C64 Visual Assembler — Felhasználói kézikönyv

Verzió: 2.7.8

Vizuális, blokk-alapú 6502 assembler a Commodore 64-hez. A programot utasításblokkok fogd-és-vidd módszerrel történő elrendezésével építed fel, a generált assembly és gépi kód pedig valós időben frissül.

> Ez a magyar fordítás az angol nyelvű **Visual Assembler Manual.md** alapján készült; hivatalos forrásnak az angol változat számít.

Tartalomjegyzék

- A 2.7.8 verzió újdonságai
- A 2.7.7 verzió újdonságai
- A 2.7.6 verzió újdonságai
- A 2.7.5 verzió újdonságai
- A 2.7.4 verzió újdonságai
- A 2.7.3 verzió újdonságai
- A 2.7.2 verzió újdonságai
- A 2.7.1 verzió újdonságai
- A 2.7.0 verzió újdonságai
- A 2.6.9 verzió újdonságai
- A 2.6.8 verzió újdonságai
- A 2.6.7 verzió újdonságai
- A 2.6.6 verzió újdonságai
- A 2.6.5 verzió újdonságai
- A 2.6.4 verzió újdonságai
- A 2.6.3 verzió újdonságai
- A 2.6.2 verzió újdonságai
- A 2.6.1 verzió újdonságai
- A 2.6.0 verzió újdonságai
- A 2.5.9 verzió újdonságai
- A 2.5.8 verzió újdonságai
- A 2.5.7 verzió újdonságai
- A 2.5.6 verzió újdonságai
- A 2.5.5 verzió újdonságai
- A 2.5.4 verzió újdonságai
- A 2.5.3 verzió újdonságai
- A 2.5.2 verzió újdonságai
- A 2.5.1 verzió újdonságai
- A 2.5.0 verzió újdonságai
- A 2.4.9 verzió újdonságai
- A 2.4.8 verzió újdonságai
- A 2.4.7 verzió újdonságai

- A 2.4.6 verzió újdonságai
- A 2.4.5 verzió újdonságai
- A 2.4.3 verzió újdonságai
- A 2.4.2 verzió újdonságai
- A 2.4.1 verzió újdonságai
- A 2.4.0 verzió újdonságai
- A 2.3.9 verzió újdonságai
- 1. A felület áttekintése
- 2. Blokk paletta
  - Paraméterek az operandus mezőben
- 3. Programterület
  - Blokk panel minimap
  - Operandus bevitel
- 4. ASM nézet
  - Kimeneti módok
  - Toolkit fül
  - Compile info és kimeneti beállítások
  - ASM sorra kattintás
  - ASM sorszámok
  - Fordítási folyamat modal
- 5. Beállítások és eszköztár
  - Verziókezelés
    - Projekt mód
    - Git
  - Frissítés az alkalmazásból
  - Ingyenes és nyílt forrású
  - Hibabejelentés
  - Támogatás
  - Munkaterületek
  - Memóriaablak és zero page foglaltság
  - Projekt sablonok
  - Tutorials és a lecke-szerkesztő
    - Online leckék
  - Build projekt
  - Példaprogramok
  - .asm fájl betöltése (gyorsreferencia)
    - Import parse-jegyzetek és jó gyakorlatok
  - Kick import (Kick Assembler)
    - Kick projektek (Kick projekt importálása…)
- UltimateBasic mód
  - A UB szerkesztő megnyitása
  - Szerkesztő eszközök
  - Projektek, fülek és startup fájlok
  - Buildelés és diagnosztika
  - Kódméret soronként és tippek
  - Futtatás, D64 és Exomizer
  - Debugger szimbólumok és disassembly
  - Ultimate Basic kézikönyv és forrás

- 6. Expert mód
  - Módváltás
  - Szerkesztő elrendezés
  - Eszköztár gombok
  - Expert szerkesztő minimap
  - Ciklusszámok
  - Optimalizálási tippek
  - Hibakiemelés
  - Szintaxiskiemelés
  - Forrásformázó
  - Projekt panel és fülek
  - Fül-sáv
  - ACME mód
- 7. Címzési módok
  - Címke-kifejezések operandusként
  - A `*` program counter kifejezésekben
  - Lokális (pontos) címkék
  - Önmódosító kód operandus-címkék
- 8. Szabványos 6502 utasítások
  - Data Movement
  - Arithmetic
  - Logic
  - Jumps & Branches
    - LBNE / LBEQ / … (Hosszú ugrások)
  - Register Operations
  - Shift & Rotate
  - Stack
  - System / Flags
  - Illegal / dokumentálatlan utasítások
- 9. Makró blokkok — referencia
  - LABEL
  - COMMENT
  - BYTE
  - WORD
  - FILL
  - ALIGN
  - TEXT
  - STRING
  - DATA
  - RAWBYTES
  - RAWTEXT
  - PETSCII
  - CHARSET
  - CHARDEF
  - BOX_HIT
  - INCBIN
  - KOALA
  - LEVEL / LEVEL_ROW

        - LEVEL
        - LEVEL_ROW
    - SID
    - SID_PLAYER
    - INCLUDE
    - TABLE
    - ORG
    - LOOP / NEXT
        - LOOP
        - NEXT
    - FOR / ENDF
        - FOR
        - ENDF
    - PUSH / PULL
        - PUSH
        - PULL
    - END / RTS alias
    - MACRO / ENDM / INVOKE
        - MACRO (definíció eleje)
        - ENDM (definíció vége)
        - INVOKE
    - REGION / ENDREGION
    - DEFINE / IF / ELSE / ENDIF
        - DEFINE
        - IF
        - ELSE
        - ENDIF
    - .ASSERT
    - CONST
    - VAR
    - Futásidejű IF / ELSE / ENDIF
    - WHILE / ENDW
    - REPEAT / UNTIL
    - MEMCPY / MEMSET
    - PRINT / PRINT_CHAR / PRINT_HEX / CLEAR_SCREEN / WAIT_KEY / DELAY / SET_BORDER / SET_BG
        - PRINT
        - PRINT_CHAR
        - PRINT_HEX
        - CLEAR_SCREEN
        - WAIT_KEY
        - DELAY
        - SET_BORDER / SET_BG
    - IRQ_SETUP
    - RAND
    - SPRITE_INIT
    - SPRITE_POS
    - WAIT_RASTER
    - JOYSTICK

    - MOUSE
    - SPRITE_COL
    - LOADFILE
    - EXODECRUNCH
    - REU_CHECK
    - REU_STASH / REU_FETCH / REU_SWAP
    - TURBO_SET
    - SUPERCPU_DETECT
    - TURBO_ENABLE
    - MAP_COPY
    - MAP_COPY16X16
    - SPRITE_ANIM
    - SCORE_BCD
- 10. Debugger integráció
    - RetroDebugger
    - Breakpoint blokkok
    - Debugger flag-ek
    - VICE Monitor (hot reload, élő változók, profiler)
    - Beépített C64 és debugger
- 11. Tudásbázis linkek
- 12. D64 export és futtatás
    - Osztott Run gomb
    - Export to D64 dialógus
    - D64 metaadat a projektekben
    - D64 Editor (meglévő lemezkép böngészése és szerkesztése)
- 12b. CRT export (Magic Desk 64K cartridge)
    - Mi kerül a cartba
    - Betöltési cím és belépési pont
    - Méretkorlát
    - Kompatibilitás
- 13. Beállítások
    - VMA fordító
    - VICE emulátor
    - Exomizer
    - Retro Debugger
    - C64 Ultimate / 1541 Ultimate
    - Az Ultimate emulátor telepítése lépésről lépésre
    - Kung Fu Flash 2
    - Verziókezelés (git)
- 14. Vizuális szerkesztők (Toolkit)
    - Hi-Res / Multicolor Editor
    - Sprite Editor
    - C64 karakter-ROM böngésző („Char map")
    - Character Editor (Charset)
    - Charset Canvas Editor
    - Map Editor (több-rétegű tilemap-ek)
    - Pálya szerkesztő (Level Editor)
    - SID Editor (3-voice tracker)

    - SID Player
        - HVSC (High Voltage SID Collection)
        - Spektrum-analizátor
        - Csatornagörbék
    - Curve Editor
    - Táblagenerátor
    - Számrendszer-átváltó

A 2.7.8 verzió újdonságai

- **Gyorsabb lett** — a fordítás, a futtatás és gépelés közben az élő nézetek is gyorsabbak: a natív Rust fordító egyszer fordítja a programot, és minden nézet ebből az egy fordításból dolgozik.
- **Egyetlen fordító** — minden programot a Rust VMA fordító (VMA 2.3.0) fordít, és a disassembly, a monitor, a memóriatérkép, a ciklus-gutter, a tippek és az ASM nézet ugyanebből a fordításból dolgozik. A JavaScript fordító és a **VMA fordító** beállítás kikerült; a fordításhoz az asztali alkalmazás kell. Lásd: VMA fordító.
- **Lokális címkék abszolút operandusként** — egy másik hatókör vagy eltolással írt lokális címke, pl. `JMP Second.lp` vagy `LDA .lp+1,X`, lefordul; korábban érvénytelen operandus volt.

A 2.7.7 verzió újdonságai

- **Ultimate 64 paletta makrók** — a `.u64_palette "fájl.vpl"`, a `.u64_palette_color index, $RRGGBB` és a `.u64_palette_reset` az Ultimate 64 / C64 Ultimate videó-palettáját állítja a Command Interface-en át. Lásd: U64_PALETTE / U64_PALETTE_COLOR / U64_PALETTE_RESET.
- **Ultimate 64 paletta demó** — új minta a Hardver kategóriában: az 1 és 2 billentyű két palettafájlt tölt, a 3 két színt cserél, a 0 visszaállítja a menü palettáját. Valódi gépen vagy az Ultimate emulátor futtatási módban fut.
- **Hi-Res MIX** — keverék-toll multicolor képekhez: 8 hely, helyenként 3 szín, 8 minta a képhez rögzítve. Lásd: Hi-Res / Multicolor Editor.
- **Beépített C64** — a Futtatás üres programot (csak ORG, megjegyzés vagy üres sor) már nem indít el, mint a többi futtatási mód.
- **Fordító beállítás** — a **Rust + JavaScript ellenőrzés** mód kikerült a Beállítások → Általános → Program alól; a JavaScript (alapértelmezett) és a Rust marad.
- **Minták fájlokkal** — a minta saját `.incbin` / `.koala` / `.u64_palette` / `.sid` fájljai vele együtt betöltődnek.

A 2.7.6 verzió újdonságai

- **Csillagos értékelés** — a SID Player lejátszási listájának minden sora 1–5 csillagot kapott az egyetlen kedvenc-jel helyett. A **Csak a csillagosak maradjanak** megerősítés után törli a többit. Lásd: SID Player.
- **Duplikátumok törlése** — a SID Player új eszköztár-gombja törli a listában többször szereplő számokat (azonos HVSC-útvonal, vagy azonos fájl). Az első példány marad, a példányai közül a legtöbb csillaggal.
- **Torzításmentes hangerő-kiegyenlítés** — ha a halk kezdetű dalt felerősítette, és hangosabb rész jön, a szint most azonnal, a rész megszólalása előtt csökken, ahelyett hogy két másodperc alatt csúszna le, miközben a hangos rész

eleje torzult.

- **HVSC letöltés** — a még töltődő dal **+** gombja helyén forgó jel látszik, és az újabb kattintás nem veszi fel másodszor.
- **Restore64 források saját BASIC sorral** — a visszafejtés, amely a `10 SYS 2061` sorát a kód után, az adatblokkok között írja a `$0801`-re, ismét bájtra az eredetivel azonosra fordul. Eddig a sor a kód első 12 bájtjára került, és a program nem csinált semmit. Lásd: Kick import.

A 2.7.5 verzió újdonságai

- **Csatornagörbék** — a SID Player és a SID Editor közös ablaka hangonként görgő szintgörbét és nagy hangnevet mutat. Lásd: Csatornagörbék.
- **Spektrum csatornánként** — a Spectrum ablak **View** listája a kimenet keveréke (**Mix**) és a hangonkénti külön analizátor (**Channels**) között vált. Lásd: Spektrum-analizátor.
- **Belső zeneszámok egymás után** — a SID Player a fájl hátralévő subtune-jait is lejátssza, mielőtt továbblép, véletlen lejátszásnál is. Az új eszköztár-gomb kapcsolja, alapból be van kapcsolva.

A 2.7.4 verzió újdonságai

- **Kijelölés a vízszintes trackerben** — Shift+húzással kerettel vagy Ctrl+kattintással jelölhetők ki a hangok, majd a fejléc új gombjaival vagy Ctrl+X / C / V és Del billentyűvel kivághatók, másolhatók, beilleszthetők (az egér vagy a kurzor helyére) és törölhetők.
- **Kvantálás** — a **Q** vagy az új gomb a kijelölt hangokat a rács legközelebbi sorára teszi (2, 4, 8 vagy 16 sor, a tracker eszköztárán), a hosszuk megmarad. Bekapcsolt **Felvételnél** mellett a felvétel a leállításkor kvantálódik.
- **Visszavonás / Mégis a vízszintes trackerben** — a tracker eszköztárának két gombja a fejlécében is megvan.
- **Tisztább szerkesztés** — egy hang törlése után az előtte lévő hang megtartja a végét, és egy hosszabb hangra illesztés nem hagy maga után kóbor note-offot.
- **Hangszerek törlése** — a **+** melletti **−** törli az aktuális hangszert, a Files → **Nem használt hangszerek törlése** a hang nélkülieket.
- **A dal hangereje** — a tracker eszköztárának **Hangerő** tekerője a dal hangereje (két SID-nél chipenként egy); a hangszerek már nem állítják át a chip hangerejét.
- **PAL / NTSC** — a dal időzítése: NTSC-n 60 frame másodpercenként az NTSC órajelén, saját hangjegytáblával és `.sid` fejléccel.

A 2.7.3 verzió újdonságai

- **Linux** — a SID Player és a SID szerkesztő néma maradt, ahol hiányoztak a GStreamer-pluginek: az AppImage már tartalmazza őket, a `.deb` és `.rpm` csomag pedig függ tőlük.
- **Visszavonás / Mégis a SID szerkesztőben** — két új gomb a tracker eszköztárán, vagy Ctrl+Z és Ctrl+Y. A hangokra, hangszerekre, a songra, a tempóra és a dal betöltésére is érvényes, legfeljebb 100 lépésig. Egy teljes felvétel egy lépés.
- **Tempó** — a tracker tempómezőjének neve most az, ami valójában: **Frame/sor**, a nagyobb szám lassabb. A mellette lévő **BPM** az ebből adódó tempót mutatja.

- **Hangok mozgatása a vízszintes trackerben** — a hang törzsét húzva a hang időben és hangmagasságban is mozog. Megtartja a hosszát, a hangszerét és az effektjét, és egy sorral a sáv következő hangja előtt megáll.
- **Hanghossz a vízszintes trackerben** — a hang jobb vége könnyen megfogható, egysoros hangnál is: a kurzor dupla nyíllá válik, a hang végét világos sáv jelzi. Húzással a hang nyújtható vagy rövidíthető, és a hang megáll a sáv következő hangjánál, nem törli azt.
- **MIDI csatlakoztatása kérdés nélkül** — Windowson a webnézet már nem kér engedélyt külön párbeszédablakban.
- **.bin betöltése** — a tracker a kijelölt első song-lépésen nyílik meg: ha a dal nem a 0. patternnel kezdődött, eddig a 0. pattern látszott.
- **SID Player szünet** — a Pause mostantól mindig megállít: az épp elkészült zenerészlet vagy a SID szerkesztő egy hangja eddig visszakapcsolhatta a hangot, miközben a lejátszó szünetet mutatott. A Pause újbóli megnyomása folytatja a lejátszást.
- **Stop a „Play: All" módban** — a tracker visszaáll a lejátszás előtt megnyitott patternre, és nem marad az utoljára játszotton, miközben az első song-lépés van kijelölve.
- **Polifon felvétel MIDI-ről** — bekapcsolt **Polifon felvétel** mellett (a MIDI bemenet mellett) a felvétel alatt lenyomva tartott billentyűk külön-külön hangra kerülnek, a kijelölt hangtól kezdve. A háromhangú akkord három hangra kerül, a billentyű felengedése note-offot ír.
- **Nincs pattanás reSID-en** — sem a billentyűzetről vagy a VA-VK-ról játszott hangok előtt, sem a nem az első sorban kezdődő dal első hangjánál, sem 6581-gyel, sem 8580-nal.

A 2.7.2 verzió újdonságai

- **reSID a WebSID helyett** — a SID Player és a SID szerkesztő a backenden szól, a resid-rs motoron (a reSID Rust portja, ez az alapértelmezett) vagy a VA SID Core-on, mindkettő MOS 6581 vagy 8580 chippel. A **Beállítások** → **Általános** → **SID előnézet** alatt választható. A tekerés, a sztereó 2SID és 3SID dalok és a C64 mag mindkettővel működik. A WebSID kikerült.
- **SID Player puffer** — a **Beállítások** → **Általános** → **SID előnézet** → **SID Player puffer** már azt mutatja, mennyire előre renderelődik a dal: 10, 15, 20 vagy 30 mp.

A 2.7.1 verzió újdonságai

- **Debugger VICE-hoz** — a **Beállítások** → **Általános** → **Debugger** választja ki a Debug gomb debuggerét: a **Visual Assembler** a futtatási mód gépén keres hibát (VICE, Ultimate emulátor, beépített C64), a **RetroDebugger** a külső programot nyitja. VICE-ban az ablak a saját VICE-odon dolgozik: töréspontok, lépés be / át / ki, regiszterek, disassembly és memória, a szerkesztő követi a CPU-t. Lásd: Beépített C64 és debugger.
- **Ultimate emulátor** — két új futtatási mód, az **Ultimate emulátor** és a **D64 az Ultimate emulátoron**, a programot az UE2-C64U-Emulator emulátoron futtatja, amely a változatlan C64 Ultimate / Ultimate 64 firmware-t futtatja a számítógépeden, ugyanazon a REST API-n át, mint a valódi eszközön. A VA az első futtatáskor indítja az emulátort, a `.ue2` firmware-t flash-képbe telepíti, és az appal együtt zárja be az emulátort. Ezekben a módokban a **Debug** debuggert nyit (töréspontok, lépés be / át / ki, regiszterek, disassembly, memória), a **VICE Monitor** ablak pedig az emulátorhoz kapcsolódik. Beállítás: **Beállítások** → **Integrációk** → **Ultimate emulátor (UE2)**. Új példák: az **Ultimate turbó demo (64 sprite)** és az **Ultimate turbó kocka** (Példák → IRQ és hardver) 3 másodpercenként váltja a CPU sebességét a `$D031`-gyel 1 és 64 MHz között, és csak bekapcsolt turbó-regiszterű Ultimate-en fut. Lásd: C64 Ultimate / 1541 Ultimate.
- **SID Editor: két SID** — a tracker eszköztárán a **SID: 1 / 2** 6 hangosra váltja a dalt, a második SID címe választható (`$D420` – `$D7E0`, `$DE00` – `$DFE0`). Az előnézet, a `.bin` mentés, a `.sid` és az assembly export, valamint a

`.sid` import is kezeli mindkét chipet. Lásd: SID Editor.

- **SID Player: 2SID és 3SID dalok** — a VA SID Core sztereóban játssza őket: SID 1 balra, SID 2 jobbra, a harmadik SID középre. Lásd: SID Player.
- **Csatorna-spektrum** — a SID Editorban tracker-lejátszás közben minden hangnak külön, 16 sávos analizátora van.
- **SID Editor** — **Files** → **New tune**, a dal címe a fejlécben, egy gomb a tracker kilógatásához és visszatételéhez, valamint rendezettebb hangszerlista a vízszintes trackerben.
- **Javítások** — a **D64 on hardware** a csatolási kérésben elküldi a lemezkép típusát, amit a C64 Ultimate 1.1.0 firmware megkövetel; a szerkesztő leállítása már nem zárja be a SID Playert.

A 2.7.0 verzió újdonságai

- **Beépített C64** — a VICE 3.10 (x64sc, WebAssembly-re fordítva) mostantól az appon belül fut, nem kell telepíteni. Válaszd a **Beépített C64** pontot a Run menüben, vagy nyisd meg az eszköztár gombjával. Lásd: Beépített C64 és debugger.
- **Debugger** — a töréspontnál az utasítás előtt áll meg; **Folytatás**, **Megállítás**, **Lépés be / át / ki**; szerkeszthető regiszterek és flagek, disassembly a PC körül, memórianézet és töréspontlista. A töréspontok a szerkesztőből, Block módból és a disassemblyből jönnek, a szerkesztő követi a CPU-t. A **Debug** gomb ebben a módban ezt használja, és a valódi hardveres módokban is, ha a **Visual Assembler** a választott debugger.
- **Gép beállításai** — PAL vagy NTSC, SID 6581 vagy 8580, reSID vagy FastSID, REU 16 MB-ig, joystick port, warp és hang.
- **Saját fájlok** — a **Files** menü PRG-t tölt be, D64-et és cartridge-et (`.crt`) csatol. A státuszsor mutatja a meghajtó LED-jét és a tracket, a villogó LED lemezhibaként jelenik meg, például ha egy fájl nem található.
- **Kép** — CRT szűrő scanline-okkal, teljes képernyő és átméretezhető ablak. Az ablak és a beállításai a következő indításkor visszatérnek.
- **Hi-Res szerkesztő** — a multicolor cellákban a papír mellett legfeljebb 3 szín lehet, a korlátot túllépő cellák piros keretet kapnak, és a kép szélén túlra húzás bekapcsolt pixelrácsnál már nem ad hibát.
- **Jobb klikkes menü a szerkesztőkben** — az Expert és az Ultimate Basic szerkesztőben: Kivágás, Másolás, Beillesztés, Az összes kijelölése, valamint NAGYBETŰS / kisbetűs a kijelölt szövegre.

A 2.6.9 verzió újdonságai

- **Git ágak** — a Verziókezelés ablak ágat vált (az ágválasztó listával), ágat hoz létre, nevez át és töröl, és egy ágat az aktuálisba olvaszt (merge). Ütközésnél kiírja és pirossal jelöli a fájlokat, a **Merge megszakítása** visszaáll a merge előtti állapotra; ágváltás, merge és stash után a nyitott fülek újratöltődnek a lemezről. Lásd: Git.
- **Fájlonként és stash** — minden változott fájlnak saját **Stage / Unstage / Elvetés** gombja van (a nem követett fájl törölhető), a nevére kattintva megnyílik a diffje. A **Stash** félreteszi a változásokat; a stash-listából alkalmazható, visszavehető (pop) vagy törölhető. A **Minden változás stage-elése** nélkül a projekt-commit csak a stage-elt fájlokat viszi.
- **Szebb diff** — fájlonkénti kártya állapottal és +/− számokkal, régi és új sorszám, assembly / Ultimate Basic szintaxisszínezés, a sor megváltozott része kiemelve; a **Kétoldali** gomb egymás mellé teszi a régi és az új szöveget, a ⤢ nagyobb helyet ad.
- **A Változások elvetése tényleg elvet** — Git módban előbb rákérdez, aztán a lemezen is visszaállítja a fájlokat a HEAD-re (`git restore`), és újratölti a füleket, így utána ágat válthatsz. A menüből indítva az eredménye üzenetként jelenik meg.

- **Kick projektek** — a ciklusszám, a hibák és a tippek az indítófájl által nem importált fájlban is működnek: abból a fából jönnek, amelyikhez a fájl tartozik (pl. `main.asm` egy `boot.asm` indítófájl mellett). A `.pseudopc`-n belüli előre mutató ág (`bne !+`) a logikai címhez képest relatív, a karakterliterál pedig egyenlő az egykarakteres sztringgel (`s.charAt(i) == '1'`), ahogy a KickAssemblerben.
- **Javítások és megjelenés** — a **Save project** a megnyitott Expert projektet a helyére menti, a mentetlen fájl-füleivel együtt, fájlnév kérése nélkül; a Verziókezelés ablak a képernyőn belül marad, és a szerkesztőkéhez hasonló ikonos eszközgombokat kapott; rövidebb számmezők a Táblagenerátorban.

A 2.6.8 verzió újdonságai

- **Toolkit Expert és UB módban** — az eszköztár *Toolkit* gombja a szerkesztő mellett mutatja a C64 palettát és a PETSCII kódokat: Expert módban oldalpanelként a Disassembly / Monitor mellett, UB módban a jobb oldali panelen (a Disassemblyvel és a Monitorral váltja egymást).
- **Megjegyzett panelek** — az Expert mód Monitor és Toolkit panele a Disassemblyhez hasonlóan méretezhető; a nyitva hagyott panelek (Paletta, Disassembly, Monitor, Toolkit, Projekt) és a szélességük a következő indításkor visszaállnak, bármelyik módban indul az app — UB módban is.
- **VMA nyelv** — a zero page-en lévő `.const` vagy `.var` miatt az `lda zp` zero page utasítás lesz (`A5 FB`, mint a KickAssemblerben; az `lda.abs zp` abszolút marad); `JMP (címke)` / `JMP (table+2)`; kifejezések, mint az `lda $1234+1` és a `sta $0400+40*2,x`; makró hívhat makrót; címkék a `.data` / `.rawbytes` bájtjaiban (pointertáblák); a `:név` címke mutathat címkére.
- **Include-ok** — a beemelt fájl saját `.include`, `.incbin` és `.sid` sorai is betöltődnek (a saját mappájához képest), a makróhívásai kibomlanak; a hiányzó vagy önmagát beemelő include hiba, nem üres könyvtár.
- **Biztonságosabb fordítás** — csendesen rossz kód helyett hiba: feloldatlan TURBO_SET / SPRITE_ANIM érték, később definiált címkéből vett MEMCPY / MEMSET / JOYSTICK méret; a `.data` / `.rawbytes` makró-argumentum automatikus puffere lefoglalja a bájtjait. A zero page konstansokat használó programok rövidebbek lesznek.
- **Javítások** — a 8 KB-nál nagyobb (több bankos) CRT nem indult el: a loader most RAM-ból fut; a projekt mentés-dialógusában megadott `.asm` név a forrásszöveget menti, nem a VA JSON-t.

A 2.6.7 verzió újdonságai

- **Rust VMA fordító** — a **Beállítások** → **Általános** → **Program** → **VMA fordító** választja a motort: **JavaScript** (alapértelmezett), **Rust**, vagy **Rust + JavaScript ellenőrzés**, amely mindkettővel fordít, és naplózza az eltérést. A Rust fordító ugyanazokat a bájtokat, memórianézetet, szegmenseket és lefordított hibaüzeneteket adja; a Kick projektek is rajta át fordulnak, a hibák `fájl:sor` alakban. Lásd: VMA fordító.
- **A VMA nyílt forráskódú** — a Visual Macro Assembler fordító saját GPLv3 tárolóban van: github.com/zstarczali/VMA (2.0.0), angol és magyar nyelvi kézikönyvvel és parancssori fordítóval. Az About ablak az Ultimate Basic mellett a VMA verzióját és logóját is mutatja.
- **Projekt szimbólumok és konstansok** — a **Projekt beállítások** a teljes projektre ad VMA szimbólumokat (`.define`, az `.if` / `.ifdef`-hez) és konstansokat (`.const`), a `.proj`-ba mentve; a hibák sorszáma helyes marad. Lásd: Projekt panel és fülek.
- **Fordítás ablak** — a folyamatjelző alatti sor kiírja a motort, a fájlt és a PRG méretét.
- **Build info** — kiírja, melyik fájlról szól; ACME forrásnál a valódi ACME fordításból készül (kezdőcím, vég, méret, szegmensek, címkék). Mellékattintásra már nem zárul be.

- **Gyorsabb gépelés** — ACME módban a Disassembly és a Monitor panel fordításonként egyszer épül újra, és halványan látszik, amíg a forrás újabb; a nagy Kick projektek újrahasználják az utolsó előfeldolgozást, és részletekben rajzolják a disassemblyt.
- **Projektfa** — a fájlok és mappák jobb klikkes menüjében **Fel** / **Le** mozgatás. A Kick-kompatibilitás a **Projekt beállítások**-ban kapcsolható.
- **Javítások** — az `.asm` mentése (Ctrl+S, Menu → Mentés, git commit előtt) ismét sima szöveget ír, nem VA JSON-t; a verziókezelés pillanatképei a forrásfájlokat szövegként tárolják; a felhasználói makróban hívott könyvtári makrók (SET_BORDER, SPRITE_*, JOYSTICK, TEXT …) megkapják a paramétereiket, a futásidejű IF / WHILE / REPEAT címkék hívásonként egyediek, és az `'A'` argumentum karakter marad; a mozgatható ablakok a mentett helyükön nyílnak.
- **Megjelenés** — a fordítás, a Build info, a D64 export, a fordítási hiba és a kérdés ablaknak az About keretét kapta.

A 2.6.6 verzió újdonságai

- **SID lejátszó makró** — a `.sid_player [subtune][, raster]` (a palettán **SID_PLAYER**) lejátssza a `.sid` makró zenéjét: meghívja az init rutint a subtune-nal, kikapcsolja a CIA1 időzítő IRQ-t, beköt egy raszter IRQ-t, és a beépített handleréből képkockánként meghívja a play rutint. Egy zenélő program most ennyi: `.sid "zene.sid"` + `.sid_player`. Lásd: SID_PLAYER.
- **SID címkék** — a `.sid` makró megadja a `SID_LOAD`, `SID_INIT` és `SID_PLAY` címkét (egyedi betöltési címnél azzal együtt eltolva), így a saját IRQ kezelőd is írhatja: `JSR SID_PLAY`. Lásd: SID.
- **Csak Expert mód** — a **Projekt beállítások** kapcsolója a projekt fájljain kikapcsolja a blokk módot, méretüktől függetlenül (a `.proj`-ba mentődik). Lásd: Projekt panel és fülek.
- **SID Player puffer** — a **Beállítások** → **Általános** → **SID előnézet** → **SID Player puffer** 2048–16384 mintára állítható; nagyobb pufferrel nem recseg a hang, ha az app épp dolgozik. A WebSID újraindítás után használja, a VA SID Core 10 / 15 / 20 / 30 mp-et renderel előre. Lásd: SID Player.
- **Egyéni C64 paletta** — a **Beállítások** → **Általános** → **Szerkesztők** → **Egyéni paletta szerkesztése…** alatt mind a 16 szín RGB értékét magad adhatod meg, bármelyik beépített palettából kiindulva; Photoshop `.act`, VICE `.vpl`, GIMP `.gpl` paletta vagy hex színlista importálható, `.act` / `.vpl` exportálható. Lásd: 14. Vizuális szerkesztők (Toolkit).
- **Megjelenés** — a C64 memóriapanel összecsukva dokkol vissza; a What's New, a frissítés, a Beállítások és a Knowledge Base ablaknak ugyanolyan kerete van, mint az About-nak; a Projekt beállítások checkboxai VA kapcsolók.
- **Javítások** — a hibabejelentő a kiadott buildben is elfogadja a PNG és JPG képernyőképet; a nyitóképernyő `READY.` utáni villogó kurzora tömör C64 cella, a macOS által rárajzolt vonalak nélkül; macOS-en nincs prediktív szöveg az Expert és az Ultimate Basic szerkesztőben.

A 2.6.5 verzió újdonságai

- **KickAssembler projektek** — a **Menu** → **Program** → **Kick projekt importálása…** egy teljes KickAssembler forrásfát (a fő fájlt az `#import` moduljaival és `.import binary` assetjeivel) projektté tesz: mellé ír egy `.proj`-ot, a források a helyükön maradnak. A Visual Assembler a saját assemblerével fordítja, bájtra úgy, mint a KickAssembler 5.25 — a Futtatás, a Build PRG / D64 / CRT, az Exomizer, a Debug, a VICE Monitor és a Build project is működik, a hibák `fájl:sor` alakúak, és gépelés közben a teljes fa ellenőrződik. A Dark Chambers C64 port és a KickAssembler mind a tizenegy kódot tartalmazó példaprogramja azonosan fordul. Lásd: Kick projektek (Kick projekt importálása…).

- **Projekt beállítások** — a projektpanel fogaskereke (vagy **Menu** → **File** → **Projekt beállítások…**) megmutatja és szerkeszti a projekt nevét, indítófájlját, assemblerét, KickAssembler-kompatibilitását, `#define` szimbólumait és `cmdLineVars` változóit. Lásd: Projekt panel és fülek.
- **Kick-kompatibilis kapcsoló** — egy kapcsoló (2.6.7 óta a **Projekt beállítások**-ban) bármely VMA projektet KickAssembler-kompatibilissé tesz; a VA makrók és direktívák a KickAssembler szkript mellett is működnek.
- **KickAssembler nyelv** — `#import` / `#importonce` / `#importif`, `#define` / `#if`, `.const` / `.label` (zero page vagy abszolút az értéktől függően), `.for`, `.while`, `.if … else`, `.var` / `.eval`, `.function`, `.macro { }`, `.pseudocommand`, többszörös címkék, scope-ok és névterek, `.pseudopc`, `.import binary`, `LoadBinary` (template-ekkel, pl. `BF_KOALA`), `LoadSid`, `virtual` memóriablokk, `cmdLineVars`, `BasicUpstart2`, `List` / `Hashtable` / `.struct` / stringek, `.enum`, a matematikai könyvtár, a szín- és opkód-konstansok, a `*` mint programszámláló, `;` az utasítások között, `.error` / `.errorif`.
- **Build Info több oszlopban** — szélesebb ablak: fent a kezdő- és végcím, a méret és a szegmensek száma, alatta a szegmensek, a hibák és a `.print` kimenet, a címkék, konstansok és makrók oszlopokban. Kick projektnél a lefordított fát mutatja, a belső címkéket `scope.címke` alakban.
- **C64 színpaletta** — a **Beállítások** → **Általános** → **Szerkesztők** → **C64 színpaletta** adja meg a 16 C64 szín RGB értékét minden szerkesztőben (sprite, karakter, Charset Canvas, Hi-Res, térkép, pálya, Toolkit színminták): alapból **Colodore** (a valódi C64-hez legközelebbi), vagy Pepto, VICE, Deekay, C64 Wiki / Godot. Csak a megjelenítés változik; a Hi-Res képimport is ezt használja. Lásd: 14. Vizuális szerkesztők (Toolkit).
- **Színválasztó** — a Hi-Res, Sprite, Karakter (multicolor) és Charset Canvas szerkesztőben pipetta eszköz; az **Alt+kattintás** bármelyik eszközzel színt vesz fel. Lásd: Hi-Res / Multicolor Editor.
- **Ctrl + görgő nagyítás** a Hi-Res, Sprite, Map és Level szerkesztőben, a kurzor körül. A Hi-Res Raster kapcsolója a Grid mellé került.
- **ACME mód** — a memóriatérkép és az átfedés-jelzés az ACME fordítást követi; az ACME projekt az indítófájlját fordítja; az ACME sablonok csak működő ACME mellett jelennek meg, és ACME projekt létrehozása a Beállításokat ACME módra állítja. Lásd: ACME mód.
- **Új projekt** — az **Új projekt sablonból…** a menü **Új projekt** része alá került (a három külön menüpont megszűnt); a Visual Assembler sprite multiplexer sablon kiírja a címét.
- **Javítások** — a `.byte` / `.word` címke-kifejezéssel (`<(címke+1)`, `.word címke` a címke előtt) már nem 0-ra fordul; az operandust olvasó illegális `NOP` alakok fordulnak (lásd: Illegal / dokumentálatlan utasítások); az Ultimate Basic megnyitás-gombja minden nyelven **.ub fájl betöltése**.

A 2.6.4 verzió újdonságai

- **ACME assembler mód** — a **Beállítások** → **Általános** → **Assembler mód** a *VMA mód* (a Visual Assembler beépített assemblere) és az *ACME mód* között vált: az Expert forrást az ACME 0.97 fordítja, és a Futtatás, a Build, a Disassembly / Monitor panel, a debugger-szimbólumok, a ciklusszámok, az optimalizálási tippek és a VICE Monitor is a valódi ACME kimenetet használja. Lásd: ACME mód.
- **Jobb Kick Assembler import** — a Program menü gombja mostantól **Kick import**. A sima `.text "…"` sorok az `.encoding`-ot követik, a `.label NÉV = * + N` utasítás közepére tesz címkét, az `.abs` / `.zp` mnemonik-utótag kikényszeríti a címzési módot. A Restore64 visszafejtései bájtra ugyanúgy fordulnak, mint KickAssemblerrel. Lásd: Kick import (Kick Assembler).
- **A BASIC sor alá töltődő programok is elindulnak** — ha a program a BASIC SYS sor alá vagy rá töltődik, a Futtatás kis indító töltőt tesz elé; a forrásban később álló, alacsonyabb ORG már nem kerül fel a kezdőcímre. Lásd a **BASIC SYS stub** sort a Beállítások és eszköztár szakaszban.
- **Exomizer** — a tömörített program második RUN-ja már nem ad `?SYNTAX ERROR`-t: a BASIC sorral kezdődő PRG `sfx basic`-kel, minden más `sfx $<betöltési cím>`-mel tömörül.

- **SID szerkesztő** — lejátszási LCD a tracker eszköztárán (song-pozíció, pattern, sor, idő; zöld vagy piros a **Beállítások** → **Általános** → **SID előnézet** → **LCD színe** alatt), hangszer-tekerőgomb LCD-vel a VA-VK-n, folyamatjelző az **Import .sid...**-hez. Lásd: SID Editor.
- **A Beállítások három fülön** — **Általános**, **Fájlok és útvonalak**, **Integrációk**. Lásd a 13. szakaszt.
- **Apróságok** — a **Ctrl+N** új fület nyit; a dialógusok szövege véletlenül sem jelölhető ki (a szövegmezők és a másolható kimenetek, pl. a fordítási hibák, továbbra is); a nagy Expert források szerkesztése gyorsabb; az `LSR A` és a `JMP ($nnnn)` importja helyes; a Tudásbázisban ott a Restore64 (restore64.dev), amely C64 PRG-ből KickAssembler / ACME forrást készít a **Kick import**hoz vagy az ACME módhoz. A `$0801` alatti szegmenst tartalmazó ACME forrás (memória-dump) már nem tolódik a BASIC SYS sor mögé: a saját címein fordul, és indító töltőt kap.

A 2.6.3 verzió újdonságai

- **.sid import a SID szerkesztőben** — a **Files** → **Import .sid...** bármely PSID / RSID dalból szerkeszthető hangszereket, patterneket és songot csinál: hangkezdetek és legato hangok, a hangokra pontosan illesztett sorok (funktempo és swing `F` sebességváltással), wave / arpeggio táblák fix magasságú dob-attackkal, pulzus- és szűrőmozgás, gate-időzítés és vibrato. Lásd: SID Editor.
- **A hosszú dalok helyesen szólnak** — ha a dal adata nem fér el a `$C000`-s lejátszó mellett, külön szegmensbe kerül `$A000` alá, és nem lóg bele az I/O-ba; a `.sid` export is ezt követi.
- **A VA SID Core előnézet újraindítja a hangokat** — a tartott gate mellett újraindított hang új attackot kap, mint a valódi lejátszóban.
- **SID szerkesztő** — a fejléc mutatja a használt SID-emulációt (WebSID vagy VA SID Core a chip-modellel); a Speed mezőn − / + gomb van, értéke 1–255; a song legfeljebb 256 patternt használhat.
- **Frissítési szerver beállítása** — a **Beállítások** → **Általános** → **Frissítés** adja meg, melyik szerverről olvassa a kiadásokat az alkalmazáson belüli frissítés (alapértelmezés `https://www.c64va.pro/`). Lásd: Frissítés az alkalmazásból.
- **Ultimate Basic 1.6.6** — `PRINT USING "formátum"; értékek`, sztring-összehasonlítás PETSCII sorrendben, lebegőpontos `rnd` és `mod`, lebegőpont IRQ / NMI kezelőkben, `option basic print` a C64 BASIC PRINT-térközhöz, kisebb programok (csak a használt lebegőpontos rutinok kerülnek bele) és 16 bites címkifejezések (1.6.5). Az Ultimate Basic szerkesztő kiemeli a `print using` és az `option basic print` szavakat.

A 2.6.2 verzió újdonságai

- **HVSC a SID Playerben** — keresés a High Voltage SID Collection mintegy 61 000 dalában cím, zeneszerző vagy mappa szerint; lejátszás, lejátszási listára vétel vagy mentés; a teljes gyűjtemény egyben letölthető offline használatra, vagy megadható egy már kibontott HVSC mappa. A HVSC-ből érkező dalok a valódi hosszukkal jelennek meg a pozíciósávon, és a lista a dal végén lép tovább. Lásd: HVSC (High Voltage SID Collection).
- **Spektrum-analizátor** — lebegő LED-oszlopos megjelenítő a SID Playerhez 8–32 sávval, halványuló cellákkal és tükröződő padlóval. Lásd: Spektrum-analizátor.
- **Lecke-szerkesztő** — a saját programodból lépésről lépésre vezetett lecke készíthető, mint a beépített „Építsd meg: első szöveges program”: magyarázatok, mappa, kipróbálás, mentés, `.vatut` export / import. Lásd: Tutorials és a lecke-szerkesztő.
- **Online leckék** — a www.c64va.pro oldalon közzétett leckék egy kattintással a Tutorials listába tölthetők, frissítéssel együtt, a sajátodat pedig a lecke-szerkesztőből töltheted fel jóváhagyásra. Lásd: Online leckék.

- **Minden makró az operandus mezőben kapja a paramétereit** — `SET_BG $06`, `REGION Main`, `ORG $C000`, `SPRITE_POS 1, 200, 50` …: a beírt szöveg a makró Expert-sorának paramétere, a mező pedig mutatja a várt formát. Lásd: Paraméterek az operandus mezőben.
- **Apróbb javítások** — a használt lebegő ablak ki van emelve, a Support ablak csevegőként néz ki, az egérgörgő vízszintesen görgeti a fülsávot, és az Ultimate Basic kódméret-oszlop már nem takarja a sorszámokat.

A 2.6.1 verzió újdonságai

- **Ciklusszámok Expert módban** — a gutter minden utasítás ciklusát mutatja, a kijelölés összegét és azt, hány raszter sort foglal. Lásd: Ciklusszámok.
- **Optimalizálási tippek** — az Expert szerkesztő jelöli a rövidebben vagy gyorsabban írható sorokat (JSR + RTS, felesleges `CMP #0`, újratöltés, `INC` / `DEC` minta…). Lásd: Optimalizálási tippek.
- **Táblagenerátor** — képernyő- és bitmap-sorcímek, pixelmaszk, szorzó- / osztótáblák, negyed-négyzet, log / exp, bittükrözés, saját kifejezés és kigörgetett kitöltő / másoló speedcode. Lásd: Táblagenerátor.
- **VICE Monitor** — hot reload a futó VICE-ba, élő változók címkére és címre, PC-mintavételes profiler. Lásd: VICE Monitor.
- **Zero page foglaltsági térkép** és **leválasztható memóriaablak**. Lásd: Memóriaablak és zero page foglaltság.
- **Ultimate Basic** — kódméret soronként, tippek, és Sprite Editor → Ultimate Basic export animációval.
- **SID Editor → .sid (PSID) export** és **sprite multiplexer** projekt sablon (16 sprite).
- Több lebegő ablak lehet nyitva egyszerre; amelyikre kattintasz, az kerül felülre.

A 2.6.0 verzió újdonságai

- **A Visual Assembler teljesen ingyenes lett** — megszűnt a Freemium / Professional licencelés. Minden szerkesztő, build és hardveres futtatás, mind a 7 felületnyelv és minden téma mindenkinek elérhető, regisztráció és sorozatszám nélkül. Az ok egyszerű: a magyar adószabályok mellett az eladásból az adók és az adminisztráció után túl kevés maradna ahhoz, hogy megérje pénzért adni. A már meglévő sorozatszámra nincs többé szükség. Lásd: Ingyenes és nyílt forrású.

A 2.5.9 verzió újdonságai

- **Commodore 64 téma** — a C64 bekapcsolási képernyője: világoskék keret a kék panelek körül, világoskék szöveg, inverz gombok, a C64 palettájából vett szintaxisszínek és C64 betűkészlet a monitornézetben.
- **Commodore Beige téma** — a klasszikus „kenyérdoboz”: bézs tok, mustársárga funkcióbillentyű-gombok, a logó szivárványcsíkja és C64 kék monitornézet. A témaválasztóban vagy a téma gombbal léptetve érhető el.
- **Commodore Starlight téma** — világos téma az átlátszó C64 tok nyomán: jégfehér üveg, amelyen átdereng a szivárványos RGB-fény, magenta kiemelés, szivárványos szintaxisszínek és füstös sötét monitornézet.
- **Verziókezelés ablak** — a fejlécénél fogva mozgatható, és ott nyílik meg újra, ahol hagytad; ha a napló vagy egy diff megnöveli, a képernyőn marad, és kívülre kattintva már nem zárul be (Esc-re igen). Lásd: Verziókezelés.
- **Projektszintű verziókezelés** — megnyitott `.proj` projekt fülénél a **Fájl / Projekt** kapcsolóval a commit a projekt összes forrásfájlját (projektfa + include-olt fájlok) egy bejegyzésben rögzíti. A napló commitonként mutatja a

változott fájlokat, a Diff fájlonként dolgozik, a Checkout minden fájlt visszatölt a szerkesztő füleibe, és az automatikus mentések projektszinten is készülnek. Git módban a napló és a diff az egész repóra vonatkozik. Lásd: Projekt mód.

A 2.5.8 verzió újdonságai

- **Verziókezelés** — a projekt snapshotokból git-szerű verziókezelés lett: commitok üzenettel és rövid hash-sel, napló **HEAD** jelöléssel, **Diff** a munkapéldánnyal, **Checkout** a szerkesztőbe és külön jelölt automatikus mentések. A Block módú (JSON) programokat assembly-sorokként hasonlítja össze, így a diff a valódi változásokat mutatja. Block, Expert és Ultimate Basic fülön is működik. Lásd: Verziókezelés.
- **Git integráció** — a `git.exe` a **Beállítások** → **Verziókezelés** alatt adható meg (vagy az app megkeresi). Ha a projekt git repóban van, a dialógus mutatja az ágat, a változott fájlokat, és tud commitot (előtte ment), naplót, diffet **RAW** nézettel, checkoutot, **Pull** / **Push**-t és **git init**-et.
- **Hibabejelentés** — beépített bejelentő űrlap legfeljebb 4 képernyőképpel (fájl, Ctrl+V vagy ráhúzás), opcionális e-maillel, rendszerinfóval és a legutóbbi hibákkal. Lásd: Hibabejelentés.
- **SID Player** — **véletlen lejátszás** és számonkénti **kedvencek** (★). Lásd: SID Player.

A 2.5.7 verzió újdonságai

- **VA SID Core 1.2** — a SID Player mostantól natív Rust C64 magot használ: ciklusra pontos 6510 precíz interrupt-időzítéssel, CIA 6526A, VIC-II időzítés badline-okkal, sprite-DMA-val és raster interruptokkal, valamint a 6510 I/O port. Lásd: SID Player.
- **Hi-Res szerkesztő** — két szín minden 8×8-as cellában: új **Cella háttérszín** eszköz. Lásd: Hi-Res / Multicolor Editor.
- **Pontos karakter-ROM** a ROM-nézőben, a Character Editor **Load ROM** funkciójában és a Map / Level szerkesztőben.
- **Sprite szerkesztő** — hires réteg, kitöltés, kijelölés / mozgatás, forgatás, tükrözött animáció, sprite sheet import, hitbox, X / Y ×2 előnézet. Lásd: Sprite Editor.
- **Pálya szerkesztő** — objektumok, karakter-attribútumok, él-ellenőrzés, képernyőnkénti színek, memóriamérleg RLE-vel, Futtatás VICE-ban. Lásd: Pálya szerkesztő.
- **Karakterszerkesztő** — visszavonás / újra, rajzeszközök, 90°-os forgatás, másolás / beillesztés és húzásos csere a térképen, 2×2-es csempék, kisbetűs ROM-készlet, szöveges előnézet és tartomány-export. Lásd: Character Editor.
- **Charset Canvas** — nagyítás 2×–24× (csúszka vagy Ctrl+görgő) pixelráccsal, háttérszín hires módban, hires karakterek multicolor módon belül, és **ECM** (karakterenkénti háttér). Lásd: Charset Canvas Editor.

A 2.5.6 verzió újdonságai

- **SID Player** — új toolbar gomb nyit egy lebegő lejátszót `.sid` (PSID / RSID) zenékhez: lejátszási lista fogd-és-vidd támogatással, előző / következő zene, subtune-ok, tekerés és ±10 mp, hangerő, a lista mentése / megnyitása (`.sidpl`); a lista és az ablak állapota megmarad. Lásd: SID Player.
- **VA SID Core 1.1** — beépített SID-emuláció (MOS 6581 / 8580, MOS 6581/8580-alapú 6581 szűrővel) és ciklusra pontos PSID / RSID lejátszó (6510 CPU, CIA / VIC időzítés). A **Beállítások** → **Általános** → **SID előnézet** alatt

választható; a SID Player kiírja, melyik motor szól.
- **SID szerkesztő** — tekerőgombok a hangszer-paraméterekhez, szélesebb ADSR-grafikon, **MIDI bemenet**, **felvétel** a trackerbe (ha az aktuális pattern betelt, új pattern jön), **metronóm** ütemlámpákkal, **pattern törlése**, master hangerő a VA-VK-n és **GoatTracker .sng import**. Lásd: SID Editor.
- **Beépített Ultimate Basic 1.6.4**.

