Ústav radioelektroniky
Vysoké učení technické v Brně

Instrukční soubor mikrokontrolérů

Mikroprocesorová technika a embedded systémy

Přednáška 2

doc. Ing. Tomáš Frýza, Ph.D.

Obsah přednášky

Vývoj aplikací pro mikrokontroléry

Formáty instrukcí
- Jednobytové, dvoubytové, tříbytové, konstantní šířka instrukcí RISC

Direktivy překladače

Programátorský model

Typy instrukcí
- Aritmetické, logické, přesun dat, bitové operace, větvení programu
- Ukázky zdrojových kódů pro AVR

Proces dekódování a vykonání instrukcí

Ukázka programu v JSA a C pro AVR
- Podmíněné větvení programu

Obsah přednášky

Vývoj aplikací pro mikrokontroléry

Formáty instrukcí
- Jednobytové, dvoubytové, tříbytové, konstantní šířka instrukcí RISC

Direktivy překladače

Programátorský model

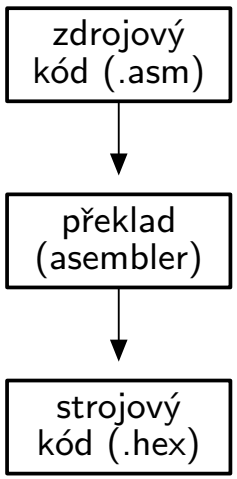
Typy instrukcí
- Aritmetické, logické, přesun dat, bitové operace, větvení programu
- Ukázky zdrojových kódů pro AVR

Proces dekódování a vykonání instrukcí

Ukázka programu v JSA a C pro AVR
- Podmíněné větvení programu

Program mikrokontroléru

- Od von Neumannovy architektury je mikroprocesor řízen programem, který je tvořen posloupností instrukcí v binární podobě. (Instrukce je vlastně číselně vyjádřený pokyn co dělat.)
- Instrukce jsou uloženy–v podobě tzv. strojového kódu–v programové paměti (u AVR typ Flash); jsou načítány a vykonávány sekvenčně, tj. jak jdou po sobě.
- V mikroprocesorové technice obsahují instrukce dvojí informaci:
  (1) jaká operace má být vykonána (tzv. operační kód – op code),
  (2) adresa operandů, se kterými se má pracovat.
- Každý typ mikrokontroléru obsahuje JINOU instrukční sadu; běžné velikosti instrukcí: 1–4bytové.



Obrázek: Překlad zdrojového kódu z jazyka symbolických adres do strojového kódu.

- Řídicí jednotka pracuje s binárně vyjádřenými pokyny – strojový kód.
- Psát aplikaci přímo ve strojovém jazyce je nepřehledné, neefektivní, složité – využívá se symbolický zápis (JSA).
- Převod z JSA do strojového kódu provádí tzv. assembler – program, který každou instrukci nahradí binární posloupností, podle definovaných pravidel.

Programování v jazyce symbolických adres (JSA)

- Program (tj. strojový kód instrukcí) je v paměti uložen v binární podobě; pro přehlednost se ale zapisuje v hexadecimální soustavě. (Snadný převod mezi hex. a bin. soustavou.)
- Zdrojový kód může obecně obsahovat 4 formy zápisu:

  (1) `[návěští:]  direktiva pro assembler/překladač  [operandy]  [; komentář]`

  (2) `[návěští:]  instrukce  [operandy]  [; komentář]`

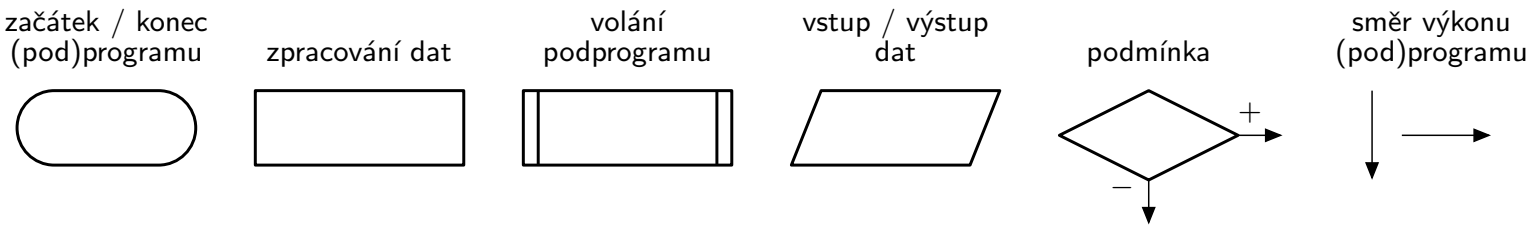
  (3) `; komentář`

  (4) prázdný řádek

Pozn.: označení v závorkách [·] je nepovinné.
- Návěští je symbolické označení místa v paměti – vždy končí dvojtečkou.
- Komentář vždy za středník.

- Zažitá pravidla při psaní kódu:
  - musí být srozumitelné nejen pro autora,
  - tabelátor před instrukcí; návěští na kraji řádku,
  - používání komentářů u instrukcí či jednotlivých funkčních bloků.

Vývojové diagramy



Obrázek: Symboly používané ve vývojových diagramech.

- Ustálené grafické značky pro popis specifické funkce vyvíjeného algoritmu.
- Tvorba vývojového diagramu by měla předcházet jakémukoliv programu.

Příklad

Vytvořte vývojový diagram pro binární čítač s výstupem pomocí LED diod.

