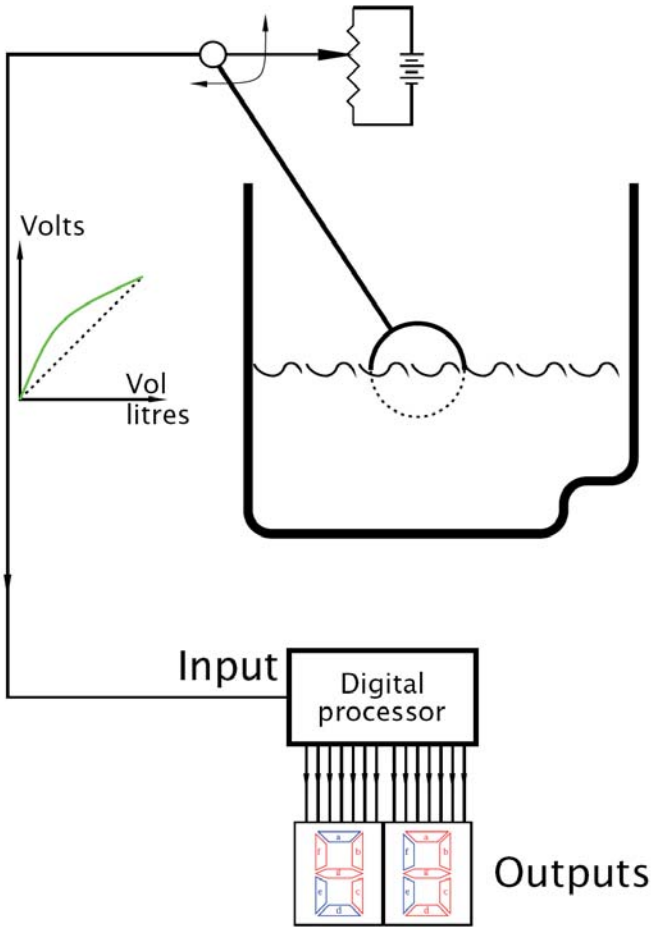


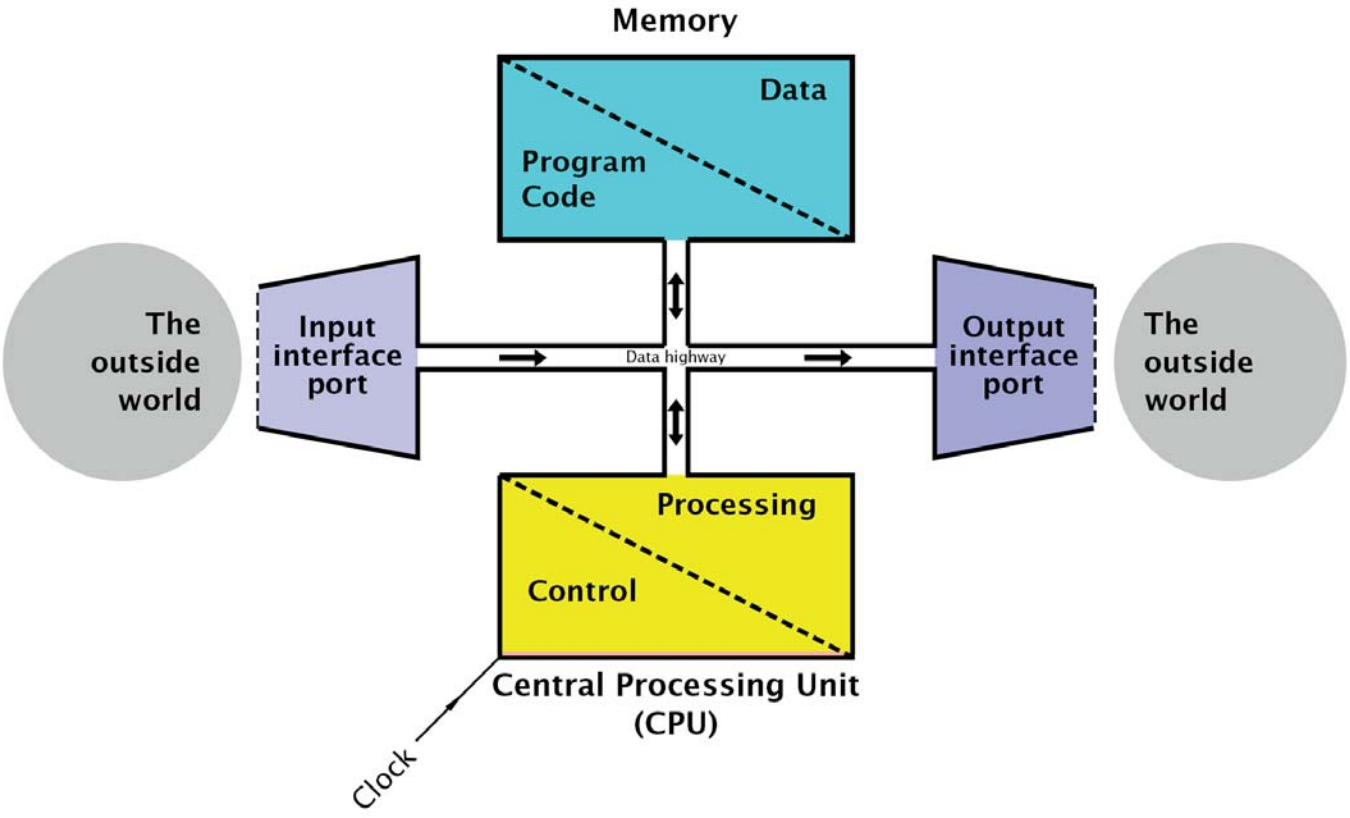


Figure 2: The von-Neumann computer structure.

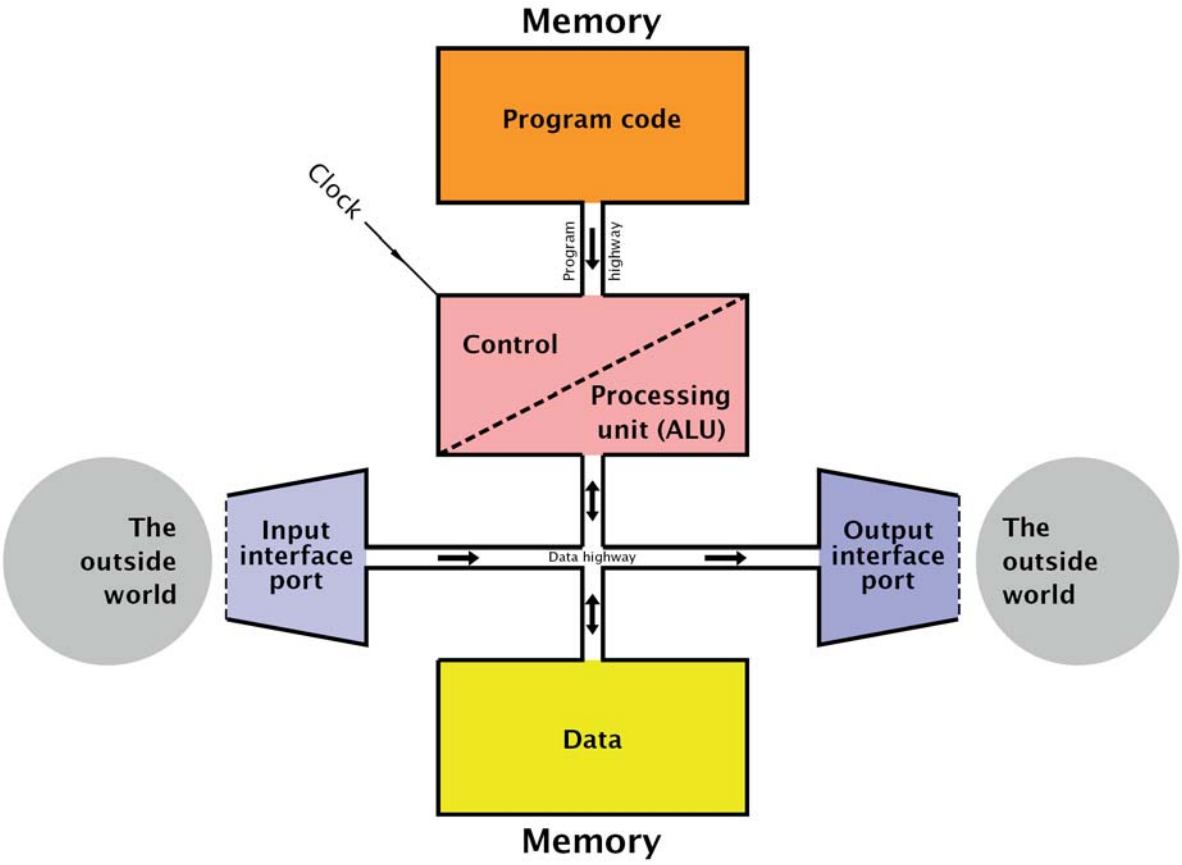
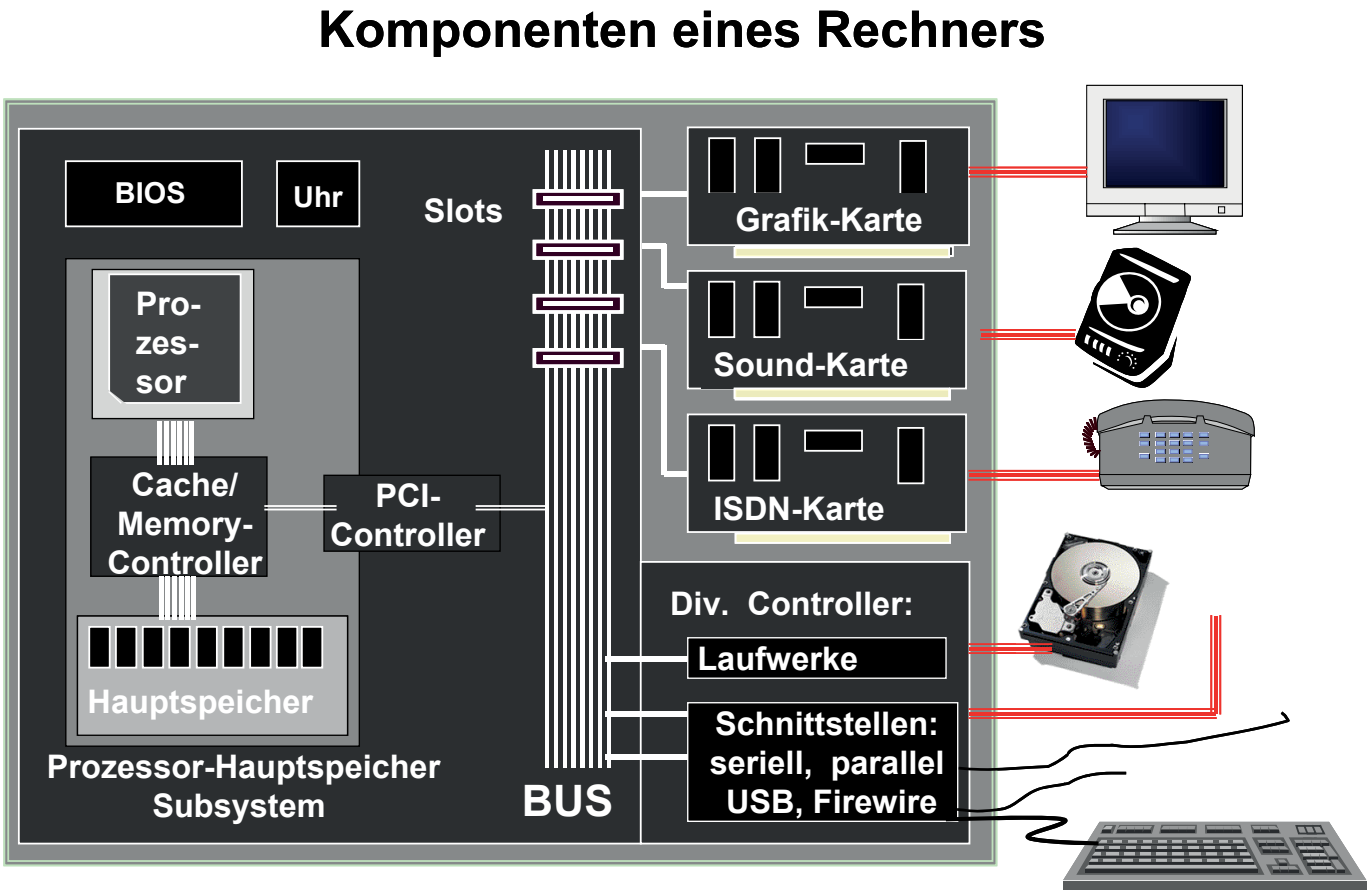


Figure 3: A Harvard architecture-based computer.



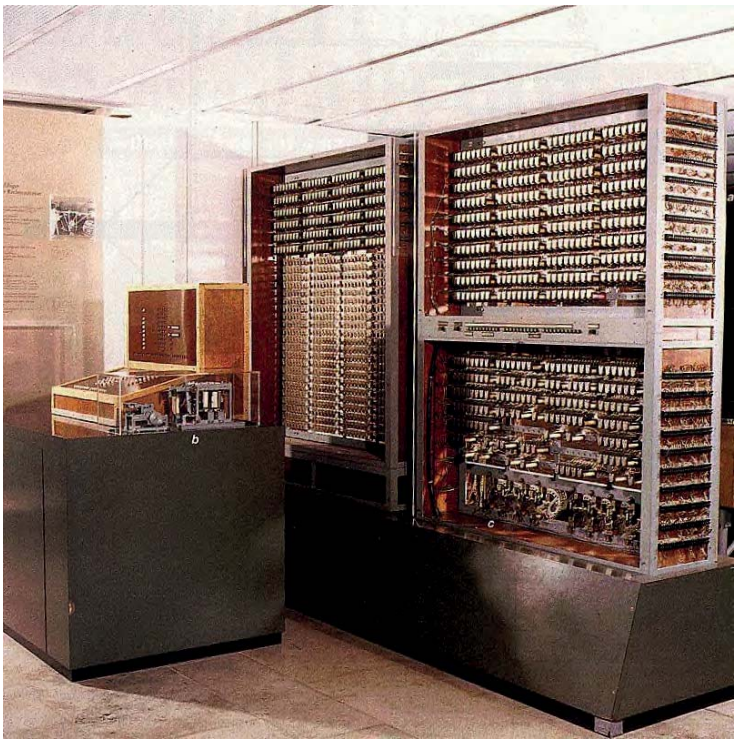
Mikroprozessoren:

Historische Anfänge:

Z3:

- 600 Relais im Rechenwerk
  1400 Relais im Speicherwerk
- Erster vollfunktionsfähiger 22 bit Rechenautomat, Programmgesteuert
- Eine Multiplikation benötigte 3s!

Ein 32 bit Prozessor erledigt dies heute in weniger als 10 ns!



