

# rtf65004

© 2019 Robert Finch

## Overview

The rtf65004 is a superscalar processor capable of executing the 6502 instruction set. 6502 instructions are translated into 1 to 4 risc instructions called micro-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>4</sub>	Rd <sub>3</sub>	Rn <sub>3</sub>		0	no constant
LDW	Ld <sub>4</sub>	Rd <sub>3</sub>	Rn <sub>3</sub>		1	the value 1
STB	Ld <sub>4</sub>	Rs <sub>3</sub>	Rn <sub>3</sub>		2	the value 2
STW	Ld <sub>4</sub>	Rs <sub>3</sub>	Rn <sub>3</sub>		3	the value3
ADDW	Ld <sub>4</sub>	Rd <sub>3</sub>	Rn <sub>3</sub>		4	
ADDB	Ld <sub>4</sub>	Rd <sub>3</sub>	Rn <sub>3</sub>		5	
ADCB	Ld <sub>4</sub>	Rd <sub>3</sub>	Rn <sub>3</sub>		6	
SBCB	Ld <sub>4</sub>	Rd <sub>3</sub>	Rn <sub>3</sub>		7	the value \$100
ANDB	Ld <sub>4</sub>	Rd <sub>3</sub>	Rn <sub>3</sub>		8	reference 8 bits from macro op (byte 2)
ORB	Ld <sub>4</sub>	Rd <sub>3</sub>	Rn <sub>3</sub>		9	reference 16 bits from macro op (bytes 2,3)
EORB	Ld <sub>4</sub>	Rd <sub>3</sub>	Rn <sub>3</sub>		10	
CMPB	Ld <sub>4</sub>	Rd <sub>3</sub>	Rn <sub>3</sub>		11	
BEQ	8	~			12	
BNE	8	~			13	the value -3
BCS	8	~			14	the value -2
BCC	8	~			15	the value -1
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

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

4	pc
5	tmp
6	pc+2
7	sr

4	-1 (value)
5	tmp
6	2 (value)

## Micro-ops Bundles

75 74	73 66	65 64	63 48	47 32	31 16	15 0
Count <sub>2</sub>	Flags To Update <sub>8</sub>	Active Flag Instruction <sub>2</sub>	First u-op	Second u-op	Third u-op	Fourth u-op

Micro-ops are bundled together into a group of four plus some book-keeping bits as shown above for each macro-instruction.

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

### ASLB src1,dst,src2

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

**Operation:**  $dst = dst \ll 1$

### 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)  $pc = pc + src1 + 2$

### BITB src1,dst,src2

**Description:** Perform a bitwise 'and' operation, discarding the result. Usually either src1 or src2 is the value -1. BITB does not update the target register. Instead the flag results may be updated for the macro instruction. The appropriate micro-op bit instruction must be marked as updating a subset of the macro flags register; otherwise this operation will be treated as a NOP.

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

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

### LSRB src1,dst,src2

**Description:** Perform a right shift operation, storing the result in dst. A zero is shifted into the most significant bit. Usually either src1 or src2 is the value 0.

**Operation:**  $dst = dst \gg 1$

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

## ROLB src1,dst,src2

**Description:** Perform a left shift operation, storing the result in dst. The carry flag is shifted into the least significant bit. Usually either src1 or src2 is the value 0.

**Operation:**  $\text{dst} = \text{dst} \ll 1 + \text{cf}$

## RORB src1,dst,src2

**Description:** Perform a right shift operation, storing the result in dst. The carry flag is shifted into the most significant bit. Usually either src1 or src2 is the value 0.

**Operation:**  $\text{dst} = \text{cf}, \text{dst} \gg 1$

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

## Select Micro-op sequences

SBC,CMP,AND,ORA,EOR are all similar to ADC, with the appropriate byte operation performed.

### ADC #Imm

```
ADCB  acc,<R8>,0
```

### ADC zp

```
LDB   tmp,<R8>,0
ADCB  acc,#0,tmp
```

### ADC zp,x

```
LDB   tmp,<R8>,x
ADCB  acc,#0,tmp
```

### ADC (zp,x)

```
LDW   tmp,<R8>,x
LDB   tmp,$0000[tmp]
ADCB  acc,#0000,tmp
```

### ADC (zp),y

```
LDW   tmp,<R8>,0
ADDW  tmp,#0,y
LDB   tmp,$0[tmp]
ADCB  acc,#0,tmp
```

### ADC abs

```
LDB   tmp,<R16>,0
ADCB  acc,#0,tmp
```

### ADC abs,x

```
LDB   tmp,<R16>,x
ADCB  acc,#0,tmp
```

### ADC abs,y

```
LDB   tmp,<R16>,y
ADCB  acc,#0,tmp
```

## BRK

```

ADDB  sp,#-3 ; subtract 3 from stack pointer
STB   sr,1[sp] ; store status reg
STW   pc,2[sp] ; store program counter
SEI   ; set interrupt mask
LDW   tmp,$FFFE ; fetch break address
JMP   $0000[tmp] ; jump to break routine

```

## JSR addr

```

ADDB  sp,#-2,0
ADDW  tmp,#2,pc
STW   tmp,1[sp]
JMP   0,<R16>,0

```

## LDA (zp,x)

```

LDW   tmp,<R8>,x
LDB   acc,$0000[tmp]

```

## LDA (zp),y

```

LDW   tmp,<R8>,0
ADDW  tmp,#0,y
LDB   acc,$0[tmp]

```

## PHA

```

ADDB  sp,#-1,0
STB   acc,1[sp]

```

## RTI

```

ADDB  sp,#3 ; adjust stack pointer
LDB   sr,-2[sp] ; load flags
LDW   tmp,-1[sp] ; load return address
JMP   $0000[tmp] ; jump to return address

```

## RTS

```

LDW tmp,1[sp]
ADD.B sp,#2,0
JMP $1[tmp]

```