студент: ..... група:..... преподавател: .....

МЕТОДИ ЗА АДРЕСАЦИЯ. ПРОГРАМИРАНЕ НА АСЕМБЛЕР AS 11.

I. ТЕОРЕТИЧНА ПОСТАНОВКА. ПРИМЕРИ

1. ПРОГРАМЕН МОДЕЛ. КЛАСИФИКАЦИЯ НА СИСТЕМАТА ОТ ИНСТРУКЦИИ НА *68HC11*

1.1. Програмен модел (Приложение 1)

1.2. Инструкции за работа с акумулаторите и паметта (Приложение 2-A):

- ✓ зареждане, изпращане и прехвърляне
- ✓ аритметични операции
- ✓ логически операции
- ✓ побитова обработка
- ✓ преместване и ротация

1.3. Инструкции за стека и индексните регистри (Приложение 2-B)

1.4. Инструкции за регистъра на състоянието (Приложение 2-C)

1.5. Инструкции за управление на програмата (Приложение 2-D):

- ✓ преходи
- ✓ подпрограми
- ✓ прекъсвания

2. МЕТОДИ ЗА АДРЕСАЦИЯ (начини за указване на мястото, където се намират операндите).

Илюстрирани са с кратки програмки за събиране на числа, въвеждани от примерен адрес $1000 с помощта на вградения в монитора поредов асемблер (команда ASM). Понеже той работи само с шестнадесетични числа, с помощта на фрагмента STAA 1004 от предходното упражнение #2, крайният резултат се визуализира и в двоичен вид от светодиодите PB0 (младши бит) - PB7 (старши бит). Инструкциите със съответната адресация са подчертани с по-тъмен шрифт.

<u>Непосредствена адресация</u>. При нея фактически няма реално адресиране, защото в полето на операнда се намират самите данни (а не техният адрес). Поради това не може да се използва за четене/изпращане на данни от/на конкретен физически адрес (например, порт). Данните са разположени в един или два байта (първо старша, после - младша част) непосредствено след кода на операцията.

$1000	PRE
$1001	КОД
$1002	C^{tn}_{H} (данни)
$1003	C^{tn}_{L} (данни)

Пример: Прибавяне на число към съдържанието на акумулатор. Програмата няма нужда от инициализация (начално установяване), защото числата са записани в самата програма:

```
0100    xxxxx    >LDAA #12     Зарежда числото $12 в акумулатор A
        86 12
0102    xxxxx    >ADDA #13     Прибавя $13 към акумулатора
        8B 13
0104    xxxxx    >STAA 1004    Визуализира резултата в двоичен вид със светодиодите
        B7 10 04               (PB7 PB6 PB5 PB4 PB3 PB2 PB1 PB0)
0107    xxxxx    >SWI          Прекъсва софтуерно изпълнението на програмата и показва
        3F                     съдържанието на регистрите (резултатът е в акумулатор A)
```

Вътрешна (неявна) адресация. Устройствата (само регистри на процесора), които съдържат операнда, се адресират неявно чрез самия код на операцията (т.е., те се подразбират от кода на операцията)**.** Поради това командата няма поле за операнд.

$1000	КОД

Пример: Събиране на съдържанията на двата акумулатора.

```
0100    xxxxx    >LDAA #12     Зарежда числото $12 в акумулатор A
        86 12
0102    xxxxx    > LDAB #13    Зарежда числото $13 в акумулатор B
        C6 13
0104    xxxxx    >ABA          Събира съдържанията на двата акумулатора
        1B
0105    xxxxx    >STAA 1004    Визуализира резултата в двоичен код
        B7 10 04
0108    xxxxx    >SWI          Прекъсва софтуерно изпълнението на програмата и показва
        3F                     съдържанието на регистрите (резултатът е в акумулатор A)
```

Забележка: Числата може да се зареждат предварително в акумулаторите с командата RM на монитора и програмата да се стартира направо от адрес $104. Постарайте се сумата им да не надвишава 255. Защо? Как можете да проверите дали това условие е спазено?

