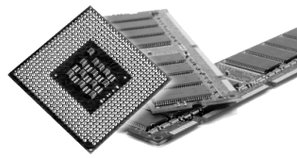
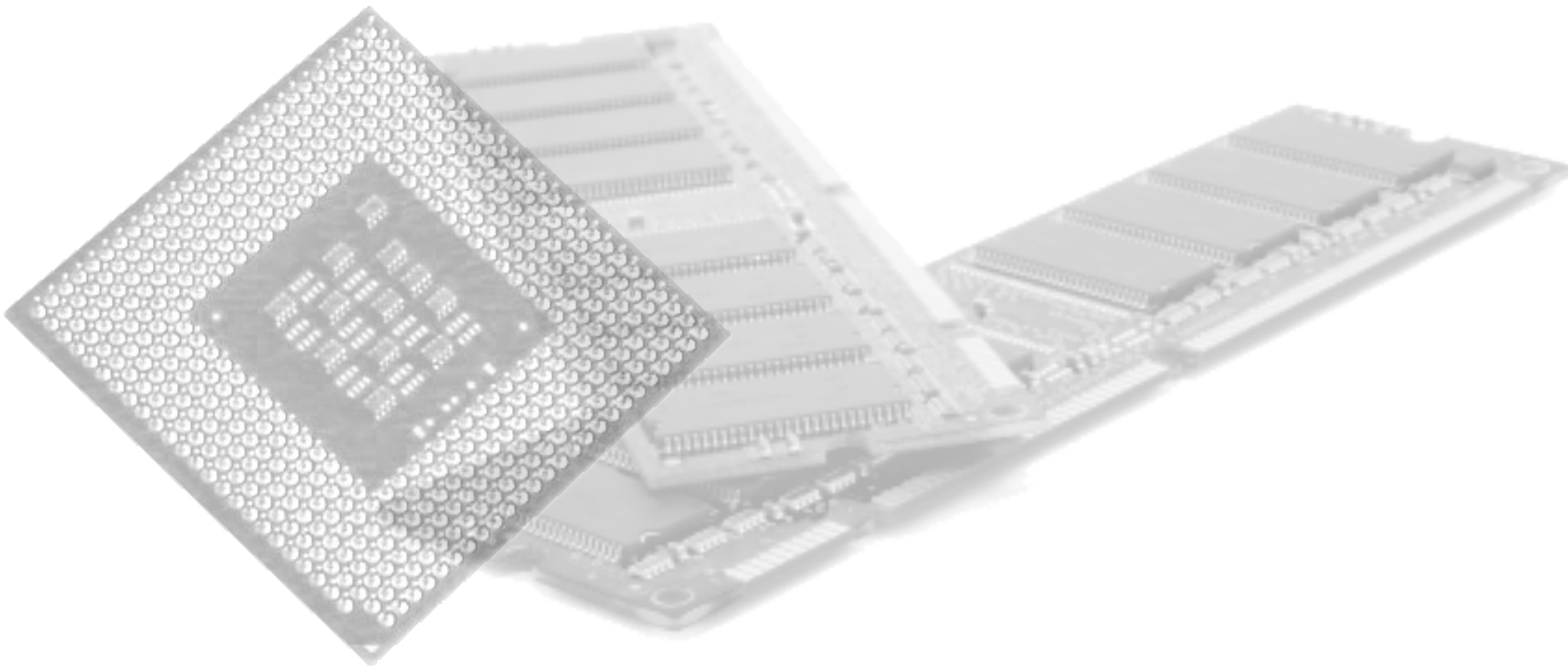
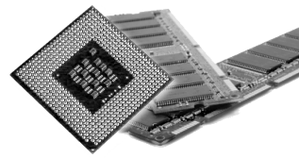


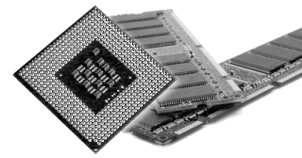
CPU / Assembler - erweiterte Informationen





CPUID

- Befehl um Prozessortyp und Befehlsunterstützung zu ermitteln
- EAX bestimmt die zu ermittelnden Informationen
- EAX = 0
  - höchster Parameter wird in EAX zurückgegeben
  - + die Hersteller ID



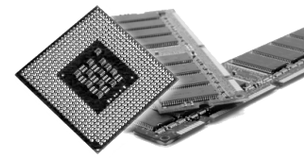
CPUID

The following are known processor manufacturer ID strings:

- `"AMDisbetter!"` – early engineering samples of AMD K5 processor
- `"AuthenticAMD"` – AMD
- `"CentaurHauls"` – Centaur (Including some VIA CPU)
- `"CyrixInstead"` – Cyrix
- `"GenuineIntel"` – Intel
- `"TransmetaCPU"` – Transmeta
- `"GenuineTMx86"` – Transmeta
- `"Geode by NSC"` – National Semiconductor
- `"NexGenDriven"` – NexGen
- `"RiseRiseRise"` – Rise
- `"SiS SiS SiS "` – SiS
- `"UMC UMC UMC "` – UMC
- `"VIA VIA VIA "` – VIA
- `"Vortex86 SoC"` – Vortex

The following are known ID strings from virtual machines:

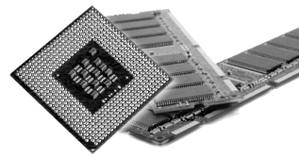
- `"KVMKVMKVM"` – KVM
- `"Microsoft Hv"` – Microsoft Hyper-V or Windows Virtual PC
- `" lrpepyh vr"` – Parallels (it possibly should be "prl hyperv ", but it is encoded as " lrpepyh vr")
- `"VMwareVMware"` – VMware
- `"XenVMMXenVMM"` – Xen HVM



CPUID

```
13      mov rax, 0
14      cpuid
15      PRINT_HEX 8, rax
16      NEWLINE
17      mov [s1], rbx
18      mov [s2], rdx
19      mov [s3], rcx
20      PRINT_STRING s1
21      PRINT_STRING s2
22      PRINT_STRING s3
```

```
16
GenuineIntel
```



CPUID

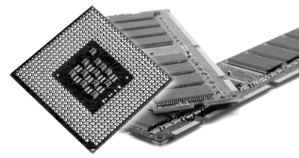
- EAX = 1
  - Prozessorinfos
  - Features

The format of the information in EAX is as follows:

- 3:0 – Stepping
- 7:4 – Model
- 11:8 – Family
- 13:12 – Processor Type
- 19:16 – Extended Model
- 27:20 – Extended Family

