

# ENIAC, IAS, EDVAC und EDSAC: Die ersten Röhrenrechner in den USA und GB

Bastian Hofmann, Florens Werner  
Technische Universität München

04.05.2016

# Zusammenfassung

Dieses Dokument soll einen Überblick über die ersten elektronischen Rechner basierend auf Elektronenröhren geben, deren Funktionsweise erklären, sowie sie in historischen Kontext stellen. Überwiegend wird dabei konkret auf den ENIAC der amerikanischen University of Pennsylvania, seinen Nachfolger den EDVAC, die IAS-Maschine des amerikanischen Institute for Advanced Study und den EDSAC der britischen University of Cambridge eingegangen, welche alle maßgeblich von John von Neumann beeinflusst wurden.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>3</b>	5.2.7 Datenein- und Ausgabe . . . . .	11
<b>2 Funktionsweise der Elektronenröhre</b>	<b>3</b>	5.2.8 Erweiterungen . . . . .	12
<b>3 Geschichtliche Einteilung</b>	<b>3</b>	5.3 Programmierung des EDVAC . . . . .	12
<b>4 Der ENIAC</b>	<b>4</b>	5.4 Bezug und Relevanz zu heutigen Rechnern . . . . .	13
4.1 Einleitung . . . . .	4	6 Die IAS-Maschine	13
4.2 Technische Daten und Funktionsweise	4	6.1 Geschichte . . . . .	13
4.2.1 Grundlegendes . . . . .	4	6.2 Technische Beschreibung . . . . .	14
4.2.2 Die Einheiten des ENIAC . .	4	6.2.1 Zahlenrepräsentation . . . . .	14
4.2.3 Die <i>initiating unit</i> . . . . .	4	6.2.2 Speicherwerk . . . . .	14
4.2.4 Die <i>cycling unit</i> . . . . .	5	6.2.3 Rechenwerk . . . . .	15
4.2.5 Der Akkumulator . . . . .	5	6.2.4 Steuerwerk . . . . .	15
4.2.6 Der Multiplikator . . . . .	6	6.3 Besonderheiten . . . . .	15
4.2.7 Der Dividierer . . . . .	7	7 Der EDSAC	17
4.2.8 Die <i>master programmer unit</i> .	7	7.1 Geschichte . . . . .	17
4.3 Programmierung des ENIAC . . . .	8	7.2 Technische Beschreibung . . . . .	17
4.4 Bezug und Relevanz zu heutigen Rechnern . . . . .	9	7.2.1 Zahlenrepräsentation . . . . .	17
<b>5 Der EDVAC</b>	<b>9</b>	7.2.2 Ein-/Ausgabe . . . . .	17
5.1 Einleitung . . . . .	9	7.2.3 Instruktionssatz . . . . .	18
5.2 Technische Daten und Funktionsweise	10	7.2.4 Speicher . . . . .	19
5.2.1 Grundlegendes . . . . .	10	7.2.5 Order Sequence . . . . .	19
5.2.2 Die <i>central control unit</i> . . .	10	7.3 Programmierung und Besonderheiten .	19
5.2.3 Die <i>central arithmetical unit</i> .	10	7.3.1 Initial Orders . . . . .	19
5.2.4 Die Speichereinheit . . . . .	10	<b>8 Schluss</b>	<b>20</b>
5.2.5 Der <i>dispatcher</i> . . . . .	11		
5.2.6 Der <i>timer</i> . . . . .	11		

# 1 Einleitung

Die Geschichte des maschinellen automatisierten Rechnens reicht von mechanischen Computern, wie der Analytical Engine von Charles Babbage Anfang des 19. Jahrhunderts, bis hin zu *very large-scale integrated circuits*, die heutzutage eingesetzt werden und auf Transistoren basieren. Ein wichtiger technologischer Zwischenschritt auf diesem Weg war der Sprung von mechanischen beziehungsweise elektromagnetischen Rechnern zu Röhrenrechnern, das heißt: automatisierte elektronische Computer, deren grundlegende Komponente die Elektronenröhre ist. Ein Vorteil der Elektronenröhre gegenüber mechanischen und elektromagnetischen Komponenten war, dass sie eine deutlich kürzere Reaktionszeit hatte und somit schnellere Operationen umgesetzt werden konnten. Vier der wichtigsten Computer, die darauf basieren sind die drei amerikanischen ENIAC, EDVAC und IAS-Maschine, sowie der britische EDSAC. Anschließend an eine Beschreibung der Funktionsweise der Elektronenröhre im Allgemeinen wird geklärt, wie diese vier Rechner im Detail aufgebaut waren und welche Funktionen sie bereit stellten.

## 2 Funktionsweise der Elektronenröhre

Ein sehr einfacher Typ der Elektronenröhre ist die Diode. Sie besteht im Wesentlichen aus drei Komponenten: einer (beheizten) negativ geladenen Kathode, einer positiv geladenen Anode und einem Gehäuse, in dem Vakuum herrscht. Wird nun zwischen der Kathode und Anode elektrische Spannung angelegt, findet ein Elektronenstrom von der Kathode hin zur Anode statt. Bei einer Triode wird zusätzlich eine weitere Elektrode verbaut: ein Gitter zwischen Kathode und Anode, das durch Anlegen einer geringen Spannung den Elektronenfluss verstärken oder dämpfen kann. Dadurch erhält man einen Verstärker, der das Ausgangssignal im Verhältnis zum Eingangssignal vergrößern kann. Dieser Effekt kann in der elektronischen Informationstechnologie sehr gut genutzt werden, da man dadurch ein Instrument erhält, mit dem zwei verschiedene Zustände (Elektronen fließen

oder nicht beziehungsweise Spannung liegt an oder nicht) zuverlässig voneinander unterschieden werden können: Man erhält aus Informationssicht also ein Bit. Dieser Sachverhalt bildet die Grundlage für die im folgenden vorgestellten Röhrenrechner.

## 3 Geschichtliche Einteilung

Bevor die konkreten Röhrenrechner im Detail vorgestellt werden, soll im Folgenden deren historischer Kontext näher erläutert werden. Allgemein ist festzustellen, dass sowohl ENIAC, EDVAC, EDSAC als auch die IAS-Maschine gegen Ende des zweiten Weltkriegs beziehungsweise kurz nach dem zweiten Weltkrieg in Auftrag gegeben oder fertiggestellt wurden und jeweils etwa 10 Jahre in Betrieb waren. Im Falle des ENIAC bestand auch ein thematischer Zusammenhang zum zweiten Weltkrieg: Die Motivation seiner Entwicklung war militärisch begründet, er sollte primär zur Berechnung ballistischer Flugbahnen von Artilleriegeschützen, Granaten et cetera dienen, was zuvor per Hand berechnet wurde. Beim EDVAC, EDSAC und der IAS-Maschine, die zeitlich leicht nach dem ENIAC und auch nach dem zweiten Weltkrieg entworfen und von seinem Konzept beeinflusst worden sind, hatte man bereits den Nutzen elektronischer Rechner für allgemeinere Anwendungen erfasst. Den EDVAC kann man als Nachfolger des ENIAC ansehen, die primäre Motivation seiner Erstellung war, einen besseren ENIAC zu bauen. Er war dementsprechend wesentlich kompakter, bezüglich des logischen Aufbaus deutlich strukturierter und aus Sicht der Bedienung erheblich einfacher zu programmieren. Die IAS-Maschine wurde etwa zeitgleich zum EDVAC entwickelt und maßgeblich von John von Neumann beeinflusst. Daraus resultiert, dass sie von vielen als der erste Rechner mit Von-Neumann-Architektur bezeichnet wird, da sie mit die erste Maschine ist, bei der das Programm und Daten in einem gemeinsamen Speicher gehalten wurden. Während diese ersten drei Elektronenrechner aus den USA stammen, ist der EDSAC ein Konkurrenzprodukt aus dem Vereinigten Königreich. Entwickelt wurde er von der Cambridge-Universität und wurde von John von Neumanns "First Draft of a Report on the EDVAC"

inspiriert.

## 4 Der ENIAC

### 4.1 Einleitung

Wie eingangs erwähnt war der primäre Einsatzzweck des **Electronic Numerical Integrator And Computer**, ballistische Berechnungen für das US-amerikanische Militär ausführen zu können. Dieses gab die Entwicklung des ENIAC 1943 in Auftrag und zwei Jahre später wurde er fertiggestellt, wobei die öffentliche Bekanntgabe und Inbetriebnahme erst im Jahr 1946 statt fand. Auftragnehmer war die Moore School of Electrical Engineering, ein Teil der University of Pennsylvania. Dort war der ENIAC zunächst auch aufgebaut und in Betrieb, er wurde jedoch 1947 auf eine Militärbasis in Maryland verlagert, wo er bis zu seiner Abschaltung 1955 in Betrieb war. Bemerkenswert ist, dass die Entwicklungskosten im Vorfeld vertraglich auf 61.700 US-Dollar festgelegt wurden, tatsächlich aber Kosten in Höhe von etwa 487.000 US-Dollar entstanden. [6] [4]

### 4.2 Technische Daten und Funktionsweise

#### 4.2.1 Grundlegendes

Die primäre Komponente in jedem der in dieser Arbeit vorgestellten Rechner ist die Elektronenröhre. Der ENIAC besaß von diesen etwa 17.500 Stück. Weiterhin wurden 7.200 Dioden, 1.500 Relais 70.000 Widerstände sowie 10.000 Kondensatoren verbaut. Diese hohe Anzahl an Komponenten beeinflusste auch stark das Gewicht, die Größe und den Energiebedarf des ENIAC: Er wog 27 Tonnen, nahm 167 Quadratmeter an Platz ein (also in etwa so groß wie die Spielfläche eines Volleyball-Felds) und hatte eine Leistungsaufnahme von 174 Kilowatt (also in etwa so hoch wie ein PKW-Motor mit 240 PS). Weiterhin konnte er dadurch 5000 Additionen, 385 Multiplikationen, 40 Divisionen sowie 3 Wurzelberechnungen pro Sekunde ausführen. Anders ausgedrückt: Eine Addition dauerte 0,2 Millisekunden, eine Multiplikation bis zu 2,8 ms, eine Division 24 ms und Wurzelziehen etwa 333

ms. Außerdem konnten Zahlenwerte in sogenannten Akkumulatoren gespeichert werden, wobei anzumerken ist, dass hierfür (und auch für alle Berechnungen) das Dezimalsystem verwendet wurde. [3]

Bezogen auf den logischen Aufbau war der ENIAC grundlegend in 40 *Panelen* (engl.: panels) aufgeteilt, die U-förmig an drei Wänden des Raums, in dem er stand, in Reihe aufgestellt waren. Diese 40 Panelen waren in 30 logisch zusammen gehörende *Einheiten* (engl.: units) aufgeteilt. So gab es eine *initiating unit* bei der Einstellungen vorgenommen werden konnten, die sich auf den initialen Start des Systems beziehen sowie eine *cycling unit*, die für die Synchronisation des Systems verantwortlich war und effektiv einen Takt vorgab. Beide stellten sowohl eine Einheit dar und waren jeweils auf einem Panel untergebracht. Im Gegensatz dazu bestand die Einheit, die für die Multiplikation zuständig war, aus drei Panelen. Außerdem gab es eine *master programmer unit* bestehend aus zwei Panelen, durch die unter anderem Schleifen implementiert werden konnten, sowie eine Divisions- beziehungsweise Quadratwurzeleinheit (ein Panel), einer Einheit um Eingaben per Lochkarten einzulesen (drei Panelen), einer Einheit um errechnete Werte auszugeben (drei Panelen), drei Einheiten um im Vorfeld definierte Funktionstabellen einzulesen zu können (jeweils zwei Panelen) Und das Herzstück des ENIAC: 20 Akkumulatoreinheiten (jeweils ein Panel), die als Speicher dienten, sowie die für alle Berechnungen zu Grunde liegende Addition ausführte. Ein schematischer Aufbau des ENIAC ist in Abbildung 1 zu sehen. Im Folgenden werden einige der wichtigsten Einheiten näher erläutert. [3]

