

CISC-RISC-EPIC

eine Zwangs-Evolution?!

Heutiges Programm:

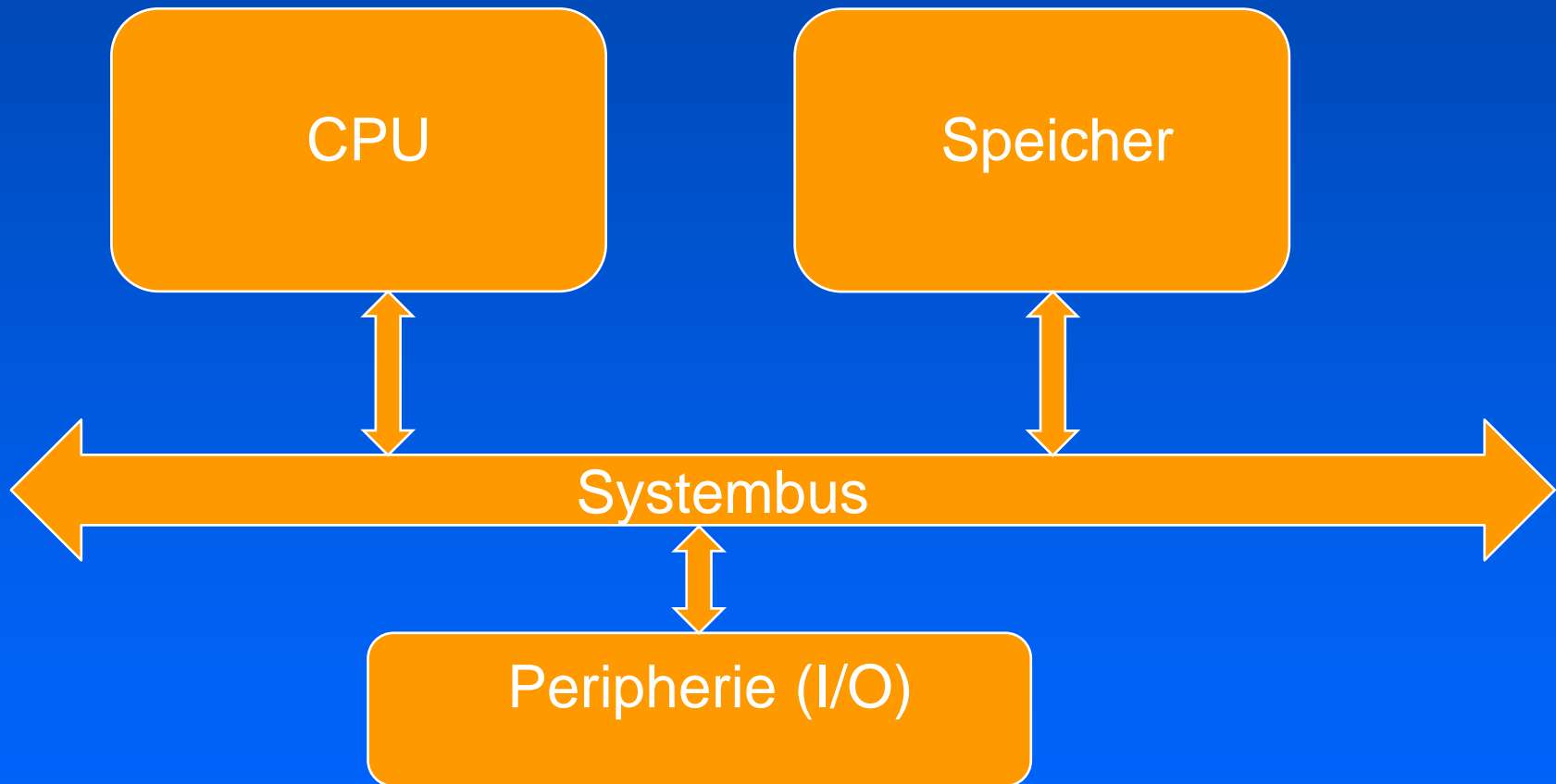
- CISC-RISC-EPIC - Begriffserklärung
- von Neumann Rechnerarchitektur
- Evolution der Architektur
- Beispiel: CS.c
- ...

CISC-RISC-EPIC

- CISC – complex instruction set computers
- RISC – reduced instruction set computers
- EPIC – explicitly parallel instruction computing
- VLIW – very large instruction word computers

- Entwicklung CISC→RISC→EPIC
- Begriffe für verschiedene Evolutionsstufen der Rechnerarchitektur
- CISC – erst als Abgrenzung zu RISC erfunden

Blockdiagramm Rechner



Blockdiagramm Rechner

- dieses Diagramm ist allgemeingültig, d.h. es trifft im wesentlichen auf „alle“ heute verwendeten Rechner(-Architekturen) zu
- es ist zeitlos, d.h. es ist schon eine Weile so und wird sich so schnell nicht ändern (?)