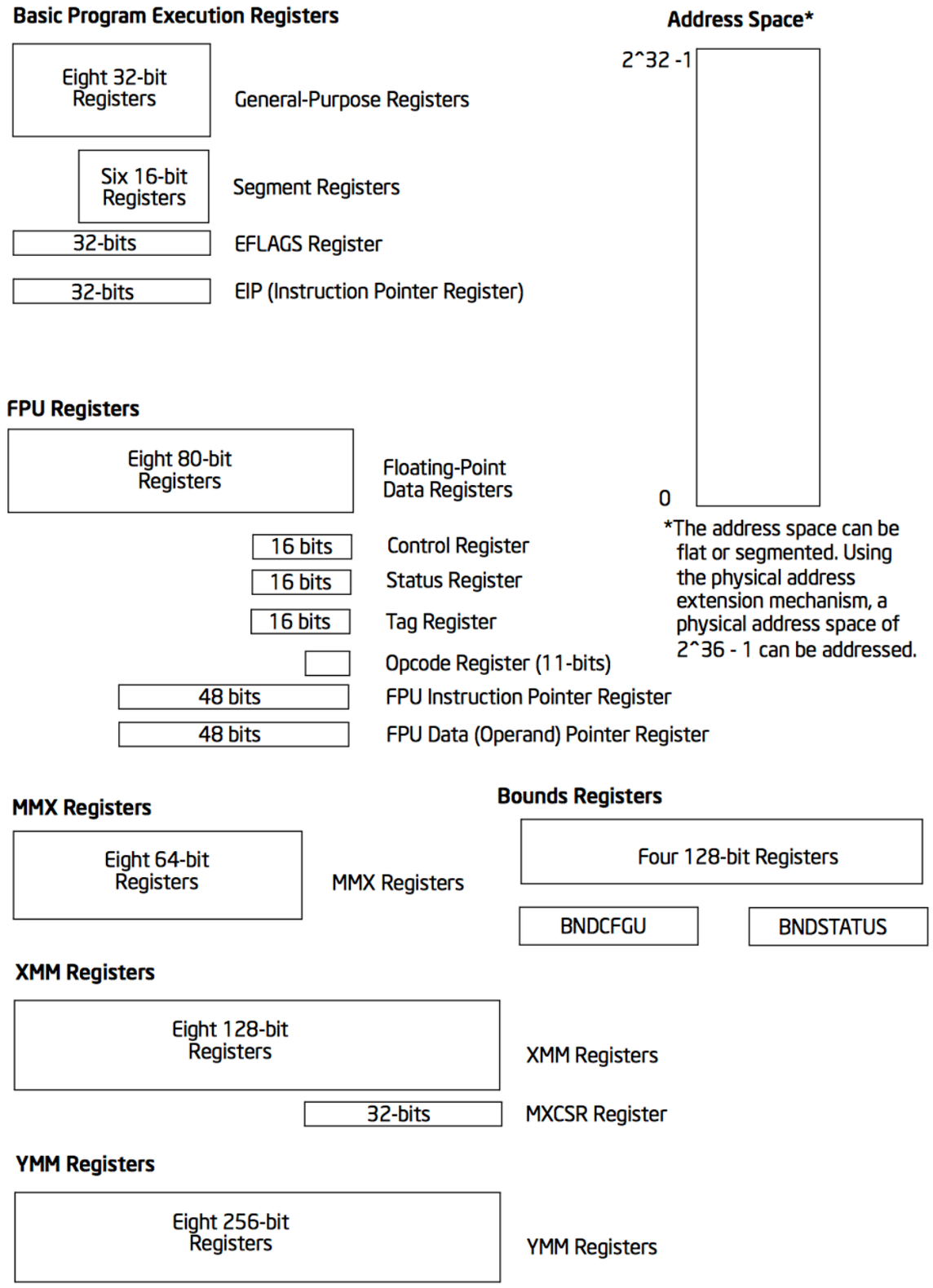
EAX=1 CPUID feature bits

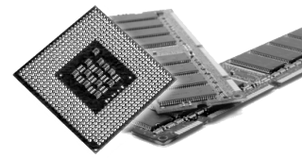
Bit	EDX		ECX	
	Short	Feature	Short	Feature
0	fpu	Onboard x87 FPU	sse3	Prescott New Instructions-SSE3 (PNI)
1	vme	Virtual 8086 mode extensions (such as VIF, VIP, PIV)	pclmulqdq	PCLMULQDQ support
2	de	Debugging extensions (CR4 bit 3)	dtes64	64-bit debug store (edx bit 21)
3	pse	Page Size Extension	monitor	MONITOR and MWAIT instructions (SSE3)
4	tsc	Time Stamp Counter	ds-cpl	CPL qualified debug store
5	msr	Model-specific registers	vmx	Virtual Machine eXtensions
6	pae	Physical Address Extension	smx	Safer Mode Extensions (LaGrande)
7	mce	Machine Check Exception	est	Enhanced SpeedStep
8	cx8	CMPXCHG8 (compare-and-swap) instruction	tm2	Thermal Monitor 2



