01

...

02

CPU

Leistungsmerkmale CPU := Befehlszahl, Zykluszeit, CPI

- Viele Befehle => Befehle zur Berechnung eines Problems sinkt
  - Anzahl Befehle hängt vom Compiler und Befehlssatz der CPU ab
- Taktzyklus(zeit) (Angabe als Frequenz): e.g. 4GHz = 0.25ns
  - Kleine Taktzykluszeit => viele Befehle können pro Zeiteinheit ausgeführt werden
  - Abh. von CPU-Implementierung
  - soll minimal sein; Optimierung nach seltenen Befehlen (nach häufigen -> nicht möglich), Reduktion von Leerlaufzeiten
- CPI := clock cycles per instruction (Taktzyklen pro Befehl, d.h. øAnz. Taktzyklen, welche ein Befehl zur Ausführung benötigt)
  - Abh. von CPU-Implementierung
- Ausführungsdauer Program P: tP= (Anz. Befehle) * Zykluszeit * CPI = e.g. 100 000 * 2ns * 1 (CPI=1) = 0.2ms
- Komponenten = {Befehlzähler/Befehlsregister, Steuerwerk, Register, ALU=Rechenwerk} // inkl. dazugehörige Steuer-, Daten-, Adressleitungen, Datenlogik, Speicher (entspricht d. Von-Neumann-Rechner-Architektur)
  - Für die Bearbeitung von Befehlen wird auf den Speicher zurückgegriffen
  - Befehle führen arithmetisch-logische Funktionen aus (mittels ALU ~ Arithmetisch Logical Unit)
- Schaltnetz
  - besteht aus logischen Bauelementen (keine Speicherbausteine)
  - Eingangswerte bestimmen Ausgangswerte
  - Kombination von Schaltnetzen sind Schaltnetze
  - e.g. Schaltnetz
    - 3-Bit-Multiplexer: n Steuereingänge, 2^n Eingänge, 1 Ausgang (genau 1 Eingang wird via Steuerausgänge selektiert und mit Ausgang verbunden/durchgeschaltet)

    * 3-Bit-Demultiplexer: n Steuereingänge, 1 Eingang, 2^n Ausgänge (Eingang wird via Steuereingänge mit genau einen Ausgang verbunden/durchgeschaltet)

- Optimierte Schaltungen
  - e.g. Addierer (theor. realisierbar über bools. Schaltungen, in Praxis aber Addierwerke := spezielle bzl. Geschwindigkeit optimierte Schaltungen)
  - e.g. Optimierte Schaltungen: Halb- & Volladierer, Carry-Ripple, Carry-Skip, Carry-Look-Ahead, Conditional Sum Addition, Carry-Select
- Schaltwerk
  - können intern Datenwerkte speichern
  - min. 1 Dateneingang, 1 Datenausgang, 1 Takteingang
  - Takteingang bestimmt, wann Werte geschrieben werden
  - Unterscheidung: pegel- und flankengesteuerte (edge triggered) Schaltwerke
  - heute: meist frankengesteuerte genutzt (Hazard, Glitch, race-condition)
- Schaltwerk vs. Schaltnetz
  - Schaltwerke = Speicher, Schaltnetz = Intelligenz
  - Eingabe von: Schaltwerk, Berechnung: Schaltnetz, Ausgabe nach: Schaltwerk (SW1 -> SN -> SW2)
  - Bei flankengest. Schaltw.: Lesen & Schreiben von/ins gleiche Schaltw. (SW <- SN)
- Zykluszeit (Faktoren)
  - Architektur, Hardware, Befehlssatz so designen, dass alle Befehle gleich lange brauchen (zur Ausführung)
  - Befehle in Gruppen unterteilen, die in etwa gleich lang brauchen (für Ausführung)
  - Pipelining: Mehrere Befehle überlappend/zeitgleich ausführen

  1. Befehl wird geladen
  2. Befehl wird decodiert (Steuerwerk)

  - Operanden werden bereitgestellt
  - Rechenoperation wird durchgeführt (ALU) + Ergebnis wird geschrieben
- Pipelining
  - structural hazard := Gleichzeitiger Zugriff auf Ressourcen durch aufeinanderfolgende Befehle

  - data hazard := Befehl B greift auf Daten des Befehls A zu, wobei A noch nicht abgeschlossen ist
  - control hazard := ..
  - Branch prediction: sprungbefehle können zu >90% vorhergesagt werden, Ursache: meist eingesetzt bei Schleifen. e.g. bei n Durchgängen wird n-1 mal richtig geraten, beim Verlassen des Loops falsch geraten. Daher Fehlerquote <10%!

03

Einführung

- Wörter ~ Befehle, Sprache ~ Befehlssatz (instruction set)
- Intelligenz im Rechner steckt im Programm (Rechner wird von Befehlen gesteuert)
- Video: Security Phorensiker, der Viren mit public domain sw programmiert hat, welche via bluetooth o.ä. eingeschläust werden kann, auf HW-Ebene. Theoretisch möglich: HW manipuliert (China), SW manipuliert bzw. OS (NSA)
- Kryptographie/Internetsicherheit (als Vertiefung)
- Begriff := Rechner berechnet arithmetisch-logische Funktionen (e.g. arithmetisch: Addition, logisch: OR)
  - e.g. OP-Code, Operand-1, Operand-2, Operand-3, (Option)
  - e.g. ADD, a, b, s, - // s=a+b, keine Option (-)
  - e.g. OR, u, v, w, - // w = u OR v, keine Option (-)
  - e.g. ADD, R-1, 100, R-2 // Addition mit Register (R-1, R-2) und Wert 100 (Schreibe (Inhalt R-1)+100 nach R-2)
- Register: sehr schnell, teuer, hardwaretechnisch gross (daher nicht beliebig viele einsetzbar). Daher werden arithmetisch-logische Funktionen oft mittels Operanden-Register berechnet.

Sprungbefehle

- Theoretisch 1 Sprungbefehl ausreichend, um alle abzudecken. Denoch werden verschienene implementiert, da somit der Umfang des Programmcodes erheblich reduziert werden kann ("Bequemlichkeit")
- e.g. Bnull, R-1 := Branch if null
- e.g. IF Condition: `... Test 2x0 // test if condition true Jump // jump to line xy`
- e.g. CASE SWITCH: mehrere IF-Anweisungen nacheinande
- e.g. FOR LOOP: TBD!!!!!!!!!!!!!!!!!
- e.g. Prozedur

04 TBD

- Hit Time ~ Zugriffszeit bei Treffer
- Miss Penalty ~ Fehlzugriffaufwand

05

Caches

- Optimierung des Speichers bringt am meisten Performance
- Mehr Taktrate ~ mehr Hitze, mehr Stromverbrauch, heute daher in der Entwicklung nicht mehr so interessant
- Cache kann die Rechenleistung erheblich steigern -> Verbesserung der Leistungsfähigkeit des Caches bringt erheblich mehr als Steigerung der Rechengeschwindigkeit (e.g. Taktrate)

-