Arsitektur dan Organisasi Komputer:
Set Instruksi (Mode Pengalamatan & Format)

Ir. Heru Nurwarsito, M.Kom
Barlian Henryranu P, ST, MT
Eko Saksi Pramukantoro, S.Kom, M.Kom

1. PENDAHULUAN
   - Pengantar
   - Tujuan
   - Latar Belakang
2. KARAKTERISTIK INSTRUKSI MESIN
3. JENIS OPERAND
4. JENIS DATA INTEL X86 DAN ARM
5. JENIS OPERASI
6. JENIS OPERASI INTEL X86 DAN ARM

10

SELF-PROPAGATING ENTREPRENEURIAL EDUCATION DEVELOPMENT

1. PENDAHULUAN

1.1 Pengantar

Unsur-unsur penting dari sebuah instruksi komputer adalah opcode, yang menentukan operasi yang akan dilakukan; sumber dan tujuan referensi operan, yang menentukan lokasi input dan output untuk operasi; dan instruksi berikutnya, yang biasanya implisit. Opcode menentukan operasi di salah satu kategori umum berikut: operasi aritmatika dan logika; movement data antara dua register, register dan lokasi memori, atau dua memori; I / O; dan kontrol. Operan menentukan lokasi register atau memori data operan. Jenis data mungkin alamat, angka, karakter, atau data logika. Fitur umum arsitektur pada prosesor adalah penggunaan stack, yang mungkin atau tidak mungkin terlihat oleh programmer. Stack ini digunakan untuk mengelola panggilan prosedur dan dapat diberikan sebagai bentuk alternatif pengalamatan memori. Operasi dasar stack adalah PUSH, POP, dan operasi pada satu atau dua lokasi atas stack. Stack biasanya diimplementasikan untuk tumbuh dari alamat yang lebih tinggi ke alamat yang lebih rendah. Pengalamatan prosesor dapat dikategorikan sebagai big endian, little endian, atau bi-endian. Nilai numerik multibyte disimpan pada MSB tersimpan pada alamat numerik terendah dalam big-endian mode. Model Little-endian menyimpan MSB dalam alamat numerik tertinggi. Prosesor bi-endian dapat melakukan keduanya.

1.2 Tujuan

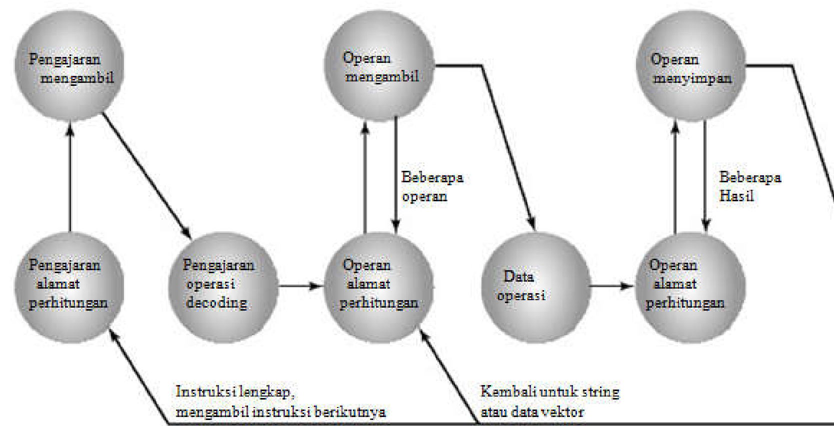
1. Mahasiswa dapat memahami karakeristis opcode
2. Mahasiswa dapat memahami fungsi opcode
3. Mahasiswa mampu menggunakan opcode dalam operasi-operasi prosesor

1.3 Latar Belakang

Banyak dari apa yang dibahas dalam buku ini tidak mudah dijelaskan bagi pengguna atau programmer komputer. Jika programmer menggunakan bahasa tingkat tinggi, seperti Pascal atau Ada, sangat sedikit dari arsitektur dari mesin yang dapat terlihat. Salah satu batas di mana desainer komputer dan programmer komputer dapat melihat mesin yang sama adalah dengan set instruksi mesin. Dari sudut pandang desainer, set instruksi mesin menyediakan kebutuhan fungsional untuk prosesor: penerapan prosesor adalah tugas yang sebagian besar melibatkan menerapkan set instruksi mesin. Pengguna yang memilih untuk program dalam bahasa mesin akan menjadi mudah memahami struktur register dan memori, jenis data langsung didukung oleh mesin, dan fungsi dari ALU. Penjelasan mengenai set instruksi mesin komputer berjalan jauh menuju menjelaskan prosesor komputer. Dengan demikian, kita fokus pada instruksi mesin dalam bab ini dan berikutnya.

KARAKTERISTIK INSTRUKSI MESIN

Operasi prosesor ditentukan oleh instruksi yang dijalankan, disebut sebagai mesin instruksi atau instruksi komputer. Kumpulan instruksi pada prosesor yang dapat mengeksekusi disebut sebagai set instruksi.



Gambar 10.1 Diagram State Siklus Instruksi

Elemen Instruksi Mesin

Setiap instruksi harus berisi informasi yang diperlukan oleh prosesor untuk eksekusi. Gambar 10.1, yang mengulangi Gambar 3.6, menunjukkan langkah-langkah dalam instruksi eksekusi dan, dengan implikasi, mendefinisikan elemen dari instruksi mesin. Elemen-elemen tersebut adalah sebagai berikut:

- Operasi kode: Menentukan operasi yang akan dilakukan (misalnya, ADD, I / O).
- Operasi ini ditentukan oleh kode biner, yang dikenal sebagai kode operasi,atau opcode.
- Sumber operan referensi: Operasi mungkin melibatkan satu atau lebih sumber operan, yaitu operand yang merupakan masukan untuk operasi.
- Hasil referensi operan: Operasi mungkin menghasilkan hasil.
- Berikutnya instruksi referensi: Ini memberitahu prosesor mana untuk mengambil berikutnya dalam konstruksi setelah pelaksanaan instruksi ini selesai.

Alamat dari instruksi berikutnya yang akan diambil bisa berupa alamat nyata atau alamat virtual, tergantung pada arsitekturnya. Secara umum, perbedaan adalah transparan untuk arsitektur set instruksi. Dalam kebanyakan kasus, instruksi berikutnya yang diambil segera mengikuti instruksi saat ini. Dalam kasus tersebut, tidak ada referensi eksplist ke instruksi berikutnya. Ketika sebuah referensi eksplisit diperlukan, maka memori utama atau alamat memori virtual harus diberikan. Sumber dan hasil operand dapat berada di salah satu dari empat bidang:

•**Utama atau memori virtual:** Seperti referensi instruksi berikutnya, utama atau alamat memori virtual terlalu konseptual harus diberikan.

•**Prosesor register:** Dengan pengecualian, prosesor berisi satu atau lebih registers yang dapat direferensikan oleh instruksi mesin. Jika hanya satu register ada,

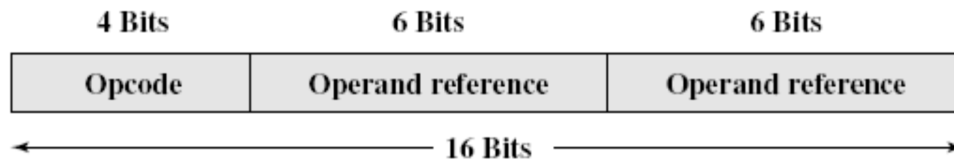


Figure 10.2 A Simple Instruction Format

referensi untuk itu mungkin implisit. Jika lebih dari satu register ada, maka setiap register diberikan sebuah nama yang unik atau nomor, dan instruksi harus berisi nomor dari register yang diinginkan.

•**Segera:** Nilai dari operan terkandung dalam field dalam instruksi yang dieksekusi.

•**I / O device:** Instruksi harus menentukan modul I / O dan perangkat untuk operasi. Jika memori-mapping I / O digunakan, ini hanyalah utama atau alamat memori virtual.

Instruksi Representasi

Dalam komputer, setiap instruksi diwakili oleh urutan bit. Instruksi diplot-plot, sesuai dengan unsur utama instruksi. Sebuah contoh sederhana dari format instruksi ditunjukkan pada Gambar 10.2. Sebagai contoh lain, format instruksi IAS ditunjukkan pada Gambar 2.2. Kebanyakan set instruksi menggunakan lebih dari satu format. Selama eksekusi instruksi, instruksi dibaca ke register instruksi (IR) dalam prosesor. Prosesor harus dapat mengekstrak data dari berbagai bidang instruksi untuk melakukan operasi yang diperlukan.

Sulit bagi programmer dan pembaca buku teks memahami representasi pengalamatan biner dari instruksi mesin. Dengan demikian, untuk lebih mudahnya digunakan *representasi simbolis* instruksi mesin. Contohnya telah digunakan untuk set instruksi IAS, pada Tabel 2.1.

Opcode diwakili oleh singkatan, yang disebut *mnemonik,* yang menunjukkan operasi. Contoh:

ADD	Menambahkan
SUB	Mengurangi
MUL	Mengalikan
DIV	Membagi
LOAD	Load data dari memori
STOR	Simpan data ke memori

Operand juga diwakili secara simbolis. Sebagai contoh, instruksi

ADD R, Y

mungkin berarti menambahkan nilai yang terkandung di lokasi data Y ke isi register R. Dalam contoh ini, Y mengacu ke lokasi sebuah alamat memori, dan R mengacu pada register

tertentu. Perhatikan bahwa operasi tersebut dilakukan pada isi dari lokasi, bukan pada alamatnya.

Dengan demikian, Hal ini mungkin untuk menulis sebuah program bahasa mesin dalam bentuk simbolis. Setiap simbol opcode memiliki representasi biner tetap, dan programmer menentukan lokasi setiap simbol operan. Sebagai contoh, programmer mungkin mulai dengan daftar definisi:

X=513

Y=514

dan sebagainya. Sebuah program sederhana akan menerima masukan simbol, mengkonversi opcode dan operan ke bentuk biner, dan membangun instruksi biner mesin. Bahasa mesin programer jarang ke titik noneksistensi. Kebanyakan program ini ditulis dalam bahasa tingkat tinggi atau bahasa assembly. Namun, bahasa simbolis mesin tetap menjadi alat yang berguna untuk mendeskripsikan instruksi mesin, dan kami akan menggunakannya untuk tujuan itu.

Jenis Instruksi

Mempertimbangkan sebuah instruksi bahasa tingkat tinggi yang dapat disajikan dalam bahasa seperti BASIC atau FORTRAN. Sebagai contoh,

X=X+Y

Pernyataan ini menginstruksikan komputer untuk menambah nilai yang disimpan dalam Y untuk nilai disimpan di X dan meletakkan hasilnya di X. Bagaimana hal ini dicapai dengan mesin instruksi? Mari kita asumsikan bahwa variabel X dan Y sesuai dengan lokasi 513 dan 514. Jika kita mengasumsikan set instruksi mesin sederhana, operasi ini bisa tercapai dengan tiga instruksi:

1. Memuat register dengan isi dari lokasi memori 513.
2. Tambahkan isi dari lokasi memori 514 untuk register.
3. Menyimpan isi dari register dalam lokasi memori 513.

Seperti dapat dilihat, instruksi DASAR tunggal mungkin membutuhkan tiga mesin instruksi. Ini merupakan hubungan antara bahasa tingkat tinggi dan bahasa mesin. Sebuah bahasa tingkat tinggi menyatakan operasi dalam bentuk aljabar, dengan menggunakan variabel. Sebuah bahasa mesin mengungkapkan operasi dalam bentuk dasar dalam volving movement data ke atau dari register.

Dengan contoh sederhana ini sebagai pengantar kita, marilah kita mempertimbangkan jenis instruksi yang harus dimasukkan dalam komputer. Sebuah komputer harus memiliki satu set instruksi yang memungkinkan pengguna untuk merumuskan setiap tugas

pengolahan data. Cara lain untuk melihatnya adalah dengan mempertimbangkan kemampuan sebuah bahasa pemrograman tingkat tinggi. Apa pun program yang ditulis dalam bahasa tingkat tinggi harus diterjemahkan ke dalam bahasa mesin akan dieksekusi. Dengan demikian, set instruksi mesin harus cukup untuk mengekspresikan salah satu instruksi dari bahasa tingkat tinggi. Dengan demikian kita dapat mengkategorikan jenis instruksi sebagai berikut:

- **Pengolahan data:** Aritmatika dan logika instruksi
- **Penyimpanan data:** Movement data masuk atau keluar dari register dan atau lokasi memori
- **Data movement:** Instruksi I / O
- **Control:** Test dan instruksi percabangan

Instruksi *Aritmatika* menyediakan kemampuan komputasi untuk pengolahan Data numeric. Instruksi *Logika* (Boolean) beroperasi pada bit-bit dari sebuah word sebagai bit daripada sebagai nomor, dengan demikian, mereka memberikan kemampuan untuk memproses jenis data pengguna lainnya yang mungkin ingin digunakan. Operasi ini dilakukan terutama pada data dalam register prosesor. Oleh karena itu, harus ada instruksi memori untuk menggerakkan data antara memori dan register. instruksi *I / O* yang dibutuhkan untuk mentransfer program dan data ke dalam memori dan hasil perhitungan kembali kepada pengguna. Instruksi *Tes* yang digunakan untuk menguji nilai dari sebuah data word atau status dari computasi. Instruksi *Percabangan* digunakan untuk percabangan untuk satu set instruksi yang berbeda tergantung pada keputusan yang dibuat.