A 2.5.5 verzió újdonságai

- **Hi-Res szerkesztő: pixelrács és 16×-os nagyítás** — bekapcsolt **Grid** mellett 4×-es nagyítástól minden pixel saját rácsvonalat kap, a karaktercellák határán erősebbet, és a vonal árnyalata úgy igazodik, hogy mind a 16 színen látsszon. A nagyítás csúszkája mostantól 16×-ig megy. Lásd: Hi-Res / Multicolor Editor.
- **Jobb klikkes másolás / beillesztés menü** — a Hi-Res és a Map szerkesztő vásznán (másolás, beillesztés, áttetsző beillesztés), valamint a Sprite szerkesztőben (frame másolása, frame beillesztése, a frame-csíkon is). A menüből indított beillesztés előnézete a kattintás helyén jelenik meg.
- **Számrendszer-átváltó** — a toolbar gombja kiemelve marad, amíg az átváltó látszik.
- **Javítva** — a Hi-Res szerkesztő multicolor cellarácsa kétszer olyan széles volt, mint egy karaktercella; a Hi-Res eszköztár **Paste** gombja mindig áttetszően illesztett be; a SID szerkesztő jobb klikkes menüje a kurzortól távolabb nyílt meg, amíg a dialógus középen állt.
- **Ez a kézikönyv** — a Hi-Res szerkesztő kijelölése, másolása / beillesztése és áttetsző beillesztése mostantól le van írva.

A 2.5.4 verzió újdonságai

- **Javítva: újraindításkor nem jelent meg az ablak** — ha az alkalmazást lekicsinyített vagy rejtett ablakkal zártad be (főleg macOS-en, illetve Linux / Wayland alatt), a következő indításkor előfordult, hogy nem jelent meg ablak. A láthatóságot az alkalmazás már nem menti (az ablak méretét továbbra is), és az ablak induláskor mindig megjelenik.
- **Beépített Ultimate Basic 1.6.3** — QBasic-stílusú szintaxis: `dim név as típus` változókhoz és bármilyen típusú tömbhöz, `function` / `sub` `as` típusokkal, `elseif`, `do … loop`, `while … wend`, `select case`, `exit do` / `exit sub`, egysoros `if`, `'` megjegyzés és `print a; b; c`; futásidejű stringek (`+`, `left$` / `right$` / `mid$`) és előjeles 16 bites `integer`. A UB szerkesztő kiemeli, kiegészíti és formázza, a Commands panelen új **DIM**, **FUNCTION**, **ELSEIF**, **DO**, **WEND**, **SELECT CASE**, **END**, **EXIT SUB** és **LEFT$** / **RIGHT$** / **MID$** bejegyzéssel. Lásd: UltimateBasic mód.

A 2.5.3 verzió újdonságai

- **Javítva: VICE / RetroDebugger indítása Windowson** — nem bukik el *program not found* hibával, ha a PowerShell hiányzik a PATH-ból; a VICE a saját mappájából indul (a linkelt `x64sc.exe`-t a valódira oldja fel), és a szóközös útvonalak idézőjelezve mennek át. A VICE útvonala a VICE `bin` mappájában lévő `x64sc.exe` legyen, a DLL-ek mellett.

A 2.5.2 verzió újdonságai

- **SID Editor előnézetek a valódi SID-en** — a hangszer-előnézet, a virtuális billentyűzet, a hangbeírás, az akkord / arpeggio, a sor- és az egy patternes lejátszás a WebSid-en szól; a szintetizátor csak tartalék. Lásd: SID Editor.
- **Note-off** — az **1** billentyű `===` -t ír a trackerbe: elengedi a gate-et, és lefut a hangszer release-e, mint más trackerek key-off-ja; az **Enter** szerkeszti a kijelölt cellát.
- **Leválasztható tracker** — a tracker (song, tracker és harmony sorok, rács) külön, átméretezhető lebegő panelbe tehető, és vissza.
- **Wave / arpeggio előbeállítások** — a lista mutatja a választott előbeállítást, amíg a táblát kézzel nem szerkeszted.
- **VA-VK** — a virtuális billentyűzet új neve, Play / Stop gombokkal; megjegyzi a helyét, a nyitott állapotát, az oktávot és az *Insert into tracker*-t; a SID szerkesztő **master hangereje** megmarad.
- **Tudásbázis** — C64 OS fejlesztői környezet és png2c64 linkek, kétoszlopos lista. Lásd: Tudásbázis linkek.

A 2.5.1 verzió újdonságai

- **SID Editor hangszerek** — hangonkénti szűrő-útválasztás, **3 OFF** és rezonáló cutoff-**söprés** (kezdet, vég, sebesség, oda-vissza); hangszer-**vibrato** (késleltetés, sebesség, mélység); **pulzusszélesség-moduláció**; **gate időzítő**, **hard restart** és első frame hullámforma; **wave / arpeggio tábla** előbeállításokkal; **RING** és **SYNC**. Lásd: SID Editor.
- **Új SID lejátszó** — az exportált lejátszó frame-enként futtatja a hangszer-modulációkat (saját IRQ-ból `JSR sid_frame`); az előnézet ugyanezt a lejátszót rendereli egy kis SID-szintetizátoron. A `.bin` mentés megőrzi a hangszerneveket és a sebességet.
- **Virtuális billentyűzet** — 2, 3 vagy 4 oktáv, Commodore 64 szinti kinézettel.
- **Beépített Ultimate Basic 1.6.2** — a `tune` új sorai: `imod` , `ifilt` , `igate` , `itab` ; az **Export to UltimateBasic** ezeket is kiírja.
- **Könyvjelzők és Ugrás sorra** az Expert és az Ultimate Basic szerkesztőben — Ctrl+F2 / F2 / Shift+F2, Ctrl+G.
- **Javítva:** a címkés kifejezések (`.assert` , `LEVEL_MULTICOLOR*16` , `*`) a telepített appban is működnek (nincs `eval` a kiadott CSP alatt).
- A **D64 Editor** dialógusai az aktív munkamappában nyílnak; a **Build projekt** jelentése megnevezi a kimeneti mappát.

A 2.5.0 verzió újdonságai

- **Pálya szerkesztő (Level Editor)** — új toolbar szerkesztő, ami a Map Editor képernyőit egymás alá, egymás mellé vagy 2D rácsba rendezi, és sima görgetéssel végig mutatja a teljes pályát. Lásd: Pálya szerkesztő.
- **LEVEL / LEVEL_ROW makrók** — a `.level "level.bin"[, $cím]` egy Pálya szerkesztő projektből csak az elhelyezett képernyőket és a pointer-táblákat teszi a memóriába, és `LEVEL_*` címkéket, konstansokat ad; a `.level_row dst, col[, color]` egy 40 karakteres pályasort másol egy pufferbe. Lásd: LEVEL / LEVEL_ROW.
- **JOYSTICK változós mód** — a `.joystick port, vx, vy[, fire]` sprite mozgatása helyett változókba írja az irányt (−1 / 0 / +1); a `3` -as port mindkét joystick portot olvassa. Lásd: JOYSTICK.
- **Pályagörgető demó** — a `samples/level-demo.asm` egy 2×5 képernyős pályát görget joystickkal pixelenként, minden irányban, tépésmentesen, és helyet hagy sprite-oknak és játékkódnak.

- **Számrendszer-átváltó** — nem modális DEC / HEX / BIN dialógus a toolbaron. Lásd: Számrendszer-átváltó.
- **Példák dialógus** — a **Menu** → **Példák...** kereshető, kategóriákba sorolt listában mutatja a beépített példaprogramokat. Lásd: Példaprogramok.
- **Beépített Ultimate Basic 1.6.1** — a `map load "level.bin"[, cím]` közvetlenül olvassa a Pálya szerkesztő projektet, az új `map view px, py` pedig simán, tépésmentesen görgeti a térképet egy pixelpozícióra (minta: `level_scroll_demo.ub`). Egy 16 bites összegben álló bájt-szorzat már nem csonkul 8 bitre. Az UB parancsreferenciában szerepel a `map view`, a keresés pedig elöl hozza a parancsnév-találatokat.

A 2.4.9 verzió újdonságai

- **Projekt sablonok** — a **Menu** → **Új projekt** → **Új projekt sablonból...**, vagy az új toolbar gomb a **Menu** és a **Tutorials** között, egy azonnal fordítható `.proj` projektet hoz létre saját mappában. A dialógusnak van egy **Assembler** szekciója (üres program, játék váz, hires és multicolor bitmap) és egy **UltimateBasic** szekciója (üres program, játék váz, hires bitmap, raszter IRQ), szintaxiskiemelt előnézettel. A saját `.asm` / `.ub` sablonjaidat a saját sablonok mappájából veszi. Lásd: Projekt sablonok.
- **Build projekt** — a **Menu** → **Build** → **Build projekt** a nyitott projekt (vagy a nyitott fülek) minden fordítható fájlját a forrás mellé fordítja `.prg`-be, ahol lehet, debugger szimbólumfájlokat is ír, az include-olt modulokat kihagyja, és fájlonkénti jelentést mutat. Lásd: Build projekt.
- **Ctrl + egérgörgő zoom** az Expert és az Ultimate Basic szerkesztőben (macOS-en `Cmd` + görgő; touchpad csippentés is működik).
- **A munkaterület a projekteket is megjegyzi** — az Expert és az Ultimate Basic projektfa, a startup fájl és a mentetlen projektváltozások a `.vaws` munkaterületbe kerülnek, és vele együtt állnak vissza. A régebbi munkaterületek továbbra is megnyílnak.
- **Ultimate Basic** — a beépített fordító gyorsabb, kisebb 6502 kódot generál; a memóriatérkép megkülönbözteti a valódi kódot és adatot a kitöltéstől és a hosszú `org` résektől.
- **A fájl-dialógusok az aktív munkamappában nyílnak** — módtól függően a Block/Expert vagy az Ultimate Basic munkamappában.

A 2.4.8 verzió újdonságai

- **Új KOALA makró** — a `.koala "picture.koa"` (opcionálisan `, $2000`) betölt egy Koala Painter képet, és egy 86 bájtos megjelenítő rutinnal meg is mutatja. Lásd: KOALA.
- **Koala import/export a Hires szerkesztőben** — **Files** → **Koala betöltés (.koa)...** / **Koala mentés (.koa)...** (szabványos 10003 bájtos fájl; a hires kép mentéskor multicolorra alakul). A `.koa` / `.kla` fájlokat a **Kép / .bin betöltés** is felismeri.
- **Beépített Ultimate Basic 1.5.9** — új `chain "FILE" [, eszköz]` (másik program betöltése a futó program fölé és indítása); a `load "FILE"` a fájl saját címére, a `load "FILE", addr` az `addr`-ra tölt; D64 javítások. A `chain` ki van emelve és szerepel a parancsreferenciában.
- **Frissítés-jelző az Expert statusbarban** — ugyanaz a *Naprakész* / kattintható *Frissítés elérhető* jelző, mint Ultimate Basic módban.
- **Mentési dialógusok** — a fájl saját típusát kínálják (pl. *KOA*) a *Binary (*.bin)* helyett.
- **Megbízhatóbb frissítés-ellenőrzés** — a telepítők listája a c64va.pro/downloads/VA/ címen érhető el, így az ellenőrzés az új letöltési oldallal is működik; a Linux frissítő segéd már nem örökli az AppImage könyvtár-

útvonalait.

A 2.4.7 verzió újdonságai

- **Frissítés az alkalmazásból** — a **Menu** → **Check for Update** megkeresi az újabb kiadást a c64va.pro-en, és jóváhagyás után letölti a platformodnak megfelelő telepítőt (folyamatjelzővel), majd telepíti. Lásd: Frissítés az alkalmazásból. Az ellenőrzés minden induláskor csendben is lefut.
- **Frissítés-jelző az Ultimate Basic statusbarban** — **Naprakész (vX)**, vagy kiemelt, kattintható **Frissítés elérhető: vY**, ami megnyitja a frissítés dialógust.
- **Curve Editor → UltimateBasic export** — a **Files** → **Export to UltimateBasic** a táblát inicializált UB tömbként írja ki; a **Files** → **Export UltimateBasic demo** kész, futtatható UB sprite demót készít. Lásd: Curve Editor.
- **Beépített Ultimate Basic 1.5.8** — az új `data <tömb>: v, …` alak induláskor feltölt egy deklarált `array` / `array_word` tömböt (a további sorok ott folytatják, ahol az előző abbahagyta, word tömbnél 16 bites értékek), a `read` adatfolyamot nem érinti.
- **Ablakhoz igazodó elrendezés** — a szerkesztő panelek pontosan a maradék ablakmagasságot töltik ki, így nincs többé oldalszintű függőleges görgetősáv; az ASM kimeneti panel legalább 440 px széles marad; a munkamappa útvonala halvány jelvényben az ASM nézet címe fölött áll, és ugyanilyen jelvényt kapott az Expert és az Ultimate Basic eszköztár is. A dialógusgombok felirata nem törik több sorba.
- **Ultimate Basic parancsreferencia** — a `tune` és a `music` teljes, kommentezett példát mutat (cellaformátum, effektek, lejátszó-megjegyzések, `music play N`); az `inst`, `order`, `pat` és a tune `speed` saját bejegyzést kapott; a `data` az új tömbös alakot is mutatja; a `load sid` szintaxisa javítva (`load sid "tune.sid", $2000`).

A 2.4.6 verzió újdonságai

- **SID Editor → UltimateBasic export** — a **Files** → **Export to UltimateBasic** a dalt `tune ... end` blokként írja ki (hangszerek, sorrend lista és pattern sorok szövegként), utána `music play`-jel, és az Ultimate Basic szerkesztő kurzorához szúrja be (UB módon kívül a vágólapra másolja). Az UltimateBasic fordító ebből fordítási időben zero-page-mentes lejátszót épít, így a `music play` / `music stop` vezérli a dalt, és semmi sem ütközik az UB változóival. Vibrato, csúszás, note-cut és tempóváltás támogatott; legfeljebb 8 pattern, tetszőleges hosszú sorrend lista. `tune`-támogatású UltimateBasic fordítót igényel.
- **Ultimate Basic nyelvi frissítés** — a szintaxiskiemelés és a parancsreferencia mostantól ismeri a **`sfx`**-et (nem blokkoló SID effekt: `sfx csatorna, freq, frames [, wave]`), az **`org`**-ot (a sub/fn kód folytatása egy címen, nullákkal kitöltött réssel, ami `incbin` adatot is fogadhat), a **`charset on`** / **`charset off`**-ot (a VIC-II átkapcsolása a RAM karakterkészletre és vissza), valamint az **`incbin`** **szeletelést** (`incbin "font.prg", $3C00, 2` kihagyja az első bájtokat, az opcionális negyedik argumentum a hosszt korlátozza).
- **Többes kijelölés kiemelése az ASM nézetben** — több blokk kijelölésekor az összes hozzájuk tartozó ASM sor egyszerre kiemelt, a görgetési pozíció stabil marad kijelölés közben, a sorszámok pedig lágyabbak, finom függőleges elválasztó vonallal.
- **Az assembler mód megmarad az Ultimate Basic váltásoknál** — ha Ultimate Basic módban nyitsz meg projektet vagy beépített mintát, az app előbb kilép az UB-ből, és az utolsó Block/Expert módba tér vissza, így az UB puffer többé nem írhatja felül az új fül módját vagy címét. Az új fülek a helyes módban jönnek létre.
- **Kung Fu Flash 2 beállítások** — a Hardware Settings új **Detect USB** gombja újra megkeresi a cartridge soros portját, a KFF2 előfeltételek pedig már megnevezik a szükséges Kung Fu Flash firmware-t (v2.07+).

A 2.4.5 verzió újdonságai

- **Kung Fu Flash 2 támogatás — futtasd a programjaidat valódi Commodore 64/128-on.** Az osztott **Run** menü új **Kung Fu Flash 2** pontja lefordítja a programot, és USB-n keresztül közvetlenül feltölti egy valódi, bekapcsolt C64-re (vagy C128-ra), amelyben Kung Fu Flash 2 cartridge van — lemezkép, SD-kártyára másolás és emulátor nélkül. Blokk, Expert és Ultimate Basic módban egyaránt működik. A cartridge USB soros portját az app automatikusan felismeri (kézzel is megadható); a beállítást és a pontos munkafolyamatot lásd: Kung Fu Flash 2.
- **Átméretezhető Program / ASM panelek** — a Program panel és az ASM/Kimenet panel között egy vékony függőleges elválasztó húzásával állíthatod a határt (panelenként legalább 260 px). A nyilakkal léptethető, dupla kattintásra visszaáll 50/50-re, a pozíciót az app megjegyzi. Expert és Ultimate Basic módban, valamint a keskeny, egymás alá rendezett elrendezésben az elválasztó rejtve van.
- **Tisztább makrósorok az ASM nézetben** — a makró blokkok mostantól a vezető megjegyzésjel nélkül jelennek meg (`.print "HELLO"` , `.if A == 1` , `.sprite_init …` a `; .print …` helyett), az ASM szintaxiskiemelő pedig makróként színezi őket. A sima COMMENT blokkok és a régiójelölők továbbra is `;` -vel kezdődnek.
- **Egyedi makrók kiemelése az Expert szerkesztőben** — a `.macro` , `.invoke` és `.call` utáni makrónév (és a Kick-stílusú `:név(...)` hívás) külön színt kap, ahogy a makró törzsében a `{param}` behelyettesítések és a `.macro` fejléc paraméternevei is — minden témában.
- **Megújult About dialógus** — szélesebb, kétoszlopos elrendezés csoportosított projekt-, licenc- és köszönet-hivatkozásokkal; az Ultimate Basic banner háttere átlátszó, így minden témába beleolvad.
- **Web64 a termékoldalon** — új Web64 IDE szekció a c64va.pro oldalon.
- **Minta frissítés** — a `macro-with-parameter` minta a zárójeles `.macro setColor(color)` alakot használja.

A 2.4.3 verzió újdonságai

- **SID Editor: Song/sorrend lista** — a patternek elrendezése mostantól egy valódi song/sorrend listán keresztül történik a nyers pattern-bank fölött, így egy pattern ismétlődhet, és a dal hosszabb is lehet, mint a tárolt patternek száma. Lépések hozzáadása, eltávolítása és átrendezése közvetlenül a SID szerkesztőből; a lejátszás és az exportált lejátszó is ezt a listát járja be, nem a nyers pattern-sorrendet.
- **SID Editor: Effekt-oszlop** — minden tracker-cella mostantól a note mellett effektet is hordozhat: **V**ibrato, csúszás **U**p / **D**own, note-**C**ut és tempóváltás (**F**). Közvetlenül a cellába gépelhető, pl. `C-4 01 V24` (note + hangszer + vibrato) vagy `F06` (csak tempóváltás). Az effektek soronként egyszer futnak le, megegyezően az exportált lejátszóval, és mostantól helyesen szólnak mind az app-on belüli Web Audio előnézetben, mind a lefordított 6502 exportban — ez egy valódi, lejátszható effekt-motor, nem csak export-váz.
- **Visszafelé kompatibilis SID mentési formátum** — a SID mentések mostantól verzió-jelzőt hordoznak, így a régebbi, effekt-oszlop előtti `.bin` mentések pontosan úgy töltődnek be, mint eddig, effektek nélkül.
- **D64 Editor: explicit Save munkafolyamat** — minden szerkesztés (hozzáadás/kimentés/átnevezés/törlés, és mostantól a Block Editor byte-szintű szerkesztései is) egy privát munkapéldányon történik; semmi nem érinti a valódi `.d64` fájlodat, amíg meg nem nyomod a **Save**-et, ami előtte automatikus biztonsági másolatot is készít. A nem mentett módosításokkal való bezárás vagy lemezváltás mostantól megerősítést kér, ahelyett hogy csendben eldobná őket.
- **D64 Editor: Block Editor hex grid** — a kiválasztott 256 byte-os blokk nyers byte-nézete mostantól egy valódi byte-onkénti hex grid egy sima szövegdoboz helyett, és a dialógusa már nem halványítja el az app többi részét, amíg nyitva van.

A 2.4.2 verzió újdonságai

- **Húzd-és-ejtsd a D64 Editorba** — egy lefordított `.prg` / `.bin` fájl (pl. a SID/sprite/char szerkesztők exportja, vagy bármi más) közvetlenül a lemezképre húzható; a Hozzáadás panel előre kitöltve nyílik meg. Több fájl egyszerre húzva sorban egymás után kerül feldolgozásra.
- **Include fájl generálása** — új eszköztár-gomb, ami egy `.inc` fájlba írja minden lemezen lévő bejegyzéshez a `.const NÉV = $CÍM` sort, hogy a fő program hivatkozhasson a betöltési címekre anélkül, hogy azokat kézzel kellene újra beírni.
- **Blokk szerkesztő (új)** — egy eszköztár-kapcsoló a D64 Editort nyers track/sector szintű hozzáférésre kapcsolja a directory szint alatt: kattintható blokktérkép (a BAM szabad/foglalt bitmapje szerint színezve, a BAM-ot és a directory láncot külön jelölve), hex nézet/szerkesztő a kiválasztott 256 byte-os blokkhoz, valamint előző/következő szektor navigáció. Közvetlenül a lemezképet olvassa/írja (`read_bin_file` / `write_bin_file`), a `c1541` -től függetlenül.

A 2.4.1 verzió újdonságai

- **Lengyel és olasz felületi nyelv** — teljes felület-fordítás (menük, dialógusok, mnemonik leírások, Ultimate Basic parancsreferencia) a meglévő magyar, angol, spanyol, német és holland mellett.
- **Online Help eszköztár-gomb** — új ikon-gomb a Debug után, a többnyelvű dokumentációs oldalt (c64va.pro/docs.html) nyitja meg közvetlenül az appból.
- **Mnemonik leírások központosítása:** a nyelvenkénti mnemonik leírások a közös fordítási fájlba kerültek a többi felületi szöveg mellé. Emellett javítva lett egy régóta fennálló hiba, ahol a `PRINT` , `PRINT_HEX` , `CLEAR_SCREEN` , `WAIT_KEY` , `SET_BORDER` , `SET_BG` , `IRQ_SETUP` és `RAND` angol leírásait egy véletlenül ottmaradt spanyol szöveg felülírta.

A 2.4.0 verzió újdonságai

- **D64 Editor** — teljes értékű lemezkép-böngésző az eszköztáron (a Curve Editor mögött). Meglévő `.d64` megnyitása, új üres lemez létrehozása, vagy a nyitott lemez közvetlen indítása VICE-ben — mindezt egy, a többi vizuális szerkesztőhöz hasonló Files ▾ menüből. Lásd: D64 Editor (meglévő lemezkép böngészése és szerkesztése).
- **Program hozzáadás / kimentés / átnevezés / törlés a D64 Editorban** — helyi fájl hozzáadása a lemez directory-jához, kijelölt bejegyzés kimentése `.prg` -ként, bejegyzés átnevezése helyben a táblázatban, vagy törlése — minden művelet közvetlenül a `.d64` fájlon hajtódik végre `c1541` -en keresztül, külön mentés lépés nélkül.
- **Betöltési cím, kicsomagolási cím és Exomizer a D64 Editorban** — fejléc nélküli nyers fájl hozzáadásakor megadható egy opcionális betöltési cím, egy Exomizer kicsomagolási célcím, és a fájl tömöríthető hozzáadás közben, ugyanazokkal a `mem` / `sfx` tömörítési módokkal, mint a Build D64 dialógus extra fájljainál. Egy már saját fejléccel rendelkező `.prg` -nél ezek a mezők teljesen kimaradnak.
- **Lemez bejegyzés típus választó** — az újonnan hozzáadott fájlhoz PRG / SEQ / USR / REL típus választható, nem csak PRG.
- **Hiteles directory lista** — a D64 Editor fájllistája a becsomagolt C64 Pro fonttal, nagybetűsen jelenik meg, a klasszikus `LOAD"$",8` hatásért.

- **Javítva:** a D64 Editorban egy bejegyzés átnevezésekor a szövegmezőbe kattintás többé nem törli azonnal a szerkesztést.
- **Javítva:** világos témában a mód-jelző eszköztár-jelvény (BLOCK MODE / EXPERT MODE / …) sötétebb és olvashatóbb, a csillanás animáció ismét látszik.

A 2.3.9 verzió újdonságai

Öt assembler kényelmi funkció, mind használható Expert mód szövegében és (ahol értelmes) blokként is. Mindegyikhez tartozik egy részletes szakasz lentebb:

- `*` **bármely kifejezésben** — a program counter szimbólum mostantól működik operandus-kifejezéseken belül is, nem csak önmagában: `BNE *-5`, `JMP *+20`, `LDA #<*`, `LDA #>(*+63)`. Az érték után álló `*` (`STRIDE*2`) továbbra is szorzás. Lásd a 7. szakaszt → A `*` program counter kifejezésekben.
- **Lokális (pontos) címkék** — a `.loop`-hoz hasonló címke a legközelebbi előtte lévő *globális* (nem pontos) címke hatókörébe tartozik, így a `DrawSprite` és a `ClearScreen` is definiálhatja a saját `.loop`-ját ütközés nélkül. Lásd: Lokális (pontos) címkék.
- **Long-branch pszeudo-utasítások** — az `LBNE`, `LBEQ`, `LBCC`, `LBCS`, `LBMI`, `LBPL`, `LBVC`, `LBVS` egy `JMP` fölötti invertált elágazásra fordul (mindig 5 byte), így a célpont tetszőleges távolságra lehet. Új **Hosszú ugrások** paletta-kategória. Lásd: LBNE / LBEQ / … (Hosszú ugrások).
- `.assert` **direktíva** — a `.assert end - start <= 256` vagy `.assert * < $A000, "üzenet"` fordítási időben kiértékelődik, és ha a kifejezés hamis, a build a tényleges értékkel együtt hibára fut. Lásd: .ASSERT.
- **Önmódosító kód (SMC) operandus-címkék** — a `LDA value:#$00` a `value` címkét az utasítás operandus-byte-jára helyezi, így a `STA value` közvetlenül azt módosítja. Lásd: Önmódosító kód operandus-címkék.
- **Barátságosabb tartományon kívüli branch-hibák** — a −128…+127 tartományon kívülre eső elágazás mostantól pontosan jelzi, mennyivel lóg túl, és javasolja a megfelelő `LBxx` hosszú ugrást.

---

1. A felület áttekintése

Az app három fő panelre oszlik:

Panel	Leírás
Bal — Paletta	Minden elérhető utasítás- és makró-blokk. Keresés vagy böngészés kategória szerint.
Közép — Program	A programod. Ide húzod a blokkokat, itt rendezed át őket, itt szerkeszted az operandusokat.
Jobb — Kimenet	Élő ASM nézet és/vagy memória monitor kimenet.

A fejléc jobb szélén lévő mód-jelvény mutatja az aktív **Blokk**, **Expert** vagy **Ultimate Basic** szerkesztőt. Azonnal frissül, amikor a szerkesztőmód változik.

Blokk módban a Program és a Kimenet panel között egy vékony **függőleges elválasztó** található. Húzással bármelyik panelnek több helyet adhatsz (mindkettő legalább 260 px marad), a ←/→ billentyűkkel 16 px-es lépésekben léptethető, dupla kattintásra pedig visszaáll az 50/50 arány. A pozíció a UI beállításokkal együtt mentődik. Expert és Ultimate Basic módban, illetve amikor az ablak annyira keskeny, hogy a panelek egymás alá kerülnek, az elválasztó rejtve van.

---

2. Blokk paletta

A bal oldali paletta minden elérhető blokkot kategóriákba csoportosítva listáz:

- **Data movement** — LDA, LDX, STA, STX, …
- **Arithmetic** — ADC, SBC, INC, DEC, CMP, …
- **Logic** — AND, ORA, EOR, BIT
- **Jumps & Branches** — JMP, JSR, RTS, BNE, BEQ, …
- **Long branches** — LBNE, LBEQ, LBCC, LBCS, LBMI, LBPL, LBVC, LBVS (tetszőleges távolságra ugrik; lásd a 8. szakaszt)
- **Register operations** — TAX, TAY, INX, DEX, …
- **Shift & Rotate** — ASL, LSR, ROL, ROR
- **Stack** — PHA, PHP, PLA, PLP
- **System** — CLC, SEC, NOP, BRK, …
- **Illegal instructions** — LAX, SAX, DCP, …
- **Structure** — LABEL, COMMENT, REGION, ENDREGION
- **Macros** — LOOP, NEXT, FOR, ENDF, PUSH, PULL, END, TEXT, BYTE, WORD, FILL, ALIGN, ASSERT, STRING, DATA, RAWBYTES, RAWTEXT, PETSCII, CHARSET, INCBIN, SID, INCLUDE, TABLE, ORG, MACRO, ENDM, INVOKE, IF, ELSE, ENDIF, VAR, WHILE, ENDW, REPEAT, UNTIL, MEMCPY, MEMSET, PRINT, PRINT_CHAR, PRINT_HEX, CLEAR_SCREEN, WAIT_KEY, DELAY, SET_BORDER, SET_BG, IRQ_SETUP, RAND, SPRITE_INIT, SPRITE_POS, WAIT_RASTER, JOYSTICK, MOUSE, SPRITE_COL, LOADFILE, REU_CHECK, REU_STASH, REU_FETCH, REU_SWAP, TURBO_SET, SUPERCPU_DETECT, TURBO_ENABLE, MAP_COPY, MAP_COPY16X16, SPRITE_ANIM, SCORE_BCD

A paletta tetején lévő **keresőmezővel** név szerint szűrhetsz. Kattints az **Add selected block** gombra, vagy húzz egy blokkot a programterületre.

Paraméterek az operandus mezőben

Minden paraméteres blokk az **Add** előtt az **operandus mezőben** kapja a paramétereit. Utasításnál ez az operandus (`06` a `LDA #$06` -hoz); a TEXT, BYTE, LABEL, REGION és a többi szöveges vagy listás makrónál a szövegük vagy értékük; minden más makrónál az, ami Expert módban a kulcsszava után áll:

Blokk	Operandus mező	Eredmény
SET_BG	`$06`	`.set_bg $06`
REGION	`Main`	`.region Main`
ORG	`$C000`	`* = $C000`
SPRITE_POS	`1, 200, 50`	`.sprite_pos 1, 200, 50`
LOOP	`Y, $05, myloop`	`.loop Y, $05, myloop`

A mező tippként a makró alapértelmezett paramétereit mutatja. A paraméter nélküli blokkoknál (ENDREGION, ELSE …) és a fájlos makróknál (INCBIN, KOALA, LEVEL, SID, INCLUDE — ezek a blokkban választják a fájlt) a mező le van tiltva.

---

3. Programterület

- **Fogd-és-vidd** blokkokat a palettáról, vagy **rendezd át** a meglévő blokkokat a fogantyújuk (≡) húzásával.

- Minden blokk mutatja a **mnemonikját**, az **operandus-mezőjét** és a **címzési mód választót** (ahol értelmezhető).
- A ▸ / ▾ kapcsolóval csukod be vagy nyitod ki a blokkot.
- A blokk **× (törlés)** gombjával távolítod el.
- A **Collapse All** gomb egyszerre csukja be az összes blokkot.

Blokk panel minimap

A Program panel fejlécében van egy kapcsolható **minimap** gomb. Bekapcsolva egy keskeny `56 px`-es canvas-sáv jelenik meg a panel jobb szélén, amely minden blokkot színkódolt vízszintes sávként ábrázol:

Sáv színe	Blokk típusa
Cián	Címkék
Kék/lila	Makrók és direktívák
Sárga	Utasítások
Zöld	Megjegyzések és üres sorok
Piros	Érvényesítési hibás blokkok

A becsukott blokkok csökkentett átlátszatlansággal jelennek meg. Kattints vagy húzz bárhol a minimapon, hogy a programlistát arra a pozícióra görgesd. A viewport-jelző (kiemelt színű téglalap) a lista látható részét követi. Az állapot a UI beállításokban tárolódik (`blockMinimap` kulcs).

Operandus bevitel

- Elágazás/ugrás utasításoknál (`BNE` , `JMP` , `JSR` , stb.) egy **címke-választó** legördülő jelenik meg — kattints egy definiált címkére a beszúráshoz.
- A számformátum követi az eszköztár **HEX / DEC** kapcsolóját (lásd az 5. szakaszt).

---

4. ASM nézet

A jobb oldali panel valós időben mutatja a generált kimenetet.

Kimeneti módok

Mód	Leírás
ASM	6502 assembly forrás címekkel és címkékkel
Monitor	Hex / byte dump (C64 monitor stílus)
Disassembler	Tiszta 6502 disassembly: cím · hex byte-ok · mnemonikok feloldott numerikus operandusokkal. A makrók egyedi utasításokra bomlanak (TEXT → LDA/STA párok, LOOP → LDX, stb.). A BYTE/WORD/FILL adat darabolt hex dumpként jelenik meg. A kimenetben nincsenek makró-nevek, megjegyzések vagy jegyzetek.
Both	Fent ASM, lent monitor
Toolkit	C64 referencia panel: 16 színes paletta-minta + PETSCII vezérlőkód és nyomtatható-karakter puska. Csak olvasható — részletekért lásd lentebb a „Toolkit fül" alszakaszt.

A makró blokkok az ASM és a Both nézetben megjegyzésjel nélküli, sima makrósorként jelennek meg (`.print "HELLO"` , `.wait_raster $FF` , `.invoke setColor(blue)`), és makróként vannak színezve; csak a valódi COMMENT blokkok, a régiójelölők és a figyelmeztetések kezdődnek `;` -vel.

Toolkit fül

Az ASM nézet **Toolkit** füle egy csak olvasható gyorsreferencia panel — soha nem módosítja a programodat. Két szakasz:

Szakasz	Tartalom
C64 színpaletta	16 mintás rács, minden C64 színnel, indexével (0–15 / `$00` – `$0F`) és nevével. Kattints egy mintára a hex indexének vágólapra másolásához. Vidd fölé az egeret a szín nevéért (Light Blue, Brown, stb.).
PETSCII vezérlőkódok	Gyakori vezérlőkódok a `CHROUT` ($FFD2) számára: színváltó kódok (`$05` fehér, `$1C` piros, `$1E` zöld, `$1F` kék, …), kurzormozgatás (`$11` / `$1D` / `$91` / `$9D`), inverz be/ki (`$12` / `$92`), `$93` képernyőtörlés, `$8E` / `$0E` karakterkészlet-váltás. Emellett nyomtatható tartomány puska (32-64 írásjelek, 65-90 A-Z, 91-95 zárójelek, 96-127 grafika, 160-191 shiftelt grafika, 192-223 tükör).

A Toolkit a leggyorsabb módja egy színindex vagy egy PETSCII vezérlő byte megkeresésének anélkül, hogy elhagynád a szerkesztőt.

Expert módban és **UltimateBasic módban** ugyanez a Toolkit az eszköztár **Toolkit** gombjával nyílik: Expert módban oldalpanelként a Disassembly / Monitor mellett (a bal oldali elválasztó húzásával méretezhető), UB módban a jobb oldali panelben, ahol a Disassemblyvel és a Monitorral váltja egymást. A nyitott állapotát és a szélességét megjegyzi.

Compile info és kimeneti beállítások

- A **Compile info** (a kimenet fölött) összefoglalót mutat a lefordított programról: a kód kezdőcíme, mérete és a BASIC SYS stub.
- A program kezdőcímét egy **ORG** blokk adja (`* = $C000`): az első ORG a betöltési cím, a további ORG-ok újabb szakaszokat indítanak más címeken.
- A kimeneti beállítások a **Beállítások** → **Általános** → **ASM kimenet** alatt vannak: **Macro source** (a MACRO…ENDM definíciók forrása az ASM nézetben), **Régió-megjegyzések**, **Memória-átfedések** és **Memória panel**.
- A RetroDebugger flag-ek (`-jmp` , `-unpause` , `-wait`) a **Beállítások** → **Általános** → **RetroDebugger** alatt vannak — lásd: Debugger flag-ek.

ASM sorra kattintás

Kattints bármelyik sorra az ASM nézetben, hogy **kiemeld a hozzá tartozó blokkot** a programterületen. Ha több blokk van kijelölve, mindegyik ASM sora együtt kiemelt.

ASM sorszámok

Az ASM panel **sorszámokat** jelenít meg (`001 |` , `002 |` , …), hogy megkönnyítse a hibakeresést, amikor egy fordítási hiba egy konkrét sorra mutat.

- A vizuális sorszámok csak diagnosztikai célúak.
- Az **ASM export** (Program menü) továbbra is a tiszta forrásszöveget másolja, sorszám-prefixek **nélkül**.

Fordítási folyamat modal

Nehezebb műveletek alatt egy középre igazított folyamatjelző modal jelenik meg progress bárral:

- **Futtatás** (VICE, D64, C64 Ultimate, Ultimate emulátor, Kung Fu Flash 2) — a PRG fordítása és indítása.
- **Debug** — a PRG fordítása és a RetroDebugger indítása.
- **Build project**, **Build D64** és **Exomizer** tömörítés.

A sáv alatti sor az éppen futó lépést mutatja a fájllal — például *Kick: main.asm előfeldolgozása (14 fájl)…*, *VMA (Rust): main.asm fordítása…*, *A PRG kész: 2061 bájt, $0801 címtől*, *Exomizer: … tömörítése*, *VICE: … indítása*. A modal automatikusan bezárul, amikor a művelet befejeződik vagy meghiúsul.

---

5. Beállítások és eszköztár

Vezérlő	Leírás
Számrendszer (HEX / DEC / BIN)	Beállítja az operandusok megjelenítési/beviteli formátumát az egész felületen. BIN módban az értékek binárisan, `%` prefixszel jelennek meg (pl. `%11111000`). Az ASM nézet mindig a saját formátumában mutatja az egyes blokkokat.
Nyelv	A felület nyelvének váltása angol, magyar, spanyol, német, holland (Nederlands), lengyel és olasz között (**Menu** → **Beállítások**)
Téma	Light / Dark / OLED / Black / Commodore 77 / Commodore Starlight / Commodore Beige / Commodore 64 — a **Menu** → **Nézet** témaválasztójából. Az OLED tiszta fekete hátteret használ AMOLED kijelzőkhöz. A Commodore 77 egy neon-sárga-feketén téma; ha ez az aktív téma, az indítási splash panel a téma panel-színét használja (a message card-hoz igazítva), egy kisebb, dedikált Commodore 77 logót és egy sárga progress bárt mutat. A Commodore Starlight világos téma az átlátszó C64 tok nyomán: jégfehér, áttetsző panelek, amelyeken átdereng a szivárványos RGB-fény, magenta kiemelés, szivárványos szintaxisszínek és füstös, sötét monitornézet. A Commodore Beige a klasszikus „kenyérdoboz”: bézs tok, mustársárga funkcióbillentyű-gombok, a logó szivárványcsíkja és C64 kék képernyő a monitornézetben. Decemberben a választóban a Karácsony téma is megjelenik (éjkék, fenyőzöld és fehér, minden kiemelés díszszínű); utána az app visszaáll a korábbi témára. A választott téma a következő indításkor az első kirajzolás előtt érvényesül.
CRT retro mód	Teljes képernyős CRT szűrőt kapcsol: scanline-ok, foszfor vignettálás, villódzás és hordódisztorzió. Az állapot munkamenetek között mentődik.
Show memory panel	Globális kapcsoló (**Beállítások** → **Általános** → **ASM kimenet**), amely megjeleníti vagy elrejti a teljes C64 memória panelt
BASIC SYS stub	(**Beállítások** → **Általános** → **Program**) Egy BASIC sort tesz a program elé, amely SYS-szel hívja a program originjét (az első ORG-ot). Az első ORG-nál alacsonyabbra tett szegmens a saját címén marad. Ha a program a BASIC sor alá vagy rá töltődik (`$0801`–`$080C`, például egy `$071C`-tól induló importált dump), a Futtatás és a Build helyette kis indító töltőt tesz elé: a `SYS 2061` egy másolót tesz a kazettapufferbe (`$033C`), az a helyére rakja a programot, és az első ORG-ra ugrik.
Példák…	Megnyitja a példaböngészőt, és betölt egy beépített példaprogramot — lásd Példaprogramok
Zoom in / out	A blokk-felület méretezése (minden blokk-elemre hat)
Program mentése (`Menu → File`)	Az aktuális fül mentése: a blokkprogram `.json` fájlként, a forrásfájl (`.asm`, `.s`, `.inc` …) sima szövegként (a Ctrl+S ugyanezt teszi)
Program mentése másként (`Menu → File`)	Ugyanez, minden alkalommal új fájl-dialógussal

Vezérlő	Leírás
Program betöltése (`Menu → File`)	Korábban mentett program betöltése
Save Workspace	A jelenleg nyitott, fájlhoz kötött fülek pontos halmazának mentése — az aktív füllel és minden fül szerkesztőmódjával (Block/Expert/Ultimate Basic) — `.vaws` munkaterület-fájlba
Save Workspace As	Az aktuális munkaterület mentése, minden alkalommal új fájl-dialógussal
Open Workspace	Minden nyitott fül bezárása és a `.vaws` munkaterület-fájlban tárolt fájlkészlet újranyitása
Set working folder	A fájlválasztók és mentés-dialógusok alapértelmezett mappájának kiválasztása. Az útvonal az app konfigjában tárolódik, és a menü-előnézetek az útvonal végét tartják láthatóan.
Új projekt sablonból… (`Menu → Új projekt` , vagy a **Menu** melletti toolbar gomb)	Új `.proj` projekt létrehozása Visual Assembler, ACME vagy UltimateBasic sablonból, saját mappában — lásd: Projekt sablonok
Open Project (`Menu → File`)	Több-fájlos `.proj` projekt megnyitása, minden forrásfájl fülként megnyitva
Save Project (`Menu → File`)	Az aktuális `.proj` projekt mentése (a projekt panelnek nyitva kell lennie)
Close Project (`Menu → File`)	A jelenleg nyitott projekt és minden fájlfülének bezárása. Rákérdez a nem mentett változtatások mentésére. A projekt panel visszaáll üres állapotba.
Load .asm file (Expert eszköztár)	`.asm` fájl megnyitása Expert módban, új fülön
Build PRG (`Menu → Build`)	A lefordított bináris exportja `.prg` fájlként
Build D64 (`Menu → Build`)	A program és az extra fájlok `.d64` lemezképbe csomagolása — lásd: Export to D64 dialógus
Build CRT	A program exportja 64K Magic Desk (`.crt` , cartridge type 19) fájlként. Lásd a 12b. szakaszt.
Build projekt (`Menu → Build`)	A projekt (vagy a nyitott fülek) minden fordítható fájlját a forrás mellé fordítja `.prg` -be — lásd: Build projekt
Run (osztott gomb)	A fő ▶ **Run** gomb az aktuális módot futtatja; a ▾ nyíllal válthatsz: **Run as PRG** (fordítás és VICE közvetlen indítása), **Run via D64** (.d64 lemezképbe csomagolás és VICE indítása), **Run on hardware** (PRG küldése egy C64 Ultimate / 1541 Ultimate eszközre), **D64 on hardware** (D64 küldése ugyanoda), vagy **Kung Fu Flash 2** (a PRG feltöltése USB-n egy Kung Fu Flash 2 cartridge-dzsel felszerelt valódi C64/C128-ra). Lásd a 12. és 13. szakaszt. A **Beépített C64** az appon belüli C64-en futtatja a programot, lásd: Beépített C64 és debugger.
Debug (RetroDebugger)	Fordítás és indítás RetroDebuggerben breakpointokkal, szimbólumokkal és autostart flag-ekkel (lásd a 10. szakaszt)
Run with Exomizer	Jelölőnégyzet a **Beállítások → Általános → Program** alatt — bekapcsolva minden Run és Build művelet `exomizer sfx` -szel tömöríti a PRG-t indítás vagy mentés előtt (BASIC sorral kezdődő PRG-nél `sfx basic` , így a program visszatérése után a RUN újra működik; egyébként `sfx $<betöltési cím>`). Működik a Run as PRG, Run via D64, Run on Hardware, Build PRG és Build D64 műveletekkel. Először állítsd be az Exomizer futtathatót a **Beállítások → Fájlok és útvonalak** alatt.
Automatikus mentések a	Jelölőnégyzet a **Beállítások → Általános → Verziókezelés** alatt. Bekapcsolva az app kb. 2,5 másodperccel azután, hogy abbahagytad egy fül szerkesztését, automatikus mentést tesz a

Vezérlő	Leírás
verziókezelésbe	verziótörténetbe (projekthez tartozó fülnél a projekt történetébe is). Kapcsold ki, ha csak a saját commitjaidat szeretnéd. Lásd: Verziókezelés.
Beállítások (korábban Hardware Settings)	Megnyitja a beállítások dialógust három füllel: **Általános** (assembler mód, VMA fordító, BASIC SYS stub, Exomizer, ASM kimenet, SID előnézet, debugger kiválasztása és paraméterei, verziókezelés), **Fájlok és útvonalak** (VICE, RetroDebugger, frissítési szerver, ACME, Exomizer, git) és **Integrációk** (C64 Ultimate, Ultimate emulátor, Kung Fu Flash 2). Lásd a 13. szakaszt.
Új program... (`Menu → Program`)	Megerősítő dialógust nyit, majd törli az összes blokkot a programterületről
ASM export (`Menu → Program`)	A generált assembly forrás másolása
Collapse All	Minden blokk becsukása
About	Verzióinformáció
What's New	Changelog
Knowledge Base	Referencia linkek (6502 opcode-ok, C64 KERNAL, memóriatérkép, színek)
Check for Update	Újabb kiadás keresése a c64va.pro-en, letöltése és telepítése — lásd Frissítés az alkalmazásból

Verziókezelés

Minden fülnek saját verziótörténete van, mint egy kis git repónak; a megnyitott `.proj` projekthez tartozó fül egész projektként is verziókezelhető (lásd: Projekt mód). A **Menu → Versioning → Verziókezelés...** (vagy **Commit...**) megnyitja a dialógust; a **Változások elvetése (vissza a HEAD-re)** megerősítés után visszaállítja az utolsó commitot (Git módban a lemezen lévő fájlokat is).

A szokásos menet: írj **commit üzenetet**, és nyomd meg a **Commit** gombot (vagy Ctrl+Enter) → dolgozz tovább → a **Diff** megmutatja, mi változott → újra commit.

- **Commit** — a fül aktuális állapotát menti el az üzeneteddel. Minden commit rövid hash-t kap (pl. `1bd55ba`); a legújabb a **HEAD**. Ha a HEAD óta nincs változás, nem jön létre új commit.
- **Állapotsor** — *Nincs mit commitolni, a munkapéldány tiszta* vagy *Commitolatlan változások a `<hash>` óta.*
- **Diff** — a HEAD óta történt változások; egy naplóbejegyzés **Diff** gombja azt a verziót hasonlítja a szerkesztőhöz. Az új sorok zöldek, a töröltek pirosak, a fejléc **+N / −M** összesítőt mutat. A Block módú programokat assembly-sorokként (blokkonként egy sor), az Expert forrást soronként, az Ultimate Basic fület a forrása alapján hasonlítja össze, így a JSON-ként mentett Block módú projekt változásai is olvashatók. Minden fájl összecsukható kártya az állapotával (**M** / **A** / **D** / **R**) és a számaival, a sorok mellett a régi és az új sorszám áll, a szerkesztő szintaxisszíneivel, és a megváltozott sorban a különböző rész kiemelve látszik. A **Kétoldali** gomb egymás mellé teszi a régi és az új változatot (megjegyzi), a ⤢ nagyobb helyet ad a diffnek. A forrásfájl (`.asm` , `.s` , `.inc` ...) a szövegeként tárolódik, így egy verziója is forrásfájl marad.
- **Checkout** — az adott verziót a szerkesztőbe tölti. A lemezen lévő fájl csak mentéskor változik; a **Változások elvetése** visszavisz a HEAD-re.
- **Automatikus mentések** — a **Beállítások → Általános → Verziókezelés → Automatikus mentések a verziókezelésbe** bekapcsolásával az app kb. 2,5 másodperccel a szerkesztés abbahagyása után bejegyzést tesz a naplóba. Ezek *autosave* jelölést kapnak; a legutóbbi 20 marad meg, a legfeljebb 100 commit mellett.
- Az ablak a fejlécénél fogva mozgatható; ugyanott nyílik meg újra, és **Esc**-re vagy a bezáró gombbal zárul (kívülre kattintva nem).

- A történet a projektfájl mellett, `<projekt>.snapshots.json` fájlban tárolódik (mentés előtt az app konfig mappájában), így túléli az újraindítást. A korábbi verziók snapshotjai commitként jelennek meg.

Projekt mód

Ha a fül egy megnyitott `.proj` projekthez tartozik (Expert vagy Ultimate Basic projekt panel), a dialógus fejlécében **Fájl / Projekt** kapcsoló jelenik meg; alapból a **Projekt** van kiválasztva. A projekten kívüli fülek a fent leírt fájlonkénti történetet használják.

- A **Commit** a projekt összes forrásfájlját egyetlen bejegyzésben rögzíti: a projektfa fájljait és az általuk include-olt szövegfájlokat (`.include` , `include` , `#import` , `!source` , Block módú INCLUDE blokkok). A nyitott fülek az aktuális, akár mentetlen tartalmukkal számítanak, a többi fájl a lemezről olvasódik. A projektfa bináris fájljai (fontok, `.koa` , `.bin` , `.sid` …, fájlonként legfeljebb 1 MB) is bekerülnek; a diffjük a méretváltozást mutatja, a **Checkout** pedig közvetlenül a lemezre írja vissza őket.
- A napló minden bejegyzésnél mutatja a commit üzenetet, a rövid hash-t és a **változott fájlokat** (**A** új, **M** módosult, **D** törölt).
- A **Diff** fájlonként dolgozik: minden változott fájl saját fejlécet és szakaszokat kap.
- A **Változások elvetése** ugyanígy a legutóbbi projekt-commitra (HEAD) áll vissza.
- A **Checkout** (megerősítés után) az adott verzió összes fájlját a szerkesztőbe tölti: a nyitott fülek frissülnek, a hiányzó fájlok új fülön nyílnak meg. A lemezre csak mentéskor kerül; az azóta felvett projektfájlok változatlanok maradnak.
- Automatikus mentések projektszinten is készülnek.
- A projekt története a projektfájl mellett, `<projekt>.proj.history.json` fájlban tárolódik.
- **Git** módban a napló az egész repót mutatja, commitonként a változott fájlokkal (`git log --name-status`); egy bejegyzés **Diff** gombja a commitot fájlonként mutatja, a **Diff** gomb a HEAD-et a szerkesztőben lévő projektfájlokkal veti össze (a **RAW** a repó `git diff HEAD` kimenete). A **Commit** elmenti az összes mentetlen projekt-fület, és minden változást stage-el (`git add -A`); a **Checkout** az adott commit projektfájljait a szerkesztőbe tölti.