Директна адресация. В полето на операнда е разположен непълният адрес (само младшата част A_L, старшата е нула) на устройствата (клетки от паметта, регистри, портове)**.** Това е абсолютен адрес в началната област от 256 байта $00**00**-00**FF** на адресното пространство.

$1000	КОД
$1001	A_L (адрес-мл.част)

Пример: Събиране на две числа, записани в клетки $20 и $21 от паметта и записване на резултата в клетка $22:

```
0100    xxxxx    >LDAA 20      Зарежда числото от клетка с адрес $0020 в акумулатор A
        96 20
0102    xxxxx    >ADDA 21      Прибавя числото от клетка с адрес $0021 към акумулатор A
        9B 21
0104    xxxxx    >STAA 22      Записва резултата в клетка с адрес $0022
        97 22
0106    xxxxx    >STAA 1004    Визуализира резултата в двоичен код
        B7 10 04
0109    xxxxx    >SWI          Прекъсва софтуерно изпълнението на програмата и показва
        3F                     съдържанието на регистрите (резултатът е в акумулатор A)
```

Разширена адресация. В полето на операнда е разположен пълният адрес (старша -> младша част) на данните**.** С него може да се избере всяко едно устройство, разположено в пълното адресно пространство ($0000-FFFF).

Пример: Събиране на две числа, записани в клетки $120 и $121 от паметта и записване на резултата в клетка $122:

$1000	PRE
$1001	КОД
$1002	A_H (адрес-ст.част)
$1004	A_L (адрес-мл.част)

0100	xxxxx B6 01 20	>**LDAA 120**	Зарежда числото от клетка с адрес $0120 в акумулатор А
0103	xxxxx BB 01 21	>**ADDA 121**	Прибавя числото от клетка с адрес $0021 към А
0106	xxxxx B7 01 22	>**STAA 122**	Записва резултата в клетка с адрес $0122
0109	xxxxx B7 10 04	**>STAA 1004**	Визуализира резултата в двоичен код (в клетка с адрес $1004)
010C	xxxxx 3F	>SWI	Прекъсва софтуерно изпълнението на програмата и показва съдържанието на регистрите (резултатът е в акумулатор А)

<u>Относителна адресация</u> - в полето на операнда (един байт) е разположено отместване (число със знак в допълнителен код /ДК/ в диапазона от -128 до +127). То се добавя към съдържанието на програмния брояч след изпълнението на инструкцията (т.е., към адреса на следващата я инструкция)**.** Относителната адресация се използва в командите за преход, където позволява адресиране в ограничен диапазон от 256 байта.

$1000	КОД
$1001	±B (отместване-число със знак в ДК)

Пример: Събиране на пет числа (едномерен масив), записани в последователни клетки $10, $11, $12, $13 и $14 от паметта и записване на резултата в клетка $15. Инструкцията **BNE 103** на адрес $109 е с относителна адресация, защото вграденият в монитора асемблер заменя абсолютния дрес $103 с относителен (отместването $F8 в допълнителен код, което се добавя към началния адрес $10B на следващата инструкция). Това спестява необходимостта да се изчислява отместването.

✓ **Понятие за цикъл в програма**

0100	xxxxx C6 05	>LDAB #5	Зарежда броя елементи (5) в акумулатор В
0102	xxxxx 4F	>CLRA	Нулира началната сума, разположена в акумулатор А
0103	xxxxx 9B 10	>ADDA 10	Добавя първия (поредния) елемент към текущата сума в акумулатор А, започвайки от стойността на клетката с адрес $0010
0105	xxxxx 7C 01 04	>INC 104	Подготвя избора на следващия елемент като увеличава с единица текущия адрес (втория байт на командатата ADDA, намиращ се на адрес $0104)
0108	xxxxx 5A	>DECB	Отброява още един елемент
0109	xxxxx 26 F8	>**BNE 103**	Продължава да сумира елементи, ако не са изчерпани. Асемблерът заменя абсолютния адрес $0103 с относителен (отместването $F8, записано в допълнителен код във втория байт на инструкцията)
010B	xxxxx 97 15	>STAA 15	Записва резултата в клетка с адрес $0015
010D	xxxxx B7 10 04	>STAA 1004	Визуализира резултата в двоичен код
0110	xxxxx 3F	>SWI	Прекъсва софтуерно изпълнението на програмата и показва съдържанието на регистрите (резултатът е в акумулатор А)

Съвет: Вмъкнете точка на прекъсване вътре в цикъла и наблюдавайте как се променят акумулаторите.

