Прекъсвания и директен достъп до паметта

Автор: гл. ас. д-р инж. Любомир Богданов

Европейски съюз

ПРОЕКТ BG051PO001--4.3.04-0042
„Организационна и технологична инфраструктура за учене през целия живот и развитие на компетенции”
Проектът се осъществява с финансовата подкрепа на
Оперативна програма „Развитие на човешките ресурси”,
съфинансирана от Европейския социален фонд на Европейския съюз
Инвестира във вашето бъдеще!

ЕСФ България
Европейски социален фонд

Съдържание

Обслужване на прекъсвания

Прекъсване (interrupt) е процес, при който микропроцесорът спира изпълнението на главната програма и започва да изпълнява кода на друга програма, в следствие на хардуерно събитие.

Това събитие може да е **синхронно** или **асинхронно** на изпълнението на главната програма.

С помощта на прекъсванията се премахва нуждата от **постоянна проверка** дали дадено събитие е настъпило (метод "**polling**"), което е **излишно изразходване на изчислителен ресурс**.

Обслужване на прекъсвания

Хендлер на прекъсване (interrupt handler) - допълнителна програма, различна от основната main, която се изпълнява вследствие на прекъсване. Нейната цел е да обслужи прекъсването, т.е. да се извършат дейности в отговор на постъпилото прекъсване.

Вектор на прекъсване (interrupt vector) - адресът от паметта, на който се намира кодът, изпълняван при настъпило прекъсване. От софтуерна гледна точка това е указател към функцията (хендлерът), която се извиква при настъпило събитие.

Обслужване на прекъсвания

Векторна таблица (interrupt vector table) - масив от указатели към функции, който се използва за обслужване прекъсванията. Всеки елемент от този масив е вектор на прекъсване и при настъпване на събитие се извлича един от много хендлери на прекъсване по хардуерен път.

Приоритет на прекъсване (interrupt priority) – при настъпването на две или повече събития се налага приоритизиране на извикването на хендлерите, защото микропроцесорът може да изпълнява само една програма в даден момент. Прекъсването с по-висок приоритет ще се обслужи преди прекъсването с по-нисък приоритет.

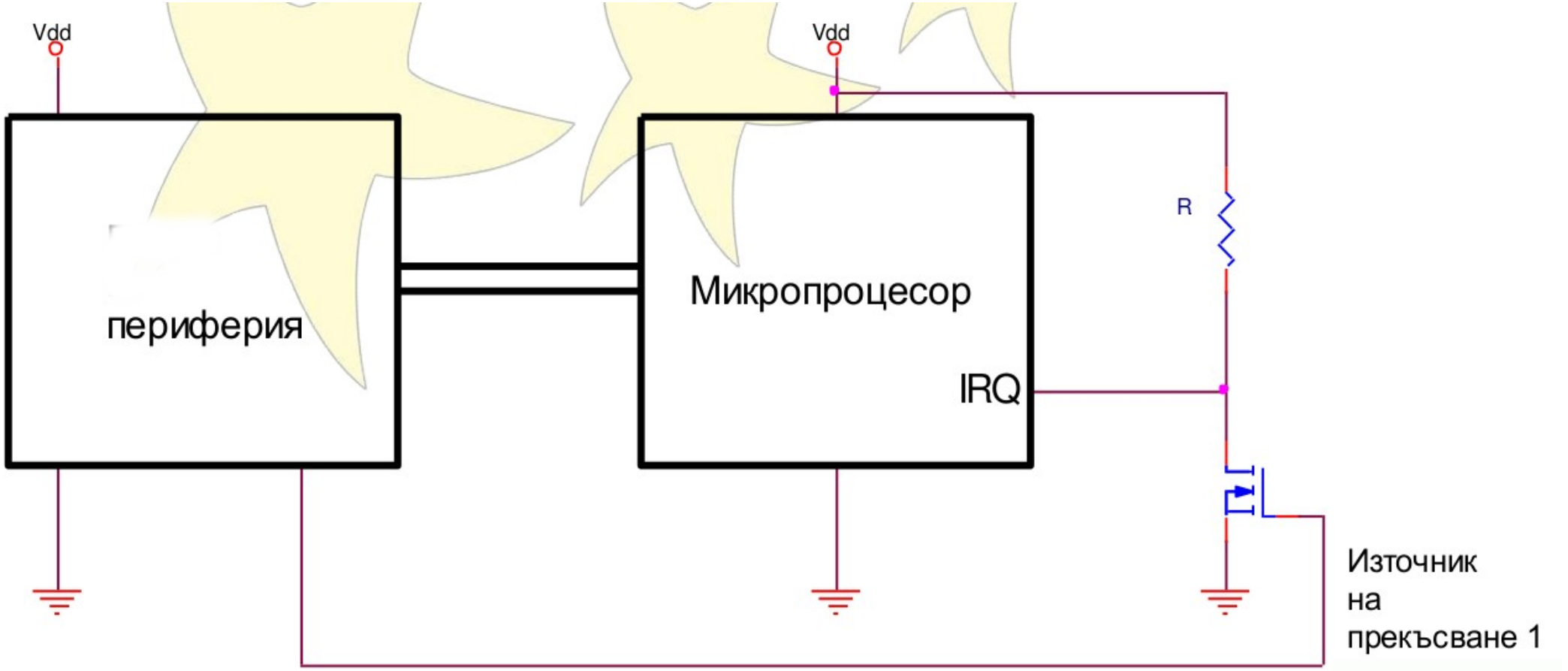
Обслужване на прекъсвания

На фигурата на следващия слайд е демонстриран пример за реализация на прекъсване от един източник. Както се вижда микропроцесорът превключва от изпълнението на главната програма в изпълнение на хендлера на прекъсване. Източник на сигнал IRQ може да е периферно устройство, което след извършване на дадена функция да сигнализира на микропроцесора за събитието чрез прекъсване.

Броят на IRQ входовете се определя от броя на микропроцесорните ядра. В многоядрените системи софтуерът трябва да определи кое прекъсване от кое ядро да се обслужи (в частност – това се прави от операционната система).

Забележка: суперскаларен не означава многоядрен!

Обслужване на прекъсвания

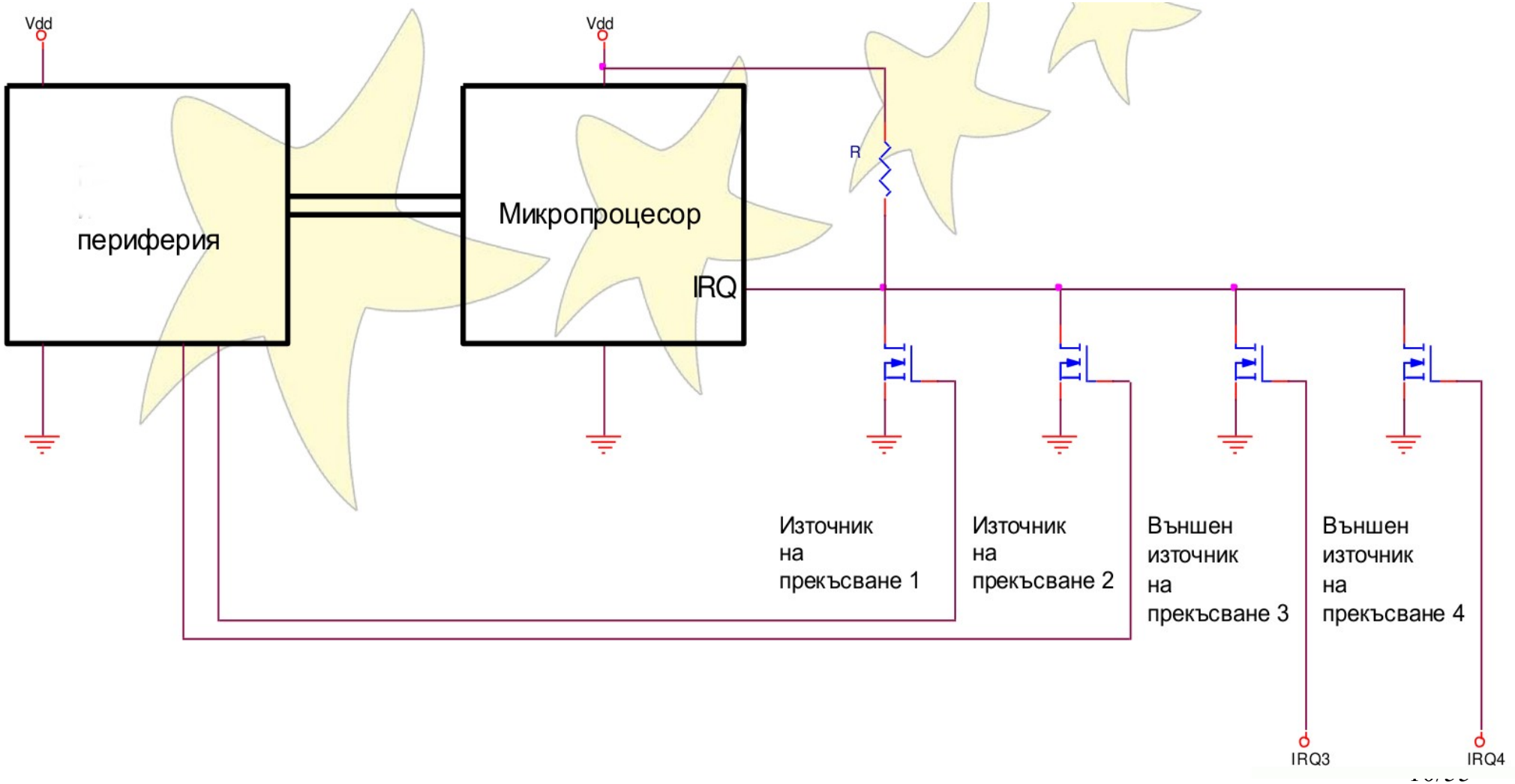


Обслужване на прекъсвания

В практиката по-често срещания вариант е получаване на прекъсване от два или повече източника, както е показано на фигурата. Транзисторите образуват логическата функция **жично ИЛИ**.

