

Hinweise – Dekodierung

Dr.-Ing. Volkmar Sieh

Department Informatik 4
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universität Erlangen-Nürnberg

WS 2017/2018



Beispiel CPU 6510 (Prozessor Commodore-64):

	0	1	2	3	4	5	6	L 7
0	BRK	ORA(z,x)	JAM i	SLO(z,x)	NOP z	ORA z	ASL z	SLO z
1	BPL r	ORA(z),y	JAM i	SLO(z),y	NOP z,x	ORA z,x	ASL z,x	SLO z,x
2	JSR a	AND(z,x)	JAM i	RLA(z,x)	BIT z	AND z	ROL z	RLA z
3	BMI r	AND(z),y	JAM i	RLA(z),y	NOP z,x	AND z,x	ROL z,x	RLA z,x
4	RTI s	EOR(z,x)	JAM i	SRE(z,x)	NOP z	EOR z	LSR z	SRE z
5	BVC r	EOR(z),y	JAM i	SRE(z),y	NOP z,x	EOR z,x	LSR z,x	SRE z,x
6	RTS s	ADC(z,x)	JAM i	RRA(z,x)	NOP z	ADC z	ROR z	RRA z
7	BVS r	ADC(z),y	JAM i	RRA(z),y	NOP z,x	ADC z,x	ROR z,x	RRA z,x
8	NOP #	STA(z,x)	NOP #	SAX(z,x)	STY z	STA z	STX z	SAX z
9	BCC r	STA(z),y	JAM i	SHA a,x	STYz,x	STA z,x	STX z,y	SAX z,y
A	LDY #	LDA(z,x)	LDX #	LAX(z,x)	LDY z	LDA z	LDX z	LAX z
B	BCS r	LDA(z),y	JAM i	LAX(z),y	LDY z,x	LDA z,x	LDX z,y	LAX z,y
C	CPY #	CMP(z,x)	NOP #	DCP(z,x)	CPY z	CMP z	DEC z	DCP z
D	BNE r	CMP(z),y	JAM i	DCP(z),y	NOP z,x	CMP z,x	DEC z,x	DCP z,x
E	CPX #	SBC(z,x)	NOP #	ISB(z,x)	CPX z	SBC z	INC z	ISB z
H F	BEQ r	SBC(z),y	JAM i	ISB(z),y	NOP z,x	SBC z,x	INC z,x	ISB z,x



	8	9	A	B	C	D	E	L F
0	PHP s	ORA #	ASL A	ANC #	NOP a	ORA a	ASL a	SLO a
1	CLC i	ORA a,y	NOP i	SLO a,y	NOP a,x	ORA a,x	ASL a,x	SLO a,x
2	PLP s	AND #	ROL A	ANC #	BIT a	AND a	ROL a	RLA a
3	SEC i	AND a,y	NOP i	RLA a,y	NOP a,x	AND a,x	ROL a,x	RLA a,x
4	PHA s	EOR #	LSR A	ASR #	JMP a	EOR a	LSR a	SRE a
5	CLI i	EOR a,y	NOP i	SRE a,y	NOP a,x	EOR a,x	LSR a,x	SRE a,x
6	PLA s	ADC #	ROR A	ARR #	JMP (a)	ADC a	ROR a	RRA a
7	SEI i	ADC a,y	NOP i	RRA a,y	NOP a,x	ADC a,x	ROR a,x	RRA a,x
8	DEY i	NOP #	TXA i	ANE #	STY a	STA a	STX a	SAX a
9	TYA i	STA a,y	TXS i	SHS a,x	SHY a,y	STA a,x	SHX a,y	SHA a,y
A	TAY i	LDA #	TAX i	LXA #	LDY a	LDA a	LDX a	LAX a
B	CLV i	LDA a,y	TSX i	LAE a,y	LDY a,x	LDA a,x	LDX a,y	LAX a,y
C	INY i	CMP #	DEX i	SBX #	CPY a	CMP a	DEC a	DCP a
D	CLD i	CMP a,y	NOP i	DCP a,y	NOP a,x	CMP a,x	DEC a,x	DCP a,x
E	INX i	SBC #	NOP i	SBC #	CPX a	SBC a	INC a	ISB a
H F	SED i	SBC a,y	NOP i	ISB a,y	NOP a,x	SBC a,x	INC a,x	ISB a,x



Beispiel CPU 6510 (Prozessor Commodore-64):

