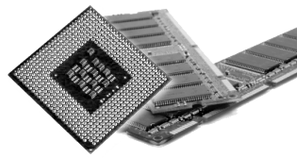
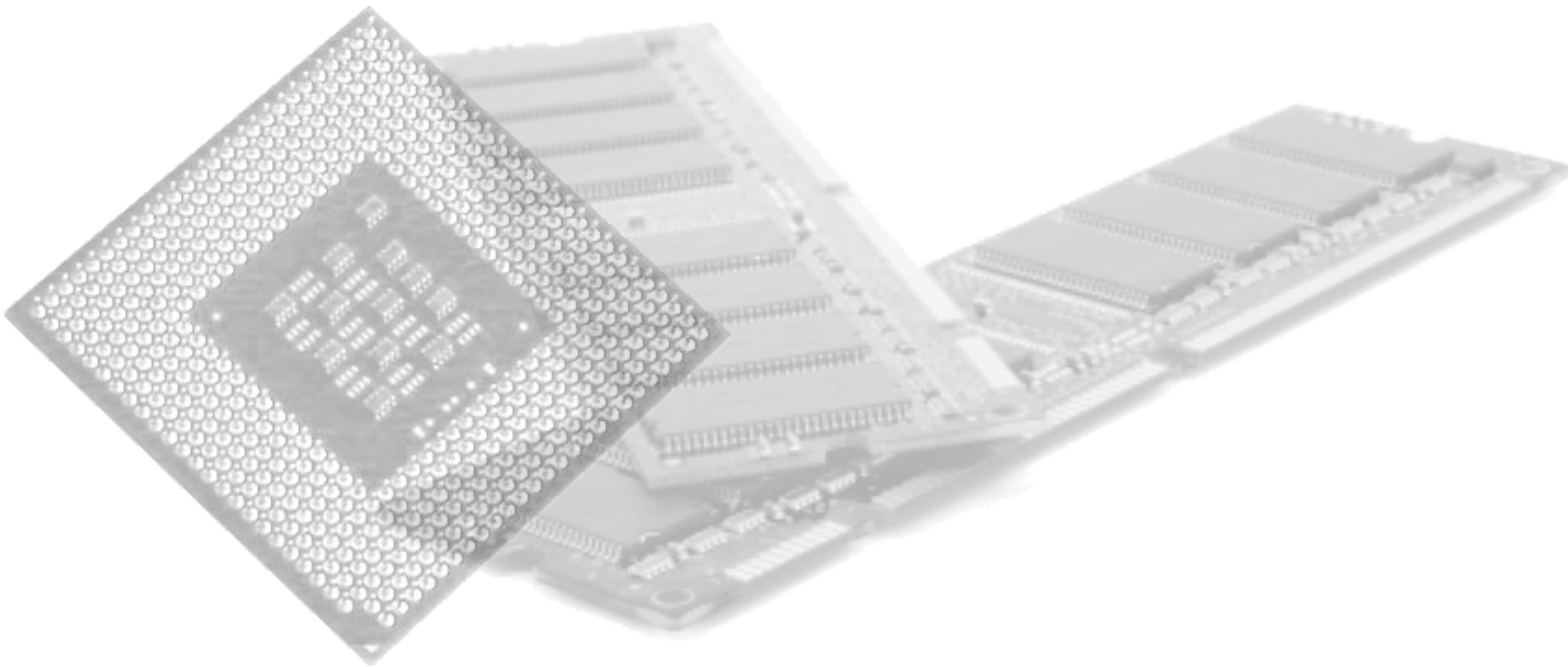
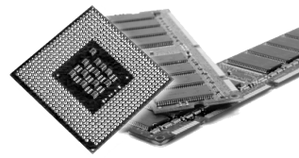


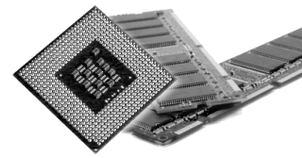
Assembler Grundlagen





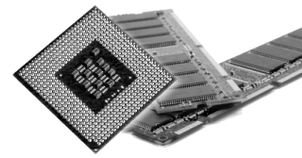
Architektur

- IA-32 „Intel Architecture 32-Bit“
- x86: Architektur und Befehlssatz
- x64: auch x86-64 (basiert auf der x86 Architektur)
  - auf 64 Bit “erweitert”
- ARM



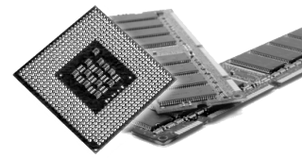
Bsp. Assembler

```
section .text
global CMAIN
CMAIN:
    mov ebp, esp; for correct debugging
    ;write your code here

    mov eax, [Var]
    mov ecx, 0

Marke:
    inc ecx
    mul DWORD [Var2]
    jc Ende
    jmp Marke

Ende:
    mov [Var3], eax
    mov [Var3+4], edx
```



Maschinensprache / Assembler

- Maschinenbefehle als Bitfolgen
- Siehe Aufbau Opcode 8086
- Assembler als lesbares Mapping zwischen „Text“ und Bitfolgen
  - Übersetzung von Assemblercode in Bitfolgen
- Assembler als Werkzeuge zur Übersetzung (vs. Disassembler)
- Für verschiedene Computertypen gibt es eine spezielle Assemblersprache
  - auf den Befehlssatz des Prozessors angepasst