В този случай микропроцесорът се нуждае от допълнителен механизъм за **разпознаване на източника**.

Обслужване на прекъсвания



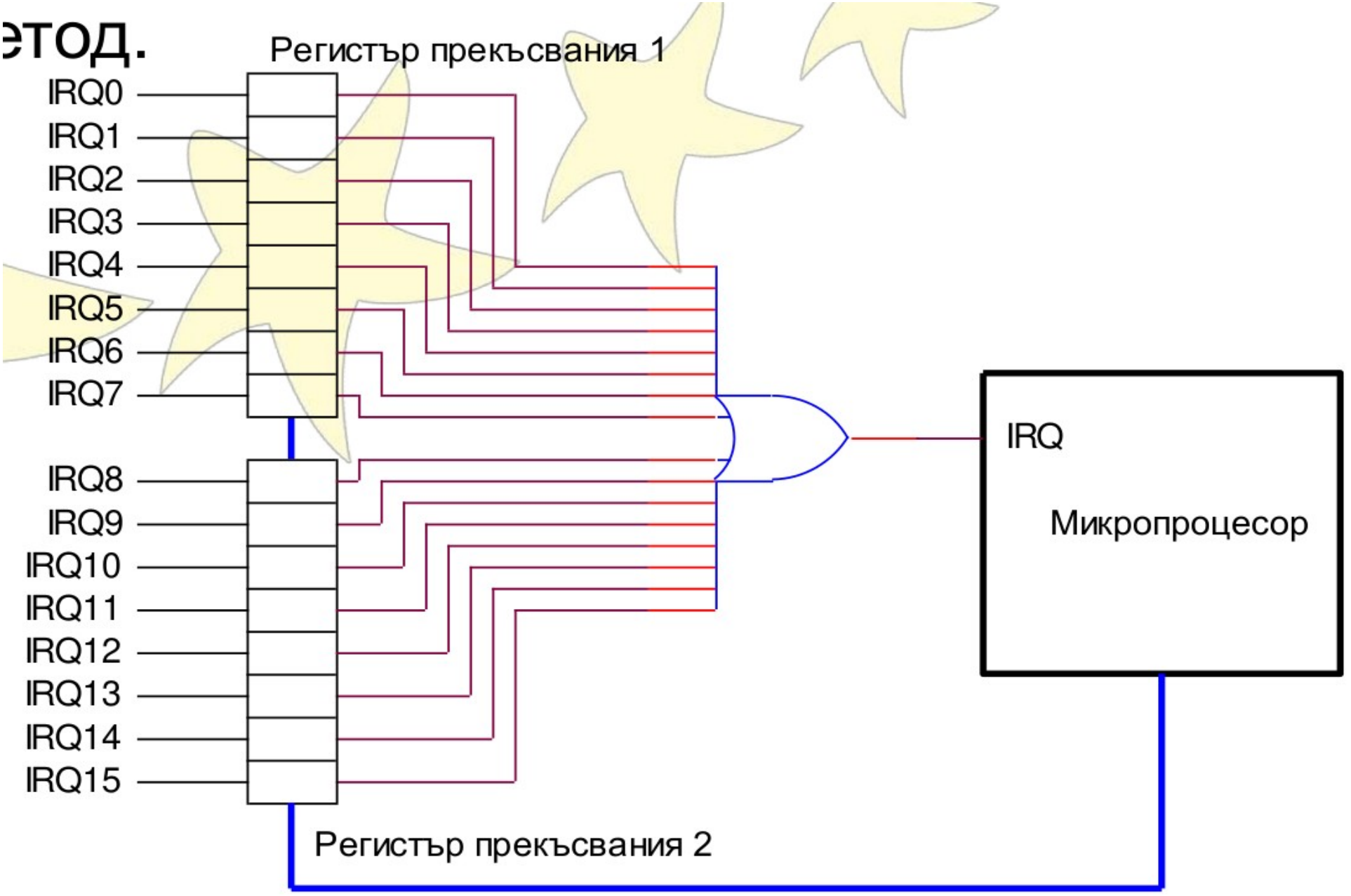
Обслужване на прекъсвания

За да може микропроцесорът да разбере от кой точно източник е получена заявката за прекъсване, **сигналите се буферират в един или няколко регистъра**, които са **част от адресното поле**.

След получаване на сигнал за прекъсване микропроцесорът прочита тези регистри и, знаейки тяхната предишна стойност, определя източника.

На фигурата на следващия слайд е демонстриран този метод.

Обслужване на прекъсвания



Обслужване на прекъсвания

Всеки бит от тези буферни регистри се нарича **флаг на прекъсване**.

Флаговете на прекъсване са винаги активни и отразяват **текущото състояние** на събитията в периферията.

Дали определени събития ще се подадат на входа за прекъсване на μPU зависи от още една група битове, поместени в друг регистър, който служи за разрешаване на прекъсванията.

Обслужване на прекъсвания

Почти винаги е вярно твърдението:
*На всеки флаг за прекъсване съответства един бит за разрешаване на прекъсването. Изключение правят важните прекъсвания, които са винаги разрешени.
(виж по-следващия слайд и немаскируеми прекъсвания)

При обслужване на прекъсването флаговете, които са го предизвикали, **трябва да бъдат нулирани**. В противен случай отново ще се генерира прекъсване и фърмуера ще увисне.

Някои μCU имат периферия, която сама чисти прекъсванията си при прочитане на статус регистъра за прекъсванията.

Обслужване на прекъсвания

При влизане в прекъсване програмата трябва да извърши следните операции:

***трябва да се забранят всички прекъсвания временно;**
*трябва да прочете статус регистъра на прекъсванията, за да разбере точно от кой източник е дошло прекъсването;
*трябва да нулира флага на прекъсването;
*трябва да изпълни действие в отговор на това прекъсване;
***трябва да се разрешат всички прекъсвания отново;**

Първата и последната операция се препоръчват само в критични части от кода, които не трябва да бъдат повлияни от други прекъсвания с по-голям приоритет. Този отрязък от код трябва да е възможно най-кратък или системата ще стане бавнореагираща.

Обслужване на прекъсвания

Пример – обслужване на прекъсвания в TM4C1294, UART модул.

```
void UARTIntHandler(void){
   uint32_t status;
   char ch;

   //Get the interrrupt status.
   status = MAP_UARTIntStatus(UART0_BASE, true);

   //Clear the asserted interrupts.
   MAP_UARTIntClear(UART0_BASE, status);

   //Echo the characters in the receive FIFO.
   while(MAP_UARTCharsAvail(UART0_BASE)){
      ch =MAP_UARTCharGetNonBlocking(UART0_BASE);
      MAP_UARTCharPutNonBlocking(UART0_BASE, ch);
   }
}
```

Обслужване на прекъсвания

Прекъсванията могат да се разделят на два вида според възможността да бъдат контролирани – маскируеми и немаскируеми.

Маскируеми прекъсвания – прекъсвания, които могат да бъдат забранявани. Ако дадено прекъсване е забранено, при появата на сигнал от източника няма да се подаде сигнал към микропроцесора.

Пример – приети данни в преместващия регистър на UART модула.

Пример – преобразуването с АЦП е завършило.