Git

Ha a `git` telepítve van, ugyanez a dialógus valódi git repóval is dolgozik.

- **Beállítások → Fájlok és útvonalak → git** / **Beállítások → Általános → Verziókezelés: git.exe kiválasztása…** vagy **Automatikus keresés** (a PATH-ban és a szokásos telepítési mappákban); alatta a git verziója látszik. A **Git használata, ha a projekt git repóban van** (alapból bekapcsolva) a repóban lévő fájloknál magától git módba váltja a dialógust. A dialógus fejlécében a **Beépített / Git** kapcsolóval kézzel is válthatsz.
- A fejléc mutatja az ágat ↑ (push-ra váró) és ↓ (pull-ra váró) commitokkal. Az állapotsor alatti eszköztáron (ikon-gombok, a nevük rámutatáskor látszik) van az **ágválasztó lista**, az ág-eszközök, a **Stash** és a **Pull / Push / Frissítés**.
- **Ágak** — a listában másik ágat választva átvált rá (`git switch`); az **Új ág** a HEAD-ből hoz létre egyet és átvált rá, az **Átnevezés** az aktuális ágat nevezi át, az **Ág törlése** egy másikat töröl (a még be nem olvasztott ágat csak egy második megerősítés után). Ágváltáshoz mentett fülek kellenek: előbb mentsd vagy vesd el a szerkesztő változásait. Váltás után a repó nyitott fülei újratöltődnek a lemezről.
- A **Merge…** egy kiválasztott ágat olvaszt az aktuálisba (`git merge --no-edit`). Ütközésnél az ütköző fájlok listázódnak és pirosak, az állapotsor *Merge folyamatban*-t mutat: oldd fel őket a szerkesztőben, mentsd, jelöld mindegyiket **Feloldva** gombbal, és commitolj — vagy a **Merge megszakítása** (`git merge --abort`) visszavisz.
- **A változáslista** a repó minden változott fájlját mutatja a `git status` kódjával. A névre kattintva megnyílik a diffje (HEAD a lemezen lévő fájllal szemben); a mellette lévő gombok **Stage**-elik (`git add`), **Unstage**-elik (`git`

`restore --staged`) vagy **elvetik** (a lemezen vissza a HEAD-re, megerősítés után) — a nem követett fájl helyette **törölhető**.
- A **Stash** opcionális üzenettel félreteszi a nem commitolt változásokat (`git stash push`); a változások alatti **stash-lista** egy bejegyzést **alkalmaz**, **visszavesz (pop)** vagy **töröl**.
- A **Változások elvetése** előbb rákérdez, aztán a lemezen visszaállítja a változott fájlokat a HEAD-re (`git restore --staged --worktree`), és újratölti a füleiket — projektnél az egész projektmappát, egyébként a fül fájlját; a nem követett fájlok megmaradnak.
- A **Commit** előbb elmenti a fájlt, csak ezt a fájlt stage-eli (vagy minden változást a **Minden változás stage-elése (git add -A)** bejelölésével), majd `git commit`-ot futtat. Projektben a **Minden változás stage-elése** alapból be van kapcsolva; kikapcsolva a commit csak a változáslistában stage-elt fájlokat viszi.
- A napló ennek a fájlnak a commitjait mutatja az ágakkal és tagekkel. Egy bejegyzés **Diff** gombja a commitot a szülőjével veti össze; a **Diff** gomb a HEAD-et a szerkesztővel, a mentetlen változásokkal együtt. A projektfájlokat ugyanúgy hasonlítja, mint a beépített mód; a **RAW** a nyers `git diff`-et mutatja.
- A **Checkout** az adott commit fájlját a szerkesztőbe tölti (`git show`), a lemezhez nem nyúl.
- **Pull** (`--ff-only`), **Push** (az első push beállítja az upstreamet: `-u origin <ág>`), és **Repó létrehozása (git init)**, ha a mappa még nem repó.
- A git konzolablak nélkül fut, és soha nem kér jelszót: a hitelesítésnek credential managerből vagy SSH kulcsból kell jönnie, különben a hibaüzenet a dialógusban jelenik meg.

Frissítés az alkalmazásból

A **Menu** → **Check for Update** megkeresi a legújabb kiadást a c64va.pro-en (a platformodhoz tartozó letöltési linkeket olvassa ki), és összeveti a futó verzióval:

- **Naprakész** — a dialógus ezt jelzi.
- **Frissítés elérhető** — az **Update now** letölti a telepítőt (a letöltést folyamatjelző mutatja), a **Later** bezárja a dialógust, a **Skip this version** pedig kikapcsolja az adott kiadás automatikus felajánlását.

A letöltés után az app telepíti a frissítést és újraindul:

Platform	Mi történik
Windows	A telepítő passzív módban fut (csak folyamatjelző), és a végén elindítja az új verziót. Gépszintű telepítésnél rendszergazdai jogot kér (UAC).
macOS	A DMG felcsatolódik, az app bundle helyben lecserélődik, majd elindul az új verzió. Ha az app mappája nem írható, csak megnyílik a DMG, és kézzel húzhatod át az appot.
Linux	Az AppImage lecserélődik az újra. A `.deb` / `.rpm` telepítés `pkexec`-kel frissül (jelszót kér); `pkexec` nélkül a csomag a rendszer szoftvertelepítőjében nyílik meg.

Ha vannak mentetlen fülek, az app bezárás előtt rákérdez. Ugyanez az ellenőrzés minden induláskor néhány másodperc után csendben is lefut: csak akkor jelenik meg bármi, ha van újabb verzió. Az eredmény az Ultimate Basic statusbarban is látszik — **Naprakész (vX)**, vagy kiemelt **Frissítés elérhető: vY**, ami megnyitja a frissítés dialógust.

Frissítési szerver: a **Beállítások** → **Fájlok és útvonalak** → **Frissítés** → **Frissítési szerver** adja meg, honnan keresi a kiadásokat (alapértelmezés `https://www.c64va.pro/`); a telepítőket a `<szerver>downloads/VA/` alól olvassa. A cím `https://` legyen (sima `http://` csak `localhost`-ra, teszteléshez); az **Alapértelmezett** visszaáll a c64va.pro-ra.

Ingyenes és nyílt forrású

A Visual Assembler ingyenes: minden funkció — a vizuális szerkesztők (SID, Sprite, Char, Charset, Hi-Res, Map, Level, D64, Curve), a SID Player, a **Build PRG**, **Build D64**, **Build CRT**, **Build project**, az Exomizer tömörítés, a futtatás C64 Ultimate-en és Kung Fu Flash 2-n, valamint minden felületi nyelv és téma — regisztráció és sorozatszám nélkül működik. A forráskód a GNU General Public License v3.0 alatt érhető el (lásd **Menu** → **About**). A 2.5.9-es verzióig az alkalmazás Freemium / Professional változatra oszlott; a 2.6.0 óta az ott megadott sorozatszámra már nincs szükség.

Ha tetszik a program, a fejlesztését adománnyal támogathatod: a **Menu** → **Adományozás** (a Nézet alatt) megnyitja a c64va.pro adományoldalát, ahol PayPallal tetszőleges összeget adhatsz.

Használati statisztika. Az alkalmazás indításkor, naponta legfeljebb egyszer névtelen jelzést küld a c64va.pro-nak az app verziójával, az operációs rendszerrel, a felület nyelvével, az előző jelzés óta eltelt indítások számával és aktív használati perceivel (egy perc akkor számít, ha az ablak látható, és az utolsó 5 percben használtad az egeret vagy a billentyűzetet), azzal, hogy van-e még 2.5.6–2.5.9 közötti sorozatszám elmentve (csak igen / nem), és egy véletlen telepítés-azonosítóval (az app `config.json`-jában, `launchPing` néven). E-mail cím, sorozatszám, projekt vagy forráskód nem kerül elküldésre; internet nélkül nem történik semmi. Az alkalmazás alkalmanként rövid üzenetet is lekér a c64va.pro-ról (például új verzióról, néha csak a támogatóknak); minden üzenet egyszer jelenik meg.

Hibabejelentés

A **Menu** → **Hiba bejelentése** egy űrlapot nyit, amely a bejelentést közvetlenül a fejlesztőnek küldi.

- Az **Összefoglaló** és a **Mi történt?** kötelező — írd le, mit csináltál, mit vártál, és mi történt helyette. Az **E-mail** nem kötelező; add meg, ha választ szeretnél.
- **Képernyőképek:** legfeljebb 4 kép a **Kép hozzáadása…** gombbal, **Ctrl+V** beillesztéssel (pl. Win+Shift+S után) vagy az űrlapra húzva. A nagy képeket az app 1920 pixelre kicsinyíti.
- A **Rendszerinfó és a legutóbbi hibák csatolása** (alapból bekapcsolva) elküldi az app verzióját, az operációs rendszert, a nyelvet, a témát, a szerkesztő- és futtatási módot, a képernyőméretet, a blokkok számát és a legutóbbi app-hibákat. A **Mi kerül elküldésre?** megmutatja az adatokat. A forráskódod nem kerül elküldésre.
- A **Bejelentés küldése** a bejelentés számával nyugtáz. A **Küldés e-mailben** helyette a levelezőprogramot nyitja meg.

Támogatás

A **Menü** → **Támogatás** egy csevegőablakot nyit a fejlesztővel: írd be a kérdést, ötletet vagy problémát, majd kattints a **Küldés** gombra (vagy nyomd meg a **Ctrl+Enter**-t). Legfeljebb 3 fájlt csatolhatsz (összesen 4 MB): képet, forrásfájlt (`.asm`, `.ub`, `.txt` …), C64-fájlt (`.prg`, `.d64`, `.sid`, `.crt` …) vagy `.zip`-et — a **Fájl csatolása…** gombbal, beillesztéssel (**Ctrl+V**) vagy az ablakra húzva. A válasz ugyanebben az ablakban jelenik meg, akár fájlokkal együtt: a képek előnézetként látszanak, egy fájlra kattintva elmented. Az ablak nem modális: a fejlécénél fogva mozgatható, a jobb alsó sarkánál átméretezhető, és közben tovább dolgozhatsz. Az alkalmazás indításkor és utána percenként ellenőrzi, érkezett-e válasz; új válasznál rövid hangjelzés szól (az ablak fejlécében lévő hangszóró gombbal némítható), a tálcaikon villog, ha az alkalmazás a háttérben van, piros pötty jelenik meg a **Menü** gombon, és az ablak egyszer magától megnyílik. Ha már van beszélgetésed, a **Menü** gomb mellett egy boríték-ikon nyitja meg; olvasatlan válasznál pirosan villog, a darabszámmal. Egy üzenet melletti **×** az adott üzenetet, az ablak fejlécében lévő kuka gomb a teljes beszélgetést törli a csatolmányokkal együtt (a fejlesztőnél is; mindkettő előbb rákérdez). Az e-mail cím nem kötelező: ha megadod, a választ e-mailben is megkapod. A beszélgetés az alkalmazás véletlen telepítés-azonosítójához kötődik, így nincs fiók és jelszó.

Munkaterületek

Egy **munkaterület** (`.vaws` fájl) megjegyzi, mely valós, lemezen lévő fájlok voltak nyitva az egyes füleken — beleértve minden fül szerkesztőmódját és azt, hogy melyik fül volt aktív —, így később újranyithatod pontosan ezt a készletet. Ez elkülönül a `.proj` projekttől: egy munkaterület átfoghatja a Block/Expert `.json` projektfájlok, önálló `.asm` fájlok és Ultimate Basic `.ub` / `.proj` források tetszőleges keverékét több fülön keresztül.

- A munkaterületek **automatikusan mentődnek** néhány száz milliszekundummal egy változtatás után, amint már mentetted vagy megnyitottad egyet.
- Az app induláskor **automatikusan visszaállítja az utolsó munkaterületet**, így a nyitott füleid ott folytatódnak, ahol abbahagytad.
- Csak a lemezen lévő valós fájlhoz kötött fülek mentődnek a munkaterületbe; egy nem mentett mintát vagy csak memóriában lévő programot tartalmazó fülnek nincs mit megőriznie, így az kihagyódik (értesítéssel, ha egyik nyitott fül sem alkalmas).
- Egy munkaterület megnyitása először bezár minden jelenleg nyitott fület — a folytatás előtt megerősítést kér.
- Ha egy munkaterület olyan fájlra hivatkozik, amelyet azóta áthelyeztek vagy töröltek, az a bejegyzés kihagyódik, és betöltés után név szerint jelentődik.
- A nyitott Expert és Ultimate Basic **projektek** is mentődnek: a projektfa (a mentetlen hozzáadásokkal és törlésekkel együtt), a startup fájl és a `.proj` mentési célja a munkaterülettel együtt áll vissza. A régebbi verzióval mentett munkaterületek ezek nélkül nyílnak meg.

Memóriaablak és zero page foglaltság

A **Teljes C64 memória** panel leválasztható: a fejlécében lévő **leválasztás** gomb külön, nem modális ablakba teszi, amely szerkesztés közben is mozgatható és átméretezhető (az ütközés-jelvény vele megy). A **visszadokkolás** gomb (vagy × / Esc) visszarakja. Az ablak megjegyzi, hogy le volt-e választva, és a méretét.

Ha a program használja a zero page-et, a memóriatérkép tetején megjelenik a **Zero page foglaltság** rész: egy 16 × 16-os rács a `$00–$FF` címekről, területenként színezve — CPU port (`$00–$01`), szabad (`$02` , `$FB–$FE`), BASIC (`$03–$8F` , `$FF`) és KERNAL (`$90–$FA`). A használt cellák kitöltöttek; a keret **írást**, a sraffozás **pointert** jelöl (`(zp),Y` / `(zp,X)`). Egy használt cellára kattintva látod, mely sorok érik el (a sorra kattintva oda ugrik), és figyelmeztetést kapsz, ha BASIC- vagy KERNAL-területre írsz.

Ha a programban benne van a SID Editor exportált lejátszója (`sid_` címkék), a lejátszó zero page-e (`$02–$1C` , `$2F` , `$FB–$FD` , és `$50–$69` , ha a SID Editorban lévő dal két SID-es) is jelölve van, és minden cím, amelyet a saját kódod is használ, **ütközésként** jelenik meg (piros keret, figyelmeztetés a fejlécben).

Projekt sablonok

A **Menu** → **Új projekt** → **Új projekt sablonból…** — vagy a **Menu** és a **Tutorials** közötti toolbar gomb — megnyitja a sablon dialógust:

1. Fent válassz szekciót: **Visual Assembler**, **ACME** vagy **UltimateBasic** (a dialógus az aktuális szerkesztőmódnak megfelelő szekción nyílik). Az **ACME** szekció csak akkor látszik, ha a **Beállítások** → **Fájlok és útvonalak** → **ACME** alatt működő ACME van beállítva.
2. Válassz sablont a listából; jobb oldalon a leírása és a szintaxiskiemelt előnézete látszik, a projekt nevével kitöltve.
3. Add meg a **projekt nevét** és a **helyet** (**Tallózás…**). Az űrlap alatti sor mutatja a pontos `.proj` útvonalat, ami létrejön.
4. Kattints a **Projekt létrehozása** gombra.

Szekció	Beépített sablonok
Visual Assembler	Üres program, Üres játék (cím / játék / vége állapotok, joystick, játékos sprite), Hires bitmap, Multicolor bitmap, Sprite multiplexer (16 sprite, képkockánként Y szerint rendezve, raszter IRQ-lánccal megjelenítve)
ACME	Üres ACME projekt, Raszter IRQ, Sprite multiplexer (16 sprite)
UltimateBasic	Üres program, Üres játék (ugyanaz az állapotgép UB-ben), Hires bitmap (`line` / `rect` / `circle` `color pen` -nel), Raszter IRQ (egy `sub` , amit a raszter megszakítás minden képkockában meghív)

A projekt saját mappában jön létre: `<hely>/<név>/<név>.proj` , mellette `main.asm` (Visual Assembler vagy ACME) vagy `main.ub` (UltimateBasic; a `.proj` Ultimate Basic projektként van jelölve). Az ACME projekt `.proj` -ja az ACME assemblert rögzíti, és a létrehozása a **Beállítások → Általános → Assembler mód**-ot ACME módra állítja (Visual Assembler projekt visszaállítja VMA módra). Meglévő projektet sosem ír felül — ilyenkor a dialógus jelez. A nyitott fülek előbb bezáródnak (mentetlen változásnál rákérdez), majd az új projekt Expert vagy Ultimate Basic módban nyílik meg, látható projekt panellel, a fő fájl startup fájlként beállítva.

Saját sablonok: a **Saját sablonok mappája** (a dialógus bal alsó sarkában) megnyitja a felhasználói sablonok mappáját az app config könyvtárában. Minden oda bedobott `.asm` vagy `.ub` fájl megjelenik a megfelelő szekcióban (a `; @assembler acme` fejlécsoros `.asm` az **ACME** alatt) (a *Saját sablonok* alatt, hacsak nem ad meg kategóriát). A sablon egy közönséges forrásfájl, a tetején opcionális, kommentsorokból álló fejléccel — assemblerben `;` , Ultimate Basicben `#` :

```
; @name Raster bars
; @name.hu Raszter csíkok
; @category Effects
; @desc Two raster bars in the border.
; @order 50
```

A `@name` , `@category` és `@desc` nyelvenként is megadható (`@name.hu` , `@desc.de` , …); az `@order` a lista sorrendjét adja. A fejléc utáni rész lesz az új fő fájl, a `{{PROJECT}}` helyére a projekt neve kerül (kommentben használd, hogy maga a sablon is forduljon).

Tutorials és a lecke-szerkesztő

A **Tutorials** ablak mappánként sorolja a bemutatókat és a leckéket. A lista alján lévő **Lecke-szerkesztő…** gomb (és a saját leckéken a **Lecke szerkesztése**) a **lecke-szerkesztőt** nyitja, amely a saját kódodból a „Építsd meg: első szöveges program”-hoz hasonló, lépésről lépésre vezetett leckét készít:

1. Írd meg a programot a Blokk vagy az Expert szerkesztőben.
2. A lecke-szerkesztőben nyomd meg a **Lépések a jelenlegi programból** gombot. Minden sorból a tanuló lépései lesznek: mnemonik kiválasztása (egy rossz előre ki van választva), operandus beírása, **Add**. Ahol a paletta nem tud mindent beállítani (például a TEXT pozícióját), a **Paraméterek beállítása** lépés az új blokkra mutat, és továbblép, amint egyezik.
3. Adj a leckének címet, leírást, nehézséget és **mappát** (bármely Tutorials kategória). Magyarázathoz tegyél be **szöveges lépést**, amely opcionálisan az ablak egy részére mutat (paletta, program panel, ASM nézet, Run gomb, memóriatérkép). Ha egy lépés címét vagy magyarázatát üresen hagyod, a generált szöveg jelenik meg.
4. **Lecke kipróbálása** (törli a jelenlegi programot), majd **Lecke mentése** (Ctrl+S).

A szövegek az aktuális felületnyelven mentődnek; más nyelvű tanuló ezeket látja, vagy ahol nem írtál semmit, a generált szöveget. A **Makró-paraméterek automatikus kitöltése** kihagyja a paraméter-lépéseket a tanulónak. A **Files** ▾ menü leckét hoz létre, töröl, `.vatut` fájlba exportál, importál, és megnyitja a leckék mappáját.

Online leckék

A Tutorials ablak **Online leckék…** gombja listázza a www.c64va.pro oldalon közzétett leckéket mappával, nehézséggel és lépésszámmal. A **Letöltés** a lecke mappájába teszi a Tutorials listában; a **Frissítés** akkor jelenik meg, ha az oldalon újabb változat van (a lecke-szerkesztő minden mentéskor növeli a lecke verzióját). Saját leckét a lecke-szerkesztő **Files** ▾ → **Feltöltés a c64va.pro-ra…** menüpontjával oszthatsz meg: add meg a neved (szerzőként jelenik meg), és ha akarod, egy e-mail címet és megjegyzést. A lecke előbb jóváhagyásra vár, utána mindenki Online leckék listájában megjelenik. Az e-mail címed sosem lesz nyilvános.

Build projekt

A **Menu** → **Build** → **Build projekt** egy lépésben lefordítja a teljes projektet. A nyitott `.proj` projekt fájljai fordulnak — projekt nélkül a nyitott fájl-fülek:

- minden fordítható forrás (`.asm` , blokk `.json` , `.ub`) azonos nevű `.prg` -be fordul **a forrása mellé**; bekapcsolt **Run with Exomizer** esetén a kimenet tömörített;
- ahol elérhető, a debugger sidecar fájlok (szimbólumok / source map) a PRG mellé íródnak;
- a más fájlba `.include` / `include` által behúzott modulok (és a `.inc` fájlok) kimaradnak, ahogy a soha nem mentett fülek is;
- a jelentés minden fájlt felsorol a méretével, a kihagyás okával vagy a fordítási hibával, a statusbar pedig *N / M fájl lefordítva*-t mutat.

Példaprogramok

A **Menu** → **Példák…** megnyitja a példaböngészőt: a beépített példaprogramok témák szerint csoportosítva (Alapok, Makrók és vezérlés, Grafika, Sprite-ok és bemenet, Fájlok és betöltés, Zene, IRQ és hardver), két hasábban.

- A keresőmezőbe gépelve név vagy kategória szerint szűrhetsz; alul látszik a találatok száma.
- Kattintás kijelöl, a ↑ **/** ↓ lépteti a kijelölést; **dupla kattintás**, **Enter** vagy a **Betöltés** gomb betölti a példát.
- A dialógus a legutóbb betöltött példán nyílik meg.

.asm fájl betöltése (gyorsreferencia)

Az Expert-mód `.asm` betöltő elfogadja a gyakori 6502 forrásmintákat, és blokkokká alakítja őket:

- `* = $1500` → ORG blokk
- `Label:` → LABEL blokk
- `Label: .byte 0` → LABEL + BYTE blokkok
- `.byte ...` → BYTE blokk
- `; comment` (vagy inline `; ...`) → COMMENT blokk
- utasítások (`lda` , `jsr` , `beq` , stb.) → utasításblokkok felismert címzési móddal

Import parse-jegyzetek és jó gyakorlatok

- A lokális címkék, mint a `.wait` , megtartják a pontot: a fölöttük lévő legközelebbi globális címkéhez tartoznak (lásd: Lokális (pontos) címkék).
- A `($zp),Y` / `($zp,X)` stílusú címzéshez használj konkrét zero-page byte-ot (`$FB` , `$FC` , stb.) a legjobb kompatibilitásért.
- Kerüld az elágazási kontextusban a hexnek tűnő, kétértelmű rövid címkéket (`cc1` , `dead` , `beef`); inkább `loop_cc1` -hez hasonló neveket használj.

- Ha a programod adattal (`.byte`) kezdődik a futtatható kód előtt, tegyél egy explicit belépési ugrást (például `JMP Start`) a tetejére.

Kick import (Kick Assembler)

A Program menü **Kick import** gombja nyers Kick Assembler forrást olvas be egy új blokk-módú fülbe. Ez elkülönül a fenti Expert-mód `Load .asm file` -tól — az egyéni tooltipje jelzi, hogy **csak Kick Assembler kód támogatott** (más assemblerek részben parse-olhatók, de a round-trip nem garantált).

Támogatott minták:

- `.pc = $XXXX` origin direktíva → ORG blokk
- `.const NAME = value` , `.label NAME = value` → CONST equate
- `.macro NAME(p1, p2, ...) { ... }` `{` / `}` kapcsos test-tel vagy `.endm` -mel → felhasználói makró-definíció
- Makró-hívás `NAME(args)` , Kick kettőspont-prefix `:NAME(args)` és `.invoke NAME(args)` — mind a Kick kettőspontos formán keresztül round-trip-el
- `@local` címkék (`@loop:` , `BEQ @loop`) megőrzik a `@` prefixet változatlanul
- Operandus `label + N` / `label - N` (pl. `STA mod1+2` , `LDA xp+1`)
- Sor-megjegyzések `// ...` és `;` — mindkettő elfogadott, a `/* ... */` blokkok egyetlen megjegyzés-sorként kezelődnek
- `.text "…"` képernyő-koordináták nélkül → BYTE blokk a kódolt bájtokkal (az eredeti sor a blokk megjegyzése marad). Több elem is lehet: `"…"` , `@"…"` a `\n` `\r` `\t` `\$NN` escape-ekkel, és `$NN` / decimális bájt, `,` -vel vagy `+` -szal összefűzve
- `.encoding "név"` a következő `.text` sorok kódolását állítja: `screencode_mixed` (a Kick alapértéke), `screencode_upper` , `petscii_mixed` , `petscii_upper` , `ascii` . A kódolásban nem létező karaktert a blokk hibaként jelzi
- `.label NÉV = *` , `.label NÉV = * + N` , `.label NÉV = * - N` → címke a programszámlálónál, N-nel eltolva (például ugrás egy `BIT` operandusába: `.label loc_c474 = * + 1`)
- Mnemonik-utótagok `.abs` / `.a` és `.zp` / `.z` (`sta.abs $00fc,x` , `ldx.zp ptr,y`): abszolút vagy zero-page címzést kényszerítenek
- `*=$XXXX "név"` és `.pc = $XXXX "név"` szegmensek tetszőleges sorrendben: az elsőnél alacsonyabb szegmens a saját címén marad
- BASIC autostart átvitel: ha a program `$0801` -en kezdődik a szabványos `SYS 2061` byte stubbal (`.byte $0B,$08,$0A,$00,$9E,$32,$30,$36,$31,$00,$00,$00`), a fordító szó szerint bocsátja ki a PRG-t, ahelyett hogy egy második BASIC SYS-t csomagolna köré

A Restore64 visszafejtései (KickAssembler szintaxis) bájtra ugyanúgy importálódnak és fordulnak, mint KickAssemblerrel. A fejlécükben megadott belépési pont az első `.pc` . Ha a forrás a saját BASIC sorát (`10 SYS 2061`) a kód után, az adatblokkok között írja a `$0801` -re, az a sor a `$0801` -en marad, és a program a `SYS` -ből indul. Plus/4 / C16 program (BASIC a `$1001` -en, TED regiszterek a `$FFxx` -en) ugyanígy lefordul, de csak Plus/4 emulátorban fut, például a VICE `xplus4` -ben, a Futtatással nem.

Ismert korlát:

- Azok a konstansok, amelyek zero-page címre oldódnak fel (például a `.const BYTEADDR = $FC` `STA BYTEADDR` -ként használva) jelenleg abszolút módú utasításokra fordulnak (3 byte) a zero-page (2 byte) helyett. A lefordított kód még mindig a helyes memóriacímre ír, csak kis méret- és ciklus-többlettel a Kick Assemblerrel épített ugyanazon forráshoz képest.

Kick projektek (Kick projekt importálása…)

A **Kick projekt importálása…** (Program menü, a **Kick import** alatt) egy teljes KickAssembler forrásfát — egy fő fájlt `#import` modulokkal és `.import binary` assetekkel — VA projektté tesz. Válaszd ki a fő fájlt (`main.asm`); a VA követ minden `#import` / `.import source` / `.import binary` sort (a nem teljesülő `.if` ágakban is), és mellé ír egy `.proj` -ot a mappa nevével (`src` / `source` / `asm` mappánál a szülőmappa nevével). A projekt a mappaszerkezetükben listázza a fájlokat; a források a helyükön maradnak, így egy meglévő KickAssembler build-szkript továbbra is működik. A nem található fájlokat a hiba-ablak sorolja fel.

Kick-kompatibilis kapcsoló: bármely VA (VMA) projekt Kick-kompatibilissé tehető a **Projekt beállítások KickAssembler-kompatibilis** kapcsolójával (az importált Kick projekteknél be van kapcsolva). A beállítás a `.proj` -ban mentődik (`"kick": true`); kikapcsolva a projekt sima VMA projekt, és pontosan úgy fordul, mint eddig.

Kick-kompatibilis projektben a VA makrói és direktívái a KickAssembler szkript mellett is működnek: a `.text x, y, "…"` , `.set_border` , LOOP / NEXT, SPRITE_INIT, `.include` , `.incbin` , `.sid` , `.define` / `.if SZIMBÓLUM` / `.else` / `.endif` , a VA makrók (`.macro név(p)` … `.endm` `{p}` -vel) és a többi változatlanul a VA assemblerhez kerül, a Kick konstansok számként. A mindkettőben meglévő direktívákat az alakjuk különbözteti meg:

VA (VMA)	KickAssembler
`.for X, $0A, for1` … `.endf for1`	`.for (var i = 0; i < 10; i++) { … }`
`.if DEBUG` … `.else` … `.endif`	`.if (kifejezés) { … } else { … }`
`.macro név(p)` … `.endm` , hívása `.invoke`	`.macro Név(p) { … }` , hívása `Név(arg)`
`.var név = érték` (futásidejű változó)	`.var név = érték` (szkript-változó) — Kick-kompatibilis projektben a `.var` mindig a KickAssembleré

Élő ellenőrzés: gépelés közben a hibák, a ciklus-gutter és a peephole tippek abból a fából jönnek, amelyikhez a nyitott fájl tartozik — az indítófájl fájából, vagy ha az nem importálja a fájlt (pl. `main.asm` egy `boot.asm` indítófájl mellett), annak a projektforrásnak a fájából, amelyik importálja. A Futtatás és a Build mindig az indítófájlt fordítja.

A projektet a VA assemblere fordítja, nem a KickAssembler: minden build (Futtatás, Build PRG, D64, CRT, Exomizer, Debug, VICE Monitor, Build project, a Build gomb) előbb egy KickAssembler-előfeldolgozón futtatja át az indítófájlt, a nyitott fülek szerkesztőbeli szövegével, mentve vagy sem. Az eredmény ugyanaz a PRG, mint a KickAssembleré. Támogatott:

- `#import` , `.import source` , `#importonce` ; `.import binary` / `c64` / `text` opcionális eltolással és hosszal
- `.const` / `.label` bármilyen sorrendben, kifejezésekkel; a zero page vagy abszolút címzés az értéktől függ, mint a KickAssemblerben (a `.abs` / `.zp` továbbra is kikényszeríti)
- `.for (var i = 0; i < n; i++) { … }` és az egysoros forma, `.var` , `.eval` (`=` , `+=` , `++` …), `.if (…) { … } else .if (…) { … } else { … }` és az egysoros forma
- `.function Név(paraméterek) { .return kifejezés }` és `.macro Név(paraméterek) { … }` , hívása `Név(argumentumok)` vagy `:Név(argumentumok)`
- többszörös címkék: `!:` / `!név:` , hivatkozás `!+` , `!-` , `!++` , `!név+` , `!név-`
- `.pseudopc cím { … }` : a benne lévő címkék a logikai címet kapják, a bájtok a helyükön maradnak
- `BasicUpstart2(címke)` / `BasicUpstart(címke)` , `* = cím "szegmens"` , `.fill n, kifejezés` (`i` -vel) és `.fill n, [a, b]` , `.align` , `.byte` , `.word` , `.dword` , `.text`
- `.assert "leírás", kifejezés, true` ; a `cmdLineVars.containsKey(…)` / `cmdLineVars.get(…)` és a `#define` szimbólumok a **Projekt beállítások**-ból jönnek (a KickAssembler `:név=érték` és `-define NÉV` megfelelője)
- `LoadBinary("fájl"[, template])` : `get(i)` / `uget(i)` / `getSize()` , a template (pl. `"C64FILE, Bitmap=$0000, ScreenRam=$1f40"` , vagy `BF_KOALA` , `BF_FLI` , `BF_DOODLE` , `BF_BITMAP_SINGLECOLOR` ,

`BF_C64FILE`) blokkonként `getBitmap(i)` / `getBitmapSize()` hívást ad; `LoadSid("fájl.sid")`: `location`, `init`, `play`, `songs`, `startSong`, `size`, `name`, `author`, `copyright`, `getData(i)` …; a szkript-értékekkel írt `.fill` azonnal kibomlik

- `* = cím "név" virtual`: virtuális memóriablokk — a címkéi címet kapnak, de bájtot nem ír
- `.while (…) { … }`, `i++` / `--i` a kifejezésekben, érték nélküli `.var név`
- a `*` mint programszámláló (`jmp *`, `bne *-1`, a `.pseudopc`-ben is)
- `;` az utasítások között (`lda #1; sta $d020`, `.var a = 1; .var b = 2`): a kód után szóköz nélkül álló `;`, amelyet utasítás követ. Minden más `;` megjegyzést kezd, mint a VA-ban, így a `lda #1 ; megjegyzés` továbbra is működik
- a matematikai könyvtár (`sin`, `cos`, `toRadians`, `toDegrees`, `round`, `floor`, `sqrt`, `pow`, `atan2`, `log10`, `random` …), a színkonstansok (`BLACK` … `LIGHT_GREY`), az opkód-konstansok (`RTS`, `LDA_IMM`, `LDA_ZPX`, `JMP_IND`, `BNE_REL` …) és az `asmCommandSize()`
- `BasicUpstart2(címke)`, ahol a címke bárhol lehet, ameddig egy `* =` óta a méretek ismertek (például `* = $4000` … `start:` → SYS 16384)
- scope-ok, mint a KickAssemblerben: a `{ }` blokkoknak, az `.if` ágainak, a `.for` / `.while` köreinek és a makróhívásoknak saját címkéik és konstansaik vannak (a benne lévő `loop:` nem ütközik a külsővel; a `.pseudopc` nem scope). A scope előtti címkén át a belső címkék elérhetők: `név: { … }` → `név.belső`, `lp: .for (…)` → `lp[2].belső`, `myIf: .if (…)` → `myIf.belső`, `c1: Makró()` → `c1.belső`, `.label base = $1000 { … }` → `base.belső`
- `.namespace név { … }` és `.filenamespace név` (a fájl végéig): a címkék `név.címke` alakban érhetők el; a `@címke:`, `.label @x = …`, `.function @f` / `.macro @m` a gyökérbe teszi a szimbólumot, a `@név` a gyökérbeli szimbólumra hivatkozik. A függvények és makrók mindenhonnan láthatók, egy névtér `@` nélküliek is
- `.pseudocommand név a : b { … }`: mnemonikként hívható (`mov #10 : $1000`, `mov src,x : ($30),y`); az argumentumok megtartják a címzési módjukat (`getType()` = `AT_IMMEDIATE`, `AT_ABSOLUTE`, `AT_ABSOLUTEX` / `Y`, `AT_INDIRECT`, `AT_IZEROPAGEX` / `Y`, hiányzónál `AT_NONE`; `getValue()`), új argumentum: `CmdArgument(típus, érték)`. Mnemonik nevű pszeudo-utasításhoz kettőspont kell (`:adc …`); nélküle a mnemonik fordul
- a preprocesszor: `#define`, `#undef`, `#if` / `#elif` / `#else` / `#endif` (`!`, `&&`, `||`), `#importif`
- `List()` (`add`, `get`, `set`, `size`, `remove`, `sort`, `reverse`, `shuffle`, `addAll`), `Hashtable()` (`put`, `get`, `containsKey`, `remove`, `keys`), `.struct Név {a, b}` (`Név(1, 2)`, `p.a`, `.eval p.a = 3`, `getFieldNames()` …), stringek (`+`, `charAt`, `substring`, `size`, `toUpperCase`, `asNumber(számrendszer)` …, `toIntString` / `toHexString` / `toBinaryString` minimális hosszal), `.enum {A, B, C = 5}`, `.define név, … { … }`, `.eval lista.add(…)`, karakter a `.byte`-ban és string-kifejezés a `.text`-ben (az aktuális `.encoding` szerint)
- `.error "üzenet"`, `.errorif feltétel, "üzenet"`; a `.print` / `.printnow` kimenete az állapotsorban (az első üzenet és a darabszám) és teljesen a **Build Info**-ban látszik. A csak jelző (`.error` …) `.if` kimarad, ha a feltételéhez címkecím kellene
- `LoadPicture("fájl"[, List().add($rrggbb, …)])` GIF-re, PNG-re és JPG-re: `width`, `height`, `getPixel(x, y)`, `getSinglecolorByte(x, y)`, `getMulticolorByte(x, y)`; színtábla nélkül a színek a megjelenésük sorrendjében. Változóban átadott képnév (`LoadPicture(filename)`) is működik
- szegmensek: `.segmentdef Név [start, startAfter, align, min, max, fill, fillByte, segments, prgFiles, sidFiles, virtual, outPrg, outBin, modify="BasicUpstart", _start]`, `.segment Név [paraméterek] "blokk"`, `.segmentout [segments=…]`, `.file [name=…, segments=…, allowOverlap]`, `.disk [filename=…, name=…] { [name=…, type=…, segments=…], … }`, `.memblock`. A `start` nélküli szegmens $2000-n kezdődik, mint a KickAssemblerben. A **Futtatás** a Default szegmenst indítja; ha az üres, az első PRG kimenetet (előbb a $0801-eset). A **Build projekt** minden `.file` / `outPrg` / `outBin` kimenetet az indítófájl mellé ír, a `.disk` lemezképet c1541-gyel építi

- `.fillword n, kif`, `.lohifill n, kif` (címkével: `név.lo` / `név.hi`); mnemonik-utótagok `.im` / `.i`, `.z` / `.zp`, `.a` / `.abs`
- `.cpu _6502NoIllegals` / `_6502` (az alapértelmezés, az illegális opkódokkal) / `dtv` (`bra`, `sac`, `sir`) / `_65c02` (`(zp)` mód, `bra`, `phx` / `phy` / `plx` / `ply`, `stz`, `trb`, `tsb`, `rmb0-7` / `smb0-7`, `bbr0-7` / `bbs0-7`, `stp`, `wai`, `inc` / `dec` A, `bit #`, `jmp (abs,x)`); minden NMOS illegális opkód az összes módjával és más nevével (`sha` / `ahx`, `shx`, `shy`, `tas` / `shs`, `las` / `lae`, `xaa` / `ane`, `lxa`, `anc2`, `sbc2` …) — ezeket a VA assembler is ismeri
- `.break ["feltétel"]` és `.watch cím[, vég][, "store"]`: töréspontok és figyelők a VICE szimbólumfájlban (`break` / `watch` sorok, ahogy a KickAssembler írja) és a RetroDebugger `.dbg`-ben; a `.break` sorok a **Debug** töréspontjai is
- `Vector(x, y, z)` (`+`, `-`, `*` szám / skalárszorzat, `/`, `X()` vektoriális szorzat, `get(n)`, `getX()` …) és 4×4 mátrixok: `Matrix()`, `RotationMatrix(ax, ay, az)`, `ScaleMatrix`, `MoveMatrix`, `PerspectiveMatrix(z)`, `get(n, m)`, `set(n, m, v)`, mátrix × mátrix, mátrix × vektor
- `getNamespace()`; `.asserterror "név", kifejezés` és a kód-formák: `.assert "név", { … }, { … }` / `.asserterror "név", { … }` (az eredmény `.print` üzenet, mint a KickAssembler assert-jelentése); `* - címke` konstansban (`.label size = * - start`)
- a `.modify` / `.filemodify` KickAssembler Java-plugint igényel: a blokk módosító nélkül fordul, üzenettel

Gépelés közben a teljes fa ellenőrződik: a nyitott fájl hibás sorai jelölve vannak a szerkesztőben, az állapotsor az elsőt mutatja, a fordítási hiba ablak `fájl:sor` alakban nevezi meg őket — kattintásra megnyílik a fájl az adott sorral. A memóriatérkép a lefordított programot mutatja, a ciklus-gutter pedig a nyitott fájl ciklusait (a `.for`-ral vagy makróval kibontott sor `4×8` alakban: egy kör, nyolcszor; a kijelölés összege minden kört beszámít). A peephole-tippek a nyitott fájl soraira kerülnek; a Disassembly és Monitor panel a lefordított programot mutatja. A szerkesztő a KickAssembler szintaxisát emeli ki (`#import`, `.segment`, szkriptfüggvények, `!címkék`, `;` a `.for (…)`-ban), a kiegészítés a Kick-direktívákat (`.`), a preprocesszort (`#`) és a szkriptfüggvényeket kínálja. Az előfeldolgozó üzenetei az app nyelvén jelennek meg. A **Debug** (RetroDebugger) forrástérképet kap az eredeti fájlokkal és sorokkal, olvasható szimbólumneveket (`scope.címke`) és a projekt minden nyitott fájljának töréspontjait. A projektfa bináris fájljai nem nyílnak meg szövegként. A Dark Chambers C64 port (19 forrásfájl, 15 bináris fájl) és mind a tizenegy kódot tartalmazó KickAssembler-példa bájtra azonosan fordul a KickAssembler 5.25-tel.

UltimateBasic mód

A Visual Assembler tartalmaz egy teljes **Ultimate Basic IDE**-t. Az Ultimate Basic egy modern, fordított BASIC nyelv C64 programok, játékok és demók készítéséhez anélkül, hogy minden műveletet alacsony szintű 6502 assemblyben kellene megírni. A fordító helyben fut, és natív C64 PRG kimenetet generál.

A UB szerkesztő megnyitása

Válaszd a **UB** ikont a fő eszköztáron az Ultimate Basic módra váltáshoz. A választott szerkesztőmód megmarad az alkalmazás újraindításai között. Egy új forrás így kezdődik:

```
color bg 0
color border 0

print "HELLO FROM ULTIMATE BASIC"
```

A UB mód `.ub` forrásfájlokkal dolgozik. A **New**, **Open**, **Save** és **Save As** az aktív UB fülre vonatkozik. Egy `.ub` fájl megnyitása automatikusan aktiválja a hozzá tartozó szerkesztőfület.

Az eszköztár mutatja az aktuális UB munkamappát. Ez a mappa a Block/Expert munkamappától elkülönítve tárolódik. Amíg a UB mód aktív, a **File → Set working folder** a UB mappát választja ki; a tooltipje jelzi az aktív hatókört. A UB Open/Save dialógusok innen indulnak, és a nem mentett források ezt használják alapként a relatív `include` és `incbin` útvonalakhoz.

Szerkesztő eszközök

A UB eszköztár ugyanazt a vizuális nyelvet és egyéni tooltipeket követi, mint az Expert mód. Nyújt:

- szintaxiskiemelést az aktuális Ultimate Basic nyelvi referencia alapján;
- sorszámokat, amelyek hosszú fájloknál is szinkronban maradnak;
- minimapot és szerkesztő zoom vezérlőket (a `Ctrl` + egérgörgő is nagyít); kattints a minimapra az ugráshoz, vagy húzd a viewport-jelölőjét folyamatos görgetéshez;
- Keresést (`Ctrl+F` / `Cmd+F`) az Expert-stílusú keresősávval;
- forrásformázást szerkezet-tudatos behúzással;
- autocomplete-ot parancsokhoz és beépített függvényekhez;
- egy kereshető **Commands** panelt szintaxissal, leírással és használati útmutatóval — a leírások követik az aktuális felületi nyelvet (magyar, angol, spanyol, német, holland, lengyel, olasz), angolra visszaesve mindenhez, ami még nincs lefordítva;
- egymástól függetlenül kapcsolható **Project** és **Commands** panelt, egymás mellett megjelenítve, ha mindkettő engedélyezett;
- egymástól függetlenül kapcsolható és átméretezhető **Build Output** és **Disassembly** panelt; a **Disassembly**, a **Monitor** és a **Toolkit** a jobb oldali panelen osztozik (egyszerre egy). Hogy melyik van nyitva, és a panelek szélességét megjegyzi.

A Disassembly panelen van egy **Copy** gomb, amely a teljes megjelenített forrást vágólapra másolja. A parancssúgó a beépített fordítót követi: például a `sprite_frame id, data_address [, frame]` egymást követő 64 byte-os sprite frame-ekből választ animációs képet.

A szerkesztő az Ultimate Basic 1.6.3 QBasic-stílusú szintaxisát is ismeri: `dim név as típus` , `function … end function` , `sub … end sub` , `elseif` , `do … loop [while | until]` , `while … wend` , `select case` , `exit do` / `exit for` / `exit while` / `exit sub` / `exit function` , a `left$` / `right$` / `mid$` stringfüggvények, valamint a `'` jellel kezdődő megjegyzés a `#` és a `rem` mellett. Azok a szavak, amelyeket a fordító csak egy adott helyen kezel kulcsszóként, csak ott kapnak színt, névként továbbra is használhatók: az `as` a BASIC típusnevekkel (`byte` , `integer` , `single` , `double`), a `case is` , a `map view` , a `wait key` és a `scroll` opciói (`row` , `x` , `y` , `left` , `right` , `narrow` , `wide`). A **Format** behúzza a `function` , `do … loop` , `while … wend` és `elseif` blokkokat, az egysoros ciklusokat (`do while f : … : loop`) nem húzza be, a Project panel a `function` blokkokat a függvények között listázza, a Commands panelen pedig új **DIM**, **FUNCTION**, **ELSEIF**, **DO**, **WEND**, **SELECT CASE**, **END**, **EXIT SUB**, **LEFT$**, **RIGHT$** és **MID$** bejegyzés van.

A parancslista szándékosan magasság-korlátozott, hogy a parancs-részlet kártya kitölthesse a maradék panel-magasságot. A részlet-terület függetlenül görgethető a hosszabb szintaxis-leírásokhoz.

Projektek, fülek és startup fájlok

Az Ultimate Basic projektek `.proj` fájlokat használnak, és több `.ub` forrást tartalmazhatnak. A Project panel listázza a nyitott fájlokat, jelöli a nem mentett füleket, és mutatja a felfedezett címkéket, függvényeket és szubrutinokat. A projekt-

műveletekkel létrehozhatsz, megnyithatsz, menthetsz és bezárhatsz egy projektet, vagy hozzáadhatsz egy újabb forrásfájlt. A Verziókezelés dialógus a projekt összes fájlját együtt is commitolja, összeveti és visszaállítja — lásd: Projekt mód. Új projekt UltimateBasic sablonból is indulhat — lásd: Projekt sablonok.

Kattints egy projektfájl melletti csillagra, hogy **startup fájlnak** jelöld. A Build, Run, D64, C64 Ultimate és Debug parancsok azt a startup forrást fordítják, még akkor is, ha éppen egy másik fül aktív. Startup-választás nélkül az aktív UB fül használódik.

A projekt panel eszköztárán két hozzáadó gomb van, a fájldialógusok pedig alapból az **UB working foldert** nyitják meg (Menu → File → Set working folder, UB-specifikus):

- **Add file** — egyszerre több fájlt is kiválaszthatsz (bármilyen típus — `.ub`, `.bin`, `.prg`, kép, font stb.). A `.ub` forrásokat új fülként is megnyitja; minden más csak hivatkozásként kerül be, hogy az `incbin` / `loadfile` asseteid a projekttel utazzanak.
- **Add folder** — egy teljes mappát (pl. egy `incbin` asset mappát, mint a `fonts/`) hivatkozásként ad hozzá a projekthez.

A fában a mappáknak van nyitó/csukó nyilacskájuk, és **jobbklikkel egy mappára** egy vagy több fájlt tudsz beszúrni alá (behúzva jelenik meg). Minden bejegyzésnek van törlés (✕) gombja; a nem `.ub` bejegyzések csak hivatkozások, kattintásra nem nyílnak meg.

A **Save project** gomb automatikusan letiltja magát és "Elmentve" állapotot mutat, amikor nincs mentetlen változás, és újra aktívvá válik, amint fájlt vagy mappát adsz hozzá, törölsz, vagy átrendezel.

Buildelés és diagnosztika

A **Build** gomb ugyanazt a középre igazított folyamatjelző élményt nyitja meg, mint a többi Visual Assembler futtatási munkafolyamat. A sikeres buildek frissítik a Build Output, Build Info és Disassembly paneleket. Engedélyezd a **Verbose**-t, hogy a fordító memóriatérkép-részletei, belső zero-page allokációi és a generált kód adatai is bekerüljenek.

Ha a fordítás meghiúsul:

- a Build Output automatikusan láthatóvá válik;
- a fordítói hibák pirossal jelennek meg;
- a középre igazított fordítási dialógus megjeleníti a hibát;
- a forrássort tartalmazó hibák kiválasztják azt a sort az aktív UB szerkesztőben.

A Build Info jelenti a load/end címeket, a kód- és PRG-méreteket, az Exomizer állapotot, a változókat, tömböket, függvényeket/szubrutinokat és címkéket.

Kódméret soronként és tippek

A **kódméret** eszköztár-gomb (láng ikon) build után megmutatja, hány bájtra fordult az egyes forrássor — a szám a sorszám-gutterben áll, a narancs sáv pedig a legdrágább sorhoz mért súlyát mutatja. A státuszsor a kurzor alatti sor bájtszámát és becsült ciklusszámát is kiírja (egyenes futás, ciklusok nélkül).

Gépelés közben a státuszsorban **tippek** gomb jelenik meg, ha valami gyanús; rákattintva megjelenik a lista, egy tippre kattintva a sorra ugrik:

- `sys $EA31` / `sys $EA81` — a `sys` `JSR`-t fordít, így a KERNAL `RTI`-je szemétre tér vissza; IRQ-ból `irq_exit` vagy `asm { JMP $EA31 }` kell;
- `wait key` — nem dokumentált forma, a `waitkey` a helyes;

- `var` / `dim` / `const`, amelyet sehol nem használsz;
- build után: `asm` blokkbeli utasítás olyan zero page címen, ahol egy Ultimate Basic változó vagy a futtatókörnyezet saját zero page-e van.