WARUM?

unsere System basieren alle auf dem **von Neumann Prinzip**

von Neumann Architektur

- John von Neumann, 1945 in den USA (Princeton) veröffentlicht
- Prinzipien bereits vorher von Zuse 1936 ausgearbeitet
- Referenzmodell einer universellen Architektur
- „Rechner“ gab es schon: feste Verdrahtung der Programme, basierend auf Turingmaschinen → Abarbeitung eines festen Programms
- jetzt der Ansatz, den Rechner ohne feste Verdrahtung universell zu gestalten und beliebige Programme abzuarbeiten

von Neumann Architektur

- 4 Komponenten (Funktionseinheiten):
 1. Rechenwerk (ALU – arithmetic logic unit, CPU)
 2. Steuerwerk (CU - control unit)
 3. Speicher (Memory)
 4. Eingabewerk/Ausgabewerk (I/O)
- Komponenten über Bussystem verbunden
- ALU & CU → in CPU (Prozessor) zusammenfasst
- manchmal: 5 Komponenten (I+O)

von Neumann Architektur

- warum ist sie so erfolgreich? (andere Architekturen sind zwar Gegenstand der Forschung, spielen aber noch keine wesentliche Rolle)
- weil sich das Modell an biologischen Vorbildern – der Informationsverarbeitung des Menschen – orientiert
- die meisten Methoden, Werkzeuge, Programmiersprachen und Denkgewohnheiten sind an der von-Neumann-Architektur ausgerichtet

von Neumann Architektur

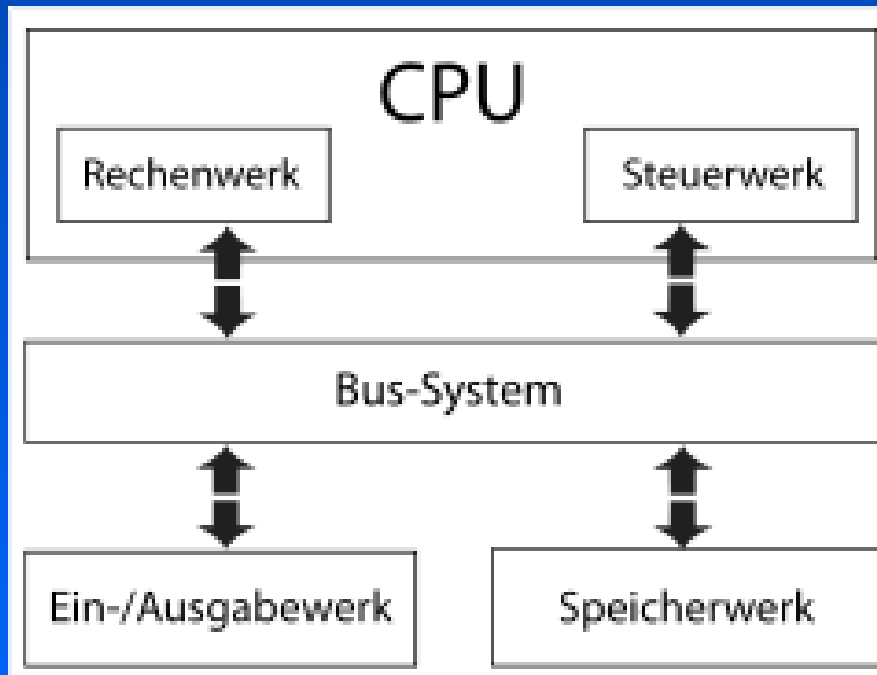
- Prinzipien:

1. Der Rechner besteht aus 4 Funktionseinheiten: dem Steuerwerk, dem Rechenwerk, dem Speicher, dem Eingabe-/Ausgabewerk
2. Die Struktur des von-Neumann-Rechners ist unabhängig von den zu bearbeitenden Problemen. Zur Lösung eines Problems muss von außen das Programm eingegeben und im Speicher abgelegt werden. Ohne dieses Programm ist die Maschine nicht arbeitsfähig.
3. Programme, Daten, Zwischen- und Endergebnisse werden in demselben Speicher abgelegt.
4. Der Speicher ist in gleich große Zellen unterteilt, die fortlaufend durchnummeriert sind. Über die Nummer (Adresse) einer Speicherzelle kann deren Inhalt abgerufen oder verändert werden.
5. Aufeinanderfolgende Befehle eines Programms werden in aufeinanderfolgenden Speicherzellen abgelegt. Das Ansprechen des nächsten Befehls geschieht vom Steuerwerk aus durch Erhöhen der Befehlsadresse um Eins.
6. Durch Sprungbefehle kann von der Bearbeitung der Befehle in der gespeicherten Reihenfolge abgewichen werden.
7. Es gibt verschiedene Befehle: arithmetisch, logisch, Sprungbefehle, ...

