Міністерство освіти і науки, молоді та спорту України
Державний вищий навчальний заклад
„НАЦІОНАЛЬНИЙ ГІРНИЧИЙ УНІВЕРСИТЕТ“

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра автоматизації та комп’ютерних систем

РОБОЧИЙ ЗОШИТ

до конспекту лекцій з дисциплін
“Основи побудови мікропроцесорних систем керування”, “Мікропроцесорна техніка”,
“Програмні засоби систем керування”
для студентів спеціальностей
АГ – 8.092501 Автоматизоване управління технологічними процесами
АТ,МЕ – 8.091401 Системи управління й автоматики
СМ – 8.091501 Комп`ютерні системи та мережі

Частина перша

Дніпропетровськ
НГУ
2013

Робочий зошит до конспекту лекцій з дисциплін “Основи побудови мікропроцесорних систем керування”, “Мікропроцесорна техніка”, “Програмні засоби систем керування” для студентів спеціальностей АГ – 8.092501 Автоматизоване управління технологічними процесами; АТ,МЕ – 8.091401 Системи управління й автоматики; СМ – 8.091501 Комп`ютерні системи та мережі / В.В. Ткачов, М.В. Козар, В.І. Шевченко та ін. – Д.: Національний гірничий університет, 2013. – 122 с.

Автори: В.В. Ткачов, д-р техн. наук, проф.
М.В. Козар, асист.
В.І. Шевченко, асист.
С.М. Проценко, старш. викл.
О.В. Карпенко, асист.
В.В. Надточий, старш. викл
М.О. Ткачук, асист.

Затверджено редакційною радою ДВНЗ «Національний гірничий університет» (протокол № 1 від 21 січня 2013).

Відповідальний за випуск завідувач кафедри автоматизації та комп`ютерних систем В.В. Ткачов, д-р техн. наук, проф.

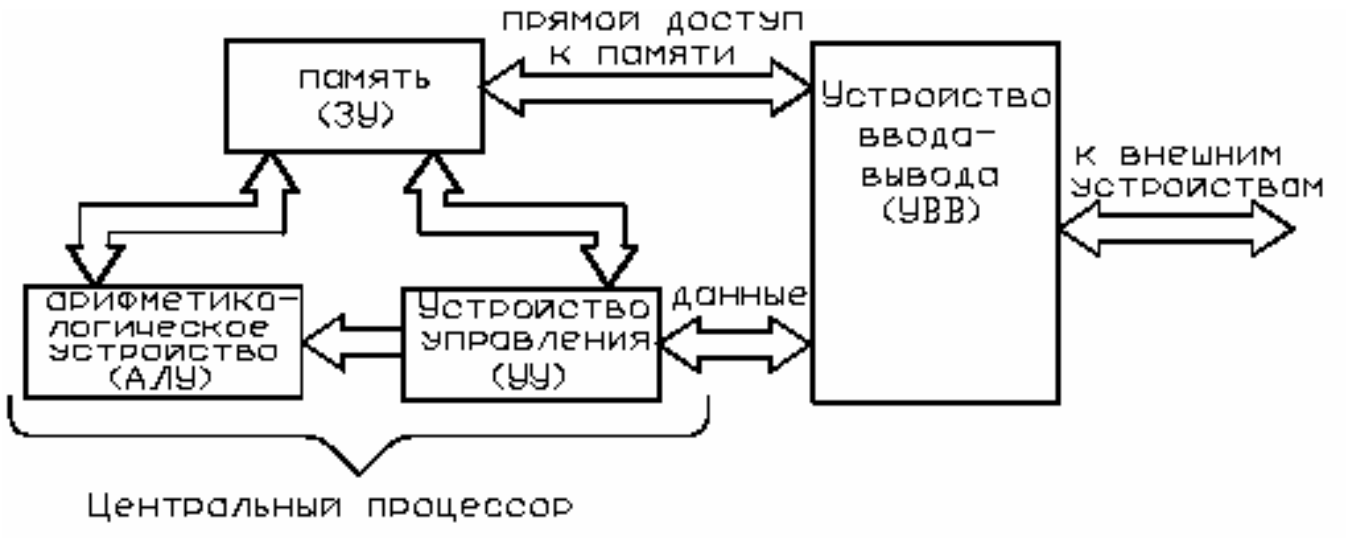
Дані методичні вказівки призначуються для вивчення апаратних та програмних засобів мікропроцесорних систем управління (МПС).

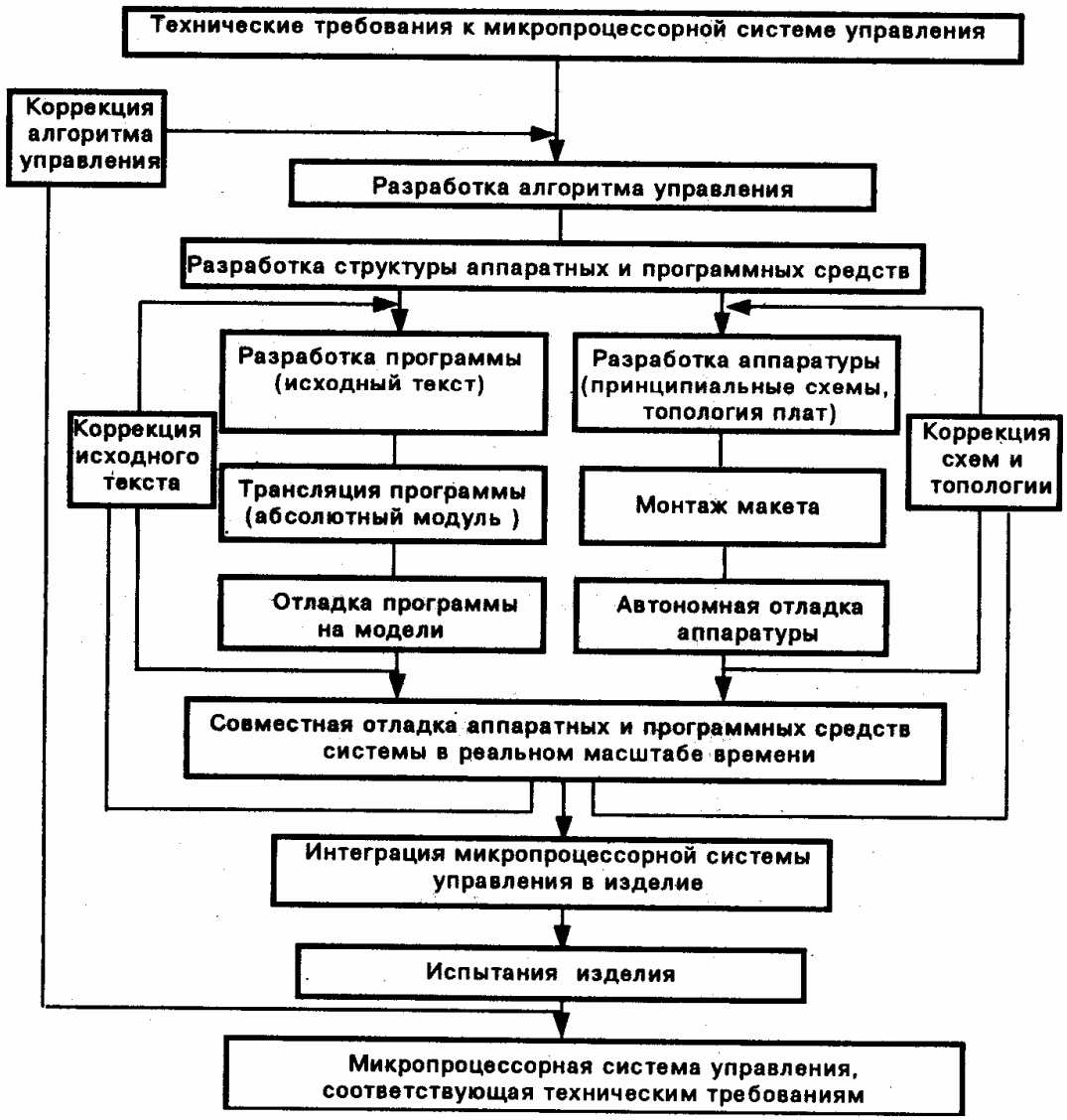
У процесі вивчення дисципліни "Мікропроцесорні системи" студенти вивчають структуру, архітектуру, сигнали та системи команд однокристального мікроконтролера К1816 ВЕ51.

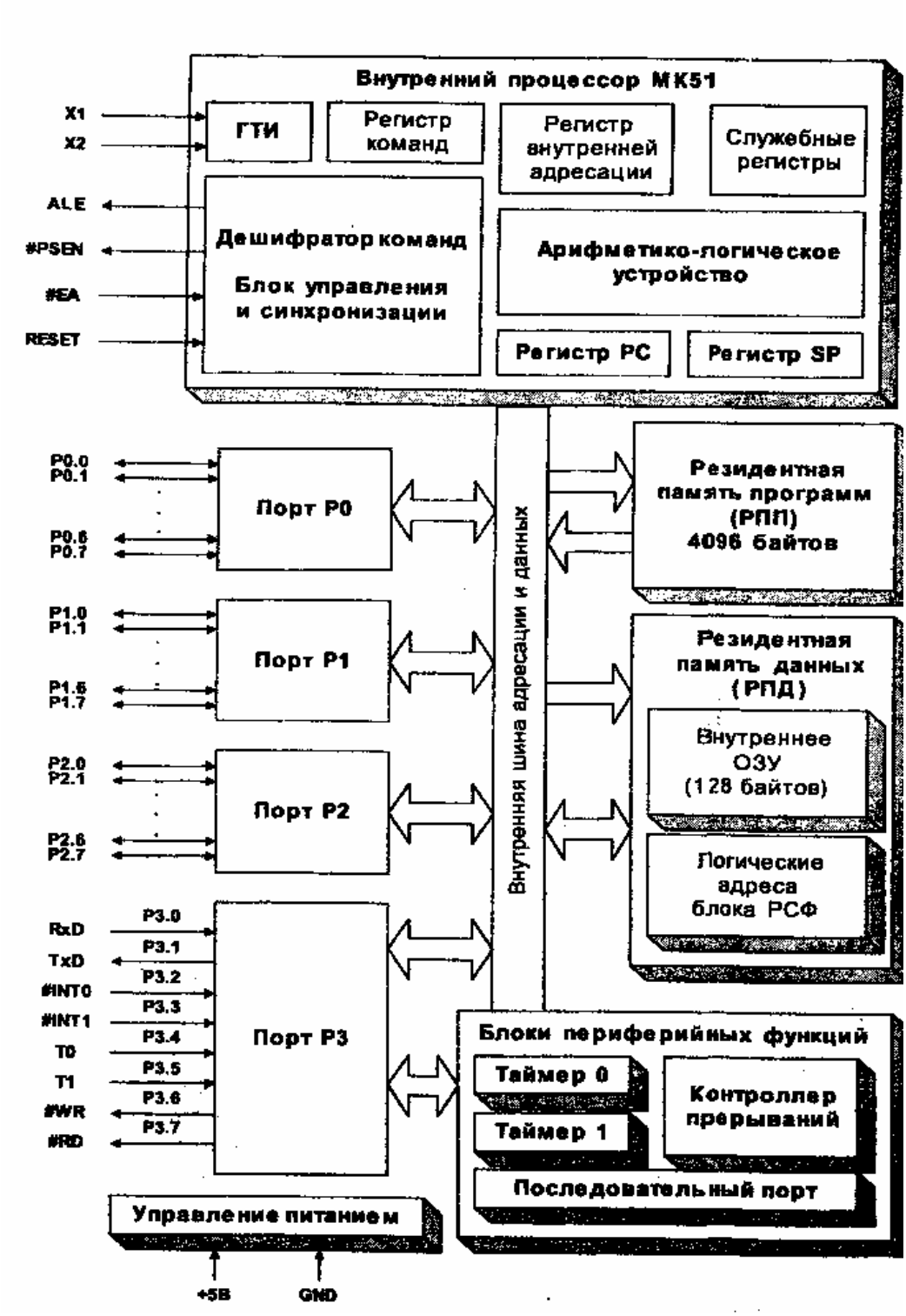
Розглянуті питання організації паралельного та послідовного вводу-виводу, оганізація мікропроцесорних контролерів МПК).

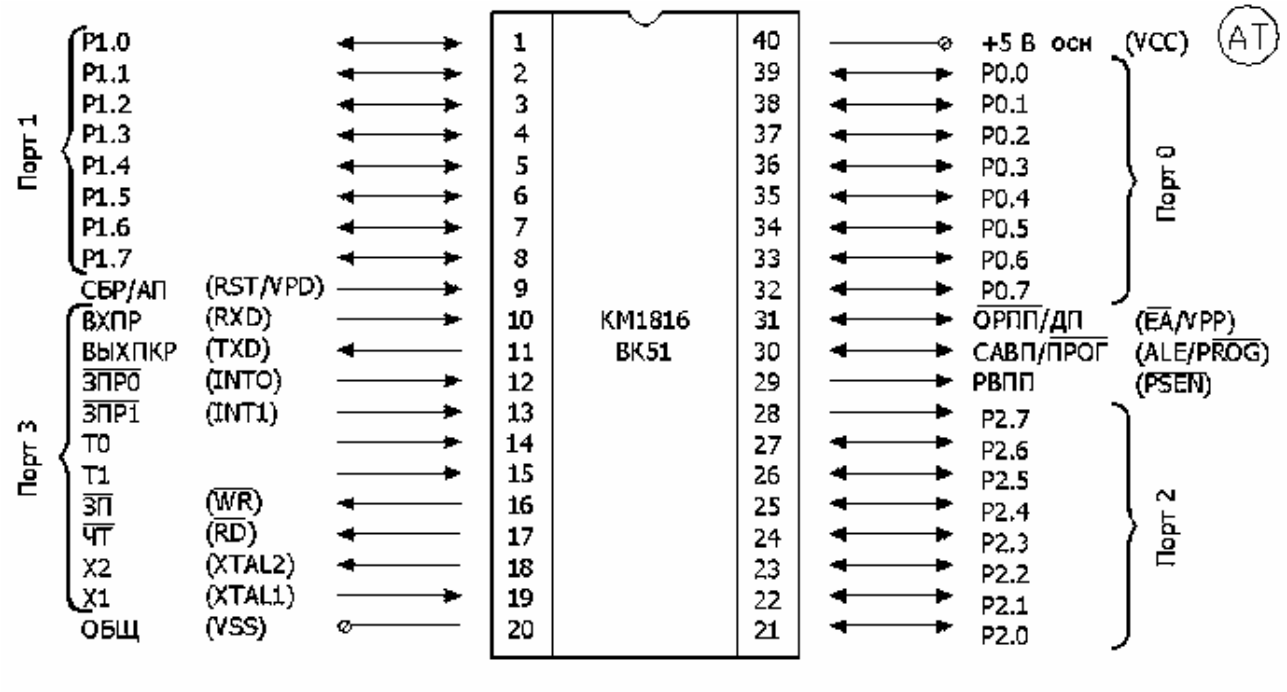
Схемні рішення, які приведені у методичних вказівках можуть бути використані при виконанні курсових і дипломних проектів студентами спеціальностей "Комп’ютеризовані системи управління і автоматики" (АТ) і "Автоматизація технологічних процесів гірничих підприємств" (АГ) та "Комп’ютерні системи та мережі" (СМ).

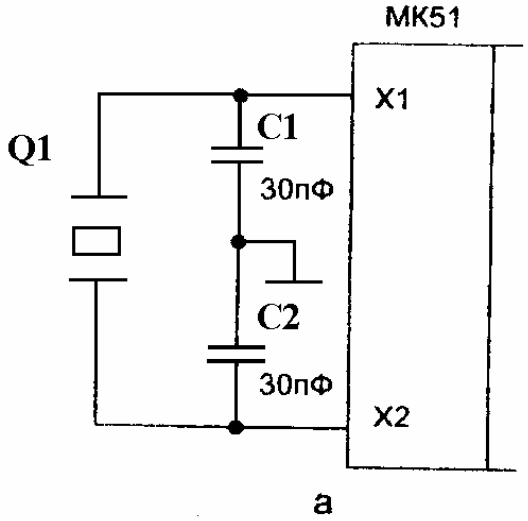

Технологический датчик № 1
Технологічский датчик № 2
Технологический датчик № n
Объект управления
Исполнительный механизм № 1
Исполнительный механизм №,2
Исполнительный механизм № n
$Y_1 = \overline{X_1} X_2 \overline{X_3} \vee \overline{X_1} X_3 \overline{X_2} \ldots$
(Логическая функция управления)
$Y_2 = X_1 X_2 \overline{X_3} \vee \ldots$
(Логическая функция управления)
$Y_3 = K_1 \cdot X_{вх} + K_2 \cdot \int X_{вх} \cdot dt + K_3 \cdot \frac{dx_{вх}}{dt}$
(Регуляторы)
Устройство управления

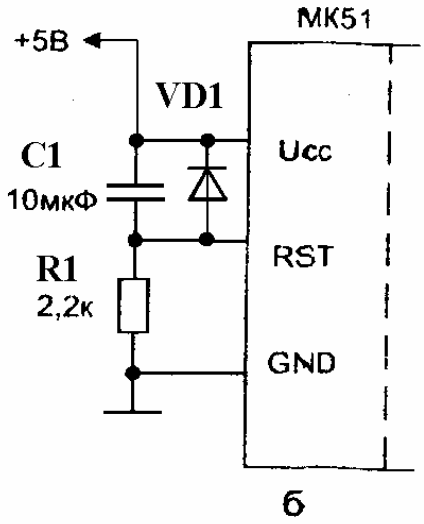

прямой доступ к памяти
память (ЗУ)
Устройство ввода-вывода (УВВ)
к внешним устройствам
арифметико-логическое устройство (АЛУ)
Устройство управления (УУ)
данные
Центральный процессор

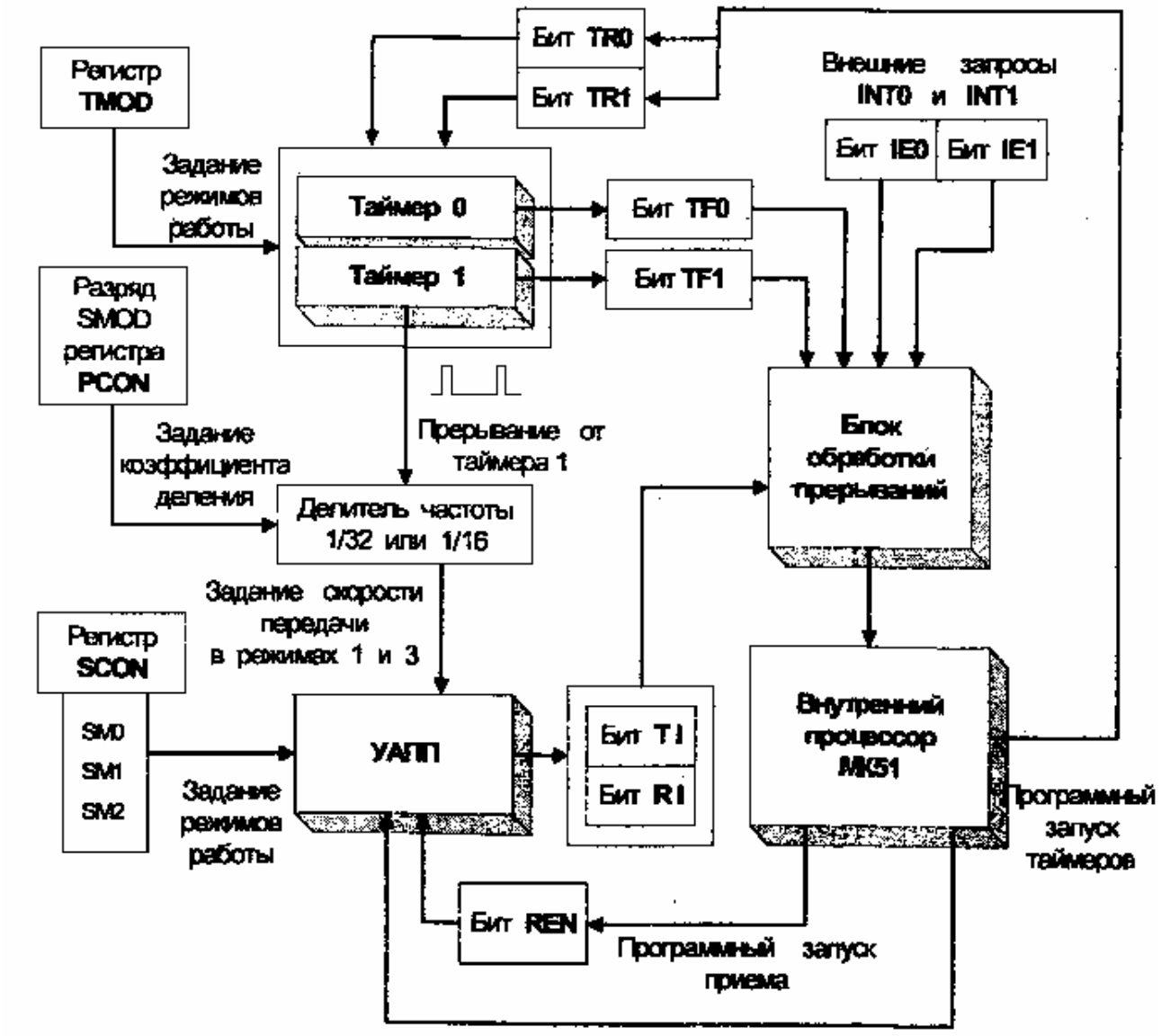

Технические требования к микропроцессорной системе управления
Коррекция алгоритма управления
Разработка алгоритма управления
Разработка структуры аппаратных и программных средств
Разработка программы (исходный текст)
Разработка аппаратуры (принципиальные схемы, топология плат)
Коррекция исходного текста
Коррекция схем и топологии
Трансляция программы (абсолютный модуль)
Монтаж макета
Отладка программы на модели
Автономная отладка аппаратуры
Совместная отладка аппаратных и программных средств системы в реальном масштабе времени
Интеграция микропроцессорной системы управления в изделие
Испытания изделия
Микропроцессорная система управления, соответствующая техническим требованиям

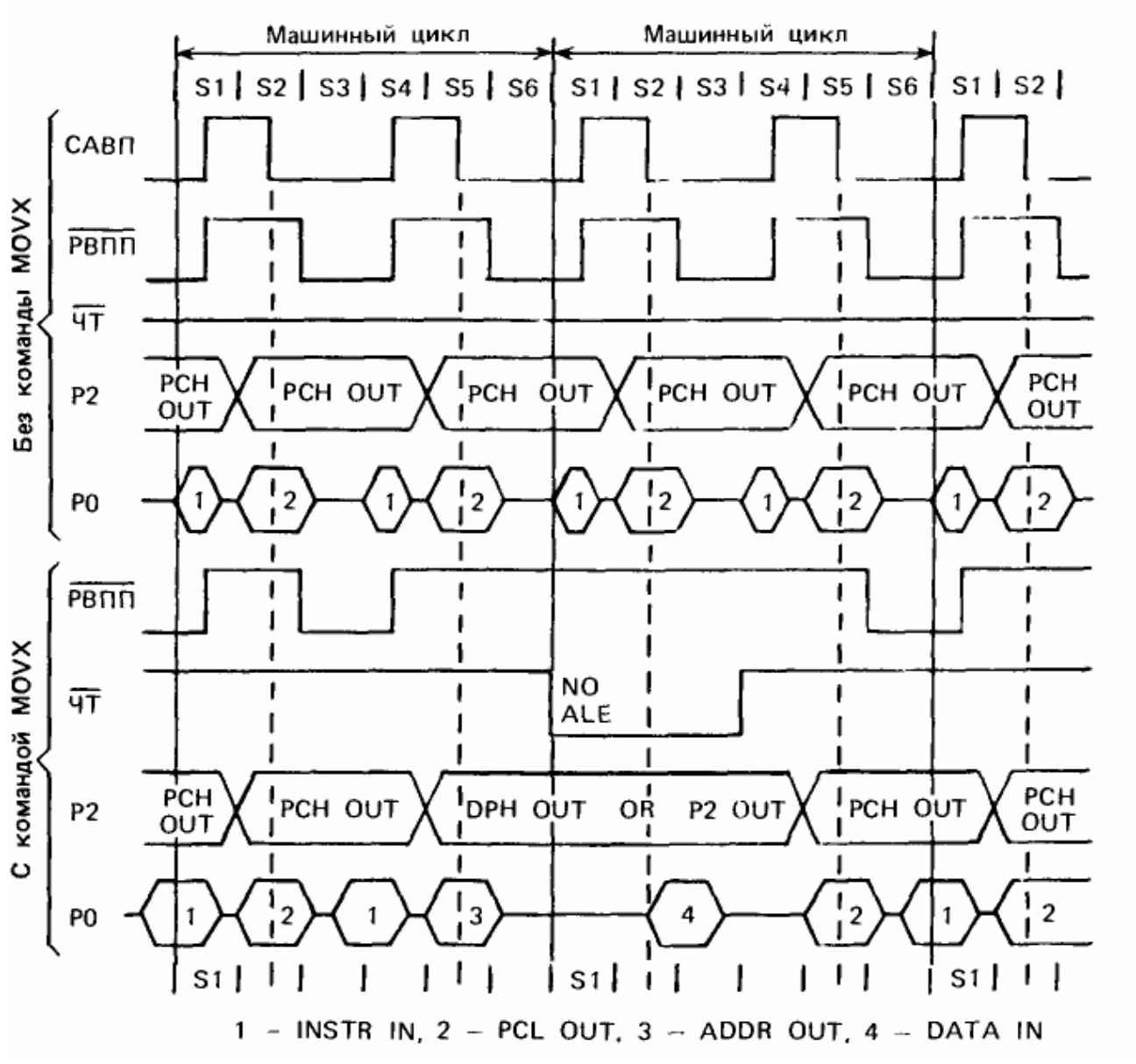

Внутренний процессор МК51
X1
X2
ГТИ
Регистр команд
Регистр внутренней адресации
Служебные регистры
ALE
#PSEN
#EA
RESET
Дешифратор команд
Блок управления и синхронизации
Арифметико-логическое устройство
Регистр PC
Регистр SP
P0.0
P0.1
P0.6
P0.7
Порт P0
P1.0
P1.1
P1.6
P1.7
Порт P1
P2.0
P2.1
P2.6
P2.7
Порт P2
RxD
TxD
#INT0
#INT1
T0
T1
#WR
#RD
P3.0
P3.1
P3.2
P3.3
P3.4
P3.5
P3.6
P3.7
Порт P3
Внутренняя шина адресации и данных
Резидентная память программ (РПП) 4096 байтов
Резидентная память данных (РПД)
Внутреннее ОЗУ (128 байтов)
Логические адреса блока РСФ
Блоки периферийных функций
Таймер 0
Таймер 1
Контроллер прерываний
Последовательный порт
Управление питанием
+5В
GND

