langsung ke memori utama dalam fetch

Mode Pengalamatan

* Fitur racangan ini juga dapat menyederhanakan set instruksi dan unit control.
* Dengan mode pengalamatan yang sederhana akan didapatkan operasi pengambilan data dan penyimpanan data semakin cepat

Format Instruksi

* Umumnya hanya digunakan sebuah format atau beberapa format saja untuk menyederhanakan implementasi perangkat kerasnya.
* Panjang instruksi tetap dan disamakan dengan panjang word yang digunakan.
* Panjang field dibuat sama dn tetap
* Kelebihannya adalah, denagn menggunakan filed yang tetap maka pengkodean opcode dan pengaksesn operand register dapat dilakukan secara bersamaan.
* Format yang sederhana juga akan memudahkan kerja unit control.

Ringkasan Rancangan RISC Tidak

1. Instruksi berukuran tunggal
2. Ukuran instruksi umumnya 4 byte
3. Jumlah mode pengalamatan data sedikit, biasanya kurang dari lima macam. Tidak mengenal pengalamatan tak langsung.
4. Tidak terdapat operasi yang menggabungkan operasi ambil data dan simpan data dengan operasi-operasi aritmatika
5. Tidak terdapat lebih dari satu operand beralamat memori per instruksi
6. Jumah maksimum pemakaian memori manajemen (MMU) bagi suatu alamat data adalah satu instruksi
7. Jumlah bit bagi integer-integer specifier sama dengan lima atau lebih. Ini berarti sedikitnya 32 buah register integer dapat direferensikan sekaligus secara eksplisit
8. Jumlah bit bagi floating point register specifier sama dengan empat atau lebuh sehingga sedikitnya 16 buah register floating point dapat direferensikan bersama secara eksplisit.

**RISC**

RISC adalah singkatan dari Reduced Instruction Set Computer(Komputasi set instruksi yang disederhanakan), kata “reduced” berarti pengurangan pada set instruksinya. RISC merupakan

rancangan arsitektur CPU yang mengambil dasar filosofi bahwa prosesor dibuat dengan arsitektur yang tidak rumit dengan membatasi jumlah instruksi hanya pada instruksi dasar yang diperlukan saja. Dengan kata lain RISC adalah arsitektur komputer dengan kumpulan perintah (instruksi) yang sederhana, tetapi dalam kesederhanaan tersebut didapatkan kecepatan operasi setiap siklus instruksinya. Kebanyakan pada prosesor RISC, instruksi operasi dasar aritmatik hanya penjumlahan dan pengurangan. Untuk perkalian dan pembagian sudah dianggap operasi yang kompleks. RISC menyederhanakan rumusan perintah sehingga lebih efisien dalam penyusunan kompiler yang pada akhirnya dapat memaksimumkan kinerja program yang ditulis dalam bahasa tingkat tinggi.

Beberapa elemen penting pada arsitektur RISC, yaitu:

· Set Intruksi yang terbatas dan sederhana

· Register General Purpose yang berjumah banyak, atau penggunaan teknologi kompiler untuk mengoptimalkan pemakaian registernya

· Penekana pada pengoptimalan pipeline intruksi Konsep Arsitektur RISC banyak menerapkan proses eksekusi pipeline. Meskipun jumlah perintah tunggal yang diperlukan untuk melakukan pekerjaan yang diberikan mungkin lebih besar, eksekusi secara pipeline memerlukan waktu yang lebih singkat daripada waktu untuk melakukan pekerjaan yang sama dengan menggunakan perintah yang lebih rumit. Mesin RISC memerlukan memori yang lebih besar untuk mengakomodasi program yang lebih besar. Salah satu contoh adalah IBM 801 adalah prosesor komersial pertama yang menggunakan pendekatan RISC. Untuk lebih lanjut memahami RISC, diawali dengan tinjauan singkat tentang karakteristik eksekusi Instruksi yaitu Aspek komputasi yang ditinjau dalam merancang mesin RISC adalah sbb.:

· Operasi-operasi yang dilakukan:

Hal ini menentukan fungsi-fungsi yang akan dilakukan oleh CPU dan interaksinya dengan memori.