Въпроси: Защо при повторно изпълнение програмата не работи правилно? Какво трябва да се направи за да се реши този проблем? Какъв недостатък има този начин за обработка на масиви?

Индексна адресация. Подобно на относителната адресация, в полето на операнда (един байт) е разположено отместване, но тук то е число без знак в диапазона от 0 до 255). То се добавя към съдържанието на индексния регистър и този ефективен адрес се използва за адресация на операнда. Индексната адресация се използва най-често за обработка на масиви, където позволява адресиране в диапазон от 64K байта.

Пример (аналогичен на горния): Събиране на пет числа (едномерен масив), записани в последователни клетки $150, $151, $152, $153 и $154 от паметта и записване на резултата в клетка $155:

0100	xxxxx CE 01 50	>LDX #150	Зарежда адреса ($0150) на първия елемент в индексния регистър X
0103	xxxxx C6 05	>LDAB #5	Зарежда броя елементи (5) в акумулатор B
0105	xxxxx 4F	>CLRA	Нулира началната сума, разположена в акумулатор A
0106	xxxxx AB 00	>**ADDA 0,X**	Сумира поредния елемент с адрес, разположен в индексния регистър, с числото, намиращо се в акумулатор A и го записва пак там (отместването не се използва)
0108	xxxxx 08	>INX	Подготвя избора на следващия елемент - увеличава съдържанието на индексния регистър X с единица, с което подготвя избора на следващия адрес
0109	xxxxx 5A	>DECB	Отброява още един елемент
010A	xxxxx 26 FA	>BNE 0106	Продължава да сумира елементи, ако не са изчерпани. $10C - $106 = -$06 ($FA в допълнителен код)
010C	xxxxx 97 01 55	>STAA 155	Записва резултата в клетка с адрес $0155
010F	xxxxx B7 10 04	>STAA 1004	Визуализира резултата в двоичен код
0112	xxxxx 3F	>SWI	Прекъсва софтуерно изпълнението на програмата и показва съдържанието на регистрите (резултатът е в акумулатор A)

Съвет: Вмъкнете точка на прекъсване вътре в цикъла и наблюдавайте как се променят акумулаторите и индексния регистър.

Въпрос: Какви предимства има този начин за обработка на масиви спрямо предишния?

- **Обобщение на адресациите:** Непосредствената адресация указва данните, директната и разширената – абсолютния адрес на данните, относителната – промяната на адреса спрямо текущия, индексната – косвения адрес (зареден предварително в съответния индексен регистър). Индексната адресация в HC11 всъщност е една по-сложна, непряка адресация), която включва в себе си относителна адресация (отместването в командата), неявна адресация (вида на индексния регистър – X или Y) и самата индексна адресация (адреса, записан в индексния регистър).

3. ПРОГРАМИРАНЕ НА АСЕМБЛЕР AS 11

3.1. Директиви на асемблера.

3.2. Асемблиране и изпълнение на програма в средата *AsmIDE*.

Стъпка 1: **Въвеждане и редактиране**

Пример (аналогичен на предишния): Събиране на пет числа (едномерен масив), записани в последователни клетки с начален адрес BEGA и записване на резултата в клетка с адрес RESULT.

