CHIP-8-Interpreter

Das ist der komplette(!) originale CHIP-8-Interpreter des COSMAC VIP. Quelle: VIPER-Magazin 1-02 August 1978.

Der CHIP-8-Interpreter ist extrem platzsparend programmiert, er belegt nicht einmal 0.5 KByte! Diese Kompaktheit spiegelt sich beispielsweise auch in den Hex-Werten der Maschinencode-Instruktionen wieder: Die Befehle der F-Gruppe haben einen zweistellige Nummer, diese entspricht der Startadresse der zugehörigen Befehlsinterpretation. Damit sparte man sich die Bytes für eine zusätzliche Sprungtabelle.

CDP 1802

Das Handbuch zum COSMAC VIP enthält auch das Datenblatt zum Prozessor CDP 1802 incl. Befehlsliste.

- http://visual6502.org/images/pages/RCA_1802.html riesiges Bild des Chips (7164 x 5496 Pixxel, 19 MByte!)
- http://www.visual6502.org/wiki/index.php?title=RCA_1802E 1082 Beschreibung

COSMAC Register Summary

```
D   8 Bits     Data Register (Accumulator)
N   4 Bits     Holds Low-Order Instr. Digit
DF  1 Bit     Data Flag (ALU Carry)
I   4 Bits     Holds High-Order Instr. Digit
R   16 Bits 1 of 16 Scratchpad Registers
  R0.1 R0.0
  R1.1 R1.0
  ...
  RF.1 RF.0
T   8 Bits     Holds old X, P after Interrupt (X is high byte)
P   4 Bits     Designates which register is Program Counter
                   zeigt auf Registerarray
IE  1 Bit     Interrupt Enable
X   4 Bits     Designates which register is Data Pointer
                   zeigt auf Registerarray
Q   1 Bit     Output Flip Flop
```

instruction set

```
MNEM    NAME                                        OPCODE
ADC     Add with Carry                              74
ADCI b  Add with Carry Immediate                    7C bb
ADD     Add                                         F4
ADI b   Add Immediate                               FC bb
AND     Logical AND                                 F2
```

```
ANI b   AND Immediate                           FA bb
B1 a    Branch on External Flag 1               34 aa
B2 a    Branch on External Flag 2               35 aa
B3 a    Branch on External Flag 3               36 aa
B4 a    Branch on External Flag 4               37 aa
BDF a   Branch if DF is 1                       33 aa
BN1 a   Branch on Not External Flag 1           3C aa
BN2 a   Branch on Not External Flag 2           3D aa
BN3 a   Branch on Not External Flag 3           3E aa
BN4 a   Branch on Not External Flag 4           3F aa
BNF a   Branch if DF is 0                       3B aa
BNQ a   Branch if Q is off                      39 aa
BNZ a   Branch on Not Zero                      3A aa
BQ a    Branch if Q is on                       31 aa
BR a    Branch unconditionally                  30 aa
BZ a    Branch on Zero                          32 aa
DEC r   Decrement Register                      2r
DIS     Return and Disable Interrupts           71
GHI r   Get High byte of Register               9r
GLO r   Get Low byte of Register                8r
IDL     Idle                                    00
INC r   Increment Register                      1r
INP p   Input to memory and D (for p = 9 to F)  6p
IRX     Increment R(X)                          60
LBDF aa Long Branch if DF is 1                  C3 aaaa
LBNF aa Long Branch if DF is 0                  CB aaaa
LBNQ aa Long Branch if Q is off                 C9 aaaa
LBNZ aa Long Branch if Not Zero                 CA aaaa
LBQ aa  Long Branch if Q is on                  C1 aaaa
LBR aa  Long Branch unconditionally             C0 aaaa
LBZ aa  Long Branch if Zero                     C2 aaaa
LDA r   Load D and Advance                      4r
LDI b   Load D Immediate                        F8 bb
LDN r   Load D via N (for r = 1 to F)           0r
LDX     Load D via R(X)                         F0
LDXA    Load D via R(X) and Advance             72
LSDF    Long Skip if DF is 1                    CF
LSIE    Long Skip if Interrupts Enabled         CC
LSKP    Long Skip                               C8
LSNF    Long Skip if DF is 0                    C7
LSNQ    Long Skip if Q is off                   C5
LSNZ    Long Skip if Not Zero                   C6
LSQ     Long Skip if Q is on                    CD
LSZ     Long Skip if Zero                       CE
MARK    Save X and P in T                       79
NOP     No Operation                            C4
OR      Logical OR                              F1
ORI b   OR Immediate                            F9 bb
OUT p   Output from memory (for p = 1 to 7)     6p
PHI r   Put D in High byte of register          Br
PLO r   Put D in Low byte of register           Ar
```

```
REQ     Reset Q                                      7A
RET     Return                                       70
SAV     Save T                                       78
