Mikroprocesorová technika

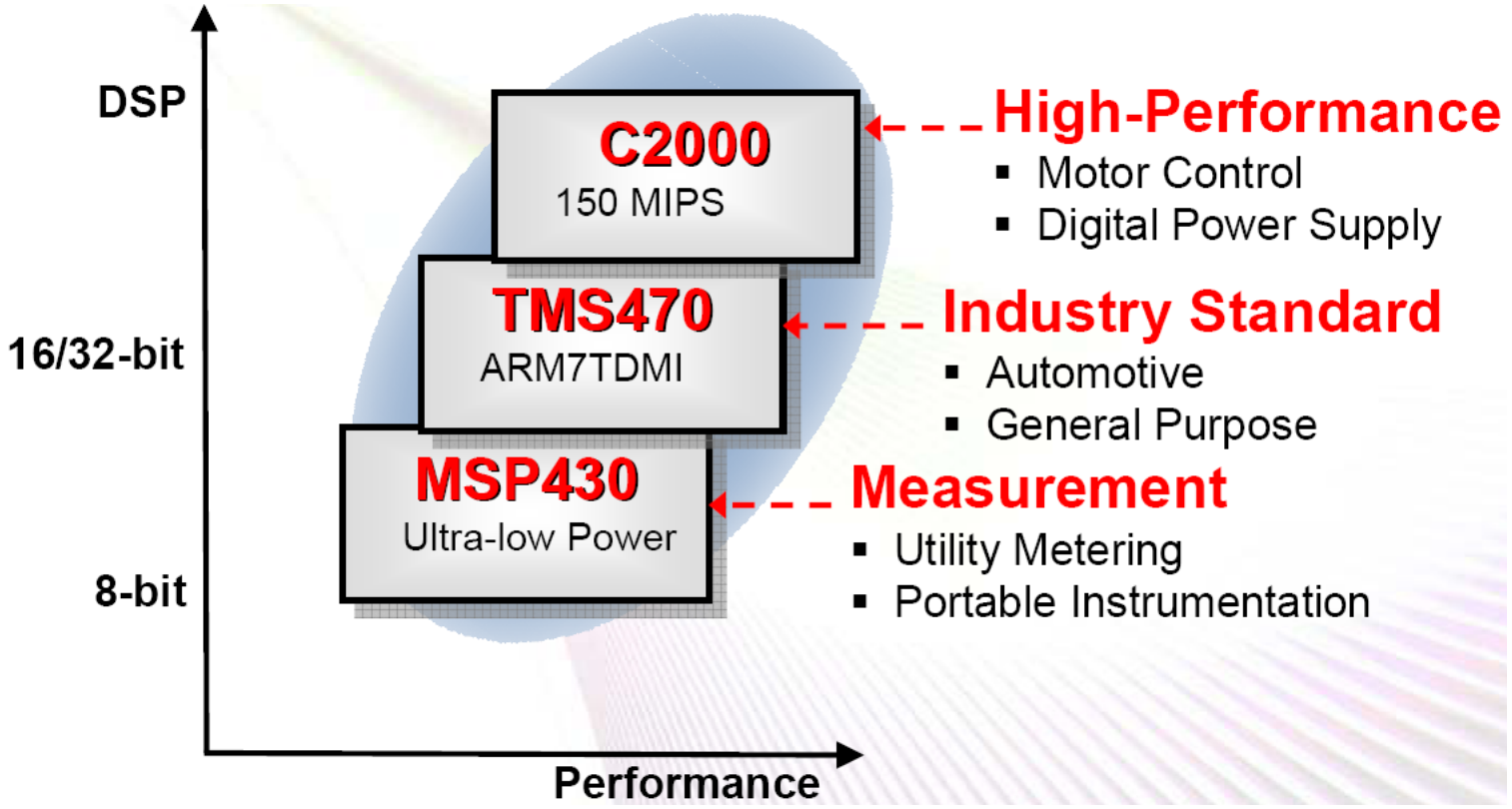
Prednáška č. 1

Centrálna procesorová jednotka – CPU,

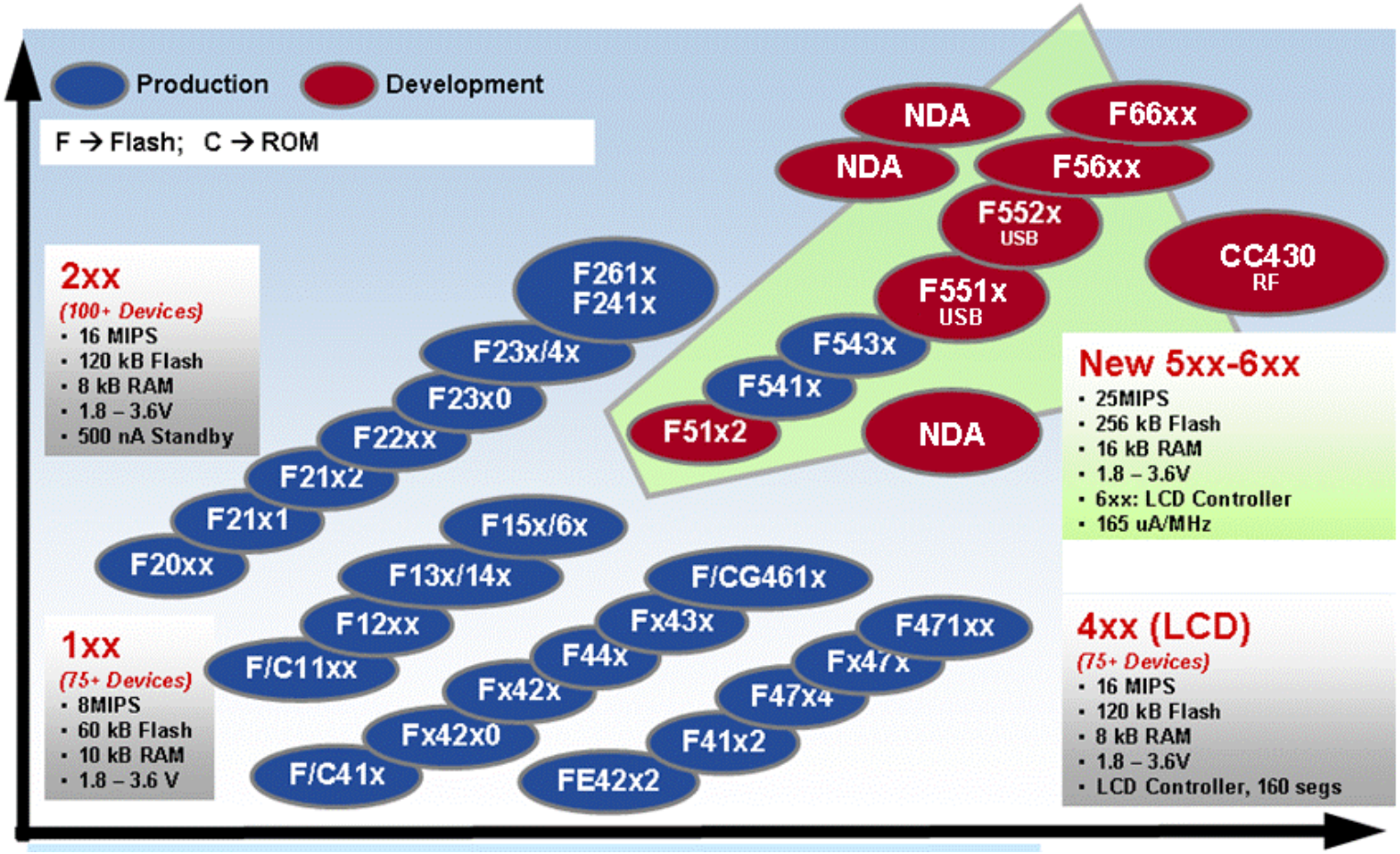
pamäťový model, režimy adresovania, inštrukčný súbor

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
KATEDRA RÁDIOELEKTRONIKY
Laboratórium DSP a mikroradičov