Futtatás, D64 és Exomizer

A fő osztott **Run** gomb minden szokásos célponton támogatja az Ultimate Basic-et:

Futtatási mód	Ultimate Basic viselkedés
Run as PRG	Fordítás és a PRG közvetlen indítása VICE-ban.
Run via D64	Fordítás, a szabványos D64 csomagoló dialógus megnyitása, majd a lemez indítása VICE-ban.
Run on Ultimate	A PRG feltöltése és futtatása a beállított C64 Ultimate REST kapcsolaton keresztül.
Run D64 on hardware	D64 csomagolása és küldése a beállított C64 Ultimate-re.
Kung Fu Flash 2	Az UB startup forrás lefordítása és a PRG feltöltése USB-n egy Kung Fu Flash 2 cartridge-dzsel felszerelt valódi C64/C128-ra. Lásd: Kung Fu Flash 2.

A globális **Run with Exomizer** opció (**Beállítások → Általános → Program**) a UB buildekre és a szokásos futtatási célpontokra is vonatkozik; nincs külön UB eszköztár kapcsoló. A debugger indítások szándékosan a tömörítetlen PRG-t használják, hogy a fordítói címek és szimbólumok továbbra is egyezzenek a végrehajtott programmal.

Engedélyezd a **Beállítások → Általános → Program → UltimateBasic ASM source generálása (.asm)** opciót, hogy a fordító által generált assembly egy PRG vagy D64 build mellé, azonos alapnévvel elmentődjön. Ez egy build opció, így a UB eszköztár nem tartalmaz külön ASM export gombokat. Egy `load "NAME", $address` utasítás megadja a hozzá tartozó D64 extra fájl PRG betöltési címét is.

Debugger szimbólumok és disassembly

A buildek Ultimate Basic debug információt kérnek, és három kompatibilis sidecart állítanak elő:

- `.sym` a KickAssembler-stílusú szimbólumokhoz;
- `.dbg` a C64Debugger/RetroDebugger forrás- és szegmensinformációhoz;
- `.vs` a VICE monitor címkékhez.

A színezett UB Disassembly panel feloldja az ismert címkéket, és megjeleníti a címeket, byte-okat, mnemonikokat és operandusokat. A **Debug** gomb elindítja a RetroDebuggert a nyers UB PRG-vel, a debug sidecarokkal és a fordító címkéivel, függvényeivel, szubrutinjaival, változóival és tömbjeivel. A debug wait és unpause beállítások közösek a szokásos Visual Assembler debugger konfigurációval.

Ultimate Basic kézikönyv és forrás

A UB eszköztár könyv-ikonja offline megnyitja a hozzá tartozó Ultimate Basic `MANUAL.pdf`-et; az indítási welcome dialógus manual gombja ugyanazt a kézikönyvet nyitja. A Visual Assembler a fordítót és a PDF-et is a rögzített upstream Git/Cargo függőségből veszi, így az IDE nem tart karban egy második másolatot az Ultimate Basic implementációból. Az About dialógus és a splash képernyő a tényleges függőség-verziót mutatja.

Az Ultimate Basic önálló nyílt forráskódú projektként is elérhető:

https://github.com/zstarczali/UltimateBasic

A fordító be van csomagolva a Visual Assemblerbe, így futásidőben nincs szükség külön `ub` futtathatóra.

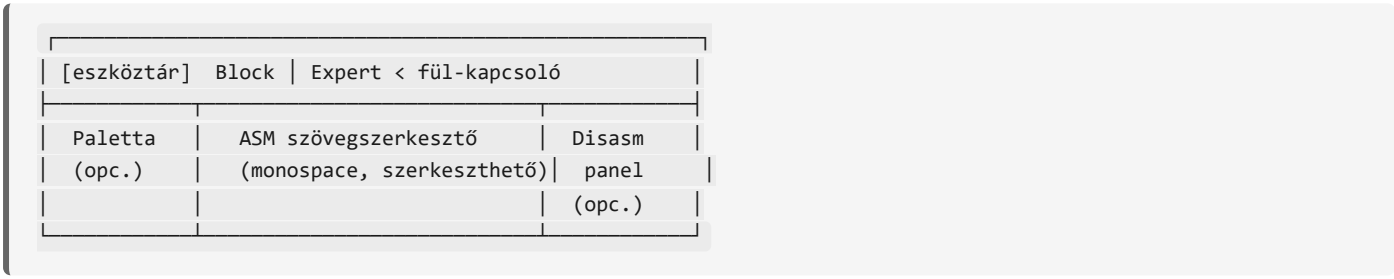
6. Expert mód

Az Expert mód egy teljes értékű, közvetlen szöveges 6502 assembly szerkesztő, amely a blokk-szerkesztő mellett él. Minden fül lehet Blokk vagy Expert módban — bármikor szabadon válthatsz köztük a felső sávban lévő **Block / Expert** kapcsolóval.

Módváltás

- **Block → Expert:** az aktuális program szöveggé szerializálódik (soronként egy utasítás, címkék, makrók direktívaként). Az Expert módban végzett szerkesztések visszaszinkronizálódnak a blokk-tömbbe, amikor visszaváltasz vagy egy műveletet indítasz.
- **Expert → Block:** a szöveg elemzése után az eredmény lecseréli a blokk-programot. Ha a parse meghiúsul, fordítási hiba dialógus jelenik meg.
- Az **üres sorok** megőrződnek a round-tripek során: az Expert szerkesztő üres sorai vékony szaggatott elválasztóként jelennek meg Blokk módban, és üres sorként állnak vissza az Expertbe visszaváltáskor.

Szerkesztő elrendezés



Panel	Kapcsoló	Leírás
Paletta	`#expert-palette-btn`	A bal oldali blokk-paletta — húzz blokkokat a szerkesztőbe, vagy kattints a kurzorhoz beszúráshoz
ASM szerkesztő	mindig látható	Teljes monospace textarea élő szintaxiskiemelés-overlay-jel
Disasm panel	`#expert-disasm-btn`	Tiszta 6502 disassembly: minden utasítás mutatja a címet, hex byte-okat és numerikus operandusokat; a makrók teljesen kifejtve

Eszköztár gombok

Gomb	ID	Funkció
Format	`#expert-format-btn`	Forrás automatikus formázása (címkék a 0. oszlopba, 4 szóköz behúzás, 1 szóköz mnemonik/operandus közt)
Load .asm	`#expert-load-asm-btn`	`.asm` fájl megnyitása — a tartalom egy **új fülbe** töltődik, a fájlnév a fül címkéje. Minden betöltött fájl önálló füllé válik saját program-blokkokkal és szerkesztő-állapottal.
Save .asm	`#expert-save-asm-btn`	A szerkesztő tartalmának mentése `.asm` fájlba (első mentéskor fájl-dialógus). Forrásfájlba mindig a szövege kerül: a Ctrl+S, a **Menü → Mentés** és az `.asm`,

Gomb	ID	Funkció
		`.s`, `.inc`, `.a`, `.a65` vagy `.lib` nevet kapó mentés-dialógus sosem írja bele a VA JSON-t
Build Info	`#expert-build-info-btn`	A Build Info dialógus megnyitása: fent kezdő- és végcím, méret, szegmensszám, alatta a szegmensek, a hibák (Kick projektben `fájl:sor`) és a `.print` kimenet, a címkék, konstansok és makrók oszlopokban
Autocomplete	`#expert-autocomplete-btn`	Az expert autocomplete javaslatok be/ki. Kikapcsolva nem jelenik meg direktíva-, mnemonik- vagy címke-popup az expert szerkesztőben.
Region selection	`#expert-region-selection-btn`	Az automatikus régió-kiemelés kapcsolása Expert módban. A fold-állapot tárolva marad, de kikapcsolva a szerkesztő a teljes forrást láthatóan tartja, és nem választja ki automatikusan az aktuális régiót.
Collapse / expand all regions	`#expert-region-fold-all-btn`	Minden `.region` blokk becsukása vagy kinyitása egy kattintással. Ha bármelyik régió nyitva van, a gomb mindet becsukja; ha minden régió már csukva van, a következő kattintás mindet kinyitja. A gomb világít, ha minden régió becsukott.
Line numbers	`#expert-line-numbers-btn`	A sorszám-margó kapcsolása a szerkesztő bal oldalán. A margó szinkronban marad a görgetési pozícióval, és élőben frissül gépelés közben.
Find	`#expert-find-btn`	A lebegő Keresés sáv megnyitása (`Ctrl+F`). Gépelj a kereséshez; a találatok kiemelődnek az overlay-ben. `Enter` / `Shift+Enter` navigál a találatok között. `Escape` bezárja a sávot.
Zoom out / in	`#expert-zoom-out-btn` / `#expert-zoom-in-btn`	A szerkesztő betűmérete csökken / nő (8–28 px). A szerkesztő fölött a `Ctrl` + egérgörgő (macOS-en `Cmd` + görgő, vagy touchpad csippentés) ugyanezt teszi. A beállítás megőrződik.
Palette	`#expert-palette-btn`	A bal oldali mnemonik-paletta megjelenítése/elrejtése
Disasm	`#expert-disasm-btn`	A disassembly panel megjelenítése/elrejtése (tiszta 6502, makrók kifejtve)
Monitor	`#expert-monitor-btn`	A monitor hex-dump panel megjelenítése/elrejtése; a bal oldali elválasztó húzásával méretezhető
Toolkit	`#expert-toolkit-btn`	A Toolkit oldalpanel (C64 paletta, PETSCII kódok) megjelenítése/elrejtése; a Monitorhoz hasonlóan méretezhető
Minimap	`#expert-minimap-btn`	A kód-minimap sáv megjelenítése/elrejtése a szerkesztő jobb oldalán

A nyitva hagyott panelek (Paletta, Disasm, Monitor, Toolkit, Projekt) és a szélességük a következő indításkor visszaállnak, bármelyik módban indul az app.

Szerkesztő gyorsbillentyűk: a `Ctrl+/` (`Cmd+/` macOS-en) megjegyzésbe teszi az aktuális sort vagy minden kijelölt sort; `Shift`-tel az elöl álló megjegyzés-jelölő eltávolítható azokból a sorokból. Az inline megjegyzések (például `LDA $12 ; magyarázat`) az utasítás során maradnak Expert és Blokk mód közti váltáskor. Blokk módban zöld dőlt `; comment` jelölőként jelennek meg a blokk fejlécében; a jelölő fölé víve az egeret a teljes szöveg feltárul, ha csonkolt.

Expert szerkesztő minimap

Az Expert szerkesztő minimapja egy keskeny canvas-sáv (`88 px`) a szerkesztő-terület jobb szélén. Minden forrássor kicsinyített reprezentációját kirajzolja:

Sáv színe	Token típus
Megjegyzés-szín	`;` -vel kezdődő sorok
Címke-szín	`label:` definíciót tartalmazó sorok
Direktíva-szín	`.byte` , `.macro` , `.region` és minden más direktíva
Mnemonik-szín	Minden más (utasítások)

Egy **félig áttetsző viewport-jelző** (kiemelt színű téglalap) mutatja, a forrás mely része látható éppen. Kattints bárhol a minimapra az odaugráshoz; húzd a folyamatos görgetéshez. A minimap függetlenül görget, hogy a viewport-jelzőt középen tartsa. Az állapot a UI beállításokban tárolódik (`expertMinimap` kulcs).

Ciklusszámok

A **ciklusszámok** eszköztár-gomb (óra ikon) egy guttert ad, amely minden utasítássor 6510-es ciklusszámát mutatja:

Kijelzés	Jelentés
`4`	fix költség
`4+`	indexelt olvasás (`abs,X` , `abs,Y` , `(zp),Y`): +1, ha az index laphatárt lép át; laphatárra igazított bázisnál (`$xx00`) a pontos érték látszik
`2/3`	elágazás: nem ugrik / ugrik
`2/4` (piros)	laphatáron átugró elágazás — időzített kódnál fontos

Több sort kijelölve a státuszsor az összeget mutatja: utasításszám, min–max ciklus és hogy ez hány PAL raszter sor (63 ciklus). A makrók és az adatsorok nem számítanak. A beállítás megmarad.

Optimalizálási tippek

Az Expert szerkesztő a lefordított kódban gyakori peephole-mintákat keres. A tippes sorok sárga csíkot kapnak, a státuszsorban pedig megjelenik egy **N tipp** gomb; rákattintva látod a listát, egy tippre kattintva a sorra ugrik. A tippek sosem írják át a kódot:

- `JSR x` , utána `RTS` → `JMP x` (1 bájttal rövidebb, 9 ciklussal gyorsabb);
- `CMP #0` / `CPX #0` / `CPY #0` közvetlenül olyan utasítás után, amely már beállította az adott regiszter N/Z flagjét (a compare a C-t is állítja!);
- ugyanaz a betöltés kétszer, vagy az épp eltárolt érték visszatöltése (`STA x` / `LDA x`);
- `LDA m` / `CLC` / `ADC #1` / `STA m` → `INC m` , és a `SEC` / `SBC #1` változat → `DEC m` ;
- `JMP` vagy elágazás a közvetlenül következő utasításra, és `PHA` közvetlenül utána `PLA` -val.

Ha két utasítás közt címke áll (lehetséges ugrási cél), az összevonásukat javasló tipp elmarad.

Hibakiemelés

A fordítani nem sikerülő sorok valós időben **pirossal** emelődnek ki (árnyalt háttér + bal oldali kiemelő szegély), 350 ms-mal minden billentyűleütés után. Az első hibaüzenet az állapotsávban is megjelenik. Javítsd a sort, és a kiemelés automatikusan eltűnik.

Szintaxiskiemelés

A szerkesztő egy átlátszó `<div>` overlay-t (`expert-hl`) használ, amely színes `<span>` elemekkel tükrözi a textarea tartalmát.

Szín	Token
Sárgászöld	Mnemonikok (`LDA` , `STA` , `JMP` , …) és long-branch pszeudo-utasítások (`LBNE` , `LBEQ` , …)
Kék	Direktívák (`.byte` , `.word` , `.fill` , `.assert` , `*=` , …)
Narancs	Számok (`$FF` , `%1010` , `255`)
Cián	Címkék — `:` -re végződő sorok, beleértve a lokális címkéket (`.loop:`) és az önmódosító kód operandus-címkéit (`value:` a `LDA value:#$00` -ban)
Türkiz	String literálok
Ibolya	Egyedi makrónevek — a `.macro` , `.invoke` és `.call` utáni név, valamint a Kick-stílusú `:név(...)` hívások
Aqua	Makró-paraméterek — a `.macro` fejléc paraméternevei és a makró törzsében lévő `{param}` behelyettesítések
Sötétzöld	Megjegyzések (`; …`)

A `REGION` / `ENDREGION` direktívák a többi assembler-direktívához hasonlóan emelődnek ki. A becsukott régiókból csak a régió-fejléc marad látható a szerkesztőben, amíg újra ki nem nyitod. Az új **Region selection** eszköztár-kapcsoló csak az automatikus aktuális-régió kiemelést vezérli Expert módban; kikapcsolva a forrás láthatóan marad a fold-állapot változtatása nélkül.

Forrásformázó

Kattints a **Format** gombra (`#expert-format-btn`) az aktuális forrás automatikus formázásához:

- A címke-definíciók a 0. oszlopba kerülnek.
- Az utasítások 4 szóközzel vannak behúzva.
- A mnemonikok nagybetűsítve.
- Pontosan egy szóköz a mnemonik és az operandus közt (a fölösleges whitespace normalizálódik).
- Ha a forrás már formázott, `"Already formatted"` állapot jelenik meg.

Projekt panel és fülek

Az Expert mód támogat egy **projekt panelt** (`#expert-project-panel`) több-fájlos `.proj` projektekhez:

- A `.proj` fájl egy JSON manifeszt, amely felsorolja a forrásfájlokat és metaadataikat.
- **Projekt beállítások** (fogaskerék a panel fejlécében, vagy **Menu → File → Projekt beállítások…**): megmutatja és szerkeszti a projektet: név, `.proj` útvonal, indítófájl, assembler (VMA vagy ACME), **KickAssembler-kompatibilis**, Kick projektnél a preprocesszor-szimbólumok (`#define` , mint a KickAssembler `-define` -ja) és a parancssori változók (`cmdLineVars` , soronként egy `név=érték` , mint a `:név=érték`), VMA projektnél a szimbólumok (`.define`) és konstansok (`.const` , soronként egy `NÉV=érték`) — a projekt által fordított forrás elé kerülnek, így a `.if NÉV` / `.if NÉV == 3` a forrás szerkesztése nélkül vált buildet —, valamint a forrás- és egyéb fájlok száma, és a **Csak Expert mód**: a projekt fájljain a blokk mód ki van kapcsolva, méretüktől függetlenül. Minden a `.proj` -ba mentődik.
- Nyiss meg egy projektet a **Menu → File → Open project** menüvel, vagy húzz egy `.proj` fájlt az ablakra.
- A projekt minden fájlja külön **fülként** nyílik meg a szerkesztő tetején lévő fül-sávban.

- A **Close Project** (`Menu → File → Close project` / `#menu-close-project`) egyszerre bezárja az aktuális projektet és minden fájlfülét. A bezárás előtt rákérdez a nem mentett változtatások mentésére. A projekt panel visszaáll üres állapotba.
- Minden fájl megjelölhető **startup fájlként** (★ csillag ikon). Ha van startup fájl beállítva, a **Run** gomb (PRG, D64, Ultimate) mindig annak a fájlnak a kódját assembleálja és futtatja — függetlenül attól, melyik fül aktív éppen. Ez Blokk és Expert módban is működik.
- Az eszköztár **Add** gombja egyszerre több fájlt is kiválaszt (bármilyen típus, nem csak `.json` / `.asm`), és hivatkozásként adja hozzá a projekthez anélkül, hogy megnyitná — ezzel vihetők bináris assetek (pl. `incbin` adat). A dialógus alapból a VA working foldert nyitja meg. A mellette lévő **Add folder** gomb egy teljes mappát (pl. egy asset mappát) ad hozzá ugyanígy, hivatkozásként.
- A fában a mappáknak van nyitó/csukó nyilacskájuk, és **jobbklikkel egy mappára** egy vagy több fájlt tudsz beszúrni alá (behúzva jelenik meg).
- A **Save project** gomb automatikusan letiltja magát és "Elmentve" állapotot mutat, amikor nincs mentetlen változás, és újra aktívvá válik, amint fájlt vagy mappát adsz hozzá, törölsz, vagy átrendezel.
- A projekt panel alján lévő **szimbólumok** szakasz függőlegesen átméretezhető a fájlfa és a szimbólumlista közti elválasztóval, így a hosszú szimbólumlisták több helyet kaphatnak, amikor szükséges.
- A **verziókezelés** az egész projektre is működik: a Verziókezelés dialógus **Projekt** kapcsolójával egy commit a projekt összes fájlját rögzíti, a Checkout pedig mindet visszatölti a fülekbe. Lásd: Projekt mód.

Fül-sáv

A fül-sáv a szerkesztő fölött jelenik meg, ha egynél több fül van nyitva.

Funkció	Leírás
Dirty pont	A fül nevén lévő kis kiemelt színű pont nem mentett változtatást jelez
Görgető nyilak	Bal/jobb görgető gombok jelennek meg, ha több fül van, mint amennyi elfér a sávban
Bezárás (×)	Bezárja a fület; ha dirty, rákérdez a mentésre
Fájlkiterjesztés	A teljes fájlnév a kiterjesztéssel együtt (`.json` , `.asm` , `.ub` ...) látszik
Ctrl+N	Új, üres fület nyit (modális ablak alatt nem)
Jobbklikk menü	Jobbklikk egy fülre (vagy üres fül-sáv területre): **New tab**, **Close tab**, **Close other tabs**, **Close tabs to the right**, **Close all tabs**. A csoportos bezárás dirty fülönként rákérdez, és megáll, ha megszakítod. A **Close all** mindig hagy egy üres fület.

> **Tipp:** A paletta-szinkron (`#expert-palette-sync-btn`) szinkronban tartja a paletta-kijelölést a kurzornál lévő mnemonikkal. Kapcsold ki, ha nem szeretnéd, hogy a paletta ugráljon szerkesztés közben.

ACME mód

A **Beállítások → Általános → Assembler mód** választja ki az Expert szerkesztő assemblerét:

- **VMA mód** (alapértelmezett) — a Visual Assembler beépített assemblere a blokkjaival, makróival és `.` direktíváival.
- **ACME mód** — a forrás ACME 0.97 szöveg, és a **Beállítások → Fájlok és útvonalak → ACME** alatt beállított ACME futtatható fordítja (**ACME kiválasztása**). A nyelvet az `ACME-0.97-reference.pdf` írja le.

A mód a `.proj` projekttel együtt mentődik, így minden projekt a saját assemblerével nyílik meg. ACME módban csak szöveges szerkesztés van: a Blokk / Expert kapcsoló zárolt, és a **Tutorialok** gombja nem jelenik meg (a leckék VMA

programot építenek blokkokból).

Ami az ACME fordítását követi:

- A **Build** (kalapács gomb az Expert eszköztáron, *ACME fordítás*), a **Futtatás** (PRG, D64, C64 Ultimate, Ultimate emulátor, Kung Fu Flash 2), a **Build PRG**, a **Build D64**, a **Build CRT**, a **Build project** és a **Debug (RetroDebugger)** ACME-vel fordít; az ACME hibái és figyelmeztetései a szokásos fordítási hiba ablakban jelennek meg, a forrással.
- A gépelési szünet után a forrás újrafordul, így a **memóriatérkép** (memóriasáv, szegmens-panel, átfedés-figyelmeztetés), a **Disassembly** és a **Monitor** panel és a ciklus-gutter a valódi ACME kimenetet mutatja; a panelek az ACME címkéit használják. Átfedés az a cím, amelyet két forrássor is ír (a PRG-ben a későbbi nyer).
- A **debug fájlok** (*Debug fájlok generálása* a Beállítások → Általános alatt) az ACME címkelistájából készülnek: `.sym` (KickAssembler stílus) és `.vs` (VICE címkék) a PRG mellé; `.dbg` forrás-térkép ACME módban nincs. A **Debug (RetroDebugger)** ugyanezeket a címkéket kapja, az Expert breakpointokat pedig az ACME lista alapján helyezi el (kód nélküli sorra tett breakpoint a következő kódsoron áll meg).
- A **ciklusszámok** és az **optimalizálási tippek** az utolsó fordítás ACME listájából (`-r`) jönnek.
- A **VICE Monitor** hot reloadja csak azokat a címtartományokat írja, amelyeket az ACME ténylegesen lefordított; az élő figyelés és a profiler az ACME címkéit és forrássorait használja.
- A SID, Hi-Res, Level, Curve és Table szerkesztő **Export to blocks** funkciója ACME forrást fűz a fájl végére (`!byte` , `!word` , `!fill` , `NÉV = érték` ; a VA makrók utasításokként íródnak ki).

A kiemelés és az automatikus kiegészítés ismeri az ACME szintaxist: `!` pszeudo-opkódok, `+név` makróhívások, kettőspont nélküli címkék, névtelen `-` / `+` címkék és `;` megjegyzések. A VMA direktíva (`.byte` , `.macro` …) ACME forrásban és az ACME `!` direktíva VMA forrásban hibaként jelenik meg.

```
        * = $c000
start   ldx #0
-       lda text,x
        beq +
        sta $0400,x
        inx
        bne -
+       rts
text    !scr "hello acme", 0
```

A `samples/ACME/acme-crt-demo.asm` kis ACME tesztprogram a Futtatáshoz, a Build PRG / D64-hez és a Build CRT-hez: üzenet a képernyő közepén és raszter-színsávok a keretben, KERNAL-hívás nélkül, így cartridge-ről is fut.

---

7. Címzési módok

Minden 6502 utasítás egy vagy több címzési módot támogat. A mód-választó minden blokkon megjelenik.

Mód	Címke	Példa	Leírás
implied	Implied	`NOP`	Nincs operandus; az utasítás önmagában teljes
immediate	Immediate	`LDA #$FF`	Inline konstans; az assembler automatikusan hozzáadja a `#` -t
zeroPage	Zero page	`LDA $10`	Egy byte-os cím a nulladik lapon (0–255)
zeroPageX	Zero page,X	`LDA $10,X`	Zero page cím + X regiszter offset (az eredmény a 0. lapon körbefordul)

Mód	Címke	Példa	Leírás
zeroPageY	Zero page,Y	`LDX $FB,Y`	Zero page cím + Y regiszter offset
absolute	Absolute	`LDA $0400`	Teljes 16 bites memóriacím
absoluteX	Absolute,X	`LDA $0400,X`	16 bites cím + X regiszter offset
absoluteY	Absolute,Y	`LDA $0400,Y`	16 bites cím + Y regiszter offset
relative	Relative/Label	`BNE loop`	Elágazás utasításokhoz; adj meg egy címkenevet vagy célcímet
indirectX	Indirect,X	`LDA ($FB,X)`	Zero page indexelt indirekt (operandus = zero page cím, 1 byte)
indirectY	Indirect,Y	`LDA ($FB),Y`	Zero page indirekt indexelt (operandus = zero page cím, 1 byte)
indirect	Indirect	`JMP ($0100)`	Indirekt; csak JMP-vel használható

Konstansok a zero page-en

A csak konstansokból (`.const` , `név = érték` , `.var`) álló, egy bájtba férő operandus az utasítás **zero page** módját kapja, pontosan úgy, mint maga a szám: `.const zp = $FB` után az `lda zp` `A5 FB` lesz (2 bájt, 3 ciklus), az `lda zp,x` pedig `B5 FB` . Ha az utasításnak nincs zero page változata az adott módban, abszolút marad (`lda zp,y` → `B9 FB 00`). Címkére nem szűkít — a címe csak az elrendezés után ismert —, az `lda.abs zp` pedig kikényszeríti az abszolút módot. A szám megtartja a leírt szélességét: a `$FB` zero page, a `$00FB` abszolút. *(Új a 2.1-ben.)*

Címke-kifejezések operandusként

Bármely operandus-mező, amely címet vagy immediate értéket elfogad, közvetlenül elfogad egy **konstans nevet** is (`CONST` blokkból vagy `LABEL` -ből). Ezen felül **címke+offset** vagy **címke−offset** kifejezésekkel hivatkozhatsz egy nevesített konstanshoz képesti címre:

Szintaxis	Példa	Leírás
`label`	`STA screen_ram,X`	A címke/konstans értékére oldódik
`label+$hex`	`STA screen_ram+$0100,X`	Címke-cím plusz hex offset
`label+decimal`	`STA screen_ram+256,X`	Címke-cím plusz decimális offset
`label-$hex`	`LDA table-$10`	Címke-cím mínusz hex offset
`$hexa+kifejezés`	`LDA $1234+1` , `STA $0400+40*2,X`	Számokból, címkékből és konstansokból álló bármely kifejezés *(2.1)*
`(címke)`	`JMP (vector)` , `JMP (table+2)`	Indirekt ugrás címkén vagy kifejezésen át *(2.1)*
`#<label`	`LDA #<screen_ram`	A címke-cím alsó byte-ja
`#>label`	`LDA #>screen_ram`	A címke-cím felső byte-ja
`*`	`BNE *`	Aktuális program counter (az utasítás saját címe); `*` -gal az elágazások végtelen önhurkot generálnak (offset `$FE`)

Példa — két képernyőlap törlése CONST-tal:

```
; .CONST screen_ram = $0400
    LDX #$00
```

```
clear:
    STA screen_ram,X
    STA screen_ram+$0100,X
    DEX
    BNE clear
```

A `*` program counter kifejezésekben

(Új a 2.3.9-ben.) A `*` már nem korlátozódik arra, hogy az egész operandus legyen — megjelenhet **bárhol egy operandus-kifejezésen belül**, és annak az utasításnak a címét jelöli, amelyre írták. Fordítási időben oldódik fel az adott utasítás valódi címéhez képest, így nincs szükség címkére a rövid relatív ugrásokhoz vagy a PC-relatív adatolvasásokhoz.

Szintaxis	Példa	Jelentés
`*`	`BNE *`	Ugrás önmagára (végtelen hurok, offset `$FE`)
`*-n` / `*+n`	`BNE *-5`, `BEQ *+4`	Elágazás az aktuális PC-hez képest *n* byte-tal
`JMP *+n`	`JMP *+20`	Az aktuális PC-ből számított abszolút ugrás
`#<*` / `#>*`	`LDA #<*`, `LDA #>*`	Az aktuális PC alsó / felső byte-ja
`#>(*+n)`	`LDA #>(*+63)`	Egy PC-relatív cím alsó / felső byte-ja

PC vs. szorzás. A `*` csak akkor számít program counternek, ha *érték-pozícióban* áll — a kifejezés elején, vagy közvetlenül egy operátor, `(`, `,`, `<`, `>` vagy whitespace után. A szám, `)` vagy azonosító után álló `*` közönséges szorzás, így a `LDA table*2` és a `CONST_A*4` változatlan.

Hol működik. Bármely operandusban, amely már elfogad kifejezést: elágazási célpontok, `JMP` / `JSR`, `LDA` / `STA` /… abszolút és indexelt, immediate alsó/felső-byte operátorok, és a `.assert` kifejezése. A `*` sosem változtatja meg egy utasítás méretét, így minden címzési módban biztonságos.

Lokális (pontos) címkék

(Új a 2.3.9-ben.) Az a címke, amelynek a neve ponttal kezdődik — `.loop`, `.skip`, `.done` — **lokális címke**. A legközelebbi előtte lévő **globális** (nem pontos) címke hatókörébe tartozik, és belsőleg `<globális>.<név>` lesz belőle. Két azonos rövid nevű lokális címke különböző globális címkék alatt **nem** ütközik.

```
DrawSprite:
    LDX #0
.loop:                  ; == DrawSprite.loop
    LDA SpriteData,X
    STA $2000,X
    INX
    CPX #63
    BNE .loop           ; a DrawSprite-on belül oldódik fel
    RTS

ClearScreen:
    LDX #0
.loop:                  ; == ClearScreen.loop — nincs ütközés
    STA $0400,X
    INX
    BNE .loop
    RTS
```

- Egy hatókörön belül `.név`-ként hivatkozz a lokális címkére.
- Másik hatókörből explicit módon `Globális.név`-ként (pl. `JMP ClearScreen.loop`).
- Az első globális címke előtt írt `.név` sima top-level `.név` marad.
- A lokális címkék változatlanul round-trip-elnek Blokk ⇄ Expert mód közt; a `<globális>.` prefix csak layout-idejű részlet, sosem tárolódik a blokk-programban.

Önmódosító kód operandus-címkék

(Új a 2.3.9-ben.) Tedd egy utasítás operandusa elé a `címke:` prefixet, hogy a címke az **operandus-byte-ra** kerüljön, ne az opcode-ra. Az utasítás abból fordul, ami a kettőspont után áll.

```
setup:
    LDA value:#$00     ; 'value' -> a #$00 operandus-byte címe
    ...
patch:
    LDA #new
    STA value          ; közvetlenül az operandus-byte-ot írja — klasszikus SMC
```

A `value` az `<utasítás címe> + 1`-re mutat (az első operandus-byte), minden címzési módban. Ez leváltja a régebbi `STA instruction+1` / `instruction: LDA #$00` mintát. Round-trip-el Blokk ⇄ Expert mód közt (az operandus-mező megtartja a `címke:` prefixet).

---

8. Szabványos 6502 utasítások

Data Movement

Mnemonik	Leírás	Módok
`LDA`	Akkumulátor betöltése	immediate, zeroPage, absolute, absoluteX, absoluteY, indirectX, indirectY
`LDX`	X regiszter betöltése	immediate, zeroPage, zeroPageY, absolute, absoluteY
`LDY`	Y regiszter betöltése	immediate, zeroPage, absolute, absoluteX
`STA`	Akkumulátor tárolása	zeroPage, absolute, absoluteX, absoluteY, indirectX, indirectY
`STX`	X regiszter tárolása	zeroPage, zeroPageY, absolute
`STY`	Y regiszter tárolása	zeroPage, absolute

Arithmetic

Mnemonik	Leírás	Megjegyzés
`ADC`	Összeadás carryvel	A legtöbb esetben állítsd be a carryt `SEC`-kel használat előtt
`SBC`	Kivonás carryvel	Állítsd be a carryt `SEC`-kel kivonás előtt
`INC`	Memória növelése	—
`DEC`	Memória csökkentése	—
`CMP`	Összehasonlítás A-val	Flag-eket állít; nem módosítja A-t

Mnemonik	Leírás	Megjegyzés
`CPX`	Összehasonlítás X-szel	—
`CPY`	Összehasonlítás Y-nal	—

Logic

Mnemonik	Leírás
`AND`	Logikai ÉS az akkumulátorral
`ORA`	Logikai VAGY az akkumulátorral
`EOR`	Kizáró VAGY az akkumulátorral
`BIT`	Bitek tesztelése a memóriában A ellenében (N, V, Z flag-eket állít)

Jumps & Branches

Mnemonik	Leírás
`JMP`	Feltétel nélküli ugrás (abszolút vagy indirekt)
`JSR`	Ugrás szubrutinba (a visszatérési címet a stackre menti)
`RTS`	Visszatérés szubrutinból
`RTI`	Visszatérés megszakításból
`BNE`	Ugrás, ha nem egyenlő (Z=0)
`BEQ`	Ugrás, ha egyenlő (Z=1)
`BCC`	Ugrás, ha carry törölve (C=0)
`BCS`	Ugrás, ha carry beállítva (C=1)
`BMI`	Ugrás, ha negatív (N=1)
`BPL`	Ugrás, ha pozitív (N=0)
`BVC`	Ugrás, ha overflow törölve (V=0)
`BVS`	Ugrás, ha overflow beállítva (V=1)

LBNE / LBEQ / … (Hosszú ugrások)

(Új a 2.3.9-ben.) A **Hosszú ugrások** paletta-kategória nyolc pszeudo-utasítást tartalmaz, amelyek feltételes elágazásként viselkednek, de **bármely címet** elérnek, nem csak −128…+127-et. Mindegyik egy invertált elágazásra fordul, amely átugrik egy 3 byte-os `JMP`-t — mindig **5 byte**:

```
LBEQ done      ; erre fordul:   BNE *+3   ($D0 $03)
               ;                JMP done  ($4C lo hi)
```

Hosszú op	Feltétel	Kibocsátott forma
`LBNE`	nem egyenlő (Z=0)	`BEQ *+3 / JMP target`
`LBEQ`	egyenlő (Z=1)	`BNE *+3 / JMP target`

Hosszú op	Feltétel	Kibocsátott forma
`LBCC`	carry törölve (C=0)	`BCS *+3 / JMP target`
`LBCS`	carry beállítva (C=1)	`BCC *+3 / JMP target`
`LBMI`	negatív (N=1)	`BPL *+3 / JMP target`
`LBPL`	pozitív (N=0)	`BMI *+3 / JMP target`
`LBVC`	overflow törölve (V=0)	`BVS *+3 / JMP target`
`LBVS`	overflow beállítva (V=1)	`BVC *+3 / JMP target`

- Operandus: címke, `*`-kifejezés vagy literál cím — ugyanaz, mint egy normál elágazási célpont.
- Költség: 5 byte és 1 extra ciklus a végrehajtott ágon egy rövid elágazáshoz képest. Nincs automatikus előléptetés rövid elágazásból — te választod az `LBxx`-et explicit módon.
- Amikor egy sima elágazás (`BNE`, `BEQ`, …) tartományon kívülre esik, a fordítói hiba mostantól megnevezi a pontos túllépést, és javasolja a megfelelő `LBxx`-et.

Register Operations

Mnemonik	Leírás
`TAX`	Átvitel A → X
`TAY`	Átvitel A → Y
`TXA`	Átvitel X → A
`TYA`	Átvitel Y → A
`TSX`	Átvitel stack pointer → X
`TXS`	Átvitel X → stack pointer
`INX`	X növelése
`DEX`	X csökkentése
`INY`	Y növelése
`DEY`	Y csökkentése

Shift & Rotate

Mnemonik	Leírás
`ASL`	Aritmetikai balra tolás
`LSR`	Logikai jobbra tolás
`ROL`	Balra forgatás a carryn keresztül
`ROR`	Jobbra forgatás a carryn keresztül

Stack

Mnemonik	Leírás
`PHA`	Akkumulátor push a stackre
`PHP`	Processzor státusz push a stackre
`PLA`	Akkumulátor pull a stackről
`PLP`	Processzor státusz pull a stackről

System / Flags

Mnemonik	Leírás
`CLC`	Carry flag törlése
`CLD`	Decimal mód törlése
`CLI`	Interrupt tiltás törlése
`CLV`	Overflow flag törlése
`SEC`	Carry flag beállítása
`SED`	Decimal mód beállítása
`SEI`	Interrupt tiltás beállítása
`NOP`	Nincs művelet
`BRK`	Kényszerített break / szoftveres megszakítás

Illegal / dokumentálatlan utasítások

Ezek haladó használatra támogatottak. Óvatosan használd — a viselkedés chipenként eltérhet.

`LAX`, `SAX`, `DCP`, `ISC`, `SLO`, `RLA`, `SRE`, `RRA`, `ANC`, `ALR`, `ARR`, `AXS`, valamint az operandust olvasó `NOP` alakok: `NOP #$nn` ($80), `NOP $nn` ($04), `NOP $nn,X` ($14), `NOP $nnnn` ($0C), `NOP $nnnn,X` ($1C)

---

9. Makró blokkok — referencia

A makró blokkokkal gyakori feladatokat végezhetsz el egy lépésben — ahelyett, hogy 10–20 utasítást írnál kézzel, leteszel egy blokkot, és az assembler generálja neked a kódot. Gondolj rájuk beépített szubrutinként.

LABEL

Mint egy **sorszám BASIC-ben** — de szám helyett névvel. Ugrási célpont a `JMP`, `JSR`, `BNE`, stb. számára.

Mező	Leírás
Címkenév	Azonosító, amit a `JMP`, `JSR`, `BNE`, stb. használ

Expert szintaxis:

```
loop:
```

Generált ASM:

```
loop:  ; $0820
```

Az aktuális cím megjegyzésként jelenik meg. A címkék **0 byte méretűek**.

COMMENT

Mint a **REM BASIC-ben** — jegyzet magadnak, amit az assembler teljesen figyelmen kívül hagy.

Expert szintaxis:

```
; Ide jön a megjegyzés szövege
```

Generált ASM:

```
; Ide jön a megjegyzés szövege
```

BYTE

Mint a **DATA BASIC-ben** — nyers byte-értékek listáját tárolja inline a programban.

Mező	Leírás
Operandus	Vesszővel elválasztott byte-értékek (pl. `$01, $02, $FF` vagy `1, 2, 255`)

Expert szintaxis:

```
.byte $01, $02, $FF
```

Generált ASM:

```
    .byte $01, $02, $FF
```

Lo/hi byte címke-hivatkozások: a BYTE elfogadja a KickAssembler / ca65 stílusú `<label` (alsó byte) és `>label` (felső byte) tokeneket a numerikus értékek mellett. Az assembler fordítási időben feloldja a címke címét, és beszúrja a megfelelő byte-ot. Példa:

```
    .byte <frame_0, >frame_0, <frame_1, >frame_1
```

Ez a `frame_0` címének alsó byte-ját tárolja, majd a felsőt, majd ugyanezt a `frame_1`-re. Hasznos jump table-ök és címlisták építéséhez.

Méret: a lista byte-jainak száma.

WORD

Mint a **DATA BASIC-ben, de 16 bites számokhoz**. Minden érték két byte-ként tárolódik (alsó byte először, majd felső byte — 6502 little-endian sorrend).

Mező	Leírás
Operandus	Vesszővel elválasztott 16 bites értékek (pl. `$0400, $C000`)

Expert szintaxis:

```
.word $0400, $C000
```

Generált ASM:

```
    .word $0400, $C000
```

Méret: 2 byte szavanként.

FILL

Mint a `FOR I=1 TO N : POKE addr+I, val : NEXT` — egy memóriablokkot tölt fel ugyanazzal a byte-tal, de egyetlen blokkban. Kiváló területek törlésére vagy táblák előfeltöltésére.

Mező	Leírás
Operandus	`count,value` — pl. `256,0` 256 byte-ot tölt fel nullával

Expert szintaxis:

```
.fill 256, $00
```

Generált ASM:

```
    .fill 256, $00
```

Kifejezés-szintaxis: a `count` és a `value` is elfogad aritmetikai kifejezéseket. Hivatkozhatsz CONST nevekre, használhatsz hex/bináris literálokat, és hívhatsz beépített matematikai függvényeket:

Kifejezés	Jelentés
`TILE_COUNT, $00`	count egy CONST-ból, value hex literál
`40*25, 0`	inline szorzás
`round(sin(PI/4)*255), $80`	trigonometria

Beépített függvények: `sin()`, `cos()`, `round()`, `max(a,b)`, `min(a,b)`, `abs()`, `PI` konstans

Operátorok: `+ - * /` Literálok: `$FF` (hex), `%10110000` (bináris) Alsó/felső byte: `lo(expr)`, `hi(expr)`

Méret: a count érték byte-ban.

ALIGN

Előrébb csúsztatja az aktuális címet a következő tiszta határra nulla kitöltő byte-ok beszúrásával. A C64 megköveteli, hogy a sprite adat 64 byte-os határon kezdődjön — az `ALIGN 64` ezt automatikusan intézi.

Mező	Leírás
Határ	Igazítási érték — pl. `64` (sprite határ), `256` (lap), `$2000` (bitmap)

Expert szintaxis:

```
.align 64
.align $2000
```

Generált ASM:

```
    ; ALIGN 64 → $0840 (12 byte kitöltés)
```

Méret: dinamikus — az aktuális program counter pozíciójától függ.

> **Tipp:** használj `ALIGN 64`-et a sprite adat előtt, `ALIGN 256`-ot a lap-igazított táblákhoz.

TEXT

Mint a **PRINT AT** — szöveget ír közvetlenül a C64 képernyőjére adott oszlopban és sorban, a KERNAL használata nélkül. Karakterenként egy LDA/STA párt generál, a `$0400`-nál lévő screen RAM-ot célozva.

Mező	Leírás
Text	A megjelenítendő string
X	Oszlop (0–39)
Y	Sor (0–24)
Label (opcionális)	Címkét rendel, amely a kiszámolt képernyőcímre mutat
Lowercase charset	Jelölőnégyzet — lásd lentebb

Karakterkészlet módok:

A C64-nek két, futásidőben választható karakterkészlete van:

Mód	$D018 bit 1	Nagybetűs bemenet	Kisbetűs bemenet
Nagybetű/grafika (alap)	0	`A`–`Z` → screen code $01–$1A ✓	szintén nagybetűként kezelve
Kisbetű/nagybetű (CHARSET lower után)	1	`A`–`Z` → $01–$1A (nagybetű)	`a`–`z` → $41–$5A (kisbetű) ✓

- **Nagybetűs karakterkészlet (alap, jelölőnégyzet üres):** azt írd, amit nagybetűvel akarsz látni. A `"HELLO"` `HELLO`-ként jelenik meg. A kisbetűs bemenet nagybetűs screen code-okra képződik.
- **Kisbetűs karakterkészlet (jelölőnégyzet bepipálva):** pontosan azt a kis/nagybetűzést írd, amit látni akarsz. `"hello"` → kisbetűs megjelenítés, `"HELLO"` → nagybetűs. Futásidejű karakterkészlet-váltást igényel a képernyőre írás előtt (használd a **CHARSET lower** makrót).

Generált ASM (nagybetűs mód, `"HELLO"`):

```
    LDA #$08      ; 'H' screen code $08
    STA $0400
    LDA #$05      ; 'E' screen code $05
    STA $0401
    ...
```

Expert szintaxis:

```
.text 0, 2, "HELLO"           ; nagybetűs karakterkészlet (alap)
.text 0, 2, "hello", lower    ; kisbetűs karakterkészlet
```

A karakterek **screen code-ként** kódolódnak (nem PETSCII). **Méret:** `text.length × 5` byte (LDA + STA karakterenként).

STRING

Mint egy string **POKE-olása** bármely memóriacímre futásidőben. LDA/STA párokat generál, amelyek minden karakter screen code-ját egymást követő címekre másolják.

Mező	Leírás
Text	A kiírandó string
Address	Cél memóriacím — `$C000` hex vagy **címkenév**
Label (opcionális)	Címkét rendel, amely a célcímre mutat
Shift	Hex érték (00–FF), amely minden screen code byte-hoz hozzáadódik (pl. `$80` = inverz)
Lowercase charset	Jelölőnégyzet — ugyanaz a szemantika, mint a TEXT-nél (lásd a TEXT szakaszt)

Expert szintaxis:

```
.string $C000, "HELLO"                 ; nagybetűs karakterkészlet (alap)
.string $C000, "hello", lower          ; kisbetűs karakterkészlet
.string $C000, "HELLO", 80             ; shifttel (inverz)
.string $C000, "hello", 80, lower      ; shift + kisbetű
.string $C000, "HELLO" :my_string      ; macroLabel-lel
```

Generált ASM:

```
    LDA #$08      ; 'H' screen code
    STA $C000
    LDA #$05      ; 'E' screen code
    STA $C001
    ...
```

A karakterek **screen code-ként** kódolódnak (nem PETSCII). Az opcionális **Shift** érték minden byte-hoz hozzáadódik, pl. `$80` az inverz videóhoz. **Méret:** `text.length × 5` byte (egy LDA + egy STA karakterenként).

DATA

Mint egy **POKE hurok** — nyers byte-ok listáját írja egy memóriacímre futásidőben, byte-onként egy LDA/STA párral.

Mező	Leírás
Bytes	Vesszővel elválasztott byte-értékek
Address	Cél memóriacím — `$C000` hex vagy **címkenév**
Label (opcionális)	Címkét rendel, amely a célcímre mutat

Expert szintaxis:

```
.data $C000, $01, $02, $03          ; hex cím
.data my_buf, $01, $02, $03         ; címke-cím
.data $C000, $01, $02, $03 :mydata  ; macroLabel-lel
```

Generált ASM:

```
    LDA #$01
    STA $C000
    LDA #$02
    STA $C001
    ...
```

Méret: `byte_count × 5` byte (egy LDA + egy STA byte-onként).

> **2.1:** a bájtok címkét is nevezhetnek: a `.data $2010, <target, >target` a `target` címét írja, később definiált címkéét is. A makró `:név` címkéje olyan célcímre is mutathat, amely maga is címke.

RAWBYTES

Mint a **DATA, amely közvetlenül a memóriába töltődik** — egyáltalán nincs futásidejű kód. A byte-ok a PRG betöltésének pillanatától jelen vannak, még mielőtt a kódod elindulna. Ezt használd sprite adathoz, pályatérképekhez, lookup táblákhoz, bármihez, aminek csak egy konkrét címen kell lennie.

Mező	Leírás
Bytes	Vesszővel elválasztott byte-értékek
Address	Cél memóriacím — `$C000` hex vagy **címkenév**
Label (opcionális)	Címkét rendel, amely a célcímre mutat

Expert szintaxis:

```
.rawbytes $C000, $00, $00, $00      ; hex cím
.rawbytes sprite_data, $00, $00     ; címke-cím
.rawbytes $0C50, $00, $00 :nev      ; macroLabel-lel – más kód használhat LDA nev,X-et
```

Méret a kódban: 0 byte. Az adat a megadott címre kerül a kimenetben.

> **DATA vs RAWBYTES:** a DATA LDA/STA kódot generál, amely futásidőben másolja a byte-okat (lassabb, de működik, ha az adatnak dinamikusnak kell lennie). A RAWBYTES egyszerűen elhelyezi a byte-okat — nincs kód, azonnali, nulla költség.

> **2.1:** a bájtok címkét is nevezhetnek (`.rawbytes $2000, <target, >target` — pointertábla); a `:név` címke célcíme lehet címke, később definiált is. Makró puffer-argumentumaként a bájtok saját helyet kapnak a program utáni pufferekben.

RAWTEXT

Mint a RAWBYTES, de szöveghez — a stringet screen code-ként kódolja, és a byte-okat fix címre helyezi **futásidejű kód nélkül**. A szöveg a PRG betöltésének pillanatában készen áll a memóriában.

Mező	Leírás
Text	A kódolandó string
Address	Cél memóriacím — `$C000` hex vagy **címkenév**
Label (opcionális)	Címkét rendel, amely a célcímre mutat
Shift	Hex érték (00–FF), amely minden screen code byte-hoz hozzáadódik (pl. `$80` = inverz)
Lowercase charset	Jelölőnégyzet — ugyanaz a szemantika, mint a TEXT-nél (lásd a TEXT szakaszt)

Expert szintaxis:

```
.rawtext $C000, "HELLO"                ; nagybetűs karakterkészlet (alap)
.rawtext $C000, "hello", lower         ; kisbetűs karakterkészlet
.rawtext $C000, "HELLO", 80            ; shifttel (inverz)
.rawtext $C000, "hello", 80, lower     ; shift + kisbetű
.rawtext $0400, "HELLO" :my_text       ; macroLabel-lel
```

Generált ASM:

```
; .rawtext "HELLO" -> $C000
; $C000
    .byte $08, $05, $0C, $0C, $0F   ; H E L L O (nagybetűs screen code-ok)

; .rawtext "hello", lower -> $C000
; $C000
    .byte $48, $45, $4C, $4C, $4F   ; h e l l o (kisbetűs screen code-ok, $41–$5A tartomány)
```

Méret a kódban: 0 byte. Az adat a megadott címre kerül a kimenetben.

> **STRING vs RAWTEXT:** a STRING LDA/STA kódot generál, amely futásidőben másolja a szöveget. A RAWTEXT betöltéskor beépíti a byte-okat a PRG-be — nincs kód, nincs várakozás.

PETSCII

Mint a **RAWBYTES, de KERNAL kimenethez** — a stringet PETSCII byte-ként kódolja (kompatibilis a `$FFD2`-nél lévő CHROUT-tal), és fix címre helyezi futásidejű kód nélkül. Ezt használd, amikor karaktereket akarsz kiírni `JSR $FFD2`-vel egy hurokban; az új `PRINT` makró ugyanezt a kódolót és kisbetűs jelölőnégyzet-viselkedést használja.