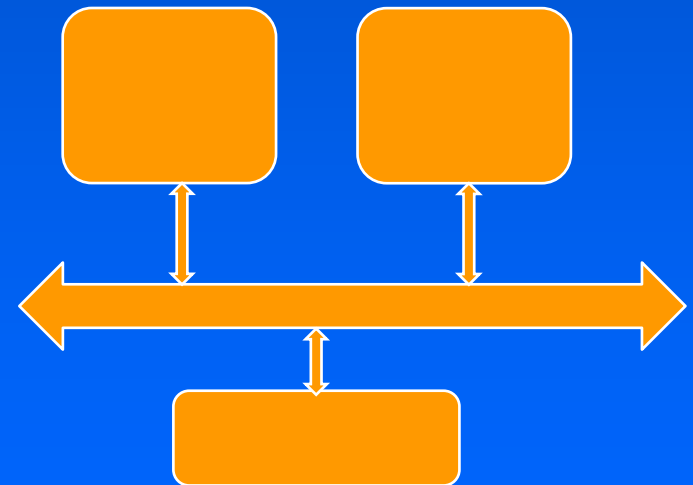
von Neumann Architektur

- Abarbeitung:
 - Laden des nächsten Befehls unter Adresse PC
 - Decodieren des geladenen Befehls
 - Interpretation des Befehls, ggf. „Nachladen“ der Operanden
 - Ausführen des Befehls
 - Ansteuern der CU, z.B. zum Speichern von Ergebnissen
 - PC auf nächsten Befehl setzen, wenn kein Sprungbefehl

Diagramm gefällig?



Genau! Das kennen wir schon ...



CISC-RISC-EPIC

- alle von Neumann Architektur
- wo sind die Unterschiede?
 - bezogen auf die Architektur: keine
- warum Evolution?
 - bezogen auf die Umsetzung zur immer größeren Steigerung der Effizienz und Leistung haben wir eine massive Entwicklung
 - trotz gleichbleibender (Basis-)Architektur
 - Architektur: jetzt als Umsetzung des von Neumann Prinzips

Code Beispiel

- Berechnung einer Prüfsumme nach „Modulo 0 mit Wichtung 3“
- Beispiel:

a)	4	0	1	2	3	4	5	1	2	3	4	5	-
b) x	1	3	1	3	1	3	1	3	1	3	1	3	-
c) =	4	0	1	6	3	12	5	3	2	9	4	15	-
d)	Produktsumme 64												
e)	aufrunden auf 70												
f)	$64 + 6 = 70$, 6 = Prüfziffer												

Code Beispiel

```
void checkSum3 (char *shipmentNR)
{
    char * p = shipmentNR ;
    int  chkSum = 0 ;
    int  chkDgt = 0 ;
    int  idx   = 0 ;
    int  hasCheckDigit = 0 ;

    if (*p == '*')
    {
        p++ ;
        hasCheckDigit = 1 ;
    }

    while (*p)
    {
        if (hasCheckDigit && *(p+1) == '\0')
            chkSum = chkSum + (*p++ - '0') ;
        else
            chkSum = chkSum + ((*p++ - '0') * ((idx++ & 1) ? 1 : 3)) ;
    }

    chkDgt = 10 - (chkSum % 10) ;
    if (chkDgt == 10)
        chkDgt = 0 ;

    printf ("* check sum (3): %s - %d / %d -> check digit: %d\n", shipmentNR, chkSum, chkSum % 10, chkDgt) ;
}
```

CISC - VAX

- CPU „langsamer“ als Bussystem (und Speicher)
- Kosten für CPU und Speicher deutlich höher als für Bussystem
- Bussystem kein Flaschenhals
- Optimierung auf Nutzung des Speichers (Minimierung Code-Größe - Bytepackung)
- Optimierung Befehle – möglichst viel mit dem geladenem Befehl „erledigen“

CISC - VAX

- Ist der Befehlssatz wirklich so komplex?
- Wie viele ADD-Befehle haben wir?

VAX Instruction Set		
ADD		
Add		
Format		
2operand:	opcode	add.rx, sum.mx
3operand:	opcode	add1.rx, add2.rx, sum.wx
Condition Codes		
N	<←	sum LSS 0;
Z	<←	sum EQL 0;
V	<←	{integer overflow};
C	<←	{carry from most-significant bit};
Exceptions		
		integer overflow
Opcodes		
80	ADDB2	Add Byte 2 Operand
81	ADDB3	Add Byte 3 Operand
A0	ADDW2	Add Word 2 Operand
A1	ADDW3	Add Word 3 Operand
C0	ADDL2	Add Long 2 Operand
C1	ADDL3	Add Long 3 Operand

- 1? → ADD-MACRO-Mnemonik
- 2? → 2 & 3 OP-Variante
- 6? → OP-Codes?

NEIN → UNZÄHLIGE!!!!

Der Befehlssatz ist das Produkt aus OP-Codes und Adressierungsarten!

