

Überblick

- Einleitung
 - Lit., Motivation, Geschichte, v. Neumann-Modell, VHDL
- Befehlsschnittstelle
- Mikroarchitektur
- Speicherarchitektur
- Ein-/Ausgabe
- Multiprozessorsysteme, ...

JR - RA - SS02

Kap. 3.2

3.2/1

Kap.3 Mikroarchitektur

**Prozessoren,
interne Sicht**

- **3.1** elementare Datentypen, Operationen und ihre Realisierung (siehe 2.1)
- **3.2** Mikroprogrammierung
- **3.3** Einfache Implementierung von MIPS
- **3.4** Pipelining

JR - RA - SS02

Kap. 3.2

3.2/3

3.2 Mikroprogrammierung

- Realisierung eines Befehlssatzes auf einer Hardware-Struktur
- Befehlsabarbeitung bei CISC und ihre Folgen

JR - RA - SS02

Kap. 3.2

3.2/4

Zur Erinnerung: CISC

- Charakterisierung eines CISC Rechners
 - grosser Satz von Maschinenbefehle, zum Teil auch Maschinenbefehle, die recht komplexe Operationen auszuführen haben
 - sehr unterschiedliche Ausführungszeiten der verschiedenen Maschinenbefehle
 - Steuerung nicht durch Hardware, sondern durch ein Mikroprogramm, das in einem ROM auf dem Prozessor abgespeichert ist.
- Erster moderner Rechner (IBM 369) war ein CISC Rechner
- Ein "kleiner" CISC-Rechner zur Illustration

JR - RA - SS02

Kap. 3.2

3.2/5

Maschinen(spiel)sprache (Teil A)

- Drei für den Benutzer sichtbare Register:
 - ac (Akkumulator)
 - sp (Stackpointer)
 - pc (Befehlszähler)
- Die Maschinensprache

■ 0000xxxxxxxx	LODD	ac:=mem[x]	// 12-Bit Adressenbus
■ 0001xxxxxxxx	STOD	mem[x]:=ac	
■ 0010xxxxxxxx	ADDD	ac:=ac+mem[x]	
■ 0011xxxxxxxx	SUBD	ac:=ac-mem[x]	
■ 0100xxxxxxxx	JPOS	if ac>0 then pc:=x	
■ 0101xxxxxxxx	JZER	if ac=0 then pc:=x	
■ 0110xxxxxxxx	JUMP	pc:=x	
■ 0111xxxxxxxx	LOCO	ac:=x (0≤x≤4095)	
■ 1000xxxxxxxx	LODL	ac:=mem[sp+x]	
■ 1001xxxxxxxx	STOL	mem[x+sp]:=ac	

JR - RA - SS02

Kap. 3.2

3.2/6

Maschinen(spiel)sprache (Teil B)

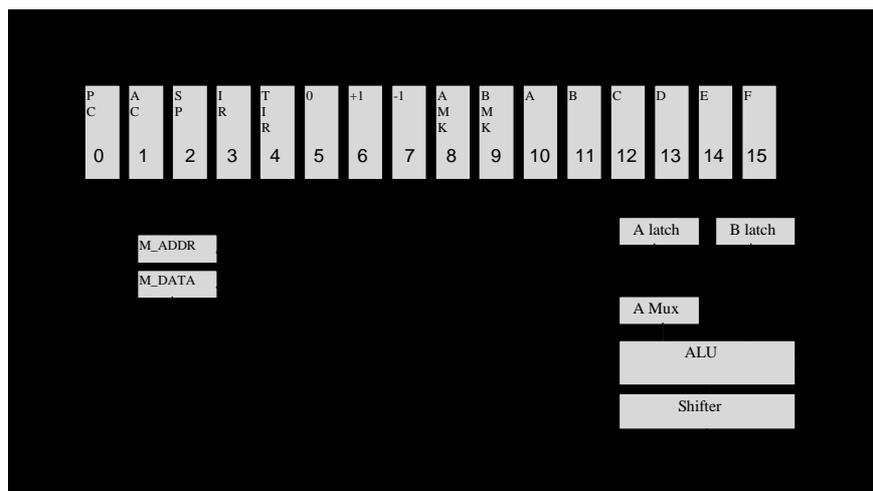
I	1010xxxxxxxxxxxx	ADDL	ac:=ac+mem[sp+x]
I	1011xxxxxxxxxxxx	SUBL	ac:=ac-mem[sp+x]
I	1100xxxxxxxxxxxx	JNEG	if ac < 0 then pc:=x;
I	1101xxxxxxxxxxxx	JNEZ	if ac ≠ 0 then pc:=x;
I	1110xxxxxxxxxxxx	CALL	sp:=sp-1; mem[sp]:=pc; pc:=x;
I	1111000.....	PSHI	sp:=sp-1; mem[sp]:=m[ac];
I	1111001.....	POPI	mem[ac]:=mem[sp]; sp:=sp+1;
I	1111010.....	PUSH	sp:=sp-1; mem[sp]:=ac;
I	1111011.....	POP	ac:=mem[sp]; sp:=sp+1;
I	1111100.....	RTN	pc:=mem[sp]; sp:=sp+1;
I	1111101.....	SWAP	tmp:=sp; sp:=ac; ac:=tmp;
I	1111110..yyyyyyy	INSP	sp:=sp+y; (0≤y≤255)
I	1111111..yyyyyyy	DESP	sp:=sp-y; (0≤y≤255)