Обслужване на прекъсвания

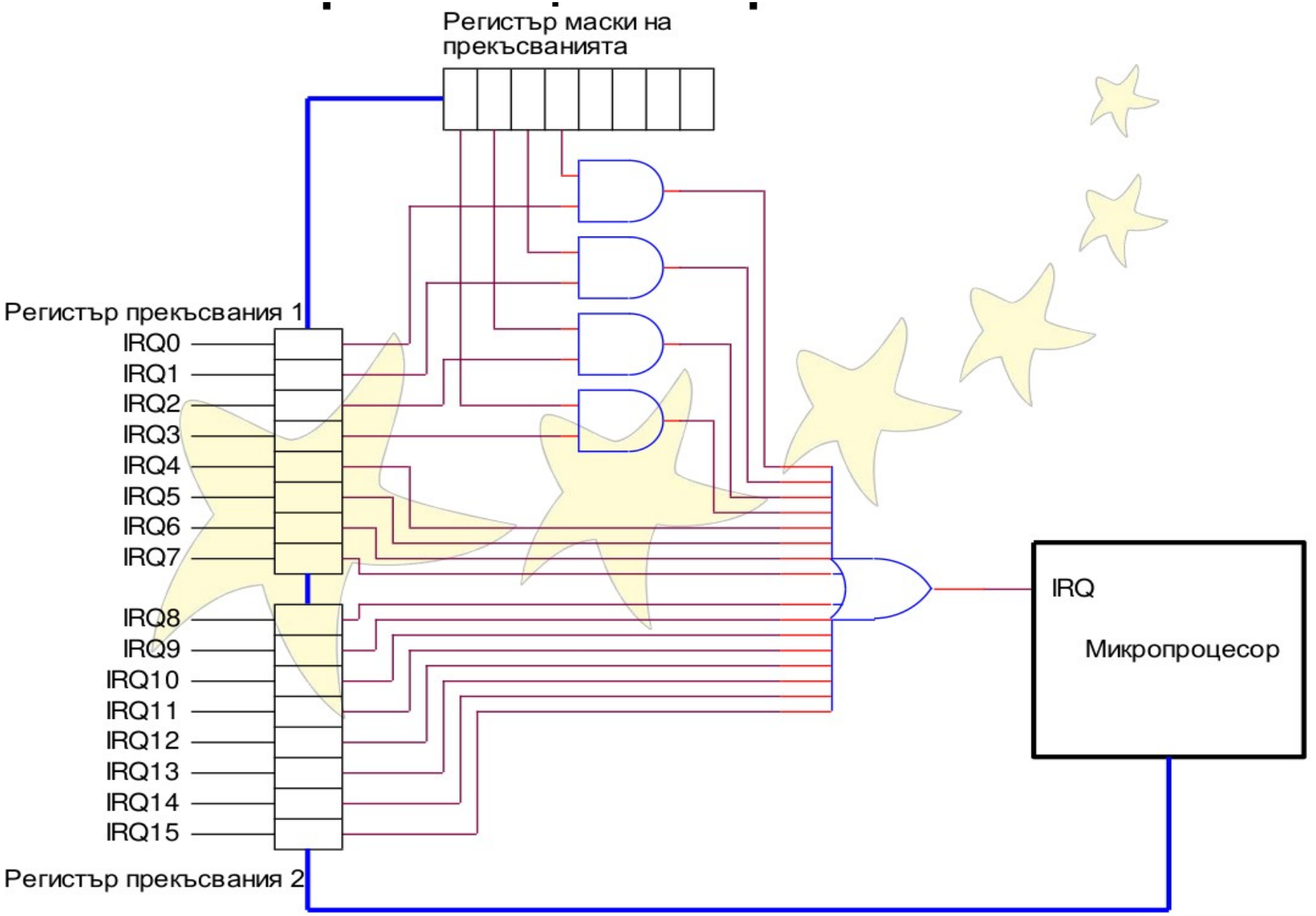
Немаскируеми прекъсвания – прекъсвания, които винаги трябва да се обслужват.

Пример - RESET сигнала – микропроцесорът ще бъде рестартиран винаги при наличието на активно ниво на този сигнал.

Пример - прекъсване при прегряване на чипа.

На следващия слайд е показан пример с 4 маскируеми прекъсвания.

Обслужване на прекъсвания



Обслужване на прекъсвания

Пример – UART модула на MSP430FR6989 има двойката регистри UCAxIFG и UCAxIE, които служат съответно за флагове на прекъсванията и разрешаващи битове на прекъсванията.

Обслужване на прекъсвания

Figure 30-22. UCAxIFG Register

15	14	13	12	11	10	9	8
Reserved							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
Reserved				UCTXCPTIFG	UCSTTIFG	UCTXIFG	UCRXIFG
r-0	r-0	r-0	r-0	rw-0	rw-0	rw-1	rw-0

Table 30-18. UCAxIFG Register Description

Bit	Field	Type	Reset	Description
15-4	Reserved	R	0h	Reserved
3	UCTXCPTIFG	RW	0h	Transmit complete interrupt flag. UCTXCPTIFG is set when the entire byte in the internal shift register got shifted out and UCAxTXBUF is empty. 0b = No interrupt pending 1b = Interrupt pending
2	UCSTTIFG	RW	0h	Start bit interrupt flag. UCSTTIFG is set after a Start bit was received 0b = No interrupt pending 1b = Interrupt pending
1	UCTXIFG	RW	1h	Transmit interrupt flag. UCTXIFG is set when UCAxTXBUF empty. 0b = No interrupt pending 1b = Interrupt pending
0	UCRXIFG	RW	0h	Receive interrupt flag. UCRXIFG is set when UCAxRXBUF has received a complete character. 0b = No interrupt pending 1b = Interrupt pending

Обслужване на прекъсвания

eUSCI_Ax Interrupt Enable Register

Figure 30-21. UCAxIE Register

15	14	13	12	11	10	9	8
Reserved							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
Reserved				UCTXCPTIE	UCSTTIE	UCTXIE	UCRXIE
r-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0

Table 30-17. UCAxIE Register Description

Bit	Field	Type	Reset	Description
15-4	Reserved	R	0h	Reserved
3	UCTXCPTIE	RW	0h	Transmit complete interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
2	UCSTTIE	RW	0h	Start bit interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
1	UCTXIE	RW	0h	Transmit interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
0	UCRXIE	RW	0h	Receive interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled

Обслужване на прекъсвания

Прекъсванията могат да се разрешават на различни нива (спрямо главния вход за прекъсване) – глобално, локално и избирателно.

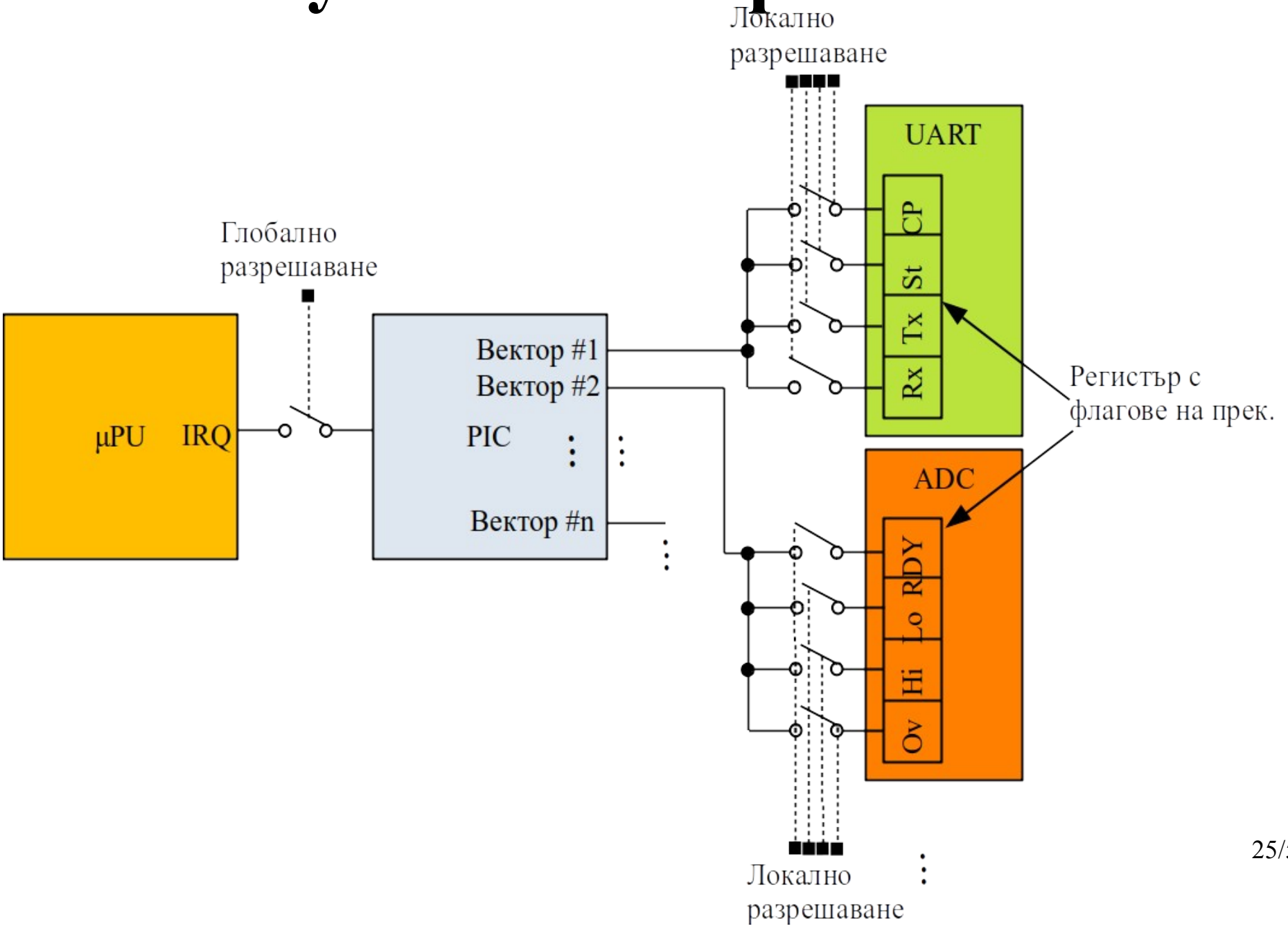
Глобално разрешаване на прекъсванията – входът за прекъсвания на едноядрен μPU се разрешава или забранява. В резултат на това или всички разрешени прекъсвания ще бъдат обслужвани, или нито едно прекъсване няма да бъда обслужено – без значение дали е разрешено или не.