#### 4.2.2 Die Einheiten des ENIAC

#### 4.2.3 Die *initiating unit*

Die *initiating unit* bot grundsätzliche Funktionen für die Initialisierung des Systems. Einerseits befand sich auf ihr der Startknopf, mit dem der ENIAC (genauer: ein Programmablauf auf dem ENIAC) gestartet werden konnte, andererseits war es möglich, ein spezielles Programm aufzurufen, das den ENIAC in einen definierten Zustand brachte, das heißt alle Transistoren beziehungsweise Flip-Flops auf einen gültigen

Zustand setzte. Dies war notwendig, da beim ersten Einschalten des ENIAC die Flip-Flops zufällige Werte annahmen. [3]

#### 4.2.4 Die *cycling unit*

Die *cycling unit* war primär für die Synchronisierung des ENIAC zuständig. Dazu sendete sie jede  $\frac{1}{5000}$  Sekunde einen bestimmten Puls, den CPP (*central programming pulse*) an die anderen Einheiten des ENIAC. Dieser markierte den Beginn und das Ende eines Zyklus. Wenn eine Einheit einen CPP erhielt, begann sie damit, ihre programmierte Operation auszuführen und gab am Ende ihrer Ausführung (die sich über mehrere Zyklen strecken konnte) den nächsten CPP der *cycling unit* weiter an die folgende Einheit, die dann wiederum mit ihrer Operation begann. Die Einheiten des ENIAC waren derart konzipiert, dass sie für ihre Operationen ein Vielfaches dieses Zyklus brauchten. Ein Akkumulator brauchte beispielsweise genau einen Zyklus, also  $\frac{1}{5000}$  Sekunde, für eine Addition, weshalb der Zyklus auch geläufig Additionszyklus (*addition cycle*) genannt wird. Die Multiplikationseinheit brauchte beispielsweise etwa  $p + 4$  (Additions-) Zyklen bei einem  $p$ -stelligen Multiplikator. Die Ausführungszeiten weiterer Operationen ist in Tabelle 1 zu sehen. Weiterhin bot die *cycling unit* sozusagen Debugging-Modi an: wurde der entsprechende Schalter betätigt, ging der ENIAC in den sogenannten "one addition time"-Operationsmodus über, in dem die Maschine nur auf Kommando genau einen Zyklus, also  $\frac{1}{5000}$  Sekunde beziehungsweise eine Addition, weiter rechnete und ansonsten anhielt. Der zweite Operationsmodus "one pulse time" arbeitete ähnlich, nur dass hierbei der ENIAC auf Kommando Impulssignal für Impulssignal weiter rechnete, womit eine sehr genau Analyse des Verhaltens des ENIAC möglich war." [3]

#### 4.2.5 Der Akkumulator

In einem *accumulator* konnte eine vorzeichenbehaftete 10-stellige Dezimalzahl gespeichert werden. Hierfür wurden sogenannte Zähler (engl.: counter) entwickelt, indem zehn Flip-Flops derart aneinander geschaltet wurden, dass das zehnte Flip-Flop wieder mit dem

Tabelle 1: Operationen des ENIAC und ihre Laufzeiten [3]

Einheit	Operation	Laufzeit in Zyklen
Akkumulator	Zahl erhalten und auf Akkumulator addieren	1
	Zahl im Akkumulator (oder das Komplement) $r$ Mal auf Datenleitung schicken; $1 \leq r \leq 9$	$r$
Multiplikator	Vorzeichenhaftes Produkt aus 10-stelligem Multiplikand und $p$ -stelligem Multiplikator	$p + 4$
Wurzel- und Divisionseinheit	findet $p$ Stellen des Quotienten mit Vorzeichen	ca. $13*(p+1)$
	findet das Zweifache der Quadratwurzel	ca. $13*(p+1)$
Funktions-tabelle	gibt Funktionswert von 2-stelligem Parameter oder 4 beliebige benachbarte Werte $r$ Mal aus	$r + 4$
Konstanten-übermittler	gibt eine vorzeichenbehaftete 5- oder 10-stellige Zahl aus	1

ersten verbunden war, man spricht daher auch von einem Ringzähler. Ein Flip-Flop wurde mit Hilfe von zwei Trioden umgesetzt, hatte - wie auch heute üblich - zwei Eingangs- und zwei Ausgangssignale und konnte dadurch einen Zustand speichern. Die beiden Zustände wurden beim Ringzähler des ENIAC als *abnormal* und *normal* bezeichnet, wobei *abnormal* den Zustand bezeichnet, der heute als *an*, *aktiv*, *true* oder *1* interpretiert werden würde. Der Ringzähler hatte drei wichtige Eigenschaften: Erstens konnte und musste genau ein Flip-Flop in *abnormalem* Zustand sein während die anderen neun in *normalem* Zustand waren. Zweitens konnte der Zähler zurückgesetzt werden, das heißt es gab eine definierte Grundeinstellung, in der ein bestimmtes Flip-Flop *abnormal* war während die übrigen *normal* waren. Drittens wies der Ringzähler folgendes Verhalten auf: ging am Eingang ein Impulssignal ein, so erhielten dies alle Flip-Flops. War ein Flip-Flop in *normalem* Zustand, so blieb es darin. War es in *abnormalem* Zustand, so ging es in den *normalen* Zustand über und gab außerdem ein Signal an seinen Nachfolger weiter, sodass dieser vom *normalen* in den *abnormalen* Zustand wechselte. So wurden die Flip-Flops des Zählers also mit nacheinander eingehenden Signalen durch iteriert. Auf diese Weise war es möglich, den Ringzähler als Ziffer einer Dezimalzahl zu interpretieren: Die einzelnen Flip-Flops standen der Reihe nach für die Werte 0 bis 9 und ein Eingangssignal am Zähler bedeutete ein Inkrementieren der Ziffer um den Wert 1, wobei die definierte Grundeinstellung 0 darstellte. Damit ein Akkumulator nun eine 10-stellige Dezimalzahl speichern konnte, wurden darin zehn Ringzähler aneinander geschaltet, wobei bei einem Überlauf einer Ziffer ein (als um den Wert 1 inkrementierend interpretiertes) Signal an die nächsthöherwertige Ziffer gesendet wurde. Zusätzlich gab es ein Flip-Flop, das das Vorzeichen der Zahl angab. Weiterhin konnte der Akkumulator seine gespeicherte Zahl ausgeben, beispielsweise an einen weiteren Akkumulator oder die Einheit, die für die Ausgabe zuständig war. Die Weitergabe von Zahlen an eine andere Einheit gestaltete sich recht simpel: Für jede der zehn Ziffern einer 10-stelligen Dezimalzahl gab es jeweils eine Datenleitung zwischen den beiden betreffenden Einheiten, also insgesamt zehn. Wenn der Akkumulator nun dar-

auf programmiert war, seine gespeicherte Zahl weiter zu geben, legte er so viele Impulssignale an jeder Datenleitung an, wie sie dem Wert der zugehörigen Ziffer entsprachen. Sollte also beispielsweise die Zahl 425 übergeben werden, so wurden auf Leitung 1 fünf Impulse, auf Leitung 2 zwei Impulse, auf Leitung 3 vier Impulse und auf den restlichen Leitungen keine Impulse gesendet. Die Additionsfunktion des Akkumulators baut auf simple Weise auf diesem Konzept auf. Erhielt er von einer anderen Einheit eine Zahl, so wurden die Impulse jeder einzelnen Datenleitung an den jeweiligen Ringzähler übergeben, der für die jeweilige Ziffer stand, der folglich entsprechend inkrementiert wurde. Dem obigen Beispiel entsprechend kamen also fünf Impulssignale an dem Ringzähler an, der die Einerstelle repräsentiert, und er wurde fünf Mal hintereinander um den Wert 1 inkrementiert. Die zuvor im Akkumulator gespeichert war, wurde somit nicht überschrieben, sondern weiter gezählt. Der Akkumulator hatte fünf separate Dateneingänge (mit jeweils zehn Ziffernleitungen und einer Vorzeichenleitung) sowie zwei Ausgänge: einen A-Ausgang für die (positive) in ihm gespeicherte Zahl und einen S-Ausgang für das Zweierkomplement dieser Zahl. Wurde das Zweierkomplement an einen Akkumulator weitergegeben, so konnte er durch Addition die Differenz der in ihm gespeicherten Zahl und der übergebenen Zahl berechnen. [3]

#### 4.2.6 Der Multiplikator

Der *multiplier*, bestehend aus drei Panelen, war fest mit vier Akkumulatoren verbunden. In zwei befanden sich jeweils der Multiplikand und Multiplikator, in den anderen beiden wurde das Produkt dieser beiden, also das Ergebnis, gespeichert. Pro Zyklus berechnete die Multiplikationseinheit die (Teil-) Produkte von einer Ziffer des Multiplikators mit jeweils allen Ziffern des Multiplikanden. Das zweistellige Ergebnis wurde aufgeteilt: Der Wert der Einerstelle wird auf den ersten Ergebnissakkumulator addiert, der Wert der Zehnerstelle auf den zweiten. Dies wird so oft wiederholt bis das Produkt vollständig berechnet ist, wobei bei jedem Folgeschritt vor dem Addieren auf die Ergebnissakkumatoren das Ergebnis eines Teilprodukts entsprechend geshiftet wird. Schlussendlich

wird das Endergebnis berechnet, indem beide Ergebnisakkumulatoren aufaddiert werden. [3]

#### 4.2.7 Der Dividierer

Für das Dividieren werden ebenfalls vier Akkumulatoren benötigt: in einem steht der Zähler, in einem der Nenner, in einem das Ergebnis und ein Akkumulator wird benötigt um den Rest beim ganzzahligen Teilen der *divider unit* zu shiften. Dabei wird iterativ der Nenner mit dem Zähler verrechnet. Falls beide das gleiche Vorzeichen haben, wird der Nenner vom Zähler abgezogen, falls sie unterschiedliche Vorzeichen haben, werden sie addiert. Dies geschieht so oft, bis die Null übertritten wird und die Anzahl der benötigten Schritte werden an die entsprechende Stelle des Ergebnisakkumulators übergeben. Anschließend wird der verbleibende Rest der Division im dafür zuständigen Akkumulator nach links geschiftet, in den Zähler-Akkumulator geschrieben und die Operation wird mit dem neuen Zähler wiederholt. [3]

#### 4.2.8 Die *master programmer unit*

Eine weitere wichtige Einheit war die *master programmer unit*, mit der die Programmierung von Schleifen möglich war. Ihre wichtigste Komponente war der sogenannte *stepper counter*, ein Zähler wie der Dekadenzähler in einem Akkumulator, allerdings mit sechs statt zehn Stufen. Dieser Zähler war jedoch mit bis zu fünf Dekadenzählern verbunden, die ähnlich wie bei der Akkumulatoreinheit zu einer 5-stelligen Dezimalzahl verbunden wurden. Die *master programmer unit* hatte insgesamt zehn dieser Kombinationen aus *stepper counter* und Dekadenzähler und jede davon hatte jeweils ein *input terminal* sowie jeweils sechs *output terminals*. Kam am Eingang nun ein Signal an, so wurden die Dekadenzähler um den Wert 1 inkrementiert und ein Signal wurde aus einem der sechs *output terminals* gesendet, je nachdem in welcher der sechs Stufen der *stepper counter* war. Jeder dieser Stufen war eine Menge von Dekadenschaltern zugeordnet, die im Vorfeld auf einen gewünschten Wert eingestellt werden konnten. Eine Schleife konnte nun so umgesetzt werden: Kamen am Eingang des *stepper counters* nacheinander so viele Signale an,