32 Bit Register

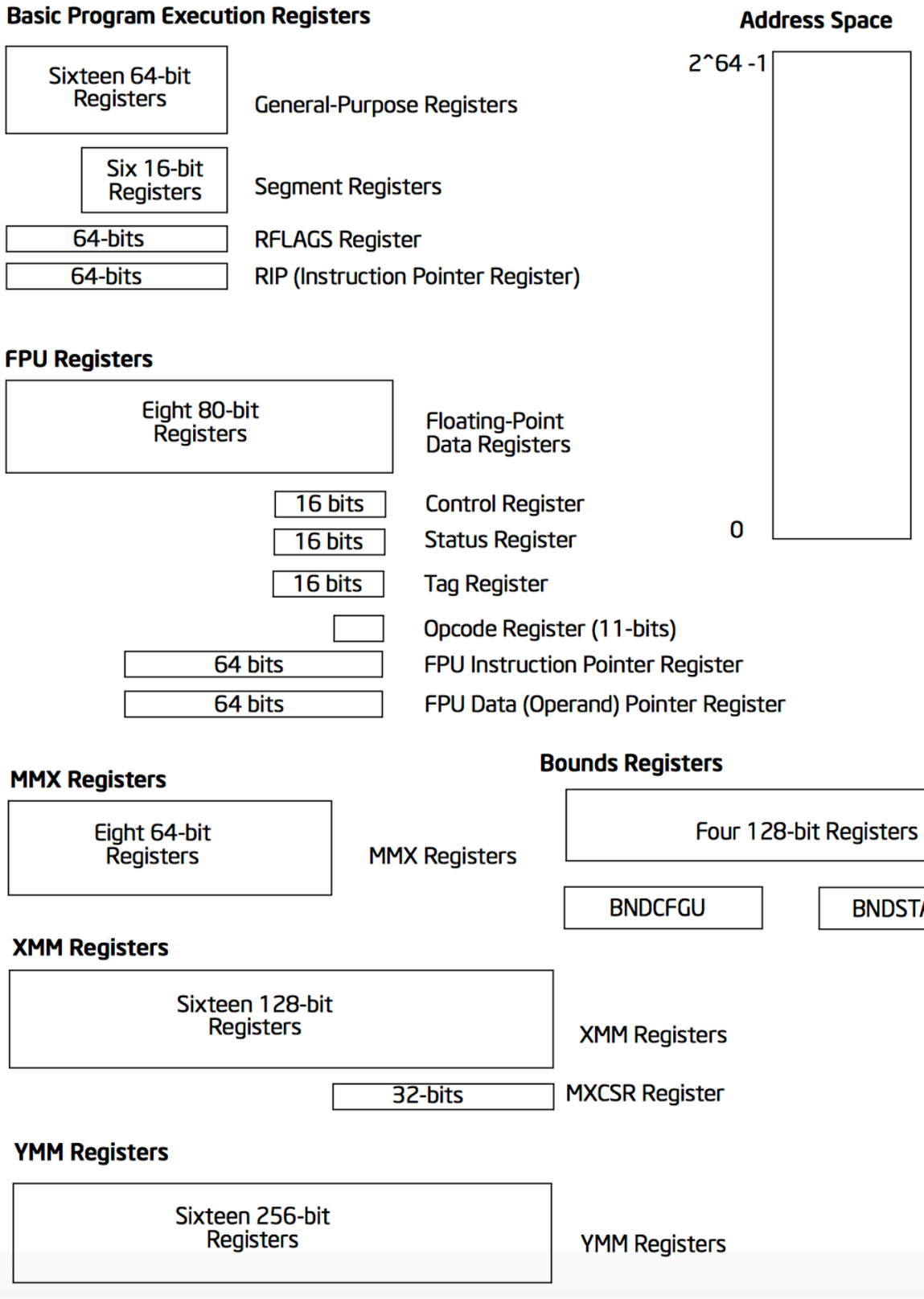
- Intel 32 Bit CPU

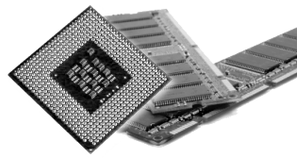




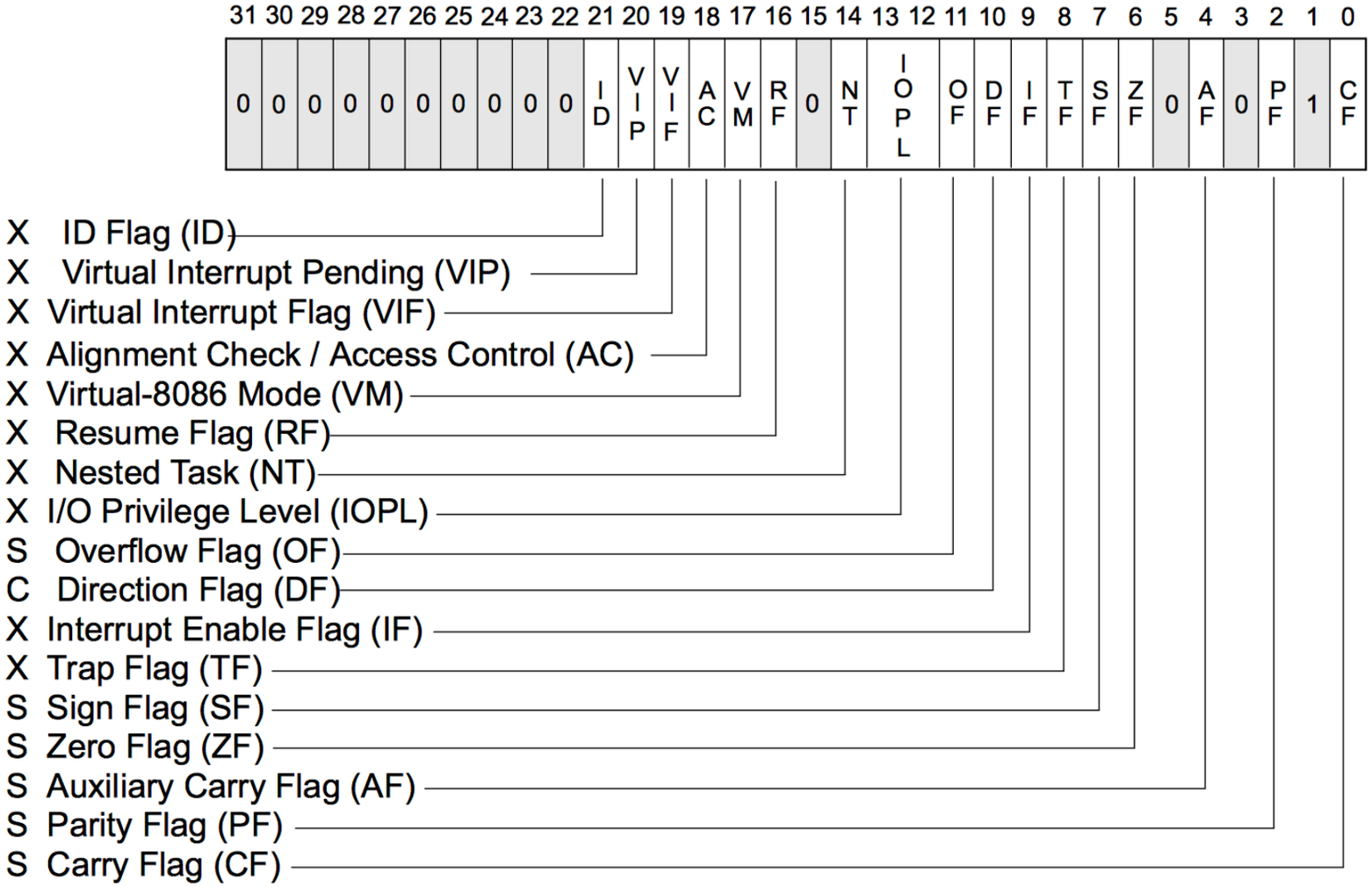
64 Bit Register

- Intel 64 Bit CPU



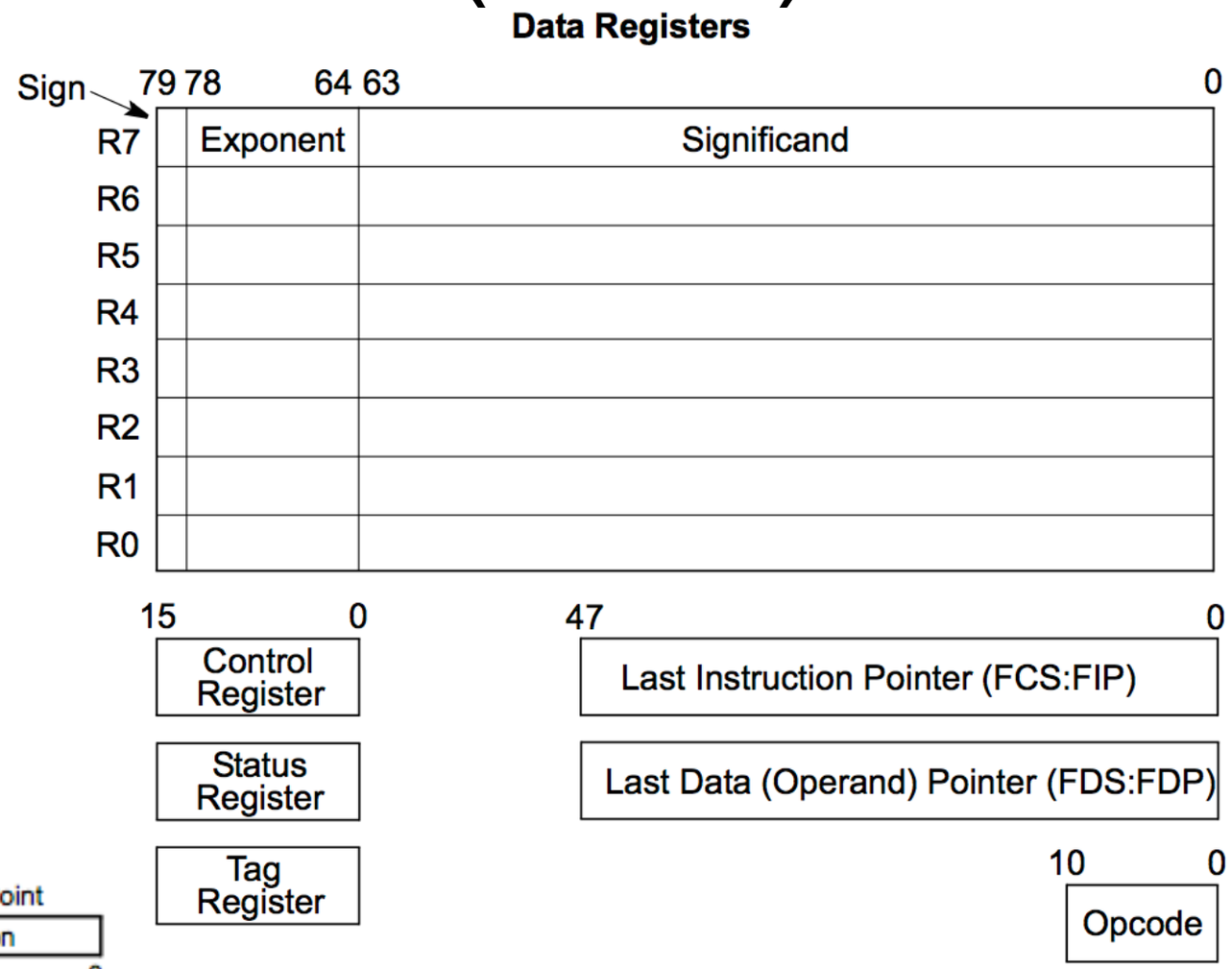


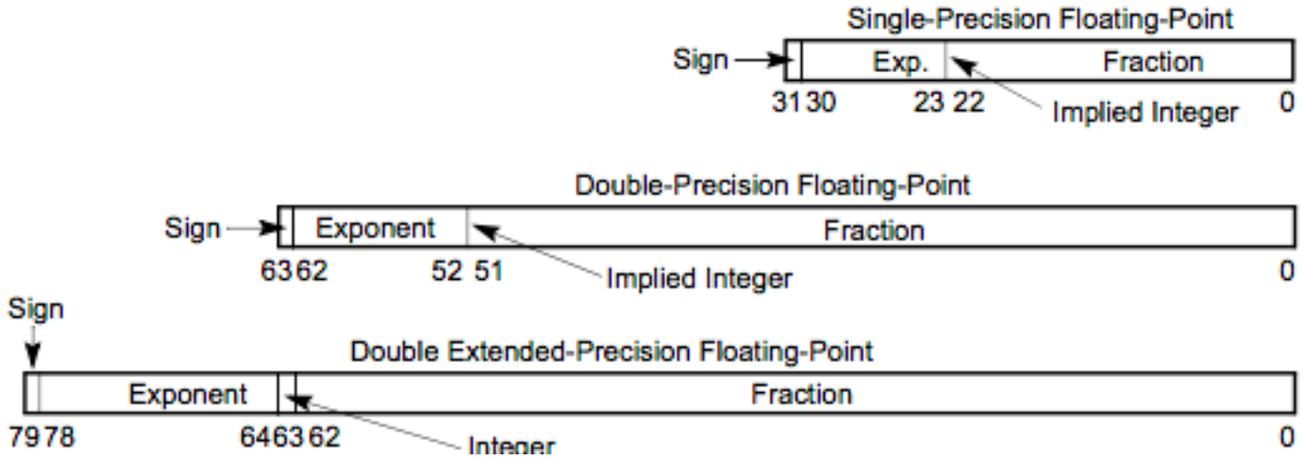
Flags

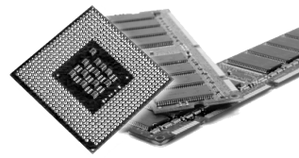

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 ID VIP VIF AC VM RF 0 NT IOPL OF DF IF TF SF ZF 0 AF 0 PF 1 CF
X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check / Access Control (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)
S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag
Reserved bit positions. DO NOT USE.
Always set to values previously read.

Floating-Point Unit (FPU)

