Die PERM und ALGOL

Steffen Manthey & Klaus Leibrandt

02.07.2002

Inhaltsverzeichnis

1 Einleitung

In unserer Ausarbeitung möchten wir zunächst auf die PERM, die Programmierbare Elektronische Rechenanlage München eingehen. Sie war eine der ersten in Deutschland gebauten von Neumann-Rechner und den an ihren Bau beteiligten Personen gingen bedeutende Impulse für die Informatik aus. Hierzu stellen wir zunächst die Entwicklung und den Bau des Rechners dar und gehen dann näher auf ihren technischen und logischen Aufbau ein. Auch beleuchten wir ihre Auswirkungen auf die Informatik. Im zweiten Teil möchten wir auch noch auf ALGOL eingehen, eine der ersten universellen Sprachen, deren Entwicklung eng mit der PERM verknüpft ist.

2 Die PERM

2.1 Geschichtliches zur PERM

Der Bau einer Rechenanlage an der Technischen Hochschule München wurde von *Robert Piloty* motiviert. Nach Abschluss seines Studiums an der TU München, hatte er 1948 die Gelegenheit wahrgenommen für mehrere Monate am MIT zu arbeiten und hatte dabei die Arbeiten am „Wirlwind"-Projekt, einer programmgesteuerten elektronischen Rechenanlage, mit verfolgen können. Nach der Rückkehr von dieser Studienreise, drängte er seinen Vater Prof. *Hans Piloty*, der zu diesem Zeitpunkt Direktor des Institutes für Nachrichtentechnik an der TU-München war, dazu ebenfalls den Bau einer Rechenanlage ins Auge zu fassen. So reifte der Plan an der TU München ebenfalls eine programmgesteuerte elektronischen Rechenanlage zu bauen - die spätere PERM (Programmgesteuerte Elektronische Rechenanlage München).
Dieses Projekt wurde im Laufe des Jahres 1950 gemeinsam mit dem damaligen Direktor des mathematischen Institutes Prof. *Robert Sauer* Angriff genommen. Seine Arbeitsgruppe, der unter anderem auch Prof. *Bauer* und Prof. *Samelson* angehörten, übernahm die funktionelle und programmiertechnische Planung der PERM. Prof. *Sauer* selbst war Aerodynamiker und damit jemand, der damals Überschallraketen berechnen konnte. Dies war auch seine Motivaton an diesem Projekt teilzunehmen, denn solche Berechnungen dauerten von Hand entsetzlich lang und waren fehleranfällig und ungenau. Mit einem Rechenautomaten ließen sich diese Berechungen in einem Bruchteil der Zeit wesentlich genauer erledigen.
Es war also von Anfang an geplant die Anlage später praktisch einzusetzten, insbesondere auch um den hohen Aufwand zu rechtfertigen. Auf der anderen Seite war man aber auch bestrebt, die Technik der Rechenanlagen anhand eines praktischen Beispiels zu erlernen, das so kompliziert war, dass man sich mit möglichst vielen Bauregeln und Eigenschaften der Technik befassen musste. Da der Rechner in erster Linie als Studienobjekt gedacht war, lag der Focus weniger darauf einen möglichst großen, und leistungsfähigen Rechner zu bauen, als vielmehr möglichst viele Funktionen einzubauen, deren Erforschung interessant und sinnvoll war.
Während der Planungsphase des Rechners griff man zwar intensiv auf den Erfahrungsschatz und die Forschungsergebnisse der Amerikaner zurück insbesondere auf die Literatur *John von Neumanns*. Trotzdem musste man sich viel selbst erarbeiten und man übernahm nichts ohne es genau zu überprüfen, denn man hatte sich vorgenommen die Amerikaner zu überflügeln oder zumindest mit ihnen gleich zu ziehen. Im Laufe der Beschäftigung mit der Thematik wurden die folgenden Leitgedanken für den Bau der PERM entwickelt:

1. Die Maschine sollte so flexibel wie möglich werden, um möglichst vielen Anforderungen gerecht zu werden. Die Handhabung der Maschine sollte möglichst einfach sein und die Programmierung möglichst geringen Aufwand bereiten.

2. Beim Bau der PERM sollte nicht mit behelfsmäßigen Laborschaltungen gearbeitet werden, sondern es sollten industrielle Maßstäbe angelegt werden.

3. Die PERM sollte als Parallelmaschine gebaut werden, bei der die Ziffern einer Zahl gleichzeitig im Raum stehen, statt in zeitlicher Abfolge, wie es bei Serienmaschinen der Fall ist. Der Grund für diese Entscheidung war nicht nur die größere Rechengeschwindigkeit, sondern auch die Tatsache, dass sich in Deutschland noch niemand sonst mit dieser Technik auseinandergesetzt hatte.

4. Die Zahlendarstellung in der Maschine sollte binär sein. Anfangs wurde darüber nachgedacht sich auf Festkommazahlen zu beschränken, aber es wurden dann doch aufgrund allgemeiner Tendenz und dem Rat von erfahrenen Mathematikern Gleitkommazahlen realisiert, obwohl der Aufwand dafür wesentlich größer war. Festes Komma war aber mit der PERM trotzdem möglich.

5. In der Ersten Ausbaustufe wollte man möglichst wenig Arbeit für die Ein-/Ausgabe Geräte verwenden. Deshalb hat man sich für bestehende Geräte aus der Fernmeldetechnik, also leicht modifizierte Fernschreiber entschieden. Es waren außerdem auch Lochstreifenleser verfügbar.

Im Sommer 1952 begann man schließlich mit dem Bau der Anlage. Dabei war der Kreis der daran beteiligten Personen erstaunlich klein, den sowohl die Arbeitsgruppe von Prof. *Sauer* als

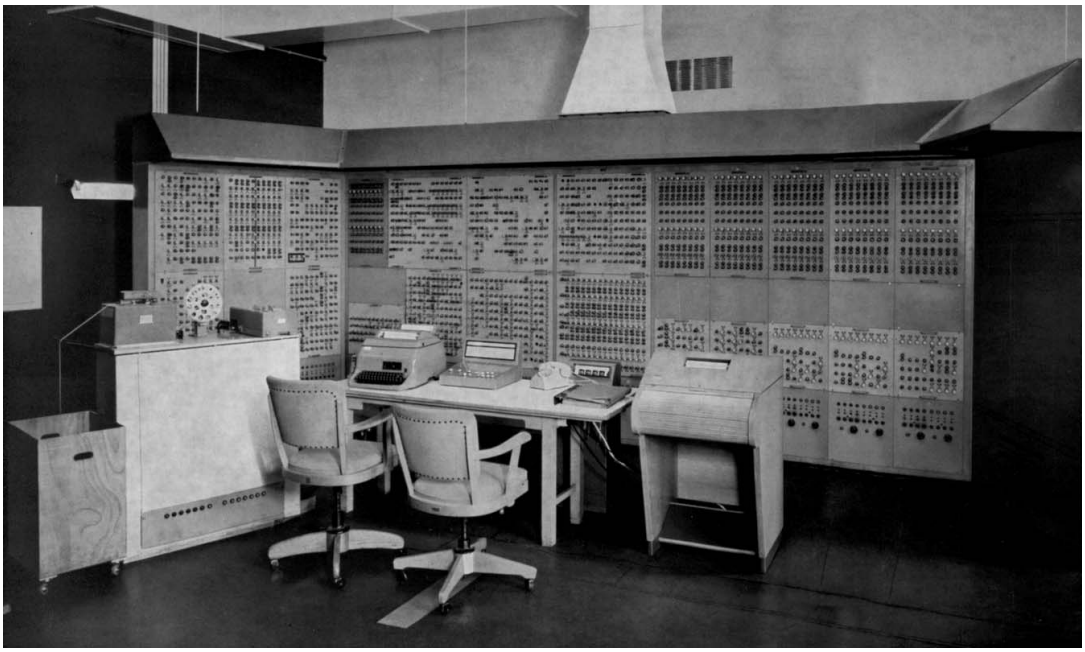

Abbildung 1. Die PERM während des Betriebes an der TU-München.