wie es dem Wert des entsprechenden Dekadenschalters entsprach, so wurden entsprechend viele Ausgangssignale auf dem zugehörigen *output terminal* gesendet (die angeschlossen an eine andere Einheit jeweils eine Operation in Gang setzen konnten) und anschließend ging der *stepper counter* in die nächste Stufe über und die Dekadenzähler wurden zurückgesetzt. Sollten zum Beispiel erst 20 Additionen und anschließend 15 Subtraktionen auf den beiden gleichen Akkumulatoren durchgeführt werden (wobei Akkumulator 1 den ersten Summanden an Akkumulator 2 weitergibt, der diesen auf seinen gespeicherten Wert addiert), so wurden im Vorfeld des Programmablaufs die Dekadenschalter für die erste Stufe des *stepper counter* auf den Wert 20 und die Dekadenschalter für die zweite Stufe auf den Wert 15 eingestellt. Das *input terminal* des *stepper counter* empfing den *program output pulse* von Akkumulator 2, das *output terminal* von Stufe eins war so verbunden, dass Akkumulator 1 bei einem Signal seinen A-Ausgang an Akkumulator 2 weitergab und eine Addition in Akkumulator 2 anstieß und das *output terminal* von Stufe 2 war analog so verbunden, dass bei Akkumulator 1 der S-Ausgang weiter gegeben wurde und Akkumulator 2 entsprechend das Zweierkomplement addierte. Zunächst befand sich der *stepper counter* in Stufe 1, sodass bei den ersten zwanzig eingehenden Impulssignalen die Dekadenzähler schrittweise auf den Wert 20 inkrementiert wurden und jeweils eine Addition angestoßen wurde. Anschließend entsprach der Wert in den Dekadenzählern dem Wert, der durch die Dekadenschalter eingestellt worden war, also ging der *stepper counter* in Stufe 2 über und die Dekadenzähler wurden auf den Wert 0 zurück gesetzt. Die 15 nächsten eingehenden Impulssignale bewirkten im Unterschied dazu, dass Akkumulator 1 seine am S-Ausgang anliegende Zahl, also das Zweierkomplement, an Akkumulator 2 weitergab, welcher wiederum 15 Additionen durchführte. Außerdem wurde der Wert in den Dekadenzählern schrittweise auf 15 inkrementiert. Der Wert der Dekadenschalter, die Stufe 2 zugeordnet waren, entsprach nun dem in den Dekadenzählern gespeicherten Wert, sodass der *stepper counter* in Stufe 3 übergang und die Dekadenzähler auf 0 zurück gesetzt wurden und die Schleifen beendet waren. Eine parallele Berechnung war auch möglich, da es wie

erwähnt insgesamt zehn dieser Kombinationen aus *stepper counter* und Dekadenzähler gab, deren Eingänge und Ausgänge mit wenigen Einschränkungen beliebig mit den anderen Einheiten des ENIAC verbunden werden konnten. [3]

### 4.3 Programmierung des ENIAC

Die Programmierung des ENIAC gestaltete sich aufgrund des physischen und logischen Aufbaus des ENIAC schwierig. Aus logischer Sicht musste jedes komplexere mathematische Problem, das man mit Hilfe des ENIAC lösen wollte, auf einfache Berechnungen zurück geführt werden, da dieser nur addieren, subtrahieren, multiplizieren, dividieren und wurzelziehen konnte. Es gab keine voreingestellten Routinen oder Programme, die man nutzen konnte um zum Beispiel Fakultät oder eine Potenz auszurechnen. Zusätzlich musste logisch auf die Limitierungen des ENIAC eingegangen werden. Mit nur 20 Akkumulatoren war zum Beispiel der Speicher sehr begrenzt, es konnten lediglich 20 10-stellige Dezimalzahlen gespeichert werden. Dies konnte umgangen werden, indem man zwei Akkumulatoren auf eine bestimmte Weise zusammen schalten konnte, sodass die Speicherung einer 20-stelligen Dezimalzahl möglich war oder man musste es berücksichtigen, indem man Berechnungen separat auf verschiedenen Akkumulatoren ausführte und die Ergebnisse im Anschluss entsprechend interpretierte. Auch möglich war, berechnete Werte auf Lochkarten zu stanzen, um sie später wieder für weitere Berechnungen einzulesen. Als Konsequenz waren auch die Berechnungen entsprechend limitiert, da sie alle auf die Akkumulatoreinheit zurück greifen. Diese Einschränkungen mussten vor dem eigentlichen Programmieren bedacht werden. [3]

Die zweite, wesentlich größere Schwierigkeit ergab sich aus dem physischen Aufbau und betraf hauptsächlich das Umsetzen eines konzeptionellen Programmablaufs in einen konkreten physischen. Beim ENIAC gab es keine Software, also auch keine Programme in Form von maschinenfähigem Code. Ein Programm bestand im Wesentlichen aus den Steckverbindungen zwischen den verschiedenen

Panelen beziehungsweise Einheiten und den Einstellungen der Schalter an den Panelen (siehe Abbildung 2). In dieser Hinsicht war der ENIAC nahezu modular aufgebaut: Fast jeder Akkumulator konnte mit fast jedem anderen Akkumulator verbunden werden (wenige Ausnahmen waren beispielsweise die Akkumulatoren der Multiplikationseinheit), die *output terminals* der *master programming unit* konnten nahezu willkürlich mit anderen Einheiten verbunden werden um Schleifen umzusetzen und auch die Einheit, die für das Drucken, also das Stanzen der Lochkarten, verantwortlich war, konnte ihre Eingabedaten durch eine Steckerverbindung von nahezu jedem Akkumulator erhalten. Diese Modularität hatte einerseits den Vorteil, dass allgemeine Berechnungen möglich waren und der ENIAC an die mathematischen Probleme angepasst werden konnte, andererseits bedeutete das auch, dass die Verbindungen für jede Änderung des Programms auch neu gesteckt werden mussten. Dadurch mussten Änderungen am Programmablauf nicht nur aus Sicht der Programmlogik gut vorbereitet werden, sondern auch aus Sicht der physikalischen Steckerverbindungen und Schalttereinstellungen. Zusätzlich zur theoretischen Vorbereitung benötigte auch das tatsächliche Stecken der Verbindungen nicht wenig Zeit, auch weil man jede Verbindung kontrollieren musste. Diese Eigenschaften führten dazu, dass ein Programmieren des ENIAC zwar möglich und auch mächtig war, sich aber sehr umständlich und zeitaufwendig gestaltete. [3]

Um Daten eingeben zu können bediente man sich des Konstantenübermittlers (*constant transmitter*), welcher mit einem Lochkartenlesegerät von IBM verbunden war. Dieses las Lochkarten mit bis zu 80 Ziffern ein, welche anschließend mit Hilfe von Relais im Konstantenübermittler gespeichert wurden und durch elektrische Signale den anderen Einheiten übergeben werden konnten. Als Ausgabe dienten ebenfalls Lochkarten, die von einem Lochkartenstanzer von IBM erstellt wurden, der mit dem Konstantenübermittler verbunden war. [3]

Ein wichtiger Teil des Programmierens ist bekanntlich die Fehleranalyse beziehungsweise das Debug-

ging, weswegen der ENIAC dem Programmierer auch in dieser Hinsicht Hilfsmittel bot. Zum Einen gab es, wie in Kapitel 4.2.4 erwähnt, die Möglichkeit, in den "one addition time"-Operationsmodus oder den "one pulse time"-Operationsmodus zu wechseln, indem entsprechende Schalter an der *cycling unit* betätigt wurden. Dadurch war es möglich, den ENIAC zu jedem Zeitpunkt anzuhalten und auf Knopfdruck einen Impuls beziehungsweise einen Zyklus weiter rechnen zu lassen, ähnlich wie Breakpoints in heutigen IDEs für höhere Programmiersprachen. Des Weiteren befand sich für jedes Flip-Flop eine von außen sichtbare Neonlampe am ENIAC, die so geschaltet waren, dass sie leuchteten, wenn das entsprechende Flip-Flop in *abnormalem* Zustand war. Im Zusammenspiel mit dem schrittweisen Weiterschalten des ENIAC konnte man so zu jedem Zeitpunkt detailliert feststellen, in welchem Zustand sich jede Komponente befand und eventuelle Fehler analysieren." [3]

#### 4.4 Bezug und Relevanz zu heutigen Rechnern

Der ENIAC hatte zweifelsfrei viel Einfluss auf die Entwicklung von Rechnerarchitekturen während und nach seiner Zeit (wie zum Beispiel auf die anderen in dieser Arbeit behandelten Rechner) und bildete somit auch gewissermaßen das Fundament für heutige moderne Recheneinheiten. Andererseits wurden einige Konzepte, die er einföhrte, auch schnell wieder verworfen, beziehungsweise weiter entwickelt. So ist das Konzept, als Grundelement eine Komponente zu verwenden, die binäre oder binär interpretierte Signale aussendet, erhalten geblieben, allerdings wurden die Elektronenröhren mit der Zeit durch Transistoren ersetzt, da diese günstiger herzustellen, kleiner und weniger fehleranfällig sind. Die allgemeine Programmierbarkeit eines Rechners wird heute auch geschätzt, allerdings findet diese im Normalfall nicht mehr wie beim ENIAC durch Einstellung der Hardware statt, sondern wird durch Programmierschnittstellen wie Assembler beziehungsweise durch im Vorfeld bereitgestellte Routinen gewährleistet. Auch das Dezimalsystem wurde sehr bald durch das Binärsystem ersetzt und die Modularität der Hardware konnte sich in dieser komplexen Form nicht halten be-

ziehungsweise wird heutzutage gegebenenfalls durch standardisierte Schnittstellen umgesetzt. Im Allgemeinen ist der logische Aufbau in die vielen verschiedenen Einheiten heutzutage deutlich vereinfachter, es gibt eine eindeutigere Trennung der Funktionen von Komponenten als beim ENIAC.

## 5 Der EDVAC

### 5.1 Einleitung

Wie in Kapitel 3 erwähnt, kann der **Electronic Discrete Variable Automatic Computer** als Nachfolger des ENIAC gesehen werden. Seine Entwicklung wurde direkt im Anschluss an die Fertigstellung des ENIAC 1946 in Auftrag gegeben und nach drei Jahren wurde er 1949 fertiggestellt. Abermals waren Auftraggeber das US-Militär und Auftragnehmer die Moore School of Electrical Engineering, Teil der University of Pennsylvania. Die veranschlagten Kosten von etwa 100.000 US-Dollar wurden mit ungefähr 467.000 US-Dollar tatsächlich entstanden. Die Kosten erneut erheblich übertroffen. Im Gegensatz zum ENIAC stand er während seiner gesamten Lebenszeit von 1951 bis 1961 auf der Militärbasis in Maryland. Die primäre Motivation war, die komplizierte Programmierung des ENIAC zu vereinfachen und von Anfang an die Fähigkeit, allgemeine Berechnungen durchzuführen, stärker ins Konzept einzubinden, im Gegensatz zum ENIAC, bei dem das Augenmerk auf dem militärischen Einsatzzweck, den ballistischen Berechnungen, lag. Bereits während der Entwicklung des ENIAC wurde John von Neumann, der damals am Manhattan-Projekt in Los Alamos beteiligt war, auf die Projekte der Moore School of Electrical Engineering aufmerksam und entwarf ein Konzept für den EDVAC, das bereits einen gemeinsamen Speicher für Daten und Programm vorsah, was später unter dem Begriff Von-Neumann-Architektur bekannt wurde. [9]

## 5.2 Technische Daten und Funktionsweise

### 5.2.1 Grundlegendes

Im Vergleich zum ENIAC besaß der EDVAC mit etwa 6.000 Stück schon deutlich weniger Elektronenröhren. Auch hatte er lediglich 180 Relais, 26.000 Widerstände, 6.000 Kondensatoren, wog 7,8 Tonnen, hatte eine Leistung von 56 Kilowatt und nahm etwa 40 Quadratmeter an Platz ein, allerdings beziehen sich diese Angaben auf verschiedene Zeitpunkte zwischen 1949 und 1961, während denen der EDVAC teilweise stark modifiziert wurde. Dennoch zeigen die Größenverhältnisse im Vergleich zum ENIAC, dass der EDVAC deutlich kompakter und weniger komplex aufgebaut war. Dabei verlor er aber etwas an Rechenleistung: Er konnte durchschnittlich nur 1.157 Additionen sowie Subtraktionen und 347 Multiplikationen pro Sekunde ausführen, dafür stieg die Anzahl der Divisionen pro Sekunde deutlich auf 341 (vergleiche Kapitel 4.2.1). Der EDVAC rechnete bereits im Binärsystem mit 44-Bit-Wörtern, deren Struktur aufgeteilt war in Operationscode, Operanden, Zieladresse und Adresse der nächsten Instruktion. [7]

