

ZX SPECTRUM NEXT · NATIVE Z80 ASSEMBLER

```
NextForge 1.2 by MB Maniax

; Hello, NextForge
    org $8000
Start:
    ld hl,0
    ld de,16384
    ld bc,6144
    ret

HELLO.ASM  Ln 1 Col 1  Lines 7
```

Ilustrační schéma editoru

NEXTFORGE

Uživatelský manuál

Editor · assembler · monitor a debugger · unit testy

Obsah

ČÁST I · EDITOR A ASSEMBLER

1. Seznámení a první spuštění
2. První binární soubor
3. Editor a ovládání
4. Soubory na SD kartě
5. Bloky, hledání a návěští
6. Překlad a výsledkový report
7. Syntaxe, symboly a výrazy
8. Direktivy a makra
9. Projekty s include a výstup
10. Podporované instrukce a limity
11. Když se něco nepovede
12. Přehled zkratk a tisk

ČÁST II · MONITOR A DEBUGGER

13. Seznámení s debuggerem
14. První ladění krok za krokem

15. Obrazovka debuggeru
16. Menu, kontextová menu a nápověda
17. Krokování a běh programu
18. Breakpointy, podmínky a sledování
19. Monitor: paměť, registry a úpravy
20. Zdrojový kód a návěští
21. Jak debugger spouští program
22. Omezení a řešení problémů
23. Přehled kláves debuggeru

ČÁST III · UNIT TESTY

24. Seznámení s unit testy
25. První testy krok za krokem
26. Direktivy pro testy
27. Jak testy běží
28. Tipy a omezení

VERZE MANUÁLU

Popisuje současné funkce NextForge 1.2. Připravované funkce jsou výslovně označené.

Editor a assembler

První část provede editorem, soubory na SD kartě, assemblerem a jeho direktivami — od prvního spuštění po hotový binární nebo NEX soubor.

Kapitoly 1–12. Ladění přeloženého programu popisuje Část II · Monitor a debugger, automatické testy Část III · Unit testy.

1. Seznámení a první spuštění

NextForge je editor a nativní assembler pro ZX Spectrum Next. Zdrojový kód píšete přímo na Nextu, soubory ukládáte na SD kartu a překlad vytváří binární soubor. Tento manuál popisuje vývojovou verzi **1.2** ve stavu k 22. říjnu 2026.

NOVINKY VE VERZI 1.2

- Sloupec taktů v editoru: **EXT** → **K** nebo **Assemble** → **T-states** ukáže na konci řádků s instrukcemi, kolik taktů (T-states) trvají, podle posledního překladu. Text ani komentáře nepřekryje (kapitola 3).

NOVINKY VE VERZI 1.1

- Debugger má lištu menu, kontextová menu na pravé tlačítko a vlastní vyhledávatelnou nápovědu.
- Programy mohou ležet od \$2000 (tedy i ORG \$8000); ladí se i stránkovaný výstup MMU / SAVENEX se vstupem a zásobníkem ze SAVENEX.
- Krokování simuluje DI/EI/HALT, porty, NEXTREG i stránkování; rychlý běh zastaví BREAK.
- Podmínky na libovolný registr, pár nebo paměť s operátory = <> < > <= >=, i bez breakpointů.
- Delší výpisy kódu, zásobníku a paměti, náhled paměti na PC, breakpointy jako červené tečky, nové klávesy J a V.
- Obrazovka programu v debuggeru: F ji ukáže kdykoli, L při pomalém trasování a rychlý běh vždy — ULA, LoRes, Timex, Layer 2, tilemapa, sprity a palety. Rychlý běh dostane stránky na svá místa, takže funguje i jeho vlastní stránkování.
- A zavolá podprogram na kurzoru, kontextové menu nabízí Run from here.
- Klávesové zkratky pro časté akce: **EXT** → **A** přeloží, **EXT** → **R** přeloží a otevře debugger, **EXT** → **D** otevře monitor a debugger.
- Unit testy: bloky TESTCASE / ENDTEST s kontrolami EXPECT, vložené přes TESTCLUDE; **EXT** → **T** je přeloží a spustí, okno reportu je vypíše i s takty a editor obarví výsledky. Každý test začíná se stejnou pamětí STUB nahradí volání ROM a esxDOS a MOCK podvrhne vlastní rutinu (Část III).
- Breakpointy přímo v editoru: **EXT** → **B** nebo kontextové menu obarví řádek červeně a debugger jej převezme; B v debuggeru označí řádek ve zdroji.
- Výrazy znají operátory SjASMPlus s jeho prioritami: posuny, porovnání, bitové a logické operátory i zbytek po dělení (kapitola 7). ASSERT tak spojí více podmínek.
- Větší tabulka symbolů: názvy až 64 znaků, rychlé hledání a podle potřeby další stránky z poolu dokumentů.
- Oprava ukládání: zdroj i binárka se při uložení nejdřív vyprázdní. Na některých instalacích NextZXOS ve verzi 1.0 zůstával na konci uloženého souboru starý text a překlad hlásil Save failed: Set binary length esx=19.
- NextForge přeloží vlastní zdrojový kód a vytvoří stejný NEX jako SjASMPlus. Přibyly k tomu DISP/ENT, SAVENEX BANK, REPT s počítadlem, odkazy Global.Local, delší makra s více parametry a hledání includů v nadřazených adresářích (kapitola 7).
- Podmíněný překlad IF/IFN/ELSEIF/ELSE/ENDIF a IFDEF/IFNDEF s DEFINE (kapitola 8).
- STRUCT, DZ, DC, DD a D24; lokální návěští v makrech (kapitola 8).
- Debugger počítá takty a testy je ověří přes EXPECT CYCLES; editor umí hledání s nahrazením.
- Manuál je rozdělený na dvě části a doplněný snímky obrazovky.

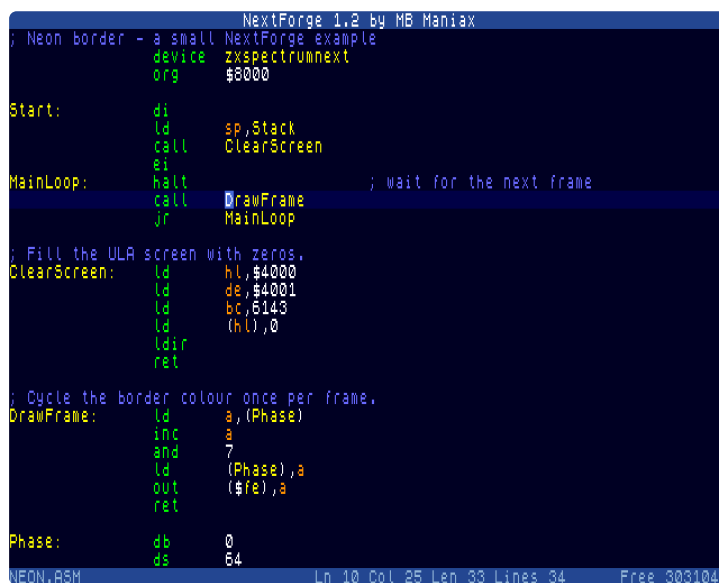
Debugger podrobně popisuje Část II.

Co potřebujete

ZX Spectrum Next s NextZXOS a SD kartou, případně CSpect s odpovídajícím systémem a obrazem SD karty. Program používá 80sloupcový textový režim. Myš usnadňuje ovládání menu a dialogů; základní práce je možná i klávesnicí.

NextForge běží na Nextu s 1 MB i 2 MB paměti. Soubor `NEXTFORGE.NEX` má 176,5 KB (11 bank po 16 KB). Za běhu NextForge pevně zabírá 288 KB a pro dokumenty, výstup assembleru a debugger si vyhradí dalších 320 KB (Next s 1 MB), nebo 960 KB (2 MB). Které stránky to jsou a čemu se vyhnout v laděném programu, popisuje kapitola 21.

1. Zkopírujete `NEXTFORGE.NEX` na SD kartu.
2. V prohlížeči NextZXOS vyberte tento soubor a spusťte jej.
3. Po spuštění se zobrazí prázdný editor. Menu otevřete kliknutím na horní lištu nebo pomocí **EXT** → **0**.
4. Pomocí **File** → **Load...** otevřete zdrojový soubor, nebo začněte psát do prázdného editoru.



```
NextForge 1.2 by MB Maniax
; Neon border - a small NextForge example
device zxspectrumnext
org $8000

Start:
    di
    ld sp,Stack
    call ClearScreen
    ei
MainLoop:
    halt ; wait for the next frame
    call DrawFrame
    jr MainLoop

; Fill the ULA screen with zeros.
ClearScreen:
    ld hl,$4000
    ld de,$4001
    ld bc,6143
    ld (hl),0
    ldir
    ret

; Cycle the border colour once per frame.
DrawFrame:
    ld a,(Phase)
    inc a
    and 7
    ld (Phase),a
    out ($fe),a
    ret

Phase:
    db 0
    ds 54

NEON.ASH Ln 10 Col 25 Len 33 Lines 34 Free 303104B
```

Editor s otevřeným zdrojem: nahoře název a menu, uprostřed text se zvýrazněním syntaxe, dole stavová lišta.

MONITOR A DEBUGGER

Menu **Debug** nebo **EXT** → **D** otevře monitor a debugger: program zastavený před první instrukcí, krokování, breakpointy s podmínkami, paměť a registry. Popisuje je Část II tohoto manuálu.

Jak číst tento manuál

Názvy menu odpovídají anglickým textům programu. **CS** znamená CAPS SHIFT, **SS** znamená SYMBOL SHIFT. Zkratka **EXT** → **klávesa** znamená stisknout CS + SS, obě klávesy uvolnit a poté stisknout uvedenou klávesu. Přepínač CZE/ENG mění jazyk této stránky; volbu si prohlížeč pamatuje.

KLÁVESY / MENU	AKCE
EXT → 0	Otevřít menu.
Help → Help... / EXT → H	Otevřít vyhledávatelnou nápovědu.

2. První binární soubor

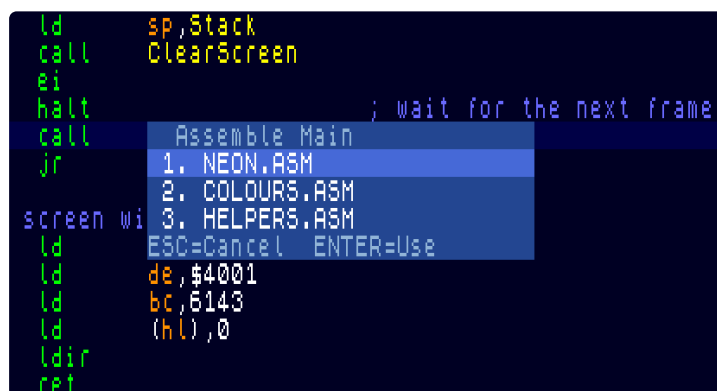
Tento malý příklad lze přepsat do editoru a ihned přeložit. Nastaví počáteční adresu, vloží několik instrukcí a určí jméno výstupu. Sám nic nevykresluje.

```
org $8000

Start:
    ld hl,0
    ld de,16384
    ld bc,6144
    ret

End:
    savebin "hello.bin",$8000,End-$8000
```

1. Napište kód do editoru. Návěští `Start:` a `End:` patří na samostatné řádky; instrukce můžete odsadit.
2. Zvolte **File** → **Save As...** a zadejte `hello.asm`. Vyberte adresář a potvrďte jej klávesou SPACE.
3. Zvolte **Assemble** → **Assemble** nebo stiskněte **EXT** → **A**. V nabídce hlavního zdroje vyberte `hello.asm` a potvrďte ENTER.
4. Po úspěchu report zobrazí velikost **10 bajtů** a cestu k `hello.bin`. Soubor vznikne vedle zdroje.
5. Report zavřete ENTER nebo BREAK. Binární soubor je určený k načtení na adresu `$8000`.



```
ld sp,Stack
call ClearScreen
ei
halt ; wait for the next frame
call Assemble Main
jr 1. NEON.ASM
jr 2. COLOURS.ASM
screen wi 3. HELPERS.ASM
ld ESC=Cancel ENTER=Use
ld de,$4001
ld bc,6143
ld (hl),0
ldir
ret
```

Výběr hlavního zdroje před překladem (zde se třemi otevřenými soubory).

NÁVĚŠTÍ FUNGUJÍ

Návěští není třeba odstraňovat kvůli překladu. Používejte stejnou velikost písmen při definici i odkazu: `Start` a `START` jsou dva různé symboly.

ČERSTVĚ NAPSANÝ ZDROJ

Editor lze přeložit i bez předchozího načtení ze SD. Uložení před překladem je praktické kvůli záloze a jednoznačnému adresáři projektu; nepojmenovaný zdroj používá výchozí výstup `untitled.bin`, pokud jej nezmění `savebin`.

KLÁVESY / MENU	AKCE
SPACE	Potvrdit cílový adresář pro uložení.
ENTER	Potvrdit hlavní zdroj / zavřít report.

CS + SPACE (BREAK)

Zrušit dialog / zavřít report.

3. Editor a ovládání

Horní lišta obsahuje název programu a menu. Uprostřed je zdrojový text se zvýrazněním syntaxe. Spodní stavová lišta ukazuje jméno souboru, řádek **Ln**, sloupec **Col**, délku řádku **Len**, počet řádků **Lines** a volnou paměť **Free**.

Řádek v editoru má nejvýše **80 znaků**. Delší komentář editor při otevření souboru zalomí na další řádky pod jeho středník (při středníku dál než ve sloupci 50 od začátku řádku), takže se nic neztratí a soubor zůstane přeložitelný:

```
PERIPHERAL_3_NR_08 equ $08    ; ABC/ACB Stereo, Internal Speaker, SpecDrum,  
                                ; Timex Video Modes, Turbo Sound Next, RAM  
                                ; contention and [un]lock 128k paging.
```

Delší řádek s kódem, například dlouhý řetězec v DB, se ořízne; takový soubor z editoru neukládejte. Text je uložen v paměti rozdělené mezi otevřené soubory. Editor podporuje nejvýše osm otevřených souborů; dostupný prostor závisí na paměti stroje.

```
NextForge 1.2 by MB Maniax  
; Neon border - a small NextForge example  
device zxspectrumnext  
org $8000  
  
Start: di  
      ld sp,Stack  
      call ClearScreen  
      ei  
MainLoop: halt ; wait for the next frame  
         call DrawFrame  
         jr MainLoop  
  
; Fill the ULA screen with zeros.  
ClearScreen: ld hl,$4000  
            ld de,$4001  
            ld bc,6143  
            ld (hl),0  
            ldir  
            ret  
  
; Cycle the border colour once per frame.  
DrawFrame: ld a,(Phase)  
           inc a  
           and 7  
           ld (Phase),a  
           out ($fe),a  
           ret  
  
Phase: db 0  
       ds 64  
  
NEON.ASM Ln 10 Col 25 Len 33 Lines 34 Free 303104B
```

Obrazovka editoru s ukázkovým programem NEON.ASM.

- | | | | |
|---|---|---|---------------------------|
| 1 | Název programu; po najetí myši nebo EXT → 0 lišta menu. | 2 | Komentář. |
| 3 | Návěští. | 4 | Instrukce nebo direktiva. |

5 Operandy: registry, čísla a symboly mají vlastní barvy.

7 Jméno aktivního souboru.

9 Volná paměť pro text.

6 Aktuální řádek a kurzor.

8 Řádek, sloupec, délka řádku a počet řádků.

Pohyb a úpravy

KLÁVESY	AKCE
CS + 5 / 6 / 7 / 8	Vlevo / dolů / nahoru / vpravo.
CS + SS + 7 / 6	O stránku nahoru / dolů; klávesy držte současně.
SS + Q / E	Začátek / konec řádku.
ENTER	Nový řádek.
CS + 0	Smazat znak před kurzorem.
CS + 9	Smazat znak pod kurzorem.
CS + 3	Tabulátor / odsazení v assemblerovém řádku.
SS + W	Zformátovat aktuální assemblerový řádek.
CS + SPACE	BREAK: zavřít nebo zrušit dialog.
Kolečko myši	Posouvat text editoru.

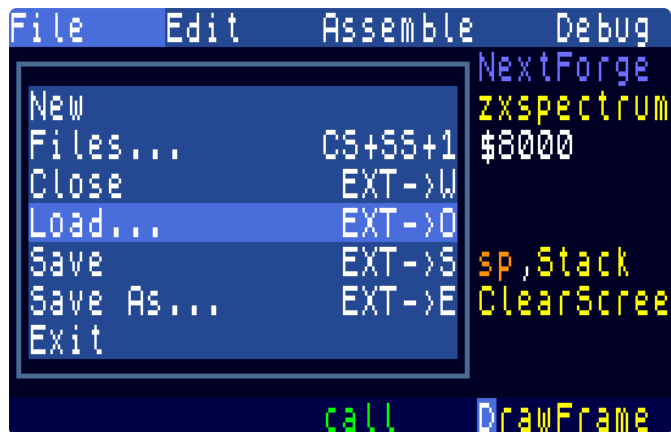
KLÁVESNICE V EMULÁTORU

Zkratky jsou uvedené pro klávesnici Spectra. Na PC záleží na mapování CAPS SHIFT a SYMBOL SHIFT v emulátoru; použijte klávesy, které tyto signály skutečně vysílají.

Menu

Menu otevřete **EXT** → **0** nebo myší na horní liště. Šípkami vybíráte nabídku a položku, ENTER potvrzuje, BREAK zavírá. Myš umožňuje výběr položek menu a souborů v dialogu.

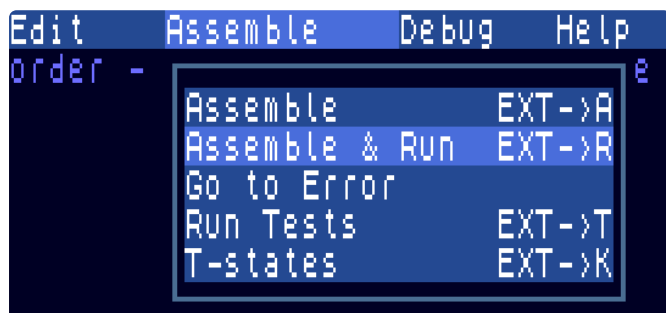
KLÁVESY / MENU	AKCE
EXT → 0	Otevřít menu.
CS + 5 / 8	Předchozí / další nabídka.
CS + 7 / 6	Předchozí / další položka.
ENTER	Spustit vybranou položku.
CS + SPACE (BREAK)	Zavřít menu.



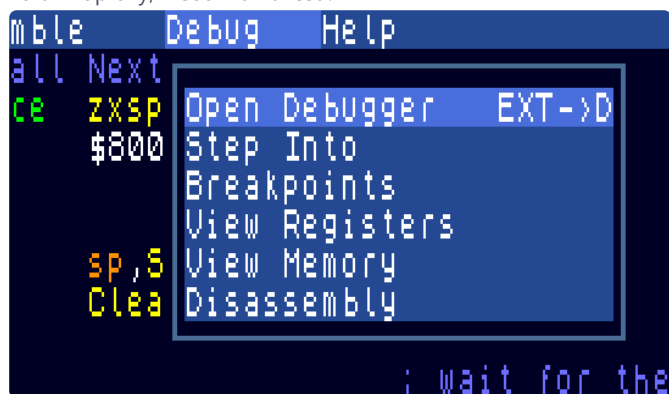
File — soubory a ukončení.



Edit — úpravy, hledání a návěští.



