Jazyk symbolických adres

Proč programovat v JSA

- Pro některé procesory resp. MCU jsou překladače JSA dostupnější.
- Některé překladače vyšších jazyků neumí využít určité speciální vlastnosti procesoru.
- Možnost vytvořit optimalizovaný kód (?).
- „Cvičné důvody“ – programátor se musí důkladně seznámit s daným procesorem.

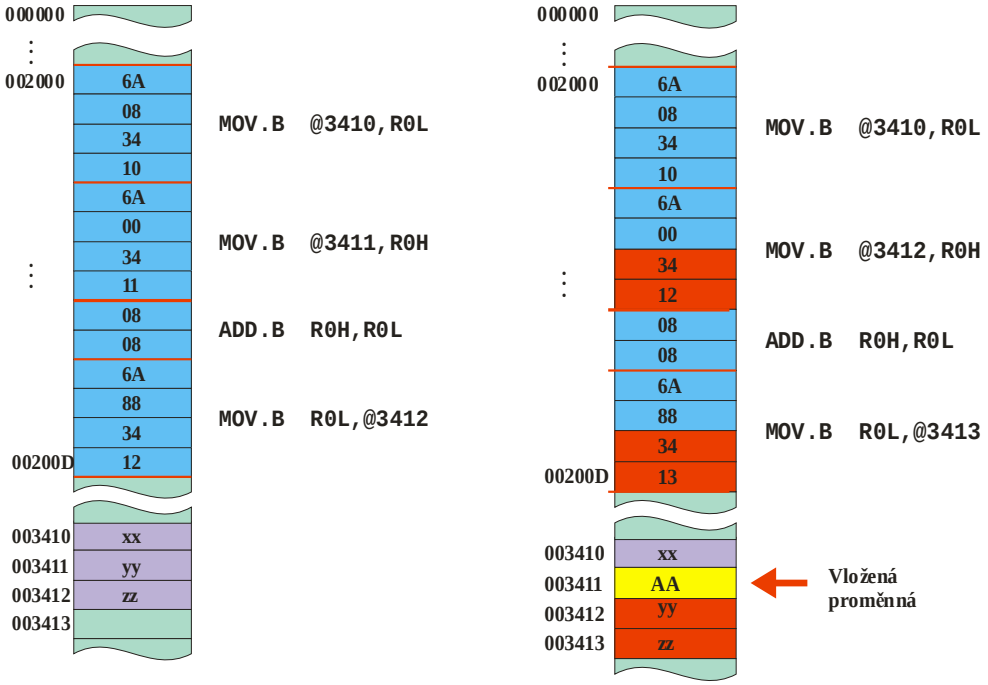
Strojový kód

- Program přeložený do binární podoby.
  - Obsahuje binární kódy instrukcí,
  - Obsahuje absolutní adresy operandů.

⇓

- Jediná forma programu, kterou umí procesor přímo zpracovat.
- Velmi obtížné úpravy programu.
- Obtížně srozumitelná pro programátora.

Obtížné změny ve strojovém kódu

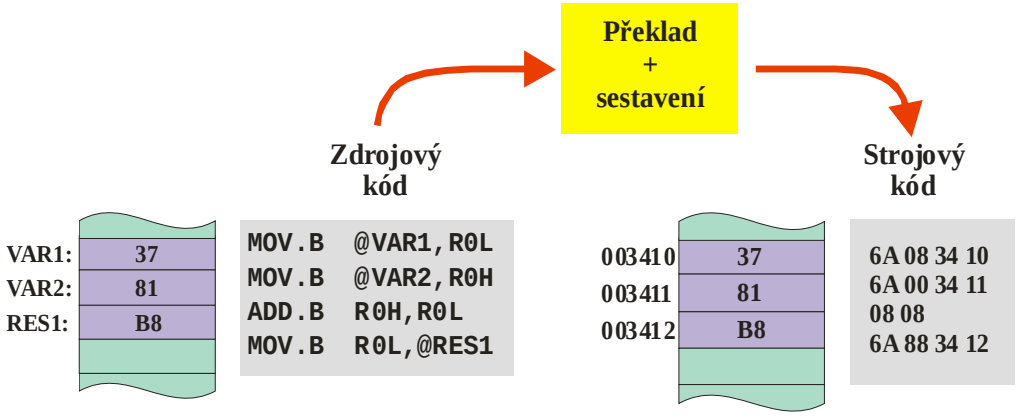


Symbolická adresa (1)

- Symbolická adresa nahrazuje ***ve zdrojovém kódu*** skutečnou (absolutní) adresu.
- Převod symbolická adresa → absolutní adresa provede překladač + sestavovací program.

