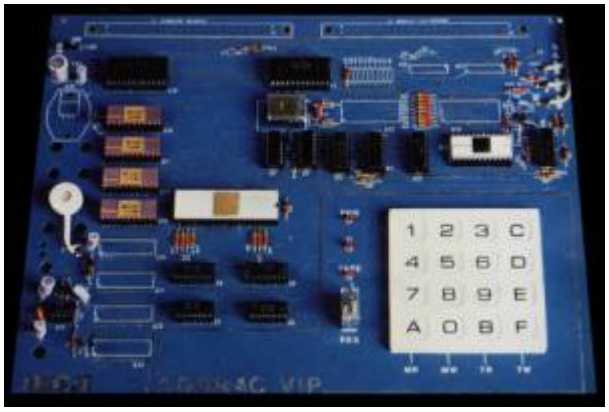
CHIP-8

Durch Zufall bin ich auf eine recht alte Programmiersprache namens CHIP-8 gestoßen.

CHIP-8 wurde von Joseph Weisbecker, Mitarbeiter der RCA Laboratories, Princetown, Mitte der 70er-Jahre entwickelt. Ursprünglich wurde die Sprache für Heimcomputer wie den COSMAC VIP oder den TELMAC konzipiert. Ziel war es, eine einfache Sprache zur plattformunabhängigen Entwicklung von Videospielen bereitzustellen.

COSMAC VIP

Der **COSMAC VIP** ist ein Einplatinenrechner, vergleichbar mit dem LC-80. Ein CDP-1802 8-Bit-Mikrokontroller, 2 KByte RAM, eine Hexadezimaltastatur, 1,76 MHz-Taktfrequenz, ein Kassettenrekorder dient zur Programmspeicherung. Unterschiedlich zum LC-80 ist die Anzeige. Während letzterer nur eine 6-stellige 7-Segment-Anzeige besitzt, hat der COSMAC VIP einen Grafikprozessor zum Anschluss an einen Fernseher. Es konnten 64×32 monochrome Pixel dargestellt werden.
Hergestellt wurde der COSMAC VIP bereits 1977, der LC-80 folgte erst 8 Jahre später!



Merkmal	Beschreibung
CPU	CDP-1802
ROM	0.5 KByte
RAM	2 KByte
Takt	1,76 MHz
Anzeige	TV, 64×32 Pixel, monochrom
Tastatur	16 Tasten (Hex)
Peripherie	Kassettenrekorder
Software	Programmiersprache CHIP-8

Links

- http://www.chip8.com Die größte Sammlung an Programmen und Infos zu CHIP-8, Handbuch des COSMAC VIP
- http://oldcomputers.net/rca-cosmac-vip.html Infos zum COSMAC VIP

- http://mattmik.com/documents.html die komplette Sammlung der VIPER-Magazine für die COSMAC VIP-Computer.

CHIP-8

CHIP-8 ist eine Maschinencode-Sprache für einen (theoretischen) 8-Bit-Prozessor. Es gibt nur 31(35) Maschinencode-Befehle; allerdings sind einige darunter, die das Programmieren von Telespielen besonders erleichtern, so z.B. eine Sprite-Ausgabe mit Kollisionserkennung oder bedingte Sprünge bei nicht-/gedrückter Taste.

Auf dem COSMAC VIP muss ein CHIP-8-Interpreter eingegeben werden. Dieser interpretiert dann den CHIP-8-Programmcode und führt so die Programme aus. 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. Auch der Zeichensatz ist clever komprimiert (s. Bild und VIPER 1-01).

Einen guten Einstieg in die Programmierung mit CHIP-8 liefert das Handbuch zum COSMAC VIP (RCA COSMAC VIP CDP18S711 Instruction Manual; 130 S.). Hier sind auch 20 Spiele abgedruckt, die mit CHIP-8 laufen (PONG, TIC-TAC-TOE, SNAKE, ...)

Im VIPER-Magazin 1-01 June 1978 werden zusätzliche Hinweise zur Programmierung mit CHIP-8 gegeben.

Ausgabe VIPER-Magazin 1-02 August 1978 liefert Informationen über die Arbeitsweise des CHIP-8-Interpreters. Es gibt Ablaufpläne und kommentierte Listings. → CHIP-8-Interpreter

http://mattmik.com/chip8.html beschreibt ausführlich den CHIP-8-Maschinencode.

virtueller Prozessor	16 Register V0..VF Index-/Adressregister I (Stackregister SP) (Program Counter PC) Delay Timer DT Sound Timer ST
Grafik	64×32 Pixel, Torus
Tastatur	4×4-Tastenfeld
Sound	Beepton

Der originale CHIP-8-Interpreter ist eine coole Software, umfasst nur 500 Byte, davon 1/3 für Sprite-Befehl! Es sind viele speicherplatzsparende Programmkniffe enthalten, ein Blick in den Quellcode lohn sich! z.B. Fx-Befehle, Bitmuster der Hexziffern

Die Virtuelle Maschine

Speicher

Die CHIP-8 Speicher-Adressen liegen im Bereich von 200h bis FFFh, das reicht für 3.584 Bytes. Der

Grund für den Speicher ab 200h ist, dass im VIP Cosmac und Telmac 1800 die ersten 512 Byte für den CHIP8-Interpreter reserviert sind. Auf diesen Maschinen wurden die obersten 256 Bytes (F00h-FFFh auf einem 4K-Maschine) für die Anzeige aktualisieren vorbehalten, und die 96 Byte unterhalb (EA0H-EFFh) wurden für den Call-Stack, den internen Gebrauch, und die Variablen vorbehalten.