Der Großrechner

Er suchte das "mechanische Gehirn", heraus kam ein Computer. Vor 100 Jahren wurde Konrad Zuse geboren: Er baute den ersten frei programmierbaren Rechner der Welt, doch ein deutscher Bill Gates wurde nie aus ihm.

http://www.sueddeutsche.de/digital/computerpionier-konrad-zuse-der-grossrechner-1.963104

Konrad Ernst Otto Zuse

* 22. Juni 1910 in Berlin

† 18. Dezember 1995 in Hünfeld bei Fulda

Deutscher Bauingenieur, Erfinder und Unternehmer (Zuse KG). Mit seiner Entwicklung der Z3 im Jahre 1941 baute er den ersten vollautomatischen, programmgesteuerten und frei programmierten, in binärer Gleitpunktrechnung arbeitenden Computer der Welt.

Z1 - Faulheit als Motor

"Ich war zu faul zum Rechnen"

Die Faulheit erfordert zunächst aber großen Einsatz: 30.000 Bleche vom Altwarenhändler, beweglich gelagert und mit Stiften verbunden, sollen im "mechanischen Gehirn" die logischen Operationen übernehmen. Drei Jahre dauert es, bis sie von Zuses Vater, seinen Freunden und seiner Theatergruppe ausgesägt und montiert sind.

Die Maschine, die dabei herauskommt, ist so groß wie ein Esstisch für acht Personen. Sie ist quadratisch-flach, sie wird von einem Staubsaugermotor angetrieben, sie macht einen Höllenlärm. Kurbeln, Walzen und Stangen aus dem Metallbaukasten rotieren dann um die Wette. Ihre Instruktionen bekommt die Maschine von einem Filmstreifen, den Zuse mit einem Locher malträtierthat.

Meist verklemmen sich die Bleche und verhindern so die Kalkulationen. Kommt die Maschine aber zum Rechnen, dann stimmen die Ergebnisse. Zuse, nie ein Freund allzu großer Bescheidenheit, nennt sein Erstlingswerk Z1, Zuse 1.

Heute bemisst man die maximale Anzahl der bearbeiteten Maschinenbefehle pro Sekunde in MIPS (Million Instructions Per Second) oder GIPS (Giga Instruction Per Second).

Ein anderes Maß:

maximale Anzahl von Gleitkommaoperationen FLOPS, MFLOPS, GFLOPS

Obwohl die Entwicklung der Mikrorechner und Mikroprozessoren rasant ist, hat sich am Grundprinzip kaum etwas geändert:

- Abarbeitung der Befehle durch Mikroprogramm
- Befehle werden durch "Takt" weitergeschaltet

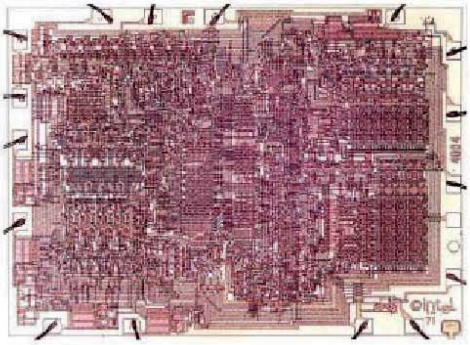
Ein Mikroprozessor "versteht" > 300 Maschinenbefehl: CISC- Rechner (Complex Instruction Set Computer)

Befehle werden in einen Mikrocode umgesetzt und dann in mehreren Taktzyklen abgearbeitet!

Übersicht Prozessorentwicklung

Erster Computer auf einem Chip: Intel 4004

- **4-Bit, 8-Bit Mikroprozessoren**
  - Intel 4004 (~1971)
  - Intel 8008
- **16-Bit Mikroprozessoren**
  - Texas Instruments TMS 9900 (~1977)
  - Intel 8086 (29000 Transistoren, 8 MHz)
    - Erweiterte Akkumulator-Architektur
    - 20 Bit Adressierung durch segmentiertes Adressierungsschema
  - Zilog Z8000
  - Motorola MC68000 (68000 Transitoren, 8 MHz)
    - Mikroprogrammierung, 32 Bit Allzweckregister, 8 Adress- und 8 Datenregister



- 4-bit accumulator architecture
- 8µm pMOS
- 2,300 transistors
- 3 x 4 mm²
- 750kHz clock
- 8-16 cycles/inst.

(~1978-1980)

4

Beispiel Intel-Familie:

Name	Date	Transistors	Microns	Clock speed	Data width	MIPS
8080	1974	6,000	6	2 MHz	8 bits	0.64
8088	1979	29,000	3	5 MHz	16 bits 8-bit bus	0.33
80286	1982	134,000	1.5	6 MHz	16 bits	1
80386	1985	275,000	1.5	16 MHz	32 bits	5
80486	1989	1,200,000	1	25 MHz	32 bits	20
Pentium	1993	3,100,000	0.8	60 MHz	32 bits 64-bit bus	100
Pentium II	1997	7,500,000	0.35	233 MHz	32 bits 64-bit bus	~300
Pentium III	1999	9,500,000	0.25	450 MHz	32 bits 64-bit bus	~510
Pentium 4	2000	42,000,000	0.18	1.5 GHz	32 bits 64-bit bus	~1,700

64/32-Bit Mikroprozessoren

ROSS Technologies hyperSPARC, SUN Microsystems superSPARC, Motorola 88110

IBM, Motorola PowerPC 601 (MPC601)

5

Eigenschaften moderner Mikroprozessoren:

64-Bit Struktur

interne Parallelarbeit (Befehls-Pipelining, arithmetisches Pipelining, nebenläufige Abarbeitung von Maschinenbefehlen)

Integration von Befehls- und Datencaches

Virtuelle Speicherverwaltung

Speicher- und Peripherieanbindung

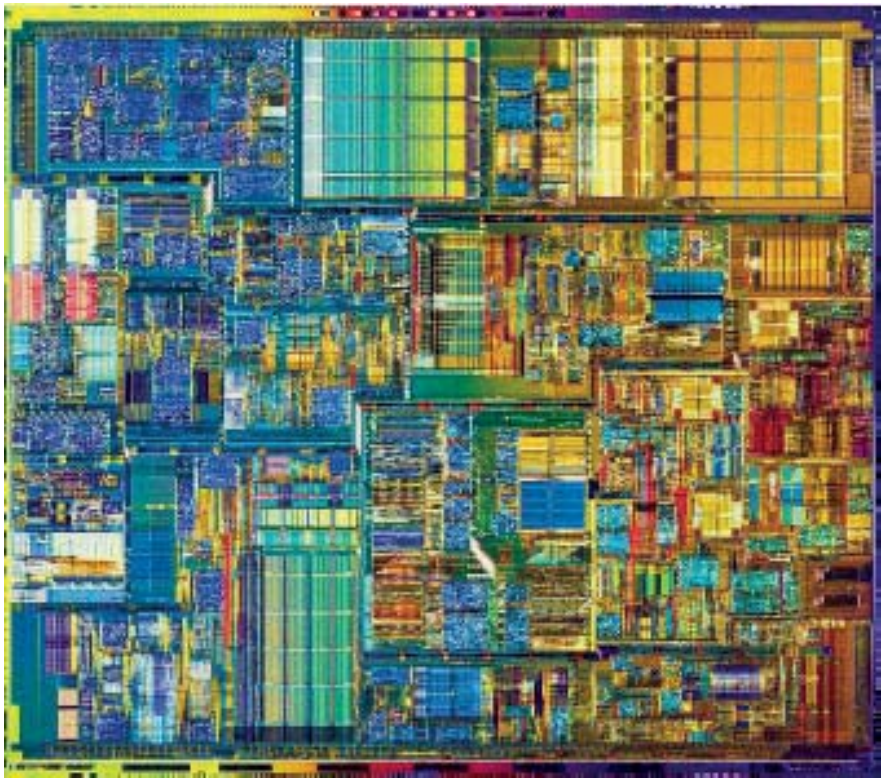
Intel PENTIUM:

- 1993 auf den Markt, schafft zwei Befehle pro Taktzyklus (zwei interne 8 kB cache Speicher)
- externer Datenbus 64 Bit breit
- 1993 → 5 Volt Technik, 1994 → 3,3 Volt Technik
- Pentium Pro → Erstmals ist der second level cache in die CPU integriert

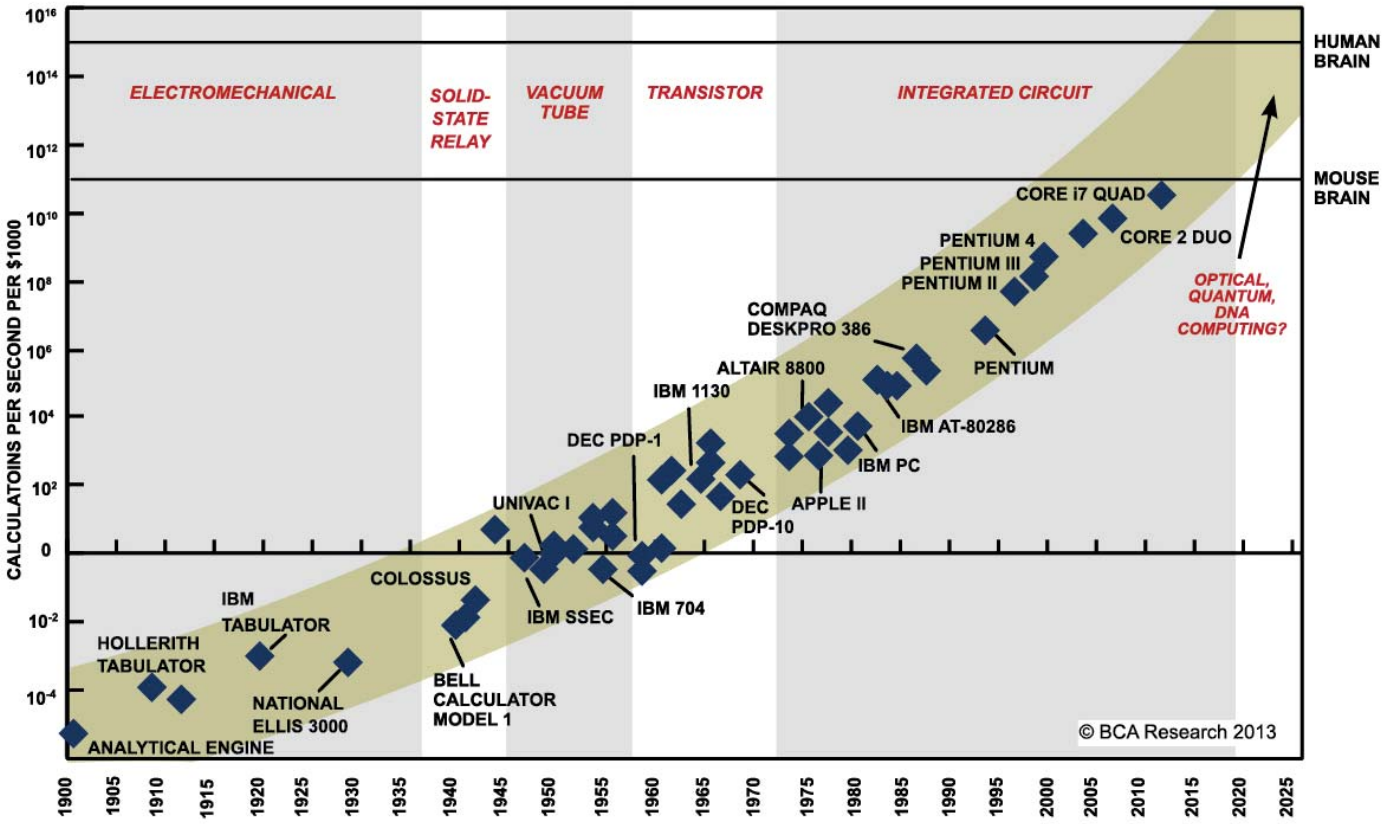
2006 sollen die Prozessoren mit 350 Millionen Transistoren auf dem Markt sein
bis 2011 sollen Prozessoren mit 10-15 GHz Taktung und 1 Milliarde Transistoren Standard werden

Geschichtliche Entwicklung

- **Intel Pentium 4 (2000)**



Moore's Law - Gordon Moore, co-founder of Intel



SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS TRANSCEND BIOLOGY", P.67, *THE VIKING PRESS*, 2006. DATAPOINTS BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

CISC-Architektur (complex instruction set computer):

- komplexe Instruktionen werden direkt verarbeitet
- geringe Zahl Register
- mächtiger Befehlsumfang
- Befehle haben unterschiedliche Größe und bestehen aus mehreren Anweisungen

RISC-Architektur (reduced instruction set computer):

- Befehlsatz beschränkt sich auf wirklich notwendige Befehle
- mehr Register als bei CISC-Architektur
- Herstellungskosten geringer
- Schnellere Befehlsausführung

Kein Etablierung von RISC Prozessoren im PC Bereich wegen der Marktlage
→ workstations, Großrechner

Prozessor	Takt (Mhz)	SPEC fp_Base 2000	Durschn. Befehle pro MHz
Pentium 3	1080	1824	1.689
Pentium 4	1700	3456	2.033
Ultrasparc III	900	3760	4.178
Alpha 21264A	833	4121	4.947

RISC- Prozessoren

Umsetzung des Befehls erfolgt **nicht** über Mikrocode, sondern für jeden Befehl in der Maschinensprache steht ein sequentielles Netzwerk aus Gattern zur Verfügung, das die Ausführung des Befehls in nur einem einzigen Taktzyklus ermöglicht!