	0	1	2	3	4	5	6	L 7
0	BRK	ORA(z,x)	JAM i	SLO(z,x)	NOP z	ORA z	ASL z	SLO z
1	BPL r	ORA(z),y	JAM i	SLO(z),y	NOP z,x	ORA z,x	ASL z,x	SLO z,x
2	JSR a	AND(z,x)	JAM i	RLA(z,x)	BIT z	AND z	ROL z	RLA z
3	BMI r	AND(z),y	JAM i	RLA(z),y	NOP z,x	AND z,x	ROL z,x	RLA z,x
4	RTI s	EOR(z,x)	JAM i	SRE(z,x)	NOP z	EOR z	LSR z	SRE z
5	BVC r	EOR(z),y	JAM i	SRE(z),y	NOP z,x	EOR z,x	LSR z,x	SRE z,x
6	RTS s	ADC(z,x)	JAM i	RRA(z,x)	NOP z	ADC z	ROR z	RRA z
7	BVS r	ADC(z),y	JAM i	RRA(z),y	NOP z,x	ADC z,x	ROR z,x	RRA z,x
8	NOP #	STA(z,x)	NOP #	SAX(z,x)	STY z	STA z	STX z	SAX z
9	BCC r	STA(z),y	JAM i	SHA a,x	STYz,x	STA z,x	STX z,y	SAX z,y
A	LDY #	LDA(z,x)	LDX #	LAX(z,x)	LDY z	LDA z	LDX z	LAX z
B	BCS r	LDA(z),y	JAM i	LAX(z),y	LDY z,x	LDA z,x	LDX z,y	LAX z,y
C	CPY #	CMP(z,x)	NOP #	DCP(z,x)	CPY z	CMP z	DEC z	DCP z
D	BNE r	CMP(z),y	JAM i	DCP(z),y	NOP z,x	CMP z,x	DEC z,x	DCP z,x
E	CPX #	SBC(z,x)	NOP #	ISB(z,x)	CPX z	SBC z	INC z	ISB z
H F	BEQ r	SBC(z),y	JAM i	ISB(z),y	NOP z,x	SBC z,x	INC z,x	ISB z,x



	8	9	A	B	C	D	E	L F
0	PHP s	ORA #	ASL A	ANC #	NOP a	ORA a	ASL a	SLO a
1	CLC i	ORA a,y	NOP i	SLO a,y	NOP a,x	ORA a,x	ASL a,x	SLO a,x
2	PLP s	AND #	ROL A	ANC #	BIT a	AND a	ROL a	RLA a
3	SEC i	AND a,y	NOP i	RLA a,y	NOP a,x	AND a,x	ROL a,x	RLA a,x
4	PHA s	EOR #	LSR A	ASR #	JMP a	EOR a	LSR a	SRE a
5	CLI i	EOR a,y	NOP i	SRE a,y	NOP a,x	EOR a,x	LSR a,x	SRE a,x
6	PLA s	ADC #	ROR A	ARR #	JMP (a)	ADC a	ROR a	RRA a
7	SEI i	ADC a,y	NOP i	RRA a,y	NOP a,x	ADC a,x	ROR a,x	RRA a,x
8	DEY i	NOP #	TXA i	ANE #	STY a	STA a	STX a	SAX a
9	TYA i	STA a,y	TXS i	SHS a,x	SHY a,y	STA a,x	SHX a,y	SHA a,y
A	TAY i	LDA #	TAX i	LXA #	LDY a	LDA a	LDX a	LAX a
B	CLV i	LDA a,y	TSX i	LAE a,y	LDY a,x	LDA a,x	LDX a,y	LAX a,y
C	INY i	CMP #	DEX i	SBX #	CPY a	CMP a	DEC a	DCP a
D	CLD i	CMP a,y	NOP i	DCP a,y	NOP a,x	CMP a,x	DEC a,x	DCP a,x
E	INX i	SBC #	NOP i	SBC #	CPX a	SBC a	INC a	ISB a
H F	SED i	SBC a,y	NOP i	ISB a,y	NOP a,x	SBC a,x	INC a,x	ISB a,x



Beispiel CPU 6510 (Prozessor Commodore-64):