- IA-32: 8 * 80-bit Register
- double extended-precision floating-point format





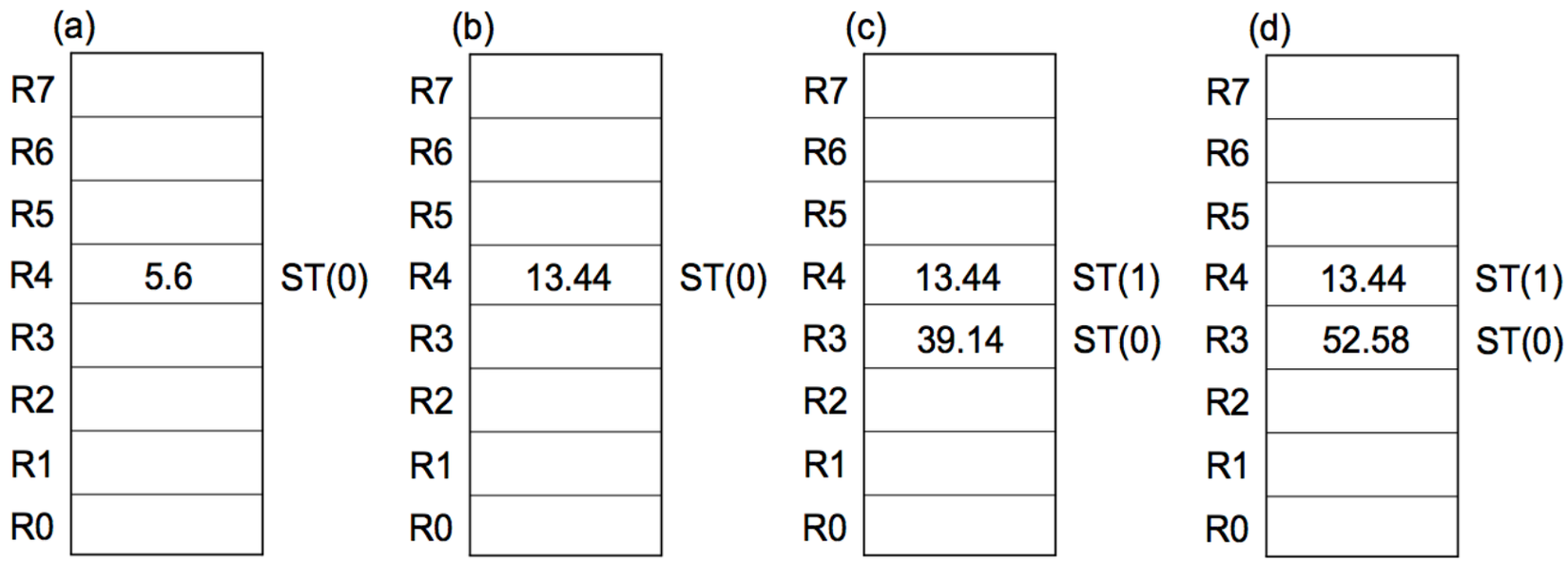


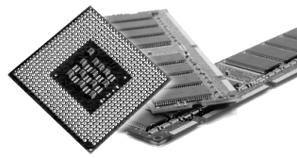
Floating-Point Unit (FPU)

Computation
Dot Product = (5.6 x 2.4) + (3.8 x 10.3)

Code:

```
FLD  value1 ;(a) value1 = 5.6
FMUL value2 ;(b) value2 = 2.4
FLD  value3 ; value3 = 3.8
FMUL value4 ;(c)value4 = 10.3
FADD ST(1)  ;(d)
```

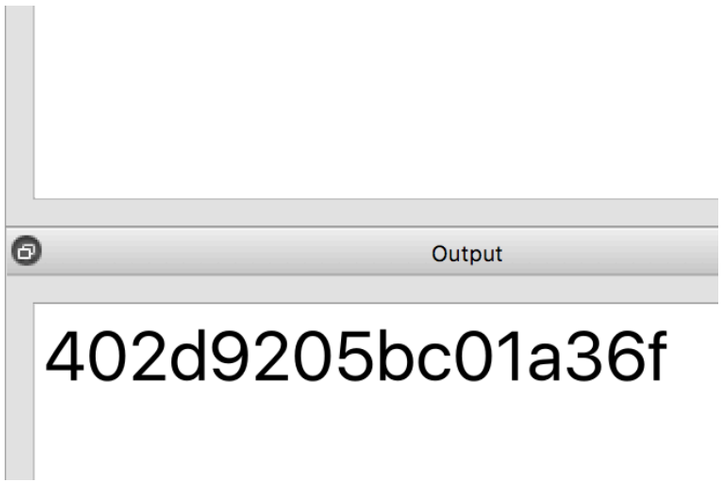


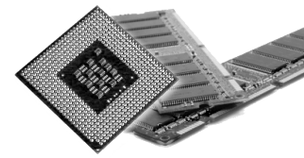


FP Beispiel

```
1  %include "io64.inc"
2  section .data
3
4  z1 DQ 3.33
5  z2 DQ 4.44
6
```

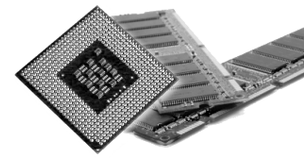
```
fld qword [z1]
fmul qword [z2]
fstp qword [z1]

PRINT_HEX 8, z1
```





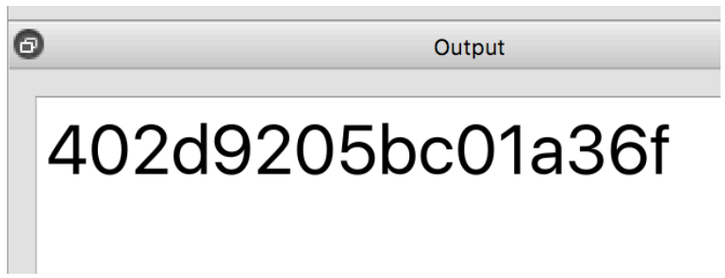
FP Beispiel

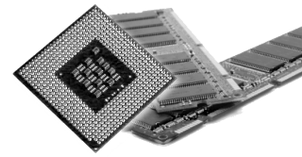
- 3,33 * 4,44 = 14,7852
- 1110,1100100100000010110111100000000011010 00110110111000…
- 1,1101100100100000010110111100000000011010 00110110111000… * 2^3
- Exp.: 3 + 2^(11-1) - 1 = 1026 = 0x402
- Mantisse: 0xd9205bc01a36e…



FP Beispiel

- 3,33 * 4,44 = 14,7852
- 1110,11001001000000101101111000000000011010 00110110111000…
- 1,11011001001000000101101111000000000011010 00110110111000… * 2^3
- Exp.: 3 + 2^(11-1) - 1 = 1026 = 0x402
- Mantisse: 0xd9205bc01a36e…