Der logische Aufbau des EDVAC wurde im Unterschied zum ENIAC grundlegend vereinfacht. Es gab keine 40 Paneele mit 30 Einheiten mehr, sondern die Grundfunktionen wurde auf sechs Komponenten aufgeteilt, die sich wiederum in zwölf Gehäusen befanden, die eine Frontfläche von etwa 76 cm auf 220 cm mit variierender Tiefe hatten (siehe Abbildung 3): Die *central control unit* bot (zusammen mit dem Lochbandstreifendrucker und -leser) die einzige Schnittstelle zwischen Mensch und Maschine. Die (doppelt vorhandene) *central arithmetical unit* führte alle Berechnungen durch und sendete die Ergebnisse an den *Speicher*, der in zwei Einheiten aufgeteilt war. Der *dispatcher* dekodiert Befehle von der *central control unit* oder der Speichereinheit und sendet Kontrollsignale an andere Einheiten und der *timer* war für das Taktsignal und die Synchronisierung verantwortlich. Im Folgenden werden diese Einheiten genauer erläutert. [4]

### 5.2.2 Die *central control unit*

An der *central control unit* befanden sich primär die (Dreh-) Schalter, mit denen verschiedene Parameter eingestellt und Operationen angestoßen werden konnten. So wurde dort beispielsweise eingestellt ob beide Speichereinheiten verwendet werden sollten oder nur eine, was zur Folge hatte, dass auch nur eine oder beide arithmetische Einheiten aktiv waren. Auch konnte man sich entscheiden, was bei einem Überlauf geschehen sollte: ihn ignorieren, anhalten oder zu einer bestimmten voreingestellten Adresse springen. Außerdem war es möglich, manuell ein 44-Bit-Wort einzugeben und dies dann über die Operation MR (manual read) einzulesen. Außerdem befanden sich an der *central control unit* Neonlampen, die Rückmeldung über den Zustand der Maschine gaben. 10 Neonlampen waren zum Beispiel oben zu sehen, die angaben was im *initial address register*, was heute dem Programmzähler entspricht, stand. Zusätzlich waren 44 Neonlampen angebracht, mit denen man sich den Inhalt eines beliebigen Wortes im Speicher anzeigen lassen konnte. [9]

### 5.2.3 Die *central arithmetical unit*

In der *central arithmetical unit* wurden alle Berechnungen durchgeführt, wobei die notwendigen Daten von der Speichereinheit geholt und auch wieder dort abgespeichert wurden. Eine Besonderheit war, dass sie doppelt vorhanden war, da man befürchtete, dass man aufgrund der hohen Geschwindigkeit und der daraus resultierenden hohen Anzahl an Berechnungen einen Fehler nicht zuverlässig erkennen konnte. Daher ließ man alle Berechnungen parallel auf einer zweiten arithmetischen Einheit laufen und überprüfte an fünf bestimmten Berechnungsschritten die Zwischenergebnisse auf Gleichheit. [9]

### 5.2.4 Die Speichereinheit

Die Speichereinheit war in zwei gleich große Komponenten aufgeteilt, die jeweils 64 akustische Verzögerungsleitungen zu wiederum je acht 44-Bit-Wörter hatten, insgesamt konnten also 1024 Wörter gespeichert werden. Bei diesen akustischen Verzögerungsleitungen wurde das elektrische Signal in Ultraschall

umgewandelt und anschließend durch ein Medium mit definierter Schallgeschwindigkeit (im Falle des EDVAC Quecksilber) geleitet und danach wieder in ein elektrisches Signal umgewandelt. An dieser Stelle konnte das Signal wieder abgegriffen und weitergeleitet werden oder wurde wieder in das Medium geführt. Zusätzlich zum Hauptspeicher gab es noch drei Verzögerungsleitungen mit je einem Wort, auf der die *central arithmetical unit* ihre Berechnungen durchführte. [7]

### 5.2.5 Der *dispatcher*

Der *dispatcher* hatte die Aufgabe, ein 44-Bit-Wort zu dekodieren, es also in seine Bestandteile aufzuteilen und entsprechend an andere Einheiten weiter zu schicken. Dabei bestand ein Wort aus folgenden Teilen: Die ersten vier Bit befand sich der Operationscode der auszuführenden Aktion, danach kamen konsekutiv die Adresse des ersten Operanden, des zweiten Operanden, die Adresse in die das Ergebnis geschrieben werden soll und die Adresse der nächsten Instruktion, wobei jede Adresse mit zehn Bit adressiert wurde. Eine Liste der Befehle und ihrer Bedeutung finden sich in Tabelle 2. [9]

### 5.2.6 Der *timer*

Diese Einheit war dafür zuständig, ein regelmäßiges Taktsignal für die Synchronisierung und Datenübertragung bereit zu stellen. Hierfür wurde jede Mikrosekunde ein Signal emittiert, durch das (bezogen auf die Datenübertragung) jeweils ein Bit weitergeleitet wurde. Pro Wort wurden allerdings 48 statt 44 Signale gesendet, damit sich die Schaltkreise zwischen aufeinanderfolgenden Wörtern stabilisieren konnten. [9]

### 5.2.7 Datenein- und Ausgabe

Der ursprüngliche Plan war, Daten über Magnetdrahtspeicher einzulesen. Dieser sollte zunächst Daten von regulären Lochkarten auf den Magnetdraht konvertieren, damit er, angeschlossen an eine von drei Leitungen am EDVAC, die Informationen in elektrische Signale umwandeln konnte. Die Ausgabe sollte analog dazu in umgekehrter Reihenfolge stattfinden.

Tabelle 2: Operationen des EDVAC

Operation	Beschreibung
A <i>a b g d</i>	addiere den Inhalt von <i>a</i> zum Inhalt von <i>b</i> , speichere Ergebnis in <i>g</i> , gehe zu Instruktion in <i>d</i>
S <i>a b g d</i>	subtrahiere den Inhalt von <i>a</i> vom Inhalt von <i>b</i> , speichere Ergebnis in <i>g</i> , gehe zu Instruktion in <i>d</i>
M <i>a b g d</i>	multipliziere den Inhalt von <i>a</i> mit dem Inhalt von <i>b</i> (von 84 Bit auf 44 gerundet), speichere Ergebnis in <i>g</i> , gehe zu Instruktion in <i>d</i>
m <i>a b g d</i>	multipliziere den Inhalt von <i>a</i> mit dem Inhalt von <i>b</i> (exakt), speichere Ergebnis in <i>g</i> und <i>g + 1</i> , gehe zu Instruktion in <i>d</i>
D <i>a b g d</i>	dividiere (gerundet) - ähnlich zu M <i>a b g d</i>
d <i>a b g d</i>	dividiere (exakt) - ähnlich zu m <i>a b g d</i>
C <i>a b g d</i>	vergleiche den Inhalt von <i>a</i> mit Inhalt von <i>b</i> , gehe abhängig vom Ergebnis entweder zu Instruktion in <i>g</i> oder in <i>d</i>
W <i>a b g d</i>	(Wire) - lies oder schreibe Information vom oder auf den Magnetdrahtspeicher (später abgeändert zu "write" für Lochbandstreifen)
R <i>a b g d</i>	(Read) - Leseoperation; wurde erst mit dem Wechsel auf das Lochbandstreifensystem eingeführt
E <i>a b g d</i>	(Extract) - shifte ein Wort nach links oder rechts und ersetze einige der Bits mit den Bits eines anderen Wortes (wurde benutzt um die Adressen einer Instruktion zu verändern)
MR <i>a b g d</i>	(Manual Read) - nimm das Bit-Muster, das durch die Schalter der Bedienoberfläche gegeben wurde und speichere es in <i>a</i> , <i>b</i> und <i>g</i> , gehe zu Instruktion in <i>d</i>
H	halte die Maschine an

Es war geplant, dass jede Magnetdrahtrolle 50.000 Wörter halten sollte, was in Verbindung mit den drei Anschlussmöglichkeiten einen gesamten Ausgabepufferspeicher von 150.000 Wörtern ermöglichen sollte. Dieses System stellte sich allerdings aufgrund der Magnetdrähte als zu kompliziert und unzuverlässig heraus, sodass kurz vor Fertigstellung des EDVAC ein Ein-/Ausgabe-System auf Basis von Lochbandstreifen improvisiert wurde. Wegen ihrer kurzfristigen Implementierung war diese Methode selbst im Vergleich zu damaliger Technik etwas primitiv: Das Band musste manuell eingeschoben und herausgezogen werden, es hatte keinen dafür zuständigen Motor. Dabei musste beachtet werden, dass zwischen Informationsblöcken genügend freier Platz gelassen wurde, damit das Band sicher angehalten und gestartet werden konnte. Wollte die Maschine nun eine Leseinstruktion ausführen, so musste per Hand die Länge eines Informationsblocks weitergeschoben werden. Dies stellte offensichtlich eine starke Limitierung dar, da die mechanischen Bewegungen der Bedienperson mit Abstand die langsamste Komponente im System war, sodass ein Beschreiben des kompletten Speichers mehrere Minuten dauerte. [9]

### 5.2.8 Erweiterungen

Eine Besonderheit des EDVAC war, dass er im Laufe der Jahre immer wieder Erweiterungen erhielt. So wurde beispielsweise im Jahr 1954 das langsame Ein-/Ausgabe-System durch einen automatischen Papierbandleser mit hoher Geschwindigkeit ersetzt. Außerdem stellte sich bald heraus, dass der Speicher von 1024 Wörtern auch stark limitierte, weswegen 1955 eine Magnettrommel zur Verfügung stand, die zusätzlich 4.608 Wörter halten konnte. Eine weitere Speichererweiterung fand 1960 statt, als am System Magnetbänder installiert wurden. 1958 wurde auch die Funktion des ENIAC durch eine Einheit erweitert, die Gleitkommaoperationen ermöglichte. Zuvor musste man Gleitkommaberechnungen durch eigens geschriebene Programme umsetzen, wobei man sich der Tatsache bediente, dass von den 16 möglichen Operationscodes nur tatsächlich 12 Verwendung fanden, sodass die restlichen vier für Gleitkommaberechnungen verwendet wurden. Durch die Erweiterung wurden

jedoch schließlich Gleitkommaberechnung durch die Hardware ermöglicht, was einen Geschwindigkeitsvorteil bei entsprechenden Operationen um den Faktor 12 bewirkte. [9]

## 5.3 Programmierung des EDVAC

Die Programmierung des EDVAC war im Vergleich zum ENIAC bereits deutlich komfortabler. Während man bei letzterem sowohl die Speicheradressierung als auch den Programmablauf durch Steckerverbindungen realisieren musste, war dies beim EDVAC bereits durch die Kodierung der Eingabedaten und eine Implementierung von Operationscodes in diese möglich. Die konkreten grundlegenden Operationen, wie Addition oder Multiplikation, waren bereits ähnlich wie heutzutage Mikroinstruktionen in der Hardware umgesetzt. Aufgrund des immer gleichen Aufbaus der Wörter und der Einteilung in Operationscode und Adressen, war das Erstellen eines Programms auch nicht sonderlich kompliziert und gerade die Adressierung jeder Speicherzelle über das eingegebene Wort ermöglichte eine hohe Flexibilität beim Programmieren. Da die Eingabe auch fast ausschließlich über Lochbandstreifen beziehungsweise Lochkarten statt fand, konnte man im Gegensatz zum ENIAC fast das komplette Programm unabhängig vom Zugang zur Hardware im Vorfeld schon erstellen. Hierdurch konnte parallel berechnet und die nächsten Programme bereits geschrieben werden und auch ein Programmwechsel gestaltete sich äußerst kurz, da am Gerät selbst keine Kabel umgesteckt werden mussten oder Ähnliches, sondern lediglich das neue Programm eingelesen und einige wenige Einstellungen durch Drehschalter vorgenommen werden mussten.