SD      Subtract D from memory                       F5
SDB     Subtract D from memory with Borrow           75
SDBI b  Subtract D with Borrow, Immediate            7D bb
SDI b   Subtract D from memory Immediate byte        FD bb
SEP r   Set P                                        Dr
SEQ     Set Q                                        7B
SEX r   Set X                                        Er
SHL     Shift D Left                                 FE
SHLC    Shift D Left with Carry                      7E
SHR     Shift D Right                                F6
SHRC    Shift D Right with Carry                     76
SKP     Skip one byte                                38
SM      Subtract Memory from D                       F7
SMB     Subtract Memory from D with Borrow           77
SMBI b  Subtract Memory with Borrow, Immediate       7F bb
SMI b   Subtract Memory from D, Immediate            FF bb
STR r   Store D into memory                          5r
STXD    Store D via R(X) and Decrement               73
XOR     Exclusive OR                                 F3
XRI b   Exclusive OR, Immediate                      FB bb
```

Interpreter

```
        ; J.W. Wentworth's Analysis of VIP CHIP 8 Interpreter
        ; retyped vp130331

        cpu    1802
        ; '*' before opcode marks branch destination

0000: 91      GHI  R1    ; 0X ->    RB.1 (where 0X is then highest memory
page no. as determined
0001: BB      PHI  RB    ; by Operating System) -- designates display page
0002: FF 01   SMI  01h    ; 0X-1 -> R2.1 (stack pointer)
0004: B2      PHI  R2
0005: B6      PHI  R6    ; 0X-1 -> R6.1 (VX pointer)
0006: F8 CF   LDI  CF
0008: A2      PLO  R2    ; CF ->    R2.0 (stack pointer set    to 0YCF,
where Y=X-1)
0009: F8 81   LDI  81h    ; Set R1 (PC for Interrupt Routine) to 8146
000B: B1      PHI  R1
000C: F8 46   LDI  46h
000E: A1      PLO  R1
000F: 90      GHI  R0    ; Pre-set R4 to    001B in    preparation for
assignment as PC
0010: B4      PHI  R4
0011: F8 1B   LDI  1Bh
```

```
0013: A4      PLO  R4
0014: F8 01   LDI  01h    ; 01 ->    R5.1
0016: B5      PHI  R5
0017: F8 FC   LDI  FC    ; FC ->    R5.0  (R5 now set to 01FC; will    serve
as PC for    CHIP-8
0019: A5      PLO  R5    ; instructions,    commencing with    two
instructions included on page 01
001A: D4      SEP  R4    ; 4 -> P (R4 becomes PC    at this    point)

        ; FETCH    AND DECODE ROUTINE

001B: 96    * GHI  R6    ; 0Y ->    R7.1 (high byte    of VY pointer)
001C: B7      PHI  R7
001D: E2      SEX  R2    ; 2 -> X
001E: 94      GHI  R4    ; 00 ->    RC.1
001F: BC      PHI  RC
0020: 45      LDA  R5    ; Load by R5 and advance (Fetch    first byte of
Chip-8 Instruction)
0021: AF      PLO  RF    ; Put in RF.0 (temporary storage)
0022: F6      SHR        ; Shift    right 4    times (MSD to LSD position)
0023: F6      SHR
0024: F6      SHR
0025: F6      SHR
0026: 32 44   BZ   44h    ; If D=0 (i.e.,    if Op Code digit is zero),
branch to 0044
0028: F9 50   ORI  50h    ; OR Immediate with 50
002A: AC      PLO  RC    ; Put in RC.0 (RC now points to 005a, where "a" is
MSD of CHIP-8 Instruction)
002B: 8F      GLO  RF    ; Get RF.0 (high byte of instruction)
002C: FA 0F   ANI  0Fh    ; AND with 0F, OR with F0 (thus forming byte Fb,
where "b" is the second
002E: F9 F0   ORI  F0    ; hex digit in the CHIP-8 Instruction, used in some
instructions to designate VX)
0030: A6      PLO  R6    ; Put in R6.0 (R6 becomes VX pointer)
0031: 05      LDN  R5    ; Load via R5 (second byte of CHIP-8 Instruction)
0032: F6      SHR        ; Shift right 4 times
0033: F6      SHR
0034: F6      SHR
0035: F6      SHR
0036: F9 F0   ORI  F0    ; OR with F0
0038: A7      PLO  R7    ; and put in R7.0 (sets VY pointer)
0039: 4C      LDA  RC    ; Load via RC and advance (Loads high byte of
address for appropriate subroutine)
003A: B3      PHI  R3    ; Put in R3.1
003B: 8C      GLO  RC    ; Get RC.0 (=5(a+1))
003C: FC 0F   ADI  0Fh    ; Add 0F, put in RC.0 (points to low byte of start
address for appropriate
003E: AC      PLO  RC    ; subroutine to execute CHIP-8 Instruction)
003F: 0C      LDN  RC    ; Load via RC
0040: A3    * PLO  R3    ; Put in R3.0
0041: D3      SEP  R3    ; Call subroutine designated by first digit of
```

```
CHIP-8 Instruction (if not zero)
0042: 30 1B * BR   1Bh    ; Branch to 001B to fetch next instruction

        ; ROUNTINE FOR FIRST DIGIT 0

0044: 8F    * GLO  RF    ; Get RF.0 (high byte of instruction)
0045: FA 0F   ANI  0Fh    ; AND with 0F to save LSD only
0047: B3      PHI  R3    ; Put in R3.1 (select page on which subroutine will
found)
0048: 45      LDA  R5    ; Load via R5 and advance (2nd byte of inst.)