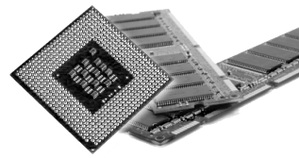
Vergleich Assembler / andere Sprachen

Hochsprache

Assembler

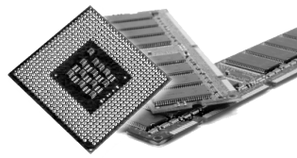
Maschinensprache

Prozessor

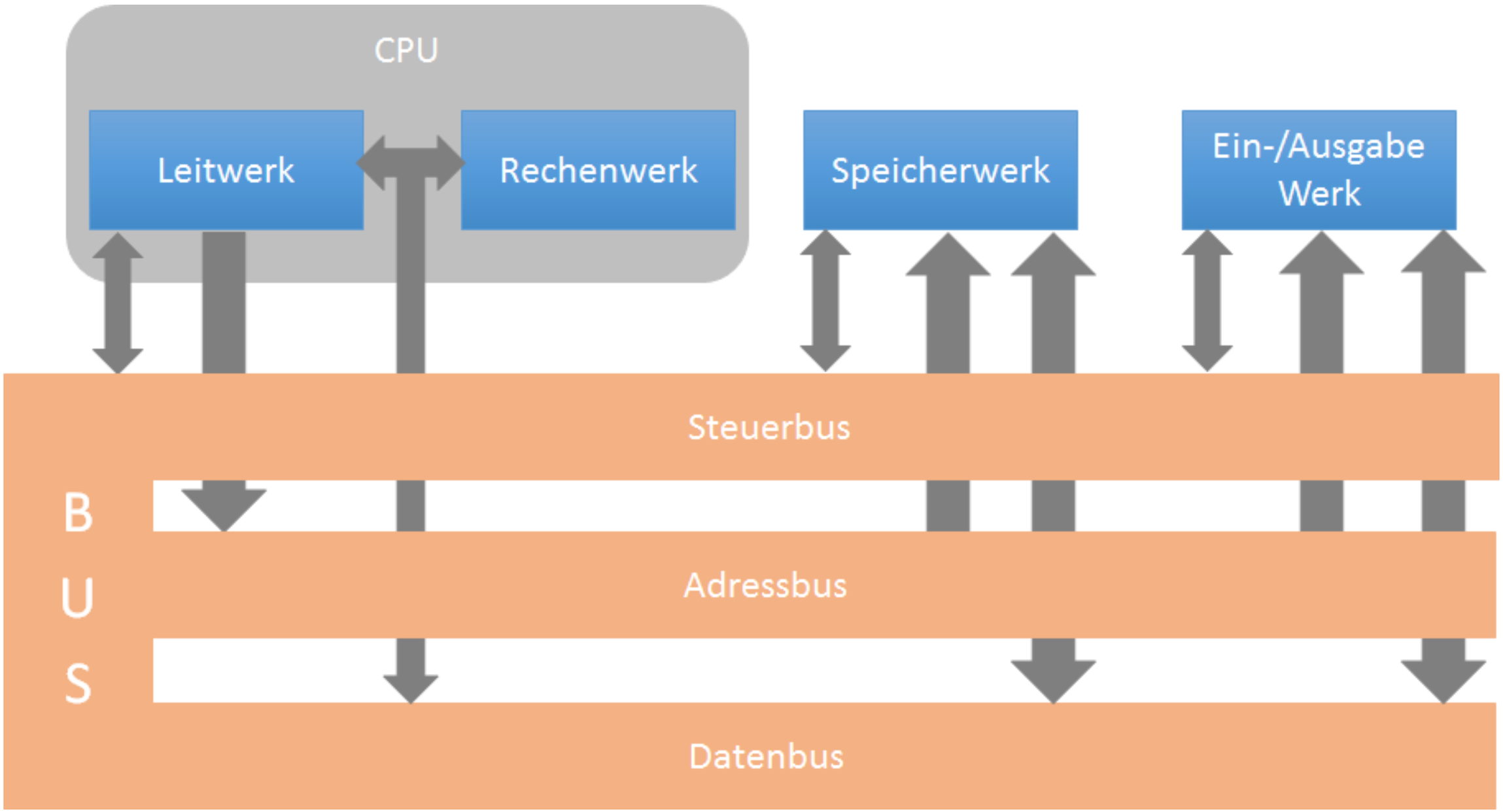


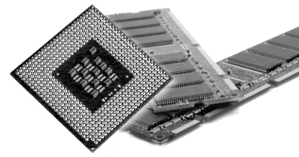
Vergleich Assembler / andere Sprachen

Assembler	Hochsprache (z.B. Java)
nicht ohne Anpassungen auf andere Computerarchitekturen portierbar	anderer Compiler für eine andere Computerarchitektur
längerer Quellcode, da die Instruktionen weniger komplex sind	leichtere Wartung
effizientere Code	Performance Compilerabhängig
Hardwarenah	sehr systemnahe Operationen ggf. nicht verfügbar
besseres Verständnis für die Rechnerarchitektur	Assemblercode kann meist integriert werden
	Meistens werden die Programme zuerst in Assemblercode übersetzt



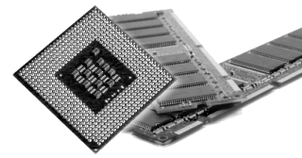
Der von-Neumann Rechner





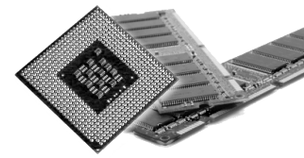
Register

	Accumulator		Counter		Data		Base	
64 Bit	RAX		RCX		RDX		RBX	
32 Bit		EAX		ECX		EDX		EBX
16 Bit			AX		CX		DX		BX
8 Bit			AH	AL	CH	CL	DH	DL	BH	BL



Assembler in C

```
6
7  #include <stdio.h>
8
9  int main(int argc, const char * argv[]) {
10     // insert code here...
11     asm("movl %ebx, %eax");
12     printf("Hello, World!\n");
13     return 0;
14 }
15
```



