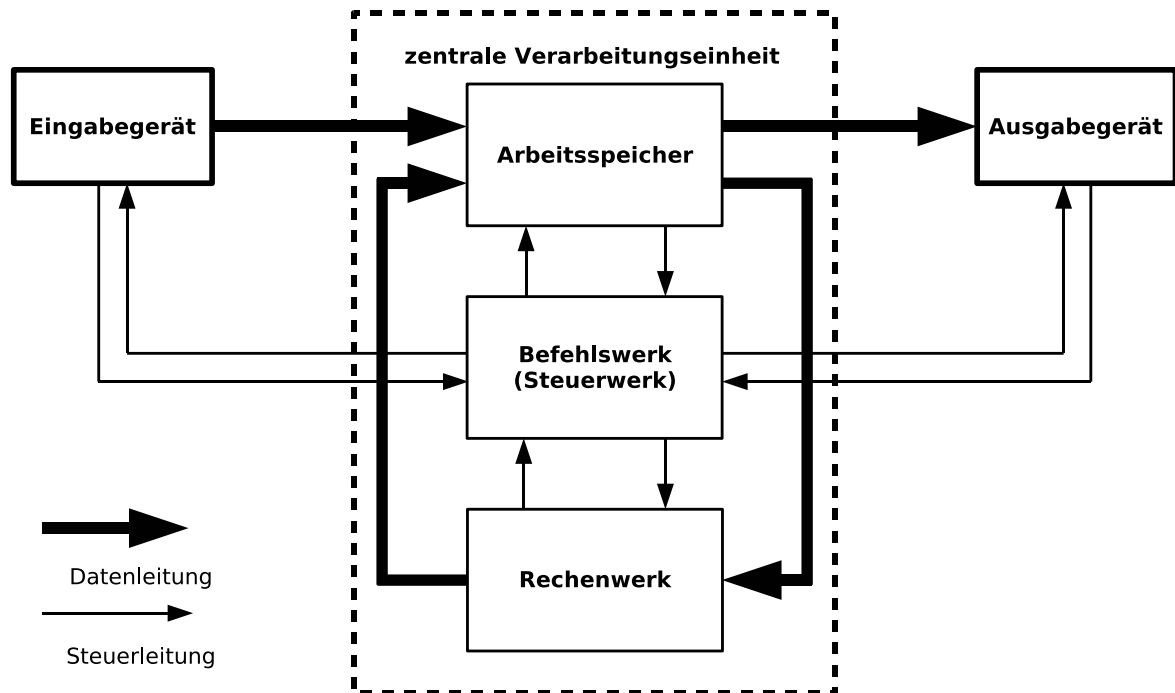


## A1 Einführung

Eine Datenverarbeitungsanlage besteht allgemein aus einem oder mehreren Eingabegeräten (Messfühler (Sensoren), Tastatur, Schalter, mobiler Datenspeicher (USB-Stick, Diskette)...), einer zentralen Verarbeitungseinheit (Computer, Mikrocontroller) und einer Ausgabereinheit (Drucker, Bildschirm, Lautsprecher, LED ...).



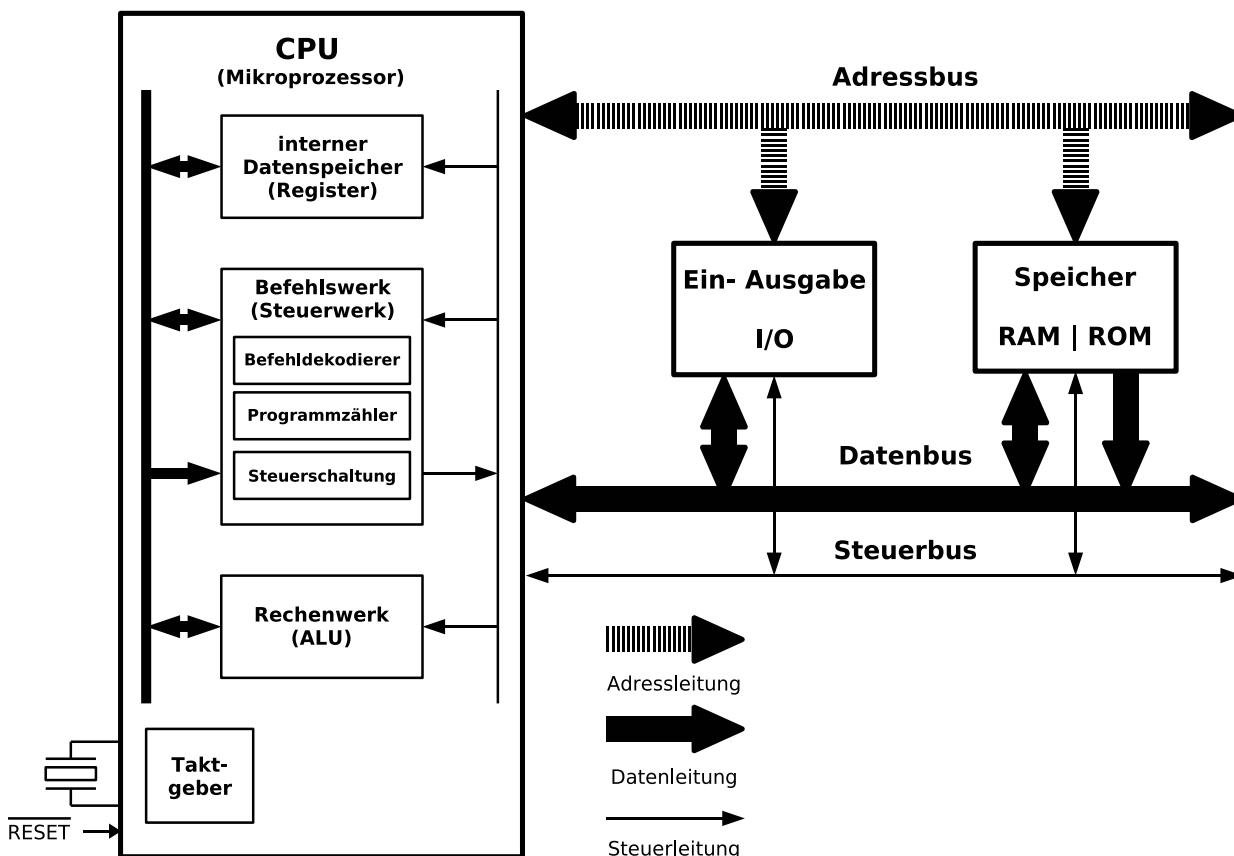
Die **Verarbeitungseinheit** oder **Zentraleinheit** ist für die Verarbeitung der Daten mittels eines Programms, für die Ein- und Ausgabe der Daten und für die Programmsteuerung verantwortlich. Sie besteht aus:

- **Rechenwerk** (Arithmetische und logische Einheit, ALU). Hier werden alle die Daten betreffenden Operationen ausgeführt.
- **Arbeitsspeicher** (Zentralspeicher, Hauptspeicher). Die Daten erhält das Rechenwerk aus dem Arbeitsspeicher und speichert sie auch wieder dahin zurück.
- **Befehlswerk** (Leitwerk, Steuerwerk). Das Befehlswerk steuert alle Arbeitsvorgänge und bewirkt das richtige Zusammenspiel dieser Vorgänge. Es führt alle von den Befehlen vorgegebenen Operationen aus.