⇓

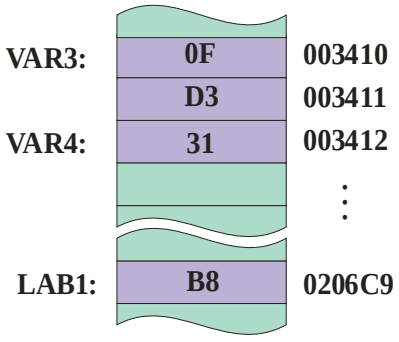
- Programátor nemusí znát skutečné umístění „proměnné“ v paměti.



Symbolická adresa (2)

- Symbolická adresa má
  - **Hodnotu** – odpovídá adrese, kterou reprezentuje.
  - **Obsah** – odpovídá obsahu paměťového místa (bytu, slova, …) na kterou odkazuje.
  - **Typ** – relativní nebo absolutní.

Př:



Symbolická adresa	Hodnota	Obsah
VAR3	003410	0FD3
VAR4	003412	31
LAB1	0206C9	B8

Symbolická adresa (3)

- Symbolická adresa může být
  - **Absolutní** – hodnota je známa při překladu, tj. může ji určit přímo asembler (překladač).
  - **Relativní** – hodnotu určí linker (sestavovací program) při sestavování programu.

- Výrazy se symbolickými adresami

1. operand	2. operand	Operace	Výsledek
Absolutní	Absolutní	±	Absolutní
Relativní	Absolutní	±	Relativní
Relativní	Relativní	+	Relativní
Relativní	Relativní	–	Absolutní*

* uvnitř jednoho segmentu

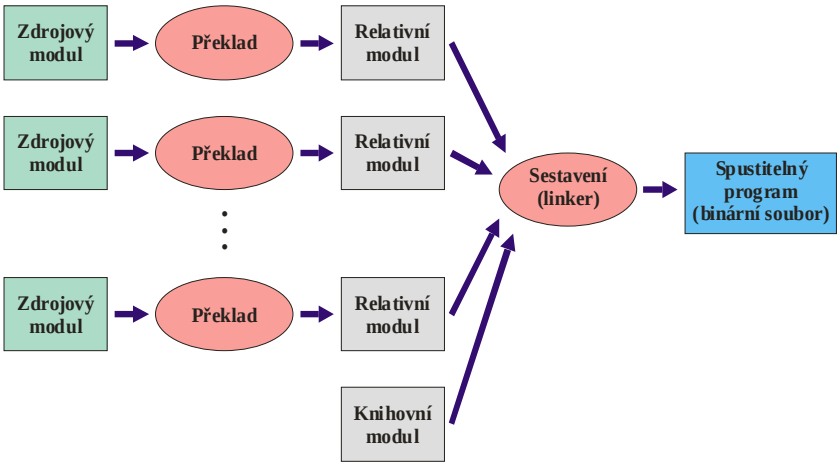
Symbolická adresa (4)

- Použití symbolické adresy
  - **Návěští** – cílová adresa skoku nebo volání procedury.
  - **Proměnná** – adresa pro manipulaci s daty.

```
JMP       @LAB1
....
MOV.W     @VAR3,R1
MOV.L     #VAR4,ER2
```

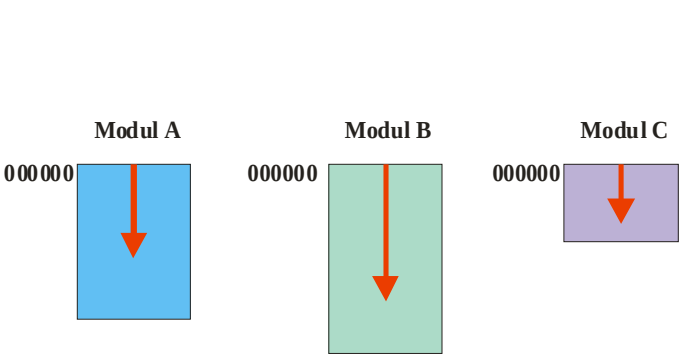
Překlad a sestavení programu

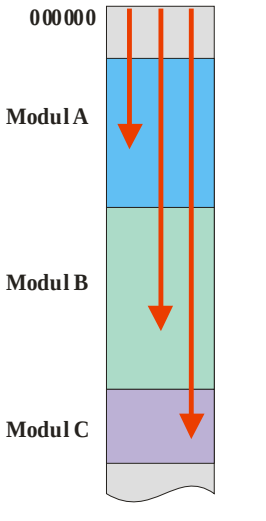
- Program je sestaven z jednoho nebo více modulů.
- Moduly se překládají samostatně.
- Přeložené (relativní) moduly se spojí sestavovacím programem do výsledného souboru.



