Rechnerarchitektur

Zusammengetragen vom Marc Landolt

1 Die verschiedenen Betrachtungsebenen

Rechnerebene
Hauptblockebene
Registertransferebene
Schaltwerkebene
Ebene elektrischer Schaltungen
Physikalische Ebene

2 Blockschaltbild eines Rechners

Für detaillierte Betrachtungsweisen nehme man das Schema eines Mainboards zur Hand:
z.B. http://www.eserviceinfo.com/downloadsm/4015/Intel_810.html

Computer

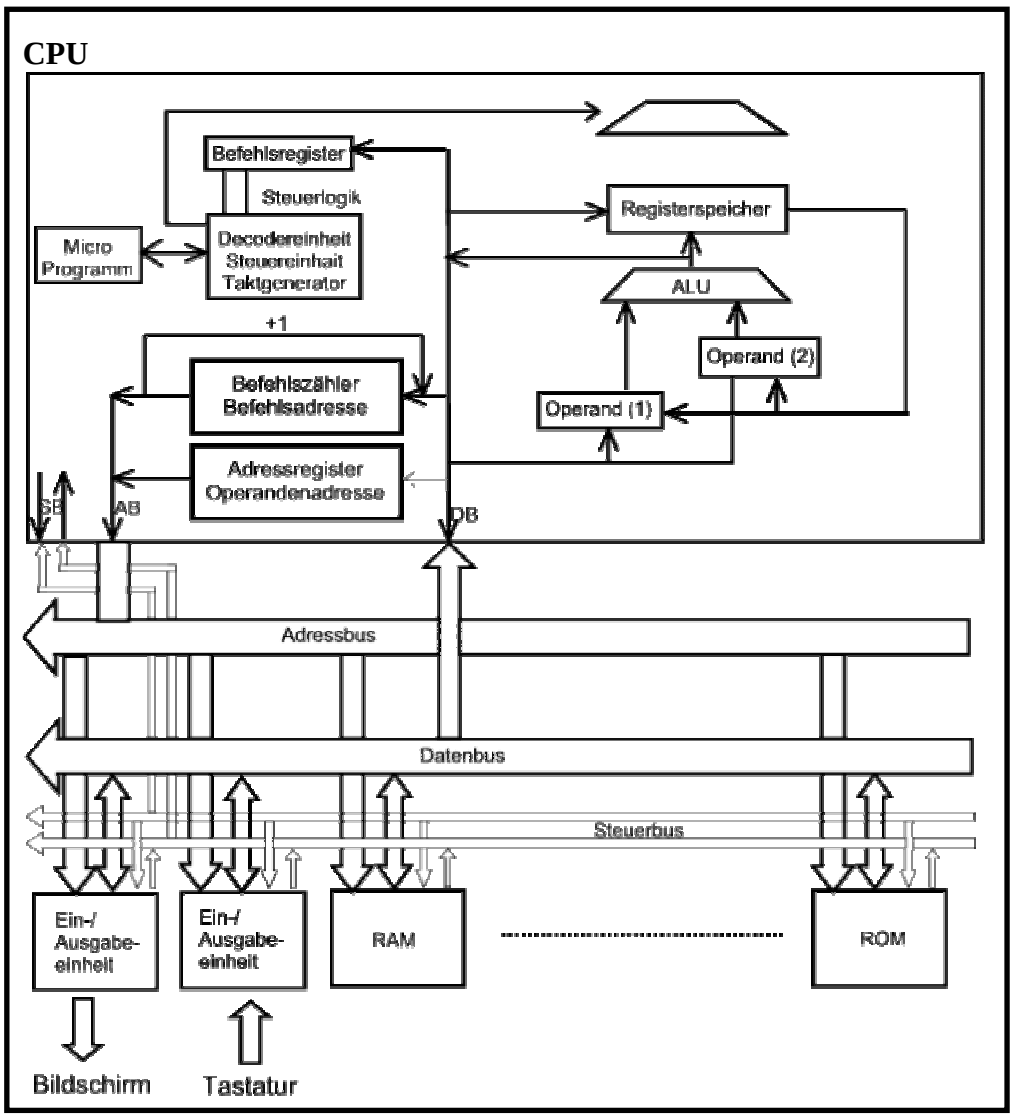


Abbildung 1[1]