auch die Arbeitsgruppe von Prof. *Piloty* umfasste jeweils nur drei Personen. Finanziert wurde das Projekt durch die Deutsche Forschungsgemeinschaft, die die Entwicklung des Projektes auch durch Gründung einer Kommission unterstützte, die als Bindeglied zwischen den drei westdeutschen Entwicklungsstellen für Rechenanlagen und den dort arbeiteten Personen diente. Außerdem hat die Kommission die Aufgaben abgestimmt, die an den verschiedenen Entwicklungsstellen zu leisten waren und sorgte unter anderem durch die Veranstaltung von Fachtagungen für regen Gedankenaustausch.
Die erste Ausbaustufe der PERM wurde am 7.5.1956 in Betrieb genommen. Später fanden jedoch noch zahlreiche Erweiterungen statt. So wurden der Magnetbandspeicher, der Pufferkernspeicher und der Kernspeicher erst später hinzugefügt. Auch die Anzahl der befehle und die Adressbreite wurden erhöht.

2.2 Überblick

Die PERM ist, wie bereits erwähnt, eine Parallelmaschine. Hierbei handelt es sich um einen Rechner, der (wie heute allgemein üblich) mit Speicherworten fester Länge arbeitet, die er in einem Schritt übertragen bzw. verarbeiten kann. Im Gegensatz dazu standen damals Rechenanlagen, die Speicherinhalte sequentiell, Stelle für Stelle übertragen bzw. verarbeitet haben.
Während es sich bei heutigen Wortlängen üblicherweise um Vielfache von 8 Bit handelt, betrug die Wortlänge der PERM 51 Bit und entsprechend des der Anlage zugrunde gelegten Parallelprinzips, ist ihr interner Speicher in Zellen dieser Größe unterteilt, die in einem Schritt ausgelesen bzw. geschrieben werden konnten. Erwähnenswert ist jedoch, das keines der Register der PERM auf diese Wortlänge ausgelegt ist, sondern dass die unterschiedlichen Teile des Speicherwortes, je nach ihrer Bedeutung in verschiedenen Registern zum liegen kommen, wenn sie aus dem Speicher gelesen werden. Dies ist einerseits beabsichtigt lässt sich Teilweise aber sicherlich auch auf die damaligen technischen Beschränkungen zurückführen.
Sowohl die Register als auch Rechen- und Steuerwerk der PERM waren Röhrenbasiert. Insgesamt bestand die PERM in aus mehr als 2400 Röhren, Germaniumdioden, von denen immerhin auch 3000 verbaut wurden, wurden nur zur Entkopplung zusammenlaufender Leitungen oder zur Begrenzung von Spannungen verwendet. Der Speicher war nicht aus Röhren aufgebaut,

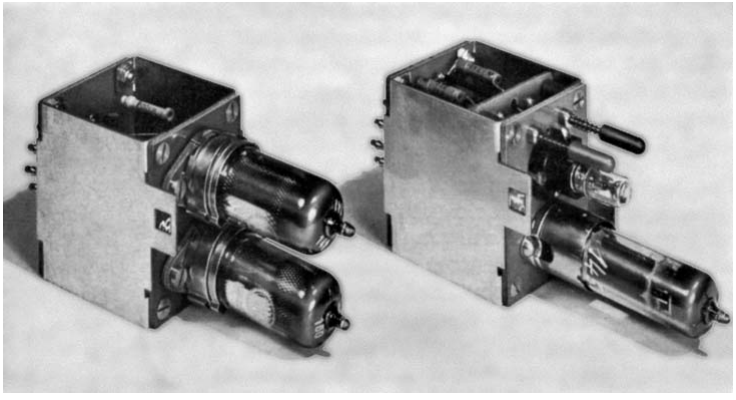

Abbildung 2. Ein Verschiebegatter (links) und ein FlipFlop (rechts) der PERM, beide als steckbare Einschubrahmen realisiert.

da diese Bauelemente hierzu zu raum-, kosten- und wartungsintensiv gewesen währen. Immerhin wäre für jedes FlipFlop und damit auch für jedes Bit eine Röhre notwendig gewesen. Statt dessen wurde der interne Speicher der PERM als Kern- und Magnettrommelspeicher realisiert. Beide Speicher verfügten über einen gemeinsamen Adressraum von insgesamt 10240 Worten. Des weiteren verfügte die PERM über externen Speicher in Form von Magnetbändern und Pufferkernspeicher. Für die Ein- und Ausgabe standen Lochstreifenlese- und -schreibgeräte sowie Fernschreiber zur Verfügung.
Besondere Erwähnung verdient auch, das die PERM, obwohl sie ein Forschungsprojekt war, nicht als Versuchsschaltung gebaut wurde, sondern bei der Fertigung industrielle Maßstäbe angewandt wurden. So wurde die gesamte Anlage nach dem Baukastenprinzip aufgebaut, mit entsprechenden Gestellen und Einschüben die auf übersichtliche Weise miteinander verdrahtet wurden. Die Elemente der PERM, also Grundbausteine wie zum Beispiel FlipFlops, Gatter und Verstärker, wurden zudem als steckbare Einschubrahmen realisiert, um eine leichte Auswechselbarkeit zu gewährleisten und das Löten im Hauptgestell der Anlage zu vermeiden.

2.3 Der interne Speicher der PERM

Wie bereits im vorangegangen Abschnitt erwähnt, wurde der interne Speicher der PERM nicht mittels einer einzigen Technik realisiert sondern besteht aus zwei Teilen: einem Kernspeicher und einem Magnettrommelspeicher. Diese Unterteilung war historisch bedingt, denn anfangs war nur der Trommelspeicher realisiert worden. Der Kernspeicher wurde dann erst in einer zweiten Ausbauphase hinzugefügt.
Beide Speicher verfügten allerdings über einen gemeinsamen Adressraum von insgesamt 10240 Worten. Hierbei entfielen 2048 Worte auf den Kernspeicher, er umfasste die Adressen vom 8912

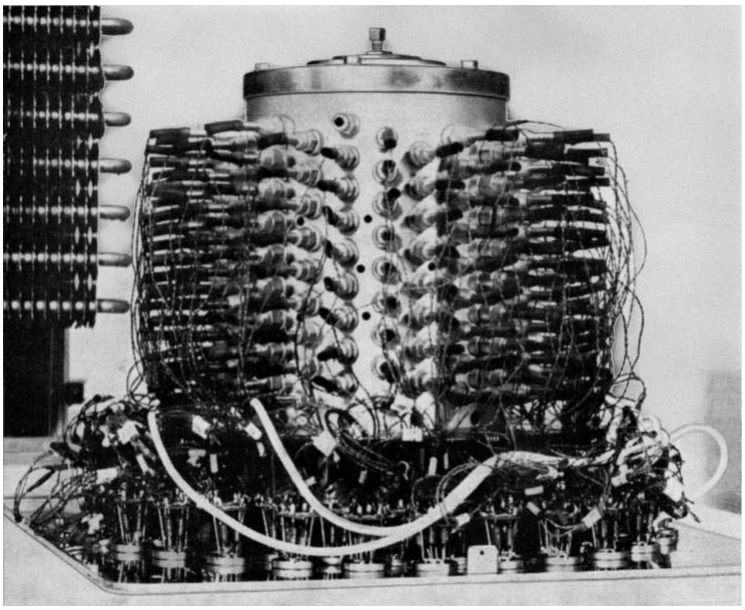

Abbildung 3. Der Magnettrommelspeicher der Perm. Die spiralförmig um die Trommel angeordneten Schreib-/Leseköpfe sind deutlich zu erkennen.

