

# Микропроцесорна системотехника

## Лабораторно упражнение №2

### “Видове адресации и инструкции на микроконтролер MSP430FR6989”

#### 2.1. Въведение.

Всяка една програма на C за микроконтролери се преобразува в програма на Асемблер, а след това всяка една асемблерна инструкция се преобразува в двоичен код, който се зарежда в микроконтролера. Познаването на Асемблер за поне един микроконтролер е от особено значение в програмирането на вградени системи. Макар Асемблерите да са различни за различните микроконтролери, между тях има много общи черти. Ето и някои примери, в които познаването на Асемблер е от значение:

а) Понякога програмистът не може да предвиди, как точно един ред от C програмата ще се „преведе“ на Асемблер. Това води до грешки, чието отстраняване (debug) изисква преглед на „преведената“ програма. Например, празен for-цикъл от вида `for(i = 0; i < 20; i++){ }` може да се премахне от C-компилятора, който да счита тази конструкция за ненужна. Реално обаче програмистът може да я използва като софтуерно закъснение. Тогава единствено чрез поглеждане в „преведената“ на Асемблер програма може да се види липсата на този код.

б) Оптимизация от гледна точка на бързодействие и/или размер на програмата може да се постигне чрез програмиране на Асемблер. Микроконтролерите обикновено разполагат с минимален хардуерен ресурс (в сравнение с настолните компютри) и комбинирането C/Асемблер е често срещано. Пример → инициализиращ код на Linux за вградени системи.

#### 2.2. Видове инструкции.

Всяка една инструкция се състои от код на операцията и операнди. Фирмата Texas Instruments дефинира четири вида инструкции на микропроцесора MSP430:

Инструкция с два операнда (наричани още **Формат I инструкции**).

*Пример:* `add R3, R4` → Събери стойностите на регистри R3 и R4 и запиши резултата в R4 върху старата стойност.

Инструкция с един операнд (наричани още **Формат II инструкции**).

*Пример:* `RRA R3` → премести всички битове на числото в R3 с една позиция надясно като най-младшия бит запиши в STATUS регистъра на позиция Carry).

Инструкции за преход (наричани още **Формат III инструкции**).

*Пример:* `jmp 0x1000` → иди на абсолютен адрес 0x1000.

Инструкции, които не попадат в никоя категория (наричани още **Разни инструкции**)

*Пример:* Хендлер на прекъсване:

`add R3, R4`

`rra R3`

`reti`

→ Излез от хендлер на прекъсване

*Прекъсванията и хендлерите на прекъсвания са обяснени в лабораторното упражнение за входно/изходните портове.*

Формат I и II инструкциите имат наставка .b или .w, които означават работа със съответно 8-битови и 16-битови операнди. Така например инструкцията `add` може да се запише като `add.b R3, R4` или `add.w R3, R4`. Ако не се укаже наставка, асемблерът по подразбиране слага .w наставка.

Използването на наставки произлиза от факта, че част от периферните модули са разположени на адреси  $0x00 \div 0xFF$ , а друга част на  $0x100 \div 0x1FF$ . Първите изискват работа с „b” инструкции, а вторите – работа с „w” инструкции. Не може да се използва .w на периферия в обхвата  $0x00 \div 0xFF$ , както и .b на периферия в  $0x100 \div 0x1FF$ .

Микропроцесорът MSP430 има 16-битова адресна шина. Това води до ограничение на адресираната памет от 0 до 65535 адреса (или  $0 \div 64kB$ ). За да може да се произведат микроконтролери с повече памет и периферия (периферията се адресира като памет), от Тексас Инструментс са проектирали разширен вариант на MSP430 и това е MSP430X. В него адресна шина е 20-битова, което позволява  $0 \div 2^{20} = 0 \div 1048576$  (или  $0 \div 1 MB$ ). MSP430FR5739 е с MSP430X микропроцесор. За да се адресира модул или памет, разположен на адрес по-голям от  $0xFFFF$  се използва наставката A след мнемониката на инструкцията, например BR  $0xFFFF$  (16-bit) и BRA  $0x10000$  (20-bit).

