

[b]Immediate:[/b]

ADC.B Acc,#imm

[b]zero page:[/b]

LDB tmp,zp

ADC.B Acc,tmp

[b]zp[/b]

LDB tmp,zp

ADC.B Acc,tmp

[b]zp,x[/b]

LDB tmp,zp,x

ADC.B Acc,tmp

[b]zp,y[/b]

LDB tmp,zp,y

ADC.B Acc,tmp

[b] (zp,x)[/b]

LDW tmp,zp,x

LDB tmp,[tmp]

ADC Acc,tmp

[b](zp),y[/b]

LDW tmp,zp

ADD tmp,y

LDB tmp,[tmp]

ADC.B Acc,tmp

[b]abs[/b]

LDB tmp,abs

ADC Acc,tmp

[b]abs,x[/b]

```

LDB tmp,abs,x
ADC Acc,tmp
[b]abs,y[/b]
LDB tmp,abs,y
ADC Acc,tmp

LDB tmp,zp
LDB tmp,zp,x
LDB tmp,zp,y
LDB tmp,[tmp]
LDB tmp,abs
LDB tmp,abs,x
LDB tmp,abs,y
LDB tmp,sp          ; for stack pulls
LDW tmp,zp
LDW tmp,zp,x
LDW tmp,[tmp]       ; for JMP($abs)
MOV.B Acc,tmp
ADD.W tmp,x,#zp
ADD.W tmp,y

ADC.B Acc,#imm
ADC.B Acc,tmp

```

I'm toying with the idea of a superscalar 6502. It would work by changing 6502 opcodes into micro-ops in a manner similar to what's done for the x86. So, I need to have worked out an appropriate set of micro-ops. The micro-ops would be a load / store architecture with a fixed 13-bit instruction format. I think all instructions can be implemented with a maximum of four micro-ops. This means a table of 54 bits for each instruction (2 bits used to indicate # of micro-ops). The table would be indexed by the opcode byte and a field (Ld3) in the micro-op instruction indicates when to take values from the macro-op instruction. Necessary for constants supplied by macro-ops.

Opcode <sub>6</sub>	Ld <sub>4</sub>	Rd <sub>3</sub>	Rn <sub>3</sub>		Ld <sub>4</sub>	Meaning
NOP	~	~	~			
LDB	Ld <sub>3</sub>	Rd <sub>3</sub>	Rn <sub>2</sub>		0	no constant
LDW	Ld <sub>3</sub>	Rd <sub>3</sub>			1	the value 1
STB	Ld <sub>3</sub>	Rs <sub>3</sub>			2	the value 2
ADD.W	Ld <sub>3</sub>	Rd <sub>3</sub>			3	the value3
ADD.B	Ld <sub>3</sub>	Rd <sub>3</sub>			4	
ADC.B	Ld <sub>3</sub>	Rd <sub>3</sub>			5	
SBC.B	Ld <sub>3</sub>	Rd <sub>3</sub>			6	
AND.B	Ld <sub>3</sub>	Rd <sub>3</sub>			7	the value \$100
OR.B	Ld <sub>3</sub>	Rd <sub>3</sub>			8	reference 8 bits from macro op (byte 2)
EOR.B	Ld <sub>3</sub>	Rd <sub>3</sub>			9	reference 16 bits from macro op (bytes 2,3)
CMP.B	Ld <sub>3</sub>	Rd <sub>3</sub>			10	
ADC.B tmp	Ld <sub>3</sub>	Rd <sub>3</sub>			11	
SBC.B tmp	Ld <sub>3</sub>	Rd <sub>3</sub>			12	
AND.B tmp	Ld <sub>3</sub>	Rd <sub>3</sub>			13	the value -3
OR.B tmp	Ld <sub>3</sub>	Rd <sub>3</sub>			14	the value -2
EOR.B tmp	Ld <sub>3</sub>	Rd <sub>3</sub>			15	the value -1
CMP.B tmp	Ld <sub>3</sub>	Rd <sub>3</sub>	Rn <sub>3</sub>			
BEQ	8	~				
BNE	8	~				
BCS	8	~				
BCC	8	~				
BVS	8	~				
BVC	8	~				
BMI	8	~				
BPL	8	~				
CLC	~	~				
SEC	~	~				
CLV	~	~				
SEI	~	~				
CLI	~	~				
JMP	~	Rd <sub>3</sub>				
LDIB	Ld <sub>4</sub>	Rd <sub>3</sub>	0			

Rd <sub>3</sub>	Register
0	Acc
1	x
2	y
3	sp
4	pc
5	tmp

Rn <sub>3</sub>	Index Register
0	z (none)
1	x
2	y
3	sp
4	-1
5	tmp

6	pc+2
7	sr


Some sample instruction breakdowns:

[code]

[b]pha[/b]

SB acc,sp

ADD.B sp,#-1

[b]adc (zp),y[/b]

LDW tmp,zp

ADD tmp,y

LDB tmp,[tmp]

ADC.B Acc,tmp

[b]rti[/b]

ADD.B sp,#3

LDB sr,-2[sp]

LDW tmp,-1[sp]

JMP 0[tmp]

[b]rts[/b]

ADD.B sp,#2

LDW tmp,-1[sp]

JMP 1[tmp]

[/code]

## Micro-ops Instruction Set Description

**Src1** is a constant value or a reference to constant value defined in the macro-instruction. The four-bit **Ld<sub>4</sub>** field specifies **src1**.