Befehlsanordnung

- Grundlegendes Prinzip:

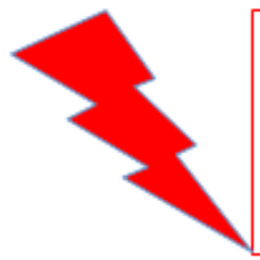
```
Befehl Ziel, Quelle
```

- Ein Operand muss ein Register sein

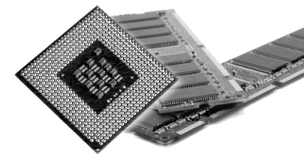
- Bei zwei Operanden: Verknüpfung von Quelle und Ziel, dann Speicherung in Ziel

- Ausnahme: Multiplikation / Division

```
MOV AX, BX
MOV AH, 123
MOV AL, 0CFh
MOV CX, [121]
```

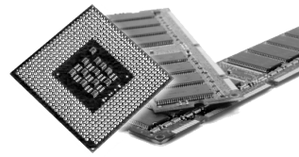


```
MOV BX, CL
MOV 123, AH
MOV AL, CFh
```

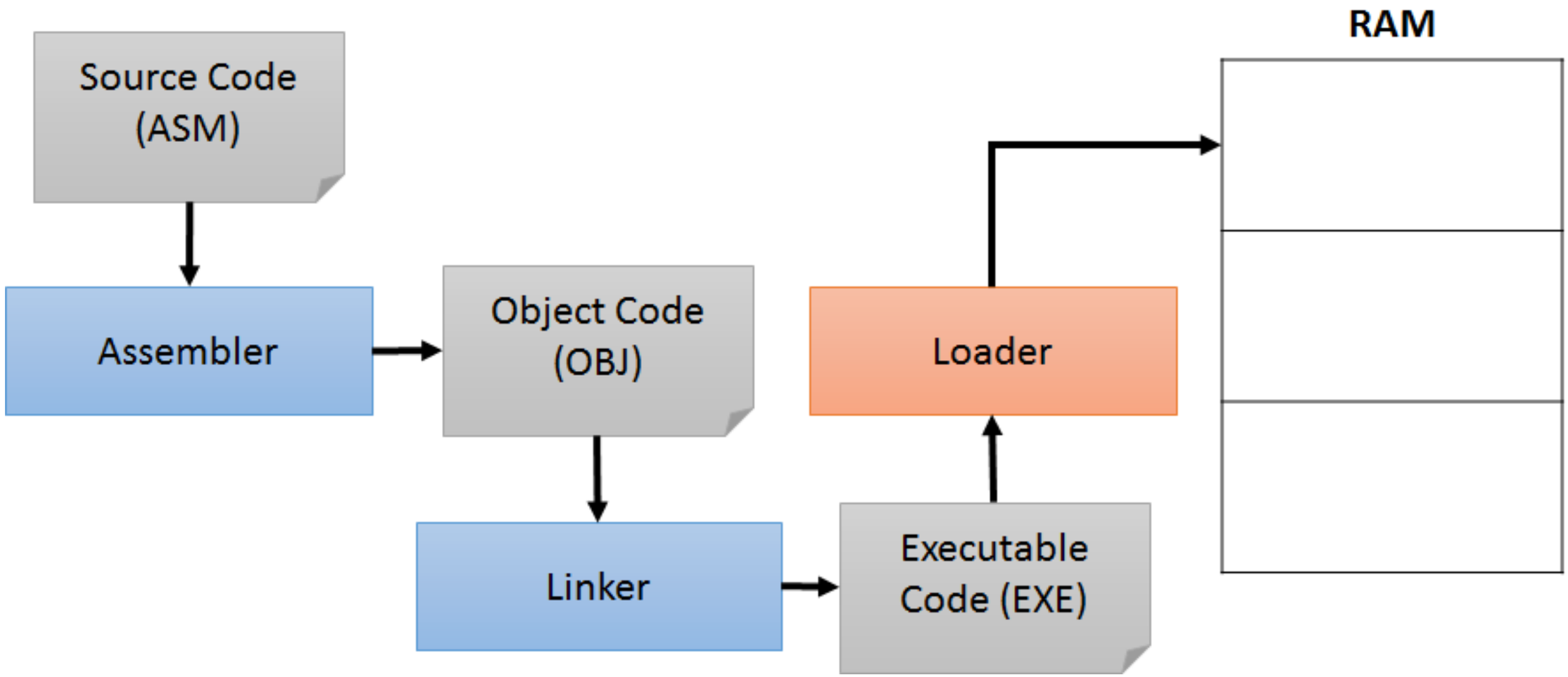


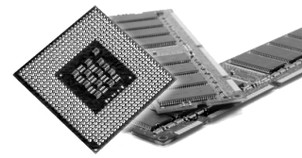
Assemblerwerkzeuge

- Assembler
  - erzeugt Maschinencode in einer OBJ Datei
- Linker
  - erzeugt ausführbare Datei (z.B. “.exe”)
- Debugger
  - „Überwacht“ Ausführung
  - zeigt den Inhalt der Register