Jumlah Alamat

Salah satu cara tradisional untuk menggambarkan arsitektur prosesor adalah dengan jumlah alamat yang terkandung dalam setiap instruksi. Dimensi ini telah menjadi kurang signifikan dengan meningkatnya kompleksitas desain prosesor. Namun demikian, pada saat ini berguna untuk menarik dan menganalisis perbedaan ini.

Berapa jumlah maksimum alamat yang mungkin dalam sebuah instruksi? Terbukti, aritmatika dan logika instruksi akan membutuhkan operan. Secara tidak nyata semua operasi aritmatika dan logika yang baik unary (satu sumber operand) atau biner (dua operand sumber). Jadi, kita akan membutuhkan maksimal dua iklangaun untuk referensi sumber operand. Hasil operasi harus disimpan, menunjukkan alamat ketiga, yang mendefinisikan sebuah operan tujuan. Akhirnya, setelah penyelesaian instruksi, instruksi berikutnya harus diambil, dan alamatnya diperlukan.

Baris ini menunjukkan bahwa penalaran instruksi yang diminta dapat mengandung referensi empat alamat: operand dua sumber, satu operan tujuan, dan alamat dari instruksi berikutnya. Dalam sebagian besar arsitektur, instruksi yang paling memiliki satu, dua, atau tiga alamat operan, dengan alamat instruksi berikutnya yang implisit (diperoleh dari program counter). Kebanyakan arsitektur juga memiliki beberapa instruksi tujuan khusus dengan lebih operan. Sebagai contoh load beberapa instruksi dari arsitektur ARM, dijelaskan dalam Bab 11, menunjuk hingga 17 register operan di dalam sebuah instruksi.

Gambar 10.3 membandingkan satu, dua, dan tiga alamat instruksi yang dapat digunakan untuk menghitung Y =(A -B) [C +(D*E)]. Dengan tiga alamat, setiap instruksi menetapkan dua lokasi sumber operand dan lokasi operan tujuan. Karena kita memilih untuk tidak mengubah nilai salah satu lokasi operand, sebuah lokasi sementara, T, digunakan untuk menyimpan beberapa hasil sementara. Perhatikan bahwa ada empat instruksi dan bahwa ekspresi asli memiliki lima operan.

Format instruksi tiga alamat yang tidak umum karena memerlukan format instruksi relatif panjang untuk memegang tiga referensi alamat. Dengan dua instruksi alamat, dan untuk operasi biner, satu alamat harus melakukan tugas ganda sebagai kedua operand dan hasil. Dengan demikian, instruksi SUB Y, B melakukan perhitungan

Instruction		Comment
SUB	Y, A, B	$Y \leftarrow A - B$
MPY	T, D, E	$T \leftarrow D \times E$
ADD	T, T, C	$T \leftarrow T + C$
DIV	Y, Y, T	$Y \leftarrow Y \div T$

(a) Three-address instructions

Instruction		Comment
MOVE	Y, A	$Y \leftarrow A$
SUB	Y, B	$Y \leftarrow Y - B$
MOVE	T, D	$T \leftarrow D$
MPY	T, E	$T \leftarrow T \times E$
ADD	T, C	$T \leftarrow T + C$
DIV	Y, T	$Y \leftarrow Y \div T$

(b) Two-address instructions

Instruction		Comment
LOAD	D	$AC \leftarrow D$
MPY	E	$AC \leftarrow AC \times E$
ADD	C	$AC \leftarrow AC + C$
STOR	Y	$Y \leftarrow AC$
LOAD	A	$AC \leftarrow A$
SUB	B	$AC \leftarrow AC - B$
DIV	Y	$AC \leftarrow AC \div Y$
STOR	Y	$Y \leftarrow AC$

(c) One-address instructions

Figure 10.3 Programs to Execute $Y = \frac{A - B}{C + (D \times E)}$

Y-B dan menyimpan hasilnya dalam Y. Format dua alamat mengurangi ruang membutuhkan pemerintah tetapi juga memperkenalkan beberapa kejanggalan. Untuk menghindari mengubah nilai suatu operan, instruksi MOVE digunakan untuk memindahkan salah satu dari nilai hasil atau temporary lokasi sebelum melakukan operasi. Contoh program kami diperluas menjadi enam instruksi.

Yang sederhana adalah instruksi satu alamat. Cara kerjanya adalah alamat kedua harus implisit. Ini sering terjadi pada mesin sebelumnya, dengan alamat implisit menjadi daftar prosesor yang dikenal sebagai **accumulator** (AC). Akumulator juga mempertahankan salah satu operan dan digunakan untuk menyimpan hasilnya. Dalam contoh kita, delapan instruksi diperlukan untuk menyelesaikan operasi.

Hal ini, pada kenyataannya, mungkin melakukannya dengan nol alamat untuk beberapa instruksi. Instruksi nol alamat yang berlaku pada organisasi memori khusus, yang disebut sebuah *stack.* Stack adalah set last-in-first-out dari lokasi. Stack berada dalam lokasi yang diketahui dan sering, setidaknya atas dua elemen dalam register prosesor. Dengan demikian, instruksi nol alamat akan mereferensi dua elemen teratas dari stack.

Tabel 10.1 merangkum interpretasi untuk instruksi dengan nol, satu, dua, atau tiga alamat. Dalam setiap kasus dalam tabel, diasumsikan bahwa alamat dari instruksi berikutnya adalah implisit, dan bahwa satu operasi dengan dua sumber operan, dan satu operan adalah hasil.

Jumlah alamat per instruksi adalah dasar keputusan desain. Lebih sedikit alamat per hasil instruksi dalam instruksi yang lebih kuno, membutuhkan prosesor yang kurang kompleks. Ini juga menghasilkan instruksi yang lebih pendek. Di sisi lain, program berisi lebih dari semua instruksi, yang pada umumnya, eksekusi yang membutuhkan waktu lama

Table 10.1 Utilization of Instruction Addresses (Nonbranching Instructions)

Number of Addresses	Symbolic Representation	Interpretation
3	OP A, B, C	A ← B OP C
2	OP A, B	A ← A OP B
1	OP A	AC ← AC OP A
0	OP	T ← (T − 1) OP T

AC = accumulator
T = top of stack
(T − 1) = second element of stack
A, B, C = memory or register locations

memiliki program yang lebih kompleks. Juga, ada ambang batas penting antara instruksi satu alamat dan multiple-alamat. Dengan instruksi satu alamat, programmer umumnya hanya menyediakan satu register umum, accumulator. Dengan instruksi multi-alamat, memiliki beberapa register umum. Hal ini memungkinkan beberapa operasi yang akan dilakukan hanya pada register. Karena register lebih cepat daripada memori, hal ini mempercepat eksekusi. Untuk alasan fleksibilitas dan kemampuan untuk menggunakan beberapa register, paling kontemporer mesin rary menggunakan campuran dua dan tiga-alamat instruksi.

Desain trade-off memilih jumlah alamat instruksi yang rumit oleh faktor lain. Ada masalah apakah alamat referensi lokasi memori atau register. Karena ada register yang lebih sedikit, lebih sedikit bit diperlukan untuk referensi register. Juga, sebagaimana akan kita lihat

dalam bab berikutnya, mesin mungkin memberikan berbagai mode pengalamatan, dan spesifikasi dari mode mengambil bit satu atau lebih. Hasilnya adalah bahwa kebanyakan desain prosesor melibatkan berbagai format instruksi.

Desain Set Instruksi

Salah satu aspek yang paling menarik, dan paling dianalisis dalam desain komputer adalah desain set instruksi. Desain set instruksi sangat kompleks karena mempengaruhi banyak aspek dari sistem komputer. Set instruksi mendefinisikan banyak fungsi dilakukan oleh prosesor dan tentunya memiliki dampak yang signifikan terhadap pelaksanaan prosesor. Set instruksi adalah cara programmer mengontrol processor. Hal ini merupakan persyaratan yang perlu diperhatikan programmer dalam merancang set instruksi.

Ini mungkin akan mengejutkan Anda untuk mengetahui bahwa beberapa masalah yang paling mendasar berkaitan pada desain instruksi set tetap dalam perdebatan. Memang, dalam beberapa tahun terakhir, tingkat ketidaksepakatan tentang dasar ini telah benar-benar muncul. Yang paling penting masalah ini desain dasar meliputi:

- **Operasi repertoar:** Seberapa banyak dan operasi yang mana disediakan dan sekompleks apa seharusnya
- **Tipe data: J**enis data operasi dilakukan
- **Instruksi format:** Panjang instruksi (dalam bit), jumlah alamat, ukuran berbagai bidang, dan sebagainya
- **Register:** Jumlah register prosesor yang dapat direferensikan oleh instruksi, dan yang digunakan
- **Pengalamatan:** Mode alamat operand ditentukan

TIPE OPERAND

Instruksi mesin beroperasi pada data. kategori Umum yang paling penting dari data adalah:

- Alamat
- Number
- Karakter
- Logical data

Kita akan melihat, pembahasan mode pengalamatan di Bab 11, bahwa alamat pada kenyataannya dalam bentuk data. Dalam banyak kasus, beberapa perhitungan harus dilakukan pada operan referensi dalam sebuah instruksi untuk menentukan memori utama atau alamat virtual. Dalam konteks ini, alamat dapat dianggap sebagai unsigned integer.

Numbers

Semua bahasa mesin termasuk jenis data numerik. Bahkan dalam proses data nonnumeric, ada kebutuhan untuk number untuk bertindak sebagai counter, lebar field, dan sebagainya. Perbedaan penting antara nomor yang digunakan dalam matematika dan number umum yang disimpan dalam komputer adalah keterbatasan penyimpanan. Hal ini berlaku dalam dua pengertian. Pertama, ada batas dengan besarnya nomor representasi number pada mesin dan kedua, dalam kasus floating-point, batas untuk kepresisian number. Dengan demikian, programmer dihadapkan dengan konsekuensi pemahaman dari pembulatan, overflow, dan underflow. Tiga jenis data numerik yang umum di komputer:

- Biner integer atau biner fixed point
- Biner floating point
- Desimal

Meskipun semua operasi internal komputer adalah biner, pengguna awam tetap akrab dengan sistem angka desimal. Jadi, ada kebutuhan untuk mengkonversi dari desimal ke biner pada input dan dari biner ke desimal pada output. Untuk applications di mana ada banyak I / O dan relatif sedikit, relative perhitungan sederhana, lebih baik untuk menyimpan dan beroperasi pada angka-angka dalam bentuk decimal. Representasi paling umum untuk tujuan ini adalah **packed decimal.**

Dengan sistem desimal, setiap digit desimal diwakili oleh kode 4-bit, dalam obvious cara, dengan dua digit disimpan per byte. Jadi, 0 =0000, 1 =0001, Sebuah, 8 =1000, dan 9 =1001. Catatan bahwa ini adalah kode lebih efisien karena hanya 10 dari 16 yang mungkin untuk nilai 4-bit. Untuk membentuk angka, 4-bit kode ini dirangkai, biasanya dalam kelipatan 8 bit. Dengan demikian, kode untuk 246 adalah 0000 0010 0100 0110. Kode ini jelas kurang kompak dari representasi biner lurus, tapi ia menghindari terjadinya overhead. Angka negatif dapat direpresentasikan dengan memasukkan tanda 4-bit digit di kedua ujung kiri atau kanan dari serangkaian angka desimal packed. Standar tanda nilai adalah 1100 untuk positif (+) dan 1101 untuk negatif (-). Banyak mesin memberikan instruksi aritmatika untuk melakukan operasi langsung pada angka desimal packed. Algoritma yang cukup mirip dengan yang dijelaskan pada Bagian 9.3 tetapi harus mempertimbangkan operasi membawa desimal.

Karakter

Bentuk umum dari data adalah teks atau string. Sedangkan data tekstual yang paling convenient bagi manusia, mereka NOT dapat, dalam bentuk karakter, dengan mudah disimpan atau transmitted oleh pengolahan data dan sistem komunikasi. Sistem ini dirancang untuk data biner. Dengan demikian, sejumlah kode telah dirancang dimana karakter diwakili

oleh urutan bit. Mungkin contoh paling awal umum dari hal ini adalahdengan kode Morse. Hari ini, kode karakter yang paling umum digunakan dalam InternasionalReferensi Alphabet (IRA), disebut di Amerika Serikat sebagai Amerika Standard Kode untuk Informasi Interchange (ASCII, lihat Lampiran F). Setiap karakter dalam kode ini diwakili oleh pola 7-bit yang unik, dengan demikian, 128 karakter yang berbeda dapat diwakili. Ini adalah jumlah yang lebih besar daripada yang diperlukan untuk mewakili dicetak karakter, dan beberapa pola mewakili *mengontrol* karakter. Beberapa dari karakter kontrol harus dilakukan dengan mengendalikan pencetakan karakter pada sebuah halaman. Lainnya prihatin dengan prosedur komunikasi. IRA-encoded karakter hampir selalu disimpan dan dikirim menggunakan 8 bit per karakter. Bit ke delapan mungkin diatur ke 0 atau digunakan sebagai bit paritas untuk mendeteksi kesalahan. Dalam kasus terakhir, bit adalah diatur sedemikian rupa sehingga jumlah total 1s biner dalam setiap oktet selalu ganjil (paritas ganjil) atau selalu genap (even parity).