> **PETSCII vs screen code-ok:** a PETSCII és a screen code-ok két különböző kódolás. A screen code `$01` = az A betű; a PETSCII `$41` = az A betű (CHROUT-on át). PETSCII-t csak KERNAL-on keresztüli nyomtatáshoz használj; a screen RAM-ba közvetlen íráshoz használj TEXT/STRING/RAWTEXT-et.

Mező	Leírás
Text	PETSCII byte-ként kódolandó string
Address	Cél memóriacím — `$C000` hex vagy **címkenév**
Label (opcionális)	Címkét rendel, amely a célcímre mutat
Lowercase PETSCII	Jelölőnégyzet — lásd lentebb

Karakterkészlet módok:

Mód	Nagybetűs bemenet (`A – Z`)	Kisbetűs bemenet (`a – z`)
Nagybetű (alap, üres)	`$41` – `$5A` (PETSCII nagybetű CHROUT-on át)	szintén `$41` – `$5A`-ra képezve
Kisbetű (bepipálva)	Az alfabetikus betűk átképződnek, hogy a látható kis/nagybetűzés konzisztens maradjon a kisbetű/nagybetű karakterkészleten	Ugyanaz a szabály

Expert szintaxis:

```
.petscii $C000, "HELLO"              ; nagybetűs PETSCII (alap)
.petscii $C000, "hello", lower       ; kisbetűs PETSCII ($61–$7A)
.petscii $C000, "HELLO", null        ; null lezáróval
.petscii $C000, "hello", lower, null ; kisbetű + null lezáró
.petscii $C000, "HELLO" :my_msg      ; macroLabel-lel
```

Generált byte-ok (nagybetű, `"HELLO"`):

```
; $C000
    .byte $48, $45, $4C, $4C, $4F   ; H E L L O (PETSCII $41–$5A tartomány)
```

Méret a kódban: 0 byte. Az adat a célcímre kerül halasztott adat-szegmensként (mint a RAWBYTES-nál).

Null lezáró: pipáld be a *„`$00` hozzáfűzése (null lezáró)"* jelölőnégyzetet, hogy automatikusan egy `$00` byte kerüljön a szöveg után. Ideális null-lezárt hurkokhoz:

```
    LDX #$00
loop:
    LDA msg,X
    BEQ done        ; a $00 leállítja a hurkot
    JSR $FFD2
    INX
    BNE loop
```

```
done:
    RTS
```

Kódolási szabályok:

Bemenet	Nagybetűs mód	Kisbetűs mód
`A` – `Z`	`$41` – `$5A`	`$61` – `$7A`
`a` – `z`	`$41` – `$5A` (kényszerített nagybetű)	`$41` – `$5A`
Szóköz, számjegyek, írásjelek (32–126)	változatlanul	változatlanul
Sortörés	`$0D` (RETURN)	`$0D`
Egyéb	`$20` (szóköz)	`$20`

> **Tipp:** PETSCII-t olyan adathoz használj, amelyet CHROUT-on (`$FFD2`) keresztül fogsz kiírni. A screen RAM-ba közvetlen íráshoz inkább STRING-et vagy RAWTEXT-et.

CHARSET

A VIC-II karakter-ROM-ot váltja nagybetű/grafika mód (C64 alap) és kisbetű/nagybetű mód közt, a `$D018` 1. bitjének futásidejű módosításával.

Mező	Leírás
Mód	**Lowercase** — engedélyezi a kisbetű/nagybetű karakterkészletet; **Uppercase** — visszaállítja az alapértelmezett nagybetű/grafika karakterkészletet

Expert szintaxis:

```
.charset lower    ; váltás kisbetűs karakterkészletre
.charset upper    ; vissza a nagybetű/grafika karakterkészletre
```

Generált ASM:

Kisbetűs mód:

```
    LDA $D018
    ORA #$02      ; 1. bit be → kisbetű/nagybetű ROM $1800-nál
    STA $D018
```

Nagybetűs mód:

```
    LDA $D018
    AND #$FD      ; 1. bit törlése → nagybetű/grafika ROM $1000-nál
    STA $D018
```

Méret: 8 byte (LDA abs + ORA/AND imm + STA abs).

Miért ORA/AND közvetlen írás helyett? A `$D018` a screen RAM helyét is vezérli (7–4. bitek). Csak az 1. bit kapcsolása megőrzi a regiszter többi részét.

Tipikus munkafolyamat:

```
    CHARSET lower             ; váltás kisbetűs karakterkészletre
    TEXT 0, 0, "hello world"  ; [jelölőnégyzet: Lowercase charset]
    ...
    CHARSET upper             ; végén állítsd vissza az alapot
```

Vagy expert módban:

```
.charset lower
.text 0, 0, "hello world", lower
.charset upper
```

Expert módban a `.charset` blokk mostantól a mód-legördülőn keresztül is round-trip-el, így a blokk-előnézet és az exportált forrás szinkronban marad.

> **Megjegyzés:** a CHARSET makró csak a VIC karakter-ROM pointerét változtatja. Nem hívja a `$E544` -et (KERNAL charset init). A legtöbb esetben ez elég; csak akkor hívj előtte `JSR $E544` -et, ha azt akarod, hogy a KERNAL saját print rutinjai is figyeljék a változást.

CHARDEF

Egyetlen 8×8-as egyéni karaktert definiál egy RAM-alapú karakterkészletben. Inline futásidejű kódot bocsát ki, amely 8 byte-ot másol a `base + index * 8` címre végrehajtáskor — nincs szükség előtte címkére vagy `ORG` -ra.

Mező	Leírás
Charset base	A RAM karakterkészlet báziscíme (alap `$3800`). Igazítottnak kell lennie, hogy a VIC-II lássa (lásd lentebb).
Character index	Melyik karakter-slotot definiáljuk újra, 0–255. `65` = 'A' az alapértelmezett screen code elrendezésben.
8 byte	Vesszővel elválasztott bitmap sorok, fentről lefelé. Minden byte 7. bitje = a legbaloldalibb pixel.

Expert szintaxis:

```
.chardef $3800, 65, $18,$3C,$66,$7E,$66,$66,$66,$00
```

Generált ASM (8 × `LDA #b` / `STA target+n` , összesen 40 byte):

```
    LDA #$18
    STA $3A08
    LDA #$3C
    STA $3A09
    LDA #$66
    STA $3A0A
    LDA #$7E
    STA $3A0B
    LDA #$66
    STA $3A0C
    LDA #$66
    STA $3A0D
    LDA #$66
    STA $3A0E
```

```
    LDA #$00
    STA $3A0F
```

Célcím: `$3800 + 65 * 8 = $3A08`. Fordítási időben számolódik, és be van égetve az STA operandusokba.

Méret: 40 byte karakterenként (8 × 5).

Tipikus munkafolyamat:

```
    ; VIC-II a $3800-nál lévő RAM charsetre mutasson
    LDA $D018
    AND #$F1
    ORA #$0E       ; 3-1. bitek = %111 → charset a bank+$3800-nál
    STA $D018

    CHARDEF $3800, 65, $18,$3C,$66,$7E,$66,$66,$66,$00   ; 'A' újradefiniálása
    CHARDEF $3800, 66, $7C,$66,$66,$7C,$66,$66,$7C,$00   ; 'B' újradefiniálása
    ; ... további CHARDEF hívások minden egyéni karakterhez
```

Mikor CHARDEF vs alternatívák:

Megközelítés	Használd, ha
CHARDEF	Néhány egyéni karakter kell (mondjuk 1–20). Futásidejű költség 40 byte darabonként.
RAWBYTES @ $3800	Teljes egyéni karakterkészleted van (256 karakter). Összesen 2 KB adat, nincs futásidejű másolás.
INCBIN "charset.bin" @ $3800	Külső karakterkészlet-fájl (a Character Editorral építve). A legtisztább opció.
Charset Canvas + INCBIN	Teljes 256 karakteres bitmap egyetlen 128×128-as képként megfestve.

> **Igazítási emlékeztető:** a VIC-II a charset bázist `$0800` többszörösén várja. Érvényes bankok: `$0000`, `$0800`, `$1000`, … , `$3800` (az aktuális 16 KB-os VIC bankon belül). A RAM charsetek jellemzően a `$2000`, `$2800`, `$3000` vagy `$3800` címen élnek.

BOX_HIT

Tengely-igazított bounding-box (AABB) ütközésteszt két, 4 byte-os zero-page struktúrával leírt téglalap közt. Az eredményt az akkumulátorban adja vissza: **A = 1** átfedésnél, **A = 0** egyébként. Tiszta inline assembly, nincs szubrutinhívás.

Mező	Leírás
Box1 ZP cím	Az első box 4 byte-os struktúrájának zero-page bázisa (alap `$FB`).
Box2 ZP cím	A második box 4 byte-os struktúrájának zero-page bázisa (alap `$F7`).

Struktúra elrendezés (4 byte boxonként, előjel nélküli 8 bites koordináták):

Offset	Mező
`+0`	Left

Offset	Mező
`+1`	Top
`+2`	Right
`+3`	Bottom

Expert szintaxis:

```
.box_hit $FB, $F7
```

Generált ASM (30 byte, teljesen PC-relatív — nincs szubrutin, nincs abszolút ugrás):

```
    LDA $FD        ; b1 right
    CMP $F7        ; b2 left
    BCC no         ; R1 < L2 → nincs találat
    LDA $F9        ; b2 right
    CMP $FB        ; b1 left
    BCC no
    LDA $FE        ; b1 bottom
    CMP $F8        ; b2 top
    BCC no
    LDA $FA        ; b2 bottom
    CMP $FC        ; b1 top
    BCC no
    LDA #$01       ; találat
    BNE done       ; feltétel nélküli (A ≠ 0)
no: LDA #$00
done:
```

Méret: 30 byte.

Tipikus munkafolyamat:

```
    ; Játékos sprite box a $FB..$FE címen
    LDA sprite_x            ; L
    STA $FB
    LDA sprite_y            ; T
    STA $FC
    CLC
    ADC #23                 ; +23 → B (24 pixel magas sprite)
    STA $FE
    LDA $FB
    CLC
    ADC #23                 ; R
    STA $FD

    ; Ellenség box a $F7..$FA címen (hasonlóan feltöltve)
    ; ...

    BOX_HIT $FB, $F7        ; játékos vs ellenség teszt → A = 0 vagy 1

    CMP #$01
    BNE no_collision
    JSR handle_hit
no_collision:
```

Megkötések:

- Mindkét zero-page címnek `≤ $FC`-nek kell lennie (minden box 4 egymást követő byte-ot igényel: `zp`, `zp+1`, `zp+2`, `zp+3`).
- A koordináták **előjel nélküli 8 bitesként** kezelődnek (0–255). Ezen tartományon kívüli előjeles sprite-koordinátáknál tárolás előtt normalizálj.
- A két box átfedhet a ZP térben, ha akarod, de általában 8 különálló byte-ot szeretnél.

Miért nem szubrutin? Az inline generálás elkerüli a JSR/RTS overhead-et (14+ ciklus), és a tesztet a cache-ben tartja a szoros game hurkokhoz. Ha sok párt kell tesztelned, tegyél kézzel egy saját `JSR box_hit_sub`-ot egyetlen BOX_HIT blokk köré.

Összehasonlítás a UB `box_hit()` függvényével: az Ultimate Basic ugyanezt a 6502 logikát csomagolja futásidejű függvényként, amely egy változóba tér vissza. VA-ban a BOX_HIT-et inline helyezed oda, ahol a tesztre szükség van; az eredmény az `A`-ban van.

INCBIN

Mint a **BLOAD BASIC-ben** — beemel egy külső binárisfájlt (`.bin`, `.prg`, `.sid`, `.raw`), és közvetlenül beágyazza a lefordított PRG-be a megadott címre.

Mező	Leírás
File	Tallózz egy `.bin`, `.prg`, `.sid` vagy `.raw` fájl kiválasztásához
Address	Cél betöltési cím (pl. `$C000`)

Expert szintaxis:

```
.incbin "music.bin", $C000
```

Generált ASM megjegyzés:

```
    ; INCBIN "music.bin" @ $C000 (2048 byte)
   .byte $01, $02, ...
```

Méret a kódban: 0 byte (halasztott adat-szegmens). A bináris a megadott címre ágyazódik.

KOALA

Egy **Koala Painter** képet (`.koa` / `.kla`, 160×200 multicolor) tölt be és jelenít meg. A képadat az `INCBIN`-hez hasonlóan fix címre ágyazódik (a fájl 2 bájtos `$6000` betöltési címét levágja), maga a blokk pedig egy 86 bájtos megjelenítő rutinra fordul, ami a blokk helyén fut.

Mező	Leírás
File	Tallózz egy `.koa` / `.kla` fájlt (10003 bájt, vagy 10001 betöltési cím nélkül)

Mező	Leírás
Address	A bitmap címe: `$6000` (alapértelmezett, Koala natív, VIC bank 1, screen RAM `$4400`) vagy `$2000` (VIC bank 0, screen RAM `$0400`)

A megjelenítő rutin a screen RAM részt a képernyőre, a color RAM részt a `$D800`-ra másolja, a háttér bájtot a `$D021`-be írja, a `$DD00`-ban kiválasztja a VIC bankot (a port többi bitjét megtartva), és bekapcsolja a multicolor bitmap módot (`$D011 = $3B`, `$D016 = $18`, `$D018 = $18`). Más címet elutasít: a többi 8K-s terület a BASIC/KERNAL ROM alatt van (a másoló ciklus a ROM-ot olvasná), vagy a karakter-ROM árnyékba esik.

Expert szintaxis:

```
.koala "picture.koa"
.koala "picture.koa", $2000
```

Példa — kép megjelenítése és várakozás:

```
* = $0801
.koala "picture.koa"
wait:
    JMP wait
```

Méret a kódban: 86 byte (megjelenítő rutin); a 10001 bájt képadat a megadott címre kerül. A Hires szerkesztő **Files** → **Koala mentés (.koa)…** menüpontja kompatibilis fájlt készít.

LEVEL / LEVEL_ROW

Egy **Pálya szerkesztő** projektet tesz a programba, és pályasorokat másol egy képernyő-pufferbe — ezek egy görgetős játék építőkövei. Teljes, joystickkal görgetett pályáért lásd a `samples/level-demo.asm`-et.

LEVEL

INCBIN-szerű blokk, ami **0 byte kódot** generál. A Pálya szerkesztő projektből (a **Files** → **Pálya mentése** által mentett `.bin`) csak az **elhelyezett** képernyők kerülnek a memóriába a megadott címtől (alapértelmezés `$2000`), hely-sorrendben, képernyőnként 2000 bájt (1000 screen code + 1000 szín). Az azonos képernyők egyszer tárolódnak, az üres helyek egy közös üres képernyőn osztoznak. A képernyők után jönnek a pointer-táblák. Egy pálya legfeljebb 7 × 10 képernyős lehet.

Expert szintaxis:

```
.level "level.bin"
.level "level.bin", $4000
```

Definiált név	Jelentés
`LEVEL_SLOT_LO` / `LEVEL_SLOT_HI`	Az egyes helyek képernyőjének címe (soronként, soronként `LEVEL_W` hely)
`LEVEL_ROW_SLOT`	Pálya karaktersor → a képernyősorának első helye

Definiált név	Jelentés
`LEVEL_ROW_LO/HI`, `LEVEL_CROW_LO/HI`	Pálya karaktersor → a screen code / szín sorának eltolása a képernyőn belül
`LEVEL_COL_SLOT`, `LEVEL_COL_OFS`	Nézet kezdőoszlop → képernyőoszlop és oszlop a képernyőn belül
`LEVEL_W`, `LEVEL_H`	Pályaméret képernyőkben
`LEVEL_COLS`, `LEVEL_ROWS`	Pályaméret karakterekben
`LEVEL_MAX_X`, `LEVEL_MAX_Y`	Egy 40×25-ös nézet legnagyobb pixeles görgetési pozíciója
`LEVEL_BG`, `LEVEL_MC1`, `LEVEL_MC2`, `LEVEL_MULTICOLOR`	A pálya háttér- és multicolor színei, módja
`LEVEL_DATA`, `LEVEL_SIZE`, `LEVEL_END`	A pályaadat kezdete, mérete és az utána következő első szabad cím
`LEVEL_OBJ_X_LO/HI`, `LEVEL_OBJ_Y_LO/HI`, `LEVEL_OBJ_TYPE`, `LEVEL_OBJ_PARAM`	Objektumtábla (a Pálya szerkesztő objektumrétege): pozíció pálya-pixelben, típus és paraméterbájt, görgetési sorrendben — `LEVEL_OBJ_COUNT` elem (max. 255)
`LEVEL_OBJ_T_START` … `LEVEL_OBJ_T_TRIGGER`	Objektumtípus-számok: START 0, ENEMY 1, ITEM 2, DOOR 3, EXIT 4, TRIGGER 5
`LEVEL_START_X`, `LEVEL_START_Y`	Az első Kezdőpont objektum pozíciója (csak ha van ilyen a pályán)
`LEVEL_ATTR`, `LEVEL_ATTR_SOLID` … `LEVEL_ATTR_COLLECT`	Karakter-attribútum tábla (screen code-onként egy jelzőbájt; csak ha a pályán van attribútum) és a bitek: 1, 2, 4, 8
`LEVEL_BORDER`, `LEVEL_SCR_COLORS`, `LEVEL_SCR_BG/BORDER/MC1/MC2`	A pálya kerete; helyenkénti színtáblák (soronként, mint a `LEVEL_SLOT_*`), ha egy képernyőnek saját színei vannak (`LEVEL_SCR_COLORS` = 1)

Tipikus védelem: `.assert LEVEL_END <= $9200, "level data runs into the code"`.

LEVEL_ROW

A pálya egy 40 karakteres sorát másolja egy pufferbe: az **X**-ben megadott pályasort, a `col` változóban tárolt pályaoszloptól kezdve, a `dst` pointerben (`dst` / `dst+1`, lo/hi) tárolt címre. A harmadik, `color` argumentummal a screen code-ok helyett a színeket másolja (color RAM-hoz). A két képernyőre átnyúló 40 karakteres ablakot önmódosító másolóciklus kezeli (~14 ciklus/bájt). Az X megmarad, az A és az Y elvész. Kell hozzá egy `.level` blokk a programban.

Expert szintaxis:

```
.level_row dst_lo, view_x
.level_row dst_lo, view_x, color
```

Méret a kódban: 132 byte — ha egy elágazásnak át kell ugrania, használj `LBEQ` / `LBNE`-t.

SID

Mint a **BLOAD zenéhez** — betölt egy `.sid` fájlt a PRG-be, és automatikusan kiolvassa az Init és Play címeket a fejlécből. Hívd az Init-et egyszer induláskor, majd hívd a Play-t az IRQ kezelődből minden frame-ben.

Mező	Leírás
File	Tallózz egy `.sid` fájl kiválasztásához
Custom address (opcionális)	Felülírja a SID natív betöltési címét (pl. `$1000`). Hagyd üresen a SID fejlécből származó cím használatához.

A blokk megjeleníti:

- **Title / Author** a SID fejlécből
- **Load address** — hova kerül az adat a memóriában (a felülírás utáni tényleges cím)
- **Init address** — ezt hívd JSR-rel a zene inicializálásához (relokáció esetén korrigálva, ha egyéni címet használsz)
- **Play address** — ezt hívd JSR-rel minden frame-ben egy IRQ kezelőben (relokációhoz korrigálva)
- Egy **(relocated)** jelvény jelenik meg, amikor egy egyéni cím elmozdítja az adatot az eredeti pozíciójából

Expert szintaxis:

```
.sid "Ikari_Warriors.sid"
.sid "Ikari_Warriors.sid", $1000
```

Generált ASM megjegyzés:

```
    ; SID "Ikari_Warriors.sid" @ $1000  Init:$1000  Play:$1006  (4096 byte)
```

Méret a kódban: 0 byte inline. A SID bináris a megadott címre kerül halasztott chunkként a PRG-ben.

> **Fontos:** a legtöbb SID fájl beégetett belső abszolút címeket tartalmaz. Csak akkor relokálhatók, ha az egész bináris ugyanazzal az offszettel tolódik el. Ha egy SID belső ugrásokat tartalmaz `$10xx`-re, akkor `$1000`-en kell maradnia — más címre mozgatva ezek a belső hivatkozások elromlanak.

> **Tipikus használat:** tegyél egy ORG blokkot a SID blokk elé a címének beállításához. Hívd az Init-et egyszer induláskor, majd hívd a Play-t minden frame-ben egy raszter IRQ kezelőből — vagy bízd mindkettőt a SID_PLAYER makróra.

Címkék: a SID blokk megadja a `SID_LOAD` (ahova az adat kerül), a `SID_INIT` és a `SID_PLAY` címkét (az init és a play címe, egyedi címnél azzal együtt eltolva). A `SID_PLAY` hiányzik, ha a zenének nincs play címe (saját IRQ-t telepítő RSID). A saját kódodban is használhatod őket, pl. `JSR SID_PLAY` egy IRQ kezelőben.

SID_PLAYER

Egyetlen blokkal lejátssza a **SID** makró zenéjét: inicializálja, majd egy raszter interruptból folyamatosan hívja a play rutinját, így a programod közben a saját dolgát végezheti.

Mező	Leírás
Song	Az init rutinnak A-ban átadott subtune száma (alapból `0`)
Raster	Az interrupt rasztersora, `$00` – `$FF` (alapból `$FA`)

Mit csinál: `SEI`, `LDA #song` / `JSR SID_INIT`, kikapcsolja a CIA1 időzítő IRQ-t (`$DC0D`), az IRQ vektort (`$0314/$0315`) a beépített handlerére állítja, beállítja a rasztersort (`$D012`, a `$D011` 8. bitje törölve) és a raszter IRQ-t

(`$D01A`), `CLI` , majd átugorja a handlert. A handler nyugtázza az interruptot (`ASL $D019`), meghívja a `JSR SID_PLAY` -t, és a KERNAL-ban folytatja (`JMP $EA31` , ez a billentyűzetet is olvassa).

Expert szintaxis:

```
.sid "Ikari_Warriors.sid"
.sid_player
.sid_player 1, $80
```

Méret a kódban: 58 byte.

> **Megjegyzés:** kell hozzá egy SID blokk a programban (`SID_INIT` és `SID_PLAY`), különben a fordítás hibával megáll. A CIA1 időzítő IRQ helyett a KERNAL képkockánként egyszer, a raszter IRQ-ból fut — a billentyűzet és a `GETIN` továbbra is működik.

INCLUDE

Mint a **MERGE BASIC-ben** — beemel egy másik fájlt, és a blokkjait inline kifejti ezen a pozíción. Kiváló újrahasználható szubrutin-könyvtárakhoz. A beemelt blokkok csak olvashatók az aktuális projektben.

Kétféle fájl támogatott:

- **Visual Assembler projekt** (`.json`) — a projekt blokkjai változatlanul beszúródnak.
- **Sima assembly forrás** (`.inc` , `.asm` , `.s`) — a fájl szövegként olvasódik, és ugyanúgy parse-olódik, mint az expert mód. Minden fordításkor a fájl újraolvasódik a lemezről (az igazság forrása = a fájl), így bármilyen szerkesztővel külsőleg szerkesztheted.

Mező	Leírás
File	Tallózz egy `.json` projekt vagy `.inc` / `.asm` / `.s` assembly forrás kiválasztásához
Load address (opcionális)	Ha beállított (hex, pl. `C000`), a beemelt blokkok arra a címre kerülnek — egy szintetikus `ORG` szúródik eléjük, felülírva a beemelt fájlon belüli bármely ORG blokkot. Hagyd üresen, hogy a beemelt fájl saját ORG blokkjai vezéreljék az elhelyezést.

Expert-mód szintaxis:

```
.include "library.json"
.include "macros.inc", $1500
.include "sprites.asm"
```

- A **fájlkiterjesztés kötelező expert módban** — egy csupasz név, mint a `.include "macros"` , `.include "macros.json"` -ként kezelődik.
- Útvonal-feloldás: először a projektfájl mellett próbálja (relatív), majd az app beépített `samples/` könyvtárára esik vissza.

Generált ASM (nincs cím-felülírás):

```
    ; .include "library.json" — 12 blokk
   ... (a kifejtett blokkok következnek)
```

Generált ASM (`C000` betöltési címmel):

```
    ; .include "library.json" @ $C000 — 12 blokk
   *=$C000
   ... (a kifejtett blokkok következnek)
```

> **Tipp:** használd az INCLUDE-ot újrahasználható szubrutin-könyvtárak építésére, amelyeket megoszthatsz projektek közt. Az `.inc` / `.asm` / `.s` fájlok akkor a legjobbak, ha sima szövegszerkesztőben akarod szerkeszteni a könyvtárat, vagy más 6502 assemblerekkel megosztani; a `.json` akkor, ha a könyvtár magában a Visual Assemblerben készült. Állíts be betöltési címet, ha a könyvtárnak nincs saját ORG-ja, vagy ha felül akarod írni az alapértelmezett elhelyezését.

> **2.1:** a nem található include hiba, és az is, ha egy fájl önmagát emeli be (közvetlenül vagy mélyebben). A beemelt fájl saját `.include`, `.incbin` és `.sid` sorai is betöltődnek a saját mappájához képest, és a makróhívásai is kibomlanak — legfeljebb 16 szint mélyen.

TABLE

Mint a **DIM egy konkrét címen** — nevet ad egy lookup táblának, és beállítja, hol él a memóriában. Tegyél utána BYTE, WORD vagy FILL blokkokat a tábla tartalmának definiálásához.

Mező	Leírás
Name	Címke-azonosító a táblához (pl. `color_table`)
Address	Fix cím, ahol a tábla kezdődik (pl. `$C000`)

Expert szintaxis:

```
.table color_table $C000
```

Generált ASM:

```
color_table:
```

A program counter a megadott címre ugrik. Tegyél BYTE/WORD/FILL blokkokat a TABLE után a tartalom feltöltéséhez.

Méret: 0 byte.

ORG

Beállítja, hova kerül a program (vagy egy szakasza) a memóriában — mintha egy kezdőcímet választanál, mielőtt gépi kódot gépelnél. Minden programnak legalább egy ORG kell. A szabványos C64 BASIC-ből betölthető kezdet a `$0801`.

Mező	Leírás
Address	Az új origin cím (pl. `0801` HEX-ben, vagy `2049` DEC-ben)
HEX / DEC	A cím-bemenet váltása hexadecimális és decimális megjelenítés közt

Expert szintaxis:

```
* = $C000
```

Generált ASM:

```
* = $C000
```

Méret: 0 byte. Az ORG blokk maga nem generál gépi kódot.

Minden ORG blokk új szakaszt kezd. Az utána következő blokkok arról a címről kezdve assembleálódnak. A PRG exportjakor minden szakasz egyetlen fájlba olvad — a szakaszok közti rések nullákkal töltődnek fel.

Példa — kód a `$0801`-en, adattábla a `$C000`-en:

```
* = $0801
    LDX #$00
loop:
    LDA $C000,X
    STA $D800,X
    INX
    BNE loop
    RTS

* = $C000
    .byte $01, $02, $03, ...
```

> **Tipp:** minden programnak ORG blokkal kell kezdődnie. C64 BASIC-ből betölthető programnál a tipikus kezdőcím a `$0801` (2049 decimális). Ha a **BASIC SYS stub** engedélyezett, az assembler egy rövid BASIC sort tesz a `$0801`-re, és a kódod a `$080D`-n kezdődik.

LOOP / NEXT

Mint a `FOR X=N TO 1 STEP -1 : ... : NEXT X` BASIC-ben — N-től 1-ig számol lefelé az X vagy Y regiszterrel. Tegyél le egy LOOP blokkot, tedd az utasításaidat közte és a NEXT közé, és automatikusan a helyes számú alkalommal hurokban fut.

LOOP

Mező	Leírás
Register	`X` vagy `Y` — a számláló regiszter
Count	Ciklus-iterációk száma (hex vagy decimális, pl. `0A` = 10)
Label	Automatikusan generált loop címke (pl. `loop0`)

Expert szintaxis:

```
.loop X, 10, loop0
```

Generált ASM:

```
    LDX #$0A
loop0:
```

Méret: 2 byte (LD_ opcode + immediate operandus).

NEXT

Mező	Leírás
Register	Automatikusan a LOOP regiszteréhez illesztve
Label	Automatikusan a LOOP címkéjéhez kötve

Expert szintaxis:

```
.next loop0
```

Generált ASM:

```
    DEX
    BNE loop0
```

Méret: 3 byte (DEX + BNE + branch offset).

Példa — 10 képernyőcella törlése:

```
    LDX #$0A
loop0:
    LDA #$20        ; szóköz karakter
    STA $0400,X
    DEX
    BNE loop0
```

FOR / ENDF

Mint a `FOR X=0 TO N-1 : ... : NEXT X` BASIC-ben — 0-tól számol *felfelé*. Ideális, amikor előre haladó indexre van szükséged, pl. egy string vagy tömb végigjárásához.

FOR

Mező	Leírás
Register	`X` vagy `Y` — a számláló regiszter
Count	Ciklus-limit (hex vagy decimális, pl. `$12` = 18). Az X/Y 0-tól limit-1-ig fut.
Label	Automatikusan generált loop címke (pl. `for0`)

Expert szintaxis:

```
.for X, $12, for0
```

Generált ASM:

```
    LDX #$00
for0:
```

Méret: 2 byte (LD_ opcode + `#$00`).

ENDF

Mező	Leírás
Register	Automatikusan a FOR regiszteréhez illesztve
Label	Automatikusan a FOR címkéjéhez kötve
Count	Automatikusan a párosított FOR-ból másolva

Expert szintaxis:

```
.endf for0
```

Generált ASM:

```
    INX
    CPX #$12
    BNE for0
```

Méret: 5 byte (IN_ + CP_ #imm + BNE offset).

Példa — null-lezárt string kiírása:

```
    LDX #$00
for0:
    LDA msg,X       ; msg = PETSCII string fix címen
    BEQ done        ; null lezáró → kilépés
    JSR $FFD2       ; CHROUT
    INX
    CPX #$12        ; max 18 karakter
    BNE for0
done:
    RTS
```

> **LOOP vs FOR:** a LOOP lefelé számol (N→1) — jó késleltetésekhez, kitöltésekhez, pixel hurkokhoz. A FOR felfelé számol (0→N) — jó string/tömb eléréshez. Mindkettő használhat X-et vagy Y-t.

PUSH / PULL

Mint **változók mentése egy GOSUB előtt és visszaállítása utána** — de a 6502 hardveres stacket használja. Ha egy szubrutin használja A-t, X-et vagy Y-t, vedd körül PUSH-sal és PULL-lal, hogy a hívó kód regiszterei megmaradjanak.

PUSH

Egy vagy több regisztert push-ol a stackre. A sorrend mindig A → X → Y (legbelső először).

Mező	Leírás
Registers	Bármely kombináció: `A`, `X`, `Y`, `AX`, `AY`, `XY`, `AXY`

Expert szintaxis:

```
.push AXY
```

Generált ASM (példa: `AX`):

```
    PHA
    TXA
    PHA
```

Méret: 1 byte A-hoz (`PHA`), 2 byte X-hez vagy Y-hoz (transfer + push).

PULL

Visszaállítja a regisztereket a stackről **fordított sorrendben** (Y → X → A).

Mező	Leírás
Registers	Ugyanaz, mint a PUSH — meg kell egyeznie a megfelelő PUSH blokkal

Expert szintaxis:

```
.pull AXY
```

Generált ASM (példa: `AX`):

```
    PLA
    TAX
    PLA
```

> **Szabály:** a PUSH és PULL mindig **azonos regiszterkészletet** kell használjon. `PUSH AX` → `PULL AX` (belsőleg fordítva állít vissza: előbb X, majd A).

END / RTS alias

Mint az **RTS barátságosabb makró-névvel** — a `.end` egyetlen `RTS` byte-ot bocsát ki, és rövid szubrutin-lezáróként viselkedik expert módban.

Expert szintaxis:

```
.end
```

Generált ASM:

```
    RTS
```

Méret: 1 byte.

Ezt akkor használd, ha egy szubrutin-vége jelölőt szeretnél, ami kicsit inkább makrónak, mint nyers utasításnak olvasható.

MACRO / ENDM / INVOKE

Mint **egy nevesített GOSUB paraméterekkel** — definiálj egyszer egy újrahasználható kódrészletet (MACRO…ENDM), majd hívd bárhol INVOKE-kal. Adj át minden alkalommal más argumentumértékeket ahelyett, hogy blokkokat másolgatnál.

MACRO (definíció eleje)

Mező	Leírás
Name	A makró azonosítója (pl. `setColor`)
Params	Opcionális, vesszővel elválasztott paraméternevek (pl. `color` vagy `color, count`)

Jelöli a makró-definíció elejét. A MACRO és ENDM közti blokkok a makró teste. A paraméteres makró **nem generál kódot** ott, ahol a definíció van. A paraméter nélküli makró ott szubrutinként is tárolódik, így a `JSR név` meghívja: a program folyása átugorja (`JMP`), a végére pedig `RTS` kerül. Ugyanez érvényes a fix címre include-olt könyvtár ilyen makróira is (`JMP` nélkül, mert a könyvtár a program folyásán kívül van). Használd a `{paramName}`-t helyőrzőként az argumentumokhoz.

Generált ASM:

```
; .MACRO setColor (color)
    ... (test blokkok)
; .ENDM
```

Expert-mód szintaxis:

```
.macro setColor color
    LDA {color}
    STA $D020
.endm
```

ENDM (definíció vége)

Lezárja az aktuális makró-definíciót. Nincs mező.

INVOKE

Meghív egy definiált makrót ezen a pozíción, és behelyettesíti a megadott argumentumértékeket a test `{paramName}` helyőrzőibe.

Mező	Leírás
Macro name	Válaszd ki a definiált makrók legördülőjéből

Mező	Leírás
Arguments	Vesszővel elválasztott argumentumértékek, amelyek illeszkednek a makró paraméterlistájához (pl. `#$07`)

Generált ASM:

```
; .invoke setColor(#$07)
    LDA #$07
    STA $D020
```

Expert-mód szintaxis:

```
; egyetlen argumentum:
.invoke setColor(#$07)

; több argumentum:
.invoke drawPixel($10, $20)

; nincs argumentum:
.invoke clearScreen

; szöveg / string argumentumok (idézőjelben):
.invoke printText("Hello, World!")

; .call alias (a .invoke szinonimája):
.call setColor(#$07)
```

A makró teste inline kifejtődik, a `{paramName}` a tényleges argumentumokra cserélve. A szóközzel elválasztott forma (`.invoke setColor #$07`) is elfogadott.

Argumentum-típusok:

- **Numerikus**: `#$07`, `$10`, `255` — hex vagy decimális értékek
- **Szöveges stringek**: `"Hello, World!"` — idézőjeles stringek; az idézőjeleken belüli vesszők a szöveg részének számítanak, nem argumentum-elválasztónak
- **Vegyes**: `#$07, "hello", $20` — bármely kombináció

> **Tipp:** definiáld a makrókat a programod tetején (vagy alján), majd INVOKE-old őket, ahol szükséges. A makrók többször is meghívhatók különböző argumentumokkal.

> **2.1:** egy makró másik makrót is hívhat (a `both(a, b)` a `setc(a)`-t); a belső hívás a külső hívás argumentumait kapja, saját egyedi lokális címkékkel. Az önmagát hívó makró 16 szint után hibával áll meg, nem akad el.

REGION / ENDREGION

Tisztán vizuális csoportosítás — **nulla byte**, nulla hatás a lefordított kódra. Mint egy BASIC-program szakaszának nevesített blokká hajtása, hogy becsukhasd, és másra fókuszálhass.

Mező	Leírás
Region name	Szabad szöveges címke a szakaszhoz (pl. `init`, `game_loop`, `sprite_setup`)

Expert szintaxis:

```
.region init
    ; blokkok...
.endregion
```

Vezérlők a REGION blokk fejlécén (mindig láthatók):

- ▸ / ▾ **kapcsoló** — becsukja vagy kinyitja a teljes régiót. Becsukva a REGION és ENDREGION közti minden blokk el van rejtve.
- ↕ **Expand all** — kinyit minden egyénileg becsukott blokkot a régión belül, és szükség esetén magát a régiót is.
- ⊖ **Select in ASM** — kiemeli a teljes régió kód-tartományát az ASM nézetben (a `; ===[ name ]===` -tól a `; ===[/name]===` -ig), és odagörget. Automatikusan az ASM fülre vált, ha nem látható.
- ⧉ **Copy region** — a REGION blokkot, minden gyerekblokkot és a hozzá tartozó ENDREGION-t vágólapra másolja. Egy ✓ villanás megerősíti a másolást.
- ⎘ **Paste region** — a másolt régiót új régióként szúrja be közvetlenül az aktuális régió ENDREGION-je után, és odagörget. A gomb halvány, amíg nincs másolt régió.

Generált ASM:

```
; region init
    SEI
    LDA #$00
    STA $D020
; endregion init
```

Méret: 0 byte a REGION és az ENDREGION esetén is.

Példa munkafolyamat:

1. Adj hozzá egy `REGION` blokkot, a régió nevét állítsd `init`-re.
2. Add hozzá az inicializáló utasításaidat alá.
3. Adj hozzá egy `ENDREGION` blokkot a szakasz lezárásához.
4. Kattints a ▸-re a REGION-on, hogy egy sorrá csukd az egész szakaszt, amíg a program más részein dolgozol.

> **Megjegyzés:** a régiók **egymásba ágyazhatók**. Minden ENDREGION a legközelebbi nyitott REGION-t zárja. Nincs hatás a lefordított kimenetre.

DEFINE / IF / ELSE / ENDIF

Mint **egy kapcsoló, amit az assembler olvas** — a `DEFINE DEBUG` bekapcsol egy szimbólumot, majd minden `IF DEBUG` blokk bekerül, és az `ELSE` ága kimarad. Vedd ki a DEFINE blokkot, és az IF blokk eltűnik a kimenetből. Release buildekhez nem kell kódot törölni.

DEFINE

Mező	Leírás
Symbol	Egy vagy több, vesszővel elválasztott aktiválandó azonosító (pl. `DEBUG` vagy `DEBUG, PAL`)

Expert szintaxis:

```
.define DEBUG, PAL
```

Generált ASM:

```
; .DEFINE DEBUG
; .DEFINE DEBUG, PAL
```

Egy `DEFINE` blokk egyszerre több szimbólumot is aktiválhat (vesszővel elválasztva). Tedd a DEFINE blokkokat a programod tetejére. A blokk eltávolítása azonnal deaktiválja minden szimbólumát.

IF

Mező	Leírás
Condition	`.define`-olt szimbólum (aktív), `.const` név (aktív, ha nem nulla) vagy konstans-kifejezés (`LEVEL == 2`); ami nem értékelhető ki, hiba

Expert szintaxis:

```
.if DEBUG
```

Generált ASM:

```
; .IF DEBUG
```

Az `IF` és `ENDIF` (vagy `ELSE`) közti blokkok bekerülnek vagy kimaradnak attól függően, hogy a feltétel-szimbólumnak van-e illeszkedő `DEFINE`-ja a programban. A kimaradt blokkok `; [IF skipped] …` megjegyzésként jelennek meg, és **nulla byte-ot** generálnak.

ELSE

Nincs mező. Jelöli az alternatív ágat — akkor assembleálódik, amikor az `IF` feltétel *nem* aktív.

Expert szintaxis:

```
.else
```

Generált ASM:

```
; .ELSE
```

ENDIF

Nincs mező. Lezárja a feltételes blokkot.

Expert szintaxis:

```
.endif
```

Generált ASM:

```
; .ENDIF
```

Méret: 0 byte mind a négy blokknál. Csak a *közöttük* lévő tartalom számít.

Példa — debug keret-villantás, a release build kihagyja:

```
; .DEFINE DEBUG

; .IF DEBUG
    LDA #$02        ; piros keret
    STA $D020
; .ELSE
    LDA #$00        ; fekete (release)
    STA $D020
; .ENDIF
```

Példa — több szimbólum egy DEFINE blokkban:

```
; .DEFINE DEBUG, PAL

; .IF PAL
    LDA #$xx        ; PAL időzítési konstans
; .ELSE
    LDA #$xx        ; NTSC időzítési konstans
; .ENDIF
```

Az egymásba ágyazott `IF` blokkok támogatottak. Ha egy külső blokk kimarad, a belső blokkok is kimaradnak.

> **Megjegyzés:** ez fordítási idejű feltételkezelés. Futásidejű compare/branch cukorért lásd a **Futásidejű IF / ELSE / ENDIF**-et lentebb.

.ASSERT

(Új a 2.3.9-ben.) Egy **fordítási idejű józansági ellenőrzés**. A `.assert` egy kifejezést értékel ki assembleálás közben; ha hamis (`0`), a build egy beszédes hibával leáll, amely tartalmazza a tényleges értéket. Ha igaz (nem nulla), semmit nem bocsát ki.

Mező	Leírás
Expression	Bármely assembler-kifejezés: címkék, `CONST`-ok, `*` (program counter), aritmetika és összehasonlítások (`<` , `<=` , `>` , `>=` , `==` , `!=`)
Message	Opcionális szöveg, amely a hibaüzenethez fűződik

Expert szintaxis:

```
.assert spriteData < $C000
.assert * < $A000
.assert end - start <= 256, "a sprite tábla túllépett egy lapot"
```

Viselkedés:

- **Méret:** 0 byte.
- Egy hamis assertion megszakítja az assembleálást: `` `.assert end - start <= 256` hamis (érték: 0). a sprite tábla túllépett egy lapot ``
- Az a kifejezés is hibára fut, amelyet nem lehet kiértékelni (nem definiált címke, stb.), *„nem értékelhető ki fordítási időben"* üzenettel.
- Az összehasonlítások `1` / `0`-t adnak; tedd a `.assert`-et bárhova a program folyamában — azon a címen ellenőrződik, ahol áll, így a `.assert * < $D000` az aktuális kimeneti pozíciót teszteli.

Példa — sprite blokk védése lap-átlépés ellen:

```
* = $3000
sprite_data:
    .byte $00, $00, $00        ; … 63 byte …
sprite_data_end:
    .assert sprite_data_end - sprite_data == 63, "a sprite pontosan 63 byte kell legyen"
    .assert (sprite_data % 64) == 0, "a sprite 64 byte-ra igazított kell legyen"
```

CONST

Mint **egy nevesített változó, amely sosem változik** — `SCREEN = $0400`. Használd a nevet ahelyett, hogy mindenhol nyers címeket gépelnél, így a kód olvashatóbb és később könnyebben módosítható.

Mező	Leírás
Name	A konstans azonosítója (pl. `SCREEN`)
Value	Numerikus érték a választott számrendszerben (pl. `0400` HEX-ben = `$0400` cím), vagy egy PC-relatív kifejezés (lásd lentebb)
Format	HEX vagy DEC — vezérli, hogyan adod meg és jelenik meg az érték

Expert szintaxis:

```
.const SCREEN = $0400
.const FRAMES_1S = 60
```

Generált ASM:

```
; .CONST SCREEN = $0400
```

A konstans neve megjelenik a **címke-választó** legördülőben az utasításblokkokon — csak kattints rá a beszúráshoz.

PC-relatív kifejezések (`*+N` / `*-N`):

Az érték-mező elfogad `*+N`-t vagy `*-N`-t is, ahol a `*` magának a CONST blokknak a fordítási címe. Ezzel egy nevesített aliast hozhatsz létre egy közeli utasításon belüli byte-hoz — a klasszikus önmódosító kód minta:

```
CONST op = *+1      ; op → a következő LDA immediate operandusának címe
LDA #$00            ; a $00-t futásidőben módosítjuk
...
STA op              ; felülírja a #$00 byte-ot → az LDA legközelebb az új értéket olvassa
```

A CONST 0 byte-ot bocsát ki; a címke fordítási időben `aktuális_cím + 1`-re oldódik.

Aritmetikai kifejezések:

Az érték-mező elfogad általános aritmetikát, beleértve a korábban definiált CONST nevekre való hivatkozásokat, hex/bináris literálokat és beépített matematikai függvényeket:

```
.const SCREEN      = $0400
.const SCREEN_END  = SCREEN + 40*25   ; 1000 byte-tal később
.const COLOR_RAM   = $D800
.const MID_X       = 160
.const SIN_TABLE   = round(sin(PI/8) * 127)   ; előre kiszámolt szinuszérték
```

Beépített függvények: `sin()`, `cos()`, `round()`, `max(a,b)`, `min(a,b)`, `abs()`, `PI` konstans

Operátorok: `+ - * /` Literálok: `$FF` (hex), `%10110000` (bináris) Alsó/felső byte: `lo(expr)`, `hi(expr)`

Méret: 0 byte.

VAR

Mint a **CONST, de automatikusan foglalva** — a `VAR` zero-page tárhelyet foglal egy címkéhez anélkül, hogy be kellene gépelned a címet. Használd számlálókhoz, pointerekhez és rövid életű, ZP-be való állapothoz.

Mező	Leírás
Name	Változónév / címke
Size (opcionális)	Foglalandó byte-ok száma. Hagyd el egy byte-hoz.

Expert szintaxis:

```
.var counter
.var timer, 2
.var lives
```

Gyakorlati példa:

```
.region Vars
.var counter
.var timer, 2
.endregion

LDA #$00
STA counter
```

Generált ASM:

```
; .var counter
```

Méret: alapból 1 byte, vagy `N` byte, ha megadod a méretet.

Az allokátor egy konfigurálható zero-page kurzort jár be (`$02` -tól `$FE` -ig), és a következő szabad slotot rendeli hozzá. Ha a kért terület átfed egy már használt címkével, a fordító figyelmeztetést ad.

Futásidejű IF / ELSE / ENDIF

Mint **egy valódi elágazás-sablon** — ez a változat futásidőben működik, nem fordítási időben. Összehasonlítja `A` , `X` vagy `Y` értékét egy immediate értékkel, és a helyes `CMP` / `CPX` / `CPY` + elágazás szekvenciát bocsátja ki neked.

Mező	Leírás
Register	`A` , `X` vagy `Y`
Operator	`==` , `!=` , `<` , `<=` , `>` , `>=`
Value	Immediate érték HEX vagy DEC formátumban

Expert szintaxis:

```
.if A == #$10
    LDA #$07
.else
    LDA #$0F
.endif
```

Méret: a választott elágazásoktól és az összehasonlítási formától függ.

Az összehasonlítás alapból előjel nélküli. A `<=` és `>` esetén a makró a kiválasztott regiszterhez a legrövidebb ekvivalens elágazás-láncra fejt ki.

Minta: Futásidejű IF / ELSE demo (`samples/runtime-if-demo.asm`).

WHILE / ENDW

Mint **egy futásidejű hurok teszttel a tetején** — a test addig fut, amíg a feltétel igaz marad.

Mező	Leírás
Register	`A` , `X` vagy `Y`
Operator	`==` , `!=` , `<` , `<=` , `>` , `>=`
Value	Immediate érték HEX vagy DEC formátumban

Expert szintaxis:

```
.while A != #$00
    JSR getchar
.endw
```

Méret: a hurok testétől és az összehasonlítási formától függ.

Használd a `WHILE`-t, ha a hurok az első iteráció befejezése előtt is véget érhet. Ez a szám-alapú `LOOP / NEXT` segéd futásidejű megfelelője.

REPEAT / UNTIL

Mint **egy futásidejű hurok teszttel az alján** — a test mindig lefut legalább egyszer, majd a feltétel dönt a leállásról.

Mező	Leírás
Register	`A`, `X` vagy `Y`
Operator	`==`, `!=`, `<`, `<=`, `>`, `>=`
Value	Immediate érték HEX vagy DEC formátumban

Expert szintaxis:

```
.repeat
    JSR getchar
.until A == #$00
```

Méret: a hurok testétől és az összehasonlítási formától függ.

Használd a `REPEAT / UNTIL`-t, ha azt akarod, hogy a test legalább egyszer lefusson a kilépési ellenőrzés előtt.

MEMCPY / MEMSET

Mint **kis memória-rutinok, amelyekhez folyton nyúlsz** — a `MEMCPY` egy összefüggő blokkot másol, a `MEMSET` egy tartományt tölt fel egy byte-tal.

Makró	Mezők
`MEMCPY`	`src`, `dst`, `size`
`MEMSET`	`addr`, `value`, `size`

Expert szintaxis:

```
.memcpy src=$C000, dst=$D000, size=$0100
.memset addr=$0400, value=#$20, size=$03E8
```

Generált ASM: inline másoló/kitöltő hurkok, a kért mérethez választva.

A 256 byte-ig terjedő méretek rövid 8 bites hurkot használnak. A nagyobb méretek automatikusan 16 bites számlálóra váltanak.

> **2.1:** a méretet már a kód elrendezésekor ismerni kell — szám vagy konstans. A később definiált címke (amelynek értékét az elrendezés még nem ismeri) hiba, nem a helyére nem férő kód.

PRINT / PRINT_CHAR / PRINT_HEX / CLEAR_SCREEN / WAIT_KEY / DELAY / SET_BORDER / SET_BG

PRINT

Mint **PETSCII kimenet a sablonkód nélkül** — egy stringet ír ki `CHROUT`-on át ugyanazzal a kis/nagybetű kezeléssel, mint a PETSCII blokk. A kisbetűs jelölőnégyzet közös a PETSCII kódolóval, így a szöveg-útvonal konzisztens marad.

Expert szintaxis:

```
.print "HELLO"
.print "hello", lower
```

PRINT_CHAR

Egyetlen PETSCII byte-ot ír ki numerikus kód alapján, `CHROUT`-on átküldve. Az érték lehet nevesített const vagy címke is, amely assembleáláskor byte-ra oldódik — Blokk és Expert módban egyaránt.

Expert szintaxis:

```
.print_char 65
.print_char $41
.print_char color
```

PRINT_HEX

Egy 8 bites értéket ír ki hexadecimális szövegként a szokásos KERNAL kimeneti útvonalon.

Expert szintaxis:

```
.print_hex A
```

CLEAR_SCREEN

Rövidítés a szabványos C64 képernyőtörlés vezérlőkódhoz.