## Aufbau eines Mikroprozessorsystems

Das Herzstück eines heutigen Mikroprozessorsystems (Computer, Mikrocomputer) ist der Mikroprozessor (**CPU**<sup>4</sup>, **C**entral **P**rocessing **U**nit, Zentraleinheit), der das Rechenwerk und das Befehlswerk enthält.

Um funktionsfähig zu sein benötigt der Mikroprozessor zusätzlich noch Speicher und Ein-/Ausgabe-Bausteine.



Ein Mikroprozessorsystem (Mikrocomputer) ist eine Datenverarbeitungs- und Steuereinheit. Sie besteht aus:

- ☞ **Mikroprozessor (CPU, Central Processing Unit), Zentraleinheit)**
- ☞ **Speicher (Memory (Read Only, Random Access), Zentralspeicher)**
- ☞ **Ein-/Ausgabe-Bausteinen (Input/Output, I/O)**

<sup>4</sup> Genau genommen wird die CPU nur dann Mikroprozessor genannt, wenn sie in einem einzigen Chip integriert wurde. Dies ist heute fast immer der Fall.

## Der Mikroprozessor ( $\mu P$ )

Der Mikroprozessor ist der Chef (*master*) im System. Er steuert den gesamten Ablauf der Datenverarbeitung und bearbeitet die Daten.

### Beispiel: Der 8080 von Intel

(Quellen: Datenblatt Intel, Wikipedia [http://de.wikipedia.org/wiki/Intel\\_8080](http://de.wikipedia.org/wiki/Intel_8080))

Der Intel 8080 ist ein 1974 eingeführter 8-Bit-Mikroprozessor von Intel. Er wird allgemein als erster vollwertiger Mikroprozessor angesehen. Als Nachfolger des Intel 8008 war sein Befehlssatz zu diesem kompatibel.

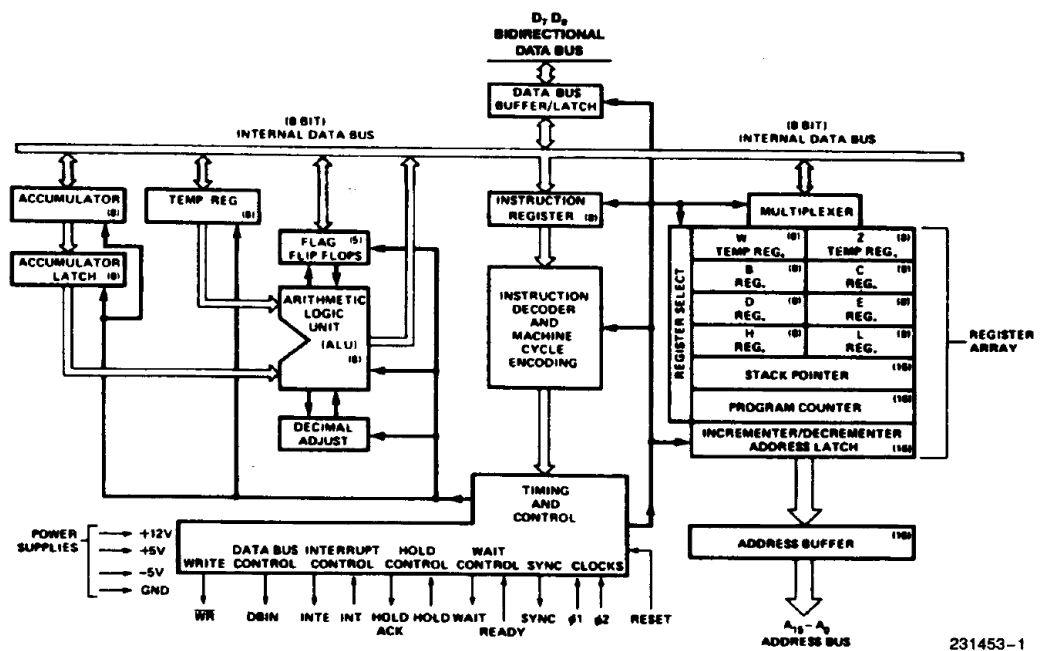


Figure 1. Block Diagram

Das großzügige Design mit 40 Anschlussstiften (Pins) ermöglichte einen Adressbus mit 16 Bit, so dass der 8080 64 KiB Speicher adressieren konnte. Außerdem konnten 256 vom Adressbereich unabhängige Ein-/Ausgabe-Register angeschlossen werden.

Er verfügte über sieben 8-Bit-Register, von denen sechs zu drei 16-Bit-Registern kombiniert werden konnten, sowie einen Stapelzeiger (*Stackpointer*) und einen Programmzeiger (*Program Counter PC*) mit je 16 Bit. Das Betriebssystem CP/M wurde für den 8080 entwickelt und stellte ein knappes Jahrzehnt lang das vorherrschende System für Mikrocomputer dar, ähnlich wie später MS-DOS.

Eingesetzt wurde der 8080 in Steuergeräten (z.B. Marschflugkörpern) und den ersten PCs (u.a. Micral, IMSAI und Altair 8800).

Neben dem eigentlichen Prozessor benötigte der 8080 noch zwei weitere Bausteine: einen separaten Taktgenerator und einen Buscontroller.

1976 brachte die Firma Zilog mit dem Z80 einen 8080-kompatiblen, aber stark erweiterten Prozessor heraus.

Technische Daten:

- Taktfrequenz: 2 MHz
- Anzahl Transistoren: 6.000 bei  $6 \mu$
- Datenbus: 8 Bit
- Adressbus: 16 Bit
- Adressierbarer Speicher: 64 KiB

Auch moderne Prozessoren basieren im Grundaufbau noch immer auf der beim 8080 verwendeten Architektur (siehe Von-Neumann-Architektur).

Moderne Prozessoren arbeiten allerdings mit 64-Bit-Daten und besitzen sogar mehrere interne Kerne, so dass eine echte parallele Verarbeitung der Daten möglich ist, was ihre Verarbeitungsgeschwindigkeit vervielfacht.

