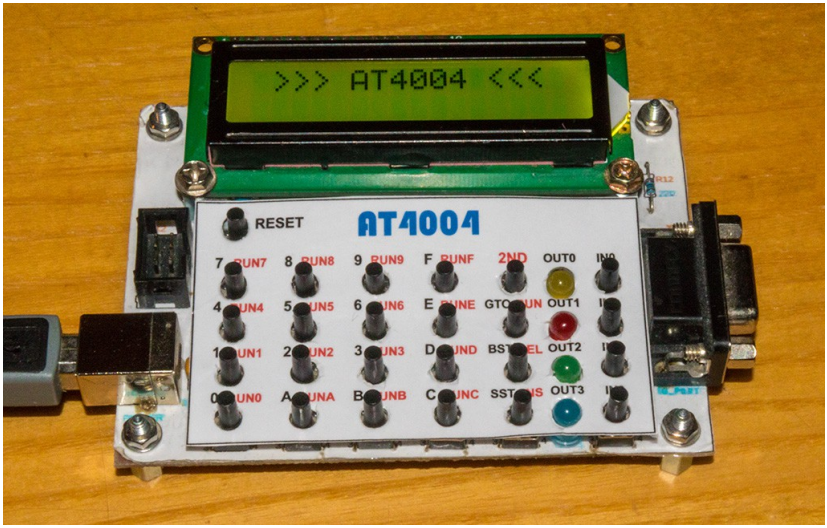
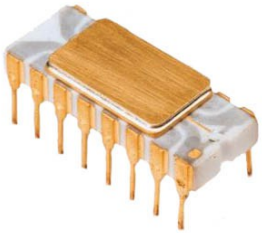
VÝVOJOVÝ KIT PROCESORU INTEL 4004/4040

AT4004

s emulací kalkulátoru Busicom 141-PF

uživatelská příručka





Vývojový kit procesoru Intel 4004/4040

AT4004

uživatelská příručka

open source

(c) Miroslav Němeček, říjen 2021

Panda38@seznam.cz

www s popisem konstrukce: http://www.breatharian.eu/hw/at4004/

OBSAH

1. Rozložení klávesnice

Programovací mód, základní rozložení kláves

7	**8**	**9**	**F**	**2ND**
4	**5**	**6**	**E**	**GTO**
1	**2**	**3**	**D**	**BST**
0	**A**	**B**	**C**	**SST**

Programovací mód, alternativní význam po stisku tlačítka **2ND**

RUN7	**RUN8**	**RUN9**	**RUNF**	**1ST**
RUN4	**RUN5**	**RUN6**	**RUNE**	**RUN**
RUN1	**RUN2**	**RUN3**	**RUND**	**DEL**
RUN0	**RUNA**	**RUNB**	**RUNC**	**INS**

Emulace kalkulátoru Busicom, základní rozložení kláves

7	**8**	**9**	**:**	**2ND**	
4	**5**	**6**	**x**	**RM**	
1	**2**	**3**	**-**	**C**	
0	**.**	**=**	**+**	**CE**	**ADV**

Emulace kalkulátoru Busicom, alternativní význam po stisku **2ND**

FL	**DP8**	**N**	**SUB**	**1ST**
DP4	**DP5**	**DP6**	**SQRT**	**S**
DP1	**DP2**	**DP3**	**M-**	**CM**
DP0	**54**	**%**	**M+**	**EX**

Poznámka: *Oproti původnímu kalkulátoru chybí klávesy M=+, M=-, 00 a 000. Klávesu M=+ lze nahradit stiskem kláves '=' a M+. Klávesu M=- lze nahradit stiskem kláves '=' a M-.*

2. Popis a použití

AT4004 je kit sloužící k základnímu seznámení s programováním procesoru Intel 4004 a 4040. Procesor je simulován v reálném čase pomocí procesoru ATmega8, časovaného interním oscilátorem 8 MHz. Součástí kitu je emulace kalkulátoru Busicom 141-PF. Emulované programy lze kompilovat buď jako součást vestavěné ROM paměti procesoru ATmega8, nebo editovat přímým zápisem do interní EEPROM paměti ve vestavěném editoru. Z emulovaného programu lze přistupovat ke 4-bitovému vstupnímu a výstupnímu portu, vyvedenému jednak na 9pinový konektor CANON, a jednak na vestavěná tlačítka a LED diody. Je možné též přistupovat na ostatní tlačítka a znakový LCD displej. Pro emulovaný program je k dispozici ROM paměť o velikosti 14 stránek po 256 bajtech (tj. 3584 bajtů) a EEPROM paměť, určená k přímé editaci programu, o velikosti 2 stránky po 256 bajtech (tj. 512 bajtů). Jako RAM má emulovaný program k dispozici 5 paměťových bank, to je celkem 1600 číslic, resp. 800 bajtů.

Kit lze napájet přes USB kabel buď z USB konektoru počítače, nebo z USB +5V nabíječky. Po zapnutí napájení se aktivuje programovací mód na adrese 0. Na adresách 0x000 až 0xDFF lze prohlížet ROM paměť s emulovaným programem. V této části se nachází kód kalkulačky Busicom 141-PF. Samotný kód kalkulačky je velký 5 stránek paměti ROM, to je 1280 bajtů. Zbylých 9 stránek lze využít pro uživatelské programy. Tuto část paměti nelze přímo editovat, do procesoru je zavedena překladem v hostitelském počítači a naprogramováním procesoru ATmega8. ROM program musí být ukončen na adrese 0xE00, kde začíná EEPROM program.

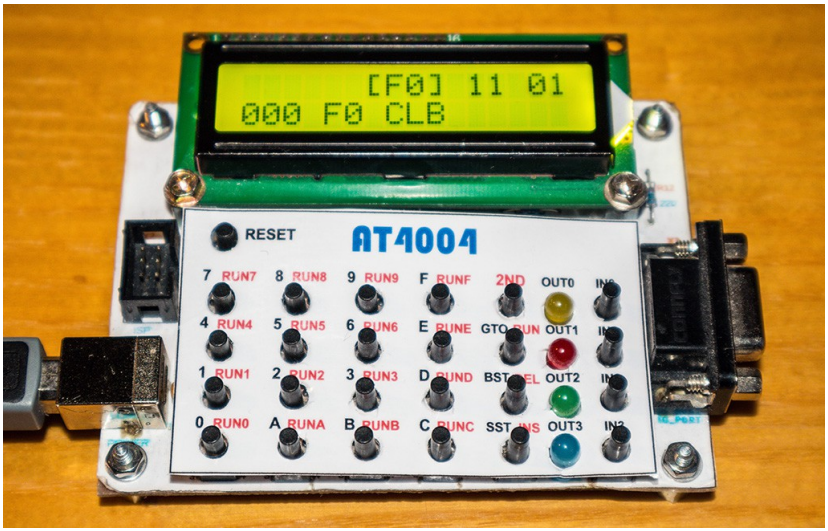
Na adresách 0xE00 až 0xFFF se nachází 2 stránky paměti EEPROM (to je 512 bajtů). Tuto část lze editovat přímo v kitu a může být zachována i při přeprogramování procesoru ATmega8.

Program v EEPROM lze editovat zápisem kódu programu v HEX kódu (číslice 0 až F). Kód lze připravit kompilátorem AS4 a poté ručně opsat do EEPROM paměti. U ROM paměti ruční přepis není zapotřebí, protože je kód programu zakomponován do programu procesoru ATmega8.

Emulovaný program lze spouštět buď od aktuální zobrazené adresy, nebo od začátku stránek 0 až F. Kód kalkulátoru Busicom 141-PF lze po resetu kitu spustit tlačítkem **2ND** **RUN** nebo **2nd** **RUN0**.

3. Editace programu

Po zapnutí kitu AT4004 se spustí programovací editační mód. V tomto módu lze prohlížet emulovaný program v ROM paměti a editovat program v EEPROM paměti. Na prvním řádku displeje se zobrazí 5 bajtů paměti v okolí aktuální adresy. Bajt z aktuální adresy je ohraničen lomenými závorkami. Bajty mimo platný rozsah adres se nezobrazí. Všechny číselné údaje jsou uvedeny v HEX kódu.



Na druhém řádku displeje je obsah aktuální adresy. Na začátku řádku jsou 3 číslice aktuální adresy, v rozsah 0x000 až 0xFFF. Adresy 0x000 až 0xDFF obsahují needitovatelnou ROM paměť, adresy 0xE00 až 0xFFF obsahují editovatelnou EEPROM paměť. Za adresou následuje obsah bajtu na aktuální adrese. Za bajtem je mnemotechnický tvar instrukce na aktuální adrese. Instrukce je zobrazena jen zjednodušeně symbolicky - zobrazí se jen jméno instrukce a případně číselný parametr jako 1 HEX číslice vyplývající z kódu instrukce. Nezobrazí se přídavný parametr u instrukcí zabírajících 2 bajty.

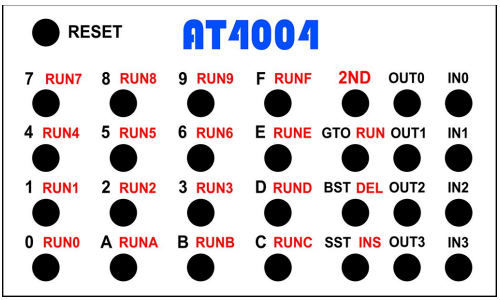
Poznámka: *Programem se listuje po bajtech, ne po instrukcích. To znamená, že u dvoubajtových instrukcí se na následující adrese zobrazí kód instrukce odpovídající číselnému parametru instrukce. To může být zpočátku matoucí, proto je potřeba se spíše držet listingu přeloženého*

programu, pracovat s HEX zápisem a textový tvar instrukcí používat jen jako lehké vodítko.

Při procházení programu lze k listování po bajtech používat tlačítka **SST** (krok vpřed) a **BST** (krok zpět). Tlačítkem **GTO** lze skočit na zadanou adresu. Zadávání cílové adresy **GTO** lze přerušit stiskem nečíselné klávesy (např. **SST** nebo **BST**).

Při editaci programu v EEPROM (adresy 0xE00 až 0xFFF) se přepíše bajt na aktuální adrese zadáním 2 číslic. Číslicemi jsou myšleny HEX číslice, tedy 0..9 a A..F. Zadávání lze přerušit po zadání první číslice stiskem nečíselné klávesy (např. **SST** nebo **BST**).

Program lze spustit buď z aktuální adresy tlačítkem **2ND** **RUN** nebo ze začátku některé stránky tlačítky **2ND** **RUN0** až **RUNF**.



Klávesy v programovacím módu:

2ND ... Klávesa pro alternativní význam. Po jejím stisku se změní význam následujícího tlačítka. Druhým stiskem tlačítka **2ND** se význam vrátí zpět na základní význam.

0 až **9**, **A** až **F** ... Číselné klávesy. Použijí se po tlačítku **GTO** k nastavení aktuální adresy nebo při editaci programu v EEPROM na adresách 0xE00 až 0xFFF k zadání hodnoty bajtu. Je-li potřeba přerušit zadávání adresy nebo dat, lze to provést stiskem nečíselné klávesy (např. **SST** nebo **BST**).