bis 10239, und 8192 Worte auf den Trommelspeicher, der dementsprechend die Adressen von 0 bis 8191 umfasste. Bei einer Wortlänge von 51 Byte sind dies insgesamt fast 64 kByte Speicher (Kernspeicher 13 kByte, Trommelspeicher 51 kByte) auch wenn sich diese Werte natürlich nicht ohne weiteres gleich setzten lassen. Durch den gemeinsamen Adressraum der beiden Speicher bestand für den Programmierer der Maschine theoretisch kein Unterschied zwischen ihnen, im praktischen war aber die Zugriffszeit des Kernspeichers um ein vielfaches kleiner als die des Magnettrommelspeichers.
Da der Magnettrommelspeicher der PERM auch an der TU selbst entwickelt wurde (nämlich am Lehrstuhl für Nachrichtentechnik) und nirgends sonst in der Welt schnellere Trommelspeicher gebaut wurden, möchte ich an dieser Stelle auch einige Worte über ihn verlieren. Beim Trommelspeicher wurde die Information auf der magnetisierbaren Mantelfläche einer rotierenden Trommel gespeichert. Die Oberfläche war in Spuren aufgeteilt, die zum Schreiben und Lesen der Information von je einem Magnetkopf bestrichen wurden. Auf jeder Spur konnten 2048 Punkte unabhängig entweder in der einen (0) oder in der anderen Richtung (L) magnetisiert werden wobei die Trommel 217 Spuren trug. Die Länge der Trommel betrug 22 cm und ihr Durchmesser 10 cm. Sie rotierte mit 15 000 U/min . Die Tourenzahl wurde so hoch gewählt um die maximale Zugriffszeit zu den einzelnen Elementen möglichst klein zu halten. Sie betrug eine Umlaufzeit, in unserem Fall also 4 msec. Dies liegt unter den Werten heutiger Hochleistungsfestplatten. Allerdings liegt die Datendichte heutiger Festplatten mit 25,7 GBit/Quadratzoll wesentlich höher als bei der Trommel mit ca. 5 kBit/Quadratzoll.
Bei der PERM wurde, wie bereits wiederholt erwähnt viel Wert auf das Prinzip der Parallelmaschine gelegt. Dementsprechend war auch der Trommelspeicher so beschaffen, das man alle Binärstellen eines Wortes in einem Schritt auslesen konnte. Jeder Spur des Trommelspeichers war ein Schreib-/Lesekopf zugeordnet. Je 51 Köpfe waren in 4 Gruppen zum Lesen bzw. Schreiben eines Wortes angeordnet. Die restlichen 13 Spuren dienten Spezialaufgaben wie zum Beispiel dem Speichern von Informationen, welche die Adressierung der Speicherzellen auf der Trommel erlaubten.

2.4 Ein vereinfachtes Modell der PERM

Da die PERM ein Vielzahl von Registern mit zum Teil sehr speziellen Funktionen enthält, deren Kenntnis für ein grundlegendes Verständnis der Anlage nicht notwendig ist, führen an dieser Stelle zunächst ein stark vereinfachtes Modell der PERM ein, das bei Bedarf entsprechend erweitert wird. Da die Idee für den Bau der PERM letztendlich auf eine Studienreise von *R. Piloty* in die Vereinigten Staaten zurückging, ist es wahrscheinlich nicht erstaunlich, das die PERM den Prinzipien *von Neumann's* folgte. So lässt sich ihre Struktur in Steuer-, Rechen- und Speicherwerk gliedern.
Das Steuerwerk der PERM verwendete im wesentlichen die folgenden Register:

BZ der Befehlszähler enthielt die Adresse des Befehls, der als nächstes auszuführenden war

KR das Koinzidenzregister diente zur Aufnahme des Adressteils des Befehls, der gerade ausgeführt wurde

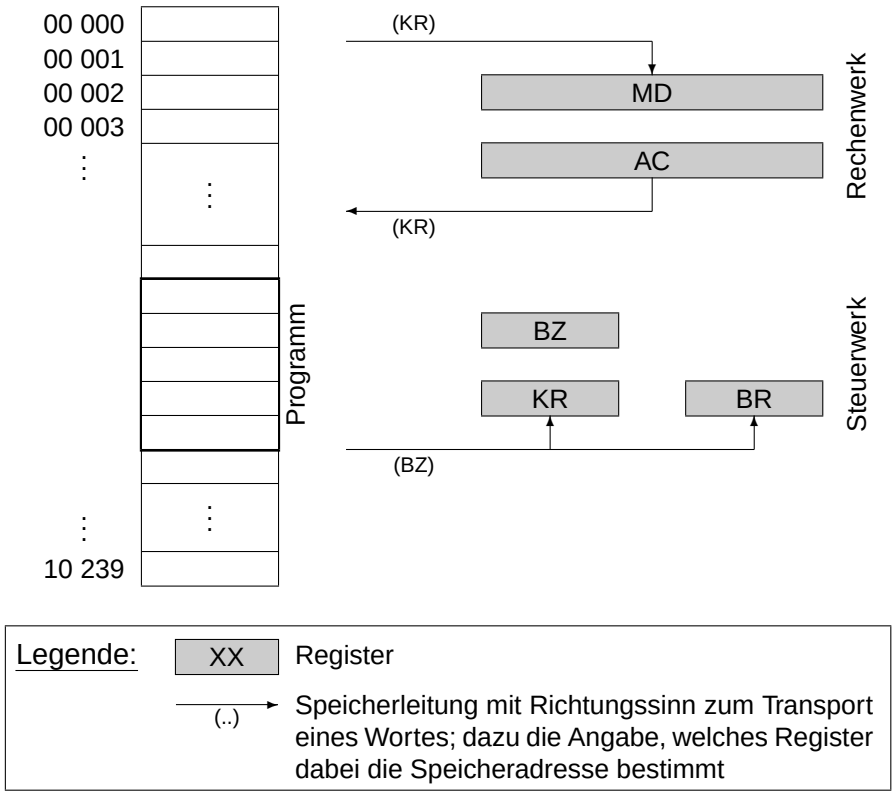

Abbildung 4. Ein stark vereinfachtes Modell der PERM

BR das Befehlsregister diente zur Aufnahme des Befehlsteils des Befehls, der gerade ausgeführt wurde

Jede Befehlsausführung begann jeweils damit, dass der Inhalt derjenigen Speicherzelle ins KR bzw. BR gebracht wurde, dessen Adresse im BZ angeben war. Dabei kamen die Adressenstellen des Wortes ins KR, die Befehlstetraden ins BR. Danach wurde sogleich der Stand des BZ um eines erhöht und im Anschluss daran die im BR stehenden Anweisungen ausgeführt.
Das Rechenwerk der PERM, konnte nur auf Operanden arbeiten die in einem seiner Register standen. Eines dieser Register wurde dabei als Akkumulator verwendet, so dass bei allen arithmetischen und logischen Operationen das verwendete Register implizit feststand und nicht extra angeben werden musste. Die beiden wichtigsten Register des Rechenwerks sind:

MD das Multiplikantenregister

AC der Akkumulator

2.5 Befehls- und Zahlenformat

Wenden wir uns nunmehr der Frage zu, wie Zahlen und Befehle im Speicher der PERM abgelegt wurden. Zunächst ist zu bemerken, das Zahlen im Speicher grundsätzlich als Fließkommazahl abgelegt wurden. Dabei entfielen auf den Exponenten 9 und auf die Mantisse 41 Bit. Der Exponent umfasste die Stellen 1 bis 9 des Speicherwortes und wurde im Zweierkomplement abgespeichert. Aufgrund der Darstellung im Zweierkomplement wurde erste Stelle auch

als Vorzeichen des Exponenten (VE) bezeichnet. Die Mantisse umfasste die Stellen 10 bis 50 wobei die Bits die Stellenwerte von 2^0 bis 2^{-40} annahmen. Da die negative Mantisse modulo 2 dargestellt wurde, wurde das 9. Bit mit der Wertigkeit 2^0 in Analogie zum Exponenten als Vorzeichen der Mantisse (VM) bezeichnet.