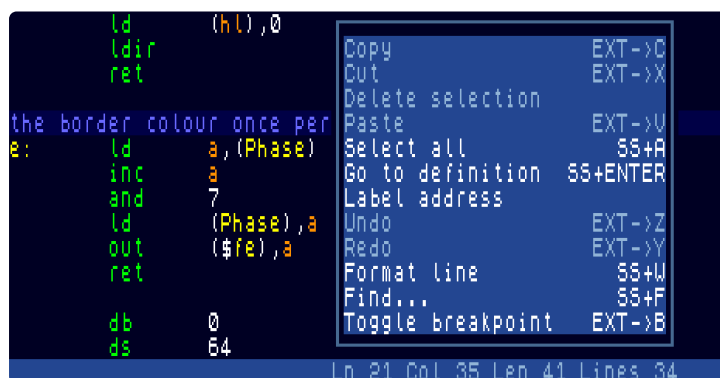
Assemble — překlad (EXT → A), překlad a ladění (EXT → R), skok na chybu, unit testy (EXT → T, Část III).



Debug — monitor a debugger (EXT → D, Část II).

Myš a kontextové menu

Levým tlačítkem umístíte kurzor, tažením označíte text — i jen část řádku. Kolečko posouvá zdroj. Pravé tlačítko otevře kontextové menu pro místo pod ukazatelem; právě nedostupné položky jsou ztlumené (například Copy bez výběru nebo Undo bez historie). **Label address** ukáže adresu návěští pod kurzorem z posledního úspěšného překladu. **Toggle breakpoint** označí řádek jako breakpoint (viz níže).



Kontextové menu nad textem.

Breakpointy v editoru

EXT → B nebo **Toggle breakpoint** v kontextovém menu označí řádek pod kurzorem jako breakpoint; opakováním značku zrušíte. Označený řádek má tmavě červené pozadí (řádek s kurzorem světlejší) a barvy syntaxe zůstávají. Značka text nemění: soubor se nestane změněným a nezabere krok historie úprav.

```

        call    ClearScreen
        ei
MainLoop: halt                ; wait for the next frame
        call    DrawFrame
        jr      MainLoop

; Fill the ULA screen with zeros.
ClearScreen: ld    hl,$4000
             ld    de,$4001
             ld    bc,6143
             ld    (hl),0

```

Značky breakpointů: tmavě červený řádek a světlejší řádek s kurzorem.

Značka putuje se svým řádkem při vkládání a mazání řádků, ENTER na začátku řádku ji posune dolů spolu s textem a spojení řádků Backspacem zachová značky obou. Vložený text a bloky ze schránky přicházejí bez značek. Při otevření debuggeru se značky stanou breakpointy — podrobnosti jsou v kapitole 18. Značky se neukládají do souboru; platí, dokud je soubor otevřený.

Takty instrukcí (T-states)

EXT → **K** nebo **Assemble** → **T-states** zapne sloupec, který na konci každého řádku s instrukcí ukáže šedě, kolik taktů (T-states) trvá; opakováním sloupec skryjete. Nad sloupcem je na horní liště nadpis **T-states**. Hodnoty pocházejí z posledního překladu, jsou stejné jako v debuggeru a unit testech a neberou v úvahu zpomalení paměti (contention) ani čekací cykly Nextu na 28 MHz.

NextForge 1.2 by MB Maniax		T-states
; Neon border - a small NextForge example		
org	\$8000	
Start:		
	di	4
	ld sp,Stack	10
	call ClearScreen	17
	ei	4
MainLoop:	halt ; wait for the next frame	4
	call DrawFrame	17
	jr MainLoop	12
; Fill the ULA screen with zeros.		
ClearScreen:	ld hl,\$4000	10
	ld de,\$4001	10
	ld bc,6143	10
	ld (hl),0	10
	ldir	16/21
	ret	10
; Cycle the border colour once per frame.		
DrawFrame:	ld a,(Phase)	13
	inc a	4
	and 7	7
	ld (Phase),a	13
	out (\$fe),a	11

Sloupec taktů po překladu: 16/21 u LDIR, komentář na řádku s HALT zůstává celý.

- Dvě čísla, například 7/12, mají podmíněné instrukce: menší platí, když se skok neprovede, větší, když ano (JR cc, DJNZ, CALL cc, RET cc). Blokové instrukce jako LDIR ukazují 16/21: poslední průchod a každý průchod před ním. Smyčky se nesčítají.

- Řádek s makrem ukáže součet celého rozvinutí makra.
- Datové řádky (DB, DW, DS, INCBIN...), návěští a komentáře hodnotu nemají.
- Sloupec nikdy nepřekryje text: řádek, jehož text nebo komentář sahá až k místu hodnoty, zůstane celý a číslo se u něj nezobrazí. Číslo se skryje také, když je kurzor v jeho místě.
- Po každé změně zdroje čísla zmizí a nadpis se změní na T:EXT->A, dokud zdroj znovu nepřeložíte (EXT → A). Undo, které vrátí text do přeloženého stavu, je vrátí.
- Hodnoty jsou jen pro plochý obraz (BIN, SAVEBIN). Překlad s DEVICE do NEX nemá zdrojovou mapu, takže sloupec zůstane prázdný.

Integrovaná nápověda s vyhledáváním

Přehled funkcí a zkratk otevřete z Help → Help... nebo přes EXT → H. Začněte psát například label, MMU nebo SS+ENTER: seznam se okamžitě zúží na shody v popisech nebo zkratkách. Zobrazí se počet shod; prázdný filtr ukazuje celý přehled. Filtr má nejvýše 48 znaků. Okno při hledání ani posouvání neblíká a zavření zachová zdroj i pozici kurzoru. Text integrované nápovědy je anglicky.

KLÁVESY / MENU	AKCE
Help → Help... / EXT → H	Otevřít nápovědu; EXT → H ji také zavře.
Psaní textu	Filtrovat popisy i zkratky bez rozlišení velikosti písmen.
CS + 0	Smazat poslední znak filtru.
CS + 7 / 6	Posunout přehled nahoru / dolů.
CS + SS + 7 / 6	Předchozí / další stránka shod.
SS + Q / E	První / poslední stránka shod.
Kolečko myši	Posunout přehled.
CS + SPACE (BREAK) / [Close]	Zavřít nápovědu a vrátit se do editoru.

```

NextForge 1.2 by MB Maniax
NEXTFORGE 1.2 - SEARCHABLE HELP [ Close ]
FUNCTION / CONTEXT                                SHORTCUT / MENU
Blocks: mark start / confirm end (whole lines)    CS+4
Blocks: select all lines                          SS+A
Blocks: copy selected text or lines                EXT->C
Blocks: cut selection or current line              EXT->X
Blocks: paste clipboard, including another file    EXT->U
Blocks: cancel selection                          BREAK / CS+SPACE

Filter: block
Matches: 6
Type to filter  UP/DOWN/PGUP/PGDN/WHEEL scroll  BREAK / EXT->H close
NEON.ASM                               Ln 10 Col 25 Len 33 Lines 34      Free 303104B

```

Nápověda filtrovaná slovem „block“.

4. Soubory na SD kartě

POLOŽKA / ZKRATKA	VÝZNAM
File → Load... / EXT → O	Otevřít zdroj přes prohlížeč NextZXOS.
File → Save / EXT → S	Uložit aktivní soubor pod jeho jménem do jeho adresáře.
File → Save As... / EXT → E	Zadat jméno, vybrat adresář a uložit.
File → Files... / CS + SS + 1	Přepnout mezi otevřenými soubory.
File → Close / EXT → W	Zavřít aktivní soubor; při změnách nabídne uložení.
File → New	Vyprázdnit aktuální editorový buffer.
File → Exit	V této verzi resetuje celý editor.

Ovládání dialogů

KLÁVESY / MENU	AKCE
CS + 7 / 6	Vybrat otevřený soubor.
ENTER / kliknutí	Potvrdit soubor nebo jméno pro Save As.
SPACE	V prohlížeči uložit do zvoleného adresáře.
CS + SPACE (BREAK)	Zrušit výběr nebo ukládání.
ENTER / S	Uložit změny při zavírání souboru.
D	Při zavírání zahodit neuložené změny.

Uložit jako

Nejprve zadejte jméno souboru a potvrďte ENTER. Poté se otevře prohlížeč adresářů. Přejděte do cílového adresáře a klávesou **SPACE** zvolte „uložit zde“. BREAK operaci zruší. U obyčejného Save se používá adresář uložený u daného souboru.

```
out      ($fe),a
ret
Phase:   db      0
         ds      64
Save as: NEON.ASM
```

Save As: jméno se zadává ve stavovém řádku, ENTER pak otevře prohlížeč adresářů.

Více otevřených souborů

Dialog **Files...** zobrazuje soubory, které jsou již v editoru. Šipkami nebo myší vyberte soubor a potvrďte ENTER či kliknutím. Při zaplnění osmi míst některý soubor zavřete. Includes není nutné otevírat do editoru; assembler je načítá ze SD.

```

ld      sp,Stack
call    ClearScreen
ei
halt    ; wait for the next frame
call    Open Files
jr      1. NEON.ASM
        2. COLOURS.ASM
screen wi 3. HELPERS.ASM
ld      ESC=Cancel ENTER=Open
ld      de,$4001
ld      bc,6143
ld      (hl),0
ldir
ret

```

Dialog Files... se třemi otevřenými soubory.

NEW A EXIT

V současné verzi **New** vymaže obsah aktuálního bufferu a **Exit** vymaže všechny otevřené buffery. Exit zatím nevrací do NextZXOS. Před použitím těchto položek si práci uložte; nepoužívejte je jako náhradu bezpečného zavření souboru.

5. Bloky, hledání a návěští

Práce s bloky

Blokové operace pracují s celými řádky. Pomocí **CS + 4** označte začátek bloku, pohybem kurzoru rozšířte výběr a dalším **CS + 4** potvrďte konec. Zvolený blok zkopírujte nebo vyjměte; tím se označení odstraní. Schránku můžete vložit také do jiného otevřeného souboru. Vyjmutí bez označeného bloku vyjme aktuální řádek.

ZKRATKA	AKCE
CS + 4	Označit začátek / potvrdit konec bloku.
SS + A	Vybrat všechny řádky.
EXT → C	Kopírovat blok.
EXT → X	Vyjmout blok nebo aktuální řádek.
EXT → V	Vložit blok ze schránky.

Myší lze tažením označit i část řádku (znakový výběr); kopírování a vyjmutí pak pracují jen s označeným textem.

```
call DrawFrame
jr MainLoop

; Fill the ULA screen with zeros.
ClearScreen: ld hl,$4000
             ld de,$4001
             ld bc,6143
             ld (hl),0
             ldir
             ret

; Cycle the border colour once per frame.
```

Výběr tažením myši od operandu prvního řádku po instrukci posledního.

Hledání

Otevřete **Edit → Find...** nebo stiskněte **SS + F**. Zadejte hledaný text a potvrďte ENTER. Hledání prochází otevřené soubory; další ENTER v dialogu hledá další výskyt. BREAK dialog zavře.

Nahrazení: v dialogu hledání stiskněte znovu **SS + F**, zadejte nový text a potvrďte ENTER. Výskyty v aktuálním souboru od kurzoru dolů se postupně označí a pro každý zvolíte **Y** (nahradit), **N** (přeskočit) nebo **A** (nahradit tento i všechny další); BREAK skončí. Stavový řádek pak ukáže počet nahrazení. Každé Y a celé A jsou jeden krok Undo. Náhrada, po které by řádek přesáhl 80 znaků, nahrazování zastaví.

KLÁVESY / MENU	AKCE
Edit → Find... / SS + F	Otevřít hledání v otevřených souborech.
ENTER	Najít první / další výskyt.
SS + F v dialogu	Nahradit: zadat nový text, pak Y / N / A u každého výskytu.
CS + 0	Smazat poslední znak hledaného textu.
CS + SPACE (BREAK)	Zavřít dialog hledání.

```

        ld     bc,6143
        ld     (hl),0
        ldir
        ret

; Cycle the border colour once per frame.
DrawFrame:  ld     a,(Phase)
            inc     a
            and     7
            ld     (Phase),a
            out     ($fe),a
            ret

Phase:      db     0
            ds     64
Find: Phase

```

Hledání: text se píše ve stavovém řádku, nalezený výskyt má kurzor.

Doplňování návěstí

Napište nejméně **čtyři znaky** začátku názvu, například `lores`, a ponechte kurzor bezprostředně za nimi. Stiskněte **CAPS SHIFT + ENTER**. Otevře se okno **Label completion** se shodnými návěstími z otevřených souborů. Nabídne nejvýše 32 shod, z nichž osm je vidět najednou.

ZKRATKA	AKCE
Edit → Complete label / CS + ENTER	Otevřít nabídku shod po napsání alespoň čtyř znaků.
CS + 7 / CS + 6	Předchozí / další shoda (nahoru / dolů).
CS + SS + 7 / 6	O stránku shod nahoru / dolů.
ENTER	Doplnit zvolené návěstí do zdroje a zavřít okno.
CS + SPACE (BREAK)	Zavřít nabídku bez vložení.

```

Start:      di
            ld     sp,Stack
            call   ClearScreen
            ei     Label completion
MainLoop:   ha     ClearScreen
            ca
            jr
            ca

; Fill the ULA screen
ClearScreen: ld
            ld
            ld     UP/DOWN, PGUP/PGDN, ENTER, ESC
            ld     (hl),0
            ldir
            ret

; Cycle the border colour once per frame.
DrawFrame:  ld     a,(Phase)

```

Doplňování po napsání `Clea` a CS + ENTER.

Při příliš krátkém názvu se zobrazí `Type 4+ label chars`; pokud není shoda, `No label matches`. Nejde o automatický popup při každém napsaném znaku: nabídku vyvoláte uvedenou zkratkou.

Skok na definici

KLÁVESY / MENU	AKCE
Edit → Go to definition / SS + ENTER	Přejít na definici názvu pod kurzorem v otevřených souborech.

Na řádku například `call lores_init` umístěte kurzor na název `lores_init` a stiskněte **SYMBOL SHIFT + ENTER**. Editor vyhledá jeho definici, případně přepne do jiného otevřeného souboru a umístí kurzor na řádek s návěštím. Při neúspěchu ukáže `Definition not found`.

DVĚ RŮZNÉ ZKRATKY

CS + ENTER doplňuje rozepsaný název z okna shod. **SS + ENTER** přejde na implementaci již napsaného názvu. Include, který je pouze na SD, nejprve otevřete; editor tyto funkce provádí nad otevřenými soubory.

POLOŽKY EDIT

Undo a Redo fungují pomocí `EXT → Z / Y` i v menu Edit. Každý otevřený soubor uchovává osm kroků; nová úprava zruší jeho větev Redo. Při nedostatku paměti se historie uvolní ve prospěch textu. Cut, Copy a Paste používají stejné operace jako klávesové zkratky.

Levý klik přesune kurzor; tažení označuje text po znacích i přes řádky. Psaní nahradí výběr, Backspace a Delete ho smažou. Pravý klik otevře kontextové menu s kopírováním, vyjmutím, mazáním, vložením, výběrem všeho, Undo/Redo, formátováním a hledáním. U návěští nabízí skok na definici a globální adresu z posledního úspěšného překladu; po úpravě je nutný nový překlad. Pravý klik mimo výběr přesune kurzor na místo kliknutí. BREAK zavře menu nebo zruší výběr.

??dek s kurzorem m? jemn? zv?razn?n? pozad?; barvy syntaxe z?st?vaj? zachovan? a ozna?en? text m? p?ednost.

Pokud skok na definici nenajde n?v??t? v otev?en?ch souborech, nab?dne **Search includes**. Hled?n? na SD za?ne a? po potvrzen?. Samostatn? okno ukazuje otev?ran? a prohled?van? soubor, ??dek a po?et soubor?. Proch?z? i vno? en? include, u otev?en?ch soubor? pou??v? aktu?ln? neulo?en? text. BREAK nebo tla??tko Cancel hled?n? zru??. Nalezen? soubor otev?e a p?esune kurzor na definici; p?i pln?ch slotech nejprve zav?ete jin? soubor. Hled?n? porovn?v? p?esn? z?pis n?zvu; opakovan? lok?ln? n?zvy nerozli?uje podle scope a podm?nky assembleru nevyhodnocuje.

6. Překlad a výsledkový report

1. Zvolte **Assemble** → **Assemble** nebo stiskněte **EXT** → **A**.
2. V dialogu vyberte hlavní zdroj projektu z otevřených souborů. Pro projekt s includes je to soubor, který určuje ORG a obsahuje INCLUDE, nikoli jednotlivý modul.
3. Assembler provede dva průchody: první vytvoří tabulku symbolů a rozvržení adres, druhý vyhodnotí odkazy a zapíše bajty.
4. Po úspěšném překladu uloží binární výstup. Při chybě zdroje se ukládání nespustí.

KLÁVESY / MENU	AKCE
CS + 7 / 6	Posouvat report nahoru / dolů.
CS + SS + 7 / 6	O stránku nahoru / dolů.
Kolečko myši	Posouvat report.
ENTER / CS + SPACE (BREAK) / kliknutí	Zavřít report.

Co znamená průběh

Za běhu report ukazuje číslo průchodu, právě zpracovávaný hlavní soubor nebo include, číslo řádku a počet procházených řádků include. Otáčející se indikátor a zvýrazněné údaje ukazují aktivitu. Aktualizují se jen změněné znaky, takže okno při běžném průběhu neblíká. Příprava stránkovaného NEX ukazuje počet procházených bank. Při ukládání jsou vidět zapsané/celkové bajty; soubory nad 64 KB používají KiB.

Running · Pass 2 / 2

src/main.asm

Line 29 · Include lines 640

Při ukládání se zobrazuje zapisovaný počet bajtů a fáze operace: příprava adresářů, otevření souboru, zápis, nastavení délky, synchronizace a zavření. Jde o průběžné údaje, nikoli procenta celého překladu — celkový počet řádků zahrnutého projektu není předem znám.

Hotovo nebo chyba

Úspěšný výsledek ukazuje velikost a cestu k binárce. Chyba zdroje obsahuje řádek, sloupec a při chybě v include také jeho jméno. Chyba při ukládání ukáže cílovou cestu, neúspěšnou operaci a kód esxDOS; nejde o chybu prvního řádku zdroje.

```

NextForge 1.2 by MB Maniax
; Neon border - a small NextForge example
device zxspectrumnext
org $8000

Start:
    Assembly report done

MainLo
    ✓ 05 Scan source lines
    ✓ 06 Skip comments and blank lines
    ✓ 07 Validate labels
    ✓ 08 Recognize directives
    ✓ 09 Recognize mnemonics
    ✓ 10 Check required operands
    ✓ 11 Reject unexpected operands
    ✓ 12 Reject empty operand slots
    ✓ 13 Check operand separators
    ✓ 14 Save binary / store error
    ✓ 15 Restore screen banks
    ✓ 16 Present result

Result: Binary OK Bytes 16896
/project/neon.hex
UP/DOWN/PGUP/PGDN/WHEEL scroll ENTER/ESC/CLICK close

out ($fe),a
ret
Phase: db 0
        ds 64
NEON.ASM Ln 10 Col 25 Len 33 Lines 34 Free 319488B

```

Úspěch: všechny kroky zelené, velikost a cesta k výstupu.

```

NextForge 1.2 by MB Maniax
; Neon border - a small NextForge example
device zxspectrumnext
org $8000

Start:
    Assembly report done

MainLo
    ✓ 05 Scan source lines
    ✓ 06 Skip comments and blank lines
    ✓ 07 Validate labels
    ✓ 08 Recognize directives
    ✗ 09 Recognize mnemonics
    ? 10 Check required operands
    ? 11 Reject unexpected operands
    ? 12 Reject empty operand slots
    ? 13 Check operand separators
    ? 14 Save binary / store error
    ? 15 Restore screen banks
    ? 16 Present result

Result: error line 10 col 1
UP/DOWN/PGUP/PGDN/WHEEL scroll ENTER/ESC/CLICK close

out ($fe),a
ret
Phase: db 0
        ds 64
Assemble error: Undefined symbol line 10 col 1

```

Chyba: červený křížek u kroku, řádek a sloupec; stavová lišta uvádí příčinu.

