Rechnerarchitekturen 1*

X86-Assembler

Prof. Dr. Alexander Auch

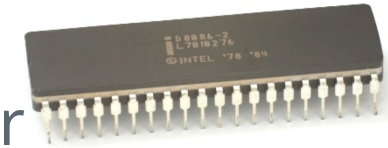
*Teilweise entnommen aus Patterson & Hennessy

Ziele der Veranstaltung

- **Rechnerentwurf**:
  - Prozessor, Speicher, Ein-/Ausgabe
  - Entwurfs- und Optimierungsmöglichkeiten
- **Prozessorentwurf**:
  - Befehlsverarbeitung
  - Entwurfs- und Optimierungsmöglichkeiten
- **Assemblerprogrammierung**:
  - im MIPS-Simulator MARS

x86/AMD64-Architektur

- Bekanntester Vertreter einer CISC-Architektur
- 1978: 8086 16-bit-Prozessor
- 1979: 8088 mit 8-bit-Datenbus, erster IBM-PC
- 1985: 80386 als 32-bit-Prozessor



- 1993: Pentium
  superskalare Architektur
  → werden wir bei Pipelining besprechen

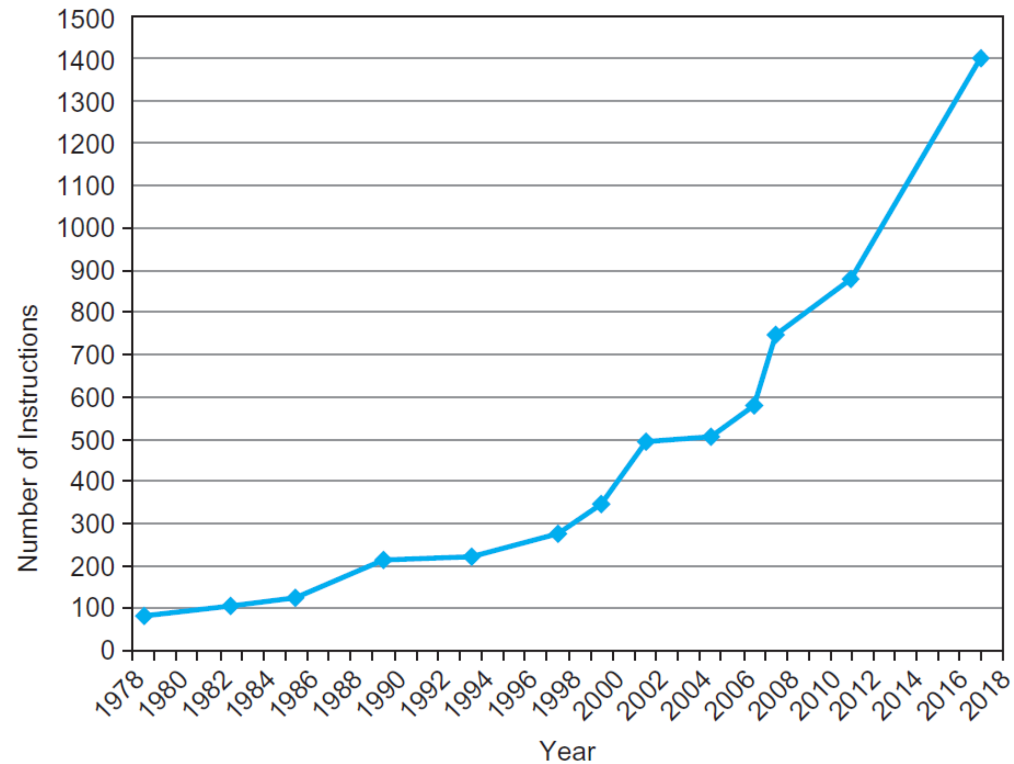
- Heute:
  Core i7, Xeon, AMD Opteron
  64-bit-Architekturen, SIMD
  Intern: RISC mit Microcode

Weiterentwicklung X86

- 2003: AMD64 – Erweiterung auf 64bit
- 2004: EM64T – Extended Memory 64 Technology, SSE3
  Dies wurde auch von Intel übernommen