Текст на програмата:

```
* SumArr1B.asm --- Сумира елементите (1-байтови) на масив ---
*
* Сумата се намира в клетка RESULT

NUMBER    EQU   5                  ; Задава броя елементи на масива

          ORG   $100
* Резервира памет за масива от 1-байтови елементи
BEGA      RMB   NUMBER
RESULT    RMB   1

          ORG   $D000
* Задава начални стойности на променливите
          LDAB  # NUMBER           ; Зарежда броя елементи (5) в акумулатор B
          CLRA                     ; Нулира началната сума в  акумулатор  A
          LDX   #BEGA              ; Зарежда адреса ($150) на първия елемент в индексния
                                   ; регистър X
* Сумира елементите на масива
SUM       ADDA  0,X                ; Сумира поредния елемент
          INX                      ; Подготвя  избора на следващия  елемент
          DECB
          BNE   SUM                ; Край на масива?
          STAA  RESULT             ; Да, записва резултата в клетка RESULT
          SWI
```

Стъпка 2: **Асемблиране (създаване на обектен код)**

Листинг на програмата след асемблирането:

```
AS11, an absolute assembler for Motorola MCU's, version 1.2e

                    * SumArr1B.asm --- Сумира елементите (1-байтови) на масив ---
                    * Сумата се намира в клетка RESULT

0005                NUMBER    EQU   5       ; Задава броя елементи на масива

0100                          ORG   $100
                  * Резервира памет за масива от 1-байтови елементи
0100                BEGA      RMB   NUMBER
```

```
0105            RESULT  RMB   1

d000                    ORG     $D000
                * Задава начални стойности на променливите
d000 c6 05              LDAB    #NUMBER    ; Зарежда броя елементи (5) в акумулатор B
d002 4f                 CLRA               ;  Нулира началната сума в  акумулатор  A
d003 ce 01 00           LDX     #BEGA      ; Зарежда адреса ($150) на първия елемент
                *                          ; в индексния регистър
                * Сумира елементите на масива
d006 ab 00      SUM     ADDA    0,X        ; Сумира поредния елемент
d008 08                 INX                ; Подготвя  избора на следващия  елемент
d009 5a                 DECB
d00a 26 fa              BNE     SUM        ; Край на масива?
d00c b7 01 05           STAA    RESULT     ; Да, записва резултата в клетка RESULT
d00f 3f                 SWI
```

Executed: Fri Mar 09 23:09:26 2012
Total cycles: 37, Total bytes: 16
Total errors: 0, Total warnings: 0

***Стъпка 3*: Зареждане в развойната система**

***Стъпка 4*: Изпълнение**

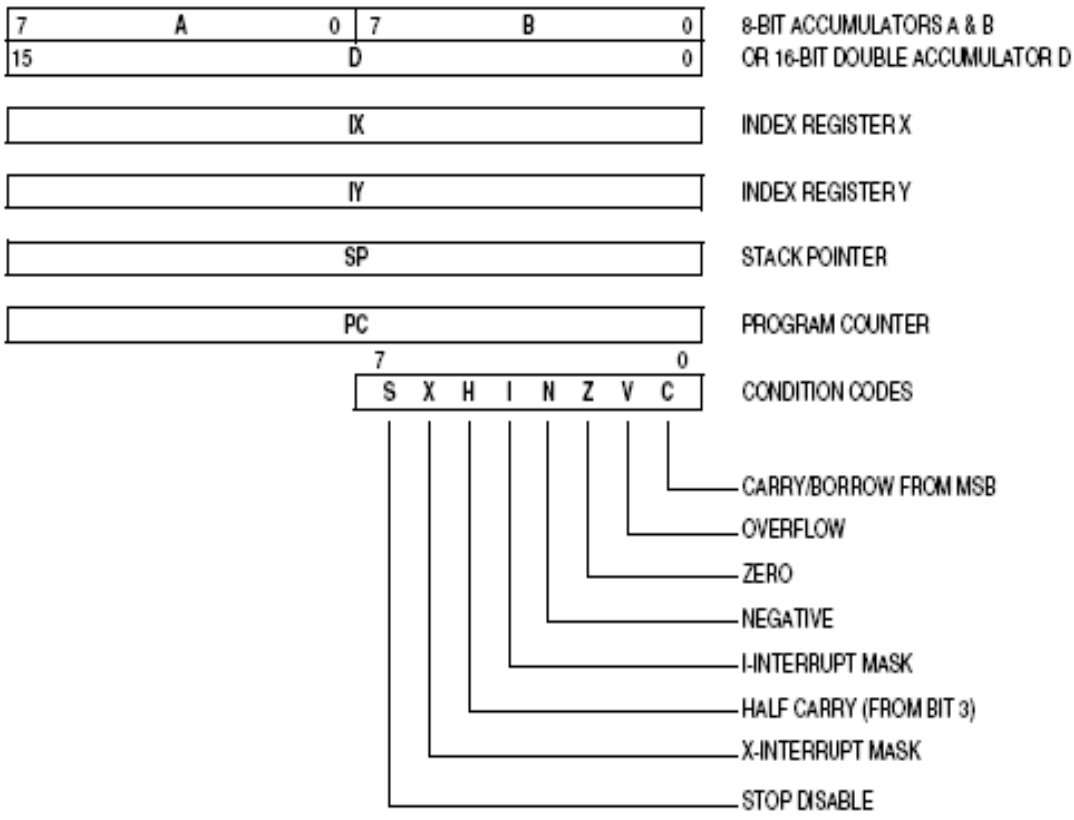
Съвет: Вмъкнете точка на прекъсване вътре в цикъла и наблюдавайте как се променят регистрите.

