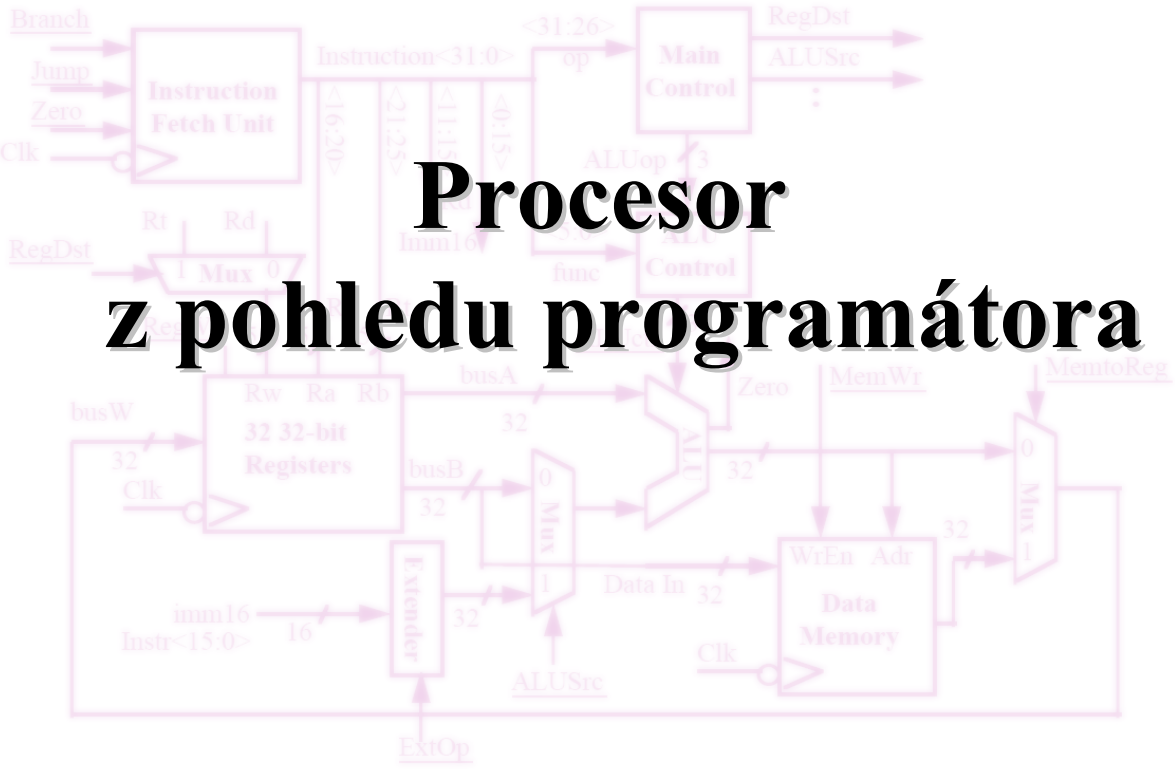
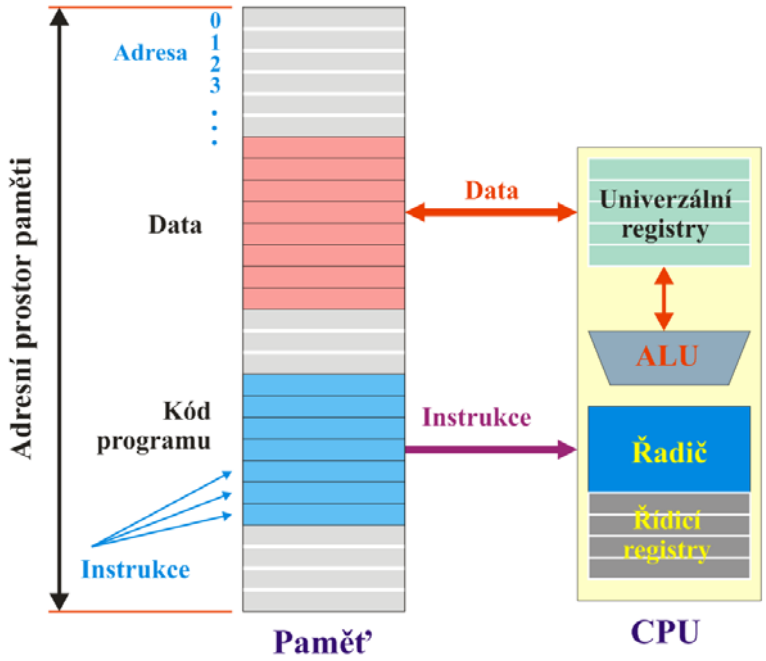


