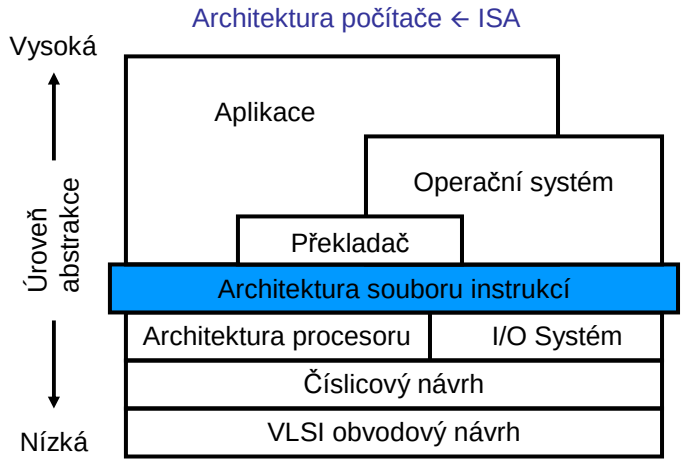
# Y36SAP 8

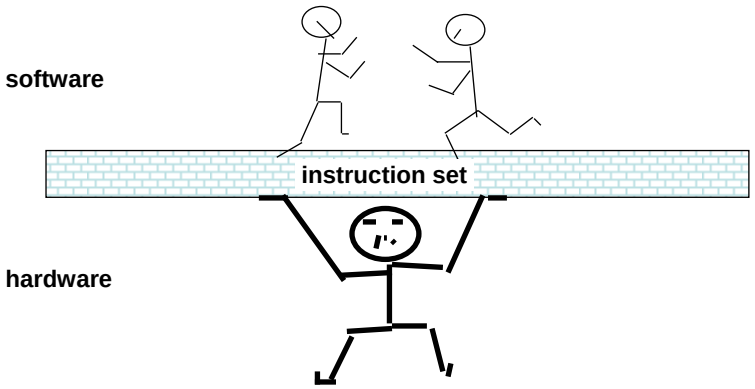
Strojový kód
Jazyk symbolických instrukcí – asembler
JSA pro ADOP a AVR