0049: 30 40   BR   40h    ; Branch to 0040 to call subroutine (00E0 for
erase page, 00EE for return from
                ; subroutine, 0MMM for machine-language subroutine)

004B: 22      DEC  R2    ; Decrement R2 (stack pointer)
004C: 69      INP  1h    ; Turn display ON (interrupts will occur,
controlled by routine at 8146)
004D: 12      INC  R2    ; Increment R2
004E: D4      SEP  R4    ; return to 0042
004F: 00      IDL        ; Filler
0050: 00      IDL        ; Filler

0051: 01      db 1        ; High bytes for pointer to start of subroutines
selected by first digit
0052: 01      db 1        ; of CHIP 8 instructions (1 through F)
0053: 01      db 1
0054: 01      db 1
0055: 01      db 1
0056: 01      db 1
0057: 01      db 1
0058: 01      db 1
0059: 01      db 1
005A: 01      db 1
005B: 01      db 1
005C: 01      db 1
005D: 00      db 0
005E: 01      db 1
005F: 01      db 1
;
0060: 00    IDL        ; Filler

;               ; Low Bytes for subroutine pointers
0061: 7C        db 7Ch      ; 1 -> 017C
0062: 75        db 75h      ; 2 -> 0175
0063: 83        db 83h          ; 3 -> 0183
0064: 8B        db 8Bh          ; 4 -> 018B
0065: 95        db 95h          ; 5 -> 0195
0066: B4        db 0B4h         ; 6 -> 01B4
0067: B7        db 0B7h         ; 7 -> 01B7
0068: BC        db 0BCh         ; 8 -> 01BC
0069: 91        db 91h          ; 9 -> 0191
```

```
006A: EB        db 0EBh         ; A -> 01EB
006B: A4        db 0A4h         ; B -> 01A4
006C: D9        db 0D9h         ; C -> 01D9
006D: 70        db 70h          ; D -> 0070
006E: 99        db 99h          ; E -> 0199
006F: 05        db 05h          ; F -> 0105
;

        ; DISPLAY SUBROUTINE (1st Digit = D)

0070: 06      * LDN  R6    ; Load by R6 (VX)
0071: FA 07     ANI  07h    ; Put last 3 bits in RE.1
0073: BE        PHI  RE
0074: 06        LDN  R6    ; Load by R6
0075: FA 3F     ANI  3Fh    ; AND with 3F (save lower 6 bits)
0077: F6        SHR        ; shift right 3 times (save middle 3 digits)
0078: F6        SHR
0079: F6        SHR
007A: 22        DEC  R2    ; Decrement Stack
007B: 52        STR  R2    ; Store in Stack
007C: 07        LDN  R7    ; Load via R7 (VY)
007D: FA 1F     ANI  1Fh    ; AND with 1F (save 5 lowest bits)
007F: FE        SHL        ; shift left 3 times
0080: FE        SHL
0081: FE        SHL
0082: F1        OR        ; OR with top of stack
0083: AC        PLO  RC    ; Put Result in RC.0
0084: 9B        GHI  RB    ; Get RB.1, put in RC.1 (0X) -- RC now points to
start address
0085: BC        PHI  RC    ; of first byte of pattern
0086: 45        LDA  R5    ; Load via R5 and advance -- fetches 2nd byte of
CHIP 8 instruction
0087: FA 0F     ANI  0F    ; Put LSD in both RD.0 and R7.0 (No. of bytes in
pattern)
0089: AD        PLO  RD
008A: A7        PLO  R7
008B: F8 D0     LDI  D0    ; D0 -> R6.0
008D: A6        PLO  R6
008E: 93      * GHI  R3    ; 00 -> RF.0
008F: AF        PLO  RF
0090: 87        GLO  R7    ; Get R7.0 (No. of bytes); branch if D=0 to 00F3
(when branch occurs,
0091: 32 F3     BZ   F3    ; display bytes have been processed and stored
commencing at 0YD0)
0093: 27        DEC  R7    ; Decrement R7
0094: 4A        LDA  RA    ; Load via RA (I pointer) and advance (Loads
display byte)
0095: BD        PHI  RD    ; Put in RD.1
0096: 9E        GHI  RE    ; Get RE.1 (3 LSB's of VX), Put in RE.0
0097: AE        PLO  RE
0098: 8E        GLO  RE    ; Get RE.0; if D=0, branch to 00A4 (When branch
```

```
occurs, display bytes
0099: 32 A4   BZ   A4    ; will have been split into two parts in the event
that display address
                ; did not coincide with as memory byte address)
009B: 9D      GHI  RD    ; Get RD.1,
009C: F6      SHR        ; shift right,
009D: BD      PHI  RD    ; and return to RD.1 (left portion of split diplay
byte)
009E: 8F      GLO  RF    ; Get RF.0,
009F: 76      SHRC        ; ring shift right (picking up carry, if any, from
step 009C),
00A0: AF      PLO  RF    ; return to RF.0 -- thes instructions form right
portion of split display byte.