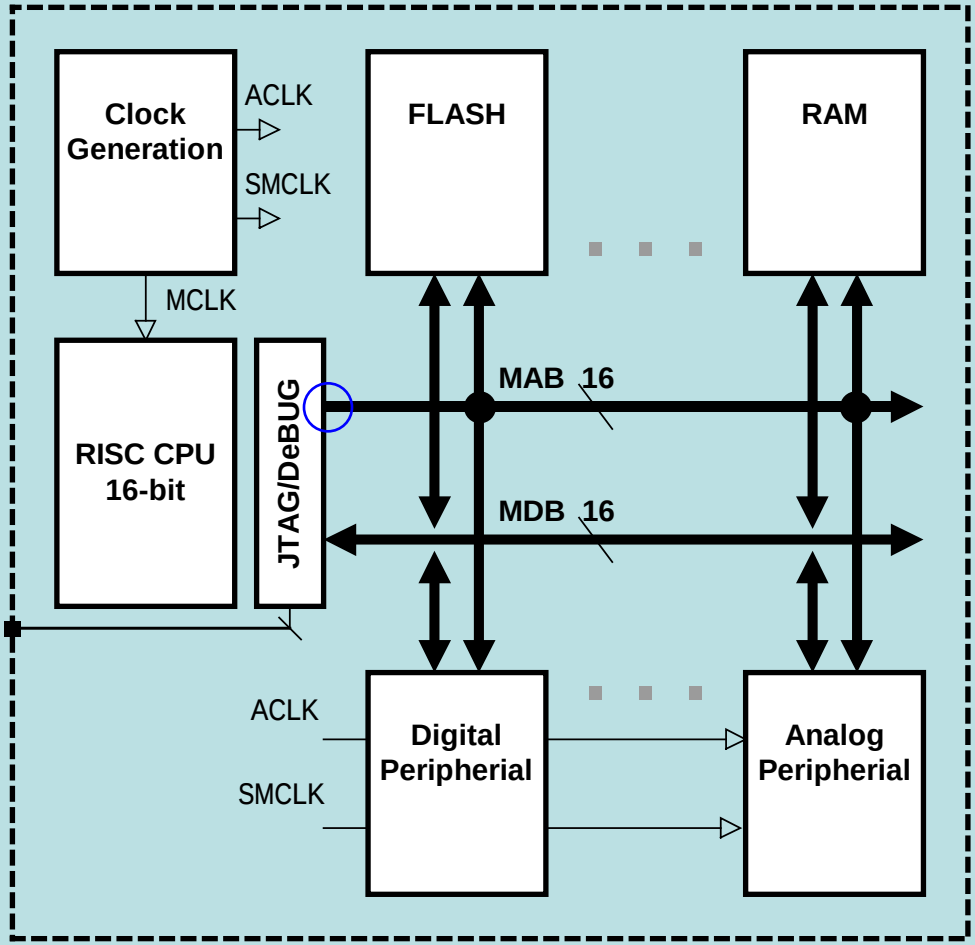
:: Mikroradiče firmy Texas Instruments



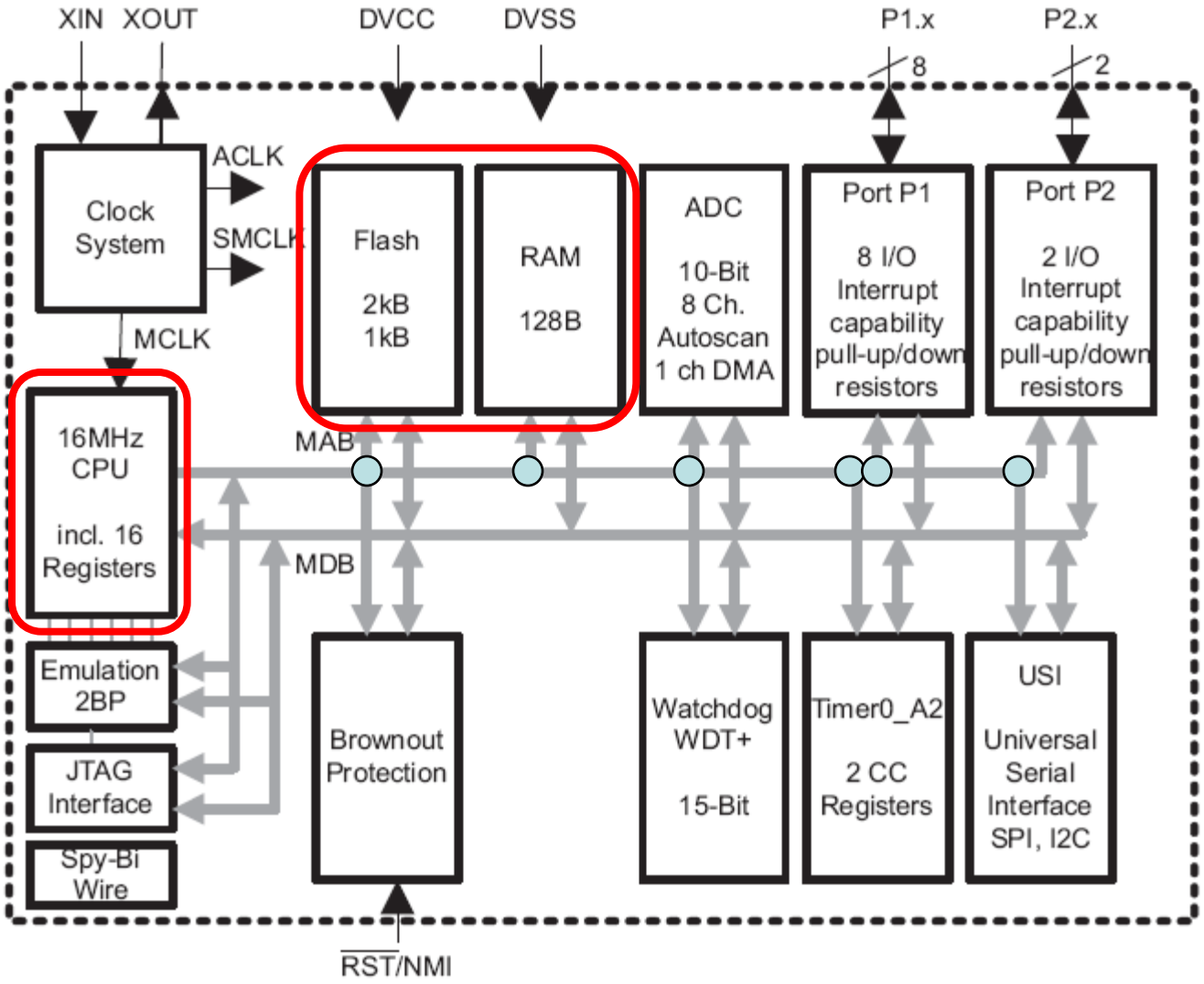
:: Mikroradiče MSP430



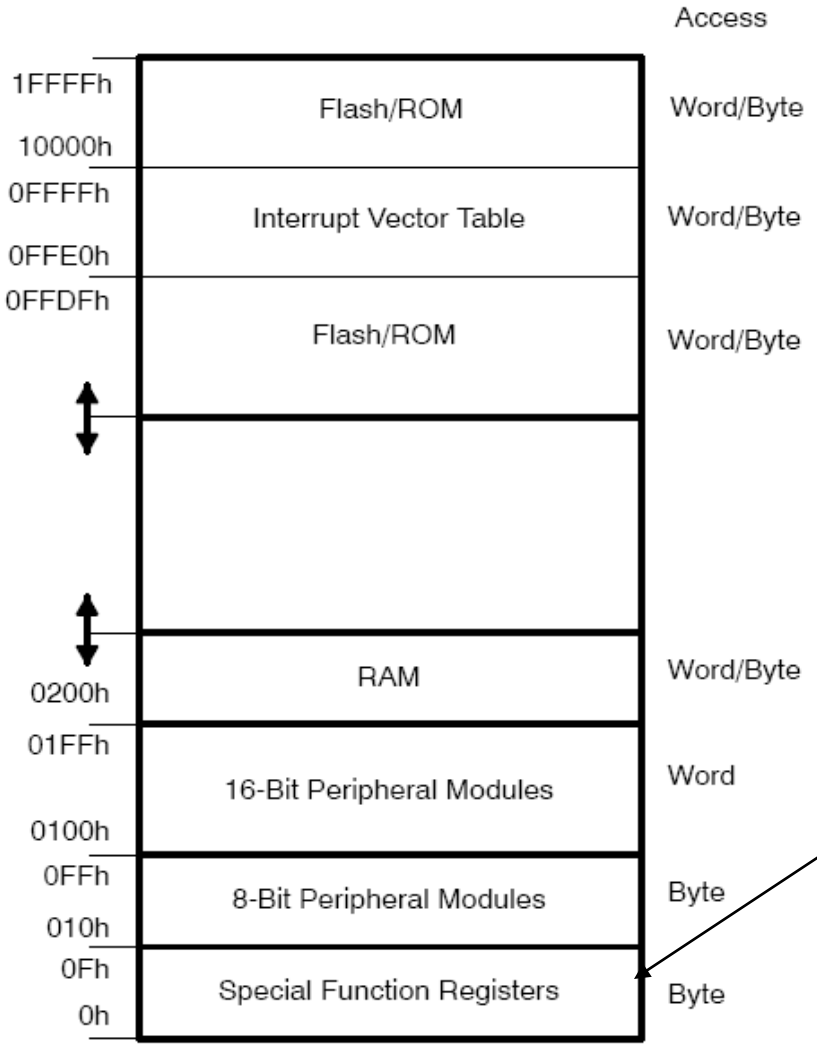
:: Architektúra jadra MSP430



:: Funkčná bloková schéma MSP430G2231



:: Pamäťový model MSP430G2231



MSP430 je mikroradič založený na von-Neumannovej architektúre
→ má jeden adresovací priestor zdieľaný:

- registrami so špeciálnymi funkciami (SFR),
- registrami pre konfiguráciu periférií,
- dátovou pamäťou RAM
- a programovou pamäťou FLASH.

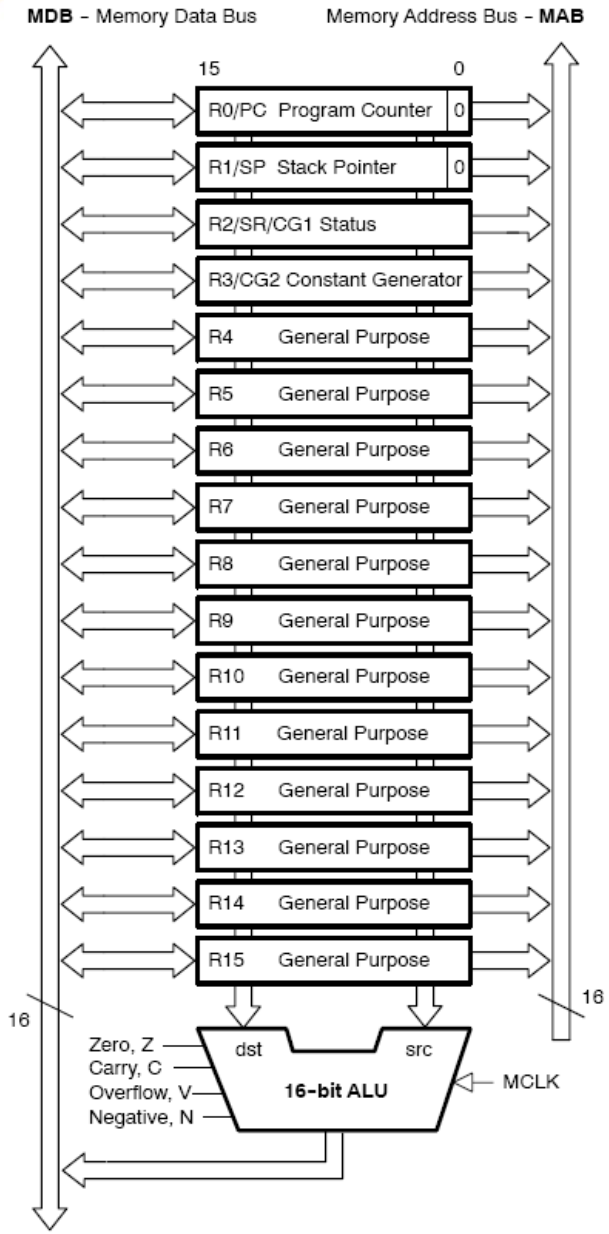
:: Vlastnosti CPU

- vlastnosti CPU mikroradiča MSP430G2231 sú navrhnuté s ohľadom na moderné programovacie techniky, ako je napr. práca s tabuľkami a používanie vyšších programovacích jazykov
- CPU je schopná adresovať celý rozsah pamäťového priestoru bez potreby stránkovania
- architektúra RISC s 27 inštrukciami a 7 adresovacími režimami
- ortogonálna architektúra (každá inštrukcia môže využívať každý adresovací režim)
- plný prístup ku všetkým registrom zahŕňajúc špeciálne registre (programové počítadlo, stavový register a smerník na zásobník)
- všetky operácie s registrami sú jednocyklové