Die Arbeitsgeschwindigkeit einer CPU hängt aber auch vom verwendeten rechteckförmigen Taktsignal ab, das alle Abläufe innerhalb des Mikrocomputers steuert. Dieses Signal wird meist von einem in der CPU integrierten Quarzgenerator erzeugt. Der Quarz wird extern angeschlossen. Mit unterschiedlichen Quarzen sind sogar oft unterschiedliche Verarbeitungsgeschwindigkeiten möglich.

## **Der Speicher (Zentralspeicher)**

Der externe Speicher dient der Datenspeicherung, muss schnell sein, und wird direkt von der CPU angesteuert. Langsamere periphere Speicher die eher der Aufbewahrung von Daten dienen (Festplatten, DVDs, Speichersticks ...) werden über spezielle Ein-/Ausgabebausteine (Controller) angesteuert.

## **RAM (Random Access Memory)**

Der größte Teil des Speichers ist schneller **flüchtiger Speicher**. Dieser **Schreib-Lese-Speicher** besteht meist aus hoch integriertem dynamischen RAM und verliert beim Abschalten seinen Inhalt. Er wird zum Zwischenspeichern der Daten während der Verarbeitung benötigt.

## **ROM (Read Only Memory)**

Für Daten, die erhalten werden müssen (Programme, Einstellungsparameter, ...) wird **nichtflüchtiger Speicher** benötigt. Früher war dieser nicht veränderbar (*read only*). Heutige Flash-Speicher lassen sich aber relativ leicht auch beschreiben, was zum Beispiel ein verändern der Firmware<sup>5</sup> eines Gerätes ermöglicht.

## **Ein-/Ausgabe-Bausteine**

Sie sind die Schnittstellen zur Außenwelt. Oft werden sie als Input/Output-Ports (I/O-Ports), Tore (Türen, Pforten) zur Ein- und Ausgabe von Daten, bezeichnet. An ihnen wird die Peripherie (externe Geräte) angeschlossen. Die meisten dieser Bausteine übernehmen einen Teil der Verarbeitung der Daten und entlasten so die CPU. Sie steigern dadurch die Arbeitsgeschwindigkeit des Systems.

---

5 **Firmware** ist die Betriebssystem-Software, die der Hersteller in seinem elektronische Geräten unterbringt.

Wichtige Ein/Ausgabe-Bausteine sind zum Beispiel die seriellen (SIO) und parallelen Schnittstellen (PIO), die Bausteine zur Steuerung der Festplattenspeicher, der Kartenleser, des Monitors usw..

## Adress-, Daten- und Steuerbus

Da Daten und Adressen im Mikroprozessorsystem und auch in der CPU parallel übertragen werden sind viele Leitungen nötig um dies zu erreichen.

**Die Gesamtheit solcher parallelen Leitungen wird als Bus (lat.: omnibus = alle) bezeichnet.**

An einen solchen Bus werden alle Baugruppen des Mikroprozessorsystem (CPU, Speicher und die I/O-Bausteine) parallel angeschlossen, d.h. dass jede Busleitung mit jedem Baustein verbunden wird. Die Leitungen werden dabei von Null an durchnummeriert. Ein 64-Bit Datenbus besitzt also die Anschlüsse D0-D63.

Auf dem **Datenbus** werden die Daten übertragen.

Über den **Adressbus** wird der Baustein ausgewählt, an den die Daten übermittelt werden sollen. Gleichzeitig wird dabei die Adresse übermittelt, wo die Daten z.B. abgespeichert werden sollen.

Der **Steuerbus** gibt dabei z.B. an ob es sich um eine Lese- oder Schreib-Operation handelt.

Adress- und Steuerbus dienen also zusammen der Steuerung des Informationstransports, da auf den Datenbus immer nur ein Sender und ein Empfänger zugreifen dürfen.

**Bemerkung:** Ein Bus verbindet im Gegensatz zur Schnittstelle mehr als zwei Teilnehmer.

## *Vom Mikroprozessor zum Mikrocontroller*

Da die Integrationsdichte von integrierten Bausteinen sich immer weiter erhöht hat, wurde es auf einmal möglich fast alle Komponenten eines Mikrocomputers auf einem Chip zu vereinen.

**Sind die CPU, der Speicher und einige Ein-/Ausgabe-Bausteine in einem Chip vereint, so spricht man von einem Mikrocontroller ( $\mu\text{C}$ , *embedded computer*).**

Mikrocontroller haben einen geringen Platzbedarf, kommen bei geringen Taktfrequenzen mit sehr wenig Energie aus und können in hohen Stückzahlen für wenig Geld produziert werden.

Sie können vielfältige Aufgaben mit unterschiedlicher Komplexität übernehmen und sind heute in fast allen elektronischen Geräten, oft sogar mehrfach, zu finden.

Es existieren Mikrocontroller in vielen Varianten. Auf modernen Mikrocontrollern befinden sich neben den üblichen Pheripheriebausteinen wie z.B. Timer, Taktgeneratoren, EEPROM-Speicher oder serielle Schnittstelle auch oft speziellere Peripherieblöcke mit USB- (*Universal Serial Bus*), I<sup>2</sup>C- (*Inter-Integrated Circuit*), SPI- (*Serial Peripheral Interface*), CAN- (*Controller Area Network*), LIN- (*Local Interconnect Network*), oder Ethernet-Schnittstellen. Auch LCD-Controller und -Treiber, PWM-Ausgänge (Puls-Weiten-Modulation) oder hochauflösende Analog-Digital-Wandler (mit 8 bis 24 Bit Auflösung und bis zu 16 Kanälen) sind oft verfügbar.

## CISC und RISC

### CISC (Complex Instruction Set Computing<sup>6</sup>)

**CISC** bezeichnet die klassischen Befehlssätze von Mikroprozessoren. Ein CISC-Befehlssatz zeichnet sich durch viele verhältnismäßig leistungsfähige Einzelbefehle unterschiedlicher Länge aus. CPUs mit CISC-Befehlssatz sind in der Regel mikroprogrammiert, das heißt, dass das im Steuerwerk enthaltene Mikroprogrammwerk eine Menge von Mikroprogrammen (Mikrocode, Liste von Steuersignalen) enthält. In diesen Mikroprogrammen ist verschlüsselt wie der Prozessor die Befehle ausführt.

Die Entwicklung von Mikroprogrammen ist zeitaufwändig und fehleranfällig. Sie bot früher aber einen Zeitgewinn bei der Entwicklung von kürzerem Maschinencode. Auch konnte so Zentralspeicher eingespart werden, der zur Anfangszeit der Computertechnik sehr teuer war.