Obsah přednášky

Vývoj aplikací pro mikrokontroléry

Formáty instrukcí

Jednobytové, dvoubytové, tříbytové, konstantní šířka instrukcí RISC

Direktivy překladače

Programátorský model

Typy instrukcí

Aritmetické, logické, přesun dat, bitové operace, větvení programu
Ukázky zdrojových kódů pro AVR

Proces dekódování a vykonání instrukcí

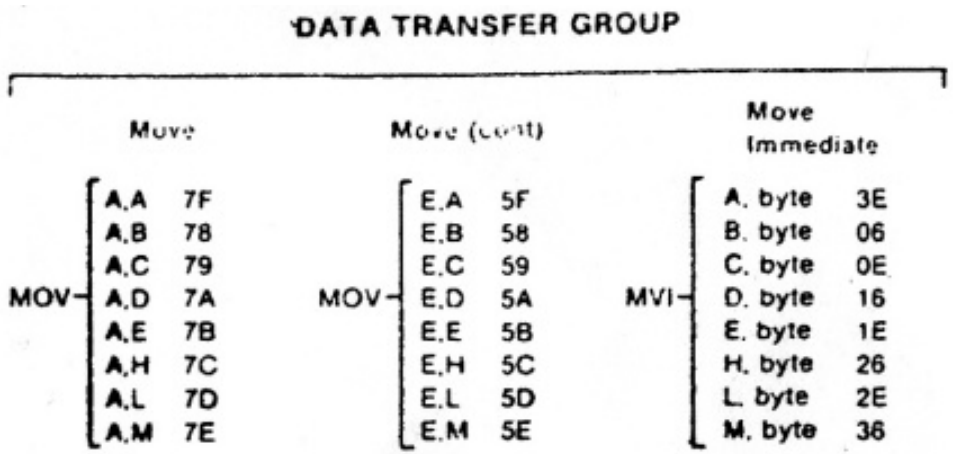
Ukázka programu v JSA a C pro AVR

Podmíněné větvení programu

Jednobytové instrukce

- Jednobytové instrukce informují pouze o op kódu.
- Nepoužívají se žádné operandy, není tedy možné přistupovat k pozicím v paměti. Je ale možné rozlišovat "adresu" registrů.
- Příklad 1bytových instrukcí procesoru typu '51:
  - `CLR A` – nuluje všechny bity v akumulátoru
    strojový kód: `1110 0100`,
  - `ADD A, Rr` – k akumulátoru přičte obsah registru Rr, kde $r = 0, 1, \ldots, 7$
    strojový kód: `0010 1rrr`.
  - Ukázka: Překlad instrukce `ADD A, R2` a její uložení v paměti od adresy 0x0005
    `0005  2A  ; operační kód instrukce ADD A, R2`

Obrázek: Část instrukční sady Intel 8085 – 1bytové instrukce MOV (kopírování dat mezi pracovními registry), MVI (naplnění pracovního registru 1bytovou hodnotou; ekvivalent LDI u AVR).

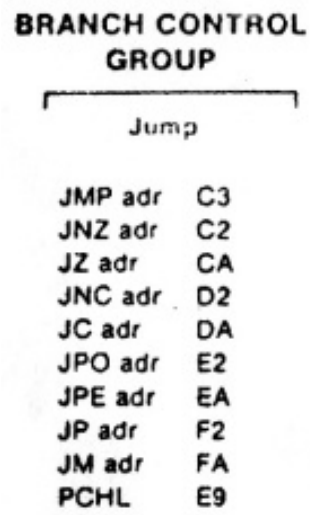


Tříbytové instrukce

- Obsahují 8bitový op kód a 16bitovou adresu operandu. Jedná se o totožný případ, jako u 2B instrukcí, jen s tím rozdílem že adresa operandů je 16bitová.
- Pořadí uložení tří slov v paměti:
  (1) op kód,
  (2) nižší byte adresy,
  (3) vyšší byte adresy.
- Příklad 3bytové instrukce procesoru Intel 8085:
  - `JMP adresa` – nepodmíněný skok na adresu
    strojový kód: `1100 0011  16bitová adresa`.
  - Ukázka: Překlad instrukce `JMP 201F` a její uloženi v paměti od adresy 0x2008

```
2008   C3   ; operační kód instrukce JMP
2009   1F   ; nižší byte adresy
200A   20   ; vyšší byte adresy
```

Obrázek: Část instrukční sady Intel 8085 – 3bytové instrukce pro větvení programu.



BRANCH CONTROL GROUP

Jump

JMP adr	C3
JNZ adr	C2
JZ adr	CA
JNC adr	D2
JC adr	DA
JPO adr	E2
JPE adr	EA
JP adr	F2
JM adr	FA
PCHL	E9

Struktura instrukcí RISC procesorů

- Mikroprocesory s architekturou typu RISC (redukovaná instrukční sada) mají konstantní délku všech instrukcí. Umožňuje to rychlejší dekódování instrukcí a tím i zvyšování početního výkonu programu.
- Příklady instrukcí procesoru typu RISC (AVR):
  - `ret` – návrat z podprogramu
    strojový kód: `1001 0101 0000 1000`,
  - `inc Rd` – inkrementace obsahu registru Rd, kde $d = 0, 1, \ldots, 31$
    strojový kód: `1001 010d dddd 0011`,
  - `ldi Rd, K` – naplnění registru Rd 8bitovou konstantou K, kde $d = 16, 17, \ldots, 31$ a $K = 0, 1, \ldots, 255$
    strojový kód: `1110 KKKK dddd KKKK`,
  - `add Rd, Rr` – součet obsahu registrů Rd a Rr; výsledek do Rd
    strojový kód: `0000 11rd dddd rrrr`.

Výrazy podporované JSA pro mikrokontroléry AVR

- Překladač (AVR assembler) umožňuje zápis hodnot ve třech soustavách:
  - desítková soustava: 10, 255, . . .,
  - hexadecimální: **0x**09, **$**F5 (dvojí možnost zápisu),
  - binární soustava: **0b**00101110.

Příklad

Zápis 0b1100 reprezentuje 8mi bitovou hodnotu 0b0000 1100 (12) nebo 0b1100 0000 (192)?

Příklad

Hodnota 0xA reprezentuje zápis 0xA0 (160) nebo 0x0A (10)?

