Basic hacking tutorial vol.2

By Con

Contents

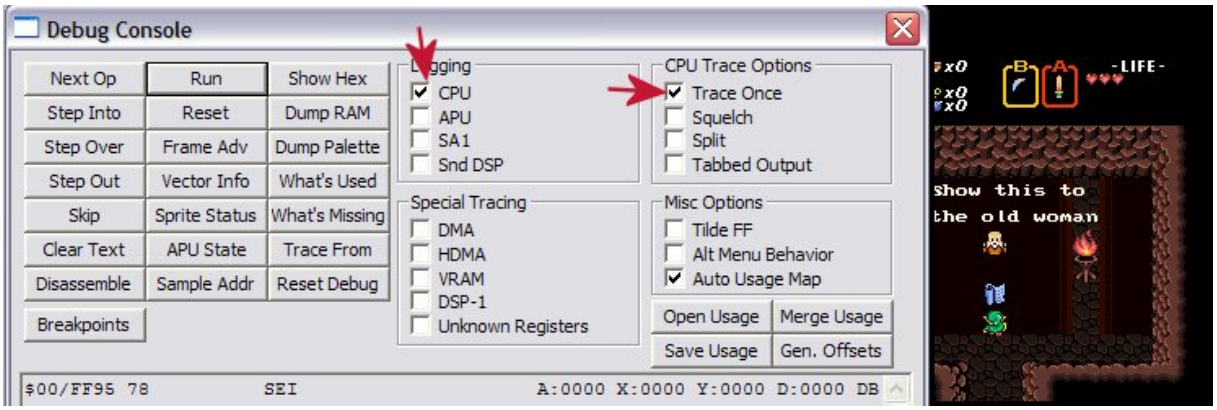
I. Introduction

This guide (vol.2) starts after your lessons from vol.1 are succeeded. Here I give some basic opcodes and you will learn how jumps and branches work.

II. Tracing with Geiger

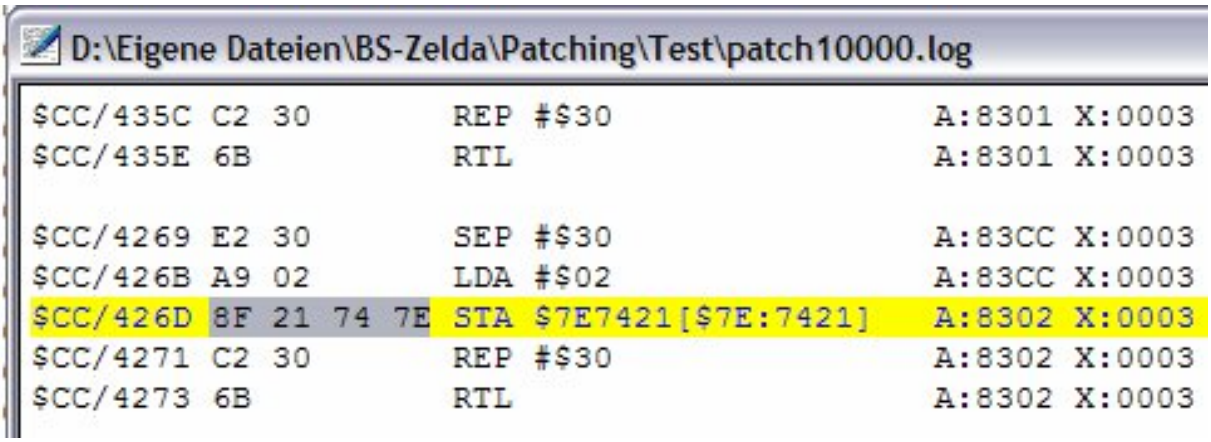
Tracing with Geiger is absolutely necessary for advanced rom hacking.

1. Start Geiger and emulate the rom "patch1.smc"
2. Walk into the cave with the letter
3. Just before you collect it click on the console the "Trace once" and "CPU"
4. Collect the letter and exit Geiger.