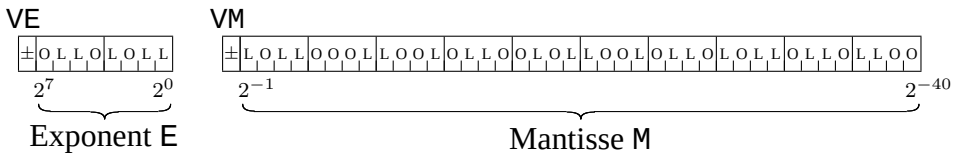

Abbildung 5. Das Zahlenformat der PERM mit Angabe der Wertigkeit der Stellen.

Auch wenn es kein spezielles Format für Integerzahlen gab, so war es doch möglich mit Festkomma zu rechnen. Hierzu unterschied die PERM vier Rechenzustände (Bark, Perm, Quasifest und Fest), die die Befehle für Addition, Multiplikation und Division beeinflussten. Es gab also keine speziellen Befehle für Integer- oder Fließkomma-Arithmetik sondern die Wirkung der arithmetischen Befehle hing vom eingestellten Rechenzustand ab. Der Rechenzustand wiederum konnte mittels entsprechender Befehle gesetzt werden. In den Zuständen Fest und Quasifest wurde mit festen Komma gerechnet, während in den anderen beiden Zuständen mit gleitenden Komma gerechnet wurde.
Auch die Befehlsworte der PERM waren 51 Bits lang. Sowohl der Adressen- als auch der Operationsteil waren in dem Teil des Speicherwortes untergebracht, der bei der Zahlendarstellung der Mantisse (ohne Vorzeichen) entsprach. Die Adressteil hatte eine Länge von 15 Bit damit ließen sich also Adressen (bzw. Zahlen) a aus dem Bereich $0 \leq a < 32768 = 2^{15}$ darstellen.
Der Befehls- oder Operationsteil des Befehlswortes bestand aus 5 Tetraden. Jede Tetrade war mit einer der Zahlen 0 bis 15 besetzt, die in diesem Fall durch die Buchstaben O, A, B, ...,P wiedergegeben wurden. Die Befehlstetraden standen in den Mantissenstellen von 2^{-17} bis 2^{-36}. Die verbleibenden vier Bits der Mantisse bildeten die sogenannte Code-Check-Tetrade, die bei Befehlsworten unbesetzt war. Der Buchstabe O stand in allen 5 Tetraden für den Leerbefehl, dies war der einzige Befehl, der in allen Tetraden verwendet werden konnte. Alle anderen Befehle konnten jeweils nur in einer bestimmten Tetrade auftreten. Somit hatten die restlichen

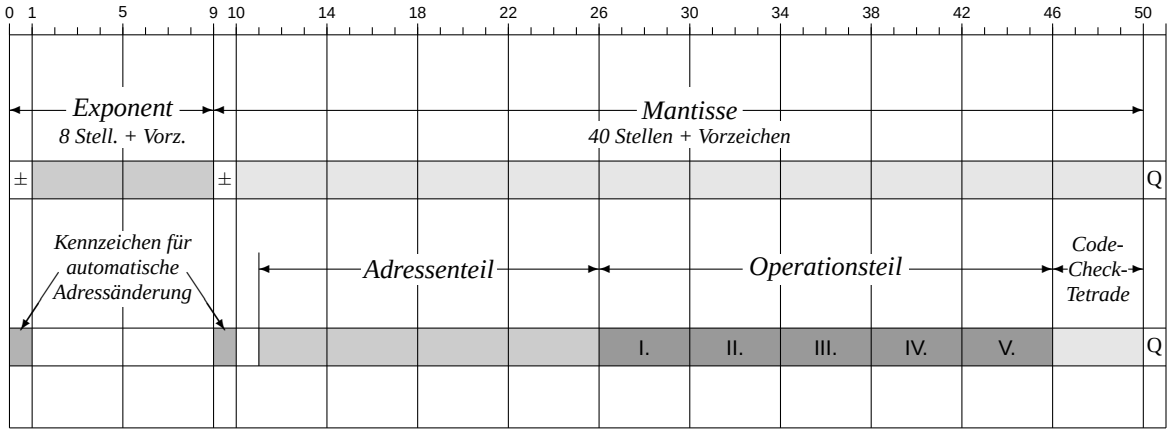

Abbildung 6. Stellenaufteilung innerhalb eines Maschinenwortes

Buchstaben in unterschiedlichen Tetraden auch eine unterschiedliche Bedeutung.
Zum Beispiel stand der Buchstabe `M` in der 1. Befehlstetrade für einen Befehl, der den Inhalt einer Speicherzelle als Zahl interpretierte, die Mantisse negierte und das Ergebnis dieser Operation in das Multiplikantenregister schrieb. In der 2. Befehlstetrade stand dieser Buchstabe für den Multiplikationsbefehl und in der 3. Tetrade für einen Sprungbefehl.
Bei Inbetriebnahme der PERM wurden zunächst nur 4 Befehlstetraden verwendet, die zudem nicht vollständig mit Befehlen belegt waren. Im Endausbau beherrschte die PERM dann 75 verschiedene Befehle, von denen hier einige (ohne Anspruch auf Vollständigkeit) aufgelistet sind:

- Arithmetische Befehle: Addition, Multiplikation, Division, Negation
- Bitoperationen: Exklusives-Oder, Konjunktion, verschiedene Shift-Operationen
- Transportbefehle: Laden von Worten aus dem Speicher in ein Register und umgekehrt; Transport zwischen den Registern
- Sprungbefehle: unbedingte und bedingte Sprünge, in Abhängigkeit vom Inhalt des Akkumulators oder der Zählerregister
- Ein/Ausgabebefehle: Lesen bzw. Schreiben von Daten auf Magnetband, Lochstreifen oder Pufferkernspeicher, Lesen vom Fernschreiber, Konvertierung zwischen verschiedenen Zahlenformaten
- Spezialbefehle: Stop oder bedingter Stop

Die beiden Vorzeichenstellen des Zahlwortes haben auch im Befehlswort eine besondere Bedeutung. Sind werden für Adressmanipulationen verwendet, worauf wir im nächsten Abschnitt genauer eingehen.
Wurde ein Befehlswort über den Fernschreiber eingegeben, so wurden zunächst die beiden Vorzeichen und dann die Adresse gefolgt von den fünf Befehlstetraden eingegeben. Abgeschlossen wurde die Eingabe eines Befehls durch einen Stern. Ein Befehl, der bei negativem Akkumulator-Inhalt einen Sprung zu Adresse 316 ausführt hätte damit folgendes Format:

```
++   316 OOMOO   *
```

Bisher unerwähnt blieb – sowohl beim Zahlen- als auch beim Befehlsformat – das letzte Bit des Speicherwortes, das `Q`-Zeichen. Es konnte verwendet werden um Speicherworte zu markieren. Auch hierauf werden wir später aber noch einmal zurückkommen.