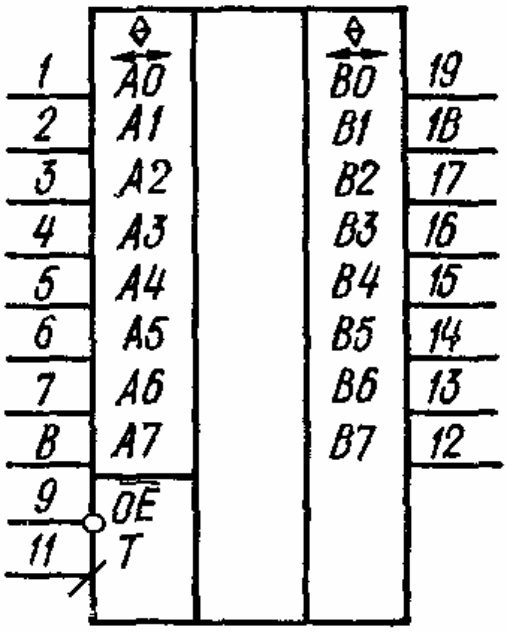

Порт 1
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
СБР/АП (RST/VPD)
Порт 3
ВХПР (RXD)
ВЫХПКР (TXD)
ЗПР0 (INT0)
ЗПР1 (INT1)
T0
T1
ЗП (WR)
ЧТ (RD)
X2 (XTAL2)
X1 (XTAL1)
ОБЩ (VSS)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
КМ1816
ВК51
40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21
+5 В осн (VCC)
P0.0
P0.1
P0.2
P0.3
P0.4
P0.5
P0.6
P0.7
Порт 0
ОРПП/ДП (EA/VPP)
САВП/ПРОГ (ALE/PROG)
РВПП (PSEN)
P2.7
P2.6
P2.5
P2.4
P2.3
P2.2
P2.1
P2.0
Порт 2
AT

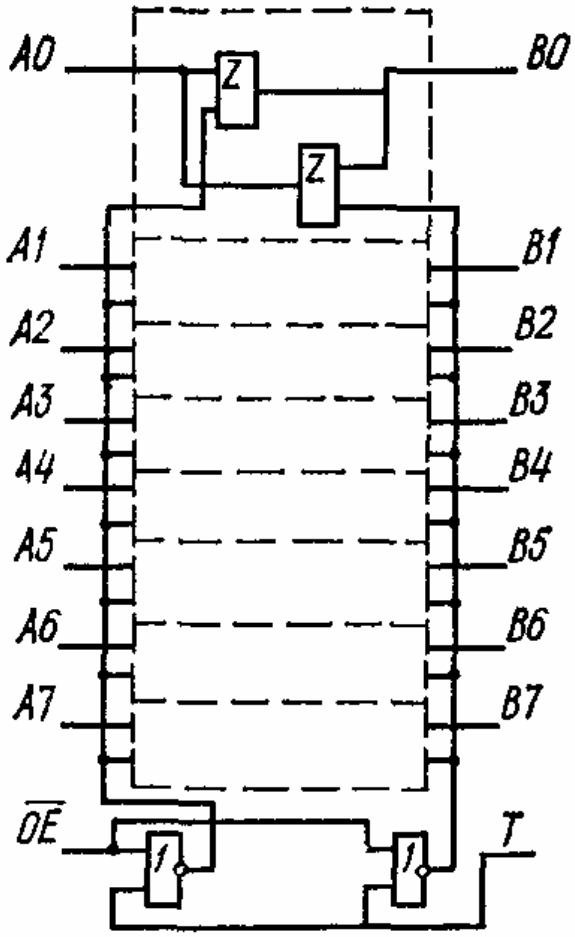

МК51
Q1
C1
30пФ
C2
30пФ
X1
X2
а

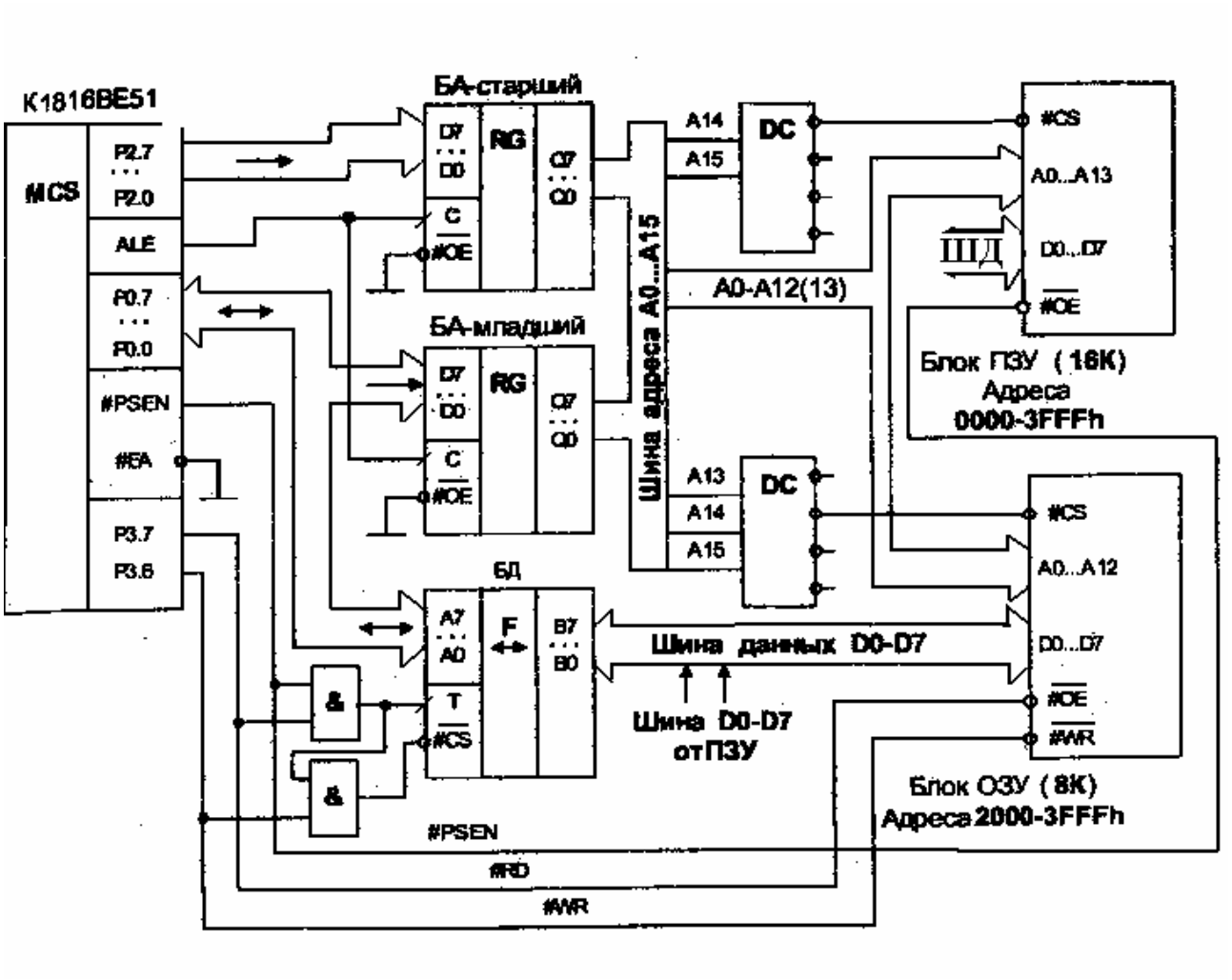

МК51
+5В
VD1
C1
10мкФ
R1
2,2к
Ucc
RST
GND
б

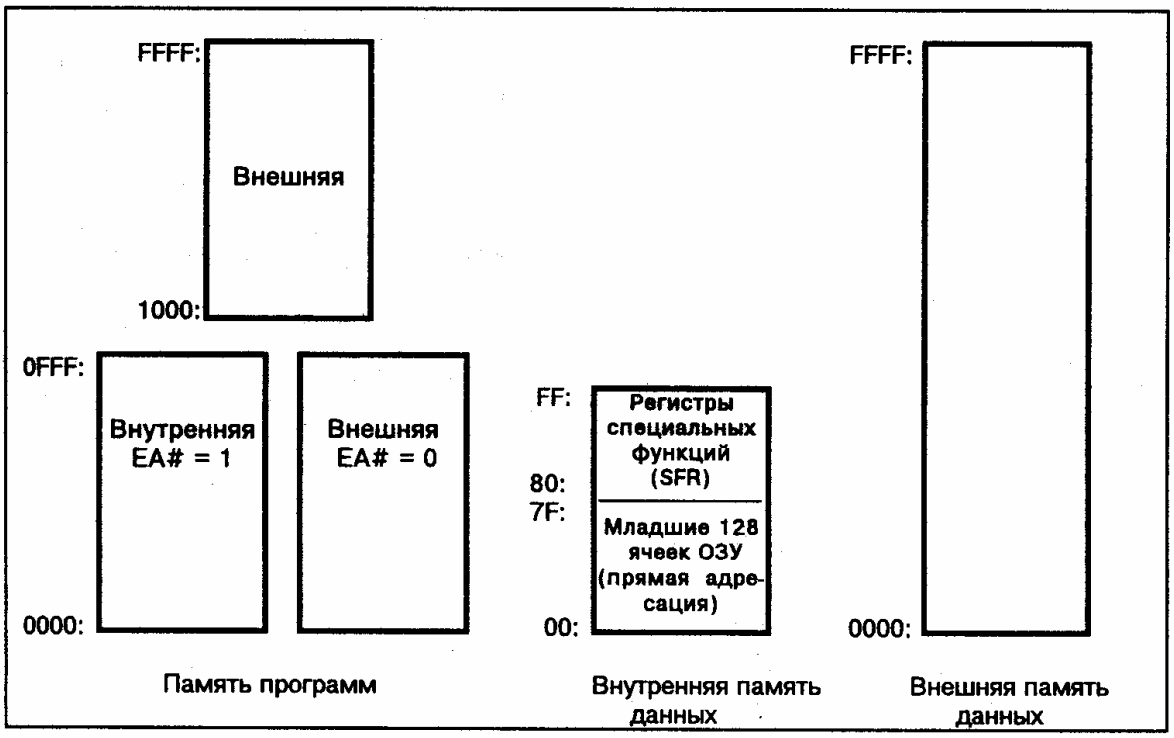

Регистр TMOD
Бит TR0
Бит TR1
Внешние запросы INT0 и INT1
Бит IE0
Бит IE1
Задание режимов работы
Таймер 0
Таймер 1
Бит TF0
Бит TF1
Разряд SMOD регистра PCON
Прерывание от таймера 1
Блок обработки прерываний
Задание коэффициента деления
Делитель частоты 1/32 или 1/16
Задание скорости передачи в режимах 1 и 3
Регистр SCON
SM0
SM1
SM2
УАПП
Бит TI
Бит RI
Внутренний процессор MK51
Программный запуск таймеров
Задание режимов работы
Бит REN
Программный запуск приема


Машинный цикл
Машинный цикл
S1 S2 S3 S4 S5 S6 S1 S2 S3 S4 S5 S6 S1 S2
Без команды MOVX
САВП
РВПП
ЧТ
P2
PCH OUT
PCH OUT
PCH OUT
PCH OUT
PCH OUT
PCH OUT
P0
С командой MOVX
РВПП
ЧТ
NO ALE
P2
PCH OUT
PCH OUT
DPH OUT OR P2 OUT
PCH OUT
PCH OUT
P0
S1 S1 S1
1 – INSTR IN, 2 – PCL OUT, 3 – ADDR OUT, 4 – DATA IN


1
2
3
4
5
6
7
8
9
11
A0
A1
A2
A3
A4
A5
A6
A7
OE
T
B0
B1
B2
B3
B4
B5
B6
B7
19
18
17
16
15
14
13
12


A0
A1
A2
A3
A4
A5
A6
A7
OE
T
B0
B1
B2
B3
B4
B5
B6
B7
1
1


К1816ВЕ51
#CS
P2.7
P2.0
ALE
P0.7
P0.0
#PSEN
#EA
P3.7
P3.6
БА-старший
D7
D0
RG
Q7
Q0
C
#OE
БА-младший
БД
A7
A0
F
B7
B0
T
#CS
&
&
Шина адреса A0...A15
A14
A15
DC
A0-A12(13)
A13
A14
A15
DC
ШД
#CS
A0...A13
D0...D7
#OE
Блок ПЗУ (16К)
Адреса
0000-3FFFh
#CS
A0...A12
D0...D7
#OE
#WR
Блок ОЗУ (8К)
Адреса 2000-3FFFh
Шина данных D0-D7
Шина D0-D7
отПЗУ
#PSEN
#RD
#WR


FFFF:
Внешняя
1000:
0FFF:
Внутренняя
EA# = 1
0000:
Внешняя
EA# = 0
Память программ
FF:
Регистры специальных функций (SFR)
80:
7F:
Младшие 128 ячеек ОЗУ (прямая адресация)
00:
Внутренняя память данных
FFFF:
0000:
Внешняя память данных

B (F0H)							
A.7	A.6	A.5	A.4	A.3	A.2	A.1	A.0
E7H	E6H	E5H	E4H	E3H	E2H	E1H	E0H

A (E0H)							
A.7	A.6	A.5	A.4	A.3	A.2	A.1	A.0
E7H	E6H	E5H	E4H	E3H	E2H	E1H	E0H

PSW (D0H)							
CY	AC	F0	RS1	RS0	OV	---	P
D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H

P3 (B0H)							
RD	WR	T1	T0	INT1	INT0	TxD	RxD
B7H	B6H	B5H	B4H	B3H	B2H	B1H	B0H

P2 (A0H)							
P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.1
A7H	A6H	A5H	A4H	A3H	A2H	A1H	A0H

P1 (90H)							
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
97H	96H	95H	94H	93H	92H	91H	90H

P0 (80H)							
P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
87H	86H	85H	84H	83H	82H	81H	80H

IP (B8H)							
---	---	---	PS	PT1	PX1	PT0	PX0
BFH	BEH	BDH	BCH	BBH	BAH	B9H	B8H

IE (A8H)							
EA	---	---	ES	ET1	EX1	ET0	EX0
AFH	AEH	ADH	ACH	ABH	AAH	A9H	A8H

SCON (98H)							
SM0	SM1	SM2	REN	TB8	RB8	TI	RI
9FH	9EH	9DH	9CH	9BH	9AH	99H	98H

TCON (88H)							
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
8FH	8EH	8DH	8CH	8BH	8AH	89H	88H

TMOD (89H)							
G1	C/T1	M1.1	M0.1	G0	C/T0	M1.0	M0.0

PCON (87H)							
smod	---	---	---	GF1	GF0	PD	IDL

T1 [16bit]	
TH1 (8DH)	TL1 (8BH)

T0 [16bit]	
TH0 (8CH)	TL0 (8AH)

SBUF (99H)

PC [16bit]

DPTR [16bit]	
DPH (83H)	DPL (82H)

SP (81H)

ROM	
	max 32KB (7FFFH)
TxD&RxD	(0023H)
T1	(001BH)
INT1	(0013H)
T0	(000BH)
INT0	(0003H)
	(0000H)

RAM [byte address]
30H - 7FH

RAM [bite and byte address]							
(2FH)							
7FH	7EH	7DH	7CH	7BH	7AH	79H	78H
(2EH)							
77H	76H	75H	74H	73H	72H	71H	70H
(2DH)							
6FH	6EH	6DH	6CH	6BH	6AH	69H	68H
(2CH)							
67H	66H	65H	64H	63H	62H	61H	60H
(2BH)							
5FH	5EH	5DH	5CH	5BH	5AH	59H	58H
(2AH)							
57H	56H	55H	54H	53H	52H	51H	50H
(29H)							
4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
(28H)							
47H	46H	45H	44H	43H	42H	41H	40H
(27H)							
3FH	3EH	3DH	3CH	3BH	3AH	39H	38H
(26H)							
37H	36H	35H	34H	33H	32H	31H	30H
(25H)							
2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
(24H)							
27H	26H	25H	24H	23H	22H	21H	20H
(23H)							
1FH	1EH	1DH	1CH	1BH	1AH	19H	18H
(22H)							
17H	16H	15H	14H	13H	12H	11H	10H
(21H)							
0FH	0EH	0DH	0CH	0BH	0AH	09H	08H
(20H)							
07H	06H	05H	04H	03H	02H	01H	00H

BANK3							
R7	R6	R5	R4	R3	R2	R1	R0
1FH	1EH	1DH	1CH	1BH	1AH	19H	18H

BANK2							
R7	R6	R5	R4	R3	R2	R1	R0
17H	16H	15H	14H	13H	21H	11H	10H

BANK1							
R7	R6	R5	R4	R3	R2	R1	R0
0FH	0EH	0DH	0CH	0BH	0AH	09H	08H
BANK 0							
R7	R6	R5	R4	R3	R2	R1	R0
07H	06H	05H	04H	03H	02H	01H	00H

Назначение разрядов регистра PSW

<table>
<tr><th>Название бита</th><th>Позиция</th><th>Назначение</th></tr>
<tr><td>C</td><td>PSW.7</td><td>Флаг переноса. Устанавливается и сбрасывается аппаратно при выполнении арифметических, логических и битовых операций, а также программно</td></tr>
<tr><td>AC</td><td>PSW.6</td><td>Флаг вспомогательного переноса. Устанавливается и сбрасывается только аппаратно при выполнении команд суммирования и вычитания в случае возникновения переноса или займа в бите 3 аккумулятора</td></tr>
<tr><td>FO</td><td>PSW.5</td><td>Свободный флаг. Может быть изменен программно и используется по назначению, установленному программистом</td></tr>
<tr><td>RS1
RS0</td><td>PSW.4
PSW.3</td><td>Выбор банка регистров. Биты устанавливаются и сбрасываются программно для выбора активного (рабочего) банка регистров:
<table>
<tr><th>RS1</th><th>RS0</th><th>Активный банк</th><th>Адреса РПД</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>00H-07H</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>08H-0FH</td></tr>
<tr><td>1</td><td>0</td><td>2</td><td>10H-17H</td></tr>
<tr><td>1</td><td>1</td><td>3</td><td>18H-1FH</td></tr>
</table></td></tr>
<tr><td>OV</td><td>PSW.2</td><td>Флаг переполнения. Устанавливается и сбрасывается аппаратно при выполнении арифметических операций в случае переполнения аккумулятора. Дает возможность корректно выполнять действия над числами, представленными в дополнительном коде</td></tr>
<tr><td>-</td><td>PSW.1</td><td>Не используется</td></tr>
<tr><td>P</td><td>PSW.0</td><td>Флаг паритета. Устанавливается и сбрасывается аппаратно в каждом цикле команды, фиксирует факт нечетного количества «1» в аккумуляторе</td></tr>
</table>

Таблица.7.*Команды передачи данных*

Название команды	Мнемокод	КОП	Т	Б	Ц	Операция
Пересылка в аккумулятор из регистра (n=0÷7)	MOV A, Rn	11101rrr	1	1	1	(A) ← (Rn)
Пересылка в аккумулятор прямоадресуемого байта	MOV A, ad	11100101	3	2	1	(A) ← (ad)
Пересылка в аккумулятор байта из РПД (i=0,1)	MOV A, @Ri	1110011i	1	1	1	(A) ← ((Ri))
Загрузка в аккумулятор константы	MOV A, #d	01110100	2	2	1	(A) ← #d
Пересылка в регистр из аккумулятора	MOV Rn, A	11111rrr	1	1	1	(Rn) ← (A)
Пересылка в регистр прямоадресуемого байта	MOV Rn, ad	10101rrr	3	2	2	(Rn) ← (ad)
Загрузка в регистр константы	MOV Rn, #d	01111rrr	2	2	1	(Rn) ← #d
Пересылка по прямому адресу аккумулятора	MOV ad, A	11110101	3	2	1	(ad) ← (A)
Пересылка по прямому адресу регистра	MOV ad, Rn	10001rrr	3	2	2	(ad) ← (Rn)
Пересылка прямоадресуемого байта по прямому адресу	MOV add, ads	10000101	9	3	2	(add) ← (ads)
Пересылка байта из РПД по прямому адресу	MOV ad, @Ri	1000011i	3	2	2	(ad) ← ((Ri))
Пересылка по прямому адресу константы	MOV ad, #d	01110101	7	3	2	(ad) ← #d
Пересылка в РПД из аккумулятора	MOV @Ri, A	1111011i	1	1	1	((Ri)) ← (A)
Пересылка в РПД прямоадресуемого байта	MOV @Ri, ad	0110011i	3	2	2	((Ri)) ← (ad)
Пересылка в РПД константы	MOV @Ri, #d	0111011i	2	2	1	((Ri)) ← #d
Загрузка указателя данных	MOV DPTR, #d16	10010000	13	3	2	(DPTR) ← #d16
Пересылка в аккумулятор байта из ПП	MOVC A, @A+DPTR	10010011	1	1	2	← ((A) +(DPTR))
Пересылка в аккумулятор байта из ПП	MOVC A, @A+PC	10000011	1	1	2	(PC) ← (PC)+1, (A) ← ((A)+(PC))
Пересылка в аккумулятор байта из ВПД	MOVX A, @Ri	1110001i	1	1	2	(A) ← ((Ri))
Пересылка в аккумулятор байта из расширенной ВПД	MOVX A,@DPTR	11100000	1	1	2	(A) ← ((DPTR))
Пересылка в ВПД из аккумулятора	MOVX @Ri, A	1111001i	1	1	2	((Ri)) ← (A)
Пересылка в расширенную ВПД из аккумулятора	MOVX @DPTR,A	11110000	1	1	2	((DPTR)) ← (A)
Загрузка в стек	PUSH ad	11000000	3	2	2	(SP) ← (SP) + 1, ((SP)) ← (ad)
Извлечение из стека	POP ad	11010000	3	2	2	(ad) ← (SP), (SP) ← (SP) - 1
Обмен аккумулятора с регистром	XCH A, Rn	11001rrr	1	1	1	(A) ↔ (Rn)
Обмен аккумулятора с прямоадресуемым байтом	XCH A, ad	11000101	3	2	1	(A) ↔ (ad)
Обмен аккумулятора с байтом из РПД	XCH A, @Ri	1100011i	1	1	1	(A) ↔ ((Ri))
Обмен младших тетрад аккумулятора и байта РПД	XCHD A, @Ri	1101011i	1	1	1	$(A_{0...3}) \leftrightarrow ((Ri)_{0...3})$

Таблица.8.*Арифметические операции.*

Название команды	Мнемокод	КОП	Т	Б	Ц	Операция
Сложение аккумулятора с регистром (n=0÷7)	ADD A, Rn	00l01rrr	1	1	1	(A) ← (A) + (Rn)
Сложение аккумулятора с прямоадресуемым байтом	ADD A, ad	00100101	3	2	1	(A) ← (A) + (ad)
Сложение аккумулятора с байтом из РПД (i = 0,1)	ADD A, @Ri	0010011i	1	1	1	(A) ← (A) + ((Ri))
Сложение аккумулятора с константой	ADD A, #d	00100100	2	2	1	(A) ← (A) + #d
Сложение аккумулятора с регистром и переносом	ADDC A, Rn	00111rrr	1	1	1	(A) ← (A) + (Rn) + (C)
Сложение аккумулятора с прямоадресуемым байтом и переносом	ADDC A, ad	00110101	3	2	1	(A) ← (A) + (ad) + (C)
Сложение аккумулятора с байтом из РПД и переносом	ADDC A, @Ri	0011011i	1	1	1	(A) ← (A) + ((Ri)) + (C)
Сложение аккумулятора с константой и переносом	ADDC A, #d	00110100	2	2	1	(A) ← (A) + # d + (C)
Десятичная коррекция аккумулятора	DA A	11010100	1	1	1	Если ($A_{0...3}$)>9 или ((AC)=1),то($A_{0...3}$)← ($A_{0...3}$) + 6, затем если ($A_{4...7}$) >9 или ((C)=1), то ($A_{4...7}$) ← ($A_{4...7}$) + 6
Вычитание из аккумулятора регистра и заёма	SUBB A, Rn	10011rrr	1	1	1	(A) ← (A) - (C) - (Rn)
Вычитание из аккумулятора прямоадресуемого байта и заема	SUBB A, ad	10010101	3	2	1	(A) ← (A) - (C) - ((ad))
Вычитание из аккумулятора байта РПД и заема	SUBB A, @Ri	1001011i	1	1	1	(A) ← (A) - (C) - ((Ri))
Вычитание из аккумулятора константы и заема	SUBB A, d	10010100	2	2	1	(A) ← (A) - (C) - #d
Инкремент аккумулятора	INC A	00000100	1	1	1	(A) ← (A) + 1
Инкремент регистра	INC Rn	00001rrr	1	1	1	(Rn) ← (Rn) + 1
Инкремент прямоадресуемого байта	INC ad	00000101	3	2	1	(ad) ← (ad) + 1
Инкремент байта в РПД	INC @Ri	0000011i	1	1	1	((Ri)) ← ((Ri)) + 1
Инкремент указателя данных	INC DPTR	10100011	1	1	2	(DPTR) ← (DPTR) + 1
Декремент аккумулятора	DEC A	00010100	1	1	1	(A) ← (A) – 1
Декремент регистра	DEC Rn	00011rrr	1	1	1	(Rn) ← (Rn) – 1
Декремент прямоадресуемого байта	DEC ad	00010101	3	2	1	(ad) ← (ad) – 1
Декремент байта в РПД	DEC @Ri	0001011i	1	1	1	((Ri)) ← ((Ri)) - 1
Умножение аккумулятора на регистр B	MUL AB	10100100	1	1	4	(B)(A) ← (A)*(B)
Деление аккумулятора на регистр B	DIV AB	10000100	1	1	4	(B).(A) ← (A)/(B)

Таблица.9.*Логические операции*

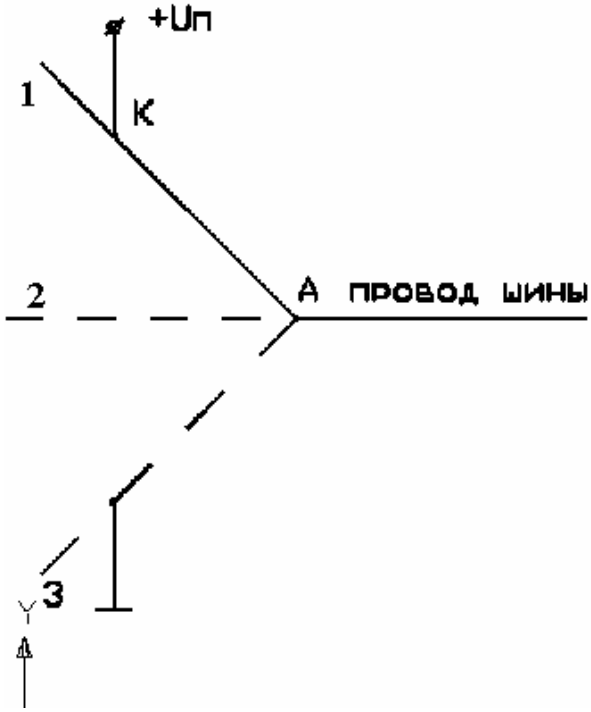
Название команды	Мнемокод	КОП	Т	Б	Ц	Операция
Логическое И аккумулятора и регистра	ANL A, Rn	01011rrr	1	1	1	(A) ← (A) AND (Rn)
Логическое И аккумулятора и прямоадресуемого байта	ANL A, ad	01010101	3	2	1	(A) ← (A) AND (ad)
Логическое И аккумулятора и байта из РПД	ANL A, @Ri	0101011i	1	1	1	(A) ← (A) AND ((Ri))
Логическое И аккумулятора и константы	ANL A, #d	01010100	2	2	1	(A) ← (A) AND #d
Логическое И прямоадресуемого байта и аккумулятора	ANL ad, A	01010010	3	2	1	(ad) ← (ad) AND (A)
Логическое И прямоадресуемого байта и константы	ANL ad, #d	01010011	7	3	2	(ad) ← (ad) AND #d
Логическое ИЛИ аккумулятора и регистра	ORL A, Rn	01001rrr	1	1	1	(A) ← (A) OR (Rn)
Логическое ИЛИ аккумулятора и прямоадресуемого байта	ORL A, ad	01000101	3	2	1	(A) ← (A) OR (ad)
Логическое ИЛИ аккумулятора и байта из РПД	ORL A, @Ri	0100011i	1	1	1	(A) ← (A) OR ((Ri))
Логическое ИЛИ аккумулятора и константы	ORL A, #d	01000100	2	2	1	(A) ← (A) OR #d
Логическое ИЛИ прямоадресуемого байта и аккумулятора	ORL ad, A	01000010	3	2	1	(ad) ← (ad) OR (A)
Логическое ИЛИ прямоадресуемого байта и константы	ORL ad, #d	01000011	7	3	2	(ad) ← (ad) OR #d
Исключающее ИЛИ аккумулятора и регистра	XRL A, Rn	01101rrr	1	1	1	(A) ← (A) XOR (Rn)
Исключающее ИЛИ аккумулятора и прямоадресуемого байта	XRL A, ad	01100101	3	2	1	(A) ← (A) XOR (ad)
Исключающее ИЛИ аккумулятора и байта из РПД	XRL A, @Ri	0110011i	1	1	1	(A) ← (A) XOR ((Ri))
Исключающее ИЛИ аккумулятора и константы	XRL A, #d	01100100	2	2	1	(A) ← (A) XOR #d
Исключающее ИЛИ прямоадресуемого байта и аккумулятора	XRL ad, A	01100010	3	2	1	(ad) ← (ad) XOR (A)
Исключающее ИЛИ прямоадресуемого байта и константы	XRL ad, #d	01100011	7	3	2	(ad) ← (ad) XOR #d
Сброс аккумулятора	CLR A	11100100	1	1	1	(A) ← 0
Инверсия аккумулятора	CPL A	11110100	1	1	1	(A) ← NOT(A)
Сдвиг аккумулятора влево циклический	RL A	00100011	1	1	1	$(A_{n+1}) \leftarrow (A_n)$, n=0÷6, $(A_0) \leftarrow (A_7)$
Сдвиг аккумулятора влево через перенос	RLC A	00110011	1	1	1	$(A_{n+1}) \leftarrow (A_n)$, n=0÷6 $(A_0) \leftarrow (C)$, $(C) \leftarrow (A_7)$
Сдвиг аккумулятора вправо циклический	RR A	00000011	1	1	1	$(A_n) \leftarrow (A_{n+1})$, n=0÷6, $(A_7) \leftarrow (A_0)$
Сдвиг аккумулятора вправо через перенос	RRC A	00010011	1	1	1	$(A_n) \leftarrow (A_{n+1})$, n=0÷6 $(A_7) \leftarrow (C)$, $(C) \leftarrow (A_0)$
Обмен местами тетрад в аккумуляторе	SWAP A	11000100	1	1	1	$(A_{0...3}) \leftrightarrow (A_{4...7})$