CISC - VAX

8.8 Summary of General Mode Addressing

Table 8–5 General Register Addressing

Hex	Dec	Name	Assembler	r m w a v	PC	SP	AP FP	Can Be Indexed?
0–3	0–3	Literal	S^#literal	y f f f f	—	—	—	f
4	4	Indexed	i[Rx]	y y y y y	f	y	y	f
5	5	Register	Rn	y y y f y	u	uq	uo	f
6	6	Register deferred	(Rn)	y y y y y	u	y	y	y
7	7	Autodecrement	-(Rn)	y y y y y	u	y	y	ux
8	8	Autoincrement	(Rn)+	y y y y y	p	y	y	ux
9	9	Autoincrement deferred	@(Rn)+	y y y y y	p	y	y	ux
A	10	Byte displacement	B^D(Rn)	y y y y y	p	y	y	y
B	11	Byte displacement deferred	@B^D(Rn)	y y y y y	p	y	y	y
C	12	Word displacement	W^D(Rn)	y y y y y	p	y	y	y
D	13	Word displacement deferred	@W^D(Rn)	y y y y y	p	y	y	y
E	14	Longword displacement	L^D(Rn)	y y y y y	p	y	y	y
F	15	Longword displacement deferred	@L^D(Rn)	y y y y y	p	y	y	y

CISC - VAX

- jeder ADD-OP-Code kann je Operand mit jedem Adressmodus kombiniert werden und ergibt jeweils andere Codes (Codierung) im Speicher
- dabei immer unterschiedliche Länge der Befehle → je nach Operanden und Adressierung der Operanden
- klar, z.B. bei Literalen: Byte, Word, Longword (direkt im Code/Speicher)
- von Neumann (lässt Grüßen): holt Befehl samt Operanden aus dem Speicher, wo der PC hinzeigt
- Beispiel:
 - ADD B^#17, R0
 - ADD L^#17, R0
 - ADD #12345677, R0

Beispielcode VAX

```
425      2      chkSum = chkSum + ((*p++ - '0') * ((idx++ & 1) ? 1 : 3)) ;
          52 F8 AD D0      0120      movl   -8(fp),r2
          50 01 F8 AD C1      0124      addl3  -8(fp),#1,r0
          F8 AD 50 D0      0129      movl   r0,-8(fp)
          51 62 98      012D      cvtbl  (r2),r1
          51 30 C2      0130      subl2  #48,r1
          52 EC AD D0      0133      movl   -20(fp),r2
          50 01 EC AD C1      0137      addl3  -20(fp),#1,r0
          EC AD 50 D0      013C      movl   r0,-20(fp)
          52 FFFFFFFE 8F CA      0140      bicl2  #-2,r2
          52 D5      0147      tstl   r2
          09 13      0149      beql   sym.11
          E7 AD 01 90      014B      movb   #1,-25(fp)
          07 11      014F      brb   sym.12
          50 D5      0151      tstl   r0
          01      0153      nop
          0154      sym.11:
          E7 AD 03 90      0154      movb   #3,-25(fp)
          0158      sym.12:
          50 E7 AD 98      0158      cvtbl  -25(fp),r0
          51 50 C4      015C      mull2  r0,r1
          51 F4 AD C0      015F      addl2  -12(fp),r1
          F4 AD 51 D0      0163      movl   r1,-12(fp)
          0167      sym.13:
```

CISC - VAX

- große Anzahl Befehle (Kombinationen aus OP-Codes und Adressierungsarten)
- ermöglichen geringe Codegröße und „effiziente“ Ausnutzung der CPU
- Problem: Laden Opcode, Interpretation, dann erst Interpretation und Laden Operanden möglich und erst dann weiß man, wo der nächste Befehl im Speicher steht

Evolution

- CPU und Speicher wurden immer schneller
- Bussystem-Geschwindigkeit entwickelte sich nicht proportional in dem Maße mit
- u.a. wegen technologischer „Probleme“ → Bus besteht aus physischen Leitungen (Kabeln), getaktetes Protokoll, Interferenzen bei großen Taktraten
- Ansätze: Caches, ...“rein physische“ Bereitstellung der Daten
- aber: **Entkopplung Befehlsverarbeitung von Speicher/Bus um CPU maximal auszunutzen**