GTO (Go to) ... Skok na zadanou adresu. Je-li potřeba přerušit zadávání adresy, lze to provést stiskem nečíselné klávesy (např. **SST** nebo **BST**).

BST (Back Step) ... Posun adresy o 1 bajt zpět.

SST (Single Step) ... Posun adresy o 1 bajt vpřed.

2ND RUN0 až **2ND RUNF** ... Start programu od začátku stránky 0 až 15, tedy od adresy 0x000, 0x100 až 0xF00. Každá stránka má 256 bajtů.

2ND RUN ... Start programu od aktuální adresy.

2ND DEL ... Smazání bajtu na aktuální adrese. Bajty až po konec aktuální stránky se přisunou a na konec aktuální stránky se vloží bajt 0xFF. Funkce neovlivní ostatní stránky. Každá stránka má 256 bajtů.

2ND INS ... Vložení bajtu na aktuální adresu. Bajty až po konec aktuální stránky se odsunou a na aktuální adresu se vloží bajt 0xFF. Funkce neovlivní ostatní stránky. Každá stránka má 256 bajtů.

RESET ... Resetování kitu a nový start editoru programu.

Při startu programu se provedou následující úkony:

- vymazání displeje
- vymazání RAM paměti
- vynulování registrů
- vynulování akumulátoru a registru příznaků
- posuvné registry klávesnice a tiskárny se vyplní 0xFF
- přepínač rozsahu zaokrouhlení se nastaví na 8
- přepínač módu zaokrouhlení se nastaví na FL
- vynulují se registry DCL a SCR
- vynuluje se ukazatel zásobníku

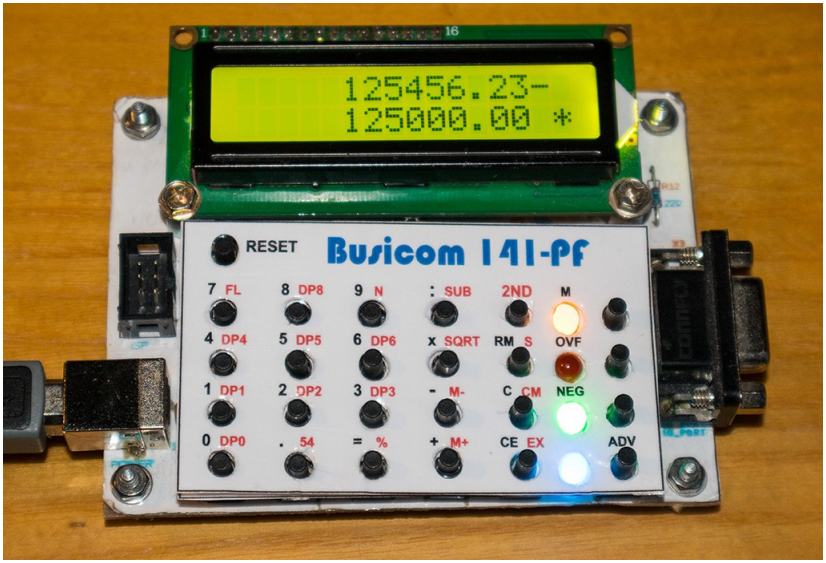
Probíhající program lze přerušit pouze tlačítkem RESET nebo odpojením napájení.

4. Emulace kalkulátoru Busicom 141-PF

Kalkulátor **Busicom 141-PF** (**UniCom 141P** byla jeho OEM verze) vznikl v roce 1971 a byl postaven na procesoru Intel 4004, který firma Intel pro tento účel vyvinula. Procesor pracoval s hodinovou frekvencí 740 kHz a dobou instrukcí 10,8 a 21,6 us. Kalkulátor neobsahoval displej, ale tiskárnu s otočným tiskovým bubnem a se šířkou tisku 18 znaků. Dvě poslední tiskové pozice byly vyhrazeny pro symboly operací, jedna pozice byla oddělovací a 15 pozic obsahovalo číslice s desetinnou tečkou. Kalkulátor mohl zobrazit až 14 číslic s plovoucí desetinnou tečkou, vnitřně počítal s přesností 16 číslic.

Emulaci kalkulátoru Busicom 141-PF spustíte po zapnutí kitu stiskem tlačítek **2ND RUN** (start od aktuální adresy 0) nebo stiskem **2ND RUN0**. Na rozdíl od originálu neuslyšíte hučení bubnu tiskárny a dokonce ani neuvidíte nic na displeji po stisku číselných kláves. Displej kitu emuluje výstup na tiskárnu a tak se údaj objeví až po provedení operace.

Dokonce, aby se neztratily výstupní řádky, emulátor potlačuje výstup prázdných řádků, kromě případu odrolování výstupu tlačítkem **ADV**.



Emulátor kalkulačky používá kromě základních tlačítek i vstupní a výstupní porty AT4004. LED na výstupních pinech indikují stav kalkulačky:

- OUT0 (žlutá LED, označená "M") indikuje nenulový obsah paměti
- OUT1 (červená LED, označená "OVF") indikuje chybu přetečení
- OUT2 (zelená LED, označená "NEG") indikuje záporné znaménko

Emulátor obsahuje ještě čtvrtý výstup OUT3 (modrá LED), který kalkulátor neobsahuje, a tak se zde zobrazí náhodný stav zapsaný kalkulátorem na porty. Obvykle se OUT3 rozsvítí shodně s indikací záporného znaménka OUT2.

Je nutno podotknout, že původní kalkulátor nezobrazuje záporná čísla znaménkem mínus, ale červenou barvou tisknutého řádku. Displej emulátoru nemá tuto možnost a proto je záporné číslo indikováno pouze rozsvícením zelené LED OUT2 s označením "NEG".

Druhá poznámka se týká délky tisknutého řádku. Původní kalkulátor používá délku řádku 18 pozic, zatímco emulátor má k dispozici pouze 16 znaků na řádek. Vypuštěním oddělovacího sloupce 16 se počet potřebných znaků sníží na 17. Zbylou jednu nadbytečnou pozici se emulátor snaží zredukovat vypuštěním nejméně potřebného znaku. Začíná-li text řádku mezerou, vypustí mezeru ze začátku řádku. Obsahuje-li poslední stavový

znak (ve sloupci 18) mezeru, tuto mezeru vypustí. V posledním případě přenese znak ze sloupce 18 do sloupce 17, protože druhý stavový znak obsahuje obvykle důležitější informaci.

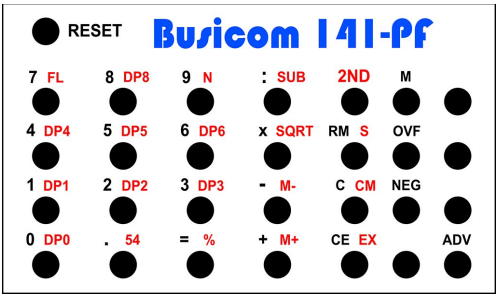
Při ovládání kalkulátoru je potřeba zohlednit nízkou rychlost kalkulátoru. Klávesnice kalkulátoru je obsluhována poměrně pomalu (zřejmě se v té době tolik nespěchalo) a tak se při rychlejším stisku může stát, že se některý stisk tlačítka ztratí (např. v čísle bude chybět číslice). Zadávejte proto tlačítka pomaleji, asi tak s frekvencí max. 2 stisky za sekundu.

Ovládání kalkulátoru není tak intuitivní jak bychom si v dnešní době představovali. Nemůžeme jednoduše zapsat "1 + 1 =". Je to jedna z prvních kalkulaček, kdy byli lidé ještě zvyklí na mechanické kalkulačky, a tak se tomu ovládání přizpůsobilo. Podrobný popis ovládání naleznete v manuálu "UniCom 141P", který je přiložen ke zdrojovému kódu kitu AT4004. Zde uvedu jen stručný souhrn.

Při sčítání a odečítání zadáte nejdříve číslo a potom operaci **+** nebo **-**, která se s číslem má provést, a to i pro první a poslední číslo. Výsledek zobrazíte tlačítkem **=**. Např. "12,34 - 56,78" zapíšete jako **12.34+56.78-=**. Je to z důvodu, že čísla se přičítají/odečítají od akumulátoru.

Při násobení a dělení se naopak postupuje tak, jak jsme zvyklí z novějších kalkulaček, např. zapíšete **12.3x4.56=** .

Nastavení počtu desetinných míst a módu zaokrouhlení se projeví až při výstupu výsledku. Zvláštností je mód zaokrouhlení FL (floating). Znamená, že se odstraní nevýznamné nuly z konce čísla a číslo se zobrazí v maximální přesnosti, nezávisle na přepínači volby přesnosti. Ovšem tento mód funguje pouze u násobení a dělení, při součtu a rozdílu se neuplatní. Ptáte-li se, proč takové podivnosti v chování kalkulátoru - vážně netuším, asi to tehdy líp neuměli udělat. :-)



Klávesy při emulaci kalkulátoru Busicom 141-PF:

2ND ... Klávesa pro alternativní význam. Po jejím stisku se změní význam následujícího tlačítka. Druhým stiskem tlačítka **2ND** se význam vrátí zpět na základní význam. Tlačítko není součástí kalkulátoru, je pomocným tlačítkem emulátoru k rozšíření počtu tlačítek.

C (Clear) ... Vynulování pracovních registrů a příznaku přetečení.

CE (Clear Error) ... Vynulování chybně zadaného čísla a příznaku přetečení.

0 až **9** ... Číselné klávesy.

. ... Desetinná tečka.

S ... Záporné znaménko zadaného čísla.

+ a **-** ... Sčítání a odečítání. Kalkulátor pracuje s hlavním a vedlejším akumulátorem. Nejdříve je nutno zapsat číslo a potom až stisknout klávesu **+** nebo **-** znamenající, zda se číslo má k akumulátoru přičíst nebo odečíst. Výsledný stav akumulátoru se zobrazí po stisku tlačítka **=**. Tento styl práce připomíná "obrácenou polskou notaci", známou v pozdější době z kalkulaček Hewlett-Packard.

x a **:** ... Násobení a dělení. Použití se liší od sčítání a odečítání. Nejdříve se zadá první operand, pak klávesa požadované operace, druhý operand a zobrazení výsledku stiskem **=**.

= ... Po sčítání a odečítání zobrazení obsahu hlavního akumulátoru. Při násobení a dělení provedení výpočtu a zobrazení výsledku.

SUB (Sub Total) ... Zobrazení a vynulování akumulátoru mezivýsledků.

EX (Exchange) ... Záměna registrů během násobení a dělení.

% ... Výpočty procent.

CM (Clear Memory) ... Zobrazení a vynulování obsahu paměti.

RM (Recall Memory) ... Zobrazení obsahu paměti. Paměť zůstane nezměněna.

M+ a **M-** ... Přičtení nebo odečtení od paměti. Původní kalkulátor obsahuje ještě tlačítka **M=+** a **M=-**, která se v emulátoru nahradí stiskem tlačítka **=** následovaného stiskem **M+** nebo **M-**.

