Duale Hochschule Manheim DHBW

Kurs TINF21AI1

Rechnerarchitekturen I

CPU 2

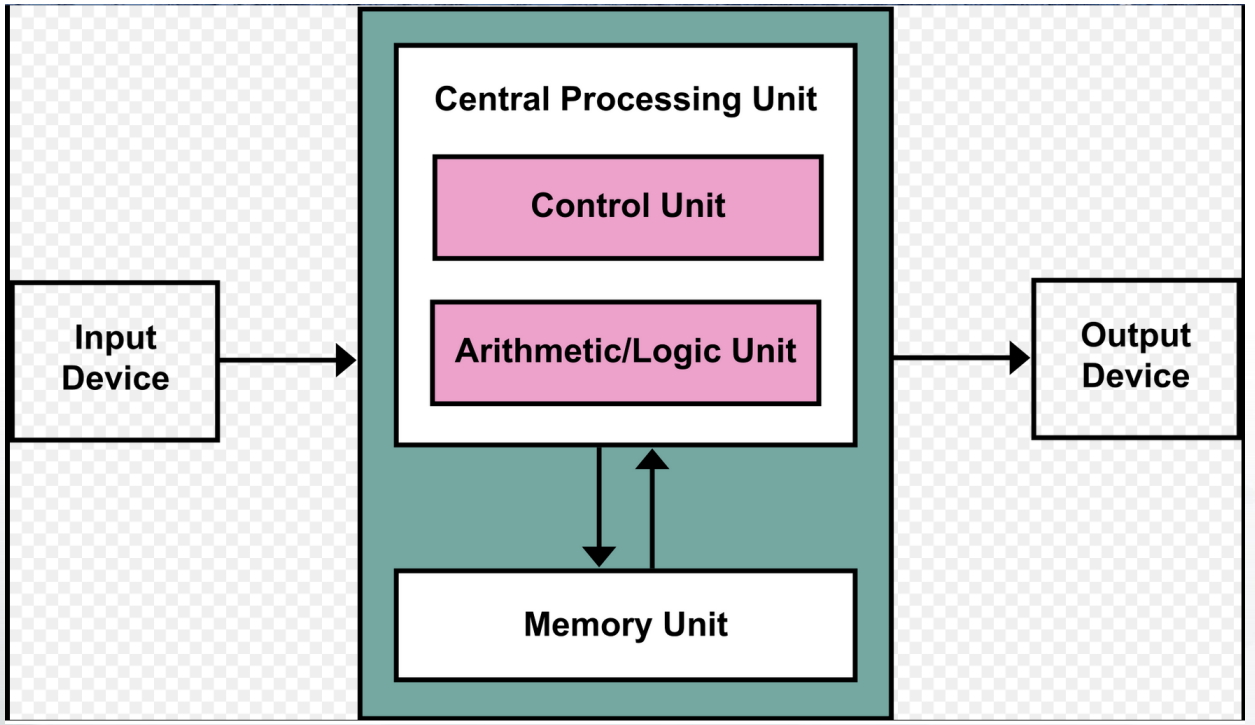
Wiederholung von Neumann Rechner

Wir haben in den folgenden CPUs einen gemeinsamen Speicher nach von Neumann.

- CPU
  ALU (Arithmetic Logic Unit) Rechenwerk
  Control Unit Steuerwerk oder Leitwerk
- BUS Bus System (Steuerbus, Adressbus, Datenbus)
- Memory – (RAM/Arbeitsspeicher) Speicherwerk/MMU/Cache
- I/O Unit – Eingabe-/Ausgabewerk in der Regel über Steuerbus anzusprechen

Hierzui werden drei Pointer verwendet

- Programm/Instruktion pointer
- Daten Pointer
- Stack Pointer



https://commons.wikimedia.org/wiki/File:Von_Neumann_Architecture.svg#/media/File:Von_Neumann_Architecture.svg

Die Intel Welt

8086 und 8088 unterstützen nur(MAX) 20 Adressleitungen, was den adressierbaren Speicher auf 1 MB entspricht.