:: Vlastnosti CPU

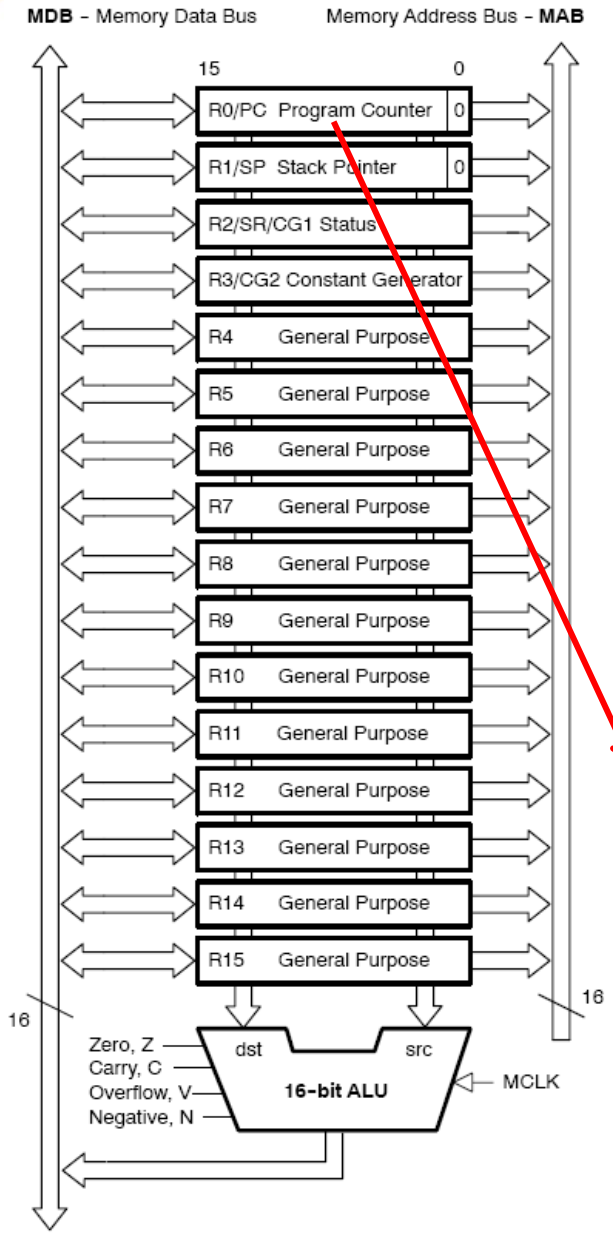
- 16-bitový registrový súbor redukuje počet prístupov do pamäti
- 16-bitová adresová zbernica umožňuje priamy prístup a pohyb v rámci celého pamäťového priestoru
- 16-bitová dátová zbernica umožňuje priamu manipuláciu so slovne (word) orientovanými argumentami
- generátor konštánt poskytuje šesť najčastejšie používaných konštánt a umožňuje tak redukovať dĺžku zdrojového kódu
- priamy transfer dát medzi oblasťami pamäte bez potreby používať registre pre medzipresuny
- formáty inštrukcií pre slovné (word, W, 16 bitov) aj bajtové (byte, B, 8 bitov) adresovanie

:: Funkčná bloková schéma CPU



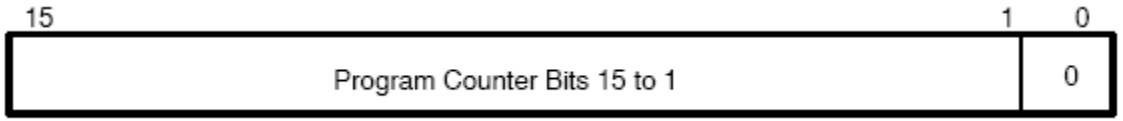
- CPU zahŕňa šesťnásť 16-bitových registrov
- R0, R1, R2 a R3 majú špeciálne funkcie
- R4 až R15 nazývame pracovné registre a sú určené pre všeobecné použitie

:: Registre CPU

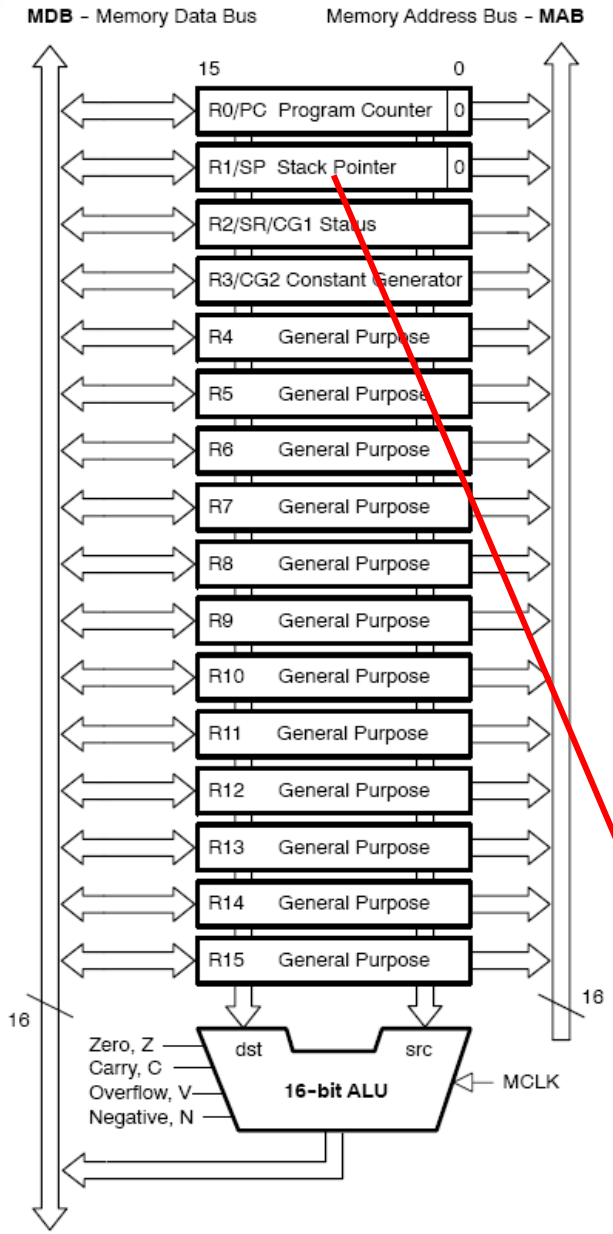


Programové počítadlo

- 16-bitový register (PC/R0) drží adresu inštrukcie, ktorá bude vykonaná po skončení vykonávania aktuálne spracovávanej inštrukcie
- inštrukcie v pamäti zaberajú vždy párny počet bytov (2, 4 alebo 6) a obsah PC je na základe tohto inkrementovaný
- PC môžu adresovať všetky inštrukcie s využitím všetkých adresovacích režimov

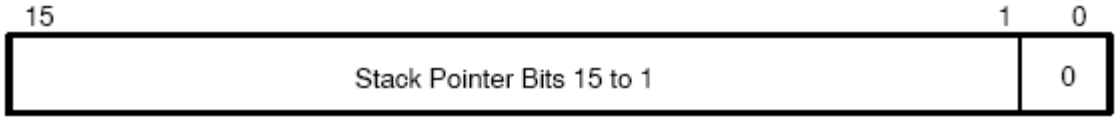


:: Registre CPU

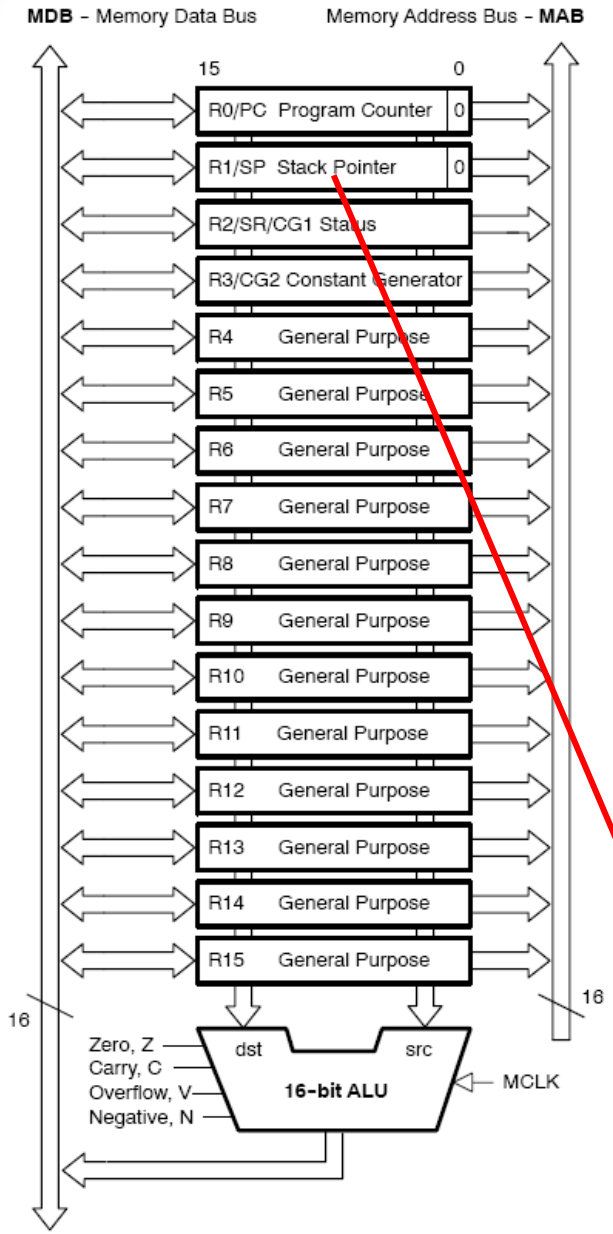


Smerník na zásobník (SP/R1)

- CPU využíva tento register k definovaniu vrcholu zásobníka a ten potom k odkladaniu návratových adries pri volaní podprogramov alebo prerušení
- využíva režim preddekrementácie a postinkrementácie
- SP môžu adresovať všetky inštrukcie s využitím všetkých adresovacích režimov
- inicializáciu registra SP zabezpečuje programátor na začiatku zdrojového kódu, smerník vždy ukazuje na nejakú pozíciu v pamäti RAM (párne adresy)