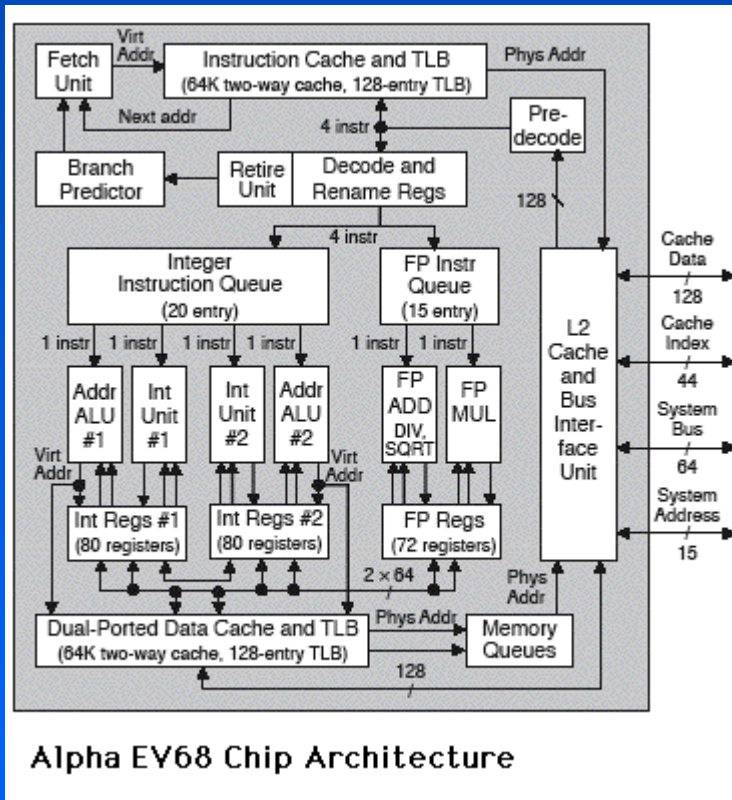
RISC – AXP/Alpha

- RISC hat nicht weniger „Befehle“ → ADDL, ADDQ, S4ADDL, S8ADDL, S4ADDQ, S8ADDQ
- aber: weniger Adressierungsmodus!!!
- warum: Entkopplung Laden der Befehle und Operanden von der Abarbeitung
- das Laden passiert unabhängig von der Ausführung → Pipeline in der CPU
- parallele und unabhängige Einheiten: LOAD, EXECUTE, STORE

RISC – AXP/Alpha

- Architektur:
 - Design für Geschwindigkeit
 - 64-Bit „LOAD and STORE“ RISC
 - 32 64-Bit Integer-Register (R31 = 0)
 - 32 64-Bit FP-Register (F31 = 0)
 - alle Befehle haben feste Länge von 32 Bits!!!
 - alle Operationen in CPU
 - alle Operanden werden zuvor geladen
 - 4 OP-Typen: Memory, Operate, Branch, PAL
 - alle „Operate“: 3-OP, Register-to-Register

RISC – AXP/Alpha



- Parallelisierung auf 2 Ebenen:
 - Entkopplung Laden und Verarbeiten → Instruction Queue
 - mehrere ALU's (I/FP) → parallele Abarbeitung von Befehlen

VAX - AXP

- VAX-Code:
 - ADD #17, R0, 10
- ein Befehl
- Adressen/Werte im Speicher
- Laden/Decodieren nicht unterbrechbar
- nächster PC erst nach Abarbeitung bekannt
- AXP-Code:
 - MOV #17, R30
 - ADDL R0, R30, R30
 - STL R30, 10
- mehr Befehle
- Berechnung nur in Registern
- können mit anderen Befehlen „gemixt“ werden
- alles LW-Adressen

Beispielcode AXP

```
-----30644-----chkSum = chkSum + (((*p++ - '0') * ((idx++ & 1) ? 1 : 3)) ;
4A7400D3      02A8      EXTBL R19, R20, R19                                ; 030639
2021FFD0      02AC      LDA R1, -48(R1)                                    ; 030642
42410012      02B0      ADDL chkSum, R1, chkSum                            ; R18, R1, R18          ; 030644
F67FFFF2      02B4      BNE R19, Ls23                                     ; 030639
C3E00046      02B8      BR __Implicit43
2FFE0000      02BC      UNOP
                02C0      Ls24:
2E740000      02C0      LDQ_U R19, (R20)                                  ; 030644
47E03401      02C4      MOV_1, R1
22140001      02C8      LDA R16, 1(R20)
22940001      02CC      LDA p, 1(R20)                                     ; R20, 1(R20)
440072C1      02D0      CMOVLBC idx, 3, R1                               ; R0, 3, R1
40003000      02D4      ADDL idx, 1, idx                                 ; R0, 1, R0
4A700F53      02D8      EXTQH R19, R16, R19
4A671790      02DC      SRA R19, 56, R16
2E740000      02E0      LDQ_U R19, (R20)                                  ; 030639
2210FFD0      02E4      LDA R16, -48(R16)                                ; 030644
4E010001      02E8      MULL R16, R1, R1
4A7400D3      02EC      EXTBL R19, R20, R19                              ; 030639
42410012      02F0      ADDL chkSum, R1, chkSum                            ; R18, R1, R18          ; 030644
F67FFFE2      02F4      BNE R19, Ls23                                     ; 030639
C3E00036      02F8      BR __Implicit43
2FFE0000      02FC      UNOP
                0300      Ls27:
2C340000      0300      LDQ_U R1, (R20)                                  ; 030644
47E03413      0304      MOV_1, R19
2E140001      0308      LDQ_U R16, 1(R20)                                ; 030639
23540001      030C      LDA R26, 1(R20)                                  ; 030644
440072D3      0310      CMOVLBC idx, 3, R19                              ; R0, 3, R19
4680F11C      0314      BIC R20, 7, R28
40003000      0318      ADDL idx, 1, idx                                 ; R0, 1, R0
22940001      031C      LDA p, 1(R20)                                     ; R20, 1(R20)
A3FC0200      0320      LDL R31, 512(R28)
483A0F41      0324      EXTQH R1, R26, R1
4A1400D0      0328      EXTBL R16, R20, R16                              ; 030639
22940001      032C      LDA p, 1(R20)                                     ; R20, 1(R20)          ; 030644
48271781      0330      SRA R1, 56, R1
2021FFD0      0334      LDA R1, -48(R1)
4C330001      0338      MULL R1, R19, R1
42410012      033C      ADDL chkSum, R1, chkSum                            ; R18, R1, R18
E6000024      0340      BEQ R16, __Implicit43                             ; 030639
2E74FFFF      0344      LDQ_U R19, -1(R20)                               ; 030644
4A740F53      0348      EXTQH R19, R20, R19
47E03410      034C      MOV_1, R16
4A671781      0350      SRA R19, 56, R1
440072D0      0354      CMOVLBC idx, 3, R16                              ; R0, 3, R16
2E740000      0358      LDQ_U R19, (R20)                                  ; 030639
2021FFD0      035C      LDA R1, -48(R1)                                  ; 030644
4C300001      0360      MULL R1, R16, R1
4A7400D0      0364      EXTBL R19, R20, R16                              ; 030639
40003000      0368      ADDL idx, 1, idx                                 ; R0, 1, R0          ; 030644
22940001      036C      LDA p, 1(R20)                                     ; R20, 1(R20)
4A740F53      0370      EXTQH R19, R20, R19
```