Catatan pada Tabel F.1 (Lampiran F) bahwa untuk 011XXXX sedikit pola IRA, yang angka 0 sampai 9 yang diwakili oleh biner setara mereka, 0000 melalui 1001, diyang paling kanan 4 bit. Ini adalah kode yang sama seperti desimal packed. Hal ini memudahkan konversi dari barang antara 7-bit IRA dan 4-bit representasi desimal packed. Lain kode yang digunakan untuk mengkodekan karakter adalah Binary Coded Perpanjangan Desimal Interchange Code (EBCDIC). EBCDIC digunakan pada mainframe IBM. Itu adalah kode 8-bit. Seperti IRA, EBCDIC kompatibel dengan desimal packed. Dalam kasus EBCDIC, kode 11110000 11111001 melalui mewakili angka 0 sampai 9.

Logical data

Biasanya, setiap word atau unit beralamat lain (byte, halfword, dan sebagainya) diperlakukan sebagai satu kesatuan data. Kadang-kadang berguna, namun, untuk mempertimbangkan n-bit unit sebagai terdiri dari n1-bit item data, setiap item memiliki nilai 0 atau 1. Bila data yang dilihat cara ini, mereka dianggap *logis* data.

Ada dua keuntungan ke tampilan berorientasi bit. Pertama, kita kadang-kadang dapat ingin menyimpan sebuah array dari item data Boolean atau biner, dimana setiap item dapat mengambil hanya pada nilai 1 (benar) dan 0 (false). Dengan data logis, memori dapat digunakan paling efisien untuk penyimpanan ini. Kedua, ada kesempatan ketika kami ingin memanipulasi bit dari item data. Misalnya, jika operasi floating-point diimplementasikan dalam perangkat lunak, kita harus mampu untuk menggeser bit yang signifikan dalam beberapa operasi. Lain form cukup: Untuk mengkonversi dari IRA ke desimal packed, kita perlu mengekstrak paling kanan 4 bit setiap byte.

Perhatikan bahwa, pada contoh sebelumnya, data yang sama kadang-kadang diperlakukan sebagai kali logis dan lainnya sebagai numerik atau teks. "Type" dari sebuah unit data adalah menghalangi ditambang oleh operasi yang dilakukan di atasnya. Meskipun hal ini tidak biasanya terjadi di bahasa tingkat tinggi, hampir selalu terjadi dengan bahasa mesin.

JENIS DATA INTEL X86 DAN ARM

x86 Tipe Data

X86 dapat pengalamatan jenis data 8 (byte), 16 (word), 32 (doubleword), 64 (quadword), dan 128 (dua quadword) bit panjangnya. Untuk memungkinkan fleksibilitas maksimum dalam struktur data dan penggunaan memori yang efisien, word-word NOT perlu bahkan selaras pada alamat bernomor; doublewords NOT perlu selaras pada alamat merata dibagi oleh 4; dan quadwords NOT perlu selaras pada alamat merata dibagi 8, dan sebagainya pada. Namun, ketika data yang diakses di bus 32-bit, transfer data berlangsung di unit doublewords, dimulai pada alamat yang habis dibagi 4. Prosesor ini mengubah permintaan untuk nilai-nilai sejajar ke urutan permintaan untuk transfer bus. Sebagai dengan semua mesin 80x86 Intel x86, menggunakan gaya little-endian, yaitu, byte paling signifikan disimpan dalam alamat terendah (lihat Lampiran 10B untuk diskusi endianness).

Byte, word, doubleword, quadword, dan quadword ganda disebut sebagai tipe data umum. Selain itu, x86 mendukung jajaran tertentu tipe data yang dikenali dan dioperasikan oleh instruksi tertentu. Tabel 10.2 merangkum jenis ini.

Table 10.2 x86 Data Types

Data Type	Description
General	Byte, word (16 bits), doubleword (32 bits), quadword (64 bits), and double quadword (128 bits) locations with arbitrary binary contents.
Integer	A signed binary value contained in a byte, word, or doubleword, using twos complement representation.
Ordinal	An unsigned integer contained in a byte, word, or doubleword.
Unpacked binary coded decimal (BCD)	A representation of a BCD digit in the range 0 through 9, with one digit in each byte.
Packed BCD	Packed byte representation of two BCD digits; value in the range 0 to 99.
Near pointer	A 16-bit, 32-bit, or 64-bit effective address that represents the offset within a segment. Used for all pointers in a nonsegmented memory and for references within a segment in a segmented memory.
Far pointer	A logical address consisting of a 16-bit segment selector and an offset of 16, 32, or 64 bits. Far pointers are used for memory references in a segmented memory model where the identity of a segment being accessed must be specified explicitly.
Bit field	A contiguous sequence of bits in which the position of each bit is considered as an independent unit. A bit string can begin at any bit position of any byte and can contain up to 32 bits.
Bit string	A contiguous sequence of bits, containing from zero to $2^{32} - 1$ bits.
Byte string	A contiguous sequence of bytes, words, or doublewords, containing from zero to $2^{32} - 1$ bytes.
Floating point	See Figure 10.4.
Packed SIMD (single instruction, multiple data)	Packed 64-bit and 128-bit data types

Gambar 10.4 mengilustrasikan tipe data numerik x86. Bilangan bulat ditandatangani di berpasangan melengkapi representasi dan mungkin 16, 32, atau 64 bit panjang. tipe Floating titik sebenarnya mengacu pada satu set jenis yang digunakan oleh unit floating-point dan dioperasikan oleh instruksi floating-point. Tiga floating-point perwakilan sesuai dengan IEEE 754 standar. SIMD packed (single-instruksi-multiple-data) tipe data adalah introdiproduksi dengan arsitektur x86 sebagai bagian dari perpanjangan dari set instruksi untuk Optimasi kinerja aplikasi multimedia. Ekstensi ini mencakup MMX (Ekstensi multimedia) dan SSE (Streaming SIMD extensions). Konsep dasar adalah bahwa beberapa operan yang packed ke dalam memori item tunggal direferensikan dan bahwa ini beberapa operan dioperasikan pada secara paralel. Tipe data adalah sebagai berikut:

•**Packed byte dan byte bilangan bulat penuh:** Bytes packed menjadi quadword 64-bit atau 128-bit ganda quadword, ditafsirkan sebagai bidang bit atau sebagai integer

•**Packed word dan bilangan bulat word penuh:** 16-bit packed menjadi 64-bit quadword atau 128-bit quadword ganda, ditafsirkan sebagai bidang bit atau sebagai integer

•**Packed doubleword dan packed doubleword integer:** 32-bit doublewords packed menjadi quadword 64-bit atau 128-bit quadword ganda, ditafsirkan sebagai bidang bit atau sebagai integer.

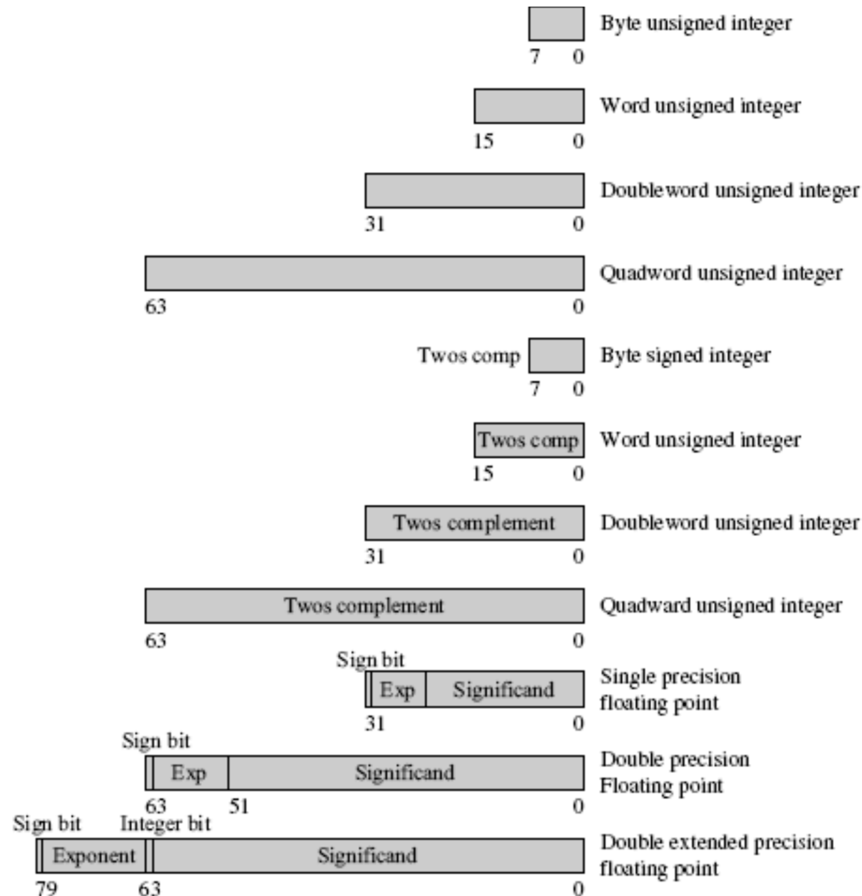


Figure 10.4 x86 Numeric Data Formats

•**Packed quadword dan packed qaudword integer:** Dua 64-bit quadwords packed menjadi quadword 128-bit ganda, ditafsirkan sebagai bidang bit atau sebagai integer

•**Packed satu-presisi floating-point dan packed ganda presisi floatingtitik:** Empat 32-bit floating-point atau dua 64-bit floating-point nilai packed menjadi quadword 128-bit ganda

Tipe Data ARM

Prosesor ARM mendukung tipe data 8 (byte), 16 (halfword), dan 32 (word) bit dalam panjang. Biasanya, halfword akses harus selaras dan word halfword mengakses seharusnya word selaras. Untuk akses upaya nonblok, arsitektur mendukung tiga alternatif.

•**Standar kasus**:

- Alamat diperlakukan sebagai terpotong, dengan alamat bit [01:00] diperlakukan sebagai nol untuk Word akses, dan bit alamat [0] diperlakukan sebagai nol untuk akses halfword.

- Load instruksi word tunggal ARM arsitektur didefinisikan untuk memutar benar data word-blok ditransfer oleh alamat word-blok non satu, dua, atau tiga byte tergantung pada nilai dari dua paling signifikan alamat bit.

•**Keselarasan pemeriksaan:** Apabila bit kontrol yang tepat diatur, data batalkan sinyal menunjukkan adanya kesalahan alignment untuk mencoba akses teralign.

•**Unaligned akses:** Bila pilihan ini diaktifkan, prosesor menggunakan satu atau lebih memori akses untuk menghasilkan transfer yang dibutuhkan byte yang berdewordn transparanently ke pemrogram

Untuk ketiga jenis data (byte, halfword, dan word) interpretasi unsigned didukung, di mana merupakan nilai integer, unsigned negatif. Semua tiga tipe data juga dapat digunakan untuk bilangan bulat pelengkap berpasangan ditandatangani.

Sebagian besar implementasi prosesor ARM NOT memberikan yang mengambang titik perangkat keras, yang menyimpan kekuasaan dan daerah. Jika aritmatika floating-point diperlukan dalam prosesor tersebut, maka harus diimplementasikan dalam perangkat lunak. ARM NOT mendukung sebuah opnasional floating-point coprocessor yang mendukung single dan double-presisi jenis floating point data yang didefinisikan dalam IEEE 754.

Endian SUPPORT Sedikit state (T-bit) dalam daftar sistem kontrol diatur dan dibersihkan di bawah kontrol program dengan menggunakan instruksi SETEND. E-bit mendefinisikan endian untuk memuat dan menyimpan data yang. Gambar 10.5 mengilustrasikan fungsi asosiasiciated dengan bit E-untuk beban word atau operasi toko. Mekanisme ini memungkinkan data dinamis efisien beban / toko untuk desainer sistem yang tahu bahwa mereka perlu access struktur data dalam endianness berlawanan dengan OS mereka / lingkungan. Mencatat bahwa alamat setiap data byte adalah tetap dalam memori. Namun, jalur byte dalam register yang berbeda.