· Operand-operand yang digunakan:

Jenis-jenis operand dan frekuensi pemakaiannya akan menentukan organisasi memori untuk menyimpannya dan mode pengalamatan untuk mengaksesnya.

· Pengurutan eksekusi:

Hal ini akan menentukan kontrol dan organisasi pipeline.

Salah satu jenis dari arsitektur, dimana Superscalar adalah sebuah Uniprocessor yang dapat mengeksekusi dua atau lebih operasi scalar dalam bentuk paralel. Merupakan salah satu rancangan untuk meningkatkan kecepatan CPU. Kebanyakan dari komputer saat ini menggunakan mekanisme Superscalar ini. Standar Pipeline yang digunakan adalah untuk pengolahan bilangan matematika integer (bilangan bulat, bilangan yang tidak memiliki pecahan), kebanyakan CPU juga memiliki kemampuan untuk pengolahan untuk data Floating Point (bilangan berkoma). Pipeline yang mengolah integer dapat juga digunakan untuk mengolah data bertipe floating point ini, namun untuk aplikasi tertentu, terutama untuk aplikasi keperluan ilmiah CPU yang memiliki kemampuan pengolahan floating point dapat meningkatkan kecepatan prosesnya secara dramatis. Peristiwa menarik yang bisa dilakukan dengan metoda superscalar ini adalah dalam hal memperkirakan pencabangan instruksi (brach prediction) serta perkiraan eksekusi perintah (speculative execution).

Peristiwa ini sangat menguntungkan buat program yang membutuhkan pencabangan dari kelompok intruksi yang dijalankankannya. Program yang terdiri dari kelompok perintah bercabang ini sering digunakan dalam pemrograman. Contohnya dalam menentukan aktifitas yang dilakukan oleh suatu sistem berdasarkan umur seseorang yang sedang diolahnya, katakanlah jika umur yang bersangkutan lebih dari 18 tahun, maka akan diberlakukan instruksi yang berhubungan dengan umur tersebut, anggaplah seseorang tersebut dianggap telah dewasa, sedangkan untuk kondisi lainnya dianggap belum dewasa. Tentu perlakuannya akan dibedakan sesuai dengan sistem yang sedang dijalankan. Lalu apa yang dilakukan oleh CPU untuk hal ini? Komputer akan membandingkan nilai umur data yang diperolehnya dengan 18 tahun sehingga komputer dapat menentukan langkah dan sikap yang harus diambilnya berdasarkan hasil perbandingan tersebut. Sikap yang diambil tentu akan diambil berdasarkan pencabangan yang ada. Pada CPU yang mendukung perintah pencabangan ini, CPU membutuhkan lumayan banyak clock cycle, mengingat CPU menempatkan semuanya pada pipeline dan menemukan perintah berikutnya yang akan dieksekusinya. Sirkuit untuk branch prediction melakukan pekerjaan ini bekerja sama dengan pipeline, yang dilakukan sebelum proses di

ALU dilaksanakan, dan memperkirakan hasil dari pencabangan tersebut. Jika CPU berfikir bahwa branch akan menuju suatu cabang, biasanya berdasarkan pekerjaan sebelumnya, maka perintah berikutnya sudah dipersiapkan untuk dieksekusi berikut datadatanya, bahkan dengan adanya pipeline ini, bila tidak diperlukan suatu referensi dari instruksi terakhir, maka bisa dilaksanakan dengan segera, karena data dan instruksi yang dibutuhkan telah dipersiapkan sebelumnya.. Dalam hal speculative execution, artinya CPU akan menggunakan melakukan perhitungan pada pipeline