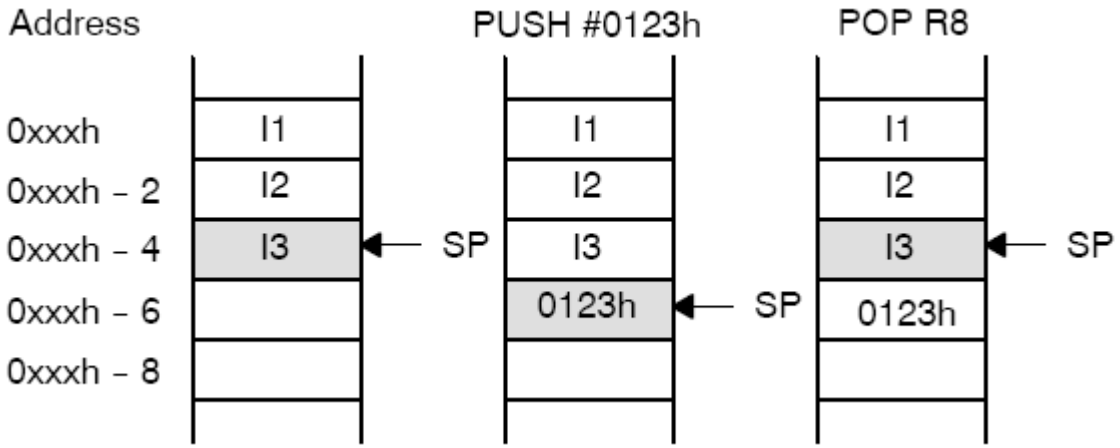


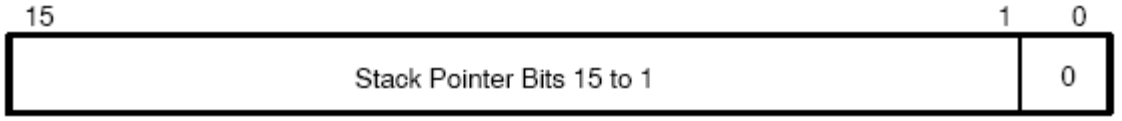
:: Registre CPU



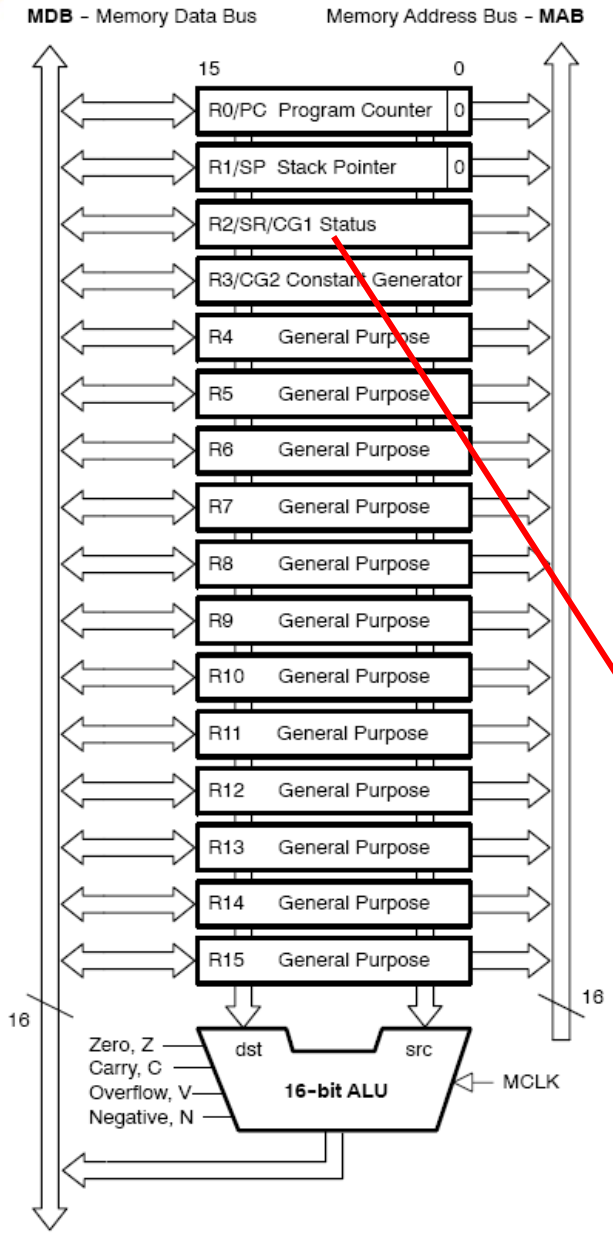
Smerník na zásobník (SP/R1)

- príklad použitia:



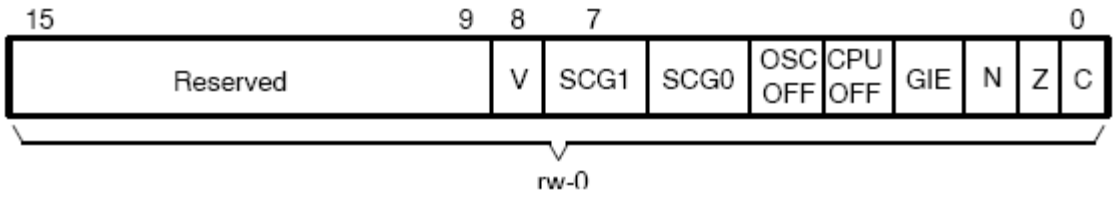


:: Registre CPU



Stavový register (SR/R2)

- umožňuje niektoré základné nastavenia procesora
- drží stav po vykonaní poslednej aritmetickej inštrukcie
- používa sa ako zdrojový aj cieľový register
- je možné ho použiť iba v registrovom adresovacom režime s inštrukciou orientovanou slovne

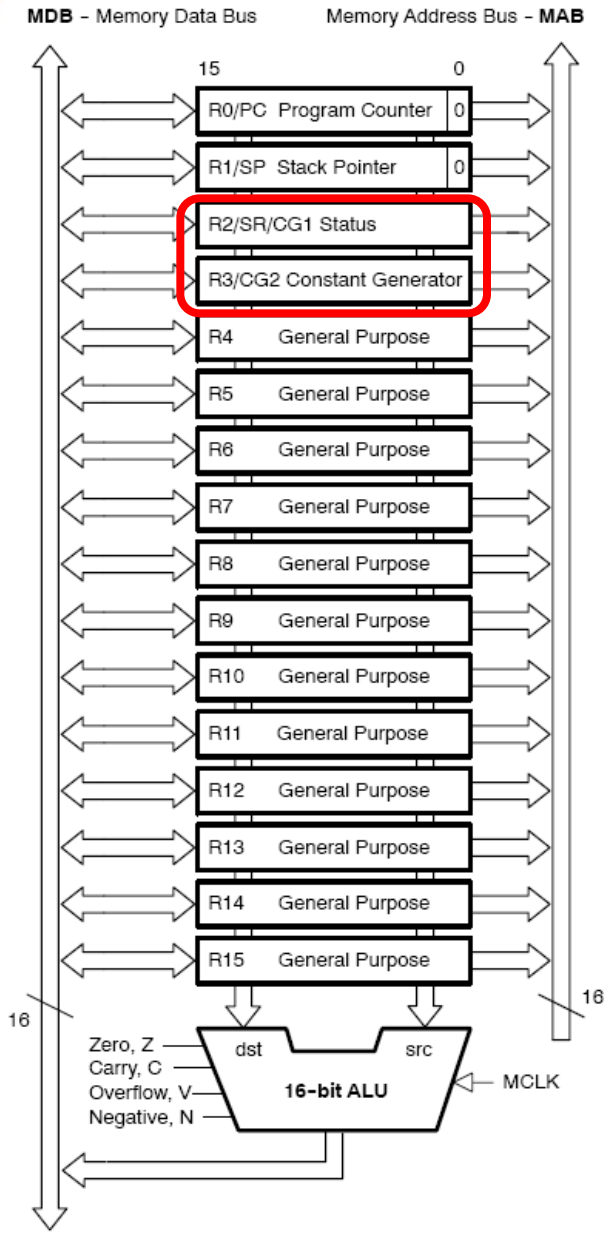


:: Registre CPU

Stavový register (SR/R2), opis bitov

V	**Pretečenie.** Bit je nastavený v prípade, keď výsledok aritmetickej operácie prekročil rozsah v závislosti od znamienka. **ADD(.B),ADDC(.B)** Bude nastavený keď: Positive + Positive = Negative, Negative + Negative = Positive, inak bude zmazaný. **SUB(.B),SUBC(.B),CMP(.B)** Bude nastavený keď: Positive − Negative = Negative, Negative − Positive = Positive, inak bude zmazaný.
SCG1	**System Clock Generator 1.** Nastavením bitu vypíname SMCLK.
SCG0	**System Clock Generator 0.** Nastavením bitu vypíname dc generátor DCO, ak DCOCLK nie je použitý pre MCLK alebo SMCLK.
OSCOFF	**Oscillator Off.** Nastavením bitu vypíname kryštálový oscilátor LFXT1, ak LFXT1CLK nie je použitý pre MCLK alebo SMCLK.
CPUOFF	**CPU Off.** Nastavením bitu vypíname CPU.
GIE	**General Interrupt Enable.** Nastavením bitu povoľujeme maskovateľné prerušenia. resete sú všetky maskovateľné prerušenia zakázané.
N	**Negative bit.** Ak je výsledok bytovej alebo slovnej operácie záporný, bit je nastavený, inak je zmazaný. v Bit je nastavený podľa MSb, t.j. v prípade slovnej operácie je bit nastavený podľa 15. bitu výsledku a v prípade bytovej operácie podľa 7. bitu.
Z	**Zero bit.** Ak je výsledok bytovej alebo slovnej operácie nula, bit je nastavený, inak je zmazaný.
C	**Carry bit.** Ak výsledok bytovej alebo slovnej operácie spôsobí prenos do vyššieho rádu, bit je nastavený, inak je zmazaný.

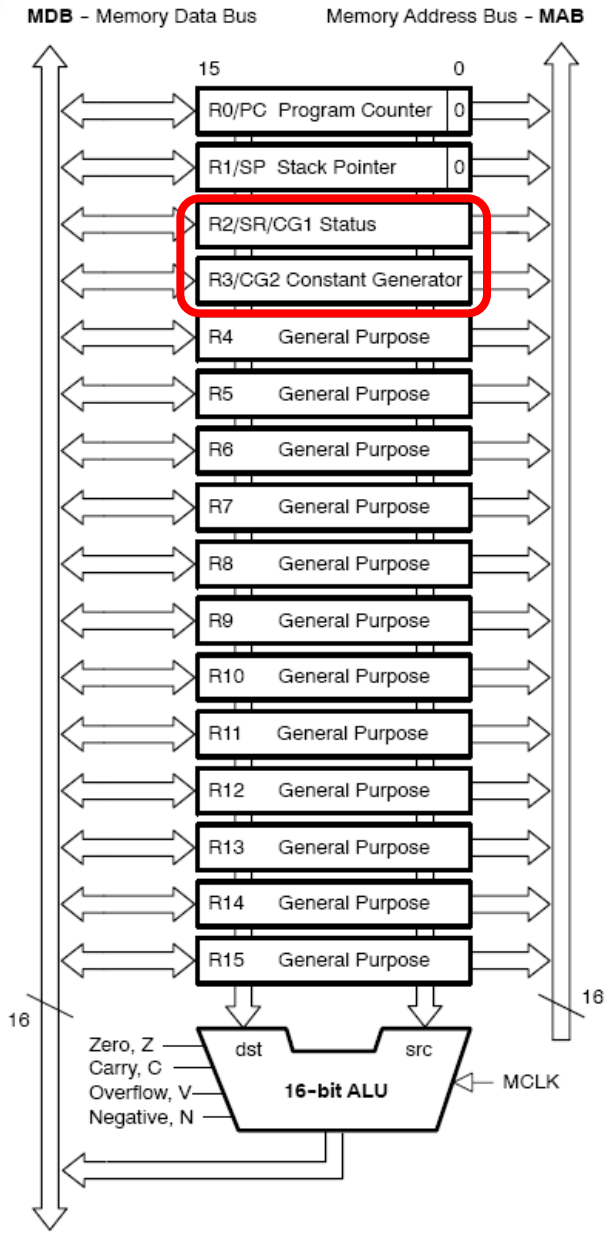
:: Registre CPU



Generátor konštánt (CG1/R2, CG2/R3)

- umožňuje automatické generovanie šiestich základných konštánt (-1, 0, 1, 2, 4, 8)
- nevyžaduje špeciálne inštrukcie
- nevyžaduje pre dané konštanty zdrojový kód navyše
- pre získanie konštanty nie je potrebný prístup do pamäte
- asembler pri preklade používa generátor konštánt automaticky akonáhle sa v zdrojovom kóde objaví niektorá zo zadefinovaných šiestich konštánt vo forme okamžitého operandu

:: Registre CPU



Generátor konštánt (CG1/R2, CG2/R3)

- inštrukčný súbor MSP430 obsahuje iba 27 inštrukcií (RISC)
- vďaka generátoru konštánt podporuje asembler MSP430 ďalších 24 emulovaných inštrukcií

Napr. inštrukcia s jedným operandom:

```
CLR     dst
```

je emulovaná inštrukciou s dvomi operandami s rovnakou dĺžkou:

```
MOV     R3,dst
```

pričom asembler pomocou generátora konštánt automaticky uloží do registra R3 hodnotu #0.
Inštrukcia