Обслужване на прекъсвания

Локално разрешаване на прекъсванията – флаговете за прекъсванията от един периферен модул се разрешават или забраняват селективно. В резултат на това или тези прекъсвания ще бъдат предадени на глобалния вход за прекъсвания на μPU, или не.

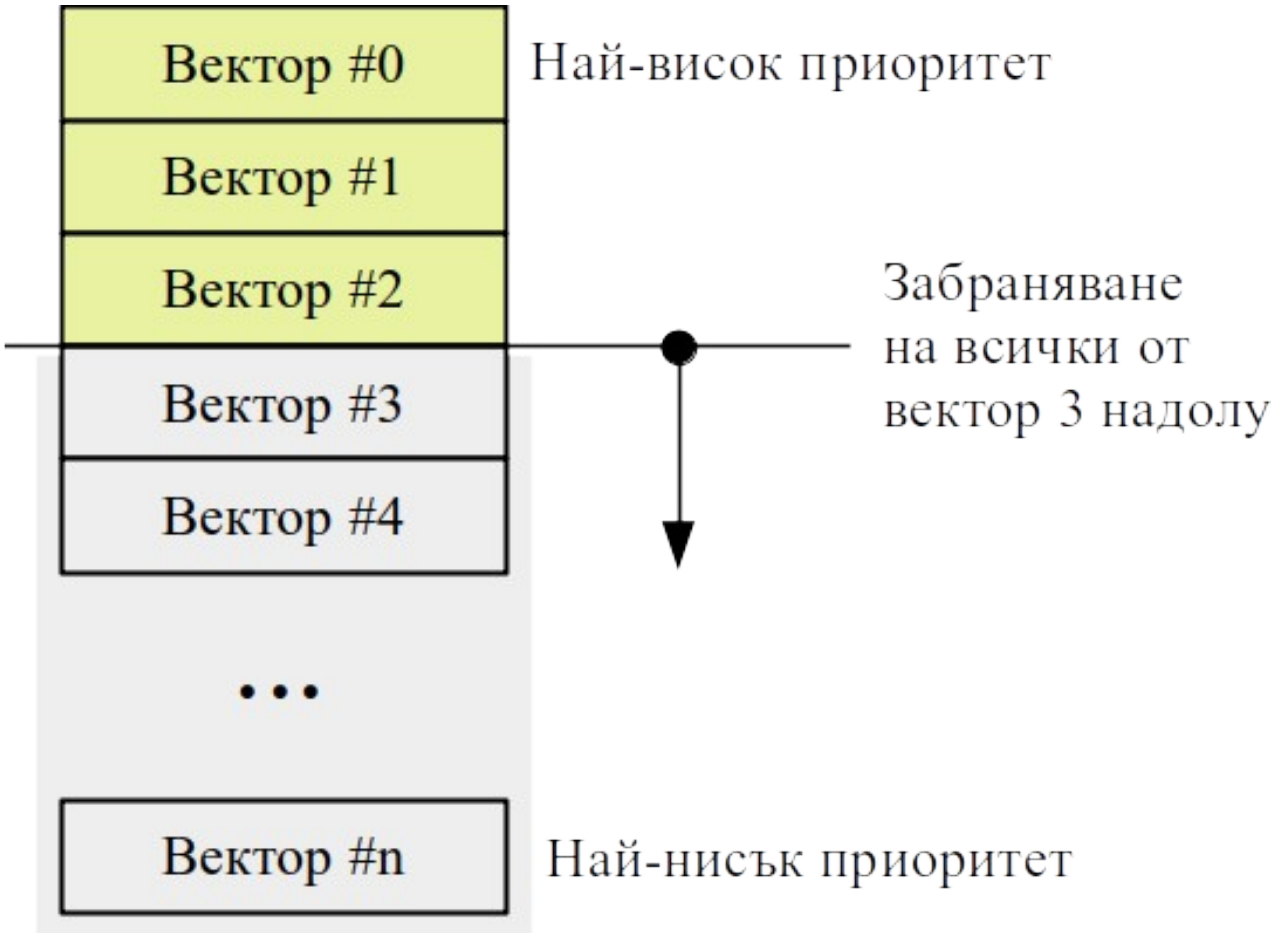
Това означава, че с помощта на локалните прекъсвания временно могат да се забранят дадени функции на системата, докато други останат активни, и обратно.

Обслужване на прекъсвания



Обслужване на прекъсвания

Селективно разрешаване на прекъсванията – всички прекъсвания с приоритет по-нисък от дадена стойност могат да бъдат разрешавани и забранявани.



Обслужване на прекъсвания

Пример – в MSP430 прекъсванията могат да се разрешават локално и глобално.

```
#include <msp430.h>

int main(void){
  WDTCTL = WDTPW | WDTHOLD; // Stop WDT
  P1DIR |= 0x01;                          // P1.0 output
  TA0CCTL0 |= CCIE;                       // CCR0 local interrupt enabled
  TA0CCR0 = 50000;
 TA0CTL = TASSEL_2 | MC_1 | TACLR; //SMCLK, upmode, clear TAR

  __bis_SR_register(GIE); //Enable interrupts globally
  while(1){
       __bis_SR_register(LPM0_bits + GIE); // Enter LPM0,
   }
}

// Timer0 A0 interrupt service routine
void __attribute__ ((interrupt(TIMER0_A0_VECTOR))) timer0_a0_isr (void){
  P1OUT ^= 0x01;   // Toggle P1.0
}
```

Обслужване на прекъсвания

Етапите, през които преминава микропроцесорът при поява на заявка за прекъсване са [1] [2] [3]:
*μPU копира съдържанието на **стековата група** (stack frame) в стека, т.е. някъде около края на SRAM при намаляващ стек, или някъде около началото на SRAM при растящ стек. **Това става хардуерно**, т.е. никъде в кода няма да се видят асемблерни инструкции, които да копират регистри.

Пример – на MSP430 стековата група е съставена от PC, SR.
Пример - ARM Cortex-M стековата група е съставена от PSR, PC, R14, R12, R3, R2, R1, R0.

Обслужване на прекъсвания

За инструкции с числа с плаваща запетая, виж лекцията за FPU.

*паралелно с PUSH-ването на стековата група се **прочита адреса на хендлера** от векторната таблица и се записва в програмния брояч (PC). Това е еквивалентно на влизане в хендлера.
***изпълнява се хендлера** на прекъсването. Добре-написаният фърмуер има много малко код в хендлерите. В хендлерите трябва да се записват флагове, които да сигнализират на main( ) функцията, че събитието се е случило. Обработката на събитието е добре да се случи в main( ).

Времезакъснения (delay) и извеждането на дебъг съобщения (printf) трябва да се избягват в хендлерите.

Обслужване на прекъсвания

Ако в хендлера се използват регистри от ядрото, **които не са част от стековата група**, те трябва да бъдат копирани чрез асемблерни инструкции (PUSH) на стека (SRAM) в началото на хендлера.

*на **излизане от хендлера** се изпълнява една последна специална инструкция, която кара μPU да копира обратно стековата група от SRAM в регистрите на ядрото. Това включва PC, в който се зарежда адреса, до който е било стигнало изпълнението на програмата преди прекъсването. Същото важи и за останалите регистри, които са специфични за всеки μPU (при MSP430 – SR, при ARM Cortex - PSR, R14, R12, R3, R2, R1, R0).

Обслужване на прекъсвания

Инструкцията за излизане от прекъсване при MSP430 е:

reti

Инструкцията за излизане от прекъсване при ARM Cortex е преход (branch) към адрес, записан в специален регистър (link register, lr = r14) :

b r14

Всъщност в r14 има служебен код, който казва на ядрото да възстанови стековата група и да направи преход към стойността на r14, която е PUSH-ната на стека.

Обслужване на прекъсвания

Исторически, цифровата схема, отговорна за съхраняване на векторите на прекъсване, приоритизирането им и максирането им, е била на отделен чип, наречен програмируем контролер на прекъсванията (PIC, **P**rogrammable **I**nterrupt **C**ontroller).

На схемата е показан класическият 8259A, съвместим с 8086, 8088.