Information: **Tracing**
In a second thousands of opcodes and instructions are executed. Geiger can log all these instructions and summarizes them as a txt.
CPU are the executed opcodes, APU is necessary for the sound instructions (or so). The executed opcodes are often a loop, so you should click "Trace once". Otherwise you will get a log file of hundreds of MB. If you press "Trace once", already executed opcodes will not be logged a 2nd time.

Now you have a log file, the name is patch1000.txt.
1. Open it
2. search for 8f 21 74 7E (remember, our hack for the magic boomerang)



This is an example how you can find the executed opcodes, what is indispensable for later hacking!

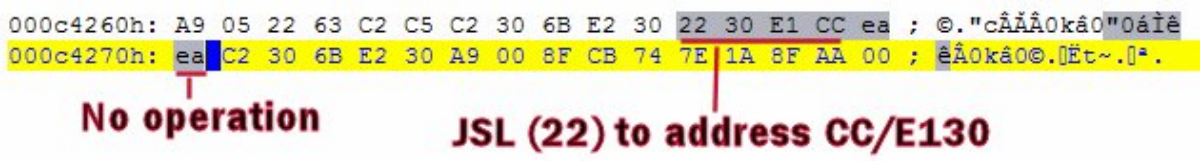
III. Jumps

Imagine you want a patch, which not only gives you the magic boomerang after collecting the letter, but also… let's say the power bracelet.
The theory is simple: you need to include the code
7E7421-02 for the magic boomerang and 7E742E-01 for the power bracelet, beginning from $CC/426B. The sequence would be just
a9 02 8F 21 74 7E a9 01 8F 2E 74 7E

But… there's a problem! There is no space to include the additional sequence for the power bracelet. If you write over C2 30 6B, the game will crash. So we need a jump to an empty area in the rom where no other code is executed!

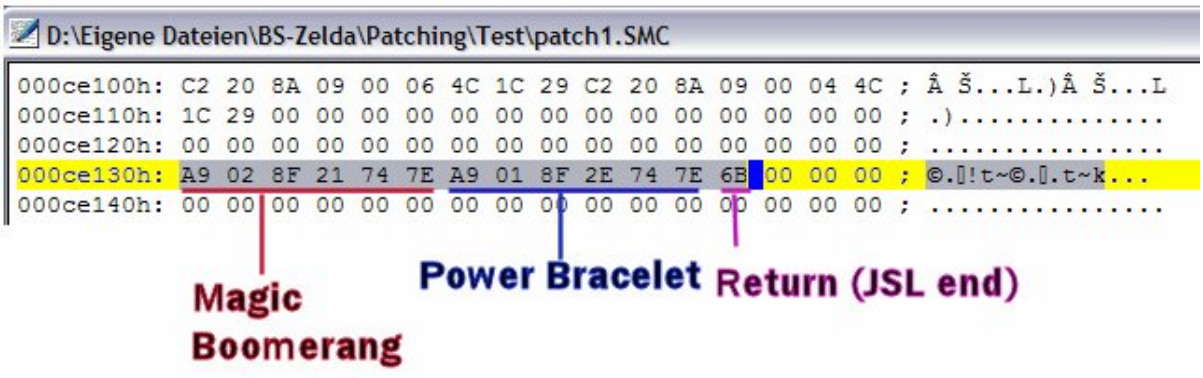
Information: **Jump or Jump to subroutines**
You have 2 options doing this, the JMP (5C) or the JSL (22) opcode. At this stage it doesn't matter which one you use.
If you use a normal Jump you must include a jump back to the address you started the jump.
If you use a JSL, you simply return to the position you jumped from by a 6B. Means, using a jump you can afterwards jump back to another address, while with a jsl you return to from where you started.

Let's check this for JSL out!
1. Scroll down to a position in the rom where you see many "00" - this is ~mostly~ an unused region, but be careful - this is not always the case. I found an obviously unused region at address $CC/E130.
2. Let's hack a JSL to this point. The sequence for the magic boomerang (a9028f21747e) will be written to a JSL linking to CC/E130



The addressing must be written reversed as mentioned already in vol.1! Means address CC/E130 will be 30 E1 CC in the code. Than you have 2 bytes left which must be nopped out, because they are not needed. That is rather important. Nop them out using the EA opcode (no operation).