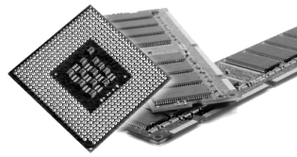
Programmablauf



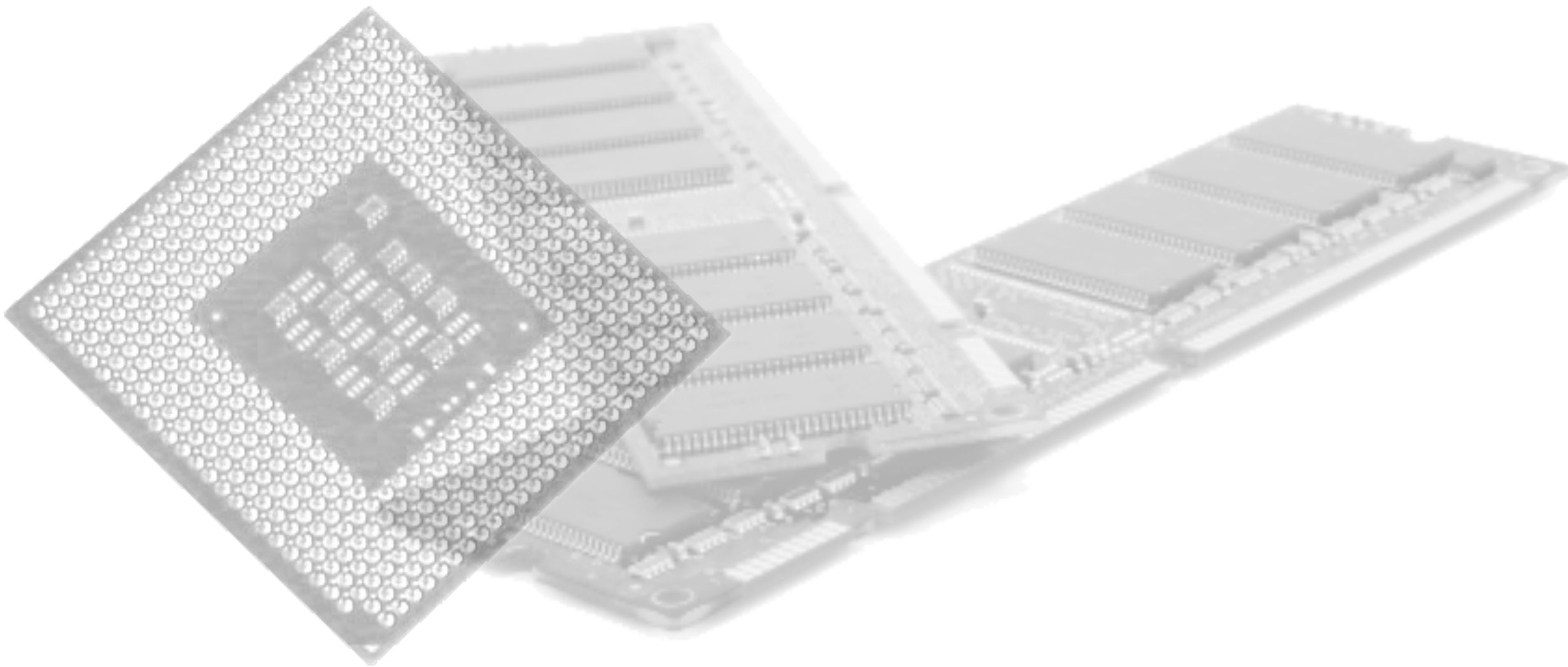


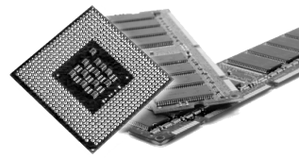
Übersicht Assembler

- TASM
  - Turbo Assembler (Borland)
- NASM
  - Netwide Assembler
- FASM
  - Flat Assembler
- MASM
  - Microsoft Assembler
- GAS
  - GNU Assembler



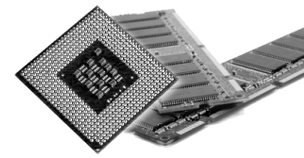
SASM IDE





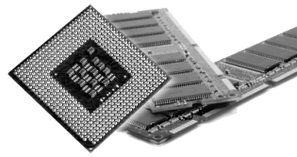
SASM

- SimpleASM
- Open Source IDE
- **NASM**, MASM, GAS, FASM (x86 und x64)
- für Windows und Linux
- mit Debugger
- "io.inc" NASM macro library
- http://dman95.github.io/SASM/english.html



"io.inc" NASM macro library

- PRINT_UDEC / PRINT_DEC
- PRINT_HEX
- PRINT_CHAR
- PRINT_STRING
- NEWLINE
- GET_UDEC / GET_DEC
- GET_HEX
- GET_CHAR
- GET_STRING