	0	1	2	3	4	5	6	L 7
0	BRK	ORA(z,x)	JAM i	SLO(z,x)	NOP z	ORA z	ASL z	SLO
1	BPL r	ORA(z,y)	JAM i	SLO(z,y)	NOP z,x	ORA z,x	ASL z,x	SLO
2	JSR a	AND(z,x)	JAM i	RLA(z,x)	BIT z	AND z	ROL z	RLA
3	BMI r	AND(z,y)	JAM i	RLA(z,y)	NOP z,x	AND z,x	ROL z,x	RLA
4	RTI s	EOR(z,x)	JAM i	SRE(z,x)	NOP z	EOR z	LSR z	SRE
5	BVC r	EOR(z,y)	JAM i	SRE(z,y)	NOP z,x	EOR z,x	LSR z,x	SRE
6	RTS s	ADC(z,x)	JAM i	RRA(z,x)	NOP z	ADC z	ROR z	RRA
7	BVS r	ADC(z,y)	JAM i	RRA(z,y)	NOP z,x	ADC z,x	ROR z,x	RRA
8	NOP #	STA(z,x)	NOP #	SAX(z,x)	STY z	STA z	STX z	SAX
9	BCC r	STA(z,y)	JAM i	SHA a,x	STY z,x	STA z,x	STX z,y	SAX
A	LDY #	LDA(z,x)	LDX #	LAX(z,x)	LDY z	LDA z	LDX z	LAX
B	BCS r	LDA(z,y)	JAM i	LAX(z,y)	LDY z,x	LDA z,x	LDX z,y	LAX
C	CPY #	CMP(z,x)	NOP #	DCP(z,x)	CPY z	CMP z	DEC z	DCP
D	BNE r	CMP(z,y)	JAM i	DCP(z,y)	NOP z,x	CMP z,x	DEC z,x	DCP
E	CPX #	SBC(z,x)	NOP #	ISB(z,x)	CPX z	SBC z	INC z	ISB
H F	BEQ r	SBC(z,y)	JAM i	ISB(z,y)	NOP z,x	SBC z,x	INC z,x	ISB z



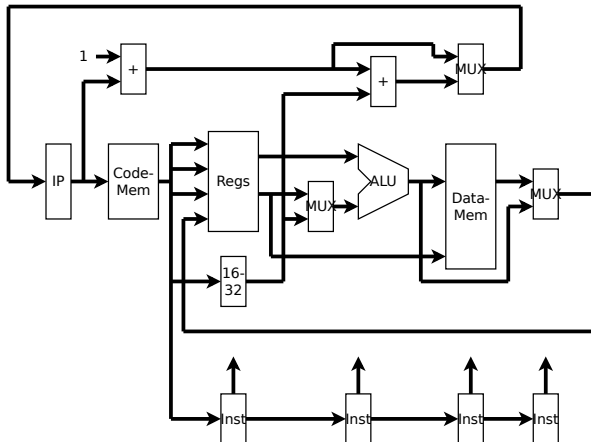
	8	9	A	B	C	D	E	L F
0	PHP s	ORA #	ASL A	ANC #	NOP a	ORA a	ASL a	SLO a
1	CLC i	ORA a,y	NOP i	SLO a,y	NOP a,x	ORA a,x	ASL a,x	SLO a,x
2	PLP s	AND #	ROL A	ANC #	BIT a	AND a	ROL a	RLA a
3	SEC i	AND a,y	NOP i	RLA a,y	NOP a,x	AND a,x	ROL a,x	RLA a,x
4	PHA s	EOR #	LSR A	ASR #	JMP a	EOR a	LSR a	SRE a
5	CLI i	EOR a,y	NOP i	SRE a,y	NOP a,x	EOR a,x	LSR a,x	SRE a,x
6	PLA s	ADC #	ROR A	ARR #	JMP (a)	ADC a	ROR a	RRA a
7	SEI i	ADC a,y	NOP i	RRA a,y	NOP a,x	ADC a,x	ROR a,x	RRA a,x
8	DEY i	NOP #	TXA i	ANE #	STY a	STA a	STX a	SAX a
9	TYA i	STA a,y	TXS i	SHS a,x	SHY a,y	STA a,x	SHX a,y	SHA a,y
A	TAY i	LDA #	TAX i	LXA #	LDY a	LDA a	LDX a	LAX a
B	CLV i	LDA a,y	TSX i	LAE a,y	LDY a,x	LDA a,x	LDX a,y	LAX a,y
C	INY i	CMP #	DEX i	SBX #	CPY a	CMP a	DEC a	DCP a
D	CLD i	CMP a,y	NOP i	DCP a,y	NOP a,x	CMP a,x	DEC a,x	DCP a,x
E	INX i	SBC #	NOP i	SBC #	CPX a	SBC a	INC a	ISB a
H	F	SED i	SBC a,y	NOP i	ISB a,y	NOP a,x	SBC a,x	INC a,x



Woher kommen diese Muster...?



Dekodierung – Pipelining



Einfachste Pipeline-Realisierung:

Jede Pipeline-Stufe verwendet jeweils für sie bestimmte Bits einer Instruktion:

Register-File: Bits für Register-Nummern

ALU: Bits für Operation (+, -, ...)

Multiplexer: Bits für Auswahl

...: Bits für ...

Weitergegebene Instruktion vermindert sich jeweils um die bereits ausgewerteten Bits.