II. ЗАДАЧИ ЗА ИЗПЪЛНЕНИЕ:

1. Разгледайте системните регистри на HC11 – Приложение 1. Колко индексни регистъра има EMK 68HC11? За какво служат отделните битове в регистъра на състоянието (контролния регистър) CCR? Как можете да ги използвате в примерните програми?
2. Разучете структурата на инструкциите при отделните видове адресации. Какви са особеностите и предназначението на всяка адресация? Каква е разликата между относителната и индексната адресация?
3. Изпълнете примерите, представени по-горе в заданието в средата AsmIDE чрез вградения в монитора поредов асемблер.
   3.1 Изпълнете примера за вътрешна адресация, като вместо числата $12 и $13, съберете числата $88 и $84 (сума $10C). Преобразувайте резултата в двоичен вид. Реален ли ще бъде резултата в акумулатор A след команда ABA? Кой флаг от регистъра CCR е указател за препълване на 8-битовата решетка за резултата? Проверете го за горния пример. Какво е съдържанието на 16-битовия акумулатор D преди команда ABA?
   3.2 Изпълнете някои от примерите за събиране, но за числа със знак: положително и по-малко (по абсолютна стойност) отрицателно число; положително и по-голямо (по абсолютна стойност) отрицателно число. Каква е ролята на флаговете V и C в този случай?

Приложение 1

Програмен модел на ЕМК 68HC11



ACC A (акумулатор A) - 8-битов;

ACC B (акумулатор B) - 8-битов или общо →16-битов акумулатор **ACC D**;

IX (индексен регистър X);

IY (индексен регистър Y);

SP (указател на стека);

PC (програмен брояч);

CCR (регистър на състоянието, 8-битов).

Приложение 2-А

Инструкции за работа с акумулаторите и паметта

- зареждане, изпращане и прехвърляне

Function	Mnemonic	IMM	DIR	EXT	INDX	INDY	INH
Clear Memory Byte	CLR			X	X	X	
Clear Accumulator A	CLRA						X
Clear Accumulator B	CLRB						X
Load Accumulator A	LDAA	X	X	X	X	X	
Load Accumulator B	LDAB	X	X	X	X	X	
Load Double Accumulator D	LDD	X	X	X	X	X	
Pull A from Stack	PULA						X
Pull B from Stack	PULB						X
Push A onto Stack	PSHA						X
Push B onto Stack	PSHB						X
Store Accumulator A	STAA	X	X	X	X	X	
Store Accumulator B	STAB	X	X	X	X	X	
Store Double Accumulator D	STD	X	X	X	X	X	
Transfer A to B	TAB						X
Transfer A to CCR	TAP						X
Transfer B to A	TBA						X
Transfer CCR to A	TPA						X
Exchange D with X	XGDX						X
Exchange D with Y	EGDY						X