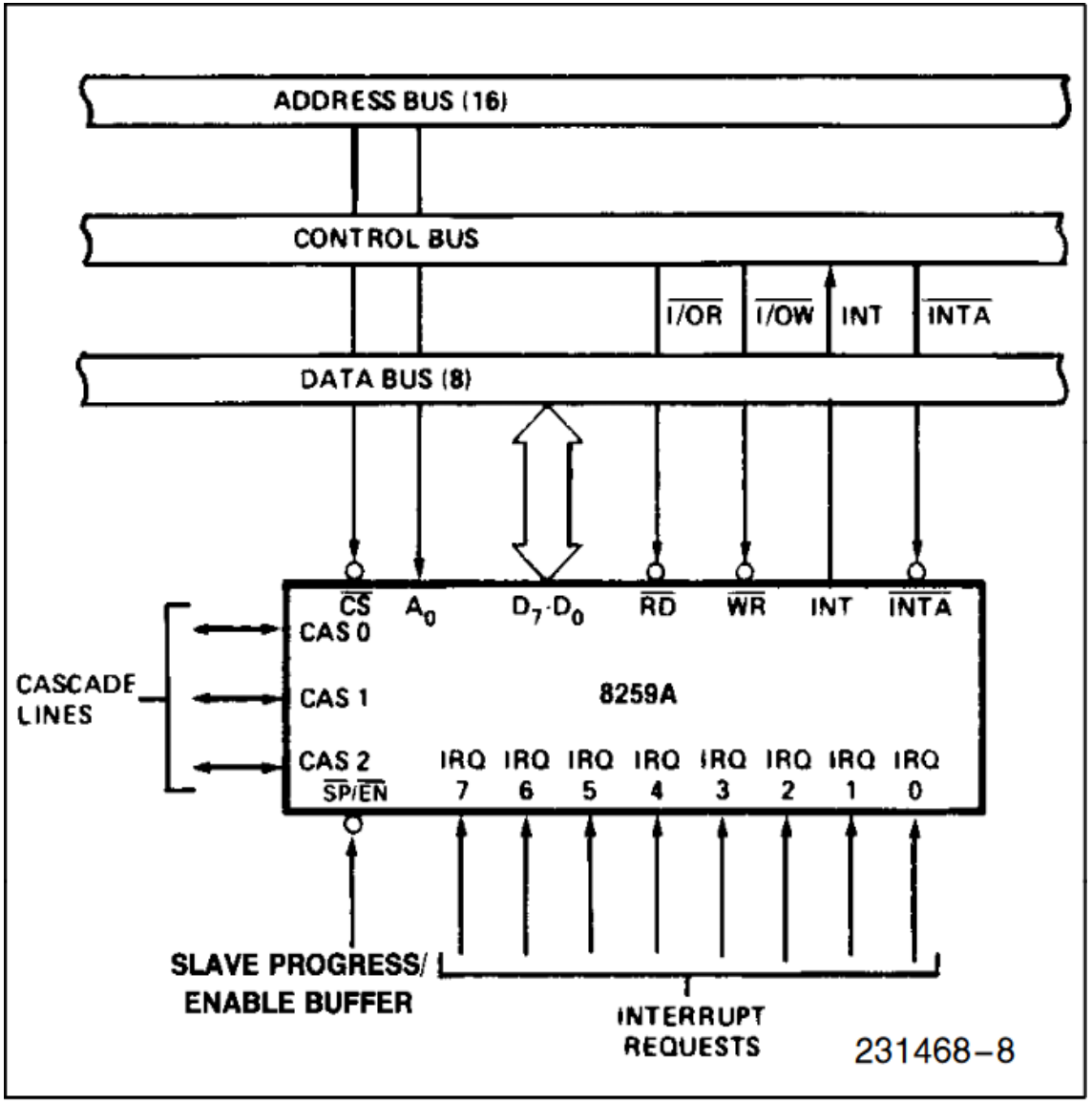


Figure 5. 8259A Interface to Standard System Bus

Обслужване на прекъсвания

В съвременните вградени системи **контролерът на прекъсванията е част от µPU** и се интегрира на чипа.

Източниците на прекъсвания се свързват към входовете на контролера на прекъсванията от **производителя**. Това означава, че приоритетите на прекъсванията са фиксирани от производителя.

ARM Cortex могат да **препрограмират приоритетите** по време на изпълнението на програмата.

При MSP430 всяка периферия има регистър, който указва точно от **кой източник е дошло прекъсването**, което ускорява обслужването му.

Обслужване на прекъсвания

Пример – MSP430, UART модул, IV регистър. Прилича на “малка векторна таблица”, локално в модула.

30.4.12 UCAxIV Register

eUSCI_Ax Interrupt Vector Register

Figure 30-23. UCAxIV Register

15	14	13	12	11	10	9	8
UCIVx							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
UCIVx							
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

Table 30-19. UCAxIV Register Description

Bit	Field	Type	Reset	Description
15-0	UCIVx	R	0h	eUSCI_A interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Receive buffer full; Interrupt Flag: UCRXIFG; Interrupt Priority: Highest 04h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG 06h = Interrupt Source: Start bit received; Interrupt Flag: UCSTTIFG 08h = Interrupt Source: Transmit complete; Interrupt Flag: UCTXCPTIFG; Interrupt Priority: Lowest

Обслужване на прекъсвания

```
void __attribute__ ((interrupt(USCI_A0_VECTOR))) USCI_A0_ISR (void){
  switch(UCA0IV)){
  case 0:
       //Vector 0 - no interrupt
      break;
  case 2:
       //Vector 2 - RXIFG
       ...
      break;
  case 4:
       //Vector 4 – TXIFG
       ...
      break;
  default:
      break;
  }
}
```

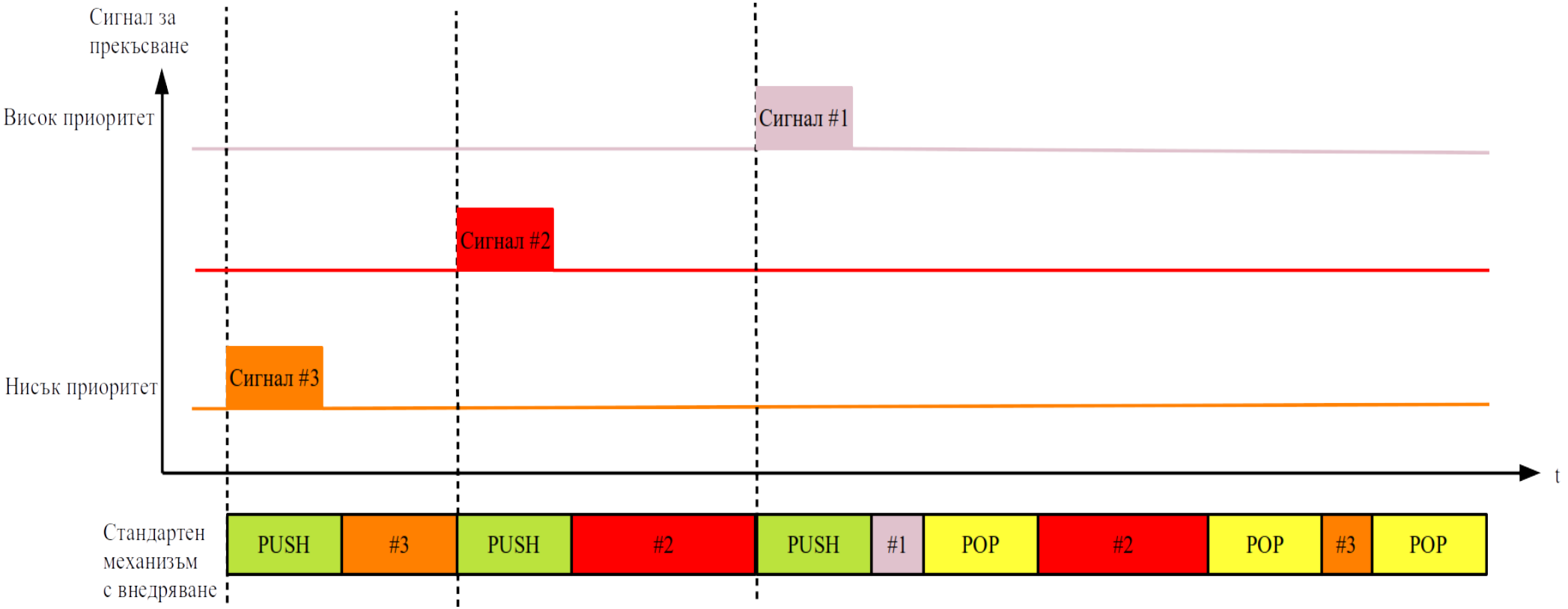
Обслужване на прекъсвания

Прекъсванията изискват специални механизми за обслужване, в случаите когато постъпят два или повече сигнала за прекъсване в един и същ момент.

Внедряване на прекъсванията (interrupt preemption) – изпълнението на хендлера на едно прекъсване може да бъде временно спряно, за да се изпълни хендлера на друго прекъсване. Това може да стане само, ако приоритета на новодошлото прекъсване е по-голям от настоящо-изпълняващото се прекъсване.

Не може да съществуват прекъсвания с един и същ приоритет (освен ако не се въведат суб-приоритети, както е при ARM Cortex).