Register

CHIP-8 verfügt über 16 8-Bit-Register V0..VF. Das VF-Register dient auch als Carry-Flag.

- 16 x 8-Bit-Register V0..VF
- 16-Bit-Index-/Adressregister I
- 8-Bit-Register Delay Timer DT
- 8-Bit-Register Sound Timer ST
- (16-Bit Stackregister SP)
- (16-Bit Program Counter PC)

Stack

Der Stack wird nur verwendet, um die Rückkehr-Adressen zu speichern, wenn Unterprogramme aufgerufen werden. Original ist Speicher für bis zu 12 Verschachtelungsebenen vorhanden.

Timer

CHIP-8 verfügt über zwei Timer. Beide werden automatisch mit 60 Hz dekrementiert, bis sie 0 erreichen. Delay Timer DT: Dieser Timer soll für das Timing der Ereignisse von Spielen verwendet werden. Sein Wert kann eingestellt und gelesen werden. Sound- Timer ST: Dieser Timer ist für Sound-Effekte gedacht. Solange der Wert ungleich Null ist, wird ein Piepton erzeugt.

Tastatur

Die Eingabe erfolgt mit einer Hex-Tastatur mit 16 Tasten von 0 bis F. '8', '4', '6 'und '2' dienen in der Regel als Cursortasten. Es gibt drei Opcodes zur Tastaturabfrage. Eine überspringt eine Anweisung, wenn eine bestimmte Taste gedrückt wird, eine weitere überspringt eine Anweisung, wenn eine bestimmte Taste nicht gedrückt wird. Die dritte wartet auf einen Tastendruck, und speichert die Taste dann in einem der Datenregister.

Grafik und Sound

Die Display-Auflösung beträgt 64 × 32 Pixel, und die Farbe ist einfarbig. Grafiken werden auf dem Bildschirm allein durch Sprites gezeichnet, die 8 Pixel breit sind und von 1 bis 15 Pixel hoch sein können. Sprite-Pixel, die gesetzt sind, invertieren die Farbe der entsprechenden Bildschirm-Pixel, während nicht gesetzten Sprite-Pixel nichts verändern.

Wenn beim Zeichnen des Sprites alle Bildschirm-Pixel invertiert wurden, wird das Carry-Flag (VF) wird auf 1 gesetzt, sonst ist es 0.

Wie zuvor beschrieben, wird ein Signalton abgespielt, wenn der Wert der Sound Timer ungleich Null ist.

Opcode Tabelle

CHIP-8 verfügt über 35 Opcodes, die alle zwei Byte lang sind. Das höchstwertige Byte wird zuerst gespeichert. Die Opcodes sind unten in hexadezimal und mit den folgenden Symbolen aufgelistet:

- mmm: Adresse
- kk: 8-Bit-Konstante
- n: 4-Bit-Konstante
- x und y: 4-Bit-Register

Hex	**Symbolisch**	**Beschreibung**
1mmm	GO mmm	Go to 0MMM
Bmmm	GO mmm+V0	Go to 0MMM + V0
2mmm	DO mmm	Do subroutine at 0MMM (must end with 00EE)
00EE	RET	Return from subroutine
3xkk	SKIP;Vx EQ kk	Skip next instruction if VX = KK
4xkk	SKIP;Vx NE kk	Skip next instruction if VX <> KK
5xy0	SKIP;Vx EQ Vy	Skip next instruction if VX = VY
9xy0	SKIP;Vx NE Vy	Skip next instruction if VX <> VY
Ex9E	SKIP;Vx EQ KEY	Skip next instruction if VX = Hex key (LSD)
ExA1	SKIP;Vx NE KEY	Skip next instruction if VX <> Hex key (LSD)
6xkk	Vx=kk	Let VX = KK
Cxkk	Vx=RND	Let VX = Random Byte (KK = Mask)
7xkk	Vx=Vx+kk	Let VX = VX + KK
8xy0	Vx=Vy	Let VX = VY
8xy1	Vx=Vx/Vy	Let VX = VX / VY (VF changed)
8xy2	Vx=Vx&Vy	Let VX = VX & VY (VF changed)
8xy4	Vx=Vx+Vy	Let VX = VX + VY (VF = 00 if VX + VY ⇐ FF, VF = 01 if VX + VY > FF)
8xy5	Vx=Vx-Vy	Let VX = VX - VY (VF = 00 if VX < VY, VF = 01 if VX >= VY)
Fx07	Vx=TIME	Let VX = current timer value
Fx0A	Vx=KEY	Let VX = hex key digit (waits for any key pressed)
Fx15	TIME=Vx	Set timer = VX (01 = 1/60 second)
Fx18	SND=Vx	Set tone duration = VX (01 = 1/60 second)
Ammm	I=mmm	Let I = 0MMM
Fx1E	I=I+Vx	Let I = I + VX
Fx29	I=Vx(LSDP)	Let I = 5-byte display pattern for LSD of VX
Fx33	MI=Vx(3DD)	Let MI = 3-decimal digit equivalent of VX (I unchanged)
Fx55	MI=V0:Vx	Let MI = V0 : VX (I = I + X + 1)
Fx65	V0:Vx=MI	Let V0 : VX = MI (I = I + X + 1)
00E0	ERASE	Erase display (all 0's)