Was heißt das für eine VM?



```
step() {  
    /* 1. Pipeline-Stufe: Instruktion holen */  
    inst = fetch(ip++);  
  
    /* 2. Pipeline-Stufe: Operanden holen */  
    op1 = reg_read((inst >> 0) & 7);  
    op2 = reg_read((inst >> 3) & 7);  
  
    /* 3. Pipeline-Stufe: Rechnen */  
    switch ((inst >> 6) & 7) {  
        case 0: res = op1 + op2; break;  
        case 1: res = op1 - op2; break;  
        case 2: res = op1 & op2; break;  
        ...  
    }  
  
    /* 4. Pipeline-Stufe: Ergebnis speichern */  
    reg_write((inst >> 9) & 7, res);  
}
```



In der Praxis: mehrere Instruktionsformate. Beispiel ARM:

		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		
Data processing immediate shift	cond [1]	0	0	0	opcode				S	Rn				Rd				shift amount				shift	0	Rm											
Miscellaneous instructions: See Figure A3-4	cond [1]	0	0	0	1	0	x	x	0	x x x x x x x x x x x x x x x x x x																					0	x x x x			
Data processing register shift [2]	cond [1]	0	0	0	opcode				S	Rn				Rd				Rs				0	shift	1	Rm										
Miscellaneous instructions: See Figure A3-4	cond [1]	0	0	0	1	0	x	x	0	x x x x x x x x x x x x x x x x x x																					0	x x x x			
Multiplies: See Figure A3-3 Extra load/stores: See Figure A3-5	cond [1]	0	0	0	x x x x x x				x x x x x x x x x x x x x x x x x x												1	x x x x													
Data processing immediate [2]	cond [1]	0	0	1	opcode				S	Rn				Rd				rotate				immediate													
Undefined instruction	cond [1]	0	0	1	1	0	x	0	0	x x x x x x x x x x x x x x x x x x																									
Move immediate to status register	cond [1]	0	0	1	1	0	R	1	0	Mask				SBO				rotate				immediate													
Load/store immediate offset	cond [1]	0	1	0	P	U	B	W	L	Rn				Rd				immediate																	
Load/store register offset	cond [1]	0	1	1	P	U	B	W	L	Rn				Rd				shift amount				shift	0	Rm											
Media instructions [4]: See Figure A3-2	cond [1]	0	1	1	x x x x x x x x x x x x x x x x x x																									1	x x x x				
Architecturally undefined	cond [1]	0	1	1	1	1	1	1	1	x x x x x x x x x x x x x x x x x x														1	1	1	1	x x x x							
Load/store multiple	cond [1]	1	0	0	P	U	S	W	L	Rn				register list																					
Branch and branch with link	cond [1]	1	0	1	L				24-bit offset																										
Coprocessor load/store and double register transfers	cond [3]	1	1	0	P	U	N	W	L	Rn				CRd				cp_num				8-bit offset													
Coprocessor data processing	cond [3]	1	1	1	0	opcode1				CRn				CRd				cp_num				opcode2	0	CRm											
Coprocessor register transfers	cond [3]	1	1	1	0	opcode1				L	CRn				Rd				cp_num				opcode2	1	CRm										
Software interrupt	cond [1]	1	1	1	1	swi number																													
Unconditional instructions: See Figure A3-6	1	1	1	1	x x																														



Grund für verschiedene Instruktions-Formate:

Unterschiedliche Befehle brauchen unterschiedliche Parameter.

Beispiele:

- Sprünge brauchen
 - einen (großen) Offset
- bedingte Sprünge brauchen
 - eine Bedingung
 - einen (nicht ganz so großen) Offset
- ALU-Operationen brauchen
 - drei Register-Nummernoder
 - zwei Register-Nummern
 - einen (größeren) Immediate-Wert

...



```
step() {  
    /* 1. Pipeline-Stufe: Instruktion holen */  
    inst = fetch(ip++);  
  
    switch (inst_format(inst) {  
    case ALU_REG_REG:  
        /* 2. Pipeline-Stufe: Operanden holen */  
        op1 = reg_read((inst >> 0) & 0xf);  
        op2 = reg_read((inst >> 4) & 0xf);  
  
        /* 3. Pipeline-Stufe: Rechnen */  
        res = alu((inst >> 24) & 0xf, op1, op2);  
  
        /* 4. Pipeline-Stufe: Ergebnis speichern */  
        reg_write((inst >> 20) & 0xf, res);  
        break;  
  
    case ALU_REG_IMM:  
        /* 2. Pipeline-Stufe: Operanden holen */  
        op1 = reg_read((inst >> 0) & 0xf);  
        op2 = (inst >> 4) & 0xffff;  
  
        /* 3. Pipeline-Stufe: Rechnen */  
        res = alu((inst >> 24) & 0xf, op1, op2);  
  
        /* 4. Pipeline-Stufe: Ergebnis speichern */  
        reg_write((inst >> 20) & 0xf, res);  
        break;  
  
    case ...  
        ...  
    }  
}
```