Wird bei modernen CPUs Mikrocode verwendet, so ist dieser oft veränderbar, wodurch der Hersteller Bugfixes einspielen kann.

Bei dem CISC Modell hat die CPU üblicherweise wenige Arbeitsregister und viele Adressierungsarten. Klassische CISC-Prozessoren sind zum Beispiel die 808x und 80x86-Serie von Intel, die 680x0 von Motorola sowie der Z80 von Zilog. Ab den Prozessoren 80486, Pentium und 68060 wurden Elemente der RISC-Prozessoren mit eingebaut.

Neue Prozessoren sind kaum noch mikroprogrammiert. Es hat sich herausgestellt, dass Programmierer und Compiler den umfangreichen Befehlssatz nur zu einem kleinen Teil nutzen. Auch billiger schneller Speicher mit großer Kapazität macht eine Reduktion des Code nicht mehr notwendig.

Ab dem Pentium Pro verfügen die Intel-Prozessoren über eine vorgeschaltete Funktionseinheit, welche komplexe Befehle in RISC-Befehle übersetzt.

---

<sup>6</sup> Rechnen mit komplexem Befehlssatz

## RISC-Prozessoren (*Reduced Instruction Set Computing*<sup>7</sup>)

Bei RISC-Prozessoren werden die Befehle meist direkt durch die Hardware implementiert. Dies ermöglicht eine hohe Ausführungsgeschwindigkeit da der Decodierungsaufwand auf Seiten der CPU gering ist. Man versucht auf komplexe Befehle konsequent zu verzichten, und somit wird kein Mikrocode verwendet.

Schnelle kurze Befehle erlauben es dem Prozessor schneller auf Unterbrechungen (*interrupts*) zu reagieren, und so zum Beispiel schneller auf externe Signale (oft Sensoren) zu reagieren. Da dies die Hauptaufgabe von Mikrocontrollern ist, werden diese daher meist als RISC-Prozessor ausgeführt.

Wie erwähnt sind die Befehle beim RISC-Prozessor fest verdrahtet. Das ist der Grund warum erste RISC-Prozessoren einen stark reduzierten Befehlssatz enthielten. Bei neuen Controllern werden allerdings auch komplexere Befehle integriert, was den Befehlssatz vervielfacht (Bsp.: PIC 33 Befehle, ATmega8A 130 Befehle).

**Eigenschaften von RISC-Prozessoren** ([http://de.wikipedia.org/wiki/Reduced\\_Instruction\\_Set\\_Computing](http://de.wikipedia.org/wiki/Reduced_Instruction_Set_Computing)):

- Nur die Befehle LOAD und STORE greifen auf den Speicher zu, die restlichen Befehle arbeiten mit Registeroperanden.
- Dadurch, dass ein Speicherzugriff immer einen expliziten Befehl benötigt, erhöht ein großer Registersatz, d. h. viele allgemeine Register (*general purpose register*, Arbeitsregister), die Effizienz, da Zwischenergebnisse dann lokal vorgehalten werden können.
- RISC-CPU's sind so ausgelegt, dass sie meist nur einen Takt benötigen um einen Befehl abzuarbeiten. Dies wird unter anderem durch „*pipelining*“ erreicht. Abweichungen gibt es nur bei LOAD, STORE, und Verzweigungsbefehlen.
- RISC-CPU's stellen weniger Befehle zur Verfügung als CISC-CPU's. Diese Befehle besitzen in der Regel alle die gleiche Codebreite und sind registerorientiert.

**Bemerkung:** CISC-Prozessoren benutzen oft Elemente des RISC-Design. Umgekehrt gibt es auch viele RISC-Prozessoren, welche zusätzlich komplexe Befehle integrieren (z.B. der Multiplikationsbefehl bei AVR<sup>®</sup>-Controllern). Geht es um Geschwindigkeit, ist der RISC-Design erste Wahl.

## Die "Von Neumann-Architektur" und die "Havard-Architektur"

**Von Neumann-Architektur** (<http://de.wikipedia.org/wiki/Von-Neumann-Architektur>)

In den ersten Rechnern war ein festes Programm entweder hardwaremäßig verdrahtet, oder es konnte mit Lochkarten eingelesen werden. Der Mathematiker John von Neumann begründete 1945 die nach ihm benannte Architektur, mit der es möglich wurde,

<sup>7</sup> Rechnen mit reduziertem Befehlssatz

Änderungen an Programmen sehr schnell durchzuführen und ganz verschiedene Programme ablaufen zu lassen, ohne Veränderungen an der Hardware vornehmen zu müssen.

Dieser universelle Von-Neumann-Rechner blieb bis heute zum größten Teil erhalten und basiert auf den im Kapitel "Aufbau eines Mikroprozessorsystems" gesehenen Komponenten:

- **Rechenwerk** (ALU, CPU).
- **Steuerwerk** (*control unit*, Taktgeber, Befehlszähler)
- **Speicher** (*memory*) speichert sowohl Programme als auch Daten
- **Eingabe-/Ausgabe-Einheit**
- **Bus-System**

Bei der Von-Neumann-Architektur werden im Speicher sowohl Programme wie auch Daten abgelegt.

Der Von-Neumann-Rechner arbeitet sequentiell d.h. der nächste Befehl kann erst verarbeitet werden, wenn die Verarbeitung des ersten Befehls abgeschlossen ist. Er arbeitet nach folgenden Regeln:

### Prinzip des gespeicherten Programms:

- Bei der Von-Neumann-Architektur sind die Befehle in einem RAM-Speicher mit durchgehendem (linearem, eindimensionalem) Adressraum abgelegt.
- Ein Befehls-Adressregister (Befehlszähler, Programmzähler) zeigt auf den momentan auszuführenden Befehl.
- Befehle können so geändert werden wie Daten.

### Prinzip der sequentiellen Programmausführung:

- Befehle werden aus einer Zelle des RAM-Speichers gelesen, dekodiert und dann ausgeführt (*fetch, decode, execute*).
- Erfolgt keine Verzweigung, so wird der Inhalt des Befehlszählers um den Wert Eins erhöht (inkrementiert).
- Es existieren ein oder mehrere Sprung-Befehle, die den Inhalt des Befehlszählers um einen anderen Wert als +1 verändern.
- Es existieren ein oder mehrere Verzweigungsbefehle, die in Abhängigkeit vom Wert eines Entscheidungsbits den Befehlszähler inkrementieren oder einen Sprung-Befehl ausführen.