Обслужване на прекъсвания

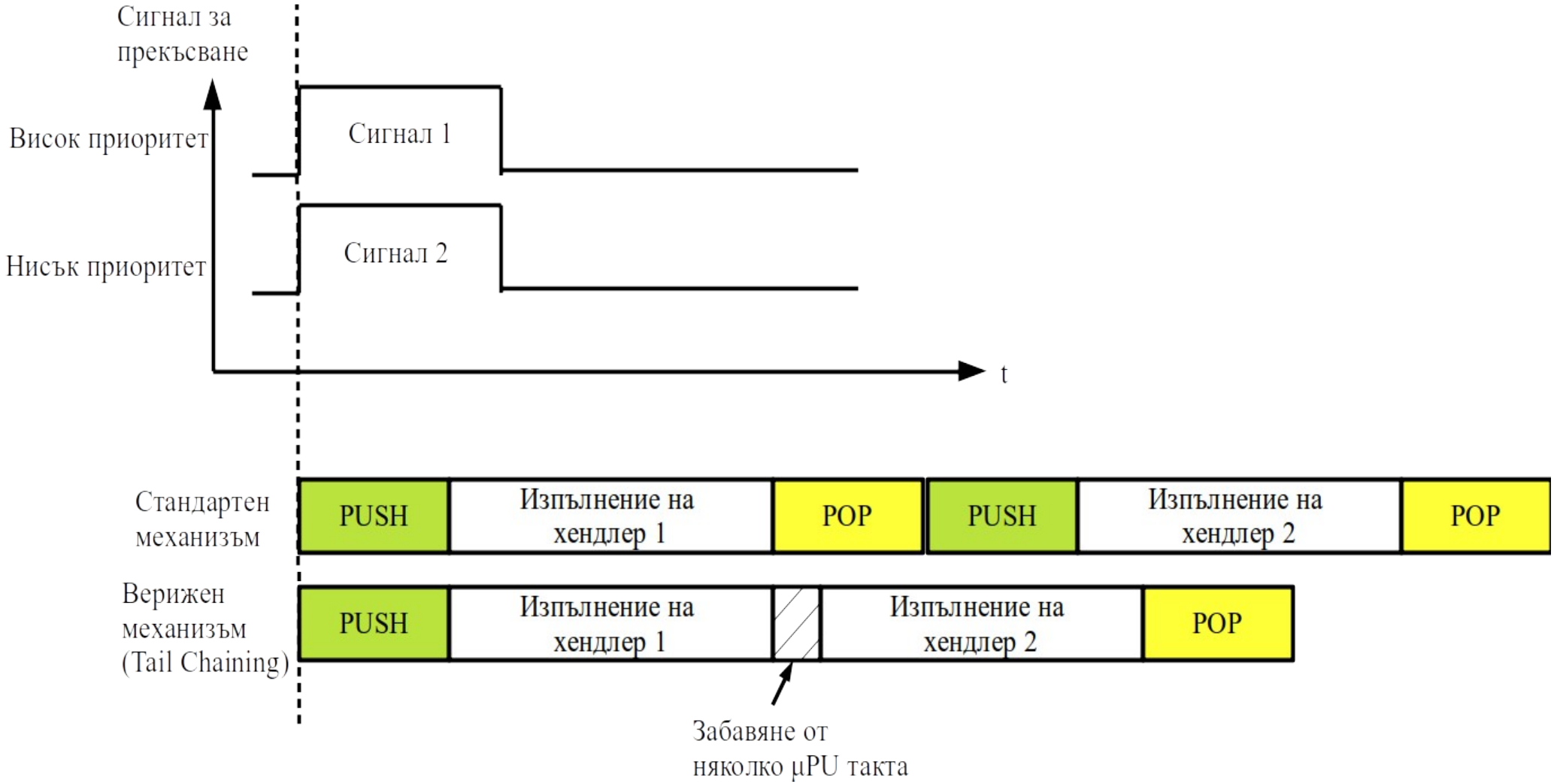


Обслужване на прекъсвания

Верижен механизъм (tail chaining) – при едновременно постъпване на сигнали за прекъсване, преминаването от хендлер 1 в хендлер 2 става без да се записва (PUSH) и възстановява (POP) стека. Резултатът – по-бързо обслужване на прекъсванията спрямо стандартния подход. Методът е демонстриран на следващия слайд[3].

Преминаването от един хендлер в друг не става мигновено, а изисква няколко системни такта, обусловени от вътрешната структура на μPU.

Обслужване на прекъсвания

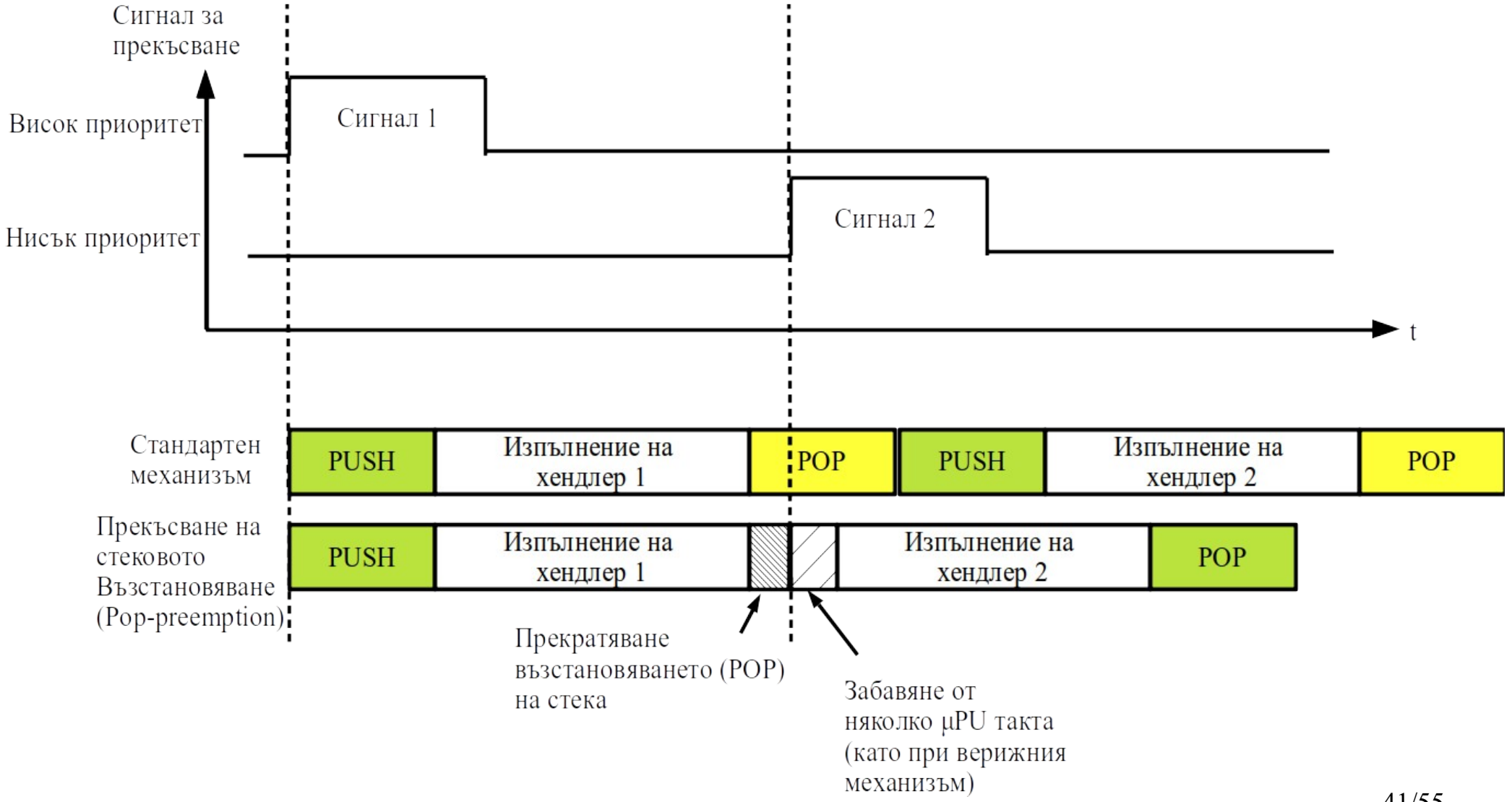


Обслужване на прекъсвания

Прекъсване на стековото възстановяване (pop pre-emption) – при постъпване на сигнал за прекъсване, докато μPU излиза от друго прекъсване, се прекратява възстановяването на стека (POP) и се преминава към изпълняване на следващия хендлер. Методът е демонстриран на следващия слайд.

Прекратяването на процеса по възстановяване на стека не може да стане мигновено и са необходими няколко системни такта за целите на микропроцесорната логика.