Procesor z pohledu programátora

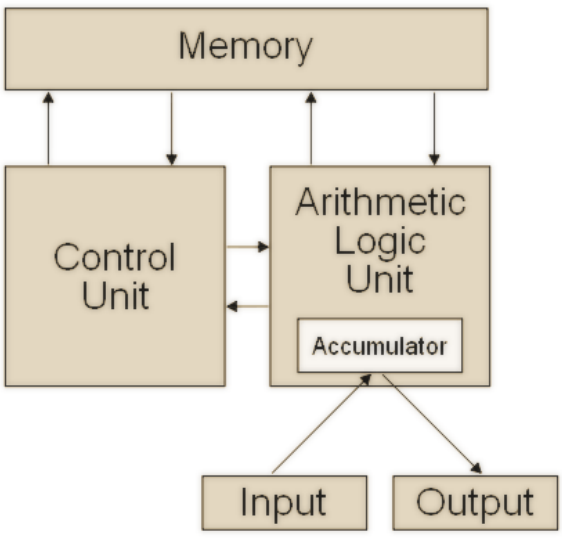
Terminologie

- **Procesor** (CPU) = řadič + ALU .
- **Mikroprocesor** = procesor vyrobený monolitickou technologií na 1 čipu.
- **Mikropočítač** = počítač postavený na bázi mikroprocesoru.
- **Mikrokontrolér** = mikropočítač se speciálními periferiemi vyrobený na 1 čipu.