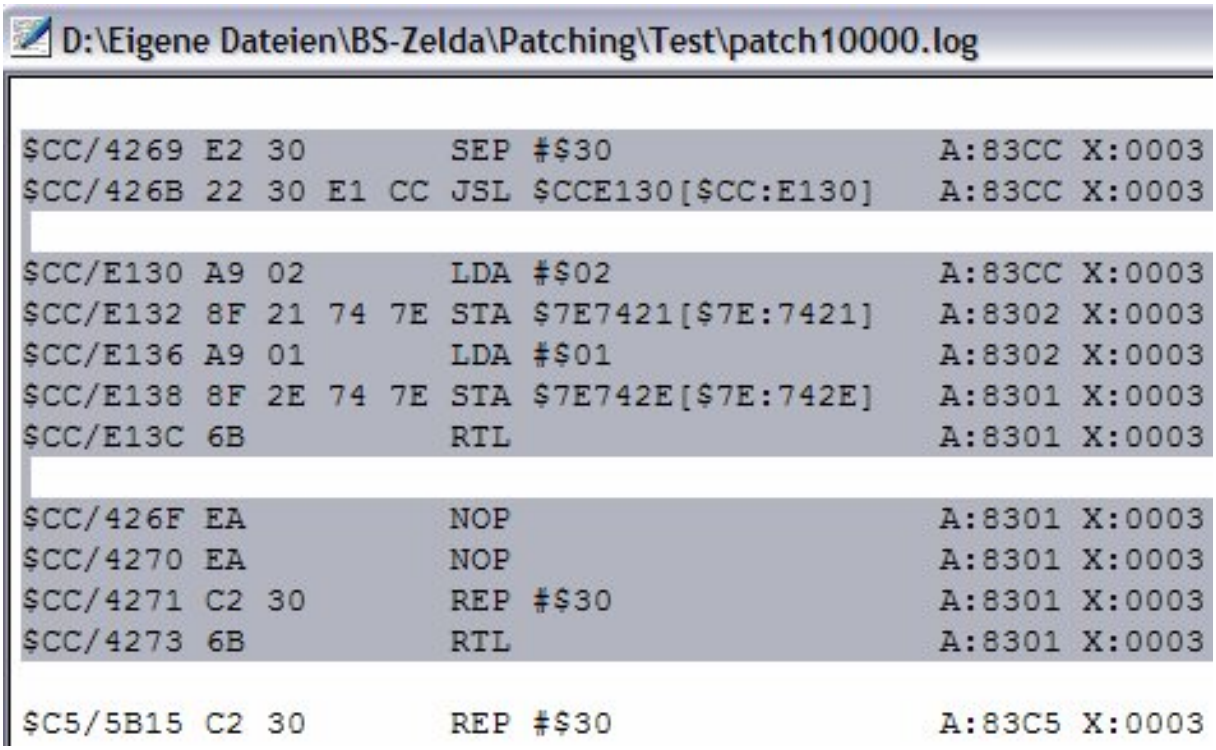
Now you only have to insert the code for the magic boomerang and power bracelet to $CC/E130:



Note: If you used a jump command at $CC/426B instead of a jsl (means 5C 30 E1 CC instead of 22 30 E1 CC); you'd make a jump back at $CC/E13C.
Means instead of a 6B you'll have to write 5C 6F 42 CC. In this case you'd land at the first ea.

Information: **Long or short jumps**
You have noticed that we remain in the same bank, **$CC.** Long Jumps and Jsl's are actually only necessary if you jump into other banks, than that you are at the moment (e.g. $C1, CD, etc.). Since we made a jump in the same bank ($CC) you can also make a short jump or jsr. This saves one byte! *In any case, also long jumps work!*
Short JSR: write 20 instead of 22 and leave the CC away. Means you'd write 20 30 E1 instead of 22 30 E1 CC. At $CC/E13C you'd have to end with a 60 instead of a 6B than!!!!
Short Jump: write 4C instead of 5C and leave the CC away. Means you'd write 4C 30 E1 instead of 5C 30 E1 CC. At $CC/E13C you'd again have to make a jump back

Now, it is time to test your hack word. Trace it with Geiger. If you did everything as described above, you will get the following trace log:



So, now you have the magic boomerang, as well as the power bracelet in your menu!