Expert szintaxis:

```
.clear_screen
```

WAIT_KEY

Vár, amíg egy billentyűt le nem nyomnak, így nem kell minden alkalommal kézzel megírni a `GETIN` hurkot.

Expert szintaxis:

```
.wait_key
```

DELAY

A kért számú frame-et várja egy közös segédrutinon keresztül. Ezt használd rövid szünetekhez és időzítési résekhez, amikor egy teljes egyéni hurok túlzás lenne. A frame-szám lehet nyers szám vagy nevesített const, és a `.wait` csak a

`.delay` aliasa.

Expert szintaxis:

```
.delay 29
.wait 29
.delay frames=FRAMES_1S
```

Blokk módban a delay mező kompakt const-választót használ, ha van szimbolikus érték, így nem kell minden alkalommal kézzel begépelni a nevet.

SET_BORDER / SET_BG

Kényelmi burkolók a VIC-II szín-regiszterekhez. A szín-érték lehet nyers szám vagy nevesített const, amely 0–15-re oldódik. Blokk és Expert mód is elfogad szimbolikus const-neveket itt.

Expert szintaxis:

```
.set_border 6
.set_bg 0
.set_border color
.set_bg color
```

Méret: minden segéd egy pici regiszter-írási szekvenciára vagy rövid KERNAL hívásra fejt ki.

Blokk módban ezek a mezők is a const-választót használják, így a szimbolikus érték látható marad ahelyett, hogy nyers számra cserélődne.

IRQ_SETUP

Egy lépésben állít be egy raszter IRQ kezelőt. A makró beírja az IRQ vektort, engedélyezi a raszter IRQ-kat, beállítja a sort, letiltja a gyakori CIA IRQ forrásokat, és `CLI`-vel tér vissza a normál végrehajtáshoz.

Mező	Leírás
Handler	IRQ rutin címkéje (pl. `my_irq`)
Raster	Raszter sor hex vagy decimális formában (pl. `$FA`)

Expert szintaxis:

```
.irq_setup handler=my_irq, raster=$FA
```

Méret: egy kis beállító szekvencia; a pontos hossz a választott raszter sortól függ.

Ezt akkor használd, ha a gyakori „SEI / kezelő beírása / IRQ engedélyezés / CLI" sablonkódot szeretnéd anélkül, hogy szétszórnád a programban.

RAND

Mint **egy pici beépített PRNG** — 8 bites pszeudo-véletlen értéket ad egy kompakt zero-page seedből.

Mező	Leírás
Seed	Opcionális zero-page seed byte vagy címke (alap `$FB`)

Expert szintaxis:

```
.rand
```

Méret: néhány byte, a választott implementációs úttól függően.

A generátor játékmenetre, effekt-variációra és gyors tesztadatra való. Szándékosan kicsi, nem kriptográfiailag kifinomult.

SPRITE_INIT

Egy blokkban állít be egy VIC-II sprite-ot — ahelyett, hogy ~6 POKE utasítást írnál BASIC-ben, csak töltsd ki a mezőket. Beállítja a sprite adat-pointerét, bekapcsolja, opcionálisan engedélyezi a multicolor módot, és beállítja a színét.

Mező	Leírás
Sprite #	Sprite szám 0–7
Colour	Színindex 0–15 (C64 paletta)
Data page	Sprite adat cím / 64 (pl. `$21`, ha az adat a `$0840`-en van)
Multicolor	A sprite multicolor bitjét kapcsolja (`$D01C`)

Expert szintaxis:

```
.sprite_init 0, 7, $21
.sprite_init 0, 7, $21, multicolor
.sprite_init 0, 7, $21, mono
```

Generált ASM:

```
    LDA #$21
    STA $07F8       ; sprite pointer regiszter ($07F8 + N)
    LDA $D015
    ORA #$01        ; sprite 0 engedély bit beállítása
    STA $D015
    LDA $D01C
    AND #$FE        ; sprite 0 multicolor bit törlése
    STA $D01C
    LDA #$07
    STA $D027       ; sprite 0 szín-regiszter
```

Méret: 26 byte.

> **Sprite data page:** `data_address ÷ 64`. Az alapértelmezett BASIC SYS stubbal a `JMP main` utáni `ALIGN 64` a sprite adatot a `$0840`-re helyezi → page = `$21`.

SPRITE_POS

Mint a `POKE 53248, x : POKE 53249, y` BASIC-ben — beállítja egy sprite kezdőpozícióját. A koordináták assembleáláskor be vannak égetve, és a mezők elfogadnak const-okat Blokk módban; animációhoz használj `INC` / `DEC` -et közvetlenül a sprite regiszteren.

Mező	Leírás
Sprite #	Sprite szám 0–7
X	Vízszintes pozíció 0–319
Y	Függőleges pozíció 0–255

Expert szintaxis:

```
.sprite_pos 0, 152, 100
```

Generált ASM (példa: sprite 0, X=152, Y=100):

```
    LDA #$98        ; X alsó byte
    STA $D000       ; sprite 0 X regiszter
    LDA $D010
    AND #$FE        ; sprite 0 X MSB törlése (X ≤ 255)
    STA $D010
    LDA #$64        ; Y = 100
    STA $D001       ; sprite 0 Y regiszter
```

X > 255 esetén a makró a `$D010` megfelelő bitjét beállítja ahelyett, hogy törölné.

Méret: 18 byte.

> **Megjegyzés:** a `SPRITE_POS` beégeti az X/Y-t a kódba (`LDA #$xx`). Sprite futásidejű animálásához használj `INC $D000` / `DEC $D000` -t — lásd a `sprite-macro-demo` mintát.

WAIT_RASTER

Megvárja, míg a VIC-II elektronsugár elér egy adott scanline-t — mint a TV frame-re szinkronizálás. Tedd ezt a game hurkod tetejére a sprite tearing megelőzéséhez. Nincs JSR, nem kell címke.

Mező	Leírás
Raster line	Cél raszter sor hexben (pl. `FF` = 255. sor)

Expert szintaxis:

```
.wait_raster $FF
```

Generált ASM:

```
wait:
    LDA $D012       ; aktuális raszter sor
```

```
    CMP #$FF        ; cél sor
    BNE wait        ; vissza a hurokba (-7 byte)
```

Méret: 7 byte (a `BNE` offset `$F9` = −7 mindig az `LDA`-ra mutat vissza).

> **Tipp:** tedd a `WAIT_RASTER`-t a game hurkod tetejére a kijelzővel való szinkronizáláshoz és a sprite tearing megelőzéséhez.

JOYSTICK

Mint a **`PEEK($DC00)`** olvasása, majd a sprite pozíció POKE-olása — de egy blokkban. Beolvassa az egyik CIA joystick portot, és ennek megfelelően állítja egy sprite X/Y regisztereit. Teljesen inline, nem kell JSR.

Mező	Leírás
Port	`1` = port 1 (`$DC01`) vagy `2` = port 2 (`$DC00`)
Sprite #	Sprite szám 0–7 (melyik X/Y regiszterpár frissül)

Expert szintaxis:

```
.joystick 2, 0
```

Generált ASM (port 2, sprite 0):

```
    LDA $DC00       ; CIA port 2 olvasása
    LSR             ; 0. bit → carry (Fel)
    BCS skip_up     ; carry beállítva = NINCS nyomva
    DEC $D001       ; Y−1 (fel mozgatás)
skip_up:
    LSR             ; 1. bit → carry (Le)
    BCS skip_down
    INC $D001       ; Y+1 (le mozgatás)
skip_down:
    LSR             ; 2. bit → carry (Balra)
    BCS skip_left
    DEC $D000       ; X−1 (balra mozgatás)
skip_left:
    LSR             ; 3. bit → carry (Jobbra)
    BCS skip_right
    INC $D000       ; X+1 (jobbra mozgatás)
skip_right:
```

Joystick bittérkép (active-LOW — bit = 0 jelenti a nyomást):

Bit	Irány	CIA regiszter
0	Fel	$DC00 (port 2) / $DC01 (port 1)
1	Le	
2	Balra	
3	Jobbra	
4	Tűz	(ez a makró nem kezeli)

Méret: 27 byte. A `BCS` offset mindig `+3` (átugorja a következő 3 byte-os `DEC` / `INC abs` utasítást).

Változós mód — `.joystick port, vx, vy[, fire]`: sprite mozgatása helyett az irányt változókba írja: `vx` / `vy` = `-1` / `0` / `+1` (`$FF` / `$00` / `$01`), az opcionális `fire` = `1` , amíg a gomb le van nyomva, egyébként `0` . Görgetett kamerához, saját fizikájú játékoshoz vagy menükhöz. A `3` -as port egyszerre olvassa mindkét joystick portot (`LDA $DC00` / `AND $DC01`), így bármelyik port működik.

```
.joystick 2, vx, vy          ; 35 byte
.joystick 3, vx, vy, fire    ; mindkét port + tűz: 45 byte
```

> **Tipikus használat:** tedd egy `gameloop` címke belsejébe, elöl `WAIT_RASTER` -rel:
>
> ```
> gameloop:
>     WAIT_RASTER ($FF)
>     JOYSTICK (port=2, sprite=0)
>     JMP gameloop
> ```

> **2.1:** a portot már a kód elrendezésekor ismerni kell (szám vagy konstans), mert a 3-as port 3 bájttal hosszabb kódot ad; a később definiált címke hiba.
>
> **MOUSE**

Beolvas egy Commodore 1351 arányos egeret, és mozgat egy sprite-ot. Teljesen **inline** — nem kell JSR vagy címke. A makró kiválasztja a CIA portot, megvárja, míg a SID paddle bemenetek beállnak, majd dekódolja a delta mozgást a szabványos 1351 driver mintával, és alkalmazza a sprite regiszterekre.

Mező	Leírás
Port	`1` = CIA `$DC00` `7:6` bitek = `%01` ; `2` = `%10`
Sprite #	Sprite szám 0–7
ZP byte X	Zero-page cím (hex) az előző POTX minta tárolására (pl. `FD`)
ZP byte Y	Zero-page cím (hex) az előző POTY minta tárolására (pl. `FE`)

Generált ASM forma (port 1, sprite 0, ZP `$FD` / `$FE`):

```
    ; CIA port kiválasztás + beállás
    LDA $DC00
    AND #$3F
    ORA #$40
    STA $DC00
    LDX #$67
wait:
    DEX
    BNE wait

    ; X tengely — szabványos 1351-stílusú 7 bites delta dekódolás
    LDA $D419
    TAY
    SEC
```

```
    SBC $FD
    AND #$7F
    LDX #$00
    CMP #$40
    BCS xneg
    LSR A
    BEQ xdone
    STY $FD
    CLC
    ADC $D000
    STA $D000
    TXA
    ADC #$00
    AND #$01
    BEQ xdone
    LDA $D010
    EOR #$01
    STA $D010
    JMP xdone
xneg:
    ORA #$C0
    CMP #$FF
    BEQ xdone
    SEC
    ROR
    DEX
    STY $FD
    CLC
    ADC $D000
    STA $D000
    TXA
    ADC #$00
    AND #$01
    BEQ xdone
    LDA $D010
    EOR #$01
    STA $D010
xdone:

    ; Y tengely – ugyanaz a dekódolás, majd invertálás alkalmazás előtt
    LDA $D41A
    TAX
    SEC
    SBC $FE
    AND #$7F
    CMP #$40
    BCS yneg
    LSR A
    BEQ ydone
    STX $FE
    EOR #$FF
    SEC
    ADC $D001
    STA $D001
    JMP ydone
yneg:
    ORA #$C0
    CMP #$FF
    BEQ ydone
    SEC
```

```
    ROR
    STX $FE
    EOR #$FF
    SEC
    ADC $D001
    STA $D001
ydone:
```

Méret: 142 byte.

Expert-mód szintaxis:

```
.mouse port, spriteNum, zpX, zpY
; példa:
.mouse 2, 0, FD, FE
```

> **Fontos:** az első hívás előtt inicializáld a zero-page byte-okat az aktuális POTX/POTY értékekkel, hogy elkerüld az ugrást az első frame-en:
>
> ```
> ;     port 1: LDA $DC00 : AND #$3F : ORA #$40 : STA $DC00
> ;     port 2: LDA $DC00 : AND #$3F : ORA #$80 : STA $DC00
>    LDA $D419 : LSR A : AND #$3F : STA $FD
>    LDA $D41A : LSR A : AND #$3F : STA $FE
> ```

> **Tipp:** olvasd az egeret frame-enként egyszer — tedd a `WAIT_RASTER` -t a game hurokban a `MOUSE` elé.

SPRITE_COL

Mint a **`PEEK($D01E)`** BASIC-ben — ellenőrzi a VIC-II hardveres ütközés-regisztereket, és megmondja, hogy egy sprite eltalált-e egy másik sprite-ot vagy a hátteret. Teljesen inline, nem kell JSR.

Mező	Leírás
Sprite #	Sprite szám 0–7 (melyik sprite bitjét ellenőrizze)
Collision type	`Sprite-Sprite ($D01E)` — ütközés másik sprite-tal; `Sprite-Background ($D01F)` — ütközés a háttér-grafikával

Expert szintaxis:

```
.sprite_col 0, sprite
.sprite_col 0, background
```

Generált ASM (sprite 0, sprite–sprite):

```
    LDA $D01E       ; sprite-sprite ütközés-regiszter olvasása (törli is!)
   AND #$01        ; 0. bit izolálása (sprite 0)
                   ; A ≠ 0 → ütközés történt
```

Méret: 5 byte.

> **Fontos:** a `$D01E` / `$D01F` olvasása **törli a regisztert**. Olvasd frame-enként egyszer, és azonnal reagálj az eredményre `BEQ` / `BNE` -vel.

Tipikus használat:

```
gameloop:
    WAIT_RASTER ($FF)
    JOYSTICK (port=2, sprite=0)
    SPRITE_COL (sprite=0, type=sprite-sprite)
    BEQ no_hit          ; A = 0 → nincs ütközés
    LDA #$02
    STA $D020           ; piros keret = találat!
    JMP gameloop
no_hit:
    LDA #$0E
    STA $D020           ; világoskék keret = tiszta
    JMP gameloop
```

> **Lásd még:** `collision-demo` minta — zöld labda (sprite #0) vs. piros kereszt (sprite #1).

LOADFILE

Mint a **`LOAD "file",8`** BASIC-ben — betölt egy fájlt egy D64 lemezről futásidőben a KERNAL LOAD rutinjával. Ezzel tölthetsz adatot, zenét vagy extra kódot lemezről, miközben a programod fut.

Mező	Leírás
Filename	Fájlnév a lemezen (max 16 karakter, automatikus nagybetűsítés; a `,` , `"` , `/` , `\` , `:` , `*` , `?` , `<` , `>` , `|` karakterek kiszűrődnek)
Device	Eszközszám 8–30 (alap `8`)
Override address (opcionális)	Hex betöltési cím (pl. `C000`). Ha beállított, a fájl erre a címre töltődik (`sec=0` , a PRG fejlécet figyelmen kívül hagyva). Hagyd üresen a fájl saját 2 byte-os PRG fejlécének használatához (`sec=1`).
Error label (opcionális)	Ha beállított, egy `BCS` utasítás generálódik a JSR LOAD után. Ha a KERNAL carry beállítással (hiba) tér vissza, a végrehajtás erre a címkére ugrik.

Expert szintaxis:

```
.loadfile "DEMO-COLORS", 8
.loadfile "DEMO-COLORS", 8, $C000
.loadfile "DEMO-COLORS", 8, $C000, error_label
```

Generált kód szerkezete:

```
    JMP skip_filename      ; átugrik az inline fájlnéven
    .byte "DEMO-COLORS"    ; fájlnév byte-ok (PETSCII, 11 karakter)
skip_filename:
    LDA #11                ; fájlnév hossza
    LDX #<fname            ; pointer lo
    LDY #>fname            ; pointer hi
    JSR $FFBD              ; SETNAM
```

```
    LDA #1                ; logikai fájl #1
    LDX #8                ; device 8
    LDY #0                ; sec=0 (override cím) vagy #1 (fájl saját címe)
    JSR $FFBA             ; SETLFS
    LDA #0                ; LOAD parancs (nem VERIFY)
    JSR $FFD5             ; LOAD
    BCS fail              ; (csak ha van error label)
```

Méret: `3 + fájlnév_hossz + 9 (SETNAM) + 9 (SETLFS) + (4 ha override) + 5 (LOAD) + (2 ha error label)` byte. Minimum 27 byte.

> **Fontos:** a fájlnév inline tárolódik a gépi kódban közvetlenül egy `JMP skip_filename` után. A lemezen lévő fájlnévnek nagybetűs PETSCII-nek kell lennie — ami megegyezik a sima ASCII nagybetűkkel (`A` – `Z`). A makró ezt automatikusan kikényszeríti.

> **Éles programokhoz mindig használj error label-t** — ha a fájl nincs meg, a KERNAL beállítja a carry flag-et, és a végrehajtás átcsúszik arra, ami utána van.

> **Lásd még:** `loadfile-demo` minta — `DEMO-COLORS.PRG` betöltése D64-ről BCS hibaágyal és vizuális hibaképernyővel.

EXODECRUNCH

Programon belüli **Exomizer kitömörítés**. Ezt a makrót közvetlenül egy olyan `LOADFILE` után használd, amely egy Exomizer `mem` -módú tömörített streamet töltött be — az EXODECRUNCH visszafelé csomagolja ki a streambe ágyazott címre.

Mező	Leírás
Depacker address	Ahol a depacker kód él a memóriában (alap `B000`). 16 bites hex cím kell legyen.

Expert szintaxis:

```
.exodecrunch
.exodecrunch depacker=$B000
```

Generált kód (19 byte):

```
    LDA $AE           ; KERNAL load-end lo (az előző LOAD állította be)
    STA $04           ; depacker src-end pointer lo
    LDA $AF           ; KERNAL load-end hi
    STA $05           ; depacker src-end pointer hi
    LDA #$36          ; BASIC ROM ki ($A000-$BFFF RAM lesz)
    STA $01
    JSR depacker      ; a depacker hátulról előre olvas a $04/$05-ön át
    LDA #$37          ; alapértelmezett mapping visszaállítása (BASIC + KERNAL + I/O)
    STA $01
```

Hogyan működik:

1. A KERNAL `LOAD` ($FFD5) a ZP `$AE/$AF` -t az utoljára betöltött byte utáni pozícióra állítja. Az EXODECRUNCH ezt a ZP `$04/$05` -re másolja, ami a hivatalos Exomizer visszafelé-forrás-vég konvenció.

2. A depacker jellemzően a `$B000`-en van (a BASIC ROM-ra mappelt tartományon belül). A makró `$01 = $36`-ra kapcsol, hogy a CPU ott RAM-ot lásson a JSR alatt, majd utána visszaállítja `$01 = $37`-re.
3. A kitömörítési célcím **magába a tömörített streambe van kódolva**, amikor `exomizer mem -l <load> file, <target>`-rel tömörítesz — a depacker a stream első byte-jaiból olvassa ki.

Depacker bináris: az előre elkészített visszafelé-depacker a `samples/exo-decrunch.bin` (477 byte, ORG $B000). Ez a hivatalos `exodecrunch.asm` Kick Assembler-es burkolása, minden olvasáshoz hozzáadott `INC $D020`-szal, hogy a kitömörítés alatt látható keret-villantás legyen. Tedd a programodba egy `INCBIN` blokkal a depacker címén.

Nyers bemenet: az extra nyers fájlként, a Dst címre töltve megy az Exomizerhez (`file@<Dst>`), így a depacker minden bájtot, az elsőtől kezdve, pontosan a beírt címre ír.

> **Lásd még:** `exo-multicolor-demo` minta — teljes végponttól végpontig példa: LOADFILE-lal egy tömörített multicolor bitmap a $C000-re, EXODECRUNCH kicsomagolja a $2000-re, majd screen → $0400 és color → $D800 másolás, és a VIC-II multicolor bitmap módra váltás.

> **Integrációs teszt:** `cargo test --test exomizer_integration` (a `src-tauri/`-ban) ellenőrzi a teljes tömörítés + kitömörítés round-tripet egy 6502 emulátoron a valódi depacker binárissal. Elfogadási kritérium: 10000 byte byte-azonos a forrás `multi-color.bin`-nel.

REU_CHECK

Detektálja, hogy be van-e dugva egy Commodore RAM Expansion Unit (REU) — mint a `PEEK($D010)` ellenőrzése, hogy jelen van-e a hardver. Két minta beírásával és visszaolvasásával tesztel a REU `$DF04` regiszterén.

Mező	Leírás
Nincs	Nincs operandus

Generált kód (34 byte):

```
LDA #$55
STA $DF04
LDA $DF04
CMP #$55
BNE fail
LDA #$AA
STA $DF04
LDA $DF04
CMP #$AA
BNE fail
LDA #$00
CMP #$FF   ; Z=0 => REU jelen van
BNE done
fail:
LDA #$FF
CMP #$FF   ; Z=1 => nincs REU
done:
```

> A makró normalizálja az eredményt, hogy a következő elágazás egyszerű maradjon: `BNE` = REU jelen van, `BEQ` = REU hiányzik.

Expert szintaxis:

```
.reu_check
```

Eredmény a flag-ekben:

- **Z = 0** (eredmény ≠ 0) → REU jelen van → használj `BNE`-t
- **Z = 1** (eredmény = 0) → nincs REU → használj `BEQ`-t

Nincs konfigurálható mező — a makró minden alkalommal ugyanazt a kódot generálja.

Tipikus használat:

```
REU_CHECK
BEQ no_reu        ; kihagyás, ha nincs REU
; ... REU kód itt ...
no_reu:
```

REU_STASH / REU_FETCH / REU_SWAP

DMA blokk-átvitel a C64 RAM és a REU bővítő memória közt — mint egy nagyon gyors POKE hurok, de a CPU nem dolgozik (a REU chip másolja az adatot, míg a CPU meg van állítva). Egy `$1000` byte-os átvitel gyakorlatilag azonnali.

Makró	Irány	`$DF01` parancs
`REU_STASH`	C64 RAM → REU	`$90`
`REU_FETCH`	REU → C64 RAM	`$91`
`REU_SWAP`	C64 RAM ↔ REU	`$92`

Mezők:

Mező	Leírás	Példa
C64 address	Forrás/cél a C64 RAM-ban (hex)	`C000`
REU address	Forrás/cél a REU-ban (hex, 16 bites)	`0000`
REU bank	REU memória-bank (0–7)	`0`
Length	Átviendő byte-ok száma (hex, 16 bites)	`1000`

Expert szintaxis:

```
.reu_stash $C000, $0000, 0, $1000
.reu_fetch $C000, $0000, 0, $1000
.reu_swap $C000, $0000, 0, $1000
```

Generált kód (40 byte):

```
LDA #c64lo    STA $DF02    ; C64 cím LO
LDA #c64hi    STA $DF03    ; C64 cím HI
LDA #reuLo    STA $DF04    ; REU cím LO
```

```
LDA #reuHi    STA $DF05    ; REU cím HI
LDA #bank     STA $DF06    ; REU bank
LDA #lenLo    STA $DF07    ; hossz LO
LDA #lenHi    STA $DF08    ; hossz HI
LDA #$00      STA $DF09    ; address control (fix)
LDA #$00      STA $DF0A    ; interrupt mask (fix)
LDA #cmd      STA $DF01    ; DMA végrehajtás ($90/$91/$92 = stash/fetch/swap, azonnali)
```

> **Megjegyzés:** a parancsok `$90/$91/$92`-t használnak (4. bit be = azonnali DMA mód). A `$DF01`-be írás elindítja az átvitelt; a CPU folytatódik, amikor befejeződik.

TURBO_SET

Beállítja az **Ultimate-64 (U64) CPU sebességet** a `$D031` regiszteren keresztül. Valódi C64-en vagy más emulátoron nincs hatása.

Mezők:

Mező	Leírás	Tartomány
Speed	CPU sebesség-index	0 = 1 MHz … 7 ≈ 10 MHz … 15 ≈ 48 MHz
Badline	Badline emuláció	Engedélyezve (C64 kompatibilis) / Letiltva (turbo)

A sebesség-byte így számolódik: `(speedIndex & 0x0F) | (badline_letiltva ? 0x80 : 0x00)`.

Generált kód (5 byte):

```
A9 xx   LDA #speed_byte
8D 31 D0   STA $D031
```

Expert-mód szintaxis:

```
.turbo_set 7,0    ; speed=7 (~10 MHz), badline engedélyezve
.turbo_set 15,1   ; speed=15 (~48 MHz), badline letiltva
```

> **Megjegyzés:** ez a makró csak U64 hardvert érint. Valódi C64-en vagy más emulátoron a `$D031`-be ír, ami hatással lehet a CIA-ra, vagy figyelmen kívül maradhat.

U64_PALETTE / U64_PALETTE_COLOR / U64_PALETTE_RESET

Az **Ultimate 64 / C64 Ultimate videó-palettáját** állítják az Ultimate Command Interface-en át (`$DF1C` – `$DF1F`). 3.15-ös vagy újabb firmware kell, és az interfésznek bekapcsolva kell lennie: menü (F2) → **C64 and Cartridge Settings** → **Command Interface** = Enabled. A rutin előbb a `$DF1D`-t olvassa; ha az nem `$C9` (VICE, beépített C64, régebbi firmware, kikapcsolt interfész), átugorja magát, így a program bárhol fut.

Makró	Expert szintaxis	Méret	Parancs
U64_PALETTE	`.u64_palette "fájl.vpl"`	109 byte	`$52` + 16 × R, G, B: minden szín

Makró	Expert szintaxis	Méret	Parancs
U64_PALETTE_COLOR	`.u64_palette_color index, $RRGGBB`	65 byte	`$53` index, R, G, B: egy szín
U64_PALETTE_RESET	`.u64_palette_reset`	61 byte	`$54`: a menüben beállított paletta

Palettafájl: VICE `.vpl` (soronként `RR GG BB dither`), GIMP `.gpl`, bármilyen szöveg 16 `#RRGGBB` / `$RRGGBB` / `0xRRGGBB` értékkel, vagy Photoshop `.act`; az első 16 szín számít. A fájlt fordításkor olvassa, mint egy INCBIN-t (a projekthez képest, Blokk módban a Tallózás gombbal); a színek a generált kódban vannak, máshová nem kerül semmi a memóriában.

A `.u64_palette_color` **indexe és színe** kifejezés lehet (`BLUE`, `$3050C0`, `COL + 1`); az index 0–15, a szín `$000000` – `$FFFFFF` lehet.

A kód megvárja, amíg az interfész szabad (a korábbi parancsból maradt adatot elfogadja), a parancsot a `$DF1D` -be írja, elindítja (`$01` → `$DF1C`), vár, amíg dolgozik, majd elfogadja a választ (`$02` → `$DF1C`). Az A és X elvész. A paletta azonnal vált, és a következő resetig vagy paletta-parancsig marad.

Példa (az **Ultimate 64 paletta demó** minta ezt csinálja billentyűkre):

```
.u64_palette "u64-palette-sunset.vpl"   ; mind a 16 szín
.u64_palette_color 6, $5000A0           ; kék -> lila
.u64_palette_reset                      ; vissza a menü palettájára
```

> **Megjegyzés:** valódi gépen (Futtatás C64 Ultimate-en) vagy az **Ultimate emulátor** futtatási móddal próbáld ki; a VICE-alapú módokban nincs Command Interface, ott nem változik semmi.

SUPERCPU_DETECT

Ellenőrzi, hogy telepítve van-e egy **CMD SuperCPU** gyorsító — mint a `PEEK($D0B8)`, hogy `$FF` -től eltérő értéket ad-e vissza.

Generált kód (5 byte):

```
AD B8 D0   LDA $D0B8
C9 FF      CMP #$FF
```

Eredmény a flag-ekben:

- **Z = 0** → SuperCPU jelen van → használj `BNE` -t
- **Z = 1** → SuperCPU nincs → használj `BEQ` -t

Nincs konfigurálható mező.

Expert szintaxis:

```
.supercpu_detect
```

Tipikus használat:

```
SUPERCPU_DETECT
BEQ no_scpu       ; kihagyás, ha nincs SuperCPU
; ... SuperCPU turbo kód itt ...
no_scpu:
```

TURBO_ENABLE

Be- vagy kikapcsolja a **CMD SuperCPU turbo módot**. Előbb hívj `SUPERCPU_DETECT`-et, és hagyd ki ezt, ha a SuperCPU nincs jelen.

Mód	Regiszter	Hatás
Enable	`$D07A`	Turbo bekapcsolása (SuperCPU-val akár 20 MHz)
Disable	`$D07B`	Vissza az 1 MHz-es kompatibilitási módba

Generált kód (5 byte):

```
A9 00         LDA #$00
8D 7A D0      STA $D07A    ; (vagy $D07B a letiltáshoz)
```

Expert-mód szintaxis:

```
.turbo_enable on
.turbo_enable off
```

> **Megjegyzés:** előbb hívj `SUPERCPU_DETECT`-et, és ágazz el a makró körül, ha a SuperCPU nincs jelen.

MAP_COPY

Egy tilemap-et másol egy forráscímről a screen RAM-ba (és opcionálisan a color RAM-ba) `LDA abs,X` / `STA abs,X` hurkok sorozatával. Hurok-iterációnként egy 256 byte-os lap másolódik; a végén egy részleges lap `CPX #rem / BNE`-vel áll le. Nem kell JSR — minden kód inline generálódik.

Mező	Leírás
Source addr (screen)	Hex cím, ahol a map adat betöltés után él (pl. `C000`)
Screen RAM dest	Hová másolja a screen code-okat (pl. `0400`)
Size (bytes)	Összes másolandó byte — jellemzően `$03E8` = 1000 (40×25 karakter)
Combined .bin	Bepipálva screen code-okat vár, amelyeket közvetlenül a color adat követ a `source + size` címen; a color adatot egy második menetben a **Color RAM dest**-re másolja
Color RAM dest	A color adat célja — alap `D800` (C64 color RAM)

Generált ASM (1000 byte-os map, csak screen):

```
    LDX #$00
    LDA $C000,X   ; 0. lap
    STA $0400,X
    INX
    BNE *-9       ; hurok, míg X vissza nem fordul 0-ra (256 iteráció)
    LDA $C100,X   ; 1. lap
    STA $0500,X
    INX
    BNE *-9
    LDA $C200,X   ; 2. lap
    STA $0600,X
    INX
    BNE *-9
    LDA $C300,X   ; maradék – 232 byte
    STA $0700,X
    INX
    CPX #$E8
    BNE *-11
```

Méret: `2 (LDX) + fullPages×9 + (rem > 0 ? 11 : 0)` byte szakaszonként. A combined mód megduplázza (screen szakasz + azonos color szakasz).

Párosítás a Map Editorral:

A Map Editor **Files → Save map + color RAM (.bin)** menüpontja egyetlen binárist exportál, ahol az első `size` byte screen code, a következő `size` byte pedig color RAM érték. Használd a MAP_COPY-t **Combined .bin** bepipálva, és a **Source addr**-t oda mutasd, ahova ez a fájl betöltődik (pl. INCBIN-nel a `$C000`-en):

```
* = $C000
    INCBIN "map-color.bin" @ $C000   ; screen code-ok $C000–$C3E7, color $C3E8–$C7CF
* = $0801
    ; ...
    MAP_COPY src=$C000 dst=$0400 size=1000 combined color_dst=$D800
```

Expert-mód szintaxis:

```
.map_copy $C000, $0400, 1000               ; csak screen
.map_copy $C000, $0400, 1000, auto, $D800  ; combined (color a src+size címen)
.map_copy $C000, $0400, 1000, $C3E8, $D800 ; explicit color forráscím
```

MAP_COPY16X16

Egy 16×16 karakteres területet másol egy kompakt 256 byte-os screen-code blokkból plusz egy hozzá tartozó 256 byte-os Color RAM blokkból. Charset Canvas exportokhoz és kis tile/kép darabokhoz való, ahol tizenhat különálló MAP_COPY sor zajos lenne.

Alapértelmezett elrendezés:

Adat	Alapértelmezett cím
16×16 screen code-ok	Forráscím (`src`)
16×16 color értékek	`src + 256`

Adat	Alapértelmezett cím
Screen RAM cél	`$0400 + row×40 + col`
Color RAM cél	`$D800 + row×40 + col`

Expert-mód szintaxis:

```
.map_copy16x16 $3000, 12, 4
.map_copy16x16 $3000, 12, 4, $0400, $3100, $D800
```

A rövid forma a screen byte-okat a `$3000`-ről, a color byte-okat a `$3100`-ról másolja, és a 16×16 blokkot a 12. oszlop, 4. sorba helyezi. Az érvényes bal-felső pozíciók `col = 0..24` és `row = 0..9`, így a teljes 16×16 terület a 40×25-ös C64 szövegképernyőn marad.

Generált viselkedés:

- Tizenhat inline sor-másolást generál.
- Minden sor 16 screen byte-ot és 16 color byte-ot másol.
- Nem kell JSR; a kód közvetlenül a makró pozíciójára kerül.
- Normál karakter móddal és multicolor karakter móddal is működik; a Color RAM byte-ok hordozzák minden cella karakterszínét / multicolor engedély bitjét.

Tipikus párosítás Charset Canvas-szal:

```
* = $0801
    SEI
    LDA #$00
    STA $D021
    LDA #$05
    STA $D022
    LDA #$0D
    STA $D023
    .map_copy16x16 $3000, 12, 4
loop:
    JMP loop

.incbin "16x16+ram+color.bin", $3000
.incbin "16x16+charset.bin", $2000
```

SPRITE_ANIM

Minden híváskor lépteti egy sprite animációs frame-jét, és frissíti a VIC-II sprite adat-pointerét. Tárolj frame-enként egy byte-ot egy táblában (a sprite adat-lap száma = `data_address / 64`), mutass rá a SPRITE_ANIM-mal, és hívd game frame-enként egyszer — nem kell JSR.

Mező	Leírás
Sprite #	Sprite szám 0–7
Frame list address	A frame-tábla hex címe — frame-enként egy byte, minden byte = sprite lap (`data_addr / 64`)
Frame count	Frame-ek száma összesen (1–255)

Mező	Leírás
Frame ZP	Frame-számlálóként használt zero-page byte (pl. `FB`)

Generált ASM (sprite 0, 4 frame, ZP `$FB`, lista a `$C100`-on):

```
    INC $FB         ; frame-számláló léptetése
    LDA $FB
    CMP #$04        ; frame-szám
    BCC *+6         ; ha számláló < szám, ugorjuk át a resetet
    LDA #$00
    STA $FB
    TAX             ; X = aktuális frame-index
    LDA $C100,X     ; ehhez a frame-hez tartozó sprite adat-lap betöltése
    STA $07F8       ; sprite 0 pointer frissítése ($07F8 + sprite#)
```

Méret: 19 byte.

Tipikus használat:

```
frameTable:
    .byte $21, $22, $23, $24   ; 4 frame a $0840, $0880, $08C0, $0900 címeken

gameloop:
    WAIT_RASTER ($FF)
    SPRITE_ANIM (sprite=0, list=$C100, count=4, zp=$FB)
    JMP gameloop
```

Expert-mód szintaxis:

```
.sprite_anim spriteNum, frameListAddr, frameCount, zpByte
; példa:
.sprite_anim 0, C100, 4, FB
```

> **Tipp:** tedd a frame-táblát RAWBYTES blokként fix címre. A számláló ZP byte-ot (`$FB`) `$00`-ra kell inicializálni az első hívás előtt. Ha a kódod máshoz használja a `$FB`-t, válassz szabad ZP helyet.

SCORE_BCD

Egy fix pontszámot ad hozzá egy memóriában tárolt több-byte-os BCD pontszámhoz, majd minden számjegyet screen RAM-ba renderel screen-code karakterként. A 6502 decimal módot (`SED` / `CLD`) használja carry-biztos BCD aritmetikához — nem kell kézzel carryzni.

Mező	Leírás
Score address	A BCD pontszám-byte-ok hex címe (pl. `C200`). Alsó byte először.
Digits	BCD byte-ok száma (minden byte két számjegyet tárol: `$99` = „99"). `4` byte = akár 99999999.
Add points	Híváskénti decimális hozzáadandó érték (pl. `100`).
Screen address	Hová írja a számjegy screen code-okat (pl. `0400`). Számjegyenként egy byte (felső nibble először).

Expert szintaxis:

```
.score_bcd $C200, 4, 100, $0400
```

Generált ASM (4 byte, +100 pont, pontszám a `$C200` -on, screen a `$0400` -on):

```
    SED             ; BCD / decimal mód engedélyezése
    CLC
    LDA $C200       ; 0. byte (1–2. számjegy)
    ADC #$00        ; a 100 alsó byte-ja BCD-ben = $00
    STA $C200
    LDA $C201       ; 1. byte (3–4. számjegy)
    ADC #$01        ; a 100 középső byte-ja BCD-ben = $01 (carry propagál)
    STA $C201
    LDA $C202
    ADC #$00
    STA $C202
    LDA $C203
    ADC #$00
    STA $C203
    CLD             ; vissza bináris módba

    ; számjegyek renderelése
    LDX #$00
digit_loop:
    LDA $C200,X
    PHA
    LSR : LSR : LSR : LSR  ; felső nibble → alsó nibble
    ORA #$30        ; + '0' screen code
    STA $0406,X     ; jobbra igazítva: screen_addr + (digits*2 - 2) - X*2
    PLA
    AND #$0F        ; alsó nibble
    ORA #$30
    STA $0407,X
    INX
    CPX #$04
    BNE digit_loop
```

Méret: `3 + digits×8` byte (SED + CLC + CLD overhead + BCD byte-onként 8 byte az ADC-hez + display hurok).

Expert-mód szintaxis:

```
.score_bcd $C200, 4, 100, $0400
```

> **Tipp:** inicializáld a pontszám-byte-okat `$00` -ra induláskor. A pontszám-cím zero page-ben vagy abszolút RAM-ban legyen — ne ROM-ban. A screen-cím a legbaloldalibb számjegycellára mutasson; a számjegyek balról jobbra íródnak (legnagyobb helyi értékű byte először).

> **BCD tartomány:** `digits=4` byte → 8 decimális számjegy → max pontszám 99 999 999. Minden byte két BCD számjegyet kódol: `$00` – `$99` .

---

10. Debugger integráció

Az app **RetroDebuggert** támogat külső C64 debuggerként. Ez a lefordított programból generált breakpointokat, szimbólumokat és autostart flag-eket kap.

RetroDebugger

A RetroDebugger egy platformfüggetlen Commodore 64 debugger breakpoint-támogatással, memória-vizsgálattal és címke-tudatos disassemblyvel.

Beállítás: a **Beállítások → Fájlok és útvonalak → Retro Debugger** alatt mutasd a `RetroDebugger` binárisra.

Indítás: kattints a **Debug (RetroDebugger)** gombra az eszköztáron. Az app:

1. Assembleálja a programot egy `.prg` fájlba egy ideiglenes könyvtárban.
2. Ír egy **breakpoints fájlt** — minden töréspont-blokk és Expert töréspont-sor címével.
3. Kiírja a **szimbólumokat** VICE / RetroDebugger címke-formátumban (`al C:addr .name`): címkék, konstansok és változók.
4. C64Debugger-stílusú sidecarokat ír a PRG mellé: `.dbg` (forrástérkép), `.sym` és `.vs` .
5. Elindítja a RetroDebuggert a PRG-vel, a töréspontokkal, a szimbólumokkal és a lenti flag-ekkel.

Breakpoint blokkok

Kattints a breakpoint ikonra (●) bármelyik utasításblokkon, hogy breakpointként kapcsold. A breakpointolt blokkok pirosan emelődnek ki. A címeik minden debugger-indításkor a breakpoints fájlba kerülnek.

Debugger flag-ek

A **Beállítások → Általános → RetroDebugger** alatt vannak.

Kapcsoló	Flag	Hatás
`-jmp` BE	`-jmp $ADDR`	Betöltés után közvetlenül a program kezdőcímére ugrik
`-unpause` BE	`-unpause`	Betöltéskor azonnal folytatja a debuggert
`-wait` BE	`-wait <ms>`	`<ms>` milliszekundumot vár a folytatás előtt — 500 ms vagy 1000 ms

> **Tipp:** a legtöbb programnál kapcsold be a `-jmp` -et és a `-unpause` -t az azonnali autostarthoz. Használj `-wait 500` -at vagy `-wait 1000` -et, ha a programod IRQ-kat vagy SID zenét állít be, aminek idő kell az inicializáláshoz az első raszter előtt.

VICE Monitor (hot reload, élő változók, profiler)

A **VICE Monitor** eszköztár-gomb egy nem modális ablakot nyit, amely a futó VICE-szal a binary monitorán át beszél. A **Run (VICE)** a 6502-es porton nyitott monitorral indítja a VICE-t (**Beállítások → Fájlok és útvonalak → VICE Emulator → VICE remote monitor**, alapból bekapcsolva). A más módon indított VICE-t zárd be és indítsd újra. A fejléc a kapcsolat állapotát mutatja; böngészős előnézetben *Csak az asztali alkalmazásban érhető el* áll ott.

Gomb	Mit csinál
Hot reload	A lefordított programot a futó VICE RAM-jába írja (csak a foglalt tartományokat — a hézagokhoz nem nyúl). Adathoz jó: sprite, tábla, szöveg.
Hot reload + újraindítás	Ugyanez, utána a SYS címtől újraindít (a verem kiürül; a végső `RTS` a `READY.` -hez tér vissza).

Gomb	Mit csinál
Automatikus hot reload szerkesztés után	Minden szerkesztés után kb. egy másodperccel betölt és újraindít, ha a program fordul.
Élő változók	0,4 mp-enként frissíti a watch-listát; a megváltozott érték felvillan.
Profilozás	Kb. 5 másodpercig mintavételezi a programszámlálót (újra kattintva hamarabb leáll).

Watch-lista: írj be egy címkét vagy címet — `score`, `$D020`, `53280`, vagy bájtszámmal `table, 8` —, majd **+** vagy Enter. Minden sor a címet, a bájtokat hexában és 1–2 bájtnál a decimális értéket mutatja. Ultimate Basic módban a nevek az utolsó UB buildből jönnek (változók, címkék, sub-ok, tömbök).

Profiler eredmény: két tábla. A *Rutin* a mintákat a legközelebbi címke szerint csoportosítja (Ultimate Basicben `sub` szerint), %-os aránnyal és azokkal a raszter sorokkal, amelyekben futott — a KERNAL vagy BASIC ROM-ban töltött idő külön sor. A *Sor* a legdrágább forrássorokat sorolja fel; egyre kattintva oda ugrik. Minden minta egy pillanatra megállítja az emulátort, ezért a profil statisztikai kép, nem pontos ciklusszámlálás.

---

11. Tudásbázis linkek

Gyorsreferencia linkek az appban a **Knowledge Base** alatt:

Erőforrás	URL
6502 Opcodes Reference	http://www.6502.org/tutorials/6502opcodes.html
C64 KERNAL Functions	https://sta.c64.org/cbm64krnfunc.html
C64 Memory Map	https://sta.c64.org/cbm64mem.html
C64 Color Codes	https://sta.c64.org/cbm64col.html
VIC-II Article	https://www.cebix.net/VIC-Article.txt
C64 Codebase	https://codebase.c64.org/
The Turbo Assembler	https://turbo.style64.org/
RetroDebugger	https://github.com/slajerek/RetroDebugger/
C64 OS Development Environment	https://www.c64os.com/c64os/programmersguide/?s=devenvironment
png2c64 Image Converter	https://png2c64.app/
Restore64 visszafejtő (C64 PRG → KickAssembler / ACME forrás)	https://restore64.dev/

Beépített C64 és debugger

Az app egy teljes C64-et tartalmaz: a VICE 3.10-et (x64sc) WebAssembly-re fordítva. Nem kell telepíteni vagy beállítani, és külső VICE nélkül is működik.

Program indítása: válaszd a **Beépített C64** pontot az osztott **Run** menüben, majd nyomd meg a **Run** gombot. A program lefordul, a gép hard resetet kap, és a program elindul: `$0801`-es BASIC sorral `RUN`, egyébként `SYS` a betöltési címre. Az eszköztár gombja futtatás nélkül nyitja meg az ablakot; az ablak eszköztárának első gombja újrafordít és újraindít.

A billentyűzet a C64-é, amíg a képernyőn van a fókusz (kattints rá). A **Tab** a CTRL, az **Alt** a Commodore billentyű, és Ctrl+V-vel szöveget illeszthetsz be. Kiválasztott **Joystick** port mellett a nyilak és a Szóköz vagy a jobb Ctrl a joystick.

A Files menü

Menüpont	Mit csinál
PRG betöltése…	Autostarttal elindít egy `.prg`-t
D64 csatolása…	A 8-as meghajtóra csatolja a képet, resetel, majd `LOAD"*",8,1` és `RUN`
Cartridge csatolása…	Csatol egy `.crt`-t és resetel
Lemez / Cartridge leválasztása	Leveszi a képet; a cartridge leválasztása resetel is

A státuszsor mutatja, mi van csatolva, valamint a 8-as meghajtó LED-jét és a fej trackjét. Betöltés közben a LED folyamatosan ég. A villogó LED azt jelenti, hogy a meghajtó hibát jelez, például nem talál vagy nem tud olvasni egy fájlt: a LED piros lesz, és a státuszsorban **Lemezhiba** látszik, amíg a meghajtó nem törli a hibát. A csatolt fájlokat nem jegyzi meg. A saját programod indítása a szerkesztőből előbb leveszi a csatolt cartridge-et, különben a cartridge indulna helyette.

A gép beállításai az eszköztáron:

Beállítás	Értékek	Megjegyzés
Videó	PAL, NTSC	Hard reset
SID chip	6581, 8580	Azonnal érvényes
SID motor	reSID, FastSID	Azonnal érvényes. A VA SID Core itt nem érhető el: ez a gép a VICE SID-motorjait használja
REU	Ki, 128 KB – 16 MB	Hard reset
Joystick	—, Port 1, Port 2	

Az eszköztáron van még **Reset**, **Szünet**, **Gyorsítás (warp)**, **Hang**, **CRT szűrő** és **Teljes képernyő**. A CRT szűrő a VICE CRT-emulációját, sima nagyítást és scanline-okat ad; a scanline-ok akkor jelennek meg, ha egy C64-sor legalább 3 képpont magas. A teljes képernyő a képernyőre duplán kattintva is indul, Esc-kel vagy újabb dupla kattintással lehet kilépni. Az ablak a fejlécénél mozgatható, a jobb alsó sarkánál méretezhető. A helye, mérete, nyitott állapota és a beállításai a következő indításkor visszatérnek.

Hibakeresés: nyomd meg a **Debug** gombot, miközben a futtatási mód **Beépített C64**. A Debug gomb a valódi hardveres módokban is ezt a debuggert használja, ha a **Visual Assembler** a választott debugger. VMA-val, Kick projekttel, ACME-vel és Ultimate Basic-kel egyaránt működik. A program az első elért töréspontnál áll meg, töréspont nélkül az első utasításánál. Az ablak bezárása vagy a debugger-panel elrejtése lezárja a munkamenetet: a töréspontok megszűnnek, a megállított program pedig fut tovább (bezárt ablaknál az újranyitás után). Az Ultimate emulátor és a VICE debugger-ablaka bezáráskor ugyanígy tesz.

A debugger kiválasztása: a **Beállítások → Általános → Debugger → Debug gomb** értéke **Visual Assembler** vagy **RetroDebugger**. A **RetroDebugger** csak akkor választható, ha az útvonala be van állítva a **Beállítások → Fájlok és útvonalak** alatt; ha még nem választottál, RetroDebugger, amennyiben be van állítva, különben Visual Assembler. A

Visual Assembler a futtatási mód gépén keres hibát: a saját VICE-odban (*Run as PRG*, *Run via D64*), az Ultimate emulátoron vagy a beépített C64-en (a valódi hardveres módokban, C64 Ultimate és Kung Fu Flash 2, is ezen). A **RetroDebugger** a VICE-os és a hardveres módokban a külső programot nyitja. A *Beépített C64* és az *Ultimate emulátor* futtatási mód mindig a saját debuggerét használja.

Hibakeresés VICE-ban: ha a **Visual Assembler** van kiválasztva, a *Run as PRG* és *Run via D64* módban a **Debug** ugyanazt a debugger-ablakot nyitja a saját VICE-odhoz: a VA a binary monitorral indítja a VICE-t (a **VICE monitor port**on, alapból 6502), vagy az ott már futót használja, élesíti a töréspontokat és autostarttal indítja a programot, amely töréspont nélkül az első utasításán áll meg. A töréspontok, a lépés be / át / ki, a regiszterek, a disassembly és a memória ugyanúgy működik, mint a beépített C64-ben, és a szerkesztő követi a CPU-t. Az ablak bezárása után a program tovább fut a VICE-ban. Ha a **RetroDebugger** van kiválasztva, a VICE-os módok a korábbiak szerint a RetroDebuggert nyitják.

A CPU a töréspont utasítása **előtt** áll meg, így a regiszterek és a memória a lefutása előtti állapotot mutatják.

Gomb	Billentyű	Mit csinál
Folytatás	F5	Fut a következő töréspontig
Megállítás	F5 futás közben	Megállítja a futó programot
Lépés be	F11	Egy utasítást futtat
Lépés át	F10	A JSR-t a visszatérésig futtatja
Lépés ki	Shift+F11	Fut a rutin RTS / RTI-je utáni utasításig

A billentyűk akkor működnek, ha a fókusz az ablakban van, de nem a C64 képernyőjén, mert ott az F5 C64-billentyű.

A **Debugger panel** tartalma:

- **Regiszterek:** a PC, A, X, Y és SP szerkeszthető hex mezők, a flagek gombok. A módosítás a CPU továbbfutásakor érvényes.
- **Disassembly** a PC körül, a programod címkéivel. A sor elején lévő pontra kattintva tehetsz vagy vehetsz le töréspontot.
- **Memória:** a címet `$hex`, decimális szám vagy címke alakban add meg. Az olvasásnak nincs mellékhatása, így a `$DC0D` megnézése nem nyugtáz megszakítást.
- **Töréspontok:** mindegyik címe és `fájl:sor` helye.

