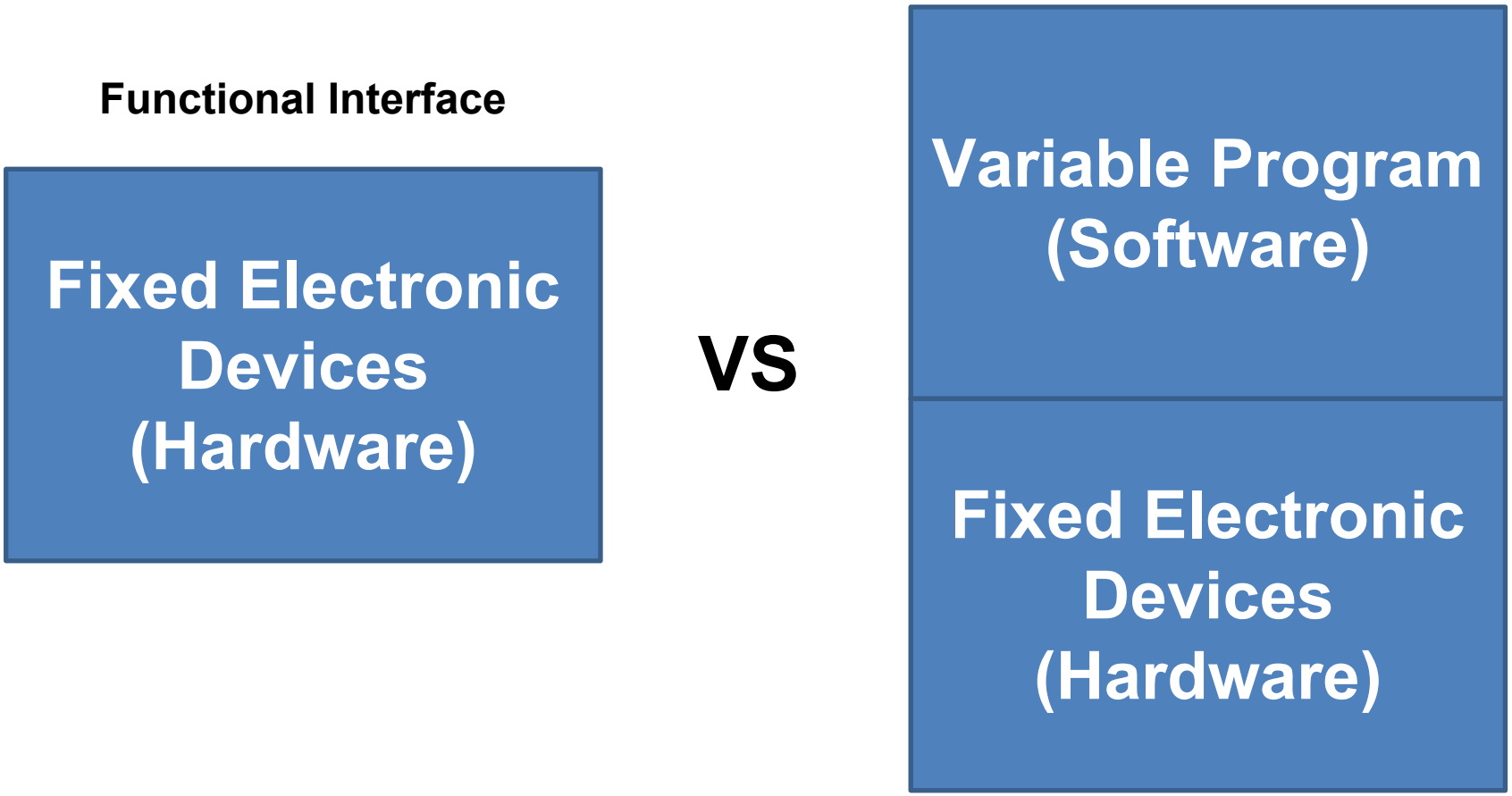