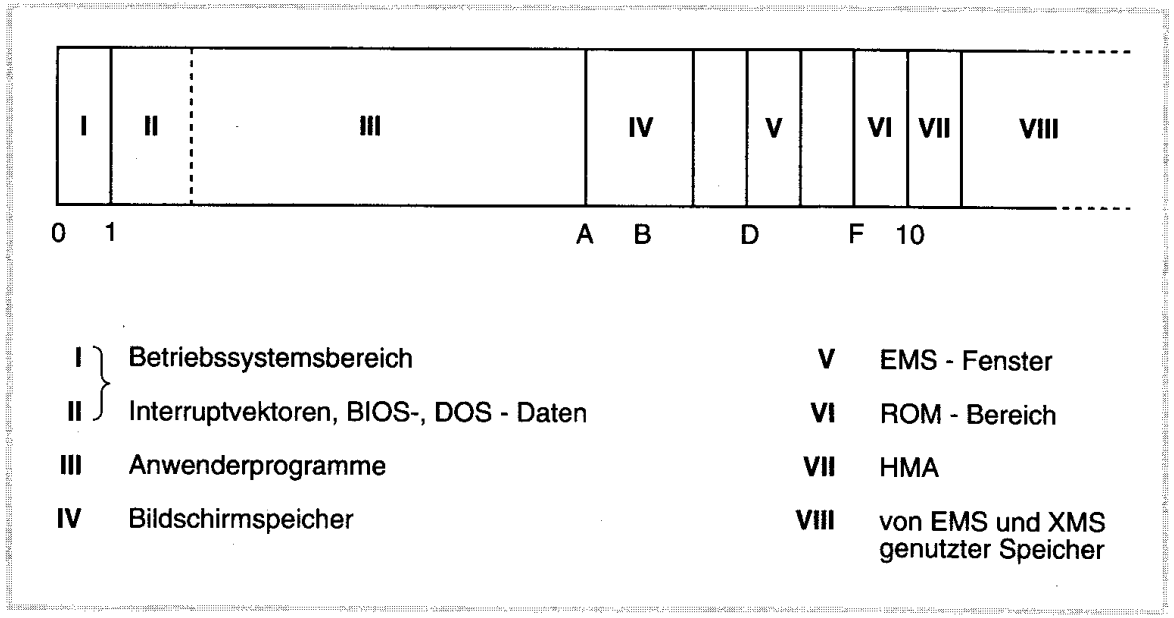
DOS Grenze von 1kByte bis 640 kByte
Interruptvektoren 0 bis 1 kByte

640 kByte bis 1 Mbyte BIOS, Extended Memory usw.

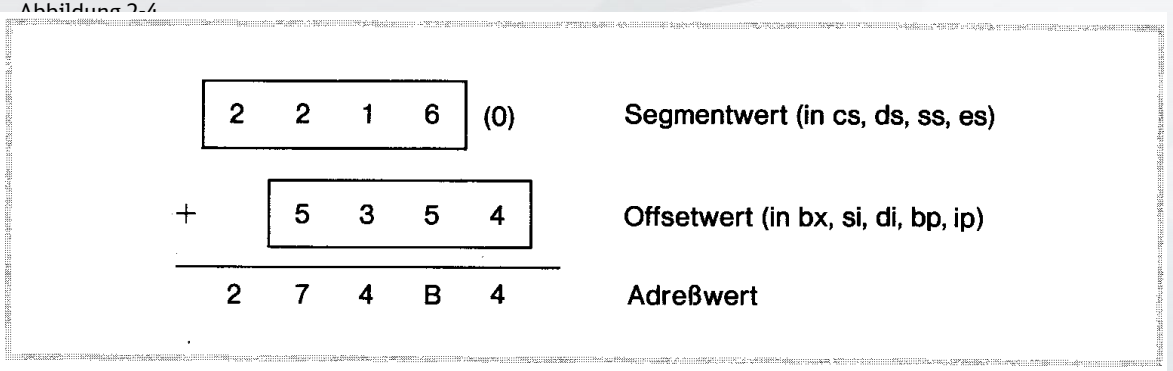
Oberhalb 1 MByte HMA Extended Memory $GW usw.

Um diesen fragmentierten Speicher zu Belegen wurde mit Offsets gearbeitet.