Report posouváte šípkami, Page Up / Page Down nebo kolečkem myši. ENTER, BREAK nebo kliknutí jej zavře.

Assemble → **Go to Error** přejde na chybné místo v hlavním zdroji; při chybě v include se orientujte podle uvedené cesty a řádku a otevřete příslušný soubor.

CO SE PŘEKLÁDÁ

Hlavní zdroj se čte z vybraného editorového bufferu. Includes a incbin se čtou ze SD karty. Změny v otevřeném include před překladem uložte, jinak assembler použije jeho starší uloženou verzi.

LADĚNÍ PROGRAMU

Monitor a debugger otevře **Assemble** → **Assemble & Run** (EXT → R: přeložit a hned ladit) nebo po úspěšném překladu menu **Debug** (EXT → D). Popisuje je samostatná Část II · Monitor a debugger. **Assemble** → **Run Tests** (EXT → T) přeloží projekt i s testy a spustí je; nic přitom neukládá (Část III).

7. Syntaxe, symboly a výrazy

Komentář začíná středníkem `;`. Prázdné řádky a komentáře se přeskočí. Mnemoniky instrukcí a názvy registrů nerozlišují velikost písmen; symboly ji rozlišují. Globální návěští jsou společná celému projektu, tedy i napříč includes; lokální návěští s tečkou používají obor předchozí globální definice.

```
Start:  ld a,1      ; instrukce na řádku s návěštím
Loop:
    djnz Loop
Count  equ 16
    dw Start,Count
```

Podporována jsou návěští s dvojtečkou, samotný název na řádku i forma `name instruction`. Kvůli přehlednosti používejte dvojtečku. Název může mít nejvýše 64 znaků. Tabulka symbolů nejdřív plní dvě vlastní stránky (asi 800 běžných názvů) a pak si podle potřeby bere stránky ze sdíleného poolu dokumentů; vrátí je až další překlad. Návěští začínající tečkou, například `.loop`, patří do oboru předchozí globální definice. Stejný lokální název tak můžete používat v různých rutinách. Lokální názvy mají nejvýše 59 znaků včetně tečky. Odkaz `.loop` používá aktuální obor. Zápis `Global.loop` odkazuje na lokální návěští jiného oboru a `Global.loop:` ho i definuje, stejně jako v SjASMPlus; taková definice zároveň otevře obor pro další lokální návěští. Moduly zatím nejsou podporovány.

Zápis čísel a operátory

ZÁPIS	VÝZNAM
123	Desítkové číslo.
\$8000, #8000, 0x8000	Šestnáctkové číslo.
%10101010	Dvojkové číslo.
%1'0000'000, \$FF'FF	Apostrof mezi číslicemi je jen oddělovač skupin, jako v SjASMPlus.
'A'	Hodnota jednoho znaku.
\$	Adresa začátku právě zpracovávaného příkazu.

Operátory mají stejné priority jako v SjASMPlus (a v jazyce C). Tabulka je řadí od nejvyšší; operátory se stejnou prioritou se vyhodnocují zleva doprava.

OPERÁTORY	VÝZNAM
+ - ~ ! HIGH LOW	Unární plus a minus, bitová negace, logická negace, horní a dolní bajt.
* / %	Násobení, dělení, zbytek po dělení.
+ -	Součet a rozdíl.
<< >> >>>	Posun vlevo, aritmetický a logický posun vpravo.
< <= > >=	Porovnání se znaménkem.
= == !=	Rovnost a nerovnost.
&	Bitové AND.
^	Bitové XOR.

OPERÁTORY	VÝZNAM
	Bitové OR.
&&	Logické AND.
	Logické OR.
()	Závorky, nejvýše 16 do hloubky.

Porovnání a logické operátory dávají -1 (pravda) nebo 0 (nepravda). Celočíselné dělení se znaménkem zaokrouhluje k nule a zbytek má znaménko dělence. Hodnoty i mezivýsledky musí být v rozsahu -65535 až 65535; číselný literál nejvýše 65535. Bitové operátory pracují s dvojkovým doplňkem, takže `~0` je -1 a `value & $FF` je bajt. `HIGH value` a `LOW value` vybírají horní a dolní bajt hodnoty; funguje i `high(label)`. V DB může znak v apostrofech začít výraz, například `'A' | $80`. Slovní operátory SjASMPlus (`and`, `shl` ...) a escape sekvence v řetězcích zatím nejsou podporovány.

```
FLAG_A equ 1 << 0
FLAG_B equ 1 << 3
MASK   equ ~(FLAG_A | FLAG_B) & $FF
Start:  ld a, MASK
        and FLAG_A | FLAG_B
        ld h1, Table + 2 * 3
        assert Table >= $8000 && Table < $C000
Table:  dw 1, 2, 3, high Table
```

Závorky kolem celého operandu znamenají přístup do paměti, například `ld h1, (table)`. Operand, který závorkou jen začíná, je hodnota: `ld bc, (ROWS-1)*80` načte číslo, stejně jako v SjASMPlus.

Dopředné odkazy

Instrukce, DB a DW mohou odkazovat na návěští definované později. Také EQU může obsahovat dopředný odkaz; nejvýše 32 takových výrazů se dořeší mezi průchody. `ORG`, `MMU`, počet bajtů `DS` a podobné hodnoty, které určují rozložení, musí být známy již při prvním zpracování; dopředné EQU v nich projde, pokud jsou do toho místa definované všechny symboly, na kterých závisí. Kruhové definice EQU a chybějící symboly způsobí chybu.

8. Direktivy a makra

DIREKTIVA	POUŽITÍ
ORG address	Počáteční adresa nebo změna adresy zápisu; po MMU lze přejít na libovolnou CPU adresu bez vyplnění mezer.
MMU first [last],page[,address]	Mapování 8KB stránek do slotů Nextu při překladu; volitelná adresa provede ORG. Příklady a export NEX jsou v kapitole 9.
name EQU expression	Definice konstanty bez vložení bajtů.
DB / DEFB / DEFM	Bajty a řetězce oddělené čárkami; bajtové výrazy – 128 až 255.
HEX 00 01 FF	Bajty z dvojic šestnáctkových číslic; mezery, čárky a uvozovky mezi nimi se nepočítají.
DZ / DC	Jako DB; DZ přidá na konec nulu, DC nastaví bit 7 posledního znaku každého řetězce.
DW / DEFW	16bitové hodnoty, nejprve nižší bajt (little-endian).
DD / DWORD / D24	32bitové a 24bitové hodnoty; výraz má 16 bitů a znaménko se rozšíří.
DS / DEFS count[,fill]	Rezervace známého počtu bajtů; výchozí výplň 0.
REPT count[,name] / ENDR	Opakování bloku při překladu; počet 0–65535 musí být známý v prvním průchodu. Volitelné <code>name</code> čísluje opakování od 0.
STRUCT name[,offset] / ENDS	Struktura s poli; viz níže.
DISPLAY / OPT	Platí jen pro SjASMPlus na PC; NextForge je přeskočí, i když je řádek DISPLAY v includeu delší než 160 znaků.
IF / IFN / ELSEIF / ELSE / ENDIF	Podmíněný překlad podle výrazu známého v místě podmínky; viz níže.
DEFINE name [value], IFDEF / IFNDEF	Příznak pro podmínky IFDEF a IFNDEF; s hodnotou i konstanta.
DISP address / ENT	Kód za DISP se přeloží pro zadanou adresu, ale uloží se na aktuální místo; ENT se vrátí. ORG uvnitř DISP mění jen přeloženou adresu.
INCLUDE "file.asm"	Vložení zdrojového souboru. Hledá se vedle vkládajícího souboru, potom v nadřazených adresářích, takže <code>include "src/x.asm"</code> funguje z kteréhokoli souboru projektu.
INCBIN "file.bin"	Vložení celého binárního souboru; hledá se jako u INCLUDE.
SAVEBIN "file"[,address,length]	Jméno výstupu a případně rozsah adres.
DEVICE zxspectrum48 / zxspectrum128 / zxspectrumnext	Akceptované názvy zařízení. DEVICE ZXSPECTRUMNEXT umožňuje stránkování pomocí MMU; ostatní zařízení zůstávají plochá.
ASSERT expression	Nenulový výraz při druhém průchodu; porovnání, <code>&&</code> a <code> </code> spojí více podmínek.

DIREKTIVA	POUŽITÍ
TESTCASE "name" / ENDTEST, EXPECT, TESTCLUDE "file", STUB RST n,label, MOCK routine,fake	Unit testy, jejich kontroly, vložení testů jen při Run Tests a náhrady volání ROM a rutin v debuggeru; popisuje je kapitola 26.

```
Message: db "NextForge",0
Table:   dw Message,Start
Buffer:  ds 32,0
        assert Buffer >= $8000
```

ASSERT podporuje porovnání `< <= > >= = == !=`. DW zapisuje spodních 16 bitů hodnoty. DB přijímá řetězce v jednoduchých i dvojitéch uvozovkách.

Opakování bloku: REPT

REPT počet a ENDR opakují zdrojový blok při překladu. Nevytvářejí běhovou smyčku. Počet 0–65535 musí být známý v prvním průchodu; při nule se tělo přeskočí včetně definic návěstí. Hodnota \$ se vyhodnotí znovu při každém opakování.

```
REPT 4
db low $
ENDR
```

Podporovány jsou čtyři vnořené úrovně a celkem 128 uložených řádků přes aktivní bloky; vnořená těla spotřebují další místo. REPT i ENDR píšete na samostatné řádky bez návěstí. Blok se musí uzavřít ve stejném souboru. Tělo může vkládat INCLUDE a volat již definovaná makra; REPT uvnitř volaného makra není podporováno. Opakované definice stejného návěstí vyvolají chybu. Funguje s BIN i bankovaným NEX; příklad je v `tests/fixtures/rept`.

Podmíněný překlad

Řádky mezi IF výraz a ENDIF se přeloží, jen když je výraz nenulový; IFN podmínku obrací. ELSEIF výraz a ELSE přidávají další větve a platí první pravdivá. Výraz musí jít vyhodnotit v místě podmínky, tedy jen ze symbolů definovaných dříve, aby oba průchody zvolily stejné větve.

```
DEBUG    equ 1
        DEFINE NEXT_ONLY

        IF DEBUG
        call DumpRegisters
        ELSE
        nop
        ENDIF

        IFDEF NEXT_ONLY
        nextreg $07, 3          ; 28 MHz
        ENDIF
```

IFDEF jméno a IFNDEF jméno testují jména z DEFINE jméno. Návěstí a konstanty EQU nevidí a velikost písmen rozlišují, stejně jako SjASMPPlus. DEFINE s hodnotou navíc definuje konstantu s touto hodnotou (výrazem), což pokryje běžná číselná použití; obecnou textovou náhradu SjASMPPlus NextForge nemá. Přeskočené řádky nedefinují

návěští ani nevolají makra a mohou obsahovat i direktivy, které NextForge nezná. Podmínky se dají vnořit do 16 úrovní a fungují v blocích REPT i v makrech, kde mohou testovat parametr; IF v makru se musí uzavřít v jeho těle.

Makra

Makro definujete před prvním použitím pomocí `macro name [param1, param2, ...]` a ukončete `endm`. Parametrů může být několik, oddělených čárkami, a volání musí předat stejný počet argumentů. Argumenty se dělí na čárkách mimo uvozovky, takže `' , '` nebo `"a,b"` zůstanou jedním argumentem. Jména maker a parametrů nerozlišují velikost písmen; SjASMPlus je u parametrů rozlišuje, proto je pro kompatibilitu pište v těle stejně. Parametr se nahrazuje jako celý identifikátor, nikoli uvnitř uvozovek.

```
macro ESXCALL code
    rst $08
    db code
    ret
endm

ReadFile:
    ESXCALL $9D

macro SETMMU slot, page
    nextreg $50+slot, page
endm

SETMMU 6, 40
```

K dispozici je osm maker. Jména parametrů a řádky těla se ukládají za sebou do 960 bajtů, takže se vejdou desítky krátkých řádků, každý do 160 znaků. Návěští s tečkou (`.loop`) jsou lokální pro každé volání, takže makro může obsahovat smyčku; návěští bez tečky se definuje při každém volání, druhé volání tedy ohlásí duplicitu. Makro může volat jiné makro, nejvýše do hloubky 6. INCLUDE v těle makra podporovaný není.

Struktury: STRUCT

Mezi `STRUCT` `jméno[,offset]` a `ENDS` popište pole: `[návěští] typ [výchozí hodnota]`. Typy jsou `BYTE/DB`, `WORD/DW`, `D24`, `DWORD/DD`, `BLOCK/DS` počet[,výplň], `ALIGN` [hranice[,výplň]] a dříve definovaná struktura. Stejně jako v SjASMPlus má `jméno` hodnotu velikosti struktury a `jméno.pole` offset pole. Instance `[návěští] jméno [inicializace]` vloží bajty struktury a s návěští definuje `návěští.pole` jako adresy, včetně vnořených polí.

```
STRUCT sprite
x:    BYTE 0
y:    BYTE 0
pattern: BYTE 1
flags: WORD $8000
ENDS

player: sprite 120, 96      ; x, y; the rest keeps its defaults
    ld ix, player
    ld a, (ix + sprite.y)
    ld (player.x), a
```

Inicializace vyplňuje pole `BYTE`, `WORD`, `D24` a `DWORD` v pořadí definice; `BLOCK` a `ALIGN` se přeskakují. Prázdná položka (`{ , 5}`) nechá výchozí hodnotu a složené závorky seskupí hodnoty vnořené struktury (`{{1, 2}, 3}`).

Struktura může být 16 se 128 poli celkem.

Zarovnání dat

`ALIGN boundary[,fill]` zarovná adresu na známou mocninu dvou (1 až 32768). Mezilehlé bajty vyplní nulou nebo zadanou hodnotou. Například `align 256` připraví tabulku tak, aby její dolní adresový bajt byl nula.

9. Projekty s include a výstup

Rozdělte projekt do modulů a ponechte jeden hlavní soubor, který určuje jejich pořadí. Následující příklad předpokládá, že `src/main.asm` definuje návěští `Start:` a že všechny uvedené soubory existují:

```
device zxspectrum128
org $2000
jp Start

include "src/constants.asm"
include "src/main.asm"
incbin "assets/font.bin"

End:

assert End-$2000 <= $2000
savebin "build/demo.bin", $2000, End-$2000
```

Jak se hledají soubory

Relativní cesty `INCLUDE` a `INCBIN` se vyhodnocují vůči adresáři souboru, který je obsahuje. Pokud `src/main.asm` vloží `include "util.asm"`, hledá se `src/util.asm`. Relativní `SAVEBIN` se naopak vždy vztahuje k adresáři hlavního zdroje.

Použit lze lomítka i zpětná lomítka, absolutní cesty a prefixy disků. Cesty s mezerami píšete do uvozovek. Includes lze vnořit do čtyř aktivních úrovní; kruhové vkládání překročí tento limit. Podporovány jsou konce řádků CRLF, LF a CR, tabulátory i poslední řádek bez ukončení. Include může mít řádky do 160 znaků; delší řádek projde, pokud před 160. znakem začíná komentář, protože se ztratí jen jeho konec. Editor při otevření dlouhé komentáře zalomí, ale delší řádky s kódem ořízne; soubor s nimi z editoru neukládejte, jinak přijdete o jejich konce.

Výstupní soubor

Bez `SAVEBIN` vznikne z `main.asm` soubor `main.bin`; u jiných přípon se přidá `.bin`. Jeden překlad vytváří jeden výstup. Pokud je `SAVEBIN` uvedeno několikrát, rozhoduje poslední. Chybějící nadřazené adresáře výstupu se vytvoří automaticky.

Binárka je souvislý blok, nejvýše 65535 bajtů. Neobsahuje hlavičku ani adresu pro načtení. Výchozí rozsah začíná prvním emitovaným bajtem; `SAVEBIN` s adresou a délkou jej může omezit. Bez MMU jsou nové mezery vytvořené `ORG` vyplněné nulami. V plochem režimu se pozdější `ORG` může vrátit na již použité adresy, nikoli pod první emitovanou adresu; původní bajty zůstávají, dokud je nepřepíšete.

OVĚŘENÝ VĚTŠÍ PROJEKT

Regresní sada obsahuje celý projekt `PAPER` s includes, makrem `ESXCALL` a odděleným rozvržením kódu a proměnných. Jeho 7155bajtový výstup je porovnáván bajt po bajtu se `SjASMPlus`. Vzor najdete v `tests/fixtures/paper`.

BEZPEČNOST VÝSLEDKU

Chyba ve zdroji ponechá předchozí binárku beze změny. Pokud ale selže SD karta při samotném ukládání, může zůstat částečný soubor. Za platný nový výstup považujte jen překlad s úspěšným dokončením.

Stránkování při překladu: MMU

Po `DEVICE ZXPECTRUMNEXT` lze použít MMU `první_slot [poslední_slot]`, `první_stránka [,adresa]`. Sloty jsou 0–7, stránky 0–223 a jedna stránka má 8 KB. Rozsah mapuje po sobě jdoucí stránky. Místo slotu lze napsat jeho počáteční adresu, například \$8000 pro slot 4. Volitelná poslední adresa zároveň provede ORG; jinak se PC nemění.

```
device zxspectrumnext
MUSIC_CODE_PAGE_LO equ 24
    MMU 6 7,0
    org $C000
Start: ret
    MMU 4,MUSIC_CODE_PAGE_LO
    org $8000
    include "music.asm"
    savenex open "music.nex",Start,$BFF0
    savenex auto
    savenex close
```

MMU řídí umístění dat při překladu a nevkládá žádnou instrukci. Pro přepnutí stránek za běhu programu použijte NEXTREG. Různé stránky na stejné adrese zůstávají oddělené a NEX ukládá jejich skutečný obsah. Data se ukládají do volné paměti editoru; při jejím vyčerpání se zobrazí **assembly output memory full**. Otevřené zdrojové soubory zůstávají zachované.

Po MMU může ORG přejít i na nižší adresu; nevyplňuje mezery a nemaže dříve zapsaná data. Pro BIN zadejte výslovně `SAVEBIN "name.bin",adresa,délka`, který čte konečné mapování slotů. Připravte všechna data před SAVENEX AUTO. Varianty MMU e/w/n, PAGE/SLOT a 16KB stránkování ZX128 zatím nejsou podporované.

Spustitelný NEX pro NextZXOS

Pomocí `SAVENEX` vznikne soubor s hlavičkou NEX V1.2 a daty bank, který lze spustit z prohlížeče NextZXOS:

```
device zxspectrumnext
    org $8000
Start:
    nextreg 7,3
    mul d,e
    ret
    savenex open "demo.nex",Start,$BFF0
    savenex core 3,0,0
    savenex cfg 7,0,0,1
    savenex auto
    savenex close
```

OPEN určí jméno, vstupní adresu a zásobník. **CORE** určí minimální verzi jádra. **CFG 7,0,0,1** znamená bílý okraj, zavřít soubor po načtení, resetovat NextRegs a vyžadovat rozšířenou RAM. **AUTO** vybere nenulové banky a **CLOSE** dokončí nastavení; fyzické uložení proběhne až po úspěšném překladu.