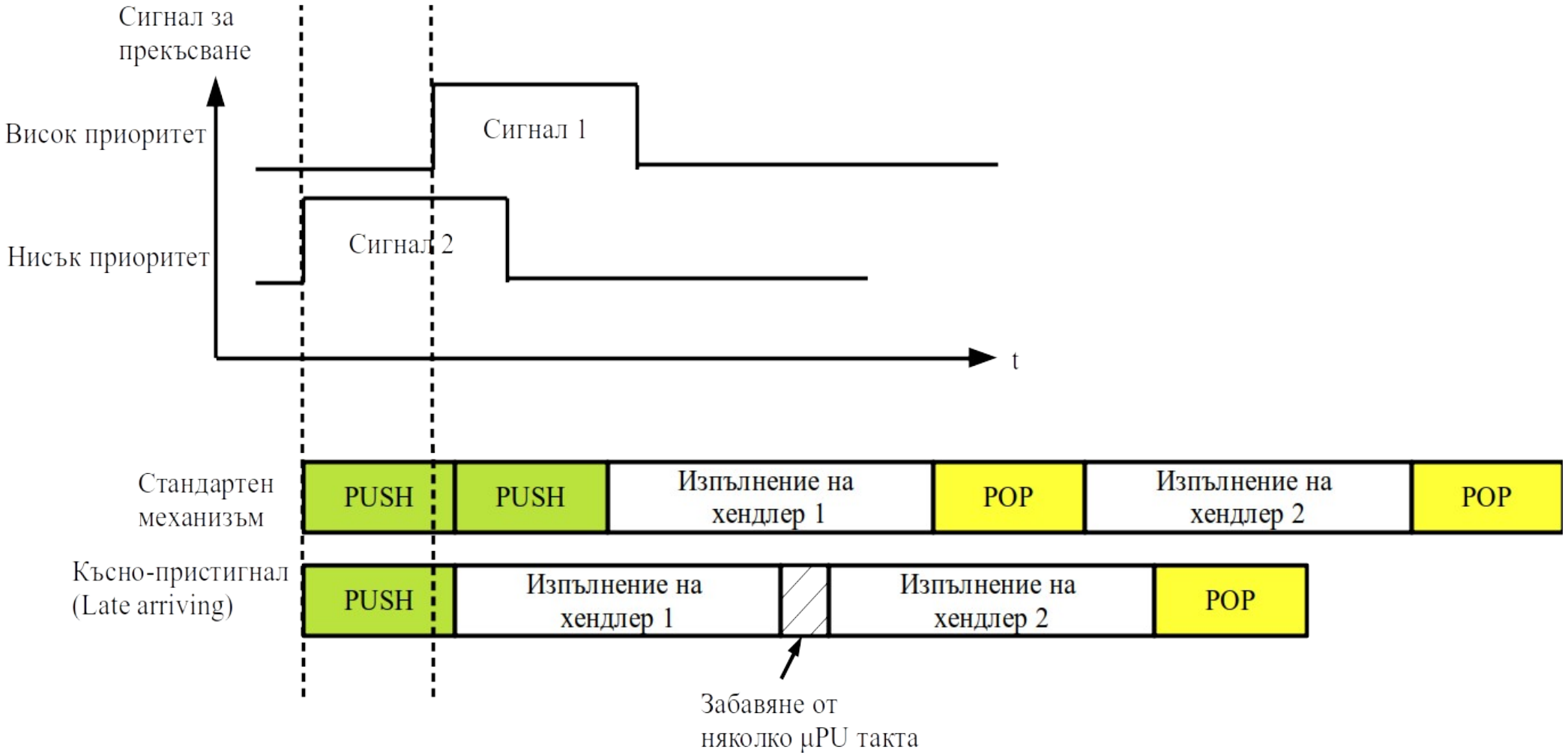
Обслужване на прекъсвания



Обслужване на прекъсвания

Късно-пристигнал (late arriving) – изпълнение на хендлер с висок приоритет във времевия слот на хендлер с нисък приоритет, ако първият е пристигнал по време на PUSH операцията на втория.

Обслужване на прекъсвания



Контролер за директен достъп (DMA)

Контролерът за директен достъп до паметта (**D**irect **M**emory **A**ccess controller - DMA) служи за автоматично прехвърляне на данни от един адрес на друг без намесата на микропроцесора.

Това се прави с цел повишаване производителността на микропроцесорната система.

Използват се трансфери от периферен модул и/или RAM паметта.

Контролер за директен достъп (DMA)

Съществуват три вида реализации на DMA[4]:
*DMA със спиране на μPU
*DMA с отнемане на цикли (interleaving)
*Паралелно DMA

DMA със спиране на μPU – DMA е свързан към магистралата/магистралите на μPU. Докато трае директния трансфер, μPU е блокиран и не изпълнява инструкции. Когато приключи трансфера, μPU продължава изпълнението на програмата.

Контролер за директен достъп (DMA)

DMA с отнемане на цикли (interleaving) – DMA е свързан към магистралата/магистралите на μPU. Докато DMA извършва трансфер, μPU е блокиран и не изпълнява инструкции. След даден брой трансфери (определен от програмиста) DMA се блокира и се пуска отново μPU. След няколко такта μPU се блокира отново и се дава времеви прозорец на DMA и т.н.

Така μPU и DMA никога не достъпват системната магистрала/и едновременно. Когато DMA трансфера приключи, μPU продължава да работи на 100 %.

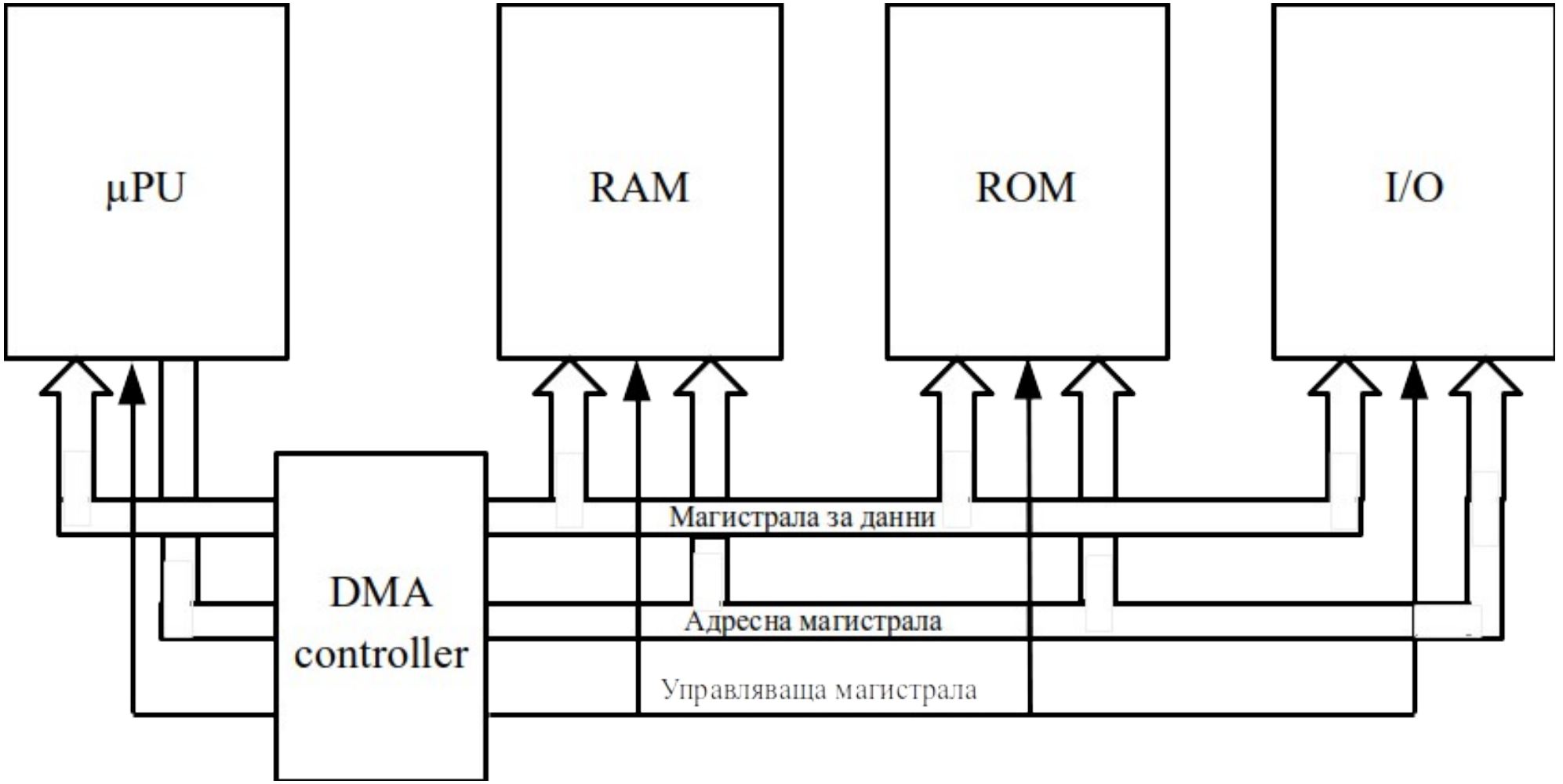
Изброените методи са показани на по-следващия слайд.
Пример – Intel 8080 може да използва външната схеMSP430

Контролер за директен достъп (DMA)

Пример – Intel 8080 може да използва външната DMA ИС 8257 и да работи в режим със спиране. За целта 8080 има сигнал HOLD.

Пример – MSP430 може да работи с вградения си DMA контролер режим със спиране (single & block transfer) или в режим с отнемане на цикли (burst-block mode).

Контролер за директен достъп (DMA)



Контролер за директен достъп (DMA)

Защо ни е DMA, ако трансферите блокират μPU?

Контролер за директен достъп (DMA)

При MSP430 DMA копира данни от кой-да-е адрес към друг адрес за **3 такта** [5]. Броят байтове/думи може да се задава.