[1] http://www.bergers.co.at/sites/hardware/pic/cpu_big.gif

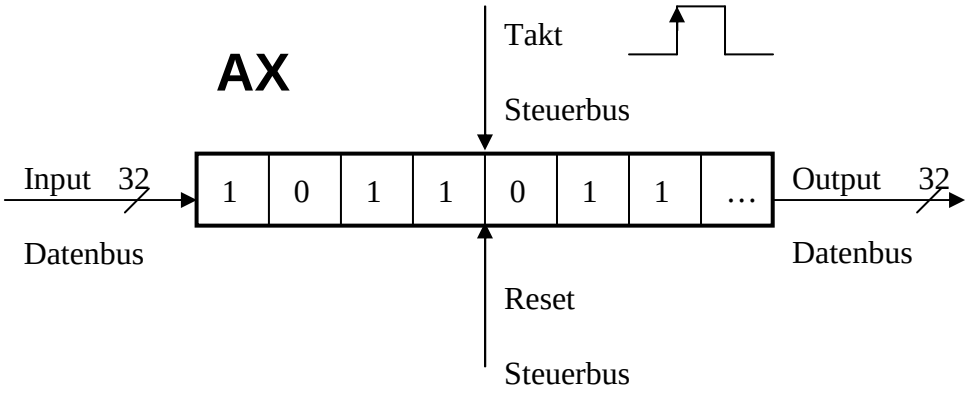
3 CPU

Die Zentrale Recheneinheit (**C**entral **P**rocessing **U**nit) ist der zentrale Ort des geschehenes. Sie besteht aus Silizium auf welchem verschiedene Funktionen „eingebrannt“ sind :

ALU
Befehlsregister
Statusregister
Weitere Register

3.1 Das Register

Das Register besteht aus Flip-Flops in unserem Fall Vorderflanken gesteuerten D-Flip-Flops. Es können darin Binärworte gespeichert werden. Im Allgemeinen sind sie so breit wie die Architektur der CPU. Entsprechend der Busbreite der CPU wird die Busbreite des Systems gewählt. Ausnahme war 386 AX, dort wurde, da die Industrie noch auf 16 Bit eingerichtet war z.B. das CL Register auf den Bus gegeben, das heisst intern wurde mit 32 Bit gerechnet, jedoch musste, sobald der Bus des Systems angesprochen werden sollte auf 16 Bit „umrechnet“ werden. Die Vermutung liegt nahe, dass diese suboptimale Lösung nicht die eines Technikers sondern eines „Marketing-Fritzen“ war.



Es können Werte vom Datenbus übernommen werden, bzw. auf den Datenbus gelegt werden. Dies funktioniert einfach gesagt so:
Eingabe: Im Grunde stehen die Werte des Datenbusses immer am Register an, da dies die Werte aber nur dann übernimmt, wenn der Takt gerade von Null auf Eins springt, wird nur dem Register den **Takt** geben, welches den Wert übernehmen soll.
Mit dem **Reset** wird das Register auf einen festen Wert zurückgesetzt.
Weiter ist zu sagen, dass aus Gründen des Timings meist nicht die Taktleitung verwendet wird, sie sollte mit möglichst wenig Logik ausgestattet sein. Stattdessen wird über eine zusätzliche **write enable** Leitung geregelt. Der Wert muss eine bestimmte Zeit anliegen, vor dem Lesen muss der wert die so genannte **setup time** anstehen, dann wenn das Register die Werte zu übernehmen beginnt muss das Signal für die **hold time** anstehen. Wonach der Wert nach einer typischen Zeit, dem **propagation delay**, am Ausgang ansteht.
Um nun, sagen wir mal 4, Register mit der Breite von 1 Bit so miteinander zu vernetzen, braucht es bereits n(n+1) Verbindungen, vor jedes der n Register wird ein Multiplexer so geschaltet, dass er alle Ausgänge der Anderen Register wählen kann, so entstehen dann bei 4 Register 20 Leitungen. Um das ganze mit weniger Leitungen machen zu können hängt man an jeden Registerausgang ein 3-state Gatter, welches folgende zustände annehmen kann:

EN	IN	OUT
0	0	Hochohmig
0	1	Hochohmig
1	0	0
1	1	1

Die Ausgänge der 3-state Schaltungen gehen dann auf eine Busleitung wo ein anderes Register dann den Wert wieder holen kann, dies geschieht über eine Write Enable (we)

Intermezzo: Tri-State

Ein 3-State Schaltung ist im Grunde eine Gegentakt Schaltung, mit einer Logik. Sie hat 3 Zustände: Hochohmig, wenn das EN auf Null ist, sonst hat sie den Wert des Eingangs (IN). Dabei ist zu beachten, dass wenn aus Versehen zwei 3-State gleichzeitig auf den selben Bus einen unterschiedlichen Wert geben zerstört dies den oberen Transistors Q1 des 3-States mit dem Wert 1 und/oder Q2 des 3-States mit dem Wert Null.

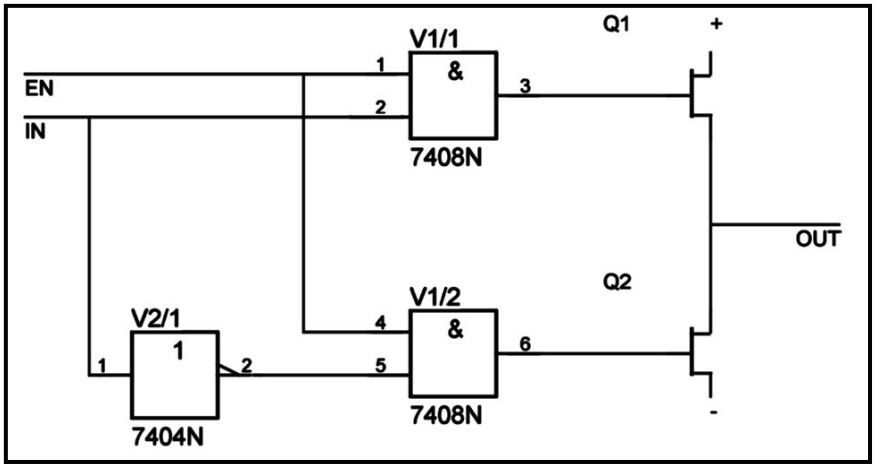


Abbildung 2[2]

Jedes Register hat einen eineindeutingen (Bijektiv) Namen und sind somit anders als das RAM direkt ansprechbar, z.B. der Befehl Mov AX, FFh füllt das AX mit Einsen. Die ALU (Arithmetische Logische Einheit) kann nur mit den Werten in den Registern rechnen. Z.B. ADD AX, BX addiert den Inhalt des BX-Registers zum AX Register hinzu. Somit stehen die Register im Brennpunkt des Geschehens.

[2] genaueres unter: http://www.princeton.edu/~wolf/modern-vlsi/Overheads/CHAP7-2/sld032.htm (11.2006)

3.2 Die ALU (Arithmetic Logic Unit = Arithm. Logische Einheit)

Die ALU entspricht dem Rechenwerk (Zumindest im ersten Semester) es kann arithmetische und logische Operationen durchführen:

ADD	addieren	(Arithmetische Operation)
SUB	subtrahieren	(Arithmetische Operation)
MUL	multiplizieren	(Arithmetische Operation)
AND	Logisches Und	(Logische Operation)
XOR	Exklusiv Oder	(Logische Operation)
…		

Das Resultat wird im so genannten Akkumulator (Accu) gespeichert und zusätzlich wird das **Statusregister (F = Flagregister)** gesetzt:

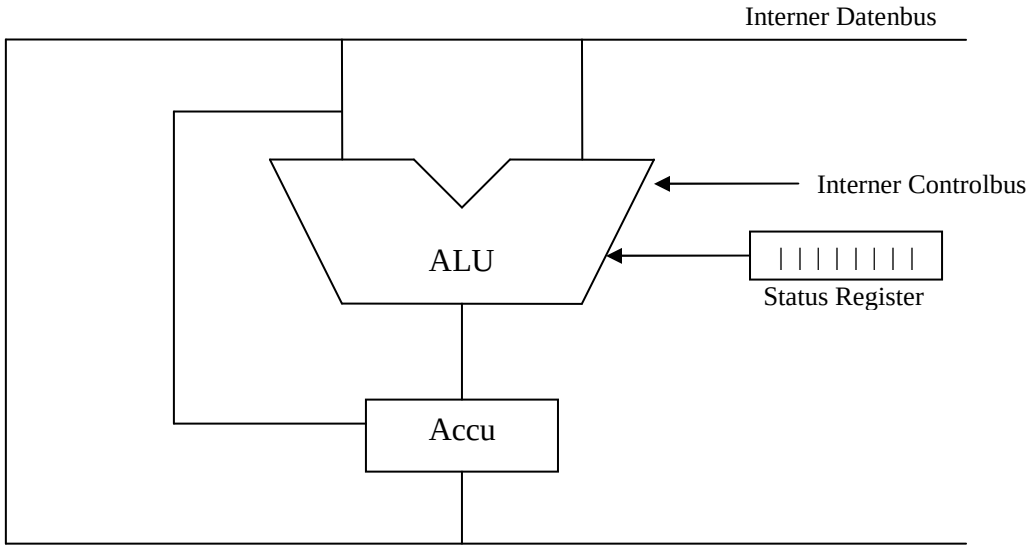
DIV 0	Division durch Null
OF	Overflow
C	Carry (Übertrag) Flag
Z	Zero-(Null) Flag
…	

bei 8Bit zum Beispiel gibt FFh + 01h = 100h, da aber nur 2 Stellen vorhanden sind wird es als 00h angezeigt und das C-Flag gesetzt, welches dann mit JC (**J**ump if **C**arry) ausgewertet werden kann, in dem man z.B. in die Fehlerbehandlungsroutine springt, beziehungsweise das Carry Flag als zusätzliche Stelle betrachtet:

FFh + FFh =	1FEh	=	1 FEh ↑ AX (oder auch BX oder sonst ein Register) Carry Flag (9. Bit)