- AVR assembler také umožňuje použití funkcí (vrací hodnotu). Některé z podporovaných funkcí:
  - `LOW(výraz)` – vrací nižší byte 16bitové hodnoty `výraz`,
  - `HIGH(výraz)` – vrací vyšší byte 16bitové hodnoty `výraz`, (využití např. při definování zásobníku - viz později)
  - `LWRD(výraz)` – vrací nižší slovo 32bitové hodnoty `výraz`,
  - `HWRD(výraz)` – vrací vyšší slovo 32bitové hodnoty `výraz`.

Pozn.: Slovo (word) reprezentuje 16 bitů.

Obsah přednášky

Direktivy (příkazy) pro překladač AVR

(1) `[návěští:] direktiva pro assembler/překladač [operandy] [; komentář]`

- Direktivy NEjsou překládány do strojového kódu, pouze informují překladač jak sestavit výsledný strojový kód.
- Nejčastější použití direktiv:
  - vkládaní souborů (`.include`),
  - definice "proměnných" (`.def`, `.equ`, `.set`),
  - určení typu paměti pro uložení kódu (`.cseg`, `.dseg`, ...),
  - definování maker (`.macro`, `.endmacro`), ...

Tabulka: Direktivy překladače AVR.

Direktiva	Význam	Direktiva	Význam
`.byte`	Alokace bytů v SRAM.	`.eseg`	EEPROM segment.
`.cseg`	Flash segment.	`.exit`	Opuštění zdrojového souboru.
`.db`	Alokace 8bitové hodnoty.	`.include`	Vložení zdrojového souboru.
`.def`	Přejmenování registru.	`.list`	Zapnutí generování listfile.
`.device`	Definování typu MCU.	`.listmac`	Zapnutí generování info o makru.
`.dseg`	Segment SRAM.	`.macro`	Začátek definice makra.
`.dw`	Alokace 16bitové hodnoty.	`.nolist`	Vypnutí generování do listfile.
`.endmacro`	Konec definice makra.	`.org`	Označení konkrétní adresy.
`.equ`	Definice výrazu/konstanty.	`.set`	Definice výrazu.

Část výpisu popisného souboru `m16def_mod.inc`

```
1  ;***** Specify Device
2  .device ATmega16
3
4  ;***** I/O Register Definitions
5  .equ    SREG    =$3f
6  .equ    SPH     =$3e
7  .equ    SPL     =$3d
8  .equ    OCR0    =$3c
9  .equ    GICR    =$3b        ; New name for GIMSK
10 ...
11
12 ; ***** PORTB ************************
13 ; PORTB - Port B Data Register
14 .equ    PORTB0  = 0 ; Port B Data Register bit 0
15 .equ    PB0 = 0 ; For compatibility
16 .equ    PORTB1  = 1 ; Port B Data Register bit 1
17 .equ    PB1 = 1 ; For compatibility
18 ...
19
20 .equ    RAMEND =$45F
21 ...
22
23 ; ***** INTERRUPT VECTORS ********************************************
24 .equ    INT0addr    = 0x0002    ; External Interrupt Request 0
25 .equ    INT1addr    = 0x0004    ; External Interrupt Request 1
26 .equ    OC2addr = 0x0006    ; Timer/Counter2 Compare Match
27 ...
```

Použití maker v programu

- Makra se používají pro často se opakující úseky programu.
- Pokud překladač z JSA narazí v programu na název makra, vloží místo něj kód definovaný mezi direktivou `.macro` a `.endmacro`.
- Rozdíl mezi makrem a podprogramem je v tom, že makro se vloží přímo do kódu a jeho kód se překládá a vkládá při každém užití makra; podprogram se přeloží jen jednou a skáče se na něj.
- Makro může mít až 10 vstupních parametrů.
- V samotném těle makra jsou tyto hodnoty symbolicky označovány @0 (první operand) až @9 (desátý operand).

Ukázka definice a použití makra

```
1  ; inicializace aplikace, včetně nového makra
2  .include <m16def_mod.inc>    ; vložit upravený popisný soubor pro ATmega16
3  .macro ADD16                 ; začátek nově definovaného makra s názvem ADD16
4      add    @0, @2            ; součet dvou nižší bytů
5      adc    @1, @3            ; vyšší byty s přenosem
6  .endmacro                    ; konec makra
7
8      ...
9  ; použití makra v programu
10     ADD16 R16,R17, R18,R19   ; sečti dvě 16bitová čísla
```

- Příklad použití makra:

  `ADD16 R16,R17, R18,R19`

  - Toto makro obsahuje 4 vstupní operandy: R16 (překladač s ním pracuje prostřednictví označení "@0") až R19 (@3).
  - Při předzpracování zdrojového kódu bude nově-definované makro, reprezentující výpočet součtu dvou 16bitových čísel, `ADD16` nahrazeno následujícím kódem:

```
1      add    R16, R18
2      adc    R17, R19
```