You can now rename the rom, include the header again, and make your second patch!

IV. Branches

The last important chapter for your hacking abilities are branches. When is a branch needed? Whenever there's a decision to make. Imagine you want to buy anything in the job. You walk into an item and the game checks for your rupees. If you have enough, the subroutines for the item are executed, (rupee loss, item in your menu, etc…). If you don't have enough these subroutines will be branched over.
The same is e.g. for the final boss battle. I used a code to check for the complete Triforce, if yes, open the door. If you don't have enough Triforce pieces, the door remains closed.
You see, branches are essential for any qualified rom hacking.

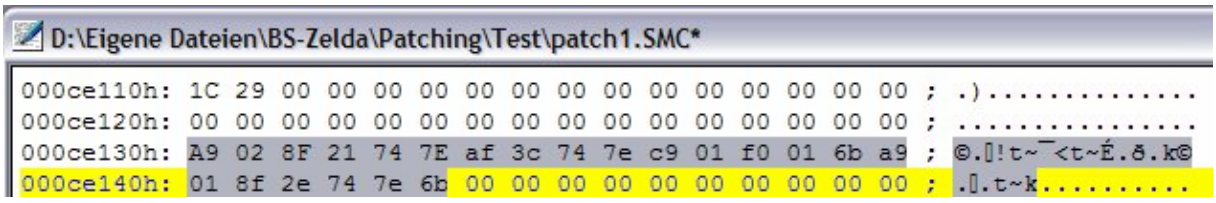
Let's use our patch2.smc rom. You want to make a patch that you obtain the magic boomerang when you collect the letter, but you want also that you only get the power bracelet if you have the bow from Dungeon 1.

Now I have to give you some more opcodes first

- AF LDA: load value from *Ram address* (remember A9? This LDA opcode loads a spec. value into the Accumulator
- C9 CMP: Compare the value in the accumulator with another value
- F0 BEQ: Branch if equal
- D0 BNE: Branch if not equal

Let's start. The bow PAR cheat is 7E743C-01

Now it is just logic – add the following sequence to $CC/E130
A9 02 Load 02 into Accumulator
8F 21 74 7E Store value from Accumulator into the ram address 7E7421 (Magic Boomerang)
AF 3C 74 7E Load value from ram address 7E743C into accumulator (Bow)
C9 01 Compare accumulator value with value 01 (01=bow; 00=no bow)
F0 01 If equal (comparison) branch over 1 byte (the next)
6B Return
a9 01 Load 01 into Accumulator
8F 2E 74 7E Store 01 to this ram address (obtain the Power Bracelet)
6B Return



You see, the code after the first 6B is only executed if you have the bow. Means, without a bow you will not receive the power bracelet!

Note: For practicing here's the code for branch if not equal:
A9 02 Load 02 into Accumulator
8F 21 74 7E Store value from Accumulator into the ram address 7E7421 (Magic Boomerang)
AF 3C 74 7E Load value from ram address 7E743C into accumulator (Bow)
C9 01 Compare accumulator value with value 01 (01=bow; 00=no bow)
D0 06 If **not** equal (comparison) branch over the next 6 bytes
a9 01 Load 01 into Accumulator
8F 2E 74 7E Store 01 to this ram address (obtain the Power Bracelet)
6B Return

If you have not the bow, the sequence for the power bracelet will be branched over.

Now, again, include the header, save the rom as patch3.smc and create your third patch!