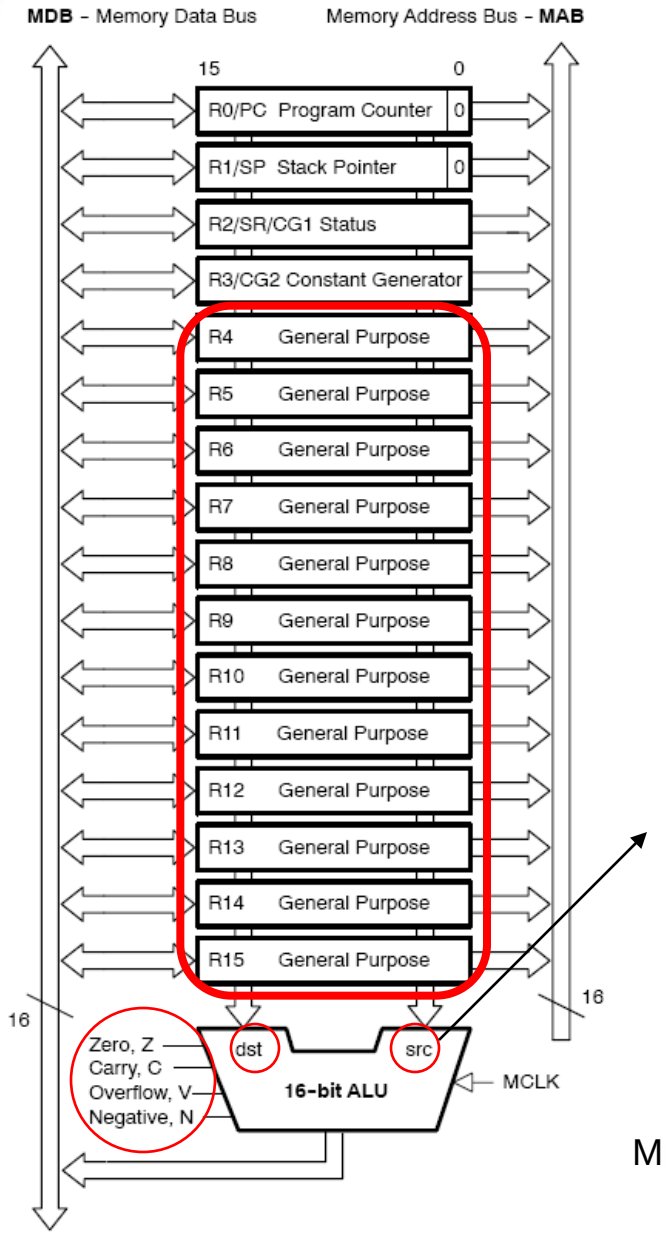
```
INC dst
```

je nahradená:

```
ADD 0(R3),dst
```

a pod.

:: Registre CPU



Pracovné registre R4 – R15 pre všeobecné použitie

- tieto registre je možné používať ako dátové registre, smerníky alebo indexové registre
- môžu k nim pristupovať bytovo aj slovne orientované inštrukcie

Carry - (nosiť, prenášať)

oVerflow - (pretiecť)

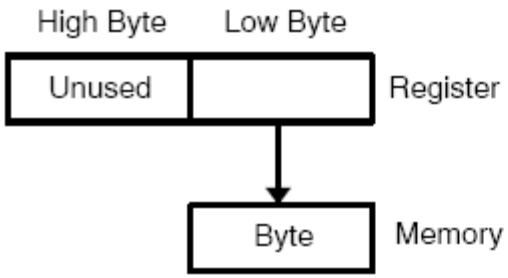
src - source (zdroj, odkiaľ)

dst - destination (cieľ cesty, kam)

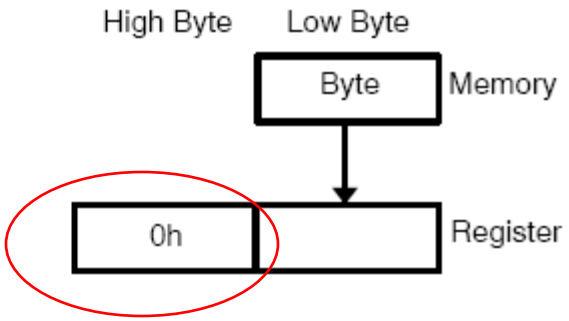
ALU - Arithmetic Logic Unit

:: Registrovo-bytové / bytovo-registrové operácie

Register-Byte Operation



Byte-Register Operation



Example Register-Byte Operation

R5 = 0A28Fh
R6 = 0203h
Mem(0203h) = 012h

```
ADD.B          R5,0(R6)
```

```
      08Fh
    + 012h
    ------
      0A1h
```

Mem (0203h) = 0A1h
C = 0, Z = 0, N = 1

```
  (Low byte of register)
+ (Addressed byte)
------------------------
->(Addressed byte)
```

Example Byte-Register Operation

R5 = 01202h
R6 = 0223h
Mem(0223h) = 05Fh

```
ADD.B          @R6,R5
```

```
      05Fh
    + 002h
    ------
    00061h
```

R5 = 00061h
C = 0, Z = 0, N = 0

```
  (Addressed byte)
+ (Low byte of register)
------------------------
->(Low byte of register, zero to High byte)
```

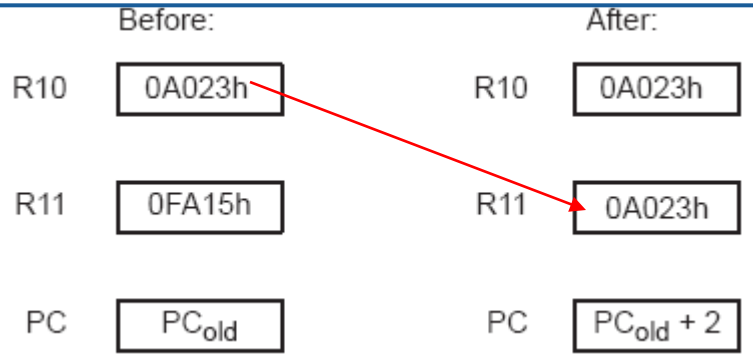
:: Adresovacie režimy

- asembler mikroradiča MSP430 umožňuje programátorovi využívať pri adresovaní celého adresného priestoru bez výnimky až sedem adresovacích režimov pre zdrojové operandy a štyri adresovacie režimy pre cieľové operandy
- v nižšie uvedenej tab. je stručný prehľad dostupných adresovacích režimov spolu s obsahom bitov As (source - zdroj) a Ad (destination - cieľ), ktoré definujú použitý adresovací režim

As/Ad	Adresovací režim	Syntax	Opis
00/0	Registrový režim	Rn	Operandami sú údaje použitých registrov.
01/1	Indexový režim	X(Rn)	Smerník (Rn + X) ukazuje na operand. Hodnota X je uložená v nasledujúcom slove.
01/1	Symbolický režim	ADDR	Smerník (PC + X) ukazuje na operand. Hodnota X je uložená v nasledujúcom slove. V podstate sa jedná o indexový režim X(PC).
01/1	Absolútny režim	&ADDR	Slovo nasledujúce za inštrukciou obsahuje absolútnu adresu. Hodnota X je uložená v nasledujúcom slove. V podstate sa jedná o indexový režim X(SR).
10/-	Nepriamy registrový režim	@Rn	Register Rn predstavuje smerník na operand.
11/-	Nepriama autoinkrementácia	@Rn+	Register Rn predstavuje smerník na operand. Smerník (údaj registra Rn) je po vykonaní inštrukcie inkrementovaný o 1 v prípade inštrukcie typu .B a o 2 v prípade inštrukcie typu .W
11/-	Režim okamžitého adresovania	#N	Slovo nasledujúce za inštrukciou obsahuje okamžitú konštantu. V podstate sa jedná o nepriamy autoinkrementačný režim @PC+.

:: Režim registrového adresovania

Zdrojový kód:	**MOV**	**R10,R11**
Obsah ROM:	**MOV**	**R10,R11**
Dĺžka:	Jedno alebo dve slová.	
Opis:	Presunie obsah R10 do R11, pričom obsah R10 nie je nijako ovplyvnený.	
Poznámka:	Platí pre zdrojový aj cieľový operand.	
Príklad:	**MOV**	**R10,R11**



Pozn.

K dátam v registroch môžeme pristupovať buď bytovo (.B) alebo slovne (.W) orientovanými inštrukciami. Ak používame .B inštrukciu, vyšší byte výsledku je vždy nula. Stavové bity v stavovom registri SR sú nastavované podľa výsledku .B inštrukcie.

:: Režim indexového adresovania (1)

Zdrojový kód: **`MOV 2(R5),6(R6)`**

Obsah ROM: **`MOV X(R5),Y(R6) ; X = 2, Y = 6`**

Dĺžka: Dve alebo tri slová.

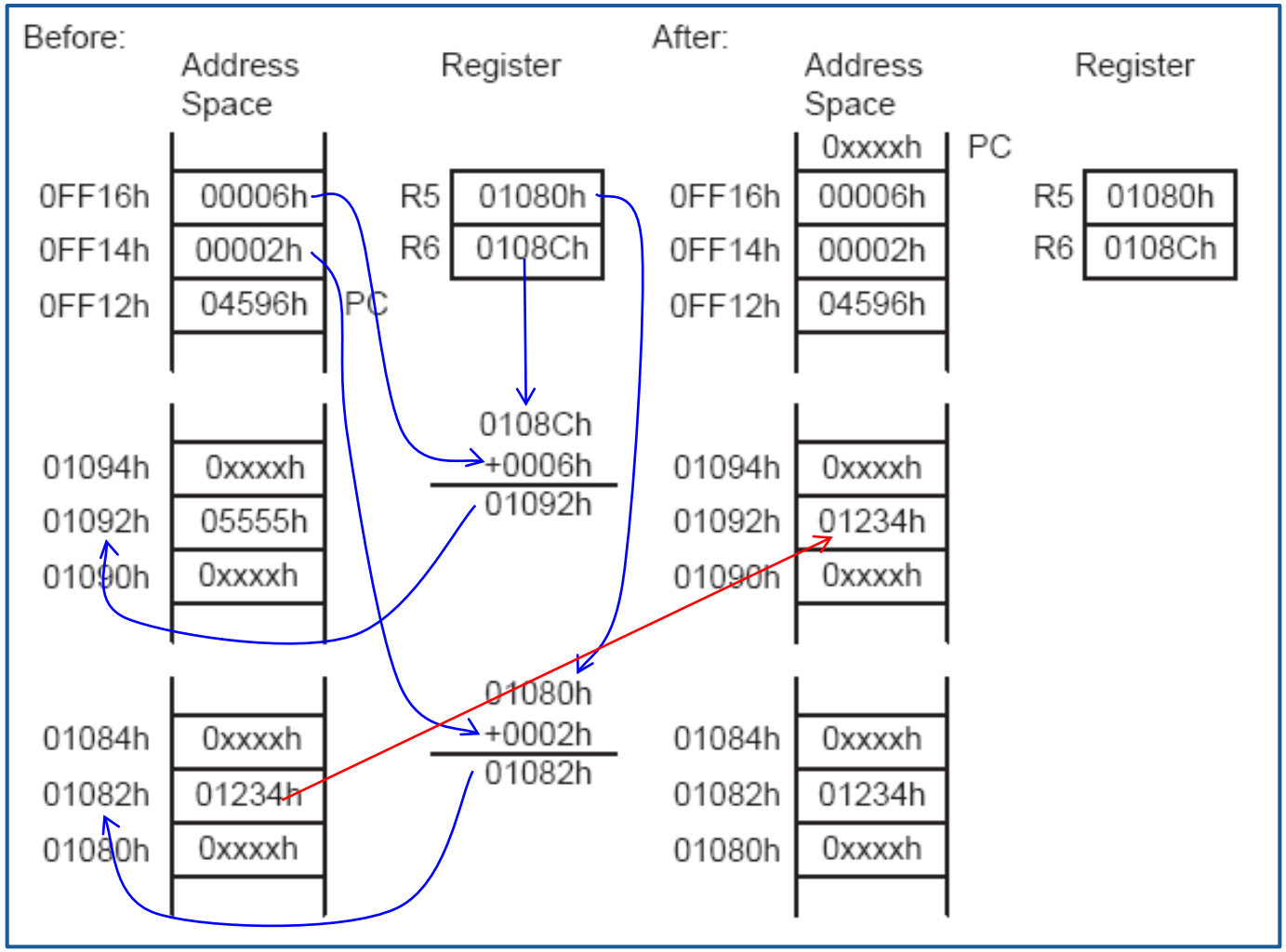
Opis: Presunie obsah zo zdrojovej adresy (danej súčtom R5 + 2) na cieľovú adresu (danú súčtom R6 + 6). Zdrojový a cieľový register nie sú ovplyvnené. V režime indexového adresovania je PC inkrementovaný automaticky, takže vykonávanie programu pokračuje ďalšou inštrukciou.

