Fortsetzung DemoComPlus

Erweiterung des Speichers auf $2^8 = 256$ Adressen à 8 Bit:

$$\underbrace{\underbrace{4\,Bit}_{Opcode}\ \underbrace{4\,Bit}_{Objekt}}_{\rightarrow 8\,Bit} \quad \rightarrow \quad \underbrace{\underbrace{8\,Bit}_{Opcode}\ \underbrace{8\,Bit}_{Objekt}}_{\rightarrow 16\,Bit}$$

- Busgröße von 8 Bit → Zwischenspeicher wird durch Register Z eingebaut
  ➔ Speichert die Adresse eines Befehls
- Beispiel: Ablauf des Befehls LDA 00 0F (00000000 00001111)
  Steuerschritt 1 (S1): PC → BUS, BUS → MAR
  S2: RAM → BUS, BUS → IR
  S3: PC = PC + 1
  S4: PC → BUS, BUS → MAR
  S5: RAM → BUS, BUS → AKKU
  S6: PC = PC + 1
  ➔ S1-S3: FETCH Phase, S4-S6: EXECUTE Phase
- Neuer Befehl, welcher im DemoComPlus dazu dient, Werte in bestimmten Speicheradressen zu speichern → STA <Adresse>
- STA Ablauf (ab S4, ohne Nutzung des Registers Z):
  S4: PC → BUS, BUS → MAR
  S5: RAM → BUS, BUS → MAR
  S6: AKKU → BUS, BUS → RAM, PC = PC + 1

Pipelining:

- Unterteilung eines Machinenbefehls in Stufen → FETCH, DECODE, OPERAND, EXECUTE, WRITE-BACK
- Bei bestimmten Schritten arbeiten nicht alle Bestandteile des Prozessors
  ➔ Es kann daher in den nicht arbeitenden Bestandteilen bereits der Nachfolgebefehl bearbeitet werden
  ➔ Fließbandprinzip → während eine Stufe etwas macht, kann die davorliegende schon den nächsten Befehl bearbeiten
- Bsp. für die Ersparnis:
  - Ohne Pipelining: 1 Befehl → 5 Stufen, 5 Prozesse → 25 Stufen, 10 Prozesse → 50 Stufen
  - Mit Pipelining: 1 Befehl → 5 Stufen (→ Latenz), 5 Prozesse → 9 Stufen, 10 Prozesse → 14 Stufen

  ➔ Auslastung von 100 % für alle Stufen
- Latenzzeit wird benötigt → Pipeline muss erst voll sein, damit der Ablauf optimiert wird
- Probleme in der Praxis:
  - Hauptspeicherzugriffe (→ Hauptspeicher ist langsamer als Prozessor)
  - Sprungbefehle
  - Datenabhängigkeiten
  - Ressourcenkonflikte
  -

- Speicherzugriffe:
    - Mehrere Zyklen nötig, bis der Operand gelesen wird (Praxis ca. 30 Zyklen)
    - In der Pipeline entstehen Lücken, diese können auch nicht gefüllt werden

  ➔ Optimierung: Cache (schneller Zwischenspeicher)
- Sprungbefehle:
    - Bedingung für den Sprung wird erst in der EXECUTE Phase ausgewertet
    - Es sind aber schon Befehle bearbeitet worden, welche nun nicht mehr benötigt werden
    - Im schlimmsten Fall müssen Operationen sogar rückgängig gemacht werden

  ➔ Optimierung: Branch Prediction (Sprungzielvorhersage)
- Branch Prediction (Beispiel):
  500 x wird nach einander gesprungen → der Prozessor wertet dies aus und entscheidet entsprechend, dass es wahrscheinlich ist, dass im 501. Durchlauf auch gesprungen wird

  ➔ Dadurch können einige unnötige Befehle weggelassen werden, diese belegen nicht unnötig die Pipeline
- Verwendung von Unterprogrammen ist sinnvoll, dadurch lassen sich Bedingungsfälle umschreiben, in denen der Prozessor oft springen müsste

Caching:

- Der Cache ist ein schneller Zwischenspeicher, der aus einem SRAM Speicher und einem Controller besteht
- Er befindet sich „zwischen“ CPU und Hauptspeicher und ist in beiden Richtungen (CPU → Speicher, Speicher → CPU) mit den entsprechenden Busleitungen verbunden
- Beispiel:
    - Adresse 500 wird aufgerufen → hat der Cache diese gespeichert, sendet er diese direkt an die CPU, was wesentlich schneller geschieht, als wenn diese erst aus dem Hauptspeicher geladen werden müsste
    - Adresse 501 wird aufgerufen → diese hat der Cache nicht gespeichert, liest sie allerdings ab, wenn der Hauptspeicher sie sendet und speichert diese

  ➔ Der Cache speichert auch beim Schreiben in den Hauptspeicher von der CPU zwischen, er kennt also die am häufigsten benutzten Speicheradressen und deren Inhalte → der Cache kann die CPU dadurch schneller mit den Speicherinhalten bedienen
- Der Cache-Controller muss die Häufigkeiten zählen und anhand dieser ermitteln, welche Speicherinhalte gelöscht werden können (weil diese nicht so oft abgefragt wurden)
- Welche Daten vermutlich als nächstes in einer Abfolge kommen, wird durch das Lokalitätsprinzip festgelegt

Lokalitätsprinzip:

- Programme nutzen die Speicherinhalte oft mehrfach
- Dabei unterscheiden sich die Befehls- und die Datenlokalität
- Fallunterscheidung:
    - Zeitliche Lokalität: Oft wird kurz nach der Verwendung wegen z.B. Schleifen auf ein Element noch ein weiteres Mal zugegriffen
    - Räumliche Lokalität: Der PC zählt in der Regel um 1 hoch, daher macht es Sinn, Folgeadressen zu speichern