**2.3. Дисасемблер (Асемблерен листинг).** Дисасемблер (disassembly), или още асемблерен листинг, е изходен файл, съдържащ програма на Асемблер, заедно с машинния код, съответстващ на всяка една инструкция и адреса, от който тя се изпълнява. На **фиг. 2.1** е показан дисасемблера на C програма.

e04:	b5f8	push	{r3, r4, r5, r6, r7, lr}
e06:	4e0b	ldr	r6, [pc, #44] ; (e34 <UARTwrite+0x30>)
e08:	460d	mov	r5, r1
e0a:	4604	mov	r4, r0
e0c:	1847	adds	r7, r0, r1
e0e:	42bc	cmp	r4, r7
e10:	d00d	beq.n	e2e <UARTwrite+0x2a>
e12:	f814 3b01	ldrb.w	r3, [r4], #1
e16:	2b0a	cmp	r3, #10
e18:	d103	bne.n	e22 <UARTwrite+0x1e>
e1a:	6830	ldr	r0, [r6, #0]
e1c:	210d	movs	r1, #13
e1e:	f7ff ffb1	bl	d84 <UARTCharPut>

Адрес  
на инстр.

Машинен  
код

Програма на  
Асемблер

**Фиг. 2.1**

Байтовете, които се зареждат в паметта на контролера са показани в средната колонка. Сама по себе си тази информация не е достатъчна за успешното изпълнение на програмата от микропроцесора. Трябва да се знае всеки един байт на кой адрес се намира. Адресите са показани в най-лявата колонка. Микропроцесорът „разбира“ само от числата в тези две колонки. Всичко друго – програма на Асемблер, програма на C, програма на C++, програма на Java е абстракция за улеснение на програмиста.

## 2.4. Видове адресации.

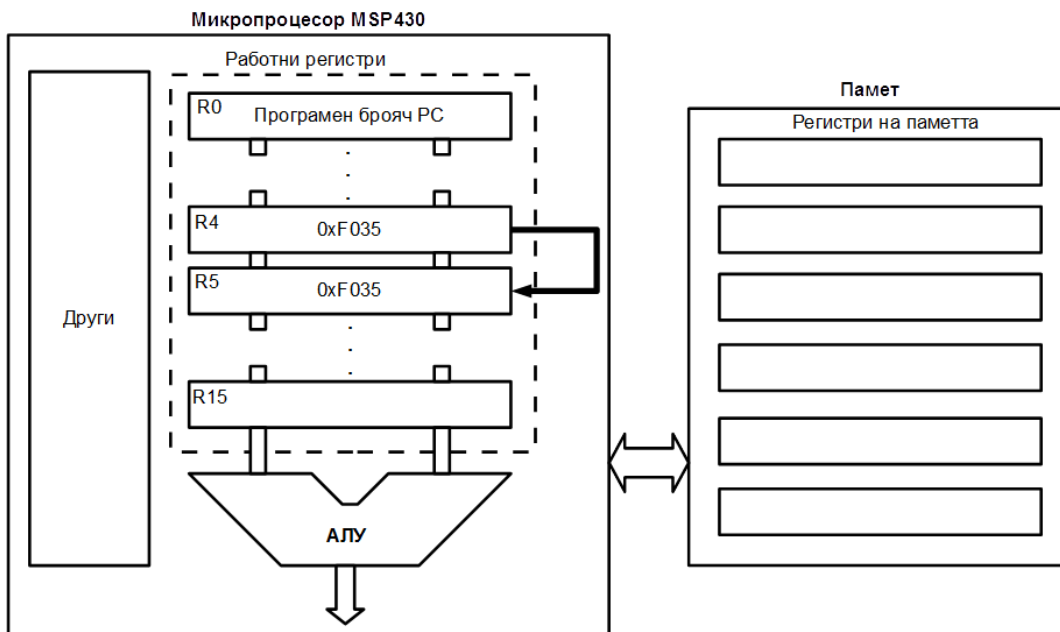
Операндите на една инструкция могат да се извлекат по различен начин, от различни адреси в паметта или работните регистри на ядрото. Това обуславя различни видове адресации или още режими на адресиране (Addressing modes). Техният брой варира при различните микропроцесори. За MSP430 са 7 броя. Трябва да се прави разлика между работните регистри на ядрото и външната (за ядрото) памет.