Poznámka: Platí pre zdrojový aj cieľový operand.

Príklad: **`MOV 2(R5),6(R6)`**

:: Režim indexového adresovania (2)

Príklad: **MOV 2(R5),6(R6)**



:: Režim symbolického adresovania (1)

Zdrojový kód: **MOV EDE,TONI**

Obsah ROM: **MOV X(PC),Y(PC) ; X = EDE - PC, Y = TONI - PC**

Dĺžka: Dve alebo tri slová.

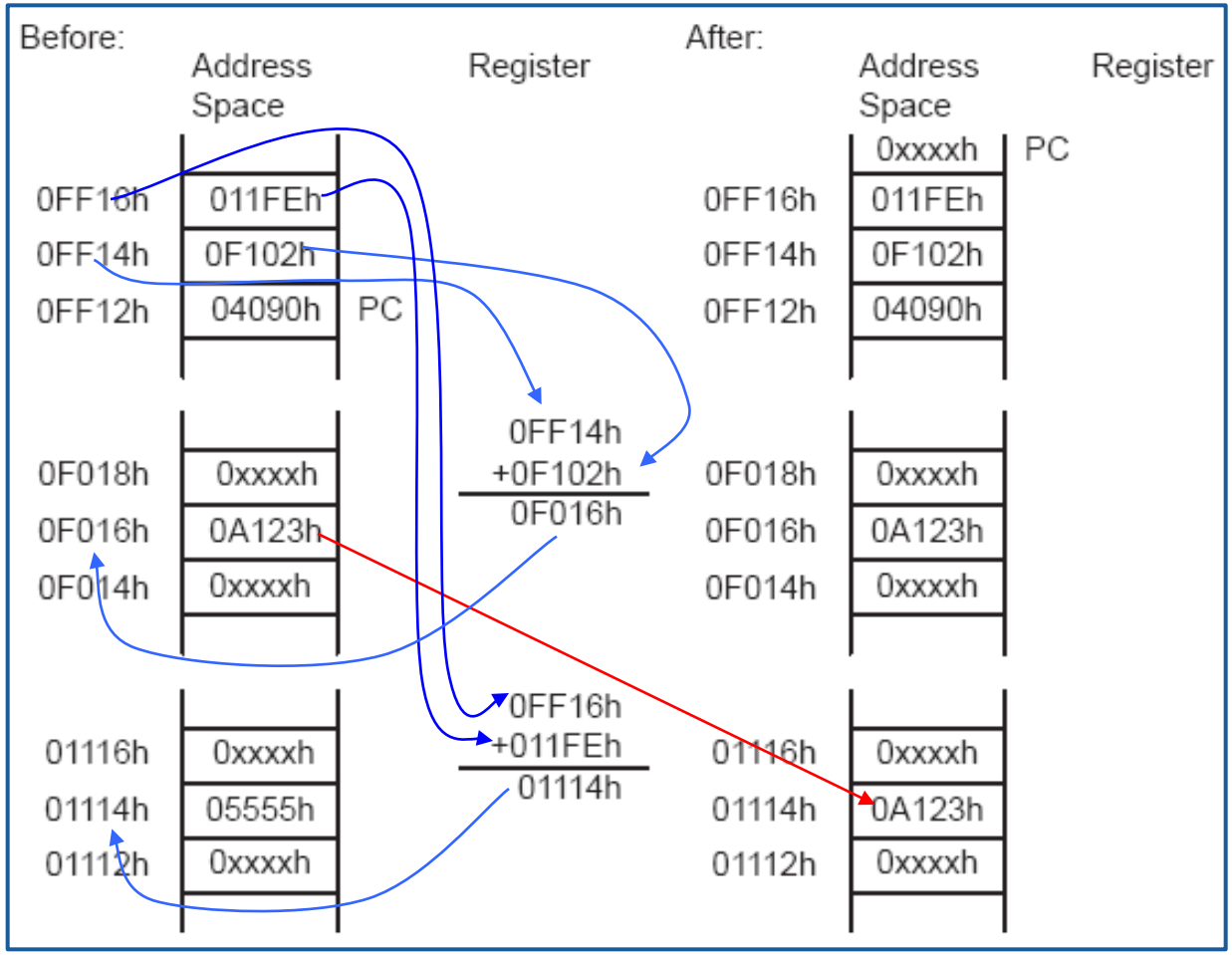
Opis: Presunie obsah zo zdrojovej adresy EDE (danej súčtom PC+X) na cieľovú adresu TONI (danú súčtom PC+Y). Slová nasledujúce za inštrukciou obsahujú rozdiel X medzi obsahom PC a zdrojovou adresou a rozdiel Y medzi obsahom PC a cieľovou adresou. Asembler vypočíta a použije ofsety X a Y automaticky. V režime symbolického adresovania je PC inkrementovaný automaticky, takže vykonávanie programu pokračuje ďalšou inštrukciou.

Poznámka: Platí pre zdrojový aj cieľový operand.

Príklad:

```
MOV     EDE,TONI        ;zdrojova adresa EDE = 0F016h
                        ;cielova adresa TONI = 01114h
```

Príklad: **MOV EDE,TONI** **;zdrojova adresa EDE = 0F016h**
;cielova adresa TONI = 01114h



:: Režim absolútneho adresovania (1)

Zdrojový kód: **MOV &EDE,&TONI**

Obsah ROM: **MOV X(0),Y(0) ; X=EDE, Y=TONI**

Dĺžka: Dve alebo tri slová.

Opis: Presunie obsah zo zdrojovej adresy EDE na cieľovú adresu TONI. Slová za inštrukciou obsahujú absolútne adresy zdroja aj cieľa. V režime absolútneho adresovania je PC inkrementovaný automaticky, takže vykonávanie programu pokračuje ďalšou inštrukciou.

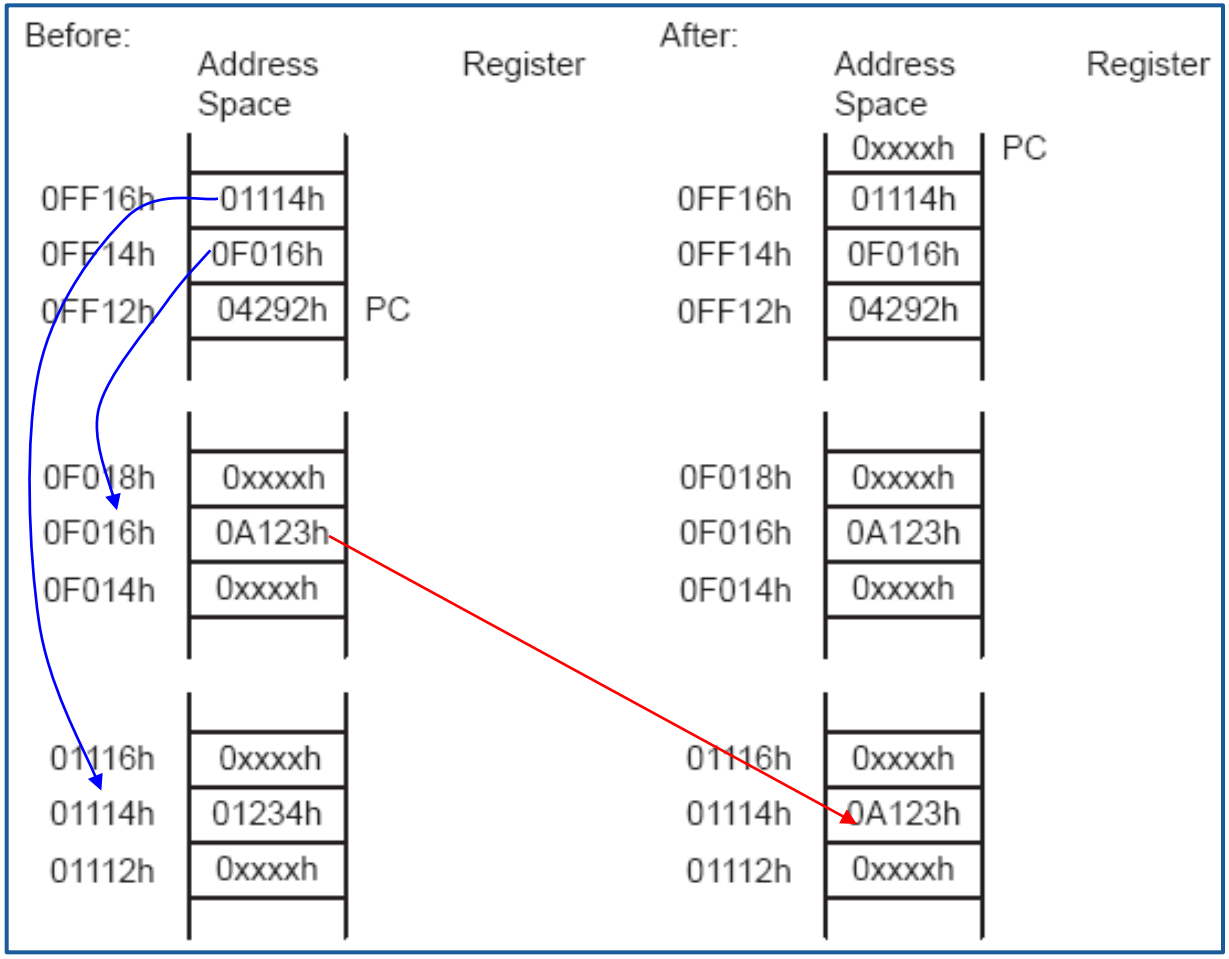
Poznámka: Platí pre zdrojový aj cieľový operand.