2.6 Adressenmodifikation und Befehlszyklus

Bevor wir nun detaillierter auf Adressenmodifikation und den Befehlszyklus eingehen wird es notwendig ein weiteres Register der PERM einzuführen:

`IR` Das Indexregister diente zur Aufnahme einer Adresse; genauer einer ganzen Zahl aus dem Bereich $-16384 \leq \texttt{IR} \leq 16383$, die im Zweierkomplement gespeichert wurde.

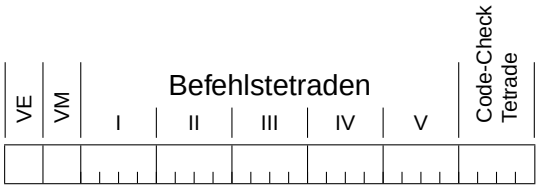

Abbildung 7. Die Aufteilung des Befehlsregisters (BR-Register).

Um zu verstehen wie dieses Register verwendet wir beschäftigen wir uns zunächst genauer mit dem Befehlszyklus der PERM. Zunächst wurden aus der Speicherzelle deren Nummer im Befehlszähler (BZ) steht, die 5 Befehlstetraden, die Code-Check-Tetrade und die beiden Vorzeichen in das Befehlsregister (BR) bzw. die Adressenstellen in das Koinzidenzregister (KR) überführt. Danach wurde der BZ-Stand um Eins erhöht.
Nun wurde das erste Vorzeichen VE des BR geprüft. Ist es negativ, so wurde aus der Speicherzelle, deren Adresse jetzt im KR stand, das Exponentenvorzeichen nach VE und die Adressenstellen nach KR geschafft (hierbei veränderte sich BR abgesehen von VE nicht) und mit der Prüfung von VE die Routine erneut begonnen. Diese Form der Adressenmodifikation kann innerhalb der Bereitstellungphase eines Befehls beliebig oft hintereinander vorkommen. War VE schließlich einmal positiv, so wurde wie folgt fortgefahren: Nun wurde das zweite Vorzeichen VM des BR geprüft. War es negativ, so wurde zum Inhalt des KR der Inhalt des IR addiert; die Summe stand dann im KR. IR blieb unverändert. Ist dagegen VM positiv, so wurde die Bereitstellung des Befehlswortes beendet. Steht die modifizierte Adresse im KR bereit, so wurden die I. bis V. Befehlstetrade der Reihe nach abgefragt und die entsprechenden Operationen wurden ausgeführt.
Damit standen auf der Perm sowohl die indirekte Adressierung als auch die relative Adressierung mittels IR zur Verfügung. Ergänzt wurde diese Möglichkeit der Adressmanipulation durch eine Reihe von Befehlen, die es erlaubten IR zu manipulieren. So war es möglich IR zu dekrementieren oder den Inhalt von KR oder BZ nach IR zu verschieben.
Erst diese Formen der Adressmanipulation ergänzt um die zugehörigen Befehle machten es möglich auf der PERM Kellerdatenstrukturen effizient zu implementieren und waren damit auch die Grundlage dafür, das auf der PERM ein ALGOL-Compiler implementiert werden konnte.
Zu bemerken ist noch, das die PERM noch über ein weiteres Register verfügte, das Adressen aufnehmen konnte. Das Zählerregister ZR. Dieses Register konnte zwar nicht zur relativen Adressierung verwendet werden verfügte aber in etwa über die gleichen Befehle wie das IR und konnte damit zu Beispiel als Schleifenzähler eingesetzt werden.

2.7 Das Q-Zeichen

Eine Besonderheit der PERM ist das Q-Zeichen, mit dem es möglich ist ein Speicherwort zu markieren. Wenn immer ein Wort vom internen oder externen Speicher geladen wird, wird das Q-Zeichen in einem speziell dafür vorgesehenen Register gespeichert. Da es möglich ist auf das Flag mit einem bedingten Sprung zu reagieren kann man das Q-Zeichen zur Manipulation von Programmcode verwenden. So war es denkbar, ein Programm dadurch verschiebbar zu gestalten, dass man alle Befehle, die zum Programmanfang relativen Adressen enthielten, mit dem Q-Flag versah und ein vom Programm unabhängiger Loader zu diesen relativen Adressen

dann die Startadresse des Programmes addierte.

2.8 Zusammenfassung und Ausblick

Die PERM war eine der ersten Architekturen auf der es möglich war einen Compiler effizient zu implementieren. Jeder Compiler benutzt Keller, um geklammerte Strukturen oder Formeln effizient zu analysieren, und ein solcher Keller lässt sich erst mit Hilfe von Adressmodifikation effizient implementieren. Auf der PERM stand hierfür die indirekte Adressierung zur Verfügung. Die relative Adressierung mit Hilfe des Indexregisters `IR` ließ sich nicht direkt verwenden, da hierzu ein Befehl fehlte um `IR` zu dekrementieren. Sie ließ sich aber anderweitig sinnvoll nutzen, um zum Beispiel im Speicher verschiebbare Programme zu erstellen.
An dieser Stelle ist insbesondere auch zu erwähnen, dass das Kellerprinzip von zwei Miterbauern der PERM erfunden wurde. Bereits in den Jahren 1950/51 begann Prof. *Bauer* mit dem Entwurf und dem Bau des „formelgesteuerten Logikrechners STANISLAUS“, der logische Formeln auf ihre Wohlgeformtheit hin untersuchte. Dieser Rechner wurde 1956 fertiggestellt. Dieser Logikrechner bildete schließlich Ausgangspunkt für die Entwicklung des Kellerprinzipes durch Prof. *Bauer* und Prof. *Samelson* im Jahre 1955. Sie entwarfen eine Maschine, die dieses Prinzip realisierte, für die sie 1957 ein deutsches und ein US-amerikanisches Patent erhielten. Auch im weiteren blieben Prof. *Bauer* und Prof. *Samelson* dem Compilerbau verbunden. So waren sie 1958 wesentliche an der der Erarbeitung der Programmiersprache ALGOL beteiligt. Dazu später mehr.
An der PERM wurde eine Vielzahl von weiteren Entwicklungs- und Forschungsarbeiten durchgeführt, so zum Beispiel:

- 1959: Entwicklung des auf der Welt ersten ALGOL-Übersetzers (Prof. *Bauer*, Prof. *Samelson*, *Paul*)
- 1962: Erste Systeme zur Erzeugung von Syntaxanalysatoren (*Eickel*, *Paul*, Prof. *Bauer*, Prof. *Samelson*)

Nachdem die PERM fast 20 Jahre nützliche Arbeit geleistet hatte wurde sie im März 1974 abgeschaltet, ßchlafen gelegt", wie es auf einem nostalgischen Erinnerungstäfelchen hieß. Am 7. Juli 1987 wurde sie schließlich ins Deutsche Museum gebracht, wo sie noch heute ausgestellt ist.