## Architektura souboru instrukcí, ISA - Instruction Set Architecture

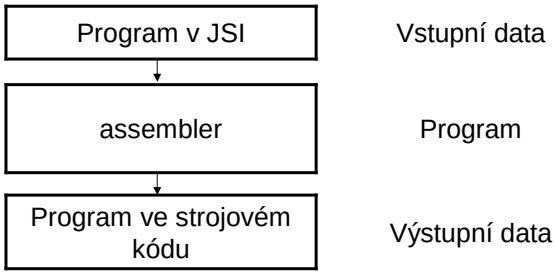


# Instrukční soubor – kritický interface

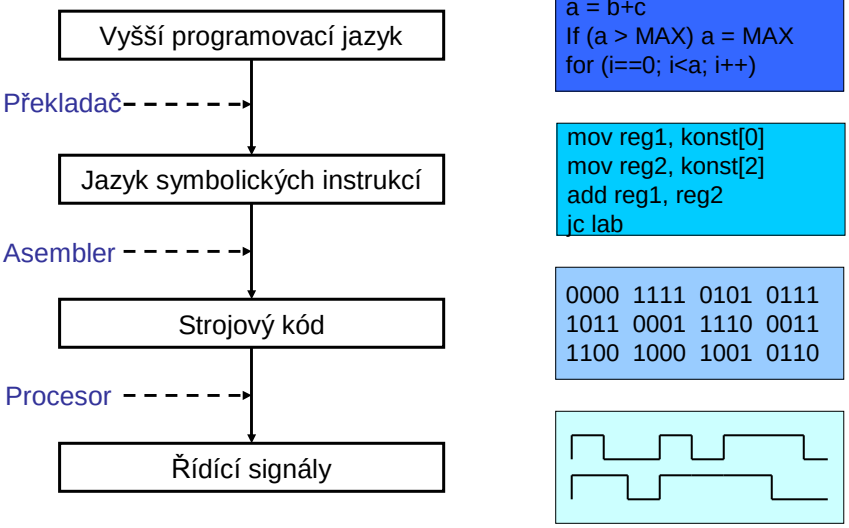





# Vytváření programů ve strojovém kódu

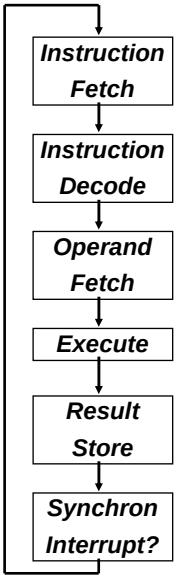


## Vývoj softwaru – úrovně abstrakce




2008-Kubátová Y36SAP-strojový kód 5

## ISA: Co musí být definováno?




**Formát a kódování instrukcí**
- Jak se instrukce dekóduje?

**Umístění operandů a výsledku**
- Kolik explicitních operandů je v instrukci pro ALU?
- Jak jsou operandy umístěny v paměti nebo jinde?
- Který operand může být v paměti?

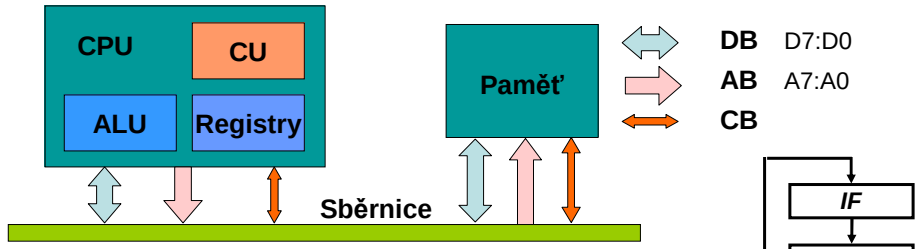
**Typy dat a velikosti operandů**

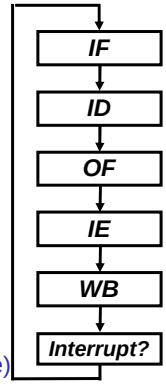
**Operace v ISA**
- Které jsou podporovány ?

**Výběr další instrukce**
- Skoky, větvení programu, volání podprogramů
- *„fetch-decode-execute“ je implicitní!*

2008-Kubátová Y36SAP-strojový kód 6

## Jednoduchý procesor von Neumannova typu




CPU musí obsahovat ALU, CU a registry pro dočasné uchování dat, instrukcí a pomocných proměnných

CPU pro běh programu prochází základním cyklem počítače:

1. Instruction Fetch — IF – načtení instrukce
2. Instruction Decode — ID – dekódování instrukce
3. Operand Fetch — OF – načtení operandu(ů)
4. Instruction Execution — IE – vykonání instrukce
5. Write Back — WB – zapsání výsledku (také Result Store)

2008-Kubátová Y36SAP-strojový kód 7

## Jaké instrukce?

- ?? bitový procesor
- Registry – kolik, jakých, jak přístupných, PC, IR, PSW, adresace?, datové?
- Paměť ?? 16 bitů adresuje 64 Kslov.... slovo??
- příznaky Z (Zero), S (Sign), C (Carry), O (Overflow) a další?
- instrukce v JSI:
  - aritmetické **ADD, SUB, AND, OR, XOR, NEG, NOT, MOV, CMP**
    - **Přímý operand, nepřímý operand**
  - řídící **JMP, JO, JNO, JS, JNS, JC, JNC, JZ, JNZ, JE, JNE, JG, JL, JGE, JLE, JA, JB, JAE, JBE**
  - uložení do paměti – **ST**
  - **HALT, OUT, IND**
- reprezentace hodnot – přímý kód, doplňkový kód, nezáporná čísla
- syntaxe jazyka symbolických instrukcí

2008-Kubátová Y36SAP-strojový kód 8

## Návrh instrukcí procesoru ADOP

- jaké
- jak kódované
- jak dlouhé
- jaká adresace operandů
- kolik registrů



## Architektura ADOP

Registry … 16 registrů dostupných programátorovi:

R0 – R11 universálních (datových) registrů

SP – ukazatel zásobníku

PC – programový čítač

PSW – stavový registr,

Příznaky Z … zero, C … carry, S … sign, O … overflow, ES … extended sign (znaménko 2. operandu v binárních operacích)

ZR – obsahuje konstantní nulu

Paměť …big endian, kapacita $2^{16}$ B – 64KB

## Přesuny dat

- MOV *kam, co*

*kam* .... registr

*co* ... registr (obsah registru), přímý operand 4, 8, 16 bitový, operand nepřímo adresovaný

- ST *co, kam*

- PUSH, POP



## Deklarace proměnných

- pseudoinstrukce
  - vyhrazení místa v paměti (pro výsledek)
  - zadání vstupních dat

data:

**SHORT** - jednobytový operand se znaménkem

**BYTE** - jednobytový operand bez znaménka

**WORD** - dvoubytový operand

deklarace ... DS, DB, DW

## Aritmetické

- binární
  - ADD
  - ADC
  - SUB
  - SBB
  - CMP
- unární
  - NOT
  - INC
  - DEC



## CMP - porovnání

jako SUB, ale neuloží výsledek, jen příznaky

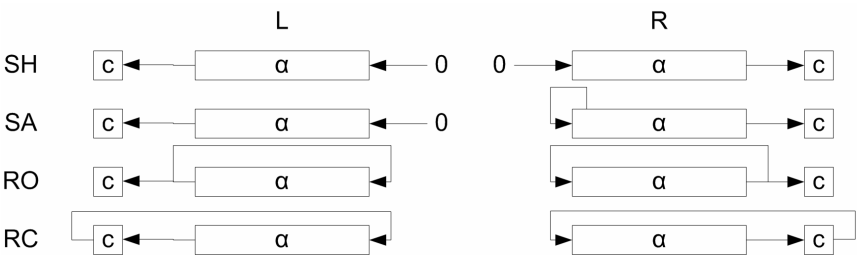
# Logické

- binární
  - AND
  - OR
  - XOR
- unární
  - NEG



# Posuvy

- SHL,
- SHR,
- ASR – aritmetický posun vpravo,
- RRC,
- RLC



# Skoky … nepodmíněný skok

**JMP label // label .... návěští**

```
Př.                MOV  r0, 1     1
                   JMP  label1    2
          cosi:    ADD  r0, r5    -
          label1:  rrc  r0        3
```



# Skoky ... podmíněné skoky

| | | |
|---|---|---|
| **JC** *náv* | … | Je-li C=1, skok na *náv (jinak nic)* |
| **JNC** *náv* | … | Je-li C=0, skok na *náv (jinak nic)* |

Př.: AX:=max(BX,CX) – nezáporná čísla

```
        MOV AX,BX
        CMP CX,BX
        JC OK
        MOV AX,CX
OK:
```

**Analogicky: JZ,JNZ,JS,JNS,JO,JNO,** JP, JNP

# Podmínky ve skocích

- Z, NZ,
- C, NC,
- S, NS
- O, NO,

Složené podmínky se používají pro vyhodnocení porovnání (CMP) resp. odečtení dvou operandů a testují logický výrazy nad příznaky:

- GE (gretaer or equal) – !SF && !OF || SF && OF (výraz je ekvivalentní s SF == OF)
- LT (less then) – negace podmínky GE,
- GT (greater then) – GE && !Z
- LE (less or equal) – negace podmínky GT,
- AT (above then) – C && !Z
- BE, (below equal) – negace podmínky AT
- TRUE – podmínka vždy splněna

| Relace | Doplňkový kód | Nezáporná čísla |
|---|---|---|
| < | JL | JB |
| ≤ | JLE | JBE |
| = | JE | JE |
| ≠ | JNE | JNE |
| ≥ | JGE | JAE |
| > | JG | JA |

E … Equal
L … Less
G … Greater
B … Below
A … Above

**Podmínka skoku: příslušné příznaky. Např.:**

**JB … CF=1 → JB=JC**

**JBE … CF∨ZF**

**JE … SF⊕OF = 1**

**JGE … SF⊕OF = 0 atd.**

2008-Kubátová Y36SAP-strojový kód 20

# Příklad

Napište program v JSI ADOP, který nalezne největší číslo v poli.



```
// nalezení maximálního prvku v poli
        mov r0, 0x8000          // doposud max hodnota do R0
        mov r2, pole    // pointer na začátek pole do R2
rep:    cmp r0, [r2]    // porovnání dosavadní maximální hodnoty
        jge next
        mov r0, [r2]    // nalezení nové maximální hodnoty
next:   add r2, 2       // přesun na další prvek v poli
        cmp r2, endPole  // otestování zda je dosaženo konce pole
        jeq end                         // skok na konec
        jmp rep                         // opakování
end:    st r0, [max]            // uložení výsledku
        halt
max: dw 0
pole:dw 1, -13, 33, 0x7777
endPole:
```

## Simulátor

http://service.felk.cvut.cz/jws/proc/procwww/

umožňuje psát programy v JSI, překládat, krokovat, spouštět a sledovat změny obsahu registrů a paměti