Allerdings gab es auch einige Nachteile beim Programmieren. Wollte man sein Programm bezogen auf die Laufzeit optimieren, so musste man aufgrund der akustischen Verzögerungsleitungen, auf denen der Speicher basiert, mit einplanen, zu welchem Zeitpunkt man an welcher Leitung welches Wort abgreifen konnte. Da eine Leitung acht Wörter hielt, die zyklisch durch das Medium wanderten, war es sinnvoll, sein Programm darauf zu optimieren, dass beim

Beenden einer Operation an der Adresse der nächsten Operation auch das gewünschte Wort abgreifbar war, indem man die Speicherorte seiner Operanden günstig wählte, ansonsten kam es zu Verzögerungen. Diese Optimierung erforderte aber natürlich tiefes technisches Verständnis und sehr detailliertes Vorgehen und eine Fehleranalyse des Programms war diesbezüglich nur schwierig. Außerdem konnten zwar Subroutinen umgesetzt werden, allerdings war deren Verwendung recht kompliziert: die Übergabe der Rücksprungadresse musste über eine bestimmte Speicheradresse erfolgen, die mit Hilfe der E-Instruktion in das *d*-Feld der letzten Instruktion der Subroutine geschrieben werden musste. Zusätzlich hatten die Subroutinen eine festgelegte Form, sie waren in drei Abschnitte eingeteilt, wobei in zwei davon nur Konstanten und Variablen stehen durften während in der dritten Instruktionen und Variablen aber keine Konstanten stehen konnten. In Verbindung mit der ursprünglich niedrigen Speicherkapazität und dem nur elementaren Befehlssatz war es vonnöten, sein Programm sehr genau und speziell anzupassen, um es optimieren zu können.

#### 5.4 Bezug und Relevanz zu heutigen Rechnern

Der Aufbau des EDVAC hat starke Ähnlichkeiten zu heutigen Konzepten, vor allem im Vergleich zum ENIAC. Zunächst ist die Von-Neumann-Architektur, also die Speicherung von Programmen und Daten im gleichen Speicher, von der man sagen kann, dass sie im EDVAC Verwendung fand, heute natürlich sehr relevant, da sie primär bei heutigen Rechnern Verwendung findet. Ein wichtiger Schritt war auch die Vereinfachung des logischen Aufbaus und der Programmierung. Der EDVAC hat eine klar eindeutigere funktionale Trennung der Einheiten, wie sie auch in heutigen Rechnern zu finden ist. Den Vorteil erkennt man auch daran, dass es möglich war, den EDVAC im Laufe seines Lebenszyklus durch so viele Komponenten zu erweitern. In der erleichterten Programmierbarkeit findet man wohl den größten Vorzug im Vergleich zum ENIAC und im Hinblick auf heutige Rechner. Heutzutage findet der Großteil des Programmierens auf einer von der zu Grunde liegenden

Technik abstrahierten hohen Ebene statt, was entscheidende Vorteile mit sich bringt und das Programmieren erleichtert. Diese Abstraktion ist beim Schritt vom ENIAC zum EDVAC deutlich zu erkennen und stellt den Anfang einer Entwicklung dar, die bis zu heutigen hohen Programmiersprachen wie C++ oder Java stattgefunden hat. Selbst mit der Aufteilung eines Wortes in Operationscode und Argumente wurde damals das Fundament für die Struktur von Steuerinformationen gelegt, das bis heute Bestand hat.

## 6 Die IAS-Maschine

### 6.1 Geschichte

Die IAS Maschine, MANIAC-0 (Mathematical Analyzer, Numerical Integrator, and Computer oder Mathematical Analyzer, Numerator, Integrator, and Computer), oder einfach nur IAS basiert auf John von Neumanns Arbeit "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument" und wurde, unter seiner Aufsicht, am Institute for Advanced Studies (IAS) in Princeton, New Jersey entwickelt. Mit der Entwicklung wurde 1945 begonnen und der Computer wurde 1951 fertiggestellt. Bis zu diesem Zeitpunkt wurden Universalrechner hauptsächlich für militärische Zwecke verwendet, zum Beispiel um Flugbahnen von Artilleriegeschossen zu berechnen. Nun wurde erstmals ein Universalrechner an einer Universität entwickelt, der dazu gedacht war wissenschaftliche Erungenschaften zu ermöglichen. Dieses Konzept führte unter anderem dazu, dass sämtliche Dokumente, Beschreibungen der Maschine und Schaltpläne veröffentlicht wurden. Auf Grundlage dieser Dokumente konnten Universitäten auf der ganzen Welt ihre eigenen IAS Maschinen bauen. Auch die Programmierbare Elektronische Rechenmaschine München (PERM), die in der Informatikabteilung des Deutschen Museums in München ausgestellt ist eine solche IAS Maschine. [2]

Die Architektur die in dem oben genannten Bericht erläutert wird, wird heute allgemein Von-Neumann-Architektur genannt obwohl John von Neumann nur einer der Verfasser der Abhandlung ist. Neben

John von Neumann haben auch Arthur Burks und Herman Goldstine an der Architektur gearbeitet. Es ist davon auszugehen, dass die Architektur wesentlich von John von Neumann beeinflusst wurde und deshalb nach ihm benannt ist. [4]

## 6.2 Technische Beschreibung

Im Folgenden möchte ich näher auf Aufbau, Funktionsweise und Architektur des Computers eingehen. Und einen groben Überblick über die einzelnen Bausteine geben, aus denen die IAS Maschine besteht. Es ist bereits bekannt, dass ein Computer der eine Von-Neumann-Architektur besitzt aus vier wesentlichen Bausteinen besteht: Speicherwerk, Rechenwerk, Ein-/Ausgabewerk und dem Steuerwerk. Diese vier Bausteine sind auch in der IAS Maschine zu finden in werden in der Abhandlung von John von Neumann beschrieben.

Zunächst werden aber allgemeine Eigenschaften der Maschine beschrieben.

### 6.2.1 Zahlenrepräsentation

Man entschied sich für eine Wortbreite von 40 Bit, da diese Wortbreite für die meisten geplanten Anwendungen ausreichend war. Es konnten aber in der Theorie mehrere Wörter als eine Zahl interpretiert werden. Somit war man in der Lage mit sehr viel grösseren Zahlen zu rechnen. Bei der Planung der Maschine wurde angenommen, dass die meisten Zahlen, die verarbeitet werden sollen, zwischen -1 und 1 liegen. Deshalb wurde eine Kodierung gewählt, die die 40 Bit von links nach rechts zu Vorzeichenbit, gefolgt von  $2^{-1}, 2^{-2}, \dots, 2^{-39}$  kodiert. Wie gewohnt wird das Vorzeichenbit mit 0 für eine positive und 1 für eine negative Zahl kodiert. Tatsächlich ist diese Kodierung aber mehr eine Konvention antstatt eine tatsächliche Eigenschaft der Maschine. Die Konvention wird nur bei der Multiplikation und Division tatsächlich so gefordert. Bei Addition und Subtraction kann der Programmierer seine eigene Kodierung wählen und so zum Beispiel auch mit Integern rechnen, indem der

die 40 Bit als  $2^{39}, 2^{38}, \dots, 2^0$  betrachtet. Der Wertebereich beträgt dann  $[0, 1099511627776]$ . [1]

### 6.2.2 Speicherwerk

Für den Speicher war eine Größe von etwa 4000 Wörtern geplant. Dafür wurden 40 solcher Röhren benötigt. Diese Eigenschaft des Speichers hatte eine weitere wichtige Eigenschaft. Da die Wortbreite des Computers 40Bit betrug, konnte man in einem Schritt ein ganzes Wort aus dem Speicher lesen, indem man einfach je ein Bit aus jeder Röhre las. Somit hatte man die Möglichkeit eine parallele Maschine zu bauen und musste nicht jedes Bit seriell, wie bei vorherigen Maschinen (zum Beispiel EDVAC), ein Bit nach dem anderen verrechnen.

Zwischen Speicherwerk und Rechenwerk gibt es eine Art Puffer, das Selectronregister. Dort werden die aus dem Speicher kommen hinein geladen bevor sie in das Rechenwerk gehen. Andersherum funktioniert das Laden von Werten aus dem Rechenwerk in den Speicher analog. [1]

Natürlich wäre es optimal wenn man für den Speicher eine Reihe von Registern, die aus Elektronenröhren-Flip-Flops bestehen, zur Verfügung hätte. Da man dann aber dazu  $40n$  Flip-Flops, für  $n$  Speicherwörter benötigen würde und  $n$  möglichst groß sein soll, hat man sich für ein Speicherwerk entschieden, das das Speichern von Werten mit sogenannten Selectron-Tubes vorsieht. [1]

**Selectron-Tube** Die Selectron-Tubes wurden ebenfalls am Institute for Advances Studies in Princeton entwickelt. Ein Selectron ist eine Vacuumröhre in der Werte durch elektrostatische Ladungen auf einer isolierten Oberfläche gespeichert werden. In der Röhre befindet sich eine Glühkathode über der isolierten Oberfläche, die als Anode fungiert. Zwischen Kathode und Anode befinden sich zwei Schichten von parallel angeordneten Metallstäben. Die Stäbe sind jeweils orthogonal zu den Stäben der anderen Schicht gesetzt, sodass sich von oben gesehen ein Schachbrett muster ergibt. Durch diese Fenster werden die Flächen auf der darunter liegenden isolierten Oberfläche festgelegt, auf denen dann jeweils ein binäres Zeichen gespeichert werden kann.

Auf die Stäbchen kann unabhängig voneinander eine positive oder negative Spannung angelegt werden. Somit kann man durch anlegen entsprechender Spannungen den Elektronenfluss durch alle Fenster, bis auf ein erwünschtes Fenster, stoppen. Zwischen der Auswahlstruktur und der isolierten Oberfläche befindet sich eine Gitterelektrode an der einheitlich eine Spannung angelegt werden kann. An der Rückseite der isolierten Oberfläche befindet sich eine Metallscheibe. Im folgenden werde ich nun die Lese- und Schreiboperationen genauer beschreiben. [5]

Es wird ein Paar der Metallstäbchen aus der Auswahlstruktur betrachtet. Elektronen können zwischen zwei nebeneinander liegenden Stäbchen hindurch fließen, wenn beide Stäbchen die gleiche positive Ladung haben. Andererseits werden die Elektronen zwischen den beiden Stäbchen abgeblockt, wenn diese eine negative Ladung haben. Das selbe gilt, wenn ein Stäbchen positiv und das daneben liegende negativ geladen ist. Es wird nun klar, dass falls eine zweite Reihe von parallelen Stäbchen rechtwinklig über die andere Reihe gesetzt wird, eine Auswahlstruktur erstellt, in der Elektronen durch ein Fenster fließen können, falls alle vier anliegenden Stäbchen eine positive Ladung haben. Alle anderen Fenster bleiben geschlossen. [5]

### 6.2.3 Rechenwerk

In der Recheneinheit der IAS Maschine gibt es zwei für den Programmierer benutzbare Register: Den Akkumulator und das Multiplikationsregister. Der Akkumulator ist eine parallele Speichereinheit die eine Zahl empfangen kann und diese auf die gespeicherte Zahl addieren kann. Um eine solche binäre Addition durchzuführen werden zwei Schritte benötigt.

Im ersten Schritt werden die Zahlen übereinander gelegt und jedes Paar von Bits addiert. Im zweiten Schritt werden die Carries, die im ersten Schritt entstehen, behandelt. Die Carries müssen nacheinander behandelt werden, da ein Carry ein zweites auslösen kann. Schlimmstenfalls werden so 39 Carries ausgelöst. Subtraktion wird durch Umwandlung der Zahl ins Zweierkomplement und anschließender Addition

umgesetzt. Außerdem kann das Rechenwerk Multiplizieren und Dividieren. [1]

### 6.2.4 Steuerwerk

Wie in der von Neumann-Architektur festgelegt, soll das auszuführende Programm zusammen mit den Daten im Speicher stehen. Um das Programm zu beschreiben, waren zwei Typen von Befehlen geplant. Ein Befehl des ersten Typs holt erst einen Wert aus einer bestimmten Stelle im Speicher in das Selectronregister und wendet dann eine Funktion des Rechenwerks darauf an (normalerweise wird dabei ein Wert, der sich bereits in der Recheneinheit befindet, mit einbezogen). Das Ergebnis der Rechnung wird dabei vorerst in der Recheneinheit behalten und gespeichert.