yang berbeda berdasarkan kemungkinan yang diperkirakan oleh komputer. Jika kemungkinan yang dilakukan oleh komputer tepat, maka hasilnya sudah bisa diambil langsung dan tinggal melanjutkan perintah berikutnya, sedangkan jika kemungkinan yang diperkirakan oleh komputer tidak tepat, maka akan dilaksanakan kemungkinan lain sesuai dengan logika instruksi tersebut. Teknik yang digunakan untuk pipeline dan superscalar ini bisa melaksanakan Branch Prediction dan speculative execution tentunya membutuhkan ekstra transistor yang tidak sedikit untuk hal tersebut. Sebagai perbandingan, komputer yang membangkitkan pemrosesan pada PC pertama yang dikeluarkan oleh IBM pada mesin 8088 memiliki sekitar 29.000 transistor. Sedangkan pada mesin Pentium III, dengan teknologi superscalar dan superpipeline, mendukung branch prediction, speculative execution serta berbagai kemampuan lainnya memiliki sekitar 7,5 juta transistor. BeberapA CPU terkini lainnya seperti HP 8500 memiliki sekitar 140 juta transistor.

**Ciri-Ciri Prosesor RISC, yaitu:**

· Instruksi berukuran tunggal

· Ukuran yang umum adalah 4 byte

· Jumlah pengalamatan data sedikit, biasanya kurang dari 5 buah.

· Tidak terdapat pengalamatan tak langsung yang mengharuskan melakukan sebuah akses

memori agar memperoleh alamat operand lainnya dalam memori.

· Tidak terdapat operasi yang menggabungkan operasi load/store dengan operasi aritmatika,

seperti penambahan ke memori dan penambahan dari memori.

· Tidak terdapat lebih dari satu operand beralamat memori per instruksi

· Tidak mendukung perataan sembarang bagi data untuk operasi load/ store.

· Jumlah maksimum pemakaian memori manajemen bagi suatu alamat data adalah sebuah

instruksi .

· Jumlah bit bagi integer register spesifier sama dengan 5 atau lebih, artinya sedikitnya 32

buah register integer dapat direferensikan sekaligus secara eksplisit.

· Jumlah bit floating point register spesifier sama dengan 4 atau lebih, artinya sedikitnya 16

register floating point dapat direferensikan sekaligus secara eksplisit.

**Karakteristik- Karakteristik pada umunya, yaitu:**

· Siklus mesin ditentukan oleh waktu yang digunakan untuk mengambil dua buah operand

dari register, melakukan operasi ALU, dan menyimpan hasil operasinya kedalam register,

dengan demikian instruksi mesin RISC tidak boleh lebih kompleks dan harus dapat mengeksekusi secepat mikroinstruksi pada mesin-mesin CISC. Dengan menggunakan instruksi sederhana atau instruksi satu siklus hanya dibutuhkan satu mikrokode atau tidak sama sekali, instruksi mesin dapat dihardwired. Instruksi seperti itu akan dieksekusi lebih cepat dibanding yang sejenis pada yang lain karena tidak perlu mengakses penyimapanan kontrol mikroprogram saat eksekusi instruksi berlangsung.

· Operasi berbentuk dari register-ke register yang hanya terdiri dari operasi load dan store yang mengakses memori . Fitur rancangan ini menyederhanakan set instruksi sehinggamenyederhanakan pula unit control. Keuntungan lainnya memungkinkan optimasi

pemakaian register sehingga operand yang sering diakses akan tetap ada di penyimpan

berkecepatan tinggi. Penekanan pada operasi register ke register merupakan hal yang unik bagi perancangan RISC.

· Penggunaan mode pengalamatan sederhana, hampir sama dengan instruksi menggunakan pengalamatan register,. Beberapa mode tambahan seperti pergeseran dan pe-relatif dapatdimasukkan selain itu banyak mode kompleks dapat disintesis pada perangkat lunakdibanding yang sederhana, selain dapat menyederhanakan sel instruksi dan unit kontrol.