## LABORATOŘ - Mikropočítač **AVR**

Produkt firmy Atmel, které mají na jednom čipu integrován

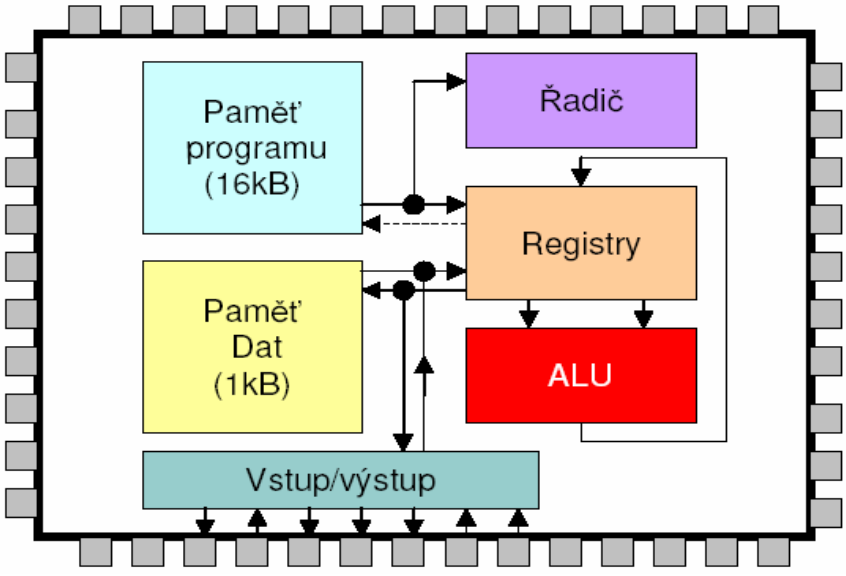
- procesor
- paměť programu
- paměť dat
- rozsáhlou sadu periferií (vstupně-výstupní brány, čítače, časovače, převodníky, komunikační rozhraní apod.)

# AVR Butterfly

- LCD displej - 6 alfanumerických znaků (po 14 segmentech)
- Joystick - 5 směrů (nahoru, dolů, doleva, doprava, kolmý stisk (enter))
- Čidlo teploty
- Čidlo osvětlení
- Krystalový oscilátor 32,768 kHz (pro měření času)
- Elektroakustický měnič (piezo)
- Napájení z baterie

2008-Kubátová Y36SAP-strojový kód 25

# Architektura **ATMEGA169**

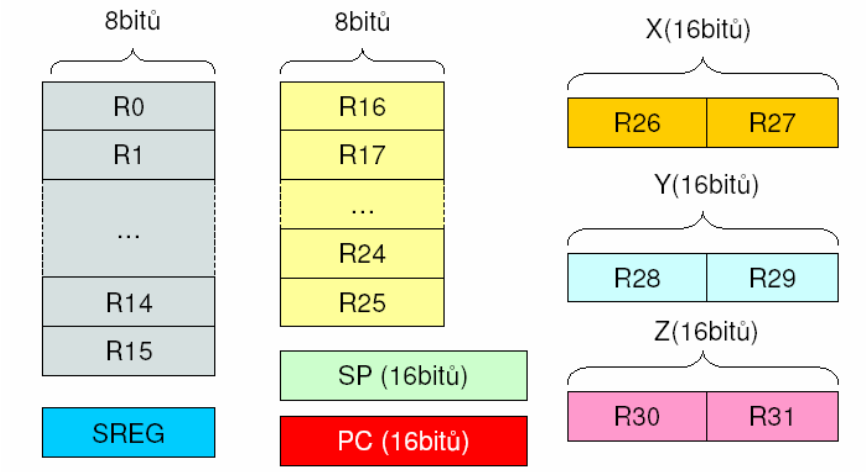


# ATMEGA169

- Univerzální registry - 32 osmibitových registrů (R0 až R31)
- Posledních šest z nich tvoří po dvojicích tři indexregistry (Slouží k adresování přístupu do paměti)
- Speciální registry
  - Programový čítač - 16bitový (Program Counter - PC)
  - Ukazatel na vrchol zásobníku - 16bitový (Stack Pointer - SP)
  - Registr příznaků - 8bitový (Status Register - SREG)
- Paměť dat: 1 KB, organizace 1K x 8 bitů, čtení i zápis
- Paměť programu: 16KB, organizace 8K x 16 bitů, pouze čtení (zápis speciálním způsobem)
- Periferie – např. řadič displeje (LCD)
- Vnitřní zdroj hodinového kmitočtu