Príklad:

```
MOV     &EDE,&TONI      ;zdrojova adresa EDE = 0F016h
                        ;cielova adresa TONI = 01114h
```

:: Režim absolútneho adresovania (2)

Príklad: **MOV &EDE,&TONI ;zdrojova adresa EDE = 0F016h**
;cielova adresa TONI = 01114h



Pozn.:

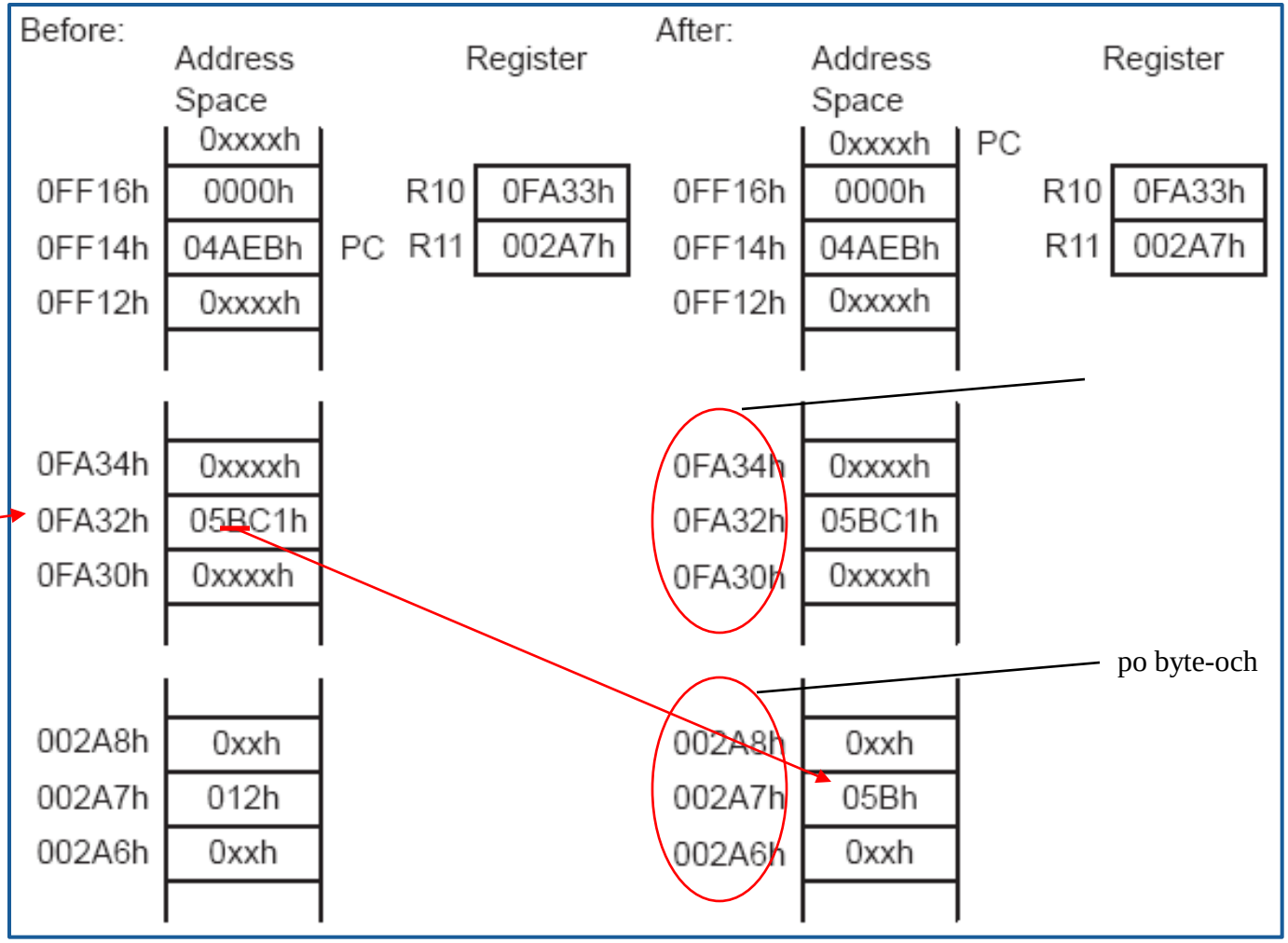
Režim absolútneho adresovania používame najmä pri adresovaní periférnych modulov mikroradiča, ktorých riadiace a dátové registre sú umiestnené na absolútnych (fixných) adresách.
Absolútne adresovanie periférnych modulov vychádza z potreby zabezpečenia prenositeľnosti zdrojového kódu (napr. aby bol kód nezávislý od umiestnenia v pamäti).

:: Režim nepriameho registrového adresovania (1)

Zdrojový kód:	**`MOV @R10,0(R11)`**
Obsah ROM:	**`MOV @R10,0(R11)`**
Dĺžka:	Jedno alebo dve slová.
Opis:	Presunie obsah zo zdrojovej adresy (uloženej v R10) na cieľovú adresu (uloženú v R11). Obsah registrov samotných nie je modifikovaný.
Poznámka:	Platí len pre zdrojový operand. Syntax cieľového operandu je 0(Rd).
Príklad:	**`MOV.B @R10,0(R11)`**

:: Režim nepriameho registrového adresovania (2)

Príklad: **`MOV.B @R10,0(R11)`**



5Bh je na adrese 0FA33h

0C1h je na adrese 0FA32h

po byte-och

:: Režim nepriameho autoinkrementačného adresovania (1)

Zdrojový kód: **`MOV @R10+,0(R11)`**

Obsah ROM: **`MOV @R10+,0(R11)`**

Dĺžka: Jedno alebo dve slová.

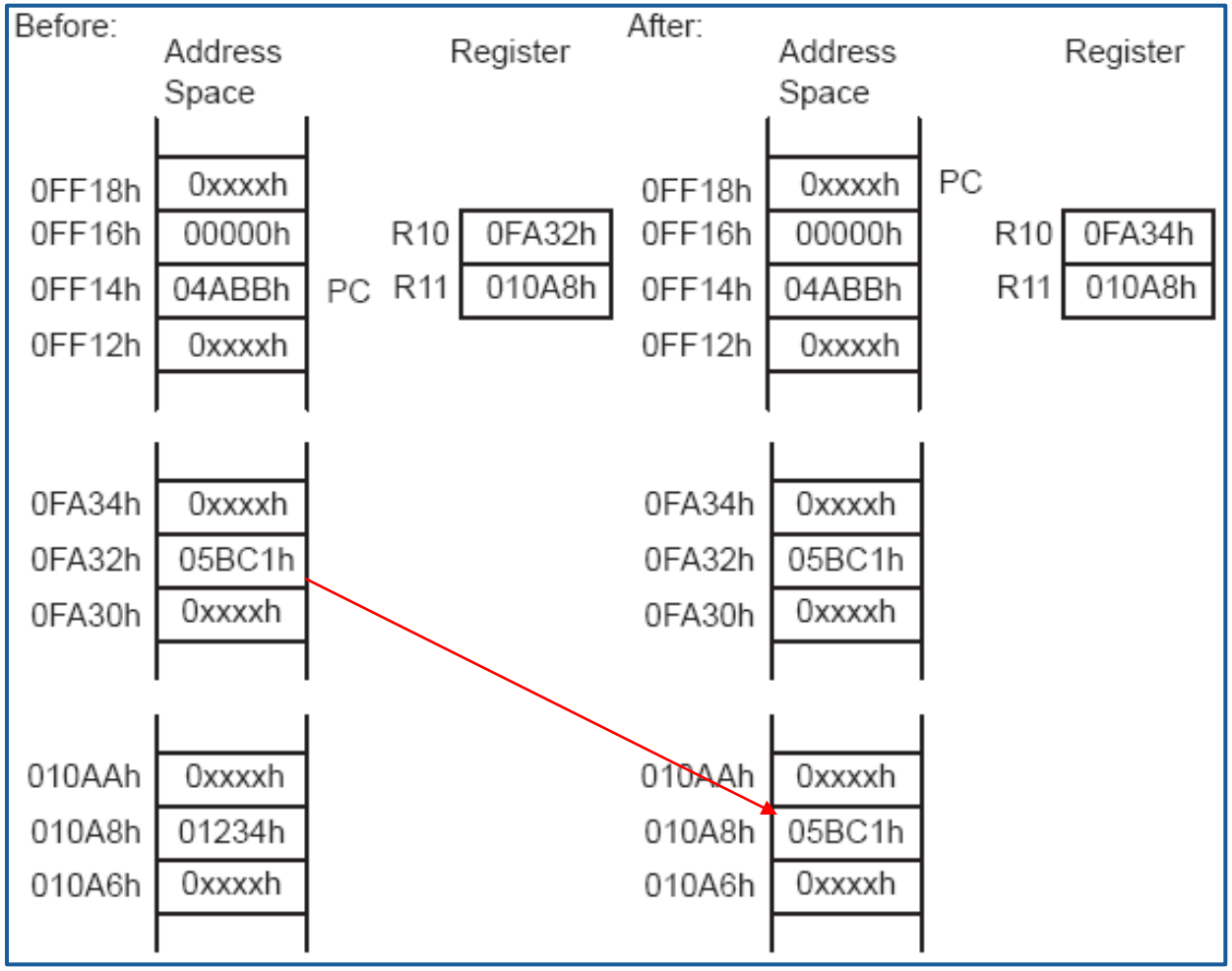
Opis: Presunie obsah zo zdrojovej adresy (uloženej v R10) na cieľovú adresu (uloženú v R11). Register R10 je inkrementovaný o 1 pri bytovej operácii a o 2 pri slovnej operácii ihneď po vykonaní inštrukcie, takže R10 ukazuje na ďalšiu adresu. Tento režim je výhodný najmä v prípade spracovávania tabuliek.

Poznámka: Platí len pre zdrojový operand. Syntax cieľového operandu je 0(Rd). Ak potrebujeme zároveň inkrementovať aj cieľovú adresu, môžeme použiť inštrukciu INCD Rd.

Príklad: **`MOV @R10+,0(R11)`**

:: Režim nepriameho autoinkrementačného adresovania (2)

Príklad: **MOV @R10+,0(R11)**



Pozn: Autoinkrementácia zdrojového registra nastane až po načítaní operandu.

:: Režim okamžitého adresovania (1)

Zdrojový kód: **MOV #45h,TONI**

Obsah ROM: **MOV @PC+,X(PC) ; 45, X = TONI - PC**

Dĺžka: Dve alebo tri slová. Má o jedno slovo menej, ak je možné použiť konštantu z CG1 alebo CG2.

