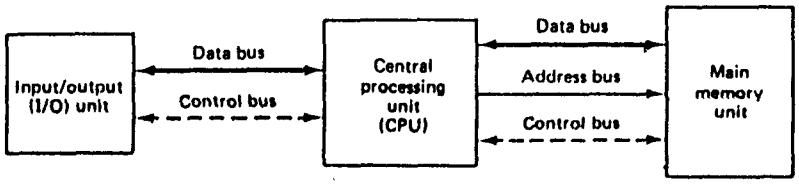


Gambar 4-1 Komponen dasar pada sebuah komputer digital

tukaran informasi dalam komputer dan dengan dunia luar melalui unit I/O. Unit memori terdiri dari sejumlah besar lokasi yang menyimpan program dan data yang sedang aktif digunakan oleh CPU. Ketiga unit tersebut saling dihubungkan dengan memakai berbagai macam bus. Sebuah **bus** dapat dikatakan sekelompok kawat—sebuah jalur fisik—yang berfungsi menghubungkan register-register dengan unit-unit fungsional yang berhubungan dengan tiap-tiap modul. Informasi saling dipertukarkan di antara modul dengan melalui bus.

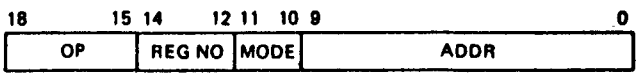
Bus, demikian juga fungsi-fungsi yang berhubungan dengan tiap-tiap unit, akan dibahas secara detail pada Bab 5, 6 dan 7. Bagaimanapun juga, sebelum mempelajari kompleksitas setiap unit pada komputer, kita akan melihat sistem komputer secara keseluruhan, sederhana dan dalam satu kesatuan. Kita mulai dengan membahas operasi-operasi mikro yang menggambarkan penerapan sebuah sistem.

4.2 OPERASI-MIKRO

Organisasi bagian dalam dari sebuah komputer sangat ditentukan oleh kumpulan instruksi yang dapat dijalankannya. Sebuah **instruksi** adalah sebuah kaidah yang digunakan oleh komputer (1) untuk mendefinisikan operasi-operasi seperti *add*, *store*, *load* dan *jump* dan (2) untuk menentukan lokasi data di mana operasi akan dilakukan. Kumpulan dari semua instruksi, disebut **kumpulan instruksi**, mencakup beragam operasi aritmatika dan logika, operasi perpindahan data, operasi masukan/keluaran dan operasi pengendalian. Kombinasi dari operasi-operasi ini, dikelompokkan bersama-sama, membentuk sebuah program mesin.

Kumpulan instruksi bervariasi dari komputer yang satu ke lainnya, begitu pula dengan format instruksinya. Bagaimanapun juga, secara umum sebuah intruksi

komputer merupakan sebuah kode biner yang terbagi atas beberapa **field**. Field operasi, disebut **opcode**, menjabarkan operasi yang harus dilakukan. Operasi ini dijalankan pada beberapa koleksi data, disebut **operand**, yang bisa berupa bagian dari instruksi tersebut atau dimasukkan nilainya pada register atau memori. Jika sebuah operand disimpan dalam sebuah register atau memori, instuksi tersebut harus menjabarkan lokasi tersebut dalam sebuah field **alamat** (*address*). Cara pemilihan operand selama pelaksanaan program tergantung kepada field **mode pengalamatan** dari instruksi itu. Sebagai contoh, format sebuah instruksi bisa berupa:



Dalam kasus ini, digunakan 4 bit untuk menjabarkan opcode dari instruksi tersebut, yang secara keseluruhan memberikan 16 kemungkinan instruksi; satu dari delapan register dijabarkan sebagai sebuah operand oleh 3 bit berikutnya; 2 bit digunakan untuk menjabarkan mode pengalamatan yang akan digunakan untuk operand kedua; dan ada 10 bit untuk menjabarkan address memori sesungguhnya dari operand kedua. Detail lebih lanjut mengenai jenis-jenis umum dari instruksi dan mode pengalamatan akan diberikan pada Bab 5.