RISC – AXP/Alpha

- Probleme & Herausforderungen:
 - Pipeline muss immer maximal gefüllt sein
 - Caches vorausschauend belegen
 - Sprünge (und Cache)? → branch prediction
 - Rolle der Compiler: RISC = „relegate important stuff (to the) compiler“
 - Read/Write Order Probleme, Conditional Writes
 - größerer Speicher erforderlich bei gleichem Code (auch auf der Disk)
 - MP-Systeme und Caches

EPIC – Itanium IA64

- Herausforderung: immer schnelleres „Silizium“ auch wirklich ausnutzen
- Verteilung aller Aufgaben auf viele parallel arbeitende Einheiten
- EPIC ist Weiterentwicklung von RISC (insbesondere, was die LOAD/STORE-Architektur betrifft)
- Ansätze Weiterentwicklung RISC (VLIW-Versionen):
 - dynamische Verteilung: Hardware analysiert den Befehlsstrom und versucht eine optimale Parallelisierung zu erzeugen
 - programmstatisch: Compiler erzeugen optimierten Code → es müssen genug Ressourcen in der CPU vorhanden sein und die CPU muss sich mit Zusatzinformationen über die Parallelisierung füttern lassen

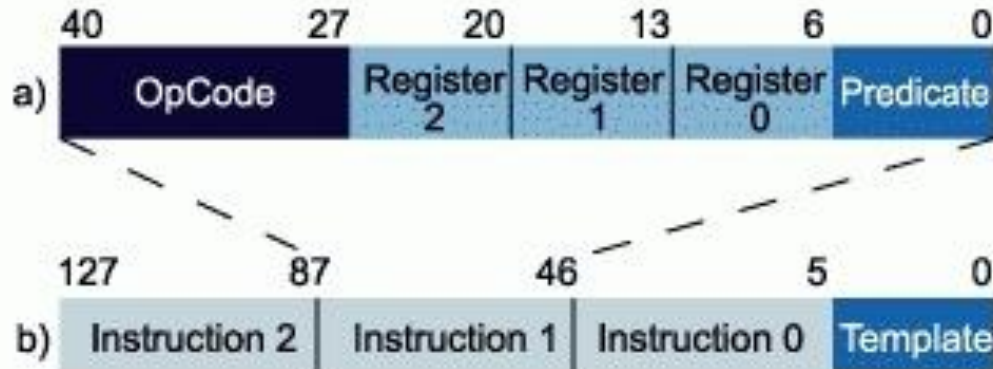
EPIC – Itanium IA64

- Merkmale EPIC (IA-64)
 - programmstatisch → Compiler generiert „optimalen“ Code, damit keine komplexe Hardware zur dynamischen Parallelisierung
 - Mechanismen, die dem Compiler ein Instruction Scheduling auf effizient Weise ermöglichen
 - „langes“ Befehlswort (128 Bit) → 3 Befehle gleichzeitig → Parallelisierung
 - genügend Ressourcen (z.B. 128 64-Bit I-Register, 128 82-Bit FP-Register, 64 1-Bit Branch Prediction Register) um Operationen parallel abzuarbeiten und parallel zu speichern
 - Möglichkeiten, die Hardware mit zusätzlichen Informationen (insbesondere Parallelisierung, Branch Prediction, ...) zu versorgen → spezielles Format „Befehlswort“, halt eben VLIW

EPIC – Itanium IA64

- Befehlsformat:
 - 3 * 41 Bit + „Template Informationen“
 - Template: sagt, aus welchen Gruppen die Befehle kommen → darüber Zuordnung zu (parallel nutzbaren) Funktionseinheiten
 - 41-Bit: Befehl (Befehlsslot)
 - Befehle: 3-Operanden Register-To-Register
 - Funktionseinheiten mehrfach (z.B. mehrere Integer-ALU, FP-ALU, ...)