Таблица.10.*Операции с битами*

Название команды	Мнемокод	КОП	Т	Б	Ц	Операция
Сброс переноса	CLR C	11000011	1	1	1	(C) ← 0
Сброс бита	CLR bit	11000010	4	2	1	(b) ← 0
Установка переноса	SETB C	11010011	1	1	1	(C) ← 1
Установка бита	SETB bit	11010010	4	2	1	(b) ← 1
Инверсия переноса	CPL C	10110011	1	1	1	(C) ← NOT(C)
Инверсия бита	CPL bit	10110010	4	2	1	(b) ← NOT(b)
Логическое И бита и переноса	ANL C, bit	10000010	4	2	2	(C) ← (C) AND (b)
Логическое И инверсии бита и переноса	ANL C, /bit	10110000	4	2	2	(C) ← (C) AND (NOT(b))
Логическое ИЛИ бита и переноса	ORL C, bit	01110010	4	2	2	(C) ← (C) OR (b)
Логическое ИЛИ инверсии бита и переноса	ORL C, /bit	10100000	4	2	2	(C) ← (C) OR (NOT(b))
Пересылка бита в перенос	MOV C, bit	10100010	4	2	1	(C) ← (b)
Пересылка переноса в бит	MOV bit, C	10010010	4	2	2	(b) ← (C)

Таблица.11.*Команды передачи управления*

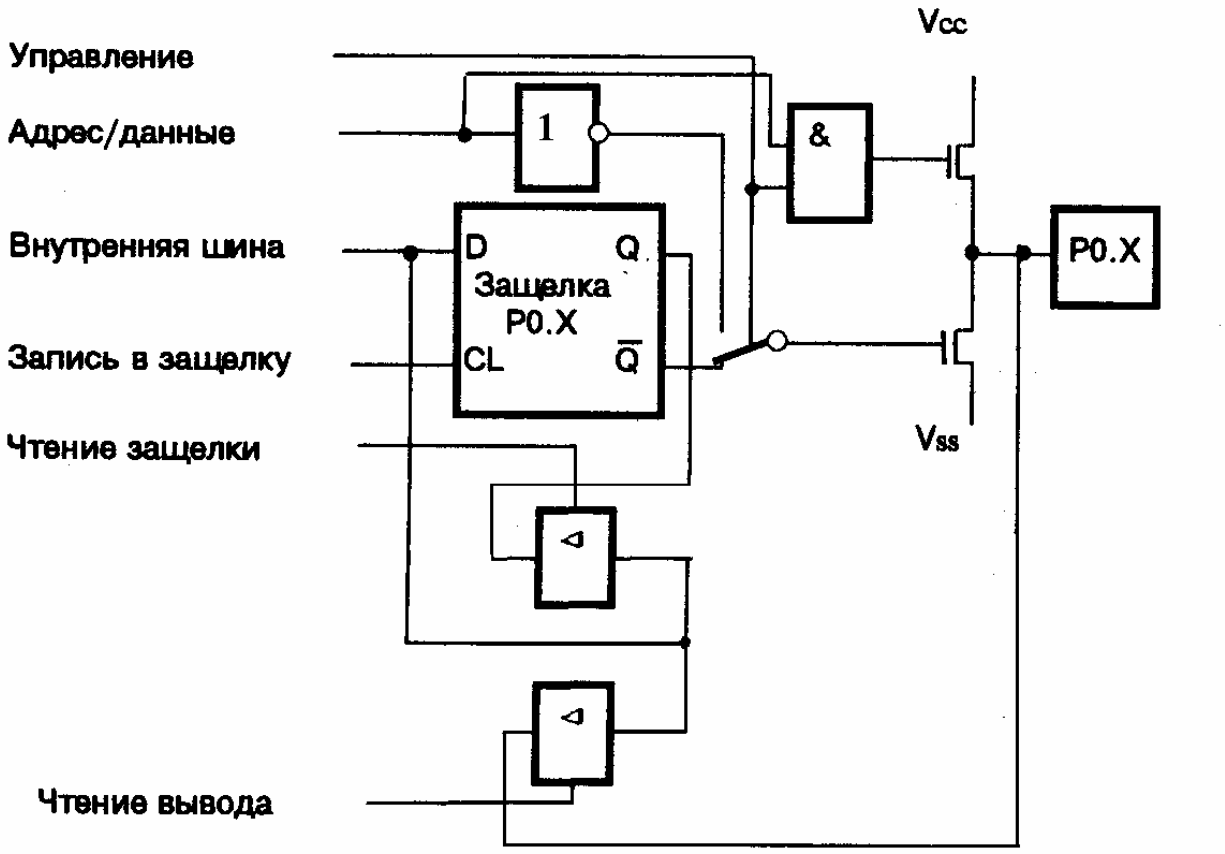
Название команды	Мнемокод	КОП	Т	Б	Ц	Операция
Длинный переход в полном объеме ПП	LJMP ad16	00000010	12	3	2	(PC) ← ad16
Абсолютный переход внутри страницы в 2 Кб	AJMP ad11	$a_{10}a_9a_8$00001	6	2	2	(PC) ← (PC) + 2, (PC_{0-10}) ← ad11
Короткий относительный переход внутри страницы в 256 байт	SJMP rel	10000000	5	2	2	(PC) ← (PC) + 2, (PC) ← (PC) +rel
Косвенный относительный переход	JMP @A+DPTR	01110011	1	1	2	(PC) ← (A) + (DPTR)
Переход, если аккумулятор равен нулю	JZ rel	01100000	5	2	2	(PC)←(PC)+2, если (A)=0, то (PC)←(PC)+rel
Переход, если аккумулятор не равен нулю	JNZ rel	01110000	5	2	2	(PC)←(PC)+2, если (A)≠0, то (PC)←(PC)+rel
Переход, если перенос равен единице	JC rel	01000000	5	2	2	(PC)←(PC)+2, если (C)=1, то (PC)←(PC)+rel
Переход, если перенос равен нулю	JNC rel	01010000	5	2	2	(PC)←(PC)+2, если (C)=0, то (PC)←(PC)+rel
Переход, если бит равен единице	JB bit, rel	00100000	11	3	2	(PC)←(PC)+3, если (b)=I, то (PC)←(PC)+rel
Переход, если бит равен нулю	JNB bit, rel	00110000	11	3	2	(PC)←(PC)+3, если (b)=0, то (PC)←(PC)+rel
Переход, если бит установлен, с последующим сбросом бита	JBC bit, rel	00010000	11	3	2	(PC) ← (PC) + 3, если (b)=1, то (b) ← 0 и (PC)← (PC) + rel
Декремент регистра и переход, если не нуль	DJNZ Rn, rel	11011rrr	5	2	2	(PC) ← (PC) + 2, (Rn) ← (Rn) - 1, если (Rn) ≠ 0, то (PC) ← (PC) + rel
Декремент прямоадресуемого байта и переход, если не нуль	DJNZ ad, rel	11010101	8	3	2	(PC) ← (PC) + 2, (ad) ← (ad) - 1, если (ad) ≠ 0, то (PC) ← (PC) + rel
Сравнение аккумулятора с прямоадресуемым байтом и переход, если не равно	CJNE A, ad, rel	10110101	8	3	2	(PC) ← (PC) + 3,если (A) ≠ (ad), то (PC) ← (PC) + rel,если (A) < (ad), то (C) ← 1, иначе (C) ← 0
Сравнение аккумулятора с константой и переход, если не равно	CJNE A, #d, rel	10110100	10	3	2	(PC) ← (PC) + 3,если (A) ≠ #d, то (PC) ← (PC) + rel,если (A) < #d, то (C) ← 1, иначе (C) ← 0
Сравнение регистра с константой и переход, если не равно	CJNE Rn, #d, re	10111rrr	10	3	2	(PC) ← (PC) + 3,если (Rn) ≠ #d, то (PC) ← (PC) + rel,если (Rn) < #d, то (C) ← 1, иначе (C) ← 0
Сравнение байта в РПД с константой и	CJNE @Ri,#d,re	1011011i	10	3	2	(PC) ← (PC) + 3,если ((Ri)) ≠ #d, то
переход, если не равно						(PC) ← (PC) + rel,если ((Ri)) < #d, то (C) ← 1, иначе (C) ← 0
Длинный вызов подпрограммы	LCALL adl6	00010010	12	3	2	(PC) ← (PC) + 3, (SP) ← (SP) +1, ((SP)) ← ($PC_{0...7}$), (SP) ← (SP) + 1, ((SP)) ← ($PC_{8...15}$), (PC) ← ad16
Абсолютный вызов подпрограммы в пределах страницы в 2 Кб	ACALL ad11	$a_{10}a_9a_8$10001	6	2	2	(PC) ← (PC) + 2, (SP) ← (SP) + 1, ((SP)) ← ($PC_{0...7}$), (SP) ← (SP) + 1, ((SP)) ← ($PC_{8...15}$), (PC_{0-10}) ← ad11
Возврат из подпрограммы	RET	00100010	1	1	2	($PC_{8...15}$) ← ((SP)), (SP) ← (SP) - 1, ($PC_{0...7}$) ← ((SP)), (SP) ← (SP) – 1
Возврат из подпрограммы обработки прерывания	RETI	00110010	1	1	2	($PC_{8...15}$) ← ((SP)), (SP) ← (SP) - 1, ($PC_{0...7}$) ← ((SP)), (SP) ← (SP) – 1
Пустая операция	NOP	00000000	1	1	1	(PC) ← (PC) + 1

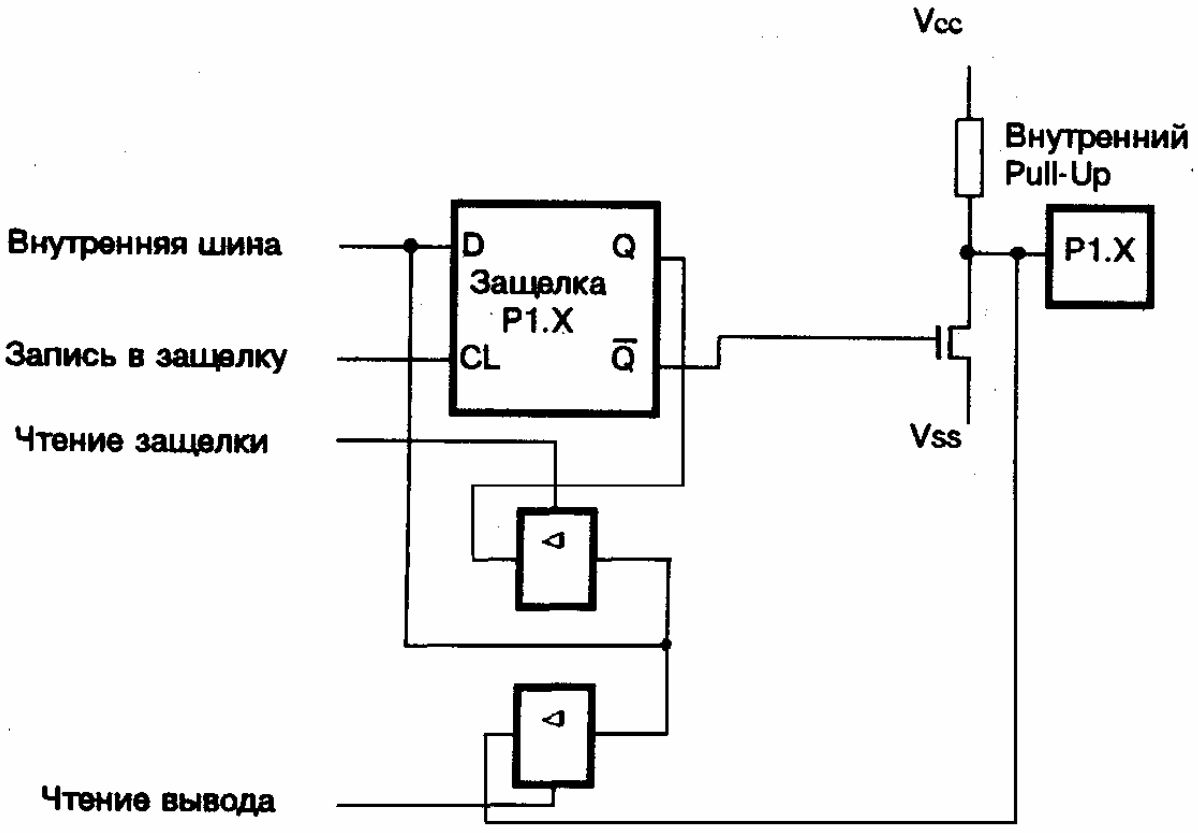


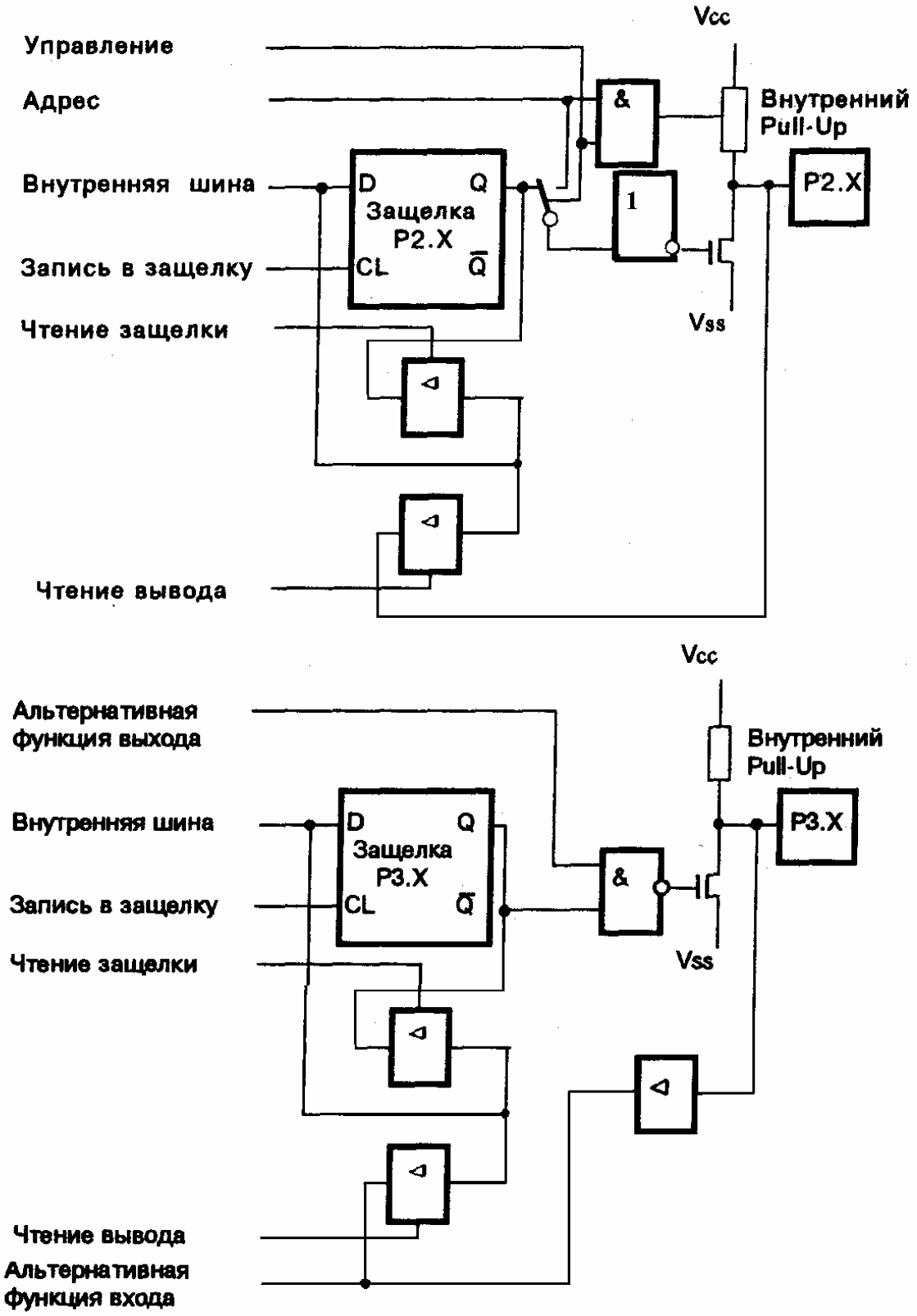
б)

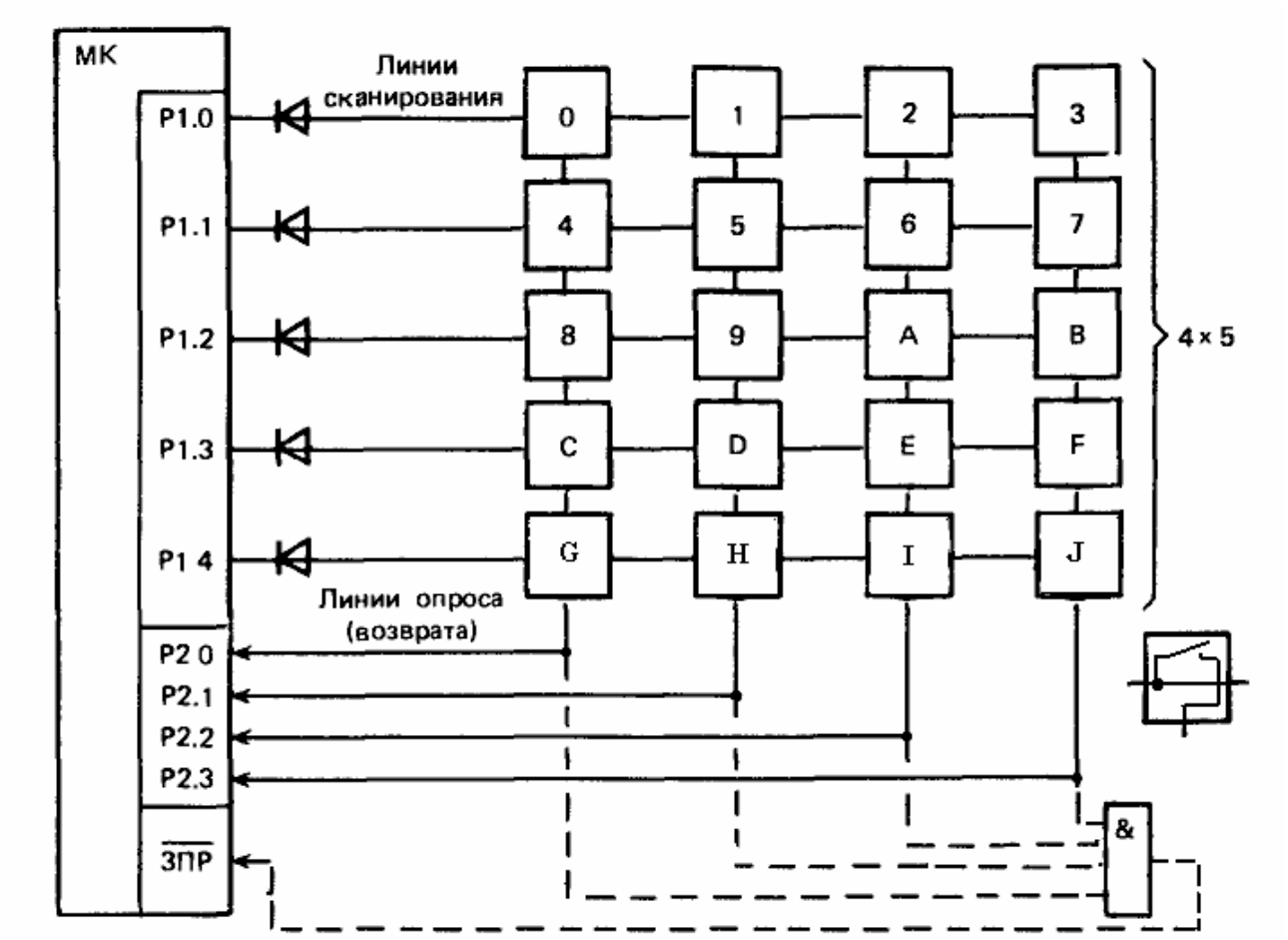
АТ

положение переключателя	1	2	3
состояние провода шины	1	∞	0


Управление
Адрес/данные
Внутренняя шина
Запись в защелку
Чтение защелки
Чтение вывода
1
&
D
Q
Защелка
P0.X
CL
Q
Vcc
Vss
P0.X

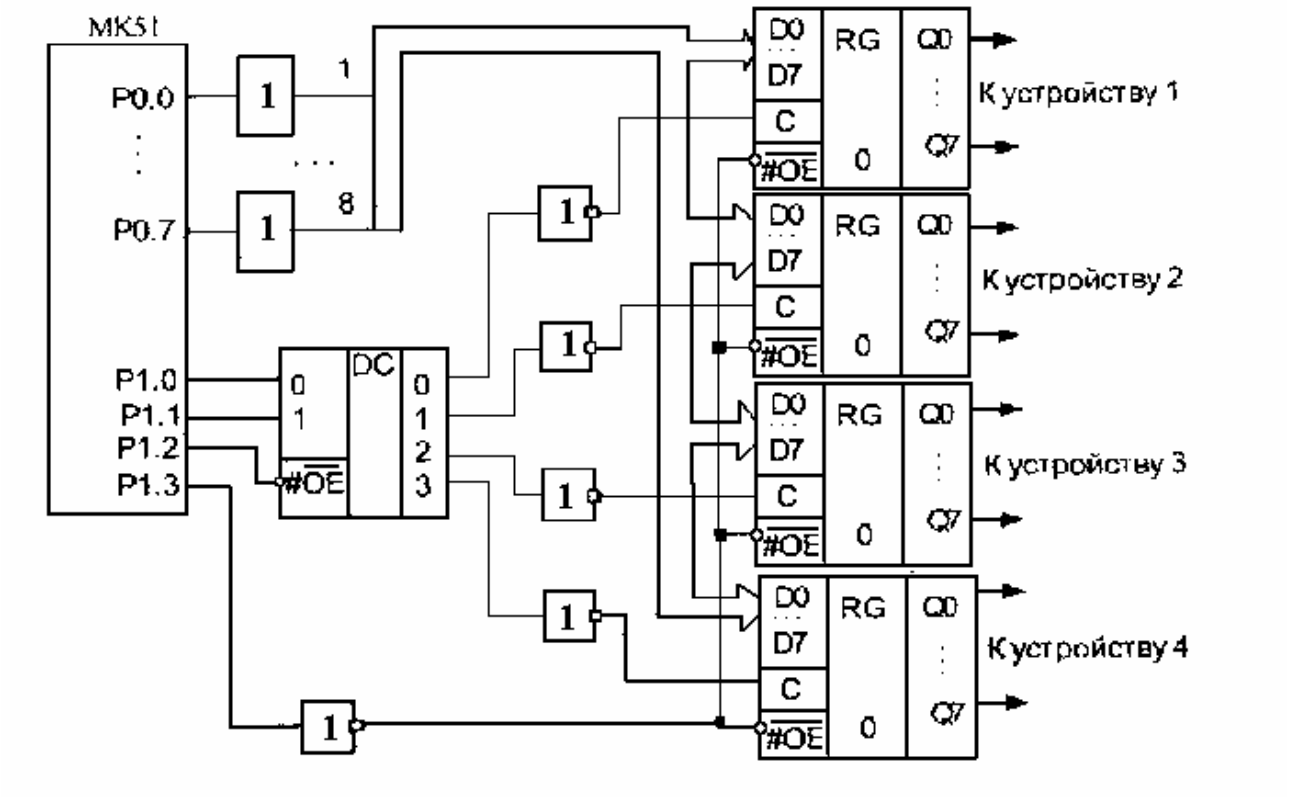

Внутренняя шина
Запись в защелку
Чтение защелки
Чтение вывода
D
Q
Защелка
P1.X
CL
Q
Vcc
Внутренний
Pull-Up
P1.X
Vss