`SAVENEX BANK n[,n...]` uloží uvedené banky, i prázdné, v pořadí NEX (5, 2, 0, 1, 3, 4, 6, ...); banku, která v tomto pořadí už minula, odmítne. AUTO potom přidá neprázdné banky za poslední z nich. Bez BANK i AUTO obsahuje soubor jen hlavičku.

Podporováno je výchozí mapování i stránkování MMU. AUTO ukládá nenulové banky v pořadí 5,2,0,1,3,4,6,7,...,111. Každá banka zabírá 16 KB; nezapsané oblasti se doplní nulami. Vstupní banka může být 0–111, verze souboru je 2.

Banky 48–111 vyžadují příznak 2MB RAM, například `SAVENEX CFG 7,0,0,1`. `SCREEN`, `BAR`, `CFG3` a připojené soubory zatím nejsou podporované.

LoRes tutorial s předpočítanými tabulkami

Otevřete `lowres_nextforge.asm` a ponechte ve stejném adresáři `wave_table.bin` a `ribbon_texture.bin`. Překlad vytvoří `lowres_tutorial.nex` o velikosti 16896 bajtů, shodný s původním tutorialem přeloženým SjASMPlus.

Varianta pro NextForge nahrazuje dvě Lua tabulky pomocí INCBIN a vynechává hostitelské `OPT/DISPLAY`. Obecný Lua interpreter v NextForge zatím není.

10. Podporované instrukce a limity

Assembler podporuje následující skupiny základního Z80 v uvedených podobách. Nejde o úplnou kompatibilitu se všemi rozšířeními SjASMPlus.

SKUPINA	PODPOROVANÉ PODOBY
LD	8bitové registry, (HL), okamžité hodnoty; BC/DE/HL/SP; A s (BC)/(DE) nebo absolutní adresou; slovní registry s absolutní pamětí; SP,HL; A ↔ I/R.
INC, DEC, PUSH, POP	INC/DEC pro 8bitové registry, (HL) a BC/DE/HL/SP; PUSH/POP pro BC/DE/HL/AF.
Aritmetika	ADD, ADC, SUB, SBC, AND, XOR, OR, CP s registrem, (HL) nebo okamžitou hodnotou; ADD/ADC/SBC HL,rr.
Skoky a návraty	JP/CALL/RET se všemi osmi podmínkami; JP (HL); JR bez podmínky nebo NZ/Z/NC/C; DJNZ; RST.
Bity a posuny	RLC/RRC/RL/RR/SLA/SRA/SLL/SRL a BIT/RES/SET pro registry a (HL).
Ostatní	EX DE,HL; EX (SP),HL; EX AF,AF'; IM 0/1/2; IN/OUT s (C), port s okamžitou hodnotou a A; běžné pevné instrukce a ED blokové přenosy, hledání a I/O.
IX a IY	Načtení/uložení 16bitové hodnoty nebo absolutní paměti; LD SP,index; INC/DEC; PUSH/POP; ADD index,BC/DE/stejný index/SP; EX (SP),index; JP (index).
Indexovaná paměť	(IX/IY), (IX/IY±výraz): načtení/uložení bajtu, INC/DEC, aritmetika, posuny a BIT/RES/SET. Podporovány jsou i obyčejné H/L.

Relativní skoky a indexové posuny musí být v rozsahu -128 až 127. `jp (ix)` funguje, ale `jp (ix+0)` se odmítne — tato instrukce nemá posun.

Instrukce Z80N

Podporováno je všech 29 oficiálních instrukčních forem Z80N. Fungují také kompatibilní zápisy `MUL DE`, `MUL`, `MIRROR`, `PIXELAD HL` a `PIXELDN HL`. Úplná kontrola kódování je v `docs/assembler-z80n.md`.

Podporovány jsou `NEXTREG n,n`, `NEXTREG n,A`, `MUL D,E`, `SWAPNIB`, `MIRROR A`, `TEST n`, `BSLA/BSRA/BSRL/BSRF/BRLC DE,B`, `ADD HL/DE/BC,A` a `ADD` s okamžitou 16bitovou hodnotou, `PUSH nn`, `OUTINB`, `PIXELDN`, `PIXELAD`, `SETAE`, `JP (C)`, `LDIX`, `LDWS`, `LDDX`, `LDIRX`, `LDPIRX` a `LDDRX`. `PUSH nn` zapisuje okamžitou hodnotu v pořadí horní, dolní bajt. Instrukce `Nextu` jsou dostupné bez hostitelského `OPT --zxnext`.

Další technické podrobnosti jsou v `docs/assembler-binary.md` a `docs/assembler-includes.md` v repozitáři.

11. Když se něco nepovede

PROBLÉM	CO ZKONTROLOVAT
Unknown operand / undefined symbol	Přesnou velikost písmen návěští, překlep, existenci definice a zda je načten její include. Start není START.
Unsupported directive / mnemonic	Podporovanou syntaxí v kapitolách 8 a 10. Není každý příkaz SjASMPlus již implementován.
Chyba v include	Cestu a číslo řádku v reportu. Otevřete tento soubor; údaj se vztahuje k include, ne nutně k hlavnímu souboru.
Include nelze otevřít	Soubor musí existovat na SD; relativní cesta vychází z adresáře vkládajícího souboru.
Příliš dlouhý řádek	V editoru nejvýše 80 znaků; include načítaný přímo assemblerem nejvýše 160.
Chyba rozsahu	DB – 128..255; indexový posun a relativní skok – 128..127; adresy nesmí přetéct přes \$FFFF.
ASSERT selhal	Podmínku a očekávanou velikost či adresu programu. ASSERT se kontroluje ve druhém průchodu.
Binary save failed	Cílovou cestu, volné místo, stav SD a uvedenou operaci / kód esxDOS. Oprava návěští tuto chybu nevyřeší.
Assembly output memory full	Výstup MMU potřebuje volnou 8KB stránku paměti editoru pro každou zapsanou virtuální stránku. Zavřete nepotřebné zdrojové soubory, nebo zmenšete překládaná data.
Překlad dlouho trvá	Sledujte průchod, soubor, řádek a indikátor. Dva průchody znovu čtou includes; větší projekt na nativním Z80 může trvat déle.
All file slots full	Zavřete některý otevřený soubor pomocí File → Close.

PŘI HLÁŠENÍ CHYBY

Uvedte verzi NextForge, hlavní soubor, potřebné includes a binární vstupy, celý text chyby a zda jde o Next nebo CSpect. U chyby ukládání přidejte cílovou cestu a kód esxDOS.

12. Přehled zkratk a tisk

CS = CAPS SHIFT, SS = SYMBOL SHIFT. EXT → znamená stisknout a uvolnit CS + SS a následně stisknout klávesu. Kombinace CS + SS + číslo se naopak drží současně.

AKCE	ZKRATKA
Menu	EXT → 0
Integrovaná nápověda	EXT → H
Otevřít / uložit / uložit jako / zavřít	EXT → O / S / E / W
Otevřené soubory	CS + SS + 1
Pohyb vlevo / dolů / nahoru / vpravo	CS + 5 / 6 / 7 / 8
Stránka nahoru / dolů	CS + SS + 7 / 6
Začátek / konec řádku	SS + Q / E
Smazat před / pod kurzorem	CS + 0 / 9
Odsadit / zformátovat řádek	CS + 3 / SS + W
Označit blok / vybrat vše	CS + 4 / SS + A
Kopírovat / vyjmout / vložit	EXT → C / X / V
Hledat	SS + F
Breakpoint na řádku	EXT → B
Přeložit / přeložit a ladit	EXT → A / R
Otevřít monitor a debugger	EXT → D
Přeložit s testy a spustit je	EXT → T
Sloupec taktů (T-states) za řádky	EXT → K
Skok na definici / okno doplnění návěští	SS + ENTER / CS + ENTER
Potvrdit výběr v popupu / zrušit dialog	ENTER / CS + SPACE
Debugger: menu / nápověda	EXT → 0 / H
Debugger: krok / přes / ven / pokračovat	S / O / Q / C
Debugger: běh na řádek / nastavit PC	T / J
Debugger: breakpoint / podmínka / sledování	B / K / W
Debugger: testy znovu, s breakpointy	EXT → T

Úplný přehled kláves debuggeru je v kapitole 23.

HTML a PDF

Manuál funguje i lokálně bez připojení k internetu. Přepínač CZE/ENG zachová aktuální kapitolu. Pro tisk vyberte jazyk a použijte tisk prohlížeče; rozvržení je připravené pro A4 a přepínač se netiskne.

Hotové soubory: české PDF a anglické PDF. Při úpravách lze obě PDF znovu vytvořit příkazem `py tools/build_manual.py` z kořene repozitáře; je potřeba Chrome nebo Edge.

NextForge 1.2 · MB Maniax · Manuál aktualizován 22. října 2026.

Monitor a debugger

Druhá část popisuje, jak přeložený program prohlížet a ladit přímo na Nextu: zastavit jej před první instrukcí, krokovat, nastavovat breakpointy a podmínky, prohlížet a měnit paměť i registry a sledovat, co program dělá se stránkováním a hardwarem.

The screenshot displays the NextForge debugger interface with the following sections:

- Debugger Run Breakpoints View Help**: PAUSED before instruction PC \$801A. Controls: [S Step] [O Over] [C Continue] [T Run to] [Q Out] [U Reset] [H Help] [Z Slow].
- Focus: CODE**: Origin \$8000, Entry \$8000, Size \$0054.
- Disassembly**:

\$801A	3200C0	LD (\$C000),A
\$801D	013B24	LD BC,\$243B
\$8020	3E14	LD A,\$14
\$8022	ED79	OUT (C),A
\$8024	C9	RET
Frame:		
\$8025	3A3080	LD A,(Phase)
\$8028	C602	ADD A,\$02
\$802A	323080	LD (Phase),A
\$802D	D3FE	OUT (\$FE),A
\$802F	C9	RET
Phase:		
\$8030	00	NOP
Palette:		
\$8031	E3	EX (SP),HL
\$8032	1C	INC E
\$8033	03	INC BC
- Stack**:

\$052	800A	AF E344	AF' 0044
\$054	0000	BC 0000	BC' 0000
\$055	0000	DE 0000	DE' 0000
\$058	0000	HL 8031	HL' 0000
\$05A	0000	IX 0000	SP 8052
\$05C	0000	IY 0000	PC 801A
\$05E	0000	I/R 00/6D	IM1 DI
\$060	0000		
\$062	0000	S Z H P/V N C	
\$064	0000	0 1 0 1 0 0	
\$066	0000		
\$068	0000	MMU slots 0-7 (pages)	
\$06A	0000	-- 0F 0A 0B 04 05 20 01	
\$06C	0000	T +88 -68	
\$06E	0000	Mode SLOW trace	
\$070	0000	NEXTREG 56=20	
\$072	0000	Watch 8030=00	
- Memory hex / ASCII**:

C000	00 00 00 00 00 00 00 00	
C008	00 00 00 00 00 00 00 00	
C010	00 00 00 00 00 00 00 00	
C018	00 00 00 00 00 00 00 00	
C020	00 00 00 00 00 00 00 00	
C028	00 00 00 00 00 00 00 00	
C030	00 00 00 00 00 00 00 00	
- Memory at registers**:

HL	8031	E3 1C 03 00 00 00 00 00
DE	0000	C3 AB 02 00 00 00 00 00
BC	0000	C3 AB 02 00 00 00 00 00
IX	0000	C3 AB 02 00 00 00 00 00
IY	0000	C3 AB 02 00 00 00 00 00
SP	8052	0A 80 00 00 00 00 00 00
PC	801A	32 00 C0 01 3B 24 3E 14
- Source**: line \$001C main.asm
- Breakpoints**: 8020 802A Condition HL<>8030
- Paused**: S step, O over, C continue, T run to, Q step out, J set PC, H help.

Debugger se zastaveným programem: disassembly, zásobník, registry, paměť a náhledy paměti jsou vidět současně.

Kapitoly 13–23. Začněte kapitolou 14 — krátký příklad vás provede vším podstatným.

13. Seznámení s debuggerem

NextForge obsahuje **monitor** a **debugger** v jednom okně. Monitor ukazuje paměť jako hexadecimální výpis i jako disassembly s návěstími, zobrazuje registry, příznaky a zásobník a dovoluje hodnoty měnit. Debugger k tomu přidává řízené spouštění: program běží po jednotlivých instrukcích nebo až k breakpointu a mezitím si prohlédnete, co udělal.

Debugger pracuje s **posledním úspěšným překladem**. Program běží ve vlastní, oddělené paměti; editor, otevřené soubory, historie úprav ani samotný NextForge nemohou být programem přepsány.

Jak debugger otevřít

1. Přeložte hlavní zdroj projektu (**Assemble** → **Assemble**, **EXT** → **A**).
2. Zvolte **Assemble** → **Assemble & Run** (**EXT** → **R**, překlad a hned debugger), nebo **Debug** → **Open Debugger** (**EXT** → **D**). Unit testy spouští **Assemble** → **Run Tests** (**EXT** → **T**) v okně reportu; **D** v tomto okně otevře debugger a spustí je v něm (Část III).
3. Debugger se otevře **zastavený před vstupní instrukcí**: nic ještě neproběhlo, registry jsou vynulované a PC ukazuje na vstup programu.

MENU DEBUG V EDITORU	CO UDĚLÁ
Open Debugger	Otevře debugger s aktivním panelem kódu.
Step Into	Otevře debugger a rovnou provede jednu instrukci.
Breakpoints	Otevře debugger s náповědou k breakpointům ve stavovém řádku.
View Registers / View Memory / Disassembly	Otevře debugger s aktivním panelem registrů / paměti / kódu.

Návrat do editoru

BREAK (CS + SPACE) na zastavené obrazovce nebo **Debugger** → **Return to editor** debugger zavře a obnoví editor. Zastavená relace zůstane uložena: po novém otevření (**EXT** → **D**) pokračujete tam, kde jste skončili. Zahodí ji teprve nový překlad.

CO DEBUGGER POTŘEBUJE

- Úspěšný překlad hlavního zdroje — plochý binární výstup i stránkovaný výstup s MMU / SAVENEX .
- Vstupní adresu v rozsahu \$2000-\$FFFF ; oblast \$0000-\$1FFF patří jádru debuggeru.
- Dost volné paměti pro kopii programu; při nedostatku zavřete nepotřebný soubor a přeložte znovu.

OVLÁDÁNÍ

Vše je dostupné z klávesnice i myši: lišta menu nahoře (**EXT** → **O**), kontextové menu na pravé tlačítko a vyhledávatelná nápověda (**H**). Úplný přehled kláves je v kapitole 23.

14. První ladění krok za krokem

Vyzkoušejte si debugger na krátké rutině. Je v repozitáři jako `tests/fixtures/monitor/monitor.asm`; můžete ji také opsat do prázdného editoru.

```
org $a000
Start:
    ld hl,$1234
    ld de,7
    call AddValue
    ld bc,$0203
    ld ix,$4567
    ld iy,$89ab
    ld ($b100),hl
    ld a,$42
    ld ($b102),a
    ret
AddValue:
    add hl,de
    ret
```

1. **Spuštění.** Zvolte **Assemble** → **Assemble & Run (EXT → R)** a v dialogu potvrďte hlavní soubor. Debugger zastaví na `Start`: zelený řádek je PC `$A000`, nahoře svítí *PAUSED before instruction*.
2. **Krok.** Stiskněte **S**. Proveďte se `LD HL,$1234`, HL se zbarví žlutě (změnil se) a PC se posune. Druhé **S** nastaví DE na 0007.
3. **Krok přes podprogram.** PC stojí na `CALL AddValue`. **O** (Step Over) provede celý podprogram najednou: HL = 123B a PC je za instrukcí CALL.
4. **Restart.** **U** a ENTER vrátí program na vstup a vynuluje registry. Breakpointy a ručně změněné bajty zůstávají.
5. **Breakpoint.** Stiskněte **G**, napište `AddValue` a ENTER — pohled kódu skočí na podprogram. **B** na vybraném řádku přidá breakpoint (červená tečka). **C** spustí program; ten se zastaví na tečce, ještě před `ADD HL,DE`.
6. **Návrat k volajícímu.** **Q** (Step Out) doběhne do konce podprogramu a zastaví za instrukcí CALL.
7. **Paměť.** **M** aktivuje panel paměti, **G B100** ENTER jej přesune. **C** dokončí program: v paměti uvidíte `3B 12 42` a nahoře *RETURNED*.
8. **Podmínka.** **U** ENTER, **X** smaže breakpointy. **K** otevře seznam cílů; stiskněte **1** (pár HL) a napište `=123B` ENTER. **C** zastaví hned za `ADD HL,DE`, kde HL poprvé získá hodnotu 123B.
9. **Zdroj.** **N** zavře debugger a v editoru otevře zdrojový řádek, na kterém stojí pohled kódu. Do debuggeru se vrátíte přes **EXT → D** nebo menu **Debug** — relace pokračuje.

```

Debugger Run Breakpoints View Help PAUSED before instruction PC $A01D
[S Step] [O Over] [C Continue] [T Run to] [Q Out] [U Reset] [H Help] [Z Slow]
Focus: CODE Origin $A000 Entry $A000 Size $001F
Disassembly > cursor ● breakpoint Stack Registers
> A01D AddValue: 1FFA A009 AF 0044 AF' 0044
● A01D 19 ADD HL,DE 1FFC 028A BC 0000 BC' 0000
A01E C9 RET 1FFE 0000 DE 0007 DE' 0000
A01F 00 NOP 2000 0000 HL 1234 HL' 0000
A020 00 NOP 2002 0000 IX 0000 SP 1FFA
A021 00 NOP 2004 0000 IY 0000 PC A01D
A022 00 NOP 2006 0000 I/R 00/3B IM1 EI
A023 00 NOP 2008 0000
A024 00 NOP 200A 0000 S Z H P/V N C
A025 00 NOP 200C 0000 0 1 0 1 0 0
A026 00 NOP 200E 0000
A027 00 NOP 2010 0000 MMU slots 0-7 (pages)
A028 00 NOP 2012 0000 -- 0F 0A 0B 04 05 00 01
A029 00 NOP 2014 0000 T +37 =37
A02A 00 NOP 2016 0000 Mode SLOW trace
A02B 00 NOP 2018 0000 I/O none
A02C 00 NOP 201A 0000 Watch off (W)
Memory hex / ASCII Memory at registers
B100 00 00 00 00 00 00 00 00 ..... HL 1234 00 00 00 00 00 00 00 00
B108 00 00 00 00 00 00 00 00 ..... DE 0007 00 C3 AB 02 00 00 00 00
B110 00 00 00 00 00 00 00 00 ..... BC 0000 C3 AB 02 00 00 00 00 00
B118 00 00 00 00 00 00 00 00 ..... IX 0000 C3 AB 02 00 00 00 00 00
B120 00 00 00 00 00 00 00 00 ..... IY 0000 C3 AB 02 00 00 00 00 00
B128 00 00 00 00 00 00 00 00 ..... SP 1FFA 09 A0 8A 02 00 00 00 00
B130 00 00 00 00 00 00 00 00 ..... PC A01D 19 C9 00 00 00 00 00 00
Source: line $0012 main.asm
Breakpoints: A01D
Stopped at a breakpoint. S steps, C continues, B on this line removes it.