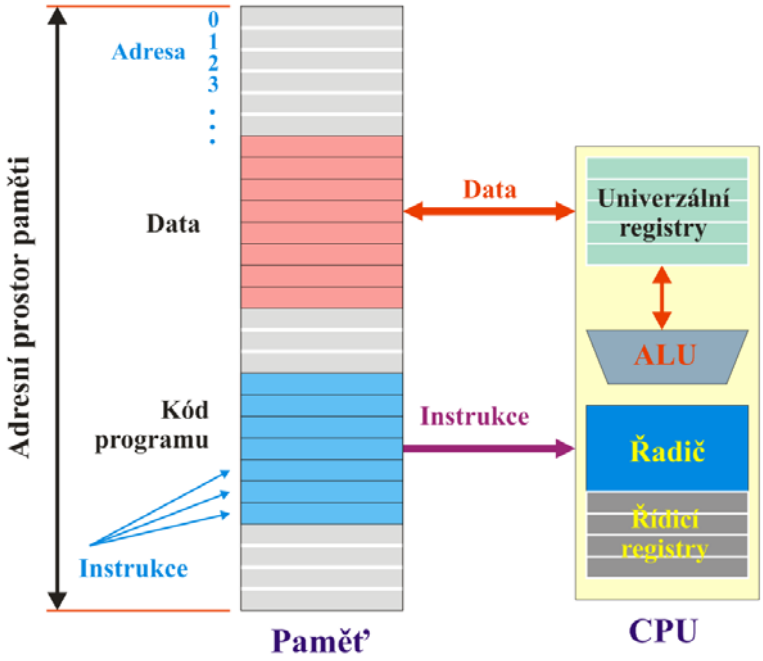
Poznámka: CPU a paměť



Poznámka: CPU a paměť

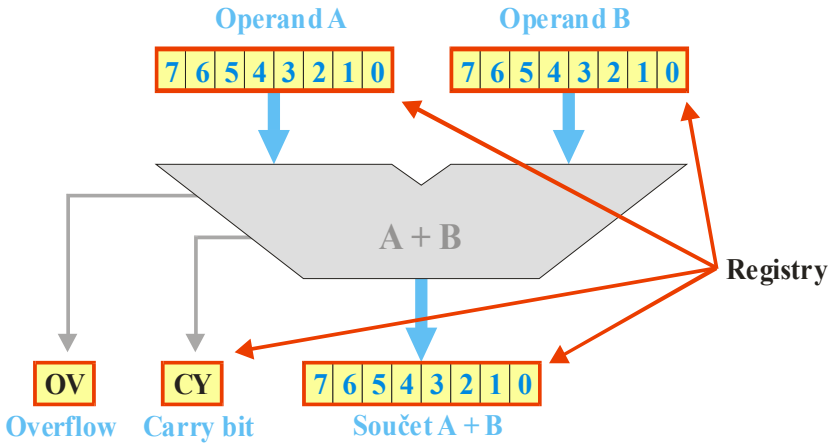


Poznámka: CPU a paměť



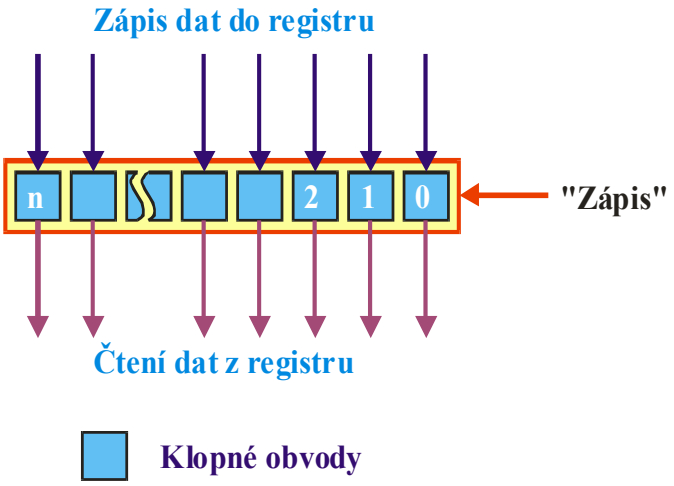
Poznámka: Registr

- Registr slouží v počítači jako dočasná paměť pro uložení určité hodnoty (operandy pro aritmetické operace, kód instrukce, ...).



Poznámka: Registr

- Technická realizace: Registr je sestaven z klopných obvodů.
- V každém klopném obvodu je uložen 1 bit.



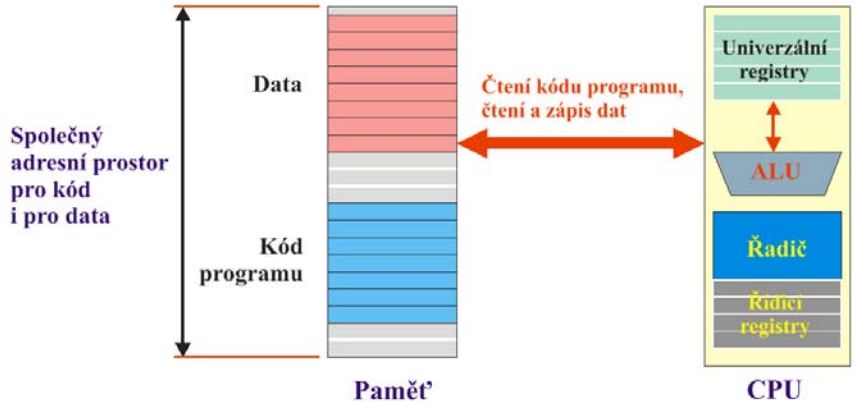
Architektura počítače

- Dvě základní koncepce
  - **Von Neumannova** (Princetonská) architektura: společný adresní prostor pro data i pro kód programu.
  - **Harwarská** architektura: data a kód programu jsou v oddělených adresních prostorech.

Von Neumannova architektura

Paměť (adresní prostor) je společná pro kód programu i pro data

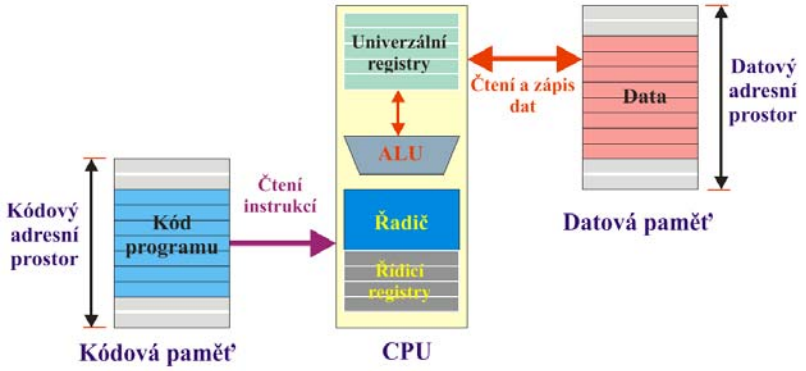
- Výhody:
  - poměr velikostí kód/data lze měnit podle okamžité potřeby,
  - procesor má přístup do kódové paměti i pro zápis.
- Nevýhody:
  - procesor nemůže současně číst kód i data ⇒ omezení rychlosti.