00A1: 2E      DEC  RE    ; Decrement RE
00A2: 30 98   BR   98    ; Branch to 0098
        ;
00A4: 9D    * GHI  RD    ; Get RD.1 (left portion of split diplay byte)
00A5: 56      STR  R6    ; and store via R6
00A6: 16      INC  R6    ; Increment R6
00A7: 8F      GLO  RF    ; Get RF.0 (right portion of split diplay byte)
00A8: 56      STR  R6    ; and store via R6
00A9: 16      INC  R6    ; Increment R6
00AA: 30 8E   BR   8E    ; Branch to 008E
        ;
00AC: 00    * IDL        ; Wait for display interrupt
00AD: EC      SEX  RC    ; C -> X (C points to start address for first new
byte in display page)
00AE: F8 D0   LDI  D0    ; D0 -> R6.0 (points to first processed display
byte)
00B0: A6      PLO  R6
00B1: 93      GHI  R3    ; 00 -> R7.0 (R7.0 will be used as a marker for
"collisions"
00B2: A7      PLO  R7    ; between new and existing pattern)
00B3: 8D    * GLO  RD    ; Get RD.0 (no. of bytes remaining);
00B4: 32 D9   BZ   D9    ; if D=0, branch to 00D9
00B6: 06      LDN  R6    ; Load via R6 (processed display byte)
00B7: F2      AND        ; AND with contents at current address in pattern
on display page
00B8: 2D      DEC  RD    ; Decrement RD
00B9: 32 BE   BZ   BE    ; if D=0 (i.e., no "collision" occurs) branch to
00BE
00BB: F8 01   LDI  01    ; 01 -> R7.0 (marker to indicate that a "collision"
has occured)
00BD: A7      PLO  R7
00BE: 46    * LDA  R6    ; Load via R6 (processed display byte)
00BF: F3      XOR        ; XOR with existing byte
00C0: 5C      STR  RC    ; Store via RC (in display page)
00C1: 02      LDN  R2    ; Load from top of stack (3 LBS's of display page
address),
00C2: FB 07   XRI  07    ; XOR with 07;
00C4: 32 D2   BZ   D2    ; if result is zero, branch to 00D2 (display
```

```
pattern is at
                ; right-hand edge of display "window")
00C6: 1C      INC  RC    ; Increment RC
00C7: 06      LDN  R6    ; Load via R6 (right portion of processed diplay
byte),
00C8: F2      AND        ; AND via R6 (existing contents of display page
address);
00C9: 32 CE   BZ   CE    ; branch if result is zero (i.e., if there is no
"collision") to 00CE
00CB: F8 01   LDI  01    ; 01 -> R7.0 (marker to indicate that a "collision"
has occured)
00CD: A7      PLO  R7
00CE: 06    * LDN  R6    ; Load via R6,
00CF: F3      XOR        ; XOR with contents already present at designated
address of display
00D0: 5C      STR  RC    ; page, and store via RC (on display page)
00D1: 2C      DEC  RC    ; Decrement RC
00D2: 16    * INC  R6    ; Increment R6
00D3: 8C      GLO  RC    ; Get RC.0,
00D4: FC 08   ADI  08    ; Add 08,
00D6: AC      PLO  RC    ; and return to RC.0
00D7: 3B B3   BNF  B3    ; if DF=0 (i.e., if next byte location remains on
display page), branch to 00B3
00D9: F8 FF * LDI  FF    ; FF -> R6.0 (R6 points to Variable F)
00DB: A6      PLO  R6
00DC: 87      GLO  R7
00DD: 56      STR  R6
00DE: 12      INC  R2    ; Increment stack
00DF: D4    * SEP  R4    ; Return to FETCH routine at 0042

        ; ERASE DISPLAY PAGE -- Inst. 00E0

00E0: 9B      GHI  RB    ; 0X -> RF.1
00E1: BF      PHI  RF
00E2: F8 FF   LDI  FF    ; FF -> RF.0
00E4: AF      PLO  RF
00E5: 93    * GHI  R3    ; 00 -> D
00E6: 5F      STR  RF    ; Store via RF
00E7: 8F      GLO  RF    ; Get RF.0;
00E8: 32 DF   BZ   DF    ; if zero, branch to 00DF for exit to FETCH
00EA: 2F      DEC  RF    ; Decrement RF
00EB: 30 E5   BR   E5    ; Branch to 00E5

00ED: 00      IDL        ; Filler

        ; INST. 00EE -- Return from Subroutine

00EE: 42      LDA  R2    ; Load from stack and advance,
00EF: B5      PHI  R5    ; put in R5.1
00F0: 42      LDA  R2    ; Load from stack and advance,
00F1: A5      PLO  R5    ; put in R5.0 (R5 now points to next CHIP 8
```

```
instruction)
00F2: D4      SEP  R4    ; Return to FETCH routine at 0042

        ; PART OF DISPLAY SUBROUTINE