```

Krok 5: program stojí na breakpointu v `AddValue`. Řádek s návěštím předchází instrukci; tečka je u instrukce, zelený řádek je PC a změněné registry (SP, PC) jsou žluté.

15. Obrazovka debuggeru

Všechny panely jsou vidět současně. **Fokus** (aktivní panel) určuje jen to, na co reagují šipky, kolečko, **G** a **E**.

Program zastavený v podprogramu Init: dva breakpointy, podmínka na HL a sledování bajtu Phase.

- 1 Lišta menu a stav relace (PAUSED, TRACING, RETURNED...) s adresou PC.
- 2 Panel nástrojů — tlačítka lze kliknout; [Z Slow/Fast] ukazuje režim běhu.
- 3 Aktivní panel, počátek, vstup a velikost programu; MMU u stránkovaného výstupu.
- 4 Disassembly: adresa, bajty, instrukce, návěští.
- 5 Zásobník od SP: adresa a uložené slovo.
- 6 Registry, I/R, režim přerušení a příznaky.
- 7 Mapa MMU programu, režim, poslední I/O a sledovaný bajt.
- 8 Paměť: osm bajtů na řádek, hex a ASCII.
- 9 Náhledy paměti na adresách z HL, DE, BC, IX, IY, SP a PC.
- 10 Zdrojový soubor a řádek pro adresu v pohledu.
- 11 Breakpointy a podmínka.
- 12 Stavová hláška; při zadávání hodnot řádky 11–12 slouží jako vstup.

Barvy a značky

ZNAČKA	VÝZNAM
Zelený řádek	Instrukce na PC — provede se jako další.
Modrý řádek a >	Kurzor pohledu (vybraný řádek kódu); první řádek paměti, když má paměť fokus.
Červená tečka	Breakpoint na řádku instrukce.
Žlutá hodnota registru	Registr změněný posledním krokem či během.
Barvy instrukcí	Stejně jako v editoru: návěští, instrukce, registry a čísla.

Stav relace

TEXT V LIŠTĚ	VÝZNAM
PAUSED before instruction	Program stojí; instrukce na PC ještě neproběhla.
TRACING	Probíhá pomalé trasování; BREAK jej pozastaví.
RETURNED	Program skončil závěrečným RET. U jej spustí znovu.
IMAGE ONLY	Obraz lze prohlížet, ale ne spustit (vstup pod \$2000 nebo mimo výstup).
NO PROGRAM	Žádný úspěšný překlad.

Panel registrů

Horních šest řádků obsahuje AF, BC, DE, HL, IX, IY, SP, PC a alternativní AF', BC', DE', HL'. Pod nimi je **I/R** (R obsahuje i režii debuggeru) a **IMn EI/DI** — simulovaný režim přerušení a stav jejich povolení. Následují příznaky **S Z H P/V N C**, řádek **MMU slots 0–7** s virtuálními stránkami programu (slot 0 je jádro debuggeru), režim **Mode SLOW/FAST** (**SLOW** +screen se zapnutým L), řádek **I/O** s posledním simulovaným přístupem (například **NEXTREG 56=20** nebo **I/O OUT 02FE=02**) a **Watch** se sledovaným bajtem.

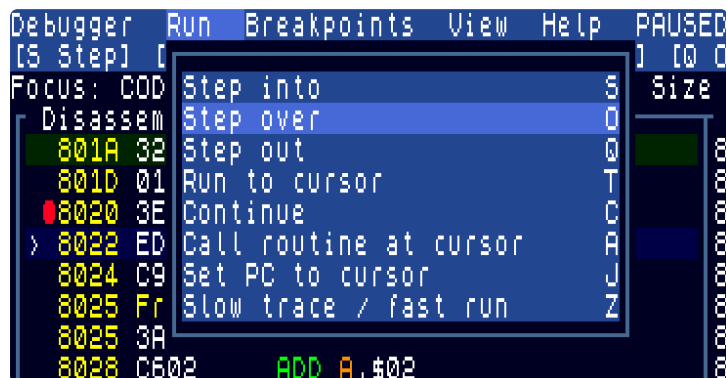
Řádek **T +n =m** počítá takty (T-states) při krokování a pomalém trasování: +n od začátku posledního příkazu (S, O, C, běh na kurzor...), =m od startu relace nebo U. Platí dokumentované časování Z80 a Z80N: provedený skok přidá své takty a opakovaná bloková instrukce jako LDIR se počítá za každé opakování. Rychlý běh takty nepočítá a místo čísla ukáže ?.

Zásobník a náhledy paměti

Panel zásobníku vypisuje 17 slov od SP směrem nahoru — návratové adresy a uložené registry. Náhledy paměti ukazují osm bajtů od adresy v HL, DE, BC, IX, IY, SP a PC; klávesy **1–7** přesunou hlavní panel paměti na tutéž adresu.

16. Menu, kontextová menu a nápověda

Horní řádek je lišta menu **Debugger · Run · Breakpoints · View · Help**. Otevřete ji **EXT** → **0** nebo kliknutím na název menu. Šipky vlevo/vpravo přepínají menu, nahoru/dolů vybírají položku, ENTER nebo kliknutí ji spustí. Písmeno či číslice vpravo u položky ji spustí přímo. BREAK nebo kliknutí mimo menu je zavře.

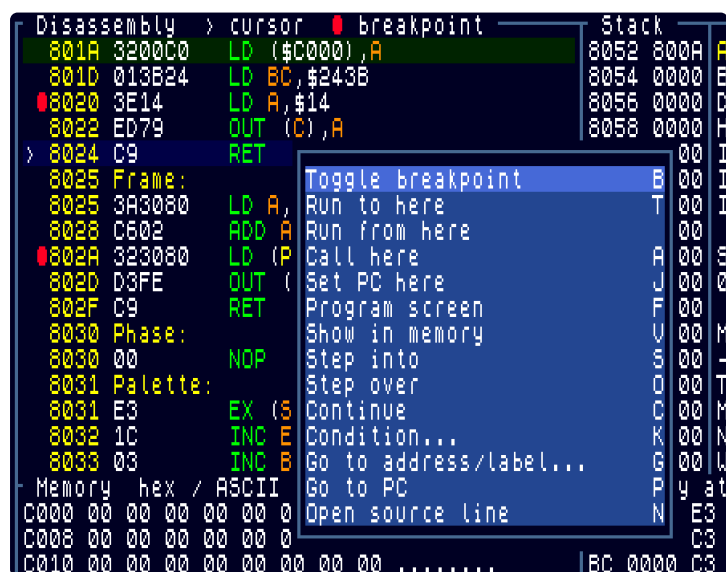


Menu Run s vybranou položkou Step over.

MENU	POLOŽKA	KLÁVESY
Debugger	Return to editor · Open source line · Reset at entry · Set restart entry · Debugger help	BREAK · N · U · I · H
Run	Step into · Step over · Step out · Run to cursor · Continue · Call routine at cursor · Set PC to cursor · Slow trace / fast run	S · O · Q · T · C · A · J · Z
Breakpoints	Toggle breakpoint · Condition · Watch byte change · Clear breakpoints	B · K · W · X
View	Focus code · memory · registers · Next panel · Go to address/label · Go to PC · Program screen · Screen in slow traces · Memory at code cursor · Edit value · Memory at HL, DE, BC, IX, IY, SP, PC	D · M · R · TAB · G · P · F · L · V · E · 1-7
Help	Debugger help · Return to editor	H · BREAK

Kontextová menu

Pravé tlačítko na řádku kódu, bajtu paměti nebo registru nejprve vybere místo pod ukazatelem a pak otevře menu s příkazy, které se k němu hodí. Položku vyberete levým tlačítkem, šipkami a ENTER nebo jejím písmenem; pravé tlačítko či BREAK menu zavře.



Kontextové menu řádku kódu.

PANEL	POLOŽKY
Kód	Toggle breakpoint, Run to here, Run from here, Call here, Set PC here, Program screen, Show in memory, Step into, Step over, Continue, Condition, Go to address/label, Go to PC, Open source line.
Paměť	Edit byte, Watch byte change, Condition, Go to address/label, Memory at PC, Memory at SP.
Registry	Edit register, Condition, Memory at HL, Memory at SP, Go to PC.

Vyhledávatelná nápověda

H (nebo EXT → H) otevře okno nápovědy debuggeru. Funguje stejně jako nápověda editoru: psaním filtrujete popisy i klávesy (bez ohledu na velikost písmen), Backspace maže filtr, šipky, Page Up / Down a kolečko posouvají seznam a BREAK, EXT → H nebo kliknutí na **[Close]** okno zavře. Nápověda obsahuje všechny příkazy i pravidla, jak debugger spouští program.

```

Debugger  Run  Breakpoints  View  Help  PAUSED before instruction  PC $801A
NEXTFORGE DEBUGGER - SEARCHABLE HELP [ Close ]
FUNCTION / CONTEXT                                SHORTCUT / MENU
Continue to breakpoint, watch, condition or RET    C
Condition: pick register/pair/memory, then a value K, = <> <> <= >=
Condition stops at breakpoints; with none, anywhere K; X clears
Clear all breakpoints, editor marks, condition      X

Filter: cond
Matches: 4
Type to filter  UP/DOWN/PGUP/PGDN/WHEEL scroll  BREAK / EXT->H close
S/O step  C continue  T run to  Q step out  EXT->Q menu  H help  BREAK editor

```

Nápověda filtrovaná slovem „cond“.

17. Krokování a běh programu

KLÁVESY	AKCE
S — Step into	Provede jednu instrukci. U CALL a RST vstoupí do podprogramu.
O — Step over	Provede CALL celý, včetně vnořených volání, a zastaví za ním. U ostatních instrukcí funguje jako S.
Q — Step out	Doběhne k volajícímu podle návratové adresy na vrcholu zásobníku. Nejdřív proto vraťte vlastní PUSH. V hlavní rutině běží až k závěrečnému RET.
T — Run to cursor	Běží na vybraný řádek kódu; dřív zastaví na breakpointu, který potká po cestě.
C — Continue	Běží až k breakpointu, splněné podmínce, změně sledovaného bajtu nebo závěrečnému RET.
A — Call routine	Zavolá podprogram na vybraném řádku: aktuální PC uloží na zásobník jako návratovou adresu a běží, dokud se podprogram nevrátí. Registry zůstanou, jak je podprogram nechal.
J — Set PC	Přesune PC na vybraný řádek bez provedení kódu — přeskočí nebo zopakuje instrukce. Registry a zásobník se nemění.
Run from here	Kontextové menu kódu: PC na vybraný řádek a pokračovat (jako J a C).
F / L	Obrazovka programu / obrazovka při pomalém trasování (viz níže).
P — Go to PC	Přesune pohled kódu na PC.
U — Reset	Po potvrzení ENTER obnoví registry, zásobník, stránkování a simulovaný hardware na vstupu. Paměť programu (i ručně změněná) a breakpointy zůstávají; nový obraz načte až nový překlad.
I — Entry	Nastaví vstupní adresu pro restart (hex adresa nebo návěští).

Když program provede závěrečný RET, relace skončí a lišta ukáže *RETURNED*. **U** nebo **S** pak spustí novou relaci od vstupu.

Pomalé trasování a rychlý běh

Pokračování (C), Step Over a Run to cursor mají dva režimy. Přepíná je **Z** nebo tlačítko **[Z Slow]** / **[Z Fast]**.

	POMALÉ TRASOVÁNÍ (VÝCHOZÍ)	RYCHLÝ NATIVNÍ BĚH
Provádění	Instrukce po instrukci, hardware se simuluje.	Program běží naplno přímo na Z80.
DI/EI/HALT, I/O, NEXTREG, MMU	Bezpečně simulované (kapitola 21).	Jdou do skutečného hardwaru.
BREAK	Pozastaví kdykoli, kontrola jednou za snímek.	Zastaví, jen když má program povolená přerušení.
Podmínky a sledování	Ano.	Vždy přepnou na trasování.

	POMALÉ TRASOVÁNÍ (VÝCHOZÍ)	RYCHLÝ NATIVNÍ BĚH
Rychlost	Tisíce instrukcí za sekundu, obrazovka se obnovuje asi 3× za sekundu.	Plná rychlost.
Obrazovka	Panel debuggeru; s L obrazovka programu.	Obrazovka programu.

RYCHLÝ REŽIM

Rychlý běh je skutečný: program ovládá porty, registry Nextu, stránkování i obrazovku. Debugger před během přesune každou stránku programu na fyzické místo podle jejího čísla (i obrazové banky 5 a 7 a obě banky Layer 2), takže stránkování, které si program dělá sám, najde jeho data a displej ukazuje jeho obraz. Po zastavení stránky vrátí, převezme mapu MMU, přečte stav displeje a obnoví své zobrazení. Přesun trvá asi 15 ms na 8KB stránku, proto i krátké O nebo T v rychlém režimu na okamžik ukáže obrazovku programu.

Obrazovka programu

Debugger si vede stín zobrazovacího stavu programu: Layer 2 (banky, rozlišení, scroll, port `$123B`), LoRes, režimy Timex, stínovou obrazovku z banky 7, tilemapu, scroll ULA, sprity a pořadí vrstev, průhlednost, palety, ořezová okna i řízení copperu. Krokování zapisuje do stínu, rychlý běh přímo do hardwaru a debugger si stav po běhu přečte zpět. Barvu borderu a vzory spritů posílá rovnou do hardwaru.

KLÁVESY	CO UVIDÍTE
F — Program screen	Obrazovku programu tak, jak právě je — i po jeho skončení. Libovolná klávesa nebo tlačítko myši vrátí debugger.
L — Screen in slow traces	Pomalé trasování (C, T, Q, O) ukazuje místo panelu obrazovku programu; každá instrukce se projeví hned. BREAK nebo zastavení vrátí panel. Řádek režimu ukazuje <i>Mode SLOW +screen</i> .
Z = Fast	Rychlý běh ukazuje obrazovku programu po celou dobu běhu.

PŘÍKLAD: LORES

Krokujte inicializaci LoRes klávesou **S** a stiskněte **F** — uvidíte, co už program nakreslil. **L** a **C** ukážou kreslení živě, instrukci po instrukci. Po **Z** a **C** program běží plnou rychlostí na své obrazovce, dokud nestisknete BREAK nebo nenarazí na breakpoint.

Hlášky po zastavení

HLÁŠKA	VÝZNAM
Paused: S step...	Krok nebo rychlý běh skončil na breakpointu či cílové adrese.
Stopped at a breakpoint	Trasování dosáhlo breakpointu.
Condition met at breakpoint	Breakpoint se splněnou podmínkou.
Condition became true	Podmínka bez breakpointů se právě stala pravdivou.
Watch changed after instruction at \$xxxx	Sledovaný bajt změnila instrukce na uvedené adrese.
Trace paused by BREAK	Trasování přerušeno klávesou BREAK.
Fast run stopped by BREAK	Rychlý běh zastavený BREAKem; PC je přerušena instrukce.

HLÁŠKA	VÝZNAM
Stopped at RST	Program volal ROM nebo esxDOS (kapitola 22). <code>STUB RST n,label</code> volání nahradí vlastní rutinou (kapitola 26).
Routine returned	Program skončil závěrečným RET.

18. Breakpointy, podmínky a sledování

Breakpointy

B (nebo EXT → B) přepne breakpoint na vybraném řádku kódu, případně na adrese zvolené klávesou **G**. Totéž nabízí kontextové menu kódu. Breakpoint označuje červená tečka u instrukce a adresy všech breakpointů jsou na předposledním řádku obrazovky. Najednou jich může být nejvýše osm, v oblasti `$2000-$FFFF`. **X** smaže všechny breakpointy (i značky v editoru) a podmínku.

Breakpoint patří k adrese procesoru, ne ke konkrétní stránce paměti: platí pro kód, který je na této adrese právě namapovaný.

Breakpointy z editoru

Řádky označené v editoru (EXT → B, kapitola 3) se stanou breakpointy pokaždé, když debugger otevřete — z menu **Debug** (EXT → D) i přes **Assemble & Run** (EXT → R). Breakpointy tak přežijí nový překlad: překlad je smaže, ale značky v editoru je při dalším otevření vrátí.

- **Řádek bez kódu** (návěští, komentář, prázdný řádek) zastaví na první instrukci, která za ním následuje ve stejném souboru.
- **Include soubory**: značky fungují v každém otevřeném souboru, který se do překladu dostal přes `INCLUDE`; soubor se pozná podle cesty.
- **Z debuggeru do editoru**: B na instrukci, kterou začíná zdrojový řádek, označí tento řádek v editoru. Zrušení breakpointu zruší i značku, také značku na návěští nebo komentáři nad ním, která k instrukci vedla. **X** smaže značky ve všech otevřených souborech.
- **Vlastní breakpointy debuggeru**: breakpoint na adrese, kterou žádný řádek nezačíná (uvnitř dat, adresa zadaná přes G), nebo v souboru, který v editoru otevřený není, patří jen debuggeru a při otevření zůstává.
- **Po úpravě zdroje** už mapa řádků z překladu neplatí: debugger si ponechá svůj seznam a nové značky převezme až po dalším překladu.
- Platí limit osmi breakpointů; další značky se nepoužijí.

Podmínky

Podmínka porovná registr nebo paměť s hodnotou. **K** (také **Breakpoints** → **Condition...** a kontextová menu) otevře seznam cílů:



Výběr cíle; předvybraný je současný cíl.

Předvybraný je současný cíl, jinak pár vybraný v panelu registrů, kurzor paměti (fokus na paměti) nebo A. Po výběru napište volitelný operátor a hodnotu — hexadecimální číslo (\$) nepovinné) nebo globální návěští:



Zadání podmínky pro pár HL.

OPERÁTOR	PLATÍ, KDYŽ HODNOTA...	PŘÍKLAD
= (nebo nic)	se rovná	=00, 42
<>, !=	se nerovná	<>4000
<, <=	je menší / nejvýše	<9000
>, >=	je větší / nejméně	>=10

Porovnání je bez znaménka. Bajtové cíle (registry A–L a bajt v paměti) přijímají 00–FF. Podmínka se zobrazí vedle breakpointů, například Condition HL<>4000 nebo Condition byte@B100=FF.

Kdy podmínka zastaví

- **S breakpointy** je podmínka společná pro všechny: program se zastaví na breakpointu jen tehdy, když podmínka platí.
- **Bez breakpointů** se podmínka vyhodnocuje po každé instrukci a program zastaví tam, kde se stane pravdivou. Na řádku je pak (every step). Podmínka, která už platí, zastaví až po změně na nepravdivou a zpět — pokračováním se tak dostanete k dalšímu výskytu.

Sledování bajtu

W začne sledovat vybraný bajt (kurzor paměti nebo adresa vybraná v kódu); opakované W sledování vypne. Po každé instrukci se hodnota porovná a při změně se program zastaví s hláškou *Watch changed after instruction at \$xxxx*, která udává adresu instrukce, jež bajt změnila. Panel registrů ukazuje sledovanou adresu a hodnotu.

CÍL	KLÁVESY
Registry A, B, C, D, E, H, L	A–L
Páry HL, DE, BC	1, 2, 3
IX, IY, SP	4, 5, 6
Alternativní HL', DE', BC'	šipky
Bajt / slovo na kurzoru paměti	M / W
Bez podmínky (zrušit)	X

CO SLEDOVÁNÍ A PODMÍNKY NEZACHYTÍ

Hodnoty se porovnávají vždy po celé instrukci: zápis stejné hodnoty, čtení ani mezikroky blokové instrukce (LDIR) nezachytí. Podmínky i sledování vždy používají pomalé trasování.