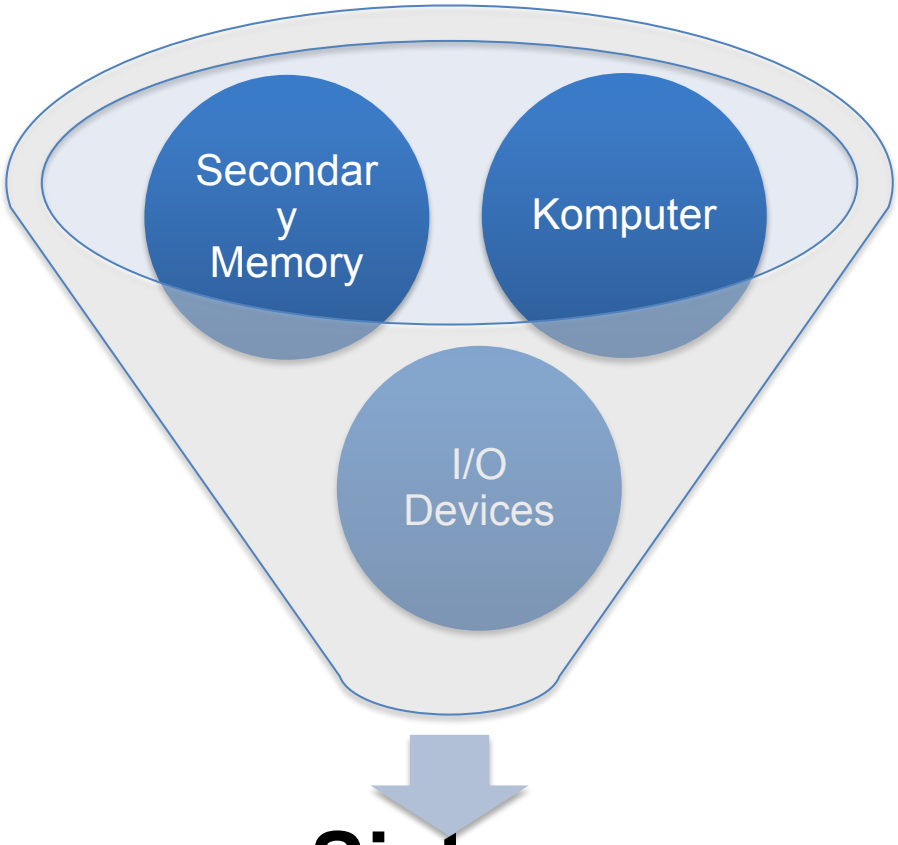
Pengenalan Sistem Komputer