3 ALGOL

3.1 Motivation für die Entwicklung einer Hochsprache

In der Zeit als der die PERM entwickelt wurde war das Programmieren der Rechenanlagen sehr aufwendig und fehleranfällig. Im Allgemeinen war spezielles Personal notwendig, dass die, zum Beispiel vom Mathematiker entwickelten, Algorithmen und Problemstellungen in entsprechenden Maschinencode umsetzte. Die Implementierung eines Algorithmus für eine bestimmte Maschine war natürlich nicht portabel und so war es nicht möglich, einen einmal entwickeltes Programm ohne Neuimplementierung auf einer anderen Maschine zu Laufen zu bringen.
Computer waren in Anschaffung und Unterhaltung sehr teuer und so musste jede Sekunde auf dem Rechner sinnvoll genutzt werden. Es war daher nicht wünschenswert, wenn Programme fehlerhaft rechneten und debugt werden mussten. Allerdings galt auch, dass die Zeiten für die Befehlsausführung im ms-Bereich lagen und man von daher, keine Kompromisse bei der Laufzeit von Programmen zulassen wollte.
Ein erster Schritt für eine vereinfachte Eingabe von Programmen waren Assembler. Die Programmierung in Assembler brachte keine Performance-Einbußen war aber weniger fehleranfällig für den Programmierer. Insgesamt war das Erstellen von Programmen aber immer noch kompliziert. Zudem ließen sich Algorithmen nicht einfach und lesbar in Assembler formulieren, sondern mussten weiterhin erst abstrakt formuliert und dann auf die Maschinenbefehle umgesetzt werden. Auch hatte sich Portabilität der Programme damit nicht verbessert.
Von daher entstand auch bei den Entwicklern der PERM, hier insbesondere bei Prof. *Bauer* und Prof. *Samelson* die Idee, eine maschinenunabhängige Sprache, zu entwickeln, in der man einerseits Algorithmen abstrakt formulieren konnte und die andererseits auch auf möglichst vielen Rechenanlagen lief. Insbesondere hatte ja Prof. *Bauer* ja das Kellerprinzip entwickelt, mit dem es möglich war mathematische Formeln und geklammerte Ausdrücke schnell zu analysieren, und hatte sich damit zum einen bereit an der Materie vertraut und hatte zum anderen einen wesentlichen Beitrag zum Compilerbau geliefert. Auf der Suche nach einer hierfür geeigneten Sprache wurde von beiden unter anderem die Sprache FORTRAN von IBM in betracht gezogen. Man entschied sich aber gegen diese Sprache, da sie unter dem Diktat von IBM stand, und man sich nicht an eine Firma ausliefern wollte, und weil sie zahlreiche Unzulänglichkeiten aufwies. Prof. *Bauer* und Prof. *Samelson* versuchten daher eine möglichst breite Basis für die Entwicklung einer neuen Sprache zu gewinnen. Hierbei wandten sie sich unter anderem an ihre amerikanischen Wissenschaftskolegen und zum anderen an die GAMM, die Gesellschaft für Angewandte Mathematik und Mechanik, die sich damals in Deutschland für die Entwicklung einer solchen Sprache stark machte. Auch unternahm Prof. *Bauer* zwischen 1957 und 1958 Reisen in die USA und konnte auch das Interesse der ACM, der Association for Computer Machinery gewinnen.
Im Herbst 1957 kam es dann in Lugano zu einer ersten Konferenz und ein erster Entwurf der Sprache wurde fixiert. Im Mai 1958 fand in Zürich eine weitere Konferenz statt, auf der auch ein „preliminary draft“ der Programmiersprache erarbeitet wurde. Diese Sprache wurde mit ALGOL- für ALGOrithmic Language - bezeichnet. Trotzdem diese Konferenz fünf Tage dauerte, war es natürlich nicht möglich alle Fragen zu klären und so fand im Januar 1960 in Paris eine weitere Konferenz statt. Hier wurde dann ein endgültiger Standard verabschiedet, der mit ALGOL 60 bezeichnet wurde.

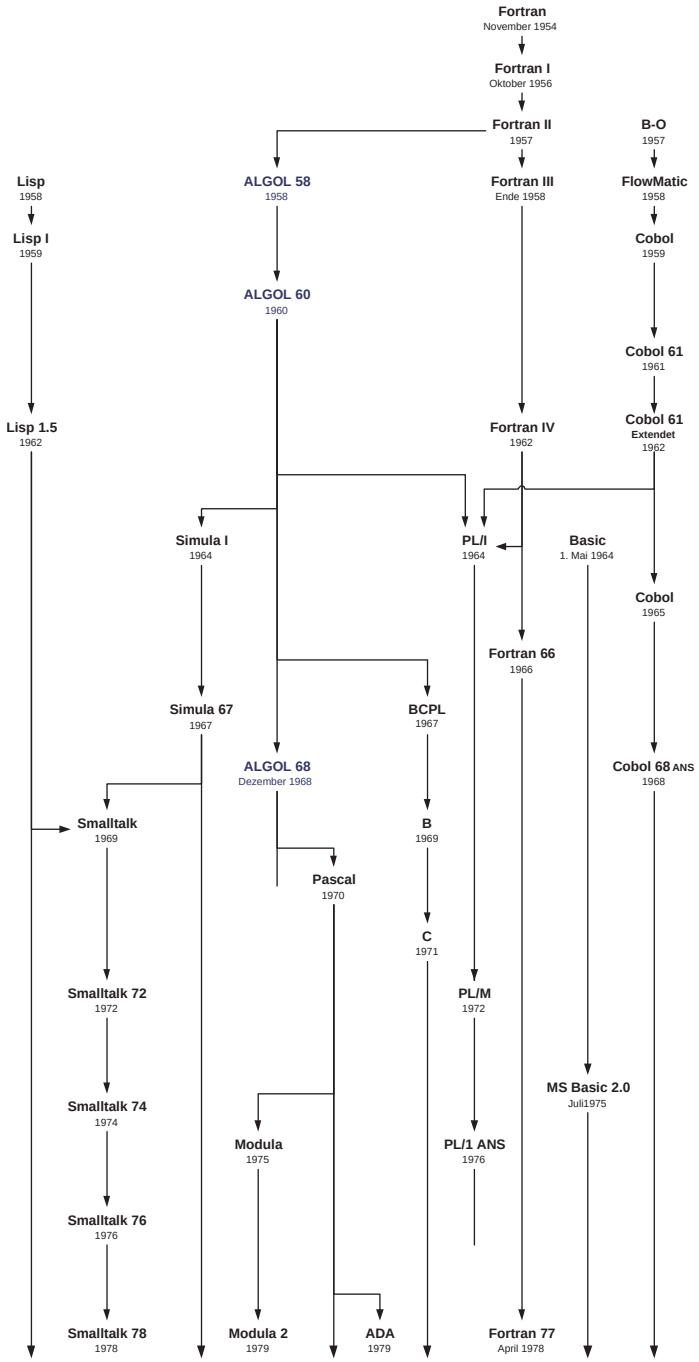

Abbildung 8. Ein Stammbaum für die wichtigsten Programmiersprachen bis zum Jahr 1979.

Schon 1959 entwickelten Prof. *Bauer*, Prof. *Samelson* und Prof. *Paul* den ersten ALGOL-Übersetzer der Welt auf der PERM. Dieser basierte natürlich noch auf dem „preliminary draft“, der auch als ALGOL 58 bezeichnet wurde.
Die Entwicklung von ALGOL und die Verabschiedung der Standards weckte viel Interesse, sowohl im positiven als auch im negativen Sinne. Einerseits kam von europäischer Seite viel Unterstützung, andererseits standen amerikanische Firmen wie IBM der Sprache feindlich gegenüber. Bei dem Bemühen einen Nachfolgestandard zu ALGOL 60, dass spätere ALGOL 68, zu definieren, kam es dann zum Teil sogar zum offenen Eklat zwischen den Wissenschaftlern. So stürmte *Nicolaus Wirth* mit den Worten: „Und wenn ihr nicht meins nehmen wollt, dann lasst Ihr's sein. Ich gehe!“ wutentbrannt aus einer der Sitzungen. Er entwickelte später eine eigene Version von ALGOL und schließlich Pascal.
Ein große Gruppe von Wissenschaftlern darunter auch Prof. *Bauer* und Prof. *Samelson* beteiligten sich jedoch weiter konstruktiv an der Entwicklung der Sprache. Allerdings wurde das Ergebnis nicht sehr erfolgreich, da viele Kompromisse geschlossen wurden, um den Vorstellungen aller gerecht zu werden, so dass letztendlich niemand damit zufrieden sein konnte.
Trotzdem war die Entwicklung der Sprache ALGOL ein großer Erfolg. Wie in Abbildung 8 zu sehen ist, wurden die Ideen und Konzepte, die mit ALGOL eingeführt wurden, von vielen heute sehr populären Sprachen verwendet. Es war auch die erste Sprache bei der versucht wurde die Sprachkonstrukte formal klar zu definieren und sich nicht nach obskuren rechnerspezifischen Gesichtspunkten gerichtet wurde.