JR - RA - SS02

Kap. 3.2

3.2/7

Hardware eines Spielrechners: Datenpfad

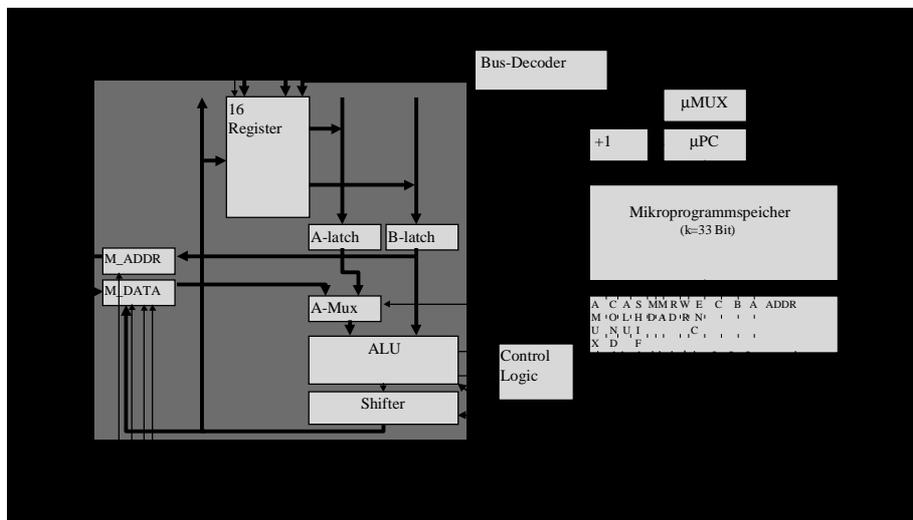


JR - RA - SS02

Kap. 3.2

3.2/8

Gesamtarchitektur des (Spiel-)Rechners



Aufbau eines Mikroprogrammbefehls

1	AMUX	0 → A-Latch;	1 → M_DATA
2-4	COND	0 → No jump;	1 → Jump, if N;
		2 → Jump, if Z;	3 → Jump
		4 → Jump, if Wait	
5-6	ALU	0 → A+B;	1 → AB;
		2 → A;	3 → A'
7-8	SHIFT	0 → No shift;	1 → right shift
		2 → left shift	
9	MD	0 → No load;	1 → load
10	MA	0 → No load;	1 → load
11	RD	0 → No read;	1 → read
12	WR	0 → No write;	1 → write
13	ENC	0 → disable C;	1 → enable C
14-25	A, B, C	determine the active registers	
26-33	ADDR	Microprogram jump address	

Zu lösende Arbeit

- Mikroprogramm, das auf unserer Spielhardware jedes Programm, das in der Maschinen(spiel)sprache geschrieben ist, korrekt ausführt.

⇔ Mikroprogrammzyklus

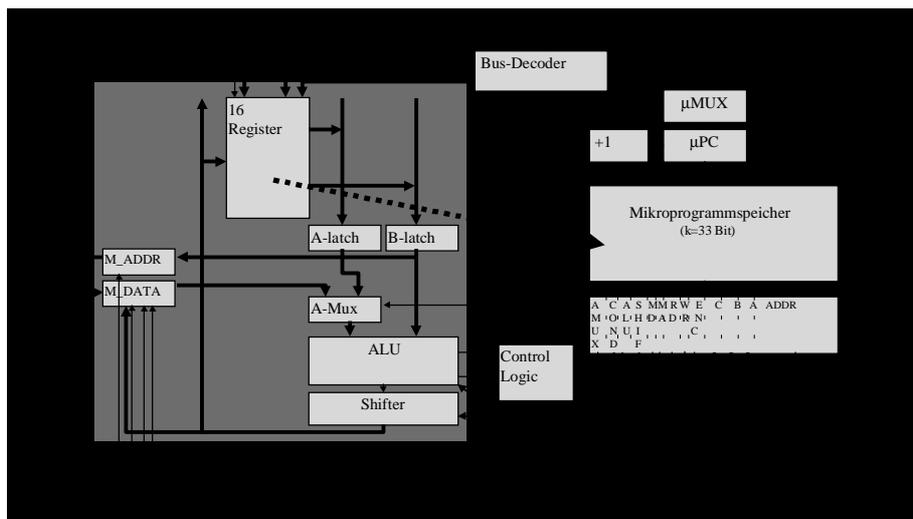
- Lade den durch PC adressierten Befehl aus dem Hauptspeicher und schreibe ihn in das Instruktionsregister IR
- Dekodiere den in IR abgespeicherten Befehl
- Lege die Operanden auf den A- bzw. B-Bus
(Achtung: es können wait-Zykeln auftreten)
- Führe die verlangte Operation in der ALU und dem Shifter aus
- Speichere das Ergebnis ab.