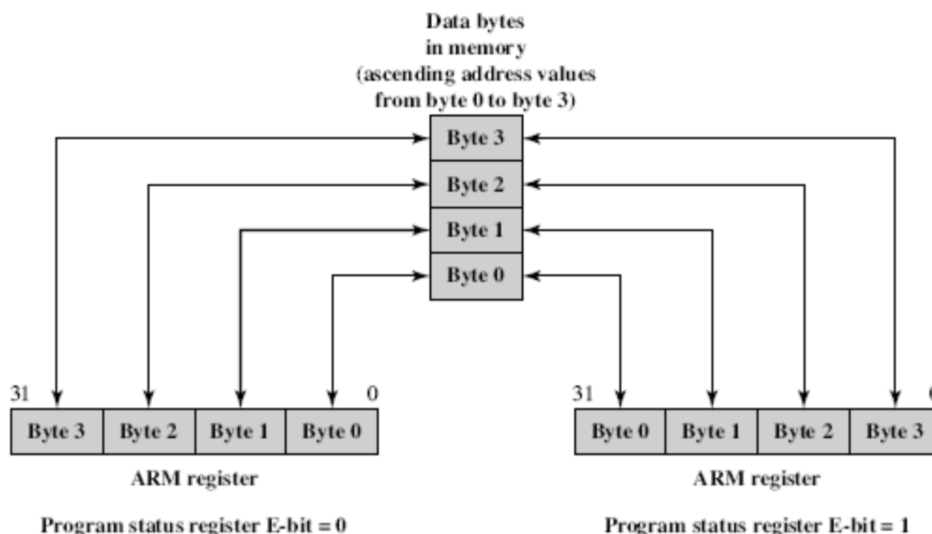


Figure 10.5 ARM Endian Support—Word Load/Store with E-Bit

JENIS OPERASI

Jumlah opcode berbeda sangat bervariasi dari mesin ke mesin. Namun, jenis umum yang sama operasi tersebut ditemukan pada semua mesin. Sebuah berguna dan khas kategorisasi adalah sebagai berikut:

- Transfer data
- Hitung
- Logis
- Konversi
- I / O
- Sistem kontrol
- Transfer control

Tabel 10.3 (berdasarkan [HAYE98]) daftar jenis instruksi umum di setiap kategori masing-masing. Bagian ini menyediakan survei singkat dari berbagai jenis operasi, bersama dengan diskusi singkat tentang tindakan yang diambil oleh prosesor untuk mengeksekusi khususnya jenis operasi (dirangkum dalam Tabel 10.4).

Type	Operation Name	Description
Logical	AND	Perform logical AND
	OR	Perform logical OR
	NOT (complement)	Perform logical NOT
	Exclusive-OR	Perform logical XOR
	Test	Test specified condition; set flag(s) based on outcome
	Compare	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome
	Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control, etc.
	Shift	Left (right) shift operand, introducing constants at end
	Rotate	Left (right) shift operand, with wraparound end
Transfer of Control	Jump (branch)	Unconditional transfer; load PC with specified address
	Jump Conditional	Test specified condition; either load PC with specified address or do nothing, based on condition
	Jump to Subroutine	Place current program control information in known location; jump to specified address
	Return	Replace contents of PC and other register from known location
	Execute	Fetch operand from specified location and execute as instruction; do not modify PC
	Skip	Increment PC to skip next instruction
	Skip Conditional	Test specified condition; either skip or do nothing based on condition
	Halt	Stop program execution
	Wait (hold)	Stop program execution; test specified condition repeatedly; resume execution when condition is satisfied
	No operation	No operation is performed, but program execution is continued

Type	Operation Name	Description
Logical	AND	Perform logical AND
	OR	Perform logical OR
	NOT (complement)	Perform logical NOT
	Exclusive-OR	Perform logical XOR
	Test	Test specified condition; set flag(s) based on outcome
	Compare	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome
	Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control, etc.
	Shift	Left (right) shift operand, introducing constants at end
	Rotate	Left (right) shift operand, with wraparound end
Transfer of Control	Jump (branch)	Unconditional transfer; load PC with specified address
	Jump Conditional	Test specified condition; either load PC with specified address or do nothing, based on condition
	Jump to Subroutine	Place current program control information in known location; jump to specified address
	Return	Replace contents of PC and other register from known location
	Execute	Fetch operand from specified location and execute as instruction; do not modify PC
	Skip	Increment PC to skip next instruction
	Skip Conditional	Test specified condition; either skip or do nothing based on condition
	Halt	Stop program execution
	Wait (hold)	Stop program execution; test specified condition repeatedly; resume execution when condition is satisfied
	No operation	No operation is performed, but program execution is continued
Input/Output	Input (read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
	Output (write)	Transfer data from specified source to I/O port or device
	Start I/O	Transfer instructions to I/O processor to initiate I/O operation
	Test I/O	Transfer status information from I/O system to specified destination
Conversion	Translate	Translate values in a section of memory based on a table of correspondences
	Convert	Convert the contents of a word from one form to another (e.g., packed decimal to binary)

Table 10.4 Processor Actions for Various Types of Operations

Data Transfer	Transfer data from one location to another
	If memory is involved: Determine memory address Perform virtual-to-actual-memory address transformation Check cache Initiate memory read/write
Arithmetic	May involve data transfer, before and/or after
	Perform function in ALU
	Set condition codes and flags
Logical	Same as arithmetic
Conversion	Similar to arithmetic and logical. May involve special logic to perform conversion
Transfer of Control	Update program counter. For subroutine call/return, manage parameter passing and linkage
I/O	Issue command to I/O module
	If memory-mapped I/O, determine memory-mapped address

Transfer Data

Jenis yang paling mendasar dari instruksi mesin adalah instruksi transfer data. Instruksi transfer data harus menentukan beberapa hal. Pertama, lokasi dari sumber dan tujuan operan harus ditentukan. Setiap lokasi dapat memori, register, atau bagian atas stack. Kedua, panjang data yang akan transferred harus ditunjukkan. Ketiga, karena dengan semua instruksi dengan operan, mode pengalamatan untuk operan masing-masing harus ditentukan. Poin terakhir dibahas dalam Bab 11.

Pemilihan instruksi transfer data untuk dimasukkan dalam instruksi menetapkan exemplifies jenis pertukaran desainer harus membuat. Sebagai contoh, umum lokasi (memori atau register) dari operand dapat ditunjukkan baik dalam spesifikasi dari opcode atau operan. Tabel 10.5 menunjukkan contoh IBM yang paling umum EAS/390 transfer data instruksi. Perhatikan bahwa ada varian untuk menunjukkan jumlah data yang ditransfer (8, 16, 32, atau 64 bit). Juga, ada yang berbeda instruksi untuk register untuk register, register ke memori, memori untuk register, dan memory untuk transfer memori. Sebaliknya, VAX memiliki bergerak (MOV) instruksi dengan varian untuk jumlah yang berbeda dari data yang akan dipindahkan, tetapi menentukan apakah operan register atau memori sebagai bagian dari operand. Pendekatan VAX adalah beberapa apa yang lebih mudah bagi programmer, yang memiliki lebih sedikit mnemonik untuk pengalamatan. Namun, juga agak kurang kompak dibandingkan dengan pendewordn EAS/390 IBM karena lokasi menurut tion (register

dibandingkan memori) dari operan masing-masing harus ditentukan secara terpisah dalam instruksi. Kami akan kembali ke perbedaan ini ketika kita membahas format instruksi, di bab berikutnya.

Dalam hal tindakan prosesor, operasi transfer data mungkin adalah yang paling sederhana jenis. Jika kedua sumber dan tujuan adalah register, maka prosesor hanya menyebabkan data yang akan ditransfer dari satu register ke yang lain, ini adalah operasi internal untuk

Table 10.5 Examples of IBM EAS/390 Data Transfer Operations

Operation Mnemonic	Name	Number of Bits Transferred	Description
L	Load	32	Transfer from memory to register
LH	Load Halfword	16	Transfer from memory to register
LR	Load	32	Transfer from register to register
LER	Load (Short)	32	Transfer from floating-point register to floating-point register
LE	Load (Short)	32	Transfer from memory to floating-point register
LDR	Load (Long)	64	Transfer from floating-point register to floating-point register
LD	Load (Long)	64	Transfer from memory to floating-point register
ST	Store	32	Transfer from register to memory
STH	Store Halfword	16	Transfer from register to memory
STC	Store Character	8	Transfer from register to memory
STE	Store (Short)	32	Transfer from floating-point register to memory
STD	Store (Long)	64	Transfer from floating-point register to memory

prosesor. Jika satu atau kedua operan adalah di memori, maka prosesor harus permembentuk beberapa atau semua tindakan berikut:

1. Hitung alamat memori, berdasarkan modus alamat (dibahas dalam Bab 11).
2. Jika alamat mengacu pada memori virtual, menerjemahkan dari virtual untuk memori nyata alamat.
3. Tentukan apakah item yang dibahas adalah dalam cache.
4. Jika NOT, mengeluarkan perintah ke modul memori.

Aritmatika

Kebanyakan mesin menyediakan operasi aritmatika dasar menambah, mengurangi, mengalikan, dan membagi. Ini selalu disediakan untuk ditandatangani integer (fixed-point) angka. Seringkali mereka juga disediakan untuk angka desimal floating-point dan packed. Operasi lain yang mungkin termasuk berbagai tunggal operan instruksi, sebagai contoh,

Absolute: Ambil nilai absolut dari operand.

Meniadakan: Meniadakan operan.

Selisih: Tambahkan 1 ke operand.

Pengurangan: Kurangi 1 dari operan.

Table 10.6 Basic Logical Operations

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P = Q
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

Pelaksanaan instruksi aritmatika dapat melibatkan transfer data operations untuk posisi operan untuk input ke ALU, dan untuk memberikan output dari ALU. Gambar 3.5 menggambarkan gerakan terlibat baik dalam transfer data dan arithmetic operasi. Selain itu, tentu saja, bagian ALU prosesor melakukan operasi yang diinginkan.

Logika

Kebanyakan mesin juga menyediakan berbagai operasi untuk memanipulasi bit individual dari sebuah word atau unit dialamatkan lain, sering disebut sebagai Mereka adalah "sedikit memutar-mutar." didasarkan pada operasi Boolean (lihat Bab 20).

Beberapa operasi logis dasar yang dapat dilakukan pada Boolean atau binary data yang ditunjukkan pada Tabel 10.6. Operasi NOT membalikkan sedikit. AND, OR, dan Exclusive-OR (XOR) adalah fungsi logis yang paling umum dengan dua operan. SAMA adalah tes biner berguna.

Operasi-operasi logika dapat diterapkan bitwise untuk n-bit data logis unit. Dengan demikian, jika dua register berisi data

(R1) =10100101

(R2) =00001111

kemudian

(R1) DAN (R2) =00000101

dimana notasi (X) berarti isi dari lokasi X. Dengan demikian, operasi AND

dapat digunakan sebagai *masker* yang memilih bit tertentu dalam word dan nol keluar sisanya bit. Sebagai contoh lain, jika dua register mengandung

(R1) =10100101

(R2) =11111111

kemudian

(R1) XOR (R2) =01011010

Dengan satu word diatur untuk semua 1s, operasi XOR membalikkan semua bit yang lain word (yang melengkapi). Selain operasi logis bitwise, kebanyakan mesin menyediakan berbagai pergeseraning dan berputar functions.The operasi paling dasar diilustrasikan pada Gambar 10.6.With sebuah **logika shift,** bit dari sebuah word yang bergeser kiri atau kanan. Pada salah satu ujungnya, sedikit bergeser keluar hilang. Pada ujung yang lain, 0 yang digeser masuk pergeseran logis berguna terutama untuk mengisolasi

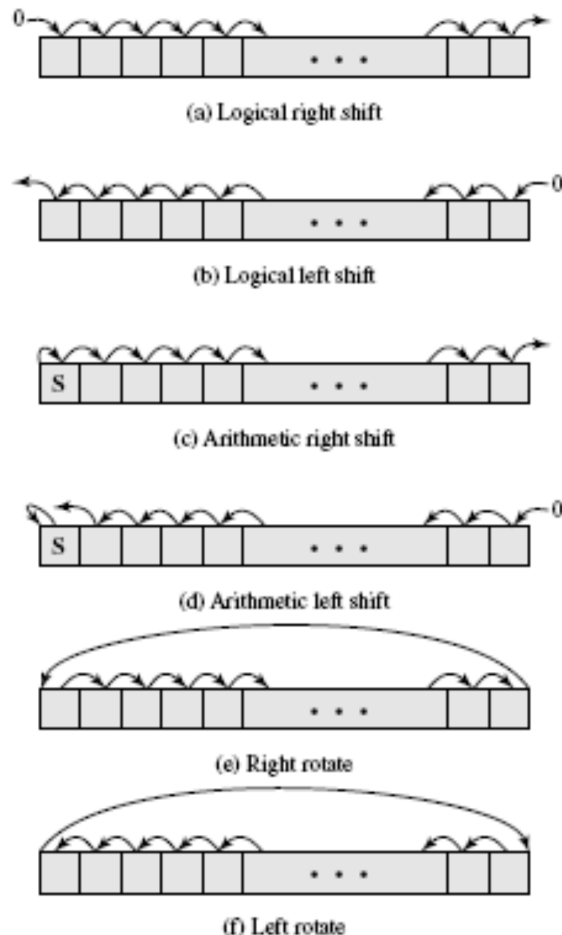


Figure 10.6 Shift and Rotate Operations

bidang dalam 0s word. Yang bergeser menjadi sebuah word menggantikan informasi yang tidak diinginkan yang bergeser dari ujung lainnya. Sebagai contoh, misalkan kita ingin mengirimkan karakter data ke I / O dewakil 1 karakter pada suatu waktu. Jika setiap word memori adalah 16 bit panjang dan berisi dua karakter, kita harus *membongkar* karakter sebelum mereka dapat dikirim. Untuk mengirim dua karakter dalam word,

1. Memuat word ke register.

2. Bergeser ke delapan kali benar. Hal ini akan merubah karakter yang tersisa di sebelah kanan setengah dari register.

3. Modul I / O membaca urutan lebih rendah-8 bit dari data bus.

Hasil langkah sebelumnya dalam mengirimkan karakter kiri. Untuk mengirim kanan tangan karakter,

1. Memuat word lagi ke register.
2. DAN dengan 0000000011111111. Ini masker keluar karakter di sebelah kiri.
3. Lakukan I / O.