Untuk menjalankan instruksi-instruksi yang tersimpan di dalam memori, setiap instruksi **diambil** (*fetched* atau disalin dari memori oleh CPU), ditempatkan ke dalam sebuah register dan dijalankan. Bagaimanapun juga, sebuah instruksi adalah sebuah entitas yang kompleks yang pelaksanaannya tidak dapat dilakukan selama satu periode waktu. Dalam kenyataannya, hal tersebut membutuhkan lebih dari satu periode waktu untuk mem-fetch instruksi. Karena itu CPU menghasilkan urutan dari fungsi-fungsi yang mem-fetch instruksi. Fungsi-fungsi ini disebut sebagai **operasi-mikro** (*micro-ops*), atau terkadang sebagai **instruksimikro**. Operasi mikro adalah sebuah operasi tingkat rendah yang dapat dilakukan dalam satu periode waktu. Urutan dari operasi mikro disebut sebagai **siklus CPU**. Siklus yang paling umum adalah **pengambilan** (*fetch*), **penterjemahan alamat**, **eksekusi** dan **interupsi**. Hal-hal tersebut akan dibahas lebih lanjut pada Bagian 4.3.

Karena operasi mikro menyebabkan data ditransfer antarregister, mereka dapat digambarkan dengan menggunakan suatu **bahasa transfer register**. Tidak ada standar tunggal untuk bahasa ini, meskipun banyak versi yang sangat mirip.

Disini, kita akan menjelaskan bahasa transfer register yang akan kita pakai untuk menjabarkan operasi-operasi mikro.

Bahasa Transfer Register

Sebuah **register** *n*-bit adalah sebuah kumpulan dari sejumlah *n* flip-flop yang yang mampu menampung sejumlah *n* bit informasi biner. Tambahan bagi flip-flop, sebuah register memiliki gerbang yang melakukan dan/atau mengendalikan operasi logika tertentu, seperti LOAD, TRANFER dan SHIFT. Waktu pelaksanaan operasi-operasi ini tergantung dari *clock*, seperti yang nanti akan kita lihat pada bab ini.

Bahasa transfer register yang disajikan di sini akan disebut sebagai RTL (*Register Transfer Language*). Untuk memperkenalkan notasi RTL, anggaplah bahwa kita akan mentransfer isi register A ke register B. Dalam RTL operasi ini dinotasikan sebagai

$$B \leftarrow (A) \tag{4.1}$$

dimana (A) melambangkan isi register A. Operasi transfer $\leftarrow$, berlaku untuk transfer pasangan-bit antara register A dan B. Dengan X_i mewakili bit ke-*i* pada register X, kita dapat juga menulis operasi (4.1) sebagai

$$B_i \leftarrow (A_i), \text{ untuk } i = 0, 1, ..., n\text{-}1 \tag{4.2}$$

dimana *n* adalah ukuran register A dan B dalam bit. Dalam kedua operasi transfer register ini, isi register A tidak berubah dan isi register B sebelumnya berubah. (Perhatikan bahwa hal ini mempunyai efek yang sama seperti perintah tugas pada bahasa pemrograman tingkat tinggi.)

Tambahan bagi proses transfer seluruh register [operasi (4.1)] atau masing-masing bit pada register [operasi 4.2], dapat juga terjadi proses transfer atas bagian-bagian dari register yang disebut **field**. Sebuah field pada sebuah register dinotasikan dengan menggunakan tanda kurung. Pembatasan yang ada di sini hanyalah bahwa ukuran dari field atau register yang terletak di sebelah kiri maupun kanan tanda panah harus sama besar. Sebuah contoh untuk jenis transfer ini adalah

$$PC \leftarrow (IR[AD]) \tag{4.3}$$

Pada contoh ini, field AD dari register IR ditransfer ke register PC. Namun, field AD harus punya ukuran yang sama dengan keseluruhan register PC. Jika ada bagian dari sebuah register yang tak bernama harus ditransfer, mereka dapat dinotasikan sebagai jarak lokasi bit. Sebagai contoh, pada operasi

$$R1[0..3] \leftarrow (X) \tag{4.4}$$

isi register X ditransfer ke bit 0 sampai 3 pada register R1. Hal ini berarti bahwa X mempunyai panjang 4 bit.