19. Monitor: paměť, registry a úpravy

Pohyb a fokus

KLÁVESY	AKCE
TAB / D / M / R	Další panel / kód / paměť / registry. Panel lze aktivovat i kliknutím.
Šipky / kolečko	Kód: výběr řádku a posun pohledu. Paměť: posun o 8 bajtů. Registry: výběr registru.
CS + SS + 7 / 6	Posun pohledu o 256 bajtů.
G	Přesun aktivního pohledu na hex adresu (\$ nepovinné) nebo globální návěští. PC se nemění.
P	Pohled kódu na PC.
1–7	Paměť na adrese z HL, DE, BC, IX, IY, SP, PC.
V	Paměť na adrese vybraného řádku kódu.

Návěští v **G** (i v **K** a **I**) lze psát v libovolné velikosti písmen: přednost má přesná shoda, jinak se použije první návěští, které se liší jen velikostí písmen (`start` najde `Start`). Návěští platí jen do úpravy zdroje; po ní debugger hlásí *Source changed since assembly* a je potřeba znovu přeložit (**EXT** → **A**). Lokální návěští s tečkou se nevyhledávají a jméno, které vypadá jako hex číslo (například `BEEF`), se vezme jako číslo.

Panel paměti

Sedm řádků po osmi bajtech v hex i ASCII (netisknutelné znaky jsou tečky). Kliknutí na bajt jej vybere pro **E**, **W** a **K**. Paměť, kterou program ještě nepoužil, se čte jako 00. Oblast `$0000-$1FFF` během relace ukazuje jádro debuggeru a výchozí zásobník.

Úpravy

E upraví hodnotu v aktivním panelu:

- **Bajt v paměti** — hex 00–FF na adrese `$0700-$FFFF`. Stránka se přidělí při prvním zápisu. Změna sledovaného bajtu nastaví nový výchozí stav sledování.
- **Registr** — slovo vybraného registru, jen u zastaveného programu. PC musí být v `$2000-$FFFF`; SP může být 0 (zásobník od `$FFFF`) nebo nejméně `$0700`. I a R se nemění.

Registr vyberete klávesou **R** a šipkami nebo kliknutím; vybraný registr je zvýrazněný.

Disassembly

Disassembler zná celou sadu Z80 včetně IX/IY, CB a ED a instrukce Z80N (například `NEXTREG`, `MUL D,E`, `PUSH nn`). Neznámé ED kódy ukáže jako `DB`. Adresy, které odpovídají návěští, se v operandech nahradí jeho jménem; samotné návěští má vlastní řádek nad instrukcí. Bajtové hodnoty zůstávají čísla.

20. Zdrojový kód a návěští

Debugger používá tabulku symbolů a mapu zdrojových řádků z posledního úspěšného překladu.

- **Návěští** se zobrazují v disassembly a dají se použít v **G**, **I**, **E** i v podmínkách. Pokud změníte zdroj, nahoře se objeví *Symbols stale*, návěští se skryjí a zadávání jmen nefunguje — přeložte znovu.
- **Řádek Source** ve spodním rámu ukazuje soubor a řádek (hexadecimálně od jedničky) pro adresu v pohledu kódu, případně kurzoru paměti.
- **N** zavře debugger a otevře tento řádek v editoru. Include, který není otevřený, se načte ze SD karty se správným adresářem; otevřený se znovu použije.

MAPA ZDROJE

Překlad uloží až 1024 adresních rozsahů a 31 cest include, včetně vnořených include a opakování REPT. Rozvinuté makro ukazuje na řádek volání. Data, INCBIN i výplň ORG ukazují na řádek, který je vytvořil. Při překročení kapacity je mapa částečná a za jménem souboru je `*`.

TIP

Includes se překládají ze SD karty. Před překladem je uložte, jinak debugger ukáže řádky staré verze.

21. Jak debugger spouští program

Paměť programu

OBLAST	OBSAH BĚHEM RELACE
\$0000-\$1FFF (slot 0)	Jádro debuggeru: RST vektory, tabulka přerušení, kód pro zachycení stavu a výchozí zásobník (\$1FFC). Data lze zapisovat od \$0700.
\$2000-\$FFFF (sloty 1–7)	Soukromé stránky programu: kód, data, zásobník i breakpointy.

Každá 8KB stránka, kterou program použije, má v debuggeru vlastní kopii. Editor, dokumenty ani kód NextForge proto program nepřepíše.

Počáteční stav

- **Vstup** je startovní adresa ze SAVENEX OPEN, jinak první vygenerovaný bajt. Změníte jej klávesou I.
- **Zásobník** je ze SAVENEX, jinak \$1FFC v jádru. Na vrcholu je návratová adresa, takže závěrečný RET relaci ukončí.
- **Registry** jsou nulové, příznaky jako po XOR A, přerušení povolena v IM 1 — jako po spuštění z BASICu.
- **Stránkování**: plochý výstup používá mapu 14, 15, 10, 11, 4, 5, 0, 1; NEX mapu nahrávače (banky 5, 2 a vstupní banka, ROM ve slotech 0/1); stránkovaný výstup bez SAVENEX konečnou mapu z direktiv MMU.

Simulace hardwaru

Krokování a pomalé trasování nikdy neposílají hardwarové instrukce do Nextu:

INSTRUKCE	KROK A POMALÉ TRASOVÁNÍ	RYCHLÝ BĚH
DI, EI	Mění simulovaný stav přerušení.	Nativně.
IM 0/1/2, LD I,A, LD R,A, LD A,I	Simulované.	Nativně (IM se nesleduje).
HALT	Dokončí se; v IM 2 vstoupí do obsluhy programu přes vektor I×256+\$FF. Po DI zastaví s vysvětlením.	Čeká na přerušení.
RETI, RETN	Návrat jako RET.	Nativně.
NEXTREG \$50–\$57, porty \$243B/\$253B	Přepnou virtuální stránku; čtení \$253B vrátí simulovanou hodnotu MMU.	Skutečné MMU.
Porty \$7FFD, \$DFFD	Přepnou banku na \$C000.	Skutečné stránkování.
Zobrazovací NEXTREG a porty (Layer 2, LoRes, Timex, tilemapa, palety, sprity, copper, ořez, scroll)	Jdou do stínu obrazovky programu (kapitola 17); s L i do hardwaru. Border a vzory spritů jdou rovnou do hardwaru.	Skutečný hardware.
Jiné OUT, NEXTREG, OUTI/OTIR, OUTINB	Zaznamenají se do řádku I/O, do hardwaru se nepošlou.	Skutečný hardware.
IN, INI/INIR	Čtení \$253B a \$123B vrátí hodnotu ze stínu; jinak \$FF (žádná klávesa), Kempston 0.	Skutečný hardware.
RST \$00–\$38	Zastaví s vysvětlením — ROM ani esxDOS nejsou k dispozici. RST se STUB zavolá svou náhradu.	Zastaví s vysvětlením. RST se STUB skočí na svou náhradu.
JP (C)	Odmítne se.	Nativně.

Stránkovaný výstup MMU a NEX

Když zdroj používá direktivu `MMU`, stránky výstupu z překladu převezme debugger. Program pak v debuggeru přepíná stejné stránky, jaké vytvořil assembler, a řádek **MMU slots** ukazuje, která virtuální stránka je ve kterém slotu. Reset (U) vrátí mapu na počáteční stav.

Rychlý běh

V rychlém režimu jádro nejdřív vymění stránku: každá stránka programu (kromě stránky 30 s jádrem), obrazové banky 5 a 7 i banky Layer 2 se dostanou na fyzické místo podle svého čísla a stránky NextForge zatím čekají v kopiích programu. Pak jádro nastaví zobrazovací registry a takt procesoru programu, vrátí mu registry a předá procesor. Má-li program povolená přerušení, prochází každé přerušení snímku jádrem, které kontroluje BREAK; program s vlastním IM 2 používá svou obsluhu. Po zastavení jádro přečte mapu MMU (stránkování, které program provedl, debugger převezme) a vymění stránku zpět; debugger přečte registry displeje, ořezová okna a paletu a obnoví textový displej, sprity, Layer 2, copper a takt procesoru.

Stránky paměti a NextForge

Mapa stránek se týká jen programů laděných v debuggeru. Program spuštěný mimo NextForge — po resetu počítače z prohlížeče NextZXOS nebo z BASICu — už NextForge v paměti nemá a může používat libovolné stránky podle pravidel NextZXOS.

KRÁTCE

V debuggeru může program používat téměř libovolná čísla stránek 0–223, i ta, která používá NextForge: debugger mu je dá jako soukromé kopie a v rychlém běhu je vymění na místo. Vyhněte se jen stránce 30 a slotu 0 a stránky, které program mapuje sám, nechte debugger poznat dřív, než je program použije v rychlém běhu.

Debugger stránku zná, když:

- do ní assembler zapsal výstup (`ORG` , `MMU` , `SAVENEX BANK`),
- je namapovaná v některém slotu, nebo ji program namapoval při krokování či pomalém trasování,
- je to obrazová stránka: banka 5, první polovina banky 7 a buffery Layer 2, jakmile je program použije.

Takovou stránku program bezpečně používá pod jejím číslem. Pozor na:

- **Stránku 30** (dolní polovina 16KB banky 15) — jádro debuggeru. Rychlý běh ji nevymění na místo, takže program, který ji sám namapuje, přepíše jádro.
- **Slot 0** (`$0000–$1FFF`) — během relace patří jádru.
- **Stránky, které program poprvé namapuje až v rychlém běhu** — číslo spočítané za běhu, buffer mimo výstup assembleru nebo Layer 2 zapnutá až v rychlém běhu. Ty jsou fyzické: patří-li NextForge, program přepíše jeho kód, data nebo otevřené dokumenty a debugger jejich obsah po zastavení nezná. Například výchozí Layer 2 z NextZXOS (banky 9–11, stránky 18–23) zahrnuje stránku 22 debuggeru. Takové stránky namapujte jednou při krokování nebo pomalém trasování, nebo je dejte do výstupu.
- **Množství paměti** — kopie stránek programu berou místo ze sdíleného poolu spolu s dokumenty; když dojde, debugger hlásí *Not enough free memory*. Rychlý běh vymění nejvýše 96 stránek.

Fyzická mapa stránek

STRÁNKY	BANKA 16K	OBSAH
0–1	0	Hlavní kód NextForge.
10–11	5	NextZXOS: systémové proměnné a obrazovka ULA. NextForge je zálohuje do 24–25.
14	7 (dolní)	Tilemapa a font 80sloupcového textu.
15	7 (horní)	NextZXOS: prohlížeč souborů.
22	11 (dolní)	Debugger: uložená obrazovka editoru a paleta programu.
24–25	12	Záloha stránek 10–11.
26	13	Proměnné, zásobník a tabulka přerušení IM 2.
27, 31	13, 15	Záznamy tabulky symbolů (další si bere z poolu).
28–30	14–15	Kód debuggeru (28–29) a jeho jádro (30).
32–39	16–19	Výstup assembleru (64 KB).
40–41	20	Makra a dopředné EQU.
42–43	21	Hledání souborů include, přenos breakpointů z editoru, běh unit testů a sloupec taktů v editoru.
44–45	22	Kód tabulky symbolů a výrazů.
46–47	23	Nápověda a kód obrazovky programu; prázdná textová stránka.
48–49	24	Další kód editoru a assembleru.
50–51	25	Zásobník include; menu a nápověda debuggeru.
52–55	26–27	Assembler, rozšíření editoru a proměnné debuggeru.
56–95 (1 MB)	28–47	Sdílený pool: text dokumentů, historie úprav, stránkovaný výstup MMU, větší tabulka symbolů a kopie stránek laděného programu.
56–175 (2 MB)	28–87	

Stránky 2–9, 12–13, 16–21 a 23 (na 2MB Nextu i 176–223) NextForge nepoužívá. Pro stránky, které program mapuje až v rychlém běhu, ale nejsou automaticky bezpečné: NextZXOS si stránky 0–23 rezervuje pro sebe (systém a výchozí Layer 2) a nejvyšší stránky dostávají nainstalované ovladače.

Kolik paměti NextForge zabírá

Soubor `NEXTFORGE.NEX` má 176,5 KB: hlavičku a 11 bank po 16 KB s kódem a daty. Za běhu NextForge pevně používá 36 stránek (288 KB: stránky 0–1, 14, 22 a 24–55). K tomu si vyhradí sdílený pool — na Nextu s 1 MB 40 stránek (320 KB), s 2 MB 120 stránek (960 KB) — a stránky z něj přiděluje podle potřeby. Celkem je to 608 KB ze 768 KB, které má 1MB Next pro programy, nebo 1 248 KB z 1 792 KB u 2MB Nextu. Kopie stránek laděného programu bere debugger z poolu.

22. Omezení a řešení problémů

HLÁŠKA NEBO SITUACE	PŘÍČINA	CO UDĚLAT
NO PROGRAM / No program image	Žádný úspěšný překlad.	Přeložte hlavní zdroj.
IMAGE ONLY	Vstup pod <code>\$2000</code> nebo na stránce bez výstupu.	Změňte ORG nebo vstup (I).
Not enough free memory	Plný sdílený pool stránek.	Zavřete nepotřebný soubor a přeložte znovu.
Cannot step here	PC pod <code>\$2000</code> , JP (C) nebo příliš mnoho prefixů.	J na platný řádek, nebo U.
RST ... calls ROM/esxDOS / Stopped at RST	Program volá ROM nebo esxDOS.	J volání přeskočí; ROM služby debugger nenabízí.
EXPECT failed / FAIL n/N ...	Kontrola unit testu neplatila.	C pokračuje za kontrolou; viz kapitola 27.
HALT with interrupts disabled	HALT po DI by nikdy neskončil.	Přidejte EI, nebo J.
Breakpoint rejected	Adresa pod <code>\$2000</code> nebo už je jich osm.	Odeberte jiný breakpoint.
Watch/condition needs...	Program nestojí nebo adresa pod <code>\$0700</code> .	U ENTER, jiná adresa.
Not accepted	Neplatné číslo či návěští, nebo hodnota nad FF pro bajt.	Opravte zadání.
Step out needs...	Na vrcholu zásobníku není návratová adresa.	Krokujte k RET, nebo T.
Symbols stale / Source changed since assembly	Zdroj byl změněn po překladu; návěští a zdrojové řádky nejsou k dispozici.	Přeložte znovu (EXT → A).
No source range	Adresa nepochází ze zdroje (prázdná paměť).	—
Rychlý běh nejde zastavit	Smyčka po DI bez breakpointu.	Reset počítače; takové části ladíte trasováním.
BREAK zavřel debugger	BREAK na zastavené obrazovce opouští debugger.	Otevřete Debug znovu — relace pokračuje.
Obrazovka programu je černá	Program ještě nic nenakreslil, nebo kreslí do stránky mimo zobrazení.	Krokujte dál a použijte F; u Layer 2 zkontrolujte NEXTREG <code>\$12</code> .
Rychlý běh bliká obrazovkou	Každý rychlý běh (i O a T) ukáže obrazovku programu.	Krokujte v pomalém režimu (Z).

Omezení

- Nejvýše osm breakpointů, jedna podmínka a jeden sledovaný bajt.
- Značky breakpointů v editoru se neukládají do souboru.
- Breakpointy patří k adrese procesoru, ne ke stránce paměti.
- Sledování a podmínky porovnávají stav po celé instrukci, nesledují sběrnici.
- Atributy spritů nejdou přecházet zpět: při návratu do debuggeru se sprity programu skryjí (vzory zůstanou, kromě vzoru 63 pro ukazatel myši). Program je obvykle při dalším snímku nastaví znovu.
- Copper se při zobrazení debuggeru zastaví; znovu se spustí s posledním NEXTREG `$62` zapsaným při krokování.
- Stránkování Layer 2 portem `$123B` (zápis do `$0000–$3FFF`) funguje jen v rychlém běhu; při krokování mapujte Layer 2 přes MMU.
- Živá obrazovka při trasování (L) ukáže stránky 2–25 a 32–39 (banky 5 a 7, Layer 2 v bankách 1–12 a 16–19); zbytek ukáže F nebo rychlý běh. Stránku 30 (jádro) zobrazit nelze.

- ROM a esxDOS nejsou dostupné (volání přes RST nahradí `STUB`, kapitola 26); IN vrací `$FF`, program tedy nečte klávesnici.
- Slot 0 (`$0000-$1FFF`) patří jádru; zápis programu tam může jádro poškodit.
- Stránku 30 program používat nemůže; stránky, které poprvé namapuje až v rychlém běhu, jsou fyzické (viz Stránky paměti a NextForge).
- Registr R obsahuje i režii debuggeru. Změnu IM v rychlém režimu debugger nesleduje; IM 2 pozná podle změny registru I.
- Na samomodifikující kód pozor: nesmí přepsat aktivní breakpoint.

23. Přehled kláves debuggeru

KLÁVESY / MYŠ	AKCE
EXT → 0 / klik na řádek 0	Lišta menu.
Pravé tlačítko	Kontextové menu kódu, paměti nebo registru.
Levé tlačítko / kolečko	Výběr řádku, registru nebo bajtu / posun aktivního panelu.
S / O / Q	Krok do / přes CALL / ven k volajícímu.
C / T	Pokračovat / běžet na vybraný řádek.
A	Zavolat podprogram na vybraném řádku.
F / L	Obrazovka programu / obrazovka při pomalém trasování.
J / P	Nastavit PC na vybraný řádek / pohled na PC.
U, ENTER / I	Reset na vstupu / zvolit vstup.
Z	Pomalé trasování / rychlý běh.
B nebo EXT → B	Přepnout breakpoint (i značku řádku v editoru).
K	Podmínka: cíl, operátor, hodnota.
W	Sledovat / přestat sledovat bajt.
X	Smazat breakpointy, značky v editoru a podmínku.
TAB / D / M / R	Další panel / kód / paměť / registry.
Šipky / CS + SS + 7, 6	Výběr a posun / o 256 bajtů.
G	Na adresu nebo návěští.
1–7 / V	Paměť na HL, DE, BC, IX, IY, SP, PC / na kurzoru kódu.
E	Upravit bajt nebo registr.
N	Otevřít zdrojový řádek v editoru.
H nebo EXT → H	Vyhledávatelná nápověda.
EXT → T	Spustit unit testy znovu, s breakpointy (Část III).
BREAK	Pozastavit trasování, zavřít menu, opustit debugger.