Aritmatika pergeseran operasi memperlakukan data sebagai integer ditandatangani dan tidak menggeser bit tanda. Pada pergeseran aritmatika yang benar, sedikit tanda direplikasi ke bit pokerasnya terhadap haknya. Pada pergeseran aritmatika kiri, pergeseran logis kiri dilakukan pada semua bit tapi sedikit tanda, yang dipertahankan. Operasi ini dapat mempercepat aritmatika tertentu operasi. Dengan angka dalam notasi komplemen dua, pergeseran aritmatika kanan cormenanggapi pembagian dengan 2, dengan pemotongan untuk nomor ganjil. Kedua aritmatika yang tersisa bergeser dan pergeseran kiri yang logis sesuai dengan perkalian dengan 2 bila NOT ada overflow. Jika overflow terjadi, aritmatika dan logika operasi shift kiri menghasilkan dif-Hasil ferent, tetapi pergeseran kiri aritmatika mempertahankan tanda nomor. Karena potensi untuk overflow, banyak prosesor tidak termasuk instruksi ini, termasuk PowerPC dan Itanium. Lainnya, seperti IBM EAS/390, lakukan menawarkan instruksi. Anehnya, arsitektur x86 mencakup shift kiri aritmetika tetapi mendefinisikan hal itu terjadi identik dengan pergeseran kiri logis.
Memutar, atau siklik pergeseran, operasi melestarikan semua bit yang dioperasi. Salah satu penggunaan rotate adalah untuk membawa setiap bit berurutan ke dalam bit paling kiri, di mana ia dapat diidentifikasi dengan menguji tanda data (diperlakukan sebagai angka).

Seperti operasi aritmatika, operasi logis melibatkan aktivitas ALU dan mungkin melibatkan operasi data transfer. Tabel 10.7 memberikan contoh semua shift dan memutar operasi dibahas pada bagian ini.

Konversi

Instruksi konversi adalah mereka yang mengubah format atau beroperasi pada format data. Contohnya adalah mengkonversi dari desimal ke biner. Sebuah contoh yang lebih instruksi editing kompleks adalah Translate EAS/390 (TR) instruksi. Ini instruksition dapat digunakan untuk mengkonversi dari satu 8-bit kode ke yang lain, dan dibutuhkan tiga Operand:

TR R1 (L), R2

R2 operan berisi alamat awal tabel 8-bit kode. L byte mulai dari alamat yang ditentukan dalam R1 dijabarkan, setiap byte yang diganti oleh isi dari entri tabel diindeks oleh byte itu. Misalnya, untuk menerjemahkan dari EBCDIC ke IRA, pertama kita membuat tabel 256-byte dalam lokasi penyimpanan, wordkanlah, 1000 - 10FF heksadesimal. Tabel berisi karakter dari kode IRA di sequence dari representasi biner dari kode EBCDIC, yang, kode IRA adalah ditempatkan di meja di lokasi relatif sama dengan nilai biner dari EBCDIC

Table 10.7 Examples of Shift and Rotate Operations

Input	Operation	Result
10100110	Logical right shift (3 bits)	00010100
10100110	Logical left shift (3 bits)	00110000
10100110	Arithmetic right shift (3 bits)	11110100
10100110	Arithmetic left shift (3 bits)	10110000
10100110	Right rotate (3 bits)	11010100
10100110	Left rotate (3 bits)	00110101

kode dari karakter yang sama. Dengan demikian, lokasi 10F0 melalui 10F9 akan berisi nilai-nilai 30 sampai 39, karena F0 adalah kode EBCDIC untuk 0 digit, dan 30 adalah IRA kode untuk digit 0, dan seterusnya sampai 9 digit. Sekarang misalkan kita memiliki EBCDIC ini untuk digit 1984 mulai dari lokasi 2100 dan kita ingin menerjemahkan ke IRA. Menganggap sebagai berikut:

- Lokasi 2100-2103 mengandung F1 F9 F4 F8.
- R1 berisi 2100.
- R2 berisi 1000.

Kemudian, jika kita menjalankan

TR R1 (4), R2

lokasi 2100-2103 akan berisi 31 39 38 34.

Input / Output

Input / output petunjuk dibahas secara rinci dalam Bab 7. Seperti kita lihat, ada berbagai pendewordn yang diambil, termasuk terisolasi diprogram I / O, memori dipetakan I diprogram / O, DMA, dan penggunaan prosesor I / O. Banyak implementasi, sultasi hanya memberikan beberapa instruksi I / O, dengan tindakan spesifik yang ditentukan oleh parameter, kode, atau word-word perintah.

Sistem Pengendalian

Sistem instruksi kontrol adalah mereka yang dapat dieksekusi hanya ketika prosesor dalam keadaan istimewa tertentu atau sedang mengeksekusi sebuah program di daerah istimewa khusus memori. Biasanya, instruksi ini dicadangkan untuk penggunaan operasi sistem.

Beberapa contoh operasi sistem kontrol adalah sebagai berikut. Sebuah sistem control instruksi dapat membaca atau mengubah kontrol register; kita membahas register kontrol dalam Bab 12. Contoh lain adalah instruksi untuk membaca atau memodifikasi kunci perlindungan penyimpanan, seperti digunakan dalam sistem memori EAS/390. Contoh lain adalah akses ke memproses blok kontrol dalam sistem multiprogramming.

Transfer Kontrol

Untuk semua jenis operasi dibahas sejauh ini, instruksi berikutnya yang akan dilakukan adalah salah satu yang segera mengikuti, dalam memori, instruksi saat ini. Namun, fraksi yang signifikan dari instruksi dalam program apapun miliki sebagai fungsi mereka changing urutan eksekusi instruksi. Untuk petunjuk ini, operasi per dibentuk oleh prosesor adalah untuk memperbarui program counter mengandung alamat beberapa instruksi dalam memori.

Ada beberapa alasan mengapa transfer-of-control operasi yang diperlukan. Di antara yang paling penting adalah sebagai berikut:

1. Dalam penggunaan praktis komputer, adalah penting untuk dapat mengeksekusi setiap inkonstruksi lebih dari sekali dan mungkin ribuan kali. Ini mungkin memerlukan ribuan atau mungkin jutaan instruksi untuk menerapkan aplikasi. Ini akan terpikirkan jika setiap instruksi harus ditulis secara terpisah. Jika sebuah tabel atau daftar item yang akan diproses, sebuah loop program yang dibutuhkan. Salah satu sequence instruksi dieksekusi berulang kali untuk memproses semua data.

2. Hampir semua program melibatkan beberapa pengambilan keputusan. Kami ingin computer untuk melakukan satu hal jika salah satu kondisi memegang, dan hal lain jika satu condition berlaku. Sebagai contoh, urutan instruksi menghitung akar kuadrat dari angka. Pada awal urutan, tanda nomor diuji. Jika jumlah negatif, perhitungan NOT dilakukan, tetapi kesalahan-kondisi tion dilaporkan.

3. Untuk membuat program komputer dengan benar besar atau bahkan ukuran sedang adalah mantan ceedingly sulit tugas. Ini membantu jika ada mekanisme untuk melanggar tugas menjadi potongan kecil yang dapat bekerja pada satu per satu.

Kita sekarang beralih ke pembahasan yang paling umum mentransfer terkendali operations ditemukan di set instruksi: cabang, melompat, dan panggilan prosedur.

CABANG INSTRUKSI Sebuah instruksi cabang, juga disebut instruksi lompatan, memilikisebagai salah satu operand-nya alamat dari instruksi berikutnya yang akan dieksekusi. Paling sering, instruksi adalah **bersyarat cabang** instruksi. Artinya, cabang dibuat (uptanggal program counter ke alamat yang sama ditentukan dalam operan) hanya jika kondisi tertentutin terpenuhi. Jika NOT, instruksi berikutnya dalam urutan dijalankan (kelipatan program counter seperti biasa). Sebuah instruksi cabang di cabang yang selalu diambil adalah **cabang tanpa syarat.**

Ada dua cara umum untuk menghasilkan kondisi yang akan diuji dalam consional instruksi cabang. Pertama, mesin yang paling memberikan con-1-bit atau beberapa bit dition kode yang ditetapkan sebagai hasil dari beberapa operasi. Kode ini dapat dianggap sebagai register pengguna-terlihat pendek. Sebagai contoh, operasi aritmatika (ADD, Mengurangi, dan sebagainya) dapat menetapkan kode kondisi 2-bit dengan salah satu dari berikut empat nilai: 0, positif, negatif, meluap. Pada mesin tersebut, mungkin ada empat berbeda instruksi cabang kondisional:

BRP X

BRN X

BRZ X

BRO X

Cabang ke lokasi X jika hasilnya positif.

Cabang ke lokasi X jika hasilnya adalah negatif.

Cabang ke lokasi X jika hasilnya adalah nol.

Cabang ke lokasi X jika overflow terjadi.

Dalam semua kasus ini, hasil yang dimaksud adalah hasil yang paling baru operasi yang mengatur kode kondisi. Pendekatan lain yang dapat digunakan dengan format instruksi tiga alamat untuk melakukan perbandingan dan menentukan cabang di instruksi yang sama. Sebagai contoh,

Bre R1, R2, X Cabang ke X jika isi dari R1 =isi R2.

Gambar 10.7 menunjukkan contoh operasi ini. Perhatikan bahwa cabang dapat either *depan* (Instruksi dengan alamat lebih tinggi) atau *ke belakang* (Alamat lebih rendah). Contoh ini menunjukkan bagaimana sebuah tanpa syarat dan cabang bersyarat dapat digunakan untuk membuat loop mengulangi instruksi. Petunjuk di lokasi 202 melalui 210 akan dieksekusi berulang-ulang sampai hasil mengurangkan Y dari X adalah 0.

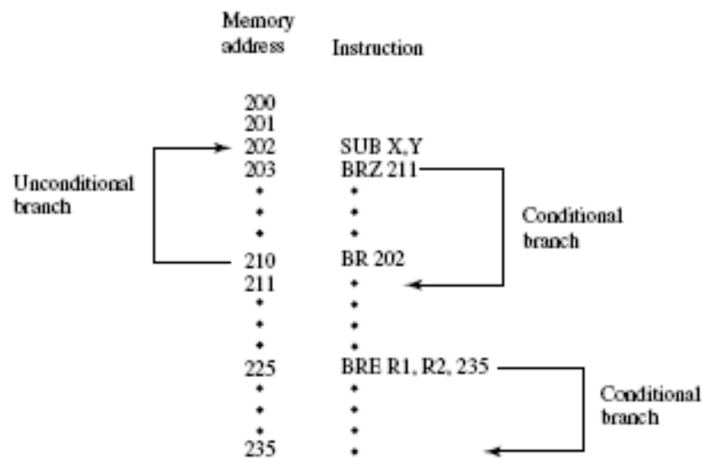


Figure 10.7 Branch Instructions

SKIP INSTRUKSI Bentuk lain dari transfer-of-control instruksi adalah dilewati dikonstruksi. Instruksi loncat mencakup alamat tersirat. Biasanya, loncat menyiratkan yang satu instruksi dilewati, dengan demikian, alamat tersirat sama dengan alamat instruksi berikutnya ditambah satu panjang instruksi.

Karena instruksi loncat NOT memerlukan field alamat tujuan, itu adalah bebas untuk melakukan hal-hal lain. Sebuah contoh khas adalah kenaikan-dan-skip-jika-nol (isz) instruksi. Pertimbangkan fragmen program berikut:

```
301
•
•
•
309    ISZ    R1
310    BR     301
311
```

Dalam fragmen ini, dua transfer-of-control instruksi yang digunakan untuk mengimplementasikan sebuah iteratif loop. R1 diatur dengan negatif dari jumlah iterasi menjadi perterbentuk. Pada akhir loop, R1 bertambah. Jika NOT 0, program cabang kembali ke awal loop. Jika NOT, cabang akan dilewati, dan program berlanjut dengan instruksi berikutnya setelah akhir loop.