⇨ große Zahl von Gattern => reduzierter Befehlssatz

RISC´s arbeiten im allgemeinen nicht stack-orientiert, sondern on-Chip-Register-orientiert (bis zu 1000 eigene Register/Mikroprozessor)

Moderne Prozessoren →

pipelining, branch prediction, Superskalar (mehrere Befehler auf einmal)

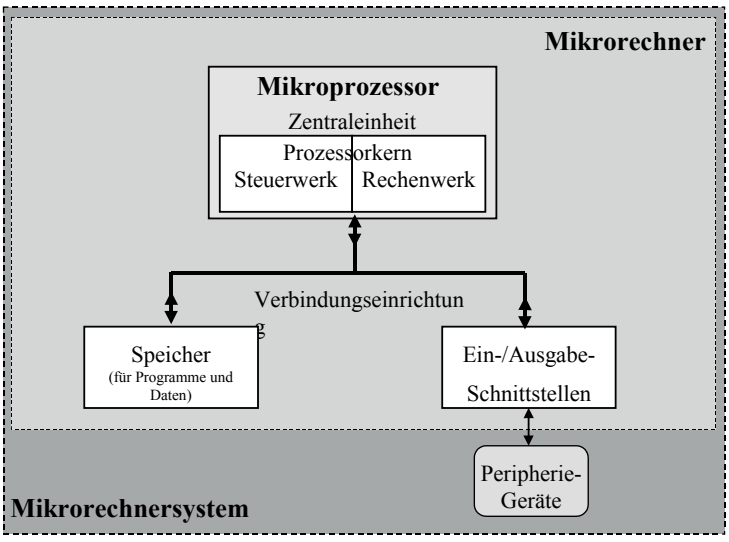
Basiskomponenten eines Mikroprozessors

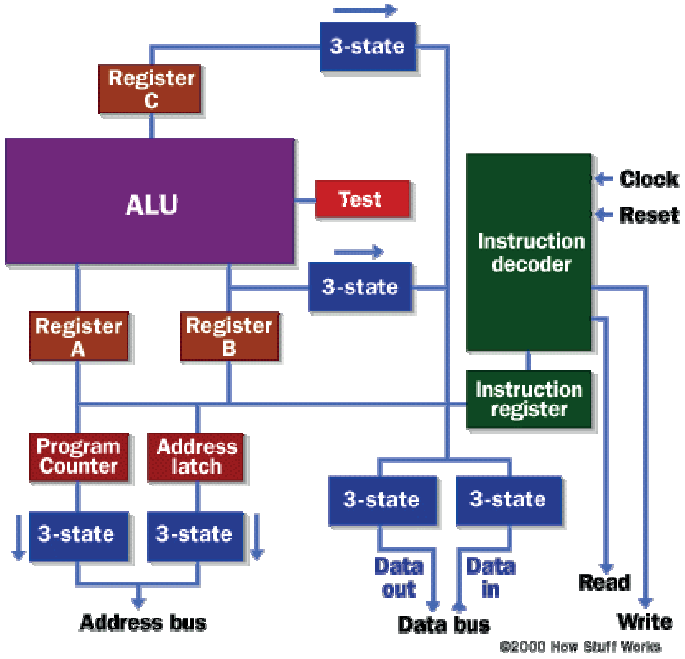
- Rechenwerk
- Steuerwerk
- Schnittstelle zur Außenwelt

Prozessorkern

Weitere Komponenten (je nach Komplexität)

- Cache
- Virtuelle Speicherverwaltung





Prozessor Komponenten:

BIU: realisiert Datentransfer zwischen Hauptspeicher und Prozessorregister

IP: Instruction pointer

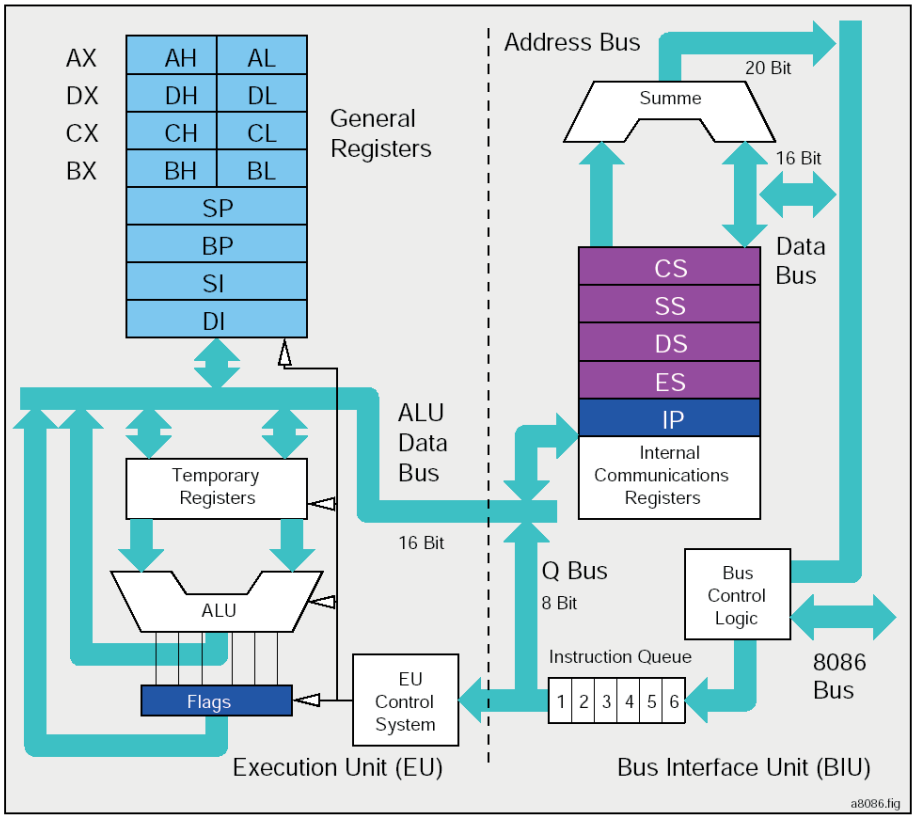
instruction queue:
Befehls-Warteschlange

ALU: arithmetic logic unit

EU-Control System:
Kontroll und Zeitsteuerungs-Einheit

Akku: Akkumulator,
spez. Register für ALU

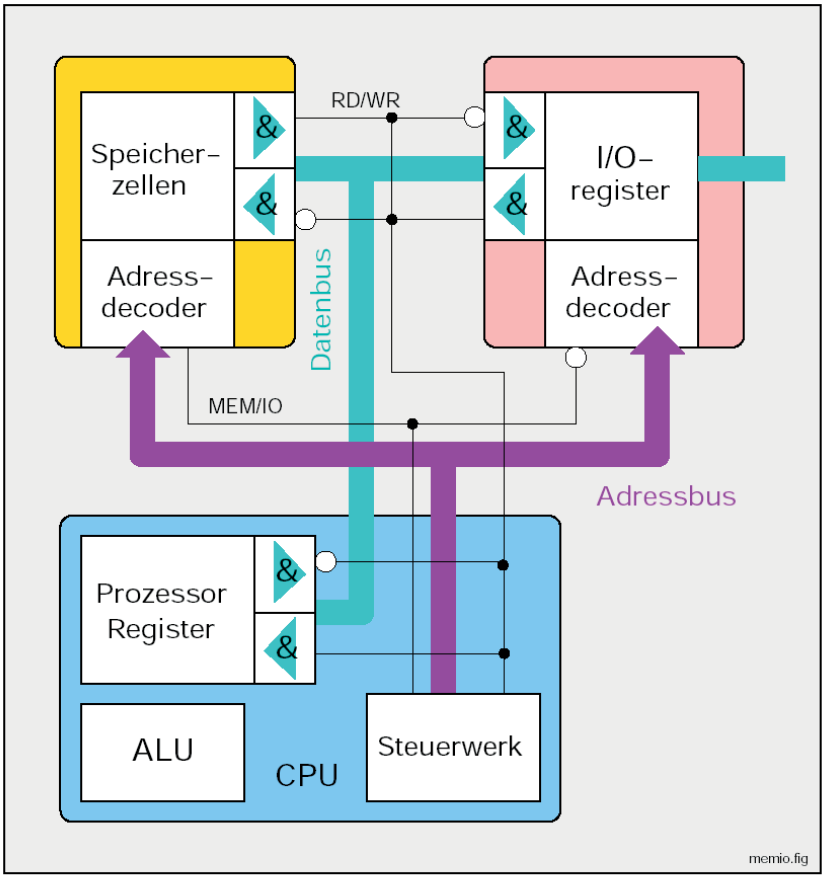
Flags: Ausgabekanäle der ALU für schnelle Entscheidungen
Carry- Flag, Overflow- Flag, Zero-Flag, Sign-Flag etc.



Unterscheidung der Speicherobjekte durch ihre Adresse

Ein- und Ausgabe erfordert Register (I/O-Ports)

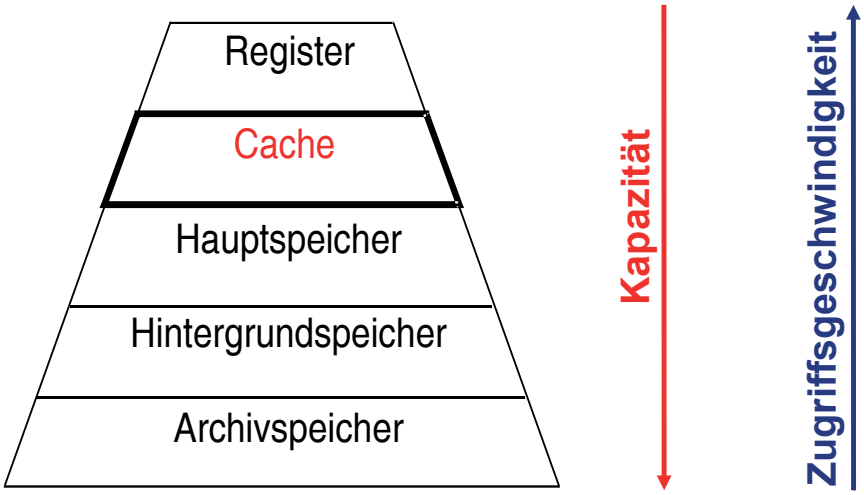
I/O-Ports sind ebenfalls durch Adressen zu unterscheiden



Zusammenwirken CPU – Speicher – Peripherie

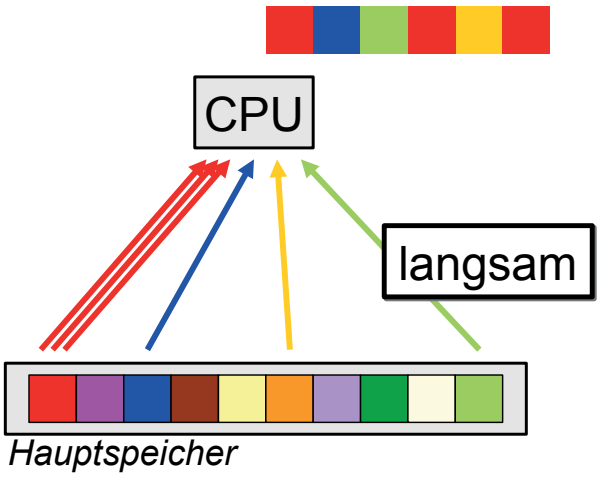
Speicherhierarchie

- Ausnützen der Lokalitätseigenschaft von Programmen
- Kompromiß zwischen Preis und Leistungsfähigkeit
- Hierarchie von Speicherkomponenten:
  - Speicherkomponenten mit unterschiedlichen Geschwindigkeiten und Kapazitäten

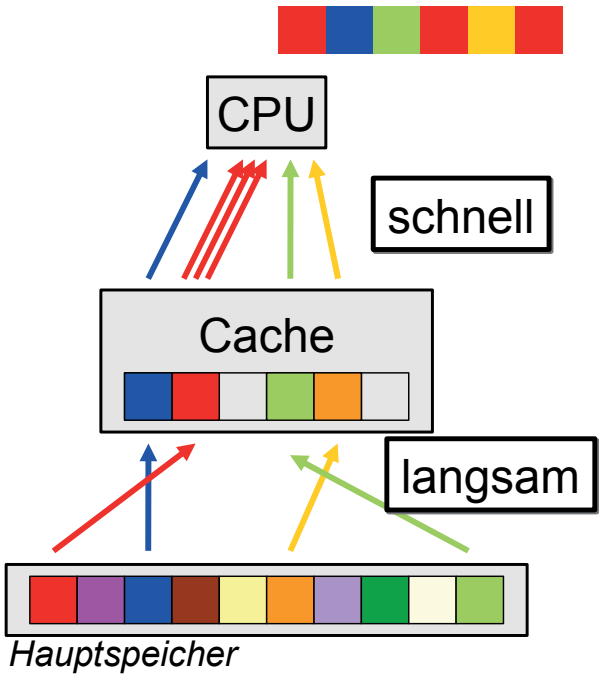


Grundprinzip Cachespeicher

Ohne Cache



Mit Cache



- Redundante Transaktionen
  Rote Blöcke mehrfach!

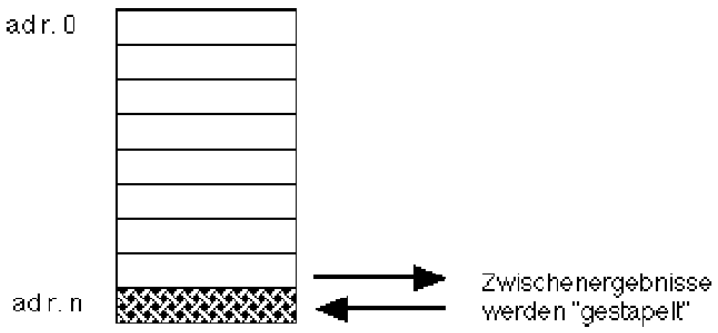
Grundprinzip Cachespeicher

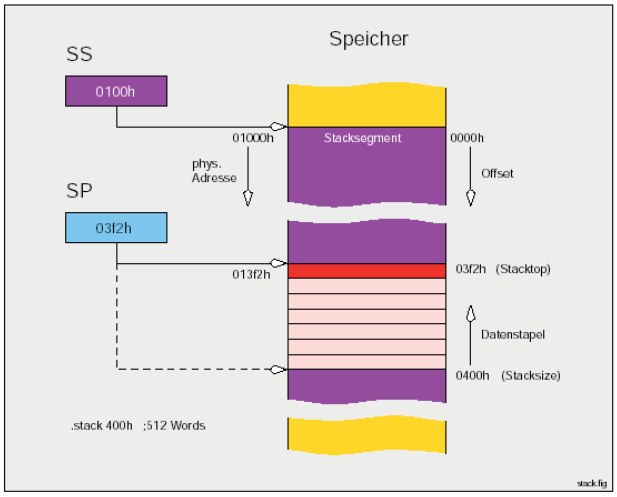
- Schnelle Speicher “in CPU-Nähe”
  - Unterschiedliche Speichertechnologie
  - SRAM (teurer, aber schneller) statt DRAM
- Bei Bedarf werden Daten (oder Instruktionen) in den Cache geladen
  - Zur späteren Wiederverwendung
  - Keine langsamen Zugriffe
- Wichtigste Eigenschaft: Transparenz
  - Benutzer sieht den Cache nicht
  - Lediglich Einfluß auf Geschwindigkeit

Digitalelektronikkomponenten im Prozessor:

- Register → D-latches (FLIP-FLOPs)
- Programmzähler → latch, welches set- und reset-Eingang hat
- tri-state buffer → CMOS Schaltung, welche 0,1 und hochohmig gesetzt werden kann
- ALU → Addierer/Subtrahierer, Schieberegister (Multiplizieren)
- Halbleiterspeicher (SRAM)
- Decoder

Stack:
Viele Mikroprozessoren arbeiten stack-orientiert → Zwischenergebnisse werden "gestapelt"