· Penggunaan format-format instruksi sederhana, panjang instruksinya tetap dan disesuaikan

dengan panjang word. Fitur ini memiliki beberapa kelebihan karena dengan menggunakan

field yang tetap pendekodean opcode dan pengaksesan operand register dapat dilakukan

secara bersama-sama

Perbedaan antara RISC dan CISC



**RISC**

**Sejarah**

**Reduced Instruction Set Computing** (RISC) atau “Komputasi set instruksi yang disederhanakan” pertama kali digagas oleh John Cocke, peneliti dari IBM di Yorktown, New York pada tahun 1974 saat ia membuktikan bahwa sekitar 20% instruksi pada sebuah prosesor ternyata menangani sekitar 80% dari keseluruhan kerjanya. Komputer pertama yang menggunakan konsep RISC ini adalah IBM PC/XT pada era 1980-an. Istilah RISC sendiri pertama kali dipopulerkan oleh David Patterson, pengajar pada University of California di Berkely.

**Definisi**

RISC, yang jika diterjemahkan berarti “Komputasi Kumpulan Instruksi yang Disederhanakan”, merupakan sebuah [arsitektur komputer](http://id.wikipedia.org/wiki/Arsitektur_komputer) atau arsitektur komputasi modern dengan instruksi-instruksi dan jenis eksekusi yang paling sederhana. Arsitektur ini digunakan pada komputer dengan kinerja tinggi, seperti komputer vektor. Selain digunakan dalam komputer vektor, desain ini juga diimplementasikan pada prosesor komputer lain, seperti pada beberapa mikroprosesor [Intel 960](http://id.wikipedia.org/w/index.php?title=Intel_960&action=edit&redlink=1), [Itanium](http://id.wikipedia.org/wiki/Itanium) ([IA64](http://id.wikipedia.org/w/index.php?title=IA64&action=edit&redlink=1)) dari [Intel Corporation](http://id.wikipedia.org/wiki/Intel), [Alpha AXP](http://id.wikipedia.org/wiki/Alpha_AXP) dari [DEC](http://id.wikipedia.org/wiki/Digital_Equipment_Corporation), [R4x00](http://id.wikipedia.org/w/index.php?title=R4x00&action=edit&redlink=1) dari [MIPS Corporation](http://id.wikipedia.org/wiki/MIPS), [PowerPC](http://id.wikipedia.org/w/index.php?title=PowerPC&action=edit&redlink=1) dan [Arsitektur POWER](http://id.wikipedia.org/w/index.php?title=Arsitektur_POWER&action=edit&redlink=1) dari [International Business Machine](http://id.wikipedia.org/wiki/IBM). Selain itu, RISC juga umum dipakai pada [Advanced RISC Machine (ARM)](http://id.wikipedia.org/w/index.php?title=ARM&action=edit&redlink=1) dan [StrongARM](http://id.wikipedia.org/w/index.php?title=ARM&action=edit&redlink=1) (termasuk di antaranya adalah Intel [XScale](http://id.wikipedia.org/w/index.php?title=XScale&action=edit&redlink=1)), [SPARC](http://id.wikipedia.org/w/index.php?title=Sun_SPARC&action=edit&redlink=1)dan [UltraSPARC](http://id.wikipedia.org/w/index.php?title=Sun_SPARC&action=edit&redlink=1) dari [Sun Microsystems](http://id.wikipedia.org/wiki/Sun_Microsystems), serta [PA-RISC](http://id.wikipedia.org/w/index.php?title=PA-RISC&action=edit&redlink=1) dari [Hewlett-Packard](http://id.wikipedia.org/wiki/Hewlett-Packard).

Selain RISC, desain [*Central Processing Unit*](http://id.wikipedia.org/wiki/CPU) yang lain adalah [CISC](http://id.wikipedia.org/wiki/CISC) (*Complex Instruction Set Computing*), yang jika diterjemahkan ke dalam Bahasa Indonesia berarti Komputasi Kumpulan Instruksi yang kompleks atau rumit.

**CISC**

**Definisi**

***Complex instruction-set computing*** atau ***Complex Instruction-Set Computer*** (**CISC**; “Kumpulan instruksi komputasi kompleks”) adalah sebuah arsitektur dari set instruksi dimana setiap instruksi akan menjalankan beberapa operasi tingkat rendah, seperti pengambilan dari memory, operasi aritmetika, dan penyimpanan ke dalam memory, semuanya sekaligus hanya di dalam sebuah instruksi. Karakteristik CISC dapat dikatakan bertolak-belakang dengan RISC.

