Nama :JANKY DAUSAT

Nim : 202369040082

Jurusan : Teknik informatika /2C

**Tugas individu arsitektur dan organisasi komputer**

1. Jelasakan Apa perbedaan prosesor CISC dengan RISC

2. Jelaskan kekurangan yang ada pada prosesor CISC

3. Jelaskan fungsi-fungsi kerja dari masing-masing register pada prosesor CIS

**Jawaban:**

**1.Perbedaan Prosesor CISC dan RISC**

1. **Definisi:**

* **CISC (Complex Instruction Set Computer):** CISC adalah arsitektur prosesor yang dirancang untuk mengurangi jumlah instruksi per program dengan memungkinkan satu instruksi untuk melakukan berbagai tugas kompleks. Tujuan utamanya adalah untuk meminimalisir jumlah instruksi yang dieksekusi oleh program.
* **RISC (Reduced Instruction Set Computer):** RISC adalah arsitektur prosesor yang menggunakan set instruksi yang lebih kecil dan lebih sederhana. Setiap instruksi dirancang untuk dieksekusi dalam satu siklus clock, dengan tujuan meningkatkan kecepatan dan efisiensi eksekusi instruksi.

2. **Instruksi:**

* **CISC:**
  + Memiliki set instruksi yang besar dan kompleks.
  + Instruksi dapat melakukan operasi kompleks (misalnya, akses memori dan operasi aritmatika dalam satu instruksi).
  + Contoh: Intel x86, VAX.
* **RISC:**
  + Memiliki set instruksi yang kecil dan sederhana.
  + Setiap instruksi umumnya melakukan satu operasi dasar (misalnya, operasi aritmatika atau akses memori sederhana).
  + Contoh: ARM, MIPS, SPARC.

3. **Desain dan Implementasi:**

* **CISC:**
  + Instruksi yang kompleks dapat memerlukan beberapa siklus clock untuk dieksekusi.
  + Prosesor CISC cenderung lebih kompleks dengan berbagai mode addressing dan tipe instruksi.
  + Menggunakan mikrokode untuk mendekode instruksi yang lebih kompleks.
* **RISC:**
  + Instruksi yang sederhana dan seragam memungkinkan eksekusi dalam satu siklus clock.
  + Desain prosesor RISC lebih sederhana dan efisien, dengan sedikit mode addressing.
  + Menggunakan hard-wired control unit untuk dekoding instruksi, yang lebih cepat daripada mikrokode.

4. **Pipelining:**

* **CISC:**
  + Pipelining lebih sulit diimplementasikan karena instruksi yang kompleks dan bervariasi dalam waktu eksekusi.
  + Hazard dan penundaan lebih umum terjadi karena instruksi yang panjang dan kompleks.
* **RISC:**
  + Pipelining lebih mudah diimplementasikan karena instruksi yang seragam dan sederhana.
  + Efisiensi pipelining lebih tinggi karena setiap instruksi memerlukan satu siklus clock.

5. **Efisiensi Memori:**

* **CISC:**
  + Instruksi yang kompleks mengurangi jumlah total instruksi, yang dapat mengurangi penggunaan memori untuk program.
  + Setiap instruksi dapat melakukan lebih banyak kerja, sehingga program mungkin lebih pendek.
* **RISC:**
  + Instruksi yang sederhana dan satu per satu dapat meningkatkan jumlah total instruksi, yang dapat meningkatkan penggunaan memori.
  + Program mungkin lebih panjang karena setiap instruksi hanya melakukan satu tugas dasar.

6. **Contoh:**

* **CISC:**
  + Prosesor Intel x86 (digunakan dalam kebanyakan komputer pribadi).
  + Motorola 68000 series.
* **RISC:**
  + Prosesor ARM (digunakan dalam kebanyakan perangkat mobile).
  + Prosesor MIPS (digunakan dalam berbagai perangkat embedded).