2008-Kubátová Y36SAP-strojový kód 27

# Registry




2008-Kubátová Y36SAP-strojový kód 28

# Stavový registr – SREG („flagy“)

| C | Z | N | V | S | H | T | I |
|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | - |

**C:** **Carry flag**

**Z:** **Zero flag**

**N:** **Negative flag**

**V:** **Two‘s complement overflow indicator**

**S:** **N⊕V for signed tests**

**H:** **Half carry flag**

**T:** **Transfer bit used by BLD and BST instructions**

**I:** **Global Interrupt Enable/Disable flag**



# Skoky

- Nepodmíněný skok JMP
- Podmíněné skoky BRxx *(BR namísto J)*

změna běhu programu bez zapamatování místa odkud se „skákalo“

Na rozdíl od podprogramu, kdy je třeba si zapamatovat, kam se vrátit.

## BRBC: branch if bit in SREG is cleared

Syntax:

BRBC S,k

Operands:

$0 \le s \le 7$, $-64 \le k \le +63$

Operation:

If SREG(s) = 0 then PC ← PC + k + 1 else PC ← PC + 1

Program counter:

PC ← PC + k + 1

PC ← PC + 1 if condition is false

**Podobně BRBS**

2008-Kubátová Y36SAP-strojový kód 31

## Další podmíněné skoky

- BRCS – BRCC ..... S – set, C – clear

Syntax: BRCS k

- BREQ ... Z=1, BRNE ..... Z=0,
- BRGE ... N xor V = 0
- ....
- Procesor umí nastavit či vynulovat příznaky v SREG

2008-Kubátová Y36SAP-strojový kód 32

# Podprogramy

- Volání ... CALL k *k je adresa*

Program counter:
PC ← k

Stack:
STACK ← PC + 2
SP ← SP – 2, (2 bytes, 16 bits)

*post-dekrementace*

- Návrat z podprogramu .... RET

Při volání se ukládá návratová adresa na zásobník,
Při návratu se obnovuje původní obsah PC ze zás.

*pre-inkrementace*

Stack:
SP ← SP + 2, (2 bytes, 16 bits)



# Zásobník - stack

Operace:

PUSH Rr... uložení registru na zásobník

Rr → STACK

Program counter:
PC ← PC + 1

Stack:
SP ← SP – 1

POP Rd ... nahrání registru ze zásobníku

Rd ← STACK

Program counter:
PC ← PC + 1

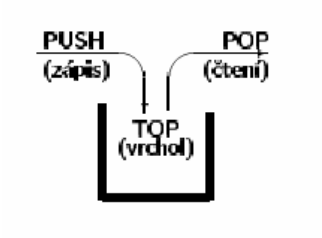
Stack:
SP ← SP + 1

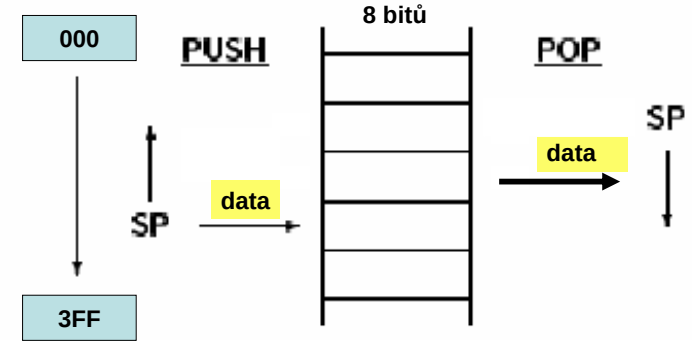
# Zásobník ...

- Zásobník roste směrem k nižším adresám
- Obsah SP registru ukazuje na vrchol zásobníku – na první prázdnou položku, na kterou se bude zapisovat např. instrukcí PUSH
- Je simulován v hlavní paměti



# .. zásobník



## Přesuny dat

- MOV ... Přesun dat mezi registry
- LD, LDS ... Přesun dat z paměti do registru
- LDI ... Nahrání konstanty do registru
- ST ... Přesun dat z registru do paměti

Poznámka: LD a ST používá nepřímou adresaci



## Posuvy

- LSL, LSR – posuvy logické (SHIFT)
- ROL, ROR – rotace přes Carry
- ASR – aritmetický posuv vpravo

(? Vlevo?)

..... A řada dalších instrukcí, viz web SAP