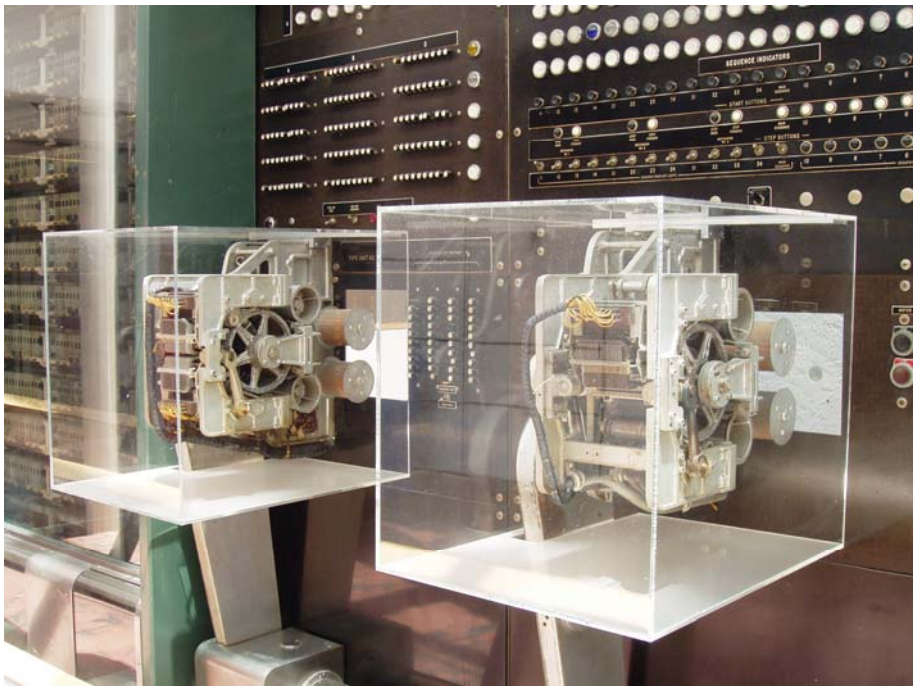
Harwardská architektura

Adresní prostory datové paměti a kódové paměti jsou oddělené

- Výhody:
  - možnost číst současně data i kód programu (⇒ vyšší rychlost),
  - šířka slova kódové paměti může být optimalizovaná.
- Nevýhody:
  - poměr velikostí datové a kódové paměti je pro daný případ pevný,
  - někdy komplikovanější zavádění a ladění programu.



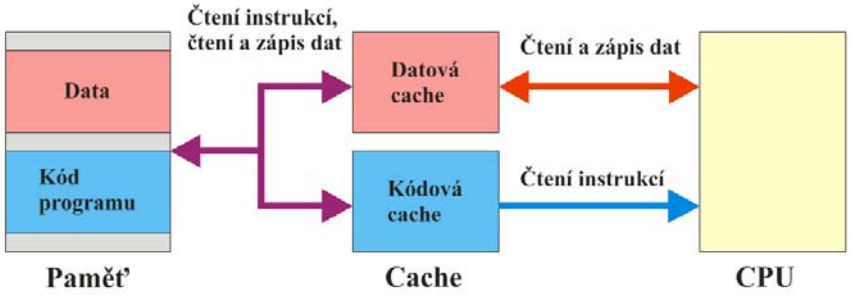
Harwardská architektura



Snímače 24stopé děrné pásky pro čtení instrukcí (Mark I)

Kombinace obou architektur

Často se používá Von Neumannova architektura se samostatnou datovou a kódovou cache.