Unit testy

Třetí část ukazuje, jak rutiny v assembleru ověřovat automatickými testy přímo na Nextu: k rutinám napíšete testy s očekávanými hodnotami registrů, příznaků a paměti a jedna klávesa projekt přeloží, testy spustí a v okně reportu vypíše, které prošly a které ne.

```

NextForge 1.2 by MB Maniax
Arithmetic and memory routines with their unit tests in arithmetic.tst.
; Assemble (EXT->A) builds the routines alone; Run Tests (EXT->T) also
; takes in the TESTCLUDE file at the end and runs its tests.
; Each routine states what it takes, what it returns and what it keeps:
; exact
;
; HL =
; add16:
; A =
; add8sa
; HL =
; The
; sub16:
; Copi
; BC = 0 copies nothing (a plain LDIR would copy 65536 bytes).
; memmove:
; ld a,b
; or c
; ret z
; push hl
;
; arithmetic.asm Ln 47 Col 17 Len 41 Lines 48 Free 294912B

Assembly report done

; HL =
; add16:
; A =
; add8sa
; HL =
; The
; sub16:
; Copi
; BC = 0 copies nothing (a plain LDIR would copy 65536 bytes).
; memmove:
; ld a,b
; or c
; ret z
; push hl
;
; arithmetic.asm Ln 47 Col 17 Len 41 Lines 48 Free 294912B

01 add16: 1000 + 234 68 T
02 add16: overflow sets carry 68 T
03 add8sat: below the limit 56 T
04 add8sat: stops at 255 67 T
05 sub16: 1234 - 234 72 T
06 sub16: equal numbers give zero 72 T
07 sub16: borrow when DE is bigger 72 T
08 sub16: ignores the carry flag: HL=#0006, expected #0007 72 T
09 memmove: copies 8 bytes 596 T
10 memmove: zero length copies nothing 96 T
11 memmove: overlap, move up by 2 404 T
12 memmove: overlap, move down by 2 357 T

Result: 1 of 12 tests failed.

UP/DOWN/WHEEL scroll ... D debugger ENTER/ESC/CLICK close

```

Okno Run Tests: jedenáct testů prošlo (zelená fajfka), test 8 selhal (červený křížek) a výsledek je červený.

Kapitoly 24–28. Začněte kapitolou 25 — příklad se sčítáním, odčítáním a přesunem paměti ukáže vše podstatné.

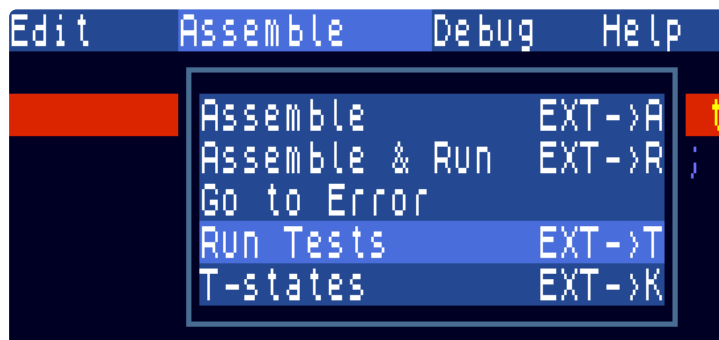
24. Seznámení s unit testy

Unit test je krátký kus kódu, který zavolá jednu rutinu s připravenými vstupy a ověří její výsledek. Když rutinu později upravíte, stačí testy spustit znovu a hned vidíte, jestli pořád dělá, co má — i v případech, které byste ručně nezkoušeli: přetečení, nulová délka, překrývající se paměť.

NextForge testy spouští **přímo na Nextu**, ve stejném izolovaném prostředí jako debugger z Části II: testovaný kód běží ve vlastní paměti, takže editor ani otevřené dokumenty nepřepíše, a kontrolují se skutečné registry, příznaky a paměť procesoru Z80N.

Jak to funguje

1. Testy píšete mezi `TESTCASE "jméno"` a `ENDTEST`, obvykle do samostatného souboru s příponou `.tst`.
2. Uvnitř testu nastavíte vstupy běžnými instrukcemi, zavoláte rutinu a direktivou `EXPECT` popíšete, co má platit: hodnotu registru, příznak nebo obsah paměti.
3. Hlavní soubor projektu vloží testy direktivou `TESTCLUE "soubor.tst"`. Běžný překlad ji přeskočí, takže výsledný program testy neobsahuje.
4. **Assemble → Run Tests (EXT → T)** přeloží projekt i s testy a spustí je na pozadí. Okno reportu místo kroků překladu vypíše všechny testy: zelená fajfka prošel, červený křížek selhal (i s důvodem). Řádek *Result* je zelený, nebo červený.
5. Po zavření okna mají řádky `TESTCASE` v editoru barvu podle výsledku a stavový řádek ukazuje souhrn. Klávesa **D** v okně otevře debugger, který testy spustí znovu a zastaví se u prvního selhání.



Run Tests v menu Assemble (EXT → T).

CO JE TEST

Každý blok `TESTCASE ... ENDTEST` je podprogram: `ENDTEST` vloží `RET`. Testy se volají s vynulovanými registry, vlastním zásobníkem a pamětí ve stavu před prvním testem, jako by se program spouštěl znovu. Kontrola `EXPECT` je v kódu instrukce `RST $30` s několika bajty dat; debugger ji pozná a vyhodnotí jako jeden krok.

PROČ PŘÍMO NA NEXTU

Testy běží na skutečném procesoru. Neúspěšný test hned prohlédnete v debuggeru, který už znáte, odkrojujete a opravíte. Nepotřebujete PC ani další nástroje.

25. První testy krok za krokem

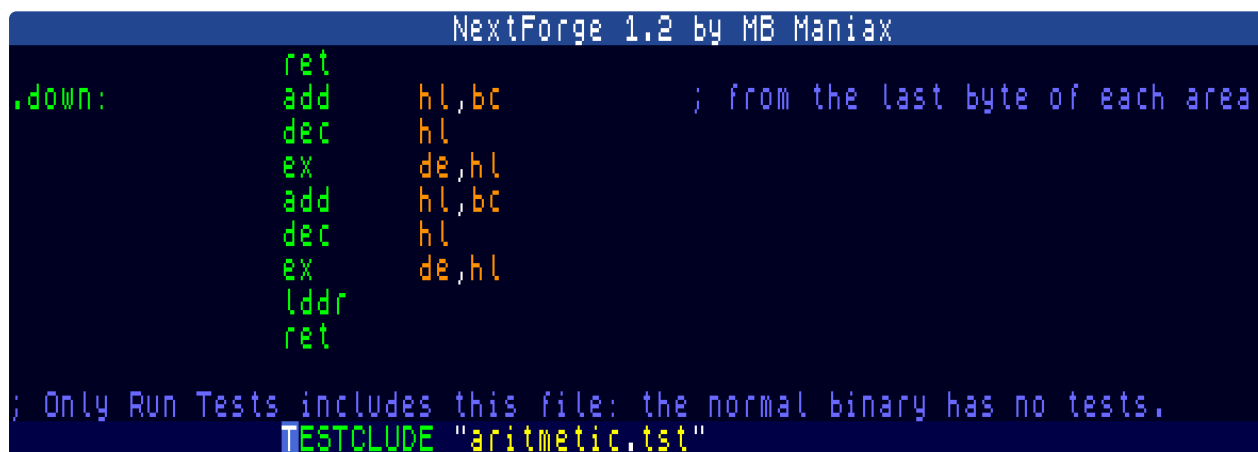
Příklad je v repozitáři v adresáři `tests/fixtures/unittest:` `arithmetic.asm` s rutinami a `arithmetic.tst` s jejich testy. Každá rutina má v komentáři, co bere, co vrací a co zachovává — přesně to testy ověřují:

```
org      $8000

; HL = HL + DE. Carry set when the sum overflows 16 bits. Keeps DE.
add16:   add     hl,de
        ret

; HL = HL - DE. Carry set when DE was bigger, Z when the result is 0.
; The carry flag on entry does not matter. Keeps DE.
sub16:   or      a                ; SBC also subtracts the carry
        sbc     hl,de
        ret

...
; Only Run Tests includes this file: the normal binary has no tests.
TESTCLUDE "arithmetic.tst"
```



NextForge 1.2 by MB Maniax

```
.down:   ret
        add     hl,bc           ; from the last byte of each area
        dec     hl
        ex      de,hl
        add     hl,bc
        dec     hl
        ex      de,hl
        lddr
        ret

; Only Run Tests includes this file: the normal binary has no tests.
TESTCLUDE "arithmetic.tst"
```

Konec `arithmetic.asm`: `TESTCLUDE` vloží testy jen při Run Tests.

Soubor dále obsahuje `add8sat` (sčítání omezené na 255) a `memmove` (přesun paměti, i do překrývající se oblasti). `arithmetic.tst` začíná řádkem `include "arithmetic.asm"` a má 12 testů; tři z nich pro odčítání:

```

TESTCASE "sub16: 1234 - 234"
ld      hl,1234
ld      de,234
call    sub16
EXPECT  HL,1000
EXPECT  NC
ENDTEST

TESTCASE "sub16: borrow when DE is bigger"
ld      hl,1
ld      de,2
call    sub16
EXPECT  HL,-1          ; $FFFF
EXPECT  C
ENDTEST

TESTCASE "sub16: ignores the carry flag"
scf                                ; a carry left over by the caller
ld      hl,10
ld      de,3
call    sub16
EXPECT  HL,7
ENDTEST

```

A dva pro přesun paměti: kopie nesmí zapsat za konec cíle a přesun do překrývajících se oblastí nesmí data „rozmazat“. Test si vstupní data připraví sám — hodnoty 1–8 zapíše smyčkou, buffer naplní instrukcí LDIR — takže je z testu hned vidět, s čím rutina pracuje. Co zapsal předchozí test, se před dalším testem stejně vrátí (kapitola 27).

```

source:      ds      8          ; the tests fill it themselves
target:      ds      8
guard:       db      $aa        ; a copy must never reach it
...
                TESTCASE "memmove: copies 8 bytes"
                ld      hl,source    ; prepare: source = 1,2,...,8
                ld      a,1
                ld      b,8
.fill:        ld      (hl),a        ; .fill is local to this test
                inc     hl
                inc     a
                djnz    .fill
                ld      hl,source
                ld      de,target
                ld      bc,8
                call    memmove
                EXPECT  (target),1,2,3,4,5,6,7,8
                EXPECT  (guard),$aa    ; nothing written past the end
                ENDTST

                TESTCASE "memmove: overlap, move up by 2"
                ld      hl,letters    ; prepare: buffer = ABCDE
                ld      de,buffer
                ld      bc,5
                ldir
                ld      hl,buffer
                ld      de,buffer+2
                ld      bc,5
                call    memmove
                EXPECT  (buffer),"ABABCDE" ; LDIR alone gives ABABABA
                ENDTST

```

1. **Běžný překlad.** Otevřete `arithmetic.asm` (**EXT** → **O**) a přeložte jej (**EXT** → **A**). Binárka obsahuje jen rutiny; `arithmetic.tst` se vůbec nečte.
2. **Spuštění testů.** Stiskněte **EXT** → **T** a potvrďte hlavní soubor (stejně dobře můžete **EXT** → **T** stisknout v `arithmetic.tst`, viz kapitola 26). Po překladu se testy spustí přímo v okně reportu: místo kroků překladu se objeví 12 řádků se zelenými fajfkami, vpravo s takty každého testu, a zelený výsledek *All 12 tests passed*. **ENTER** okno zavře.
3. **Chyba v rutině.** V `sub16` smažte `or a`. Takovou chybu ruční zkouška snadno přehlédne: dokud je carry náhodou nulový, výsledek sedí.
4. **Nové spuštění.** **EXT** → **T**. Okno ukáže u testu 08 červený křížek a důvod *HL=\$0006, expected \$0007 — SBC odečetl i přenos, který zůstal nastavený z volajícího kódu*. Ostatní testy proběhly také a prošly; výsledek *1 of 12 tests failed*. je červený.
5. **Výsledek v editoru.** **ENTER** okno zavře. Stavový řádek ukazuje červeně *1 of 12 tests failed*. a v `arithmetic.tst` je řádek `TESTCASE` testu 8 červený, ostatní zelené. Najedte na červený řádek kurzorem nebo myši: stavový řádek ukáže důvod *FAIL sub16: ignores the carry flag: HL=\$0006, expected \$0007*. Na zeleném řádku ukáže zeleně takty testu, třeba *PASS sub16: borrow when DE is bigger: 72 T*.
6. **Prohlídka v debuggeru.** Spustěte testy znovu a v okně stiskněte **D**. Debugger testy spustí a zastaví se na kontrole testu 8: kurzor kódu na `RST $30`, PC hned za ní, HL = 0006, řádek *Source* ukazuje `EXPECT HL,7` a dolní řádek červeně *FAIL 8/12* **N** otevře řádek v editoru, **U** **ENTER** test restartuje a můžete jej krokovat (**S**).
7. **Oprava.** Vraťte `or a` a znovu **EXT** → **T**: všech 12 testů projde a řádky zezelenají.

```

NextForge 1.2 by MB Maniax
Arithmetic and memory routines with their unit tests in arithmetic.tst.
Assemble (EXT->A) builds the routines alone; Run Tests (EXT->T) also
takes in the TESTCLUDE file at the end and runs its tests.
Each routine states what it takes, what it returns and what it keeps:
exac

Assembly report done

; HL =
add16:  ✓ 01 add16: 1000 + 234          68 T
        ✓ 02 add16: overflow sets carry 68 T
        ✓ 03 add8sat: below the limit  56 T
        ✓ 04 add8sat: stops at 255     67 T
        ✓ 05 sub16: 1234 - 234         72 T
add8sa:  ✓ 06 sub16: equal numbers give zero 72 T
        ✓ 07 sub16: borrow when DE is bigger 72 T
        ✗ 08 sub16: ignores the carry flag: HL=$0006, expected $0007 596 T
        ✓ 09 memmove: copies 8 bytes 96 T
        ✓ 10 memmove: zero length copies nothing 96 T
        ✓ 11 memmove: overlap, move up by 2 404 T
        ✓ 12 memmove: overlap, move down by 2 357 T

Result: 1 of 12 tests failed.

UP/DOWN/WHEEL scroll  D debugger  ENTER/ESC/CLICK close

; Copi
BC = 0 copies nothing (a plain LDIR would copy 65536 bytes).
memmove:  ld      a,b
          or      c
          ret     z
          push   hl

```

Krok 4: test 8 selhal i s důvodem, ostatní prošly.

```

Debugger Run Breakpoints View Help PAUSED before instruction PC $8193
[Is Step] [To Over] [IC Continue] [IT Run to] [IO Out] [IU Reset] [IH Help] [IZ Slow]
Focus: CODE Origin $8000 Entry $8184 Size $02A6
Disassembly > cursor breakpoint
Registers
> 818E F7 RST $30 1FFC 028A AF 0002 AF' 0044
818F 54 LD D,H 1FFE 0000 BC 0000 BC' 0000
8190 85 ADD A,(HL) 2000 0000 DE 0003 DE' 0000
8191 07 RLC 2002 0000 HL 0005 HL' 0000
8192 00 NOP 2004 0000 IX 0000 SP 1FFC
8193 C9 RET 2005 0000 IY 0000 PC 8193
8194 6D LD L,L 2008 0000 I/R 00/22 IM1 EI
8195 65 LD H,L 200A 0000
8196 6D LD L,L 200C 0000 S Z H P/V N C
8197 6D LD L,L 200E 0000 0 0 0 0 1 0
8198 6F LD L,A 2010 0000
8199 76 HALT 2012 0000 MMU slots 0-7 (pages)
819A 65 LD H,L 2014 0000 -- 0F 0A 0B 04 05 00 01
819B 3A2063 LD A,($320) 2016 0000 T +66 +66
819C 6F LD L,A 2018 0000 Mode SLOW trace
819D 70 LD (HL),B 201A 0000 I/O none
819E 69 LD L,C 201C 0000 Watch off (W)
Memory hex / ASCII
8020 00 00 00 00 00 00 00 00 ..... HL 0005 00 00 C3 AB 02 00 00 00
8028 00 00 00 00 00 00 00 00 ..... DE 0003 00 00 00 00 00 C3 AB 02
8030 AA 41 42 43 44 45 00 00 .ABCDE.. BC 0000 C3 AB 02 00 00 00 00 00
8038 00 00 00 00 00 00 61 64 .....ad IX 0000 C3 AB 02 00 00 00 00 00
8040 64 31 36 3A 20 31 30 30 d16: 100 IY 0000 C3 AB 02 00 00 00 00 00
8048 30 20 2B 20 32 33 34 00 0 + 234, SP 1FFC 8A 02 00 00 00 00 00 00
8050 21 E6 03 11 EA 00 CD 00 !..... PC 8193 C9 6D 65 6D 6D 6F 76 65
Source: line $0055 /Project/arithmetic.tst
Breakpoints: none

```

Krok 6: debugger na kontrole testu 8; kurzor na kontrole, PC za ní, HL = 0006.

```

; Copies BC bytes from HL to DE; the two areas may overlap.
; BC = 0 copies nothing (a plain LDIR would copy 65536 bytes).
memmove:  ld      a,b
          or      c
          ret     z
          push   hl

1 of 12 tests failed.

```

Krok 5: souhrn ve stavovém řádku editoru po zavření okna.

ZKUSTE I DALŠÍ CHYBU

Smažte v `memmove` řádek `jr c, .down`, takže rutina vždy kopíruje dopředu instrukcí LDIR. Test *11 memmove: overlap, move up by 2* pak hlásí `($8039)=$41, expected $43`: místo `ABABCDE` vznikne `ABABABA`.

26. Direktivy pro testy

DIREKTIVA	VÝZNAM
TESTCASE "name"	Začátek testu. Jméno je nepovinné; uloží se do výstupu před kód testu a objeví se ve výsledcích. Testy nelze vnořovat.
ENDTEST	Konec testu; vloží <code>RET</code> . Každý TESTCASE musí mít svůj ENDTEST.
EXPECT reg,value	Registr nebo pár musí mít danou hodnotu. Hodnota je výraz; u bajtových registrů –128 až 255.
EXPECT cc	Podmínka nad příznaky musí platit: NZ, Z, NC, C, PO, PE, P nebo M — stejně jako u <code>JP cc</code> .
EXPECT (address),bytes	Paměť od adresy musí obsahovat bajty zapsané jako v DB: čísla, znaky a řetězce oddělené čárkami, nejvýše 255 bajtů.
EXPECT CYCLES op n	Takty od začátku testu nebo od předchozí kontroly CYCLES musí splnit porovnání <code>= <> < <= > >=</code> s n (do 65535), například <code>EXPECT CYCLES <= 300</code> . Test s touto kontrolou běží vždy pomalým trasováním, i když má debugger nastavený rychlý běh (kapitola 27); samotné kontroly takty nemají.
STUB RST n,label	V debuggeru zavolá RST n místo ROM nebo esxDOS rutinu <code>label</code> ; n je <code>\$08</code> , <code>\$10</code> , <code>\$18</code> , <code>\$20</code> , <code>\$28</code> nebo <code>\$38</code> . Do výstupu nic nevloží (viz Náhrada ROM a esxDOS níže).
MOCK routine,fake	Do konce testu skočí každé volání rutiny <code>routine</code> do podvrhu <code>fake</code> ; pak se rutina vrátí (viz Podvrh vlastní rutiny níže).
TESTCLUDE "file"	Při Run Tests stejné jako INCLUDE, při ostatních překladech se přeskočí. Patří do hlavního souboru projektu.

Registry a podmínky

SKUPINA	NÁZVY
Bajty	A, F, B, C, D, E, H, L, IXH, IXL, IYH, IYL, I
Páry	AF, BC, DE, HL, IX, IY, SP
Alternativní	AF; ostatní alternativní páry ověříte po <code>EXX</code>
Podmínky	NZ, Z, NC, C, PO, PE, P, M

```

EXPECT A,'x'           ; znak
EXPECT HL,buffer+4     ; výraz s návěštím
EXPECT DE,-1           ; $FFFF
EXPECT NC               ; bez přenosu
EXPECT (result),$34,$12 ; slovo $1234 v paměti
EXPECT (text),"OK",0    ; řetězec i s nulou

```

Jak testy připojit k projektu

ZPŮSOB	JAK	KDY SE HODÍ
TESTCLUEDE	Hlavní soubor projektu obsahuje <code>TESTCLUEDE "modul.tst"</code> ; soubor <code>.tst</code> začíná <code>INCLUDE "modul.asm"</code> a pokračuje testy. <code>EXT → T</code> funguje v obou souborech.	Testy patří k projektu: běžný překlad je vynechá, Run Tests spustí všechny. Příklad <code>arithmetic</code> .
Samostatný soubor	Testovací soubor má vlastní <code>ORG</code> , rutiny vloží přes <code>INCLUDE</code> a následují testy. <code>EXT → T</code> se tiskne v testovacím souboru.	Knihovna rutin bez vlastního programu. Příklad <code>strings_test.asm</code> .