A töréspontok a szerkesztő sorszám-gutteréből (a build minden nyitott füle), a Block módú töréspontokból és a disassemblyből jönnek. A szerkesztőbeli változás a következő Folytatásnál vagy lépésnél számít. Megálláskor a szerkesztő megnyitja a fájlt és jelöli a sort, Block módban a blokkot jelöli ki.

A **VICE Monitor** ablak továbbra is csak külső VICE-szal működik, ehhez a géphez nem kapcsolódik.

12. D64 export és futtatás

A programodat (és további adatfájlokat) egy C64 D64 lemezképbe csomagolhatod, és VICE-ban indíthatod — vagy a lemezképet máshol való használatra exportálhatod.

Osztott Run gomb

Az eszköztár **Run** gombja **osztott gomb**:

Rész	Művelet
▶ **Run** (fő)	A jelenleg kiválasztott futtatási módot hajtja végre
▾ (nyíl)	Megnyitja a mód-választót

Elérhető futtatási módok:

Mód	Leírás
Run as PRG	Assembleálás egy ideiglenes `.prg`-be, és VICE közvetlen indítása. Klasszikus viselkedés.
Run via D64	Assembleálás, egy `.d64` lemezkép építése (c1541-gyel), a beállított extra fájlok hozzáadása, majd VICE indítása a lemezről. Ezt használd, amikor a programod futásidőben tölt fájlokat (pl. a LOADFILE makróval).
Run on hardware	Assembleálás PRG-be, és küldés egy **1541 Ultimate / Ultimate 64** eszközre a helyi hálózaton. Lásd a 13. szakaszt.
Kung Fu Flash 2	Assembleálás PRG-be, és feltöltés USB soros kapcsolaton egy **Kung Fu Flash 2** cartridge-dzsel felszerelt valódi C64/C128-ra. Lásd: Kung Fu Flash 2.

A menü csak azokat a módokat mutatja, amelyek eszköze be van állítva: az első kettőhöz a VICE, a C64 Ultimate-hez az eszköz címe a **Beállítások → Integrációk** alatt, a Kung Fu Flash 2-höz a port (induláskor felismeri, vagy ott megadható), az Ultimate emulátorhoz a program, a firmware és a ROM-mappa. A **Beépített C64** mindig ott van.

A kiválasztott mód munkamenetek közt mentődik. Ha az eszköze hiányzik, a Futtatás a VICE-t vagy a beépített C64-et használja, a mentett mód pedig visszajön, amint az eszköz be van állítva.

Export to D64 dialógus

A **Menu → Build → Build D64** nyitja meg (a Run via D64 is). A dialógus lehetővé teszi:

1. A **lemeznév** (max 16 karakter) és a **program neve** beállítása — ezek jelennek meg a C64 lemez-könyvtárban.
2. **Extra fájlok hozzáadása** — kattints a **+**-ra bármely binárisfájl (`.prg`, `.bin`, `.sid`, stb.) kiválasztásához. Minden extrához:
   - **Name** — hogyan jelenik meg a D64 könyvtárban (max 16 karakter, automatikus nagybetű).
   - **Addr** (betöltési cím, opcionális) — ha megadott, egy 2 byte-os PRG fejléc kerül elé. Hagyd üresen, hogy nyers byte-okat írjon fejléc nélkül.
   - **Dst** (kitömörítési cél, csak EXO-val) — hova rakja a depacker az adatot a C64-en. Ha az EXO engedélyezett, az extra `exomizer mem -l <Addr> file@<Dst>`-vel (nyers fájl a Dst címen) tömörödik a D64-be írás előtt.
   - **EXO** — jelölőnégyzet, amely bekapcsolja a visszafelé `mem`-módú tömörítést ehhez a bejegyzéshez. A lemezen lévő méret jellemzően az eredeti 5–20%-a.
3. Kattints az **Export**-ra a `.d64` fájl generálásához a VICE `c1541` eszközével.

Párosítás az EXODECRUNCH-csal: amikor EXO=be beállítással szállítasz egy fájlt, az azt olvasó programnak az **Addr** címre kell LOAD-olnia (sec=1, a fájl saját PRG fejléce), majd hívnia az **EXODECRUNCH** makrót, hogy visszafelé kicsomagolja a **Dst**-re. A teljes mintáért lásd az `exo-multicolor-demo` mintát.

D64 metaadat a projektekben

A lemeznév, a program neve és az extra fájl-lista a projekt JSON-ban mentődik (a `d64` kulcs alatt). Amikor újratöltesz egy projektet vagy D64 metaadatot tartalmazó mintát, az extrák automatikusan visszaállnak — nem kell minden alkalommal újra hozzáadni őket.

A **loadfile-demo** minta előre be van állítva a `DEMO-COLORS.PRG`-vel extra fájlként. Válaszd ki, nyisd meg a **Run via D64**-et, és kattints a **Run**-ra a teljes betöltési folyamat működés közbeni megtekintéséhez.

> **Követelmény:** a D64 export és a Run via D64 is megköveteli a VICE (`c1541`) beállítását a Beállítások → Fájlok és útvonalak alatt.

D64 Editor (meglévő lemezkép böngészése és szerkesztése)

A Table Generator utáni eszköztár-ikon nyitja meg a **D64 Editort** — egy önálló eszközt egy meglévő `.d64` lemezkép közvetlen szerkesztésére, a jelenleg nyitott programtól függetlenül. A fenti Export to D64 dialógustól eltérően (ami mindig egy *új* lemezt épít a lefordított PRG-ből), a D64 Editor a lemezképet a helyén szerkeszti `c1541`-en keresztül, így egy könnyűsúlyú lemezkezelőként is funkcionál.

Files ▾ menü:

Menüpont	Művelet
New D64…	Célelérési út kiválasztása, majd egy frissen formázott, üres lemezkép létrehozása ott.
Open D64…	Meglévő `.d64` fájl kiválasztása és a directory betöltése.
Save As…	A jelenleg nyitott lemezkép másolása egy új útvonalra, és a szerkesztés folytatása a másolaton.
Run in VICE	A jelenleg nyitott lemezkép közvetlen indítása VICE-ban (`-drive8type 1541`).

Eszköztár:

Ikon	Művelet
Add program	Helyi fájl kiválasztása és beírása a lemez directory-jába.
Extract selected	A kijelölt bejegyzés byte-jainak mentése helyi `.prg` fájlba.
Rename selected	A bejegyzés nevének helyben szerkesztése a táblázatban — Enter megerősít, Escape megszakít.
Delete selected	A kijelölt bejegyzés törlése a lemezről.
Refresh	A directory újraolvasása, pl. ha közben másik eszközzel módosítottad a lemezt.
Save	A munkapéldány módosításainak visszaírása a valódi `.d64` fájlba, előtte automatikus biztonsági mentéssel. Csak akkor aktív, ha van mentetlen módosítás.

Program hozzáadása: ha már `.prg`-re végződő fájlt választasz, csak egy **Name** és egy lemez **Type** (PRG/SEQ/USR/REL) mezőt kér — a `.prg` már tartalmazza a saját betöltési cím fejlécét, így változtatás nélkül kerül fel. Bármely más fájl (pl. nyers `.bin`) kiválasztásakor emellett megjelenik:

- **Load address** (hex, opcionális) — 2 byte-os PRG fejléc elhelyezése ezen a címen; hagyd üresen, hogy a byte-ok nyersen kerüljenek fel.
- **Decompress address** (hex, opcionális) — csak Exomizerrel együtt használatos; a célcím, ahová a depacker kicsomagolja az adatot.
- **Exomizer** jelölőnégyzet — a fájl tömörítése feltöltés előtt, ugyanazokkal a `mem` / `sfx` tömörítési módokkal, mint a fenti Export to D64 dialógus extra fájljainál.

A directory lista ugyanazzal a fonttal és nagybetűs stílussal jelenik meg, mint egy valódi C64 `LOAD"$",8` lista.

Mentési munkafolyamat: minden szerkesztés — hozzáadás, kimentés, átnevezés, törlés, és a Block Editor byte-szintű szerkesztései — egy privát munkapéldányon történik, nem a valódi fájlodon. A **Save** eszköztár-gomb (csak mentetlen módosítás esetén aktív) visszaírja a munkapéldányt a valódi `.d64`-re, előtte automatikus biztonsági másolatot készítve; a fejléc • jelzést mutat, amíg van mentetlen módosítás, és a szerkesztő bezárása vagy másik lemez megnyitása mentetlen módosításokkal megerősítést kér ahelyett, hogy csendben eldobná őket. A **Save As…** (Files ▾ menü) a munkapéldányt egy új útvonalra írja, és ott folytatja a szerkesztést.

Block Editor: az eszköztár rács-ikonja egy nyers track/sector nézetre vált a directory alatt — kattintható blokktérkép a BAM szabad/foglalt bitmapje szerint színezve (a BAM-ot és a directory láncot külön jelölve), byte-onkénti hex grid a kiválasztott 256 byte-os blokkhoz, valamint előző/következő szektor navigáció. A block-szintű szerkesztések ugyanazon a munkapéldány/Save munkafolyamaton mennek át, mint a fenti directory-műveletek.

> **Követelmény:** az Export to D64-hez hasonlóan a D64 Editor is megköveteli a VICE (`c1541`) beállítását a Beállítások → Fájlok és útvonalak alatt.

12b. CRT export (Magic Desk 64K cartridge)

A **Menu** → **Build** → **Build CRT** egy Commodore 64 cartridge képet (`.crt` , **cartridge type 19 — Magic Desk / Domark / HES Australia**) állít elő, amely fut VICE-ban, TheC64-en, valódi hardveren EasyFlash / Kung Fu Flash révén, és 1541 Ultimate II+ cartridge slotokban. Elérhető **blokk módban** és **Expert módban** is, és — a jelenlegi buildtől — **UltimateBasic módban** is.

Mi kerül a cartba

- **8 × 8 KB bank** a `$8000`-en, bank-váltás a `$DE00`-n át (Magic Desk konvenció: alsó 3 bit = bank, 7. bit = cart letiltása).
- A **0. bank** egy 128 byte-os fejlécet + boot loadert tartalmaz:
  - `$8000/$8002` cold + warm start vektorok a `$8009`-re mutatnak.
  - `$8004–$8008` = a `CBM80` szignatúra, amit a KERNAL reset kód megkövetel.
  - `$8009–$807F` = a loader: SEI / stack init / `JSR $FDA3` (IOINIT) / `JSR $FD50` (RAMTAS) / `JSR $FD15` (RESTOR) / `JSR $FF5B` (CINT), majd a 60 byte-os **másoló rutint** (`$8044–$807F`) a `$0100`-ra másolja, és oda ugrik. A RAM-ból futó rutin a payloadot a cart ROM-ból a betöltési címére másolja, bankot vált a `$DE00`-on keresztül, amikor a `$FC` eléri az `$A0`-t, letiltja a cartot `LDA #$80 : STA $DE00`-val, és a belépési pontra `JMP`-el. (RAM-ból kell futnia: a bankváltás kicseréli a `$8000` alatti ROM-ot, a loader saját kódját is.)
- A **payload** a 0. bankban a `$8080`-on kezdődik, és szükség szerint az 1–7. bankba folyik át. Maximum payload = `8 * 8192 – 128 = 65 408 byte`.

Betöltési cím és belépési pont

A Build CRT sosem használ Exomizert (a depacker nem futtatható cart ROM-ból). A szabványos autostart pipeline-nal fordítja az aktuális fület, és a betöltési címet a PRG fejlécből, a belépési pontot a SYS célból veszi:

- **Blokk / Expert mód BASIC SYS stubbal be:** load = `$0801`, belépés = a SYS cél (jellemzően `$080D` vagy a felhasználó originje).

- **Blokk / Expert mód BASIC SYS stubbal ki:** load = felhasználói origin (a klasszikus `$0801 → $C000` fallbackkel), belépés = betöltési cím.
- **UltimateBasic mód:** a load és a belépés is a UB fordító map-jéből jön (`build.map.loadAddress`). A payloadon belüli UB autostart stub ezután pontosan úgy fut le, ahogy lemezről `LOAD "...",8,1 : RUN` után tenné.

Az ASM kimenetben látott origin cím megőrződik; a loader egyszerűen bemásolja a lapos memória-képet a PRG-ből RAM-ba, és a belépési pontra ugrik, miután a cart ROM ki van mappelve.

Méretkorlát

Mivel a payload lineárisan tárolódik — egyetlen lapos kép a legalacsonyabb címtől a legmagasabbig, nullákkal töltött résekkel — egy egymástól távoli ORG szegmensekkel rendelkező program (pl. `$0801` + `$C000` + `$E000`) minden közbeeső byte-ot beleszámol a 65 408 byte-os keretbe. Ha túlléped a korlátot, a build hibával megszakad — vagy tömörítsd a memória-elrendezést, vagy oszd szét az adatot.

> ⚠️ **Kiemelt figyelmeztetés — olvasd el CRT szállítása előtt**
> A loader a szabványos reset szekvencia részeként hívja a KERNAL `RESTOR` **($FD15)**-öt. Ez szándékosan visszaírja a szabványos I/O vektorokat a `$0314/$0315` , `$0316/$0317` , `$0318/$0319` , `$0328/$0329` és társai címeken a ROM alapértékeikre. Következmények:
>
> - **Minden IRQ / NMI / BRK hook, amit a CRT bootolása előtt állítottak be, törlődik.** A programodnak magának kell beállítania őket a belépés után — pontosan úgy, mint egy friss `LOAD "",8,1 : RUN` szalagról/lemezről.
> - Az **UltimateBasic programok**, amelyek nem alapértelmezett KERNAL vektorok élő voltára támaszkodnak a belépéskor, explicit `SYS` -t vagy init hívást igényelhetnek az autostart stubban. A szabványos UB autostart out of the box működik; a vektorokat *RUN előtt* hookoló bővítő könyvtárak nem.
> - A **CIA1 / CIA2** újrainicializálódik az `IOINIT` által. Az egyéni timer-beállításokat (raszter IRQ-k, zenelejátszó CIA-A) újra kell programozni a belépés után.
> - A másoló rutin a **RAM-ban, a `$0100–$013B` területen** fut, így a bankváltás és a záró `STA $DE00` nem húzza ki a kódot a CPU alól. Belépéskor ne támaszkodj a `$0100–$013B` tartalmára, és a payload ne essen erre a területre, se a loader által használt `$02–$04` / `$FB–$FE` zero page byte-okra.
>
> Ha egy CRT fut VICE-ban, de valódi hardveren elbukik, először azt ellenőrizd, hogy a program feltételez-e egy konkrét KERNAL vektort vagy CIA timer állapotot a belépéskor. Állítsd be az állapotot explicit módon az init rutinodban, és mindkettőn ugyanúgy fog viselkedni.

Kompatibilitás

Platform	Állapot
VICE (`x64sc` , `x64`)	Működik a **File** → **Attach cartridge image** menüvel.
TheC64 / TheC64 Mini	Működik a beépített cartridge loaderrel.
Kung Fu Flash	Működik — natív Magic Desk mód.
EasyFlash cartridge	Működik, Magic Deskként programozva.
1541 Ultimate II+ / Ultimate 64	Működik a **Cartridge** → **Load cart image** menüvel.
Chameleon / Turbo Chameleon	Működik.

---

13. Beállítások

A **Menu** → **Beállítások...** nyitja meg (korábban *Hardware Settings*). A dialógusnak három füle van: **Általános** (assembler mód, VMA fordító, BASIC SYS stub, Run with Exomizer, ASM kimenet, SID előnézet, debugger kiválasztása és C64 Debugger paraméterek, snapshot / verziókezelés), **Fájlok és útvonalak** (VICE, RetroDebugger, frissítési szerver, ACME, Exomizer, git) és **Integrációk** (C64 Ultimate, Ultimate emulátor, Kung Fu Flash 2). Az alábbi szakaszok a külső eszközöket és hardvereket írják le.

VMA fordító

Minden VMA (és Kick projekt) forrást a nyílt forráskódú, Rustban írt VMA fordító fordít, amely az alkalmazásba van építve: a Futtatás, a Build, a D64, a CRT, a debugger és minden élő nézet (disassembly, monitor, memóriatérkép, ciklus-gutter, tippek, ASM nézet) ugyanazt a fordítást használja. Fordító-beállítás már nincs (a 2.7.7-ig JavaScript fordító is volt). A fordításhoz az asztali alkalmazás kell. A VMA verziója az About ablakban látszik.

Ugyanez a fordító parancssorból is fut (VMA 2.1.0-tól): a `vma build game.asm` a `game.prg`-t írja, a `--crt game.crt` Magic Desk cartridge-et (ugyanazokkal a bájtokkal, mint a **Build CRT**), a `--d64` pedig 1541-es lemezképet a programmal (a `--add` további fájlokat tesz rá) — c1541 nélkül. Részletek a VMA tároló kézikönyvében.

VICE emulátor

Beállítás	Leírás
Select VICE	Tallózz az `x64sc` (vagy `x64`) VICE futtatható fájlra
Status	Mutatja, hogy a futtatható útvonala érvényes és elérhető-e
VICE remote monitor	A 6502-es porton nyitott binary monitorral indítja a VICE-t; a VICE Monitor ehhez kapcsolódik

A VICE szükséges a **Run as PRG**, **Run via D64** és **Export to D64** műveletekhez.

Exomizer

Beállítás	Leírás
Select Exomizer	Tallózz az `exomizer` futtatható fájlra
Border flash during decompression	Bekapcsolva az SFX-tömörített PRG-k az exomizer beépített `-x1` gyors keret-villantás effektjét használják; kikapcsolva `-n` kerül átadásra a csendes kitömörítéshez
Status	Mutatja, hogy a futtatható útvonala érvényes és elérhető-e

Munkafolyamat:

1. Telepítsd az Exomizer binárist:
   - **Windows:** töltsd le az előre elkészített `win32/exomizer.exe`-t a https://bitbucket.org/magli143/exomizer/wiki/Home vagy a https://csdb.dk/release/?id=244342 címről.
   - **macOS:** `brew install exomizer` (Magnus Lind hivatalos 3.1.2 buildjét telepíti).
2. Állítsd be az útvonalat a **Beállítások** → **Fájlok és útvonalak** → **Exomizer** alatt.
3. Kapcsold be a **Run with Exomizer** jelölőnégyzetet a **Settings menüben**.
4. Minden **Run** művelet (PRG, D64, hardver) és **Build** művelet (Build PRG, Build D64) mostantól `exomizer sfx`-szel tömöríti a lefordított programot indítás vagy mentés előtt: `sfx basic`-kel, ha a PRG BASIC sorral kezdődik

(kicsomagolás után a RUN az eredeti `SYS` sort futtatja, így a második RUN is működik), egyébként `sfx $<betöltési cím>`-mel.

Az Exomizer Windowson és macOS-en ugyanúgy működik — a CLI-t a Tauri backend hívja; az integrációban semmi nem platformfüggő.

Két tömörítési mód használódik belsőleg:

Mód	Használja	Hívási konvenció
`sfx basic` / `sfx $<cím>`	Build/Run with Exomizer kapcsoló (fő PRG út)	Önkicsomagoló PRG beépített dekruncherrel; a Border-flash beállítás vezérli `-x1` vs `-n`
`mem` (visszafelé)	Run via D64 → per-fájl **EXO** jelölőnégyzet	Minden extra fájlt `mem`-módú streamre tömörít; a program futásidőben csomagolja ki az **EXODECRUNCH** makróval és egy előre elkészített depackerrel (`samples/exo-decrunch.bin`)

> **Tipp:** ha az Exomizer útvonal nincs beállítva, de a jelölőnégyzet be van kapcsolva, indítás helyett egy beszédes hiba-toast jelenik meg. Kapcsold ki a jelölőnégyzetet tömörítés nélküli futtatáshoz.

Retro Debugger

Beállítás	Leírás
Select RetroDebugger	Tallózz a `RetroDebugger` binárisra
Status	Mutatja, hogy az útvonal érvényes-e

A teljes debugger dokumentációért lásd a 10. szakaszt.

C64 Ultimate / 1541 Ultimate

Lefordított PRG-ket futtass közvetlenül valódi hardveren a helyi hálózaton az Ultimate REST API-n keresztül.

Beállítás	Leírás
Host (IP)	Az eszköz IP-címe (pl. `192.168.1.100`)
Password	Opcionális — ha az eszköz hitelesítést igényel

Munkafolyamat:

1. Csatlakoztasd az 1541 Ultimate / Ultimate 64-et a helyi hálózatodhoz.
2. Add meg az IP-címét (és jelszavát, ha van) a **Beállítások** → **Integrációk** → **C64 Ultimate** alatt.
3. Válaszd a **Run on hardware**-t az osztott run menüből.
4. Kattints a ▶ **Run**-ra — a PRG lefordul, és HTTP POST-tal a `/v1/runners:run_prg`-re kerül. Az eszköz azonnal betölti és futtatja a C64-en.

Hardver nélkül: az osztott run menü **Ultimate emulátor** pontja az UE2-C64U-Emulator emulátoron futtat, amely a változatlan Ultimate firmware-t futtatja a számítógépeden. Külön kell telepíteni (lásd Az Ultimate emulátor telepítése lépésről lépésre), majd a **Beállítások** → **Integrációk** → **Ultimate emulátor (UE2)** alatt három dolgot kell megadni: az `ue2emu` programot, egy firmware-t (`.ue2` , `.elf` vagy `.app` , nincs mellékelve) és a ROM-mappát (egy 1541ultimate klón `roms/` mappája). A mappából csak a `chars.bin` kötelező (2048 bájt, a menü betűkészlete), a Commodore firmware a saját C64 ROM-jaival bootol. Az első futtatás a `.ue2`-t egy flash-képbe telepíti, firmware-enként külön, az

app konfigurációs mappájában, és minden indítás bekapcsolja a firmware webszolgáltatását, amely a Commodore firmware-ben alapból ki van kapcsolva. Az emulátor az első futtatáskor indul, és nyitva marad a következőkhöz; a REST API-ja a `127.0.0.1` címen, a **REST port**on figyel (alapból 8064). Az appal együtt vagy az **Emulátor leállítása** gombbal zárul. Ugyanebben a menüben a **D64 az Ultimate emulátoron** D64-et csomagol (ehhez a VICE `c1541` -e kell), az A meghajtóra csatolja, és futtatja az első programot. Amíg e két futtatási mód egyike van kiválasztva, a **VICE Monitor** ablak (hot reload, élő változók, profiler) a VICE helyett az emulátorhoz kapcsolódik. Ezekben a módokban a **Debug** az emulátor debuggerét nyitja meg: regiszterek, disassembly, memória és töréspontok külön ablakban, a kép az emulátor ablakában. A program az első elért töréspontnál áll meg, töréspont nélkül az első utasításánál; a szerkesztő sora követi a CPU-t, az F5, F10, F11 és Shift+F11 akkor is működik, ha a szerkesztőben van a fókusz. Engedélyezett megszakításoknál egy timer IRQ a lépést a KERNAL-ba viheti, mint a VICE monitorában. A VA az emulátor flash-képében a **Turbo Control**-t is a turbó-regiszterekre állítja, így a program a `$D031` -gyel állíthatja a CPU sebességét (reset után `$00` , azaz 1 MHz).

Minta: **Ultimate turbó demo (64 sprite)** (`samples/ultimate-turbo-demo.asm`) — a sebesség 3 másodpercenként vált 1 és 64 MHz között; 1 MHz-en golyók tűnnek el és csökken a képkockaszám, 4 MHz-től mind a 64 sprite 50 fps-sel fut. Csak bekapcsolt turbó-regiszterű Ultimate-en fut, máshol ezt írja ki. Az **Ultimate turbó kocka** (`samples/ultimate-box-demo.asm`) ugyanezt teszi a box minta kockájával: minél gyorsabb a CPU, annál gyorsabban forog a kocka (1 MHz-en 5, 16 MHz-től 50 képkocka másodpercenként). Az **Ultimate 64 paletta demó** (`samples/u64-palette-demo.asm`) az 1, 2, 3 és 0 billentyűkre a videó-palettát váltja (lásd: U64_PALETTE); a menüben bekapcsolt Command Interface kell hozzá.

> **Tipp:** nem kell USB kábel vagy driver — a REST API be van építve az Ultimate firmware-be. A számítógépednek és az eszköznek ugyanazon a helyi hálózaton kell lennie.

Az Ultimate emulátor telepítése lépésről lépésre

Az **Ultimate emulátor** és a **D64 az Ultimate emulátoron** futtatási módhoz három dolog kell, ami nem része a Visual Assemblernek: maga az emulátor, egy Ultimate firmware és egy ROM-mappa. Ezeket egyszer kell beállítani, a következő módon.

1. Telepítsd az emulátort (UE2-C64U-Emulator):

- **Windows:** töltsd le az emulátor Releases oldaláról az `ue2emu-<verzió>-windows-x86_64.zip` fájlt, és csomagold ki egy maradandó mappába, például `C:\Tools\ue2emu` . A fájlok maradjanak együtt: az `ue2emu.exe` -nek kellenek a mellette lévő DLL-ek.
- **macOS:** telepítsd Homebrew-val. A Visual Assembler ott magától megtalálja:

  ```
  brew install jondalar/ue2emu/ue2emu
  ```

- **Linux:** kész csomag még nincs; az emulátor dokumentációja szerint forrásból kell fordítani (Rust és libslirp kell hozzá).

2. Szerezz egy firmware-t. Az emulátor egy **C64 Ultimate** vagy **Ultimate 64 Elite II** firmware-ét futtatja változatlanul: a frissítőfájlt (`.ue2`), amelyet az eszközre is telepítenél, vagy egy saját fordítású `ultimate.elf` -et. Sem a Visual Assembler, sem az emulátor nem tartalmaz ilyet. A C64 Ultimate 1.1.0-s firmware-ével biztosan működik.

3. Szerezd be a ROM-mappát. Töltsd le a GitHubról az 1541ultimate tárolót (**Code** → **Download ZIP**, vagy `git clone`), és csomagold ki. Ebből a `roms` mappa kell. A Visual Assembler csak a benne lévő `chars.bin` -t ellenőrzi (2048 bájt, a menü betűkészlete).

4. Töltsd ki a beállításokat. Nyisd meg a **Beállítások** → **Integrációk** → **Ultimate emulátor (UE2)** részt:

Gomb	Mit válassz
ue2emu kiválasztása	Az 1. lépés `ue2emu.exe`-jét. Ha az emulátor a `PATH`-on van, vagy Homebrew-val telepítetted, a mező már ki van töltve.
Firmware kiválasztása	A 2. lépés `.ue2` (vagy `.elf`) fájlját.
ROM-mappa kiválasztása	Magát a `roms` mappát, ne a fölötte lévő tároló-mappát.
REST port	Maradjon **8064**, hacsak nem foglalt. Az emulátor a következő két portot is használja (vezérlés és VICE monitor).

5. Futtasd. Válaszd az osztott run menüben az **Ultimate emulátor**-t (a beállítások kitöltése után jelenik meg), és nyomd meg a ▶ **Run**-t. Az első futtatás `.ue2` esetén a firmware-t egy saját flash-képbe telepíti, ez kicsit tovább tart; utána megnyílik az emulátor ablaka, és elindul a programod. A beállítások alatti állapotsor ezt mutatja: **Fut, REST: 127.0.0.1:8064**.

6. Dolgozz tovább. A következő futtatások a nyitva hagyott emulátort használják. A beállításokban az **Emulátor leállítása** zárja be, és a Visual Assembler bezárása is; közben a flash-kép elmentődik, így a firmware saját beállításai megmaradnak.

Ha valami nem működik:

Üzenet	Teendő
Ultimate emulátor: nincs beállítva a Beállításokban: …	4. lépés: az üzenetben megnevezett elem üres, vagy a fájl már nincs a helyén.
Ultimate emulátor: hiányzik a ROM-mappából: chars.bin	A tároló-mappát vagy egy fölötte lévő kicsomagolt ZIP-mappát választottad: magát a `roms` mappát válaszd.
Az Ultimate emulátor REST API-ja nem válaszolt időben	Egy másik program használja a portot: állíts be másik **REST port**ot.
A **D64 az Ultimate emulátoron** nem tudja elkészíteni a lemezképet	Ez a mód a VICE `c1541`-ével csomagolja a D64-et: add meg a VICE útvonalát a **Beállítások** → **Fájlok és útvonalak** alatt. Maga az **Ultimate emulátor** mód nem igényel VICE-t.

Kung Fu Flash 2

Az assembleált programokat **valódi Commodore 64-en vagy C128-on** futtathatod — lemezmeghajtó, lemezkép, SD-kártyára másolás és emulátor nélkül. A Visual Assembler a cartridge USB soros portján keresztül kommunikál vele: újraindítja a MENU loaderbe, és feltölti a PRG-t, ami ezután elindul a gépen.

Mire van szükség:

- egy Commodore 64 / C128 **Kung Fu Flash 2** cartridge-dzsel, bekapcsolva;
- a cartridge-on **Kung Fu Flash firmware v2.07 vagy újabb** — a régebbi firmware-verziók nem támogatják a Visual Assembler által használt feltöltési protokollt. Firmware kiadások: https://codeberg.org/KimJorgensen/KungFuFlash2/releases/;
- egy **USB kábel**, amely a cartridge-ot a Visual Assemblert futtató számítógéphez köti (virtuális soros portként jelenik meg).

Beállítás	Leírás
Soros port	A cartridge USB soros portja, pl. `COM4` (Windows), `/dev/cu.usbmodem…` (macOS) vagy `/dev/ttyACM0` (Linux). Az USB azonosítója (`0483:5740`) alapján az app indításakor automatikusan megtalálja; a mező alatti státuszsor `KFF2: <port>` szöveget mutat, vagy kézi megadásra kér, ha nem találta. Az érték munkamenetek között megmarad.

Munkafolyamat:

1. Dugd a Kung Fu Flash 2-t a C64 cartridge-portjába, kapcsold be a gépet, és csatlakoztasd az USB kábelt a PC-hez vagy Machez.
2. Nyisd meg a **Beállítások** → **Integrációk** fület, és ellenőrizd a **Kung Fu Flash 2** szekciót — a portnak már ki kell lennie töltve.
3. Válaszd a **Kung Fu Flash 2**-t az osztott run menüből (a választás megmarad, és a projektfájlokba is bekerül).
4. Kattints a ▶ **Run**-ra. A program lefordul, a cartridge újraindul a MENU loaderbe (a soros port rövid, 1200 baudos „érintése"), rövid handshake következik, majd a PRG feltöltődik. Az átvitel végén a program a valódi gépen fut.

Blokk, **Expert** és **Ultimate Basic** módból egyaránt működik; az Ultimate Basic programok a projekt startup fájljából fordulnak.

> **Korlátok és ellenőrzések:** a PRG-nek tartalmaznia kell a kétbájtos betöltési címet és legalább egy adatbájtot, bele kell férnie a C64 64 KiB-os címtartományába, és nem lehet nagyobb, mint amit a cartridge loadere fogad. A hibákat — nincs beállítva port, a port nem nyitható meg, a cartridge nem tért vissza a MENU-ból, vagy a feltöltést elutasították — egy toast jelzi a pontos okkal.

> **Tipp:** ha a live feltöltés helyett cartridge-képet szeretnél, a **Build CRT** (Magic Desk 64K) is fut a Kung Fu Flash-en — lásd a 12b. szakaszt.

Verziókezelés (git)

Beállítások → **Általános** → **Verziókezelés** és **Beállítások** → **Fájlok és útvonalak** → **git**:

Beállítás	Leírás
Automatikus mentések a verziókezelésbe	A szerkesztés abbahagyása után kb. 2,5 mp-cel automatikus mentést tesz a verziótörténetbe (projekt fülénél a projekt történetébe is)
git.exe kiválasztása…	A `git` futtatható fájl kiválasztása (Windowson `git.exe`)
Automatikus keresés	Törli a választott útvonalat, és a PATH-ban, valamint a szokásos telepítési mappákban keres
Git használata, ha a projekt git repóban van	A git repóban lévő fájloknál git módban nyitja a Verziókezelés dialógust
Állapot	A git verzióját mutatja, vagy hogy nem található (a beépített verziókezelés ekkor is működik)

14. Vizuális szerkesztők (Toolkit)

A vizuális szerkesztők a saját eszköztár-gombjukkal nyílnak. Mindegyiknek van egy **Files** ▾ menüje az adat (többnyire nyers `.bin`) betöltésére, mentésére és a programba illesztésére (**Export to blocks**); a `.bin` `INCBIN` -nel kerül a programba. A Hi-Res szerkesztő közvetlenül D64 lemezre is ment (**Save to D64…**).

Színpaletta: a **Beállítások** → **Általános** → **Szerkesztők** → **C64 színpaletta** adja meg, milyen RGB értékekkel rajzolja minden szerkesztő a 16 C64 színt (sprite-, karakter-, Charset Canvas, Hi-Res, térkép- és pályaszerkesztő, Toolkit színminták). Az alapértelmezés a **Colodore**, ez áll a legközelebb egy kompozit / S-Video kijelzős valódi C64-hez; választható még a **Pepto** (a korábbi verziók sötétebb, kevésbé telített palettája), a **VICE**, a **Deekay** és a **C64 Wiki / Godot**. Csak a megjelenítés változik — az adatokban a színindexek ugyanazok maradnak. A Hi-Res képimport is a szerkesztő palettáját használja, ha ott nem választasz másikat.

Egyéni paletta: ha egyik sem illik a monitorodhoz vagy a valódi C64-edhez, válaszd az **Egyéni** palettát, és kattints az **Egyéni paletta szerkesztése…** gombra. Mind a 16 színhez színválasztó és hex mező (`#RRGGBB`) tartozik; a **Kiindulás** + **Átmásol** egy beépített palettával tölti ki mind a 16-ot. Az **Import…** Photoshop `.act`, VICE `.vpl`, GIMP `.gpl` fájlt vagy bármilyen szöveget olvas, amelyben 16 `#RRGGBB` / `$RRGGBB` érték van (az első 16 színt használja); az **Export .act** és az **Export .vpl** Photoshopnak vagy VICE-nak (VICE-ban *Settings* → *Video* → *Palette*) írja ki a palettát, így az emulátorban ugyanezeket a színeket vetheted össze. A **Mentés** eltárolja, és **Egyéni**-re vált.

A vizuális szerkesztő dialógusok a fejlécüknél húzva mozgathatók a teljes Visual Assembler munkaterületen. Egy mentett pozíció nélküli dialógus középen nyílik; mozgatás után az utolsó pozíciója a UI beállításokban tárolódik, és a következő megnyitáskor visszaáll.

Hi-Res / Multicolor Editor

Pixel-szintű bitmap szerkesztő 320×200 hi-res és 160×200 multicolor módokkal. A Hi-Res Editor eszköztár-gombjával nyílik.

Funkció	Leírás
Mode toggle	A **Multicolor** jelölőnégyzet vált hi-res (celláként mono) és multicolor (celláként 4 szín) közt.
Eszközök	Ceruza, radír, vonal, téglalap, kitöltött téglalap, ovális, kitöltött ovális, flood fill, színválasztó, cella háttérszín, kijelölés.
Színválasztó	A kattintott pixel látható színe lesz az ink (hi-resben a cella inkje vagy papírja, amelyiket a pixel mutatja); utána a szerkesztő visszavált az előző eszközre. **Alt+kattintás** bármelyik eszközzel színt vesz fel.
Cella háttérszín (hi-res)	Hi-resben minden 8×8-as cellának két színe lehet: az ink (beállított pixelek, a cella screen RAM bájtjának felső nibble-je) és a papír (törölt pixelek, alsó nibble). A **Cella háttérszín** eszköz a kattintott vagy áthúzott cellák papírját állítja a kiválasztott színre; a **PAPER** továbbra is az összes cellát egyszerre. A radír pixelt töröl, így a cella saját papírja látszik. Multicolorban le van tiltva, ott a háttér ($D021) közös.
Spray eszköz	Airbrush-stílusú festő, amely pixeleket szór a kurzor köré rajzolás közben.
Spray intenzitás	A spray eszköz melletti legördülő vezérli, mennyire sűrű minden spray húzás.
Színpaletta	Előtér (ink) + papír (háttér) választók. Multicolorban minden 4×8-as cellában a papír mellett 3 szín lehet: a 4. színt elutasítja, és a státuszsorban jelzi. A papír cseréje minden háttérpixelt átszínez, ahogy a `$D021` is. A korlátot túllépő cellák (pl. nyers, 32000 bájtos fájlból) piros keretet kapnak, és az export figyelmeztet rájuk, mert a többletszínek háttérszínűek lennének.
MIX (multicolor)	A **MIX** sor egyetlen szín helyett három szín ismétlődő mintázatát festi, így a cella négy színén túli árnyalatok is kikeverhetők. Kapcsold **Be**, és válassz a 8 hely közül (a helyre kattintás be is kapcsolja); minden helynek három színe (a színmintára kattintás a palettán kiválasztott színt veszi át) és egy mintája van: Sorok, Sávok, Tégla, Sakktábla, Lágy (4×4), Oszlopok, Átló, Árnyalat. A ceruzával, a spray-jel, a vonallal, a téglalappal, az oválissal és a kitöltéssel működik; a radírra és a színválasztóra nem hat. A minta a képhez rögzített, nem az alakzathoz, így az azonos keverékkel festett szomszédos alakzatok varrat nélkül találkoznak. A cellakorlát továbbra is érvényes: a papír színét tartalmazó keverék a cella három színéből csak kettőt foglal. A helyek a munkamenetek között megmaradnak. Hi-resben le van tiltva.

Funkció	Leírás
Undo / Redo	Húzásonkénti előzmény, ctrl-Z / ctrl-Y.
Kijelölés	A szaggatott négyzet ikonú eszköz: húzással jelölhető ki egy téglalap alakú terület a vásznon. Az **Esc** megszünteti a kijelölést.
Másolás / beillesztés	A **Copy** (`Ctrl/Cmd+C`) a kijelölt területet másolja. A **Paste** (`Ctrl/Cmd+V`) után a másolat előnézete követi a kurzort: kattintás a vászonra elhelyezi, az **Esc** megszakítja. A másolat megmarad, így újabb `Ctrl/Cmd+V` és kattintás akárhányszor beilleszti. Hi-res módban a beillesztett pixelekre is érvényes a cellánkénti két szín, ezért a színek megtartásához igazítsd a területet a cellarácshoz. A Copy, a Paste és az áttetsző beillesztés a vászon jobb klikkes menüjében is megvan; az onnan indított beillesztés előnézete a kattintás helyén jelenik meg.
Áttetsző beillesztés	A `Ctrl/Cmd+Shift+V` (vagy az eszköztár gombja) kihagyja a másolat papír színű pixeleit, így az alatta lévő kép megmarad.
Grid + Raster	A **Grid** a karaktercellák rácsát mutatja: hi-resben 8×8 pixel, multicolorban 4×8 dupla széles pixel. 4×-es nagyítástól halványabb vonallal minden egyes pixelt is körberajzol, így a pixelek pontosan elhelyezhetők. A **Raster** raszter-sor overlay-t ad.
Zoom	Csúszka 1×-től 16×-ig, vagy Ctrl+görgő a vásznon (a kurzor alatti pont a helyén marad).
Kép importja	A canvas-ra dobott PNG/JPEG/GIF automatikusan a 16 színes C64 palettára kvantálódik.
Export blokkok	Betöltőt fűz a programhoz: a `HIRES` fájlt lemezről a `$2000` -re tölti, a screen RAM-ot a `$0400` -ra (multicolorban a color RAM-ot a `$D800` -ra) másolja, majd bekapcsolja a bitmap módot. A kép nincs a blokkokban: a **Save to D64** teszi a lemezre.
Export `.bin`	A natív multicolor formátumot menti (10000 byte: 8000 bitmap + 1000 screen + 1000 color), készen a $2000-re való LOADFILE-hoz.

Sprite Editor

24×21 pixeles sprite szerkesztő több-frame-es animációval. A Sprite Editor eszköztár-gombjával nyílik.

Funkció	Leírás
Frame-ek	Frame-ek hozzáadása / eltávolítása / átrendezése; a frame-csík alul látszik. A frame-sor **Copy** / **Paste** gombja a teljes aktuális frame-et másolja; ugyanez a két pont a vászon és a frame-csík jobb klikkes menüjében is megvan.
Mode	Mono / multicolor kapcsoló.
Eszközök	Ceruza, kitöltés, színválasztó, vonal, téglalap, kör — plusz vízszintes tükrözés, függőleges tükrözés, balra/jobbra/fel/le tolás (opcionális körbefordulással). A shape eszközök élő előnézetet mutatnak húzás közben; elengedésre véglegesít.
Színválasztó	A kattintott pixel színhelyét választja ki (sprite szín, MC1, MC2, háttér vagy overlay szín, a szerkesztett rétegen); utána a szerkesztő visszavált az előző eszközre. **Alt+kattintás** bármelyik eszközzel.
Zoom	Csúszka 3×-tól 40×-ig, vagy Ctrl+görgő a vásznon.
Undo / redo	Teljes undo/redo verem frame-enként. Ctrl/Cmd+Z / Ctrl/Cmd+Y vagy az eszköztár gombjai.
Kép importja	PNG vagy JPEG importja a Files → Import image menüvel. Az importáló minden pixelt a legközelebbi C64 palettaszínre képez, és az aktuális frame-be írja.
Animáció előnézet	Play / Stop konfigurálható sebességgel.
Export blokkok	Egy RAWBYTES blokkot szúr be 64 byte-ra igazított címre minden frame-hez, plusz egy sprite pointer beállítást.

Funkció	Leírás
Save `.bin`	Frame-enként 64 byte-ot ír (nyers sprite adat kitöltés nélkül).
Kijelölés	**Kijelölés / mozgatás**: húzz ki egy téglalapot, majd belül húzva mozgasd a pixeleket (az üres pixelek átlátszók); a nyilak egy pixellel (multicolorban kettővel) mozgatják, a Delete törli, az Esc elengedi. A **Forgatás 90°** (mindkét irányba) a kijelölést vagy a teljes frame-et fordítja. A kijelölés, tükrözés, tolás és forgatás a testre és a hires rétegre együtt hat.
Hires réteg	A **Hires réteg** minden frame fölé egy második, egyszínű hires sprite-ot tesz — a klasszikus körvonal a multicolor test fölött. A **Réteg** lista választja a szerkesztett réteget; a test szerkesztésekor a réteg halványan látszik. A C64-en külön sprite: kapjon kisebb sorszámot, mint a test. A `.bin` és a **Copy bytes** a test sprite-jai után írja (`sprite_f1_ov_1`).
Tükrözött animáció	Minden frame-ről (és hitboxáról) vízszintesen tükrözött másolatot fűz a végére — egy kattintással a másik irányú járásciklus.
Sprite sheet import	Files → **Sprite sheet import…**: a cella szélessége / magassága, eltolás, rés és frame-szám alapján a képet az aktuális méretű frame-ekre vágja (multicolorban a cella spritonként 12 pixelre skálázódik). A színek a képből jönnek, vagy a sprite sajátjai maradnak; OKLab egyeztetés dithering nélkül, rendezett (Bayer) vagy Floyd–Steinberg ditheringgel. Lecseréli a frame-eket, vagy hozzáfűzi.
Hitbox	A **Hitbox** eszköz frame-enként egy téglalapot húz (szaggatott cián); az oldalsávban **Illesztés**, **Illesztés minden frame-en**, **Másolás minden frame-re** és **Törlés**. A Files → **Hitbox tábla másolása** `sprite_hit_x / _y / _w / _h` táblát ad (frame-enként egy bájt, képernyőpixel a bal felső saroktól); hitbox nélküli frame a pixelei befoglalóját kapja.
X / Y ×2	Az előnézetet a `$D01D` / `$D017` sprite-nagyítással mutatja.

Export to UltimateBasic (Files menü): minden frame-et (legfeljebb 8-at) `sprdef` -ként ír ki, beállítja a 0. sprite-ot (szín, multicolor, nagyítás), animációnál pedig raszterhez szinkronizált ciklust ad hozzá, amely a szerkesztő FPS-évl lépteti a frame-eket. Ultimate Basic módban a kurzorhoz szúrja be, egyébként a vágólapra másolja.

C64 karakter-ROM böngésző („Char map")

A C64 karakter-ROM (a beépített PETSCII font) csak olvasható nézője. A **C64 chargen** eszköztár-gombbal nyílik. Hasznos egy glif screen code-jának megtalálásához, mielőtt `RAWBYTES` -szal vagy `TEXT` -tel kiírnád.

Funkció	Leírás
Két karakterkészlet	1. fül: **Set 1 — Upper/Graphics** (bekapcsolás utáni alap). 2. fül: **Set 2 — Lower/Upper** (a `$0E` váltás után).
Glif-rács	16×16-os rács mind a 256 karakterrel. Kattints egy glifre a részlet-paneljéhez: nagyított 8×8 pixel nézet, screen code (decimális + hex), PETSCII kódok (alap és shiftelt is), és a nyers 8 byte-os bitmap.
Részlet-panel	Mutatja a kiválasztott glif screen code-ját, PETSCII kódjait és a nyolc nyers byte-ot — készen a `RAWBYTES` vagy BYTE blokkba illesztéshez.
Csak olvasható	Itt nincs szerkesztés — a glifek módosításához használd a Character Editort (lentebb).

Character Editor (Charset)

256 karakteres 8×8 charset szerkesztő. A Character Editor eszköztár-gombjával nyílik.

Funkció	Leírás
Load ROM	A C64 karakter-ROM-ot tölti be — a pontos 901225-01 glifákkal, fájlválasztó nélkül. Az első gomb (és **Files** → **ROM: nagybetűs / grafikus**) az 1. készletet, a második gomb (vagy Shift+kattintás, vagy **Files** → **ROM: kisbetűs / nagybetűs**) a 2. készletet tölti be.
Load `.bin`	Külső 2048 byte-os charset binárist importál.
Karakter-előnézet	16 széles rács mind a 256 glif-fel, az aktuális cella kiemelve.
Pixel szerkesztő	A kijelölt karaktert, vagy 2×1, 1×2, 2×2 szomszédos karakterből álló **csempét** szerkeszt egy képként (Csempe választó).
Rajzeszközök	Ceruza, vonal, téglalap, kitöltött téglalap, kitöltés és színválasztó. Ha a pixel már a kiválasztott színű, a kattintás törli. A színválasztó (csak multicolorban; **Alt+kattintás** bármelyik eszközzel) a kattintott pixel színét — FG, BG, MC1 vagy MC2 — teszi a tollá.
Átalakítások	Törlés, invertálás, vízszintes / függőleges tükrözés, körbeforduló eltolás, 90°-os forgatás jobbra / balra (csak négyzetes csempénél) — az egész csempén.
Visszavonás / újra	Toolbar gombok, Ctrl+Z és Ctrl+Y (Ctrl+Shift+Z).
Másolás / beillesztés	Toolbar gombok, Ctrl+C / Ctrl+V, vagy jobb klikk a térképen; a teljes csempét másolja a színeivel.
Húzás a térképen	Egy karaktert a térképen egy másikra húzva felcseréli őket; Shift+húzás másol.
Szöveges előnézet	A beírt szöveg az aktuális charsettel jelenik meg. Választható a nagybetűs vagy kisbetűs készlet szerinti leképezés; a `{65}` vagy `{$41}` képernyőkódot szúr be.
Tartomány-export	Add meg a Tól / Ig karaktert, majd **Files** → **Tartomány mentése (.bin)** (nyers bájtok) vagy **Tartomány másolása ASM-ként** (`.byte` sorok).
Karakterenkénti szín	Alapértelmezett Color RAM értéket tárol minden glifhez. Multicolor karakter módban ez a cellákénti multicolor engedély bitet és a karakter saját alsó-3-bites színét is megőrzi.
Metaadat round-trip	Charset Canvas / Map munkafolyamatokból kompatibilis adat betöltésekor a karakterenkénti szín-metaadat megőrződik, így a szerkesztések folytathatók a szín-szándék elvesztése nélkül.
Export blokkok	RAWBYTES-t fűz a $0800-ra (2. blokk) vagy a $3800-ra (7. blokk) a kódolt charsettel.

Charset Canvas Editor

Teljes-canvas charset festő képernyők építéséhez egy teljes 256 karakteres charsetből. A Charset Canvas eszköztár-gombjával nyílik.