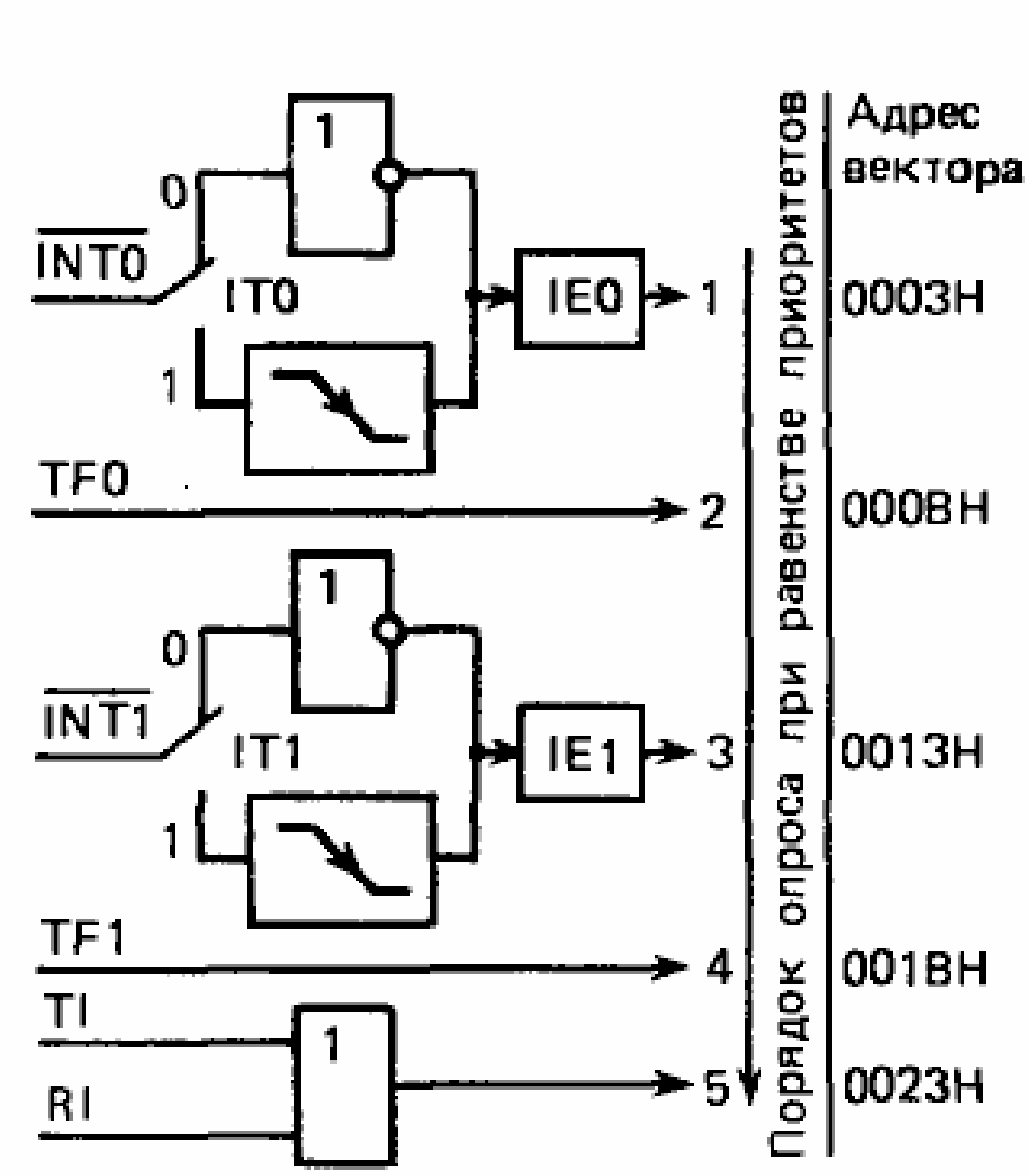

Vcc
Управление
Адрес
&
Внутренний
Pull-Up
Внутренняя шина
D
Q
Защелка
P2.X
1
P2.X
Запись в защелку
CL
Q
Чтение защелки
Vss
Чтение вывода
Vcc
Альтернативная
функция выхода
Внутренний
Pull-Up
Внутренняя шина
D
Q
Защелка
P3.X
P3.X
&
Запись в защелку
CL
Q
Чтение защелки
Vss
Чтение вывода
Альтернативная
функция входа

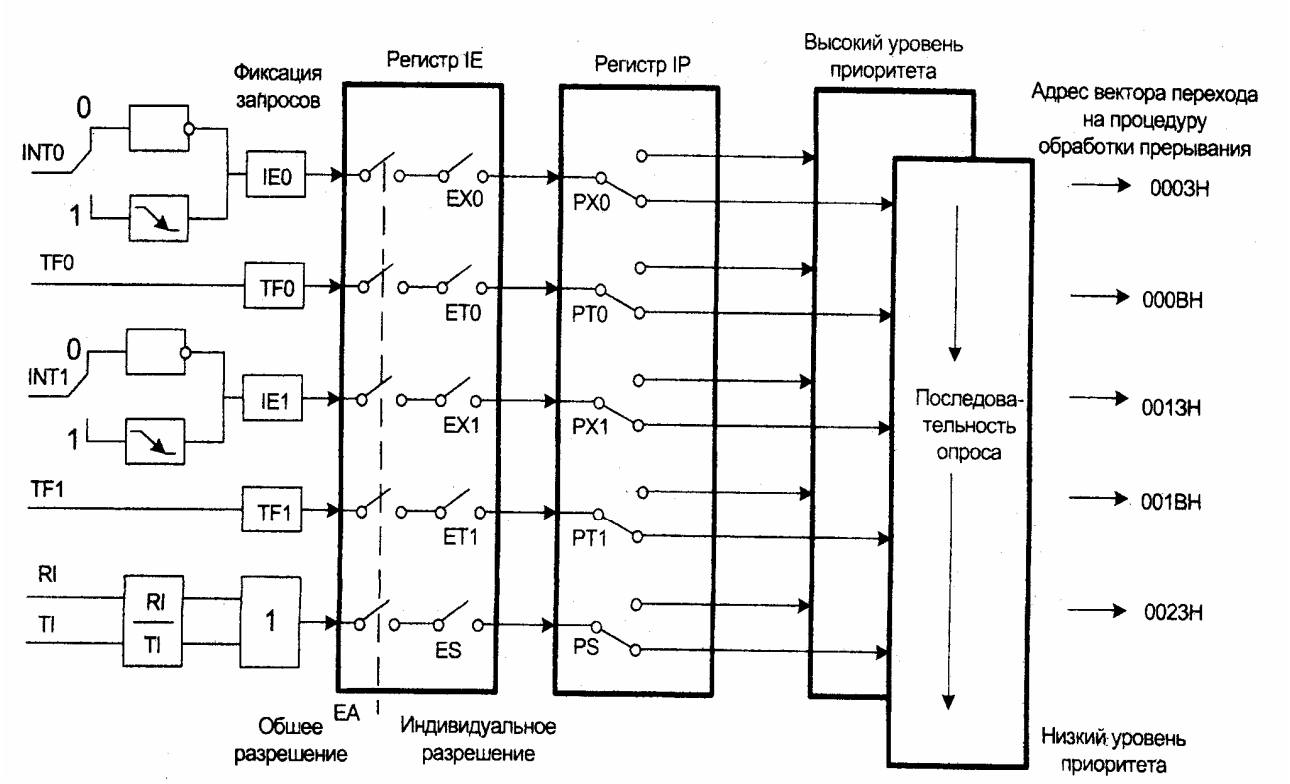

МК
Линии сканирования
P1.0
P1.1
P1.2
P1.3
P1 4
0
1
2
3
4
5
6
7
8
9
A
B
C
D
E
F
G
H
I
J
4×5
Линии опроса (возврата)
P2 0
P2.1
P2.2
P2.3
ЗПР
&

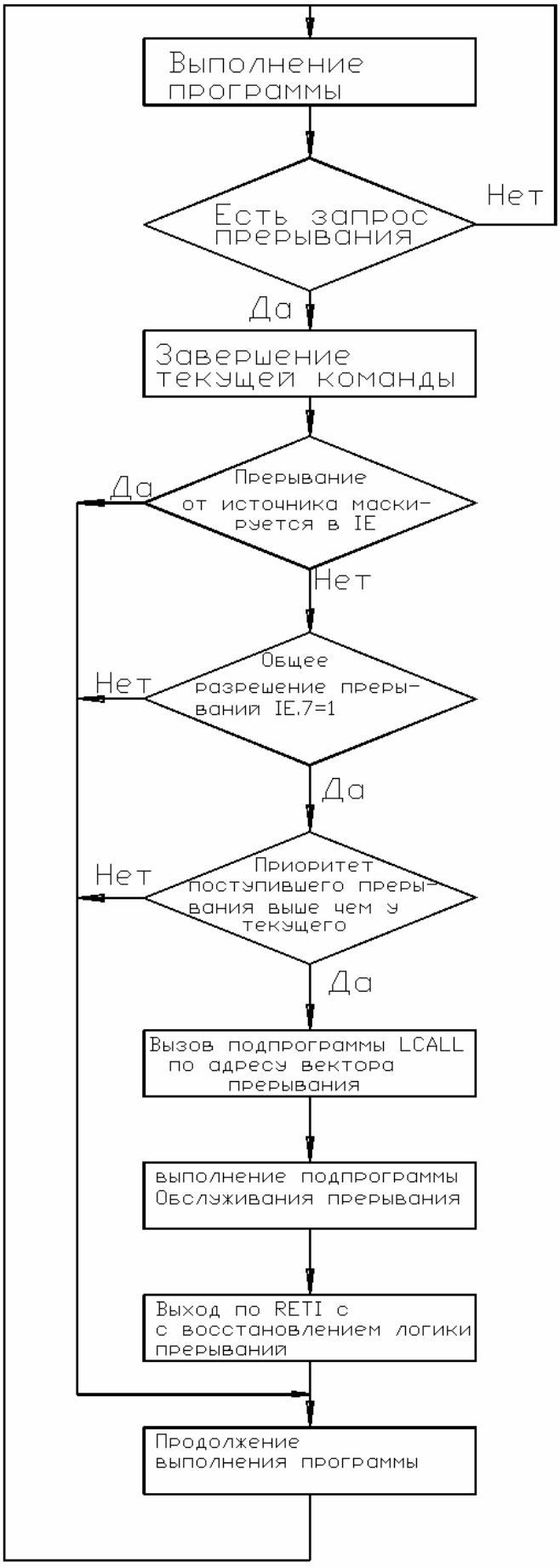
Альтернативные функции порта 3

Символ	Позиция	Имя и назначение
$\overline{RD}$	P3.7	Чтение. Активный сигнал низкого уровня формируется аппаратурно при обращении к ВПД
$\overline{WR}$	P3.6	Запись. Активный сигнал низкого уровня формируется аппаратурно при обращении к ВПД
T1	P3.5	Вход таймера/счетчика 1 или тест-вход
T0	P3.4	Вход таймера/счетчика 0 или тест-вход
$\overline{INT1}$	P3.3	Вход запроса прерывания 1. Воспринимается сигнал низкого уровня или срез
$\overline{INT0}$	P3.2	Вход запроса прерывания 0. Воспринимается сигнал низкого уровня или срез
TXD	P3.1	Выход передатчика последовательного порта в режиме УАПП. Выход синхронизации в режиме сдвигающего регистра
RXD	P3.0	Вход приемника последовательного порта в режиме УАПП. Ввод/вывод данных в режиме сдвигающего регистра


MK51
P0.0
P0.7
P1.0
P1.1
P1.2
P1.3
1
8
DC
0
1
2
3
#OE
D0
D7
C
RG
Q0
Q7
К устройству 1
К устройству 2
К устройству 3
К устройству 4


Адрес вектора
Порядок опроса при равенстве приоритетов
1
0
INT0
IT0
IE0
1
0003H
1
TF0
2
000BH
1
0
INT1
IT1
IE1
3
0013H
1
TF1
4
001BH
TI
RI
1
5
0023H


Фиксация запросов
Регистр IE
Регистр IP
Высокий уровень приоритета
Адрес вектора перехода на процедуру обработки прерывания
0
INT0
1
IE0
EX0
PX0
0003H
TF0
TF0
ET0
PT0
000BH
0
INT1
1
IE1
EX1
PX1
0013H
Последова-тельность опроса
TF1
TF1
ET1
PT1
001BH
RI
TI
RI
TI
1
ES
PS
0023H
EA
Общее разрешение
Индивидуальное разрешение
Низкий уровень приоритета


Выполнение программы
Есть запрос прерывания
Нет
Да
Завершение текущей команды
Прерывание от источника маскируется в IE
Да
Нет
Общее разрешение прерываний IE.7=1
Нет
Да
Приоритет поступившего прерывания выше чем у текущего
Нет
Да
Вызов подпрограммы LCALL по адресу вектора прерывания
выполнение подпрограммы Обслуживания прерывания
Выход по RETI с с восстановлением логики прерываний
Продолжение выполнения программы

Организация регистра разрешения прерываний IE.

Символическое обозначение	Позиционное обозначение	Название и назначение
EA	IE.7	Бит разрешения прерываний. Программно сбрасывается для запрета всех прерываний и устанавливается для разрешения прерываний в зависимости от состояния битов IE.4-IE.0.
	IE.6	Не используется.
	IE.5	Не используется.
ES	IE.4	Бит управления прерываниями последовательного порта. Программно устанавливается/сбрасывается для разрешения/запрета прерываний по флагам TI и RI.
ET1	IE.3	Бит разрешения прерывания от таймера 1. Программно устанавливается и сбрасывается для разрешения/запрета прерываний от таймера/счетчика 1.
EX1	IE.2	Бит разрешения внешнего прерывания 1. Программно устанавливается/сбрасывается для разрешения/запрета аппаратного прерывания INT1.
ET0	IE.1	Бит разрешения прерывания от таймера 0. Программно устанавливается/сбрасывается для разрешения/запрета прерываний от таймера/счетчика 0.
EX0	IE.0	Бит разрешения внешнего прерывания 0. Программно устанавливается/сбрасывается для разрешения/запрета аппаратного прерывания INT0.

Организация регистра приоритетов прерываний IP

Символическое обозначение	Позиционное обозначение	Название и назначение
	IP.7	Не используется.
	IP.6	Не используется.
	IP.5	Не используется.
PS	IP.4	Бит управления приоритетом прерываний последовательного порта. Программно устанавливается/сбрасывается для присвоения высокого/низкого приоритета прерываниям последовательного порта.
PT1	IP.3	Бит управления приоритетом прерывания от таймера 1. Программно устанавливается/сбрасывается для присвоения высокого/низкого приоритета прерываниям от таймера/счетчика 1.
PX1	IP.2	Бит управления приоритетом внешнего прерывания 1. Программно устанавливается/сбрасывается для присвоения высокого/низкого приоритета аппаратному прерыванию INT1.
PT0	IP.1	Бит управления приоритетом прерывания от таймера 0. Программно устанавливается/сбрасывается для присвоения высокого/низкого приоритета прерываниям от таймера/счетчика 0.
PX0	IP.0	Бит управления приоритетом внешнего прерывания 0. Программно устанавливается/сбрасывается для присвоения высокого/низкого приоритета аппаратному прерываний INT0.

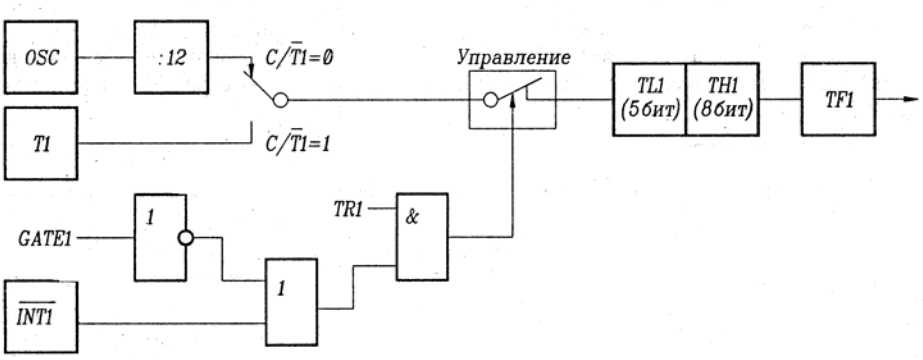


Рис. 1. Логика работы таймера/счетчика 1 в режиме 0

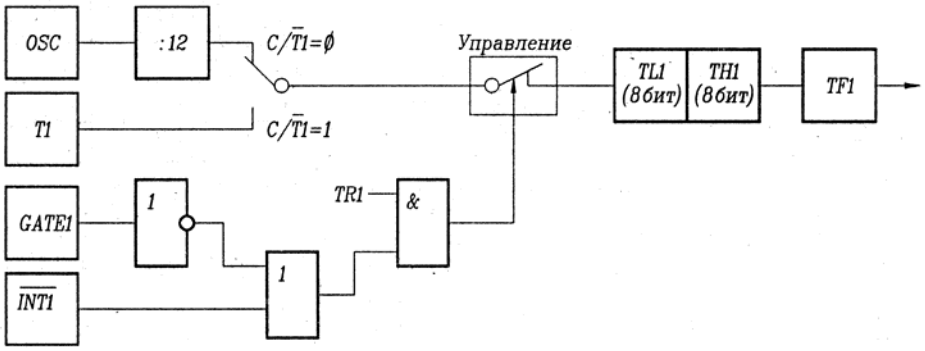


Рис. 2. Логика работы таймера/счетчика 1 в режиме 1

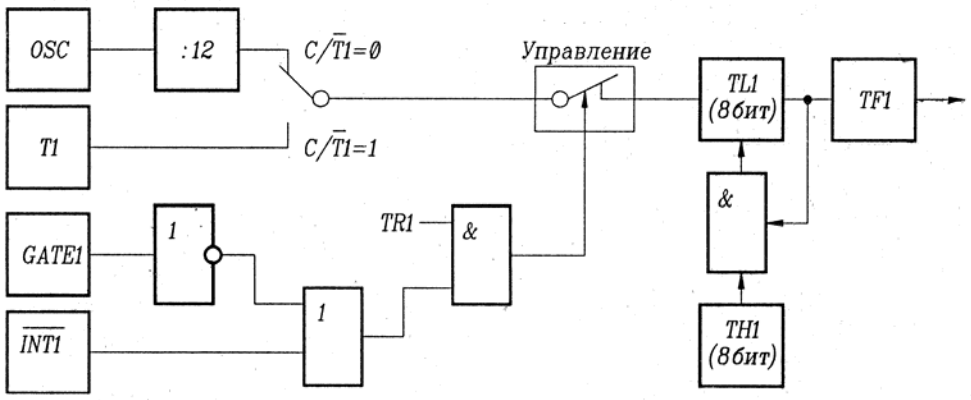


Рис. 3. Логика работы таймера/счетчика 1 в режиме 2

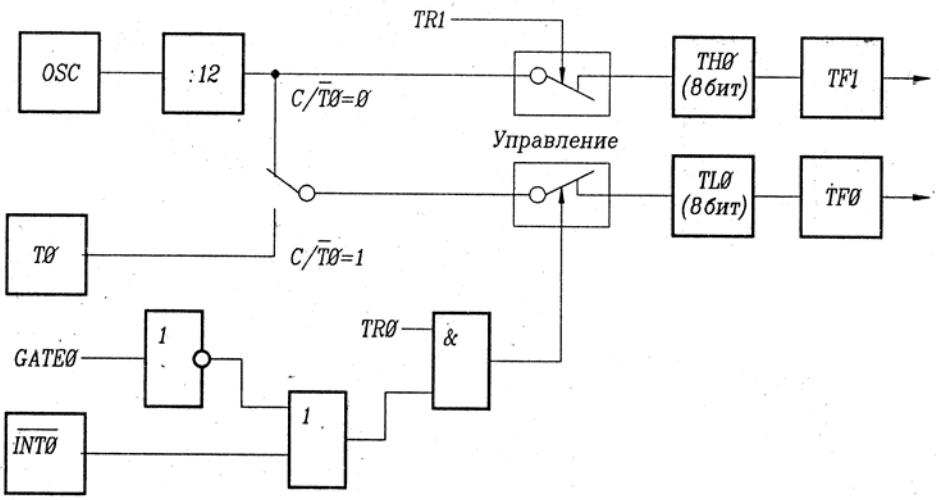


Рис. 4. Логика работы таймера/счетчика 0 в режиме 3

TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

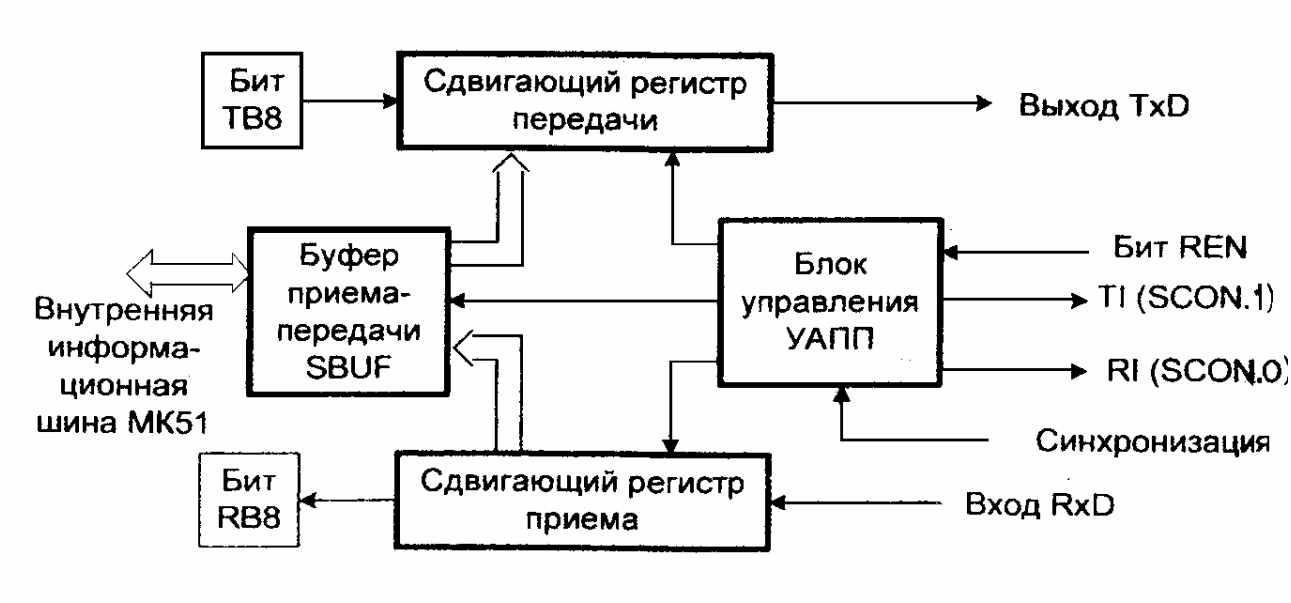
Имя бита	Номер бита	Функция
TF1	TCON.7	Флаг переполнения Таймера 1. Устанавливается при переходе счетного регистра таймера из состояния FFH в состояние 00H. Очищается при передаче управления на процедуру обработки прерывания.
TR1	TCON.6	Бит запуска Таймера 1. При TR1=1 счет разрешен.
TF0	TCON.5	Флаг переполнения Таймера 0. Устанавливается при переходе счетного регистра таймера из состояния FFH в состояние 00H. Очищается при передаче управления на процедуру обработки прерывания.
TR0	TCON.4	Бит запуска Таймера 0. При TR0=1 счет разрешен.
IE1	TCON.3	Флаг запроса прерывания по вход INT1#.
IT1	TCON.2	Бит селектора типа активного сигнала на входе INT1#. При IT1=1 активным является переход "1" – "0", при IT1=0 активным является низкий уровень сигнала.
IE0	TCON.1	Флаг запроса прерывания по вход INT0#.
IT0	TCON.0	Бит селектора типа активного сигнала на входе INT0#. При IT0=1 активным является переход "1" – "0", при IT0=0 активным является низкий уровень сигнала.

TMOD	GATE1	C/T1#	M1.1	M1.0	GATE0	C/T0#	M0.1	M0.0

Имя бита	Номер бита	Функция
GATE1	TMOD.7	Бит управления Таймером 1. При GATE1=1 Таймер 1 работает всегда при TR1=1. При GATE1=0 для работы необходимо условие TR1=1 и INT1#=1.
C/T1#	TMOD.6	Бит выбора типа событий для Таймера 1. При C/T1#=1 он работает как счетчик, при C/T1#=0 как таймер.
M1.1	TMOD.5	Бит 1 определения режима работы Таймера 1.
M1.0	TMOD.4	Бит 0 определения режима работы Таймера 1.
GATE0	TMOD.3	Бит управления Таймером 0. При GATE0=1 Таймер 0 работает всегда при TR0=1. При GATE0=0 для работы необходимо условие TR0=1 и INT0#=1.
C/T0#	TMOD.2	Бит выбора типа событий для Таймера 0. При C/T0#=1 он работает как счетчик, при C/T0#=0 как таймер.
M1.0	TMOD.1	Бит 1 определения режима работы Таймера 0.
M0.0	TMOD.0	Бит 0 определения режима работы Таймера 0.

Биты M1 и M0 следующим образом определяют режимы работы таймеров/счетчиков:

M1	M0	Режим работы
0	0	**Режим 0.** THx как 8-разрядный таймер/счетчик. TLx как 5-разрядный п редделитель
0	1	**Режим 1.** 16-разрядный таймер/счетчик. THx и TLx включены последовательно
1	0	**Режим 2.** 8-разрядный таймер/счетчик TLx с автоперезагрузкой значением из THx
1	1	**Режим 3.** TL0 как 8-разрядный таймер/счетчик, управляемый битами управления Таймера 0. TH0 как 8-разрядный таймер/счетчик, управляемый битами управления Таймера 1. Таймер 1 не работает.


Бит TB8
Сдвигающий регистр передачи
Выход TxD
Буфер приема-передачи SBUF
Внутренняя информа-ционная шина МК51
Блок управления УАПП
Бит REN
TI (SCON.1)
RI (SCON.0)
Синхронизация
Бит RB8
Сдвигающий регистр приема
Вход RxD

Организация регистра управления/состояния последовательного порта SCON

Символическое обозначение	Позиционное обозначение	Название и назначение
SM0 SM1	SCON.7 SCON.6	Биты управления режимом последовательного порта. Программно устанавливаются и сбрасываются. SM0=0, SM1=0 - режим расширения ввода/вывода (сдвиговый регистр); SM0=0, SM1=1 - 8-битный последовательный асинхронный порт, скорость передачи определяется частотой таймера/счетчика 1; SM0=1, SM1=0 - 9-битный последовательный асинхронный порт, скорость передачи: fgQ/64 или fgQ/32; SM0=1, SM1=1 - 9-битный последовательный асинхронный порт, скорость передачи определяется частотой таймера/счетчика 1.
SM2	SCON.5	Бит 2 управления режимом последовательного порта. Программно устанавливается для запрета приема кадров с нулевым 8-м битом.
REN	SCON.4	Бит управления разрешением приема. Программно устанавливается и сбрасывается для разрешения/запрета приема последовательных данных.
TB8	SCON.3	8-й бит при передаче данных. Программно устанавливается и сбрасывается для задания 8-го бита в режиме 9-битного последовательного порта.
RB8	SCON.2	8-й бит при приеме данных. Аппаратно устанавливается и сбрасывается в зависимости от значения 8-го бита в режиме 9-битного последовательного порта.
TI	SCON.1	Флаг прерывания при окончании передачи. Устанавливается аппаратно после передачи байта, сбрасывается программно после обслуживания прерывания.
RI	SCON.0	Флаг прерывания при окончании приема. Устанавливается аппаратно после приема байта, сбрасывается программно после обслуживания прерывания.

Организация регистра управления потреблением PCON.

Символическое обозначение	Позиционное обозначение	Название и назначение
SMOD	PCON.7	Бит управления (удвоения) скорости приема/передачи. При SMOD=1 скорость приема/передачи удваивается.
	PCON.6	Не используется.
	PCON.5	Не используется.
	PCON.4	Не используется.
GF1	PCON.3	Флаг пользователя. Программно доступен по записи и чтению.
GF0	PCON.2	Флаг пользователя. Программно доступен по записи и чтению.
PD	PCON.1	Бит установки режима микропотребления. Этот бит имеет высший приоритет по отношению к биту IDL.
IDL	PCON.0	Бит установки режима холостого хода.