3.2 FORTRAN und seine Fehler

FORTRAN war die erste wirklich eingesetzte Programmiersprache. Sie wurde von IBM entwickelt und es verwundert von daher sicher nicht, dass sie zu Beginn stark an der Verarbeitung von Lochkarten orientiert war. Auf Grund der eingeschränkten Rechenleistung und ihres Preises wurde bei FORTRAN alles der Performance untergeordnet und der Compiler als solcher erzeugte tatsächlich sehr effizienten Code. Auch wenn FORTRAN gegenüber der Assemblerprogrammierung ein großer Schritt nach vorne war, denn es war wesentlich einfacher FORTRAN-Programme zu lesen, verstehen und zu debugen als Assemblercode, enthielt die Sprache noch sehr viele Fehler.
FORTRAN hatte in den ersten Versionen keine übersichtliche Sprachdefinition. Ein Beispiel hierfür sind Ausdrücke in FORTRAN. Wenn c eine Konstante und v eine Variable ist, dann ist ein Ausdruck in FORTRAN:

- Innerhalb eines **WRITE**-Statements: c, v, $v + c$ oder $v - c$.
- Als Parameter: c oder v.
- Als ein Array-Index: c, v, $c * v$, $v + c$, $v - c$, $c * v + c$ oder $c * v - c$.
- Als RHS einer Zuweisung: alles

Im Vergleich dazu ist in ALGOL 60 ist ein Ausdruck ein Ausdruck! Hier gibt es keine Ausnahmen und dadurch übersichtlicher und einfacher zu lernen und zu implementieren. Diese Einfachheit und Übersichtlichkeit wurde nicht zuletzt durch die Verwendung der BNF zur Syntaxdefinition erreicht.

Auch besaß FORTRAN keine reservierten Schlüsselworte, so war `IF (I) = 1` war eine Zuweisung an ein Arrayelement. Dabei besitzen reservierte Schlüsselworte eine Reihe von Vorteilen. Sie vermeiden Zweideutigkeiten in der Sprache und vereinfachen die Implementierung des Compilers.
Die Sprachdefinition von FORTRAN wies noch einige weitere Fehler auf. So benötigt FORTRAN keine Variablendeklarationen. Aus vertippern wurden somit automatisch neue Variablen. Des weiteren ignorierte der FORTRAN-Compiler Leerraum. Dies führte zu einem Verhalten, dass heute als Negativbeispiel in fast jeder Compilerbauvorlesung erwähnt wird: Aus einem Schleifenkopf mit der Zählervariable `I`

```
DO 50 I = 1, 10
```

konnte durch einen Vertipper, bei dem das Komma durch einen Punkt vertauscht wurde

```
DO 50 I = 1. 10
```

eine Zuweisung an eine Variable `DO50I` werden.
Auch die Mächtigkeit der Sprache war in der ersten Version stark eingeschränkt. So kannte FORTRAN keine Subroutinen (wohl aber Bibloteksfunktionen) und **DO** , **IF**, und **GO TO** waren die einzigen Kontrollstrukturen.

3.3 ALGOL

Bei der Definition von ALGOL versuchte man diese Unzulänglichkeiten auszuräumen. Denn generell gilt, dass es einfacher ist Fehler zu verbieten, als sie zuzulassen und dann zu finden. Auch die Mächtigkeit der Sprache sollte gegenüber FORTRAN deutlich vergrößert werden.
Diese Ziele wurden recht gut verwirklicht und die Bedeutung von ALGOL kann man sich insbesondere daran verdeutlichen, das die meisten heute verwendeten Programmiersprachen noch Konstrukte verwenden, die damals für ALGOL definiert wurden.
Insbesondere war ALGOL die erste Sprache die in BNF definiert wurde. Tatsächlich war sowohl J.W. Backus als auch P. Naur an der Definition von ALGOL beteiligt. Dies trug wahrscheinlich wesentlich zur Übersichtlichkeit der Sprache bei. Statt der vielen Sonderfälle bei Ausdrücken in FORTRAN, war in ALGOL ein Ausdruck immer ein Ausdruck, egal in welchem Kontext er stand.
Im folgenden möchten wir die wesentlichen Sprachmerkmale und Neuerungen von ALGOL kurz aufführen:

- In ALGOL waren alle Schlüsselworte reserviert.
- ALGOL ist eine blockstrukturierte Sprache. Insbesondere wurden mit ALGOL verschachtelte Gültigkeitsbereiche von Bezeichnern und Verdeckung von Variablen eingeführt (nested scopes).
- Es besteht die Möglichkeit Funktionen und Prozeduren zu definieren. ALGOL die führte auch die Möglichkeit ein Funktionen rekursiv aufzurufen.
- Variablen müssen deklariert werden. Hierzu stehen die Datentypen **'INTEGER'**, **'REAL'** und **'BOOLEAN'** zur Verfügung.

- Mit ALGOL wurde die **IF-THEN-ELSE** und die **SWITCH** Anweisung zur bedingten Ausführung eingeführt. FORTRAN kannte nur das einfache **IF** ohne **ELSE**.

- Neben einem **FOR-UNTIL**-Schleifenkonstrukt (**FOR**-Schleife) das in ähnlicher Form auch FORTRAN schon kannte führte ALGOL ein **FOR-WHILE**-Schleifenkonstrukt (**WHILE**-Schleife) und damit flexible Schleifen ein.

- Es sind unbedingte Sprünge mittels **GOTO** möglich.

- In ALGOL können Arrays dynamisch deklariert werden. Zum Beispiel können die folgenden Deklarationen verwendet werden:

```
'ARRAY' a, b, c [7:n, 2:m], s [-2:10];
'INTEGER' 'ARRAY' A ['IF' c<0 'THEN' 2 'ELSE' 1:20];
'REAL' 'ARRAY' q [-7:-1]
```

ALGOL-Quelltext 1. Beispiele für Array-Deklarationen.

- Bei der Übergabe von Parametern an Prozeduren war sowohl call-by-name (Namensaufruf) als auch call-by-value (Wertaufruf) möglich. Standardmäßig wurden alle Parameter per call-by-name übergeben. Um Parameter per call-by-value zu übergeben, müssen sie im Kopf der Prozedurdeklaration nach der optionalen **VALUE**-Anweisung aufgeführt werden.

  Beim call-by-name verhält sich das Unterprogramm so, als ob es eine textuelle Kopie des Parameters übergeben bekommt. Betrachten wir hierzu das Beispielprogramm 2:

```
'INTEGER' 'PROCEDURE' test(x);
'INTEGER' x;
'BEGIN'
    'INTEGER' a, b;
    a := 3;
    b := 5;
    test := a * x + b
'END'
```

ALGOL-Quelltext 2. Ein Unterprogramm zur Demonstration von call-by-name.

  Ein Aufruf dieser Funktion mittels a := 9; r := test(a+1); liefert durch call-by-name 17 und nicht 35 wie es der Fall wäre, wenn call-by-value verwendet werden würde.

  call-by-name mag mathematisch gesehen sehr sauber sein, letztendlich verletzt es aber die Prinzipien der Kapselung und des Information Hiding, denn die Bezeichner im aktuellen Parameter interferieren mit den Bezeichnern im Rumpf des Unterprogramms.