ADV (Advance) ... Posun papíru o 1 řádek.

FL (Floating) ... Nastavení přepínače módu zaokrouhlení na plovoucí desetinnou čárku. Nadbytečné nuly z konce čísla se odstraní a číslo se zobrazí s maximální přesností. Nastavení přepínače počtu desetinných míst se neuplatní. Tento mód se projeví pouze u násobení a dělení.

N (No rounding) ... Nastavení přepínače módu zaokrouhlení na ořezávání výsledku, číslo se nezaokrouhlí.

54 ... Výsledek se zaokrouhlí k nejbližší číslici.

DP0 až **DP6**, **DP8** ... Zkrácení výsledku na 0 až 6 nebo 8 desetinných míst. Přepínač se neuplatní v módu FL při násobení a dělení.

5. Procesor 4004 a 4040

AT4004 simuluje procesor Intel 4004 a většinu rozšíření procesoru Intel 4040. Simulovaný hardware odpovídá kalkulátoru Busicom 141-PF.

Specifikace simulace procesoru 4004/4040

Frekvence krystalu: 740 kHz

Strojový cyklus: 8 hodinových taktů, tj. 10,8 us

Délka instrukcí: 1 nebo 2 bajty

Čas zpracování instrukcí: 1 strojový cyklus tj. 10,8 us (instrukce 1 bajt) nebo 2 strojové cykly tj. 21,6 us (instrukce 2 bajty a instrukce FIN)

Indexové registry: 24 registrů po 4 bitech rozdělených do 3 sad. 1. sada jsou registry R0 až R7, 2. sada registry R8 až R15 a 3. sada registry R0' a R7'. Registry R8 až R15 (2. sada) jsou dostupné vždy. Registry z 1. a 3. sady (R0 až R7, R0' až R7') se přepínají pomocí instrukcí SB0 a SB1. Přepínání sady indexových registrů je možné jen u procesoru 4040. Procesor 4004 používá pouze základní sadu registrů R0 až R15.

K indexovým registrům lze přistupovat buď jednotlivě, R0 až R15 (16 registrů po 4 bitech), nebo jako k registrovým párům, R0R1, R2R3, ... R14R15 (8 registrových párů po 8 bitech). První registr (sudé číslo, tj. R0, R2, ...) obsahuje vyšší 4 bity dat, druhý registr (liché číslo, tj. R1, R3, ...) obsahuje nižší 4 bity dat.

Velikost zásobníku návratových adres: 8. Původní procesor 4004 umožňuje 3 návratové adresy a procesor 4040 7 návratových adres.

Velikost paměti RAM: 800 bajtů

Organizace paměti RAM: 5 bank, každá banka 4 čipy, každý čip 4 registry, každý registr 16 znaků (4-bitové slabiky) a 4 stavové znaky. To je celkem 5 x 4 x 4 x (16+4) = 1600 slabik (po 4 bitech) neboli 800 bajtů.

Velikost paměti ROM: Needitovatelná paměť ROM 14 stránek po 256 bajtech, tj. 3584 bajtů. V adresovém prostoru se nachází na adresách

0x000 až 0xDFF a obsahuje ROM emulovaného kalkulátoru 141-PF, s rezervou pro doplnění vlastního kódu. Za pamětí ROM se nachází 2 stránky paměti EEPROM, tj. celkem 512 bajtů, na adresách 0xE00 až 0xFFF, do které je možné dodatečně psát vlastní programy.

Specifikace simulace kalkulátoru Busicom 141-PF

Signál Test - signál sektorů tiskového bubnu, frekvence 35,7 Hz, interval 28 ms.

Index signál - indikuje sektor 0 tiskového bubnu. Sektorů je 13. Frekvence indexového signálu je 2,74 Hz, perioda 364 ms.

Port ROM0 (výstup): výstup na posuvný registr

bit0 = hodiny posuvného registru řádků matice klávesnice
bit1 = data posuvného registru (společné pro tisk i klávesnici)
bit2 = hodiny posuvného registru tiskárny (1=zápis)

Port ROM1 (vstup): vstup sloupců z matice klávesnice (1=stisk)

Port ROM2 (vstup):

bit0 = vstup indexového signálu 0 tiskového bubnu (1=aktivní)
bit3 = vstup tlačítka posuvu papíru tiskárny (1=stisknuto)

Port ROM3 (vstup): vstup externích pinů simulátoru

bit0 = vstup pinu IN0
bit1 = vstup pinu IN1
bit2 = vstup pinu IN2
bit3 = vstup pinu IN3 (tlačítko ADV)

Port ROM4 (výstup): výstup volby řádku displeje emulátoru 0..1

bit0 = 0 horní řádek, 1 dolní řádek

Zápis vynuluje ukazatel pozice na displeji a resetuje flip-flop horní/dolní poloviny znaku.

Port ROM5 (výstup): výstup volby pozice na řádku displeje emulátoru

bit0..bit3 = pozice na řádku displeje 0..15

Zápis resetuje flip-flop horní/dolní poloviny znaku.

Port ROM6 (výstup): výstup poloviny znaku na displej emulátoru

bit0..bit3 = horní nebo dolní polovina znaku

První se zapisuje horní polovina znaku, poté dolní polovina znaku, v závislosti na přepínači flip-flop. Při zápisu horní poloviny se neprovede fyzický výstup na displej, pouze se uchová v pomocném bufferu dokud se neprovede zápis i dolní poloviny. Pozice na řádku se automaticky inkrementuje.

Port RAM0 (výstup): výstup řízení tisku

bit0 = barva tisku (0=černá, 1=červená)
bit1 = úder tiskových kladívek (úder při 0 -> 1)
bit3 = posun papíru tiskárny (posun při 0 -> 1)

Port RAM1 (výstup): výstup stavových světel

bit0 = paměť M, výstup emulátoru OUT0, žlutá LED1
bit1 = přetečení OVF, výstup emulátoru OUT1, červená LED2
bit2 = negativní NEG, výstup emulátoru OUT2, zelená LED3
bit3 = výstup emulátoru OUT3, modrá LED4

Přehled instrukcí 4004/4040

0x00 NOP, žádná operace
0x01 HLT, zastavení a čekání na přerušení
0x02 BBS, návrat z přerušení
0x03 LCR, načtení DCL registru do ACC
0x04 OR4, logické OR registru 4 a akumulátoru
0x05 OR5, logické OR registru 5 a akumulátoru
0x06 AN6, logické AND registru 6 a akumulátoru
0x07 AN7, logické AND registru 7 a akumulátoru
0x08 DB0, aktivace ROM banky 0
0x09 DB1, aktivace ROM banky 1
0x0A SB0, aktivace banky indexových registrů 0
0x0B SB1, aktivace banky indexových registrů 1
0x0C EIN, povolení přerušení
0x0D DIN, zákaz přerušení
0x0E RPM, čtení programové paměti
0x1n JCN, podmíněný skok (JT, JC, JZ, JR, JNT, JNC, JNZ)
0x2n (sudá) FIM, načtení konstanty
0x2n (lichá) SRC, odeslání řídicí adresy
0x3n (sudá) FIN, nepřímé čtení z ROM

0x3n (lichá) JIN, nepřímý skok
0x4n JUN (JMP), nepodmíněný skok
0x5n JMS (CALL), skok do podprogramu
0x6n INC, inkrementace registru
0x7n ISZ (IJNZ), inkrementace a přeskočení při nule
0x8n ADD, přičtení registru a carry k akumulátoru
0x9n SUB, odečtení registru a carry od akumulátoru
0xAn LD, načtení registru do akumulátoru
0xBn XCH, záměna registru a akumulátoru
0xCn BBL (RET), návrat z podprogramu
0xDn LDM (LDI), načtení konstanty do akumulátoru
0xE0 WRM, zápis do RAM paměti
0xE1 WMP, zápis do RAM portu
0xE2 WRR, zápis do ROM portu
0xE3 WPM, zápis do programové paměti
0xE4 WR0, zápis do RAM stavového registru 0
0xE5 WR1, zápis do RAM stavového registru 1
0xE6 WR2, zápis do RAM stavového registru 2
0xE7 WR3, zápis do RAM stavového registru 3
0xE8 SBM, odečtení RAM paměti
0xE9 RDM, načtení RAM paměti
0xEA RDR, načtení ROM portu
0xEB ADM, přičtení RAM paměti
0xEC RD0, načtení RAM stavového registru 0
0xED RD1, načtení RAM stavového registru 1
0xEE RD2, načtení RAM stavového registru 2
0xEF RD3, načtení RAM stavového registru 3
0xF0 CLB, vymazání obou
0xF1 CLC, vynulování carry
0xF2 IAC (INC A), inkrementace akumulátoru
0xF3 CMC, komplement carry
0xF4 CMA, komplement akumulátoru
0xF5 RAL, rotace vlevo
0xF6 RAR, rotace vpravo
0xF7 TCC, přenos carry
0xF8 DAC (DEC A), dekrementace akumulátoru
0xF9 TCS, přenos carry odečtení
0xFA STC, nastavení carry
0xFB DAA, dekadická korekce
0xFC KBP, korekce kláves
0xFD DCL, řídicí registr adresy

Mapa instrukcí 4004/4040 (hex kód a mnemonický zápis):

0 až 15 ... konstanta nebo podmínka
a ... adresa 8 bitů
0a až Fa ... adresa 12 bitů (4 bity číslice + 8 bitů z parametru)
r0 až r15 ... indexový registr 0 až 15
r01 až r1415 ... pár indexových registrů 0..7

00	NOP		10	JCN	0,a	20	FIM	r01,d	30	FIN	r01
01	HLT		11	JCN	1,a	21	SRC	r01	31	JIN	r01
02	BBS		12	JCN	2,a	22	FIM	r23,d	32	FIN	r23
03	LCR		13	JCN	3,a	23	SRC	r23	33	JIN	r23
04	OR4		14	JCN	4,a	24	FIM	r45,d	34	FIN	r45
05	OR5		15	JCN	5,a	25	SRC	r45	35	JIN	r45
06	AN6		16	JCN	6,a	26	FIM	r67,d	36	FIN	r67
07	AN7		17	JCN	7,a	27	SRC	r67	37	JIN	r67
08	DB0		18	JCN	8,a	28	FIM	r89,d	38	FIN	r89
09	DB1		19	JCN	9,a	29	SRC	r89	39	JIN	r89
0A	SB0		1A	JCN	10,a	2A	FIM	r1011,d	3A	FIN	r1011
0B	SB1		1B	JCN	11,a	2B	SRC	r1011	3B	JIN	r1011
0C	EIN		1C	JCN	12,a	2C	FIM	r1213,d	3C	FIN	r1213
0D	DIN		1D	JCN	13,a	2D	SRC	r1213	3D	JIN	r1213
0E	RPM		1E	JCN	14,a	2E	FIM	r1415,d	3E	FIN	r1415
0F	-		1F	JCN	15,a	2F	SRC	r1415	3F	JIN	r1415

