2.B: Von-Neumann-Architektur

Ist eine grundlegende CPU-Architektur, die nach John von Neumann benannt wurde und bis heute in vielen Computern genutzt wird. Die wesentlichen Komponenten dieser Architektur sind:

1. **Speicher**: Ein zentraler Speicher, der sowohl Programme (Befehle) als auch Daten enthält. Das Besondere an der Von-Neumann-Architektur ist, dass Programme und Daten im gleichen Speicher gespeichert werden. Dies bedeutet, dass sowohl die Anweisungen des Programms als auch die Daten, die das Programm verarbeitet, denselben Speicherbereich verwenden.
2. **Zentrale Recheneinheit (CPU)**: Die CPU besteht aus zwei Hauptkomponenten:
    - **Rechenwerk (ALU - Arithmetisch Logische Einheit)**: Hier werden die eigentlichen Berechnungen und logischen Operationen durchgeführt, wie zum Beispiel das Addieren von Zahlen oder das Vergleichen von Werten.
    - **Steuerwerk (Control Unit)**: Das Steuerwerk liest die Befehle aus dem Speicher, dekodiert sie und sorgt dafür, dass die notwendigen Operationen im Rechenwerk und anderen Teilen der CPU ausgeführt werden.
3. **Bus-System**: Es gibt einen gemeinsamen Bus, der für die Übertragung von Daten zwischen Speicher, CPU und Eingabe-/Ausgabegeräten (I/O) genutzt wird. Da der Bus sowohl für Daten als auch für Befehle verwendet wird, kann es zu Engpässen kommen, wenn beides gleichzeitig benötigt wird. Dies wird oft als Von-Neumann-Bottleneck bezeichnet.
4. **Eingabe- und Ausgabesysteme (I/O)**: Diese ermöglichen es dem Computer, mit der Aussenwelt zu kommunizieren. Typische Peripheriegeräte sind Tastatur, Maus, Festplatten, Drucker und Bildschirme. Der CPU-Bus verbindet die Eingabe-/Ausgabe-Einheiten mit dem Speicher und der CPU.

Funktionsweise:

Die CPU arbeitet nach einem festen Zyklus, bei dem sie Befehle aus dem Speicher lädt, dekodiert, ausführt und das Ergebnis dann entweder im Speicher oder in einem Register ablegt. Diese Abfolge wird als Von-Neumann-Zyklus bezeichnet und ist der Motor der CPU.

2.C: Unterschied zwischen Von-Neumann- und Harvard-Architektur

1. Speicherstruktur:

- **Von-Neumann-Architektur**: Es gibt einen einzigen Speicher, der sowohl die Daten als auch die Programme enthält. Die CPU greift über denselben Bus sowohl auf die Programmanweisungen als auch auf die zu verarbeitenden Daten zu. Dies führt zu einer effizienteren Nutzung des Speichers, kann jedoch zu Engpässen führen, da der Bus nur eine Operation zur Zeit (entweder Daten- oder Befehlszugriff) durchführen kann.
- **Harvard-Architektur**: Im Gegensatz dazu trennt die Harvard-Architektur den Speicher für Daten und Programme. Die CPU kann über zwei separate Busse gleichzeitig auf die Daten und die Befehle zugreifen, was die

Verarbeitungsgeschwindigkeit erheblich erhöhen kann. Dies wird in eingebetteten Systemen und DSPs (Digital Signal Processors) oft verwendet, wo Geschwindigkeit und Effizienz besonders wichtig sind.

2. Datenfluss:

- In der Von-Neumann-Architektur können Befehle und Daten nicht gleichzeitig geladen werden, da sie denselben Bus nutzen. Das führt zu einer sequentiellen Abarbeitung, bei der die CPU abwechselnd Befehle liest und Daten verarbeitet. Dies ist in vielen Fällen ausreichend, führt jedoch bei hochkomplexen oder datenintensiven Anwendungen zu Verzögerungen.
- In der Harvard-Architektur hingegen ermöglicht die Trennung von Daten- und Befehlsspeicher eine parallele Verarbeitung. Während ein Befehl verarbeitet wird, können gleichzeitig Daten geladen oder geschrieben werden. Das steigert die Effizienz, ist jedoch in der Implementierung aufwändiger und teurer.

3. Einsatzbereiche:

- **Von-Neumann-Architektur**: Diese Architektur ist nach wie vor die dominierende Architektur in Desktop-Computern, Laptops und Servern, da sie eine einfachere Implementierung ermöglicht und für allgemeine Rechenaufgaben effizient genug ist.
- **Harvard-Architektur**: Sie wird oft in eingebetteten Systemen, wie Mikrocontrollern oder Signalprozessoren verwendet. Diese Architektur wird überall dort eingesetzt, wo es auf Geschwindigkeit und Parallelität ankommt, wie z.B. bei Echtzeitanwendungen in der Automobilindustrie oder in Mobiltelefonen.