Relativní a absolutní adresy

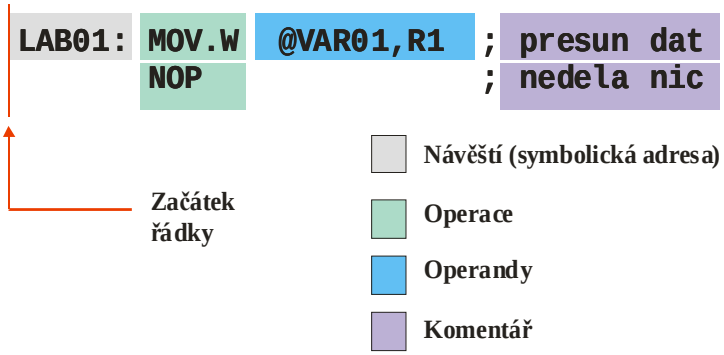
- V relativních modulech jsou adresy počítány od začátku modulu.
- V sestaveném programu jsou adresy počítány od začátku paměti.





Zápis programu

- Program se zapisuje do 4 sloupců.
- Některá pole se mohou podle situace vynechat.
- Každá řádka obsahuje jednu instrukci, direktivu nebo rozvinutí makra.
  - Návěští – definuje symbolickou adresu.
  - Operace – symbolický název **instrukce** nebo **direktivy**.
  - Operandy – operandy instrukce nebo parametry direktivy.
  - Komentář – je oddělen středníkem.



Instrukční soubor

- Instrukce lze podle funkce rozdělit do několika skupin:
  - přesuny dat,
  - aritmetické operace,
  - logické operace,
  - posuvy a rotace,
  - bitové operace,
  - nepodmíněné a podmíněné skoky,
  - řídicí instrukce.

Instrukční soubor H8S (1)

Function	Instruction	Addressing Modes													
		#xx	Rn	@ERn	@(d:16,ERn)	@(d:32,ERn)	@–ERn/@ERn+	@aa:8	@aa:16	@aa:24	@aa:32	@(d:8,PC)	@(d:16,PC)	@@aa:8	–
Data transfer	MOV	BWL	BWL	BWL	BWL	BWL	BWL	B	BWL	—	BWL	—	—	—	—
	POP, PUSH	—	—	—	—	—	—	—	—	—	—	—	—	—	WL
	LDM, STM	—	—	—	—	—	—	—	—	—	—	—	—	—	L
	MOVEPE, MOVTPE	—	—	—	—	—	—	—	B	—	—	—	—	—	—
Arithmetic operations	ADD, CMP	BWL	BWL	—	—	—	—	—	—	—	—	—	—	—	—
	SUB	WL	BWL	—	—	—	—	—	—	—	—	—	—	—	—
	ADDX, SUBX	B	B	—	—	—	—	—	—	—	—	—	—	—	—
	ADDS, SUBS	—	L	—	—	—	—	—	—	—	—	—	—	—	—
	INC, DEC	—	BWL	—	—	—	—	—	—	—	—	—	—	—	—
	DAA, DAS	—	B	—	—	—	—	—	—	—	—	—	—	—	—
	MULXU, DIVXU	—	BW	—	—	—	—	—	—	—	—	—	—	—	—
	MULXS, DIVXS	—	BW	—	—	—	—	—	—	—	—	—	—	—	—
	NEG	—	BWL	—	—	—	—	—	—	—	—	—	—	—	—
	EXTU, EXTS	—	WL	—	—	—	—	—	—	—	—	—	—	—	—
	TAS*2	—	—	B	—	—	—	—	—	—	—	—	—	—	—
	MAC*1	—	—	—	—	—	○	—	—	—	—	—	—	—	—
	CLRMAC*1	—	—	—	—	—	—	—	—	—	—	—	—	—	○
	LDMAC*1, STMAC*1	—	L	—	—	—	—	—	—	—	—	—	—	—	—

Instrukční soubor H8S (2)

Function	Instruction	Addressing Modes													
		#xx	Rn	@ERn	@(d:16,ERn)	@(d:32,ERn)	@–ERn/@ERn+	@aa:8	@aa:16	@aa:24	@aa:32	@(d:8,PC)	@(d:16,PC)	@@aa:8	–
Logic operations	AND, OR, XOR	BWL	BWL	—	—	—	—	—	—	—	—	—	—	—	—
	NOT	—	BWL	—	—	—	—	—	—	—	—	—	—	—	—
Shift		—	BWL	—	—	—	—	—	—	—	—	—	—	—	—
Bit manipulation		—	B	B	—	—	—	B	B	—	B	—	—	—	—
Branch	Bcc, BSR	—	—	—	—	—	—	—	—	—	—	○	○	—	—
	JMP, JSR	—	—	—	—	—	—	—	—	○	—	—	—	○	—
	RTS	—	—	—	—	—	—	—	—	—	—	—	—	—	○
System control	TRAPA	—	—	—	—	—	—	—	—	—	—	—	—	—	○
	RTE	—	—	—	—	—	—	—	—	—	—	—	—	—	○
	SLEEP	—	—	—	—	—	—	—	—	—	—	—	—	—	○
	LDC	B	B	W	W	W	W	—	W	—	W	—	—	—	—
	STC	—	B	W	W	W	W	—	W	—	W	—	—	—	—
	ANDC, ORC, XORC	B	—	—	—	—	—	—	—	—	—	—	—	—	—
	NOP	—	—	—	—	—	—	—	—	—	—	—	—	—	○
Block data transfer		—	—	—	—	—	—	—	—	—	—	—	—	—	BW