40	JUN	0a	50	JMS	0a	60	INC	r0	70	ISZ	r0,a
41	JUN	1a	51	JMS	1a	61	INC	r1	71	ISZ	r1,a
42	JUN	2a	52	JMS	2a	62	INC	r2	72	ISZ	r2,a
43	JUN	3a	53	JMS	3a	63	INC	r3	73	ISZ	r3,a
44	JUN	4a	54	JMS	4a	64	INC	r4	74	ISZ	r4,a
45	JUN	5a	55	JMS	5a	65	INC	r5	75	ISZ	r5,a
46	JUN	6a	56	JMS	6a	66	INC	r6	76	ISZ	r6,a
47	JUN	7a	57	JMS	7a	67	INC	r7	77	ISZ	r7,a
48	JUN	8a	58	JMS	8a	68	INC	r8	78	ISZ	r8,a
49	JUN	9a	59	JMS	9a	69	INC	r9	79	ISZ	r9,a
4A	JUN	Aa	5A	JMS	Aa	6A	INC	r10	7A	ISZ	r10,a
4B	JUN	Ba	5B	JMS	Ba	6B	INC	r11	7B	ISZ	r11,a
4C	JUN	Ca	5C	JMS	Ca	6C	INC	r12	7C	ISZ	r12,a
4D	JUN	Da	5D	JMS	Da	6D	INC	r13	7D	ISZ	r13,a
4E	JUN	Ea	5E	JMS	Ea	6E	INC	r14	7E	ISZ	r14,a
4F	JUN	Fa	5F	JMS	Fa	6F	INC	r15	7F	ISZ	r15,a

80	ADD	r0	90	SUB	r0	A0	LD	r0	B0	XCH	r0
81	ADD	r1	91	SUB	r1	A1	LD	r1	B1	XCH	r1
82	ADD	r2	92	SUB	r2	A2	LD	r2	B2	XCH	r2
83	ADD	r3	93	SUB	r3	A3	LD	r3	B3	XCH	r3
84	ADD	r4	94	SUB	r4	A4	LD	r4	B4	XCH	r4
85	ADD	r5	95	SUB	r5	A5	LD	r5	B5	XCH	r5
86	ADD	r6	96	SUB	r6	A6	LD	r6	B6	XCH	r6
87	ADD	r7	97	SUB	r7	A7	LD	r7	B7	XCH	r7
88	ADD	r8	98	SUB	r8	A8	LD	r8	B8	XCH	r8
89	ADD	r9	99	SUB	r9	A9	LD	r9	B9	XCH	r9

8A	ADD	r10	9A	SUB	r10	AA	LD	r10	BA	XCH	r10
8B	ADD	r11	9B	SUB	r11	AB	LD	r11	BB	XCH	r11
8C	ADD	r12	9C	SUB	r12	AC	LD	r12	BC	XCH	r12
8D	ADD	r13	9D	SUB	r13	AD	LD	r13	BD	XCH	r13
8E	ADD	r14	9E	SUB	r14	AE	LD	r14	BE	XCH	r14
8F	ADD	r15	9F	SUB	r15	AF	LD	r15	BF	XCH	r15

C0	BBL	0	D0	LDM	0	E0	WRM		F0	CLB	
C1	BBL	1	D1	LDM	1	E1	WMP		F1	CLC	
C2	BBL	2	D2	LDM	2	E2	WRR		F2	IAC	
C3	BBL	3	D3	LDM	3	E3	WPM		F3	CMC	
C4	BBL	4	D4	LDM	4	E4	WR0		F4	CMA	
C5	BBL	5	D5	LDM	5	E5	WR1		F5	RAL	
C6	BBL	6	D6	LDM	6	E6	WR2		F6	RAR	
C7	BBL	7	D7	LDM	7	E7	WR3		F7	TCC	
C8	BBL	8	D8	LDM	8	E8	SBM		F8	DAC	
C9	BBL	9	D9	LDM	9	E9	RDM		F9	TCS	
CA	BBL	10	DA	LDM	10	EA	RDR		FA	STC	
CB	BBL	11	DB	LDM	11	EB	ADM		FB	DAA	
CC	BBL	12	DC	LDM	12	EC	RD0		FC	KBP	
CD	BBL	13	DD	LDM	13	ED	RD1		FD	DCL	
CE	BBL	14	DE	LDM	14	EE	RD2		FE	-	
CF	BBL	15	DF	LDM	15	EF	RD3		FF	-	

0x00 NOP, žádná operace

kód: 0000 0000
zápis: NOP
počet bajtů: 1
počet strojových cyklů: 1

0x01 HLT, zastavení a čekání na přerušení

kód: 0000 0001
zápis: HLT
počet bajtů: 1

Pouze procesor 4040. V emulátoru nepodporováno.

0x02 BBS, návrat z přerušení

kód: 0000 0010
zápis: BBS
počet bajtů: 1

Pouze procesor 4040. V emulátoru nepodporováno

0x03 LCR, načtení DCL registru do ACC

kód: 0000 0011
zápis: LCR
počet bajtů: 1
počet strojových cyklů: 1
operace: (DCL) -> ACC

Instrukce LCR (Load Command Register) načte DCL registr (4 bity) do akumulátoru.

Pouze procesor 4040.

0x04 OR4, logické OR registru 4 a akumulátoru

kód: 0000 0100
zápis: OR4
počet bajtů: 1
počet strojových cyklů: 1
operace: (reg4) OR (ACC) -> ACC

Instrukce provede bitovou operaci OR mezi indexovým registrem 4 (4 bity) a akumulátorem. Výsledek je uložen do akumulátoru. Registr 4 zůstane nezměněn. Příznak Carry zůstane neovlivněn.

Pouze procesor 4040.

0x05 OR5, logické OR registru 5 a akumulátoru

kód: 0000 0101
zápis: OR5
počet bajtů: 1
počet strojových cyklů: 1
operace: (reg5) OR (ACC) -> ACC

Instrukce provede bitovou operaci OR mezi indexovým registrem 5 (4 bity) a akumulátorem. Výsledek je uložen do akumulátoru. Registr 5 zůstane nezměněn. Příznak Carry zůstane neovlivněn.

Pouze procesor 4040.

0x06 AN6, logické AND registru 6 a akumulátoru

kód: 0000 0110
zápis: AN6
počet bajtů: 1
počet strojových cyklů: 1
operace: (reg6) AND (ACC) -> ACC

Instrukce provede bitovou operaci AND mezi indexovým registrem 6 (4 bity) a akumulátorem. Výsledek je uložen do akumulátoru. Registr 6 zůstane nezměněn. Příznak Carry zůstane neovlivněn.

Pouze procesor 4040.

0x07 AN7, logické AND registru 7 a akumulátoru

kód: 0000 0111
zápis: AN7
počet bajtů: 1
počet strojových cyklů: 1
operace: (reg7) AND (ACC) -> ACC

Instrukce provede bitovou operaci AND mezi indexovým registrem 7 (4 bity) a akumulátorem. Výsledek je uložen do akumulátoru. Registr 7 zůstane nezměněn. Příznak Carry zůstane neovlivněn.

Pouze procesor 4040.

0x08 DB0, aktivace ROM banky 0

kód: 0000 1000
zápis: DB0
počet bajtů: 1

počet strojových cyklů: 1

Pouze procesor 4040. V emulátoru nepodporováno.

0x09 DB1, aktivace ROM banky 1

kód: 0000 1001
zápis: DB1
počet bajtů: 1
počet strojových cyklů: 1

Pouze procesor 4040. V emulátoru nepodporováno.

0x0A SB0, aktivace banky indexových registrů 0

kód: 0000 1010
zápis: SB0
počet bajtů: 1
počet strojových cyklů: 1

Instrukce SB0 (Select Index Register Bank 0) aktivuje základní banku indexových registrů 0, tj. registry R0 až R7. Registry R8 až R15 zůstávají aktivní nezávisle na vybrané bance. Banka 0 je vybrána automaticky při resetu procesoru.

Pouze procesor 4040.

0x0B SB1, aktivace banky indexových registrů 1

kód: 0000 1011
zápis: SB1
počet bajtů: 1
počet strojových cyklů: 1

Instrukce SB1 (Select Index Register Bank 1) aktivuje alternativní banku indexových registrů 1, tj. registry R0' až R7'. Registry R8 až R15 zůstávají aktivní nezávisle na vybrané bance.

Pouze procesor 4040.

0x0C EIN, povolení přerušení

kód: 0000 1100
zápis: EIN
počet bajtů: 1
počet strojových cyklů: 1

Pouze procesor 4040. V emulátoru nepodporováno

0x0D DIN, zákaz přerušení

kód: 0000 1101
zápis: EIN
počet bajtů: 1
počet strojových cyklů: 1

Pouze procesor 4040. V emulátoru nepodporováno

0x0E RPM, čtení programové paměti

kód: 0000 1110
zápis: RPM
počet bajtů: 1

Pouze procesor 4040. V emulátoru nepodporováno

0x1n JCN, podmíněný skok (JT, JC, JZ, JR, JNT, JNC, JNZ)

kód: 0001 cccc aaaa aaaa
zápis: JCN c,a (JT a, JC a, JZ a, JR a, JNT a, JNC a, JNZ a)
počet bajtů: 2
počet strojových cyklů: 2
operace: IF (c) THEN a -> PCL

Instrukce JCN (Jump Conditional) provede podmíněný skok v případě splnění podmínky. Je-li podmínka splněna, provede se skok na 8-bitovou adresu 'a' načtenou z druhého bajtu instrukce. Vyšší bajt adresy PCH zůstane nezměněn, tj. skok se provede v rámci stejné 256-B stránky ROM

paměti, jako je ta kde se nachází instrukce JCN. Není-li podmínka splněna, program pokračuje následující instrukcí.

Podmínka 'c' se skládá ze 4 bitů:

bit 3: 1=inverze podmínky
bit 2: 1=obsah akumulátoru A je 0
bit 1: 1=carry flag C je 1
bit 0: 1=test signal T je 0

Více podmínek se kombinuje operací OR.

c	bit 3	bit 2	bit 1	bit 0		význam
0	0	0	0	0		neprovede se
1	0	0	0	1	JT a	T=0
2	0	0	1	0	JC a	C=1
3	0	0	1	1		T=0 or C=1
4	0	1	0	0	JZ a	A=0
5	0	1	0	1		T=0 or A=0
6	0	1	1	0		C=1 or A=0
7	0	1	1	1		T=0 or C=1 or A=0
8	1	0	0	0	JR a	nepodmíněný skok
9	1	0	0	1	JNT a	T=1
10	1	0	1	0	JNC a	C=0
11	1	0	1	1		T=1 and C=0
12	1	1	0	0	JNZ a	A<>0
13	1	1	0	1		T=1 and A<>0
14	1	1	1	0		C=0 and A<>0
15	1	1	1	1		T=1 and C=0 and A<>0

Poznámka: Pokud instrukce JCN začíná na offsetu 254 nebo 255 na stránce ROM, skok je proveden v rámci následující stránky ROM (tj. uplatní se stav PC za koncem instrukce).