Hex	Symbolisch	Beschreibung
DxyN	SHOW nMI@VxVy	Show n-byte MI pattern at VX-VY coordinates. I unchanged. MI pattern is combined with existing display via EXCLUSIVE-OR function. VF = 01 if a 1 in MI pattern matches 1 in existing display.
0mmm	MLS@mmm	Do 1802 machine language subroutine at 0MMM (subroutine must end with D4 byte)

Beispiel: Panzer

; Tank, from VIPER vol 1, issue 1 june 1978, pp. 14

```
200 6120  init: V1=20        ;initialize
202 6210    V2=10
204 A240    I=240
206 D127  show: SHOW 7MI@V1V2    ;show tank
208 6002  key:  V0=2        ;key wait
20A E0A1  key1: SKIP;V0 NE KEY
20C 1216        GO erase    ; 216
20E 7002    V0+2
210 300A    SKIP;V0 EQ 0A
212 120A    GO key        ; 20A    ;loop back to check next key
214 1208    GO key1     ; 208    ;loop back to recheck next key
216 D127  erase: SHOW 7MI@V1V2  ;erase tank
218 4002    SKIP;V0 NE 02    ;change x or y
21A 72FF    V2+=FF        ; = V2-1; move up
21C 4004    SKIP;V0 NE 04
21E 71FF    V1+=FF        ; = V1-1; move left
220 4006    SKIP;V0 NE 06
222 7101    V1+=01        ; move right
224 4008    SKIP;V0 NE 08
226 7201    V2+=01        ; move down
228 4002    SKIP;V0 NE 02    ; set pointer
22A A240    I=s_up        ;240
22C 4004    SKIP;V0 NE 04
22E A253    I=s_left    ;253
230 4006    SKIP;V0 NE 06
232 A24D    I=s_right       ;24D
234 4008    SKIP;V0 NE 08
236 A246    I=s_down    ;246
238 1206    GO show        ;206    ; jump to show

; Sprites
240 10    s_up:  db    00010000b    ; ...#....
241 54       db    01010100b       ; .#.#.#..
242 7C       db    01111100b       ; .#####..
243 6C       db    01101100b       ; .##.##..
244 7C       db    01111100b       ; .#####..
245 7C       db    01111100b       ; .#####..
246                                          ;
```

```
246 44    s_down: db    01000100b    ; .#...#..
247 7C       db    01111100b       ; .#####..
248 7C       db    01111100b       ; .#####..
249 6C       db    01101100b       ; .##.##..
24A 7C       db    01111100b       ; .#####..
24B 54       db    01010100b       ; .#.#.#..
24C 10       db    00010000b       ; ...#....
24D                                            ;
24D 00    s_right: db   00000000b    ; ........
24E FC       db    11111100b       ; ######..
24F 78       db    01111000b       ; .####...
250 6E       db    01101110b       ; .##.###.
251 78       db    01111000b       ; .####...
252 FC       db    11111100b       ; ######..
253                                            ;
253 00    s_left: db    00000000b    ; ........
254 3F       db    00111111b       ; ..######
255 1E       db    00011110b       ; ...####.
256 76       db    01110110b       ; .###.##.
257 1E       db    00011110b       ; ...####.
258 3F       db    00111111b       ; ..######
259 00       db    00000000b       ; ........

25A         END
```

Emulation

Der CHIP-8-Interpreter eignet sich aufgrund seiner einfachen Befehlssatzes ideal als Einstiegsobjekt in das Programmieren von Emulatoren. Es gibt mehrere Seiten im Netz, die das schrittweise vormachen.

- Listenpunkt

Und es gibt diverse Emulatoren, z.B. den VISION von M.Kogel.

- Listenpunkt

Z9001

Beim KC-Club Treffen 2013 habe ich einen Vortrag über die Programmiersprache CHIP-8 und die Implementation eines CHIP-8-Interpreters für den Z9001 gehalten.