00F3: 8D    * GLO  RD    ; Get RD.0 (remaining no. of bytes in pattern),
00F4: A7      PLO  R7    ; put in R7.0
00F5: 87    * GLO  R7    ; Get R7.0;
00F6: 32 AC   BZ   AC    ; if zero, branch to 00AC (When branch occurs, RA
(I Pointer)
                ; will have returned to its initial value)
00F8: 2A      DEC  RA    ; Decrement RA
00F9: 27      DEC  R7    ; Decrement R7
00FA: 30 F5   BR   F5    ; Branch to 00F5
        ;
00FC: 00      IDL        ; Fillers
00FD: 00      IDL
00FE: 00      IDL
00FF: 00      IDL
0100: 00      IDL
0101: 00      IDL
0102: 00      IDL
0103: 00      IDL
0104: 00      IDL

        ; FINAL DECODING OF "F" instructions

0105: 45      LDA  R5    ; Load via R5 and advance (2nd byte of CHIP 8
instruction)
0106: A3      PLO  R3    ; put in R3 (go to designated address)

        ; Instruction FX07 (Let VX = Timer)

0107: 98      GHI  R8    ; Get R8.1
0108: 56      STR  R6    ; Store via R6 (i.e., as VX)
0109: D4      SEP  R4    ; Return to 0042

        ; Instruction FX0A (Let VX = Hex Key)

010A: F8 81   LDI  81    ; RC = 8195 (keyboard scanning subroutine)
010C: BC      PHI  RC
010D: F8 95   LDI  95
010F: AC      PLO  RC
0110: 22      DEC  R2    ; Decrement stack pointer
0111: DC      SEP  RC    ; Call keyboard scanning subroutine at 8195 (key
entry is in D upon return)
0112: 12      INC  R2    ; Increment stack pointer
0113: 56      STR  R6    ; Store via R6 (i.e., as VX)
0114: D4      SEP  R4    ; Return to 0042

        ; Instruction FX15 (Set Timer to VX)
```

```
0115: 06      LDN  R6    ; Load via R6 (loads VX),
0116: B8      PHI  R8    ; put in R8.1
0117: D4      SEP  R4    ; Return to 0042

        ; Instruction FX18 (Set tone duration = VX)

0118: 06      LDN  R6    ; Load via R6 (loads VX)
0119: A8      PLO  R8    ; Put in R8.0 (tone timer)
011A: D4      SEP  R4    ; Return to 0042

        ; Constants needed for Instruction FX33

011B: 64    * db    100
011C: 0A      db    10
011D: 01      db    1

        ; Instruction FX1E

011E: E6      SEX  R6    ; 6 -> X
011F: 8A      GLO  RA    ; Get RA.0
0120: F4      ADD        ; Add VX
0121: AA      PLO  RA    ; Put in RA.0 (as updated I pointer)
0122: 3B 28   BNF  28    ; If DF=0 (i.e., if updated I remains on the same
memory page), branch to 0128
0124: 9A      GHI  RA    ; Increment RA.1
0125: FC 01   ADI  01
0127: BA      PHI  RA
0128: D4    * SEP  R4    ; Return to 0042

        ; Instruction FX29 (Let I = 5-byte diplay pattern for LSD of VX)

0129: F8 81   LDI  81    ; 81 -> RA.1
012B: BA      PHI  RA
012C: 06      LDN  R6    ; Load via R6 (VX)
012D: FA 0F   ANI  0F    ; AND with 0F (save last digit),
012F: AA      PLO  RA    ; put in RA.0
0130: 0A      LDN  RA    ; Load via RA (start address for 5-byte pattern of
hex digit),
0131: AA      PLO  RA    ; put in RA.O
0132: D4      SEP  R4    ; Return to 0042

        ; Instruction FX33 (Let MI = 3-decimal digit equivalent of VX)

0133: E6      SEX  R6    ; 6 -> X
0134: 06      LDN  R6    ; Load via R6 (VX),
0135: BF      PHI  RF    ; put in RF.1
0136: 93      GHI  R3    ; 01 -> RE.1
0137: BE      PHI  RE
0138: F8 1B   LDI  1B    ; 1B -> RE.0 (RE = 011B)
013A: AE      PLO  RE
013B: 2A      DEC  RA    ; Decrement RA
```

```
013C: 1A    * INC  RA    ; Increment RA (cancels prev. step upon first entry
into pgm loop,
                ; but needed in later "passes" around the loop}
013D: F8 00   LDI  00    ; Store 00 via RA (I pointer)
013F: 5A      STR  RA
0140: 0E    * LDN  RE    ; load via RE (Decimal 100, 10 or 1)
0141: F5      SD        ; Subtract from M(R6) -- i.e., subtract from YX
0142: 3B 4B   BNF  4B    ; Branch if Minus to 014B
0144: 56      STR  R6    ; Store result via R6
0145: 0A      LDN  RA    ; Increment memory location contents pointed to by
RA (= I Pointer)