SASM

SASM

File Edit Build Debug Settings Help

U1.asm

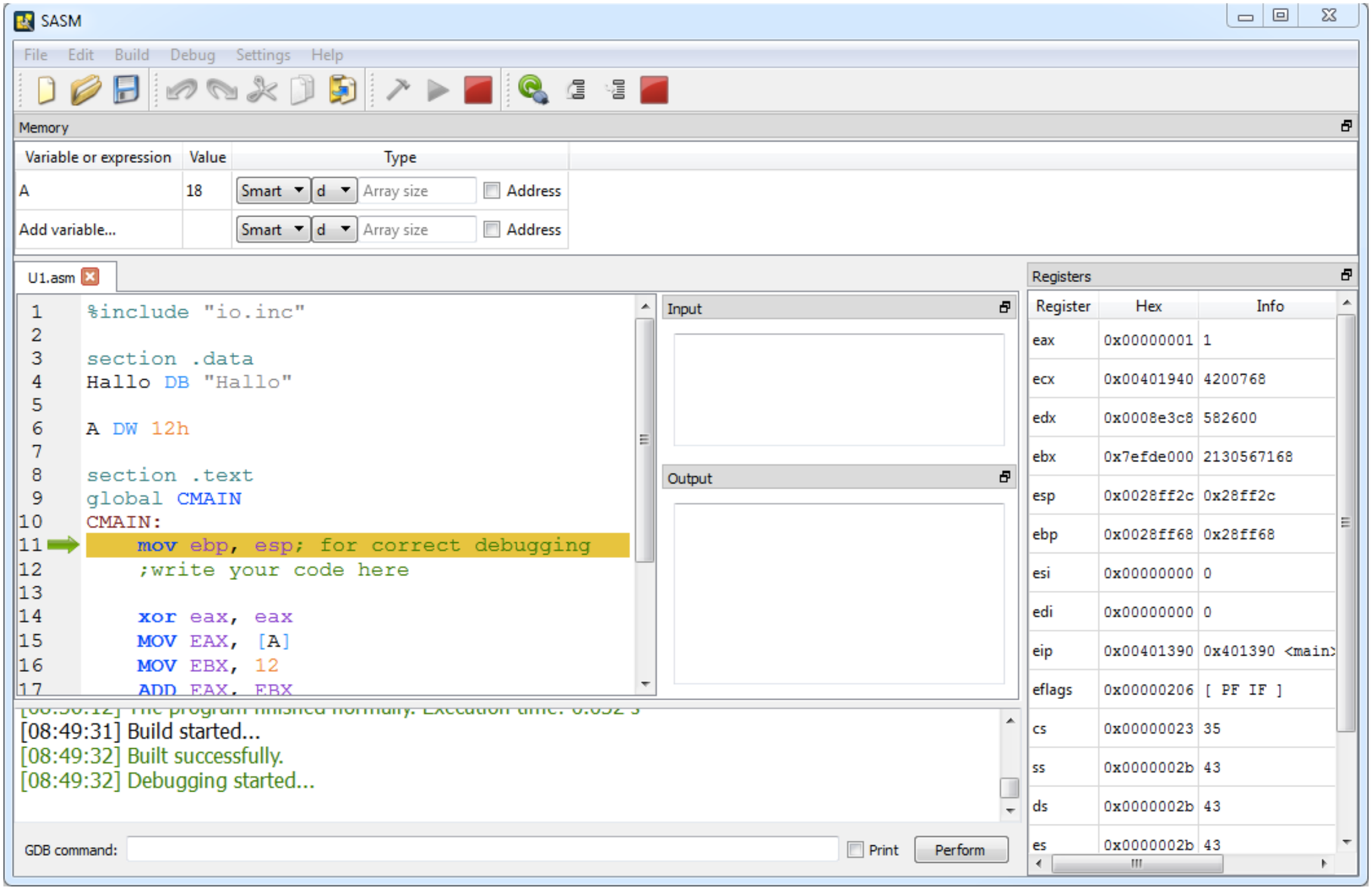
```
1   %include "io.inc"
2
3   section .data
4   Hallo DB "Hallo"
5
6   A DD 12h
7
8   section .text
9   global CMAIN
10  CMAIN:
11      mov ebp, esp; for correct debugging
12      ;write your code here
13
14      xor eax, eax
15      MOV EAX, [A]
16      MOV EBX, 12
17      ADD EAX, EBX
18
19      PRINT_STRING Hallo
20      PRINT_HEX 4, EAX
21
22      xor eax, eax
23      ret
```

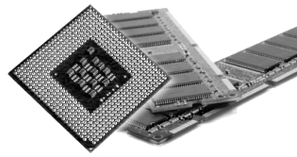
Input

Output

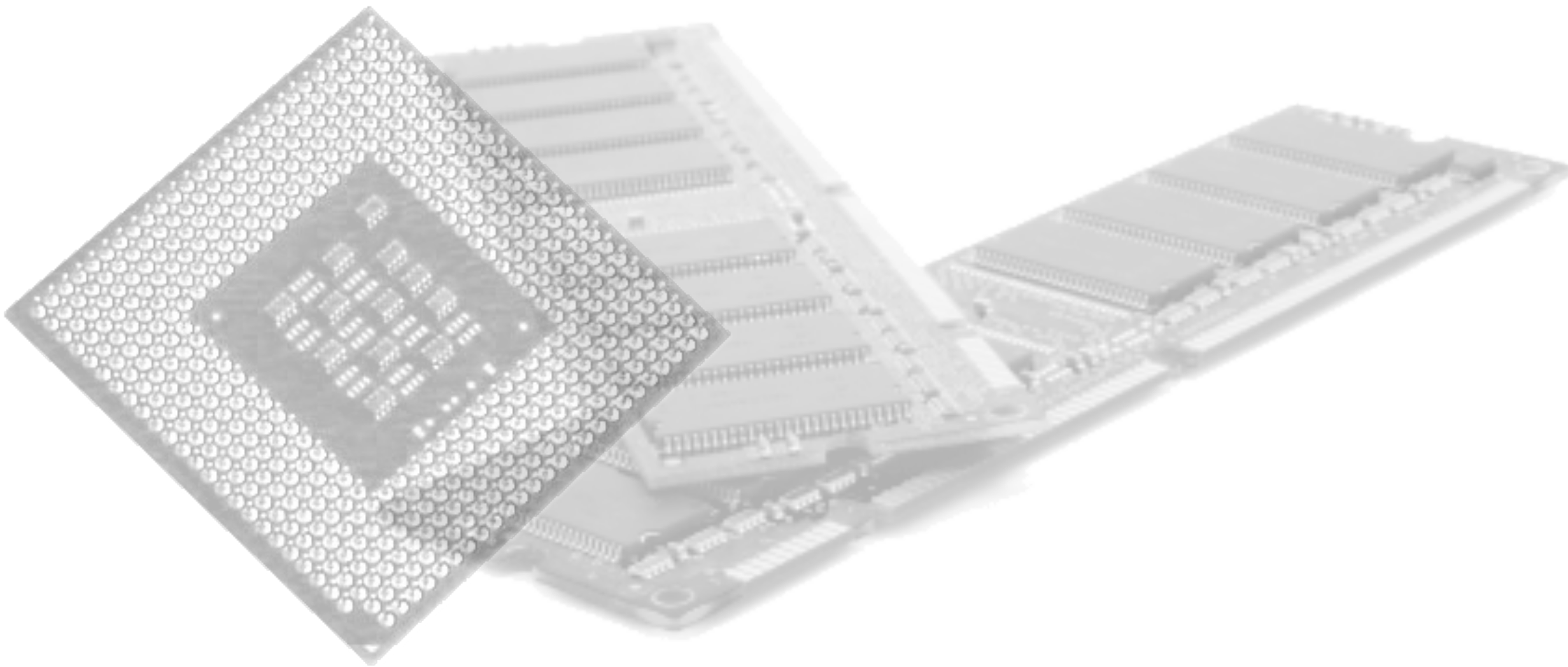
Build log:

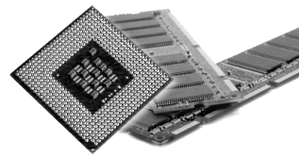
SASM


SASM
File Edit Build Debug Settings Help
Memory
Variable or expression
Value
Type
A
18
Smart
d
Array size
Address
Add variable...
U1.asm
1 %include "io.inc"
2
3 section .data
4 Hallo DB "Hallo"
5
6 A DW 12h
7
8 section .text
9 global CMAIN
10 CMAIN:
11 mov ebp, esp; for correct debugging
12 ;write your code here
13
14 xor eax, eax
15 MOV EAX, [A]
16 MOV EBX, 12
17 ADD EAX, EBX
Input
Output
Registers
Register
Hex
Info
eax 0x00000001 1
ecx 0x00401940 4200768
edx 0x0008e3c8 582600
ebx 0x7efde000 2130567168
esp 0x0028ff2c 0x28ff2c
ebp 0x0028ff68 0x28ff68
esi 0x00000000 0
edi 0x00000000 0
eip 0x00401390 0x401390 <main>
eflags 0x00000206 [PF IF]
cs 0x00000023 35
ss 0x0000002b 43
ds 0x0000002b 43
es 0x0000002b 43
[08:49:31] Build started...
[08:49:32] Built successfully.
[08:49:32] Debugging started...
GDB command:
Print
Perform



Assembler Befehle



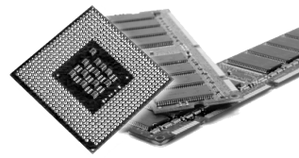


MOV

- MOV <Ziel>, <Quelle>

<Ziel>	<Quelle>
Register	Register
Register	Speicheradresse
Speicheradresse	Register
Register	Konstante
Speicheradresse	Konstante

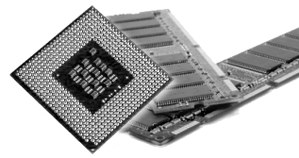
- mov eax, ebx
  - den Inhalt von ebx in eax schreiben
- mov eax, 100h
  - 100h in eax schreiben



ADD

- ADD <Ziel>, <Quelle>
- Ziel = Ziel + Quelle
- add eax, ebx
  - den Inhalt von ebx zu dem Inhalt von eax addieren
- add eax, 100h
  - 100h zu eax addieren

<Ziel>	<Quelle>
Register	Register
Register	Speicheradresse
Speicheradresse	Register
Register	Konstante
Speicheradresse	Konstante

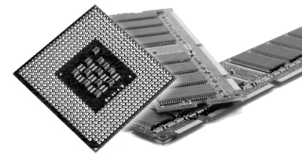


SUB

- SUB <Ziel>, <Quelle>
  - Ziel = Ziel - Quelle

<Ziel>	<Quelle>
Register	Register
Register	Speicheradresse
Speicheradresse	Register
Register	Konstante
Speicheradresse	Konstante

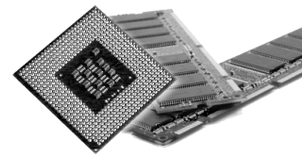
- sub eax, ebx
  - den Inhalt von eax minus den Inhalt von ebx
- sub eax, 100h
  - 100h von eax subtrahieren



MUL

- MUL <Quelle>
- Quelle: Register
- EAX = EAX * Quelle
- mov eax, 12h
- mov ebx, 14h
- mul ebx
  - eax = eax * ebx = 12h * 14h

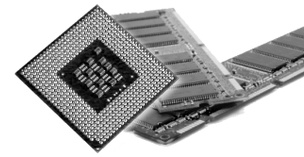
Operand	1 Byte	2 Byte	4 Byte
Operand 2	AL	AX	EAX
höherwertiger Teil	AH	DX	EDX
niederwertiger Teil	AL	AX	EAX



DIV

- DIV <Quelle>
- Quelle: Register
- EAX = EDX:EAX / Quelle
- EDX = Rest
- mov edx, 0
- mov eax, 4441h
- mov ebx, 2h
- div ebx
  - eax = edx:eax / ebx
  - edx = rest (eax / ebx)

Divisor	1 Byte	2 Byte	4 Byte
Dividend	AX	DX:AX	EDX:EAX
Rest	DX	DX	EDX
Quotient	AX	AX	EAX