Spojení funguje oběma směry, protože se **hlavní zdroj nikdy nekládá znovu**: `INCLUDE` nebo `TESTCLUEDE` souboru, který se právě překládá, se přeskočí (jméno i adresář, bez ohledu na velikost písmen). Stisknete-li `EXT → T` v `arithmetic.tst`, jeho `INCLUDE` vloží rutiny a jejich `TESTCLUEDE` se přeskočí; v `arithmetic.asm` `TESTCLUEDE` vloží testy a přeskočí se jejich `INCLUDE`.

Soubory `.tst` nabízí prohlížeč NextZOS stejně jako `.asm`; přípona je jen zvyk, assembler přeloží jakýkoli zdroj.

Jak se kontrola přeloží

`EXPECT` vloží do výstupu instrukci `RST $30 ($F7)`, znak `T`, druh kontroly a očekávaná data. Kontrola bajtového registru zabere 4 bajty, páru 5, podmínky 3 a paměti 6 + počet bajtů; `MOCK` je stejný záznam o 7 bajtech. Disassembly v debuggeru proto ukáže `RST $30` a za ním nesmyslné instrukce; řádek *Source* přitom ukazuje řádek `EXPECT`.

Náhrada ROM a esxDOS: STUB

Debugger nemá ROM ani esxDOS, takže rutina, která volá `RST $10` (tisk znaku) nebo `RST $08` (služba esxDOS), test zastaví. `STUB RST n,label` zařídí, že `RST n` v debuggeru zavolá vaši rutinu `label`: návratová adresa jde na zásobník jako u skutečného `RST` a `RET` náhrady se vrátí za `RST`. Náhrada může volání zaznamenat a test pak ověří, co rutina tiskla. Příklad `print.asm` a `print.tst` v `tests/fixtures/unittest`:

```
STUB    RST $10,fake_print

; A = character: appended to output. Keeps every register but F.
fake_print:    push    h1
               ld      h1,(out_ptr)
               ld      (h1),a
               inc     h1
               ld      (out_ptr),h1
               pop     h1
               ret
out_ptr:      dw      output
output:       ds      16

TESTCASE "print: HELLO"
ld      h1,hello
call    print           ; RST $10 for each character
EXPECT (output),"HELLO"
ENDTEST
```

- `STUB` do výstupu nic nevloží a mimo debugger nic nedělá. Platí pro celý překlad, i mimo `TESTCASE`, při krokování i v rychlém běhu: vektor `RST` v jádře debuggeru skočí rovnou na náhradu.

- RST \$30 patří kontrolám EXPECT a breakpointům a RST \$00 by v rychlém běhu zapnul divMMC; nahradit je nelze.
- Náhrada musí ležet v \$2000-\$FFFF. Pro jeden RST platí poslední STUB.
- Za RST \$08 následuje bajt s číslem služby esxDOS: náhrada jej přečte přes návratovou adresu a vrátí se za něj (`ex (sp),h1, ld a,(h1), inc h1, ex (sp),h1`).
- RST s náhradou počítá 11 taktů jako na Z80; takty náhrady se připočtou.

Podvrh vlastní rutiny: MOCK

Testovaná rutina často volá jinou, kterou v testu nechcete: čtení klávesnice, práci s diskem, náhodné číslo. `MOCK` routine, `fake` zapíše přes začátek rutiny `routine` skok `JP fake`, takže každé její volání až do konce testu skončí v podvrhu `fake`. Na konci testu debugger původní tři bajty vrátí. Příklad `player.asm` a `player.tst` v `tests/fixtures/unittest` takto podvrhne klávesnici; test sám určí, které klávesy jsou stisknuté:

```
; The fake keyboard: returns fake_held and counts the calls.
fake_keys:      ld      h1,fake_calls
                inc     (h1)
                ld      a,(fake_held)
                ret
fake_held:      db      0
fake_calls:     db      0

TESTCASE "P moves right"
MOCK read_keys,fake_keys
ld a,%01 ; P
ld (fake_held),a
call step ; step calls read_keys
EXPECT (player_x),17
EXPECT (fake_calls),1 ; the keyboard was read once
ENDTEST
```

- `MOCK` je záznam jako `EXPECT`: debugger jej provede jako jeden krok, nezmění žádný registr a nepočítá takty.
- Skok platí od místa, kde `MOCK` v testu stojí, do konce testu. Další testy vidí původní rutinu, i když se paměť kvůli nedostatku RAM neobnovuje.
- Podvrh si může zapsat, kolikrát a s čím byl volán, a test to ověří přes `EXPECT`. Počítadlo začíná v každém testu od nuly, protože se paměť obnovuje.
- Rutina i podvrh leží v \$2000-\$FFFF. Skok přepíše první tři bajty rutiny; kratší rutina přijde do konce testu i o začátek té následující. Volání ROM přes `CALL` takto nahradit nelze, na RST je STUB.
- V jednom testu nejvýše 16 podvrhů; zopakovaný `MOCK` téže dvojice se nepočítá.

PRAVIDLA

- Nejvýše 32 testů v jednom překladu.
- `EXPECT` lze použít i mimo `TESTCASE`; vyhodnotí se při krokování a trasování v debuggeru.
- Každý `TESTCASE` začíná nový rozsah lokálních návěští: `.loop` nebo `.fill` může mít každý test vlastní.
- Direktivy fungují i v makrech, v `REPT` a ve vložených souborech.
- `TEST` zůstává instrukcí Z80N (`test n`), proto se blok testu jmenuje `TESTCASE`.
- Direktivy testů zná jen NextForge. Rutiny, které chcete překládat i v `SjASMPlus`, držte v souboru bez testů a bez `TESTCLUE`.

27. Jak testy běží

Okno Run Tests

Run Tests přeloží hlavní zdroj i se soubory z TESTCLUDE. Když překlad uspěje, spustí testy na pozadí, dokud je okno reportu otevřené; řádek *Result* mezitím ukazuje, který test právě běží. Každý test začíná jako nové spuštění programu: registry nulové, příznaky jako po `XOR A`, přerušení povoleno v IM 1, výchozí zásobník `$1FFC` s návratovou adresou, počáteční stránkování a paměť programu ve stavu, v jakém ji našel první test. Test prošel, když doběhne až k `RET` z `ENDTEST` a všechny jeho kontroly `EXPECT` platily.

- **Spustí se všechny testy.** Selhání jednoho testu ostatní nezastaví.
- **Breakpointy se přeskočí** a zůstanou nastavené; zapomenutý breakpoint v rutině běh nezastaví.
- **Běží se pomalým trasováním**, i když debugger má nastavený rychlý běh: hardware se jen simuluje a obrazovka zůstane na reportu.
- **BREAK** běh ukončí; testy, na které nedošlo, zůstanou nespustěné.

V SEZNAMU	VÝZNAM
Zelená fajfka · 01 add16: 1000 + 234 ... 68 T	Test prošel; vpravo jsou jeho takty od začátku po <code>RET</code> z <code>ENDTEST</code> , včetně volaných rutin.
Červený křížek · 08 sub16: ...: HL=\$0006, expected \$0007	Kontrola neplatila: registr, skutečná a očekávaná hodnota; u podmínky <i>NZ expected</i> , <i>F=\$44</i> ; u paměti první rozdílnej bajt (<i>\$8039</i>)= <i>\$41</i> , <i>expected \$43</i> .
Červený křížek · ... stopped: ...	Test se zastavil z jiného důvodu, který následuje: volání ROM (RST bez STUB), HALT po DI.
Červený křížek · ... cannot start: ...	Kód testu leží pod <code>\$2000</code> (patří jádru debuggeru), nebo chybí volná paměť.
Otazník	Test se nespustil, protože běh ukončil BREAK.

Seznam posunete šipkami, Page Up / Page Down nebo kolečkem myši. Řádek *Result* je zelený (*All 12 tests passed.*), nebo červený (*1 of 12 tests failed., Stopped by BREAK after 9 of 12 tests, 1 failed., No TESTCASE blocks in the assembled source.*). ENTER, BREAK nebo kliknutí okno zavře; stavový řádek editoru pak ukáže stejný souhrn ve stejné barvě. **D** okno zavře a otevře debugger.

```
tes in the TESTCLUDE file at the end and runs its tests,
ch routine states what it takes, what it returns and what it keeps:
ac
Assembly report done

=  ✓ 01 add16: 1000 + 234 68 T
6:  ✓ 02 add16: overflow sets carry 68 T
  ✓ 03 add8sat: below the limit 56 T
  ✓ 04 add8sat: stops at 255 67 T
=  ✓ 05 sub16: 1234 - 234 76 T
58: ✓ 06 sub16: equal numbers give zero 76 T
  ✓ 07 sub16: borrow when DE is bigger 76 T
  ✓ 08 sub16: ignores the carry flag 80 T
  ✓ 09 memmove: copies 8 bytes 596 T
  ✓ 10 memmove: zero length copies nothing 96 T
  ✓ 11 memmove: overlap, move up by 2 404 T
  ✓ 12 memmove: overlap, move down by 2 357 T

Result: All 12 tests passed.
UP/DOWN/WHEEL scroll D debugger ENTER/ESC/CLICK close

= 0 copies nothing (a plain LDIR would copy 65536 bytes).
```

Všechny testy prošly.

```
tes in the TESTCLUDE file at the end and runs its tests,
ch routine states what it takes, what it returns and what it keeps:
ac
Assembly report done

=  ✓ 01 add16: 1000 + 234 68 T
6:  ✓ 02 add16: overflow sets carry 68 T
  ✓ 03 add8sat: below the limit 56 T
  ✓ 04 add8sat: stops at 255 67 T
=  ✓ 05 sub16: 1234 - 234 76 T
58: ✓ 06 sub16: equal numbers give zero 76 T
  ✓ 07 sub16: borrow when DE is bigger 76 T
  ✗ 08 sub16: ignores the carry flag: HL=$0006, expected $0007 596 T
  ✓ 09 memmove: copies 8 bytes 96 T
  ? 10 memmove: zero length copies nothing 404 T
  ? 11 memmove: overlap, move up by 2 357 T
  ? 12 memmove: overlap, move down by 2

Result: Stopped by BREAK after 9 of 12 tests, 1 failed.
UP/DOWN/WHEEL scroll D debugger ENTER/ESC/CLICK close

= 0 copies nothing (a plain LDIR would copy 65536 bytes).
```

BREAK během testu 10: testy 10–12 se nespustily.

KAŽDÝ TEST ZAČÍNÁ SE STEJNOU PAMĚTÍ

Když se spustí první test, debugger si zkopíruje všechny stránky programu a před každým dalším testem je vrátí; stránky, které test namapoval jako první, uvolní. Co jeden test zapíše, další tedy neuvidí, a po běhu je paměť jako před ním. Kopie potřebují volné stránky ve sdílené paměti s dokumenty. Když nestačí, testy paměť sdílejí a souhrn končí (*memory not reset: low RAM*); pomůže zavřít nepotřebný soubor. Stránky, které program poprvé namapuje při rychlém běhu v debuggeru, debugger nezná a neobnoví.

Barvy v editoru

Po každém běhu editor obarví řádky `TESTCASE` ve všech otevřených souborech, které se do překladač dostaly — i ve vloženém souboru `.tst`:

POZADÍ ŘÁDKU	VÝZNAM
Zelené	Test prošel.
Červené	Test selhal.
Beze změny	Test se nespustil (BREAK), nebo v debuggeru nedoběhl.

```
NextForge 1.2 by MB Maniax

TESTCASE "sub16: ignores the carry flag"
scf                                ; a carry left over by the caller
ld    hl,10
ld    de,3
call  sub16
EXPECT HL,7
ENDTEST

; --- moving memory -----

TESTCASE "memmove: copies 8 bytes"
ld    hl,source                    ; prepare: source = 1,2,...,8
ld    a,1
ld    b,8
.fill: ld    (hl),a                ; .fill is local to this test
inc    hl
inc    a
djnz  .fill
ld    hl,source
ld    de,target
ld    bc,8
call  memmove
EXPECT (target),1,2,3,4,5,6,7,8
EXPECT (guard),0xaa                ; nothing written past the end
ENDTEST

TESTCASE "memmove: zero length copies nothing"
ld    a,0xee
ld    (target),a                    ; a marker that must survive
FAIL sub16: ignores the carry flag: HL=$0006, expected $0007
```

Po běhu ukončeném BREAKem: test 8 selhal, test 9 prošel, test 10 se nespustil. Kurzor stojí na řádku testu 8, takže stavový řádek ukazuje důvod selhání.

Na červeném řádku ukazuje stavový řádek červeně **důvod selhání**, na zeleném zeleně **takty testu** (*PASS sub16: borrow when DE is bigger: 72 T*; po rychlém běhu v debuggeru jen jméno) — když na řádku stojí kurzor, nebo na něj ukážete myší (ukazatel má přednost). Jakmile kurzor i ukazatel opustí obarvené řádky, vrátí se běžný stavový řádek. Výsledek se najde podle jména testu v uvozovkách, takže platí i po úpravách, které řádky posunou; u přejmenovaného testu stavový řádek hlásí, že výsledek patří staršímu běhu.

Barva patří poslednímu běhu, do souboru se neukládá. Další běh ji obnoví; když po překladač změníte zdroj, řádky se nenajdou a barvy zůstanou z předchozího běhu. Breakpoint na řádku má přednost (tmavě červená).

Testovací překlad nic neukládá

Run Tests nezapíše výstupní soubor (binárku ani NEX): program s testy zůstane jen v paměti a dříve uložený výsledek běžného překladu se nepřepíše. Spadlý test tak nikdy nezanechá na kartě nic, co by vypadalo jako hotový program; výsledný soubor vytváří jen **Assemble** (EXT → A).

Testy v debuggeru

D v okně Run Tests nebo **EXT** → **T** přímo v debuggeru spustí stejné testy v debuggeru. Takový běh se liší od běhu v okně:

- zastaví se u **prvního** selhání nebo jiného zastavení a debugger zůstane na tom místě;
- **breakpointy platí** — test se na nich zastaví s hláškou *Test ... stopped: Stopped at a breakpoint* a můžete jej krokovat;
- běží v aktuálním režimu debuggeru: pomalé trasování, nebo rychlý nativní běh (**Z**), který se zastaví na každé kontrole a ukáže obrazovku programu. Test s **EXPECT CYCLES** běží vždy pomalým trasováním, protože rychlý běh takty nepočítá; zvolený režim se po běhu vrátí.

Dolní řádek debuggeru ukáže výsledek zeleně (*All 12 tests passed.*), nebo červeně (*FAIL 8/12 name: HL=\$0006, expected \$0007*). Po neúspěšném testu stojí kurzor kódu na kontrole, PC na instrukci za ní a registry, příznaky i paměť jsou ve stavu z okamžiku kontroly:

- **C** dokončí tento test; další kontroly se vyhodnotí také.
- **U** ENTER spustí tento test znovu od začátku, třeba po krocích (**S**). Paměť i podvrhy (MOCK) zůstanou, jak je test nechal.
- **N** otevře zdrojový řádek kontroly v editoru.
- **EXT** → **T** spustí všechny testy znovu bez nového překladu. Podvrhy zastaveného testu se nejdříve vrátí; ostatní paměť zůstane, jak ji test nechal, čistý obraz programu dá nový překlad.

Kontroly při krokování

I při ručním krokování (S, O, C) je každý EXPECT i MOCK jeden krok. Platná kontrola se jen přeskočí; neplatná ukáže červeně *EXPECT failed: ...* a PC se posune za ni, takže můžete pokračovat.

28. Tipy a omezení

Tipy

- Testujte malé rutiny s jasnými vstupy a výstupy; v komentáři rutiny popište, co bere, vrací a zachovává.
- Pojmenujte testy podle toho, co ověřují — jméno uvidíte při selhání.
- Zkoušejte okrajové případy: nulu, 255 a 256, \$FFFF, přetečení, prázdný řetězec, nulovou délku, konec bufferu.
- Ověřujte i to, co se změnit nemá: zachované registry (EXPECT DE, 234) a strážný bajt za koncem bufferu.
- Kontrolujte příznaky, na které se volající spoléhá (EXPECT C při přetečení), a otestujte obě větve podmínky.
- Po opravě chyby přidejte test, který by ji odhalil.
- Rutinu, která volá ROM nebo esxDOS přes RST, testujte se STUB: náhrada si zapíše, co rutina tiskla nebo zavolala, a test to ověří.
- Rutinu, která závisí na klávesnici, čase nebo jiné rutině, testujte s MOCK: podvrh vrátí přesně ty hodnoty, které test potřebuje.

Omezení

- Nejvýše 32 testů v jednom překladu a 255 bajtů v jedné kontrole paměti.
- Kód testů musí ležet v \$2000-\$FFFF na stránce, kterou má program na začátku namapovanou; u projektů se SAVENEX vložte TESTCLUE tam, kam se překládá hlavní kód.
- Volání ROM a esxDOS přes RST debugger neposkytuje; rutina, která je volá, test zastaví, pokud RST nenahradíte direktivou STUB. Volání ROM instrukcí CALL nahradit nelze.
- V okně Run Tests běží testy vždy pomalým trasováním; dlouhé testy spustte v debuggeru rychlým během (Z, D). Testy s EXPECT CYCLES běží pomalu i tam.
- Obnova paměti mezi testy potřebuje volné stránky na kopii programu (kapitola 27).
- EXPECT porovnává stav po celé instrukci, nesleduje sběrnici; vstup a výstup ověříte jen nepřímo (řádek I/O debuggeru).
- Mimo debugger je EXPECT obyčejná instrukce RST \$30: do programu proto testy vkládejte jen přes TESTCLUE.

CHYBA PŘEKLADU	PŘÍČINA
bad operand u TESTCASE	Vnořený TESTCASE nebo jméno bez uvozovek.
bad operand u ENDTEST nebo na konci zdroje	ENDTEST bez TESTCASE, nebo TESTCASE bez ENDTEST.
bad operand u EXPECT	Neznámý registr nebo podmínka, chybí hodnota nebo čárka za adresou.
too many operands u EXPECT	Víc hodnot u registru; seznam bajtů patří jen ke kontrole paměti.
value/address out of range	Hodnota bajtového registru mimo -128..255, nebo více než 32 testů.
missing operand u TESTCLUE	Chybí jméno souboru.
bad operand u STUB	Chybí slovo RST nebo návěští náhrady.
bad operand u MOCK	Chybí rutina nebo podvrh.
value/address out of range u MOCK	Rutina nebo podvrh leží pod \$2000.
value/address out of range u STUB	RST \$00 nebo \$30, číslo, které není vektor RST, nebo náhrada pod \$2000.
undefined symbol v testu	Testovací soubor přeložený jako hlavní nekládá testované rutiny: přidejte na jeho začátek INCLUDE "modul.asm".
file not found opening ...	Soubor z TESTCLUE (hledá se jen při Run Tests) nebo z INCLUDE neexistuje.