0x2n (sudá) FIM, načtení konstanty

kód: 0010 rrr0 dddd dddd
zápis: FIM rr,d
počet bajtů: 2
počet strojových cyklů: 2
operace: dH -> rL, dL -> rH

Instrukce FIM (Fetched Immediate from ROM) uloží do registrového páru konstantu z druhého bajtu instrukce. Do prvního (nižšího, sudého) registru uloží vyšší 4 bity dat, do druhého (vyššího, lichého) uloží nižší 4 bity.

0x2n (lichá) SRC, odeslání řídicí adresy

kód: 0010 rrr1
zápis: SRC rr
počet bajtů: 1
počet strojových cyklů: 1
operace: rLrHL -> adr

Instrukce SRC (Send Register Control) odešle na adresovou sběrnici obsah registrového páru. Význam závisí na typu následující instrukce:

čtení a zápis RAM dat:

bit 0..3: výběr 4-bitové slabiky 0 až 15 v registru
bit 4..5: výběr registru 0 až 3 v RAM čipu
bit 6..7: výběr RAM čipu 0..3

čtení a zápis RAM stavové slabiky:

bit 4..5: výběr registru 0 až 3 v RAM čipu
bit 6..7: výběr RAM čipu 0..3

zápis do RAM portu:

bit 6..7: výběr RAM čipu 0..3

čtení a zápis ROM portu:

bit 4..7: výběr ROM čipu 0..15

Spolu s instrukcí SRC se uplatní i registr DCL, který vybere RAM banku 0..7.

0x3n (sudá) FIN, nepřímé čtení z ROM

kód: 0011 rrr0
zápis: FIN rr

počet bajtů: 1
počet strojových cyklů: 2
operace: ROM(r0r1) -> rLrH

Instrukce FIN (Fetch Indirect from ROM) použije registrový pár R0R1 jako nižší bajt (8 bitů) adresy v aktuální ROM stránce, načte z ROM bajt a uloží ho do zadaného registrového páru. Do prvního (nižšího, sudého) registru uloží vyšší 4 bity dat, do druhého (vyššího, lichého) uloží nižší 4 bity. Obsah registrového páru R0R1 zůstane nezměněn, pokud nebude současně cílovým registrovým párem.

Poznámka: Pokud instrukce FIN začíná na offsetu 255 na stránce ROM, data se načtou z následující stránky ROM (tj. uplatní se stav PC za koncem instrukce).

Poznámka 2: Ačkoliv je instrukce FIN dlouhá 1 bajt, její provedení zabere 2 strojové cykly.

0x3n (lichá) JIN, nepřímý skok

kód: 0011 rrr1
zápis: JIN rr
počet bajtů: 1
počet strojových cyklů: 1
operace: (rLrH) -> PCL

Instrukce JIN (Jump Indirect) načte obsah zadaného registrového páru do nižšího bajtu čítače instrukcí a provede skok na danou adresu.

Poznámka: Pokud instrukce JIN začíná na offsetu 255 na stránce ROM, skok se provede v rámci následující stránky ROM (tj. uplatní se stav PC za koncem instrukce).

0x4n JUN (JMP), nepodmíněný skok

kód: 0100 aaaa aaaa aaaa
zápis: JUN a (JMP a)
počet bajtů: 2
počet strojových cyklů: 2
operace: a -> PC

Instrukce JUN (Jump Unconditional) provede nepodmíněný skok na zadanou adresu. Na rozdíl od jiných skokových instrukcí, cílová adresa pokrývá celý rozsah ROM (12 bitů, tj. 4096 bajtů).

0x5n JMS (CALL), skok do podprogramu

kód: 0101 aaaa aaaa aaaa
zápis: JMS a (CALL a)
počet bajtů: 2
počet strojových cyklů: 2
operace: (PC) -> Stack, a -> PC

Instrukce JMS (Jump to Subroutine) uloží do zásobníku adres návratovou adresu (tj. adresu následující za instrukcí JMS) a provede nepodmíněný skok na zadanou adresu. Cílová adresa pokrývá celý rozsah ROM (12 bitů, tj. 4096 bajtů).

Zásobník adres má kapacitu 4 registry (procesor 4004) nebo 8 registrů (procesor 4040 a emulátor). Aktuálně vybraný registr zásobníku (na který ukazuje ukazatel zásobníku) je použit jako čítač programu PC, proto instrukce JMS může uložit 3 nebo 7 návratových adres. Emulátor AT4004 nepoužívá registr zásobníku jako programový čítač PC, zásobník má kapacitu 8 návratových adres.

K návratu z podprogramu slouží instrukce BBL (RET).

0x6n INC, inkrementace registru

kód: 0110 rrrr
zápis: INC r
počet bajtů: 1
počet strojových cyklů: 1
operace: (rrrr) + 1 -> rrrr

Instrukce INC (Increment) inkrementuje (tj. zvýší o 1) obsah indexového 4-bitového registru. Příznak Carry zůstane nezměněn.

0x7n ISZ (IJNZ), inkrementace a přeskočení při nule

kód: 0111 rrrr aaaa aaaa

zápis: ISZ r,a (IJNZ r,a)
počet bajtů: 2
počet strojových cyklů: 2
operace: (rrrr) + 1 -> rrrr, když výsledek 0: a -> PC

Instrukce ISZ (Increment and Skip if Zero, alternativa IJNZ=Increment and Jump if Not Zero) inkrementuje (tj. zvýší o 1) obsah indexového 4-bitového registru. Je-li výsledek nula, přeskočí se bajt adresy a provede se následující instrukce. Není-li výsledek inkrementace nula, provede se skok na zadanou adresu 'a' v rámci aktuální 256-B stránky ROM. Příznak Carry zůstane nezměněn.

Poznámka: Pokud instrukce ISZ začíná na offsetu 254 nebo 255 na stránce ROM, skok je proveden v rámci následující stránky ROM (tj. uplatní se stav PC za koncem instrukce).

0x8n ADD, přičtení registru a carry k akumulátoru

kód: 1000 rrrr
zápis: ADD r
počet bajtů: 1
počet strojových cyklů: 1
operace: (rrrr) + (CY) + (Acc) -> Acc, CY

Instrukce ADD sečte akumulátor s obsahem 4-bitového indexového registru a s Carry, výsledek uloží do akumulátoru. Příznak Carry bude nastaven na 1 v případě výsledku většího než 15 (přetečení). Nedojde-li k přetečení, příznak Carry se vynuluje.

Příklad: reg:1010 (10) + c:0 + Acc:0111 (7) -> c:1, Acc:0001 (1)

0x9n SUB, odečtení registru a carry od akumulátoru

kód: 1001 rrrr
zápis: SUB r
počet bajtů: 1
počet strojových cyklů: 1
operace: (/rrrr) + (/CY) + (Acc) -> Acc, CY

Instrukce SUB sečte akumulátor s invertovaným obsahem 4-bitového indexového registru a s invertovaným Carry, výsledek uloží do

akumulátoru. Příznak Carry bude nastaven na 1 v případě výsledku většího než 15 (přetečení). Nedojde-li k přetečení, příznak Carry se vynuluje. Instrukce má význam odečtení registru a carry, carry bude na výstupu obsahovat invertovaný příznak podtečení (/borrow).

Příklad:

Acc = 0011 (3)
C = 0 ... /C = 1
reg = 0101 (5) ... /reg = 1010 (10)
reg:1010 (10) + c:1 + Acc:0011 (3) -> c:0, Acc:1110 (14)

0xAn LD, načtení registru do akumulátoru

kód: 1010 rrrr
zápis: LD r
počet bajtů: 1
počet strojových cyklů: 1
operace: (rrrr) -> Acc

Instrukce LD (Load index register) načte obsah 4-bitového indexového registru do akumulátoru.

0xBn XCH, záměna registru a akumulátoru

kód: 1011 rrrr
zápis: XCH r
počet bajtů: 1
počet strojových cyklů: 1
operace: (rrrr) -> Acc, (Acc) -> rrrr

Instrukce XCH (Exchange) zamění obsah 4-bitového indexového registru a akumulátoru.

0xCn BBL (RET), návrat z podprogramu

kód: 1100 dddd
zápis: BBL d (RET d)
počet bajtů: 1

počet strojových cyklů: 1
operace: (Stack) -> PC, d -> Acc

Instrukce BBL (Branch Back and Load Data) navrátí programový čítač PC ze zásobníku adres a načte konstantu do akumulátoru. Instrukce slouží k návratu z podprogramu.

0xDn LDM (LDI), načtení konstanty do akumulátoru

kód: 1101 dddd
zápis: LDM d (LDI d)
počet bajtů: 1
počet strojových cyklů: 1
operace: d -> Acc

Instrukce LDM (Load Data to Accumulator) načte 4-bitovou konstantu do akumulátoru.

0xE0 WRM, zápis do RAM paměti

kód: 1110 0000
zápis: WRM
počet bajtů: 1
počet strojových cyklů: 1
operace: (Acc) -> M

Instrukce WRM (Write Accumulator into RAM Memory) uloží obsah akumulátoru do 4-bitové slabiky RAM. Adresa slabiky je určena registry SRC a DCL.

0xE1 WMP, zápis do RAM portu

kód: 1110 0001
zápis: WMP
počet bajtů: 1
počet strojových cyklů: 1
operace: (Acc) -> RAM port

Instrukce WMP (Write Memory Port) odešle obsah akumulátoru na 4-bitový výstupní port RAM. Adresa portu je určena registry SRC a DCL.

0xE2 WRR, zápis do ROM portu

kód: 1110 0010
zápis: WRR
počet bajtů: 1
počet strojových cyklů: 1
operace: (Acc) -> ROM port

Instrukce WRR (Write ROM Port) odešle obsah akumulátoru na 4-bitový výstupní port ROM. Adresa portu je určena registrem SRC.

0xE3 WPM, zápis do programové paměti

kód: 1110 0011
zápis: WPM
počet bajtů: 1
počet strojových cyklů: 1

V emulátoru nepodporováno.

0xE4 WR0, zápis do RAM stavového registru 0

kód: 1110 0100
zápis: WR0
počet bajtů: 1
počet strojových cyklů: 1
operace: (Acc) -> S0

Instrukce WR0 (Write Accumulator into RAM Status Register 0) uloží obsah akumulátoru do 4-bitového RAM stavového registru 0. Adresa registru je určena registry SRC a DCL.

0xE5 WR1, zápis do RAM stavového registru 1

kód: 1110 0101
zápis: WR1
počet bajtů: 1
počet strojových cyklů: 1
operace: (Acc) -> S1

Instrukce WR1 (Write Accumulator into RAM Status Register 1) uloží obsah akumulátoru do 4-bitového RAM stavového registru 1. Adresa registru je určena registry SRC a DCL.

0xE6 WR2, zápis do RAM stavového registru 2

kód: 1110 0110
zápis: WR2
počet bajtů: 1
počet strojových cyklů: 1
operace: (Acc) -> S2

Instrukce WR2 (Write Accumulator into RAM Status Register 2) uloží obsah akumulátoru do 4-bitového RAM stavového registru 2. Adresa registru je určena registry SRC a DCL.