INC

- INC <Ziel>

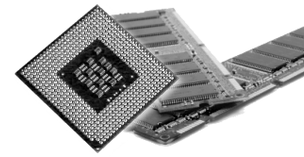
- Der Zieloperand wird um eins erhöht

- Ziel: Speicheradresse oder Register

- schnell

- inc EAX

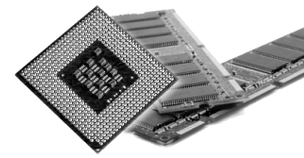
  - erhöht den Inhalt von EAX um eins (EAX + 1)



DEC

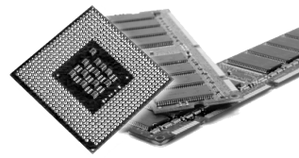
- DEC <Ziel>

- Der Zieloperand wird um eins vermindert
- Ziel: Speicheradresse oder Register
- schnell
- dec EAX
  - vermindert den Inhalt von EAX um eins (EAX - 1)



RET

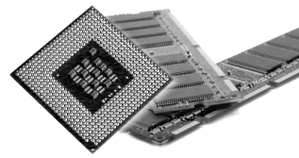
- RET
- Ein mit CALL aufgerufenes Unterprogramm wird beendet



Demo

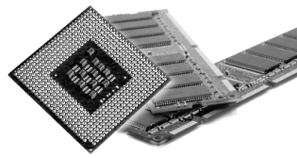
Demo.asm

```
1   %include "io.inc"
2
3   section .text
4   global CMAIN
5   CMAIN:
6       mov ebp, esp; for correct debugging
7       ;write your code here
8       mov eax, 100h
9       mov ebx, 0A43h
10
11      add eax, ebx
12
13      sub eax, 12h
14
15      mul ebx
16
17      inc eax
18
19      div ebx
20
21      dec eax
22
23      xor eax, eax
24      ret
```



Demo 2

- 4444444447h / 22222222h
- = 200h
- Rest: 47h



Demo 2

SASM

File Edit Build Debug Settings Help

Memory

Variable or expression	Value	Type
Add variable...		Smart d Array size Address

Demo2.asm

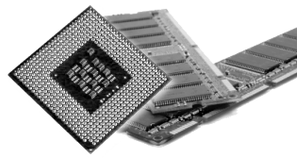
```
1   %include "io.inc"
2
3   section .text
4   global CMAIN
5   CMAIN:
6       mov ebp, esp; for correct debugging
7       ;write your code here
8       mov eax, 44444447h
9       mov edx, 44h
10      mov ebx, 22222222h
11
12      div ebx
13
14      xor eax, eax
15      ret
```

Input

Output

Registers

Register	Hex	Info
eax	0x44444447	1145324615
ecx	0x00401860	4200544
edx	0x00000044	68
ebx	0x22222222	572662306
esp	0x0028ff2c	0x28ff2c
ebp	0x0028ff2c	0x28ff2c
esi	0x00000000	0
edi	0x00000000	0
eip	0x004013a1	0x4013a1 <main+17
eflags	0x00000206	[PF IF]
cs	0x00000023	35
ss	0x0000002b	43



Demo 2

SASM

File Edit Build Debug Settings Help

New

Memory

Variable or expression	Value	Type
Add variable...		Smart d Array size Address

Demo2.asm

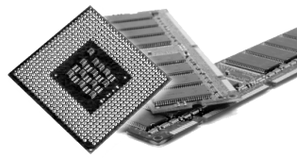
```
1   %include "io.inc"
2
3   section .text
4   global CMAIN
5   CMAIN:
6       mov ebp, esp; for correct debugging
7       ;write your code here
8       mov eax, 44444447h
9       mov edx, 44h
10      mov ebx, 22222222h
11
12      div ebx
13
14      xor eax, eax
15      ret
```