**2.4.1 Регистрова адресация (Register)** – съдържанието на работен регистър от ядрото е операнд на инструкцията.

*Пример:* mov.w R4,R5

*Преди:* R4 =  $0xF035$ , R5 =  $0x5555$

*След:* R4 =  $0xF035$ , R5 =  $0xF035$

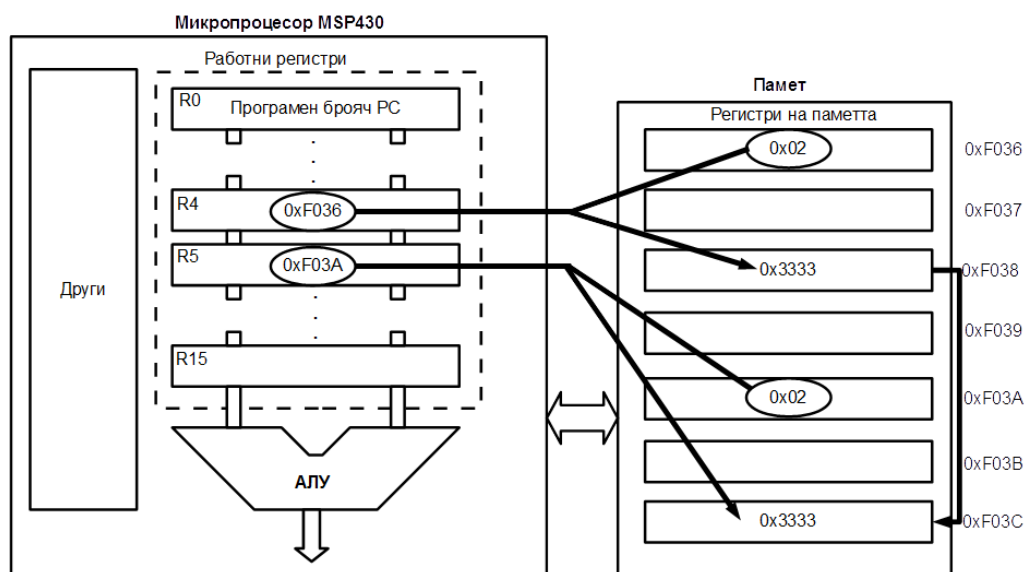


**2.4.2. Индексна адресация (Indexed)** – съдържанието на работен регистър от ядрото + съдържанието на регистър от програмната памет (отместване) указват адреса (някъде в паметта на микроконтролера) на операнда.

*Пример:* `mov.w 0x02(R4), 0x02(R5)`

*Преди:*  $R4 = 0xF036$ ,  $R5 = 0xF03A$ ,  $MEM(0xF038) = 0x3333$ ,  $MEM(0xF03C) = 0x5555$

*След:*  $R4 = 0xF036$ ,  $R5 = 0xF03A$ ,  $MEM(0xF038) = 0x3333$ ,  $MEM(0xF03C) = 0x3333$