Sebelum proses RISC didesain untuk pertama kalinya, banyak arsitek komputer mencoba menjembatani celah semantik”, yaitu bagaimana cara untuk membuat set-set instruksi untuk mempermudah pemrograman level tinggi dengan menyediakan instruksi “level tinggi” seperti pemanggilan procedure, proses pengulangan dan mode-mode pengalamatan kompleks sehingga struktur data dan akses array dapat dikombinasikan dengan sebuah instruksi. Karakteristik CISC yg “sarat informasi” ini memberikan keuntungan di mana ukuran program-program yang dihasilkan akan menjadi relatif lebih kecil, dan penggunaan memory akan semakin berkurang. Karena CISC inilah biaya pembuatan komputer pada saat itu (tahun 1960) menjadi jauh lebih hemat.

Memang setelah itu banyak desain yang memberikan hasil yang lebih baik dengan biaya yang lebih rendah, dan juga mengakibatkan pemrograman level tinggi menjadi lebih sederhana, tetapi pada kenyataannya tidaklah selalu demikian. Contohnya, arsitektur kompleks yang didesain dengan kurang baik (yang menggunakan kode-kode mikro untuk mengakses fungsi-fungsi hardware), akan berada pada situasi di mana akan lebih mudah untuk meningkatkan performansi dengan tidak menggunakan instruksi yang kompleks (seperti instruksi pemanggilan procedure), tetapi dengan menggunakan urutan instruksi yang sederhana.

Satu alasan mengenai hal ini adalah karena set-set instruksi level-tinggi, yang sering disandikan (untuk kode-kode yang kompleks), akan menjadi cukup sulit untuk diterjemahkan kembali dan dijalankan secara efektif dengan jumlah transistor yang terbatas. Oleh karena itu arsitektur -arsitektur ini memerlukan penanganan yang lebih terfokus pada desain prosesor. Pada saat itu di mana jumlah transistor cukup terbatas, mengakibatkan semakin sempitnya peluang ditemukannya cara-cara alternatif untuk optimisasi perkembangan prosesor. Oleh karena itulah, pemikiran untuk menggunakan desain RISC muncul pada pertengahan tahun 1970 (Pusat Penelitian Watson IBM 801 – IBMs)

Contoh-contoh prosesor CISC adalah System/360, VAX, PDP-11, varian Motorola 68000 , dan CPU AMD dan Intel x86.

Istilah RISC dan CISC saat ini kurang dikenal, setelah melihat perkembangan lebih lanjut dari desain dan implementasi baik CISC dan CISC. Implementasi CISC paralel untuk pertama kalinya, seperti 486 dari Intel, AMD, Cyrix, dan IBM telah mendukung setiap instruksi yang digunakan oleh prosesor-prosesor sebelumnya, meskipun efisiensi tertingginya hanya saat digunakan pada subset x86 yang sederhana (mirip dengan set instruksi RISC, tetapi tanpa batasan penyimpanan/pengambilan data dari RISC). Prosesor-prosesor modern x86 juga telah menyandikan dan membagi lebih banyak lagi instruksi-instruksi kompleks menjadi beberapa “operasi-mikro” internal yang lebih kecil sehingga dapat instruksi-instruksi tersebut dapat dilakukan secara paralel, sehingga mencapai performansi tinggi pada subset instruksi yang lebih besar.

Tabel CISC vs RISC

Karakteristik RISC dan CISC

>> Rancangan RISC dapat memperoleh keuntungan dengan mengambil sejumlah feature CISC dan Rancangan CISC dapat memperoleh keuntungan dengan mengambil sejumlah feature RISC.

>> Hasilnya adalah bahwa sejumlah rancangan RISC yang terbaru, yang dikenal sebagai PowerPC, tidak lagi “murni” RISC dan rancangan CISC yang terbaru, yang dikenal sebagai Pentium, memiliki beberapa karakteristik RISC.