Příklad: Instrukce MOV.W (1)

2.2.39 (5) MOV (W)

MOV (MOVe data) **Move**

Operation

(EAs) → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

Assembly-Language Format

`MOV.W` <EAs>, Rd

H: Previous value remains unchanged.
N: Set to 1 if the transferred data is negative; otherwise cleared to 0.
Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Operand Size

Word

Description

This instruction transfers the source operand contents to a 16-bit register Rd, tests the transferred data, and sets condition-code flags according to the result.

Příklad: Instrukce MOV.W (2)

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	
Immediate	MOV.W	#xx:16, Rd	7 9	0 rd	IMM						2
Register indirect	MOV.W	@ERs, Rd	6 9	0 ers rd							2
Register indirect with displace-ment	MOV.W	@(d:16, ERs), Rd	6 F	0 ers rd	disp						3
	MOV.W	@(d:32, ERs), Rd	7 8	0 ers 0	6 B	2 rd	disp				5
Register indirect with post-increment	MOV.W	@ERs+, Rd	6 D	0 ers rd							3
Absolute address	MOV.W	@aa:16, Rd	6 B	0 rd	abs						3
	MOV.W	@aa:32, Rd	6 B	2 rd	abs						4

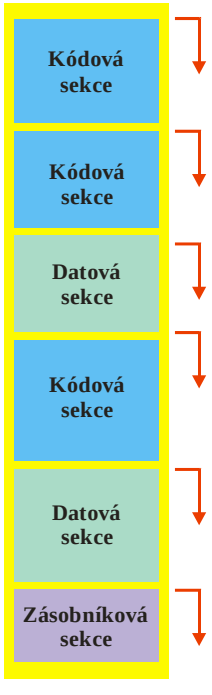
Důležité direktivy

Direktivy (povely pro překladač):

- definice sekcí (segmentů),
- definice dat a symbolů,
- makra,
- podmíněný překlad,
- další: strukturovaný překlad, listing, … .

Struktura modulu

- Modul obsahuje jednu nebo více sekcí (segmentů).
- Každá sekce má nezávislé adresování od svého začátku.
- Pořadí sekcí ve zdrojovém souboru není podstatné – upraví se při sestavení.



Základní typy sekcí (GNU as)

- Datová sekce
  - Obsahuje inicializovaná data („proměnné“) programu.
- Kódová sekce
  - Obsahuje kód programu.
- Další sekce
  - Neinicializovaná data, zásobník, přerušovací vektory, další uživatelem (programátorem) definované sekce.

Definice sekce

- Hlavička sekce (zjednodušeně)
  - Standardní sekce GNU as a ld:

```
.data  [subsekce]      začátek datové sekce
.text  [subsekce]      začátek kódové sekce
```

  - Libovolné další sekce:

```
.section  jméno        začátek sekce jméno
```

```
  ...
.text 0
  kód programu
.data 3
  data (proměnné)
.section MOJE_SEKCE
  ...
```

Počítadlo adres

- Každá sekce má (při překladu) samostatné počítadlo adres (PLC – Programm Location Counter)
- Není-li určeno jinak, inicializuje se PLC na 0 na začátku sekce.
- Možnosti nastavení PLC:

```
.org      výraz
.align    uložení
```

`.org` = nastaví PLC na hodnotu ***výraz***

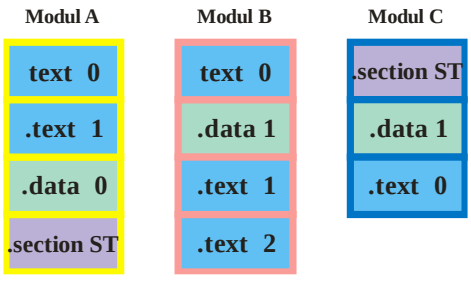
`.align` = nastaví PLC na hodnotu **MOD(2^*uložení*)**