Otázka

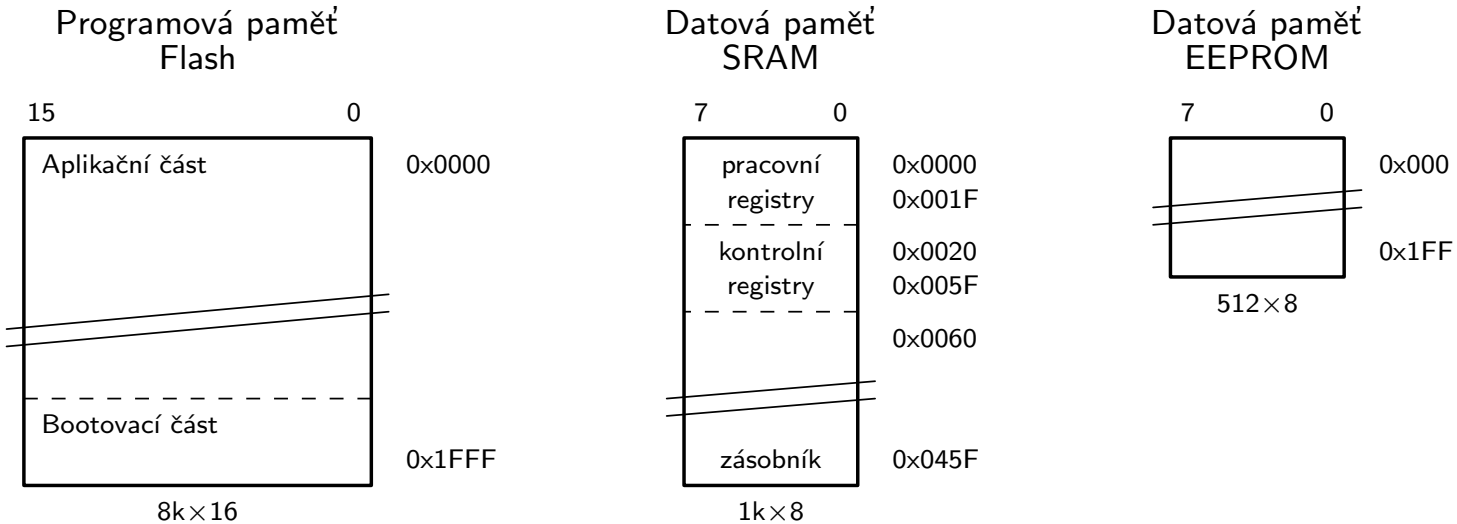
Jaký je rozdíl mezi instrukcí `ADD` a `ADC`?

Obsah přednášky

Programátorský model

Definice (Programátorský model)

Programátorský model mikrokontroléru (nebo také softwarový pohled na MCU) obsahuje popis paměťového prostoru MCU, soubor registrů, jejich názvy, adresy a funkce, které může programátor využívat.



Obrázek: Paměťový prostor mikrokontroléru ATmega16.

- Obecné pracovní registry (anglicky: General Purpose Registers) – probráno.

Vstupně/výstupní (kontrolní) registry AVR

- I/O registry slouží k řízení jednotlivých periférií i samotného jádra mikrokontroléru. Představují nejdůležitější informace pro programátora!
- Každý bit má specifický význam – vždy nutné konzultovat s katalogovým listem konkrétního mikrokontroléru.
- Pro ATmega16 jsou I/O registry namapovány na adresách 0x20 až 0x5F v SRAM.
- Přistupuje se k nim pomocí instrukcí `IN` a `OUT`!

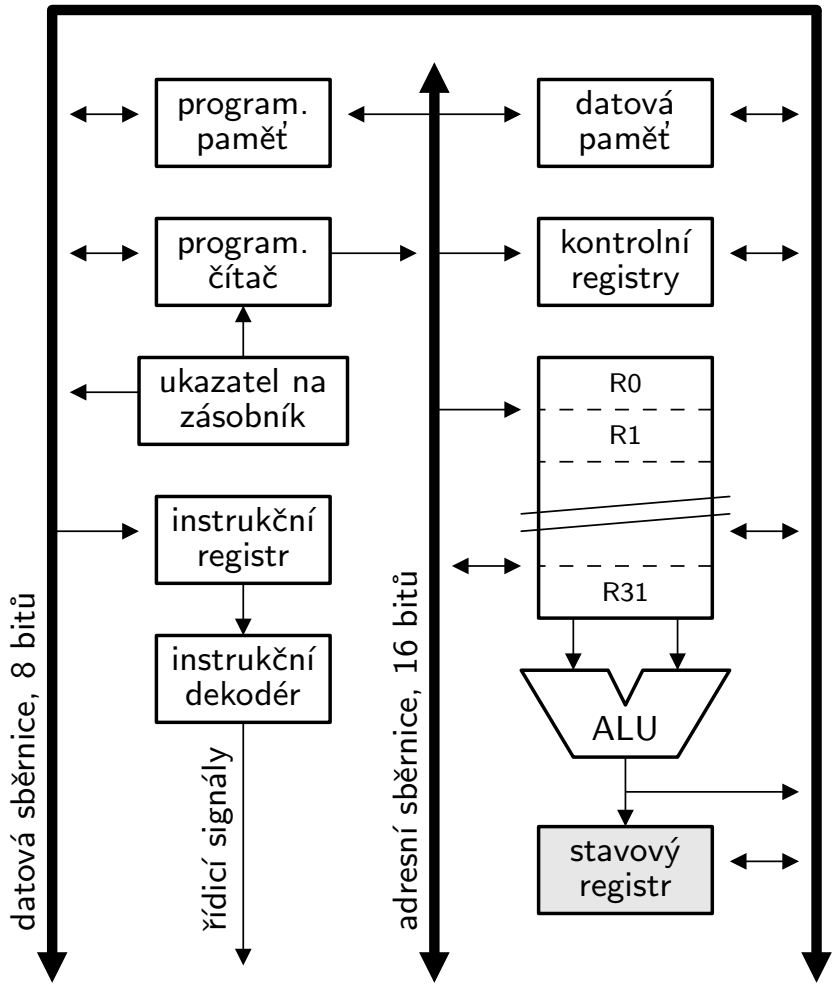
Tabulka: Vybrané kontrolní registry AVR.

Adresa	Registr	Funkce registru
...		
0x39	PINA	Vstupní registr portu A.
0x3A	DDRA	Směrový registr portu A.
0x3B	PORTA	Datový registr portu A.
...		
0x52	TCNT0	Obsah 8bitového čítače/časovače 0.
...		
0x5B	GICR	Řídicí registr přerušení.
0x5C	OCR0	Komparační registr čítače/časovače 0.
0x5D	SPL	Ukazatel na zásobník (nižší byte).
0x5E	SPH	Ukazatel na zásobník (vyšší byte).
0x5F	SREG	Stavový registr.