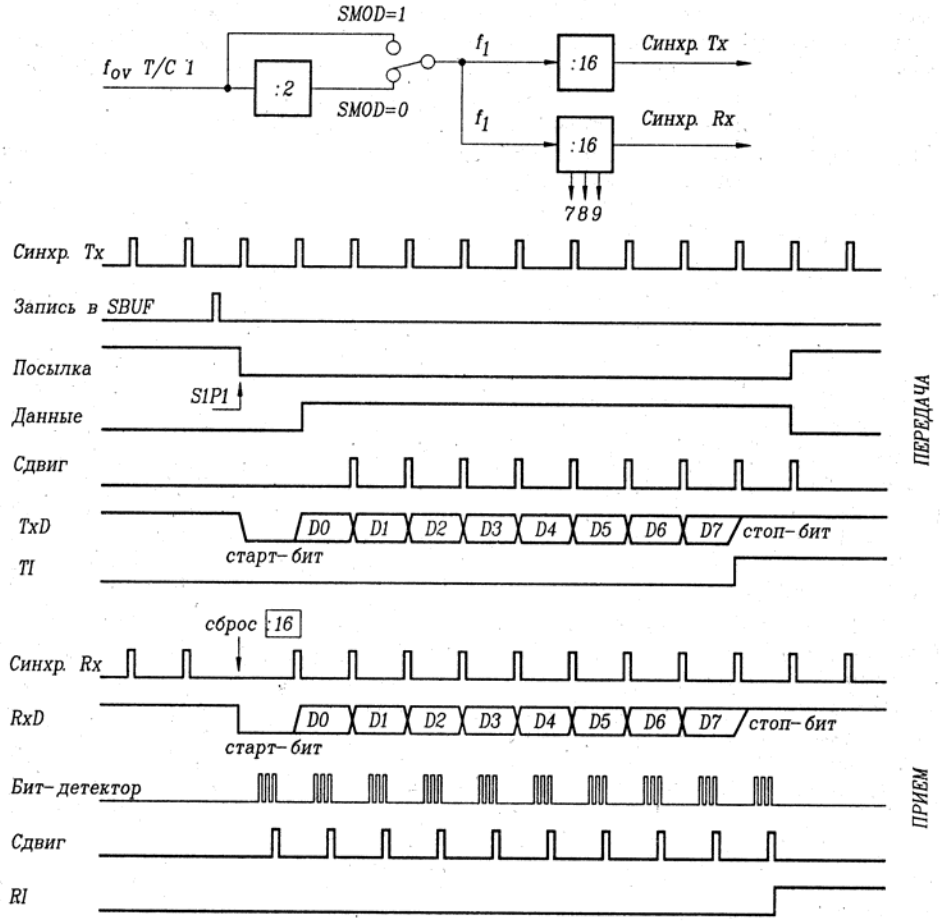
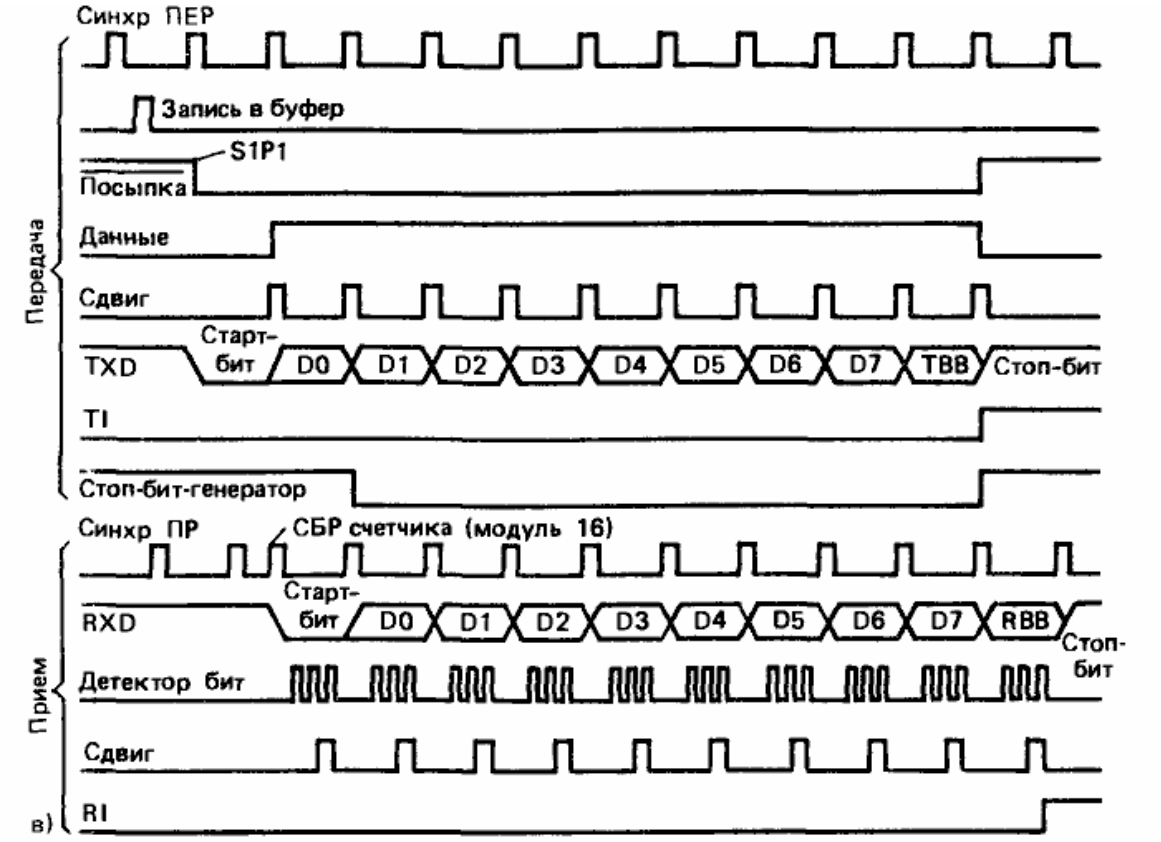


Рис. 6. Работа последовательного порта в режиме 1


Синхр ПЕР
Запись в буфер
S1P1
Посылка
Данные
Сдвиг
Старт-бит
D0
D1
D2
D3
D4
D5
D6
D7
TBB
Стоп-бит
TXD
TI
Стоп-бит-генератор
Передача
Синхр ПР
СБР счетчика (модуль 16)
Старт-бит
RXD
RBB
Стоп-бит
Детектор бит
Сдвиг
RI
Прием
в)

1. СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА МК51

В машинном коде команда занимает один, два или три байта. Все команды выполняются за один или два машинных цикла, команды умножения и деления - за четыре. Один машинный цикл при частоте тактового генератора 12 МГц составляет 1 мкс.

Все команды условных переходов осуществляются относительно содержимого счетчика команд. Адрес перехода вычисляется микроконтроллером во время выполнения команды.

Систему команд МК51 условно можно разбить на пять групп:

арифметические команды;

логические команды;

команды передачи данных;

команды битового процессора;

команды ветвления программ и передачи управления.

Способы адресации операндов:

регистровая адресация;

прямая адресация;

косвенно-регистровая адресация;

непосредственная;

косвенная адресация по сумме базового и индексного регистра.

Описание каждой команды микроконтроллеров семейства МК51 состоит из предложения на языке ассемблера, кода, длины команды в байтах, времени ее выполнения, алгоритма и примера. При описании операций используются обозначения, приведенные в таблице.

Обозначение, символ	Назначение
1	2
A	Аккумулятор
Rr	Регистры текущего банка регистров общего назначения
r	Номер загружаемого регистра, указанного в команде
direct	Прямо адресуемый 8-битовый адрес, который может быть ячейкой внутреннего ОЗУ (00H-7FH) или регистром специальных функций (80H-FFH)
@Rr	Косвенно адресуемая 8-битовая ячейка внутреннего ОЗУ
data 8	8-битовое непосредственное данное
data 16	16-битовое непосредственное данное
data H	Старшие биты (15-8) непосредственных 16-битовых данных
data L	Младшие биты (7-0) непосредственных 16-битовых данных
addr 11	11-битовый адрес назначения
addr 16	16-битовый адрес назначения
addr L	Младшие биты адреса назначения

Продолжение таблицы

1	2
bit	Бит с прямой адресацией, находящийся во внутреннем ОЗУ или в регистрах специальных функций
a15,a14...a0	Биты адреса назначения
(X)	Содержимое элемента X
((X))	Содержимое по адресу хранящемуся в элементе X
(X)[M]	Разряд M элемента X
(X)[M1-M2]	Группа разрядов M1-M2 элемента X
+	Операция сложения
-	Операция вычитания
*	Операция умножения
/	Операция деления
AND	Операция логического умножения (операция «И»)
OR	Операция логического сложения (операция «ИЛИ»)
XOR	Операция сложения по модулю 2 (операция «Исключающее ИЛИ»)
/X	Инверсия элемента X

Команда ACALL <addr 11>. Команда "абсолютный вызов подпрограммы" вызывает, безусловно, подпрограмму, размещенную по указанному адресу. При этом счетчик команд увеличивается на 2 для получения адреса следующей команды, после чего полученное 16-битовое значение PC помещается в стек (сначала следует младший байт), и содержимое указателя стека также увеличивается на два. Адрес перехода получается с помощью конкатенации старших бит увеличенного содержимого счетчика команд, битов старшего байта команды и младшего байта команды.

Ассемблер: ACALL <метка>

Код	A10 A9 A8 1	0 0 0 1	A7 A6 A5 A4	A3 A2 A1 A0

Время: 2 цикла

Алгоритм: (PC):=(PC)+2, (SP):=(SP)+1, ((SP)):=(PC[7-0]), (SP):=(SP)+1, ((SP)):=(PC[l5-8]),
(PC[10-0]):=A10 A9 A8 || A7 A6 A5 A4 A3 A2 A1 A0, где || - знак конкатенации (сцепление)

Пример: ;(SP)=07H, метка MT1 соответствует адресу: 0345H,
(PC)=02F8H
ACALL MT1 ;(PC)=028DH,(SP)=09H, (PC)=0345H, ОЗУ [08]=8FH,
ОЗУ [09]=02H.

Команда ADD A, <байт-источник>. Эта команда ("сложение") складывает содержимое аккумулятора A с содержимым байта-источника, оставляя результат в аккумуляторе. При появлении переносов из разрядов 7 и 3, устанавливаются флаги переноса (C) и дополнительного переноса (AC) соответственно, в противном случае эти флаги сбрасываются. При сложении

целых чисел без знака флаг переноса "С" указывает на появление переполнения. Флаг переполнения (OV) устанавливается, если есть перенос из бита 6 и нет переноса из бита 7, или есть перенос из бита 7 и нет - из бита 6, в противном случае флаг OV сбрасывается. При сложении целых чисел со знаком флаг OV указывает на отрицательную величину, полученную при суммировании двух положительных операндов или на положительную сумму для двух отрицательных операндов.

Для команды сложения разрешены следующие режимы адресации байта-источника:

1) регистровый;
2) косвенно-регистровый;
3) прямой;
4) непосредственный.

Ассемблер: 1) ADD A.Rn ;где n=0-7

Код	0 0 1 0	1 r r r

где rrr=000-111

Время: 1 цикл

Алгоритм: (A):=(A)+(Rn) ;где n=0-7
C:=X, OV:=X, AC:=X ;где X=(0 или 1)

Ассемблер: 2) ADD A,@Ri ;где i=0,1

Код	0 0 1 0	0 1 1 i

Время: 1 цикл

Алгоритм: (A):=(A)+((Ri)) ;где i=0,1
C:=X, OV:=X, AC:=X ;где X=(0 или 1)

Ассемблер: 3) ADD A,<direct>

Код	0 0 1 0	0 1 0 1	direct address

Время: 1 цикл

Алгоритм: (A):=(A)+(direct)
C:=X, OV:=X, AC:=X ;где X=(0 или 1)

Ассемблер: 4) ADD A,<#data>

Код	0 0 1 0	0 1 0 0	#data

Время: 1 цикл

Алгоритм: (A):=(A)+#data
C:=X, OV:=X, AC:=X ;где X=(0 или 1)

Примеры:

1) ;(A)=C3H, (R6)=AAH
ADD A,R6 ;(A)=6DH, (R6)=AAH ,(AC)=0, (C)=l, (OV)=1
2) ;(A)=95H, (R1)=31H, (ОЗУ [31])=4CH
ADD A,@R1 ;(A)=E1H, (ОЗУ [31])=4CH, (C)=0, (AC)=1, (OV)=0

```
3)                      ;(A)=77H, (ОЗУ [90])=FFH
      ADD A,90H         ;(A)=76H, (ОЗУ (90])=FFH, ;(C)=1, (OV)=0,
                        ; (AC)=1
4)                      ;(A)=09H
      ADD A,#0D3H       ;(A)=DCH, (C)=0, (OV)=0, (AC)=0
```

Команда ADDC A. <байт-источник>. Эта команда ("сложение с переносом") одновременно складывает содержимое байта-источника, флаг переноса и содержимое аккумулятора А, оставляя результат в аккумуляторе. При этом флаги переноса и дополнительного переноса устанавливаются, если есть перенос из бита 7 или бита 3, и сбрасываются в противном случае. При сложении целых чисел без знака флаг переноса указывает на переполнение. Флаг переполнения (OV) устанавливается, если имеется перенос бита 6 и нет переноса из бита 7 или есть перенос из бита 7 и нет - из бита 6, в противном случае OV сбрасывается. При сложении целых чисел со знаком OV указывает на отрицательную величину, полученную при суммировании двух положительных операндов, или на положительную сумму от двух отрицательных операндов. Для этой команды разрешены следующие режимы адресации байта-источника:

1) регистровый;
2) косвенно-регистровый;
3) прямой;
4) непосредственный.

Ассемблер: 1) ADDC A,Rn ;где n=0-7

Код

0 0 1 1	1 r r r

где rrr=000-111

Время: 1 цикл

Алгоритм: (A):=(A)+(C)+(Rn)
(C):=X, (AC):=X, (OV):=X ;где X=(0 или 1)

Ассемблер: 2) ADDC A,@Ri ;где i=0,l

Код

0 0 1 1	0 1 1 i

Время: 1 цикл

Алгоритм: (A):=(A)+(C)+((Ri))
(C):=X, (AC):=X, (OV):=X, где X=(0 или 1)

Ассемблер: 3) ADDC A,<direct>

Код

0 0 1 1	0 1 0 1	direct address

Время: 1 цикл

Алгоритм: (A):=(A)+(C)+(direct)
(C):=X, (AC):=X, (OV):=X ;где X=(0 или 1)

Ассемблер: 4) ADDC A,#data

Код

0 0 1 0	0 1 0 1	#data

Время: 1 цикл
Алгоритм: (A):=(A)+(C)+(direct)
(C):=X, (AC):=X, (OV):=X ;где X=(0 или 1)

Примеры:

```
1)                      ;(A)=B2H, (R3)=99, (C)=1
      ADDC A,R3         ;(A)=4CH, (R3)=99, (C)-1, (AC)=0, (OV)=1
2)                      ;(A)=D5H, (R0)=3AH, (ОЗУ[ЗА])=1АН, (С)=l
      ADDC A,@R0        ;(A)=F0H, (ОЗУ[ЗА]=1АН), ;(C)=0, (AC)=1,
                        ; (OV)=0
3)                      ;(A)=11H, (ОЗУ[80])=DFH, (С)=l
      ADDC A,80H        ;(A)=F1H, (C)=0, (AC)=1, (OV)=0
4)                      ;(A)=55H, (C)=0
      ADDC A,#55H       ;(A)=AAH, (C)=0, (AC)=0, (OV)=1
```

Команда AJMP <addr11>. Команда "абсолютный переход", передает управление по указанному адресу, который получается при конкатенации пяти старших бит счетчика команд PC (после увеличения его на два), 7-5 битов кода операции и второго байта команды. Адрес перехода должен находится внутри одной страницы объемом 2 Кбайт памяти программы, определяемой пятью старшими битами счетчика команд.

Ассемблер: AJMP <метка>

Код	A10 A9 A8 0	0 0 0 1	A7 A6 A5 A4	A3 A2 A1 A0

Время: 2 цикла
Алгоритм: (PC[15-0]):=(PC[15-0])+2, (PC[10-0]):=<addr11>

```
Пример:                 ;(PC)=028FH, Метке MT2 соответствует адрес
                        ;034AH
      AJMP MT2          ;(PC)=034AH
```

Команда ANL <байт-назначения>, <байт-источник>. Команда "логическое "И" для переменных-байтов" выполняет операцию логического "И" над битами указанных переменных и помещает результат в байт-назначения. Эта операция не влияет на состояние флагов. Два операнда обеспечивают следующие комбинации шести режимов адресации:

байтом назначения является аккумулятор (A):

1) регистровый;
2) прямой;
3) косвенно-регистровый;
4) непосредственный;

байтом назначения является прямой адрес (direct):

5) прямой аккумуляторный;
6) непосредственный (байт-источник равен константе).

Ассемблер: 1) ANL A, Rn ;где n=0-7

Код | 0 1 0 1 | 1 r r r |

где rrr=000-111

Время: 1 цикл

Алгоритм: (A):=(A) AND (Rn)

Ассемблер: 2) ANL A, <direct>

Код | 0 1 0 1 | 0 1 0 1 | direct address |

Время: 1 цикл

Алгоритм: (A):=(A) AND (direct)

Ассемблер: 3) ANL A, @Ri; где i=0,l

Код | 0 1 0 1 | 0 1 1 i |

Время: 1 цикл

Алгоритм: (A):=(A) AND ((Ri))

Ассемблер: 4) ANL A, #data

Код | 0 1 0 1 | 0 1 0 0 | #data |

Время: 1 цикл

Алгоритм: (A):=(A) AND #data

Ассемблер: 5) ANL <direct>, A

Код | 0 1 0 1 | 0 0 1 0 | direct address |

Время: 1 цикл

Алгоритм: (direct):=(direct) AND (A)

Ассемблер: 6) ANL <direct>,#data

Код | 0 1 0 1 | 0 0 1 0 | direct address | #data8 |

Время: 2 цикла

Алгоритм: (direct):=(direct) AND #data

Примеры:

```
1)                      ;(A)=FEH, (R2)=C5H
      ANL  A,R2         ;(A)=C4H, (R2)=C5H
2)                      ;(A)=A3H, (PSW)=86H
      ANL  A,PSW        ;(A)-82H, (PSW)=86H
3)                      ;(A)=BCH, (ОЗУ[35])=47H, (R0)=35H
      ANL  A,@R0        ;(A)=04H, (ОЗУ[351)=47H
4)                      ;(A)=36H
      ANL  A,#0DDH      ;(A)=14H
5)                      ;(A)=55H, (P2)=AAH
      ANL  P2,A         ;(P2)=00H, (A)=55H
6)                      ;(P1)=FFH
      ANL  P1,#73H      ;(P1)=73H
```

Примечание. Если команда "ANL" применяется для изменения содержимого порта, то значение, используемое в качестве данных порта, будет считываться из "защелки" порта, а не с выводов БИС.

Команда ANL C,<бит источника>. Команда "логическое "И" для переменных-битов", выполняет операцию логического "И" над указанными битами. Если бит-источник равен "0", то происходит сброс флага переноса, в противном случае флаг переноса не изменяет текущего значения. "/" перед операндом в языке ассемблера указывает на то, что в качестве значения используется логическое отрицание адресуемого бита, однако сам бит источника при этом не изменяется. На другие флаги эта команда не влияет. Для операнда-источника разрешена только прямая адресация к битам.

Ассемблер: 1) ANL C,<bit>

Код	1 0 0 0	0 0 1 0	bit address

Время: 2 цикла

Алгоритм: (C):=(C) AND (bit)

Ассемблер: 2) ANL C,</bit>

Код	1 0 1 1	0 0 0 0	bit address

Время: 2 цикла

Алгоритм: (C):=(C) AND (/bit)

Примеры:

```
1)                      ;(C)=1, P1[0]=0
      ANL C,P1.0        ;(C)=0, P1[0]=0
2)                      ;(C)=1, (AC)=0
      ANL C,/AC         ;(C)=1, (AC)=0
```

Команда CJNE <байт назначения>, <байт источник>, <смешение>. Команда "сравнение и переход, если не равно" сравнивает значения первых двух операндов и выполняет ветвление, если операнды не равны. Адрес перехода (ветвления) вычисляется при помощи сложения значения (со знаком), указанного в последнем байте команды, с содержимым счетчика команд после увеличения его на три.

Флаг переноса "C" устанавливается в "1", если значение целого без знака <байта назначения> меньше, чем значение целого без знака <байта источника>, в противном случае перенос сбрасывается (если значения операндов равны, флаг переноса сбрасывается). Эта команда не оказывает влияния на-1 операнды.

Операнды, стоящие в команде, обеспечивают комбинации четырех режимов 1 адресации:

если байтом-назначения является аккумулятор:

1) прямой;
2) непосредственный,

если байтом-назначения является любая ячейка ОЗУ с косвенно-регистровой или регистровой адресацией:

3) непосредственный к регистровому;
4) непосредственный к косвенно-регистровому

Ассемблер: 1) CJNE A, <direct>, <метка>

Код	1 0 1 1	0 1 0 1	direct address	rel8

Время: 2 цикла

Алгоритм: (PC):=(PC)+3,
если (direct) < (A) то (PC):=(PC)+<rel8>, C:=0;
если (direct) > (A), то (PC):=(PC)+<rel8>, C:=1;

Ассемблер: 2) CJNE A,#data,<метка>

Код	1 0 1 1	0 1 0 0	#data8	rel8

Время: 2 цикла

Алгоритм: (PC):=(PC)+3,
если #data8<(A), то (PC)+<re18>,C:=0;
если #data8>(A),то (PC):=(PC)+<rel8>, C:=1

Ассемблер: 3) CJNE Rn,#data,<метка> ;где n=0-7

Код	1 0 1 1	1 r r r	#data8	rel8

Время: 2 цикла

Алгоритм: (PC):=(PC)+3,
если #data8<(Rn), то (PC):=(PC)+<rel8>, C:=0;
если #data8>(Rn), то (PC)+<rel8>, C:=1

Ассемблер: 4) CJNE @Ri,#data, <метка> ;где i=0,l

Код	1 0 1 1	0 1 1 i	#data8	rel8

Время: 2 цикла.

Алгоритм: (PC):=(PC)+3,
если #data<((Ri)), то (PC)+<rel8>,C:=0
если #data8>((Ri)), то (PC)+<rel8>, C:=1

Примеры:

```
1)                      ;(A)=97H, (P2)=F0H, (C)=0
      CJNEA,P2,MT3
MT3: CLR  A             ;(A)=97H, (P2)=F0H, (C)=l. Адрес,
                        ;соответствующий метке MT3
                         ;вычисляется, как (PC):=(PC)+3+(re18)
2)                      ;(A)=FCH, (C)=1
      CJNEA,#0BFH,MT4
MT4: INC   A            ;(A)=FDH, C-0, (PC):=(PC)+3+(rel8)
3)                      ;(R7)=80H, (C)=0
      CJNER7,#81H,MT5
MT5: NOP                ;(R7)=80H, (C)=l, (PC):=(PC)+3+(rel8)
4)                      ;(R0)=41H, (C)=l, (ОЗУ[41])=57H
      CJNE@R0,#29H,MT6
MT6: DEC  R0            ;(ОЗУ[41])=57H, (C)=0, (PC):=(PC)+3+(rel8)
```

Команда CLR A. Команда "сброс аккумулятора" сбрасывает (обнуляет) содержимое аккумулятора А. На флаги команда не влияет.

Ассемблер: CLR A

Код	1 1 1 0	0 1 0 0

Время: 1 цикл.
Алгоритм: (A):=0
Пример: ;(A)=6DH, (C)=0, (AC)=1
CLR A ;(A)=00H, (C)=0, (AC)=1

Команда CLR <bit>. Команда "сброс бита" сбрасывает указанный бит в нуль. Эта команда работает с флагом переноса "С" или любым битом с прямой адресацией.

Ассемблер: 1) CLR C

Код	1 1 0 0	0 0 1 1

Время: 1 цикл
Алгоритм: (C):=0

Ассемблер: 2) CLR <bit>

Код	1 1 0 0	0 0 1 0	bit address

Время: 1 цикл
Алгоритм: (bit) :=0

Примеры:
1) ;(C)=l
CLR C ;(C)=0
2) ;(P1)=5EH (01011110B)
CLR P1.3 ;(P1)=56H (01010110B)

Команда CPL A. Команда "инверсия аккумулятора" каждый бит аккумулятора инвертирует (изменяет на противоположный). Биты, содержащие "единицы", после этой команды будут содержать "нули", и наоборот. На флаги эта операция не влияет.

Ассемблер: CPL A

Код	1 1 1 1	0 1 0 0

Время: 1 цикл
Алгоритм: (A):=/(A)
Пример: ;(A)=65H (01100101B)
CPL A ;(A)=9AH (10011010B)

Команда CPL <bit>. Команда "инверсия бита" инвертирует (изменяет на противоположное значение) указанный бит. Бит, который был "единицей", изменяется в "нуль" и наоборот. Команда CPL может работать с флагом

переноса или с любым прямо адресуемым битом. На другие флаги команда не влияет.

Ассемблер: 1) CPL <bit>

Код	1 0 1 1	0 0 1 0	bit address

Время: 1 цикл
Алгоритм: (bit):=/(bit)

Ассемблер: 2) CPL C

Код	1 0 1 1	0 0 1 1

Время: 1 цикл
Алгоритм: (C):=/(C)

Пример:

```
1)                      ;(P1)=39H (00111001B)
      CPL  P1.1
      CPL  P1.3         ;(P1)=ЗЗH (00110011B)
2)                      ;(C)=0, (AC)=1, (OV)=0
      CPL  C            ;(C)=1, (AC)=1, (OV)=0
```

Примечание: Если эта команда используется для изменения информации на выходе порта, значение, используемое как исходные данные, считывается из "защелки" порта, а не с выводов БИС.

Команда DA A. Команда "десятичная коррекция аккумулятора для сложения" упорядочивает 8-битовую величину в аккумуляторе после выполненной ранее команды сложения двух переменных (каждая в упакованном двоично-десятичном формате). Для выполнения сложения может использоваться любая из типов команд ADD или ADDC. Если значение битов 3 - 0 аккумулятора (A) превышает 9 (XXXX 1010 - XXXX 1111) или если флаг AC равен "1", то к содержимому (A) прибавляется 06, получая соответствующую двоично-десятичную цифру в младшем полубайте. Это внутреннее побитовое сложение устанавливает флаг переноса, если перенос из поля младших четырех бит распространяется через все старшие биты, а в противном случае - не изменяет флаг переноса. Если после этого флаг переноса равен "1", или если значение четырех старших бит (7-4) превышает 9 (1010 XXXX - 1111 XXXX), значения этих старших бит увеличивается на 6, создавая соответствующую двоично-десятичную цифру в старшем полубайте. И снова при этом флаг переноса устанавливается, если перенос получается из старших битов, но не изменяется в противном случае. Таким образом, флаг переноса указывает на то, что сумма двух исходных двоично-десятичных переменных больше чем 100. Эта команда выполняет десятичное преобразование с помощью сложения 06, 60, 66 с содержимым аккумулятора в зависимости от начального состояния аккумулятора и слова состояния программы (PSW).

Ассемблер: DA A

Код	1 1 0 1	0 1 0 0

Время: 1 цикл

Алгоритм: если ((A[3-0])>9 или (AC)=1), то A[3-0]:=A[3-0]+6
если ((A[7-4])>9 или (C)=1), то A[7-4]:=A[7-4]+6

Примеры:

```
1)                      ;(A)=56H, (R3)=67H, (C)=1
   ADDC    A,R3
   DA   A               ;(A)=24H, (R3)=67H, (C)=1
2)                      ;(A)=30H, (C)=0
   ADD A,#99H
   DA   A               ;(A)=29, (C)-=l
```

Примечание: Команда DA A не может просто преобразовать шестнадцатеричное значение в аккумуляторе в двоично-десятичное представление и не применяется, например, для десятичного вычитания.

