

PŘÍSTUP K JEDNOTLIVÝM VÝVODŮM PROCESORU

Přístup k jednotlivým vývodům procesoru závisí na:

- ❖ Architektuře procesoru
- ❖ Na způsobu připojení bran.

První procesory měli brány umístěné ve **vstupně/výstupním** (I/O) prostoru. Byl-li vyžadován rychlejší přístup, pak byly mapovány do prostoru **datové paměti** (paměťově-mapované). Vstupy i výstupy byly dostupné po bytech nebo slovech. Přístup k bitům jednotlivých bytů řešil u starých a starších procesorů paměťové nároky pro proměnné typu indikátor. Změna jednoho bitu (nastavení/nulování) spočívalo v realizaci operací čtení, modifikace (logický součin nebo součet s konstantou) a zápis.

S nástupem jednočipových procesorů se objevila možnost ovlivňovat jednotlivé vývody samostatně, stejně jako možnost používat v programu bitové proměnné.

PŘÍSTUP K JEDNOTLIVÝM VÝVODŮM PROCESORU ŘADY 8051

V případě procesoru z řady 8051 mohou být jednotlivé vývody označeny symbolickým názvem, který vystihuje jeho funkci například

```
sbit LED = P1^4; // Označení vývodu P1.4 brány P1 v jazyce C. V
                // programu se pak používá zápis LED=0; nebo
                // LED=1. Vývod je dostupný přímou adresou
                // vývodu.
```

V jazyce symbolických adres (assembleru) bude definice čtvrtého vývodu na bráně P3

```
MOTOR set P3.3 ; Označení vývodu P3.3 brány
```

V programu pro vynulování vývodu P3.3 použijeme instrukci

```
CLR MOTOR nebo CLR P3.3 ; Překladač označení vývodu
                          ; nahradí konkrétní adresou CLR 93h (třetí
                          ; možnost zápisu).
```

```
SETB MOTOR ; Nastaví vývod P3.3 na log.1. Ani jedna z
            uvedených variant nebrání použít instrukce (čtení-modifikace-zápis)
```

```
ORL P3, #08h.
```

PŘÍSTUP K JEDNOTLIVÝM VÝVODŮM PROCESORU AVR

V případě procesorů AVR s menším počtem vývodů (<64) je situace obdobná jako u procesoru 8051.

U procesorů s větším počtem vývodů jsou nižší brány PA, PB, PC, PD, PE dostupné **bitově** a vyšší brány (záleží na procesoru) jako PF, PG, atd. jsou dostupné **bytově**.

Přístup k vývodům závisí na použitém vývojovém prostředí:

- ❖ Programově orientované prostředí např. GCC – přístup k bitům nepodporuje
- ❖ Hardwarově orientované prostředí např. CodeVision AVR, IAR podporují práci s bitovými proměnnými i jednotlivými bity bran. Například pro CodeVision AVR

```
#define MOSI   PORTB.2    // Výstupní signál sběrnice SPI,  
                        // vývod PB.2  
#define MISO   PINB.3    // Vstupní signál sběrnice SPI  
                        // vývod PB.3
```

V programu pak použijeme zápis **MOSI=0;** nebo **MOSI=1;**. Pro vstupní vývod například **if (MISO==1) hodnota |=1;**

PŘÍSTUP K JEDNOTLIVÝM VÝVODŮM PROCESORU AVR

V JSA můžeme k perifériím ležících ve spodní polovině I/O prostoru (adresy 0x20÷0x40), přistupovat instrukcemi přímého přístupu k bitům I/O registrů.

Například **SBI \$02,4 (SBI 0x02,4)**, kde SBI je nastavení bitu, 0x02(\$02) představuje adresu výstupního registru PA v I/O oblasti (v adresách celého datového prostoru 0x22) a 4 představuje pátý bit registru. Pro vynulování bitu slouží instrukce **CBI**.