0146: FC 01   ADI  01
0148: 5A      STR  RA
0149: 30 40   BR   40    ; Branch to 0140
014B: 4E    * LDA  RE    ; load via RE and advance
014C: F6      SHR        ; Shift Right
014D: 3B 3C   BNF  3C    ; If DF=0 (i.e.,if decimal 100's, 10's and 1's have
not been processed), branch to 013C
014F: 9F      GHI  RF    ; Get RF. 1 (original value of VX)
0150: 56      STR  R6    ; Stcre via R6 (restores original value of VX)
0151: 2A      DEC  RA    ; Decrement RA twice
0152: 2A      DEC  RA
0153: D4      SEP  R4    ; Return to 0042

0154: 00      IDL        ; Filler

        ; Instruction FX55 (Let MI = V0:VX)

0155: 22      DEC  R2    ; Decrement stack pointer
0156: 86      GLO  R6    ; Get R6.0 (pointer for VX),
0157: 52      STR  R2    ; and store in stack
0158: F8 F0   LDI  F0    ; F0 -> R7.0
015A: A7      PLO  R7
015B: 07    * LDN  R7    ; Load via R7
015C: 5A      STR  RA    ; Store via RA (I pointer)
015D: 87      GLO  R7    ; Get R7.0
015E: F3      XOR        ; XOR with top of stack (VX pointer)
015F: 17      INC  R7    ; Increment R7
0160: 1A      INC  RA    ; Increment RA
0161: 3A 5B   BNZ  5B    ; If D<>0 (i.e., if R7 at Step 5E has not yet
reached value of YX pointer), branch to 015B
0163: 12      INC  R2    ; Increment stack pointer
0164: D4      SEP  R4    ; Return to 0042

        ; Instruction FX65 (Let V0:VX = MI)

0165: 22      DEC  R2    ; Decrement stack pointer
0166: 86      GLO  R6    ; Get R6.0 (pointer for VX)
0167: 52      STR  R2    ; Store in stack
0168: F8 F0   LDI  F0    ; F0 -> R7.0
016A: A7      PLO  R7
```

```
016B: 0A    * LDN  RA    ; Load via RA (i.e., via I)
016C: 57      STR  R7    ; Store via R7
016D: 87      GLO  R7    ; Get R7.0
016E: F3      XOR        ; XOR with top of stack (VX pointer)
016F: 17      INC  R7    ; Increment R7
0170: 1A      INC  RA    ; Increment RA
0171: 3A 6B   BNZ  6B    ; If D<>0 (i.e., if R7 at Step 6E has not yet
reached value of YX pointer), branch to 016B
0173: 12      INC  R2    ; Increment stack pointer
0174: D4      SEP  R4    ; Return to 0042

        ; Instruction 2MMM (do subroutine at MMM)

0175: 15      INC  R5    ; Increment R5 (point to next CHIP 8 instruction
after return)
0176: 85      GLO  R5    ; Get R5.0
0177: 22      DEC  R2    ; Decrement stack pointer
0178: 73      STXD         ; Store in stack and decrement
0179: 95      GHI  R5    ; Get R5.1
017A: 52      STR  R2    ; Store in stack
017B: 25      DEC  R5    ; Decrement R5 (points to low byte of current
instruction)
017C: 45      LDA  R5    ; Load via R5 and advance
017D: A5      PLO  R5    ; Put in R5.0
017E: 86      GLO  R6    ; Get R6.0 (contains 2nd digit of current
instruction)
017F: FA 0F   ANI  0F    ; AND with OF (save 2nd digit of Chip 8
instruction)
0181: B5      PHI  R5    ; and put in R5.1 (R5 now points to first
instruction of subroutine commencing at 0MMM)
0182: D4    * SEP  R4    ; Return to 0042

        ; Instruction 3XKK (Skip if VX=KK)

0183: 45      LDA  R5    ; Load by R5 and advance (KK -> D)
0184: E6    * SEX  R6    ; 6 -> X
0185: F3      XOR        ; XOR (operands are KK and VX)
0186: 3A 82   BNZ  82    ; If D <> 0 (i.e., if VX <> KK), branch to 0182
0188: 15    * INC  R5    ; Increment R5 twice (causing skip of next Chip 8
instruction)
0189: 15      INC  R5
018A: D4      SEP  R4    ; Return to 0042

        ; Instruction 4XKK (Skip if VX<>KK)

018B: 45      LDA  R5    ; Load by R5 and advance (KK -> D)
018C: E6    * SEX  R6    ; 6 -> X
018D: F3      XOR        ; XOR (operands are KK and VX)
018E: 3A 88   BNZ  88    ; If D <> 0, branch to 0188
0190: D4      SEP  R4    ; Return to 0042
```

```
        ; Instruction 9XY0 (Skip if VK<>VY)

0191: 45      LDA  R5    ; Load by R5 and advance
0192: 07      LDN  R7    ; Load by R7 (VY ->  D)
0193: 30 8C   BR   8C    ; Branch to 018C {operands for subsequent XOR