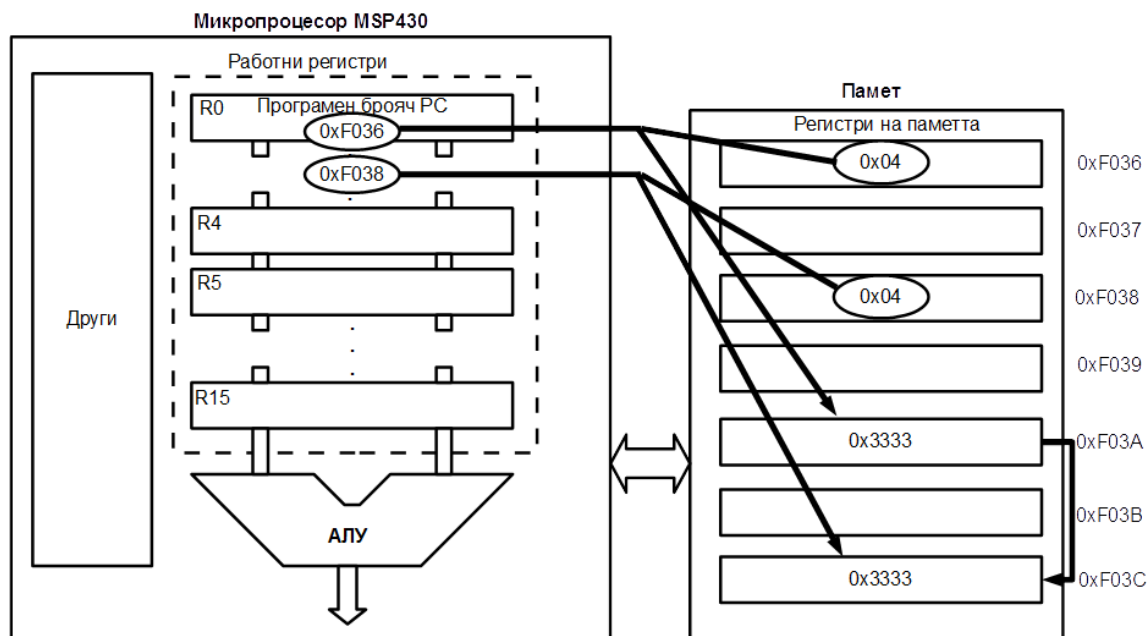


**2.4.4. Относителна (Symbolic) адресация** – съдържанието на регистър от програмната памет (отместване) + стойността на програмния брояч PC указват адреса на операнда. Тази адресация е аналогична на индексната, с тази разлика, че се използва не-кой да е регистър от ядрото, а точно регистър R0 (т.е. програмния брояч).

*Пример:* `mov.w Label1, Label2` (етикетите в Асемблер са относителни, или още условни адреси, които се заменят с абсолютни след линкването на програмата). Нека Label1 е с адрес 0xF036, а Label2 е с адрес 0xF038.