- 2008: AVX – Advanced Vector Extensions

→ Über die Jahre wurde die ISA immer komplexer:
→ Abwärtskompatibilität



X86 - Register

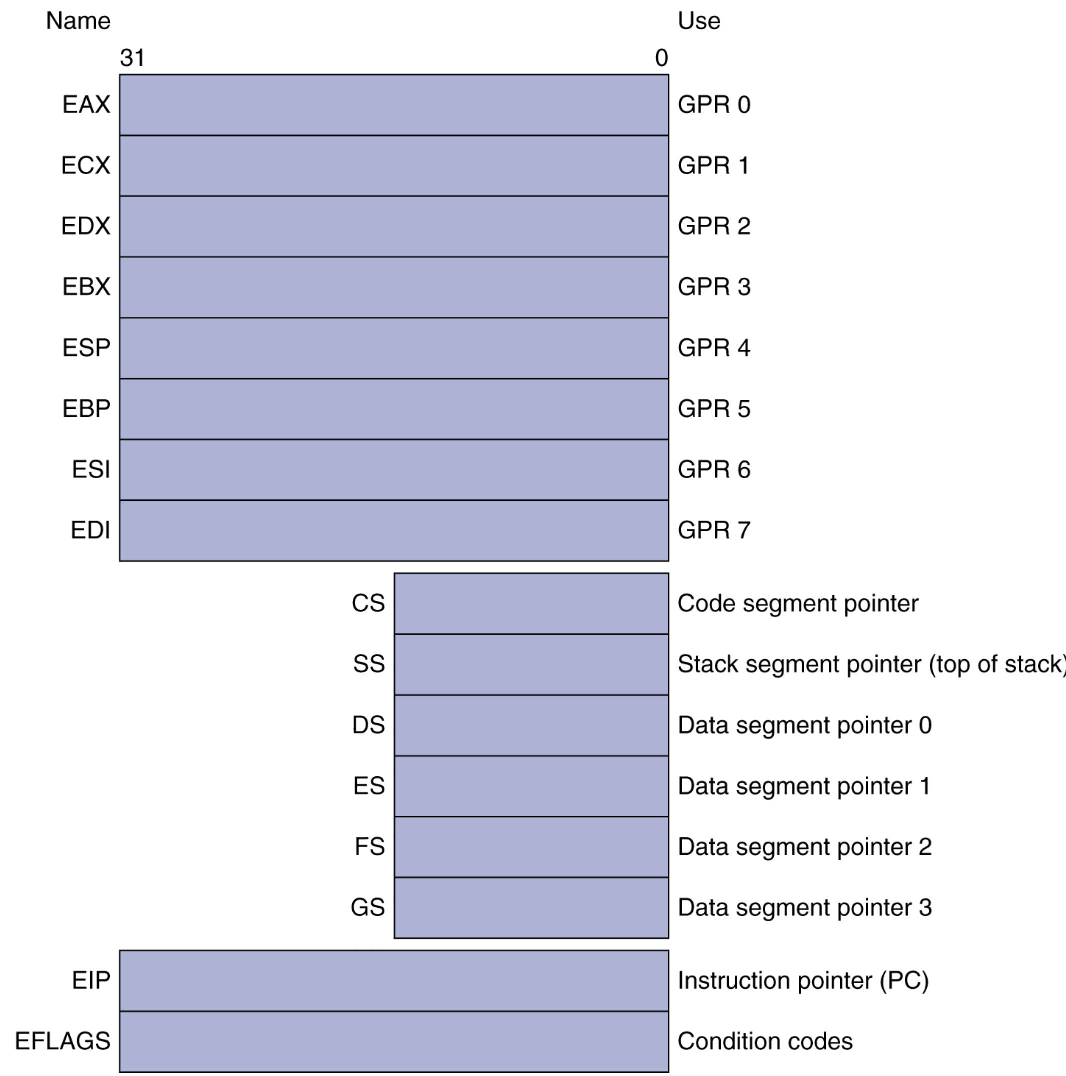
Mit dem 386 wurden die ursprünglich 16bit-Register auf 32bit erweitert

Im Vergleich zu MIPS sind es viel weniger Register

Mit AMD64 wurden 64bit-Register eingeführt: rax,...rdi
Zusätzlich wurden Register r8 bis r15 zugefügt.