Команда DEC <байт>. Команда "декремент" производит вычитание "1" из указанного операнда. Начальное значение 00H перейдет в 0FFH. Команда DEC не влияет на флаги. Этой командой допускается четыре режима адресации операнда:

1) к аккумулятору;
2) регистровый;
3) прямой;
4) косвенно-регистровый;

Ассемблер: 1) DEC A

Код	0 0 0 1	0 1 0 0

Время: 1 цикл

Алгоритм: (A):=(A)-1

Ассемблер: 2) DEC Rn ;где n=0-7

Код	0 0 0 1	1 r r r

где rrr=000-111

Время: 1 цикл

Алгоритм: (Rn):=(Rn)-l

Ассемблер: 3) DEC <direct>

Код	0 0 0 1	0 1 0 1	direct address

Время: 1 цикл

Алгоритм: (direct):=(direct)-l

Ассемблер: 4) DEC @Ri ;где i=0,l

Код	0 0 0 1	0 1 1 i

Время: 1 цикл

Алгоритм: ((Ri)):=((Ri)-l)

Примеры:

```
1)                  ;(A)=11H, (C)=1, (AC)=1
      DEC  A        ;(A)=10H, (C)=1, (AC)=1
2),4)               ;(R1)=7FH, (ОЗУ[7F])=40H, (ОЗУ[7F])=00H
      DEC  @R1
      DEC  R1
      DEC  @R1      ;(R1)=7EH, (ОЗУ[7F])=3EH, (03У[7F])=FFH
3)                  ;(SCON)=A0H, (C)=l, (AC)=1
      DEC  SCON     ;(SCON)=9FH, (C)=l, (AC)=1
```

Примечание: Если эта команда используется для изменения информации на выходе порта, значение, используемое как исходные данные, считывается из "защелки'" порта, а не с выводов БИС.

Команда DIV AB. Команда "деление" делит 8-битовое целое без знака из аккумулятора А на 8-битовое целое без знака в регистре В. Аккумулятору присваивается целая часть частного (старшие разряды), а регистру В - остаток. Флаги переноса (С) и переполнения (0V) сбрасываются. Если (А) < (В), то флаг дополнительного переноса (АС) не сбрасывается. Флаг переноса сбрасывается в любом случае.

Ассемблер: DIV AB

Код	1 0 0 0	0 1 0 0

Время: 4 цикла

Алгоритм: (A):=((A)/(B)) [15-8], (B):=((A)/(B)) [7-0]

Пример: Пусть аккумулятор содержит число 251 (0FBH или 11111011B), а регистр В - число 18 (12Н или 00010010В). После выполнения команды DIV AB в аккумуляторе будет число 13 (0DH или 00001101B), а в регистре В - число 17 (11Н или 00010001В), т.к. 251=(13*18)+17. Флаги С и OV будут сброшены.

Примечание: Если В содержит 00Н, то после команды DIV содержимое аккумулятора А и регистра В будут не определены. Флаг переноса сбрасывается, а флаг переполнения устанавливается в 1.

Команда DJNZ <байт>, <смещение>. Команда "декремент и переход, если не равно нулю" выполняет вычитание "1" из указанной ячейки и осуществляет ветвление по вычисляемому адресу, если результат не равен нулю. Начальное значение 00Н перейдет в 0FFH. Адрес перехода (ветвления) вычисляется сложением значения смещения (со знаком), указанного в последнем байте команды, с содержимым счетчика команд, увеличенным на длину команды DJNZ. На флаги эта команда не влияет и допускает следующие режимы адресации:

1) регистровый
2) прямой

Ассемблер: 1) DJNZ Rn,<метка>;где n=0-7

Код	1 1 0 1	1 r r r	rel8

Время: 2 цикла

Алгоритм: (PC):=(PC)+2, (Rn):=(Rn)-l, если ((Rn)>0 или (Rn)<0), то (PC):=(PC)+<re18>

Ассемблер: 2) DJNZ<direct>,<метка>

Код	1 1 0 1	0 1 0 1	direct address	rel8

Время: 2 цикла

Алгоритм: (PC):=(PC)+3, (direct) :=(direct)-l, если ((direct)>0 или (direct)<0), то (PC):=(PC)+<rel8>

Примеры:

```
1)                              ;(R2)=08H, (P1)=FFH (11111111B)
LAB4:   CPL   P1.7
        DJNZ  R2,LAB4           ;(R2)={07-00}. Эта последовательность
                                ;команд переключает P1.7
                                ;восемь раз и приводит к появлению
                                ;четырех импульсов ;на выводе
                                ;БИС, соответствующем биту P1.7.
2)                              ;(ОЗУ40)=01H, (ОЗУ[50])=80H, (ОЗУ 60)=
                                ;=25H
        DJNZ  40H,LAB1          ;(ОЗУ[40]):=00H
        DJNZ  50H,LAB2          ;(ОЗУ[50]):=7FH
        DJNZ  60H,LAB3          ;(ОЗУ[60]):=25H
        . . .  . . .
LAB1:   CLR   A
        . . .  . . .
LAB2:   DEC   R1                ;осуществился переход на метку LAB2
```

Примечание: Если команда DJNZ используется для изменения выхода порта, значение, используемое как операнд, считывается из "защелки" порта, а не с выводов БИС.

Команда INC <байт>. Команда "инкремент" выполняет прибавление "1" к указанной переменной и не влияет на флаги. Начальное значение 0FFH перейдет в 00H. Эта команда допускает четыре режима адресации:

1) к аккумулятору;
2) регистровый;
3) прямой;
4) косвенно-регистровый.

Ассемблер: 1) INC A

Код	0 0 0 0	0 1 0 0

Время: 1 цикл

Алгоритм: (A):=(A)+1

Ассемблер: 2) INC Rn ;где n=0-7

Код	0 0 0 0	1 r r r

где rrr=000-111
Время: 1 цикл
Алгоритм: (Rn):=(Rn)+1

Ассемблер: 3) INC <direct>

Код	0 0 0 0	0 1 0 1	direct address

Время: 1 цикл
Алгоритм: (direct):=(direct)+1

Ассемблер: 4) INC @Ri ;где i=0,l

Код	0 0 0 0	0 1 1 i

Время: 1 цикл
Алгоритм: ((Ri)):=((Ri))+1

Примеры:

```
1)                      ;(A)=1FH, (AC)=0
      INC  A            ;(A)=20H, (AC)=0
2)                      ;(R4)=FFH, (C)=0, (AC)=0
      INC  R4           ;(R4)=00H, (C)=0, (AC)=0
3)                      ;(ОЗУ[43])=22H
      INC  43H          ;(ОЗУ[43])=23H
4)                      ;(R1)=41H, (ОЗУ[41])=4FH, (AC)=0
      INC  @RI          ;(R1)=41H, (ОЗУ[41])=50H, (AC)=0
```

Примечание: При использовании команды INC для изменения содержимого порта величина, используемая как операнд, считывается из "защелки" порта, а не с выводов БИС.

Команда INC DPTR. Команда "инкремент указателя данных" выполняет инкремент (прибавление "1") содержимого 16-битового указателя данных (DPTR). Прибавление "1" осуществляется к 16 битам, причем переполнение младшего байта указателя данных (DPL) из FFH в 00H приводит к инкременту старшего байта указателя данных (DPH). На флаги эта команда не влияет.

Ассемблер: INC DPTR

Код	1 0 1 0	0 0 1 1

Время: 2 цикла
Алгоритм: (DPTR):=(DPTR)+1

```
Пример:                 ;(DPH)=12H, (DPL)=FEH
      INC  DPTR
      INC  DPTR
      INC  DPTR         ;(DPH)=13H, (DPL)=01H
```

Команда JB <bit>,<rel8>. Команда "переход, если бит установлен" выполняет переход по адресу ветвления, если указанный бит равен" 1", в противном случае выполняется следующая команда. Адрес ветвления вычисляется с помощью прибавления относительного смещения со знаком в третьем байте команды (re18) к содержимому счетчика команд после прибавления к нему 3. Проверяемый бит не изменяется. Эта команда на флаги не влияет.

Ассемблер: JB (bit),<метка>

Код	0 0 1 0	0 0 0 0	bit address	rel8

Время: 2 цикла

Алгоритм: (PC):=(PC)+3, если (bit)=l, то (PC):=(PC)+<rel8>

```
Пример:                          ;(A)=96H (10010110B)
        JB    ACC.2,LAB5         ;эта команда обеспечивает переход на метку
                                 ;LAB5
        . . .   . . .
LAB5:   INC   A
```

Команда JBC <bit>,<rel8>. Команда "переход, если бит установлен и сброс этого бита", выполняет ветвление по вычисляемому адресу, если бит равен "1". В противном случае выполняется следующая за JBC команда. В любом случае указанный бит сбрасывается. Адрес перехода вычисляется сложением относительного смещения со знаком в третьем байте команды (rel8) и содержимого счетчика команд после прибавления к нему 3. Эта команда не влияет на флаги.

Ассемблер: JBC (bit),<метка>

Код	0 0 0 1	0 0 0 0	bit address	rel8

Время: 2 цикла

Алгоритм: (PC):=(PC)+3, если (bit)=1, то (bit):=0, (PC):=(PC)+<rel8>

```
Пример:                          ;(A)=76H (0111 0110B)
        JBC   ACC.3,LAB6         ;Перехода на LAB6 нет, т.к. (A[3])=0
        JBC   ACC.2,LAB7         ;(A)=72H (0111 0010B) и переход на адрес,
                                 ;соответствующий
                                 ;метке LAB7.
```

Примечание: Если эта команда используется для проверки бит порта, то значение, используемое как операнд, считывается из "защелки" порта, а не с вывода БИС.

Команда JC <rel8>. Команда "переход, если перенос установлен" выполняет ветвление по адресу, если флаг переноса равен "1", в противном случае выполняется следующая команда. Адрес ветвления вычисляется с помощью сложения относительного смещения со знаком во втором байте команды (re18) и содержимого счетчика команд после прибавления к нему 2.

Эта команда на флаги не влияет.

Ассемблер: JC <метка>

Код	0 1 0 0	0 0 0 0	rel8

Время: 2 цикла

Алгоритм: (PC):=(PC)+2, если (C)=1, то (PC):=(PC)+<re18>

```
Пример:                          ;(C)=0
        JC   LAB8                ;нет перехода на метку LAB8
        CPL  C                   ;(C):=1
LAB8:   JC   LAB9                ; переход на метку LAB9, т.к. (C)=1
        . . .  . . .
LAB9:   NOP
```

Команда JMP @A+DPTR. Команда "косвенный переход" складывает 8-битовое содержимое аккумулятора без знака с 16-битовым указателем данных (DPTR) и загружает полученный результат в счетчик команд, содержимое которого является адресом для выборки следующей команды. 16-битовое сложение выполняется по модулю 2^{16}, перенос из младших восьми бит распространяется на старшие биты программного счетчика. Содержимое аккумулятора и указателя данных не изменяется. Эта команда на флаги не влияет.

Ассемблер: JMP @A+DPTR

Код	0 1 1 1	0 0 1 1

Время: 2 цикла

Алгоритм: (PC):=(A)[7-0]+(DPTR) [15-0]

```
Пример:                          ;(PC)=034EH, (A)=86H, (DPTR)=0329H
        JMP  @A+DPTR             ;(PC)=03AFH, (A)=86H, (DPTR)=0329H
```

Команда JNB <bit>,<rel8>. Команда "переход, если бит не установлен" выполняет ветвление по адресу, если указанный бит равен "нулю", в противном случае выполняется следующая команда. Адрес ветвления вычисляется с помощью сложения относительного смещения со знаком в третьем байте команды (rel8) и содержимого счетчика команд после прибавления к нему 3. Проверяемый бит не изменяется. Эта команда на флаги не влияет.

Ассемблер: JNB (bit),<метка>

Код	0 0 1 1	0 0 0 0	bit address	rel8

Время: 2 цикла

Алгоритм: (PC):=(PC)+3, если (bit)=0, то (PC):=(PC)+<re18>

```
Пример:                          ;(P2)=CAH (11001010B), (A)=56H (0101
                                 ;0110B)
        JNB  P1.3,LAB10          ;нет перехода на LAB10
        JNB  ACC.3,LAB11         ;переход на метку LAB11
        . . .  . . .
LAB11:  INC  A
```

Команда JNC <rel8>. Команда "переход, если перенос не установлен" выполняет ветвление по адресу, если флаг переноса равен нулю, в противном случае выполняется следующая команда. Адрес ветвления вычисляется с помощью сложения относительного смещения со знаком во втором байте команды (re18) и содержимого счетчика команд после прибавления к нему 2. Флаг переноса не изменяется. Эта команда на другие флаги не влияет.

Ассемблер: JNC <метка>

Код	0 1 0 1	0 0 0 0	rel8

Время: 2 цикла

Алгоритм: (PC):=(PC)+2, если (C)=0, то (PC):=(PC)+<re18>

```
Пример:                         ;(C)=1
        JNC  LAB12              ;нет перехода на LAB12
        CPL  C
LAB12:  JNC  LAB13              ;переход на метку LAB13
```

Команда JNZ <re18>. Команда "переход, если содержимое аккумулятора не равно нулю" выполняет ветвление по адресу, если хотя бы один бит аккумулятора равен "1", в противном случае выполняется следующая команда. Адрес ветвления вычисляется сложением относительного смещения со знаком во втором байте команды (rel8) и содержимого счетчика команд (PC) после прибавления к нему 2. Содержимое аккумулятора не изменяется. Эта команда на флаги не влияет.

Ассемблер: JNZ <метка>

Код	0 1 1 1	0 0 0 0	rel8

Время: 2 цикла

Алгоритм: (PC):=(PC)+2, если ((A)>0 или (A)<0), то (PC):=(PC)+<re18>

```
Пример:                         ;(A)=00H
        JNC  LAB14              ;нет перехода на LAB14
        INC  A
LAB14:  JNZ  LAB15              ;переход на метку LAB15
. . .   . . .
LAB15:  NOP
```

Команда JZ <re18>. Команда "переход, если содержимое аккумулятора равно "0" выполняет ветвление по адресу, если все биты аккумулятора равны "0", в противном случае выполняется следующая команда. Адрес ветвления вычисляется сложением относительного смещения со знаком во втором байте команды (rel8) с содержимым счетчика команд после прибавления к нему 2. Содержимое аккумулятора не изменяется. Эта команда на флаги не влияет.

Ассемблер: JZ <метка>

Код	0 1 1 0	0 0 0 0	rel8

Время: 2 цикла

Алгоритм: (PC):=(PC)+2, если (A)=0, то (PC):=(PC)+<re18>

```
Пример:                          ;(A)=01H
        JZ    LAB16              ;нет перехода на LAB16
        DEC   A
LAB16:  JZ    LAB17              ;переход на метку LAB17
. . .   . . .
LAB17:  CLR   A
```

Команда LCALL <addrl6>. Команда "длинный вызов" вызывает подпрограмму, находящуюся по указанному адресу. По команде LCALL к счетчику команд (PC) прибавляется 3 для получения адреса следующей команды и после этого полученный 16-битовый результат помещается в стек (сначала следует младший байт, за ним - старший), а содержимое указателя стека (SP) увеличивается на 2. Затем старший и младший байты счетчика команд загружаются, соответственно, вторым и третьим байтами команды LCALL. Выполнение программы продолжается командой, находящейся по полученному адресу. Подпрограмма, следовательно, может начинаться в любом месте адресного пространства памяти программ объемом до 64 Кбайт. Эта команда на флаги не влияет.

Ассемблер: LCALL <метка>

Код	0 0 0 1	0 0 1 0	addr[15-8]	addr[7-0]

Время: 2 цикла

Алгоритм: (PC):=(PC)+3, (SP):=(SP)+1, ((SP)):=(PC[7-0]),
(SP):=(SP)+1, ((SP)):=(PC[15-8]), (PC):=<addr[15-0]>

```
Пример:                  ;(SP)=07H, метке PRN соответствует адрес 1234H,
                          ;по адресу 0126H находится команда LCALL
        LCALLPRN         ;(SP)=09H, (PC)=1234H, (ОЗУ[08])=26H,
                         ; (ОЗУ[09])=01H
```

Команда LJMP <addr16>. Команда "длинный переход" выполняет безусловный переход по указанному адресу, загружая старший и младший байты счетчика команд (PC) соответственно вторым и третьим байтами, находящимися в коде команды. Адрес перехода, таким образом, может находиться по любому адресу пространства памяти программ в 64 Кбайт. Эта команда на флаги не влияет.

Ассемблер: LJMP<метка>

Код	0 0 0 0	0 0 1 0	addr[15-8]	addr[7-0]

Время: 2 цикла

Алгоритм: (PC):=<addr[15-0]>

Команда MOV <байт-назначения>, <байт-источника>. Команда "переслать переменную-байт" пересылает переменную-байт, указанную во втором операнде, в ячейку, указанную в первом операнде. Содержимое байта

источника не изменяется. Эта команда на флаги и другие регистры не влияет. Команда "MOV" допускает 15 комбинаций адресации байта-источника и байта-назначения:

Ассемблер: 1) MOV A,Rn ;где n=0-7

Код | 1 1 1 0 | 1 r r r |

где rrr=000-111

Время: 1 цикл

Алгоритм: (A):=(Rn)

Ассемблер: 2) MOV A,<direct>

Код | 1 1 1 0 | 0 1 0 1 | direct address |

Время: 1 цикл

Алгоритм: (A):=(direct)

Ассемблер: 3) MOV A,@Ri ;где i=0,1

Код | 1 1 1 0 | 0 1 1 i |

Время: 1 цикл

Алгоритм: (A):=((Ri))

Ассемблер: 4) MOV A,#data

Код | 0 1 1 1 | 0 1 0 0 | #data8 |

Время: 1 цикл

Алгоритм: (A):=<#data8>

Ассемблер: 5) MOV Rn,A ;где n=0-7

Код | 1 1 1 1 | 1 r r r |

где rrr=000-111

Время: 1 цикл

Алгоритм: (Rn):=(A)

Ассемблер: 6) MOV Rn,<direct> ;где n=0-7

Код | 1 0 1 0 | 1 r r r | direct address |

Время: 2 цикла

Алгоритм: (Rn):=(direct)

Ассемблер: 7) MOV Rn,#data ;где n=0-7

Код | 0 1 1 1 | 1 r r r | #data8 |

Время: 1 цикл

Алгоритм: (Rn):=<#data8>

Ассемблер: 8) MOV <direct>,A

Код | 1 1 1 1 | 0 1 0 1 | direct address |

Время: 1 цикл

Алгоритм: (direct):=(A)

Ассемблер: 9) MOV <direct>,Rn ;где n=0-7

Код	1 0 0 0	1 r r r	direct address

Время: 2 цикла
Алгоритм: (direct):=(Rn)

Ассемблер: 10) MOV <direct>,<direct>

Код	1 0 0 0	0 1 0 1	direct address	direct address

Время: 2 цикла
Алгоритм: (direct):=(direct)

Ассемблер: 11) MOV <direct>,@Ri ;где i=0,l

Код	1 0 0 0	0 1 1 i	direct address

Время: 2 цикла
Алгоритм: (direct):=((Ri))

Ассемблер: 12) MOV <direct>,#data

Код	0 1 1 1	0 1 0 1	direct address	#data8

Время: 2 цикла
Алгоритм: (direct):=<#data8>

Ассемблер: 13) MOV @Ri,A ;где i=0,l

Код	1 1 1 1	0 1 1 i

Время: 1 цикл
Алгоритм: ((Ri)):=(A)

Ассемблер: 14) MOV @Ri,<direct> ;где i=0,l

Код	1 0 1 0	0 1 1 i	direct address

Время: 2 цикла
Алгоритм: ((Ri)):=(direct)

Ассемблер: 15) MOV @Ri,#data ;где i=0,l

Код	0 1 1 1	0 1 1 i

Время: 1 цикл
Алгоритм: ((Ri)):=<#data8>
Примеры:

```
1)                      ;(A)=FAH, (R4)=93H
      MOV A,R4          ;(A)=93H, (R4)=93H
2)                      ;(A)=93H, (ОЗУ[40])=10H, (R0)=40H
      MOV A,40H         ;(A)=10H, (ОЗУ[40])=10H, (R0)=40H
3)                      ;(A)=10H, (R0)=41H, (ОЗУ[41])=0CAH
      MOV A,@R0         ;(A)=CAH, (R0)=41H, (ОЗУ[41])=0CAH
4)                      ;(A)=C9H (11001001B)
      MOV A,#37H        ;(A)=37H (00110111B)
5)                      ;(A)=38H, (R0)=42H
```

```
        MOV R0,A        ;(A)=38H, (R0)=38H
6)                      ;(R0)=39H, (P2)=0F2H
        MOV R0,P2       ;(R0)=F2H
7)                      ;(R0)=0F5H
        MOV R0,#49H     ;(R0)=49H
8)                      ;(P0)=FFH, (A)=4BH
        MOV P0,A        ;(P0)=4BH, (A)=4BH
9)                      ;(PSW)=C2H, (R7)=57H
        MOV PSW,R7      ;(PSW)=57H, (R7)=57H
10)                     ;(ОЗУ[45])=33H, (03У[48])=0DEH
        MOV 48H,45H     ;(ОЗУ[45])=33H, (ОЗУ[48])=33H
11)                     ;(R1)=49H, (ОЗУ[49])=0E3H
        MOV 51H,@R1     ;(ОЗУ[51])=0E3H, (ОЗУ[49])=0E3H
12)                     ;(ОЗУ[5F])=9BH
        MOV 5FH,#07H    ;(ОЗУ[5F])=07H
13)                     ;(R1)=48H, (ОЗУ[48])=75H, (A)=0BDH
        MOV @R1,A       ;(ОЗУ[48])=0BDH
14)                     ;(R0)=51H, (ОЗУ[51])=0E3H, (P0)=0ACH
        MOV @R0,P0      ;(ОЗУ[51])=0ACH
15)                     ;(ОЗУ[7E])=67H, (R1)=7EH
        MOV @R1,#0A9H;(ОЗУ[7E])=0A9H, (P1)=7EH
```

Команда MOV <бит назначения>, <бит источника>. Команда "переслать бит данных" битовую переменную, указанную во втором байте, копирует в разряд, который указан в первом операнде. Одним из операндов должен быть флаг переноса C, а другим может быть любой бит, к которому возможна прямая адресация.

Ассемблер: 1) MOV C,<bit>

Код	1 0 1 0	0 0 1 0	bit address

Время: 2 цикла
Алгоритм: (C):=(bit)
Ассемблер: MOV <bit>,C

Код	1 0 0 1	0 0 1 0	bit address

Время: 2 цикла
Алгоритм: (bit):=(C)
Примеры:

```
1)                      ;(C)=0, (P3)=D5H (11010101B)
        MOV C,P3.0      ;C:=1
        MOV C,P3.3      ;C:=0
        MOV C,P3.7      ;C:=1
2)                      ;(C)=1, (P0)=20H (00100000B)
        MOV P0.1,C
        MOV P0.2,C
        MOV P0.3,C      ;(C)=1, (P0)=2EH (00101110B)
```

Команда MOV DPTR,#datal6. Команда "загрузить указатель данных 16-битовой константой" загружает указатель данных DPTR 16-битовой константой, указанной во втором и третьем байтах команды. Второй байт команды загружается в старший байт указателя данных (DPH), а третий байт - в младший байт указателя данных (DPL). Эта команда на флаги не влияет и является единственной командой, которая одновременно загружает 16 бит данных.

Ассемблер: MOV DPTR,#<data16>

Код	1 0 0 1	0 0 0 0	#data[15-8]	#data[7-0]

Время: 2 цикла
Алгоритм: (DPTR):=#data[15-0], причем DPH:=#data[15-8], DPL:=#data[7-0]
Пример: ;(DPTR)=01FDH
MOV DPTR,#1234H ;(DPTR)=1234H, (DPH)=12H, (DPL)=34H

Команда MOVC A,@A+(<R16>). <R16> - 16-разрядный регистр. Команда "переслать байт из памяти программ" загружает аккумулятор байтом кода или константой из памяти программы. Адрес считываемого байта вычисляется как сумма 8-битового исходного содержимого аккумулятора без знака и содержимого 16-битового регистра. В качестве 16-битового регистра может быть:
1) указатель данных DPTR;
2) счетчик команд PC.

В случае, когда используется PC, он увеличивается до адреса следующей команды перед тем, как его содержимое складывается с содержимым аккумулятора. 16-битовое сложение выполняется так, что перенос из младших восьми бит может распространяться через старшие биты. Эта команда на флаги не влияет.

Ассемблер: 1) MOVC A,@A+DPTR

Код	1 0 0 1	0 0 1 1

Время: 2 цикла
Алгоритм: (A):=((A)+(DPTR))

Ассемблер: 2) MOVC A,@A+PC

Код	1 0 0 0	0 0 1 1

Время: 2 цикла
Алгоритм: (A):=((A)+(PC))
Примеры:

```
1)                                  ;(A)=1BH, (DPTR)=1020H,
                                    ; (ПЗУ[103B])=48H,
      MOVC    A,@A+DPTR             ;(A)=48H, (DPTR)=1020H
2)                                  ;(A)=FAH, (PC)=0289,
                                    ; (ПЗУ[0384])=9BH
      MOVC    A,@A+PC               ;(A)=9BH, (PC)=028AH
```

Команда MOVX <байт приемника>,<байт источника>. Команда "переслать во внешнюю память (из внешней памяти) данных" пересылает данные между аккумулятором и байтом внешней памяти данных. Имеется два типа команд, которые отличаются тем, что обеспечивают 8-битовый или 16-битовый косвенный адрес к внешнему ОЗУ данных.

В первом случае содержимое R0 или R1 в текущем банке регистров обеспечивает 8-битовый адрес, который мультиплексируется с данными порта P0. Для расширения дешифрации ввода-вывода или адресации небольшого массива ОЗУ достаточно восьми бит адресации. Если применяются ОЗУ, немного больше чем 256 байт, то для фиксации старших битов адреса можно использовать любые другие выходы портов, которые переключаются командой, стоящей перед командой MOVX.

Во втором случае при выполнении команды MOVX указатель данных DPTR генерирует 16-битовый адрес. Порт P2 выводит старшие восемь бит адреса (DPH), а порт P0 мультиплексирует младшие 8 бит адреса (DPL) с данными. Эта форма является эффективной при доступе к большим массивам данных (до 64К байт), так как для установки портов вывода не требуется дополнительных команд.

Ассемблер: 1) MOVX A,@Ri ;где i=0,1

Код	1 1 1 0	0 0 1 i

Время: 2 цикла
Алгоритм: (A):=((Ri))

Ассемблер: 2) MOVX A,@DPTR

Код	1 1 1 0	0 0 0 0

Время: 2 цикла
Алгоритм: (A):=((DPTR))

Ассемблер: 3) MOVX @Ri,A ;где i=0,1

Код	1 1 1 1	0 0 1 i

Время: 2 цикла
Алгоритм: ((Ri)):=(A)

Ассемблер: 4) MOVX @DPTR,A

Код	1 1 1 1	0 0 0 0

Время: 2 цикла
Алгоритм: ((DPTR)):=(A)
Примеры:

```
1)                          ;(A)=32H, (R0)=83H, ячейка внешнего ОЗУ
                            ;по адресу 83H
                            ;содержит B6H
     MOVX   A,@R0           ;(A)=B6H, (R0)=83H
2)                          ;(A)=5CH, (DPTR)=1ABEH, ячейка
                            ;внешнего ОЗУ по адресу
```

```
                                ;1ABEH содержит 72H
        MOVX    A,@DPTR         ;(A)=72H, (DPTR)=2ABEH
3)                              ;(A)=95H, (R1)=FDH, ячейка внешнего ОЗУ
                                ;с адресом FDH
                                ;содержит 00
        MOVX    @R1,A           ;(A)=95H, (R1)=FDH, ячейка внешнего ОЗУ
                                ;с адресом FDH
                                ;содержит 95H
4)                              ;(A)=97H, (DPTR)=1FFFH, ячейка внешнего
                                ;ОЗУ с адресом 1FFFH
                                ;содержит 00
        MOVX    @DPTR,A         ;(A)=97H, ячейка внешнего ОЗУ с адресом
                                ;1FFFH содержит 97H
```

Команда MUL AB. Команда "умножение" умножает 8-битовые целые числа без знака из аккумулятора и регистра B. Старший байт 16-битового произведения помещается в регистр B, а младший - в аккумулятор A. Если результат произведения больше, чем 0FFH (255), то устанавливается флаг переполнения (OV), в противном случае он сбрасывается. Флаг переноса всегда сбрасывается.