Functional Interface
Fixed Electronic Devices (Hardware)
VS
Variable Program (Software)
Fixed Electronic Devices (Hardware)

Unit-unit operasional dan interkoneksi untuk mewujudkan spesifikasi arsitektur.

1. Aspek yang memberikan pengaruh langsung kepada eksekusi logik dari program.
2. Atribut yang “visible” kepada programmer.



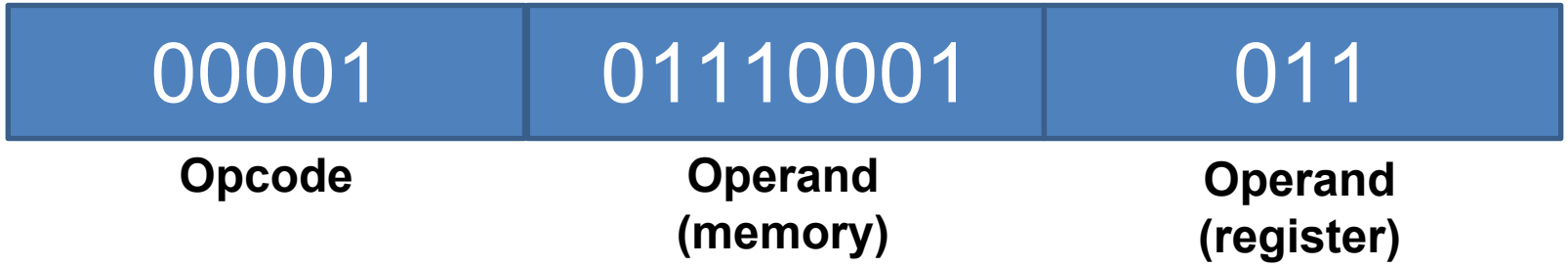
Sistem Komputer

1. Data Movement
2. Data Storage
3. Processing I/O to Storage
4. Processor from Storage to I/O

Melakukan operasi aritmatika dan logik.

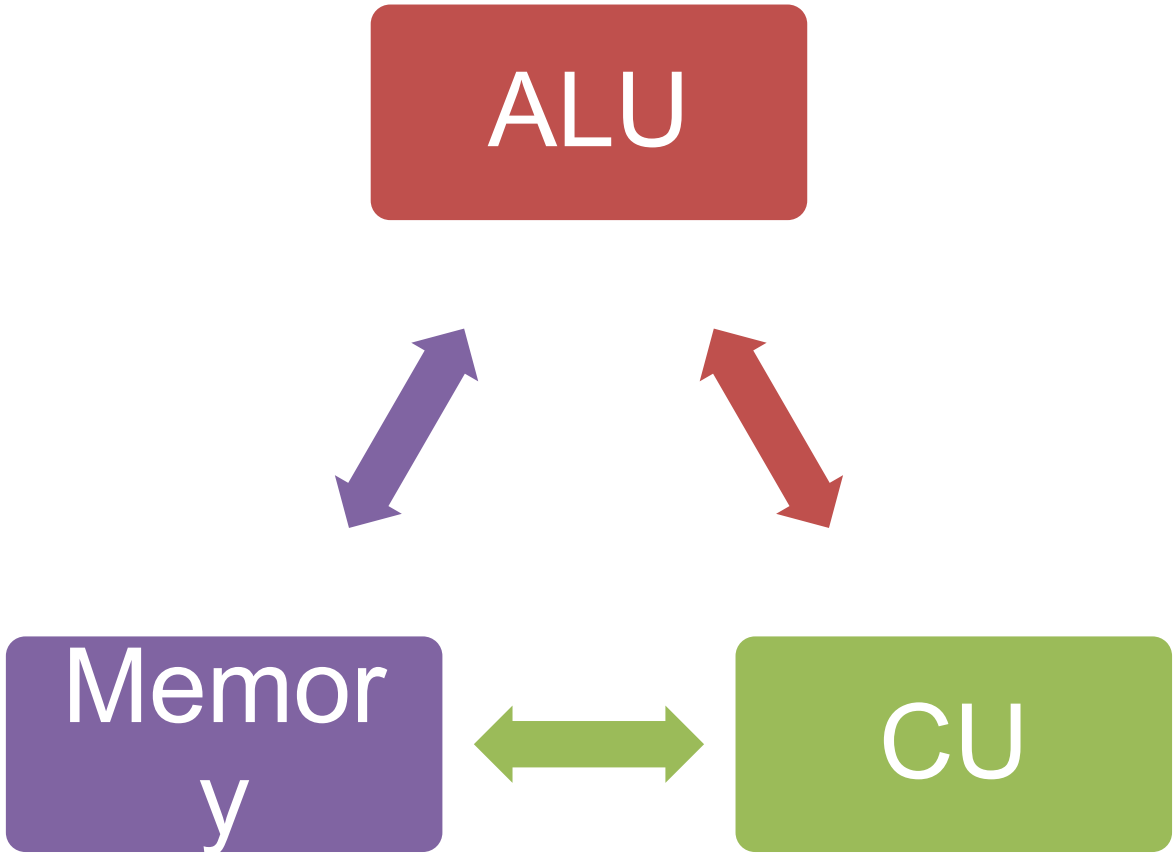
1. Menentukan urutan operasi atas instruksi yang tersimpan pada executable memory.
2. Retrieve dan Execute instruksi.