Do celého datového prostoru (včetně souboru registrů R0÷R31 a registrům I/O prostoru) se dostaneme instrukcemi s přímým přístupem **LDS R2, \$adresa** (čtení do R2 z uvedené adresy). Pro zápis použijeme instrukci **STS \$adresa,R2** (zápis).

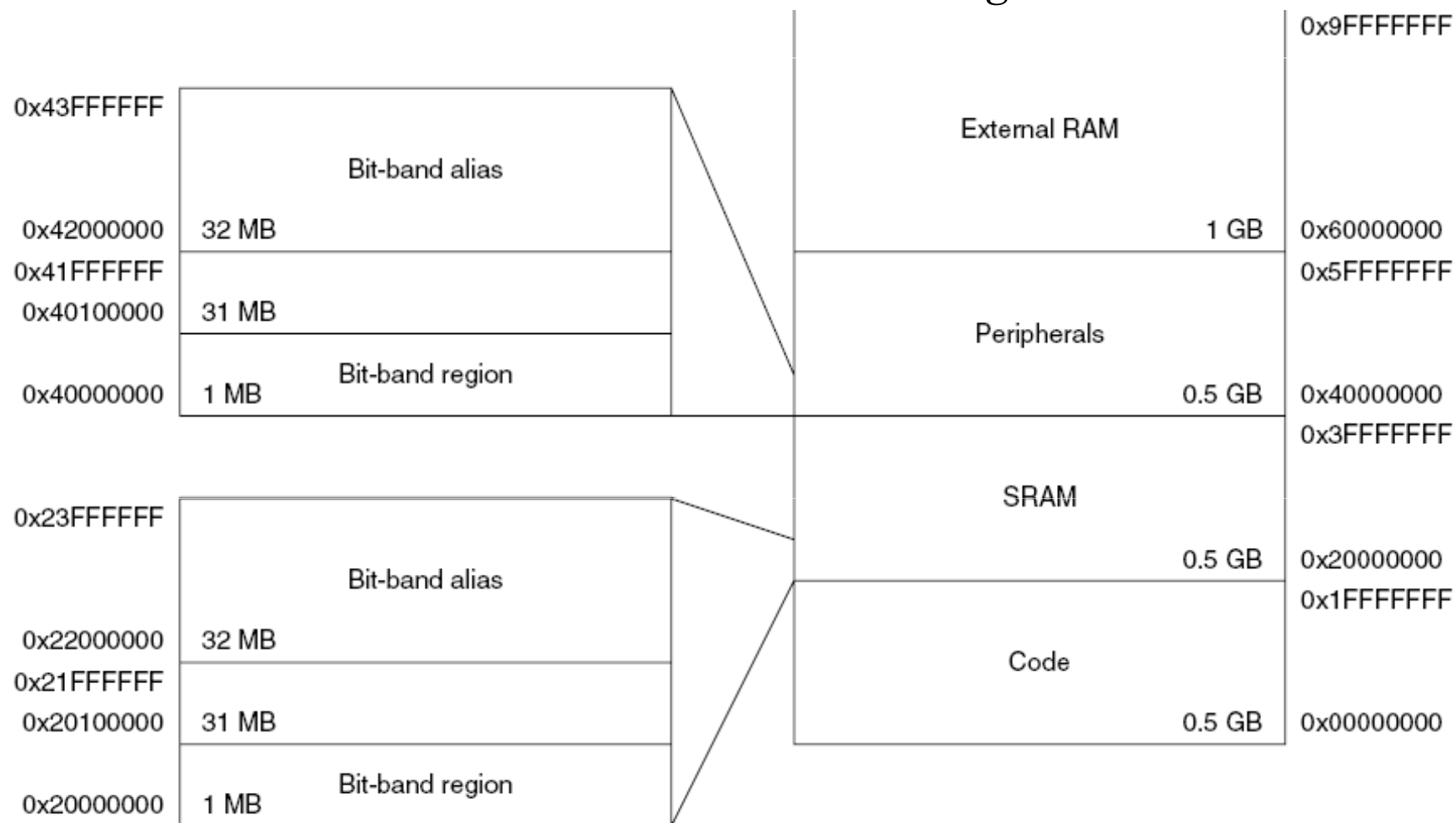
Pro nepřímý přístup pomocí registrů X (R27,R26), Y (R29,R28) a Z (R31,R30) obsahující adresu v datové paměti pomocí instrukcí

LD R3,Y(Y+) nebo **LDD R3,Y+2** (pro čtení).