Beispiele:

- **Von-Neumann**: Die **Intel x86**-Architektur ist ein klassisches Beispiel für eine Von-Neumann-Architektur, die in PCs verwendet wird.
- **Harvard**: **ARM-Prozessoren**, die in den meisten modernen Smartphones zu finden sind, basieren auf der Harvard-Architektur.

2.D: Der Von-Neumann-Zyklus

Ist das Herzstück der CPU und beschreibt, wie die CPU Anweisungen ausführt. Die Schritte dieses Zyklus sind:

1. **Fetch**: Die CPU lädt den nächsten Befehl aus dem Speicher. Der Speicherort dieses Befehls wird durch den **Programmzähler (Program Counter)** bestimmt, der die Adresse des nächsten auszuführenden Befehls enthält.
2. **Decode**: Im nächsten Schritt wird der Befehl im **Steuerwerk (Control Unit)** dekodiert, das den Befehl in eine für die CPU verständliche Form umwandelt. Die CPU muss verstehen, ob es sich um eine Rechenoperation, eine Speicheroperation oder einen anderen Befehl handelt.
3. **Execute**: Nun führt die CPU den Befehl aus. Dies kann eine einfache mathematische Operation (z.B. Addition) sein oder eine Speicheroperation (z.B. das Laden von Daten in ein Register).

4. **Writeback**: Das Ergebnis der Operation wird entweder im Speicher oder in einem Register abgelegt, abhängig davon, was der Befehl fordert.

Was hält die CPU am Laufen?

Die CPU wird durch elektrische Signale und Taktraten gesteuert. Ein Taktgeber (Clock) sorgt dafür, dass die Befehle regelmässig und mit einer bestimmten Frequenz ausgeführt werden. Der eigentliche „Treibstoff“ der CPU ist also die Stromversorgung, die für den Betrieb aller internen Komponenten notwendig ist.

Wie wird der Ablauf gesteuert?

Die Reihenfolge der Befehle wird durch den Programmzähler gesteuert. Nach jedem Befehl wird der Programmzähler inkrementiert, um zum nächsten Befehl im Speicher zu gelangen. Die Steuereinheit (Control Unit) sorgt dafür, dass die richtige Reihenfolge der Operationen eingehalten wird und steuert die Ausführung der Anweisungen in der ALU.

2.E: Zugriff der CPU auf Peripheriegeräte

In der Von-Neumann-Architektur greift die CPU auf externe Geräte (Peripheriegeräte) über den Datenbus zu. Typische Peripheriegeräte sind Tastaturen, Mäuse, Festplatten, Netzwerkkarten und Bildschirme. Der Datenfluss zwischen der CPU und den Peripheriegeräten erfolgt durch spezielle Ein-/Ausgabebefehle (I/O-Befehle), die den Datenfluss zwischen dem Speicher und den Peripheriegeräten steuern.

Datentransfer:

Der Datentransfer zwischen der CPU und den Peripheriegeräten erfolgt über den gemeinsamen Bus. In modernen Systemen wird oft ein Interrupt-System verwendet. Wenn ein Peripheriegerät Daten senden oder empfangen möchte, sendet es einen Interrupt an die CPU, die daraufhin die aktuelle Operation unterbricht und die Datenübertragung initiiert. Dies ermöglicht eine effizientere Kommunikation zwischen CPU und Peripheriegeräten, ohne dass die CPU ständig auf Daten von den Peripheriegeräten warten muss.

2.F: CISC vs. RISC

Der Befehlssatz (Instruction Set) einer CPU definiert die Befehle, die der Prozessor ausführen kann. Dabei gibt es zwei Hauptarten von Befehlssätzen:

CISC (Complex Instruction Set Computing):

- **Komplexe Befehle**: CISC-Prozessoren wie die **Intel x86**-Architektur verfügen über viele komplexe Befehle, die oft mehrere Arbeitsschritte in einem einzigen Befehl kombinieren. Das macht die Programmierung einfacher, da weniger Befehle geschrieben werden müssen. Allerdings sind die Befehle in der Ausführung oft langsamer, da sie mehrere Zyklen benötigen.
- **Beispiel**: Ein CISC-Befehl könnte „Addiere den Inhalt zweier Speicherzellen und speichere das Ergebnis in einer dritten“ in einem einzigen Befehl ausführen.

RISC (Reduced Instruction Set Computing):

- **Einfachere Befehle**: RISC-Prozessoren wie **ARM** (in den meisten Smartphones) haben einen reduzierten Befehlssatz, der auf einfache, schnelle Befehle setzt. Jeder Befehl benötigt oft nur einen Taktzyklus zur Ausführung, was die Gesamtleistung verbessert, besonders bei wiederholten Operationen.
- **Beispiel**: Im Gegensatz zu CISC würden mehrere RISC-Befehle benötigt, um die gleiche Operation durchzuführen. Ein RISC-Prozessor könnte zunächst zwei Zahlen laden, sie addieren und das Ergebnis speichern – in drei separaten Befehlen.

Unterschiede:

- **CISC**: Komplexe Befehle, langsamere Ausführung, aber weniger Befehle nötig.
- **RISC**: Einfache Befehle, schnellere Ausführung, mehr Befehle nötig.