Damit man einen ungefähren Eindruck von ALGOL erhält, zeigen wir an dieser Stelle, als Beispiel zwei kleine ALGOL-Programme, die beide die Fakultät berechnen:
Obwohl fast 20 Jahre lang alle Algorithmen im akademischen und wissenschaftlichen Bereich in ALGOL pupliziert wurden konnte ALGOL nie die selbe praktische Bedeutung wie FORTRAN

```
'INTEGER' 'PROCEDURE' iFactorial (n); 'VALUE' n;
'INTEGER' n;
'BEGIN'
    'INTEGER' i,fac;
    'FOR' i:=1 'STEP' 1 'UNTIL' n 'DO' 'BEGIN'
        fac := fac * i
    'END';
    iFactorial := fac
'END'iFactorial
```

ALGOL-Quelltext 3. Ein iteratives Beispielprogramm, das die Fakultät einer Zahl n berechnet.

```
'INTEGER' 'PROCEDURE' rFactorial (n); 'VALUE' n;
'INTEGER' n;
'BEGIN'
    'IF' n = 0 'THEN' rFactorial := 1;
    'ELSE' rFactorial := n * rFactorial (n - 1)
'END'rFactorial
```

ALGOL-Quelltext 4. Ein rekursives Beispielprogramm, das die Fakultät einer Zahl n berechnet.

erreichen. Dies lag zum einen daran das FORTRAN im Gegensatz zu ALGOL von IBM unterstützt wurde und drei Jahre eher da war, zum anderen wurden aber trotz allem einige Fehler gemacht.
Die Sprachdefinition von ALGOL 60 sah kein I/O vor, da I/O als hardwarespezifisch angesehen wurde. Die meisten Implementierungen entliehen daher die I/O-Funktionen von Fortran. call-by-name war zwar mächtig aber schwierig zu implementieren und fehleranfällig insbesondere interferiert mit nested scopes.

3.4 ALGOL im Vergleich mit FORTRAN

Feature	ALGOL	FORTRAN
Variablendeklaration obligatorisch	✓	-
dynamische Arrays	✓	-
benuzerdefinierte Datentypen	-	-
Prozeduren und Funktionen	✓	-
Rekursion	✓	-
Bibliotheksfunktionen	✓	✓
Blockstrukturiert	✓	-
nested scopes	✓	-
for-Schleifen	✓	✓
flexible Schleifen	✓	-
if-Anweisung	✓	✓
if-then-else-Anweisung	✓	-
switch-Anweisung	✓	-

3.5 ALGOL und die PERM

Da die PERM indirekte Adressierung und Keller unterstützt, war es möglich auf ihr einen ALGOL-Compiler zu realisieren. Insbesondere wurde der erste ALGOL-Compiler überhaupt auf der PERM realisiert. Besonders interessant ist in diesem Zusammenhang das Projekt FUE 67. Es handelt sich dabei um eine Dreiteilige Diplomarbeit, die von *Manfred Kunas* (Teil 1), *Ulrich Peters* (Teil 2) und *Werner Streitwieser* (Teil 3) erstellt wurde. Die Zielsetzungen dieses Projektes waren:

- die Erhöhung der Betriebssicherheit der PERM
- eine einfache Bedienung für den Operator
- schnelle Übersetzungszeiten für ALGOL Programme
- gute Fehlererkennung während der Programmübersetzung
- Meldung von Fehlern, die zur Laufzeit auftreten
- die sprachlichen Einschränkungen gegenüber ALGOL 60 gegenüber dem bestehenden Compiler auf ein Minimum zu reduzieren (vor allem sollte volle Rekursivität zugelassen werden und unbeschränkter Namensaufruf ermöglicht werden) und es sollten Testhilfen und Eingriffsmöglichkeiten für das Objekt-Programm zur Laufzeit geschaffen werden.

FUE 67 stellt also im wesentlichen ein Betriebssystem mit integriertem ALGOL-Compiler dar. Das System war auf den Übersetzerbetrieb zugeschnitten.
Die Eingabe der ALGOL-Programme erfolgte über Lochstreifen. Beim einlesen wurde jeweils die Anzahl der **'BEGIN'** und **'END'** Statements gezählt, wodurch es möglich wurde das Programmende zu erkennen. Das war genau dann, wenn das zum ersten **'BEGIN'** passende **'END'** eingelesen wurde.
Das Programm wurde dann über mehrere Zwischensprachen in die endgültige Objektform überführt und konnte dann entweder auf Lochstreifen, oder auf Magnetband gespeichert werden. Es kann außerdem auf eine Programm-/Funktions-Bibliothek zurückgegriffen werden.

Literatur

[1] Elementare Einführung in den PERM-Interncode
mit einem Auszug aus der Maschinenbeschreibung als Anhang
Christoph von Conta
Bericht, Technische Universität München, 1971

[2] Pioniere der Informatik
Interviews mit F. L. Bauer, C. Floyd, J. Weizenbaum, N. Wirth und H. Zemanek
Siefkes, Dirk
Springer Berlin, 1999

[3] Realisierung eines Algol-Übersetzers für die PERM Teil I
Manfred Kunas
Diplomarbeit, Technische Universität München, 1969

[4] Realisierung eines Algol-Übersetzers für die PERM Teil II
Ulrich Peters
Diplomarbeit, Technische Universität München, 1969

[5] Realisierung eines Algol-Übersetzers für die PERM Teil III
Werner Streitwieser
Diplomarbeit, Technische Universität München, 1969

[6] Die Programmgesteuerte elektronische Rechenanlage München (PERM), 1. Teil
H. Piloty, R. Piloty, H. O. Leilich, W. E. Proebster
Nachrichtentechnische Zeitschrift, 1955, Heft 11, Seite 603-609

[7] Die Programmgesteuerte elektronische Rechenanlage München (PERM), 2. Teil
H. Piloty, R. Piloty, H. O. Leilich, W. E. Proebster
Nachrichtentechnische Zeitschrift, 1955, Heft 12, Seite 650-658

[8] ALGOL 60 compiler and interpreter
RHA (Minisystems) Ltd.
http://www.angelfire.com/biz/rhaminisys/algol60.html

[9] ALGOL (ALGOrithmic Language)
Programmierung und Programmiersprachen
Prof. Dr. E. Rahm
Universität Leipzig, Institut für Informatik
http://dbs.uni-leipzig.de/skripte/PPS/HTML/kap2-3.html

[10] Programming Languages
David Matuszek
Villanova University Department of Computing Sciences
http://www.csc.vill.edu/~dmatusze/resources/general/beginning.ppt

[11] Computer Languages History
Éric Lévénez
http://perso.wanadoo.fr/levenez/lang/

[12] Die Entwicklung der Informatik in München
Alexander Bock
Technische Universität München, Institut für Informatik
http://wwwbib.informatik.tu-muenchen.de/Fak_Schriften/Fak_Schrift_97/IN_Zeit_2.html

[13] Die Informatik an der TUM
Guenter Heiss
Technische Universität München, Institut für Informatik
http://www3.informatik.tu-muenchen.de/studienberatung/aktuelles/info96/info/node3.html

[14] Laudatio zur Verleihung der Ehrendoktorwürde an Herrn Prof. Dr. Dr. h.c. mult. Friedrich Ludwig Bauer
Daniel Gerold
Technische Universität München, Institut für Informatik
http://wwwbib.informatik.tu-muenchen.de/Stroehlein/Auszeichnungen/auszeichnungen98/FLB_Ehrenprom/Laudatio.html

[15] Rechen-Meister
Martina Crasovschi
Technische Universität München, Institut für Informatik
http://wwwbib.informatik.tu-muenchen.de/Fak_Schriften/Fak_Schrift_97/rechenm.html

[16] Revised Report on the Algorithmic Language ALGOL 60
Peter Naur
mass:werk - media environments
http://www.masswerk.at/algol60/report.htm