ST Z(-Z),R5 nebo **STD Z+2, R5** (pro zápis).

PŘÍSTUP K BITŮM U PROCESORU ARM

U procesorů ARM byla modifikována myšlenka z procesoru 8048 (1976), kde jak vývody, tak i část datového prostoru je bitově dostupná (bit-band region, bit-band alias). V adresovém prostoru ARM jsou dvě oblasti o velikosti 1MB označované bit-band region viz. obrázek.



PŘÍSTUP K BITŮM U PROCESORU ARM

Jedna oblast je na začátku interní datové paměti a druhá na začátku oblasti prostoru periférií. Do těchto oblastí (bit-band region) můžeme přistupovat jak po 32 bitech, tak i po jednotlivých bitech. Přístup k jednotlivým bitům 32 bitových slov z oblasti (bit-band region) se realizuje operací s nejnižším bitem slova ve virtuální oblasti (bit-band alias). Vazba mezi jednotlivými adresami v případě oblasti datové paměti je zobrazena na obrázku na následující stránce.

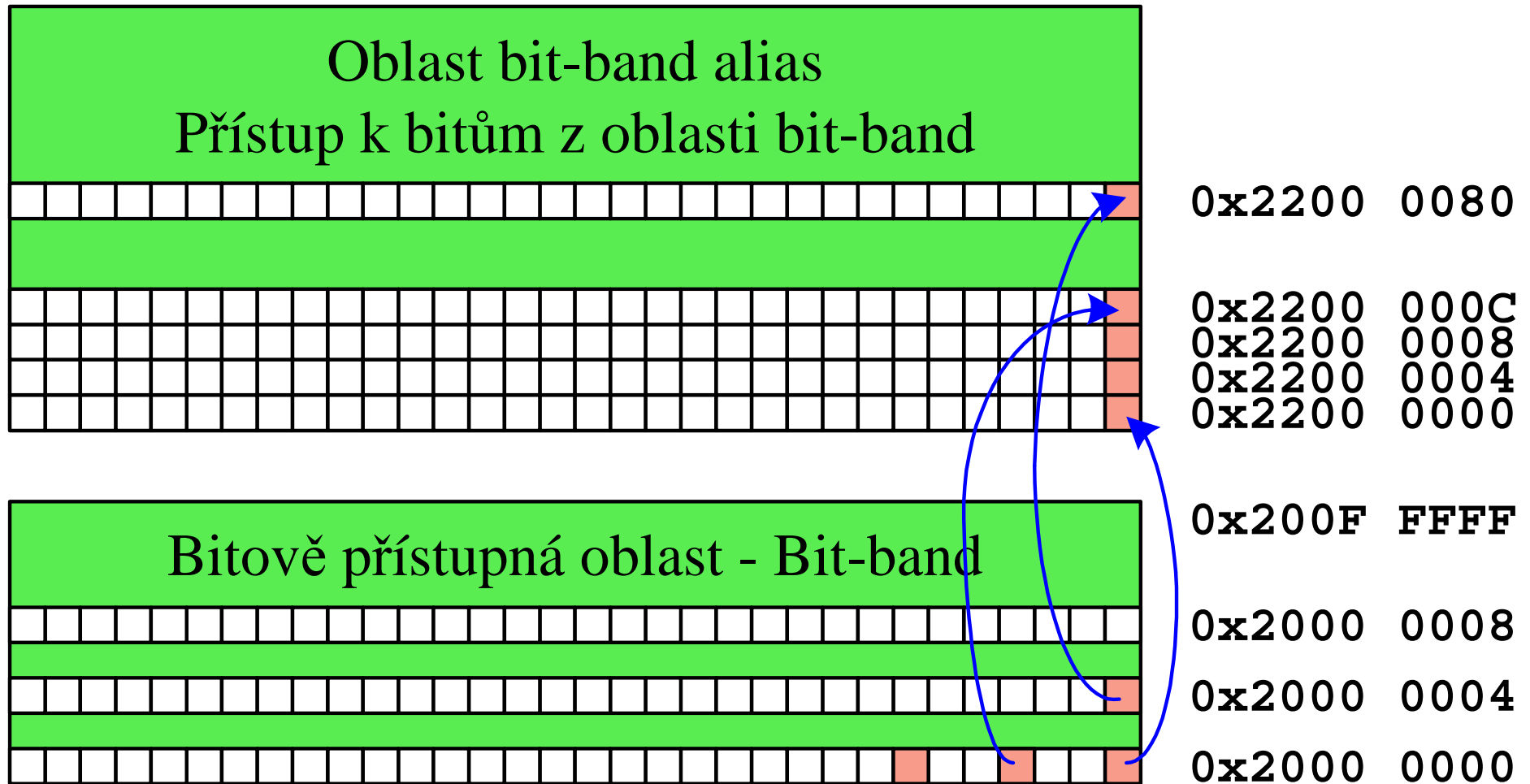
Adresu v oblasti bit-band alias (ABBA) datové paměti vypočítáme z adresy bit-band region (ABBR) a potřebného bitu (BIT) ze slova na adrese ABBR pomocí vztahu