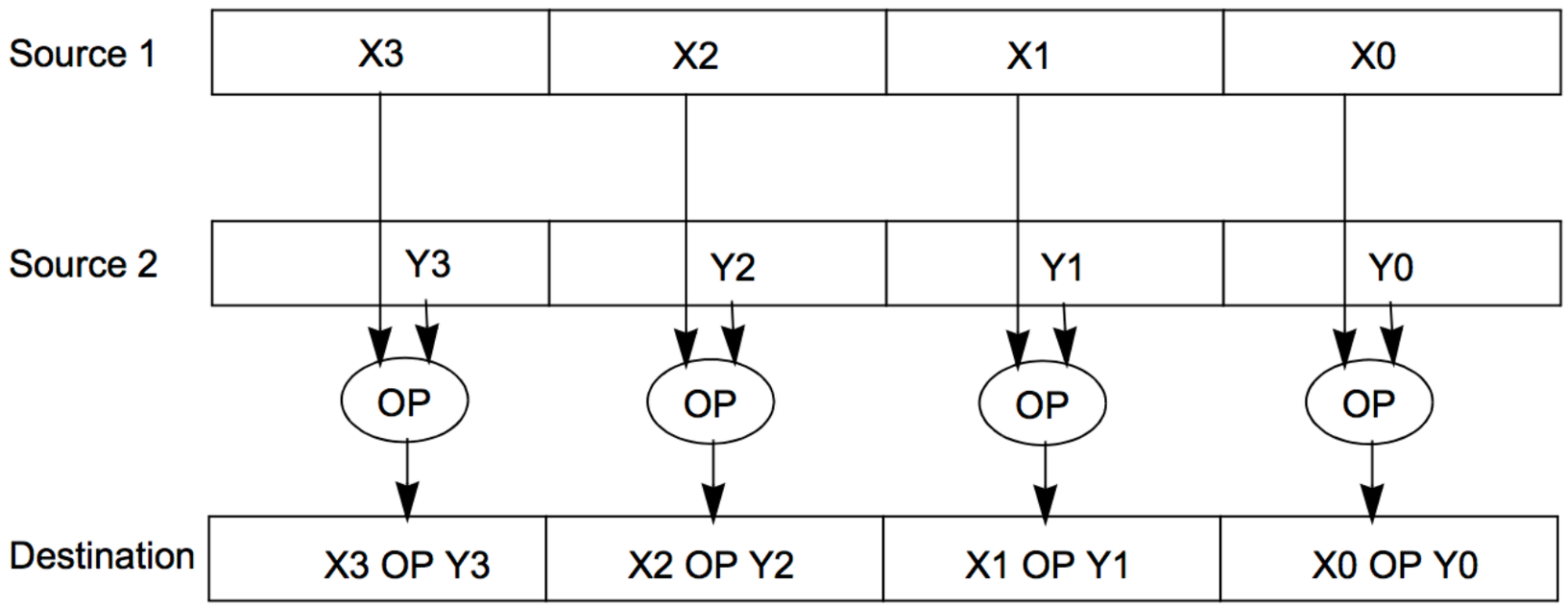
FP Beispiel

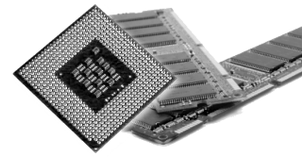
```
17      fld qword [z1]
18      fmul qword [z2]
19      fstp qword [z1]
20
21      push rbp
22
23      mov rdi, fmt
24      movq xmm0, qword [z1]
25      mov rax, 1
26
27      call printf
28
29      pop rbp
30
```

```
14.785200
```

Single Instruction Multiple Data

- 8 MMX Register (64bit)

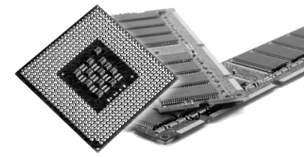




Single Instruction Multiple Data

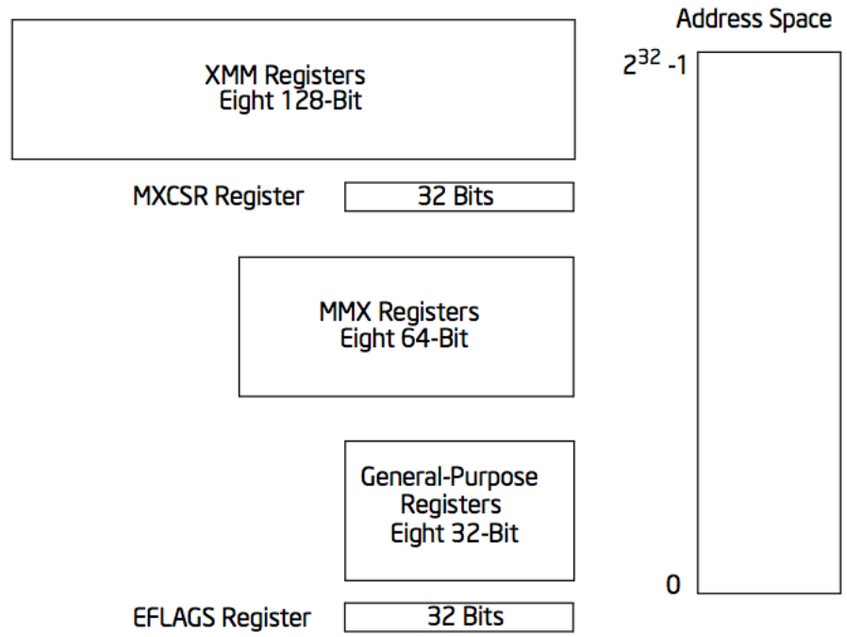
Table 9-2. MMX Instruction Set Summary

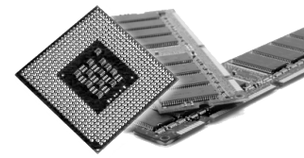
Category		Wraparound	Signed Saturation	Unsigned Saturation
Arithmetic	Addition	PADDB, PADDW, PADDD	PADDSB, PADDSW	PADDUSB, PADDUSW
	Subtraction	PSUBB, PSUBW, PSUBD	PSUBSB, PSUBSW	PSUBUSB, PSUBUSW
	Multiplication	PMULL, PMULH		
	Multiply and Add	PMADD		
Comparison	Compare for Equal	PCMPEQB, PCMPEQW, PCMPEQD		
	Compare for Greater Than	PCMPGTPB, PCMPGTPW, PCMPGTPD		
Conversion	Pack		PACKSSWB, PACKSSDW	PACKUSWB
Unpack	Unpack High	PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ		
	Unpack Low	PUNPCKLBW, PUNPCKLWD, PUNPCKLDQ		
		Packed		**Full Quadword**
Logical	And			PAND
	And Not			PANDN
	Or			POR
	Exclusive OR			PXOR
Shift	Shift Left Logical	PSLLW, PSLLD		PSLLQ
	Shift Right Logical	PSRLW, PSRLD		PSRLQ
	Shift Right Arithmetic	PSRAW, PSRAD		
		Doubleword Transfers		**Quadword Transfers**
Data Transfer	Register to Register	MOVD		MOVQ
	Load from Memory	MOVD		MOVQ
	Store to Memory	MOVD		MOVQ
Empty MMX State		**EMMS**		



Streaming SIMD extensions

- Intel SSE
- 2D, 3D Grafik, Video, Bild, Audio, … Berechnung
- 8 * 128bit XMM Register (x64: 16 * 128bit)
- z.B. 4 * single precision floating point





Advanced Vector Extensions

- AVX / AVX2 / AVX-512
- SIMD mit 128bit, bzw. 256bit oder 512bit
- YMM Register
  - 16 * 256bit (x64)