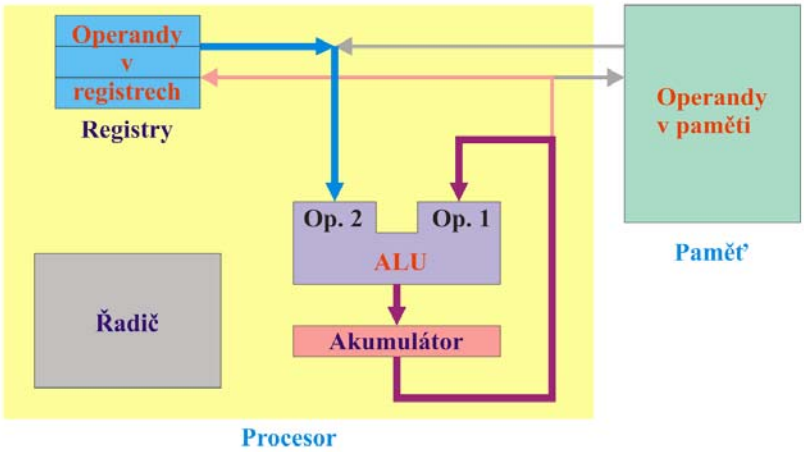


„Programátorsky zajímavé“ vlastnosti procesoru

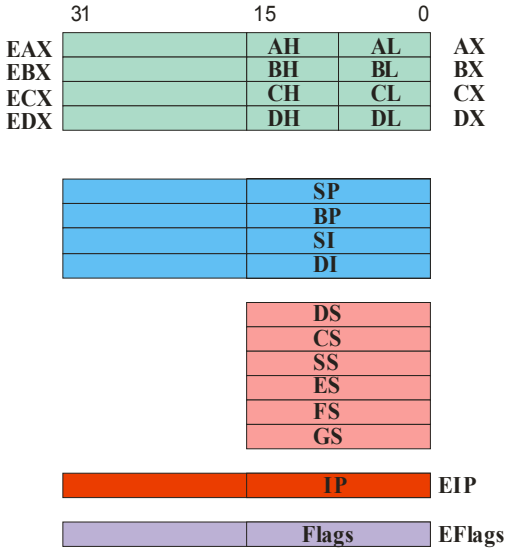
- Základní uspořádání (architekturu) procesoru popisuje tzv. ISA (Instruction Set Architecture):
  - Registrová sada procesoru,
  - Instrukční soubor,
  - Adresní prostory (paměti a IO).
- Rozlišují se 2 základní typy ISA:
  - Akumulátorově orientovaná,
  - Registrově orientovaná.
  - (Případně další spíše teoretické možnosti, např. zásobníkově orientovaná).

Akumulátorově orientovaná ISA

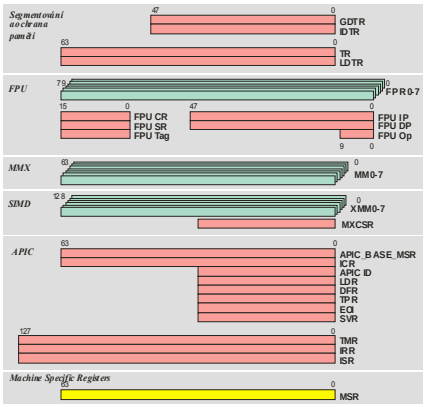
- Instrukční soubor orientován na použití **akumulátoru**.
  - Akumulátor (ACC) = speciální registr pro uložení operandů.
- Typická operace: ACC ← ACC * Operand_2.
- Operand_2 může být v univerzálních registrech nebo v paměti.
- Obvykle malý počet univerzálních registrů.

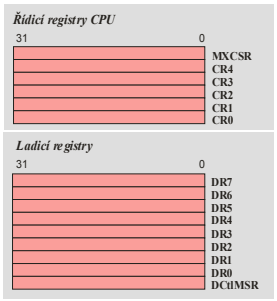


Registrová sada procesoru (Intel IA-32)



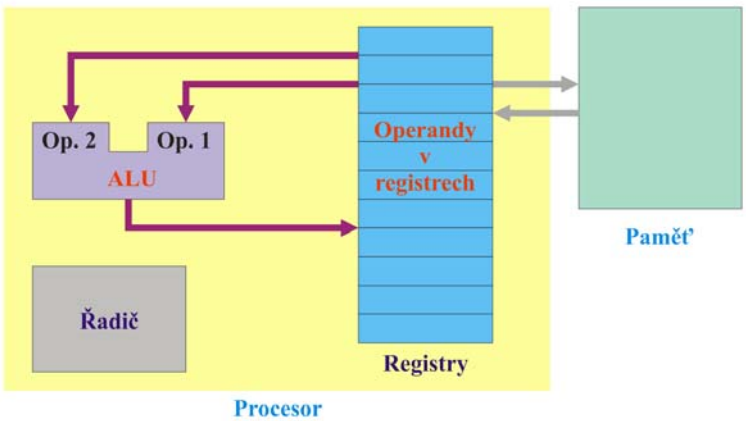
- 4 ks. univerzálních registrů 32 bitů (EAX = akumulátor),
- 4 ks. bázových a indexových registrů,
- 6 ks. segmentových registrů,
- IP (PC) a příznakový registr,
- 8 ks. FP registrů 80 bitů,
- 8 ks. SIMD registrů 128 bitů,
- 8 ks. MMX registrů 64 bitů.