A canvas 16×16 karakter, 128×128 pixel. Multicolor karakter módban minden karakter vagy multicolor (széles pixelek, 4 szín), vagy hires (8 keskeny pixel, 2 szín), pontosan úgy, mint a C64-en.

| Funkció | Leírás | |---|---| | Mono / multicolor | A mono mód 1 bites 8×8 glifeket tárol. A multicolor mód kétbites vízszintes pixel-párokat tárol, és a multicolor cellákat Color RAM 3. bittel jelöli. | | Karaktermód eszköz | Multicolor módban a karakterekre kattintva vagy rajtuk húzva váltasz multicolor és hires között (Color RAM 3. bit). Az alak megmarad: a beállított pixelpár a karakter saját színe lesz. A hires karakter színei a háttér és a saját szín (0–7); MC1 / MC2 ott nem fest. | | Háttér | A BG legördülő mono módban is látszik; az MC1 / MC2 csak multicolorban. | | ECM | Az **ECM** jelölőnégyzet (kiterjesztett színmód, multicolorral együtt nem megy): minden karakter hires, saját színnel (0–15) és a négy háttér egyikével, BG0–BG3 (`$D021` – `$D024` ; a BG1 / BG2 az MC1 / MC2 regiszter). Válassz BG színt a palettán (B0–B3 jelölés), majd a **Karakter háttere** eszközzel kattints vagy húzd a karaktereken. Csak 64 különböző karakter fér el: az

állapotsor számolja, mentéskor az azonos karakterek összevonódnak — a charset 64 karakteres (512 bájt), a képernyőkód `karakter | háttér << 6`. Bekapcsolás a C64-en: `$D011` 6. bit. | | C64 színmodell | A háttér a `$D021` -et használja; a megosztott multicolor 1 a `$D022` -t; a megosztott multicolor 2 a `$D023` -at; minden karakter saját színe a Color RAM 0–2. bitjéből jön. | | Rajzeszközök | Ceruza, radír, vonal, téglalap, ovális és flood fill karakterhatárokon át működik. | | Színválasztó | A kattintott pixel látható színét választja ki (multicolor karakterben a BG / MC1 / MC2 / saját szín tollat is); utána a szerkesztő visszavált az előző eszközre. **Alt+kattintás** bármelyik eszközzel. | | Spray eszköz | Airbrush-stílusú rajzolás, amely pixeleket szór a szomszédos cellák/karakterek közt. | | Spray intenzitás | A spray eszköz melletti legördülő beállítja a húzás sűrűségét. | | Grid kapcsoló | A Grid jelölőnégyzet megjeleníti vagy elrejti a 16×16 karakter-rácsot. | | Nagyítás | A nagyítás-csúszka (2×–24×, egész eszközpixel C64-pixelenként) vagy Ctrl+görgő a vásznon; a nagyítás megmarad. 8×-tól a Grid minden pixelhatárra halvány vonalat is rajzol. | | Save charset `.bin` | A 2048 byte-os karakter-bitmap adatot menti. | | Save 16×16 map + Color RAM `.bin` | 256 screen code-ot, majd 256 Color RAM értéket ment. Ezt a `MAP_COPY16X16` -tal használd. | | Betöltés | Betölthet charset-canvas mentéseket, sima charset adatot és kompatibilis Character Editor charset adatot, beleértve a tárolt karakterenkénti színeket, ha jelen vannak. |

Fontos C64 korlát: multicolor karakter módban a két megosztott szín globális az egész képernyőre (`$D022` / `$D023`). Csak a karakter saját színe cellánkénti, és az a 0–7 színekre korlátozódik, mert a Color RAM 3. bit választja a multicolor módot. Így egy karakterben legfeljebb **4 szín** lehet (háttér, MC1, MC2 és a saját szín), az egész képernyőn legfeljebb 3 + 8. A palettán az osztott színek BG / 1 / 2 jelölést kapnak, a nem osztott 8–15 színek halványak. Ilyen szín választásakor az MC2-be vagy MC1-be kerül, de csak amíg azt a regisztert egy pixel sem használja; különben a legördülőben állítsd át (ez az egész képernyőt átszínezi). Hires (mono) módban egy karakter 2 színű: háttér és a saját szín (0–15).

Map Editor (több-rétegű tilemap-ek)

Réteges tilemap szerkesztő statikus díszlethez, sprite spawn map-ekhez, ütközési adathoz és hasonlókhoz. A Map Editor eszköztár-gombjával nyílik.

Funkció	Leírás
Rétegek	Több nevesített réteg, mindegyik saját tileset-tel és átlátszatlansággal.
Ecsetek	Egy-tile, kitöltés, vonal, téglalap és kör módok. A shape eszközök élő előnézetet mutatnak húzás közben; elengedésre véglegesít.
Zoom	Csúszka, vagy Ctrl+görgő a térképen.
Undo / redo	Teljes undo/redo verem rétegenként. Ctrl/Cmd+Z / Ctrl/Cmd+Y vagy az eszköztár gombjai.
Clear menü	Rétegenkénti vagy teljes-map törlés megerősítéssel.
Kép importja	Dobj be egy tilemap PNG-t; a szerkesztő automatikusan tile-okra szeleteli.
Copy / paste	Másolj egy kijelölt tile-tartományt, majd illeszd be normálisan, vagy használj átlátszó beillesztést, hogy az üres tile-ok átlátszók maradjanak. Ugyanezek a pontok a vászon jobb klikkes menüjében is megvannak.
Multicolor-tudatos tile-színezés	Kompatibilis charset metaadattal a festés a tile tárolt Color RAM alapértékeit használja (a multicolor-hoz kapcsolódó kódolással) egy általános lapos szín helyett.
Egyéni charset színek	Ha egy charset `charColors` metaadatot hordoz, a Map Editor a kiválasztott tile alapértelmezett Color RAM értékét használja festés közben.
Export blokkok	RAWBYTES blokkokat bocsát ki a tileset grafikához + map adathoz.
Save Screen RAM (.bin)…	Csak az aktuális map réteg screen code-jait menti (40×25 = 1000 byte).
Save Screen RAM + Color RAM (.bin)…	Screen code-okat a Color RAM értékekkel összefűzve ment egyetlen 2000 byte-os fájlba (`screen[0..999]` , majd `color[0..999]`). Ezt a **MAP_COPY** makróval használd (`auto` color forrással)

Funkció	Leírás
	mindkettő egy művelettel való visszaállításához futásidőben.

Pálya szerkesztő (Level Editor)

Map Editor képernyőket rendez egyetlen pályába — függőleges lövöldözőshöz, lefelé görgő autós játékhoz, oldalra görgetőshöz vagy 2D térképhez —, és sima görgetéssel végig mutatja a teljes pályát. A Map Editor melletti **Pálya szerkesztő** toolbar gombbal nyílik.

Funkció	Leírás
Elrendezés	**Függőleges**, **Vízszintes** vagy **Rács** (sorok × oszlopok, sor/oszlop hozzáadása és törlése gombokkal).
Képernyők	**Új üres képernyő**, **Hozzáadás a Map Editorból** (az éppen nyitott map) vagy **Map(ok) betöltése (.bin)…** (több Map Editor export egyszerre). Húzz egy képernyőt a listából egy helyre; a helyeket egymásra húzva felcserélheted.
Szerkesztés	A **Szerkesztés a Map Editorban** megnyitja a kijelölt képernyőt a Map Editorban, és visszahozza a változásokat; duplikálás, mozgatás fel/balra vagy le/jobbra, eltávolítás; visszavonás / újra.
Előnézet	Az előnézetet húzva vagy a nyilakkal pixelenként görgethetsz; a **Lejátszás** automatikusan görget (irány, sebesség 1–8, ismétlés), Szóköz = lejátszás / leállítás. A *Határok* jelöli a képernyőhatárokat.
Zoom	Az áttekintő nagyítás-csúszkája (25–400 %), vagy Ctrl+görgő az áttekintőn.
Ellenőrzések	Karakterkészlet-infó, és figyelmeztetés, ha a képernyők eltérő karakterkészletet vagy színeket használnak (a pálya az első képernyő beállításait használja).
Karakter-attribútumok	A **Karakter-attribútumok** gomb az áttekintőt attribútum-festésre váltja: válaszd ki a **Szilárd**, **Létra**, **Veszélyes** vagy **Gyűjthető** jelzőt, kattints egy karakterre, és az adott screen code minden cellája megkapja (vagy elveszíti); húzással több kódra is. Színes rétegként látszanak, és a 256 bájtos `LEVEL_ATTR` táblába kerülnek (`LEVEL_ATTR_SOLID` = 1, `_LADDER` = 2, `_HAZARD` = 4, `_COLLECT` = 8).
Él-ellenőrzés	Pirossal jelöli, ahol a szomszédos képernyők nem folytatódnak: a határ egyik oldala üres (szóköz), a másik nem, vagy csak az egyik oldal szilárd. Az oldalsáv megszámolja az éleket és cellákat.
Színek	**Pálya kerete**, és a kijelölt képernyőhöz **Saját színek** (háttér, keret, multicolor 1 / 2); az előnézet kerete mutatja a keretszínt. Helyenként `LEVEL_SCR_BG / _BORDER / _MC1 / _MC2` táblaként exportálódik (`LEVEL_SCR_COLORS` = 1), ha egy elhelyezett képernyőnek saját színei vannak.
Memória	A `.level` adat mérete egy választott címen (pontosan úgy számolva, ahogy a makró építi), figyelmeztetéssel a BASIC / KERNAL ROM, az I/O terület és a `$0800` alatti memória esetén, plusz a pálya RLE-mérete. Az **RLE exportálása blokkokba / RLE asm másolása** a pályát RLE-tömörítve adja a `level_unrle` kicsomagolóval (`($FB)` → `($FD)`).
Futtatás VICE-ban	A pályából joystickkal görgethető tesztprogramot épít (a `level-demo` scrollere, kód `$C000` -n, pályaadat `$2000` -tól, egyedi karakterkészlet `$2000` -n), és elindítja VICE-ban — a színek a nézet bal felső képernyőjével váltanak, a kamera az első Kezdőpont objektumnál indul.
Objektumok	Az **Objektumréteg** gomb az áttekintőt objektumszerkesztésre váltja: válassz típust (Kezdőpont, Ellenfél, Tárgy, Ajtó, Kijárat, Esemény) és **Param** bájtot (ellenfélfajta, tárgy-azonosító, ajtó célja …), kattintással lerakod, húzással mozgatod (8 px-es rácsra, Shift = szabadon), egy objektumra kattintva kijelölöd és módosíthatod a típusát / paraméterét, a nyilak mozgatják, a Delete törli. Az objektumok a pályapozícióhoz tartoznak, nem egy képernyőhöz. Az oldalsáv listázza, az előnézet mutatja őket, a pályával együtt mentődnek, és görgetési sorrendben (függőlegesen Y, vízszintesen X, rácsban soronként) `LEVEL_OBJ_*` táblaként exportálódnak — az **Asm másolása / Exportálás blokkokba** és a `.level` makró is.

Files menü:

Művelet	Mit csinál
Új pálya / **Pálya megnyitása (.bin)…** / **Pálya mentése (.bin)…**	A pályaprojekt: minden képernyő és az elrendezés. Ezt a fájlt olvassa a `.level` makró és az Ultimate Basic `map load` -ja.
Pálya Screen RAM mentése (.bin)…	A teljes pálya egyetlen karaktertérképként (csak screen code-ok).
Pálya Screen + Color RAM mentése (.bin)…	A teljes pálya screen code-jai, utánuk a színei.
Exportálás blokkokba / **Asm másolása**	A pályaadat blokkokként vagy asm szövegként (vágólapra).

A pálya használata: assemblerben `.level "level.bin"` + `.level_row` (lásd LEVEL / LEVEL_ROW és `samples/level-demo.asm`); Ultimate Basic 1.6.1+ alatt `map load "level.bin", $4000` és `map view px, py` (minta: `level_scroll_demo.ub`).

SID Editor (3-voice tracker)

Több-hangszeres 3-voice tracker. A SID Editor eszköztár-gombjával nyílik. A lejátszás és minden előnézet (**Play: All**, hangszer-előnézet, virtuális billentyűzet, hangbeírás, akkord / arpeggio, sor- és egy patternes lejátszás) az emulált SID-en szól: az exportált 6502 lejátszó frame-pontos JS modellje a backendnek küldi a SID-regisztereit, a hangot az rendereli. Az asztali appon kívül egy kis SID-szintetizátor szól helyette.

Előnézeti motor: a **Beállítások** → **Általános** → **SID előnézet** alatt a **reSID (resid-rs)**, ez az alapértelmezett, vagy a **VA SID Core (Rust)**, az alkalmazás saját SID-emulációja választható, és a chip (MOS 6581 / 8580). A szerkesztő fejléce mutatja a használt motort (pl. · *reSID (MOS 8580)*). A hangszer-paraméterek tekerőgombokkal állíthatók.

Két SID: a tracker eszköztárán a **SID: 1 / 2** 3 és 6 hang között váltja a dalt. Két SID-nél a mellette lévő lista a második SID címét állítja: `$D420` – `$D7E0` vagy `$DE00` – `$DFE0`, `$20` -as lépésekben. Az oszlopok fejléce **S1 V1** … **S2 V3**. Mindkét SID-nek saját szűrője és hangereje van, és egy hang csak a saját chipjéét vezérli. A két SID sztereóban szól (SID 1 balra, SID 2 jobbra). A hang-előnézetek az első SID-en szólnak. Egy SID-re visszaváltva a második SID hangjai törlődnek, és ha van bennük valami, előbb rákérdez.

Hangszerenkénti vezérlők:

- Hullámforma jelölőnégyzetek (TRI / SAW / PUL / NOI) — több hullámforma a SID-hez hasonlóan ÉS-elődik —, plusz **RING** (a háromszög gyűrűmodulációja az előző hanggal: 1 ← 3, 2 ← 1, 3 ← 2) és **SYNC** (hard sync az előző hanghoz).
- ADSR (attack / decay / sustain / release) drag-grafikonként a négy csúszka fölött.
- Pulzusszélesség csúszka (0-4095).
- **Szűrő:** *A hang menjen át a szűrőn*, LP / BP / HP és **3 OFF** (a 3. hang némítása, ha nem megy át a szűrőn), cutoff, rezonancia, hangerő. A **Vége** + **Söprés** a cutoffot a Cut értéktől a Vége felé viszi frame-enként Söprés lépéssel; az **Oda-vissza** a kettő között pattog — ebből lesz a rezonáló szűrő-söprés. A szűrőt az a hang birtokolja, amelyik szűrőzött hangszert indított (cutoff, rezonancia, mód); minden hang átmegy rajta, amelyiknek a hangszere szűrőzött.
- **Vibrato: Késleltetés** frame után háromszög LFO, ami **Sebesség** frame-enként fordul, a **Mélység** 0-15 a hang félhangjához skálázva (független a `V` effekttől).
- **Pulzusszélesség-moduláció:** frame-enként **Sebesség** adódik a pulzusszélességhez, **Min** és **Max** között pattogva.
- **Gate:** a **Gate időzítő** ennyi frame-mel a hang következő hangja előtt elengedi a gate-et (legyen kisebb a Speednél); a **Hard restart** ekkor az ADSR-t is nullázza a tiszta indításhoz; az **Első frame** a hang első frame-jén hullámformát ír (pl. `$09` test + gate).

- **Wave / arpeggio tábla:** legfeljebb 32 lépés hullámformával (— = marad) és hanggal (eltolás félhangban, vagy **abs** = abszolút hang), lépésenként **Sebesség** frame, **Ismétlés** egy lépésre vissza, vagy megáll az utolsón. **Előbeállítások** (25, csoportosítva; a lista mutatja a választottat, amíg a táblát kézzel nem szerkeszted): arpeggiók (dúr, moll, szeptimakkordok, sus4, szűkített, bővített, power akkord, oktáv, trilla), dobok (lábdob, pergő, tam, taps, zárt / nyitott lábcin), lead- és basszus-indítások, hangeffektek (lézer, felvétel, robbanás).

Tracker rács:

- 3 voice (második SID-del 6) × akár 256 pattern × 32 sor; a song / sorrend lista legfeljebb 255 lépés.
- Soronként: note + hangszer-index, plusz egy opcionális effekt (lásd lentebb az **Effekt-oszlop**-ot). Az üres sorok az előző note-ot tartják: más trackerekhez hasonlóan (GoatTracker, SID-Wizard) a hang gate-je nyitva marad (sustain szint) a voice következő hangjáig vagy egy **note-off**-ig.
- Az **Enter** (vagy dupla kattintás) szövegként szerkeszti a kijelölt cellát (`C-4 01 V24`); az **Enter** ment és a következő sorra lép, az **Esc** elvet.
- **Note-off** (`===`): **1** billentyű (vagy `` ` `` / `=`), vagy `===` / `OFF` a cellába írva. Elengedi a gate-et, így lefut a hangszer release-e — ugyanaz, mint a hang nélküli `C` effekt (`C00` -ként mentődik és exportálódik).
- Válassz egy cellát normálisan, vagy tartsd a **Shift**-et kattintás vagy nyílbillentyű közben egy téglalap-kijelölés kiterjesztéséhez sorokon és bármelyik voice-on át. A kijelölt területen belüli jobbklikk megőrzi a tartományt.
- A Copy, Cut, Paste és Clear elérhető az ikon-eszköztárból és az ikon-alapú context menüből. A `Ctrl/Cmd+C` és `Ctrl/Cmd+V` ugyanazon a téglalap-kijelölésen működik.
- Harmónia segéd: válassz alaphangot, akkord-típust és oktávot, hallgasd meg az akkordot az aktuális hangszerrel, majd szúrd be a hangzatot közvetlenül a trackerbe. Az elérhető típusok: Major, Minor, Diminished, Augmented, Sus2, Sus4, Dominant 7, Major 7, Minor 7, 6, Minor 6, 9, b9, #9, Dim7, 7sus4.
- Arpeggio segéd: hallgasd meg vagy szúrd be a kiválasztott akkordból 4-, 8- vagy 16 lépéses note-futamokat fel, le vagy fel/le irányban.
- A **Preview row** meghallgatja a kiválasztott sort az összes voice-on a pattern-lejátszás indítása nélkül.
- A cella-tartomány beillesztés mostantól a kijelölt tartomány kezdőcellájánál kezdődik, és tisztán megáll a voice- és sor-határoknál, ahelyett hogy a következő oszlopba vagy sorba fordulna.
- Hangszerek törlése: a **+** melletti **−** törli az aktuális hangszert (a hangjai kérdés után az előtte lévőt játsszák), a Files → **Nem használt hangszerek törlése** pedig minden olyat, amelyet egy hang sem játszik. A tracker számai követik; a Visszavonás visszahozza őket.
- A dal hangereje: a tracker eszköztárának **Hangerő** tekerője (0–15) a dal hangereje, vagyis a SID fő hangereje mindhárom hangra. Két SID-es dalnál chipenként van egy tekerő (**SID 1**, **SID 2**). A hangszereknek nincs saját hangerejük (egy hang szintjét a sustain adja), a korábban mentett dal az első hangszere hangerejét kapja.
- PAL / NTSC: a BPM melletti kapcsoló adja meg, melyik gépre szól a dal. NTSC-n másodpercenként 60 frame fut az NTSC órajelén: ehhez igazodik az előnézet, a BPM, az exportált lejátszó hangjegytáblája és a `.sid` fejléce.
- **Kvantálás** (a tracker eszköztárán): a rács 2, 4, 8 vagy 16 sor. A vízszintes trackerben a **Q** vagy a gombja a kijelölt hangokat a legközelebbi rácssorra teszi, a hosszuk megmarad; bekapcsolt **Felvételnél** mellett a felvett hangok a felvétel végén ugyanígy kvantálódnak.
- **Kijelölés a vízszintes trackerben:** Shift+húzás: keret (Ctrl+Shift+húzás: hozzáadás a kijelöléshez), Ctrl+kattintás egy hangon, Ctrl+A: mind, Esc: törli a kijelölést. A Ctrl+X / C / V és a Del, vagy a fejléc gombjai kivágnak, másolnak, beillesztenek és törölnek. A beillesztés az egér alatti sorra kerül, egyébként a tracker kurzorához, és felülírja, ami ott van. Egy hang törlése után az előtte lévő hang megtartja a végét. A fejlécben **Visszavonás** és **Mégis** gomb is van.
- A **Visszavonás / Mégis** (a hang másolása előtti gombok, vagy Ctrl+Z és Ctrl+Y) végiglép a hangok, hangszerek, a song és a tempó változásain és a dal betöltésén, legfeljebb 100 lépésig. Egy tekerőgomb húzása vagy egy teljes felvétel egy lépés, a nézet a megváltozott hangszerre és patternre ugrik. A szövegmezők megtartják a saját Ctrl+Z-jüket.

- A **Frame/sor** (1–255, tehát a nagyobb szám lassabb) beírható, vagy a mellette lévő − / + gombokkal léptethető; az `F` effekt lejátszás közben változtatja. A mellette lévő **BPM** az ebből adódó tempót mutatja: ütésenként 4 sorral, PAL-on 6 frame/sor = 125 BPM.

Song / sorrend lista:

- A patternek elrendezése egy song/sorrend listán keresztül történik a nyers pattern-bank fölött — ugyanaz a pattern ismétlődhet, és a dal hosszabb is lehet, mint a tárolt patternek száma.
- Lépések hozzáadása, eltávolítása és átrendezése a song-lista panelről; az aktuális lépés lejátszás közben kiemelve jelenik meg.
- Mind az app-on belüli lejátszás, mind minden export ezt a listát járja be, nem a nyers pattern-sorrendet — a sorrend-lista bevezetése előtt készült exportok továbbra is helyesen töltődnek be, egyszerű 0..N-1 dalként.

Effekt-oszlop: Minden cella a note és hangszer mellett hordozhat egy effektet is, rövid kóddal megadva közvetlenül a note után (pl. `C-4 01 V24`), vagy önmagában, note nélküli effekt-sorként (pl. `F06`):

Kód	Effekt	Érték
`V`	Vibrato	alsó nibble = mélység, felső nibble = tartási hossz
`U`	Csúszás fel (portamento)	soronként a frekvenciához adott mennyiség
`D`	Csúszás le (portamento)	soronként a frekvenciából levont mennyiség
`C`	Note-cut	elnémítja a voice-ot újraütés nélkül; az érték nem használt
`F`	Tempóváltás	új frame/sor érték, azonnal érvénybe lép

Az effektek soronként egyszer frissülnek (megegyezően az exportált lejátszó időzítésével), azonosan futnak a Web Audio előnézetben és a lefordított 6502 exportban, és veszteségmentesen mentődnek/töltődnek be a verziózott `.bin` formátumban.

Lejátszás és virtuális billentyűzet:

- A Play eszköztár-gomb lejátszás közben Pause-ra, szüneteltetve Resume-ra vált; a Stop leállítja a lejátszást és visszaállítja az állapotot.
- A tracker eszköztár végén lévő **LCD** lejátszás közben a song-pozíciót (POS), a patternt (PAT), a sort (ROW) és a lejátszási időt (TIME) mutatja, leállítva a szerkesztési pozíciót. A **Beállítások → Általános → SID előnézet → LCD színe** zöldre vagy pirosra állítja.
- A tracker eszköztárán a **Csatorna-spektrum** gomb egy ablakot nyit, amelyben hangonként egy 16 sávos analizátor látszik (két SID-nél 6). A lejátszott SID-regiszterekből számolódik — frekvencia, hullámforma, pulzusszélesség, közelített ADSR, szűrő és hangerő —, nem a hangból, mert a kimenet egyetlen keverék. A gyűrűmoduláció, a sync és a szűrő rezonanciája nem látszik, és csak tracker-lejátszás közben mozog, hang-előnézetnél nem.
- A billentyűzet eszköztár-gomb a **VA-VK**-t, egy nem-modális zongorát nyit (2, 3 vagy 4 oktáv; a helyét, a nyitott állapotát, az oktávot és az *Insert into tracker*-t megjegyzi) saját **Play / Stop** gombokkal, amely használható marad, míg a SID szerkesztő aktív. Húzd a fejlécét a fő alkalmazás fölé bárhova.
- A VA-VK **Hangszer** tekerőgombja (húzás, görgő vagy nyílbillentyűk) választja ki a hangszert, a mellette lévő LCD a számát és a nevét mutatja; a szerkesztő hangszerlistáját követi. A VA-VK címe melletti jelvény a használt SID-emulációt mutatja (reSID vagy VA SID Core, a chip-modellel).
- Engedélyezd az **Insert into tracker**-t, hogy minden lejátszott note az aktuális tracker-kurzorra íródjon, és a következő sorra lépjen. Kapcsold ki a note-ok meghallgatásához szerkesztés nélkül.

- Az akkord- és arpeggio-előnézetek megvilágítják a megfelelő zongorabillentyűket, amikor a billentyűzet nyitva van.

Felvétel, metronóm és MIDI:

- A **Record** (piros pont, a tracker eszköztárán és a VA-VK-n) ütemre beírja a játszott hangokat — VA-VK-ról, a számítógép billentyűzetéről vagy MIDI billentyűzetről — a kijelölt voice-ba, a kijelölt sortól, az aktuális tempó szerint. Ha a pattern betelt, az aktuális song-lépés után új, üres pattern jön, és ott folytatódik a felvétel. A Record újbóli megnyomása leállítja.
- A **Metronóm** (tracker eszköztár és VA-VK) 4 soronként kattan, a 4 ütemes taktus első ütésén hangsúlyosan. A mellette lévő ütemlámpák lejátszás és felvétel közben minden ütésre felvillannak, kikapcsolt kattanással is.
- **MIDI:** a hangszernév melletti **MIDI** listában válaszd ki a bemenetet, majd nyomd meg a **Connect MIDI** gombot; a hangok az aktuális hangszerrel szólnak (felvétel közben vagy *Insert into tracker* mellett a trackerbe is beíródnak). A gomb melletti lámpa minden beérkező MIDI üzenetre felvillan. Bekapcsolt **Polifon felvétel** mellett a felvétel alatt lenyomva tartott billentyűk külön hangra kerülnek a kijelölt hangtól kezdve, a felengedés note-offot ír; ha minden hang foglalt, a legrégebbi billentyű adja át a hangját.
- **Pattern törlése:** a **Pat +** melletti **−** gomb törli az aktuális patternt (ha van benne hang, előbb rákérdez). Az azt használó song-lépések kikerülnek, a többi átszámozódik; legalább egy pattern mindig megmarad.
- A **master hangerő** tekerőgombja a VA-VK-n van.

Files menü exportok:

Export	Mit csinál
`Save .bin…`	A szerkesztő natív szerializált formátumát írja (hangszerek minden modulációs beállítással és névvel, sebesség, patternek, song). A régebbi mentések is betölthetők.
`New tune`	Üres dalt kezd a jelenlegi helyett, előbb rákérdez.
`Export blocks (data only)`	Hangszer-táblát + pattern blokkokat fűz a programhoz a `* = $C000`-en.
`Export blocks + miniplayer`	Hozzáadja a teljes lejátszót (sid_init / sid_irq / sid_frame / sid_play_row / sid_set_voice) plusz a PAL frekvencia-táblákat. Export után tegyél egy `JSR sid_init`-et a fő kódodba, ahol a zenének indulnia kell; saját IRQ-ból ehelyett frame-enként egyszer `JSR sid_frame`.
`Export asm (clipboard)`	A teljes assembly forrást vágólapra másolja.
`Import GoatTracker .sng…`	Egy GoatTracker 2 dalt tölt be az aktuális helyére — lásd lentebb a **GoatTracker import** részt.
`Import .sid…`	Egy PSID / RSID dalt alakít szerkeszthető songgá az aktuális helyére — lásd lentebb a **.sid import** részt.
`Export to UltimateBasic`	Egy `tune ... end` blokkot (hangszerek, sorrend lista, pattern sorok szövegként) és `music play`-t ír ki, és az Ultimate Basic szerkesztő kurzorához szúrja be; UB módon kívül a vágólapra másolja. A lejátszót az UltimateBasic fordító építi, így a `music play` / `music stop` vezérli. A hangszer-modulációk `imod` / `ifilt` / `igate` / `itab` sorokba kerülnek (Ultimate Basic 1.6.2+).

GoatTracker import: a **Files → Import GoatTracker .sng…** egy GoatTracker 2 dalt (`.sng`, GTS3–GTS5) tölt be az aktuális helyére. Az átalakítás közelítő, mert a két tracker másképp működik:

- A voice-onkénti sorrend listák (transpose-zal és repeat-tel) egyetlen 3 voice-os dallá lapulnak, 32 soros patternekre vágva; az azonos patternek közösek.

- Hangszerek: az AD / SR, a first wave, a gate timer és a hard restart átkerül; a wavetable-ből wave / arpeggio tábla, a pulse táblából PWM, a filter táblából szűrőmód, rezonancia és cutoff-söprés, a speed tábla bejegyzéséből hangszer-vibrato lesz.
- Pattern parancsok: key off → note-off, portamento → `U` / `D` (soronként legfeljebb 255), tempó → `F`; a toneporta sima hang lesz. A többi parancs kimarad, és egy üzenet felsorolja, mi lett közelítve vagy kihagyva.
- A wavetable által elengedett gate a következő szabad sorban note-off lesz. Csak az első subtune töltődik be.

.sid import: a **Files** → **Import .sid…** egy PSID / RSID fájlt nyit meg (csak az asztali alkalmazásban, mert a VA SID Core-t használja). Válaszd ki a subtune-t és a hosszt másodpercben, majd **Import**: a dal némán lefut, a SID-regiszterek frame-ről frame-re rögzülnek, és ebből lesz a song az aktuális helyére. A felvételt folyamatjelző követi, az átalakítást az *Átalakítás…* állapot.

- **Hangok:** a hangkezdet egy gate-él (2 frame-en belüli újragate egy hangnak számít); ha tartott gate mellett a hangmagasság új félhangra vált és legalább 5 frame-ig ott marad, az új, újraindított (legato) hang lesz. A hangmagasság a hang fő hullámformájának leggyakoribb félhangja; az NTSC dalok a PAL hangtáblára hangolódnak át.
- **Sorok:** a sorok pontosan a hangkezdeteken indulnak. Az alapsorhossz a leghosszabb, amelyre szinte mind illeszkedik; az ennyi vagy egy frame-mel hosszabb sorokat (funktempo, swing, tempóváltás) egy sorral előbb álló `F` effekt állítja.
- **Hangszerek:** AD / SR, a fő hullámforma, pulzusszélesség és PWM, a szűrő (routing, mód, rezonancia, cutoff-söprés), valamint a hang minden frame-jének hullámformája és hangmagassága a következő hangig: ez a wave / arpeggio tábla (előtag és ismétlődő loop, legfeljebb 32 lépés). A fix magasságú dob-attack abszolút, az arpeggio relatív hangokat kap, az azonos hangzású hangok közös döntése alapján; a szabályos, félhangnál kisebb kitérés a hangszer vibratója lesz. A gate-ből note cut, gate időzítő (hard restarttal, ha az ADSR nullázva van) vagy release-sebességű lecsengés lesz. Az azonos hangszerek közösek, legfeljebb 128.
- Egy üzenet felsorolja, mi lett közelítve, például a félhanglépésekké váló glide-ok vagy a 32 lépésnél hosszabb táblák. A BASIC programként induló RSID dalok nem támogatottak.
- **Szűrő és pulzus mozgása:** a szűrősöprés a teljes hangra számít, akkor is, ha a hullámforma már elnémult, és ott ér véget, ahol egy másik hang újraindítja a szűrőt. A gyorsan eső szűrőtáblát az számítja, mikor éri el a végét, a lejátszó frame-enkénti 127-es korlátján belül. A szűrő-LFO a szélsőértékei között pattog. Egy hangszer az összes hangjának pulzustartományát megtartja. A pulzus és a szűrő minden szűrőn átvezetett hangnál újraindul, ezért a sok hangon át szabadon futó söprés csak közelítés, a félhangok közé eső hangmagasság pedig a legközelebbi félhangra kerekül.
- **Több SID:** a 2SID-es dal hat hangos lesz, és a második SID címe is átjön. A 3SID-es dalból az első két SID kerül át, erről üzenet is szól.

Lejátszó ZP használat: `$02` – `$1C` (pattern-mutatók, pattern offset, temp) és `$2F` (aktuális sebesség), valamint `$FB` – `$FD` (tick számláló, sor-index, sorrend-pozíció). A hang-, modulációs és szűrő-állapot a lejátszó kódja utáni `sid_state` területen van. A ZP bájtok ütköznek, ha a fő kódod használja őket — szükség esetén relokálj Expert módon. Két SID-nél a lejátszó a `$50` – `$67` (a 4–6. hang mutatói) és a `$68` / `$69` (az aktuális hang SID-regiszterei) címeket is használja.

Ismert korlátok:

- A lejátszó és állapota `$C000`-n van. Ha a dal adata nem fér el mellettük `$D000` alatt, az adat külön szegmensbe kerül, amely `$A000` alatt végződik (az exportált asm-ben egy második `* =` sor); ne ütközzön a saját programoddal.
- A két SID-es dalhoz kétszer annyi pattern-adat kell: a `.sid` és az assembly export legfeljebb 46 patternt tartalmaz, a többi kimarad, erről az assembly forrásban megjegyzés szól. A szerkesztőben a dal teljes marad. Az **Export to UltimateBasic** csak az első három hangot írja ki, megjegyzéssel.

- A C64 `$D418` globális hangerő megosztott a voice-ok közt — a hangszerenkénti hangerő-csúszka informatív; a sustain szint (az ADSR `S`-e) a tényleges per-voice hangerő.
- A hang-előnézetet a backend rendereli, mielőtt megszólalna, ezért egy pillanattal a billentyű után indul. Minden előnézeti hangnak saját chipje van, így a hangok nem veszik el egymás voice-át.

.sid (PSID) export… (Files menü): a dalt a lejátszható `$C000`-s lejátszóval szabványos PSID fájlba menti — lejátszható a SID Playerben, a VICE `vsid`-jében és bármely más SID lejátszóban. A két SID-es dal PSID v3-ként mentődik, a második SID címével.

SID Player

A **SID Player** toolbar gomb egy lebegő ablakot nyit, amely `.sid` zenefájlokat (PSID és RSID) játszik le, például a HVSC gyűjtemény zenéit.

- **Lejátszási lista:** zenét az **Add .sid files…** gombbal vagy az ablakra húzva adhatsz hozzá, és kattintásra lejátszhatod; a **Remove** és a **Clear playlist** rendet tart. A **Save playlist…** / **Open playlist…** egy `.sidpl` fájlba írja és onnan olvassa a listát, a zenékkel együtt. A lista, az aktuális zene és az ablak nyitott állapota megmarad.
- **Vezérlés:** Play, Pause, Stop, **Previous** / **Next** zene, több dalt tartalmazó fájloknál **Previous / Next subtune**, valamint időszámláló.
- **Tekerés:** a pozíciósáv és a **Back / Forward 10 s**; a kihagyott rész némán fut le.
- **Hangerő:** a tekerőgomb közös a SID szerkesztő master hangerejével.
- **Puffer: Beállítások → Általános → SID előnézet → SID Player puffer** (10, 15, 20 vagy 30 mp). Nagyobb pufferrel nem recseg a hang, ha az app épp dolgozik: a következő számtól ennyire előre renderelődik.
- **Motor:** az ablak kiírja, melyik motor szól (reSID vagy VA SID Core, a SID chippel); a **Beállítások → Általános → SID előnézet** alatt választható. Mindkettő natív Rust C64 magon futtatja a zenét: ciklusra pontos 6510 precíz interrupt-időzítéssel, CIA 6526A, VIC-II időzítés badline-okkal, sprite-DMA-val és raster interruptokkal, valamint a 6510 I/O port — így a raster IRQ-s, CIA-időzített és digi zenék is valódi gép időzítésével szólnak. A 2SID és 3SID dalok (PSID v3 / v4) sztereóban szólnak: SID 1 balra, SID 2 jobbra, a harmadik SID középre.
- **Véletlen lejátszás:** a **Next** melletti kapcsoló véletlen sorrendben játssza a listát — egy szám sem ismétlődik, amíg mind le nem ment, a **Previous** pedig a véletlen sorrendben lép vissza. A beállítást megjegyzi.
- **Belső zeneszámok egymás után:** a **Back 10 s** előtti, alapból bekapcsolt gomb a fájl hátralévő subtune-jait egymás után lejátssza, mielőtt a lista következő elemére lép, véletlen lejátszásnál és egyetlen fájlnál is. Egy subtune a HVSC szerinti hosszáig szól, ha ismert, különben három percig. Kikapcsolva a lista az aktuális subtune után lép tovább. A lista végén, véletlen lejátszás nélkül, az utolsó subtune addig szól, amíg le nem állítod.
- **Hangerő-kiegyenlítés:** a spektrum-analizátor melletti gomb minden dalt kb. azonos hangerővel játszik: egy szint felé emeli vagy halkítja, legfeljebb 12 dB-lel, torzítás nélkül. A dalt az első két másodpercén méri, utána a szint lassan követi. Ha a halk kezdetű dalt felerősítette, és hangosabb rész jön, a szint azonnal, még a rész megszólalása előtt csökken, így az nem torzul. A beállítást megjegyzi.
- **Csillagok:** minden sorban 1–5 csillag van; egy csillagra kattintva értékeled a számot, ugyanarra újra kattintva törlöd. Az értékelés a mentett állapotban és a `.sidpl` lejátszási listákban is megmarad (a régi lista kedvence egy csillag lesz). Az eszköztár **Csak a csillagosak maradjanak** gombja megerősítés után törli a csillag nélküli számokat.
- **Duplikátumok törlése:** az eszköztár gombja törli a listában többször szereplő számokat: azonos HVSC-útvonal, vagy azonos fájltartalom. Az első példány marad, a példányai közül a legtöbb csillaggal.

HVSC (High Voltage SID Collection)

A SID Player eszköztárának **HVSC** gombja keresőpanelt nyit a High Voltage SID Collection mintegy 61 000 C64 dalához.

- **Keresés:** írj be címet, zeneszerzőt vagy mappát (`hubbard commando` , `games/s-z`); minden szónak egyeznie kell. A **Véletlen dalok** 25 véletlenszerű dalt listáz.
- **Zeneszerző:** a keresőmező melletti listában a **Minden** helyett a **Zeneszerző** csak a szerzők nevében keres (`hubbard` , `rob hubbard` , `galway`), és szerzők szerint rendezve listáz. A név a dal `MUSICIANS` alatti mappájából jön, ezért a szerzőt a mappa nevén találod meg, ami lehet művésznév is (`laxity`). A `GAMES` és `DEMOS` mappák kb. 4600 dalának nincs szerzői mappája: ezeket a **Minden** találja meg cím vagy mappa alapján.
- **Lejátszás / hozzáadás / mentés:** kattintásra szól, a **+** a lejátszási listára teszi, a letöltés gomb elmenti a `.sid` fájlt. Letöltés közben a **+** helyén forgó jel látszik, és a második kattintás nem veszi fel újra. A **Minden találat a listára** legfeljebb 100 találatot vesz fel.
- **Dalhosszak:** a HVSC minden aldal hosszát ismeri, ezért a pozíciósáv a valódi hosszt mutatja, és a lista az aldal végén lép tovább.
- **Honnan jönnek a dalok:** a katalógus és az egyes dalok első használatkor egy HVSC tükörről töltődnek le, és az app gyorsítótárába kerülnek, így egyszer lejátszott dal offline is szól. A **Katalógus frissítése** a legújabb listát kéri le.
- **A teljes gyűjtemény: A teljes gyűjtemény letöltése…** a hivatalos HVSC archívumot (kb. 85 MB) a választott mappába tölti és ott kibontja (kb. 380 MB); onnantól a lejátszó ezt a másolatot használja, offline is. A **Helyi HVSC mappa használata…** egy már meglévő gyűjteményre állítja.

Spektrum-analizátor

A SID Player eszköztárának oszlopdiagram-gombja a **Spectrum** ablakot nyitja: LED-szerű oszlopok zöldtől sárgán át narancsig, lassan halványuló cellák és perspektivikus tükörpadló. Azt mutatja, ami a SID Playerben vagy a SID szerkesztőben szól, bármelyik motorral. A **Bands** 8, 16, 24 vagy 32 sávot állít. A **View** a **Mix** (a kimenet analizátora) és a **Channels** (hangonként külön analizátor, két SID-nél 6, a lejátszott regiszterekből számolva, nem a hangból) nézet között vált. Az ablak mozgatható és átméretezhető, a SID Playerrel együtt zárul és nyílik, és csak nyitva rajzol.

A SID Editornak saját, hangonkénti **Csatorna-spektrum** ablaka van: lásd SID Editor.

Csatornagörbék

A **Csatornagörbék** gomb a SID Player spektrum-gombja mellett és a SID Editor tracker-eszköztárán a kettő közös ablakát kapcsolja. Hangonként görgő szintgörbét és nagy hangnevet mutat (két SID-nél 6 hang), és azt követi, ami éppen szól, nem az előre renderelt hangot.

- **SID Player:** mindkét motorral működik, az asztali alkalmazásban.
- **SID Editor:** tracker-lejátszás közben működik, hang-előnézetnél nem. Az ablak a tracker mellett is használható, a Play és a Stop elérhető marad.
- **Mit mutat:** a szint a lejátszott regiszterekből és közelített ADSR-ből számolódik, tehát szint-előzmény, nem oszcilloszkóp. A kötőjel néma hangot jelent. Szünet alatt a görbék állnak.
- **Ablak:** a fejléccel mozgatható, a jobb alsó kis fogantyúval méretezhető; a helyét és a méretét megjegyzi. A SID Playerrel és a SID Editorral együtt zárul és nyílik újra. A gomb, a bezárás gomb vagy az Esc véglegesen bezárja.

Curve Editor

Kész `.byte` lookup táblákat generál matematikai görbékből — szinusz, easing-ek, háromszög/fűrészfog/négyszög és bounce. Ideális sprite-mozgáshoz, raszter-effektekhez, szín-ciklushoz, vagy bármely előre számolt táblával vezérelt animációhoz. A felső eszköztár **Curve Editor** ikonjával nyílik (a SID Editor gomb mellett).

Görbék: Sine, Cosine, Linear, Ease In/Out/InOut (Quad és Cubic), Ease In/Out (Circ), Triangle, Sawtooth, Square, és Ease In/Out/InOut Bounce.

Vezérlők:

Vezérlő	Cél
Start / End value	Kimeneti tartomány. 0..255 8 bites módban, 0..320 16 bites módban.
Number of values	Tábla hossza, 4–512 bejegyzés.
Cycles	Hány oszcilláció a táblán át (csak sine/cosine/triangle/sawtooth/square). Törteket is elfogad (pl. `3.625`).
Phase	Fázis-eltolás fokban (csak sine/cosine).
Combine second curve	Egy második görbe keverése **Mix / Add / Multiply / Min / Max / Subtract**-tel, saját cycles/phase-szel és egy mix-mennyiséggel. Mindkét forrásgörbe szaggatott vezetőként rajzolódik a grafikonra.
Label	Tábla-címke (a görbe nevéből automatikusan javasolva).
Number format	`$XX` hex vagy decimális.
Values per line	8 / 16 / 32 byte `.byte` soronként.

Kimeneti módok:

Mód	Kibocsát
8-bit	Egyetlen `.byte` tábla (0..255 értékek). `LDX #index / LDA table,X`-szel olvasható. Opcionálisan kibocsát egy **sprite-Y olvasó rutint** (`<label>_set_y`) — `LDA <label>,X` / `STA $D001+2N` — egy választható 0-7 sprite-számhoz.
16-bit	Két párhuzamos byte-tábla — `<label>_lo` (alsó 8 bit) és `<label>_hi` (9. bit, 0/1) — **ugyanazzal** az X-szel indexelve (bejegyzésenként 2 byte). A teljes képernyőn átívelő sprite X-hez kell (0..320 > egy byte). Opcionálisan kibocsát egy **sprite-X olvasó rutint** (`<label>_set_x`), amely az alsó byte-ot a `$D000+2N`-be írja, és beállítja/törli a sprite MSB-jét a `$D010`-ben, egy választható 0-7 sprite-számhoz.

Minden Copy / Insert kimenet egy fejléc-megjegyzéssel kezdődik, amely dokumentálja a görbét, a tényleges min/max tartományt, a bejegyzés-számot és a pontos használatot (melyik regisztert táplálja az egyes táblák).

Előnézet:

- **Grafikon** — a görbe kirajzolva a 0 értékkel **felül** és a maximummal **alul**, a C64 sprite-Y / raszter konvencióhoz igazodva (így amit látsz, azt vezérli a tábla hardveren). A grafikon alatti meta-sor mutatja a byte-számot, a generált értékek tényleges Min / Max-át és a görbe nevé(ei)t.
- **Pattogó labda** — animál egy jelölőt a táblán át a **Tempo**-nál (5–240 érték/mp). Tempo 50-nél ez PAL-on (50 Hz) frame-enként egy érték, azaz indexenként egy `.wait_raster` lépés. Play/Pause és Restart gombok, az utolsó ~24 pozíció halványuló nyoma, és élő `Index · Value` kijelzés.

Copy / Insert: az eszköztár két ikonja — a **Copy** vágólapra teszi a táblát; az **Insert into editor** blokkokként fűzi a táblát (és az olvasót, ha engedélyezett) az aktuális programhoz. Az újra-beszúrás **lecseréli** a korábbi Curve Editor beszúrást ahelyett, hogy duplikátumokat halmozna (Blokk és Expert módban is működik).

Files menü:

Művelet	Mit csinál
Save curve (.bin)…	A nyers tábla-byte-okat pontosan úgy menti, ahogy a C64 `INCBIN`-nel olvasná. 16-bit: N lo byte, majd N hi byte.
Load curve (.bin)…	Nyers tábla-byte-okat tölt vissza a szerkesztőbe, az aktuális bitmélység szerint értelmezve (16-bit: első fele lo, második fele hi). A betöltött tábla változatlanul jelenik meg, amíg bármely görbe-vezérlő friss görbét nem generál.
Export demo to blocks	Egy teljes, futtatható sprite demót fűz hozzá: sprite init, raszter-szinkronizált fő hurok, a beágyazott tábla és a labda sprite adata. Az X 0..320-at söpör 8.8 fixpontban a `$D010` MSB-vel, míg a tábla vezérli a sprite Y-t — pontosan a szerkesztő előnézetéhez illeszkedve. Az újra-export lecseréli a korábbi Curve Editor beszúrást.
Export to UltimateBasic	A táblát inicializált Ultimate Basic tömbként írja ki — `var név = array(N)` (16 bitnél `array_word(N)`), utána `data név: …` sorok —, és az UB szerkesztő kurzorához szúrja be (UB módon kívül vágólapra másolja). A címke érvényes, kisbetűs UB névvé alakul; a táblát `név[i]`-vel indexeld. Ultimate Basic 1.5.8+ kell hozzá (beépítve).
Export UltimateBasic demo	Kész, futtatható UB programot készít: labda sprite (`sprdef 0`), a tábla, és egy raszterhez szinkronizált hurok, ami frame-enként egy táblaelemet játszik le. 8 bites táblánál a tábla a sprite **Y**-t vezérli, az X 0..320-at söpör; 16 bites (0..320) táblánál a tábla a sprite **X**-et vezérli, az Y 0..200-at söpör. 256-nál hosszabb táblánál az első 256 elem játszódik le.

Az előnézet reprodukálása a C64-en: az előnézet **lineárisan, 0 → N-1 → 0 hurokban, frame-enként egy értéket** olvas a táblából. Ennek pontos reprodukálásához vezéreld a táblát ugyanígy (növeld az indexet frame-enként egyszer, fordulj körbe a tábla hosszánál). Egy ping-pong vagy részleges tartományú lejátszás máshogy mozog, még ha a byte-értékek azonosak is. Egy működő 16 bites sprite-X példáért lásd a `samples/curve-new-demo.asm`-et.

Táblagenerátor

A **Táblagenerátor** eszköztár-gomb (rács ikon, a Curve Editor mellett) keresőtáblákat és speedcode-ot készít. Válaszd ki a tábla típusát, állítsd be a paramétereit, és nézd az előnézetet; az alsó sor az elemszámot, a méretet és az értéktartományt mutatja (kódnál a bájtokat és a becsült ciklusokat / raszter sorokat).

Típus	Értékek
Képernyősor-címek	bázis + 40 · sor (tetszőleges bázis és sorhossz)
Bitmap sorcímek	pixel Y → bájtcím a 0. oszlopban (8 × 8-as cellaelrendezés)
Bitmap oszlop-eltolás	pixel X → bájt-eltolás a soron belül (hires vagy multicolor)
Pixel bitmaszk	X → bitmaszk (hires / multicolor, `AND`-hez invertálva is)
Szorzás / osztás / maradék konstanssal	x · k, ⌊x / k⌋, x mod k
Negyed-négyzet	⌊(x − eltolás)² / 4⌋, 512 elem — a · b = sqr(a + b) − sqr(a − b)
Reciprok	⌊N / x⌋
Logaritmus / exponenciális	round(log2 x · s), round(2^(x / s))
Bittükrözés	megfordítja a bájtot (multicolorban a bitpárokat) a sprite-ok és karakterek tükrözéséhez
Saját kifejezés	tetszőleges képlet `i`-re (`k` = 0-tól induló index, `n` = darabszám); `sin`, `cos`, `abs`, `min`, `max`, `round`, `floor`, `PI`, `$hex`, `%bin`

Típus	Értékek
Speedcode: memória kitöltése / másolása	szubrutin; *tárolás / iteráció* 0 = teljesen kigörgetett, N = X-ciklus N tárolással menetenként

A 255 fölötti értékekből **lo / hi táblapár** (`címke_lo` , `címke_hi`) vagy `.word` tábla lesz; a negatív értékek kettes komplemensek. Beállítások: címke, hex / decimális, értékek soronként, **Laphatárra igazítás** (`.align 256` , elkerüli a laphatár-ciklust). A **Másolás** vágólapra teszi az assemblyt; a **Beszúrás** a programhoz adja (ugyanazzal a címkével újra beszúrva a korábbit cseréli) — Ultimate Basic módban `array` -t `data` sorokkal szúr be. A Files menüben `.bin` -be is menthető (előbb a lo, utána a hi bájtok), és Ultimate Basicbe exportálható.

Számrendszer-átváltó

Kis, **nem modális** DEC / HEX / BIN átváltó, a SID Player melletti toolbar gombbal nyílik (újabb kattintás vagy Esc zárja). A gomb kiemelve marad, amíg az átváltó látszik. Nyitva maradhat, miközben bármelyik módban tovább szerkeszted a programot.

- Bármelyik mezőbe (**DEC**, **HEX**, **BIN**) írsz, a másik kettő követi. Elfogadja a `$` , `0x` , `%` , `0b` előtagot és az elválasztókat (szóköz, `_`); a negatív decimális érték kettes komplemens lesz az aktuális szélességben; a hibás bevitel pirosan jelölődik.
- **8 / 16 bit** szélesség — 255 fölötti érték automatikusan 16 bitre vált.
- **Bitsor** — egy bitre kattintva átbillented.
- Az infósor az **előjeles** értéket, 16 bites módban az **alsó** és **felső** bájtot mutatja.
- A **másolás** gombok előtaggal másolnak (`53280` , `$D020` , `%1101000000100000`).
- A fejlécénél fogva húzható; megjegyzi az értéket, a szélességet és a pozíciót, és újraindítás után is megnyílik, ha kilépéskor nyitva volt.

© 2026 Zsolt Tarczali — C64 Visual Assembler