Ciri-ciri RISC

>> Instruksi berukuran tunggal

>> Ukuran yang umum adalah 4 byte.

>> Jumlah mode pengalamatan data yang sedikit, biasanya kurang dari lima buah.

>> Tidak terdapat pengalamatan tak langsung.

>> Tidak terdapat operasi yang menggabungkan operasi load/store dengan operasi aritmetika (misalnya, penambahan dari memori, penambahan ke memori).

Ciri-ciri CISC

>> Penekanan pada perangkat keras (hardware)

>> Termasuk instruksi kompleks multi-clock

>> Memori-ke-memori: “LOAD” dan “STORE” saling bekerjasama

>> Ukuran kode kecil, kecepatan rendah

>> Transistor digunakan untuk menyimpan instruksi-instruksi kompleks

Contoh aplikasi RISC

Atmel AVR adalah modifikasi arsitektur *Harvard* 8-bit RISC *single chip microcontroller*  yang di kembangkan oleh atmel pada 1996 . AVR adalah satu dari keluarga mikrokontroller pertama yang menggunakan memori on-chip flash untuk penyimpanan program, sebagai lawan untuk progammable one time pada ROM, EPROM atau EEPROM digunakan oleh mikrokontroller lain.

RISC represents a design philosophy for the ISA (Instruction Set Architecture) and the CPU microarchitecture that implements that ISA.

RISC is not a set of rules; there is no “pure RISC” design.

The acronym CISC, standing for “**C**omplex **I**nstruction **S**et **C**omputer”, is a term applied by the proponents of RISC to computers that do not follow that design.

The first designed called “RISC” date to the early 1980’s. The movement began with two experimental designs

 The IBM 801 developed by IBM in 1980

 The RISC I developed by UC Berkeley in 1981.

We should note that the original RISC machine was probably the CDC–6400 designed and built by Mr. Seymour Cray, then of the Control Data Corporation.

In designing a CPU that was simple and very fast, Mr. Cray applied many of the techniques that would later be called “RISC” without himself using the term.

**Why CISC?**

Early CPU designs could have followed the RISC philosophy, the advantages of which were apparent early. Why then was the CISC design followed?

Here are two reasons:

1. CISC designs make more efficient use of memory. In particular, the “code
 density” is better, more instructions per kilobyte.

 After all, memory is very expensive and prone to failure.

2. CISC designs close the “semantic gap”; they produce an ISA with
 instructions that more closely resemble those in a higher–level language.
 This should provide better support for the compilers.

**Memory Is Expensive, Isn’t It?**

Here is an updated table for memory expenses. It will soon be obsolete.

|  |  |  |  |
| --- | --- | --- | --- |
| Vendor | Date | Cost of memory | Cost of disk drive |
| MC6800 | 1974 | $500 for 16KB RAM | $55,000 for 40 MB |
| MC68000 | 1979 | $200 for 64 KB RAM | $5,000 for 10 MB |
| Micron 4 | 4/10/2002 | $49 for 128 MB RAM | $149 for 20 GB |
| HewlettPackard | 3/13/2007 | $120 for 1 GB$400 for 3 GB | $30 for 320 GB$50 for 400 GB |

Note that the cost of each of random access memory and disk memory has dropped by about four orders of magnitude since 1974.

The costs of memory no longer justify a complex design.

**What About Support for High Level Languages?**

Experimental studies conducted in 1971 by Donald Knuth and
in 1982 by David Patterson showed that

 1) nearly 85% of a programs statements were simple assignment,
 conditional, or procedure calls.

 2) None of these required a complicated instruction set.

The following table summarizes some results from experimental studies of code emitted by typical compilers of the 1970’s and 1980’s.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Language | Pascal | FORTRAN | Pascal | C | SAL |
| Workload | Scientific | Student | System | System | System |
| Assignment | 74 | 67 | 45 | 38 | 42 |
| Loop | 4 | 3 | 5 | 3 | 4 |
| Call | 1 | 3 | 15 | 12 | 12 |
| If | 20 | 11 | 29 | 43 | 36 |
| GOTO | 2 | 9 | -- | 3 | -- |
| Other |  | 7 | 6 | 1 | 6 |

**Summary of High–Level Language Support**

Here is a summary of the results of the works cited above.