Input

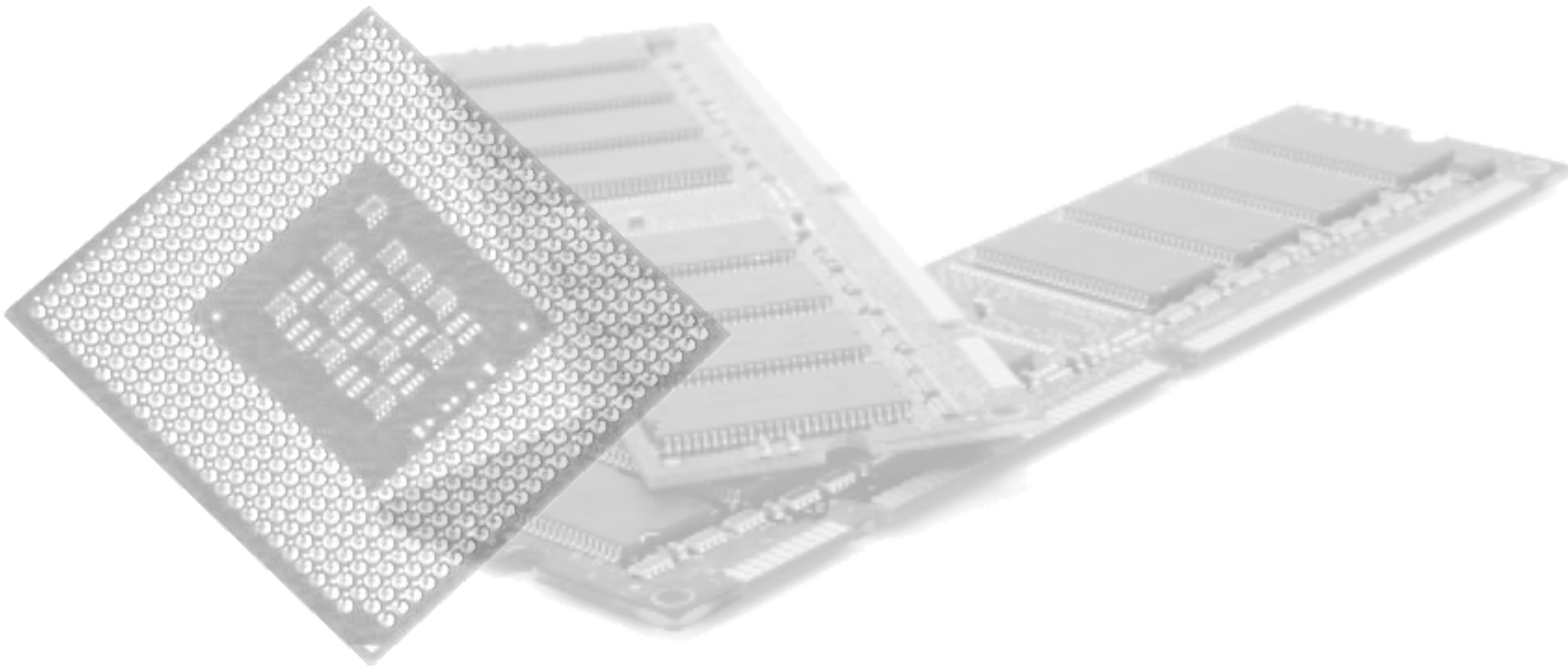
Output

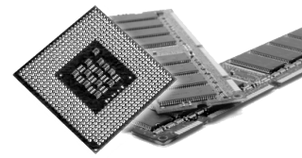
Registers

Register	Hex	Info
eax	0x00000200	512
ecx	0x00401860	4200544
edx	0x00000047	71
ebx	0x22222222	572662306
esp	0x0028ff2c	0x28ff2c
ebp	0x0028ff2c	0x28ff2c
esi	0x00000000	0
edi	0x00000000	0
eip	0x004013a3	0x4013a3 <main+19
eflags	0x00000206	[PF IF]



Variablen



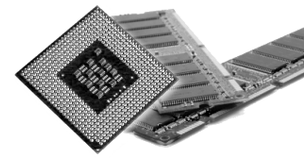


Variablen

- Variablen sind statische Speicherstellen, welche einen Namen im Programm zugewiesen bekommen

- Deklaration in section .data

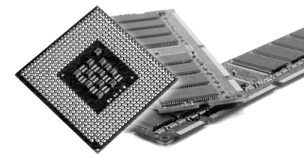
- z.B.:
  - Hallo DB “Hallo Welt”
  - A DD 12h



Variablen - Typen

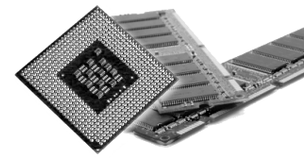
- **DB** - Define Byte (8 Bit)
- **DW** - Define Word
- **DD** - Define Double Word
- **DQ** - Define Quad Word
- …

DB	1 Byte
DW	2 Byte
DD	4 Byte
DQ	8 Byte



Variablen - Zugriff

- section .data
  - A DD 12h
- section .text
  - CMAIN:
  - mov EAX, [A]
- Bei den SASM Makros:
  - PRINT_HEX A

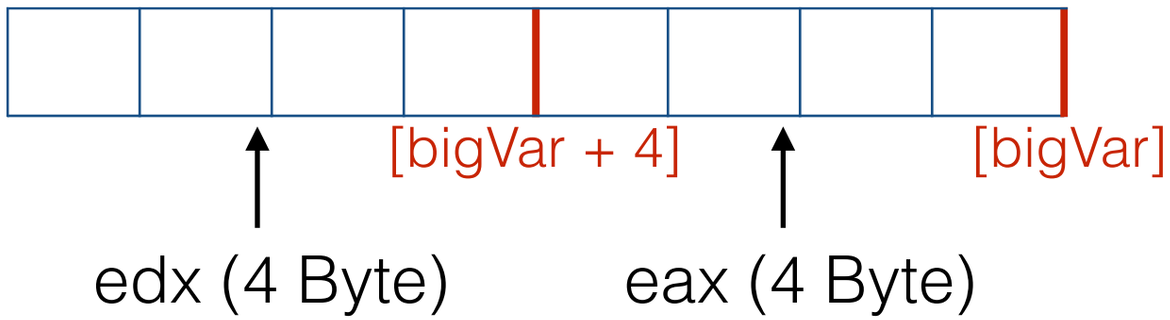


Variablen - Zugriff

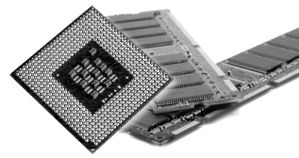
- section .data
  - A DD 12h
- section .text
  - CMAIN:
  - **mov EAX, [A] (Wert der Variable A in EAX)**
  - **mov EAX, A (Adresse der Variable A in EAX)**

Verwendung von DQ

- section .data
- bigVar DQ 0

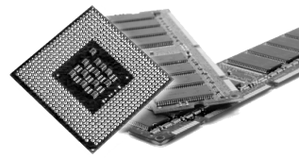


- section .text
- mov [bigVar], eax
- mov [bigVar+4], edx



Größenangaben

- Wie viele Bytes werden für den Befehl verwendet
  - Beim Schreiben in Variablen (die Größenangabe wird dabei nicht überprüft)
  - Wenn nicht auf alle Bytes einer Variable oder eines Registers zugegriffen werden soll
- Bsp.:
  - mov DWORD [bigVar], eax
  - mov [Var2], eax
  - mov [Var2], 0FFh FALSCH!!!
  - mov DWORD [Var2], 0FFh RICHTIG!



Position im Speicher

A	DW	99h
B	DD	86h
C	DW	12h
D	DD	54h