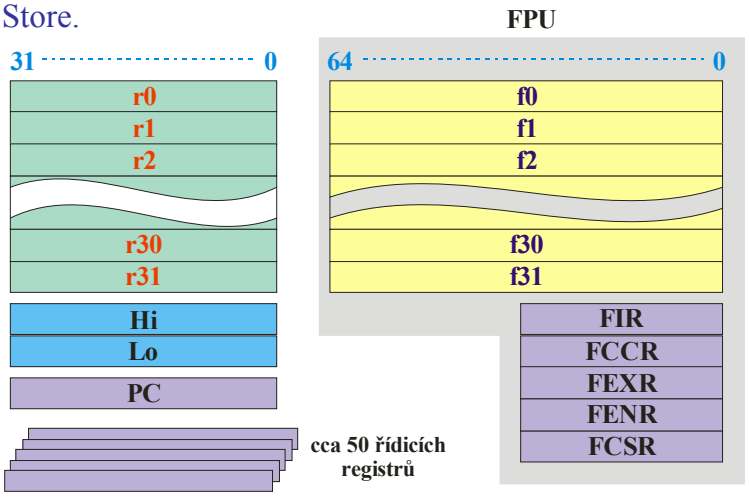
Registrově orientovaná ISA

- Instrukční soubor orientován na použití registrů.
- Typická operace: Registr_3 ← Registr_1 * Registr_2.
- Nelze přímo použít operandy v paměti.
  - Operandy je nutné nejprve uložit do registrů (architektura Load – Store).
- Obvykle velký počet registrů (typicky 32).



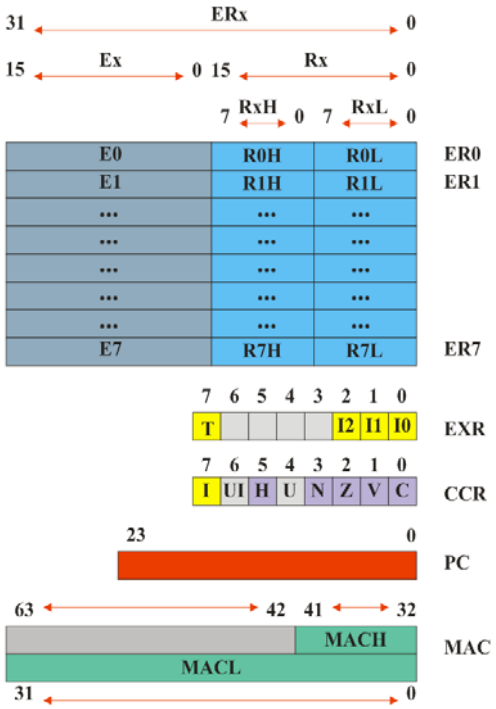
Registrová sada procesoru (MIPS)

- 32 ks. 32bitových registrů pro operandy interger (r0 ... r31),
- 32 ks. 64bitových registrů pro operandy float (f0 ... f31),
- cca 50 řídicích registrů.
- Nemá příznakový registr a SP.
- Všechny operace jsou typu registr ↔ registr, s pamětí pracují pouze instrukce Load a Store.



Registrová sada procesoru H8S

- **R*x*L, R*x*H** – 8bitové registry
- **R*x*, E*x*** – 16bitové registry
- **ER*x*** – 32bitové registry
- **EXR** – řídicí registr
- **CCR** – příznakový registr
- **PC** – programový čítač (24 bitů)
- **MAC** – Multiply Accumulate
- **ER7** slouží implicitně jako **SP**

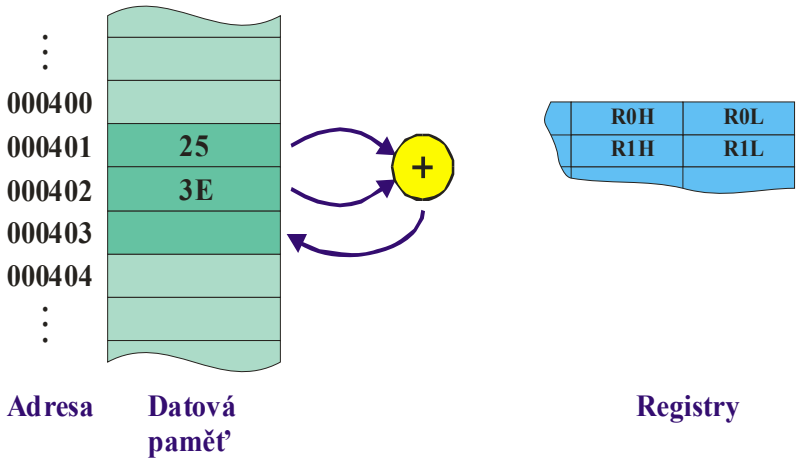


PC, SP, CCR, EXR

- **PC** = Program Counter
  Při provádění určité instrukce obsahuje adresu následující instrukce.
- **SP** = Stack Pointer
  Ukazatel na vrchol zásobníku. U H8S slouží jako SP registr ER7.
- **CCR** = Condition Code Register
  Obsahuje soubor příznaků pro větvení programu.
- **EXR** = Extended Control Register
  Pro řízení přerušení a trasování.