PROSEDUR PANGGILAN INSTRUKSI Mungkin inovasi yang paling penting dalam pengembangan bahasa pemrograman adalah *prosedur.* Prosedur adalah diri yang berisi program komputer yang dimasukkan ke dalam program yang lebih besar. Pada setiap titik dalam program prosedur dapat dipanggil, atau *disebut.* Prosesor ini distructed untuk pergi dan melaksanakan seluruh prosedur dan kemudian kembali ke titik dari yang panggilan berlangsung. Dua alasan utama untuk penggunaan prosedur adalah ekonomi dan MODUlarity.

Sebuah prosedur memungkinkan potongan kode yang sama untuk digunakan berkali-kali. Ini adalah important bagi perekonomian dalam upaya pemrograman dan untuk membuat paling efisien penggunaan

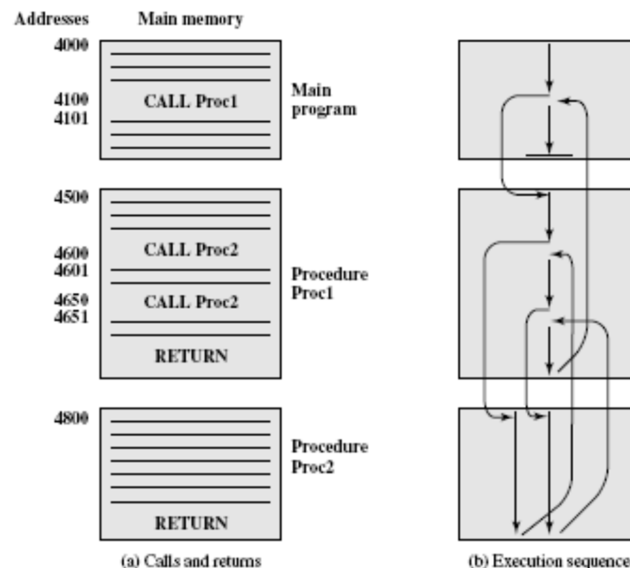


Figure 10.8 Nested Procedures

ruang penyimpanan dalam sistem (program harus disimpan). Prosedur juga memungkinkan tugas pemrograman besar untuk dibagi lagi menjadi unit yang lebih kecil. Ini penggunaan *modularitas* sangat memudahkan tugas pemrograman. Mekanisme prosedur melibatkan dua instruksi dasar: sebuah instruksi panggilan bahwa cabang-cabang dari lokasi prosedur, dan instruksi pengembalian yang mengembalikan dari prosedur ke tempat dari mana itu disebut. Kedua adalah bentuk-bentuk percabangan instruksi.

Gambar 10.8a mengilustrasikan penggunaan prosedur untuk membuat program. Dalam mantan cukup, ada program utama mulai dari lokasi 4000. Program ini mencakup panggilan prosedur PROC1, mulai dari lokasi 4500. Ketika instruksi ini panggilan encountered, prosesor menunda pelaksanaan program utama dan mulai pelaksanaan PROC1 dengan mengambil instruksi berikutnya dari lokasi 4500. Dalam PROC1, ada dua panggilan ke PROC2 di lokasi 4800. Dalam setiap kasus, pelaksanaan PROC1 adalah suspended dan PROC2 adalah pernyataan RETURN executed.The menyebabkan prosesor untuk pergi kembali ke program menelepon dan melanjutkan eksekusi pada instruksi setelah corresponding instruksi CALL. Perilaku ini digambarkan pada Gambar 10.8b.

Tiga hal perlu diketahui:

1. Sebuah prosedur dapat dipanggil dari lebih dari satu lokasi.
2. Panggilan prosedur dapat muncul dalam prosedur. Hal ini memungkinkan *bersarang* dari prosedurprosedur di ke kedalaman sewenang-wenang.
3. Setiap panggilan prosedur yang cocok dengan kembali dalam program yang disebut.

Karena kami ingin dapat memanggil prosedur dari berbagai poin, prosesor entah bagaimana harus menyimpan alamat pengirim sehingga kembali dapat mengambil menempatkan dengan tepat. Ada tiga tempat umum untuk menyimpan alamat pengirim:

•Register

•Mulai dari yang disebut prosedur

•Atas stack

Pertimbangkan mesin bahasa instruksi X CALL, yang merupakan singwordn dari *memanggil prosedur di lokasi X.* Jika pendewordn register digunakan, CALL X menyebabkan tindakan berikut:

RN -PC + ¢

PC -X

mana RN adalah register yang selalu digunakan untuk tujuan ini, PC adalah program counter, dan ¢adalah panjang instruksi. Prosedur yang disebut sekarang dapat menghemat con-the tenda dari RN yang akan digunakan untuk mengembalikan kemudian. Kemungkinan kedua adalah untuk menyimpan alamat pengirim pada awal prosedur. Dalam hal ini, HUBUNGI penyebab X

X-PC + ¢

PC -X+1

Hal ini cukup berguna. Alamat pengirim telah disimpan dengan aman pergi. Kedua pendewordn sebelumnya bekerja dan telah digunakan. Para limita-satunya tion pendewordn ini adalah bahwa mereka mempersulit penggunaan *reentrant* prosedur. Sebuah prosedur reentrant adalah satu di mana adalah mungkin untuk beberapa panggilan terbuka untuk itu pada waktu yang sama. Sebuah prosedur rekursif (yang menyebut dirinya) adalah contoh dari penggunaan fitur ini (lihat Lampiran H). Jika parameter dilewatkan melalui register atau memori untuk prosedur reentrant, beberapa kode harus bertanggung jawab untuk menyimpan parameter sehingga register atau ruang memori yang tersedia untuk panggilan prosedur lainnya. Pendewordn yang lebih umum dan kuat adalah dengan menggunakan stack (lihat Lampiran 10A untuk diskusi dari stack). Ketika prosesor mengeksekusi panggilan, ia menempatkan kembali alamat pada stack. Ketika dijalankan kembali, menggunakan alamat pada stack. Gambar 10.9 mengilustrasikan penggunaan stack. Selain memberikan alamat pengirim, juga sering diperlukan untuk lulus parameters dengan panggilan prosedur. Ini dapat berlalu dalam register. Kemungkinan lain adalah untuk menyimpan parameter dalam memori hanya setelah instruksi CALL. Dalam hal ini,

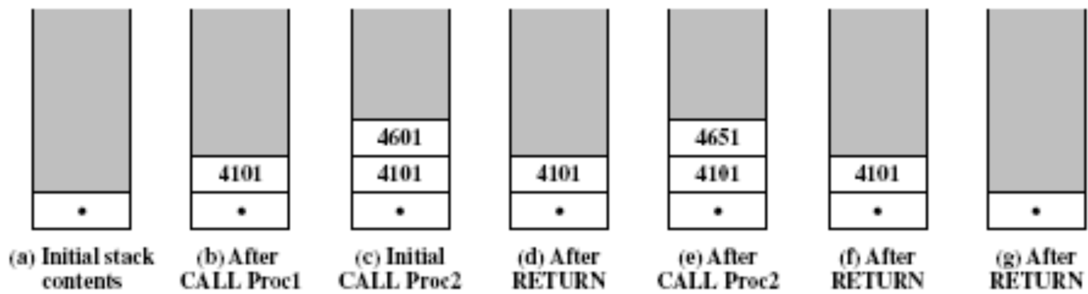


Figure 10.9 Use of Stack to Implement Nested Subroutines of Figure 10.8

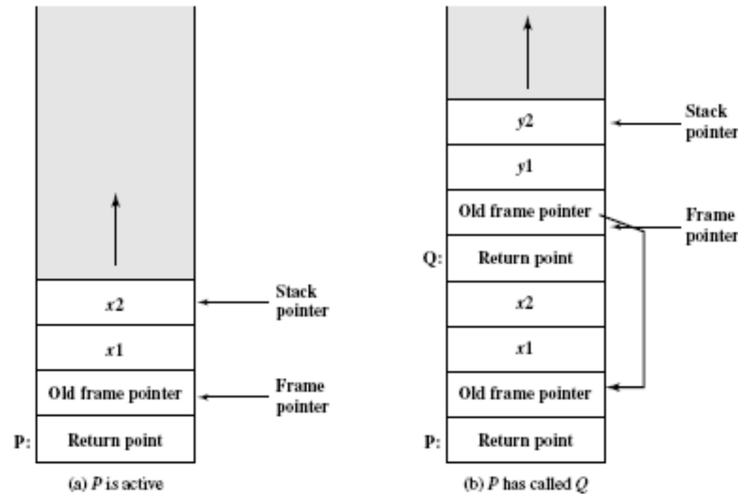


Figure 10.10 Stack Frame Growth Using Sample Procedures P and Q

kembali harus ke lokasi setelah parameter. Sekali lagi, kedua approaches memiliki kelemahan. Jika register yang digunakan, program yang disebut dan panggilan yang Program harus ditulis untuk memastikan bahwa register digunakan dengan benar. Para menyimpan parameter dalam memori membuat sulit untuk bertukar sejumlah variabel parameter. Kedua pendekatan mencegah penggunaan prosedur reentrant. Pendekatan yang lebih fleksibel untuk passing parameter stack. Ketika processor mengeksekusi panggilan, NOT hanya stack alamat pengirim, stack parameter untuk dilewatkan ke prosedur yang disebut. Prosedur yang disebut dapat mengakses parameter dari stack. Setelah kembali, parameter kembali juga dapat ditempatkan pada stack. Itu set parameter keseluruhan, termasuk alamat pengirim, yang disimpan untuk suatu prosedurpanggilan ini disebut sebagai *stack bingkai.*

Sebagai contoh adalah pada Gambar 10.10. Contoh ini mengacu pada prosedur dalam P mana variabel lokal x1 dan x2 dinyatakan, dan prosedur Q, yang dapat memanggil P dan di mana variabel lokal y1 dan y2 diumumkan. Dalam gambar ini, kembali point untuk setiap prosedur adalah item pertama disimpan di stack frame yang sesuai.

Berikutnya disimpan pointer ke awal dari frame sebelumnya. Hal ini diperlukan jika jumlah atau panjang parameter yang akan ditumpuk adalah variabel.

JENIS OPERASI X86 DAN ARM

Operasi x86 Jenis

X86 menyediakan array kompleks jenis operasi, termasuk sejumlah khusus instruksi. Tujuannya adalah untuk memberikan alat bagi penulis compiler untuk menghasilkan dioptimalkan mesin terjemahan bahasa tingkat tinggi program bahasa. Tabel 10,8

Table 10.8 x86 Operation Types (with Examples of Typical Operations)

Instruction	Description
Data Movement	
MOV	Move operand, between registers or between register and memory.
PUSH	Push operand onto stack.
PUSHA	Push all registers on stack.
MOVSX	Move byte, word, dword, sign extended. Moves a byte to a word or a word to a doubleword with twos-complement sign extension.
LEA	Load effective address. Loads the offset of the source operand, rather than its value to the destination operand.
XLAT	Table lookup translation. Replaces a byte in AL with a byte from a user-coded translation table. When XLAT is executed, AL should have an unsigned index to the table. XLAT changes the contents of AL from the table index to the table entry.
IN, OUT	Input, output operand from I/O space.
Arithmetic	
ADD	Add operands.
SUB	Subtract operands.
MUL	Unsigned integer multiplication, with byte, word, or double word operands, and word, doubleword, or quadword result.
IDIV	Signed divide.
Logical	
AND	AND operands.
BTS	Bit test and set. Operates on a bit field operand. The instruction copies the current value of a bit to flag CF and sets the original bit to 1.
BSF	Bit scan forward. Scans a word or doubleword for a 1-bit and stores the number of the first 1-bit into a register.
SHL/SHR	Shift logical left or right.
SAL/SAR	Shift arithmetic left or right.
ROL/ROR	Rotate left or right.
SETcc	Sets a byte to zero or one depending on any of the 16 conditions defined by status flags.
Control Transfer	
JMP	Unconditional jump.
CALL	Transfer control to another location. Before transfer, the address of the instruction following the CALL is placed on the stack.
JE/JZ	Jump if equal/zero.
LOOPE/LOOPZ	Loops if equal/zero. This is a conditional jump using a value stored in register ECX. The instruction first decrements ECX before testing ECX for the branch condition.
INT/INTO	Interrupt/Interrupt if overflow. Transfer control to an interrupt service routine.