*Преди:*  $PC = 0xF036$ ,  $MEM(0xF03A) = 0x3333$ ,  $MEM(0xF03C) = 0x5555$

*След:*  $PC = 0xF03A$ ,  $MEM(0xF03A) = 0x3333$ ,  $MEM(0xF03C) = 0x3333$

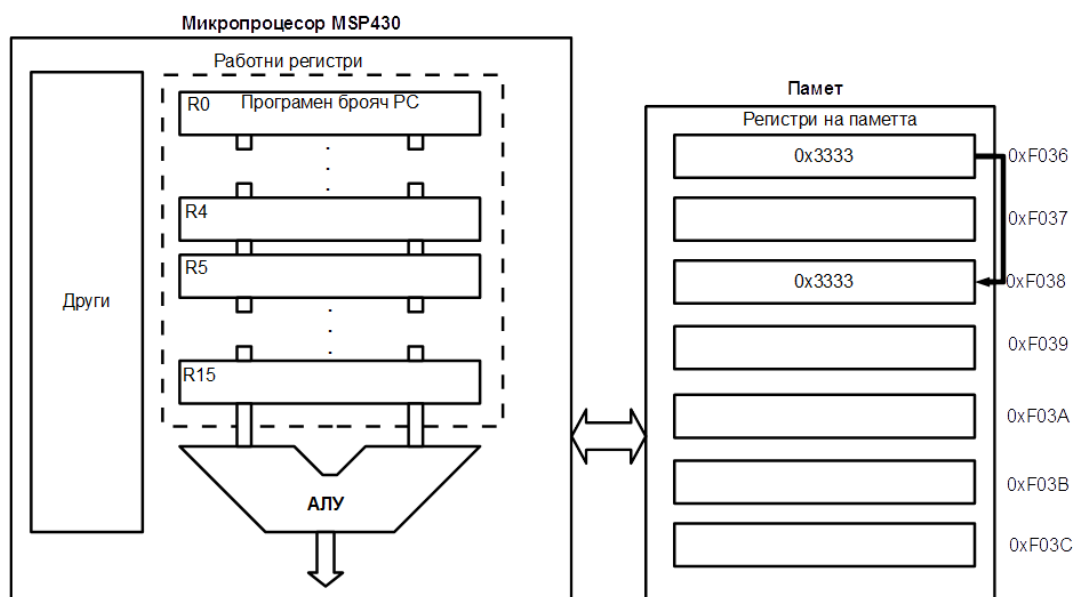


**2.4.3. Абсолютна адресация (Absolute)** – съдържанието на регистър от програмната памет указва адреса (някъде в паметта на микроконтролера) на операнда.

*Пример:* `mov.w &0xF036,&0xF038` (иди на адрес 0xF036 и копирай каквото има там на адрес 0xF038)

*Преди:* `MEM(0xF036) = 0x3333, MEM(0xF038) = 0x5555`

*След:* `MEM(0xF036) = 0x3333, MEM(0xF038) = 0x3333`

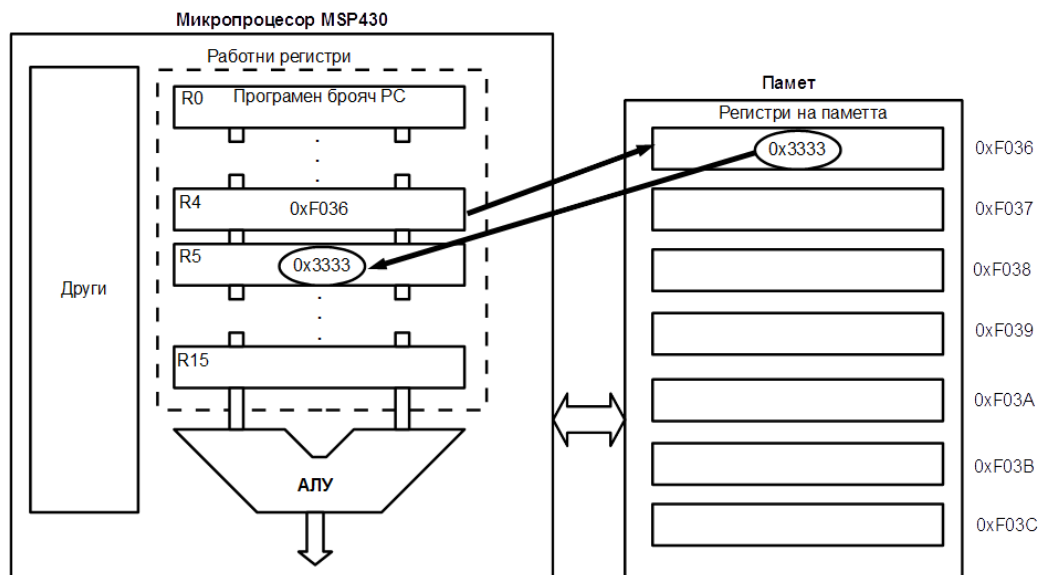


**2.4.5. Индиректна регистрова адресация (Indirect Register)** – аналогична на абсолютната. Разликата – съдържанието на работен регистър от ядрото указва абсолютен адрес от паметта, на който се намира операнда.