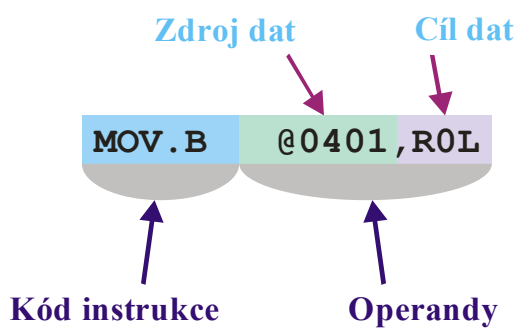
Použití registrů

Jednoduchý program: sečtení dvou čísel v paměti, uložení výsledku do paměti
- H8S neumí pracovat s operandy v paměti → operandy se musí uložit do registrů,
- výsledek operace sčítání je v registru → musí se uložit do paměti.

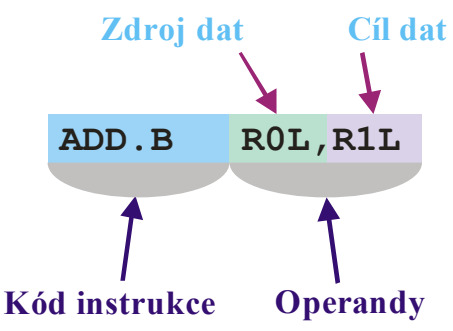


Poznámka: Instrukce (1)

- Zjednodušeně:



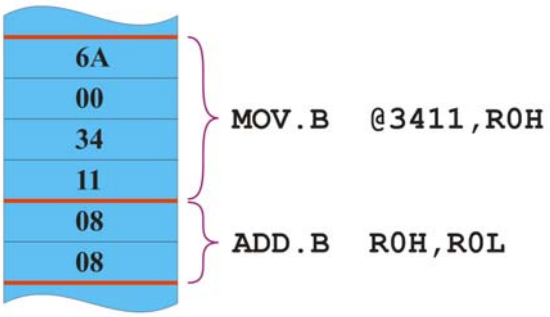
R0L ← @0401



R1L ← R0L + R1L

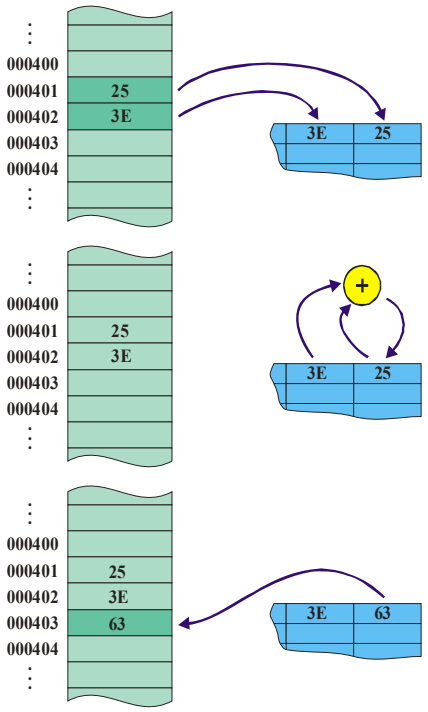
Poznámka: Instrukce (2)

- Instrukce je v paměti zakódována v několika bytech.



Jednoduchý program

```
MOV.B   @0401,R0L
MOV.B   @0402,R0H
ADD.B   R0H,R0L
MOV.B   R0L,@0403
```



Registr PC

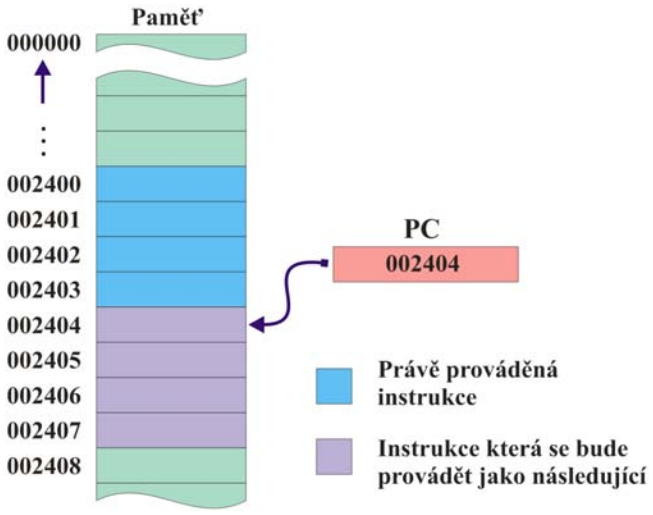
PC = Program Counter
(Programový čítač)

Při provádění určité instrukce obsahuje adresu následující instrukce.