VI. Final Word

This was a very short hacking guide for those who begin at point zero. I hope you got a little bit interested in hacking business ☺
However, this is of course only the surface of the possibilities you have. You can nearly do anything with some hacking skills to a rom. There are many more things to know, namely with X and Y registers, stack pointers, DMA transfer, Sound… also take care of E2 and C2 opcodes. These opcodes tell the CPU how many bytes shall be read. E.g. if you make a comparison (C9) the emulator needs to know with how many bytes the comparison shall be made. For example: you have in the Accumulator the value A:3F01. Now you can make a C9 01 or a C9 01 3F – means compare either 1 or 2 bytes.
If you have given the wrong E2 or C2 values, there will occur a frameshift and the game will crash! With this thing I experienced many problems, and had to trace the failure all the time with Geiger.

In any case: Never give up. These things are easier than you think ☺

Appendix I - Important links

http://nesdev.parodius.com/6502guid.txt
Assembly in one step - Beginner guide

http://www.romhacking.net/docs/117/
Romhacking doc, advance.

Appendix II - Opcode list

Opcode	Mnemonic	Addressing Mode	Bytes	Cycles| Reference
00	BRK	Stack/Interrupt	2**	7 | 9
01	ORA	DP Indexed Indirect,X	2	6 | 1,2
02	COP	Stack/Interrupt	2**	7 | 9
03	ORA	Stack Relative	2	4 | 1
04	TSB	Direct Page	2	5 | 2,5
05	ORA	Direct Page	2	3 | 1,2
06	ASL	Direct Page	2	5 | 2,5
07	ORA	Direct Page Indirect Long	2	6 | 1,2
08	PHP	Stack (Push)	1	3 |
09	ORA	Immediate	2*	2 | 1
0A	ASL	Accumulator	1	2 |
0B	PHD	Stack (Push)	1	4 |
0C	TSB	Absolute	3	6 | 5
0D	ORA	Absolute	3	4 | 1
0E	ASL	Absolute	3	6 | 5
0F	ORA	Absolute Long	4	5 | 1
10	BPL	Program Counter Relative	2	2 | 7,8
11	ORA	DP Indirect Indexed,Y	2	5 | 1,2,3
12	ORA	Direct Page Indirect	2	5 | 1,2
13	ORA	SR Indirect Indexed,Y	2	7 | 1
14	TRB	Direct Page	2	5 | 2,5
15	ORA	Direct Page Indexed,X	2	4 | 1,2
16	ASL	Direct Page Indexed,X	2	6 | 2,5
17	ORA	DP Indirect Long Indexed,Y	2	6 | 1,2
18	CLC	Implied	1	2 |
19	ORA	Absolute Indexed,Y	3	4 | 1,3
1A	INC	Accumulator (INA)	1	2 |
1B	TCS	Implied	1	2 |
1C	TRB	Absolute	3	6 | 5
1D	ORA	Absolute Indexed,X	3	4 | 1,3
1E	ASL	Absolute Indexed,X	3	7 | 5,6
1F	ORA	Absolute Long Indexed,X	4	5 | 1
20	JSR	Absolute	3	6 |
21	AND	DP Indexed Indirect,X	2	6 | 1,2
22	JSR	Absolute Long	4	8 |
23	AND	Stack Relative	2	4 | 1
24	BIT	Direct Page	2	3 | 1,2
25	AND	Direct Page	2	3 | 1,2
26	ROL	Direct Page	2	5 | 2,5
27	AND	Direct Page Indirect Long	2	6 | 1,2