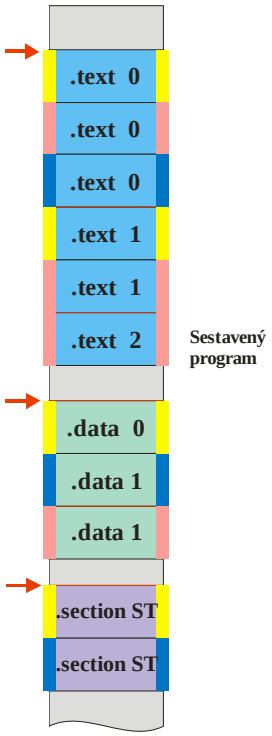
☝ Všechny adresy jsou vztaženy k začátku sekce. Je-li sekce relativní, musí se jiným způsobem zabezpečit potřebné umístění sekce v paměti.

```
.text
MOV     @VAR1,R1
JMP     @LAB1
.org    0x000400
JMP     @LAB2
.align  4
JMP     LAB3
...
```


Sestavení sekcí a modulů

- Sestavovací program spojí stejné sekce dohromady.
- Jsou-li definovány subsekce, spojí dohromady i stejné subsekce





Definice dat a symbolů

- Definice a přiřazení hodnoty symbolu

```
.equ  symbol,výraz
symbol = výraz
```

symbol – symbol, kterému bude přiřazena hodnota výrazu.

výraz – jeho hodnota bude přiřazena symbolu.

- **.equ** a **=** pouze definují symbol a jeho hodnotu. Nevyhrazují místo v paměti.
- Platnost symbolu je omezena na modul ve kterém je symbol definován.
- Hodnotu symbolu nelze změnit.

```
        ...
.equ    BASE,0x008000
.equ    LIMIT,0x100
.equ    NEXT,BASE+LIMIT
        NIC = 0
        ...
```


Definice dat (1)

- Definice místa pro „proměnnou“

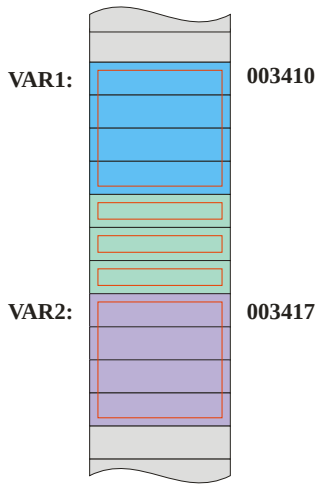
```
[návěští]  .space   položky
```

návěští – symbolická adresa,

položky - výrazy definující počet položek

- Vyhradí v paměti místo určené délky (počtu bytů).
- Je-li uvedeno návěští, odpovídá jeho hodnota adrese 1. bytu dat.

```
        ...
VAR1:   .space  4
        .space  1
        .space  1
        .space  1
VAR2:   .space  4
        ...
```



Definice dat (2)

- Definice „proměnné“ s počáteční hodnotou

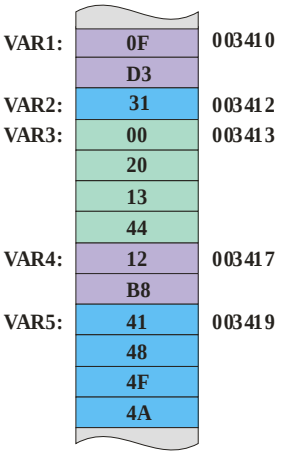
```
[návěští]  .byte   výrazy
[návěští]  .word   výrazy
[návěští]  .long   výrazy

[návěští]  .ascii   řetězec
[návěští]  .asciz   řetězec
```

návěští – symbolická adresa,
výrazy – výrazy definující obsah jednotlivých položek, oddělené čárkou.

- Vyhradí v paměti místo, jehož obsah je dán jednotlivými výrazy.
- Je-li uvedeno návěští, odpovídá jeho hodnota adresa 1. bytu dat.

```
        ...
VAR1:   .word   0x0FD3
VAR2:   .byte   '1'
VAR3:   .byte   0,32,19,68
VAR4:   .word   0x1200+0xB8
VAR5:   .ascii  „AHOJ“
        ...
```



Sdílení dat mezi moduly (1)

- Symboly mají platnost jen v modulu, ve kterém jsou definovány.
- Rozšíření platnosti (export) symbolů:

```
.global  symboly
```

symboly – seznam exportovaných symbolů oddělených čárkami.

- Použití symbolů definovaných v jiném modulu (import):

```
.extern symboly
```

symboly – seznam importovaných symbolů oddělených čárkami.

- Lze použít pouze pro symboly, definované jako návěští (ne EQU).