PC se mění:

1. Automaticky při čtení instrukcí z paměti.
2. Programově – provedením skokové instrukce.

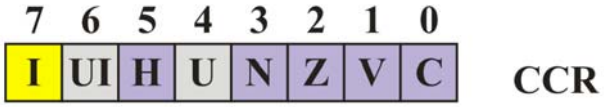
V H8S je PC dlouhý 24 bitů ⇒ může adresovat paměť v rozsahu 2^{24} = 16 M bytů.



Funkce PC

Registr CCR

- Obsahuje příznakové bity (flag).
- Nastavuje se automaticky podle výsledku provedené operace (kromě bitu I).
  - V případě potřeby lze použít např. instrukci CMP (Compare).
- Použití: ke větvení programu.
  - Pro větvení programu slouží instrukce Bcc.



Příznakové bity u H8S:

C	Carry
V	Overflow
Z	Zero (nulový výsledek)
N	Negative (znaménkový bit výsledku)
H	Half Carry
I	Interrupt Enable (povolení / zákaz přerušení)
U, UI	Nevyužité

Větvení programu podle příznaků (1)

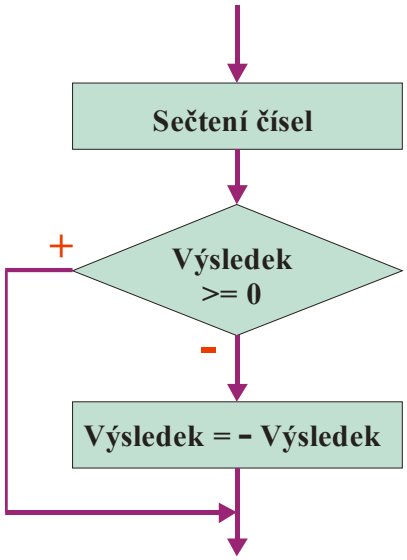
- K větvení programu slouží instrukce **Bcc** (Branch Conditionally).
- Testuje jeden nebo několik příznakových bitů.
- Podle výsledku testu buď provede nebo neprovede skok (přičtení určité hodnoty k PC).
- Názvy instrukcí odpovídají situaci při předchozí instrukci **CMP Y,X** .

Mnemonic	Meaning	cc	Condition	Signed/Unsigned*
BRA (BT)	Always (true)	0000	True	
BRN (BF)	Never (false)	0001	False	
BHI	HIgh	0010	C∨Z = 0	X > Y (unsigned)
BLS	Low or Same	0011	C∨Z = 1	X ≤ Y (unsigned)
BCC (BHS)	Carry Clear (High or Same)	0100	C = 0	X ≥ Y (unsigned)
BCS (BLO)	Carry Set (LOw)	0101	C = 1	X < Y (unsigned)
BNE	Not Equal	0110	Z = 0	X ≠ Y (unsigned or signed)
BEQ	EQual	0111	Z = 1	X = Y (unsigned or signed)
BVC	oVerflow Clear	1000	V = 0	
BVS	oVerflow Set	1001	V = 1	
BPL	PLus	1010	N = 0	
BMI	MInus	1011	N = 1	
BGE	Greater or Equal	1100	N⊕V = 0	X ≥ Y (signed)
BLT	Less Than	1101	N⊕V = 1	X < Y (signed)
BGT	Greater Than	1110	Z∨(N⊕V) = 0	X > Y (signed)
BLE	Less or Equal	1111	Z∨(N⊕V) = 1	X ≤ Y (signed)

Větvení programu podle příznaků (2)

Příklad:

Výpočet absolutní hodnoty ze součtu dvou čísel.



Větvení programu podle příznaků (3)

Příklad:

Výpočet absolutní hodnoty ze součtu dvou čísel.

```
        MOV.B   @0401,R0L   ; 1. operand do R0L
        MOV.B   @0402,R0H   ; 2. operand do R1L
        ADD.B   R0H,R0L     ; nastaví příznaky
        BPL     LAB1        ; skok při N == 0
        NEG.B   R0L         ; dvojkový doplněk
LAB1:   MOV.B   R0L,@0403   ; uložení výsledku
```

Poznámka:

`LAB1` je návěští, které označuje instrukci `MOV.B R0L,@0403`.

Použití dalších registrů

- Také registry **SP** a **EXR** mají speciální použití.
- **SP** je pro nás důležitý, ale nejprve probereme obecné možnosti adresování dat a instrukcí v paměti a kódování instrukcí.
- **EXR** má využití při obsluze přerušení a při ladění programu → **pro nás** není důležitý.