Speicher 20 Adressbits = Adressregister 16 Bits + 4 Bit Offset



PC-Assemblerkurs 2. Auflage, Heiss, Peter, ISBN 2-88229-016-1, Abbildung 2-4



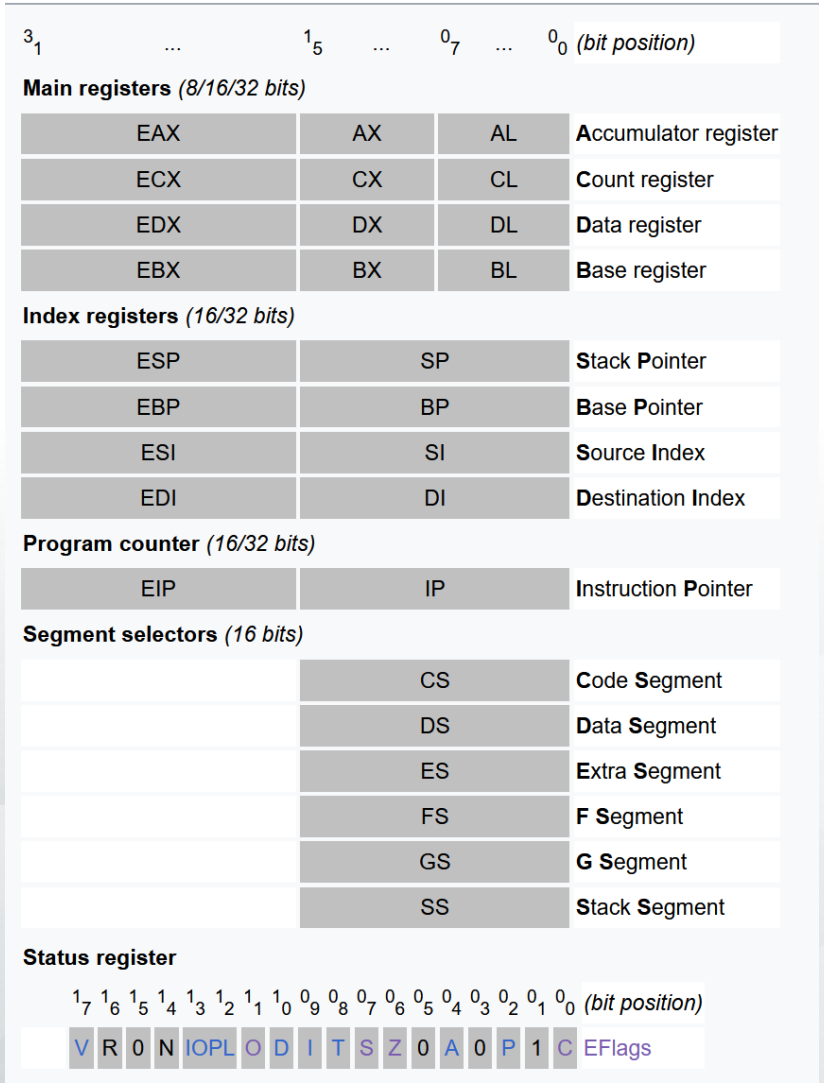
PC-Assemblerkurs 2. Auflgag, Heiss, Peter, ISBN 2-88229-016-1, Abbildung 2-3

Intel ab i386 und i486

- 32-Bit-Architektur für Adressen, Register, ALU und Befehle arbeiten mit bis zu 32 Bit
- Geschützter Modus für ein mehrstufiges Berechtigungsmechanismus
- 80386-MMU mit einem flachen Speichermodell mit einer 32-Bit-Adresse das auf 4-GB-Bereich zugreift.
  => Eine Manipulation von Segmentregistern und Offsets ist nicht mehr erforderlich.
- Dreistufige Befehlspipeline
- Hardware-Debuging-Register, Debug-Register mit Unterstützung von Break Points

Datentypen

- Byte: 8 bits
- Word: 16 bits
- Doubleword: 32 bits
- Quadword: 64 bits
- Double quadword: 128 bits



https://en.wikipedia.org/wiki/I386

Wichtige Adressierungsarten für Intel

Alle Adressierungen beziehen sich immer auf die Sicht der Register.

Implied Addressing Mode
clc ; Clear the carry flag (CF in the EFLAGS register)

Register addressing
mov eax, ecx ; kopiere zwischen zwei Register

Immediate addressing
mov eax, 9 ; Move 9 in das Eax register

Direct memory addressing
mov eax, [0a6bch] ; Coper den Wert aus Adresse 0a6bc in das EAX Register

Register indirect addressing
mov eax, [esi] ; Kpoiere den 32-bit Wert in das Eax Register der aus dem Register ESI kommt (pointer reference a variable in C/C++)

Wichtige Adressierungsarten für Intel

Indexed addressing
mov eax, [esi + 0ah] ; Kopiere den 32-bit Wert von der Adresse (ESI + Offset 0ah) ins EAX Register

Based indexed addressing
mov eax, [ebx + esi + 10] ; Kopiere den 32-bit Wert beginnend mit der address EBX + ESI + 10(Offset) ins EAX Register

Based indexed addressing with scaling
mov eax, [ebx + esi*4 + 10] ; Kopiere den 32-bit Wert beginend mit der Address (EBX + ESI*4 + 10) ins EAX Regiser
wobei der Faktor 1, 2, 4 und 8 verwendet werden kann.