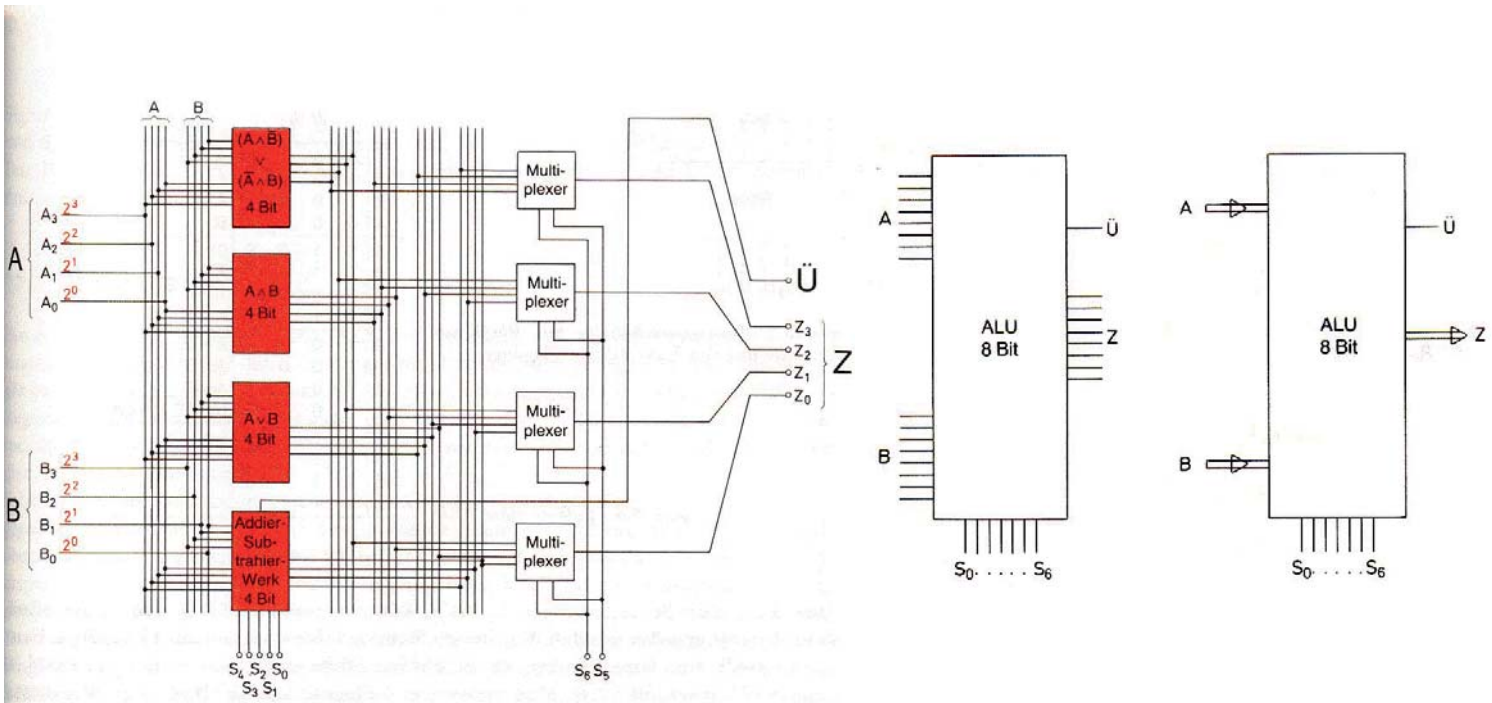
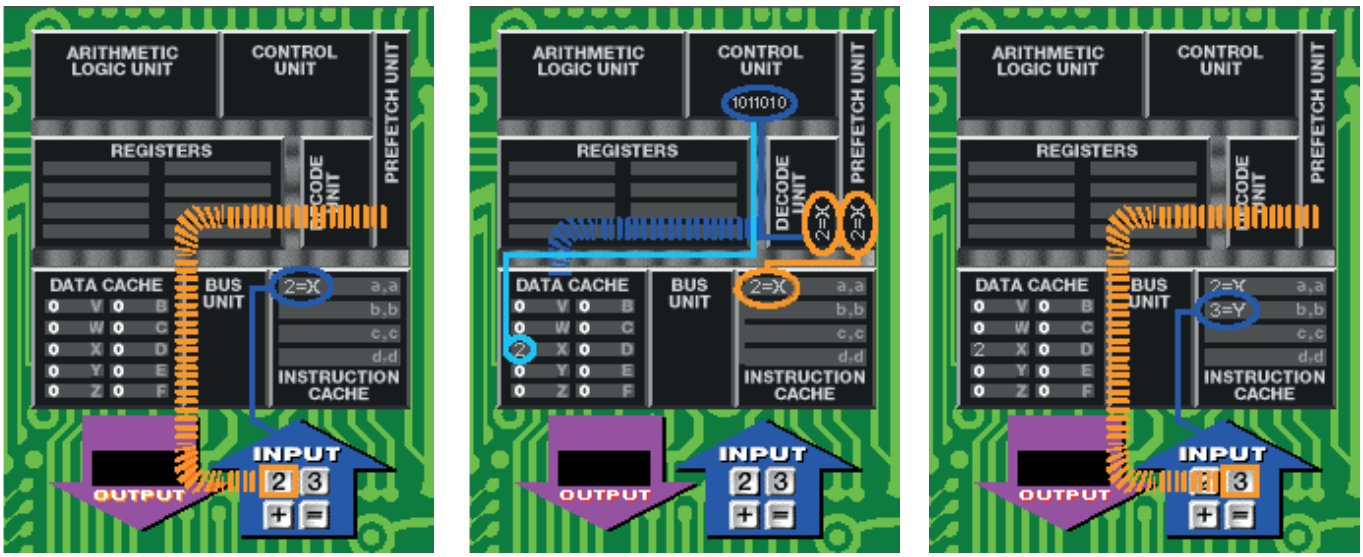
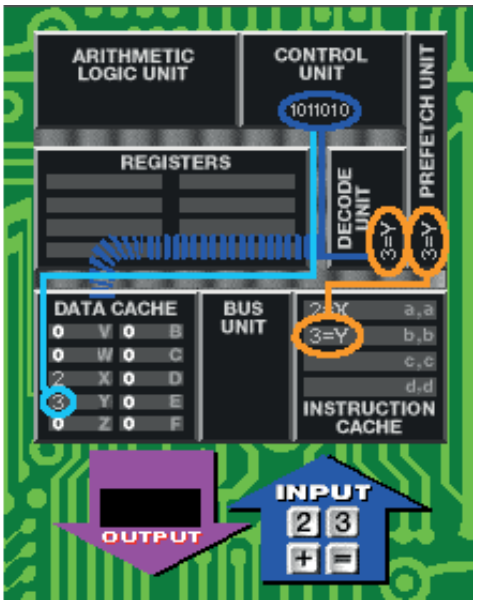
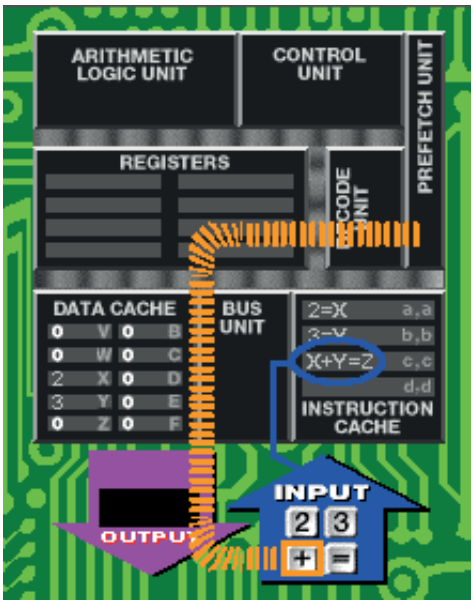


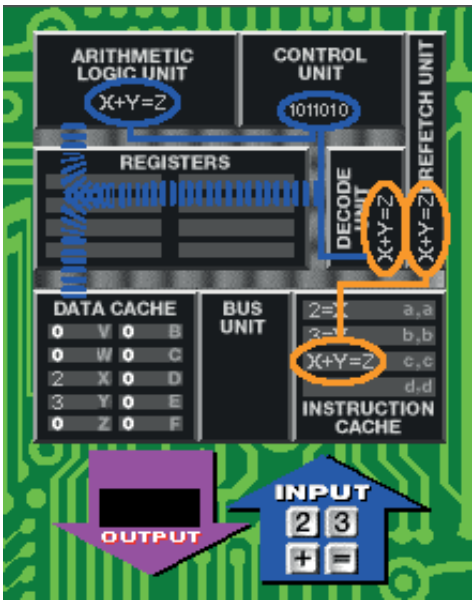
Bild 15.3 Schaltbild einer Arithmetisch-logischen Einheit für 4 Bit

Vereinfachte Darstellung eines Befehlsablauf am Beispiel 2+3 =5:

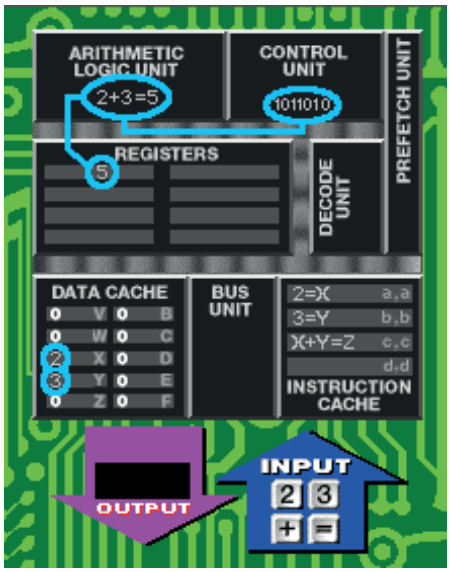


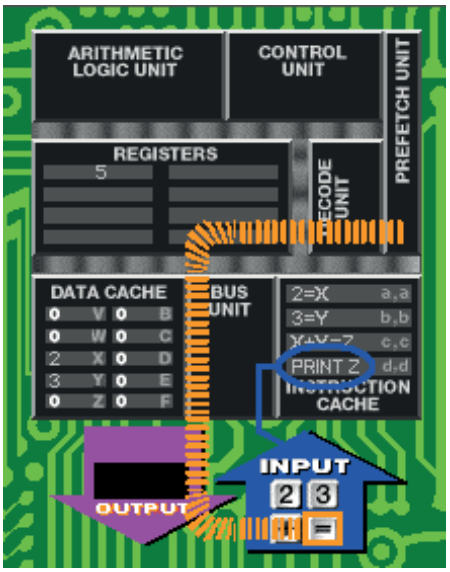

ARITHMETIC LOGIC UNIT
CONTROL UNIT
1011010
REGISTERS
DECODE UNIT
PREFETCH UNIT
3=Y
3=Y
DATA CACHE
BUS UNIT
2=X a,a
3=Y b,b
c,c
d,d
INSTRUCTION CACHE
OUTPUT
INPUT
2 3
+ =

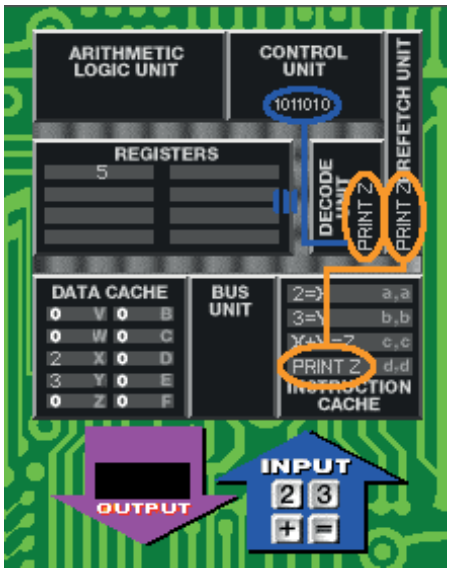

ARITHMETIC LOGIC UNIT
CONTROL UNIT
REGISTERS
PREFETCH UNIT
DATA CACHE
BUS UNIT
2=X a,a
X+Y=Z c,c
d,d
INSTRUCTION CACHE
OUTPUT
INPUT
2 3
+ =

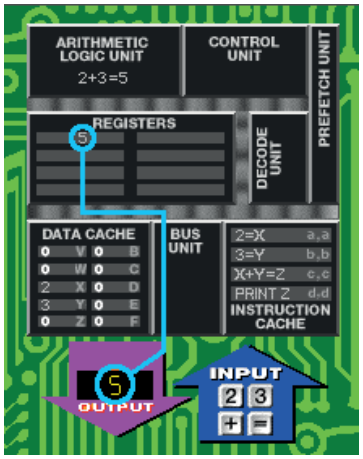

ARITHMETIC LOGIC UNIT
X+Y=Z
CONTROL UNIT
1011010
REGISTERS
DECODE UNIT
PREFETCH UNIT
X+Y=Z
X+Y=Z
DATA CACHE
BUS UNIT
2=X a,a
b,b
X+Y=Z c,c
d,d
INSTRUCTION CACHE
OUTPUT
INPUT
2 3
+ =

15

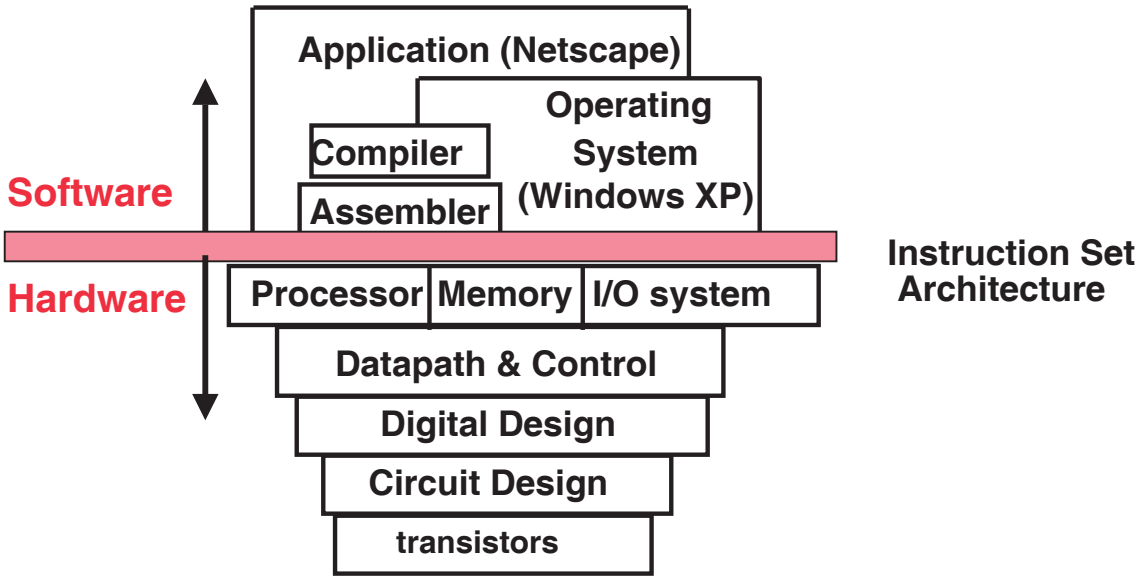

ARITHMETIC LOGIC UNIT
2+3=5
CONTROL UNIT
1011010
REGISTERS
5
DECODE UNIT
PREFETCH UNIT
DATA CACHE
BUS UNIT
2=X a,a
3=Y b,b
X+Y=Z c,c
d,d
INSTRUCTION CACHE
OUTPUT
INPUT
2 3
+ =


ARITHMETIC LOGIC UNIT
CONTROL UNIT
REGISTERS
5
DECODE UNIT
PREFETCH UNIT
DATA CACHE
BUS UNIT
2=X a,a
3=Y b,b
PRINT Z d,d
INSTRUCTION CACHE
OUTPUT
INPUT
2 3
+ =


ARITHMETIC LOGIC UNIT
CONTROL UNIT
1011010
REGISTERS
5
DECODE UNIT
PREFETCH UNIT
PRINT Z
PRINT Z
DATA CACHE
BUS UNIT
2=X a,a
3=Y b,b
PRINT Z d,d
INSTRUCTION CACHE
OUTPUT
INPUT
2 3
+ =


ARITHMETIC LOGIC UNIT
2+3=5
CONTROL UNIT
REGISTERS
5
DECODE UNIT
PREFETCH UNIT
DATA CACHE
BUS UNIT
2=X a,a
3=Y b,b
X+Y=Z c,c
PRINT Z d,d
INSTRUCTION CACHE
5
OUTPUT
INPUT
2 3
+ =

