# AVC3: the bulk copy upgrade

<u>* Shifts left and right have an extra overflow argument, producing the bits needed to OR into subsequent words for shifting multiple words.</u>
<u>* There are now rotate instructions.</u>
<u>* There are now bit test conditional jump instructions.</u>
<u>* There are now decode instructions. 0 -> 1, 1 -> 2, 2 -> 4, 3 -> 8, etc.</u>
* There's now block copy and block fill instructions.
* There are now pre/post inc/dec memory peek register addressing modes
* Library includes heap memory allocation, self-emulator

# New register prefixes

- pipe character: apply logical NOT on reads or writes to register

# Instruction set

## Miscellany

### 0, all other unassigned opcodes below 256: HALT

Stops the virtual computer

### 1: MOVE V R

Move value V into register R.

### 2: COPY R1 R2

Set R2 to the value from R1.

### 3: SWAP R1 R2

Swap the values of R1 and R2.

### 4: JUMP M

Jump to memory address M.

### 5: REFR

Refresh the screen.

### 6: VSYNCI V

Wait V frames.

### 7: VSYNCR R

Wait R frames.

### 8: DEC R

Subtract 1 from R.

### 9: INC R

Add 1 to R.

## File IO

### 10: COMPILE R1 R2 R3 RF

Load and compile a VC9 program with null-terminated name at [R1] into memory starting at [R2] and store location of last word in R3. This instruction loads programs with labels and relocations offset so that they run correctly. RF is set to 1 if the compile is successful and zero if it is not.

### 11: LOAD R1 R2 R3 RF

Same as compiling but for plaintext. RF is set if successful and reset if not.

### 12: DATASAVE R1 R2 R3 RF

Saves as data the memory starting at [R1], with length R2 and null-terminated name at [R3]. RF is set to 1 if the save successfully completes and zero if it does not.

### 13: SAVE R1 R2 R3 RF

Same as datasave but saves as plaintext. RF is set to 1 if the save successfully completes and zero if it does not.

## Load/Store

### 14: LOADR R1 R2

Load from memory address from R1 into R2.

### 15: LOADM M R

Load from memory address M into register R.

### 16: ILOADR V R1 R2

Load from address [V+R1] into R2.

## 17: RLOADR R1 R2 R3

Load from adress [R1+R2] into R3.

## 18: STORER R1 R2

Store R1 in address from R2.

## 19: STOREM R M

Store R in address M.

## 20: ISTORER R1 R2 V

Store R1 in [R2+V].

## 21: RSTORER R1 R2 R3

Store R1 in [R2+R3].

## 90: STOREIR V R

Store V in R

## 91: STOREIM V M

Store V in M

# ALU

## 22: ADDR R1 R2 R3 RF

Add R3 to the result of R2+R1 and set RF to the overflow.

## 23: ADDI V R1 R2 RF

Set R2 to R1+V and set RF to the overflow.

## 24: SUBR R1 R2 R3 RF

Set R3 to the result of R2-R1 and set RF to the overflow.

## 25: SUBI V R1 R2 RF

Set R2 to the result of R1-V and set RF to the underflow.

## 26: MULR R1 R2 R3 R4

Set R3 and R4 to most and least significant bits of R1*R2, respectively.

## 27: MULI V R1 R2 R3
Set R2 and R3 to most and least significant bits of V*R1, respectively.

## 28: DIVR R1 R2 R3 R4
Divide R2 by R1, placing the remainder in R3 and result in R4.

## 29: DIVI V R1 R2 R3
Divide R1 by V, placing the remainder in R2 and result in R3.

## 92: SDIVR R1 R2 R3 R4
Signed divide R2 by R1, placing the remainder in R3 and result in R4.

## 93: SDIVI V R1 R2 R3
Signed divide R1 by V, placing the remainder in R2 and result in R3.

## 30: TWO R1 R2
Set R2 to the two's complement inverse of R1.

## 31: ANDR R1 R2 R3
Set R3 to R2 AND R1.

## 32: ANDI V R1 R2
Set R2 to R1 AND V.

## 33: ORR R1 R2 R3
Set R3 to R2 OR R1.

## 34: ORI V R1 R2
Set R2 to R1 OR V.

## 35: XORR R1 R2 R3
Set R3 to R2 XOR R1.

## 36: XORI V R1 R2
Set R2 to R1 AND V.

## 37: NOT R1 R2
Set R1 to NOT R2.

### 38: LEFTR R1 R2 R3 R4

Set R3 to R2 << R1 and R4 to bits that rolled off the end.

### 39: LEFTI V R1 R2 R3

Set R2 to R1 << V. and R3 to bits that rolled off the end.

### 40: RIGHTR R1 R2 R3 R4

Set R3 to R2 >> R1 and R4 to bits that rolled off the end.

### 41: RIGHTI V R1 R2 R3

Set R2 to R1 >> V to bits that rolled off the end

### 94: ROLR R1 R2 R3

Set R3 to R2 rotated left by R1.

### 95: ROLI V R1 R2

Set R2 to R1 rotated left by R1.