**dst** is the target register and the first source operand register. Some micro-op instruction such as **CMP** don't update the target. The **Rd<sub>3</sub>** field of the micro-op specifies the target register.

**src2** is the second source operand register, which is the indexing register for memory operations. The special value representing the value minus one may also be used.

### ADCB **src1**,**dst**,**src2**

**Description:** Perform a byte addition operation including the carry flag, storing the result in **dst**. Usually either **src1** or **src2** is the value 0.

**Operation:**  $dst = dst + src1 + src2 + \text{carry flag}$

### ADDB **src1**,**dst**,**src2**

**Description:** Perform a byte addition operation, storing the result in **dst**. Usually either **src1** or **src2** is the value 0.

**Operation:**  $dst = dst + src1 + src2$

### ANDB **src1**,**dst**,**src2**

**Description:** Perform a bitwise 'and' operation, storing the result in **dst**. Usually either **src1** or **src2** is the value -1.

**Operation:**  $dst = dst \& src1 \& src2$

### BCC **src1**

**Description:** Branch if the carry flag is clear. This instruction modifies the program counter to the target address. **src1** should be a reference to an eight-bit constant in the instruction. The eight-bit constant plus two (the length of a branch instruction) will be added to the program counter value of the current instruction.

Branches are predicted and taken in the fetch stage of the processor. The branch micro-op verifies that the branch was predicted correctly. If an incorrect prediction occurred the processor pipeline is flushed of following instructions and the branch operation performed (taken or not taken). Otherwise the branch operation is not performed, and the micro-op is treated like a NOP instruction.

**Operation:** if (!c)  $pc = pc + src1 + 2$

### BCS src1

**Description:** Branch if the carry flag is set. This instruction modifies the program counter to the target address. src1 should be a reference to an eight-bit constant in the instruction. The eight-bit constant plus two (the length of a branch instruction) will be added to the program counter value of the current instruction.

Branches are predicted and taken in the fetch stage of the processor. The branch micro-op verifies that the branch was predicted correctly. If an incorrect prediction occurred the processor pipeline is flushed of following instructions and the branch operation performed (taken or not taken). Otherwise the branch operation is not performed, and the micro-op is treated like a NOP instruction.

**Operation:** if (lc)  $pc = pc + src1 + 2$

### BEQ src1

**Description:** Branch if the zero flag is set. This instruction modifies the program counter to the target address. src1 should be a reference to an eight-bit constant in the instruction. The eight-bit constant plus two (the length of a branch instruction) will be added to the program counter value of the current instruction.

Branches are predicted and taken in the fetch stage of the processor. The branch micro-op verifies that the branch was predicted correctly. If an incorrect prediction occurred the processor pipeline is flushed of following instructions and the branch operation performed (taken or not taken). Otherwise the branch operation is not performed, and the micro-op is treated like a NOP instruction.

**Operation:** if (z)  $pc = pc + src1 + 2$

### BMI src1

**Description:** Branch if the negative flag is set. This instruction modifies the program counter to the target address. src1 should be a reference to an eight-bit constant in the instruction. The eight-bit constant plus two (the length of a branch instruction) will be added to the program counter value of the current instruction.

Branches are predicted and taken in the fetch stage of the processor. The branch micro-op verifies that the branch was predicted correctly. If an incorrect prediction occurred the processor pipeline is flushed of following instructions and the branch operation performed (taken or not taken). Otherwise the branch operation is not performed, and the micro-op is treated like a NOP instruction.

**Operation:** if (n)  $pc = pc + src1 + 2$

### BNE src1

**Description:** Branch if the zero flag is clear. This instruction modifies the program counter to the target address. src1 should be a reference to an eight-bit constant in the instruction. The eight-bit constant plus two (the length of a branch instruction) will be added to the program counter value of the current instruction.

Branches are predicted and taken in the fetch stage of the processor. The branch micro-op verifies that the branch was predicted correctly. If an incorrect prediction occurred the processor pipeline is flushed of following instructions and the branch operation performed (taken or not taken). Otherwise the branch operation is not performed, and the micro-op is treated like a NOP instruction.

**Operation:**  $pc = pc + src1 + 2$

### BPL src1

**Description:** Branch if the negative flag is clear. This instruction modifies the program counter to the target address. src1 should be a reference to an eight-bit constant in the instruction. The eight-bit constant plus two (the length of a branch instruction) will be added to the program counter value of the current instruction.