Selain itu, dapat juga dipakai konstanta pada sisi sebelah kanan tanda panah. Di sini

$$L \leftarrow 5 \tag{4.5}$$

artinya simpan nilai 5 di register L. Perhatikan bahwa tidak mungkin untuk menulis (5) di sisi sebelah kanan pada operasi (4.5). Tidak mungkin kita menuliskan

$$L_1 \leftarrow 5 \tag{4.6}$$

karena nilai 5 tidak dapat disimpan dalam satu bit saja.

Dengan menggunakan RTL, kita dapat menggambarkan berbagai macam operasi-mikro aritmatika dan logika. Sebagai contoh, perhatikan operasi-mikro ADD dijabarkan sebagai

$$A3 \leftarrow (A1) + (A2) \tag{4.7}$$

Operasi ini berarti bahwa isi register A1 dan A2 dijumlahkan, dengan menggunakan sirkuit adder biner dan hasil jumlahnya ditransfer ke register A3. Pengulangan penjumlahan tersebut akan menyebabkan *overflow*. Untuk menampung overflow, dapat digunakan register 1-bit sebagai pelengkap A3. Dengan menggunakan V sebagai register overflow, operasi (4.7) dapat ditulis kembali sebagai

$$VA3 \leftarrow (A1) + (A2) \tag{4.8}$$

Tambahan bagi operasi-mikro ADD; beberapa operasi aritmatika lainnya yang dapat direpresentasikan di sini adalah

```
A ← (A) + 1        ; increment isi A oleh 1
A ← (A) - 1        ; decrement isi A oleh 1
```

A ← (A) ; menurunkan komplemen A
A ← (A) + (B) + 1 ; lakukan A - B dengan menambahkan komplemen 2's B ke A

Seperti yang ditunjukkan pada contoh-contoh tersebut, kata keterangan (jika ada) mengikuti tanda titik-koma. Perhatikan juga bahwa hanya operasi aritmatika sederhana yang termasuk pada contoh-contoh di atas. Ingat bahwa operasi transfer register ini mewakili operasi-mikro. Jadi meskipun sintaksis dari sebuah operasi perkalian sudah jelas, semantiknya berarti bahwa CPU dapat melakukan perkalian tersebbut dalam periode satu clock. Karena hal seperti ini biasanya jarang terjadi, kita tidak melibatkan operasi aritmatika yang lebih rumit pada pembahasan RTL kita.

Bagaimanapun juga, kita dapat menggambarkan operasi logika pada satu register atau lebih. Sebagai contoh,

$$C \leftarrow (A) \text{ OR } (B) \tag{4.9}$$

mempunyai arti bahwa logika OR dari isi register A dan B ditransfer ke register C. Begitu juga operasi AND akan tampak seperti berikut:

$$C \leftarrow (A) \text{ AND } (B) \tag{4.10}$$

Operasi transfer register geser juga dapat dengan mudah digambarkan. Pada sebuah register geser (lihat Bagian 3.7), isi setiap sel digeser ke sel di sampingnya — ke kanan untuk register geser-kanan dan ke kiri untuk register geser kiri. Apa yang terjadi pada sel yang terakhir tergantung pada pergeseran yang dilakukan, sirkular atau serial. Kita dapat menggambarkan pergeseran serial ke kiri pada sebuah register 4-bit A, dengan input dari register 1-bit B, sebagai berikut:

$$\begin{aligned} \text{hilang} &\leftarrow (A_3) \\ A3 &\leftarrow (A_2) \\ A2 &\leftarrow (A_1) \\ A1 &\leftarrow (A_0) \\ A0 &\leftarrow (B) \end{aligned}$$

RTL juga dapat digunakan untuk menggambarkan transfer data ke dan dari word memori. Setiap memori sebenarnya adalah sebuah register yang dinamai sesuai address memorinya. Dalam RTL, kita menganggap unit memori utama pada komputer sebagai M dan menulis word ke-*i* dalam memori sebagai M[i].

Karena itu, proses **pembacaan memori** (*memory read*) dapat ditunjukkan oleh pernyataan RTL

$$B \leftarrow (M[A]) \tag{4.11}$$

dan proses **penulisan memori** (*memory write*) dinyatakan dalam RTL

$$M[A] \leftarrow B \tag{4.12}$$

Pada kedua operasi tersebut, (4.11) dan (4.12), word memori yang alamatnya ditunjukkan oleh register A ditransfer ke atau dari register B dalam CPU. Pada beberapa komputer, A dan B dapat berupa register umum (*general register*) atau bahkan konstanta, sedangkan pada komputer lainnya, A dan B merupakan register khusus (*special-purpose register*).