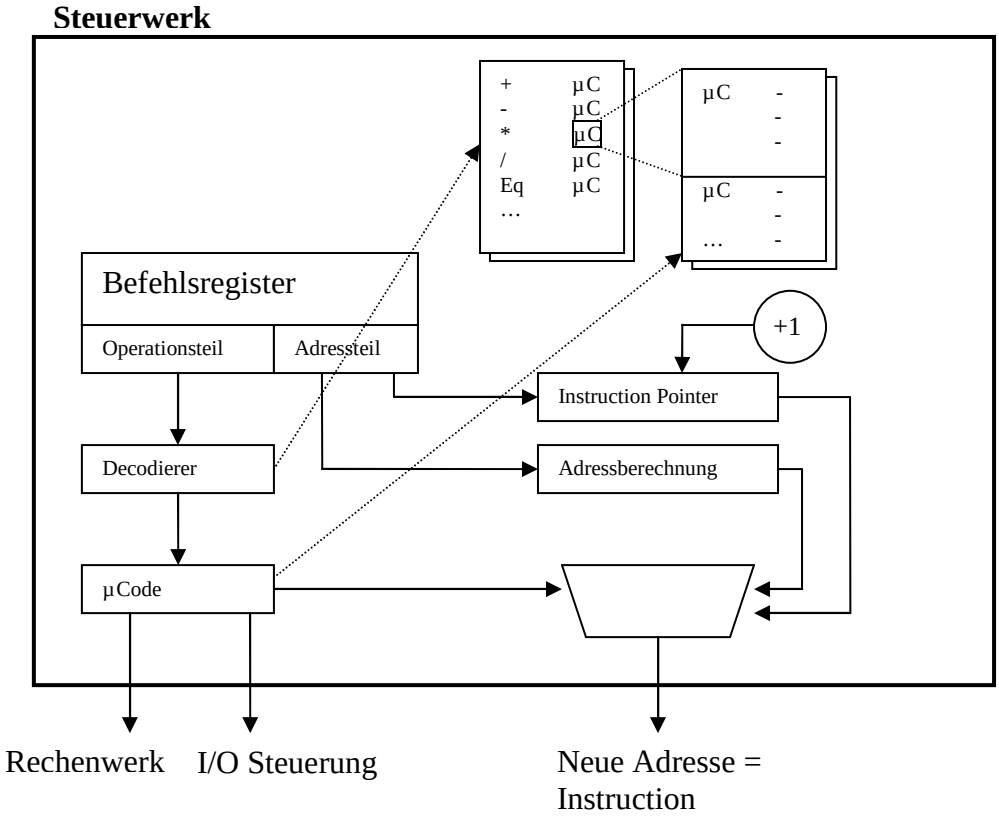
somit stehen defakto 9 Bit zur Verfügung. Das OS bzw. die Applikation hat die Verantwortung das Statusregister abzufragen und entsprechend der gesetzten Flags zu reagieren.

In der Intel Architektur stehen mehrere ALU's zur Verfügung, welche intern verschiedene Aufgaben erledigen, z.B. gibt es eine ALU für das Rechnen mit Gleitkommazahlen (Floating Point)



3.3 Steuerwerk

Die Maschinenbefehle werden in der CPU, genauer im Steuerwerk zuerst ins Befehlsregister geladen und von dort aus decodiert. Danach wird der entsprechende Mikrocode aus dem Rom gelesen und entsprechend dieses Mikrocodes werden die Verschiedenen Elemente wie z.B. die Alu, I/O Steuerung geschaltet, bzw. gesteuert.



3.3.1 Das Befehlsregister

Das Befehlsregister ist Teil des Steuerwerkes und enthält den **aktuellen Befehl**, nicht zu verwechseln mit dem Befehlszeiger (**I**nstruction **P**ointer)
Der Befehl wird als erstes ins Befehlsregister geladen.

z.B. **MOV AX**, FFh

Dies geschieht folgendermassen: Der CPU schaut nach, wohin der Instruction Pointer (IP) zeigt, und holt von dort den Befehl. Dieser Befehl bleibt im Befehlsregister, bis er komplett abgearbeitet ist. Der Befehl besteht aus zwei Teilen:

1. Dem Operationsteil (z.B. MOV)
2. Dem Adressteil (z.B. AX)

An Hand des Befehles sieht der CPU, ob noch weitere Parameter benötigt werden (beim MOV AX {Laden des AX Registers} ist klar, dass genau ein weiterer Wert benötigt wird.
Bei anderen Befehlen:

z.B. **NOP** = No OPeration = keine Operation
oder **INC** AX = INCrement = AX um eins erhöhen

wird kein weiterer Wert gebraucht. Sobald der Befehl in das Befehlsregister geladen wurde, wird der Befehlszeiger (**I**nstruction **P**ointer) um eins erhöht und zeigt dann je nach Anzahl der Parameter gleich auf den nächsten Befehl oder auf einen Wert, der zum ausführen des Befehls benötigt wird, wo dann nach dem Laden dessen wiederum um eins erhöht wird. Siehe 3.3.2