0xE7 WR3, zápis do RAM stavového registru 3

kód: 1110 0111
zápis: WR3
počet bajtů: 1
počet strojových cyklů: 1
operace: (Acc) -> S3

Instrukce WR3 (Write Accumulator into RAM Status Register 3) uloží obsah akumulátoru do 4-bitového RAM stavového registru 3. Adresa registru je určena registry SRC a DCL.

0xE8 SBM, odečtení RAM paměti

kód: 1110 1000
zápis: SBM
počet bajtů: 1
počet strojových cyklů: 1
operace: (/M) + (/C) + (Acc) -> C,Acc

Instrukce SBM sečte akumulátor s invertovaným obsahem 4-bitové slabiky RAM a s invertovaným Carry, výsledek uloží do akumulátoru. Příznak Carry bude nastaven na 1 v případě výsledku většího než 15 (přetečení). Nedojde-li k přetečení, příznak Carry se vynuluje. Instrukce má význam

odečtení slabiky a carry, carry bude na výstupu obsahovat invertovaný příznak podtečení (/borrow). Adresa slabiky je určena registry SRC a DCL.

0xE9 RDM, načtení RAM paměti

kód: 1110 1001
zápis: RDM
počet bajtů: 1
počet strojových cyklů: 1
operace: (M) -> Acc

Instrukce RDM (Read RAM Memory) načte obsah 4-bitové slabiky RAM do akumulátoru. Adresa slabiky je určena registry SRC a DCL.

0xEA RDR, načtení ROM portu

kód: 1110 1010
zápis: RDR
počet bajtů: 1
počet strojových cyklů: 1
operace: (ROM port) -> Acc

Instrukce RDR (Read ROM Port) načte 4-bitový vstupní port ROM do akumulátoru. Adresa portu je určena registrem SRC.

0xEB ADM, přičtení RAM paměti

kód: 1110 1011
zápis: ADM
počet bajtů: 1
počet strojových cyklů: 1
operace: (M) + (C) + (Acc) -> C,Acc

Instrukce ADM sečte akumulátor s obsahem 4-bitové slabiky RAM a s Carry, výsledek uloží do akumulátoru. Příznak Carry bude nastaven na 1 v případě výsledku většího než 15 (přetečení). Nedojde-li k přetečení, příznak Carry se vynuluje. Adresa slabiky je určena registry SRC a DCL.

0xEC RD0, načtení RAM stavového registru 0

kód: 1110 1100
zápis: RD0
počet bajtů: 1
počet strojových cyklů: 1
operace: (S0) -> Acc

Instrukce RD0 (Read RAM Status Register 0) načte obsah 4-bitového RAM stavového registru 0 do akumulátoru. Adresa registru je určena registry SRC a DCL.

0xED RD1, načtení RAM stavového registru 1

kód: 1110 1101
zápis: RD1
počet bajtů: 1
počet strojových cyklů: 1
operace: (S1) -> Acc

Instrukce RD1 (Read RAM Status Register 1) načte obsah 4-bitového RAM stavového registru 1 do akumulátoru. Adresa registru je určena registry SRC a DCL.

0xEE RD2, načtení RAM stavového registru 2

kód: 1110 1110
zápis: RD2
počet bajtů: 1
počet strojových cyklů: 1
operace: (S2) -> Acc

Instrukce RD2 (Read RAM Status Register 2) načte obsah 4-bitového RAM stavového registru 2 do akumulátoru. Adresa registru je určena registry SRC a DCL.

0xEF RD3, načtení RAM stavového registru 3

kód: 1110 1111

zápis: RD3
počet bajtů: 1
počet strojových cyklů: 1
operace: (S3) -> Acc

Instrukce RD3 (Read RAM Status Register 3) načte obsah 4-bitového RAM stavového registru 3 do akumulátoru. Adresa registru je určena registry SRC a DCL.

0xF0 CLB, vymazání obou

kód: 1111 0000
zápis: CLB
počet bajtů: 1
počet strojových cyklů: 1
operace: 0 -> Acc, 0 -> CY

Instrukce CLB (Clear Both) vynuluje akumulátor a příznak Carry.

0xF1 CLC, vynulování carry

kód: 1111 0001
zápis: CLC
počet bajtů: 1
počet strojových cyklů: 1
operace: 0 -> CY

Instrukce CLC (Clear Carry) vynuluje příznak Carry.

0xF2 IAC (INC A), inkrementace akumulátoru

kód: 1111 0010
zápis: IAC (INC A)
počet bajtů: 1
počet strojových cyklů: 1
operace: (Acc) + 1 -> Acc

Instrukce IAC (Increment Accumulator) inkrementuje (zvýší o 1) akumulátor. Příznak Carry bude nastaven na 1 v případě výsledku většího než 15 (přetečení). Nedojde-li k přetečení, příznak Carry se vynuluje.

0xF3 CMC, komplement carry

kód: 1111 0011
zápis: CMC
počet bajtů: 1
počet strojových cyklů: 1
operace: (/CY) -> CY

Instrukce CMC (Complement Carry) invertuje příznak Carry.

0xF4 CMA, komplement akumulátoru

kód: 1111 0100
zápis: CMA
počet bajtů: 1
počet strojových cyklů: 1
operace: (/Acc) -> Acc

Instrukce CMA (Complement Accumulator) invertuje obsah akumulátoru. Příznak Carry zůstane nezměněn.

0xF5 RAL, rotace vlevo

kód: 1111 0101
zápis: RAL
počet bajtů: 1
počet strojových cyklů: 1
operace: C -> a0 .. a3 -> C

Instrukce RAL (Rotate Left) rotuje obsah akumulátoru doleva, přes příznak Carry.

0xF6 RAR, rotace vpravo

kód: 1111 0110
zápis: RAR
počet bajtů: 1
počet strojových cyklů: 1
operace: C -> a3 .. a0 -> C

Instrukce RAR (Rotate Right) rotuje obsah akumulátoru doprava, přes příznak Carry.

0xF7 TCC, přenos carry

kód: 1111 0111
zápis: TCC
počet bajtů: 1
počet strojových cyklů: 1
operace: 0 -> Acc, (CY) -> a0, 0 -> CY

Instrukce TCC (Transmit Carry and Clear) vynuluje akumulátor, příznak Carry načte do bitu 0 akumulátoru a příznak Carry vynuluje.

0xF8 DAC (DEC A), dekrementace akumulátoru

kód: 1111 1000
zápis: DAC (DEC A)
počet bajtů: 1
počet strojových cyklů: 1
operace: (Acc) - 1 -> C,Acc

Instrukce DAC (Decrement Accumulator) dekrementuje (sníží o 1) akumulátor. V případě podtečení výsledku pod 0 vynuluje příznak Carry. Není-li podtečení, příznak Carry se nastaví na 1. Carry má zde význam invertovaného podtečení (/borrow).

Instrukce se provede přičtením hodnoty 15 k akumulátoru. Je-li výsledek větší než 15 (není podtečení, tj. obsah akumulátoru před operací byl > 0), nastaví se příznak Carry. V opačném případě se Carry vynuluje.

0xF9 TCS, přenos carry odečtení

kód: 1111 1001
zápis: TCS
počet bajtů: 1
počet strojových cyklů: 1
operace: je-li CY = 0: 9 -> Acc, je-li CY = 1: 10 -> Acc, 0 -> CY

Instrukce TCS (Transfer Carry Subtract) načte do akumulátoru negovaný příznak Carry. Byl-li před operací Carry nulový, načte do akumulátoru hodnotu 9. Byl-li Carry nastaven na 1, načte do akumulátoru hodnotu 10. Na závěr příznak Carry vynuluje.

0xFA STC, nastavení carry

kód: 1111 1010
zápis: STC
počet bajtů: 1
počet strojových cyklů: 1
operace: 1 -> CY

Instrukce STC (Set Carry) nastaví příznak Carry.

0xFB DAA, dekadická korekce

kód: 1111 1011
zápis: DAA
počet bajtů: 1
počet strojových cyklů: 1
operace: je-li CY = 1 nebo (Acc) > 9: (Acc) + 6 -> C, Acc

Instrukce DAA zvýší hodnotu akumulátoru o 6 v případě, je-li Carry nastavený na 1 nebo je-li hodnota akumulátoru větší než 9. V případě přetečení výsledku se nastaví Carry na 1. V opačném případě zůstane Carry nezměněn.

0xFC KBP, korekce kláves

kód: 1111 1100
zápis: KBP
počet bajtů: 1
počet strojových cyklů: 1
operace: převede bit v Acc na jeho pozici

Instrukce KBP převede bit 0 až 3 v akumulátoru na číslo jeho bitové pozice 1 až 4. Není-li žádný z bitů nastaven na '1', výsledkem bude hodnota 0. Je-li nastaveno více bitů na hodnotu '1' než 1, bude výsledkem 15.

Acc před operací	Acc po operaci
0 0 0 0 (0)	0 0 0 0 (0)
0 0 0 1 (1)	0 0 0 1 (1)
0 0 1 0 (2)	0 0 1 0 (2)
0 0 1 1 (3)	1 1 1 1 (15)
0 1 0 0 (4)	0 0 1 1 (3)
0 1 0 1 (5)	1 1 1 1 (15)
0 1 1 0 (6)	1 1 1 1 (15)
0 1 1 1 (7)	1 1 1 1 (15)
1 0 0 0 (8)	0 1 0 0 (4)
1 0 0 1 (9)	1 1 1 1 (15)
1 0 1 0 (10)	1 1 1 1 (15)
1 0 1 1 (11)	1 1 1 1 (15)
1 1 0 0 (12)	1 1 1 1 (15)
1 1 0 1 (13)	1 1 1 1 (15)
1 1 1 0 (14)	1 1 1 1 (15)
1 1 1 1 (15)	1 1 1 1 (15)

0xFD DCL, řídicí registr adresy

kód: 1111 1101
zápis: DCL
počet bajtů: 1
počet strojových cyklů: 1
operace: (Acc) -> CM

Instrukce DCL odešle obsah akumulátoru na adresovou sběrnici. Spolu s registrem SRC slouží k adresování RAM paměti.

Není-li k procesoru připojen dekodér adres, slouží spodní 3 bity k přímému výběru paměťových bank RAM. Hodnota X001 vybere banku 1, hodnota X010 vybere banku 2 a hodnota X100 vybere banku 3. Není-li žádný z bitů nastaven (hodnota X000), je vybrána banka 0.

Je-li k procesoru připojen dekodér adres, vybere hodnota X000 až X111 paměťovou banku 0 až 7.

Nejvyšší bit 3 registru DCL není použit k výběru bank RAM. U procesoru 4040 je bit 3 použit k výběru banky ROM0 nebo ROM1 (v emulátoru nepodporováno).

6. Assembler AS4

Součástí kitu AT4004 je překladač AS4 sloužící k překladu zdrojového kódu instrukcí assembleru procesoru Intel 4004 a 4040 do binárního kódu.