**Ringkasan:**

* **CISC**: Memiliki set instruksi yang besar dan kompleks, bertujuan untuk mengurangi jumlah instruksi per program, lebih sulit untuk melakukan pipelining, dan lebih efisien dalam penggunaan memori.
* **RISC**: Memiliki set instruksi yang kecil dan sederhana, bertujuan untuk meningkatkan kecepatan eksekusi dengan instruksi yang dieksekusi dalam satu siklus clock, lebih mudah untuk melakukan pipelining, dan program mungkin lebih panjang.

**2.**

**1. Kompleksitas Instruksi:**

* **Desain Lebih Rumit:** Prosesor CISC memiliki set instruksi yang sangat besar dan kompleks, sehingga desain hardware menjadi lebih rumit. Ini meningkatkan waktu pengembangan dan biaya produksi.
* **Mikrokode:** Untuk menangani instruksi kompleks, CISC sering menggunakan mikrokode yang dapat memperlambat eksekusi karena instruksi harus diuraikan menjadi langkah-langkah mikro sebelum dieksekusi oleh hardware.

**2. Eksekusi Instruksi yang Lambat:**

* **Multiple Siklus Clock:** Banyak instruksi pada prosesor CISC memerlukan beberapa siklus clock untuk dieksekusi. Hal ini mengurangi kecepatan keseluruhan prosesor dibandingkan dengan arsitektur yang dapat mengeksekusi instruksi dalam satu siklus clock.
* **Pipelining Kurang Efisien:** Kompleksitas instruksi dan variasi dalam waktu eksekusi membuat implementasi pipelining lebih sulit dan kurang efisien. Hal ini menyebabkan lebih banyak hambatan (hazard) dalam pipeline dan penundaan eksekusi.

**3. Overhead Dekoding Instruksi:**

* **Dekoder Instruksi:** Karena keragaman instruksi yang besar, dekoder instruksi pada prosesor CISC harus lebih kompleks, yang menambah overhead dan dapat mengurangi efisiensi.
* **Pemanfaatan Register:** Instruksi yang kompleks sering kali mengurangi pemanfaatan register secara efisien karena banyak operasi yang melibatkan akses memori langsung.

**4. Efisiensi Energi yang Lebih Rendah:**

* **Konsumsi Daya:** Kompleksitas desain dan kebutuhan untuk mengelola instruksi yang kompleks meningkatkan konsumsi daya. Ini kurang ideal untuk aplikasi yang membutuhkan efisiensi energi tinggi, seperti perangkat mobile.
* **Panas Berlebih:** Prosesor dengan konsumsi daya tinggi juga menghasilkan lebih banyak panas, yang dapat menyebabkan masalah termal dan membutuhkan solusi pendinginan tambahan.

**5. Skalabilitas yang Terbatas:**

* **Ekspansi Set Instruksi:** Menambah instruksi baru ke set instruksi yang sudah kompleks bisa menjadi sulit dan sering kali memerlukan perubahan signifikan dalam desain hardware dan mikrokode.
* **Upgrade yang Sulit:** Mengupgrade arsitektur CISC untuk meningkatkan performa sering kali membutuhkan perubahan besar, yang bisa meningkatkan biaya dan waktu pengembangan.

**6. Penggunaan Memori:**

* **Efisiensi Memori:** Meskipun instruksi yang kompleks dapat mengurangi jumlah total instruksi yang dieksekusi, program-program tersebut sering kali memerlukan lebih banyak ruang memori karena ukuran instruksi yang lebih besar dan kompleksitasnya.
* **Overhead Mikrokode:** Mikrokode yang digunakan untuk mengelola instruksi kompleks juga memerlukan ruang memori tambahan, yang bisa mengurangi jumlah memori yang tersedia untuk aplikasi.

**7. Kompatibilitas Mundur:**

* **Instruksi Warisan:** Prosesor CISC seperti Intel x86 sering kali harus mempertahankan kompatibilitas dengan instruksi-instruksi lama. Ini membatasi fleksibilitas dalam desain dan optimalisasi arsitektur baru karena harus mempertahankan dukungan untuk instruksi lama yang mungkin sudah tidak efisien.