*Пример:* `mov.w @R4,R5`

*Преди:* `MEM(0xF036) = 0x3333, R4 = 0xF036, R5 = 0x5555`

*След:* `MEM(0xF036) = 0x3333, R4 = 0xF036, R5 = 0x3333`



**2.4.6. Индиректна автоинкрементираща адресация (Indirect Autoincrement)** – съдържанието на работен регистър от ядрото указва абсолютен адрес от паметта, на който се намира операнда. След изпълнение на инструкцията, съдържанието на работния регистър се увеличава автоматично с 1, ако инструкцията е с наставка „b” или се увеличава с 2, ако инструкцията е с наставка „w”.

*Пример:* mov.w @R4+,R5 (R4 се увеличава с 2)

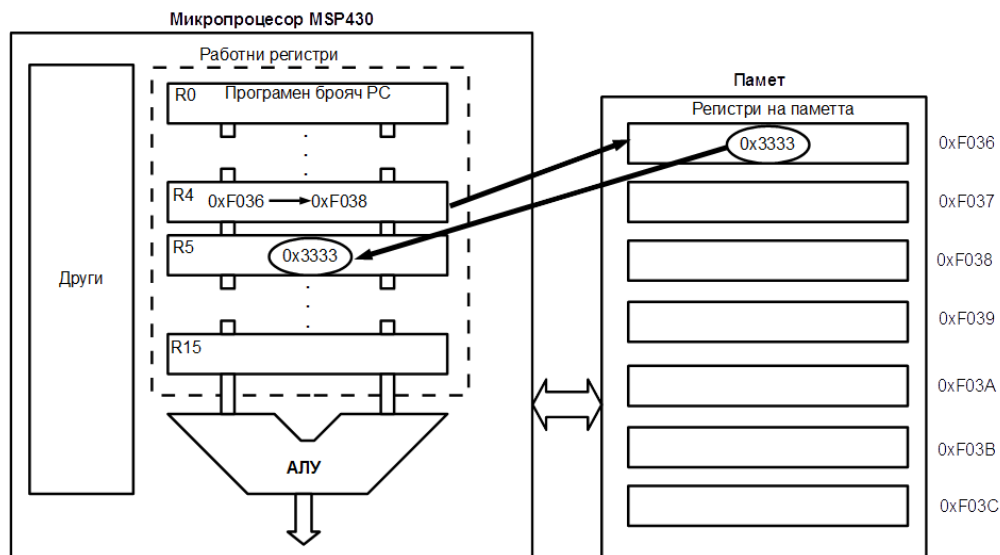
*Преди:* MEM(0xF036) = 0x3333, R4 = 0xF036, R5 = 0x5555

*След:* MEM(0xF036) = 0x3333, R4 = 0xF038, R5 = 0x3333

*Пример:* mov.b @R4+,R5 (R4 се увеличава с 1)

*Преди:* MEM(0xF036) = 0x3333, R4 = 0xF036, R5 = 0x5555

*След:* MEM(0xF036) = 0x3333, R4 = 0xF037, R5 = 0x0033 (старшата част се нулира заради „b” наставката)