Překladač je kompatibilní s původní syntaxí zápisu instrukcí procesoru, ale doplňuje několik rozšíření usnadňujících zápis programu:

.strict	Přísný mód, registry musí mít jméno R0 až R15 a registrové páry musí mít jméno R01 až R1415. Přepínač .strict zabraňuje častému omylu při zápisu instrukcí, kdy si programátor myslí že používá konstantu a překladač očekává registr.
.page [fill]	Zarovnání kódu na 256-bajtovou stránku. Lze zadat bajt výplně, jinak je implicitně vyplněno bajtem 0x00.
***=addr[,fill]**	Nastavení ukazatele adres. Lze zadat bajt výplně, jinak je implicitně vyplněno bajtem 0x00.
.set a b	Symbol 'a' bude nahrazen textem 'b' - může to být text nebo číslo, ale nesmí obsahovat mezery.
R01	Registrový pár lze zadávat jménem R01, R23, R45, R67, R89, R1011, R1213 nebo R1415.
@label	Nižší bajt adresy bude použit jako datový bajt.
-number	Před datovým bajtem lze uvést znaménko mínus.
'A'	Jako datový bajt lze uvést ASCII znak (v uvozovkách).
$12, 0x12, 12h	HEX bajt může být zadán s $, 0x nebo H.
par,par2	Parametry mohou být odděleny čárkami.
= a,b	Konstantní data mohou mít 1 nebo 2 bajty.

Kromě standardního zápisu instrukcí lze použít alternativy, které mohou přispět k přehlednějšímu kódu:

JCN 0001,a ... JCN 1,a ... JCN TZ,a ... JT a ... TEST signál je aktivní
JCN 1001,a ... JCN 9,a ... JCN TN,a ... JNT a ... TEST signál není aktivní
JCN 0010,a ... JCN 2,a ... JCN C1,a ... JC a ... carry je nastavený
JCN 1010,a ... JCN 10,a ... JCN C0,a ... JNC a ... carry není nastavení
JCN 0100,a ... JCN 4,a ... JCN AZ,a ... JZ a ... ACC je nula
JCN 1100,a ... JCN 12,a ... JCN AN,a ... JNZ a ... ACC není nula
JCN 1000,a ... JCN 8,a ... JR a ... relativní skok, nepodmínění krátký skok

JUN a ... JMP a ... dlouhý skok
JMS a ... CALL a ... skok do podprogramu
ISZ r,a ... IJNZ r,a ... "Increment and Jump if Not Zero"
IAC ... INC A ... inkrementace akumulátoru
DAC ... DEC A ... dekrementace akumulátoru
BBL n ... RET n ... návrat z podprogramu
LDM d ... LDI d ... "Load Immediate", načtení konstanty

Symbolika syntaxe instrukcí:

d ... 8-bit number (byte)
n ... 4-bit number (nibble)
s ... 8-bit address (in current page)
a ... 12-bit address
r ... register
rr ... register pair
c ... condition (4-bit number)
C ... carry
A ... accumulator

Skupina instrukcí 0x00..0x0F:

NOP ... no operation
HLT ... stop (only I4040)
BBS ... return from interrupt (only I4040)
LCR ... load DCL into A (only I4040)
OR4 ... OR r4 to A (only I4040)
OR5 ... OR r5 to A (only I4040)
AN6 ... AND r6 to A (only I4040)
AN7 ... AND r7 to A (only I4040)
DB0 ... select ROM bank 0 (only I4040)

DB1 ... select ROM bank 1 (only I4040)
SB0 ... select register bank 0 (only I4040)
SB1 ... select register bank 1 (only I4040)
EIN ... enable interrupt (only I4040)
DIN ... disable interrupt (only I4040)
RPM ... read program memory (only I4040)

Skupina instrukcí 0x10..0xDF:

JCN n,s ... jump with condition
    JT s ... jump if TEST is active (TEST=0)
    JNT s .. jump if TEST is not active (TEST=1)
    JC s ... jump if carry is set
    JNC s ... jump if carry is not set
    JZ s ... jump if A is zero
    JNZ s ... jump if A is not zero
    JR s ... jump relative (unconditional short jump)
FIM rr,d ... fetch immediate data byte d into register pair rr
SRC rr ... send register control from register pair
FIN rr ... fetch indirect data byte from ROM address R0R1 into rr
JIN rr ... jump indirect to ROM address rr
JUN a (JMP a) ... jump to address a
JMS a (CALL a) ... call subroutine a
INC r ... increment register r
ISZ r,s (IJNZ r,s) ... increment r and jump if not zero to address s
ADD r ... add register r and C to A (A = A + r + C, C is 1 if carry)
SUB r ... subtract register r and C from A
LD r ... load register r to A
XCH r ... exchange register r and A
BBL n (RET n) ... return from subroutine with code in A
LDM n (LDI n) ... load immediate to A

Skupina instrukcí 0xE0..0xEF:

WRM ... write A into RAM memory
WMP ... write A into RAM port
WRR ... write A into ROM port
WPM ... write A into program half-byte RAM memory
WR0 ... write A into RAM status 0
WR1 ... write A into RAM status 1
WR2 ... write A into RAM status 2
WR3 ... write A into RAM status 3

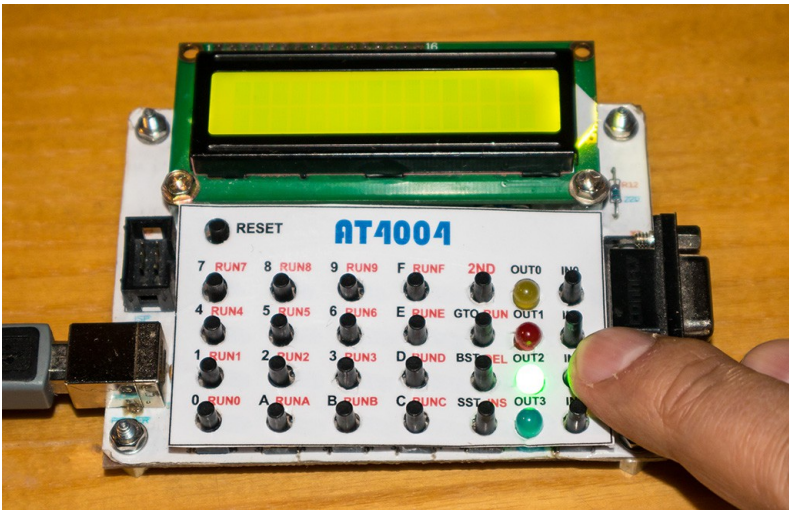
SBM ... subtract RAM and C from A
RDM ... read RAM memory into A
RDR ... read ROM port into A
ADM ... add RAM and C to A
RD0 ... read RAM status 0 into A
RD1 ... read RAM status 1 into A
RD2 ... read RAM status 2 into A
RD3 ... read RAM status 3 into A

Skupina instrukcí 0xF0..0xFF:

CLB ... clear both (A and C)
CLC .. clear C
IAC (INC A) ... increrment A (overflow -> C)
CMC ... complement C
CMA ... complement A
RAL ... rotate A left through C (C<-a3<-a2<-a1<-a0<-C)
RAR ... rotate A right through C (C->a3->a2->a1->a0->C)
TCC ... transmit C to A (0 or 1), clear C
DAC (DEC A) ... decrement A (~borrow -> C)
TCS ... transfer C subtract to A (0->9 or 1->10), clear C
STC ... set carry
DAA ... decimal adjust A
KBP ... keyboard process
DCL ... designate command line

7. Příklad 1 - kopie vstupu na výstup

Jako první příklad si ukážeme jednoduchý program, který nedělá nic jiného, než že opakovaně kopíruje data ze vstupního portu kitu na výstupní port. Program přeložíme od adresy 0xE00, což je začátek editovatelné EEPROM paměti. Překladač vygeneruje soubor listingu *.list, kde vidíme kódy přeložených instrukcí. Kódy přepíšeme do kitu od adresy 0xE00 a spustíme tlačítky **2ND RUNE**. Program testujeme mačkáním tlačítek vstupního portu, stejně tak se musí rozsvěcovat výstupní LED.




```
; Demo - copy buttons to LEDs.

        .strict                 ; strict mode (requires register names,
                                ; not register numbers)

; start address
*=$e00 $ff

; ----- prepare
; DCL register: bit 0..2: RAM bank CM0..CM7

Reset:  clb                     ; clear A and Carry
        dcl                     ; send A (with content 0) to DCL
                                ; - select RAM bank 0 (CM0)

; ----- select ROM3 input port
; SRC register:
```

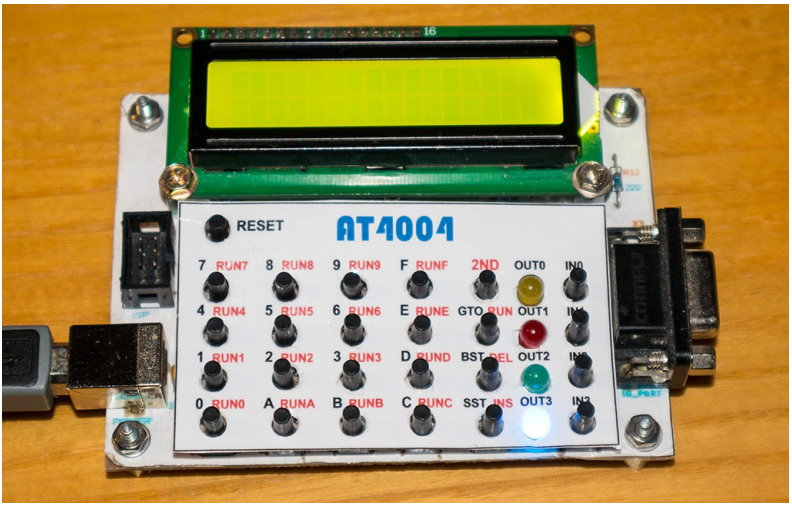
```
;       ROM port:
;               bit 4..7: ROM chip 0..15

; ROM3: emulator inputs
;       bit0 = emulator IN0 input
;       bit1 = emulator IN1 input
;       bit2 = emulator IN2 input
;       bit3 = emulator IN3 input (ADV button)

Loop:   fim     R01,$30         ; prepare address of ROM3 port:
                                ; bit 4..7 <- 3
        src     R01             ; send register pair R0R1 to address
                                ; bus, select ROM3 input port

; ----- read input from test buttons

        rdr                     ; read data from ROM3 input port
                                ; (button: 1=released, 0=pressed)
        cma                     ; complement accumulator

; ----- select RAM1 output port
; SRC register:
;       RAM output port:
;               bit 6..7: RAM chip 0..3

; RAM1: status light outputs
;       bit0 = memory lamp M, emulator OUT0 output, LED1 yellow
;       bit1 = overflow lamp OVF, emulator OUT1 output, LED2 red
;       bit2 = minus sign lamp NEG, emulator OUT2 output, LED3 green
;       bit3 = not used, emulator OUT3 output, LED4 blue

        fim     R01,$40         ; prepare RAM port address:
                                ; bit 6..7 <- 01, select chip RAM1
        src     R01             ; send register pair R0R1 to address bus

; ----- output state to LEDs (LED: 1=light, 0=dark)

        wmp                     ; send A to RAM port RAM1
        jmp     Loop            ; continue main loop
```