Intermezzo: Mnemonics und Opcode

Im Prinzip könnte man direkt so Programmieren:
10111011 11111111
dies würde jedoch wohl kein normaler Mensch mehr verstehen, deshalb hat man die so genannten Mnemonics erfunden. Mnemonics sind Abkürzungen, die einfach zu merken sind:
MOV AX, FFh heisst zum Beispiel, lade das AX Register mit dem Wert FFh.
Was nun der Assembler macht, ist eigentlich nichts anderes als das Übersetzen dieser Mnemonics in Maschinenbefehle, also in Nullen und Einsen. Dies funktioniert einfach gesagt so, dass er in einer Tabelle nachzuschauen was MOV AX für einen so genannten Opcode (OPeration CODE) hat. Also welches Binärwort der Prozessor benötigt um einen Wert ins AX Register zu laden. Nicht zu verwechseln mit dem Decodieren des Befehles in der CPU.

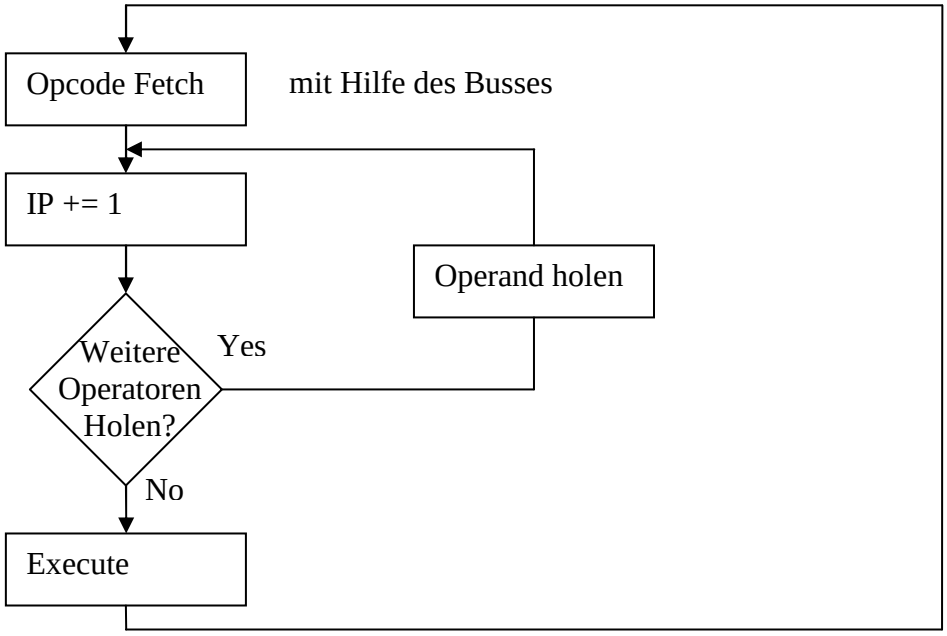
Opcodes können ein 2 Verschiedene Arten unterteilt werden

- Steuernachricht
- Operatoren Nachricht

3.3.2 Fetch – Decode – Execute

Instruktionsebene

to fetch = engl. holen



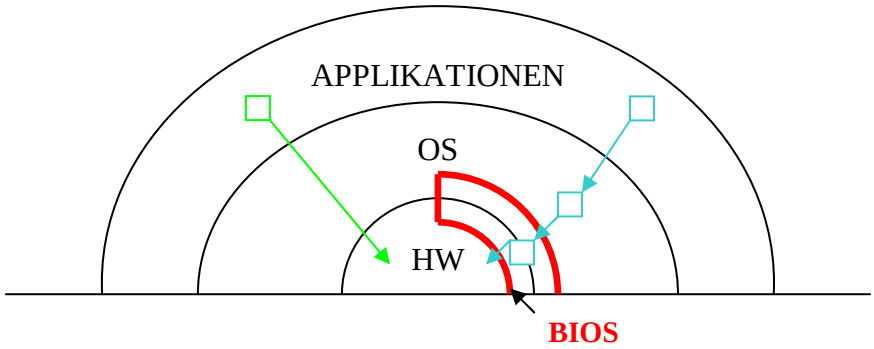
4 Bios

Basic Input Output System

Das Bios ist zuständig für Kompatibilität während der Laufzeit, in dem es Software Interrupts zur Verfügung stellt und für den Startvorgang.