JR - RA - SS02

Kap. 3.2

3.2/11

Wie kann ein Maschinenbefehl dekodiert/ausgeführt werden ?



Lesbare Schreibweise für Mikrobefehle

- Mikroprogramme sind in ihrer eigentlichen Form schwer lesbar

0 000 00 00 0 0 0 0 1 0001 0001 1010 00000000

- Mikroinstruktionen sind aber, wie gesehen, in Bereiche eingeteilt.

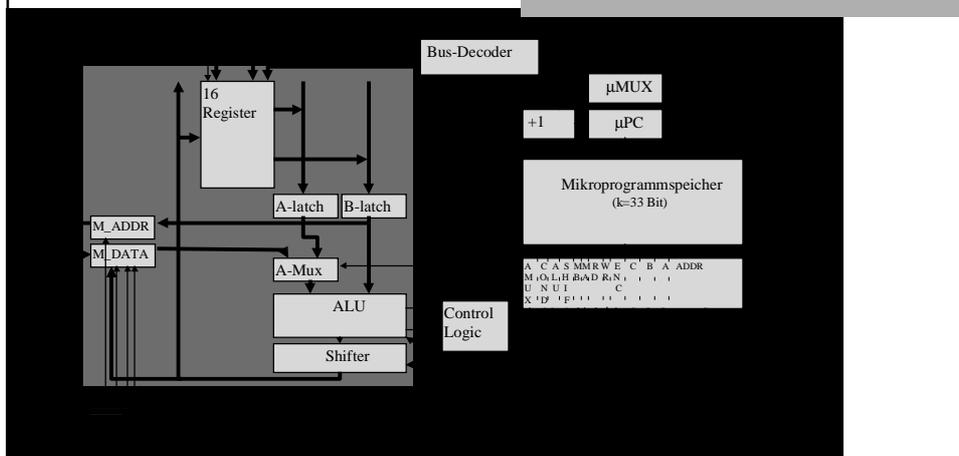
⇒ Schreibe im folgenden nur die Teile hin, die verschieden von 0 sind

ENC=1, C=1, B=1, A=10

bzw. verwende eine PASCAL-ähnliche Notation

C:=A+B;

1	AMUX	0 → A-Latch; 1 → M_DATA
2-4	COND	0 → No jump; 1 → Jump, if N; 2 → Jump, if Z; 3 → Jump 4 → Jump, if Wait
5-6	ALU	0 → A+B; 1 → AB; 2 → A; 3 → A'
7-8	SHIFT	0 → No shift; 1 → right shift 2 → left shift
9	MD	0 → No load; 1 → load
10	MA	0 → No load; 1 → load
11	RD	0 → No read; 1 → read
12	WR	0 → No write; 1 → write
13	ENC	0 → disable C; 1 → enable C
14-25	A, B, C	determine the active registers
26-33	ADDR	Microprogram jump address



Beispiele für erlaubte "PASCAL"-Notationen

- M_ADDR:=PC; RD // ALU=2; MA=1; RD=1; MD=1;
- RD; // ALU=2; RD=1; MD=1
- IR:=M_DATA // ALU=2; AMUX=1; ENC=1; C=3
- PC:=PC+1 // ENC=1; B=6;
- M_ADDR:=IR; M_DATA:=AC; WR // ALU=2; MD=1; MA=1; WR=1; A=1; B=3;
- ALU:=TIR; If N then goto 15 // ALU=2; COND=1; A=4; ADDR=15

JR - RA - SS02

Kap. 3.2

3.2/15

Interpretation des Maschinenbefehls ADDD

Befehl ADDD: 0010xxxxxxxxxxxx ac:=ac+m[x]

Interpretation durch das Mikroprogramm

1. M_ADDR:=PC; RD; // hole den Befehl aus dem Hauptspeicher
2. If WAIT then goto 2; MD:=1; // warten bis Hauptspeicher Datum liefert
3. PC:=PC+1; // erhöhen des Befehlszählers
4. IR:=M_DATA; // schreibe Befehl ins Instruktionsregister
5. // Beginn Dekodierung
- ...
- x. // Ende Dekodierung
- x+1. M_ADDR:=TIR; RD // Lese m[x]
- x+2. If WAIT then goto x+2; MD:=1; // warten bis Hauptspeicher Datum liefert
- x+3. AC:=AC+M_DATA;

JR - RA - SS02

Kap. 3.2

3.2/16

Aufbau eines Mikroprogramms

- Enthält die Maschinensprache n Instruktionen, so besteht das Mikroprogramm aus $n+1$ Teilen:
 - ein Block zum Dekodieren der Befehle
 - zu jeder Maschineninstruktion ein Mikrocode
- Das Mikroprogramm ist im Mikroprogramm Speicher (ROM) vom Hersteller abgelegt und für den Programmierer nicht zugreifbar

JR - RA - SS02

Kap. 3.2

3.2/19

CISC/RISC