Аналогична програма на Асемблер би изглеждала така:

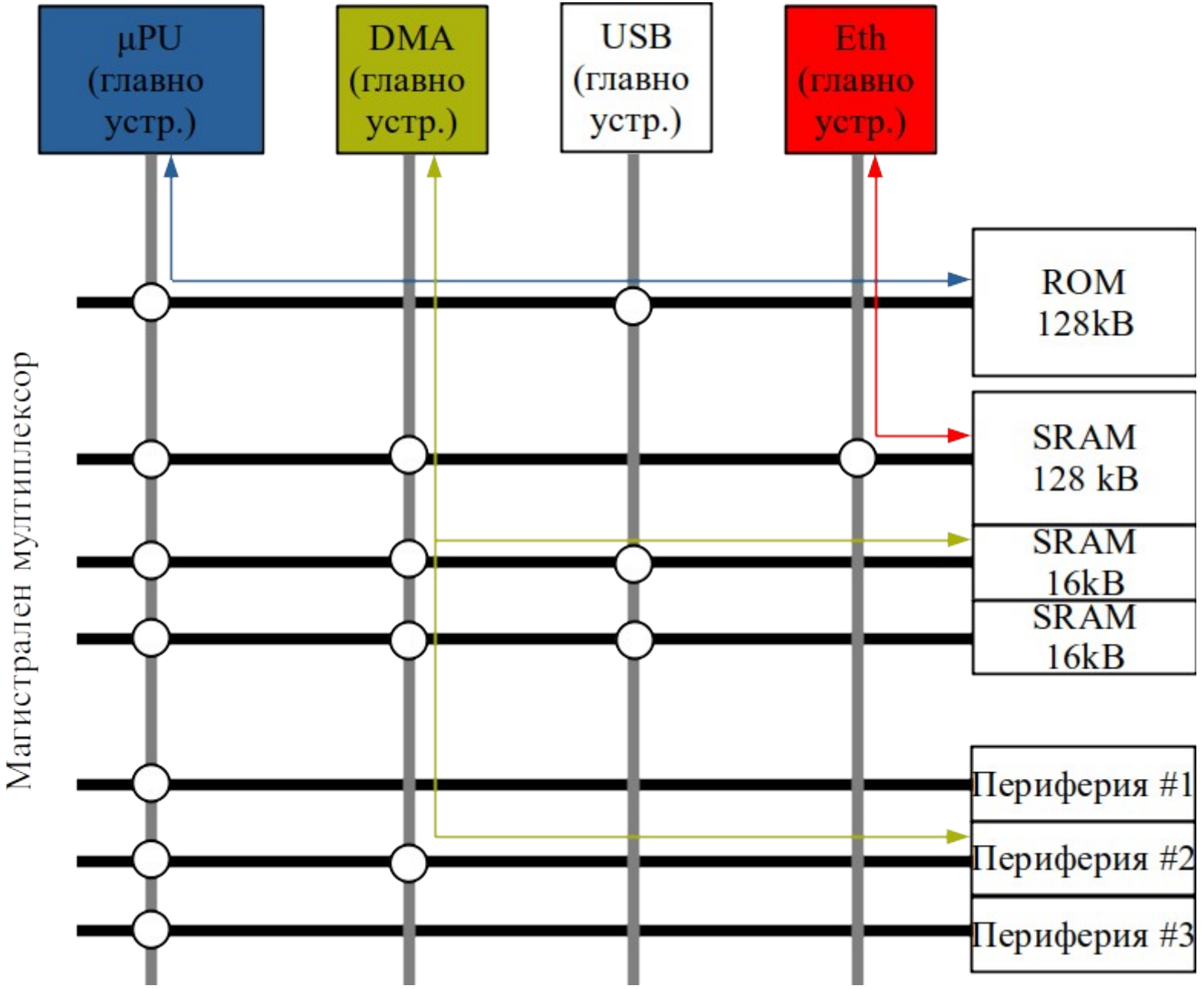
```
        mov.w #0x1c00, r8
        mov.w #0x1c40, r9
        mov.w #0x1c50, r7

copy: mov.w   @r8+, 0(r9)    ;4 такта
       add.w  #2, r9         ;2 такта
       cmp    r7, r9         ;1 такт
       jnz    copy           ;2 такта
```

Контролер за директен достъп (DMA)

Паралелно DMA – DMA е свързан към мултиплексор на магистрали (switch matrix). В системата има няколко магистрали, всяка от които се свързва само с определена периферия. Ако μPU и DMA достъпват периферия по магистрали, които не се застъпват, DMA трансферите са изцяло паралелни [3]. Ако се застъпват – използва се режим с отнемане на цикли.

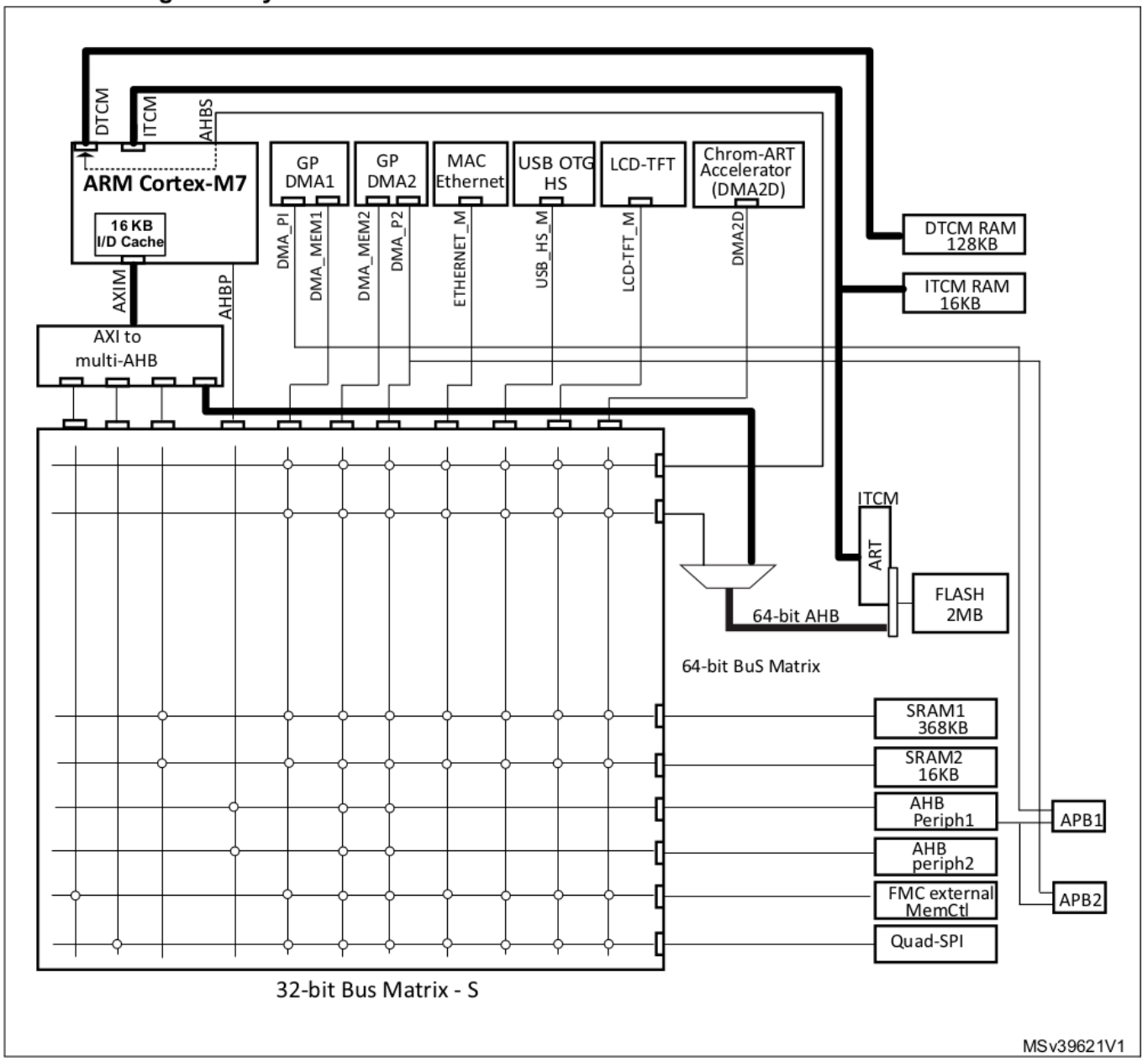
Контролер за директен достъп (DMA)



Контролер за директен достъп (DMA)

Пример – STM32F769 разполага с магистрален мултиплексор и 8 набора от магистрали.

Figure 1. System architecture for STM32F76xxx and STM32F77xxx devices



Контролер за директен достъп (DMA)

Осцилограми с активността на μPU и DMA в трите режима на работа.

Литература

[1]Pedro Dinis Gaspar, Antonio Santo, Bruno Ribeiro, Humberto Santos, “Device Systems and Operating Modes”, chapter 5, TI & University of Beira Interior (PT), 2009.

[2]Н. Кенаров, “PIC Микроконтролери”, Част 1, Млад Конструктор, Варна, 2003.

[3]M. Trevor, „The Designer’s Guide to the Cortex-M Processor Family – A Tutorial Approach“, Elsevier, 2013.

[4]К. Боянов, В. Кисимов, Л. Бончев, К. Янев, А. Петков, “Сборник приложни схеми с микропроцесори”, Техника, 1981.

[5]Pedro Dinis Gaspar, Antonio Santo, Bruno Ribeiro, Humberto Santos, “Device Systems and Operating Modes”, chapter 11, TI & University of Beira Interior (PT), 2009.