Der Nachteil des "Von-Neumann-Rechners" ist der Flaschenhals für die Daten, der zwischen CPU und Speicher auftritt. Seit Mitte der 90er Jahre des 20. Jahrhunderts stieg die Geschwindigkeit der CPUs schneller als die der verwendeten Speicherbausteine (RAM) und der übertragenden Busse. Die CPU wurde nicht mehr ausreichend schnell mit Daten

versorgt. In der Praxis versucht man diesen Effekt durch die Nutzung von Datencaches abzuschwächen.

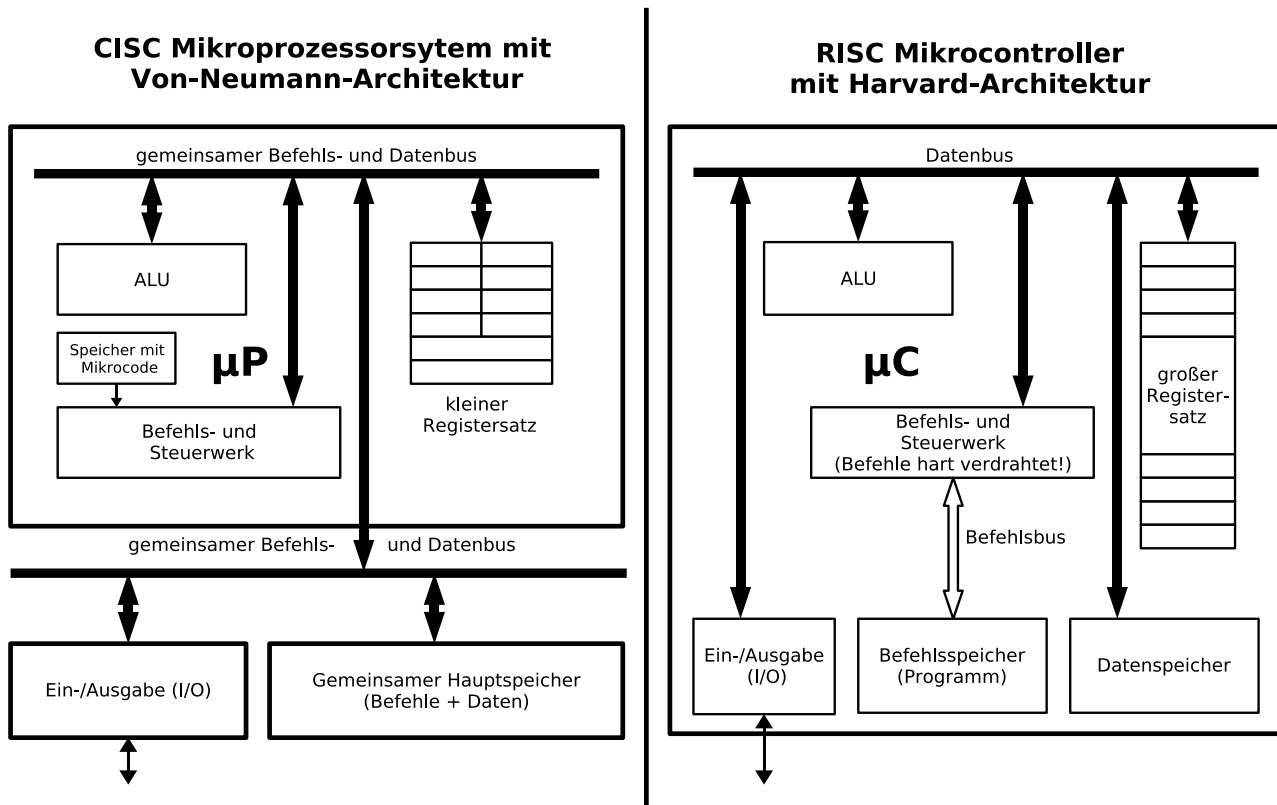
## Harvard-Architektur (<http://de.wikipedia.org/wiki/Harvard-Architektur>)

**Bei der Harvard-Architektur sind Befehlsspeicher und Datenspeicher physisch voneinander getrennt und beide werden über getrennte Busse angesteuert.**

### Vorteile der Harvard-Architektur:

- Befehle und Daten können gleichzeitig geladen bzw. geschrieben werden. Bei der Von-Neumann-Architektur sind dazu mindestens zwei aufeinander folgende Buszyklen notwendig. Mit der Harvard-Architektur lassen sich besonders schnelle CPUs, Mikrocontroller und Signalprozessoren realisieren.
- Moderne Prozessoren in Harvard-Architektur sind in der Lage, parallel mehrere Rechenwerke gleichzeitig mit Daten und Befehlen zu füllen.
- Durch die Trennung von Befehl- und Datenspeicher kann die Datenwortbreite und Befehlswortbreite unabhängig festgelegt werden. Dadurch kann z.B. die Effizienz des Programmspeicherbedarfs in Mikrocontroller-Systemen verbessert werden, da sie nicht direkt von den Datenbusbreiten abhängig ist, sondern ausschließlich vom Befehlssatz.
- Die gefürchteten Pufferüberläufe, die für die meisten Sicherheitslöcher in modernen Systemen verantwortlich sind, werden bei stärkerer Trennung von Befehlen und Daten besser beherrschbar.

Die Harvard-Architektur wird überwiegend in RISC-Prozessoren konsequent umgesetzt.



**Bemerkung:** Die Bezeichnung Harvard-Architektur wird auch auf Prozessoren in herkömmlichen von-Neumann-Systemen angewendet, deren Prozessoren zwar einen gemeinsamen Hauptspeicher für Code und Daten verwenden, diese jedoch in unterschiedlichen Caches platzieren.