Arten von Befehle

- Data movement Daten Befehle ohne impact auf die Flagregister wie mov usw.
- Stack manipulation Zum arbeiten mit dem Stack wie push usw.
- Aritmetic und Logic Behle add usw.
- Conversions, Befehle zum Umwandeln von Daten wir cbw byte in word
- Control flow Sprünge, Schleifen mit Flag Bezug
- String Manupulation, Befehle zur arbeit mit dem String cmps
- Flag manipulation, Veränderung, setzen von Flagn usw.
- Input/Putput Befehle um Daten zwischen Register und I/O Devices zu verschieben
- Protected Mode für Betriebssysteme Multitasking BS wichtig

Weiter Kategorien

- Floating-point instructions: These instructions are executed by the x87 floating-point unit.
- SIMD instructions: This category includes the MMX, SSE, SSE2, SSE3, SSE4, AVX, AVX2, and AVX-512 instructions. Some of the instruction sets in this category were introduced in the SIMD processing section of Chapter 8, Performance-Enhancing Techniques.
- AES instructions: These instructions support encryption and decryption using the Advanced Encryption Standard (AES).
- MPX instructions: The memory protection extensions (MPX) enhance memory integrity by preventing errors such as buffer overruns.
- SMX instructions: The safer mode extensions (SMX) improve system security in the presence of user trust decisions.
- TSX instructions: The transactional synchronization extensions (TSX) enhance the performance of multithreaded execution using shared resources.
- VMX instructions: The virtual machine extensions (VMX) support the secure and efficient execution of virtualized operating systems.
- Usw.

32-bit ARM

Reine RISC Cpu mit einfacheren Adress-Schema.

Alles ist ein Register, es gibt drei Befehle zum Daten austausch.

- ldr instruction lädt ein register aus dem Speicher
- str Speichert ein register in den Speicher
- mov Copiert zwischen Register Daten

Datentypen

- Byte: 8 bits
- Halfword: 16 bits
- Word: 32 bits
- Doubleword: 64 bits

3-Adressbefehle
ADD R2, R2, R1, dies würde R2 = R2 + R1 bedeuten.

Verwendung einer Pipeline.
Eine FPU ist an Cortex-M4 enthalten.

#	Purpose
R0	General purpose
R1	General purpose
R2	General purpose
R3	General purpose
R4	General purpose
R5	General purpose
R6	General purpose
R7	Holds Syscall Number
R8	General purpose
R9	General purpose
R10	General purpose
R11	Frame Pointer
Special Purpose Registers	
R12	Intra Procedural Call
R13	Stack Pointer
R14	Link Register
R15	Program Counter

https://developer.arm.com/documentation/dui0473/m/overview-of-the-arm-architecture/arm-registers

Adressierung 1

Register direct
Copiere ein register in ein anders
mov r0, r1

Register indirect
Verwende den Inhalt eines Registers als Adresse.
ldr r0, [r1] // Lade Adresse r1 in das Register r0
str r1, [r3] // Specere r1 in Adresse aus r3

Register indirect with offset
Lade auf der Adresse eines Register mit einem Offset
ldr r2, [r1, #32] // Lade r2 mit der Address [r1+32]
str r0, [r1, #8] // Speichere Register r0 in die Adresse [r1+8]

Register indirect with offset, pre-incremented
Die Adresse wird mit einem offset versehen und man hält dies nach um durch Daten zu laufen. Adresse nach dem Zugriff updaten.
ldr r2, [r1, #16]! // Lade r2 mit Adresse [r1+16] und update r1 to (r1+16)
str r2, [r1, #16]! // Spechere r2 mit Adresse [r1+16] und update r1 to (r1+16)

Adressierung 2

Register indirect mit offset und einem post-incremented
Die Addresse wird für den Zugriff genutz und dann mit dem offset upgedatet
ldr r0, [r1], #16 // Lade aus adresse [r1] in das Register r0 und dann date r1 + 16 ab
str r0, [r1], #4 // Speichere ro in Adress von [r1] ab und dann addiere 4 r1

Double register indirect
Verwende die Summe zweier Register für den Zugriff
ldr r0, [r1, r2] // Lade r0 mit der Summe von [r1+r2]
str r0, [r1, r2] // Speicher r1 in der Adresse der Summer aus [r1+r2]

Double register indirect with scaling
Greife mit hilfe eines Scalings auf die Adresse zu
ldr r0, [r1, r2, lsl #5] // Lade r0 mit [r1+(r2*32)]
str r0, [r1, r2, lsr #2] // Speichere r0 mit [r1+(r2/4)]

Befehlsgruppenübersicht

Load/store Befehle zu laden und speichern wie ldr

Stack manipulation Stack Befehle wie push {r0, r2, r4-r11}

Register movement Daten zwischen Register kopieren mov, usw.

Arithmetic and logic Arthmetik Befehle add, usw.

Comparisons Befehle zum Vergleichen

Control Flow Befehle in Abhängigkeit zu Flags usw.

Conditional execution Befehle die mit der Möglichkeit für Verzweigungen versehen sind.

Spezial Befehle Breakpont, supervisor Befhle

Quelle

https://en.wikipedia.org/wiki/Main_Page

https://de.wikipedia.org/wiki/Wikipedia:Hauptseite

https://en.wikipedia.org/wiki/DOS/4G

https://www2.eecs.berkeley.edu/Pubs/TechRpts/1982/CSD-82-106.pdf