28	PLP	Stack (Pull)	1	4 |
29	AND	Immediate	2*	2 | 1
2A	ROL	Accumulator	1	2 |
2B	PLD	Stack (Pull)	1	5 |
2C	BIT	Absolute	3	4 | 1
2D	AND	Absolute	3	4 | 1
2E	ROL	Absolute	3	6 | 5
2F	AND	Absolute Long	4	5 | 1
30	BMI	Program Counter Relative	2	2 | 7,8
31	AND	DP Indirect Indexed,Y	2	5 | 1,2,3
32	AND	Direct Page Indirect	2	5 | 1,1
33	AND	SR Indirect Indexed,Y	2	7 | 1
34	BIT	Direct Page Indexed,X	2	4 | 1,2
35	AND	Direct Page Indexed,X	2	4 | 1,2
36	ROL	Direct Page Indexed,X	2	6 | 2,5
37	AND	DP Indirect Long Indexed,Y	2	6 | 1,2
38	SEC	Implied	1	2 |
39	AND	Absolute Indexed,Y	3	4 | 1,3
3A	DEC	Accumulator	1	2 |
3B	TSC	Implied	1	2 |
3C	BIT	Absolute Indexed,X	3	4 | 1,3
3D	AND	Absolute Indexed,X	3	4 | 1,3
3E	ROL	Absolute Indexed,X	3	7 | 5,6
3F	AND	Absolute Long Indexed,X	4	5 | 1
40	RTI	Stack/RTI	1	6 | 9
41	EOR	DP Indexed Indirect,X	2	6 | 1,2
42	WDM		2	16 |
43	EOR	Stack Relative	2	4 | 1
44	MVP	Block Move	3	| 13
45	EOR	Direct Page	2	3 | 1,2
46	LSR	Direct Page	2	5 | 2,5
47	EOR	Direct Page Indirect Long	2	6 | 1,2
48	PHA	Stack (Push)	1	3 | 1
49	EOR	Immediate	2*	2 | 1
4A	LSR	Accumulator	1	2 |
4B	PHK	Stack (Push)	1	3 |
4C	JMP	Absolute	3	3 |
4D	EOR	Absolute	3	4 | 1
4E	LSR	Absolute	3	6 | 5
4F	EOR	Absolute Long	4	5 | 1
50	BVC	Program Counter Relative	2	2 | 7,8
51	EOR	DP Indirect Indexed,Y	2	5 | 1,2,3
52	EOR	Direct Page Indirect	2	5 | 1,2
53	EOR	SR Indirect Indexed,Y	2	7 | 1
54	MVN	Block Move	3	| 13
55	EOR	Direct Page Indexed,X	2	4 | 1,2
56	LSR	Direct Page Indexed,X	2	6 | 2,5
57	EOR	DP Indirect Long Indexed,Y	2	6 | 1,2
58	CLI	Implied	1	2 |
59	EOR	Absolute Indexed,Y	3	4 | 1,3
5A	PHY	Stack (Push)	1	3 | 10

5B	TCD	Implied	1	2 |
5C	JMP	Absolute Long	4	4 |
5D	EOR	Absolute Indexed,X	3	4 | 1,3
5E	LSR	Absolute Indexed,X	3	7 | 5,6
5F	EOR	Absolute Long Indexed,X	4	5 | 1
60	RTS	Stack (RTS)	1	6 |
61	ADC	DP Indexed Indirect,X	2	6 | 1,2,4
62	PER	Stack (PC Relative Long)	3	6 |
63	ADC	Stack Relative	2	4 | 1,4
64	STZ	Direct Page	2	3 | 1,2
65	ADC	Direct Page	2	3 | 1,2,4
66	ROR	Direct Page	2	5 | 1
67	ADC	Direct Page Indirect Long	2	6 | 1,4
68	PLA	Stack (Pull)	1	4 | 1
69	ADC	Immediate	2*	2 | 1,4
6A	ROR	Accumulator	1	2 |
6B	RTL	Stack (RTL)	1	6 |
6C	JMP	Absolute Indirect	3	5 | 11,12
6D	ADC	Absolute	3	4 | 1,4
6E	ROR	Absolute	3	6 | 5
6F	ADC	Absolute Long	4	5 | 1,4
70	BVS	Program Counter Relative	2	2 | 7,8
71	ADC	DP Indirect Indexed,Y	2	5 | 1,2,3,4
72	ADC	Direct Page Indirect	2	5 | 1,2,4
73	ADC	SR Indirect Indexed,Y	2	7 | 1,4
74	STZ	Direct Page Indexed,X	2	4 | 1,2
75	ADC	Direct Page Indexed,X	2	4 | 1,2,4
76	ROR	Direct Page Indexed,X	2	6 | 2,5
77	ADC	DP Indirect Long Indexed,Y	2	6 | 1,2,4
78	SEI	Implied	1	2 |
79	ADC	Absolute Indexed,Y	3	4 | 1,3,4
7A	PLY	Stack (Pull)	1	4 | 10
7B	TDC	Implied	1	2 |
7C	JMP	Absolute Indexed Indirect	3	6 |
7D	ADC	Absolute Indexed,X	3	4 | 1,3,4
7E	ROR	Absolute Indexed,X	3	7 | 5,6
7F	ADC	Absolute Long Indexed,X	4	5 | 1,4
80	BRA	Program Counter Relative	2	3 | 8
81	STA	DP Indexed Indirect,X	2	6 | 1,2
82	BRL	Program Counter Relative Long	3	4 |
83	STA	Stack Relative	2	4 | 1
84	STY	Direct Page	2	3 | 2,10
85	STA	Direct Page	2	3 | 1,2
86	STX	Direct Page	2	3 | 2,10
87	STA	Direct Page Indirect Long	2	6 | 1,2
88	DEY	Implied	1	2 |
89	BIT	Immediate	2*	2 | 1
8A	TXA	Implied	1	2 |
8B	PHB	Stack (Push)	1	3 |
8C	STY	Absolute	3	4 | 10
8D	STA	Absolute	3	4 | 1