EPIC – Itanium IA64



Itanium Instruktionsformat: a) Einzelner Befehl, b) gesamtes Bundle.

- Template: codiert parallel und unabhängig voneinander ausführbare Einzelbefehle
- nicht alle Gruppierungen sind erlaubt
- ist Aufgabe des Compilers, dies zu erstellen

EPIC – Itanium IA-64

- Funktionseinheiten:
 - Integer – ALU
 - Memory (Load/Store)
 - Floating Point
 - Branch
 - Extended (z.B. bei 64-Bit Operanden (Literalen))

Beispielcode IA64

```
TextPad - C:\Dokumente und Einstellungen\spi\Eigene Dateien\Decus\2008\cs-i64-noop.lis
File Edit Search View Tools Macros Configure Window Help
cs-i64-noop.lis
2470  chkDgt = 10 - (chkSum % 10) ;
2471  if (chkDgt == 10)
2472      chkDgt = 0 ;
2473
2474  printf ("* check sum (4): %s - %d / %d -> check digit: %d\n", shipmentNR, chkSum, chkSum % 10, chkDgt) ;
2475 }
2476
-----
Source Listing                               11-SEP-2008 12:41:04  HP C++ V7.3-023                Page 3
                                             11-SEP-2008 12:41:03  DsSPI:[WORK]CS.CXX;3
-----
Machine Code Listing
-----

NOTICE

There are two types of machine code listings supported by this
compiler. The compiler determines which type to use based on
whether or not the user is generating an object file as part of the
compilation.

/LIST/MACHINE_CODE/OBJECT

When the user is generating an object file, the machine code listing
uses a feature of ANALYZE/OBJECT to disassemble the actual
instructions used in the object file.

/LIST/MACHINE_CODE/NOOBJECT

When the user is not generating an object file, the compiler
generates a machine code listing which includes source correlation
information. This type of machine code listing may or may not
accurately reflect the instructions actually used in an object file.
This type of machine code listing should only be used to aid in
debugging. It is known to have defects, but is still useful to
obtain the correlation between the source code and the machine
instructions.

The following machine code listing is a disassembly of the object
file produced by ANALYZE/OBJECT/DISASSEMBLE.

-----
Analyze/Disassemble Object File              11-SEP-2008 12:41:04.45                Page 1
DsSPI:[WORK]CS.OBJ;2
ANALYZ I01-55

Machine Code Listing                          11-SEP-2008 12:41                HP C++ V7.3-023 for OpenVMS I6
DsSPI:[WORK]CS.CXX;3

131 68 Read Ovr Block Sync Rec Caps
```

Beispielcode IA64

```
TextPad - C:\Dokumente und Einstellungen\spi\Eigene Dateien\Decus\2008\cs-i64-noop.lis
File Edit Search View Tools Macros Configure Window Help
cs-i64-noop.lis
-----
000000000000 1: I19 break.i 000000
000000000000 2: I19 break.i 000000 }
-----
CX3s_Z9CHECKSUM3PC0AU0PU7:
84000804.60467A33 00C00580.36810809 00000B00: { .mmi
002C01B40840 0: M34 alloc r33 = ar.pfs, 1B, 05, 00 // out0==r59..out4==r63
0119E8CC0300 1: A4 add sp = -0120, sp
0108001008C0 2: A4 mov r35 = gp ;; }
00C40004.40000200 00000001.00000000 00000B10: { .mii
000008000000 0: M48 nop.m 000000
000008000000 1: I18 nop.i 000000
000188000880 2: I22 mov r34 = rp }
00586004.C0420080 02501018.18012009 00000B20: { .mmi
0080C0C00900 0: M1 lds r36 = [sp]
010802000940 1: A4 mov r37 = r32
0000B0C00980 2: I29 sxt4 r38 = sp ;; }
-----
Analyze/Disassemble Object File 11-SEP-2008 12:41:04.53 Page 22
DSSPI:[WORK]CS.OBJ;2
ANALYZ I01-55
00586005.0023309D 28002101.4DC1380B 00000B30: { .mmi
01080A6E09C0 0: A4 add r39 = 00F0, r38 ;;
008CC274A000 1: M4 st8 [r39] = r37
0000B0C00A00 2: I29 sxt4 r40 = sp ;; }
00586005.602020A4 02A02101.51C1480B 00000B40: { .mmi
01080A8E0A40 0: A4 add r41 = 00F0, r40 ;;
008082900A80 1: M1 ld4 r42 = [r41]
0000B0C00AC0 2: I29 sxt4 r43 = sp ;; }
00586005.A02320B1 50002100.5681600B 00000B50: { .mmi
010802B40B00 0: A4 add r44 = 0020, r43 ;;
008C82C54000 1: M4 st4 [r44] = r42
0000B0C00B40 2: I29 sxt4 r45 = sp ;; }
00586005.E02320B8 00002100.5AA1700B 00000B60: { .mmi
010802D50B80 0: A4 add r46 = 0028, r45 ;;
008C82E00000 1: M4 st4 [r46] = r0
0000B0C00BC0 2: I29 sxt4 r47 = sp ;; }
00586006.202320C0 00002100.5EC1800B 00000B70: { .mmi
010802F60C00 0: A4 add r48 = 0030, r47 ;;
008C83000000 1: M4 st4 [r48] = r0
0000B0C00C40 2: I29 sxt4 r49 = sp ;; }
00586006.602320C8 00002100.62E1900B 00000B80: { .mmi
010803170C80 0: A4 add r50 = 0038, r49 ;;
008C83200000 1: M4 st4 [r50] = r0
0000B0C00CC0 2: I29 sxt4 r51 = sp ;; }
00586006.A02300D0 00002100.6701A00B 00000B90: { .mmi
010803380D00 0: A4 add r52 = 0040, r51 ;;
```