Stavový registr, příznakové bity

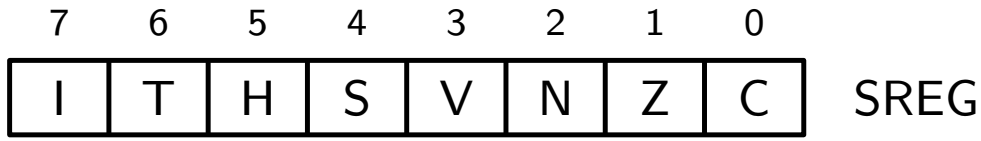
- Příznakové bity (anglicky: Flags) informují o typických výsledcích poslední aritmetické, logické, příp. bitové operace.
- Jsou nastavovány/nulovány aritmeticko logickou jednotkou (ALU), příp. datovou sběrnicí.
- Lze využít např. k řízení běhu programu (nejčastěji k větvení).
- Jsou uloženy ve stavovém registru (SREG – Status Register) – jeden z I/O registrů.
- Typy příznakových bitů se mohou lišit podle typu mikrokontroléru (AVR, '51, Freescale, ...).
- Liší se i označení stavového registru (např.: SREG – AVR, PSW – Program Status Word, Intel 8051, CCR – Code Condition Register, Freescale).

Stavový registr AVR (SREG)



Obrázek: Stavový registr (SREG) u mikrokontrolérů AVR.

Stavový registr AVR (SREG)



- C (carry flag) – příznak přenosu z nejvýznamnějšího bitu po poslední aritmetické operaci nebo po bitovém posuvu.

```
1       LDI   R16, 225          ; R16 = 225
2       LDI   R17, 50           ; R17 = 50
3       ADD   R16, R17          ; R16 = 225 + 50 = 19, C = 1
```

- Z (zero flag) – příznak nulového výsledku aritmetické nebo logické operace, příp. bitového posuvu:
  - Z=1, pokud je výsledek ROVEN nule,
  - Z=0, pokud je výsledek různý od nuly.

```
1       LDI   R16, 225          ; R16 = 225
2       SUBI  R16, 225          ; R16 = R16 - 225 = 0, Z = 1
```

Stavový registr AVR (SREG)

- N (negative flag) – MSB výsledku = 1.

```
1     LDI   R16, 0b01100000   ; R16 = 192 (dek)
2     LSL   R16               ; bitový posun doleva (logical shift left)
3                             ; R16 = 0b1100 0000, N = 1
```

- V (overflow flag) – příznak přetečení (chybný výsledek) při počítání se znaménkovými čísly (dvojkový doplněk - viz číselné soustavy).

```
1     LDI   R16, 0b11000000   ; R16 = -64 (dek)
2     LDI   R17, 0b10000000   ; R17 = -128 (dek)
3     ADD   R16, R17          ; R16 = 0b0100 0000 = 64
4                             ; V = 1
```

- S (sign flag) – informuje, zda je výsledek znaménkové číslo. Dopočítává se: $N \oplus V$.

Stavový registr AVR (SREG)

- H (half carry flag) – příznak tzv. pomocného nebo poloviční přenosu z nižšího niblu do vyššího (využívá se pro BCD aritmetiku).

```
1     LDI   R16, 0b00001000   ; R16 = 8 (dek)
2     LSL   R16               ; bitový posun doleva
3                             ; R16 = 0b0001 0000, H = 1
```

- T (transfer bit) – využíván instrukcemi `BST Rd, b` a `BLD Rd, b` pro přesun jednoho bitu z/do registru Rd ($d = 0, 1, \ldots, 31$; $b = 0, 1, \ldots, 7$).

```
1     LDI   R16, 0b01100100   ; R16 = 100 (dek)
2                             ; Bit STORE from Bit ...
3     BST   R16, 2            ; přesun bitu č. 2 do T v registru SREG
4                             ; Bit LOAD from T Flag ...
5     BLD   R16, 7            ; přesun T bitu na pozici 7 v registru R16
6                             ; R16 = 0b1110 0100
```

Stavový registr AVR (SREG)

- I (global interrupt enable) – povolení všech přerušení.

```
1     SEI    ; povolení všech přerušení, I = 1
2     CLI    ; zakázaní všech přerušení, I = 0
```

- Různé procesory mohou mít odlišné příznakové bity!
  - '51: P – parita, F0 – uživatelský, RS1:0 – volba registrové banky.
- Obsah stavového registru není automaticky zálohován při obsluze přerušení – je potřeba zajistit softwarově.
- Důležitá vlastnost instrukce je zda mění/nemění příznakové bity (příznaky nastavuje ALU) a doba výkonu. Tyto informace lze vyčíst z manuálu.

Tabulka: Vybrané instrukce AVR.

Instrukce	Popis	Funkce	Příznaky	Cykly
`ADD Rd, Rr`	Součet bez přenosu	Rd=Rd+Rr	Z,C,N,V,H	1
`INC Rd`	Inkrementace	Rd=Rd+1	Z,N,V	1
`SEC`	Nastavení příznaku C	C=1	C	1
`MOV Rd, Rr`	Kopírování obsahu reg.	Rd=Rr	žádný	1

Obsah přednášky

Typy instrukcí

`[návěští:]  instrukce  [operandy]  [; komentář]`

- Obecně lze instrukce podle jejich typu rozdělit na:
  - aritmetické operace (součet, rozdíl, inkrementace, ...),
  - logické operace (AND, XOR, ...),
  - skokové operace (nepodmíněné, podmíněné větvení programu),
  - přesun dat (přesun mezi registry, čtení z paměti, ...),
  - bitové operace (nastavení/nulování jednotlivých bitů, prohození niblů, ...),
  - řídicí instrukce (nop, sleep, watchdog reset, ...).
- Pro AVR (ATmega16) existuje 131 instrukcí, ale některé se liší jen nepatrně. Nižší řady mikrokontrolérů AVR nemusí obsahovat všechny instrukce.
- Pseudoinstrukce – ve skutečnosti není součásti instrukční sady procesoru. Je to vhodné označení (pojmenování) jedné nebo více existujících instrukcí.