### 96: RORR R1 R2 R3

Set R3 to R2 rotated right by R1.

### 97: RORI V R1 R2

Set R2 to R1 rotated right by R1.

### 100: DECODE R1 R2

Decode R1 into a single bit in R2. For values 16 and above, write zero to R2.

### 42: TRUE R1 R2

Set R2 to 1 if R1 is non-zero, otherwise set R2 to zero.

### 43: SIGN R1 R2

Set R2 to -1 if R1 is negative, otherwise the same as TRUE.

### 44: RND R

Set R to random word.

# Input

## 45: TOUCH R1 R2 R3

R1 = touch x coordinate
R2 = touch y coordinate
R3 = touch time

## 46: CLIPR R1 R2

Get R1st zero-indexed character of the clipboard and store it in R.

## 47: CLIPI V R

Get Vst zero-indexed character of the clipboard and store it in R.

## 48: CLIPL R

Put the total length of the clipboard in R.

## 49: BUTTON R

Get current button presses and write them to R.

## 50: MIL R1 R2

Get current millisecond count, writing most significant bits to R1 and least significant bits to R2.

## 51: TIME R1 R2 R3

Put hours in R1, minutes in R2 and seconds in R3.

## 52: DATE R1 R2 R3

Put year in R1, month in R2 and day in R3.

# Output

## 53: OUT R

Output the contents of R into the output string.

## 54: BEEP R1 R2

Beep with R1 as instrument and R2 as pitch.

## Conditional Jumps

### 55: JER R1 R2 M

Jump to M if R1 = R2.

### 56: JEI V R M

Jump to M if V = R.

### 57: JNR R1 R2 M

Jump to M if R1 != R2.

### 58: JNI V R M

Jump to M if V != R.

### 59: JGR+ R1 R2 M

Jump to M if R1 > R2.

### 60: JGR- R1 R2 M

Jump to M if R1 > R2, using signed 2's complement arithmetic.

### 61: JGI+ V R M

Jump to M if V > R.

### 62: JGI- V R M

Jump to M if V > R, using signed 2's complement arithmetic.

### 63: JLR+ R1 R2 M

Jump to M if R1 < R2.

### 64: JLR- R1 R2 M

Jump to M if R1 < R2, using signed 2's complement arithmetic.

### 65: JLI+ V R M

Jump to M if V < R.

### 66: JLI- V R M

Jump to M if V < R, using signed 2's complement arithmetic.

### 98: JAR R1 R2 M

Jump to M if R1 AND R2 is nonzero.

### 99: JAI V R M

Jump to M if V AND R is nonzero.

### 67: FORKR+ R1 R2 M1 M2

Fork instruction. If R1 < R2 then jump to M1. If R1 > R2 then jump to M2.

### 68: FORKR- R1 R2 M1 M2

Fork instruction. If R1 < R2 then jump to M1. If R1 > R2 then jump to M2 using 2's complement arithmetic.

### 69: FORKI+ V R M1 M2

Fork instruction. If V < R then jump to M1. If V > R then jump to M2.

### 70: FORKI- V R M1 M2

Fork instruction. If V < R then jump to M1. If V > R then jump to M2 using 2's complement arithmetic.

## Stack manipulation

### 71: PUSHR R1 R2

Push R1 to stack R2.

### 72: PUSHI V R

Push V to stack R.

### 73: POP R1 R2

Pop from stack R1 into R2.

### 74: R R

Push R onto stack 11

### 75: I V

Push V onto stack 11

### 76: G R

Get from stack 11 into R.

### 77: PSR R

Preserve register. Push R onto stack 12.

### 78: RSR R

Recover register. Pop from stack 12 into R.

### 79: S V

Send registers given by V, using stack 11.

### 80: REC V

Receive regisetrs given by V, using stack 11.

### 81: PE V

Preserve registers given by V, using stack 12.

### 82: RE V

Recover registers given by V, using stack 12.

### 83: PR V1 V2

Preserve, then receive registers given by V1 and V2 respectively.

### 84: SR V1 V2

Send, then recover registers given by V1 and V2 respectively.

## Subroutine calls

### 85: [ M

Call routine at address M and place return address on stack 12.

### 86: ]

Return from routine.

### 87: EXIT V

Recover registers given by V and return from subroutine.

### 88: FUNC V M

Send registers given by V, then call the subroutine located at memory address M.

### 89: SRE V1 V2

Send, recover and exit. Send registers given by V1 and recover those given by V2 before returning from the subroutine.

## Bulk memory instructions

### 101: MCOPYR R1 R2 R3

Copy from location in R1, length in R2 to location in R3. Length is signed, negative values will copy backwards from R1 the specified negative length.

### 102: MCOPYI V R1 R2

Copy from location in V, length in R1 to location in R2. Length is signed.

### 103: MFILLR R1 R2 R3

Fill location R1, length R2 with filler byte R3.

### 104: MFILLI V R1 R2

Fill location V, length R1 with filler byte R2.

## DEX

Any opcode that's not assigned is treated as a subroutine call.