operation will be VY and VX)

        ; Instruction 5XY0 (Skip if VK=VY)

0195: 45      LDA  R5    ; Load by R5 and advance
0196: 07      LDN  R7    ; Load by R7 (VY ->  D)
0197: 30 84   BR   84    ; Branch to 0184 {operands for subsequent XOR
operation will be VY and VX)

        ; Instruction EX9E (Skip if VX=Key)
        ; and EX91 (Skip if VX<>Key)

0199: E6      SEX  R6    ; 6 -> X
019A: 62      OUT  2    ; Output VX to keyboard latch, increment R6
019B: 26      DEC  R6    ; Decrement R6 (cancel advance of prev 0 step)
019C: 45      LDA  R5    ; load by R5 and advance (either 9E or A1 is loaded
into D)
019D: A3      PLO  R3    ; Put in R3 (go to designated address)
        ;
019E: 36 88   B3   88    ; if EF3 = 1 (i.e., if key matching LSD of VX is
down) go to 0188
01A0: D4      SEP  R4    ; Return to 0042
        ;
01A1: 3E 88   BN3  88    ; If EF3 = 0 (hex key matching LSD of VX not
pressed), branch to 0188
01A3: D4      SEP  R4    ; Return to 0042

        ; Instruction BMMM (Go to 0MMM + V0)

01A4: F8 F0   LDI  F0    ; FO -> R7.0 (R7 points to V0)
01A6: A7      PLO  R7
01A7: E7      SEX  R7    ; 7 -> X
01A8: 45      LDA  R5    ; Load by R5 and advance (Loads 2nd byte of
instruction)
01A9: F4      ADD        ; Add V0
01AA: A5      PLO  R5    ; Put in R5.0
01AB: 86      GLO  R6    ; Get LSD af R6.0 (2nd digit of instruction)
01AC: FA 0F   ANI  0F
01AE: 3B B2   BNF  B2    ; If DF=0 (i.e., if there was no carry from
addition operation at Step A9), branch to 0182
01B0: FC 01   ADI  01    ; Add 01
01B2: B5      PHI  R5    ; Put in R5.1
01B3: D4      SEP  R4    ; Return to 0042

        ; Instruction 6XKK (Let VX = KK)
```

```
01B4: 45      LDA  R5    ; Load by R5 and advance (KK -> D)
01B5: 56      STR  R6    ; Store via R6 (as VX)
01B6: D4      SEP  R4    ; Return to 0042

        ; Instruction 7XKK (Let VX = VX+KK)

01B7: 45      LDA  R5    ; Load by R5 and advance (KK -> D)
01B8: E6      SEX  R6    ; 6 -> X
01B9: F4      ADD        ; Add (D = VX+KK)
01BA: 56      STR  R6    ; Store via R6 (as updated VX)
01BB: D4      SEP  R4    ; Return to 0042

        ; Instruction 8XYN (ALU operations with VX and VY as operands)

01BC: 45      LDA  R5    ; Load by R5 and advance (Loads 2nd byte of
instruction)
01BD: FA 0F   ANI  0F    ; AND with 0F (save 2nd digit)
01BF: 3A C4   BNZ  C4    ; If D<>0, branch to 01C4
01C1: 07      LDN  R7    ; load by R7 (VY -> D)
01C2: 56      STR  R6    ; Store via R6 (as VX)
01C3: D4      SEP  R4    ; Return to 0042
        ;
01C4: AF    * PLO  RF    ; Put in RF.0
01C5: 22      DEC  R2    ; Decrement stack pointer
01C6: F8 D3   LDI  D3    ; D3->D
01C8: 73      STXD        ; Store in stack and decrement
01C9: 8F      GLO  RF    ; Get RF.0 (last digit of instruction)
01CA: F9 F0   ORI  F0    ; OR with F0 (forms an instruction code in the ALU
group
                ; --codes F1, F2, F3, F4, F5, F6, F7 and FE are valid}
01CC: 52      STR  R2    ; Store in stack (stack now holds a 2-instruction
routine)
01CD: E6      SEX  R6    ; 6 -> X
01CE: 07      LDN  R7    ; load by R7 (VY -> D)
01CF: D2      SEP  R2    ; 2 -> D (calls routine developed in stack)
01D0: 56      STR  R6    ; Store result via R6 (as VX)
01D1: F8 FF   LDI  FF    ; FF -> R6.0 (points to VF)
01D3: A6      PLO  R6
01D4: F8 00   LDI  00    ; 00 -> D
01D6: 7E      SHLC        ; Ring Shift Left (moves 0F to LSB)
01D7: 56      STR  R6    ; Store via R6 (as VF)
01D8: D4      SEP  R4    ; Return to 0042

        ; Instruction CXKK (Let VX = Random Byte, masked by KK)

01D9: 19      INC  R9    ; Increment R9
01DA: 89      GLO  R9    ; Get R9.0, put in RE.0 (NOTE: R9 is a special
pointer for this random-number
                ; generator, and is incremented once for every TV scan by
the interrupt routine.)