Tabulka: Vybrané pseudoinstrukce AVR.

Instrukce	Popis	Funkce	Operační kód
`ADD Rd, Rr`	Součet hodnot bez přenosu.	Rd=Rd+Rr	`0000 11rd dddd rrrr`
`LSL Rd`	Bitový posun doleva.	Rd=Rd+Rd	`0000 11dd dddd dddd`
`LDI Rd, K`	Naplnění registru hodnotou.	Rd=K	`1110 KKKK dddd KKKK`
`SER`	Nastavení celého registru.	Rd=$FF	`1110 1111 dddd 1111`
`AND Rd, Rr`	Logický součin.	Rd=Rd·Rr	`0010 00rd dddd rrrr`
`TST Rd`	Testování nul./záporné hodn.	Rd=Rd·Rd	`0010 00dd dddd dddd`

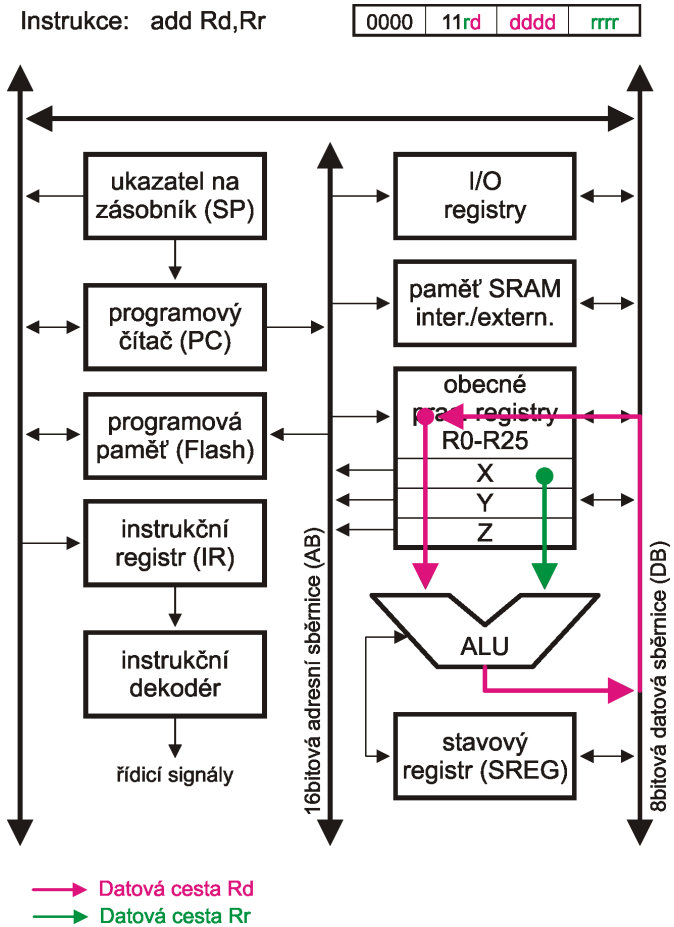
Aritmeticko/logické operace ATmega16

Tabulka: Vybrané aritmetické a logické instrukce AVR.

Instrukce	Popis	Funkce	Příznaky	Cykly
`ADD Rd, Rr`	Součet bez přenosu.	Rd=Rd+Rr	Z,C,N,V,H	1
`ADC Rd, Rr`	Součet s přenosem.	Rd=Rd+Rr+C	Z,C,N,V,H	1
`SUB Rd, Rr`	Rozdíl dvou registrů.	Rd=Rd-Rr	Z,C,N,V,H	1
`AND Rd, Rr`	Logický součin.	Rd=Rd and Rr	Z,N,V	1
`OR Rd, Rr`	Logický součet.	Rd=Rd or Rr	Z,N,V	1
`EOR Rd, Rr`	Exkluzivní součet.	Rd=Rd xor Rr	Z,N,V	1
`COM Rd`	Jednotkový doplněk.	Rd=$FF-Rd	Z,C,N,V	1
`NEG Rd`	Dvojkový doplněk.	Rd=$00-Rd	Z,C,N,V,H	1
`SBR Rd, K`	Nastavení bitu v reg.	Rd=Rd or K	Z,N,V	1
`CBR Rd, K`	Nulování bitu v reg.	Rd=Rd and ($FF-K)	Z,N,V	1
`INC Rd`	Inkrementace registru.	Rd=Rd+1	Z,N,V	1
`DEC Rd`	Dekrementace registru.	Rd=Rd-1	Z,N,V	1
`TST Rd`	Test nuly nebo záporné hod.	Rd=Rd and Rd	Z,N,V	1
`CLR Rd`	Vynulování registru.	Rd=Rd xor Rd	Z,N,V	1
`SER Rd`	Nastavení registru.	Rd=$FF	žádný	1
`MUL Rd, Rr`	Násobení neznaménkových hod.	R1:0=RdxRr	Z,C	2
`MULS Rd, Rr`	Násobení znaménkových hodnot.	R1:0=RdxRr	Z,C	2
`...`				

Aritmetické operace AVR

- První parametr určuje operand a současně destinaci výsledku.
- Druhý parametr určuje druhý operand (konstantu, ...).
- Všechny aritmeticko/logické instrukce nastavují některé příznakové bity (kromě `SER Rd`) a většina trvá jeden hodinový cyklus (konkrétní případ AVR).
- Ukázka: Op kód a výkon instrukce součtu `ADD Rd, Rr`.



Obrázek: Výkon instrukce `ADD`.