16



- Coordination of many *levels of abstraction*

Programming Levels of Representation

High Level Language Program (e.g., C)

Compiler

Assembly Language Program (e.g. ARM)

Assembler

Machine Language Program (ARM)

Machine Interpretation

Control Signal Specification

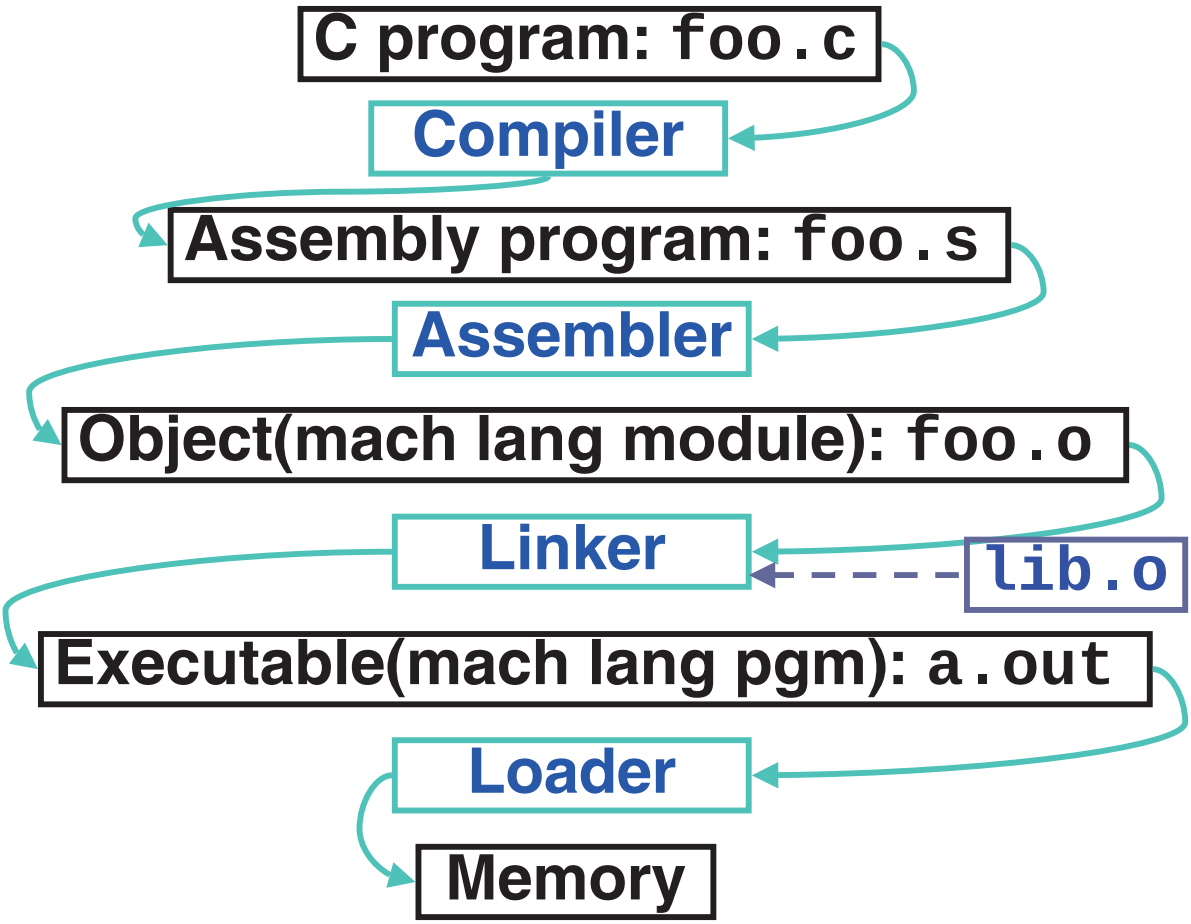
```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
ldr  r0 , [r2, #0]
ldr  r1 , [r2, #4]
str  r1 , [r2, #0]
str  r0 , [r2, #4]
```

```
1110 0101 1001 0010 0000 0000 0000 0000
1110 0101 1001 0010 0000 0000 0000 0100
1110 0101 1000 0010 0001 0000 0000 0000
1110 0101 1000 0010 0001 0000 0000 0100
```

ALUOP[0:3] <= InstReg[9:11] & MASK

Steps to Starting a Program



Beispiele für Prozessorbefehle:

- **LOADA mem** - Load register A from memory address
- **LOADB mem** - Load register B from memory address
- **CONB con** - Load a constant value into register B
- **SAVEB mem** - Save register B to memory address
- **SAVEC mem** - Save register C to memory address
- **ADD** - Add A and B and store the result in C
- **SUB** - Subtract A and B and store the result in C
- **MUL** - Multiply A and B and store the result in C
- **DIV** - Divide A and B and store the result in C
- **COM** - Compare A and B and store the result in test
- **JUMP addr** - Jump to an address
- **JEQ addr** - Jump, if equal, to address
- **JNEQ** addr - Jump, if not equal, to address
- **JG addr** - Jump, if greater than, to address
- **JGE addr** - Jump, if greater than or equal, to address
- **JL addr** - Jump, if less than, to address
- **JLE addr** - Jump, if less than or equal, to address
- **STOP** - Stop execution

Compiler → Übersetzen des Programmcodes in Maschinencode (hier ist der Prozessorbefehlsatz entscheident)

sehr gute Erklärung siehe http://computer.howstuffworks.com/

Instruction	Mnemonic	Operation	Flags	
			Z	C
Arithmetic				
Add W and File	addwf f,d	(d) <- W + (f)	√	√
Add Literal and W	addlw L	W <- #L + W	√	√
Bit Clear File bit n	bcf f,n	f_n <- 0	•	•
Bit Set File bit n	bsf f,n	f_n <- 1	•	•
Clear File	clrf f	(f) <- 00	√	•
Increment File	incf f,d	(d) <- (f) + 1	√	•
Decrement File	decf f,d	(d) <- (f) - 1	√	•
Subtract W from File	subwf f,d	(d) <- (f) - W	√	√
Movement				
Move File	movf f,d	(d) <- (f)	√	•
Move W to File	movwf f	W <- (f)	•	•
Move Literal to W	movlw L	W <- #L	•	•
Logic				
AND W and File	andwf f,d	(d) <- W · (f)	√	•
AND Literal and W	andlw L	W <- #L · W	√	•
Complement File	comf f	(f) <- $\overline{(f)}$	√	•
Inclusive OR W and File	iorwf f,d	(d) <- W + (f)	√	•
Inclusive OR Literal and W	iorlw L	W <- #L + W	√	•
Skip and Jump				
Bit Test File, Skip if Clear	btfsc f,b	b == 0 ? PC++ : PC	•	•
Bit Test File, Skip if Set	btfss f,b	b == 1 ? PC++ : PC	•	•
Call (jump to) subroutine	call aaa	(TOS) <- pc, pc <- aaa	•	•
Go to address	goto aaa	pc <- aaa	•	•
Return from subroutine	return	pc <- (TOS)	•	•

√	: Flag operates normally
aaa	: Address of instruction
C	: Carry or $\overline{\text{Borrow}}$, bit 0 in F3
d	: Destination; 0 = W, 1 = file
L	: 8-bit Literal
++	: Increment
A?S1:S2	: IF A is True THEN DO S1 ELSE DO S2
f	: File
pc	: Program Counter

•	: Flag not affected
b	: Bit b (0–7) in file
Z	: Zero, bit 2 in F3
W	: Working register
==	: Equivalent to
( )	: Contents of
TOS	: Top Of Stack
d	: File or W

Beispiele für Assembler-Programmierung

```
movf  NUM_1,w     ; Copy the variable NUM_1 to W
addlw 4           ; Add the literal constant 4 to it
movwf NUM_2       ; Copy NUM_1 + 4 into NUM_2
```

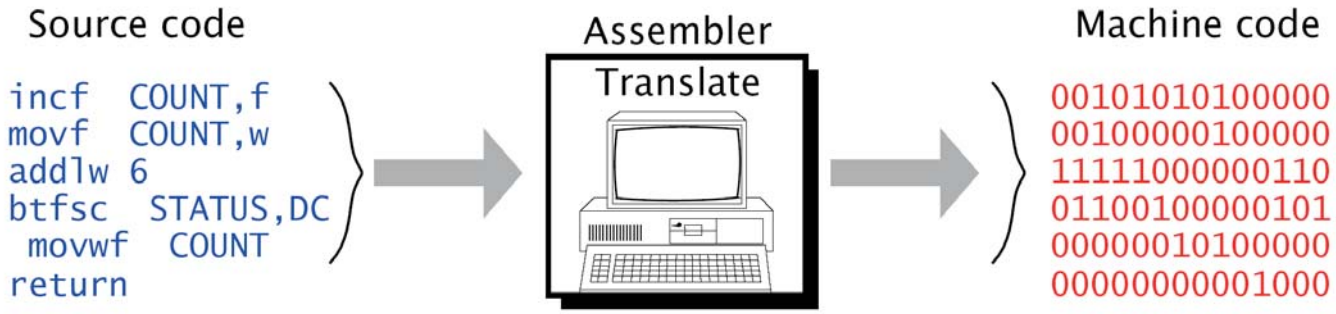
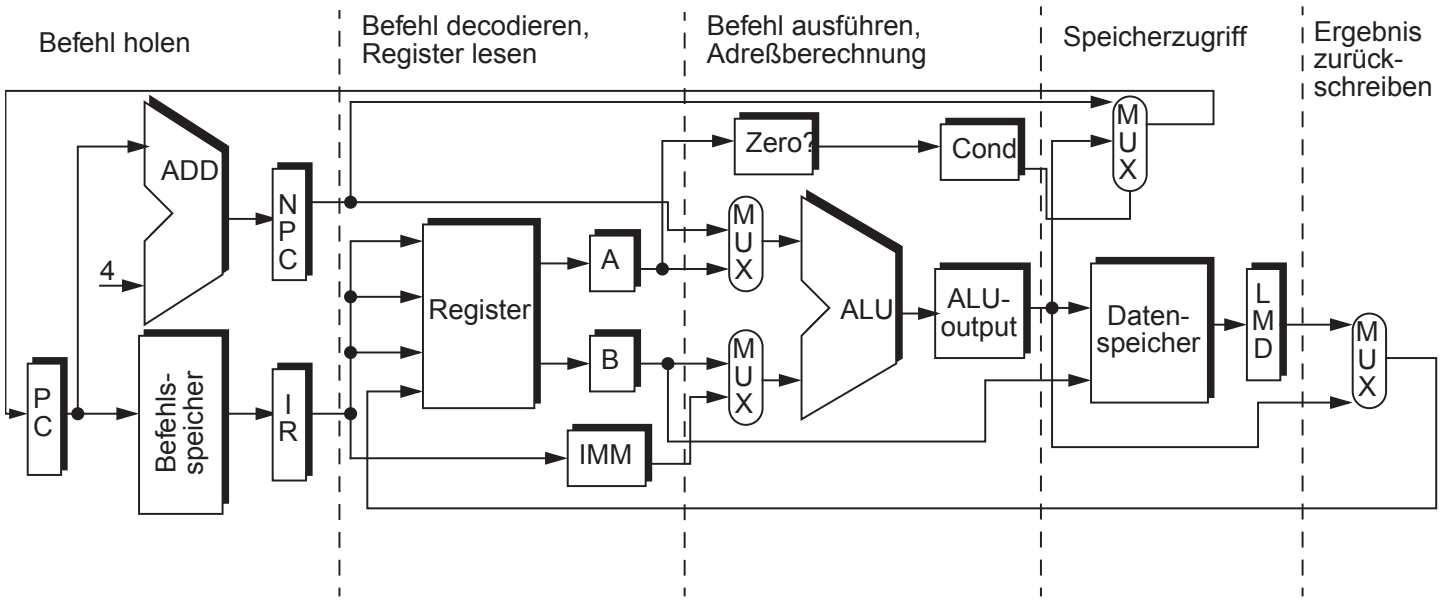


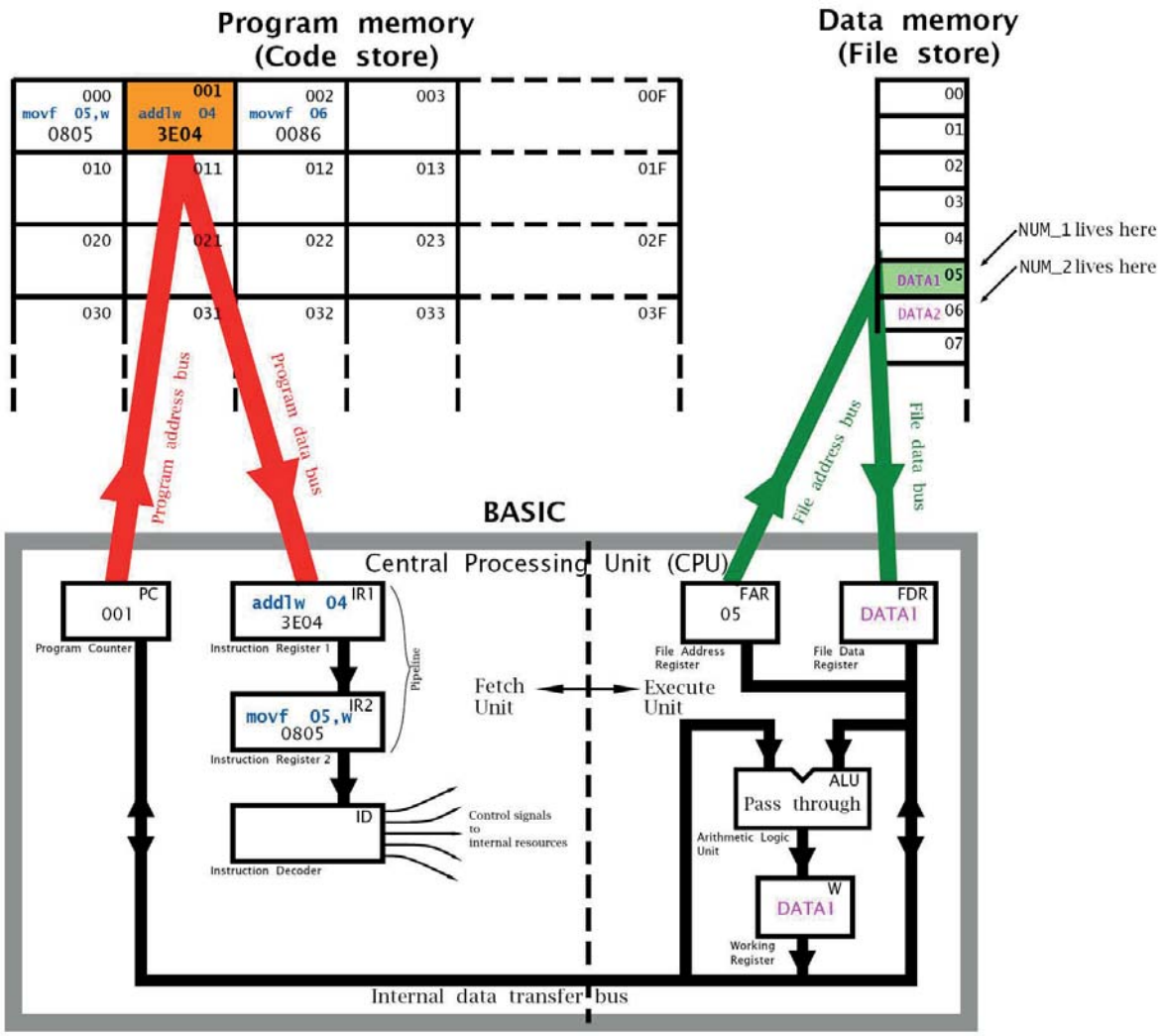
Figure 8: Conversion from assembly-level source code to machine code.