Beispielcode IA64

```
TextPad - C:\Dokumente und Einstellungen\spi\Eigene Dateien\Decus\2008\cs-i64-noop.lis
File Edit Search View Tools Macros Configure Window Help
cs-i64-noop.lis
000008000000 2: I18 nop.i 000000 ;;
0051C807.40000200 00001000.7001C80B 00000BC0: { .mii
008003800E40 0: M1 ld1 r57 = [r56] ;;
000008000000 1: M48 nop.m 000000
0000A3900E80 2: I29 sxt1 r58 = r57 ;;
}
-----
Analyze/Disassemble Object File 11-SEP-2008 12:41:04.53 Page 23
DSSPI:[WORK]CS.OBJ;2
ANALYZ I01-55
00040000.00420032 81403988.74A80001 00000BD0: { .mii
01CC43A54000 0: A8 cmp4.eq p0, p8 = 2A, r58
010800CA0500 1: A4 add r20 = 0050, sp
000008000000 2: I18 nop.i 000000 ;;
}
00040000.00480000 01301198.288C0001 00000BE0: { .mii
008CC1446000 0: M4 st8 [r20] = r35
0120000004C0 1: A5 mov r19 = 000000
000008000000 2: I18 nop.i 000000 ;;
}
00040000.00420032 C1402400.00049901 00000BF0: { .mii
0120000024C8 0: A5 (p8) mov r19 = 000001
010800CB0500 1: A4 add r20 = 0058, sp
000008000000 2: I18 nop.i 000000 ;;
}
42000088.04000200 00001198.284C0011 00000C00: { .mib
008CC1426000 0: M4 st8 [r20] = r19
000008000000 1: I18 nop.i 000000
008400011008 2: B1 (p8) br.cond.dptk.many $+0000080 ;;
}
00040000.00002C30 02300001.00000001 00000C10: { .mii
000008000000 0: M48 nop.m 000000
0000B0C008C0 1: I29 sxt4 r35 = sp
000008000000 2: I18 nop.i 000000 ;;
}
00040000.00202090 02502100.4681200B 00000C20: { .mii
010802340900 0: A4 add r36 = 0020, r35 ;;
008082400940 1: M1 ld4 r37 = [r36]
000008000000 2: I18 nop.i 000000 ;;
}
00040000.00002C94 02600001.00000001 00000C30: { .mii
000008000000 0: M48 nop.m 000000
0000B2500980 1: I29 sxt4 r38 = r37
000008000000 2: I18 nop.i 000000 ;;
}
00040000.00002C30 02802100.4C053801 00000C40: { .mii
0108026029C0 0: A4 add r39 = 0001, r38
0000B0C00A00 1: I29 sxt4 r40 = sp
000008000000 2: I18 nop.i 000000 ;;
}
00586005.402320A5 38002100.5081480B 00000C50: { .mii
010802840A40 0: A4 add r41 = 0020, r40 ;;
008C8294E000 1: M4 st4 [r41] = r39
000008000000 2: I29 sxt4 r42 = sp ;;
}
```

RISC → EPIC

- Steigerung der Parallelität
 - paralleles Laden der Befehle mit VLIW
 - bessere Vorgabe und Anordnung der parallelen Abarbeitung (Template im VLIW)
 - noch mehr, unabhängiger ALU's (sowohl Integer als auch Floating Point) der CPU
- Multicore → andere Ebene → nicht Architektur eines Cores (einer CPU)

Evolution?

- CISC-RISC-EPIC: alle von Neumann
- technologische Probleme (Bus, Physik der Peripherie im Gegensatz zu Strukturbreiten im Chip) werden durch Parallelisierung gelöst
- Entwicklung zwangsweise: **PARALLELITÄT**
- es gilt der „Satz von der Erhaltung der Komplexität“ → Probleme Caches, Branch Prediction, Kosten „unaligned access“, Rolle der Compiler, Plattenplatz, ...

Fragen & Diskussionen

- Fragen
- Hinweise
- Tipps