8E	STX	Absolute	3	4 | 10
8F	STA	Absolute Long	4	5 | 1
90	BCC	Program Counter Relative	2	2 | 7,8
91	STA	DP Indirect Indexed,Y	2	6 | 1,2
92	STA	Direct Page Indirect	2	5 | 1,2
93	STA	SR Indirect Indexed,Y	2	7 | 1
94	STY	Direct Page Indexed,X	2	4 | 2,10
95	STA	Direct Page Indexed,X	2	4 | 1,2
96	STX	Direct Page Indexed,Y	2	4 | 2,10
97	STA	DP Indirect Long Indexed,Y	2	6 | 1,2
98	TYA	Implied	1	2 |
99	STA	Absolute Indexed,Y	3	5 | 1
9A	TXS	Implied	1	2 |
9B	TXY	Implied	1	2 |
9C	STZ	Absolute	3	4 | 1
9D	STA	Absolute Indexed,X	3	5 | 1
9E	STZ	Absolute Indexed,X	3	5 | 1
9F	STA	Absolute Long Indexed,X	4	5 | 1
A0	LDY	Immediate	2+	2 | 10
A1	LDA	DP Indexed Indirect,X	2	6 | 1,2
A2	LDX	Immediate	2+	2 | 10
A3	LDA	Stack Relative	2	4 | 1
A4	LDY	Direct Page	2	3 | 2,10
A5	LDA	Direct Page	2	3 | 1,2
A6	LDX	Direct Page	2	3 | 2,10
A7	LDA	Direct Page Indirect Long	2	6 | 1,2
A8	TAY	Implied	1	2 |
A9	LDA	Immediate	2*	2 | 1
AA	TAX	Implied	1	2 |
AB	PLB	Stack (Pull)	1	4 |
AC	LDY	Absolute	3	4 | 10
AD	LDA	Absolute	3	4 | 1
AE	LDX	Absolute	3	4 | 10
AF	LDA	Absolute Long	4	5 | 1
B0	BCS	Program Counter Relative	2	2 | 7,8
B1	LDA	DP Indirect Indexed,Y	2	5 | 1,2,3
B2	LDA	Direct Page Indirect	2	5 | 1,2
B3	LDA	SR Indirect Indexed,Y	2	7 | 1
B4	LDY	Direct Page Indexed,X	2	4 | 2,10
B5	LDA	Direct Page Indexed,X	2	4 | 1,2
B6	LDX	DP Indexed,Y	2	4 | 2,10
B7	LDA	DP Indirect Long Indexed,Y	2	6 | 1,2
B8	CLV	Implied	1	2 |
B9	LDA	Absolute Indexed,Y	3	4 | 1,3
BA	TSX	Implied	1	2 |
BB	TYX	Implied	1	2 |
BC	LDY	Absolute Indexed,X	3	4 | 3,10
BD	LDA	Absolute Indexed,X	3	4 | 1,3
BE	LDX	Absolute Indexed,Y	3	4 | 3,10
BF	LDA	Absolute Long Indexed,X	4	5 | 1
C0	CPY	Immediate	2+	2 | 10