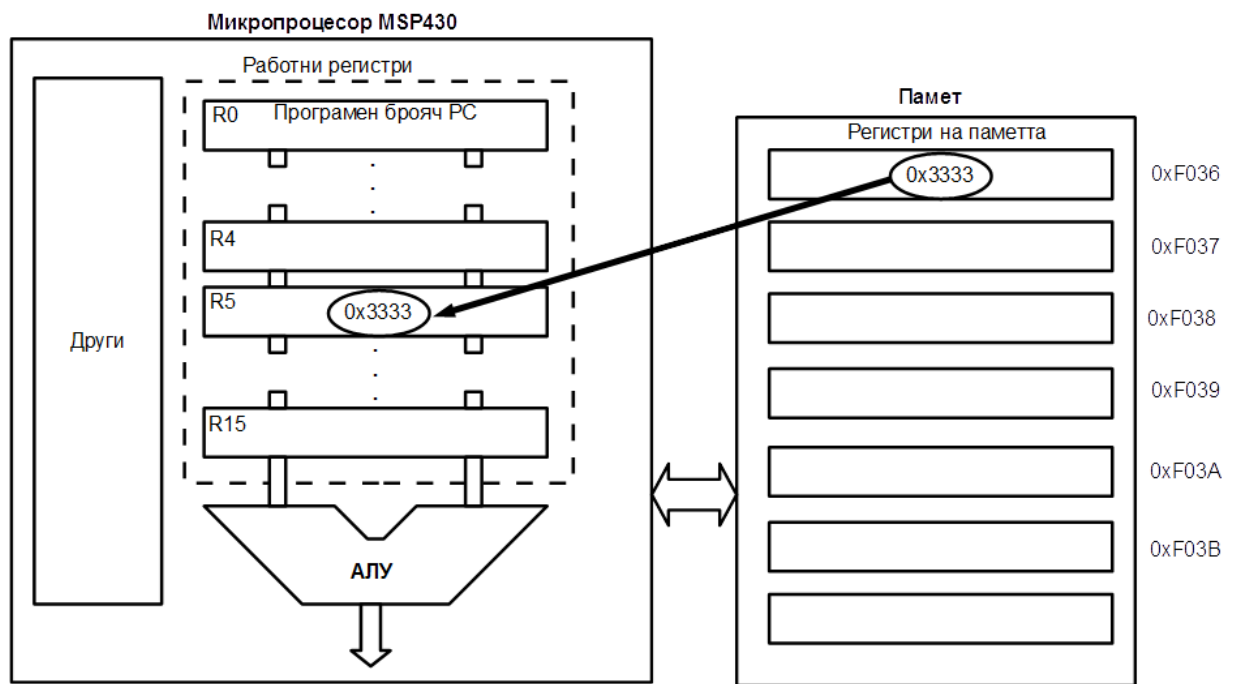


**2.4.7. Непосредствена адресация (Immediate)** – съдържанието на регистър от програмната памет (константа) е операнд на инструкцията.

*Пример:* mov.w #0x3333,R5

*Преди:* R5 = 0x5555

*След:* R5 = 0x3333



## ЗАДАЧИ ЗА ИЗПЪЛНЕНИЕ

1. Да се създаде нов проект с име **Lab\_2\_1** в папка **/Desktop/MSHT\_GR\_XX\_N**, да се копира програмата на асемблер, която извършва примерни операции, демонстриращи видовете адресации. Използва се само една инструкция – MOV с цел опростяване, НО адресациите са валидни за всички инструкции на MSP430.

2. Да се постави точка на прекъсване точно след реда:

**MOV.W #WDTPW+WDTHOLD,&WDTCTL**

Да се изпълнява стъпка по стъпка, като едновременно с това се попълва таблицата, показана по-долу. Програмата приключва на реда „**JMP \$**“.

1	2	3	4					5		8			
№	Инструкция	Инструкция в шестнадесетичен вид	Съдържание на регистрите					Вид Адресация (Reg, IX, Sym, Abs, Ind, Ind+, Imm)		RAM			
			PC	SP	SR	R4	R5	SRC	DST	0x1C00	0x1C01	0x1C04	0x1C05

Инструкцията в шестнадесетичен вид се взима от прозореца **Disassembly**.

Адресацията се записва за левия (SRC – source) и десния (DST – destination) операнд. Ако има само един операнд се пише само в DST. Ако няма операнди се поставя типе.

Показване на регистрите на ядрото: **View → Register → Core Registers**

Показване на регистрите на паметта: **View → Memory Browser**. Изберете наблюдение на регион 0x1C00 (началото на SRAM).

Показване на дисасемблер: **View → Disassembly**