Menyampaikan data dari/ke prime memory dan menyimpannya pada media yang lebih permanen.


00001
01110001
011
Opcode
Operand
(memory)
Operand
(register)

Pipelining

1. Ambil instruksi (Fetch Instruction)
2. Lihat tabel instruksi (Decode)
3. Ambil operand (Fetch Operand)
4. Eksekusi (Execute Instruction).


ALU
Memor
y
CU

1. Untuk memaksimalkan tingkat keparalelan perbanyak overlap.
2. 1 stage instruksi = satu siklus mesin (clock)

$$(T_{ex}/2)*8=4*T_{ex}$$

$$(T_{ex}/6)*12=2*T_{ex}$$

1. Structure Hazard
2. Data Hazard
3. Control Hazard

Hazard yang terjadi ketika resource (memori atau unit fungsional) dibutuhkan oleh dua instruksi yang berbeda pada waktu yang bersamaan.

Penalti: 1 Cycle

Hazard yang terjadi ketika ada instruksi yang membutuhkan data dari instruksi sebelumnya (data depedency).

Penalti: 2 Cycle

1. Hazard yang terjadi ketika ada percabangan.
2. Dua macam percabangan:
    a. Unconditional Branch
    b. Conditional Branch

Penalti: 3 Cycle

1. Branch is taken.

2. Branch is not taken.

Penalti: 3 Cycle

Penalti: 3 Cycle

Instruction Pipelining

1. Percabangan dapat menyebabkan penalti yang mengganggu dalam pipeline.
2. Cara reduksi:
    a. Delayed Branching
    b. Branch Prediction

1. Ambil satu instruksi yang belum dijalankan dan tidak punya depedensi dengan branch.
2. Tempatkan tepat di bawah branching (Branch Delayed Slot).
3. Jika tidak ada instruksi yang independent maka tempatkan NOP (no operation) di Branch Delayed Slot.

1. Ambil target dari branching (tidak peduli percabangan diambil atau tidak)
2. Kalau branching diambil maka penalti tidak bertambah.
3. Kalau branching salah maka penalti bertambah

1. Static Branch Prediction

   Branch prediction yang tidak memperhatikan histori dari branching yang sudah dilakukan.

2. Dynamic Branch Prediction

   Branch prediction yang memperhatikan histori dari branching yang sudah dilakukan.

1. Predict kondisi selalu tdk terpenuhi/ never taken (Motorolla 68020).
2. Predict kondisi selalu terpenuhi/ always taken.
3. Predict depending on the branch prediction.

1. One bit prediction.

2. Two bits prediction.

RISC

Arsitektur yang meningkatkan kinerja CPU dengan menyederhanakan set instruksi dari CPU.

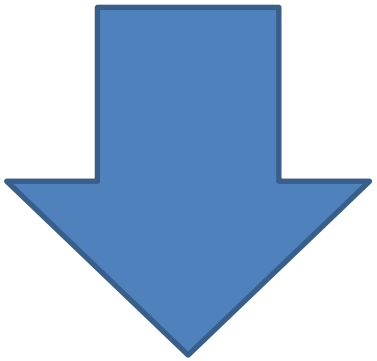
1. Set instruksi dibatasi dan hanya terdiri dari instruksi-instruksi yang sederhana.
2. Load and Store architecture.
3. Instruksi hanya menggunakan beberapa mode pengalamatan.

4. Instruksi memiliki panjang yang tetap dan bentuk yang seragam.
5. Memerlukan jumlah register yang banyak.

1. Banyak yang berbda pendapat jika ditanya tentang perihal manakah yang lebih baik antara RISC dan CISC.
2. Cara pandang RISC lebih menguntungkan jika instruksi yang diolah sederhana.
3. Cara pandang CISC lebih menguntungkan jika instruksi yang diolah sangat kompleks.