Ассемблер: MUL AB

Код	1 0 1 0	0 1 0 0

Время: 4 цикла

Алгоритм: (A)[7-0]=(A)*(B), (B)[15-8]=(A)*(B)

Примеры:

```
1)                      ;(A)=50H (50H=80 DEC), (C=1), (B)=0A0H
                        ; (A0H=160 DEC), (OV)=0
        MUL AB          ;(A)=00H, (B)=32H, (C)=0, (OV)=1
2)                      ;(A)=2HH, (OV)=1, (B)=06H, (C)=l
        MUL AB          ;(A)=0D8H, (B)=00H, (OV)=0, (C)=0
```

Команда NOP. Команда "нет операции" выполняет холостой ход и не влияет на регистры и флаги, кроме как на счетчик команд (PC).

Ассемблер: NOP

Код	0 0 0 0	0 0 0 0

Время: 1 цикл

Алгоритм: (PC):=(PC)+1

Пример: Пусть требуется создать отрицательный выходной импульс на порте P1[6] длительностью 3 цикла. Это выполнит следующая последовательность команд:

```
        CLR  P1.6       ;P1[6]:=0
        NOP
        NOP
```

```
        NOP
        SETB    P1.6        ;P1[6]:=1
```

Команда ORL <байт назначения>,<байт источника>. Команда "логическое "ИЛИ" для переменных-байтов" выполняет операцию логического "ИЛИ" над битами указанных переменных, записывая результат в байт назначения. Эта команда на флаги не влияет. Допускается шесть комбинаций режимов адресации:

если байтом назначения является аккумулятор:

1) регистровый;
2) прямой;
3) косвенно-регистровый;
4) непосредственный;

если байтом назначения является прямой адрес:

5) к аккумулятору
6) к константе

Ассемблер: 1) ORL A,Rn ;где n=0-7

Код	0 1 0 0	1 r r r

Время: 1 цикл

Алгоритм: (A):=(A) OR (Rn), где OR - операция логического "ИЛИ"

Ассемблер: 2) ORL A,<direct>

Код	0 1 0 0	0 1 0 1	direct address

Время: 1 цикл

Алгоритм: (A):=(A) OR (direct)

Ассемблер: 3) ORL A,@Ri ;где i=0,1

Код	0 1 0 0	0 1 1 i

Время: 1 цикл

Алгоритм: (A):=(A) OR ((Ri))

Ассемблер: 4) ORL A,#<data>

Код	0 1 0 0	0 1 0 0	#data8

Время: 1 цикл

Алгоритм: (A):=(A) OR #<data>

Ассемблер: 5) ORL (direct),A

Код	0 1 0 0	0 0 1 0	direct address

Время: 1 цикл

Алгоритм: (direct):=(direct) OR (A)

Ассемблер: 6) ORL (direct),#<data>

Код	0 1 0 0	0 0 1 1	direct address	#data8

Время: 2 цикла

Алгоритм: (direct):=(direct) OR#<data>

Примеры:

```
1)                      ;(A)=15H, (R5)=6CH
      ORL  A,R5         ;(A)=7DH, (R5)=6CH
2)                      ;(A)=84H, (PSW)=C2H
      ORL  A,PSW        ;(A)=C6H, (PSW)=C2H
3)                      ;(A)=52H, (R0)=6DH, (ОЗУ[6D])=49H
      ORL  A,@R0        ;(A)=5BH, (ОЗУ[6D])=49H
4)                      ;(A)=F0H
      ORL  A,#0AH       ;(A)=FAH
5)                      ;(A)=34H, (IP)=23H
      ORL  IP,A         ;(IP)=37H, (A)=34H
6)                      ;(P1)=00H
      ORL  P1,#0C4H     ;(P1)=11000100B (C4H)
```

Примечание: Если команда используется для работы с портом, величина, используемая в качестве исходных данных порта, считывается из "защелки" порта, а не с выводов БИС.

Команда ORL C, <бит источника>. Команда "логическое "ИЛИ" для переменных-битов" устанавливает флаг переноса C, если булева величина равна логической "1", в противном случае устанавливает флаг C в "0". Косая дробь ("/") перед операндом на языке ассемблера указывает на то, что в качестве операнда используется логическое отрицание значения адресуемого бита, но сам бит источника не изменяется. Эта команда на другие флаги не влияет.

Ассемблер: 1) ORL C,<bit>

Код	0 1 1 1	0 0 1 0	bit address

Время: 2 цикла
Алгоритм: (C):=(C) OR (bit)

Ассемблер: 2) ORL C,/<bit>

Код	1 0 1 0	0 0 0 0	bit address

Время: 2 цикла
Алгоритм: (C):=(C) OR /(bit)
Примеры:

```
1)                      ;(C)=0, (P1)=53H (01010011B)
      ORL  C,P1.4       ;(C)==1, (P1)=53H (01010011B)
2)                      ;(C)=0, (ОЗУ[25])=39H (00111001B)
      ORL  C,/2A        ;(C)=1, (ОЗУ[25])=39H (00111001B)
```

Команда POP <direct>. Команда "чтение из стека" считывает содержимое ячейки, которая адресуется с помощью указателя стека, в прямо адресуемую ячейку ОЗУ, при этом указатель стека уменьшается на единицу. Эта команда не воздействует на флаги и часто используется для чтения из стека промежуточных данных.

Ассемблер: POP <direct>

Код	1 1 0 1	0 0 0 0	direct address

Время: 2 цикла

Алгоритм: (direct):=((SP)), (SP):=(SP)-1

```
Пример:                 ;(SP)=32H, (DPH)=01, (DPL)=ABH,
                        ; (ОЗУ[32])=12H,(ОЗУ[31])=56H,
                        ;(ОЗУ[30])=20H
        POP  DPH
        POP  DPL        ;(SP)=30H, (DPH)=12H, (DPL)=56H,
                        ; (ОЗУ[32])=12H, (ОЗУ[31])=56H
        POP  SP         ;(SP)=20H, (ОЗУ[30])=20H
```

Команда PUSH <direct>. Команда "запись в стек" увеличивает указатель стека на единицу и после этого содержимое указанной прямо адресуемой переменной копируется в ячейку внутреннего ОЗУ, адресуемого с помощью указателя стека. На флаги эта команда не влияет и используется для записи промежуточных данных в стек.

Ассемблер: PUSH <direct>

Код	1 1 0 0	0 0 0 0	direct address

Время: 2 цикла

Алгоритм: (SP):=(SP)+1, ((SP)):=(direct)

```
Пример:                 ;(SP)=09H, (DPTR)=1279H
        PUSH    DPL
        PUSH    DPH     ;(SP)=0BH, (DPTR)=1279H, (ОЗУ[0A])=79H,
                        ; (ОЗУ[0B])=12H
```

Команда RET. Команда "возврат из подпрограммы" последовательно выгружает старший и младший байты счетчика команд из стека, уменьшая указатель стека на 2. Выполнение основной программы обычно продолжается по адресу команды, следующей за ACALL или LCALL. На флаги эта команда не влияет.

Ассемблер: RET

Код	0 0 1 0	0 0 1 0

Время: 2 цикла

Алгоритм: (PC)[15-8]:=((SP)), (SP):=(SP)-1, (PC)[7-0]:=((SP)), (SP):=(SP)-1

```
Пример:                 ;(SP)=0DH, (ОЗУ[0C])=93H, (ОЗУ[0D])=02H
        RET             ;(SP)=0BH, (PC)=0293H
```

Команда RETI. Команда "возврат из прерывания" выгружает старший и младший байты счетчика команд из стека и устанавливает "логику прерываний", разрешая прием других прерываний с уровнем приоритета, равным уровню приоритета только что обработанного прерывания. Указатель стека уменьшается на 2. Слово состояния программы (PSW) не

восстанавливается автоматически. Выполнение основной программы продолжается с команды, следующей за командой, на которой произошел переход к обнаружению запроса на прерывание. Если при выполнении команды RETI обнаружено прерывание с таким же или меньшим уровнем приоритета, то одна команда основной программы успевает выполниться до обработки такого прерывания.

Ассемблер: RETI

Код	0 0 1 1	0 0 1 0

Время: 2 цикла

Алгоритм: (PC)[15-8]:=((SP)), (SP):=(SP)-1, (PC)[7-0]:=((SP)), (SP):=(SP)-1

```
Пример:                 ;(SP)=0BH, (ОЗУ[0A]).=2AH, (ОЗУ[0B])=03H,
                        ;(PC)=YYYYH, где Y=0-FH
        RETI            ;(SP)=09H, (PC)=032AH
```

Команда RL A. Команда "сдвиг содержимого аккумулятора влево", сдвигает восемь бит аккумулятора на один бит влево, бит 7 засылается на место бита 0. На флаги эта команда не влияет.

Ассемблер: RL A

Код	0 0 1 0	0 0 1 1

Время: 1 цикл

Алгоритм: (A[N+1]):=(A[N]), где N=0-6
(A[0]):=(A[7])

```
Пример:                 ;(A)=0D5H (11010101B), (C)=0
        RL  A           ;(A)=0ABH (10101011B), (C)=0
```

Команда RLC A. Команда "сдвиг содержимого аккумулятора влево через флаг переноса" сдвигает восемь бит аккумулятора и флаг переноса влево на один бит. Содержимое флага переноса помещается на место бита 0 аккумулятора, а содержимое бита 7 аккумулятора переписывается в флаг переноса. На другие флаги эта команда не влияет.

Ассемблер: RLC A

Код	0 0 1 1	0 0 1 1

Время: 1 цикл

Алгоритм: (A[N+1]):=(A[N]), где N=0-6
(A[0]):=(C), (C):=(A[7])

```
Пример:                 ;(A)=56H (01010110B), (C)=1
        RLC A           ;(A)=0ADH (10101101B), (C)=0
```

Команда RR A. Команда "сдвиг содержимого аккумулятора вправо" сдвигает вправо на один бит все восемь бит аккумулятора. Содержимое бита 0 помещается на место бита 7. На флаги эта команда не влияет.

Ассемблер: RR A

Код	0 0 0 0	0 0 1 1

Время: 1 цикл
Алгоритм: (A[N]):=(A[N+1]), где N=0-6
(A[7]):=(A[0])
Пример: ;(A)=0D6H (11010110B), (C)=1
RR A ;(A)=6BH (01101011B), (C)=l

Команда RRC A. Команда "сдвиг содержимого аккумулятора вправо через флаг переноса" сдвигает восемь бит аккумулятора и флаг переноса на один бит вправо. Бит 0 перемещается в флаг переноса, а исходное содержимое флага переноса помещается в бит 7. На другие флаги эта команда не влияет.

Ассемблер: RRC A

Код	0 0 0 1	0 0 1 0

Время: 1 цикл
Алгоритм: (A[N]):=(A[N+1]), где N=0-6
(A[7]):=(C), (C):=(A[0])
Пример: ;(A)=95H (10010101B), (C)=0
RRC A ;(A)=4AH (01001010B), (C)=1

Команда SETB <бит>. Команда "установить бит" устанавливает указанный бит в "1". Адресуется:
1) к флагу переноса (C);
2) к биту с прямой адресацией.

Ассемблер: 1) SETB C

Код	1 1 0 1	0 0 1 1

Время: 1 цикл
Алгоритм: (C):=1

Ассемблер: 2) SETB <bit>

Код	0 0 0 1	0 1 0 0	bit address

Время: 1 цикл
Алгоритм: (bit):=1

Примеры:

```
1)                      ;(C)=0
    SETB    C           ;(C)=1
2)                      ;(P2)=38H (00111000B)
    SETB    P2.0
    SETB    P2.7        ;(P2)=B9H (10111001B)
```

Команда SJMP <метка>. Команда "короткий переход" выполняет безусловное ветвление в программе по указанному адресу. Адрес ветвления вычисляется сложением смещения со знаком во втором байте команды с содержимым счетчика команд после прибавления к нему 2. Таким образом,

адрес перехода должен находиться в диапазоне от 128 байт, предшествующих команде, до 127 байт, следующих за ней.

Ассемблер: SJMP <метка>

Код	1 0 0 0	0 0 0 0	rel8

Время: 2 цикла

Алгоритм: (PC):=(PC)+2, (PC):=(PC)+(re18)

```
Пример:                 ;(PC)=0418H, метка MET1 соответствует адресу
                        ;039AH
        SJMPMET1        ;(PC)-039AH, где (rel8)=80H= -128 DEC
        SJMPMET2        ;(PC)=041AH, где метка MET2 соответствует
                        ;адресу 041AH,
                         ;(rel8)=7DH= +125 DEC
```

Команда SUBB A,<байт источника>. Команда "вычитание с заемом" вычитает указанную переменную вместе с флагом переноса из содержимого аккумулятора, засылая результат в аккумулятор. Эта команда устанавливает флаг переноса (заема), если при вычитании для бита 7 необходим заем, в противном случае флаг переноса сбрасывается. Если флаг переноса установлен перед выполнением этой команды, то это указывает на то, что заем необходим при вычитании с увеличенной точностью на предыдущем шаге, поэтому флаг переноса вычитается из содержимого аккумулятора вместе с операндом источника. (AC) устанавливается, если заем необходим для бита 3 и сбрасывается в противном случае. Флаг переполнения (0V) устанавливается, если заем необходим для бита 6, но его нет для бита 7, или есть для бита 7, но нет для бита 6.

При вычитании целых чисел со знаком (0V) указывает на отрицательное число, которое получается при вычитании отрицательной величины из положительной, или положительной число, которое получается при вычитании положительного числа из отрицательного. Операнд источника допускает четыре режима адресации:

1) регистровый;
2) прямой;
3) косвенно-регистровый;
4) непосредственный (к константе).

Ассемблер: 1) SUBB A.Rn ;где n=0-7

Код	1 0 0 1	1 r r r

Время: 1 цикл

Алгоритм: (A):=(A)-(C)-(Rn); (C):=X, (AC):=X, (OV):=X, где X=(0 или 1)

Ассемблер: 2) SUBB A,<direct>

Код	1 0 0 1	0 1 0 1	direct address

Время: 1 цикл

Алгоритм: (A):=(A)-(C)-(direct); (C):=X, (AC):=X, (OV):=X, где X=(0 или 1)

Ассемблер: 3) SUBB A,@Ri ;где i=0,l

Код | 1 0 0 1 | 0 1 1 i |

Время: 1 цикл

Алгоритм: (A):=(A)-(C)-((Ri)); (C):=X, (AC):=X, (OV):=X, где X=(0 или 1)

Ассемблер: 4) SUBB A,#data

Код | 1 0 0 1 | 0 1 0 0 | #data8 |

Время: 1 цикл

Алгоритм: (A):=(A)-(C)-(#data8); (C):=X, (AC):=X, (OV):=X, где X=(0 или 1)

Примеры:

```
1)                              ;(A)=C9H, (R2)=54H, (C)=l
      SUBB      A,R2            ;(A)=74H, (R2)=54H, (C)==0, (AC)=0,
                                (OV)=1
2)                              ;(A)=97H, (B)=25H, (C)=0
      SUBB      A,B             ;(A)=72H, (B)=25H, (C)=0, (AC)=0, (OV)=1
3)                              ;(A)=49H, (C)=l, (R0)=33H, (ОЗУ[33])=68H
      SUBB A,@R0                ;(A)=E0H, (C)=l, (AC)=0, (OV)=0
4)                              ;(A)=0BEH, (C)=0
      SUBB      A,#3FH          ;(A)=7FH, (C)=0, (AC)=1, (OV)=1
```

Команда SWAP A. Команда "обмен тетрадами внутри аккумулятора" осуществляет обмен между младшими четырьмя и старшими четырьмя битами аккумулятора (между старшей и младшей тетрадами).Эта команда может рассматриваться так же, как команда четырехбитового циклического сдвига. На флаги эта команда не влияет.

Ассемблер: SWAP A

Код | 1 1 0 0 | 0 1 0 0 |

Время: 1 цикл

Алгоритм: (A[3-0]):=(A[7-4]), (A[7-4]):=(A[3-0])

Пример:

```
                                ;(A)=0D7H (11010111B)
      SWAP      A               ;(A)=7DH (01111101B)
```

Команда XCH A,<байт>. Команда "обмен содержимого аккумулятора с переменной-байтом" осуществляет обмен содержимого аккумулятора с содержимым источника, указанным в команде. Операнд источника может использовать следующие режимы адресации:

1) регистровый;
2) прямой;
3) косвенно-регистровый.

Ассемблер: 1) XCH A,Rn ;где n=0-7

Код | 1 1 0 0 | 1 r r r |

Время: 1 цикл

Алгоритм: (A):=(Rn), (Rn):=(A)

Ассемблер: 2) XCH A,<direct>

Код	1 1 0 0	0 1 0 1	direct address

Время: 1 цикл

Алгоритм: (A):=(direct), (direct):=(A)

Ассемблер: 3) XCH A,@Ri, где i=0,1

Код	1 1 0 0	0 1 1 i

Время: 1 цикл

Алгоритм: (A):=((Ri)), ((Ri)):=(A)

```
1)                          ;(A)=3CH, (R4)=15H
        XCH A,R4            ;(A)=15H, (R4)=3CH
2)                          ;(A)=0FEH, (P3)=0DAH
        XCH A,P3            ;(A)=0DAH, (P3)=0FEH
3)                          ;(R1)=39H, (ОЗУ[39])=44H, (A)=0BCH
        XCH A,@R1           ;(ОЗУ[39])=0BCH, (A)=44H
```

Команда XCHD A,@Ri. Команда "обмен тетрадой" выполняет обмен младшей тетрады (биты 3 - 0) аккумулятора с содержимым младшей тетрады (биты 3 - 0) ячейки внутреннего ОЗУ, косвенная адресация к которой производится с помощью указанного регистра. На старшие биты (биты 7 - 4) и на флаги эта команда не влияет.

Ассемблер: XCHD A,@Ri ;где i=0,l

Код	1 1 0 1	0 1 1 i

Время: 1 цикл

Алгоритм: (A[3-0]):=((Ri[3-0])), ((Ri[3-0]))=(A[3-0])

```
Пример:                     ;(R0)=55H, (A)=89H, (0ЗУ[55])=0A2H
        XCHD    A,@R0       ;(A)=82H, (ОЗУ[55])=0A9H
```

Команда XRL <байт назначения>, <байт источника>. Команда "логическое "ИСКЛЮЧАЮЩЕЕ ИЛИ" для переменных-байтов" выполняет операцию "ИСКЛЮЧАЮЩЕЕ ИЛИ" над битами указанных переменных, записывая результат в байт назначения. На флаги эта команда не влияет. Допускается шесть режимов адресации:

байтом назначения является аккумулятор:

1) регистровый;
2) прямой;
3) косвенно-регистровый;
4) непосредственный;

байтом назначения является прямой адрес:

5) к аккумулятору;
6) к константе.

Ассемблер: 1) XRL A,Rn ;где n=0-7

Код	0 1 1 0	1 r r r

Время: 1 цикл
Алгоритм: (A):=(A) XOR (Rn)

Ассемблер: 2) XRL A,<direct>

Код	0 1 1 0	0 1 0 1	direct address

Время: 1 цикл
Алгоритм: (A):=(A) XOR (direct)

Ассемблер: 3) XRL A,@Ri ;где i=0,1

Код	0 1 1 0	0 1 1 i

Время: 1 цикл
Алгоритм: (A):=(A) XOR ((Ri))

Ассемблер: 4) XRL A,#data

Код	0 1 1 0	0 1 0 0	#data8

Время: 1 цикл
Алгоритм: (A):=(A) XOR <data>

Ассемблер: 5) XRL <direct>,A

Код	0 1 1 0	0 0 1 0	direct address

Время: 1 цикл
Алгоритм: (direct):=(direct) XOR (A)

Ассемблер: 6) XRL <direct>,#data

Код	0 1 1 0	0 0 1 1	direct address	#data8

Время: 2 цикла
Алгоритм: (direct):=(direct) XOR #data
Примеры:

```
1)                      ;(A)=C3H, (R6)=AAH
      XRL  A,R6         ;(A)=69H, (R6)=AAH
2)                      ;(A)=0FH, (P1)=A6H
      XRL  A,P1         ;(A)=A9H, (P1)=A6H
3)                      ;(A)=55H, R1=77H, (03У[77])=5AH
      XRL  A,@R1        ;(A)=0FH, (ОЗУ[77])=5AH
4)                      ;(A)=0C3H
      XRL  A,#0F5H      ;(A)=36H
5)                      :(A)=31H, (P1)=82H
      XRL  P1,A         ;(A)=31H, (P1)=B3H
6)                      ;(IP)=65H
      XRL  IP,#65H      ;(IP)=00H
```

Примечание: Если эта команда используется для работы с портами, то значение, используемое в качестве операнда, считывается из "защелки" порта, а не с выводов БИС.

2. КОМПИЛЯТОР ДЛЯ МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА МК51

2.1 СИСТЕМНЫЕ СОГЛАШЕНИЯ

Следующие расширения имен файлов будут использованы по умолчанию программами пакета фирмы 2550 A.D:

asm входной файл для ассемблера (компилятора);
obj выходной файл из ассемблера;
pak упакованный выходной файл;
lst файл листинга.

Отметим, что в выходной файл Ассемблера включается дополнительная информация, и Редактор связей (линкер) должен быть исполнен, даже если программа размещена с требуемого адреса и не содержит внешних ссылок. При этом удаляется дополнительная информация и генерируется файл в нужном формате.

2.2 СИНТАКСИС ЯЗЫКА АССЕМБЛЕРА

Определители основания системы счисления:

B Binary (двоичная);
O | Q Octal (восьмеричная);
D Decimal (десятичная);
H Hex (шестнадцатеричная);
"X" | 'X' Ascii (ASCII-коды).

Предопределены значения двух последовательных символов, заключенных в одиночные или двойные символы. Однако, чтобы пользоваться ими необходима директива TWOCHAR ON ('CR' - возврат каретки, 'LF' - перевод строки, 'SP' - пробел, 'HT' - горизонтальная табуляция, 'NL' - пустой символ).

Комментарии. Строки комментариев должны начинаться с точки с запятой или звездочки в первой колонке, за исключением использования директивы COMMENT. Комментарии после инструкций не сопровождаются точкой с запятой, если ассемблирование выполняется в Spaces Off моде. Если ассемблирование выполняется в Spaces On моде, все комментарии должны начинаться с точки с запятой. Подробнее смотрите директиву SPACES.

Программный счетчик. Специальные символы доллар ($) и звездочка (*) следует использовать в выражениях, чтобы определить программный счетчик. Величина, присвоенная знаку доллар, соответствует значению счетчика команд в начале этой инструкции.

Метки. Нелокальные метки могут состоять из любого числа символов, но только 32 символа будут значащими. Метки ставятся в любой колонке, если имя оканчивается двоеточием. Если двоеточие не используется, метка должна

начинаться с первой колонки. Большие и маленькие буквы считаются различными.

Локальные метки. Локальные метки могут использоваться подобно нелокальным меткам. Различие в том, что локальная метка определена только между нелокальными меткам. Когда программа переходит от одной локальной зоны к другой, имя локальной метки используется повторно. Эта особенность используется для меток, вызываемых только в локальной области, как описано выше, и новые имена меток не требуются. Ассемблер определяет локальные метки по символу доллар в начале или конце имени. Этот идентификатор может быть изменен с помощью директивы LLCHAR.

Старший байт. Для загрузки старшего байта 16-битной величины используется арифметический символ "больше чем" ">". Это позволит битам с 8 по 15 использоваться в качестве байта, величина которого смещена.

Младший байт. Для загрузки младшего байта 16-битной величины используется арифметический символ "меньше чем" "<". Это позволит битам с 0 по 7 использоваться в качестве байта, величина которого смещена.

Строчные и прописные символы. Метки, записанные строчными и прописными буквами, рассматриваются как различные метки. Метки, используемые для имен секций или макросов, различаются аналогичным образом.

2.3 ДИРЕКТИВЫ АССЕМБЛЕРА

Эта часть описывает директивы Ассемблера, которые позволяют управлять процессом компиляции. Полный перечень директив Ассемблера занимает 65 страниц, поэтому в данном разделе приводятся наиболее употребляемые и необходимые директивы.

ORG VALUE

Устанавливает адрес программы. Если директива не выполнена адрес по умолчанию устанавливается в 0000.

END VALUE

Эта директива отмечает конец программы или подключенного файла. Выражение, следующее за END - дополнительное, и если существует, то указывает стартовый адрес программы. Этот адрес заносится в выходной файл, если в выходном формате существует тип записи стартового адреса.

LABEL: DB VALUE

Ассемблер помещает величины в последовательные ячейки памяти. Если после директивы не следует выражение, один байт резервируется и обнуляется. Метка является необязательной.

LABEL: DW VALUE

Эта директива помещает 16-битные величины в память. Несколько слов могут быть заданы, записав несколько выражений через запятую. Если выражение не дано, резервируется и обнуляется одно слово. Метка не является обязательной.

LABEL: LONG VALUE

Эта директива помещает 32-битные величины в память. Несколько слов могут быть заданы, записав несколько выражений через запятую. Если выражение не дано, резервируется и обнуляется одно слово. Метка не является обязательной.

LABEL: DS SIZE,VALUE

Эта директива резервирует фиксированное число байт, определяемое величиной SIZE. Никакие величины не запоминаются в резервируемой области.

LABEL: EQU VALUE

Присваивает имени LABEL значение VALUE. VALUE может быть другим символом или разрешенным математическим выражением.

LABEL: VAR VALUE

Присваивает имени LABEL значение VALUE, но значение может изменяться по тексту программы. Имя, определенное как VAR, не следует переопределять директивой EQU.

RADIX <значение>

Основание	Значение
2 или B	Двоичное
8 или O или Q	Восьмеричное
10 или D	Десятичное
16 или H	Шестнадцатеричное

Отсутствие выражения означает возврат к стандартному (используемому по умолчанию) режиму (т.е. с основанием системы счисления 10); при этом подразумевается, что все другие системы счисления будут обозначаться с помощью литер B, Q, D или H после константы. Следует отметить, что при задании основания системы счисления 16 не существует способа описания десятичного или двоичного числа, поскольку как литера D, так и литера B являются допустимыми шестнадцатеричными цифрами.

INCLUDE <имя-файла>

Указывает ассемблеру включить данный файл в процесс ассемблирования. Имена файлов могут включать в себя маршрутные имена. Расширения имен должны задаваться полностью. Вложенное включение файлов не допускается.

2.4 ВЫЧИСЛЕНИЯ ВО ВРЕМЯ ТРАНСЛИРОВАНИЯ

Представленный далее список содержит перечень вычислительных операций, допустимых в период ассемблирования. Указаны также их уровни приоритетов. Операции, которые имеют уровень приоритетов 7, должны быть выполнены в первую очередь. Для изменения порядка выполнения вычислительных операций могут использоваться круглые скобки. Вычисления осуществляются при помощи 80-и разрядных целых чисел за исключением операций возведения в степень, в которых используется 8-и разрядное значение показателя степени. Максимальное число незавершенных операций равно 16.

Операция	Приоритет	Описание
Унарный +	7	Может задавать положительный операнд
Унарный -	7	Меняет знак последующего выражения
\ или .NOT.	7	Инвертирует (дополняет) последующее выражение
Унарный >	7	Хранит старший байт последующего адреса. Должна использоваться для получения значений перемещаемых адресов байтов
Унарный <	7	Хранит младший байт последующего адреса. Должна использоваться для получения значений перемещаемых адресов байтов
**	6	Беззнаковое возведение в степень
*	5	Беззнаковое перемножение
/	5	Беззнаковое деление
.MOD.	5	Вычисление остатка
.SHR.	5	Сдвиг предыдущего выражения вправо (с заполнением нулями) на число позиций, заданных последующим выражением
.SHL.	5	Сдвиг предыдущего выражения влево (с заполнением нулями) на число позиций, заданных последующим выражением
+	4	Сложение
-	4	Вычитание
&или.AND.	3	Логическое И
^ или.OR.	2	Логическое ИЛИ
.XOR.	2	Логическое исключающее ИЛИ

2.5 СРАВНЕНИЯ ВО ВРЕМЯ ТРАНСЛИРОВАНИЯ

В представленном далее списке приводятся операции сравнения периода ассемблирования, возвращаемым значением которых являются все единицы (если высказывание является истинным) или все нули (если высказывание является ложным).