Skokové operace AVR

Zkratka	Operandy	Popis funkce	Operace	Příznaky	Cykly
RJMP	k	Relative Jump	PC=PC+k+1	None	2
IJMP		Indirect Jump to (Z)	PC=Z	None	2
JMP	k	Direct Jump	PC=k	None	3
RCALL	k	Relative Subroutine Call	PC=PC+k+1	None	3
ICALL		Indirect Call to (Z)	PC=Z	None	3
CALL	k	Direct Subroutine Call	PC=k	None	4
RET		Subroutine Return	PC=STACK	None	4
RETI		Interrupt Return	PC=STACK	I	4
CPSE	Rd, Rr	Compare, Skip if Equal		None	1/2/3
CP	Rd, Rr	Compare	Rd-Rr	Z,N,V,C,H	1
CPC	Rd, Rr	Compare with Carry	Rd-Rr-C	Z,N,V,C,H	1
CPI	Rd, K	Compare Register with Immediate		Z,N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared		None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set		None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared		None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set		None	1/2/3
...		...			

Přesun dat AVR

Zkratka	Operandy	Popis funkce	Operace	Příznaky	Cykly
MOV	Rd, Rr	Move Between Registers	Rd=Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd=Rr+1:Rr		1
LDI	Rd, K	Load Immediate	Rd=K	None	1
LD	Rd, X	Load Indirect	Rd=(X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd=(X), X=X+1		2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd=(Z+q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd=(k)	None	2
ST	X, Rr	Store Indirect	(X)=Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X)=Rr, X=X+1		2
STS	k, Rr	Store Direct to SRAM	(k)=Rr	None	2
LPM		Load Program Memory	R0=(Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd=(Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd=(Z), Z=Z+1		3
SPM		Store Program Memory	(Z)=R1:R0	None	-
IN	Rd, P	In Port	Rd=P	None	1
OUT	P, Rr	Out Port	P=Rr	None	1
PUSH	Rr	Push Register on Stack	STACK=Rr	None	2
POP	Rd	Pop Register from Stack	Rd=STACK	None	2
...		...			

Bitové operace AVR

Zkratka	Operandy	Popis funkce	Operace	Příznaky	Cykly
SBI	P,b	Set Bit in I/O Register	I/O(P,b)=1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b)=0	None	2
LSL	Rd	Logical Shift Left		Z,C,N,V	1
LSR	Rd	Logical Shift Right		Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry		Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry		Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right		Z,C,N,V	1
SWAP	Rd	Swap Nibbles		None	1
BSET	s	Flag Set	SREG(s)=1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s)=0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T=Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b)=T	None	1
SEC		Set Carry	C=1	C	1
CLC		Clear Carry	C=0	C	1
SEN		Set Negative Flag	N=1	N	1
CLN		Clear Negative Flag	N=0	N	1
SEZ		Set Zero Flag	Z=1	Z	1
CLZ		Clear Zero Flag	Z=0	Z	1
SEI		Global Interrupt Enable	I=1	I	1
CLI		Global Interrupt Disable	I=0	I	1
...		...			

Řídicí operace AVR

Zkratka	Operandy	Popis funkce	Operace	Příznaky	Cykly
NOP		No Operation		None	1
SLEEP		Sleep		None	1
WDR		Watchdog Reset		None	1
BREAK		Break For On-Chip Debug Only		None	N/A
...		...			

Ukázka zdrojového kódu logické operace

Příklad

Negace jediného bitu v registru pomocí funkce XOR.

Řešení

- *Help: op kód instrukce* `XOR`
  - `EOR Rd, Rr ; Rd = Rd XOR Rr`
    Strojový kód: `0010 01rd dddd rrrr`

Tabulka: Pravdivostní tabulka XOR.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Ukázka zdrojového kódu logické operace

```
1  .include <m16def.inc>          ; popisný soubor ATmega16
2  .def temp  = R16               ; symbolický název registru R16
3  .def maska = R17               ; symbolický název registru R17
4
5  .cseg                          ; paměťový segment Flash
6  .org 0x0000                    ; ulož od adresy 0x0000
7
8  start:
9      LDI  temp,  0b11010111     ; testovací data
10     LDI  maska, 0b00100000
11
12     EOR  temp, maska           ; negace bitu 5
13     EOR  temp, maska           ; negace bitu 5
14
15 loop:                          ; konec programu
16     RJMP loop                  ; skok na návěští loop
```

```
Adresa     Instr. Zkratka Operandy
-----------------------------------
+00000000: start
+00000000:  ED07  LDI     R16,0xD7
+00000001:  E210  LDI     R17,0x20
+00000002:  2701  EOR     R16,R17
+00000003:  2701  EOR     R16,R17
@00000003: loop
+00000003:  CFFF  RJMP    PC-0x0000
```