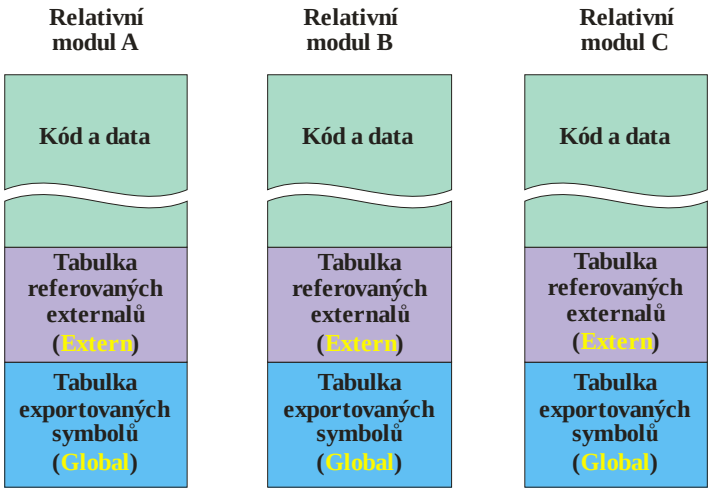
Sdílení dat mezi moduly (2)

```
;       A_MODUL
        .global VAR_A1,LAB_A1
        .extern VAR_B1,LAB_B1
VAR_A1: .word   0x0FD3
VAR_A2: .space  2
        ...
LAB_A1: MOV.W   @VAR_A2,R1
        ...
        MOV.W   R1,@VAR_B1
        JMP     @LAB_B1
        ...
```

```
;       B_MODUL
        .global VAR_B1,LAB_B1
        .extern VAR_A1,LAB_A1
VAR_B1: .word   0x0FD3
VAR_B2: .space  2
        ...
LAB_B1: MOV.W   @VAR_B2,R1
        ...
        MOV.W   R1,@VAR_A1
        JMP     @LAB_A1
        ...
```


Sdílení dat mezi moduly (3)

- Relativní moduly obsahují tabulku referovaných externích symbolů (Extern) a tabulku exportovaných symbolů (Global).
- Sestavovací program hledá externí symboly v tabulkách Global ostatních modulů.



Makra (definice)

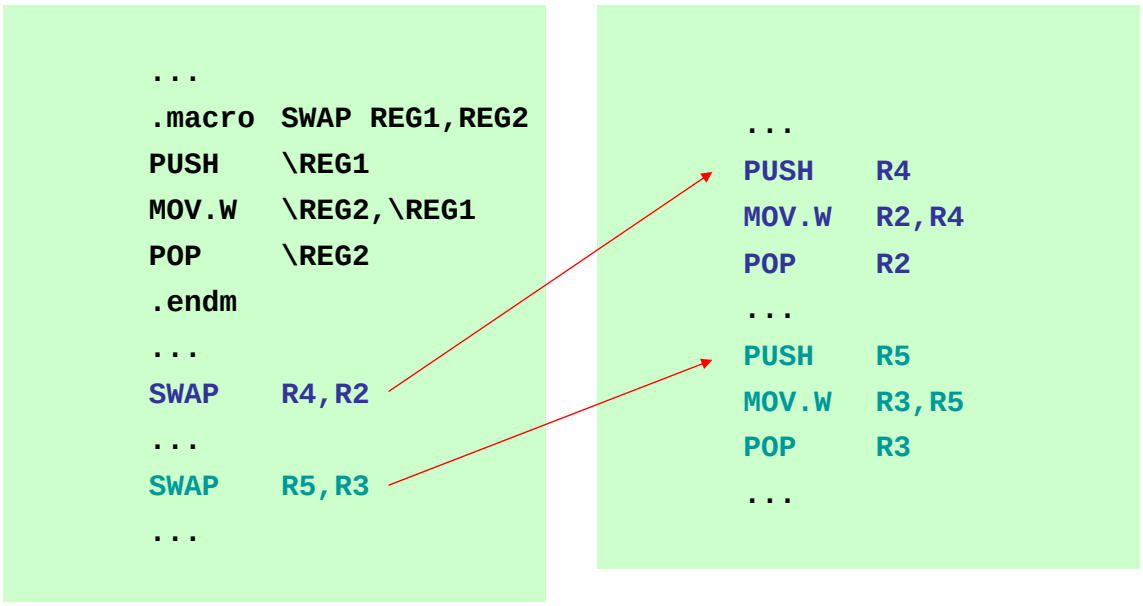
- Umožňuje definovat část programu, která bude použita na více místech.

```
.macro jméno argumenty
   tělo makra
.endm
```

jméno – jméno makra.
argumenty – seznam symbolických argumentů. V těle se referují s \ na začátku.
tělo makra – jednotlivé instrukce.
.endm – ukončuje rozvoj makra.

- Rozvinutí makra vloží tělo makra, tj. **kopie jednotlivých instrukcí** do daného místa programu.

Makra (rozvinutí)



Zápis v programu

Rozvinutí makra

Makra (lokální symboly)

- Je-li v makru definován symbol (např. návěští), vznikají při vícenásobném rozvinutí problémy.
- Symbol se musí definovat jako lokální v makru.