Ein Befehl des zweiten Typs holt einen Wert aus dem Rechenwerk in das Selectronregister und lädt im zweiten Schritt den Inhalt des Selectronregisters in eine gegebene Stelle im Speicher. Außerdem gibt es bedingte Sprungbefehle und Befehle, die die Ein- und Ausgabe steuern.

Die Wortbreite ist auf 40 Bit festgelegt, aber da 8 Bit für OP-Code und 12 Bit für den Adressteil des Befehls ausreichen, haben in jedem Wort zwei Befehle Platz. Somit ist das komplette Programm aus Paaren von Befehlen, die jeweils einen von den oben beschriebenen Typen haben, aufgebaut. Eigentlich hätten sogar 6 Bit für den OP-Code genügt, aber da man für Daten eine höhere Genauigkeit benötigte, hatte man sich für dieses Format entschieden. Einige Bits des OP-Codes blieben in der entgültigen Maschine wohl unbenutzt.

Während der Ausführung wird zuerst der allererste Befehl ausgewertet, gefolgt von dem zweitem Befehl im aktuellen Befehlspaar. Dieser Prozess wird bis zum Ende des Programms wiederholt. [1]

## 6.3 Besonderheiten

Es ist zu bemerken, dass es sich um einen parallelen Computer handelt. Dies hatte zwar einen klaren Geschwindigkeitsvorteil, machte die Maschine aber durchaus komplexer als ihre Vorgänger.

Tabelle 3: Operationen der IAS Maschine

Operation	Abk.	Beschreibung
$S(x) \rightarrow Ac+$	$x$	Lösche den Akkumulator und addiere die Zahl in Speicherzelle $x$ auf den Akkumulator.
$S(x) \rightarrow Ac-$	$x-$	Lösche den Akkumulator und subtrahiere die Zahl in Speicherzelle $x$ vom Akkumulator.
$S(x) \rightarrow AcM$	$xM$	Lösche den Akkumulator und addiere den absoluten Wert der Zahl in Speicherzelle $x$ auf den Akkumulator.
$S(x) \rightarrow Ac-M$	$x-M$	Lösche den Akkumulator und subtrahiere den absoluten Wert der Zahl in Speicherzelle $x$ vom Akkumulator.
$S(x) \rightarrow Ah+$	$xh$	Addiere die Zahl in Speicherzelle $x$ auf den Akkumulator.
$S(x) \rightarrow Ah-$	$xh$	Subtrahiere die Zahl in Speicherzelle $x$ vom Akkumulator.
$S(x) \rightarrow AhM$	$xhM$	Addiere den absoluten Wert der Zahl in Speicherzelle $x$ auf den Akkumulator.
$S(x) \rightarrow Ah-M$	$x-hM$	Subtrahiere die Zahl in Speicherzelle $x$ vom Akkumulator.
$S(x) \rightarrow xR$	$xR$	Lösche das arithmetische Register und addiere die Zahl in Speicherzelle $x$ auf das Register.
$R \rightarrow A$	$A$	Lösche den Akkumulator und addiere die Zahl im arithmetischen Register auf den Akkumulator.
$S(x) \times R \rightarrow A$	$xX$	Lösche den Akkumulator und multipliziere die Zahl in Speicherzelle $x$ mit der Zahl im arithmetischen Register. Lade die linken 39 Bit der Lösung in den Akkumulator und die rechten 39 Bit in das arithmetische Register.

Tabelle 4: Operationen der IAS Maschine (weiter)

Operation	Abk.	Beschreibung
$A \div S(x) \times A \rightarrow R$	$x \div$	Lösche das arithmetische Register und teile die Zahl im Akkumulator durch die Zahl in Speicherzelle $x$ . Lade den Quotient in das arithmetische Register und den Rest in den Akkumulator.
$Cu \rightarrow S(x)$	$xC$	Springe zu dem linken Befehl in Speicherzelle $x$ .
$Cu' \rightarrow S(x)$	$xC'$	Springe zu dem rechten Befehl in Speicherzelle $x$ .
$Cc \rightarrow S(x)$	$xCC$	Falls die Zahl im Akkumulator $\geq 0$ , springe zu dem linken Befehl in Speicherzelle $x$ .
$Cc' \rightarrow S(x)$	$xCC'$	Falls die Zahl im Akkumulator $\geq 0$ , springe zu dem rechten Befehl in Speicherzelle $x$ .
$At \rightarrow S(x)$	$xS$	Speichere die Zahl im Akkumulator in Speicherzelle $x$ .
$Ap \rightarrow S(x)$	$xSp$	Tausche die linken 12 Bits des linken Befehls in Speicherzelle $x$ mit den linken 12 Bit des Akkumulator.
$Ap' \rightarrow S(x)$	$xSp'$	Tausche die linken 12 Bits des rechten Befehls in Speicherzelle $x$ mit den linken 12 Bit des Akkumulator.
$L$	$L$	Multipliziere die Zahl im Akkumulator mit 2. Wird als Shift um eine Stelle nach links implementiert.
$R$	$R$	Dividiere die Zahl im Akkumulator durch 2. Wird als Shift um eine Stelle nach rechts implementiert.

Außerdem waren alle Dokumente die von gesamten MANIC-0 Team verfasst wurden von fast jedem einzusehen und wurden teilweise auch an wissenschaftliche Institutionen in der ganzen Welt verschickt. Dadurch entstand eine lange Liste an Computern, die auf dem selben IAS Design basierten. Insofern kann man sagen, dass die IAS Maschine zu dieser Zeit und auch danach einen großen Einfluss auf die Struktur und Architektur von Rechenmaschinen hatte.

Der Ingenieur Julian Bigelow fasste den Einfluss dieser technischen Errungenschaft mit folgenden Worten zusammen: ‘‘Es geschah hier... und wir hatten das Glück dabei zu sein... Eine Flutwelle an Rechenleistung war kurz davor zu brechen und alles in Wissenschaft und vielen anderen Bereichen zu überschwemmen und danach würde nichts mehr so sein wie zuvor. Es würde unklare Bereiche, die sich seit Jahrhunderten angehäuft haben, reinigen und lösen.’’ [2]

## 7 Der EDSAC

### 7.1 Geschichte

Der Electronic Delay Storage Automatic Calculator (kurz EDSAC) wurde von 1946 bis 1949 an der University of Cambridge in England konstruiert. Das Projekt leitete Maurice Wilkes, er baute den Computer mit seinem Team im Mathematischen Laboratorium basierend auf den Ideen, die John von Neumann in seinen Abhandlungen darlegte. 5 Die Maschine beruht also in der Tat auf der Von Neumann Architektur und wurde mit der damals aus Rundfunk bewährten Röhrentechnologie implementiert. [8]

### 7.2 Technische Beschreibung

Im Folgenden möchte ich näher auf den Aufbau und die technischen Details der Maschine eingehen.

#### 7.2.1 Zahlenrepräsentation

Jede Zahl und jeder Befehl ist durch eine Reihe von Impulsen codiert. Eine Befehl wird durch 18 Bit und eine Zahl durch entweder 18 Bit für eine kleine Zahl

Tabelle 5: Beispiel der Zahlenrepräsentation

EDSAC	Dezimal
0.1011000000000000	$2^0 + 2^{-2} + 2^{-3} = 1,375$
1.0101000000000000	$-2^{-1} + 2^{-3} = -0,375$

(Short) oder 36 Bit für eine große Zahl (Long) dargestellt. Für Shorts codieren die ersten 16 Bit in einer Zahl die Ziffern mit der niedrigwertigsten zuerst. Danach folgt ein Vorzeichen Bit und ein unbenütztes Bit.

Das Vorzeichenbit ist 0 für eine positive Zahl und 1 für eine negative Zahl.

Bei den Zahlen handelt es sich um Festkommazahlen. Das Binärkomma befindet sich dabei zwischen dem Vorzeichenbit und dem höchstwertigem Bit. Das heißt für Shorts zwischen dem 17. und 16. Bit und für Longs zwischen dem 35. und 34. Bit. Dabei wird das Komma nicht als Impuls abgebildet, sondern die Maschine interpretiert die Zahlen lediglich auf diese Weise.

Negative Zahlen werden durch das Komplement dargestellt. Dabei ist anzumerken, dass alle Bits bis zu einschließlich der ersten 1 gleich bleiben. Alle folgenden Bits einschließlich des Vorzeichens werden invertiert. Ein Beispiel der Zahlenrepräsentation ist hier zu finden 5.

Es ist zu beobachten, dass negative Zahlen um 1 zu hoch sind. Dies berücksichtigt die Maschine. Außerdem kann man erkennen, dass für jede  $n$  Zahl in der Maschine gilt:  $-1 \leq n < 1$ . [8]

#### 7.2.2 Ein-/Ausgabe

Die Maschine betrieb einen Lochstreifenleser um Eingaben vom Programmierer entgegen zu nehmen und einen Creed Teleprinter um Ausgaben für den Programmierer verständlich darzustellen. Diese Geräte wurden vom Ein-/Ausgabewerk betrieben. [8]

**Eingabewerk** Der Lochstreifenleser kann einen Lochstreifen mit 5 Kanälen mit einer Geschwindigkeit von 50 Zeichen pro Sekunde einlesen.

Da jeder Kanal des Lochstreifen ein Bit encodiert,

jedes Loch kann entweder gestanzt sein oder nicht, kann jedes Zeichen  $2^5 = 32$  Werte von 0 bis 31 annehmen. In der Maschine gibt es zwei verschiedene Codierungen. Die Buchstabencodierung und die Zahlencodierung. Die Buchstabencodierung encodiert die Werte 0 bis 31 wie folgt: *P, Q, W, E, R, T, Y, U, I, O, J, figs, S, Z, K, lets, null, F, cr, D, sp, H, N, M, lf, L, X, G, A, B, C und V*. Die Zahlencodierung encodiert die Werte 0 bis 31 wie folgt: *0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ?, figs, “, +, (, lets, null, \$, cr, :, sp, £, „, lf, ), /, #, -, ?, : und =.* Wobei die Zeichen *figs* (Figureshift) und *lets* (Lettershift) jeweils zwischen den beiden Codierungen wechseln. [8]

Dabei wird das höchstwertige Bit (der linke Kanal) complementiert, sodass die Zeilen  ,  und  interpretiert werden als 0(P), 7(U) und 27(G). Das kleine Führungsloch wird dabei benutzt um den Lochstreifenleser in der richtigen Spur und Takt zu halten.

Der Lochenstreifenleser kann mit dem Befehl *In* bedient werden. Über die Lochstreifen wird das Programm mit beinhalteten Daten an die Maschine übergeben. Außerdem kann eine Eingabe über den Teleprinter (Tastatur), mit dem Befehl *Fn* erfolgen. [8]

### 7.2.3 Instruktionssatz

In dem Instruktionssatz kann man erkennen, dass die meisten Befehle aus einem Funktionsteil (OP-Code) und einem numerischen Teil, der eine Speicheradresse angibt, bestehen. Nur einige Befehle bestehen nur aus einem Funktionsteil, ohne Speicheradresse.