Eflags/rflags ist wichtig für Sprunganweisungen:

- OF Overflow Flag
- SF Sign Flag
- ZF Zero Flag
- AF Auxiliary carry/Adjust Flag
- PF Parity Flag
- CF Carry Flag



X86 – Adressierungsmodi

- Zwei Operanden pro Instruktion:

Source/dest operand	Second source operand
Register	Register
Register	Immediate
Register	Memory
Memory	Register
Memory	Immediate

- Memory addressing modes
  - Address in register
  - Address = R_{base} + displacement
  - Address = $R_{base} + 2^{scale} \times R_{index}$ (scale = 0, 1, 2, or 3)
  - Address = $R_{base} + 2^{scale} \times R_{index}$ + displacement

X86 – Adressierungsmodi

Mode	Description	Register restrictions	MIPS equivalent
Register indirect	Address is in a register.	Not ESP or EBP	`lw $s0,0($s1)`
Based mode with 8- or 32-bit displacement	Address is contents of base register plus displacement.	Not ESP	`lw $s0,100($s1) # <= 16-bit # displacement`

Base plus scaled index	The address is Base + (2^{Scale} x Index) where Scale has the value 0, 1, 2, or 3.	Base: any GPR Index: not ESP	`mul $t0,$s2,4`
			`add $t0,$t0,$s1`
			`lw $s0,0($t0)`

Base plus scaled index with 8- or 32-bit displacement	The address is Base + (2^{Scale} x Index) + displacement where Scale has the value 0, 1, 2, or 3.	Base: any GPR Index: not ESP	`mul $t0,$s2,4`
			`add $t0,$t0,$s1`
			`lw $s0,100($t0) #<=16-bit # displacement`

FIGURE 2.38 x86 32-bit addressing modes with register restrictions and the equivalent MIPS code. The Base plus Scaled Index addressing mode, not found in ARM or MIPS, is included to avoid the multiplies by 4 (scale factor of 2) to turn an index in a register into a byte address (see Figures 2.25 and 2.27). A scale factor of 1 is used for 16-bit data, and a scale factor of 3 for 64-bit data. A scale factor of 0 means the address is not scaled. If the displacement is longer than 16 bits in the second or fourth modes, then the MIPS equivalent mode would need two more instructions: a lui to load the upper 16 bits of the displacement and an add to sum the upper address with the base register $s1. (Intel gives two different names to what is called Based addressing mode—Based and Indexed—but they are essentially identical and we combine them here.)

Patterson/Hennessy

Die wichtigsten X86 – Befehle

Instruction	Function
je name	if equal(condition code) {EIP=name}; EIP-128 <= name < EIP+128
jmp name	EIP=name
call name	SP=SP-4; M[SP]=EIP+5; EIP=name;
movw EBX,[EDI+45]	EBX=M[EDI+45]
push ESI	SP=SP-4; M[SP]=ESI
pop EDI	EDI=M[SP]; SP=SP+4
add EAX,#6765	EAX= EAX+6765
test EDX,#42	Set condition code (flags) with EDX and 42
movsl	M[EDI]=M[ESI]; EDI=EDI+4; ESI=ESI+4

FIGURE 2.39 Some typical x86 instructions and their functions. A list of frequent operations appears in Figure 2.40. The CALL saves the EIP of the next instruction on the stack. (EIP is the Intel PC.)

Patterson/Hennessy

Die wichtigsten X86 – Befehle

Instruction	Meaning
Control	**Conditional and unconditional branches**
jnz, jz	Jump if condition to EIP + 8-bit offset; JNE (forJNZ), JE (for JZ) are alternative names
jmp	Unconditional jump—8-bit or 16-bit offset
call	Subroutine call—16-bit offset; return address pushed onto stack
ret	Pops return address from stack and jumps to it
loop	Loop branch—decrement ECX; jump to EIP + 8-bit displacement if ECX ≠ 0
Data transfer	**Move data between registers or between register and memory**
move	Move between two registers or between register and memory
push, pop	Push source operand on stack; pop operand from stack top to a register
les	Load ES and one of the GPRs from memory
Arithmetic, logical	**Arithmetic and logical operations using the data registers and memory**
add, sub	Add source to destination; subtract source from destination; register-memory format
cmp	Compare source and destination; register-memory format
shl, shr, rcr	Shift left; shift logical right; rotate right with carry condition code as fill
cbw	Convert byte in eight rightmost bits of EAX to 16-bit word in right of EAX
test	Logical AND of source and destination sets condition codes
inc, dec	Increment destination, decrement destination
or, xor	Logical OR; exclusive OR; register-memory format
String	**Move between string operands; length given by a repeat prefix**
movs	Copies from string source to destination by incrementing ESI and EDI; may be repeated
lods	Loads a byte, word, or doubleword of a string into the EAX register

Patterson/Hennessy

X86 – Instruction Encoding

Befehlslänge ist unterschiedlich

Prefix-Bytes und Postfix-Bytes

- Postfix für Adressierungsmodi
- Prefixe ändern Ausführung (Loops, Operandenlänge, ...)

1 bis 17 Bytes