## ***Aufbau eines Mikrocontrollersystems***

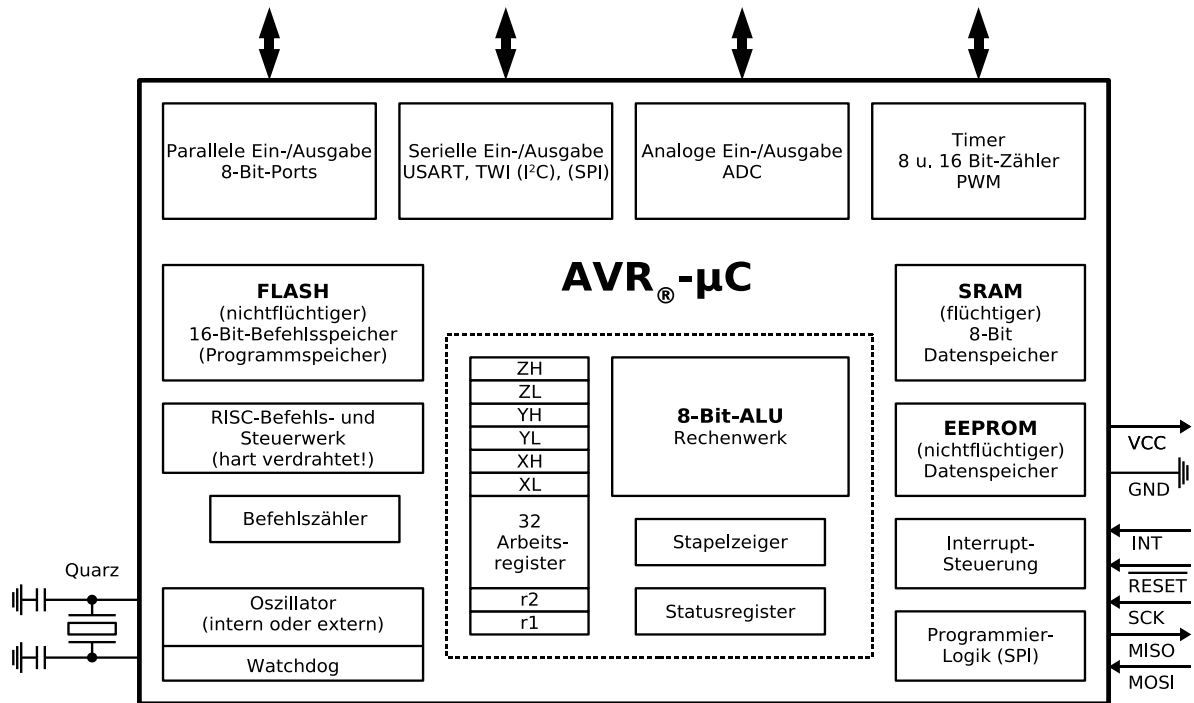
Außer dem Mikrocontroller werden im Mikrocontrollersystem nicht unbedingt weitere Komponenten benötigt. Reicht der interne Oszillator nicht aus, so kann ein externer Quarz angeschlossen werden. Auch ist es natürlich möglich den Speicher durch externe Bausteine zu vergrößern oder zusätzliche Schnittstellenbausteine zum Anschließen der Peripherie zu verwenden.

Die meisten Mikrocontroller können im System, d.h. ohne den Baustein aus seiner Schaltung zu entfernen, programmiert werden (ISP<sup>8</sup>). Dazu wird meist ein externes Programmiergerät verwendet, das mit der parallelen, seriellen oder USB-Schnittstelle eines PC verbunden wird.

Ein Teil des internen Mikrocontrollerspeichers (Flash) kann auch für ein so genanntes "Bootloader-Programm" verwendet werden. Das Programmiergerät ist dann nicht mehr unbedingt erforderlich. Der Mikrocontroller kann dann über einen Schnittstellenbaustein direkt mit der Schnittstelle des PC verbunden werden.

## Der Mikrocontroller ( $\mu\text{C}$ )

Der klassische Mikrocontroller besitzt ein schnelles RISC-Design und ist in der Harvard-Architektur aufgebaut. Dies trifft auch auf die 8-Bit-AVR<sup>®</sup>-Familie der Firma ATMEL<sup>®</sup> zu.



Im Gegensatz zum Computer sind beim Mikrocontrollersystem alle Blöcke der zentralen Verarbeitungsanlage in einem IC integriert.

Im folgenden soll der Aufbau eines Controllers anhand von Bausteinen der AVR<sup>®</sup>-Familie untersucht werden. Im Anhang befinden sich die beiden Blockschaltbilder des ATmega8A und des ATmega32A.

### Wir unterscheiden folgende Blöcke:

**Speicher:** Der Speicher teilt sich in nichtflüchtigen<sup>9</sup> FLASH-Befehlsspeicher (16-Bit) und EEPROM-Datenspeicher (8-Bit) sowie in flüchtigen<sup>10</sup> SRAM-Datenspeicher (8-Bit) auf. Im nächsten Kapitel wird detailliert auf den Speicher eingegangen.

**CPU:** Sie besteht aus dem 8-Bit-Rechenwerk (ALU) mit Statusregister, dem Stapelzeiger und 32 8-Bit Arbeitsregistern (Arbeitsspeicher). Sechs dieser Register (r26-r31) können als Doppelregister (16-Bit) X, Y und Z adressiert werden. Die Arbeitsregister befinden sich im flüchtigen SRAM; ihr Inhalt bleibt also nicht erhalten.

<sup>9</sup> Der Inhalt des Speichers bleibt nach dem Ausschalten erhalten. Nichtflüchtiger Speicher kann mittels des Programmiergeräts beschrieben werden.

<sup>10</sup> Der Inhalt ist nach dem Einschalten unbestimmt! Das bedeutet, dass jede Speicherzelle nach dem Einschalten einen beliebigen Wert zwischen  $0$  und  $0xFF$  enthalten kann.

- Befehlswerk:** Das RISC-Befehls- und Steuerwerk mit dem Befehlszähler (Programmzähler PC) arbeitet die meist 16-Bit breiten Befehle oft in einem einzigen Takt ab. Das bedeutet, dass mit einem 16 MHz-Quarz fast 16 MIPS (*Million Instructions per Second*) verarbeitet werden können. Das ist ungefähr soviel wie ein Intel 80386 Prozessor um 1990 schaffte.
- Ein-/Ausgabe:** Die Peripherie wird über die Ein-/Ausgabe-Einheiten angesteuert. Je nach Größe des Controllers sind mehr oder weniger der folgenden Funktionen enthalten:
- 1-4 parallele digitale Ein- und Ausgänge (8-Bit-Ports)
  - (mehrere) 8- und 16-Bit-Timer
  - RTC (*Real Time Clock*) mit eigenem Quarz
  - mehrere PWM (*Pulse Wide Modulation*)-Ausgänge
  - serielle asynchrone Schnittstelle (EIA232 (RS232) bzw. V.24) mit dem UART (*Universal Asynchronous Receiver and Transmitter*)
  - serielle synchrone Schnittstelle mit dem USART (*Universal Synchronous and Asynchronous serial Receiver and Transmitter*)
  - eine serielle Master/Slave-Schnittstelle SPI (*Serial Peripheral Interface*) die oft zur Programmierung des FLASH benutzt wird
  - eine serielle Schnittstelle TWI (*Two-Wire serial Interface*) die einer I<sup>2</sup>C-Schnittstelle entspricht, aber wahrscheinlich aus rechtlichen Gründen anders genannt wird.
  - mehrkanalige (8) 10-Bit Analog/Digital-Wandler
- Oszillator:** Der Controller kann ohne externen Quarz mit seinem internen kalibrierten RC-Oszillator mit 1, 2, 4 oder 8 MHz betrieben werden. Mit externem Quarz sind 16MHz möglich. Sollen Timer nicht mit der Betriebsfrequenz laufen, so kann ein zusätzlicher Quarz für die Timerfunktionen angeschlossen werden. Ein programmierbarer Watchdog-Timer (Wachhund) mit eigenem integriertem RC-Oszillator verhindert auf Wunsch, dass ein Hängenbleiben des Programms nicht erkannt wird.
- Interrupts:** Eine vielfältige Interruptsteuerung mit externen und internen Interrupts ist möglich. Auch kann der Controller in bis zu sechs unterschiedliche Schlafmodi versetzt werden.
- Prog.-Logik:** Über eine sechs adrige SPI-Schnittstelle kann man den Controller "*in-system*<sup>11</sup>" elektrisch programmieren und wieder löschen (FLASH, EEPROM, *Fuse- und Lock-Bits*<sup>12</sup>).