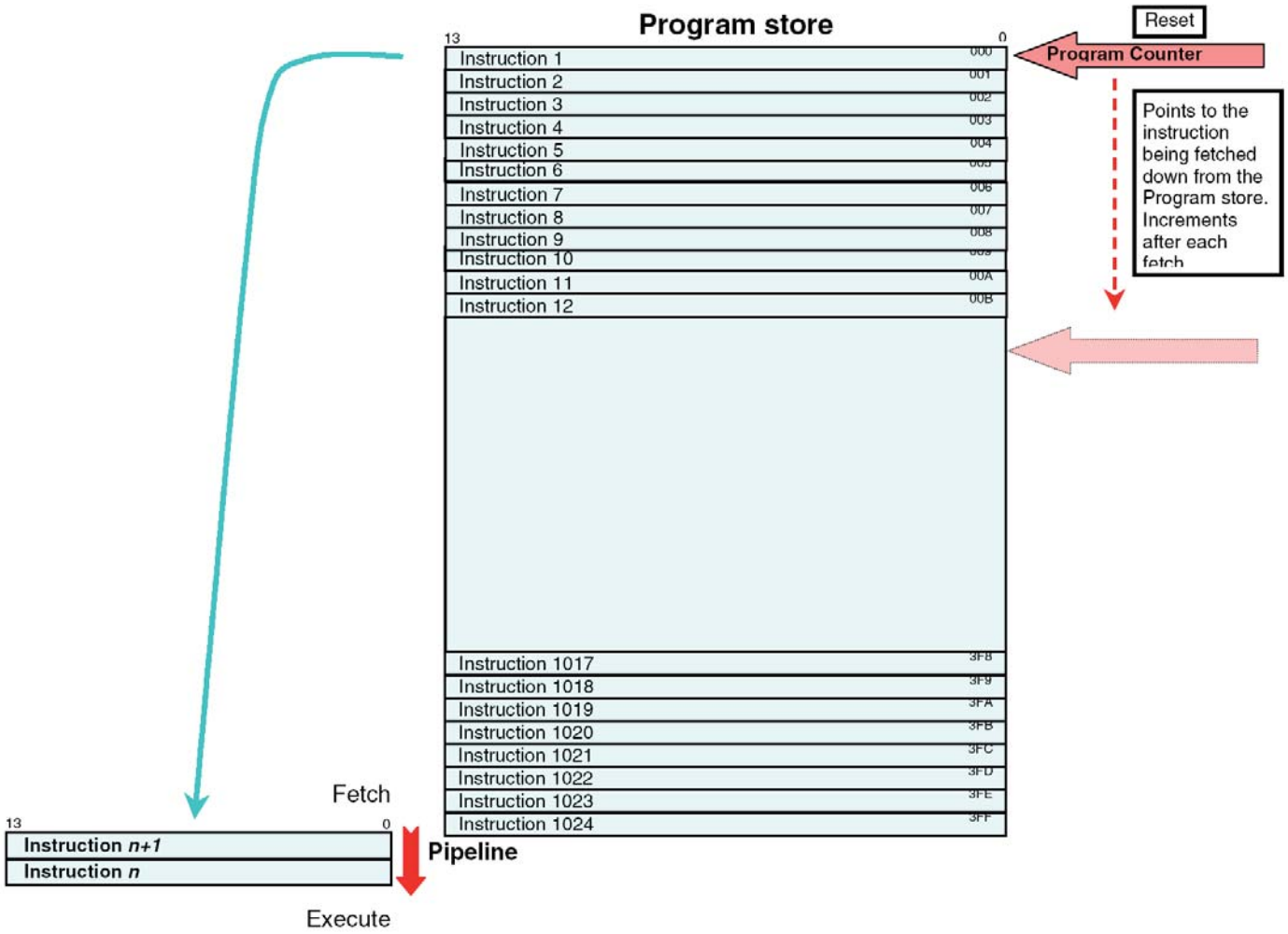
MS nybble / LS nybble	h'0' b'000'	h'1' b'001'	h'2' b'010'	h'3' b'011'	h'4' b'100'	h'5' b'101'	h'6' b'110'	h'7' b'111'
h'0' b'0000'	NUL	DLE	SP	0	@	P	‘	p
h'1' b'0001'	SOH	XON	!	1	A	Q	a	q
h'2' b'0010'	STX	DC2	"	2	B	R	b	r
h'3' b'0011'	ETX	XOFF	#	3	C	S	c	s
h'4' b'0100'	EOT	DC4	$	4	D	T	d	t
h'5' b'0101'	ENQ	NAK	%	5	E	U	e	u
h'6' b'0110'	ACK	SYN	&	6	F	V	f	v
h'7' b'0111'	BEL	ETB	’	7	G	W	g	w
h'8' b'1000'	BS	CAN	(	8	H	X	h	x
h'9' b'1001'	HT	EM	)	9	I	Y	i	y
h'A' b'1010'	LF	SUB	*	:	J	Z	j	z
h'B' b'1011'	VT	ESC	+	;	K	[	k	{
h'C' b'1100'	FF	FS	,	<	L	\	l	|
h'D' b'1101'	CR	GS	-	=	M	]	m	}
h'E' b'1110'	SO	RS	.	>	N	^	n	~
h'F' b1111'	SI	US	/	?	O	_	o	DEL

7-bit American Standard Code for Information Interchange (ASCI)

- **Phasen des Maschinenbefehlszyklus**




Program memory
(Code store)
000
movf 05,w
0805
001
addlw 04
3E04
002
movwf 06
0086
003
00F
010
011
012
013
01F
020
021
022
023
02F
030
031
032
033
03F
Program address bus
program data bus
Data memory
(File store)
00
01
02
03
04
NUM_1 lives here
NUM_2 lives here
DATA1 05
DATA2 06
07
File address bus
File data bus
BASIC
Central Processing Unit (CPU)
PC
001
Program Counter
addlw 04
3E04
IR1
Instruction Register 1
Pipeline
movf 05,w
0805
IR2
Instruction Register 2
ID
Instruction Decoder
Control signals
to
internal resources
Fetch
Unit
Execute
Unit
FAR
05
File Address
Register
FDR
DATA1
File Data
Register
ALU
Pass through
Arithmetic Logic
Unit
W
DATA1
Working
Register
Internal data transfer bus


Program store
13
0
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
Instruction 8
Instruction 9
Instruction 10
Instruction 11
Instruction 12
Instruction 1017
Instruction 1018
Instruction 1019
Instruction 1020
Instruction 1021
Instruction 1022
Instruction 1023
Instruction 1024
Reset
Program Counter
Points to the instruction being fetched down from the Program store. Increments after each fetch
Fetch
13
0
Instruction n+1
Instruction n
Pipeline
Execute

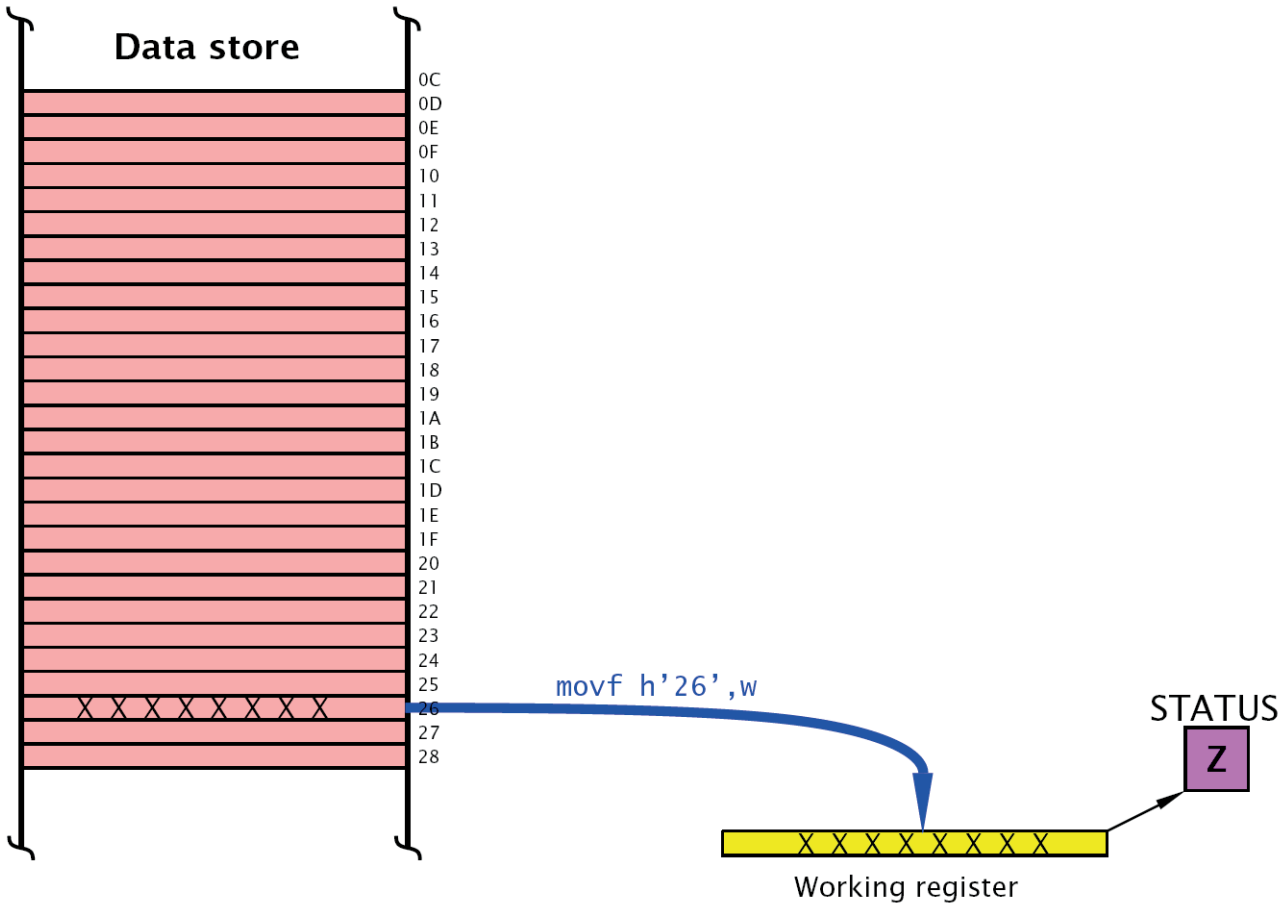


Figure 2: Showing the `movf h'26',w` instruction.