Операция	Описание
= или .EQ.	Равно
> или .GT.	Больше чем
< или .LT.	Меньше чем
.UGT.	Беззнаковое больше чем
.ULT.	Беззнаковое меньше чем

2.6 СООБЩЕНИЯ ОБ ОШИБКАХ АССЕМБЛИРОВАНИЯ

SYNTAX ERROR. (Синтаксическая ошибка). Обычно эта ошибка возникает из-за пропущенной запятой или круглой скобки.

CAN'T RESOLVE OPERAND. (Невозможна идентификация обращения к операнду). Непонятно, что в данном случае предполагалось программистом.

ILLEGAL ADDRESSING MODE. (Не верный режим адресации). Адресация операнда с использованием данной формы адресации недопустима.

ILLEGAL ARGUMENT. (Не верный аргумент). В данном контексте операнд не может быть использован.

MULTIPLY DEFINED SYMBOL. (Символ уже описан). Данный символ уже описан ранее (не включая '.VAR').

ILLEGAL MNEMONIC. (Неверное мнемоническое обозначение). Данное мнемоническое обозначение не существует, и не было использовано для задания макроса.

 # TOO LARGE. (Число слишком велико). Результат слишком велик для данного операнда.

ILLEGAL ASCII DESIGNATOR. (Неверный код таблицы ASCII). Неверный символ пунктуации или код таблицы Ascii.

HEX # AND SYMBOL ARE IDENTICAL. (Шестнадцатеричное число и символ являются идентичными). Существует некоторая метка, которая в точности идентична шестнадцатеричному числу, используемому в качестве операнда. Для того, чтобы возникла эта ошибка, даже индикатор шестнадцатеричного числа должен быть на той же самой позиции.

UNDEFINED SYMBOL. (Символ не определен). Символ не был описан в период выполнения первого прохода ассемблера.

RELATIVE JUMP TOO LARGE. (Слишком далекий относительный переход). Результирующий адрес перехода находится на другой странице.

EXTRA CHARACTERS AT END OF OPERAND. (В конце операнда присутствуют лишние символы). Обычно возникновение этого сообщения связано с наличием синтаксической ошибки или ошибки формата.

Примечание: Данная ошибка возникает при последней проверке какой-либо команды перед тем, как Ассемблер переходит к следующей строке, и она обозначает, что после допустимого символа завершения операнда стоят лишние символы.

LABEL VALUE CHANGED BETWEEN PASSES. (Значение метки изменилось между проходами Ассемблера). Значение символа (идентификатора), декодированное в период выполнения прохода 1, не равняется обнаруженному при выполнении прохода 2.
Примечание: Эта ошибка обычно возникает в том случае, когда Ассемблер обрабатывает другие части программы при выполнении прохода 1 по сравнению с проходом 2 из-за изменения значений аргументов директив условной компиляции. Для обнаружения подобных типов ошибок может оказаться полезной директива PASS1 ON/OFF.

ATTEMPTED DIVISION BY ZERO. (Предпринята попытка деления на ноль). При выполнении деления делитель оказался равным 0.

ILLEGAL EXTERNAL REFERENCE. (Не верная внешняя ссылка). В данном контексте внешняя ссылка использоваться не может.

NESTED CONDITIONAL ASSEMBLY UNBALANCE DETECTED. (Обнаружен дисбаланс при анализе вложенных структур ассемблера). Обнаружена какая-либо директива класса '.IF', которой не соответствует парная директива '.ENDIF'.

ILLEGAL REGISTER. (Недопустимое использование регистра). Для данной команды недопустимо использование указанного регистра.

CANT RECOGNIZE NUMBER BASE. (Невозможно определение основания системы счисления числа). Заданное основание системы счисления числа не допускается ассемблером.

NOT ENOUGH PARAMETERS. (Не хватает параметров). В макросе число аргументов превышает число параметров.

ILLEGAL LABEL 1ST CHARACTER. (Неверен первый символ метки). Метка должна начинаться с алфавитного символа.

MAXIMUM EXTERNAL SYMBOL COUNT EXCEEDED. (Превышено максимальное число внешних символов). В модуле было задано слишком большое число внешних символов (идентификаторов).
Примечание: В модуле допустимо приблизительно 500 внешних символов (идентификаторов).

MUST BE IN SAME SECTION. (Должен находиться в той же самой секции). Операнд команды находится в другой секции.

NON-EXISTENT INCLUDE FILE. (Включаемый файл не существует). Данный включаемый файл не может быть найден.

ILLEGAL NESTED INCLUDE. (Не верное вложенное включение файлов). Один включаемый файл содержит директиву .INCLUDE. Данное сообщение об ошибке может также указывать на отсутствие оператора END в каком-либо включаемом файле.

NESTED SECTION UNBALANCE. (Дисбаланс вложенных секций). В описании вложенной секции отсутствует ENDS.

MISSING DELIMITERS ON MACRO CALL LINE. (В строке вызова макроса пропущены ограничители). Обнаружено несовпадение ограничителей при вызове макроса.

MULTIPLE EXTERNALS IN THE SAME OPERAND. (В одном и том же операнде обнаружено множественное число внешних символов (идентификаторов)). В одном и том же операнде существует более одного внешнего символа (идентификатора)).

A LABEL IS ILLEGAL ON THIS INSTRUCTION. (Метка не применима к данной команде). Данное сообщение об ошибке используется для того, чтобы обозначить метки, не получающие перемещаемого значения, такие как ENDM или MACEND. Использование этих меток недопустимо для данной команды.

MACRO STACK OVERFLOW. (Переполнение стека макросов). Использована слишком большая степень вложения макросов.
Примечание: Данная ошибка может быть вызвана слишком большим числом рекурсивных вызовов макросов. Стек допускает использование приблизительно 700 вложенных или рекурсивных вызовов макросов. На число макросов оказывает влияние число аргументов, которые используются в макросе.

MISSING LABEL. (Пропущена метка). В данной команде требуется присутствие метки.

OPERAND MUST BE DEFINED AS AN 8 BIT RELOCATABLE VALUE. (Операнд может быть описан как 8-и битовое перемещаемое значение). Это сообщение об ошибке возникает, когда 16-битовый адрес используется в команде, допускающей 8-битовый операнд. Для того чтобы сделать это значение перемещаемым, необходимо использовать знак <или>.

MISSING RIGHT ANGLE BRACKET. (Пропущена правая угловая скобка). Правая угловая скобка является обязательной.

MACRO NAME MUST APPEAR ON SAME LINE AS MACRO DEFINITION. (Имя макроса должно появляться не той же самой строке, что и описание макроса).

ILLEGAL LOCAL LABELS. (Недопустимо использование локальных меток). Метки не могут быть описаны как локальные. Например, .VAR.

MISSING MODULE DIRECTIVE. (Пропущена директива описания модуля MODULE).

MISSING ENDMOD DIRECTIVE. (Пропущена директива ENDMOD (задания конца модуля)).

'Module' CAN'T BE IN 'Include' FILE. (Директива 'Module' не может находиться во 'включаемом' файле).

'Endmod' CAN'T BE IN 'Include' FILE. (Директива 'Endmod' не может находиться во 'включаемом' файле).

2.7 МЕТОДИКА РАБОТЫ С КОМПИЛЯТОРОМ X8051

Компилятор X8051 может работать в диалоговом режиме, режиме командной строки и из среды текстового редактора Multi Edit.

2.7.1 ДИАЛОГОВЫЙ РЕЖИМ.

Для вызова Ассемблера необходимо загрузить x8051.exe. Ассемблер в ответ запросит:
Listing Destination ?(N,T,D,E,L,<CR>=N).
где аббревиатуры означают следующее:
N печати нет;
T терминал;
P принтер;
D диск;
E только ошибки;
L печать вкл/выкл.

Затем Ассемблер запросит имя файла, содержащего исходные коды:
Input filename:

При вводе имени файла можно опустить расширение, если оно asm. Далее Ассемблер запросит имя выходного файла:
Output filename:

Если пользователь ответит возвратом каретки, имя выходного файла будет образовано из имени входного с расширением obj. Если имя файла в ответе не будет содержать расширение, то оно предполагается равным obj.

Если выдача листинга идет под управлением директивы ассемблера LIST ON/OFF, то возникает дополнительный запрос:
LIST ON/OFF Listing Destination (T,P,D,<CR>=T):
Сокращения соответствуют предыдущим.

2.7.2 РЕЖИМ КОМАНДНОЙ СТРОКИ

Ассемблер может воспринимать командную строку. В этом случае имя входного файла определяется первым, дополнительно может идти имя выходного файла и список опций. Общая форма команды (необязательные поля показаны в квадратных скобках) будет следующей:

asm8051[-q]input_filename[output_filename][-t,-p,-d,-px,-dx]

Если введена опция -q, на экран выводятся только сообщения об ошибках и соответствующие строки. Эта опция должна предшествовать имени файла:

-t вывод на терминал
-p вывод на принтер
-x вывод листинга с таблицей перекрестных ссылок
-d вывод на диск
-e вывод только сообщений об ошибках
-l вывод блоков помеченных в тексте LIST ON/OFF

2.7.3 РЕЖИМ РЕДАКТИРОВАНИЯ И КОМПИЛИРОВАНИЯ ИЗ СРЕДЫ ТЕКСТОВОГО РЕДАКТОРА MULTI EDIT

Компилятор X8051 позволяет выполнять процесс компиляции непосредственно из среды текстового редактора Multi Edit (используя при этом все возможности самого редактора). Процесс создания исходного (ассемблерного) файла и его компиляция из среды Multi Edit состоит из следующих этапов:

загрузка среды текстового редактора Multy Edit;
создание (редактирование) исходного файла;
компиляция;
проверка наличия ошибок;
исправление ошибок и переход к этапу «компиляция» при наличии ошибок, выход из среды редактора при отсутствии ошибок.

Процесс создания исходного (ассемблерного) файла и компиляцию из среды текстового редактора Multi Edit рассмотрим на конкретном примере.

ПРИМЕР. Программа должна выполнять подсчет суммы значений элементов массива. Длина массива - 8 байт. Начальный адрес массива (адрес ячейки памяти в которой находится 1-й элемент массива) - 40H. Значение суммы элементов массива сохранить в регистрах R7 (старший байт) и R6 (младший байт) банка регистров общего назначения N0.

Исходный текст программы:

```
M_HOME      EQU  40H      ;Начало массива
M_LONG      EQU  8        ;Длина массива
ORG         0000H
LJMP        HOME          ;Переход на начало программы.
ORG         0030H         ;Директива ассемблеру расположить
                          программу начиная с адреса 0030H. (т.к. по
                          адресам 03).HOME:    MOV R0,#M_HOME
```

```
        ;R0 - указатель на i-тый элемент массива.
        Установка указателя в начало массива.
                CLR  A                  ;Обнуление аккумулятора.
                MOV R7,A                ;Обнуление регистра R7 (ст. байт суммы).
                ;----------------------------Цикл подсчета суммы элементов массива
M0:             CLR  C                  ;Обнуление флага переноса.
                ADDC        A,@R0       ;Сложение содержимого аккумулятора с
                                        ;содержимым ячейки памяти, на которую
                                        ;указывает регистр R0.
                JNC  M1                 ;Если флаг переноса равен 0, то переход на
                                        ;M1
                INC  R7                 ;Флаг переноса равен 1 - увеличить на 1 R7.
M1:             INC  R0                 ;Увеличение указателя на 1.
                CJNE R0,#M_HOME+M_LONG,M0;Проверка указателя: если
                                                ;указатель указывает не
                                                ;на последний элемент массива,
                                                ;то переход на M0.
                MOV R6,A                ;Сохранение мл. байта значения
                                        ;суммы элементов массива в
                                        ;регистре R6.
                END
```

Загрузить текстовый редактор Multi Edit. Для этого необходимо ввести команду:
Drive:\Path\me.exe
Примечание: Для загрузки текстового редактора с автоматической загрузкой файла с исходным текстом программы (данный файл должен существовать) необходимо ввести команду:
Drive:\Path\me.exe filename.asm

Отредактировать (создать) исходный ассемблерный файл, т. е. ввести текст исходной программы с клавиатуры. При этом используйте все возможности текстового редактора Multi Edit (копирование, перенос, и т.д.). Подробности по использованию Multi Edit смотрите в разделе «Помощь». Рекомендация: в процессе ввода текста программы не забывайте периодически записывать файл на диск. Пусть файл исходного текста программы будет иметь имя «first.asm».

Выполнить компиляцию программы. Это можно сделать несколькими способами, но рассмотрим самый короткий. Для этого необходимо нажать комбинацию клавиш <Ctrl> + <F8>. В открывшемся окне выбрать пункт «87C51» и нажать клавишу <Enter>. Процесс компиляции будет отображаться в статусной строке Multi Edit.

Проверить наличие (отсутствие) ошибок. Для этого необходимо переключиться в соседнее окно редактора Multi Edit - нажать комбинацию клавиш <Ctrl> + <F1>. В этом окне будет представлен файл отчета о процессе компиляции «meerr.tmp».

При наличии ошибок в файле отчета будут представлены строки исходного текста программы, в которых имеются ошибки, а также номера этих строк и тип ошибки. Для исправления ошибок необходимо переключиться в окно с исходным ассемблерным файлом - нажать комбинацию клавиш <Shift> + <F1>. Для быстрого перехода к строке, в которой содержится ошибка, можно воспользоваться командой «переход к строке N...». Для этого необходимо нажать комбинацию клавиш <Alt> + <F8>, в открывшемся окне ввести номер строки и нажать клавишу <Enter>. Исправить ошибки, после чего перейти к этапу «Выполнить компиляцию программы».
При отсутствии ошибок выйти из среды редактора Multi Edit, для этого необходимо нажать комбинацию клавиш <Alt> + <X>.

При успешной компиляции программы будут сгенерированы объектный файл с расширением «obj» и листинг программы с расширением «lst», в нашем примере это будут файлы «first.obj» и «first.lst».

3. РЕДАКТОР СВЯЗЕЙ ДЛЯ КОМПИЛЯТОРА МИКРОКОНТРОЛЛЕРА СЕМЕЙСТВА МК51

Везде далее под линкером предполагается программа редактора связей. Линкер позволяет пользователю записать программу языка ассемблера, содержащую несколько программных модулей. Линкер учитывает внешние ссылки и выполняет размещение в адресном пространстве. Он способен создавать выходные файлы для всех наиболее применяемых форматов.

Линкер может быть вызван в диалоговом, командном режимах или под управлением из файла. Выходной формат выбирается директивой в исходном файле или в параметрах команды LINK. Полное описание возможностей и методики работы с линкером излагается в руководстве объемом 38 страниц, поэтому для практической работы рассматривается только режим командной строки.

РЕЖИМ КОМАНДНОЙ СТРОКИ.

Линкер может быть вызван в командной строке. Формат такой команды показан ниже с последующей расшифровкой элементов:

Drive:\Path\Link.exe [-q]-c file1[-Innnn]file2[-Innnn]...[-ofile][-Llibfile][-options]

-q Линкер в режиме Quit. В этом случае выдается только сообщение об ошибках на консоль.

-c Требуется для указания, что будет использован режим командной строки, а не управляющего файла. Следом за ключом -c идет список входных файлов, состоящий (для приведенной строки) из файлов fil1 и fil2. За каждым файлом может следовать адрес смещения секций, начинающийся с -I. Если этот адрес отсутствует, то текущая секция является продолжением предыдущей с тем же именем.

-o Используется для указания выходного файла. Этот элемент не обязателен. Если он отсутствует, Линкер создаст выходной файл с тем же именем, что и первый входной, и с расширением, определяемым типом генерируемого формата.

-L Используется для задания библиотек. Максимум может быть указано 50 библиотек.

-options. Определяет дополнительные параметры. Знак минус требуется в начале списка, и в него может входить столько параметров, сколько необходимо (детально это описано в "Параметры Линкера").

ПАРАМЕТРЫ ЛИНКЕРА:

Параметры указываются в диалоговом режиме после имени библиотечных файлов. Эти параметры могут быть заданы и в режиме управления из файла и в командной моде. Когда указано несколько конкурирующих параметров, последний параметр отменяет действие предыдущего.

D Создать дисковый файл, содержащий все ошибки линкирования, символьную таблицу глобальных переменных и карту памяти (MAP) загрузки. Файл имеет то же имя, что и выходной файл, но с расширением map.

S Создать символьный файл для процесса отладки. Этот файл содержит все глобальные символы и их величины. Каждый символ имеет в длину 32 буквы.

A Создать символьный файл, но с символами в 10 букв. Он используется для совместимости с линкером 2500 A.D. версии 3.0.

M Создать символьный файл в целях отладки в формате Microtek.. Он содержит все символы, как глобальные, так и локальные. Для того чтобы этот формат мог быть создан, в исходном файле должна присутствовать директива SYMBOLS ON.

Z Создать символьный файл в целях отладки в формате ZAX. Этот файл содержит и локальные и глобальные переменные. Для того чтобы этот формат мог быть создан, в исходном файле должна присутствовать директива SYMBOLS ON.

X Создать выполняемый выходной файл.

H Создать шестнадцатеричный файл формата Intel.

E Создать шестнадцатеричный файл расширенного формата Intel.

T Создать шестнадцатеричный файл формата Tektronix.

1 Создать выходной файл формата S19 фирмы Motorola.

2 Создать выходной файл формата S28 фирмы Motorola.

3 Создать выходной файл формата S37 фирмы Motorola.

ПРИМЕР. Имеется объектный файл first.obj, необходимо выполнить процесс линкирования с созданием выполняемого (в двоичном коде) выходного файла.
Для этого необходимо выполнить следующую команду:
Drive:\Path\link.exe -c first.obj -x
При этом будет создан выходной файл с расширением «tsk» (в нашем примере first.tsk), код программы будет расположен с адреса 0000H (по умолчанию).

4. ПОЛНОЭКРАННЫЙ ОТЛАДЧИК АССЕМБЛЕРНЫХ ПРОГРАММ ДЛЯ МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА МК51

Полноэкранный отладчик-эмулятор для программ, написанных на языке ассемблера микроконтроллеров семейства МК51, предназначен для логической отладки программ. Каких-либо аппаратных средств отладчик не поддерживает.

Отладчик позволяет:

загрузить для отладки tsk-файлы, вырабатываемые имеющимися кросс-средствами (транслятором с языка ассемблера), т.е. файлы чистого двоичного кода;

просмотреть на экране дисассемблированный текст загруженной программы, включая адреса и коды команд, область имитируемого ОЗУ данных, область внешней памяти, памяти программ, содержимое всех регистров микроконтроллера;

выполнить загруженную программу по шагам с просмотром результатов после каждого шага и в непрерывном режиме с остановом по точкам прерывания по достижении задаваемых пользователем адресов;

внести изменения в загруженную программу в мнемонических обозначениях языка ассемблера, а также в машинных кодах;

внести изменения в содержимое регистров, флагов и памяти в командном режиме и в режиме полноэкранного редактирования;

вывести на печать или дисковые носители дисассемблированный текст, дампы памяти;

сохранить содержимое любой области памяти в файле на дисковом носителе;

загрузить память из дискового файла;

получить трассировку программы;

определить время выполнения загруженной программы и ее частей по встроенному счетчику.

4.1 ЗАПУСК ОТЛАДЧИКА

Для загрузки программы отладчика необходимо в командной строке ввести команду:

Drive:\Path\fd51.exe

При этом загружается среда отладчика FD51 и отладчик готов к работе (где Drive:\Path - имя диска и путь к файлу fd51.exe).

4.2 ВВОД КОМАНД

Сразу после запуска отладчик готов к приему команд пользователя - курсор находится в командной строке. В нижней строке экрана имеется меню функциональных клавиш F1-F10 - они выполняют наиболее употребительные команды. Остальные команды вводятся пользователем с клавиатуры с использованием алфавитно-цифровых клавиш. При вводе этих команд можно пользоваться для редактирования клавишами <Ins>, <Del>, <BackSpace>,

<Home>, <End>, <Esc>. После начала ввода команды и до нажатия клавиши <Enter> функциональные клавиши недоступны. Если команда неверна, выдается сообщение об ошибке и звуковой сигнал.

4.3 ОПИСАНИЕ КОМАНД

4.3.1 ФУНКЦИОНАЛЬНЫЕ КЛАВИШИ

F1 - выполнить текущую инструкцию загруженной программы. Текущая инструкция - это инструкция, выделенная в окне дисассемблированного текста светлым прямоугольником. После выполнения на экране можно сразу наблюдать результаты ее выполнения.

F2 - выполнять программу до следующей по адресу за текущей инструкцией. Эта клавиша позволяет выполнить подпрограмму или цикл как одну инструкцию, что удобно, так как не нужно просматривать уже отлаженные подпрограммы.

F3 - позволяет представить числовую информацию на экране (содержимое регистров и памяти) в десятичной, а при повторном нажатии - в двоичной форме. После запуска информация представлена в шестнадцатеричном виде.

F4 - переключает большое окно памяти с внутренней (INT RAM) на внешнюю (EXT RAM) и обратно.

F5 - установка точек прерывания (см. п. 5).

F6 - переключает форму представления памяти в окне в двоичную и обратно.

F7 - листает окно памяти данных вверх на одну строку.

F8 - листает окно памяти данных вниз на одну строку.

F9 - листает окно памяти программ вверх на одну строку.

F10- листает окно памяти программ вниз на одну строку.

Для быстрого листания можно пользоваться следующими клавишами:

<Home> - листает окно памяти данных вверх на одну страницу.
<End]> - листает окно памяти данных вниз на одну страницу.
<PgUp> - листает окно памяти программ вверх на одну страницу.
<PgDn> - листает окно памяти программ вниз на одну страницу.

4.3.2 КОМАНДЫ ОТЛАДЧИКА

Для быстрого получения справки по командам можно ввести команду "H" или нажать комбинацию клавиш <Ctrl>+<H>".

В настоящем описании используются следующие обозначения:

параметры заключены в угловые скобки, например <адрес>.

необязательные параметры заключены в квадратные скобки, например [<адрес>].

Все числовые значения должны иметь шестнадцатеричный формат, при этом не требуется указывать символ "h".

L [<тип памяти><нач. адрес>,] <файл. спец.>[/A]. Загрузить файл в память. <Тип памяти> может быть I, E или P. В соответствии с этим параметром файл загружается во внутреннюю (Int), внешнюю (Ext) или программную (Pgm) память. <Нач. адрес> и <тип памяти> указывается только при загрузке чистого двоичного кода. При загрузке файла, выработанного ISIS-II MACRO-ASSEMBLER'ом, нужно указать только спецификацию файла и ключ /A.
Пример: L I 01F,A:\PGM\T1 - загрузить двоичный файл во внутреннюю память с адреса 01F.

S <тип памяти><нач. адрес>-<кон. адрес>,<файл. спец.>. Сохранить область памяти в дисковом файле (вообще говоря, во всех командах в качестве <файл. спец.> допускается любая корректная в DOS спецификация файла, например COM1). <нач.адрес> и <кон.адрес> указывают соответственно начало и конец сохраняемой области. Сохраненный командой S файл можно потом снова загрузить командой L.
Пример: S P 20-642,C:\PGMLIB\MYFILE

PRT <тип памяти><нач. адрес>-<кон. адрес>[,<файл. спец.>]. Распечатать дамп области памяти в шестнадцатеричном формате. Если не указан <файл. спец.>, то дамп выводится на принтер.

PRTD <нач. адрес>,<количество команд>[,<файл. спец.>]. Распечатать дисассемблированный текст, начиная с <нач. адреса>. Вывод по умолчанию на принтер.

R <номер регистра>=<число>. Занести число в регистр текущего банка. Число должно быть байтом.
Пример: R4=FF

<Имя регистра>=<число>. Занести число в регистр специального назначения. Можно использовать следующие имена: A, B, TH0, TH1, TL0, TL1, DPH, DPL, DPTR, SP, IP, IE, TMOD, TCON, SCON, SBUF, PC. Число для PC и DPTR может быть и двухбайтовой величиной.
Пример: SP=20 DPTR=FF00

<Имя флага>=<число>. Установить или сбросить флаг в PSW. Имена флагов: C, AC, F0, S1, S0, OV, P. Если число=0, то флаг сбрасывается, иначе - устанавливается.
Пример: S1=0

PO <номер порта>=<число>. Занести число в порт. Номер порта может быть 0-3.
Пример: PO2=12

D <адрес>. Установить адрес дисассемблированного текста в окне.
Пример: D 0240

<Тип памяти><адрес>[-<кон. адрес>]=<число>. Занести число в память. Если указан <кон. адрес>, то этим числом заполняется область памяти.
Пример: I 22=55 P 0-40=FF
Возможно возникновение неоднозначности при заполнении некоторых ячеек памяти. Например, команда "P C=23" будет воспринята не как занесение числа 23 в память программ по адресу 0C, а как команда установки счетчика команд (PC) в значение 23. В этом случае нужно явно указать, что это адрес: P 0C=23.

<Тип памяти><адрес>.<номер бита>=<число>. Установить или сбросить бит в памяти. <Номер бита> может быть 7-0 (старший бит - 7).
Пример: I 20.6=1

<Имя регистра>.<номер бита>=<число>. Установить или сбросить бит в регистре специального назначения (A, B, P00-P03, IP, IE, TMOD, TCON, SCON).
Пример: TMOD.3=0

M <тип памяти><нач. адрес>. Установить начальный адрес памяти в окне.
Пример: M I 20 M E 0FF M P 0

G [<нач. адрес>[,<кон. адрес>]]. Выполнить программу с <нач. адреса> до <кон. адреса>. Если <нач. адрес> не указан, выполнение начинается с текущей команды (текущая команда выделена белым прямоугольником). <Кон. адрес> можно не указывать, если используются точки прерывания. Выполняющуюся программу можно остановить нажатием любой клавиши. G без параметров можно ввести нажатием <Alt-F10>. Можно указать только конечный адрес, но запятая должна присутствовать.
Пример: G 100-FF0 G,2200

T ON [,<файл. спец.>]. Включить трассировку программы. По умолчанию трассировочные записи выводятся на принтер.

T OFF. Выключить трассировку.

INT <0/1>=<число>. Имитировать высокий или низкий уровень на входах INT0 или INT1.
Пример: INT1=0

BA=<адрес>. Установить новую "точку отсчета" для дисассемблирования. Эта команда полезна при просмотре таблиц, зашитых в памяти программ, когда при дисассемблировании "назад" неизвестно откуда вести дисассемблирование.

RSTC. Сбросить счетчик времени выполнения программы.

QUIT. Выход в DOS.

RST. Имитируется сброс процессора.

Віктор Васильович Ткачов
Микола Володимирович Козар
Владислав Іванович Шевченко
Станіслав Миколайович Проценко
Олег Вікторович Карпенко
Володимир Валентинович Надточий
Марина Олексіївна Ткачук

РОБОЧИЙ ЗОШИТ

з дисциплін
“Основи побудови мікропроцесорних систем керування”, “Мікропроцесорна техніка”, “Програмні засоби систем керування”
для студентів спеціальностей
АГ – 8.092501 Автоматизоване управління технологічними процесами
АТ,МЕ – 8.091401 Системи управління й автоматики
СМ – 8.091501 Комп`ютерні системи та мережі

Видано за редакцією авторів.

Підп. до друку 16.01.2013. Формат 30х42/4.
Папір офсет. Ризографія. Ум. друк. арк. 7,1.
Обл. вид. арк. 8,3. Тираж 100 пр. Зам. №

ДВНЗ «Національний гірничий університет»
49027, м. Дніпропетровськ, просп. К. Маркса, 19.