1. As time progresses, programs will be more and more written in a
 high–level language, with assembly reserved for legacy programs.

2. The compilers now written do not make use of complex Instruction
 Set Architectures, but tend to use very simple constructs:
 Assignments, Jumps, Calls, and simple math.

3. What compiler writers would really like is provision of a large
 number of general purpose registers.

4. A more complex ISA implies a slower control unit, as the clock rate
 must be set for the data path timing of the slowest instruction in the
 ISA, even if it is never used in actual code.

**The RISC Design Strategies**

The basic RISC principle: “A simpler CPU is a faster CPU”.

The focus of the RISC design is reduction of the number and complexity
of instructions in the ISA.

A number of the more common strategies include:

 1) Fixed instruction length, generally one word.
 This simplifies instruction fetch.

 2) Simplified addressing modes.

 3) Fewer and simpler instructions in the instruction set.

 4) Only load and store instructions access memory;
 no add memory to register, add memory to memory, etc.

 5) Let the compiler do it. Use a good compiler to break complex
 high-level language statements into a number of simple assembly
 language statements.

**More RISC Design Principles**

1. Use optimizing compilers that issue simpler instructions.
 Complex compilers are easy to develop and test.

2. Emphasize an ISA that allows simple and efficient instruction decoding.

3. Microcode is not magic. It should be used sparingly.
 Those instructions that require microcode for their implementation should
 be inspected closely and considered for replacement by simpler instructions.

4. Operations that access memory should be avoided as memory access
 is a very time consuming operation.

5. All arithmetic operations should take only registers as operands.
 Only register load from memory and register store to memory should
 access memory addresses.

**Comparison of RISC and CISC**

This table is taken from an IEEE tutorial on RISC architecture.

|  |  |  |
| --- | --- | --- |
|  | CISC Type Computers | RISC Type |
|  | IBM 370/168 | VAX-11/780 | Intel 8086 | RISC I | IBM 801 |
| Developed | 1973 | 1978 | 1978 | 1981 | 1980 |
| Instructions | 208 | 303 | 133 | 31 | 120 |
| Instruction size (bits) | 16 – 48 | 16 – 456 | 8 – 32 | 32 | 32 |
| Addressing Modes | 4 | 22 | 6 | 3 | 3 |
| General Registers | 16 | 16 | 4 | 138 | 32 |
| Control Memory Size | 420 Kb | 480 Kb | Not given | 0 | 0 |
| Cache Size | 64 Kb | 64 Kb | Not given | 0 | Not given |

**Experience on the VAX
by Digital Equipment Corporation (DEC)**

The VAX–11/780 is the classic CISC design with
a very complex instruction set.

DEC experimented with two different implementations of the VAX architecture.
These are the VLSI VAX and the MicroVAX–32.

The VLSI VAX implemented the entire VAX instruction set.

The MicroVAX–32 design was based on the following observation about the more complex instructions.

 they account for 20% of the instructions in the VAX ISA,

 they account for 60% of the microcode, and

 they account for less than 0.2% of the instructions executed.

The MicroVAX implemented these instructions in system software.

**Results of the DEC Experience**

The VLSI VAX uses five to ten times the resources of the MicroVAX.

The VLSI VAX is only about 20% faster than the MicroVAX.

Here is a table from their study.

|  |  |  |
| --- | --- | --- |
|  | VLSI VAX | MicroVAX 32 |
| VLSI Chips | 9 | 2 |
| Microcode | 480K | 64K |
| Transistors | 1250K | 101K |

Notes: 1. The MicroVAX used two VLSI chips”
 One for the basic instruction set, and
 one for the optional floating–point processor.

 2. Note that two MicroVAX–32 computers, used together,
 might have about 160% of the performance of the VLSI VAX
 at about half the cost.