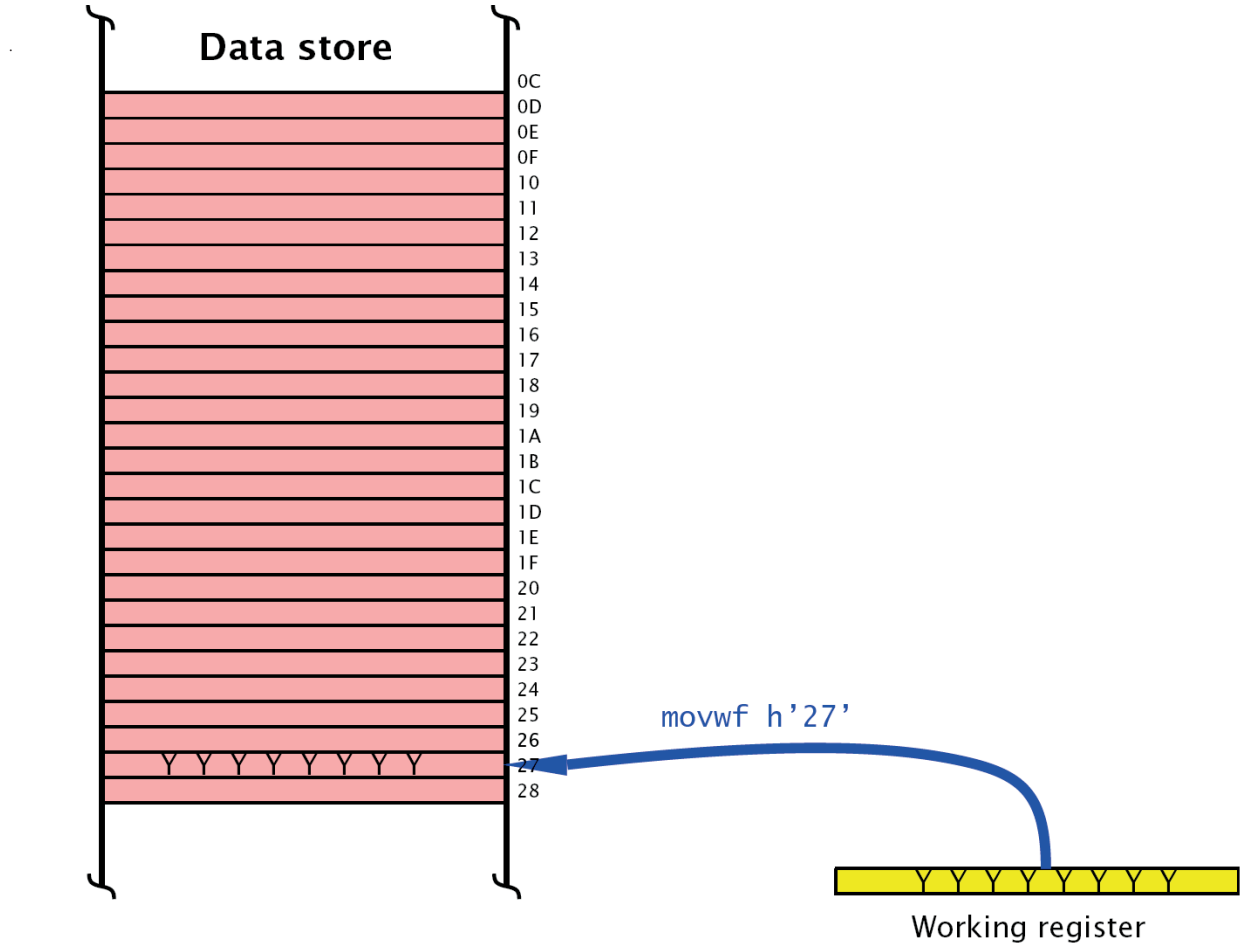
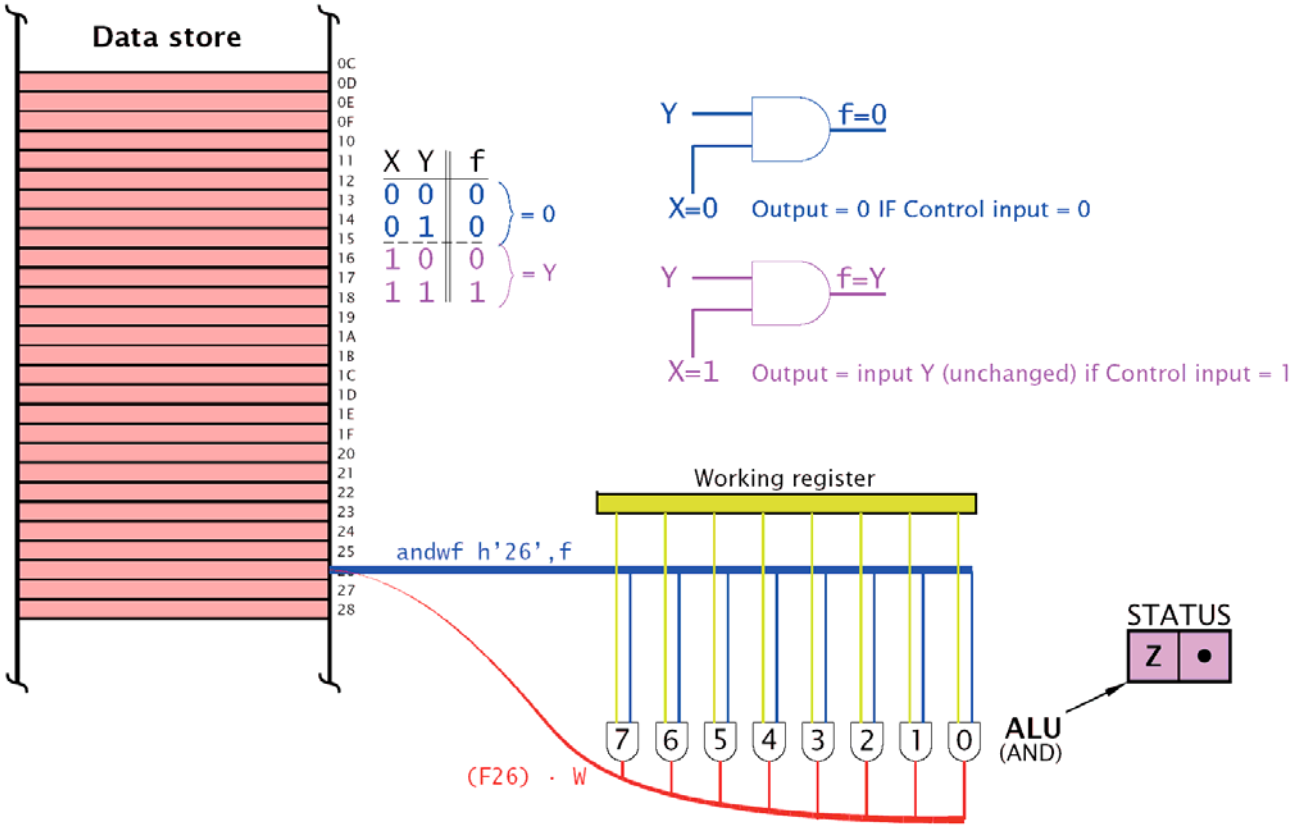
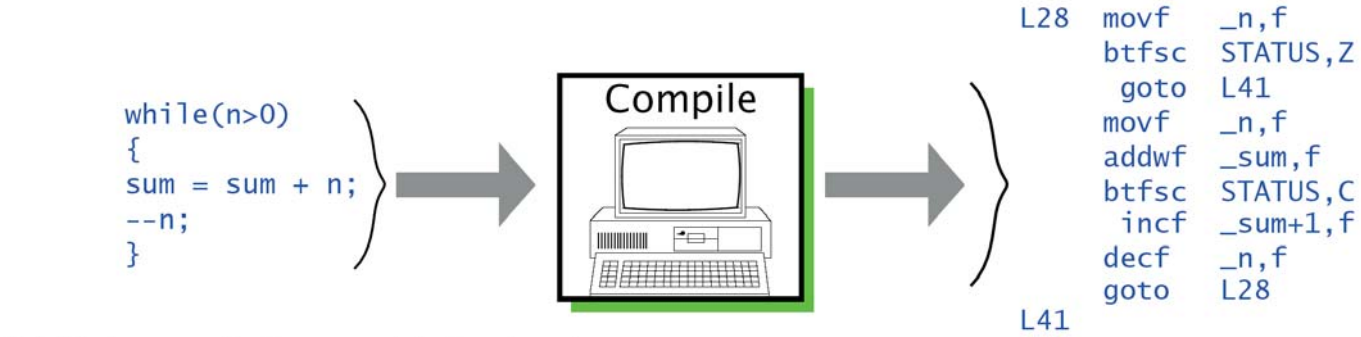


Figure 3: Showing the `movwf h'27'` instruction.

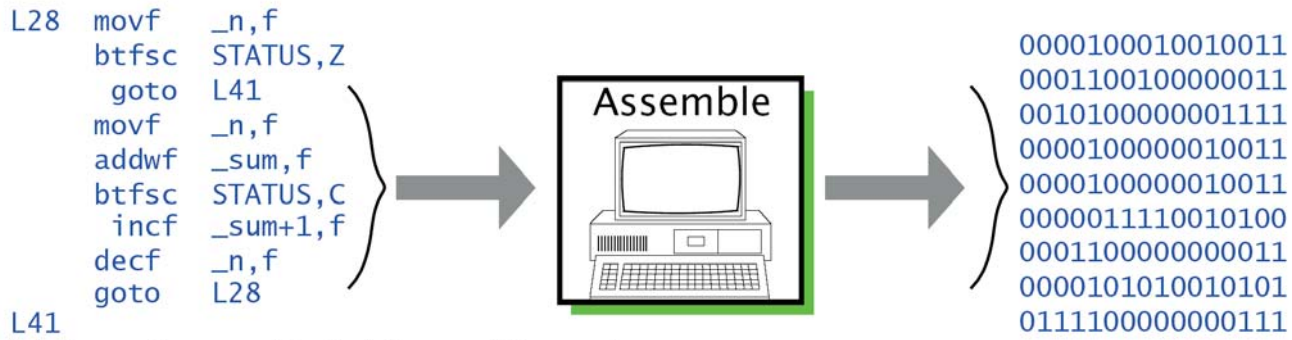


Bitwize ANDing the contents of a File with the contents of the Working register

```
FUEL      equ    h'30'     ; Read the fuel in litres @ File h'30'
DISPLAY   equ    h'25'     ; Display devices are connected to File h'25'
GREEN_LED equ    0         ; in which bit0 is connected to the Green LED
RED_LED   equ    1         ; and bit1 is connected to the Red LED
BUZZER    equ    2         ; and bit2 is connected to the Buzzer
STATUS    equ    3         ; The Status register is File 3
C         equ    0         ; and the Carry/NOT Borrow flag is bit 0

; --------------------------
MAIN      clrf   DISPLAY    ; Turn off all LEDs and Buzzer

; Check IF higher or equal to 11 litres ------------
          movf   FUEL,w     ; Copy Fuel reading into W
          addlw  -d'11'     ; Subtract 11; i.e. W = FUEL - 11
          btfsc  STATUS,C   ; IF a Borrow (C==0) THEN lower than 11 litres
           goto  GREEN      ; ELSE indicate Green

; Check for higher or equal to 6 litres ------------
          movf   FUEL,w     ; Copy Fuel reading into W
          addlw  -6         ; Subtract 6; i.e. W = FUEL - 6
          btfsc  STATUS,C   ; IF a Borrow (C==0) THEN lower than 6 litres
           goto  RED        ; ELSE indicate Red

; Must be the buzzer! ------------------
         bsf     DISPLAY,BUZZER    ; Sound the Buzzer (Bit 2 in File h'25')
         goto    NEXT

RED      bsf     DISPLAY,RED_LED   ; Light the Red LED (Bit 1)
         goto    NEXT

GREEN    bsf     DISPLAY,GREEN_LED ; Light the Green LED (Bit 0)

NEXT     ....    .....             ; Next part of the program
```



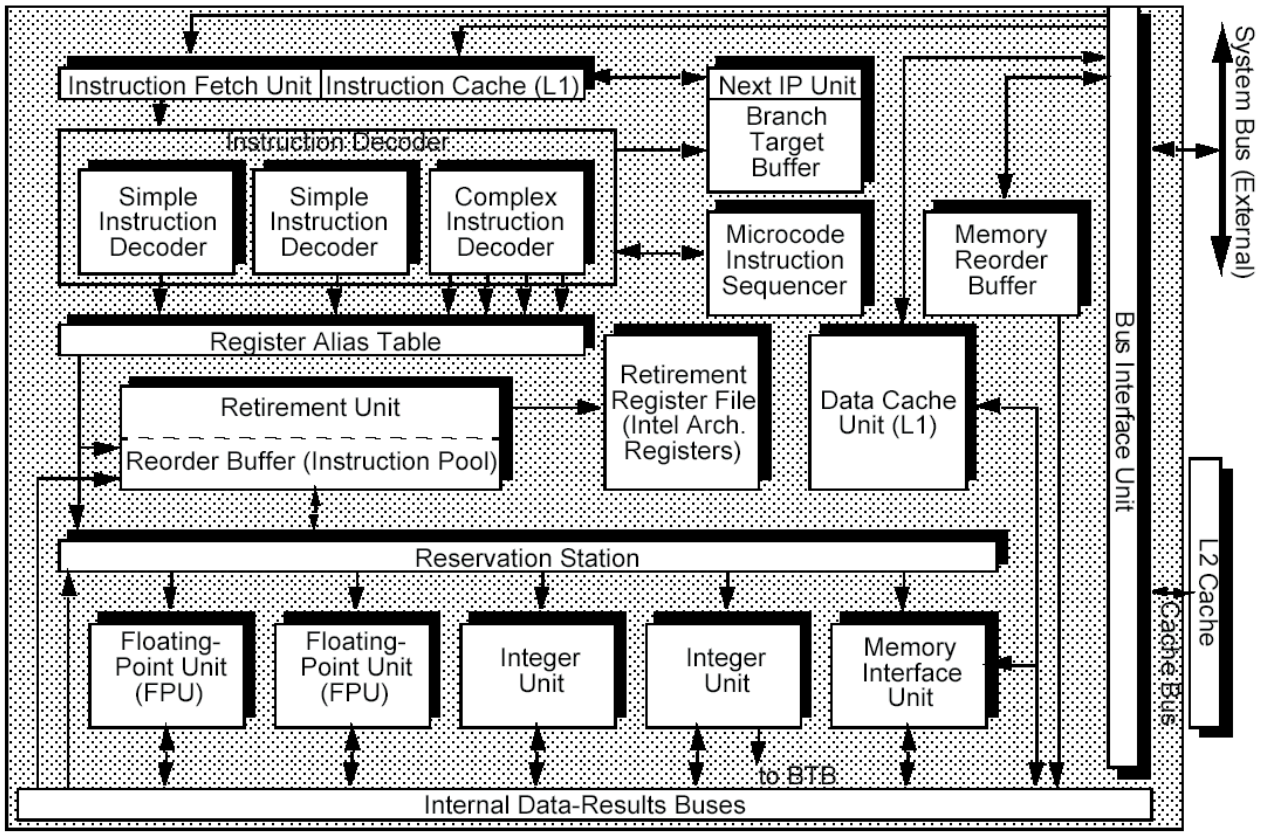
(a) First, compile to assembly-level code.



(b) Second, assemble-link to machine code.

Figure 2: Conversion from high-level **C** source code to machine code.

❑ Beispiel Intel Pentium II

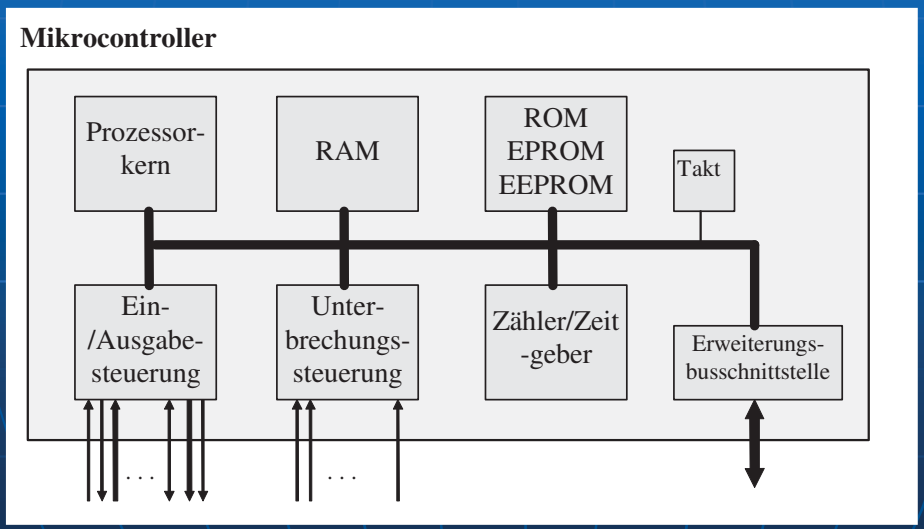


Mikrocontroller

- Unterschied zum Mikroprozessor
  - Verschiedene Funktionen auf dem selben Chip integriert
  - Timer, Zähler, Interfaces, A/D-Wandler, Watchdog…
  - Mikrorechner auf einem Chip
- Meist auf spezifische Anwendungen zugeschnitten
  - Haushalt (Telefon, Fernseher)
  - KFZ-Technik (Motormanagement)
  - Automatisierungstechnik (Produktionsanlagen)
  - Medizintechnik (Beatmungsgeräte)
  - …