4.1 Runtime (Laufzeit)

Dies wird am besten am Schalenmodell erläutert:



Das Bios bietet einfach gesagt eine Schnittstelle an, mit welcher das Betriebssystem grundlegende Funktionen ausführen kann. Hätte man kein Bios, müsste man das Windows etwa folgendermassen bestellen: Windows für Dell Inspirion 8100 mit Intel xy Netzwerkkartenchip, xy Grafikkarte auf Interrupt z mit ….

Das Bios stellt nun Software Interrupts bereit, welche die Hardware richtig ansprechen, egal welcher Hardware Hersteller es ist. Oder anders gesagt bietet es eine Schnittstelle zur Hardware an, ohne dessen Adressen wissen zu müssen. Der Aufruf:
MOV AX, 3
INT 10h
Löscht zum Beispiel den Bildschirm (Videomodus 3) egal um welche Grafikkarte es sich handelt. Nicht zu verwechseln mit IR10 des Interrupt-Controllers (Chip: 8295A) der z.B. für die serielle Schnitstelle 3 zuständig ist.

Das Bios kann aber auch umgangen werden, denn bei gewissen Instruktionen macht es Sinn diese direkt aufzurufen z.B. FSIN zum Berechnen von Winkelfunktionen.

4.2 Boot time (Startvorgang)

Der Startvorgang geht in etwa folgendermassen:

Das Bios ist Memory Mapped, das heisst es lässt sich wie das Ram direkt von der CPU ansprechen. Verschiedenen Quellen zu folge wird die CPU beim Reset auf CS:IP FFFF:0000h[3] eingestellt, wo sie dann die erste Instruktion des Bios liest. Andere quellen sprechen von 0xC8000 bis 0xF0000[4] wo sich andere Karten, bzw. ihr Bios einnisten.

Auf jedenfall wird das Bios als erstes geladen, es liest die definierten Adressbereiche und versuicht so alle vorhandene Hardware zu erkennen. Danach werden diese getestet (POST / Power on selfe test) auch wird überprüft wie viel RAM vorhanden und ob es in Ordnung ist (Butterfly Test), ausserdem kopiert sich das Bios selber ins RAM, da ROM relativ langsam ist, danach lädt es den ersten Teil des Betriebssystem ins RAM und setzt den Befehlspointer auf dessen erste Instruktion, was danach geschieht wird vom Betriebssystem bestimmt.

[3] http://savage.net.au/Ron/html/hex-ram-tutorial.html :: BIOS = Basic Input Output System
[4] http://en.wikipedia.org/wiki/BIOS :: Firmware on Adapter Cards

5 Das RAM (*R*andom *A*ccess *M*emory)

Wobei zu sagen ist, dass RAM nicht für Zufallsspeicher steht, denn im Computer sollte nichts dem Zufall überlassen werden. RAM bedeutet beliebig zugreifbaren Speicher.
Im Prinzip besteht RAM auch aus Flip-Flops (SRAM) bzw. Kondensatoren mit vor geschaltetem Transistor (DRAM), mit dem Unterschied, dass die für alle Speicherstellen im RAM nur ein Anschluss vorhanden ist, somit muss es einen Weiteren Anschluss geben um die Speicherstelle im RAM auszuwählen, auch muss unterschieden werden, ob aus dem RAM gelesen oder ins RAM geschrieben wird.

Somit hat ein RAM Baustein folgende Anschlüsse:

- Daten z.B. 16 Bit
- Adresse z.B. 16 Bit
- Takt (Synchrones Ram)
- Reset für alle Speicherstellen (MMU)
- Read/not write bzw. Write Enable (WE)
- Output enable (OE)
- Power (Supply Voltage / V_{cc})
- Ground (V_{ss})
- Chip Enable