C1	CMP	DP Indexed Indirect,X	2	6 | 1,2
C2	REP	Immediate	2	3 |
C3	CMP	Stack Relative	2	4 | 1
C4	CPY	Direct Page	2	3 | 2,10
C5	CMP	Direct Page	2	3 | 1,2
C6	DEC	Direct Page	2	5 | 2,5
C7	CMP	Direct Page Indirect Long	2	6 | 1,2
C8	INY	Implied	1	2 |
C9	CMP	Immediate	2*	2 | 1
CA	DEX	Implied	1	2 |
CB	WAI	Implied	1	3 | 15
CC	CPY	Absolute	3	4 | 10
CD	CMP	Absolute	3	4 | 1
CE	DEC	Absolute	3	6 | 5
CF	CMP	Absolute Long	4	5 | 1
D0	BNE	Program Counter Relative	2	2 | 7,8
D1	CMP	DP Indirect Indexed,Y	2	5 | 1,2,3
D2	CMP	Direct Page Indirect	2	5 | 1,2
D3	CMP	SR Indirect Indexed,Y	2	7 | 1
D4	PEI	Stack (Direct Page Indirect)	2	6 | 2
D5	CMP	Direct Page Indexed,X	2	4 | 1,2
D6	DEC	Direct Page Indexed,X	2	6 | 2,5
D7	CMP	DP Indirect Long Indexed,Y	2	6 | 1,2
D8	CLD	Implied	1	2 |
D9	CMP	Absolute Indexed,Y	3	4 | 1,3
DA	PHX	Stack (Push)	1	3 | 10
DB	STP	Implied	1	3 | 14
DC	JMP	Absolute Indirect Long	3	6 |
DD	CMP	Absolute Indexed,X	3	4 | 1,3
DE	DEC	Absolute Indexed,X	3	7 | 5,6
DF	CMP	Absolute Long Indexed,X	4	5 | 1
E0	CPX	Immediate	2+	2 | 10
E1	SBC	DP Indexed Indirect,X	2	6 | 1,2,4
E2	SEP	Immediate	2	3 |
E3	SBC	Stack Relative	2	4 | 1,4
E4	CPX	Direct Page	2	3 | 2,10
E5	SBC	Direct Page	2	3 | 1,2,4
E6	INC	Direct Page	2	5 |
E7	SBC	Direct Page Indirect Long	2	6 | 1,2,4
E8	INX	Implied	1	2 |
E9	SBC	Immediate	2*	2 |
EA	NOP	Implied	1	2 |
EB	XBA	Implied	1	3 |
EC	CPX	Absolute	3	4 | 10
ED	SBC	Absolute	3	4 | 1,4
EE	INC	Absolute	3	6 | 5
EF	SBC	Absolute Long	4	5 | 1,4
F0	BEQ	Program Counter Relative	2	2 | 7,8
F1	SBC	DP Indirect Indexed,Y	2	5 | 1,2,3,4
F2	SBC	Direct Page Indirect	2	5 | 1,2,4
F3	SBC	SR Indirect Indexed,Y	2	7 | 1,4

F4	PEA	Stack (Absolute)	3	5 |
F5	SBC	Direct Page Indexed,X	2	4 | 1,2,4
F6	INC	Direct Page Indexed,X	2	6 | 2,5
F7	SBC	DP Indirect Long Indexed,Y	2	6 | 1,2,4
F8	SED	Implied	1	2 |
F9	SBC	Absolute Indexed,Y	3	4 | 1,3,4
FA	PLX	Stack (Pull)	1	4 | 10
FB	XCE	Implied	1	2 |
FC	JSR	Absolute Indexed Indirect	3	8 |
FD	SBC	Absolute Indexed,X	3	4 | 1,3,4
FE	INC	Absolute Indexed,X	3	7 | 5,6
FF	SBC	Absolute Long Indexed,X	4	5 | 1,4

==========