$$ABBA=32*(ABBR-0x20000000)+0x22000000+4*BIT$$

Pro oblast periférií bude vztah upraven

$$ABBA=32*(ABBR-0x40000000)+0x42000000+4*BIT$$

HLAVNÍ RYSY PROCESORŮ ARM – BIT-BAND ALIAS ADDRESS



PŘÍSTUP K JEDNOTLIVÝM VÝVODŮM PROCESORU ARM

Bit-band alias usnadňuje změnu bitu bez operace AND a OR. K jednotlivým vývodům procesoru se v jazyce C můžeme dostat pomocí funkcí setbit, clearbit nebo zápisem

```
GPIOA->ODR|=(1ul << 5); // Nastavení 6-tého bitu na bráně GPIOA
```

Uvedený příkaz přeloží překladač následovně

```
0x0800075C 4845      LDR      r0,[pc,#276] ; @0x08000874
0x0800075E 6940      LDR      r0,[r0,#0x14]
0x08000760 F0400020  ORR     r0,r0,#0x20
0x08000764 4943      LDR      r1,[pc,#268] ; @0x08000874
0x08000766 6148      STR     r0,[r1,#0x14]
```

První instrukce načte za pomoci čítače instrukcí (PC) s posunem z uložené tabulky bázovou adresu brány GPIOA. Druhá přičte posun registru ODR od bázové adresy a třetí do něj přidá 1 na 6 bit. Pak je zbytečně opět načtena bázová adresa a upravená hodnota zapsána na místo s posunem. V zápisu programu v assembleru by mohla být adresa v první instrukci načtena do registru R1 a druhá instrukce by přečetla hodnotu do R0 za pomoci R1 s posunem.

PŘÍSTUP K JEDNOTLIVÝM VÝVODŮM PROCESORU ARM

Nastavení, nulování nebo změna bitu v paměti RAM by mohla být realizována 4 instrukcemi.