-
- аритметични операции

Function	Mnemonic	IMM	DIR	EXT	INDX	INDY	INH
Add Accumulators	ABA						X
Add Accumulator B to X	ABX						X
Add Accumulator B to Y	ABY						X
Add with Carry to A	ADCA	X	X	X	X	X	
Add with Carry to B	ADCB	X	X	X	X	X	
Add Memory to A	ADDA	X	X	X	X	X	
Add Memory to B	ADDB	X	X	X	X	X	
Add Memory to D (16 Bit)	ADDD	X	X	X	X	X	
Compare A to B	CBA						X
Compare A to Memory	CMPA	X	X	X	X	X	
Compare B to Memory	CMPB	X	X	X	X	X	
Compare D to Memory (16 Bit)	CPD	X	X	X	X	X	
Decimal Adjust A (for BCD)	DAA						X
Decrement Memory Byte	DEC			X	X	X	
Decrement Accumulator A	DECA						X
Decrement Accumulator B	DECB						X
Increment Memory Byte	INC			X	X	X	
Increment Accumulator A	INCA						X
Increment Accumulator B	INCB						X

•

Function	Mnemonic	IMM	DIR	EXT	INDX	INDY	INH
Twos Complement Memory Byte	NEG			X	X	X	
Twos Complement Accumulator A	NEGA						X
Twos Complement Accumulator B	NEGB						X
Subtract with Carry from A	SBCA	X	X	X	X	X	
Subtract with Carry from B	SBCB	X	X	X	X	X	
Subtract Memory from A	SUBA	X	X	X	X	X	
Subtract Memory from B	SUBB	X	X	X	X	X	
Subtract Memory from D (16 Bit)	SUBD	X	X	X	X	X	
Test for Zero or Minus	TST			X	X	X	
Test for Zero or Minus A	TSTA						X
Test for Zero or Minus B	TSTB						X

- логически операции

Function	Mnemonic	IMM	DIR	EXT	INDX	INDY	INH
AND A with Memory	ANDA	X	X	X	X	X	
AND B with Memory	ANDB	X	X	X	X	X	
Bit(s) Test A with Memory	BITA	X	X	X	X	X	
Bit(s) Test B with Memory	BITB	X	X	X	X	X	
One's Complement Memory Byte	COM			X	X	X	
One's Complement A	COMA						X
One's Complement B	COMB						X
OR A with Memory (Exclusive)	EORA	X	X	X	X	X	
OR B with Memory (Exclusive)	EORB	X	X	X	X	X	
OR A with Memory (Inclusive)	ORAA	X	X	X	X	X	
OR B with Memory (Inclusive)	ORAB	X	X	X	X	X	

- побитова обработка

Function	Mnemonic	IMM	DIR	EXT	INDX	INDY
Bit(s) Test A with Memory	BITA	X	X	X	X	X
Bit(s) Test B with Memory	BITB	X	X	X	X	X
Clear Bit(s) in Memory	BCLR		X		X	X
Set Bit(s) in Memory	BSET		X		X	X
Branch if Bit(s) Clear	BRCLR		X		X	X
Branch if Bit(s) Set	BRSET		X		X	X

- преместване и ротация

Function	Mnemonic	IMM	DM	EXT	INDX	INDY	INH
Arithmetic Shift Left Memory	ASL			X	X	X	
Arithmetic Shift Left A	ASLA						X
Arithmetic Shift Left B	ASLB						X
Arithmetic Shift Left Double	ASLD						X
Arithmetic Shift Right Memory	ASR			X	X	X	
Arithmetic Shift Right A	ASRA						X
Arithmetic Shift Right B	ASRB						X
(Logical Shift Left Memory)	(LSL)			X	X	X	
(Logical Shift Left A)	(LSLA)						X
(Logical Shift Left B)	(LSLB)						X
(Logical Shift Left Double)	(LSLD)						X
Logical Shift Right Memory	LSR			X	X	X	
Logical Shift Right A	LSRA						X
Logical Shift Right B	LSRB						X
Logical Shift Right D	LSRD						X
Rotate Left Memory	ROL			X	X	X	
Rotate Left A	ROLA						X
Rotate Left B	ROLB						X
Rotate Right Memory	ROR			X	X	X	
Rotate Right A	RORA						X
Rotate Right B	RORB						X