**3.**

**1. General-Purpose Registers (GPR):**

* **Fungsi:** Register serbaguna yang digunakan untuk menyimpan data sementara selama eksekusi instruksi. Mereka dapat menyimpan operand aritmetika, alamat memori, atau hasil dari operasi.
* **Contoh:** Pada prosesor Intel x86, register EAX, EBX, ECX, dan EDX sering digunakan untuk operasi aritmetika dan logika.

**2. Segment Registers:**

* **Fungsi:** Menyimpan segmen alamat memori yang digunakan dalam model memori segmentasi. Register ini membantu dalam menentukan lokasi segment memori yang sedang diakses.
* **Contoh:** Pada prosesor x86, register CS (Code Segment), DS (Data Segment), SS (Stack Segment), dan ES (Extra Segment).

**3. Index and Pointer Registers:**

* **Fungsi:** Digunakan untuk pengalamatan dan pengindeksan memori. Mereka sering digunakan dalam operasi yang melibatkan array atau pemrosesan string.
* **Contoh:**
  + **SI (Source Index) dan DI (Destination Index):** Digunakan untuk operasi string.
  + **BP (Base Pointer) dan SP (Stack Pointer):** BP digunakan untuk mengakses data pada stack frame, dan SP digunakan untuk melacak top of the stack.

**4. Instruction Pointer (IP) Register:**

* **Fungsi:** Menyimpan alamat dari instruksi berikutnya yang akan dieksekusi. IP otomatis diperbarui untuk menunjuk ke instruksi berikutnya selama eksekusi program.
* **Contoh:** Pada arsitektur x86, disebut EIP dalam mode 32-bit dan RIP dalam mode 64-bit.

**5. Flags/Status Registers:**

* **Fungsi:** Menyimpan informasi tentang status operasi yang baru saja dilakukan, seperti hasil perbandingan atau aritmetika. Flag ini digunakan untuk pengambilan keputusan dalam instruksi bersyarat.
* **Contoh:**
  + **ZF (Zero Flag):** Diset jika hasil operasi adalah nol.
  + **CF (Carry Flag):** Diset jika ada carry out dari operasi aritmetika.
  + **SF (Sign Flag):** Diset jika hasil operasi negatif.

**6. Control Registers:**

* **Fungsi:** Menyimpan informasi kontrol yang mengatur operasi CPU, seperti mode proteksi, paging, dan debugging.
* **Contoh:** Pada arsitektur x86, register CR0, CR1, CR2, dan CR3 digunakan untuk berbagai fungsi kontrol.

**7. Debug Registers:**

* **Fungsi:** Digunakan untuk debugging perangkat keras dengan menyimpan breakpoint dan kontrol debug.
* **Contoh:** Pada arsitektur x86, register DR0 hingga DR7 digunakan untuk debugging.

**8. Floating-Point Registers:**

* **Fungsi:** Digunakan untuk operasi floating-point (desimal) yang memerlukan presisi tinggi.
* **Contoh:** Pada arsitektur x86 dengan FPU (Floating-Point Unit), register ST0 hingga ST7 digunakan untuk operasi floating-point.

**9. Segment Descriptor Registers:**

* **Fungsi:** Menyimpan deskriptor segmen yang menentukan basis, limit, dan atribut segmen memori. Digunakan dalam mode proteksi untuk akses memori yang aman.
* **Contoh:** Pada arsitektur x86, register GDTR (Global Descriptor Table Register) dan LDTR (Local Descriptor Table Register).

**10. Model-Specific Registers (MSR):**

* **Fungsi:** Register khusus yang digunakan untuk konfigurasi perangkat keras dan optimasi kinerja. MSR menyediakan informasi spesifik model CPU dan dapat digunakan untuk pemantauan kinerja.
* **Contoh:** Pada prosesor modern seperti Intel dan AMD, MSR digunakan untuk berbagai konfigurasi kinerja dan pengaturan daya.