Branches are predicted and taken in the fetch stage of the processor. The branch micro-op verifies that the branch was predicted correctly. If an incorrect prediction occurred the processor pipeline is flushed of following instructions and the branch operation performed (taken or not taken). Otherwise the branch operation is not performed, and the micro-op is treated like a NOP instruction.

**Operation:** if (!n)  $pc = pc + src1 + 2$

### BVC src1

**Description:** Branch if the overflow flag is clear. This instruction modifies the program counter to the target address. src1 should be a reference to an eight-bit constant in the instruction. The eight-bit constant plus two (the length of a branch instruction) will be added to the program counter value of the current instruction.

Branches are predicted and taken in the fetch stage of the processor. The branch micro-op verifies that the branch was predicted correctly. If an incorrect prediction occurred the processor pipeline is flushed of following instructions and the branch operation performed (taken or not taken). Otherwise the branch operation is not performed, and the micro-op is treated like a NOP instruction.

**Operation:** if (!v)  $pc = pc + src1 + 2$

### BVS src1

**Description:** Branch if the overflow flag is set. This instruction modifies the program counter to the target address. src1 should be a reference to an eight-bit constant in the instruction. The eight-bit constant plus two (the length of a branch instruction) will be added to the program counter value of the current instruction.

Branches are predicted and taken in the fetch stage of the processor. The branch micro-op verifies that the branch was predicted correctly. If an incorrect prediction occurred the processor pipeline is flushed of following instructions and the branch operation performed (taken or not taken). Otherwise the branch operation is not performed, and the micro-op is treated like a NOP instruction.

**Operation:** if (v)  $pc = pc + src1 + 2$

### CLC

**Description:** clear the carry flag. src1, dst, and src2 fields of the instruction are ignored. The micro-op table must indicate that the CLC micro-op updates the carry flag. Otherwise this instruction will be treated as a NOP.

**Operation:**  $c = 0$

### CLV

**Description:** clear the overflow flag. src1, dst, and src2 fields of the instruction are ignored. The micro-op table must indicate that the CLV micro-op updates the overflow flag. Otherwise this instruction will be treated as a NOP.

**Operation:**  $v = 0$

### CMPB src1,dst,src2

**Description:** Perform a comparison operation. Usually either src1 or src2 is the value 0. CMP does not update the target register. Instead the flag results may be updated for the macro instruction. The appropriate micro-op compare instruction must be marked as updating a subset of the macro flags register; otherwise this operation will be treated as a NOP.

**Operation:**  $dst = src1 - src2$

### EORB src1,dst,src2

**Description:** Perform a bitwise 'exclusive or' operation, storing the result in dst. Usually either src1 or src2 is the value 0.

**Operation:**  $dst = dst \wedge src1 \wedge src2$



### JMP src1,dst,src2

**Description:** Jump to the address specified as the sum of src1 and src2. Normally src1 specifies a reference to a 16-bit constant supplied by the macro instruction. src2 is an indexing register. The dst field is not used and should specify zero.

Jumps are predicted in the fetch stage of the processor, so this micro-op verifies that the correct prediction was made, correcting the flow path if the prediction was incorrect. If the prediction was incorrect then the processor pipeline is flushed of following instructions, and instructions from the correct path begin fetching.

**Operation:**  $pc = src1 + src2$

### LDB src1,dst,src2

**Description:** Perform a memory byte load operation. Src1 determines an address constant. Src2 is an indexing register.

**Operation:**  $dst = Memory_8[src1+src2]$

### LDW src1,dst,src2

**Description:** Perform a memory word load operation. Src1 determines an address constant. Src2 is an indexing register.

**Operation:**  $dst = Memory_{16}[src1+src2]$

## NOP

**Description:** Perform nothing. NOP's are used to pad the right-hand side of the micro-op table. The src1, dst, and src2 fields of the instruction are ignored.

**Operation:** none

## ORB src1,dst,src2

**Description:** Perform a bitwise 'or' operation, storing the result in dst. Usually either src1 or src2 is the value 0.

**Operation:**  $\text{dst} = \text{dst} \mid \text{src1} \mid \text{src2}$

## SEC

**Description:** set the carry flag. src1, dst, and src2 fields of the instruction are ignored.

**Operation:**  $c = 1$

## STB src1,dst,src2

**Description:** Perform a memory byte store operation. Src1 determines an address constant. Src2 is an indexing register. dst specifies a register containing the value to be stored.

**Operation:**  $\text{Memory}_8[\text{src1} + \text{src2}] = \text{dst}$

## STW src1,dst,src2

**Description:** Perform a memory word store operation. Src1 determines an address constant. Src2 is an indexing register. dst specifies a register containing the value to be stored.

**Operation:**  $\text{Memory}_{16}[\text{src1} + \text{src2}] = \text{dst}$