Инструкции за стека и индексните регистри Приложение 2-В

Function	Mnemonic	IMM	DIR	EXT	INDX	INDY	INH
Add Accumulator B to X	ABX						X
Add Accumulator B to Y	ABY						X
Compare X to Memory (16 Bit)	CPX	X	X	X	X	X	
Compare Y to Memory (16 Bit)	CPY	X	X	X	X	X	
Decrement Stack Pointer	DES						X
Decrement Index Register X	DEX						X
Decrement Index Register Y	DEY						X
Increment Stack Pointer	INS						X
Increment Index Register X	INX						X
Increment Index Register Y	INY						X
Load Index Register X	LDX	X	X	X	X	X	
Load Index Register Y	LDY	X	X	X	X	X	
Load Stack Pointer	LDS	X	X	X	X	X	
Pull X from Stack	PULX						X
Pull Y from Stack	PULY						X
Push X onto Stack	PSHX						X
Push Y onto Stack	PSHY						X
Store Index Register X	STX	X	X	X	X	X	
Store Index Register Y	STY	X	X	X	X	X	
Store Stack Pointer	STS	X	X	X	X	X	
Transfer SP to X	TSX						X
Transfer SP to Y	TSY						X
Transfer X to SP	TXS						X
Transfer Y to SP	TYS						X
Exchange D with X	XGDX						X
Exchange D with Y	XGDY						X

Инструкции за регистъра на състоянието

Приложение 2-C

Function	Mnemonic	INH
Clear Carry Bit	CLC	X
Clear Interrupt Mask Bit	CLI	X
Clear Overflow Bit	CLV	X
Set Carry Bit	SEC	X
Set Interrupt Mask Bit	SEI	X
Set Overflow Bit	SEV	X
Transfer A to CCR	TAP	X
Transfer CCR to A	TPA	X

Инструкции за управление на програмата

Приложение 2-D

- преходи

Function	Mnemonic	REL	DIR	INDX	INDY	Comments
Branch if Carry Clear	BCC	X				C = 0 ?
Branch if Carry Set	BCS	X				C = 1 ?
Branch if Equal Zero	BEQ	X				Z = 1 ?
Branch if Greater Than or Equal	BGE	X				Signed ≥
Branch if Greater Than	BGT	X				Signed >
Branch if Higher	BHI	X				Unsigned >
Branch if Higher or Same (same as BCC)	BHS	X				Unsigned ≥
Branch if Less Than or Equal	BLE	X				Signed ≤
Branch if Lower (same as BCS)	BLO	X				Unsigned <
Branch if Lower or Same	BLS	X				Unsigned ≤
Branch if Less Than	BLT	X				Signed <
Branch if Minus	BMI	X				N = 1 ?
Branch if Not Equal	BNE	X				Z = 0 ?
Branch if Plus	BPL	X				N = 0 ?
Branch if Bit(s) Clear in Memory Byte	BRCLR		X	X	X	Bit Manipulation
Branch Never	BRN	X				3-cycle NOP
Branch if Bit(s) Set in Memory Byte	BRSET		X	X	X	Bit Manipulation
Branch if Overflow Clear	BVC	X				V = 0 ?
Branch if Overflow Set	BVS	X				V = 1 ?
Jump	JMP	X	X	X	X	

- подпрограми

Function	Mnemonic	REL	DIR	EXT	INDX	INDY	INH
Branch to Subroutine	BSR	X					
Jump to Subroutine	JSR		X	X	X	X	
Return from Subroutine	RTS						X

- прекъсвания

Function	Mnemonic	INH
Return from Interrupt	RTI	X
Software Interrupt	SWI	X
Wait for Interrupt	WAI	X