Kadang-kadang perlu ditentukan bahwa transfer register hanya akan terjadi pada kondisi tertentu saja. Hal ini dilakukan dengan satu dari dua cara berikut: dengan menggunakan pernyataan **kondisi logikal** (*logical condition*) atau dengan menentukan **kondisi pengendalian** (*control condition*). Pernyataan kondisi logikal merupakan pernyataan IF-THEN yang sederhana. Misalnya, operasi

$$\text{IF } (V) > (W) \text{ THEN } Q \leftarrow 0 \tag{4.13}$$

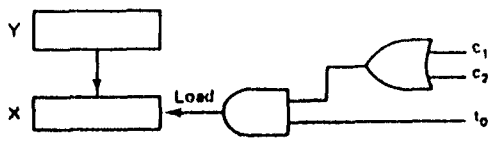
men-*set* 0 ke register Q hanya jika nilai register V lebih besar dari nilai register W. Perhatikan bahwa pernyataan kondisi logikal hanya didefinisikan untuk IF-THEN dan tidak untuk ELSE.

Kondisi pengendalian adalah cara lain untuk menentukan operasi kondisional. Dengan metode ini, kondisinya merupakan fungsi logikal dari variabel biner yang mengatur input register. Fungsi-fungsi ini dijabarkan di sebelah kiri dari operasi transfer register dan diikuti oleh tanda titik dua. Sebagai contoh, perhatikan operasi transfer register yang digambarkan dalam Gambar 4-2. Transfer ini digambarkan dengan pernyataan RTL sebagai berikut:

$$t_0(c_1 + c_2): X \leftarrow (Y) \tag{4.14}$$

Ini berarti bahwa isi Y dipindahkan ke X hanya jika t_0 bernilai 1 dan salah satu c_1 atau c_2 juga bernilai 1. Untuk menandakan bahwa transfer harus terjadi jika kondisi tertentu adalah 0, digunakan simbol utama (') harus digunakan. Sehingga jika kondisi pernyataan RTL pada persamaan (4.14) adalah $t'_0(c_1 + c_2)$:, maka transfer hanya akan terjadi jika t_0 bernilai 0 dan salah satu c_1 atau c_2 bernilai 1.

Kita telah menjabarkan konsep dasar RTL. Lebih lanjut kita akan melihat lebih banyak pernyataan RTL yang kita gunakan untuk menggambarkan suatu sistem tertentu.



Gambar 4-2 Transfer register yang diatur oleh sebuah fungsi pengendalian

4.3 SEBUAH CONTOH KOMPUTER SEDERHANA

Pada bagian ini, kita masuk ke dalam detail perancangan arsitektur untuk sebuah komputer yang sangat sederhana. Komputer yang kita gunakan didasarkan pada *Simplified Instructional Computer* (**SIC**) yang dipersembahkan oleh Beck (1985). Di sana, SIC dijabarkan oleh ukuran memori, ukuran word, register, format data, format instruksi, mode pengalamatan, kumpulan instruksi dan provisi input/output-putnya. Berdasarkan informasi ini, kita dapat menentukan sebuah rancangan arsitektur untuk SIC. Tentu saja, ada kemungkinan untuk rancangan lain yang sesuai dengan definisi SIC; namun satu rancangan sudah cukup untuk bahasan ini. Pertama-tama, kita akan menggambarkan struktur SIC (dimodifikasi sedikit dari definisi Beck) kemudian melihat pada detail rancangannya.

Struktur Mesin SIC

Mesin SIC terdiri atas CPU, unit memori dan minimal satu piranti I/O. CPU terdiri atas 13 register khusus. Tujuan dan ukuran register (dalam bit) diberikan dalam Tabel 4.1. Register *akumulator*, A, digunakan untuk semua operasi aritmatika dan logika. Dalam instruksi semacam ini, A selalu merupakan salah satu dari operand dan hasilnya selalu disimpan kembali dalam A. Register *indeks*, X, digunakan untuk menghitung address memori dari operand tertentu tergantung pada mode pengalamatan instruksi. Alamat memori dari instruksi berikutnya