Alle Mikrocontroller der AVR<sup>®</sup>-Familie sind gleich aufgebaut. Sie unterscheiden sich hauptsächlich in der Größe des Speichers und ihrer Ausstattung (Anzahl der Ports, Anzahl der Timer, ADC, RTC ...).

<sup>11</sup> "Im System": Ohne dass der Chip aus der Schaltung ausgebaut werden muss!

<sup>12</sup> Mit diesen programmierbaren Bits kann zum Beispiel der Controller fürs Auslesen gesperrt werden oder der externe Quarzoszillator aktiviert werden (siehe später).

Hier eine kleine Auswahl von Mikrocontrollern der Firma ATMEL.

Device	Flash (KiB)	EEPROM (KiB)	SRAM (Byte)	Max I/O Pins	F.max (MHz)	16-bit Timers	8-bit Timer	PWM (ch.)	RTC	UART	I2C	ISP	10-bit A/D (ch.)	Analog Comp.	Watch-dog	On Chip Osc.	Interrupts	Ext. Int.	Self Progr.
ATtiny11	1	--	--	6	6	--	1	--	--	--	--	--	--	Yes	Yes	Yes	4	1	--
ATmega8A	8	0.5	1024	23	16	1	2	3	Yes	1	Yes	Yes	8	Yes	Yes	Yes	18	2	Yes
ATmega8535	8	0.5	512	32	16	1	2	4	--	1	Yes	Yes	8	Yes	Yes	Yes	20	3	Yes
ATmega16A	16	0.5	1024	32	16	1	2	4	Yes	1	Yes	Yes	8	Yes	Yes	Yes	20	3	Yes
ATmega32A	32	1	2048	32	16	1	2	4	Yes	1	Yes	Yes	8	Yes	Yes	Yes	19	3	Yes
ATmega64	64	2	4096	54	16	2	2	8	Yes	2	Yes	Yes	8	Yes	Yes	Yes	34	8	Yes
ATmega128	128	4	4096	53	16	2	2	8	Yes	2	Yes	Yes	8	Yes	Yes	Yes	34	8	Yes

## Weitere Eigenschaften von Controllern der AVR®-Mikrocontroller Familie:

- Systemtakt bis 20 MHz
- Betriebsspannungsbereich von 1,8 V bis 5,5 V
- Sechs unterschiedliche Gehäuseformen
- Erweiterter Befehlssatz mit Hardware- Multiplikationsbefehlen
- Neu- und Umprogrammierung während des Betriebs (Bootloader)
- JTAG-Interface zum Anschluss von Testsystemen (Debuggen)

**Bemerkung:** Anfang 2008 erschienen die neue ATxmegas der AVR®-Familie. Vorteile der ATxmegas ist die höhere Taktfrequenz bis 32 MHz, der geringere Stromverbrauch, *Direct Memory Access* (DMA), eine Crypto-Engine für AES und DES und ein neues *Event*-System. Es gibt allerdings keine ATxmegas in DIL-Bauform und die maximale Spannung liegt bei 3,6 V.

## ***Der interner Speicher des ATmega32A***

### **Der Programm- oder Befehlspeicher (ROM, Flash-EPROM<sup>13</sup>)**

Der Flash-EPROM (Flash) lässt sich vom Anwender mittels eines Programmiergeräts "*in-system*" elektrisch programmieren und wieder löschen. Der Inhalt des Flash bleibt nach dem Abschalten erhalten und so stehen die programmierten Befehlssequenzen und die im Programm enthaltenen konstanten Daten gleich nach dem Einschalten zur Verfügung. ATMEL® garantiert, dass der Flash mehr als 10000mal beschrieben werden kann.

<sup>13</sup> Erasable Programmable Read-Only Memory

Die Befehle sind alle 2 Byte (1 Wort) groß<sup>14</sup> und werden meist in einem einzigen Taktzyklus abgearbeitet.

**Der Programmspeicher (ROM, Flash) wird wortweise (2 Byte) adressiert! Die Flash-Adressen sind also Wort-Adressen!**

Nach dem Einschalten beginnt das Programm mit dem ersten Befehl auf der Adresse **0x0000**. Hier befindet sich meist ein Sprungbefehl zur eigentlichen ersten Adresse des Programms, da sich ab **0x0000** die Einsprungadressen für Interrupt-Routinen befinden.

Konstante Programmdateien können auch im Flash abgelegt werden und hier ist dann sogar eine byteweise Adressierung über Indexregister möglich.

Beim ATmega32A beträgt der Flash 32 KiB, also 16Ki-Worte!

Beim ATmega8A sind es 8 KiB also nur 4 Ki-Worte.

- ☞ **A100**
- a) Wie viel Bit muss der Adresszähler besitzen um die 32 KiB des ATmega32A zu adressieren? Berechne die höchste Adresse im Speicher (Bezeichnung "**FLASHEND**"<sup>15</sup>).
  - b) Wie viel Bit muss der Adresszähler besitzen um die 8 KiB des ATmega8A zu adressieren? Berechne die höchste Adresse im Speicher (**FLASHEND**).

## Der Arbeits- oder Datenspeicher

Der Arbeitsspeicher teilt sich in einen flüchtigen schnellen SRAM-Speicher und in einen nichtflüchtigen EEPROM<sup>16</sup>-Speicher auf. Im EEPROM bleiben die Daten auch nach dem Abschalten erhalten. Beide werden im Gegensatz zum Programmspeicher byteweise adressiert.

**Der Datenspeicher (RAM, SRAM) wird byteweise adressiert! Die RAM-Adressen sind also Byte-Adressen!**

<sup>14</sup> Einige Ausnahmen benötigen mehr als 16 Bit und werden auf mehrere Adressen aufgeteilt.

<sup>15</sup> Da man nicht alle wichtigen Adressen als Zahlenwert behalten kann, und diese Werte auch noch von Controller zu Controller unterschiedlich sind wurden sie mit Namen versehen. Die Zuweisung Name = Adresse erfolgt in der jeweiligen Definitionsdatei "**\*.inc**" des Controller (siehe später).