LOCAL *symboly*

symboly – seznam lokálních symbolů, oddělených čárkami.

- Překladač vytvoří v každém rozvinutí unikátní jméno symbolu .
- Před použitím LOCAL se musí použít direktiva **.altmacro**

Makra (lokální symboly)

```
        ...
        .altmacro
        .macro ONES VAR,RESULT
        LOCAL   LAB1,LAB2
        MOV.W   @\VAR,R1
        MOV.B   #16,R0L
        XOR.B   R0H,R0H
LAB1:   ROTR    R1
        BCC     LAB2
        INC     R0H
LAB2:   DEC     R0L
        BNE     LAB1
        MOV.B   R0H,@\RESULT
        .endm
        .noaltmacro
        ...
```

Definice makra

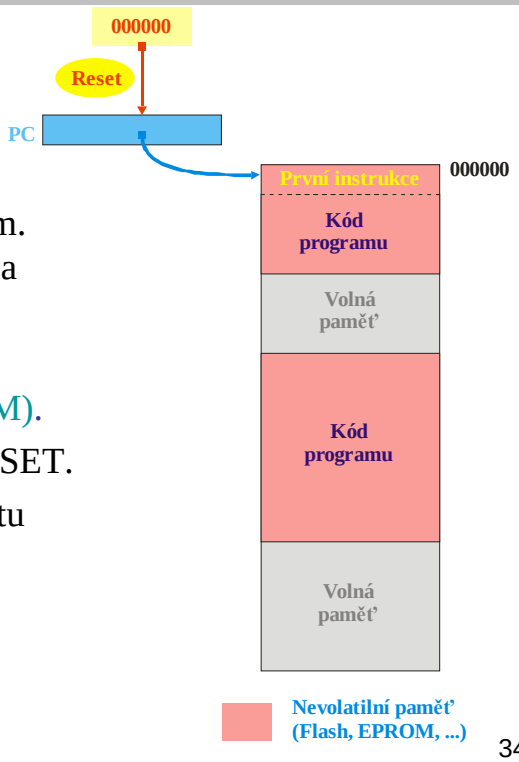
```
        ...
VAR1:   .space  2
VAR2:   .space  1
VARX:   .space  2
VARY:   .space  1
        ...
        ONES    VAR1,VAR2
        ...
        ONES    VARX,VARY
        ...
```

Použití makra

Spojování programů v JSA a vyšších jazycích

- Typický případ: procedura v JSA volaná z programu ve vyšším jazyku (např. C).
- Je nutné znát:
  - pravidla pro vytváření jmen (proměnných, segmentů, ...),
  - pravidla pro předávání argumentů do procedury,
  - pravidla pro předávání hodnoty funkce do volajícího programu,
  - pravidla pro zacházení se zásobníkem,
  - pravidla pro sestavení programu ve vyšším jazyku (inicializační modul, knihovny, ...).
- Výše uvedené bývá popsáno v příslušném manuálu.

Start procesoru (1)

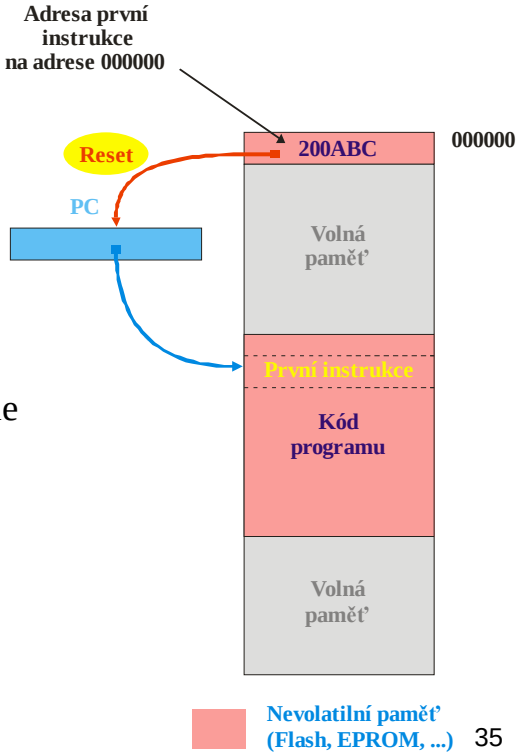


1. V paměti musí být připraven program. První instrukce programu musí být na určité pevně dané adrese (obvykle 0x0...00).
   - Paměť musí být nevolatilní (ROM).
2. Po připojení napájení se provede RESET.
3. Při resetu nastaví CPU do PC hodnotu 0x0...00 (nebo jinou pevně danou hodnotu).