Insgesamt gibt es 18 Befehle mit der die ALU bedient, Werte aus dem Speicher in die Register geladen, Werte aus den Registern in den Speicher geladen und bdingte Sprünge ausgeführt werden können. Außerdem gibt es Befehle um die Ein- und Ausgabe zu steuern und einen Halt-Befehl, der das Programm stoppt und die eingebaute Glocke läuten lässt. [8]

Name	Beschreibung
<i>An</i>	Addiere den Wert in Speicherzelle n auf den Akkumulator.
<i>Sn</i>	Substrahiere den Wert in Speicherzelle n von den Akkumulator.
<i>Hn</i>	Kopiere den Wert in Speicherzelle n in den Akkumulator.
<i>Vn</i>	Multipliziere den Wert in Speicherzelle n mit dem Wert im Multiplikationsregister und addiere das Resultat auf dem Akkumulator.
<i>Nn</i>	Multipliziere den Wert in Speicherzelle n mit dem Wert im Multiplikationsregister und substrahiere das Resultat von dem Akkumulator.
<i>Tn</i>	Kopiere den Wert im Akkumulator in die Speicherzelle n und lösche den Wert im Akkumulator.
<i>Un</i>	Kopiere den Wert im Akkumulator in die Speicherzelle n.
<i>Cn</i>	Verunde die Speicherzelle n mit dem Multiplierregister.
<i>R2<sup>n-2</sup></i>	Shifte den Wert im Akkumulator n Stellen nach rechts.
<i>L2<sup>n-2</sup></i>	Shifte den Wert im Akkumulator n Stellen nach links.
<i>En</i>	Falls der Wert im Akkumulator größer gleich 0 ist, setze die Ausführung an Speicherzelle n fort.
<i>Gn</i>	Falls der Wert im Akkumulator kleiner als 0 ist, setze die Ausführung an Speicherzelle n fort.
<i>In</i>	Ließ die nächsten 5 Bit vom Lochstreifen und Speichere sie in die niedrigwertigsten Bits in Speicherzelle n.
<i>On</i>	Gibt den Buchstaben der den 5 höchstwertigen Bits in Speicherzelle n entspricht aus.
<i>Fn</i>	Ließ die nächsten 5 Bit vom Teleprinter in die höchstwertigen Bits der Speicherzelle n.
<i>X</i>	Runde den Wert im Akkumulator auf 16 Bit.
<i>Y</i>	Runde den Wert im Akkumulator auf 34 Bit.
<i>Z</i>	Stoppe die Maschine und läute die Glocke.

### 7.2.4 Speicher

Der Speicher besteht aus akustischen Verzögerungsleitungen. Es gibt 32 sogenannte Tanks, wobei jeder Tank etwa 1,5m lang ist und 32 Werte mit je 17 Bit speichern kann. Insgesamt können also 1024 Wörter gespeichert werden. [8]

### 7.2.5 Order Sequence

Die sogenannte order sequence (Ausführung von einem Befehl) ist in zwei Schritte unterteilt. Das Laden und das Ausführen des Befehls.

**Laden** Im ersten Schritt wird der Befehl aus der entsprechenden Speicherzelle im Hauptspeicher in das Befehlsregister (Ordertank) geladen.

Der Ordertank ist ein einzelner kürzerer Tank und kann genau einen Befehl speichern (18 Bit mit einem unbenützten Bit). Die derzeitige Position im Programm ist ebenfalls in einem kurzen Tank, dem Befehlszähler (Sequence control tank), gespeichert. An dem Befehlszähler ist eine Additionsschaltung geschlossen, die den Befehlszähler in jeder Befehlausführung inkrementiert. Wenn ein Bedingungsbefehl  $En$ , oder  $Gn$  geladen wird und die Bedingung wahr ist, wird der Befehlszähler nicht inkrementiert, sondern mit dem Adressfeld  $n$  aus dem Befehl geladen. [8]

**Ausführen** Im zweiten Schritt wird der geladene Befehl im Ordertank ausgeführt.

## 7.3 Programmierung und Besonderheiten

### 7.3.1 Initial Orders

Um das Programmieren zu erleichtern gab es in der Maschine ein Programm, die sogenannten initial orders, das beschrieb, wie ein Benutzerprogramm einzulesen war. Das Programm wandelte dann die Befehle des Benutzers, die in einer Art Assembler-Code geschrieben wurden, in ein für den Computer verständliches Programm um.

Das initial orders Programm besteht aus nur 31

Befehlen und ist fest verdrahtet auf motorisierten Währscheiben gespeichert. Zu Beginn der Ausführung drehen sich diese Währscheiben, und dass Programm wird in die Speicherzellen  $m[0 - 30]$  geladen.

Die Hauptschleife der Initial Orders, beginnt bei Speicherzelle  $m[6]$  mit  $m[0 - 5]$  als Datenstellen.  $m[0 - 5]$  werden wir folgt genutzt.

In der Hauptschleife werden zuerst 5 Bit (OP-Code) vom Lochstreifen in den Akkumulator gelesen. Die Zahl wird dann in die obersten 5 Bit der Speicherzelle  $m[0]$  geladen. Speicherstelle  $m[1]$  ist jetzt leer.

Nun werden die nächsten 5 Bit in den Akkumulator geladen. Falls der Wert kleiner als 10 ist, handelt es sich um eine Zahl. Das Programm konvertiert den Buchstabencode nun in eine Binärzahl. Dies wird wiederholt, bis der geladene Wert größer gleich 10 ist.

Der Befehl in  $m[25]$  ist in der Form  $TnS$  (zu Beginn der Ausführung  $T31S$ ) und wird benutzt, um eine Instruktion in  $m[n]$  zu laden. Der Befehl wird von den Instruktionen 26-28 angepasst, sodass das Adressfeld um eins inkrementiert wird. In der nächsten Ausführung der Schleife wird dann also in die nächste Speicherstelle  $m[n + 1]$  geladen. Der erste Befehl auf jedem Lochstreifen muss immer  $TnS$  lauten, wobei  $n - 1$  die letzte Adresse des Programms angibt. [8]

#	Befehl	Kommentar
0	$T0S$	Daten: Hält das erste Zeichen des Befehls.
1	$H2S$	Daten: Hält den Adressteil des Befehls.
2	$T0S$	Daten: Hält das zweite Zeichen des Befehls.
3	$E6S$	Daten: Hält Größenangabe vom Adressteil des Befehls.
4	$P1S$	Daten
5	$5S$	Daten: Konstante (10). Benötigt um nach Ende Adressteil zu prüfen.
6	$T0S$	Lösche Akkumulator.
7	$I0S$	Ließ nächste Zeile vom Lockstreifen nach $m[0]$ .
8	$A0S$	Lade $m[0]$ in den Akkumulator.
9	$R16S$	Shifte Akkumulator 16 Stellen nach rechts.
10	$T0L$	Speichere den ganzen Akkumulator in $m[0]$ .
11	$I2S$	Ließ nächste Zeile vom Lockstreifen nach $m[2]$
12	$A2S$	Lade $m[2]$ in den Akkumulator.
13	$S5S$	Substrahiere 10 vom Akkumulator.
14	$E21S$	Springe nach $m[21]$ , falls Akkumulator größer gleich 0.
15	$T3S$	Speiche den halben Akkumulator in $m[3]$ .
16	$V1S$	Adressteil multipliziert mit Multiplikationsregister in den Akkumulator.
17	$L8S$	Shifte Akkumulator 8 Stellen nach links.
18	$A2S$	Addiere $m[2]$ auf den Akkumulator.
19	$S5S$	Substrahiere 10 vom Akkumulator.
20	$E11S$	Springe nach $m[11]$ , falls Akkumulator größer gleich 0.
21	$R4S$	Shifte Akkumulator 4 Stellen nach rechts.
22	$A1S$	Addiere $m[1]$ auf den Akkumulator.

#	Befehl	Kommentar
23	$L0L$	Hat keinen Effekt.
24	$A0S$	Addiere $m[0]$ auf den Akkumulator.
25	$T(31)S$	Lade den fertigen Befehl in die entsprechende Speicherzelle geladen.
26	$A25S$	Lade $m[25]$ in den Akkumulator.
27	$A4S$	Addiere Wert aus $m[4]$ auf den Akkumulator (inkrementiert Adressteil).
28	$U25S$	Speichere Akkumulator in $m[25]$ .
29	$S31S$	Substrahiere $m[31]$ vom Akkumulator.
30	$G6S$	Springe nach $m[6]$ , falls Akkumulator kleiner gleich 0.

## 8 Schluss

Nachdem wir nun vier Röhrencomputer genauer betrachtet haben, ist wohl zu diskutieren in welchem Maße diese auch heute noch relevant sind, beziehungsweise inwiefern diese vier Computer unsere heutigen Rechenmaschinen beeinflusst haben. Ein wichtiger Gesichtspunkt der dabei zu nennen ist, ist der Aufbau dieser Maschinen. Die Von-Neumann-Architektur ist bei allen der beschriebenen Computer angewandt und auch Heute noch werden Prozessoren nach diesem Prinzip strukturiert. Auch wenn sich in den Details einiges geändert hat und man heutzutage sehr viel mehr Rechenleistung zur Verfügung hat, funktionieren die Computer von Damals und Heute doch sehr ähnlich. Als die ersten Röhrenrechner entworfen wurden, war das ein absoluter technologischer und wissenschaftlicher Durchbruch. Auf einem Schlag konnten Probleme in einem Bruchteil der Zeit, die ein Mathematiker per Hand benötigt hätte, gelöst werden. Und mit der IAS Maschine wurden die Baupläne für diese Wunderwerke der Technik für jeden verfügbar.

Zum Schluss möchte ich noch einen letzten Satz von John von Neumann zitieren, der einem doch zu denken gibt, wie man über unsere heutigen Laptops oder auch Supercomputer in wenigen Jahrzehnten denkt. Von Neumann soll diesen Satz nach Vollendung der

IAS gesagt haben.

“It would appear that we have reached the limits of what is possible to achieve with computer technology, although one should be careful with such statements, as they tend to sound pretty silly in five years.” (John von Neumann, ca.1949) [2]

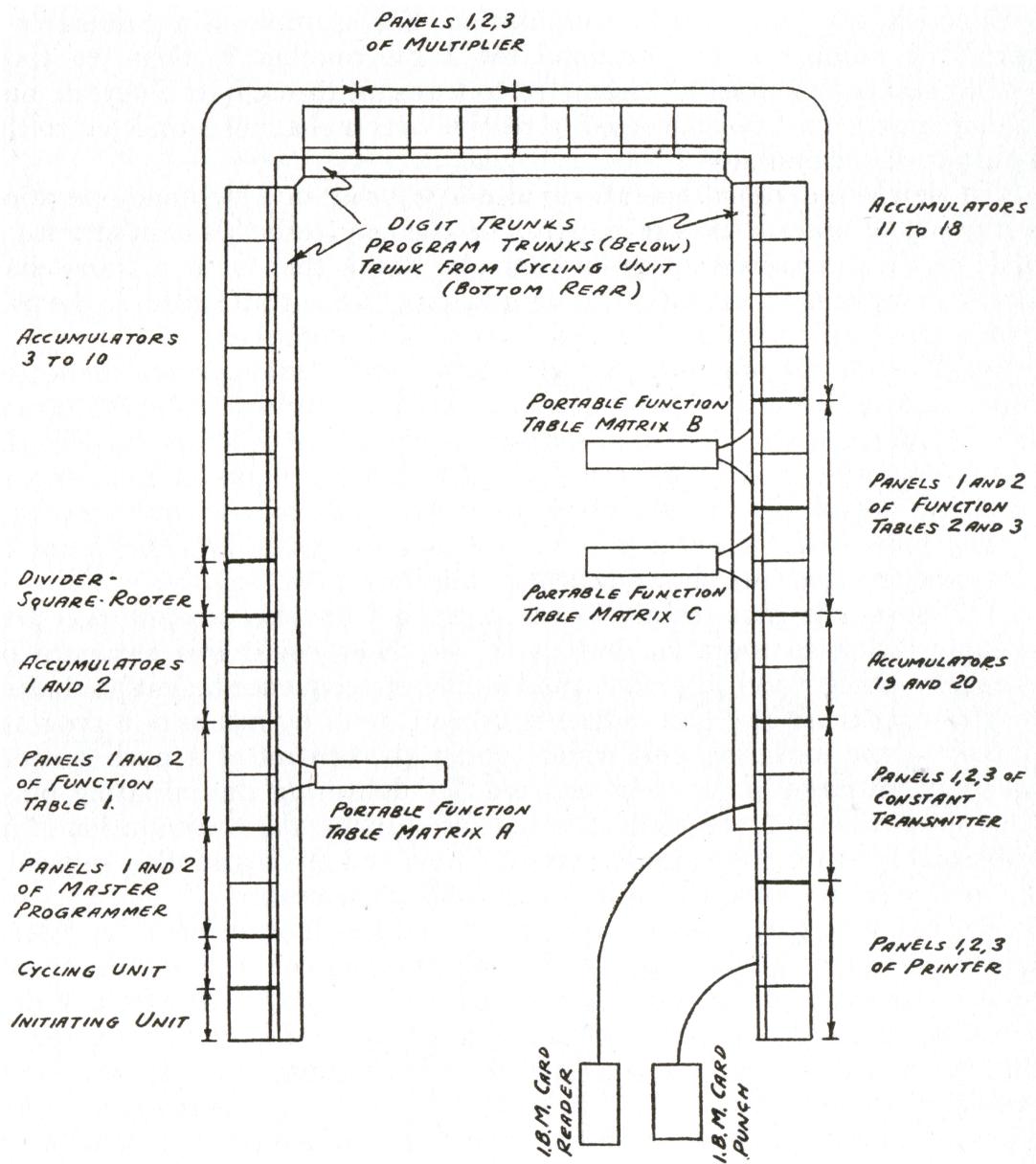


Abbildung 1: U-förmiger Aufbau des ENIAC.