01DB: AE      PLO  RE
```

```
01DC: 93      GHI  R3    ; 01 -> RE.1
01DD: BE      PHI  RE    ; (RE now points to some byte on memory page 01)
01DE: 99      GHI  R9    ; Get R9.1 (Random byte resulting from last
previous use of this instruction)
01DF: EE      SEX  RE    ; E -> X
01E0: F4      ADD        ; Add byte pointed to by RE
01E1: 56      STR  R6    ; Store via R6
01E2: 76      SHRC        ; Ring shift right
01E3: E6      SEX  R6    ; 6 -> X
01E4: F4      ADD        ; Add original byte formed at Step 01E0 to its
ring-shifted version
01E5: B9      PHI  R9    ; Put in R9.1 (as starting point for next use of
this instruction)
01E6: 56      STR  R6    ; Store via R6 (byte still un-masked)
01E7: 45      LDA  R5    ; Load via R5 and advance (loads 2nd byte of Ch ip
8 instruction, KK)
01E8: F2      AND        ; AND with byte pointed to by R6
01E9: 56      STR  R6    ; Store result via R6 (as VX)
01EA: D4      SEP  R4    ; Return to 0042

        ; Instruction AMMM (Let I = 0MMM)

01EB: 45      LDA  R5    ; Load by R5 and advance (2nd byte of instruction)
01EC: AA      PLO  RA    ; Put in RA.0
01ED: 86      GLO  R6    ; Get R6.0
01EE: FA 0F   ANI  0F
01F0: BA      PHI  RA    ; Put 2nd digit in RA.1
01F1: D4      SEP  R4    ; Return to 0042
        ;
01F2: 00      IDL        ; Fillers
01F3: 00      IDL
01F4: 00      IDL
01F5: 00      IDL
01F6: 00      IDL
01F7: 00      IDL
01F8: 00      IDL
01F9: 00      IDL
01FA: 00      IDL
01FB: 00      IDL

        ; Preliminary CHIP 8 instructions to precede every CHIP 8 programm

01FC: 00 E0 * dw    00E0    ; calls routine at 00E0 to erase display page
01FE: 00 4B   dw    004B    ; calls routine at 0042 to turn on display

        END

; SUMMARY OF REGISTER FUNCTIONS IN CHIP 8 PROGRAMS
;
;                                    Initial Setting
; RO DMA Pointer                             ----
```

```
; R1 Program Counter (PC) for Interrupt Routine             8146
; R2 Stack Pointer                                  0YCF
; R3 PC for Interpreter Subroutines                         ----
; R4 PC for Interpreter FETCH AND DECODE routine            001B
; R5 Pointer for CHIP 8 Instructions                    01FC
; R6 VX Pointer: in DXYN (Display) Instructions, also serves        0Y--*
;    as pointer for processed display bytes, later as VF pointer
; R7 VY Pointer for instructions involving VY; V0 Pointer for       0Y--*
;    BMMM Instructions; R7.0 is a "scratch pad" register in
;    DXYN, FX55 and FX65 Instructions
; R8 Timers controlled by Interrupt Routine (R8.1 is a generalpurpose   ----
;    timer: R8.0 is a tone and de-bounce timer)
; R9 Special Pointer and "Scratch Pad" used in Random Number      ----
;    Generally utilized in CXKK Instructions--changed by
;    Interrupt Routine
; RA I Pointer for CHIP 8 Instructions                  ----
; RB RB.1 is Display Page Pointer: RB.0 is "Scratch Pad" for      0Y--*
;    Interrupt Routine
; RC Temporary Pointer for FETCH AND DECODE Routine;            00--
;    Destination Address Pointer for DXYN Instructions; PC for
;    Keyboard Scanning Subroutine in FX0A Instructions.
; RD Both Sections Used as "Scratch Pad" Registers in DXYN        ----
;    (Display Instruetions)
; RE Pointer for Constants Needed in FX33 Instructions: RE.1        ----
;    is a "Scratch Pad" Register for DXYN (Display) Instructions
; RF Display Page Address Pointer for 00E0 (Erase) Instructions;    ----
;    RF.0 Used as "Scratch Pad" in FETCH AND DECODE
;    Routine and also in DXYN Instructions
;
;
; NOTE: Registers available for machine-language subroutines are R7, RC,
;    RD, RE and RF, but subroutines themselves must provide any initial
settings
;    required--CHIP 8 instructions may alter these register settings, as
indicated
;    above.
; *: In basic VIP system with 2K RAM, 0X = 07 and 0Y = 06. In general,
;    0X, is highest memory page and 0Y = 0X-1.
```