Superscalar

Ada lebih dari satu instruksi pada stage yang sama dapat dieksekusi.



Penggunaan unit fungsional yang berbeda.

Kendala	Pemecahan
Resource conflict	Menambah resource
Control dependency	1. Mengurangi branch penalti 2. Fixed instruction length 3. Fixed instruction format
Data conflict	Ada kebebasan menentukan instruksi

1. Meningkatkan level dari parallelisme.
2. Penjadwalan instruksi yang dinamis.
3. Speculative Execution.

1. Jumlah dari parallel pipeline.

2. Jumlah dari instruksi yang independent.

VLIW

1. Compiler menganalisa program sebelum run program.
2. Mendeteksi instruksi-instruksi yang bisa dideteksi secara paralel.
3. Semua instruksi/operasi dibungkus dalam large instruction.

1. Tidak ada run time detection of parallelism.

2. Tidak ada masalah limitation of window size.

1. Simple hardware.

2. Global analysis of computer program

1. Perlu banyak register untuk mempertinggi parallelism.
2. Perlu bandwidth yang besar untuk komunikasi.
3. Kemungkinan wasted pada instruction word.
4. Tidak semua arsitektur komputer cocok menggunakan VLIW.

1. Loop Unrolling.
2. Trace scheduling.
3. Branch Prediction.
4. Speculative loading.

1. Memanfaatkan field instruksi yang masih kosong.
2. Melakukan unroll instruksi terhadap field instruksi dan clock cycles yang masih kosong sehingga tingkat keparallelan semakin meningkat.
3. Butuh dukungan dari register dan bus yang besar.

1. Trace selection.
2. Instruction scheduling.
3. Compensation.

1. Tidak ada waktu yang hilang.
2. Jika branch benar maka commited sebaliknya discarded.
3. Memperhatikan data dependency.

Load dipindah ke urutan atas instruksi

untuk menghindari memory latency.

Arsitektur Komputasi Paralel

1 CPU ≈ Keterbatasan

Subjek:

N CPU untuk 1 aplikasi.

1. Jumlah CPU dan kompleksitas masing-masing CPU.
2. Ketersediaan common memory.
3. Topologi interkoneksi antar CPU.
4. Performa jaringan interkoneksi.
5. I/O device.

1. Single Instruction Single Data.
2. Single Instruction Multi Data.
3. Multi Instruction Single Data.
4. Multi Instruction Multi Data.

1. Seberapa jauh pengembang arsitektur bisa exploitir parallelism.
2. Real application seberapa cepat?
3. Bagaimana mengukur peningkatan kinerja komputer parallel.

Relevansi dengan Multithreading

1. Fine grained multithreading berelevansi dengan delayed branching.
2. Fine grained mengeksekusi thread yang berbeda pada setiap clocknya dan mengabaikan thread yang stall. Konsep ini mirip dengan delayed branching.

3. Delayed branching menunda pengeksekusian branching dengan instruksi yang lain (reduce control hazard penalties).
4. Perbedaannya terletak di stall. Stall terjadi belum tentu karena control hazard.

1. Course grained multithreading berelevansi dengan delayed branching.
2. Course grained menukar thread ketika threadnya terjadi stall.
3. Thread lain akan menunggu thread utama stall terlebih dahulu sebelum thread lain dieksekusi.

1. SMT berelevansi dengan renaming register.
2. Renaming register dilakukan jika ada konflik pada resource yang sama maka salah satu akan ditukar registernya supaya tidak bentrok.

3. SMT juga berelevansi dengan VLIW.
4. SMT dan VLIW sama-sama membutuhkan register tambahan untuk meningkat performa.
5. SMT meningkatkan penjadwalan menggunakan renaming table.
6. VLIW mempertinggi parallelisme.