Instruction	Description
	String Operations
MOVS	Move byte, word, dword string. The instruction operates on one element of a string, indexed by registers ESI and EDI. After each string operation, the registers are automatically incremented or decremented to point to the next element of the string.
LODS	Load byte, word, dword of string.
	High-Level Language Support
ENTER	Creates a stack frame that can be used to implement the rules of a block-structured high-level language.
LEAVE	Reverses the action of the previous ENTER.
BOUND	Check array bounds. Verifies that the value in operand 1 is within lower and upper limits. The limits are in two adjacent memory locations referenced by operand 2. An interrupt occurs if the value is out of bounds. This instruction is used to check an array index.
	Flag Control
STC	Set Carry flag.
LAHF	Load AH register from flags. Copies SF, ZF, AF, PF, and CF bits into A register.
	Segment Register
LDS	Load pointer into DS and another register.
	System Control
HLT	Halt.
LOCK	Asserts a hold on shared memory so that the Pentium has exclusive use of it during the instruction that immediately follows the LOCK.
ESC	Processor extension escape. An escape code that indicates the succeeding instructions are to be executed by a numeric coprocessor that supports high-precision integer and floating-point calculations.
WAIT	Wait until BUSY# negated. Suspends Pentium program execution until the processor detects that the BUSY pin is inactive, indicating that the numeric coprocessor has finished execution.
	Protection
SGDT	Store global descriptor table.
LSL	Load segment limit. Loads a user-specified register with a segment limit.
VERR/VERW	Verify segment for reading/writing.
	Cache Management
INVD	Flushes the internal cache memory.
WBINVD	Flushes the internal cache memory after writing dirty lines to memory.
INVLPG	Invalidates a translation lookaside buffer (TLB) entry.

daftar jenis dan memberikan contoh masing-masing. Sebagian besar adalah konvensional instruksi ditemukan di set mesin yang paling instruksi, tetapi beberapa jenis instruksi disesuaikan dengan arsitektur x86 dan kepentingan tertentu. Lampiran A dari [CART06] daftar instruksi x86, bersama dengan operan untuk setiap dan effect dari instruksi di kode kondisi. Lampiran B dari perakitan NASM bahasa pengguna memberikan gambaran lebih rinci dari setiap instruksi x86.

CALL / RETURN INSTRUKSI X86 ini menyediakan empat instruksi untuk mendukung procedure call / return: CALL, ENTER, LEAVE, RETURN. Ini akan menjadi pelajaran untuk melihat support yang diberikan oleh petunjuk ini. Ingat dari Gambar 10.10 bahwa sarana umum menerapkan prosedur call / kembali mekanisme adalah melalui penggunaan dari frame stack. Ketika sebuah prosedur baru ini disebut, berikut ini harus dilakukan pada saat masuk ke prosedur baru:

- Dorong titik kembali di stack.
- Dorong pointer frame di stack.
- Salin stack pointer sebagai nilai baru dari frame pointer.
- Sesuaikan stack pointer untuk mengalokasikan bingkai.

Instruksi PANGGILAN mendorong nilai instruksi pointer saat ini ke stack dan menyebabkan melompat ke titik masuk prosedur dengan menempatkan alamat dari entry point dalam pointer instruksi. Dalam mesin 8088 dan 8086, khas Prosedur dimulai dengan urutan

```
PUSH EBP
MOV EBP, ESP
SUB ESP, space_for_locals
```

mana EBP adalah pointer frame dan ESP adalah stack pointer. Dalam 80286 dan kemudian mesin, instruksi ENTER melakukan semua operasi tersebut dalam instruksi tunggal. Instruksi ENTER ditambahkan ke set instruksi untuk memberikan support langsungport untuk compiler. Instruksi ini juga mencakup fitur untuk mendukung apa disebut prosedur bersarang dalam bahasa seperti Pascal, COBOL, dan Ada (tidak ditemukan di C atau FORTRAN). Ternyata ada cara yang lebih baik pengalamatan bersarang procedure panggilan untuk bahasa tersebut. Selanjutnya, meskipun instruksi ENTER menghemat beberapa byte dari memori dibandingkan dengan PUSH ini, MOV, urutan SUB (4 byte dibandingkan 6 byte), sebenarnya waktu lebih lama untuk mengeksekusi (10 siklus clock dibandingkan 6 jam siklus). Jadi, meskipun mungkin tampak ide yang baik untuk set instruksi desainer untuk menambahkan fitur ini, akan merumitkan pelaksanaan prosesor sambil memberikan manfaat sedikit atau NOT ada.

Kita akan melihat bahwa, sebaliknya, pendewordn RISC untuk desain prosesor akan menghindari instruksi kompleks seperti ENTER dan mungkin memperkenalkan implementasi yang lebih efisien dengan urutan instruksi sederhana.

MEMORY MANAGEMENT Satu set instruksi khusus berkaitan dengan memori segmentasi. Ini adalah instruksi privileged yang hanya dapat dijalankan dari operating sistem. Mereka memungkinkan tabel segmen lokal dan global (disebut deskriptor tabel) untuk dimuat dan membaca, dan untuk tingkat perlakuan segmen yang akan diperiksa dan diubah.

Table 10.9 x86 Status Flags

Status Bit	Name	Description
C	Carry	Indicates carrying or borrowing out of the left-most bit position following an arithmetic operation. Also modified by some of the shift and rotate operations.
P	Parity	Parity of the least-significant byte of the result of an arithmetic or logic operation. 1 indicates even parity; 0 indicates odd parity.
A	Auxiliary Carry	Represents carrying or borrowing between half-bytes of an 8-bit arithmetic or logic operation. Used in binary-coded decimal arithmetic.
Z	Zero	Indicates that the result of an arithmetic or logic operation is 0.
S	Sign	Indicates the sign of the result of an arithmetic or logic operation.
O	Overflow	Indicates an arithmetic overflow after an addition or subtraction for twos complement arithmetic.

STATUS FLAGS DAN KODE KONDISI Status flag adalah bit dalam register khusus yang mungkin diatur oleh operasi tertentu dan digunakan dalam jangka instructions.The cabang bersyarat *kondisi kode* mengacu pada pengaturan dari satu atau lebih flag status. Dalam x86 dan banyak arsitektur lain, flag status ditetapkan oleh aritmatika dan membandingkan operasi. Itu membandingkan operasi dalam bahasa yang paling mengurangi dua operan, seperti halnya kurangi operasi.

Perbedaannya adalah bahwa operasi membandingkan hanya menetapkan flag status, sedangkan suboperasi saluran juga menyimpan hasil dari pengurangan dalam operan tujuan. Beberapa arsitektur juga menetapkan flag status untuk instruksi transfer data. Tabel 10,9 daftar flag status yang digunakan pada x86. Setiap flag, atau kombinasi dari flag, dapat diuji untuk lompatan bersyarat. Tabel 10.10 menunjukkan kode kondisi (combinations nilai flag status) yang opkode lompatan bersyarat sudah ditetapkan.

Beberapa pengamatan yang menarik dapat dibuat tentang daftar ini. Pertama, kita mungkin ingin untuk menguji dua operan untuk menentukan apakah satu nomor lebih besar dari yang lain. Tapi ini akan tergantung pada apakah nomor tersebut terdaftar atau NOT. Sebagai contoh, 8-bit 11111111 jumlah lebih besar dari 00000000 jika dua angka diinterpretasikan sebagai unditandatangani bilangan bulat (255 70) tetapi kurang jika mereka dianggap sebagai 8-bit pelengkap berpasangan angka (-1 6 0). Bahasa perakitan itu banyak

memperkenalkan dua set istilah untuk membedakan dua kasus: Jika kita membandingkan dua angka sebagai bilangan bulat ditandatangani, kita menggunakan istilah *kurang dari* dan *lebih besar dari;* jika kita membandingkan mereka sebagai unsigned integer, kami menggunakan istilah *di bawah* dan *di atas.*

Pengamatan kedua menyangkut kompleksitas membandingkan bilangan bulat ditandatangani. Hasil ditandatangani lebih besar dari atau sama dengan nol jika (1) sedikit tanda adalah nol dan ada tidak ada overflow (S =0 DAN O =0), atau (2) sedikit tanda adalah satu dan ada overflow. Sebuah studi dari Gambar 9.4 harus meyakinkan Anda bahwa kondisi diuji untuk berbagai operasi ditandatangani sesuai.

X86 SIMD INSTRUKSI Pada tahun 1996, Intel memperkenalkan teknologi MMX yang ke Pentium lini produk. MMX set instruksi yang sangat optimal untuk tugas-tugas multimedia. Ada 57 instruksi baru yang memperlakukan data dalam SIMD (single-instruksi, multipledata) mode, yang memungkinkan untuk melakukan operasi yang sama, seperti penambahan atau perkalian, pada elemen data sekaligus. Setiap instruksi biasanya membutuhkan jam satu siklus untuk mengeksekusi. Untuk aplikasi yang tepat, ini paralel operasi cepat

Table 10.10 x86 Condition Codes for Conditional Jump and SETcc Instructions

Symbol	Condition Tested	Comment
A, NBE	C=0 AND Z=0	Above; Not below or equal (greater than, unsigned)
AE, NB, NC	C=0	Above or equal; Not below (greater than or equal, unsigned); Not carry
B, NAE, C	C=1	Below; Not above or equal (less than, unsigned); Carry set
BE, NA	C=1 OR Z=1	Below or equal; Not above (less than or equal, unsigned)
E, Z	Z=1	Equal; Zero (signed or unsigned)
G, NLE	[(S=1 AND O=1) OR (S=0 and O=0)] AND [Z=0]	Greater than; Not less than or equal (signed)
GE, NL	(S=1 AND O=1) OR (S=0 AND O=0)	Greater than or equal; Not less than (signed)
L, NGE	(S=1 AND O=0) OR (S=0 AND O=1)	Less than; Not greater than or equal (signed)
LE, NG	(S=1 AND O=0) OR (S=0 AND O=1) OR (Z=1)	Less than or equal; Not greater than (signed)
NE, NZ	Z=0	Not equal; Not zero (signed or unsigned)
NO	O=0	No overflow
NS	S=0	Not sign (not negative)
NP, PO	P=0	Not parity; Parity odd
O	O=1	Overflow
P	P=1	Parity; Parity even
S	S=1	Sign (negative)

dapat menghasilkan speedup dua sampai delapan kali lebih algoritma yang sebanding yang tidak menggunakan instruksi MMX [ATKI96]. Dengan diperkenalkannya 64-bit arsitektur x86, Intel telah memperluas ekstensi ini untuk memasukkan ganda quadword (128 bit) Operand dan operasi floating-point. Dalam ayat ini, kita menggambarkan fitur MMX.

Fokus dari MMX adalah pemrograman multimedia. Video dan audio data typturun tajam terdiri dari array besar tipe data kecil, seperti 8 atau 16 bit, sedangkan coninstruksi konvensional dirancang untuk beroperasi pada 32 - atau 64-bit data. Berikut adalah beberapa contoh: Dalam grafis dan video, adegan tunggal terdiri dari array piksel, 2 dan ada 8 bit untuk setiap pixel atau 8 bit untuk setiap pixel komponen warna (merah, hijau, biru). Sampel audio Khas dikuantisasi menggunakan 16 bit. Untuk beberapa grafis 3D algoritma rithms, 32 bit biasa digunakan untuk tipe data dasar. Untuk menyediakan untuk operasi paralel pada ini panjang data, tiga baru tipe data didefinisikan dalam MMX. Setiap jenis data 64 bit panjang dan terdiri dari beberapa bidang data yang lebih kecil, masing-masing memegang sebuah fixed-point integer. Jenis adalah sebagai berikut:

- **Packed byte:** Delapan byte packed menjadi satu 64-bit kuantitas
- **Packed word:** Empat 16-bit packed ke dalam 64 bit
- **Packed doubleword:** Dua 32-bit doublewords packed ke dalam 64 bit

Table 10.11 MMX Instruction Set

Category	Instruction	Description
Arithmetic	PADD [B, W, D]	Parallel add of packed eight bytes, four 16-bit words, or two 32-bit doublewords, with wraparound.
	PADDS [B, W]	Add with saturation.
	PADDUS [B, W]	Add unsigned with saturation.
	PSUB [B, W, D]	Subtract with wraparound.
	PSUBS [B, W]	Subtract with saturation.
	PSUBUS [B, W]	Subtract unsigned with saturation.
	PMULHW	Parallel multiply of four signed 16-bit words, with high-order 16 bits of 32-bit result chosen.
	PMULLW	Parallel multiply of four signed 16-bit words, with low-order 16 bits of 32-bit result chosen.
	PMADDWD	Parallel multiply of four signed 16-bit words; add together adjacent pairs of 32-bit results.
Comparison	PCMPEQ [B, W, D]	Parallel compare for equality; result is mask of 1s if true or 0s if false.
	PCMPGT [B, W, D]	Parallel compare for greater than; result is mask of 1s if true or 0s if false.
Conversion	PACKUSWB	Pack words into bytes with unsigned saturation.
	PACKSS [WB, DW]	Pack words into bytes, or doublewords into words, with signed saturation.
	PUNPCKH [BW, WD, DQ]	Parallel unpack (interleaved merge) high-order bytes, words, or doublewords from MMX register.
	PUNPCKL [BW, WD, DQ]	Parallel unpack (interleaved merge) low-order bytes, words, or doublewords from MMX register.
Logical	PAND	64-bit bitwise logical AND
	PNDN	64-bit bitwise logical AND NOT
	POR	64-bit bitwise logical OR
	PXOR	64-bit bitwise logical XOR
Shift	PSLL [W, D, Q]	Parallel logical left shift of packed words, doublewords, or quadword by amount specified in MMX register or immediate value.
	PSRL [W, D, Q]	Parallel logical right shift of packed words, doublewords, or quadword.
	PSRA [W, D]	Parallel arithmetic right shift of packed words, doublewords, or quadword.
Data Transfer	MOV [D, Q]	Move doubleword or quadword to/from MMX register.
State Mgt	EMMS	Empty MMX state (empty FP registers tag bits).