<sup>16</sup> *Electrically Erasable Programmable Read-Only Memory*

## Programmspeicher (Flash) wortweise Adressierung

Adresse:

0x0106		
0x0105		
0x0104		
0x0103		
0x0102	0xBA	0x98
0x0101	0x76	0x54
0x0100	0x32	0x10

## Datenspeicher (SRAM) byteweise Adressierung

Adresse:

0x0106	
0x0105	0xBA
0x0104	0x98
0x0103	0x76
0x0102	0x54
0x0101	0x32
0x0100	0x10

Werden Worte (2 Byte) im RAM abgelegt, so steht das niederwertige Byte (LByte) an der niedrigen Adresse, das höherwertige Byte (HByte) an der höheren Adresse.

Diese Art der Adressierung wird auch Little-Endian (Klein-Endender zuerst) oder Intel-Format genannt.

## Das statische SRAM (Static Random Access Memory)

Der statische Schreib/Lese-Speicher ist flüchtig. Nach dem Abschalten sind die Daten verloren. Beim Einschalten ist sein Inhalt undefiniert! Er dient dazu die veränderlichen Daten während dem Betrieb aufzunehmen.

### Die Arbeitsregister (*r0-r31*)

Auf den ersten 32 Adressen (**0x00-0x1F**) des SRAM liegen die **32 Arbeitsregister (r0-r31)**. Mit ihnen lassen sich alle arithmetische und logische Operationen über die ALU<sup>17</sup> durchführen. Register 0 bis 15 (**r0-r15**) erlauben leider keine unmittelbare Adressierung und werden deshalb weniger oft verwendet als die Register 16 bis 31 (**r16-r31**). Register 24 bis 31 lassen sich zusätzlich auch als 16-Bit-Register (Doppelregister) verwenden. Register 26 bis 31 können als Adresszeiger (*Pointer*) bei der indirekten Adressierung eingesetzt werden und werden mit **X (r27:r26)**, **Y (r29:r28)** und **Z (r31:r30)** bezeichnet.

### Die SF-Register (*special function register, Sonderfunktions-Register*)

Dann folgen die **64 "Special-Funktion"-Register (SF-Register)**. Sie dienen dazu die interne und externe Peripherie anzusteuern (zum Beispiel: **PORTD**) und zu initialisieren. Einzelne SF-Register sind dabei Datenregistern (I/O-Register), andere dienen als Steuerregister<sup>18</sup>. Sie befinden sich auf den Adressen **0x20-0x5F** (siehe Anhang). Um die SF-Register zu adressieren benutzt man am einfachsten die von ATMEL® in den Definitionsdateien angegebenen Bezeichner.

<sup>17</sup> Arithmetic Logic Unit (Arithmetisch-Logische-Einheit)

<sup>18</sup> Leider werden im Datenblatt öfters alle SF-Register als I/O-Register bezeichnet, was zu Verwirrungen führen kann.

## **Der Stapelspeicher**

Auf der höchsten Adresse des SRAM-Speichers (**RAMEND** in den Definitionsdateien) wird der Stapel zur Rettung von Variablen und Rücksprungadressen beim Aufruf von Unterprogramm- und Interrupt-Routinen initialisiert (Kapitel Stapelspeicher). Der Stapel läuft dann zu niedrigeren Adressen hin und darf die anderen Daten nicht überschreiben.

Beim ATmega32A hat der SRAM-Speicher eine Größe von 2 KiB (ohne die 32 Arbeitsregister und die 64 SF-Register!!)

Beim ATmega8A ist es 1 KiB.

- ☞ **A101** Berechne die höchste SRAM-Adresse (**RAMEND**) für den ATMEGA32A und für den ATmega8A.

## **Der nichtflüchtige Speicher (EEPROM)**

Das EEPROM lässt sich entweder über das Programmiergerät oder während des laufenden Betriebs programmieren. Es kann mindestens 100000mal beschrieben werden. Im Betrieb geschieht dies über eine Steuereinheit und ist dementsprechend langsam. Der EEPROM-Speicher eignet sich also nicht als Arbeitsspeicher. In ihm werden feste Steuerwerte und Parameter aufbewahrt.

Beim ATmega32A hat der EEPROM-Speicher eine Größe von 1 KiB!

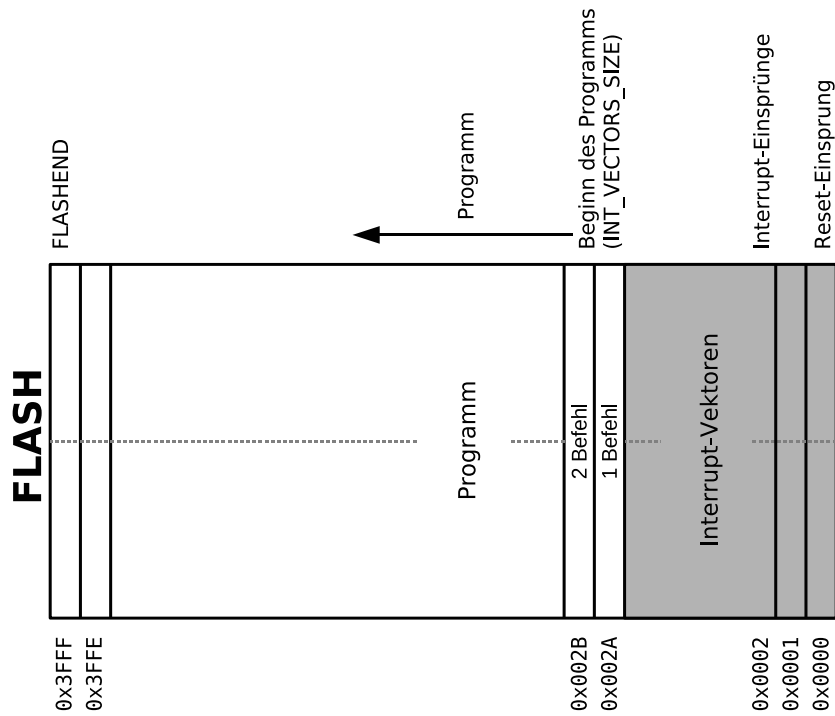
Beim ATmega8A sind es 512Byte.

- ☞ **A102** Berechne die höchste EEPROM-Adresse (**EEPROMEND**) für den ATMEGA32A und für den ATmega8A.

## Zusammenfassung:

# ATmega32

**Programmspeicher (32KiB)**  
nichtflüchtiger Speicher, 16 Bit breit



**Datenspeicher**  
8 Bit breit