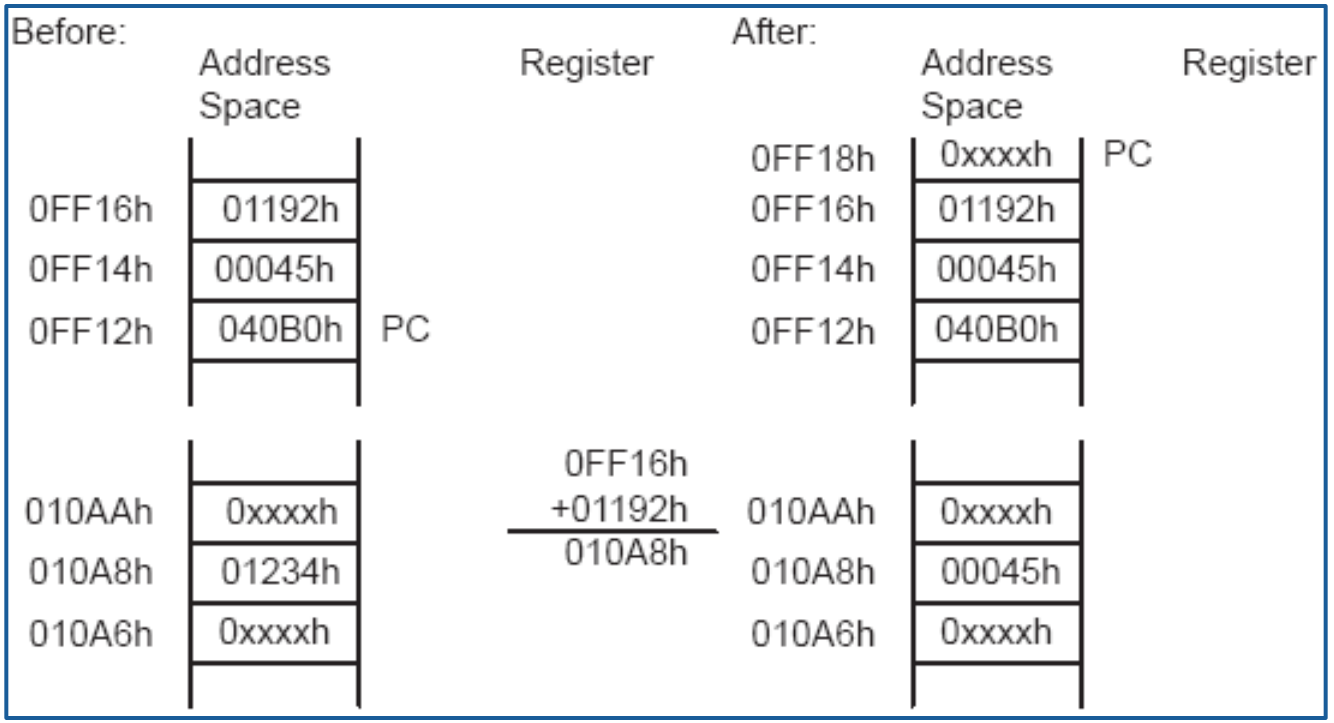
Opis: Presunie okamžitú konštantu 45h uloženú v slove, ktoré nasleduje za inštrukciou na cieľovú adresu TONI. Pri načítavaní zdroja ukazuje PC na slovo nasledujúce za inštrukciou a presunie obsah tohto slova na cieľovú adresu.

Poznámka: Platí len pre zdrojový operand.

Príklad: **MOV #45h,TONI**

:: Režim okamžitého adresovania (2)

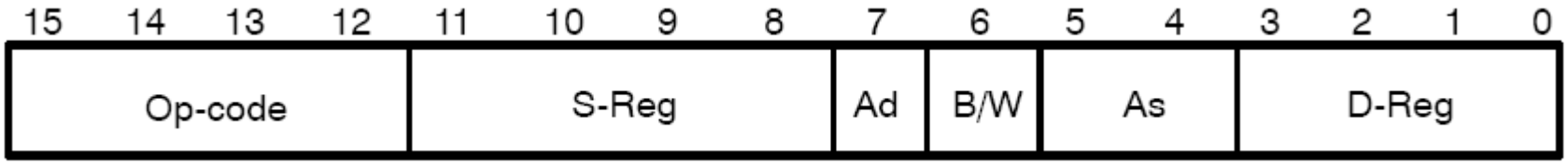
Príklad: **MOV #45h,TONI**



:: Inštrukčný súbor

- úplný inštrukčný súbor mikroradiča MSP430 obsahuje
  - *27 základných a*
  - *24 emulovaných inštrukcií*
- základné inštrukcie majú každá vlastný opkód
- emulované inštrukcie umožňujú zapisovať zdrojový kód jednoduchšie a prehľadnejšie a uľahčujú jeho čítanie, ale nemajú vlastný opkód
- z hľadiska zdrojového kódu ani jeho výkonu neexistujú pri používaní emulovaných inštrukcií žiadne obmedzenia
- z hľadiska formátu môžeme základné inštrukcie rozdeliť do troch kategórií:
  - *inštrukcie s dvoma operandami,*
  - *inštrukcie s jedným operandom,*
  - *inštrukcie skoku.*
- všetky inštrukcie s jedným alebo dvoma operandami môžu byť bytovo alebo slovne orientované (v zdrojovom kóde toto špecifikujeme príponou .B alebo .W)
- ak neuvedieme v zdrojovom kóde za inštrukciou žiadnu príponu, asembler explicitne predpokladá, že sa jedná o slovne orientovanú inštrukciu

Formát operačného kódu I.



Pozn.:

Zdrojový a cieľový operand inštrukcie definujeme nasledovnými poliami:

src - zdrojový (source) operand definovaný bitmi As a zdrojovým registrom S-reg,

dst - cieľový (destination) operand definovaný bitmi Ad a cieľovým registrom D-reg,

As - adresovacie bity definujúce adresovací režim použitý pre zdroj (src),

S-reg - pracovný register použitý pre zdroj (src),

Ad - adresovacie bity definujúce adresovací režim použitý pre cieľ (dst),

D-reg - pracovný register použitý pre cieľ (dst),

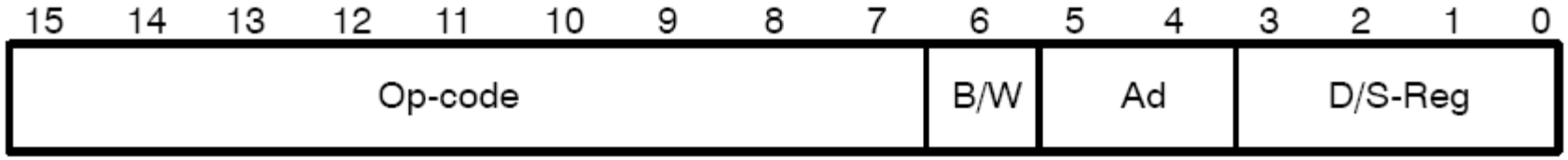
B/W - bytová alebo slovná operácia; 0: slovná operácia, 1: bytová operácia.

:: Inštrukcie s dvoma operandami

Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
MOV(.B)	src,dst	src → dst	–	–	–	–
ADD(.B)	src,dst	src + dst → dst	*	*	*	*
ADDC(.B)	src,dst	src + dst + C → dst	*	*	*	*
SUB(.B)	src,dst	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src,dst	dst + .not.src + C → dst	*	*	*	*
CMP(.B)	src,dst	dst - src	*	*	*	*
DADD(.B)	src,dst	src + dst + C → dst (decimally)	*	*	*	*
BIT(.B)	src,dst	src .and. dst	0	*	*	*
BIC(.B)	src,dst	.not.src .and. dst → dst	–	–	–	–
BIS(.B)	src,dst	src .or. dst → dst	–	–	–	–
XOR(.B)	src,dst	src .xor. dst → dst	*	*	*	*
AND(.B)	src,dst	src .and. dst → dst	0	*	*	*

*	The status bit is affected
–	The status bit is not affected
0	The status bit is cleared
1	The status bit is set

Formát operačného kódu II.



Pozn.:

Zdrojový a cieľový operand inštrukcie definujeme nasledovnými poliami:

src - zdrojový (source) operand definovaný bitmi As a zdrojovým registrom S-reg,

dst - cieľový (destination) operand definovaný bitmi Ad a cieľovým registrom D-reg,

As - adresovacie bity definujúce adresovací režim použitý pre zdroj (src),

S-reg - pracovný register použitý pre zdroj (src),

Ad - adresovacie bity definujúce adresovací režim použitý pre cieľ (dst),

D-reg - pracovný register použitý pre cieľ (dst),

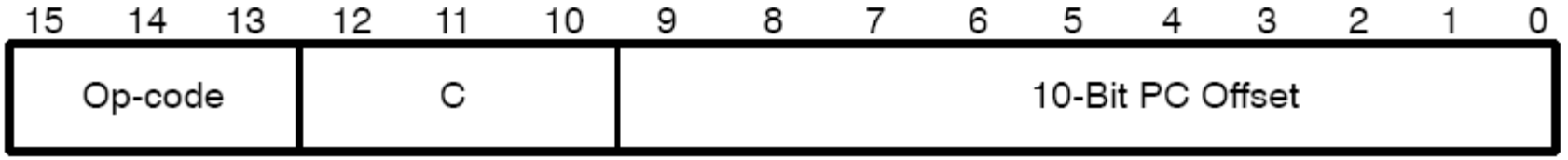
B/W - bytová alebo slovná operácia; 0: slovná operácia, 1: bytová operácia.

:: Inštrukcie s jedným operandom

Mnemonic	S-Reg, D-Reg	Operation	Status Bits V	N	Z	C
RRC(.B)	dst	C → MSB →.....LSB → C	*	*	*	*
RRA(.B)	dst	MSB → MSB →....LSB → C	0	*	*	*
PUSH(.B)	src	SP - 2 → SP, src → @SP	-	-	-	-
SWPB	dst	Swap bytes	-	-	-	-
CALL	dst	SP - 2 → SP, PC+2 → @SP dst → PC	-	-	-	-
RETI		TOS → SR, SP + 2 → SP TOS → PC,SP + 2 → SP	*	*	*	*
SXT	dst	Bit 7 → Bit 8.....Bit 15	0	*	*	*

*	The status bit is affected
-	The status bit is not affected
0	The status bit is cleared
1	The status bit is set

Formát inštrukcií skokov



Pozn.:

C - podmienka (condition)

:: Inštrukcie skokov

Mnemonic	S-Reg, D-Reg	Operation
JEQ/JZ	Label	Jump to label if zero bit is set
JNE/JNZ	Label	Jump to label if zero bit is reset
JC	Label	Jump to label if carry bit is set
JNC	Label	Jump to label if carry bit is reset
JN	Label	Jump to label if negative bit is set
JGE	Label	Jump to label if (N .XOR. V) = 0
JL	Label	Jump to label if (N .XOR. V) = 1
JMP	Label	Jump to label unconditionally

:: Otázky ku skúške

1. Nakreslite zjednodušenú architektúru procesora MSP430!
2. Uveďte základné rozdiely medzi von Neumannovou a harvardskou architektúrou! Aký typ architektúry predstavuje jadro MSP430?
3. Stručne opíšte pamäťovú mapu procesora MSP430!
4. Aký je rozdiel medzi RISC a CISC inštrukčnou sadou? Akým typom inštrukčnej sady disponuje procesor MSP430?
5. Čo znamená pojem ortogonálna architektúra?
6. Stručne opíšte CPU MSP430!
7. Opíšte špeciálne registre CPU R0 – R3!
8. Vysvetlite pojmy základná a emulovaná inštrukcia a opkód inštrukcie! Ako pracuje generátor konštánt?
9. Vysvetlite pojem adresovací režim! Vymenujte všetky adresovacie režimy procesora MSP430! K trom vybraným adresovacím režimom uveďte príklad!
10. Uveďte tri formáty základných inštrukcií procesora MSP430!

Koniec prednášky č. 1

Centrálna procesorová jednotka – CPU,

pamäťový model, režimy adresovania, inštrukčný súbor