Catatan: Jika instruksi mendukung beberapa jenis data [byte (B), word (W), doubleword (D), quadword (Q)], yang tipe data ditunjukkan dalam tanda kurung.

Tabel 10.11 berisi daftar set instruksi MMX. Sebagian besar instruksi melibatkan parallel operasi pada byte, word, atau doublewords. Sebagai contoh, instruksi PSLLW permembentuk

pergeseran logis kiri terpisah pada masing-masing dari empat word dalam word packed operan; instruksi PADDB membutuhkan operan byte packed sebagai masukan dan melakukan Yang terbaru paralel pada setiap posisi byte secara independen untuk menghasilkan output byte packed.

Salah satu fitur yang NOT biasa dari set instruksi baru adalah pengenalan **kejenuhan hitung** untuk byte dan 16-bit operand word. Dengan aritmatika unsigned biasa, ketika sebuah overflows operasi (yaitu, melaksanakan dari bit yang paling signifikan), maka tambahan sedikit terpotong. Hal ini disebut sebagai sampul, karena efek dari truncation dapat, misalnya, untuk menghasilkan hasil samping yang lebih kecil dari dua masukan operan. Pertimbangkan penambahan dari dua word, dalam heksadesimal, dan F000h 3000h. Jumlahnya akan dinyatakan sebagai

F000h = 1111 0000 0000 0000

3000h = 0011 0000 0000 0000

10010 0000 0000 0000 =2000h

Jika dua angka diwakili intensitas gambar, maka hasil dari penambahan adalah untuk membuat kombinasi dua warna gelap berubah menjadi lebih ringan. Ini adalah typi-Cally NOT apa yang dimaksudkan. Dengan aritmatika saturasi, jika hasil penambahan overaliran atau pengurangan hasil dalam underflow, hasilnya diatur ke terbesar atau terkecil nilai representable. Untuk contoh sebelumnya, dengan aritmatika kejenuhan, kita memiliki

F000h = 1111 0000 0000 0000

3000h = 0011 0000 0000 0000

10010 0000 0000 0000

1111 1111 1111 1111 =FFFFh

Untuk memberikan nuansa untuk penggunaan instruksi MMX, kita melihat contoh, diambil dari [PELE97]. Sebuah aplikasi video yang umum adalah fade-out, fade-in efek, dalam yang satu adegan bertahap larut ke yang lain. Dua gambar digabungkan dengan rata-rata tertimbang:

Result_ pixel =A_ pixel *luntur +B_ pixel *(1 -memudar)

Perhitungan ini dilakukan pada setiap posisi pixel dalam A B. dan Jika seri dari frame video diproduksi sementara secara bertahap mengubah nilai memudar dari 1 ke 0 (Ditingkatkan tepat untuk integer 8-bit), hasilnya adalah memudar dari gambar A ke gambar B.

Gambar 10.11 menunjukkan urutan langkah yang diperlukan untuk sekumpulan piksel. Itu 8-bit komponen pixel dikonversi menjadi 16-bit elemen untuk mengakomodasi MMX 16-bit kalikan kemampuan. Jika gambar-gambar ini menggunakan 640 *480 resolusi, dan teknik melarutkan menggunakan ke-255 nilai yang mungkin dari nilai memudar, maka total jumlah

instruksi dieksekusi menggunakan MMX adalah 535 juta. Para perhitungan yang sama tion, dilakukan tanpa instruksi MMX, membutuhkan 1,4 miliar instruksi eksekusi [INTE98].

ARM Operasi Jenis

Arsitektur ARM menyediakan koleksi besar jenis operasi. Berikut ini adalah kategori utama:

•**Memuat dan menyimpan instruksi:** Dalam arsitektur ARM, hanya memuat dan menyimpan dalamstructions lokasi memori akses; aritmatika dan instruksi logis dilakukan hanya pada register dan nilai-nilai segera dikodekan dalam instruksi.

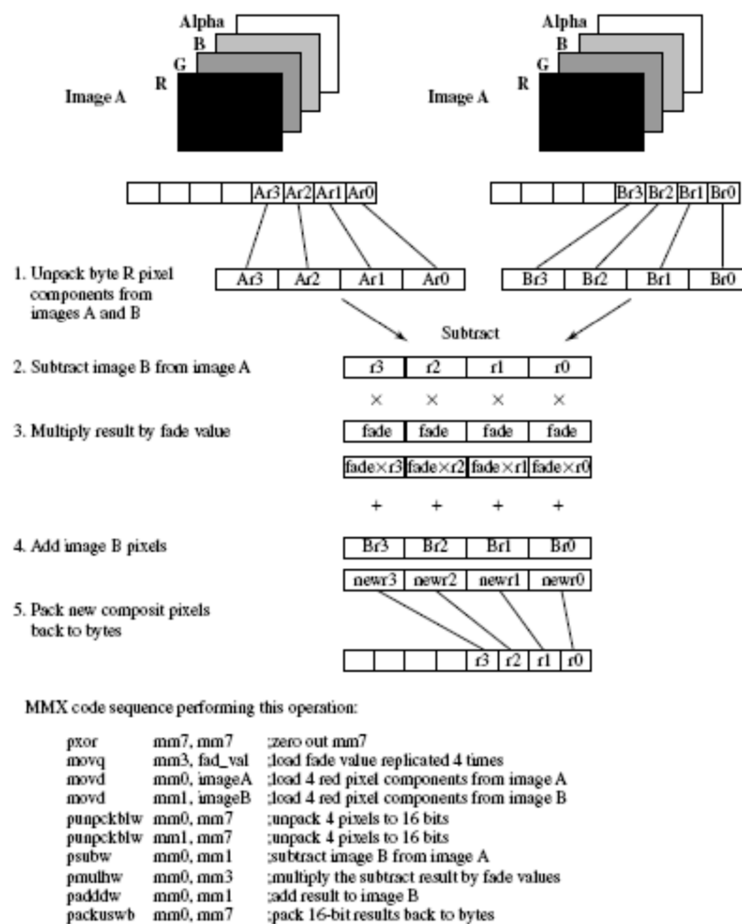


Figure 10.11 Image Compositing on Color Plane Representation

Keterbatasan ini adalah karakteristik dari desain RISC dan dieksplorasi lebih lanjut dalam Bab 13. Arsitektur ARM mendukung dua jenis yang luas dari instruksi bahwa beban atau menyimpan nilai dari sebuah register tunggal, atau sepasang register, dari atau ke memori: (1) memuat atau menyimpan word 32-bit atau byte 8-bit unsigned, dan (2) memuat atau menyimpan halfword 16-bit unsigned, dan memuat dan menandatangani memperpanjang halfword 16-bit atau byte 8-bit

•**Cabang instruksi:** ARM mendukung instruksi cabang yang memungkinkan kondisinasional

cabang maju atau mundur hingga 32 MB. Sebagai program counter adalah salah satu tujuan umum register (R15), sebuah cabang atau melompat juga bisa dihasilkan dengan menulis nilai R15. J subroutine panggilan dapat dilakukan oleh varian dari instruksi cabang standar. Serta memungkinkan cabang maju atau mundur hingga 32 MB, Cabang dengan Link (BL) instruksi mempertahankan alamat dari instruksi setelah cabang (return address) di LR (R14). Cabang ditentukan oleh medan kondisi 4-bit di instruksi.

•**Pengolahan data instruksi:** Kategori ini berisi petunjuk logis (AND, OR, XOR), menambah dan mengurangi instruksi, dan menguji dan membandingkan instruksi.

•**Kalikan instruksi:** Instruksi kali integer beroperasi pada word atau operan halfword dan dapat menghasilkan hasil yang normal atau panjang. Sebagai contoh, ada instruksi multiply yang membutuhkan dua 32-bit operand dan menghasilkan 64-bit hasil.

•**Paralel penambahan dan pengurangan instruksi:** Selain data normal pengolahan dan instruksi multiply, ada satu set Selain paralel dan pengurangan instruksi, di mana bagian-bagian dari dua operan dioperasikan pada secara paralel. Sebagai contoh, ADD16 menambahkan halfwords atas dua register untuk membentuk halfword atas hasil dan menambahkan halfwords bawah sama dua register untuk membentuk halfword bawah hasilnya. Ini instruksitions berguna dalam aplikasi pengolahan gambar, mirip dengan MMX x86 instruksi

•**Memperpanjang instruksi:** Ada beberapa instruksi untuk membongkar data dengan tanda atau nol byte untuk memperluas halfwords atau word-word, dan halfwords word-word.

•**Akses daftar instruksi status:** ARM menyediakan kemampuan untuk membaca dan juga menulis bagian-bagian dari register status.

KONDISI KODE Arsitektur ARM mendefinisikan flag empat kondisi yang disimpan dalam status program register: N, Z, C, dan V (Negatif, Zero, Carry dan overflow), dengan makna dasarnya sama dengan S, Z, C, dan V flag di arsitektur x86. Keempat flag merupakan sebuah kode kondisi di ARM.

Tabel 10.12 menunjukkan kombinasi kondisi yang eksekusi kondisional didefinisikan. Ada dua aspek yang NOT biasa untuk penggunaan kode kondisi di ARM:

1. Semua instruksi, bukan hanya instruksi cabang, termasuk bidang kode kondisi, yang berarti bahwa hampir semua instruksi dapat dieksekusi secara kondisional. Setiap kombinasi pengaturan flag kecuali 1110 atau 1111 dalam sebuah instruksi kontradition kolom kode menandakan bahwa instruksi akan dieksekusi hanya jika condition terpenuhi.

2. Semua instruksi pengolahan data (aritmatika, logis) termasuk bit S yang signi-SPPK apakah instruksi update flag kondisi.

Penggunaan dan pengaturan eksekusi kondisional bersyarat dari flag kondisi membantu dalam desain program pendek yang menggunakan memori kurang. Di sisi lain, semua instruksi meliputi 4 bit untuk kode kondisi, sehingga ada trade-off dalam yang lebih sedikit bit dalam instruksi 32-bit yang tersedia untuk opcode dan operan. Karena

Table 10.12 ARM Conditions for Conditional Instruction Execution

Code	Symbol	Condition Tested	Comment
0000	EQ	Z=1	Equal
0001	NE	Z=0	Not equal
0010	CS/HS	C=1	Carry set/unsigned higher or same
0011	CC/LO	C=0	Carry clear/unsigned lower
0100	MI	N=1	Minus/negative
0101	PL	N=0	Plus/positive or zero
0110	VS	V=1	Overflow
0111	VC	V=0	No overflow
1000	HI	C=1 AND Z=0	Unsigned higher
1001	LS	C=0 OR Z=1	Unsigned lower or same
1010	GE	N=V [(N=1 AND V=1) OR (N=0 AND V=0)	Signed greater than or equal
1011	LT	N≠V [(N=1 AND V=0) OR (N=0 AND V=1)]	Signed less than
1100	GT	(Z=0) AND (N=V)	Signed greater than
1101	LE	(Z=1) OR (N≠V)	Signed less than or equal
1110	AL	—	Always (unconditional)
1111	—	—	This instruction can only be executed unconditionally

ARM adalah desain RISC yang sangat bergantung pada register pengalamatan, ini tampaknya menjadi wajar trade-off.

REFERENSI

Stalling, W. 2010. Computer Organization and Architecture design dan Performance eighth edition. Prentice Hall

PROPAGASI

A. Latihan dan Diskusi (Propagasi vertical dan Horizontal)

10.1 What are the typical elements of a machine instruction?

10.2 What types of locations can hold source and destination operands?

10.3 If an instruction contains four addresses, what might be the purpose of each address?

10.4 List and briefly explain five important instruction set design issues.

10.5 What types of operands are typical in machine instruction sets?

B. Pertanyaan (Evaluasi mandiri)

10.1 Show in hex notation:

a. The packed decimal format for 23

b. The ASCII characters 23

10.2 For each of the following packed decimal numbers, show the decimal value:

a. 0111 0011 0000 1001

b. 0101 1000 0010

c. 0100 1010 0110

10.3 A given microprocessor has words of 1 byte.What is the smallest and largest integer that can be represented in the following representations:

a. Unsigned

b. Sign-magnitude

c. Ones complement

d. Twos complement

e. Unsigned packed decimal

f. Signed packed decimal

C. QUIZ -mutiple choice (Evaluasi)

D. PROYEK (Eksplorasi entrepreneurship, penerapan topic bahasan pada dunia nyata)