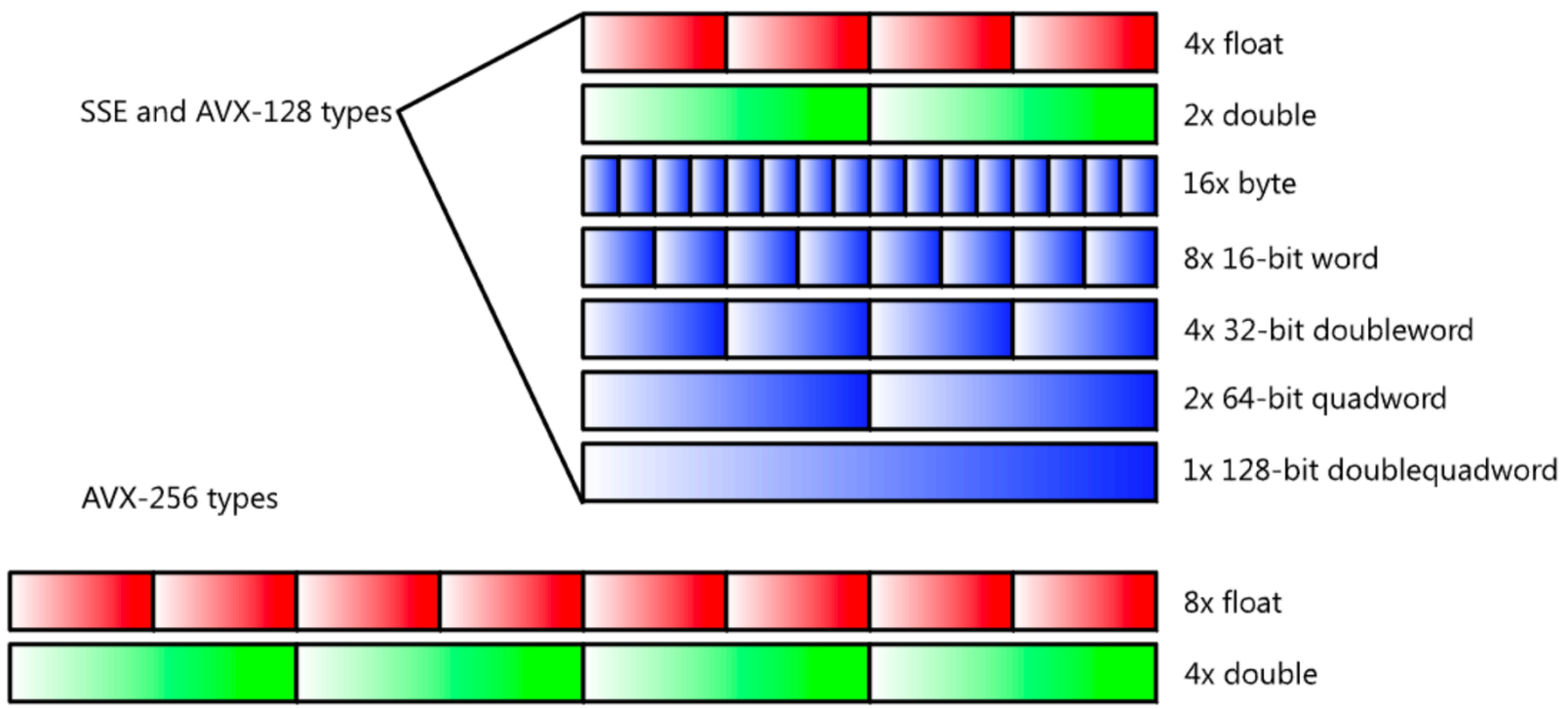
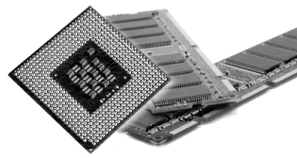
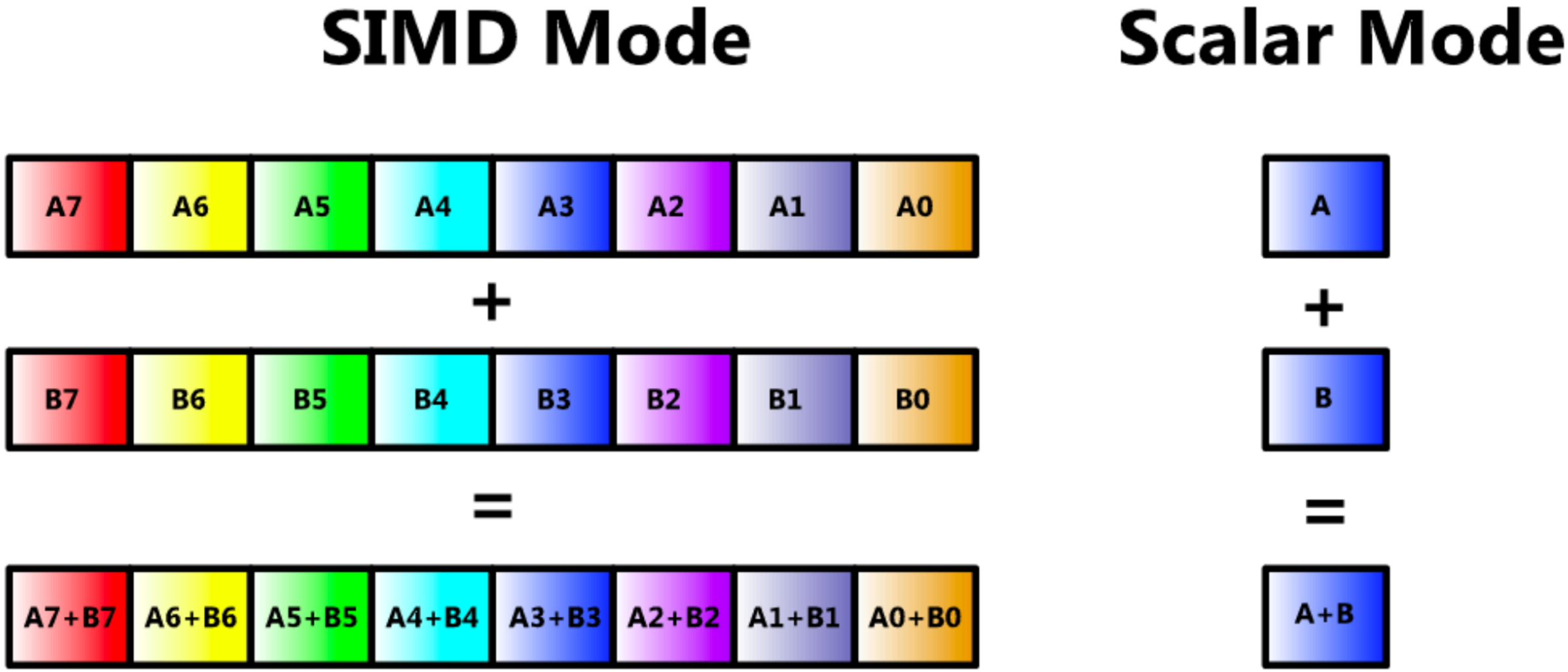
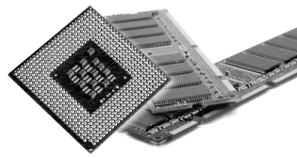