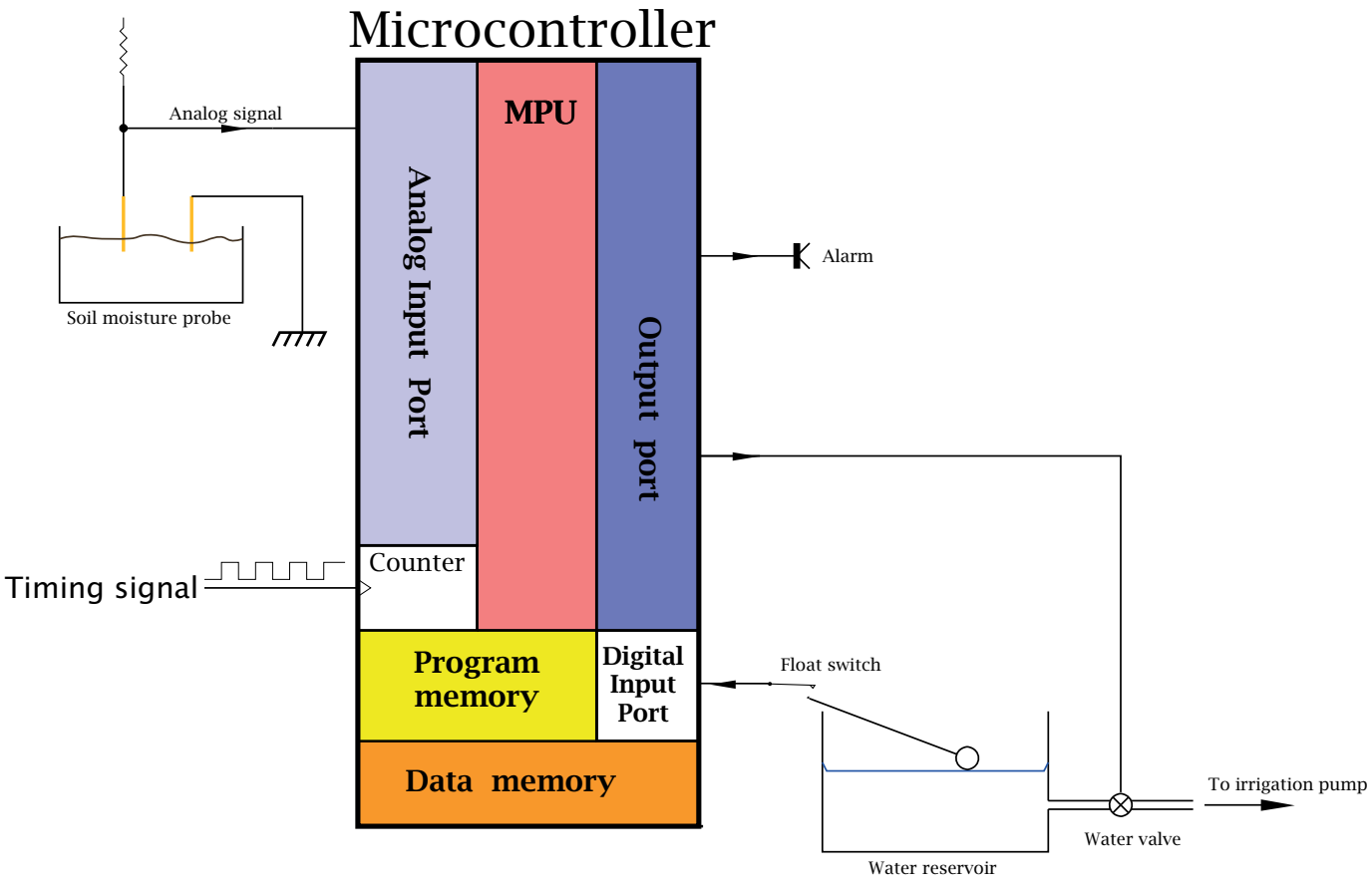
Microcontroller

- Ein-Chip-Mikrorechner mit speziell zugeschnittener Peripherie



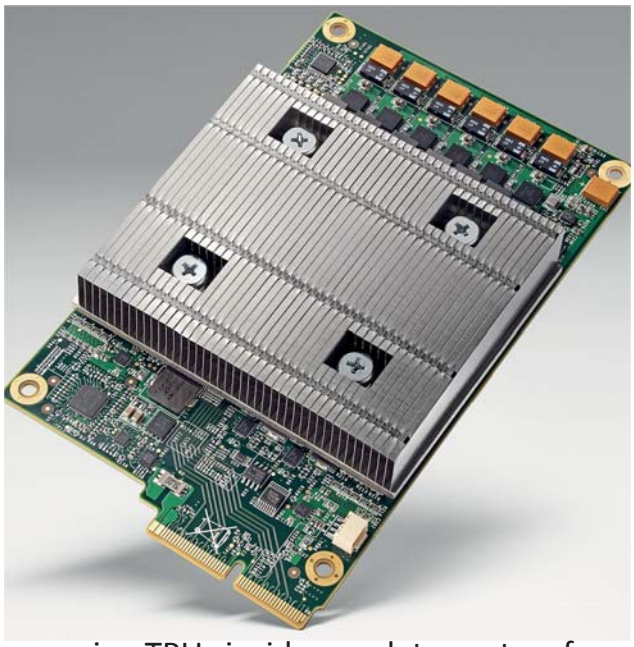
Anwendungsbeispiele

- Mikrocontroller für Motor- und Getriebesteuerungen
  - Neue Chips von Infineon mit bis zu 400 MHZ und bis zu 2MB Speicher
  - Senken von Kraftstoffverbrauch und Schadstoffausstoß
  - Verzehnfachung der Leistung in 4 Jahren. Ziel: 1 Liter Auto



Google supercharges machine learning tasks with TPU custom chip

Wednesday, May 18, 2016



The result is called a Tensor Processing Unit (TPU), a custom ASIC we built specifically for machine learning — and tailored for TensorFlow. We've been running TPUs inside our data centers for more than a year, and have found them to deliver an order of magnitude better-optimized performance per watt for machine learning.

The result is called a Tensor Processing Unit (TPU), a custom ASIC we built specifically for machine learning — and tailored for TensorFlow. We've been running TPUs inside our data centers for more than a year, and have found them to deliver an order of magnitude better-optimized performance per watt for machine learning.

TPU is tailored to machine learning applications, allowing the chip to be more tolerant of reduced computational precision, which means it requires fewer transistors per operation. Because of this, we can squeeze more operations per second into the silicon, use more sophisticated and powerful machine learning models and apply these models more quickly, so users get more intelligent results more rapidly. A board with a TPU fits into a hard disk drive slot in our data center racks.

Googles neue Deep-Learning-Hardware braucht Flüssigkühlung

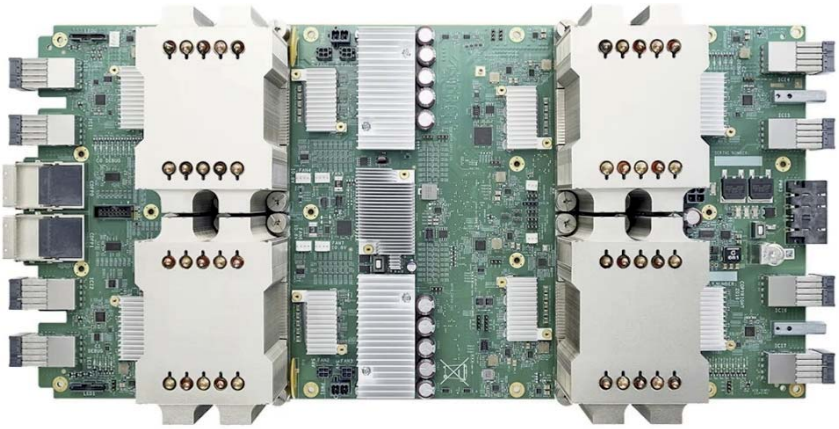
Google I/O 2018

Für die dritte Version von Googles TPU nutzt das Unternehmen erstmals eine Flüssigkeitskühlung. Ein TPU-System für das maschinelle Lernen mit der neuen Generation soll über 100 Petaflops an Rechenleistung erreichen.

So bietet eine einzelne Einheit der zweiten Generation der TPU eine Leistung von 180 Teraflops. Diese Module wiederum werden zu einem Pod aus 64 einzelnen Geräten zusammengefasst. Die etwas mehr als achtfache Leistung eines dieser Pods entspricht den in der Ankündigung genannten 100 Petaflops.

LRZ: **SuperMUC-NG**

Start of of Phase 1 is planned for January 2019.



Number of Nodes	**6,480**
Number of Core Cores	311,040
PEAK @ nominal (PFlop/s)	26.9

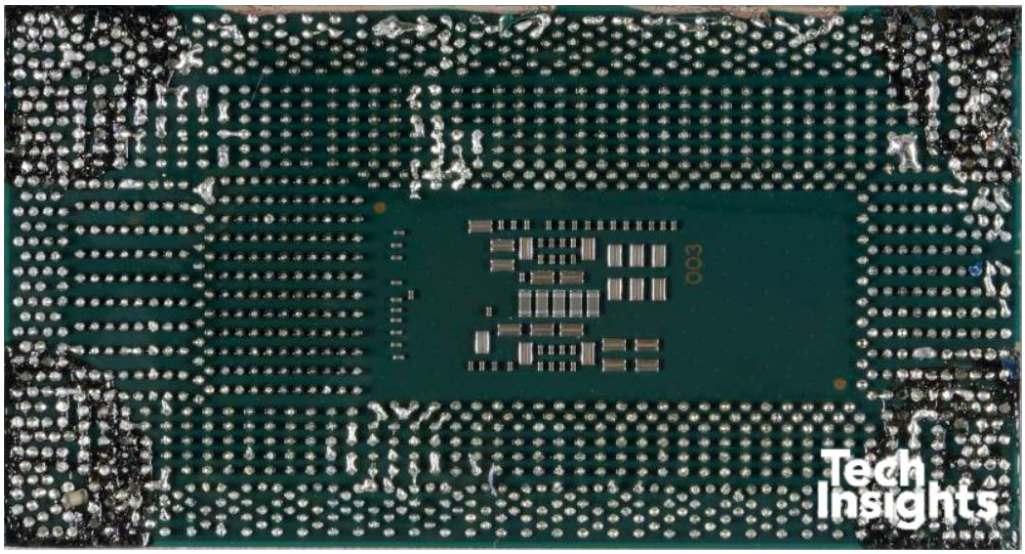
Cannon Lake U/Y

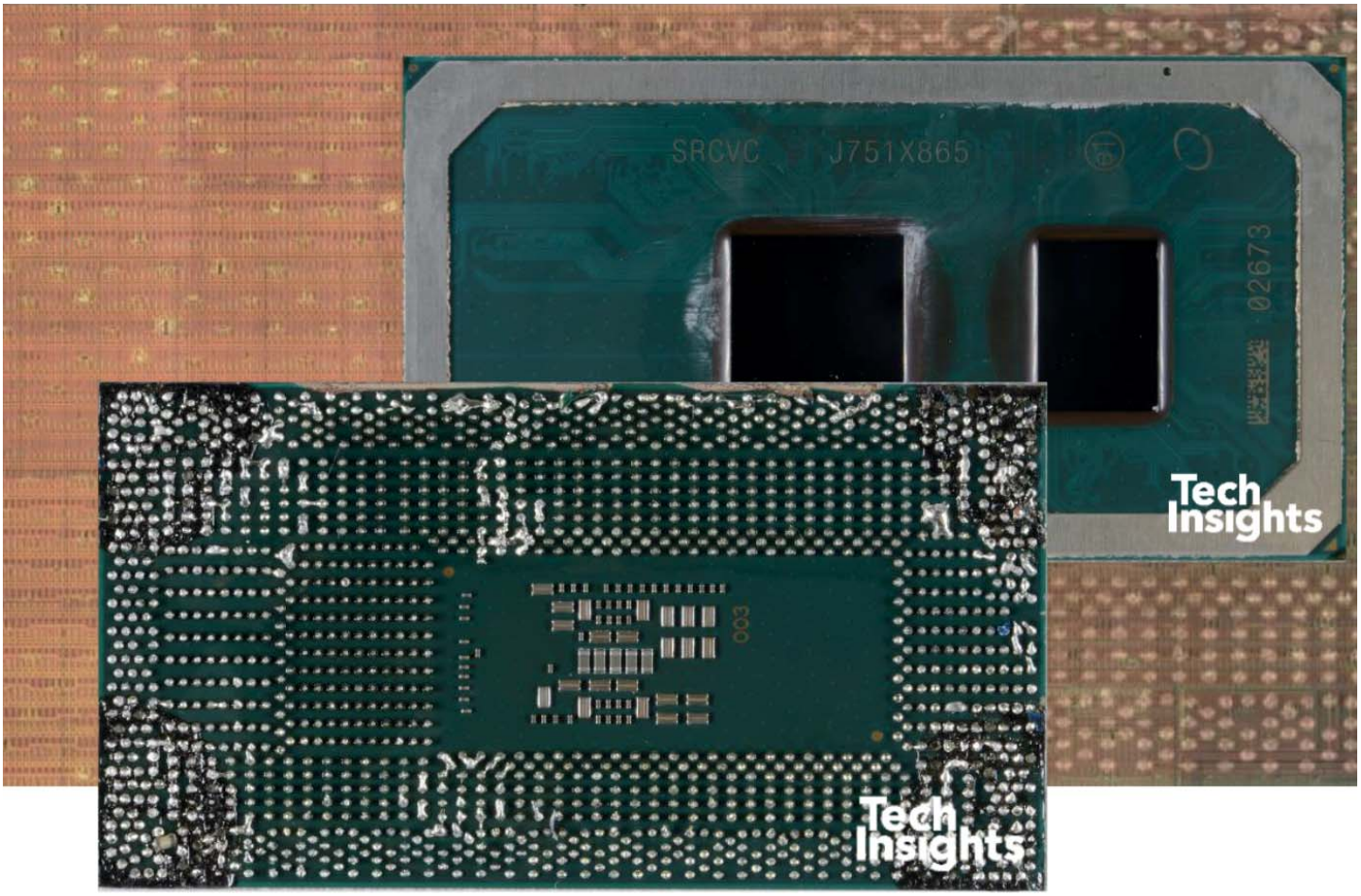
Intels erster 10-nm-Chip misst 70 mm^2

Mittlerweile ist der Core i3-8121U, ein Cannon Lake U, in Form eines Lenovo-Notebooks erhältlich. Wie gut Intels 10-nm-Verfahren rein von der Fläche her skaliert, zeigt der Core i3-8121U aus dem Ideapad 330-15ICN von Lenovo.

Basierend auf der Package-Abmessung von 45 x 24 mm lässt sich die Größe des Dies auf 70 mm^2 berechnen, der Chipsatz bringt es auf 48 mm^2.

- 10-nm-FinFet-Verfahren
- zwei CPU-Kerne mit Skylake-Architektur + GT2-Grafikeinheit mit Gen10-Architektur
- AVX-512-Block, der viel Platz benötigt + GPU mit 40 Execution Units





Intels Problem ist nicht die mangelnde Skalierung des 10-nm-Verfahrens, sondern die **derzeit miserable Fertigungsqualität**. Besagter Core i3-8121U wird mit deaktivierter Grafikeinheit ausgeliefert, da die Chip-Ausbeute (yield) zu viele Defekte hervorruft.

Intel reizt für **10 nm die Grenzen der Immersionslithografie** aus, beispielsweise beträgt der Interconnect Pitch gerade einmal 36 nm, es wird Kobalt für einige Layer eingesetzt und kein anderer Hersteller nutzt Contact Over Active Gate für die Transistoren.

Andere Fertiger wie Samsung erreichen eine ähnlich kompakte Technologie bei ihrem **7-nm-Node**, der aber **extrem ultraviolette Strahlung** inkludiert, auf die Intel verzichtet.

Die **TSMC produziert bereits 7-nm-Chips ohne EUV**, der Interconnect Pitch ist mit 40 nm etwas entspannter.

Weil 10 nm derartige Probleme bereitet, schiebt Intel weitere Generationen mit 14++ nm nach: Die heißen Whiskey Lake U und Comet Lake U.

https://www.golem.de/news/cannon-lake-u-y-intels-erster-10-nm-chip-misst-70-mm-1806-134990.amp.html

Putting it all in perspective…

"If the automobile had followed the same development cycle as the computer, a Rolls-Royce would today cost $100, get a million miles per gallon, and explode once a year, killing everyone inside."

– Robert X. Cringely

Technical Writer, Broadcaster and Computer Guy

http://www.pbs.org/cringely/about/

Triumph of the Nerds

A history of the PC industry,
An ABC program a few years ago