Bez využití bit-band

```
LDR R0,=0x20000000
LDR R1,[R0]
ORR R1,R1,#0x4 ; Třetí bit=1
STR R1,[R0] ; Zápis zpět
```

S využitím bit-band alias

```
LDR R0,=0x22000008
MOV R1,#1
STR R1,[R0]
```

UKÁZKA BITOVÉHO OVLÁDÁNÍ VÝVODU NA PROCESORU ARM

Realizace ovládání bitu periferie

```
#define motor value.i
#define BRANA_A 0x40020014

typedef struct {
    int i : 1;
    int j : 2;
    int k : 3;
} BB __attribute__((bitband));
BB value__attribute__((at(0x40020014)));
nebo
BB value__attribute__((at(BRANA_A))); // Placed object

void main()
{
    MOTOR=1;
konec: goto konec;
}
```

PŘÍSTUP K BITOVÝM PROMĚNNÝM PROCESORU 8051

Bitové proměnné pochází z doby procesoru 8048/8051, které měly malou kapacitou paměti a bylo potřeba nepoužívat k indikátoru celý byte. Bitové proměnné byly v části BDATA datové paměti (bitová oblast). Do oblasti byl možný přístup po bytech i bitech. Bit byl dostupný přímou adresou a jeho dostupnost se definovala:

```
bit stav_A, motor;           // Definice bitových proměnných
```

Nebylo možné přímo definovat pole bitů (bit indikátory[8];), kde potřebujeme přístup přes nepřímé adresování. Je ale možné definovat pole bitů jako strukturu pro práci s jednotlivými bity nebo skupinami bitů.

```
typedef struct {
    unsigned char b0 : 1; // Bitová proměnná
    unsigned char b1 : 2; // Dvoubitová proměnná
    unsigned char b3 : 5; // Pětibitová proměnná
} polebitu                // Označení struktury
polebitu value;           // value - obsahuje proměnné b0,b1,b3
                           // a je uložena v proměnné unsigned int
```

PŘÍSTUP K BITOVÝM PROMĚNNÝM PROCESORU 8051/AVR

Přístup k bitovým a více bitovým proměnným se realizuje přes **system masek** s využitím operace **AND**, **OR**, **EX-OR** a **rotace**. Pro práci s bity bylo efektivnější umístit proměnnou do oblasti BDATA a potom označit potřebné bity, s kterými potřebujeme manipulovat takto:

```
bdata unsigned char nastaveni; // Proměnná v bitové oblasti
sbit LD = nastaveni^6;         // Označení 7-bitu proměnné nastavení
                                // (z oblasti bytově i bitově přístupné)
sbit F0 = PSW ^5;              // Označení bitu bitově přístupného sfr
                                // (systémového funkčního registru) – vhodné pro individuální nastavení
                                // vlastností mikroprocesoru nebo bitu ovlivňujícího jeho periférii.
V jazyce C pak píšeme LD=0; F0=1; V assembleru SETB F0, CLR LD.
```

Ačkoliv procesory AVR umožňují používat část svých registrů R0 až R13 pro bitové proměnné, podporují tuto možnost jen vývojové prostředky od **obvodově orientovaných firem** (IAR, CodeVisionAVR).

Přístup k bitovým proměnným AVR realizujeme ve shodě s popisem pro 8051. Bitově a bytově dostupná oblast na AVR není.

Vývojová prostředí od programátorů (GCC) tuto výhodu **nepodporují**.

PŘÍSTUP K BITOVÝM PROMĚNNÝM PROCESORU ARM

V prostředí KEIL můžeme přístup k bitovým a více bitovým proměnným realizovat polem bitů. Přístup se pak realizuje systémem masek.

```
typedef struct {          // Pole bitů
    int b0 : 1;
    int b12 : 2;
    int b3 : 1;
} BB;
BB value;                // O umístění value (int) rozhoduje linker
void update_value(void)
{ value.b0 = 1;
  value.b12 = 1; }
```

Pro využití principu bit-band region/alias, pak musíme k definici struktury přidat informaci o využití bit-banding takto:

```
BB __attribute__((bitband));
```

Přístup k jednobitovým proměnným se pak bude realizovat přes adresování oblasti bit-band alias s menším počtem instrukcí.

Realizace ovládání bitového indikátoru

```
typedef struct {
    int i : 1;
    int j : 2;
    int k : 3;
} BB __attribute__((bitband));
BB value; // Unplaced object

#define MOTOR value.i

void main()
{
    MOTOR=1;
konec: goto konec;
}
```