Figure 2. *Intel® AVX and Intel® SSE data types*

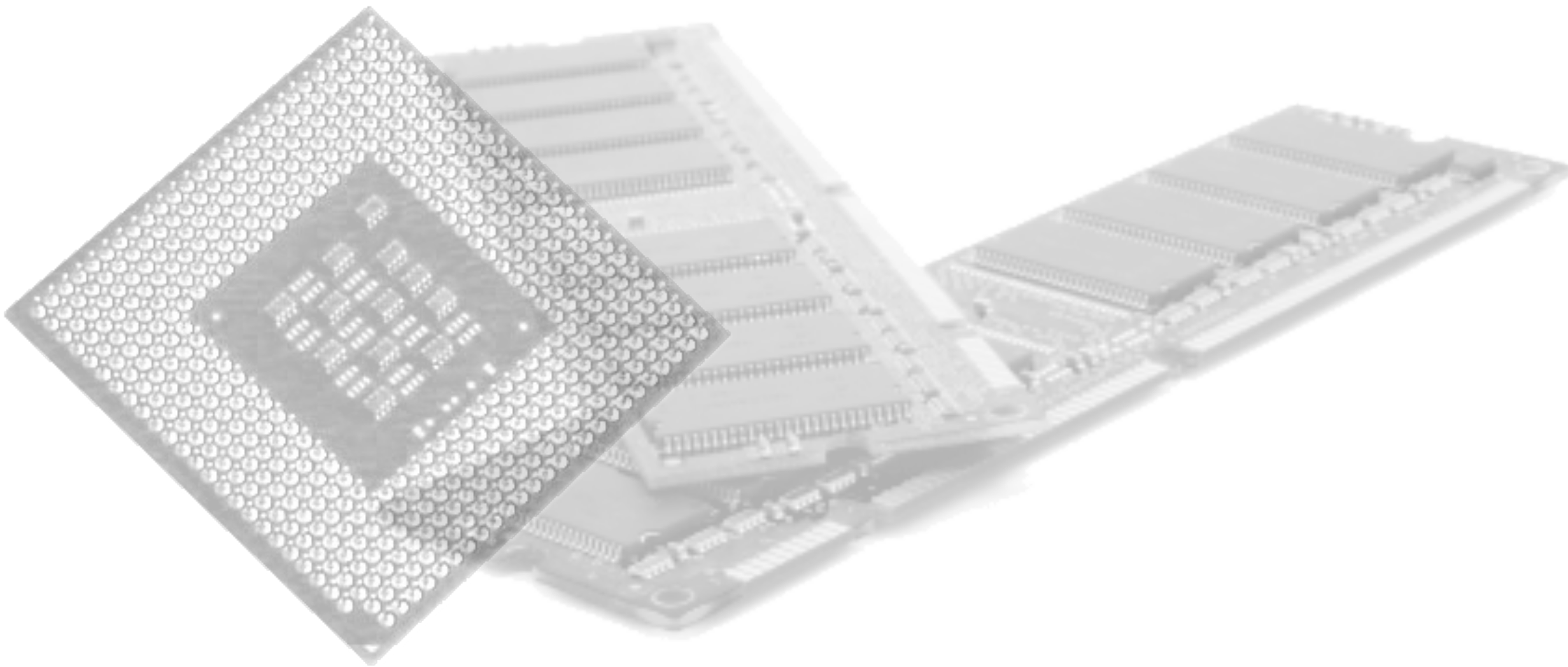


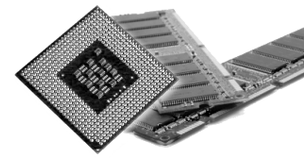
Advanced Vector Extensions





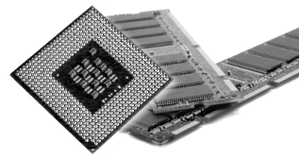
Inline Assembler





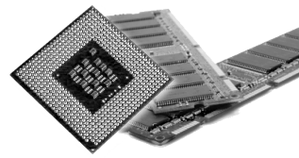
Assembler in C

```
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
```



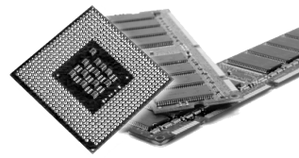
asm

- asm (code : output operand list : input operand list : clobber list);



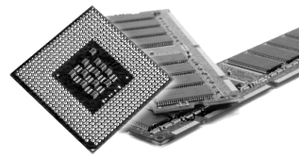
asm

- asm (code : output operand list : input operand list : clobber list);
- **code**: Assember Befehle:
  - "add %0, %1 \n"
  - "inc %1"
- Mehrere Zeilen mit \n



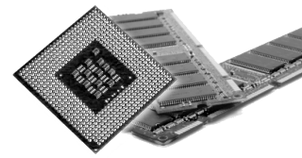
asm

- asm (code : output operand list : input operand list : clobber list);
- **output operand list:**
  - "+r"(Var1),"+r"(Var2)
- Mit Komma getrennt
- + steht für read and write, = overwrite
- r für Register, m für Memory, usw.
- Var1 und Var2 sind C Variablen



asm

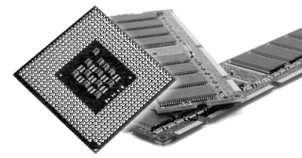
- asm (code : output operand list : input operand list : clobber list);
- **input operand list:**
  - "r"(Var3)
  - Mit Komma getrennt
  - r für Register, m für Memory, usw.
  - Var3 ist eine C Variable
  - Die Operanden können mittels %0, %1, %2, … verwendet werden



asm

- asm (code : output operand list : input operand list : clobber list);

- **clobber list:**
  - Können Register angegeben werden, welche für den Assembler Code frei gelassen werden sollen
  - Die Register werden für keine input oder output Variablen verwendet



Asm Beispiel

- long Var1 = 0xf;
- long Var2 = 0xf;
- **asm ("add %0, %1 \n"**
- **"dec %0”**
- **:"+r"(Var1) : "r"(Var2));**
- printf("%lx \n",Var1);

```c
#include <stdio.h>
int main(){

    long Var1 = 0xf;
    long Var2 = 0xf;

    asm ("add %0,%1 \n"
         "dec %0 \n"
         :"+r"(Var1):"r"(Var2));

    printf("%lx \n",Var1);

    return 0;
}
```

```
1d
Program ended with exit code: 0
```