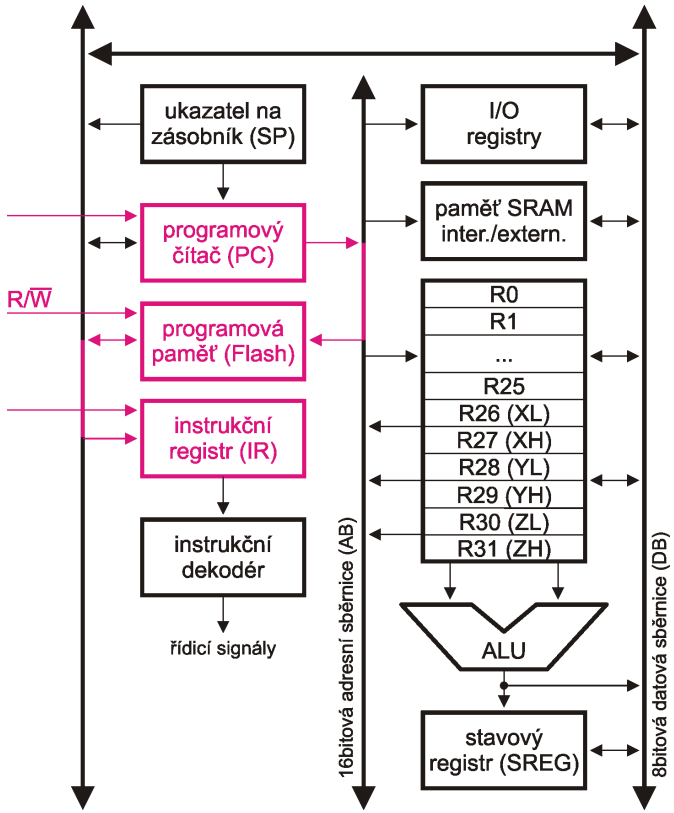
Obsah přednášky

Proces dekódování a vykonání instrukcí

- Proces dekódování a vykonání instrukcí = běh programu.

(1) Načtení (fetch) op kódu instrukce:

- řídicí jednotka generuje signál, který umístí obsah 16bitového, programového čítače (PC – Program Counter) na adresní sběrnici. (Programový čítač obsahuje adresu vykonávané instrukce),
- řídicí jednotka generuje signál $R/\overline{W}$ =1 pro čtení z paměti; čtená instrukce se tak zapíše na datovou sběrnici,
- řídicí jednotka generuje signál pro načtení obsahu datové sběrnice do instrukčního registru (IR – Instruction Register).



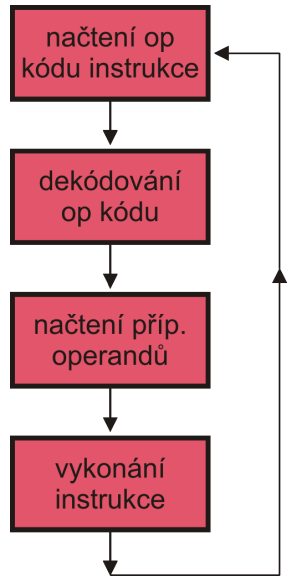
Obrázek: Načtení instrukce.

Proces dekódování a vykonání instrukcí, pokračování

(2) Inkrementace obsahu PC:
- obsah programového čítače je zvětšen o jedničku, takže obsahuje adresu následující instrukce, příp. adresy operandu v programové paměti.

(3) Dekódování op kódu instrukce:
- obsah instrukčního registru je přesunut do instrukčního dekodéru, kde je identifikována instrukce,
- pokud instrukce vyžaduje operandy, jsou načteny (krok 1) do datového registru.

(4) Vykonání instrukce:
- řídicí jednotka generuje signály nutné pro vykonání příslušné instrukce.



Obrázek: Výkon programu.

Obsah přednášky

Podmíněné větvení programu v JSA a v C

- Podmíněné větvení programu slouží k řízení běhu programu. "Pokud je splněna podmínka něco vykonej."
- Mikrokontroléry AVR obsahují 25 různých instrukcí pro podmíněné větvení programu.
- Použité instrukce:
  - `SBIS PIND, 0x00` – přeskoč následující instrukci, pokud je pin č. 0 na portu D roven hodnotě 1.
- Ukázka jiné instrukce podmíněného skoku:
  - `DEC const`
  - `BRNE loop` – pokud je `const` různé od nuly skoč na návěští `loop` (využívá nulový příznakový bit).
- Použití podmínek pro překladač AVR-GCC:
  - `if( bit_is_set( PINA, 0 ))` jestliže pin č. 0 na portu A je roven 1,
  - `if( bit_is_clear( PINC, 1 ))` jestliže pin č. 1 na portu C = 0.

Podmíněné větvení programu v JSA a v C

```
.include <m16def_mod.inc>  ; vložit upravený popisný soubor pro ATmega16
.def temp = R16            ; definovat symbolický název pracovního reg. R16

reset:
    clr   temp             ; nuluj všechny bity v registru temp
    out   DDRD, temp       ; port D = vstupní
    ser   temp             ; nastav všechny bity v registru temp
    out   DDRB, temp       ; port B = výstupní

loop:
    sbis  PIND, 0x00       ; pokud pin 0 na portu D = 1 přeskoč následující instr.
    inc   temp             ; inkrementuj obsah registru temp
    sbis  PIND, 0x01       ; pokud pin 1 na portu D = 1 přeskoč následující instr.
    dec   temp             ; dekrementuj obsah registru temp
    out   PORTB, temp      ; pošli obsah temp na port B
    rjmp  loop             ; skok na návěští/místo v programu loop
```

```
#include "avr\io.h"          // vložení popisného/hlavičkového souboru AVR

int main( void )
{
    char temp = 0 ;          // deklarace lokální 8bitové proměnné temp = 0
    DDRD = 0x00 ;            // port D = vstupní
    DDRB = 0xFF ;            // port B = výstupní

    while( 1 ){
        /* pokud pin 0 na portu D = 0, inkrementuj temp */
        if( bit_is_clear( PIND, 0 )) temp++ ;
        /* pokud pin 1 na portu D = 0, dekrementuj temp */
        if( bit_is_clear( PIND, 1 )) temp-- ;
        PORTB = temp ;       // zápis hodnoty na výstupní port B
    }
    return( 1 ) ;            // návratová hodnota funkce main = 1
}
```

Zdroje informací


VÁŇA, V.
Mikrokontroléry ATMEL AVR; popis procesorů a instrukční soubor.
Ben – technická literatura, Praha, 2003, ISBN 80-7300-083-0.


LEVENTHAL, L.; WALSH, C.
Microcomputer Experimentation with the Intel SDK-85.
Prentice-Hall, Inc., New Jersey (USA), 1980, ISBN 0-13-580860-X.


Atmel Corporation.
Atmel Microcontrollers, (září 2013).
`http://www.atmel.com/products/microcontrollers/default.aspx`