Přeložený kód programu:

```
E00 F0          Reset:  clb                     ; clear A and Carry
E01 FD                  dcl                     ; send A to DCL
E02 20 30       Loop:   fim     R01,$30         ; address of ROM3 port
E04 21                  src     R01             ; send R0R1 to SRC
E05 EA                  rdr                     ; read ROM3 port
E06 F4                  cma                     ; complement accumulator
E07 20 40               fim     R01,$40         ; RAM port address
E09 21                  src     R01             ; send R0R1 to SRC
E0A E1                  wmp                     ; send A to RAM port
E0B 4E 02               jmp     Loop            ; continue main loop
```

8. Příklad 2 - blikání s LED

Druhý příklad bliká se 4 výstupními LED, s celkovou periodou 1 sekunda, tedy interval 250 ms na jednu LED. Program přeložíme od adresy 0xE80 a po přepsání do EEPROM spustíme příkazy **GTO E 8 0 2ND RUN**. K výpočtu časování smyček můžeme využít přiložený Excel XLS soubor.




```
; Demo - blinking with LEDs with period 1 second.

        .strict                 ; strict mode (requires register names,
                                ; not register numbers)

; start address
*=$e80 $ff

; ----- select port RAM1
; DCL register: bit 0..2: RAM bank CM0..CM7
; SRC register:
;       RAM output port:
;               bit 6..7: RAM chip 0..3

        clb                     ; clear A and Carry
        dcl                     ; send A (with content 0) to DCL
                                ; - select RAM bank 0 (CM0)
        fim     R01,$40         ; prepare RAM port address:
                                ; bit 6..7 <- 01, select chip RAM1
        src     R01             ; send register pair R0R1 to address bus
```

```
; ----- set LED1 ON
; RAM1: status light outputs
;       bit0 = memory lamp M, emulator OUT0 output, LED1 yellow
;       bit1 = overflow lamp OVF, emulator OUT1 output, LED2 red
;       bit2 = minus sign lamp NEG, emulator OUT2 output, LED3 green
;       bit3 = not used, emulator OUT3 output, LED4 blue

Loop:   ldi     0001            ; load A <- constant 0001 (binary form),
                                ; to set LED1 ON
        wmp                     ; send A to RAM port RAM1
        call    Delay           ; delay 0.25 second

; ----- set LED2 ON

        ldi     0010            ; load A <- constant 0010 (binary form),
                                ; to set LED2 ON
        wmp                     ; send A to RAM port RAM1
        call    Delay           ; delay 0.25 second

; ----- set LED3 ON

        ldi     0100            ; load A <- constant 0100 (binary form),
                                ; to set LED3 ON
        wmp                     ; send A to RAM port RAM1
        call    Delay           ; delay 0.25 second

; ----- set LED4 ON

        ldi     1000            ; load A <- constant 1000 (binary form),
                                ; to set LED4 ON
        wmp                     ; send A to RAM port RAM1
        call    Delay           ; delay 0.25 second

        jmp     Loop            ; continue main loop

; ------------------------------------------------------------------
;                 Delay subroutine - wait 0.25 second
; ------------------------------------------------------------------
; With quartz 744 kHz:
; --------------------
; 1 machine cycle = 10.75 us.

; Required machine cycles:
;   0.25 second/10.75 us = 23256 machine cycles
; Subtract base instructions:
;   23256 - 2 - 2*2 - 1 = 23249 (required cycles per loops)

; inner loop R5 whole delay: 16*2 = 32
; semi-inner loop R4 whole delay: 16*(32+2) = 544
; semi-outer loop R3 whole delay: 16*(544+2) = 8736

; outer loop R2 whole loops: (23249-2)/(8736+2) = 2 ... !!!
; outer loop R2 remains: 23249-2-2*(8736+2) = 5771
; semi-outer loop R3 whole loops: (5771-2)/(544+2) = 10 ... !!!
; semi-outer loop R3 remains: 5771-2-10*(544+2) = 309
; semi-inner loop R4 whole loops: (309-2)/(32+2) = 9 .... !!!
```

```
; semi-inner loop R4 remains: 309-2-9*(32+2) = 1
; inner loop R5 loops: 1/2 = 0, minimum 1 ... !!!
; inner loop R5 remains: 1-1*2 = -1 (1 clock missing)

; R2 register initial value: 16-(2+1) = 13
; R3 register initial value: 16-(10+1) = 5
; R4 register initial value: 16-(9+1) = 6
; R5 register initial value: 16-1 = 15

       ; call  Delay          ; [2]

Delay: fim    R23,$D5         ; [2] prepare outer loop counters
                              ;   (R2 outer loop, R3 semi-outer loop)
       fim    R45,$6F         ; [2] prepare inner loop counters
                              ;   (R4 semi-inner loop, R5 inner loop)
Delay2:ijnz   R5,Delay2       ; [2] R5: first delay 1*2=2,
                              ;   whole delay 16*2=32
       ijnz   R4,Delay2       ; [2] R4: first delay 2+2 + 9*(32+2)
                              ;   = 310, whole delay 16*(32+2)=544
       ijnz   R3,Delay2       ; [2] R3: first delay 310+2 + 10*(544+2)
                              ;   = 5772, whole delay 16*(544+2)=8736
       ijnz   R2,Delay2       ; [2] R2: delay = 5772+2 + 2*(8736+2)
                              ;   = 23250
       ret    0               ; [1]
```

Přeložený kód programu:

```
E80 F0                clb                 ; clear A and Carry
E81 FD                dcl                 ; send A to DCL
E82 20 40             fim    R01,$40      ; RAM port address
E84 21                src    R01          ; send R0R1 to SRC
E85 D1          Loop: ldi    0001         ; load A <- constant 0001
E86 E1                wmp                 ; send A to RAM port RAM1
E87 5E 97             call   Delay        ; delay 0.25 second
E89 D2                ldi    0010         ; load A <- constant 0010
E8A E1                wmp                 ; send A to RAM port RAM1
E8B 5E 97             call   Delay        ; delay 0.25 second
E8D D4                ldi    0100         ; load A <- constant 0100
E8E E1                wmp                 ; send A to RAM port RAM1
E8F 5E 97             call   Delay        ; delay 0.25 second
E91 D8                ldi    1000         ; load A <- constant 1000
E92 E1                wmp                 ; send A to RAM port RAM1
E93 5E 97             call   Delay        ; delay 0.25 second
E95 4E 85             jmp    Loop         ; continue main loop
E97 22 D5       Delay: fim    R23,$D5      ; [2] prepare outer loop
E99 24 6F             fim    R45,$6F      ; [2] prepare inner loop
E9B 75 9B      Delay2:ijnz   R5,Delay2    ; [2] R5: first delay
E9D 74 9B             ijnz   R4,Delay2    ; [2] R4: first delay
E9F 73 9B             ijnz   R3,Delay2    ; [2] R3: first delay
EA1 72 9B             ijnz   R2,Delay2    ; [2] R2: delay
EA3 C0                ret    0            ; [1]
```

9. Příklad 3 - zobrazení textu

Třetí příklad zobrazí na displeji emulátoru text. Program přeložíme od adresy 0xF00 a po přepsání do EEPROM spustíme příkazy **2ND RUNF**.


```
; Demo - output text to display.

        .strict                 ; strict mode (requires register names,
                                ; not register numbers)

; start address
*=$f00 $ff

; ----- prepare

Reset:

; ----- select ROM4 output port
; SRC register:
;       ROM port:
;               bit 4..7: ROM chip 0..15

; ROM4: emulator output current display row
;       bot0: 0=top, 1=bottom
;       Output display row also resets display position
;       to 0 and resets character nibble flip-flop.

        fim     R23,$40         ; prepare address of ROM4 port:
```

```
                                ; bit 4..7 <- 4
        src     R23             ; send register pair R2R3 to address
                                ; bus, select ROM4 output port

; ----- select display rop row

        ldi     0               ; A <- 0 (index of top row)
        wrr                     ; write to ROM port, select top row

; ----- select ROM6 output port
; ROM6: emulator output character to display row at current position
;       bit0..bit3: First output high nibble
;               (it will use temporary buffer), then low nibble.
;       Row position auto incremented by 1.

        fim     R23,$60         ; prepare address of ROM6 port:
                                ; bit 4..7 <- 6
        src     R23             ; send register pair R2R3 to address
                                ; bus, select ROM6 output port

; ----- prepare to output text (here is R3 = 0)

        fim     R01,@Text       ; prepare pointer to output text

; ----- output character to display

Loop:   fin     R45             ; read character from ROM to R4R5
                                ; (pointer R0R1)
        ld      R4              ; A <- R4, high nibble of character
        wrr                     ; write to ROM port, send high nibble
                                ; of character
        ld      R5              ; A <- R5, low nibble of character
        wrr                     ; write to ROM port, send low nibble
                                ; of character

; ----- increase text pointer

        inc     R1              ; increment pointer LOW
        ld      R1              ; load pointer LOW
        jnz     Next            ; skip if pointer LOW not 0
        inc     R0              ; increment pointer HIGH

; ----- next character

Next:   ijnz    R3,Loop         ; increment R3, loop 16x

; ----- stop program

Stop:   jmp     Stop

; ----- output text "* Hello World! *" (16 characters)

Text:
        = '*'
        = ' '
        = 'H'
```

```
        = 'e'
        = 'l'
        = 'l'
        = 'o'
        = ' '
        = 'W'
        = 'o'
        = 'r'
        = 'l'
        = 'd'
        = '!'
        = ' '
        = '*'
```

Přeložený kód programu:

```
F00 22 40            fim    R23,$40        ; address of ROM4 port
F02 23               src    R23            ; send R2R3 to SRC
F03 D0               ldi    0              ; A <- 0
F04 E2               wrr                   ; write to ROM port
F05 22 60            fim    R23,$60        ; address of ROM6 port
F07 23               src    R23            ; send R2R3 to SRC
F08 20 18            fim    R01,@Text      ; pointer to text
F0A 34         Loop: fin    R45            ; read character from ROM
F0B A4               ld     R4             ; A <- R4, high nibble
F0C E2               wrr                   ; send high nibble
F0D A5               ld     R5             ; A <- R5, low nibble
F0E E2               wrr                   ; send low nibble
F0F 61               inc    R1             ; increment pointer LOW
F10 A1               ld     R1             ; load pointer LOW
F11 1C 14            jnz    Next           ; skip if pointer not 0
F13 60               inc    R0             ; increment pointer HIGH
F14 73 0A      Next: ijnz   R3,Loop        ; increment R3, loop 16x
F16 4F 16      Stop: jmp    Stop
F18 2A               = '*'
F19 20               = ' '
F1A 48               = 'H'
F1B 65               = 'e'
F1C 6C               = 'l'
F1D 6C               = 'l'
F1E 6F               = 'o'
F1F 20               = ' '
F20 57               = 'W'
F21 6F               = 'o'
F22 72               = 'r'
F23 6C               = 'l'
F24 64               = 'd'
F25 21               = '!'
F26 20               = ' '
F27 2A               = '*'
```