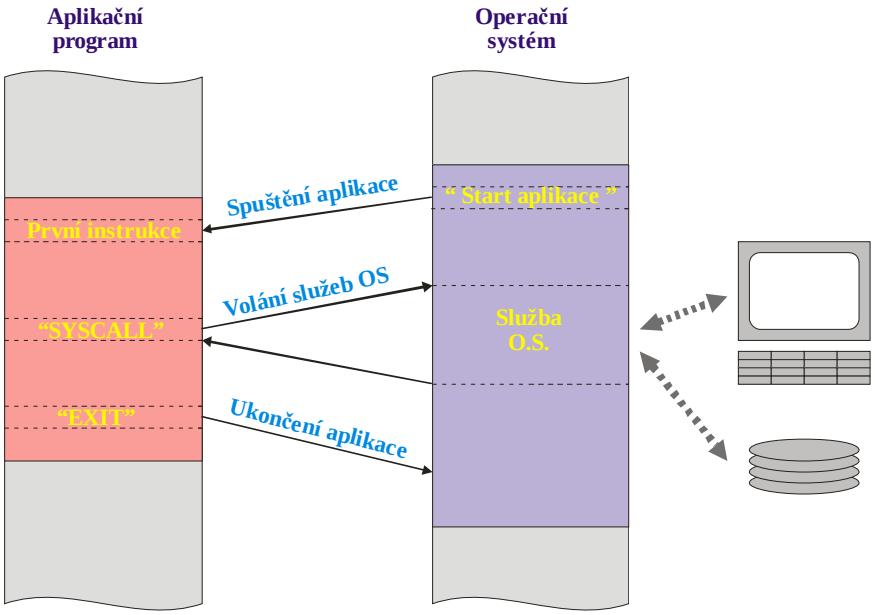
Start procesoru (2)

- **Jiná možnost (použitá i u H8S):**

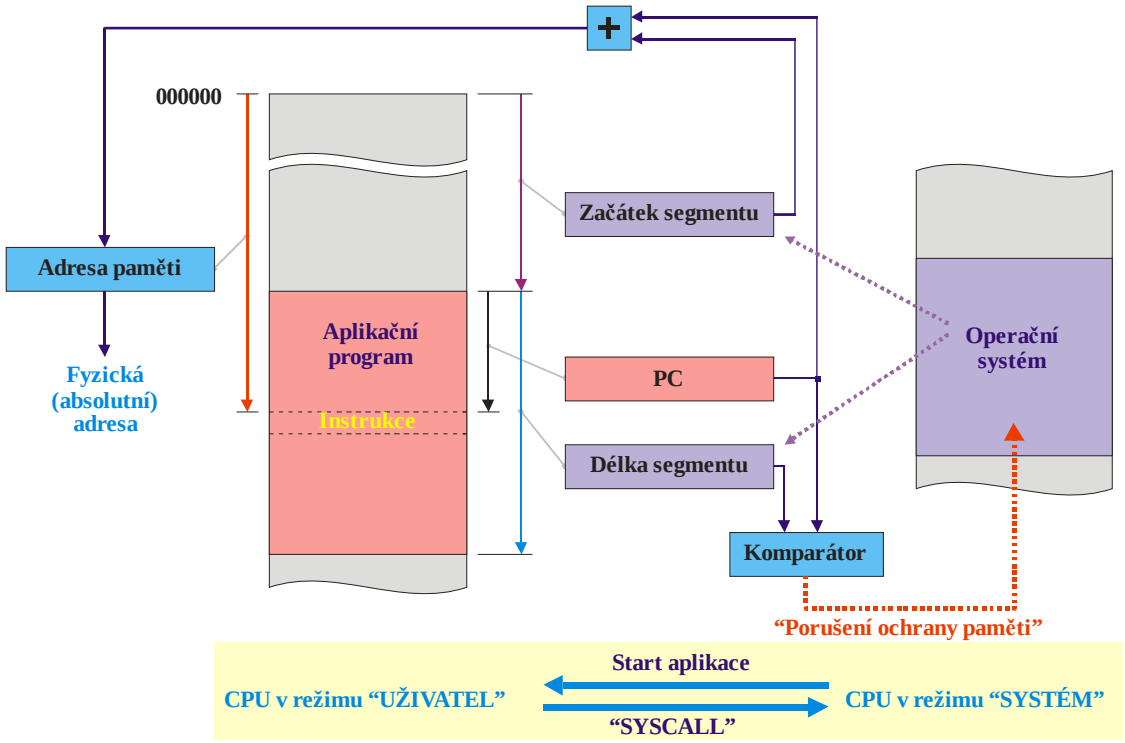
1. Po připojení napájení se provede RESET.
2. V paměti musí být připraven program. Adresa první instrukce programu musí být na určité pevně dané adrese (obvykle 0x0...00).
   – Paměť musí být nevolatilní (ROM).
3. Při resetu nastaví CPU do PC obsah adresy 0x0...00.



Operační systém



Správa paměti



Správa paměti