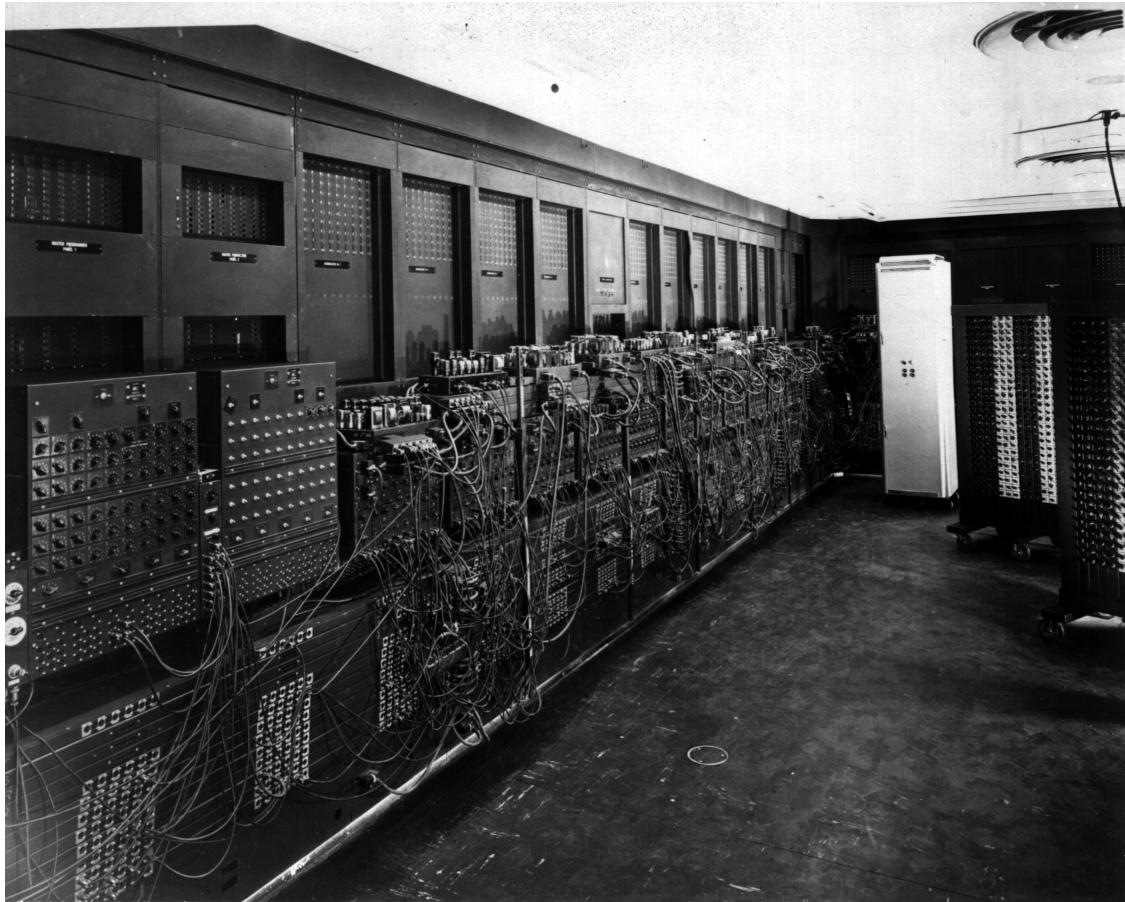


Abbildung 2: ENIAC. Links: Drehschalter für Einstellungen, Mitte: Steckverbindungen zwischen Paneelen/Einheiten.



Abbildung 3: EDVAC. Die Bedienperson stellt an der *central control unit* ein, welche Speichereinheit aktiv sein soll.

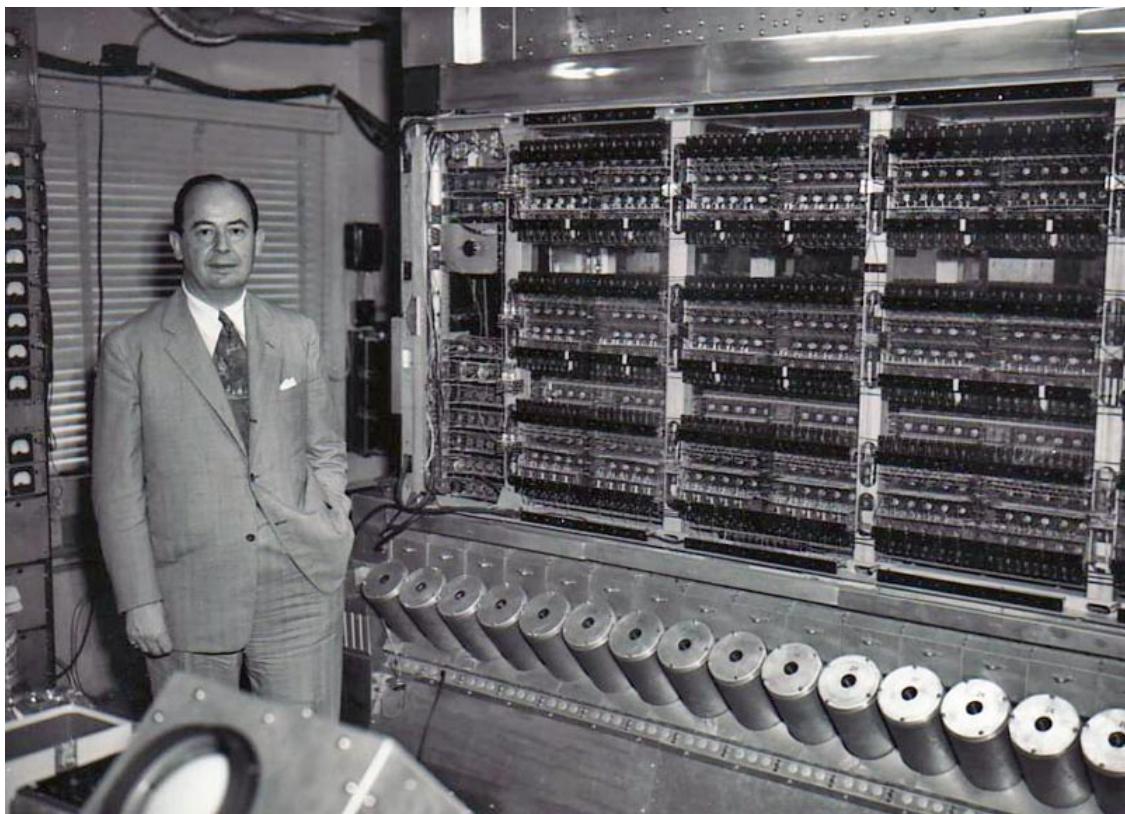


Abbildung 4: John von Neumann vor der IAS Maschine.

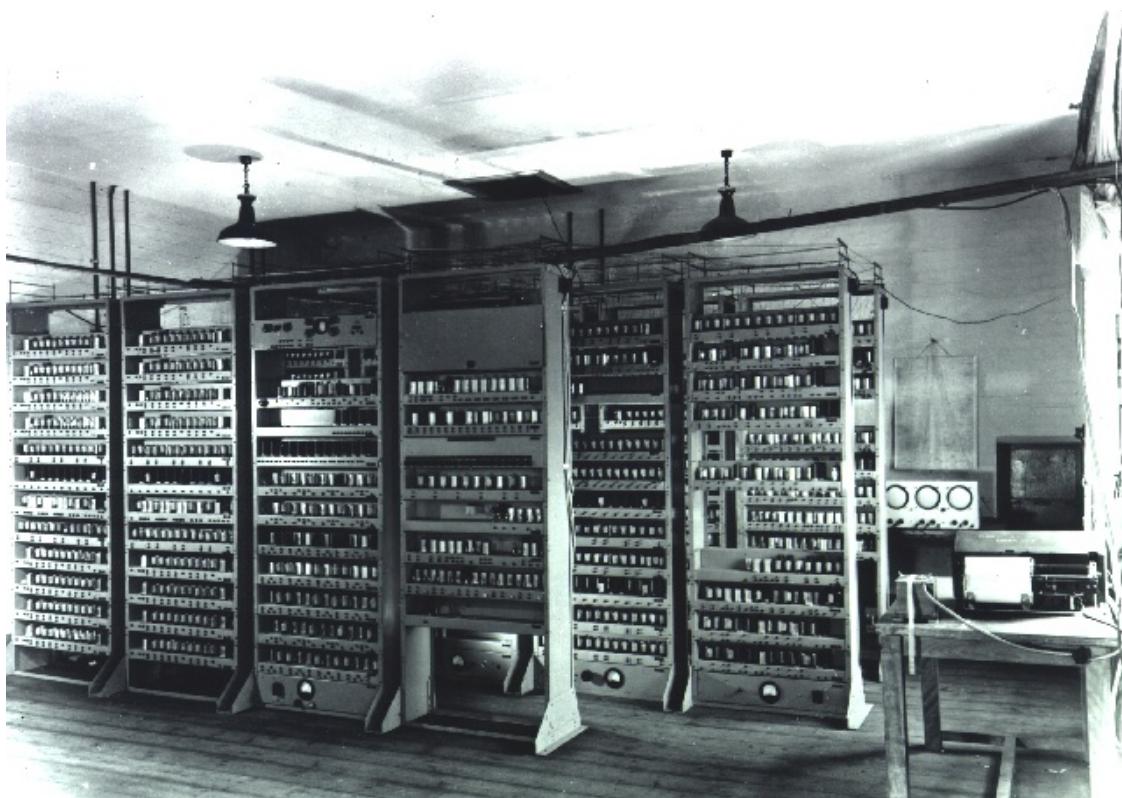


Abbildung 5: Der EDSAC.

## Literatur

- [1] Herman H. Goldstine Arthur W. Burks and John von Neumann. Preliminary discussion of the logical design of an electronic computing instrument.
- [2] Jon R. Edwards. A history of early computing at princeton. <https://www.princeton.edu/turing/alan/history-of-computing-at-p/>.
- [3] H.H. Goldstine and Adele Goldstine. Electronic Numerical Integrator and Computer (ENIAC). *IEEE Annals of the History of Computing*, 18(1):10–16, 1996.
- [4] Karl Kempf. Electronic computers within the ordnance corps. <http://ftp.arl.mil/mike/comphist/61ordnance/chap3.html>.
- [5] Jan A. Rajchman. The selectron - a tube for selective electrostatic storage. 1947.
- [6] Martin H. Weik. A Survey of Domestic Electronic Digital Computing Systems. *Ballistic Research Laboratories Report*, 971:41–42, 1955.
- [7] Martin H. Weik. A Survey of Domestic Electronic Digital Computing Systems. *Ballistic Research Laboratories Report*, 971:27–28, 1955.
- [8] M. V. Wilkes and W. Renwick. *The EDSAC. Report of a Conference on High Speed Automatic Calculating Machines*. Cambridge: University Mathematical Laboratory, 1 1950.
- [9] M. R. Williams. The Origins, Uses, and Fate of the EDVAC. *IEEE Annals of the History of Computing*, 15(1):22–38, 1993.